# Procedural Elements for Computer Graphics

David F. Rogers

Professor of Aerospace Engineering
and
Director, Computer Aided Design
and Interactive Graphics
United States Naval Academy, Annapolis, Md.

McGraw-Hill Book Company

New York St. Louis San Francisco
Auckland Bogotá Hamburg London
Madrid Mexico Montreal New Delhi
Panama Paris São Paulo Singapore
Sydney Tokyo Toronto

Д. Роджерс

# Алгоритмические основы машинной графики

Перевод с английского
С. А. Вичеса, Г. В. Олохтоновой
и П. А. Монахова
под редакцией
Ю. М. Баяковского и В. А. Галактионова

Тверская областная библиотека
им. А.М. Горького

Москва «Мир» 1989

ББК 32.97 + 22.1
Р60
УДК 681.3.082.5

**Роджерс Д.**

Р60 Алгоритмические основы машинной графики: Пер. с англ. — М.: Мир, 1989. — 512 с., ил.

ISBN 5-03-000476-9

Книга известного американского специалиста, знакомого советским читателям по переводу его «Математических основ машинной графики» (М.: Машиностроение, 1980). Новая книга содержит анализ алгоритмов и методов современных графических систем, особое внимание уделено методам растровой графики. Алгоритмы доведены до программ на псевдокоде, легко преобразуемом в языки Паскаль, Фортран и Бейсик. Книга изобилует иллюстрациями и примерами, содержит задания для самостоятельного решения.

Для математиков-вычислителей, специалистов, аспирантов и студентов, интересующихся машинной графикой и автоматизацией проектирования.

$\text{Р}\frac{1602110000-181}{041\,(01)-89}-24-89$ **ББК 681.3.082.5**

*Редакция литературы по математическим наукам*

ISBN 5-03-000476-9 (русск.)
ISBN 0-07-053534-5 (англ.)

© 1985 by McGraw-Hill, Inc.
© перевод на русский язык, с изменениями, «Мир», 1989

# Предисловие редакторов перевода

Специалистам по машинной графике имя автора книги профессора Дэвида Роджерса хорошо известно по написанной им совместно с Дж. Адамсом монографии «Математические основы машинной графики» (М.: Машиностроение, 1980).

В новой книге, представляющей собой естественное продолжение упомянутой монографии, рассматриваются алгоритмы и методы, лежащие в основе современных растровых графических систем. От других книг по машинной графике, известных читателю, ее отличает глубина изложения и более полное представление материала. Автор выделяет некоторые наиболее важные разделы современной растровой графики — такие, как растровая развертка отрезков и многоугольников, отсечение, удаление невидимых поверхностей и др., — и подробно их анализирует. При этом рассматривается и сравнивается, как правило, несколько различных методов и подходов. Геометрические алгоритмы сопровождаются подробными блок-схемами и программами на псевдокоде, а также многочисленными примерами.

В основе книги лежит курс лекций по машинной графике, который автор читает в университете Дж. Гопкинса. Поэтому ее отличает методически продуманный отбор материала, а также простота и доступность изложения.

Настоящую монографию и известную книгу У. Фоли и А. вэн Дэма [1-3] разделяет по времени выхода в свет лишь три года. Однако машинная графика сделала за это время большой скачок в своем развитии. Появился ряд новых, более эффективных алгоритмов в традиционных разделах растровой графики (например, алгоритмы Лианга — Барского отсечения отрезков и многоугольников)

разработаны методы, открывающие принципиально новые возможности для синтеза реалистических изображений. Это трассировка лучей с использованием глобальной модели освещения, позволяющей учитывать как отражение, так и преломление света от многих источников, метод систем частиц, алгоритмы устранения ступенчатости и размытости вследствие движения, методы генерации фрактальных кривых и поверхностей, создание различных оптических эффектов и др. Все эти актуальные вопросы, не нашедшие пока отражения в монографической литературе, и рассматриваются в книге.

Своевременность перевода книги на русский язык подкрепляется еще одним существенным обстоятельством. Машинная графика стала необходимым инструментом в работе ученых и инженеров. При наличии источников данных большого объема (таких, как суперЭВМ, искусственные спутники Земли, предназначенные для исследования природных ресурсов, медицинские томографы), проблема визуализации приобрела исключительную важность. Отметим, что 50% нейронов мозга человека так или иначе связано с обработкой визуальной информации. Следовательно, машинная графика должна обеспечить эффективное и рациональное использование этого главного канала связи между человеком и ЭВМ.

Проблемы визуализации становятся камнем преткновения на пути развития научных исследований и увеличения мощности вычислительных мащин. В связи с этим Национальным научным фондом США разрабатывается проект «Визуализация в научных вычислениях». Аналогичная программа формируется в НАСА.

Характерно, что и в нашей стране интерес к машинной графике проявляют представители различных специальностей: конструкторы, технологи, геофизики, биологи, медики, дизайнеры, художники-мультипликаторы. Существуют области, в которых без средств машинной графики применение ЭВМ вообще теряет смысл. Неудивительно поэтому, что постоянно ощущается острый дефицит литературы, раскрывающей возможности современной машинной графики. К такого рода изданиям и относится настоящая книга.

Книгу перевели С. А. Вичес (гл. 3,4), П. А. Монахов (предисловие, гл. 1, 2), Г. В. Олохтонова (гл. 5). При переводе книги были исправлены некоторые ошибки и опечатки. О части из них любезно сообщил нам автор, с которым мы поддерживали постоянный контакт при работе над переводом. Мы благодарны проф. Роджерсу за помощь и сотрудничество.

*Ю. М. Баяковский*
*В. А. Галактионов*

# Предисловие к русскому изданию

Теперь, после выхода русского издания, книга «Алгоритмические основы машинной графики» доступна читателям на 6 языках: английском, японском, французском, итальянском, китайском и русском. Я искренне рад, что русские коллеги пополнили этот список.

За время, прошедшее с момента выхода первого издания на английском языке, особенно большой прогресс был достигнут в области совершенствования аппаратуры. При продолжающемся резком снижении стоимости превосходных по своим характеристикам инженерных и научных автоматизированных рабочих мест, снабженных полным набором графических средств, быстро возрастают их вычислительные и графические возможности. Алгоритмы, на выполнение которых 3—4 года назад затрачивались десятки часов, теперь укладываются в десятки минут. Сегодня можно за вполне разумную цену купить инженерное рабочее место в настольном варианте, в котором аппаратно реализуются алгоритм удаления невидимых граней с помощью $z$-буфера и модели закраски Гуро и Фонга. Обычными в широкой практике стали алгоритмы синтеза изображений с множественными подвижными источниками света. Если 10 лет назад манипулирование каркасной геометрической моделью в реальном масштабе времени считалось из ряда вон выходящим событием, то сегодня стало уже привычным манипулирование моделью сплошного тела с закраской Гуро в реальном масштабе времени. Оборудование, приобретенное по вполне доступным ценам, позволяет в реальном масштабе времени манипулировать полноцветными изображениями, формируемыми методом трассировки лучей, правда, пока при умеренном разрешении.

Существенно улучшилось наше понимание фундаментальных

физических и математических принципов в моделях освещения, и более совершенные модели быстро осваиваются в алгоритмах синтеза изображений. Значительное продвижение связано с воспроизведением диффузного света, когда учитывается излучательность, исходя из соображений переноса энергии. Повышение эффективности в алгоритмах трассировки лучей носит поистине драматический характер. Алгоритм, в котором сочетается излучательность для диффузного отражения и трассировка лучей для зеркального отражения, устанавливает сегодняшнюю норму реалистичности изображения.

Машинная графика продолжает определяющим образом влиять на состояние дел в разработке ЭВМ и вообще в информатике. Это воистину захватывающее дух поле деятельности.

июль 1988 г.
Аннаполис, Мериленд

*Дэвид Ф. Роджерс*

*Посвящаю моим родителям Глэдис Мэрион (Цоллер) Роджерс (р. в 1906 г.) и Льюису Фриману Роджерсу (1906—1981 г.), так облегчивших мои первые шаги самостоятельной жизни.*

# Предисловие

Машинная графика в настоящее время уже вполне сформировалась как наука. Существует аппаратное и программное обеспечение для получения разнообразных изображений — от простых чертежей до реалистичных образов естественных объектов. Десятилетие назад подобные средства стоили сотни тысяч долларов, а сегодня их цена уменьшилась в десятки раз. Во многих случаях вполне приемлемые результаты дает оборудование стоимостью всего в несколько тысяч долларов. Машинная графика используется почти во всех научных и инженерных дисциплинах для наглядности восприятия и передачи информации. Знание ее основ в наше время необходимо любому ученому или инженеру. Машинная графика властно вторгается в бизнес, медицину, рекламу, индустрию развлечений. Применение во время деловых совещаний демонстрационных слайдов, подготовленных методами машинной графики и другими средствами автоматизации конторского труда, считается нормой. В медицине становится обычным получение трехмерных изображений внутренних органов по данным компьютерных томографов. В наши дни телевидение и другие рекламные предприятия часто прибегают к услугам машинной графики и компьютерной мультипликации. Использование машинной графики в индустрии развлечений охватывает такие несхожие области как видеоигры и полнометражные художественные фильмы. Как свидетельствуют некоторые цветные фотографии, помещенные в книге, даже искусство не может устоять против этого вторжения.

Около десяти лет назад вышла в свет сходная по тематике кни-

га «Математические основы машинной графики»[1]. С тех пор в растровой графике удалось достичь существенного прогресса и в данной книге внимание сосредоточено именно на этих новых аспектах машинной графики. Книга начинается с введения в аппаратное обеспечение машинной графики, причем речь идет в основном о принципах работы дисплеев на электронно-лучевых трубках (ЭЛТ) и интерактивных устройств. В последующих главах рассматривается программное обеспечение растровой графики: вычерчивание отрезков и окружностей; заливка многоугольников и алгоритмы устранения лестничного эффекта; дву- и трехмерное отсечение, в том числе отсечение по произвольному выпуклому объему; алгоритмы удаления невидимых линий и поверхностей, в том числе трассировка лучей и, наконец, визуализация, понимаемая как «искусство» построения реалистичных изображений, в том числе локальная и глобальная модели освещения, фактура, тени, прозрачность и цветовые эффекты. Методически книга построена так же, как и ее предшественница: после обсуждения каждой темы следует подробный алгоритм или пример, а иногда и то и другое.

Книга в целом может быть использована для полугодового вводного курса машинной графики (прежде всего растровой) для студентов старших курсов или аспирантов. Если вводный курс уже существует и он основан на материале книги «Математические основы машинной графики», то данная книга предоставляет идеальный материал для более углубленного курса. Именно таким образом ее использовал автор. Если в односеместровом курсе желательно охватить более широкий круг вопросов, то можно воспользоваться обеими книгами. В этом случае предлагаются следующие темы: гл. 1 обеих книг, за ней гл. 2 и 3 с отдельными темами из гл. 4 «Математических основ...», затем некоторые разделы из гл. 2 (с разд. 2.1 по разд. 2.5, 2.7, с разд. 2.15 по разд. 2.19, 2.22, 2.23, 2.28), гл. 3 (разд. 3.1, 3.2, с разд. 3.4 по разд. 3.6, 3.9, 3.11, 3.15, 3.16), гл. 4 (разд. 4.1, часть разд. 4.2 с алгоритмом отбрасывания заведомо невидимых граней, разд. 4.3, 4.4, 4.7, 4.9, 4.11, 4.13) и гл. 5 (с разд. 5.1 по разд. 5.3, 5.5, 5.6, 5.14). Мы надеемся, что книга будет также полезна профессиональным программистам, инженерам и ученым. Благодаря тому, что алгоритмы и примеры изложены весьма подробно, книгу могут самостоятельно изучать читатели разной квалификации. Для понимания излагаемого материала

---

[1] Имеется русский перевод: Роджерс Д. Ф., Адамс Дж. Математические основы машинной графики. — М.: Машиностроение, 1980. — *Прим перев.*

достаточно знания математики на уровне колледжа и знакомства с языком программирования высокого уровня. Знания в области структур данных желательны, но не обязательны.

В книге используется два способа представления алгоритмов, первый — детальное описание алгоритма в словесной форме. Второй способ более формален и основан на алгоритмическом «языке». Из-за большой популярности машинной графики выбор языка вызвал большие затруднения. Автор опросил по этому поводу ряд специалистов, но к единому мнению прийти не удалось. Преподаватели информатики предпочитают Паскаль, но с сильным уклоном к языку Си. Специалисты, работающие в промышленности, отдают предпочтение Фортрану из соображений совместимости с существующим программным обеспечением. Сам автор предпочитает Бейсик ввиду простоты его использования. В результате детальные описания алгоритмов представлены на псевдокоде, который основан на большом опыте обучения машинной графике лиц, не знакомых с общепринятыми языками программирования. Псевдокод легко переводится на любой из этих языков. Описание псевдокода приведено в приложении. Все представленные в книге алгоритмы, написанные на псевдоколе, либо были непосредственно реализованы на ЭВМ на основе псевдокода, либо были получены из работающей программы, написанной на одном или нескольких общепринятых языков программирования. Диапазон языков реализации простирается от Бейсика на Apple IIe до PL/1 на IBM 4300. Демонстрационные программы можно получить у автора.

Возможно, читателю будет интересно узнать, как создавалась книга[1]. Она была набрана в корпорации TYX (Рестон, Вирджиния) на ЭВМ в системе обработки текстов $T_EX$. Первоначальный вариант был подготовлен на основе рукописного текста. Процессы редактирования и верстки проводились на гранках и двух сверстанных корректурах, полученных с лазерного принтера. Окончательный вариант, пригодный для вставки тоновых иллюстраций, был отпечатан на фотонаборном автомате. Я хотел бы выразить особую благодарность за терпение и помощь сотрудникам компании TYX — Джиму Готье и Марку Хоффману, которые помогли мне изучить систему и разрешить мои бесчисленные затруднения. Я благодарю Луизу Борер и Бет Лесселс за огромную работу, выполненную при вводе рукописного материала. Редакторы Дэвид

---

[1] Речь идет об английском оригинале. Русское издание подготовлено на фотонаборном комплексе «Компьюграфик» методом оригинала-макета. — *Прим. ред.*

Дэмстра и Сильва Уэррен из издательства McGraw-Hill продемонстрировали, как и всегда, высокий профессионализм.

Эта книга никогда не появилась бы на свет без помощи многих людей. Основу книги составляет материал курса для аспирантов, читающийся с 1978 г. в лабораторном Центре прикладной физики университета Джонса Гопкинса. Я признателен многим студентам, слушавшим этот и другие курсы, от которых я узнал так много. Я благодарю Тернера Уиттеда, прочитавшего первоначальные наброски и сделавшего ценные замечания, моих коллег Пита Атертона, Брайена Барски, Эда Кэтмулла, Роба Кука, Джона Дилла, Стива Хансена, Боба Льюенда, Гэри Мейера, Элви Рэя Смита, Дэйва Уорна и Кевина Уэйлера. Все они прочитали с красным карандашом в руках один или несколько разделов книги в рукописном варианте и благодаря их многочисленным советам и замечаниям книга стала лучше. Я признателен моим коллегам Линде Рийбак и Линде Эдлам, прочитавшим всю рукопись и проверившим примеры. Следует отметить моих студентов: Била Миера, реализовавшего алгоритм Робертса, Гэри Бохена, предложившего тест на выпуклость из разд. 3.7 и Нормана Шмидта, предложившего метод разбиения многоугольника из разд. 3.8. Я признателен Марку Мейерсону, реализовавшему алгоритмы разбиения и математически обосновавшему метод, лежащий в их основе. Особенно ценна работа Ли Билоу и Джона Меткалфа, которые подготовили все штриховые рисунки.

Не нахожу слов благодарности в адрес Стива Сэтерфилда, прочитавшего все 800 рукописных страниц и сделавшему массу замечаний.

Следует поблагодарить и моего старшего сына Стефана, который реализовал все алгоритмы удаления невидимых поверхностей из гл. 4, а также ряд других алгоритмов. Во время наших бурных дискуссий прояснился ряд ключевых моментов.

И наконец, особенно хочу отметить мою жену Нэнси и двух других моих детей, Карен и Рэнсома, которые в течение полутора лет терпели отсутствие своего мужа и отца, скрывавшегося почти все ночи и выходные в кабинете. Вот это поддержка! Спасибо.

*Дэвид Ф. Роджерс*

# 1 Введение в машинную графику

Современная машинная графика — это тщательно разработанная дисциплина. Обстоятельно исследованы [1-1 — 1-3] основные элементы геометрических преобразований и описаний кривых и поверхностей. Также изучены, но все еще продолжают развиваться методы растрового сканирования, отсечение, удаление невидимых линий и поверхностей, цвет, закраска, текстура и эффекты прозрачности. Сейчас наибольший интерес представляют именно эти разделы машинной графики.

## 1.1. ОБЗОР МАШИННОЙ ГРАФИКИ

Машинная графика — сложная и разнообразная дисциплина. Для изучения ее прежде всего необходимо разбить на обозримые части, приняв во внимание, что конечным продуктом машинной графики является изображение. Это изображение может, разумеется, использоваться для разнообразных целей. Например, оно может быть техническим чертежом, иллюстрацией с изображением деталей в разобранном виде в руководстве по эксплуатации, деловой диаграммой, архитектурным видом предлагаемой конструкции или проектируемого здания, рекламной иллюстрацией или кадром из мультфильма. В машинной графике фундаментальным связующим звеном является изображение; следовательно, мы должны рассмотреть, как

изображения представляются в машинной графике;
изображения готовятся для визуализации;
предварительно подготовленные изображения рисуются;
осуществляется взаимодействие с изображением.

Хотя во многих алгоритмах в качестве геометрических данных, описывающих изображения, выступают многоугольники и ребра, каждый многоугольник или ребро в свою очередь может быть представлен своими вершинами. Таким образом, точки являются фундаментальными строительными блоками для представления геометрических данных. Не меньшего внимания заслуживает алгоритм, показывающий, как организовать точки. В качестве иллюстрации рассмотрим единичный квадрат в первом квадранте. Он может быть представлен своими вершинами (рис. 1.1):

$$P_1(0, 0),\ P_2(1, 0),\ P_3(1, 1),\ P_4(0, 1)$$

Соответствующее алгоритмическое описание может выглядеть так:

Соединить последовательно $P_1P_2P_3P_4P_1$

Единичный квадрат также можно описать с помощью четырех ребер:

$$E_1 \equiv P_1P_2,\ E_2 \equiv P_2P_3,\ E_3 \equiv P_3P_4,\ E_4 \equiv P_4P_1$$

Здесь алгоритмическое описание таково:

Изобразить последовательно ребра $E_1E_2E_3E_4$

И наконец, для описания единичного квадрата как многоугольника можно использовать либо точки, либо ребра. Например,

$$S_1 = P_1P_2P_3P_4P_1 \text{ или } P_1P_4P_3P_2P_1$$
$$\text{или } S_1 = E_1E_2E_3E_4$$

Основные строительные блоки (точки) в зависимости от размерности пространства можно представлять либо как пары, либо как тройки чисел. Таким образом, $(x_1, y_1)$ или $(x_1, y_1, z_1)$ представили

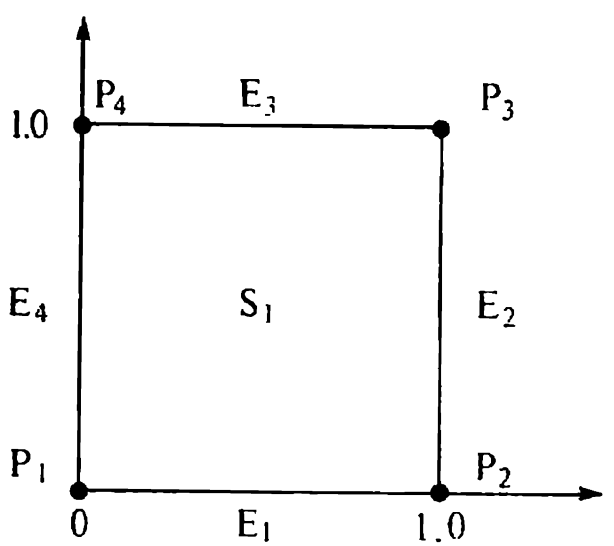



Рис. 1.1. Описания данных изображения.

бы точку в двух- или трехмерном пространстве. Две точки представили бы отрезок, или ребро, а совокупность из трех или более точек — многоугольник. Эти точки, ребра, многоугольники накапливаются, или хранятся, в базе данных. Данные, из которых получают рисунок, редко совпадают с данными, служащими непосредственно для рисования. Данные, используемые для вывода изображения, часто называют дисплейным файлом. В нем содержится некоторая часть, вид или сцена изображения, представленного в общей базе данных. Выводимое изображение обычно формируется с помощью операций поворота, переноса, масштабирования и вычисления различных проекций данных. Как правило, эти основные видовые преобразования выполняются с помощью матричных (4× 4) операций над данными, представленными в однородных координатах [1-1]. Эти операции часто реализуются аппаратно. Прежде чем рисовать окончательный результат, можно добавить удаление невидимых линий или поверхностей, произвести закраску, учесть влияние прозрачности, нанести текстуру и воспроизвести цветовые эффекты. Если не надо рисовать изображение, представленное во всей базе данных, то следует выбрать соответствующую его часть. Данный процесс называется отсечением. Отсечение может быть двух- или трехмерным. В некоторых случаях отсекающее окно или объем могут быть с дырами или иметь нерегулярную форму. Отсечение относительно стандартных областей часто реализуется аппаратно.

Почти все изображения содержат текстовую информацию. Литеры могут генерироваться либо аппаратным, либо программным образом, в последнем случае с ними можно обращаться как с любой другой частью изображения. При аппаратной генерации литеры сохраняются в виде кодов до момента непосредственного вывода. Для преобразования литер обычно предоставляются только ограниченные возможности, например некий набор углов поворота и коэффициентов масштабирования. Как правило, отсечение аппаратно генерируемых литер невозможно: либо изображается вся литера, либо она не изображается вообще.

## 1.2. ТИПЫ ГРАФИЧЕСКИХ УСТРОЙСТВ

Существует много разнообразных устройств для вывода изображений, построенных с помощью машинной графики. В качестве типичных примеров назовем перьевые графопостроители, точечно-матричные, электростатические и лазерные печатающие устройства, фильмирующие устройства, дисплеи на запоминающей трубке,

векторные дисплеи с регенерацией изображения и растровые дисплеи на электронно-лучевой трубке (ЭЛТ). Мы ограничимся обсуждением дисплеев на ЭЛТ, поскольку в большинстве систем машинной графики используются дисплеи подобного типа и именно в этом типе дисплеев воплощены наиболее фундаментальные концепции вывода изображения. Другие методы обсуждаются в [1-1 — 1-3].

Запоминающие ЭЛТ с прямым копированием изображения (рисование отрезками), векторные дисплеи с регенерацией изображения (рисование отрезками) и растровые сканирующие дисплеи с регенерацией (поточечное рисование) являются тремя основными типами дисплеев на базе ЭЛТ. Развитие электронной техники позволило использовать в одном дисплее несколько методов изображения. Мы стоим на пользовательской или концептуальной точке зрения при обсуждении различных дисплеев, т. е. в основном рассматриваем функциональные возможности дисплеев, а не особенности их электронной схемы.

## 1.3. ГРАФИЧЕСКИЕ ДИСПЛЕИ НА ЗАПОМИНАЮЩЕЙ ТРУБКЕ

Из всех дисплеев на ЭЛТ наиболее просто устроен дисплей на запоминающей ЭЛТ с прямым копированием изображения (ЗЭЛТ). Запоминающую ЭЛТ, называемую также бистабильной запоминающей трубкой, можно рассматривать как ЭЛТ, покрытую люминофором с длительным временем послесвечения. Линия или литера остаются на ней видимыми в течение длительного времени (до одного часа), прежде чем станут окончательно неразличимыми. На рис. 1.2 показан типичный дисплей такого типа. Чтобы нарисовать отрезок на дисплее, интенсивность электронного луча увеличивают до такой величины, которая вызывает запоминание следа луча на люминофоре. Для стирания изображения на всю трубку подают специальное напряжение, снимающее свечение люминофора. Экран вспыхивает и принимает исходное (темное) состояние. Стирание занимает около ½ с. Поскольку вспыхивает вся трубка, то стираются все отрезки и литеры. Таким образом, стереть отдельные линии и литеры нельзя, и изображение динамического движения или анимация невозможны. Иногда для обеспечения возможности ограниченной регенерации используется промежуточное состояние (режим рисования поверх изображения); см. ниже. В этом случае интенсивность электронного луча принимает значение, меньшее порогового,

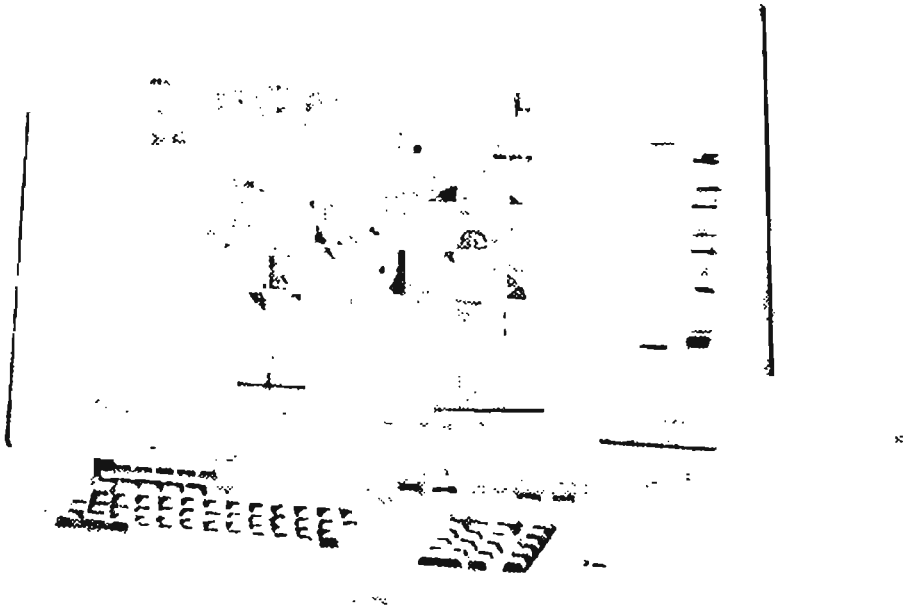


Рис. 1.2. Графический дисплей на запоминающей трубке.

которое вызывает запоминание, но достаточное для свечения люминофора. Поскольку в этом режиме изображение не сохраняется, для его видимости необходима постоянная перерисовка.

Дисплей на запоминающей трубке способен изображать фактически неограниченное количество векторов, а мерцание изображения вообще невозможно. Его разрешение обычно составляет 1024 × 1024 адресуемых точек (10 бит) на экране 8 × 8 дюймов (ЭЛТ с диагональю 11 дюймов), или 4096 × 4096 (12 бит) либо на экране 14 × 14 дюймов (ЭЛТ с диагональю 19 дюймов), либо на экране 18 × 18 дюймов (ЭЛТ с диагональю 25 дюймов). По вертикали обычно видно только 78% адресуемой области.

Дисплей на запоминающей трубке — это векторный дисплей, или дисплей с произвольным сканированием. Это означает, что отрезок (вектор) может быть нарисован непосредственно из одной адресуемой точки в любую другую. Относительно легко, быстро и недорого можно получить твердую копию экрана. Дисплеи на запоминающей трубке в некоторой степени легче программировать, нежели векторные или растровые дисплеи с регенерацией изображения. Дисплеи на ЗЭЛТ можно объединять с микрокомпьютерами в сателлитные графические системы или графические терминалы. В последнем случае алфавитно-цифровая или графическая информа-


Тверская областная библиотека

ция передается от главного компьютера в терминал посредством интерфейса. Обычно используется последовательный интерфейс, т. е. в единицу времени передается только 1 бит информации, хотя может использоваться и параллельный интерфейс. Уровень интерактивности дисплея с ЗЭЛТ ниже, чем у векторного дисплея с регенерацией или растрового дисплея вследствие небольшой скорости интерфейса и плохих характеристик стирания.

## 1.4. ВЕКТОРНЫЕ ГРАФИЧЕСКИЕ ДИСПЛЕИ С РЕГЕНЕРАЦИЕЙ ИЗОБРАЖЕНИЯ

В противоположность дисплею на запоминающей трубке в векторном (рисующем отрезки или векторы) дисплее с регенерацией изображения на базе ЭЛТ используется люминофор с очень коротким временем послесвечения. Такие дисплеи часто называют дисплеями с произвольным сканированием (см. ниже). Из-за того что время послесвечения люминофора мало, изображение на ЭЛТ за секунду должно многократно перерисовываться или регенерироваться. Минимальная скорость регенерации должна составлять по крайней мере 30 (1/с), а предпочтительнее от 40 до 50 (1/с). Скорость регенерации, меньшая 30 (1/с), приведет к тому, что изображение будет мерцать, как это бывает, когда кинофильм прокручивается слишком медленно. На такое изображение неприятно смотреть и его трудно использовать.

Для векторного дисплея с регенерацией требуется кроме ЭЛТ еще два элемента: дисплейный буфер и дисплейный контроллер. Дисплейный буфер — это непрерывный участок памяти, содержащий всю информацию, необходимую для вывода изображения на ЭЛТ. Функция дисплейного контроллера заключается в том, чтобы циклически обрабатывать эту информацию со скоростью регенерации. Сложность (число изображаемых векторов) рисунка ограничивается двумя факторами — размером дисплейного буфера и скоростью дисплейного контроллера. Еще одним ограничением является скорость обработки геометрической информации, например скорость выполнения таких операций, как преобразование и отсечение, генерация текстовой информации.

На рис. 1.3 представлены блок-схемы двух высокопроизводительных векторных дисплеев. В обоих случаях предполагается, что такие геометрические преобразования, как поворот, перенос, масштабирование, перспективное проецирование и отсечение, реализованы аппаратно в геометрическом процессоре. В первом случае

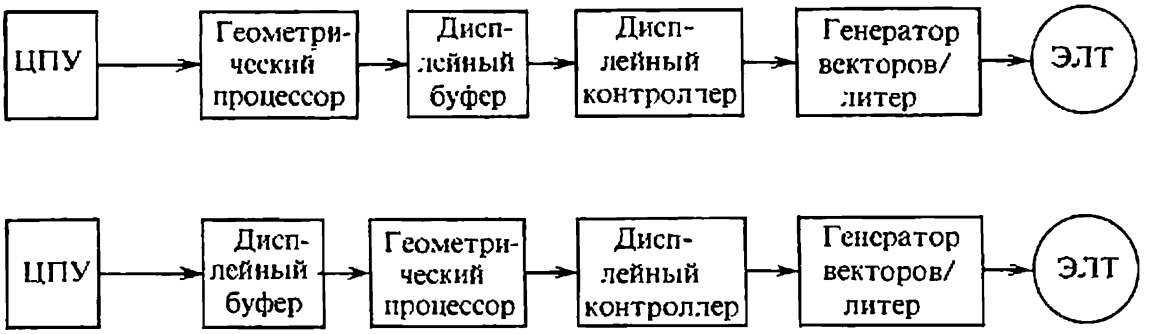



Рис. 1.3. Концептуальные блок-схемы векторных дисплеев с регенерацией.

(рис. 1.3, а) геометрический процессор работает медленнее, чем это необходимо при регенерации применяемых на практике изображений (от 4000 до 5000 векторов). Таким образом, геометрические данные, посылаемые центральным процессорным устройством (ЦПУ) графическому дисплею, обрабатываются до сохранения в дисплейном буфере. Значит, в нем содержатся только те инструкции, которые необходимы генератору векторов и литер для вывода изображения. Дисплейный контроллер считывает информацию из дисплейного буфера и посылает ее генератору векторов и литер. При достижении конца дисплейного буфера контроллер возвращается на его начало, и цикл повторяется снова.

При использовании первой схемы возникает идея двойной буферизации и раздельного изменения изображения и его регенерации. Так как в этой конфигурации геометрический процессор не успевает сгенерировать сложное новое или измененное изображение во время одного цикла регенерации, то дисплейный буфер делится на две части. В то время, как измененное изображение обрабатывается и записывается в одну половину буфера, дисплейный контроллер регенерирует ЭЛТ из другой половины буфера. При завершении изменения изображения буферы меняются ролями и этот процесс повторяется. Таким образом, новое или измененное изображение может генерироваться каждый второй, третий, четвертый и т. д. циклы регенерации. Использование двойной буферизации предотвращает одновременный вывод части старого и части нового измененного изображения в течение одного и более циклов регенерации.

Во второй схеме (рис. 1.3, b) геометрический процессор работает быстрее, чем необходимо для регенерации достаточно сложных изображений. В этом случае исходная геометрическая база данных, посланная из ЦПУ, сохраняется непосредственно в дисплейном буфере, а векторы обычно задаются в пользовательских (мировых) координатах в виде чисел с плавающей точкой. Дисплейный кон-

троллер за один цикл регенерации считывает информацию из дисплейного буфера, пропускает ее через геометрический процессор и результат передает генератору векторов. При таком способе обработки геометрические преобразования должны выполняться «на лету» в течение олного цикла регенерации.

При использовании контроллера любой схемы в дисплейном буфере существуют инструкции рисования каждого вектора, литеры и подкартинки. Следовательно, любой конкретный элемент может быть изменен независимо от любого другого. Эта особенность в совокупности с малым временем послесвечения люминофора ЭЛТ позволяет изображать динамическое движение. Эта идея иллюстрируется рис. 1.4, где показано изображение, выводимое во время четырех последовательных циклов. Сплошной отрезок — это отрезок, рисуемый для текущего цикла, а пунктирный отрезок — для предыдущего цикла. Между двумя циклами регенерации изменяются координаты конца отрезка точки $B$. Создается впечатление, что отрезок вращается вокруг точки $A$.

В большинстве случаев лишь часть изображения является динамически изменяемой. Реально большая часть картинки остается статичной. Такое разделение наводит на мысль о сегментации дисплейного буфера. Идея иллюстрируется рис. 1.5. Здесь неподвижны: горизонтальный отрезок, заштрихованная часть и буква $A$, используемые для показа опоры отрезка $AB$, т. е. они не изменяются от одного цикла регенерации к другому. В то же время для показа динамического движения положение конца отрезка $AB$ и буквы $B$ изменяются. Эти изолированные части базы данных изображения

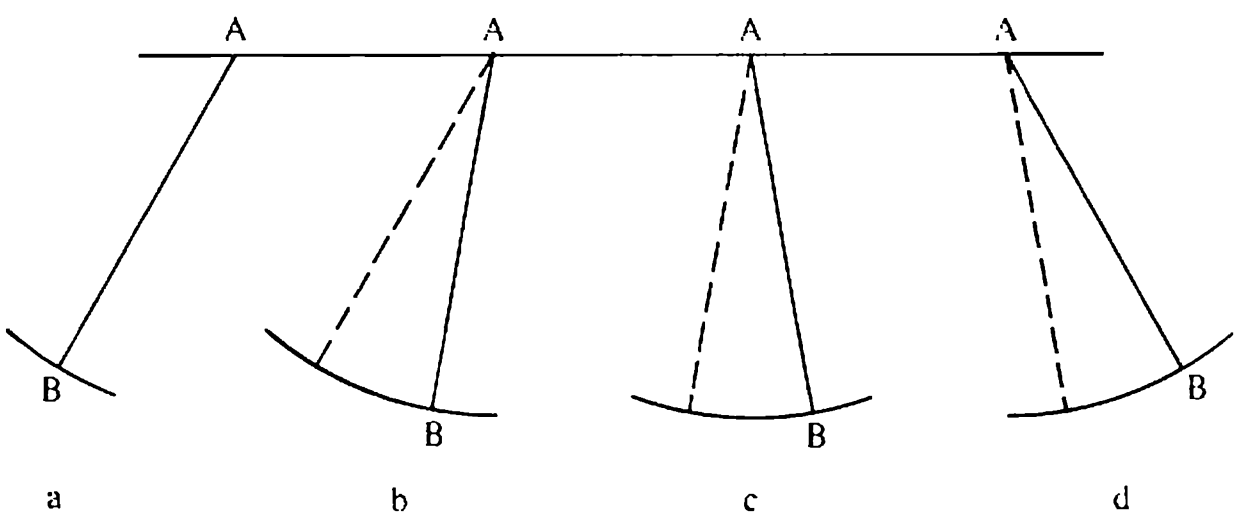



Рис. 1.4. Динамическое движение.

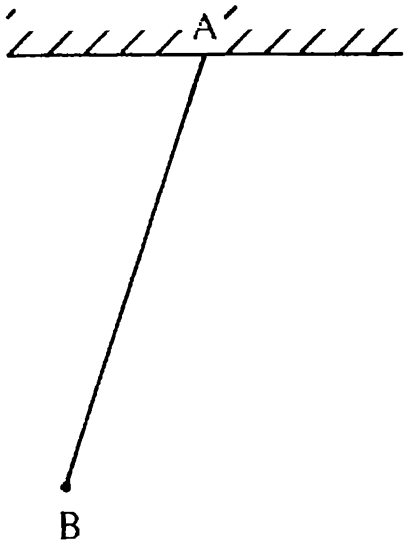



Рис. 1.5. Сегментация дисплейного буфера.

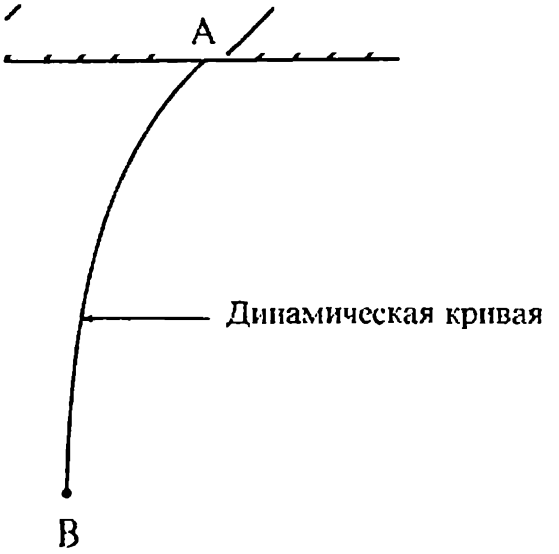



Рис. 1.6. Интеллектуальная сегментация дисплейного буфера.

помещены в отдельные сегменты дисплейного буфера. В конфигурации, показанной на рис. 1.3, а, геометрический процессор может игнорировать статический сегмент в дисплейном буфере, так как он не изменяется, а это существенно сокращает работу геометрического процессора при изменении рисунка. В данном случае должно модифицироваться только изображение в динамическом сегменте. Еще одним достоинством подобного метода является то, что при таком разделении сокращается количество данных, передаваемых из ЦПУ в геометрический процессор при каждом изменении изображения.

Разные типы сегментации возможны для конфигурации на рис. 1.3, b. Напомним, что здесь база данных изображения сохраняется в дисплейном буфере в мировых (пользовательских) координатах, а обработка изображения происходит «на лету», в каждом цикле регенерации. Для картинки, показанной на рис. 1.5, в дисплейном буфере создаются два сегмента — статический и динамический. В любом случае обработка изображения совершается «на лету». Информацию в динамическом сегменте можно изменить с помощью функций, предоставляемых геометрическим процессором. Таким образом, модификация изображения может происходить локально в графическом устройстве и для этого связь с ЦПУ не нужна. В частном случае, показанном на рис. 1.5, единственной функцией, необходимой для локального динамического изменения, является поворот вокруг точки $A$.

Для динамического изменения рис. 1.6 требуется связь с ЦПУ, т. е. некоторое интеллектуальное изменение изображения. Снова

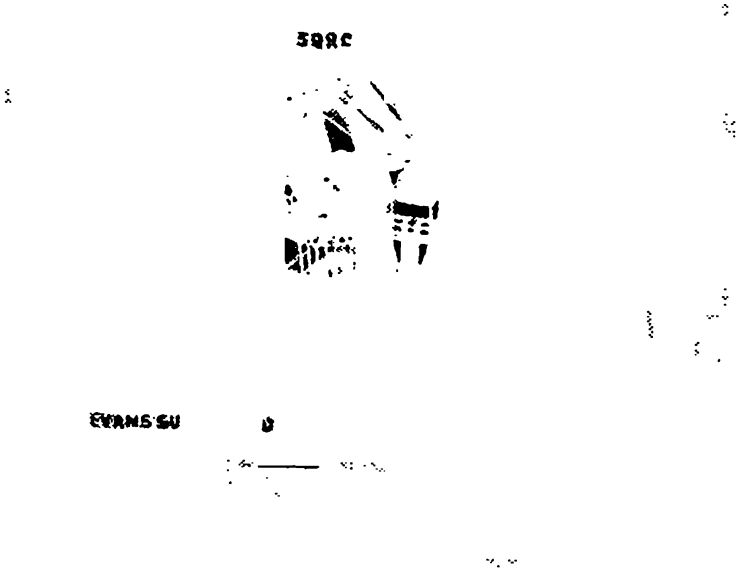


Рис. 1.7. Векторный дисплей с регенерацией. (С разрешения Evans & Sutherland Computer Corp.)

создаются два сегмента: статический сегмент, содержащий опорную линию, заштрихованную часть и букву $A$, и динамический сегмент, состоящий из кривой $AB$ и буквы $B$. Предположим, что форма кривой $AB$ должна изменяться от одного цикла регенерации к другому в зависимости от физических факторов. Это осуществляет прикладная программа в ЦПУ. Для модификации сегмента с динамически изменяющимся изображением должны быть посланы и сохранены в дисплейном буфере новые данные, например форма кривой.

Применение сегментации изображения не ограничивается только движением или анимацией, хотя само понятие было введено с помощью примеров из этой области. Сегментировано может быть любое изображение, что особенно полезно для интерактивных графических программ. Это понятие аналогично модульному программированию. Выбор модульных сегментов изображения, их размер и сложность зависят от конкретной прикладной области. Сложность элементов изображения варьируется от отдельных точек до полных описаний объектов. Дополнительную информацию по этому вопросу можно найти в [1-3].

Для того чтобы проиллюстрировать значимость скорости связи

или ширины полосы пропускания между ЦПУ и графическим устройством, рассмотрим затраты на интеллектуальное изменение кривой, описываемой 250 отрезками или точками. Каждая точка задается тремя координатами. Пусть для представления чисел с плавающей точкой используется шесть значащих цифр (литер) и пусть каждая литера задается отдельным байтом (8 бит). Тогда при скорости регенерации 30 кадров в секунду и при условии, что модификация производится в каждом цикле регенерации, требуемая ширина полосы пропускания связи составляет

30 [(кол-во тчк) (кол-во коорд./тчк) (кол-во знач. цифр/тчк) (кол-во бит/литера)]

или $$30 \cdot (250) \cdot (3) \cdot (6) \cdot (8) = 1\,080\,000 \text{ бит/с}$$

Таким образом, необходимая скорость передачи данных может легко превысить 1 Мбит/с. Для сложных трехмерных скульптурных поверхностей требуемая ширина полосы пропускания может легко возрасти в 10 раз, т. е. достигнуть 10 Мбит/с. В большинстве случаев для поддержки интеллектуальной динамической графики в реальном масштабе времени такие скорости требуют применения параллельного интерфейса или интерфейса прямого доступа в память (DMA) между ЦПУ и графическим устройством. На рис. 1.7 показан типичный векторный дисплей с регенерацией.

## 1.5. РАСТРОВЫЕ ГРАФИЧЕСКИЕ ДИСПЛЕИ С РЕГЕНЕРАЦИЕЙ ИЗОБРАЖЕНИЯ

Как дисплеи на запоминающих ЭЛТ, так и дисплеи с произвольным сканированием являются устройствами рисования отрезков, т. е. отрезок прямой может быть нарисован непосредственно из любой адресуемой точки в любую другую. Графическое устройство на растровой ЭЛТ работает по-другому. Растровое устройство можно рассматривать как матрицу дискретных ячеек (точек), каждая из которых может быть подсвечена. Таким образом, оно является точечно-рисующим устройством. Невозможно, за исключением специальных случаев, непосредственно нарисовать отрезок прямой из одной адресуемой точки или пиксела[1] в матрице в другую адресуемую точку или пиксел. Отрезок можно лишь аппроксимировать последовательностями точек (пикселов), близко лежащих к ре-

[1] В последнее время получил распространение термин ПЭЛ (pel), образованный от picture element. Он принят в качестве стандартного фирмой IBM. — *Прим. ред.*

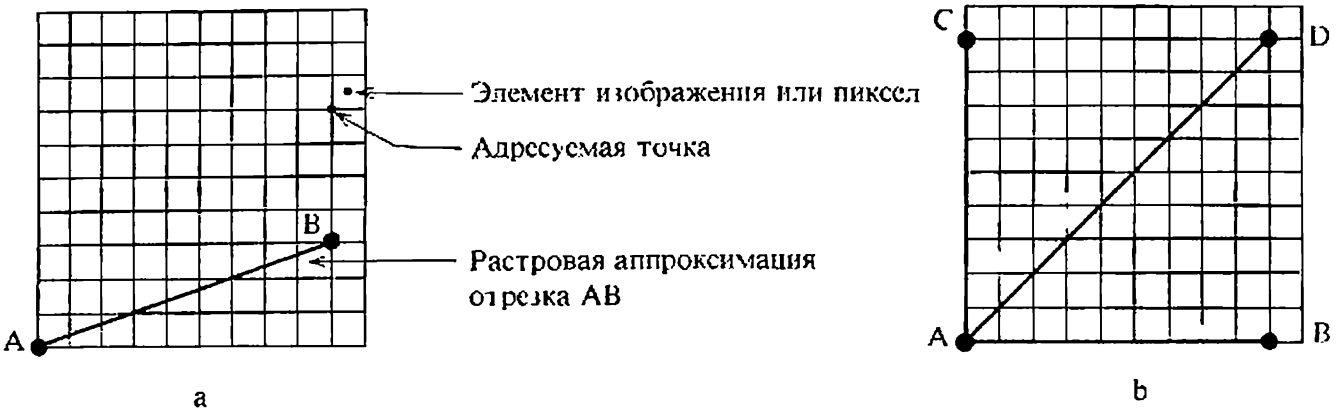



Рис. 1.8. Растровая развертка отрезка.

альной траектории отрезка. Эту идею иллюстрирует рис. 1.8. Отрезок прямой из точек (пикселов) получится только в случае горизонтальных, вертикальных или расположенных под углом 45° отрезков, как показано на рис. 1.8. Все другие отрезки будут выглядеть как последовательности ступенек. Это явление называется лестничным эффектом, или «зазубренностью». Методы, позволяющие устранять лестничный эффект, обсуждаются в гл. 2.

Чаще всего для графических устройств с растровой ЭЛТ используется буфер кадра. Буфер кадра представляет собой большой непрерывный участок памяти компьютера. Для каждой точки, или пиксела, в растре отводится как минимум один бит памяти. Эта память называется битовой плоскостью. Для квадратного растра размером 512× 512 требуется $2^{18}$ ($2^9 = 512$; $2^{18} = 512 \times 512$), или 262144 бита памяти в одной битовой плоскости. Изображение в буфере кадра строится побитно. Из-за того что бит памяти имеет только два состояния (двоичное 0 или 1), имея одну битовую плоскость, можно получить лишь черно-белое изображение. Битовая плоскость является цифровым устройством, тогда как растровая ЭЛТ — аналоговое устройство, для работы которого требуется электрическое напряжение. Поэтому при считывании информации из буфера кадра и ее выводе на графическое устройство с растровой ЭЛТ должно происходить преобразование из цифрового представления в аналоговый сигнал. Такое преобразование выполняет цифро-аналоговый преобразователь (ЦАП). Каждый пиксел буфера кадра должен быть считан и преобразован, прежде чем он будет отображен на растровой ЭЛТ. На рис. 1.9 приведена схема графического устройства с черно-белой растровой ЭЛТ, построенного на основе буфера кадра с одной битовой плоскостью.

Цвета или полутона серого цвета могут быть введены в буфер

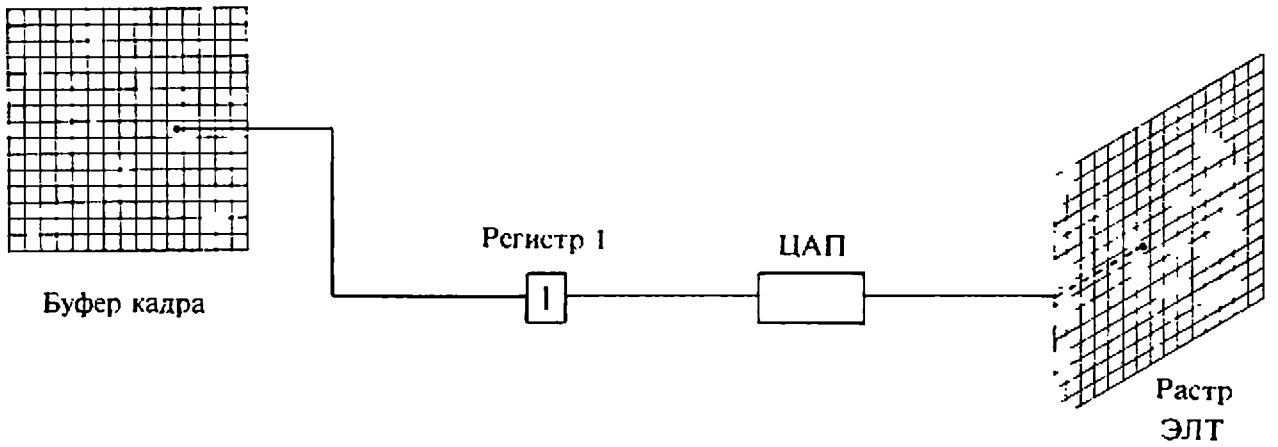



Рис. 1.9. Черно-белый буфер кадра (с одной битовой плоскостью) для растрового графического устройства.

кадра путем использования дополнительных битовых плоскостей. На рис. 1.10 показаны схема буфера кадра с $N$ битовыми плоскостями для градаций серого цвета. Интенсивность каждого пиксела на ЭЛТ управляется содержимым соответствующих пикселов в каждой из $N$ битовых плоскостей. В соответствующую позицию регистра загружается бинарная величина (0 или 1) из каждой плоскости. Двоичное число, получившееся в результате, интерпретируется как уровень интенсивности между 0 и $2^N - 1$. С помощью ЦАП это число преобразуется в напряжение между 0 (темный экран) и $2^N - 1$ (максимальная интенсивность свечения). Всего можно получить $2^N$ уровней интенсивности. Рис. 1.10 иллюстрирует систему с тремя битовыми плоскостями для 8 ($2^3$) уровней интенсивности. Для каждой битовой плоскости требуется полный объем памяти при данном разрешении растра: например, буфер кадра с тремя битовыми плоскостями для растра 512 × 512 занимает 786432 (3 * 512 * 512) битов памяти.

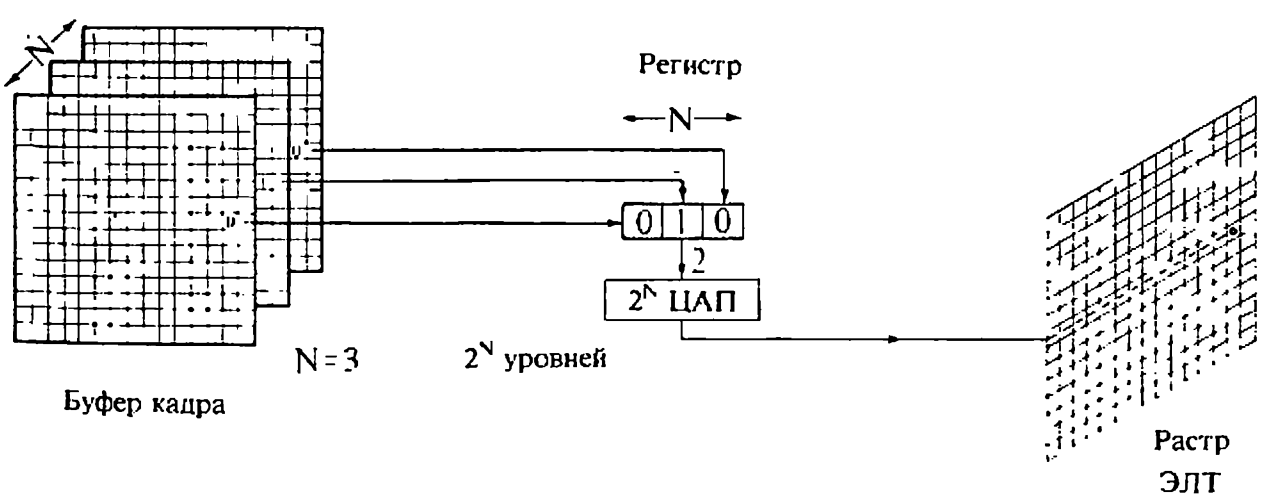



Рис. 1.10. Полутоновый черно-белый буфер кадра с N битовыми плоскостями.

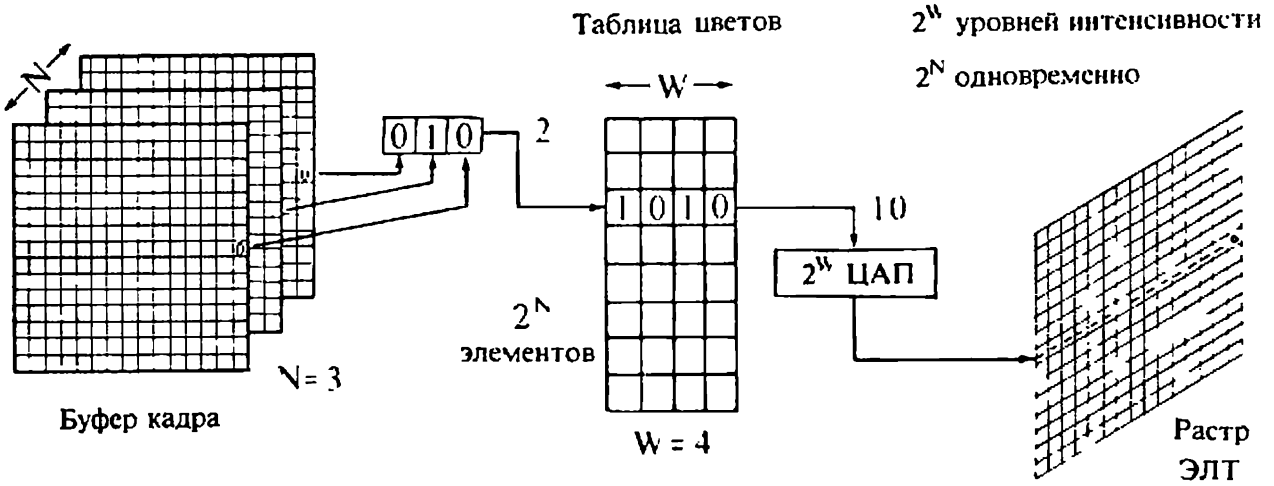



Рис. 1.11. Полутоновый черно-белый буфер кадра с N битовыми плоскостями и W-разрядной таблицей цветов.

Число доступных уровней интенсивности можно увеличить, незначительно расширив требуемую для этого память и воспользовавшись таблицей цветов, как это схематично показано на рис. 1.11. После считывания из буфера кадра битовых плоскостей получившееся число используется как индекс в таблице цветов. В этой таблице должно содержаться $2^N$ элементов. Каждый ее элемент может содержать $W$ бит, причем $W$ может быть больше $N$. В последнем случае можно получить $2^W$ значений интенсивности, но одновременно могут быть доступны лишь $2^N$ из них. Для получения дополнительных значений интенсивностей таблицу цветов необходимо изменить (перезагрузить).

Поскольку существует три основных цвета, можно реализовать простой цветной буфер кадра с тремя битовыми плоскостями, по одной для каждого из основных цветов. Каждая битовая плоскость управляет индивидуальной электронной пушкой для каждого из трех основных цветов, используемых в видеотехнике. Три основных цвета, комбинируясь на ЭЛТ, дают восемь цветов. Эти цвета и соответствующие им двоичные коды приведены в табл. 1.1. Схема простого цветового растрового буфера кадра показана на рис. 1.12.

Для каждой из трех цветовых пушек могут использоваться дополнительные битовые плоскости. На рис. 1.13 схематично показан цветной буфер кадра с 8 битовыми плоскостями на каждый цвет, то есть буфер кадра с 24 битовыми плоскостями. Каждая группа битовых плоскостей управляет 8-разрядным ЦАП. Каждая такая группа может генерировать 256 ($2^8$) оттенков или интенсивностей красного, зеленого или синего цвета. Их можно скомбинировать в 16 777 216 [$(2^8)^3 = 2^{24}$] возможных цветов. Это «полноцветный» буфер кадра.

**Таблица 1.1. Цветовые комбинации для простого буфера кадра с тремя битовыми плоскостями.**

| | Красный | Зеленый | Синий |
|---|---|---|---|
| Черный | 0 | 0 | 0 |
| Красный | 1 | 0 | 0 |
| Зеленый | 0 | 1 | 0 |
| Синий | 0 | 0 | 1 |
| Желтый | 1 | 1 | 0 |
| Голубой | 0 | 1 | 1 |
| Пурпурный | 1 | 0 | 1 |
| Белый | 1 | 1 | 1 |

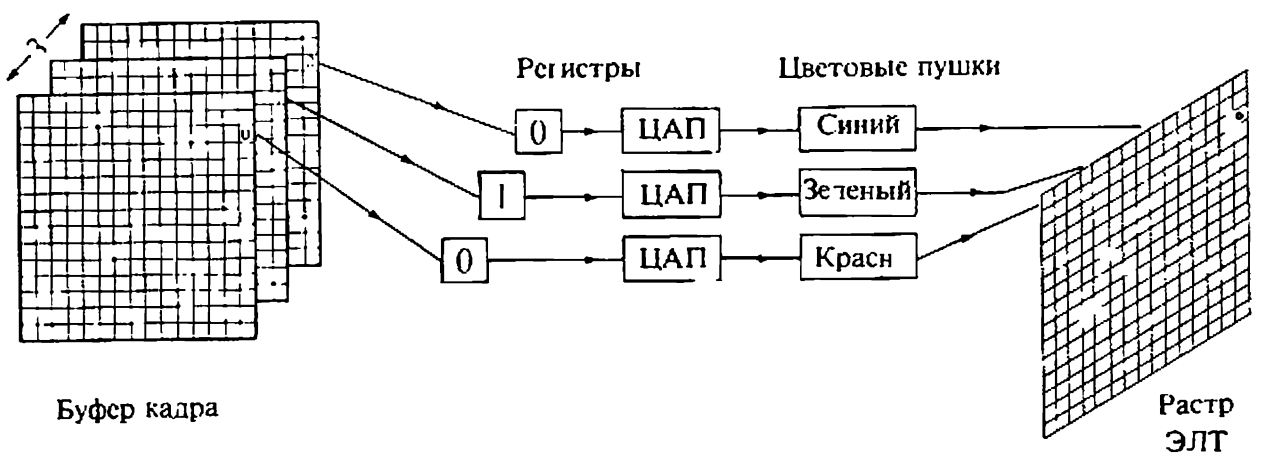



Рис. 1.12. Простой цветной буфер кадра.

Полноцветный буфер кадра может быть далее еще увеличен путем использования групп битовых плоскостей в качестве индексов в таблицах цветов, как это показано на рис. 1.14. При $N$ битах на цвет и $W$-разрядных элементах таблиц цветов одновременно может быть показано $(2^3)^N$ цветовых оттенков из палитры $(2^3)^W$ возможных цветов. Например, при буфере кадра с 24 битовыми плоскостями ($N = 8$) и тремя 10-разрядными таблицами цветов ($W = 10$) может быть получено 16 777 216 ($2^{24}$) цветовых оттенков из палитры 1 073 741 824 ($2^{30}$) цветов, т. е. около 17 млн. оттенков из палитры немногим меньше чем 1 миллиард цветов.

Из-за большого количества пикселов в растровых графических устройствах трудно достичь производительности, необходимой для работы в реальном времени, а также приемлемой скорости регенерации, или смены, кадра. Например, если среднее время доступа к каждому индивидуальному пикселу равно 200 нс ($200 \cdot 10^{-9}$), то для

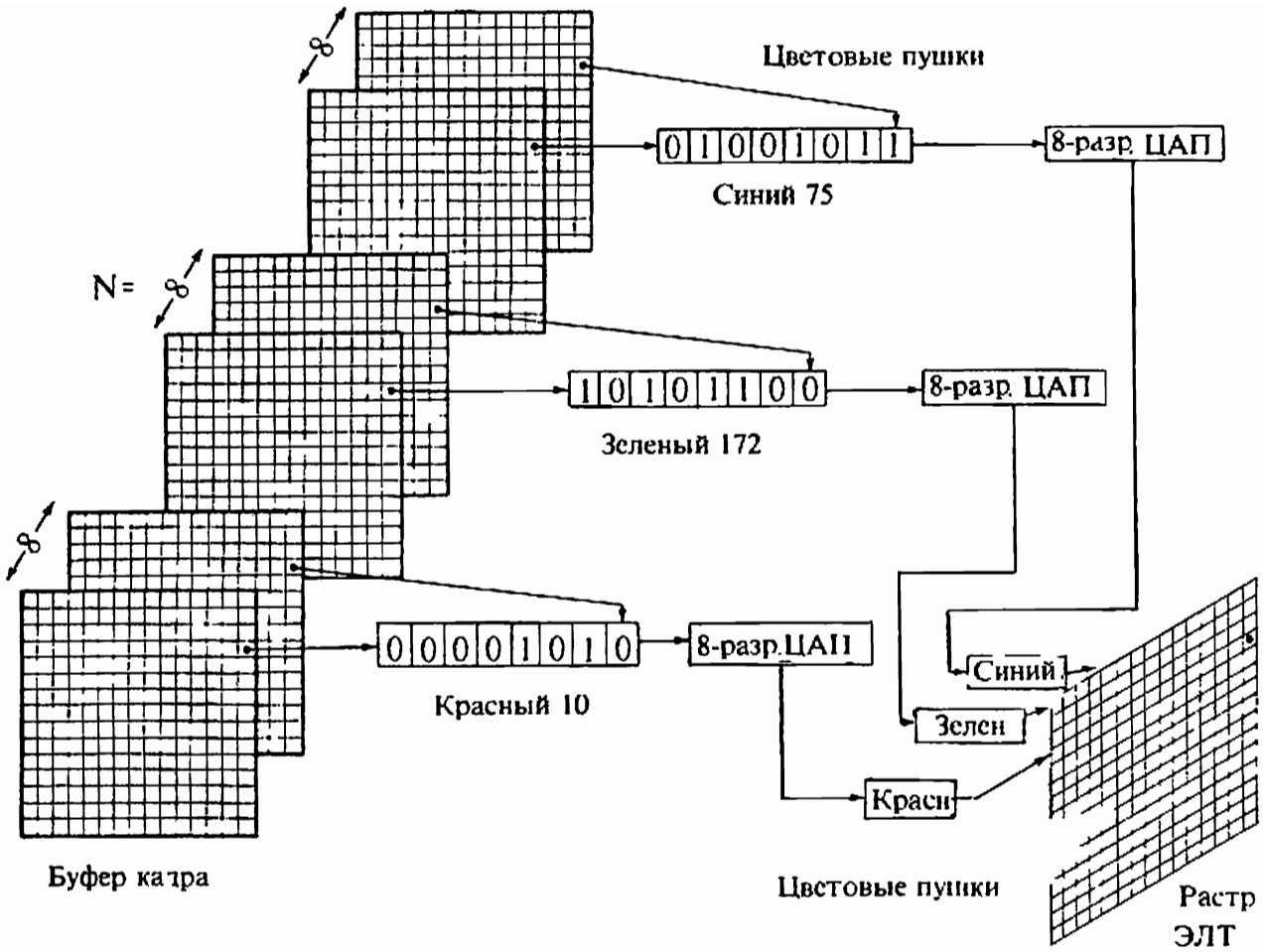



Рис. 1.13. Цветной буфер кадра с 24 битовыми плоскостями.

доступа ко всем пикселам кадра размером 512 × 512 потребуется 0.0524 с. Это эквивалентно скорости регенерации 19 кадров (картинок) в секунду, что значительно ниже минимально необходимой скорости 30 кадров в секунду. В буфере кадра размером 1024 × 1024 содержится немногим больше 1 млн бит (1 Мбит) и при среднем времени доступа 200 нс требуется 0.21 с для доступа ко всем пикселам. Это составляет 5 кадров в секунду. Буфер кадра размером 4096 × 4096 содержит 16.78 млн бит на каждую битовую плоскость! Доступ к ним займет 0.3 с. Для достижения скорости регенерации 30 кадров в секунду при таком растре требуется средняя эффективная скорость доступа 2 нс/пиксел.

Работа в реальном времени с растровыми графическими устройствами осуществляется путем одновременного доступа к группам по 16, 32, 64 и более пикселов. В случае цветного буфера кадра каждый пиксел может содержать до 32 бит, при этом все битовые плоскости для каждого пиксела доступны одновременно. При среднем времени доступа для каждой группы пикселов 1600 нс возмож-

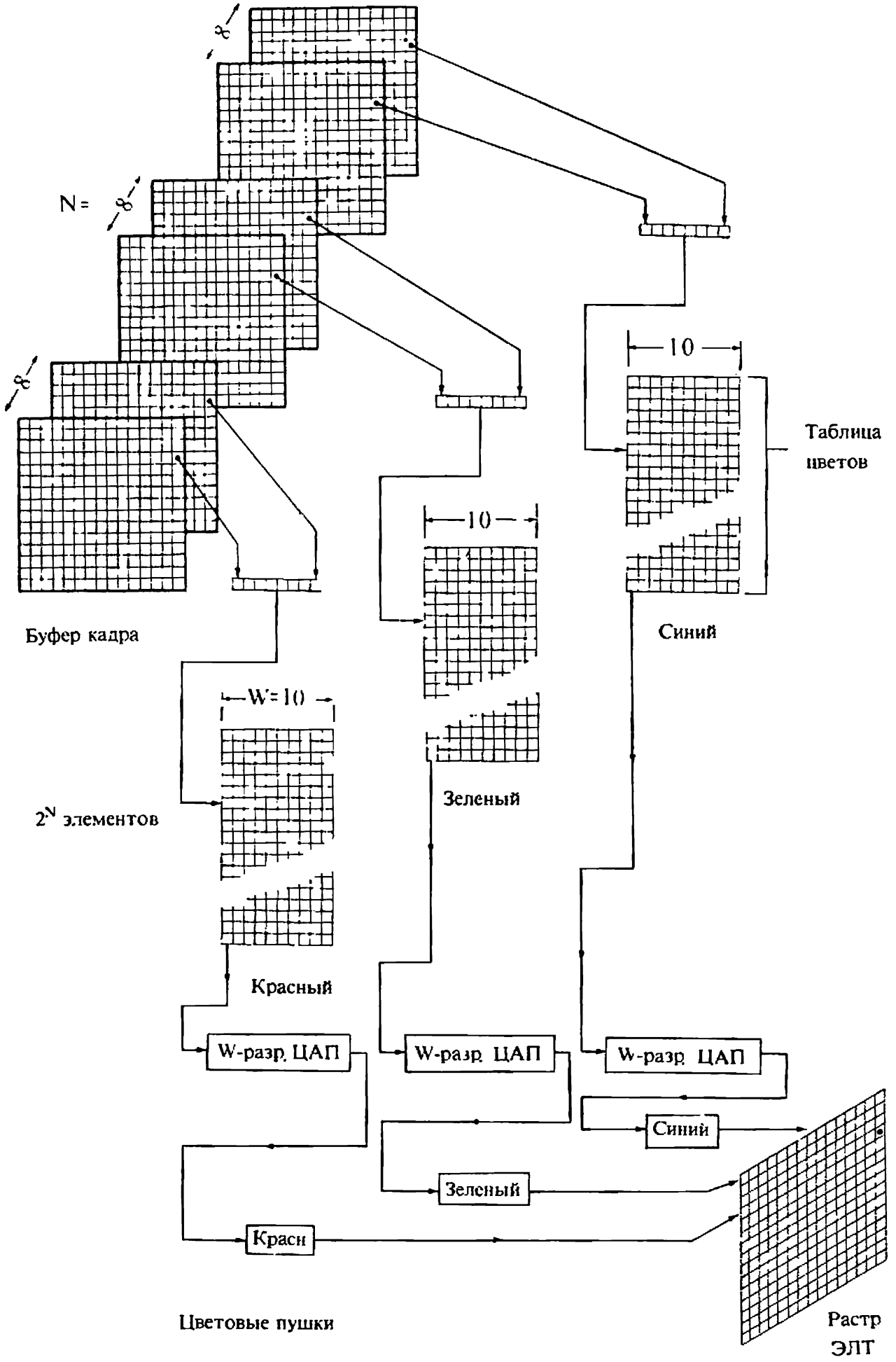



Рис. 1.14. Цветной буфер кадра с 24 битовыми плоскостями и 10-разрядной таблицей цветов.

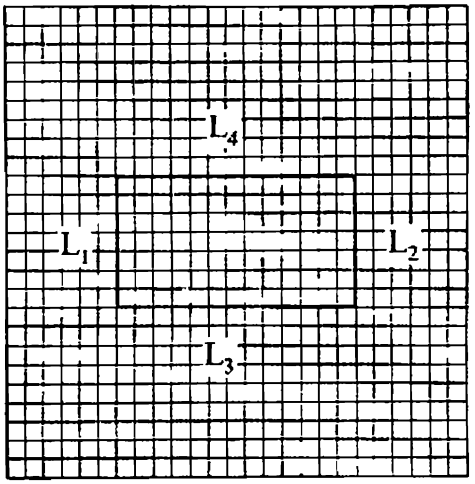



Рис. 1.15. Сплошные фигуры на растровом графическом устройстве.

на работа в реальном времени для буферов кадров размером 512 × 512 и 1024 × 1024.

Хотя производительности, необходимой для работы в реальном масштабе времени с приемлемыми скоростями регенерации, на растровых устройствах достичь труднее, чем на векторных дисплеях с регенерацией, на них легче изображать сплошные фигуры с плавными переходами цветов. Как показано на рис. 1.15, растровое представление сплошной «полигональной» фигуры концептуально просто. Здесь представление сплошной фигуры, ограниченной отрезками $L1$, $L2$, $L3$, $L4$, достигается установкой всех пикселов внутри ограничивающего многоугольника в соответствующий цвет в буфере кадра. Алгоритмы «растровой развертки» сплошной области обсуждаются в гл. 2.

## 1.6. УСТРОЙСТВО ЭЛЕКТРОННО-ЛУЧЕВОЙ ТРУБКИ

Описанный выше буфер кадра сам по себе не является устройством вывода, он просто применяется для хранения рисунка. Наиболее часто в качестве устройства отображения, используемого с буфером кадра, выступает видеомонитор. Чтобы понять принципы работы растровых дисплеев и в некоторой степени векторных дисплеев с регенерацией, нужно иметь представление о конструкции ЭЛТ и методах создания видеоизображения.

На рис. 1.16 схематично показана ЭЛТ, используемая в видеомониторах. Катод (отрицательно заряженный) нагревают до тех пор, пока возбужденные электроны не создадут расширяющегося облака (электроны отталкиваются друг от друга, так как имеют одинаковый заряд). Эти электроны притягиваются к сильно заряженному положительному аноду. На внутреннюю сторону расширенного конца ЭЛТ нанесен люминофор. Если бы электронам ни-

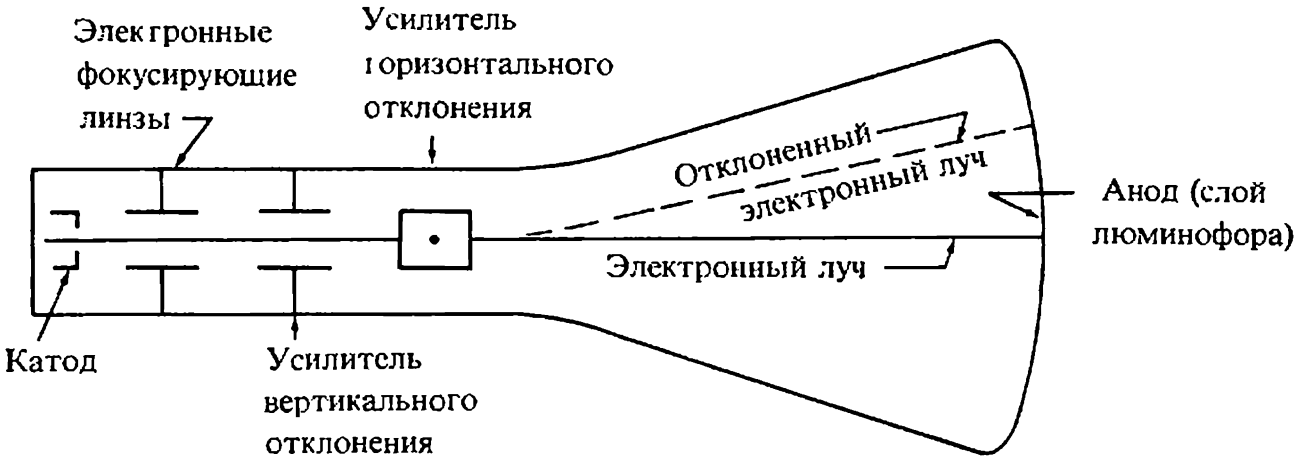



Рис. 1.16. Электронно-лучевая трубка.

что не препятствовало, то в результате их воздействия на люминофор весь экран ЭЛТ засветился бы ярким светом. Однако облако электронов с помощью электронных линз фокусируется в узкий, строго параллельный пучок. Теперь сфокусированный электронный луч дает одно яркое пятно в центре ЭЛТ. Луч отклоняется или позиционируется влево или вправо от центра и(или) выше или ниже центра с помощью усилителей горизонтального и вертикального отклонения.

Именно в данный момент проявляется отличие векторных дисплеев — как с запоминанием, так и с регенерацией — от растровых. В векторном дисплее электронный луч может быть отклонен непосредственно из любой произвольной позиции в любую другую произвольную позицию на экране ЭЛТ (аноде). Поскольку люминофорное покрытие нанесено на экран ЭЛТ сплошным слоем, в результате получается почти идеальная прямая. В отличие от этого в растровом дисплее луч может отклоняться только в строго определенные позиции на экране, образующие своеобразную мозаику. Эта мозаика составляет видеоизображение. Люминофорное покрытие на экране растровой ЭЛТ тоже не непрерывно, а представляет собой множество тесно расположенных мельчайших точек, куда может позиционироваться луч, образуя мозаику.

## 1.7. УСТРОЙСТВО ЦВЕТНОЙ РАСТРОВОЙ ЭЛТ

Цветная растровая ЭЛТ похожа на стандартную черно-белую ЭЛТ, описанную в предыдущем разделе. В ней находятся три электронные пушки, по одной на каждый основной цвет: красный, зеленый и синий. Электронные пушки часто объединены в треугольный блок, соответствующий подобному треугольному блоку точек крас-

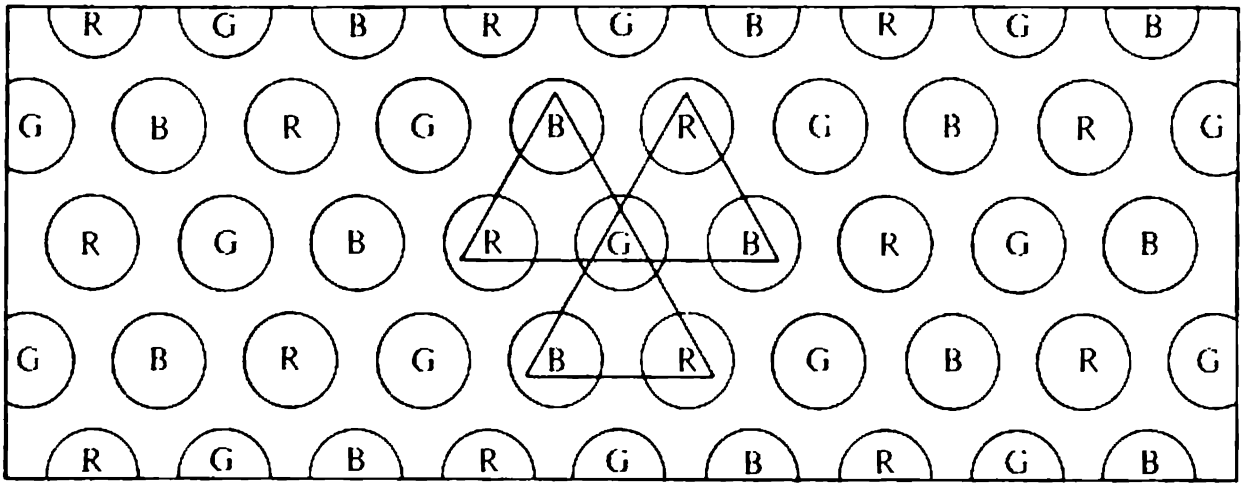



Рис. 1.17. Точечный люминофорный растр для ЭЛТ с теневой маской.

ного, зеленого и синего люминофоров на экране ЭЛТ (рис. 1.17)[1]. Для того чтобы электронные пушки возбуждали только соответствующие им точки люминофора (например, красная пушка возбуждала только точку красного люминофора), между электронными пушками и поверхностью экрана помещена перфорированная металлическая решетка. Это так называемая теневая маска стандартной цветной ЭЛТ с теневой маской. Отверстия в ней образуют такие же треугольные блоки, как и точки люминофора. Расстояния между отверстиями называются шагом. Цветовые пушки расположены таким образом, что их лучи сходятся и пересекаются в плоскости теневой маски (рис. 1.18). После прохождения через отверстие красный луч, например, защищен или маскирован от пересечения с зеленой или синей точкой люминофора. Он может пересечь

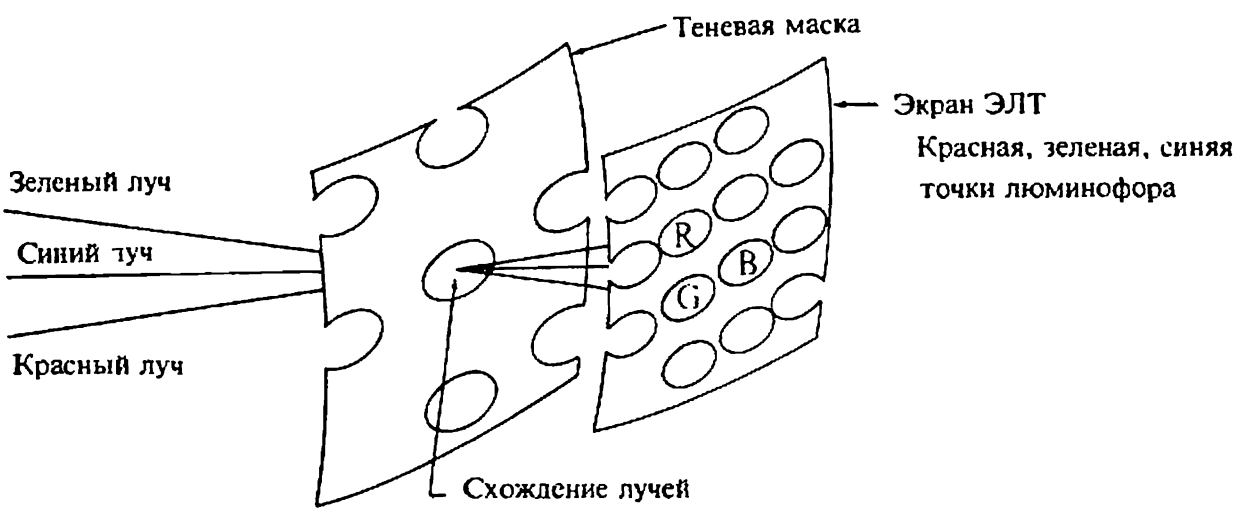



Рис. 1.18. Расположение электронной пушки и теневой маски цветной ЭЛТ.

[1] На этом и следующих рисунках R — сокращение от red (красный), G — от green (зеленый), B — от blue (синий). — *Прим. ред.*

лишь красную точку. Изменяя интенсивность электронного луча для каждого основного цвета, можно получить различные оттенки. Комбинация этих оттенков дает большое количество цветов для каждого пиксела. Обычно в дисплее с высоким разрешением на каждый пиксел приходится от двух до трех цветовых триад.

## 1.8. СИСТЕМЫ С ТЕЛЕВИЗИОННЫМ РАСТРОМ

Процесс преобразования хранящейся в буфере кадра растровой картинки в упорядоченный набор точек на телеэкране называется растровой разверткой. Сканируемый набор точек и частота воспроизведения основаны как на особенностях визуального восприятия, так и на принципах электроники. Системе визуального восприятия человека требуется конечный интервал времени для рассмотрения элементов картины. Однако обеспечить впечатление непрерывности позволит только такой интервал, который настолько мал, что инерция зрительного восприятия перекрывает мерцание. На мерцание влияет ряд факторов, включая яркость изображения и конкретный люминофор, используемый для экрана ЭЛТ. Опыт показывает, что для практических целей минимальной скоростью вывода или изменения изображения является 25 кадров в секунду при условии, что минимальная скорость регенерации или воспроизведения в два раза больше, т. е. 50 кадр/с. Аналогичная ситуация имеет место при демонстрации кинофильма. При этом показывается 24 кадр/с, но каждый кадр показывается дважды, и в результате получается эффективная скорость воспроизведения 48 кадр/с. Таким образом, для фильма скорость изменения равна 24, а скорость регенерации — 48. В телевидении тот же самый эффект достигается с помощью метода, называемого чересстрочной разверткой.

Телевидение использует метод растрового сканирования. В американской стандартной видеосистеме используется в совокупности 525 горизонтальных строк с кадровым, или видовым, отношением 4:3, т. е. высота видовой области равна трем четвертям ее ширины. Скорость воспроизведения, или смены кадра, составляет 30 кадр/с. Однако каждый кадр делится на два поля, каждое из которых содержит по половине картинки. Эти два поля чередуются или перемежаются. Они попеременно показываются через каждые 1/60 с. Одно поле содержит все строки с нечетными номерами (1, 3, 5,...), а другое — с четными (2, 4, 6,...). Сканирование начинается в верхнем левом углу экрана с нечетного поля. Каждая строка в поле сканируется или представляется слева направо. В то время, как электронный луч движется поперек экрана слева направо, он

также перемещается вертикально вниз, но с намного меньшей скоростью. Таким образом, «горизонтальная» сканирующая строка на самом деле слегка наклонена. При достижении правого края экрана луч делают невидимым и быстро возвращают к левому краю. Такой горизонтальный возврат луча обычно занимает около 17% времени, отведенного для одной сканирующей строки. Затем этот процесс повторяется со следующей нечетной строкой. Так как половина от 525 равна 262 ½ строки, то при завершении сканирования поля нечетных строк луч окажется в центре нижней строки экрана (рис. 1.19 и 1.20). Луч затем быстро переводится в центр верхней стороны экрана. Так производится вертикальный перевод луча для нечетного поля. Время, затрачиваемое на него, эквивалентно времени, затрачиваемому на сканирование 21 строки. Затем показывается поле четных строк, после чего луч оказывается в нижнем правом углу. Перевод луча для поля четных строк возвращает его в верхний левый угол, и весь процесс повторяется снова. Таким образом, показываются два поля для каждого кадра, т. е. 60 полей в секунду. Данный метод существенно уменьшает мерцание, так как глаз воспринимает скорость воспроизведения поля.

Хотя в принятой в США стандартной видеосистеме предусмотрено 525 строк, на самом деле видимы только 483 строки, так как время, затрачиваемое на сканирование 21 строки, уходит на вертикальный перевод луча[1]. В течение этого времени электронный луч

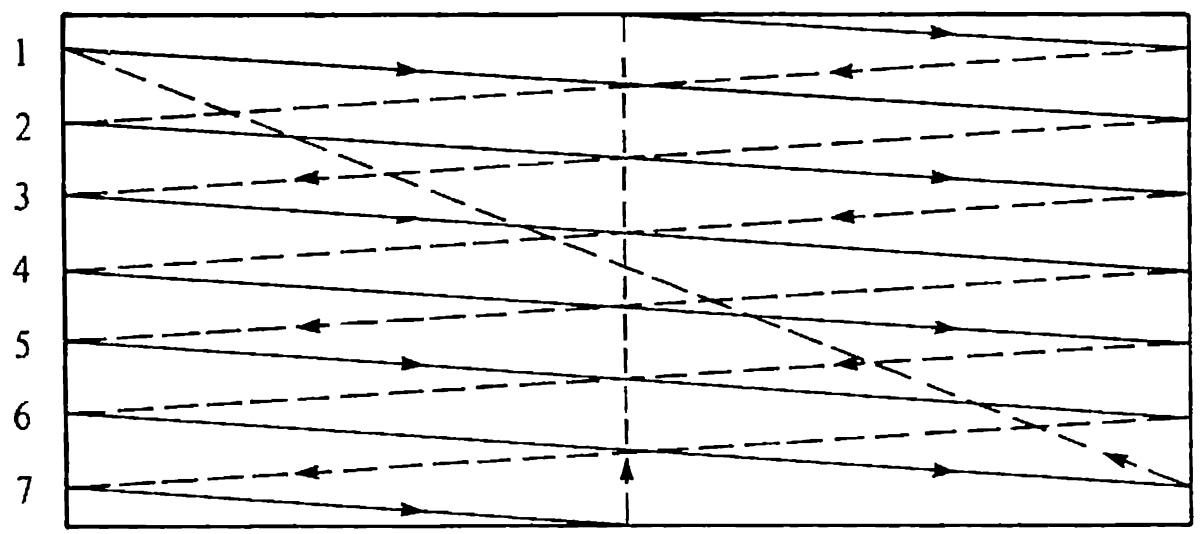



Рис. 1.19. Схема чересстрочной развертки для растра из семи строк. Нечетное поле начинается со строки 1. Пунктиром обозначен горизонтальный обратный ход луча. Вертикальный обратный ход луча для нечетного поля начинается внизу в центре, а для четного поля — внизу справа.

[1] Во многих растровых графических устройствах это время используется для обработки другой информации.

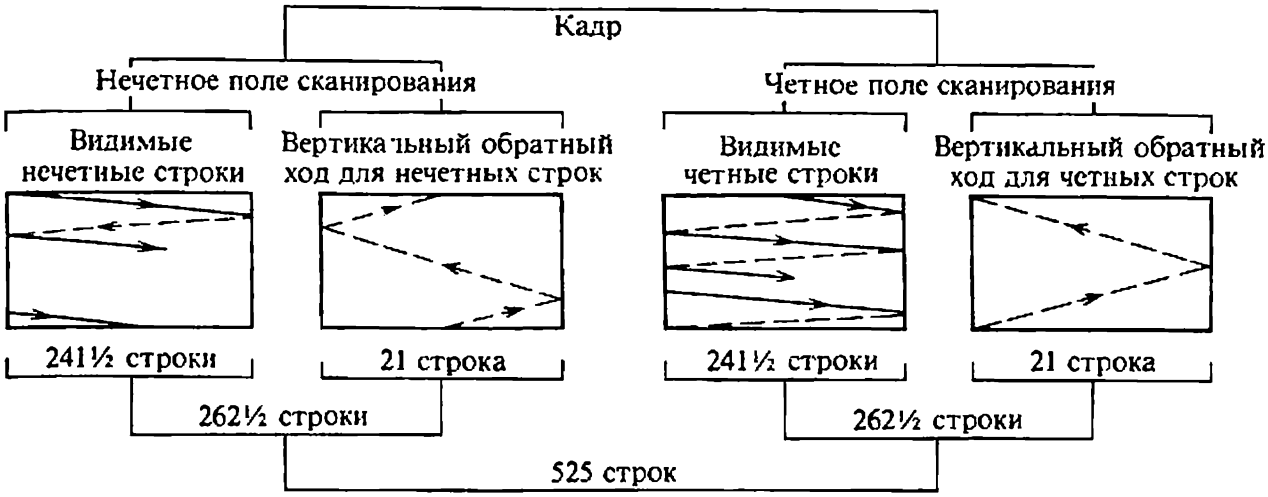



Рис. 1.20. Схема 525-строчного стандартного кадра

невидим или выключен. Время, отпущенное на каждую сканирующую строку, легко подсчитывается для частоты воспроизведения кадра 30/с следующим образом:

$$\frac{1}{30}\frac{\text{с}}{\text{кадр}} \times \frac{1}{525}\frac{\text{кадр}}{\text{строка}} = 63.5\,\frac{\text{мкс}}{\text{строка}}$$

Так как приблизительно 10 ½ мкс тратится на горизонтальный перевод луча, то вывод видимой части каждой строки должен быть завершен за 53 мкс. В строке при нормальном соотношении сторон видеообласти, равном 4:3, находится 644 пиксела. Таким образом, время, отпущенное на считывание и вывод пиксела, равно

$$53\,\frac{\text{мкс}}{\text{строка}} \times \frac{1}{644}\frac{\text{строка}}{\text{пиксел}} = 82\ \text{нс}$$

Во многих растровых дисплеях, построенных на основе буфера кадра, применяется разрешение изображения 512 пикселов в строке. На считывание и вывод пиксела при таком разрешении отводится приблизительно 103 наносекунды. Аналогичные результаты получаются для 625-строчной видеосистемы с частотой воспроизведения 25 кадр/с, используемой в Великобритании и большинстве стран Европы.

Метод чересстрочной развертки не является абсолютно необходимым при выводе видеоизображения, однако изображение без чередования строк будет несовместимым со стандартным телевизионным приемником. При отсутствии чересстрочной развертки для устранения мерцания придется увеличить частоту воспроизведения до 60 кадр/с, а это, конечно, сокращает в 2 раза время для обработки пиксела. Более высокое разрешение по числу строк и количе-

ству пикселов в строке также уменьшает это время; например, при разрешении 1024 × 1024 на считывание и вывод пиксела отводится в 4 раза меньше времени, чем при разрешении 512 × 512, — приблизительно 25 нс! В этом случае потребуется очень быстрая память для буфера кадра и такой же быстрый ЦАП.

## 1.9. ДИАЛОГОВЫЕ УСТРОЙСТВА

После того как изображение построено на экране, необходимо как-то взаимодействовать с ним или модифицировать его. Для удовлетворения этой потребности был разработан ряд диалоговых устройств. Среди них можно назвать планшет, световое перо, рычаг, мышь, ручки для ввода скалярных величин, функциональные переключатели или кнопки и, разумеется, обычную алфавитно-цифровую клавиатуру. Прежде чем перейти к обсуждению этих физических устройств, рассмотрим функциональные возможности диалоговых графических устройств. Обычно насчитывают четыре или пять таких классов [1-3 — 1-6]. Логическими диалоговыми устройствами являются локатор, валюатор, селектор и кнопка. Из-за широкой распространенности алфавитно-цифровой клавиатуры ее часто выделяют в пятый класс, называемый клавиатурой. На самом деле клавиатуру можно концептуально и функционально считать набором кнопок.

Функцией локатора является выдача координатной информации в двух или трех измерениях. Обычно выдаваемые значения координат находятся в пространстве устройства (концептуальном пространстве) и могут быть как относительными, так и абсолютными. Валюатор применяется для ввода одиночной величины. Обычно это вещественное число между нулем и некоторым вещественным максимумом. Кнопка используется для выбора и активирования событий или процедур, управляющих ходом диалога. Кнопка обычно предоставляет двоичную («включено» или «выключено») цифровую информацию. В функцию селектора входит идентификация или выбор объектов или подкартинок в выведенном изображении. Логическая клавиатура обрабатывает текстовую информацию. На рис. 1.21 показана типичная клавиатура.

Наиболее общим устройством класса локаторов является планшет (рис. 1.22). Планшеты можно использовать либо отдельно, либо в комбинации с графическим дисплеем на ЭЛТ. В первом случае их часто называют оцифровывателями. Сам по себе планшет состоит из плоской поверхности и карандаша (похожего на

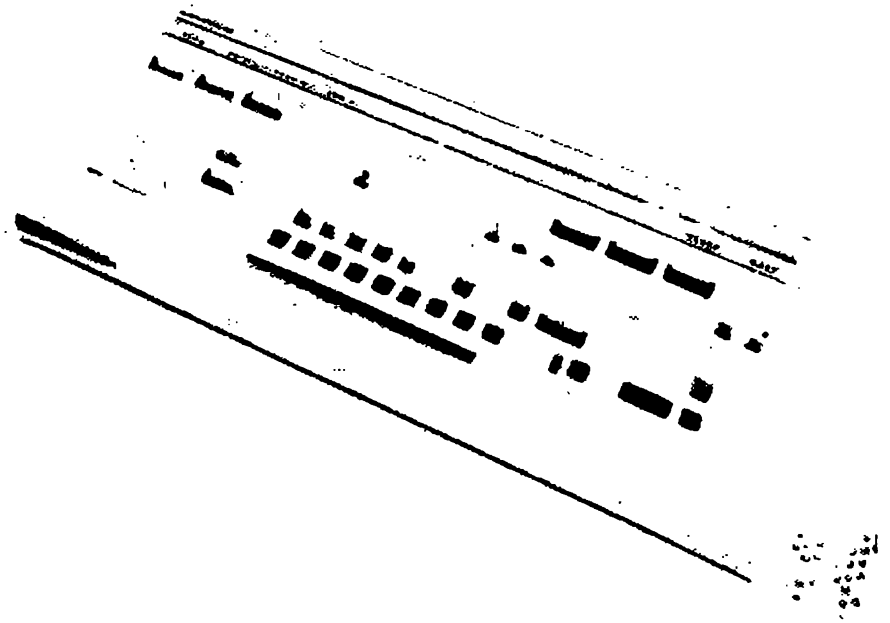


Рис. 1.21. Алфавитно-цифровая клавиатура. (С разрешения Evans & Sutherland Computer Corp.)

обычный карандаш), который служит для указания точки на поверхности планшета. Часто существует возможность узнать некоторую информацию о расстоянии между карандашом и планшетом: близко или далеко от поверхности находится карандаш. При использовании вместе с дисплеем на ЭЛТ обратная связь на экране обеспечивается с помощью небольшого символа (курсора), отслеживающего перемещение карандаша по поверхности планшета.

Рис. 1.22. Типичный планшет. (С разрешения фирмы Adage)

Рис. 1.23. Трехмерный звуковой планшет. (С разрешения фирмы Science Accessories Corp.)

При использовании в качестве отдельного оцифровывателя обратная связь обеспечивается с помощью цифровых отсчетов.

Планшеты выдают координатную информацию в двух или трех измерениях. На рис. 1.23 показан трехмерный планшет. Значения, выдаваемые планшетом, представлены в координатах устройства. Они программно преобразуются в пользовательские координаты. Обычное разрешение и точность составляют от 0.025 до 0.0025 см. При использовании вместе с дисплеем разрешение планшета должно совпадать или превосходить разрешение дисплея.

При создании планшетов используется ряд различных принципов. В первом планшете RAND [1-7] использовалась ортогональная сетка отдельных проводов, расположенных под поверхностью. Каждый провод кодируется таким образом, что карандаш, действующий как приемник информации, в каждой точке пересечения (проводов) получал уникальный цифровой код. Декодирование этого кода давало координаты $x$, $y$ карандаша. Очевидными ограничениями на разрешение таких матрично-кодируемых планшетов являются плотность расположения проводов и способность приемника обрабатывать уникальный код. Точность ограничивается линейностью конкретных проводов, а также их параллельностью в двух ортогональных направлениях.

В одной интересной реализации планшета применяются звуковые волны. Карандаш используется для создания искры электрического разряда, вызывающей звуковую волну. Она движется во всех направлениях от карандаша по поверхности планшета, образуя круговой звуковой фронт. На его краях в перпендикулярных направлениях смонтированы два чувствительных ленточных микрофона. Координатные расстояния можно определить с помощью точного измерения времени, за которое звуковая волна доходит от карандаша до микрофонов. Данную методику можно распространить на три измерения (рис. 1.23).

Наиболее популярны конструкции планшета, в основе которых лежит электромагнитный принцип. При этом электрические импульсы проходят через пластину из магнитострикционного материала, используемую в качестве поверхности планшета. Время, за которое чередующиеся импульсы, параллельные координатным осям $x$ и $y$, доходят от края планшета до карандаша, определяется с помощью карандаша и соответствующих индикаторов. Эти данные легко преобразуются в координаты $x$, $y$.

Сенсорная панель похожа на планшет и относится к классу локаторов. В типичной сенсорной панели на двух смежных сторонах расположены источники света, а на двух противоположных смежных сторонах смонтированы светочувствительные элементы. Любой предмет, например палец, прерывая два ортогональных луча света, позволяет определить пару координат $x$, $y$. Из-за низкого разрешения этот прибор лучше всего использовать для грубых операций указания. В этом качестве сенсорную панель часто монтируют поверх экрана ЭЛТ.

В таких локаторных устройствах, как рычаг, шар и мышь, в качестве делителя напряжения часто используются чувствительные переменные резисторы или потенциометры. Аналогичным образом реализуются и ручки для ввода скалярных величин, которые являются устройствами класса валюаторов. Точность всех этих устройств зависит от качества потенциометра и обычно колеблется от 0.1 до 10% всего диапазона измерения. Хотя разрешение потенциометра по существу и бесконечно, но его использование в цифровых системах требует аналого-цифрового (АЦ) преобразования. Обычно разрешение АЦ преобразователя варьируется в диапазоне от 8 до 14 бит, т. е. от 1/256 до 1/16384. Валюаторы также реализуются с помощью цифровых преобразователей углового положения, которые для каждого инкрементального угла поворота ручки выдают цифровой результат. Типичное разрешение — от 1/256 до 1/1024.

Типичным валюатором является рычаг (рис. 1.24). Подвижный рычаг обычно оборудуется двумя валюаторами — либо потенциометрами, либо преобразователями углового положения, смонтированными на основании прибора. Валюаторы выдают результаты, пропорциональные смещению ручки. В рычаг легко добавить третью степень свободы, например, используя третий валюатор для измерения угла поворота рычага. Для обратной связи обычно используют графический курсор.

Шар во многом аналогичен рычагу. Чаще всего его можно встретить в радарных установках, например, в системах управления воздушными перевозками. Сферический шар смонтирован в основании прибора, причем над поверхностью возвышается только его часть. Шар может свободно вращаться в любом направлении. В основании смонтированы два валюатора (потенциометры либо преобразователи углового положения), которые улавливают поворот шара и выдают результаты, пропорциональные относительным величинам углов. Вдобавок к обратной связи в виде обычного курсора пользователи получают тактильную обратную связь в виде скорости поворота или углового импульса вращения шара.

И рычаг, и шар имеют фиксированное положение с фиксированным началом координат. Мышь [1-8], напротив, имеет только относительное начало координат. Мышь состоит из двух покрытых

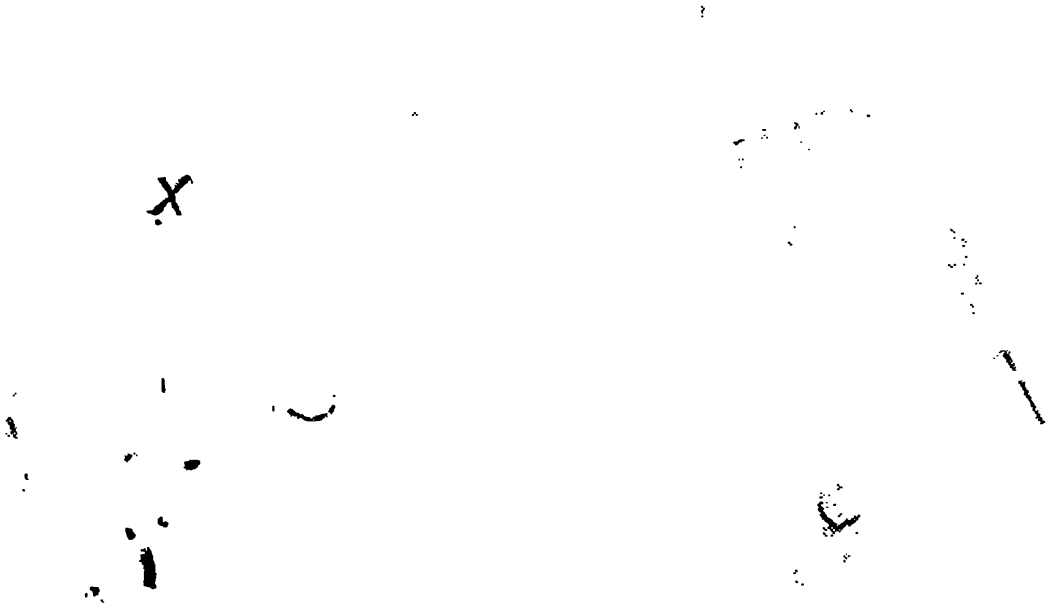


Рис. 1.24. Рычаг. (С разрешения фирмы Measurement Systems.)

Рис. 1.25. Мышь. (С разрешения фирмы Apple Computer.)

резиной колес, расположенных под прямым углом в маленьком, легком футляре. При движении мыши по поверхности колеса управляют осями двух валюаторов (потенциометров или преобразователей углового положения). Совокупное движение осей дает координаты $x, \cdot y$. Мышь показана на рис. 1.25. Устройство можно поднять, переместить в любом направлении и поставить на поверхность, по-другому ориентировав. В этом случае изменяется система координат, в которой генерируются данные, т. е. мышь, но не сама система координат, связанная с данными. Используемый для обратной связи курсор не движется, когда мышь не контактирует с поверхностью. Из-за проскальзывания колес, особенно при диагональных передвижениях, у мыши уменьшается точность. С недавних пор появились в продаже мыши, работающие как на оптическом, так и на магнитном принципе. В обоих случаях исключаются неточности, связанные с проскальзыванием колес.

Вероятно, наипростейшими из валюаторов являются ручки для ввода скалярных величин. Такие ручки, показанные на рис. 1.26, являются по существу чувствительными вращающимися потенциометрами или точными цифровыми преобразователями углового положения, которые обычно объединяют в группы. Такие устройства

Рис. 1.26. Набор ручек для ввода скалярных значений. (С разрешения фирмы Evans & Sutherland Computer Corp.)

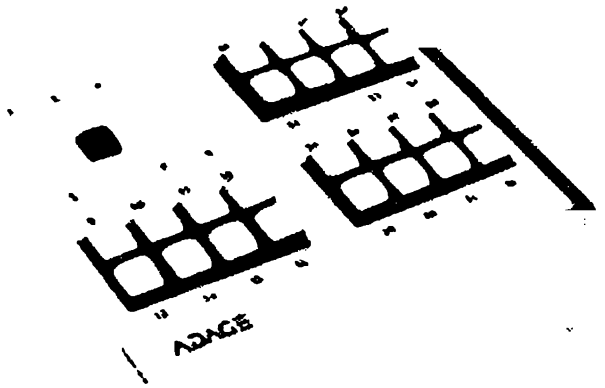



Рис. 1.27. Функциональные кнопки. (С разрешения фирмы Adage.)

особенно хорошо подходят для задания начальных значений для функций поворота, переноса, масштабирования или трансфокации.

Кнопки или функциональные переключатели, показанные на рис. 1.27, являются переключателями рычажного или нажимного типа. Они могут быть постоянно нажаты, постоянно отжаты, либо иметь кратковременный контакт. Наиболее удобный тип функционального переключателя объединяет в себе обе эти возможности. Обычно имеется возможность программно управлять световыми индикаторами, указывающими, какие переключатели или кнопки в данный момент активны. Часто кнопки и переключатели включают в другие устройства. Например, на тонком конце карандаша планшета обычно располагается переключатель, срабатывающий при нажатии карандаша. На мыши тоже находится одна или несколько кнопок.

Единственным подлинным устройством указания является световое перо. Перо, схематически показанное на рис. 1.28, содержит чувствительный фотоэлемент и соответствующую электрическую цепь. Основная информация, поставляемая световым пером — это синхронизация[1)], поэтому она зависит от того, что изображается на рисунке, и от порядка, в котором он выводится. Это исключает ис-

---

[1)] Между попадающим в поле зрения пера электронным лучом и командами программы дисплейного контроллера, вызвавшими попадание луча в данную точку. — *Прим. перев.*

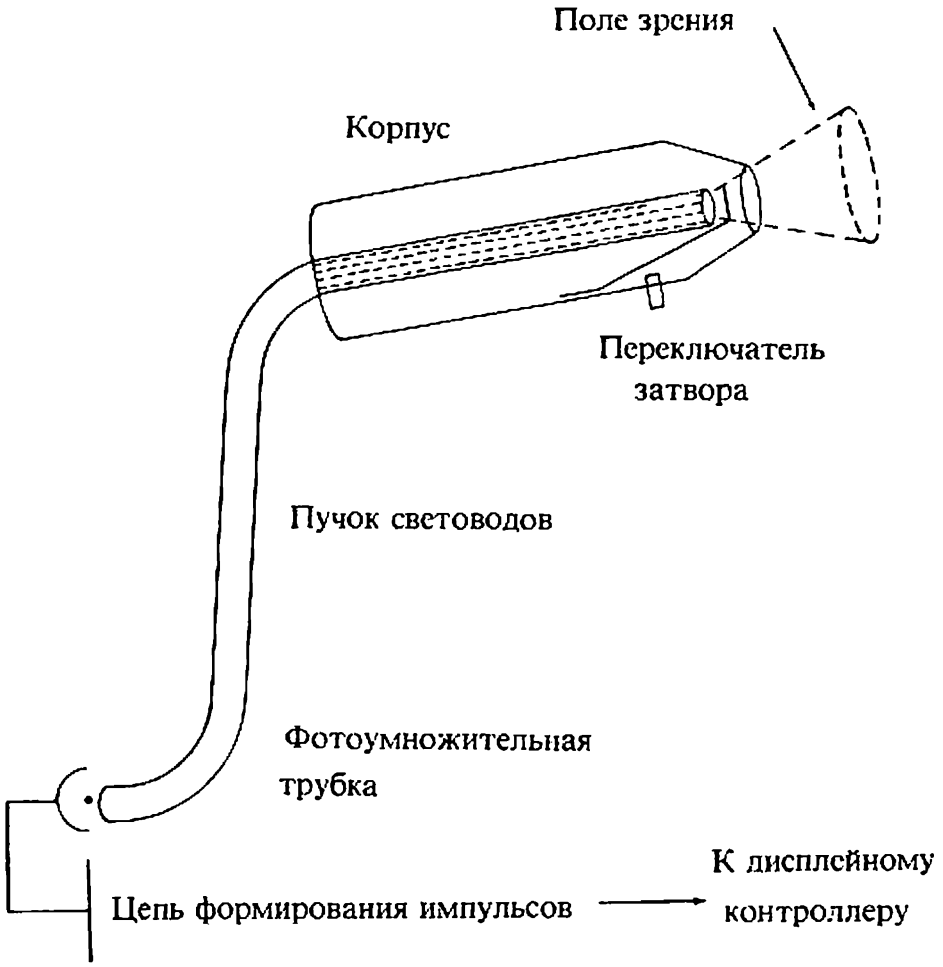



Рис. 1.28. Схема светового пера.

пользование светового пера с дисплеями на запоминающей ЭЛТ. Его можно использовать только с дисплеями с регенерацией — векторными или растровыми.

В тот момент, когда активированное световое перо оказывается над той областью ЭЛТ векторного дисплея, в которой происходит рисование, дисплейному контроллеру посылается сигнал. Он позволяет определить конкретную команду в дисплейном буфере, выполняемую в этот момент времени. Прослеживание команд назад с помощью дисплейного контроллера позволяет определить отрезок, объект или подкартину, которые были указаны. Используя графический курсор, световое перо можно применять также как локатор для векторного устройства с регенерацией.

Так как в растровом дисплее изображение генерируется в фиксированной последовательности, световое перо используется для определения горизонтальной сканирующей строки (координаты $y$) и позиции в этой строке (координаты $x$). Отслеживая в контроллере программу, можно снова определить, какой отрезок, объект или подкартина были указаны. Этот метод несколько усложняется в случае чересстрочной развертки. Приведенное выше описание означа-

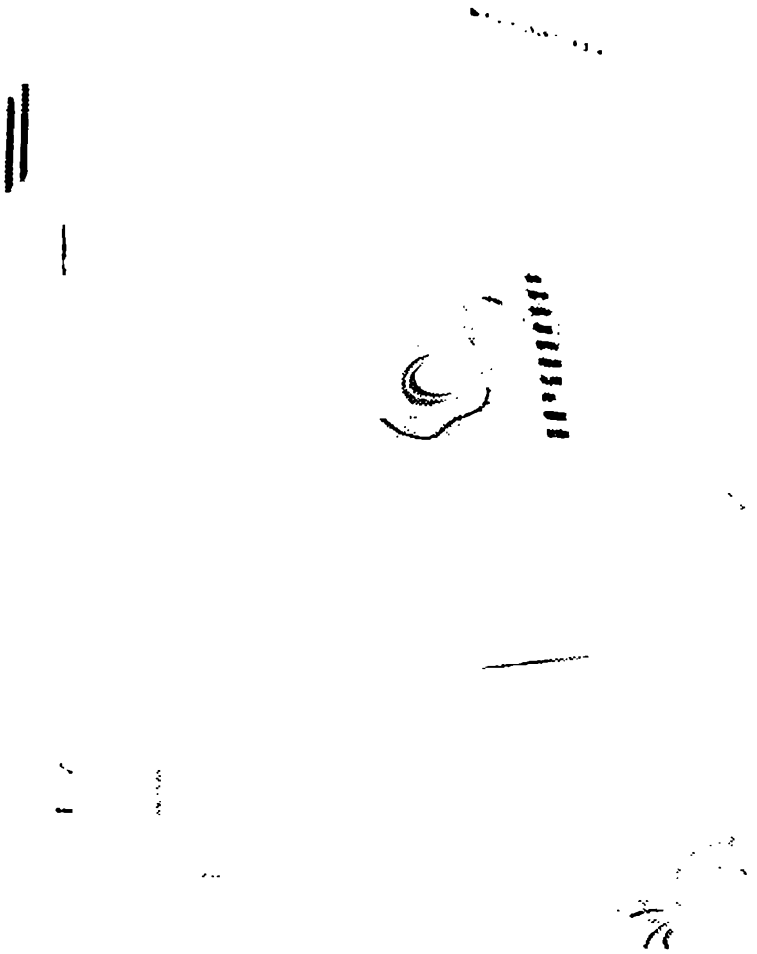


Рис. 1.29. С помощью выбора из меню световое перо использовано для моделирования функции логической кнопки. (С разрешения фирмы Adage.)

ет также, что для растрового устройства световое перо может быть использовано скорее как локатор, чем как устройство указания.

Хотя для всех логических диалоговых устройств существуют соответствующие физические устройства, в конкретной ситуации некоторые из них могут отсутствовать. Таким образом, необходимо уметь моделировать логические диалоговые устройства. На рис. 1.29 показан пример, в котором световое перо используется для моделирования функции логической кнопки путем выбора из меню световых кнопок.

Планшет — одно из самых универсальных физических устройств. Его можно использовать как оцифровыватель для получения координат $x$, $y$. Вдобавок его легко можно использовать для моделирования всех логических диалоговых функций, как это показано на рис. 1.30. Сам планшет является локатором (позиция а на рис 1.30). Используя графический курсор, можно реализовать функцию кнопки. Для этого с помощью карандаша планшета указываем

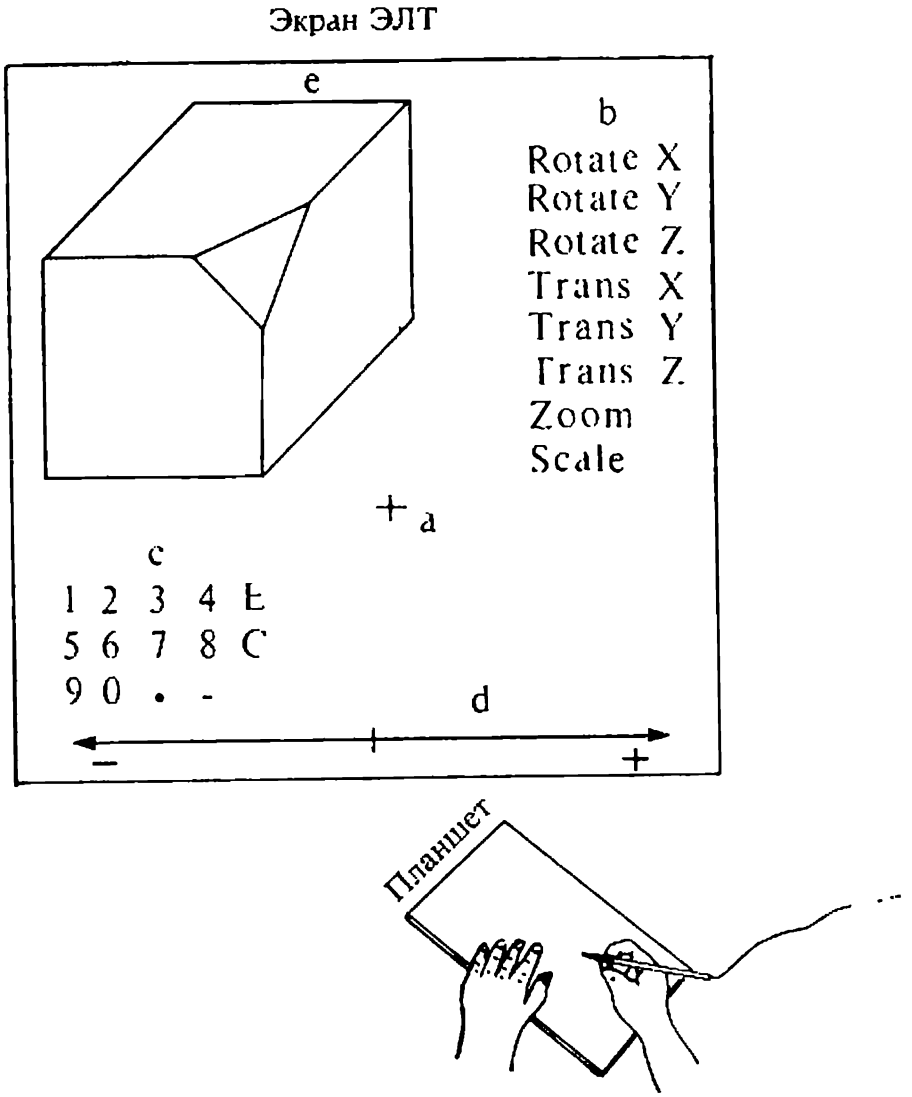



Рис. 1.30. Использование планшета для моделирования всех логических диалоговых функций: (a) локатора, (b) кнопки, (c) клавиатуры, (d) валюатора, (e) селектора.

курсором на кнопку меню (или вблизи нее). Координаты, полученные с планшета, сравниваем с известными координатами $x$, $y$ кнопок меню. Если находится подходящая кнопка, то она активируется (позиция b на рис. 1.30). Аналогичным образом можно реализовать клавиатуру (позиция c на рис. 1.30).

Отдельный валюатор обычно реализуется в сочетании с кнопкой. Конкретная функция для определения значения выбирается кнопкой, как правило, из меню. Затем валюатор можно смоделировать с помощью «числовой оси» (позиция d на рис. 1.30). Перемещение курсора вдоль оси генерирует координаты $x$, $y$, одна из которых интерпретируется как процентное отношение к диапазону валюатора.

Функцию указания можно реализовать с помощью локатора путем определения относительных координат $x$ и $y$ небольшого «окна

указания» Это окно затем становится графическим курсором, и для его позиционирования используется карандаш планшета. Координаты $x$, $y$ каждого из интересуемых отрезков, объектов или подкартинок сравниваются затем с текущим местоположением окна. Если они попадают в пределы окна, то данный элемент считается указанным. Для сложных изображений программная реализация данного алгоритма может оказаться медленной, но при аппаратной реализации заметной задержки не будет. Хотя световое перо нельзя использовать как оцифровыватель, его, подобно планшету, также можно применять для моделирования всех логических диалоговых функций.

## 1.10. РЕЗЮМЕ

В этой главе мы попытались дать обзор основных понятий машинной графики и ее аппаратного обеспечения. Более детальное и практичное представление о предмете можно получить лишь путем сравнения реального аппаратного и программного обеспечения с изложенными концепциями.

## 1.11. ЛИТЕРАТУРА


1-1 Rogers, David F., and Adams, J. Alan. *Mathematical Elements for Computer Graphics*, McGraw-Hill Book Company, New York, 1976.

1-2 Newman, William M., and Sproull, Robert F., *Principles of Interactive Computer Graphics* 2d ed., McGraw-Hill Book Company, New York, 1979.

1-3 Foley, J. D., and Van Dam, A., *Fundamentals of Interactive Computer Graphics*, Addison-Wesley Publishing Company, Reading, Mass., 1982.

1-4 Foley, J. D., and Wallace, V. L., "The Art of Natural Man-Machine Conversation," *Proc. IEEE*, Vol. 62, pp. 462–471, 1974.

1-5 Wallace, V. L., "The Semantics of Graphic Input Devices," *Computer Graphics*, Vol. 10, pp. 61–65, 1976.

1-6 Ohlson, Mark, "System Design Considerations for Graphics Input Devices," *Computer*, pp. 9–18, Nov. 1978.

1-7 Bergeron, R. D., Bono, P. R., and Foley, J. D., "Graphics Programming Using the Core System," *Computing Surveys*, Vol. 10, pp. 389–443, 1978.

1-8 Davis, M. R., and Ellis, T. O., "The RAND Tablet: A Man-Machine Graphical Communication Device," AFIPS Conf. Proc., Vol. 26, Part I, 1964 FJCC, pp. 325–332, 1964.

## ЛИТЕРАТУРА НА РУССКОМ ЯЗЫКЕ


1-1 Роджерс Д. Ф., Адамс Дж. Математические основы машинной графики. — М.: Машиностроение, 1980.

1-2 Ньюмен У., Спрулл Р. Основы интерактивной машинной графики. — М.: Мир, 1976.

1-3 Фоли Дж., вэн Дэм А. Основы интерактивной машинной графики. В 2-х т. — М.: Мир, 1985.

1-4 Фоли Дж., Уоллес В. Искусство организации естественного графического диалога человек — машина, ТИИЭР, 62, 4, 1974.

# 2 Растровая графика

Для работы с устройствами растровой графики нужны специальные методы генерации изображения, вычерчивания прямых и кривых линий, закраски многоугольников, создающей впечатление сплошных объектов. Эти методы рассматриваются в данной главе.

## 2.1. АЛГОРИТМЫ ВЫЧЕРЧИВАНИЯ ОТРЕЗКОВ

Поскольку экран растрового дисплея с электронно-лучевой трубкой (ЭЛТ) можно рассматривать как матрицу дискретных элементов (пикселов), каждый из которых может быть подсвечен, нельзя непосредственно провести отрезок из одной точки в другую. Процесс определения пикселов, наилучшим образом аппроксимирующих заданный отрезок, называется разложением в растр. В сочетании с процессом построчной визуализации изображения он известен как преобразование растровой развертки. Для горизонтальных, вертикальных и наклоненных под углом 45° отрезков выбор растровых элементов очевиден. При любой другой ориентации выбрать нужные пикселы труднее, что показано на рис. 2.1.

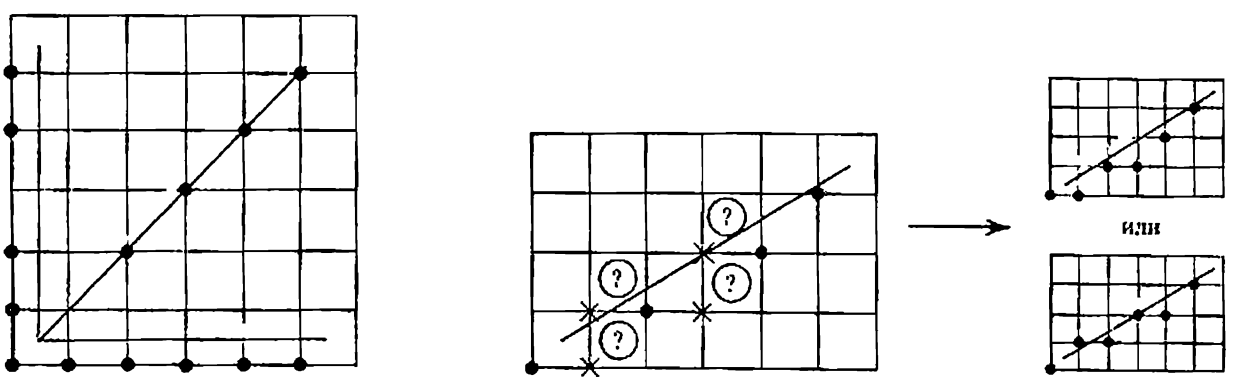



Рис. 2.1. Разложение в растр отрезков прямых.

Прежде чем приступать к обсуждению конкретных алгоритмов рисования отрезков, полезно рассмотреть общие требования к таким алгоритмам и ответить на вопрос, каковы желаемые характеристики изображения. Очевидно, что отрезки должны выглядеть прямыми, начинаться и заканчиваться в заданных точках. Далее, яркость вдоль отрезка должна быть постоянной и не зависеть от длины и наклона. Наконец, рисовать нужно быстро. Как это часто бывает, не все из перечисленных критериев могут быть полностью удовлетворены. Сама природа растрового дисплея исключает генерацию абсолютно прямых линий (кроме ряда специальных случаев), равно как и точное совпадение начала и конца отрезка с заданными точками. Тем не менее при достаточно высоком разрешении дисплея можно получить приемлемую аппроксимацию.

Постоянная вдоль всего отрезка яркость достигается лишь при проведении горизонтальных, вертикальных и наклоненных под углом 45° прямых. Для всех других ориентаций разложение в растр приведет к неравномерной яркости, как это показано на рис. 2.1. Даже для частных случаев яркость зависит от наклона: заметим, например, что расстояние между соседними пикселами для отрезка под углом 45° больше, чем для вертикальных и горизонтальных прямых. Поэтому вертикальные и горизонтальные отрезки будут выглядеть ярче, чем наклонные. Обеспечение одинаковой яркости вдоль отрезков разных длин и ориентаций требует извлечения квадратного корня, а это замедлит вычисления. Обычным компромиссом является нахождение приближенной длины отрезка, сведéние вычислений к минимуму, предпочтительное использование целой арифметики, а также реализация алгоритмов на аппаратном или микропрограммном уровне.

В большинстве алгоритмов вычерчивания отрезков для упрощения вычислений используется пошаговый метод. Приведем пример подобного алгоритма:

### *Простой пошаговый алгоритм*

```
   позиция = начало
   шаг = приращение
1  if позиция – конец < точность then 4
   if позиция > конец then 2
   if позиция < конец then 3
2  позиция = позиция – шаг
   go to 1
```

```
3 позиция = позиция + шаг
  go to 1
4 finish
```

Простой алгоритм разложения отрезка в растр, описанный в следующем разделе, иллюстрирует применение пошаговых методов.

## 2.2. ЦИФРОВОЙ ДИФФЕРЕНЦИАЛЬНЫЙ АНАЛИЗАТОР

Один из методов разложения отрезка в растр состоит в решении дифференциального уравнения, описывающего этот процесс. Для прямой линии имеем

$$\frac{dy}{dx} = \text{const} \quad \text{или} \quad \frac{\Delta y}{\Delta x} = \frac{y_2 - y_1}{x_2 - x_1}$$

Решение представляется в виде

$$\begin{aligned} y_{i+1} &= y_i + \Delta y \\ y_{i+1} &= y_i + \frac{y_2 - y_1}{x_2 - x_1}\Delta x \end{aligned} \qquad (2.1)$$

*где* $x_1, y_1$ и $x_2, y_2$ — концы разлагаемого отрезка и $y_i$ — начальное значение для очередного шага вдоль отрезка. Фактически уравнение (2.1) представляет собой рекуррентное соотношение для последовательных значений $y$ вдоль нужного отрезка. Этот метод, используемый для разложения в растр отрезков, называется цифровым дифференциальным анализатором (ЦДА). В простом ЦДА либо $\Delta x$, либо $\Delta y$ (большее из приращений) выбирается в качестве единицы растра. Ниже приводится простой алгоритм, работающий во всех квадрантах:

***Процедура разложения в растр отрезка по методу цифрового дифференциального анализатора*** (ЦДА)

*предполагается, что концы отрезка* $(x_1, y_1)$ *и* $(x_2, y_2)$ *не совпадают*

**Integer** — *функция преобразования вещественного числа в целое. Примечание: во многих реализациях функция* **Integer** *означает взятие целой части, т. е.* **Integer**$(-8.5) = -9$, *а не* $-8$. *В алгоритме используется именно такая функция.*

**Sign** — *функция, возвращающая* $-1$, $0$, $1$ *для отрицательного, нулевого и положительного аргумента соответственно*

*аппроксимируем длину отрезка*
**if** abs($x_2 - x_1$) $\geqslant$ abs($y_2 - y_1$) **then**
    Длина = abs($x_2 - x_1$)
**else**
    Длина = abs($y_2 - y_1$)
**end if**
*полагаем большее из приращений $\Delta x$ или $\Delta y$ равным единице растра*
  $\Delta x = (x_2 - x_1)$/Длина
  $\Delta y = (y_2 - y_1)$/Длина
*округляем величины, а не отбрасываем дробную часть*
*использование знаковой функции делает алгоритм пригодным для всех квадрантов*
$x = x_1 + 0.5 *$ **Sign**($\Delta x$)
$y = y_1 + 0.5 *$ **Sign**($\Delta y$)
*начало основного цикла*
i = 1
**while** (i $\leqslant$ Длина)
    **Plot** (**Integer**(x), **Integer**(y))
    $x = x + \Delta x$
    $y = y + \Delta y$
    i = i + 1
**end while**
**finish**

Приведем пример, иллюстрирующий работу алгоритма:

**Пример 2.1.** Простой ЦДА в первом квадранте

Рассмотрим отрезок из точки (0, 0) в точку (5, 5). Используем ЦДА для разложения этого отрезка в растр. Результаты работы алгоритма таковы:

начальные установки

$x_1 = 0$
$y_1 = 0$
$x_2 = 5$
$y_2 = 5$

Длина = 5

$\Delta x = 1$
$\Delta y = 1$
$x = 0.5$
$y = 0.5$

результаты пошагового выполнения основного цикла

| i | Plot | x | y |
|---|---|---|---|
| | | 0.5 | 0.5 |
| 1 | (0, 0) | | |
| | | 1.5 | 1.5 |
| 2 | (1, 1) | | |
| | | 2.5 | 2.5 |
| 3 | (2, 2) | | |
| | | 3.5 | 3.5 |
| 4 | (3, 3) | | |
| | | 4.5 | 4.5 |
| 5 | (4, 4) | | |
| | | 5.5 | 5.5 |

Полученное растровое представление отрезка изображено на рис. 2.2. Заметим, что концевые точки определены точно и выбранные пикселы равномерно распределены вдоль отрезка. Внешний вид прямой вполне удовлетворителен. Однако если начальным значением переменной $i$ сделать нуль вместо единицы, то окажется активированным пиксел с координатами (5, 5), что нежелательно. Если адрес пиксела задан целыми координатами левого нижнего угла, то активация этого пиксела даст явно неверную конечную точку отрезка (рис. 2.2). Вдобавок при вычерчивании серии последовательных отрезков пиксел (5, 5) будет активирован дважды: в конце данного отрезка и в начале следующего. Такой пиксел может выглядеть как более яркий или иметь иной (быть может, неестественный) цвет. Следующий пример иллюстрирует работу алгоритма в третьем квадранте.

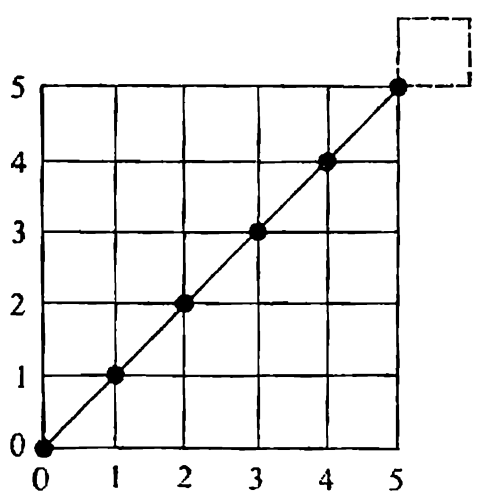



Рис. 2.2. Результаты работы простого ЦДА в первом квадранте.

**Пример 2.2.** Простой ЦДА в третьем квадранте

Рассмотрим отрезок из точки (0, 0) в точку (−8, −4) в третьем квадранте. Результаты работы алгоритма таковы:

начальные установки

$x_1 = 0$
$y_1 = 0$
$x_2 = -8$
$y_2 = -4$

Длина = 8

$\Delta x = -1$
$\Delta y = -0.5$
$x = -0.5$
$y = -0.5$

результаты пошагового выполнения основного цикла в предположении, что используется функция округления, таковы:

| i | Plot | x | y |
|---|---|---|---|
| | | −0.5 | −0.5 |
| 1 | (−1, −1) | | |
| | | −1.5 | −1.0 |
| 2 | (−2, −1) | | |
| | | −2.5 | −1.5 |
| 3 | (−3, −2) | | |
| | | −3.5 | −2.0 |
| 4 | (−4, −2) | | |
| | | −4.5 | −2.5 |
| 5 | (−5, −3) | | |
| | | −5.5 | −3.0 |
| 6 | (−6, −3) | | |
| | | −6.5 | −3.5 |
| 7 | (−7, −4) | | |
| | | −7.5 | −4.0 |
| 8 | (−8, −4) | | |
| | | −8.5 | −4.5 |

Несмотря на то что результаты, представленные на рис. 2.3, выглядят вполне приемлемыми, анализ отрезков, проведенных из точки (0, 0) в точку (−8, 4) и (8, −4), показывает, что разложенный в растр отрезок лежит по одну сторону от реального и что на одном из концов отрезка появляется лишняя точка, т. е. результат работы алгоритма зависит от ориентации. Следовательно, точность в концевых точках ухудшается. Далее, если вместо взятия целой части использовать округление до ближайшего целого, то ре-

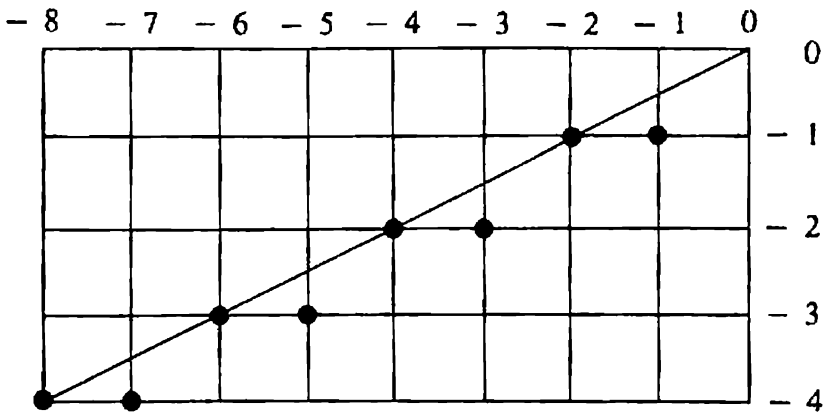



Рис. 2.3. Результаты работы простого ЦДА в третьем квадранте.

зультаты снова получатся разными. Таким образом, либо нужно использовать более сложный и более медленный алгоритм, либо надо отступиться от требования максимально точной аппроксимации. Вдобавок предложенный алгоритм имеет тот недостаток, что он использует вещественную арифметику. В следующем разделе описан более приемлемый алгоритм.

## 2.3. АЛГОРИТМ БРЕЗЕНХЕМА

Хотя алгоритм Брезенхема [2-1] был первоначально разработан для цифровых графопостроителей, однако он в равной степени подходит и для использования растровыми устройствами с ЭЛТ. Алгоритм выбирает оптимальные растровые координаты для представления отрезка. В процессе работы одна из координат — либо $x$, либо $y$ (в зависимости от углового коэффициента) — изменяется на единицу. Изменение другой координаты (либо на нуль, либо на единицу) зависит от расстояния между действительным положением отрезка и ближайшими координатами сетки. Такое расстояние мы назовем ошибкой.

Алгоритм построен так, что требуется проверять лишь знак этой ошибки. На рис. 2.4 это иллюстрируется для отрезка в первом октанте, т. е. для отрезка с угловым коэффициентом, лежащим в диапазоне от нуля до единицы. Из рисунка можно заметить, что если угловой коэффициент отрезка из точки (0, 0) больше чем 1/2, то его пересечение с прямой $x = 1$ будет расположено ближе к прямой $y = 1$, чем к прямой $y = 0$. Следовательно, точка растра (1, 1) лучше аппроксимирует ход отрезка, чем точка (1, 0). Если угловой коэффициент меньше 1/2, то верно обратное. Для углового коэффи-

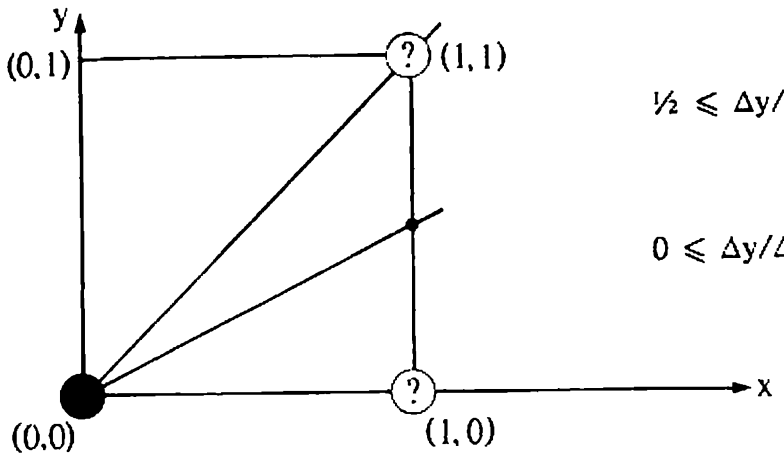



Рис. 2.4. Основная идея алгоритма Брезенхема.

циента, равного 1/2, нет какого-либо предпочтительного выбора. В данном случае алгоритм выбирает точку (1, 1).

Не все отрезки проходят через точки растра. Подобная ситуация иллюстрируется рис. 2.5, где отрезок с тангенсом угла наклона 3/8 сначала проходит через точку растра (0, 0) и последовательно пересекает три пиксела. Также иллюстрируется вычисление ошибки при

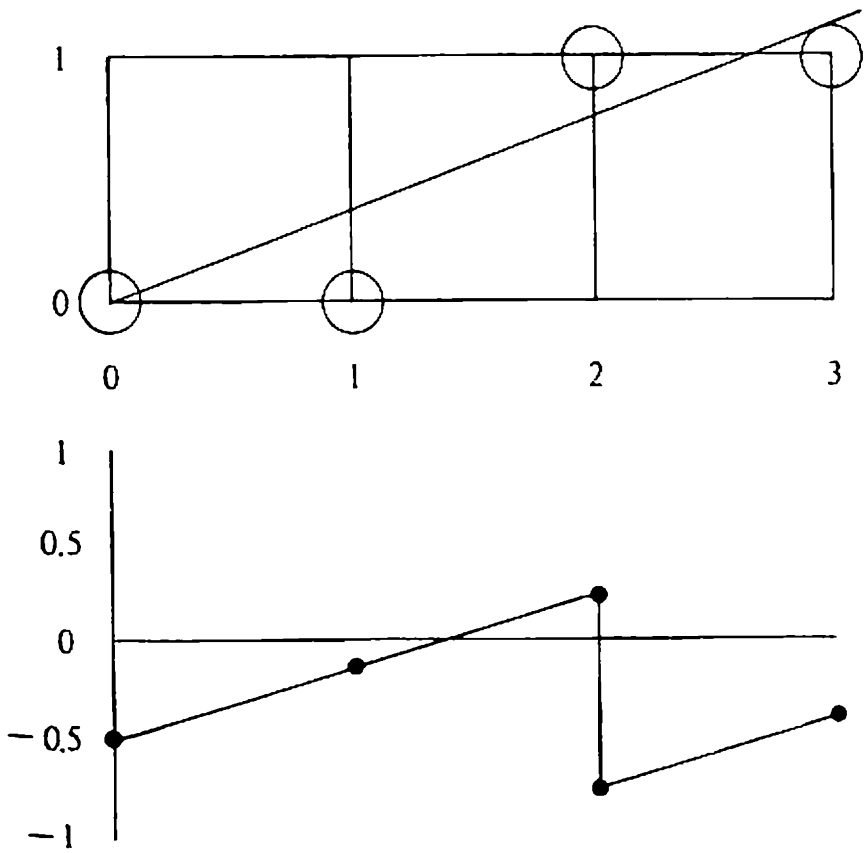



Рис. 2.5. График ошибки в алгоритме Брезенхема.

представлении отрезка дискретными пикселами. Так как желательно проверять только знак ошибки, то она первоначально устанавливается равной $-1/2$. Таким образом, если угловой коэффициент отрезка больше или равен $1/2$, то величина ошибки в следующей точке растра с координатами (1, 0) может быть вычислена как

$$e = e + m$$

где $m$ — угловой коэффициент. В нашем случае при начальном значении ошибки $-1/2$

$$e = -1/2 + 3/8 = -1/8$$

Так как $e$ отрицательно, отрезок пройдет ниже середины пиксела. Следовательно, пиксел на том же самом горизонтальном уровне лучше аппроксимирует положение отрезка, поэтому $y$ не увеличивается. Аналогично вычисляем ошибку

$$e = -1/8 + 3/8 = 1/4$$

в следующей точке растра (2, 0). Теперь $e$ положительно, а значит, отрезок пройдет выше средней точки. Растровый элемент (2, 1) со следующей по величине координатой $y$ лучше аппроксимирует положение отрезка. Следовательно, $y$ увеличивается на единицу. Прежде чем рассматривать следующий пиксел, необходимо откорректировать ошибку вычитанием из нее единицы. Имеем

$$e = 1/4 - 1 = -3/4$$

Заметим, что пересечение вертикальной прямой $x = 2$ с заданным отрезком лежит на $1/4$ ниже прямой $y = 1$. Если же перенести отрезок $1/2$ вниз, мы получим как раз величину $-3/4$. Продолжение вычислений для следующего пиксела дает

$$e = -3/4 + 3/8 = -3/8$$

Так как $e$ отрицательно, то $y$ не увеличивается. Из всего сказанного следует, что ошибка — это интервал, отсекаемый по оси $y$ рассматриваемым отрезком в каждом растровом элементе (относительно $-1/2$).

Приведем алгоритм Брезенхема для первого октанта, т. е. для случая $0 \leqslant \Delta y \leqslant \Delta x$.

### *Алгоритм Брезенхема разложения в растр отрезка для первого октанта*

*предполагается, что концы отрезка* $(x_1, y_1)$ *и* $(x_2, y_2)$ *не совпадают*

**Integer** — *функция преобразования в целое*
x, y, Δx, Δy — *целые*
e — вещественное

*инициализация переменных*
$x = x_1$
$y = y_1$
$\Delta x = x_2 - x_1$
$\Delta y = y_2 - y_1$
*инициализация* e *с поправкой на половину пиксела*
$e = \Delta y/\Delta x - 1/2$
*начало основного цикла*
**for** i = 1 **to** Δx
  **Plot (x, y)**
  **while** $(e \geqslant 0)$
    $y = y + 1$
    $e = e - 1$
  **end while**
  $x = x + 1$
  $e = e + \Delta y/\Delta x$
**next** i
**finish**

Блок-схема алгоритма приводится на рис. 2.6. Пример дан ниже.

**Пример 2.3.** Алгоритм Брезенхема

Рассмотрим отрезок, проведенный из точки (0, 0) в точку (5, 5). Разложение отрезка в растр по алгоритму Брезенхема приводит к такому результату:

начальные установки

$x = 0$
$y = 0$
$\Delta x = 5$
$\Delta y = 5$
$e = 1 - 1/2 = 1/2$

Результаты пошагового выполнения основного цикла

| i | Plot | e | x | y |
|---|---|---|---|---|
| | | 1/2 | 0 | 0 |
| 1 | (0, 0) | | | |
| | | −1/2 | 0 | 1 |
| | | 1/2 | 1 | 1 |
| 2 | (1, 1) | | | |
| | | −1/2 | 1 | 2 |
| | | 1/2 | 2 | 2 |

| | | | | |
|---|---|---|---|---|
| 3 | (2, 2) | | | |
| | | −1/2 | 2 | 3 |
| | | 1/2 | 3 | 3 |
| 4 | (3, 3) | | | |
| | | −1/2 | 3 | 4 |
| | | 1/2 | 4 | 4 |
| 5 | (4. 4) | | | |
| | | −1/2 | 4 | 5 |
| | | 1/2 | 5 | 5 |

Результат показан на рис. 2.7 и совпадает с ожидаемым. Заметим, что точка растра с координатами (5, 5) не активирована. Эту точку можно активировать путем изменения условия цикла **for-next**

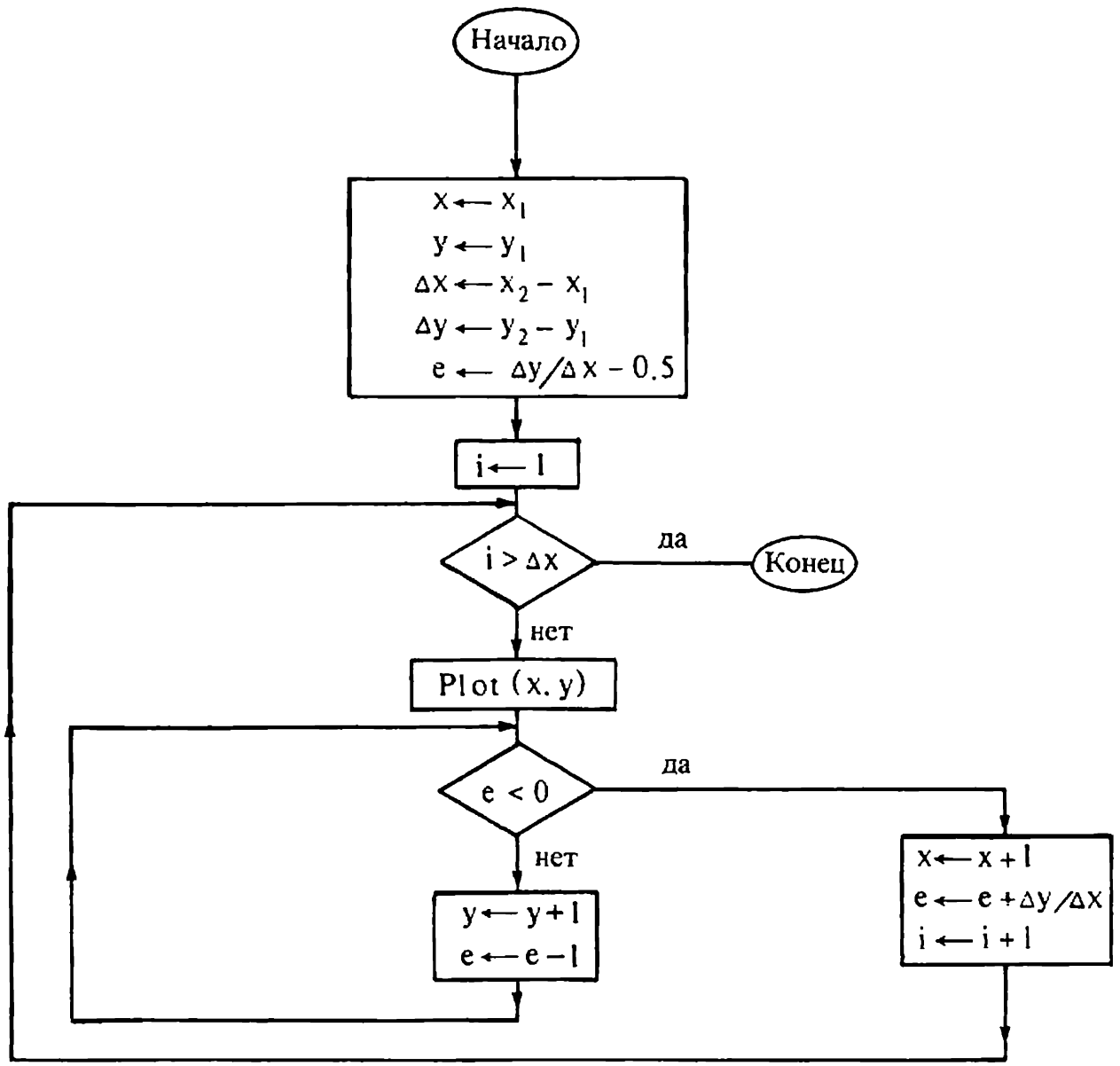



Рис. 2.6. Блок-схема алгоритма Брезенхема.

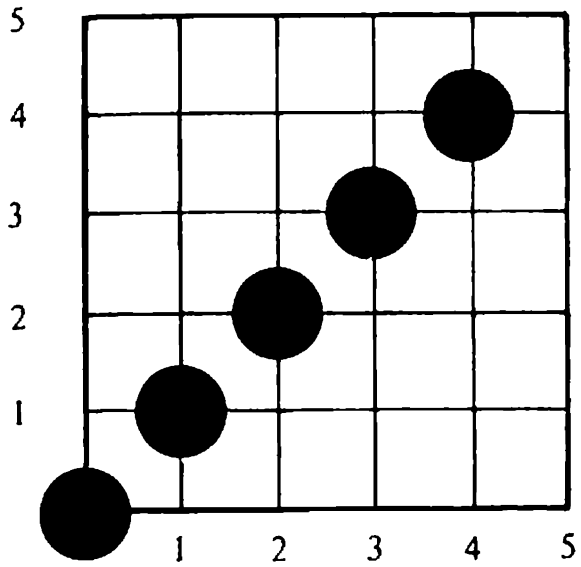



Рис. 2.7. Результат работы алгоритма Брезенхема в первом октанте.

на 0 **to** $\Delta x$. Активацию точки (0, 0) можно устранить, если поставить оператора **Plot** непосредственно перед строкой **next** i.

## 2.4. ЦЕЛОЧИСЛЕННЫЙ АЛГОРИТМ БРЕЗЕНХЕМА

Алгоритм Брезенхема в том виде, как он представлен выше, требует использования арифметики с плавающей точкой и деления (для вычисления углового коэффициента и оценки ошибки). Быстродействие алгоритма можно увеличить, если использовать только целочисленную арифметику и исключить деление. Так как важен лишь знак ошибки, то простое преобразование

$$\bar{e} = 2e\,\Delta x$$

превратит предыдущий алгорим в целочисленный и позволит эффективно реализовать его на аппаратном или микропрограммном уровне. Модифицированный целочисленный алгоритм для первого октанта, т. е. для $0 \leqslant \Delta y \leqslant \Delta x$, таков:

***Целочисленный алгоритм Брезенхема для первого октанта***

*предполагается, что концы отрезка* $(x_1, y_1)$ *и* $(x_2, y_2)$ *не совпадают и все переменные — целые*

$x = x_1$
$y = y_1$
$\Delta x = y_2 - x_1$
$\Delta y = y_2 - x_1$
*инициализируем* $\bar{e}$ *с поправкой на половину пиксела*
$\bar{e} = 2 * \Delta y - \Delta x$

```
начало основного цикла
for i = 1 to Δx
    Plot (x, y)
    while (e ≥ 0)
        y = y + 1
        e = e − 2 * Δx
    end while
    x = x + 1
    e = e + 2 * Δy
next i
finish
```

Блок-схема, помещенная на рис. 2.6, применима и в данном случае с соответствующими изменениями в вычислении ошибки.

## 2.5. ОБЩИЙ АЛГОРИТМ БРЕЗЕНХЕМА

Чтобы реализация алгоритма Брезенхема была полной, необходимо обрабатывать отрезки во всех октантах. Модификацию легко сделать, учитывая в алгоритме номер квадранта, в котором лежит отрезок и его угловой коэффициент. Когда абсолютная величина углового коэффициента больше 1, $y$ постоянно изменяется на единицу, а критерий ошибки Брезенхема используется для принятия решения об изменении величины $x$. Выбор постоянно изменяющейся (на $+1$ или $-1$) координаты зависит от квадранта (рис. 2.8). Общий алгоритм может быть оформлен в следующем виде:

***Обобщенный целочисленный алгоритм Брезенхема квадрантов***

*предполагается, что концы отрезка* $(x_1, y_1)$ *и* $(x_2, y_2)$ *не совпадают все переменные считаются целыми*
*функция* **Sign** *возвращает* $-1$, 0, 1 *для отрицательного, нулевого и положительного аргумента соответственно*

```
инициализация переменных
x = x1
y = y1
Δx = abs (x2 − x1)
Δy = abs (y2 − y1)
s1 = Sign (x2 − x1)
s2 = Sign (y2 − y1)
обмен значений Δx и Δy в зависимости от углового коэффици-
```

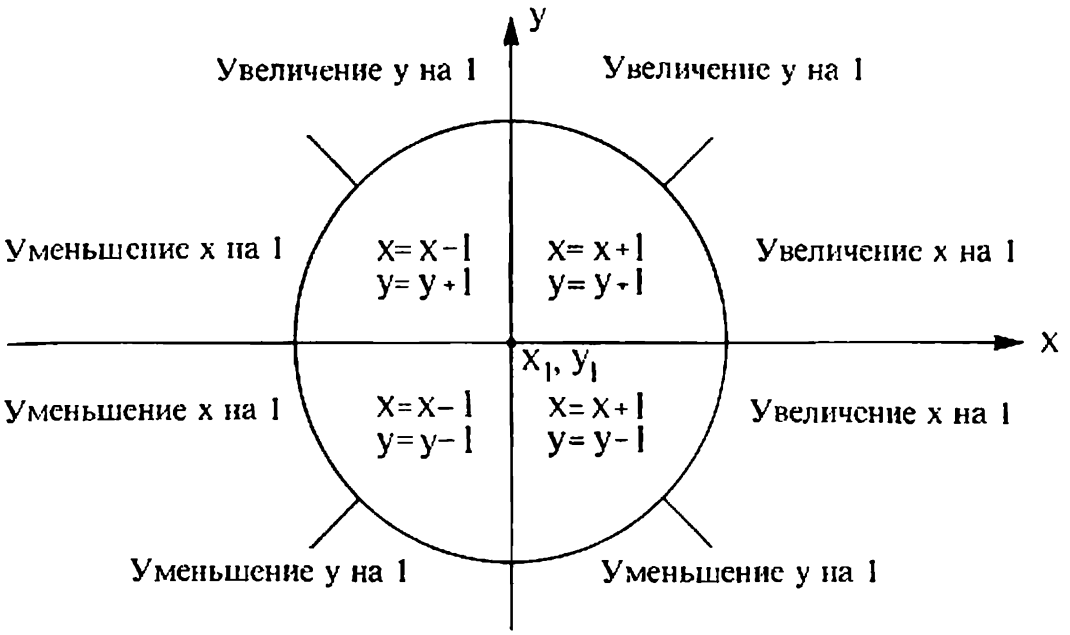



Рис. 2.8. Разбор случаев для обобщенного алгоритма Брезенхема.

```
ента наклона отрезка
  if  Δy > Δx then
      Врем = Δx
      Δx = Δy
      Δy = Врем
      Обмен = 1
  else
      Обмен = 0
  end if
  инициализация e с поправкой на половину пиксела
  e = 2 * Δy − Δx
  основной цикл
  for i = 1 to Δx
      Plot(x, y)
      while (e ⩾ 0)
        if Обмен = 1 then
          x = x + s1
        else
          y = y + s2
        end if
        e = e − 2 * Δx
      end while
      if Обмен = 1 then
        y = y + s2
      else
        x = x + s1
```

```
    end if
    ē = ē + 2 * Δy
next i
finish
```

**Пример 2.4.** Обобщенный алгоритм Брезенхема

ля иллюстрации общего алгоритма Брезенхема рассмотрим отрезок из точки (0, 0) точку (−8, −4). В примере 2.2 этот отрезок был обработан с помощью простого ЦДА:

начальные установки

$x = 0$
$y = 0$
$\Delta x = 8$
$\Delta y = 4$
$s_1 = -1$
$s_2 = -1$
Обмен = 0
$e = 0$

ошаговое выполнение основного цикла

| | Plot | c | x | y |
|---|---|---|---|---|
| | | 0 | 0 | 0 |
| | (0, 0) | | | |
| | | −16 | 0 | −1 |
| | | −8 | −1 | −1 |
| | (−1, −1) | | | |
| | | 0 | −2 | −1 |
| | (−2, −1) | | | |
| | | −16 | −2 | −2 |
| | | −8 | −3 | −2 |
| | (−3, −2) | | | |
| | | 0 | −4 | −2 |
| 6 | (−4, −2) | | | |
| | | −16 | −4 | −3 |
| | | −8 | −5 | −3 |
| | (−5, −3) | | | |
| | | 0 | −6 | −3 |
| 7 | (−6, −3) | | | |
| | | −16 | −6 | −4 |
| | | −8 | −7 | −4 |
| 8 | (−7, −4) | | | |
| | | 0 | −8 | −4 |

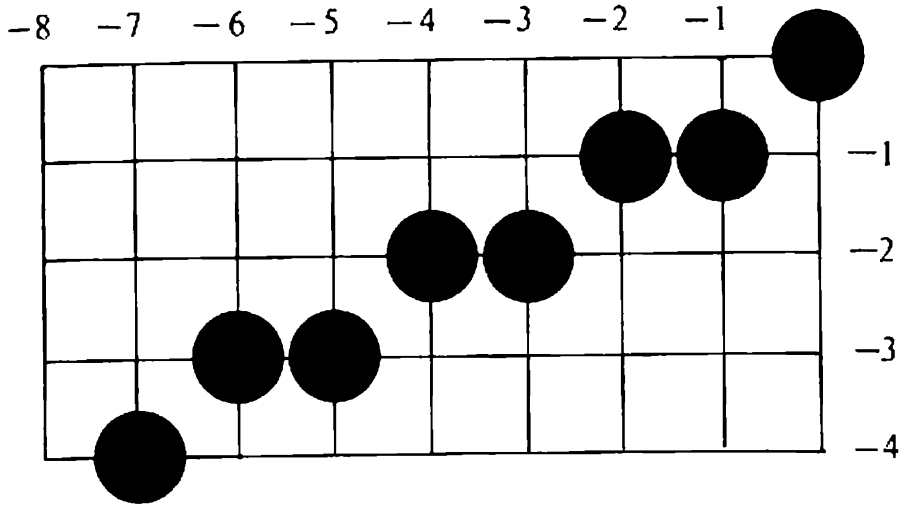



Рис. 2.9. Результат работы обобщенного алгоритма Брезенхема в третьем квадранте.

На рис. 2.9 продемонстрирован результат. Сравнение с рис. 2.3 показывает, что результаты работы двух алгоритмов отличаются.

## 2.6. АЛГОРИТМ БРЕЗЕНХЕМА ДЛЯ ГЕНЕРАЦИИ ОКРУЖНОСТИ

В растр нужно разлагать не только линейные, но и другие, более сложные функции. Разложению конических сечений, т. е. окружностей, эллипсов, парабол, гипербол, было посвящено значительное число работ [2-2 — 2-5]. Наибольшее внимание, разумеется, уделено окружности [2-6 — 2-9]. Один из наиболее эффективных и простых для понимания алгоритмов генерации окружности принадлежит Брезенхему [2-10]. Для начала заметим, что необходимо сгенерировать только одну восьмую часть окружности. Остальные ее части могут быть получены последовательными отражениями, как это показано на рис. 2.10. Если сгенерирован первый октант (от 0 до 45° против часовой стрелки), то второй октант можно получить зеркальным отражением относительно прямой $y = x$, что дает в совокупности первый квадрант. Первый квадрант отражается относительно прямой $x = 0$ для получения соответствующей части окружности во втором квадранте. Верхняя полуокружность отражается относительно прямой $y = 0$ для завершения построения. На рис. 2.10 приведены двумерные матрицы соответствуюших преобразований.

Для вывода алгоритма рассмотрим первую четверть окружности с центром в начале координат. Заметим, что если работа алгоритма начинается в точке $x = 0$, $y = R$, то при генерации окружности по часовой стрелке в первом квадранте $y$ является монотон-

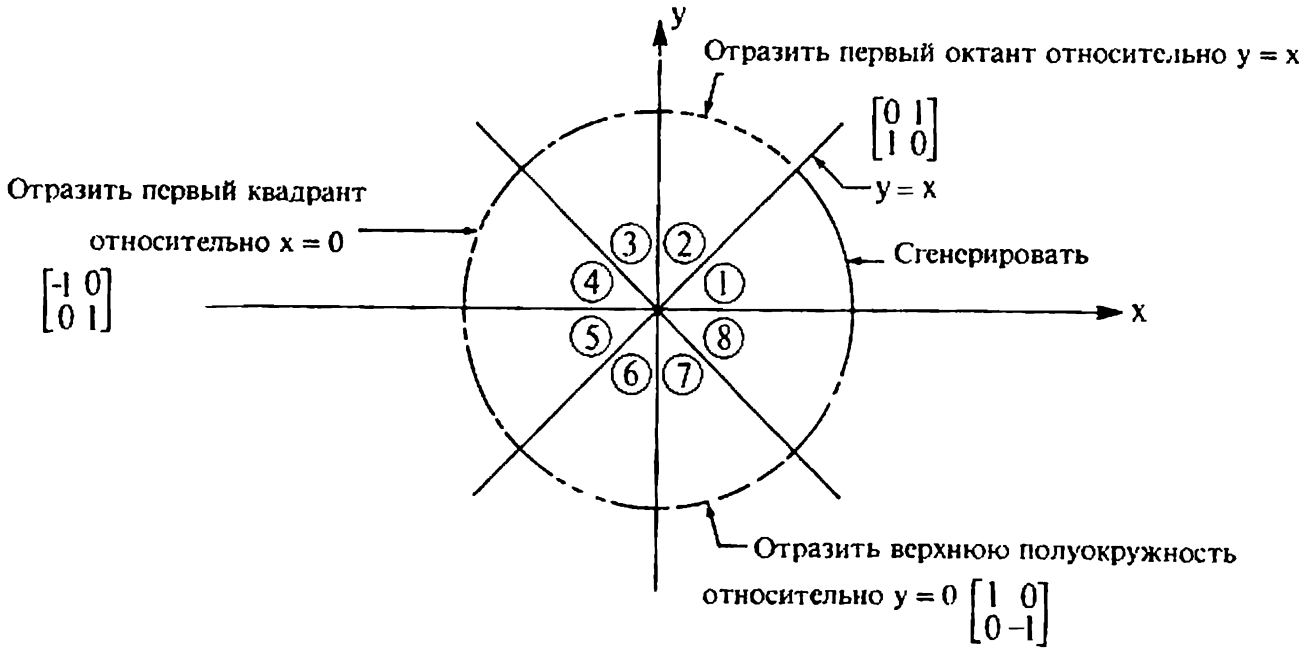



Рис. 2.10. Генерация полной окружности из дуги в первом октанте.

но убывающей функцией аргумента $x$ (рис. 2.11). Аналогично, если исходной точкой является $y = 0, x = R$, то при генерации окружности против часовой стрелки $x$ будет монотонно убывающей функцией аргумента $y$. В нашем случае выбирается генерация по часовой стрелке с началом в точке $x = 0, y = R$. Предполагается, что центр окружности и начальная точка находятся точно в точках растра.

Для любой заданной точки на окружности при генерации по часовой стрелке существует только три возможности выбрать следующий пиксел, наилучшим образом приближающий окружность: горизонтально вправо, по диагонали вниз и вправо, вертикально вниз. На рис. 2.12 эти направления обозначены соответственно $m_H$, $m_D$, $m_V$. Алгоритм выбирает пиксел, для которого минимален квадрат расстояния между одним из этих пикселов и окружностью, т. е. минимум из[1)]

$$m_H = |(x_i + 1)^2 + (y_i)^2 - R^2|$$
$$m_D = |(x_i + 1)^2 + (y_i - 1)^2 - R^2|$$
$$m_V = |(x_i)^2 + (y_i - 1)^2 - R^2|$$

Вычисления можно упростить, если заметить, что в окрестности точки $(x_i, y_i)$ возможны только пять типов пересечений окружности и сетки растра, приведенных на рис. 2.13.

---

[1)] Здесь минимизируется не квадрат расстояния, а абсолютное значение разности квадратов расстояний от центра окружности до пиксела и до окружности. — *Прим. перев.*

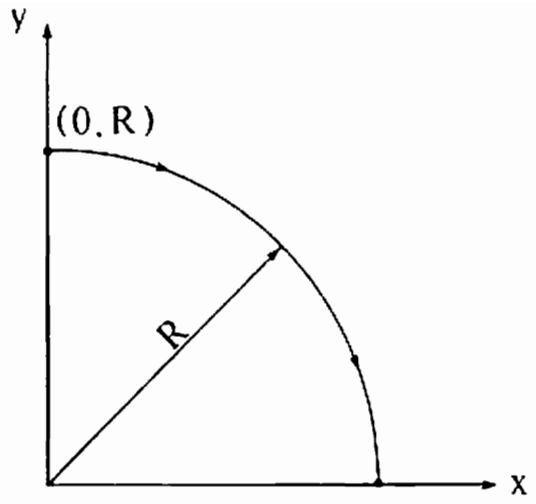



Рис. 2.11. Окружность в первом квадранте.

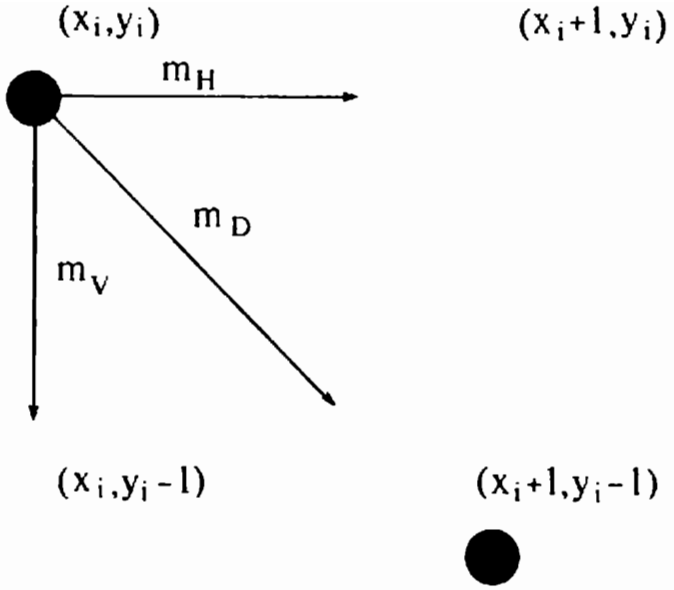



Рис. 2.12. Выбор пикселов в первом квадранте.

Разность между квадратами расстояний от центра окружности до диагонального пиксела $(x_i + 1, y_i - 1)$ и от центра до точки на окружности $R^2$ равна

$$\Delta_i = (x_i + 1)^2 + (y_i - 1)^2 - R^2$$

Как и в алгоритме Брезенхема для отрезка, для выбора соответствующего пиксела желательно использовать только знак ошибки, а не ее величину.

При $\Delta_i < 0$ диагональная точка $(x_i + 1, y_i - 1)$ находится внутри реальной окружности, т. е. это случаи 1 или 2 на рис. 2.13. Ясно, что в этой ситуации следует выбрать либо пиксел $(x_i + 1, y_i)$, т. е. $m_H$, либо пиксел $(x_i + 1, y_i - 1)$, т. е. $m_D$. Для этого сначала рассмотрим случай 1 и проверим разность квадратов расстояний от

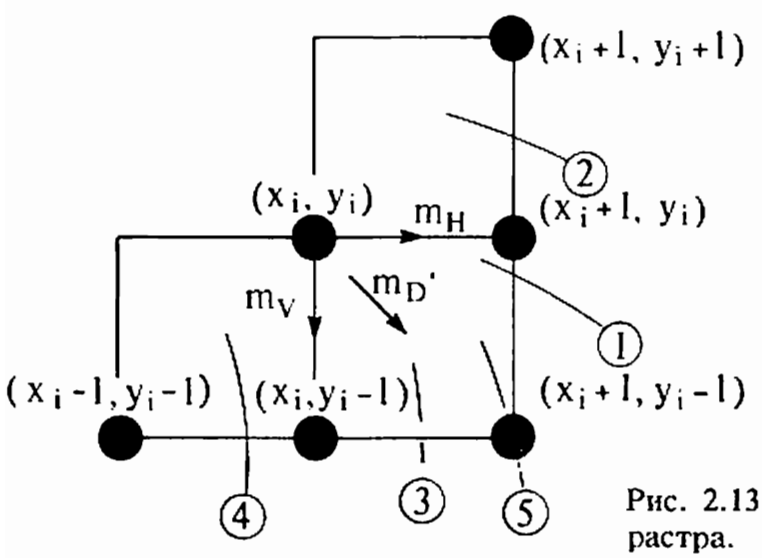



Рис. 2.13. Пересечение окружности и сетки растра.

окружности до пикселов в горизонтальном и диагональном направлениях:

$$\delta = |(x_i + 1)^2 + (y_i)^2 - R^2| - |(x_i + 1)^2 + (y_i - 1)^2 - R^2|$$

При $\delta < 0$ расстояние от окружности до диагонального пиксела ($m_D$) больше, чем до горизонтального ($m_H$). Напротив, если $\delta > 0$, расстояние до горизонтального пиксела ($m_H$) больше. Таким образом,

при $\delta \leqslant 0$ выбираем $m_H$ в $(x_i + 1, y_i)$
при $\delta > 0$ выбираем $m_D$ в $(x_i + 1, y_i - 1)$

При $\delta = 0$, когда расстояние от окружности до обоих пикселов одинаковы, выбираем горизонтальный шаг.

Количество вычислений, необходимых для оценки величины $\delta$, можно сократить, если заметить, что в случае 1

$$(x_i + 1)^2 + (y_i)^2 - R^2 \geq 0$$
$$(x_i + 1)^2 + (y_i - 1)^2 - R^2 < 0$$

так как диагональный пиксел $(x_i + 1, y_i - 1)$ всегда лежит внутри окружности, а горизонтальный $(x_i + 1, y_i)$ — вне ее. Таким образом, $\delta$ можно вычислить по формуле

$$\delta = (x_i + 1)^2 + (y_i)^2 - R^2 + (x_i + 1)^2 + (y_i - 1)^2 - R^2$$

Дополнение до полного квадрата члена $(y_i)^2$ с помощью добавления и вычитания $-2y_i + 1$ дает

$$\delta = 2\left[(x_i + 1)^2 + (y_i - 1)^2 - R^2\right] + 2y_i - 1$$

В квадратных скобках стоит по определению $\Delta_i$ и его подстановка

$$\delta = 2(\Delta_i + y_i) - 1$$

существенно упрощает выражение.

Рассмотрим случай 2 на рис. 2.13 и заметим, что здесь должен быть выбран горизонтальный пиксел $(x_i + 1, y_i)$, так как $y$ является монотонно убывающей функцией. Проверка компонент $\delta$ показывает, что

$$(x_i + 1) + (y_i)^2 - R^2 < 0$$
$$(x_i + 1) + (y_i - 1)^2 - R^2 < 0$$

поскольку в случае 2 горизонтальный $(x_i + 1, y_i)$ и диагональный $(x_i + 1, y_i - 1)$ пикселы лежат внутри окружности. Следовательно,

$\delta < 0$, и при использовании того же самого критерия, что и в случае 1, выбирается пиксел $(x_i + 1, y_i)$.

Если $\Delta_i > 0$, то диагональная точка $(x_i + 1, y_i - 1)$ находится вне окружности, т. е. это случаи 3 и 4 на рис. 2.13. В данной ситуации ясно, что должен быть выбран либо пиксел $(x_i + 1, y_i - 1)$, т. е. $m_D$, либо $(x_i, y_i - 1)$, т. е. $m_V$. Аналогично разбору предыдущего случая критерий выбора можно получить, рассматривая сначала случай 3 и проверяя разность между квадратами расстояний от окружности до диагонального $m_D$ и вертикального $m_V$ пикселов, т. е.

$$\delta' = |(x_i + 1)^2 + (y_i - 1)^2 - R^2| - |(x_i)^2 + (y_i - 1)^2 - R^2|$$

При $\delta' < 0$ расстояние от окружности до вертикального пиксела $(x_i, y_i - 1)$ больше и следует выбрать диагональный шаг $m_D$, к пикселу $(x_i + 1, y_i - 1)$. Напротив, в случае $\delta' > 0$ расстояние от окружности до диагонального пиксела больше и следует выбрать вертикальное движение к пикселу $(x_i, y_i - 1)$. Таким образом,

при $\delta' \leqslant 0$ выбираем $m_D$ в $(x_i + 1, y_i - 1)$
при $\delta' > 0$ выбираем $m_V$ в $(x_i, y_i - 1)$

Здесь в случае $\delta' = 0$, т. е. когда расстояния равны, выбран диагональный шаг.

Проверка компонент $\delta'$ показывает, что

$$(x_i + 1)^2 + (y_i - 1)^2 - R^2 \geq 0$$
$$(x_i)^2 + (y_i - 1)^2 - R^2 < 0$$

поскольку для случая 3 диагональный пиксел $(x_i + 1, y_i - 1)$ находится вне окружности, тогда как вертикальный пиксел $(x_i, y_i - 1)$ лежит внутри ее. Это позволяет записать $\delta'$ в виде

$$\delta' = (x_i + 1)^2 + (y_i - 1)^2 - R^2 + (x_i)^2 + (y_i - 1)^2 - R^2$$

Дополнение до полного квадрата члена $(x_i)^2$ с помощью добавления и вычитания $2x_i + 1$ дает

$$\delta' = 2\left[(x_i + 1)^2 + (y_i - 1)^2 - R^2\right] - 2x_i - 1$$

Использование определения $\Delta_i$ приводит выражение к виду

$$\delta' = 2(\Delta_i - x_i) - 1$$

Теперь, рассматривая случай 4, снова заметим, что следует выбрать вертикальный пиксел $(x_i, y_i - 1)$, так как $y$ является моно-

тонно убывающей функцией при возрастании $x$.

Проверка компонент $\delta'$ для случая 4 показывает, что

$$(x_i + 1)^2 + (y_i - 1)^2 - R^2 > 0$$
$$(x_i)^2 + (y_i - 1)^2 - R^2 > 0$$

поскольку оба пиксела находятся вне окружности. Следовательно, $\delta' > 0$ и при использовании критерия, разработанного для случая 3, происходит верный выбор $m_V$.

Осталось проверить только случай 5 на рис. 2.13, который встречается, когда диагональный пиксел $(x_i + 1, y_i - 1)$ лежит на окружности, т. е. $\Delta_i = 0$. Проверка компонент $\delta$ показывает, что

$$(x_i + 1)^2 + (y_i)^2 - R^2 > 0$$
$$(x_i + 1)^2 + (y_i - 1)^2 - R^2 = 0$$

Следовательно, $\delta > 0$ и выбирается диагональный пиксел $(x_i + 1, y_i - 1)$. Аналогичным образом оцениваем компоненты $\delta'$ :

$$(x_i + 1)^2 + (y_i - 1)^2 - R^2 = 0$$
$$(x_i)^2 + (y_i - 1)^2 - R^2 < 0$$

и $\delta' < 0$, что является условием выбора правильного диагонального шага к $(x_i + 1, y_i - 1)$. Таким образом, случай $\Delta_i = 0$ подчиняется тому же критерию, что и случай $\Delta_i < 0$ или $\Delta_i > 0$.

Подведем итог полученных результатов:

$\Delta_i < 0$

$\delta \leqslant 0$ выбираем пиксел $(x_i + 1, y_i) \rightarrow m_H$

$\delta > 0$ выбираем пиксел $(x_i + 1, y_i - 1) \rightarrow m_D$

$\Delta_i > 0$

$\delta' \leqslant 0$ выбираем пиксел $(x_i + 1, y_i - 1) \rightarrow m_D$

$\delta' > 0$ выбираем пиксел $(x_i, y_i - 1) \rightarrow m_V$

$\Delta_i = 0$ выбираем пиксел $(x_i + 1, y_i - 1) \rightarrow m_D$

Легко разработать простые рекуррентные соотношения для реализации пошагового алгоритма. Сначала рассмотрим горизонтальный шаг $m_H$ к пикселу $(x_i + 1, y_i)$. Обозначим это новое положение пиксела как $(i + 1)$. Тогда координаты нового пиксела и значение $\Delta_i$ равны

$$x_{i+1} = x_i + 1$$
$$y_{i+1} = y_i$$
$$\Delta_{i+1} = (x_{i+1} + 1)^2 + (y_{i+1} - 1)^2 - R^2$$

$$= (x_{i+1})^2 + 2x_{i+1} + 1 + (y_i - 1)^2 - R^2$$
$$= (x_i + 1)^2 + (y_i - 1)^2 - R^2 + 2x_{i+1} + 1$$
$$= \Delta_i + 2x_{i+1} + 1$$

Аналогично координаты нового пиксела и значение $\Delta_i$ для шага $m_D$ к пикселу $(x_i + 1, y_i - 1)$ таковы:

$$x_{i+1} = x_i + 1$$
$$y_{i+1} = y_i - 1$$
$$\Delta_{i+1} = \Delta_i + 2x_{i+1} - 2y_{i+1} + 2$$

То же самое для шага $m_V$ к $(x_i, y_i - 1)$

$$x_{i+1} = x_i$$
$$y_{i+1} = y_i - 1$$
$$\Delta_{i+1} = \Delta_i - 2y_{i+1} + 1$$

Реализация алгоритма Брезенхема на псевдокоде для окружности приводится ниже.

***Пошаговый алгоритм Брезенхема для генерации окружности в первом квадранте***

*все переменные — целые*

```
        инициализация переменных
        x_i = 0
        y_i = R
        Δ_i = 2(1 − R)
        Предел = 0
1       Plot(x_i, y_i)
        if y_i ⩽ Предел then 4
        Выделение случая 1 или 2, 4 или 5, или 3
        if Δ_i < 0 then 2
        if Δ_i > 0 then 3
        if Δ_i = 0 then 20
        определение случая 1 или 2
2       δ = 2Δ_i + 2y_i − 1
        if δ ⩽ 0 then 10
        if δ > 0 then 20
        определение случая 4 или 5
3       δ' = 2Δ_i + 2x_i − 1
        if δ' ⩽ 0 then 20
        If δ' > 0 then 30
```

```
        выполнение шагов
        шаг m_H
10      x_i = x_i + 1
        Δ_i = Δ_i + 2x_i + 1
        go to 1
        шаг m_D
20      x_i = x_i + 1
        y_i = y_i − 1
        Δ_i = Δ_i + 2x_i − 2y_i + 2
        go to 1
        шаг m_V
30      y_i = y_i − 1
        Δ_i = Δ_i − 2y_i + 1
        go to 1
4       finish
```

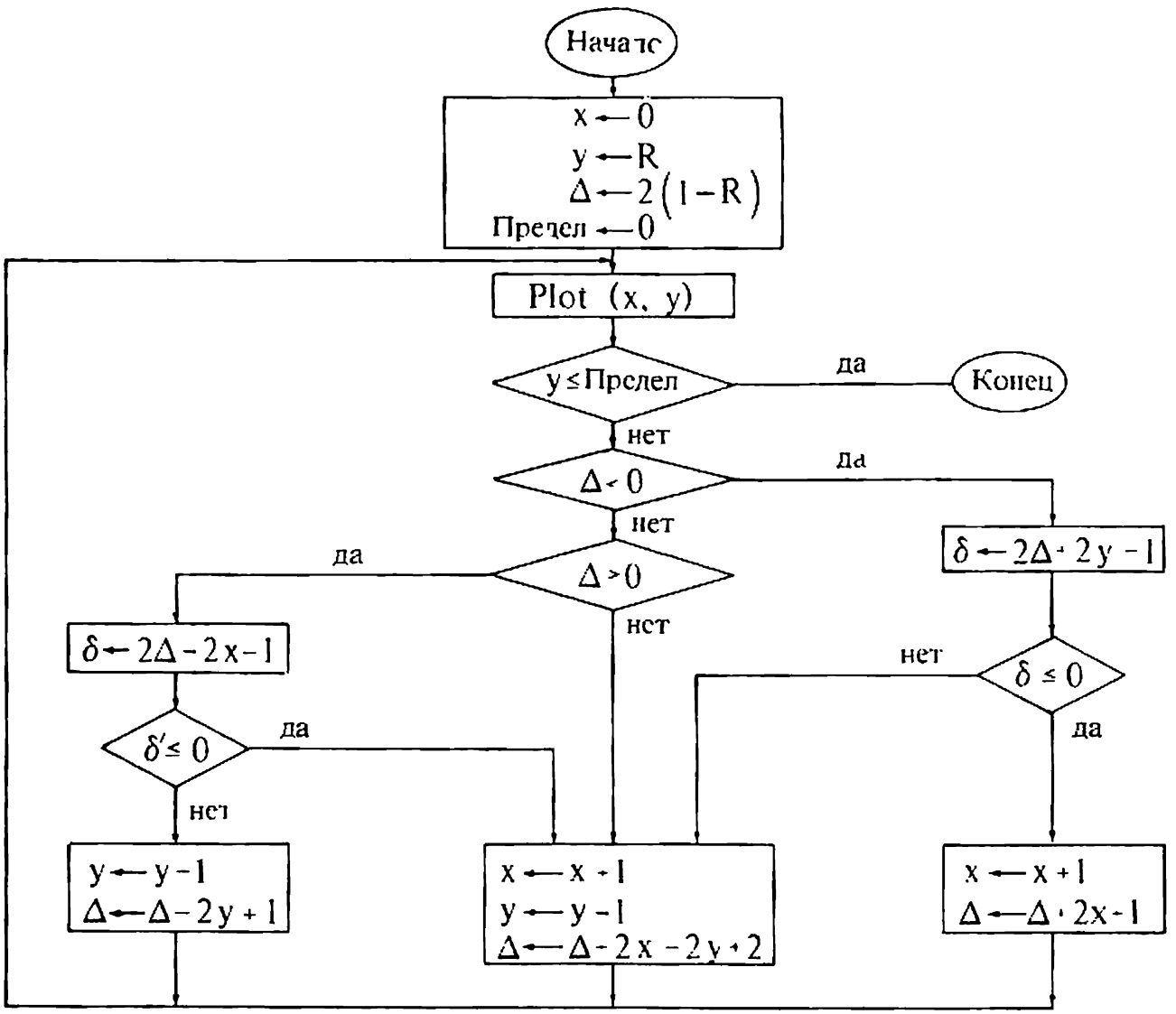



Рис. 2.14. Блок-схема пошагового алгоритма Брезенхема генерации окружности в первом квадранте.

Переменная предела устанавливается в нуль для окончания работы алгоритма на горизонтальной оси, в результате генерируется окружность в первом квадранте. Если необходим лишь один из октантов, то второй октант можно получить с помощью установки Предел = **Integer** ($R/\sqrt{2}$), а первый — с помощью отражения второго октанта относительно прямой $y = x$ (рис. 2.10). Блок-схема алгоритма приводится на рис. 2.14.

**Пример 2.5.** Алгоритм Брезенхема для окружности

Для иллюстрации работы алгоритма генерации окружности рассмотрим окружность радиуса 8 с центром в начале координат. Генерируется только первый квадрант.

начальные установки

x = 0
y = 8
$\Delta_1 = 2(1 - 8) = -14$
Предел = 0

пошаговое выполнение основного цикла

1 **Plot**(0,8)
$y_i >$ Предел продолжать
$\Delta_i < 0$ **go to** 2
2 $\delta = 2(-14) + 2(8) - 1 = -13 < 0$ **go to** 10
10 x = 0 + 1 = 1
$\Delta_1 = -14 + 2 + 1 = -11$
**go to** 1

1 **Plot**(1,8)
$y_i >$ Предел продолжать
$\Delta_i < 0$ **go to** 2
2 $\delta = 2(-11) + 2(8) - 1 = -7 < 0$ **go to** 10
10 x = 1 + 1 = 2
$\Delta_1 = -11 + 2(2) + 1 = -6$
**go to** 1

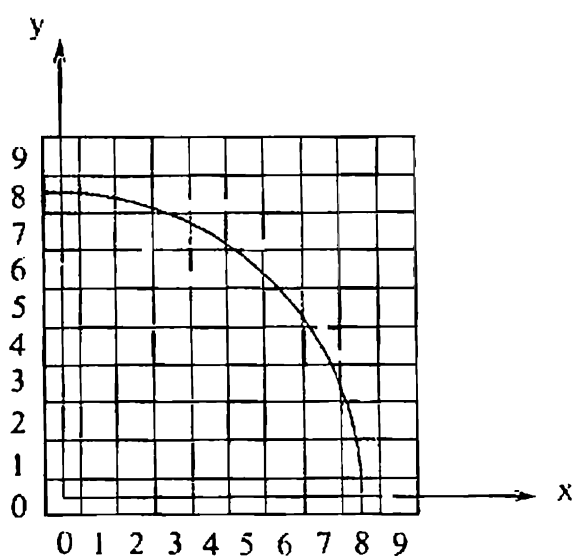
1 **Plot**(2,8)
.
.
.
продолжать

Результаты всех последовательных проходов алгоритма сведены в таблицу. Список пикселов, выбранных алгоритмом, состоит из (0, 8), (1, 8), (2, 8), (3, 7), (4, 7), (5, 6), (6, 5), (7, 4), (7, 3), (8, 2), (8, 1), (8, 0).

| Plot | $\Delta_1$ | $\delta$ | $\delta'$ | x | y |
|---|---|---|---|---|---|
| | −14 | | | 0 | 8 |
| (0, 8) | | | | | |
| | −11 | −13 | | 1 | 8 |

```
(1  8)
              -6     -7                2        8
(2, 8)
             -12      3                3        7
(3, 7)
              -3     -11               4        7
(4, 7)
              -3      7                5        6
(5, 6)
               1      5                6        5
(6, 5)
               9             -11       7        4
(7, 4)
               4              3        7        3
(7, 3)
              18             -7        8        2
(8, 2)
              17             19        8        1
(8, 1)
              18             17        8        0
(8, 0)
                  алгоритм завершен
```

Результаты показаны на рис. 2.15 вместе с реальной окружностью. Алгоритм легко обобщается для других квадрантов или дуг окружностей.




Радиус окружности равен 8

| Подсвеченные | пикселы |
|---|---|
| 0 | 8 |
| 1 | 8 |
| 2 | 8 |
| 3 | 7 |
| 4 | 7 |
| 5 | 6 |
| 6 | 5 |
| 7 | 4 |
| 7 | 3 |
| 8 | 2 |
| 8 | 1 |
| 8 | 0 |

Рис. 2.15. Результаты работы пошагового алгоритма Брезенхема генерации окружности.

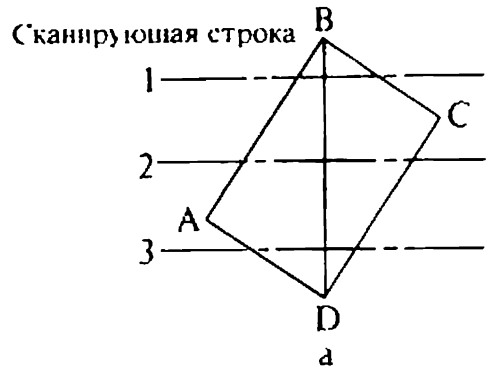
## 2.7. РАСТРОВАЯ РАЗВЕРТКА — СПОСОБ ГЕНЕРАЦИИ ИЗОБРАЖЕНИЯ

Для вывода на видеомонитор разложенный в растр образ необходимо представить в виде того шаблона, который требует дисплей (см. разд. 1.8). Это преобразование называется растровой разверткой. В отличие от дисплейного списка для векторного дисплея, содержащего информацию голько об отрезках или литерах, в данном случае дисплейный список должен содержать информацию о каждом пикселе на экране. Необходимо, кроме того, чтобы эта информация организовывалась и выводилась со скоростью видеогенерации в порядке сканирования строк, т. е. сверху вниз и слева направо. Существует четыре способа достижения такого результата — растровая развертка в реальном времени, групповое кодирование, клеточная организация и память буфера кадра.

## 2.8. РАСТРОВАЯ РАЗВЕРТКА В РЕАЛЬНОМ ВРЕМЕНИ

При развертке в реальном времени или «на лету» сцена произвольно представляется в терминах визуальных атрибутов и геометрических характеристик. Типичными визуальными атрибутами являются цвет, оттенок и интенсивность, тогда как координаты $x$, $y$, углы наклона и текст относятся к геометрическим характеристикам. Последние упорядочены по координате $y$. Во время воспроизведения каждого кадра процессор сканирует эту информацию и вычисляет интенсивность каждого пиксела на экране. При такой развертке не нужны большие количества памяти. Требования на память обычно ограничиваются необходимостью хранить дисплейный список плюс одну сканирующую строку. Более того, поскольку информация о сцене хранится в произвольно организованном дисплейном списке, добавление или удаление информации из списка осуществляется легко, а это удобно для динамических приложений. Однако сложность выводимого изображения ограничивается скоростью дисплейного процессора. Обычно это означает, что ограничено число отрезков или многоугольников в картине, количество пересечений со сканирующей строкой или число цветов или полутонов серого.

Для получения пересечений (если они есть) каждого отрезка дисплейного списка со сканирующей строкой в простейшей реализации метода всякий раз при изображении строки обрабатывается весь дисплейный список. При регенерации видеоизображения на каждую сканирующую строку, а значит, и на обработку всего списка приходится только 63.5 микросекунды. Столь малое время позволяет ис-




a

| Сканирующая строка | 1 | 2 | 3 | 1 | 2 | 3 | 1 | 2 | 3 |
|---|---|---|---|---|---|---|---|---|---|
| | BC←b | BC | BC | BA - b | BA - b | BA | BD←b | BD←b | BD←b |
| | BA | BA←b | BA | BC | BC | BC - b | BA | BA | BA |
| | BD - e | BD | BD←b | BD←e | BD | BD | BC←e | BC | BC |
| | CD | CD←e | CD | CD | CD←e | CD | CD | CD←e | CD |
| | AD | AD | AD←e | AD | AD | AD←e | AD | AD | AD - e |
| | | b | | | c | | | d | |

Рис. 2.16. Простой список активных ребер.

пользовать данный метод только для рисования несложных чертежей, не более. Так как в общем случае не каждый отрезок в сцене пересекает каждую сканирующую строку, то количество вычислений может быть сокращено путем введения списка активных ребер (САР). Этот список содержит те отрезки изображения, которые пересекают сканирующую строку.

Для организации и управления САР можно использовать ряд методов. Сначала отрезки изображения сортируются по наибольшей координате $y$. В одном из простых методов такой сортировки используются два перемещающихся указателя в отсортированном списке. Указатель начала используется для обозначения начала списка активных ребер, а указатель конца — для обозначения конца этого списка. На рис. 2.16,a представлена сцена из нескольких отрезков с тремя характерными сканирующими строками. На рис. 2.16,b показан типичный отсортированный список отрезков фигуры. Указатель начала в исходном положении устанавливается на начало этого списка, т. е. на отрезок *BC*. Указатель конца установлен на тот последний отрезок в списке, который начинается *выше* рассматриваемой сканирующей строки, т. е. на отрезок *BD*. При сканировании изображения необходимо корректировать САР, при этом указатель конца передвигают вниз, чтобы включить в список новые отрезки, начинающиеся на текущей сканирующей строке или выше нее. В то же самое время указатель начала передвигают вниз, чтобы исключить отрезки, кончающиеся выше текущей сканирующей строки. Это изображено на рис. 2.16 для скани-

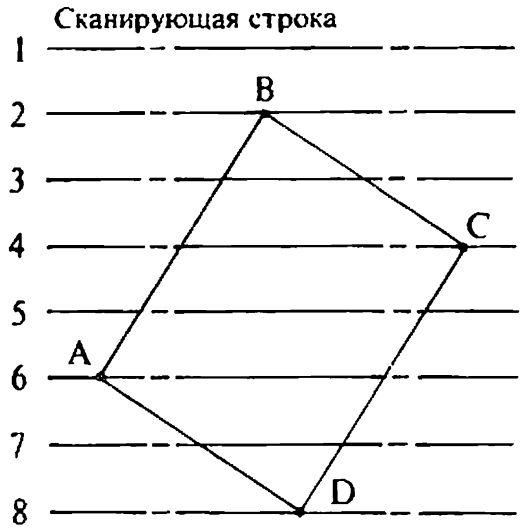
рующих строк, помеченных цифрами 2 и 3 на рис. 2.16,a. Рисунки 2.16,c и d иллюстрируют проблему, возникающую в этом простом алгоритме. Порядок сортировки отрезков, начинающихся с одной и той же координаты $y$, влияет на размер списка активных ребер. Например, отрезок $BC$ на рис. 2.16,d никогда не покинет этот список. В результате может обрабатываться больше информации, чем на самом деле необходимо.

Эту и аналогичные проблемы можно устранить путем введения дополнительной структуры данных. При этом можно упростить также вычисление пересечения каждого отрезка изображения со сканирующими строками. Сначала выполняется групповая сортировка по $y$ всех отрезков изображения. При групповой сортировке по $y$[1] (рис. 2.17,b) просто создаются области памяти или группы для каждой сканирующей строки. Если, например, применяется 512 сканирующих строк, то используется 512 групп. При просмотре отрезков в дисплейном списке информация о каждом отрезке помещается в группу, соответствующую наибольшей величине координаты $y$ для отрезка. Для простого черно-белого контурного изображения необходимо записывать только координату $x$ точки пересечения с групповой сканирующей строкой, $\Delta x$ — изменение этой координаты $x$ при переходе от одной сканирующей строки к другой, и $\Delta y$ — число сканирующих строк, пересекаемых отрезком. Для простых изображений большинство из $y$-групп будет пусто.

Список активных ребер для текущей сканирующей строки формируется добавлением информации из $y$-группы, соответствующей этой строке. Координаты $x$ точек пересечения сортируются в порядке сканирования, и ребра из САР преобразуются в растровую форму. После этого для каждого отрезка из САР $\Delta y$ уменьшается на единицу. Если $\Delta y < 0$, то отрезок исключается из списка. И наконец, для каждого отрезка координата $x$ точки пересечения для новой сканирующей строки получается добавлением к прежнему значению величины $\Delta x$. Этот процесс повторяется для всех сканирующих строк. На рис. 2.17,c приводится САР для сканирующих строк 3, 5 и 7 из простой сцены на рис. 2.17,a.

Если используется фиксированный размер $y$-групп, то для пересечений с каждой сканирующей строкой выделяется фиксированное количество памяти. Таким образом, максимальное число пересечений с произвольной сканирующей строкой предопределено заранее и, следовательно, сложность изображения ограничена. Одним из

[1] Групповая сортировка является одной из форм распределяющей сортировки, где основание счисления равно числу групп (сканирующих строк). См. Кнут [2-11].




a

y-группа

1 пусто
2 $x_{BA}, \Delta x_{BA}, \Delta y_{BA}$, $x_{BC}, \Delta x_{BC}, \Delta y_{BC}$
3 пусто
4 $x_{CD}, \Delta x_{CD}, \Delta y_{CD}$
5 пусто
6 $x_{AD}, \Delta x_{AD}, \Delta y_{AD}$
7 пусто
8 пусто

b

Список активных ребер

Сканирующая строка 3: $x_{BA} + \Delta x_{BA}, \Delta x_{BA}, \Delta y_{BA} - 1, x_{BC} + \Delta x_{BC}, \Delta x_{BC}, \Delta y_{BC} - 1$

Сканирующая строка 5: $x_{BA} + 3\Delta x_{BA}, \Delta x_{BA}; \Delta y_{BA} - 3, x_{CD} + \Delta x_{CD}, \Delta x_{CD}, \Delta y_{CD} - 1$

Сканирующая строка 7: $x_{CD} + 3\Delta x_{CD}, \Delta x_{CD}, \Delta y_{CD} - 3, x_{AD} + \Delta x_{AD}, \Delta x_{AD}, \Delta y_{AD} - 1$

c

y-группа

1 —
2 1
3 —
4 8
5 —
6 12
7 —
8 —

Индексированный список

1 $x_{BA}$
2 $\Delta x_{BA}$
3 $\Delta y_{BA}$
4 $x_{BC}$
5 $\Delta x_{BC}$
6 $\Delta y_{BC}$
7 —
8 $x_{CD}$
9 $\Delta x_{CD}$
10 $\Delta y_{CD}$
11 —
12 $x_{AD}$
13 $\Delta x_{AD}$
14 $\Delta y_{AD}$
15 —

≡ Завершение или пусто
Конец сканирующей строки

d

Рис. 2.17. Групповая сортировка по $y$, список активных ребер и структура последовательного индексированного списка.

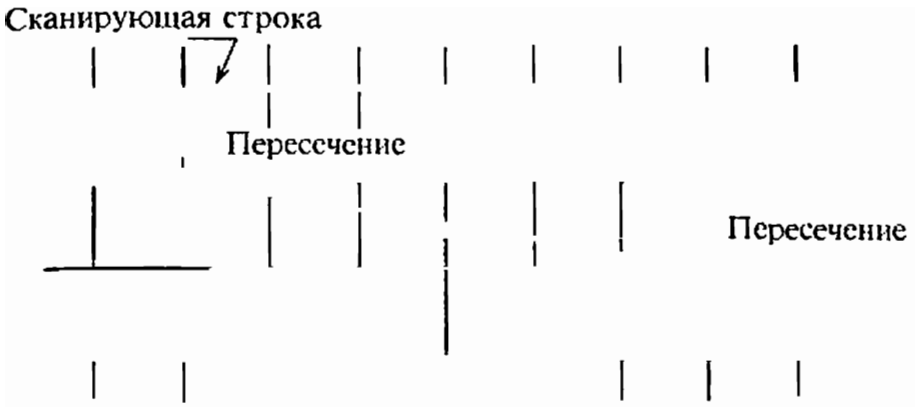



Рис. 2.18. Простой метод растровой развертки для почти горизонтальных отрезков.

методов, позволяющих преодолеть это ограничение, может служить использование в качестве структуры данных последовательного индексированного списка. В этом случае каждая $y$-группа содержит только указатель на расположение в структуре данных информации для первого отрезка из группы (т. е. начинающегося на этой сканирующей строке). На рис. 2.17, d показаны последовательный индексированный список и структура данных для рис. 2.17, a. В этой конкретной ситуации предполагается, что данные для текущей сканирующей строки выбираются группами по три до тех пор, пока не встретиться пустая ссылка или знак завершения.

Метод определения пересечений отрезков со сканирующими строками дает хорошие результаты для вертикальных и почти вертикальных отрезков. Однако для почти горизонтальных отрезков будет вычислено очень мало точек пересечения, что приведет к неприемлемому изображению отрезка. В качестве простого решения можно предложить определять пересечения на двух последовательных сканирующих строках и активировать все пикселы между точками пересечений, как это показано на рис. 2.18. Для горизонтальных отрезков используются концевые точки.

Так как все изображение обрабатывается для каждого видеокадра, развертка в реальном времени применима для высоко интерактивной графики. При использовании групповой сортировки по $y$ отрезки могут быть добавлены или удалены из дисплейного списка простым добавлением или удалением их из соответствующей $y$-группы и связанной с ней структуры данных. Как показано выше на рис. 2.17, b, это легче всего сделать для $y$-групп фиксированной длины. Для удобства добавления и удаления отрезков в сцене используется структура данных в виде связного списка (рис. 2.19). Заметим, что для связного списка на рис. 2.19, b, требуются признаки конца каждой группы данных и ссылка на следующую группу для рассматриваемой сканирующей строки, например элемент 4, а так-

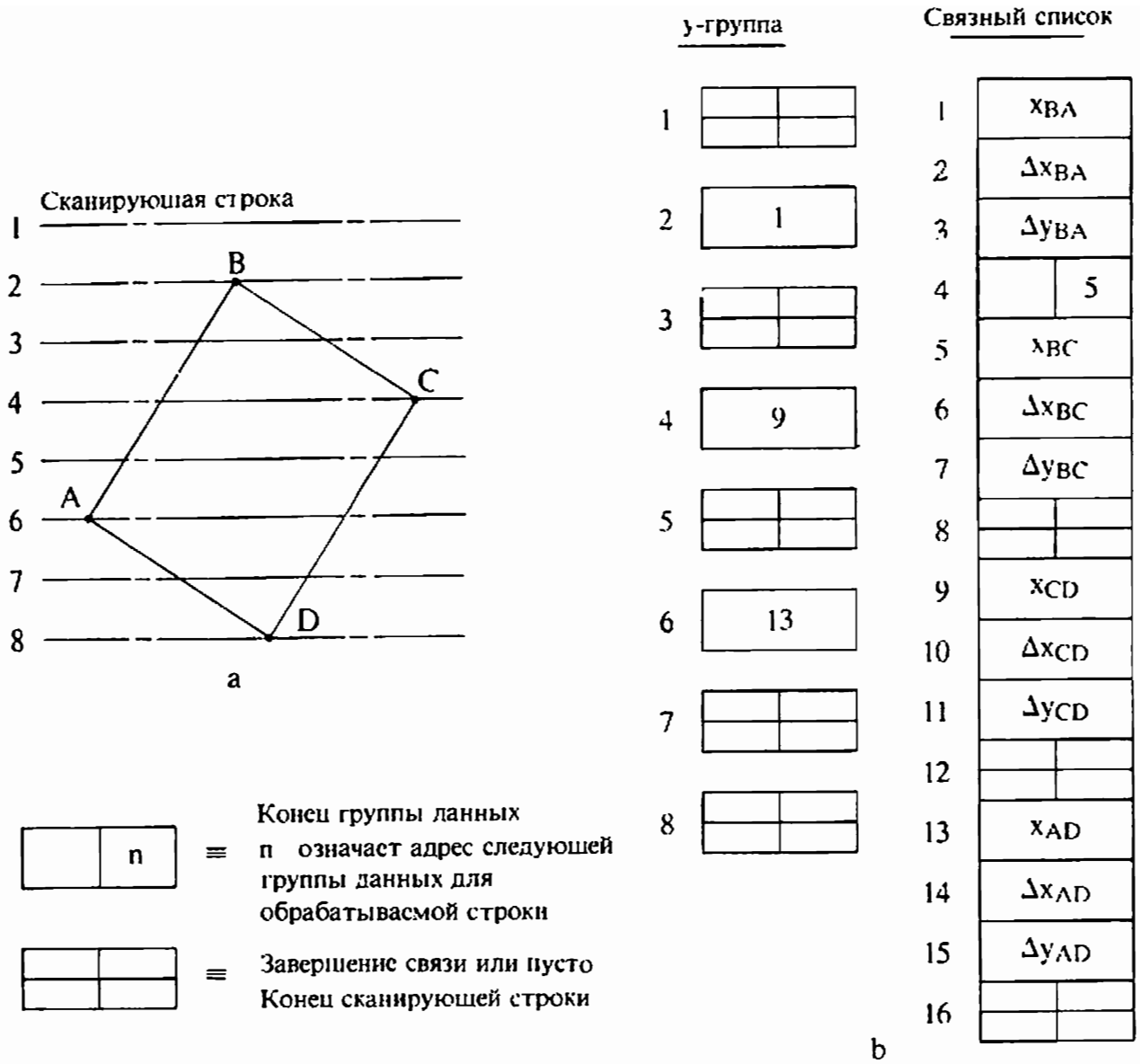



Рис. 2.19. Групповая сортировка по $y$ и связный список для случая интерактивной графики.

же признак завершения связи. При добавлении отрезка $BD$ список модифицируется так, как показано на рис. 2.19, d. Информация об отрезке $BD$ добавляется к концу списка данных. Дисплейный процессор направляется в эту ячейку с помощью модифицированной ссылки из ячейки 8. Если теперь отрезок $BC$ удаляется из фигуры, список модифицируется так, как показано на рис. 2.19, f. Заметим, что ссылка в ячейке 4 модифицирована для того, чтобы обойти ячейки, содержащие информацию об отрезке $BC$.

Этот простой пример иллюстрирует основные идеи для модификации связного списка в интерактивных графических системах. Однако здесь не приведены все необходимые подробности. Например, должно быть очевидно, что длина списка будет постоянно

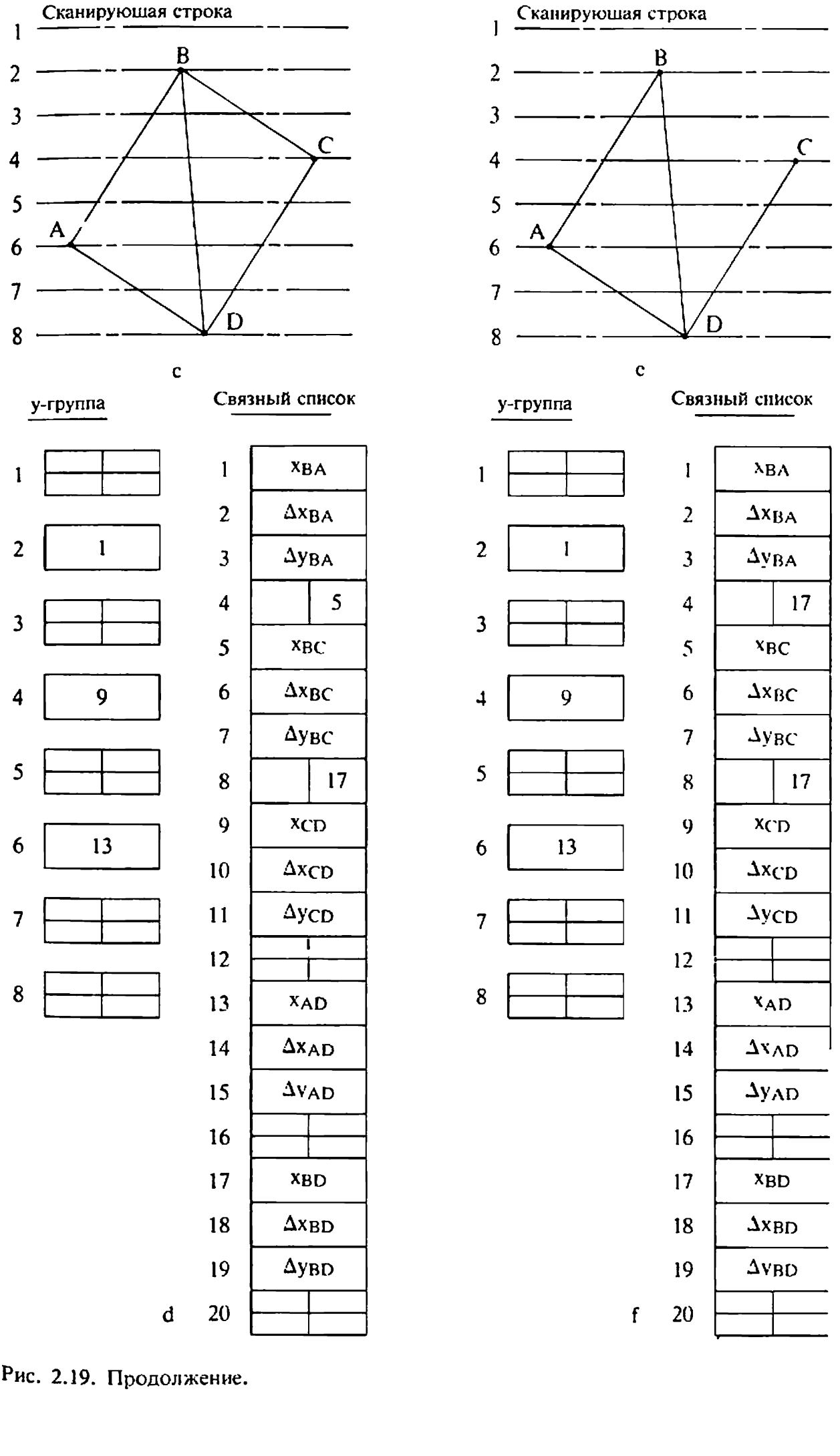



Рис. 2.19. Продолжение.

расти, если только «потерянные» ячейки (с 5 по 8 на рис. 2.19, f) не будут снова использованы или список не будет сжат. Дополнительную информацию о связных списках и структурах данных можно найти, например, в [2-12].

Так как алгоритм, работающий в таких жестких ограничениях на время обработки одного видеокадра программно реализовать трудно, то успешные программные реализации используются главным образом в имитационных системах, таких, как летные тренажеры, навигационные тренажеры для кораблей и т. п.

## 2.9. ГРУППОВОЕ КОДИРОВАНИЕ

В методе группового кодирования сделана попытка воспользоваться тем, что большие области изображения имеют одинаковую интенсивность или цвет. При простейшем групповом кодировании определяется только интенсивность и количество последовательных пикселов с этой интенсивностью на данной сканирующей строке. На рис. 2.20, a показан простой черно-белый чертеж на растре 30 × 30 и соответствующие кодирующие последовательности для сканирующих строк с номерами 1, 15 и 30. Кодирующие данные следует рассматривать группами по два. Первое число — интенсивность, второе — число последовательных пикселов на сканирующей строке с этой интенсивностью:

| Интенсивность | Длина участка |
|---|---|

Таким образом, в строке 1 на рис. 2.20, a имеется 30 пикселов нулевой интенсивности, т. е. черных или фоновых. Все изображение можно закодировать с помощью 208 чисел. Попиксельное хранение, т. е. одна порция информации на каждый пиксел (битовая карта), потребовало бы 900 значений интенсивности для растра 30 × 30. В этом случае сжатие данных с помощью группового кодирования составляет 4.33 : 1.

Легко обрабатываются с помощью данного метода сплошные фигуры, как это продемонстрировано на рис. 2.20, b с кодированием 1, 15 и 30 строк. Особый интерес представляет 15-я сканирующая строка. Все изображенные на рис. 2.20, b может быть закодировано с использованием 136 чисел при сжатии данных 6.62 : 1. Большая степень сжатия изображений со сплошными фигурами по сравнению с сеточными рисунками объясняется тем, что два ребра покрываются одной парой «интенсивность — длина».

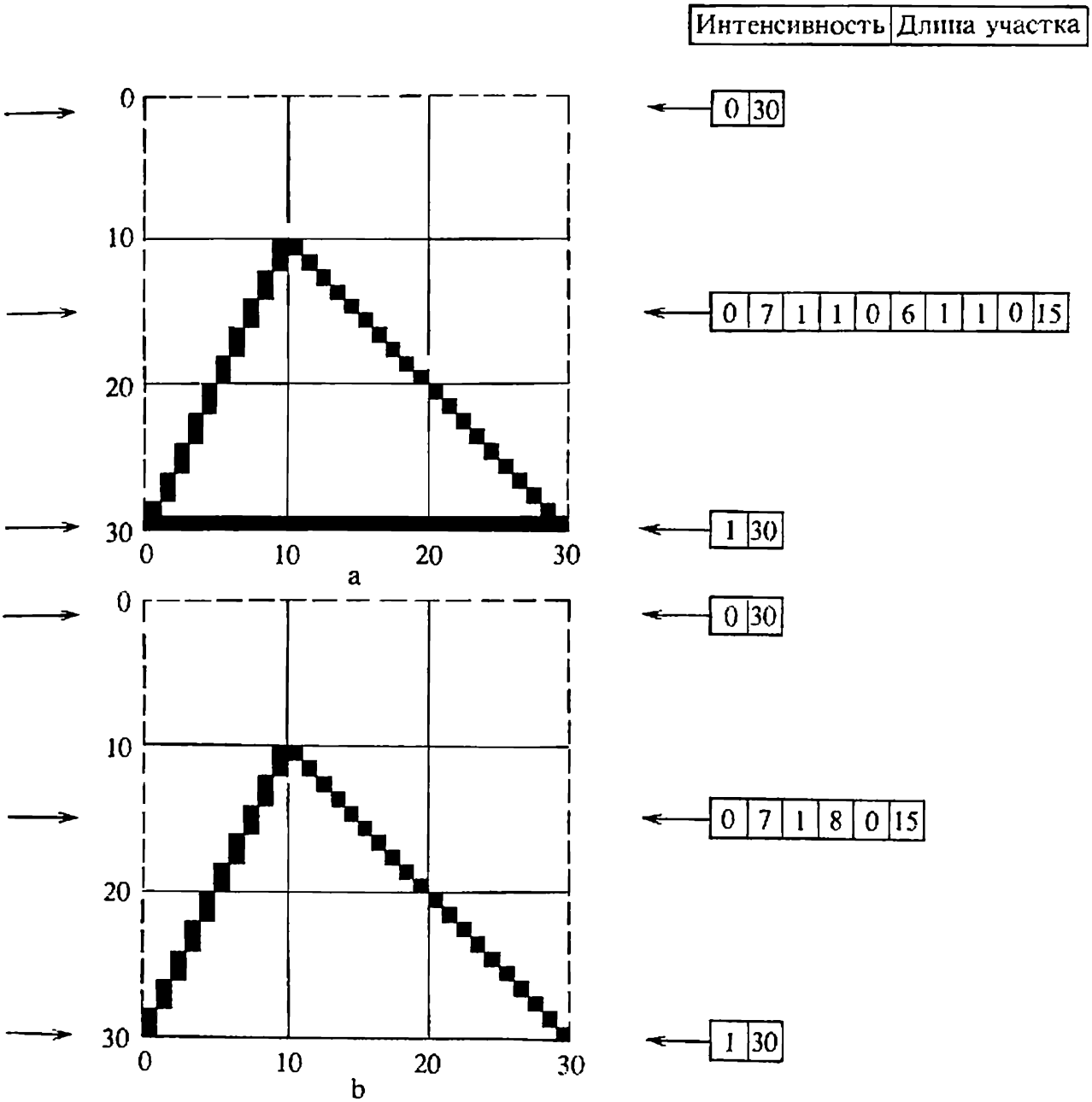



Рис. 2.20. Примеры группового кодирования.

Для добавления цвета эта простая схема группового кодирования может быть легко расширена. На данной сканирующей строке для цвета приводятся интенсивности красной, зеленой и синей цветовых пушек, а за ними — количество последовательных пикселов с этим цветом; например,

| Интенсивность красного | Интенсивность зеленого | Интенсивность синего | Длина |
|---|---|---|---|

Для простого цветного дисплея, в котром цветовая пушка либо выключена (0), либо включена (1), кодирование 15 строки

рис. 2.20, b с желтым треугольником на синем фоне (см. табл. 1.1) выглядит так:

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 7 | 1 | 1 | 0 | 8 | 0 | 0 | 1 | 15 |

Сжатие данных для изображений, закодированных группами, может приближаться к 10:1. Это существенно не только потому, что групповое кодирование просто экономит память, но и потому, что оно экономит память для машинно-синтезированных последовательностей кадров или фильма. Оно также экономит время передачи по телеграфу для фотографий и факсимиле, в которых широко используется групповое кодирование. Рассмотрим, например, потребность в памяти для изображений с разрешением 512 × 512 × 8 в 30-секундном фильме, в котором кадры следуют с частотой видеогенерации, т. е. 30 кадр/с. Требуемая память составляет

$$(512 \times 512 \times 8 \times 30 \times 30)/(8 \text{ бит/байт}) = 236 \text{ Мбайт}$$

Такое количество информации поместится только на больших дисковых устройствах. Однако даже умеренное сжатие 4:1 при групповом кодировании позволит хранить его на одном диске малого или среднего размера.

У группового кодирования есть и недостатки. Добавление или удаление отрезков или текста из изображения является трудоемкой операцией и занимает много времени из-за последовательного хранения длин участков. Кодирование и декодирование изображения влечет за собой накладные расходы. Наконец, для коротких участков одинаковой интенсивности может потребоваться в два раза больше памяти, чем при попиксельном хранении. Это иллюстрируется на рис. 2.21, где изображение состоит из чередующихся чер-

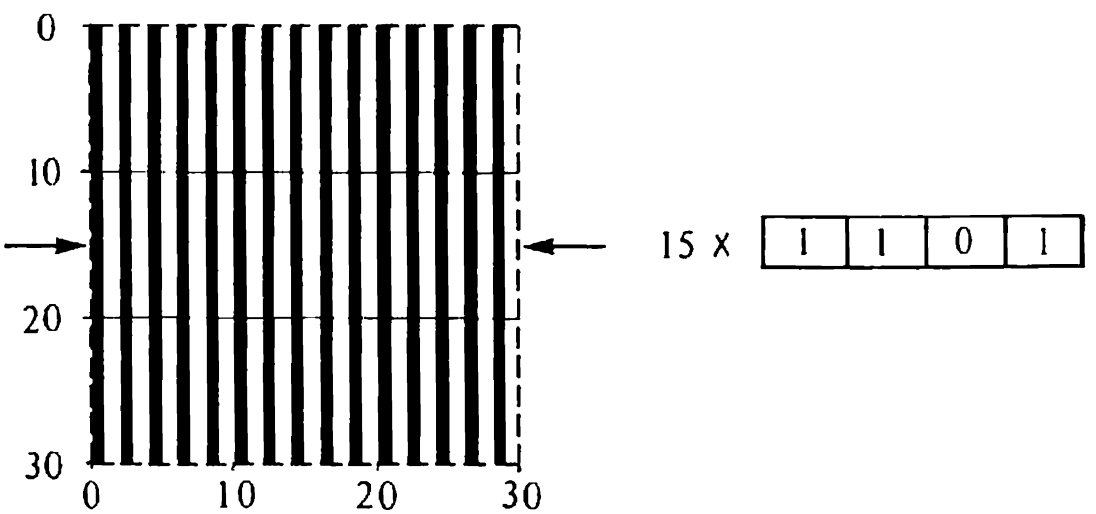



Рис. 2.21. Ограничения группового кодирования для коротких участков.

ных и белых вертикальных отрезков шириной в один пиксел. В этом случае кодирующая последовательность состоит из элементов

| 1 | 1 | 0 | 1 |
|---|---|---|---|

повторенных 15 раз. Таким образом, при построчном кодировании должно храниться 1800 значений вместо 900 при попиксельном хранении. Сжатие данных здесь составляет 1/2.

Лоос в [2-13] и Хартке, Стерлинг и Шимер в [2-14] обсуждают эффективную реализацию схем группового кодирования.

## 2.10 КЛЕТОЧНОЕ КОДИРОВАНИЕ

В методе группового кодирования изображение рассматривается как линейная или одномерная совокупность пикселов. В методе клеточного кодирования сделана попытка с помощью минимума информации представить целые области изображения, т. е. клетки. Для того чтобы в простейшем алфавитно-цифровом терминале с ЭЛТ можно было выполнять операции в реальном времени, используется клеточное кодирование. В таком терминале область экрана разбивается на клетки или области, достаточно большие, чтобы содержать одну литеру. Например, экран можно разбить на области размером 8 × 8 пикселов. Для дисплея с разрешением 512 × 512 получится 64 × 64 клетки, а для телевизионного дисплея 480 × 640 со стандартным видовым отношением 4:3 получится 60 × 80 клеток. Обычно клетка 8 × 8 пикселов используется для вывода литер с точечной матрицей размером 5 × 7. Дополнительные пикселы используются для разделения литер, а также для строчных литер с нижними выносными элементами. На рис. 2.22 приводится пример маски литер. Так как каждый второй ряд клеток для читабельности оставляется пустым, то для последней схемы получится 30 строк по 80 литер, что типично для многих алфавитно-цифровых дисплеев. Используются и другие размеры клеток. Например, для литер с матрицей 7 × 9 обычно используется клетка 8 × 10 пикселов, в результате дисплей содержит 24 строки по 80 литер в каждой. Шаблоны, составленные из пикселов, для каждой литеры хранятся в постоянном запоминающем устройстве (ПЗУ).

Метод клеточного кодирования можно применить и для вычерчивания линий, надо только хранить в ПЗУ еще и шаблоны сегментов отрезков. Тогда для построения необходимых линий могут быть использованы комбинации таких сегментов, расположенных в

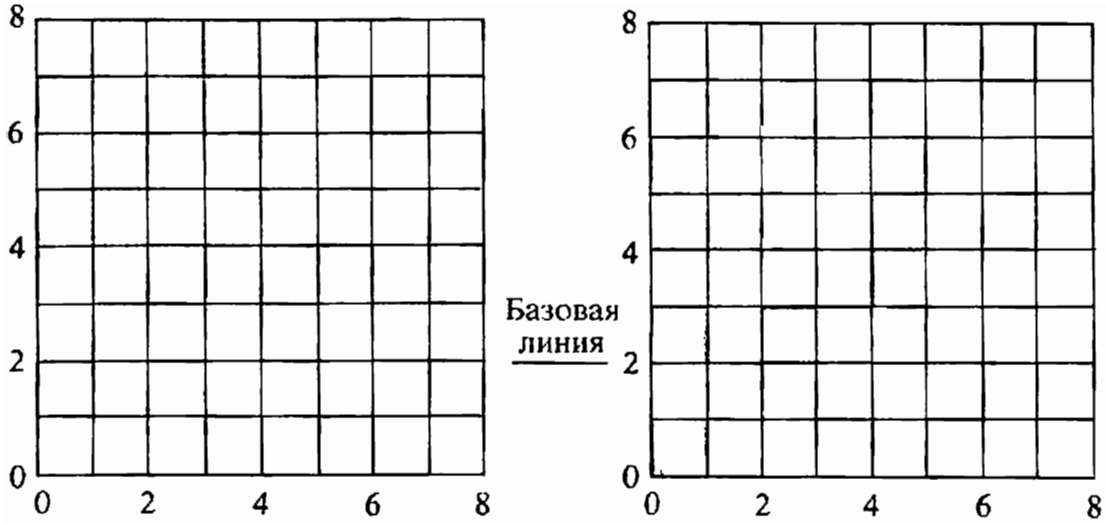



Рис. 2.22. Маска литеры при клеточном кодировании.

соседних клетках. Для произвольной клетки размером $n \times n$ существует $2^{n^2}$ возможных шаблонов, составленных из пикселов. При любом разумном значении $n$ хранить пришлось бы слишком много шаблонов; например, при $n = 8$ значение $2^{n^2} = 1.8 \times 10^{19}$. Однако не все шаблоны представляют реально возможные сегменты. Показано, например, что для алгоритма Брезенхема, обсуждавшегося выше, для отрезков с тангенсом угла наклона между 0 и 1 существует не более $2^n - 1$ шаблонов, представляющих сегменты отрезков. Наконец, Жордан и Баррет [2-15] показали, что для клетки $8 \times 8$ при использовании переноса, отражения и маскирования требуется только 108 шаблонов сегментов отрезков.

На рис. 2.23 изображен сегмент отрезка, выходящий из левого нижнего угла клетки $8 \times 8$. Этот отрезок был разложен в растр с помощью алгоритма Брезенхема, причем тангенс угла наклона положителен. В результате отражения относительно оси $x$ (рис. 2.23, a) получится отрезок, выходящий из левого верхнего угла с отрицательным тангенсом угла наклона. В результате вертикального переноса вдоль оси $y$ получится отрезок, начинающийся выше основания клетки (рис. 2.23, b), а при переносе по обеим координатам — отрезок с началом, расположенным внутри клетки.

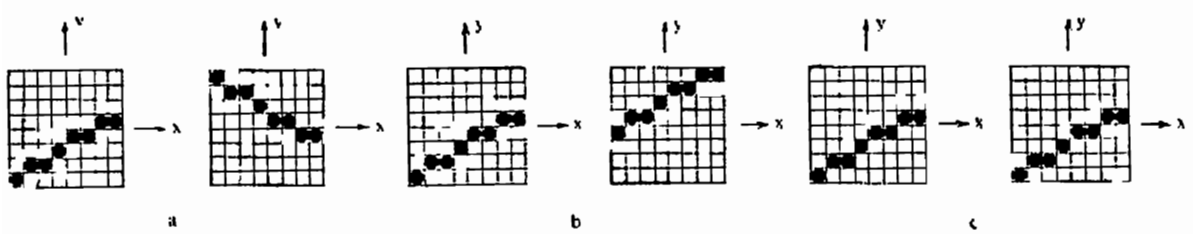



Рис. 2.23. Клеточное кодирование: (a) отражение, (b) перенос, (c) маскирование.

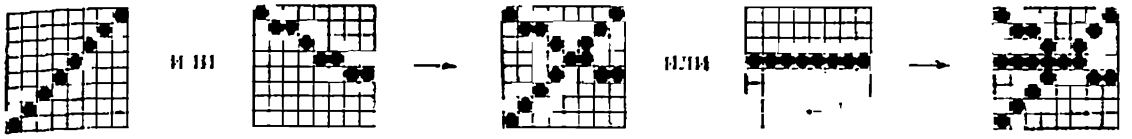



Рис. 2.24. Комбинации логического ИЛИ для сегментов отрезков.

Как показано на рис. 2.23, с, маскирование части отрезка позволяет дорисовать короткие сегменты в конце отрезка. Для вывода пересекающихся внутри клетки отрезков обеспечена возможность комбинирования шаблонов с помощью операции «Логическое ИЛИ» (**OR**). Последовательное применение этой операции позволяет бесконечно варьировать шаблоны с пересечениями, как это продемонстрировано на рис. 2.24.

В работе Баррета и Жордана [2-16] обсуждается интерактивная работа на дисплее с клеточным кодированием. Она особенно эффективна, когда для поддержки упорядоченного сверху вниз и слева направо дисплейного файла используется связный список. Однако уровень достижимой интерактивности невысок.

Метод клеточного кодирования был распространен на цветные дисплеи и на представление сплошных изображений [2-17]. При этом, однако, коэффициенты сжатия данных не настолько велики, как в случае черно-белых (двухуровневых) изображений.

## 2.11. БУФЕРЫ КАДРА

В гл. 1 при знакомстве с растровыми графическими устройствами с регенерацией предполагалось, что растровый дисплей реализуется в виде буфера кадра, состоящего из полупроводниковой памяти с произвольным доступом. Хотя это и наиболее часто встречающийся метод реализации, но для буфера кадра может быть использована и вторичная память — диск или барабан [2-18 и 2-19].

Буферы кадров можно также реализовать с помощью сдвиговых регистров [2-20]. Схематично сдвиговый регистр можно считать стеком типа FIFO (первым пришел — первым обслужен). Если стек заполнен, то при добавлении в вершину стека новых битов данных со дна выталкиваются первые биты данных. Выталкиваемые из стека данные можно интерпретировать как интенсивность пиксела сканирующей строки. Буферы кадров на сдвиговых регистрах можно реализовать, используя по одному регистру на пиксел в сканирующей строке при длине каждого регистра, равной числу строк. Другой вариант — использование единственного регистра с длиной,

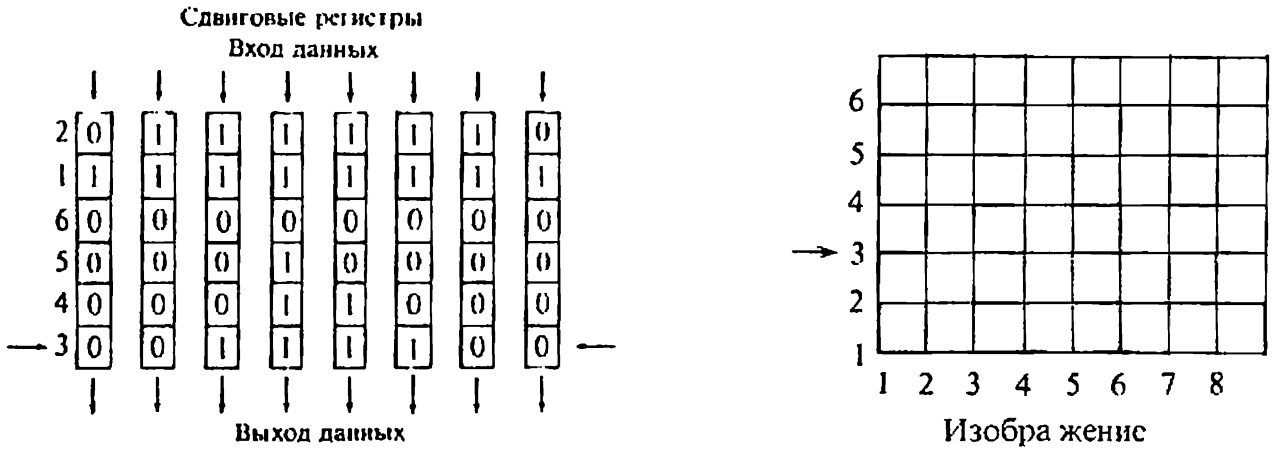



Рис. 2.25. Буфер кадра на сдвиговых регистрах.

равной числу пикселов в сканирующей строке, умноженному на число строк.

На рис. 2.25 показан простой шестистрочный дисплей по восемь пикселов на строке. На рисунке изображен также буфер кадра на сдвиговых регистрах, в который входят восемь регистров по 6 бит каждый. В буфере находится набор битов изображения. Биты для строки 3 показаны в момент выталкивания со дна сдвиговых регистров. Для согласованности со скоростью видеогенерации следует тщательно управлять последовательностью вывода сдвиговых регистров.

Для буферов кадра на вторичной памяти и на сдвиговых регистрах уровень интерактивности невысок. Для вторичной памяти причина заключается в большом времени доступа, а для сдвиговых регистров снижение эффективности интерактивной работы связано с тем, что изменения могут быть сделаны только при добавлении битов в регистр.

Как показано на рис. 2.26, схема графической системы с буфером кадра похожа на схему для векторного дисплея с регенерацией. При необходимости прикладная программа на главном компьютере модифицирует буфер кадра. Дисплейный контроллер периодически обрабатывает его в порядке сканирования строк и передает видеомонитору информацию, необходимую для регенерации изображения. Буфер кадра можно реализовать либо как часть памяти главного компьютера, либо как отдельную память. На рис. 2.27 показаны две эти схемы, реализованные со структурой общей шины. Хо-

Рис. 2.26. Графическая система с буфером кадра.

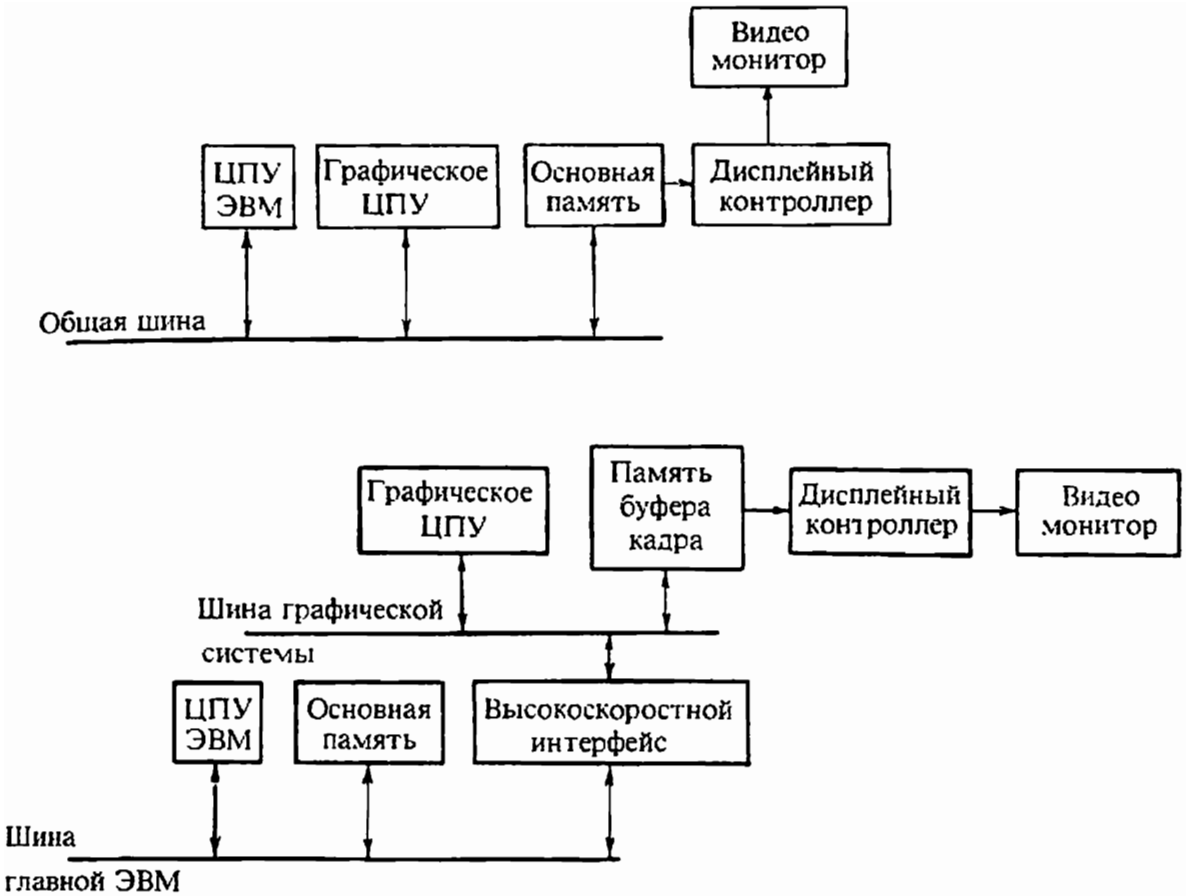



Рис. 2.27. Архитектура графических систем с буфером кадра.

тя первая схема позволяет процессору ЭВМ самому манипулировать буфером кадра (рис. 2.27, а), обычно более эффективно добавить к системе специализированный графический процессор. При получении команд от главного процессора графический процессор управляет детальной обработкой буфера кадра. При двух процессорах на общей шине и одной памяти на шине может произойти конфликтная ситуация, что сокращает среднюю производительность системы. Таким образом, для высокопроизводительных систем более предпочтительна архитектура, показанная на рис. 2.27, b. В этом случае память буфера кадра отделена от основной, что исключает конфликт на шине. Более того, графическую подсистему можно оптимизировать для улучшения характеристик модификации буфера кадров и, следовательно, для увеличения производительности системы.

## 2.12. АДРЕСАЦИЯ РАСТРА

Для простоты изложения будем считать, что пиксел в растре или буфере кадра имеет двумерные координаты $x$ и $y$, как это показано на рис. 2.28. Цифровая память, однако, организована в один линейный список адресатов, и необходимо, таким образом, преобразова-

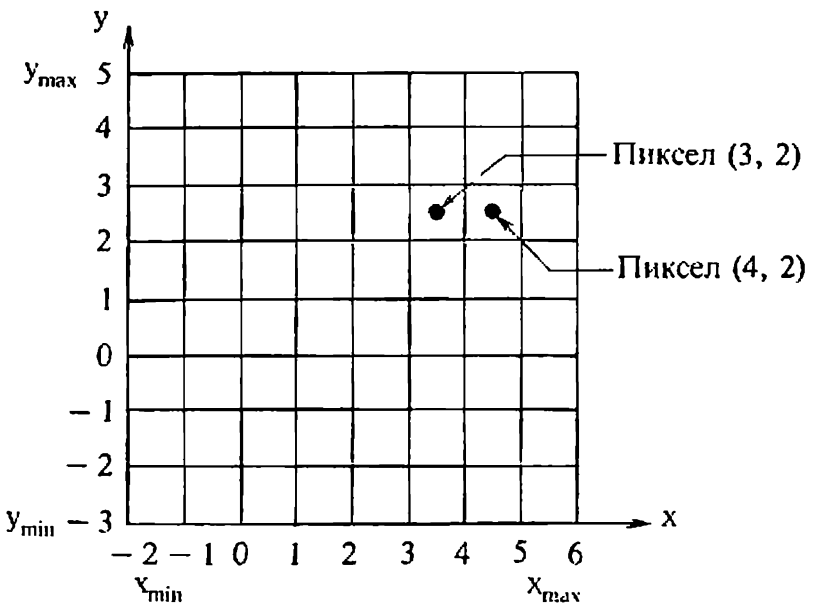



Рис. 2.28. Система координат растра.

ние координатного представления в линейное. Предположим, что начальный адрес в памяти не равен нулю, тогда преобразование задается формулой

$$\text{Адрес} = (x_{max} - x_{min})(y - y_{min}) + (x - x_{min}) + \text{базовый адрес}$$

В вычислении первого члена участвует число строк. Второй член добавляет адрес в строке, а последний — начальный адрес. Пиксел идентифицируется координатами своего левого нижнего угла.

**Пример 2.6.** Адресация растра.

Рассмотрим пиксел с координатами (3,2) в небольшом растре на рис. 2.28. Здесь $x_{max} = 6$, $x_{min} = -2$, $y_{max} = 5$, $y_{min} = -3$, причем первый пиксел из левого нижнего угла хранится в первой ячейке памяти; база или начальный адрес равен 1. Следовательно, адрес пиксела вычисляется по формуле

$$\text{Адрес} = [6 - (-2)] \times [2 - (-3)] + [3 - (-2)] + 1 = (8) \times (5) + 5 + 1 = 40 + 6 = 46$$

Данный результат можно проверить непосредственным подсчетом квадратов на рисунке.

Эта же схема работает и в случае, когда положительная ось $x$ направлена вправо, а положительная ось $y$ — вниз, при условии адресации пиксела относительно левого верхнего угла.

Как правило, для заданного буфера кадра величины $x_{max}$, $x_{min}$, $y_{min}$ и базовый адрес постоянны. Уравнение можно переписать в виде

$$\text{Адрес} = K_1 + K_2 y + x$$

где

$$K_1 = \text{базовый адрес} - K_2 y_{\min} - x_{\min}$$
$$K_2 = x_{\max} - x_{\min}$$

Вычисление адреса пиксела, следовательно, требует только двух сложений и одного умножения. При последовательной адресации пикселов для дальнейшего уменьшения работы, связанной с определением адреса, можно использовать пошаговые вычисления. В частности,

$$\text{Адрес}(x \pm 1, y) = K_1 + K_2 y + x \pm 1 = \text{Адрес}(x, y) \pm 1$$
$$\text{Адрес}(x, y \pm 1) = K_1 + K_2(y \pm 1) + x = \text{Адрес}(x, y) \pm K_2$$
$$\text{Адрес}(x \pm 1, y \pm 1) = K_1 + K_2(y \pm 1) + x \pm 1 = \text{Адрес}(x, y) \pm K_2 \pm 1$$

Здесь для горизонтального или вертикального приращения в растре требуется только одно сложение или вычитание, а для диагонального приращения — только два сложения или вычитания. Операция умножения полностью исключена из вычислений.

**Пример 2.7.** Пошаговая адресация растра

Рассмотрим пиксел с координатами (4. 2) растра на рис. 2.28. Здесь

$$K_2 = 6 - (-2) = 8$$
$$K_1 = 1 - (-8)(-3) - (-2) = 27$$
$$\text{Адрес} = 27 + (8)(2) + 4 = 47$$

Вспомнив результат для пиксела (3. 2) в предыдущем примере и использовав пошаговые вычисления, получим

$$\text{Адрес}(x + 1, y) = \text{Адрес}(x, y) + 1$$
$$\text{Адрес}(4, 2) = 46 + 1 = 47$$

## 2.13. ИЗОБРАЖЕНИЕ ОТРЕЗКОВ

Подобная адресация буфера кадра позволяет обращаться с ним как с графическим дисплеем на запоминающей трубке. Сначала буфер кадра очищается или устанавливается в фоновую интенсивность или цвет. Вместо того чтобы записывать векторы прямо на экран дисплея, для разложения в растр отрезка применяется либо алгоритм Брезенхема, либо ЦДА и соответствующие пикселы записываются в буфер кадра. Когда изображение или кадр построены, дисплейный контроллер читает буфер кадра в порядке сканирования строк и выводит результат на видеомонитор.

Выборочное стирание отрезков можно реализовать с помощью повторного использования алгоритма разложения в растр и записи

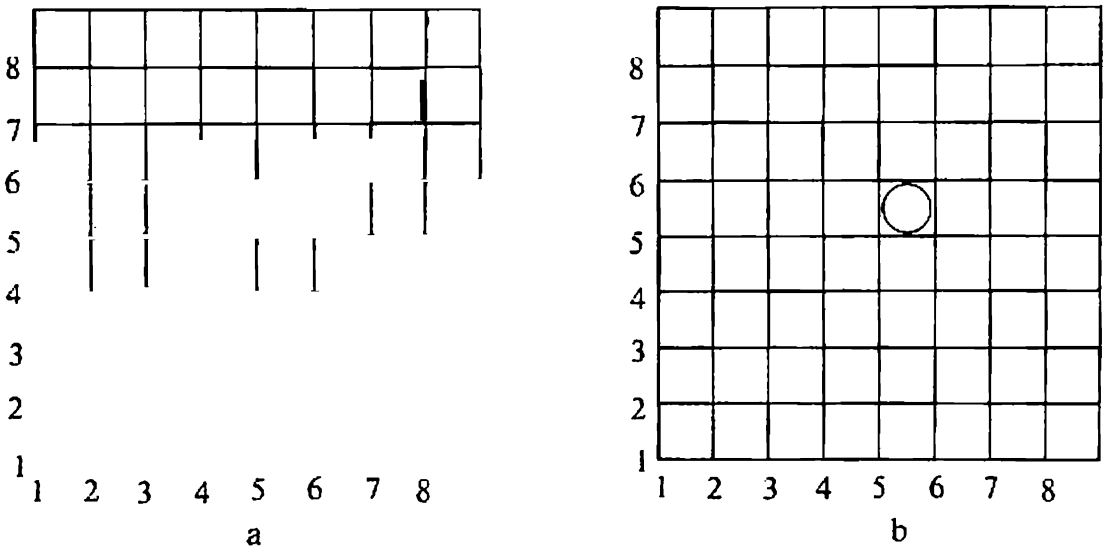



Рис. 2.29. Выборочное стирание отрезков в буфере кадра.

соответствующих пикселов с фоновой интенсивностью или цветом. Проблема, возникающая при использовании данного метода, иллюстрируется на рис. 2.29. Если удаляемый отрезок пересекает другой отрезок, то в последнем появится разрыв. На рис. 2.29,а показаны два пересекающихся отрезка. Если горизонтальный отрезок $y = 5$ стирается с помощью записи пикселов с фоновой интенсивностью или цветом в буфер кадра, то в результате в другом отрезке появится разрыв в пикселе (5, 5). Обнаружить и заполнить разрывы не составляет труда, надо только определить пересечение удаляемого отрезка со всеми другими отрезками в изображении. Данная операция для сложного изображения может занять много времени.

Для уменьшения затрат можно использовать оболочечный или минимаксный текст. Этот метод проиллюстрирован на рис. 2.30. Отрезок *ab* могут пересекать только те отрезки, которые проходят через нарисованную пунктиром прямоугольную оболочку, сформированную из минимальных и максимальных значений координат $x$,

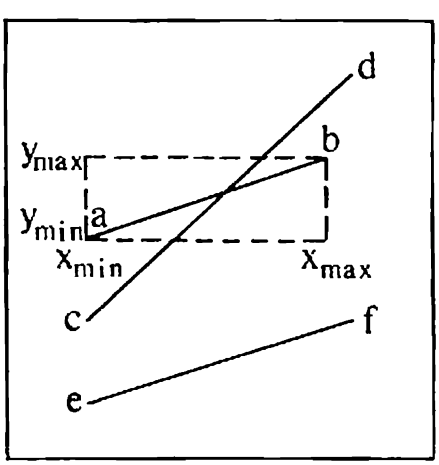



Рис. 2.30. Оболочечный или минимаксный тест.

$y$ отрезка $ab$. Тесты для каждого отрезка выглядят следующим образом:

***Минимаксный или оболочечный тест***

```
if  (Xотрmax < Xоболmin) or
    (Xотрmin > Xоболmax) or
    (Yотрmax < Yоболmin) or
    (Yотрmin > Yоболmax)
then
    пересечения нет
else
    вычислить пересечение
finish
```

## 2.14. ИЗОБРАЖЕНИЕ ЛИТЕР

Алфавитно-цифровые символы (литеры) записываются в буфер кадра с помощью маски. Литерная маска — это маленький растр, содержащий относительные адреса пикселов, используемых для представления литеры (см. рис. 2.22). С помощью литерной маски также можно представить и специальные символы, специфичные в конкретной прикладной области, например резисторы, конденсаторы или математические символы. Сама маска просто содержит двоичные величины, обозначающие, используется или нет конкретный пиксел в маске для представления формы буквы или символа. Для простых черно-белых изображений 1 обычно означает, что пиксел используется в представлении, а 0 — не используется. Для цветных изображений применяются дополнительные биты в качестве индексов в таблице цветов.

Литеру можно вставить в буфер кадра, указав адрес $(x_0, y_0)$ начала маски в буфере. Каждый пиксел в маске смещается на величины $x_0$, $y_0$. Простой алгоритм для бинарной маски приводится ниже.

***Вставка маски** в буфер кадра*

Xmin, Xmax, Ymin, Ymax — *пределы маски*
$x_0$, $y_0$ — *адрес в буфере кадра*

```
for j = Ymin to Ymax − 1
    for i = Xmin to Xmax − 1
```

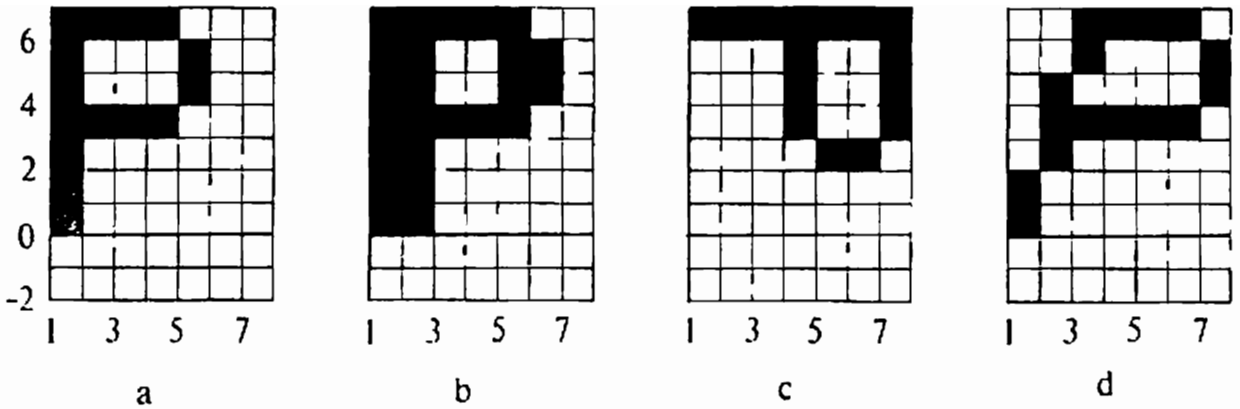



Рис. 2.31. Трансформированные маски литер.

```
        if Маска(i, j) < > 0 then
           записать Маска(i, j) в буфер кадра в (x0 + i, y0 + j)
        else
        end if
    next i
next j
finish
```

Стереть литеру в буфере кадра можно, перезаписав ее с фоновой интенсивностью или цветом.

Для создания литер различных шрифтов или ориентаций перед записью в буфер маску можно модифицировать. Некоторые из таких простых модификаций показаны на рис. 2.31. На рис. 2.31,a изображена исходная литерная маска. Записывая ее в две последовательные ячейки $x_0$ и $x_0 + 1$, получим жирную литеру (рис. 2.31,b). Литеру можно повернуть (рис. 2.31,c) или наклонить, в результате последней операции получим курсив (рис. 2.31,d).

## 2.15. РАСТРОВАЯ РАЗВЕРТКА СПЛОШНЫХ ОБЛАСТЕЙ

Для сих пор речь шла о представлении на растровом графическом устройстве отрезков прямых линий. Однако одной из уникальных характеристик такого устройства является возможность представления сплошных областей. Генерацию сплошных областей из простых описаний ребер или вершин будем называть растровой разверткой сплошных областей, заполнением многоугольников или заполнением контуров. Для этого можно использовать несколько методов, которые обычно делятся на две широкие категории: растровая развертка и затравочное заполнение.

В методах растровой развертки пытаются определить в порядке

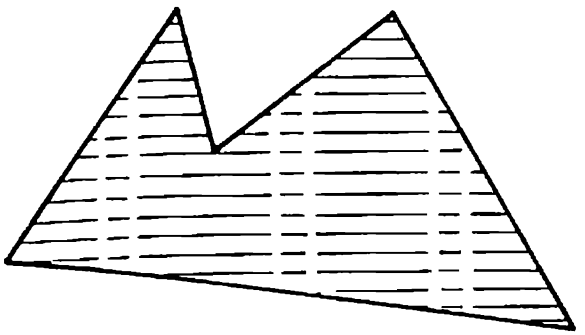


Рис. 2.32. Штриховка или закраска контура.

сканирования строк, лежит ли точка внутри многоугольника или контура. Эти алгоритмы обычно идут от «верха» многоугольника или контура к «низу». Методы развертки также применимы и к векторным дисплеям, в которых они используются для штриховки или закраски контуров, как показано на рис. 2.32.

В методах затравочного заполнения предполагается, что известна некоторая точка (затравка) внутри замкнутого контура. В алгоритмах ищут точки, соседние с затравочной и расположенные внутри контура. Если соседняя точка расположена не внутри, значит, обнаружена граница контура. Если же точка оказалась внутри контура, то она становится новой затравочной точкой и поиск продолжается рекурсивно. Подобные алгоритмы применимы только к растровым устройствам.

## 2.16. ЗАПОЛНЕНИЕ МНОГОУГОЛЬНИКОВ

Многие замкнутые контуры являются простыми многоугольниками. Если контур состоит из кривых линий, то его можно аппроксимировать подходящим многоугольником или многоугольниками. Простейший метод заполнения многоугольника состоит в проверке

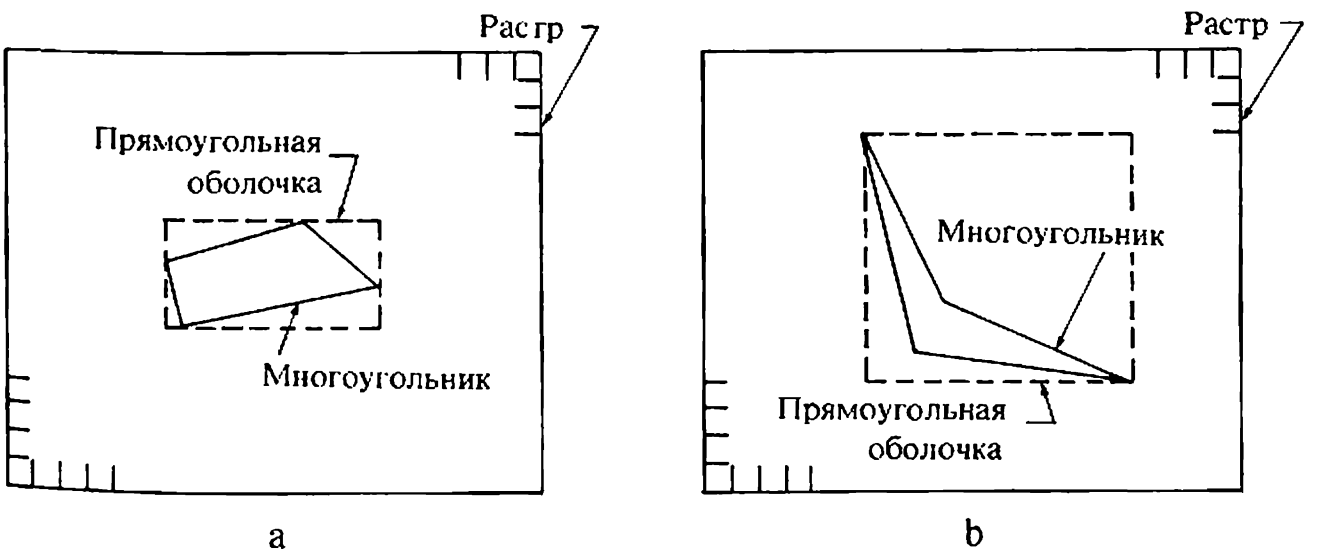



Рис. 2.33. Прямоугольная оболочка многоугольника.

на принадлежность внутренности многоугольника каждого пиксела в растре. Так как обычно большинство пикселов лежит вне многоугольника, то данный метод слишком расточителен. Затраты можно уменьшить путем вычисления для многоугольника прямоугольной оболочки — наименьшего прямоугольника, содержащего внутри себя многоугольник. Как показано на рис. 2.33, проверяются только внутренние точки этой оболочки. Использование прямоугольной оболочки для многоугольника, изображенного на рис. 2.33, a, намного сокращает число проверяемых пикселов. В то же время для многоугольника, представленного на рис. 2.33, b, сокращение существенно меньше.

## 2.17. РАСТРОВАЯ РАЗВЕРТКА МНОГОУГОЛЬНИКОВ

Можно разработать более эффективный метод, чем тест на принадлежность внутренней части, если воспользоваться тем фактом, что соседние пикселы, вероятно, имеют одинаковые характеристики (кроме пикселов граничных ребер). Это свойство называется пространственной когерентностью. Для растровых графических устройств соседние пикселы на сканирующей строке, вероятно, имеют одинаковые характеристики. Это когерентность растровых строк.

Характеристики пикселов на данной строке изменяются только там, где ребро многоугольника пересекает строку. Эти пересечения делят сканирующую строку на области.

Для простого многоугольника на рис. 2.34 строка 2 пересекает многоугольник при $x = 1$ и $x = 8$. Получаем три области:

| | |
|---|---|
| $x < 1$ | вне многоугольника |
| $1 \leqslant x \leqslant 8$ | внутри многоугольника |
| $x > 8$ | вне многоугольника |

Строка 4 делится на пять областей:

| | |
|---|---|
| $x < 1$ | вне многоугольника |
| $1 \leqslant x \leqslant 4$ | внутри многоугольника |
| $4 < x < 6$ | вне многоугольника |
| $6 \leqslant x \leqslant 8$ | внутри многоугольника |
| $x > 8$ | вне многоугольника |

Совсем необязательно, чтобы точки пересечения для строки 4 сразу определялись в фиксированном порядке (слева направо). Например, если многоугольник задается списком вершин $P_1P_2P_3P_4P_5$,

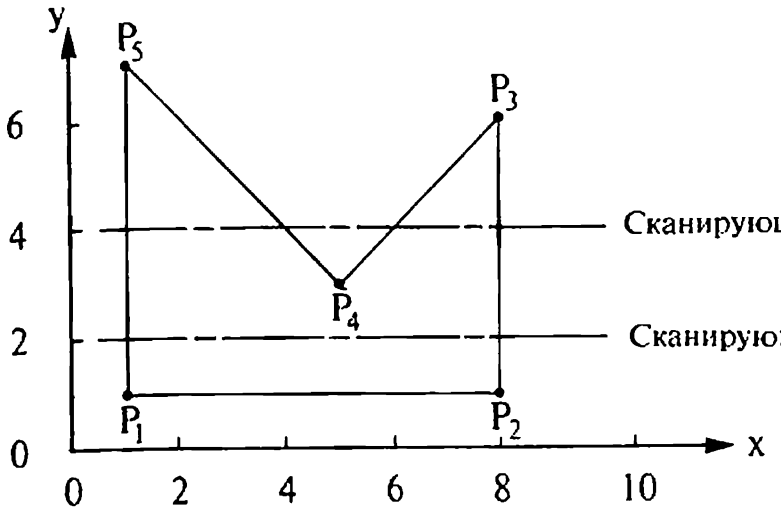



Рис. 2.34. Растровая развертка сплошной области.

а список ребер — последовательными парами вершин $P_1P_2$, $P_2P_3$, $P_3P_4$, $P_4P_5$, $P_5P_1$, то для строки 4 будут найдены следующие точки пересечения с ребрами многоугольника: 8, 6, 4, 1. Эти точки надо отсортировать в возрастающем порядке по $x$, т. е. получить 1, 4, 6, 8.

При определении интенсивности, цвета и оттенка пикселов на сканирующей строке рассматриваются пары отсортированных точек пересечений. Для каждого интервала, задаваемого парой пересечений, используется интенсивность или цвет заполняемого многоугольника. Для интервалов между парами пересечений и крайних (от начала строки до первой точки пересечения и от последней точки пересечения до конца строки) используется фоновая интенсивность или цвет. На рис. 2.34 для строки 4 в фоновый цвет установлены пикселы: от 0 до 1, от 4 до 6, от 8 до 10, тогда как пикселы от 1 до 4 и от 5 до 8 окрашены в цвет многоугольника.

Точное определение тех пикселов, которые должны активироваться, требует некоторой осторожности. Рассмотрим простой прямоугольник, изображенный на рис. 2.35. Прямоугольник имеет

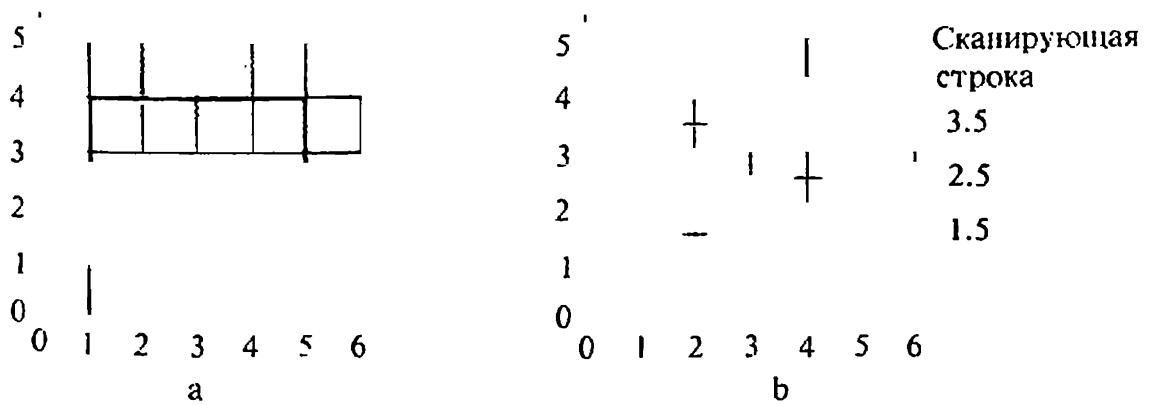



Рис. 2.35. Системы координат строк сканирования.

координаты (1,1) (5,1), (5,4), (1,4). Сканирующие строки с 1 по 4 имеют пересечения с ребрами многоугольника при $x = 1$ и 5. Вспомним, что пиксел адресуется координатами своего левого нижнего угла, значит, для каждой из этих сканирующих строк будут активированы пикселы с $x$-координатами 1, 2, 3, 4 и 5. На рис. 2.35, а показан результат. Заметим, что площадь, покрываемая активированными пикселами, равна 20, в то время как настоящая площадь прямоугольника равна 12.

Модификация системы координат сканирующей строки и теста активации устраняет эту проблему, как это показано на рис. 2.35, b. Считается, что сканирующие строки проходят через центр строк пикселов, т. е. через середину интервала, как это показано на рис. 2.35, b. Тест активации модифицируется следующим образом: проверяется, лежит ли внутри интервала центр пиксела, расположенного справа от пересечения. Однако пикселы все еще адресуются координатами левого нижнего угла. Как показано на рис. 2.35, b, результат данного метода корректен.

Горизонтальные ребра не могут пересекать сканирующую строку и, таким образом, игнорируются. Это совсем не означает, что их нет на рисунке. Эти ребра формируются верхней и нижней строками пикселов, как показано на рис. 2.35. Рис. 2.35 иллюстрирует корректность верхнего и нижнего ребер многоугольника, полученных в результате модификации системы координат сканирующих строк.

Дополнительная трудность возникает при пересечении сканирующей строки и многоугольника точно по вершине, как это показано на рис. 2.36. При использовании соглашения о середине интерва-

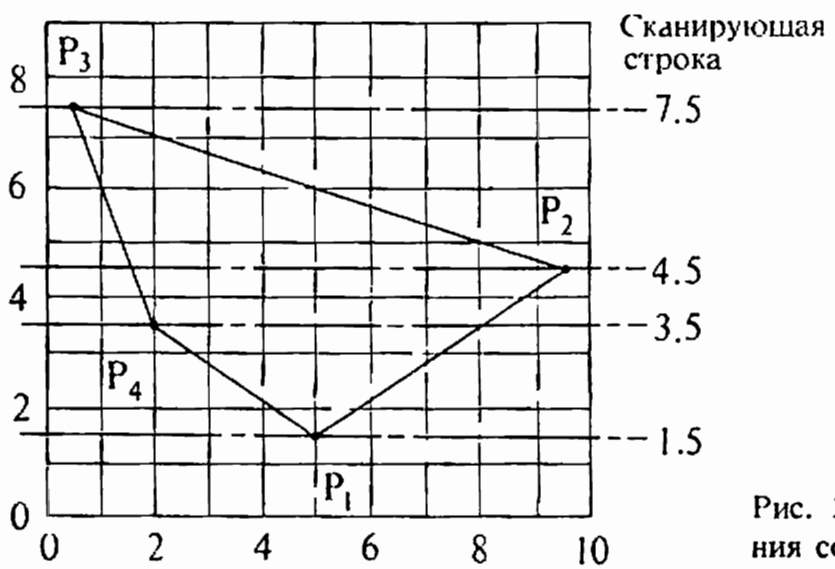



Рис. 2.36. Особенности пересечения со строками сканирования.

ла между сканирующими строками получаем, что строка $y = 3.5$ пересечет многоугольник в 2, 2 и 8, т. е. получится нечетное количество пересечений. Следовательно, разбиение пикселов на пары даст неверный результат, т. е. пикселы (0, 3), (1, 3) и от (3, 3) до (7, 3) будут фоновыми, а пикселы (2, 3), (8, 3), (9, 3) окрасятся в цвет многоугольника. Здесь возникает идея учитывать только одну точку пересечения с вершиной. Тогда для строки $y = 3.5$ получим правильный результат. Однако результат применения метода к строке $y = 1.5$, имеющей два пересечения в (5, 1), показывает, что метод неверен. Для этой строки именно разбиение на пары даст верный результат, т. е. окрашен будет только пиксел (5, 1). Если же учитывать в вершине только одно пересечение, то пикселы от (0, 1) до (4, 1) будут фоновыми, а пикселы от (5, 1) до (9, 1) будут окрашены в цвет многоугольника.

Правильный результат можно получить, учитывая точку пересечения в вершине два раза, если она является точкой локального минимума или максимума и учитывая ее один раз в противном случае. Определить локальный максимум или минимум многоугольника в рассматриваемой вершине можно с помощью проверки концевых точек двух ребер, соединенных в вершине. Если у обоих концов координаты $y$ больше, чем у вершины, значит, вершина является точкой локального минимума. Если меньше, значит, вершина — точка локального максимума. Если одна больше, а другая меньше, следовательно, вершина не является ни точкой локального минимума, ни точкой локального максимума. На рис. 2.36 точка $P_1$ — локальный минимум, $P_3$ — локальный максимум, а $P_2$, $P_4$ — ни то и ни другое. Следовательно, в точках $P_1$ и $P_3$ учитываются два пересечения со сканирующими строками, а в $P_2$ и $P_4$ — одно.

## 2.18. ПРОСТОЙ АЛГОРИТМ С УПОРЯДОЧЕННЫМ СПИСКОМ РЕБЕР

Используя описанные выше методы, можно разработать эффективные алгоритмы растровой развертки сплошных областей, называемые алгоритмами с упорядоченным списком ребер. Они зависят от сортировки в порядке сканирования точек пересечений ребер многоугольника со сканирующими строками. Эффективность этих алгоритмов зависит от эффективности сортировки. Приведем очень простой алгоритм.

Простой алгоритм с упорядоченным списком ребер

Подготовить данные:

Определить для каждого ребра многоугольника точки пересечений со сканирующими строками, проведенными через середины интервалов, для чего можно использовать алгоритм Брезенхема или ЦДА. Горизонтальные ребра игнорируются. Занести каждое пересечение $(x, y + \frac{1}{2})$ в список.

Отсортировать список по строкам и по возрастанию $x$ в строке; т. е. $(x_1, y_1)$ предшествует $(x_2, y_2)$, если $y_1 > y_2$ или $y_1 = y_2$ и $x_1 \leqslant x_2$.

Преобразовать эти данные в растровую форму:

Выделить из отсортированного списка пары элементов $(x_1, y_1)$ и $(x_2, y_2)$. Структура списка гарантирует, что $y = y_1 = y_2$ и $x_1 \leqslant x_2$. Активировать на сканирующей строке $y$ пикселы для целых значений $x$, таких, что

$$x_1 \leqslant x + \frac{1}{2} \leqslant x_2.$$

**Пример 2.8.** Простой упорядоченный список ребер

Рассмотрим многоугольник, изображенный на рис. 2.34. Его вершины: $P_1$ (1,1), $P_2$(8,1), $P_3$(8,6), $P_4$(5,3) и $P_5$(1,7). Пересечения с серединами сканирующих строк следующие:

скан. строка 1.5: (8, 1.5), (1, 1.5)
скан. строка 2.5: (8, 2.5), (1, 2.5)
скан. строка 3.5: (8, 3.5), (5.5, 3.5), (4.5, 3.5), (1, 3.5)
скан. строка 4.5: (8, 4.5), (6.5, 4.5), (3.5, 4.5), (1, 4.5)
скан. строка 5.5: (8, 5.5), (7.5, 5.5), (2.5, 5.5), (1, 5.5)
скан. строка 6.5: (1.5, 6.5), (1, 6.5)
скан. строка 7.5: нет

Весь список сортируется в порядке сканирования сначала сверху вниз и затем — слева направо

(1, 6.5), (1.5, 6.5), (1, 5.5), (2.5, 5.5), (7.5, 5.5), (8, 5.5), (1, 4.5), (3.5, 4.5), (6.5, 4.5), (8, 4.5), (1, 3.5), (4.5, 3.5), (5.5, 3.5), (8, 3.5), (1, 2.5), (8, 2.5), (1, 1.5), (8, 1.5)

Выделяя из списка пары пересечений и применяя алгоритм, описанный выше, получим список пикселов, которые должны быть активированы:

(1, 6)
(1, 5), (2, 5), (7, 5)
(1, 4), (2, 4), (3, 4), (6, 4), (7, 4)
(1, 3), (2, 3), (3, 3), (4, 3), (5, 3), (6, 3), (7, 3)

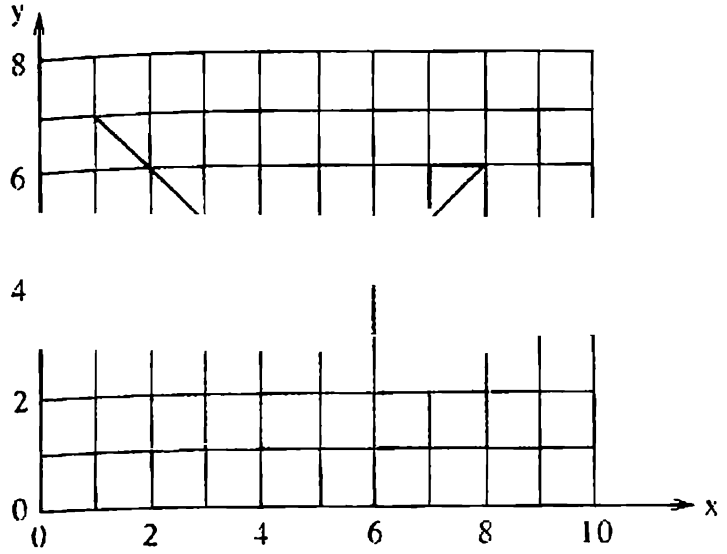



Рис. 2.37. Результат растровой развертки сплошной области, изображенной на рис. 2.34.

(1, 2), (2, 2), (3, 2) (4, 2), (5, 2), (6, 2), (7, 2) (1, 1), (2, 1), (3, 1), (4, 1), (5, 1), (6, 1), (7, 1)

Результат представлен на рис. 2.37. Заметим, что оба вертикальных и нижнее ребра изображены верно.

## 2.19. БОЛЕЕ ЭФФЕКТИВНЫЕ АЛГОРИТМЫ С УПОРЯДОЧЕННЫМ СПИСКОМ РЕБЕР

В алгоритме, описанном в предыдущем разделе, генерируется большой список, который необходимо полностью отсортировать. Алгоритм можно улучшить, если повысить эффективность сортировки. Последнее достигается разделением сортировки по строкам в направлении $y$ и сортировки в строке в направлении $x$ с помощью групповой сортировки по $y$, описанной в разд. 2.8. В частности, алгоритм теперь можно представить в следующем виде:

Более эффективный алгоритм с упорядоченным списком ребер

Подготовить данные:

Определить для каждого ребра многоугольника точки пересечений со сканирующими строками, проведенными через середины интервалов, т. е. через $y + \frac{1}{2}$. Для этого можно использовать алгоритм Брезенхема или ЦДА. Горизонтальные ребра не учитывать. Поместить координату $x$ точки пересечения в группу, соответствующую $y$.

Для каждой $y$-группы отсортировать список координат $x$ точек пересечений в порядке возрастания; т. е. $x_1$ предшествует $x_2$, если $x_1 \leqslant x_2$

Преобразовать эти данные в растровую форму:

Для каждой сканирующей строки выделить из списка коор-

динат $x$ точек пересечений пары точек пересечений. Активировать на сканирующей строке $y$ пикселы для целых значений $x$, таких, что $x_1 \leqslant x + \frac{1}{2} \leqslant x_2$.

В алгоритме сначала с помощью групповой сортировки по $y$ происходит сортировка в порядке сканирования строк, а затем сортировка в строке. Таким образом, развертка начинается до завершения всего процесса сортировки. В таком алгоритме отчасти легче добавлять или удалять информацию из дисплейного списка. Необходимо только добавить или удалить информацию из соответствующих $y$-групп; следовательно, пересортировать придется только затронутые изменением строки. Ниже данный алгоритм иллюстрируется с помощью примера.

**Пример 2.9.** Более эффективный упорядоченный список ребер

Рассмотрим снова многоугольник на рис. 2.34, обсуждавшийся в примере 2.8. Сначала создаются $y$-группы для всех сканирующих строк, как это показано на рис. 2.38. Многоугольник обрабатывается против часовой стрелки, начиная с вершины $P_1$, и получающиеся пересечения, неотсортированные по $x$, заносятся в соответствующие группы, как это показано на рис. 2.38,a. Пересечения были вычислены в соответствии с соглашением о середине интервала между сканирующими строками. В качестве иллюстрации на рис. 2.38,b показаны отсортированные пересечения. На практике можно использовать небольшой буфер сканирующей строки, содержащий отсортированные кординаты $x$ точек пересечения, как это показано на рис. 2.38,c. Такой буфер позволяет более эффективно добавлять или удалять пересечения в список, их просто можно добавлять к концу списка каждой $y$-группы, так как сортировка откладывается до момента помещения строки в буфер. Следовательно, не нужно держать полностью отсортированный список $y$-групп.

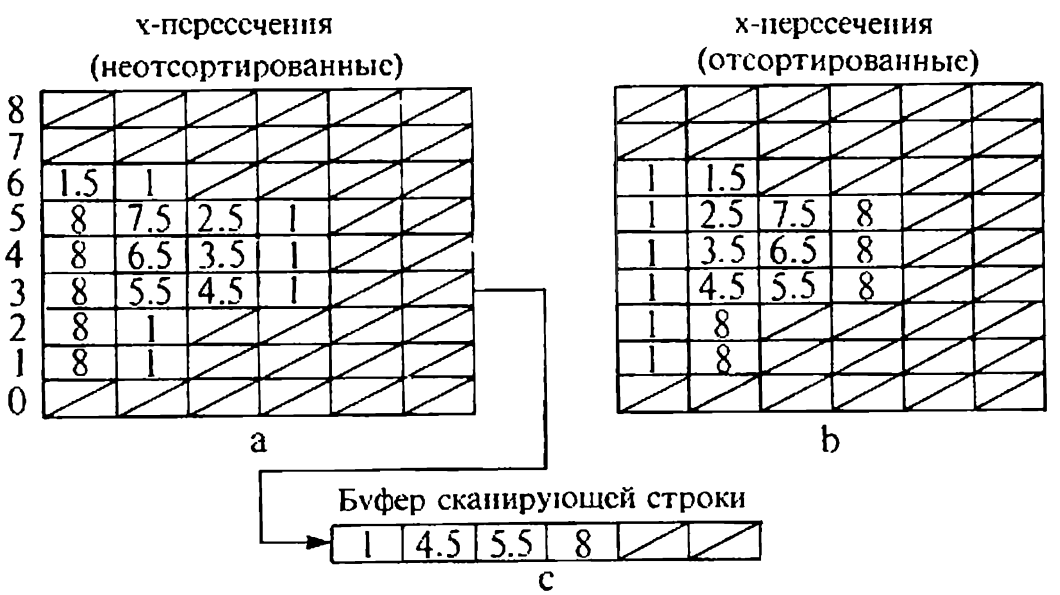



Рис. 2.38. $y$-группы сканирующих строк для многоугольника, изображенного на рис. 2.34.

В результате выделения пар пересечений из отсортированного по $x$ списка и применения алгоритма для каждой сканирующей строки получим список активируемых пикселов. Результат показан на рис. 2.37 и совпадает с результатом примера 2.8.

Хотя в этой версии алгоритма задача сортировки упрощается, в ней либо ограничено число пересечений с данной сканирующей строкой, либо требуется резервирование большого количества памяти, значительная часть которой не будет использоваться. Этот недостаток можно преодолеть благодаря использованию связного списка, т. е. путем введения добавочной структуры данных. Предварительное вычисление пересечения каждой сканирующей строки с каждым ребром многоугольника требует больших вычислительных затрат и значительных объемов памяти. Введя список активных ребер (САР), как это предполагалось для растровой развертки в реальном времени (см. разд. 2.8), можно сократить потребность в памяти и вычислять пересечения со сканирующими строками в пошаговом режиме. Алгоритм, получившийся в результате всех улучшений, выглядит следующим образом:

Алгоритм с упорядоченным списком ребер, использующий список активных ребер

Подготовить данные:

Используя сканирующие строки, проведенные через середины отрезков, т. е. через $y + \frac{1}{2}$, определить для каждого ребра многоугольника наивысшую сканирующую строку, пересекаемую ребром.

Занести ребро многоугольника в $y$-группу, соответствующую этой сканирующей строке.

Сохранить в связном списке значения: начальное значение координат $x$ точек пересечения, $\Delta y$ — число сканирующих строк, пересекаемых ребром многоугольника, и $\Delta x$ — шаг приращения по $x$ при переходе от одной сканирующей строки к другой.

Преобразовать эти данные в растровую форму:

Для каждой сканирующей строки проверить соответствующую $y$-группу на наличие новых ребер. Новые ребра добавить в САР.

Отсортировать координаты $x$ точек пересечения из САР в порядке возрастания; т. е. $x_1$ предшествует $x_2$, если $x_1 \leqslant x_2$.

Выделить пары точек пересечений из отсортированного по $x$ списка. Активировать на сканирующей строке $y$ пикселы для целых значений $x$, таких, что $x_1 \leqslant x + \frac{1}{2} \leqslant x_2$. Для каждого ребра из САР уменьшить $\Delta y$ на 1. Если $\Delta y < 0$, то исключить данное ребро из САР. Вычислить новое значение координат $x$ точек пересечения $x_{нов} = x_{стар} + \Delta x$.

Перейти к следующей сканирующей строке.

В алгоритме предполагается, что все данные предварительно преобразованы в представление, принятое для многоугольников. Уиттед в [2-21] приводит более общий алгоритм, в котором данное ограничение устранено.

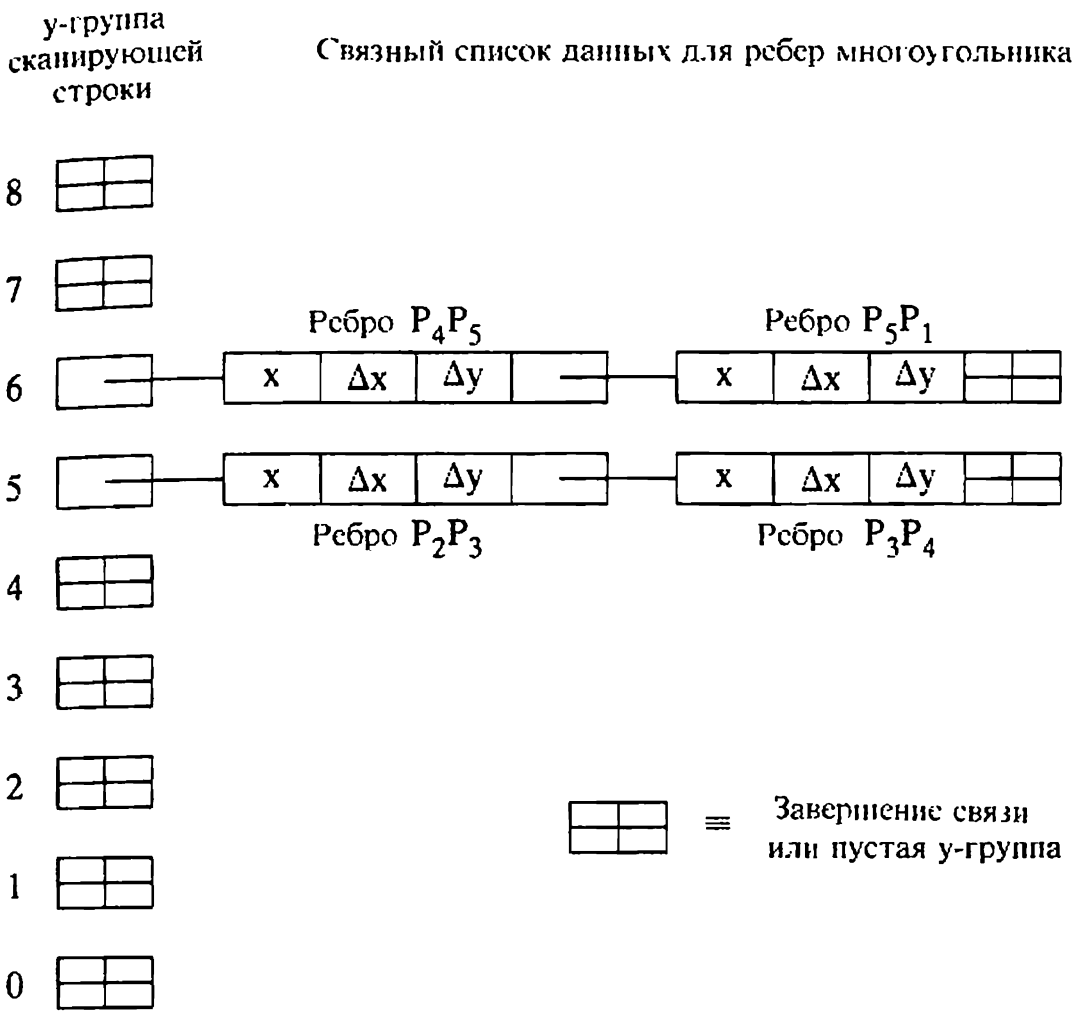
**Пример 2.10.** Упорядоченный список ребер вместе с САР

Опять рассмотрим простой многоугольник на рис. 2.34. Проверка списка ребер многоугольника показывает, что сканирующая строка 5 — наивысшая для ребер $P_2P_3$ и $P_3P_4$, а строка 6 — для ребер $P_4P_5$ и $P_5P_1$. На рис. 2.39,a схематично показана структура связанного списка, содержащая данные для девяти $y$-групп, соответствующих девяти (с 0 по 8) сканирующим строкам рис. 2.34. Заметим, что большинство $y$-групп пусто. На рис. 2.39,b показан подход, применяемый на практике. Здесь список $y$-групп — это одномерный массив, по одному элементу на каждую сканирующую строку. В этом элементе содержится только указатель на массив данных, используемый для связанного списка. Массив показан на том же рисунке.

Связный список реализован как массив размерности $n \times 4$. Для каждого индекса массива $n$ четыре элемента содержат: значение координаты $x$ точки пересечения ребра многоугольника с наивысшей строкой, пересекаемой этим ребром; шаг приращения по $x$ при переходе от одной сканирующей строки к другой; число сканирующих строк, пересекаемых ребром многоугольника; указатель на расположение данных для следующего ребра, начинающегося на этой же сканирующей строке. Это продемонстрировано на рис. 2.39,b. Заметим, что $y$-группа для строки 5 содержит указатель 1, соответствующий первому адресу в связном списке данных. Первые три колонки данных в списке содержат информацию о ребре $P_2P_3$. Число в четвертой колонке — это указатель на следующий элемент данных.

Список активных ребер реализован в виде стекового массива размерности $n \times 3$. Содержимое САР для всех девяти сканирующих строк показано на рис. 2.39,c. Сканирующие строки ($y$-группы) последовательно проверяются сверху вниз, начиная со строки 8. Так как $y$-группы для строк 8 и 7 пусты, то САР также пуст. Строка 6 добавляет в САР два элемента и строка 5 — еще два. На строке 2 значение $\Delta y$ для ребер $P_3P_4$ и $P_4P_5$ становится меньше нуля. Следовательно, эти ребра удаляются из САР. Аналогичным образом ребра $P_2P_3$ и $P_5P_1$ удаляются на строке 0. Наконец, заметим, что для строки 0 $y$-группа пуста, список активных ребер пуст и больше $y$-групп нет. Значит, работа алгоритма завершена.

Для каждой сканирующей строки выделяются координаты $x$ точек пересечения активных ребер для этой строки, сортируются по возрастанию $x$ и записываются в интервальный буфер, реализованный в виде массива $1 \times n$. Из этого буфера пересечения выделяются в пары, и затем, используя описанный алгоритм, определяется список активных пикселов. Объединенный для всех сканирующих строк список активных пикселов совпадает со списком из предыдущего примера. Результат снова представлен на рис. 2.37.




a

b

Рис. 2.39. Схема связного списка для многоугольника, изображенного на рис. 2.34.

| Сканирующая строка | Список активных ребер x | Δx | Δy | Отсортированные по x пересечения | Список пикселов |
|---|---|---|---|---|---|
| 8 | | | | | |
| 7 | | | | | |
| 6 | 1.5 | 1 | 3 | 1 1.5 | (1,6) |
| | 1 | 0 | 5 | | |
| 5 | 2.5 | 1 | 2 | 1 2.5 7.5 8 | (1,5), (2,5), (7,5) |
| | 1 | 0 | 4 | | |
| | 8 | 0 | 4 | | |
| | 7.5 | −1 | 2 | | |
| 4 | 3.5 | 1 | 1 | 1 3.5 6.5 8 | (1,4), (2,4), (3,4), (6,4), (7,4) |
| | 1 | 0 | 3 | | |
| | 8 | 0 | 3 | | |
| | 6.5 | −1 | 1 | | |
| 3 | 4.5 | 1 | 0 | 1 4.5 5.5 8 | (1,3), (2,3), (3,3), (4,3), (5,3), (6,3), (7,3) |
| | 1 | 0 | 2 | | |
| | 8 | 0 | 2 | | |
| | 5.5 | −1 | 0 | | |
| 2 | 1 | 0 | 1 | 1 8 | (1,2), (2,2), (3,2), (4,2), (5,2), (6,2), (7,2) |
| | 8 | 0 | 1 | | |
| 1 | 1 | 0 | 0 | 1 8 | (1,1), (2,1), (3,1), (4,1), (5,1), (6,1), (7,1) |
| | 8 | 0 | 0 | | |
| 0 | | | | | |

Рис. 2.39. Продолжение.

## 2.20. АЛГОРИТМ ЗАПОЛНЕНИЯ ПО РЕБРАМ

Алгоритм с упорядоченным списком ребер очень мощный. Каждый пиксел изображения активируется только один раз, следовательно, минимизированы операции ввода/вывода. До вывода вычисляются концевые точки каждой группы или интервалы активных пикселов. Это позволяет использовать алгоритмы закраски вдоль этих участков для получения полностью раскрашенных изображений. Так как этот алгоритм не зависит от деталей ввода/вывода, то можно его сделать не зависящим от устройства. Главный недостаток алгоритма состоит в больших накладных расходах, связанных с поддержкой и сортировкой различных списков. В другом методе растровой развертки сплошных областей большинство из этих списков устранено. Этот метод называется алгоритмом заполнения по ребрам [2-22].

Описываемый ниже алгоритм очень прост.

Алгоритм заполнения по ребрам

> Для каждой сканирующей строки, пересекающей ребро многоугольника в $(x_1, y_1)$, дополнить все пикселы, у которых центры лежат справа от $(x_1, y_1)$, т. е. для $(x, y_1)$, $x + \frac{1}{2} > x_1$.

Вычисление пересечений сканирующих строк с ребрами производится в соответствии с соглашением о середине интервала между сканирующими строками. К каждому ребру алгоритм применяется индивидуально, причем порядок обработки ребер многоугольника не важен. На рис. 2.40 продемонстрированы различные стадии растровой развертки сплошной области тестового многоугольника с рис. 2.34. Заметим, что в этом случае результат не совпадает с результатом для упорядоченного списка ребер. В частности, в данном алгоритме не активируются пикселы (5, 3), (6, 4), (7, 5); т. е. ребро $P_3P_4$ обработано иначе. Отличие возникает для тех пикселов, которые разделены строго пополам: половина находится внутри, а половина — вне многоугольника. В алгоритме с упорядоченным списком ребер эти пикселы всегда активируются, а в данном алгоритме они активируются только в том случае, если внутренняя часть многоугольника лежит слева от центра пиксела.

Наиболее удобно использовать описываемый алгоритм вместе с буфером кадра, что позволяет обрабатывать ребра многоугольника в совершенно произвольном порядке. При обработке каждого реб-

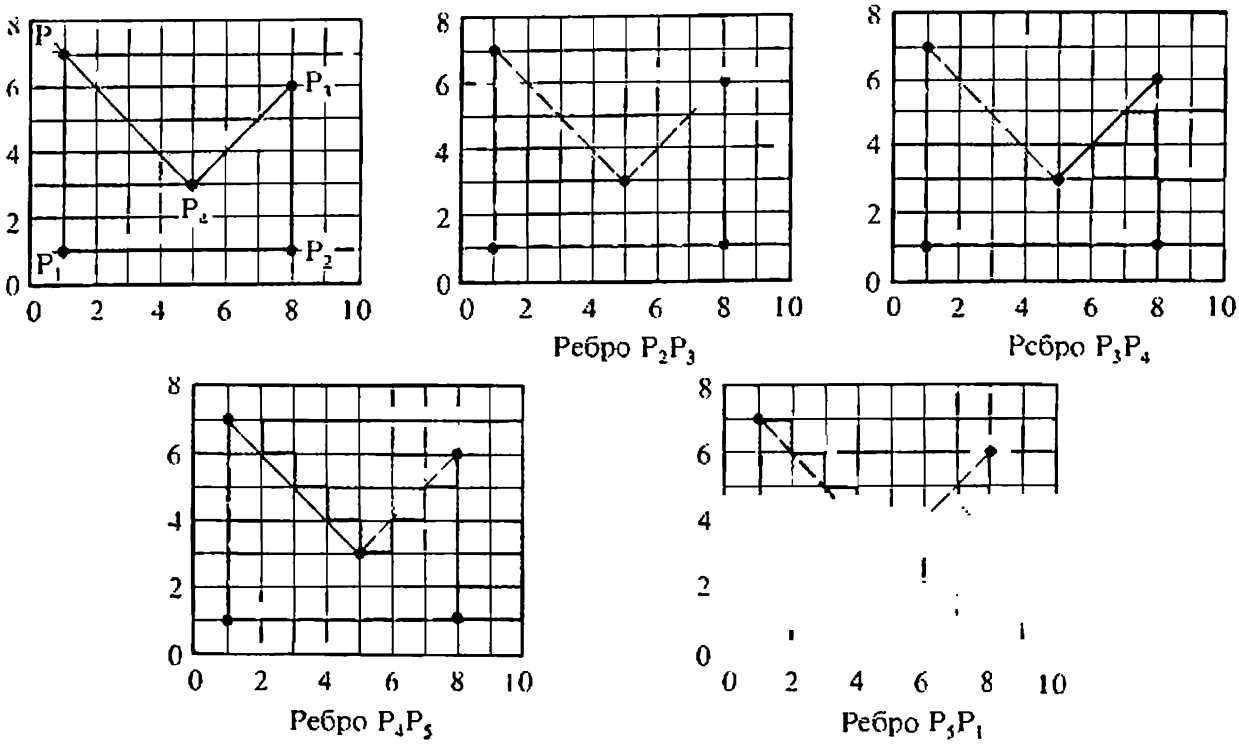



Рис. 2.40. Алгоритм заполнения по ребрам.

ра обрабатываются пикселы в буфере кадра, соответствующие пересечению ребра со сканирующей строкой. После завершения обработки всех ребер буфер кадра выводится в порядке сканирования на дисплей. На рис. 2.40 проиллюстрированы главные недостатки алгоритма — для сложного изображения каждый пиксел может обрабатываться много раз. Следовательно, эффективность алгоритма ограничена скоростью ввода/вывода.

Число обрабатываемых в данном алгоритме пикселов можно сократить, если ввести так называемую перегородку [2-23]. Основная идея алгоритма заполнения с перегородкой проиллюстрирована на рис. 2.41 для тестового многоугольника с рис. 2.34. Алгоритм можно описать следующим образом:

Алгоритм заполнения с перегородкой

Для каждой сканирующей строки, пересекающей ребро многоугольника:

Если пересечение находится слева от перегородки, то дополнить все пикселы, центры которых лежат справа от пересечения сканирующей строки с ребром и слева от перегородки.

Если пересечение находится справа от перегородки, то дополнить все пикселы, центры которых расположены слева

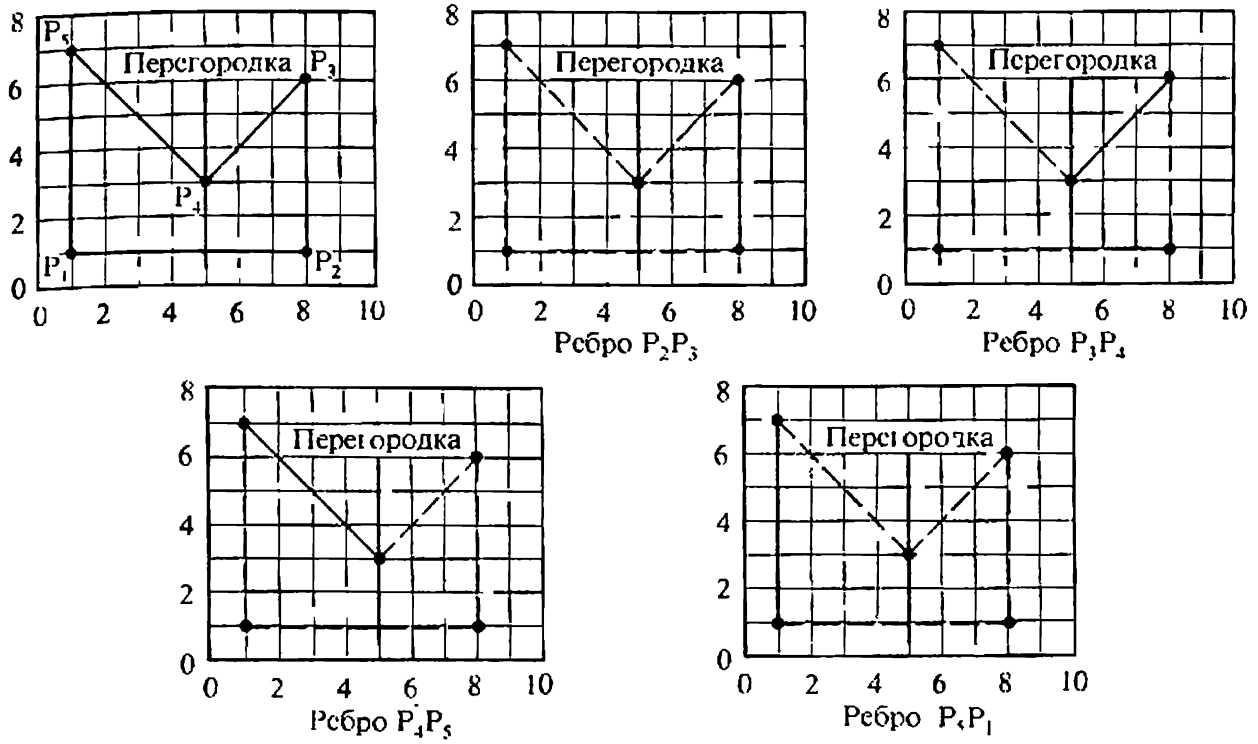



Рис. 2.41. Алгоритм заполнения с перегородкой.

или на пересечении сканирующей строки с ребром и справа от перегородки.

Используется соглашение о середине интервала между сканирующими строками. Обычно перегородка проходит через одну из вершин многоугольника, и снова удобнее всего применять данный алгоритм с буфером кадра. Недостаток как алгоритма заполнения по ребрам, так и алгоритма заполнения с перегородкой заключается в неоднократном активировании части пикселов. Преодолеть его позволяет модификация алгоритма, известная как алгоритм со списком ребер и флагом [2-24]. Применение всех этих трех алгоритмов не ограничивается только простыми многоугольниками.

## 2.21. АЛГОРИТМ СО СПИСКОМ РЕБЕР И ФЛАГОМ

Алгоритм, использующий список ребер и флаг [2-24], является двухшаговым. Первый шаг состоит в обрисовке контура, в результате чего на каждой сканирующей строке образуются пары ограничивающих пикселов. Второй шаг состоит в заполнении пикселов, расположенных между ограничивающими. Более точно алгоритм можно сформулировать в следующем виде:

Алгоритм со списком ребер и флагом

Обрисовка контура:

Используя соглашение о середине интервала между сканирующими строками для каждого ребра, пересекающего сканирующую строку, отметить самый левый пиксел, центр которого лежит справа от пересечения; т. е. $x + 1/2 > x_{\text{пересечения}}$

Заполнение:

Для каждой сканирующей строки, пересекающей многоугольник

```
Внутри = FALSE
for x = 0 (левая граница) to x = xmax (правая граница)
   if пиксел в точке x имеет граничное значение
   then инвертировать значение переменной Внутри
   if Внутри = TRUE then
      присвоить пикселу в x значение цвета многоугольника
   else
      присвоить пикселу в x значение цвета фона
   end if
next x
```

**Пример 2.11.** Алгоритм, использующий список ребер и флаг

Рассотрим применение данного алгоритма к тестовому многоугольнику на рис. 2.34. Сначала обрисовывается контур, результат этого шага показан на рис. 2.42,a. Активированы пикселы в точках (1, 1), (1, 2), (1, 3), (1, 4), (1, 5), (1, 6), (2, 6), (3, 5), (4, 4), (5, 3), (6, 3), (7, 4), (8, 4), (8, 3), (8, 2), (8, 1).

Затем многоугольник заполняется. Для иллюстрации этого процесса выделена и показана строка 3 на рис. 2.42,b. Для обрисовки контура на этой строке активированы пикселы при $x = 1, 5, 6$ и 8. Применение алгоритма заполнения дает следующие

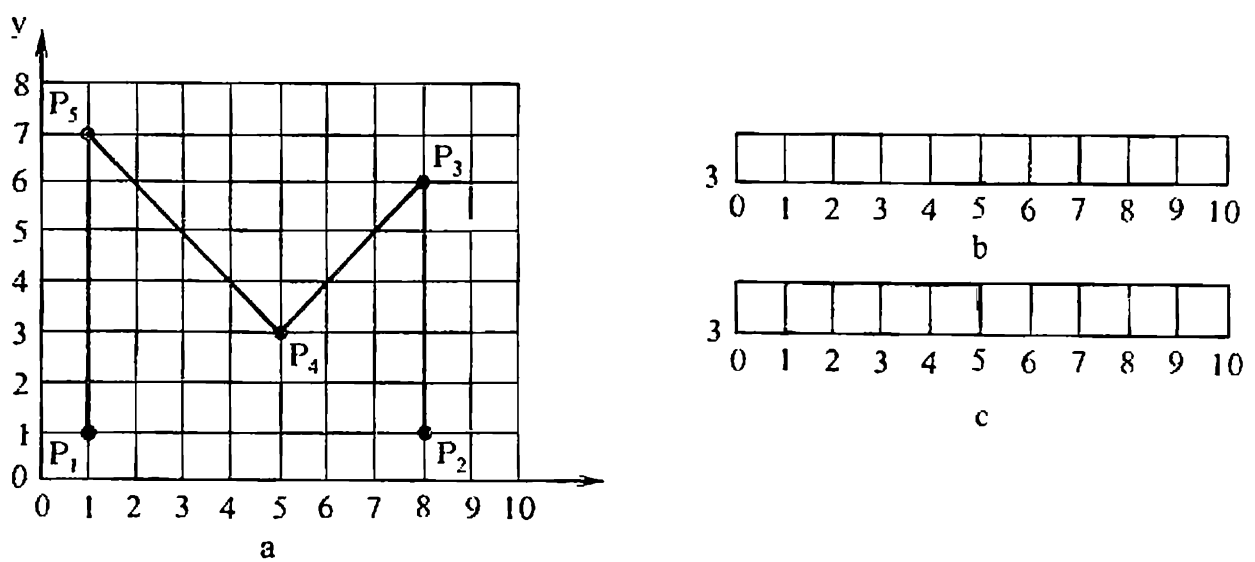



Рис. 2.42. Алгоритм заполнения по ребрам и флагу.

результаты:

| | |
|---|---|
| Сначала<br>Внутри = FALSE | |
| Для $x = 0$ | Пиксел — не граничный и Внутри = FALSE, следовательно, предпринимать никаких действий не надо |
| Для $x = 1$ | Пиксел — граничный, поэтому переменная Внутри инвертируется и получает значение TRUE. Переменная Внутри = TRUE, поэтому пикселу присваивается цвет или интенсивность многоугольника. |
| Для $x = 2, 3, 4$ | Пиксел — не граничный и Внутри = TRUE, поэтому пикселу присваивается цвет многоугольника. |
| Для $x = 5$ | Пиксел — граничный, поэтому переменная Внутри инвертируется. Внутри = FALSE, поэтому пикселу присваивается фоновый цвет. |
| Для $x = 6$ | Пиксел — граничный, поэтому переменная Внутри инвертируется и получает значение TRUE. Переменная Внутри = TRUE, поэтому пикселу присваивается цвет многоугольника. |
| Для $x = 7$ | Пиксел — не граничный и Внутри = TRUE, поэтому пикселу присваивается цвет многоугольника. |
| Для $x = 8$ | Пиксел — граничный, поэтому переменная Внутри инвертируется и получает значение FALSE. Переменная Внутри = FALSE, поэтому пикселу присваивается цвет фона. |

Результат представлен на рис. 2.42,с, а конечный результат для всего многоугольника совпадает с результатом работы алгоритма со списком ребер, показанным на рис. 2.40.

В данном алгоритме каждый пиксел обрабатывается только один раз, так что затраты на ввод/вывод значительно меньше, чем в алгоритме со списком ребер или алгоритме с перегородкой. Ни один из этих алгоритмов, если они работают с буфером кадра, не требует построения, поддержки и сортировки каких-либо списков. При программной реализации алгоритм с упорядоченным списком ребер и алгоритм со списком ребер и флагом работают примерно с одинаковой скоростью [2-21]. Однако алгоритм со списком ребер и флагом годится для аппаратной или микропрограммной реализации, в результате чего он выполняется на один-два порядка быстрее, чем алгоритм с упорядоченным списком ребер [2-24]. Для простых изображений даже возможна анимация в реальном времени.

## 2.22. АЛГОРИТМЫ ЗАПОЛНЕНИЯ С ЗАТРАВКОЙ

В обсуждавшихся выше алгоритмах заполнение происходит в порядке сканирования. Иной подход используется в алгоритмах заполнения с затравкой. В них предполагается, что известен хотя бы один пиксел из внутренней области многоугольника. Алгоритм пытается найти и закрасить все другие пикселы, принадлежащие внутренней области. Области могут быть либо внутренне-, либо гранично-определенными. Если область относится к внутренне-определенным, то все пикселы, принадлежащие внутренней части, имеют один и тот же цвет или интенсивность, а все пикселы, внешние по отношению к области, имеют другой цвет. Это продемонстрировано на рис. 2.43. Если область относится к гранично-определенным, то все пикселы на границе области имеют выделенное значение или цвет, как это показано на рис. 2.44. Ни один из пикселов из внутренней части такой области не может иметь это выделенное значение. Тем не менее пикселы, внешние по отношению к границе, также могут иметь граничное значение. Алгоритмы, заполняющие внутренне-определенные области, называются внутренне-заполняющими, а алгоритмы для гранично-определенных областей — гранично-заполняющими. Далее будут обсуждаться гранично-заполняющие алгоритмы, однако соответствующие внутренне-заполняющие алгоритмы можно получить аналогичным образом.

Внутренне- или гранично-определенные области могут быть 4- или 8-связными. Если область 4-связная, то любого пиксела в области можно достичь с помощью комбинации движений только в 4 направлениях: налево, направо, вверх, вниз. Для 8-связной области пиксела можно достичь с помощью комбинации движений в двух горизонтальных, двух вертикальных и 4 диагональных направлени-

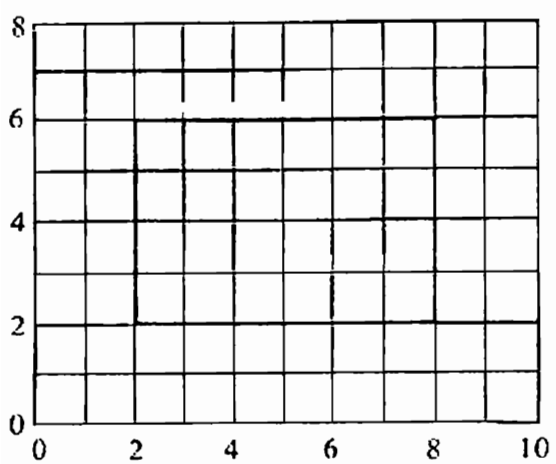



Рис. 2.43. Внутренне-определенная область.

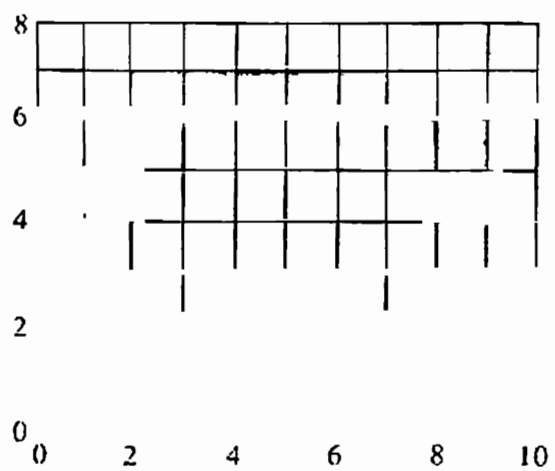



Рис. 2.44. Гранично-определенная область

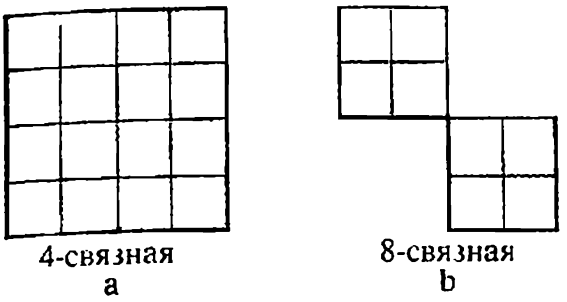



Рис. 2.45. 4- и 8-связные внутренне-определенные области.

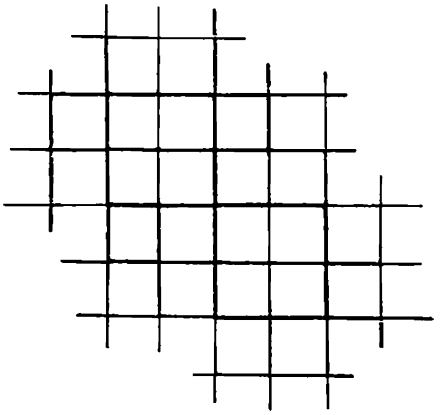


Рис. 2.46. 4- и 8-связные гранично-определенные области.

ях. Алгоритм заполнения 8-связной области заполнит и 4-связную область, однако обратное неверно. На рис. 2.45 показаны простые примеры 4- и 8-связных внутренне-определенных областей. Хотя каждая из подобластей 8-связной области на рис. 2.45,b является 4-связной, для перехода из одной подобласти в другую требуется 8-связный алгоритм. Однако в ситуации, когда надо заполнить разными цветами две отдельные 4-связные подобласти, использование 8-связного алгоритма вызовет неправильное заполнение обеих областей одним и тем же цветом.

На рис. 2.46 показана 8-связная область с рис. 2.45, переопределенная в виде гранично-определенной области. На рис.2.46 иллюстрируется тот факт, что для 8-связной области, у которой две подобласти соприкасаются углами, граница 4-связна, а граница 4-связной области 8-связна. Далее речь в основном пойдет об алгоритмах для 4-связных областей, однако их можно легко переделать для 8-связных областей, если заполнение проводить не в 4, а в 8 направлениях.

## 2.23. ПРОСТОЙ АЛГОРИТМ ЗАПОЛНЕНИЯ С ЗАТРАВКОЙ

Используя стек, можно разработать простой алгоритм заполнения гранично-определенной области. Стек — это просто массив или другая структура данных, в которую можно последовательно помещать значения и из которой их можно последовательно извлекать. Когда новые значения добавляются или помещаются в стек, все остальные значения опускаются вниз на один уровень. Когда значения удаляются или извлекаются из стека, остальные значения всплывают или поднимаются вверх на один уровень. Такой стек называется стеком прямого действия или стеком с дисциплиной обслуживания «первым пришел, последним обслужен» (FILO). Про-

стой алгоритм заполнения с затравкой можно представить в следующем виде:

Простой алгоритм заполнения с затравкой и стеком

Поместить затравочный пиксел в стек
Пока стек не пуст

Извлечь пиксел из стека
Присвоить пикселу требуемое значение
Для каждого из соседних к текущему 4-связных пикселов проверить: является ли он граничным пикселом или не присвоено ли уже пикселу требуемое значение. Проигнорировать пиксел в любом из этих двух случаев. В противном случае поместить пиксел в стек.

Алгоритм можно модифицировать для 8-связных областей, если просматривать 8-связные пикселы, а не только 4-связные. Приведем более формальное изложение алгоритма, в котором предполагается существование затравочного пиксела и гранично-определенной области

***Простой алгоритм заполнения***

```
Затравка(x, y) выдает затравочный пиксел
Push — процедура, которая помещает пиксел в стек
Pop — процедура, которая извлекает пиксел из стека
  Пиксел(x, y) = Затравка(x, y)
  инициализируем стек
  Push Пиксел(x, y)
  while (стек не пуст)
    извлекаем пиксел из стека
    Pop Пиксел(x, y)
    if Пиксел(x, y) < > Нов_значение then
      Пиксел(x, y) = Нов_значение
    end if
      проверим, надо ли помещать соседние пикселы в стек
      if (Пиксел(x + 1, y) < > Нов_значение and
        Пиксел(x + 1, y) < > Гран_значение) then
        Push Пиксел (x + 1, y)
      if (Пиксел(x, y + 1) < > Нов_значение and
        Пиксел(x, y + 1) < > Гран_значение) then
        Push Пиксел (x, y + 1)
      if (Пиксел(x - 1, y) < > Нов_значение and
        Пиксел(x - 1, y) < > Гран_значение) then
```

```
        Push Пиксел (x − 1, y)
    if (Пиксел(x, y − 1) < > Нов_значение and
        Пиксел(x, y − 1) < > Гран_значение) then
        Push Пиксел (x, y − 1)
    end if
end while
```

В алгоритме проверяются и помещаются в стек 4-связные пикселы, начиная с правого от текущего пиксела. Направление обхода пикселов — против часовой стрелки.

**Пример 2.12.** Простой алгоритм заполнения с затравкой

В качестве примера применения алгоритма рассмотрим гранично-определенную полигональную область, заданную вершинами (1, 0), (7, 0), (8, 1), (8, 4), (6, 6), (1, 6), (0, 5) и (0, 1). Область изображена на рис. 2.47. Затравочный пиксел — (4, 3). Область заполняется пиксел за пикселом в порядке, указанном на рис. 2.47 линией со стрелками. Числа, изображенные на пикселе, обозначают занимаемую при работе алгоритма позицию пиксела в стеке. Для некоторых пикселов таких чисел несколько. Это означает, что пиксел помещали в стек более одного раза. Когда алгоритм доходит до пиксела (5, 5), стек насчитывает 25 уровней глубины и содержит пикселы (7, 4), (7, 3), (7, 2), (7, 1), (6, 2), (6, 3), (5, 5), (6, 4), (5, 5), (4, 4), (3, 3), (3, 4), (3, 5), (2, 4), (2, 3), (2, 2), (2, 2), (3, 2), (5, 1), (3, 2), (5, 2), (3, 3), (4, 4), (5, 3).

Так как все пикселы, окружающие (5, 5), содержат либо граничные, либо новые значения цвета, то ни один из них в стек не помещается. Следовательно, из стека извлекается пиксел (7, 4) и алгоритм продолжает заполнять колонку (7, 4), (7, 3), (7, 2), (7, 1). При достижении пиксела (7, 1) проверка снова показывает, что окружающие пикселы либо уже заполнены, либо являются граничными пикселами. Так как в этот момент многоугольник полностью заполнен, то извлечение пикселов из стека до полного его опустошения не вызывает появления дополнительных пикселов, которые следует заполнить. Когда стек становится пустым, алгоритм завершает работу.

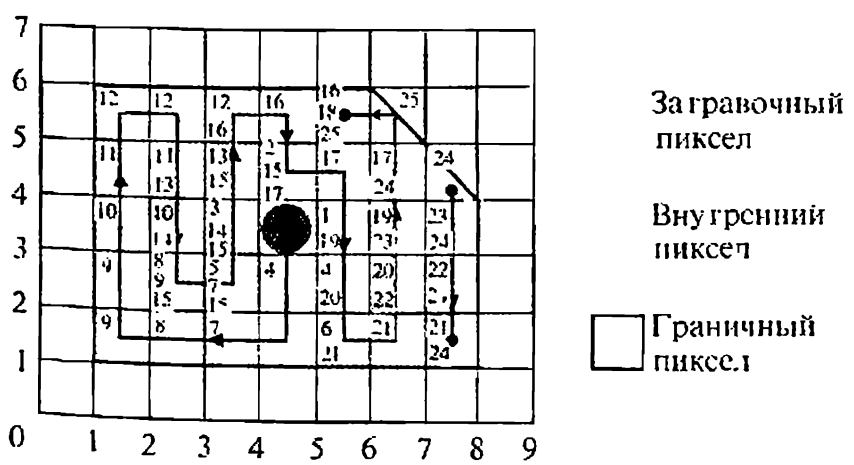



Рис. 2.47. Затравочное заполнение с помощью простого стекового алгоритма.

Многоугольник из примера 2.12 является односвязной областью, но алгоритм будет правильно заполнять и области, содержащие дыры. Это показно в следующем примере.

**Пример 2.13.** Алгоритм заполнения с затравкой для многоугольника с дыркой

В качестве примера применения алгоритма рассмотрим гранично-определенную область, содержащую дыру. Она изображена на рис. 2.48. Как и в предыдущем примере, вершины многоугольника заданы пикселами (1, 0), (7, 0), (8, 1), (8, 4), (6, 6), (1, 6), (0, 5) и (0, 1).. Внутренняя дыра определяется пикселами (3, 2), (5, 2), (5, 3), (3, 3). Затравочный пиксел — (4, 4). Из-за дыры многоугольник заполняется не в том порядке, как в примере 2.12. Новый порядок заполнения указан на рис. 2.48 линией со стрелками. Как и в предыдущем примере, числа в квадратике пиксела показывают позицию в стеке, занимаемую пикселом. Когда обработка доходит до пиксела (3, 1), все окружающие его 4-связные пикселы либо уже заполнены, либо являются граничными. Поэтому ни один из пикселов не помещается в стек. Глубина стека в этот момент равна 15. В стеке находятся пикселы (7, 1), (7, 2), (7, 3), (6, 5), (7, 4), (6, 5), (3, 1), (1, 2), (1, 3), (1, 4), (2, 5), (3, 5), (4, 5), (5, 4).

После удаления из стека пиксела (7, 1) заполняется колонка (7, 1), (7, 2), (7, 3), (7, 4), при этом ни один новый пиксел в стек не добавляется. Для пиксела (7, 4) снова все 4-связные окружающие пикселы либо уже заполнены, либо являются граничными. Обращаясь к стеку, алгоритм извлекает пиксел (6, 5), его заполнение завершает заполнение всего многоугольника. Дальнейшая обработка происходит без какого-либо заполнения, и когда стек становится пустым, алгоритм завершает работу.

## 2.24. ПОСТРОЧНЫЙ АЛГОРИТМ ЗАПОЛНЕНИЯ С ЗАТРАВКОЙ

Как видно из обоих примеров, стек может стать довольно большим. Еще один недостаток предыдущего алгоритма — стек зачастую содержит дублирующую или ненужную информацию. В построчном алгоритме заполнения с затравкой размер стека минимизируется за счет хранения только одного затравочного пиксела для

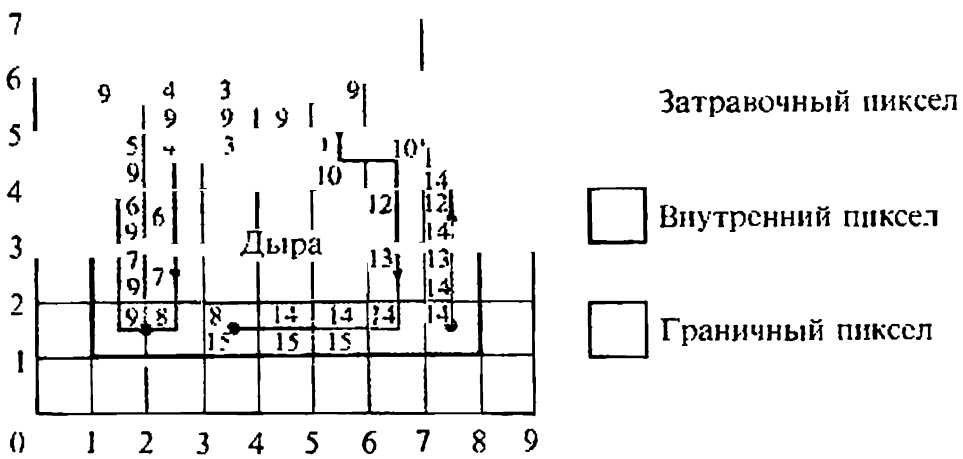



Рис. 2.48. Затравочное заполнение области с отверстием с помощью простого стекового алгоритма.

любого непрерывного интервала на сканирующей строке [2-25]. Непрерывный интервал — это группа примыкающих друг к другу пикселов (ограниченная уже заполненными или граничными пикселами). Мы для разработки алгоритма используем эвристический подход, однако также возможен и теоретический подход, основанный на теории графов [2-26].

Данный алгоритм применим к гранично-определенным областям. Гранично-определенная 4-связная область может быть как выпуклой, так и не выпуклой, а также может содержать дыры. В области, внешней и примыкающей к нашей гранично-определенной области, не должно быть пикселов с цветом, которым область или многоугольник заполняется. Схематично работу алгоритма можно разбить на четыре этапа.

Построчный алгоритм заполнения с затравкой

Затравочный пиксел на интервале извлекается из стека, содержащего затравочные пикселы.

Интервал с затравочным пикселом заполняется влево и вправо от затравки вдоль сканирующей строки до тех пор, пока не будет найдена граница.

В переменных $X_{лев}$ *и* $X_{прав}$ запоминаются крайний левый и крайний правый пикселы интервала.

В диапазоне $X_{лев} \leqslant x \leqslant X_{прав}$ проверяются строки, расположенные непосредственно над и под текущей строкой. Определяется, есть ли на них еще не заполненные пикселы. Если такие пикселы есть (т. е. не все пикселы граничные, или уже заполненные), то в указанном диапазоне крайний правый пиксел в каждом интервале отмечается как затравочный и помещается в стек.

При инициализации алгоритма в стек помещается единственный затравочный пиксел, работа завершается при опустошении стека. Как показано на рис. 2.49 и в примере ниже, алгоритм справляется с дырами и зубцами на границе. Ниже приводится более подробное описание алгоритма на псевдокоде.

**Построчный алгоритм заполнения с з атравкой**

*Затравка* (x, y) *выдает затравочный пиксел*
**Pop** — *процедура, которая извлекает пиксел из стека*

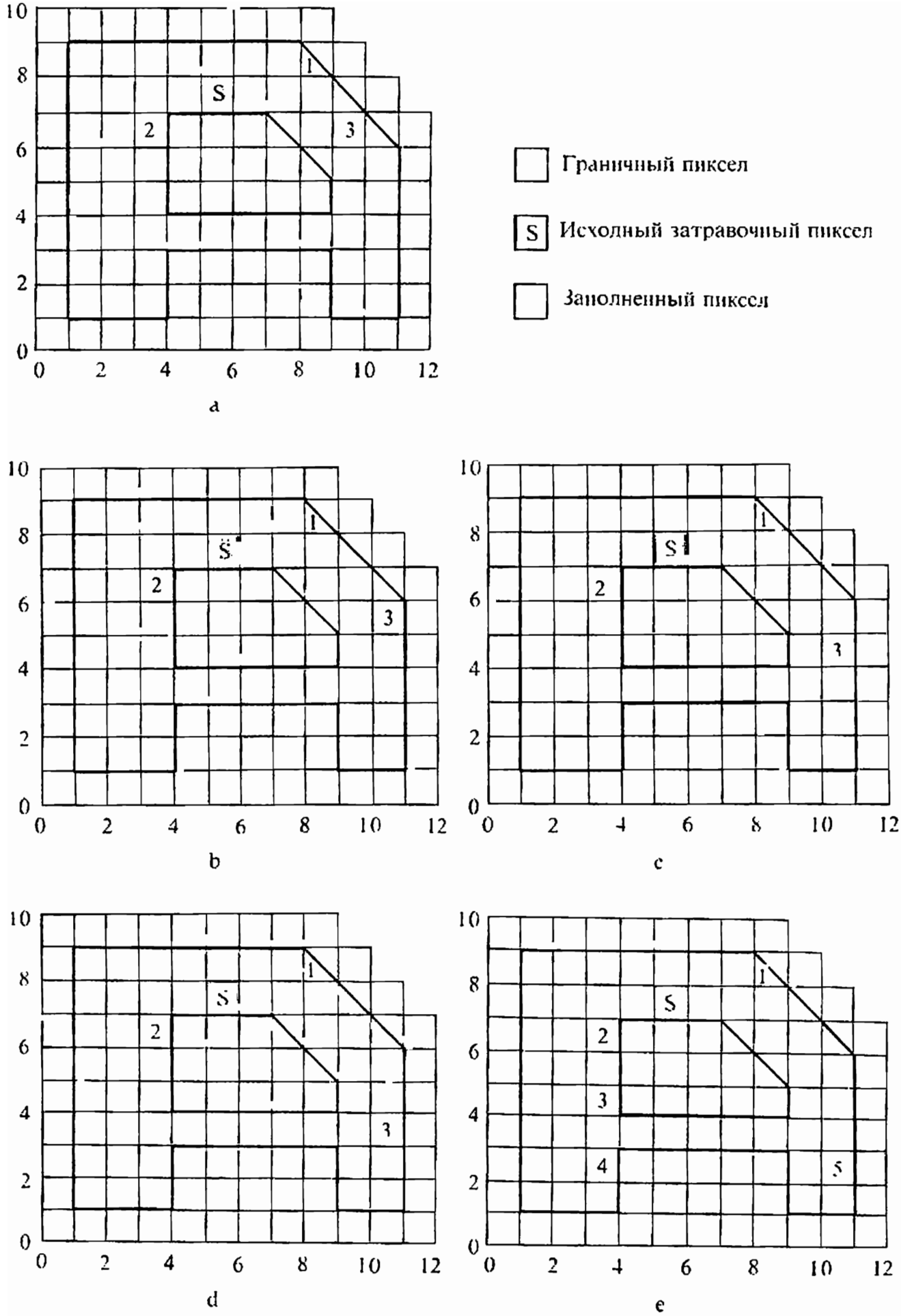



Рис. 2.49. Построчный алгоритм заполнения с затравкой для многоугольника.

```
Push — процедура, которая помещает пиксел в стек
  инициализируем стек
  Push Затравка (x, y)
  while (стек не пуст)
     извлекаем пиксел из стека и присваиваем ему новое значение
     Pop Пиксел (x, y)
     Пиксел (x, y) = Нов_значение
     сохраняем x-координату затравочного пиксела
     Врем_x = x
     заполняем интервал справа от затравки
     x = x + 1
     while Пиксел (x, y) < > Гран_значение
        Пиксел (x, y) = Нов_значение
        x = x + 1
     end while
     сохраняем крайний справа пиксел
     Хправ = x - 1
     восстанавливаем x-координату затравки
     x = Врем_x
     заполняем интервал слева от затравки
     x = x - 1
     while Пиксел (x, y) < > Гран_значение
        Пиксел (x, y) = Нов_значение
        x = x - 1
     end while
     сохраняем крайний слева пиксел
     Хлев = x + 1
     восстанавливаем x-координату затравки
     x = Врем_x
     проверим, что строка выше не является ни границей многоугольника, ни уже полностью заполненной; если это не так, то найти затравку, начиная с левого края подынтервала сканирующей строки
     x = Хлев
     y = y + 1
     while x ⩽ Хправ
        ищем затравку на строке выше
        Флаг = 0
        while (Пиксел (x, y) < > Гран_ значение and
                Пиксел (x, y) < > Нов_значение and x < Хправ
```

```
        if Флаг = 0 then Флаг = 1
        x = x + 1
      end while
      помещаем в стек крайний справа пиксел
      if Флаг = 1 then
        if (x = Хправ and Пиксел (x, y) < > Гран_значение
            and Пиксел (x, y) < > Нов_значение) then
          Push Пиксел (x, y)
        else
            Push Пиксел (x − 1, y)
        end if
        Флаг = 0
      end if
      продолжим проверку, если интервал был прерван
      Хвход = x
      while ((Пиксел (x, y) = Гран_значение or Пиксел (x, y) =
      = Нов_значение) and x < Хправ)
        x = x + 1
      end while
      удостоверимся, что координата пиксела увеличена
      if x = Хвход then x = x + 1
    end while
    проверим, что строка ниже не является ни границей много-
    угольника, ни уже полностью заполненной
        эта часть алгоритма совершенно аналогична проверке
        для строки выше, за исключением того, что вместо
        y = y + 1 надо подставить y = y − 1
end while
finish
```

Здесь функция **Pop** извлекает координаты $x$, $y$ пиксела из стека, а функция **Push** помещает их в стек.

**Пример 2.14.** Построчный алгоритм заполнения с затравкой

Рассмотрим работу алгоритма для гранично-определенной области на рис. 2.49. При инициализации в стек помещается затравочный пиксел, помеченный как Затравка (5,7) на рис. 2.49,а. Первоначально в качестве затравки интервала из стека извлекается этот пиксел. Интервал заполняется справа и слева от затравки. Найденные при этом концы интервала $X_{прав} = 9$ и $X_{лев} = 1$. Затем проверяется строка, расположенная выше текущей и оказывается, что она не граничная и не заполненная. Крайним правым пикселом в диапазоне $1 \leqslant x \leqslant 9$ оказывается пиксел (8,8), помечен цифрой 1 на рис. 2.49,а. Этот пиксел помещается в стек. Затем аналогичным

образом обрабатывается строка ниже текущей. В диапазоне $X_{лев} \leqslant x \leqslant X_{прав}$ есть два подынтервала на этой строке. Для левого интервала в качестве затравки выступает пиксел (3,6), помеченный цифрой 2 на рис. 2.49,*a*, он тоже помещается в стек. Затравка для правого подынтервала — пиксел (9,6), он помещается в стек. Заметим, что пиксел (9,6) — это не самый крайний правый пиксел на интервале, однако это самый крайний правый пиксел в диапазоне $X_{лев} \leqslant x \leqslant Y_{прав}$, т. е. в диапазоне $1 \leqslant x \leqslant 9$. На этом завершается первый проход алгоритма.

Далее из стека извлекается верхний пиксел. Здесь заполняются интервалы, расположенные в правой части многоугольника на последовательных сканирующих строках (рис. 2.49,b, c, d). Для строки 3 затравкой служит пиксел (10, 3) (рис. 2.49,d). В результате заполнения интервала справа и слева от затравки получаем новые пределы диапазона: $X_{лев} = 1$ и $Y_{прав} = 10$. Обрабатывая строку выше, получаем для левого подынтервала затравочный пиксел (3,4), который помещается в стек. Правый подынтервал к этому времени уже заполнен. Обработка строки ниже дает затравку (3,2) для левого и (10,2) для правого подынтервалов. Эти пикселы также помещаются в стек. Именно сейчас достигается максимальная глубина стека для обрабатываемого многоугольника.

Теперь остается только один интересный момент. После заполнения 4-связных полигональных подобластей с затравочными пикселами 5, 4 и 3 на рис. 2.49,e из стека извлекается пиксел, помеченный цифрой 2. Здесь мы обнаруживаем, что все пикселы на этой строке и на соседних строках (выше и ниже) уже заполнены. Таким образом, ни один пиксел в стек не помещается. Из стека извлекается пиксел 1 и строка обрабатывается, при этом вновь добавочных пикселов не появляется. Теперь стек пуст, многоугольник заполнен и работа алгоритма завершена.

По сравнению с алгоритмом из разд. 2.23 максимальная глубина стека в этом примере равна пяти. Другие методы заполнения с затравкой для многоугольника или области обсуждаются в [2-27].

## 2.25. ОСНОВЫ МЕТОДОВ УСТРАНЕНИЯ СТУПЕНЧАТОСТИ

Чтобы эффективно бороться со ступенчатостью (лестничным эффектом), приводящей к искажениям в изображении, необходимо понимать причины, ее вызывающие. Основная причина появления лестничного эффекта заключается в том, что отрезки, ребра многоугольника, цветовые границы и т. д. имеют непрерывную природу, тогда как растровое устройство дискретно. Для представления отрезка, ребра многоугольника и т. д. на растровом устройстве необходимо начертить их в дискретных координатах, что может привести к удивительным результатам. Рассмотрим, например, сигнал, изображенный на рис. 2.50,a. Второй сигнал более низкой частоты изображен на рис. 2.50,c. Если сделать выборки с одинаковой частотой из обоих сигналов (точки отмечены маленькими крестиками), то восстановленные сигналы, показанные на рис. 2.50,b и d, идентичны. Рис. 2.50,d называется искажением выборки, показанной на рис. 2.50,b и, следовательно, сигнала на рис. 2.50,a. Для

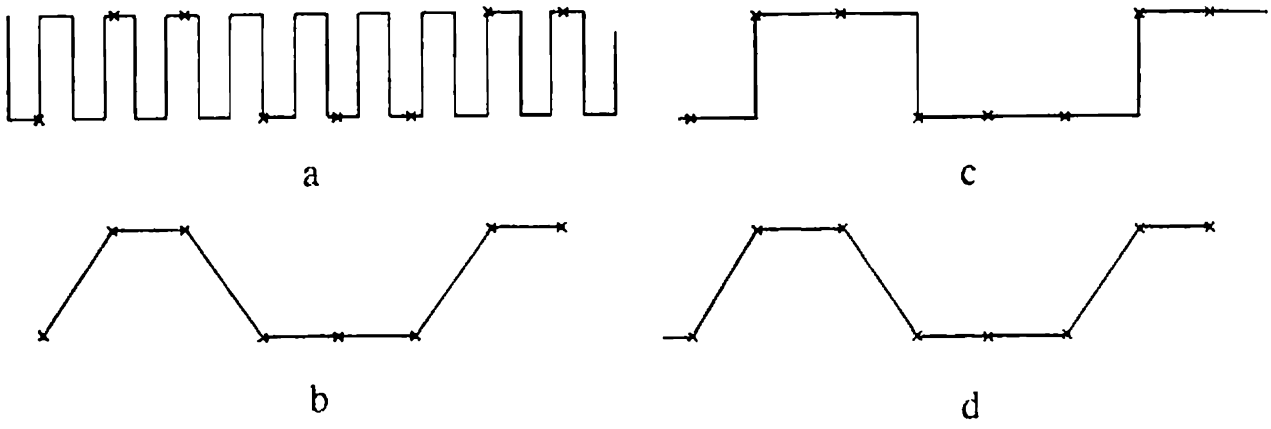



Рис. 2.50. Выборка и искажение при восстановлении.

высокочастотного сигнала (рис. 2.50,а) выборка проведена с недостаточной частотой. Для предотвращения искажения следует проводить выборку сигнала с частотой, по крайней мере вдвое превышающей наибольшую частоту сигнала. Недостаточная выборка приводит к тому, что высокопериодичные изображения визуализируются неверно. Например, ограда забора или подъемные жалюзи могут выглядеть как несколько широких полос, а не как много отдельных, более узких полосок.

Рассуждения, приведенные выше, и материал предыдущих разделов иллюстрируют два из трех общих проявлений искажений такого рода в машинно-сгенерированных изображениях: струпенчатость ребер и некорректная визуализация тонких деталей или фактуры. Третье проявление связано с очень мелкими объектами. Если объект меньше размера пиксела или не покрывает внутреннюю точку, служащую для оценки атрибутов пиксела, то он не будет учитываться в результирующем изображении. С другой стороны, если малый объем покрывает эту точку, то он может слишком сильно повлиять на атрибуты пиксела. На левом пикселе рис. 2.51. демонстрируется такая ситуация. Если для определения атрибутов используется середина пиксела, то в данной ситуации весь пиксел будет иметь атрибуты маленького объекта. На правых пикселах рис. 2.51 иллюстрируются объекты, которые будут проигнорированы или потеряны. Заметим, что могут быть проигнорированы также и длинные, тонкие объекты. Особенно заметны такие эффекты в анимационных последовательностях кинокадров. На рис. 2.52 представлен маленький треугольник на трех кадрах анимационной последовательности. Если атрибуты пиксела определяются в его центре, то на первом кадре объект невидим, на втором — видим, а на третьем — снова невидим. В совокупности это будет выглядеть как мерцание объекта.

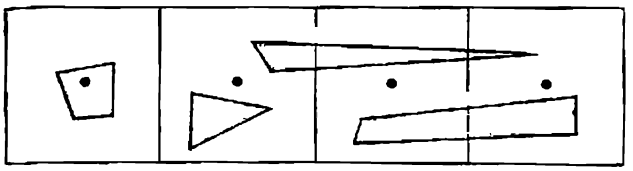


Рис. 2.51. Эффекты искажения для мелких объектов.

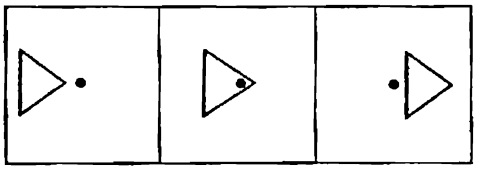


Рис. 2.52. Эффекты искажения в анимационной последовательности кинокадров.

В основном существует два метода устранения искажений изображения такого рода. Первый связан с увеличением частоты выборки, что достигается с помощью увеличения разрешения растра. Таким образом, учитываются более мелкие детали. Однако существует определенное ограничение на способность растровых графических устройств с ЭЛТ выводить очень мелкие растры. В настоящее время предел составляет около 2000 пикселов в строке. Такое ограничение предполагает, что растр надо вычислять с более высоким разрешением, а изображать с более низким, используя усреднение некоторого типа для получения атрибутов пиксела с более низким разрешением [2-28].

На рис. 2.53 показано усреднение двух типов. Равномерное усреднение окружающих пикселов для уменьшения разрешения в 2 и 4 раза демонстрируется на рис. 2.53,а. Каждый дисплейный пиксел делится на подпикселы в процессе формирования растра более высокого разрешения. Для получения атрибутов дисплейного пиксела определяются атрибуты в центре каждого подпиксела, которые затем усредняются. В некоторой степени можно получить лучшие результаты, если рассматривать больше подпикселов и учитывать их влияние с помощью весов при определении атрибутов. На

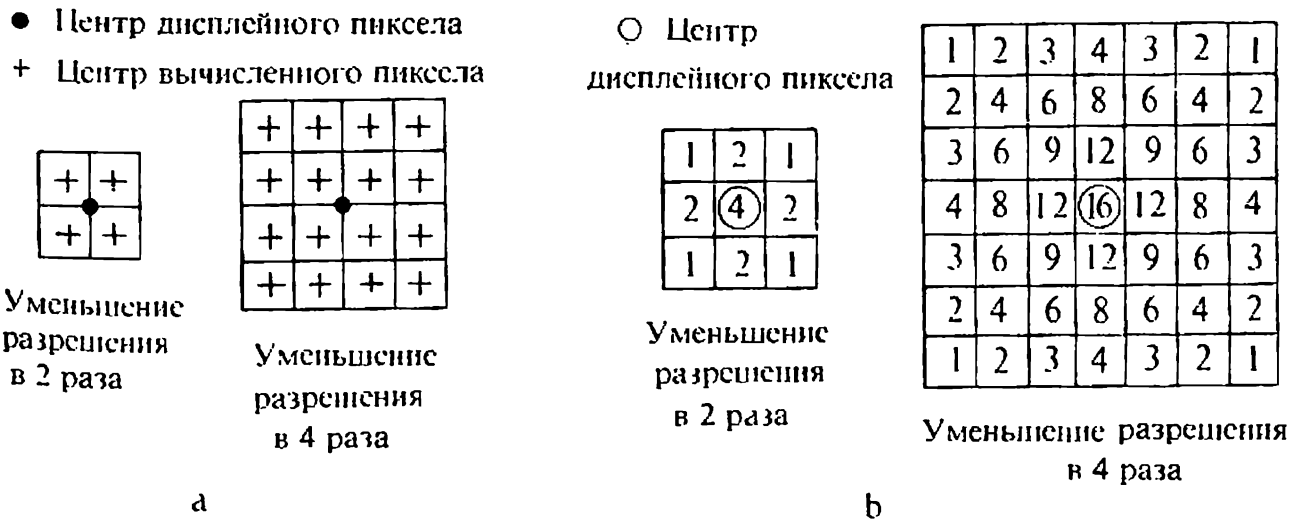



Рис. 2.53. Усреднение характеристик пиксела: (a) равномерное, (b) взвешенное (числа обозначают относительные веса).

Рис. 2.54. Изображения с высоким разрешением, выводимые в растре 256 × 256 и полученные с помощью равномерного усреднения из (a) растра 512 × 512, (b) растра 1024 × 1024. (С разрешения Ф. Кроу.)

рис. 2.53,b представлены взвешенные средние, предложенные Кроу [2-28], для уменьшения разрешения в 2 и 4 раза. Для этих взвешенных средних при уменьшении разрешения в 2 раза рассматриваются 9 подпикселов, а при уменьшении в 4 раза — 49.

На рис. 2.54 изображена сложная сцена с разрешением 256 × 256 пикселов, при этом для рис. 2.54,a она была вычислена при разреше-

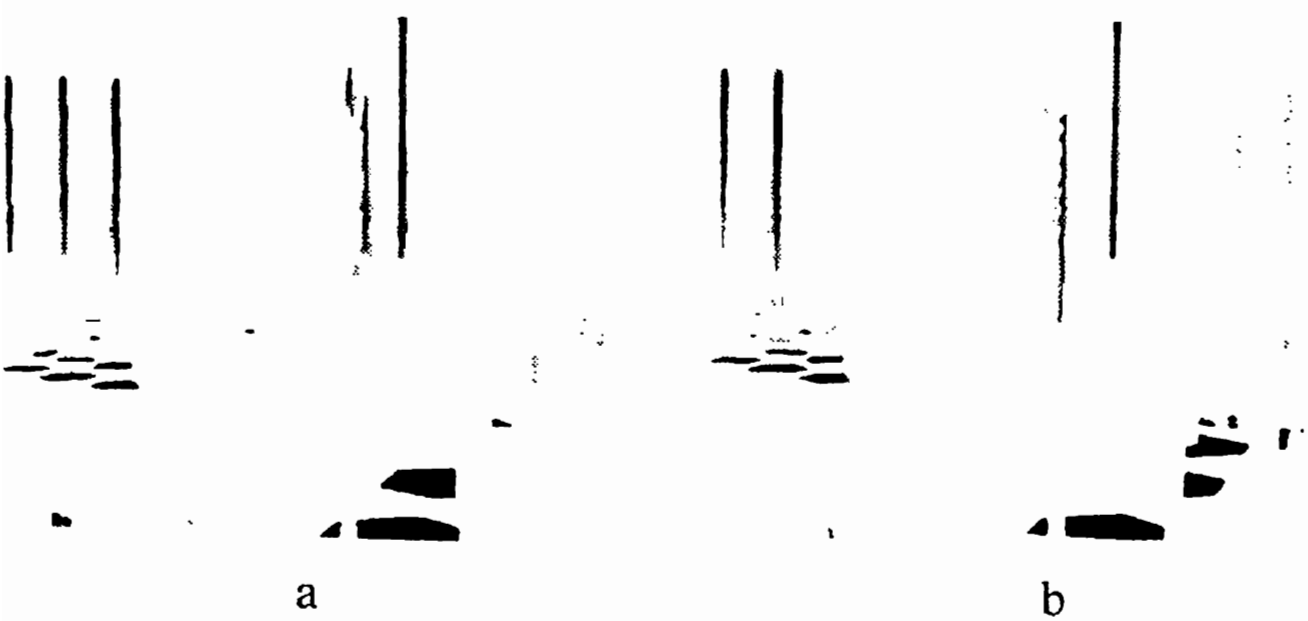



Рис. 2.55. Изображения с высоким разрешением, выводимые в растре 256 × 256 и полученные с помощью взвешенного усреднения из (a) растра 512 × 512, (b) растра 1024 × 1024.(С разрешения Ф. Кроу.)

нии 512 × 512, а для рис. 2.54,b — при 1024 × 1024. Для того чтобы получить изображение с разрешением 256 × 256 пикселов, было использовано равномерное усреднение. На рис. 2.55,a и b приведена одна и та же сцена, вычисленная при разрешении 512 × 512 и 1024 × 1024 соответственно, и изображенная с разрешением 256 × 256 с использованием взвешенных средних (рис. 2.53,b).

Второй метод устранения искажений изображения состоит в том, чтобы трактовать пиксел не как точку, а как конечную область. В следующем разделе приводится эвристический метод, а в разд. 2.27 — математическое обоснование такого подхода. Трактование пиксела как конечной области эквивалентно префильтрации изображения.

## 2.26. ПРОСТОЙ МЕТОД УСТРАНЕНИЯ ЛЕСТНИЧНОГО ЭФФЕКТА

В алгоритмах разложения отрезка в растр и заполнения многоугольника, обсуждавшихся выше, интенсивность или цвет пиксела определялись интенсивностью или цветом единственной точки внутри области пиксела. В этих методах предполагалось, что пиксел является скорее математической точкой, нежели конечной областью. Например, вспомнив рис. 2.4 и алгоритм Брезенхэма, мы увидим, что интенсивность пикселов определялась местоположением одной точки пересечения отрезка и границы пиксела. В методах растровой развертки сплошной области, тоже обсуждавшихся выше, определение местоположения пиксела (внутри или вне многоугольника) основывалось на положении центра пиксела. Если центр находился внутри, то активировался весь пиксел. Если центр находился вне, то игнорировалась вся область пиксела. Этот метод необходим для простых двухуровневых изображений, т. е. черных или белых, цвета многоугольника или цвета фона. В результате получается характерное ступенчатое или зазубренное ребро многоугольника или отрезок. Как обсуждалось в предыдущем разделе, основной причиной лестничного эффекта является то, что дискретность отрезка или ребра недостаточна для того, чтобы соответствовать дискретным пикселам экрана дисплея.

При наличии нескольких интенсивностей, т. е. полутонов серого или оттенков цвета, внешний вид ребра или отрезка может быть улучшен размыванием краев. Простой эвристический метод состоит в том, чтобы устанавливать интенсивность пиксела на ребре пропорционально площади части пиксела, находящейся внутри многоугольника. Эта простая форма устранения ступенчатости ил-

Рис. 2.56. Устранение спупенчагости ребра многоугольника: (а) без устранения ступенчатости, (в) интенсивность пропорциональна площади части пиксела, находящейся внутри многоугольника, (с) ребро, полученное с помощью модифицированного алгоритма Брезенхема.

люстрируется рис. 2.56, на котором изображено одно ребро многоугольника с тангенсом угла наклона $\frac{5}{8}$. Внутренняя часть многоугольника расположена справа. На рис. 2.56,а представлено ребро, разложенное в растр с помощью стандартного алгоритма Брезенхема с использованием только двух уровней интенсивности. Изображение имеет характерный зазубренный или ступенчатый вид. На рис. 2.56,b для выбора одного из восьми (от 0 до 7) уровней интенсивности используется площадь части пиксела, находящейся внутри многоугольника. Заметим, что так как эта площадь для некоторых пикселов меньше одной восьмой, то некоторые пикселы, полностью черные на рис. 2.56,а, окрашены в белый цвет на рис. 2.56,b.

В результате простой модификации алгоритма Брезенхема можно получить аппроксимацию площади части пиксела, находящейся внутри многоугольника [2-29]. Эту аппроксимацию можно использовать для модуляции интенсивности. Как показано на рис. 2.57, при пересечении пиксела и отрезка с тангенсом угла наклона $m$ $(0 \leqslant m \leqslant 1)$ может быть задействован либо один, либо два пиксела. Если пересекается только один пиксел (рис. 2.57,а), то площадь[1)] правее и ниже отрезка равна $y_i + m/2$. Если же надо рассмотреть два пиксела (рис. 2.57,b), то площадь нижнего пиксела составляет $1 - (1 - y_i)^2/2m$, а верхнего — $(y_i - 1 + m)^2/2m$. Для отрезка в первом октанте с тангенсом угла наклона $0 \leqslant m \leqslant 1$ площадь верхнего пиксела может быть достаточно мала для того, чтобы ее можно было проигнорировать в вышеописанном эвристическом методе, например, для пиксела (1, 1) на рис. 2.56,b. Однако объединение этой площади с площадью нижнего пиксела позволит

[1)] Далее везде, где говорится о площади пиксела, имеется в виду площадь той его части, которая находится внутри многоугольника. — *Прим. перев.*

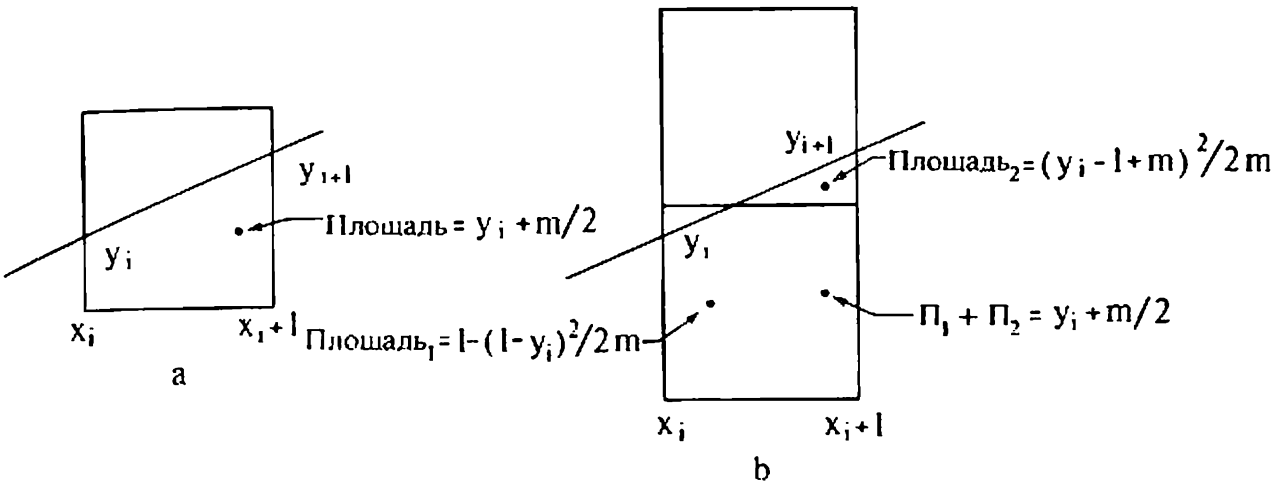



Рис. 2.57. Алгоритм Брезенхема, учитывающий площадь части пиксела для устранения ступенчатости.

более реалистично представить ребро многоугольника. Суммарная площадь для двух пикселов равна $y_i + m/2$.

Если к ошибке в исходном алгоритме Брезенхема добавить величину $w = 1 - m$, т. е. ввести преобразование $\bar{e} = e + w$, то $0 \leqslant \bar{e} \leqslant 1$. Теперь ошибка $\bar{e}$ — это мера площади той части пиксела, которая находится внутри многоугольника, т. е. $y_i + m/2$. В связи с этими модификациями начальное значение ошибки равно ½, поэтому для первого пиксела алгоритм, приводившийся на рис.2.6, всегда будет выдавать значение интенсивности, равное половине максимальной. Более реалистичное значение для первого пиксела дает перемещение оператора активирования пиксела на другое место. Более того, можно получить непосредственно значение интенсивности, а не десятичную дробь от ее максимума с помощью умножения на максимальное число доступных уровней интенсивности $I$ следующих величин: тангенса угла наклона ($m$), весового коэффициента ($w$) и ошибки $\bar{e}$. Модифицированный алгоритм выглядит следующим образом:

### *Модифицированный алгоритм Брезенхема с устранением ступенчатости для первого квадранта*

отрезок проводится из $(x_1, y_1)$ в $(x_2, y_2)$
I — число доступных уровней интенсивности
все переменные целого типа

инициализация переменных
$x = x_1$
$y = y_1$
$\Delta x = x_2 - x_1$

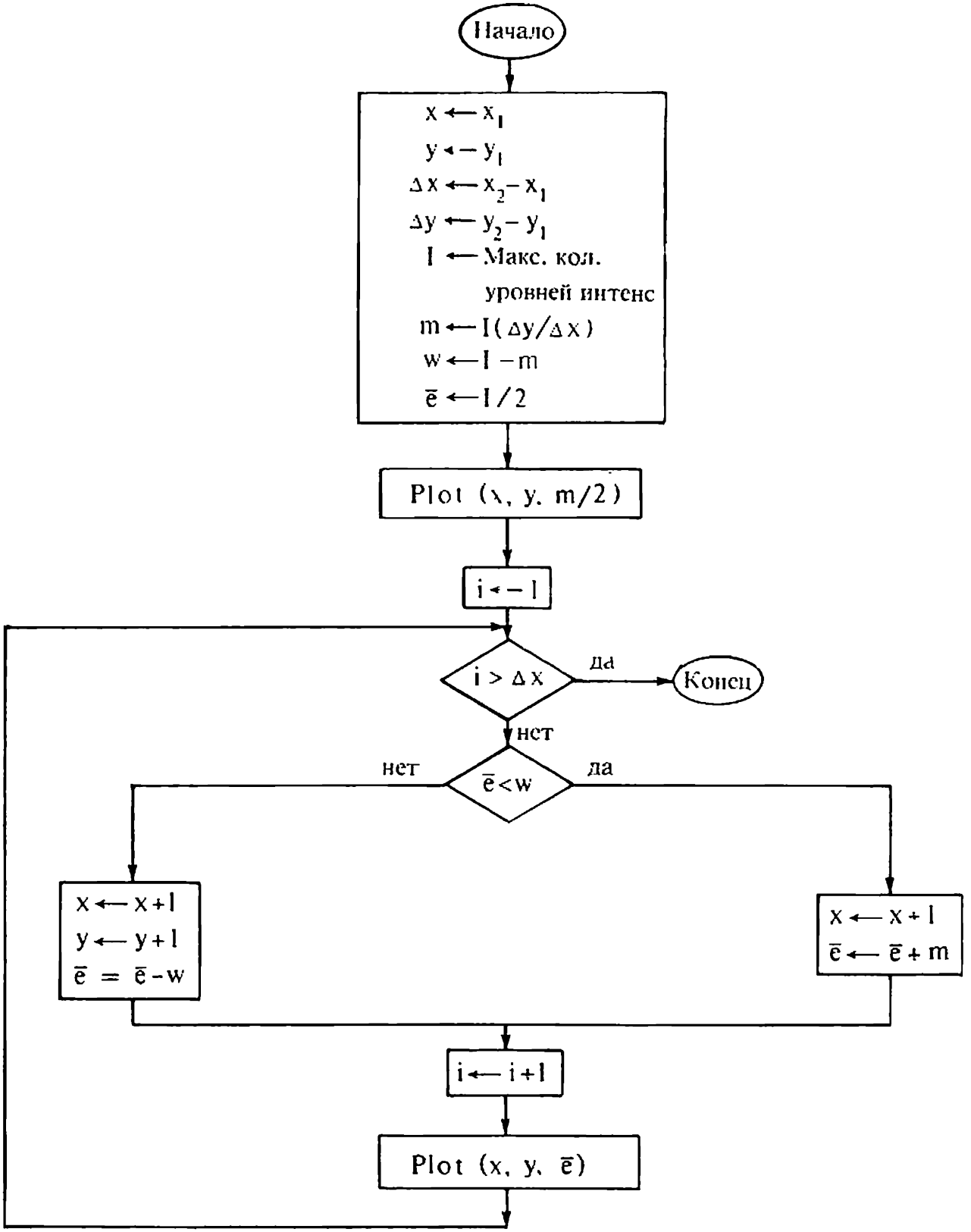



Рис. 2.58. Алгоритм Брезенхема с устранением ступенчатости.

```
Δy = y2 − y1
m = (I * Δy)/Δx
w = I − m
e = I/2
Plot (x, y, m/2)
while (x < x2)
  if e < w then
    x = x + 1
    e = e + m
  else
    x = x + 1
    y = y + 1
    e = e − w
  end if
  Plot (x, y, e)
end while
finish
```

Интенсивность для первого пиксела предполагает, что отрезок начинается с адреса пиксела. На рис. 2.58 приводится блок-схема алгоритма. Результаты для отрезка с тангенсом угла наклона $m = 5/8$ и восемью уровнями интенсивности представлены на рис. 2.56,с. Распространить работу алгоритма на другие октанты можно тем же способом, что и для основного алгоритма Брезенхема (см. разд. 2.5).

## 2.27. СВЕРТКА И УСТРАНЕНИЕ СТУПЕНЧАТОСТИ

Распространение обсуждавшихся в предыдущем разделе простых методов устранения ступенчатости (сглаживания) требует использования математического метода, называемого сверткой. Для сглаживания берется свертка сигнала, т. е. изображения, с ядром свертки, а результат используется для определения атрибутов пиксела. Свертка задается интегралом

$$c(\xi) = \int_{-\infty}^{\infty} h(\xi - x)y(x)\,dx$$

где

$h(\xi - x)$ — ядро или функция свертки,
$y(x)$ — свертываемая функция,
$c(\xi)$ — свертка $h(\xi - x)$ и $y(x)$

Представить себе физический смысл свертки из ее математического определения весьма трудно. Однако нам в этом поможет простой графический анализ [2-30].

Рассмотрим свертку функции $y(x) = x$, $0 \leqslant x \leqslant 1$ с простой прямоугольной функцией свертки $h(x) = 1$, $0 \leqslant x \leqslant 1$. В графическом виде функция свертки показана на рис. 2.59,a. Для того чтобы получить $h(-x)$, функцию надо отразить относительно ординаты (рис. 2.59,b). Отраженное ядро для получения $h(\xi - x)$ переносится вправо на величину $\xi$ (рис. 2.59,c). Затем, как показано на рис. 2.59,e, ядро свертки и свертываемую функцию $y(x)$ (рис. 2.59,d) перемножают для разных значений величины $\xi$. Площадь под объединенными кривыми (функциями) равна значению свертки $c(\xi)$, которая тоже изображена на рис. 2.59,c. Заметим, что в данном случае свертка не равна нулю только в диапазоне $0 \leqslant x \leqslant 2$. Таким образом, определение свертки эквивалентно отражению ядра свертки, его сдвигу, перемножению двух функций и определению площади, заключенной под объединенными кривыми.

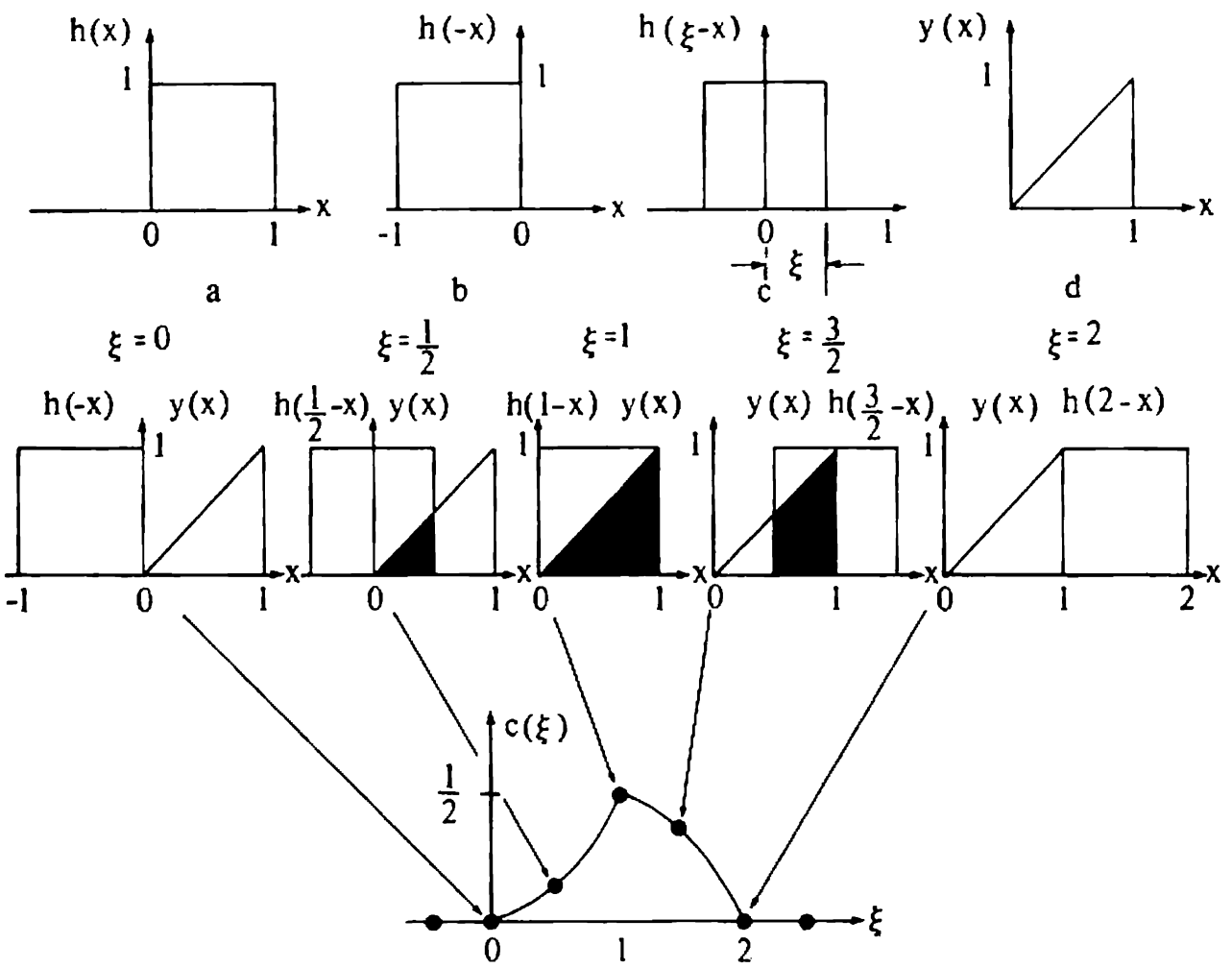



Рис. 2.59. Свертка.

Математически ядро свертки равно

$$h(x) = 1 \qquad 0 \leq x \leq 1$$

Отражение приводит к

$$h(-x) = 1 \qquad -1 \leq x \leq 0$$

Перенос на $\xi$ дает

$$h(\xi - x) = 1 \qquad \xi - 1 \leq x \leq \xi$$

Поскольку как ядро, так и свертываемая функция $y(x)$ не равны нулю лишь на конечных интервалах, то пределы интегрирования в свертке тоже конечны. Как их определить? Из рис. 2.59 ясно, что нижний предел равен максимуму начал интервалов, на которых функции не равны нулю, а верхний предел равен минимуму концов этих интервалов. Таким образом,

$$c(\xi) = \int_{-\infty}^{\infty} h(\xi - x)y(x)\,dx = \int_{0}^{\xi} h(\xi - x)y(x)\,dx \qquad 0 \leq \xi \leq 1$$

$$= \int_{\xi-1}^{1} h(\xi - x)y(x)\,dx \qquad 1 \leq \xi \leq 2$$

Подстановка функций $h(\xi - x)$ и $y(x)$ дает

$$c(\xi) = \int_{0}^{\xi} (1)(x)\,dx = \frac{x^2}{2}\Big]_0^{\xi} = \frac{\xi^2}{2} \qquad 0 \leq \xi \leq 1$$

$$= \int_{\xi-1}^{1} (1)(x)\,dx = \frac{x^2}{2}\Big]_{\xi-1}^{1} = \frac{\xi}{2}(2 - \xi) \qquad 1 \leq \xi \leq 2$$

где обе функции параболические (рис. 2.59,e). Если тангенс угла наклона равен $m$, а не 1, то результат можно обобщить в виде $m\,\xi^2/2$ и $(m\,\xi/2)(2 - \xi)$.

Чтобы понять, какая связь существует между сверткой и устранением ступенчатости (сглаживанием) вспомним эвристический метод модуляции интенсивности, в котором для определения интенсивности используется площадь пиксела. Проверка функции свертки $c(\xi)$ показывает, что для $m \leqslant 1$ значение свертки на правой стороне пиксела, т. е. при $x = \xi = 1$, равно площади той части пиксела, что находится внутри многоугольника, т. е. $m/2$ (рис. 2.57,b с $y_i = 0$). Для $m > 1$ значение свертки дает сумму площадей частей двух пересекаемых пикселов, расположенных внутри многоугольника (рис. 2.57,b с $y_i = 0$). Этот результат легко обобщить на случай

Рис. 2.60. Префильтрованное, сглаженное изображение с разрешением 256 × 256 пикселов. (С разрешения Ф. Кроу.)

$y_i \neq 0$. Таким образом, два предыдущих алгоритма (эвристический модулирования по площади и модифицированный Брезенхема) эквивалентны свертке функций ребер, т. е. прямой $y = mx + b$ и ядра свертки, вычисленных на правой стороне пиксела.

Операция свертки часто называется фильтрацией, а ядро свертки — функцией фильтра. В обсуждавшемся выше простом методе с площадями изображение предварительно фильтруется. Префильтрация выравнивает атрибуты пикселов вычисленного разрешения до вывода изображения на экране. Метод вычисления изображения с разрешением, большим, чем разрешение дисплея, и дальнейшее усреднение атрибутов нескольких пикселов для получения пикселов с меньшим разрешением можно рассматривать как постфильтрующую операцию (рис. 2.54 и 2.55).

Хотя при помощи простого прямоугольного фильтра или ядра свертки получают вполне приемлемые результаты, треугольный и гауссовский фильтры приводят к еще более качественным результатам [2-30]. Используются также двумерные фильтры. В настоящее время исследованы простое прямоугольное, пирамидальное, конечное и двумерное гауссовское ядра свертки или функции фильтра [2-30 — 2-34]. На рис. 2.60 показана та же сцена, что и на рис. 2.54 и 2.55, вычисленная с разрешением 256 × 256 пикселов, префильтрованная с простым прямоугольным фильтром и изображенная с разрешением 256 × 256.

Простые фильтры в виде свертки не всегда эффективны для маленьких многоугольников с площадью, меньшей площади пиксела или для длинных, тонких многоугольников. Тем не менее сглаживание можно реализовать с помощью отсечения (см. гл. 3 и [2-28]).

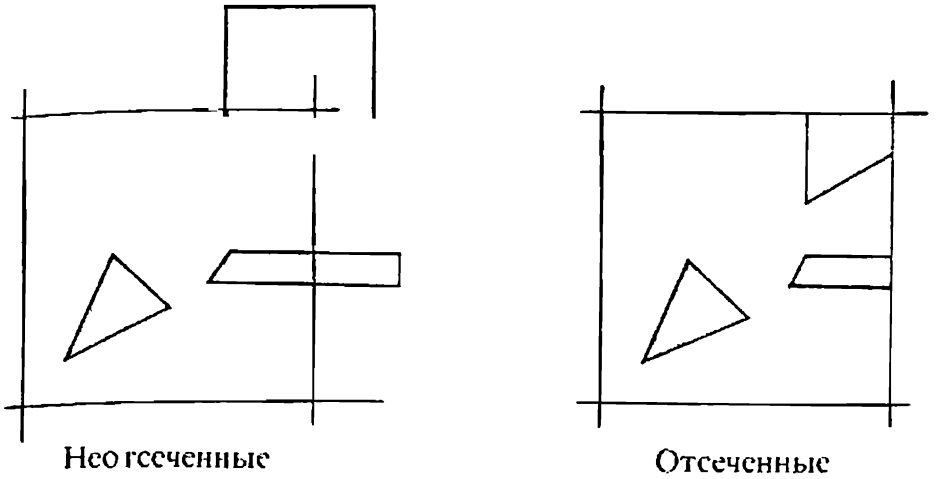



Рис. 2.61. Устранение ступенчатости с помощью отсечения.

Стороны области пиксела образуют отсекающее окно. Каждый индивидуальный многоугольник отсекается по сторонам этого окна. Для модулирования интенсивности пиксела используется отношение площади полученного в результате отсечения многоугольника к площади пиксела. Если внутри пиксела находится несколько многоугольников, то используется среднее (либо равномерное, либо взвешенное) их атрибутов для модулирования атрибутов пиксела. Пример такого отсечения приведен на рис. 2.61.

## 2.28. АППРОКСИМАЦИЯ ПОЛУТОНАМИ

Сглаживание или устранение ступенчатости — это метод улучшения визуального разрешения с использованием нескольких уровней интенсивности. Аппроксимация полутонами, с другой стороны, — это метод, в котором используется минимальное число уровней интенсивности, обычно черный и белый, для улучшения визуального разрешения, т. е. получения нескольких полутонов серого или уровней интенсивности. Метод полутонов известен довольно давно. Первоначально он использовался при изготовлении шелковых картин и других текстильных изделий. В 1880 г. Стефаном Хагеном была изобретена современная полутоновая печать. В этом методе можно получить большое количество фотографических полутонов серого, используя чисто двухуровневую среду: черную краску на белой бумаге. Полутоновая печать — это решеточный или клеточный процесс [2-35]. Размер клетки варьируется в зависимости от мелкозернистости решетки и длительности экспозиции. Для газетных фотографий из-за низкого качества бумаги применяются решетки от 50 до 90 точек на дюйм. Бумага более высокого качества, предназначенная для книг и журналов, позволяет использовать решетки с

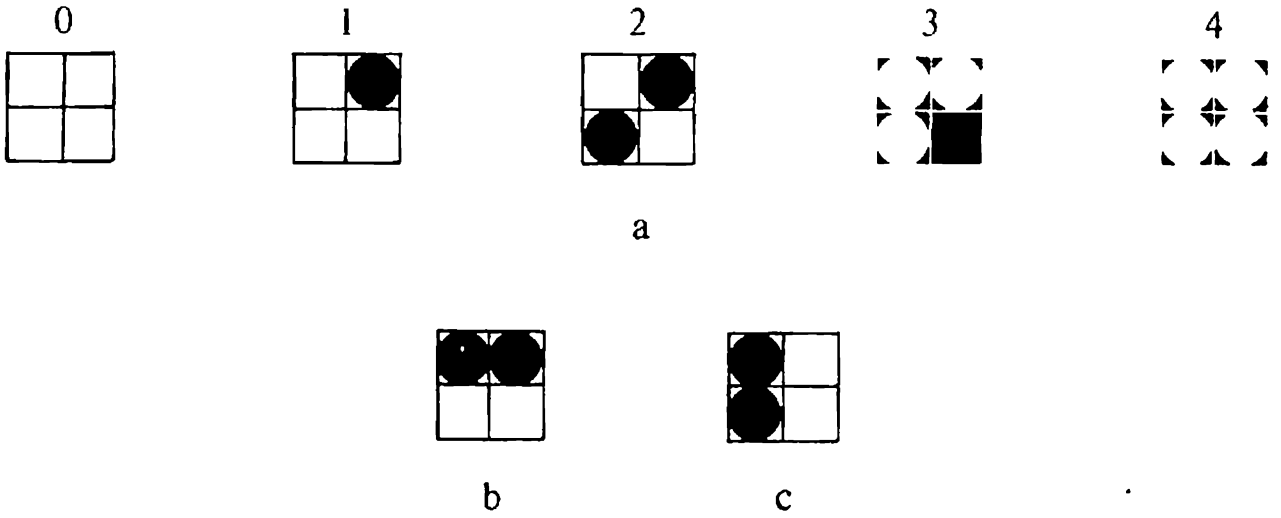



Рис. 2.62. Двухуровневые конфигурации 2 × 2.

количеством точек от 100 до 300 на дюйм. Успех метода полутонов зависит от свойства зрительной системы человека быть интегратором, т. е. объединять или сглаживать дискретную информацию.

Визуальное разрешение машинно-сгенерированных изображений можно улучшить с помощью метода, называемого конфигурированием. В противоположность полутоновой печати, в которой используются переменные размеры клеток, в данном методе обычно размеры клеток фиксированы. Для изображения с фиксированным разрешением несколько пикселов объединяются в конфигурации. Здесь ухудшение пространственного разрешения обменивается на улучшение визуального. На рис. 2.62,a показана одна из возможных групп конфигураций для двухуровневого черно-белого дисплея. Для каждой клетки используется четыре пиксела. При такой организации получается пять возможных уровней или тонов серого (0—4). В общем случае для двухуровнего дисплея число возможных интенсивностей на единицу больше числа пикселов в клетке. При выборе конфигураций следует проявлять осторожность, так как иначе могут возникнуть нежелательные мелкомасштабные структуры. Например, не следует применять ни одну из конфигураций, изображенных на рис. 2.62,b или c, иначе это приведет к тому, что для большой области с постоянной интенсивностью на изображении появятся нежелательные горизонтальные или вертикальные линии. Число доступных уровней интенсивности можно увеличить с помощью увеличения размера клетки. Конфигурации для клетки 3 × 3 пикселов приведены на рис. 2.63. Они дают десять уровней (с 0 по 9) интенсивности. Клетки конфигураций не обязательно должны быть квадратными; на рис. 2.64 изображена клетка 3 × 2 пикселов, дающая семь (0—6) уровней интенсивности.

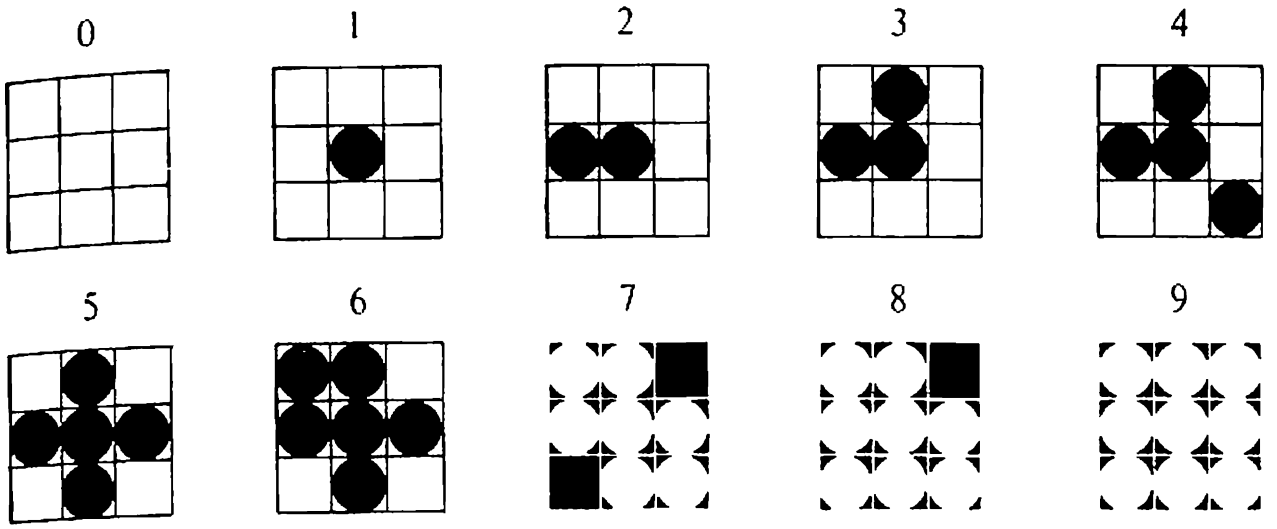



Рис. 2.63. Двухуровневые конфигурации 3 × 3.

Если точки могут быть разного размера, то можно получить дополнительное количество уровней интенсивности. На рис. 2.65 представлены конфигурации для клетки 2 × 2 пиксела с двумя размерами точек. В результате получается 9 уровней интенсивности. Подобная клетка размера 3 × 3 с точками двух размеров позволяет иметь 27 уровней интенсивности. Если на пиксел приходится больше одного бита, то также можно получить дополнительные уровни интенсивности. Для конфигурации 2 × 2 с 2 битами на пиксел получится 13 уровней интенсивности, показанные на рис. 2.66. Большее количество бит на пиксел или больший размер клетки дадут соответствующее увеличение числа уровней интенсивности [2-36].

Использование конфигураций ведет к потере пространственного разрешения, что приемлемо в случае, когда разрешение изображения меньше разрешения дисплея. Разработаны также методы улучшения визуального разрешения при сохранении пространственного [2-37]. Простейший из них состоит в применении порогового значе-

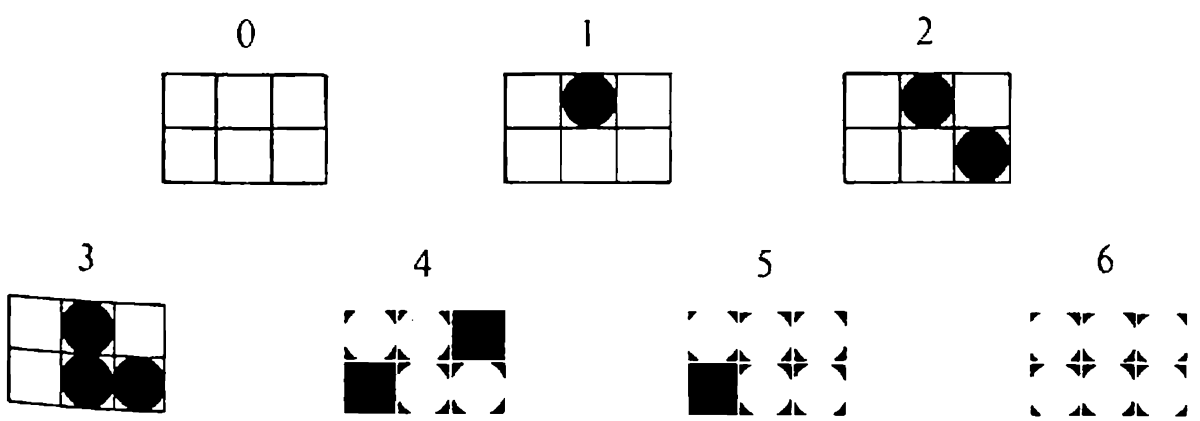



Рис. 2.64. Двухуровневые конфигурации 3 × 2.

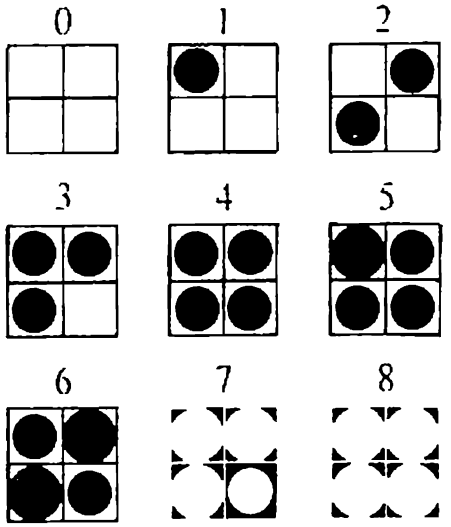



Рис. 2.65. Конфигурации 2 × 2 с точками нескольких размеров.

ния для каждого пиксела. Если интенсивность изображения превышает некоторую пороговую величину, то пиксел считается белым, в противном случае он черный:

**if** I(x, y) > T **then** Белый **else** Черный,

где $I(x, y)$ означает интенсивность пиксела $(x, y)$ изображения. Белый соответствует максимальной интенсивности для дисплея, а черный — минимальной. Пороговую величину обычно устанавливают приблизительно равной половине максимальной интенсивности. На рис. 2.67,b представлен результат для фотографии, изображенной на рис. 2.67,a, со значением $T = 150$. В точках исходной фотографии, соответствующих каждому пикселу, интенсивность была разбита на дискретные значения в диапазоне от 0 до 255, т. е. использовалось 8 бит. Как показано нп рис. 2.67,b, при простом пороговом методе наблюдается потеря большого количества мелких деталей. Особенно это заметно для волос и черт лица. Мелкие де-

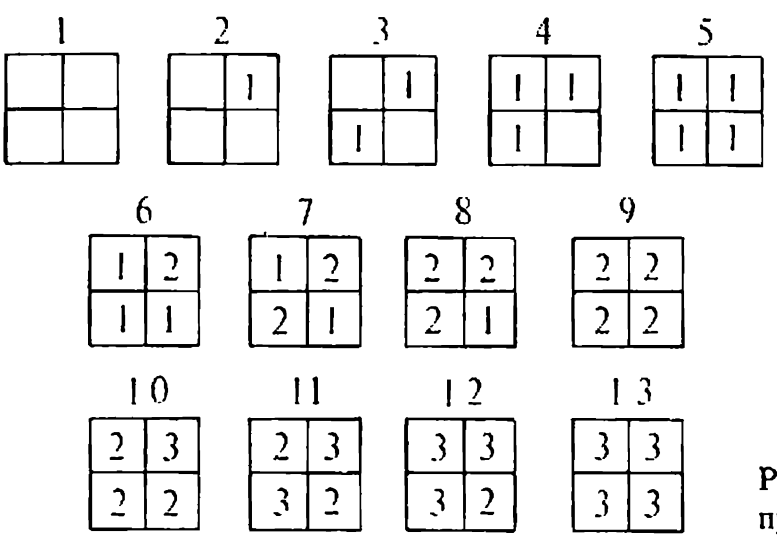



Рис. 2.66. Конфигурации 2 × 2 при двух битах на пиксел.

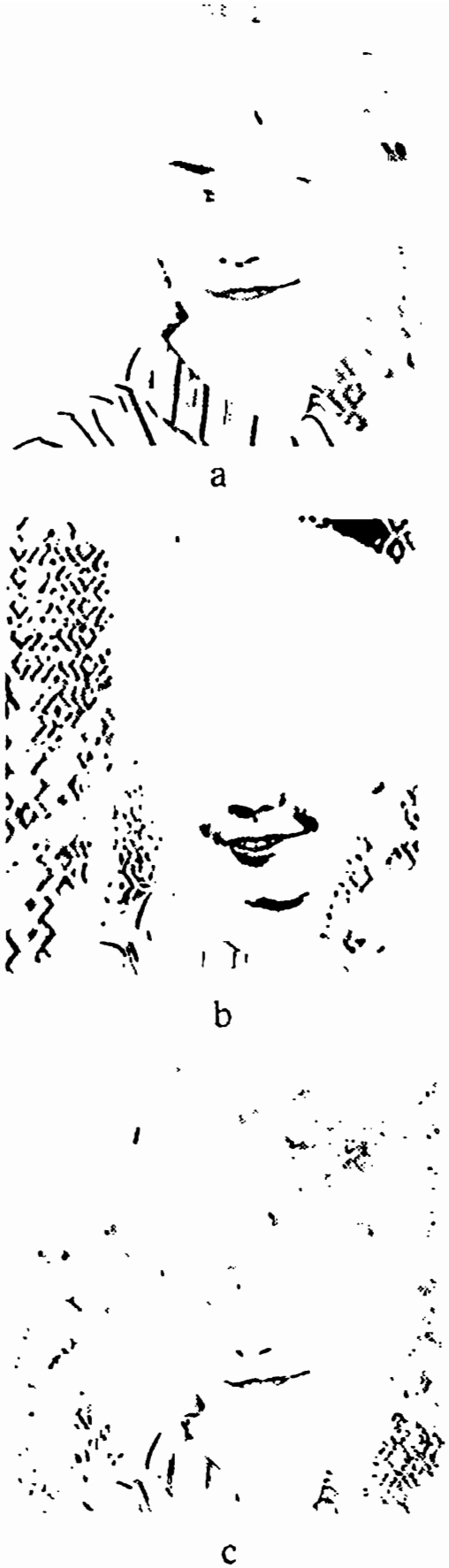



Рис. 2.67. Методы двухуровневого вывода изображений: (a) исходная фотография, (b) простой пороговый метод, (c) упорядоченное возбуждение с матрицей 8 × 8. (С разрешения Дж. Ф. Джарвис, Bell Ladoratories.)

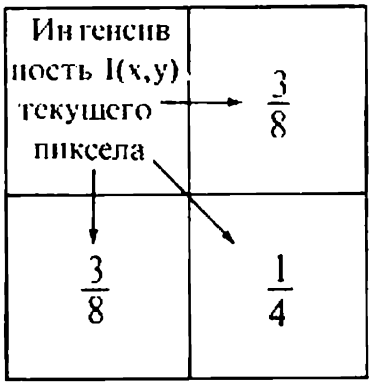



Рис. 2.68. Распределение ошибки в алгоритме Флойда — Стейнберга.

тали теряются из-за относительно больших ошибок выводимой интенсивности для каждого пиксела.

В методе, разработанном Флойдом и Стейнбергом [2-38], эта ошибка распределяется на окружающие пикселы. Распределение ошибки происходит всегда вниз и вправо. Следовательно, при генерации изображения в порядке сканирования возвращаться обратно не нужно. В частности, в алгоритме Флойда—Стейнберга $3/8$ ошибки распределяется вправо, $3/8$ — вниз и $1/4$ — по диагонали, как это показано на рис. 2.68. Для порога, равного среднему между минимальной и максимальной интенсивностями, $T$ = (Белый + Черный)/2, алгоритм формулируется следующим образом:

***Алгоритм распределения ошибки Флойда — Стейнберга***

```
Xmin, Xmax, Ymin, Ymax — пределы растра
 T = (Черный + Белый)/2
 for y = Ymax to Ymin step −1
   для каждого пиксела на строке (слева направо)
   for x = Xmin to Xmax
       определяем выводимое значение пиксела для пороговой
       величины T и вычисляем ошибку
       if I(x, y) < T then
            Пиксел (x, y) = Черный
            Ошибка = I(x, y) − Черный
       else
            Пиксел (x, y) = Белый
            Ошибка = I(x, y) − Белый
       end if
       изображаем пиксел
       Display Пиксел (x, y)
       распределяем ошибку на соседние пикселы
       I(x + 1, y) = I(x + 1, y) + 3 * Ошибка/8
       I(x, y − 1) = I(x, y − 1) + 3 * Ошибка/8
```

```
        I(x + 1, y − 1) = I(x + 1, y − 1) + Ошибка/4
    next x
next y
finish
```

Распределение ошибки на соседние пикселы улучшает вид деталей изображения, так как информация, заключенная в изображении, не теряется.

Существует другой метод улучшения визуального разрешения для двухуровневых дисплеев без уменьшения пространственного разрешения — метод возбуждения. В изображение вводится случайная ошибка, которая добавляется к интенсивности каждого пиксела до ее сравнения с выбранной пороговой величиной. Добавление совершенно произвольной ошибки не приводит к оптимальному результату. Тем не менее существует оптимальная аддитивная матрица ошибки, минимизирующая эффекты появления фактуры на изображении [2-39]. Матрица ошибки добавляется к изображению таким же способом, как расположены клетки на шахматной доске. Данный метод называется упорядоченным возбуждением. Минимальная матрица упорядоченного возбуждения имеет размер 2 × 2. Оптимальная 2 × 2-матрица, которую первым предложил Лим [2-40], имеет вид

$$[D_2] = \begin{bmatrix} 0 & 2 \\ 3 & 1 \end{bmatrix}$$

Матрицы 4 × 4, 8 × 8 и больших размеров получают с помощью рекуррентных соотношений [2-37]

$$[D_n] = \begin{bmatrix} 4D_{n/2} & 4D_{n/2} + 2U_{n/2} \\ 4D_{n/2} + 3U_{n/2} & 4D_{n/2} + U_{n/2} \end{bmatrix} \quad n \geq 4$$

где $n$ — размер матрицы и

$$[U_n] = \begin{bmatrix} 1 & 1 & \dots & 1 \\ 1 & 1 & & \\ \vdots & & & \\ 1 & & & \end{bmatrix}$$

Например, матрица возбуждения размера 4 × 4 имеет вид

$$[D_4] = \begin{bmatrix} 0 & 8 & 2 & 10 \\ 12 & 4 & 14 & 6 \\ 3 & 11 & 1 & 9 \\ 15 & 7 & 13 & 5 \end{bmatrix}$$

Как показывают два этих примера, из матрицы возбуждения $D_n$ можно породить $n^2$ интенсивностей. С увеличением $n$ изображение не теряет пространственного разрешения. Приведем алгоритм упорядоченного возбуждения.

### *Алгоритм упорядоченного возбуждения*

Xmin, Xmax, Ymin, Ymax — *пределы растра*
**Mod** — *функция, возвращающая остаток от целого деления первого аргумента на второй*

```
for y = Ymax to Ymin step −1
    для каждого пиксела на строке (слева направо)
    for x = Xmin to Xmax
        определяем позицию в матрице возбуждения
        i = (x Mod n) + 1
        j = (y Mod n) + 1
        определяем выводимое значение пиксела
        if I(x, y) < D(i, j) then
            Пиксел (x, y) = Черный
        else
            Пиксел (x, y) = Белый
        end if
        изображаем пиксел
      Display Пиксел (x, y)
      next x
next y
finish
```

На рис. 2.67,с показано изображение, полученное из фотографии на рис. 2.67,а в результате обработки ее с матрицей упорядоченного возбуждения размера 8 × 8. При использовании матрицы такого размера эффективно вводится 64 уровня интенсивности. Рис. 2.67,с свидетельствует о том, что восстанавливается много мелких деталей. Алгоритм Флойда—Стейнберга и упорядоченное возбуждение можно применять к цветным изображениям [2-41]. Конфигурационные методы тоже можно использовать с цветом [2-42].

## 2.29. ЛИТЕРАТУРА


2-1 Bresenham, J.E., "Algorithm for Computer Control of a Digital Plotter," *IBM System Journal*, Vol. 4, pp. 25–30, 1965.

2-2 Pitteway, M.L.V., "Algorithm for Drawing Ellipses or Hyperbolas with a Digital Plotter," *Computer Journal*, Vol. 10, pp. 282–289, 1967.

2-3 Jordon, B.W., Jr., Lennon, W.J., and Holm, B.D., "An Improved Algorithm for the Generation of Nonparametric Curves," *IEEE Trans. Comput.*, Vol. C-22, pp. 1052–1060, 1973.

2-4 Belser, K., Comment on "An Improved Algorithm for the Generation of Nonparametric Curves," *IEEE Trans. Comput.*, Vol. C-25, p. 103, 1976.

2-5 Ramot, J., "Nonparametric Curves," *IEEE Trans. Comput.*, Vol. C-25, pp. 103–104, 1976.

2-6 Horn, B.K.P., "Circle Generators for Display Devices," *Computer Graphics and Image Processing*, Vol. 5, pp. 280–288, 1976.

2-7 Badler, N.I., "Disk Generator for a Raster Display Device," *Computer Graphics and Image Processing*, Vol. 6., pp. 589–593, 1977.

2-8 Doros, M., "Algorithms for Generation of Discrete Circles, Rings, and Disks," *Computer Graphics and Image Processing*, Vol. 10, pp. 366–371, 1979.

2-9 Suenaga, Y., Kamae, T., and Kobayashi, T., "A High-speed Algorithm for the Generation of Straight I ines and Circular Arcs," *IEEE Trans. Comput.*, Vol. C-28, pp. 728–736, 1979.

2-10 Bresenham, J., "A Linear Algorithm for Incremental Digital Display of Circular Arcs," *CACM*, Vol. 20, pp. 100–106, 1977.

2-11 Standish, Thomas A., *Data Structures Techniques*, Addison-Wesley Publishing Company, Reading, Mass., 1980.

2-12 Knuth, Donald, E., *The Art of Computer Programming*, Vol. 3, *Sorting and Searching*, Addison-Wesley Publishing Company, Reading, Mass. 1973.

2-13 Laws, B.A., "A Gray-Scale Graphic Processor Using Run-Length Encoding," *Proc. IEEE Conf. Comput. Graphics, Pattern Recognition, Data Struct.*, pp. 7–10, May 1975.

2-14 Hartke, David H., Sterling, Warren M., and Shemer, Jack E., "Design of a Raster Display Processor for Office Applications," *IEEE Trans. Comput.*, Vol. C-27, pp. 337–348, 1978.

2-15 Jordan, B.W., and Barrett, R.C.,"A Cell Organized Raster Display for Line Drawings," *CACM*, Vol. 17, pp. 70–77, 1974.

2-16 Barrett, R.C., and Jordan, B.W., "Scan Conversion Algorithms for a Cell Organized Raster Display," *CACM*, Vol. 17, pp. 157–163, 1974.

2-17 Willett, Ken, "The 4027—Adding a Color Dimension to Graphics," *Tekscope*, Vol. 10, pp. 3–6.

2-18 Negroponte, N., "Raster Scan Approaches to Computer Graphics," *Computers & Graphics*, Vol. 2, pp. 179–193, 1977.

2-19 Baecker, Ronald, "Digital Video Display Systems and Dynamic Graphics," *Computer Graphics*, Vol. 13, pp. 48–56, 1979 (*Proc. SIGGRAPH 79*).

2-20 McCracken, T.E., Sherman, B.W., and Dwyer, S.J., III, "An Economical Tonal Display for Interactive Graphics and Image Analysis Data," *Computers & Graphics*, Vol. 1, pp. 79–94, 1975.

2-21 Whitted, Turner, "A Software Test-Bed for the Development of 3-D Raster Graphics Systems," *Computer Graphics*, Vol. 15, pp. 271–277, 1981 (*Proc. SIGGRAPH* 81).
2-22 Ackland, Bryan, and Weste, Neil, "Real Time Animation on a Frame Store Display System," *Computer Graphics*, Vol. 14, pp. 182–188, 1980 (*Proc. SIGGRAPH* 80).
2-23 Dunlavey, Michael R., "Efficient Polygon-Filling Algorithms for Raster Displays," *ACM Trans. on Graphics*, Vol. 2, pp. 264–273, 1983.
2-24 Ackland, Bryan, and Weste, Neil, "The Edge Flag Algorithm—A Fill Method for Raster Scan Displays," *IEEE Trans. Comput.*, Vol. C-30, pp. 41–48, 1981.
2-25 Smith, Alvy Ray, "Tint Fill," *Computer Graphics*, Vol. 13, pp. 276–283, 1979 (*Proc. SIGGRAPH* 79).
2-26 Shani, Uri, "Filling Regions in Binary Raster Images: A Graph-Theoretic Approach," *Computer Graphics*, Vol. 14, pp. 321–327, 1980 (*Proc. SIGGRAPH* 80).
2-27 Pavlidis, Theo, "*Algorithms for Graphics and Image Processing*," Computer Science Press, Rockville, Md. 1982.
2-28 Crow, Franklin C., "A Comparison of Antialiasing Techniques," *IEEE CG & A*, Vol. 1, pp. 40–47, 1981.
2-29 Pitteway, M.L.V., and Watkinson, D.J., "Bresenham's Algorithm with Gray Scale," *CACM*, Vol. 23, pp. 625–626, 1980.
2-30 Brigham, F. Oran, *The Fast Fourier Transform*, Prentice-Hall, Englewood Cliffs, 1974.
2-31 Crow, Franklin C., "The Aliasing Problem in Computer-Generated Shaded Images," *CACM*, Vol. 20, pp. 799–805, 1977.
2-32 Feibush, Eliot A., Levoy, Marc, and Cook, Robert L., "Synthetic Texturing Using Digital Filters," *Computer Graphics*, Vol. 14, pp. 294–301, 1980 (*Proc. SIGGRAPH* 80).
2-33 Warnock, John, "The Display of Characters Using Gray Level Sample Arrays," *Computer Graphics*, Vol. 14, pp. 302–307, 1980 (*Proc. SIGGRAPH* 80).
2-34 Gupta, Satish, and Sproull, Robert F., "Filtering Edges for Gray-Scale Displays," *Computer Graphics*, Vol. 15, pp. 1–6, 1981 (*Proc. SIGGRAPH* 81).
2-35 *Halftone Methods for the Graphic Arts* (Q3), 3d ed., Eastman Kodak, Rochester, N.Y. 1982.
2-36 Pirsch, P. and Netravali, A.N., "Transmission of Gray Level Images by Multilevel Dither Techniques," *Computers & Graphics*, Vol. 7, pp. 31–44, 1983.
2-37 Jarvis, J.F., Judice, C.N., and Ninke, W.H., "A Survey of Techniques for the Display of Continuous Tone Pictures on Bilevel Displays," *Computer Graphics and Image Processing*, Vol. 5, pp. 13–40, 1976.
2-38 Floyd, R., and Steinberg, L., "An Adaptive Algorithm for Spatial Gray Scale, SID 1975," *Int. Symp. Dig. Tech., Pap.*, pp. 36–37, 1975.
2-39 Bayer, B.E., "An Optimum Method For Two-Level Rendition of Continuous-Tone Pictures," *Int. Conf. Commun., Conf. Rec.*, pp. (26-11)–(26-15), 1973.
2-40 Limb, J.O., "Design of Dither Waveforms for Quantized Visual Signals," *Bell System Technical Journal*, Vol. 48, pp. 2555–2582, 1969.
2-41 Heckbert, Paul, "Color Image Quantization For Frame Buffer Display," *Computer Graphics*, Vol. 16, pp. 297–307, 1982, (*Proc. SIGGRAPH* 82).
2-42 Kubo, Sachio, "Continuous Color Presentation Using a Low-Cost Ink Jet Printer," *Proc. Comput. Graphics Tokyo 84*, 24–27 April, 1984, Tokyo, Japan.

2-12 Кнут Д. Искусство программирования для ЭВМ. Т.3. Сортировка и поиск. — М.: Мир, 1978

2-27 Павлидис Т. Алгоритмы машинной графики и обработки изображений. — М.: Радио и связь, 1986.

# 3 ⟫

# Отсечение

Отсечение, т. е. процесс выделения некоторой части базы данных, играет важную роль в задачах машинной графики. В гл. 2 было показано, что отсечение используется и для устранения ступенчатости, помимо своего более привычного применения для отбора той информации, которая необходима для визуализации конкретной сцены или вида, как части более обширной обстановки. В следующих главах будет показано, что отсечение применяется в алгоритмах удаления невидимых линий и поверхностей, при построении теней, а также при формировании фактуры. Алгоритмы и понятия, рассматриваемые здесь, можно применить для создания более совершенных алгоритмов, которые отсекают многогранники другими многогранниками, хотя эта задача и выходит за рамки данной книги. Такие алгоритмы можно использовать для реализации булевых операций, которые нужны в простых системах геометрического моделирования, например при вычислении пересечений и объединений простых тел, ограниченных плоскостями или поверхностями второго порядка. Получающиеся при этом приближенные решения годятся во многих приложениях.

Алгоритмы отсечения бывают дву- или трехмерными и применяются как к регулярным[1], так и к нерегулярным областям и объемам. Эти алгоритмы можно реализовать аппаратно или программно. Алгоритмы отсечения, реализованные программно, зачастую

---

[1] Имеются в виду канонические (стандартные) области и объемы. В частности, к ним относятся прямоугольники и параллелепипеды со сторонами, параллельными осям координат. В литературе такие объекты называют также изотетичными.— *Прим. ред.*

оказываются недостаточно быстродействующими для приложений, ориентированных на процессы, протекающие в реальном времени. Поэтому как трех-, так и двумерные алгоритмы отсечения реализуются аппаратными или микропрограммными средствами. В подобных реализациях обычно ограничиваются дву- или трехмерными отсекателями типовых форм. Однако с появлением сверхбольших интегральных схем (СБИС) открываются возможности для более общих реализаций, позволяющих работать в реальном времени [3-1] как с регулярными, так и с нерегулярными областями и телами.

## 3.1. ДВУМЕРНОЕ ОТСЕЧЕНИЕ

На рис. 3.1 показана плоская сцена и отсекающее окно регулярной формы. Окно задается левым (Л), правым (П), верхним (В) и нижним (Н) двумерными ребрами. Регулярным отсекающим окном является прямоугольник, стороны которого параллельны осям координат объектного пространства или осям координат экрана. Целью алгоритма отсечения является определение тех точек, отрезков или их частей, которые лежат внутри отсекающего окна. Эти точки, отрезки или их части остаются для визуализации. А все остальное отбрасывается.

Поскольку в обычных сценах или картинках необходимо отсекать большое число отрезков или точек, то эффективность алгоритмов отсечения представляет особый интерес. Во многих случаях подавляющее большинство точек или отрезков лежит целиком внутри или вне отсекающего окна. Поэтому важно уметь быстро отбирать отрезки, подобные $ab$, или точки, подобные $p$, и отбрасывать отрезки, подобные $ij$, или точки, подобные $q$ на рис. 3.1.

Точки, лежащие внутри отсекающего окна, удовлетворяют условию: $x_{Л} \leqslant x \leqslant x_{П}$ и $y_{Н} \leqslant y \leqslant y_{В}$. Знак равенства здесь показывает,

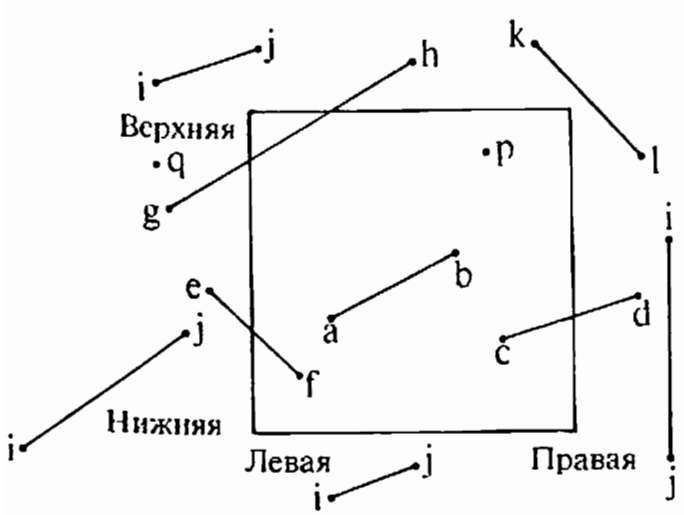



Рис. 3.1. Двумерное отсекающее окно.

что точки, лежащие на границе окна, считаются находящимися внутри него.

Отрезок лежит внутри окна и, следовательно, является видимым, если обе его концевые точки лежат внутри окна, например отрезок *ab* на рис. 3.1. Однако если оба конца отрезка лежат вне окна, то этот отрезок не обязательно лежит целиком вне окна, например отрезок *gh* на рис. 3.1. Если же оба конца отрезка лежат справа, слева, выше или ниже окна, то этот отрезок целиком лежит вне окна, а значит, невидим. Проверка последнего условия устранит все отрезки, помеченные *ij* на рис. 3.1. Но она не устранит ни отрезка *gh*, который видим частично, ни отрезка *kl*, который целиком невидим.

Пусть *a* и *b* — концы отрезка, тогда можно предложить алгоритм, определяющий все полностью видимые и большинство невидимых отрезков:

***простой алгоритм определения видимости***

a, b — *концевые точки отрезка в координатном пространстве* (x, y)
*для каждого отрезка проверить полную видимость отрезка*
*если одна из координат какого-нибудь конца отрезка находится вне окна, то отрезок не является полностью видимым*
**if** $x_a < x_Л$ **or** $x_a > x_П$ **then** 1
**if** $x_b < x_Л$ **or** $x_b > x_П$ **then** 1
**if** $y_a < y_Н$ **or** $y_a > y_В$ **then** 1
**if** $y_b < y_Н$ **or** $y_b > y_В$ **then** 1
*отрезок полностью видимый*
*визуализировать отрезок*
**go to** 3
*проверить полную невидимость отрезка*
*если оба конца отрезка лежат слева, справа, сверху или снизу от окна, то факт невидимости отрезка тривиален*

1 **if** $x_a < x_Л$ **and** $x_b < x_Л$ **then** 2
**if** $x_a > x_П$ **and** $x_b < x_П$ **then** 2
**if** $y_a > y_В$ **and** $y_b > y_В$ **then** 2
**if** $y_a < y_Н$ **and** $y_b < y_Н$ **then** 2
*отрезок частично видим или пересекает продолжение диагонали, оставаясь невидимым*
*определить пересечения отрезка с окном*

2 *отрезок невидим*
3 *переход к следующему отрезку*

Здесь через $x_Л$, $x_П$, $y_В$, $y_Н$ обозначены координаты $x$ и $y$ левого, правого, верхнего и нижнего краев окна соответственно. Порядок проведения сравнений при определении видимости или невидимости несуществен. Для некоторых отрезков может понадобиться проведение всех четырех сравнений, прежде чем определится их полная видимость или невидимость. Для других отрезков может потребоваться только одно сравнение. Несущественно также, что выявляется раньше — полностью видимые или полностью невидимые отрезки. Однако, поскольку определение пересечения отрезка с окном требует большого объема вычислений, его следует проводить в последнюю очередь.

Приведенные выше тесты полной видимости или невидимости отрезков можно формализовать, используя метод Д. Коэна и А. Сазерленда. В этом методе для определения той из девяти областей, которой принадлежит конец ребра, вводится четырехразрядный (битовый) код. Коды этих областей показаны на рис. 3.2. Крайний правый бит кода считается первым. В соответствующий бит заносится 1 при выполнении следующих условий:

для первого бита — если точка левее окна
для второго бита — если точка правее окна
для третьего бита — если точка ниже окна
для четвертого бита — если точка выше окна

В противном случае в бит заносится нуль. Отсюда, очевидно, следует, что если коды обоих концов ребра равны нулю, то обе эти точки лежат внутри окна, и отрезок видимый. Коды концевых то-

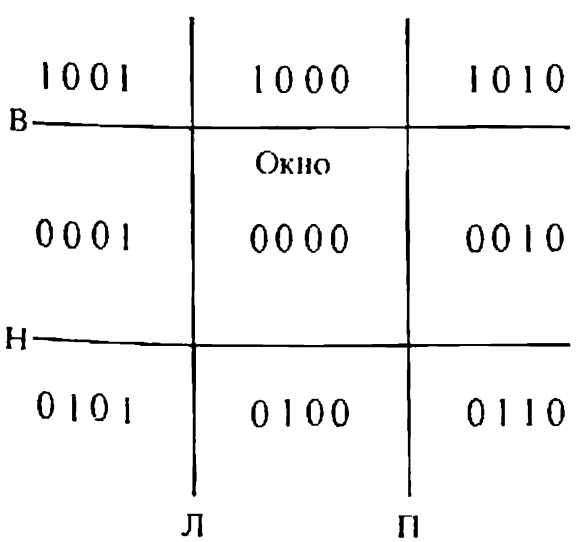



Рис. 3.2. Коды областей, которым принадлежат концевые точки.

чек можно использовать также и для тривиального отбрасывания полностью невидимых отрезков. Рассмотрим таблицу истинности, эквивалентную логическому оператору «и»:

| | | | | |
|---|---|---|---|---|
| Истина и Ложь | → | Ложь | Ложь = 0 | 1 и 0 → 0 |
| Ложь и Истина | → | Ложь | | 0 и 1 → 0 |
| Ложь и Ложь | → | Ложь | → | 0 и 0 → 0 |
| Истина и Истина | → | Истина | Истина = 1 | 1 и 1 → 1 |

Если побитовое логическое произведение кодов концевых точек отрезка *не равно нулю*, то отрезок полностью невидим и его можно отбросить тривиально. Несколько примеров, приведенных в табл. 3.1, помогут прояснить высказанные утверждения. Из табл. 3.1 видно, что если результат логического умножения не равен нулю, то фактически отрезок будет целиком невидим. Однако если логическое произведение равно нулю, то отрезок может оказаться целиком или частично видимым или даже целиком невидимым. Поэтому для определения полной видимости необходимо проверять значения кодов обоих концов отрезка по отдельности.

Проверку значений кодов концов отрезка можно легко реализовать, если воспользоваться подпрограммами, оперирующими с битами. Однако в алгоритмах, которые рассматриваются ниже, такие подпрограммы не используются.

Если прежде всего найдены целиком видимые и тривиально невидимые отрезки, то подпрограмме, вычисляющей пересечение от-

**Таблица 3.1. Коды концов отрезков**

| Отрезок (рис. 3.1) | Коды концов (рис. 3.2) | Результаты логического умножения | Примечания |
|---|---|---|---|
| *ab* | 0000 0000 | 0000 | Целиком видим |
| *ij* | 0010 0110 | 0010 | Целиком невидим |
| *ij* | 1001 1000 | 1000 | —"— |
| *ij* | 0101 0001 | 0001 | —"— |
| *ij* | 0100 0100 | 0100 | —"— |
| *cd* | 0000 0010 | 0000 | Частично видим |
| *ef* | 0001 0000 | 0000 | —"— |
| *gh* | 0001 1000 | 0000 | —"— |
| *kl* | 1000 0010 | 0000 | Целиком невидим |

резков, передаются только отрезки, которые, возможно, частично видимы, т. е. те, для которых результат логического умножения кодов их концевых точек равен нулю. Конечно же, эта подпрограмма должна правильно определять переданные ей такие целиком невидимые отрезки.

Пересечение двух отрезков можно искать как параметрическим, так и непараметрическим способами. Очевидно, что уравнение бесконечной прямой, проходящей через точки $P_1(x_1, y_1)$ и $P_2(x_2, y_2)$, имеет вид $y = m(x - x_1) + y_1$ или $y = m(x - x_2) + y_2$, где $m = (y_2 - y_1)/(x_2 - x_1)$ — это наклон данной прямой. Точки пересечения этой прямой со сторонами окна имеют следующие координаты:

| | | |
|---|---|---|
| с левой: | $x_Л,\ y = m(x_Л - x_1) + y_1$ | $m \neq \infty$ |
| с правой: | $x_П,\ y = m(x_П - x_1) + y_1$ | $m \neq \infty$ |
| с верхней: | $y_В,\ x = x_1 + (1/m)(y_В - y_1)$ | $m \neq 0$ |
| с нижней: | $y_Н,\ x = x_1 + (1/m)(y_Н - y_1)$ | $m \neq 0$ |

В примере 3.1 показано, как этот простой метод позволяет отбросить некорректные пересечения путем простого сравнения координат точек пересечения с координатами сторон окна.

**Пример 3.1.** Простое двумерное отсечение

Рассмотрим отсекающее окно и отрезки, изображенные на рис. 3.5.

Наклон отрезка от $P_1(-3/2, 1/6)$ до $P_2(1/2, 3/2)$ равен $m = (y_2 - y_1)/(x_2 - x_1) = (3/2 - 1/6) / [1/2 - (-3/2)] = 2/3$. Его пересечения со сторонами окна таковы:

с левой: $x = -1$ $\quad y = (2/3)[-1 - (-3/2)] + 1/6 = 1/2$

с правой: $x = 1$ $\quad y = (2/3)[1 - (-3/2)] + 1/6 = 11/6$

(последнее число больше, чем $y_В$, и поэтому отвергается)

с верхней: $y = 1$ $\quad x = -3/2 + (3/2)[1 - 1/6] = -1/4$

с нижней: $y = -1$ $\quad x = -3/2 + (3/2)[-1 - (1/6)] = -13/4$

(последнее число меньше, чем $x_П$, и поэтому отвергается)

Аналогично, отрезок от $P_3(-3/2, -1)$ до $P_4(3/2, 2)$ имеет

$$m = \frac{y_2 - y_1}{x_2 - x_1} = \frac{2 - (-1)}{3/2 - (-3/2)} = 1$$

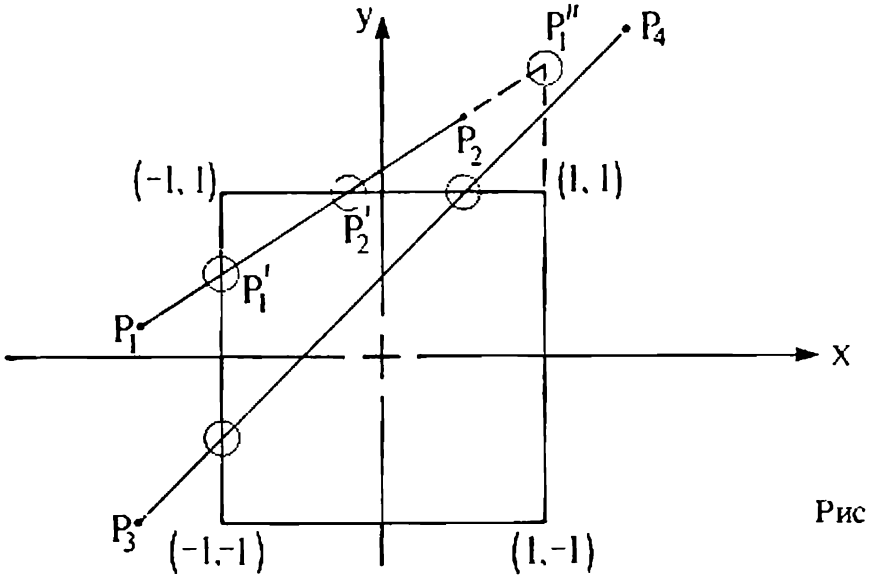



Рис. 3.3. Двумерное параметрическое отсечение

и пересечения:

с левой: $x = -1$ $\quad y = (1)[-1 - (-3/2)] + (-1) = -1/2$

с правой: $x = 1$ $\quad y = (1)[1 - (-3/2)] + (-1) = 3/2$

(последнее число больше, чем $y_в$, и поэтому отвергается)

с верхней: $y = 1$ $\quad x = -3/2 + (1)[1 - (-1)] = 1/2$

с нижней: $y = -1$ $\quad x = -3/2 + (1)[-1 - (-1)] = -3/2$

(последнее число больше, чем $x_л$, и поэтому отвергается).

Чтобы разработать схему эффективного алгоритма отсечения, необходимо сначала рассмотреть несколько частных случаев. Напомним, что, как уже указывалось, если наклон бесконечен, то отрезок параллелен левой и правой сторонам окна и надо искать его пересечения только с верхней и нижней сторонами. Аналогично, если наклон равен нулю, то отрезок параллелен верхней и нижней сторонам окна, а искать его пересечения надо только с левой и правой сторонами. Наконец, если код одного из концов отрезка равен нулю, то этот конец лежит внутри окна, и поэтому отрезок может пересечь только одну сторону окна. На рис. 3.4 приводится блок-схема алгоритма, использующего приведенные выше соображения. Ниже следует запись этого алгоритма на псевдокоде.

### *Простой алгоритм двумерного отсечения*

$P_1$, $P_2$ — *концевые точки отрезка*
$P_1'$, $P_2'$ — *концевые точки видимой части отрезка*

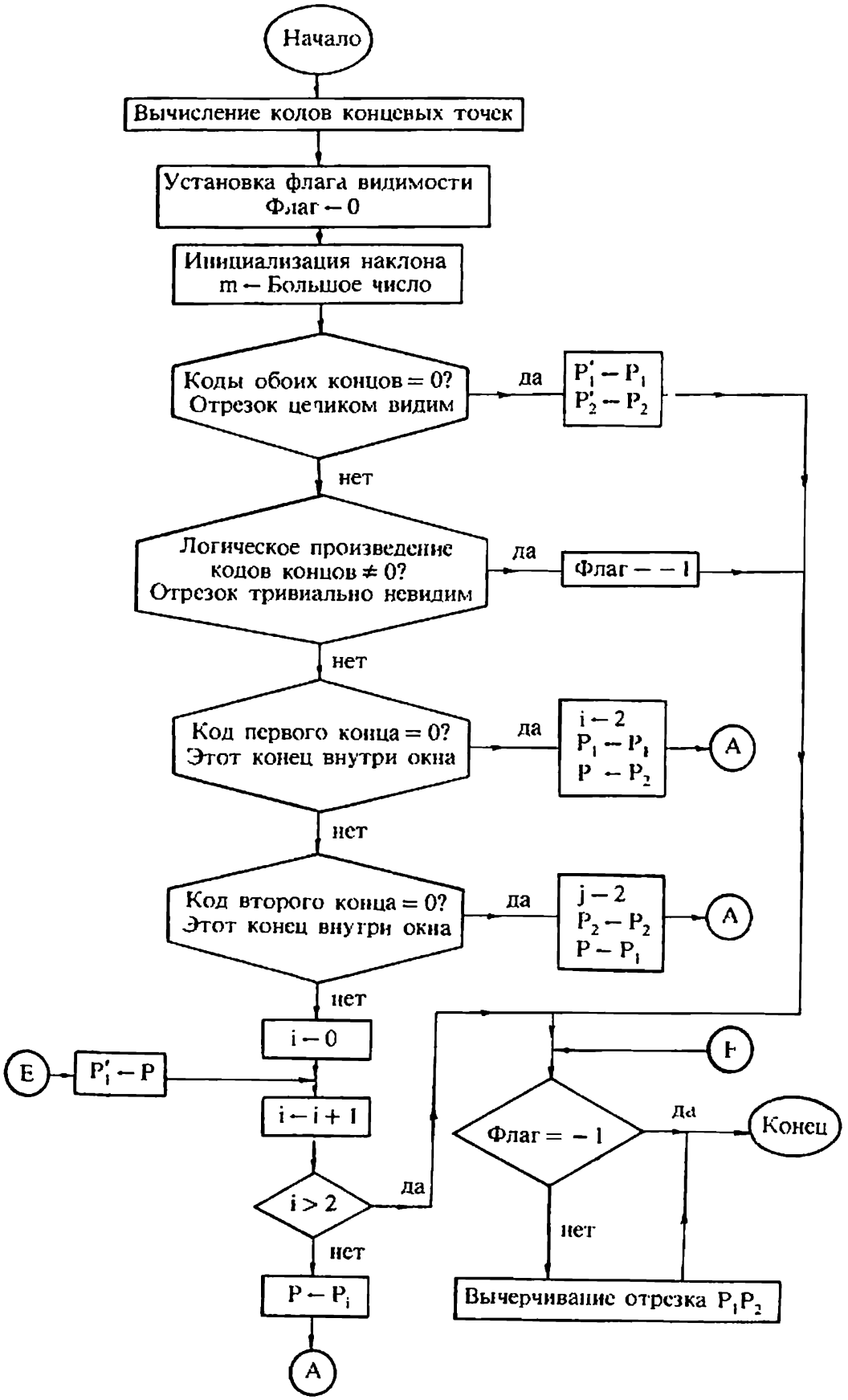



Рис. 3.4. Блок-схема алгоритма простого двумерного отсечения.

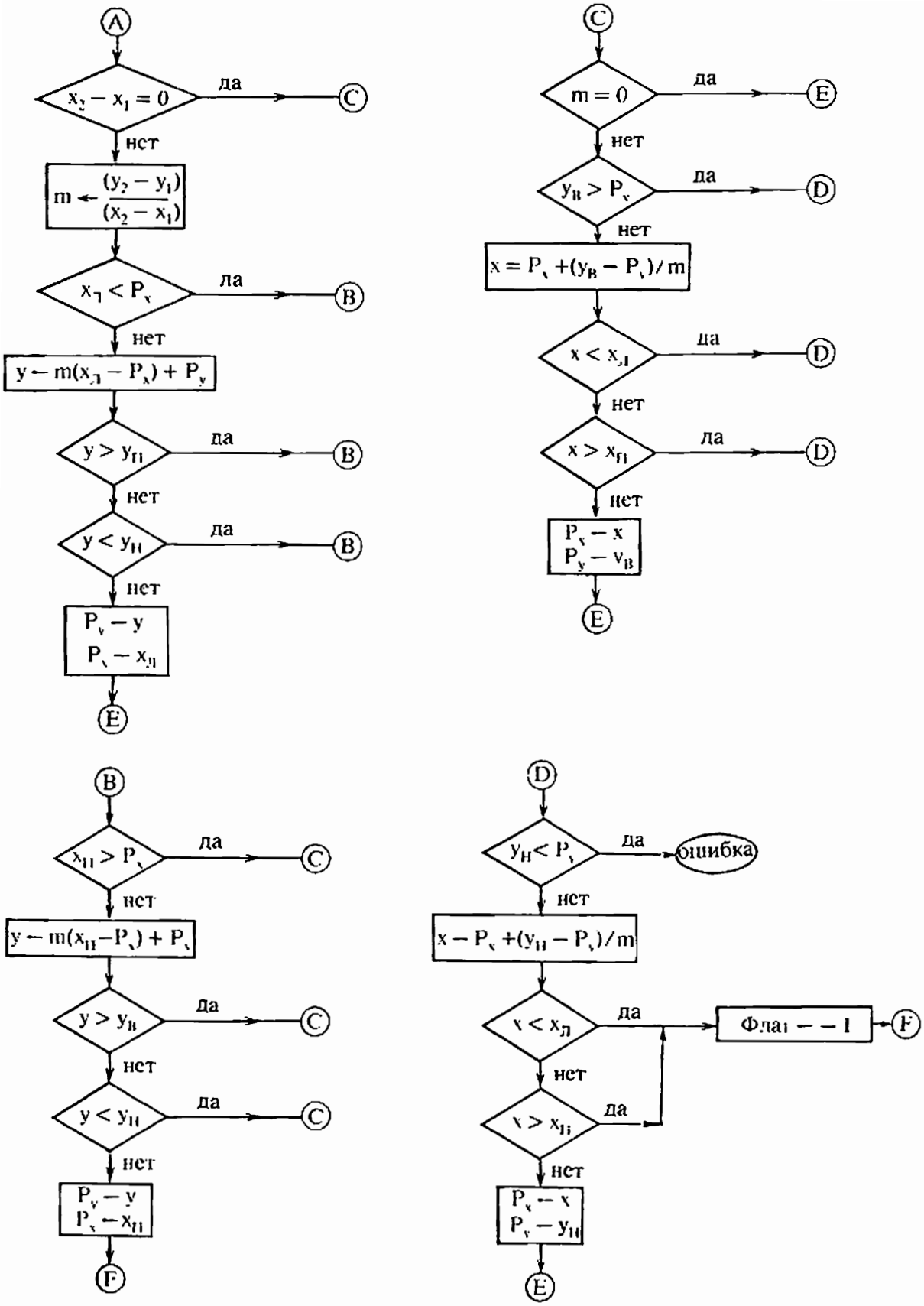



Рис. 3.4. Продолжение.

$x_Л$, $x_П$, $y_В$, $y_Н$ — *координаты левой, правой, верхней и нижней сторон окна*

Флаг — *признак видимости, равный:* 0, *видимость*; −1, *невидимость*

*вычисление кодов концевых точек*
*занесение этих кодов в два массива* T1код *и* T2код, *размерностью* 1 × 4 *каждый*
*для первого конца:* $P_1$
**if** $x_1 < x_Л$ **then** T1код(4) = 1 **else** T1код(4) = 0
**if** $x_1 > x_П$ **then** T1код(3) = 1 **else** T1код(3) = 0
**if** $y_1 < y_Н$ **then** T1код(2) = 1 **else** T1код(2) = 0
**if** $y_1 > y_В$ **then** T1код(1) = 1 **else** T1код(1) = 0
*для второго конца:* $P_2$
**if** $x_2 < x_Л$ **then** T2код(4) = 1 **else** T2код(4) = 0
**if** $x_2 > x_П$ **then** T2код(3) = 1 **else** T2код(3) = 0
**if** $y_2 < y_Н$ **then** T2код(2) = 1 **else** T2код(2) = 0
**if** $y_2 > y_В$ **then** T2код(1) = 1 **else** T2код(1) = 0
*инициализация признака видимости и видимых концевых точек*
*инициализация* m *очень большим числом, имитирующим бесконечный наклон*
Флаг = 0
$P_1' = P_1$
$P_2' = P_2$
m = Большое число
*проверка полной видимости отрезка*
Сумма1 = 0
Сумма2 = 0
**for** i = 1 **to** 4
  Сумма1 = Сумма1 + T1код(i)
  Сумма2 = Сумма2 + T2код(i)
**next** i
**if** Сумма1 = 0 **and** Сумма2 = 0 **then** 7
*отрезок не является полностью видимым*
*проверка случая тривиальной невидимости*
*вычисление логического произведения* (Произвед) *кодов концевых точек отрезка*
Произвед = 0
**for** i = 1 **to** 4
  Произвед = Произвед + Целая часть ((T1код(i) + T2код(i))/2)
  **if** Произвед < > 0 **then**
    Флаг = −1

```
            go to 7
        end if
    next i
    отрезок может быть частично видимым
    проверка попадания первой точки внутрь окна
    if    Сумма1 = 0 then
          Номер = 1
          P1' = P1
          P = P2
          go to 2
    end if
    проверка попадания второй точки внутрь окна
    if    Сумма2 = 0 then
          Номер = 2
          P1' = P2
          P = P1
          go to 2
    end if
    внутри окна нет концов отрезка
    инициализация номера конца отрезка
    Номер = 0
1   if Номер < > 0 then P'Номер = P
    Номер = Номер + 1
    if Номер > 2 then 7
    P = PНомер
    проверка пересечения с левым краем
    проверка вертикальности отрезка
2   if (x2 - x1) = 0 then 4
    m = (y2 - y1)/(x2 - x1)
    if xЛ < Px then 3
    y = m * (xЛ - Px) + Py
    if y > yВ then 3
    if y < yН then 3
    обнаружено корректное пересечение
    Py = y
    Px = xЛ
    go to 1
    проверка пересечения с правым краем
3   if xП > Px then 4
    y = m * (xП - Px) + Py
    if y > yВ then 4
```

```
   if y < y_Н then 4
   обнаружено корректное пересечение
   P_y = y
   P_x = x_П
   go to 1
   проверка пересечения с верхним краем
   проверка горизонтальности отрезка
4  if m = 0 then 1
   if y_В > P_y then 5
   x = (1/m) * (y_В − P_y) + P_x
   if x < x_Л then 5
   if x > x_П then 5
   обнаружено корректное пересечение
   P_x = x
   P_y = y_В
   go to 1
   проверка пересечения с нижним краем
5  if y_Н < P_y then ошибка
   x = (1/m) * (y_Н − P_y) + P_x
   if x < x_Л then 6
   if x > x_П then 6
   обнаружено корректное пересечение
   P_x = x
   P_y = y_Н
   go to 1
   отрезок невидим
6  Флаг = −1
   завершение работы и вызов процедуры черчения
7  if Флаг = −1 then 8
   Draw P'_1P'_2
   перейти к обработке следующего отрезка
8  finish
```

## 3.2. АЛГОРИТМ ОТСЕЧЕНИЯ САЗЕРЛЕНДА — КОЭНА, ОСНОВАННЫЙ НА РАЗБИЕНИИ ОТРЕЗКА

Алгоритм, описанный в предыдущем разделе, аналогичен тому, который предложили Коэн и Сазерленд. В предыдущем алгоритме отрезок отсекался поочередно каждой из сторон окна, а для полученных точек пересечения проверялась их принадлежность внутренней области окна, т. е. корректность пересечения. Эта процедура

применялась сначала к отрезку $P_1P_2$ и получался отрезок $P_1'P_2$, а затем к отрезку $P_1'P_2$ и получался результирующий отрезок $P_1'P_2'$.

В алгоритме Сазерленда — Коэна отрезок тоже разбивается сторонами окна. Отличие состоит в том, что здесь не производится проверки попадания точки пересечения внутрь окна, вместо этого каждая из пары получающихся частей отрезка сохраняется или отбрасывается в результате анализа кодов ее концевых точек. Рассмотрение отрезка $P_1P_2$ на рис. 3.3 сразу же демонстрирует трудность реализации этой простой идеи. Если $P_1P_2$ разбивается левой стороной окна, то получаются два новых отрезка, $P_1P_1'$ и $P_1'P_2$. Коды концевых точек каждого из этих отрезков таковы, что оба они могут быть частично видимы. Следовательно, ни один из них нельзя отвергнуть как невидимый или оставить как видимый. Ключом к алгоритму Сазерленда — Коэна является информация о том, что одна из концевых точек отрезка лежит вне окна. Поэтому тот отрезок, который заключен между этой точкой и точкой пересечения, можно отвергнуть как невидимый. Фактически это означает замену исходной концевой точки на точку пересечения. В содержательных понятиях алгоритм Сазерленда — Коэна формулируется следующим образом:

Для каждой стороны окна выполнить:

> Для каждого отрезка $P_1P_2$ определить, не является ли он полностью видимым или может быть тривиально отвергнут как невидимый.
>
> Если $P_1$ вне окна, то продолжить выполнение, иначе поменять $P_1$ и $P_2$ местами.
>
> Заменить $P_1$ на точку пересечения $P_1P_2$ со стороной окна.

Этот алгоритм иллюстрирует следующий пример.

**Пример 3.2.** Алгоритм отсечения Сазерленда — Коэна

Рассмотрим вновь отсечение отрезка $P_1P_2$ окном, показанным на рис. 3.3. Коды концевых точек $P_1(-3/2, 1/6)$ и $P_2(1/2, 1/2)$ равны (0001) и (1000) соответственно. Этот отрезок не является ни полностью видимым, ни тривиально невидимым.

Отрезок пересекает левую сторону окна. $P_1$ — вне окна.

Пересечение с левой стороной ($x = -1$) окна происходит в точке $P_1'(-1, 1/2)$. Замена $P_1$ на $P_1'$ дает новый отрезок от $P_1'(-1, 1/2)$ до $P_2(1/2, 1/2)$.

Коды концевых точек $P_1'$ и $P_2$ теперь стали (0000) и (1000) соответственно. Отрезок не является ни полностью видимым, ни тривиально невидимым.

Отрезок не пересекается с правой стороной окна. Перейти к нижней стороне.

Коды концевых точек $P_1$ и $P_2$ остаются по-прежнему равными (0000) и (1000) соответственно. Отрезок не является ни полностью видимым, ни тривиально невидимым.

Отрезок не пересекается с нижней стороной окна. Перейти к верхней стороне.

Коды концевых точек $P_1$ и $P_2$ остаются равными (0000) и (1000) соответственно. Отрезок не является ни полностью видимым, ни тривиально невидимым.

Отрезок пересекается с верхней стороной окна. $P_1$ — не снаружи окна. Поменяв $P_1$ и $P_2$ местами, мы получим новый отрезок от $P_1(1/2, 3/2)$ до $P_2(-1, 1/2)$.

Точка пересечения отрезка с верхней стороной окна ($y = 1$) равна $P_1(-1/4, 1)$. Заменив $P_1$ на $P_1$, получаем новый отрезок от $P_1(-1/4, 1)$ до $P_2(-1, 1/2)$.

Теперь коды концевых точек $P_1$ и $P_2$ равны (0000) и (0000) соответственно. Отрезок полностью виден.

Процедура завершена.

Начертить отрезок.

Запись этого алгоритма на псевдокоде приводится ниже. Поскольку в алгоритме неоднократно используются одни и те же операции, то введены подпрограммы.

### *Алгоритм двумерного отсечения Сазерленда — Коэна*

Окно — *массив* 1 × 4, *содержащий координаты* ($x_Л$, $x_П$, $y_Н$, $y_В$) *сторон окна*

$P_1$, $P_2$ — *концевые точки отрезка с координатами* ($P_1x$, $P_1y$) *и* ($P_2x$, $P_2y$) *соответственно*

Т1код, Т2код — *массивы* 1 × 4, *содержащие коды точек* $P_1$ *и* $P_2$

Флаг — *индикатор координатной ориентации отрезка, равный*: −1, *при вертикальности*, 0, *при горизонтальности*

*инициализация* Флаг
Флаг = 1
*проверка вертикальности и горизонтальности отрезка*
**if** $P_2x - P_1x = 0$ **then**
    Флаг = −1
**else**
    *вычисление наклона*
    Наклон = $(P_2y - P_1y)/(P_2x - P_1x)$
    **if** Наклон = 0 **then** Флаг = 0
**end if**
*для каждой стороны окна*

```
   for i = 1 to 4
      call Коэн (P1, P2, Окно; Видимость)
      if Видимость = да then 2
      if Видимость = нет then 3
      проверка пересечения отрезка и стороны окна
      if Т1код(5 − i) = Т2код(5 − i) then1
      проверка нахождения P1 вне окна; если P1 внутри окна, то
      поменять P1 и P2 местами
      if Т1код(5 − i) = 0 then
         Раб = Pi
         P1 = P2
         P2 = Раб
      end if
      поиск пересечений отрезка со сторонами окна
      выбор соответствующей подпрограммы вычисления пересе-
      чения
      контроль вертикальности отрезка
      if Флаг < > −1 и i ⩽ 2 then
         P1y = Наклон * (Окноi — P1x) + P1y
         P1x = Окноi
      else
         if Флаг < > 0 then
            if Флаг < > −1 then
               P1x = (1/Наклон) * (Окноi − P1y) + P1x
            end if
            P1y = Окноi
         end if
      end if
1  next i
   начертить видимый отрезок
2  Draw P1P2
3  finish
```

*подпрограмма определения видимости отрезка*

**subroutine** Коэн ($P_1$, $P_2$, Окно; Видимость)

$P_1$, $P_2$ — *концевые точки отрезка с координатами* ($P_1x$, $P_1y$) и ($P_2x$, $P_2y$) *соответственно.*

Окно — *массив* 1 × 4, *содержащий координаты* ($x_Л$, $x_П$, $y_Н$, $y_В$) *сторон окна*

Видимость — *признак видимости отрезка равный*: «*нет*»,

*«частично», «да», если отрезок соответственно: полностью невидим, видим частично или полностью видим*

*вычисление кодов концевых точек отрезка*
**call** Конец ($P_1$, Окно; Т1код, Сумма1)
**call** Конец ($P_2$, Окно; Т2код, Сумма2)
*предположим, что отрезок частично видим*
Видимость = частично
*проверка полной видимости отрезка*
**if** Сумма 1 = 0 и Сумма 2 = 0 **then**
  Видимость = да
**else**
  *проверка тривиальной невидимости отрезка*
  **call** Логическое (Т1код, Т2код, Произвед)
  **if** Произвед < > 0 **then** Видимость = нет
**end if**
*отрезок может оказаться частично видимым*
**return**

*подпрограмма вычисления кодов концевой точки отрезка*

**subroutine** Конец (Р, Окно; Ткод, Сумма)

$P_x$, $P_y$ — *координаты точки* Р
Окно — *массив* 1 × 4, *содержащий координаты* ($x_Л$, $x_П$, $y_Н$, $y_В$) *сторон окна*
Ткод — *массив* 1 × 4, *содержащий коды концевой точки*
Сумма — *сумма всех элементов массива* Ткод
*вычисление кодов концевой точки*
**if** $P_x < x_Л$ **then** Ткод(4) = 1 **else** Ткод(4) = 0
**if** $P_x > x_П$ **then** Ткод(3) = 1 **else** Ткод(3) = 0
**if** $P_y < y_Н$ **then** Ткод(2) = 1 **else** Ткод(2) = 0
**if** $P_y > y_В$ **then** Ткод(1) = 1 **else** Ткод(1) = 0
*вычисление суммы*
Сумма = 0
**for** i = 1 **to** 4
  Сумма = Сумма + Ткод(i)
**next** i
**return**

*подпрограмма вычисления логического произведения*

**subroutine** Логическое (Т1код, Т2код; Произвед)

T1код, T2код — *массивы* 1 × 4, *содержащие коды первой и второй концевых точек соответственно*

Произвед — *сумма побитовых логических произведений*

```
Произвед = 0
for i = 1 to 4
   Произвед = Произвед + Целая часть ((T1код(i) + T2код(i))/2)
next i
return
```

## 3.3. АЛГОРИТМ РАЗБИЕНИЯ СРЕДНЕЙ ТОЧКОЙ

В алгоритме из предыдущего раздела требовалось вычислить пересечение отрезка со стороной окна. Можно избежать непосредственного вычисления, если реализовать двоичный поиск такого пересечения путем деления отрезка его средней точкой. Алгоритм, основанный на этой идее и являющийся частным случаем алгоритма Сазерленда — Коэна, был предложен Спруллом и Сазерлендом [3-2] для аппаратной реализации. Программная реализация этого алгоритма медленнее, чем реализация предыдущего алгоритма. Аппаратная же реализация быстрее и эффективнее, поскольку можно использовать параллельную архитектуру, и, кроме того, аппаратные сложения и деления на 2 очень быстры. Деление на 2 аппаратно эквивалентно сдвигу каждого бита вправо. Например, четырехбитовый код десятичного числа 6 равен 0110. Побитовый сдвиг вправо на одну позицию дает 0011, что является кодом десятичного числа 3 = 6/2.

В алгоритме используются коды концевых точек отрезка и проверки, выявляющие полную видимость отрезков, например отрезка

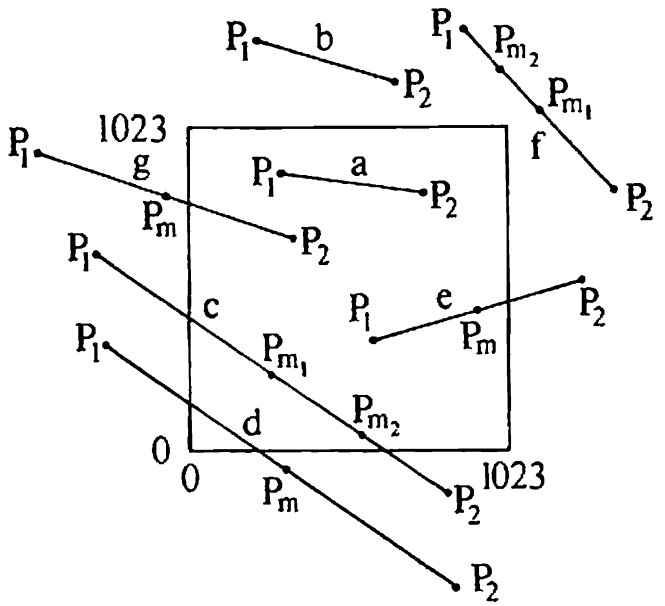



Рис. 3.5. Разбиение средней точкой.

$a$ на рис. 3.5, и тривиальную невидимость отрезков, например $b$ на рис. 3.5. Те отрезки, которые только с помощью таких простых проверок нельзя отнести к одной из двух категорий, например отрезки от $c$ до $g$ на рис. 3.5, разбиваются на две равные части. Затем те же проверки применяются к каждой из половин до тех пор, пока не будет обнаружено пересечение со стороной окна или длина разделяемого отрезка не станет пренебрежимо малой, т. е. пока он не выродится в точку. Эта идея проиллюстрирована на примере отрезка $f$ с рис. 3.5. После вырождения определяется видимость полученной точки. В результате процесс обнаружения пересечения сводится к двоичному поиску. Максимальное число разбиений пропорционально точности задания координат концевых точек отрезка.

Для иллюстрации изложенного метода рассмотрим отрезки $c$ и $f$ с рис. 3.5. Хотя отрезок $f$ невидим, он пересекает прямую, несущую диагональ окна, и не может быть тривиально отвергнут. Разбиение его средней точкой $P_{m_1}$ позволяет тривиально отвергнуть половину $P_{m_1} P_2$. Однако половина $P_{m_1} P_1$ тоже пересекает диагональ окна, и ее нельзя отвергнуть тривиально. Далее проводится разбиение точкой $P_{m_2}$, которое позволяет отвергнуть невидимый отрезок $P_{m_2} P_1$. Разбиение оставшегося куска $P_{m_1} P_{m_2}$ продолжается до тех пор, пока не будет найдено пересечение этого отрезка с прямой, несущей правую сторону окна, с наперед заданной точностью. Затем исследуется обнаруженная точка и она оказывается невидимой. Следовательно, и весь отрезок невидим.

Если судить только по кодам концевых точек, то отрезок на рис. 3.5 также не является ни полностью видимым, ни тривиально невидимым. Разбиение его средней точкой $P_{m_1}$ приводит к одинаковым результатам для обеих половин. Отложив отрезок $P_{m_1} P_{m_2}$ на потом, разобьем отрезок $P_{m_1} P_2$ точкой $P_{m_2}$. Теперь отрезок $P_{m_1} P_{m_2}$ полностью видим, а отрезок $P_{m_2} P_2$ видим частично. Отрезок $P_{m_1} P_{m_2}$ можно было бы начертить. Однако это привело бы к тому, что видимая часть отрезка изображалась бы неэффективно, как серия коротких кусков. Поэтому точка $P_{m_2}$ запоминается как текущая видимая точка, которая наиболее удалена от $P_1$. А разбиение отрезка $P_{m_2} P_2$ продолжается. Каждый раз, когда обнаруживается видимая средняя точка, она объявляется текущей наиболее удаленной от $P_1$, до тех пор пока не будет обнаружено пересечение с нижней стороной окна с заранее заданной точностью. Это пересечение и будет объявлено самой удаленной от $P_1$ видимой точкой. Затем точно так же обрабатывается отрезок $P_{m_1} P_1$. Для отрезка $c$ на рис. 3.5 наиболее удаленной от $P_2$ видимой точкой будет точка

его пересечения с левой стороной окна. Теперь можно начертить видимую часть отрезка $P_1 P_2$, заключенную между двумя найденными пересечениями.

Для отрезков, подобных *c* и *d* на рис. 3.5, в алгоритме разбиения средней точкой реализуется два двоичных поиска двух таких видимых точек, которые наиболее удалены от концов отрезка. Такими точками являются точки пересечения отрезка со сторонами окна. Каждое разбиение средней точкой дает грубую оценку искомых пересечений. Для отрезков, у которых виден один из концов, подобных *e* и *g*, один из двух поисков становится ненужным. При программной реализации алгоритма указанные процедуры поиска проводятся последовательно. А при аппаратной реализиции они проводятся параллельно. Данный алгоритм можно формально разбить на три этапа [3-3].

Для каждой концевой точки отрезка:

Если концевая точка видима, то она будет наиболее удаленной видимой точкой. Процесс завершен. Иначе — продолжить.

Если отрезок тривиально характеризуется как невидимый, то выходная информация не формируется. Процесс завершен. Иначе — продолжить.

Грубо оценить наиболее удаленную видимую точку путем деления отрезка $P_1 P_2$ средней точкой $P_m$. Применить вышеизложенные тесты к двум кускам $P_1 P_m$ и $P_m P_2$. Если $P_m P_2$ тривиально отвергается как невидимый, то средняя точка дает верхнюю оценку для наиболее удаленной видимой точки. Продолжить процедуру с отрезком $P_1 P_m$. Иначе — средняя точка дает оценку снизу для наиболее удаленной видимой точки. Продолжить процедуру с куском $P_2 P_m$. Если отрезок становится настолько мал, что его средняя точка совпадает с его концами с машинной или наперед заданной точностью, то надо оценить ее видимость и закончить процесс.

Конкретный пример лучше проиллюстрирует этот алгоритм.

**Пример 3.3.** Разбиение средней точкой

Рассмотрим окно, показанное на рис. 3.5 и имеющее экранные координаты левой, правой, нижней и верхней сторон: 0, 1023, 0, 1023 соответственно. Экранные координаты концов отрезка *c* равны: $P_1(-307, 631)$ и $P_2(820, -136)$. Коды концевых точек равны: для $P_1$ — (0001), а для $P_2$ — (0100). Оба кода не равны нулю, поэтому отрезок не является полностью видимым. Логическое произведение кодов [illegible]

чек равно (0000). Значит, отрезок нельзя тривиально отвергнуть как невидимый. Займемся поиском пересечений.

Координаты средней точки с учетом округления равны:

$$x_m = \frac{x_2 + x_1}{2} = \frac{820 - 307}{2} = 256.5 = 256$$

$$y_m = \frac{y_2 + y_1}{2} = \frac{-136 + 631}{2} = 247.5 = 247$$

Код средней точки равен (0000). Оба отрезка $P_1 P_m$ и $P_2 P_m$ не являются ни полностью видимыми, ни тривиально невидимыми. Отложим на время отрезок $P_2 P_m$ и займемся отрезком $P_1 P_m$. Процесс разбиения отрезков показан в табл. 3.2.

Таблица 3.2

| $P_1$ | $P_2$ | $P_m$ | Примечания |
|---|---|---|---|
| −307, 631 | 820, −136 | 256, 247 | Запомнить $P_m P_2$, Продолжить с $P_1 P_m$ |
| −307, 631 | 256, 247 | −26, 439 | Продолжить с $P_m P_2$ |
| −26, 439 | 256, 247 | 115, 343 | Продолжить с $P_1 P_m$ |
| −26, 439 | 115, 343 | 44, 391 | Продолжить с $P_1 P_m$ |
| −26, 439 | 44, 391 | 9, 415 | Продолжить с $P_1 P_m$ |
| −26, 439 | 9, 415 | −9, 427 | Продолжить с $P_m P_2$ |
| −9, 427 | 9, 415 | 0, 421 | Найдено пересечение |
| 256, 247 | 820, −136 | 538, 55 | Восстановить $P_m P_2$, Продолжить с $P_m P_2$ |
| 538, 55 | 820, −136 | 679, −41 | Продолжить с $P_1 P_m$ |
| 538, 55 | 679, −41 | 608, 7 | Продолжить с $P_m P_2$ |
| 608, 7 | 679, −41 | 643, −17 | Продолжить с $P_1 P_m$ |
| 608, 7 | 643, −17 | 625, −5 | Продолжить с $P_1 P_m$ |
| 608, 7 | 625, −5 | 616, 1 | Продолжить с $P_m P_2$ |
| 616, 1 | 625, −5 | 620, −2 | Продолжить с $P_1 P_m$ |
| 616, 1 | 620, −2 | 618, −1 | Продолжить с $P_1 P_m$ |
| 616, 1 | 618, −1 | 617, 0 | Найдено пересечение |

Использование точного уравнения отрезка $P_1 P_2$ дает пересечения в точках (0,422) и (620,0). Отличие этих значений от значений из табл. 3.2 объясняется погрешностью целочисленных округлений.

Блок-схема изложенного алгоритма дана на рис. 3.6. Запись его на псевдокоде приведена ниже.

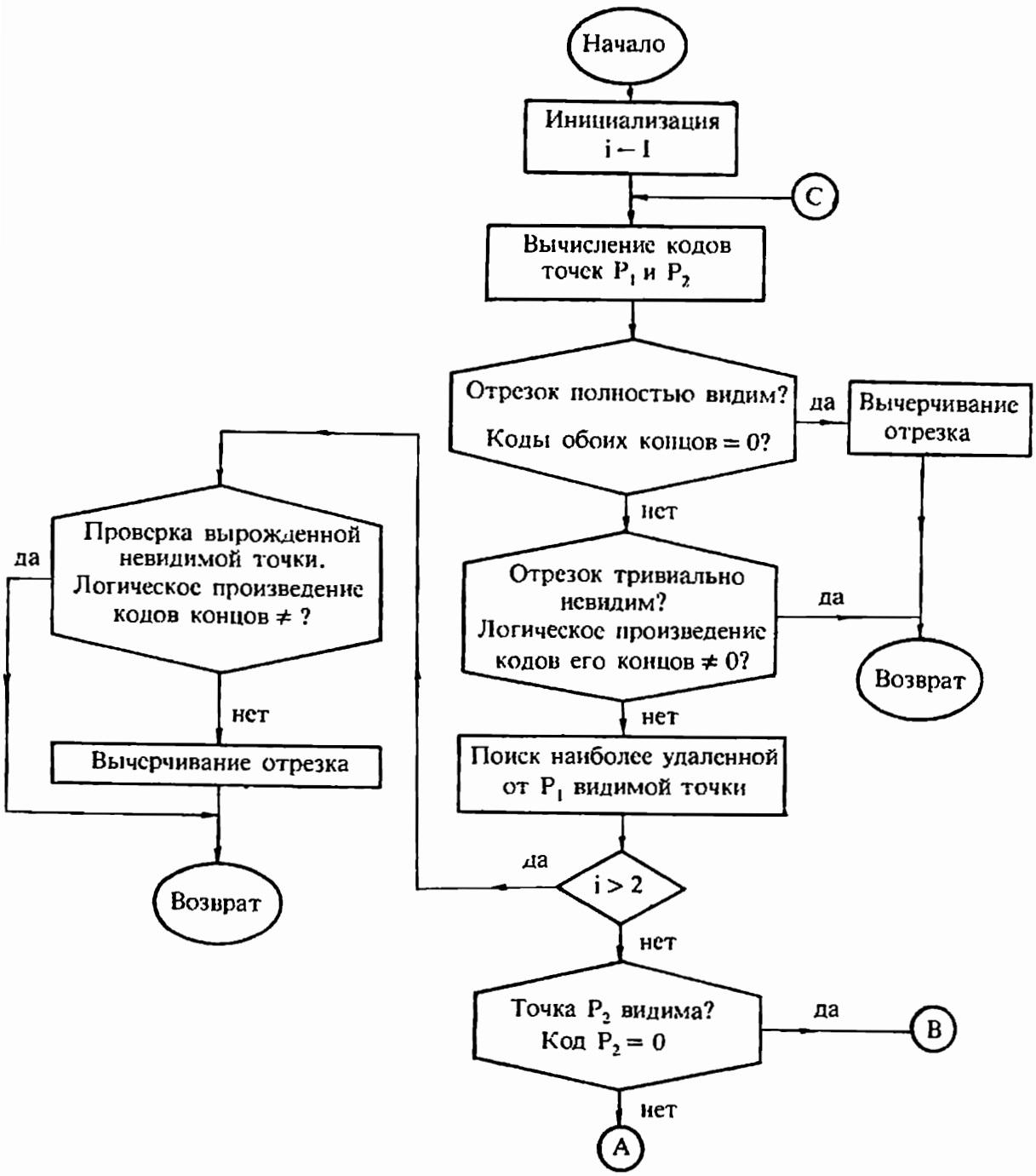
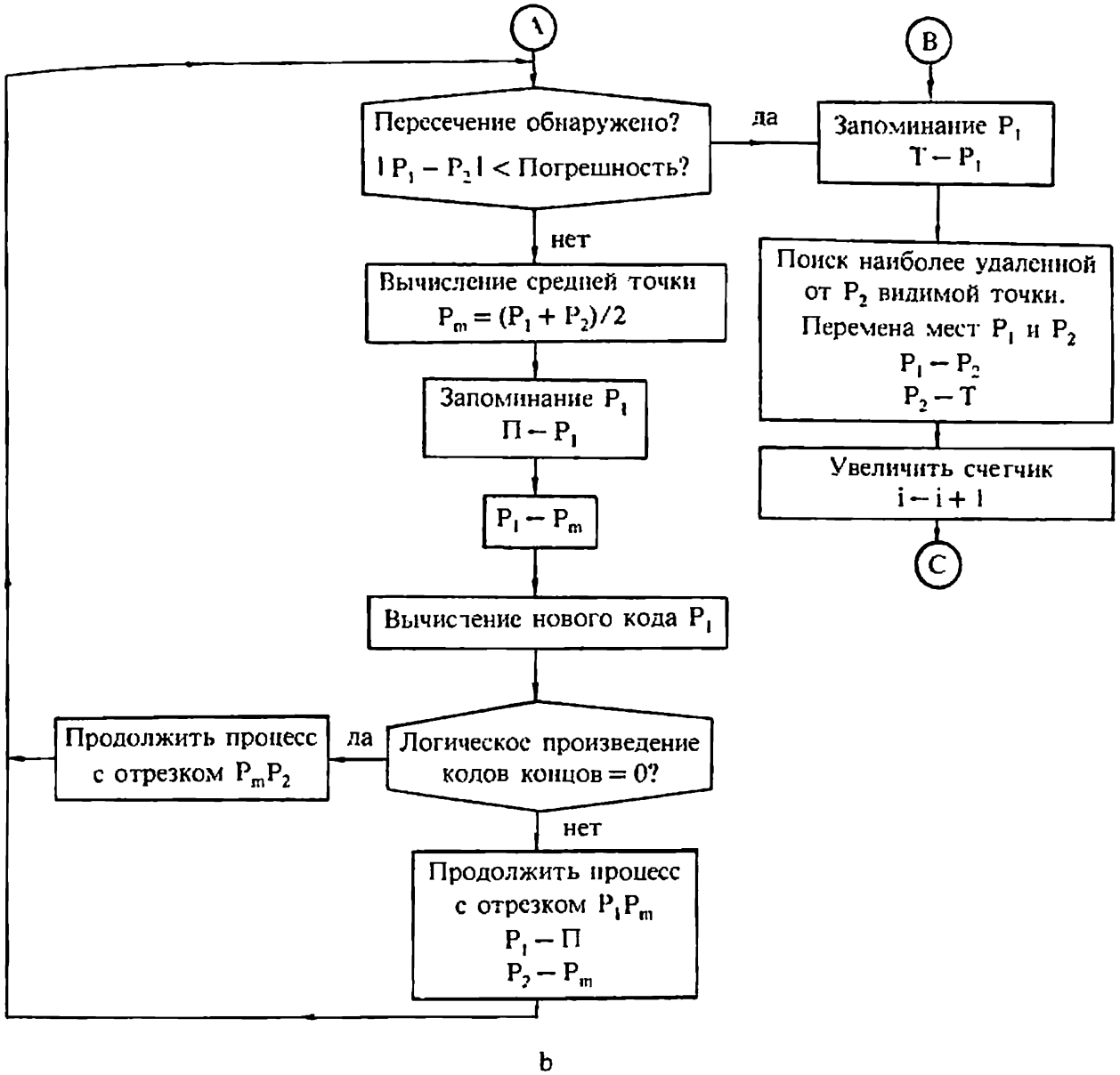



Рис. 3.6. Блок-схема алгоритма разбиения средней точкой.




b

Рис. 3.6. Продолжение.

## *Двумерный алгоритм отсечения методом разбиения отрезка средней точкой*

Окно — *массив* 1 × 4, *в котором содержатся координаты* ($x_Л$, $x_П$, $y_Н$, $y_В$) *левой, правой, нижней и верхней сторон прямоугольного отсекающего окна*

$P_1$, $P_2$ — *концевые точки отрезка*

*инициализация* i

i = 1

*вычисление кодов концевых точек*

*занесение кодов каждого конца в массивы* 1 × 4, *именуемые* T1код *и* T2код

*первая концевая точка*: $P_1$

```
1       call Конец(P1, Окно; T1код, Сумма1)
        вторая концевая точка: P2
        call Конец(P2, Окно; T2код, Сумма2)
        проверка полной видимости отрезка
        if Сумма1 = 0 и Сумма2 = 0 then 5
        отрезок не является полностью видимым
        проверка тривиальной невидимости отрезка
        call Логическое(T1код, T2код; Произвед)
        if Произвед < > 0 then 6
        отрезок может оказаться частично видимым
        поиск наиболее удаленной от P1 видимой точки отрезка
        запоминание исходной точки P1
        Раб = P1
        проверка окончания процесса решения
        if i > 2 then 4
        P2 — это наиболее удаленная от P1 видимая точка от-
        резка?
        if Сумма2 = 0 then 3
        пересечение уже найдено?
2       if | P1 − P2 | < Погрешность then 3
        вычисление средней точки
        Pm = (P1 + P2)/2
        запоминание текущей точки P1
        Память = P1
        замена P1 на Pm
        P1 = Pm
        вычисление нового кода точки P1
        call Конец (P1, Окно; T1код, Сумма1)
        проверка тривиальной невидимости отрезка PmP2
        call Логическое(T1код, T2код; Произвед)
        if Произвед = 0 then 2
        PmP2 невидим, продолжить процедуру с P1Pm
        P1 = Память
        P2 = Pm
        go to 2
        обнаружена наиболее удаленная от P1 видимая точка
        отрезка
        поиск наиболее удаленной от P2 видимой точки отрезка
        перемена мест P1 и P2
3       P1 = P2
        P2 = Раб
```

```
              увеличить счетчик
              i = i + 1
              начать процесс с «новым» укороченным отрезком
              go to 1
              теперь обнаружены оба пересечения
              проверка вырожденной невидимой точки
4             call Логическое(Т1код, Т2код; Произвед)
              if Произвед < > 0 then 6
5             Начертить отрезок
6             finish
```

*подпрограмма вычисления кодов концевой точки отрезка*

**subroutine** Конец(Р, Окно; Ткод, Сумма)

$Р_х$, $Р_у$ — *координаты* x *и* y *точки* Р
Окно — *массив* 1 × 4, *содержащий координаты* ($х_Л$, $х_П$, $у_Н$, $у_В$) *левой, правой, нижней и верхней сторон прямоугольного отсекающего окна*
Ткод — *массив* 1 × 4, *содержащий коды концевой точки*
Сумма — *поэлементная сумма* Ткод

```
              определение кодов концевой точки
              if Р_х < х_Л then Ткод(4) = 1 else Ткод(4) = 0
              if Р_х > х_П then Ткод(3) = 1 else Ткод(3) = 0
              if Р_у < у_Н then Ткод(2) = 1 else Ткод(2) = 0
              if Р_у > у_В then Ткод(1) = 1 else Ткод(1) = 0
              вычисление суммы кодов
              Сумма = 0
              for i = 1 to 4
                Сумма = Сумма + Ткод(i)
              next i
              return
```

*подпрограмма вычисления логического произведения*

**subroutine** Логическое(Т1код, Т2код; Произвед)
Т1код, Т2код — *массивы* 1 × 4, *содержащие коды концевых точек*
Произвед — *сумма битов в логическом произведении кодов концов*

```
  Произвед = 0
  for i = 1 to 4
    Произвед = Произвед + Целая часть((Т1код(i) +
    + Т2код(i))/2)
  next i
  return
```

В ранее описанном простом алгоритме отсечения коды концевых точек отрезка и их логическое произведение определялись прямо в теле алгоритма. В данном же алгоритме для этого используются подпрограммы, поскольку коды концов и их логическое произведение вычисляются неоднократно.

## 3.4. ОБОБЩЕНИЕ: ОТСЕЧЕНИЕ ДВУМЕРНОГО ОТРЕЗКА ВЫПУКЛЫМ ОКНОМ

В описанных выше алгоритмах предполагалось, что отсекающее окно является координатно ориентированным прямоугольником. Однако во многих случаях окно не таково. Например, предположим, что прямоугольное окно повернуто относительно системы координат так, как показано на рис. 3.7. В этом случае неприменим ни один из ранее описанных алгоритмов. Кирус и Бек предложили [3-4] алгоритм отсечения окном произвольной выпуклой формы.

Прежде чем подробно описывать алгоритм Кируса — Бека (это делается в следующем разделе), рассмотрим отсечение параметрически заданного отрезка прямоугольным окном. Параметрическое уравнение отрезка от $P_1$ до $P_2$ имеет вид

$$P(t) = P_1 + (P_2 - P_1)t \qquad 0 \leq t \leq 1 \tag{3.1}$$

где $t$ — параметр. Если ограничиться значениями $0 \leqslant t \leqslant 1$, то получается именно отрезок, а не бесконечная прямая. Параметрическое описание отрезка не зависит от выбора системы координат. Это свойство делает параметрическое представление особенно полезным для определения пересечения отрезка со стороной произ-

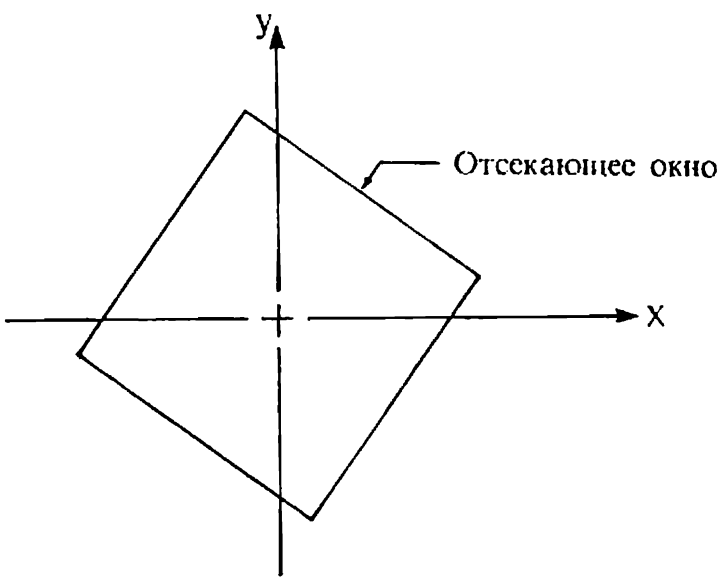



Рис. 3.7. Окно, повернутое относительно осей.

вольного выпуклого многоугольника. Проиллюстрируем этот метод сначала на примере регулярного прямоугольного окна.

В двумерной декартовой системе координат уравнение (3.1) сводится к паре одномерных параметрических уравнений вида:

$$x(t) = x_1 + (x_2 - x_1)t \qquad 0 \le t \le 1 \tag{3.2а}$$

$$y(t) = y_1 + (y_2 - y_1)t \qquad 0 \le t \le 1 \tag{3.2в}$$

В случае прямоугольного отсекающего окна одна из координат пересечения отрезка с каждой стороной известна. Нужно вычислить только вторую координату. Из уравнения (3.1) получаем: $t = (P(t) - P_1)/(P_2 - P_1)$. А из (3.2) следует, что значения $t$, соответствующие пересечениям со сторонами окна, равны:

для левой стороны: $$t = \frac{x_л - x_1}{x_2 - x_1} \qquad 0 \le t \le 1$$

для правой стороны: $$t = \frac{x_п - x_1}{x_2 - x_1} \qquad 0 \le t \le 1$$

для нижней стороны: $$t = \frac{y_н - y_1}{y_2 - y_1} \qquad 0 \le t \le 1$$

для верхней стороны: $$t = \frac{y_в - y_1}{y_2 - y_1} \qquad 0 \le t \le 1$$

Здесь через $x_л$, $x_п$, $y_н$, $y_в$ обозначены координаты левой, правой, нижней и верхней сторон окна соответственно. Если решения этих уравнений дают значения $t$ за пределами интервала $0 \leqslant t \leqslant 1$, то такие решения отвергаются, поскольку они соответствуют точкам, лежащим вне исходного отрезка.

**Пример 3.4.** Простой частично видимый отрезок

Рассмотрим частично видимый отрезок от $P_1(-3/2, -3/4)$ до $P_2'(3/2, 1/2)$, отсекаемый окном с координатами сторон $x_л$, $x_п$, $y_н$, $y_в$, равными $(-1, 1, -1, 1)$, как показано на рис. 3.8. Имеем значения $t$ для точек пересечения отрезка

с левой стороной: $$t = \frac{x_л - x_1}{x_2 - x_1} = \frac{-1 - (-3/2)}{3/2 - (-3/2)} = \frac{1/2}{3} = \frac{1}{6}$$

с правой стороной: $$t = \frac{x_п - x_1}{x_2 - x_1} = \frac{1 - (-3/2)}{3/2 - (-3/2)} = \frac{5/2}{3} = \frac{5}{6}$$

с нижней стороной: $$t = \frac{y_н - y_1}{y_2 - y_1} = \frac{-1 - (-3/4)}{1/2 - (-3/4)} = \frac{-1/4}{5/4} = \frac{-1}{5}$$

(это число меньше нуля и поэтому отвергается)
с верхней стороной:

$$t = \frac{y_B - y_1}{y_2 - y_1} = \frac{1 - (-3/4)}{1/2 - (-3/4)} = \frac{7/4}{5/4} = \frac{7}{5}$$

(это число больше единицы и также отвергается).

Видимый участок отрезка заключен в интервале $1/6 \leqslant t \leqslant 5/6$.

Значения $x$ и $y$ координат точек пересечения получаются из параметрических уравнений. В частности, при $t = 1/6$ из уравнения (3.2) имеем:

$$x(1/6) = -3/2 + [3/2 - (-3/2)](1/6) = -1$$

что, разумеется, априори известно, поскольку $x = -1$ — это абсцисса левой стороны окна. Для координаты $y$ имеем:

$$y(1/6) = -3/4 + [1/2 - (-3/4)](1/6) = -13/24$$

Аналогично при $t = 5/6$ имеем:

$$\begin{aligned}[x(5/6) \quad y(5/6)] &= [-3/2 \quad -3/4] + [3/2 - (-3/2) \quad 1/2 - (-3/4)]\,(5/6) \\ &= [1 \quad 7/24]\end{aligned}$$

Здесь вычисления значений $x$ и $y$ объединены. Опять, поскольку отрезок пересекает правую сторону окна, то координата $x$ априори известна.

После изложенного примера может показаться, что предлагаемый метод прост и естествен. Однако существует ряд трудностей, которые лучше продемонстрировать на следующих примерах.

**Пример 3.5.** Частично видимый отрезок

Рассмотрим отрезок от $P_3(-5/2, -1)$ до $P_4(3/2, 2)$, который тоже показан на рис. 3.8, а отсекающее окно опять имеет координаты сторон $(-1, 1, -1, 1)$. Теперь точки пересечения задаются следующими значениями параметра:

$$t_Л = 3/8,\ t_П = 7/8,\ t_Н = 0,\ t_В = 2/3,$$

и все эти четыре значения принадлежат интервалу $0 \leqslant t \leqslant 1$.

Хорошо известно, что если отрезок прямой пересекает выпуклый многоугольник, то возникает не более двух точек пересечения. Следовательно, нужны только два из четырех значений параметра, вычисленных в примере 3.5. Упорядочение этих четырех значений по возрастанию $t$ дает последовательность $t_Н$, $t_Л$, $t_В$, $t_П$. На рис. 3.8 показано, что искомыми являются значения $t_Л = 3/8$ и $t_В = 2/3$, соответствующие точкам пересечения $(-1, 1/8)$ и $(1/6, 1)$. Эти значения совпадают с $t_{maxmin}$ и $t_{minmax}$. Формальное вычисление этих значений сводится к простой классической задаче линейного

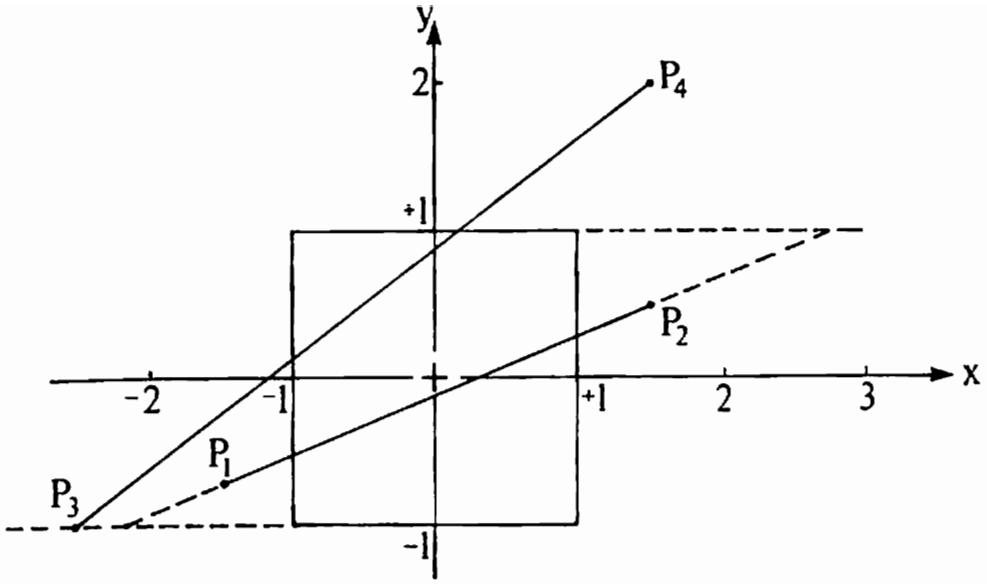



Рис. 3.8. Параметрическое отсечение частично видимых отрезков.

программирования. Один алгоритм решения этой задачи дан в следующем разделе.

Как и в любом алгоритме отсечения, здесь важно уметь быстро выявлять и отделять полностью видимые и полностью невидимые отрезки. Следующие два примера проиллюстрируют ряд новых трудностей.

**Пример 3.6.** Полностью видимый отрезок

Рассмотрим полностью видимый отрезок от $P_1(-1/2, 1/2)$ до $P_2(1/2, -1/2)$, опять отсекаемый окном со сторонами $(-1, 1, -1, 1)$, показанным на рис. 3.9. Значения параметров, соответствующие пересечениям отрезка со сторонами окна, равны:

$$t_Л = -1/2, \quad t_П = 3/2, \quad t_Н = 3/2, \quad t_В = -1/2.$$

Все эти значения находятся вне интервала $0 \leqslant t \leqslant 1$.

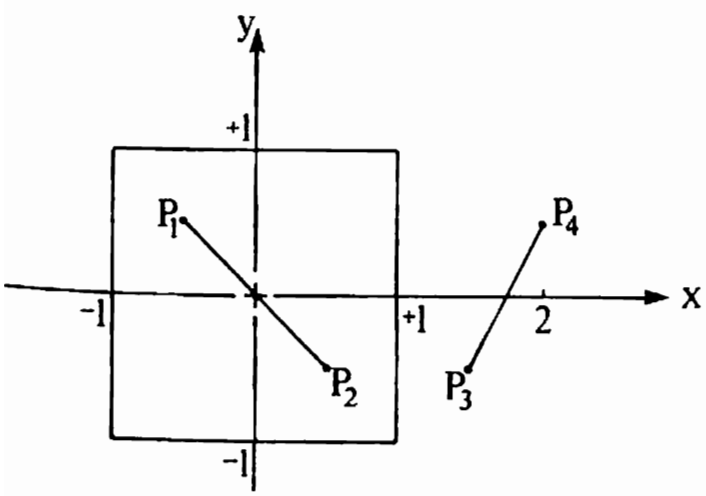



Рис. 3.9. Отсечение полностью видимых и невидимых отрезков.

Из анализа примера 3.6 может показаться, что найден метод определения полностью видимых отрезков. Однако следующий пример показывает, что это неверно.

**Пример 3.7.** Полностью невидимый отрезок

Рассмотрим полностью невидимый отрезок от $P_3(3/2, -1/2)$ до $P_4(2, 1/2)$, который тоже показан на рис. 3.9. Отсекающее окно опять задано сторонами $(-1, 1, -1, 1)$. Здесь значения параметра, соответствующие пересечениям отрезка со сторонами окна, равны:

$$t_Л = -5, \quad t_П = -1, \quad t_Н = -1/2, \quad t_В = 3/2.$$

И опять все эти значения находятся вне интервала 0.

Из примера 3.7 следует, что значения параметра удовлетворяют ранее определенным условиям полной видимости отрезка. Однако в отличие от отрезка $P_1 P_2$ из примера 3.6 отрезок $P_3 P_4$ невидим. Из последующих двух примеров следует, что для параметрически заданных отрезков нет простого и единственного метода, различающего полностью видимые и полностью невидимые отрезки. Кроме того, ясно, что эта задача требует более формального подхода.

## 3.5. АЛГОРИТМ КИРУСА — БЕКА

Для создания надежного алгоритма отсечения нужно иметь хороший метод определения местоположения относительно окна (внутри, на границе или вне его) точки, принадлежащей отрезку. Для этой цели в алгоритме Кируса — Бека [3-4] используется вектор нормали.

Возьмем выпуклую отсекающую область $R$. Хотя $R$ не обязана быть двумерной, в примерах, приведенных в данном разделе, предполагается, что она двумерна. Итак, $R$ может быть любым плоским выпуклым многоугольником. Она *не должна быть* вогнутым многоугольником. Внутренняя нормаль $\mathbf{n}$ в произвольной точке $\mathbf{a}$, лежащей на границе $R$, удовлетворяет условию: $n \cdot (\mathbf{b} - \mathbf{a}) \geqslant 0$, где $\mathbf{b}$ — любая другая точка на границе $R$. Чтобы убедиться в этом, вспомним, что скалярное произведение двух векторов $\mathbf{V}_1$ и $\mathbf{V}_2$ равно: $\mathbf{V}_1\mathbf{V}_2 \cdot |\mathbf{V}_1|\,|\mathbf{V}_2| \cos\theta$, где $\theta$ — это меньший из двух углов, образованных $\mathbf{V}_1$ и $\mathbf{V}_2$. Заметим, что если $\theta = \pi/2$, то $\cos\theta = 0$ и $\mathbf{V}_1 \cdot \mathbf{V}_2 = 0$, т. е. когда скалярное произведение пары векторов равно нулю, то они перпендикулярны. На рис. 3.10 показана выпуклая область $R$ т. е. отсекающее окно. Там же показаны внешняя (наружная) $\mathbf{n}_н$ и внутренняя $\mathbf{n}_в$ нормали к границе окна, исходящие из

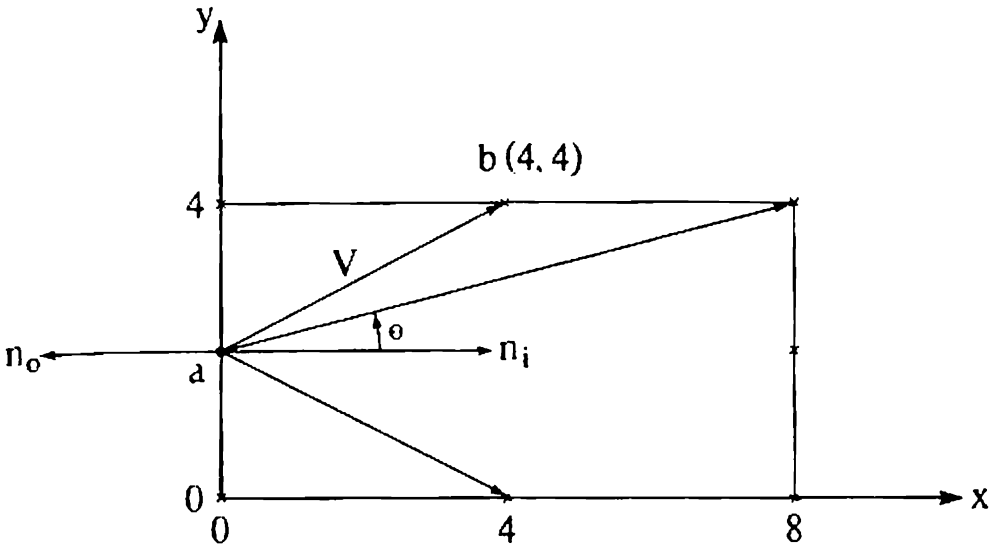



Рис. 3.10. Внутренняя и внешняя (наружная) нормали.

точки **a**, лежащей на этой границе. Кроме того, на рис. 3.10 показаны еще несколько векторов, проведенных из точки **a** в другие точки на границе окна. Угол между $\mathbf{n}_в$ и любым из таких векторов всегда принадлежит интервалу $-\pi/2 \leqslant \theta \leqslant \pi/2$. При таких значениях угла косинус его всегда положителен. Следовательно, положительно и скалярное произведение этих векторов, как было установлено выше. А вот угол между внешней нормалью и любым из подобных векторов равен $\pi - \theta$, а $\cos(\pi - \theta) = -\cos\theta$ при этом отрицателен. Для лучшего понимания этого факта рассмотрим следующий пример.

**Пример 3.8.** Внешняя и внутренняя нормали

Возьмем прямоугольную область, приведенную на рис. 3.10. Здесь внутренняя и внешняя нормали в точке *a* равны $\mathbf{n}_в = \mathbf{i}$ и $\mathbf{n}_н = -\mathbf{i}$ соответственно, где **i** — единичный вектор вдоль оси *x*. В табл. 3.3 показаны значения скалярных произведений

**Таблица 3.3.**

| *a* | *b* | $\mathbf{n}_в \cdot (\mathbf{b} - \mathbf{a})$ | $\mathbf{n}_н \cdot (\mathbf{b} - \mathbf{a})$ |
|---|---|---|---|
| (0, 2) | (0, 4) | 0 | 0 |
| | (4, 4) | 4 | −4 |
| | (8, 4) | 8 | −8 |
| | (8, 2) | 8 | −8 |
| | (8, 0) | 8 | −8 |
| | (4, 0) | 4 | −4 |
| | (0, 0) | 0 | 0 |

внешней и внутренней нормалей на векторы, проведенные из точки *a* в разные точки *в*, лежащие на границе области. Особо выделим случай точки *в* (4,4), тогда $\mathbf{b} - \mathbf{a} = 4\mathbf{i} + 2\mathbf{j}$, а $\mathbf{n}_в \cdot (\mathbf{b} - \mathbf{a}) = \mathbf{i} \cdot (4\mathbf{i} + 2\mathbf{j}) = 4$. Нули в последнем столбце табл. 3.3 означают, что соответствующие векторы перпендикулярны нормалям.

Возвращаясь к определению пересечения отрезка со стороной окна, вновь возьмем параметрическое представление отрезка $P_1 P_2$:

$$P(t) = P_1 + (P_2 - P_1)\, t,\ 0 \leqslant t \leqslant 1.$$

Если $\mathbf{f}$ — граничная точка выпуклой области $R$, а $\mathbf{n}$ — внутренняя нормаль к одной из ограничивающих эту область плоскостей, то для любой конкретной величины $t$, т. е. для любой точки отрезка $P_1 P_2$, из условия $\mathbf{n} \cdot [\mathbf{P}(t) - \mathbf{f}] < 0$ следует, что вектор $\mathbf{P}(t) - \mathbf{f}$ направлен вовне области $R$. Из условия $\mathbf{n} \cdot [\mathbf{P}(t) - \mathbf{f}] = 0$ следует, что $\mathbf{P}(t) - \mathbf{f}$ принадлежит плоскости, которая проходит через $\mathbf{f}$ и перпендикулярна нормали. Из условия $\mathbf{n} \cdot [\mathbf{P}(t) - \mathbf{f}] > 0$ следует, что вектор $\mathbf{P}(t) - \mathbf{f}$ направлен внутрь $R$, как показано на рис. 3.11. Из всех этих условий взятых вместе следует, что бесконечная прямая пересекает замкнутую выпуклую область, которая в двумерном случае сводится к замкнутому выпуклому многоугольнику ровно в двух точках. Далее, пусть эти две точки не принадлежат одной граничной плоскости или ребру. Тогда уравнение $\mathbf{n} \cdot [\mathbf{P}(t) - \mathbf{f}] = 0$ имеет только одно решение. Если точка $f$ лежит на той граничной плоскости или на том ребре, для которых $\mathbf{n}$ является внутренней нормалью, то точка на отрезке $P(t)$, которая удовлетворяет последнему уравнению, будет точкой пересечения этого отрезка с указанной граничной плоскостью.

**Пример 3.9.** Алгоритм Кируса — Бека. Частично видимый отрезок

Рассмотрим отрезок от $P_1(-1, 1)$ до $P_2(9, 3)$, отсекаемый прямоугольным окном, показанным на рис. 3.12. Уравнение прямой, несущей $P_1 P_2$, имеет вид

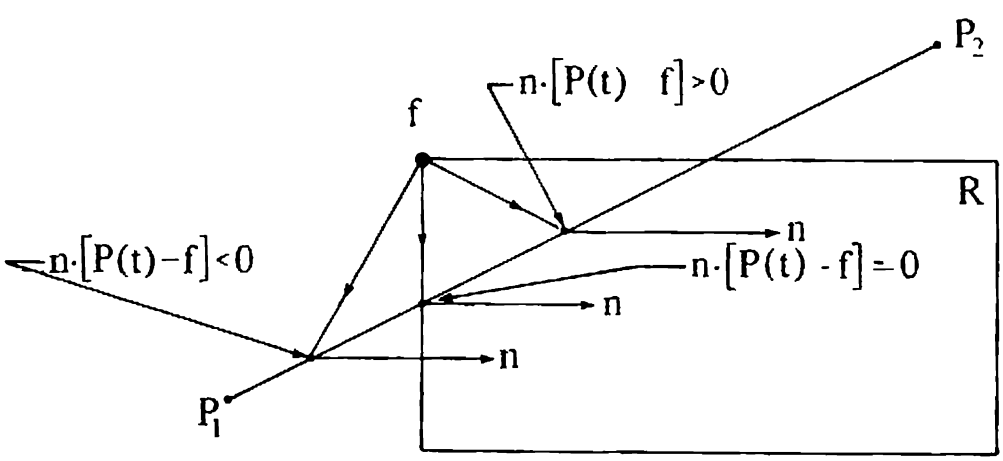



Рис. 3.11. Направления векторов.

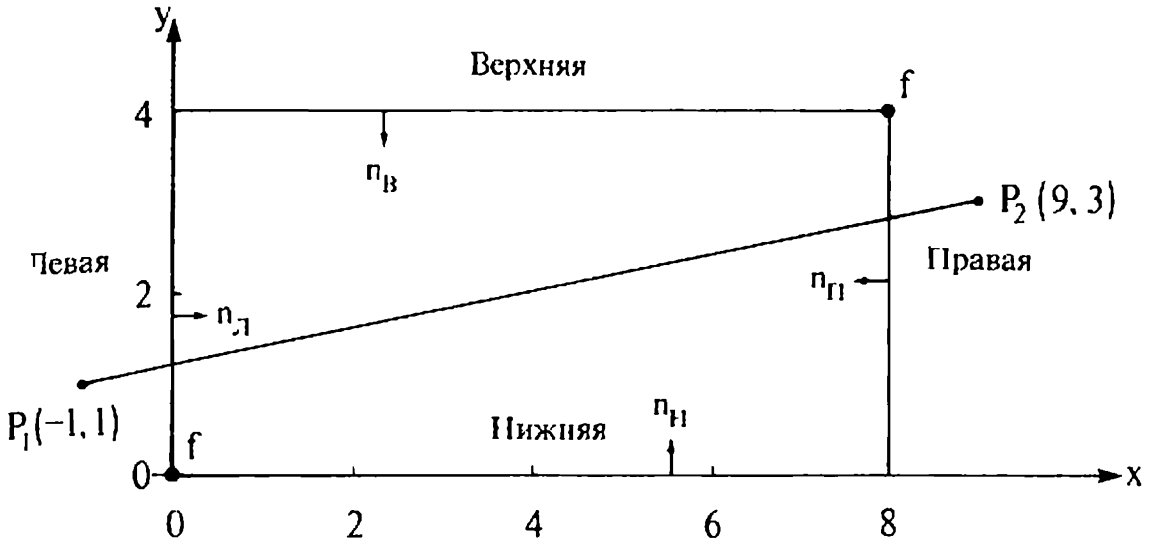



Рис. 3.12. Отсечение Кируса — Бека: частично видимый отрезок.

$y = 0.2(x - 6)$, и она пересекает окно в точках (0, 1.2) и (8, 2.8). Параметрическое описание отрезка $P_1 P_2$ таково:

$$P(t) = P_1 + (P_2 - P_1)t = [-1 \quad 1] + [10 \quad 2]t$$
$$= (10t - 1)\mathbf{i} + (2t + 1)\mathbf{j} \qquad 0 \leq t \leq 1$$

здесь $\mathbf{i}$ и $\mathbf{j}$ — единичные векторы, ориентированные вдоль осей $x$ и $y$ соответственно. Четыре внутренние нормали к сторонам окна равны: левая $\mathbf{n}_Л = \mathbf{i}$, правая $\mathbf{n}_П = -\mathbf{i}$, нижняя $\mathbf{n}_Н = \mathbf{j}$, верхняя $\mathbf{n}_В = -\mathbf{j}$.

Выбрав точку $f(0,0)$ на левой стороне окна, имеем $\mathbf{P(t)} - \mathbf{f} = (10t - 1)\mathbf{i} + (2t + 1)\mathbf{j}$, а уравнение $\mathbf{n}_Л \cdot [\mathbf{P(t)} - \mathbf{f}] = 10t - 1 = 0$ дает значение $t = 1/10$ для пересечения отрезка с левой стороной окна. Подстановка $\mathbf{P}(1\ 10) = [-1\ 1] + [10\ 2](1/10) = [0\ 1.2]$ дает то же самое значение, которое было вычислено выше другим методом.

Выбрав точку $f(8, 4)$ на правой стороне окна, имеем $\mathbf{P(t)} - \mathbf{f} = (10t - 9)\mathbf{i} + (2t - 3)\mathbf{j}$, а уравнение $\mathbf{n}_П \cdot [\mathbf{P(t)} - \mathbf{f}] = -(10t - 9) = 0$ дает значение $t = 9/10$ для пересечения отрезка с выбранной стороной. Подстановка дает $P(9/10) = [-1\ 1] + [10\ 2](9/10) = [8\ 2.8]$, что также совпадает с результатом вычисления другим методом.

Использование точки $f(0, 0)$ на нижней стороне окна приводит к уравнению $\mathbf{n}_Н \cdot [\mathbf{P(t)} - \mathbf{f}] = 2t + 1 = 0$, решением которого является значение $t = -1/2$. Оно лежит вне интервала $0 \leqslant t = 1$ и поэтому отвергается.

Использование точки $f(8, 4)$ на верхней стороне окна дает уравнение $\mathbf{n}_В \cdot [\mathbf{P(t)} - \mathbf{f}] = -(2t - 3) = 0$ с решением $t = 3/2$. Последнее значение $t$ тоже лежит вне интервала $0 \leqslant t \leqslant 1$ и также отвергается. Видимый участок отрезка $P_1 P_2$, отсекаемый прямоугольной областью, показанной на рис. 3.12, лежит в интервале $1/10 \leqslant t \leqslant 9/10$ или от точки [0 1.2] до точки [8 2.8].

Из последнего примера следует, что точки пересечения можно отыскать без большого труда. Способы идентификации полностью видимого и полностью невидимого отрезков продемонстрированы в следующих трех примерах.

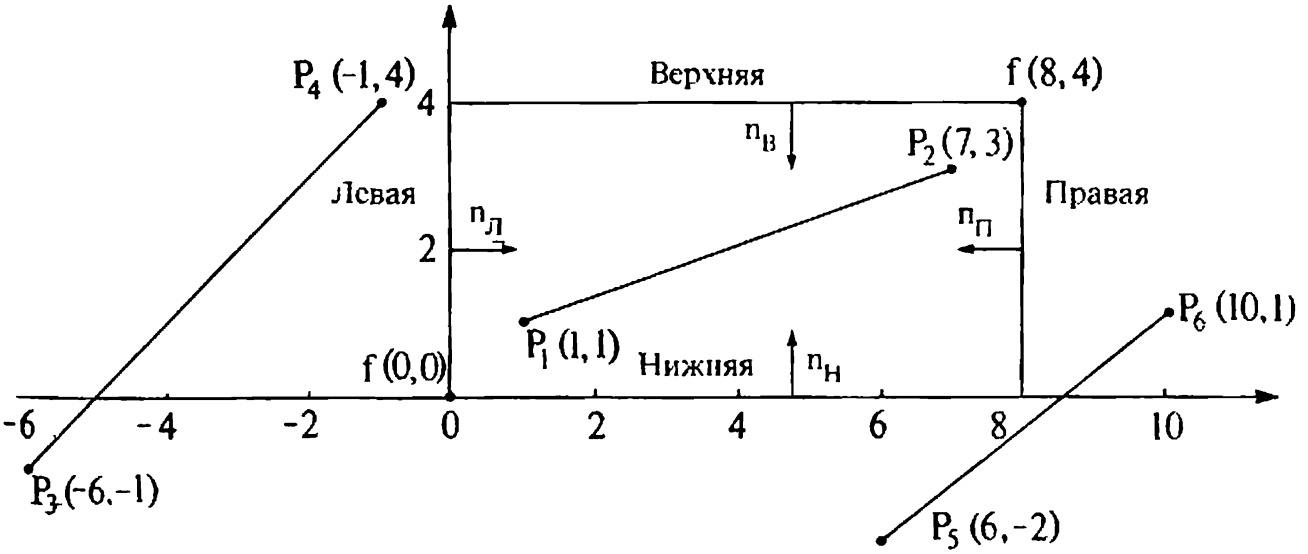



Рис. 3.13. Отсечение Кируса — Бека: полностью видимый и невидимый отрезки.

**Пример 3.10.** Алгоритм Кируса — Бека. Полностью видимый отрезок

Рассмотрим отрезок от $P_1(1, 1)$ до $P_2(7, 3)$, отсекаемый прямоугольным окном, показанным на рис. 3.13. Его параметрическое описание имеет вид $P(t) = [1\ 1] + [6\ 2]\,t$.

Результаты вычислений, использующих те же внутренние нормали и граничные точки, что и в примере 3.9, приведены в табл. 3.4. Все значения $t$, которые соответствуют точкам пересечения лежат вне интервала $0 \leqslant t \leqslant 1$. Значит, отрезок полностью видимый.

**Таблица 3.4.**

| Ребро | n | f | P(t)−f | n·[P(t)−f] | t |
|---|---|---|---|---|---|
| Левая | i | (0, 0) | $(1 + 6t)\,\mathbf{i} + (1 + 2t)\,\mathbf{j}$ | $1 + 6t$ | −1/6 |
| Правая | −i | (8, 4) | $(-7 + 6t)\,\mathbf{i} + (-3 + 2t)\,\mathbf{j}$ | $7 - 6t$ | 7/6 |
| Нижняя | j | (0, 0) | $(1 + 6t)\,\mathbf{i} + (1 + 2t)\,\mathbf{j}$ | $1 + 2t$ | −1/2 |
| Верхняя | −j | (8, 4) | $(-7 + 6t)\,\mathbf{i} + (-3 + 2t)\,\mathbf{j}$ | $3 - 2t$ | 3/2 |

В следующих двух примерах рассматриваются два типа невидимых отрезков. Один их этих отрезков целиком расположен слева от окна, и его невидимость может быть установлена анализом кодов концевых точек, который был описан ранее. Второй отрезок пересекает прямую, несущую диагональ окна за его пределами. Его нельзя объявить невидимым, используя только коды концевых точек.

**Пример 3. 11.** Алгоритм Кируса — Бека. Тривиально невидимый отрезок

Рассмотрим отрезок от $P_3(-6, -1)$ до $P_4(-1, 4)$, отсекаемый прямоугольным окном, показанным на рис. 3.13. Этот отрезок невидим. Его параметрическое представление имеет вид: $P(t) = [-6\ -1] + [5\ 5]\,t$.

Результаты вычислений, использующих те же внутренние нормали и граничные точки, которые использовались в предыдущем примере, приведены в табл. 3.5.

Анализ результатов из табл. 3.5 показывает, что оба значения параметров для точек пересечения с левой и правой сторонами лежат вне интервала $0 \leqslant t \leqslant 1$, а оба значения для пересечения с нижней и верхней сторонами лежат внутри интервала $0 \leqslant t \leqslant 1$. Поэтому сначала можно было бы предположить, что отрезок видим в интервале $1/5 \leqslant t \leqslant 1$. Однако дальнейший анализ пересечений с левой и правой сторонами показывает, что оба значения параметра больше единицы. Отсюда следует, что окно полностью расположено справа от отрезка. Значит, отрезок невидим.

**Таблица 3.5.**

| Ребро | n | f | P(t)−f | n · [P(t)−f] | t |
|---|---|---|---|---|---|
| Левая | i | (0, 0) | $(-6 + 5t)\,\mathbf{i} + (-1 + 5t)\,\mathbf{j}$ | $-6 + 5t$ | $t = 6/5$ |
| Правая | −i | (8, 4) | $(-14 + 5t)\,\mathbf{i} + (-5 + 5t)\,\mathbf{j}$ | $-(-14 + 5t)$ | $t = 14/5$ |
| Нижняя | j | (0, 0) | $(-6 + 5t)\,\mathbf{i} + (-1 + 5t)\,\mathbf{j}$ | $-1 + 5t$ | $t = 1/5$ |
| Верхняя | −j | (8, 4) | $(-14 + 5t)\,\mathbf{i} + (-5 + 5t)\,\mathbf{j}$ | $-(-5 + 5t)$ | $t = 1$ |

Если в последнем примере поменять $P_3$ и $P_4$ местами, то окно будет полностью расположено слева от отрезка. Ориентация отрезка играет важную роль при принятии решения о его невидимости. В следующем примере этот вопрос исследуется полнее.

**Пример 3.12.** Алгоритм Кируса — Бека. Нетривиально невидимый отрезок.

Здесь рассматривается отрезок от $P_5(6, -2)$ до $P_6(10, 1)$, который снова отсекается прямоугольным окном, показанным на рис. 3.13. Параметрическое описание отрезка имеет вид $P(t) = [6\ -2] + [4\ 3]\,t$.

Использование внутренних нормалей и граничных точек, взятых из предыдущих примеров, приводит к результатам, приведенным в табл. 3.6.

**Таблица 3.6.**

| Ребро | n | f | P(t)−f | n · [P(t)−f] | t |
|---|---|---|---|---|---|
| Левая | i | (0, 0) | $(6 + 4t)\,\mathbf{i} + (-2 + 3t)\,\mathbf{j}$ | $6 + 4t$ | $t = -3/2$ |
| Правая | −i | (8, 4) | $(-2 + 4t)\,\mathbf{i} + (-6 + 3t)\,\mathbf{j}$ | $-(-2 + 4t)$ | $t = 1/2$ |
| Нижняя | j | (0, 0) | $(6 + 4t)\,\mathbf{i} + (-2 + 3t)\,\mathbf{j}$ | $-2 + 3t$ | $t = 2/3$ |
| Верхняя | −j | (8, 4) | $(-2 + 4t)\,\mathbf{i} + (-6 + 3t)\,\mathbf{j}$ | $-(-6 + 3t)$ | $t = 2$ |

Эти результаты свидетельствуют о том, что значения параметра для пересечений с левой и верхней сторонами окна выходят за допустимые пределы. Однако значения параметра для пересечений с правой и нижней сторонами окна допустимы. Но учет ориентации отрезка от $P_5$ к $P_6$ показывает, что такой отрезок не может пересечь правую сторону окна при $t = 1/2$ прежде, чем пересечет его нижнюю сторону при $t = 2/3$, и при этом иметь с окном общие точки. Поэтому отрезок невидим.

Из приведенных примеров следует, что учет ориентации помогает корректно идентифицировать полностью видимые отрезки. Это соображение используется при формальной записи алгоритма Кируса — Бека, которая приводится ниже.

Для формализации этого алгоритма напомним, что параметрическое описание отрезка имеет вид:

$$P(t) = P_1 + (P_2 - P_1)t \qquad 0 \le t \le 1 \tag{3.3}$$

а скалярное произведение внутренней нормали на вектор, начинающийся в произвольной точке отрезка и заканчивающийся в другой точке, лежащей на той же границе окна, т. е.

$$\mathbf{n}_i \cdot [\mathbf{P}(\mathbf{t}) - \mathbf{f}_i] \qquad i = 1, 2, 3, \ldots \tag{3.4}$$

будет положительно, равно нулю или отрицательно — в зависимости от того, будет ли избранная точка отрезка лежать с внутренней стороны границы, на самой этой границе или с ее внешней стороны. Последнее соотношение применимо к любой плоскости или ребру номер $i$, ограничивающим область. Подстановка (3.3) в (3.4) дает условие пересечения отрезка с границей области:

$$\mathbf{n}_i \cdot [\mathbf{P}_1 + (\mathbf{P}_2 - \mathbf{P}_1)\mathbf{t} - \mathbf{f}_i] = 0 \tag{3.5}$$

В другой форме уравнение (3.5) имеет вид:

$$\mathbf{n}_i \cdot [\mathbf{P}_1 - \mathbf{f}_i] + \mathbf{n}_i \cdot [\mathbf{P}_2 - \mathbf{P}_1]\mathbf{t} = 0 \tag{3.6}$$

Заметим, что вектор $\mathbf{P}_2 - \mathbf{P}_1$ определяет ориентацию отрезка, а вектор $\mathbf{P}_1 - \mathbf{f}_i$ пропорционален расстоянию от первого конца отрезка до избранной граничной точки. Обозначим через $\mathbf{D} = \mathbf{P}_2 - \mathbf{P}_1$ директрису или ориентацию отрезка, а через $\mathbf{w}_i = \mathbf{P}_1 - \mathbf{f}_i$ — некий весовой множитель. Тогда уравнение (3.6) можно записать так:

$$\mathrm{t}(\mathbf{n}_i \cdot \mathbf{D}) + \mathbf{w}_i \cdot \mathbf{n}_i = 0 \tag{3.7}$$

Решая последнее уравнение относительно $t$, имеем:

$$t = -\frac{\mathbf{w}_i \cdot \mathbf{n}_i}{\mathbf{D} \cdot \mathbf{n}_i} \qquad \mathbf{D} \neq 0 \qquad i = 1, 2, 3, \ldots \tag{3.8}$$

Здесь $\mathbf{D} \cdot \mathbf{n}_i$ может быть равным нулю только при $\mathbf{D} = 0$[1], а это означает, что $P_2 = P_1$, т. е. вырождение отрезка в точку. Если

$$\mathbf{w}_i \cdot \mathbf{n}_i \begin{cases} < 0, \text{ то эта точка вне окна} \\ = 0, \text{ то она на границе окна} \\ > 0, \text{ то она внутри окна} \end{cases}$$

Уравнение (3.8) используется для получения значений $t$, соответствующих пересечениям отрезка с каждой стороной окна. Если значение $t$ лежит за пределами интервала $0 \leqslant t \leqslant 1$, то можно его проигнорировать. Хотя известно, что отрезок может пересечь выпуклое окно не более чем в двух точках, т. е. при двух значениях $t$, уравнения (3.8) могут дать большее число решений в интервале $0 \leqslant t \leqslant 1$. Эти решения следует разбить на две группы: нижнюю и верхнюю, в зависимости от того, к началу или к концу отрезка будет ближе соответствующая точка. Нужно найти наибольшую из нижних и наименьшую из верхних точек. Если $\mathbf{D} \cdot \mathbf{n}_i > 0$, то найденное значение $t$ рассматривается в качестве возможного нижнего предела. Если $\mathbf{D} \cdot \mathbf{n}_i < 0$, то значение $t$ рассматривается в качестве возможного верхнего предела. На рис. 3.14 дана блок-схема алгоритма, в котором данные условия используются для решения получающейся задачи линейного программирования. Ниже приводится запись этого алгоритма на псевдокоде.

### *Алгоритм двумерного отсечения Кируса — Бека*

$P_1$, $P_2$ — *концевые точки отрезка*
k — *число сторон отсекающего окна*
$\mathbf{n}_i$ — *вектор нормали к i-й стороне окна*
$f_i$ — *точка, лежащая на i-й стороне окна*
D — *директриса отрезка, равная* $P_2 - P_1$
$w_i$ — *весовая функция, равная* $P_1 - f_i$
$t_H$, $t_B$ — *нижний и верхний пределы значений параметра* t

```
инициализировать пределы значений параметра, предполагая что отрезок полностью видимый
t_H = 0
t_B = 1
вычислить директрису D
D = P_2 − P_1
начать главный цикл
for i = 1 to k
```

[1] А также при параллельности D избранной границе. — *Прим. перев.*

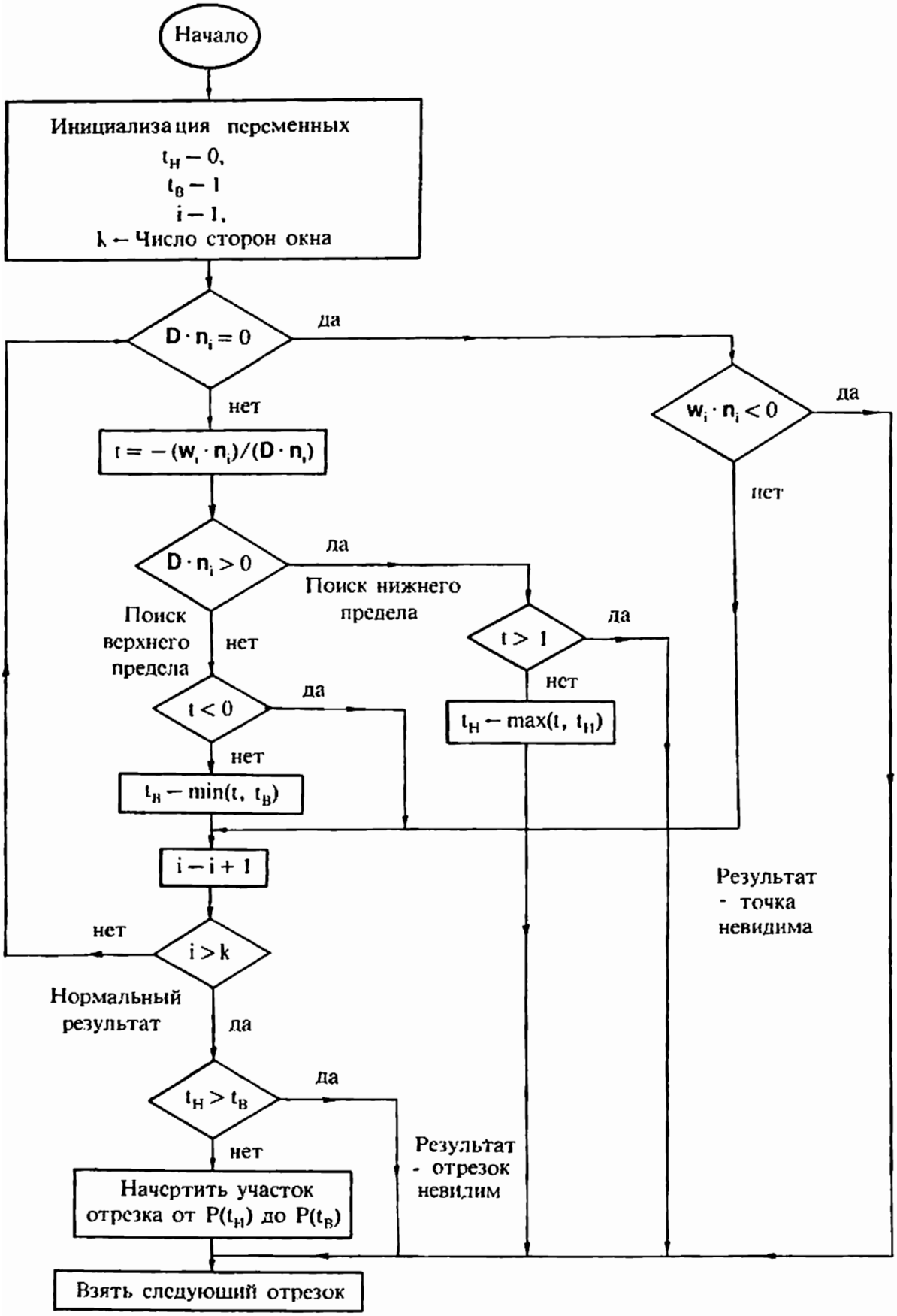



Рис. 3.14. Блок-схема алгоритма отсечения Кируса — Бека.

```
        вычислить w_i, D·n_i и w_i·n для данного i
        w_i = P_1 - f_i
        call Скал-произвед(D, n_i; D_Ск)
        call Скал-произвед(w_i, n_i; W_Ск)
        отрезок вырождается в точку?
        if D_Ск = 0 then 2
        отрезок невырожден, вычислить t
        t = -W_Ск / D_Ск
        поиск верхнего и нижнего пределов t
        if D_Ск > 0 then 1
        поиск верхнего предела
        верно ли, что 0 ⩽ t ⩽ 1?
        if t < 0 then 4
        t_В = min (t, t_В)
        go to 3
        поиск нижнего предела
        верно ли, что t ⩽ 1?
1       if t > 1 then 4
        t_Л = max (t, t_Л)
        go to 3
        отрезок выродился в точку
2       if W_Ск < 0 then 4
        точка видима относительно текущей границы
3   next i
    произошел нормальный выход из цикла
    проверка фактической видимости отрезка
    if t_Н > t_В then 4
    Начертить отрезок от P(t_Н) до P(t_В)
4   Обработать следующий отрезок
```

*подпрограмма вычисления скалярного произведения*

**subroutine** Скал-произвед (Вектор1, Вектор2; Скал)
Вектор1 — *первый вектор, заданный компонентами x и y*
Вектор2 — *второй вектор, заданный компонентами x и y*
Скал — *скалярное произведение векторо*

```
    Скал = Вектор1x * Вектор2x + Вектор1y * Вектор2y
    return
```

Для иллюстрации того, что данный алгоритм не ограничивается окнами прямоугольной формы, рассмотрим следующий пример.

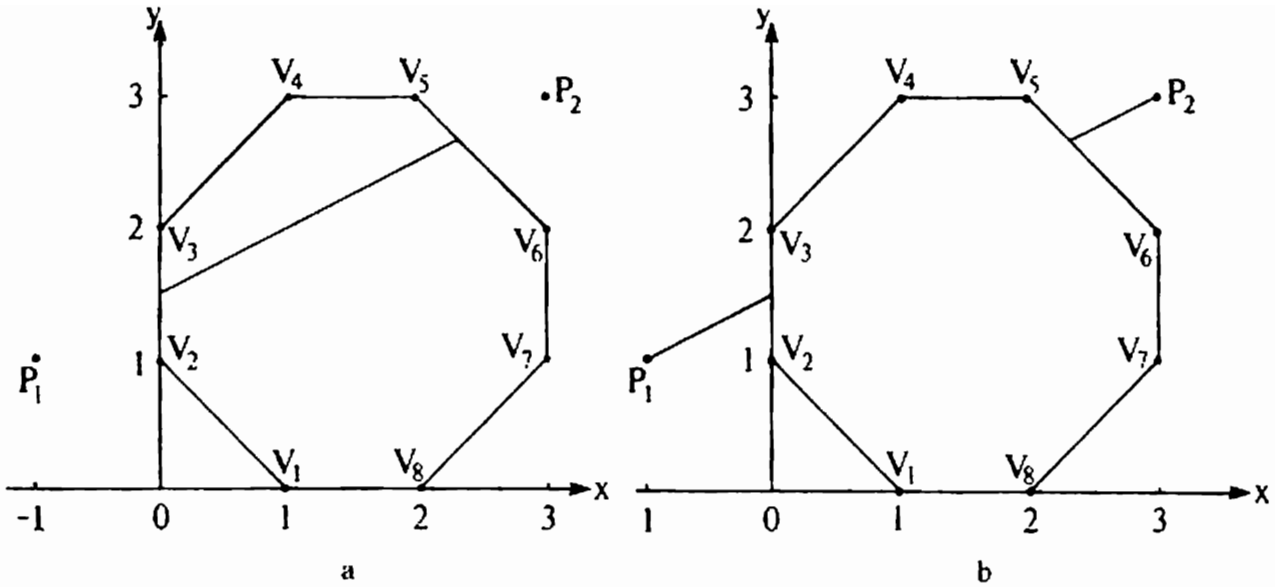



Рис. 3.15. Внутреннее (a) и внешнее (b) отсечение Кируса — Бека многоугольником.

**Пример 3.13.** Алгоритм Кируса — Бека. Нестандартное окно

На рис. 3.15, а показан восьмиугольник, служащий отсекающим окном. Отрезок от $P_1(-1, 1)$ до $P_2(3, 3)$ должен быть отсечен данным окном. В табл. 3.7 приведены результаты работы алгоритма Кируса — Бека. В качестве частного примера рассмотрим ребро от $V_5$ до $V_6$. Алгоритм дает $\mathbf{D} = \mathbf{P}_2 - \mathbf{P}_1 = [3\ 3] - [-1\ 1] = [4\ 2]$. Для граничной точки $f(2, 3)$ имеем $\mathbf{w} = \mathbf{P}_1 - \mathbf{f} = [-1\ 1] - [2\ 3] = [-3\ -2]$. Внутренняя нормаль к ребру $V_5V_6$ равна $\mathbf{n} = [-1\ -1]$. Значит, $\mathbf{D}\cdot\mathbf{n} = -6 < 0$, и найден верхний предел. Поскольку $\mathbf{w}\cdot\mathbf{n} = 5$, то $t_В = -5\ (-6) = 5/6$.

Анализ табл. 3.7 показывает, что максимальное среди значений $t_Н$ равно 1/4, а минимальное среди $t_В$ равно 5/6. Как показано на рис. 3.15, а, отрезок видим в интервале $1/4 \leqslant t \leqslant 5/6$, или от точки (0, 3/2) до точки (7 3, 8 3).

**Таблица 3.7.**

| Ребро | **n** | $f$ | **w** | **w**·**n** | **D**·**n**[1] | $t_Н$ | $t_В$ |
|---|---|---|---|---|---|---|---|
| $V_1V_2$ | [ 1 1] | (1, 0) | [ 2 1] | −1 | 6 | 1/6 | |
| $V_2V_3$ | [ 1 0] | (0, 2) | [−1 −1] | −1 | 4 | 1/4 | |
| $V_3V_4$ | [ 1 −1] | (0, 2) | [−1 −1] | 0 | 2 | 0 | |
| $V_4V_5$ | [ 0 −1] | (2, 3) | [−3 −2] | 2 | −2 | | 1 |
| $V_5V_6$ | [−1 −1] | (2, 3) | [−3 −2] | 5 | −6 | | 5/6 |
| $V_6V_7$ | [−1 0] | (3, 1) | [−4 0] | 4 | −4 | | 1 |
| $V_7V_8$ | [−1 1] | (3, 1) | [−4 0] | 4 | −2 | | 2 |
| $V_8V_1$ | [ 0 1] | (1, 0) | [−2 1] | 1 | 2 | −1/2 | |

[1] При $\mathbf{D}\cdot\mathbf{n} < 0$, верхний предел ($t_В$); при $\mathbf{D}\cdot\mathbf{n} > 0$, нижний ($t_Н$)

## 3.6. ВНУТРЕННЕЕ И ВНЕШНЕЕ ОТСЕЧЕНИЕ

В предыдущих разделах упор был сделан на отсечение отрезка внутренней областью окна. Однако существует возможность отсечения отрезка и внешней его областью. Для этого надо определить часть или части отрезка, лежащие вне окна, и начертить их. Например, видимые части отрезка $P_1P_2$, показанного на рис. 3.15, b, расположены в интервалах $0 \leqslant t < 1/6$ и $5/6 < t \leqslant 1$, или от точки $(-1, 1)$ до точки $(0, 3/2)$, а также от точки $(7/3, 8/3)$ до точки $(3, 3)$. Результаты как внутреннего, так и внешнего отсечений этого отрезка показаны на рис. 3.15.

Внешнее отсечение играет важную роль в дисплеях, допускающих работу с несколькими окнами, как это показано на рис. 3.16. На этом рисунке приоритет окон 1, 2, 3 выше приоритета окна всего экрана, а приоритет окон 1 и 3 выше приоритета окна 2. Поэтому изображение в окне экрана отсекается его собственной внутренней областью и внешними областями окон 1, 2, 3. Изображение в окне 2 отсекается его собственной областью и внешними областями окон 1 и 3. Изображения в окнах 1 и 3 нужно отсечь только соответствующими внутренними областями.

Внешним отсечением можно воспользоваться и при работе с вогнутым полигональным окном. На рис. 3.17 показан вогнутый многоугольник, заданный вершинами $V_1V_2V_3V_4V_5V_6V_1$. Этот вогнутый многоугольник можно преобразовать в выпуклый путем соединения вершин $V_3$ и $V_5$, что и показано на рис. 3.17 штриховым отрезком. Отрезок $P_1P_2$ внутренне отсекается полученным много-

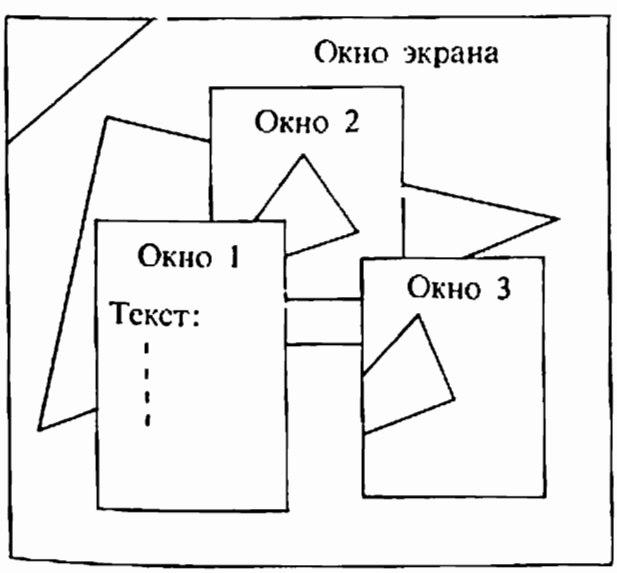



Рис. 3.16. Отсечение несколькими окнами.

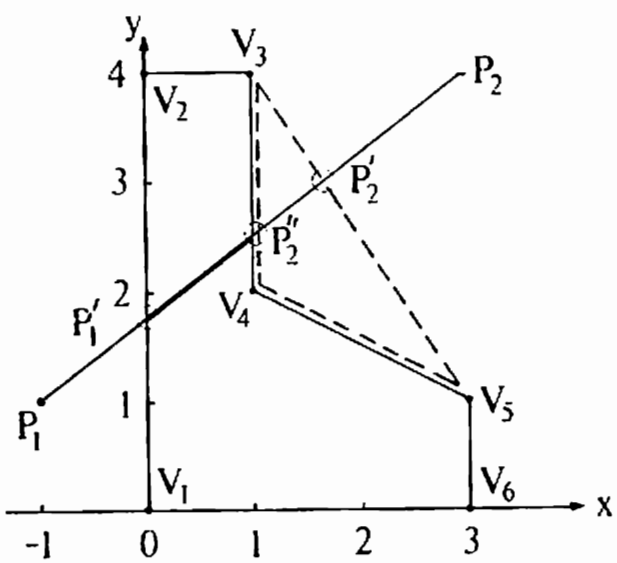



Рис. 3.17. Отсечение невыпуклым окном.

угольником с помощью алгоритма Кируса — Бека. А затем полученный при этом отрезок $P_1'P_2'$ внешне отсекается многоугольником $V_3V_5V_4V_3$, что и дает искомый результат — $P_1'P_2''$.

## 3.7. ОПРЕДЕЛЕНИЕ ФАКТА ВЫПУКЛОСТИ МНОГОУГОЛЬНИКА И ВЫЧИСЛЕНИЕ ЕГО ВНУТРЕННИХ НОРМАЛЕЙ

Для работы с алгоритмом Кируса — Бека надо прежде всего убедиться, что окно является выпуклым, а затем вычислить внутренние нормали к каждой его стороне. Факт выпуклости или невыпуклости двумерного полигонального окна можно установить путем вычисления векторных произведений его смежных сторон. Выводы, которые можно сделать из анализа знаков этих произведений, таковы:

| | |
|---|---|
| Все знаки равны нулю | — многоугольник вырождается в отрезок. |
| Есть как положительные, так и отрицательные знаки | — многоугольник невыпуклый. |
| Все знаки неотрицательные | — многоугольник выпуклый, а внутренние нормали ориентированы влево от его контура. |
| Все знаки неположительны | — многоугольник выпуклый, а внутренние нормали ориентированы вправо от его контура. |

Рис. 3.18 иллюстрирует эти правила.

Другой подход заключается в том, что одна из вершин многоугольника может быть выбрана базой, и могут вычисляться векторные произведения для пар векторов, начинающихся в этой базе и заканчивающихся в последовательных вершинах многоугольника. Результаты этого метода интерпретируются точно так же.

Векторные произведения будут перпендикулярны к плоскости многоугольника. Векторные произведения двух плоских векторов $\mathbf{V}_1$ и $\mathbf{V}_2$ равно $(V_{x_1}V_{y_2} - V_{y_1}V_{x_2})\mathbf{k}$, где $\mathbf{k}$ — единичный вектор, перпендикулярный к плоскости, несущей векторы-сомножители.

Нормаль к стороне многоугольника можно вычислить, если вспомнить, что скалярное произведение пары перпендикулярных

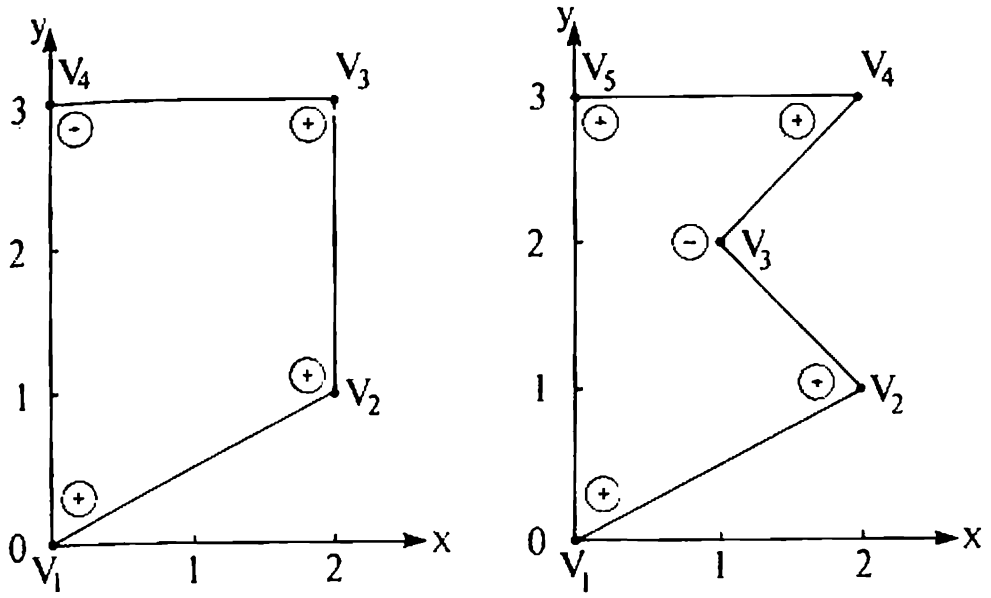



Рис. 3.18. Определение факта выпуклости многоугольника.

векторов равно нулю. Если $n_x$ и $n_y$ — неизвестные компоненты нормали к известному вектору $(V_x, V_y)$ стороны многоугольника, то

$$\mathbf{n} \cdot \mathbf{V}_e = (n_x\mathbf{i} + n_y\mathbf{j}) \cdot (V_{e_x}\mathbf{i} + V_{e_y}\mathbf{j}) = n_x V_{e_x} + n_y V_{e_y} = 0$$

$$n_x V_{e_x} = -n_y V_{e_y}$$

Поскольку нас интересует только направление нормали, то положим, что $n_y = 1$ без потери общности. Следовательно, нормаль равна $\mathbf{n} = -V_{e_y} / V_{e_x}\,\mathbf{i} + \mathbf{j}$.

Если вектор стороны многоугольника образован как разность векторов пары смежных его вершин $V_{i-1}$ и $V_i$ и если скалярное произведение нормали и вектора от $V_{i-1}$ до $V_{i+1}$ положительно, то $\mathbf{n}$ — внутренняя нормаль. В противном случае $\mathbf{n}$ — внешняя нормаль. В последнем случае внутреннюю нормаль можно получить, умножив $\mathbf{n}$ на $-1$. Один простой пример проиллюстрирует этот метод.

**Пример 3.14.** Векторное произведение

На рис. 3.18, a показан простой выпуклый многоугольник, а на рис. 3.18, b — невыпуклый многоугольник. В табл. 3.8 и 3.9 приведены результаты всех вычислений. Рассмотрим, например, векторное произведение сторон, смежных вершине $V_2$, и внутреннюю нормаль к стороне $V_1\,V_2$ для многоугольника с рис. 3.18, a.

Стороны, смежные вершине $V_2$, равны: $\mathbf{V_1\,V_2} = 2\mathbf{i} + \mathbf{j}$, $\mathbf{V_2\,V_3} = 2\,\mathbf{j}$. Векторное их произведение равно $\mathbf{V_1V_2} \otimes \mathbf{V_2\,V_3} = 4\mathbf{k}$, где $\mathbf{k}$ — единичный вектор, перпендикулярный плоскости многоугольника. Это векторное произведение положительно. В табл. 3.8 показано, что векторные произведения положительны для всех вершин

**Таблица 3.8**

| Вершина | Векторы | Векторное произведение |
|---|---|---|
| $V_1$ | $\mathbf{V_4V_1} \otimes \mathbf{V_1V_2}$ | $[\ 0\ \ -3] \otimes [\ 2\ \ \ 1] = +6$ |
| $V_2$ | $\mathbf{V_1V_2} \otimes \mathbf{V_2V_3}$ | $[\ 2\ \ \ 1] \otimes [\ 0\ \ \ 2] = +4$ |
| $V_3$ | $\mathbf{V_2V_3} \otimes \mathbf{V_3V_4}$ | $[\ 0\ \ \ 2] \otimes [-2\ \ \ 0] = +4$ |
| $V_4$ | $\mathbf{V_3V_4} \otimes \mathbf{V_4V_1}$ | $[-2\ \ \ 0] \otimes [\ 0\ \ -3] = +6$ |

**Таблица 3.9**

| Вершина | Векторы | Векторное произведение |
|---|---|---|
| $V_1$ | $\mathbf{V_5V_1} \otimes \mathbf{V_1V_2}$ | $[\ 0\ \ -3] \otimes [\ 2\ \ \ 1] = +6$ |
| $V_2$ | $\mathbf{V_1V_2} \otimes \mathbf{V_2V_3}$ | $[\ 2\ \ \ 1] \otimes [-1\ \ \ 1] = +3$ |
| $V_3$ | $\mathbf{V_2V_3} \otimes \mathbf{V_3V_4}$ | $[-1\ \ \ 1] \otimes [\ 1\ \ \ 1] = -2$ |
| $V_4$ | $\mathbf{V_3V_4} \otimes \mathbf{V_4V_5}$ | $[\ 1\ \ \ 1] \otimes [-2\ \ \ 0] = +2$ |
| $V_5$ | $\mathbf{V_4V_5} \otimes \mathbf{V_5V_1}$ | $[-2\ \ \ 0] \otimes [\ 0\ \ -3] = +6$ |

многоугольника. Поэтому данный многоугольник выпуклый. В табл. 3.9 показано, что для многоугольника с рис. 3.18, b векторное произведение сторон, смежных вершине $\mathbf{V_3}$, отрицательно, в то время как для всех других вершин оно положительно. Следовательно, этот многоугольник невыпуклый.

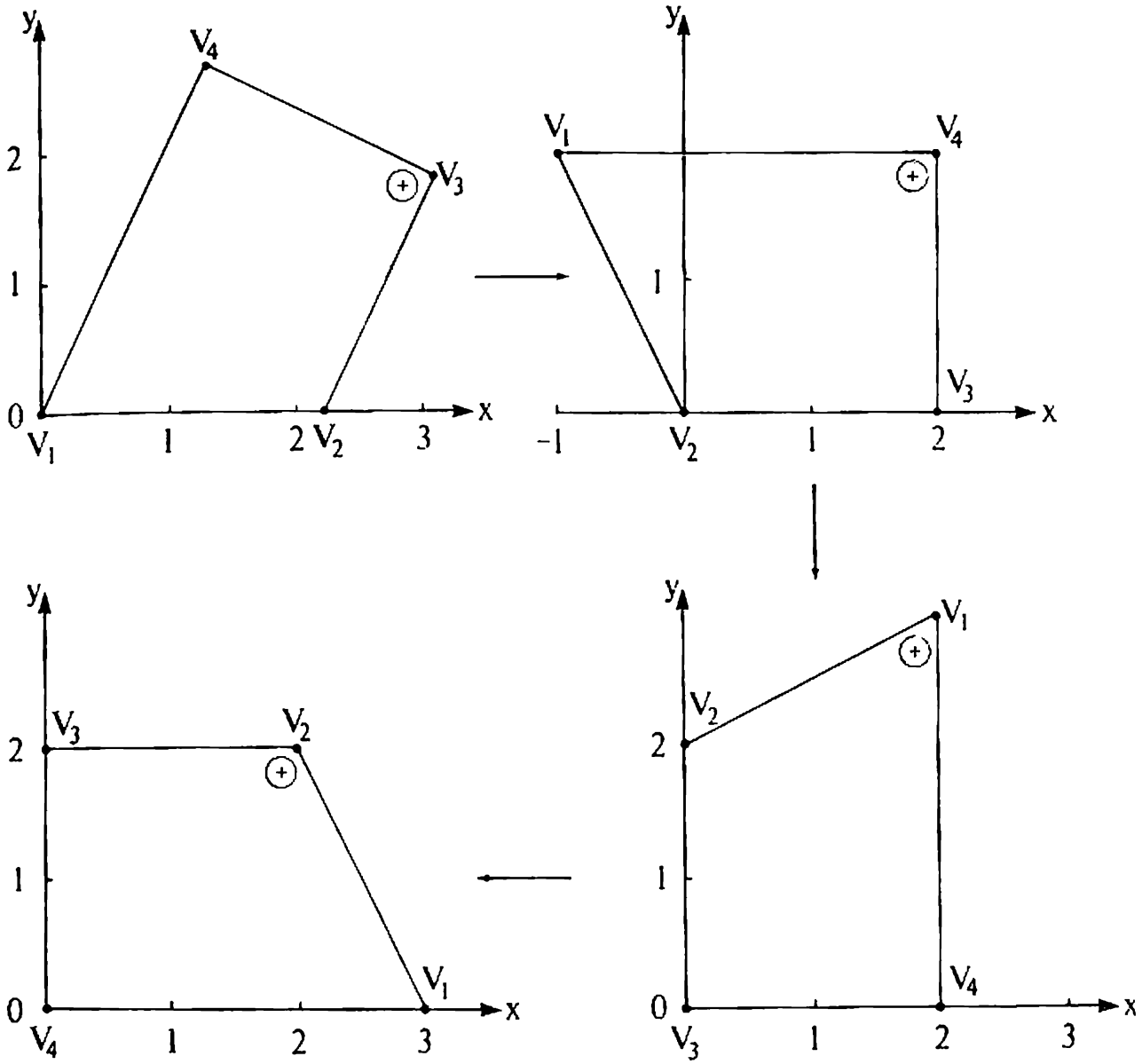
Нормаль к вектору стороны $\mathbf{V_1V_2}$ равна $\mathbf{n} = -1/2\mathbf{i} + \mathbf{j}$, или, иначе, $\mathbf{n} = -\mathbf{i} + 2\mathbf{j}$. Вектор $\mathbf{V_1V_3}$ равен $2\mathbf{i} + 3\mathbf{j}$. Значит, $\mathbf{n} \cdot \mathbf{V_1V_3} = (-\mathbf{i} + 2\mathbf{j}) \cdot (2\mathbf{i} + 3\mathbf{j}) = 4 > 0$, поэтому получена внутренняя нормаль.

Другой метод определения факта выпуклости многоугольного окна и вычисления внутренних нормалей к его сторонам состоит в реализации поворотов и переносов этого окна. Алгоритм этого метода таков:

Для каждой $i$-й вершины полигонального окна надо перенести его так, чтобы эта вершина совпала с началом координат.

Повернуть многоугольник относительно начала координат так, чтобы $(i + 1)$-я его вершина оказалась на положительной полуоси $x$.

Вычислить знак ординаты $y$ $(i + 2)$-й вершины.




а. Выпуклый

Рис. 3.19. Использование поворота и переноса для определения выпуклости окна.

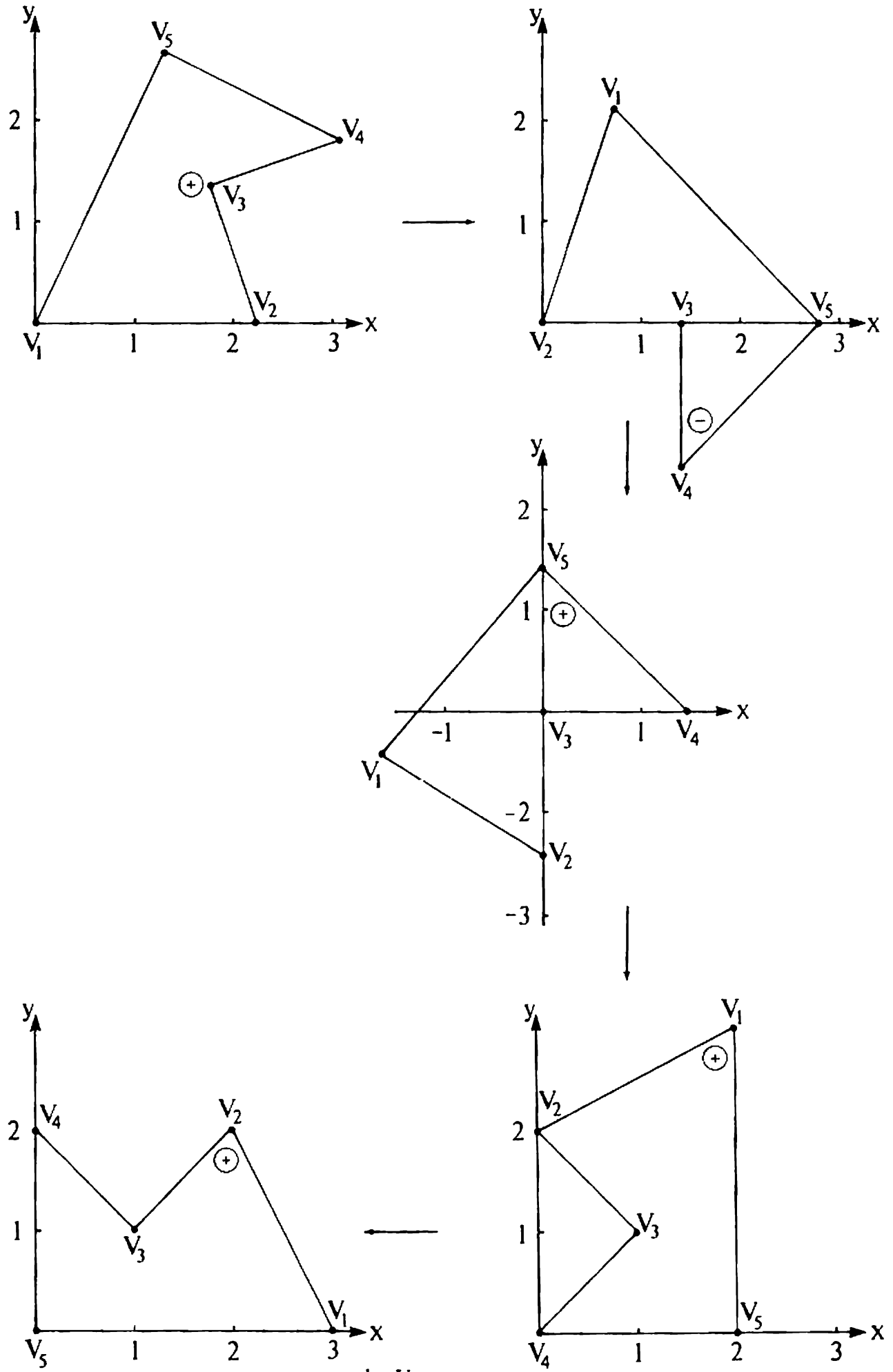
Если знаки ординат $y$ всех $(i + 2)$-х вершин совпадают, то окно выпукло; иначе, оно невыпукло.

Если для какой-нибудь из $(i + 2)$-х вершин ордината равна нулю, то $i$-я, $(i + 1)$-я и $(i + 2)$-я вершины коллинеарны.

Если все ординаты $(i + 2)$-х вершин равны нулю, то окно вырождено, т. е. является отрезком.

В повернутой системе координат внутренняя нормаль к $(i + 1)$-й стороне выпуклого многоугольника имеет нулевую абсциссу и ординату, равную знаку ординаты $(i + 2)$-й вершины.

Для определения исходной ориентации внутренней нормали используются только обратные повороты.




b. Невыпуклый

Рис.3.19. Продолжение.

На рис. 3.19 показаны разные стадии работы этого алгоритма для выпуклого и невыпуклого многоугольников с рис. 3.18. Алгоритмы, реализующие поворот и перенос, приведены в работе [1-1].

## 3.8. РАЗБИЕНИЕ НЕВЫПУКЛЫХ МНОГОУГОЛЬНИКОВ

Во многих алгоритмах требуется, чтобы полигональные отсекающие области были выпуклыми. Одним из примеров таких алгоритмов является вышеописанный алгоритм Кируса—Бека. В последующих разделах будут представлены и другие примеры. Простое обобщение метода поворотов и переносов окна, используемого для определения факта его выпуклости или невыпуклости, позволяет разбивать или разделять простой невыпуклый многоугольник на несколько выпуклых многоугольников. Эту процедуру можно встроить в изложенный выше алгоритм. Если вершины многоугольника перечисляются против часовой стрелки, то процедура будет иметь такой вид:

Для каждой $i$-й вершины многоугольника надо так его перенести, чтобы она совпадала с началом координат.

Повернуть многоугольник относительно координат по часовой стрелке так, чтобы $(i + 1)$-я его вершина оказалась на положительной полуоси $x$.

Проанализировать знак ординаты $(i + 2)$-й вершины. Если он неотрицателен, то многоугольник выпуклый в $(i + 1)$-й вершине. Если же этот знак отрицателен, то многоугольник невыпуклый; разбить его.

Многоугольник разрезается вдоль положительной полуоси $x$, т. е. ищутся все такие его стороны, которые пересекаются с осью $x$. Образуются два новых многоугольника: один состоит из вершин, лежащих выше оси $x$ и ближайшей к началу координат точки пересечения с $x > x_{i+1}$, а второй — из вершин, лежащих ниже оси $x$ и уже упомянутой точки пересечения[1].

---

[1] Это построение не вполне корректно. Правильное построение таково: один многоугольник образован вершинами исходной фигуры, начиная с $(i + 1)$-й и кончая упомянутой точкой пересечения (он целиком ниже оси $x$), а другой образован этой точкой пересечения и всеми вершинами исходной фигуры, не вошедшими в состав первого многоугольника (он может пересекать ось $x$). — *Прим. перев.*

Алгоритм рекурсивно применяется к полученным многоугольникам до тех пор, пока все они не станут выпуклыми.

Этот алгоритм не дает оптимального разбиения в смысле минимального числа выпуклых компонент. Кроме того, алгоритм не сможет корректно разбить многоугольник, стороны которого пересекаются между собой.

Проиллюстрируем работу этого алгоритма на примере.

**Пример 3.15.** Разбиение невыпуклого многоугольника

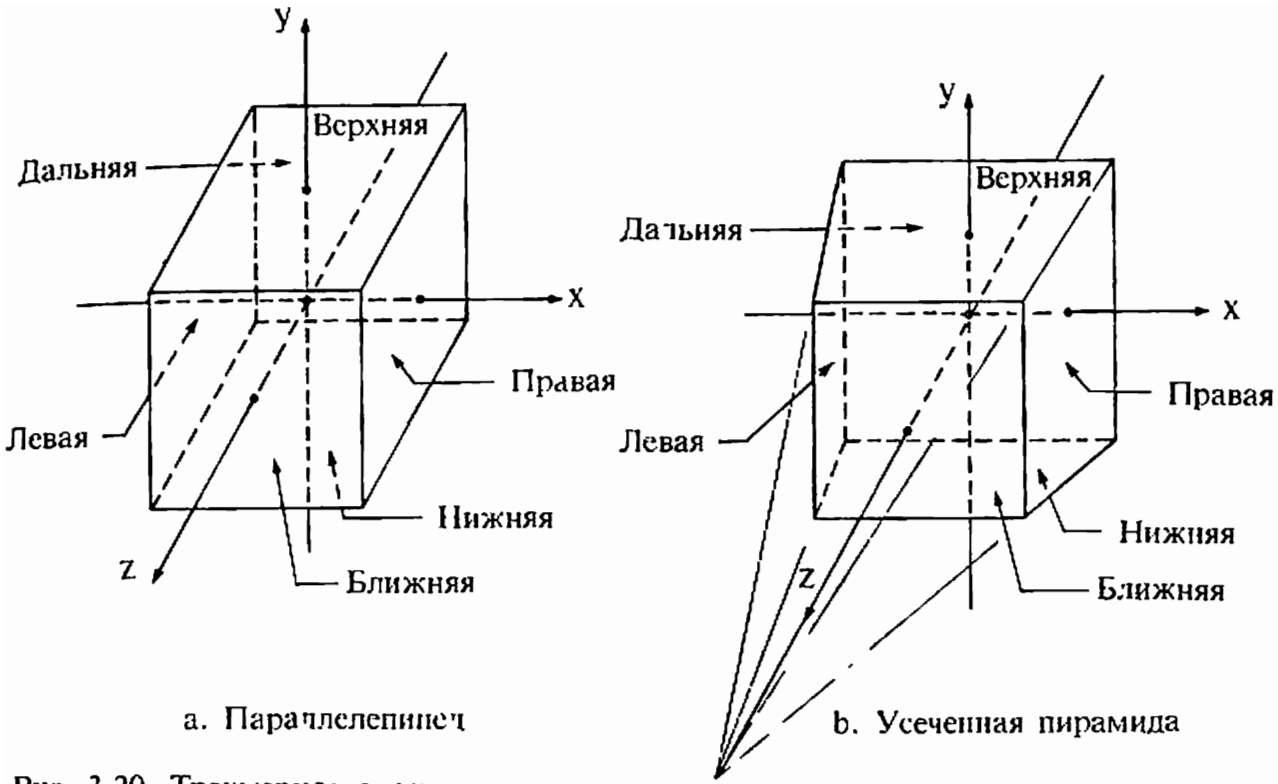
Рассмотрим невыпуклый многоугольник, показанный на рис. 3.19, b. Когда его вершина $V_2$ совпадает с началом координат, а $V_3$ лежит на положительной полуоси $x$, знак ординаты $V_4$ будет отрицательным. Значит, этот многоугольник невыпуклый. Разрезание его осью $x$ дает многоугольники $V_3\ V_4\ V_5$ ниже оси $x$ и $V_1\ V_2\ V_3\ V_5$ выше этой оси. Возобновление работы алгоритма с новыми многоугольниками показывает, что оба они выпуклы. Поэтому алгоритм прекратит дальнейшую работу.

## 3.9. ТРЕХМЕРНОЕ ОТСЕЧЕНИЕ

Прежде чем заняться обобщением изложенных методов для случая трех измерений, необходимо обсудить вопрос о форме отсекающего объема. Двумя наиболее распространенными формами трехмерных отсекателей являются: прямоугольный параллелепипед, т. е. полый брусок, используемый при параллельном или аксонометрическом проецировании, а также усеченная пирамида, часто называемая пирамидой видимости, которая используется при центральном проецировании. Эти формы показаны на рис. 3.20, у каждой из них шесть граней: левая, правая, верхняя, нижняя, ближняя и дальняя. Существует, кроме того, необходимость отсекать и по нестандартным объемам.

Как и при двумерном отсечении, отрезки, которые полностью видимы или тривиально невидимы, можно идентифицировать с использованием обобщения кодов концевых точек Коэна-Сазерленда. В трехмерном случае используется 6-битовый код. Вновь самый правый бит кода считается первым. В биты кода заносятся единицы с помощью обобщения двумерной процедуры. Конкретно единица заносится:

| | |
|---|---|
| в первый бит | — если конец ребра левее объема, |
| во второй бит | — если конец ребра правее объема, |
| в третий бит | — если конец ребра ниже объема, |
| в четвертый бит | — если конец ребра выше объема, |
| в пятый бит | — если конец ребра ближе объема, |
| в шестой бит | — если конец ребра дальше объема. |




a. Параллелепипед

b. Усеченная пирамида

Рис. 3.20. Трехмерное отсечение.

В противном случае в соответствующие биты заносятся нули. И опять, если коды обоих концов отрезка *равны нулю*, то оба конца видимы и отрезок тоже будет полностью видимым. Точно так же, если побитовое логическое произведение кодов концов отрезка *не равно нулю*, то он полностью невидим. Если же это логическое произведение равно нулю, то отрезок может оказаться как частично видимым, так и полностью невидимым. В этом случае необходимо определять пересечения отрезка с гранями отсекающего объема.

Поиск кодов точки относительно отсекающего прямоугольного параллелепипеда является прямым обобщением соответствующего двумерного алгоритма. Однако случай, когда отсекателем служит усеченная пирамида, показанная на рис. 3.20, b, заслуживает дополнительного обсуждения. Один из методов [1-3] заключается в преобразовании отсекателя в каноническую форму, где $x_{прав} = 1$, $x_{лев} = -1$, $y_{верх} = 1$, $y_{низ} = -1$ при $z_{даль} = 1$. Если $z_{ближ} = a$, где $0 < a \leqslant 1$, а центр проекции совпадает с началом левой системы координат, то проверка кодов концевых точек заметно упрощается.

В более естественном методе, меньше искажающем форму отсекателя, отрезок, соединяющий центр проекции с центром усеченной пирамиды, совмещается с осью $z$ правой системы координат, как это показано на рис. 3.20, b.

Вид усеченной пирамиды сверху показан на рис. 3.21, a. Уравнение прямой на плоскости $xz$, несущей проекцию правой грани отсе-

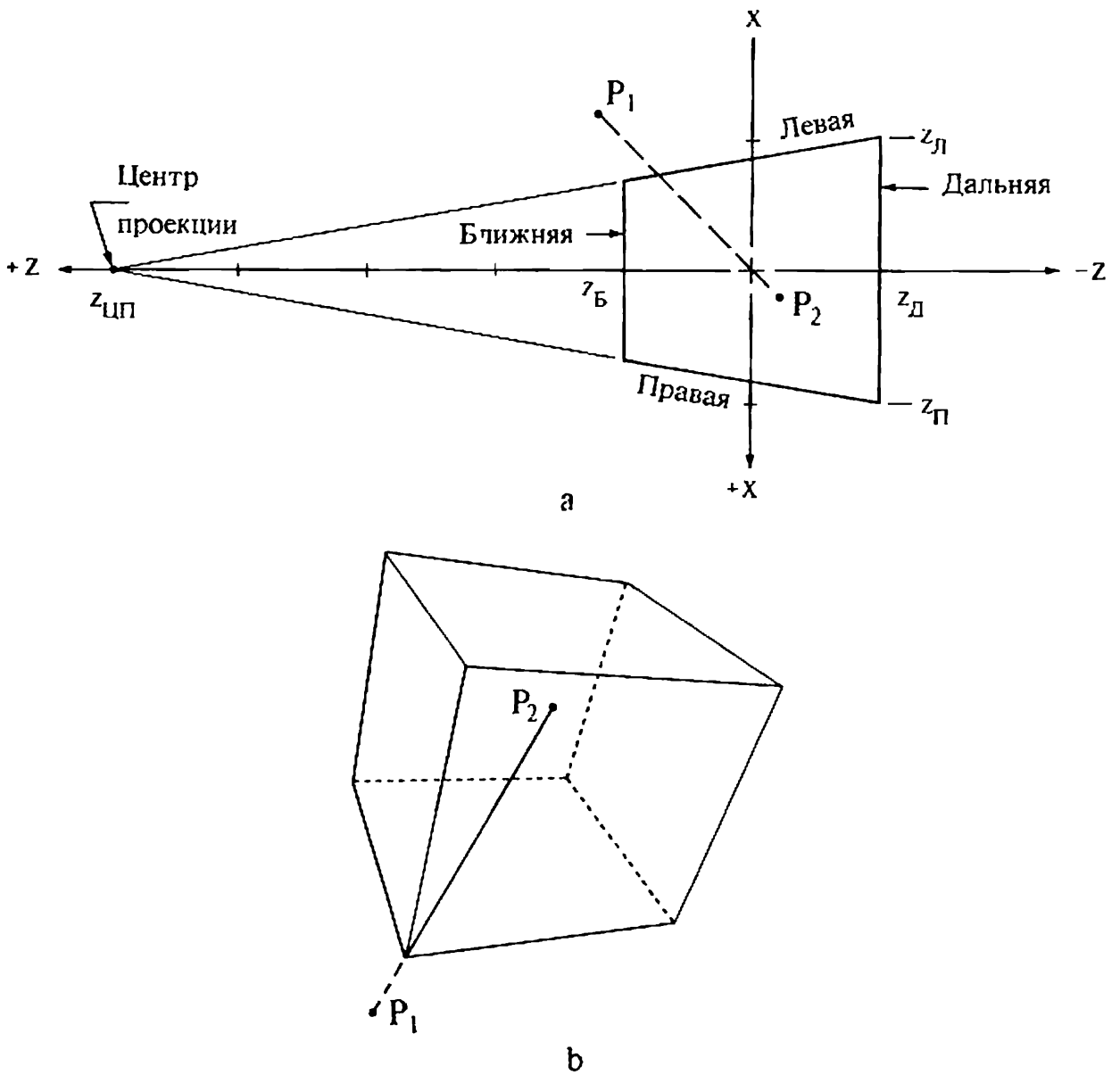



Рис. 3.21. Усеченная пирамида.

кателя, имеет вид:

$$x = (z - z_{\text{ЦП}}) \cdot x_{\text{П}} / (z_{\text{Д}} - z_{\text{ЦП}}) = z\alpha_1 + \alpha_2,$$

где $\alpha_1 = x_{\text{П}} / (z_{\text{Д}} - z_{\text{ЦП}})$ и $\alpha_2 = -\alpha_1 z_{\text{ЦП}}$.

Уравнение этой прямой можно использовать для определения местоположения точки: справа, на или слева от прямой, т. е. вне отсекателя, на плоскости, несущей его правую грань, или внутри отсекателя. Подстановка координат $x$ и $z$ точки $P$ в пробную функцию правой грани дает следующий результат:

$$f_{\text{П}} = x - z\alpha_1 - \alpha_2 \begin{cases} > 0, \text{ если } P \text{ справа от плоскости} \\ = 0, \text{ если } P \text{ на плоскости} \\ < 0, \text{ если } P \text{ слева от плоскости} \end{cases}$$

Пробные функции для левой, верхней и нижней граней имеют вид:

$$f_{\text{Л}} = z\beta_1 - \beta_2 \qquad \begin{cases} > 0, \text{ если } P \text{ справа от плоскости} \\ = 0, \text{ если } P \text{ на плоскости} \\ < 0, \text{ если } P \text{ слева от плоскости} \end{cases}$$

где $\beta_1 = x_{\text{Л}}/(z_{\text{Д}} - z_{\text{ЦП}})$, а $\beta_2 = -\beta_1 z_{\text{ЦП}}$.

$$f_{\text{В}} = y - z\gamma_1 - \gamma_2 \qquad \begin{cases} > 0, \text{ если } P \text{ выше плоскости} \\ = 0, \text{ если } P \text{ на плоскости} \\ < 0, \text{ если } P \text{ ниже плоскости} \end{cases}$$

где $\gamma_1 = y_{\text{В}}/(z_{\text{Д}} - z_{\text{ЦП}})$, а $\gamma_2 = -\gamma_1 z_{\text{ЦП}}$.

$$f_{\text{Н}} = y - z\delta_1 - \delta_2 \qquad \begin{cases} < 0, \text{ если } P \text{ ниже плоскости} \\ = 0, \text{ если } P \text{ на плоскости} \\ > 0, \text{ если } P \text{ выше плоскости} \end{cases}$$

где $\delta_1 = y_{\text{Н}}/(z_{\text{Д}} - z_{\text{ЦП}})$, а $\delta_2 = -\delta_1 z_{\text{ЦП}}$.

Наконец, пробные функции для ближней и дальней граней имеют вид:

$$f_{\text{Б}} = z - z_{\text{Б}} \qquad \begin{cases} > 0, \text{ если } P \text{ ближе плоскости} \\ = 0, \text{ если } P \text{ на плоскости} \\ < 0, \text{ если } P \text{ дальше плоскости} \end{cases}$$

$$f_{\text{Д}} = z - z_{\text{Д}} \qquad \begin{cases} < 0, \text{ если } P \text{ дальше плоскости} \\ = 0, \text{ если } P \text{ на плоскости} \\ > 0, \text{ если } P \text{ ближе плоскости} \end{cases}$$

Чем ближе $z_{\text{ЦП}}$ к бесконечности, тем больше форма отсекателя приближается к прямоугольному параллелепипеду. Пробные функции при этом тоже приближаются к соответствующим пробным функциям прямоугольного параллелепипеда.

Как указывали Лианг и Барский [3-5], последний метод может дать некорректные значения кодов, если концы отрезка лежат за центром проекции. Это происходит потому, что плоскости, несущие левую, правую, верхнюю и нижнюю грани усеченной пирамиды, пересекаются в точке центра проекции. Поэтому существуют точки, расположенные одновременно левее левой и правее правой граней. Лианг и Барский предложили способ устранения этой неопределенности. Для этого в принципе необходимо лишь обратить значения первых четырех битов кода при $z > z_{\text{ЦП}}$. См. также разд. 3.12.

## 3.10.ТРЕХМЕРНЫЙ АЛГОРИТМ РАЗБИЕНИЯ СРЕДНЕЙ ТОЧКОЙ

Двумерный алгоритм разбиения средней точкой, описанный выше (см. разд. 3.3), непосредственно обобщается на случай трех измерений. В записи этого алгоритма на псевдокоде нужно изменить размерности у массивов Ткод и Окно, а подпрограммы Конец и Логическое переписать с учетом трех измерений. Запись на псевдокоде подпрограммы вычисления кода концевой точки отрезка имеет вид:

*подпрограмма вычисления кода концевой точки отрезка относительно трехмерной усеченной пирамиды*

**subroutine** Конец(P, Окно; Ткод, Сумма)

$P_x$, $P_y$, $P_z$ — *координаты x, y и z точки* P

Окно — *массив* 1 × 7, *содержащий координаты* ($x_Л$, $x_П$, $y_Н$, $y_В$, $z_Б$, $z_Д$, $z_{ЦП}$) *левой, правой, нижней, верхней, ближней, задней сторон окна и центра проекции*

Ткод — *масссив* 1 × 6, *содержащий код концевой точки*

Сумма — *сумма элементов* Ткод

*вычисление* $\alpha_1$, $\alpha_2$, $\beta_1$, $\beta_2$, $\gamma_1$, $\gamma_2$, $\delta_1$, $\delta_2$

$\alpha_1 = x_П/(z_Д - z_{ЦП})$

$\alpha_2 = -\alpha_1 z_{ЦП}$

$\beta_1 = x_Л/(z_Д - z_{ЦП})$

$\beta_2 = -\beta_1 z_{ЦП}$

$\gamma_1 = y_В/(z_Д - z_{ЦП})$

$\gamma_2 = -\gamma_1 z_{ЦП}$

$\delta_1 = y_Н/(z_Д - z_{ЦП})$

$\delta_2 = -\delta_1 z_{ЦП}$

*определение кода концевой точки*

**if** $P_x - P_z\beta_1 - \beta_2 < 0$ **then** Ткод(6) = 1 **else** Ткод(6) = 0

**if** $P_x - P_z\alpha_1 - \alpha_2 > 0$ **then** Ткод(5) = 1 **else** Ткод(5) = 0

**if** $P_y - P_z\delta_1 - \delta_2 < 0$ **then** Ткод(4) = 1 **else** Ткод(4) = 0

**if** $P_y - P_z\gamma_1 - \gamma_2 > 0$ **then** Ткод(3) = 1 **else** Ткод(3) = 0

**if** $P_z - z_Б > 0$ **then** Ткод(2) = 1 **else** Ткод(2) = 0

**if** $P_z - z_Д < 0$ **then** Ткод(1) = 1 **else** Ткод(1) = 0

*вычисление суммы*

```
Сумма = 0
for i = 1 to 6
  Сумма = Сумма + Ткод(i)
next i
return
```

Ниже следует пример использования трехмерного алгоритма отсечения методом разбиения средней точкой.

**Пример 3.16.** Трехмерное отсечение методом разбиения средней точкой

Возьмем отрезок от $P_1(-600, -600, 600)$ до $P_2(100, 100, -100)$, заданный в экранной системе координат, расположенной на дальней плоскости, и отсекаемый пирамидой видимости с $x_П = y_В = 500$, $x_Л = y_Н = -500$. Аппликаты ближней и дальней граней таковы: $z_Б = 357.14$, $z_Д = -500$. Центр проекции лежит на $z_{ЦП} = 2500$. Вид сверху на эту сцену показан на рис. 3.21, a, а ее перспективное изображение — на рис. 3.21, b. Пробные функции отсекателя таковы:

правая: $f_П = 6x + z - 2500$
левая: $f_Л = 6x - z + 2500$
верхняя: $f_В = 6y - z - 2500$
нижняя: $f_Н = 6y - z + 2500$
ближняя: $f_Б = z - 357.14$
дальняя: $f_Д = z + 500$

Код концевой точки $P_1$ равен (010101), а код $P_2$ равен (000000). Поскольку оба этих кода не равны нулю одновременно, отрезок не является полностью видимым. Логическое произведение кодов концевых точек равно (000000). Поэтому отрезок не является и тривиально невидимым. Поскольку код точки $P_2$ равен нулю, то $P_2$ лежит внутри отсекателя. Следовательно, $P_2$ — наиболее удаленная от $P_1$ видимая точка отрезка. Значит, отрезок имеет только одно пересечение с границей отсекателя. Координаты середины отрезка равны:

$$x_c = (x_2 + x_1)/2 = [100 + (-600)]/2 = -250$$
$$y_c = (y_2 + y_1)/2 = [100 + (-600)]/2 = -250$$
$$z_c = (z_2 + z_1)/2 = (-100 + 600)/2 = 250$$

**Таблица 3.10.**

| $P_1$ | $P_2$ | $P_C$ | Примечания |
|---|---|---|---|
| −600, −600, 600 | 100, 100, −100 | −250, −250, 250 | Продолжить с $P_1$ $P_c$ |
| −600, −600, 600 | −250, −250, 250 | −425, −425, 425 | Продолжить с $P_c$ $P_2$ |
| −425, −425, 425 | −250, 250, 250 | −338, −338, 337 | Продолжить с $P_1$ $P_c$ |
| −425, −425, 425 | −338, 338, 337 | −382, −382, 381 | Продолжить с $P_c$ $P_2$ |
| −382, −382, 381 | −338, 338, 337 | −360, −360, 359 | Продолжить с $P_c$ $P_2$ |
| −360, −360, 359 | −338, −338, 337 | −349, −349, 348 | Продолжить с $P_1$ $P_c$ |
| −360, −360, 359 | −349, −349, 348 | −355, −355, 353 | Продолжить с $P_1$ $P_c$ |
| −360, −360, 359 | −355, −355, 353 | −358, −358, 356 | Продолжить с $P_c$ $P_2$ |
| −358, −358, 356 | −355, −355, 353 | −357, −357, 354 | Продолжить с $P_1$ $P_c$ |
| −358, −358, 356 | −357, −357, 354 | −358, −358, 355 | Продолжить с $P_c$ $P_2$ |
| −358, −358, 355 | −357, −357, 354 | −358, −358, 354 | Найдено пересечение |

Код этой точки равен (000000). Отрезок $P_c P_2$ — полностью видимый, отрезок $P_1 P_c$ — частично видимый. Алгоритм продолжает работу с отрезком $P_1 P_c$. Результаты процесса подразбиения даны в табл. 3.10. Фактические координаты точки пересечения равны (−357.14, −357.14, 357.14). Отличие этих значений от приведенных в табл. 3.10 объясняется тем, что в алгоритме использовалась целочисленная арифметика.

## 3.11. ТРЕХМЕРНЫЙ АЛГОРИТМ КИРУСА — БЕКА

В двумерном варианте алгоритма Кируса — Бека [3-4] на форму отсекателя не накладывалось никаких ограничений, за исключением выпуклости. Поэтому и в трехмерном варианте отсекатель может быть произвольным выпуклым объемом. Можно непосредственно воспользоваться ранее разработанной двумерной версией. Теперь $k$ обозначает не число сторон многоугольника, а число граней многогранника (см. рис. 3.14). Все векторы теперь имеют по три компоненты: $x$, $y$, $z$. Обобщение подпрограммы Скал — произвед на случай трехмерных векторов реализуется также непосредственно. Для более полной иллюстрации этого алгоритма рассмотрим следующие примеры. В первом примере отсекателем служит прямоугольный параллелепипед, т. е. полый брусок.

**Пример 3.17.** Трехмерный алгоритм Кируса — Бека

На рис. 3.22 показан отрезок от $P_1(-2, -1, 1/2)$ до $P_2(3/2, 3/2, -1/2)$, который нужно отсечь по объему с координатами $(x_Л, y_Н, y_В, y_В, z_Б, z_Д) = (-1, 1, -1, 1, 1, -1)$. Шесть внутренних нормалей к граням отсекателя, очевидно, равны:

верхняя : $\mathbf{n}_В = -\mathbf{j} = [0\ -1\ 0]$
нижняя : $\mathbf{n}_Н = \mathbf{j} = [0\ 1\ 0]$
правая : $\mathbf{n}_П = -\mathbf{i} = [-1\ 0\ 0]$
левая : $\mathbf{n}_Л = \mathbf{i} = [1\ 0\ 0]$
ближняя: $\mathbf{n}_Б = -\mathbf{k} = [0\ 0\ -1]$
дальняя : $\mathbf{n}_Д = \mathbf{k} = [0\ 0\ 1]$

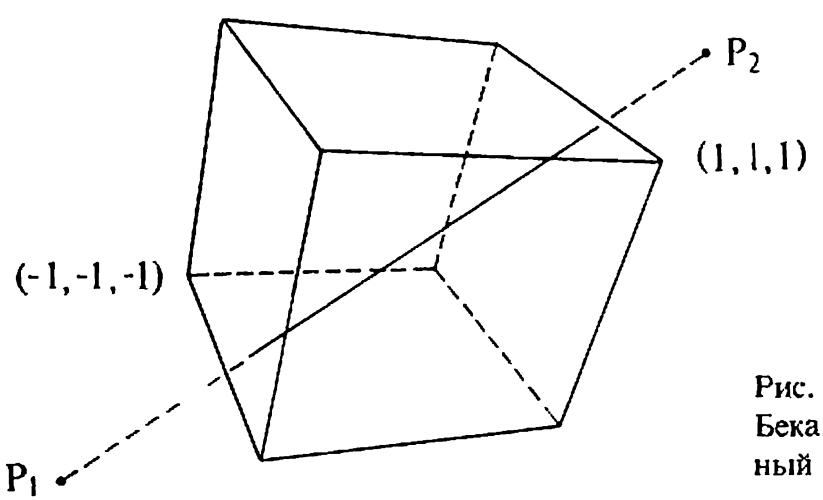



Рис. 3.22. Отсечение Кируса — Бека: трехмерный прямоугольный объем.

Таблица 3.11.

| Грань | n | f | w | w·n | D·n[1] | $t_H$ | $t_B$ |
|---|---|---|---|---|---|---|---|
| Верхняя | [ 0 −1 0] | ( 1, 1, 1) | [−3 −2 −1/2] | 2 | −5/2 | | 4/5 |
| Нижняя | [ 0 1 0] | (−1, −1, −1) | [−1 0 3/2] | 0 | 5/2 | 0 | |
| Правая | [−1 0 0] | ( 1, 1, 1) | [−3 −2 −1/2] | 3 | −7/2 | | 6/7 |
| Левая | [ 1 0 0] | (−1, −1, −1) | [−1 0 3/2] | −1 | 7/2 | 2/7 | |
| Ближняя | [ 0 0 −1] | ( 1, 1, 1) | [−3 −2 −1/2] | 1/2 | 1 | −1/2 | |
| Дальняя | [ 0 0 1] | (−1, −1, −1) | [−1 0 3/2] | 3/2 | −1 | | 3/2 |

[1] При $\mathbf{D}\cdot\mathbf{n} < 0$, верхний предел ($t_B$); при $\mathbf{D}\cdot\mathbf{n} > 0$, нижний предел ($t_H$).

Выбор точек, лежащих на каждой грани отсекателя, также очевиден. Достаточно взять две точки, лежащие на концах одной из главных диагоналей параллелепипеда. Итак,

$$f_B = f_П = f_Б(1, 1, 1), \text{ а } f_H = f_Л = f_Д(-1, -1, -1)$$

Вместо этих точек можно взять центры или любые угловые точки на соответствующих гранях.

Директриса отрезка $P_1P_2$ равна:

$$\mathbf{D} = \mathbf{P}_2 - \mathbf{P}_1 = [3/2\ 3/2\ -1/2] - [-2\ -1\ 1/2] = [7/2\ 5/2\ -1]$$

Для граничной точки $f_Л(-1, -1, -1)$:

$$\mathbf{w} = \mathbf{P}_1 - \mathbf{f} = [-2\ -1\ 1/2] - [-1\ -1\ -1] = [-1\ 0\ 3/2]$$

а внутренняя нормаль к левой грани отсекателя равна

$$\mathbf{n}_Л = [1\ 0\ 0].$$

Следовательно,

$$\mathbf{D}\cdot\mathbf{n}_Л = [7/2\ 5/2\ -1]\cdot[1\ 0\ 0] = 7/2 > 0$$

что соответствует нижнему пределу, а

$$\mathbf{w}\cdot\mathbf{n}_Л = [-1\ 0\ 3/2]\cdot[1\ 0\ 0] = -1$$

и

$$t_Л = -(-1)/(7/2) = 2/7$$

Полностью результаты работы алгоритма собраны в табл. 3.11. Анализ таблицы 3.11 показывает, что максимальным из нижних значений будет $t_H = 2/7$, а минимальным из верхних значений будет $t_B = 4/5$. Параметрическое представление отрезка $P_1P_2$ имеет вид:

$$P(t) = [-2\ -1\ 1/2] + [7/2\ 5/2\ -1]t$$

Подстановка сюда $t_H$ и $t_B$ дает:

$$P(2/7) = [-2\ -1\ 1/2] + [7/2\ 5/2\ -1]\cdot(2/7) = [-1\ -2/7\ 3/14]$$

для точки пересечения отрезка с левой гранью отсекателя и

$$P(4/5) = [-2\ -1\ 1\ 2] + [7/2\ 5/2\ -1]\cdot(4/5) = [4/5\ 1\ -3/10]$$

для точки его пересечения с верхней гранью отсекателя.

Отсечение относительно стандартной пирамиды видимости будет ненамного сложнее. Здесь внутренние нормали к граням отсекателя следует определить формально, так как их значения неочевидны.

**Пример 3.18.** Отсечение относительно пирамиды видимости

Возьмем тот же отрезок, что и в примере 3.17, от $P_1(-2, -1, 1/2)$ до $P_2(3/2, 3/2, -1/2)$, который отсекается пирамидой видимости с координатами $(x_Л, x_П, y_Н, y_В, z_Б, z_Д) = (-1, 1, -1, 1, 1, -1)$, а центр проекции расположен на $z_{ЦП} = 5$. См. рис. 3.20, b.

Значения внутренних нормалей к ближней и дальней граням очевидны. Для остальных четырех граней их можно получить, вычислив векторные произведения, образованные парами векторов, которые начинаются в центре проекции, а заканчиваются в углах соответствующих граней при $z = 0$. Эти векторы равны:

$$\mathbf{V}_1 = [1\ 1\ -5]$$
$$\mathbf{V}_2 = [-1\ 1\ -5]$$
$$\mathbf{V}_3 = [-1\ -1\ -5]$$
$$\mathbf{V}_4 = [1\ -1\ -5]$$

Внутренние нормали равны:

$$\mathbf{n}_В = \mathbf{V}_1 \otimes \mathbf{V}_2 = [0\ -10\ -2]$$
$$\mathbf{n}_Л = \mathbf{V}_2 \otimes \mathbf{V}_3 = [10\ 0\ -2]$$
$$\mathbf{n}_Н = \mathbf{V}_3 \otimes \mathbf{V}_4 = [0\ 10\ -2]$$
$$\mathbf{n}_П = \mathbf{V}_4 \otimes \mathbf{V}_1 = [-10\ 0\ -2]$$
$$\mathbf{n}_Б = [0\ 0\ -1]$$
$$\mathbf{n}_Д = [0\ 0\ 1]$$

Поскольку центр проекции принадлежит четырем из шести плоскостей, несущих грани отсекателя, то удобно выбрать граничные точки так:

$$f_В = f_Л = f_Н = f_П = (0, 0, 5)$$

а для ближней и дальней граней взять их центры

$$f_Б = (0, 0, 1) \text{ и } f_Д = (0, 0, -1)$$

Директриса $P_1P_2$ равна

$$\mathbf{D} = \mathbf{P}_2 - \mathbf{P}_1 = [7/2\ 5/2\ -1]$$

Для граничной точки на левой грани отсекателя

$$\mathbf{w} = \mathbf{P}_1 - \mathbf{f}_Л = [-2\ -1\ 1/2] - [0\ 0\ 5] = [-2\ -1\ -9/2]$$

Заметим, что

$$\mathbf{D}\cdot\mathbf{n}_Л = [7/2\ 5\ 2\ -1]\cdot[10\ 0\ -2] = 37 > 0$$

Таблица 3.12.

| Грань | n | f | w | w·n | D·n[1] | $t_H$ | $t_B$ |
|---|---|---|---|---|---|---|---|
| Верхняя | [ 0 −10 −2] | (0, 0, 5) | [−2 −1 −9/2] | 19 | −23 | | 0.826 |
| Нижняя | [ 0 10 −2] | (0, 0, 5) | [−2 −1 −9/2] | −1 | 27 | 0.037 | |
| Правая | [−10 0 −2] | (0, 0, 5) | [−2 −1 −9/2] | 29 | −33 | | 0.879 |
| Левая | [ 10 0 −2] | (0, 0, 5) | [−2 −1 −9/2] | −11 | 37 | 0.297 | |
| Ближняя | [ 0 0 −1] | (0, 0, 1) | [−2 −1 −1/2] | 1/2 | 1 | −0.5 | |
| Дальняя | [ 0 0 1] | (0, 0, −1) | [−2 −1 3/2] | 3/2 | −1 | | 1.5 |

[1] При $\mathbf{D}\cdot\mathbf{n} < 0$, верхний предел ($t_B$); при $\mathbf{D}\cdot\mathbf{n} > 0$, нижний предел ($t_H$)

значит, ищется нижний предел. Далее,

$$\mathbf{w}\cdot\mathbf{n}_Л = [-2\ -1\ -9/2]\cdot[10\ 0\ -2] = -11$$

и

$$t_H = -(-11)/37 = 11/37 = 0.297$$

Полностью результаты работы алгоритма даны в табл. 3.12. Анализ этой таблицы показывает, что максимальным из нижних значений будет $t_H = 0.297$, а минимальным из верхних будет $t_B = 0.826$. Из параметрического описания отрезка следует, что

$$P(0.297) = [-0.961\ -0.258\ 0.203]$$

и

$$P(0.826) = [0.891\ 1.065\ -0.323]$$

это соответствует пересечениям с левой и верхней гранями отсекателя

В последнем примере отсекателем будет нестандартное тело с семью гранями.

**Пример 3.19.** Отсечение относительно произвольного объема

Отсекатель изображен на рис. 3.23. Это куб, один из углов которого срезан. Грани задаются списками их вершин:

правая: (1, −1, 1), (1, −1, −1), (1, 1, −1), (1, 1, 1)
левая : (−1, −1, 1), (−1, −1, −1), (−1, 1, −1), (−1, 1, 0), (−1, 0, 1)
нижняя: (1, −1, 1), (1, −1, −1), (−1, −1, −1), (1, −1, −1)
верхняя: (1, 1, 1), (1, 1, −1), (−1, 1, −1), (−1, 1, 0), (0, 1, 1)
ближняя: (1, −1, 1), (1, 1, 1), (0, 1, 1), (−1, 0, 1), (−1, −1, 1)
дальняя: (−1, −1, −1), (1, −1, −1), (1, 1, −1), (−1, 1, −1)
косоугольная: (−1, 0, 1), (0, 1, 1), (−1, 1, 0)

В табл. 3.13 собраны все результаты работы алгоритма с отрезком от $P_1(-2, 3\ 2, 1)$ до $P_2(3/2, -1, -1/2)$, который отсекается относительно описанного объема.

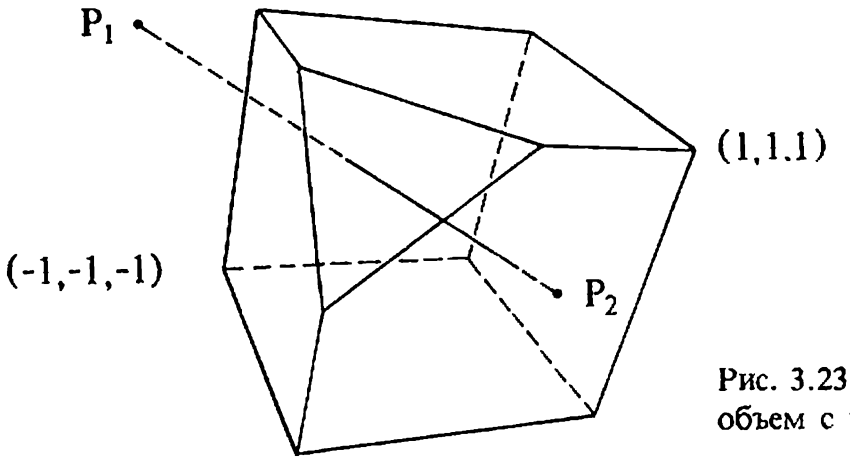



Рис. 3.23. Отсечение Кируса — Бека: объем с нечетным числом граней.

Из этой таблицы следует, что максимальным из нижних значений будет $t_н = 1/3$, а минимальным из верхних будет $t_в = 6/7$. Поэтому точками пересечения будут:

$$P(1\ 3) = [-5/6\ \ 2/3\ \ 1/2]$$

на косоугольной грани и

$$P(6/7) = [1\ \ -9/14\ \ -2/7]$$

на правой грани.

**Таблица 3.13.**

| Грань | n | f | w | w·n | D·n[1] | $t_н$ | $t_в$ |
|---|---|---|---|---|---|---|---|
| Верхняя | [ 0 −1 0] | ( 1, 1, 1) | [−3 1/2 0] | −1/2 | 5/2 | 1/5 | |
| Нижняя | [ 0 1 0] | (−1, −1, −1) | [−1 5/2 2] | 5/2 | −5/2 | | 1 |
| Правая | [−1 0 0] | ( 1, 1, 1) | [−3 1/2 0] | 3 | −7/2 | | 6/7 |
| Левая | [ 1 0 0] | (−1, −1, −1) | [−1 5/2 2] | −1 | 7/2 | 2/7 | |
| Ближняя | [ 0 0 −1] | ( 1, 1, 1) | [−3 1/2 0] | 0 | 3/2 | 0 | |
| Дальняя | [ 0 0 1] | (−1, −1, −1) | [−1 5/2 2] | 2 | −3/2 | | 4/3 |
| Косоугольная | [ 1 −1 −1] | (−1, 0, 1) | [−1 3/2 0] | −5/2 | 15/2 | 1/3 | |

[1] При $D \cdot n < 0$, верхний предел ($t_в$); при $D \cdot n > 0$, нижний предел ($t_н$).

Заметим, что оценка числа операций в алгоритме Кируса — Бека растет линейно с ростом числа сторон или граней у отсекателя.

## 3.12. ОТСЕЧЕНИЕ В ОДНОРОДНЫХ КООРДИНАТАХ

Когда отсечение необходимо провести в однородной системе координат [1-1], то нужно проявить осторожность, если при этом ис-

пользуется еще и преобразование проецирования. Главная причина возникающих затруднений объясняется тем, что отсекающая плоскость тут не обязательно разрезает отрезок на две части, одна из которых будет внутри, а другая — вне отсекателя. Отрезок может «заворачиваться» в бесконечности так, что внутри отсекателя будут лежать (и, следовательно, будут видимы) сразу две его части. Блинн [3-6] доказал, что отсечение всех отрезков *перед* завершением преобразования проецирования, осуществляемого путем деления всех координат на значение однородной координаты, удаляет отрезки, которые «возвращаются из бесконечности». Лианг и Барский [3-5] предложили алгоритм отсечения отрезков, работающий в однородных координатах. Они получили корректный результат путем искажения формы видимого объема, которым у них служила усеченная пирамида.

В алгоритме Кируса — Бека отрезок корректно отсекается относительно пирамиды видимости при условии, что он целиком расположен перед точкой наблюдения или, что то же самое, перед центром проекции (см. пример 3.18). Однако если отрезок заходит за центр проекции, то этот алгоритм отвергает его, даже если он частично видимый. На практике корректный результат получается, если отрезок сперва отсечь в обычной системе координат, а затем уже выполнить преобразование проецирования. Заметим, что и к отсекателю и к отрезку перед преобразованием проецирования можно применить любое аффинное преобразование (т. е. повороты, переносы и т. п.). Следующий пример проиллюстрирует вышеизложенные соображения.

**Пример 3.20.** Применение алгоритма Кируса — Бека к отрезку, заходящему за центр проекции

Возьмем отрезок от $P_1(0, 1, 6)$ до $P_2(0, -1, -6)$, отсекаемый пирамидой, заданной в обычной системе координат значениями $(x_Д, x_Л, y_Н, y_В, z_Б, z_Д) = (-1, 1, -1, 1, 1, -1)$, причем центр проекции расположен в точке с $z = 5$. Отрезок $P_1P_2$ пересекает видимый объем, но начинается за центром проекции.

После выполнения преобразования проецирования (см. [1-1]) концы отрезка будут иметь следующие значения однородных координат:

$$P_1[0\ 1\ 6\ -1/5] \text{ и } P_2[0\ -1\ -6\ 11/5]$$

Переход к обычным координатам путем деления на однородную координату дает

$$P_1(0, -5, -30) \text{ и } P_2(0, -5\ 11, -30/11)$$

Заметим, что исходная точка $P_1$ лежала перед отсекателем, но дальше центра проекции, а теперь она, «завернувшись» в бесконечности, оказалась за отсекателем. По-

**Таблица 3.14.**

| Грань | n | f | w | w·n | D·n[1] | $t_н$ | $t_в$ |
|---|---|---|---|---|---|---|---|
| Верхняя | [ 0 −1 0] | ( 1, 1, 1) | [−1 0 5] | 0 | 2 | 0 | |
| Нижняя | [ 0 1 0] | (−1, −1, −1) | [ 1 2 7] | 2 | −2 | | 1 |
| Правая | [−1 0 0] | ( 1, 1, 1) | [−1 0 5] | 1 | 0 | | |
| Левая | [ 1 0 0] | (−1, −1, −1) | [ 1 2 7] | 1 | 0 | | |
| Ближняя | [ 0 0 −1] | ( 1, 1, 1) | [−1 0 5] | −5 | 12 | 5/12 | |
| Дальняя | [ 0 0 1] | (−1, −1, −1) | [ 1 2 7] | 7 | −12 | | 7/12 |

[1] При $\mathbf{D}\cdot\mathbf{n} < 0$, верхний предел ($t_в$); при $\mathbf{D}\cdot\mathbf{n} > 0$, нижний предел ($t_н$).

скольку теперь оба конца отрезка оказались вне отсекателя[2], то алгоритм Кируса — Бека отбросит его как невидимый.

Взяв значения внутренних нормалей и координат точек, лежащих на каждой из граней отсекателя, из примера 3.17, проведем теперь вначале отсечение относительно объема (−1, 1, −1, 1, 1, −1). Здесь директриса отрезка $P_1P_2$ равна

$$\mathbf{D} = \mathbf{P}_2 - \mathbf{P}_1 = [0\ -1\ -6] - [0\ 1\ 6] = [0\ -2\ -12]$$

Результаты отсечения приведены в табл. 3.14.

Анализ табл. 3.14 показывает, что отрезок видим в интервале $5/12 \leqslant t \leqslant 7/12$. Координаты концов видимого участка отрезка равны:

$$P(5/12) = [0\ 1\ 6] + [0\ -2\ -12](5/12) = [0\ 1/6\ 1]$$
$$P(7/12) = [0\ 1\ 6] + [0\ -2\ -12](7/12) = [0\ -1/6\ -1]$$

Выполняя преобразование проецирования этих точек в однородных координатах [1-1], имеем для видимого куска отрезка:

$$\begin{bmatrix} 0 & 1/6 & 1 & 1 \\ 0 & -1/6 & -1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -1/5 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 5/24 & 5/4 & 1 \\ 0 & -5/36 & -5/6 & 1 \end{bmatrix}$$

Полученный результат корректен.

[2] И по одну сторону от его дальней грани. — *Прим. перев.*

## 3.13. ОПРЕДЕЛЕНИЕ ВЫПУКЛОСТИ ТРЕХМЕРНОГО ТЕЛА И ВЫЧИСЛЕНИЕ ВНУТРЕННИХ НОРМАЛЕЙ К ЕГО ГРАНЯМ.

Ранее описанный двумерный алгоритм определения выпуклости многоугольника и вычисления его внутренних нормалей, который использует повороты и переносы, можно обобщить на случай трехмерных многогранников. Эта процедура будет выглядеть так:

Для каждой полигональной грани тела выполнить следующее:

Перенести тело так, чтобы одна из вершин грани оказалась в начале координат.

Повернуть тело относительно начала координат так, чтобы одна из двух смежных выбранной вершине сторон грани совпала с одной из осей координат, например с осью $x$.

Повернуть тело вокруг выбранной оси координат так, чтобы выбранная грань легла на координатную плоскость, например на плоскость $z = 0$.

Для всех вершин тела, не принадлежащих выбранной грани, проверить знаки координаты, которая перпендикулярна этой грани; здесь это будет координата $z$.

Если эти знаки для всех вершин совпадают или равны нулю, то тело будет выпуклым относительно выбранной грани. Если тело выпукло относительно всех своих граней, то оно считается выпуклым, — в противном случае тело невыпукло.

Если для всех вершин значения координаты, перпендикулярной выбранной грани, равны нулю, то тело вырождено; т. е. оно плоское.

Вектор внутренней нормали к выбранной плоскости, заданный в повернутой системе координат, имеет все нулевые компоненты, кроме той, которая перпендикулярна этой плоскости. Знак этой компоненты для выпуклой грани будет совпадать с ранее найденным знаком.

Для определения искомой ориентации внутренней нормали в исходной системе координат нужно применить к ней только обратное преобразование поворотов.

**Пример 3.21.** Определение выпуклости тела

В качестве частного случая рассмотрим куб со срезанным углом, который уже описывался выше в примере 3.19. Этот куб изображен на рис. 3.24, а. Определим при

помощи изложенного алгоритма факт выпуклости этого отсекателя относительно грани, которая обозначена на рис. 3.24, а буквами *abc*. Сначала перенесем это тело так, чтобы точка *a* совпала с началом координат. Матрица 4 × 4 этого преобразования в однородных координатах такова (см. [1-1]):

$$[T] = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & -1 & 1 \end{bmatrix}$$

Проекция результата на плоскость $z = 0$ показана на рис. 3.24, b. Поворот тела вокруг оси $z$ на угол $\theta = -45°$ совмещает ребро $ab$ с осью $x$. Матрица этого преобразования в однородных координатах имеет вид (см. [1-1]):

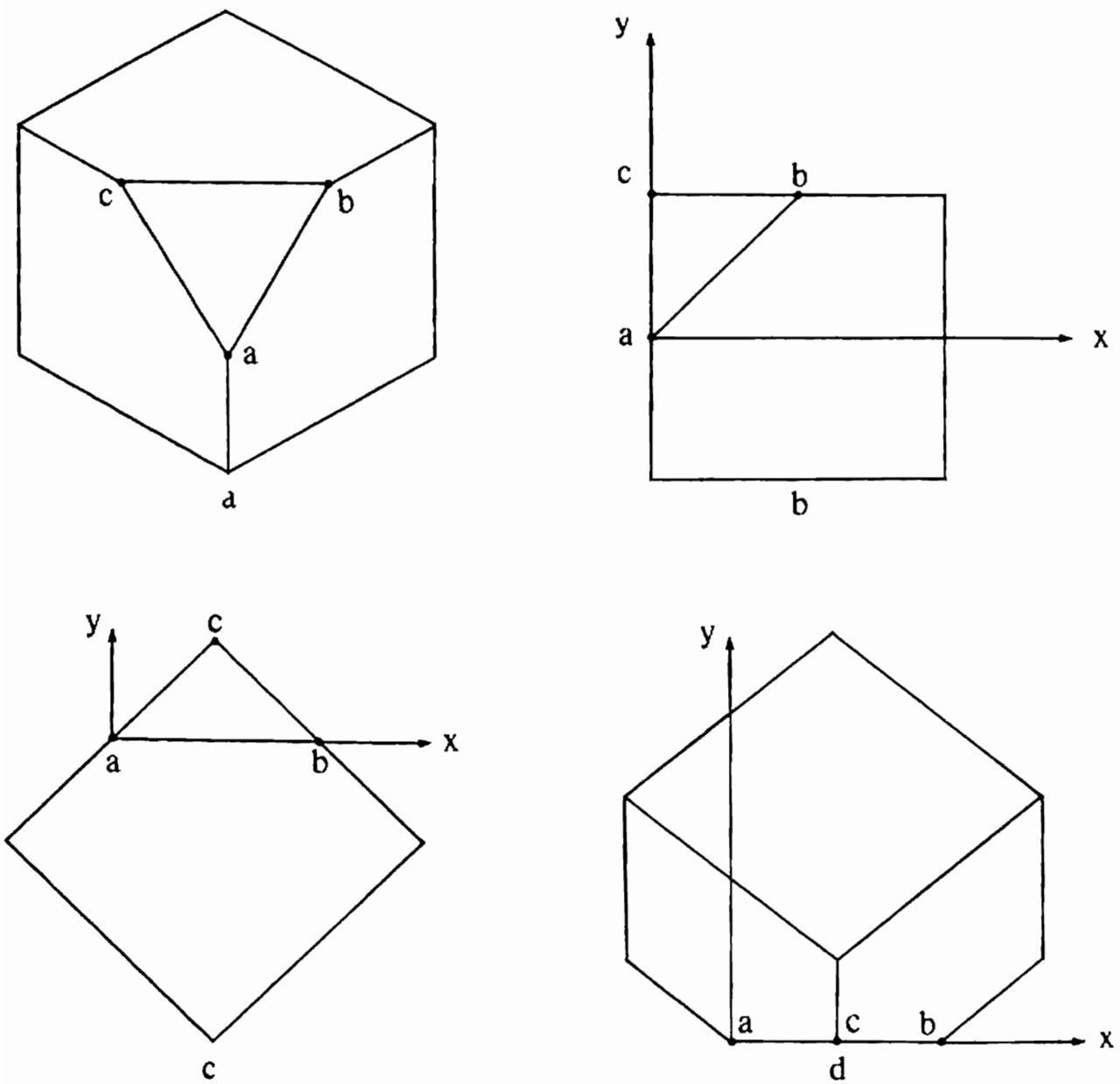



Рис. 3.24. Определение выпуклости тела и нахождение внутренней нормали.

$$[R_z] = \begin{bmatrix} \cos\theta & \sin\theta & 0 & 0 \\ -\sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Проекция результата этого преобразования на плоскость $z = 0$ показана на рис. 3.24, с. Остается сделать поворот вокруг оси $x$, чтобы совместить плоскость $abc$ с координатной плоскостью $y = 0$. Координаты точки $c$ на рис. 3.24, с равны (0.565685, 0.565685, −0.8). Угол поворота относительно оси x равен:

$$\alpha = \operatorname{arctg}(y\ z) = \operatorname{arctg}[0.565685/(-0.8)] = -35.2644°.$$

Поворот на этот угол поместит тело под координатной плоскостью $y = 0$. А поворот на угол $(180° - \alpha)$ поместит это тело над указанной плоскостью. Последний результат изображен на рис. 3.24, d в проекции на плоскость $z = 0$. Матрица этого поворота равна:

$$[R_x] = \begin{bmatrix} 1 & 0 & 0 & 0 \\ \cos\alpha & \sin\alpha & 0 & 0 \\ -\sin\alpha & \cos\alpha & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Ординаты всех вершин тела, не лежащих на плоскости $y = 0$, положительны. Значит, это тело выпукло относительно грани $abc$.

Внутренняя нормаль к грани $abc$, заданной в результирующем положении, равна

$$\mathbf{n}' = [0\ \operatorname{sign}(y)\ 0] = [0\ 1\ 0]$$

Обратный поворот даст

$$\mathbf{n} = [0.5777\ -0.5774\ -0.5774]$$

или

$$\mathbf{n} = [1\ -1\ -1]$$

что и ожидалось. Для доказательства выпуклости тела описанную процедуру следует повторить для всех остальных его граней

## 3.14. РАЗРЕЗАНИЕ НЕВЫПУКЛЫХ ТЕЛ

Трехмерная версия алгоритма Кируса — Бека работает с выпуклыми отсекателями. Вместе с тем существует потребность отсечения относительно невыпуклых тел. Ее можно удовлетворить путем реализации внутренних и внешних отсечений выпуклыми объемами, из которых состоит невыпуклое тело. Этот подход аналогичен тому методу, который уже обсуждался ранее в связи с отсечением невыпуклыми многоугольниками (разд. 3.6). Задачу разрезания простого невыпуклого тела на составляющие его выпуклые тела можно решить путем обобщения метода переносов и поворотов, который был описан в предыдущем разделе. В алгоритме предполагается, что тело представляет собой многогранник с плоскими гранями.

Процедура разрезания такова:

Для каждой полигональной грани тела

Перенести тело так, чтобы одна из вершин выбранной грани совпала с началом координат.

Повернуть тело вокруг начала координат так, чтобы одно из инцидентных ему ребер совпало с одной из осей координат, например с осью $x$.

Повернуть тело вокруг выбранной оси координат так, чтобы выбранная грань совпала с одной из координатных плоскостей, например с плоскостью $z = 0$.

Проверить знаки координаты, которая перпендикулярна выбранной грани (т. е. координаты $z$), для всех вершин тела, не лежащих на этой грани.

Если все эти знаки совпадают или равны нулю, то тело является выпуклым относительно этой грани. В противном случае оно невыпукло; разрезать тело плоскостью, несущей выбранную грань.

Повторить всю процедуру с каждым из вновь образовавшихся тел. Продолжать работу до тех пор, пока все тела не станут выпуклыми.

**Пример 3.22.** Разрезание невыпуклого тела

Рассмотрим невыпуклое тело, изображенное на рис. 3.25, а. Его грани задаются следующими вершинами:

| | |
|---|---|
| задняя: | $P_1(3, 0, 0)$, $P_2(0, 0, 0)$, $P_3(0, 2, 0)$, $P_4(1, 2, 0)$ |
| | $P_5(1, 3/2, 0)$, $P_6(3/2, 3/2, 0)$, $P_7(3/2, 2, 0)$, $P_8(3, 2, 0)$ |
| передняя: | $P_9(3, 0, 2)$, $P_{10}(0, 0, 2)$, $P_{11}(0, 2, 2)$, $P_{12}(1, 2, 2)$ |
| | $P_{13}(1, 3/2, 2)$, $P_{14}(3/2, 3/2, 2)$, $P_{15}(3/2, 2, 2)$, $P_{16}(3, 2, 2)$ |
| левая: | $P_2(0, 0, 0)$, $P_{10}(0, 0, 2)$, $P_{11}(0, 2, 2)$, $P_3(0, 2, 0)$ |
| правая: | $P_1(3, 0, 0)$, $P_8(3, 2, 0)$, $P_{16}(3, 2, 2)$, $P_9(3, 0, 2)$ |
| нижняя: | $P_1(3, 0, 0)$, $P_2(0, 0, 0)$, $P_{10}(0, 0, 2)$, $P_9(3, 0, 2)$ |
| верхняя слева: | $P_{12}(1, 2, 2)$, $P_4(1, 2, 0)$, $P_3(0, 2, 0)$, $P_{11}(0, 2, 2)$ |
| левая у выемки: | $P_{13}(1, 3/2, 2)$, $P_5(1, 3/2, 0)$, $P_4(1, 2, 0)$, $P_{12}(1, 2, 2)$ |
| нижняя у выемки: | $P_{13}(1, 3/2, 2)$, $P_{14}(3/2, 3/2, 2)$, $P_6(3/2, 3/2, 0)$, $P_5(1, 3/2, 0)$ |
| правая у выемки: | $P_6(3/2, 3/2, 0)$, $P_7(3/2, 2, 0)$, $P_{15}(3/2, 2, 2)$, $P_{14}(3/2, 3/2, 2)$ |
| верхняя справа: | $P_{16}(3, 2, 2)$, $P_8(3, 2, 0)$, $P_7(3/2, 2, 0)$, $P_{15}(3/2, 2, 2)$ |

Проверим выпуклость этого тела относительно грани, названной «левая у выемки» и помеченной буквами *abc* на рис. 3.25, а, с помощью изложенного выше алго-

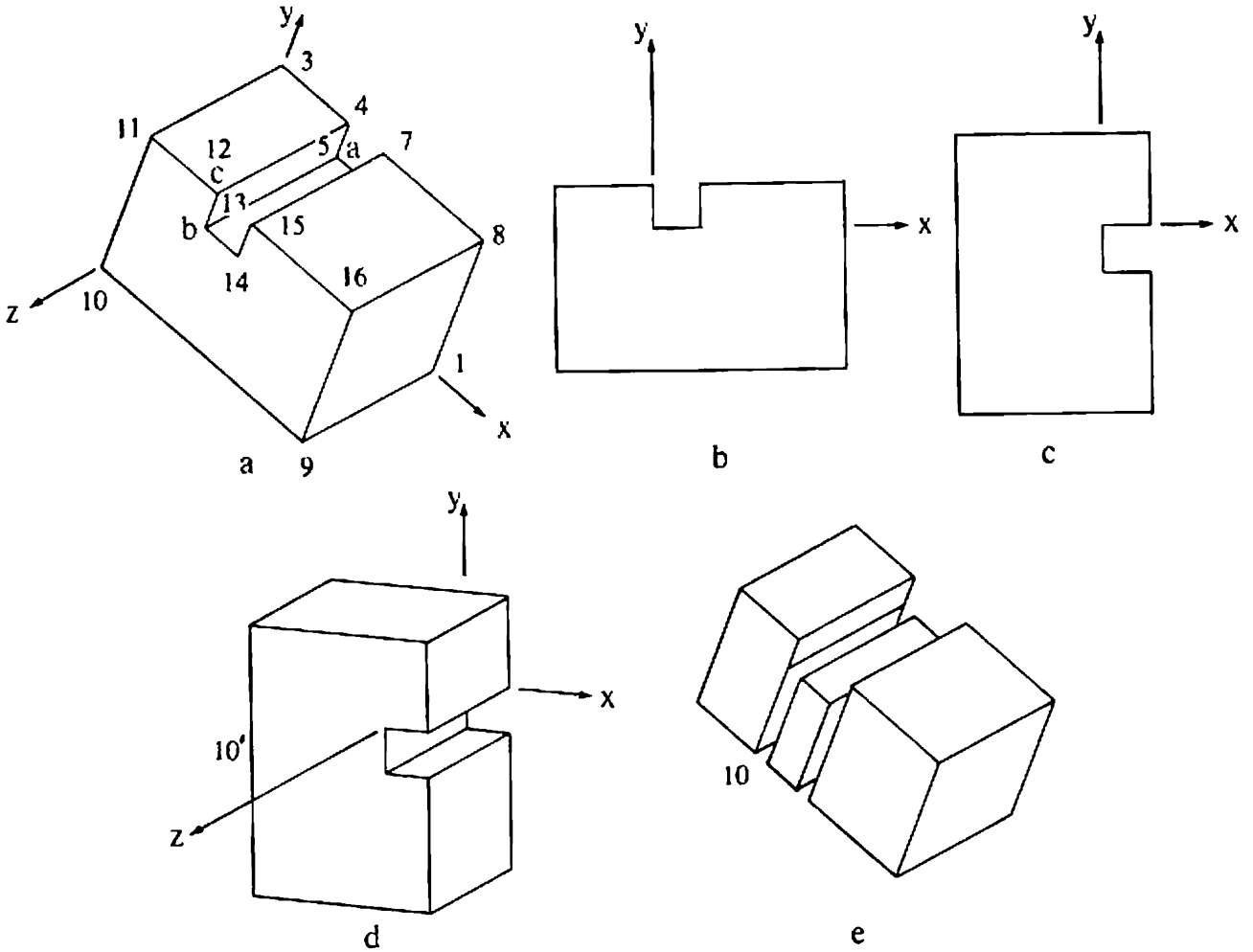



Рис. 3.25. Разрезание невыпуклого тела.

ритма. Сначала тело переносится так, чтобы вершина $P_5$, помеченная на рис. 3.25, а буквой $a$, совпала с началом координат. Одновременно этот перенос совмещает вершину $P_{13}$, помеченную на рис. 3.25, а буквой $b$, с положительной полуосью $z$. Компоненты вектора переноса относительно осей $x$, $y$, $z$ равны $(-1, -3/2, 0)$ соответственно. Результат переноса в проекции на плоскость $z = 0$ показан на рис. 3.25, b. Поворот тела на $-90°$ вокруг оси $z$ совмещает плоскость $abc$ с координатной плоскостью $y = 0$. Результат этого поворота в проекции на плоскость $z = 0$ и в перспективе показаны на рис. 3.25, c и d.

Проверка знаков ординат вершин показывает, что тело невыпукло. Оно разрезается плоскостью $y = 0$ на два тела $V_1$ и $V_2$. Тело $V_1$ лежит выше плоскости $y = 0$, а тело $V_2$ — ниже. Грани этих тел описываются следующими вершинами, координаты которых заданы в исходном положении тела:

$V_1$

| | |
|---|---|
| левая: | $P_2(0, 0, 0)$, $P_{10}(0, 0, 2)$, $P_{11}(0, 2, 2)$, $P_3(0, 2, 0)$ |
| правая (нижняя): | $P'_{10}(1, 0, 2)$, $P'_5(1, 0, 0)$, $P_5(1, 3/2, 0)$, $P_{13}(1, 3/2, 2)$ |
| правая (верхняя): | $P_{13}(1, 3/2, 2)$, $P_5(1, 3/2, 0)$, $P_4(1, 2, 0)$, $P_{12}(1, 2, 2)$ |
| верхняя: | $P_{12}(1, 2, 2)$, $P_4(1, 2, 0)$, $P_3(0, 2, 0)$, $P_{11}(0, 2, 2)$ |
| нижняя: | $P_2(0, 0, 0)$, $P'_5(1, 0, 0)$, $P'_{10}(1, 0, 2)$, $P_{10}(0, 0, 2)$ |

передняя: $P_{10}(0, 0, 2)$, $P'_{10}(1, 0, 2)$, $P_{13}(1, 3/2, 2)$, $P_{12}(1, 2, 2)$, $P_{11}(0, 2, 2)$
задняя: $P'_5(1, 0, 0)$, $P_2(0, 0, 0)$, $P_3(0, 2, 0)$, $P_4(1, 2, 0)$, $P_5(1, 3/2, 0)$

$V_2$
левая: $P_5(1, 0, 0)$, $P'_{10}(1\ 0, 2)$, $P_{13}(1, 3'2, 2)$, $P_5(1, 3/2, 0)$
правая: $P_1(3, 0, 0)$, $P_8(3, 2, 0)$, $P_{16}(3, 2, 2)$, $P_9(3, 0, 2)$
правая у выемки: $P_6(3/2, 3/2, 0)$, $P_7(3/2, 2, 0)$, $P_{15}(3/2, 2, 2)$, $P_{14}(3/2, 3/2, 2)$
нижняя у выемки: $P_{13}(1, 3/2, 2)$, $P_{14}(3/2, 3/2, 2)$, $P_6(3/2, 3/2, 0)$, $P_5(1, 3/2, 0)$
верхняя справа: $P_{16}(3, 2, 2)$, $P_8(3, 2, 0)$, $P_7(3/2, 2, 0)$, $P_{15}(3/2, 2, 2)$
нижняя $P'_5(1, 0, 0)$, $P_1(3, 0, 0)$, $P_9(3, 0, 2)$, $P'_{10}(1, 0, 2)$

После повторной обработки алгоритмом каждого из этих тел $V_1$ будет объявлен выпуклым, а $V_2$ будет разрезан на два новых тела, которые потом тоже окажутся выпуклыми. Окончательный результат в разрезанном виде показан на рис. 3.25, е.

## 3.15. ОТСЕЧЕНИЕ МНОГОУГОЛЬНИКОВ

Предыдущее обсуждение было связано с отсечением отрезков. Разумеется, многоугольник можно рассматривать как набор отрезков. В приложениях, связанных с вычерчиванием штриховых изображений, не слишком существенно, если многоугольник разбит на отрезки до его отсечения. Если замкнутый многоугольник отсекается, как набор отрезков, то исходная фигура может превратиться в один или более открытых многоугольников или просто стать совокупностью разрозненных отрезков, как показано на рис. 3.26. Однако если многоугольники рассматриваются как сплошные области, то необходимо, чтобы замкнутость сохранялась и у результата. Для примера на рис. 3.26 это означает, что отрезки *bc, ef, fg* и *ha*

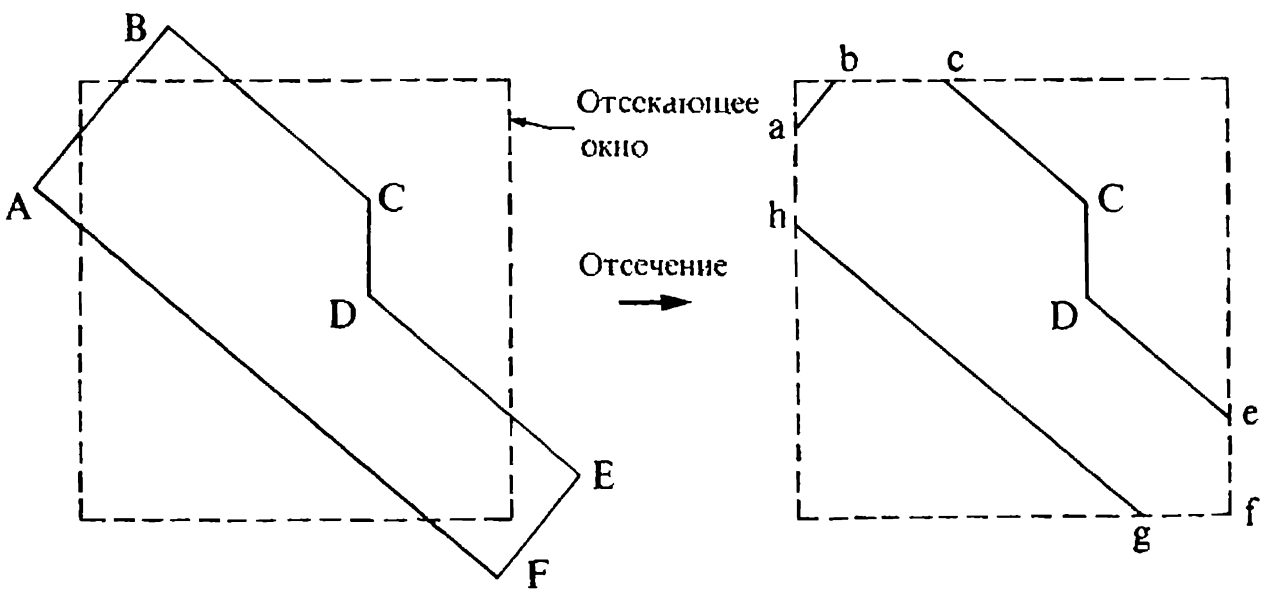



Рис. 3.26. Отсечение многоугольника: открытый многоугольник.

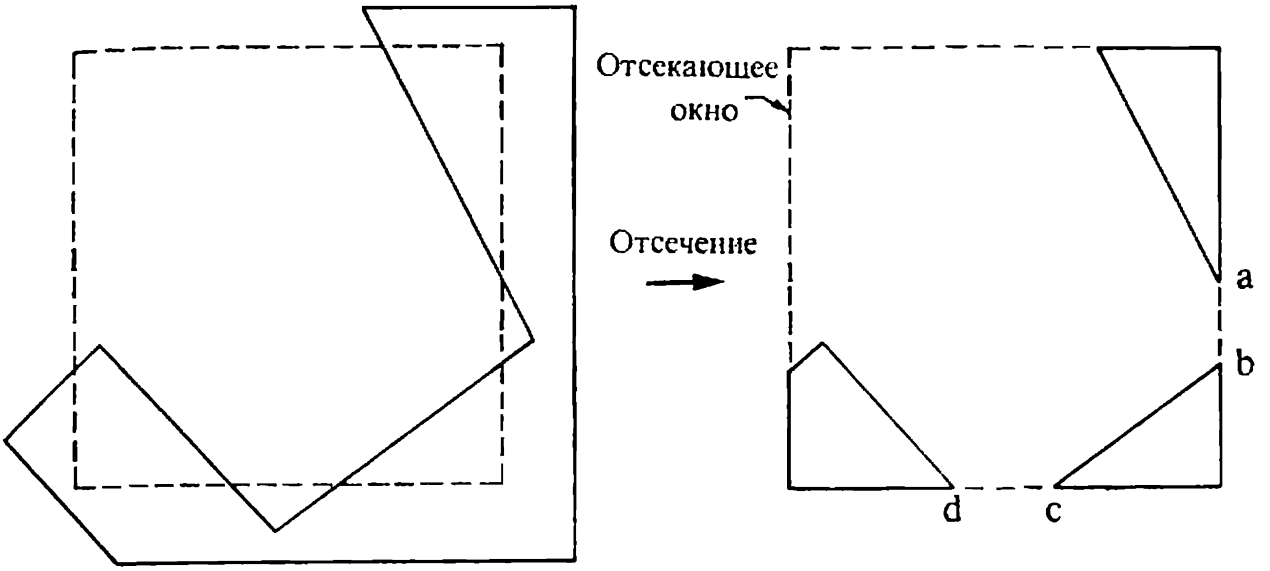



Рис. 3.27. Отсечение многоугольника: не связанные между собой многоугольники.

должны быть добавлены к описанию результирующего многоугольника. Добавление отрезков *ef* и *fg* представляет особые трудности. Трудные вопросы возникают и тогда, когда результат отсечения представляет собой несколько несвязанных между собой многоугольников меньших размеров, как это показано на рис. 3.27. Например, иногда отрезки *ab* и *cd*, показанные на рис. 3.27, включаются в описание результата. Если, например, исходный многоугольник объявлен красным на синем фоне, то отрезки *ab* и *cd* тоже будут выглядеть красными на синем фоне. Это противоречит ожидаемому результату.

## 3.16. ПОСЛЕДОВАТЕЛЬНОЕ ОТСЕЧЕНИЕ МНОГОУГОЛЬНИКА — АЛГОРИТМ САЗЕРЛЕНДА — ХОДЖМЕНА

Основная идея алгоритма Сазерленда — Ходжмена [3-7] состоит в том, что отсечь многоугольник относительно одной прямой или плоскости очень легко. В этом алгоритме исходный и каждый из промежуточных многоугольников отсекается последовательно относительно одной прямой. Работа алгоритма для прямоугольного окна показана на рис. 3.28. Исходный многоугольник задается списком вершин $P_1, \ldots, P_n$, который порождает список его ребер $P_1P_2$, $P_2P_3, \ldots, P_{n-1}P_n, P_nP_1$. На рис. 3.28 показано, что многоугольник сначала отсекается левой стороной окна, в результате чего получается промежуточная фигура. Затем алгоритм вновь отсекает эту фигуру верхней стороной окна. Получается вторая промежуточная фигура. Далее процесс отсечения продолжается с оставшимися сто-

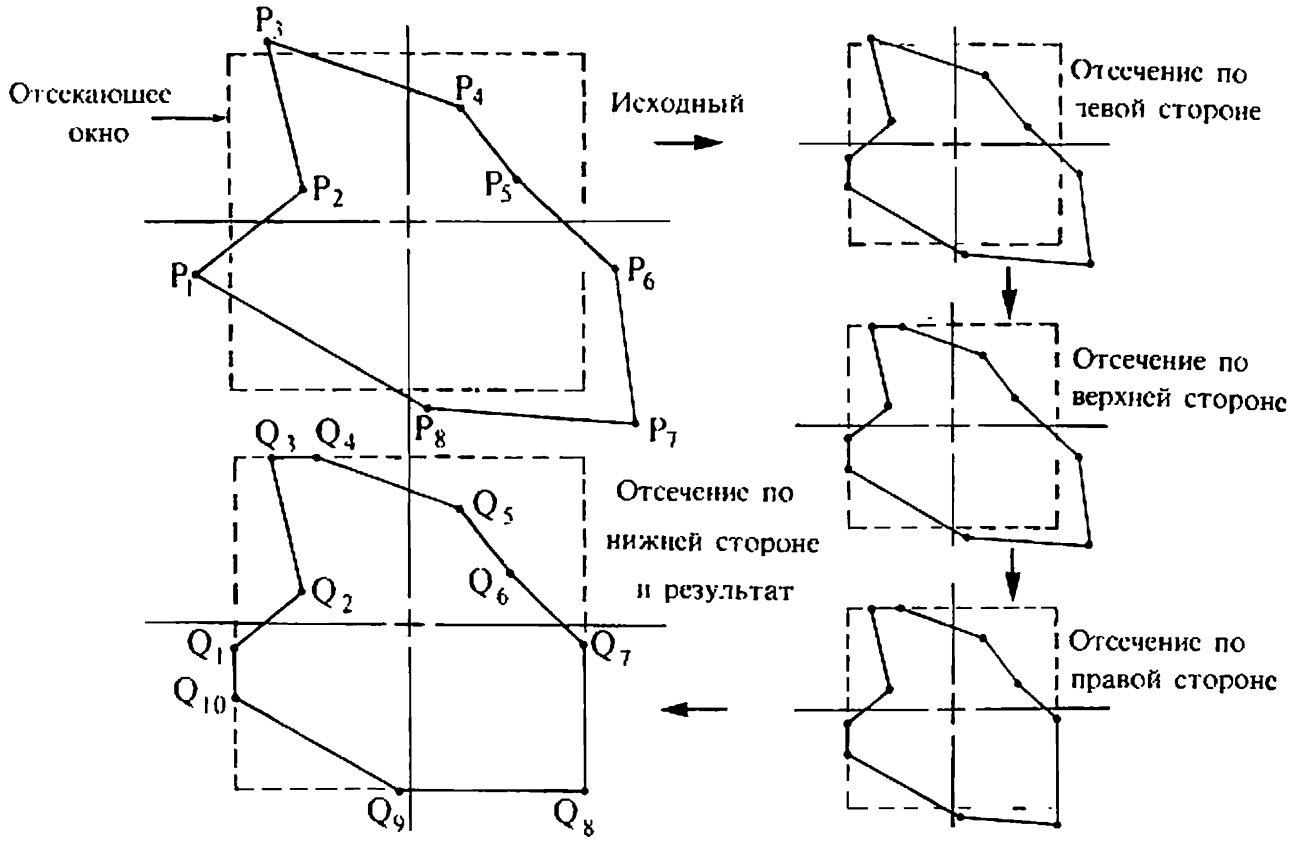
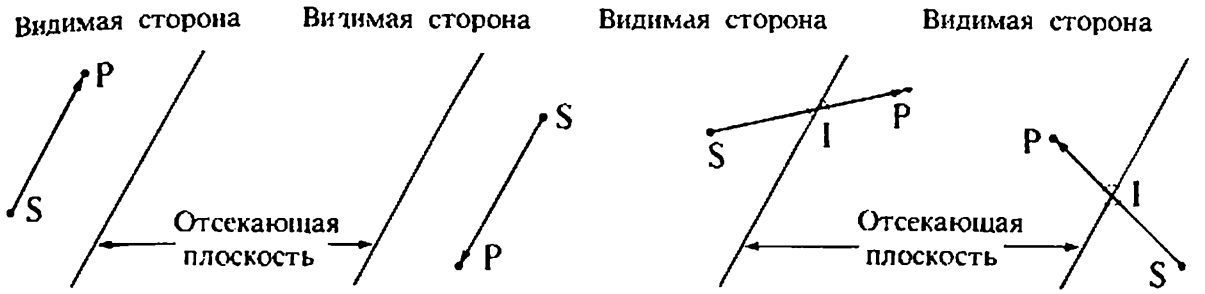



Рис. 3.28. Последовательное отсечение многоугольника.

ронами окна. Этапы отсечения показаны на рис. 3.28. Заметим, что добавление угловой точки $Q_8$ в окончательный результат отсечения теперь стало тривиальным. Этот алгоритм способен отсекать любой многоугольник, выпуклый или невыпуклый, плоский или неплоский, относительно любого окна, являющегося выпуклым многоугольником. Порядок отсечения многоугольника разными сторонами окна непринципиален.

Результатом работы алгоритма является список вершин многоугольника, у которого все вершины лежат по видимую сторону от очередной отсекающей плоскости. Поскольку каждая сторона многоугольника отсекается независимо от других, то достаточно рассмотреть только возможные ситуации расположения одного отрезка относительно одной отсекающей плоскости. Будем рассматривать каждую точку $P$ из списка вершин многоугольника, за исключением первой, как конечную точку ребра, начальной точкой $S$ которого является вершина, предшествующая $P$ в этом списке. Тогда возможны только четыре ситуации взаимного расположения ребра и отсекающей плоскости. Они показаны на рис. 3.29.

Результатом каждого сопоставления ребра многоугольника с отсекающей плоскостью будет занесение в список вершин результирующего усеченного многоугольника нуля, одной или двух вершин.




Полная видимость. Результат P. 1 точка.

Полная невидимость. Нет результата. 0 точек.

Выход из области видимости. Результат I. 1 точка.

Вход в область видимости. Результат I и P 2 точки.

Рис. 3.29. Взаимное расположение ребер и отсекающей плоскости.

Если рассматриваемое ребро полностью видимо, то результатом будет вершина $P$. Заносить в результат начальную вершину $S$ в этом случае не надо, так как если вершины рассматриваются поочередно, то $S$ уже была конечной точкой предыдущего ребра и поэтому уже попала в результат. Если же ребро полностью невидимо, то результат не изменяется.

Если ребро многоугольника видимо неполностью, то оно может или входить или выходить из области видимости отсекающей плоскости. Если ребро выходит из области видимости, то надо определить и занести в результат точку пересечения ребра и отсекающей плоскости. Если же ребро входит в область видимости, то следует поступить точно так же. Поскольку в последнем случае конечная вершина $P$ ребра видима, то она также должна попасть в результат.

Для первой вершины многоугольника необходимо определить только факт ее видимости. Если вершина видима, то она попадает в результат и становится начальной точкой $S$. Если же вершина невидима, она тоже становится начальной точкой, но в результат не попадает.

Последнее ребро — $P_n P_1$ — следует рассмотреть особо. Это реализуется путем запоминания первой вершины многоугольника в $F$. Тогда последним ребром становится $P_n F$, и его можно обрабатывать точно так же, как и любое другое ребро.

Прежде чем описать алгоритм полностью, приведем два дополнительных соображения, касающихся определения видимости точки и определения пересечения ребра многоугольника с отсекающей плоскостью. Определение видимости точки эквивалентно определению той стороны границы отсекающей плоскости, по которую лежит эта точка. Если ребра отсекающего многоугольника обходятся по часовой стрелке, то его внутренность лежит по правую сторону

от границы. При противоположном порядке обхода она лежит по левую сторону. Ранее рассматривались два метода определения положения (видимости) точки относительно ориентированного отрезка или плоскости. Первый сводится к определению знака скалярного произведения вектора нормали на вектор, начинающийся в произвольной точке на прямой или плоскости и заканчивающийся в пробной точке (см. разд. 3.5). Второй метод заключается в подстановке координат пробной точки в уравнение ориентированной прямой или плоскости (см. разд. 3.9). Последний метод является вариантом того, что было предложено Сазерлендом и Ходжменом в [3-7].

Третий метод определения видимости сводится к проверке знака координаты $z$ у векторного произведения двух векторов, лежащих в одной плоскости. Пусть две точки $P_1$ и $P_2$ лежат на отсекающей плоскости, а $P_3$ — это пробная точка. Эти три точки задают некую плоскость, на которой лежат два вектора: $\mathbf{P_1P_2}$ и $\mathbf{P_1P_3}$. Если эту плоскость считать плоскостью $xy$, то у векторного произведения векторов $\mathbf{P_1P_3} \otimes \mathbf{P_1P_2}$ ненулевой будет только компонента $z$, равная $(x_3 - x_1)(y_2 - y_1) - (y_3 - y_1)(x_2 - x_1)$. Если знак этой компоненты $z$ будет положительным, нулевым или отрицательным, то $P_3$ будет лежать соответственно справа, на или слева от прямой $P_1P_2$.

Все эти методы реализуются особенно просто для случая прямоугольных отсекающих окон, стороны которых параллельны координатным осям.

**Пример 3.23.** Определение положения точки относительно плоскости

Рассмотрим отсекающую плоскость $x = 1$, которая перпендикулярна оси $x$, как показано на рис. 3.30. Нужно определить положение двух точек $P_3(-2, 1)$ и $P_3'(2, 1)$ относительно данной отсекающей плоскости.

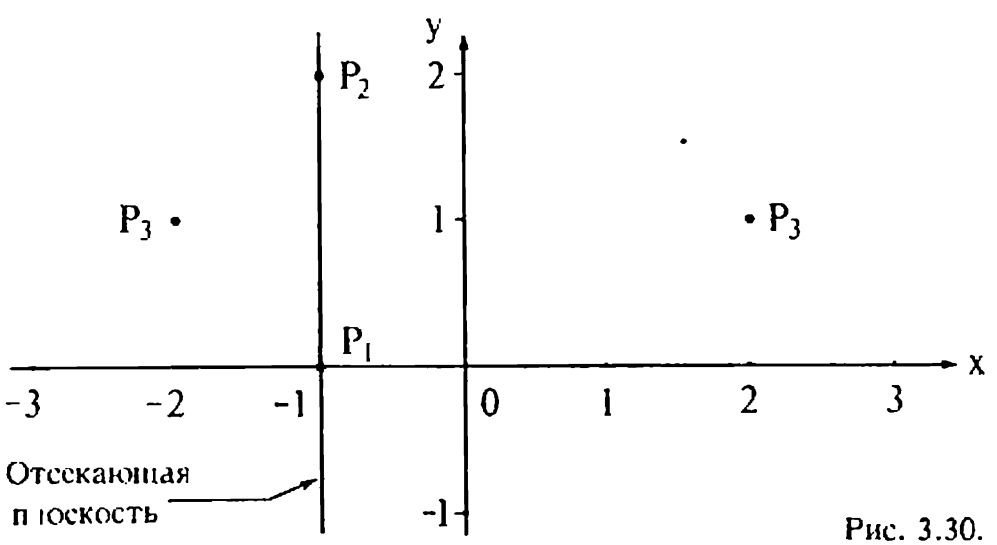



Рис. 3.30. Тесты видимости.

Используя метод векторного произведения с $P_1(-1, 0)$ и $P_2(-1, 2)$, получаем для случая точки $P_3$:

$$(x_3 - x_1)(y_2 - y_1) = [-2 - (-1)] \cdot (2 - 0) = -2 < 0,$$

это означает, что $P_3$ лежит слева от $\mathbf{P_1P_2}$. Для $P_3'$ имеем:

$$(x_3' - x_1)(y_2 - y_1) = [2 - (-1)] \cdot 2 = 6 > 0,$$

это означает, что $P_3$ лежит справа от $\mathbf{P_1P_2}$.

Метод подстановки координат особенно прост. Здесь пробная функция равна $x - w$. Для точки $P_3$:

$$x_3 - w = -2 - (-1) = -1 < 0$$

для $P_3'$:

$$x_3' - w = 2 - (-1) = 3 > 0$$

Это доказывает, что $P_3$ и $P_3'$ лежат слева и справа от $\mathbf{P_1P_2}$ соответственно.

Взяв в качестве внутренней нормали $\mathbf{n} = [1\ 0]$, а в качестве точки на отсекающей плоскости $\mathbf{f}(-1, 0)$, получаем скалярное произведение двух векторов:

$$\text{для } P_3\text{: } \mathbf{n} \cdot [\mathbf{P}_3 - \mathbf{f}] = [1\ 0]\ [-1\ 1] = -1 < 0$$
$$\text{для } P_3'\text{: } \mathbf{n} \cdot [\mathbf{P}_3' - \mathbf{f}] = [1\ 0]\ [3\ 1] = 3 > 0$$

Это опять доказывает, что $P_3$ расположена слева, а $P_3'$ — справа от отсекающей плоскости.

При использовании этих тестов видимости ребро многоугольника будет полностью видимым, если оба его конца видимы, и полностью невидимым, если оба они невидимы. Если же один конец ребра видим, а другой невидим, то ребро пересекается с отсекающей плоскостью и нужно вычислять точку пересечения. Для этого можно использовать любые изложенные выше алгоритмы отсечения отрезка, например Кируса — Бека (разд. 3.5), простой или параметрический (разд. 3.1 и 3.4) или разбиение средней точкой (разд. 3.3). Подчеркнем опять, и это было показано выше, что реализация данных алгоритмов особенно проста в случае, когда отсекающее окно является прямоугольником со сторонами, параллельными координатным осям. Алгоритмы Кируса — Бека и разбиение средней точкой, разумеется, применяются во всей своей общности. Однако пересечение двух параметрически заданных отрезков, лежащих в одной плоскости, требует некоторого уточнения.

Два отрезка с концевыми точками $P_1$, $P_2$ и $P_3$, $P_4$ соответственно можно задать параметрически следующим образом:

$$P(s) = P_1 + (P_2 - P_1)s,\ 0 \leqslant s \leqslant 1$$

и

$$P(t) = P_3 + (P_4 - P_3)t,\ 0 \leqslant t \leqslant 1$$

В точке их пересечения $P(s) = P(t)$. Напомним, что $P(s)$ и $P(t)$

являются векторно-значными функциями, т. е. $P(s) = [x(s)\ y(s)]$, а $P(t) = [x(t)\ y(t)]$. Отсюда следует, что из последнего векторного уравнения получаются два скалярных уравнения с двумя неизвестными $s$ и $t$, т. е. в точке пересечения $x(s) = x(t)$ и $y(s) = y(t)$. Если последние уравнения вообще не имеют решения, то отрезки параллельны. Если же решение есть, т. е. $s$ или $t$ выходит за пределы допустимой области, то отрезки не имеют общих точек. Особенно удобна матричная форма записи последнего уравнения.

**Пример 3.24.** Пересечение параметрически заданных отрезков

Возьмем два отрезка, показанных на рис. 3.31, от $P_1[0\ 0]$ до $P_2[3\ 2]$ и от $P_3[3\ 0]$ до $P_4[0\ 2]$. Здесь

$$P(s) = [0\ 0] + [3\ 2]s, \text{ а } P(t) = [3\ 0] + [-3\ 2]t$$

Приравнивание компонент $x$ и $y$ этих векторов дает систему из двух уравнений:

$$3s = 3 - 3t$$
$$2s = 2t$$

Ее решением является

$$s = t = 1/2$$

Точка пересечения имеет координаты:

$$P_I(s) = [0\ 0] + [3\ 2] \cdot (1/2) = [3/2\ 1]$$

Как уже упоминалось, Сазерленд и Ходжмен [3-7] предложили новый метод формирования последовательности промежуточных многоугольников. Напомним, что в их алгоритме ребра многоугольника обрабатываются поочередно. А это значит, что можно использовать с минимальными изменениями прежние коды концевых точек ребер. Последняя вершина многоугольника обрабатывается особо. На рис. 3.32 приведена блок-схема этого алгоритма, заимствованная с некоторыми изменениями из работы [3-7]. На

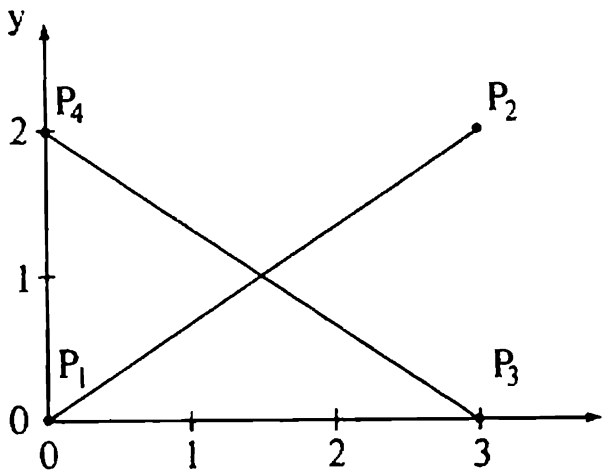



Рис. 3.31. Пересечение параметрически заданных отрезков.

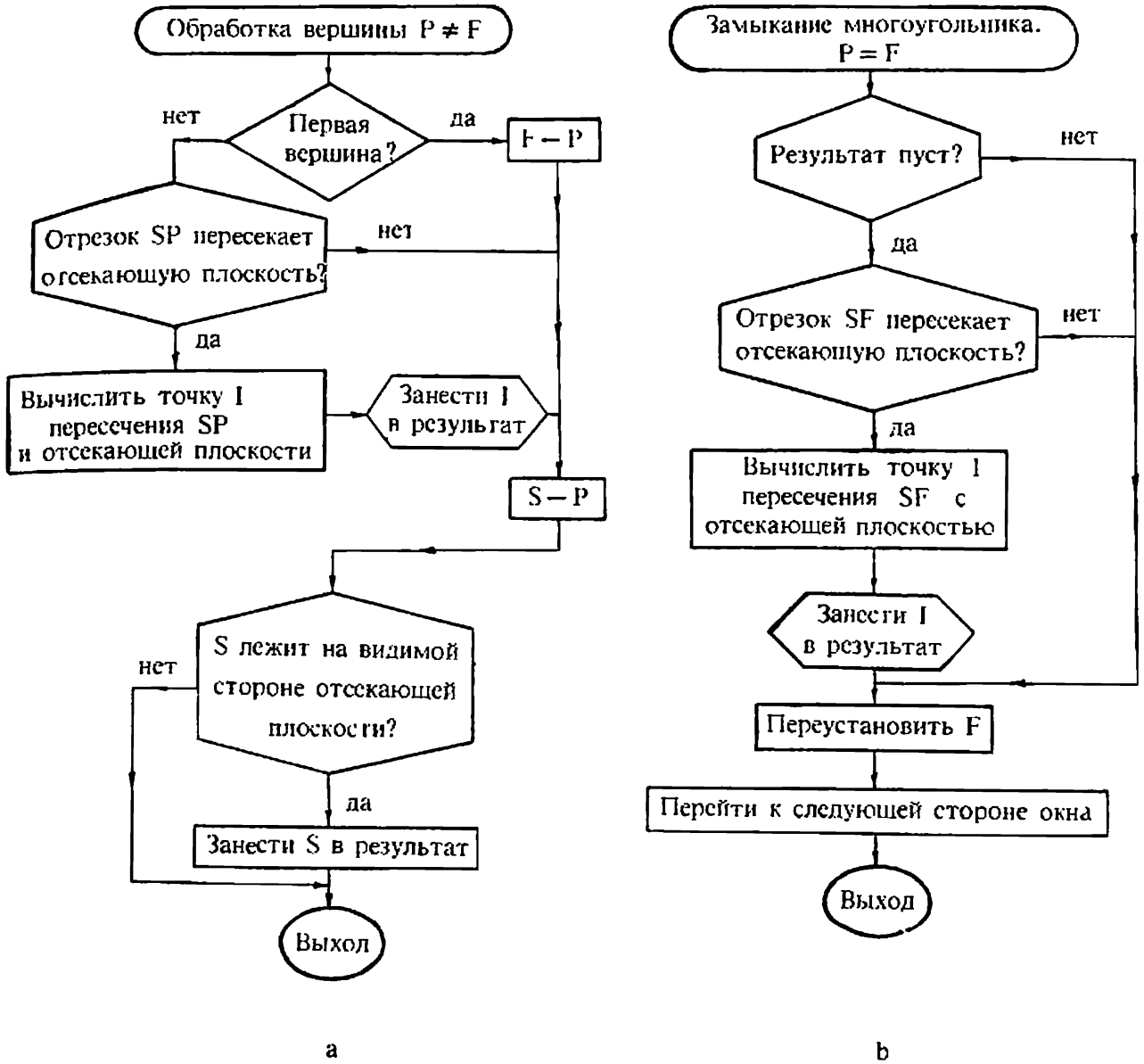



Рис. 3.32. Блок-схема алгоритма Сазерленда — Ходжмена.

рис. 3.32, а приведена процедура, применяемая к рядовой вершине, а процедура на рис. 3.32, b применима только к последней вершине многоугольника. Запись алгоритма, порождающего и запоминающего промежуточные многоугольники, приводится ниже.

Сазерленд и Ходжмен показали, как можно избежать порождения и запоминания вершин промежуточных многоугольников. Для этого вместо отсечения каждого ребра (вершины) многоугольника одной плоскостью, ограничивающей окно, надо отсекать каждое такое ребро (вершину) последовательно всеми границами окна. После отсечения очередного ребра (вершины) многоугольника по одной из границ окна, алгоритм рекурсивно обращается к самому себе, чтобы отсечь результат предыдущего обращения по следующей границе окна. Это свойство делает данный алгоритм более удобным для аппаратной реализации.

***Алгоритм Сазерленда — Ходжмена для отсечения многоугольника***

P — *массив вершин исходного многоугольника*
Q — *массив вершин результирующего многоугольника*
W — *массив вершин отсекающего окна. Первая вершина повторяется в конце массива*
$N_P$ — *число вершин исходного многоугольника*
$N_Q$ — *число вершин результирующего многоугольника*
$N_W$ — *число вершин окна плюс единица*
*вершины всех многоугольников перечисляются по часовой стрелке*

```
    для каждой стороны окна выполнить:
    for i = 1 to N_W - 1
        установить счетчик вершин результата и обнулить
        результат
        N_Q = 0
        Q = 0
        отсечь каждое ребро многоугольника по данной стороне
        окна
        for j = 1 to N_P
            особо обработать первую вершину многоугольника
            if j < > 1 then 1
            запомнить первую вершину
            F = P_j
            go to 2
            проверить факт пересечения ребром многоугольника
            стороны окна
1           call Факт—сеч(S, P_j, W_i, W_i+1; Признак)
            if Признак = нет then 2
            если ребро пересекает сторону окна, вычислить точ-
            ку пересечения
            call Пересечение(S, P_j, W_i, W_i+1; Тсечения)
            занести точку пересечения в результат
            call Выход(Тсечения, N_Q, Q)
            изменить начальную точку ребра многоугольника
2           S = P_j
            проверить видимость конечной точки (теперь
            это S) ребра многоугольника
            call Видимость(S, W_i, W_i+1; Свидимость)
            if Свидимость < 0 then 3
            если точка видима, то занести ее в результат
            call Выход(S, N_Q, Q)
```

```
    3   next j
        обработать замыкающее ребро многоугольника
        если результат пуст  то перейти к следующей стороне
        окна
        if N_Q = 0 then 4[1)]
        проверить факт пересечения последним ребром много-
        угольника стороны окна
        call Факт—сеч(S, F, W_i, W_i+1; Признак)
        if Признак = нет then 4
        факт пересечения установлен; вычислить точку пересе-
        чения
        call Пересечение(S, F, W_i, W_i+1; Тсечения)
        вывести точку пересечения в результат
        call Выход(Тсечения, N_Q, Q)
        Теперь многоугольник отсечен стороной W_i W_i+1 окна
        работа алгоритма возобновляется с результатом отсе-
        чения
    4   P = Q
        N_P = Q
5   next i
    finish
```

*подпрограмма определения факта пересечения ребра многоугольника со стороной окна*

```
subroutine Факт-сеч(Начало, Конец, W1, W2; Признак)
    определить видимость начальной точки ребра многоугольни-
    ка
    call Видимость(Начало, W1, W2; Твидимость)
    Твидимость1 = Твидимость
    определить видимость конечной точки ребра многоугольника
    call Видимость(Конец, W1, W2; Твидимость)
    Твидимость2 = Твидимость
    считается, что ребро многоугольника, которое начинается
    или заканчивается на стороне окна, не пересекается с ней.
    Эта точка должна быть занесена в результат ранее
    if Твидимость1 < 0 and Твидимость2 > 0 or
    Твидимость1 > 0 and Твидимость2 < 0 then
```

---

[1)] В этом случае следует вообще выйти из алгоритма. — *Прим. перев.*

```
            Признак = да
    else
            Признак = нет
    end if
    return
```

*подпрограмма определения видимости точки*

**subroutine** Видимость(Точка, P1, P2; Твидимость)
*видимость* Точка *следует определить относительно стороны* $P_1P_2$

$$\text{Твидимость} \begin{cases} < 0, \textit{ если} \text{ Точка } \textit{невидима} \\ = 0, \textit{ если} \text{ Точка } \textit{лежит на стороне } P_1P_2 \\ > 0, \textit{ если} \text{ Точка } \textit{видима} \end{cases}$$

*в этой подпрограмме используется вычисление векторного произведения*

Sign — *функция, принимающая значения* $-1$, 0, 1 *в зависимости от того, будет ли знак ее аргумента отрицателен, равен нулю или положителен*

```
Раб1 = (Точка x – P1x) * (P2y – P1y)
Раб2 = (Точка y – P1y) * (P2x – P1x)
Раб3 = Раб1 – Раб2
Твидимость = Sign(Раб3)
return
```

*подпрограмма вычисления точки пересечения двух отрезков*

**subroutine** Пересечение(P1, P2, W1, W2; Тсечения)
*подпрограмма использует параметрическое описание отрезков*
*отрезки* $P_1P_2$ и $W_1W_2$ *считаются двумерными*
*матричное уравнение с неизвестными значениями параметров получается путем приравнивания компонент x и y у двух параметрических описаний отрезков*

Коэф — *матрица* $2 \times 2$, *содержащая значения коэффициентов, уравнения отрезка*

Параметр — *матрица* $2 \times 1$, *содержащая значения параметров описания отрезков*

Параметр(1,1) — *значение параметра описания отрезка P*

Прав — *матрица* $2 \times 1$, *содержащая значения правых частей уравнений*

Обрат — *функция, обращающая матрицу*

Умнож — *функция умножения матриц*

*сформировать матрицу* Коэф
Коэф(1,1) = P2x − P1x
Коэф(1,2) = W1x − W2x
Коэф(2,1) = P2y − P1y
Коэф(2,2) = W1y − W2y
*сформировать матрицу правых частей*
Прав(1,1) = W1x − P1x
Прав(2,1) = W1y − P1y
*обратить матрицу коэффициентов*
*нет необходимости проверять здесь невырожденность этой матрицы, поскольку факт пересечения уже установлен*
Коэф = Обрат(Коэф)
*вычислить значения параметров в точке пересечения*
Параметр = (Коэф) Умнож (Прав)
*вычислить координаты точки пересечения*
Тсечения = P1 + (P2 − P1) * Параметр(1,1)
**return**

*подпрограмма формирования результирующего многоугольника*

**subroutine** Выход(Вершина, $N_Q$, Q)

Вершина *содержит точку, заносимую в результат*
*увеличить число вершин результата и добавить точку в* Q
$N_Q = N_Q + 1$
$Q(N_Q)$ = Вершина
**return**

Пример 3.25 продемонстрирует работу алгоритма Сазерленда — Ходжмена. Будет показана также одна особенность этого алгоритма: возникновение ложных границ. Во многих приложениях возникновение таких ложных границ несущественно, например при растровой развертке сплошных областей. Однако в ряде других приложений, например в некоторых алгоритмах удаления невидимых поверхностей, требуется устранение таких границ. Это можно проделать путем сортировки вершин так, как предложено в работе [3-7].

**Пример 3.25.** Отсечение многоугольника алгоритмом Сазерленда — Ходжмена
Рассмотрим многоугольник, вершины которого приведены в табл. 3.15. На рис. 3.33

**Таблица 3.15.**

| Вершины | Исходный многоугольник | Отсечение левой стороной | Отсечение верхней стороной | Отсечение правой стороной | Результат |
|---|---|---|---|---|---|
| $P_1$ | ( 1/2, −3/2) | (1/2, −3/2) | (1/2, −3/2) | (1/2, −3/2) | (−1, −1) |
| $P_2$ | ( −2, −3/2) | (−1, −3/2) | (−1, −3/2) | (−1 −3/2) | (−1, 1) |
| $P_3$ | ( −2, 2) | (−1, 2) | (−1, 1) | (−1, 1) | ( 1, 1) |
| $P_4$ | ( 3/2, 2) | (3/2, 2) | (3/2, 1) | ( 1, 1) | ( 1, 0) |
| $P_5$ | ( 3/2, 0) | (3/2, 0) | (3/2, 0) | ( 1, 0) | (1/2, 0) |
| $P_6$ | ( 1/2, 0) | (1/2, 0) | (1/2, 0) | (1/2, 0) | (1/2, 1) |
| $P_7$ | ( 1/2, 3/2) | (1/2, 3/2) | (1/2, 1) | (1/2, 1) | (−1, 1) |
| $P_8$ | (−3/2, 3/2) | (−1, 3/2) | (−1, 1) | (−1, 1) | (−1, 0) |
| $P_9$ | (−3/2, 1/2) | (−1, 0) | (−1, 0) | (−1, 0) | ( 0, −1) |

показано, как этот многоугольник отсекается квадратным окном, ограниченным плоскостями $x_Л = -1$, $x_П = 1$, $y_Н = -1$, $y_В = 1$. В качестве частного примера рассмотрим отсечение ребра многоугольника между вершинами $P_1$ и $P_2$ левым краем окна. При обходе краев окна по часовой стрелке внутренние или видимые их стороны остаются справа. Используя метод подстановки, описанный выше (в при-

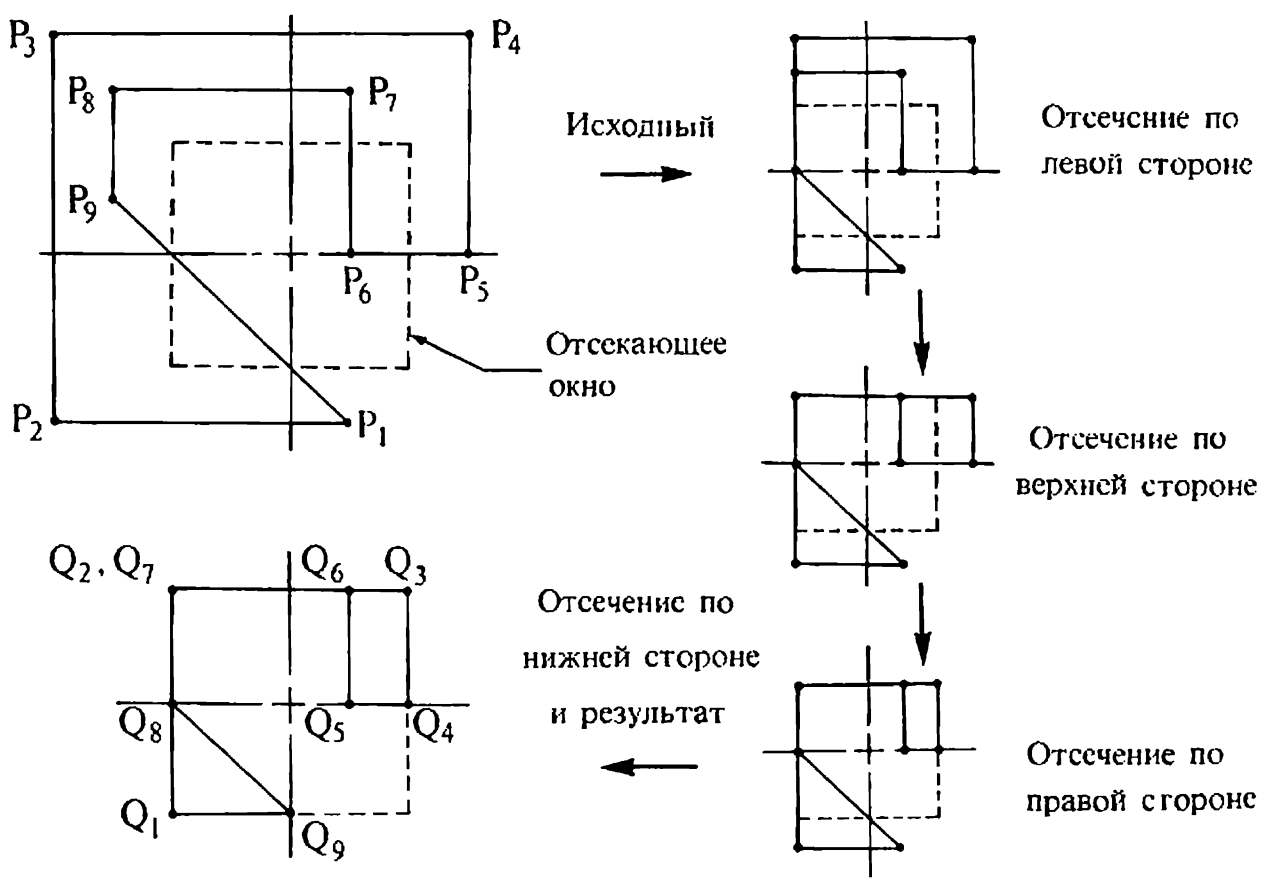



Рис. 3.33. Результаты отсечений примера 3.24.

мере 3.23), имеем здесь пробную функцию $x - w$, равную:

$$x - w = x - (-1) = x + 1.$$

Для точки $P_1(1/2, -3/2)$:

$$x + 1 = 1/2 + 1 > 0.$$

Поэтому $P_2$ лежит справа от края окна и видима.

Для точки $P(-2, -3\ 2)$:

$$x_2 + 1 = -2 + 1 < 0.$$

Поэтому $P_2$ невидима. Ребро $P_1P_2$ пересекает отсекающую плоскость. Значит, нужно вычислять точку пересечения. Воспользовавшись методом параметрического задания отрезка (см. пример 3.24), получаем $x = -1$, $y = -3/2$.

Результаты отсечений показаны на рис. 3.33. Особенно интересен последний этап отсечения, т. е. отсечение нижним краем окна. Точка $P_1$ еще не отсечена до этого этапа. Следовательно, в списке вершин промежуточного многоугольника сохраняется тот же порядок обхода, что и в списке вершин исходного многоугольника. Однако после отсечения нижним краем окна $P_1$ устраняется. Теперь список вершин начинается с промежуточной вершины, соответствующей $P_2$. Последней в списке вершин результирующего многоугольника будет точка пересечения ребра $P_9P_1$ с нижним краем окна.

Заметим, что в верхнем левом углу отсекающего окна на рис. 3.33 показаны четыре ложных ребра и три границы результирующего многоугольника.

Так, версия алгоритма Сазерленда — Ходжмена, которая описана выше, предназначена для отсечения по двумерному окну. Однако этот алгоритм фактически обладает большей общностью. Любой плоский или неплоский многоугольник можно отсечь по выпуклому объему путем вычисления его пересечений с трехмерными отсекающими плоскостями при помощи алгоритма Кируса — Бека. Алгоритм отсечения Сазерленда — Ходжмена можно использовать и для разрезания невыпуклых многоугольников (см. разд. 3.8 и работу [3-7]).

Лианг и Барский предложили [3-8] новый алгоритм отсечения многоугольников. Этот алгоритм в той форме, в какой он был описан в оригинале, оптимально приспособлен для отсечения прямоугольными окнами, однако его можно обобщить и на случай произвольных выпуклых окон. Данный алгоритм основан на идеях, заимствованных из алгоритма отсечения двух- и трехмерных отрезков, который принадлежит тем же авторам [3-5]. Эксперименты показали, что для прямоугольных окон этот оптимизированный алгоритм работает вдвое быстрее, чем алгоритм Сазерленда — Ходжмена.

## 3.17. НЕВЫПУКЛЫЕ ОТСЕКАЮЩИЕ ОБЛАСТИ — АЛГОРИТМ ВЕЙЛЕРА — АЗЕРТОНА

Для использования вышеизложенных алгоритмов отсечения требуются выпуклые отсекающие области. Однако во многих приложениях, например при удалении невидимых поверхностей, необходимо уметь отсекать и по невыпуклым областям. Этой потребности отвечает более мощный, но и более сложный алгоритм, предложенный Вейлером и Азертоном [3-9]. Данный алгоритм позволяет производить отсечение невыпуклого многоугольника с внутренними отверстиями по другому невыпуклому многоугольнику, который также имеет внутренние отверстия. Будем называть многоугольник, который отсекается, обрабатываемым многоугольником, а многоугольник, по которому производится отсечение, — отсекающим многоугольником (отсекателем). Новые границы, образуемые в результате отсечения обрабатываемого многоугольника отсекающим, совпадают с участками границ отсекателя. Никаких иных новых границ не возникает. Следовательно, число многоугольников в результате минимально.

Как обрабатываемый, так и отсекающий многоугольники описываются в алгоритме циклическими списками их вершин. Внешняя граница каждого из этих многоугольников обходится по часовой стрелке, а внутренние границы или отверстия — против часовой стрелки. Это условие означает, что при обходе вершин многоугольника в порядке их следования в соответствующем списке внутренняя его область будет расположена справа от границы. Границы обрабатываемого и отсекающего многоугольников могут пересекаться или не пересекаться между собой. Если они пересекаются, то точки пересечения образуют пары. Одно пересечение из пары возникает, когда ребро обрабатываемого многоугольника входит внутрь отсекающего многоугольника, а другое — когда оно выходит оттуда. Основная идея заключается в том, что алгоритм начинает с точки пересечения входного типа, затем он прослеживает внешнюю границу по часовой стрелке до тех пор, пока не обнаруживается еще одно ее пересечение с отсекающим многоугольником. В точке последнего пересечения производится поворот направо и далее прослеживается внешняя граница отсекателя по часовой стрелке до тех пор, пока не обнаруживается ее пересечение с обрабатываемым многоугольником. И вновь в точке последнего пересечения производится поворот направо и далее прослеживается граница обрабатываемого многоугольника. Этот процесс продолжает-

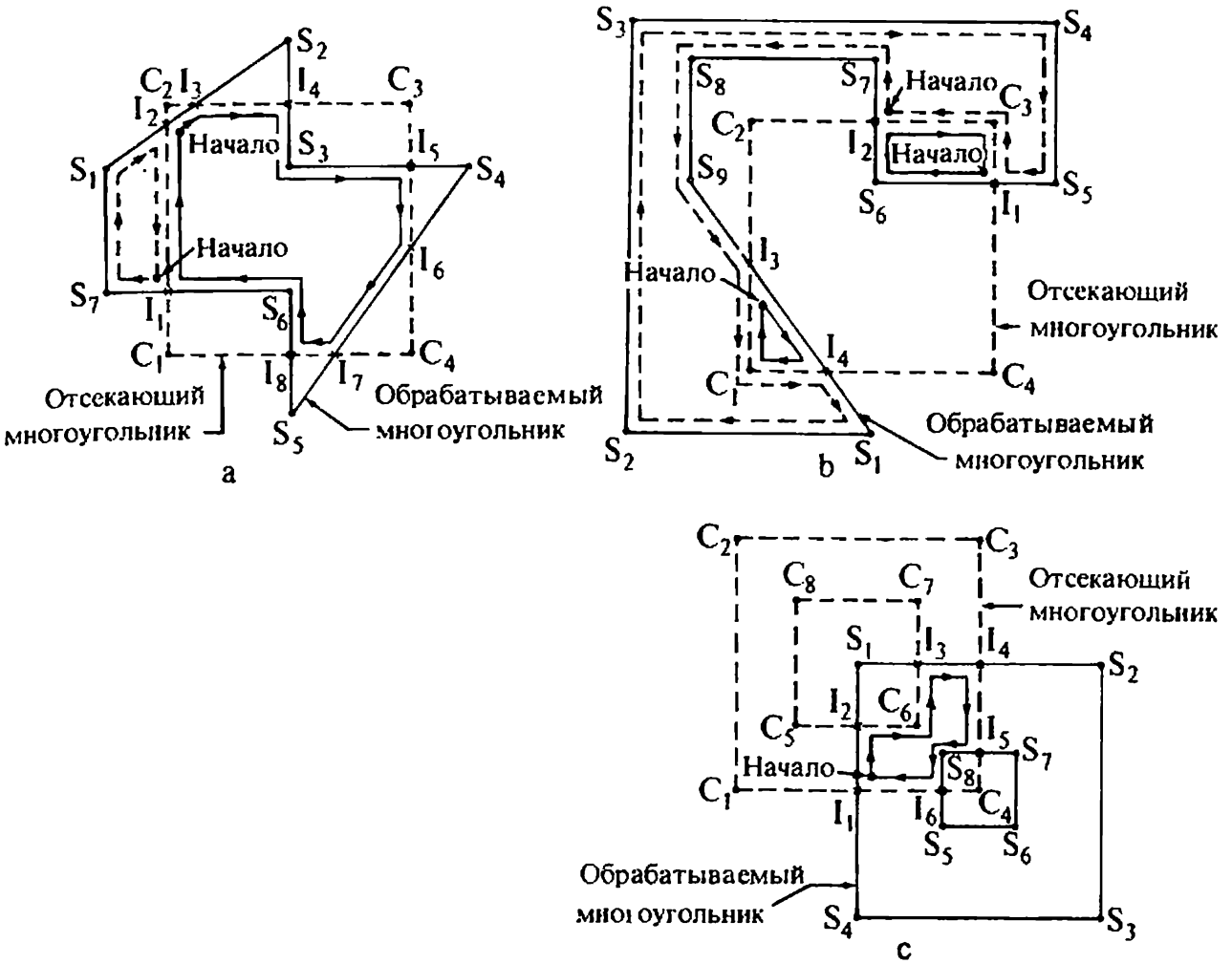



Рис. 3.34. Отсечение Вейлера — Азертона.

ся до тех пор, пока не встретится начальная вершина. Внутренние границы обрабатываемого многоугольника обходятся против часовой стрелки (см. рис. 3.34).

Более формальная запись этого алгоритма имеет следующий вид:

Вычислить все пересечения обрабатываемого и отсекающего многоугольников.

Добавить все эти пересечения к спискам вершин обрабатываемого и отсекающего многоугольников. Пометить все точки пересечения и установить двусторонние связи между списками вершин обрабатываемого и отсекающего многоугольников для каждой такой точки.

Обработать непересекающиеся границы многоугольников.

Установить два списка принадлежности, один — для границ, лежащих внутри отсекающего многоугольника, и другой — для границ, лежащих вне отсекателя. Проигнорировать такие грани-

цы отсекателя, которые лежат вне обрабатывающего многоугольника. Границы отсекателя, попавшие внутрь обрабатывающего многоугольника, образуют в нем дыры. Следовательно, копии границ отсекающего многоугольника должны попасть в оба списка принадлежности: внутренний и внешний. Занести эти границы в соответствующие списки принадлежности.

Создать два списка вершин, являющихся точками пересечения.

Один — список входов — содержит только пересечения, образованные ребрами обрабатываемого многоугольника, которые входят внутрь отсекателя. Другой — список выходов — содержит только пересечения, образованные ребрами обрабатываемого многоугольника, которые выходят изнутри отсекателя. Типы точек пересечения будут чередоваться при обходе границы. Поэтому достаточно определить тип только у одной из каждой пары точек пересечения.

Реализовать отсечение.

Поиск многоугольников, лежащих внутри отсекателя, ведется с помощью следующей процедуры.

Взять точку пересечения из списка входов. Если этот список пуст, то поиск завершен.

Просматривать список вершин обрабатываемого многоугольника до тех пор, пока не обнаружится пересечение. Скопировать вершины из списка вершин обрабатываемого многоугольника вплоть до этого пересечения в список внутренней принадлежности.

Используя двустороннюю связь, перейти к списку вершин отсекающего многоугольника.

Просматривать список вершин отсекателя до тех пор, пока не обнаружится пересечение. Скопировать вершины из списка вершин отсекателя вплоть до этого пересечения в список внутренней принадлежности.

Вернуться назад в список вершин обрабатываемого многоугольника.

Повторять эти действия до тех пор, пока не будет достигнута начальная вершина. В этот момент новый внутренний многоугольник замыкается.

Поиск многоугольников, лежащих вне отсекателя, ведется с помощью такой же процедуры, за исключением того что на-

чальная точка пересечения берется из списка выходов, а вершины из списка вершин отсекателя просматриваются в *обратном* порядке. Списки вершин многоугольников при этом копируются в список внешней принадлежности.

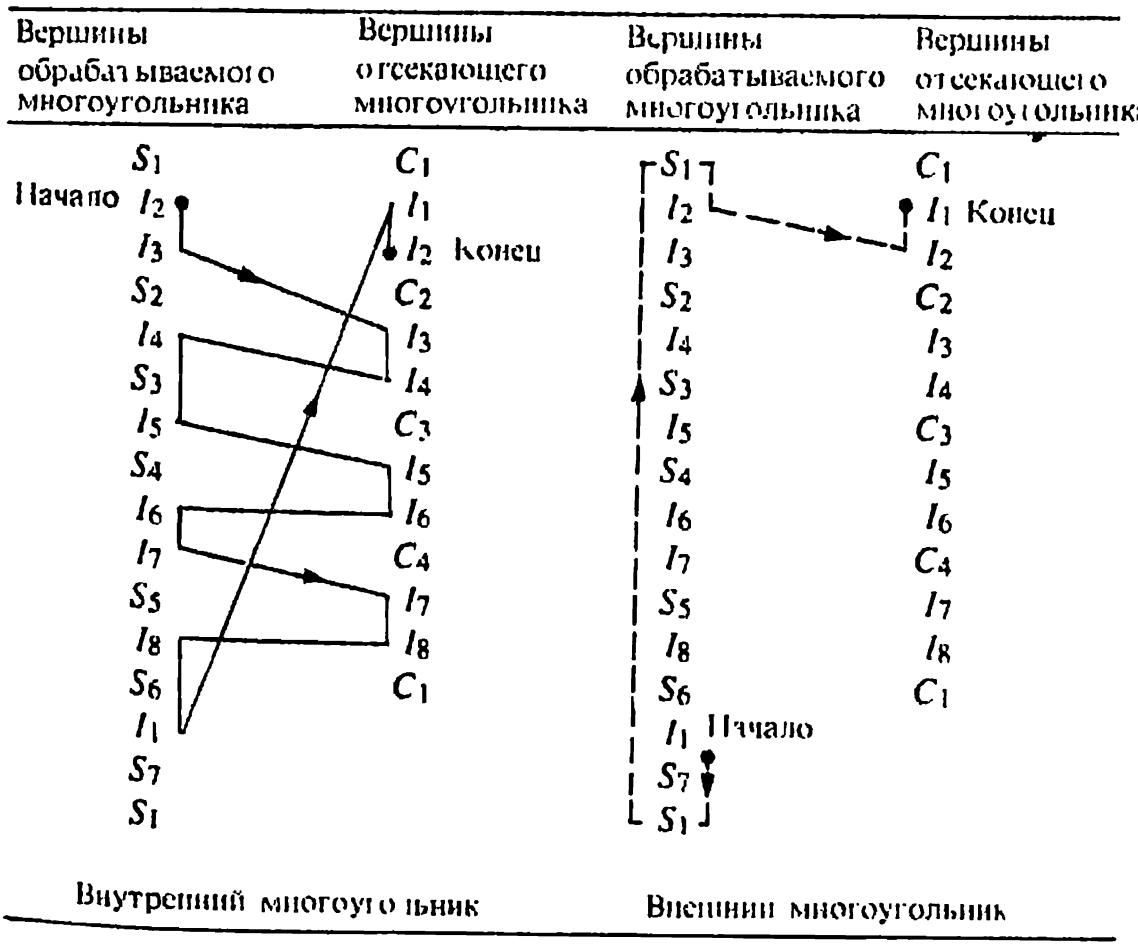
Связать все отверстия, т. е. внутренние границы, с соответствующими им внешними границами. Поскольку внешние границы ориентированы по часовой стрелке, а внутренние границы — против часовой стрелки, эту операцию удобно выполнить путем проверки ориентации границ.

Процесс завершен.

Ряд примеров послужит для более полной иллюстрации работы этого алгоритма.

**Пример 3.26.** Отсечение с помощью алгоритма Вейлера — Азертона. Случай простого многоугольника

Рассмотрим многоугольник, показанный на рис. 3.34, а, который отсекается по квадрату. На рисунке изображены также точки пересечения этих многоугольников, помеченные через $I_i$. Ниже приводятся списки вершин обрабатываемого и отсекающего многоугольников. В список входов занесены вершины $I_2$, $I_4$, $I_6$ и $I_8$, а в список выходов — вершины $I_1$, $I_3$, $I_5$, $I_7$.



Для формирования внутреннего многоугольника возьмем первую точку пересечения из списка входов — $I_2$. Описанный алгоритм дает результаты, показанные сплошной линией со стрелками на рис. 3.34, а, а также в таблице. Результирующий внутренний многоугольник таков:

$$I_2 I_3 I_4 S_3 I_5 I_6 I_7 I_8 S_6 I_1 I_2.$$

Если начинать с других точек пересечения из списка входов, т. е. с точек $I_4$, $I_6$ и $I_8$, то получится тот же самый результат.

Для формирования внешних многоугольников возьмем первую точку пересечения из списка выходов — $I_1$. Описанный выше алгоритм дает результаты, показанные штриховой линией со стрелками на рис. 3.34, а и в таблице. Заметим, что вершины из списка вершин отсекателя проходятся в обратном порядке от $I_2$ к $I_1$. Получающийся внешний многоугольник таков:

$$I_1 S_7 S_1 I_2 I_1.$$

Если же начинать построение с других точек пересечения — $I_3$, $I_5$ или $I_7$ — из списка выходов, то получатся другие внешние многоугольники, соответственно:

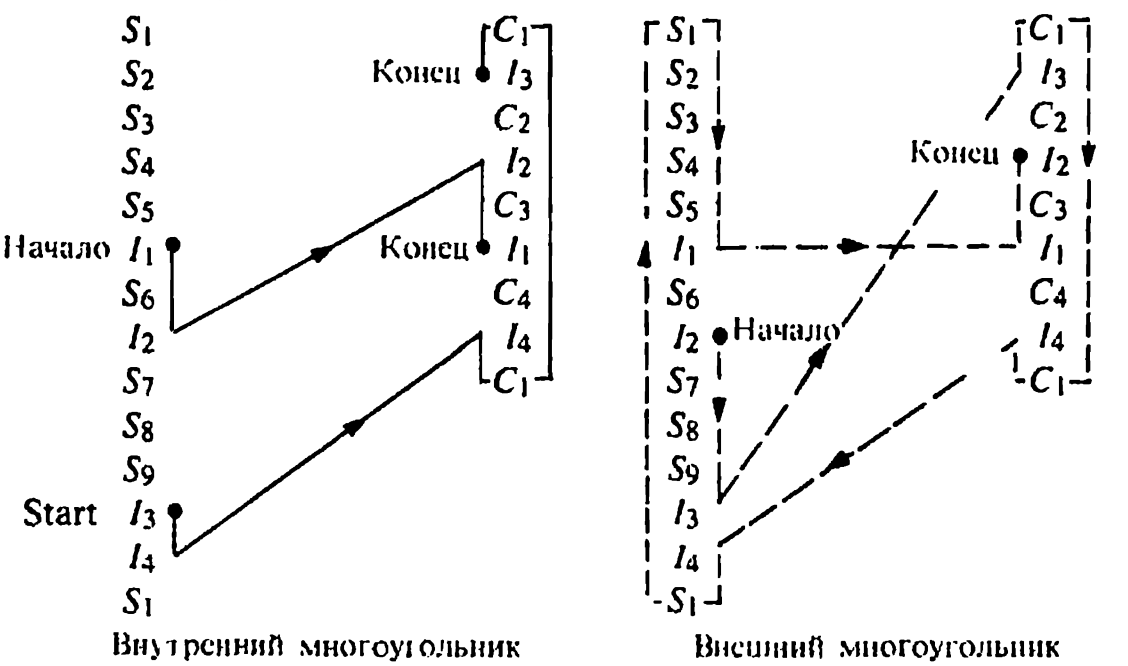
$$I_3 S_2 I_4 I_3,\ I_5 S_4 I_6 I_5 \text{ или } I_7 S_5 I_8 I_7.$$

В следующем примере рассмотрен более сложный обрабатываемый многоугольник, который частично охватывает отсекатель.

**Пример 3.27.** Отсечение с помощью алгоритма Вейлера — Азертона. Случай охвата обрабатываемым многоугольником отсекающего

Обрабатываемый и отсекающий многоугольники, а также их пересечения показаны на рис. 3.34, b. Точки пересечения $I_1$ и $I_3$ попадают в список входов, а $I_2$ и $I_4$ — в список выходов. Списки вершин обрабатываемого и отсекающего многоугольников с учетом пересечений задаются следующей таблицей:

| Вершины обрабатываемого многоугольника | Вершины отсекающего многоугольника | Вершины обрабатываемого многоугольника | Вершины отсекающего многоугольника |
|---|---|---|---|
| $S_1$ | $C_1$ | $S_1$ | $C_1$ |
| $S_2$ | Конец $I_3$ | $S_2$ | $I_3$ |
| $S_3$ | $C_2$ | $S_3$ | $C_2$ |
| $S_4$ | $I_2$ | $S_4$ | Конец $I_2$ |
| $S_5$ | $C_3$ | $S_5$ | $C_3$ |
| Начало $I_1$ | Конец $I_1$ | $I_1$ | $I_1$ |
| $S_6$ | $C_4$ | $S_6$ | $C_4$ |
| $I_2$ | $I_4$ | $I_2$ Начало | $I_4$ |
| $S_7$ | $C_1$ | $S_7$ | $C_1$ |
| $S_8$ | | $S_8$ | |
| $S_9$ | | $S_9$ | |
| Start $I_3$ | | $I_3$ | |
| $I_4$ | | $I_4$ | |
| $S_1$ | | $S_1$ | |



Внутренний многоугольник Внешний многоугольник

Чтобы получить внутренние многоугольники, берем сначала из списка входов $I_1$, а затем $I_3$. Результаты показаны сплошными линиями со стрелками на рис. 3.34, b и в таблице. Результирующие внутренние многоугольники таковы:

$$I_1 S_6 I_2 C_3 I_1 \text{ и } I_3 I_4 C_1 I_3.$$

Заметим, что в результате получилось два внутренних многоугольника.

Если начать построение с точки $I_2$ из списка выходов, то получится следующий внешний многоугольник:

$$I_2 S_7 S_8 S_9 I_3 C_1 I_1 S_1 S_2 S_3 S_4 S_5 I_1 C_3 I_2.$$

Такой же результат получится, если начать построение с точки $I_4$. Этот результат показан штриховой линией на рис. 3.34, b и в таблице. Вновь заметим, что вершины отсекателя в списке внешнего многоугольника следуют в обратном порядке.

В последнем примере показано, как невыпуклый многоугольник с дырой отсекается относительно невыпуклого окна, также имеющего дыру.

**Пример 3.28.** Отсечение с помощью алгоритма Вейлера — Азертона. Случай многоугольников с дырами

Обрабатываемый и отсекающий многоугольники, а также их пересечения показаны на рис. 3.34, c. Точки пересечения $I_1$, $I_3$ и $I_5$ помещаются в список входов, а $I_2$, $I_4$ и $I_6$ — в список выходов. Списки вершин обрабатываемого и отсекающего многоугольников приведены в следующей таблице:

| Вершины обрабатываемого многоугольника | Вершины отсекающего многоугольника | Вершины обрабатываемого многоугольника | Вершины отсекающего многоугольника |
|---|---|---|---|
| $S_1$ | $C_1$ | $S_1$ | $C_1$ |
| $I_3$ | $C_2$ | $I_3$ | $C_2$ |
| $I_4$ | $C_3$ | $I_4$ | $C_3$ |
| $S_2$ | $I_4$ Внешняя граница | $S_2$ | $I_4$ |
| $S_3$ | $I_5$ | $S_3$ | $I_5$ |
| $S_4$ | $C_4$ | $S_4$ | $C_4$ |
| Начало $I_1$ | $I_6$ | $I_1$ | $I_6$ |
| $I_2$ | $I_1$ Конец | $I_2$ Начало | $I_1$ |
| $S_1$ | $C_1$ | $S_1$ | $C_1$ |
| $S_5$ | $C_5$ | $S_5$ | $C_5$ |
| $S_6$ | $I_2$ | $S_6$ | $I_2$ Конец |
| $S_7$ | $C_6$ Граница дыры | $S_7$ | $C_6$ |
| $I_5$ | $I_3$ | $I_5$ | $I_3$ |
| $S_8$ | $C_7$ | $S_8$ | $C_7$ |
| $I_6$ | $C_8$ | $I_6$ | $C_8$ |
| $S_5$ | $C_5$ | $S_5$ | $C_5$ |
| Внутренний многоугольник | | Внешний многоугольник | |

Заметим, что для внутренних границ, т. е. отверстий, вершины обходятся против часовой стрелки. Списки вершин внешних и внутренних границ образуют независимые контуры.

Если начать с точки $I_1$ из списка входов, то алгоритм построит следующий внутренний многоугольник, показанный на рис. 3.34, с и в таблице сплошной линией со стрелками:

$$I_1 I_2 C_6 I_3 I_4 I_5 S_8 I_6 I_1$$

Если начать с точек $I_3$ или $I_5$ из списка входов, то результат будет таким же.

Если же начать с точки $I_2$, взятой из списка выходов, то получим внешний многоугольник:

$$I_2 S_1 I_3 C_6 I_2$$

Заметим, что список вершин обрабатываемого многоугольника содержит две разные границы, внутреннюю и внешнюю, каждая из которых образует свой замкнутый контур. Поэтому переход от $S_1$ — нижней записи в списке вершин внешней границы, произойдет к $S_1$ — верхней записи в том же списке, а не к записи $S_5$ из списка границы отверстия. Переход от внешней границы к внутренней всегда происходит путем перехода от обрабатываемого многоугольника к отсекающему, или наоборот, что и показано штриховой линией, соединяющей списки вершин этих фигур в вышеприведенной таблице. Аналогично, если начать с точек $I_4$ и $I_6$ из списка выходов, то придем к внешнему многоугольнику:

$$I_4 S_2 S_3 S_4 I_1 I_6 S_5 S_6 S_7 I_5 I_4$$

Для корректности работы алгоритма Вейлера — Азертона нужно аккуратно классифицировать и вычислять пересечения. Касания,

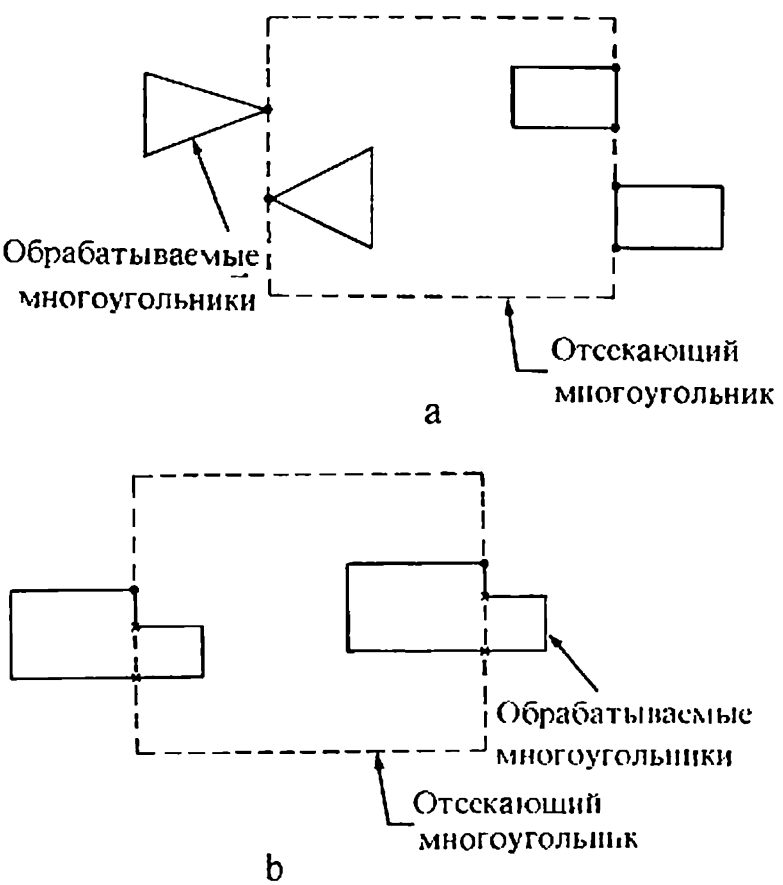



Рис. 3.35. Подробности реализации алгоритма Вейлера — Азертона.

т. е. ситуации, при которых вершина или ребро обрабатываемого многоугольника инцидентна или совпадает со стороной отсекателя, не считаются пересечениями. Примеры таких ситуаций показаны на рис. 3.35, а. Аналогично, чтобы избежать возникновения висячих ребер, нужно корректно классифицировать и такие пересечения, которые показаны на рис. 3.35, b. В частности, точки, помеченные крестиками на рис. 3.35, b, считаются пересечениями, а помеченные точкой — таковыми не считаются. Дополнительные подробности реализации этого алгоритма даны в работах [3-10, 3-11].

## 3.18. ОТСЕЧЕНИЕ ЛИТЕР

Литеры или текст можно генерировать программно, микропрограммно или аппаратно. Они могут состоять из отдельных отрезков или штрихов или быть образованными точечной матрицей. Штриховые литеры, которые сгенерированы программно, можно обрабатывать как любые другие отрезки, т. е. их можно поворачивать, переносить, масштабировать и отсекать по любым окнам с произвольной ориентацией, используя изложенные выше алгоритмы. На рис. 3.36 показан типичный пример такого отсечения.

Программно сгенерированные литеры в форме точечной матрицы можно обрабатывать точно так же. Однако эта задача более трудоемкая. В частности, если прямоугольная объемлющая оболочка литеры отсекается по окну произвольной формы, то нужно проверить, будет ли каждый пиксел из маски символа находиться

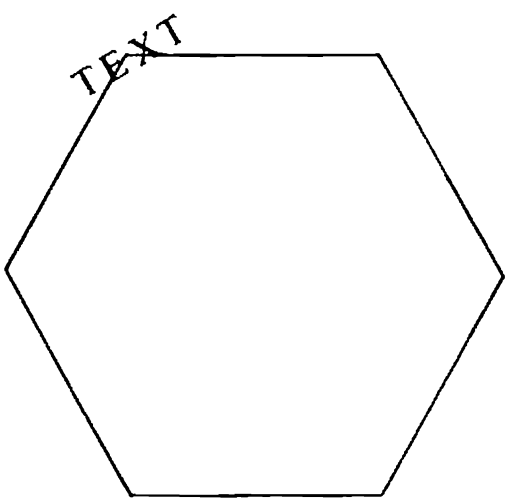



Рис. 3.36. Отсечение программно сгенерированных штриховых литер.

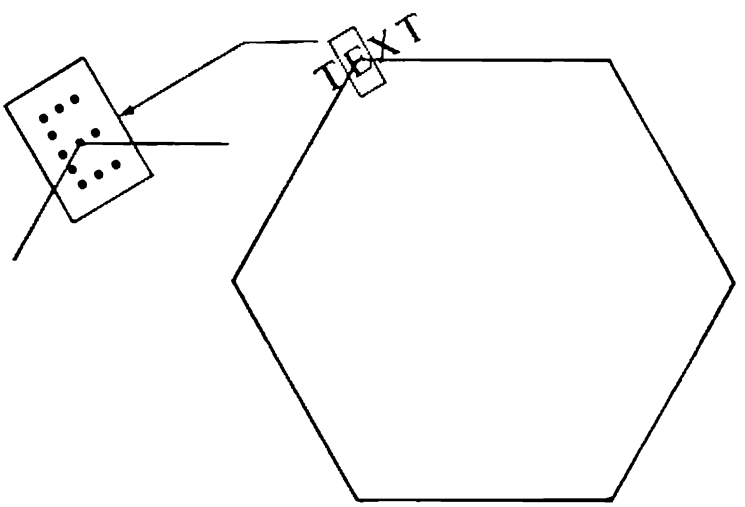



Рис. 3.37. Отсечение программно сгенерированных точечных литер.

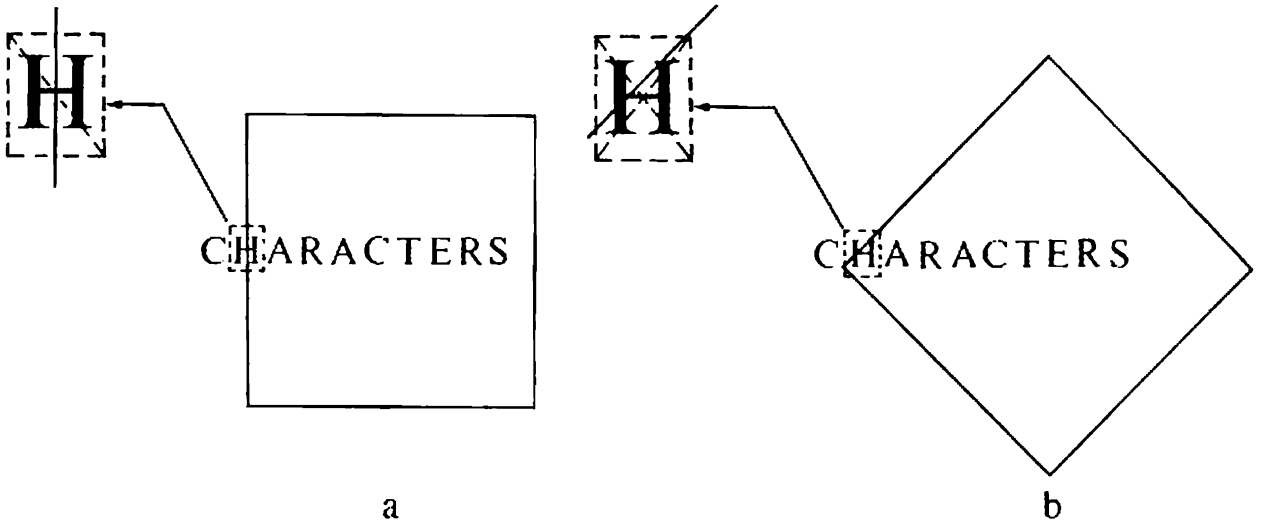



Рис. 3.38. Отсечение аппаратно сгенерированных литер.

внутри или вне отсеченной части оболочки. Если внутри, то этот пиксел активируется. В противном случае никаких действий не производится. Это показано на рис. 3.37.

На отсечение аппаратно сгенерированных литер накладывается больше ограничений. Обычно любая не полностью видимая литера удаляется. Это можно проделать путем отсечения прямоугольной оболочки литеры окном. Если вся эта оболочка лежит внутри окна, то литера изображается, в противном случае она не изображается. Если прямоугольная оболочка литеры ориентирована так же, как и прямоугольное окно, то тест видимости можно провести только для одной из диагоналей этой оболочки. Рассмотрим рис. 3.38, a. Для окон нестандартной формы или в тех случаях, когда прямоугольная оболочка литеры ориентирована иначе, чем окно, тесты видимости следует провести для обеих диагоналей оболочки, как показано на рис. 3.38, b.

При микропрограммной генерации литер средства их отсечения могут оказаться как очень ограниченными, так и очень разнообразными. Эти возможности зависят от особенностей алгоритма отсечения, который тоже реализуется микропрограммно.

## 3.19. ЛИТЕРАТУРА

3-1 Clark, James, H., "The Geometry Engine: A VLSI Geometry System for Graphics," *Computer Graphics*, Vol. 16, pp. 127–133, 1982 (Proc. SIGGRAPH 82).

3-2 Sproull, Robert, F., and Sutherland, Ivan, E., "A Clipping Divider," 1968 Fall Joint Computer Conference, Thompson Books, Washington, D.C., pp. 765–775, 1968.

3-3 Newman, William, M., and Sproull, Robert, F., *Principles of Interactive Computer Graphics*, 2d ed., McGraw-Hill Book Company, New York, 1979.
3-4 Cyrus, M., and Beck, J., "Generalized Two- and Three-Dimensional Clipping," *Computers & Graphics*, Vol. 3, pp. 23–28, 1978.
3-5 Liang, You-Dong and Barsky, Brian, "A New Concept and Method for Line Clipping," ACM *Trans. on Graphics*, Vol. 3, pp. 1–22, 1984.
3-6 Blinn, J.F., and Newell, M. E., "Clipping Using Homogeneous Coordinates," *Computer Graphics*, Vol. 12, pp. 245–251, 1978 (Proc. SIGGRAPH 78).
3-7 Sutherland, Ivan, E., and Hodgman Gary, W., "Reentrant Polygon Clipping," *CACM*, Vol. 17, pp. 32–42, 1974.
3-8 Liang, You-Dong, and Barsky, Brian, "An Analysis and Algorithm for Polygon Clipping," *CACM*, Vol. 26, pp. 868–877, 1983.
3-9 Weiler, Kevin, and Atherton, Peter, "Hidden Surface Removal Using Polygon Area Sorting," *Computer Graphics*, Vol. 11, pp. 214–222, 1977 (Proc. SIGGRAPH 77).
3-10 Weiler, Kevin, "Hidden Surface Removal Using Polygon Area Sorting," Masters Thesis, Program of Computer Graphics, Cornell University, January 1978.
3-11 Weiler, Kevin, "Polygon Comparison Using a Graph Representation," *Computer Graphics*, Vol. 14, pp. 10–18, 1980 (Proc. SIGGRAPH 80).

4 ⟫

# Удаление невидимых линий и поверхностей

Задача удаления невидимых линий и поверхностей является одной из наиболее сложных в машинной графике. Алгоритмы удаления невидимых линий и поверхностей служат для определения линий ребер, поверхностей или объемов, которые видимы или невидимы для наблюдателя, находящегося в заданной точке пространства.

## 4.1. ВВЕДЕНИЕ

Необходимость удаления невидимых линий, ребер, поверхностей или объемов проиллюстрирована рис. 4.1. На рис. 4.1, a приведен типичный каркасный чертеж куба. Каркасный чертеж представляет трехмерный объект в виде штрихового изображения его ребер. Рис. 4.1, a можно интерпретировать двояко: как вид куба сверху, слева или снизу, справа. Для этого достаточно прищуриться и перефокусировать глаза. Удаление тех линий или поверхностей, которые невидимы с соответствующей точки зрения, позволяют избавиться от неоднозначности. Результаты показаны на рис. 4.1, b и c.

Сложность задачи удаления невидимых линий и поверхностей привела к появлению большого числа различных способов ее решения. Многие из них ориентированы на специализированные приложения. Наилучшего решения общей задачи удаления невидимых линий и поверхностей не существует. Для моделирования процессов в реальном времени, например, для авиатренажеров, требуются быстрые алгоритмы, которые могут порождать результаты с частотой видеогенерации (30 кадр/с). Для машинной мультипликации, например, требуются алгоритмы, которые могут генерировать сложные реалистические изображения, в которых представле-

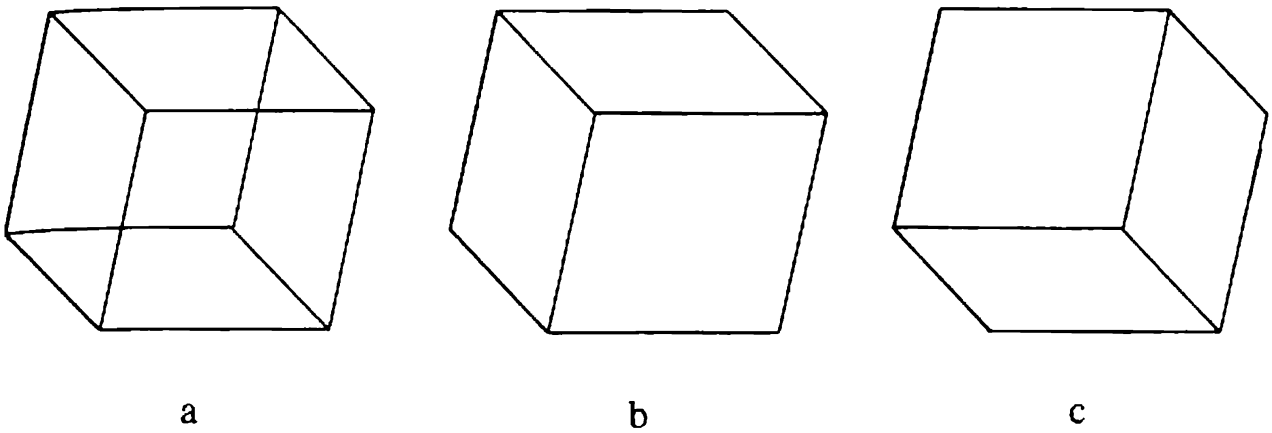



Рис. 4.1. Необходимость удаления невидимых линий.

ны тени, прозрачность и фактура, учитывающие эффекты отражения и преломления цвета в мельчайших оттенках. Подобные алгоритмы работают медленно, и зачастую на вычисления требуется несколько минут или даже часов. Строго говоря, учет эффектов прозрачности, фактуры, отражения и т. п. не входит в задачу удаления невидимых линий или поверхностей. Естественнее считать их частью процесса визуализации изображения. Процесс визуализации является интерпретацией или представлением изображения или сцены в реалистической манере. Такие эффекты подробно обсуждаются в гл. 5. Однако многие из этих эффектов встроены в алгоритмы удаления невидимых поверхностей и поэтому будут затронуты в данной главе. Существует тесная взаимосвязь между скоростью работы алгоритма и детальностью его результата. Ни один из алгоритмов не может достигнуть хороших оценок для этих двух показателей одновременно. По мере создания все более быстрых алгоритмов можно строить все более детальные изображения. Реальные задачи, однако, всегда будут требовать учета еще большего количества деталей.

Все алгоритмы удаления невидимых линий (поверхностей) включают в себя сортировку [4-1]. Порядок, в котором производится сортировка координат объектов, вообще говоря, не влияет на эффективность этих алгоритмов. Главная сортировка ведется по геометрическому расстоянию от тела, поверхности, ребра или точки до точки наблюдения. Основная идея, положенная в основу сортировки по расстоянию, заключается в том, что чем дальше расположен объект от точки наблюдения, тем больше вероятность, что он будет полностью или частично заслонен одним из объектов, более близких к точке наблюдения. После определения расстояний или приоритетов по глубине остается провести сортировку по горизонтали и по вертикали, чтобы выяснить, будет ли рассматриваемый

объект действительно заслонен объектом, расположенным ближе к точке наблюдения. Эффективность любого алгоритма удаления невидимых линий или поверхностей в большой мере зависит от эффективности процесса сортировки. Для повышения эффективности сортировки используется также когерентность сцены, т. е. тенденция неизменяемости характеристик сцены в малом. В растровой графике использование когерентности для улучшения результатов сортировки в алгоритмах удаления невидимых поверхностей приводит к алгоритмам, которые очень напоминают алгоритмы растровой развертки, обсуждавшиеся ранее в гл. 2.

Алгоритмы удаления невидимых линий или поверхностей можно классифицировать по способу выбора системы координат или пространства, в котором они работают [4-1]. Алгоритмы, работающие в объектном пространстве, имеют дело с физической системой координат, в которой описаны эти объекты. При этом получаются весьма точные результаты, ограниченные, вообще говоря, лишь точностью вычислений. Полученные изображения можно свободно увеличивать во много раз. Алгоритмы, работающие в объектном пространстве, особенно полезны в тех приложениях, где необходима высокая точность. Алгоритмы же, работающие в пространстве изображения, имеют дело с системой координат того экрана, на котором объекты визуализируются. При этом точность вычислений ограничена разрешающей способностью экрана. Обычно разрешение экрана бывает довольно низким, типичный пример — $512 \times 512$ точек. Результаты, полученные в пространстве изображения, а затем увеличенные во много раз, не будут соответствовать исходной сцене. Например, могут не совпасть концы отрезков. Алгоритмы, формирующие список приоритетов, работают попеременно в обеих упомянутых системах координат.

Объем вычислений для любого алгоритма, работающего в объектном пространстве, и сравнивающего каждый объект сцены со всеми остальными объектами этой сцены, растет теоретически как квадрат числа объектов ($n^2$). Аналогично, объем вычислений любого алгоритма, работающего в пространстве изображения и сравнивающего каждый объект сцены с позициями всех пикселов в системе координат экрана, растет теоретически, как $nN$. Здесь $n$ обозначает количество объектов (тел, плоскостей или ребер) в сцене, а $N$ — число пикселов. Теоретически трудоемкость алгоритмов, работающих в объектном пространстве, меньше трудоемкости алгоритмов, работающих в пространстве изображения, при $n < N$. Поскольку $N$ обычно равно $(512)^2$, то теоретически большинство алго-

ритмов следует реализовывать в объектном пространстве. Однако на практике это не так. Дело в том, что алгоритмы, работающие в пространстве изображения, более эффективны потому, что для них легче воспользоваться преимуществом когерентности при растровой реализации.

В следующих разделах дается подробное изложение некоторых алгоритмов, работающих как в объектном пространстве, так и в пространстве изображения. Каждый из них иллюстрирует одну или несколько основополагающих идей теории алгоритмов удаления невидимых линий и поверхностей.

## 4.2. АЛГОРИТМ ПЛАВАЮЩЕГО ГОРИЗОНТА

Алгоритм плавающего горизонта чаще всего используется для удаления невидимых линий трехмерного представления функций, описывающих поверхность в виде

$$F(x, y, z) = 0$$

Подобные функции возникают во многих приложениях в математике, технике, естественных науках и других дисциплинах.

Предложено много алгоритмов, использующих этот подход (см. [4-2]—[4-6]). Поскольку в приложениях в основном интересуются описанием поверхности, этот алгоритм обычно работает в пространстве изображения. Главная идея данного метода заключается в сведении трехмерной задачи к двумерной путем пересечения исходной поверхности последовательностью параллельных секущих плоскостей, имеющих постоянные значения координат $x$, $y$ или $z$.

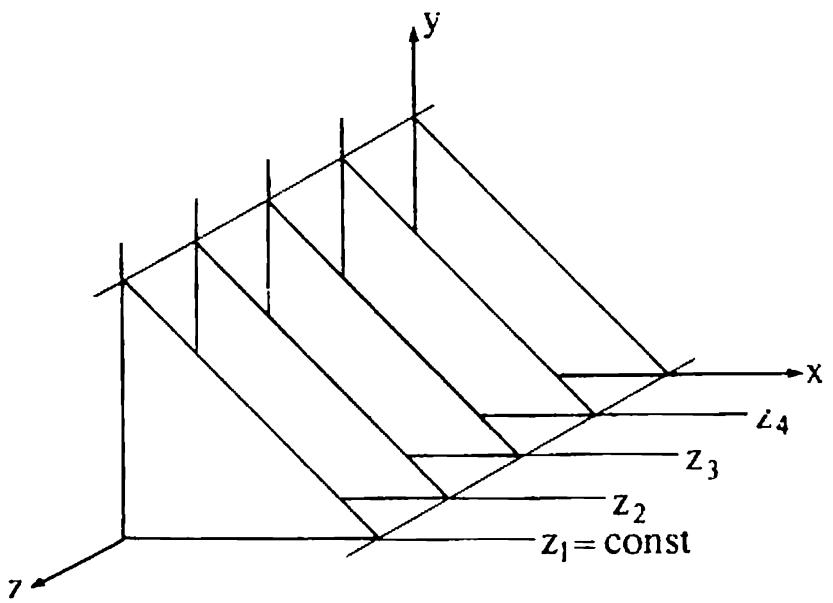



Рис. 4.2. Секущие плоскости с постоянной координатой.

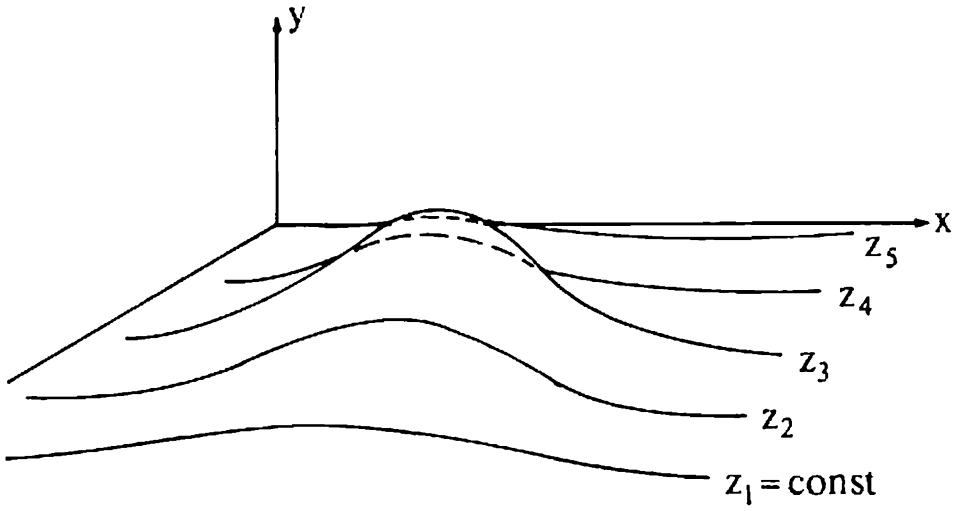



Рис. 4.3. Кривые в секущих плоскостях с постоянной координатой.

На рис. 4.2 приведен пример, где указанные параллельные плоскости определяются постоянными значениями $z$. Функция $F(x, y, z) = 0$ сводится к последовательности кривых, лежащих в каждой из этих параллельных плоскостей, например к последовательности

$$y = f(x, z) \text{ или } x = g(y, z)$$

где $z$ постоянно на каждой из заданных параллельных плоскостей.

Итак, поверхность теперь складывается из последовательности кривых, лежащих в каждой из этих плоскостей, как показано на рис. 4.3. Здесь предполагается, что полученные кривые являются однозначными функциями независимых переменных. Если спроецировать полученные кривые на плоскость $z = 0$, как показано на рис. 4.4, то сразу становится ясна идея алгоритма удаления невидимых участков исходной поверхности. Алгоритм сначала упорядочивает плоскости $z = \text{const}$ по возрастанию расстояния до них от точки наблюдения. Затем для каждой плоскости, начиная с ближайшей к точке наблюдения, строится кривая, лежащая на ней, т. е. для каждого значения координаты $x$ в пространстве изображения определяется соответствующее значение $y$. Алгоритм удаления невидимой линии заключается в следующем:

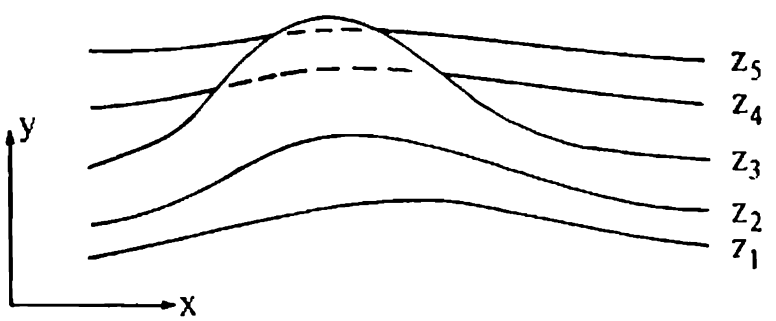



Рис. 4.4. Проекция кривых на плоскость $z = 0$.

Если на текущей плоскости при некотором заданном значении $x$ соответствующее значение $y$ на кривой больше значения $y$ для всех предыдущих кривых при этом значении $x$, то текущая кривая видима в этой точке; в противном случае она невидима.

Невидимые участки показаны пунктиром на рис. 4.4. Реализация данного алгоритма достаточно проста. Для хранения максимальных значений $y$ при каждом значении $x$ используется массив, длина которого равна числу различимых точек (разрешению) по оси $x$ в пространстве изображения. Значения, хранящиеся в этом массиве, представляют собой текущие значения «горизонта». Поэтому по мере рисования каждой очередной кривой этот горизонт «всплывает». Фактически этот алгоритм удаления невидимых линий работает каждый раз с одной линией.

Алгоритм работает очень хорошо до тех пор, пока какая-нибудь очередная кривая не окажется ниже самой первой из кривых, как показано на рис. 4.5, а. Подобные кривые, естественно, видимы и представляют собой нижнюю сторону исходной поверхности, однако алгоритм будет считать их невидимыми. Нижняя сторона поверхности делается видимой, если модифицировать этот алгоритм, включив в него нижний горизонт, который опускается вниз по ходу работы алгоритма. Это реализуется при помощи второго массива, длина которого равна числу различимых точек по оси $x$ в пространстве изображения. Этот массив содержит наименьшие значения $y$ для каждого значения $x$. Алгоритм теперь становится таким:

Если на текущей плоскости при некотором заданном значении $x$ соответствующее значение $y$ на кривой больше максимума или меньше минимума по $y$ для всех предыдущих кривых при этом $x$, то текущая кривая видима. В противном случае она невидима.

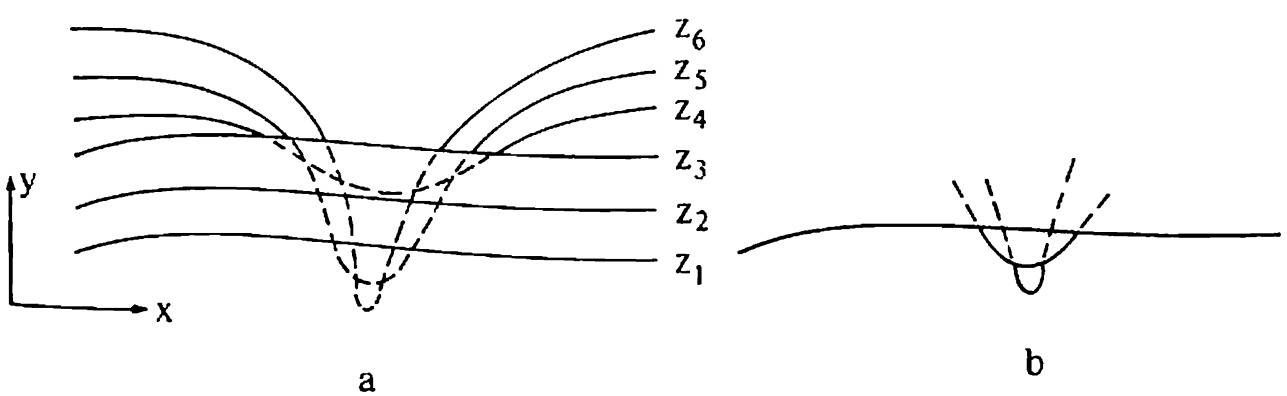



Рис. 4.5. Обработка нижней стороны поверхности.

Полученный результат показан на рис. 4.5, b.

В изложенном алгоритме предполагается, что значение функции, т. е. $y$, известно для каждого значения $x$ в пространстве изображения. Однако если для каждого значения $x$ нельзя указать (вычислить) соответствующее ему значение $y$, то невозможно поддерживать массивы верхнего и нижнего плавающих горизонтов. В таком случае используется линейная интерполяция значений $y$ между известными значениями для того, чтобы заполнить массивы верхнего и нижнего плавающих горизонтов, как показано на рис. 4.6. Если видимость кривой меняется, то метод с такой простой интерполяцией не даст корректного результата. Этот эффект проиллюстрирован рис. 4.7, а. Предполагая, что операция по заполнению массивов проводится после проверки видимости, получаем, что при переходе текущей кривой от видимого к невидимому состоянию (сегмент $AB$ на рис. 4.7, а), точка $(x_{n+k}, y_{n+k})$ объявляется невидимой. Тогда участок кривой между точками $(x_n, y_n)$ и $(x_{n+k}, y_{n+k})$ не изображается и операция по заполнению массивов не проводится. Образуется зазор между текущей и предыдущей кривыми. Если на участке текущей кривой происходит переход от невидимого состояния к видимому (сегмент $CD$ на рис. 4.7, а), то точка $(x_{m+k}, y_{m+k})$ объявляется видимой, а участок кривой между точками $(x_m, y_m)$ и $(x_{m+k}, y_{m+k})$ изображается и операция по заполнению массивов проводится. Поэтому изображается и невидимый кусок сегмента $CD$. Кроме того, массивы плавающих горизонтов не будут содержать точных значений $y$. А это может повлечь за собой дополнительные нежелательные эффекты для последующих кривых. Следо-

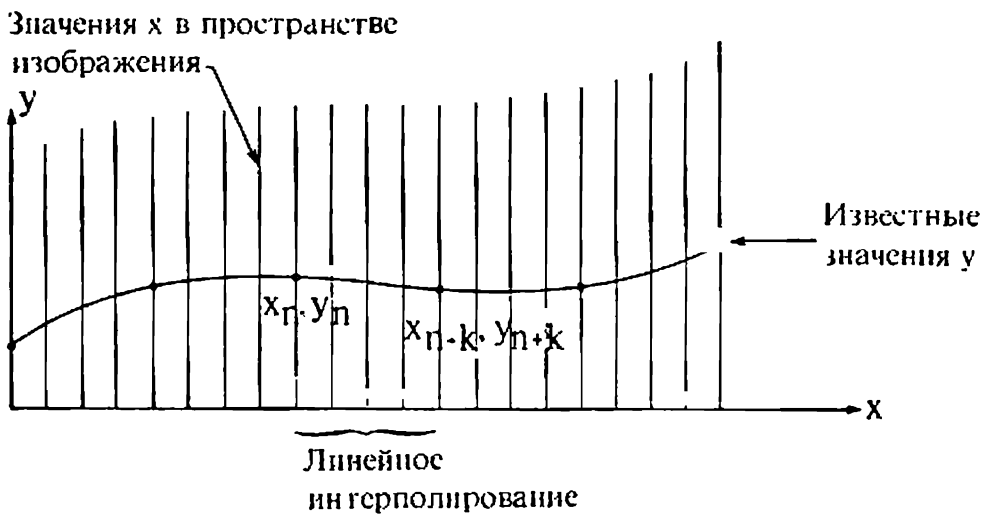



Рис. 4.6. Линейная интерполяция между заданными точками

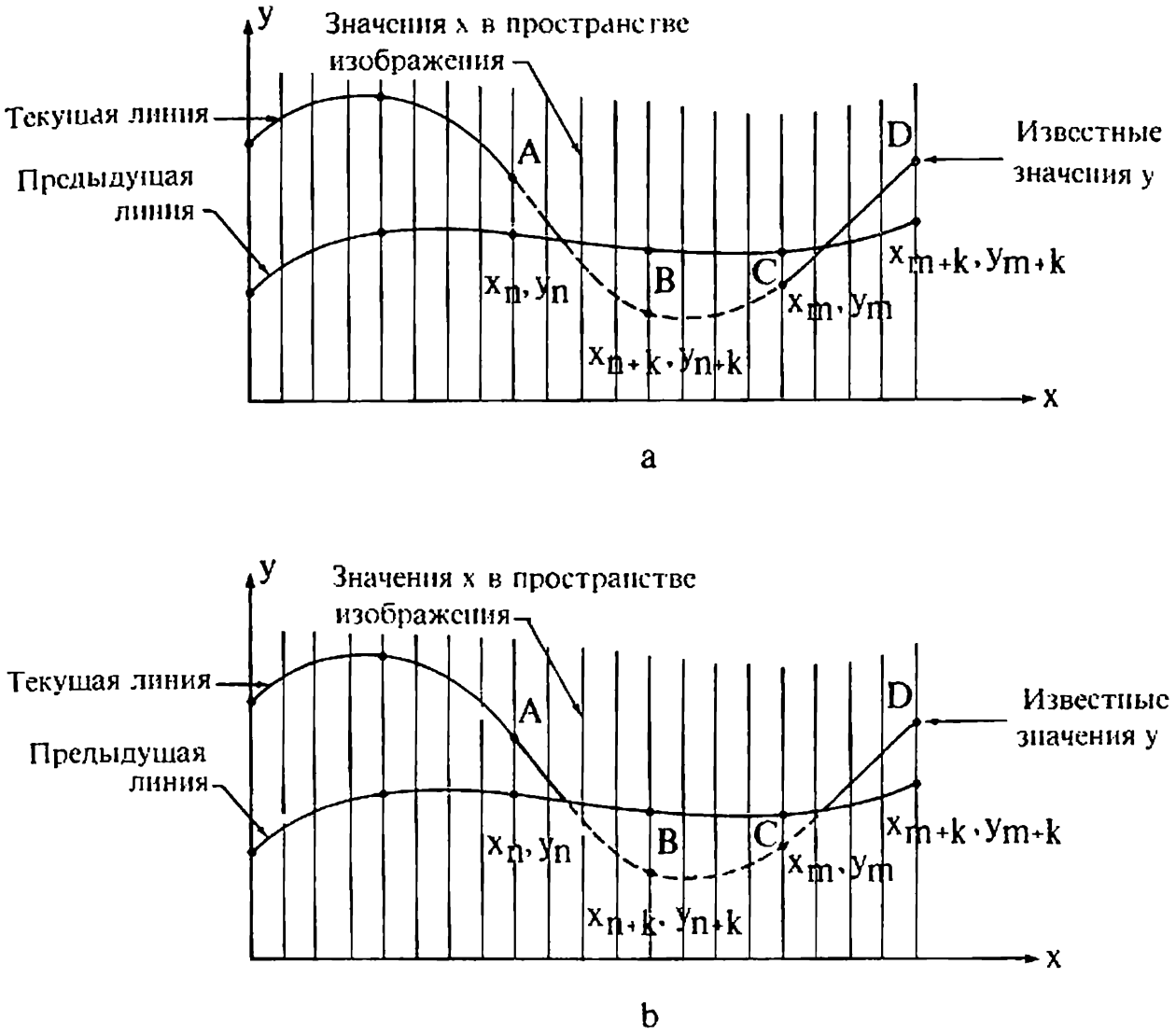



Рис. 4.7. Эффект пересекающихся кривых.

вательно, необходимо решать задачу о поиске точек пересечения сегментов текущей и предшествующей кривых.

Существует несколько методов получения точек пересечения кривых. На растровых дисплеях значение координаты $x$ можно увеличивать на 1, начиная с $x_n$ или $x_m$ (рис. 4.7, а). Значение $y$, соответствующее текущему значению координаты $x$ в пространстве изображения, получается путем добавления к значению $y$, соответствующему предыдущему значению координаты $x$, вертикального приращения $\Delta y$ вдоль заданной кривой. Затем определяется видимость новой точки с координатами $(x + 1, y + \Delta y)$. Если эта точка видима, то активируется связанный с ней пиксел. Если невидима, то пиксел не активируется, а $x$ увеличивается на 1. Этот процесс продолжается до тех пор, пока не встретится $x_{n+k}$ или $x_{m+k}$. Пересечения для растровых дисплеев определяются изложенным методом с достаточной точностью. Близкий и даже более элегантный метод определения пересечений основан на двоичном поиске [4-6].

Точное значение точки пересечения двух прямолинейных отрезков, которые интерполируют текущую и предшествующую кривые, между точками $(x_n, y_n)$ и $(x_{n+k}, y_{n+k})$ (рис. 4.7) задается формулами:

$$x = x_n - \frac{\Delta x(y_{np} - y_{nc})}{(\Delta y_p - \Delta y_c)}$$

$$y = m(x - x_n) + y_n$$

где

$$\Delta x = x_{n+k} - x_n$$

$$\Delta y_p = (y_{n+k})_p - (y_n)_p$$

$$\Delta y_c = (y_{n+k})_c - (y_n)_c$$

$$m = [(y_{n+k}) - (y_n)]/\Delta x$$

а индексы $c$ и $p$ соответствуют текущей (current) и предшествующей (previous) кривым. Полученный результат показан на рис. 4.7, b. Теперь алгоритм излагается более формально.

Если на текущей плоскости при некотором заданном значении $x$ соответствующее значение $y$ на кривой больше максимума или меньше минимума по $y$ для всех предыдущих кривых при этом $x$, то текущая кривая видима. В противном случае она невидима.

Если на участке от предыдущего $(x_n)$ до текущего $(x_{n+k})$ значения $x$ видимость кривой изменяется, то вычисляется точка пересечения $(x_i)$.

Если на участке от $x_n$ до $x_{n+k}$ сегмент кривой полностью видим, то он изображается целиком; если он стал невидимым, то изображается фрагмент от $x_n$ до $x_i$; если же он стал видимым, то изображается фрагмент от $x_i$ до $x_{n+k}$.

Заполнить массивы верхнего и нижнего плавающих горизонтов.

Изложенный алгоритм приводит к некоторым дефектам, когда кривая, лежащая в одной из более удаленных от точки наблюдения плоскостей, появляется слева или справа из-под множества кривых, лежащих в плоскостях, которые ближе к указанной точке наблюдения. Этот эффект продемонстрирован на рис. 4.8, где уже обработанные плоскости $n-1$ и $n$ расположены ближе к точке наблюдения. На рисунке показано, что получается при обработке плоскости $n+1$. После обработки кривых $n-1$ и $n$ верхний горизонт для значений $x = 0$ и 1 равен начальному значению $y$; для значений $x$ от 2

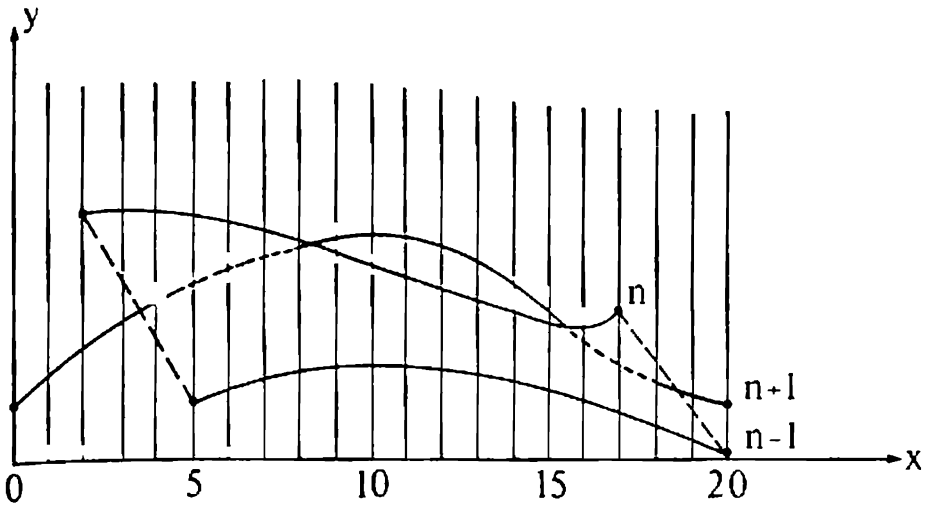



Рис. 4.8. Эффект зазубренного ребра.

до 17 он равен ординатам кривой $n$; а для значений 18, 19, 20 — ординатам кривой $n-1$. Нижний горизонт для значений $x = 0$ и 1 равен начальному значению $y$; для значений $x = 2, 3, 4$ — ординатам кривой $n$; а для значений $x$ от 5 до 20 — ординатам кривой $n-1$. При обработке текущей кривой $(n+1)$ алгоритм объявляет ее видимой при $x = 4$. Это показано сплошной линией на рис. 4.8. Аналогичный эффект возникает и справа при $x = 18$. Такой эффект приводит к появлению зазубренных боковых ребер. Проблема с зазубренностью боковых ребер решается включением в массивы верхнего и нижнего горизонтов ординат, соответствующих штриховым линиям на рис. 4.8. Это можно выполнить эффективно, создав ложные боковые ребра. Приведем алгоритм, реализующий эту идею для обеих ребер.

Обработка левого бокового ребра:

Если $P_n$ является первой точкой на первой кривой, то запомним $P_n$ в качестве $P_{n-1}$ и закончим заполнение. В противном случае создадим ребро, соединяющее $P_{n-1}$ и $P_n$.

Занесем в массивы верхнего и нижнего горизонтов ординаты этого ребра и запомним $P_n$ в качестве $P_{n-1}$.

Обработка правого бокового ребра:

Если $P_n$ является последней точкой на первой кривой, то запомним $P_n$ в качестве $P_{n-1}$ и закончим заполнение. В противном случае создадим ребро, соединяющее $P_n$ и $P_{n-1}$.

Занесем в массивы верхнего и нижнего горизонтов ординаты этого ребра и запомним $P_n$ в качестве $P_{n-1}$.

Теперь полный алгоритм выглядит так:

Для каждой плоскости $z$ = const.

Обработать левое боковое ребро.

Для каждой точки, лежащей на кривой из текущей плоскости:

Если при некотором заданном значении $x$ соответствующее значение $y$ на кривой больше максимума или меньше минимума по $y$ для всех предыдущих кривых при этом $x$, то кривая видима (в этой точке). В противном случае она невидима.

Если на сегменте от предыдущего ($x_n$) до текущего ($x_{n+k}$) значения $x$ видимость кривой изменяется, то вычисляется пересечение ($x_i$).

Если на участке от $x_n$ до $x_{n+k}$ сегмент кривой полностью видим, то он изображается целиком; если он стал невидимым, то изображается его кусок от $x_n$ до $x_i$; если же он стал видимым, то изображается его кусок от $x_i$ до $x_{n+k}$.

Заполнить массивы верхнего и нижнего плавающих горизонтов.

Обработать правое боковое ребро.

Если функция содержит очень острые участки (пики), то приведенный алгоритм может дать некорректные результаты. Этот эффект показан на рис. 4.9. Здесь самая нижняя линия ($z$ = 1) содержит пик. При $x$ = 8 следующая линия ($z$ = 2) объявляется видимой. При $x$ = 12 эта линия ($z$ = 2) объявляется невидимой, опреде-

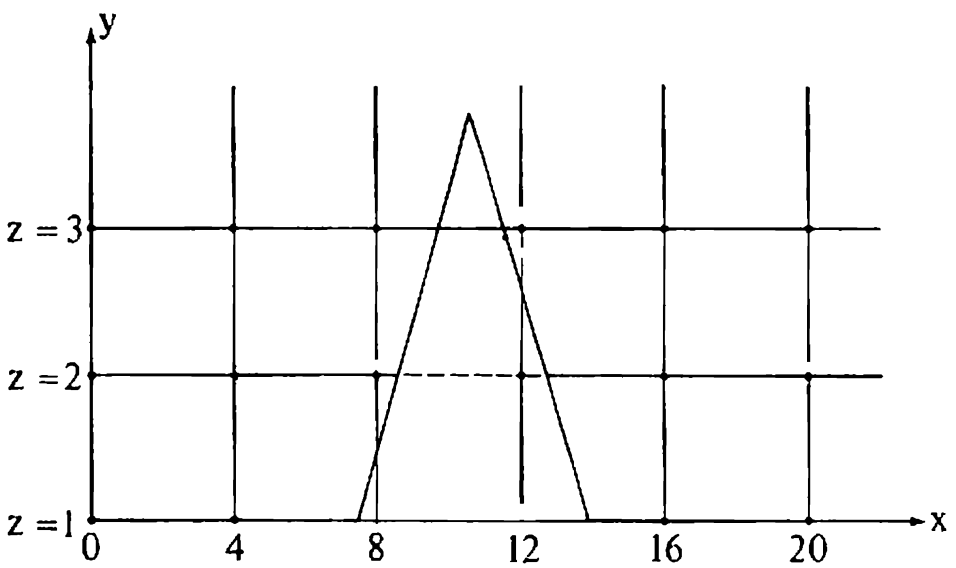



Рис. 4.9. Область с очень острым углом.

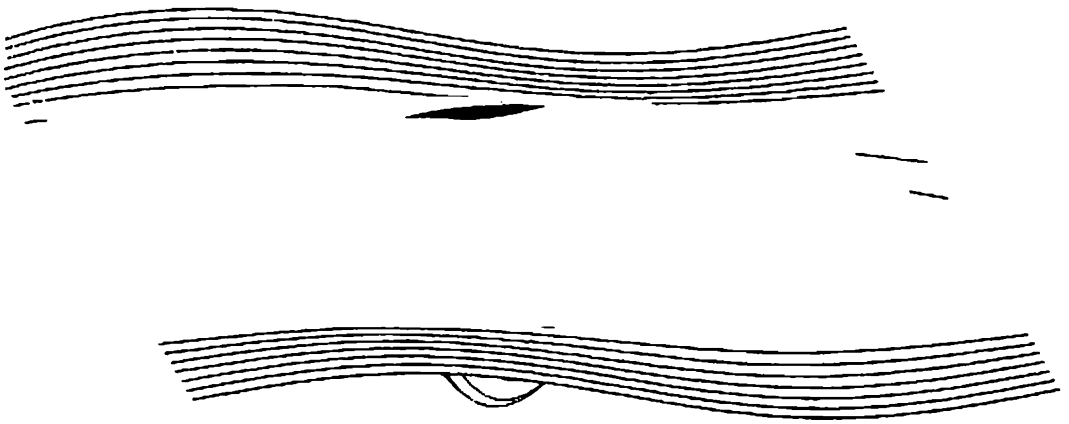


Рис. 4.10. Функция $y = (1/5)\sin x \cos z - (3/2)\cos(7a/4) \times \exp(-a)$, $a = (x - \pi)^2 + (z - \pi)^2$, изображенная в интервале $(0, 2\pi)$ с помощью алгоритма плавающего горизонта.

ляется точка пересечения и линия ($z = 2$) изображается от $x = 8$ до этой точки. На участке от $x = 12$ до $x = 16$ эта линия ($z = 2$) вновь становится видимой, определяется новая точка пересечения и кривая изображается от этого пересечения до $x = 16$. Следующая линия ($z = 3$) при $x = 8$ видима; однако она объявляется видимой и при $x = 12$. Следовательно, эта линия изображается на участке от $x = 8$ до $x = 12$, несмотря на то что она заслонена пиком. Этот эффект вызван вычислением значений функции и оценкой ее видимости на участках, меньших, чем разрешающая способность экрана, т. е. тем, что функция задана слишком малым количеством точек (см. разд. 2.25). Если встречаются узкие участки, то функцию следует вычислять в большем числе точек. Если в примере на рис. 4.9 функцию вычислять в точках с абсциссами 0, 2, 4, ..., 18, 20, вместо точек 0, 4, ..., 16, 20, то линия $z = 3$ будет изображена правильно.

На рис. 4.10 показан типичный результат работы алгоритма плавающего горизонта. Запись этого алгоритма на псевдокоде приводится ниже.

### *Алгоритм плавающего горизонта*

Гэкран — *разрешение экрана в горизонтальном направлении*
Вэкран — *разрешение экрана в вертикальном направлении*
Верх — *массив, содержащий ординаты верхнего горизонта*
Низ — *массив, содержащий ординаты нижнего горизонта*
Y — *текущее значение функции* $y = f(x, z)$ *при* $z = const$
Тфлаг — *флаг видимости для текущей точки*
Пфлаг — *флаг видимости для предыдущей точки, равный*
0 = *невидима*
1 = *видима и выше верхнего горизонта*
−1 = *видима и ниже нижнего горизонта*

**Draw** — *графическая команда, которая чертит видимую линию между точками, заданными их координатами*
Xmin, Xmax — *минимальная и максимальная абсциссы функции*
Хшаг — *приращения вдоль оси x*
Zmin Zmax — *минимальная и максимальная аппликаты функции*
Zшаг — *шаг между плоскостями z = const*

```
Dimension Верх (Гэкран), Низ (Гэкран)
инициализация переменных
Хлевое = −1; Yлевое = −1; Хправое = −1; Yправое = −1
инициализация массивов горизонтов
Верх = 0
Низ = Вэкран
Вычисление функции на каждой плоскости z = const, начиная с ближайшей к наблюдателю плоскости Zmax
for z = Zmax to Zmin step − Zшаг
   инициализация предыдущих значений по x и y:
   Хпред и Yпред
   Хпред = Xmin
   Yпред = f(Xmin, z)
   если используется видовое преобразование, то его нужно
   применить к Хпред, Yпред, z в данной точке
   обработка левого бокового ребра
   call Обрребра (Хпред, Yпред, Хлев, Yлев; Верх, Низ)
   call Видимость (Хпред, Yпред, Верх, Низ; Пфлаг)
   для каждой точки на кривой, лежащей в плоскости
   z = const
   for x =   min to Xmax step Хшаг
      y = f(x, z)
      если используется видовое преобразование, то его
      нужно применить к данной точке
      проверка видимости текущей точки и заполнение соответствующего массива горизонта
      call Видимость (x, y, Верх, Низ; Тфлаг)
      if Тфлаг = Пфлаг then
         if Тфлаг = 1 или Тфлаг = −1 then
            Draw (Хпред, Yпред, x, y)
            call Горизонт (Хпред, Yпред, x, y; Верх, Низ)
         else
         end if
      если видимость изменилась, то вычисляется пересече-
```

```
ние и заполняется массив горизонта
else
  if Тфлаг = 0 then
    if Пфлаг = 1 then
      call Пересечение (Хпред, Упред, x, y, Верх; Xi,
      Yi)
    else
      call Пересечение (Хпред, Упред, x, y, Низ; Xi,
      Yi)
    end if
  Draw (Хпред, Упред, Xi, Yi)
  call Горизонт (Хпред, Упред, Xi, Yi; Верх, Низ)
  else
    if Тфлаг = 1 then
      if Пфлаг = 0 then
        call Пересечение (Хпред, Упред, x, y, Верх;
        Xi, Yi)
        Draw (Xi, Yi, x, y)
        call Горизонт (Xi, Yi, x, y; Верх, Низ)
      else
        call Пересечение (Хпред, Упред, x, y, Низ;
        Xi, Yi)
        Draw (Хпред, Упред, Xi, Yi)
        call Горизонт (Хпред, Упред, Xi, Yi; Верх,
        Низ)
        call Пересечение (Хпред, Упред, x, y, Верх;
        Xi, Yi)
        Draw (Xi, Yi, x, y)
        call Горизонт (Xi, Yi, x, y; Верх, Низ)
    end if
  else
    if Пфлаг = 0 then
      call Пересечение (Хпред, Упред, x, y, Верх; Xi,
      Yi)
        Draw (Xi, Yi, x, y)
        call Горизонт (Xi, Yi, x, y; Верх, Низ)
      else
        call Пересечение (Хпред, Упред, x, y, Верх;
        Xi, Yi)
        Draw (Хпред, Упред, Xi, Yi)
        call Горизонт (Хпред, Упред, Xi, Yi; Верх,
```

```
                    Низ)
                    call Пересечение (Хпред, Упред, x, y, Низ;
                    Xi, Yi)
                    Draw (Xi, Yi, x, y)
                    call Горизонт (Xi, Yi, x, y; Верх, Низ)
                  end if
                end if
              end if
            end if
            Вновь инициализировать Пфлаг, Хпред, Упред
            Пфлаг = Тфлаг
            Хпред = x
            Упред = y
          next x
          Обработка правого концевого ребра
          call Обрребра (x, y, Хправ, Управ; Верх, Низ)
        next z
        finish
```

*подпрограмма обработки бокового ребра*

```
subroutine Обрребра (x, y, Хребра, Уребра; Верх, Низ)
    если Хребра = −1, то встречена первая кривая и ребро не
    создается
    if Хребра = −1 then 1
    call Горизонт (Хребра, Уребра, x, y; Верх, Низ)
1   Хребра = x
    Уребра = y
    return
```

*подпрограмма определения видимости точки*

```
subroutine Видимость (x, y, Верх, Низ; Тфлаг)
```

*видимость точки определяется по отношению к верхнему и нижнему плавающим горизонтам. Если точка лежит на самом горизонте, то она считается видимой.*

Тфлаг = 0, *если точка невидима*
= 1, *если она видима и выше верхнего горизонта*
= −1, *если она видима и ниже нижнего горизонта*

x *считается целой*

```
    if y < Верх (x) and y > Низ (x) then Тфлаг = 0
    if y ⩾ Верх (x) then Тфлаг = 1
    if y ⩽ Низ (x) then Тфлаг = −1
    return
```

*подпрограмма заполнения массивов плавающих горизонтов*
**subroutine** Горизонт (X1, Y1, X2, Y2; Верх, Низ)
*Эта подпрограмма использует линейную интерполяцию для заполнения массивов горизонтов между* X1 *и* X2
**Max** (a, b) *определяет большее из a и b*
**Min** (a, b) *определяет меньшее из a и b*

```
   проверка вертикальности наклона
   if (X2 − X1) = 0 then
      Верх (X2) = Max (Верх (X2), Y2)
      Низ (X2) = Min (Низ (X2), Y2)
   else
      Наклон = (Y2 − Y1)/(X2 − X1)
      for x = X1 to X2 step 1
         y = Наклон * (x − X1) + Y1
         Верх (x) = Max(Верх (x), y)
         Низ (x) = Min(Низ (x), y)
      next x
   end if
   return
```

*подпрограмма вычисления пересечения текущей кривой с горизонтом*
**subroutine** Пересечение (X1, Y1, X2, Y2), Массив; Xi, Yi)
*Эта процедура вычисляет пересечение двух отрезков прямых* Массив *содержит информацию о соответствующем горизонте*
**Sign** — *функция, принимающая значения −1, 0, 1, если знак ее аргумента* $<0$, $=0$, $>0$

```
   проверка бесконечности наклона
   if (X2 − X1) = 0 then
      Xi = X2
      Yi = Массив (X2)
   else
   вычисление пересечения
   обход начинается с самой левой используемой точки
   пересечение считается обнаруженным, когда изменяется
   знак разности значений y
   Наклон = (Y2 − Y1)/(X2 − X1)
   Ysign = Sign(Y1 + Наклон − Массив (X1 + 1))
   Csign = Ysign
   Yi = Y1 + Наклон
   Xi = X1 + 1
```

```
    while(Csign = Ysign)
        Yi = Yi + Наклон
        Xi = Xi + 1
        Csign = Sign(Yi − Массив (Xi))
    end while
    выбирается ближайшее целое число
    if |Yi − Наклон − Массив (Xi − 1)| ⩽ |Yi − Массив
    (Xi)| then
        Yi = Yi − Наклон
        Xi = Xi − 1
    end if
end if
return
```

Следующий пример иллюстрирует данный алгоритм.

**Пример 4.1.** Плавающий горизонт

Рассмотрим геометрические функции, описанные в табл. 4.1. Эти функции заданы на плоскостях $z = 0, 3, 6$. В каждой плоскости заданы две линии. Первая из этих линий — прямая, а вторая описывает пилообразную волну, точки которой лежат

**Таблица 4.1.**

| № кривой | № точки | $x$ | $y$ | $z$ | Комментарий |
|---|---|---|---|---|---|
| 1 | 1 | 0 | 0 | 0 | Пилообразная волна |
| | 2 | 2 | 4 | 0 | |
| | 3 | 6 | −4 | 0 | |
| | 4 | 8 | 0 | 0 | |
| 2 | 5 | 0 | 0 | 0 | Прямая линия |
| | 6 | 8 | 0 | 0 | |
| 3 | 7 | 0 | 0 | 3 | Пилообразная волна |
| | 8 | 2 | 4 | 3 | |
| | 9 | 6 | −4 | 3 | |
| | 10 | 8 | 0 | 3 | |
| 4 | 11 | 0 | 0 | 3 | Прямая линия |
| | 12 | 8 | 0 | 3 | |
| 5 | 13 | 0 | 0 | 6 | Пилообразная волна |
| | 14 | 2 | 4 | 6 | |
| | 15 | 6 | −4 | 6 | |
| | 16 | 8 | 0 | 6 | |
| 6 | 17 | 0 | 0 | 6 | Прямая линия |
| | 18 | 8 | 0 | 6 | |

как выше, так и ниже плоскости, в которой лежит прямая. Алгоритм плавающего горизонта легко справляется со случаем пары линий, лежащих в одной плоскости $z = \text{const}$. Однако порядок обработки этих линий влияет на конечный результат. Здесь первой будет рассматриваться прямая.

Перед визуализацией к поверхности, описанной в табл. 4.1, необходимо применить видовое преобразование. Поверхность сначала поворачивается на 30° вокруг оси $y$, а затем на 15° вокруг оси $x$. Результат проецируется на плоскость $z = 0$ из центра проекции, находящейся в бесконечности на оси $+z$ (см. [1-1]). Матрица размером $4 \times 4$ результирующего преобразования однородных координат есть:

$$\begin{bmatrix} 0.866 & 0.129 & 0 & 0 \\ 0 & 0.966 & 0 & 0 \\ 0.5 & -0.224 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Применение этого преобразования приводит к результатам, показанным в табл. 4.2. Эти результаты отображены на целочисленную сетку в диапазонах $0 \leqslant x \leqslant 100$ и $-50 \leqslant y \leqslant 50$, т. е. в координаты пространства изображения.

Упорядочивая кривые по приоритету глубины вдоль оси $z$ и вспоминая, что прямые должны быть обработаны (каждая на своей плоскости $z = \text{const}$) в первую очередь, мы видим, что порядок обработки кривых обратен тому, который указан в табл. 4.2. Порядок обработки таков: 6, 5, 4, 3, 2, 1.

Начальные значения верхнего и нижнего горизонтов равны соответственно $-50$ и 50, как показано в табл. 4.3, которая содержит некоторые состояния экрана, образованного горизонтами. Кроме того, в табл. 4.3 и на рис. 4.11 от *a* до *f* показаны те значения координат (округленные до ближайшего целого числа), которые алгоритм получил для каждой линии. Штриховыми линиями обозначены фиктивные левое и правое боковые ребра.

**Таблица 4.2.**

| № кривой | № точки | $x$ | $y$ |
|---|---|---|---|
| 1 | 1 | 0 | 0 |
| | 2 | 17 | 41 |
| | 3 | 52 | −31 |
| | 4 | 69 | 10 |
| 2 | 5 | 0 | 0 |
| | 6 | 69 | 10 |
| 3 | 7 | 15 | −7 |
| | 8 | 32 | 35 |
| | 9 | 67 | −38 |
| | 10 | 84 | 36 |
| 4 | 11 | 15 | −7 |
| | 12 | 84 | 36 |
| 5 | 13 | 30 | −13 |
| | 14 | 47 | 28 |
| | 15 | 82 | −44 |
| | 16 | 99 | −3 |
| 6 | 17 | 30 | −13 |
| | 18 | 99 | −3 |

**Таблица 4.3.**

| | x | 0 | 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | 100 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Начальные значения горизонтов | В | −50 | −50 | −50 | −50 | −50 | −50 | −50 | −50 | −50 | −50 | −50 |
| | Н | 50 | 50 | 50 | 50 | 50 | 50 | 50 | 50 | 50 | 50 | 50 |
| Рис. 4.11, a | В | −50 | −50 | −50 | −13 | −12 | −10 | −9 | −7 | −6 | −4 | −50 |
| Кривая 6 | Н | 50 | 50 | 50 | −13 | −12 | −10 | −9 | −7 | −6 | −4 | 50 |
| Рис. 4.11, b | В | −50 | −50 | −50 | −13 | 10 | 22 | 1 | −7 | −6 | −4 | −50 |
| Кривая 5 | Н | 50 | 50 | 50 | −13 | −12 | −10 | −9 | −19 | −40 | −25 | 50 |
| Рис. 4.11, c | В | −50 | −50 | −6 | −4 | 10 | 22 | 1 | 1 | 3 | 1 | −50 |
| Кривая 4 | Н | 50 | 50 | −9 | −13 | −12 | −10 | −9 | −19 | −40 | −25 | 50 |
| Рис. 4.11, d | В | −50 | −50 | 5 | 29 | 19 | 22 | 1 | 1 | 3 | 1 | −50 |
| Кривая 3 | Н | 50 | 50 | −9 | −13 | −12 | −10 | −23 | −30 | −40 | −25 | 50 |
| Рис. 4.11, e | В | 0 | 1 | 5 | 29 | 19 | 22 | 9 | 10 | 5 | 1 | −50 |
| Кривая 2 | Н | 0 | −4 | −9 | −13 | −12 | −10 | −23 | −30 | −40 | −25 | 50 |
| Рис. 4.11, f | В | 0 | 24 | 36 | 29 | 19 | 22 | 9 | 10 | 5 | 1 | −50 |
| Кривая 1 | Н | 0 | −4 | −9 | −13 | −12 | −28 | −23 | −30 | −40 | −25 | 50 |

В приведенных выше алгоритме и примере функция $y = F(x, z)$ рассматривалась только при $z = \text{const}$. Часто бывает удобно вычерчивать кривые, полагая постоянными как $z$, так и $x$. При этом возникает эффект перекрестной штриховки. На первый взгляд может показаться, что перекрестную штриховку можно получить путем наложения двух результатов, образованных плоскостями $z = \text{const}$ и $x = \text{const}$. На рис. 4.12 показано, что это не так [4-3]. Обратите внимание на рис. 4.12, c, где стрелки указывают на некорректные места. Верный результат, показанный на рис. 4.12, d, получен обработкой тех кривых из числа лежащих в плоскостях $z = \text{const}$ или $x = \text{const}$, которые ближе всего к горизонтальным при обычном порядке их следования. Однако после обработки каждой кривой, самой близкой к горизонтальной, необходимо обрабатывать участки кривых, лежащих в ортогональных ей плоскостях, которые находятся между указанной кривой и кривой, следующей за ней. Разумеется, при обработке обеих последовательностей кривых нужно использовать одни и те же массивы верхнего и нижнего плавающих горизонтов. В частности, если кривые для функции $y = F(x, z)$ при $z = \text{const}$ наиболее близки к горизонтальным, то после обработки кривой при $z_1$ нужно обработать участки кривых

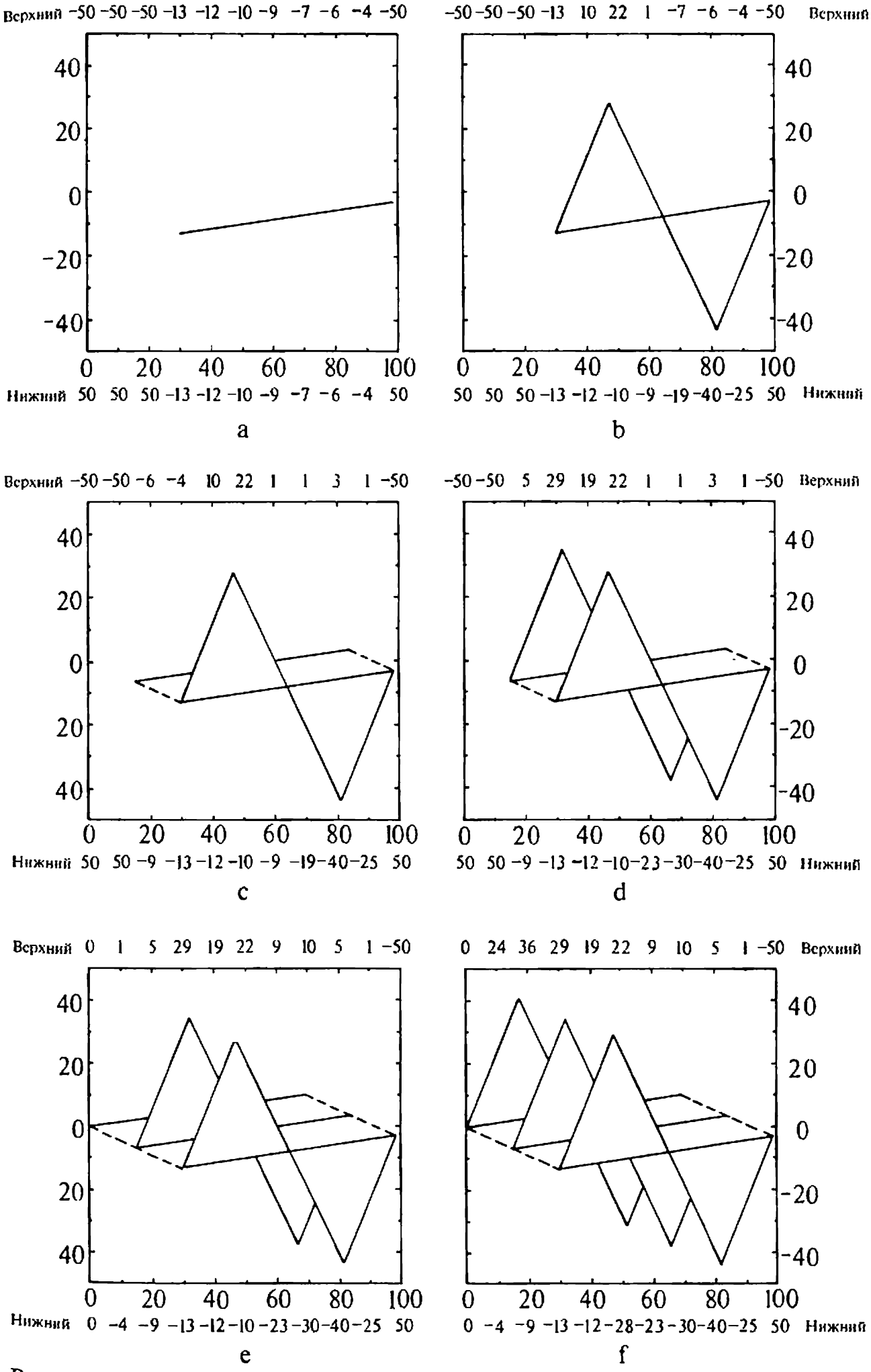



Рис. 4.11. Результаты примера 4.1.

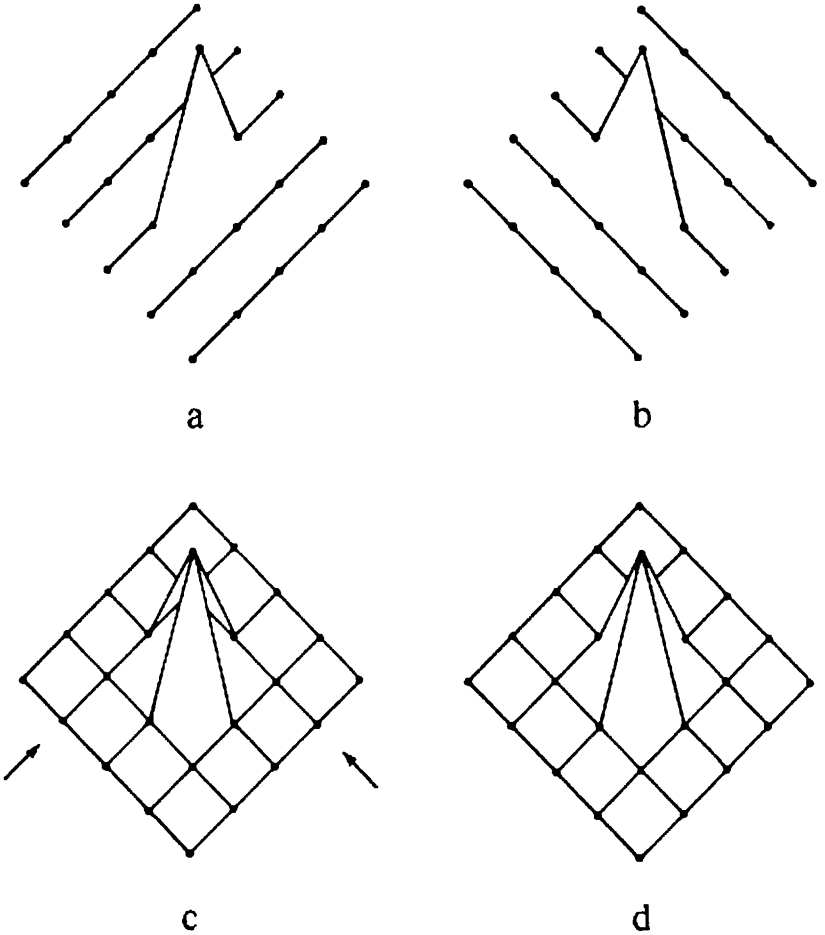



Рис. 4.12. Перекрестная штриховка: линии с постоянным значением $z$ (a); линии с постоянным значением $x$ (b); наложение a и b (c); верный результат (d).

при $x$ = const между $z_1$ и $z_2$, прежде чем обрабатывать кривую в плоскости $z_2$ = const. Если используется перекрестная штриховка, то не надо формировать левое и правое боковые ребра.

## 4.3. АЛГОРИТМ РОБЕРТСА

Алгоритм Робертса представляет собой первое известное решение задачи об удалении невидимых линий [4-7, 4-8]. Это математически элегантный метод, работающий в объектном пространстве. Алгоритм прежде всего удаляет из каждого тела те ребра или грани, которые экранируются самим телом. Затем каждое из видимых ребер каждого тела сравнивается с каждым из оставшихся тел для определения того, какая его часть или части, если таковые есть, экранируются этими телами. Поэтому вычислительная трудоемкость алгоритма Робертса растет теоретически как квадрат числа объектов. Это в сочетании с ростом интереса к растровым дисплеям, работающим в пространстве изображения, привело к снижению интереса к алгоритму Робертса. Однако математические методы, используемые в этом алгоритме, просты, мощны и точны. Кроме того, этот алгоритм можно использовать для иллюстрации некото-

рых важных концепций. Наконец, более поздние реализации алгоритма, использующие предварительную приоритетную сортировку вдоль оси $z$ и простые габаритные или минимаксные тесты, демонстрируют почти линейную зависимость от числа объектов.

В алгоритме Робертса требуется, чтобы все изображаемые тела или объекты были выпуклыми. Невыпуклые тела должны быть разбиты на выпуклые части (см. разд. 3.13). В этом алгоритме выпуклое многогранное тело с плоскими гранями должно представляться набором пересекающихся плоскостей. Уравнение произвольной плоскости в трехмерном пространстве имеет вид

$$ax + by + cz + d = 0 \tag{4.1}$$

В матричной форме этот результат выглядит так:

$$[x \quad y \quad z \quad 1] \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = 0$$

или

$$[x \quad y \quad z \quad 1]\,[P]^T = 0$$

где $[P]^T = [a\ b\ c\ d]$ представляет собой плоскость. Поэтому любое выпуклое твердое тело можно выразить матрицей тела, состоящей из коэффициентов уравнений плоскостей, т. е.

$$[V] = \begin{bmatrix} a_1 & a_2 & \dots & a_n \\ b_1 & b_2 & & b_n \\ c_1 & c_2 & & c_n \\ d_1 & d_2 & & d_n \end{bmatrix}$$

где каждый столбец содержит коэффициенты одной плоскости.

Напомним, что любая точка пространства представима в однородных координатах (см. [1-1]) вектором

$$[S] = [x \quad y \quad z \quad 1]$$

Более того, если точка $[S]$ лежит на плоскости, то $[S] \cdot [P]^T = 0$ (см. [3-5]). Если же $[S]$ не лежит на плоскости, то знак этого скалярного произведения показывает, по какую сторону от плоскости расположена точка. В алгоритме Робертса предполагается, что точки, лежащие внутри тела, дают положительное скалярное произведение. Чтобы проиллюстрировать эти идеи, рассмотрим следующий пример.

**Пример 4.2.** Матрица тела

Шесть плоскостей, описывающих единичный куб с центром в начале координат, таковы: $x_1 = 1/2$, $x_2 = -1/2$, $y_3 = 1/2$, $y_4 = -1/2$, $z_5 = 1/2$, $z_6 = -1/2$. Они изображены на рис. 4.13. Уравнение правой плоскости имеет вид

$$x_1 + 0 \cdot y_1 + 0 \cdot z_1 - (1/2) = 0$$

или

$$2x_1 - 1 = 0$$

Полная матрица тела такова:

$$[V] = \begin{matrix} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{matrix} \begin{matrix} \begin{matrix} ① & ② & ③ & ④ & ⑤ & ⑥ \end{matrix} \\ \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ -1/2 & 1/2 & -1/2 & 1/2 & -1/2 & 1/2 \end{bmatrix} \end{matrix} = \begin{matrix} \begin{matrix} ① & ② & ③ & ④ & ⑤ & ⑥ \end{matrix} \\ \begin{bmatrix} 2 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & 2 \\ -1 & 1 & -1 & 1 & -1 & 1 \end{bmatrix} \end{matrix}$$

Эту матрицу тела следует проверить с помощью одной из тех точек, о которых известно, что они лежат внутри тела, чтобы убедиться, что знаки каждого уравнения плоскости выбраны верно. Если знак скалярного произведения для какой-нибудь плоскости меньше нуля, то соответствующее уравнение плоскости следует умножить на $-1$. Точка внутри куба с координатами $x = 1/4$, $y = 1/4$, $z = 1/4$ представляется в однородных координатах в виде вектора

$$[S] = [1/4 \quad 1/4 \quad 1/4 \quad 1] = [1 \quad 1 \quad 1 \quad 4]$$

Скалярное произведение этого вектора на матрицу объема равно:

$$[S] \cdot [V] = [1 \quad 1 \quad 1 \quad 4] \begin{matrix} \begin{matrix} ① & ② & ③ & ④ & ⑤ & ⑥ \end{matrix} \\ \begin{bmatrix} 2 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & 2 \\ -1 & 1 & -1 & 1 & -1 & 1 \end{bmatrix} \end{matrix}$$

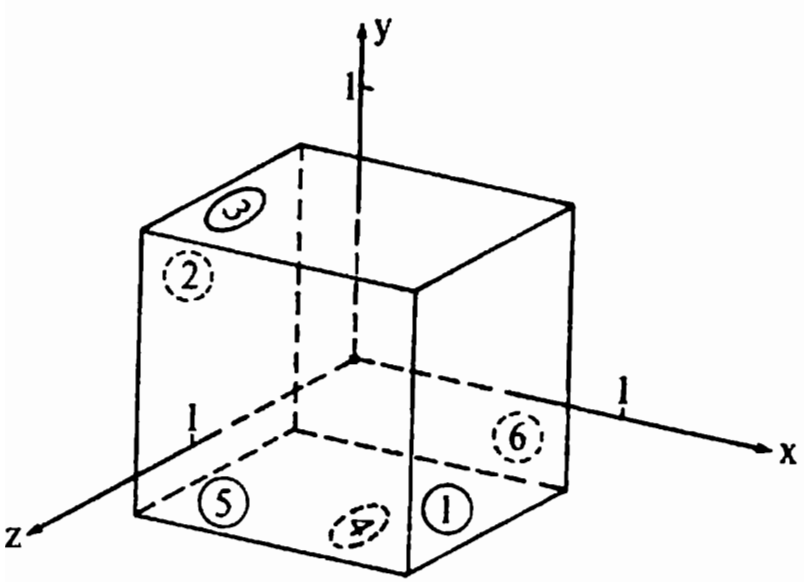



Рис. 4.13. Куб с центром в начале координат.

$$= \begin{matrix} \textcircled{1} & \textcircled{2} & \textcircled{3} & \textcircled{4} & \textcircled{5} & \textcircled{6} \\ [\ -2 & 6 & -2 & 6 & -2 & 6\ ] \end{matrix}$$

Здесь результаты для первого, третьего и пятого уравнений плоскостей (столбцов) отрицательны, и, следовательно, они составлены некорректно. Умножая эти уравнения (столбцы) на $-1$, получаем корректную матрицу тела для данного куба:

$$[V] = \begin{matrix} \textcircled{1} & \textcircled{2} & \textcircled{3} & \textcircled{4} & \textcircled{5} & \textcircled{6} \\ \begin{bmatrix} -2 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & -2 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & -2 & 2 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \end{matrix}$$

В приведенном примере корректность уравнений плоскостей была проверена экспериментально. Разумеется, это не всегда возможно. Существует несколько полезных методов для более общего случая. Хотя уравнение плоскости (4.1) содержит четыре неизвестных коэффициента, его можно нормировать так, чтобы $d = 1$. Следовательно, трех неколлинеарных точек достаточно для определения этих коэффициентов. Подстановка координат трех неколлинеарных точек $(x_1, y_1, z_1)$, $(x_2, y_2, z_2)$, $(x_3, y_3, z_3)$ в нормированное уравнение (4.1) дает:

$$ax_1 + by_1 + cz_1 = -1$$
$$ax_2 + by_2 + cz_2 = -1$$
$$ax_3 + by_3 + cz_3 = -1$$

В матричной форме это выглядит так:

$$\begin{bmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ x_3 & y_3 & z_3 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} -1 \\ -1 \\ -1 \end{bmatrix}$$

или

$$[X][C] = [D] \tag{4.2}$$

Решение этого уравнения дает значения коэффициентов уравнения плоскости:

$$[C] = [X]^{-1}[D]$$

Другой способ используется, если известен вектор нормали к плоскости, т. е.

$$\mathbf{n} = a\mathbf{i} + b\mathbf{j} + c\mathbf{k}$$

где $\mathbf{i}$, $\mathbf{j}$, $\mathbf{k}$ — единичные векторы осей $x$, $y$, $z$ соответственно. Тогда уравнение плоскости примет вид

$$ax + by + cz + d = 0 \tag{4.3}$$

Величина $d$ вычисляется с помощью произвольной точки на плоскости. В частности, если компоненты этой точки на плоскости $(x_1, y_1, z_1)$, то:

$$d = -(ax_1 + by_1 + cz_1) \tag{4.4}$$

Поскольку объем вычислений в алгоритмах удаления невидимых линий или поверхностей растет с увеличением числа многоугольников, для описания поверхностей выгодно использовать многоугольники с более чем тремя сторонами. Эти многоугольники могут быть как невыпуклыми, так и неплоскими. Метод, предложенный Мартином Ньюэлом [4-1], позволяет найти как точное решение для уравнений плоскостей, содержащих плоские многоугольники, так и «наилучшее» приближение для неплоских многоугольников. Этот метод эквивалентен определению нормали в каждой вершине многоугольника посредством векторного произведения прилежащих ребер и усреднения результатов. Если $a$, $b$, $c$, $d$ — коэффициенты уравнения плоскости, то[1)]

$$\begin{aligned} a &= \sum_{i=1}^{n} (y_i - y_j)(z_i + z_j) \\ b &= \sum_{i=1}^{n} (z_i - z_j)(x_i + x_j) \\ c &= \sum_{i=1}^{n} (x_i - x_j)(y_i + y_j) \end{aligned} \tag{4.5}$$

где

$$\textbf{if } i = n \textbf{ then } j = 1 \textbf{ else } j = i + 1$$

а $d$ вычисляется с помощью любой точки на плоскости. Этот метод иллюстрируется примером 4.3.

**Пример 4.3.** Уравнения плоскостей

Рассмотрим четырехсторонний плоский многоугольник, описываемый четырьмя вершинами $V_1$ (1, 0, 0), $V_2$ (0, 1, 0), $V_3$ (0, 0, 1) и $V_4$ (1, −1, 1) (рис. 4.14). Используя вершины $V_1$, $V_2$, $V_4$ и уравнение (4.2), получаем:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & -1 & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} -1 \\ -1 \\ -1 \end{bmatrix}$$

---

[1)] Здесь $n$ — число вершин многоугольника. — *Прим. перев.*

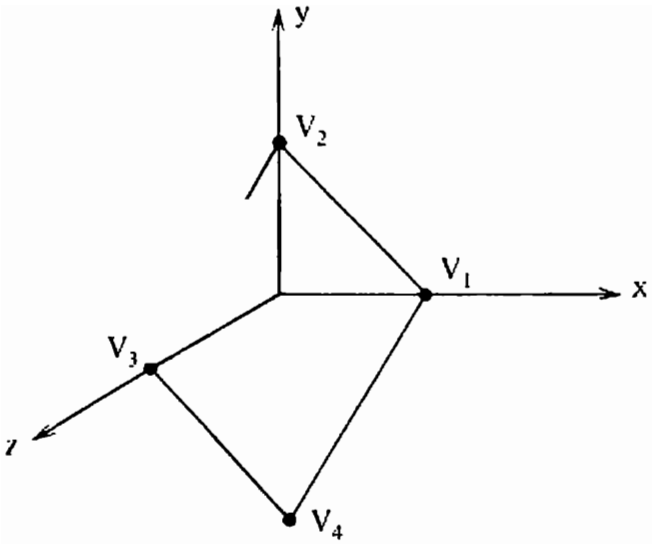



Рис. 4.14. Плоскость в трехмерном пространстве.

Или, разрешая относительно коэффициентов уравнения плоскости:

$$\begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -1 & 1 & 1 \end{bmatrix} \begin{bmatrix} -1 \\ -1 \\ -1 \end{bmatrix} = \begin{bmatrix} -1 \\ -1 \\ -1 \end{bmatrix}$$

Уравнение плоскости теперь имеет вид:

$$-x - y - z + 1 = 0$$

или

$$x + y + z - 1 = 0$$

Решая другим методом, можем получить нормаль к этой плоскости, используя векторное произведение пары векторов, являющихся смежными ребрами одной из вершин, например $V_1$:

$$\mathbf{n} = \mathbf{V_1V_2} \otimes \mathbf{V_1V_3} = \begin{vmatrix} \mathbf{i} & \mathbf{j} & \mathbf{k} \\ (x_2 - x_1) & (y_2 - y_1) & (z_2 - z_1) \\ (x_3 - x_1) & (y_3 - y_1) & (z_3 - z_1) \end{vmatrix}$$

или

$$\mathbf{n} = \begin{vmatrix} \mathbf{i} & \mathbf{j} & \mathbf{k} \\ -1 & 1 & 0 \\ -1 & 0 & 1 \end{vmatrix} = \mathbf{i} + \mathbf{j} + \mathbf{k}$$

где **i**, **j**, **k** — единичные векторы осей $x$, $y$, $z$ соответственно. Используя уравнение (4.4) и $V_4$, получаем значение постоянного члена в уравнении плоскости:

$$d = -1(1 - 1 + 1) = -1$$

Следовательно, уравнение плоскости опять имеет вид:

$$x + y + z - 1 = 0$$

Обращаясь теперь к методу Ньюэла при $n = 4$, получаем из уравнения (4.5):

$$\begin{aligned} a &= (y_1 - y_2)(z_1 + z_2) + (y_2 - y_3)(z_2 + z_3) + (y_3 - y_4)(z_3 + z_4) \\ &\quad + (y_4 - y_1)(z_4 + z_1) \\ &= (-1)(0) + (1)(1) + (1)(2) + (-1)(1) = 2 \end{aligned}$$

$$\begin{aligned} b &= (z_1 - z_2)(x_1 + x_2) + (z_2 - z_3)(x_2 + x_3) + (z_3 - z_4)(x_3 + x_4) \\ &\quad + (z_4 - z_1)(x_4 + x_1) \\ &= (0)(1) + (-1)(0) + (0)(1) + (1)(2) = 2 \end{aligned}$$

$$\begin{aligned} c &= (x_1 - x_2)(y_1 + y_2) + (x_2 - x_3)(y_2 + y_3) + (x_3 - x_4)(y_3 + y_4) \\ &\quad + (x_4 - x_1)(y_4 + y_1) \\ &= (1)(1) + (0)(1) + (-1)(-1) + (0)(-1) = 2 \end{aligned}$$

а используя $V_4$, получаем для постоянного члена:

$$d = -(2 - 2 + 2) = -2$$

После деления на 2 уравнение плоскости вновь принимает вид

$$x + y + z - 1 = 0$$

Пример 4.4 иллюстрирует метод Ньюэла для почти плоских многоугольников.

**Пример 4.4.** Неплоские многоугольники

Рассмотрим почти плоский многоугольник, описанный четырьмя вершинами $V_1$ (1,0,0), $V_2$ (0,1,0), $V_3$ (0,0,1) и $V_4$ (1.1, −1, −1). Вычисляя нормаль в каждой вершине через векторное произведение пары прилежащих ребер, имеем:

$$\begin{aligned} \mathbf{n}_1 &= \mathbf{V}_1\mathbf{V}_2 \otimes \mathbf{V}_1\mathbf{V}_4 = \mathbf{i} + \mathbf{j} + 0.9\mathbf{k} \\ \mathbf{n}_2 &= \mathbf{V}_2\mathbf{V}_3 \otimes \mathbf{V}_2\mathbf{V}_1 = \mathbf{i} + \mathbf{j} + \mathbf{k} \\ \mathbf{n}_3 &= \mathbf{V}_3\mathbf{V}_4 \otimes \mathbf{V}_3\mathbf{V}_2 = \mathbf{i} + 1.1\mathbf{j} + 1.1\mathbf{k} \\ \mathbf{n}_4 &= \mathbf{V}_4\mathbf{V}_1 \otimes \mathbf{V}_4\mathbf{V}_3 = \mathbf{i} + 1.1\mathbf{j} + \mathbf{k} \end{aligned}$$

Усредняя эти нормали, получаем

$$\mathbf{n} = \mathbf{i} + 1.05\mathbf{j} + \mathbf{k}$$

Определение постоянного члена в уравнении плоскости с использованием одной из вершин, например $V_1$, дает $d = -1$. Следовательно, приближенное уравнение плоскости таково:

$$x + 1.05y + z - 1 = 0$$

Метод Ньюэла приводит к такому же результату. В частности,

$$a = (-1)(0) + (1)(1) + (1)(2) + (-1)(1) = 2$$

$$b = (0)(1) + (-1)(0) + (0)(1.1) + (1)(2.1) = 2.1$$

$$c = (1)(1) + (0)(1) + (-1.1)(-1) + (0.1)(-1) = 2$$

Вычисление $d$ с использованием $I_1$ и деление на 2 дает такое же приближенное уравнение плоскости. Аппроксимирующая плоскость проходит через прямую $x = z$ и содержит вершины $V_1$ и $V_2$. Однако $V_2$ и $V_4$ немного смещены по разные стороны от этой плоскости.

Перед началом работы алгоритма удаления невидимых линий или поверхностей для получения желаемого вида сцены часто применяется трехмерное видовое преобразование. Матрицы тел для объектов преобразованной сцены можно получить или преобразованием исходных матриц тел или вычислением новых матриц тел, используя преобразованные вершины или точки.

Если $[B]$ — матрица однородных координат, представляющая исходные вершины тела, а $[T]$ — матрица размером $4 \times 4$ видового преобразования, то преобразованные вершины таковы [1-1]:

$$[BT] = [B][T] \tag{4.6}$$

где $[BT]$ — преобразованная матрица вершин. Использование уравнения (4.2) позволяет получить уравнения исходных плоскостей, ограничивающих тело:

$$[B][V] = [D] \tag{4.7}$$

где $[V]$ — матрица тела, а $[D]$ в правой части — нулевая матрица. Аналогично уравнения преобразованных плоскостей задаются следующим образом:

$$[BT][VT] = [D] \tag{4.8}$$

где $[VT]$ — преобразованная матрица тела. Приравнивая левые части уравнения (4.7) и (4.8), получаем

$$[BT][VT] = [B][V]$$

Подставляя уравнение (4.6), сокращая на $[B]$ и умножая слева на $[T]^{-1}$, имеем

$$[VT] = [T]^{-1}[V]$$

Итак, преобразованная матрица тела получается умножением исходной матрицы тела слева на обратную матрицу видового преобразования. Следующий пример служит этому иллюстрацией.

**Пример 4.5.** Преобразование тела

Рассмотрим перенос единичного куба с центром в начале координат на три единицы в положительном направлении оси $x$. Соответствующая матрица преобразования

размером 4×4 (см. [1-1]) имеет вид

$$[T] = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 3 & 0 & 0 & 1 \end{bmatrix}$$

а обратная к ней матрица, которая может быть получена формально или подбором, такова:

$$[T]^{-1} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -3 & 0 & 0 & 1 \end{bmatrix}$$

Умножение слева матрицы тела данного единичного куба, полученной в примере 4.2, на $[T]^{-1}$ дает матрицу тела для перенесенного куба:

$$[VT] = [T]^{-1}[V] = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -3 & 0 & 0 & 1 \end{bmatrix} \begin{matrix} \\ \\ \\ \\ \end{matrix} \begin{matrix} ① & ② & ③ & ④ & ⑤ & ⑥ \end{matrix}$$
$$\begin{bmatrix} -2 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & -2 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & -2 & 2 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

$$= \begin{matrix} ① & ② & ③ & ④ & ⑤ & ⑥ \end{matrix}$$
$$\begin{bmatrix} -2 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & -2 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & -2 & 2 \\ 7 & -5 & 1 & 1 & 1 & 1 \end{bmatrix}$$

Перенос единичного куба с центром в начале координат на три единицы вправо помещает левую грань на отметку $x = 2.5$, а правую грань — на отметку 3.5. Первый столбец в преобразованной матрице тела содержит коэффициенты уравнения плоскости правой грани:

$$-2x + 7 = 0 \text{ или } x = 3.5$$

что и требовалось. Аналогично второй столбец дает

$$2x - 5 = 0 \text{ или } x = 2.5$$

что и ожидалось для левой грани.

Напомним, что в примере 4.2 точка

$$[S] = [1/4 \quad 1/4 \quad 1 \quad 4] = [1 \quad 1 \quad 1 \quad 4]$$

лежала внутри непреобразованного тела. Следовательно, $[S]\ [V] \geqslant 0$. Однако точка $[S]$ лежит вне перпендикулярного тела. Проверка скалярного произведения $[S]$ и преобразованной матрицы тела

$$\begin{matrix} & & & & & & & \textcircled{1} & \textcircled{2} & \textcircled{3} & \textcircled{4} & \textcircled{5} & \textcircled{6} \end{matrix}$$
$$[S]\cdot[VT] = [1 \quad 1 \quad 1 \quad 4]\cdot[VT] = [26 \quad -18 \quad 2 \quad 6 \quad 2 \quad 6]$$

дает один отрицательный элемент во втором столбце, который соответствует левой грани куба. Это показывает, что данная точка лежит вне тела. Фактически она расположена слева от левой грани куба, т. е. с внешней стороны относительно левой грани, что и показал отрицательный знак.

Если преобразование матрицы точки $[S]$ получается умножением на матрицу преобразования, то

$$[ST] = [S][T] = [1 \ 1 \ 1 \ 4][T] = [13 \ 1 \ 1 \ 4] = [3.25 \ 0.25 \ 0.25 \ 1]$$

Проверка скалярного произведения преобразованной точки с $x = 3.25$ на преобразованную матрицу тела дает

$$\begin{matrix} & & & & & & \textcircled{1} & \textcircled{2} & \textcircled{3} & \textcircled{4} & \textcircled{5} & \textcircled{6} \end{matrix}$$
$$[ST]\cdot[VT] = [2 \quad 6 \quad 2 \quad 6 \quad 2 \quad 6]$$

Этот результат показывает, что преобразованная точка лежит внутри преобразованного тела.

Тот факт, что плоскости имеют бесконечную протяженность и что скалярное произведение точки на матрицу тела отрицательно, если точка лежит вне этого тела, позволяет предложить метод, в котором матрица тела используется для определения граней, которые экранируются самим этим телом. В примере 4.5 показано, что отрицательное скалярное произведение дает только такая плоскость (столбец) в матрице тела, относительно которой точка лежит снаружи. В примере 4.5 таковой объявляется левая плоскость (второй столбец) в преобразованной матрице тела $[VT]$ и непреобразованная точка $[S]$. Эти рассуждения проиллюстрированы на рис. 4.15.

Если зритель находится в бесконечности на положительной полуоси $z$ и смотрит на начало координат, то его взгляд направлен в сторону отрицательной полуоси $z$. В однородных координатах вектор такого направления равен [1-1]:

$$[E] = [0 \ 0 \ -1 \ 0]$$

который служит, кроме того, образом точки, лежащей в бесконечности на отрицательной полуоси $z$. Фактически $[E]$ представляет любую точку, лежащую на плоскости $z = -\infty$, т. е. любую точку типа $(x, y, -\infty)$. Поэтому, если скалярное произведение $[E]$ на

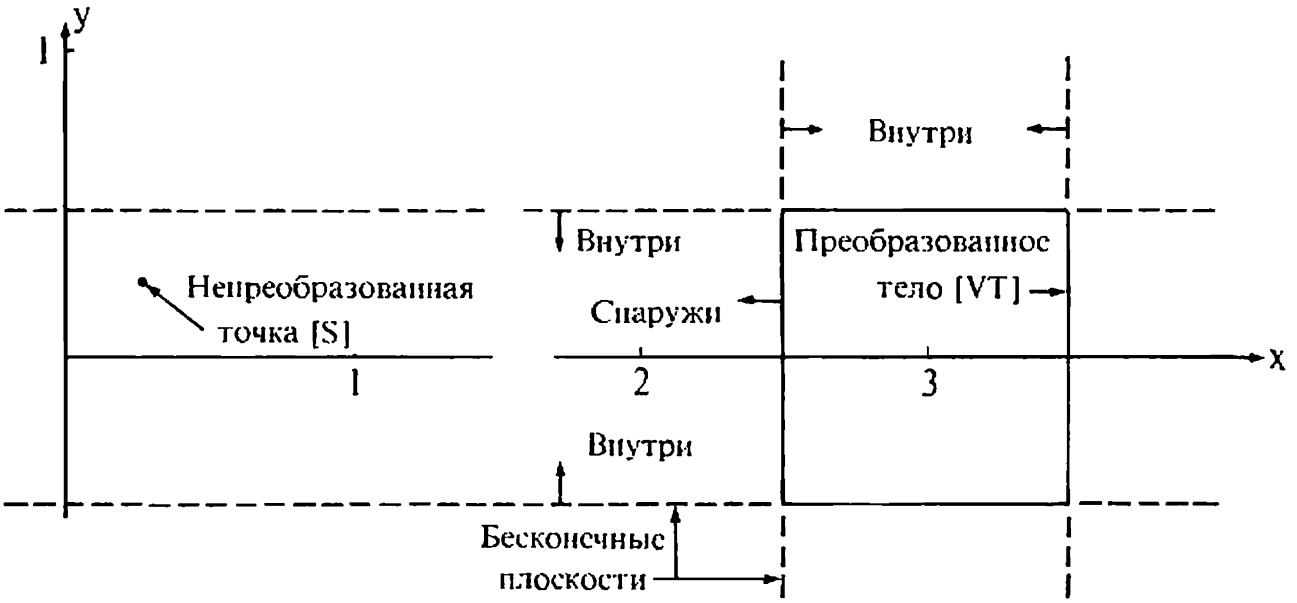



Рис. 4.15. Точка наблюдения вне тела.

столбец, соответствующий какой-нибудь плоскости в матрице тела, отрицательно, то $[E]$ лежит по отрицательную сторону этой плоскости. Следовательно, эти плоскости невидимы из любой точки наблюдения, лежащей в плоскости $z = \infty$, а пробная точка на $z = -\infty$ экранируется самим телом, как показано на рис. 4.16. Такие плоскости называются нелицевыми, а соответствующие им грани — задними. Следовательно,

$$[E] \cdot [V] < 0$$

является условием того, что плоскости — нелицевые, а их грани — задние. Заметим, что для аксонометрических проекций (точка наб-

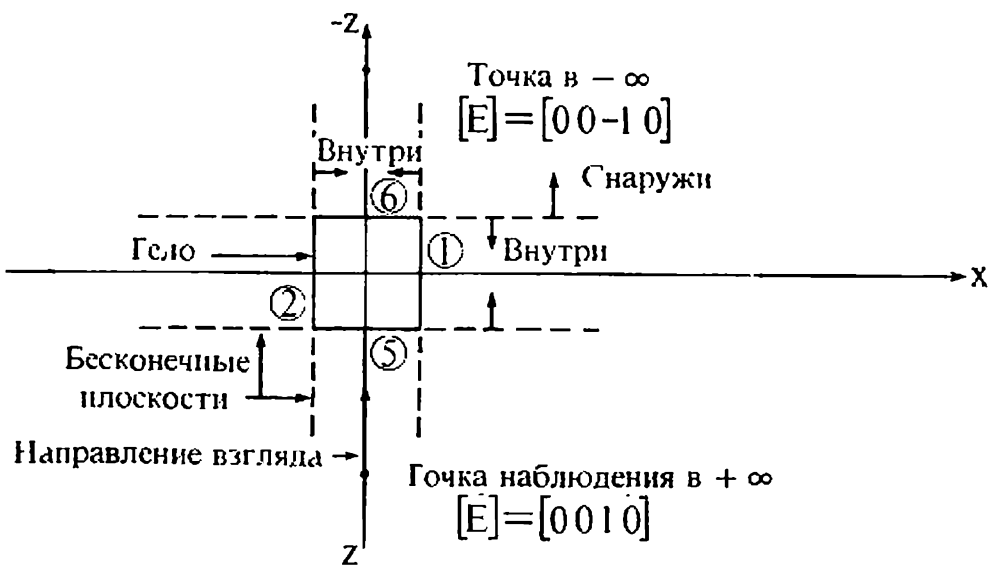



Рис. 4.16. Нелицевые плоскости.

людения в бесконечности) это эквивалентно поиску положительных значений в третьей строке матрицы тела.

Этот метод является простейшим алгоритмом удаления невидимых поверхностей для тел, представляющих собой одиночные выпуклые многогранники. Он также используется для удаления нелицевых или задних граней из сцены перед применением одного из алгоритмов удаления невидимых линий, которые обсуждаются в данной главе ниже. Этот способ часто называют отбрасыванием задних плоскостей. Для выпуклых многогранников число граней при этом сокращается примерно наполовину. Метод эквивалентен вычислению нормали к поверхности для каждого отдельного многоугольника. Отрицательность нормали к поверхности показывает, что нормаль направлена в сторону от наблюдателя и, следовательно, данный многоугольник не виден. Этот метод можно использовать также и для простой закраски (см. гл. 5). Интенсивность или цветовой оттенок многоугольника делается пропорциональным проекции нормали к поверхности на направление взгляда. Следующий пример иллюстрирует данный подход.

**Пример 4.6.** Нелицевая плоскость

Вновь рассмотрим единичный куб с центром в начале координат, как показано на рис. 4.16. Точка наблюдения находится на положительной полуоси $z$, ее координаты равны [0 0 1 0], взгляд направлен на начало координат. Скалярное произведение указанного вектора на матрицу тела дает:

$$[E]\cdot[V] = [0\ 0\ -1\ 0]\begin{bmatrix} \textcircled{1} & \textcircled{2} & \textcircled{3} & \textcircled{4} & \textcircled{5} & \textcircled{6} \\ -2 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & -2 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & -2 & 2 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} \textcircled{1} & \textcircled{2} & \textcircled{3} & \textcircled{4} & \textcircled{5} & \textcircled{6} \\ 0 & 0 & 0 & 0 & 2 & -2 \end{bmatrix}$$

Отрицательное число в шестом столбце показывает, что грань с этим номером нелицевая; рис. 4.16 подтверждает это. Нулевые результаты соответствуют плоскостям, параллельным направлению взгляда.

Данный метод определения нелицевых граней в результате формирует аксонометрическую проекцию на некую плоскость, расположенную бесконечно далеко от любой точки трехмерного пространства. Видовые преобразования, включая перспективное, производятся до определения нелицевых плоскостей. Когда видовое преобразование включает в себя перспективу, то нужно использовать полное перспективное преобразование одного трехмерного пространства в другое, а не перспективное проецирование на неко-

торую двумерную плоскость [1-1]. Полное перспективное преобразование приводит к искажению трехмерного тела, которое затем проецируется на некую плоскость в бесконечности, когда нелицевые плоскости уже определены. Этот результат эквивалентен перспективному проецированию из некоторого центра на конечную плоскость проекции.

Видовое преобразование можно применить к телу так, чтобы точка наблюдения оставалась фиксированной. При другом способе тело остается неподвижным. Соответствующие точка наблюдения и направление взгляда получаются умножением справа на матрицу, обратную матрице видового преобразования. Следующий пример служит иллюстрацией к этим методам.

**Пример 4.7.** Нелицевая плоскость с учетом видового преобразования

Рассмотрим единичный куб с центром в начале координат, повернутый на 45° вокруг оси $y$. Соответствующее видовое преобразование будет иметь вид [1-1]:

$$[R_y] = \begin{bmatrix} \cos\phi & 0 & -\sin\phi & 0 \\ 0 & 1 & 0 & 0 \\ \sin\phi & 0 & \cos\phi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}_{\phi=45^\circ} = \begin{bmatrix} 1/\sqrt{2} & 0 & -1/\sqrt{2} & 0 \\ 0 & 1 & 0 & 0 \\ 1/\sqrt{2} & 0 & 1/\sqrt{2} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Преобразованная матрица тела получается умножением исходной матрицы тела слева на матрицу, обратную матрице видового преобразования. Для чистого поворота обращение матрицы видового преобразования сводится к ее транспонированию. Поэтому

$$[R_y]^{-1} = [R_y]^T = \begin{bmatrix} \cos\phi & 0 & \sin\phi & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\phi & 0 & \cos\phi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}_{\phi=45^\circ} = \begin{bmatrix} 1/\sqrt{2} & 0 & 1/\sqrt{2} & 0 \\ 0 & 1 & 0 & 0 \\ -1/\sqrt{2} & 0 & 1/\sqrt{2} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Преобразованная матрица тела такова:

$$[VT] = [R_y]^{-1}[V] = \begin{matrix} \textcircled{1} & \textcircled{2} & \textcircled{3} & \textcircled{4} & \textcircled{5} & \textcircled{6} \end{matrix}$$
$$\begin{bmatrix} -2/\sqrt{2} & 2/\sqrt{2} & 0 & 0 & -2/\sqrt{2} & 2/\sqrt{2} \\ 0 & 0 & -2 & 2 & 0 & 0 \\ 2/\sqrt{2} & -2/\sqrt{2} & 0 & 0 & -2/\sqrt{2} & 2/\sqrt{2} \\ 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

Если смотреть на начало координат из точки наблюдения [0 0 1 0], лежащей на положительной полуоси $z$, то направление взгляда или пробная точка задаются вектo

ром:

$$[E] = [0 \quad 0 \quad -1 \quad 0]$$

Скалярное произведение $[E]$ на преобразованную матрицу тела равно:

$$\begin{matrix} & ① & ② & ③ & ④ & ⑤ & ⑥ \\ [E]\cdot[VT] = [ & -2/\sqrt{2} & 2\sqrt{2} & 0 & 0 & 2/\sqrt{2} & -2/\sqrt{2} ] \end{matrix}$$

Следовательно, первая и шестая плоскости, которым соответствуют левая и задняя грани в исходном положении куба, — нелицевые (рис 4.17 подтверждает это). Заметим также, что когда тело преобразовано и направление взгляда фиксировано, то поиск отрицательных членов в скалярном произведении пробной точки на преобразованную матрицу тела эквивалентен поиску положительных членов в третьей строке преобразованной матрицы тела.

Эквивалентная точка наблюдения для непреобразованного тела, соответствующая повороту вокруг оси $y$, равна:

$$[0 \quad 0 \quad 1 \quad 0][R_y]^{-1} = [-1/\sqrt{2} \quad 0 \quad 1/\sqrt{2} \quad 0] = [-1 \quad 0 \quad 1 \quad 0]$$

т. е. эта точка расположена на прямой $-x = y$ в бесконечности в направлении положительной полуоси $z$, как показано на рис. 4.17, b. Аналогично определяются эквивалентное направление взгляда и пробная точка:

$$[ET] = [E][R_y]^{-1} = [0 \quad 0 \quad -1 \quad 0][R_y]^{-1} = (1/\sqrt{2})[1 \quad 0 \quad -1 \quad 0]$$

Эта точка расположена на прямой $-x = z$ в бесконечности в направлении отрицательной полуоси $z$. Скалярное произведение этого эквивалентного направления взгляда на исходную матрицу тела равно:

$$\begin{matrix} & & ① & ② & ③ & ④ & ⑤ & ⑥ \\ [ET]\cdot[V] = (1/\sqrt{2})[ & -2 & 2 & 0 & 0 & 2 & -2 ] \end{matrix}$$

Оно вновь показывает, что первая и шестая плоскости нелицевые, и рис. 4.17 подтверждает это.

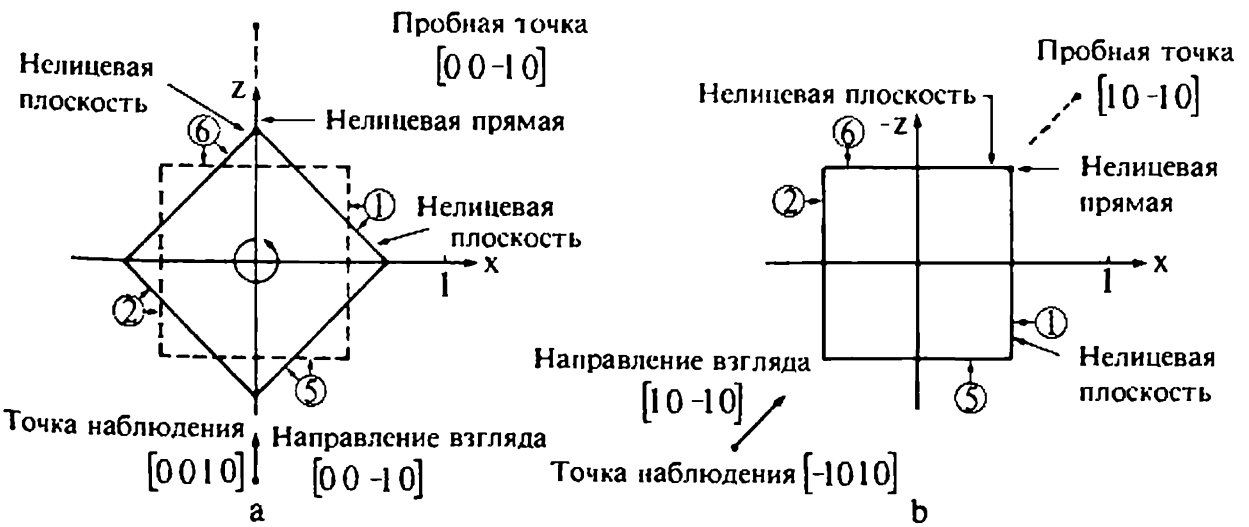



Рис. 4.17. Видовое преобразование и нелицевые плоскости.

После определения нелицевых плоскостей остается найти нелицевые отрезки. Нелицевой отрезок образуется в результате пересечения пары нелицевых плоскостей. Хотя в примере 4.6 шестая плоскость нелицевая, однако нелицевых отрезков нет, поскольку только одна плоскость заслоняет сама себя. В то же время в примере 4.7 ребро, образованное пересечением первой и шестой плоскостей, является нелицевым.

После первого этапа удаления нелицевых отрезков необходимо выяснить, существуют ли такие отрезки, которые экранируются другими телами в картинке или в сцене. Для этого каждый оставшийся отрезок или ребро нужно сравнить с другими телами сцены или картинки. При этом использование приоритетной сортировки ($z$-сортировки) и простого минимаксного или габаритного с прямоугольной объемлющей оболочкой тестов позволяет удалить целые группы или кластеры отрезков и тел. Например, если все тела в сцене упорядочены в некотором приоритетном списке, использующем значения $z$ ближайших вершин для представления расстояния до наблюдателя, то никакое тело из этого списка, у которого ближайшая вершина находится дальше от наблюдателя, чем самая удаленная из концевых точек ребра, не может закрывать это ребро. Более того, ни одно из оставшихся тел, прямоугольная оболочка которого расположена полностью справа, слева, над или под ребром, не может экранировать это ребро. Использование этих приемов значительно сокращает число тел, с которыми нужно сравнивать каждый отрезок или ребро.

Для сравнения отрезка $P_1P_2$ с телом удобно использовать параметрическое представление этого отрезка:

$$P(t) = P_1 + (P_2 - P_1)t \qquad 0 \le t \le 1$$

$$\mathbf{v} = \mathbf{s} + \mathbf{d}t$$

где $\mathbf{v}$ — вектор точки на отрезке, $\mathbf{s}$ — начальная точка, а $\mathbf{d}$ — направление отрезка. Необходимо определить, будет ли отрезок невидимым. Если он невидим, то надо найти те значения $t$, для которых он невидим. Для этого формируется другой параметрический отрезок от точки $P(t)$ до точки наблюдения $\mathbf{g}$:

$$\mathbf{Q}(\alpha, t) = \mathbf{u} = \mathbf{v} + \mathbf{g}\alpha = \mathbf{s} + \mathbf{d}t + \mathbf{g}\alpha \qquad 0 \le t \le 1 \quad \alpha \ge 0$$

Здесь $\alpha$ и $t$ выполняют аналогичные функции. Заданное значение $t$ указывает точку на отрезке $P(t)$, а $\alpha$ указывает точку на отрезке, проведенном от точки $P(t)$ до точки наблюдения. Фактически $\mathbf{Q}(\alpha, t)$ пред-

ставляет собой плоскость в трехмерном пространстве. Пара $(\alpha, t)$ определяет точку на этой плоскости. Значение $\alpha$ положительно, поскольку тела, экранирующие $P(t)$, могут находиться только в той части этой плоскости, которая заключена между отрезком $P(t)$ и точкой наблюдения.

**Пример 4.8.** Параметрическая плоскость

Рассмотрим отрезок, проведенный из точки $P_1(-2,0,-2)$ в точку $P_2(2,0,-2)$ из точки наблюдения, расположенной в бесконечности на положительной полуоси $z$ (рис. 4.18). В однородных координатах $P_1$ и $P_2$ таковы:

$$P_1 = [-2 \quad 0 \quad -2 \quad 1]$$
$$P_2 = [\ 2 \quad 0 \quad -2 \quad 1]$$

Следовательно,

$$P(t) = \mathbf{v} = \mathbf{s} + \mathbf{d}t = [-2 \quad 0 \quad -2 \quad 1] + [4 \quad 0 \quad 0 \quad 0]t$$

Вектор точки наблюдения имеет вид

$$\mathbf{g} = [0 \quad 0 \quad 1 \quad 0]$$

$$Q(\alpha, t) = \mathbf{s} + \mathbf{d}t + \mathbf{g}\alpha = [-2 \quad 0 \quad -2 \quad 1] + [4 \quad 0 \quad 0 \quad 0]t + [0 \quad 0 \quad 1 \quad 0]\alpha$$

Рис. 4.18 и табл. 4.4 показывают, что происходит при изменении $t$ и $\alpha$. В качестве частного примера рассмотрим случай, когда $t = 0.5$ и $\alpha = 3$. Тогда

$$P(0.5) = \mathbf{v} = [-2 \quad 0 \quad -2 \quad 1] + [4 \quad 0 \quad 0 \quad 0](0.5)$$
$$= [\ 0 \quad 0 \quad -2 \quad 1]$$

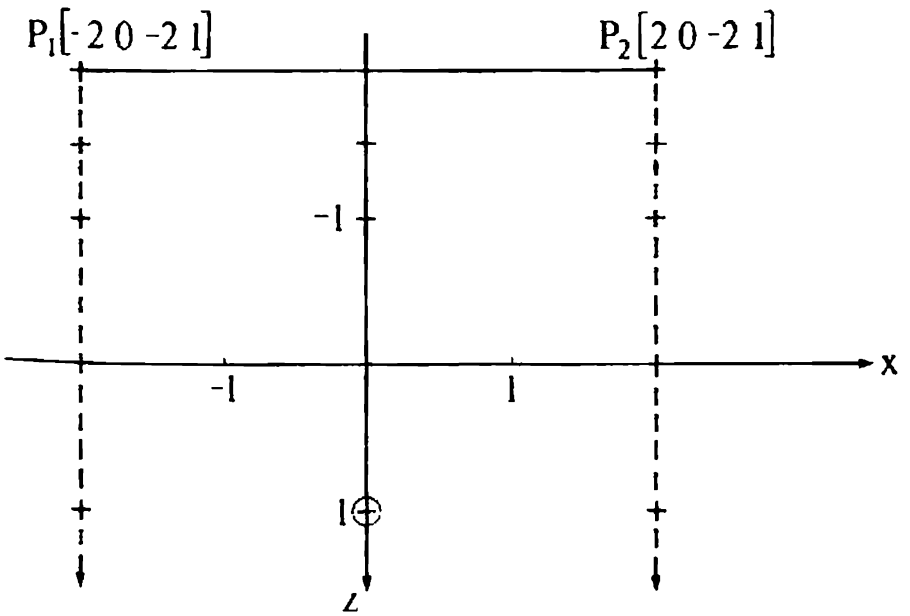



Рис. 4.18. Параметрическая плоскость.

**Таблица 4.4.**

| $t$ | $\alpha$ | $\mathbf{v}(t)$ | $Q(\alpha, t)$ |
|---|---|---|---|
| 0 | 0 | [−2 0 −2 1] | [−2 0 −2 1] |
| | 1/2 | | [−2 0 −3 2 1] |
| | 1 | | [−2 0 −1 1] |
| | 2 | | [−2 0 0 1] |
| | 3 | | [−2 0 1 0] |
| 1/2 | 0 | [ 0 0 −2 1] | [ 0 0 −2 1] |
| | 1/2 | | [ 0 0 −3/2 1] |
| | 1 | | [ 0 0 −1 1] |
| | 2 | | [ 0 0 0 1] |
| | 3 | | [ 0 0 1 0] |
| 1 | 0 | [ 2 0 −2 1] | [ 2 0 −2 1] |
| | 1/2 | | [ 2 0 −3/2 1] |
| | 1 | | [ 2 0 −1 1] |
| | 2 | | [ 2 0 0 1] |
| | 3 | | [ 2 0 1 0] |

Эта точка лежит на отрезке $P_1P_2$ и является точкой его пересечения с осью $z$ при $z = -2$. При $\alpha = 3$

$$Q(3.\ 0.5) = \mathbf{v} + \mathbf{g}\alpha = [0\ \ 0\ \ -2\ \ 1] + [0\ \ 0\ \ 1\ \ 0](3)$$
$$= [0\ \ 0\ \ 1\ \ 1]$$

что соответствует точке на оси $z$ при $z = 1$. Эта точка отмечена кружком на рис. 4.18. Каждая из точек, приведенных в табл. 4.4, отмечена крестиком на рис. 4.18. Заметим, что каждая из полученных прямых параллельна оси $z$.

Напомним, что скалярное произведение любой точки, лежащей внутри тела, на матрицу тела положительно. Если же точка лежит внутри тела, то она невидима. Поэтому для определения части отрезка, которая экранируется телом, достаточно найти те значения $\alpha$ и $t$, для которых скалярное произведение $Q(\alpha, t) = \mathbf{u}$ на матрицу

тела положительно. Это скалярное произведение равно

$$h = \mathbf{u} \cdot [VT] = \mathbf{s} \cdot [VT] + t\mathbf{d} \cdot [VT] + \alpha \mathbf{g} \cdot [VT] > 0 \qquad 0 \leq t \leq 1,\ \alpha \geq 0$$

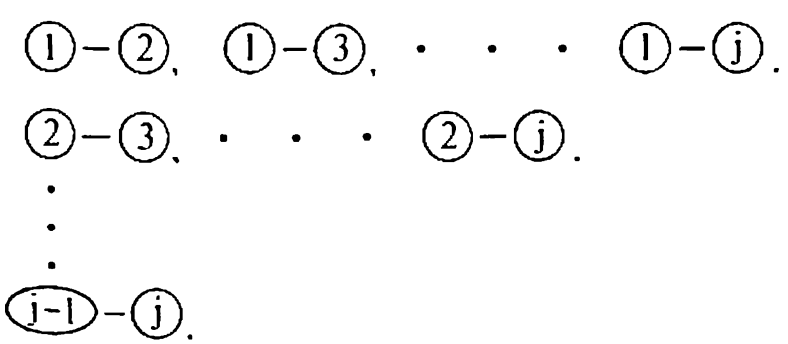
Если все компоненты $h$ неотрицательны для некоторых $t$ и $\alpha$, то отрезок при этих значениях $t$ экранируется данным телом. Обозначив

$$p = \mathbf{s} \cdot [VT]$$
$$q = \mathbf{d} \cdot [VT]$$
$$w = \mathbf{g} \cdot [VT]$$

запишем условия в виде

$$h_j = p_j + tq_j + \alpha w_j > 0 \qquad 0 \leq t \leq 1,\ \alpha \geq 0$$

где $j$ — номер столбца в матрице тела. Эти условия должны выполняться при всех значениях $j$, т. е. для всех плоскостей, ограничивающих объем тела. Пограничный случай между видимостью и невидимостью возникает, когда $h_j = 0$. При $h_j = 0$ точка лежит на плоскости. Полагая $h_j = 0$ для всех плоскостей, мы получим систему уравнений относительно $\alpha$ и $t$, которые должны удовлетворяться одновременно. Результат можно получить путем совместного решения всевозможных пар уравнений из этой системы, при этом будут найдены все значения $\alpha$ и $t$, при которых изменяется значение видимости отрезка. Схема решения показана на рис. 4.19. Число возможных решений при $j$ уравнениях (плоскостях) равно $j(j-1)/2$. Каждое решение в диапазонах $0 \leqslant t \leqslant 1$, $\alpha \geqslant 0$, подставляется во все остальные уравнения для проверки того, что условие $h_j \geqslant 0$ выполнено. Поиск корректных решений производится для того, чтобы найти минимальное среди максимальных значений параметра $t(t_{\text{minmax}})$ и максимальное среди минимальных значений $t(t_{\text{maxmin}})$. Отрезок невидим при $t_{\text{maxmin}} < t < t_{\text{minmax}}$. Последнее требование является простым следствием из классической задачи ли-




Общее число решений при j уравнениях $= \dfrac{(j-1)(j)}{2}$.

Рис. 4.19. Схема решения относительно $\alpha$ и $t$.

нейного программирования. Ниже приводится еще один алгоритм решения данной задачи, напоминающий ранее изложенный алгоритм отсечения Кируса — Бека (см. разд. 3.5). Но сначала дадим несколько примеров, которые помогут пояснить приведенные рассуждения.

**Пример 4.9.** Проверка экранирования отрезков телами

Вновь рассмотрим единичный куб с центром в начале координат. Отрезок от $P_1[-2, 0\ -2\ 1]$ до $P_2[2\ 0\ -2\ 1]$ лежит за этим кубом и частично экранируется им, как показано на рис. 4.20. Снова имеем

$$P(t) = \mathbf{v} = [-2\ \ 0\ \ -2\ \ 1] + [4\ \ 0\ \ 0\ \ 0]t$$
$$\mathbf{s} = [-2\ \ 0\ \ -2\ \ 1]$$
$$\mathbf{d} = [\ 4\ \ 0\ \ \ 0\ \ 0]$$

Для точки наблюдения, расположенной в бесконечности на положительной полуоси $z$, имеем

$$\mathbf{g} = [0\ \ 0\ \ 1\ \ 0]$$

Здесь будем считать, что куб не преобразуется. Следовательно,

$$[VT] = [V] = \begin{bmatrix} -2 & 2. & 0 & 0 & 0 & 0 \\ 0 & 0 & -2 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & -2 & 2 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

Значения $p$, $q$ и $w$ суть результат скалярного произведения $\mathbf{s}$, $\mathbf{d}$ и $\mathbf{g}$ на $[VT]$:

$$p = \mathbf{s} \cdot [VT] = [\ 5\ \ -3\ \ 1\ \ 1\ \ \ 5\ \ -3]$$
$$q = \mathbf{d} \cdot [VT] = [-8\ \ \ 8\ \ 0\ \ 0\ \ \ 0\ \ \ 0]$$
$$w = \mathbf{g} \cdot [VT] = [\ 0\ \ \ 0\ \ 0\ \ 0\ \ -2\ \ \ 2]$$

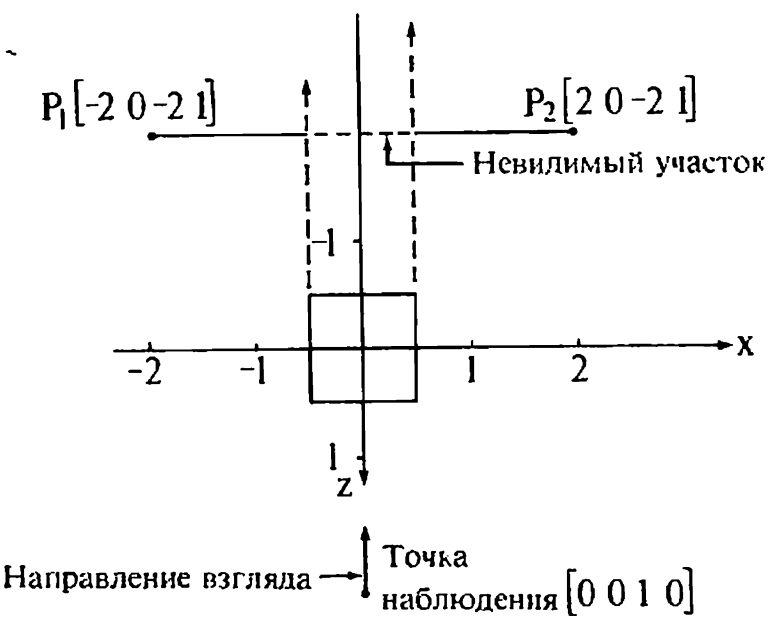



Рис. 4.20. Проверка экранирования отрезка телом.

Используя эти значения, получаем шесть неравенств, соответствующих условию $h_j = p_j + tq_j + \alpha w_j > 0$, по одному на каждую из шести плоскостей, несущих грани куба. Конкретно,

$$\begin{aligned}
&① \quad 5 - 8t && > 0\\
&② \quad -3 + 8t && > 0\\
&③ \quad 1 && > 0\\
&④ \quad 1 && > 0\\
&⑤ \quad 5 && - 2\alpha > 0\\
&⑥ \quad -3 && + 2\alpha > 0
\end{aligned}$$

Третье и четвертое из этих неравенств просто устанавливают, что соответствующие условия всегда выполняются. Они выражают тот геометрический факт, что отрезок находится целиком «внутри» бесконечных полупространств, ограниченных плоскостями, несущими верхнюю и нижнюю грани куба. Превращение оставшихся четырех неравенств в равенства дает следующие их решения: $t = 5/8$, $t = 3/8$, $\alpha = 5/2$ и $\alpha = 3/2$. Разумеется, приведенный пример элементарен. Эти уравнения можно решить, просто посмотрев на них, однако в общем случае это не так.

Каждое из полученных уравнений описывает прямую в пространстве $(\alpha, t)$. Поучительно взглянуть на графическое решение, показанное на рис. 4.21. Штриховкой обозначены те стороны прямых, где расположены возможные решения. Очевидно, что все неравенства $h_j > 0$ удовлетворяются только внутри указанной на рисунке ограниченной области. Итак, $t_{maxmin} = 3/8$, а $t_{minmax} = 5/8$.

Отрезок невидим в диапазоне $3/8 < t < 5/8$ и видим при $0 \leqslant t \leqslant 3/8$ и $5/8 \leqslant t \leqslant 1$.

Используя параметрическое уравнение отрезка, имеем

$$P(3/8) = [-2 \quad 0 \quad -2 \quad 1] + [4 \quad 0 \quad 0 \quad 1](3/8) = [-1/2 \quad 0 \quad -2 \quad 1]$$

и

$$P(5/8) = [-2 \quad 0 \quad -2 \quad 1] + [4 \quad 0 \quad 0 \quad 1](5/8) = [1/2 \quad 0 \quad -2 \quad 1]$$

что и показано на рис. 4.20.

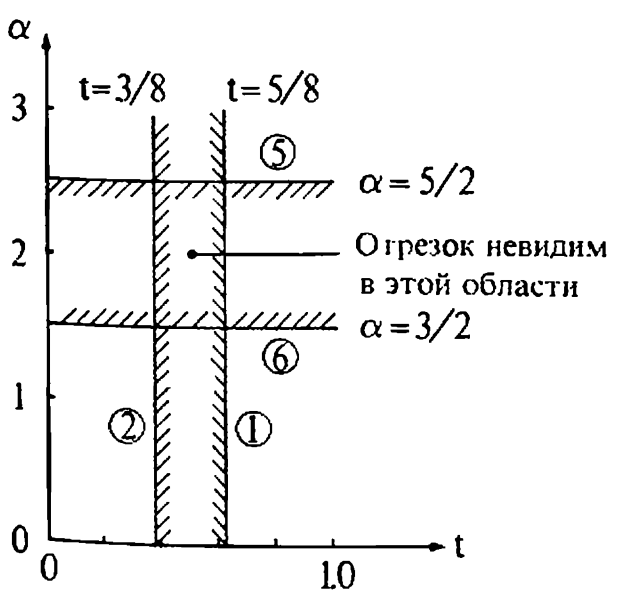



Рис. 4.21. Графическое решение примера 4.9.

В приведенном выше примере получены два значения $t$. Поэтому удается выбрать $t_{maxmin}$ и $t_{minmax}$. Что же делать, если в результате решения уравнения получится только одно значение $t$? Следующие примеры иллюстрируют эту проблему и способ ее решения.

**Пример 4.10.** Решение с единственным значением $t$

Продолжая использовать модель единичного куба с центром в начале координат, рассмотрим отрезок от $P_1[1\ 0\ -1\ 1]$ до $P_2[0\ 0\ -1\ 1]$, как показано на рис. 4.22. Здесь:

$$P(t) = \mathbf{v} = [1\ \ 0\ \ -1\ \ 1] + [-1\ \ 0\ \ 0\ \ 0]t$$

$$\mathbf{s} = [\ 1\ \ 0\ \ -1\ \ 1]$$

$$\mathbf{d} = [-1\ \ 0\ \ \ 0\ \ 0]$$

$$\mathbf{g} = [0\ \ 0\ \ 1\ \ 0]$$

Рассматривается случай непреобразованного куба, т. е. $[VT] = [V]$. Величины $p$, $q$ и $w$ равны:

$$p = \mathbf{s} \cdot [VT] = [-1\ \ \ 3\ \ 1\ \ 1\ \ \ 3\ \ -1]$$

$$q = \mathbf{d} \cdot [VT] = [\ 2\ \ -2\ \ 0\ \ 0\ \ \ 0\ \ \ 0]$$

$$w = \mathbf{g} \cdot [VT] = [\ 0\ \ \ \ 0\ \ 0\ \ 0\ \ -2\ \ \ 2]$$

Система неравенств, выражающих условия $h_j > 0$, такова:

$$\begin{aligned}
&① \quad -1 + 2t && > 0\\
&② \quad \ 3 - 2t && > 0\\
&③ \quad \ 1 && > 0\\
&④ \quad \ 1 && > 0\\
&⑤ \quad \ 3 && -2\alpha > 0\\
&⑥ \quad -1 && +2\alpha > 0
\end{aligned}$$

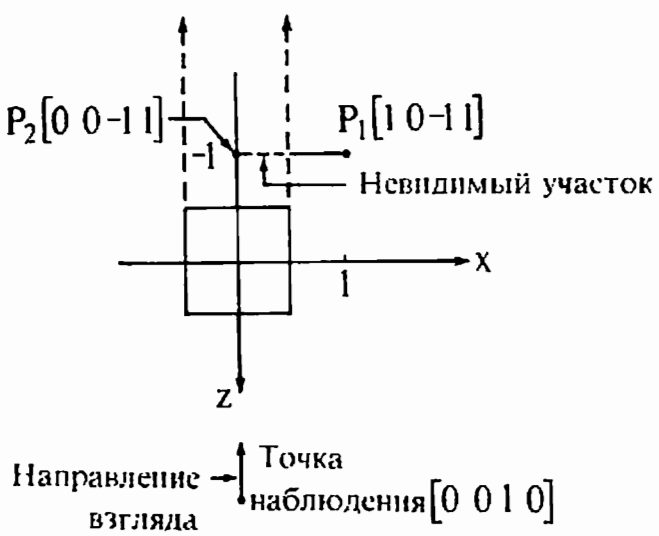



Рис. 4.22. Проверка экранирования отрезка с невидимым концом.

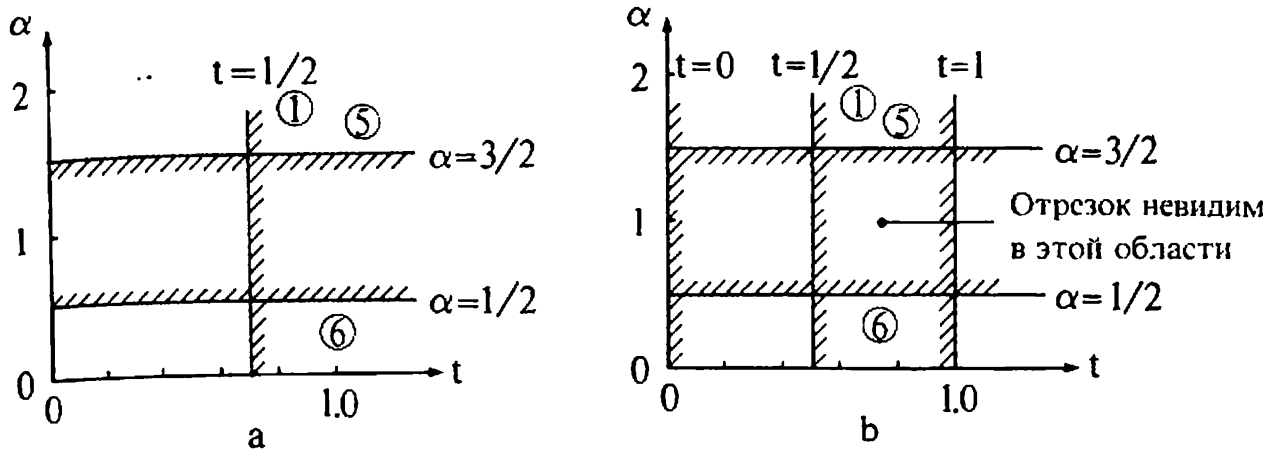



Рис. 4.23. Графическое решение примера 4.10

Решение соответствующих уравнений для $h_j = 0$ дает значения $t = 1/2$, $t = 3/2$, $\alpha = 3\ 2$ и $\alpha = 1/2$. Решение $t = 3/2$ следует отбросить, ибо оно выходит за пределы допустимого диапазона $0 \leqslant t \leqslant 1$. Следовательно, найдено только одно значение $t$. Графическое решение показано на рис. 4.23, а. Вновь штриховкой обозначена та сторона линии, по которую могут лежать допустимые решения. Очевидно, что результат не обращает ограниченную область. Однако в изложенном методе решения не использовались граничные условия, представленные прямыми $t = 0$ и $t = 1$. Как показано на рис. 4.23, b, добавление этих прямых к решению, очевидно, образует требуемую ограниченную область. Поэтому $t_{maxmin} = 1\ 2$ и $t_{minmax} = 1$. Далее, все ограничения $h_j > 0$ удовлетворяются при обоих этих значениях $t$. Следовательно, отрезок видим при $0 \leqslant t \leqslant 1/2$, т. е. от

$$P(0) = [1\ \ 0\ \ -1\ \ 1] + [-1\ \ 0\ \ 0\ \ 0](0) = [1\ \ 0\ \ -1\ \ 1]$$

до

$$P(1/2) = [1\ \ 0\ \ -1\ \ 1] + [-1\ \ 0\ \ 0\ \ 0](1/2) = [1/2\ \ 0\ \ -1\ \ 1]$$

Изменение направления отрезка, т. е. перемена мест $P_1$ и $P_2$, дает область решения от $t = 0$ до $t = 1/2$.

Следующий пример показывает, что граница $\alpha = 0$ тоже должна быть учтена.

**Пример 4.11.** Граница по параметру альфа

Рассмотрим непреобразованный куб и отрезок от $P_1[1\ 0\ 2\ 1]$ до $P_2[-1\ 0\ -2\ 1]$, как показано на рис. 4.24. Этот отрезок протыкает тело. Здесь:

$$P(t) = \mathbf{v} = [1\ \ 0\ \ 2\ \ 1] + [-2\ \ 0\ \ -4\ \ 0]t$$

$$\mathbf{s} = [\ 1\ \ 0\ \ \ 2\ \ 1]$$

$$\mathbf{d} = [-2\ \ 0\ \ -4\ \ 0]$$

Снова точка наблюдения находится в бесконечности и $\mathbf{g} = [0\ 0\ 1\ 0]$. Для непреобразованного куба, т. е. при $[VT] = [V]$

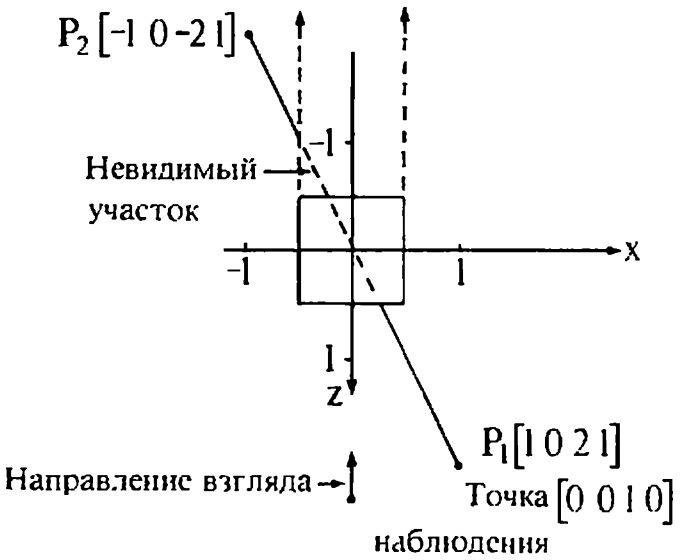



Рис. 4.24. Проверка экранирования в случае отрезка, протыкающего тело.

$$p = \mathbf{s}\cdot[VT] = [-1 \quad 3 \quad 1 \quad 1 \quad -3 \quad 5]$$
$$q = \mathbf{d}\cdot[VT] = [\ 4 \quad -4 \quad 0 \quad 0 \quad 8 \quad -8]$$
$$w = \mathbf{g}\cdot[VT] = [\ 0 \quad 0 \quad 0 \quad 0 \quad -2 \quad 2]$$

Условия $h_j > 0$ имеют вид:

$$\begin{aligned}
&① \quad -1 + 4t \qquad > 0\\
&② \quad 3 - 4t \qquad > 0\\
&③ \quad 1 \qquad > 0\\
&④ \quad 1 \qquad > 0\\
&⑤ \quad -3 + 8t - 2\alpha > 0\\
&⑥ \quad 5 - 8t + 2\alpha > 0
\end{aligned}$$

Решение соответствующих уравнений для $h_j = 0$ дает единственный результат $t = 3/4$. Это решение показано в графическом виде на рис. 4.25, а. Вновь заштрихована та сторона прямой, по которую могут лежать решения. Но решения не образуют ограниченную область. Добавляя границы $t = 0$ и $t = 1$, получаем ограниченную область между $t = 3/4$ и $t = 1$ (рис. 4.25, b). Однако, как показывает штриховка, эта область некорректна, поскольку при $t > 3/4$ не выполняется условие $h_j > 0$ для $j = 2$. Добавление границы $\alpha = 0$ дает уже корректную ограниченную область с краями $t = 3/8$ и $t = 3/4$. Именно эта область дает значения $t_{maxmin} = 3/8$ и $t_{minmax} = 3/4$. Следовательно, отрезок видим при

$$0 \leqslant t \leqslant 3/8 \text{ и } 3/4 \leqslant t \leqslant 1$$

или от $P(0) = [1\ 0\ 2\ 1]$ до $P(3/8) = [1/4\ 0\ 1/2\ 1]$

и от $P(3/4) = [-1/2\ 0\ -1\ 1]$ до $P(1) = [-1\ 0\ -2\ 1]$

Решения на границе $\alpha = 0$ возникают в случае протыкания (объектов).

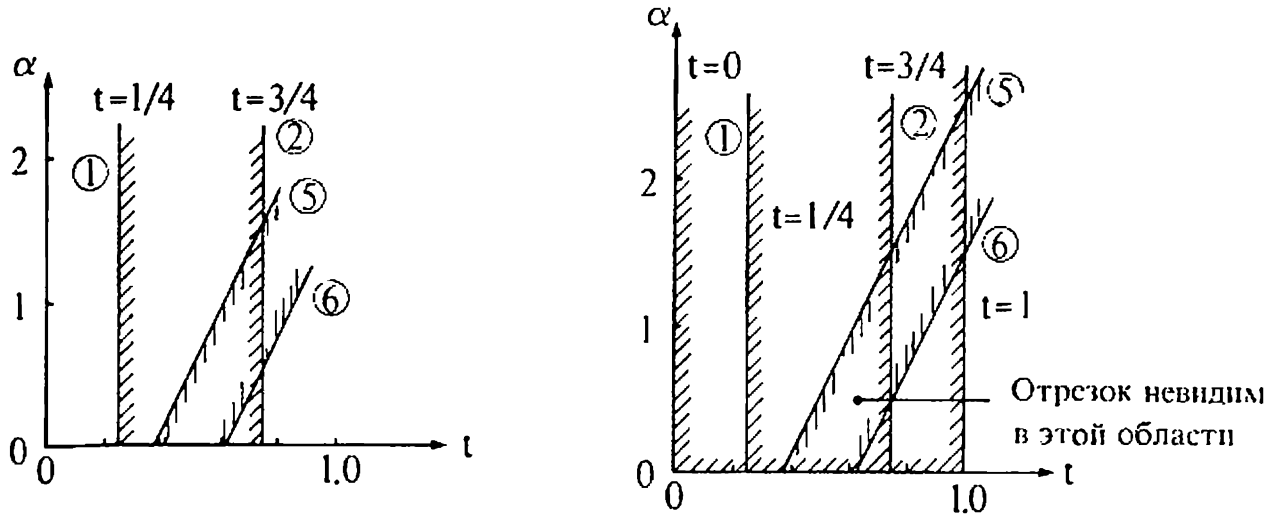



Рис. 4.25. Графическое решение примера 4.11.

Один из приемов заключается в запоминании всех точек протыкания и в добавлении к сцене отрезков, связывающих эти точки. Отрезки образуются путем соединения каждой точки протыкания для пары тел, связанных отношением протыкания, со всеми остальными точками протыкания для этой пары объектов. Затем проверяется экранирование этих отрезков данными телами. Видимые отрезки образуют структуру протыкания.

Примеры показывают, что решения, удовлетворяющие неравенствам $h_j > 0$, могут существовать и за пределами области, ограниченной условиями $0 \leqslant t \leqslant 1$ и $\alpha = 0$. Поэтому три уравнения, описывающие эти границы, т. е. $t = 0$, $t - 1 = 0$ и $\alpha = 0$, нужно добавить к множеству уравнений $h_j = 0$. Теперь число решений равно $(j + 2)(j + 3)/2$, где $j$ — количество плоскостей, ограничивающих выпуклый объем тела.

Как упоминалось ранее, выбор максимального из минимальных и минимального из максимальных значений $t$ среди возможных корректных решений указанной системы уравнений является простой задачей линейного программирования. Ее решение эквивалентно определению корректной ограниченной области, получающейся в результате графического решения, примеры которого были даны на рис. 4.21, 4.23 и 4.25. Блок-схема, приведенная на рис. 4.26, содержит алгоритм решения указанной минимаксной задачи. Предполагается, что этот алгоритм используется только для таких отрезков, о которых известно, что они частично или полностью невидимы. Все нелицевые и все полностью видимые отрезки выявлены и удалены до начала работы алгоритма. Алгоритм начинает работу с такими значениями $t$ и $\alpha$, которые являются решениями пары ли-

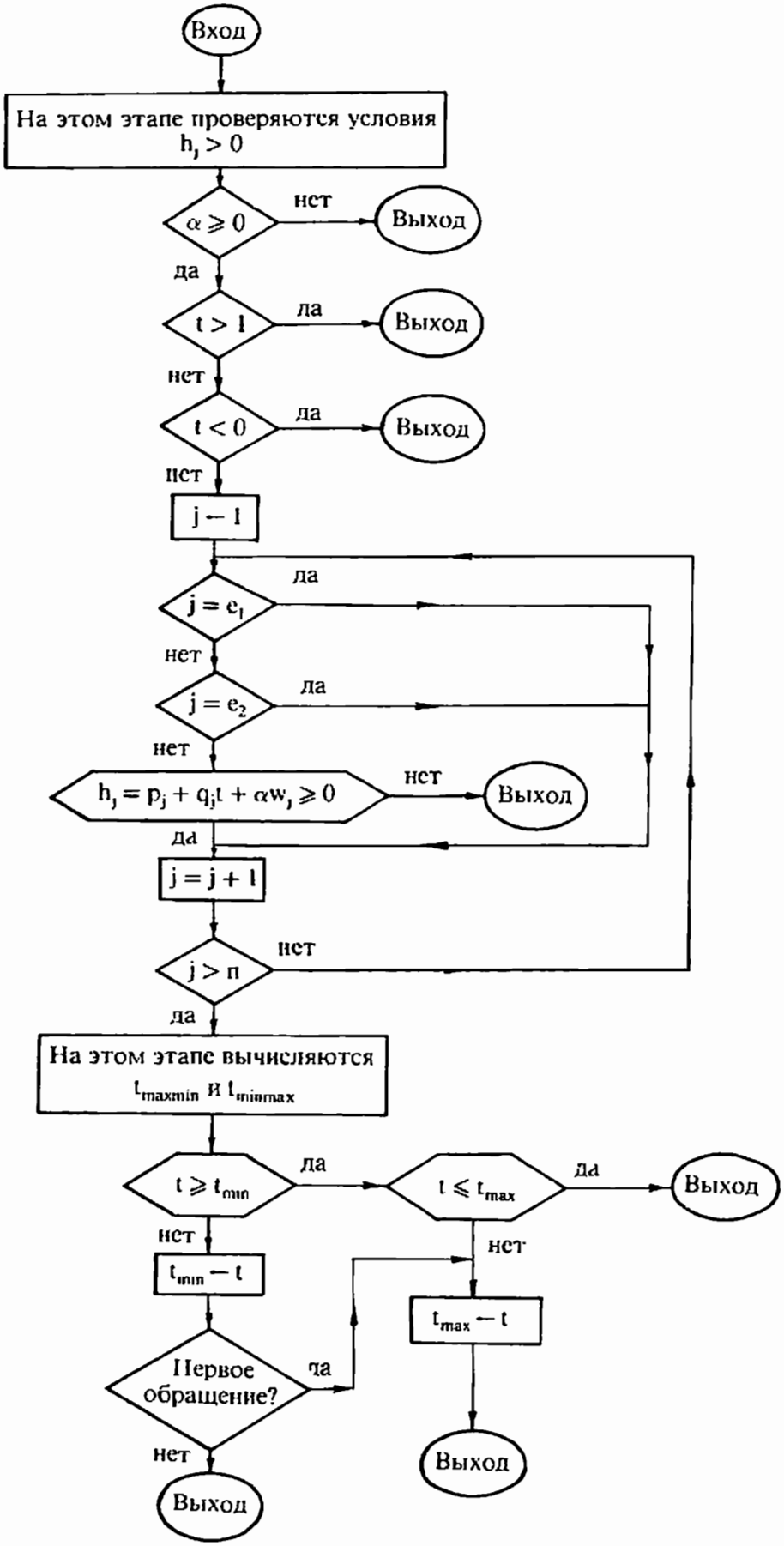



Рис. 4.26. Алгоритм нахождения $t_{maxmin}$ и $t_{minmax}$ в методе Робертса.

нейных уравнений с номерами $e_1$ и $e_2$, а также с $t_{min}$ и $t_{max}$ (текущими минимальным и максимальным значениями $t$) и с $n$ (мощностью множества уравнений). На первом этапе алгоритма проверяется выполнение условий $h_j > 0$. Если эти условия выполнены, то на втором этапе вычисляются значения $t_{min}$ и $t_{max}$. Результатом являются значения $t_{maxmin}$ и $t_{minmax}$.

Метод решения, обсуждавшийся выше, требует больших затрат машинного времени. Поэтому стоит поискать более быстрые способы определения полностью видимых отрезков. Основная идея состоит в установлении того факта, что оба конца отрезка лежат между точкой наблюдения и какой-нибудь видимой плоскостью. Напомним, что

$$\mathbf{u} = \mathbf{s} + t\mathbf{d} + \alpha\mathbf{g}$$

При $\alpha = 0$ значение $\mathbf{u}$ задает сам отрезок. Далее, если $\alpha = 0$, то при $t = 0$ и $t = 1$ получаются концевые точки отрезка. Напомним также, что

$$h_j = \mathbf{u} \cdot [VГ] = p_j + q_j t + w_j \alpha$$

и заметим, что при $t = 0$ $p_j$ является скалярным произведением концевой точки отрезка и $j$-й плоскости, ограничивающей тело. Аналогично $p_j + q_j$ является скалярным произведением другой концевой точки отрезка и $j$-й плоскости, ограничивающей тело. Наконец, напомним, что $j$-я плоскость, ограничивающая тело, видима, если $w_j \leqslant 0$. Поэтому, если $w_j \leqslant 0$ и $p_j \leqslant 0$, то один конец отрезка лежит или на видимой плоскости или между видимой плоскостью и точкой наблюдения. Если же $p_j + q_j \leqslant 0$, то другой конец отрезка также лежит либо на видимой плоскости, либо между этой плоскостью и точкой наблюдения. Следовательно, отрезок полностью видим, если для любого $j$

$$w_j \leqslant 0 \text{ и } p_j \leqslant 0 \text{ и } p_j + q_j \leqslant 0.$$

Эти условия гарантируют, что неравенства $h_j \leqslant 0$ не могут быть выполнены ни при каких $\alpha \geqslant 0$ и $0 \leqslant t \leqslant 1$. Поэтому никакая часть отрезка не может быть невидимой, т. е. отрезок полностью видим.

**Пример 4.12.** Полностью видимые отрезки

Для единичного куба с центром в начале координат рассмотрим отрезок от $P_1[-2\ 0\ 2\ 1]$ до $P_2[2\ 0\ 2\ 1]$, который, как показано на рис. 4.27, проходит перед этим кубом. Здесь

$$\mathbf{v} = \mathbf{s} + \mathbf{d}t = [-2\ \ 0\ \ 2\ \ 1] + [4\ \ 0\ \ 0\ \ 0]t$$

и с учетом того, что точка наблюдения лежит в бесконечности на положительной полуоси $z$, имеем:

$$\mathbf{s} = [-2 \quad 0 \quad 2 \quad 1]$$
$$\mathbf{d} = [\ 4 \quad 0 \quad 0 \quad 0]$$
$$\mathbf{g} = [\ 0 \quad 0 \quad 1 \quad 0]$$

Для непреобразованного куба $[VT] = [V]$ и

$$\begin{array}{rcccccc} & ① & ② & ③ & ④ & ⑤ & ⑥ \\ p = \mathbf{s}\cdot[VT] = & [\ 5 & -3 & 1 & 1 & -3 & 5] \\ q = \mathbf{d}\cdot[VT] = & [-8 & 8 & 0 & 0 & 0 & 0] \\ w = \mathbf{g}\cdot[VT] = & [\ 0 & 0 & 0 & 0 & -2 & 2] \end{array}$$

Заметим, что $w_5 < 0$, $p_5 < 0$ и $p_5 + q_5 < 0$. Значит, этот отрезок полностью видим.

Дополнительно в качестве примера рассмотрим отрезок от $P_3[-1\ 1\ 1\ 1]$ до $P_4[1\ 1\ -1\ 1]$, который проходит по диагонали над кубом. Этот отрезок тоже показан на рис. 4.27. Здесь

$$\mathbf{s} = [-1 \quad 1 \quad 1 \quad 1]$$
$$\mathbf{d} = [\ 2 \quad 0 \quad -2 \quad 0]$$
$$\mathbf{g} = [\ 0 \quad 0 \quad 1 \quad 0]$$

$$\begin{array}{rcccccc} & ① & ② & ③ & ④ & ⑤ & ⑥ \\ p = \mathbf{s}\cdot[VT] = & [\ 3 & -1 & -1 & 3 & -1 & 3] \\ q = \mathbf{d}\cdot[VT] = & [-4 & 4 & 0 & 0 & 4 & -4] \\ w = \mathbf{g}\cdot[VT] = & [\ 0 & 0 & 0 & 0 & -2 & 2] \end{array}$$

Заметим, что

$$w_5 < 0 \text{ и } p_5 < 0, \text{ но } p_5 + q_5 > 0.$$

Но $w_3 = 0$ и $p_3 < 0$ и $p_3 + q_3 < 0$. Этот отрезок также полностью видим.

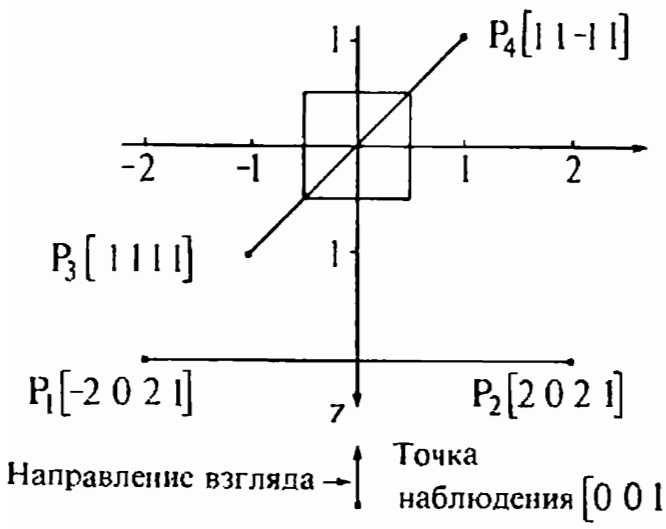



Рис. 4.27. Полностью видимые отрезки.

Хотя верхняя плоскость (плоскость 3) и лежит «под ребром» при взгляде из точки наблюдения, расположенной в бесконечности на оси $z$, математически отрезок $P_3P_4$ в приведенном выше примере расположен между этой точкой и указанной видимой плоскостью. Аналогичное условие справедливо для нижней и двух боковых плоскостей.

К сожалению, нет простого теста для полностью невидимых линий. Конечно, можно определить, что обе концевые точки отрезка расположены позади невидимой плоскости. Однако, поскольку плоскость простирается до бесконечности, невозможно определить, будут ли при этом концы отрезка позади тела (рис. 4.22). Полностью невидимые отрезки приходится обнаруживать, используя общий метод решения этой задачи. В этом случае невидимый участок будет простираться от $t = 0$ до $t = 1$.

Ниже приводится эффективная реализация алгоритма Робертса. Этот алгоритм делится на три этапа. На первом этапе каждое тело анализируется индивидуально с целью удаления нелицевых плоскостей. На втором этапе проверяется экранирование оставшихся в каждом теле ребер всеми другими телами с целью обнаружения их невидимых отрезков. На третьем этапе вычисляются отрезки, которые образуют новые ребра при протыкании телами друг друга. В данном алгоритме предполагается, что тела состоят из плоских полигональных граней, которые в свою очередь состоят из ребер, а ребра — из отдельных вершин. Все вершины, ребра и грани связаны с конкретным телом.

Удаление нелицевых плоскостей

Для каждого тела в сцене:

Сформировать многоугольники граней и ребра, исходя из списка вершин тела.

Вычислить уравнение плоскости для каждой полигональной грани тела.

Проверить знак уравнения плоскости:

Взять любую точку внутри тела, например усреднив координаты его вершин.

Вычислить скалярное произведение уравнения плоскости и точки внутри тела.

Если это скалярное произведение $< 0$, то изменить знак уравнения этой плоскости.

Сформировать матрицу тела.

Умножить ее слева на матрицу, обратную матрице видового преобразования, включающего перспективу.

Вычислить и запомнить габариты прямоугольной объемлющей оболочки преобразованного объема: $x_{max}$, $x_{min}$, $y_{max}$, $y_{min}$.

Определить нелицевые плоскости:

Вычислить скалярное произведение пробной точки, лежащей в бесконечности, на преобразованную матрицу тела.

Если это скалярное произведение $< 0$, то плоскость невидима.

Удалить весь многоугольник, лежащий в этой плоскости. Это избавляет от необходимости отдельно рассматривать невидимые линии, образуемые пересечением пар невидимых плоскостей.

Удаление из каждого тела тех ребер, которые экранируются всеми остальными телами в сцене:

Если задано только одно тело, то алгоритм завершается.

Сформировать приоритетный список этих тел:

Провести сортировку по $z$. Сортировка производится по максимальным значениям координаты $z$ вершин преобразованных тел. Первым в упорядоченном списке и обладающим наибольшим приоритетом будет то тело, у которого минимальное среди максимальных значений $z$. В используемой правой системе координат это тело будет самым удаленным от точки наблюдения, расположенной в бесконечности на оси $z$.

Для каждого тела из приоритетного списка:

Проверить экранирование всех лицевых ребер всеми другими телами сцены. Тело, ребра которого проверяются, называется пробным объектом, а тело, относительно которого в настоящий момент производится проверка, называется пробным телом[1]. Естественно, что нужно проверять экранирование пробного объекта только теми пробными телами, у которых ниже приоритеты.

---

[1] В отечественной литературе пробное тело принято называть экраном. — *Прим. перев.*

Провести проверки экранирования для прямоугольных объемлющих оболочек пробного объекта и пробного тела:

Если $x_{min}$ (пробное тело) > $x_{max}$ (пробный объект) или
$x_{max}$ (пробное тело) < $x_{min}$ (пробный объект) или
$y_{min}$ (пробное тело) > $y_{max}$ (пробный объект) или
$y_{max}$ (пробное тело) < $x_{min}$ (пробный объект),
то пробное тело не может экранировать ни одного ребра пробного объекта. Перейти к следующему пробному телу. В противном случае:

Провести предварительные проверки протыкания, чтобы увидеть, не протыкается ли пробное тело пробным объектом и существует ли возможность частичного экранирования первого последним.

Сравнить максимальное значение $z$ у пробного объекта с минимальным значением $z$ у пробного тела.

Если $z_{max}$ (пробный объект) < $z_{min}$ (пробное тело), то протыкание невозможно. Перейти к следующему телу. В противном случае:

Проверить видимое протыкание.

Если $z_{max}$ (пробный объект) > $z_{max}$ (пробное тело), то пробный объект может проткнуть переднюю грань пробного тела.

Установить флаг видимого протыкания для последующего использования. Занести проткнутое тело в список протыканий.

Если $x_{max}$ (пробный объект) > $x_{min}$ (пробное тело) или
$x_{min}$ (пробный объект) < $x_{max}$ (пробное тело),
то пробный объект может проткнуть бок пробного тела.

Установить флаг видимого протыкания для последующего использования. Занести тело в список протыканий.

Если $y_{max}$ (пробный объект) > $y_{min}$ (пробное тело) или
$y_{min}$ (пробный объект) < $y_{max}$ (пробное тело),
то пробный объект может проткнуть верх или низ пробного тела.

Установить флаг видимого протыкания для последу-

ющего использования. Занести проткнутое тело в список протыканий.

Если список протыканий пуст, то устанавливать флаг протыкания не надо.

Провести проверки экранирования ребер:

Вычислить **s** и **d** для ребра.
Вычислить $p$, $q$, $w$ для каждой плоскости, несущей грань пробного тела.

Проверка полной видимости. Если ребро полностью видимо, то перейти к следующему ребру.

Сформировать уравнения $h_j = 0$ и решить их, объединяя попарно и включив в систему уравнения границ $t = 0$ и $t = 1$. Если установлен флаг видимого протыкания, то в систему надо включить и уравнение границы $\alpha = 0$. Запомнить точки протыкания. В противном случае границу $\alpha = 0$ не учитывать.

Для каждой пары $(t, \alpha)$, являющейся решением, проверить выполнение условий $0 \leqslant t \leqslant 1$, $\alpha \geqslant 0$ и $h_j > 0$ для всех других плоскостей. Если эти условия выполнены, то найти $t_{\max\min}$ и $t_{\min\max}$.

Вычислить видимые участки отрезков и сохранить их для последующей проверки экранирования телами с более низкими приоритетами.

Определить видимые отрезки, связывающие точки протыкания:

Если флаг видимого протыкания не установлен, перейти к процедуре визуализации.

Если точек протыкания не обнаружено, перейти к процедуре визуализации.

Сформировать все возможные ребра, соединяющие точки протыкания, для пар тел, связанных отношением протыкания.

Проверить экранирование всех соединяющих ребер обоими телами, связанными отношением протыкания.

Проверить экранирование оставшихся соединяющих ребер всеми прочими телами сцены. Запомнить видимые отрезки.

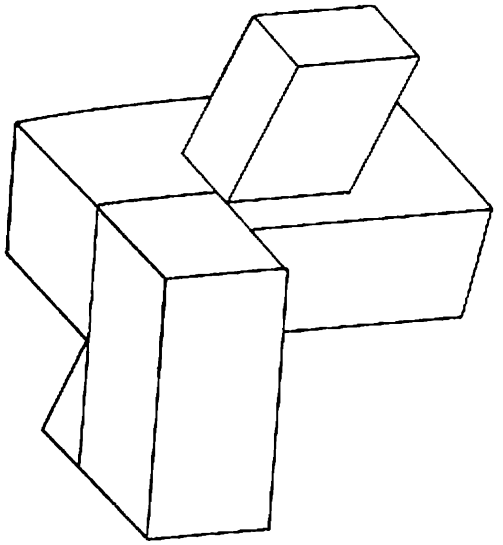


Рис. 4.28. Невидимые отрезки удалены из диметрической проекции пересекающихся объектов.

Визуализировать оставшиеся видимые отрезки ребер.

Заметим, что этот алгоритм можно реализовать и с обратным списком приоритетов. С помощью изложенного алгоритма была построена диметрическая проекция трех объектов, показанная на рис. 4.28.

Оценки по времени для сцен, аналогичных той, что показана на рис. 4.29, продемонстрировали почти линейно зависящий от числа брусков рост, причем последнее число достигало 1152 [4-9]. Петти и Мач [4-8] отметили аналогичный результат для алгоритма Робертса, реализованного с использованием метода разбиения кар-

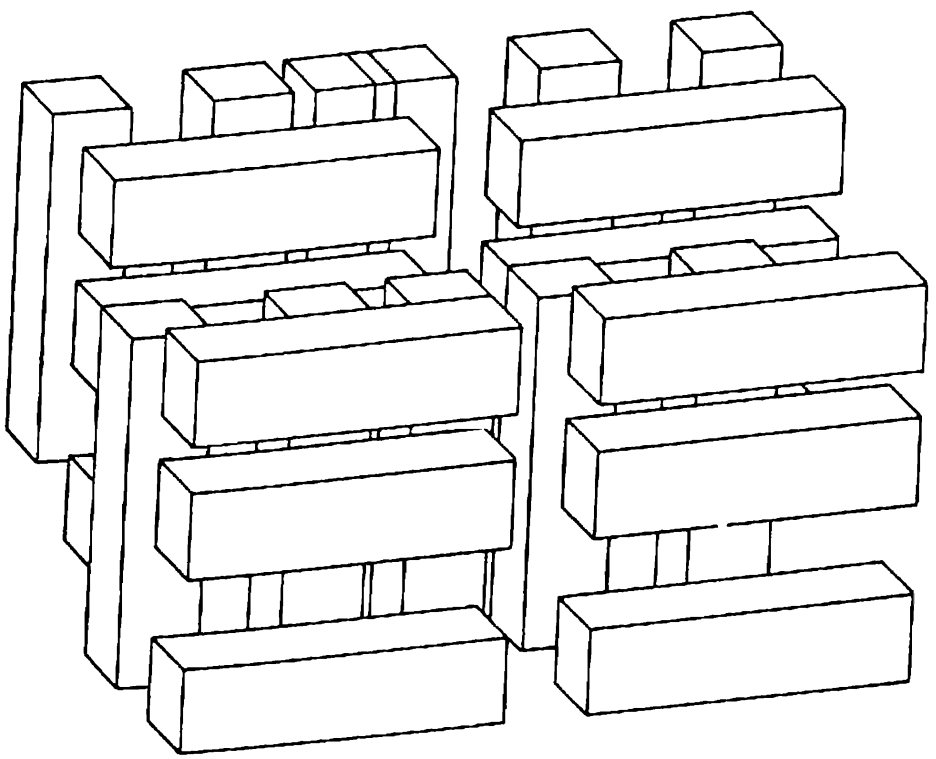


Рис. 4.29. Пробная сцена для алгоритма Робертса.

тинной плоскости в стиле Варнока (см. разд. 4.4). Принципиальным ограничением алгоритма Робертса является требование выпуклости объемов. Ниже приводится подробный пример.

**Пример 4.13.** Полный алгоритм Робертса

Рассмотрим пару пересекающихся брусков, показанную на рис. 4.30. Эти бруски описываются следующими таблицами координат вершин:

Брусок 1

| Номер вершины | x | y | z |
|---|---|---|---|
| 1 | 0 | 0 | 1 |
| 2 | 2 | 0 | 1 |
| 3 | 2 | 0 | 3 |
| 4 | 0 | 0 | 3 |
| 5 | 0 | 6 | 1 |
| 6 | 2 | 6 | 1 |
| 7 | 2 | 6 | 3 |
| 8 | 0 | 6 | 3 |

Брусок 2

| Номер вершины | x | y | z |
|---|---|---|---|
| 9 | 1 | 2 | 0 |
| 10 | 3 | 2 | 0 |
| 11 | 3 | 2 | 4 |
| 12 | 1 | 2 | 4 |
| 13 | 1 | 4 | 0 |
| 14 | 3 | 4 | 0 |
| 15 | 3 | 4 | 4 |
| 16 | 1 | 4 | 4 |

Номера вершин показаны на рис. 4.30, а. Списки ребер таковы:

Брусок 1

| Ребро | Связанные им вершины |
|---|---|
| 1 | 1–2 |
| 2 | 2–3 |
| 3 | 3–4 |
| 4 | 4–1 |
| 5 | 5–6 |
| 6 | 6–7 |
| 7 | 7–8 |
| 8 | 8–5 |
| 9 | 1–5 |
| 10 | 2–6 |
| 11 | 3–7 |
| 12 | 4–8 |

Брусок 2

| Ребро | Связанные им вершины |
|---|---|
| 13 | 9–10 |
| 14 | 10–11 |
| 15 | 11–12 |
| 16 | 12–9 |
| 17 | 13–14 |
| 18 | 14–15 |
| 19 | 15–16 |
| 20 | 16–13 |
| 21 | 9–13 |
| 22 | 10–14 |
| 23 | 11–15 |
| 24 | 12–16 |

Эти ребра образуют полигональные грани брусков:

Брусок 1

| Номер многоугольника | Ребра |
|---|---|
| 1 | 2, 11, 6, 10 |
| 2 | 4, 12, 8, 9 |
| 3 | 5, 6, 7, 8 |

Брусок 2

| Номер многоугольника | Ребра |
|---|---|
| 7 | 14, 23, 18, 22 |
| 8 | 21, 20, 24, 16 |
| 9 | 17, 18, 19, 20 |

| Брусок 1 | | Брусок 2 | |
|---|---|---|---|
| Номер многоугольника | Ребра | Номер многоугольника | Ребра |
| 4 | 1, 2, 3, 4 | 10 | 13, 14, 15, 16 |
| 5 | 3, 12, 7, 11 | 11 | 15, 24, 19, 23 |
| 6 | 1, 10, 5, 9 | 12 | 13, 22, 17, 21 |

Теперь можно сформировать матрицы тел для этих брусков в заданной ориентации, проверить корректность их знаков, взяв какую-нибудь точку внутри, а затем преобразовать, умножив слева на матрицу, обратную матрице видового преобразования. Однако в данном примере используется другой подход, заключающийся в том, что сначала преобразуется матрица вершин тела путем умножения ее справа на матрицу видового преобразования, а затем определяются уравнения преобразованных плоскостей и, следовательно, преобразованная матрица тела.

Здесь применяется видовое преобразование, состоящее из поворота на $-30°$ вокруг оси $v(\phi = -30°)$, за которым следует поворот на 15° вокруг оси $x(\theta = 15°)$. Результирующее преобразование есть [1-1]:

$$[T] = [R_x][R_v] = \begin{bmatrix} \cos\phi & \sin\phi\ \sin\theta & -\sin\phi\ \cos\theta & 0 \\ 0 & \cos\theta & \sin\theta & 0 \\ \sin\phi & -\cos\phi\ \sin\theta & \cos\phi\ \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 0.866 & -0.129 & 0.483 & 0 \\ 0.0 & 0.966 & 0.259 & 0 \\ -0.5 & -0.224 & 0.837 & 0 \\ 0.0 & 0.0 & 0.0 & 1 \end{bmatrix}$$

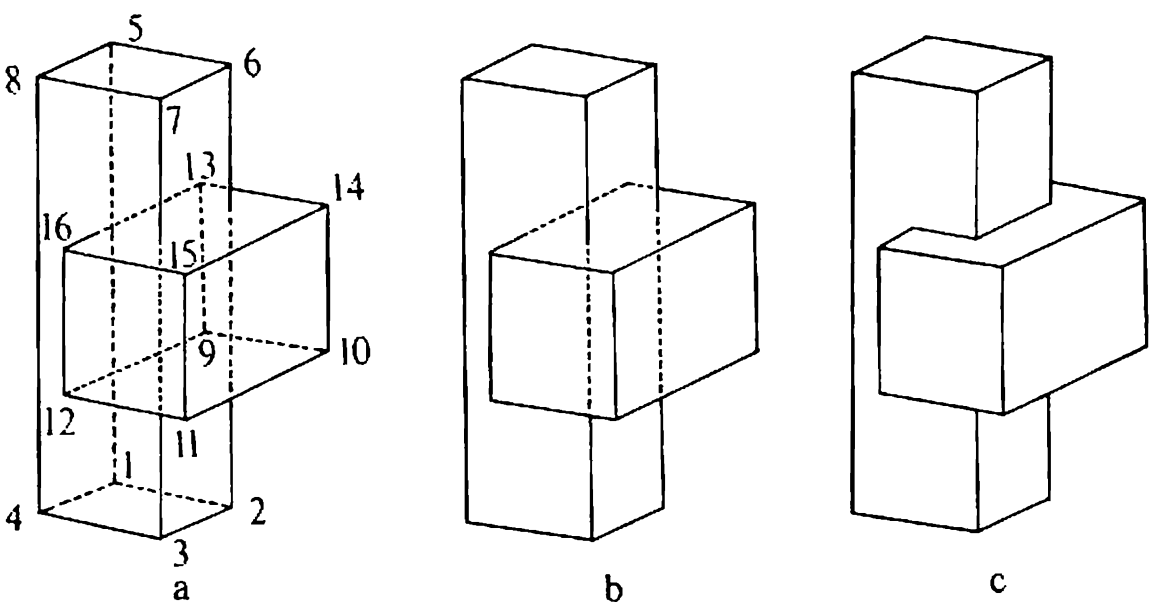



Рис. 4.30. Пересекающиеся бруски.

После преобразования матрицы вершины становятся такими:

$$[PT_1] = [P_1][T] = \begin{bmatrix} -0.5 & -0.224 & 0.837 & 1 \\ 1.232 & -0.483 & 1.802 & 1 \\ 0.232 & -0.933 & 3.475 & 1 \\ -1.5 & -0.672 & 2.510 & 1 \\ -0.5 & 5.571 & 2.389 & 1 \\ 1.232 & 5.313 & 3.355 & 1 \\ 0.232 & 4.864 & 5.028 & 1 \\ -1.5 & 5.123 & 4.062 & 1 \end{bmatrix} \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \end{matrix}$$

$$[PT_2] = [P_2][T] = \begin{bmatrix} 0.866 & 1.802 & 1.001 & 1 \\ 2.598 & 1.544 & 1.967 & 1 \\ 0.598 & 0.647 & 5.313 & 1 \\ -1.134 & 0.906 & 4.347 & 1 \\ 0.866 & 3.734 & 1.518 & 1 \\ 2.598 & 3.475 & 2.484 & 1 \\ 0.598 & 2.579 & 5.830 & 1 \\ -1.134 & 2.838 & 4.864 & 1 \end{bmatrix} \begin{matrix} 9 \\ 10 \\ 11 \\ 12 \\ 13 \\ 14 \\ 15 \\ 16 \end{matrix}$$

Уравнения плоскостей для каждой из граней этих двух брусков при данной ориентации можно получить методом Ньюэла, который был описан выше. Например, грань, образованная многоугольником 1, содержит четыре вершины с номерами 2, 3, 7 и 6 (рис. 4.30, а). Метод Ньюэла (см. пример 4.3), примененный к преобразованным точкам, дает уравнение плоскости:

$$-20.791x + 3.106y - 11.593z + 48.001 = 0$$

Перепишем этот результат так, чтобы он совпадал с результатом, получаемым преобразованием матрицы тела из исходного положения:

$$-0.866x + 0.129y - 0.483z + 2 = 0$$

Так преобразованная матрица тела имеет вид:

$$[VT_1] = \begin{array}{c} \begin{matrix} ① & ② & ③ & ④ & ⑤ & ⑥ \end{matrix} \\ \begin{bmatrix} -0.866 & 0.866 & 0 & 0 & 0.5 & -0.5 \\ 0.129 & -0.129 & -0.966 & 0.966 & 0.224 & -0.224 \\ -0.483 & 0.483 & -0.259 & 0.259 & -0.837 & 0.837 \\ 2 & 0 & 6 & 0 & 3 & -1 \end{bmatrix} \end{array}$$

и аналогично:

$$[VT_2] = \begin{array}{c} \begin{matrix} ⑦ & ⑧ & ⑨ & ⑩ & ⑪ & ⑫ \end{matrix} \\ \begin{bmatrix} -0.866 & 0.866 & 0 & 0 & 0.5 & -0.5 \\ 0.129 & -0.129 & -0.966 & 0.966 & 0.224 & -0.224 \\ -0.483 & 0.483 & -0.259 & 0.259 & -0.837 & 0.837 \\ 3 & -1 & 4 & -2 & 4 & 0 \end{bmatrix} \end{array}$$

Поскольку точка наблюдения равна [0 0 1 0], то пробная точка $[E] = [0\ 0\ -1\ 0]$. Поиск нелицевых плоскостей в первом теле дает:

$$[E]\cdot[VT_1] = \begin{matrix} \textcircled{1} & \textcircled{2} & \textcircled{3} & \textcircled{4} & \textcircled{5} & \textcircled{6} \\ [0.483 & -0.483 & 0.259 & -0.259 & 0.837 & -0.837] \end{matrix}$$

Аналогично, для второго тела:

$$[F]\cdot[VT_2] = \begin{matrix} \textcircled{7} & \textcircled{8} & \textcircled{9} & \textcircled{10} & \textcircled{11} & \textcircled{12} \\ [0.483 & -0.483 & 0.259 & -0.259 & 0.837 & -0.837] \end{matrix}$$

Отрицательные знаки показывают, что плоскости (многоугольники) 2, 4 и 6 в первом теле и 8, 10, 12 во втором — нелицевые. Пересечения этих многоугольников образуют невидимые ребра. В частности, ребра 1, 4 и 9 в первом теле и 13, 16 и 21 — во втором образованы пересечением пар нелицевых плоскостей и поэтому невидимы. Результат показан на рис. 4.30, b.

Оставшиеся линии в каждом теле проверяются на экранирование другим телом. Сначала проверяется возможность пересечения тел. Проверка протыкания тела 1 телом 2 с использованием преобразованных вершин показывает, что:

$$(z_{max})_{vol.1} = 5.028 > (z_{min})_{vol.2} = 1.001$$

Следовательно, протыкание возможно. Далее, поскольку

$$(x_{max})_{vol.1} = 1.232 > (x_{min})_{vol.2} = -1.134$$

то протыкание действительно имеет место. Поэтому уравнение границы $\alpha = 0$ нужно включить в систему.

Для оставшихся ребер тела 1 проводится проверка экранирования телом 2. В качестве примера рассмотрим ребро 2, соединяющее вершины 2 и 3. Здесь

$$\mathbf{v} = \mathbf{s} + \mathbf{d}t = [1.232\ \ -0.483\ \ 1.802\ \ 1] + [-1\ \ -0.45\ \ 1.673\ \ 0]t$$

Скалярные произведения $\mathbf{s}$ и $\mathbf{d}$ на $[VT]_2$ равны:

$$p = \mathbf{s}\cdot[VT_2] = \begin{matrix} \textcircled{1} & \textcircled{2} & \textcircled{3} & \textcircled{4} & \textcircled{5} & \textcircled{6} \\ [\ 1 & 1 & 4 & -2 & 3 & 1\ ] \end{matrix}$$

$$q = \mathbf{d}\cdot[VT_2] = \begin{matrix} \textcircled{1} & \textcircled{2} & \textcircled{3} & \textcircled{4} & \textcircled{5} & \textcircled{6} \\ [\ 0 & 0 & 0 & 0 & -2 & 3\ ] \end{matrix}$$

Поскольку точка наблюдения находится в бесконечности на положительной полуоси $z$, то $\mathbf{g} = [0\ 0\ 1\ 0]$, и

$$w = \mathbf{g}\cdot[VT_2] = \begin{matrix} \textcircled{1} & \textcircled{2} & \textcircled{3} & \textcircled{4} & \textcircled{5} & \textcircled{6} \\ [-0.483 & 0.483 & -0.259 & 0.259 & -0.837 & 0.837] \end{matrix}$$

Проверка полной видимости линий показывает, что условия:

$$w_j \leqslant 0,\ p_j \leqslant 0 \text{ и } p_i + q_j \leqslant 0$$

не выполняются ни для одной из плоскостей. Это происходит потому, что бесконечная плоскость, несущая основание (плоскость 4[1]) тела 2, может экранировать это ребро. Формирование условий невидимости ребра $h_j$ дает систему неравенств:

$$
\begin{aligned}
&① \quad 1 \qquad -0.483\alpha \geq 0\\
&② \quad 1 \qquad +0.483\alpha \geq 0\\
&③ \quad 4 \qquad -0.259\alpha \geq 0\\
&④ \quad -2 \qquad +0.259\alpha \geq 0\\
&⑤ \quad 3 \quad -2t - 0.837\alpha \geq 0\\
&⑥ \quad 1 + 2t + 0.837\alpha \geq 0
\end{aligned}
$$

Последовательное решение всех пар соответствующих уравнений показывает, что условия $h_j \geqslant 0$ не выполняются ни при каких $j$. Значит, ребро не имеет невидимого участка, и оно полностью видимо. Подробности, связанные с решениями вопросов об экранировании ребер тела 1 телом 2, даны в табл. 4.5 и 4.6. Заметим, что **g** и **w** неизменны.

Графические решения для ребер 10 и 11 показаны на рис. 4.31, a и b. Оба этих ребра протыкают тело 2. Ребро 10 невидимо при $0.244 < t < 0.667$. Этот участок соответствует отрезку от точки (1.232, 0.815, 2.150) до точки (1.232, 3.381, 2.837) Ребро 11 невидимо при $0.282 < t < 0.667$. Это соответствует отрезку между точками (0.232, 0.703, 3.913) и (0.232, 2.933, 4.510).

**Таблица 4.5.**

| Ребро | Соединяемые им вершины | **s** | | | | **d** | | | |
|---|---|---|---|---|---|---|---|---|---|
| 2 | 2–3 | [ 1.232 | −0.483 | 1.802 | 1] | [−1.0 | −0.45 | 1.673 | 0] |
| 3 | 3–4 | [ 0.232 | −0.931 | 3.46 | 1] | [−1.732 | 0.259 | −0.966 | 0] |
| 5 | 5–6 | [−0.5 | 5.571 | 2.389 | 1] | [ 1.732 | −0.259 | 0.966 | 0] |
| 6 | 6–7 | [ 1.232 | 5.313 | 3.355 | 1] | [−1.0 | −0.448 | 1.673 | 0] |
| 7 | 7–8 | [ 0.232 | 4.864 | 5.028 | 1] | [−1.732 | 0.259 | −0.966 | 0] |
| 8 | 8–5 | [−1.5 | 5.123 | 4.062 | 1] | [ 1.0 | 0.448 | −1.673 | 0] |
| 10 | 2–6 | [ 1.232 | −0.483 | 1.802 | 1] | [ 0.0 | 5.796 | 1.553 | 0] |
| 11 | 3–7 | [ 0.232 | −0.931 | 3.475 | 1] | [ 0.0 | 5.796 | 1.553 | 0] |
| 12 | 4–8 | [−1.5 | −0.672 | 2.510 | 1] | [ 0.0 | 5.796 | 1.553 | 0] |

[1] Многоугольник ей соответствующий, имеет номер 10 в общем списке. — *Прим. перев.*

**Таблица 4.6.**

| Ребро | Соединяемые им вершины | $p$ | $q$ | Примечание |
|---|---|---|---|---|
| 2 | 2–3 | [1 1 4 −2 3 1] | [ 0 0 0 0 −2 2] | Целиком видимо; решение получено длинным алгоритмом |
| 3 | 3–4 | [1 1 4 −2 1 3] | [ 2 −2 0 0 0 0] | ——— // ——— |
| 5 | 5–6 | [3 −1 −2 4 3 1] | [−2 2 0 0 0 0] | Целиком видимо; $w_3 < 0$, $p_3 < 0$, $p_3 + q_3 < 0$ |
| 6 | 6–7 | [1 1 −2 4 3 1] | [ 0 0 0 0 −2 2] | ——— // ——— |
| 7 | 7–8 | [1 1 −2 4 1 3] | [ 2 −2 0 0 0 0] | ——— // ——— |
| 8 | 8–5 | [3 −1 −2 4 1 3] | [ 0 0 0 0 2 −2] | ——— // ——— |
| 10 | 2–6 | [1 1 4 −2 3 1] | [ 0 0 −6 6 0 0] | Протыкание; невидимо при $0.244 < t < 0.667$; см. рис. 4.31, а |
| 11 | 3–7 | [1 1 4 −2 1 3] | [ 0 0 −6 6 0 0] | Протыкание; невидимо при $0.282 < t < 0.667$; см. рис. 4.31, b |
| 12 | 4–8 | [3 −1 4 −2 1 3] | [ 0 0 −6 6 0 0] | Целиком видимо; решение получено длинным алгоритмом |

Таблица 4.7.

| Ребро | Соединяемые им вершины | s | | | | d | | | |
|---|---|---|---|---|---|---|---|---|---|
| 14 | 10–11 | [ 2.598 | 1.544 | 1.967 | 1] | [−2.0 | −0.897 | 3.346 | 0] |
| 15 | 11–12 | [ 0.598 | 0.647 | 5.313 | 1] | [−1.732 | 0.259 | −0.966 | 0] |
| 17 | 13–14 | [ 0.866 | 3.734 | 1.518 | 1] | [ 1.732 | −0.259 | 0.966 | 0] |
| 18 | 14–15 | [ 2.598 | 3.475 | 2.484 | 1] | [−2.0 | −0.897 | 3.346 | 0] |
| 19 | 15–16 | [ 0.598 | 2.579 | 5.830 | 1] | [−1.732 | 0.259 | −0.966 | 0] |
| 20 | 16–13 | [−1.134 | 2.838 | 4.864 | 1] | [ 2.0 | 0.897 | −3.346 | 0] |
| 22 | 10–14 | [ 2.60 | 1.544 | 1.967 | 1] | [ 0 | 1.932 | 0.518 | 0] |
| 23 | 11–15 | [ 0.598 | 0.647 | 5.313 | 1] | [ 0 | 1.932 | 0.518 | 0] |
| 24 | 12–16 | [−1.134 | 0.906 | 4.347 | 1] | [ 0 | 1.932 | 0.518 | 0] |

Уравнение границы $\alpha = 0$ дает точки протыкания при $t = 0.333$ и 0.667 для обоих ребер. Эти значения $t$ соответствуют четырем точкам: (1.232, 1.449, 2.320) и (1.232, 3.381, 2.837) на ребре 10 и (0.232, 1.001, 3.993) и (0.232, 2.933, 4.510) на ребре 11, которые запоминаются в списке точек протыкания.

Проверка экранирования телом 1 лицевых ребер тела 2 дает результаты, приведенные в табл. 4.7 и 4.8. Ребро 17 частично экранируется телом 1. А именно, как показано на рис. 4.31, c, ребро 17 невидимо при $0 \leqslant t < 0.211$, что соответствует отрезку между точками (0.866, 3.734, 1.518) и (1.232, 3.679, 1.722). Ребро 20 протыкает переднюю грань (плоскость 5) тела 1 при $t = 0.25$. Поэтому она невидима от $0.25 < t \leqslant 1.0$, что соответствует отрезку от точки (−0.634, 3.062, 4.28) до (0.866, 3.734, 1.518). Область решения показана на рис. 4.31, d. Точка (−0.634, 3.062, 4.028) занесена в список точек протыкания. Графическое решение показывает, что точка при $t = 0.75$, $\alpha = 0$, также является точкой протыкания. Это значение $t$ соответствует (0.366, 3.511, 2.355).

Имеется шесть точек протыкания:

$$[PP] = \begin{bmatrix} 1.232 & 1.449 & 2.320 & 1 \\ 1.232 & 3.381 & 2.837 & 1 \\ 0.232 & 1.001 & 3.993 & 1 \\ 0.232 & 2.933 & 4.510 & 1 \\ -0.634 & 3.062 & 4.028 & 1 \\ 0.366 & 3.511 & 2.355 & 1 \end{bmatrix} \begin{matrix} ⑰ \\ ⑱ \\ ⑲ \\ ⑳ \\ ㉑ \\ ㉒ \end{matrix}$$

Связывая каждую из этих точек со всеми остальными, получаем 30 возможных соединяющих отрезков. Нужно проверить экранирование каждого из этих отрезков

**Таблица 4.8.**

| Ребро | Соединяемые им вершины | $p$ | $q$ | Примечание |
|---|---|---|---|---|
| 14 | 10–11 | [−1 3 4 2 3 −1] | [ 0 0 0 0 −4 4] | Целиком видимо; $w_1 < 0$, $p_1 < 0$, $p_1 + q_1 < 0$ |
| 15 | 11–12 | [−1 3 4 2 −1 3] | [ 2 −2 0 0 0 0] | Целиком видимо; $w_5 < 0$, $p_5 < 0$, $p_5 + q_5 < 0$ |
| 17 | 13–14 | [ 1 1 2 4 3 −1] | [−2 2 0 0 0 0] | Частично невидимо при $0 \leqslant t < 0.211$ |
| 18 | 14–15 | [−1 3 2 4 3 −1] | [ 0 0 0 0 −4 4] | Целиком видимо; $w_1 < 0$, $p_1 < 0$, $p_1 + q_1 < 0$ |
| 19 | 15–16 | [−1 3 2 4 −1 3] | [ 2 −2 0 0 0 0] | Целиком видимо, $w_5 < 0$, $p_5 < 0$, $p_5 + q_5 < 0$ |
| 20 | 16–13 | [ 1 1 2 4 −1 3] | [ 0 0 0 0 4 −4] | Прогыкание; невидимо при $0.25 < t \leqslant 1.0$ |
| 22 | 10–14 | [−1 3 4 2 3 −1] | [ 0 0 −2 2 0 0] | Целиком видимо; $w_1 < 0$, $p_1 < 0$, $p_1 + q_1 < 0$ |
| 23 | 11–15 | [−1 3 4 2 −1 3] | [ 0 0 −2 2 0 0] | ——— *и* ——— |
| 24 | 12–16 | [ 1 1 4 2 −1 3] | [ 0 0 −2 2 0 0] | Целиком видимо; $w_5 < 0$, $p_5 < 0$, $p_5 + q_5 < 0$ |

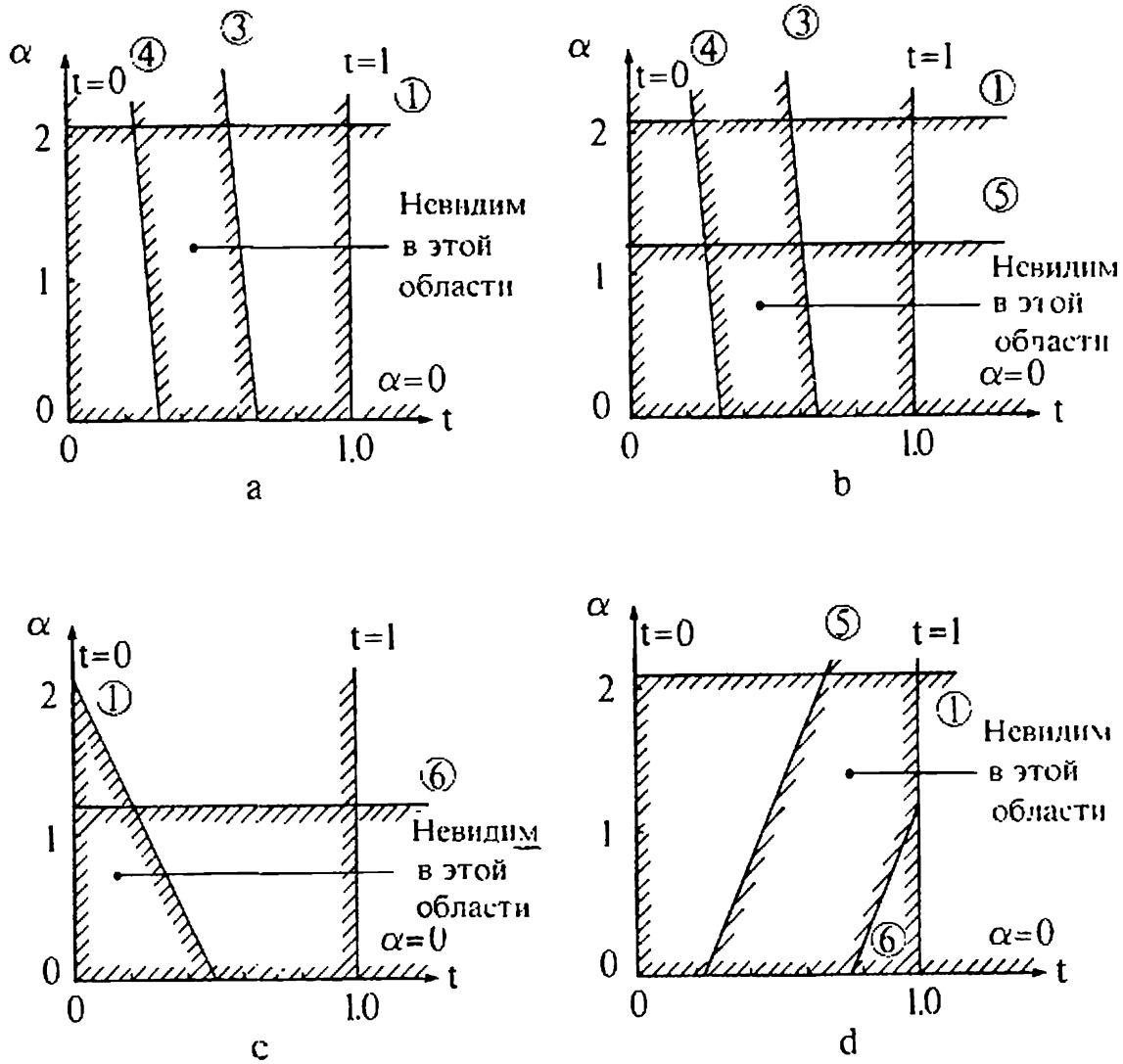



Рис. 4.31. Решение примера 4.13.

каждым телом. Подавляющее большинство их оказываются невидимыми. Проверка показывает, что потенциально видимы только отрезки, соединяющие точки 18 и 20, а также 20 и 21. Эти отрезки оказываются целиком видимыми. Фактически именно они и являются соединяющими отрезками. В качестве упражнения предлагается провести процесс решения целиком. Окончательный результат показан на рис. 4.30, с.

## 4.4. АЛГОРИТМ ВАРНОКА

Основные идеи, на которые опирается алгоритм Варнока, обладают большой общностью. Они основываются на гипотезе о способе обработки информации, содержащейся в сцене, глазом и мозгом человека. Эта гипотеза заключается в том, что тратится очень мало времени и усилий на обработку тех областей, которые содержат мало информации. Бóльшая часть времени и труда затрачивается на области с высоким информационным содержимым. В качестве примера рассмотрим поверхность стола, на которой нет ничего,

кроме вазы с фруктами. Для восприятия цвета, фактуры и других аналогичных характеристик всей поверхности стола много времени не нужно. Все внимание сосредоточивается на вазе с фруктами. В каком месте стола она расположена? Велика ли она? Из какого материала сделана: из дерева, керамики, пластика, стекла, металла? Каков цвет вазы: красный, синий, серебристый; тусклый или яркий и т. п.? Какие фрукты в ней лежат: персики, виноград, груши, бананы, яблоки? Каков цвет яблок: красный, желтый, зеленый? Есть ли у яблока хвостик? В каждом случае, по мере сужения сферы интереса, возрастает уровень требуемой детализации. Далее, если на определенном уровне детализации на конкретный вопрос нельзя ответить немедленно, то он откладывается на время для последующего рассмотрения. В алгоритме Варнока и его вариантах делается попытка извлечь преимущество из того факта, что большие области изображения однородны, например поверхность стола в приведенном выше примере. Такое свойство известно как когерентность, т. е. смежные области (пикселы) вдоль обеих осей $x$ и $y$ имеют тенденцию к однородности.

Поскольку алгоритм Варнока нацелен на обработку картинки, он работает в пространстве изображения. В пространстве изображения рассматривается окно и решается вопрос о том, пусто ли оно, или его содержимое достаточно просто для визуализации. Если это не так, то окно разбивается на фрагменты до тех пор, пока содержимое подокна не станет достаточно простым для визуализации или его размер не достигнет требуемого предела разрешения. В последнем случае информация, содержащаяся в окне, усредняется, и результат изображается с одинаковой интенсивностью или цветом. Устранение лестничного эффекта можно реализовать, доведя процесс разбиения до размеров, меньших, чем разрешение экрана на один пиксел, и усредняя атрибуты подпикселов, чтобы определить атрибуты самих пикселов (см. разд. 2.25).

Конкретная реализация алгоритма Варнока зависит от метода разбиения окна и от деталей критерия, используемого для того, чтобы решить, является ли содержимое окна достаточно простым. В оригинальной версии алгоритма Варнока [4-10, 4-11] каждое окно разбивалось на четыре одинаковых подокна. В данном разделе обсуждаются эта версия алгоритма, а также общий вариант, позволяющий разбивать окно на полигональные области. Другая разновидность алгоритма, предложенная Вейлером и Азертоном [4-12], в которой окно тоже разбивается на полигональные подокна, обсуждается в следующем разделе. Кэтмул [4-13, 4-14] применил основную

идею метода разбиения к визуализации криволинейных поверхностей. Этот метод обсуждается в разд. 4.6.

На рис. 4.32 показан результат простейшей реализации алгоритма Варнока. Здесь окно, содержимое которого слишком сложно изображать, разбито на четыре одинаковых подокна. Окно, в котором что-то есть, подразбивается далее то тех пор, пока не будет достигнут предел разрешения экрана. На рис. 4.32, a показана сцена, состоящая из двух простых многоугольников. На рис. 4.32, b показан результат после удаления невидимых линий. Заметим, что горизонтальный прямоугольник частично экранирован вертикальным. На рис. 4.32, c и d показан процесс разбиения окон на экране с разрешением 256 × 256. Поскольку $256 = 2^8$, требуется не более восьми шагов разбиения для достижения предела разрешения экрана. Пусть подокна рассматриваются в следующем порядке: нижнее левое, нижнее правое, верхнее левое, верхнее правое. Будем обозначать подокна цифрой и буквой, цифра — это номер шага разбиения, а буква — номер квадранта. Тогда для окна 1a подокна 2a, 4a, 4c, 5a, 5b оказываются пустыми и изображаются с фоновой интенсивностью в процессе разбиения. Первым подокном, содержимое которого не пусто на уровне пикселов, оказывается 8a. Теперь необходимо решить вопрос о том, какой алгоритм желательно применить: удаления невидимых линий или удаления невидимых поверхностей. Если желательно применить алгоритм удаления невидимых линий, то пиксел, соответствующий подокну 8a, активируется, поскольку через него проходят видимые ребра. В результате получается изображение видимых ребер многоугольников в виде последовательности точек размером с пиксел каждая (рис. 4.32, e).

Следующее рассмотрение окна, помеченного как 8d на рис. 4.32, d, лучше всего проиллюстрирует различие между реализациями алгоритмов удаления невидимых линий и поверхностей. В случае удаления невидимых линий окно 8d размером с пиксел не содержит ребер ни одного многоугольника сцены. Следовательно, оно объявляется пустым и изображается с фоновой интенсивностью или цветом. Для алгоритма удаления невидимых поверхностей проверяется охват этого окна каждым многоугольником сцены. Если такой охват обнаружен, то среди охватывающих пиксел многоугольников выбирается ближайший к точке наблюдения на направлении наблюдения, проходящем через данный пиксел. Проверка проводится относительно центра пиксела. Затем этот пиксел изображается с интенсивностью или цветом ближайшего многоугольника. Если охватывающие многоугольники не найдены, то окно размером с пиксел

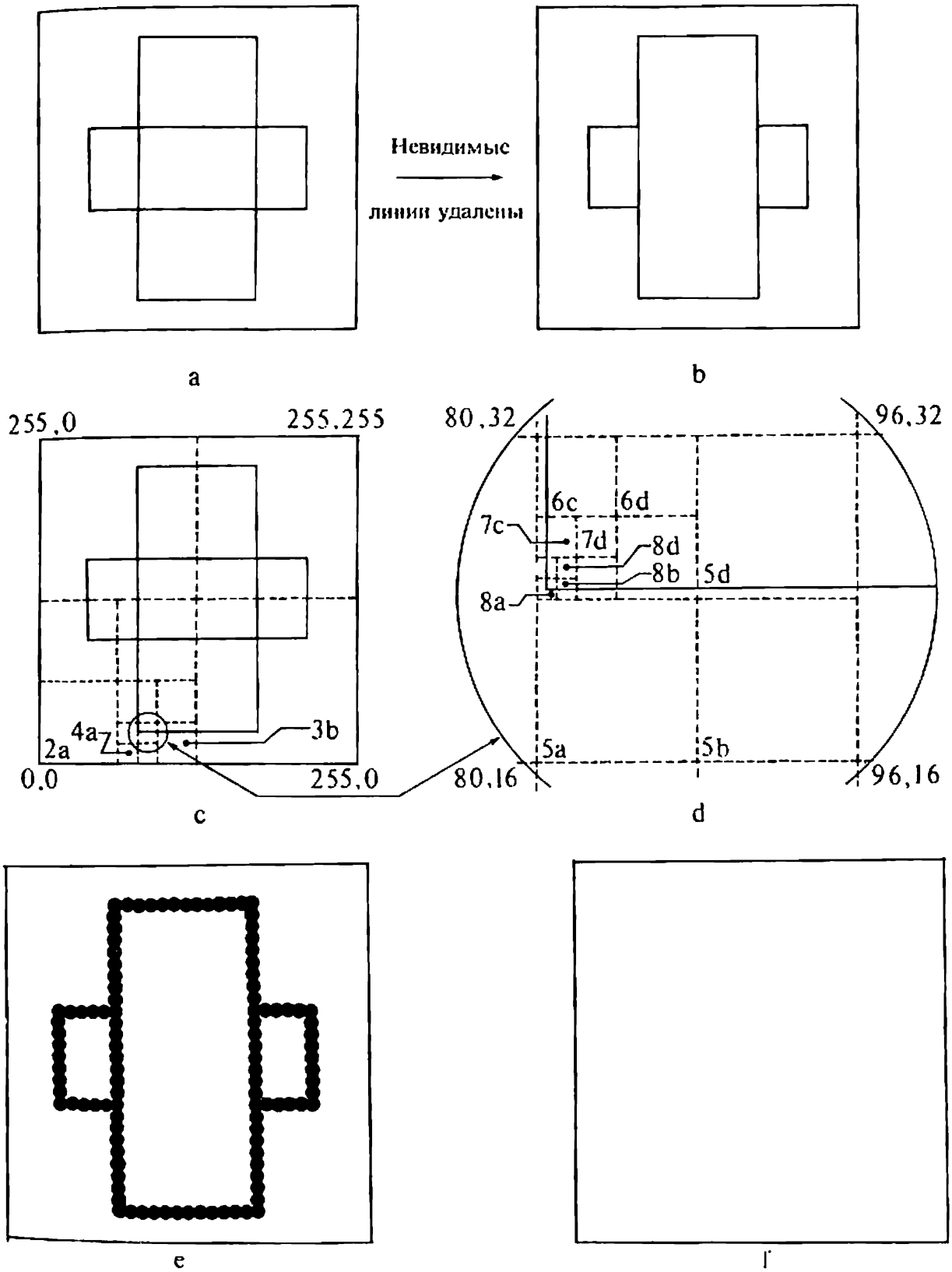



Рис. 4.32. Алгоритм разбиения Варнока.

пусто. Поэтому оно изображается с фоновым цветом или интенсивностью. Окно 8d охвачено вертикальным прямоугольником. Поэтому оно изображается с цветом или интенсивностью этого многоугольника. Соответствующий результат показан на рис. 4.32, f.

Возвратившись к рассмотрению окна 8a на рис. 4.32, d, покажем, как включить в алгоритм удаления невидимых поверхностей устранение лестничного эффекта. Разбиение этого окна дает четыре подокна с размерами, меньшими, чем размеры пиксела. Только одно из этих подокон — правое верхнее — охвачено многоугольником. Три других пусты. Усреднение результатов для этих четырех подпикселов (см. разд. 2.25) показывает, что окно 8a размером с пиксел нужно изображать с интенсивностью, равной одной четверти интенсивности прямоугольника. Аналогично пиксел 8b следует высвечивать с интенсивностью, равной половине интенсивности прямоугольника. Конечно, окна размером с пиксел можно разбивать более одного раза, чтобы произвести взвешенное усреднение характеристик подпикселов, которое обсуждалось в разд. 2.25.

Процесс подразбиения порождает для подокон структуру данных, являющуюся деревом, которое показано на рис. 4.33[1]. Корнем этого дерева является визуализируемое окно. Каждый узел изображен прямоугольником, содержащим координаты левого нижнего угла и длину стороны подокна. Предположим, что подокна обрабатываются в следующем порядке: *abcd*, т. е. слева направо на каждом уровне разбиения в дереве. На рис. 4.33 показан активный путь по структуре данных дерева к окну 8a размером с пиксел. Активные узлы на каждом уровне изображены толстой линией. Рассмотрение рис. 4.32 и 4.33 показывает, что на каждом уровне все окна слева от активного узла пусты. Поэтому они должны были визуализироваться ранее с фоновым значением цвета или интенсивности. Все окна справа от активного узла на каждом уровне будут обработаны позднее, т. е. будут объявлены пустыми или будут подразделены при обходе дерева в обратном порядке.

При помощи изложенного алгоритма можно удалить либо невидимые линии, либо невидимые поверхности. Однако простота критерия разбиения, а также негибкость способа разбиения приводят к тому, что количество подразбиений оказывается велико. Можно повысить эффективность этого алгоритма, усложнив способ и критерий разбиения. На рис. 4.34, a показан другой общий способ разбиения и дано его сравнение с изложенным ранее жестким способом, представленным на рис. 4.34, b.

---

[1] В алгоритме Варнока впервые была реализована такая структура данных, как кватернарное дерево.

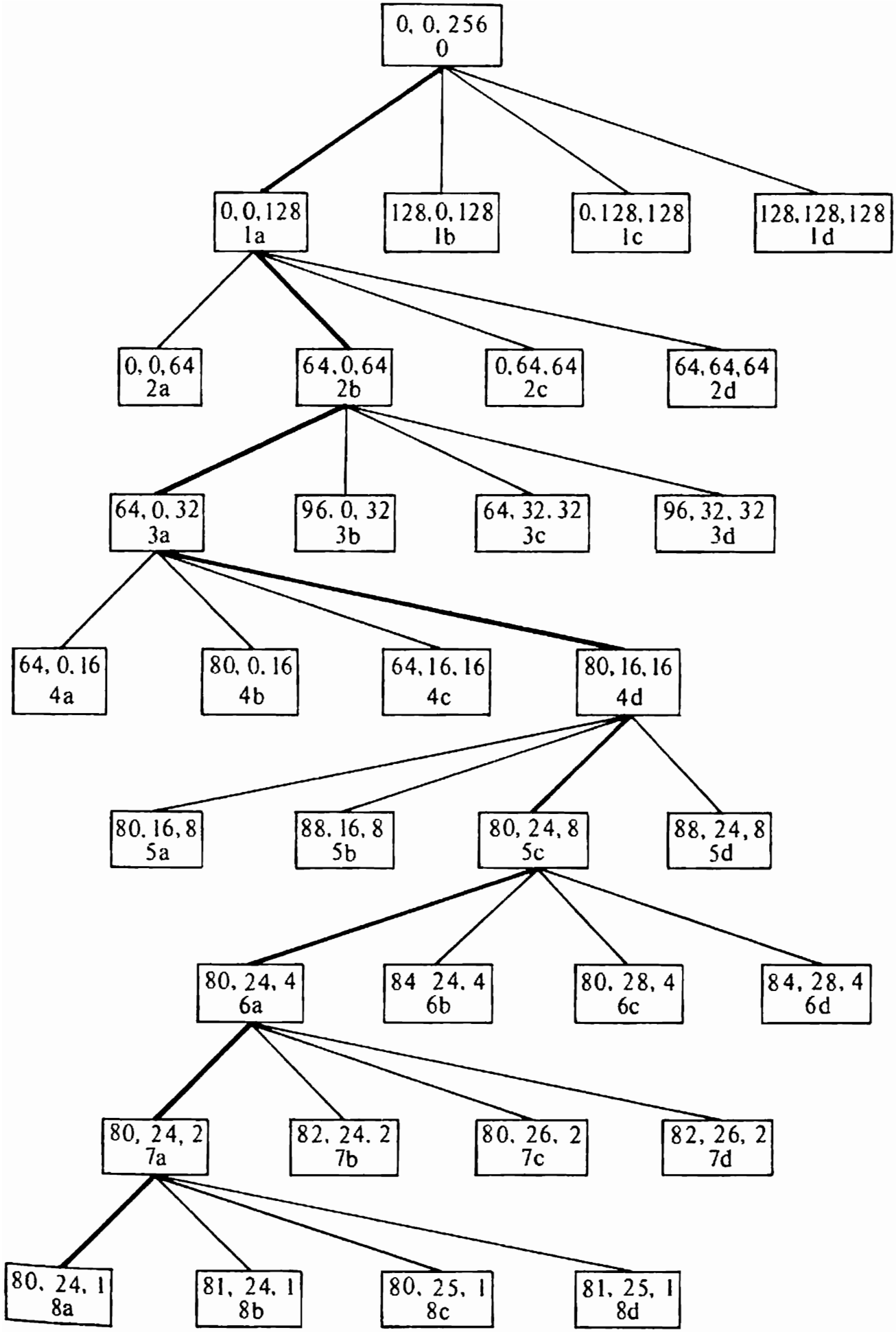



Рис. 4.33. Дерево структуры подокон.

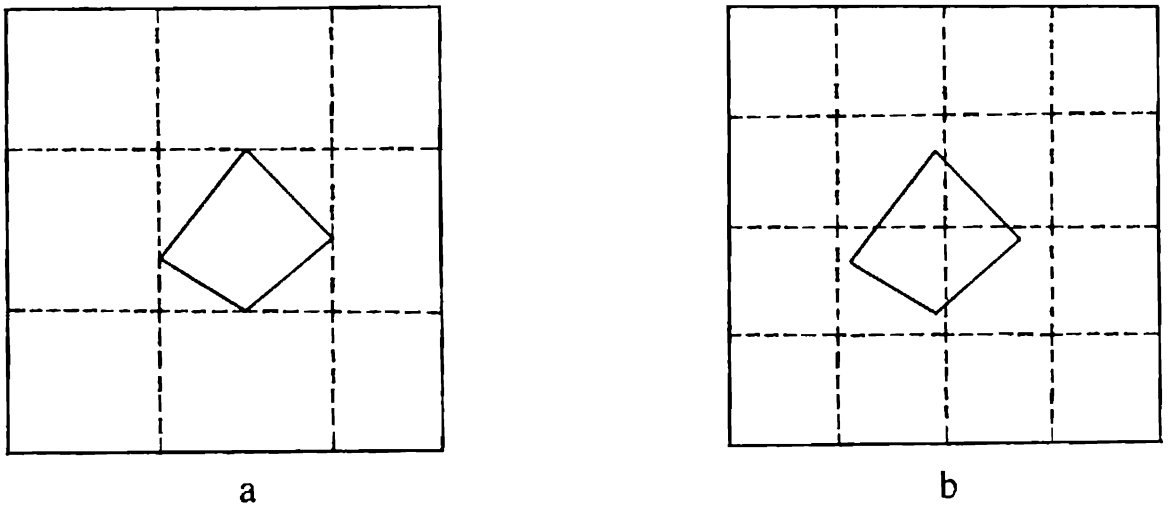



Рис. 4.34. Сравнение двух способов разбиения.

Разбиение, показанное на рис. 4.34, а, получается с использованием прямоугольной объемлющей оболочки многоугольника. Заметим, что подокна при этом могут быть неквадратными. Этот способ можно рекурсивно применить к любому многоугольнику, который полностью охвачен каким-нибудь окном или подокном. Если в окне содержится только один многоугольник и если он целиком охвачен этим окном, то его легко изобразить, не проводя дальнейшего разбиения. Такой способ разбиения полезен, в частности, при минимизации числа разбиений для простых сцен (рис. 4.34). Однако с ростом сложности сцены его преимущество сходит на нет.

Для рассмотрения более сложных критериев разбиения будет полезно определить способы расположения многоугольников относительно окна. Назовем многоугольник:

*внешним*, если он находится целиком вне окна;
*внутренним*, если он находится целиком внутри окна;
*пересекающим*, если он пересекает границу окна;
*охватывающим*, если окно находится целиком внутри него.

На рис. 4.35 показаны примеры всех этих типов многоугольников. Используя эти определения, можно предложить следующие правила обработки окна. Объединим их в алгоритм.

Для каждого окна:

Если все многоугольники сцены являются внешними по отношению к окну, то это окно пусто. Оно изображается с фоновой интенсивностью или цветом без дальнейшего разбиения.

Если внутри окна находится только один многоугольник, то площадь окна вне этого многоугольника заполняется фоновым значе-

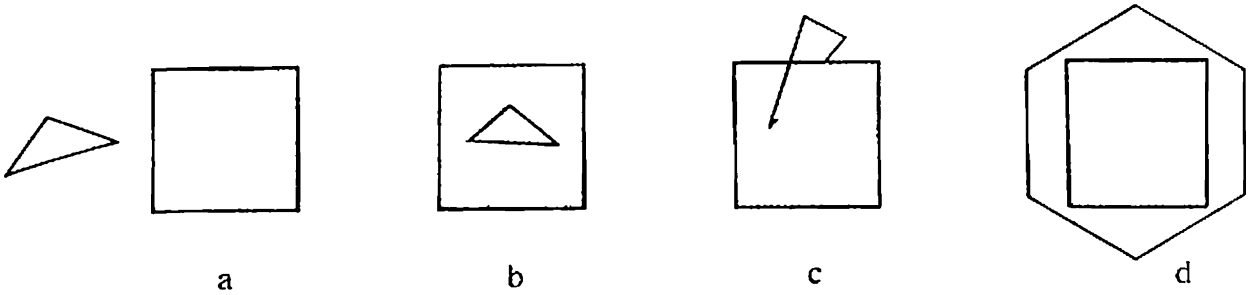



Рис. 4.35. Типы многоугольников: внешний (a), внутренний (b), пересекающий (c), охватывающий (d).

нием интенсивности или цвета, а сам многоугольник заполняется соответствующим ему значением интенсивности или цвета.

Если только один многоугольник пересекает окно, то площадь окна вне многоугольника заполняется фоновым значением интенсивности или цвета, а та часть пересекающего многоугольника, которая попала внутрь окна, заполняется соответствующей ему интенсивностью или цветом.

Если окно охвачено только одним многоугольником и если в окне нет других многоугольников, то окно заполняется значением интенсивности или цвета, которое соответствует охватывающему многоугольнику.

Если найден по крайней мере один охватывающий окно многоугольник и если этот многоугольник расположен ближе других к точке наблюдения, то окно заполняется значением интенсивности или цвета, которое соответствует охватывающему многоугольнику.

В противном случае проводится разбиение окна.

Первые четыре из этих критериев касаются ситуаций, в которых участвуют один прямоугольник и окно. Они используются для сокращения числа подразбиений. Пятое правило является ключом к решению задачи удаления невидимых поверхностей. В нем речь идет о поиске такого охватывающего многоугольника, который находился бы ближе к точке наблюдения, чем все остальные многоугольники, связанные с этим окном. Очевидно, что этот многоугольник будет закрывать или экранировать все остальные многоугольники, связанные с окном. Поэтому в сцене будут фигурировать именно его визуальные характеристики.

Для реализации этих правил требуются методы определения способа расположения многоугольника относительно окна. В случае прямоугольных окон можно воспользоваться минимаксными

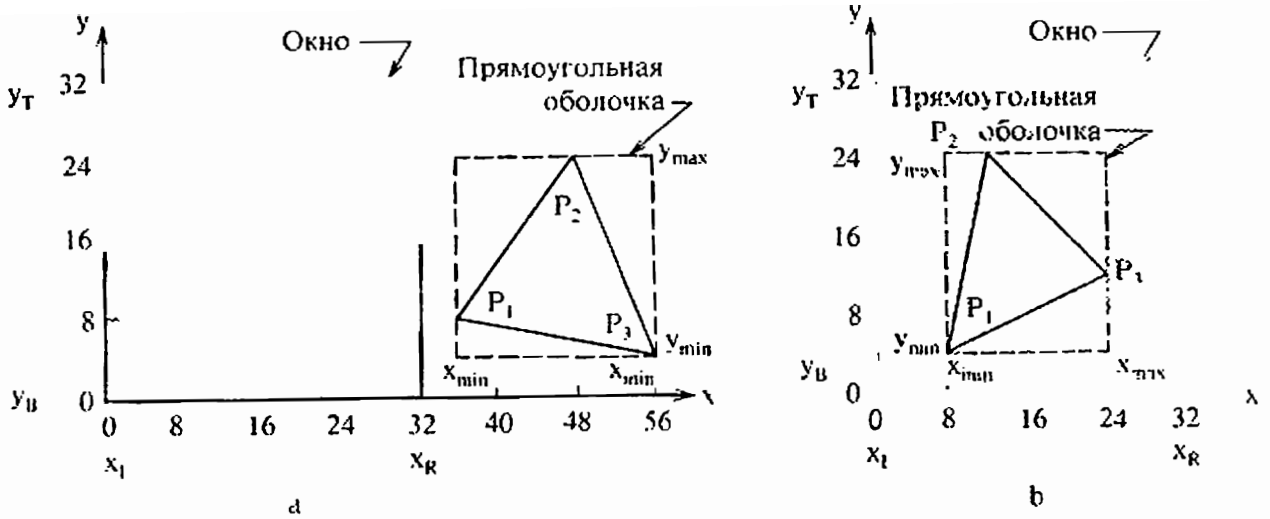



Рис. 4.36. Проверка с помощью прямоугольной оболочки для внешнего и внутреннего многоугольников.

или габаритными, с объемлющей прямоугольной оболочкой, тестами для определения, является ли многоугольник внешним по отношению к окну (см. разд. 2.13 и 3.1). В частности, если $x_Л$, $x_П$, $y_Н$, $y_В$ определяют четыре ребра окна, а $x_{min}$, $x_{max}$, $y_{min}$, $y_{max}$ — ребра прямоугольной объемлющей оболочки многоугольника, то этот многоугольник будет внешним по отношению к окну, если выполняется любое из следующих условий:

$$x_{min} > x_П,\ x_{max} < x_Л,\ y_{min} > y_В,\ y_{max} < y_Н$$

как показано на рис. 4.36, a. Кроме того, многоугольник является внутренним по отношению к окну, если его оболочка содержится внутри этого окна, т. е. если

$$x_{min} \geqslant x_Л \text{ и } x_{max} \leqslant x_П \text{ и } y_{min} \geqslant y_П \text{ и } y_{max} < y_В,$$

как показано на рис. 4.36, b.

**Пример 4.14.** Внутренний и внешний многоугольники

Рассмотрим квадратное окно, заданное значениями $x_Л$, $x_П$, $y_Н$, $y_В$, равными соответственно 0, 32, 0, 32. На рис. 4.36, a показаны два многоугольника, для которых нужно найти способ их расположения относительно окна. Первый из этих многоугольников задан вершинами $P_1$ (36, 8), $P_2$ (48, 24) и $P_3$ (56, 4), а второй — вершинами $P_1$ (8, 4), $P_2$ (12, 24) и $P_3$ (24, 12).

Прямоугольная оболочка для первого многоугольника определяется величинами $x_{min}$, $x_{max}$, $y_{min}$, $y_{max}$, равными соответственно 36, 56, 4, 24. Поскольку $(x_{min} = 36) > (x_П = 32)$, этот многоугольник является внешним по отношению к окну.

Аналогично прямоугольная оболочка для второго многоугольника определяется $x_{min}$, $x_{max}$, $y_{min}$, $y_{max}$, равными 8, 24, 4, 24, как показано на рис. 4.36, b. Здесь выполняются условия

$$(x_{min} = 8) > (x_Л = 0) \text{ и } (x_{max} = 24) < (x_П = 32) \text{ и}$$
$$(y_{min} = 4) > (y_Н = 0) \text{ и } (y_{max} = 24) < (y_В = 32)$$

Следовательно, этот многоугольник является внутренним по отношению к окну.

Можно воспользоваться простой подстановкой для проверки пересечения окна многоугольником. Координаты вершин окна подставляются в пробную функцию, заданную уравнением прямой, несущей ребро многоугольника (см. разд. 3.16 и пример 3.23). Если знак этой пробной функции не зависит от выбора вершины окна, то все его вершины лежат по одну сторону от несущей прямой и на указанной прямой нет точек пересечения. Если же эти знаки различны, то многоугольник пересекает окно. Если ни одно из ребер многоугольника не пересекает окна, то этот многоугольник либо является внешним по отношению к окну, либо охватывает его. Если уравнение прямой, проходящей через две вершины $P_1(x_1, y_1)$ и $P_2(x_2, y_2)$, имеет вид $y = mx + b$, то пробная функция (*T.F.* — test function) такова:

$$T.F. = y - mx - b$$

где

$$m = \frac{y_2 - y_1}{x_2 - x_1} \qquad x_2 - x_1 \neq 0$$

$$b = y_1 - mx_1$$

$$T.F. = x - x_1 \qquad x_2 - x_1 = 0$$

Продемонстрируем этот метод на примере.

**Пример 4.15.** Пересекающие многоугольники

Возьмем квадратное окно с $x_Л$, $x_П$, $y_Н$, $y_В$, равными соответственно 8, 32, 8, 32, и пару многоугольников, первый с вершинами $P_1$ (8, 4), $P_2$ (12, 24), и $P_3$ (40, 12) и второй — с вершинами $P_1$ (4, 4), $P_2$ (4, 36), $P_3$ (40, 36) и $P_4$ (32, 4). Они показаны на рис. 4.37. Пробная функция ребра $P_1P_2$ у многоугольника на рис. 4.37, а получается из следующих соотношений:

$$m = \frac{y_2 - y_1}{x_2 - x_1} = \frac{24 - 4}{12 - 8} = \frac{20}{4} = 5$$
$$b = y_1 - mx_1 = 4 - 5(8) = -36$$
$$T.F. = y - mx - b = y - 5x + 36$$

Подстановка координат каждой угловой точки окна в пробную функцию дает:

$$T.F.(8, 8) = 8 - 5(8) + 36 = 4$$
$$T.F.(8, 32) = 32 - 5(8) + 36 = 28$$

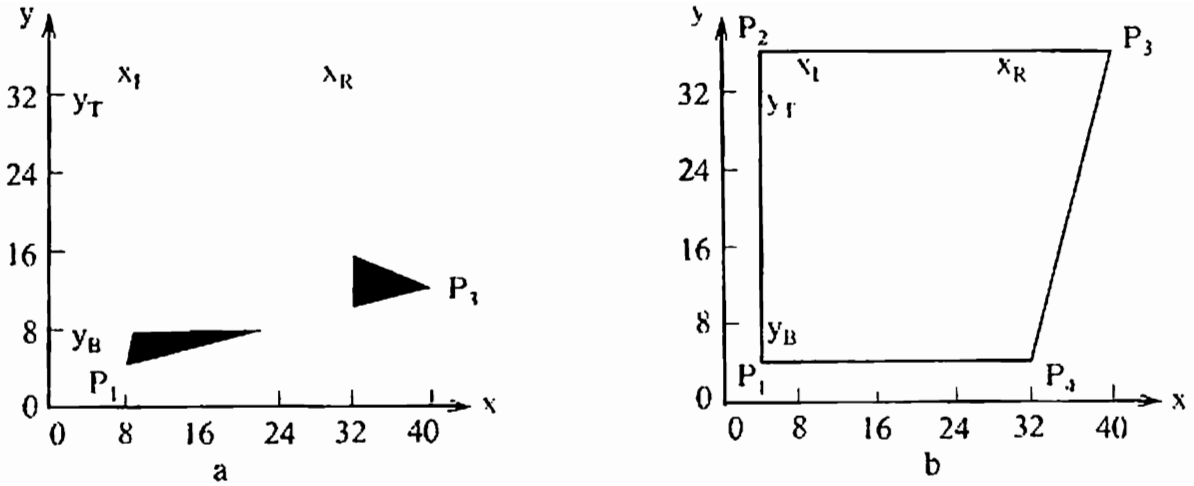



Рис. 4.37. Проверка пересечения.

**Таблица 4.9.**

| Ребро многоугольника | Пробная функция | Координаты окна | Результат подстановки | Примечание |
|---|---|---|---|---|
| $P_1P_2$ | $x - 4$ | (8, 8) | 4 | Нет пересечения |
| | | (8, 32) | 4 | |
| | | (32, 32) | 28 | |
| | | (32, 8) | 28 | |
| $P_2P_3$ | $y - 36$ | (8, 8) | −28 | Нет пересечения |
| | | (8, 32) | −4 | |
| | | (32, 32) | −4 | |
| | | (32, 8) | −28 | |
| $P_3P_4$ | $y - 4x + 124$ | (8, 8) | 100 | Нет пересечения |
| | | (8, 32) | 124 | |
| | | (32, 32) | 28 | |
| | | (32, 8) | 4 | |
| $P_4P_1$ | $y - 4$ | (8, 8) | 4 | Нет пересечения |
| | | (8, 32) | 28 | |
| | | (32, 32) | 28 | |
| | | (32, 8) | 4 | |

$$T.F.(32, 32) = 32 - 5(32) + 36 = -92$$
$$T.F.(32, 8) = 8 - 5(32) + 36 = -116$$

Поскольку знаки подстановок в пробную функцию различны, это ребро многоугольника пересекает окно, что и показано на рис. 4.37, a. Значит, заданный многоугольник является пересекающим. Проверять остальные ребра этого многоугольника нет необходимости.

Результаты проверки многоугольника, изображенного на рис. 4.37, b, сведены в табл. 4.9. Ни одно из ребер этого многоугольника не пересекает окна. Следовательно, этот многоугольник либо внешний, либо охватывающий[1]. На рис. 4.37 видно, что он охватывающий.

Простым габаритным тестом с прямоугольной оболочкой, который рассматривался выше, нельзя идентифицировать все виды внешних многоугольников. Примером может служить многоугольник, огибающий угол окна (рис. 4.38, a). Для этого нужны более сложные тесты. Особый интерес представляют тест с бесконечной прямой и тест с подсчетом угла. В обоих тестах предполагается, что внутренние и пересекающие многоугольники уже были идентифицированы. Оба теста могут быть использованы для определения внешних и охватывающих многоугольников.

В тесте с бесконечной прямой проводится луч из любой точки окна, например из угла в бесконечность. Подсчитывается число пересечений этого луча с заданным многоугольником. Если это число четное (или нуль), то многоугольник внешний; если же оно нечетное, то многоугольник охватывает окно, как показано на рис. 4.38, a. Если луч проходит через вершину многоугольника, как показано на рис. 4.38, b, то результат неопределен. Эта неопределенность устраняется, если считать касание за два пересечения ($P_2$ на рис. 4.38, b), а протыкание — за одно ($P_4$ на рис. 4.38, b) (см. также разд. 2.17). Изменение угла наклона луча тоже позволяет устранить неопределенность.

Тест с подсчетом угла проиллюстрирован на рис. 4.39. Совершим обход по ребрам многоугольника по или против часовой стрелки. При этом просуммируем углы, образованные лучами, начинающимися в любой точке окна и проходящими через начало и конец проходимого в данный момент ребра многоугольника. Как показано на рис. 4.39, удобной точкой является центр окна. Получаемая сумма углов интерпретируется следующим образом:

---

[1] Внутренние многоугольники к этому моменту уже определены. — *Прим. перев.*

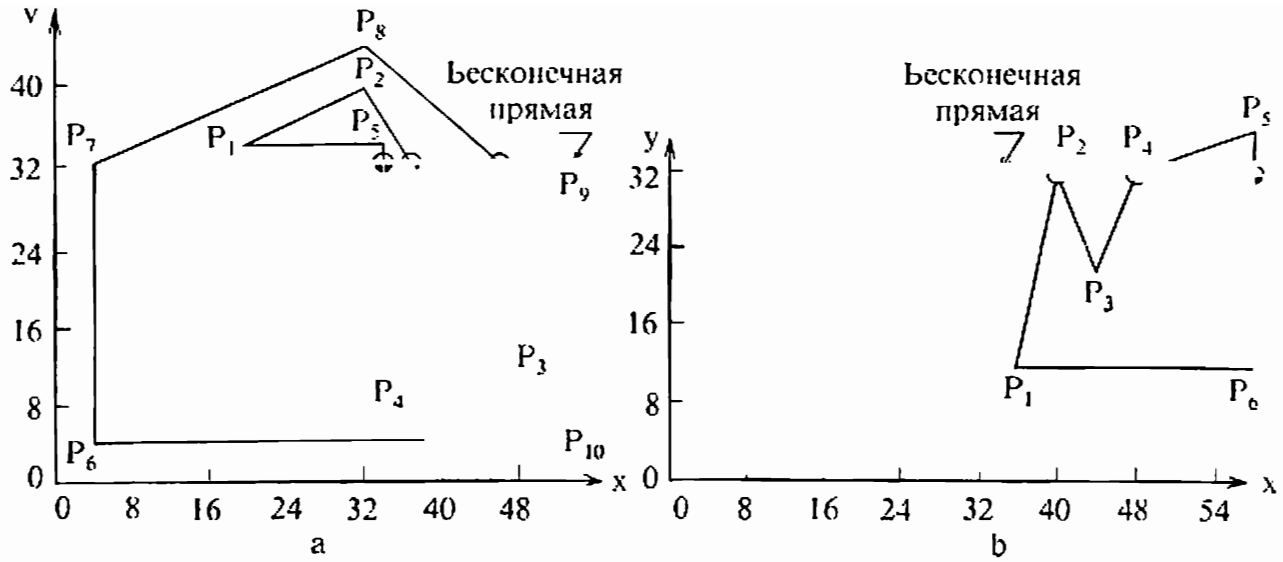



Рис. 4.38. Проверка охватывающего многоугольника.

Сумма = 0, многоугольник, внешний по отношению к окну.
Сумма = ± 360 $n$, многоугольник охватывает окно $n$ раз[1].

Практическое вычисление этой суммы значительно упрощается, если учесть, что точность вычисления каждого угла не обязана быть высокой. Фактически нужная точность получается, если считать только целые октанты (приращения по 45°), покрытые отдельными углами, как показано на рис. 4.40. Техника здесь напоминает ту, что использовалась для кодирования концов отрезка при

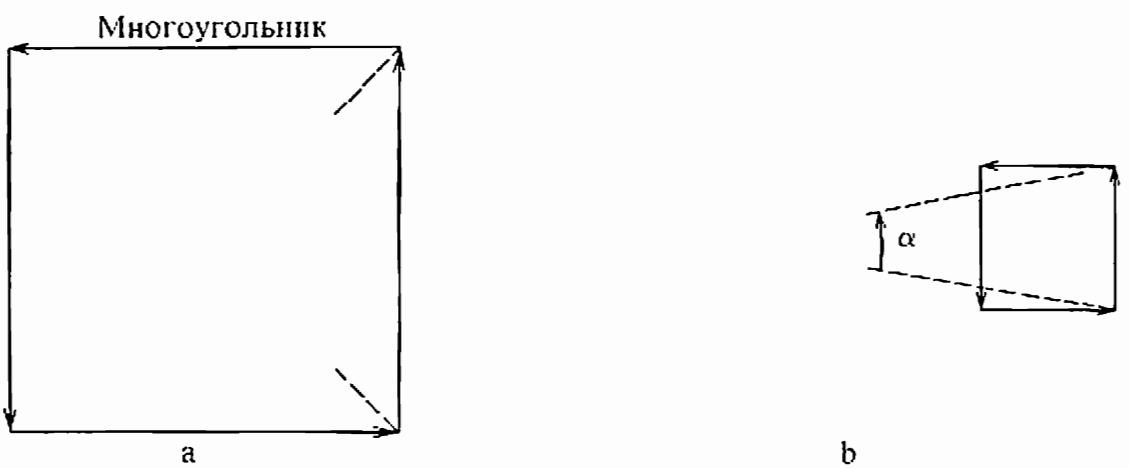



Рис. 4.39. Проверка с подсчетом угла.

[1] Простой многоугольник (не имеющий самопересечений) может охватить точку только один раз. — *Прим. перев.*

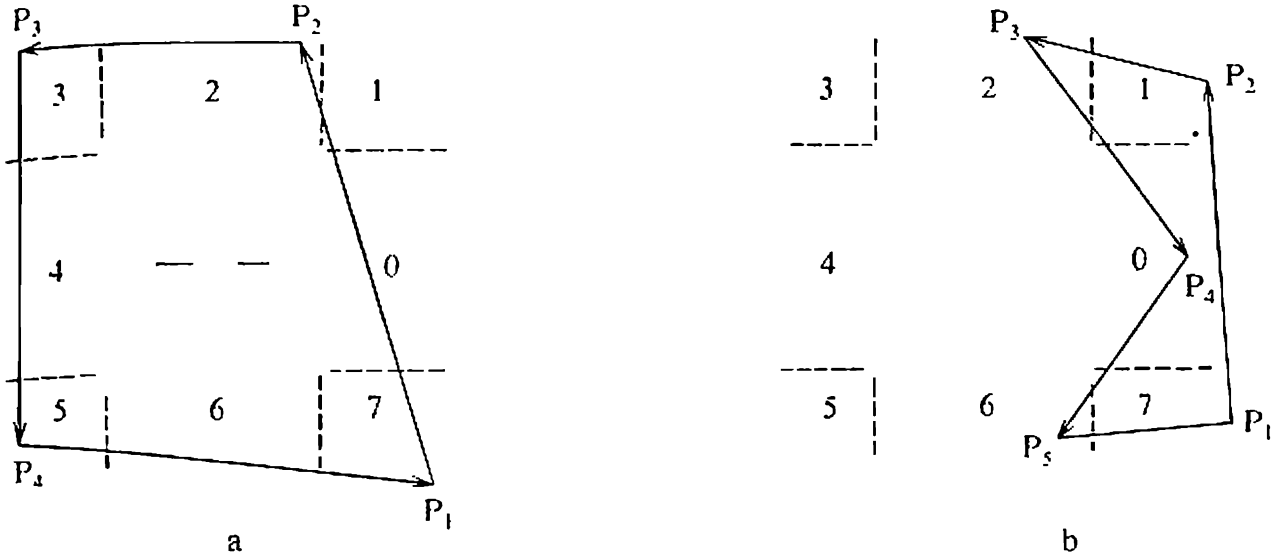



Рис. 4.40. Угловой тест для охватывающего и внешнего многоугольников.

отсечении (см. разд. 3.1). Пронумеруем октанты числами от 0 до 7 против часовой стрелки. Число целых октантов, покрытых углом, равно разности между номерами октантов, в которых лежат концы его ребер. Предлагается следующий алгоритм:

$\Delta\alpha$ = (номер октанта, в котором лежит второй конец ребра) − (номер октанта, в котором лежит первый конец ребра)

**if** $\Delta\alpha > 4$ **then** $\Delta\alpha = \Delta\alpha - 8$
**if** $\Delta\alpha < -4$ **then** $\Delta\alpha = \Delta\alpha + 8$
**if** $\Delta\alpha = 0$ **then** ребро многоугольника расщепляется ребром окна, и процесс повторяется с парой полученных отрезков.

Суммируя вклады от отдельных ребер, получаем:

$$\Sigma\Delta\alpha = \begin{cases} 0, & \text{многоугольник, внешний по отношению к окну,} \\ \pm 8\ n, & \text{многоугольник охватывает окно.} \end{cases}$$

**Пример 4.16.** Угловой тест для охватывающего и внешнего многоугольников
Рассмотрим окно и многоугольники, показанные на рис. 4.40. Для многоугольника с рис. 4.40, а получаем для ребра $P_1P_2$:

$$\Delta\alpha_{12} = 2 - 7 = -5 < -4$$
$$= -5 + 8 = 3$$

Аналогично, для других ребер многоугольника имеем:

$$\Delta\alpha_{23} = 3 - 2 = 1$$

$$\Delta\alpha_{34} = 5 - 3 = 2$$
$$\Delta\alpha_{41} = 7 - 5 = 2$$

Сумма всех углов, опирающихся на ребра этого многоугольника, равна:

$$\sum \Delta\alpha = 3 + 1 + 2 + 2 = 8$$

Поэтому данный многоугольник охватывает окно.

Для многоугольника, показанного на рис. 4.40, b, имеем:

$$\Delta\alpha_{12} = 1 - 7 = -6 < -4$$
$$= -6 + 8 = 2$$
$$\Delta\alpha_{23} = 2 - 1 = 1$$
$$\Delta\alpha_{34} = 0 - 2 = -2$$
$$\Delta\alpha_{45} = 6 - 0 = 6 > 4$$
$$= 6 - 8 = -2$$
$$\Delta\alpha_{51} = 7 - 6 = 1$$

$$\sum \Delta\alpha = 2 + 1 - 2 - 2 + 1 = 0$$

Поэтому данный многоугольник является внешним по отношению к окну.

Иерархическое использование методов, основанных на изложенных вычислительных приемах, дает дополнительный выигрыш. Если реализуется только простейший алгоритм Варнока, то нет смысла определять внутренние или пересекающие многоугольники. Разбиения будут постоянно превращать такие многоугольники во внешние или охватывающие. Все нестандартные ситуации разрешаются на уровне пикселов. В этом простом алгоритме для определения пустых окон достаточно пользоваться только тестом с прямоугольной объемлющей оболочкой. Если этот простой тест дает отрицательный результат, то алгоритм разбивает окно вплоть до достижения уровня пикселов. Поскольку внешний многоугольник в той форме, которая показана на рис. 4.40, b, может встретиться даже на уровне пикселов, то необходимо применить более мощный алгоритм для определения пустоты окна или охвата его одним или более многоугольниками.

Более сложный алгоритм, обсуждавшийся выше, делает попытку определить способ расположения многоугольника относительно окна бóльших размеров, чтобы избежать его подразбиения. Такие тесты требуют бóльших вычислительных затрат. Следовательно, существует зависимость между затратами, связанными с процессом разбиения окон, и затратами, связанными с ранним определением

окон, готовых к визуализации. Более сложный алгоритм должен реализовать тестирование каждого окна в следующем порядке.

Провести проверку при помощи простого теста с прямоугольной оболочкой для определения как можно большего числа пустых окон и окон, содержащих единственный внутренний многоугольник. Такие окна сразу же изображаются.

Провести проверку при помощи простого теста на пересечение для определения окон, пересекающих единственный многоугольник. Этот многоугольник отсекается и изображается. Например, многоугольник на рис. 4.34, b был бы изображен после первого же шага разбиения.

Провести проверку при помощи более сложных тестов для внешних и охватывающих многоугольников, чтобы определить дополнительные пустые окна и окна, охваченные единственным многоугольником. Такие окна сразу же изображаются.

После реализации этих шагов проводится разбиение окна или делается попытка обнаружить такой охватывающий многоугольник, который был бы ближе к точке наблюдения, чем любой другой многоугольник. Если проводится разбиение, то решение последнего вопроса откладывается до достижения уровня пикселов. В любом случае требуется проводить тест глубины.

Тест глубины проводится путем сравнения глубин (координат $z$) плоскостей, несущих многоугольники, в угловых точках окна. Если глубина охватывающего многоугольника больше, чем глубины всех остальных многоугольников в углах окна, то охватывающий многоугольник экранирует все остальные многоугольники в этом окне. Следовательно, данное окно можно визуализировать с тем значением интенсивности или цвета, которое соответствует охватывающему многоугольнику. Заметим, что приведенное условие по глубине является достаточным, но не необходимым для того, чтобы охватывающий многоугольник экранировал все остальные многоугольники в окне. На рис. 4.41 показано, что проведение теста глубины в углах окна для несущих грани плоскостей может привести к неудаче при определении охватывающего окно многоугольника, который экранирует все остальные многоугольники в этом окне.

В частности, если плоскость, несущая многоугольник, экранируется охватывающим многоугольником в углах окна, то сам многоугольник тоже экранируется последним в углах окна (как показано на рис. 4.41). Если же несущая плоскость не экранируется охватывающим окно многоугольником, то не очевидно, будет или нет

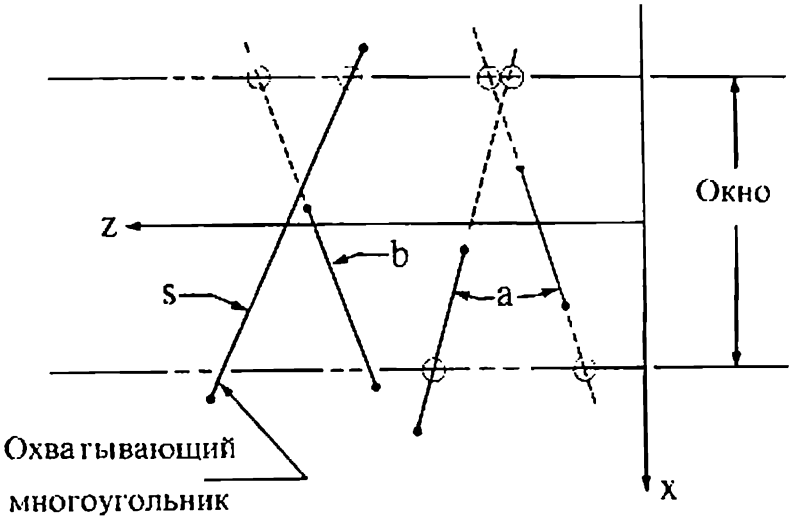



Рис. 4.41. Сравнение глубин для охватывающих многоугольников.

экранирован многоугольник, лежащий в этой плоскости (случай b на рис. 4.41). Этот конфликт разрешается путем разбиения окна

Глубину несущей плоскости в углу окна можно вычислить с помощью ее уравнения (см. разд. 4.3 и пример 4.3). Например, если уравнение этой плоскости имеет вид

$$ax + by + cz + d = 0$$

а координаты угловой точки окна равны $(x_w, y_w)$, то значение

$$z = -(d + ax_w + by_w)/c \qquad c \neq 0$$

равно искомой глубине.

Всюду выше предполагалось, что все многоугольники следует сравнивать с каждым окном. Для сложных сцен это весьма неэффективно. Эффективность можно повысить, проведя сортировку по приоритету глубины (сортировку по $z$). Сортировка проводится по значению $z$ координаты той вершины многоугольника, которая ближе других к точке наблюдения. В правой системе координат многоугольник с максимальным значением такой координаты $z$ будет ближайшим к точке наблюдения. В списке отсортированных многоугольников он появится первым.

При обработке каждого окна алгоритм ищет охватывающие его многоугольники. После обнаружения такого многоугольника запоминается величина $z$ у его самой удаленной от наблюдателя вершины $z_{smin}$. При рассмотрении каждого очередного многоугольника в списке сравнивается координата $z$ его ближайшей к наблюдателю вершины — $z_{pmax}$ с $z_{smin}$. Если $z_{pmax} < z_{smin}$, то очевидно, что очередной многоугольник экранируется охватывающим и не должен больше учитываться. Из рис. 4.41 видно, что это условие является достаточным, но не необходимым; например, многоугольники, по-

меченные буквой a на рис. 4.41, больше можно не учитывать, но многоугольник, помеченный буквой b, учитывать нужно.

Длину списка многоугольников, обрабатываемых для каждого окна, можно сократить, если воспользоваться информацией, полученной этим алгоритмом ранее. В частности, если многоугольник охватывает окно, то очевидно, что он охватывает и все подокна этого окна и его можно больше не тестировать для них. Кроме того, если многоугольник является внешним по отношению к окну, то он будет внешним и для всех его подокон, и его можно не рассматривать при обработке этих подокон. Учитывать далее следует только пересекающие и внутренние многоугольники.

Чтобы воспользоваться этой информацией, применяют три списка: для охватывающих, для внешних и для пересекающих и внутренних многоугольников [4-11]. По ходу процесса разбиения многоугольники включаются в соответствующий список или удаляются из него. Запоминается также и уровень (шаг разбиения), на котором многоугольник впервые попадает в тот или иной список. Эта информация используется при обходе дерева, изображенного на рис. 4.33, в обратном порядке. На каждом шаге разбиения первым обрабатывается список охватывающих многоугольников для того, чтобы найти ближайший к наблюдателю многоугольник такого типа. Затем обрабатывается список пересекающих и внутренних многоугольников, чтобы проверить, не экранируются ли все они охватывающим многоугольником. Список внешних многоугольников игнорируется.

Итак, были обсуждены основные идеи и методы возможных улучшений алгоритма Варнока. Важно подчеркнуть, что нет единого алгоритма Варнока. Конкретные реализации этого алгоритма различаются в деталях. Запись наиболее фундаментальной версии этого алгоритма на псевдоязыке дана ниже. Если размер окна больше разрешающей способности дисплея и если оно еще содержит нечто, представляющее интерес, то алгоритм всегда будет разбивать окно. Чтобы идентифицировать внешние многоугольники для окон, размер которых больше размера пикселов, проводится простой габаритный тест с прямоугольной оболочкой. А для окон размером с пиксел выполняется более сложный тест, в котором видимый многоугольник определяется путем сравнения координат $z$ всех многоугольников в центре пиксела. Однако здесь не используется сортировка по приоритету вдоль оси $z$, а также выигрыш, даваемый ранее накопленной информацией об относительном расположении многоугольников и окон. Алгоритм использует стек, мак-

симальная длина которого равна

$$3 \cdot (\text{разрешение экрана в битах} - 1) + 5$$

Этот простой алгоритм предназначен для демонстрации основных идей без подробностей реализации структур данных. Для выпуклых многогранников до начала работы данного алгоритма проводится устранение нелицевых граней (см. разд. 4.2). Блок-схема показана на рис. 4.42.

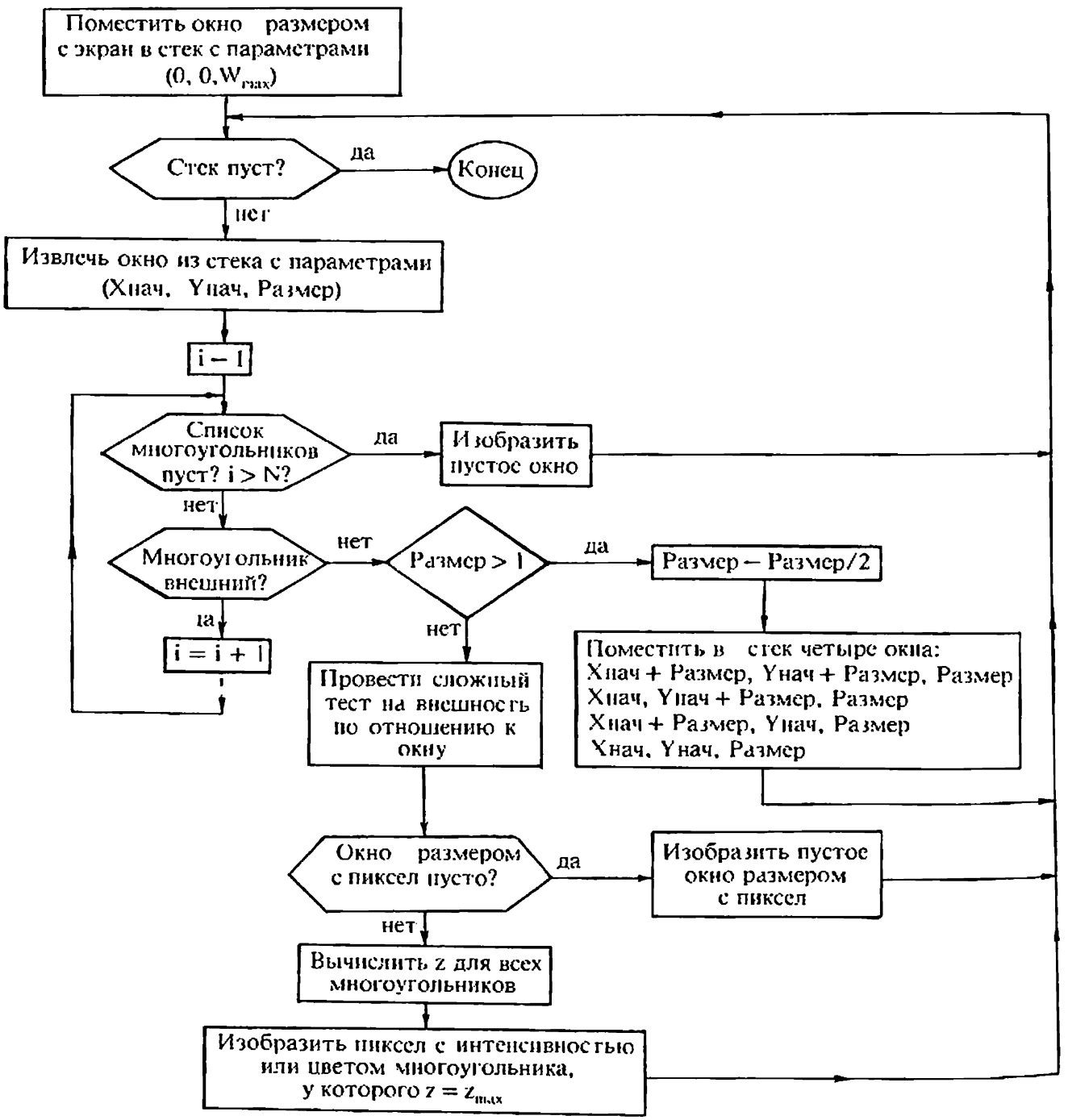



Рис. 4.42. Блок-схема простого алгоритма Варнока.

*Простая реализация алгоритма Варнока*

*предполагается, что экран у дисплея квадратный*
*если в окне что-то есть, то алгоритм разобьет это окно*
*окно разбивается на четыре одинаковых подокна*
*каждый многоугольник сравнивается с каждым окном*
*предполагается, что все значения заданы в системе координат экрана (пространстве изображения)*
*предполагается, что устранение нелицевых граней проведено до начала работы алгоритма*
Вершина — *массив размером m × 3, в котором запоминаются координаты вершин всех многоугольников*
m — *общее число вершин многоугольников в сцене. Предполагается, что эти вершины перечислены в порядке их обхода по часовой стрелке*
N — *число многоугольников в сцене*
*информация об отдельных многоугольниках запоминается в массиве*
Многоугольник *размером N × 11*
Многоугольник ( ,1) — *указатель на первую вершину многоугольника в массиве Вершина*
Многоугольник ( ,2) — *число вершин многоугольника*
Многоугольник ( ,3) — *интенсивность света или цвет многоугольника*
Многоугольник ( ,4—7) — *коэффициенты a, b, c, d уравнения плоскости, несущей многоугольник*
Многоугольник ( ,8—11) — *габариты* $x_{min}$, $x_{max}$, $y_{min}$, $y_{max}$ *прямоугольной объемлющей оболочки многоугольника*
**Push** — *функция, заносящая окна в стек*
**Pop** — *функция, извлекающая окна из стека*
$W_{max}$ — *максимальные габариты окна по x и y. Предполагается, что начало координат экрана находится в точке (0, 0)*
Окно — *массив размером 1×3, который содержит начало координат окна и его размер в форме* (Хнач, Унач, Размер)
Внешность — *флаг = 0 для пустого окна,*
*⩾1 для непустого окна*
*инициализировать черный фоновый цвет или интенсивность*
Фон = 0
*Поместить окно размером с экран в стек*

```
Push Окно(0, 0 Wmax)
while (стек не пуст)
   взять окно из стека
   Pop Окно(Хнач, Унач, Размер)
   инициализировать счетчик многоугольников
   i = 1
   Внешность = 0
   Выполнить для каждого многоугольника габаритный тест с
   прямоугольной оболочкой, чтобы найти внешние много-
   угольники
   while (i ⩽ N and внешность = 0)
      call Габарит(i, Многоугольник, Окно, Внешность)
      i = i + 1
   end while
   если хотя бы один многоугольник не является внешним, то
   разбить окно или изобразить пиксел
   if Внешность > 0 then:
      если размер окна больше пиксела, то разбить окно
      if Размер > 1 then
         Размер = Размер/2
         Push Окно(Хнач + Размер, Унач + Размер, Размер)
         Push Окно(Хнач, Унач + Размер, Размер)
         Push Окно(Хнач + Размер, Унач, Размер)
         Push Окно(Хнач, Унач, Размер)
      else
         если окно размером с пиксел, то вычислить его ат-
         рибуты
         call Покрытие(Вершина, N, Многоугольник, Окно;
         Номер)
         if Номер > 0 then
            call Визуализация(Окно, Многоугольник (Номер, 3))
         else
            Изобразить пустое окно
            call Визуализация(Окно, Фон)
          end if
      end if
   else
      call Визуализация(Окно, Фон)
   end if
end while
finish
```

*подпрограмма реализации простого габаритного теста с прямоугольной оболочкой*

**subroutine** Габарит(i, Многоугольник, Окно; Внешность)

*вычисление габаритов окна:* Хлев, Yправ, Хниз, Yверх

```
Хлев  = Окно(1, 1)
Хправ = Окно(1, 1) + Окно(1,3) − 1
Yниз  = Окно(1, 2)
Yверх = Окно(1, 2) + Окно(1, 3) − 1
```

*реализация тестов с прямоугольной оболочкой*

```
Внешность = 1
if Многоугольник(i, 8)  > Хправ then Внешность= 0
if Многоугольник(i, 9)  < Хлев then Внешность = 0
if Многоугольник(i, 10) > Yверх then Внешность= 0
if Многоугольник(i, 11) < Yниз then Внешность = 0
return
```

*подпрограмма визуализации окна*

**subroutine** Визуализация(окно, Интенсивность)

**Setpixel**(x, y, I) — *функция визуализации пиксела, координаты которого* (x, y), с интенсивностью I

```
  forj = Окно(1, 2) to Окно(1, 2) + Окно(1, 3) − 1
    fori = Окно(1, 1) to Окно(1, 1) + Окно(1, 3) − 1
      Setpixel(i, j, Интенсивность)
    next i
  next j
  return
```

*подпрограмма, проверяющая, покрывает ли многоугольник центр окна*

**subroutine** Покрытие(Вершина, N, Многоугольник, Окно; Номер)

*считается, что многоугольник покрывает окно размером с пиксел, если центр этого окна находится внутри многоугольника*

*поскольку вершины многоугольника перечисляются по часовой стрелке, его внутренность лежит справа от контура*

*данный алгоритм использует подпрограмму Видимость, описанную в разд. 3.16*

*если покрывающие многоугольники не обнаружатся, то*

```
Номер = 0
```

*если же найден хотя бы один покрывающий многоугольник,*

*то* Номер *будет равен номеру того из них, который видим*
*присвоить* Zmax *начальное значение, равное нулю. Тут предполагается, что все многоугольники расположены в положительном полупространстве* $Z \geqslant 0$

```
Zmax = 0
```
*вначале предположим, что покрывающих многоугольников нет*
```
Номер = 0
```
*установить центр окна*
```
Точкаx = Окно(1, 1) + Окно(1, 3)/2
Точкаy = Окно(1, 2) + Окно(1, 3)/2
```
*для каждого многоугольника*
```
for i = 1 to N
   Индекс = Многоугольник(i, 1)
   для каждого ребра многоугольника
   for j = 1 to Многоугольник(i, 2) – 1
      T1x = Вершина(Индекс, 1)
      T1y = Вершина(Индекс, 2)
      T2x = Вершина(Индекс + 1, 1)
      T2y = Вершина(Индекс + 1, 2)
      заметим, что Точка, T1, T2 — это сокращения для
      идентификаторов Точкаx, Точкаy и т. п.
      call Видимость(Точка, T1, T2; Твидимость),
      if Твидимость < 0 then 1
      Индекс = Индекс + 1
   next j
   проверка относительно последнего ребра многоугольника
   T1x = Вершина(Индекс, 1)
   T1y = Вершина(Индекс, 2)
   T2x = Вершина(Многоугольник (i, 1), 1)
   T2y = Вершина(Многоугольник (i, 1), 2)
   call Видимость(Точка, T1, T2, Твидимость)
   if Твидимость ⩾ 0 then
      call Вычислить(Вершина, i, Многоугольник, Окно; z)
      if z > Zmax then
         Zmax = z
         Номер = i
      end if
   end if
1 next i
  return
```

*подпрограмма вычисления интенсивности пиксела*[1)]

```
subroutine Вычислить(Вершина, i, Многоугольник, Окно; z)
```
*уравнение плоскости многоугольника используется для вычисления многоугольника, который находится ближе других к точке наблюдения для этого пиксела*

Max — *обозначение функции, вычисляющей максимум*

```
  вычисление координат x и y центра пиксела
  Хцентр = Окно(1, 1) + Окно(1, 3)/2
  Yцентр = Окно(1, 2) + Окно(1, 3)/2
  определение координаты z многоугольника в центре пиксела
  поиск ребра многоугольника, проходящего через центр пиксела
  заметим, что многоугольник такого типа может совершенно отсутствовать или появиться в качестве набора несвязанных точек — например, в результате ступенчатости
    if Многоугольник(i, 6) = 0 then
    for j = 2 to Многоугольник(i, 2)
      z = Max(Вершина(j, 3), Вершина(j −1, 3))
    next j
  else
    вычисление z по уравнению плоскости многоугольника
    A = Многоугольник(i, 4)
    B = Многоугольник(i, 5)
    C = Многоугольник(i, 6)
    D = Многоугольник(i, 7)
    z = −(A * Хцентр + B * Yцентр + D)/C
  end if
  return
```

Для иллюстрации работы этого алгоритма привелем один пример.

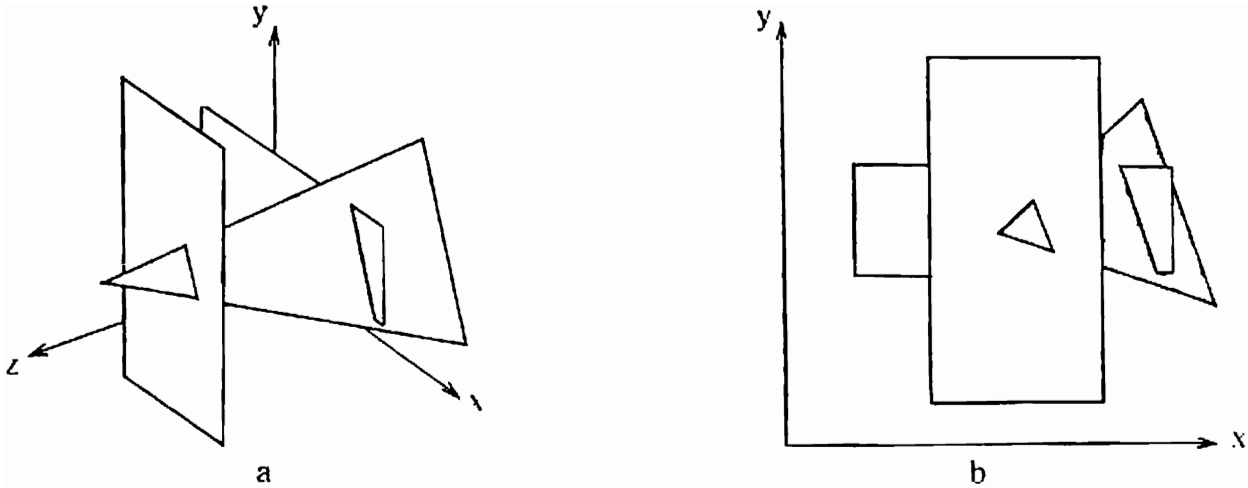
**Пример 4.17.** Алгоритм Варнока

Рассмотрим три многоугольника:

1: (10, 3, 20), (20, 28, 20), (22, 28, 20), (22, 3, 20)
2: (5, 12, 10), (5, 20, 10), (27, 20, 10), (27, 12, 20)
3: (15, 15, 25), (25, 25, 5), (30, 10, 5)

---

[1)] Точнее глубины многоугольника, покрывающего пиксел. — *Прим. перев.*



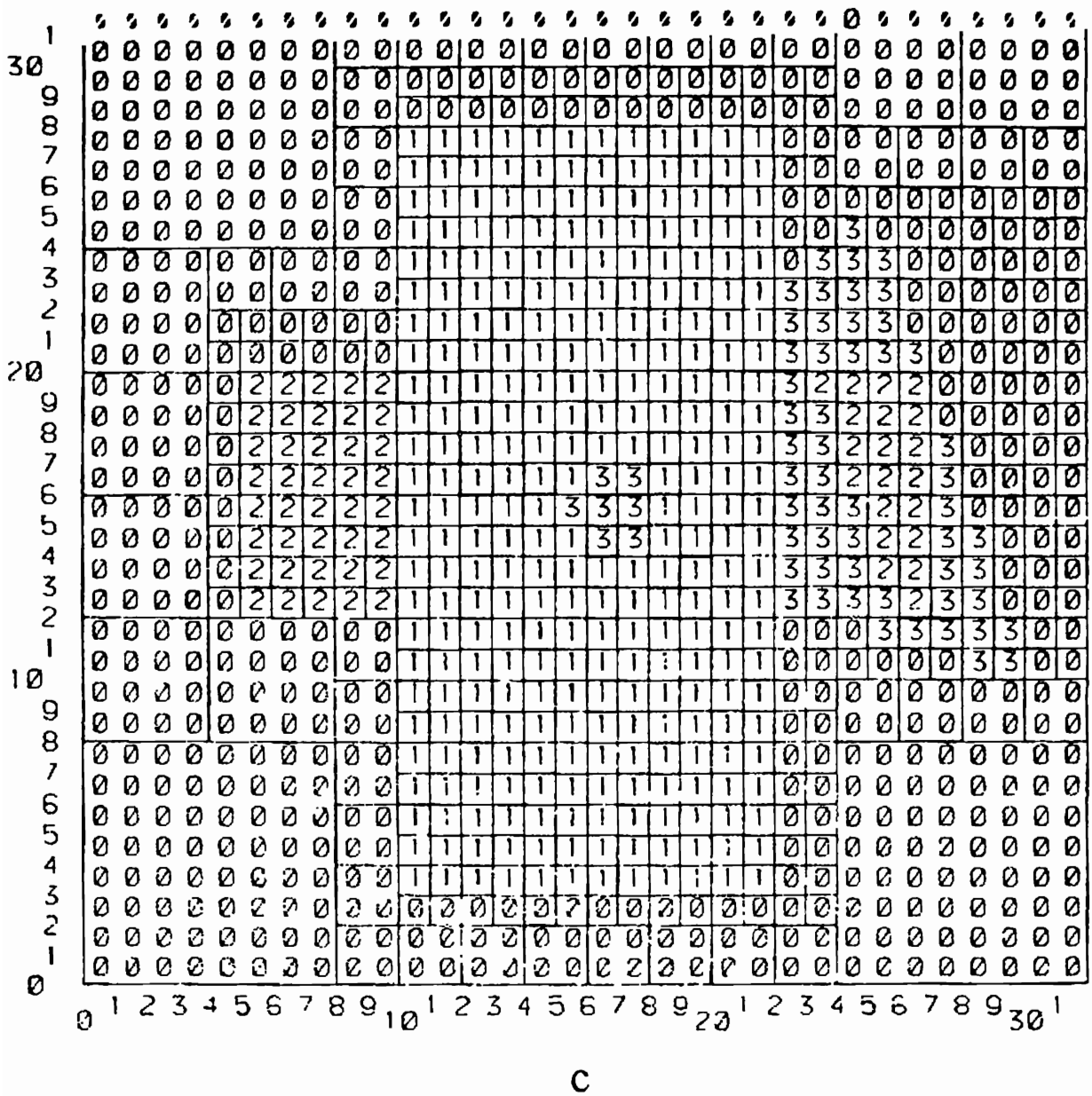




Рис. 4.43. Пример многоугольников для простого алгоритма Варнока.

которые нужно изобразить на экране с разрешением 32×32 пиксела, используя простой алгоритм Варнока, описанный выше. Два первых многоугольника являются прямоугольниками, перпендикулярными оси $z$ соответственно при $z = 20$ и $z = 10$. Третий — это треугольник, протыкающий оба прямоугольника, как показано на рис. 4.43, a. На рис. 4.43, b показан вид на эту сцену из точки, лежащей в бесконечности на положительной полуоси $z$; невидимые линии удалены с рисунка. На рис. 4.43, c показано содержимое буфера кадра после завершения работы алгоритма. Цифры в ячейках соответствуют номерам многоугольников. Алгоритм работает, начиная с левого нижнего угла, направо и вверх. Крупные ячейки показывают габариты окон, полученных в результате разбиения экрана на каждом шаге алгоритма. Обратим, например, внимание на большое пустое окно (8×8) в левом нижнем углу экрана. Это окно изображено без дальнейшего подразбиения. На рисунках показано, что треугольник частично экранирован вторым прямоугольником, затем протыкает его, становится частично видимым, потом экранируется первым прямоугольником, и, наконец, протыкает и его, так что делается видимой вершина этого треугольника.

## 4.5. АЛГОРИТМ ВЕЙЛЕРА— АЗЕРТОНА

Вейлер и Азертон [4-12] попытались минимизировать количество шагов в алгоритме разбиения типа алгоритма Варнока путем разбиения окна вдоль границ многоугольника. Основой этого алгоритма послужил другой — Вейлера — Азертона, используемый для отсечения многоугольников и обсуждавшийся ранее в разд. 3.17. Выходными данными этого алгоритма, который для достижения необходимой точности работает в пространстве объекта, служат многоугольники. Поскольку выходом являются многоугольники, то алгоритм можно легко использовать для удаления как невидимых линий, так и невидимых поверхностей. Алгоритм удаления невидимых поверхностей состоит из четырех шагов.

Предварительная сортировка по глубине.

Отсечение по границе ближайшего к наблюдателю многоугольника, называемое сортировкой многоугольников на плоскости.

Удаление многоугольников, экранированных многоугольником, ближайшим к точке наблюдения.

Если требуется, то рекурсивное подразбиение и окончательная сортировка для устранения всех неопределенностей.

Предварительная сортировка по глубине нужна для формирования списка приблизительных приоритетов. Предположим, что точка наблюдения расположена в бесконечности на положительной полуоси $z$, тогда ближайшим к ней и первым в списке будет тот многоугольник, который обладает вершиной с максимальной координатой $z$.

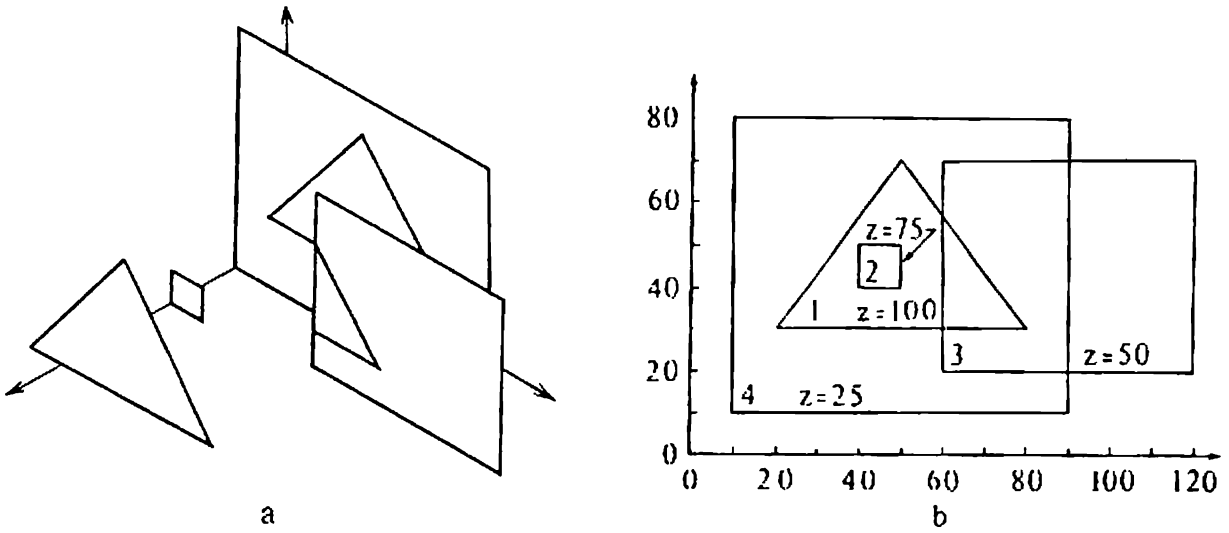



Рис. 4.44. Отсечение многоугольников по приоритетам для алгоритма уда ения невидимых поверхностей Вейлера — Азертона.

В качестве отсекающего многоугольника используется копия первого многоугольника из предварительного списка приоритетов по глубине. Отсекаться будут остающиеся в этом списке многоугольники, включая и первый многоугольник. Вводятся два списка: внутренний и внешний. С помощью алгоритма отсечения Вейлера — Азертона все многоугольники отсекаются по границам отсекающего многоугольника. Фактически это двумерная операция отсечения проекций отсекающего и отсекаемого многоугольников. Та часть каждого отсекаемого многоугольника, которая оказывается внутри отсекающего, если она имеется, попадает во внутренний список. Оставшаяся часть, если таковая есть, попадает во внешний список. Этот этап алгоритма является сортировкой на плоскости или *xy*-сортировкой. Пример приведен на рис. 4.44. На рис. 4.45 показаны внутренний и внешний списки для сцены на рис. 4.44. Теперь сравниваются глубины каждого многоугольника из внутреннего списка с глубиной отсекающего многоугольника. С использованием координат ( *x*, *y*) вершин отсекаемых многоугольников и уравнений несущих плоскостей вычисляются глубины (координаты *z*) каждой вершины. Затем они сравниваются с минимальной координатой $z$( $z_{c\,min}$) для отсекающего многоугольника. Если глубина ни одной из этих вершин не будет больше $z_{c\,min}$, то все многоугольники из внутреннего списка экранируются отсекающим многоугольником (рис. 4.44). Эти многоугольники удаляются, и изображается внутренний список. Заметим, что во внутреннем списке остался лишь отсекающий многоугольник. Работа алгоритма затем продолжается с внешним списком.

Если координата *z* какого-либо многоугольника из внутреннего

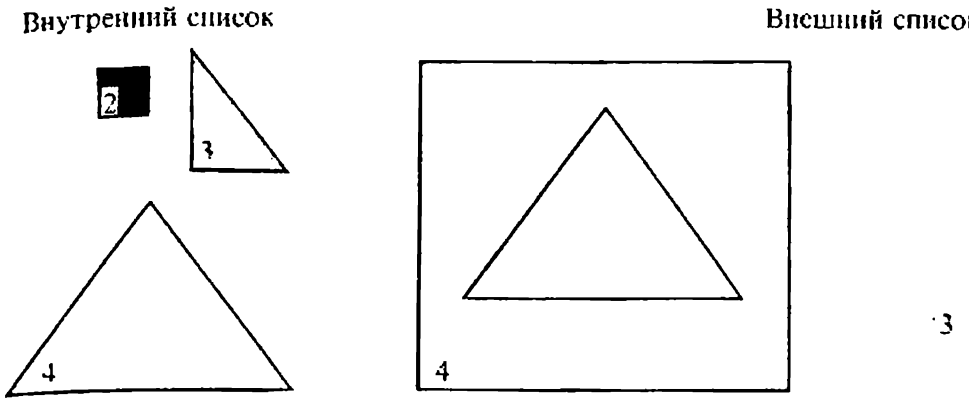



Рис. 4.45 Внутренний и внешний списки многоугольников.

списка окажется больше, чем $z_{c\,min}$, то такой многоугольник по крайней мере частично экранирует отсекающий многоугольник. На рис. 4.46 показано, как это может произойти. В подобном случае результат предварительной сортировки по глубине ошибочен. Поэтому алгоритм рекурсивно подразделяет плоскость $(x, y)$, используя многоугольник, нарушивший порядок, в качестве нового отсекающего многоугольника. Отсечению подлежат многоугольники из внутреннего списка, причем старый отсекающий многоугольник теперь сам будет подвергнут отсечению новым отсекающим многоугольником. Подчеркнем, что новый отсекающий многоугольник является копией исходного многоугольника, а не его остатка после первого отсечения. Использование копии неотсеченного многоугольника позволяет минимизировать число разбиений.

Более полной иллюстрацией этого алгоритма послужит следующий простой пример.

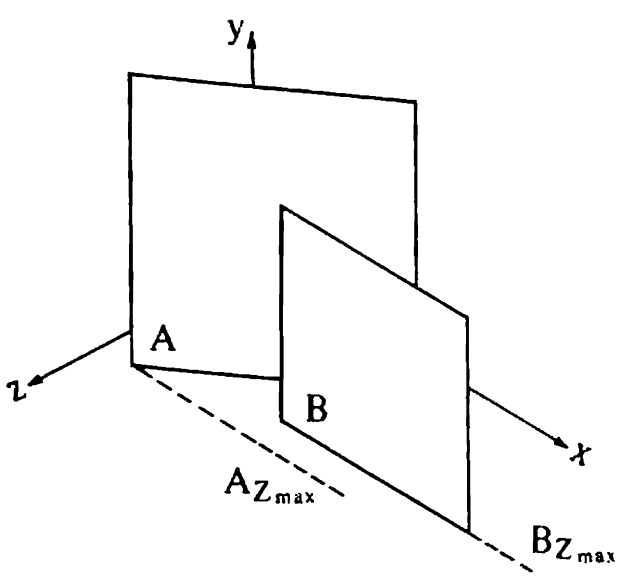



Рис. 4.46. Условие возникновения ошибочного результата в предварительной сортировке по $z$.

**Пример 4.18.** Алгоритм удаления невидимых поверхностей Вейлера — Азертона

Возьмем два прямоугольника (рис. 4.46). Вершинами прямоугольника $A$ являются (5, 0, 25), (40, 0, 5), (40, 40, 5) и (5, 40, 25). Прямоугольник $B$ имеет вершины (25, 0, 20), (55, 0, 20), (55, 30, 20) и (25, 30, 20). На рис. 4.47, a показана эта сцена до отсечения из точки наблюдения, расположенной в бесконечности на положительной полуоси $z$. Хотя $B$ экранирует часть $A$, предварительная сортировка по глубине поставит $A$ перед $B$ в упорядоченном списке приоритетов. Копия $A$ используется в качестве первого отсекающего многоугольника. Первоначальный список отсекаемых многоугольников содержит и $A$, и $B$, как показано в табл. 4.10. В табл. 4.10 и на рис. 4.47, b показан результат отсечения содержимого этого списка относительно многоугольника $A$. Теперь внутренний список содержит многоугольники $A$ и $C$, а внешний список содержит $B'$. Сравнение глубин $A$ и $C$ с глубиной отсекающего многоугольника показывает, что $C$ находится перед отсекающим многоугольником. Далее алгоритм рекурсивно разбивает плоскость, используя в качестве отсекающего многоугольник $B$, частью которого является $C$, а в качестве списка отсекаемых многоугольников — внутренний список. Результат показан на рис. 4.47, c и табл. 4.10. Участок, обозначенный $A'$, отсекается и помещается во внешний список. Участок, обозначенный $D$, помещается во внутренний список. Сравнение многоугольников $C$ и $D$ из внутреннего списка с отсекающим многоугольником $B$ пока-

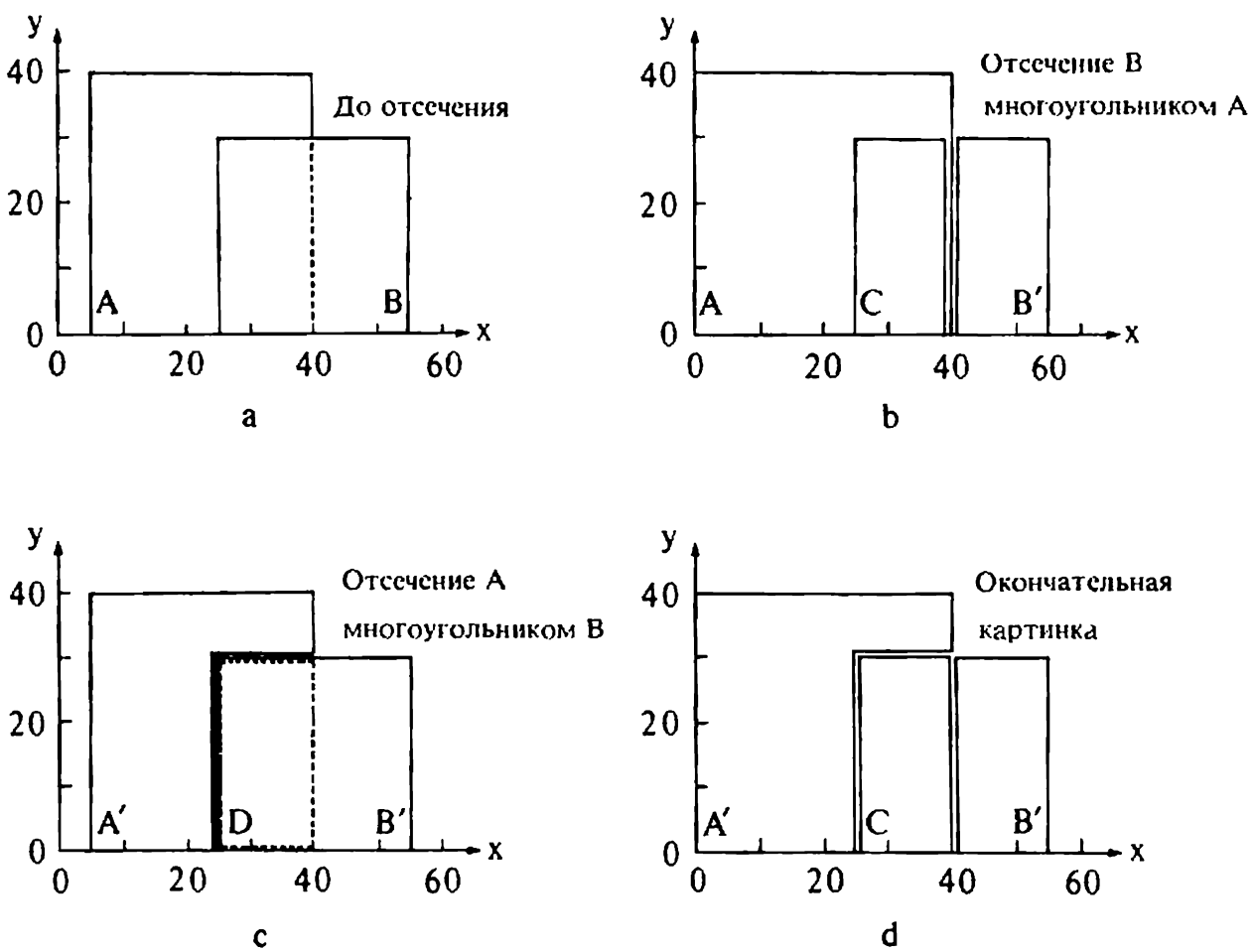



Рис. 4.47. Рекурсивное разбиение для алгоритма Вейлера — Азертона.

Таблица 4.10.

| Отсекающий многоугольник | Отсекаемый многоугольник | Внутренний список | Начальный внешний список | Конечный внешний список | Изображается | Примечание |
|---|---|---|---|---|---|---|
| $A$ | $A$ | $A$ | | $B'$ | | Рекурсия |
| | $B$ | $C$ | | | | |
| $B$ | $A$ | $C$ | $B'$ | $A'$ | $C$ | Продолжение работы с внешним списком |
| | $C$ | $D$ | | | | |
| | | | | $B'$ | | |
| $A'$ | $A'$ | $A'$ | $B'$ | $B'$ | $A'$ | |
| $B'$ | $B'$ | $B'$ | $B'$ | | $B'$ | |

зывает, что $D$ экранирован. Следовательно, он удаляется. Поскольку $C$ совпадает с отсекающим многоугольником $B$, он остается во внутреннем списке. Рекурсия не нужна. Многоугольник $C$ изображается. Алгоритм продолжает работу с многоугольниками $B'$ и $A$ , извлекаемыми из внешнего списка. Подробности приведены в табл. 4.10. Окончательный результат показан на рис. 4.47, d.

Заслуживает внимание еще одна дополнительная деталь этого алгоритма. Когда некоторый многоугольник циклически перекрывается с отсекающим, т. е. когда он лежит как впереди, так и позади отсекающего (рис. 4.48, а), то в рекурсивном разбиении необходимости нет. Дело в том, что все экранируемое циклическим многоугольником уже было удалено на предыдущем шаге отсечения. Необходимо лишь произвести отсечение исходного многоугольника по границам циклического многоугольника и изобразить результат. Ненужное рекурсивное разбиение можно предотвратить, если создать список многоугольников, которые уже использовались как

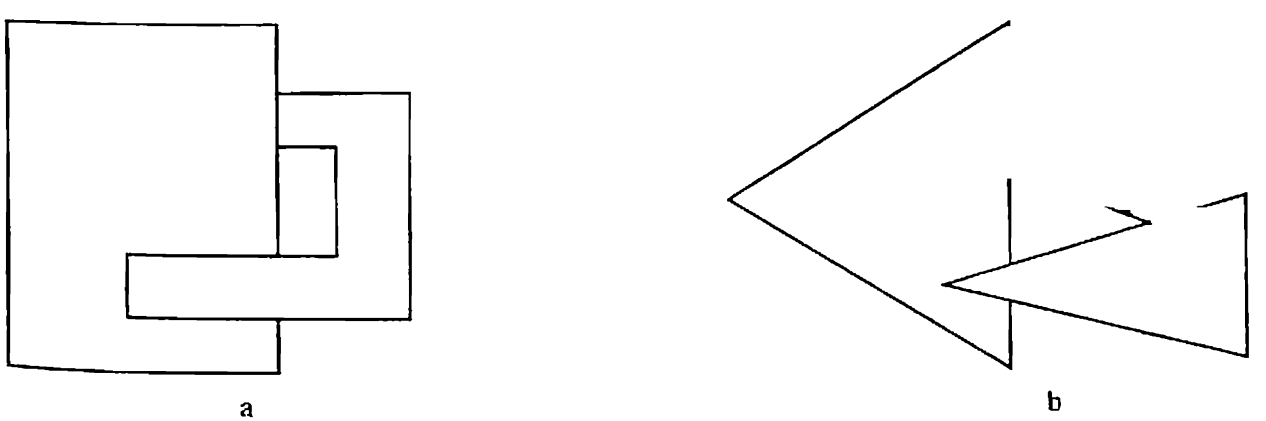



Рис. 4.48. Циклически перекрывающиеся многоугольники.

отсекающие. Тогда, если при рекурсивном разбиении текущий отсекающий многоугольник появляется в этом списке, значит, обнаружен циклически перекрывающийся многоугольник. Следовательно, не требуется никакого дополнительного разбиения. Заметим, что данный алгоритм непосредственно обрабатывает случаи циклического перекрытия сразу нескольких многоугольников, как показано на рис. 4.48, b.

## 4.6. АЛГОРИТМ РАЗБИЕНИЯ КРИВОЛИНЕЙНЫХ ПОВЕРХНОСТЕЙ

Как в основной версии алгоритма разбиения Варнока, так и в версии Вейлера — Азертона, предполагается, что сцена состоит из набора плоских многоугольников. Однако многие объекты описываются криволинейными поверхностями, например самолеты, корабли, автомобили, мебель и т. п. Полигональные аппроксимации таких поверхностей не всегда дают адекватные представления, например силуэтные линии выглядят не как непрерывные кривые, а как состоящие из отдельных коротких отрезков прямых. Кэтмул [4-13, 4-14] предложил для визуализации криволинейных поверхностей алгоритм типа алгоритма разбиения Варнока. Хотя сам Кэтмул использовал этот алгоритм для поверхностей, заданных бикубическими элементами, он обладает общностью, достаточной для применения его к любым криволинейным поверхностям. В отличие от алгоритма Варнока, который рекурсивно разбивал пространство изображения, алгоритм Кэтмула рекурсивно разбивает поверхность. Простейшее описание этого алгоритма таково:

> Рекурсивно разбивать поверхность на элементы до тех пор, пока проекция каждого элемента на пространство изображения не будет покрывать не более одного центра пиксела.
>
> Вычислить атрибуты поверхности в этом пикселе и изобразить его.

На рис. 4.49, a показан элемент поверхности и его разбиение на более мелкие элементы размером с пиксел. Чтобы убедиться в том, что криволинейный элемент покрывает ровно один центр пиксела, если поверхность не слишком искривлена, обычно бывает достаточно его полигональной аппроксимации (рис. 4.49, b). Процесс разбиения завершается, когда появляются элементы, которые не покрывают ни одного центра пиксела. Атрибуты этих элементов присваиваются ближайшим к ним центрам пикселов. Те элементы

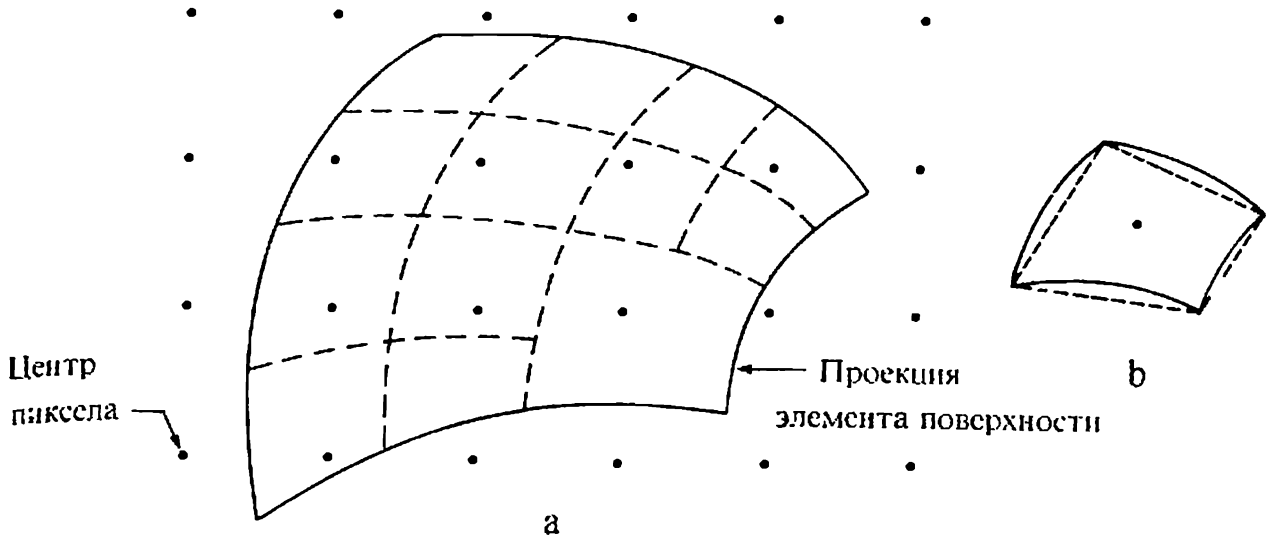



Рис. 4.49. Разбиение криволинейной поверхности.

поверхности, которые проецируются за пределы окна видимости, разумеется отбрасываются. Элементы, которые пересекают ребра окна видимости, разбиваются дальше до тех пор, пока не станет очевидным вопрос об их расположении относительно окна.

Эффективность этого алгоритма зависит от эффективности метода разбиения криволинейной поверхности. Кэтмул предложил один метод для разбиения бикубических элементов. Коэн, Лич и Ризенфельд [4-15] предложили более общий метод для поверхностей, заданных $B$-сплайнами.

## 4.7. АЛГОРИТМ, ИСПОЛЬЗУЮЩИЙ $z$-БУФЕР

Это один из простейших алгоритмов удаления невидимых поверхностей. Впервые он был предложен Кэтмулом [4-14]. Работает этот алгоритм в пространстве изображения. Идея $z$-буфера является простым обобщением идеи о буфере кадра. Буфер кадра используется для запоминания атрибутов (интенсивности) каждого пиксела в пространстве изображения. $z$-буфер — это отдельный буфер глубины, используемый для запоминания координаты $z$ или глубины каждого видимого пиксела в пространстве изображения. В процессе работы глубина или значение $z$ каждого нового пиксела, который нужно занести в буфер кадра, сравнивается с глубиной того пиксела, который уже занесен в $z$-буфер. Если это сравнение показывает, что новый пиксел расположен впереди пиксела, находящегося в буфере кадра, то новый пиксел заносится в этот буфер и, кроме того, производится корректировка $z$-буфера новым значени-

ем $z$. Если же сравнение дает противоположный результат, то никаких действий не производится. По сути, алгоритм является поиском по $x$ и $y$ наибольшего значения функции $z(x, y)$.

Главное преимущество алгоритма — его простота. Кроме того, этот алгоритм решает задачу об удалении невидимых поверхностей и делает тривиальной визуализацию пересечений сложных поверхностей. Сцены могут быть любой сложности. Поскольку габариты пространства изображения фиксированы, оценка вычислительной трудоемкости алгоритма не более чем линейна. Поскольку элементы сцены или картинки можно заносить в буфер кадра или в $z$-буфер в произвольном порядке, их не нужно предварительно сортировать по приоритету глубины. Поэтому экономится вычислительное время, затрачиваемое на сортировку по глубине.

Основной недостаток алгоритма — большой объем требуемой памяти. Если сцена подвергается видовому преобразованию и отсекается до фиксированного диапазона координат $z$ значений, то можно использовать $z$-буфер с фиксированной точностью. Информацию о глубине нужно обрабатывать с большей точностью, чем координатную информацию на плоскости $(x, y)$; обычно бывает достаточно 20 бит. Буфер кадра размером $512 \times 512 \times 24$ бит в комбинации с $z$-буфером размером $512 \times 512 \times 20$ бит требует почти *1.5 мегабайт* памяти. Однако снижение цен на память делает экономически оправданным создание специализированных запоминающих устройств для $z$-буфера и связанной с ним аппаратуры.

Альтернативой созданию специальной памяти для $z$-буфера является использование для этой цели оперативной или массовой памяти. Уменьшение требуемой памяти достигается разбиением пространства изображения на 4, 16 или больше квадратов или полос. В предельном варианте можно использовать $z$-буфер размером в одну строку развертки. Для последнего случая имеется интересный алгоритм построчного сканирования (см. разд. 4.9). Поскольку каждый элемент сцены обрабатывается много раз, то сегментирование $z$-буфера, вообще говоря, приводит к увеличению времени, необходимого для обработки сцены. Однако сортировка на плоскости, позволяющая не обрабатывать все многоугольники в каждом из квадратов или полос, может значительно сократить этот рост.

Другой недостаток алгоритма $z$-буфера состоит в трудоемкости и высокой стоимости устранения лестничного эффекта, а также реализации эффектов прозрачности и просвечивания. Поскольку алгоритм заносит пикселы в буфер кадра в произвольном порядке, то нелегко получить информацию, необходимую

для методов устранения лестничного эффекта, основывающихся на предварительной фильтрации (см. разд. 2.27). При реализации эффектов прозрачности и просвечивания (см. разд. 5.9), пикселы могут заноситься в буфер кадра в некорректном порядке, что ведет к локальным ошибкам.

Хотя реализация методов устранения лестничного эффекта, основывающихся на префильтрации, в принципе возможна [4-13], практически это сделать трудно. Однако относительно легко реализуются методы постфильтрации (усреднение подпикселов) (см. разд. 2.26). Напомним, что в методах устранения лестничного эффекта, основывающихся на постфильтрации, сцена вычисляется в таком пространстве изображения, разрешающая способность которого выше, чем разрешающая способность экрана. Поэтому возможны два подхода к устранению летничного эффекта на основе постфильтрации. В первом используется буфер кадра, заданный в пространстве изображения, разрешение которого выше, чем у экрана, и $z$-буфер, разрешение которого совпадает с разрешением экрана. Глубина изображения вычисляется только в центре той группы подпикселов, которая усредняется. Если для имитации расстояния от наблюдателя используется масштабирование интенсивности, то этот метод может оказаться неадекватным.

Во втором методе оба буфера, заданные в пространстве изображения, имеют повышенную разрешающую способность. При визуализации изображения как пикселная информация, так и глубина усредняются. В этом методе требуются очень большие объемы памяти. Например, изображение размером $512 \times 512 \times 24$ бита, использующее $z$-буфер размером 20 бит на пиксел, разрешение которого повышено в 2 раза по осям $x$ и $y$ и на котором устранена ступенчатость методом равномерного усреднения (см. рис. 2.53, а), требует почти *6 мегабайт* памяти. Более формальное описание алгоритма $z$-буфера таково:

Заполнить буфер кадра фоновым значением интенсивности или цвета.

Заполнить $z$-буфер минимальным значением $z$.

Преобразовать каждый многоугольник в растровую форму в произвольном порядке.

Для каждого Пиксел($x$, $y$) в многоугольнике вычислить его глубину $z(x, y)$.

Сравнить глубину $z(x, y)$ со значением $Z$ буфер($x$, $y$), храня-

щимся в $z$-буфере в этой же позиции.

Если $z(x, y) > Z$ буфер$(x, y)$, то записать атрибут этого многоугольника (интенсивность, цвет и т. п.) в буфер кадра и заменить $Z$ буфер$(x, y)$ на $z(x, y)$.

В противном случае никаких действий не производить.

В качестве предварительного шага там, где это целесообразно, применяется удаление нелицевых граней (см. разд. 4.2).

Если известно уравнение плоскости, несущей каждый многоугольник, то вычисление глубины каждого пиксела на сканирующей строке можно проделать пошаговым способом. Напомним, что уравнение плоскости имеет вид

$$ax + by + cz + d = 0$$

$$z = -(ax + by + d)/c \neq 0$$

Для сканирующей строки $y = \text{const}$. Поэтому глубина пиксела на этой строке, у которого $x_1 = x + \Delta x$, равна

$$z_1 - z = -(ax_1 + d)/c + (ax + d)/c = a(x - x_1)/c$$

или

$$z_1 = z - (a/c)\Delta x$$

Но $\Delta x = 1$, поэтому $z_1 = z - (a/c)$.

Дальнейшей иллюстрацией алгоритма послужит следующий пример.

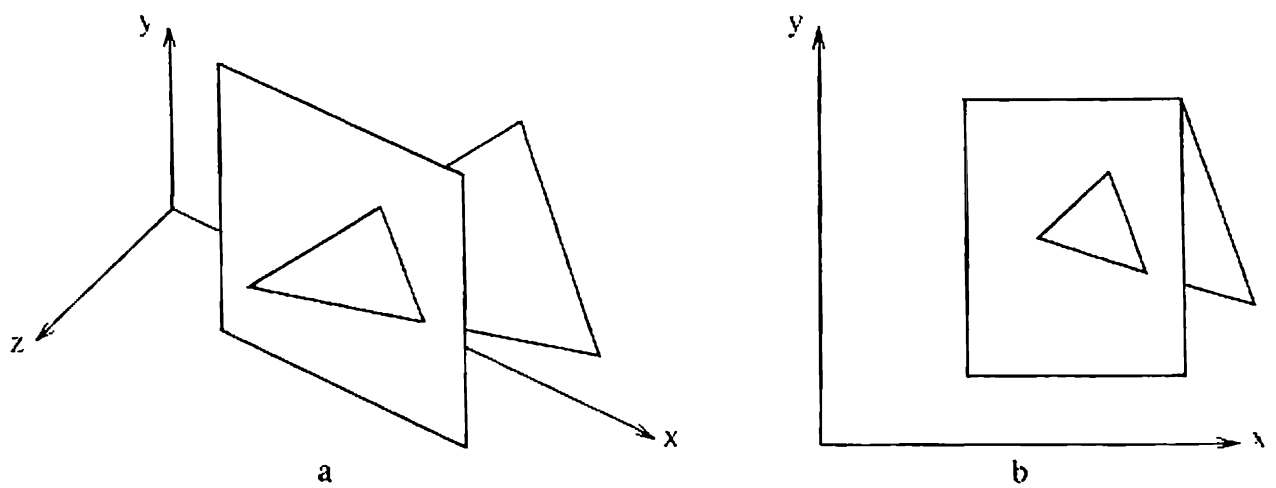



Рис. 4.50. Протыкающий треугольник.

**Пример 4.19.** Алгоритм, использующий $z$-буфер

Рассмотрим многоугольник, координаты угловых точек которого равны $P_1(10, 5, 10)$, $P_2(10, 25, 10)$, $P_3(25, 25, 10)$, $P_4(25, 5, 10)$ и треугольник с вершинами $P_5(15, 15, 15)$, $P_6(25, 25, 5)$, $P_7(30, 10, 5)$. Треугольник протыкает прямоугольник, как показано на рис. 4.50. Эти многоугольники нужно изобразить на экране с разрешением $32 \times 32$, используя простой буфер кадра с двумя битовыми плоскостями. В этом буфере фон обозначен через 0, прямоугольник — через 1, а треугольник — через 2. Под $z$-буфер отводится 4 битовых плоскости размером $32 \times 32$ бит. Таким образом, содержимое $z$-буфера окажется в диапазоне от 0 до 16. Точка наблюдения расположена в бесконечности на положительной полуоси $z$, как показано на рис. 4.50, b.

Вначале и в буфере кадра, и в $z$-буфере содержатся нули. После растровой развертки прямоугольника содержимое буфера кадра будет иметь следующий вид:

```
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

Содержимое $z$-буфера таково:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Напомним, что в левом нижнем углу находится пиксел (0, 0).

Используя метод Ньюэлла (см. разд. 4.3, пример 4.3), получаем уравнение плоскости треугольника:

$$3x + y + 4z - 120 = 0$$

Значит, глубина треугольника в любой его точке задается уравнением

$$z = -(3x + y - 120)/4$$

Для последовательных пикселов, лежащих на сканирующей строке, имеет место

$$z_1 = z - 3/4$$

Вычисление пересечений сторон треугольника со сканирующими строками развертки с учетом соглашения о половине интервала между соседними сканирующими строками дает следующие пары координат (25.2, 24.5), (25.5, 23.5), (25.8, 22.5), (26.2, 21.5), (26.5, 20.5), (26.8, 19.5), (27.2, 18.5), (27.5, 17.5), (27.8, 16.5), (28.2, 15.5), (28.5, 14.5), (28.8, 13.5), (29.2, 12.5), (29.5, 11.5), (29.8, 10.5) для строк от 24 до 10. Напомним, что активируется тот пиксел, у которого центр лежит внутри или на границе треугольника, т. е. при $x_1 \leqslant x \leqslant x_2$. Преобразование в растровую форму и сравнение глубины каждого пиксела со значением $z$-буфера дает новое состояние буфера кадра:

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 2 2 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 2 2 1 1 1 1 1 2 2 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 2 2 2 2 1 1 1 1 2 2 2 0 0 0 0
0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 2 2 2 2 2 1 1 1 1 2 2 2 0 0 0 0
0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 2 2 2 2 2 2 1 1 1 1 2 2 2 2 0 0 0
0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 2 2 2 2 2 2 1 1 1 2 2 2 2 0 0 0
0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 2 2 2 2 2 1 1 1 2 2 2 2 2 0 0
0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 0
0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 0
0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 2 2 0 0
0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

После обработки треугольника состояние $z$-буфера таково:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 6 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 6 | 5 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 11 | 10 | 10 | 10 | 10 | 10 | 6 | 5 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 12 | 11 | 10 | 10 | 10 | 10 | 10 | 6 | 6 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 13 | 12 | 11 | 10 | 10 | 10 | 10 | 10 | 7 | 6 | 5 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 10 | 10 | 10 | 10 | 10 | 10 | 14 | 13 | 12 | 11 | 11 | 10 | 10 | 10 | 10 | 7 | 6 | 5 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 10 | 10 | 10 | 10 | 10 | 15 | 14 | 13 | 12 | 12 | 11 | 10 | 10 | 10 | 10 | 7 | 6 | 6 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 10 | 10 | 10 | 10 | 10 | 10 | 14 | 13 | 13 | 12 | 11 | 10 | 10 | 10 | 10 | 7 | 7 | 6 | 5 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 12 | 11 | 11 | 10 | 10 | 10 | 8 | 7 | 6 | 5 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 8 | 7 | 6 | 6 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 8 | 7 | 7 | 6 | 5 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 0 | 0 | 6 | 5 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Для примера рассмотрим пиксел (20, 15). Оценка $z$ в центре этого пиксела дает

$$z = -[(3), (20.5) + 15.5 - 120]/4 = 43/4 = 10.75$$

Сравнивая его со значением $z$-буфера в точке (20, 15) после обработки прямоугольника, видим, что треугольник здесь расположен перед прямоугольником. Поэтому значение буфера кадра в точке (20, 15) заменяется на 2. Поскольку в нашем примере $z$-буфер состоит лишь из 4 битовых плоскостей, он может содержать числа только в диапазоне от 0 до 15. Поэтому значение $z$ округляется до ближайшего целого числа. В результате в ячейку (20, 15) $z$-буфера заносится число 11.

Линия пересечения треугольника с прямоугольником получается при подстановке $z = 10$ в уравнение плоскости, несущей треугольник. Результат таков:

$$3x + y - 80 = 0$$

Пересечения этой прямой с ребрами треугольника происходят в точках (20, 20) и (22.5, 12.5). Линия пересечения, на которой треугольник становится видимым, хорошо отражена в буфере кадра.

Алгоритм, использующий $z$-буфер, можно также применить для построения сечений поверхностей. Изменится только оператор сравнения:

$$z(x, y) > Zбуфер(x, y) \textbf{ and } z(x, y) \leqslant Zсечения$$

где Zсечения — глубина искомого сечения. Эффект заключается в том, что остаются только такие элементы поверхности, которые лежат на самом сечении или позади него.

## 4.8. АЛГОРИТМЫ, ИСПОЛЬЗУЮЩИЕ СПИСОК ПРИОРИТЕТОВ

При реализации всех обсуждавшихся алгоритмов удаления невидимых линий и поверхностей устанавливались приоритеты, т. е. глубины объектов сцены или их расстояния от точки наблюдения. Алгоритмы, использующие список приоритетов, пытаются получить преимущество посредством предварительной сортировки по глубине или приоритету. Цель такой сортировки состоит в том, чтобы получить окончательный список элементов сцены, упорядоченных по приоритету глубины, основанному на расстоянии от точки наблюдения. Если такой список окончателен, то никакие два элемента не будут взаимно перекрывать друг друга. Тогда можно записать все элементы в буфер кадра поочередно, начиная с элемента, наиболее удаленного от точки наблюдения. Более близкие к наблюдателю элементы будут затирать информацию о более далеких элементах в буфере кадра. Поэтому задача об удалении невидимых поверхностей решается тривиально. Эффекты прозрачности можно включить в состав алгоритма путем не полной, а частичной корректировки содержимого буфера кадра с учетом атрибутов прозрачных элементов (см. [4-16] и разд. 5.8).

Для простых элементов сцены, например для многоугольников, этот метод иногда называют алгоритмом художника, поскольку он аналогичен тому способу, которым художник создает картину. Сначала художник рисует фон, затем предметы, лежащие на среднем расстоянии, и, наконец, передний план. Тем самым художник решает задачу об удалении невидимых поверхностей, или задачу видимости, путем построения картины в порядке обратного приоритета.

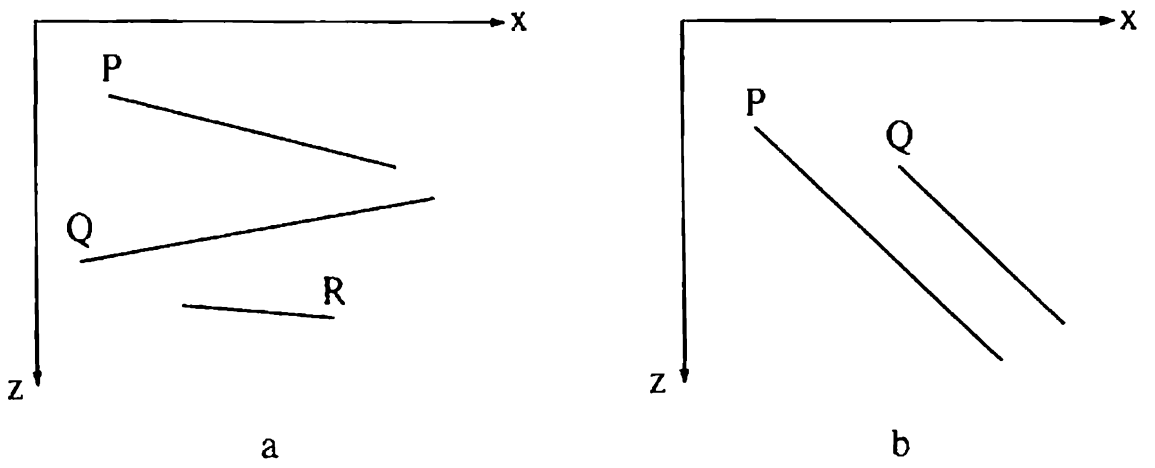



Рис. 4.51. Установление приоритетов для многоугольников.

Для простой сцены, вроде той, что показана на рис. 4.51, a, окончательный список приоритетов можно получить непосредственно. Например, эти многоугольники можно упорядочить по их максимальному или минимальному значению координаты $z$. Однако уже для сцены, показанной на рис. 4.51, b, окончательный список приоритетов по глубине невозможно получить простой сортировкой по $z$. Если $P$ и $Q$ с рис. 4.51, b упорядочены по минимальному значению координаты $z$ ($z_{min}$), окажется, что $P$ в списке приоритетов по глубине будет стоять перед $Q$. Если их записать в буфер кадра в таком порядке, то получится, что $Q$ частично экранирует $P$. Однако фактически $P$ частично экранирует $Q$. Правильный порядок в списке приоритетов будет тогда, когда $P$ и $Q$ поменяются местами.

Другие трудности показаны на рис. 4.52. Здесь многоугольники циклически перекрывают друг друга. На рис. 4.52, a $P$ находится впереди $Q$, который лежит впереди $R$, который в свою очередь находится впереди $P$. На рис. 4.52, b $P$ экранирует $Q$, а $Q$ экранирует $P$. Аналогичное циклическое экранирование возникает при протыкании многоугольников; например, на рис. 4.50 показано, как треугольник протыкает прямоугольник. Там прямоугольник экранируется треугольником, и наоборот. В обоих примерах окончательный список приоритетов невозможно установить сразу. Выход из положения заключается в циклическом разрезании многоугольников по линиям, образованным пересечениями их плоскостей до тех пор, пока не будет получен окончательный список приоритетов. Такие линии показаны пунктиром на рис. 4.52.

Ньюэл М., Ньюэл Р.и Санча [4-16] предложили специальный метод сортировки для разрешения конфликтов, возникающих при создании списка приоритетов по глубине. Этот метод включен в

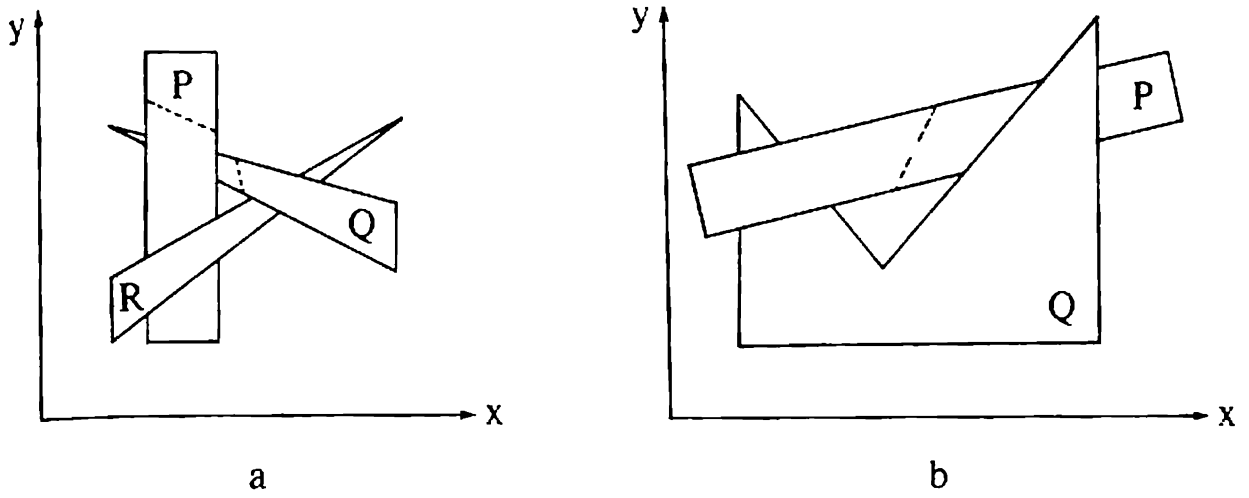



Рис. 4.52. Циклически перекрывающиеся многоугольники.

состав алгоритма Ньюэла — Ньюэла — Санча, который излагается ниже. В алгоритме динамически вычисляется новый список приоритетов перед обработкой каждого кадра сцены. Не накладывается никаких ограничений на сложность сцены и на тип многоугольников, используемых для описания элементов сцены. Первоначальный алгоритм Ньюэла — Ньюэла — Санча был предназначен для обработки трехмерных тел. Это расширение не ограничено рамками многогранников. Оно может, кроме того, обрабатывать тела смешанных типов в рамках одной сцены.

Алгоритм Ньюэла — Ньюэла — Санча для случая многоугольников

Сформировать предварительный список приоритетов по глубине, используя в качестве ключа сортировки значение $z_{min}$ для каждого многоугольника. Первым в списке будет многоугольник с минимальным значением $z_{min}$. Этот многоугольник лежит дальше всех от точки наблюдения, расположенной в бесконечности на положительной полуоси $z$. Обозначим его через $P$, а следующий в списке многоугольник — через $Q$.

Для каждого многоугольника $P$ из списка надо проверить его отношение с $Q$.

Если ближайшая вершина $P$ ($P_{z_{max}}$) будет дальше от точки наблюдения, чем самая удаленная вершина $Q$ ($Q_{z_{min}}$), т. е. $Q_{z_{min}} \geqslant P_{z_{max}}$, то никакая часть $P$ не может экранировать $Q$. Занести $P$ в буфер кадра (рис. 4.51, а).

Если $Q_{z_{min}} < P_{z_{max}}$, то $P$ потенциальное экранирует не только $Q$, но также и любой другой многоугольник типа $Q$ из списка, для

которого $Q_{z\,min} < P_{z\,max}$. Тем самым образуется множество $\{Q\}$. Однако $P$ может фактически и не экранировать ни один из этих многоугольников. Если последнее верно, то $P$ можно заносить в буфер кадра. Для ответа на этот вопрос используется серия тестов, следующих по возрастанию их вычислительной сложности. Эти тесты ниже формулируются в виде вопросов. Если ответ на любой вопрос будет положительным, то $P$ не может экранировать $\{Q\}$. Поэтому $P$ сразу же заносится в буфер кадра. Вот эти тесты:

Верно ли, что прямоугольные объемлющие оболочки $P$ и $Q$ не перекрываются по $x$?

Верно ли, что прямоугольные оболочки $P$ и $Q$ не перекрываются по $y$?

Верно ли, что $P$ целиком лежит по ту сторону плоскости, несущей $Q$, которая расположена дальше от точки наблюдения (рис. 4.53, a)?

Верно ли, что $Q$ целиком лежит по ту сторону плоскости, несущей $P$, которая ближе к точке наблюдения (рис. 4.53, b)?

Верно ли, что проекции $P$ и $Q$ не перекрываются?

Каждый из этих тестов применяется к каждому элементу $\{Q\}$. Если ни один из них не дает положительного ответа и не заносит $P$ в буфер кадра, то $P$ может закрывать $Q$.

Поменять $P$ и $Q$ местами, пометив позицию $Q$ в списке. Повторить тесты с новым списком. Это дает положительный результат для сцены с рис. 4.51, b.

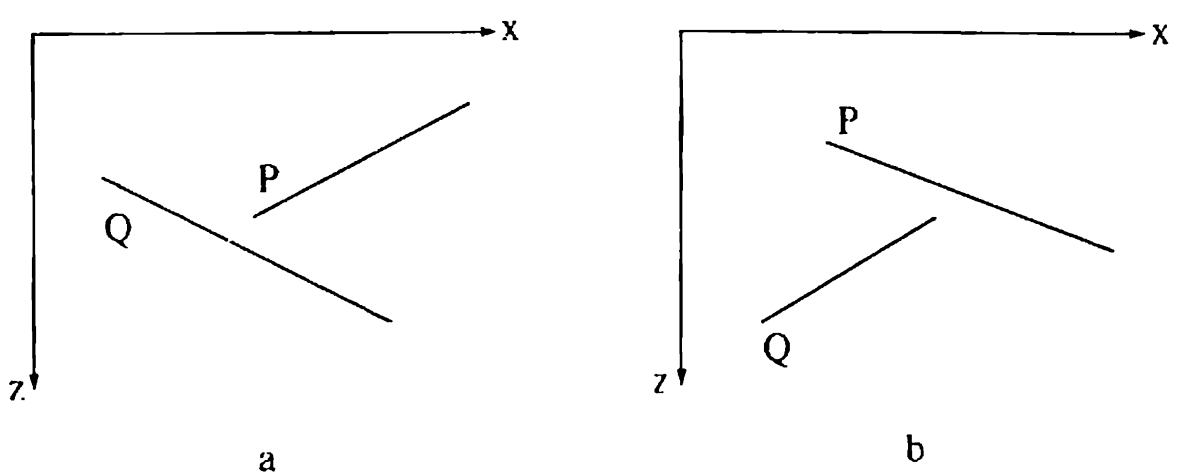



Рис. 4.53. Тесты для перекрывающихся многоугольников.

Если сделана попытка вновь переставить $Q$, значит, обнаружена ситуация циклического экранирования (рис. 4.52). В этом случае $P$ разрезается плоскостью, несущей $Q$, исходный многоугольник $P$ удаляется из списка, а его части заносятся в список. Затем тесты повторяются для нового списка. Этот шаг предотвращает зацикливание алгоритма.

Взятые вместе, первые два из приведенных выше вопросов образуют обычный габаритный тест для прямоугольных оболочек (см. разд. 2.13 и 3.1). Поскольку многие сцены не являются квадратными, то прямоугольные объемлющие оболочки будут с большей вероятностью перекрываться в одном из двух возможных направлений. Когда многоугольники преимущественно горизонтальны или вертикальны, то использование одного из этих двух тестов оказывается более эффективным. В алгоритме в той форме, в которой он записан выше, предполагается, что ширина сцены больше ее высоты, т. е. многоугольники преимущественно являются горизонтальными. Если высота сцены больше ее ширины, то тесты следует поменять местами. Если же сцена является квадратной или ее структура изоморфна, то порядок применения этих тестов не имеет значения.

Третий и четвертый тесты можно реализовать с использованием некоторых из ранее обсуждавшихся тестов видимости (см. разд. 3.16 и пример 3.23). Поскольку уравнение несущей плоскости или нормаль к ней часто известны для каждого многоугольника, то удобно применить простой тест подстановки. Если исследуется отношение многоугольника $Q$ к многоугольнику $P$, то координаты вершин $Q$ подставляются в уравнение плоскости, несущей $P$. Если все знаки результатов подстановки совпадают, то $Q$ целиком лежит по одну сторону от $P$. Здесь, так же как и в других алгоритмах удаления невидимых поверхностей, обсуждавшихся ранее, если это нужно, производится предварительное удаление нелицевых граней. Более полно эти соображения проиллюстрирует пример 4.20 для случая многоугольников, конфликтующих в пространстве.

**Пример 4.20.** Проверка отношений между многоугольниками, конфликтующими в пространстве

Рассмотрим три треугольника: $P$, $Q_1$ и $Q_2$ (рис. 4.54). Вершины этих треугольников таковы:

$P$: (1, 1, 1), (4, 5, 2), (5, 2, 5)
$Q_1$: (2, 2, 0.5), (3, 3, 1.75), (6, 1, 0.5)
$Q_2$: (0.5, 2, 5.5), (2, 5, 3), (4, 4, 5)

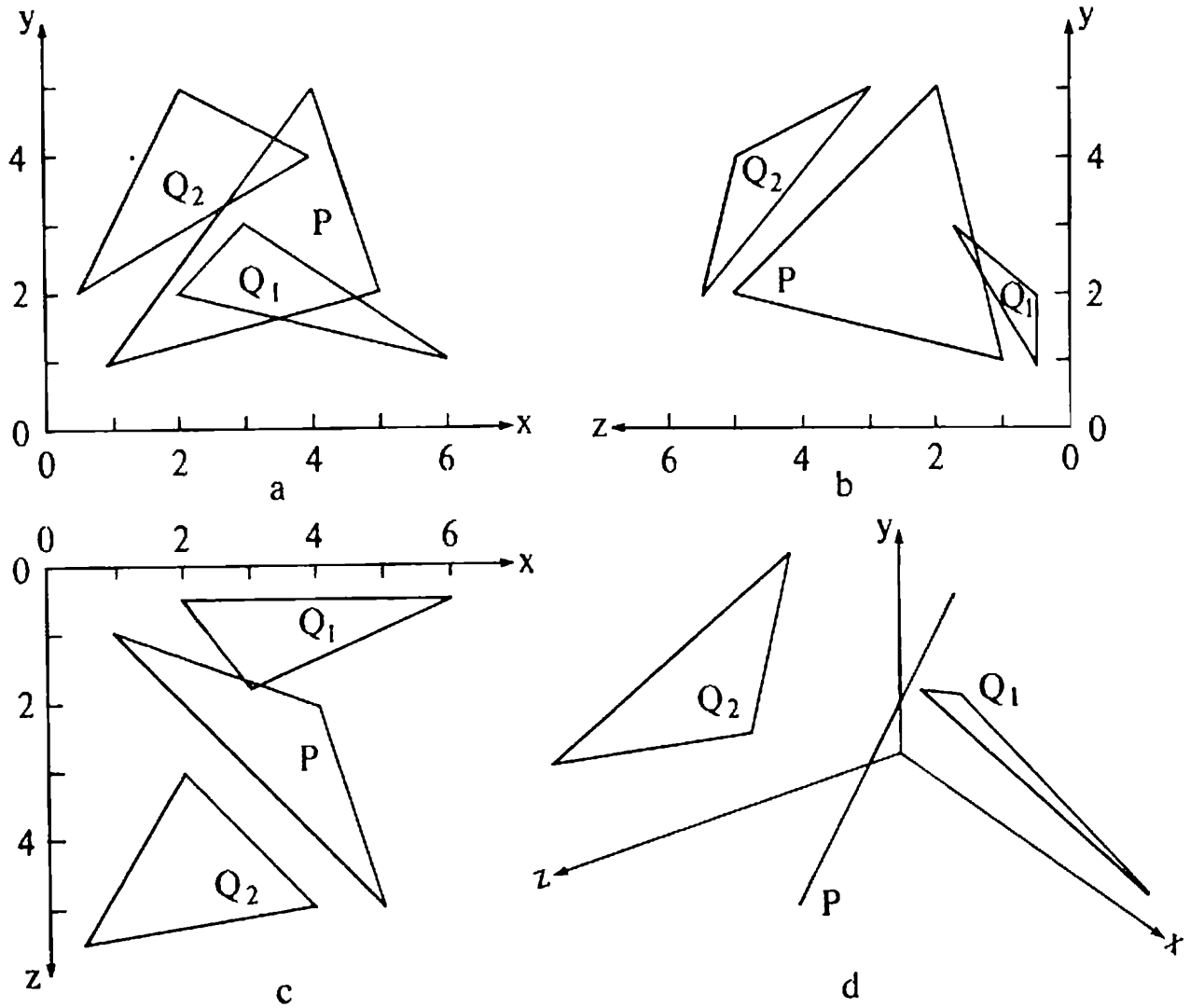



Рис. 4.54. Многоугольники, конфликтующие в пространстве.

Требуется определить, верно ли, что $Q_1$ и $Q_2$ целиком лежат по одну сторону от $P$. Три ортогональные проекции, показанные на рис. 4.54, не дают ясного ответа. Уравнение плоскости треугольника $P$ имеет вид

$$15x - 8y - 13z + 6 = 0$$

Пробная функция $T.F.$ равна

$$T.F. = 15x - 8y - 13z + 6$$

Подстановка вершин $Q_1$ в пробную функцию дает:

$$T.F._1 = 15(2) - 8(2) - 13(0.5) + 6 = 13.5 > 0$$
$$T.F._2 = 15(3) - 8(3) - 13(1.75) + 6 = 4.25 > 0$$
$$T.F._3 = 15(6) - 8(1) - 13(0.5) + 6 = 81.5 > 0$$

Поскольку знаки всех пробных функций положительны, то треугольник $Q_1$ целиком лежит по одну сторону от $P$.

Подставляя вершины $Q_2$ в пробную функцию, имеем:

$$T.F._4 = 15(0.5) - 8(2) - 13(5.5) + 6 = -74 < 0$$

$$T.F._5 = 15(2) \quad 8(5) - 13(3) \quad 6 = -43 < 0$$
$$T.F._6 = 15(4) - 8(4) \quad 13(5) + 6 = -31 < 0$$

Вновь знаки всех пробных функций совпали; поэтому треугольник $Q_2$ целиком лежит по одну сторону от $P$.

На рис. 4.54, d хорошо видно, что $Q_1$ лежит по ту сторону от плоскости треугольника $P$, которая дальше от точки наблюдения, расположенной в бесконечности на положительной полуоси $z$. Следовательно, $Q_1$ частично экранируется $P$. На рис. 4.54, d также хорошо видно, что $Q_2$ лежит по ту сторону от плоскости треугольника $P$, которая ближе к указанной точке наблюдения. Поэтому $Q_2$ частично экранирует $P$.

Из приведенного примера следует, что

Если знаки пробной функции для всех вершин некоего многоугольника совпадают и положительны или равны нулю, то этот многоугольник находится с дальней (невидимой) стороны от плоскости $P$.

Если знаки пробной функции для всех вершин некоего многоугольника совпадают и отрицательны или равны нулю, то этот многоугольник расположен с ближней (видимой) стороны от плоскости $P$.

Если значения пробной функции для каждой вершины многоугольника равны нулю, то этот многоугольник лежит на плоскости $P$.

Последний тест из приведенной серии особенно сложен с вычислительной точки зрения, поскольку требует точного определения того факта, что проекции $P$ и $Q$ не пересекаются. Метод решения этой задачи уже обсуждался ранее в рамках описания алгоритма Варнока (см. разд. 4.4).

Если имеет место циклическое экранирование, то для разбиения многоугольников вдоль линии пересечения несущих эти многоугольники плоскостей можно воспользоваться алгоритмом отсечения многоугольников Сазерленда — Ходжмена (см. разд. 3.16). Здесь плоскость, несущая $Q$, используется как секущая. Каждое ребро $P$ отсекается плоскостью $Q$, при этом формируются два новых многоугольника. Для поиска пересечения каждого ребра $P$ с плоскостью $Q$ можно воспользоваться алгоритмом отсечения Кируса — Бека (см. разд. 3.11).

В алгоритме Ньюэла — Ньюэла — Санча делается попытка решить задачу удаления невидимых поверхностей динамически путем обработки всех многоугольников сцены для каждого искомого кадра. Если же сцена сложна, а частота кадров велика, как у систем

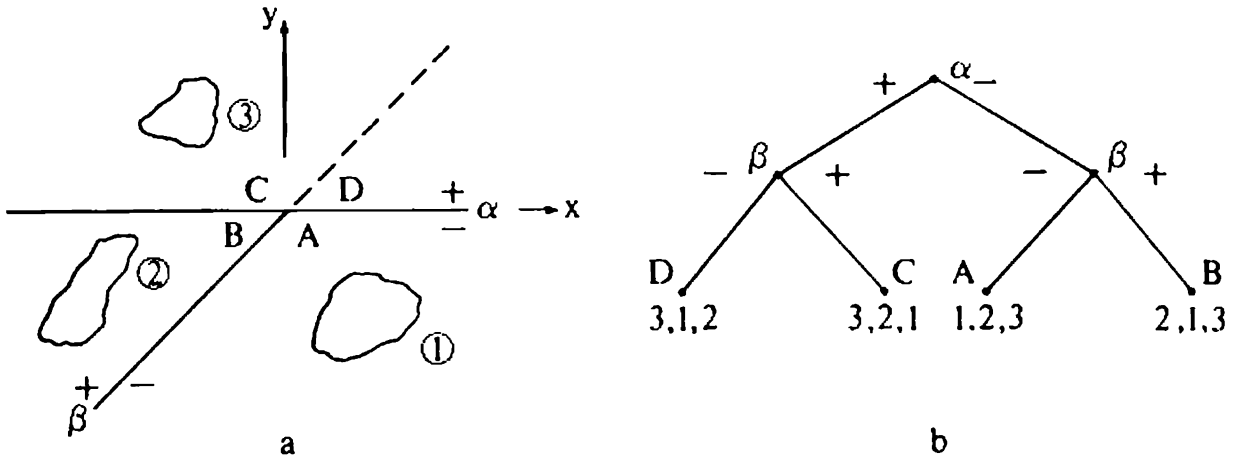



Рис. 4.55. Кластерные приоритеты.

моделирования в реальном времени, то вычислительной мощности обычных универсальных ЭВМ может не хватить (см. [4-18]). Однако во многих случаях моделирования в реальном времени, например при имитации посадки самолета, сцена статична, а меняется только точка наблюдения. Шумейкер и др. [4-19] удачно предложили предварительно вычислять в автономном режиме некоторые более общие приоритетные характеристики, такие, как список приоритетов для моделирования упомянутых выше статических сцен.

В алгоритме Шумейкера в сцене допустимы только выпуклые многоугольники. Такие многоугольники группируются в кластеры, которые являются линейно разделимыми. Кластеры считаются линейно разделимыми, если существует такая разделяющая плоскость, которая проходит между ними, не пересекая их. Несколько двумерных кластеров показано на рис. 4.55, а. Разделяющие плоскости помечены буквами $\alpha$ и $\beta$. Они разбивают сцену на четыре области $A$, $B$, $C$, $D$. Точка наблюдения может располагаться в любой из этих областей. На рис. 4.55, b показана древовидная структура, устанавливающая приоритеты кластеров в сцене. Эти приоритеты можно вычислить заранее для любой точки наблюдения в двумерной плоскости. Подстановка координат точки наблюдения в уравнения разделяющих плоскостей порождает соответствующий узел дерева кластерных приоритетов. Затем для каждого кластера в обратном порядке приоритетов решается задача удаления невидимых поверхностей.

**Пример 4.21.** Кластерные приоритеты

Предположим, что разделяющие плоскости $\alpha$ и $\beta$, показанные на рис. 4.55, пересекаются в начале координат. Далее предположим, что $\alpha$ — это плоскость $y = 0$, а $\beta$ — плоскость $y = x$; обе плоскости перпендикулярны плоскости чертежа. Уравнения этих плоскостей и соответствующие им пробные функции таковы:

$$\alpha: \quad y = 0 \qquad (F.F.)_1 = y$$
$$\beta: \quad y - x = 0 \qquad (F.F.)_2 = y - x$$

Точка наблюдения, лежащая на прямой $2y - x = 0$, например (20, 10), дает:

$$(F.F.)_1 = 10 > 0$$
$$(F.F.)_2 = 10 - 20 = -10 < 0$$

Значит, эта точка наблюдения лежит в области $D$. Из рис. 4.55, b видно, что список кластерных приоритетов таков: 3, 1, 2.

Кластеры используются для разбиения сцены. Простейшим кластером является единственный многоугольник. Кластеры могут представлять собой сложные полигональные или неполигональные поверхности и тела, для каждого типа которых существуют свои методы удаления невидимых линий, как показано Ньюэлом [4-17].

В рамках кластеров некоторых типов приоритеты отдельных многоугольников не зависят от положения точки наблюдения [4-19, 4-20]. Это обстоятельство является одним из основных достоинств алгоритма Шумейкера. Оно позволяет заранее вычислять полный список приоритетов. На рис. 4.56, a показан двумерный кластер, для которого можно заранее вычислить индивидуальные приоритеты многоугольников. Приоритет каждого многоугольника устанавливается в зависимости от возможности экранирования данным многоугольником любого другого многоугольника для произвольной точки наблюдения. Чем большее число многоугольников может экранировать данный многоугольник, тем выше его приоритет. При установлении приоритетов многоугольников в рамках кластера для заданной точки наблюдения прежде всего удаляются нелицевые многоугольники. Оставшиеся многоугольники располагаются затем в приоритетном порядке, как показано на рис. 4.56, b и c.

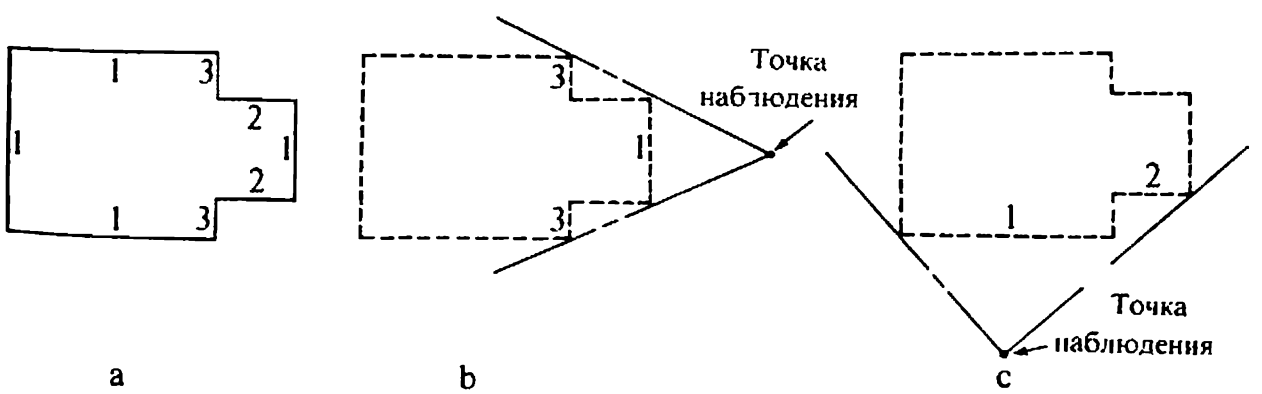



Рис. 4.56. Приоритеты в рамках кластера.

Алгоритмы, использующие список приоритетов, работают как в пространстве объекта, так и в пространстве изображения. В частности, формирование списка приоритетов ведется в пространстве объекта, а результат заносится в буфер кадра в терминах пространства изображения. Использование буфера кадра является критическим для данного алгоритма.

Поскольку подобно алгоритмам Варнока и $z$-буфера в алгоритмах, строящих список приоритетов, многоугольники обрабатываются в произвольном порядке, применение методов устранения лестничного эффекта к результирующему изображению затруднено. Однако для этой цели здесь применим метод постфильтрации (см. разд. 2.25), используемый также в алгоритмах Варнока и $z$-буфера.

Алгоритмы, строящие список приоритетов, использующие $z$-буфер и относящиеся к типу Варнока, могут также применяться и для удаления невидимых линий. При их использовании в этом качестве ребра каждого многоугольника заносятся в буфер кадра с одним атрибутом, при чем внутренняя область каждого многоугольника заносится в буфер кадра с атрибутом фона. При таком подходе многоугольники, которые находятся ближе к точке наблюдения, «заслоняют» ребра многоугольников, которые расположены дальше от нее.

## 4.9. АЛГОРИТМЫ ПОСТРОЧНОГО СКАНИРОВАНИЯ

Алгоритмы Варнока, $z$-буфера и строящий список приоритетов обрабатывают элементы сцены или многоугольники в порядке, корый не связан с процессом визуализации. В то же время алгоритмы построчного сканирования, впервые предложенные Уайли и др. [4-21], Букнайтом [4-22, 4-23, 4-24] и Уоткинсом [4-25], обрабатывают сцену в порядке прохождения сканирующей прямой. Эти алгоритмы оперируют в пространстве изображения.

Растровая развертка отдельных многоугольников обсуждалась в гл. 2. Алгоритмы удаления невидимых линий и поверхностей с построчным сканированием являются обобщением изложенных там методов. Эти алгоритмы сводят трехмерную задачу удаления невидимых линий и поверхностей к двумерной. Сканирующая плоскость определяется точкой наблюдения, расположенной в бесконечности на положительной полуоси $z$, и сканирующей строкой так, как показано на рис. 4.57. Пересечение сканирующей плоскости и трехмерной сцены определяет окно размером в одну сканирующую строку. Задача удаления невидимых поверхностей решается в преде-

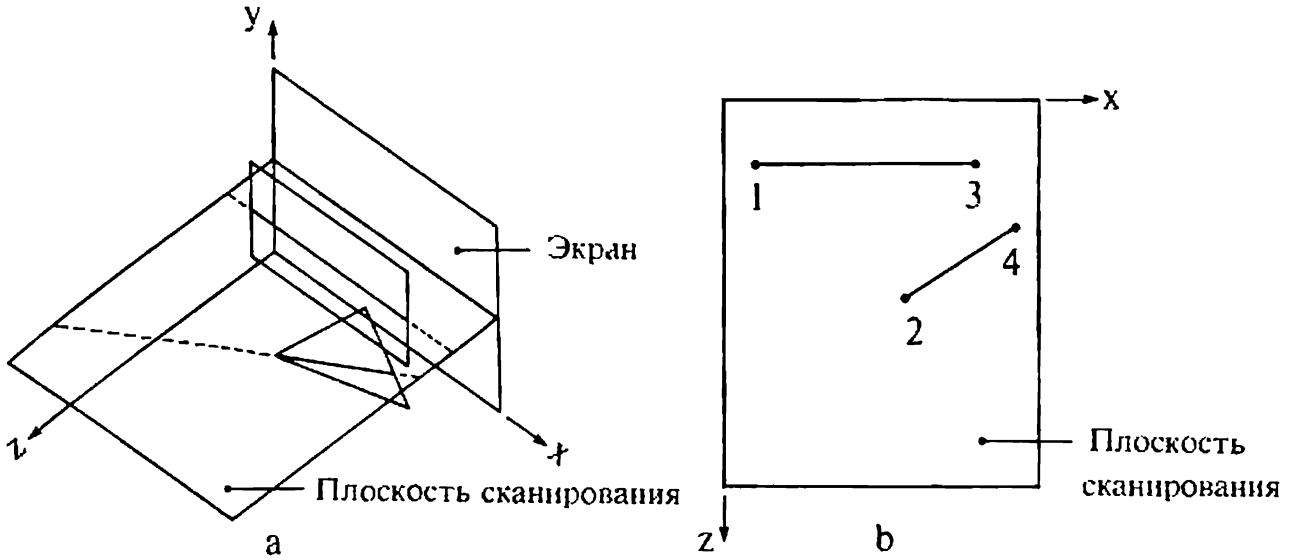



Рис. 4.57. Сканирующая плоскость.

лах этого окна, образованного сканирующей плоскостью. На рис. 4.57, b представлено пересечение сканирующей плоскости с многоугольниками. Этот рисунок показывает, что задача удаления невидимых поверхностей сводится к задаче о том, какой отрезок видим для каждой точки сканирующей строки.

На первый взгляд может показаться, что здесь можно непосредственно применить алгоритм формирования упорядоченного списка ребер, обсуждавшийся в разд. 2.19. Однако из рис. 4.57, b ясно видно, что это приведет к некорректным результатам. Например, для сканирующей строки, показанной на рис. 4.57, существуют четыре активных ребра в списке активных ребер. Пересечения этих ребер со сканирующей строкой показаны точками на рис. 4.57, b; список упорядоченных ребер обозначен цифрами. Попарная выборка пересечений приводит к тому, что пикселы между точками 1 и 2, а также между точками 3 и 4 активируются. В то же время пикселы между точками 2 и 3 не активируются. Таким образом, получается неверный результат. «Дыра» образовалась в том месте, где сканирующая строка фактически пересекла два многоугольника. В следующих двух разделах обсуждаются два корректных алгоритма построчного сканирования.

## 4.10. АЛГОРИТМ ПОСТРОЧНОГО СКАНИРОВАНИЯ, ИСПОЛЬЗУЮЩИЙ $z$-БУФЕР

Одним из простейших алгоритмов построчного сканирования, который решает задачу удаления невидимых поверхностей, является специальный случай алгоритма $z$-буфера, который обсуждался в

предыдущем разделе. Будем называть его алгоритмом построчного сканирования с $z$-буфером [4-26]. Используемое в этом алгоритме окно визуализации имеет высоту в одну сканирующую строку и ширину во весь экран. Как для буфера кадра, так и для $z$-буфера требуется память высотой в 1 бит, шириной в горизонтальное разрешение экрана и глубиной в зависимости от требуемой точности. Обеспечиваемая точность по глубине зависит от диапазона значений, которые может принимать $z$. Например, буфер кадра может иметь размер $1\times512\times24$ бит, а $z$-буфер — $1\times512\times20$ бит.

Концептуально этот алгоритм достаточно прост. Для каждой сканирующей строки буфер кадра инициализируется с фоновым значением интенсивности, а $z$-буфер — с минимальным значением $z$. Затем определяется пересечение сканирующей строки с двумерной проекцией каждого многоугольника сцены, если они существуют. Эти пересечения образуют пары, как указывалось в разд. 2.19. При рассмотрении каждого пиксела на сканирующей строке в интервале между концами пар его глубина сравнивается с глубиной, содержащейся в соответствующем элементе $z$-буфера. Если глубина этого пиксела больше глубины из $z$-буфера, то рассматриваемый отрезок будет текущим видимым отрезком. И следовательно, атрибуты многоугольника, соответствующего данному отрезку, заносятся в буфер кадра в позиции данного пиксела; соответственно корректируется и $z$-буфер в этой позиции. После обработки всех многоугольников сцены буфер кадра размером в одну сканирующую строку содержит решение задачи удаления невидимых поверхностей для данной сканирующей строки. Он выводится на экран дисплея в порядке, определяемом растровым сканированием, т. е. слева направо. В этом алгоритме можно использовать методы устранения ступенчатости, основывающиеся как на пре-, так и на постфильтрации.

Однако практически сравнение каждого многоугольника с каждой сканирующей строкой оказывается неэффективным. Поэтому используется некоторая разновидность списка упорядоченных ребер, которая обсуждалась в разд. 2.19. В частности, для повышения эффективности этого алгоритма применяются групповая сортировка[1)] по оси $y$, список активных многоугольников и список активных ребер.

---

[1)] Групповая сортировка является одной из разновидностей распределяющей сортировки (см. Кнут Д. Искусство программирования для ЭВМ. Том 3. Сортировка и поиск. — М.: Мир, 1978). — *Прим. ред.*

С использованием этих методов алгоритм построчного сканирования с $z$-буфером формулируется следующим образом:

Подготовка информации:

Для каждого многоугольника определить самую верхнюю сканирующую строку, которую он пересекает.

Занести многоугольник в группу $y$, соответствующую этой сканирующей строке.

Запомнить для каждого многоугольника, например в связном списке, как минимум следующую информацию: $\Delta y$ — число сканирующих строк, которые пересекаются этим многоугольником; список ребер многоугольника; коэффициенты $(a, b, c, d)$ уравнения плоскости многоугольника; визуальные атрибуты многоугольника.

Решение задачи удаления невидимых поверхностей:

Инициализировать буфер кадра дисплея.

Для каждой сканирующей строки:

Инициализировать буфер кадра размером с одну сканирующую строку, заполнив его фоновым значением.

Инициализировать $z$-буфер размером с одну сканирующую строку значением $z_{\min}$.

Проверить появление в группе $y$ сканирующей строки новых многоугольников. Добавить все новые многоугольники к списку активных многоугольников.

Проверить появление новых многоугольников в списке активных многоугольников. Добавить все пары ребер новых многоугольников к списку активных ребер.

Если какой-нибудь элемент из пары ребер многоугольника удаляется из списка активных ребер, то надо определить, сохранился ли соответствующий многоугольник в списке активных многоугольников. Если сохранился, то укомплектовать пару активных ребер этого многоугольника в списке активных ребер. В противном случае удалить и второй элемент пары ребер из списка активных ребер.

В списке активных ребер содержится следующая информация для каждой пары ребер многоугольника, которые пересекаются сканирующей строкой:

$x_л$ — пересечение левого ребра из пары с текущей сканирующей строкой.

$\Delta x_л$ — приращение $x_л$ в интервале между соседними сканирующими строками.

$\Delta y_л$ — число сканирующих строк, пересекаемых левой стороной.

$x_п$ — пересечение правого ребра из пары с текущей сканирующей строкой.

$\Delta x_п$ — приращение $x_п$ в интервале между соседними сканирующими строками.

$\Delta y_п$ — число сканирующих строк, пересекаемых правой стороной.

$z_л$ — глубина многоугольника в центре пиксела, соответствующего левому ребру.

$\Delta z_x$ — приращение по $z$ вдоль сканирующей строки. Оно равно $a/c$ при $c \neq 0$.

$\Delta z_y$ — приращение по $z$ в интервале между соседними сканирующими строками. Оно равно $b/c$ при $c \neq 0$.

Пары активных ребер многоугольников заносятся в соответствующий список в произвольном порядке. В пределах одной пары пересечения упорядочены слева направо. Для одного многоугольника может оказаться более одной пары активных ребер[1].

Для каждой пары ребер многоугольника из списка активных ребер:

Извлечь эту пару ребер из списка активных ребер.

Инициализировать $z$ со значением $z_л$.

Для каждого пиксела, такого, что $x_л \leqslant x + 1/2 \leqslant x_п$, вычислить глубину $z(x + 1/2, y + 1/2)$ в его центре, используя уравнение плоскости многоугольника. Для сканирующей строки это сведется к вычислению приращения: $z_{x+\Delta x} = z_x - \Delta z_x$.

Сравнить глубину $z(x + 1/2, y + 1/2)$ с величиной $Z$ буфер$(x)$, хранящейся в $z$-буфере для одной сканирующей

---

[1] В силу его невыпуклости.— *Прим. перев.*

строки. Если $z(x + 1/2, y + 1/2) > Z\text{буфер}(x)$, то занести атрибуты многоугольника в буфер кадра для одной сканирующей строки и заменить $Z\text{буфер}(x)$ на $z(x + 1/2, y + 1/2)$. В противном случае не производить никаких действий.

Записать буфер кадра для сканирующей строки в буфер кадра дисплея.

Скорректировать список активных ребер:

Для каждой пары ребер многоугольника определить $\Delta y_л$ и $\Delta y_п$. Если $\Delta y_л$ или $\Delta y_п < 0$, то удалить соответствующее ребро из списка. Пометить положение обоих ребер в списке и породивший их многоугольник.

Вычислить новые абсциссы пересечений:

$$x_{\text{лнов}} = x_{\text{лстар}} + \Delta x_л$$
$$x_{\text{пнов}} = x_{\text{пстар}} + \Delta x_п$$

Вычислить глубину многоугольника на левом ребре, используя уравнение плоскости этого многоугольника. Между сканирующими строками это сведется к вычислению приращения:

$$z_{\text{лнов}} = z_{\text{лстар}} - \Delta z_x \Delta x - \Delta z_y$$

Сократить список активных многоугольников. Если $\Delta y < 0$ для какого-нибудь многоугольника, то удалить его из списка.

Как и раньше используется предварительное удаление нелицевых плоскостей. Следующий пример послужит более полной иллюстрацией этого алгоритма.

**Пример 4.22.** Алгоритм построчного сканирования с $z$-буфером

Вновь возьмем прямоугольник и треугольник, ранее рассмотренные в примере 4.19. Напомним координаты углов прямоугольника: $P_1(10, 5, 10)$, $P_2(10, 25, 10)$, $P_3(25, 25, 10)$, $P_4(25, 5, 10)$ и вершин треугольника: $P_5(15, 15, 15)$, $P_6(25, 25, 5)$, $P_7(30, 10, 5)$, показанных на рис. 4.50. Разрешение экрана осталось равным $32 \times 32 \times 2$ бита. Как и раньше, атрибут видимости фона будет равен 0, прямоугольника — 1, а треугольника — 2. Точка наблюдения находится в бесконечности на положительной полуоси $z$. Используя соглашение о половине интервала между сканирующими строками, видим, что самая верхняя сканирующая строка, пересекающая оба многоугольника, имеет $y = 24$. Поэтому только группа с $y = 24$ содержит какую-то информацию. Все остальные группы пусты.

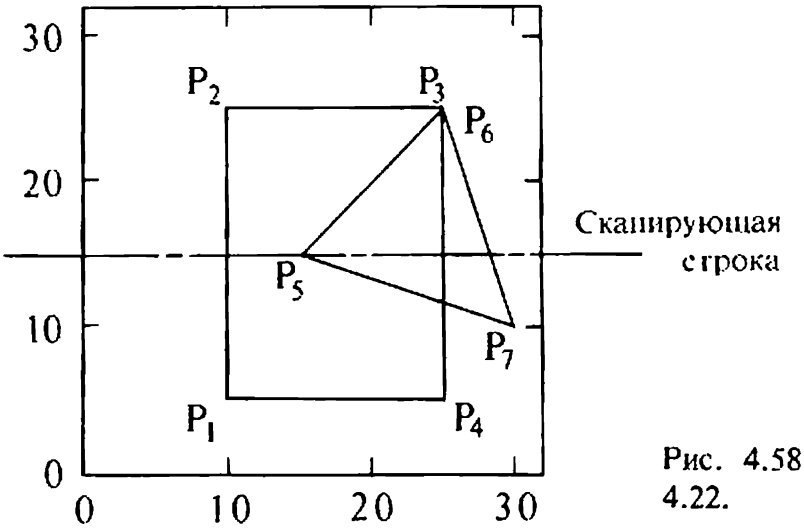



Рис. 4.58. Многоугольники из примера 4.22.

Список активных многоугольников при $y = 24$ содержит для прямоугольника (многоугольник 1) и треугольника (многоугольник 2) следующую информацию:

прямоугольник: 19, 2, $P_1P_2$, $P_3P_4$, 0, 0, 1, −10, 1;
треугольник: 14, 3, $P_5P_6$, $P_6P_7$, $P_7P_5$, 3, 1, 4, −120, 2.

Элементы этого списка суть соответственно $\Delta y$, число ребер, список ребер, коэффициенты уравнений несущей плоскости ($a$, $b$, $c$, $d$) и номер многоугольника. Заметим, что для прямоугольника этот список содержит только два ребра. Горизонтальные ребра игнорируются.

Для сканирующей строки 15 (рис. 4.58) в списке активных многоугольников содержатся оба многоугольника. Для прямоугольника $\Delta y = 11$. Для треугольника $\Delta y = 5$. Вначале список активных ребер содержит две пары пересечений; первую — для прямоугольника и вторую — для треугольника:

прямоугольник: 10, 0, 19, 25, 0, 19, 10, 0, 0
треугольник: 24½, −1, 9, 25⅙, ⅓, 14, 5½, ¾, ¼

Элементы этого списка суть соответственно $x_л$, $\Delta x_л$, $\Delta y_л$, $x_п$, $\Delta x_п$, $\Delta y_п$, $z_л$, $\Delta z_x$, $\Delta z_y$. Непосредственно перед обработкой сканирующей строки 15 список активных ребер содержит:

прямоугольник: 10, 0, 10, 25, 0, 10, 10, 0, 0
треугольник: 15½, −1, 0, 28⅙, ⅓, 5, 14½, ¾, ¼

После первого обнуления буфера кадра и $z$-буфера для одной строки и последующей растровой развертки прямоугольника эти буферы будут содержать:

Буфер кадра для строки

0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0

$z$-буфер для строки

0 0 0 0 0 0 0 0 0 0 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 0 0 0 0 0 0 0

Рассмотрим теперь треугольник. На левом ребре $z = 14.5$, что больше значения Zбуфер(15) = 10. Поэтому атрибуты треугольника заносятся в буфер кадра, корректируется и $z$-буфер для сканирующей строки. Эти результаты после растровой

развертки треугольника таковы:

Буфер кадра для строки

0 0 0 0 0 0 0 0 0 0 1 1 1 1 2 2 2 2 2 2 2 1 1 1 1 2 2 0 0 0 0 0

z-буфер для строки

0 0 0 0 0 0 0 0 0 0 10 10 10 10 15 14 13 12 12 11 10 10 10 10 10 6 6 0 0 0 0 0

Здесь значения в z-буфере округлены до ближайших целых для экономии памяти. Тот же результат получился для соответствующей сканирующей строки и в примере 4.19. Для визуализации буфер кадра копируется слева направо.

Теперь корректируется список активных ребер. В результате его сокращения $\Delta y_л = -1 < 0$. Поэтому ребро $P_6P_5$ удаляется из списка активных ребер, а треугольник помечается. Корректировка правого ребра треугольника дает:

$$x_{\text{пнов}} = x_{\text{пстар}} + \Delta x_п = 28\frac{1}{6} + \frac{1}{3} = 28\frac{1}{2}$$
$$\Delta y_{\text{пнов}} = \Delta y_{\text{пстар}} - 1 = 5 - 1 = 4$$

После корректировки списка активных ребер сокращается список активных многоугольников. Поскольку прямоугольник остается в этом списке, то следующий цикл алгоритма вставит ребро $P_5P_7$ в список активных ребер в помеченной позиции. Пересечение сканирующей строки 14 ($y = 14.5$) с ребром $P_5P_7$ дает новое значение $x_л = 16\ 1\ 2$. Глубина треугольника равна:

$$z_л = -[ax + by + d]/c = -[(3)(16.5) + (1)(14.5) - 120]/4 = 14$$

Окончательный список активных ребер для сканирующей строки 14 таков:

прямоугольник: 10, 0, 9, 25, 0, 9, 10, 0, 0

треугольник: 16½, 3, 4, 25½, 4, 14, ¾, ¼

Полные результаты приведены в примере 4.19.

## 4.11. ИНТЕРВАЛЬНЫЙ АЛГОРИТМ ПОСТРОЧНОГО СКАНИРОВАНИЯ

В алгоритме построчного сканирования с использованием z-буфера глубина многоугольника вычисляется для каждого пиксела на сканирующей строке. Количество вычислений глубины можно уменьшить, если использовать понятие интервалов, впервые введенных в алгоритме Ваткинса [4-25]. На рис. 4.59, а показано пересечение двух многоугольников со сканирующей плоскостью. Решение задачи удаления невидимых поверхностей сводится к выбору видимых отрезков в каждом из интервалов, полученных путем деления сканирующей строки проекциями точек пересечения ребер (рис. 4.59, а). Из этого рисунка видно, что возможны только три варианта:

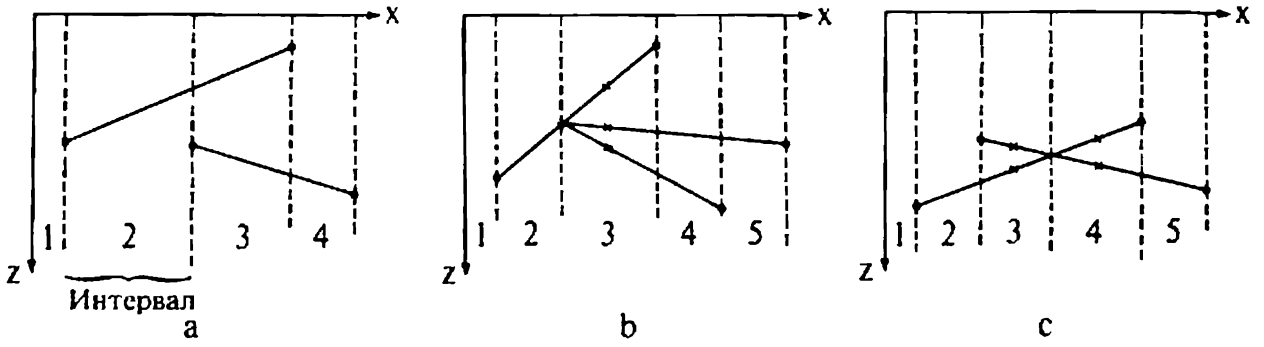



Рис. 4.59. Интервалы для построчного сканирования.

Интервал пуст, как, например, интервал 1 на рис. 4.59, a. В этом случае изображается фон.

Интервал содержит лишь один отрезок, как, например, интервалы 2 и 4 на рис. 4.59, a. В этом случае изображаются атрибуты многоугольника, соответствующего отрезку.

Интервал содержит несколько отрезков, как, например, интервал 3 на рис. 4.59, a. В этом случае вычисляется глубина каждого отрезка в интервале. Видимым будет отрезок с максимальным значением $z$. В этом интервале будут изображаться атрибуты видимого отрезка.

Если многоугольники не могут протыкать друг друга, то достаточно вычислять глубину каждого отрезка в интервале на одном из его концов. Если два отрезка касаются, но не проникают в концы интервала, то вычисление глубины производится в середине интервала, как показано на рис. 4.59, b. Для интервала 3 вычисление глубины, проведенное на его левом конце, не позволяет принять определенное решение. Реализация вычисления глубины в центре интервала дает уже корректный результат, как показано на рис. 4.59, b.

Если многоугольники могут протыкать друг друга, то сканирующая строка делится не только проекциями точек пересечения ребер со сканирующей плоскостью, но и проекциями точек их попарного пересечения, как показано на рис. 4.59, c. Вычисление глубины в концевых точках таких интервалов будет давать неопределенные результаты. Поэтому здесь достаточно проводить вычисление глубины в середине каждого интервала, как показано для оси $x$ на рис. 4.59, c.

Используя более сложные методы порождения интервалов, можно сократить их число, а следовательно, и вычислительную трудоемкость. Однако применение простых средств также может привести к удивительным результатам. Например, Ваткинс [4-25]

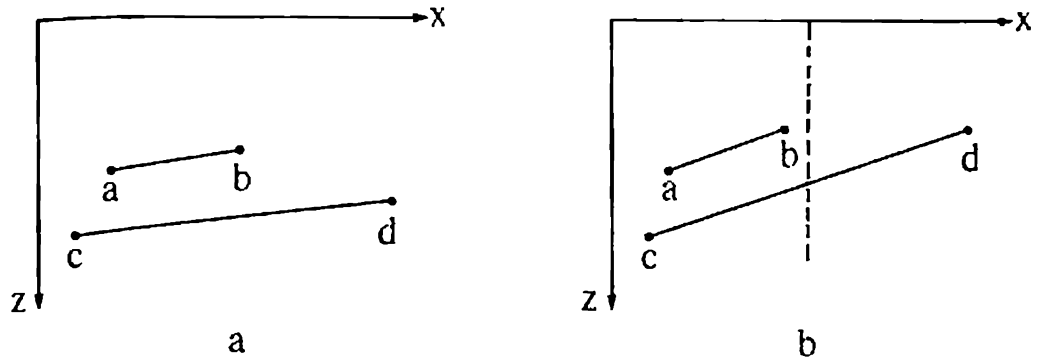



Рис. 4.60. Другой интервальный метод.

предложил простой метод разбиения отрезка средней точкой. Простым сравнением глубин концевых точек отрезков *ab* и *cd*, изображенных на рис. 4.60, a, убеждаемся, что отрезок *cd* целиком видим. Однако случай, изображенный на рис. 4.60, b, показывает, что так бывает не всегда[1]. Деление этой сцены прямой, проходящей через середину отрезка *cd*, сразу делает очевидным, что оба куска *cd* видимы.

Далее, иногда удается вообще избежать вычислений глубины. Ромни и др. [4-27] показали, что если многоугольники *не могут* протыкать друг друга и если текущая сканирующая строка пересекает те же самые многоугольники, что и предыдущая, и если порядок следования точек пересечения ребер при этом не изменится, то не изменяется и приоритет глубины отрезков в пределах каждого интервала. Значит, не нужно вычислять приоритет глубины отрезков для новой сканирующей строки. Хэмлин и Джир [4-28] показали, как при некоторых условиях можно поддерживать приоритет глубины даже тогда, кога порядок следования точек пересечения ребер изменяется.

Основная структура алгоритма построчного сканирования с $z$-буфером применима также в случае интервального алгоритма построчного сканирования типа Уоткинса. Необходимо изменить только внутренний цикл, т. е. способ обработки отдельной сканирующей строки и содержимое списка активных ребер. Здесь не нужно следить за парностью пересечений ребер многоугольника со сканирующей строкой. Ребра заносятся в список активных ребер индивидуально. Этот список упорядочивается по возрастанию $x$. Для определения левого и правого ребер из пары здесь используются идентификатор многоугольника и флаг активности многоугольни-

[1] Поскольку глубины точек *b* и *d* здесь равны.— *Прим. перев.*

ка. В начале сканирующей строки значение флага активности многоугольника равно *нулю*, и оно модифицируется при обработке каждого очередного ребра соответствующего многоугольника. Появление левого ребра многоугольника приведет к тому, что этот флаг станет равным *единице*, а встреча правого ребра вернет ему значение *нуль*. Пример 4.23, который приводится ниже, более подробно проиллюстрирует использование этого флага.

Интервалы, занимаемые каждым многоугольником, можно определять по мере обработки сканирующих строк. Если многоугольники не могут протыкать друг друга, то пересечение каждого ребра из списка активных ребер определяет границу интервала. Как указывалось выше, число активных многоугольников в пределах интервала определяет способ его обработки. Вычисление глубины проводится только тогда, когда в интервале содержится более одного активного многоугольника. Если же протыкание допустимо и в пределах интервала, определенного пересечением ребер, имеется более чем один активный многоугольник, то необходимо проверить возможность пересечения отрезков внутри этого интервала (рис. 4.59, с). Для осуществления этой проверки удобен метод, заключающийся в сравнении знаков разностей глубин пар отрезков в концевых точках интервала. Необходимо проверить каждую пару отрезков в таком интервале. Например, если глубины двух отрезков на левом и правом концах интервала равны $z_{1_л}$, $z_{1_п}$, $z_{2_л}$, $z_{2_п}$, то

$$\textbf{if sign}(z_{1_л} - z_{2_л}) \neq \textbf{sign}(z_{1_п} - z_{2_п}) \tag{4.9}$$

значит, эти отрезки пересекаются. Если отрезки пересекаются, то интервал подразбивается точкой пересечения. Этот процесс повторяется с левым подынтервалом до тех пор, пока он не станет свободным от пересечений. Для свободных интервалов вычисления глубин производятся в их серединах.

Если один из знаков равен нулю, то отрезки пересекаются в конце интервала. В таком случае достаточно определить глубину на противоположном конце интервала вместо того, чтобы проводить его разбиение.

Структура интервального алгоритма построчного сканирования такова:

Подготовка данных:

Определить для каждого многоугольника самую верхнюю сканирующую строку, пересекающую его.

Занести многоугольник в группу $y$, соответствующую этой сканирующей строке.

Запомнить в связном списке для каждого многоугольника как минимум следующую информацию: $\Delta y$ — число сканирующих строк, которые пересекаются этим многоугольником; список ребер многоугольника; коэффициенты уравнения несущей его плоскости ($a$, $b$, $c$, $d$); визуальные атрибуты многоугольника.

Решение задачи удаления невидимых поверхностей:

Для каждой сканирующей строки:

Проверить появление в группе $y$ сканирующей строки новых многоугольников. Добавить все новые многоугольники к списку активных многоугольников.

Проверить появление новых многоугольников в списке активных многоугольников. Добавить все ребра новых активных ребер. Для каждого ребра многоугольника, которое пересекается со сканирующей строкой, содержится следующая информация:

$x$ — абсцисса пересечения ребра с текущей сканирующей строкой.

$\Delta x$ — приращение по $x$ между соседними сканирующими строками.

$\Delta y$ — число сканирующих строк, пересекаемых данным ребром.

$P$ — идентификатор многоугольника.

*Флаг* — признак, показывающий активность данного многоугольника на текущей сканирующей строке.

Упорядочить список активных ребер по возрастанию $x$.

Обработать список активных ребер. Подробности обработки даны на блок-схеме, приведенной на рис. 4.61, и ее модификациях на рис. 4.62 и 4.63.

Скорректировать список активных ребер:

Для каждого пересечения ребра определить $\Delta y$. Если $\Delta y < 0$, то удалить это ребро из списка активных ребер.

Вычислить новую абсциссу пересечения:

$$x_{\text{нов}} = x_{\text{стар}} + \Delta x$$

Сократить список активных многоугольников:

Для каждого многоугольника уменьшить $\Delta y_p$. Если $\Delta y_p < 0$

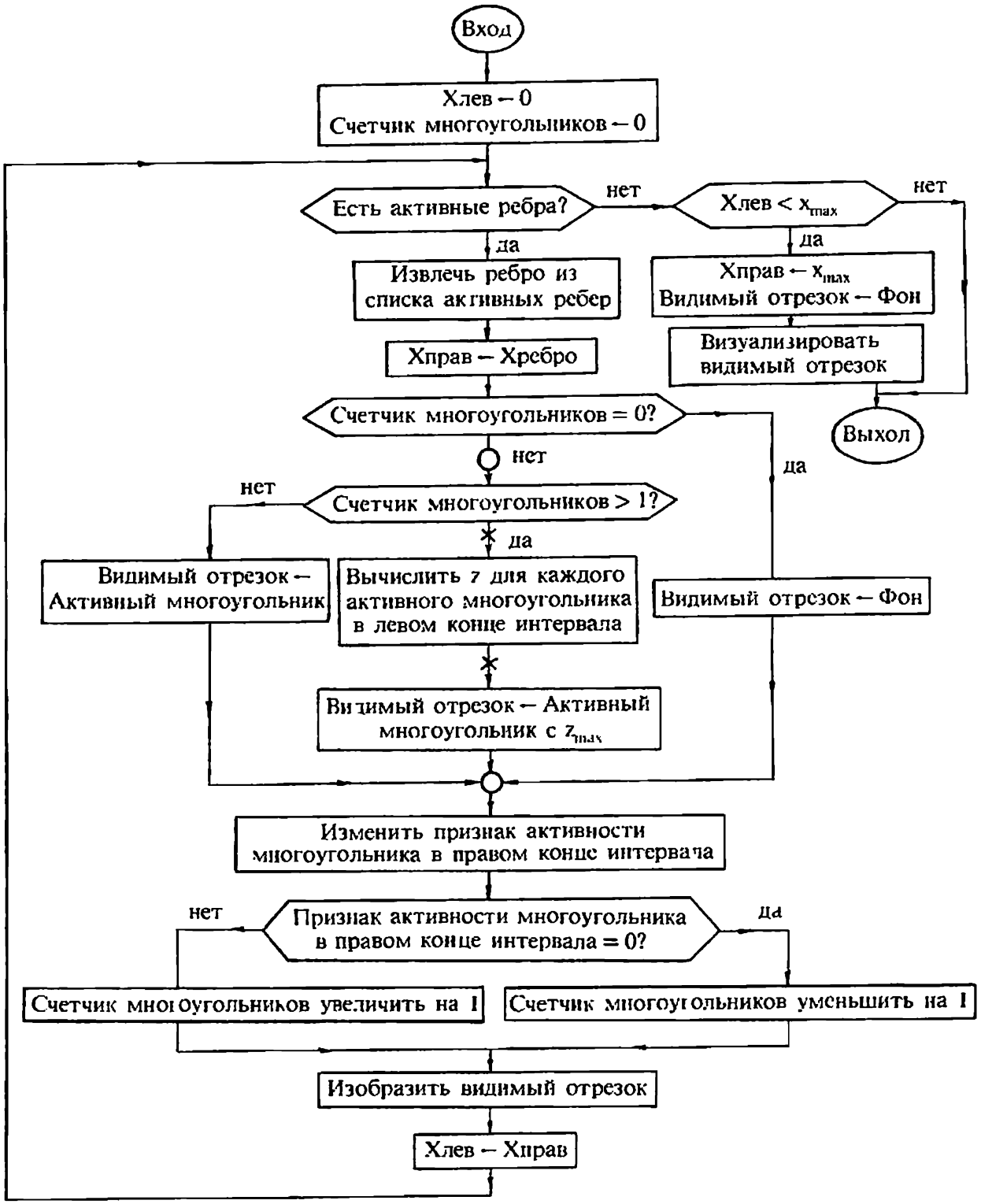



Рис. 4.61. Блок-схема интервального алгоритма для непротыкающих многоугольников.

для какого-то многоугольника, то удалить этот многоугольник из списка.

В данном алгоритме не используются преимущества, обеспечиваемые когерентностью приоритетов по глубине, которая была предложена Ромни. Если многоугольники не могут протыкать друг дру-

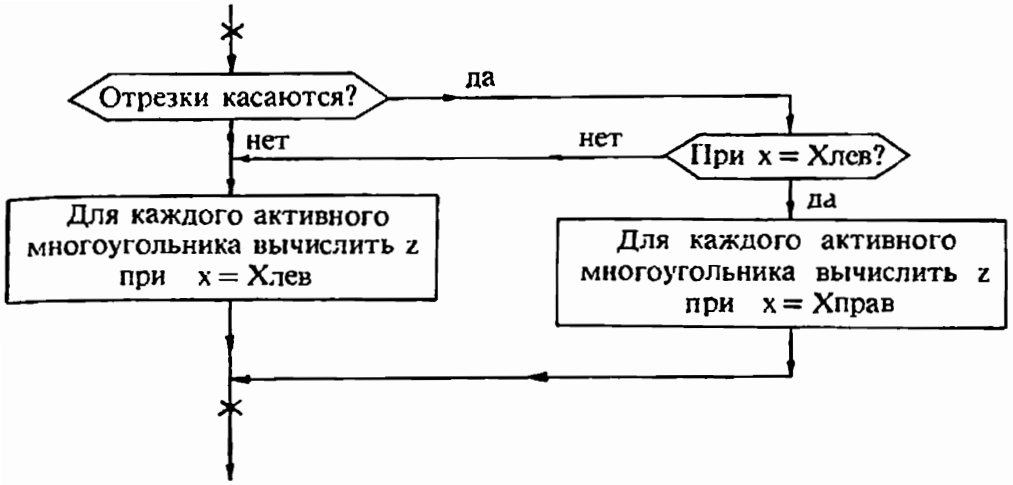



Рис. 4.62. Модификация блока вычисления глубины для блок-схемы с рис. 4.61.

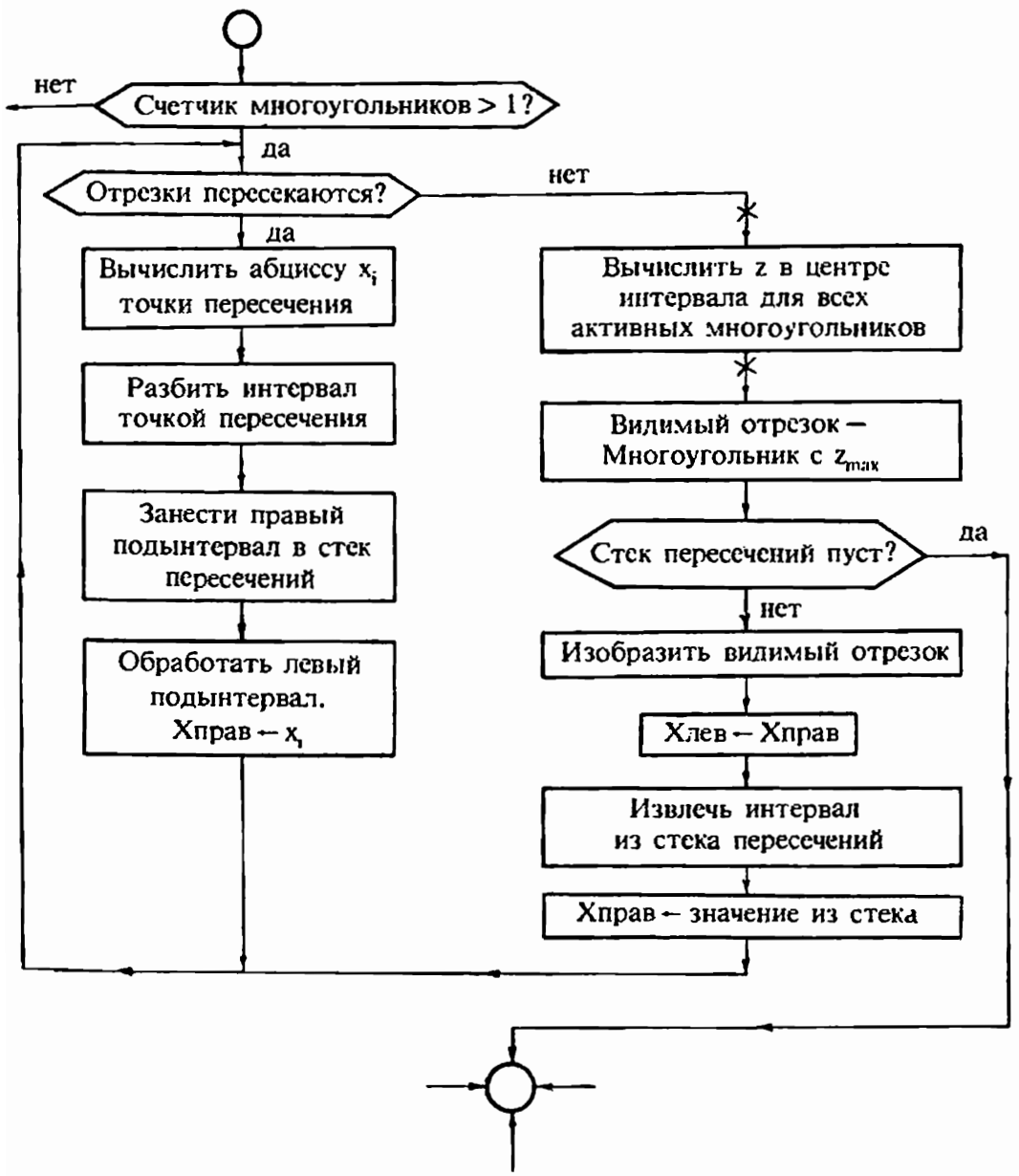



Рис. 4.63. Модификация блок-схемы с рис. 4.61 для протыкающих многоугольников.

га, то модификация этого алгоритма, построенная с учетом когерентности по глубине, приводит к значительному его улучшению.

В простом интервальном алгоритме, приведенном на рис. 4.61, предполагается, что отрезки многоугольников в пределах одного интервала не пересекаются. Если же отрезки пересекаются в концевых точках интервала, то, как указывалось ранее, вычисление глубины для этих отрезков производится в противоположных им концевых точках данного интервала. Простая модификация блока вычисления глубины в блок-схеме, показанной на рис. 4.61, приведена на рис. 4.62.

Если отрезки пересекаются внутри интервала, т. е. если многоугольники протыкают друг друга, то либо нужно использовать более сложный алгоритм разбиения на интервалы, либо точки пересечения нужно вставлять в упорядоченный список ребер. Интервальный алгоритм, показанный на рис. 4.61, можно применять и в случае допустимости пересечения многоугольников, если включить эти пересечения в список активных ребер, пометить каждое пересечение флагом, изменить порядок модификации флага активности многоугольника и выполнять вычисления глубин в центре интервала.

На рис. 4.63 показана модификация алгоритма, приведенного на рис. 4.61. В модифицированном алгоритме предполагается, что список активных ребер не содержит пересечения. Пересекающиеся отрезки нужно обнаруживать и обрабатывать сразу же по ходу алгоритма. Здесь в каждом интервале ищутся пересекающиеся отрезки. Если они обнаруживаются, что вычисляется точка пересечения и интервал разбивается в этой точке. Правый подынтервал заносится в стек. Алгоритм рекурсивно применяется к левому подынтервалу до тех пор, пока не обнаруживается такой подынтервал, в котором уже нет пересечений. Этот подынтервал изображается, и новый подынтервал извлекается из стека. Процесс продолжается до тех пор, пока стек не опустеет. Этот метод аналогичен тому, который предлагался Джексоном [4-29]. Заслуживает упоминания тот факт, что фотографии на вклейке (12 и 13) были получены с помощью алгоритма Ваткинса.

Для простоты в модифицированной версии алгоритма (рис. 4.63) предполагается, что пересечения отрезков лежат внутри каждого из них. Поскольку отрезки могут касаться в концевых точках интервала, то вычисление глубины проводится в его центре. Модифицированный блок вычисления глубины, показанный на рис. 4.62, можно подставить в схему на рис. 4.63, чтобы сократить объем вычислений. Дополнительная модификация этого алгоритма реализует сор-

тировку по глубине интервалов, содержащих точки пересечения отрезков, чтобы определить видимость пересекающихся отрезков прежде, чем производить разбиение интервала. Эта модификация позволяет сократить количество подынтервалов и повысить эффективность алгоритма. Если необходимо, то для повышения эффективности используется также предварительное удаление нелицевых граней.

**Пример 4.23.** Интервальный алгоритм построчного сканирования

Рассмотрим опять прямоугольник и протыкающий его треугольник, которые уже обсуждались в примерах 4.19 и 4.22. Возьмем сканирующую строку 15. Будем использовать соглашение о половине интервала между сканирующими строками. Пересечение сканирующей плоскости при $y = 15.5$ с многоугольниками изображено на рис. 4.64. На рис. 4.58 показана проекция этих многоугольников на плоскость $xy$. Непосредственно перед началом обработки сканирующей строки 15 список активных ребер, упорядоченный по возрастанию значения $x$, содержит следующие величины:

10, 0, 10, 1, 0, 15 1/2, −1, 0, 2, 0, 25, 0, 10, 1, 0, 28 1/6, 1/3, 5, 2, 0

где числа сгруппированы в пятерки, соответствующие значениям ($x$, $\Delta x$, $\Delta y$, $P$, *Флаг*), которые были определены в алгоритме выше. На рис. 4.64 показаны пять интервалов, порожденных четырьмя точками пересечения активных ребер со сканирующей строкой. Из этого же рисунка видно, что отрезки, соответствующие многоугольникам, пересекаются внутри третьего интервала.

Сканирующая строка обрабатывается слева направо в порядке, определяемом растровым сканированием. В первом интервале нет многоугольников. Он изображается (пикселы от 0 до 9) с фоновым значением атрибутов. Во втором интервале становится активным прямоугольник. Его флаг изменяется на 1: *Флаг* = $-$*Флаг* + 1. В этом интервале нет других многоугольников. Следовательно, он изображается (пикселы от 10 до 14) с атрибутами прямоугольника.

Третий интервал начинается при $x = 15.5$. Становится активным треугольник, и его флаг изменяется на 1, счетчик многоугольников возрастает до двух, значением левой абсциссы интервала ($X$ лев) становится 15.5, и следующее ребро при $x = 25$ выбирается из списка активных ребер. Значением правой абсциссы интервала ($X$ прав) становится 25. Поскольку значение счетчика многоугольников больше единицы, то соответствующие им отрезки проверяются на пересечение (рис. 4.63).

Уравнение плоскости треугольника (см. пример 4.19) таково:

$$3x + y + 4z - 120 = 0$$

Для сканирующей строки 15 значение $y = 15.5$; а глубина треугольника для пиксела с абсциссой $x$ равна

$$z = (120 - y - 3x)/4 = (120 - 15.5 - 3x)/4 = (104.5 - 3x)/4$$

Поэтому в центрах пикселов, т. е. при $x + 1/2$, на концах интервала:

$$z_{2_л} = [104.5 - (3)\,(15.5)]/4\text{-}14.5; \quad z_{2_п} = [104.5 - (3)\,(25.5)]/4\text{-}7.0$$

Поскольку глубина прямоугольника постоянна, то

$$z_{1_л} = 10; \quad z_{1_п} = 10$$

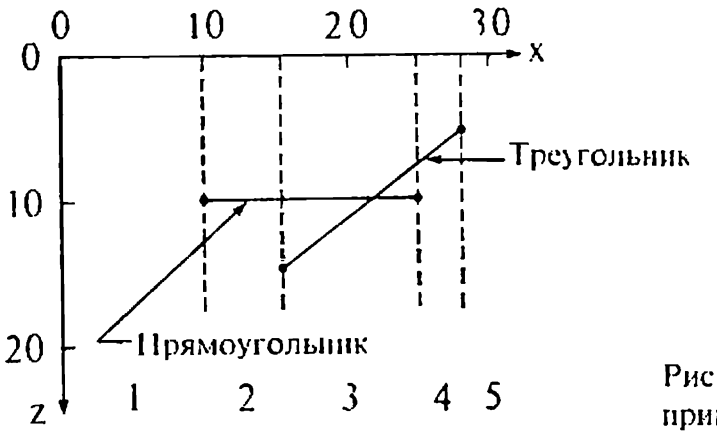



Рис. 4.64. Сканирующая плоскость для примера 4.23.

Из уравнения (4.9) имеем:

$$\mathbf{Sign}(z_{1_л} - z_{2_л}) = \mathbf{Sign}(10 - 14.5) < 0.$$
$$\mathbf{Sign}(z_{1_п} - z_{2_п}) = \mathbf{Sign}(10 - 7) > 0$$

Поскольку $\mathbf{Sign}(z_{1_л} - z_{2_л}) \neq \mathbf{Sign}(z_{1_п} - z_{2_п})$, эти отрезки пересекаются. Координаты точки пересечения таковы:

$$z = (120 - 15.5 - 3x)/4 = 10$$
$$x_i = 21.5$$

Интервал разбивается точкой с абсциссой $x = 21.5$. Значение $X$прав заносится в стек. Теперь значением $X$прав становится $x_i$, т. е. 21.5.

В подынтервале от $x = 15.5$ до $x = 21.5$ нет пересечений. Глубина треугольника в центре этого подынтервала, т. е. при $x = 18.5$, равна:

$$z_2 = (104.5 - 3x)/4 = [104.5 - (3)(18.5)]/4 = 12.25$$

что больше, чем $z_1 = 10$ для треугольника. Поэтому в этом подынтервале изображается треугольник (пикселы от 15 до 20).

Затем значением $X$лев становится значение $X$прав, и правый подынтервал извлекается из стека. Значением $X$прав становится величина, извлеченная из стека, т. е. $x = 25$. В подынтервале от $x = 21.5$ до $x = 25$ нет пересечений. Глубина треугольника в центре этого подынтервала, т. е. при $x = 23.25$ равна

$$z_2 = (104.5 - 3x)/4 = [104.5 - (3)(23.25)]/4 = 8.69$$

что меньше, чем $z_1 = 10$ для прямоугольника. Поэтому в этом подынтервале (пикселы от 21 до 24) видимым является прямоугольник.

Стек пересечений теперь пуст. Процедура, приведенная на рис. 4.63, передает управление процедуре, изображенной на рис. 4.61. В правом конце интервала находится прямоугольник. Он и перестает быть активным. Его флаг становится равным 0, что приводит к уменьшению счетчика многоугольников на 1. Сегмент изображается с атрибутами прямоугольника. $X$лев опять заменяется на $X$прав.

Следующим ребром, извлеченным из списка активных ребер, будет ребро треугольника со значением $x = 28\frac{1}{6}$. Четвертый интервал простирается от $x = 25$ до $x = 28\frac{1}{6}$.

Счетчик многоугольников здесь равен 1. Активен в этом интервале только треугольник. Поэтому отрезок изображается с атрибутами треугольника (пикселы от

25 до 27). В правом конце интервала находится треугольник. Его флаг становится равным 0, и он теперь неактивен. Счетчик многоугольников становится равным 0. $X$лев заменяется на $X$прав, т. е. на 26½.

Теперь в списке активных ребер ничего нет. Здесь $x_{max} = 32$, поэтому $X$лев $< x_{max}$. Значит, $X$прав заменяется на $x_{max}$ и последний интервал (пикселы от 28 до 31) изображается с фоновыми атрибутами. Затем $X$лев заменяется на $X$прав. Опять нет активных ребер, но $X$лев $= x_{max}$, и обработка сканирующей строки завершается.

Окончательный результат совпадает с тем, что показан на рис. 4.19.

Алгоритмы построчного сканирования можно реализовать и как алгоритмы удаления невидимых линий. Например, Арчулета [4-30] предложил такую версию алгоритма Ваткинса, которая занимается удалением невидимых линий.

## 4.12. АЛГОРИТМЫ ПОСТРОЧНОГО СКАНИРОВАНИЯ ДЛЯ КРИВОЛИНЕЙНЫХ ПОВЕРХНОСТЕЙ

Хотя алгоритм разбиения криволинейных поверхностей Кэтмула (см. разд. 4.6) прост и элегантен, но результат, который он выдает, не упорядочен по ходу сканирования строк развертки. Это неудобно для растровых устройств вывода. Криволинейную поверхность можно, разумеется, аппроксимировать многогранником и воспользоваться одним из изложенных выше алгоритмов построчного сканирования. Однако для получения высокой точности число граней многоугольника, необходимых для описания достаточно сложной сцены, огромно. Кроме того, если не использовать при этом интерполяционные методы закраски (см. гл. 5), то результат будет выглядеть состоящим из граней. В любом случае контурные ребра окажутся кусочно-линейными, т. е. будут выглядеть как ломаные, состоящие из коротких прямолинейных отрезков.

Алгоритм визуализации параметрических биполиномиальных (обычно бикубических) поверхностей непосредственно по описанию этих поверхностей в порядке, определяемом построчным сканированием, были разработаны Блинном [4-31], Уиттедом [4-32], Лейном и Карпентером [4-33, 4-34] и Кларком [4-35]. Первыми обсуждаются похожие друг на друга алгоритмы Блинна и Уиттеда. А затем будут рассмотрены алгоритмы Лейна — Карпентера и Кларка, которые также похожи друг на друга.

То, что в алгоритме построчного сканирования сцена рассекается сканирующей плоскостью, проходящей через точку наблюдения и сканирующую строку, сразу демонстрирует различие между поли-

гональной и криволинейной (скульптурной) параметрическими поверхностями. Для полигональной поверхности все пересечения со сканирующей плоскостью являются прямолинейными отрезками. Эти прямолинейные отрезки легко описать их концевыми точками. Для криволинейной параметрической поверхности ее пересечение со сканирующей плоскостью задается уравнением

$$y(u, w) = y\text{скан} = \text{const}$$

где $u$ и $w$ — параметры, задающие поверхность. В результате получается кривая, называемая линией уровня или контуром. Эта кривая не обязательно описывается однозначной функцией. Более того, контур любого уровня может содержать несколько кривых. Наконец, мало найти кривую (или кривые), образованные пересечением поверхности со сканирующей плоскостью, нужно, кроме того, найти проекции каждой ее точки на сканирующую строку, т. е. $x = x(u, w)$, и уметь вычислять глубину кривой при этом значении абсциссы, т. е. $z = z(u, w)$, чтобы определять ее видимость.

Математически последнее требование можно сформулировать так. Дана ордината $y$ сканирующей строки и абсцисса $x$ точки, лежащей на этой строке. Необходимо вычислить значения параметров $u$, $w$, т. е. найти

$$u = u(x, y) \text{ и } w = w(x, y)$$

Если значения этих параметров известны, то глубина вычисляется по формуле $z = z(u, w)$. Следовательно, можно вычислить атрибуты видимости исходной точки, лежащей на сканирующей строке. К сожалению, точное решение этих уравнений не известно. И Блинн, и Уиттед использовали численные методы для получения решения. В частности, использовался итерационный метод Ньютона — Рафсона [4-36]. Метод Ньютона — Рафсона нуждается в начальном приближении. В обоих алгоритмах используется свойство когерентности сканирующих строк для получения такого начального приближения и сокращения числа итераций на один пиксел. К сожалению, итерационный процесс в методе Ньютона — Рафсона может расходиться. Кадзия [4-37] предложил более устойчивый, хотя и более сложный метод, основанный на идеях алгебраической геометрии.

Кратко, в контексте структуры алгоритма построчного сканирования, внутренний цикл алгоритмов Блинна и Уиттеда таков:

Параметрическая поверхность задана списком активных элементов,

описанных параметрическими уравнениями:

$$x = x(u, w),\ y = y(u, w),\ z = z(u, w)$$

Для каждой сканирующей строки (ее ордината равна $y$):
Для каждого пиксела на этой строке (его абсцисса равна $x$):
Для каждой поверхности, пересекающей эту сканирующую плоскость при данном $x$:

Решить уравнения: $u = u(x, y)$, $w = w(x, y)$.
Вычислить глубину поверхности: $z = z(u, w)$.

Определить поверхность, видимую при заданных $x$ и $y$, и изобразить пиксел с ее атрибутами.

Данный алгоритм иллюстрирует еще одно важное отличие многогранника от криволинейной параметрической поверхности. В алгоритме говорится: «Для каждой поверхности, пересекающей эту сканирующую плоскость». Поверхности становятся активными на самой верхней и неактивными на самой низшей из пересекающих их сканирующих плоскостей. Эти пересечения возникают на локальных максимумах и минимумах поверхностей. Для полигональных поверхностей эти локальные экстремумы всегда совпадают с их вершинами. В алгоритмах построчного сканирования эти вершины и соединяющие их ребра многогранника используются для решения вопросов о том, когда грань следует добавить или удалить из списков активных граней и ребер.

Для криволинейных поверхностей локальные экстремумы не обязательно совпадают с их вершинами. Часто они располагаются внутри поверхности вдоль контурных ребер. Контурное ребро, лежащее внутри поверхности, определяется обнулением компоненты $z$ у вектора нормали к поверхности. Несколько примеров приведено на рис. 4.65. В случае криволинейной поверхности ее грань можно добавить в список активных граней или удалить из него при обработке контурных ребер, а интервалы вдоль сканирующей строки могут начинаться и заканчиваться на таких ребрах. Эта задача решается в алгоритмах Блинна и Уиттеда путем эффективного разбиения поверхности вдоль контурных ребер.

Алгоритмы визуализации параметрических криволинейных поверхностей, принадлежащие Лейну — Карпентеру и Кларку, основываются прежде всего на методах разбиения поверхностей. Однако в отличие от первого из алгоритмов подразбиения, предложенного Кэтмулом, в котором поверхность обрабатывается в произвольном порядке, в этих алгоритмах результат упорядочен по ходу

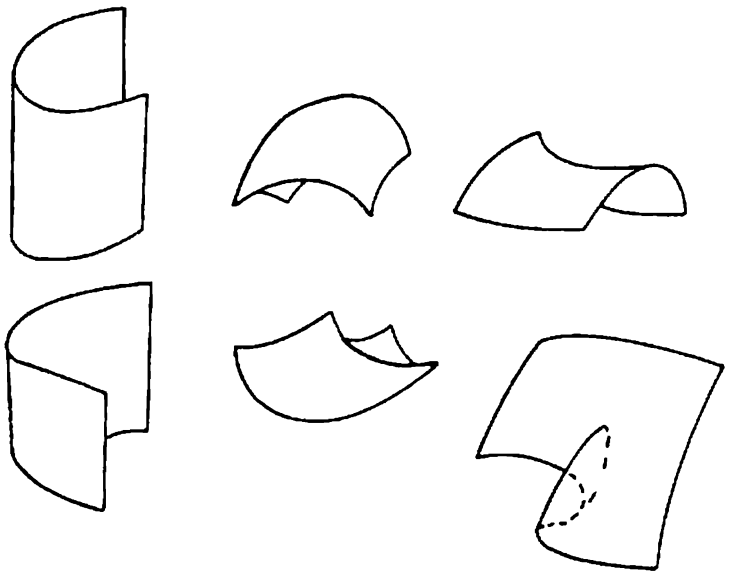


Рис. 4.65. Примеры контурных ребер.

сканирования строк развертки. Алгоритмы выполняют групповую сортировку по $y$ элементов поверхности, основываясь на максимальных значениях ординаты каждого из этих элементов. Для каждой сканирующей строки те элементы из списка активных элементов, которые пересекаются соответствующей сканирующей плоскостью, подразделяются до тех пор, пока каждый новый элемент не станет удовлетворять критерию пологости или не перестанет пересекаться со сканирующей плоскостью. Те части элементов, которые больше не пересекаются со сканирующей плоскостью, заносятся в список неактивных элементов для последующего рассмотрения. Те же части элементов, которые удовлетворяют критерию пологости, далее начинают считаться плоскими многоугольниками и преобразовываться в растровую форму с помощью алгоритма построчного сканирования для многоугольников. Однако каждый из таких приближенно плоских многоугольников остается параметрическим элементом. Вся информация, имеющаяся для подобного элемента поверхности, используется для определения визуальных атрибутов отдельных пикселов в процессе преобразования многоугольника в растровую форму. Использование этой информации позволяет произвести гладкое сопряжение этих элементов. Если плоским считать элемент, размеры которого меньше одного пиксела, то полученный контур будет гладким. Далее, задние или нелицевые многоугольники можно удалить простым вычислением нормали к поверхности (см. разд. 4.3). Если нормаль направлена не в сторону наблюдателя, то соответствующая часть элемента удаляется. Это позволяет сэкономить большой объем вычислений.

Хотя оба алгоритма Лейна — Карпентера и Кларка используют изложенную идею, в алгоритме Кларка элементы проходят до рас-

Рис. 4.66. Чайник, заданный 28 бикубическими кусками и изображенный с помощью алгоритма Лейна — Карпентера. (С разрешения Л. Карпентера.)

тровой развертки предварительную обработку, в то время как в алгоритме Лейна — Карпентера элементы динамически подразделяются в процессе обработки кадра. Алгоритм Лейна — Карпентера требует значительно меньше памяти, чем алгоритм Кларка, но осуществляет больше разбиений. Рис. 4.66 построен с использованием алгоритма Лейна — Карпентера.

В контексте алгоритма построчного сканирования внутренний цикл алгоритма Лейна — Карпентера кратко записывается так:

Для каждой сканирующей строки с ординатой $y$:

Для каждого элемента из списка активных элементов:

```
if элемент плоский then
   занести этот элемент в список многоугольников
else
   разбить этот элемент на подэлементы
   if подэлемент еще пересекается со сканирующей плоскостью
   then
      занести его в список активных элементов
   else
      занести его в список неактивных элементов
   end if
end if
```

Произвести растровую развертку многоугольников из списка многоугольников.

Оценка обеих алгоритмов Лейна — Карпентера и Кларка зависит от свойств конкретных базисных функций, используемых для генерации параметрических элементов с целью их эффективного разбие-

ния. Эти алгоритмы применимы к любым параметрическим элементам поверхности, для которых имеется эффективный алгоритм разбиения. Единственный недостаток этих алгоритмов адаптивного разбиения заключается в том, что из-за несоответствия между приближенными полигональными элементами и точными параметрическими элементами могут образовываться разрывы или дыры в поверхности.

Квадратные поверхности, вообще говоря, в чем-то проще, чем параметрические элементы поверхности. Квадратичные поверхности задаются общим уравнением второго порядка:

$$a_1x^2 + a_2y^2 + a_3z^2 + a_4xy + a_5yz + a_6zx + a_7x + a_8y + a_9z + a_{10} = 0$$

Типичными примерами квадратичных поверхностей являются сферы, конусы, цилиндры, а также эллипсоиды и гиперболоиды вращения. Если все коэффициенты от $a_1$ до $a_6$ равны нулю, то квадратичное уравнение сводится к уравнению плоскости многоугольника $a$.

Сферы, как подмножество квадратичных поверхностей, представляют особый интерес в задачах молекулярного моделирования. Разработано несколько алгоритмов построчного сканирования специально для сфер. В частности, алгоритмы Портера [4-38, 4-39] и Стодхаммера [4-40] являются примерами алгоритмов построчного сканирования с использованием $z$-буфера, которые ориентированы на сферы. Ограничиваясь случаем ортогональных проекций, Портер эффективно использовал алгоритм Брезенхема для окружностей (см. разд. 2.6), чтобы формировать контур сферы. Более того, поскольку пересечение сканирующей плоскости со сферой тоже является окружностью, то можно воспользоваться алгоритмом Брезенхема для окружностей, чтобы инкрементально вычислять глубину каждой сферы на сканирующей строке. Наконец, алгоритм Брезенхема используется для устранения ступенчатости контурных ребер (см. разд. 2.26) путем поддержания списка приоритетов сфер, основанных на глубинах их центров. Сортировка по приоритету позволяет также учесть эффекты прозрачности.

## 4.13. АЛГОРИТМ ОПРЕДЕЛЕНИЯ ВИДИМЫХ ПОВЕРХНОСТЕЙ ПУТЕМ ТРАССИРОВКИ ЛУЧЕЙ

Оценки эффективности всех алгоритмов удаления невидимых поверхностей, обсуждавшихся в предыдущих разделах, зависят от определенных характеристик когерентности той сцены, для кото-

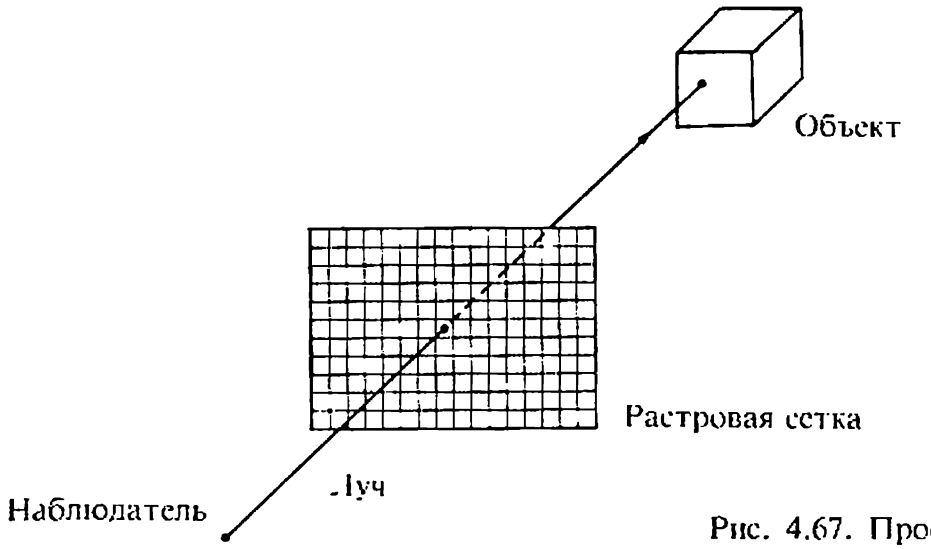



Рис. 4.67. Простая трассировка луча.

рой ведется поиск ее видимых участков. В отличие от них трассировка лучей является методом грубой силы[1]. Главная идея, лежащая в основе этого метода, заключается в том, что наблюдатель видит любой объект посредством испускаемого неким источником света, который падает на этот объект и затем каким-то путем доходит до наблюдателя. Свет может достичь наблюдателя, отразившись от поверхности, преломившись или пройдя через нее. Если проследить за лучами света, выпущенными источником, то можно убедиться, что весьма немногие из них дойдут до наблюдателя. Следовательно, этот процесс был бы вычислительно неэффективен. Аппель [4-41] первым предложил отслеживать (трассировать) лучи в обратном направлении, т. е. от наблюдателя к объекту, как показано на рис. 4.67. Этот метод был с успехом реализован в рамках дисплейной системы визуализации твердых тел MAGI [4-42]. В самой системе MAGI трассировка прекращалась, как только луч пересекал поверхность видимого непрозрачного объекта; т. е. луч использовался только для обработки скрытых или видимых поверхностей. Впоследствии Кэй [4-43, 4-44] и Уиттед [4-45] реализовали алгоритмы трассировки лучей с использованием общих моделей освещения. Эти алгоритмы учитывают эффекты отражения одного объекта от поверхности другого, преломления, прозрачности и затенения. Производится также устранение ступенчатости. Алгоритм, учитывающий эти эффекты, обсуждается в разд. 5.12. Настоящее же обсуждение ограничено применением метода трассировки лучей для определения видимых или скрытых поверхностей.

---

[1] Методом грубой силы принято называть метод, не учитывающий специфику обрабатываемого объекта.— *Прим. перев.*

Рис. 4.67 служит иллюстрацией алгоритма трассировки лучей. В этом алгоритме предполагается, что сцена уже преобразована в пространство изображения. Перспективное преобразование не используется. Считается, что точка зрения или наблюдатель находится в бесконечности на положительной полуоси $z$. Поэтому все световые лучи параллельны оси $z$. Каждый луч, исходящий от наблюдателя, проходит через центр пиксела на растре до сцены. Траектория каждого луча отслеживается, чтобы определить, какие именно объекты сцены, если таковые существуют, пересекаются с данным лучом. Необходимо проверить пересечение каждого объекта сцены с каждым лучом. Если луч пересекает объект, то определяются все возможные точки пересечения луча и объекта. Можно получить большое количество пересечений, если рассматривать много объектов. Эти пересечения упорядочиваются по глубине. Пересечение с максимальным значением $z$ представляет видимую поверхность для данного пиксела. Атрибуты этого объекта используются для определения характеристик пиксела.

Если точка зрения находится не в бесконечности, алгоритм трассировки лучей лишь незначительно усложняется. Здесь предполагается, что наблюдатель по-прежнему находится на положительной полуоси $z$. Картинная плоскость, т. е. растр, перпендикулярна оси $z$, как показано на рис 4.68. Задача состоит в том, чтобы построить одноточечную центральную проекцию на картинную плоскость [1-1].

Наиболее важным элементом алгоритма определения видимых поверхностей путем трассировки лучей, является процедура определения пересечений. В состав сцены можно включать любой объект, для которого можно создать процедуру построения пересечений. Объекты сцены могут состоять из набора плоских многоугольников, многогранников или тел, ограниченных или определяемых квадратичными или биполиномиальными параметрическими поверхностями. Поскольку 75—95% времени, затрачиваемого алгоритмом трассировки лучей, уходит на определение пересечений, то эффективность процедуры поиска пересечений оказывает значительное влияние на производительность всего алгоритма. Вычислительная стоимость определения пересечений произвольной пространственной прямой (луча) с одним выделенным объектом может оказаться высокой (см., например, [4-37]). Чтобы избавиться от ненужного поиска пересечений, производится проверка пересечения луча с объемной оболочкой рассматриваемого объекта. И если луч не пересекает оболочки, то не нужно больше искать пересечений этого объ-

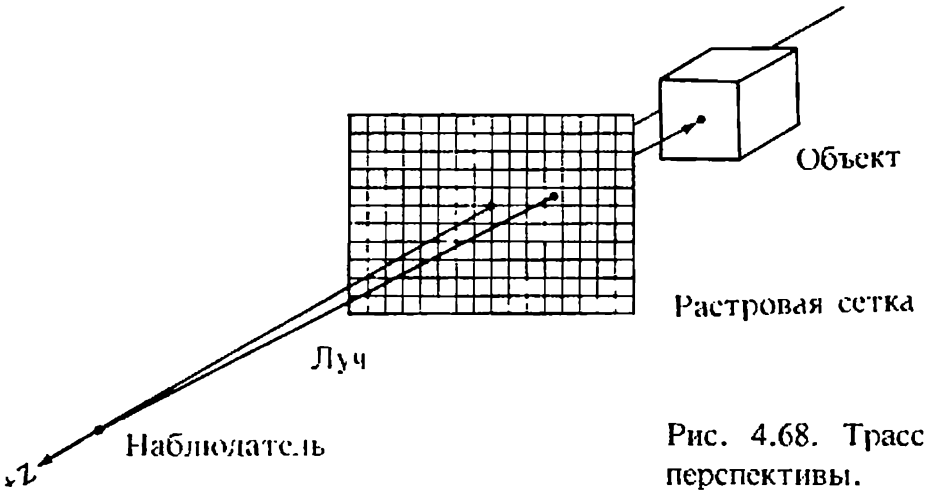



Рис. 4.68. Трассировка луча с учетом перспективы.

екта с лучом. В качестве оболочки можно использовать прямоугольный параллелепипед или сферу. Хотя, как показано, на рис. 4.69, использование сферы в качестве оболочки может оказаться неэффективным, факт пересечения трехмерного луча со сферой определяется очень просто. В частности, если расстояние от центра сферической оболочки до луча превосходит радиус этой сферы, то луч не пересекает оболочки. Следовательно, он не может пересечься и с объектом.

Поэтому тест со сферической оболочкой сводится к определению расстояния от точки до трехмерной прямой, т. е. луча. Будем использовать параметрическое представление прямой, проходящей через точки $P_1(x_1, y_1, z_1)$ и $P_2(x_2, y_2, z_2)$, т. е.:

$$P(t) = P_1 + (P_2 - P_1)t$$

с компонентами

$$\begin{aligned} x &= x_1 + (x_2 - x_1)t = x_1 + at \\ y &= y_1 + (y_2 - y_1)t = y_1 + bt \\ z &= z_1 + (z_2 - z_1)t = z_1 + ct \end{aligned}$$

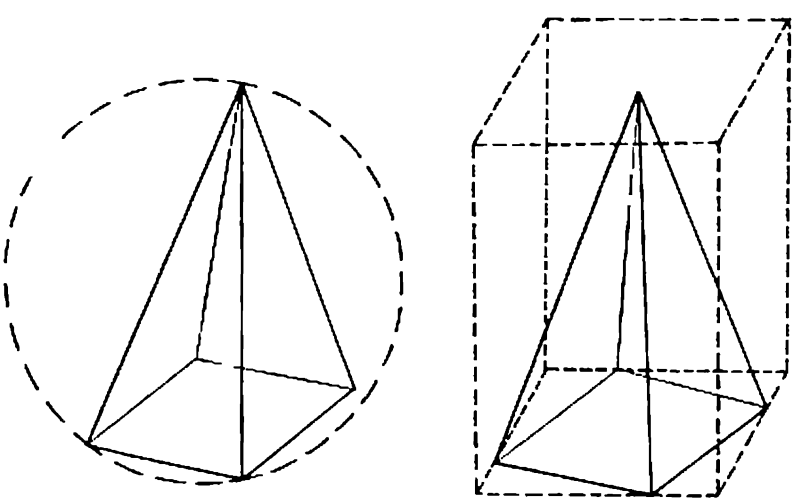


Рис. 4.69. Сферическая и прямоугольная оболочки.

Тогда минимальное расстояние $d$ от этой прямой до точки $P_0(x_0, y_0, z_0)$ равно:

$$d^2 = (x - x_0)^2 + (y - y_0)^2 + (z - z_0)^2$$

где параметр $t$, определяющий ближайшую точку $P(t)$ равен:

$$t = -\frac{a(x_1 - x_0) + b(y_1 - y_0) + c(z_1 - z_0)}{a^2 + b^2 + c^2}$$

Если $d^2 > R^2$, где $R$ — радиус сферической оболочки, то луч не может пересечься с объектом.

Выполнение габаритного теста с прямоугольной оболочкой в трехмерном пространстве требует большого объема вычислений. При этом следует проверить пересечение луча по меньшей мере с тремя бесконечными плоскостями, ограничивающими прямоугольную оболочку. Поскольку точки пересечения могут оказаться вне граней этого параллелепипеда, то для каждой из них следует, кроме того, произвести проверку на охват или попадание внутрь. Следовательно, для трех измерений тест с прямоугольной оболочкой оказывается более медленным, чем тест со сферической оболочкой.

Одной простой процедурой можно свести тест с прямоугольной оболочкой к сравнению знаков, упрощая тем самым вычисление пересечений с объектом, а также сравнения по глубине среди точек пересечения. В этой процедуре используются переносы и повороты вокруг координатных осей [1-1] для того, чтобы добиться совпадения луча с осью $z$. Аналогичным преобразованиям подвергается и прямоугольная оболочка объекта. Луч пересекает оболочку, если в новой перенесенной и повернутой системе координат знаки $x_{min}$ и $x_{max}$, *а также* $y_{min}$ и $y_{max}$ противоположны, как показано на рис. 4.70.

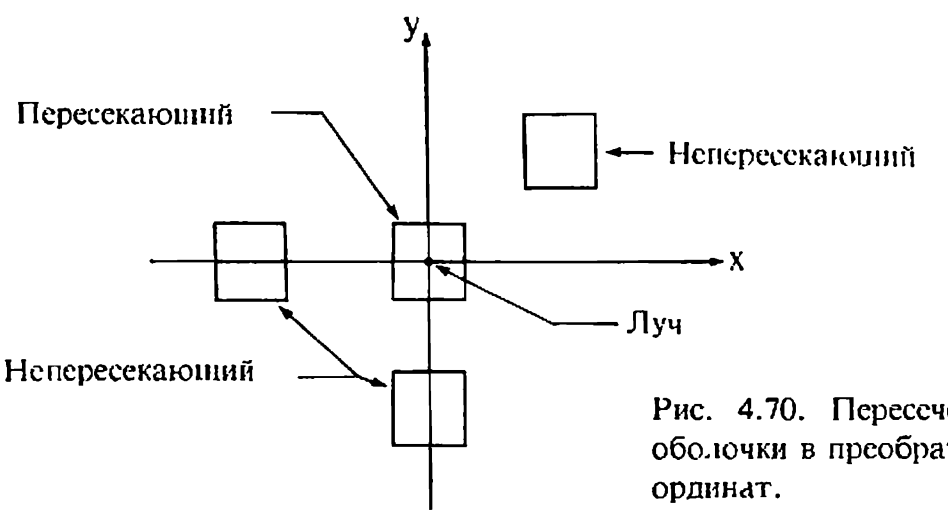



Рис. 4.70. Пересечения прямоугольной оболочки в преобразованной системе координат.

Продемонстрируем упрощение вычислений точек пересечения на примере поверхности второго порядка общего вида. В произвольной декартовой системе координат поверхностей второго порядка является геометрическим местом точек, координаты которых удовлетворяют уравнению:

$$Q(x, y, z) = a_1x^2 + a_2y^2 + a_3z^2 + b_1yz + b_2xz +$$

$$b_3xy + c_1x + c_2y + c_3z + d = 0$$

После применения преобразования, которое является комбинацией переноса и поворота и используется для совмещения луча с осью $z$, пересечение этого луча с поверхностью, если оно имеет место, возникает при $x = y = 0$. Поэтому в общем случае точки пересечения являются решениями уравнения:

$$a_3'z^2 + c_3'z + d' = 0$$

т. е.

$$z = \frac{-c_3' \pm \sqrt{c_3'^2 - 4a_3'd'}}{2a_3'}$$

где штрих сверху обозначает коэффициенты общего уравнения поверхности второго порядка после преобразования. Если $c_3'^2 - 4a_3'd' < 0$, то решения выражаются комплексными числами и луч не пересекает поверхности. Если бесконечная поверхность второго порядка (например, конус или цилиндр) ограничена плоскостями, то эти плоскости также следует преобразовать и проверить на пересечения. Если найдено пересечение с бесконечной ограничивающей плоскостью, то необходимо, кроме того, произвести проверку на попадание внутрь. Однако в преобразованной системе координат эту проверку можно произвести на двумерной проекции фигуры, образованной пересечением ограничивающей плоскости и квадратичной поверхности. Для получения точки пересечения в исходной системе координат необходимо применить обратное преобразование.

Вычисления пересечений для элементов биполиномиальных параметрических поверхностей более сложны. Уиттед [4-45] предложил простой метод разбиения для элемента бикубической поверхности. Вычисления выполняются с элементом поверхности в его исходном положении. Если луч пересекает сферическую оболочку элемента поверхности, то этот кусок разбивается с помощью алгоритма разбиения Кэтмула (см. разд. 4.6). Затем луч проверяется на пересечение со сферическими оболочками подэлементов. Если пересе-

чений не обнаружено, то луч не пересекается и с самим элементом. Если же луч пересекается со сферической оболочкой какого-нибудь подэлемента, то последний разбивается дальше. Процесс завершается, если ни одна из сферических оболочек не пересечена или если достигнут заранее определенный их минимальный размер. Эти сферические оболочки минимального размера и являются искомыми пересечениями луча и элемента поверхности.

При реализации преобразования, совмещающего луч с осью $z$, метод разбиения можно использовать скорее применительно к прямоугольным оболочкам, чем к сферическим. Это сокращает число разбиений и увеличивает эффективность алгоритма. Для параметрических поверхностей, обладающих свойством выпуклой оболочки, например для поверхностей Безье и $B$-сплайнов [1-1], число разбиений можно сократить дополнительно за счет усложнения алгоритма, если для подэлементов воспользоваться их выпуклыми оболочками вместо прямоугольных.

Кадзия [4-37] разработал метод для биполиномиальных параметрических поверхностей, который не требует их подразделения. Этот метод основан на понятиях, заимствованных из алгебраической геометрии. Решения получающихся при этом алгебраических уравнений высших степеней находятся численно. Метод, подобный этому, можно реализовать в преобразованной системе координат. Напомним, что биполиномиальная параметрическая поверхность определяется уравнением

$$Q(u, w) = 0$$

с компонентами

$$x = f(u, w)$$
$$y = g(u, w)$$
$$z = h(u, w)$$

В преобразованной системе координат выполнено условие $x = y = 0$. Значит,

$$f(u, w) = 0$$
$$g(u, w) = 0$$

Совместное решение этой пары уравнений дает значения $u$ и $w$ для точек пересечения. Подстановка этих значений в уравнение $z = h(u, w)$ дает компоненту $z$ для точек пересечения. Неудача попытки найти действительное решение означает, что луч не пересекает поверхность. Степень системы уравнений для $u$, $w$ равна произведению степеней биполиномиальных поверхностей. Бикубиче-

ская поверхность, например, имеет шестую степень. Следовательно, в общем случае потребуются численные методы решения. Там, где это допустимо, для начального приближения $u$ и $w$ можно использовать пересечения луча с выпуклой оболочкой. Для получения пересечений в исходной системе координат, как и ранее, следует использовать обратное преобразование.

Если трассируемый луч пересекает объекты сцены в нескольких точках, то необходимо определить видимое пересечение. Для алгоритмов определения видимости простых непрозрачных поверхностей, которые обсуждаются в данном разделе, пересечением с видимой поверхностью будет точка с максимальным значением координаты $z$. Для более сложных алгоритмов, учитывающих отражения и преломления, эти пересечения следует упорядочить вдоль луча по расстоянию от его начала. В преобразованной системе координат этой цели можно достичь простой сортировкой по $z$.

Алгоритм трассировки лучей для простых непрозрачных поверхностей можно представить следующим образом:

Подготовка данных для сцены:

Создать список объектов, содержащий по меньшей мере следующую информацию:

Полное описание объекта: тип, поверхность, характеристики и т. п.
Описание сферической оболочки: центр и радиус.
Флаг прямоугольной оболочки. Если этот флаг поднят, то будет выполнен габаритный тест с прямоугольной оболочкой, если же он опущен, то тест выполняться не будет. Заметим, что габаритный тест необходим не для всех объектов, например для сферы он не нужен.
Описание прямоугольной оболочки: $x_{min}$, $x_{max}$, $y_{min}$, $y_{max}$, $z_{min}$, $z_{max}$.

Для каждого трассируемого луча:

Выполнить для каждого объекта трехмерный тест со сферической оболочкой в исходной системе координат. Если луч пересекает эту сферу, то занести объект в список активных объектов.

Если список активных объектов пуст, то изобразить данный пиксел с фоновым значением интенсивности и продолжать работу. В противном случае, перенести и повернуть луч так, чтобы он совместился с осью $z$. Запомнить это комбинированное преобразование.

Для каждого объекта из списка активных объектов:

Если флаг прямоугольной оболочки поднят, преобразовать, используя комбинированное преобразование, эту оболочку в систему координат, в которой находится луч, и выполнить соответствующий тест. Если пересечения с лучом нет, то перейти к следующему объекту. В противном случае преобразовать, используя комбинированное преобразование, объект в систему координат, в которой находится луч, и определить его пересечения с лучом, если они существуют. Занести все пересечения в список пересечений.

Если список пересечений пуст, то изобразить данный пиксел с фоновым значением интенсивности.

В противном случае определить $z_{max}$ для списка пересечений.

Вычислить преобразование, обратное комбинированному преобразованию.

Используя это обратное преобразование, определить точку пересечения в исходной системе координат.

Изобразить данный пиксел, используя атрибуты пересеченного объекта и соответствующую модель освещенности.

Заметим, что алгоритм определения видимости простых непрозрачных поверхностей, не требует вычислять преобразование, обратное комбинированному, или определять точку пересечения в исходной системе координат, если в модели освещения не возникает необходимость включения в алгоритм свойств поверхности объекта или ее ориентации в точке пересечения (см. гл. 5). Эти шаги включены в данный алгоритм для полноты и удобства при реализации алгоритма трассировки лучей с учетом общей модели освещенности (см. разд. 5.12). Более полно приведенные рассуждения проиллюстрируем на примере.

**Пример 4.24.** Алгоритм трассировки лучей

Снова возьмем прямоугольник и протыкающий его треугольник, которые уже рассматривались в примерах 4.19, 4.22 и 4.23. Для простоты предположим, что наблюдатель расположен в бесконечности на положительной полуоси $x$. Следовательно, все лучи параллельны оси $z$. Ось $x$ проходит через точку (0, 0) на растре. Напомним, что углы прямоугольника расположены в точках $P_1(10, 5, 10)$, $P_2(10, 25, 10)$, $P_3(25, 25, 10)$, $P_4(25, 5, 10)$. Центр его сферической оболочки расположен в точке (17.5, 15, 10), а ее радиус равен 12.5. Значения $x_{min}$, $x_{max}$, $y_{min}$, $y_{max}$, $z_{min}$, $z_{max}$ для прямоугольной оболочки этого прямоугольника равны соответственно 10, 25, 5, 25, 10, 10.

Вершины треугольника расположены в точках $P_5(15, 15, 15)$, $P_6(25, 25, 5)$, $P_7(30, 10, 5)$. Центр его сферической оболочки расположен в точке (22.885, 15.962, 8.846), а ее радиус равен 10.048. Прямоугольная оболочка этого треугольника задается числами 15, 30, 10, 25, 5, 15.

Таким образом, в списке объектов содержится два элемента, и оба флага прямоугольных оболочек подняты.

Рассмотрим луч, проходящий через центр пиксела, соответствующего точке (20, 15). Поскольку наблюдатель находится в бесконечности, этот луч параллелен оси $z$.

Сначала возьмем прямоугольник. Поскольку луч параллелен оси $z$, расстояние от центра сферической оболочки до луча не зависит от координаты $z$. Конкретно, учитывая, что центр пиксела находится в точке (20.5, 15.5), получаем:

$$d^2 = (20.5 - 17.5)^2 + (15.5 - 15)^2 = 9.25$$

Так как $(d^2 = 9.25) < (R^2 = 156.25)$, то наш луч пересекает сферическую оболочку прямоугольника. Прямоугольник заносится в список активных объектов.

Аналогично для треугольника имеем:

$$\begin{aligned} d^2 &= (20.5 - 22.885)^2 + (15.5 - 15.962)^2 \\ &= 5.90 \end{aligned}$$

Снова это число меньше, чем квадрат радиуса сферической оболочки, т. е. $(d^2 = 5.90) < (R^2 = 100.96)$. Поэтому луч пересекает и сферическую оболочку треугольника. Этот треугольник также заносится в список активных объектов.

Поскольку список активных объектов не пуст, луч подвергается преобразованию, совмещающему его с осью $z$. В нашем случае луч переносится на $(-20.5, -15.5, 0)$ в направлениях $x$, $y$, $z$, соответственно.

Аналогично перенос прямоугольной оболочки для прямоугольника дает значения $-10.5$, 4.5, $-10.5$, 9.5, 10, 10. Поскольку знаки $x_{min}$ и $x_{max}$, *а также* $y_{min}$ и $y_{max}$ противоположны, то луч пересекает эту прямоугольную оболочку. Точка пересечения луча с прямоугольником получается с использованием уравнения плоскости, несущей этот прямоугольник. В обеих системах координат до и после преобразования уравнение этой плоскости имеет вид:

$$z - 10 = 0$$

Поэтому пересечение ее с лучом происходит при $z = 10$. Точка пересечения лежит внутри прямоугольника. Эта точка и заносится в список точек пересечения.

Перенос прямоугольной оболочки для треугольника дает значения $-5.5$, 9.5, $-5.5$, 9.5, 5, 15. Опять знаки $x_{min}$ и $x_{max}$, *а также* $y_{min}$ и $y_{max}$ противоположны; поэтому луч пересекает эту оболочку и для треугольника. В исходной системе координат уравнение плоскости треугольника таково:

$$3x + y + 4z - 120 = 0$$

В преобразованной системе координат оно имеет вид (см. разд. 4.2):

$$3x + y + 4z - 43 = 0$$

и пересечение происходит при

$$z = (43 - 3x - y)/4 = 43/4 = 10.75$$

Эта точка лежит внутри треугольника и заносится в список точек пересечения.

Список точек пересечения не пуст. Максимальным является значение $z_{max} = 10.75$, и треугольник является видимым. Обратное преобразование к исходной системе координат дает точку пересечения (20.5, 15.5, 10.75). Пиксел, соответствующий точке (20, 15), изображается с атрибутами треугольника.

Две модификации этого простого алгоритма заметно повышают его эффективность. Первая модификация основывается на понятии кластерных групп пространственно связанных объектов. Например, предположим, что сцена состоит из стола, на котором стоят ваза с фруктами и блюдо с конфетами. В вазе лежат апельсин, яблоко, банан и груша. Блюдо содержит несколько конфет разных форм и цветов. Вводятся сферические оболочки для групп или кластеров связанных объектов, например для вазы и всех плодов в ней, для блюда и всех конфет в нем, а также для стола и всех предметов на нем. Сферические оболочки, охватывающие более чем один объект, называются сферическими кластерами. Если это необходимо, то можно ввести и прямоугольные кластеры. Вводится, кроме того, наибольший сферический кластер, именуемый сферой сцены, которая охватывает все объекты в этой сцене. Затем сферические оболочки обрабатываются в иерархическом порядке. Если луч не пересекает сферу сцены, то он не может пересечь и ни одного из ее объектов. Следовательно, пиксел, соответствующий этому лучу, будет изображен с фоновым значением интенсивности. Если же луч пересекает сферу сцены, то на пересечение с лучом проверяются сферические кластеры и сферические оболочки объектов, не содержащихся ни в одном из сферических кластеров, но принадлежащих кластеру сцены. Если луч не пересекает сферический кластер, то сам этот кластер и все объекты или кластеры, содержащиеся в нем, исключаются из дальнейшего рассмотрения. Если же луч пересекает кластер, то эта процедура рекурсивно повторяется до тех пор, пока не будут рассмотрены все объекты. Если луч пересекает сферическую оболочку некоего объекта в какой-нибудь точке, то этот объект заносится в список активных объектов. Эта процедура значительно сокращает количество вычислений точек пересечения луча со сферическими оболочками и тем самым повышает эффективность всего алгоритма.

Вторая модификация использует упорядочение по приоритету, чтобы сократить число объектов, для которых вычисляются пересечения с лучом. Вместо того, чтобы немедленно производить вычисление пересечения объекта к лучом, как это делается в изложенном выше простом алгоритме, объект помещается в список пересе-

ченных объектов. После рассмотрения всех объектов сцены преобразованный список пересеченных объектов упорядочивается по приоритету глубины (см. разд. 4.8). Для определения приоритетного порядка можно использовать центры сферических оболочек или наибольшие (наименьшие) значения $z$ прямоугольных оболочек. Пересечения луча с объектами из списка пересеченных объектов определяются в порядке их приоритетов. К сожалению, как ранее указывалось в разд. 4.8, точка пересечения луча с первым из объектов в упорядоченном по приоритетам списке пересеченных объектов не обязательно будет видимой. Необходимо определить точки пересечения луча со всеми потенциально видимыми объектами из множества $\{Q\}$ (подробности см. в разд. 4.8) и занести их в список пересечений. Затем модифицированный алгоритм упорядочивает этот список пересечений так, как это делалось и в простом алгоритме. К счастью, множество $\{Q\}$ потенциально видимых объектов обычно значительно меньше числа объектов в списке пересеченных лучом. Следовательно, эффективность алгоритма возрастет. Обе эти модификации применимы также и к общему алгоритму трассировки лучей, учитывающему отражение, преломление и прозрачность, который обсуждается в разд. 5.12.

Изложенный выше простой алгоритм не использует того обстоятельства, что некоторые грани многогранника являются нелицевыми и их можно сразу удалить, не учитывается здесь и возможная когерентность сцены. Например, несущественен порядок обработки пикселов. Вместе с тем рассмотрение этих пикселов в порядке сканирования строки развертки позволило бы воспользоваться в алгоритме когерентностью сканирующих строк. Другой подход может заключаться в подразделении сцены в духе алгоритма Варнока, причем учет когерентности областей привел бы к уменьшению числа объектов, рассматриваемых для каждого луча и, следовательно, к повышению эффективности алгоритма. Хотя использование подобных приемов повышает эффективность алгоритма определения видимости непрозрачных поверхностей, их невозможно применить в общем алгоритме трассировки лучей, который учитывает отражение, преломление и прозрачность. Например, если в алгоритме учтено отражение, то объект, который полностью закрыт другим объектом, может оказаться видимым, как отражение от третьего объекта. Поскольку метод трассировки лучей является методом грубой силы, алгоритмы определения видимости непрозрачных поверхностей, обсуждавшиеся в предыдущих разделах, яв-

ляются более эффективными, и именно ими нужно пользоваться[1].

Рот [4-46] указал, что алгоритм трассировки лучей можно использовать также и для создания каркасных чертежей сплошных тел. При этом предполагается, что лучи порождаются в том порядке, в каком происходит сканирование экрана, т. е. сверху вниз и слева направо. Получающаяся процедура такова:

> Если видимая поверхность для Пиксел($x$, $y$) соответствует фону или отличается от видимой поверхности для Пиксел($x - 1$, $y$) *или* для Пиксел($x$, $y - 1$), то изобразить этот пиксел. В противном случае пиксел не изображать.

Алгоритм трассировки лучей можно использовать, кроме того, для определения физических свойств сплошного тела. Полное рассмотрение этого вопроса не входит в задачу данной книги. Однако для иллюстрации этого подхода приведем один пример. В частности, объем любого сплошного тела можно определить, аппроксимируя его суммой маленьких прямоугольных параллелепипедов. Это можно проделать, породив множество параллельных лучей, расположенных на определенных расстояниях друг от друга. Точки пересечения каждого луча с заданным объемом вычисляются и упорядочиваются вдоль направления этого луча. Если подвергнуть луч переносу, совмещающему его с осью $z$, как это было описано выше, то объем каждого прямоугольного параллелепипеда будет равен:

$$V = l_x l_y[(z_1 - z_2) + (z_3 - z_4) + \cdots + (z_{n-1} - z_n)]$$

где $l_x$ и $l_y$ — расстояние между лучами по горизонтали и вертикали соответственно. Каждое слагаемое $(z_{i-1} - z_i)$ представляет собой участок луча, лежащий внутри заданного тела. Объем тела, следовательно, равен сумме объемов всех таких прямоугольных параллелепипедов. Точность результатов зависит от числа использованных лучей. Точность можно повысить, умеренно увеличив объем вычислений и рекурсивно уменьшив размер «пиксела», в том случае, если объемы смежных прямоугольных параллелепипедов различаются более чем на заранее заданную величину. При таком подходе точ-

---

[1] Экспериментальные оценки алгоритмов, описанных в предыдущих разделах, полученные на программах, написанных на одном языке и запущенных на одной вычислительной системе, для сцены, описанной в примерах 4.19, 4.22—4.24, связаны между собой следующим отношением: Трассировка лучей: Варнок: Уоткинс: Сканирование с $z$-буфером: $z$-буфер ~ 9.2:6.2:2.1:1.9:1.

нее определяются объемы тех элементов тела, где имеют место быстрые изменения, например в окрестностях ребер тел, ограниченных криволинейными поверхностями.

Ввиду внутренне присущей алгоритму трассировки лучей параллельности вычислений (здесь все лучи обрабатываются одинаково и независимо друг от друга) его можно реализовать аппаратно на основе сверхбольших интегральных схем (СБИС) с использованием методов параллельной обработки.

## 4.14. РЕЗЮМЕ

В предыдущих разделах детально обсуждалось несколько основных алгоритмов, используемых для решения задачи удаления невидимых линий или поверхностей. Не следует думать, будто эти алгоритмы представляют собой все, что имеется. Однако, усвоив изложенные идеи, читатель сможет разбираться в новых алгоритмах по мере их появления или самостоятельно создавать алгоритмы применительно к конкретному приложению.

Например, недавно предложенный Хеджли [4-47] алгоритм удаления невидимых линий основан на идеях, продемонстрированных в алгоритме, использующем список приоритетов, который принадлежит Ньюэлу, Ньюэлу и Санча (разд. 4.8), в алгоритме разбиения области Варнока (разд. 4.4), в алгоритме построчного сканирования Уоткинса (разд. 4.9) и в тестах пересечения и видимости, встречающихся в разных разделах данной главы. Этот алгоритм работает в пространстве объекта, получает на входе выпуклые или невыпуклые многоугольники, а оценка его эффективности линейно зависит от числа объектов.

Другим примером является предложенный Азертоном [4-48] модифицированный интервальный алгоритм построчного сканирования (см. разд. 4.11), который используется для визуализации сцен, получающихся в системе конструктивного моделирования сплошных тел. Внутренний цикл этого алгоритма изменен так, чтобы можно было реализовать одномерные теоретико-множественные операции, необходимые для моделирующей системы, которая пользуется алгоритмом трассировки лучей (см. разд. 4.13). Азертон указывает, что этот модифицированный интервальный алгоритм построчного сканирования требует примерно в 60 раз меньше времени, чем обычный алгоритм трассировки лучей.

4-1 Sutherland, Ivan E., Sproull, Robert F., and Schumacker, R. A., "A Characterization of Ten Hidden-Surface Algorithms," *Computing Surveys*, Vol. 6, pp. 1–55, 1974.

4-2 Williams, Hugh, "Algorithm 420, Hidden-Line Plotting Program," *CACM*, Vol. 15, pp. 100–103, 1972.

4-3 Wright, T. J, "A Two-Space Solution to the Hidden Line Problem for Plotting Functions of Two Variables," *IEEE Trans. Comput.*, Vol. C–22, pp. 28–33, 1973.

4-4 Watkins, Steven, L., "Algorithm 483, Masked Three-Dimensional Plot Program with Rotations," *CACM*, Vol. 17, pp. 520–523, 1974.

4-5 Butland, J., "Surface Drawing Made Simple," *CAD Journal*, Vol. 11, pp. 19–22, 1979.

4-6 Gottlieb, M.,"Hidden Line Subroutines for Three Dimensional Plotting," *Byte*, Vol. 3, No. 5, pp. 49–58, 1978.

4-7 Roberts, L. G., "Machine Perception of Three Dimensional Solids," MIT Lincoln Lab. Rep., TR 315, May 1963. Also in J. T. Tippet et al. (eds.). *Optical and Electro-Optical Information Processing*, MIT Press, Cambridge pp. 159–197, 1964.

4-8 Petty, J. S., and Mach, K. D., "Contouring and Hidden-line Algorithms for Vector Graphic Displays," Air Force Applied Physics Lab. Rep., AFAPL-TR-77-3, Jan. 1977, ADA 040 530.

4-9 Rogers, David, F., Meier, William, and Adlum, Linda, "Roberts Algorithm," U.S. Naval Academy, Computer Aided Design/Interactive Graphics Group Study, 1982, unpublished.

4-10 Warnock, John, E., "A Hidden Line Algorithm for Halftone Picture Representation," University of Utah Computer Science Dept. Rep., TR 4-5, May 1968, NTIS AD 761 995.

4-11 Warnock, John, E. "A Hidden-Surface Algorithm for Computer Generated Halftone Pictures," University of Utah Computer Science Dept. Rep., TR 4-15, June 1969, NTIS AD 753 671.

4-12 Weiler, K., and Atherton, P., "Hidden Surface Removal Using Polygon Area Sorting," *Computer Graphics*, Vol. 11, pp. 214–222 (Proc. SIGGRAPH 77).

4-13 Catmull, Edwin, "A Subdivision Algorithm for Computer Display of Curved Surfaces," Ph.D. Thesis, University of Utah, Dec. 1974. Also UTEC-CSc-74-133, and NTIS A004 968.

4-14 Catmull, Edwin, "Computer Display of Curved Surfaces," *Proc. IEEE Conf. Comput. Graphics Pattern Recognition Data Struct.*, May 1975, p. 11.

4-15 Cohen, Elaine, Lyche, Tom, and Riesenfeld, Richard, F., "Discrete B-splines and Subdivision Techniques in Computer-Aided Geometric Design and Computer Graphics," *Computer Graphics and Image Processing*, Vol. 14, pp. 87–111, 1980. Also University of Utah, Computer Science Dept. Rep., UUCS-79-117, Oct. 1979.

4-16 Newell, M.E., Newell, R. G., and Sancha, T. L., "A New Approach to the Shaded Picture Problem," *Proc. ACM Natl. Conf.*, 1972, pp. 443–450.

4-17 Newell, M. E., "The Utilization of Procedure Models in Digital Image Synthesis," Ph.D. Thesis, University of Utah, 1974. Also UTEC-CSc-76-218 and NTIS AD/A 039 008/LL.

4-18 Schachter, Bruce J., *Computer Image Generation*, John Wiley, New York, 1982.

4-19 Schumacker, R. A., Brand, B., Gilliland, M., and Sharp, W., "Study for Applying Computer-generated Images to Visual Simulation," U.S. Air Force Human Resources Lab. Tech. Rep., AFHRL-TR-69-14, Sept. 1969, NTIS AD 700 375.

4-20 Fuchs, H., Abram, G. D., and Grant, E. D., "Near Real-Time Shaded Display of Rigid Objects," *Computer Graphics*, Vol. 17, pp. 65–72, 1983 (Proc. SIGGRAPH 83).

4-21 Wylie, C., Romney, G. W., Evans, D. C., and Erdahl, A. C., "Halftone Perspective Drawings by Computer," *FJCC 1967*, Thompson Books, Washington, D.C., pp. 49–58.

4-22 Bouknight, W. J., "An Improved Procedure for Generation of Half-tone Computer Graphics Representations," University of Illinois Coordinated Science Lab. Tech. Rep., R-432, Sept. 1969.

4-23 Bouknight, W. J., and Kelly, K. C., "An Algorithm for Producing Half-tone Computer Graphics Presentations with Shadows and Movable Light Sources," *SJCC 1970*, AFIPS Press, Montvale, N. J. pp. 1–10.

4-24 Bouknight, W. J., "A Procedure for Generation of Three-dimensional Half-toned Computer Graphics Representations," *CACM*, Vol. 13, pp. 527–536, 1970.

4-25 Watkins, G. S., "A Real-Time Visible Surface Algorithm," University of Utah Computer Science Dept. Tech. Rep., UTEC-CSC-70-101, June 1970, NTIS AD 762 004.

4-26 Myers, A. J., "An Efficient Visible Surface Program," Report to the NSF, Ohio State University Computer Graphics Research Group, July 1975.

4-27 Romney, G. W., Watkins, G. S., and Evans, D. C., "Real Time Display of Computer Generated Half-tone Perspective Pictures," *IFIP 1968*, North-Holland, Amsterdam, pp. 973–978.

4-28 Hamlin, G., and Gear, C., "Raster-Scan Hidden Surface Algorithm Techniques," *Computer Graphics*, Vol. 11, pp. 206–213, 1977 (Proc. SIGGRAPH 77).

4-29 Jackson, J. H., "Dynamic Scan-converted Images with a Frame Buffer Display Device," *Computer Graphics*, Vol. 14, pp. 163–169, 1980 (Proc. SIGGRAPH 80).

4-30 Archuleta, M., "Hidden Surface Line Drawing Algorithm," University of Utah Computer Science Dept. Tech. Rep., UTEC-CSc-72-121, June 1972.

4-31 Blinn, J. F., "A Scan Line Algorithm for the Computer Display of Parametrically Defined Surfaces," *Computer Graphics*, Vol. 12, 1978 (supplement to Proc. SIGGRAPH 78); see also Ref. 4-33.

4-32 Whitted, T., "A Scan-line Algorithm for Computer Display of Curved Surfaces," *Computer Graphics*, Vol. 12, 1978 (supplement to Proc. SIGGRAPH 78); see also Ref. 4-33.

4-33 Lane, J. M., Carpenter, L. C., Whitted, T., and Blinn, J. F., "Scan Line Methods for Displaying Parametrically Defined Surfaces," *CACM*, Vol. 23, pp. 23–34, 1980.

4-34 Lane, J. M., and Carpenter, L. C., "A Generalized Scan Line Algorithm for the Computer Display of Parametrically Defined Surfaces," *Computer Graphics and Image Processing*, Vol. 11, pp. 290–297, 1979.

4-35 Clark, J. H., "A Fast Scan-line Algorithm for Rendering Parametric Surfaces," *Computer Graphics*, Vol. 13, 1979 (supplement to Proc. SIGGRAPH 79).

4-36 Kunz, K. S., *Numerical Analysis*, McGraw-Hill, New York, 1957.

4-37 Kajiya, J. T., "Ray Tracing Parametric Patches," *Computer Graphics*, Vol. 16, pp. 245–254, 1982 (Proc. SIGGRAPH 82).

4-38 Porter, T., "Spherical Shading." *Computer Graphics*, Vol. 12, pp. 282–285, 1978 (Proc. SIGGRAPH 78).

4-39 Porter, T., "The Shaded Surface Display of Large Molecules," *Computer Graphics*, Vol. 13, pp. 234–236, 1979 (Proc. SIGGRAPH 79).

4-40 Staudhammer, J., "On the Display of Space Filling Atomic Models in Real Time," *Computer Graphics*, Vol. 12, pp. 167–172, 1978 (Proc. SIGGRAPH 78).

4-41 Appel, A., "Some Techniques for Shading Machine Renderings of Solids," *AFIPS 1968 Spring Joint Comput. Conf.*, pp. 37–45.

4-42 Goldstein, R. A., and Nagel, R., "3-D Visual Simulation," *Simulation*, pp. 25–31, January 1971.

4-43 Kay, Douglas S., "Transparency, Refraction and Ray Tracing for Computer Synthesized Images," Masters thesis, Program of Computer Graphics, Cornell University, Jan. 1979.

4-44 Kay, Douglas, S., and Greenberg, Donald, "Transparency for Computer Synthesized Images," *Computer Graphics*, Vol. 13, pp. 158–164, 1979 (Proc. SIGGRAPH 79).

4-45 Whitted, J. T., "An Improved Illumination Model for Shaded Display," *CACM*, Vol. 23, pp. 343–349 (Proc. SIGGRAPH 79).

4-46 Roth, Scott D., "Ray Casting for Modeling Solids," *Computer Graphics and Image Processing*, Vol. 18, pp. 109–144, 1982.

4-47 Hedgley, David R. Jr., "A General Solution to the Hidden-Line Problem," *NASA* Ref. Pub. 1085, March 1982.

4-48 Atherton, Peter R., "A Scan-line Hidden Surface Removal Procedure for Constructive Solid Geometry," *Computer Graphics*, Vol. 17, pp. 73–82, 1983 (Proc. SIGGRAPH 83).

4-49 Whitted, Turner, and Weimer, David M., "A Software Testbed for the Development of 3D Raster Graphics Systems," *ACM Trans. on Graphics*, Vol. 1, pp. 43–58, 1982.

ЛИТЕРАТУРА НА РУССКОМ ЯЗЫКЕ

4-7 Робертс Л. Автоматическое восприятие трехмерных объектов.— Интегральные роботы.— М.: Мир, 1973, с. 162—208.

# 5 ⟫

# Построение реалистических изображений

## 5.1. ВВЕДЕНИЕ

Построение реалистических изображений включает как физические, так и психологические процессы. Свет, т. е. электромагнитная энергия, после взаимодействия с окружающей средой попадает в глаз, где в результате физических и химических реакций вырабатываются электроимпульсы, воспринимаемые мозгом. Восприятие — это приобретаемое свойство. Психология зрительного восприятия широко изучалась, и о ней много написано, однако эта тема выходит за рамки данной книги. Проблемы зрительного восприятия хорошо описаны в книге [5-1].

Человеческий глаз — очень сложная система. Он имеет почти сферическую форму с диаметром около 20 мм. Воспринимаемый свет с помощью гибкого хрусталика фокусируется на сетчатке глаза, в которой есть два типа рецепторов: колбочки и палочки. В центре задней полусферы глаза собрано 6—7 млн. колбочек, чувствительных только к сравнительно высоким уровням освещенности, причем каждая из них присоединена к отдельному нерву. Колбочки позволяют различать мелкие детали. В сетчатке также находится 75—150 млн. палочек, чувствительных к очень низким уровням освещенности. К одному нерву присоединено сразу несколько палочек, поэтому они не способны различать мелкие детали. Интересно, что цвет воспринимается только колбочками, т. е. при низкой освещенности, когда колбочки теряют свою чувствительность, предметы кажутся черно-белыми.

Из опытов известно, что чувствительность глаза к яркости света изменяется по логарифмическому закону. Пределы чувствительности к яркости чрезвычайно широки, порядка $10^{10}$, однако глаз не в состоянии одновременно воспринять весь этот диапазон. Глаз реагирует на гораздо меньший диапазон значений относительно ярко-

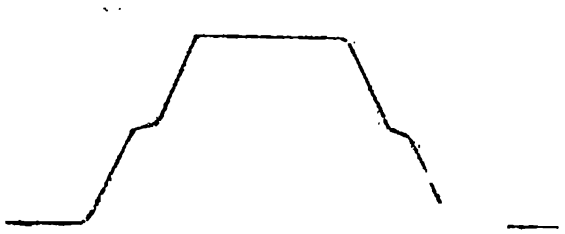
a b

Рис. 5.1. Одновременный контраст.

сти, распределенный вокруг уровня адаптации к освещенности. Чувствительность к относительной яркости имеет порядок 100—150 (2.2 логарифмической единицы). Скорость адаптации к яркости неодинакова для различных частей сетчатки, но тем не менее очень высока. Экстремумы диапазона относительной яркости воспринимаются соответственно как черный и белый.

Глаз приспосабливается к «средней» яркости обозреваемой сцены; поэтому область с постоянной яркостью (интенсивностью) на темном фоне кажется ярче или светлее, чем на светлом фоне. Это явление называется одновременным контрастом (рис. 5.1). Яркость центрального квадрата на шкале от 0 до 1 равна 0.5, а охватывающего — 0.2 (рис. 5.1, a). На рис. 5.1, b яркость центрального квадрата такая же, а охватывающего — 0.8. То же самое происходит при наблюдении уличного фонаря днем и ночью: если смотреть на фонарь днем (рис. 5.1, a), то средняя освещенность сцены выше, чем ночью (рис. 5.1, b). Поэтому уровень контраста ниже, и кажется, что интенсивность (яркость) фонаря или центрального квадрата на рис. 5.1, a меньше. Похожее на одновременный контраст явление существует и для цветов.

Еще одним свойством глаза, имеющим значение для машинной графики, является то, что границы областей постоянной интенсивности кажутся более яркими, в результате чего области с постоянной интенсивностью воспринимаются, как имеющие переменную интенсивность. Это явление называется эффектом полос Маха по имени открывшего его австрийского физика Эрнста Маха. Эффект полос Маха наблюдается, когда резко изменяется наклон кривой интенсивности. Если кривая интенсивности вогнута, то в этом месте поверхность кажется светлее, если выпукла — темнее (рис. 5.2).



a b

Рис. 5.2. Эффект полос Маха: (a) кусочно-линейная функция интенсивности, (b) функция интенсивности с непрерывной первой производной. (С разрешения университета штата Юга, [5-2].)

Эффект полос Маха особенно хорошо заметен на полутоновых поверхностях, заданных многоугольниками. Если интенсивность многоугольников определяется с учетом направления вектора нормали, то на ребрах этих многоугольников интенсивность будет резко меняться. Эффект полос Маха, мешающий глазу создавать сглаженное изображение сцены, виден на рис. 5.3, a. Увеличивая количество полигональных граней, его можно ослабить, но полностью уничтожить нельзя (рис. 5.3, b).

a b

Рис. 5.3. Эффект полос Маха на поверхностях, образованных плоскими полигональными гранями: (а) модель с 8 гранями, (б) модель с 32 гранями. (С разрешения университета штата Юта, [5-2].)

## 5.2. ПРОСТАЯ МОДЕЛЬ ОСВЕЩЕНИЯ

Световая энергия, падающая на поверхность, может быть поглощена, отражена или пропущена. Частично она поглощается и превращается в тепло, а частично отражается или пропускается. Объект можно увидеть, только если он отражает или пропускает свет; если же объект поглощает весь падающий свет, то он невидим и называется абсолютно черным телом. Количество поглощенной, отраженной или пропущенной энергии зависит от длины волны света. При освещении белым светом, в котором интенсивность всех длин волн снижена примерно одинаково, объект выглядит серым. Если поглощается почти весь свет, то объект кажется черным, а если только небольшая его часть — белым. Если поглощаются лишь определенные длины волн, то у света, исходящего от объекта, изменяется распределение энергии и объект выглядит цветным. Цвет объекта определяется поглощаемыми длинами волн.

Свойства отраженного света зависят от строения, направления и формы источника света, от ориентации и свойств поверхности. Отраженный от объекта свет может также быть диффузным или зеркальным. Диффузное отражение света происходит, когда свет как бы проникает под поверхность объекта, поглощается, а затем вновь испускается. При этом положение наблюдателя не имеет значения, так как диффузно отраженный свет рассеивается равномерно по всем направлениям. Зеркальное отражение происходит от внешней поверхности объекта.

Свет точечного источника отражается от идеального рассеивателя по закону косинусов Ламберта: интенсивность отраженного света пропорциональна косинусу угла между направлением света и нормалью к поверхности, т. е.

$$I = I_l k_d \cos\theta \qquad 0 \leq \theta \leq \pi/2$$

где $I$ — интенсивность отраженного света, $I_l$ — интенсивность точечного источника, $k_d$ — коэффициент диффузного отражения ($0 \leqslant k_d \leqslant 1$), $\theta$ — угол между направлением света и нормалью к поверхности (рис. 5.4). Если $\theta > \pi/2$, то источник света расположен за объектом. Коэффициент диффузного отражения $k_d$ зависит от материала и длины волны света, но в простых моделях освещения обычно считается постоянным.

Поверхность предметов, изображенных при помощи простой модели освещения с ламбертовым диффузным отражением, выглядит блеклой и матовой. Предполагается, что источник точечный,

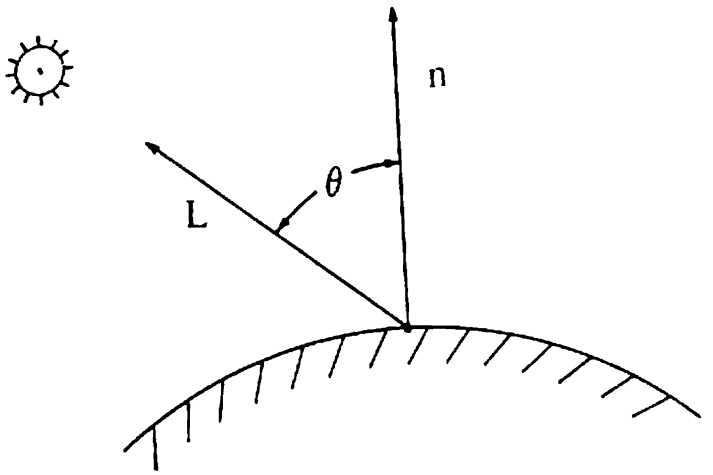



Рис. 5.4. Диффузное отражение.

поэтому объекты, на которые не падает прямой свет, кажутся черными. Однако на объекты реальных сцен падает еще и рассеянный свет, отраженный от окружающей обстановки, например от стен комнаты. Рассеянному свету соответствует распределенный источник. Поскольку для расчета таких источников требуются большие вычислительные затраты, в машинной графике они заменяются на коэффициент рассеяния — константу, которая входит в формулу в линейной комбинации с членом Ламберта:

$$I = I_a k_a + I_l k_d \cos\theta \qquad 0 \leq \theta \leq \pi/2 \tag{5.1}$$

где $I_a$ — интенсивность рассеянного света, $k_a$ — коэффициент диффузного отражения рассеянного света ($0 \leqslant k_a \leqslant 1$).

Пусть даны два объекта, одинаково ориентированные относительно источника, но расположенные на разном расстоянии от него. Если найти их интенсивность по данной формуле, то она окажется одинаковой. Это значит, что, когда предметы перекрываются, их невозможно различить, хотя интенсивность света обратно пропорциональна квадрату расстояния от источника, и объект, лежащий дальше от него, должен быть темнее. Если предположить, что источник света находится в бесконечности, то диффузный член модели освещения обратится в нуль. В случае перспективного преобразования сцены в качестве коэффициента пропорциональности для диффузного члена можно взять расстояние $d$ от центра проекции до объекта. Но если центр проекции лежит близко к объекту, то $1/d^2$ изменяется очень быстро, т. е. у объектов, лежащих примерно на одинаковом расстоянии от источника, разница интенсивностей чрезмерно велика. Как показывает опыт, большей реалистичности можно добиться при линейном затухании. В этом случае модель освещения выглядит так:

$$I = I_a k_a + \frac{I_l k_d \cos\theta}{d + K} \tag{5.2}$$

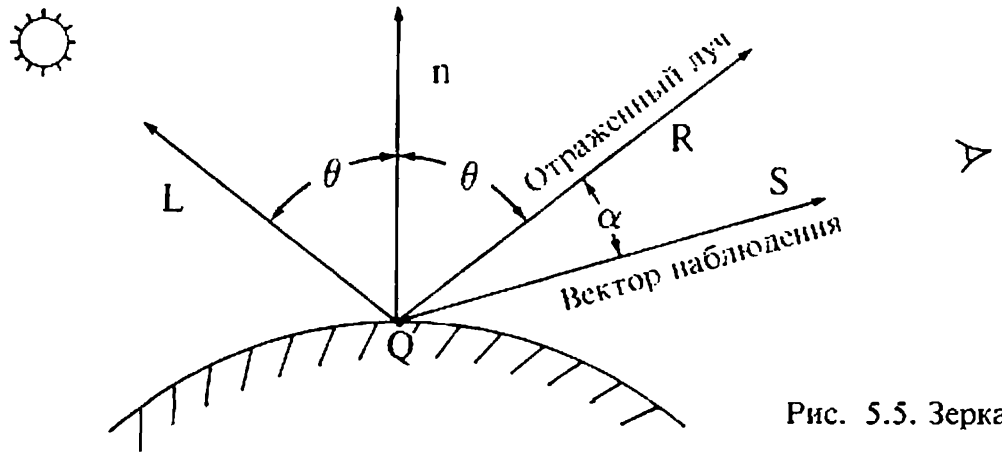



Рис. 5.5. Зеркальное отражение.

где $K$ — произвольная постоянная.

Если предполагается, что точка наблюдения находится в бесконечности, то $d$ определяется положением объекта, ближайшего к точке наблюдения. Это означает, что ближайший объект освещается с полной интенсивностью источника, а более далекие — с уменьшенной. Для цветных поверхностей модель освещения применяется к каждому из трех основных цветов.

Интенсивность зеркально отраженного света зависит от угла падения, длины волны падающего света и свойств вещества. Основное уравнение Френеля приводится в любой книге по геометрической оптике. Зеркальное отражение света является направленным. Угол отражения от идеальной отражающей поверхности (зеркала) равен углу падения, в любом другом положении наблюдатель не видит зеркально отраженный свет. Это означает, что вектор наблюдения **S** (рис. 5.5) совпадает с вектором отражения **R**, и угол $\alpha$ равен нулю. Если поверхность не идеальна, то количество света, достигающее наблюдателя, зависит от пространственного распределения зеркального отраженного света. У гладких поверхностей распределение узкое или сфокусированное, у шероховатых — более широкое.

Благодаря зеркальному отражению на блестящих предметах появляются световые блики. Из-за того что зеркально отраженный свет сфокусирован вдоль вектора отражения, блики при движении наблюдателя тоже перемещаются. Более того, так как свет отражается от внешней поверхности (за исключением металлов и некоторых твердых красителей), то отраженный луч сохраняет свойства падающего. Например, при освещении блестящей синей поверхности белым светом возникают белые, а не синие блики.

В простых моделях освещения обычно пользуются эмпирической моделью Буи-Туонга Фонга [5—2], так как физические свойства зеркального отражения очень сложны. Модель Фонга имеет вид:

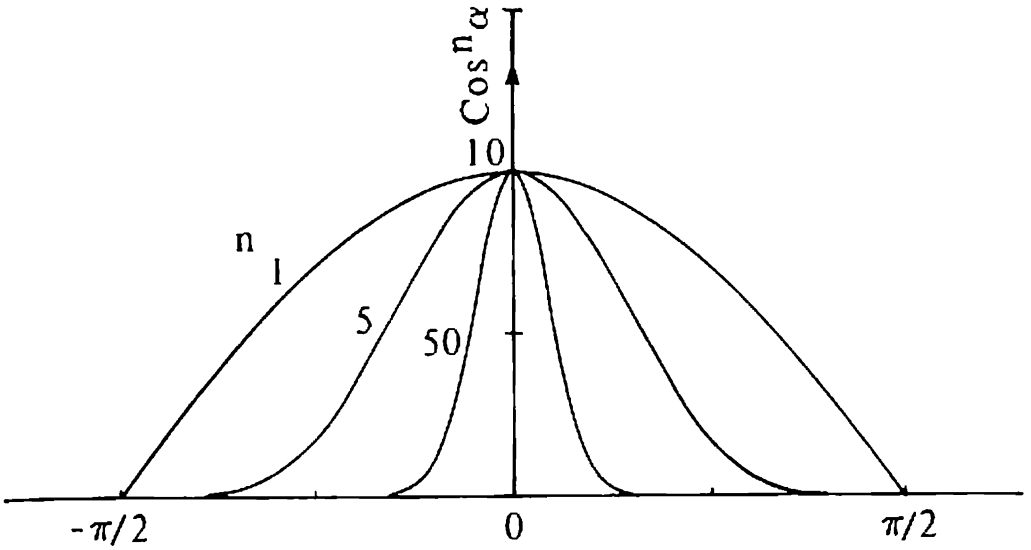



Рис. 5.6. Приближенные функции пространственного распределения для зеркального отражения.

$$I_s = I_l w(i, \lambda) \cos^n \alpha \tag{5.3}$$

где $w(i, \lambda)$ — кривая отражения, представляющая отношение зеркально отраженного света к падающему как функцию угла падения $i$ и длины волны $\lambda$; $n$ — степень, аппроксимирующая пространственное распределение зеркально отраженного света. На рис. 5.6 показана функция $\cos^n \alpha$ при $-\pi/2 \leqslant \alpha \leqslant \pi/2$ для различных $n$: большие значения $n$ дают сфокусированные пространственные распределения характеристик металлов и других блестящих поверхностей, а малые — более широкие распределения для неметаллических поверхностей, например бумаги.

Коэффициент зеркального отражения зависит от угла падения, однако даже при перпендикулярном падении зеркально отражается только часть света, а остальное либо поглощается, либо отражается диффузно. Эти соотношения определяются свойствами вещества и длиной волны. Коэффициент отражения для некоторых неметаллов может быть всего 4%, в то время как для металлических материалов — более 80%. На рис. 5.7, a коэффициенты отражения для типичных веществ при нормальном падении света показаны как функции длины волны, а на рис. 5.7, b — как функции угла падения. Обратите внимание, что при падении под скользящим углом ($\theta = 90°$) отражается весь падающий свет (коэффициент отражения 100%).

Объединяя эти результаты с формулой рассеянного света и диффузного отражения, получим модель освещения:

$$I = I_a k_a + \frac{I_l}{d + K}(k_d \cos\theta + w(i, \lambda)\cos^n \alpha) \tag{5.4}$$

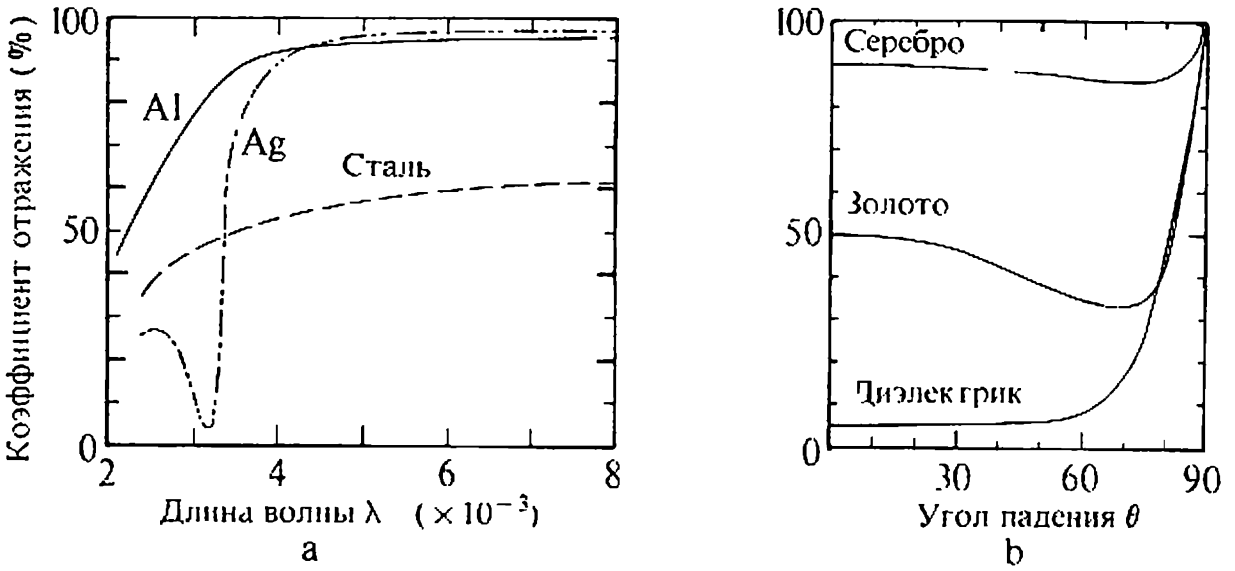



Рис. 5.7. Кривые отражения.

Функция $w(i, \lambda)$ довольно сложна, поэтому ее обычно заменяют константой $k_s$, которая либо выбирается из эстетических соображений, либо определяется экспериментально. Таким образом,

$$I = I_a k_a + \frac{I_l}{d + K}(k_d \cos\theta + k_s \cos^n \alpha) \tag{5.5}$$

В машинной графике эта модель часто называется функцией закраски и применяется для расчета интенсивности или тона точек объекта или пикселов изображения. Чтобы получить цветное изображение, нужно найти функции закраски для каждого из трех основных цветов. Константа $k_s$ обычно одинакова для всех трех основных цветов, поскольку цвет зеркально отраженного света определяется цветом падающего.

Если имеется несколько источников света, то их эффекты суммируются. В этом случае модель освещения определяется как

$$I = I_a k_a + \sum_{j=1}^{m} \frac{I_{l_j}}{d + K}(k_d \cos\theta_j + k_s \cos_j^n \alpha_j) \tag{5.6}$$

где $m$ — количество источников.

Применяя формулу скалярного произведения двух векторов, запишем

$$\cos\theta = \frac{\mathbf{n} \cdot \mathbf{L}}{|\mathbf{n}||\mathbf{L}|} = \hat{\mathbf{n}} \cdot \hat{\mathbf{L}}$$

где $\hat{\mathbf{n}}$ и $\hat{\mathbf{L}}$ — единичные векторы соответственно нормали к поверхности и направления к источнику.

Точно так же

$$\cos\alpha = \frac{\mathbf{R}\cdot\mathbf{S}}{|\mathbf{R}||\mathbf{S}|} = \hat{\mathbf{R}}\cdot\hat{\mathbf{S}}$$

где $\hat{\mathbf{R}}$ и $\hat{\mathbf{S}}$ — единичные векторы, определяющие направления отраженного луча и наблюдения. Следовательно, модель освещения для одного источника определяется как

$$I = I_a k_a + \frac{I_l}{d+K}[k_d(\hat{\mathbf{n}}\cdot\hat{\mathbf{L}}) + k_s(\hat{\mathbf{R}}\cdot\hat{\mathbf{S}})^n] \tag{5.7}$$

Рассмотрим эту простую модель на примере.

**Пример 5.1.** Простая модель освещения

Предположим, что на рис. 5.5 в точке $Q$ поверхности векторы нормали, падающего света и наблюдения следующие:

$$\mathbf{n} = \mathbf{j}$$
$$\mathbf{L} = -\mathbf{i} + 2\mathbf{j} - \mathbf{k}$$
$$\mathbf{S} = \mathbf{i} + 1.5\mathbf{j} + 0.5\mathbf{k}$$

Тогда вектор отражения $\mathbf{R}$ есть

$$\mathbf{R} = \mathbf{i} + 2\mathbf{j} + \mathbf{k}$$

Пусть на сцене находится только один объект, $d = 0, K = 1$ и интенсивность источника будет в 10 раз больше, чем интенсивность рассеянного света, т. е. $I_a = 1$, а $I_l = 10$. Объект имеет блестящую металлическую поверхность, поэтому в основном свет будет отражаться зеркально. Пусть $k_s = 0.8, k_a = k_d = 0.15$ и $n = 5$. Заметим, что $k_s + k_d = 0.95$, т. е. 5% энергии источника поглощается. Определяя элементы модели освещения, получаем

$$\hat{\mathbf{n}}\cdot\hat{\mathbf{L}} = \frac{\mathbf{n}\cdot\mathbf{L}}{|\mathbf{n}||\mathbf{L}|} = \frac{\mathbf{j}\cdot(-\mathbf{i}+2\mathbf{j}-\mathbf{k})}{\sqrt{(-1)^2+(2)^2+(-1)^2}} = \frac{2}{\sqrt{6}}$$

или

$$\theta = \cos^{-1}\left(2/\sqrt{6}\right) = 35.26^\circ$$

и

$$\hat{\mathbf{R}}\cdot\hat{\mathbf{S}} = \frac{\mathbf{R}\cdot\mathbf{S}}{|\mathbf{R}||\mathbf{S}|} = \frac{(\mathbf{i}+2\mathbf{j}+\mathbf{k})\cdot(\mathbf{i}+1.5\mathbf{j}+0.5\mathbf{k})}{\sqrt{(1)^2+(2)^2+(1)^2}\sqrt{(1)^2+(1.5)^2+(0.5)^2}} = \frac{4.5}{\sqrt{6}\sqrt{3.5}} = \frac{4.5}{\sqrt{21}}$$

или

$$\alpha = \cos^{-1}(4.5/\sqrt{21}) = 10.89^\circ$$

Окончательно

$$I = (1)(0.15) + (10/1)[(0.15)(2/\sqrt{6}) + (0.8)(4.5/\sqrt{21})^5]$$
$$= 0.15 + 10(0.12 + 0.73) = 8.65$$

Вектор наблюдения почти совпадает с вектором отражения, поэтому в точке $Q$ наблюдатель видит яркий блик. Однако при изменении положения наблюдателя, например, если

$$\mathbf{S} = \mathbf{i} \quad .5\mathbf{j} - 0.5\mathbf{k}$$

то

$$\hat{\mathbf{r}} \cdot \hat{\mathbf{S}} \quad \frac{\sqrt{2}}{\quad} = \frac{3.5}{\sqrt{2}}$$

и

$$= 40^{\circ}$$

В этом случае

$$I = 0.15 + 1000 \cdot 1^{\circ} + 0.211 = 3.45$$

и яркость блика в точке $Q$ значительно ослабевает.

## 5.3. ОПРЕДЕЛЕНИЕ НОРМАЛИ К ПОВЕРХНОСТИ

Из предыдущего раздела видно, что нормаль к поверхности представляет ее локальную кривизну, а следовательно, и направление зеркального отражения. Если известно аналитическое описание поверхности, то нормаль вычисляется непосредственно. Но для многих поверхностей бывает задана лишь их полигональная аппроксимация. Зная уравнение плоскости каждой грани, можно найти направление внешней нормали (см. разд. 4.3).

Во многих алгоритмах удаления невидимых линий и поверхностей используются только ребра или вершины, поэтому, для того чтобы объединить их с моделью освещения, необходимо знать приближенное значение нормали на ребрах и в вершинах. Пусть заданы уравнения плоскостей полигональных граней, тогда нормаль к их общей вершине равна среднему значению нормалей ко всем многоугольникам, сходящимся в этой вершине. Например, на рис. 5.8 направление приближенной нормали в точке $V_1$ есть

$$\mathbf{n}_{V_1} = (a_0 + a_1 + a_4)\mathbf{i} + (b_0 + b_1 + b_4)\mathbf{j} + (c_0 + c_1 + c_4)\mathbf{k}$$

где $a_0, a_1, a_4, b_0, b_1, b_4, c_0, c_1, c_4$ — коэффициенты уравнений плоскостей трех многоугольников $P_0$, $P_1$, $P_4$, окружающих $V_1$. Отметим, что если требуется найти только направление нормали, то делить результат на количество граней необязательно.

Если же уравнения плоскостей не заданы, то нормаль к вершине можно определить, усредняя векторные произведения всех ребер, пересекающихся в вершине. Еще раз рассматривая вершину $V_1$ на рис. 5.8, найдем направление приближенной нормали:

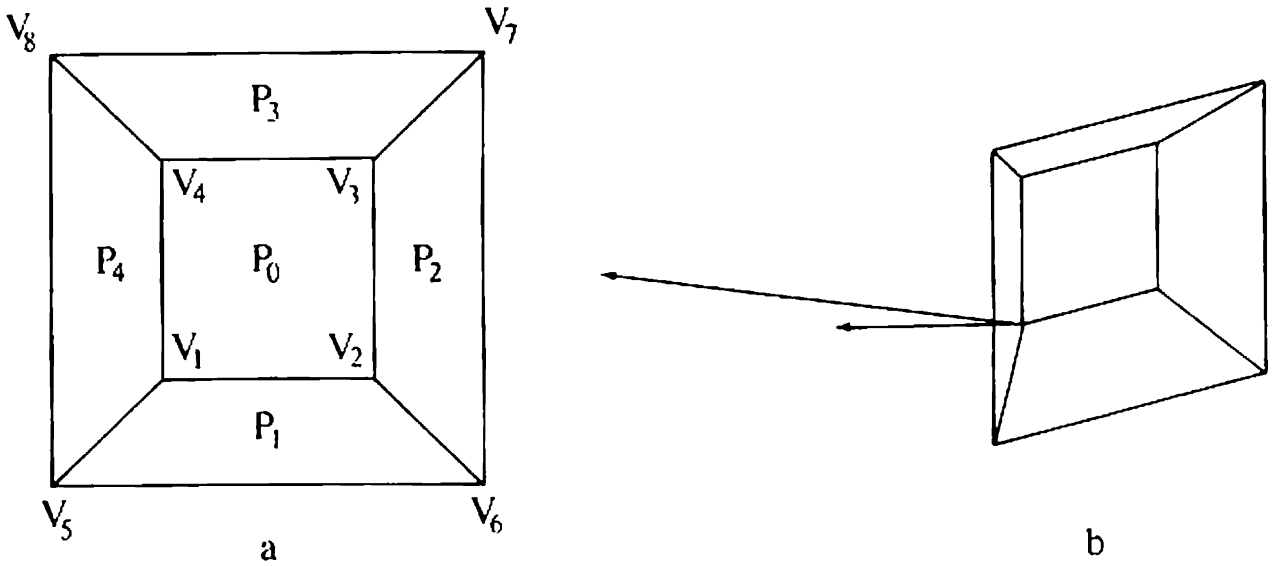



Рис. 5.8. Аппроксимация нормали к полигональной поверхности.

$$n_{V_1} = V_1V_2 \otimes V_1V_4 + V_1V_5 \otimes V_1V_2 + V_1V_4 \otimes V_1V_5$$

Следует обратить внимание на то, что необходимы только внешние нормали. Кроме того, если полученный вектор не нормируется, то его величина зависит от количества и площади конкретных многоугольников, а также от количества и длины конкретных ребер. Сильнее проявляется влияние многоугольников с большей площадью и более длинных ребер. Рассмотрим данный метод подробнее на примере.

**Пример 5.2.** Приближенное вычисление нормали к поверхности

Рассмотрим полигональную поверхность, изображенную на рис. 5.8, *a*. Координаты вершин: $V_1(-1, -1, 1)$, $V_2(1, -1, 1)$, $V_3(1, 1, 1)$, $V_4(-1, 1, 1)$, $V_5(-2, -2, 0)$, $V_6(2, -2, 0)$, $V_7(2, 2, 0)$, $V_8(-2, 2, 0)$. Это поверхность усеченной пирамиды. Уравнения плоскостей для граней $P_0$, $P_1$, $P_4$, окружающих вершину $V_1$:

$$\begin{aligned} P_0: &\quad 1 = 0 \\ P_1 &\quad -2 = 0 \\ P_4 &\quad -2 = 0 \end{aligned}$$

Приближенная нормаль в точке $V_1$ находится усреднением нормалей к окружающим многоугольникам:

$$\mathbf{n}_1 = (a_0 + a_1 \quad a_4)\mathbf{i} + (b_0 + b_1 \cdot b_4)\mathbf{j} \quad (c_0 + c_1 + c_4)\mathbf{k} \quad -\mathbf{i} \quad \mathbf{j} \quad 3\mathbf{k}$$

Абсолютная величина нормали $\mathbf{n}_1$ есть

$$|\mathbf{n}_1| = \sqrt{(\ \ 1)^2 + (-1)^2 + (3)^2} = \sqrt{11}$$

а единичная нормаль —

$$\frac{\mathbf{n}_1}{|\mathbf{n}_1|} \quad -0.3\mathbf{i} - 0.3\mathbf{j} + 0.9\mathbf{k}$$

Заметим, что простое деление на 3 не дает единичной нормали. Векторные произведения ребер, сходящихся в вершине $V_1$, суть

$$\mathbf{V_1V_2} \otimes \mathbf{V_1V_4} = 4\mathbf{k}$$
$$\mathbf{V_1V_5} \otimes \mathbf{V_1V_2} = -2\mathbf{j} + 2\mathbf{k}$$
$$\mathbf{V_1V_4} \otimes \mathbf{V_1V_5} = -2\mathbf{i} + 2\mathbf{k}$$

Усредняя векторные произведения, получаем приближенную нормаль в точке $V_1$.

$$\mathbf{n}_1 = -2\mathbf{i} - 2\mathbf{j} + 8\mathbf{k}$$

с абсолютной величиной

$$|\mathbf{n}_1| = \sqrt{(-2)^2 + (-2)^2 + (8)^2} = \sqrt{72}$$

Единичная нормаль есть

$$\frac{\mathbf{n}_1}{|\mathbf{n}_1|} = -0.24\mathbf{i} - 0.24\mathbf{j} + 0.94\mathbf{k}$$

Обратите внимание, что ненормированные нормали, полученные двумя способами, различаются как по величине, так и по направлению (рис. 5.8, b). Поэтому модель освещения дает несколько отличающиеся результаты, в зависимости от использованного метода аппроксимации.

Когда нормаль к поверхности используется для определения интенсивности и для изображения объекта или сцены выполняется перспективное преобразование, то нормаль следует вычислять до перспективного деления [1-1]. В противном случае направление нормали будет искажено, а это приведет к тому, что интенсивность, задаваемая моделью освещения, будет определена неправильно.

## 5.4. ОПРЕДЕЛЕНИЕ ВЕКТОРА ОТРАЖЕНИЯ

Для модели освещения чрезвычайно важно правильно задавать направление векторов отражения. В примере 5.1 вектор отражения был найден из наблюдения. В данном разделе рассматриваются три наиболее общих метода. По закону отражения вектор падающего света, нормаль к поверхности и вектор отражения лежат в одной плоскости, причем на этой плоскости угол падения равен углу отражения (рис. 5.9, a). Фонг [5-2] вывел отсюда простое решение для случая, когда свет падает вдоль оси $z$. Это предположение удобно для модели освещения с одним точечным источником. Если начало системы координат перенести в точку поверхности, то проекции нормали и вектора отражения на плоскость $xy$ будут лежать на одной прямой (рис. 5.9, b).

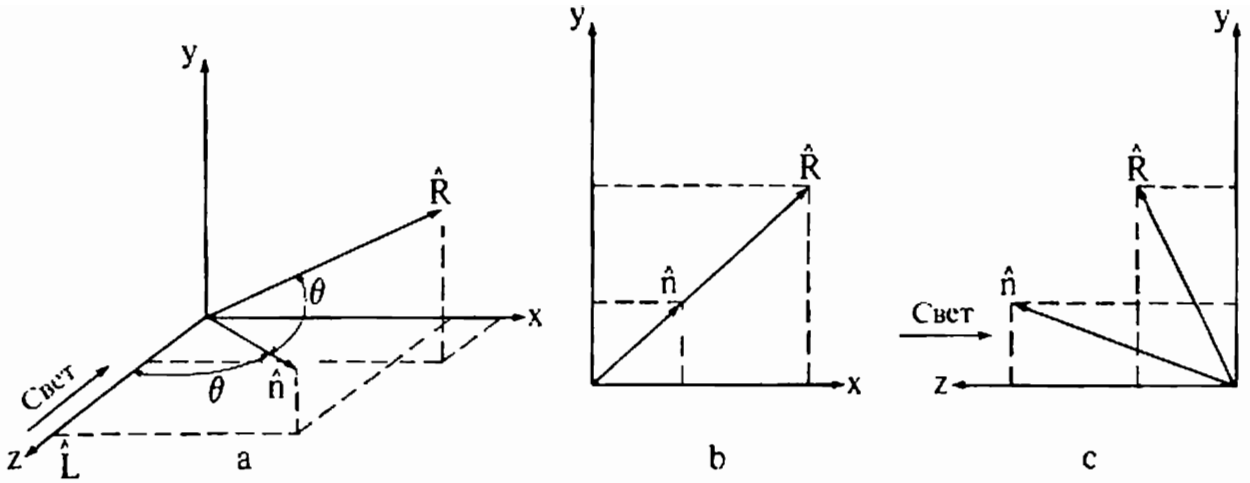



Рис. 5.9. Расчет направления отражения.

Таким образом,

$$\frac{\hat{R}_x}{\hat{R}_y} = \frac{\hat{n}_x}{\hat{n}_y} \tag{5.8}$$

где $\hat{R}_x$, $\hat{R}_y$, $\hat{n}_x$, $\hat{n}_y$ — суть $x$- и $y$-составляющие единичных векторов соответственно отражения и нормали.

Обозначим угол между единичным вектором нормали и осью $z$ через $\theta$. Тогда составляющая вектора нормали по оси $z$ есть

$$\hat{n}_z = \cos\theta \qquad 0 \le \theta \le \pi/2$$

Аналогично, угол между единичным вектором отражения и осью $z$ равен $2\theta$, поэтому

$$\hat{R}_z = \cos 2\theta = 2\cos^2\theta - 1 = 2\hat{n}_z^2 - 1 \tag{5.9}$$

Известно, что

$$R_x^2 + R_y^2 + R_z^2 = 1$$

и

$$\hat{R}_x^2 + \hat{R}_y^2 = 1 - \hat{R}_z^2 = 1 - \cos^2 2\theta$$

или

$$\hat{R}_y^2\left(\frac{\hat{R}_x^2}{\hat{R}_y^2} + 1\right) = 1 - \cos^2 2\theta$$

Используя отношение $x$- и $y$-составляющих векторов отражения и нормали (5.8) и соотношение

$$\hat{n}_x^2 + \hat{n}_y^2 + \hat{n}_z^2 = 1$$

получаем

$$\frac{\hat{R}_y^2}{\hat{n}_y^2}(\hat{n}_x^2 + \hat{n}_y^2) = \frac{\hat{R}_y^2}{\hat{n}_y^2}(1 - \hat{n}_z^2) = 1 - \cos^2 2\theta$$

Перепишем правую часть:

$$\frac{\hat{R}_y^2}{\hat{n}_y^2}(1 - \hat{n}_z^2) = 1 - (2\cos^2\theta - 1)^2 = 1 - (2\hat{n}_z^2 - 1)^2 = 4\hat{n}_z^2(1 - \hat{n}_z^2)$$

или

$$\hat{R}_y = 2\hat{n}_z\hat{n}_y \tag{5.10}$$

Из соотношения (5.8) имеем

$$\hat{R}_x = 2\hat{n}_z\hat{n}_x \tag{5.11}$$

Когда свет падает не по оси $z$ (например, когда источников несколько), таким методом воспользоваться нельзя. Можно перенести и повернуть каждый источник так, чтобы свет падал вдоль оси $z$; однако проще перенести и повернуть нормаль так, чтобы она была параллельна оси $z$, а точку $P$ поверхности принять за начало координат. Тогда плоскость $xy$ будет касательной к поверхности, а $x$- и $y$-составляющие единичных векторов падения и отражения будут иметь разные знаки; $z$-составляющие этих векторов, будут, конечно, равны. Для того чтобы получить результаты в первоначальной ориентации, надо выполнить обратное преобразование. В перемещенной и повернутой системе координат

$$\hat{R}_x = -\hat{L}_x \qquad \hat{R}_y = -\hat{L}_y \qquad \hat{R}_z = \hat{L}_z$$

Этот метод особенно удобен, когда преобразования реализованы аппаратно или микропрограммно.

В третьем методе условие того, что единичная нормаль, единичный вектор падения и единичный вектор отражения лежат в одной плоскости, записывается при помощи их векторных произведений. Равенство углов падения и отражения выражается через скалярные произведения этих векторов. Из данных условий получаем

$$\mathbf{n} \otimes \mathbf{L} = \mathbf{R} \otimes \mathbf{n}$$

или

$$(n_yL_z - n_zL_y)\mathbf{i} + (n_zL_x - L_zn_x)\mathbf{j} + (n_xL_y - L_xn_y)\mathbf{k} =$$

$$(n_zR_y - n_yR_z)\mathbf{i} + (n_xR_z - n_zR_x)\mathbf{j} + (n_yR_x - n_xR_y)\mathbf{k}$$

Векторные произведения совпадают, если равны их $x$-, $y$- и $z$-составляющие:

$$\begin{aligned} -n_zR_y + n_yR_z &= n_zL_y - n_yL_z \\ n_zR_x - n_xR_z &= n_xL_z - n_zL_x \\ -n_yR_x + n_xR_y &= n_yL_x - n_xL_y \end{aligned} \tag{5.12}$$

На первый взгляд может показаться, что вектор отражения найден. К сожалению, одно из этих уравнений всегда является линейно зависимым.

Запишем равенство углов падения и отражения:

$$\mathbf{n} \cdot \mathbf{L} = \mathbf{n} \cdot \mathbf{R}$$

или

$$n_x R_x + n_y R_y + n_z R_z = n_x L_x + n_y L_y + n_z L_z \qquad (5.13)$$

что дает необходимое добавочное условие. Матричный вид для всех уравнений с тремя неизвестными $R_x$, $R_y$ и $R_z$ таков:

$$\begin{bmatrix} 0 & -n_z & n_y \\ n_z & 0 & -n_x \\ -n_y & n_x & 0 \\ n_x & n_y & n_z \end{bmatrix} \begin{bmatrix} R_x \\ R_y \\ R_z \end{bmatrix} = \begin{bmatrix} n_z L_y - n_y L_z \\ n_x L_z - n_z L_x \\ n_y L_x - n_x L_y \\ n_x L_x + n_y L_y + n_z L_z \end{bmatrix}$$

или

$$[N][R] = [B]$$

Поскольку матрица $[N]$ не является квадратной, для решения системы нужны особые методы[1]. В частности,

$$[R] = \left[[N]^{\mathrm{T}}[N]\right]^{-1}[N]^{\mathrm{T}}[B]$$

## 5.5. ЗАКРАСКА МЕТОДОМ ГУРО

Если при построении полигональной поверхности для каждой грани используется по одной нормали, то модель освещения создает изображение, состоящее из отдельных многоугольников (рис. 5.10, а). Методом Гуро [5-3] можно получить сглаженное изображение. Для того чтобы изобразить объект методом построчного сканирования, нужно в соответствии с моделью освещения рассчитать интенсивность каждого пиксела вдоль сканирующей строки. Нормали к поверхности аппроксимируются в вершинах многоугольников так, как описано в предыдущем разделе. Однако сканирующая строка не обязательно проходит через вершины многоугольника (рис. 5.11). При закраске Гуро сначала определяется интенсивность вершин многоугольника, а затем с помощью билинейной интерполяции вычисляется интенсивность каждого пиксела на сканирующей строке.

---

[1] Обычно этим методом получают усредненное решение, но, так как одно из уравнений (5.12) является лишним, решение будет точным.

a b

Рис. 5.10. Однотонная закраска многоугольников и закраска Гуро. (С разрешения университета штата Юта.)

Рассмотрим, например, участок полигональной поверхности на рис. 5.11. Значение интенсивности в точке $P$ определяется линейной интерполяцией интенсивности в точках $Q$ и $R$. Для получения интенсивности в точке $Q$ — пересечении ребра многоугольника со сканирующей строкой — нужно линейной интерполяцией интенсивностей $A$ и $B$ найти

$$I_Q = uI_A + (1 - u)I_B \qquad 0 \leq u \leq 1$$

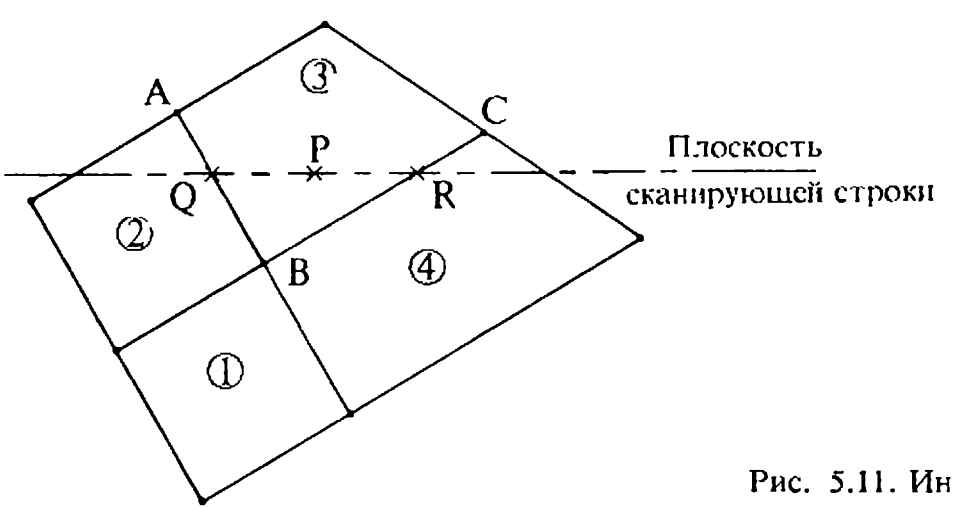



Рис. 5.11. Интерполяция закраски.

где $u = AQ/AB$. Аналогично для получения интенсивности $R$ линейно интерполируются интенсивности в вершинах $B$ и $C$, т. е.

$$I_R = wI_B + (1 - w)I_C \qquad 0 \le w \le 1$$

где $w = BR/BC$. Наконец, линейной интерполяцией по строке между $Q$ и $R$ находится интенсивность $P$, т. е.

$$I_P = tI_Q + (1 - t)I_R \qquad 0 \le t \le 1$$

где $t = QP/QR$.

Значения интенсивности вдоль сканирующей строки можно вычислять инкрементально. Для двух пикселов в $t_1$ и $t_2$ на сканирующей строке

$$I_{P_2} = t_2 I_Q + (1 - t_2)I_R$$

$$I_{P_1} = t_1 I_Q + (1 - t_1)I_R$$

Вычитая, получим, что вдоль строки

$$I_{P_2} = I_{P_1} + (I_Q - I_R)(t_2 - t_1) = I_{P_1} + \Delta I\,\Delta t$$

На рис. 5.10, b показан результат применения закраски Гуро к полигональной аппроксимации лица, изображенного на рис. 5.10, a. Видно, что качество изображения намного улучшилось, но при внимательном рассмотрении на рис. 5.10, b заметно проявление эффекта полос Маха, например на скулах, вокруг глаз и на подбородке. Это происходит потому, что такой метод интерполяции обеспечивает лишь непрерывность значений интенсивности вдоль границ многоугольников, но не обеспечивает непрерывности изменения интенсивности. Обратите внимание также на то, что контуры лица, например глаз и носа, — многоугольники.

Еще одна проблема метода Гуро иллюстрируется на рис. 5.12, a. Если нормали к вершинам $B$, $C$, $D$ вычислить усреднением нормалей к многоугольникам, то они будут одинаково ориентированы, т. е. интенсивность в этих точках будет равной. При линейной интерполяции от $B$ до $D$ значение интенсивности получится постоянным, и поверхность на данном участке будет выглядеть плоской. Для изображения плавного перехода в $B$, $C$ и $D$ необходимы дополнительные многоугольники (рис. 5.12, b). Если же нужно сохранить резкие складки, то для предотвращения сглаживания требуется выборочное усреднение нормалей к поверхности. В примере, показанном на рис. 5.12, c, $\mathbf{n}_{B_1}$ вычисляется только для одной грани справа от $B$, аналогично получается $\mathbf{n}_{D_1}$ и $\mathbf{n}_{D_2}$. В то же время $\mathbf{n}_C$ вычисляется как среднее для граней слева и справа от $C$. В этом

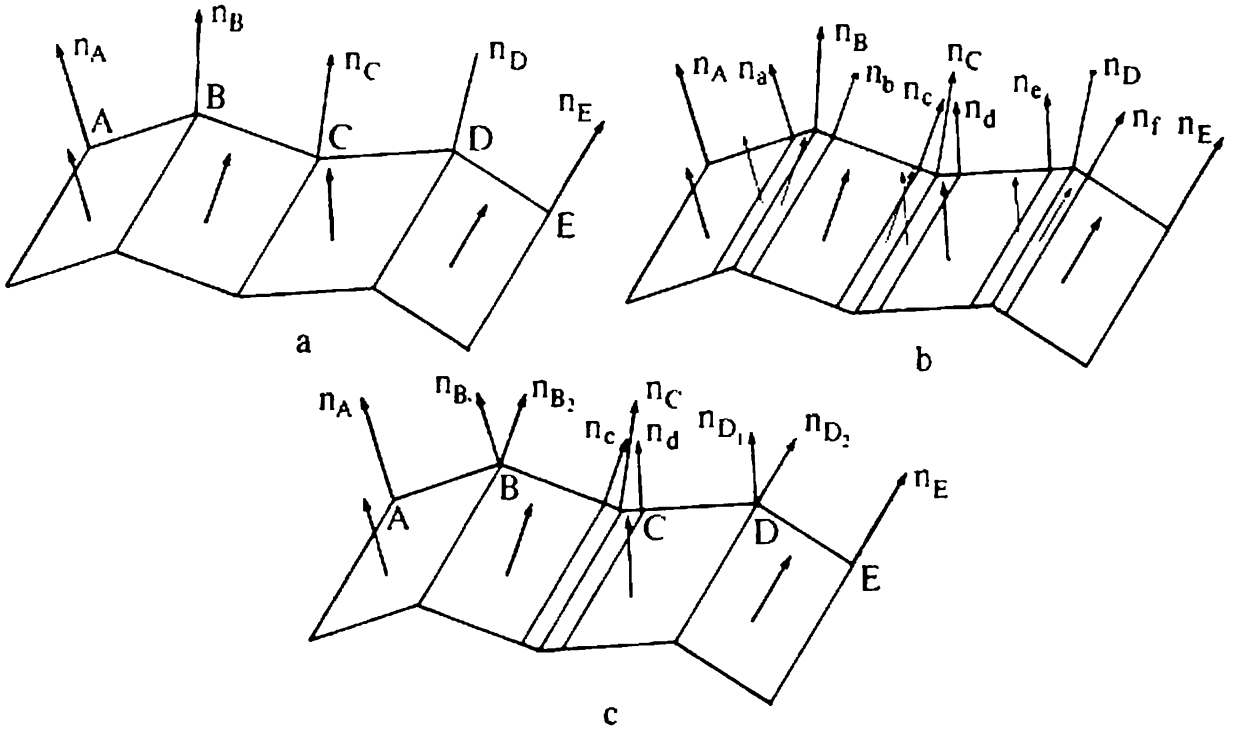



Рис. 5.12. Эффекты закраски Гуро.

случае в $B$ и $D$ получастся острое ребро, а в $C$ — плавный переход, как видно, например, на губах на рис. 5.10, b.

Закраска Гуро лучше всего выглядит в сочетании с простой моделью с диффузным отражением, описываемой уравнениями освещения (5.1) или (5.2), так как форма бликов при зеркальном отражении сильно зависит от выбора многоугольников, представляющих объект или поверхность.

## 5.6. ЗАКРАСКА ФОНГА

Закраска Фонга [5-2] требует больших вычислительных затрат, однако она позволяет разрешить многие проблемы метода Гуро. При закраске Гуро вдоль сканирующей строки интерполируется значение интенсивности, а при закраске Фонга — вектор нормали. Затем он используется в модели освещения для вычисления интенсивности пиксела. При этом достигается лучшая локальная аппроксимация кривизны поверхности и, следовательно, получается более реалистичное изображение. В частности, правдоподобнее выглядят зеркальные блики.

При закраске Фонга аппроксимация кривизны поверхности производится сначала в вершинах многоугольников путем аппроксимации нормали в вершине (см. разд. 5.3). После этого билинейной интерполяцией вычисляется нормаль в каждом пикселе. Например,

снова обращаясь к рис. 5.11, получаем нормаль в $Q$ линейной интерполяцией между $A$ и $B$, в $R$ — между $B$ и $C$ и, наконец, в $P$ — между $Q$ и $R$. Таким образом:

$$\begin{aligned} \mathbf{n}_Q &= u\mathbf{n}_A + (1-u)\mathbf{n}_B \qquad 0 \le u \le 1 \\ \mathbf{n}_R &= w\mathbf{n}_B + (1-w)\mathbf{n}_C \qquad 0 \le w \le 1 \\ \mathbf{n}_P &= t\mathbf{n}_Q + (1-t)\mathbf{n}_R \qquad 0 \le t \le 1 \end{aligned}$$

где

$$u = AQ/AB,\ w = BR/BC,\quad t = QP/QR.$$

Нормаль вдоль сканирующей строки опять можно выразить через приращение, т. е.

$$\mathbf{n}_{P_2} = \mathbf{n}_{P_1} + (\mathbf{n}_Q - \mathbf{n}_R)(t_2 - t_1) = \mathbf{n}_{P_1} + \Delta\mathbf{n}^*\Delta t$$

где индексы 1 и 2 указывают на расположение пискелов на строке.

На рис. 5.13 сравниваются однотонная закраска (слева), закраска Гуро (в центре), Фонга (справа). Модель освещения для левого и среднего торов включает рассеянный свет и диффузное отражение (уравнение (5.1)), а для правого — также зеркальное отражение, благодаря которому появляются блики (уравнение (5.5) с $d = 0$, $K = 1$). На рис. 5.14 сравниваются зеркальные блики при закраске Гуро и более реалистичной закраске Фонга.

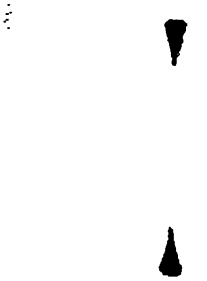


Рис. 5.13. Сравнение методов закраски: слева — однотонная, в центре — Гуро, справа — Фонга. (С разрешения Т. Уиттеда.)

a b

Рис. 5.14. Сравнение зеркальных бликов: (a) закраска Гуро, (b) Фонга. (С разрешения университета штата Юта.)

Хотя метод Фонга устраняет большинство недостатков метода Гуро, он тоже основывается на линейной интерполяции. Поэтому в местах разрыва первой производной интенсивности заметен эффект полос Маха, хотя и не такой сильный, как при закраске Гуро. Однако, как показал Дафф [5-4], иногда этот эффект проявляется сильнее у метода Фонга, например для сфер. Кроме того, оба метода могут привести к ошибкам при изображении невыпуклых многоугольников, например, таких, как на рис. 5.15. Первая сканирующая строка использует данные из вершин $QRS$, а вторая, лежащая ниже, берет также данные вершины $P$. Это может нарушить непрерывность закраски.

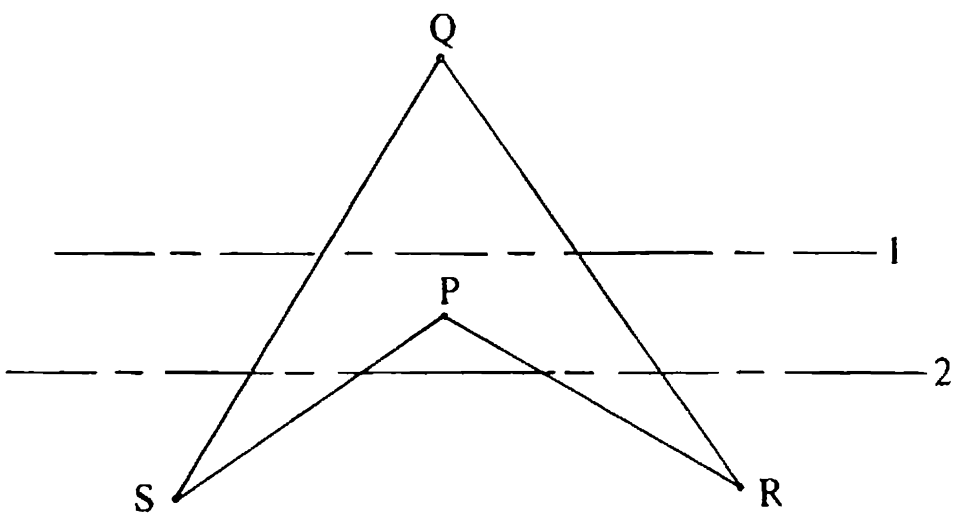



Рис. 5.15. Нарушение непрерывности закраски для невыпуклого многоугольника.

Также возникают трудности, когда любой из этих методов применяется при создании последовательности кинокадров. Например, закраска может значительно изменяться от кадра к кадру. Это происходит из-за того, что правило закраски зависит от поворотов, а обработка ведется в пространстве изображения. Поэтому, когда от кадра к кадру меняется ориентация объекта, его закраска (цвет) тоже изменяется, причем достаточно заметно. Дафф предлагает метод закраски Гуро и Фонга, инвариантный относительно поворота.

Рассмотрим отличия трех методов закраски — однотонной, Гуро и Фонга — на примере.

**Пример 5.3.** Закраска

Рассмотрим участок поверхности, изображенный на рис. 5.11. Уравнения четырех плоскостей:

$$\begin{aligned}
1:&\quad 2z - 4 = 0\\
2:&\quad -x + 1.732y + 7.5z - 17 = 0\\
3:&\quad -2.25x + 3.897y + 10z - 24.5 = 0\\
4:&\quad 5.5z - 11 = 0
\end{aligned}$$

где ось $z$ — перпендикуляр к плоскости страницы, $x$ направлена вправо, а $y$ — вверх. Координаты точек $B$; (0.366, 1.366, 2).

Пусть вектор к точке наблюдения есть $\mathbf{S}(1, 1, 1)$ и единственный точечный источник расположен в бесконечности на положительной полуоси $z$, т. е. вектор падающего света равен $\mathbf{L}(0, 0, 1)$. Модель освещения задана уравнением (5.7), где $d = 0$, $K = 1$, $I_a = 1$, $I_l = 10$, $n = 2$, $k_s = 0.8$, $k_d = k_a = 0.15$. Свет падает вдоль оси $z$, поэтому направление отраженного света можно найти методом Фонга (см. разд. 5.4).

Рассмотрим однотонную закраску. Точка $P$ лежит в многоугольнике 3. Из уравнения плоскости многоугольника 3 находим нормаль

$$\hat{\mathbf{n}}_3 = \frac{\mathbf{n}_3}{|\mathbf{n}_3|} = -0.21\mathbf{i} + 0.36\mathbf{j} + 0.91\mathbf{k}$$

Угол между нормалью и вектором падающего света есть

$$\hat{\mathbf{n}} \cdot \hat{\mathbf{L}} = (-0.21\mathbf{i} + 0.36\mathbf{j} + 0.91\mathbf{k}) \cdot \mathbf{k} = 0.91$$

откуда следует, что угол падения составляет примерно 24.2°.

Из уравнений (5.9) — (5.11) имеем:

$$\begin{aligned}
\hat{R}_z &= 2\hat{n}_z^2 - 1 = (2)(0.91)^2 - 1 = 0.66\\
\hat{R}_x &= 2\hat{n}_z\hat{n}_x = (2)(0.91)(-0.21) = -0.38\\
\hat{R}_y &= 2\hat{n}_z\hat{n}_y = (2)(0.91)(0.36) = 0.66
\end{aligned}$$

$$\hat{\mathbf{R}} = -0.38\mathbf{i} + 0.66\mathbf{j} + 0.66\mathbf{k}$$

Единичный вектор наблюдения есть

$$\hat{\mathbf{S}} = \frac{\mathbf{S}}{|\mathbf{S}|} = 0.58\mathbf{i} + 0.58\mathbf{j} + 0.58\mathbf{k}$$

С помощью этого значения найдем угол между вектором отражения и вектором наблюдения:

$$\hat{\mathbf{R}} \cdot \hat{\mathbf{S}} = (-0.38\mathbf{i} + 0.66\mathbf{j} + 0.66\mathbf{k}) \cdot (0.58\mathbf{i} + 0.58\mathbf{j} + 0.58\mathbf{k}) = 0.55$$

что приблизительно дает 57°.

Модель освещения (5.7) для точки $P$ дает

$$\begin{aligned} I_P &= I_a k_a + \frac{I_l}{d+K}[k_d(\hat{\mathbf{n}} \cdot \hat{\mathbf{L}}) + k_s(\hat{\mathbf{R}} \cdot \hat{\mathbf{S}})^n] \\ &= (1)(0.15) + (10/1)[(0.15)(0.91) + (0.8)(0.55)^2] \\ &= 0.15 + 10(0.14 + 0.24) = 0.15 + 3.8 = 3.95 \end{aligned}$$

Для закраски Гуро необходимы векторы нормали в $A$, $B$ и $C$ на рис. 5.11. Аппроксимируя их усреднением нормалей к окружающим плоскостям, получаем

$$\begin{aligned} \mathbf{n}_A &= \mathbf{n}_2 + \mathbf{n}_3 = -3.25\mathbf{i} + 5.63\mathbf{j} + 17.5\mathbf{k} \\ \mathbf{n}_B &= \mathbf{n}_1 + \mathbf{n}_2 + \mathbf{n}_3 + \mathbf{n}_4 = -3.25\mathbf{i} + 5.63\mathbf{j} + 25\mathbf{k} \\ \mathbf{n}_C &= \mathbf{n}_3 + \mathbf{n}_4 = -2.25\mathbf{i} + 3.897\mathbf{j} + 15.5\mathbf{k} \end{aligned}$$

где $\mathbf{n}_1$, $\mathbf{n}_2$, $\mathbf{n}_3$, $\mathbf{n}_4$ выведены из уравнений соответствующих плоскостей. Единичные нормали суть

$$\begin{aligned} \hat{\mathbf{n}}_A &= \frac{\mathbf{n}_A}{|\mathbf{n}_A|} = -0.17\mathbf{i} + 0.3\mathbf{j} + 0.94\mathbf{k} \\ \hat{\mathbf{n}}_B &= \frac{\mathbf{n}_B}{|\mathbf{n}_B|} = -0.12\mathbf{i} + 0.22\mathbf{j} + 0.97\mathbf{k} \\ \hat{\mathbf{n}}_C &= \frac{\mathbf{n}_C}{|\mathbf{n}_C|} = -0.14\mathbf{i} + 0.24\mathbf{j} + 0.96\mathbf{k} \end{aligned}$$

Единичные векторы отражения:

$$\begin{aligned} \hat{\mathbf{R}}_A &= -0.33\mathbf{i} + 0.57\mathbf{j} + 0.76\mathbf{k} \\ \hat{\mathbf{R}}_B &= -0.24\mathbf{i} + 0.42\mathbf{j} + 0.87\mathbf{k} \\ \hat{\mathbf{R}}_C &= -0.27\mathbf{i} + 0.46\mathbf{j} + 0.84\mathbf{k} \end{aligned}$$

Интенсивности в $A$, $B$ и $C$ равны

$$\begin{aligned} I_A &= 0.15 + 10(0.14 + 0.27) = 4.25 \\ I_B &= 0.15 + 10(0.15 + 0.30) = 4.65 \\ I_C &= 0.15 + 10(0.14 + 0.29) = 4.45 \end{aligned}$$

На заданной сканирующей строке имеем $u = IQ\ AB = 0.4$ и $w = BR\ BC = 0.7$. Интерполируя, находим интенсивность в $Q$ и $R$:

$$\begin{aligned} I_Q &= uI_A + (1-u)I_B = (0.4)(4.25) + (1-0.4)(4.65) = 4.49 \\ I_R &= wI_B + (1-w)I_C = (0.7)(4.65) + (1-0.7)(4.45) = 4.59 \end{aligned}$$

Точка $P$ на сканирующей строке расположена в $t = QP/QR = 0.5$. Интерполируя, определим интенсивность в $P$:

$$I_P = tI_Q + (1 - t)I_R = (0.5)(4.49) + (1 - 0.5)(4.59) = 4.54$$

При закраске Фонга нормаль в $P$ получается путем интерполяции нормалей в $A$, $B$, $C$. Затем эта нормаль используется для вычисления интенсивности $P$. Единичные нормали в $Q$ и $R$ суть

$$\begin{aligned}\hat{\mathbf{n}}_Q &= u\hat{\mathbf{n}}_A + (1 - u)\hat{\mathbf{n}}_B = (0.4)[-0.17 \quad 0.3 \quad 0.94] + (0.6)[-0.12 \quad 0.22 \quad 0.97] \\ &= [-0.14 \quad 0.25 \quad 0.96] = -0.14\mathbf{i} + 0.25\mathbf{j} + 0.96\mathbf{k} \\ \hat{\mathbf{n}}_R &= w\hat{\mathbf{n}}_B + (1 - w)\hat{\mathbf{n}}_C = (0.7)[-0.12 \quad 0.22 \quad 0.97] + (0.3)[-0.14 \quad 0.24 \quad 0.96] \\ &= [-0.13 \quad 0.23 \quad 0.97] = -0.13\mathbf{i} + 0.23\mathbf{j} + 0.97\mathbf{k}\end{aligned}$$

Интерполируя нормаль вдоль сканирующей строки, находим

$$\begin{aligned}\hat{\mathbf{n}}_P &= t\hat{\mathbf{n}}_Q + (1 - t)\hat{\mathbf{n}}_R = (0.5)[-0.14 \quad 0.25 \quad 0.96] + 0.5\,[-0.13 \quad 0.23 \quad 0.97] \\ &= [-0.14 \quad 0.24 \quad 0.97] = -0.14\mathbf{i} + 0.24\mathbf{j} + 0.97\mathbf{k}\end{aligned}$$

Тогда единичный вектор отражения в $P$ есть

$$\hat{\mathbf{R}}_P = -0.27\mathbf{i} + 0.46\mathbf{j} + 0.87\mathbf{k}$$

Интенсивность в $P$ равна

$$I_P = 0.15 + (10)(0.15 + 0.30) = 4.65$$

Сравним различные методы закраски:

| | |
|---|---|
| однотонная: | $I_P = 3.93$ |
| Гуро: | $I_P = 4.54$ |
| Фонга: | $I_P = 4.65$ |

## 5.7. ПРОСТАЯ МОДЕЛЬ ОСВЕЩЕНИЯ СО СПЕЦИАЛЬНЫМИ ЭФФЕКТАМИ

На основе простой модели освещения с точечным источником из разд. 5.2 Варн [5-5] разработал модель освещения, включающую специальные эффекты, которые применяются при управлении светом в профессиональных фотостудиях. К ним относятся задание направления и концентрации света, а также и возможность ограничить область, освещаемую источником света.

В нормали Варна направление света можно регулировать независимо от расположения источника (рис. 5.16, a). Теоретически направленный свет можно представить как отражение от точечной идеальной псевдоповерхности при освещении точечным источником (рис. 5.16, b). Если точечный источник находится в направлении $\mathbf{L}$, перпендикулярном отражающей псевдоповерхности, то отражение

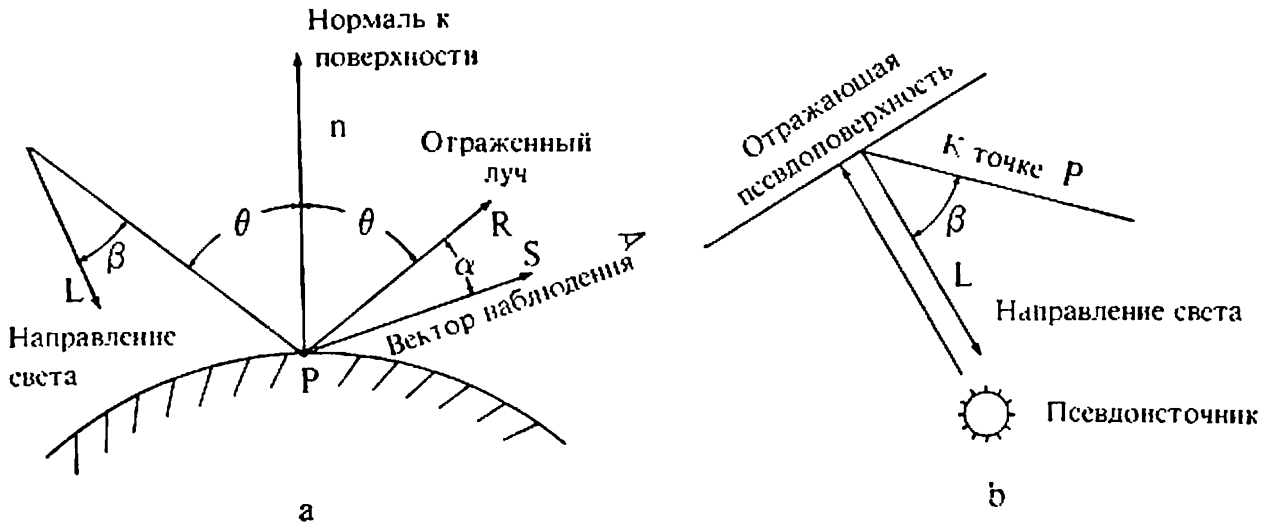



Рис. 5.16. Модель направленного освещения.

исходящего от него света освещает объект в направлении **L**. Направление света регулируется поворотом псевдоповерхности.

При такой концептуальной схеме одна и та же модель освещения годится как для направленных, так и для точечных источников света. Количество света, падающего в точку $P$ от направленного источника, зависит, как показано на рис. 5.16, а, от угла $\beta$ между вектором направления света **L** и прямой, проходящей через источник и $P$. Пользуясь аппроксимацией Фонга для зеркального отражения от идеальной поверхности, находим интенсивность света от направленного источника вдоль прямой, соединяющей источник и точку $P$:

$$I_{l_j}\cos^c\beta,$$

где $c$ — степень, определяющая пространственную концентрацию направленного источника (рис. 5.6). При большом $c$ моделируется узкий луч прожектора, а при малом $c$ — заливающий свет. Теперь интенсивность света от направленного источника в модели освещения (5.6) равна

$$I_j = I_{l_j}\cos^c\beta(k_{d_j}\cos\theta_j + k_{s_j}\cos^{n_j}\alpha_j), \qquad (5.14)$$

где $j$ обозначает конкретный источник света.

На фотостудиях достигают специальных эффектов посредством ограничения освещаемых участков заслонками, которые устанавливаются на источниках (профессиональные фотографы называют их «воротами сарая»), а также с помощью особых отражателей. Варн для создания специальных эффектов применяет заслонки и конусы. Заслонки, ориентированные по координатным плоскостям, моделируются путем ограничения протяженности освещенного участка по осям $x$, $y$ и $z$ (рис. 5.17, а). Если точка объекта лежит внутри огра-

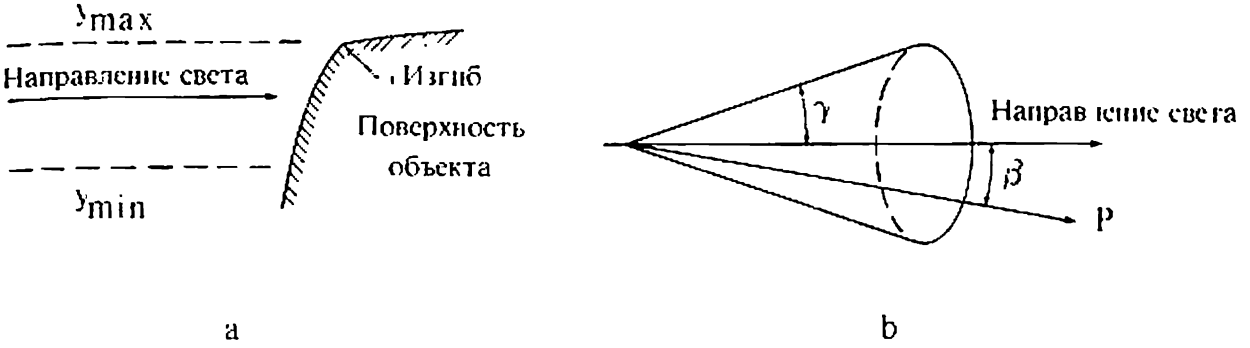



Рис. 5.17. Заслонки и воронки.

ничителей, например, если $y_{min} \leqslant y_{объект} \leqslant y_{max}$, то вычисляется компонента интенсивности, определяемая этим источником, в противном случае она игнорируется. Произвольно ориентированные заслонки реализуются непосредственно. Используя заслонки, можно создавать эффекты, не встречающиеся в природе, например можно закрыть часть сцены от некоторого источника света.

Конус (рис. 5.17, b) позволяет получить четко очерченное пятно света в отличие от направленного источника, при котором при изменении $C$ свет постепенно затухает у края. Этот эффект также заимствован из фотографии и реализуется непосредственно. Пусть вершина конуса совпадает с источником, а $\gamma$ — угол раствора конуса; тогда при $\beta > \gamma$ точка $P$ не освещается этим источником. В противном случае соответствующая компонента интенсивности включается в модель освещения. Обычно для этого сравниваются $\cos\beta$ и $\cos\gamma$, т. е. должно быть $\cos\beta > \cos\gamma$.

Изображение на цветной вклейке 1 построено в соответствии с данной моделью освещения. Шевроле Камаро 1983 освещается пятью источниками: два из них расположены слева от машины, два — справа и один — сверху, позади машины. Обратите внимание на то, как используется концентрация света для того, чтобы подчеркнуть изгиб на правой двери и вдоль правого заднего крыла. Пятый источник освещает задние фары и детали бампера. В результате получается высококачественное изображение.

## 5.8. БОЛЕЕ ПОЛНАЯ МОДЕЛЬ ОСВЕЩЕНИЯ

Модели освещения, рассмотренные в предыдущих разделах, сравнительно просты и основаны на эстетической и экспериментальной аппроксимациях, в особенности зеркальная компонента отражения. Торрэнс и Спэрроу [5-6] разработали теоретически обоснованную модель отражения, которая хорошо соответствует опытным ре-

зультатам. Блинн [5-7] и Кук и Торрэнс [5-8, 5-9] воспользовались этой моделью для создания синтетических изображений. У Блинна зеркальные блики имеют тот же цвет, что и у падающего света. Кук ввел зависимость коэффициента зеркального отражения от длины волны и получил, что цвет, т. е. длина волны зеркальных бликов, зависит от свойств вещества. Когда угол падения приближается к $\pi/2$, цвет бликов приближается к цвету источника.

Для того чтобы дополнить модель освещения, рассмотрим сначала телесный угол, противолежащий источнику. Запишем связь энергии, падающей на единичную площадь отражающей поверхности за единицу времени, с интенсивностью света, падающего на единичную проекцию площади в единичный телесный угол, противолежащий источнику:

$$E_l = I_l(\hat{\mathbf{n}} \cdot \hat{\mathbf{L}})\, d\omega$$

Если поверхность неровная, то диапазон углов отражения может быть очень широким. Зависимость интенсивности отражения от падающей энергии

$$I = rE_l$$

где $r$ — отношение отраженной в данном направлении интенсивности к энергии, падающей в другом направлении. Оно называется коэффициентом двунаправленного отражения. Подставляя одно уравнение в другое, получаем

$$I = rI_l(\hat{\mathbf{n}} \cdot \hat{\mathbf{L}})\, d\omega$$

Коэффициент двунаправленного отражения состоит из двух частей: зеркальной и диффузной, т. е.

$$r = k_d r_d + k_s r_s, \text{ где } k_d + k_s = 1$$

$k_d$ и $k_s$ — характеристики вещества, которые обычно неизвестны и поэтому рассматриваются как произвольные параметры.

Для завершения модели необходимо учесть отражение рассеянного света. Охватывающая полусфера, которую можно рассматривать как источник рассеянного света, может частично закрываться объектами сцены. При этом интенсивность отражения рассеянного света

$$I = fk_a r_a I_a$$

где $f$ — незакрытая часть полусферы. Интегрируя коэффициент двунаправленного отражения $r$ по всей полусфере, получим коэффициент отражения рассеянного света $r_a$, т. е. $r_a$ — линейная комбина-

ция $r_d$ и $r_s$. Константа $k_a$ опять зависит от свойств вещества, но, как правило, рассматривается как произвольный параметр.

Объединяя все результаты, получим модель Кука — Торрэнса для $m$ источников

$$I = fk_a r_a I_a + \sum_{j=1}^{m} I_{lj}(\hat{\mathbf{n}} \cdot \hat{\mathbf{L}})\, d\omega\, (k_d r_d + k_s r_s) \qquad (5.15)$$

В отличие от предыдущих моделей модель Кука — Торрэнса учитывает множество источников с разными интенсивностями ($I_l$) и разными площадями проекций ($\hat{\mathbf{n}} \cdot \hat{\mathbf{L}} d\omega$), что очень важно. Например, если есть два источника с одинаковой интенсивностью и углом освещения объекта, но телесный угол одного в два раза больше, то для него интенсивность отражения будет вдвое больше и объект будет выглядеть в два раза ярче. Для больших, но удаленных источников света телесный угол может быть очень маленьким — например, для Солнца он составляет 0.000068 стерадиан.

Компоненты модели зависят от длины волны падающего света, свойств вещества, шероховатости поверхности и геометрии отражения. Особенно интересно построение зеркальных бликов, так как они сильно влияют на реалистичность синтетического изображения. Этот вопрос рассматривается в работе Торрэнса — Спэрроу.

Модель отражения от шероховатой поверхности Торрэнса — Спэрроу [5-6] основана на принципах геомстрической оптики. Она предназначена для поверхностей, у которых средний размер неровностей намного превышает длину волны падающего света. Предполагается, что поверхность состоит из случайно ориентированных микроскопических зеркальных граней. Зеркальная составляющая отраженного света $r_s$ складывается из отражений от отдельных микрограней. Диффузная составляющая отражения $r_d$ является результатом множества отражений от микрограней и внутреннего рассеивания. На рис. 5.18 показана геометрия отражения от шероховатой поверхности. Здесь $\hat{\mathbf{n}}$ — единичная нормаль к поверхности, $\hat{\mathbf{L}}$ — единичный вектор в направлении к источнику, $\hat{\mathbf{R}}$ — единичный вектор отражения, $\hat{\mathbf{H}}$ — единичная нормаль к отдельной микрограни, $\hat{\mathbf{S}}$ — единичный вектор отражения от микрограни, а также направления к наблюдателю. По законам отражения $\hat{\mathbf{L}}$, $\hat{\mathbf{H}}$ и $\hat{\mathbf{S}}$ лежат в одной плоскости и углы падения и отражения $\phi$ равны. Угол между нормалью к поверхности $\hat{\mathbf{n}}$ и нормалью к микрограни $\hat{\mathbf{H}}$ обозначен $\delta$. Поскольку $\hat{\mathbf{H}}$ — биссектриса угла между $\hat{\mathbf{L}}$ и $\hat{\mathbf{S}}$, имеем

$$\hat{\mathbf{H}} = \frac{\hat{\mathbf{S}} + \hat{\mathbf{L}}}{|\hat{\mathbf{S}}| + |\hat{\mathbf{L}}|} = \frac{\hat{\mathbf{S}} + \hat{\mathbf{L}}}{2}$$

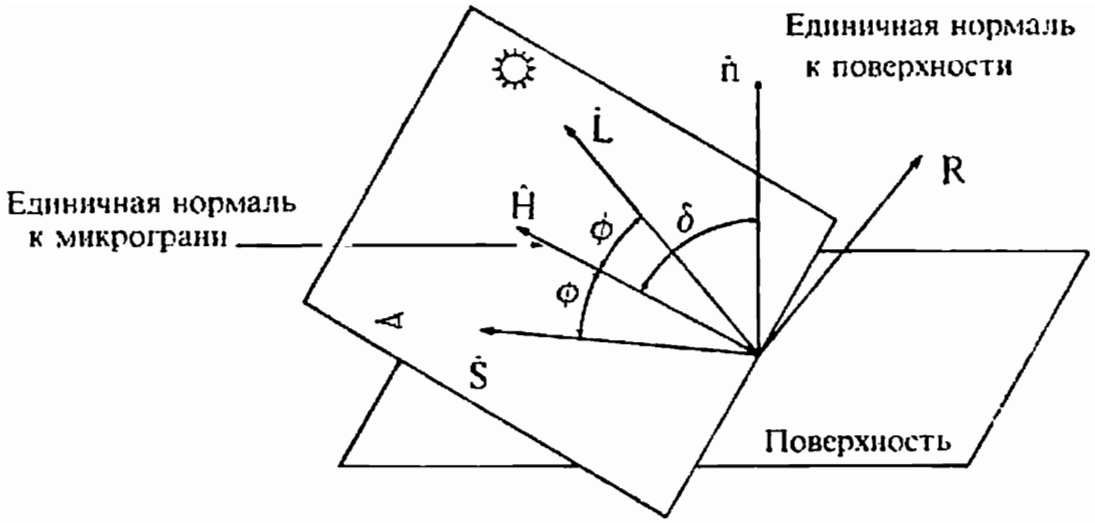



Рис. 5.18. Геометрия модели отражения Торрэнса—Спэрроу

и

$$\cos\phi = \hat{\mathbf{L}} \cdot \hat{\mathbf{H}} = \hat{\mathbf{S}} \cdot \hat{\mathbf{H}}$$

Зеркальное отражение в направлении наблюдателя $\hat{\mathbf{S}}$ формируется только теми микрогранями, нормали которых имеют составляющие в направлении $\mathbf{H}$.

Применяя медель Торрэнса — Спэрроу, Кук и Торрэнс записывают коэффициент зеркального отражения $r_s$ следующим образом:

$$r_s = \frac{F}{\pi} \frac{DG}{(\hat{\mathbf{n}} \cdot \hat{\mathbf{L}})(\hat{\mathbf{n}} \cdot \hat{\mathbf{S}})}$$

где $F$ — член Френеля, $D$ — функция распрелеления микрограней на поверхности, $G$ — коэффициент ослабления света, обусловленного тем, что грани маскируют и затеняют друг друга.

Если каждую микрограпь рассматривать как одну сторону V-образной впадины (рис. 5.19), то часть микрограни может оказаться в тени (5.19, b), либо часть отраженного света не выйдет за прелелы впадины, так как его путь прегражлает противоположная стенка (рис. 5.19, c). Влияние маскирования и затенения выражается соотношением $\bar{m}/l$. Поэтому ослабление света выражается формулой

$$G = 1 - \bar{m}/l.$$

Из рис. 5.19 видно, что ослабление света — функция угла падения света, угла между сторонами впадины и длины стороны $l$. Если никаких помех нет, то $\bar{m} = 0$ и $G = 1$. Торрэнс и Спэрроу [5-6], а также Блинн [5-7] нашли $G$ для маскирования и затенения. Составляющая маскирования (рис. 5.19, c) есть

$$G_{\bar{m}} = \frac{2(\hat{\mathbf{n}} \cdot \hat{\mathbf{H}})(\hat{\mathbf{n}} \cdot \hat{\mathbf{L}})}{\hat{\mathbf{H}} \cdot \hat{\mathbf{L}}}$$

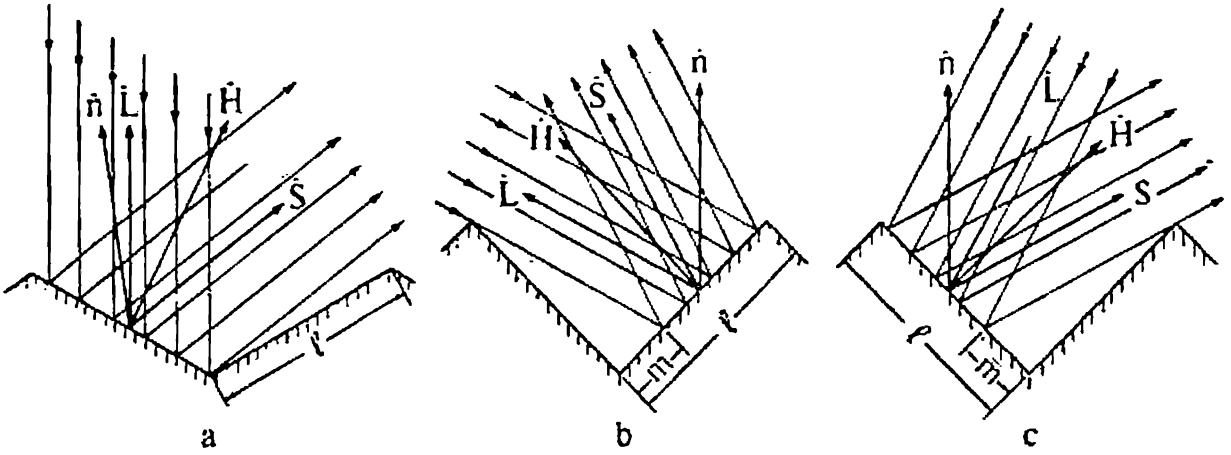



Рис. 5.19. Геометрическое затухание при загораживании и затенении: (a) без помех, (b) затенение, (c) загораживание.

Составляющая затенения (рис. 5.19, b) выражается аналогично, только вместо $\hat{\mathbf{L}}$ стоит $\hat{\mathbf{S}}$, т. е.

$$G_s = \frac{2(\hat{\mathbf{n}}\cdot\hat{\mathbf{H}})(\hat{\mathbf{n}}\cdot\hat{\mathbf{S}})}{\hat{\mathbf{H}}\cdot\hat{\mathbf{S}}} = \frac{2(\hat{\mathbf{n}}\cdot\hat{\mathbf{H}})(\hat{\mathbf{n}}\cdot\hat{\mathbf{S}})}{\hat{\mathbf{H}}\cdot\hat{\mathbf{L}}}$$

так как $\hat{\mathbf{H}}$ — биссектриса угла между $\hat{\mathbf{L}}$ и $\hat{\mathbf{S}}$. Ослабление света определяется минимальной из этих величин:

$$G = \min(1,\ G_m,\ G_s).$$

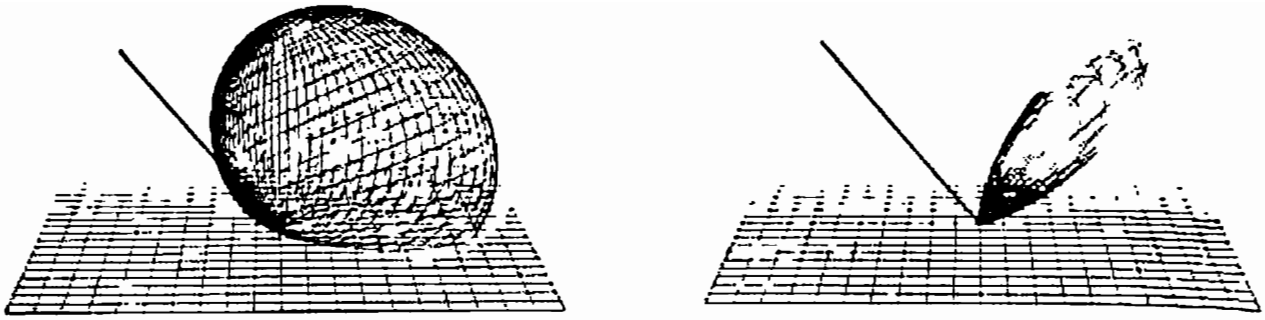
В модели Торрэнса — Сиэрроу предполагается, что микрограни ориентированы согласно распределению Гаусса, т. е.

$$D = c_1 e^{-(\delta/m)^2},$$

где $c_1$ — произвольная константа, $m$ — среднеквадратичный наклон микрограней. Кук и Торрэнс применяют теоретически более обоснованную модель Бекмана [5-10] с распределением

$$D = \frac{1}{m^2\cos^4\delta}\, e^{-(\operatorname{tg}\delta/m)^2},$$

которое выражает абсолютную величину функции распределения без произвольных констант. На рис. 5.20 сравниваются распределения Бекмана при $m = 0.2$ и $m = 0.6$, что соответствует блестящим и матовым поверхностям. Изображенные на рис. 5.20 поверхности представляют величину интенсивности отражения в направлении $\hat{\mathbf{S}}$ от точки $P$ при изменении $\hat{\mathbf{S}}$ в пределах полусферы. Для малых $m$ отраженная интенсивность сконцентрирована около направления зеркального отражения $\hat{\mathbf{R}}$, а для больших $m$ распределена более ровно. При малых $m$ поверхности выглядят блестящими, а при боль-



a b

Рис. 5.20. Функции распределения Бекмана при (a) $m = 0.2$ и (b) $m = 0.6$. (С разрешения Р. Кука, Программа машинной графики, Корнелльский университет.)

ших — тусклыми и матовыми. В первом случае функции распределения Гаусса, Бекмана и Фонга почти одинаковы, а во втором — значительно различаются.

Если на поверхности есть неровности с размерами разного порядка, то можно взять взвешенную линейную комбинацию распределений для разных значений $m$, например:

$$D = \sum_i w_i D(m_i)$$

где сумма весомых коэффициентов $w_i$ равна 1, т. е. $\sum w_i = 1$.

От длины волны $\lambda$ зависят все виды отражения: рассеянное, диффузное и зеркальное. Зависимость $r_a$, $r_l$ и $F$ от длины волны определяется свойствами вещества. Теоретически член Френеля для коэффициента зеркального отражения $r_s$ можно найти из уравнения Френеля, описывающего отражение неполяризованного света от гладкой зеркальной поверхности, т. е.

$$F = \frac{1}{2}\left[\frac{\sin^2(\phi - \theta)}{\sin^2(\phi + \theta)} + \frac{\tan^2(\phi - \theta)}{\tan^2(\phi + \theta)}\right]$$

где $\sin\theta = \sin\phi/\eta$, $\eta$ — показатель преломления вещества, $\theta = \cos^{-1}(\mathbf{L}\cdot\mathbf{H}) = \cos^{-1}(\mathbf{S}\cdot\hat{\mathbf{H}})$ — угол падения. Функция $F$, как и показатель преломления, является функцией длины волны. Если зависимость $\eta$ от $\lambda$ неизвестна, то $F$ можно вывести из экспериментальных данных (см., например, [5-11])[1]. На рис. 5.21, а изображен график $F(\lambda)$ для бронзы при нормальном падении света. Кук и Тор-

[1] Заметим, что в работе [5-11] изменение коэффициентов отражения дано для гладких поверхностей. Если поверхность шероховатая, то нужно домножить на $1/\pi$.

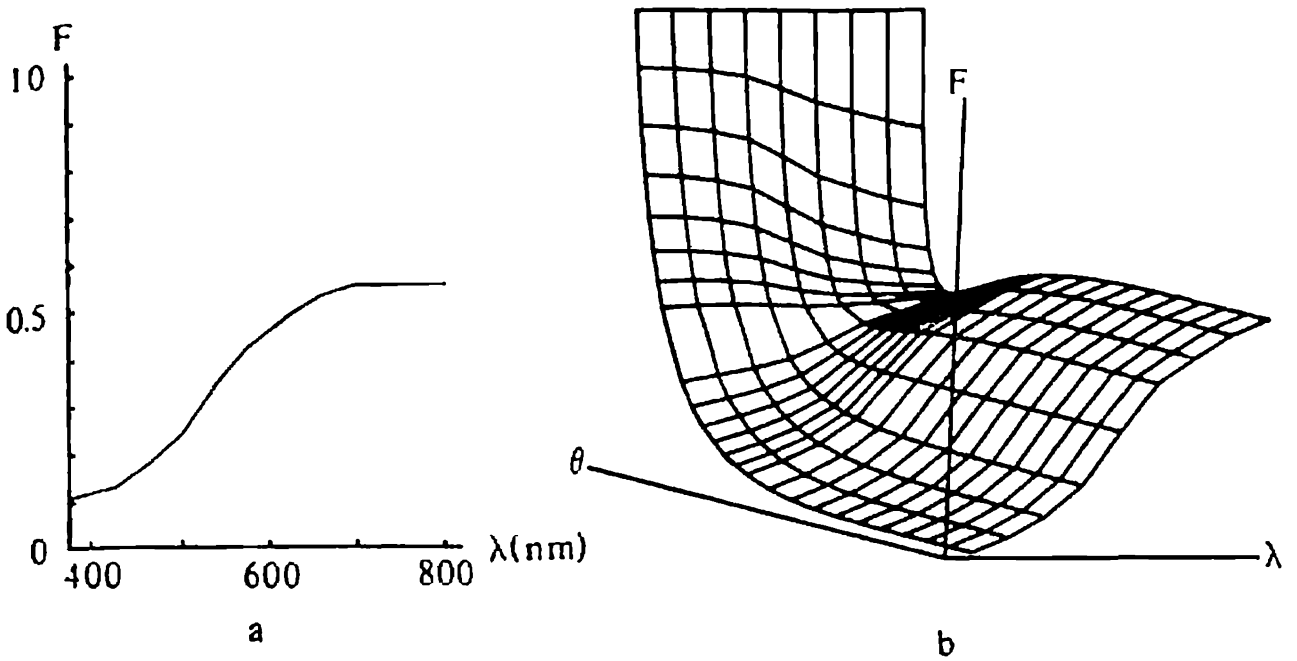



Рис. 5.21. Коэффициент отражения $\varrho$ бронзы (a) при нормальном падении света (b) как функция угла падения, полученная из (a) и уравнения Френеля. (С разрешения Р. Кука, Программа машинной графики, Корнелльский университет.)

рэнс предлагают следующий способ расчета зависимости $F(\lambda)$ от угла, если известна только зависимость при нормальном падении. Перепишем $F$ как

$$F = \frac{1}{2}\frac{(g-c)^2}{(g+c)^2}\left\{1 + \frac{[c(g+c)-1]^2}{[c(g-c)+1]^2}\right\}$$

где

$$c = \cos\phi = \hat{\mathbf{S}}\cdot\hat{\mathbf{H}} = \hat{\mathbf{L}}\cdot\hat{\mathbf{H}}$$

$$g^2 = \eta^2 + c^2 - 1$$

Заметим, что $c = 1$, $g = \eta$ при $\phi = 0$. Пользуясь этим, получим

$$F_0 = \left(\frac{\eta - 1}{\eta + 1}\right)^2$$

Решим это уравнение относительно показателя преломления $\eta$:

$$\eta(\lambda) = \frac{1 + \sqrt{F_0(\lambda)}}{1 - \sqrt{F_0(\lambda)}}$$

Затем с помощью этого значения $\eta$ из уравнения Френеля определяется $F(\lambda)$. Типичный результат приведен на рис. 5.21, b.

Из-за того что коэффициент зеркального отражения зависит от длины волны и угла падения, цвет зеркальных бликов меняется (сдвигается), когда угол падения приближается к $\pi/2$ (рис. 5.21, b).

Если свет падает почти перпендикулярно ($\phi = 0$), то блики окрашиваются в цвет вещества. Если угол падения близок к скользящему ($\phi = \pi/2$), то блики приобретают цвет источника ($F = 1$). Для того чтобы рассчитать цветовой сдвиг, необходимы сложные вычисления, поэтому Кук и Торрэнс предложили воспользоваться линейной интерполяцией между цветом при нормальном отражении ($\phi = 0$) и цветом света ($\phi = \pi/2$). Например, красная компонента (Red) есть

$$\mathrm{Red}_\theta = \mathrm{Red}_0 + (\mathrm{Red}_{\pi/2} - \mathrm{Red}_0)\frac{\max(0, F_\theta - F_0)}{F_{\pi/2} - F_0}$$

Аналогично определяются синяя и зеленая компоненты цветового пространства RGB (разд. 5.15).

Кук и Торрэнс рассматривают коэффициент диффузного отражения, $r_d$, как коэффициент нормального ($\phi = 0$) отражения от поверхности. Хотя он и зависит от угла падения, но при углах меньше 70° этой зависимостью можно пренебречь. Поэтому данное предположение достаточно обоснованно.

С помощью более полной модели освещения были изображены две вазы на цветной вклейке 2. Верхняя ваза сделана из пластмассы бронзового цвета. Это впечатление создается цветной диффузной компонентой и белыми зеркальными бликами ($F = 1$). Нижняя ваза бронзовая. Свет отражается от металлической поверхности и практически не проникает под нее, поэтому диффузного отражения почти нет. Зеркальные блики окрашены в бронзовый цвет. Подробные данные, которыми пользовался Кук при построении этих изображений, приведены в табл. 5.1.

Блинн также применял более сложную модель Торрэнса — Спэрроу с $F = 1$, т. е. без учета цветового сдвига. На рис. 5.22, построенном Блинном, сравнивается форма зеркальных бликов, полученных с использованием модели освещения Фонга при боковом освещении, т. е. когда угол между наблюдателем ($\hat{\mathbf{S}}$) и источником ($\hat{\mathbf{L}}$) составляет около 90°. Если же положения наблюдателя и источника света совпадают, то оба метода дают совершенно одинаковые результаты.

Это объясняется функциями распределения Фонга и Торрэнса — Спэрроу при углах падения света, близких к нормальному (25°) и скользящему (65°) (рис. 5.23). Выпуклость на полусфере соответствует коэффициенту зеркального отражения. Из рис. 5.23, a и b видно, что для падения света, близкого к нормальному, модели почти не отличаются. Однако для углов, близких к скользящему, выпук-

Таблица 5.1.

| | Пластмассовая ваза | Металлическая ваза |
|---|---|---|
| Два источника[1)] | $I_l$-стандартный источник МКО $D_{6500}$<br>$d\omega_i = 0.0001$ и $0.0002$ | $I_l$-стандартный источник МКО $D_{6500}$<br>$d\omega_i = 0.0001$ и $0.0002$ |
| Зеркальное | $k_s = 0.1$<br>$F$ = коэффициент отражения винилового зеркала<br>$D$ = функция Бекмана с $m = 0.15$ | $k_s = 0.1$<br>$F$ = коэффициент отражения бронзового зеркала<br>$D$ = функция Бекмана с<br>$m_1 = 0.4$<br>$w_1 = 0.4$<br>$m_2 = 0.2$<br>$w_2 = 0.6$ |
| Диффузное | $k_d = 0.9$<br>$r_d$ = коэффициент двунаправленного отражения бронзы при нормальном падении света | $k_d = 0.9$<br>$r_d$ = коэффициент двунаправленного отражения бронзы при нормальном падении света |
| Рассеянное | $I_a = 0.01 * I_l$<br>$r_a = \pi r_d$ | $I_a = 0.01 * I_l$<br>$r_a = \pi r_d$ |

[1)] См. разд. 5.15.

a b

Рис. 5.22. Сравнение зеркальных бликов при боковом освещении: (a) Фонг, (b) Торрэнс—Спэрроу, поверхность покрыта окисью магния. (С разрешения университета штата Юта.)

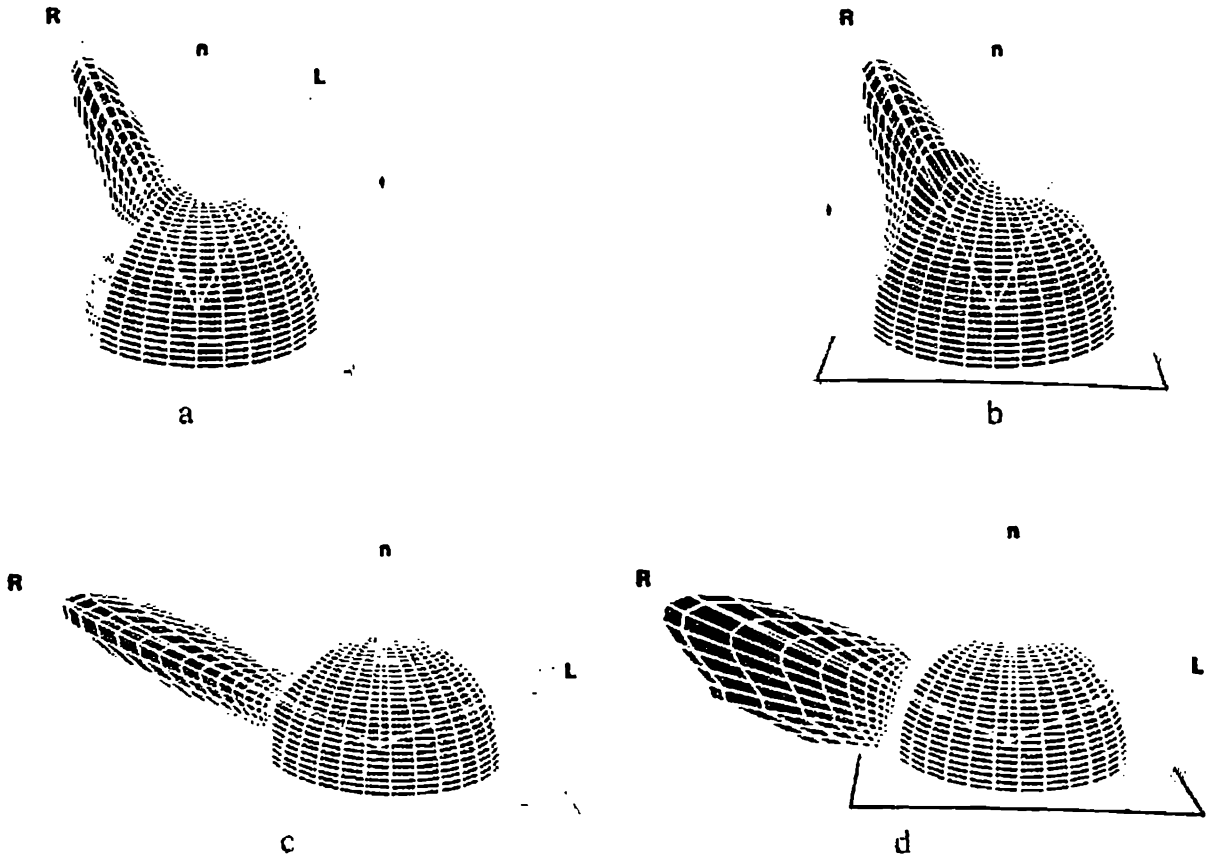



Рис. 5.23. Сравнение функций распределения света при угле падения, близком к нормальному (25°): (a) Фонг, (b) Торрэнс—Спэрроу; и при угле падения, близком к скользящему (65°): (c) Фонг, (d) Торрэнс—Спэрроу.

лость коэффициента зеркального отражения в модели Торрэнса — Спэрроу имеет более узкий вертикально ориентированный профиль, который расположен не совсем в том же направлении, что в модели Фонга. Если в модели Фонга ввести коэффициент ослабления света, то она при боковом освещении будет порождать результаты, аналогичные тем, которые получаются при использовании модели Торрэнса — Спэрроу.

## 5.9. ПРОЗРАЧНОСТЬ

В основных моделях освещения и алгоритмах удаления невидимых линий и поверхностей рассматриваются только непрозрачные поверхности и объекты. Однако существуют и прозрачные объекты, пропускающие свет, например, такие, как стакан, ваза, окно автомобиля, вода. При переходе из одной среды в другую, например из воздуха в воду, световой луч преломляется; поэтому торчащая из воды палка кажется согнутой. Преломление рассчитывается по закону Снеллиуса, который утверждает, что падающий и преломляющий лучи лежат в одной плоскости, а углы падения и преломления

связаны формулой

$$\eta_1 \sin\theta = \eta_2 \sin\theta',$$

где $\eta_1$ и $\eta_2$ — показатели преломления двух сред, $\theta$ — угол падения, $\theta'$ — угол преломления (рис. 5.24). Ни одно вещество не пропускает весь падающий свет, часть его всегда отражается; это также показано на рис. 5.24.

Так же, как и отражение, пропускание может быть зеркальным (направленным) или диффузным. Направленное пропускание свойственно прозрачным веществам, например стеклу. Если смотреть на объект сквозь такое вещество, то, за исключением контурных линий криволинейных поверхностей, искажения происходить не будет. Если свет при пропускании через вещество рассеивается, то мы имеем диффузное пропускание. Такие вещества кажутся полупрозрачными или матовыми. Если смотреть на объект сквозь такое вещество, то он будет выглядеть нечетким или искаженным.

На рис. 5.25 показаны некоторые практические следствия преломления. Показатель преломления объектов 1 и 2 одинаков и больше, чем у окружающей среды. Объекты 3 и 4 непрозрачны. Если не принимать во внимание преломление, то луч $a$ пересечется с объектом 3 (пунктирная линия). Однако из-за преломления он отклоняется и пересекается с объектом 4, т. е. объект 4 можно увидеть, только учитывая эффект преломления. Если же рассматривать луч $b$, то без учета преломления он пересекается с объектом 4, хотя на самом деле он пересекается с объектом 3, т. е. здесь объект, который видим, на самом деле увидеть нельзя. Все это необходимо иметь в виду при создании реалистических изображений.

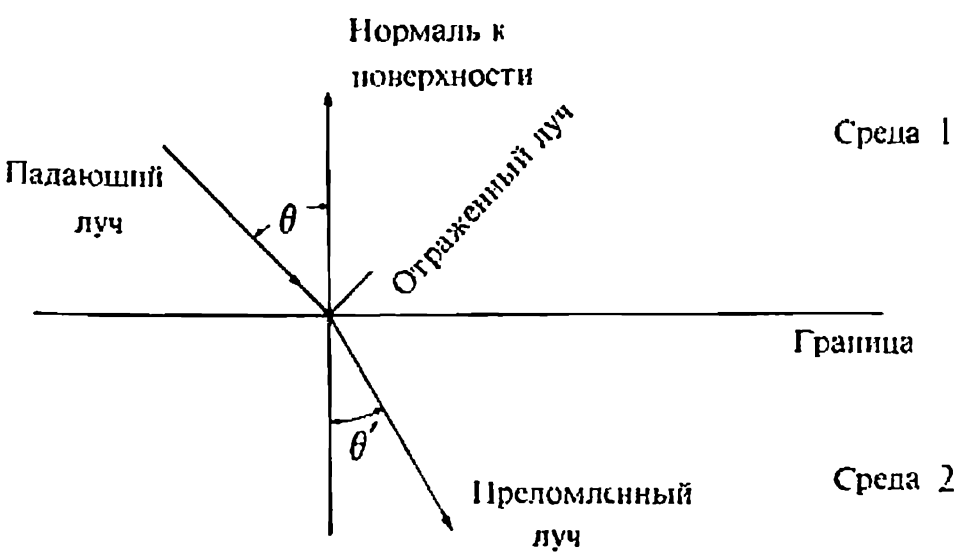



Рис. 5.24. Геометрия преломления.

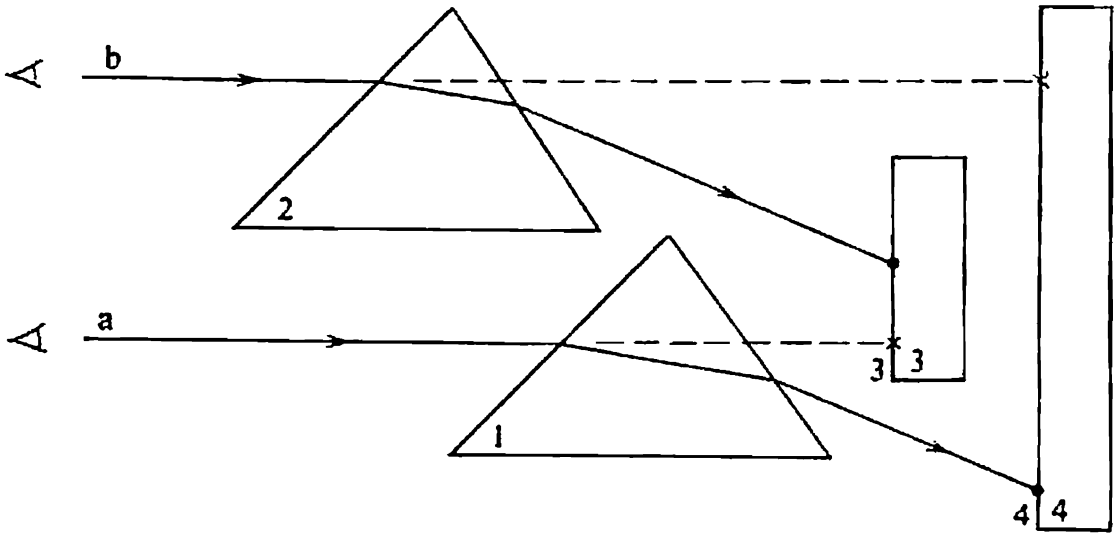



Рис. 5.25. Эффекты преломления.

Нечто похожее происходит при встраивании перспективного преобразования в видовое преобразование. Обычно перспективное преобразование проводится для того, чтобы получить искаженный объект, который затем строится в аксонометрической проекции с точкой наблюдения, удаленной в бесконечность (рис. 5.26). На рис. 5.26, а луч, исходящий из точки $P$, пересекает неискаженный объект в точке $i$ и после преломления попадает в точку $b$ плоскости фона. На рис. 5.26, b показан объект после перспективного преоб-

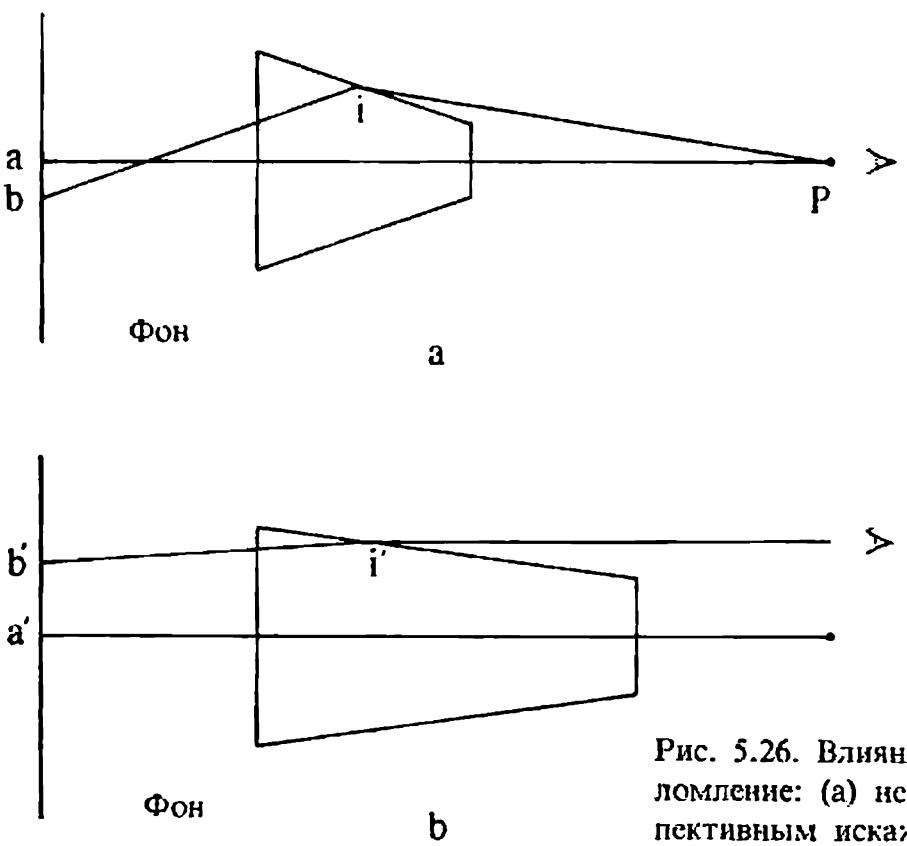



Рис. 5.26. Влияние перспективы на преломление: (a) неискаженное (b) с перспективным искажением.

разования. Теперь луч пересекает объект в преобразованной точке $i$, преломленный луч пересекается с фоном в точке $b'$ с *противоположной* стороны от центральной линии. Это происходит из-за неправильных угловых соотношений между искаженным (преобразованным) объектом и искаженным (преломленным) лучом.

На первый взгляд для получения верного результата достаточно знать истинные угловые соотношения в точках пересечения луча с объектом. Однако это не так, потому что длина пути луча в преобразованном объекте также меняется. Разница в длине пути приводит к тому, что, во-первых, не совпадают точки выхода луча из объекта, так что луч все равно не попадает в правильную точку фона. Во-вторых, меняется количество поглощенного объектом света, поэтому исходящий луч имеет другую интенсивность.

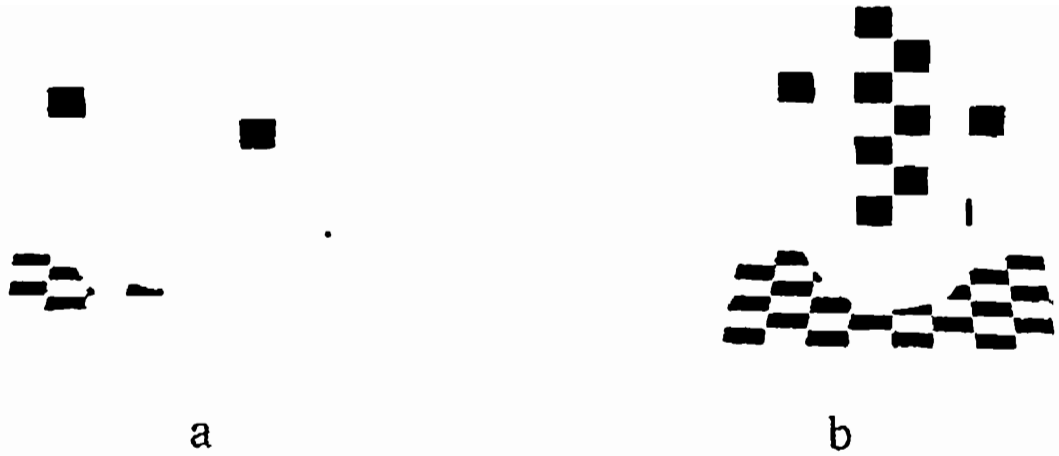
Для того чтобы устранить влияние преломления, можно либо применять алгоритмы, работающие в пространстве объекта, либо пользоваться специальными преобразованиями между пространствами объекта и изображения. Однако проще включить преломление в алгоритмы построения видимых поверхностей методом трассировки лучей, использующие глобальную модель освещения (разд. 5.12).

В простейших реализациях эффекты прозрачности преломления вообще не рассматриваются, и явления, показанные на рис. 5.25 и 5.26, не учитываются. Кроме того, не принимается во внимание, как путь, пройденный лучом в среде, влияет на его интенсивность. Самые ранние разработки в этой области принадлежат Ньюэлу [5-12] (разд. 4.8). Простое пропускание света можно встроить в любой алгоритм удаления невидимых поверхностей, кроме алгоритма с $z$-буфером.

Прозрачные многоугольники или поверхности помечаются, и если видимая грань прозрачна, то в буфер кадра записывается линейная комбинация двух ближайших поверхностей. При этом интенсивность

$$I = tI_1 + (1 - t)I_2 \qquad 0 \leq t \leq 1$$

где $I_1$ — видимая поверхность, $I_2$ — поверхность, расположенная непосредственно за ней, $t$ — коэффициент прозрачности $I_1$. Если поверхность совершенно прозрачна, то $t = 0$, а если непрозрачна, то $t = 1$. Если $I_2$ тоже прозрачна, то алгоритм применяется рекурентно, пока не встретится непрозрачная поверхность или фон. Если многоугольники записываются в буфер кадра в соответствии с приоритетами глубины, как в алгоритме Ньюэла — Ньюэла —



a b

Рис. 5.27. Сравнение простых моделей прозрачности: (a) линейная $t = 0.5$, (b) нелинейная $p = 1$. (С разрешения Д.С. Кэя, Программа машинной графики, Корнелльский университет.)

Санча, тогда $I_2$ будет соответствовать значению, записанному в буфер кадра, а $I_1$ — текущей поверхности.

Для криволинейных поверхностей, например, таких, как ваза или бутылка, линейной аппроксимации недостаточно, так как вблизи контурных линий прозрачность уменьшается из-за толщины материала. Чтобы точнее изобразить это явление, Кэй [5-13, 5-14] предложил несложную нелинейную аппроксимацию на основе $z$-составляющей нормали к поверхности. В частности, коэффициент прозрачности

$$t = t_{\min} + (t_{\max} - t_{\min})[1 - (1 - |n_z|)^p]$$

где $t_{\min}$ и $t_{\max}$ — минимальная и максимальная прозрачность объекта, $n_z$ есть $z$-составляющая единичной нормали к поверхности, $p$ — коэффициент степени прозрачности, $t$ — прозрачность пиксела или точки объекта. На рис. 5.27 сравниваются результаты двух моделей: линейной (рис. 5,27, a) и нелинейной (рис. 5.27, b).

Непосредственно в алгоритм, использующий $z$-буфер, эффект прозрачности ввести нельзя (см. разд. 4.7). Для того чтобы его учесть, необходимы отдельные буферы прозрачности, интенсивности и весовых коэффициентов [5-15], а также нужно отметить прозрачные многоугольники в структуре данных. Для алгоритма, использующего $z$-буфер, последовательность действий такова:

Для каждого многоугольника:
  если многоугольник прозрачен, то внести его в список прозрачных многоугольников;
  если многоугольник непрозрачен и $z > z_{\text{буфер}}$, то записать его

в буфер кадра для непрозрачных многоугольников и скорректировать этот буфер.

Для каждого многоугольника из списка прозрачных многоугольников:

если $z \geqslant z_{\text{буфер}}$, то прибавить его коэффициент прозрачности к значению, содержащемуся в буфере весовых коэффициентов прозрачности;

прибавить его интенсивность к значению, содержащемуся в буфере интенсивности прозрачности, в соответствии с правилом

$$I_{bn} = I_{b0}t_{b0} + I_c t_c$$

где $I_{bn}$ — новое значение интенсивности, $I_{b0}$ — старое значение интенсивности, записанное в буфере интенсивности прозрачности, $I_c$ — интенсивность текущего многоугольника, $t_{b0}$ — старый коэффициент прозрачности из буфера весовых коэффициентов прозрачности, $t_c$ — коэффициент прозрачности текущего многоугольника. Таким образом, получается взвешенная сумма интенсивностей всех прозрачных многоугольников, находящихся перед ближайшим непрозрачным многоугольником.

Объединим буферы интенсивности для прозрачных и непрозрачных многоугольников в соответствии с правилом

$$I_{fb} = t_{b0}I_{b0} + (1 - t_{b0})I_{fb0}$$

где $I_{fb}$ — окончательная интенсивность в буфере кадра для непрозрачных многоугольников, а $I_{fb0}$ — старое значение интенсивности в этом буфере.

Эту процедуру удобнее использовать в сочетании с алгоритмом построчного сканирования с $z$-буфером (см. разд. 4.10), поскольку для полного алгоритма с $z$-буфером требуется очень много памяти.

Одной из интересных прикладных задач, связанных с прозрачностью, является визуализация внутреннего вида сложных объектов или пространств.

Для этого всем многоугольникам поверхности приписываются коэффициенты прозрачности, первоначально равные 1, т. е. они считаются непрозрачными. Можно построить изображение такого объекта с удаленными невидимыми гранями. Затем коэффициенты прозрачности некоторых групп граней заменяются на 0, т. е. они делаются невидимыми. При новом построении изображения сцены получается внутренний вид этого объекта или пространства.

Для того чтобы включить преломление в модель освещения, нужно при построении видимых поверхностей учитывать не только падающий, но и отраженный и пропущенный свет (рис. 5.24). Эффективнее всего это выполняется с помощью глобальной модели освещения в сочетании с алгоритмом трассировки лучей для выделения видимых поверхностей (см. разд. 5.12). Обычно рассматриваются только зеркально отраженные и пропущенные лучи, так как диффузное отражение от просвечивающих поверхностей порождает бесконечное количество беспорядочно ориентированных лучей. Поэтому моделируются только прозрачные вещества, для которых формула расчета интенсивности является простым расширением ранее описанных моделей (см. разд. 5.2, 5.7 и 5.8). Ее общий вид:

$$I = k_a I_a + k_d I_d + k_s I_s + k_t I_t$$

где индексы $a$, $d$, $s$, $t$ обозначают рассеянный, диффузный, зеркальный и пропущенный свет. В большинстве моделей предполагается, что $k_t$ — постоянная и $I_t$ — интенсивность преломленного света определяется по закону Снеллиуса.

## 5.10. ТЕНИ

Если положения наблюдателя и источника света совпадают, то теней не видно, но они появляются, когда наблюдатель перемещается в любую другую точку. Изображение с построенными тенями выглядит гораздо реалистичнее, и, кроме того, тени очень важны для моделирования. Например, особо интересующий нас участок может оказаться невидимым из-за того, что он попадает в тень. В прикладных областях — строительстве, разработке космических аппаратов и др. — тени влияют на расчет падающей солнечной энергии, обогрев и кондиционирование воздуха.

Наблюдения показывают, что тень состоит из двух частей: полутени и полной тени. Полная тень — это центральная, темная, резко очерченная часть, а полутень — окружающая ее более светлая часть. В машинной графике обычно рассматриваются точечные источники, создающие только полную тень. Распределенные источники света конечного размера создают как тень, так и полутень [5-8]: в полной тени свет вообще отсутствует, а полутень освещается частью распределенного источника. Из-за больших вычислительных затрат, как правило, рассматривается только полная тень, образуемая точечным источником света. Сложность и, следовательно, стоимость вычислений зависят и от положения источника. Легче всего,

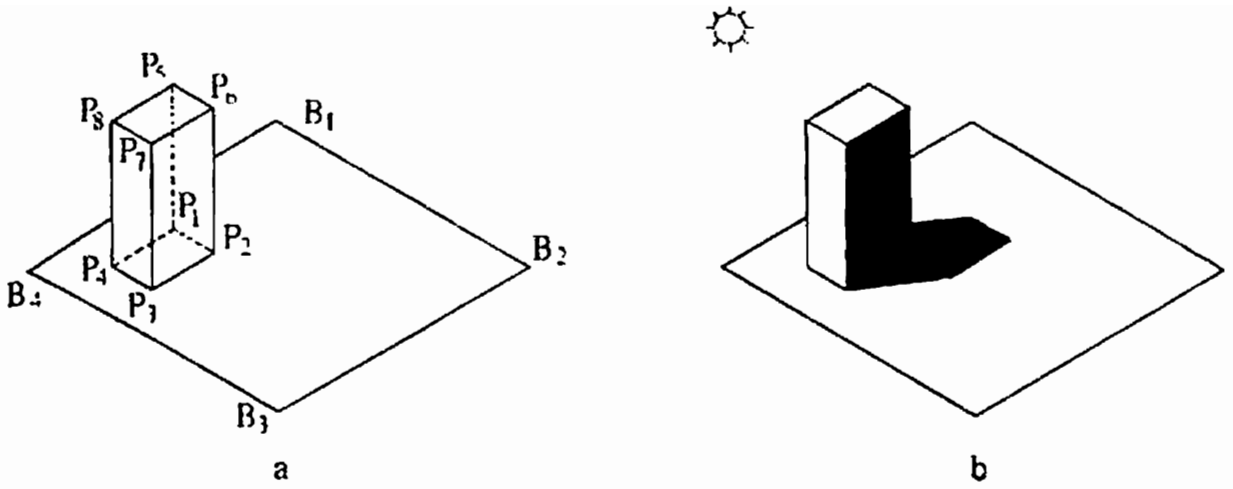



Рис. 5.28. Тени.

когда источник находится в бесконечности, и тени определяются с помощью ортогонального проецирования. Сложнее, если источник расположен на конечном расстоянии, но вне поля зрения; здесь необходима перспективная проекция. Самый трудный случай, когда источник находится в поле зрения. Тогда надо делить пространство на секторы и искать тени отдельно для каждого сектора.

Для того чтобы построить тени, нужно по существу дважды удалить невидимые поверхности: для положения каждого источника и для положения наблюдателя или точки наблюдения, т. е. это двухшаговый процесс. Рассмотрим сцену на рис. 5.28. Один источник находится в бесконечности сверху: спереди слева от параллелепипеда. Точка наблюдения лежит спереди: сверху справа от объекта. В данном случае тени образуются двояко: это собственная тень и проекционная. Собственная тень получается тогда, когда сам объект препятствует попаданию света на некоторые его грани, например на правую грань параллелепипеда. При этом алгоритм построения теней аналогичен алгоритму удаления нелицевых граней: грани, затененные собственной тенью являются нелицевыми, если точку наблюдения совместить с источником света.

Если один объект препятствует попаданию света на другой, то получается проекционная тень, например тень на горизонтальной плоскости на рис. 5.28, b. Чтобы найти такие тени, нужно построить проекции всех нелицевых граней на сцену. Центр проекции находится в источнике света. Точки пересечения проецируемой грани со всеми другими плоскостями образуют многоугольники, которые помечаются как теневые многоугольники и заносятся в структуру данных. Для того чтобы не вносить в нее слишком много много-

угольников, можно проецировать контур каждого объекта, а не отдельные грани.

После добавления теней к структуре данных, как обычно, строится вид сцены из заданной точки наблюдения. Отметим, что для создания разных видов не нужно вычислять тени заново, так как они зависят только от положения источника и не зависят от положения наблюдателя.

Рассмотрим этот метод на примере.

**Пример 5.4. Тени**

Рассмотрим параллелепипед на рис. 5.28, а. Он задан точками $P_1$ (1, 0, 3.5), $P_2$ (2, 0, 3.5), $P_3$ (2, 0, 5), $P_4$ (1, 0, 5), $P_5$ (1, 3, 3.5), $P_6$ (2, 3, 3.5), $P_7$ (2, 3, 5), $P_8$ (1, 3, 5). Параллелепипед стоит на плоскости, заданной точками $B_1$ (0, 0, 0), $B_2$ (6, 0, 0), $B_3$ (6, 0, 6), $B_4$ (0, 0, 6). Источник света расположен в бесконечности на прямой, проходящей через $P_7$ и $P_8$. Точка наблюдения находится в бесконечности на положительной полуоси $z$ после поворота сцены сначала на $-45°$ вокруг оси $y$ и затем на $35°$ вокруг оси $x$.

Для того чтобы найти собственные тени, необходимо определить нелицевые грани относительно положения источника. С помощью формальных методов, рассмотренных в разд. 4.3 и примерах 4.2, 4.6 и 4.7, получим матрицу объема параллелепипеда

$$[V] = \begin{matrix} \\ \\ \\ \\ \end{matrix} \begin{matrix} \text{Ⓡ} & \text{Ⓛ} & \text{Ⓑ} & \text{Ⓣ} & \text{Ⓝ} & \text{Ⓨ} \end{matrix} \begin{bmatrix} -1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 1 \\ 2 & -1 & 0 & 3 & 5 & 3.5 \end{bmatrix}$$

где $R$, $L$, $B$, $T$, $N$, $Y$ обозначают правую, левую, нижнюю, верхнюю, ближнюю и дальнюю грани, если смотреть на преобразованный объект из бесконечности с положительной полуоси $z$. Вектор от источника к объекту в однородных координатах

$$[F] = P_2 - P_8 = [1 \quad -3 \quad -1.5 \quad 0]$$

Скалярное произведение вектора направления к источнику и нелицевых плоскостей дает

$$\begin{matrix} & \text{Ⓡ} & \text{Ⓛ} & \text{Ⓑ} & \text{Ⓣ} & \text{Ⓝ} & \text{Ⓨ} \end{matrix}$$
$$[E] \cdot [V] = [-1 \quad 1 \quad -3 \quad 3 \quad 1.5 \quad -1.5]$$

Знак минус означает, что при наблюдении из положения источника правая, нижняя и дальняя грани являются нелицевыми и, следовательно, находятся в собственной тени.

Существует несколько способов нахождения проекционных теней. Один из них заключается в том, чтобы перенести и повернуть параллелепипед вместе с плоскостью его основания до совмещения вектора направления на источник с осью $z$. Источник находится в бесконечности, поэтому ортогональная проекция видимых граней на преобразованную плоскость основания дает проекционную грань. Значение $z$ получается подстановкой $x$- и $y$-координат вершин преобразованного параллелепипеда в уравнение преобразованной плоскости основания. Затем координаты проекционных теней приводятся к первоначальной ориентации.

Для совмещения вектора падающего из бесконечности вдоль прямой $P_8P_2$ света с осью $x$ необходимо:

перенести $P_2$ в начало координат;
выполнить поворот на 33.69° вокруг оси $y$, чтобы $P_8$ совпала с осью $z$;
выполнить поворот на 59.04° вокруг оси $x$, чтобы $P_8$ совпала с осью $z$.

Объединенное преобразование

$$[T] = \begin{bmatrix} 0.83 & 0.48 & -0.29 & 0 \\ 0 & 0.51 & 0.86 & 0 \\ 0.55 & -0.71 & 0.43 & 0 \\ -3.59 & 1.53 & -0.93 & 1 \end{bmatrix}$$

Преобразуя плоскость основания и параллелепипед, получим

$$\begin{array}{c} B_1 \\ B_2 \\ B_3 \\ B_4 \\ \\ P_1 \\ P_2 \\ P_3 \\ P_4 \\ P_5 \\ P_6 \\ P_7 \\ P_8 \end{array} \begin{bmatrix} 0 & 0 & 0 & 1 \\ 6 & 0 & 0 & 1 \\ 6 & 0 & 6 & 1 \\ 0 & 0 & 6 & 1 \\ \\ 1 & 0 & 3.5 & 1 \\ 2 & 0 & 3.5 & 1 \\ 2 & 0 & 5 & 1 \\ 1 & 0 & 5 & 1 \\ 1 & 3 & 3.5 & 1 \\ 2 & 3 & 3.5 & 1 \\ 2 & 3 & 5 & 1 \\ 1 & 3 & 5 & 1 \end{bmatrix} [T] = \begin{bmatrix} -3.59 & 1.53 & -0.93 & 1 \\ 1.39 & 4.41 & -2.67 & 1 \\ 4.69 & 0.15 & -0.09 & 1 \\ -0.29 & -2.73 & 1.65 & 1 \\ \\ -0.84 & -0.48 & 0.29 & 1 \\ 0 & 0 & 0 & 1 \\ 0.82 & -1.06 & 0.64 & 1 \\ 0 & -1.54 & 0.93 & 1 \\ -0.84 & 1.06 & 2.87 & 1 \\ 0 & 1.54 & 2.58 & 1 \\ 0.82 & 0.47 & 3.22 & 1 \\ 0 & 0 & 3.51 & 1 \end{bmatrix} \begin{array}{l} \\ \\ \\ \text{Плоскость основания} \\ \\ \\ \\ \\ \\ \\ \\ \text{Параллелепипед} \end{array}$$

Уравнение преобразованной плоскости основания найдем методом Ньюэлла (разд. 4.3, пример 4.3):

$$z = -0.6y.$$

Подставляя $x$- и $y$-координаты вершин преобразованного параллелепипеда в уравнение плоскости, определим $z$, что даст проекцию тени на плоскость основания:

$$[P'] = \begin{bmatrix} -0.84 & -0.48 & 0.29 \\ 0 & 0 & 0 \\ 0.82 & -1.06 & 0.64 \\ 0 & -1.54 & 0.93 \\ -0.84 & 1.06 & -0.64 \\ 0 & 1.54 & -0.93 \\ 0.82 & 0.48 & -0.29 \\ 0 & 0 & 0 \end{bmatrix} \begin{array}{l} P_1' \\ P_2' \\ P_3' \\ P_4' \\ P_5' \\ P_6' \\ P_7' \\ P_8' \end{array}$$

где штрих обозначает вершину проекционной тени.

Из положения источника видны только передняя, левая и верхняя грани; они и

порождают проек. ные пл ости

редняя: $P_1P_3P_6P$ — $P_1P$ [illegible]

ая: $P P P_5P_4$ — $P$ [illegible] $P P$

ерхняя: $P_2P P_5P$ — $P P_4P'P$

Отметим, что плоскости видимых граней не содержат $P$, поэтому ее проекция не входит в видимые грани. Обратным преобразованием $T^{-1}$, построенным при вычислении $T$, возвратимся в исходную ориентацию:

$$S = [P\ \|\ I]^{-1} \begin{bmatrix} 0 & 5 & 1 \\ 0 & 5 & 1 \\ 0 & 5 & 1 \\ 1 & 5 & 1 \\ 0 & 2 & 1 \\ 0 & 2 & 1 \\ 0 & 2,5 & 1 \\ 1 & 2,5 & 1 \end{bmatrix} \begin{matrix} S_1 \\ \\ S_3 \\ S_4 \\ S_5 \\ S_6 \\ S_7 \\ S \end{matrix}$$

Проекции теней на плоскости $P_1$, $S_1$ , $S$ и $S$ $S$ $S$ [illegible] общим контуром — $S_4$ $S_3$

...ая 5.28, в ...ая ... ...ается ... ... вокруг оси $z$ на ... 35 ... ... ... ... ... ...ности на положительной полуоси $z$. ... ... ... ... ...

способ получения матрицы. Букнайт и Кели проецируют сцену на сферу с центром в источнике света и применяют к спроецированным многоугольникам габаритные тесты с прямоугольной оболочкой для исключения большинства случаев. Можно воспользоваться также способом, описанным в примере 5.4, т. е. совместить вектор направления к источнику с осью $z$. Затем количество вариантов можно сильно сократить путем использования простых трехмерных габаритных тестов. Для еще большего сокращения числа вариантов можно применить более сложные методы сортировки, например приоритетную сортировку Ньюэла — Ньюэла — Санча (см. разд. 4.8). Рассмотрим для иллюстрации несложный пример.

**Пример 5.5** [illegible]

[illegible]

**Таблица 5.2** [illegible]

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| [illegible] | [illegible] | | | | | | | |
| [illegible] | Правый | 1 | 1 | 0 | 0 | 0 | 0 | 1 |
| [illegible] | Левый | 0 | | 1 | | 0 | 1 | 1 |
| [illegible] | Нижний | 1 | 0 | 1 | | 0 | 0 | 1 |
| [illegible] | Верхний | 1 | 0 | 0 | 0 | 1 | 0 | 1 |
| [illegible] | Ближний | 1 | 1 | 1 | 0 | | 0 | 1 |
| [illegible] | [illegible]льний | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| | Пересечение | | 0 | | | | 0 | 0 |

Обычно матрица включается в связный список, объединяющий тени и многоугольники.

Второй этап — обработка сцены относительно положения наблюдателя — состоит из двух процессов сканирования. В интервальном алгоритме построчного сканирования, например Уоткинса, первый процесс определяет, какие отрезки на интервале видимы (см. разд. 4.11). Второй с помощью списка теневых многоугольников находит, падает ли тень на многоугольник, который создает видимый отрезок на данном интервале. Второе сканирование для интервала производится следующим образом:

если нет ни одного теневого многоугольника, то видимый отрезок изображается;

если для многоугольника, содержащего видимый отрезок, имеются теневые многоугольники, но они не пересекают и не покрывают данный интервал, то видимый отрезок изображается;

если интервал полностью покрывается одним или несколькими теневыми многоугольниками, то интенсивность изображаемого видимого отрезка определяется с учетом интенсивностей этих многоугольников и самого отрезка;

если один или несколько теневых многоугольников частично покрывают интервал, то он разбивается в местах пересечения с ребрами теневых многоугольников. Затем алгоритм применяется рекурсивно к каждому из подынтервалов до тех пор, пока интервал не будет изображен.

Здесь предполагается, что интенсивность видимого отрезка зависит от интенсивности теневого многоугольника. В простейшем случае можно считать, что тени абсолютно черные. Однако, если взять источник света и два каких-нибудь предмета, то можно увидеть, что тень может быть не только такой. Интенсивность, т. е. чернота тени, зависит от интенсивности источника и от расстояния между затененной гранью и гранью, отбрасывающей тень. Это вызвано ограниченным размером источника и тем, что на затененную поверхность падает рассеянный свет.

Для того чтобы смоделировать такой эффект, можно установить пропорциональную зависимость интенсивности тени и источника. Если накладывается несколько теней, то их интенсивности складываются. Более сложных расчетов требует правило, позволяющее сделать интенсивность тени пропорциональной как интенсивности источника, так и расстоянию между поверхностью, на которую падает тень, и поверхностью, отбрасывающей тень.

Можно изменить алгоритм, использующий $z$-буфер (см. разд. 4.7), так, чтобы он включал построение теней [5-20]. Модифицированный алгоритм состоит из двух шагов:

Строится сцена из точки наблюдения, совпадающей с источником. Значения $z$ для этого вида хранятся в отдельном теневом $z$-буфере. Значения интенсивности не рассматриваются.

Затем сцена строится из точки, в которой находится наблюдатель. При обработке каждой поверхности или многоугольника его глубина в каждом пикселе сравнивается с глубиной в $z$-буфе-

ре наблюдателя. Если поверхность видима, то значения $x$, $y$, $z$ из вида наблюдателя линейно преобразуются в значения $x'$, $y'$, $z'$ на виде из источника. Для того чтобы проверить, видимо ли значение $z'$ из положения источника, оно сравнивается со значением теневого $z$-буфера при $x'$, $y'$. Если оно видимо, то оно отображается в буфер кадра в точке $x$, $y$ без изменений. Если нет, то точка находится в тени и изображается согласно соответствующему правилу расчета интенсивности с учетом затенения, а значение в $z$-буфере наблюдателя заменяется на $z'$.

Для этого метода можно непосредственно использовать алгоритм построчного сканирования с $z$-буфером (см. разд. 4.10). В этом случае применяется буфер размером с одну сканирующую строку. Уильямс [5-20] модифицировал метод, чтобы строить криволинейные тени на изогнутых поверхностях. Сначала создается вид из точки наблюдения, а затем, как постпроцесс, выполняется поточечное линейное преобразование к виду из источника и построение теней. Уильямс отметил, что таким способом неправильно изображаются блики: в тени они просто становятся темнее, хотя ясно, что там их вообще не должно быть. Уильямсом также были исследованы эффекты квантования, возникающие в пространстве изображения в результате преобразования от одной точки наблюдения к другой.

Азертон [5-21, 5-22] включил построение теней в алгоритм удаления невидимых поверхностей (см. разд. 4.5), основанный на методе отсечения Вейлера — Азертона (см. разд. 3.17). Его преимущество состоит в том, что он работает в объектном пространстве и результаты годятся как для точных расчетов, так и для синтеза изображений. Процесс состоит из двух шагов.

На первом шаге с помощью алгоритма удаления невидимых поверхностей выделяются освещенные, т. е. видимые из положения источника грани. Для повышения эффективности алгоритма в памяти хранятся именно они, а не грани, лежащие в тени. Если хранить теневые, т. е. невидимые многоугольники, то придется запоминать и все нелицевые грани объекта, которые обычно отбрасываются до применения алгоритма удаления невидимых поверхностей. Для выпуклого многогранника это удвоило бы количество обрабатываемых граней.

Освещенные многоугольники помечаются и преобразуются к исходной ориентации, где они приписываются к своим прототипам в качестве многоугольников детализации поверхности. Эта операция

выполняется путем присвоения своего номера каждому многоугольнику сцены. В процессе удаления невидимых граней многоугольник может быть разбит на части, которые сохраняют тот же номер. Поэтому каждый кусок освещенной грани можно связать с соответствующим исходным многоугольником или с любой его частью.

Для того чтобы не получить ложных теней, сцену надо рассматривать только в пределах видимого или отсекающего объема, определенного положением источника. Иначе область вне этого объема окажется затененной и наблюдатель увидит ложные тени. Это ограничение требует также, чтобы источник не находился в пределах сцены, так как в этом случае не существует перспективного или аксонометрического преобразования с центром в источнике, которое охватывало бы всю сцену.

На втором шаге объединенные данные о многоугольниках обрабатываются из положения наблюдателя. Если какая-либо область

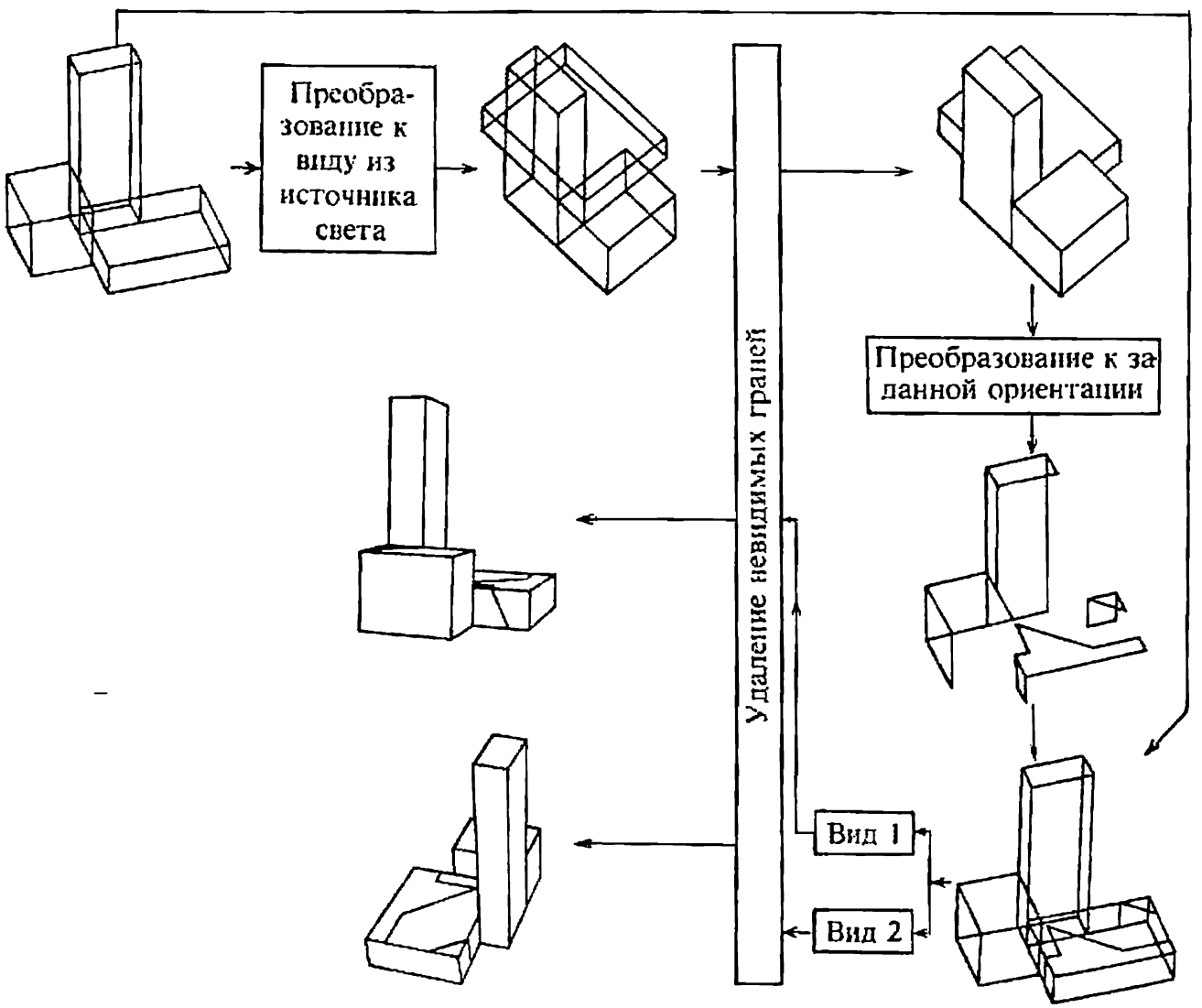



Рис. 5.29. Построение теней в алгоритме удаления невидимых поверхностей Вейлера—Азертона. (С разрешения П. Азертона, Программа машинной графики, Корнелльский университет.)

Рис. 5.30. Трассировка луча с учетом теней.

не освещена, применяется соответствующее правило расчета интенсивности с учетом затенения. Основные стадии процесса показаны на рис. 5.29.

Если источников несколько, то к базе данных добавляется несколько наборов освещенных граней. На цветной вклейке 3 приведен результат для двух источников.

Алгоритм выделения видимых поверхностей трассировкой лучей, описанный в разд. 4.13, также можно расширить, чтобы включить построение теней [5-16]. Процесс вновь делится на два этапа. На первом, как и в предыдущем случае, трассировкой луча от точки наблюдения через плоскость проекции определяются видимые точки сцены (если таковые есть).

На втором этапе вектор (луч) трассируется от видимой точки до источника света. Если между ними в сцене есть какой-нибудь объект то свет от источника не попадает в данную точку, т. е. она оказывается в тени (рис. 5.30). Для того чтобы поиск вдоль локального вектора направления света был эффективнее, можно воспользоваться методами из разд. 4.13.

Кук [5-8] предложил довольно простой способ построения полутеней, хотя, как уже говорилось, обычно они не учитываются. В модели освещения Кука — Торрэнса источнику конечного размера противолежит телесный угол $d\omega$ (см. разд. 5.8), поэтому, закрывая часть источника, можно уменьшить телесный угол, а следовательно, и интенсивность падающего от источника света. При этом соответственно снижается и отраженная интенсивность.

Это показано на рис. 5.31 для прямого края четырехугольника и сферического источника. Средняя линия полутени рассчитывается в

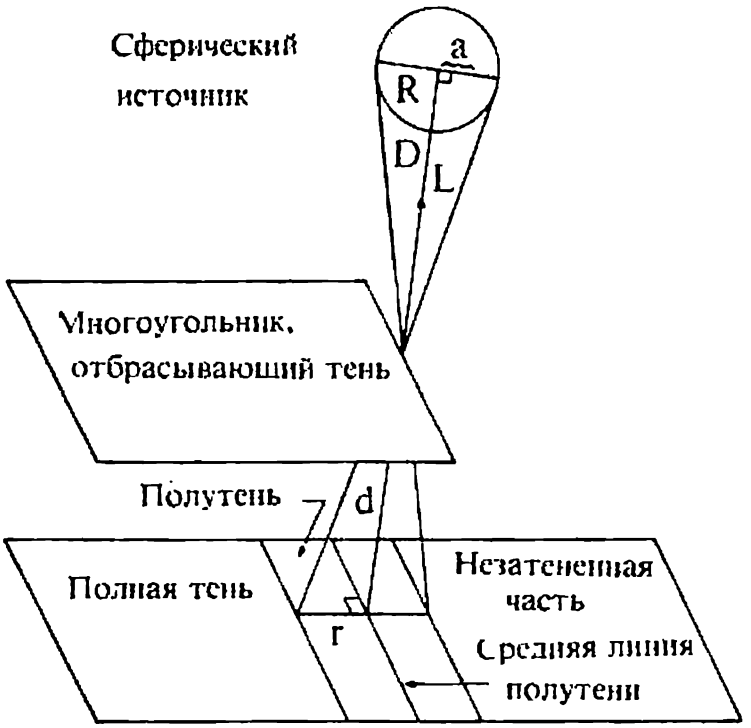



Рис. 5.31. Полутени.

предположении, что точечный источник находится в центре сферы. С помощью подобных треугольников найдем проекцию половины ширины полутени $r$ на направление $\mathbf{L}$ (рис. 5.31):

$$\frac{r(\mathbf{n}\cdot\mathbf{L})}{d} = \frac{R}{D}$$

где $d$ — расстояние от точки, отбрасывающей тень, до соответствующей точки на средней линии полутени; $D$ — расстояние от точки, отбрасывающей тень, до центра сферического источника· $R$ — радиус сферического источника.

Если смотреть от многоугольника, отбрасывающего тень, то телесный угол источника $d\omega$ есть

$$d\omega = \pi\left(\frac{R}{D}\right)^2$$

поэтому половина ширины полутени равна

$$r = \frac{d}{\hat{\mathbf{n}}\cdot\hat{\mathbf{L}}}\frac{R}{D} = \frac{d}{\hat{\mathbf{n}}\cdot\hat{\mathbf{L}}}\sqrt{\frac{d\omega}{\pi}}$$

Это означает, что если телесный угол источника меньше, то он создает более резкую тень (т. е. с меньшим $r$). У точечного источника $d\omega = 0$ и $r = 0$, поэтому полутени нет вообще. При сближении затеняемой и затеняющей поверхности $d$ и $r$ уменьшаются, и тень становится резче.

Интенсивность точек полутени определяется видимой частью источника. Для сферического источника, частично видимого от $-R$ до $a$, эта доля составляет

$$A_{frac} = \frac{1}{\pi R^2}\int_{-R}^{a} 2\sqrt{R^2 - x^2}\,dx = \frac{1}{2} + \frac{1}{\pi}\left[\frac{a}{R}\sqrt{1 - \left(\frac{a}{R}\right)^2} + \sin^{-1}\left(\frac{a}{R}\right)\right]$$

Результаты показывают, что на одном краю полутень получается более четкой. Кук рекомендует хранить эти данные в таблице цветов. Однако проще рассчитывать линейную аппроксимацию

$$A_{frac} = \frac{1}{2}\left(1 + \frac{a}{R}\right)$$

которая дает погрешность меньше 7%.

## 5.11. ФАКТУРА

В машинной графике фактурой называется детализация строения поверхности. Обычно рассматриваются два вида детализации. Первый состоит в том, чтобы на гладкую поверхность нанести заранее заданный узор. После этого поверхность все равно остается гладкой. Наложение узора на гладкую поверхность выполняется с помощью функции отображения. Второй тип детализации заключается в создании неровностей на поверхности. Такие шероховатые поверхности реализуются путем внесения возмущений в параметры, задающие поверхность.

Впервые метод для нанесения рисунка (узора) на поверхность предложил Кэтмул [5-23]. Этот способ вытекает из его алгоритма разбиения для криволинейных поверхностей (см. разд. 4.6). Блинн и Ньюэл усовершенствовали подход Кэтмула и включили в алгоритм отражение и блики на криволинейных поверхностях [5-24].

Главным при нанесении рисунка на поверхность является отображение, поэтому в данном случае задача сводится к преобразованию систем координат. Если рисунок задан в фактурном пространстве в прямоугольной системе координат $(u, w)$, а поверхность — в другой прямоугольной системе координат $(\theta, \varphi)$, то для нанесения рисунка на поверхность нужно найти или задать функцию отображения одного пространства на другое, т. е.

$$\theta = f(u, w) \quad \varphi = g(u, w)$$

или

$$u = r(\theta, \varphi) \quad w = s(\theta, \varphi)$$

Обычно, хотя необязательно, предполагается, что функция отображения линейна: $\theta = Au + B \quad \varphi = Cw + D$, где коэффициенты $A$, $B$, $C$, $D$ выводятся из соотношения между двумя известными точками в системах координат. Рассмотрим пример.

**Пример 5.6.** Отображение

Узор, показанный на рис. 5.32 a, необходимо отобразить на кусок поверхности, заданный октантом сферы (рис. 5.32, b). Узор представляет собой простую сетку из пересекающихся прямых. Параметрическое представление октанта сферы:

$$x = \sin\theta \sin\varphi, \quad 0 \leqslant \theta \leqslant \pi/2, \quad \pi/4 \leqslant \varphi \leqslant \pi/2,$$
$$y = \cos\varphi,$$
$$z = \cos\theta \sin\varphi.$$

Пусть функция отображения линейна, т. е.

$$\theta = Au + B \quad \varphi = Cw + D$$

и углы узора переходят в углы октанта:

$$u = 0,\ w = 0 \quad \text{при } \theta = 0,\ \varphi = \pi/2$$
$$u = 1,\ w = 0 \quad \text{при } \theta = \pi/2,\ \varphi = \pi/2$$
$$u = 0,\ w = 1 \quad \text{при } \theta = 0,\ \varphi = \pi/4$$
$$u = 1,\ w = 1 \quad \text{при } \theta = \pi/2,\ \varphi = \pi/4$$

Отсюда

$$A = \pi/2 \quad B = 0 \quad C = -\pi/4 \quad D = \pi/2$$

т. е. линейная функция отображения пространства $uw$ на пространство $\theta\varphi$ есть

$$\theta = \frac{\pi}{2} u, \quad \varphi = \frac{\pi}{2} - \frac{\pi}{4} w.$$

Обратное преобразование имеет вид:

$$u = \frac{\theta}{\pi/2}, \quad w = \frac{\pi/2 - \varphi}{\pi/4}.$$

В табл. 5.3 приведено отображение одной линии из пространства $uw$ в пространство $\theta\varphi$, а затем в декартовы координаты $xyz$. Законченное отображение показано на рис. 5.32 c.

**Таблица 5.3.**

| $u$ | $w$ | $\theta$ | $\varphi$ | $x$ | $y$ | $z$ |
|---|---|---|---|---|---|---|
| 1/4 | 0 | $\pi/2$ | $\pi/2$ | 0.38 | 0 | 0.92 |
| | 1/4 | | $7/16\pi$ | 0.38 | 0.20 | 0.91 |
| | 1/2 | | $3/8\pi$ | 0.35 | 0.38 | 0.85 |
| | 3/4 | | $5/16\pi$ | 0.32 | 0.56 | 0.77 |
| | 1 | | $\pi/4$ | 0.27 | 0.71 | 0.65 |

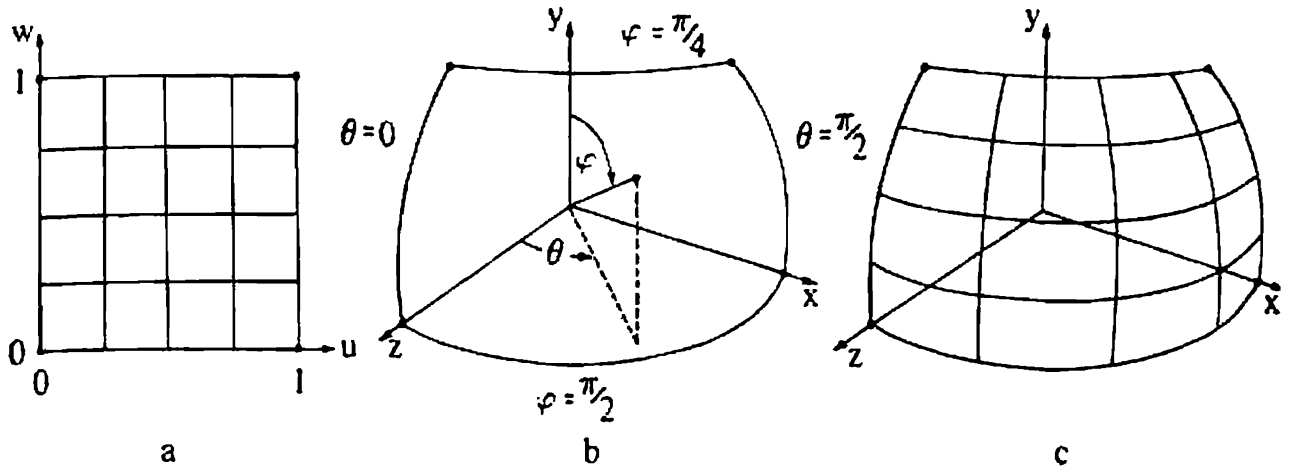



Рис. 5.32. Отображение.

Узор на рис. 5.32 задан математически, но он может быть также нарисован от руки, либо получен путем сканирования фотографий и т. д. Для нанесения рисунка на поверхность необходимо отображение объектного пространства в пространство изображения, а также рассмотренное выше преобразование из фактурного пространства в объектное. Может быть применено также любое видовое преобразование. Если для пространства изображения используется растровое устройство, то можно применить два несколько отличающихся метода.

Первый основан на алгоритме разбиения Кэтмула (см. разд. 4.6). Алгоритм Кэтмула разбивает кусок поверхности на фрагменты до тех пор, пока фрагмент не будет покрывать центр только одного пиксела. Затем интенсивность пиксела можно было бы найти по узору, который получен при отображении параметрических значений центра фрагмента или пиксела в фактурное пространство. Но, как указал Кэтмул, такой метод поточечной выборки приводит к сильному лестничному эффекту. Например, можно потерять значительную часть или даже весь узор, показанный на рис. 5.32, а, если все выбранные точки окажутся в «белых» областях узора. Чтобы этого не случилось, Кэтмул, разбивает как кусок поверхности, так и соответствующий узор. Когда найден фрагмент покрывающий центр только одного пиксела, усредненная интенсивность соответствующего фрагмента узора используется для определения интенсивности пиксела.

В общем случае полученный фрагмент узора не обязательно прямоугольный. Когда он изображается на растровом устройстве, его интенсивность равна средней взвешенной интенсивности пикселов фактуры на данном фрагменте. В качестве весовой функции берется отношение площади пикселов фактуры, находящихся внутри

Рис. 5.33. Узор, нанесенный на бутылку, составленную из кусков $\beta$-сплайна. (С разрешения Б. Барски.)

фрагмента, к их полной площади. Блинн и Ньюэл применяли более качественный пирамидальный сглаживающий фильтр 2 × 2, предложенный Кроу (см. разд. 2.25). Барски получил изображение бутылки, показанное на рис. 5.33, отобразив узор шахматной доски на кусок поверхности, заданной $\beta$-сплайнами.

Таким образом, алгоритм разбиения Кэтмула начинает свою работу на куске поверхности в объектном пространстве и развивается в двух направлениях: в пространстве изображения и фактурном пространстве. Рассмотрим этот метод на примере.

**Пример 5.7.** Алгоритм разбиения узора

Рассмотрим кусок поверхности, образованный октантом единичной сферы (рис. 5.32, b), простой узор в виде решетки (рис. 5.32, a). Кусок поверхности нужно повернуть на $-45°$ вокруг оси $y$, затем на 35° вокруг оси $x$ и изобразить в ортогональной проекции на растре 32 × 32 (рис. 5.34, a). Узор задан на растре 64 × 64 где каждая строка имеет один пиксел в высоту (рис. 5.34, b).

Сначала кусок поверхности разбивается на фрагменты, а затем преобразуется в пространство изображения. Начало координат объектного пространства находится в центре растра 32 × 32. Для того чтобы фрагмент покрывал центр только одного пиксела, необходимо четыре разбиения (рис. 5.34, a). В пространстве изображения этот фрагмент имеет прямоугольную форму. Пределы изменения параметров $\theta$ и $\varphi$ для этого фрагмента в объектном пространстве суть $0 \leqslant \theta \leqslant \frac{\pi}{32}$, $\frac{31\pi}{64} \leqslant \varphi \leqslant \frac{\pi}{2}$. С помощью функции обратного отображения из объектного пространства $\theta\varphi$ в фактурное пространство $uw$ (пример 5.6)

$$u = \frac{\theta}{\pi/2}, \quad w = \frac{\pi/2 - \varphi}{\pi/4}$$

получим координаты углов фрагмента в фактурном пространстве:

$$\theta = 0, \quad \varphi = \pi/2 \quad \rightarrow \quad u = 0, \quad w = 0,$$
$$\theta = 0, \quad \varphi = 59\pi/64 \quad \rightarrow \quad u = 0, \quad w = 1/16,$$

$\theta = \pi/32, \quad \varphi = 59\pi\ 64 \quad - \quad u = 1\ 1\!\!\!

пиксела в пространстве изображения равна средней интенсивности пикселов, покрытых этой площадью в фактурном пространстве. На полученный коэффициент умножается диффузная компонента модели освещения. Существуют, конечно, и другие, более сложные методы, позволяющие устранять лестничный эффект. Рассмотрим данный метод на простом примере.

**Пример 5.8.** Построение фактуры обратным отображением пикселов

Рассмотрим опять октант единичной сферы (рис. 5.32, b) и узор — простую решетку на растре 64 × 64 (рис. 5.34, b). Октант снова нужно повернуть на −45° вокруг оси $y$, на 35° вокруг оси $x$ и изобразить в ортогональной проекции на растре 32 × 32 (рис. 5.34, a).

Рассмотрим интенсивность пиксела при $P_x = 21$, $P_y = 15$ на рис. 5.34, a. Пиксел определяется своим левым нижним углом. Площадь пиксела ограничена $21 \leqslant P_x \leqslant 22$, $15 \leqslant P_y \leqslant 16$. Предположим, что окно в объектном пространстве, соответствующее растру 32 × 32 в пространстве изображения, задано: $-1 \leqslant x' \leqslant 1$, $-1 \leqslant y' \leqslant 1$. Тогда

$$x' = \frac{P_x}{16} - 1 \qquad y' = \frac{P_y}{16} - 1$$

Из уравнения единичной сферы $z' = \sqrt{1 - (x'^2 + y'^2)}$, где $x'$, $y'$ — координаты в объемном пространстве после видового преобразования. Координаты углов пикселов на рассматриваемой поверхности таковы:

| $P_x$ | $P_y$ | $x'$ | $y'$ | $z'$ |
|---|---|---|---|---|
| 21 | 15 | 0.3125 | −0.0625 | 0.948 |
| 22 | 15 | 0.3750 | −0.0625 | 0.925 |
| 22 | 16 | 0.3750 | 0 | 0.927 |
| 21 | 16 | 0.3125 | 0 | 0.950 |

Видовое преобразование до проецирования на плоскость картины и обратное преобразование суть

$$[T] = \begin{bmatrix} 0.707 & -0.406 & 0.579 & 0 \\ 0 & 0.819 & 0.574 & 0 \\ -0.707 & -0.406 & 0.579 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad [T]^{-1} = \begin{bmatrix} 0.707 & 0 & -0.707 & 0 \\ -0.406 & 0.819 & -0.406 & 0 \\ 0.579 & 0.574 & 0.579 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Обратное преобразование дает углы пиксела на куске поверхности в исходной ориентации, а именно:

$$[x \quad y \quad z \quad 1] = [x' \quad y' \quad z' \quad 1][T]^{-1}$$

| $P_x$ | $P_y$ | $x$ | $y$ | $z$ |
|---|---|---|---|---|
| 21 | 15 | 0.795 | 0.493 | 0.341 |
| 22 | 15 | 0.826 | 0.479 | 0.296 |
| 22 | 16 | 0.802 | 0.532 | 0.272 |
| 21 | 16 | 0.771 | 0.545 | 0.329 |

Из параметрического представления единичной сферы

$$x = \sin\theta \sin\phi$$
$$y = \cos\phi$$
$$z = \cos\theta \sin\phi$$

получаем

$$\phi = \cos^{-1} y \qquad \theta = \sin^{-1}\left(\frac{x}{\sin\phi}\right)$$

в параметрическом пространстве. С помощью преобразования из примера 5.6 отобразим параметрическое пространство в фактурное:

$$u = \frac{\theta}{\pi/2} \qquad v = \frac{\pi/2 - \phi}{\pi/4}$$

т. е. углы пиксела в фактурном пространстве

| $P_x$ | $P_y$ | $\phi$ | $\theta$ | $u$ | $v$ |
|---|---|---|---|---|---|
| 21 | 15 | 60.50° | 66.04° | 0.734 | 0.656 |
| 22 | 15 | 61.34° | 70.30° | 0.781 | 0.636 |
| 22 | 16 | 57.88° | 71.28° | 0.792 | 0.714 |
| 21 | 16 | 56.99° | 66.88° | 0.743 | 0.734 |

На рис. 5.35 показан результат, где криволинейный участок аппроксимирован четырехугольным.

Заданный на растре узор проходит через левое ребро пиксела. Интенсивность пиксела можно вычислить несколькими методами (см. разд. 2.25), например, как взвешенное среднее значение пикселов фактуры, центры которых лежат внутри пиксела изображения. В данном случае отношение «черных» пикселов фактуры, образующих узор, ко всем пикселам с центрами внутри пикселов изображения равно 5/18. На это число умножается интенсивность диффузной составляющей в модели освещения.

В рассмотренных методах рисунок наносится на гладкую поверхность и она после этого остается гладкой. Для того чтобы поверхность казалась шероховатой, можно оцифровать фотографию

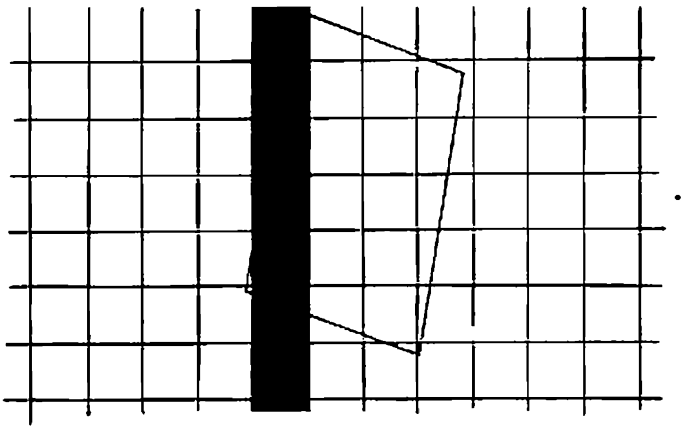


Рис. 5.35. Пиксел изображения в фактурном пространстве.

перегулярной фактуры и отобразить ее на поверхность. Однако при этом будет казаться, что неровности нарисованы на гладкой поверхности. Дело в том, что в векторе нормали к настоящей шероховатой поверхности и, следовательно, в направлении отражения есть небольшая случайная составляющая. На этой основе Блинн [5-25] разработал метод возмущения нормали для построения неровных поверхностей.

В любой точке поверхности $Q(u, w)$ частные производные по направлениям $u$ и $w$, $Q_u$ и $Q_w$, лежат в плоскости, касательной к поверхности в этой точке. Нормаль в ней определяется векторным произведением

$$\mathbf{n} = \mathbf{Q}_u \otimes \mathbf{Q}_w$$

Блинн строит новую поверхность, которая выглядит шероховатой, внося в направлении нормали функцию возмущения $P(u, w)$. Таким образом, радиус-вектор точки на новой поверхности есть

$$\mathbf{Q}'(u, w) = \mathbf{Q}(u, w) + P(u, w)\frac{\mathbf{n}}{\mathbf{n}|}$$

Нормаль к возмущенной поверхности имеет вид

$$\mathbf{n}' = \mathbf{Q}'_u \otimes \mathbf{Q}'_w$$

Частные производные $\mathbf{Q}'_u$ и $\mathbf{Q}'_w$ выражаются формулами

$$\mathbf{Q}'_u = \mathbf{Q}_u + P_u\frac{\mathbf{n}}{|\mathbf{n}|} + P\left(\frac{\mathbf{n}}{\mathbf{n}|}\right)_u$$

$$\mathbf{Q}'_w = \mathbf{Q}_w + P_w\frac{\mathbf{n}}{|\mathbf{n}|} + P\left(\frac{\mathbf{n}}{|\mathbf{n}|}\right)_w$$

Последним членом можно пренебречь, так как $P$ (функция возмущения) очень мала. Поэтому

$$\mathbf{Q}'_u \doteq \mathbf{Q}_u + P_u \frac{\mathbf{n}'}{\mathbf{n}'}$$

$$\mathbf{Q}'_w \doteq \mathbf{Q}_w + P_w \frac{\mathbf{n}}{|\mathbf{n}|}$$

Возмущенная нормаль имеет вид

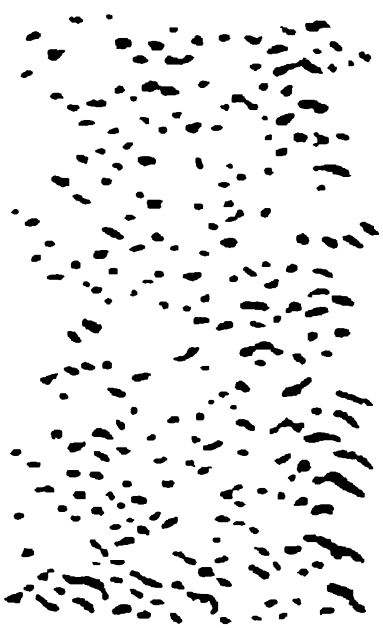
$$\mathbf{n}' = \mathbf{Q}_u \otimes \mathbf{Q}_w + \frac{P_u(\mathbf{n} \otimes \mathbf{Q}_w)}{|\mathbf{n}|} + \frac{P_w(\mathbf{Q}_u \otimes \mathbf{n})}{|\mathbf{n}|} + \frac{P_u P_w(\mathbf{n} \otimes \mathbf{n})}{|\mathbf{n}|^2}$$

Первый член — нормаль $\mathbf{n}$ к исходной поверхности, а последний равен нулю, поэтому

$$\mathbf{n}' = \mathbf{n} + \frac{P_u(\mathbf{n} \otimes \mathbf{Q}_w)}{|\mathbf{n}|} + \frac{P_w(\mathbf{Q}_u \otimes \mathbf{n})}{|\mathbf{n}|}$$

где два последних члена, приведенные к единичной длине, представляют собой возмущение нормали к поверхности и создают соответствующий эффект в модели освещения.

В качестве $P$ можно использовать почти любую функцию, у которой существуют частные производные. Блинн работал с простыми сетками, заданными математически, битовыми изображениями символов, узорами, заданными с помощью $z$-буфера, и рисунками, сделанными от руки. Например, на рис. 5.36. Т. ван Хук отобразил заданную фактуру на кусок бикубической поверхности. Если узор не



a b

Рис. 5.36. Фактура, нанесенная на бикубическую поверхность: (a) шаблон фактуры, (b) результат. (С разрешения Т. ван Хука, Adage, Inc.)

определяется аналитически, то функция возмущения записывается как двумерная таблица цветов с индексами $u$, $w$. Промежуточные значения вычисляются билинейной интерполяцией табличных величин, а производные $P_u$ и $P_w$ вычисляются методом конечных разностей.

Эффект шероховатости зависит от масштаба изображаемого объекта. Например, если размер объекта увеличится в два раза, то величина вектора нормали возрастет в четыре раза, а его возмущения — только в два. Это приводит к тому, что увеличенный объект кажется более гладким. Однако масштаб фактуры на перспективном изображении не зависит от перемещения объекта в пространстве по направлению к наблюдателю или от него.

При изображении фактуры с помощью функции возмущения может появиться лестничный эффект, но если воспользоваться рассмотренным выше способом усреднения по площади фактуры или методами устранения лестничного эффекта, основывающимися на предварительной фильтрации, то фактура может полностью сгладиться. Как указал Блинн [5-25], необходимо рассчитывать изображение с разрешением, большим, чем у дисплея, а затем отфильтровать или усреднить его и вывести с более низким разрешением экрана (см. разд. 2.25).

Один из последних методов построения нерегулярностей основан на фрактальных поверхностях. Фрактальная поверхность состоит из случайно заданных полигональных или биполиномиальных поверхностей. В машинной графике этот метод первым применили Карпентер [5-26], а также Фурнье и Фассел [5-27]. С помощью фрактальных поверхностей изображались природные объекты — камни, деревья, облака, а также ландшафты. Этот метод основан на работе Мандельброта [5-28].

Для того чтобы получить полигональную фрактальную поверхность, исходный многоугольник рекурсивно разбивается на фрагменты, как показано на рис. 5.37. Для этого можно, например, случайным образом сместить центр и середины сторон многоугольника, причем и исходный, и полученный многоугольники не обязательно должны быть плоскими.

Одно из преимуществ фрактальных поверхностей в том, что их можно разбивать «бесконечно» и получить любой уровень детализации. Он может зависеть от положения наблюдателя: чем ближе точка наблюдения, тем с большей степенью детализации изображается поверхность. Если наблюдатель находится далеко, объем вычислений значительно сокращается. Фрактальная поверхность изо-

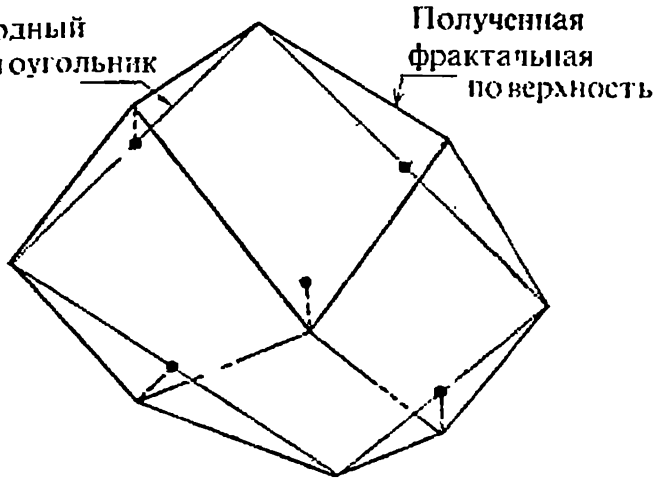



Рис. 5.37. Формирование фрактальной поверхности.

бракается с помощью любого подходящего алгоритма удаления невидимых поверхностей и любой модели освещения. Однако число разбиений возрастает со скоростью выше линейной, поэтому между количеством разбиений и уровнем детализации должен быть некоторый компромисс. Иначе потребуется слишком много вычислений.

Типичный пример (см. цветную вклейку 4) получен Кадзия [5-29] с помощью алгоритма выделения непрозрачных видимых поверхностей методом трассировки лучей. Верхняя сцена на цветной вклейке состоит из 16 384 фрактальных треугольников, а нижняя — из 262 144 треугольников. Обратите внимание на собственные тени на изображениях.

## 5.12. ГЛОБАЛЬНАЯ МОДЕЛЬ ОСВЕЩЕНИЯ С ТРАССИРОВКОЙ ЛУЧЕЙ

Модель освещения предназначена для того, чтобы рассчитать интенсивность отраженного к наблюдателю света в каждой точке (пикселе) изображения. Она может быть локальной или глобальной. В первом случае во внимание принимается только свет, падающий от источника (источников), и ориентация поверхности. Во втором учитывается также свет, отраженный от других объектов сцены или пропущенный сквозь них. Глобальная модель воспроизводит чрезвычайно важные эффекты; некоторые из них показаны на рис. 5.38

Сфера, треугольная призма и параллелепипед на рис. 5.38 непрозрачны, их поверхность зеркальна. Наблюдатель смотрит из точки 0 на точку 1 на сфере. При этом он видит не только сферу, но и точку 2 на призме. Призма, загороженная от наблюдателя параллелепипедом, становится видимой из-за отражения в сфере. Точка 5

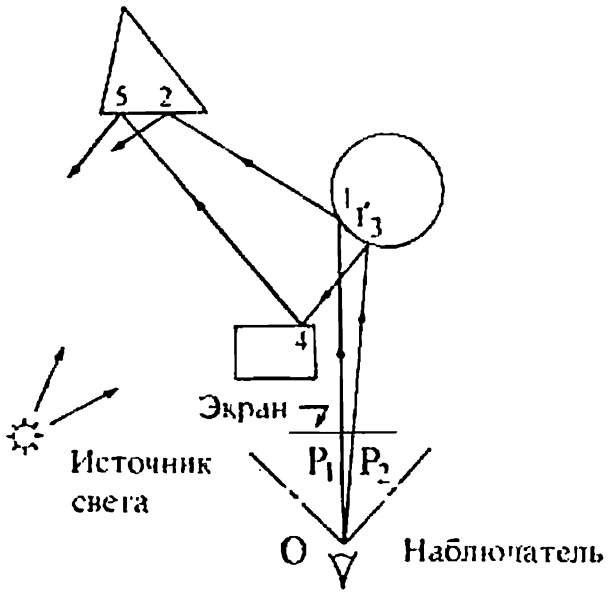



Рис. 5.38. Глобальное освещение.

на призме видима еще более косвенно. Она отражается от *обратной* стороны параллелепипеда в точке 4 к точке 3 на сфере, а затем — к наблюдателю. Кроме того, наблюдатель видит точку 5 в точке 1′ после одного отражения от сферы, поэтому на сфере находятся два изображения призмы. Изображение, расположенное вокруг точки 1, перевернуто, так как получается после одного отражения; вокруг точки 3 не перевернуто, так как получается после двух отражений. Интенсивность второго изображения меньше. Наконец, в сфере отражается обратная сторона параллелепипеда, т. е. наблюдатель видит ее, хотя на нее не падает прямой свет источника. Эта сторона освещается рассеянным светом и светом, отраженным от других объектов сцены.

Отсюда следует, что обычная операция отбрасывания задних граней, применяемая при удалении невидимых поверхностей, здесь не годится, как и предварительная сортировка по глубине. Поэтому из всех алгоритмов удаления невидимых граней, рассмотренных в гл. 4, остается *только* метод трассировки лучей. Таким образом, глобальная модель освещения является частью алгоритмов выделения видимых поверхностей путем трассировки лучей.

Первые работы в этой области принадлежат Уиттеду [5-30] и Кэю [5-13, 5-14]. Алгоритм Уиттеда более общий, он был расширен и часто используется. Синтетические изображения на цветных вклейках 5, 6 и 7 получены Уиттедом [5-30], Потметсилом [5-31, 5-32] и Барром [5-33] с помощью алгоритма Уиттеда или его расширений. На них видны эффекты отражения, преломления и пропускания, а также тени и фактура.

Обычно в машинной графике предполагается, что изображение создается камерой с маленьким отверстием. Потметсил заменил ее

в алгоритме Уиттеда на более реалистичную модель с характеристиками объектива и диафрагмы настоящего фотоаппарата, которая учитывает влияние глубины резкости, фокуса, искажения объектива и фильтров. В последовательности кинокадров возможно постепенное появление и исчезновение изображения. Метод Потмесила двухшаговый.

На первом с помощью обычного алгоритма трассировки лучей для камеры с маленьким отверстием по выборочным точкам строится изображение. Для каждого пиксела вместе с интенсивностями RGB хранится также глубина $z$ и данные о видимых поверхностях. Второй шаг работает как постпроцессор и создает модель фотоаппарата с конечной апертурой. Каждая точка преобразуется в круг по законам геометрической оптики. Распределение размеров и интенсивности кругов размытости зависит от значения $z$ точки, характеристик объектива и его апертуры. Для того чтобы найти интенсивность пиксела, нужно сложить интенсивности всех пересекающихся в нем кругов размытости. Типичный результат приведен на цветной вклейке 6.

Уиттед пользуется моделью с такими же членами рассеянного и ламбертовского диффузного отражения, а также зеркального отражения Фонга, как и в локальной модели освещения, описываемой (5.7). Члены, соответствующие глобальному зеркальному отражению и пропусканию, рассчитываются по правилу, показанному на рис. 5.39. Здесь трассируется луч $\mathbf{v}$, падающий на поверхность в точке $Q$. В этой точке он отражается в направлении $r$ и, если поверхность прозрачна, преломляется в направлении $\mathbf{p}$. Здесь $I_t$ — интенсивность света, падающего в точку $Q$ по направлению $\mathbf{p}$. Этот

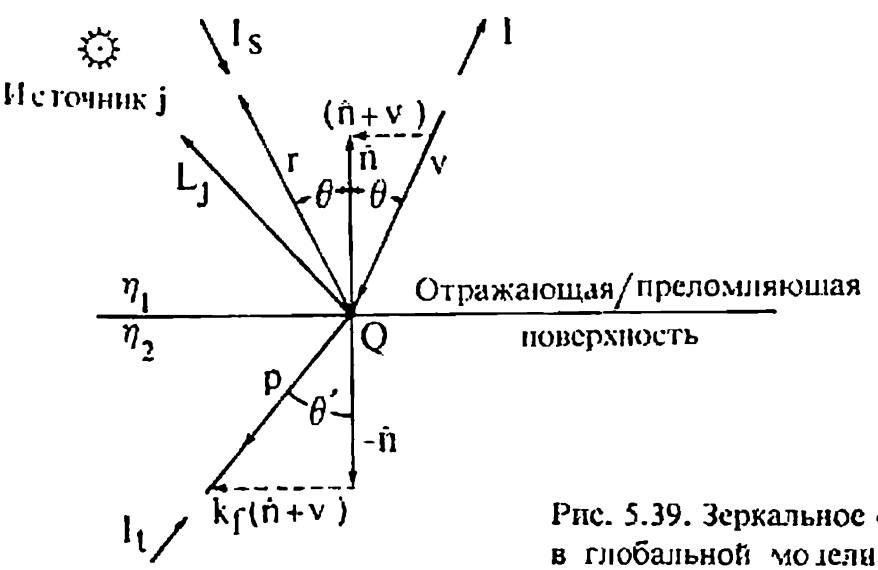



Рис. 5.39. Зеркальное отражение и преломление в глобальной модели освещения Уиттеда.

свет преломляется и достигает наблюдателя, находящегося в направлении $-\mathbf{v}$.

Аналогично $I_s$ — интенсивность зеркально отраженного света, падающего в направлении $-\mathbf{r}$ и отраженного к наблюдателю в точке $Q$, $\mathbf{n}$ — нормаль к поверхности в точке $Q$, $\mathbf{L}_j$ — направление на $j$-й источник света, $\mathbf{S}$ и $\mathbf{R}$ — локальные векторы наблюдения и отражения, $\eta$ — показатель преломления среды, $n$ — степень пространственного распределения Фонга для зеркального отражения (см. разд. 5.2). Тогда наблюдаемая интенсивность $I$ выражается формулой

$$I = k_a I_a + k_d \sum_j I_{l_j}(\hat{\mathbf{n}} \cdot \hat{\mathbf{L}}_j) + k_s \sum_j I_{l_j}(\hat{\mathbf{S}} \cdot \hat{\mathbf{R}}_j)^n + k_s I_s + k_t I_t \qquad (5.16)$$

где $k_a$, $k_d$, $k_s$ — коэффициенты рассеянного, диффузного и зеркального отражения, а $k_t$ — коэффициент пропускания. Уитгед считает их постоянными, однако с помощью любой из рассмотренных выше моделей освещения можно найти зависимость этих коэффициентов от угла падения и длины волны. Две суммы по $j$ в уравнении (5.16) — это диффузное и зеркальное отражение от множества источников.

В разд. 4.13 рассмотрен алгоритм построения видимых непрозрачных поверхностей, в котором луч трассируется до первого пересечения с объектом. В глобальной модели освещения работа на этом не кончается. Здесь предполагается, что падающий луч $\mathbf{v}$ в точке $Q$ отражается в направлении $\mathbf{r}$ и пропускается сквозь поверхность в направлении $\mathbf{p}$, как показано на рис. 5.39. Это значит, что в точке $Q$ образуются еще два луча, для которых нужно найти все

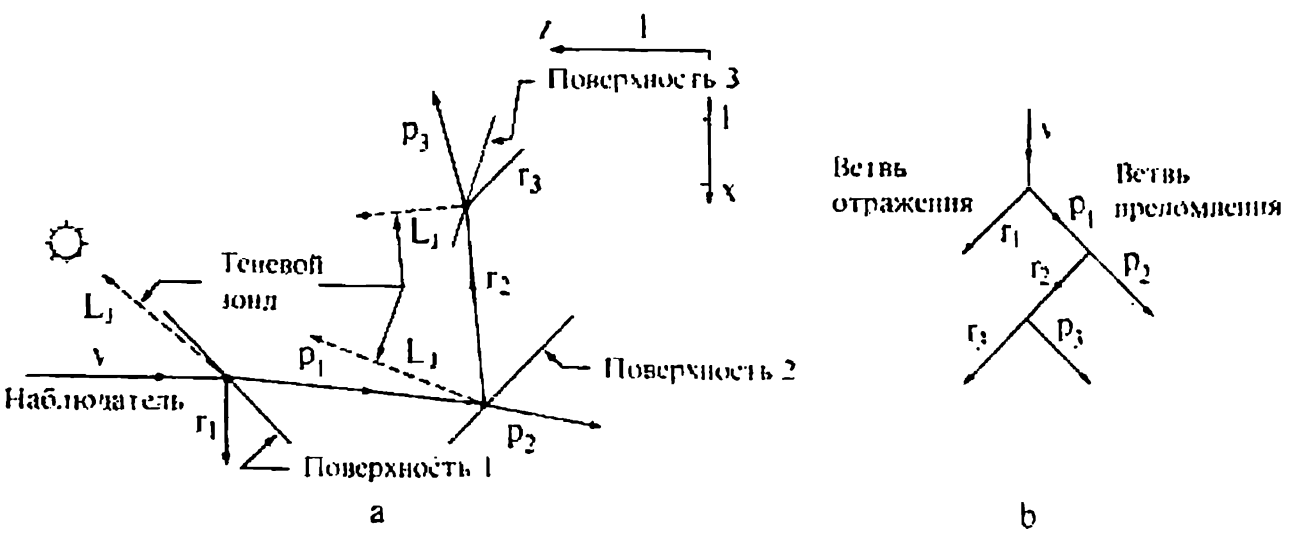



Рис. 5.40. Трассирование луча с отражением и преломлением.

пересечения с объектами сцены. Такой процесс повторяется, пока не останется ни одного пересечения. На рис. 5.40,а он изображен для случая пересечений луча с одной поверхностью, а на рис. 5.40,b построено соответствующее дерево. Каждый узел дерева представляет собой пересечение луча с поверхностью. Из каждого узла исходят две ветви: правая, порожденная преломлением, и левая — отражением. Ветвь кончается, когда луч покидает сцену.

При пересечении луча с поверхностью направления отраженного и пропущенного лучей рассчитываются по законам геометрической оптики. В частности, отраженный луч $\mathbf{r}$ и падающий луч $\mathbf{v}$ лежат в одной плоскости, и угол падения равен углу отражения (см. разд. 5.3). Пропущенный луч преломляется по закону Снеллиуса (см. разд. 5.9). В нашей модели и обозначениях направления $\mathbf{r}$ и $\mathbf{p}$ таковы

$$\mathbf{r} = \mathbf{v}' + 2\hat{\mathbf{n}}$$

$$\mathbf{p} = k_f(\hat{\mathbf{n}} + \mathbf{v}') - \hat{\mathbf{n}}$$

где

$$\mathbf{v}' = \frac{\mathbf{v}}{|\mathbf{v} \cdot \hat{\mathbf{n}}|}$$

$$k_f = (k_\eta^2|\mathbf{v}'|^2 - |\mathbf{v}' + \hat{\mathbf{n}}|^2)^{-1/2}$$

$$k_\eta = \frac{\eta_2}{\eta_1}$$

Здесь $k_\eta$ — отношение показателей преломления, а $\hat{\mathbf{n}}$ — вектор единичной нормали в направлении падающего луча. Если знаменатель $k_f$ комплексный, то происходит полное внутреннее отражение, и $I_t$ предполагается равным нулю.

Для того чтобы определить интенсивность в каждом пересечении луча с поверхностью, надо пройти дерево в обратном направлении. Интенсивность в узлах рассчитывается в соответствии с моделью освещения, причем для каждого следующего узла она ослабляется с расстоянием между точками пересечения. После прохода всего дерева получается окончательная интенсивность пиксела.

Теоретически дерево трассировки луча может быть бесконечно глубоким. Построение оканчивается, когда все лучи уходят за пределы сцены, но его можно прерывать, когда интенсивность узла падает ниже определенного уровня или когда исчерпан запас памяти.

На рис. 5.41 показано внутреннее отражение в замкнутом прозрачном теле. Лучи, зеркально отраженные от внутренних граней, остаются внутри объекта и в конце концов поглощаются. Для наблюдателя они невидимы, однако в каждом пересечении луча с по-

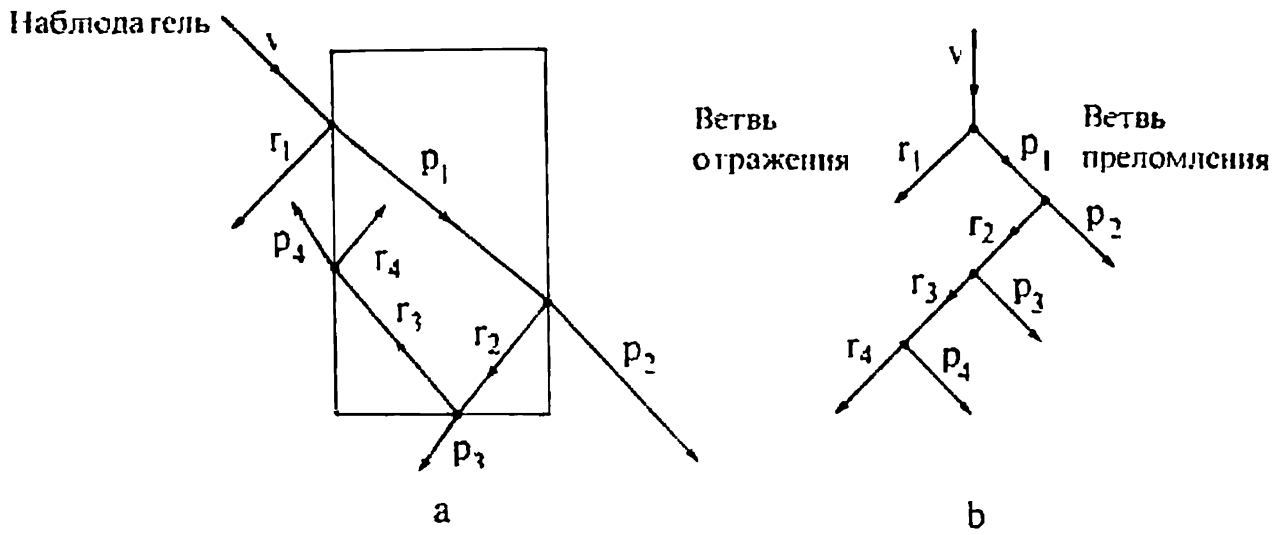



Рис. 5.41. Внутреннее отражение в прозрачных объектах.

верхностью образуется преломленный луч, который покидает объект и может прямо или косвенно достичь точки наблюдения. Такие лучи должны быть оттрассированы.

Для того чтобы включить в алгоритм тени, надо из каждого пересечения луча с поверхностью направить ко всем источникам $L_j$ теневые зонды. Если на этом направлении между данной точкой и источником лежит другой объект, то точка относительно этого источника лежит в тени и его вклад в локальное диффузное и зеркальное отражение уменьшается (см. разд. 5.10). Если поверхность, лежащая на пути луча, непрозрачна, то свет источника вообще не попадает на поверхность, а если тень отбрасывается прозрачной поверхностью, то свет ослабляется в зависимости от ее свойств. Теневые зонды показаны на рис. 5.40.

**Пример 5.9.** Глобальное освещение и трассировка лучей

Рассмотрим простую двумерную сцену, изображенную на рис. 5.40, а. Одиночные плоскости перпендикулярны плоскости бумаги, обозначенной $xz$. Наблюдатель находится в бесконечности на положительной полуоси $z$ при $x = 5$. Координаты единственного точечного источника света $x = 3$, $z = 10$. Поверхности заданы уравнениями плоскостей, т. е.

| | | |
|---|---|---|
| поверхность 1: | $x + z - 12.5 = 0,$ | $4 \leqslant x \leqslant 6;$ |
| поверхность 2: | $x - z - 2 = 0,$ | $4 \leqslant x \leqslant 6;$ |
| поверхность 3: | $x - 3z + 9 = 0,$ | $1 \leqslant x \leqslant 3.$ |

Поверхности обладают следующими оптическими свойствами:

поверхность 1: $k_{a_1} = 0.15, k_{d_1} = 0.15, k_{s_1} = 0.8, k_{t_1} = 0.5, k_{\eta_1} = 1.1;$

поверхность 2: $k_{a_2} = 0.15, k_{d_2} = 0.15, k_{s_2} = 0.8, k_{t_2} = 0.5, k_{\eta_2} = 1.1;$

поверхность 3: $k_{a_3} = 0.15, k_{d_3} = 0.15, k_{s_3} = 0.8, k_{t_3} = 0, k_{\eta_3} = 1.1;$

Интенсивность рассеянного света $I_a = 1.0$, интенсивность источника $I_l = 10$. Степень пространственного распределения Фонга для зеркального отражения $n = 50$ для всех поверхностей.

Луч отслеживается по направлению от наблюдателя к сцене. Соответствующее дерево изображено на рис. 5.40,b. Сначала луч падает на поверхность 1. Подставляя его уравнение до пересечения, $x = 5$, в уравнение плоскости, получаем

$$x + z - 12.5 = 5 + z - 12.5 = 0 - z = 7.5,$$

т. е. пересечение луча с поверхностью, соответствующее первому узлу дерева, происходит в точке $x_1 = 5$, $z_1 = 7.5$. Единичная нормаль к поверхности в этой точке

$$\mathbf{n}_1 = \frac{\mathbf{i}}{\sqrt{2}} + \frac{\mathbf{k}}{\sqrt{2}}$$

Найдем преломленный и отраженный лучи:

$$\mathbf{v}_1 = -\mathbf{k}$$

и

$$\mathbf{v}_1' = \frac{\mathbf{v}_1}{|\mathbf{v}_1 \cdot \mathbf{n}_1|} = \frac{-\mathbf{k}}{\left|(-\mathbf{k}) \cdot \left(\frac{\mathbf{i}}{\sqrt{2}} + \frac{\mathbf{k}}{\sqrt{2}}\right)\right|} = -\sqrt{2}\mathbf{k}$$

Направление отраженного луча есть

$$\mathbf{r}_1 = \mathbf{v}_1' + 2\hat{\mathbf{n}}_1 = -\sqrt{2}\mathbf{k} + 2\left(\frac{\mathbf{i}}{\sqrt{2}} + \frac{\mathbf{k}}{\sqrt{2}}\right) = \sqrt{2}\mathbf{i}$$

Так как

$$\mathbf{v}_1' + \hat{\mathbf{n}}_1 = -\sqrt{2}\mathbf{k} + \frac{\mathbf{i}}{\sqrt{2}} + \frac{\mathbf{k}}{\sqrt{2}} = \frac{\mathbf{i}}{\sqrt{2}} - \frac{\mathbf{k}}{\sqrt{2}}$$

то

$$k_{f1} = \left[k_{\eta 1}^2 |\mathbf{v}_1'|^2 - |\mathbf{v}_1' + \hat{\mathbf{n}}_1|^2\right]^{-1/2} = \left[\left(\frac{1}{1.1}\right)^2 (2) - 1\right]^{-1/2} = 1.238$$

откуда получаем преломленный луч

$$\mathbf{p}_1 = k_{f1}(\hat{\mathbf{n}}_1 + \mathbf{v}_1') - \hat{\mathbf{n}}_1 = 1.238\left(\frac{\mathbf{i}}{\sqrt{2}} - \frac{\mathbf{k}}{\sqrt{2}}\right) - \left(\frac{\mathbf{i}}{\sqrt{2}} + \frac{\mathbf{k}}{\sqrt{2}}\right)$$

$$= 0.168\mathbf{i} - 1.582\mathbf{k}$$

В этой точке отраженный луч покидает сцену и больше не рассматривается. Второй узел образуется при пересечении преломленного луча со второй поверхностью. Запишем преломленный луч $\mathbf{p}_1$ в параметрической форме:

$$x = 5 + 0.168t$$
$$z = 7.5 - 1.582t$$

Подставим в уравнение плоскости:

$$x - z - 2 = 5 + 0.168t - 7.5 + 1.582t - 2 = 1.75t - 4.5 = 0$$

Отсюда $t = 2.571$, и точка пересечения

$$x_2 = 5 + (0.168)(2.571) = 5.432$$
$$z_2 = 7.5 - (1.582)(2.571) = 3.433$$

Расстояние между двумя точками пересечения

$$d_{12} = \sqrt{(x_2 - x_1)^2 + (z_2 - z_1)^2} = \sqrt{(5.432 - 5)^2 + (3.433 - 7.5)^2} = 4.09$$

Теперь в качестве падающего луча рассмотрим $\mathbf{p}_1$ и найдем отраженный и преломленный лучи:

$$\mathbf{v}_2 = \mathbf{p}_1 = 0.168\mathbf{i} - 1.582\mathbf{k}$$

Единичная нормаль к поверхности

$$\mathbf{n}_2 = \frac{-\mathbf{i}}{\sqrt{2}} + \frac{\mathbf{k}}{\sqrt{2}}$$

В результате имеем

$$\mathbf{p}_2 = 0.215\mathbf{i} - 1.119\mathbf{k}$$
$$\mathbf{r}_2 = -1.278\mathbf{i} + 0.136\mathbf{k}$$

Преломленный луч покидает сцену, не пересекаясь с другими объектами, поэтому на ветвь дерева оканчивается. Пересечение отраженного луча $\mathbf{r}_2$ с третьей поверхностью дает третий узел дерева. Используя параметрический вид $\mathbf{r}_2$ в уравнении плоскости, найдем пересечение с третьей поверхностью. Луч

$$x = 5.432 - 1.278t$$
$$y = 3.433 + 0.136t$$

подставим в уравнение плоскости

$$-3x + 9 = 5.432 - 1.278t - 3(3.433 + 0.136t) + 9 = -1.686t + 4.133 = 0$$

Отсюда $t = 2.451$, и точка пересечения

$$x_3 = 5.432 - (1.278)(2.451) = 2.299$$
$$z_3 = 3.433 + (0.136)(2.451) = 3.766$$

Расстояние между двумя точками пересечения есть

$$d_{23} = \sqrt{(x_3 - x_2)^2 + (z_3 - z_2)^2}$$
$$\sqrt{(2.299 - 5.432)^2 + (3.766 - 3.433)^2} = 3.151$$

Рассматривая луч $\mathbf{r}_2$ как падающий, определим отраженный и преломленный лучи

$$\mathbf{v}_3 = \mathbf{r}_2 = -1.278\mathbf{i} + 0.136\mathbf{k}$$

Единичная нормаль к поверхности со стороны падения луча имеет вид

$$\hat{\mathbf{n}}_3 = \frac{\mathbf{i}}{\sqrt{10}} - \frac{3\mathbf{k}}{\sqrt{10}}$$

результате получаем

$$\mathbf{p}_3 = -1.713\mathbf{i} + 0.483\mathbf{k}$$
$$\mathbf{r}_3 = -1.765\mathbf{i} - 1.643\mathbf{k}$$

По ле того и отраженный, и преломленный лучи уходят за пределы сцены, т. е. цепочка оканчивается. Из свойств поверхности видно, что $k_{t_3} = 0$, поэтому она непрозрачна и пропущенного луча нет.

Расчет интенсивности начинается внизу дерева в третьем узле. Сквозь поверхность 3 свет не проходит. Теневой зонд позволяет определить, что между точкой пересечения луча и источником находится сама поверхность, поэтому точка пересечения освещается только рассеянным светом. Его интенсивность равна

$$I_3 = k_{a3} I_a = (0.15)(1) = 0.15$$

Она передается по направлению вектора отражения $\mathbf{r}_2$ ко второй поверхности и ослабляется с расстоянием $d_{23}$:

$$I_{32} = \frac{I_3}{d_{23}} = \frac{0.15}{3.151} = 0.0476$$

Во втором узле дерева, представляющем точку пересечения луча со второй поверхностью, теневой зонд не пересекается ни с одним объектом. Это значит, что точка освещена источником. Вектор из точки к источнику имеет вид

$$\mathbf{L}_2 = (x_l - x_2)\mathbf{i} + (z_l - z_2)\mathbf{k} = (3 - 5.432)\mathbf{i} + (10 - 3.433)\mathbf{k}$$
$$= -2.432\mathbf{i} + 6.567\mathbf{k}$$

$$\hat{\mathbf{L}}_2 = -0.347\mathbf{i} + 0.938\mathbf{k}$$

Следовательно,

$$\hat{\mathbf{n}}_2 \cdot \hat{\mathbf{L}}_2 = \left(\frac{-\mathbf{i}}{\sqrt{2}} + \frac{\mathbf{k}}{\sqrt{2}}\right)(-0.347\mathbf{i} + 0.938\mathbf{k}) = 0.909$$

Направление отражения луча от источника есть

$$\hat{\mathbf{R}}_2 = -0.938\mathbf{i} + 0.347\mathbf{k}$$

Единичный вектор наблюдения равен $-\hat{\mathbf{p}}_1$ и

$$-\hat{\mathbf{p}}_1 \cdot \hat{\mathbf{R}}_2 = (-0.168\mathbf{i} + 1.582\mathbf{k}) \cdot (-0.938\mathbf{i} + 0.347\mathbf{k}) = 0.707$$

Итак,

$$I_2 = k_{a2} I_a + I_l k_{d2}(\hat{\mathbf{n}}_2 \cdot \hat{\mathbf{L}}_2) + I_l k_{s2}(-\hat{\mathbf{p}}_1 \cdot \hat{\mathbf{R}}_2)^n + k_{s2} I_{s2} + k_{t2} I_{t2}$$
$$= (0.15)(1) + (10)(0.15)(0.909) + (10)(0.8)(0) + (0.8)(.0476) + (0.5)(0) = 1.552$$

Эта интенсивность передается вдоль вектора преломления $\mathbf{p}_1$ к первой поверхности. Учитывая ослабление света с расстоянием $d_{12}$, имеем

$$I_{t1} = \frac{I_2}{d_{12}} = \frac{1.552}{4.09} = 0.379$$

теневой зонд опять не пересекается ни с одним объектом, т. е. точка освещена источником. Вектор $\mathbf{L}_1$ от точки к источнику имеет вид

$$\mathbf{L}_1 = (x_l - x_1)\mathbf{i} + (z_l - z_1)\mathbf{k} = (3-5)\mathbf{i} + (10-7.5)\mathbf{k} = -2\mathbf{i} + 2.5\mathbf{k}$$

и

$$\hat{\mathbf{L}}_1 = -0.625\mathbf{i} + 0.781\mathbf{k}$$

ледовательно,

$$\mathbf{n}_1 \cdot \hat{\mathbf{L}}_1 = \left(\frac{\mathbf{i}}{\sqrt{2}} + \frac{\mathbf{k}}{\sqrt{2}}\right) \cdot (-0.625\mathbf{i} + 0.781\mathbf{k}) = 0.110$$

Направление отражения луча от источника представляет собой

$$\hat{\mathbf{R}}_1 = 0.781\mathbf{i} - 0.625\mathbf{k}$$

диничный вектор наблюдения равен $-\hat{\mathbf{v}}_1$ и

$$-\hat{\mathbf{v}}_1 \cdot \hat{\mathbf{R}}_1 = (\mathbf{k}) \cdot (0.781\mathbf{i} - 0.625\mathbf{k}) = -0.625$$

Поэтому

$$I_1 = k_{a1}I_a + I_l k_{d1}(\hat{\mathbf{n}}_1 \cdot \hat{\mathbf{L}}_1) + I_l k_{s1}(-\hat{\mathbf{v}}_1 \cdot \hat{\mathbf{R}}_1)^n + k_{s1}I_{s1} + k_{t1}I_{t1}$$

$$(0.15)(1) + (10)(0.15)(0.11) + (10)(0.8)(0) + (0.8)(0) + (0.5)(0.379) = 0.505$$

ту интенсивность воспринимает наблюдатель. Она невелика, так как поверхность холится под очень острым углом к источнику, и поэтому точка видна не очень четко. Заметим, что первая поверхность пропускает более трети интенсивности падающей от второй. Из-за большого $n$ нет локальных зеркальных бликов.

При построении цветных изображений приведенные расчеты выполняются три раза: отдельно для красной, зеленой и синей компонент, причем каждый раз со своими оптическими характеристиками.

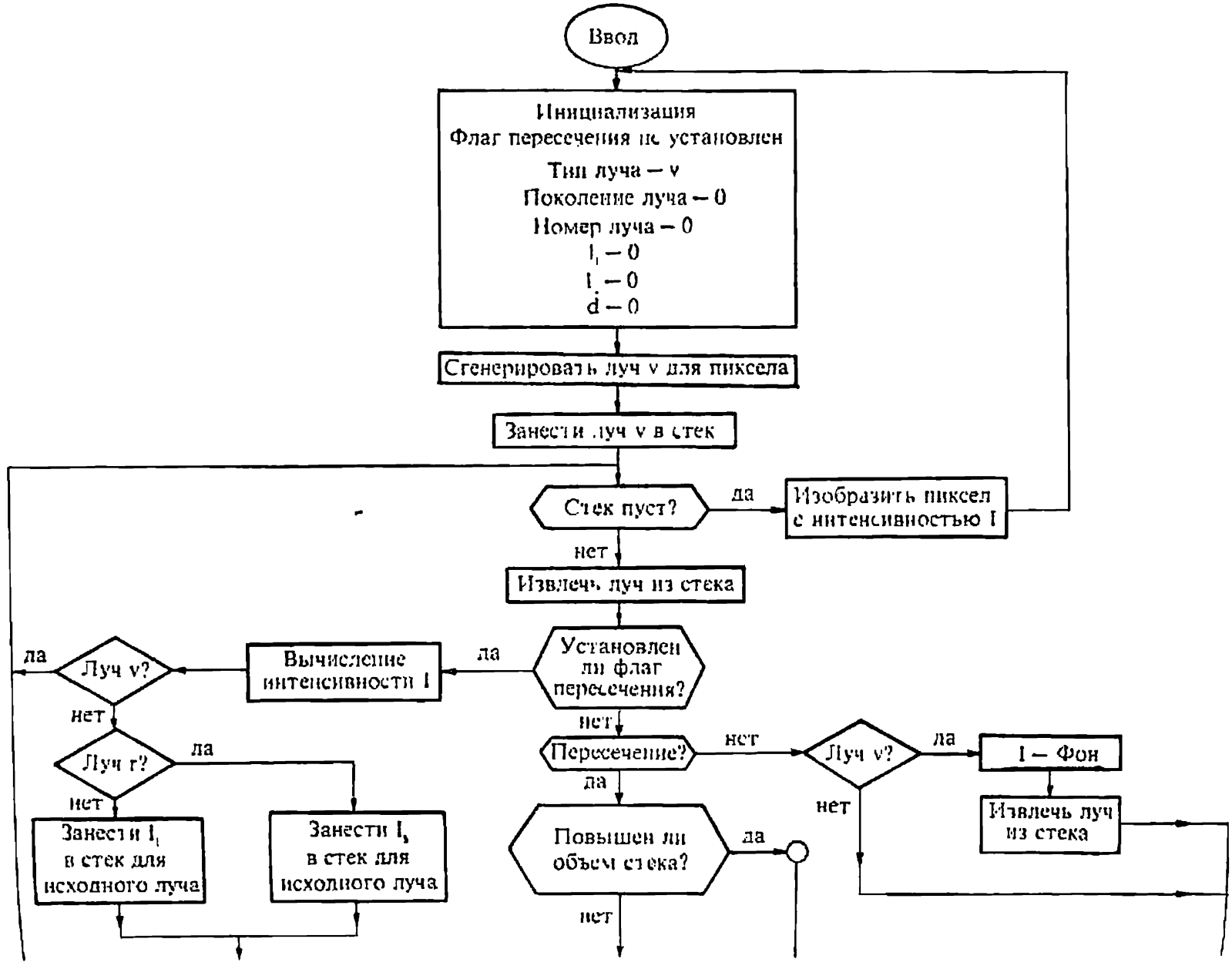
На рис. 5.42 приведена блок-схема алгоритма трассировки луча с использованием глобальной модели освещения. Здесь применяется стек лучей магазинного типа, который служит для передачи информации об отраженном и пропущенном свете между элементами дерева луча. В каждый момент времени стек содержит только часть дерева, поэтому его объем можно ограничить самой длинной предполагаемой ветвью. Ветвь дерева оканчивается, когда и отраженный, и преломленный лучи уходят за пределы сцены или когда переполняется стек. Если оба луча покидают сцену, то их вклад в интенсивность исходного луча равен нулю. Если переполняется стек, то алгоритм рассчитывает интенсивность исходного луча, используя лишь компоненты рассеянного, диффузного и зеркального отражения в точке падения луча. Алгоритм можно дополнить так, чтобы иметь возможность вводить один дополнительный уровень де

рева без переполнения стека. Блок-схема измененного алгоритма дана на рис. 5.43.

Сокращая среднюю длину дерева или стека, т. е. уменьшая объем вычислений, можно повысить эффективность алгоритма. Для этого можно, например, записывать в стек только те лучи, которые значительно влияют на интенсивность света, попадающего в глаз наблюдателя. Максимум относительного вклада конкретного узла в наблюдаемую интенсивность определяется следующим образом. Приближенная интенсивность в первом пересечении луча с поверхностью с учетом затенения вычисляется с использованием лишь локальной модели освещения, например членов, соответствующих рассеянному отражению, ламбертовскому диффузному отражению и зеркальному отражению Фонга в уравнении (5.16). Полученное значение интенсивности запоминается, и в каждом последующем пересечении максимальный вклад интенсивности аппроксимируется этой же моделью, но без учета теней. Получившаяся интенсивность ослабляется совокупными эффектами преломления и отражения и совокупным расстоянием, которое проходит луч от первого пересечения с поверхностью до текущего. Например, приближенную интенсивность на поверхности 3 (рис. 5.40) нужно умножить на $k_{s_2} k_{t_1} / d_{23} d_{12}$ (см. пример 5.9). Если при этом интенсивность превышает некоторый процент интенсивности в первом пересечении, то преломленный и отраженный лучи заносятся в стек, в противном случае в этой точке ветвь оканчивается. Холл [5-34, 5-35], применяя аналогичный метод, обнаружил, что средний размер дерева и объем вычислений уменьшается более чем в 8 раз. Однако этот метод не совсем корректен. Так, если основной вклад в наблюдаемую интенсивность вносится эффектами глобальной модели уже после построения дерева, то изображение получится неполным. Правда, вероятность этого для большинства сцен невелика, и значительная экономия оправдывает такой метод.

Дескриптор объекта похож на аналогичный список из разд. 4.13 для алгоритма выделения видимых непрозрачных поверхностей с помощью трассировки лучей. Стек лучей содержит следующие данные о каждом луче:

| | |
|---|---|
| Номер луча | однозначно определяющий луч |
| Тип луча | **v** — луч, исходящий из глаза наблюдателя и проходящий через пиксел; **r** — отраженный луч; **p** — преломленный луч |
| Номер исходного луча | номер луча, породившего данный луч |


Ввод
Инициализация
Флаг пересечения не установлен
Тип луча – v
Поколение луча – 0
Номер луча – 0
I_l – 0
I – 0
d – 0
Сгенерировать луч v для пиксела
Занести луч v в стек
Стек пуст?
да
Изобразить пиксел с интенсивностью I
нет
Извлечь луч из стека
Установлен ли флаг пересечения?
ла
Вычисление интенсивности I
ла
Луч v?
ла
нет
Луч r?
ла
нет
Занести I_l в стек для исходного луча
Занести I_r в стек для исходного луча
нет
Пересечение?
нет
да
Повышен ли объем стека?
да
нет
Луч v?
ла
нет
I – Фон
Извлечь луч из стека

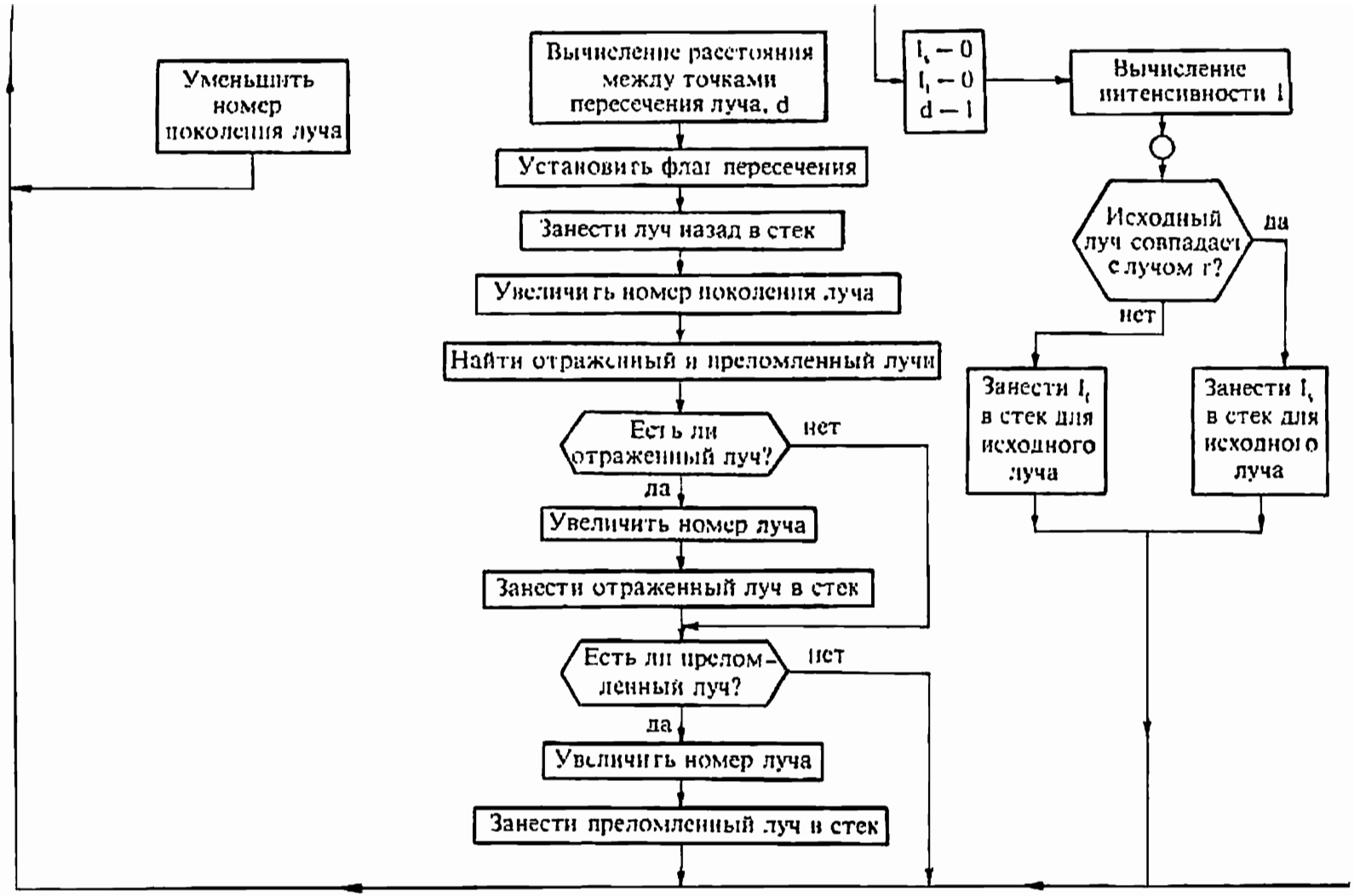



Рис 5.42. Блок-схема алгоритма трассировки луча с использованием глобальной модели освещения.

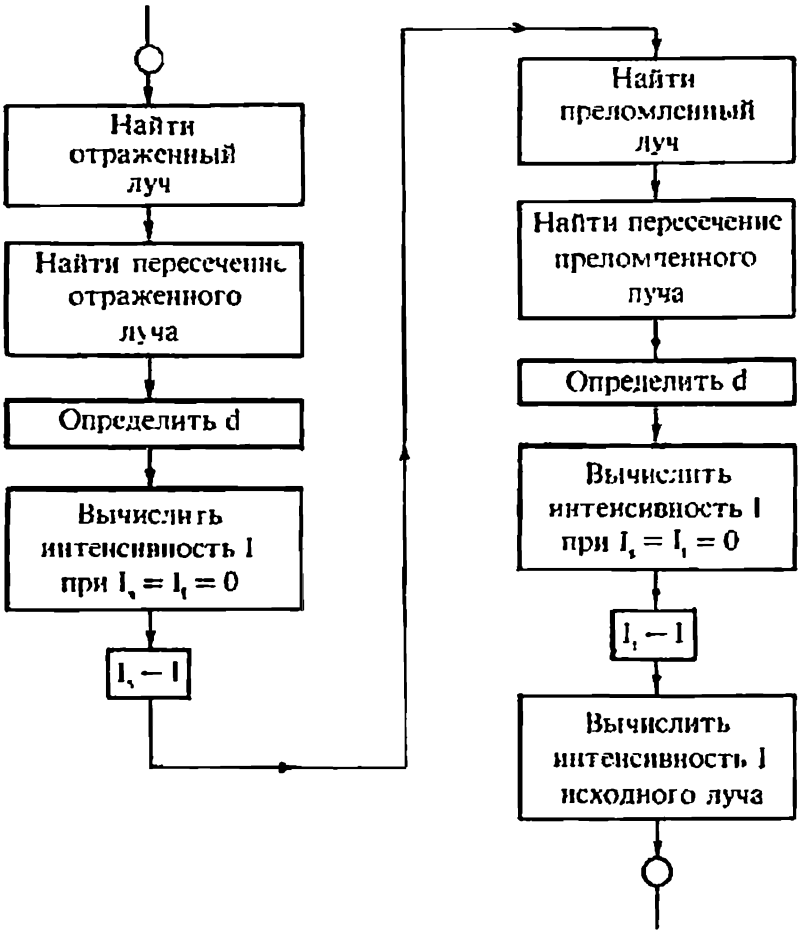



Рис. 5.43. Модификация алгоритма трассировки луча с использованием глобальной модели освещения.

| | |
|---|---|
| Тип исходного луча | **v**, **r** или **p**, как определено выше |
| Флаг пересечения | 1, если у данного луча есть пересечения; 0 в противном случае |
| Указатель объекта | определяет положение объекта, с которым пересекается луч, в дескрипторе объекта |
| Координаты пересечения | $x$, $y$ $z$ — коэффициенты точки, из которой исходит данный луч |
| Направляющие косинусы | определяют направление луча |
| $d$ | расстояние от пересечения исходного луча до пересечения данного |
| $I_t$ | интенсивность пропущенного света в направлении данного луча |
| $I_s$ | интенсивность зеркально отраженного света в направлении данного луча |

Когда луч впервые заносится в стек, значения $I_s$, $I_t$, $d$ и флага пересечения устанавливаются равными нулю. Затем они изменяются в процессе работы алгоритма.

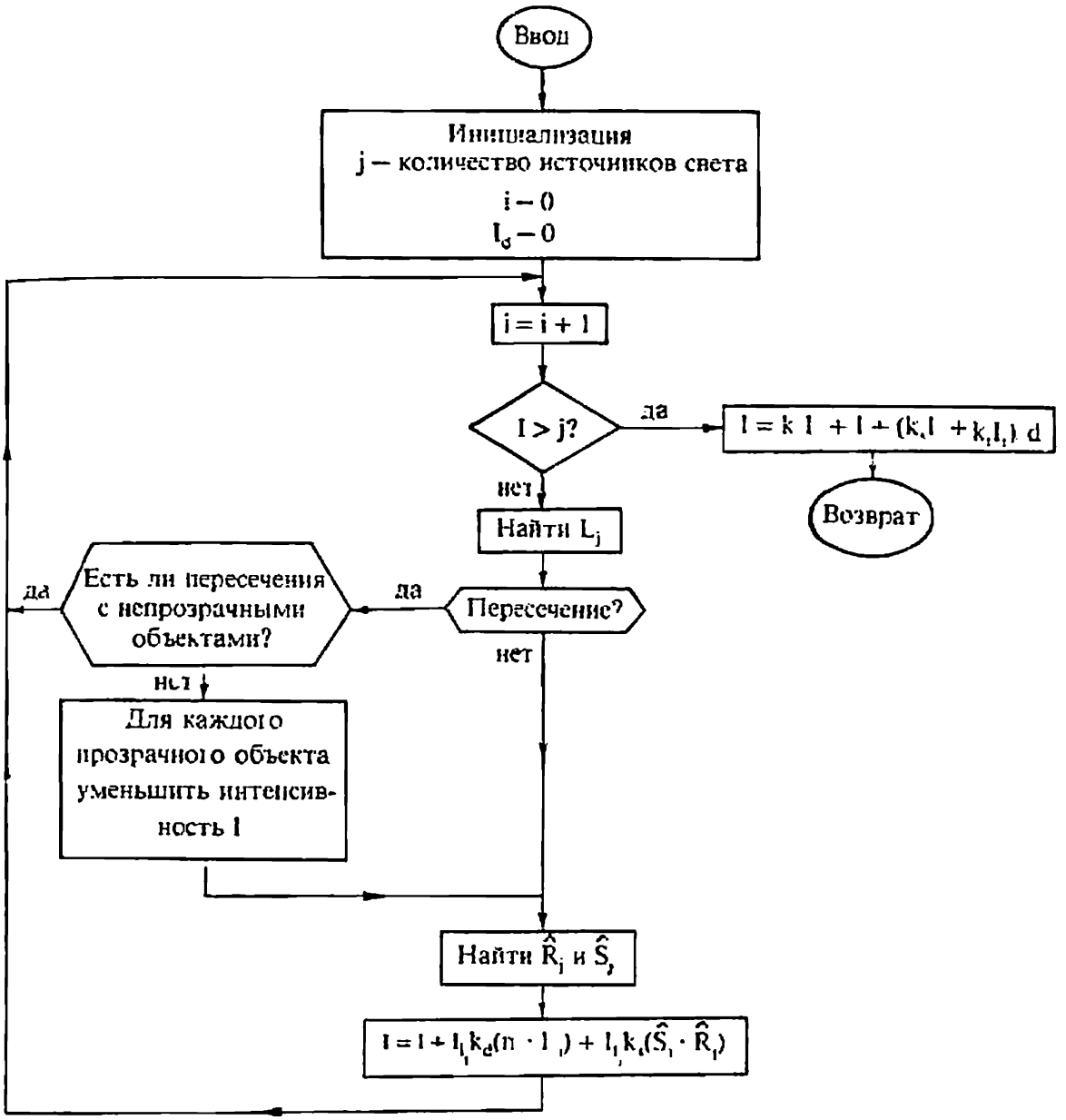



Рис. 5.44. Блок-схема модели освещения для алгоритма трассировки луча с использованием глобальной модели освещения.

В алгоритме применяется модель освещения Уиттеда (рис. 5.39). Блок-схема, показанная на рис. 5.44, соответствует блоку «Вычисление интенсивности I» на рис. 5.42. Если изображение цветное, то этот блок выполняется трижды — для каждого из основных цветов. Векторы от точки пересечения с поверхностью до источников света (теневые зонды) проверяются на пересечение с другими объектами. Если на их пути встречаются непрозрачные объекты, то данный источник не вносит вклад в локальное диффузное или зеркальное отражение в этой точке. Если все объекты, пересекающиеся с направлением теневого зонда, прозрачны, то интенсивность источника $I_l$ соответственно уменьшается. В частности, она зависит от коэффициентов пропускания закрывающих поверхностей. Так,

например, от непрозрачных объектов падает резкая черная тень, а от прозрачных — светлая. Преломление света в прозрачных объектах не учитывается. Пропущенный и зеркально отраженный в точке свет ослабляется с расстоянием между точками пересечения. В алгоритме предполагается, что нормаль к поверхности можно вывести из дескриптора обьекта. Данную программу можно изменить и включить в нее другие, более сложные модели освещения (см. разд. 5.7, 5.8, 5.13).

Процессор пересечений был рассмотрен в разд. 4.13, где он применялся в алгоритме выделения видимых непрозрачных поверхностей с помощью трассировки лучей. В нашем случае необходимо только одно изменение: перед тем как повернуть луч до совпадения с осью $z$, нужно перенести точку его пересечения с поверхностью в начало координат. Луч испускается в направлении $-z$. Таким же образом находятся пересечения теневых зондов с объектами.

Алгоритм (рис. 5.42) начинает работу с построения правой ветви «преломления» от корневого узла до окончания ветви, как указано пунктиром со стрелками на рис. 5.45. Затем при обратном проходе вычисляется интенсивность узлов. После этого строится левая ветвь «отражения» и аналогично находится интенсивность. Процесс может повторяться в каждой промежуточной вершине. На рис. 5.45 стрелки, направленные вниз, соответствуют образованию луча (занесению в стек), а направленные вверх — расчету интенсивности (извлечению из стека). После того как вклад луча в интенсивность вершины определен, луч отбрасывается. Когда остается только корень дерева, вычисление интенсивности пиксела завершается и полученное значение посылается на экран.

Уиттед [5-30] включил в алгоритм трассировки лучей устранение лестничного эффекта. Сильнее всего этот эффект проявляется в

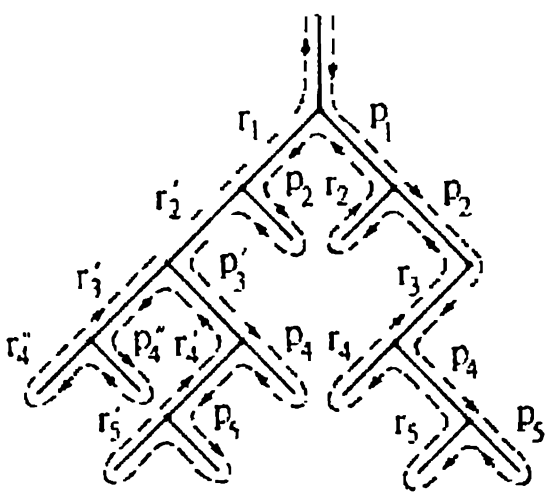



Рис. 5.45. Дерево трассировки луча.

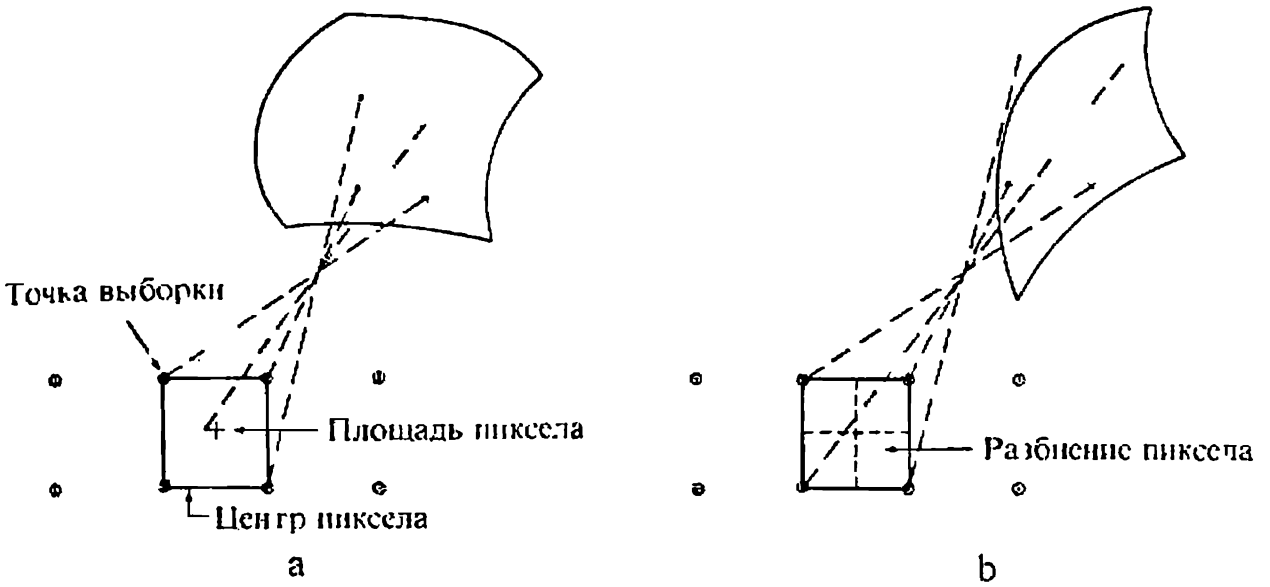



Рис. 5.46. Устранение лестничного эффекта трассировки луча.

областях с большими перепадами интенсивности, например на краях объектов, на контурных ребрах, рисунках фактуры, а также на объектах, меньших, чем расстояние между точками выборки. Для того чтобы уменьшить объем вычислений, в этом методе использовалось динамически вызываемое рекурсивное разбиение Варнока. Вместо трассировки лучей, проходящих через центры пикселов, Уиттед отслеживает лучи, проходящие через точки выборки в углах пиксела (рис. 5,46, а). Для растра размером $n \times m$ необходимо $(n + 1) \times (m + 1)$ таких точек, т. е. увеличение сравнительно небольшое. Если интенсивности в углах пиксела примерно одинаковы и внутри нет других объектов, то интенсивность пиксела считается равной средней интенсивности его углов. Если же интенсивности в углах сильно различаются (рис. 5.46, b), то площадь пиксела разбивается на 4 части и процесс повторяется. Рекурсивное разбиение продолжается до тех пор, пока угловые значения не станут почти равными либо не превысится отведенный объем памяти или разрешение компьютера. Интенсивность каждой части пиксела взвешивается в соответствии с его площадью, и их сумма составляет интенсивность всего пиксела. Хотя данный метод все еще основывается на выборочном анализе, в пределе он эквивалентен сглаживанию по площади (см. разд. 2.26).

Для реализации алгоритма нужно по мере построения изображения последовательно запоминать строки или столбцы (что короче) интенсивностей в точках выборки. Тогда не придется возвращаться или вновь генерировать ранее найденные значения интенсивности. Если пиксел разбивается, в стек заносятся промежуточные значения

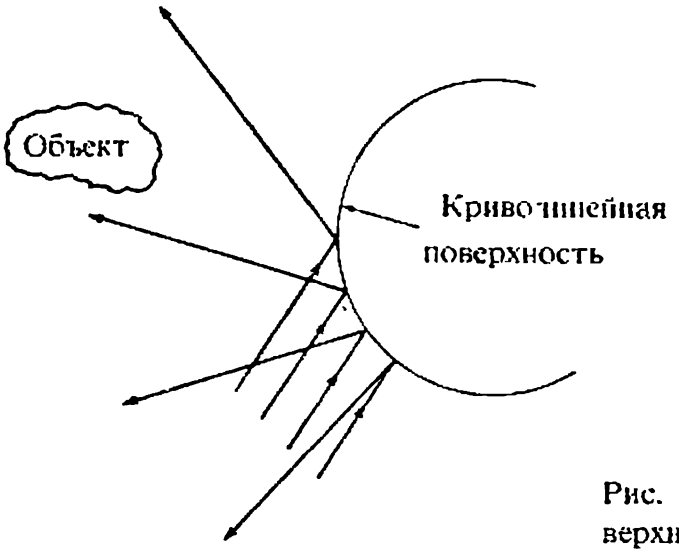



Рис. 5.47. Отражение от криволинейной поверхности.

интенсивности, которые необходимы для следующих разбиений; см. алгоритм Варнока в разд. 4.4.

Для того чтобы не потерять маленьких объектов, Уиттед применяет сферические оболочки минимального размера, который, однако, больше расстояния между точками выборки. Когда алгоритм встречает охватывающую сферу минимального радиуса, а пересечение луча с объектом не найдено, то четыре пиксела, которым принадлежит данная точка выборки, рекурсивно разбиваются до тех пор, пока объект не будет обнаружен. Этот метод годится, если объект наблюдается непосредственно или через плоские поверхности. Однако можно потерять объекты, которые видны косвенно через криволинейные поверхности. При отражении или преломлении в сильно искривленных поверхностях близко лежащие лучи могут разойтись достаточно далеко и пройти мимо объекта, как при отражении от сферы на рис. 5.47. Может случиться, что до того, как последовательность разбиений приведет к пересечению луча с объектом, произойдет выход за пределы разрядной сетки машины.

Этим способом было получено изображение на цветной вклейке 5.

## 5.13. БОЛЕЕ ПОЛНАЯ ГЛОБАЛЬНАЯ МОДЕЛЬ ОСВЕЩЕНИЯ С ТРАССИРОВКОЙ ЛУЧЕЙ

Холл [5-34] и Холл и Гринберг [5-35] рассмотрели более полную модель освещения. В глобальной модели освещения Холла учитывается как рассеивание прямого света от источников в направлении отражения, так и рассеивание в направлении преломления или пропускания. Рассеивание рассчитывается по модели Фонга. Холл использует уравнения Френеля, связывающие длину волны и угол па-

дения преломленного и отраженного света. При пропускании света через прозрачные среды он затухает в соответствии с их характеристиками. Глобальная модель освещения Хопла для $j$ источников света описывается следующим образом:

$$I = k_d \sum_j I_{l_j} R_d(\hat{\mathbf{n}} \cdot \hat{\mathbf{l}}_j) + k_s \sum_j I_{l_j} R_f(\hat{\mathbf{n}} \cdot \hat{\mathbf{H}})^n$$

$$+ k_t \sum_j I_{l_j} T_t(\hat{\mathbf{n}} \cdot \hat{\mathbf{H}}')^{n'} + k_a R_d I_a + k_s R_f I_s T_r^{d_r} + k_t F_t I_t T_t^{d_t}$$

Здесь в члены глобального диффузного рассеивания ($k_a R_d I_a$) и ламбертовского диффузного отражения прямого света от источников входит $R_d(\lambda)$ — зависимость диффузного отражения от свойств вещества и длины волны. Аналогично в член зеркального отражения прямого света от источников и член глобального зеркального отражения входит $R_f(\lambda)$ — зависимость френелевского коэффициента отражения от свойств вещества и длины волны (см. разд. 5.8). Третий член уравнения (5.17), соответствующий направленному пропусканию света после преломления, и член глобального пропускания включают $F_t$ — зависимость френелевского коэффициента пропускания от вещества и длины волны. Из закона сохранения энергии имеем $F_t = 1 - R_f$. Величины $R_f(\lambda)$ и $F_t(\lambda)$ определяются с помощью приближенного метода Кука, описанного в разд. 5.8. В члены глобального зеркального отражения и преломления также входят $T_r$ и $T_t$ — коэффициенты пропускания в расчете на единицу длины для отраженного и пропущенного (преломленного) лучей. Расстояния, пройденные отраженным и пропущенным лучом от последнего пересечения, обозначены соответственно $d_r$ и $d_t$. Для того чтобы смоделировать пропускание света сквозь вещество, $T_r$ и $T_t$ возведены в степень, как у Кэя (см. разд. 5.9). В качестве показателей степеней берутся расстояния $d_r$ и $d_t$.

Член зеркального отражения для света, падающего непосредственно от источников, взят из модели Торрэнса — Спэрроу (разд. 5.8). Угол между нормалью к поверхности $\mathbf{n}$ и биссектрисой угла между направлением на источник и направлением на наблюдателя $\mathbf{H}$, т. е. $\mathbf{n} \cdot \mathbf{H}$, возведенный в степень $n$, используется для представления рассеивания зеркально отраженного света. Аналогично угол между нормалью к поверхности и вектором $\mathbf{H}'$, возведенный в степень $n'$, используется для представления рассеивания направленно пропущенного света. Вектор $\mathbf{H}'$ — нормаль к микрограням поверхности из модели Торрэнса — Сперроу (см. разд. 5.8),

которые отражают свет, падающий непосредственно от источника в направлении **p** (рис. 5.39).

Направление вектора **H′** можно найти по закону Снеллиуса (см. разд. 5.9). С помощью подобных треугольников на рис. 5.48 получаем

$$\mathbf{ad} = \frac{\eta_2}{\eta_1}\mathbf{bd}$$

У нас

$$\mathbf{ab} = \mathbf{v} - \mathbf{p}$$

и

$$\mathbf{ad} = \mathbf{ab} + \mathbf{bd}$$

поэтому

$$\mathbf{bd} = \frac{\mathbf{v} - \mathbf{p}}{\eta_2/\eta_1 - 1}$$

Так как

$$\mathbf{H}' = \mathbf{bd} - \mathbf{p}$$

то, объединяя полученные результаты, имеем

$$\mathbf{H}' = \frac{\mathbf{v} - (\eta_2/\eta_1)\mathbf{p}}{\eta_2/\eta_1 - 1}$$

На цветной вклейке 8 сравниваются глобальные модели освещения Уиттеда и Холла. На обеих картинках изображена одна и та же сцена, построенная методом трассировки лучей: на верхней картинке — с помощью глобальной модели освещения Уиттеда, описываемой уравнением (5.16), на нижней — с помощью глобальной модели освещения Холла, описываемой (5.17). Сравните изображения металлических шаров в обеих сценах. Обратите внимание на цвет отраженного в них голубого края коврика. Сравните цвета прозрачных шаров: шар на нижнем рисунке и его тень отличаются

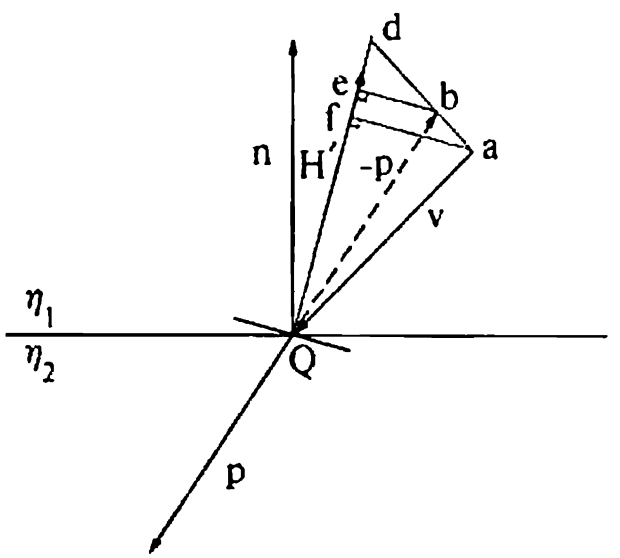



Рис. 5.48. Определение вектора H′.

легким сине-зеленым оттенком. Он появляется благодаря тому, что в модели Холла учитывается влияние вещества на пропущенный свет. Эта модель получена эмпирически, и, как указал Холл, с научной точки зрения не совсем корректна [5-34]. Однако, как видно на вклейке 8, она создает высокореалистичные изображения, одни из лучших, полученных к настоящему времени.

## 5.14. НАПРАВЛЕНИЯ СОВРЕМЕННЫХ ИССЛЕДОВАНИЙ

Рассмотренные в предыдущих разделах методы представляют мощный инструмент построения синтетических изображений, но в некоторых случаях недостаточно совершенный, например при изображении природных объектов, моделировании распределенных источников света, а также при передаче движения с учетом лестничного эффекта. Эти области продолжают разрабатываться, и их подробное обсуждение выходит за рамки данной книги. Однако можно указать некоторые направления современных исследований в этой области.

Свойства обычного точечного источника, используемого в моделях освещения, существенно определяются корпускулярной природой света. Каждый луч должен трассироваться индивидуально, но для огромного количества лучей, порождаемых источником рассеянного света и (или) диффузным отражением от поверхности, требуются недопустимо большие вычислительные затраты. Моравек [5-36] предлагает решение, основанное на волновых свойствах света. Предварительные результаты довольно интересны, но опять же необходимы значительные вычислительные ресурсы.

Эффект прерывистости в движущихся изображениях появляется в двух основных случаях. Первый — это обычное квантование; например, когда в последовательности кадров маленькие объекты то появляются, то исчезают или когда начинают «дрожать» контуры изображения. Большинство таких явлений можно устранить обычным пространственным сглаживанием каждого отдельного кадра.

Ко второму случаю относится временная прерывистость или эффект смазывания при движении. Объект, перемещающийся с большей скоростью, кажется слегка размытым. Исследованию этого вопроса посвящены три новые работы. Корейн и Бадлер [5-37] рассматривают метод создания синхронизированных во времени и в пространстве изображений движущегося объекта на одном кадре. Потметсил и Чакраварти [5-38] усовершенствовали свою модель фотокамеры [5-31], чтобы учесть эффект смазывания. Для этого

отдельные объекты дефокусируются, и на одном кадре объединяется множество изображений (экспозиций). Ривз [5-39] включил смазывание в стохастическую систему частиц путем построения специальных форм частиц.

Природные объекты трудно изображать из-за того, что они сложны и нерегулярны. Они могут быть неровными, грязными, растрескавшимися. Примерами таких объектов являются огонь, дым, облака, туман, трава, деревья. Наиболее интересны в этой области работы Блинна [5-40], Дунгана [5-41], Маршала, Уилсона и Карлсона [5-42], Ксури [5-43].

Система частиц Ривза [5-39] привлекает особое внимание. Многие природные объекты трудно смоделировать полигональными или криволинейными поверхностями, поэтому Ривз и его коллеги, а также ряд предыдущих исследователей в качестве механизма моделирования стали использовать отдельные «нечеткие» частицы, имеющие неправильную и нерегулярную форму. Под воздействием физических и стохастических процессов форма и свойства частиц с течением времени меняются. Они создаются (рождаются), перемещаются внутри системы, умирают или покидают систему. Для того чтобы построить кадр с помощью системы частиц, применяется следующая процедура:

новые частицы создаются, им приписываются свои индивидуальные атрибуты и они вводятся в систему;
старые, умершие, частицы исключаются из системы;
оставшиеся частицы перемещаются по определенным законам движения;
строится изображение оставшихся частиц.

Этим методом были получены некоторые из наиболее реалистических изображений. Пример приведен на цветной вклейке 10.

## 5.15. ЦВЕТ

Цвет уже не раз упоминался в данной главе и сейчас будет рассмотрен подробнее. Он имеет как психофизиологическую, так и психофизическую природу. Восприятие цвета зависит от физических свойств света, т. е. электромагнитной энергии, от его взаимодействия с физическими веществами, а также от их интерпретации зрительной системой человека. Эта проблема чрезвычайно широка, сложна и интересна, и ее подробное изучение выходит за пределы нашей книги. Дополнительная информация дана в работах с [5-44]

по [5-47]. Мы рассмотрим наиболее важные понятия, основы связанных с цветом физических явлений, систем представления цвета и преобразований между ними.

Зрительная система человека воспринимает электромагнитную энергию с длинам волн от 400 до 700 нм как видимый свет (1 нм = $10^{-9}$ м). Свет принимается либо непосредственно от источника, например электрической лампочки, либо косвенно при отражении от поверхности объекта или преломлении в нем.

Источник или объект является ахроматическим, если наблюдаемый свет содержит все видимые длины волн в приблизительно равных количествах. Ахроматический источник кажется белым, а отраженный или преломленный ахроматический свет — белым, черным или серым. Белыми выглядят объекты, ахроматически отражающие более 80% света белого источника, а черными — менее 3%. Промежуточные значения дают различные оттенки серого. Интенсивность отраженного света удобно рассматривать в диапазоне от 0 до 1, где 0 соответствует черному, 1 — белому, а промежуточные значения — серому цвету.

Хотя трудно определить различие между светлотой и яркостью, светлота обычно считается свойством несветящихся или отражающих объектов и изменяется от черного до белого, а яркость является свойством самосветящихся или излучающих объектов и изменяется в диапазоне от низкой до высокой.

Светлота или яркость объекта зависит от относительной чувствительности глаза к разным длипам волн. Из рис. 5.59 видно, что при дневном свете чувствительность глаза максимальна при длине

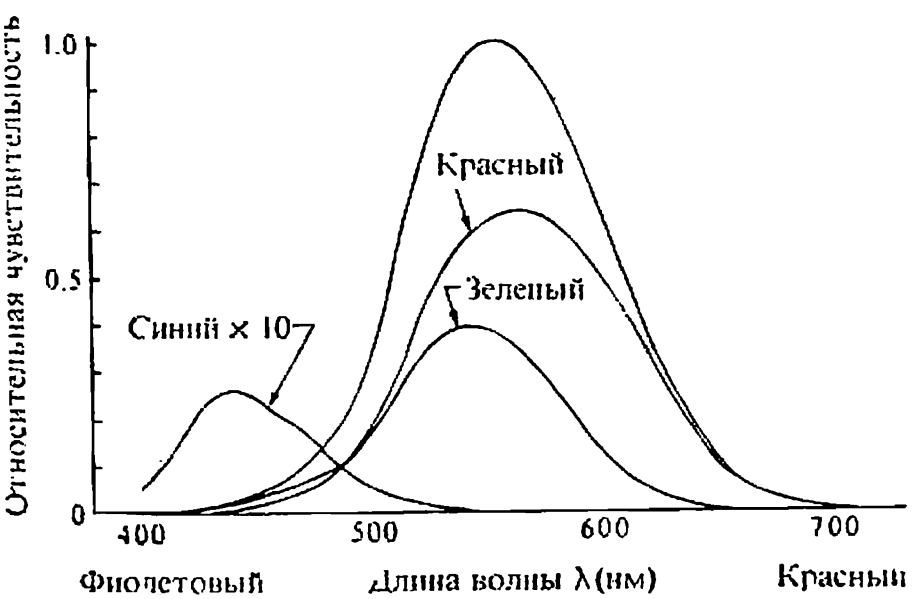



Рис. 5.49. Относительная чувствительность глаза.

волны порядка 550 нм, а на краях видимого диапазона спектра она резко падает. Кривая на рис. 5.49 называется функцией спектральной чувствительности глаза. Это мера световой энергии или интенсивности с учетом свойств глаза.

Если воспринимаемый свет содержит длины волн в произвольных неравных количествах, то он называется хроматическим[1]. Если длины волн сконцентрированы у верхнего края видимого спектра, то свет кажется красным или красноватым, т. е. доминирующая длина волны лежит в красной области видимого спектра. Если длины волн сконцентрированы в нижней части видимого спектра, то свет кажется синим или голубоватым, т. е. доминирующая длина волны лежит в синей части спектра. Однако сама по себе электромагнитная энергия определенной длины волны не имеет никакого цвета. Ощущение цвета возникает в результате преобразования физических явлений в глазу и мозге человека. Цвет объекта зависит от распределения длин волн источника света и от физических свойств объекта. Объект кажется цветным, если он отражает или пропускает свет лишь в узком диапазоне длин волн и поглощает все остальные. При взаимодействии цветов падающего и отраженного или пропущенного света могут получиться самые неожиданные результаты. Например, при отражении зеленого света от белого объекта и свет, и объект кажутся зелеными, а если зеленым светом освещается красный объект, то он будет черным, так как от него свет вообще не отражается.

Психофизиологическое представление света определяется цветовым тоном, насыщенностью и светлотой. Цветовой тон позволяет различать цвета, а насыщенность — определять степень ослабления (разбавления) данного цвета белым цветом. У чистого цвета она равна 100% и уменьшается по мере добавления белого. Насыщенность ахроматического цвета составляет 0%, а его светлота равна интенсивности этого света.

Психофизическими эквивалентами цветового тона, насыщенности и светлоты являются доминирующая длина волны, чистота и яркость. Электромагнитная энергия одной длины волны в видимом спектре дает монохроматический цвет. На рис. 5.50,а изображено распределение энергии монохроматического света с длиной волны 525 нм, а на рис. 5.50,b — для «белого» света с энергией $E_2$ и од-

---

[1] Основное значение имеют слова «воспринимаемый» и «произвольные». Из дальнейшего будет видно, что некоторые смеси хроматических цветов могут восприниматься как ахроматические цвета.

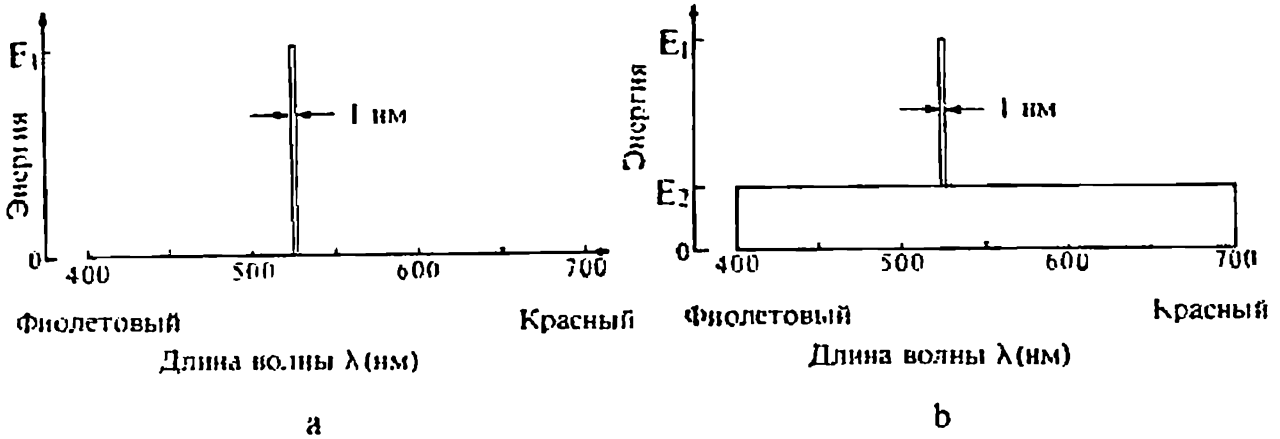



Рис. 5.50. Длины волн света.

ной доминирующей длины волны 525 нм с энергией $E_1$. На рис. 5.50,b цвет определяется доминирующей длиной волны, а чистота — отношением $E_1$ и $E_2$. Значение $E_2$ — это степень разбавления чистого цвета с длиной волны 525 нм белым: если $E_2$ приближается к нулю, то чистота цвета приближается к 100%, а если $E_2$ приближается к $E_1$, то свет становится близким к белому и его чистота стремится к нулю. Яркость пропорциональна энергии света и рассматривается как интенсивность на единицу площади.

Обычно встречаются не чистые монохроматические цвета, а их смеси. В основе трехкомпонентной теории света служит предположение о том, что в центральной части сетчатки находятся три типа чувствительных к цвету колбочек. Первый воспринимает длины волн, лежащие в середине видимого спектра, т. е. зеленый цвет; второй — длины волн у верхнего края видимого спектра, т. е. красный цвет; третий — короткие волны нижней части спектра, т. е. синий. Относительная чувствительность глаза (рис. 5.49) максимальна для зеленого цвета и минимальна для синего. Если на все три типа колбочек воздействует одинаковый уровень энергетической яркости (энергия в единицу времени), то свет кажется белым. Естественный белый свет содержит все длины волн видимого спектра; однако ощущение белого света можно получить, смешивая любые три цвета, если ни один из них не является линейной комбинацией двух других. Это возможно благодаря физиологическим особенностям глаза, содержащего три типа колбочек. Такие три цвета называются основными.

В машинной графике применяются две системы смешения основных цветов: аддитивная — красный, зеленый, синий (RGB) и субтрактивная — голубой, пурпурный, желтый (CMY). Они изобра-

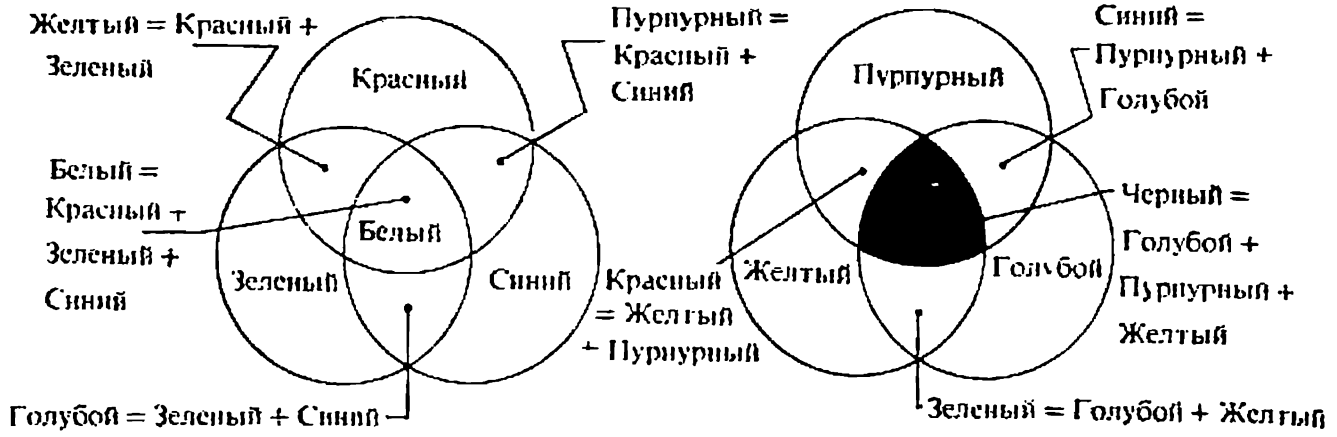



Рис. 5.51. Аддитивная (a) и субтрактивная (b) системы смешения цветов.

жены на рис. 5.51 и цветной вклейке 11. Цвета одной системы являются дополнительными к другой: голубой — к красному, пурпурный — к зеленому, желтый — к синему. Дополнительный цвет — это разность белого и данного цвета: голубой это белый минус красный, пурпурный — белый минус зеленый, желтый — белый минут синий. Хотя красный можно считать дополнительным к голубому, по традиции красный, зеленый и синий считаются основными цветами, а голубой, пурпурный, желтый — их дополнениями. Интересно, что в спектре радуги или призмы пурпурного цвета нет, т. е. он порождается зрительной системой человека.

Для отражающих поверхностей, например типографских красок, пленок и несветящихся экранов применяется субтрактивная система CMY. В субтрактивных системах из спектра белого цвета вычитаются длины волны дополнительного цвета. Например, при отражении или пропускании света сквозь пурпурный объект поглощается зеленая часть спектра. Если получившийся свет отражается или преломляется в желтом объекте, то поглощается синяя часть спектра и остается только красный цвет. После его отражения или преломления в голубом объекте цвет становится черным, так как при этом исключается весь видимый спектр (см. цветную вклейку 11). По такому принципу работают фотофильтры.

Аддитивная цветовая система RGB удобна для светящихся поверхностей, например экранов ЭЛТ или цветных ламп. Достаточно провести очень простой опыт, чтобы убедиться, что минимальное количество цветов для уравнения (составления) практически всех цветов видимого спектра равно трем. Пусть на некоторый фон падает произвольный монохроматический контрольный свет. Наблюдатель пытается опытным путем уравнять на фоне рядом с кон-

трольным светом его цветовой фон, насыщенность и светлоту при помощи монохроматических потоков света разной интенсивности. Если используется только один инструментальный (уравнивающий) цвет, то длина волны у него должна быть такой же, как у контрольного. С помощью одного монохроматического инструментального потока света можно уравнять лишь один цвет. Однако, если не принимать в расчет цветовой тон и насыщенность контрольного света, можно уравнять цвета по светлоте. Эта процедура называется фотометрией. Таким способом создаются монохроматические репродукции цветных изображений.

Если в распоряжении наблюдателя есть два монохроматических источника, то он может уравнять большее количество контрольных образцов, но не все. Добавляя третий инструментальный цвет, можно получить почти все контрольные варианты, при условии, что эти три цвета достаточно широко распределены по спектру и ни один из них не является линейной комбинацией других, т. е. что это основные цвета. Хороший выбор — когда первый цвет лежит в области спектра с большими длинами волн (красный), второй — со средними (зеленый) и третий — с меньшими длинами волн (синий). Объединение этих трех цветов для уравнивания монохроматического контрольного цвета математически выражается как

$$C = rR + gG + bB$$

где $C$ — цвет контрольного света; $R$, $G$, $B$ — красный, зеленый и синий инструментальные потоки света; $r$, $g$, $b$ — относительные количества потоков света $R$, $G$, $B$ со значениями в диапазоне от 0 до 1.

Однако сложением трех основных цветов можно уравнять не все контрольные цвета. Например, для получения сине-зеленого цвета наблюдатель объединяет синий и зеленый потоки света, но их сумма выглядит светлее, чем образец. Если же с целью сделать его темнее добавить красный, то результат будет светлее, потому что световые энергии складываются. Это наводит наблюдателя на мысль: добавить красный свет в образец, чтобы сделать его светлее. Такое предположение действительно срабатывает, и уравнивание завершается. Математически добавление красного света к контрольному соответствует *вычитанию* его из двух других уравнивающих потоков света. Конечно, физически это невозможно, потому что отрицательной интенсивности света не существует. Математически это записывается как

$$C + rR = gG + bB$$

или

$$C = -rR + gG + bB$$

На рис. 5.52 показаны функции $r$, $g$, $b$ уравнения по цвету для монохроматических потоков света с длинами волн 436, 546 и 700 нм. С их помощью можно уравнять все длины волн видимого спектра. Обратите внимание, что при всех длинах волн, кроме окрестности 700 нм, одна из функций всегда отрицательна. Это соответствует «добавлению» инструментального света к контрольному. Изучением этих функций занимается колориметрия.

Наблюдатель также замечает, что при удвоении интенсивности контрольного света интенсивность каждого инструментального потока света также удваивается, т. е. $2C = 2rR + 2gG + 2bB$. Наконец, оказывается, что один и тот же контрольный свет уравнивается двумя разными способами, причем значения $r$, $g$ и $b$ могут быть неодинаковыми. Инструментальные цвета для двух разных наборов $r$, $g$ и $b$ называются метамерами друг друга. Технически это означает, что контрольный свет можно уравнять различными составными источниками с неодинаковым спектральным распределением энергии. На рис. 5.53 изображены два сильно отличающихся спектральных распределения коэффициента отражения, которые дают одинаковый средне-серый цвет.

Результаты проведенных опытов обобщаются в законах Грассмана [5-44]:

Глаз реагирует на три различных стимула, что подтверждает трехмерность природы цвета. В качестве стимулов можно рас-

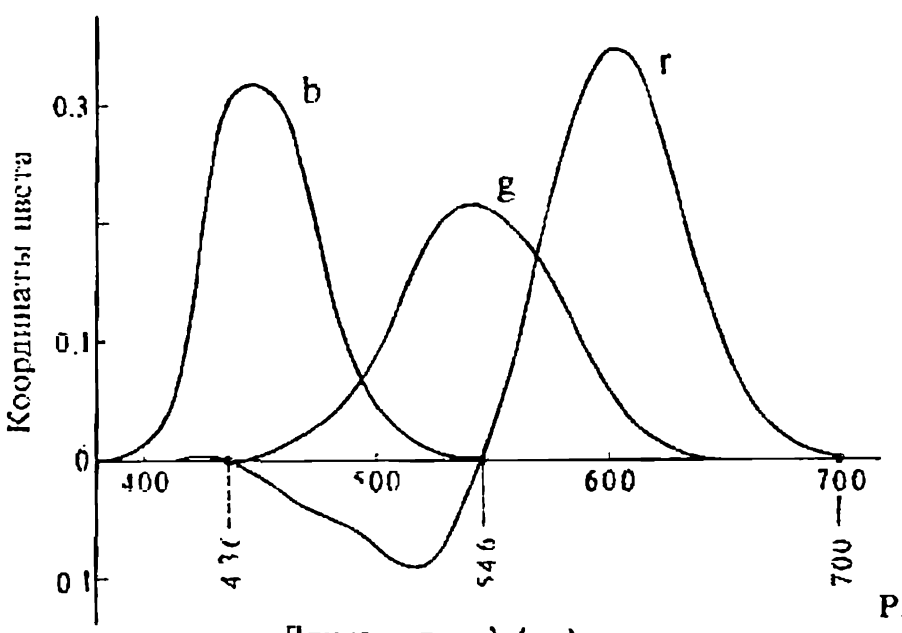



Рис. 5.52. Функции уравновешивания по цвету.

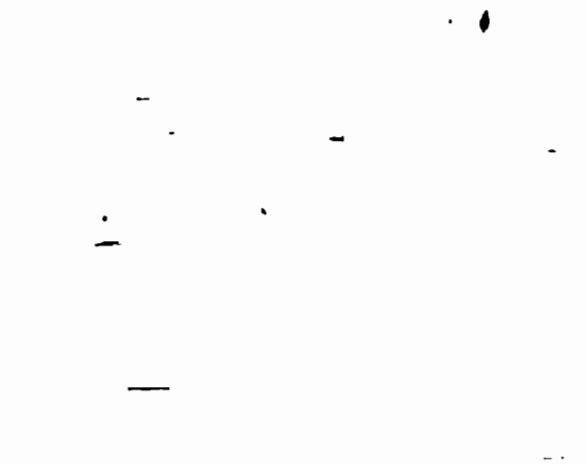
**1. Шевроле Камаро 1983.** Изображение построено Д.Варном с помощью алгоритма Уоткинса и модели освещения со специальными эффектами; см. разд. 5.7. (С разрешения General Motors Research Laboratory.)

**2. Бронзовые вазы.** Верхняя сделана из пластмассы бронзового цвета, а нижняя — из бронзы. Каждая ваза освещается двумя источниками. Изображение построено с помощью модели освещения Кука—Торрэнса; см. разд. 5.8. (С разрешения Р.Кука, Program of Computer Graphics, Корнеллский университет.)

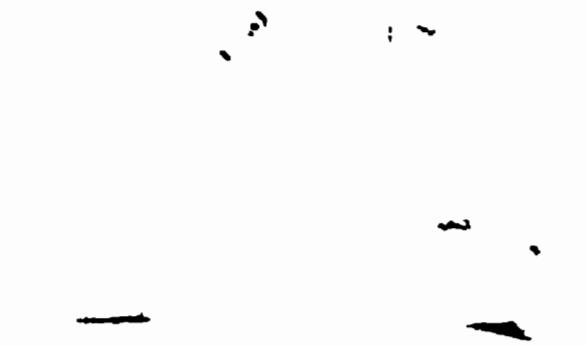
**3. Тени** для случая двух источников, построенные с помощью алгоритма Вэйлера — Азертона; см. разд. 5.10. (С разрешения П.Азертона, Program of Computer Graphics, Корнеллский университет.)

**4. Фрактальные поверхности.** Верхнее изображение состоит из 16 384 фрактальных треугольников, а нижнее — из 262 144. Обратите внимание на собственные тени от источника света справа. (С разрешения Дж. Кадзия, Cal Tech.)

**5. Шары** над красно-желтой шахматной доской изображены при помощи алгоритма трассировки лучей и глобальной модели освещения, учитывающей отражение, тени и преломление в прозрачных объектах; см. разд. 5.12. (С разрешения Т.Уиттеда, Bell Laboratories, перепечатано с разрешения Communications of the ACM, v. 23, июнь 1980, Copyright 1980, Association for Computing Machinery.)



**6. Вазы.** Непрозрачная и прозрачная вазы изображены при помощи алгоритма трассировки лучей. В каждом случае правое изображение учитывает ограниченность глубины резкости; см. [5-32]. (С разрешения М.Потмисила, Image Processing Laboratory, Rensselaer Polytechnic Institute.)



**7. Сферы и цилиндры** с отражением и преломлением построены при помощи алгоритма трассировки лучей и модели освещения Уиттеда; см. разд. 5.12. (С разрешения Аль Барра Raster Technologies, Inc.)

**8. Натюрморт.** Верхнее изображение построено при помощи модели освещения Уиттеда, нижнее — при помощи модели освещения Холла; см. разд. 5.13. Обратите внимание на сферы. (С разрешения Р.А.Холла, Program of Computer Graphics, Корнеллский университет.)

**9. Шахматные фигуры** изображены при помощи алгоритма построчного сканирования и комбинирования отдельных изображений шахматных фигур, их отражений и доски. (С разрешения Т.Уиттеда, Bell Laboratories, перепечатано с разрешения TOG, v.1, январь 1982, Copyright 1982, Association for Computing Machinery.)

**10. Белые пески.** Художник Элви Рэй Смит на основе математических разработок П.Ходжведж создал алгоритм, с помощью которого из одного элемента были выращены эти цветы. Для изображения травы использовалась система частиц Бена Ривза. Удаление скрытых граней проводил Л.Карпентер, математическое обеспечение композиции принадлежит Т.Портеру. (С разрешения Элви Рэя Смита, Lucasfilm LTD, all rights reserved.)

**11. Аддитивная** (сверху) и субтрактивная (снизу) цветовые системы. (С разрешения Polaroid Corporation.)

(a)

(b)

(c)

**12. Диметрическая** (a ортогональная (b) и изометрическая (c) проекции трех блоков, построенные Дж.О.Дженкинсом из лаборатории прикладной физики Университета Дж.Гопкинса с использованием алгоритма Уоткинса, который был разработан как часть учебного проекта.

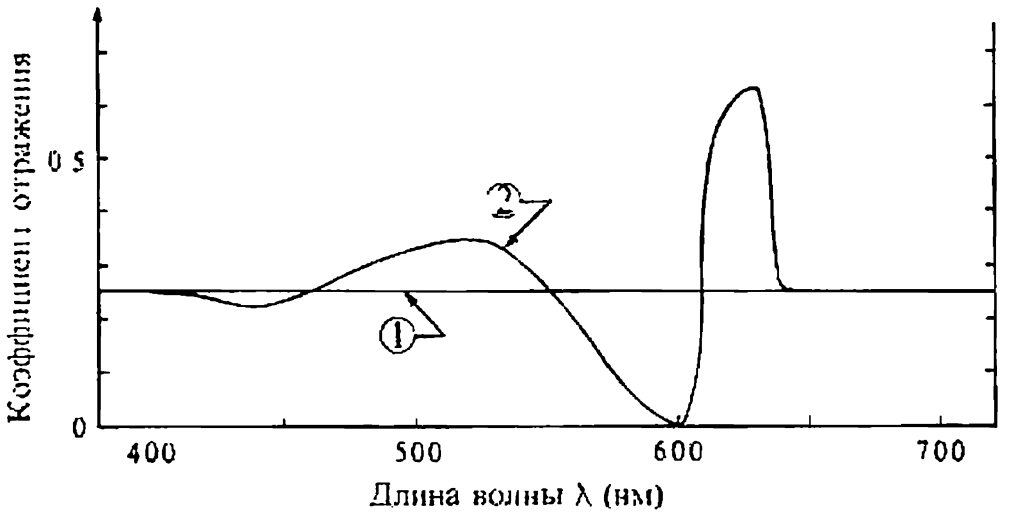



Рис. 5.53. Метамеры.

сматривать, например, доминирующую длину волны (цветовой фон), чистоту (насыщенность) и яркость (светлоту) или красный, зеленый и синий цвета.

Четыре цвета всегда линейно зависимы, т. е. $cC = rR + gG + bB$, где $c,\ r,\ g,\ b \neq 0$. Следовательно, для смеси двух цветов $(cC)_1$ и $(cC)_2$ имеет место равенство $(cC)_1 + (cC)_2 = (rR)_1 + (rR)_2 + (gG)_1 + (gG)_2 - (bB)_1 + (bB)_2$. Если цвет $C_1$ равен цвету $C$ и цвет $C_2$ равен цвету $C$, то цвет $C_1$ равен цвету $C_2$ независимо от структуры спектров энергии $C,\ C_1, C_2$.

Если в смеси трех цветов один непрерывно изменяется, а другие остаются постоянными, то цвет смеси будет меняться непрерывно, т. е. трехмерное цветовое пространство непрерывно.

Из опытов, подобных данному, известно, что зрительная система способна различать примерно 350 000 цветов. Если цвета различаются только по тонам, то в сине-желтой части спектра различными оказываются цвета, у которых доминирующие длины волн отличаются на 1 нм, в то время как у краев спектра — на 10 нм. Четко различимы примерно 128 цветовых тонов. Если меняется только насыщенность, то зрительная система способна выделить уже не так много цветов. Существует 16 степеней насыщенности желтого и 23 — красно-фиолетового цвета.

Трехмерная природа света позволяет отобразить значение каждого из стимулов на оси ортогональной системы (рис. 5.54, а). При этом получается трехкомпонентное цветовое пространство. Любой цвет $C$ можно представить как вектор с составляющими $rR$, $gG$ и $bB$. Подробное описание трехмерного цветового пространства дано в работе Мейера [5-48]. Пересечение вектора $C$ с единичной пло-

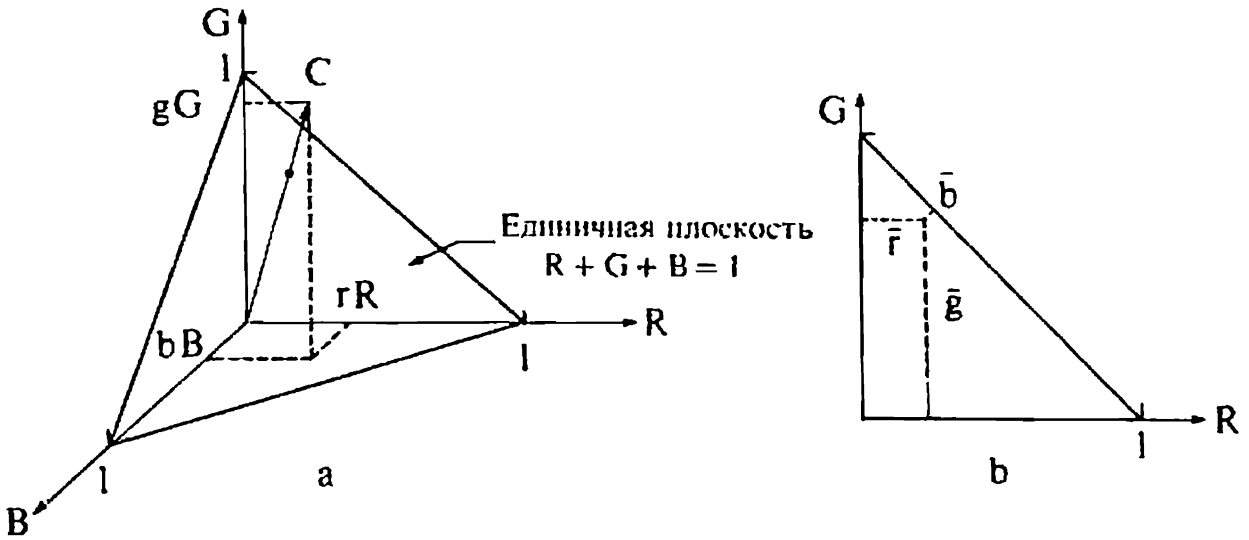



Рис. 5.54. Трехмерное цветовое пространство.

скостью дает относительные веса его красной, зеленой и синей компонент. Они называются значениями или координатами цветности:

$$\bar{r} = \frac{r}{r+g+b} \qquad \bar{g} = \frac{g}{r+g+b} \qquad \bar{b} = \frac{b}{r+g+b}$$

Следовательно, $\bar{r} + \bar{g} + \bar{b} = 1.0$. Проецируя единичную плоскость, получаем цветовой график (рис. 5.54, b). Он явно отображает функциональную связь двух цветов и неявно — связь с третьим, например $\bar{b} = 1 - \bar{r} - \bar{g}$. Если функции уравнивания по цвету (рис. 5.52) перенести в трехмерное пространство, то результат не будет целиком лежать в положительном октанте. В проекции на плоскость также будут присутствовать отрицательные значения, а это осложняет математические расчеты.

В 1931 г. в Англии состоялось заседание Международной комиссии по освещению (МКО) (Commission International de l'Eclairage), на котором обсуждались международные стандарты определения и измерения цветов. В качестве стандарта был выбран двумерный цветовой график МКО 1931 г. и набор из трех функций реакции глаза, позволяющий исключить отрицательные величины и более удобный для обработки. Основные цвета МКО получены из стандартных функций реакции глаза (рис. 5.55) и приведены в таблице в [5-44]. Гипотетические основные цвета МКО обозначаются $X$, $Y$, $Z$. На самом деле они не существуют, так как без отрицательной части они не могут соответствовать реальному физическому свету. Треугольник $XYZ$ был выбран так, чтобы в него входил весь видимый спектр. Координаты цветности МКО таковы:

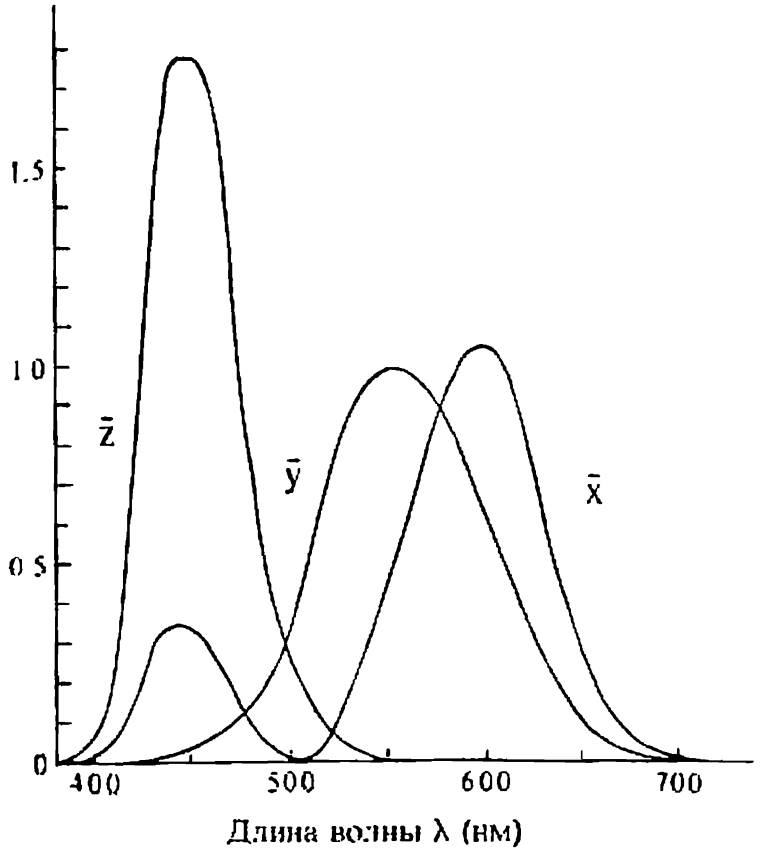



Рис. 5.55. Стандартный колориметрический наблюдатель МКО 1931 г.

$$x = \frac{X}{X+Y+Z} \qquad y = \frac{Y}{X+Y+Z} \qquad z = \frac{Z}{X+Y+Z} \tag{5.18}$$

и $x + y + z = 1$. При проекции треугольника $XYZ$ на плоскость $xy$ получается цветовой график МКО. Координаты цветности $x$ и $y$ представляют собой относительные количества трех основных цветов $XYZ$, требуемые для составления любого цвета. Однако они не задают яркость (интенсивность) результирующего цвета. Яркость определяется координатой $Y$, а $X$ и $Z$ подбираются в соответствующем масштабе. При таком соглашении ($x$, $y$, $Y$) определяют как цветность, так и яркость. Обратное преобразование координат цветности в координаты цвета $XYZ$ имеет вид

$$X = x\frac{Y}{y} \qquad Y = Y \qquad Z = (1-x-y)\frac{Y}{y} \tag{5.19}$$

Комиссия решила ориентировать треугольник $XYZ$ таким образом, чтобы равные количества гипотетических основных цветов $XYZ$ в сумме давали белый.

Цветовой график МКО 1931 г. показан на рис. 5.56. Контур, напоминающий крыло, это геометрическое место точек всех видимых длин волн, т. е. линия спектральных цветностей. Числа на контуре соответствуют длине волны в данной точке. Красный находится в

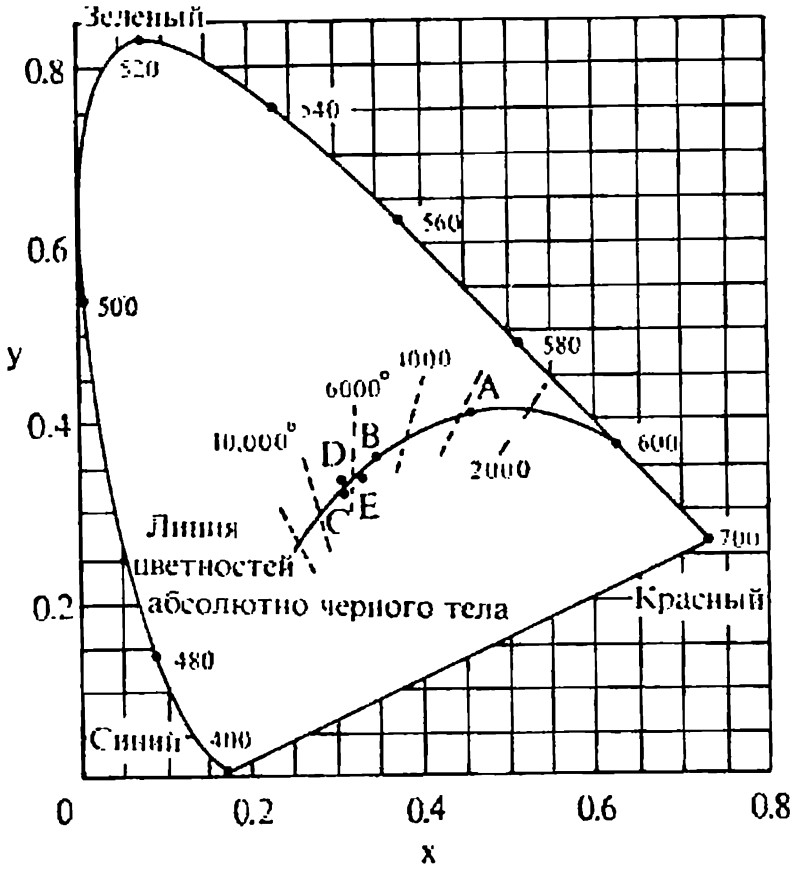



Рис. 5.56. Цветовой график МКО с линией цветностей абсолютно черного тела и точками цветностей стандартных источников $A$, $B$, $C$, $D_{6500}$ ($D$) и $E$ (белый цвет равных энергий.)

нижнем правом углу, зеленый — вверху, а синий — в левом нижнем углу графика. Отрезок, соединяющий концы кривой, называется линией пурпурных цветностей. Кривая внутри контура соответствует цвету абсолютно черного тела при нагревании от 1000° К до бесконечности. Пунктиром обозначена температура, а также направления, вдоль которых глаз хуже всего различает изменение цвета. Опорный белый — это точка равных энергий $E$ ($x = 0.333$, $y = 0.333$), а стандартные источники МКО — $A$ (0.448, 0.408), $B$ (0.349, 0.352), $C$ (0.310, 0.316), $D_{6500}$ (0.313, 0.329). Источник $A$ аппроксимирует теплый цвет газонаполненной лампы накаливания с вольфрамовой нитью при 2856° К. Он намного «краснее» остальных. Источник $B$ соответствует солнечному свету в полдень, а $C$ — полуденному освещению при сплошной облачности. Источник $C$ принят в качестве опорного белого цвета Национальным комитетом по телевизионным стандартам (NTSC). Источник $D_{6500}$, соответствующий излучению абсолютно черного тела при 6504° К, несколько «зеленее». Он применяется в качестве опорного белого цвета во многих телемониторах.

Как видно из рис. 5.57, цветовой график очень удобен. Чтобы получить дополнительный цвет, нужно продолжить прямую, проходящую через данный цвет и опорный белый, до пересечения с другой стороной кривой. Например, дополнительным к красно-

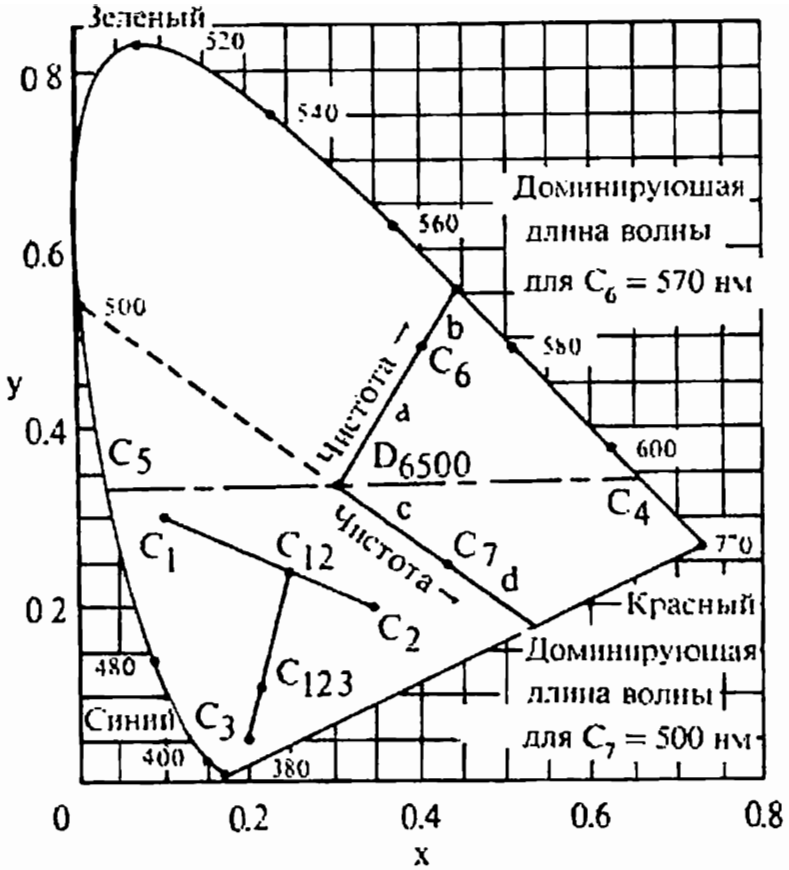



Рис. 5.57. Применение цветового графика.

оранжевому цвету $C_4(\lambda = 610$ нм) является сине-зеленый цвет $C_5(\lambda = 491$ нм). При сложении в определенной пропорции цвета и его дополнения получается белый. Для того чтобы найти доминирующую длину волны цвета, нужно продолжить прямую, проходящую через опорный белый и данный цвет, до пересечения с линией спектральных цветностей. Например, на рис. 5.57 доминирующая длина волны цвета $C_6$ равна 570 нм, т. е. она желто-зеленая. Если прямая пересекает линию пурпурных цветностей, то у этого цвета нет доминирующей длины волны в видимой части спектра. В этом случае она определяется как дополнительная доминирующая длина волны с индексом «$c$», т. е. прямая продолжается от цвета через опорный белый в обратном направлении. Например, доминирующая длина волны цвета $C_7$ на рис. 5.57 равна 500 нм.

Чистые или полностью (на 100%) насыщенные цвета лежат на линии спектральных цветностей. Опорный белый считается «полностью разбавленным», т. е. его чистота равна 0%. Чтобы вычислить чистоту промежуточных цветов, надо найти отношение расстояния от опорного белого до данного цвета к расстоянию от опорного белого до линии спектральных или пурпурных цветностей. Например, чистота цвета $C_6$ на рис. 5.57 равна $a/(a + b)$, а $C_7$ равна $c/(c + d)$.

Координаты цветности МКО для смеси двух цветов определя-

ются по законам Грассмана сложением основных цветов. Смесь цветов $C_1(x_1, y_1, Y_1)$ и $C_2(x_2, y_2, Y_2)$ является

$$C_{12} = (x_1 + x_2) + (y_1 + y_2) + (z_1 + z_2)$$

Пользуясь уравнениями (5.18) и (5.19) и введя обозначения

$$T_1 = \frac{Y_1}{y_1} \qquad T_2 = \frac{Y_2}{y_2}$$

получаем координаты цветности смеси

$$x_{12} = \frac{x_1T_1 + x_2T_2}{T_1 + T_2} \qquad y_{12} = \frac{y_1T_1 + y_2T_2}{T_1 + T_2} \qquad Y_{12} = Y_1 + Y_2$$

Этим способом можно сложить и большее количество цветов, если последовательно добавлять в смесь новые цвета. Рассмотрим пример.

**Пример 5.10.** Смешение цветов

Найдем координаты цветности МКО для смеси цветов $C_1(0.1,0.3,10)$, $C_2(0.35,0.2,10)$ и $C_3(0.2,0.05,10)$, показанных на рис. 5.57. Воспользуемся описанным выше методом для смеси $C_1$ и $C_2$:

$$T_1 = \frac{Y_1}{y_1} = \frac{10}{0.3} = 33.33 \qquad T_2 = \frac{Y_2}{y_2} = \frac{10}{0.2} = 50$$

$$x_{12} = \frac{x_1T_1 + x_2T_2}{T_1 + T_2} = \frac{(0.1)(33.33) + (0.35)(50)}{33.33 + 50} = 0.25$$

$$y_{12} = \frac{y_1T_1 + y_2T_2}{T_1 + T_2} = \frac{(0.3)(33.33) + (0.2)(50)}{33.33 + 50} = 0.24$$

$$Y_{12} = Y_1 + Y_2 = 10 + 10 = 20$$

Таким образом, смесь $C_1$ и $C_2$ дает цвет $C_{12}(0.25,0.24,20)$. Отметим, что кординаты смеси лежат на отрезке $C_1C_2$. Теперь добавим к смеси $C_{12}$ цвет $C_3$:

$$T_{12} = \frac{Y_{12}}{y_{12}} = \frac{20}{0.24} = 83.33 \qquad T_3 = \frac{Y_3}{y_3} = \frac{10}{0.05} = 200$$

$$x_{123} = \frac{x_{12}T_{12} + x_3T_3}{T_{12} + T_3} = \frac{(0.25)(83.33) + (0.2)(200)}{83.33 + 200} = 0.215$$

$$y_{123} = \frac{y_{12}T_{12} + y_3T_3}{T_{12} + T_3} = \frac{(0.24)(83.33) + (0.05)(200)}{83.33 + 200} = 0.106$$

$$Y_{123} = Y_{12} + Y_3 = 20 + 10 = 30$$

Смесь $C_1$, $C_2$ и $C_3$ — это $C_{123}(0.125,0.106,30)$. Она лежит на отрезке $C_{12}C_3$.

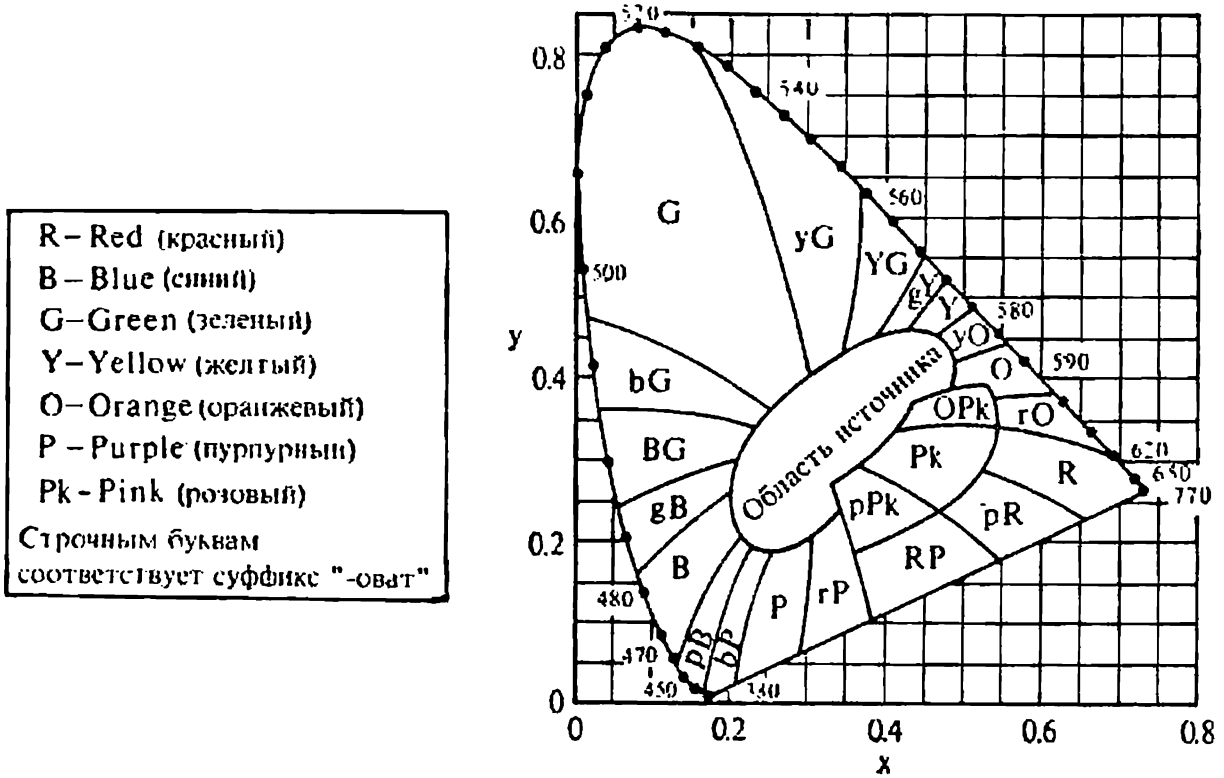



Рис. 5.58. Цветовой график МКО 1931 г. с названиями цветов.

На рис. 5.58 изображен график цветности МКО с названиями обычных воспринимаемых цветов [5-49]. В надписях на рис. 5.58 маленьким буквам в сокращенных названиях цветов соответствует суффикс «-оват», например *yG* это желт*овато*-зеленый (yellow*ish*-green). Каждый цвет на своем участке меняет насыщенность или чистоту от почти нулевой около источника (пастельный цвет) до полной (сочной) у линии спектральных цветностей. Обратите внимание, что оттенки зеленого занимают почти всю верхнюю часть графика, а красные и синие собраны внизу у линии пурпурных цветностей. Поэтому равные площади и расстояния на графике не соответствуют одинаковым различиям восприятия. Для того чтобы исправить этот недостаток, было предложено несколько преобразований этого графика. Полученные равноконтрастные цветовые пространства рассматриваются в работах [5-44 — 5-47].

Цветное телевидение, кино, многокрасочная типографская печать и т. д. не покрывают весь диапазон или охват цветов видимого спектра. Цветовой охват, который можно воспроизвести в аддитивной системе, — это треугольник на графике МКО с вершинами в основных цветах RCB. Любой цвет внутри треугольника можно получить из основных цветов. На рис. 5.59 и в табл. 5.4 показан цветовой охват для основных цветов RGB при обычном мониторе на ЭЛТ и в стандарте NTSC. Для сравнения изображена также суб-

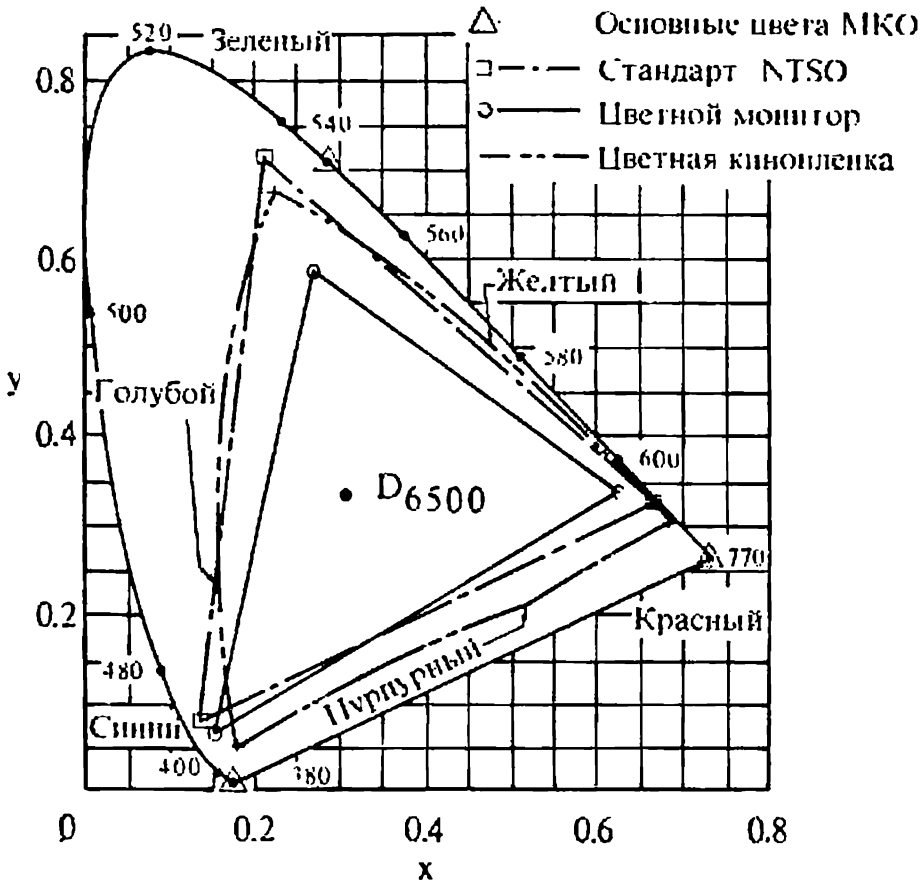



Рис. 5.59. Цветовые охваты.

трактивная цветовая система CMY, приведенная к координатам МКО, которая применяется в цветном кино. Обратите внимание, что ее охват не имеет треугольной формы и что он шире, чем охват у цветного монитора; т. е. некоторые цвета, полученные на кинопленке, нельзя воспроизвести на телеэкране. Кроме того, показаны основные цвета МКО $XYZ$, лежащие на линии спектральных цветностей: красный — 700 нм, зеленый — 543.1 нм, синий —

**Таблица 5.4.** Координаты цветности МКО для основных цветов RGB

| | | x | y |
|---|---|---|---|
| Основные цвета МКО *XYZ* | Красный | 0.735 | 0.265 |
| | Зеленый | 0.274 | 0.717 |
| | Синий | 0.167 | 0.009 |
| Стандарт NTSC | Красный | 0.670 | 0.330 |
| | Зеленый | 0.210 | 0.710 |
| | Синий | 0.140 | 0.080 |
| Цветной монитор на ЭЛТ | Красный | 0.628 | 0.346 |
| | Зеленый | 0.268 | 0.588 |
| | Синий | 0.150 | 0.070 |

435.8 нм. С их помощью были получены уравнивающие функции на рис. 5.52.

Координаты цветности МКО или координаты цвета представляют точный стандарт определения цвета. Однако в каждой отрасли промышленности, имеющей дело с цветом, существует свой набор основных цветов или обозначений. Координаты МКО оказываются полезными при передаче цветовой информации из одной области применения в другую. Поэтому представляют интерес преобразования координат МКО в различные цветовые системы и обратно. В машинной графике чаще всего требуется преобразовывать координаты МКО $XYZ$ в систему основных цветов $RGB$, применяемую в телевизионных мониторах. Здесь будут рассмотрены только эти преобразования, а более общие вопросы затрагиваются в работах [5-44 — 5-46, 5-48].

Преобразование между двумя аддитивными цветовыми системами проводится по законам Грассмана. Переход от цветового пространства RGB к пространству МКО $XYZ$ задается следующим образом:

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} X_r & X_g & X_b \\ Y_r & Y_g & Y_b \\ Z_r & Z_g & Z_b \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} \tag{5.20}$$

где $X_r$, $Y_r$, $Z_r$ — цвета для получения координаты единичного количества основного цвета $R$; аналогично для $X_g$, $Y_g$, $Z_g$ и $X_b$, $Y_b$, $Z_b$. Например, если $R = 1$, $G = 0$, $B = 0$, то $X = X_r$, $Y = Y_r$, $Z = Z_r$. Если известны координаты цветности МКО $(x, y)$ для основных цветов RGB, то

$$\begin{aligned} x_r &= \frac{X_r}{X_r + Y_r + Z_r} = \frac{X_r}{C_r} \\ y_r &= \frac{Y_r}{X_r + Y_r + Z_r} = \frac{Y_r}{C_r} \\ z_r &= 1 - x_r - y_r = \frac{Z_r}{X_r + Y_r + Z_r} = \frac{Z_r}{C_r} \end{aligned} \tag{5.21}$$

и аналогично для $x_g$, $y_g$, $z_g$ и $x_b$, $y_b$, $z_b$. При $C_g = X_g + Y_g + Z_g$ и $C_b = X_b + Y_b + Z_b$ уравнение (5.20) принимает вид

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} x_r C_r & x_g C_g & x_b C_b \\ y_r C_r & y_g C_g & y_b C_b \\ (1 - x_r - y_r) C_r & (1 - x_g - y_g) C_g & (1 - x_b - y_b) C_b \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} \tag{5.22}$$

или, более компактно,

$$[\mathbf{X}'] = [\mathbf{C}'][\mathbf{R}']$$

величины $C_r$, $C_b$ и $C_g$ необходимы для того, чтобы полностью определить преобразования между системами основных цветов. Если известна яркость $Y_r$, $Y_g$ и $Y_b$ единичных количеств основных цветов RGB, то

$$C_r = \frac{Y_r}{y_r} \qquad C_g = \frac{Y_g}{y_g} \qquad C_b = \frac{Y_b}{y_b}$$

Если заданы координаты цвета для опорного белого ($X_w$, $Y_w$, $Z_w$), то искомые результаты получаются при решении уравнения (5.22), где $[\mathbf{R}'] = [C_r\ C_g\ C_b]^T$ и $[\mathbf{X}'] = [X_w\ Y_w\ Z_w]^T$. Если же известны не координаты цвета, а координаты цветности и яркость ($x_w$, $y_w$, $Y_w$), то [5-48]:

$$\begin{aligned} C_r &= (Y_w/y_w)[x_w(y_g - y_b) - y_w(x_g - x_b) + x_g y_b - x_b y_g]/D \\ C_g &= (Y_w/y_w)[x_w(y_b - y_r) - y_w(x_b - x_r) - x_r y_b + x_b y_r]/D \\ C_b &= (Y_w/y_w)[x_w(y_r - y_g) - y_w(x_r - x_g) + x_r y_g - x_g y_r]/D \end{aligned} \tag{5.23}$$

и

$$D = x_r(y_g - y_b) + x_g(y_b - y_r) + x_b(y_r - y_g) \tag{5.24}$$

Обратное преобразование из цветового пространства МКО $XYZ$ в пространство RGB задается как

$$[\mathbf{R}'] = [\mathbf{C}']^{-1}[\mathbf{X}'] = [\mathbf{C}''][\mathbf{X}'] \tag{5.25}$$

где $[\mathbf{C}'] = [\mathbf{C}']^{-1}$ имеет следующие компоненты:

$$\begin{aligned} C''_{11} &= [(y_g - y_b) - x_b y_g + y_b x_g]\ /C_r D \\ C''_{12} &= [(x_b - x_g) - x_b y_g + x_g y_b]\ /C_r D \\ C''_{13} &= [x_g y_b - x_b y_g]\ /C_r D \\ C''_{21} &= [(y_b - y_r) - y_b x_r + y_r x_b]\ /C_g D \\ C''_{22} &= [(x_r - x_b) - x_r y_b + x_b y_r]\ /C_g D \\ C''_{23} &= [x_b y_r - x_r y_b]\ /C_g D \\ C''_{31} &= [(y_r - y_g) - y_r x_g + y_g x_r]\ /C_b D \\ C''_{32} &= [(x_g - x_r) - x_g y_r + x_r y_g]\ /C_b D \\ C''_{33} &= [x_r y_g - x_g y_r]\ /C_b D \end{aligned}$$

Рассмотрим данный метод на примере.

**Пример 5.11.** Преобразование основных цветов МКО в систему RGB

Требуется преобразовать цвет с координатами цветности МКО $x = 0.25, y = 0.2$ и яркостью $Y = 10.0$ для вывода на экран с координатами цветности RGB, заданными в табл. 5.4. Монитор настроен на опорный белый цвет $D_{5500}$, поэтому компоненты его основных цветов таковы:

$$\begin{array}{lll} x_r = 0.628 & x_g = 0.268 & x_b = 0.150 \\ y_r = 0.346 & y_g = 0.588 & y_b = 0.070 \end{array}$$

Составляющими опорного белого цвета будут

$$x_w = 0.313 \qquad y_w = 0.329 \qquad Y_w = 1.0$$

Вычислим сначала $D$:

$$\begin{aligned} D &= x_r(y_g - y_b) + x_g(y_b - y_r) + x_b(y_r - y_g) \\ &= 0.628(0.588 - 0.07) + 0.268(0.07 - 0.346) + 0.15(0.346 - 0.588) = 0.215 \end{aligned}$$

Тогда

$$\begin{aligned} DC_r / (Y_w / y_w) &= x_w(y_g - y_b) - y_w(x_g - x_b) + x_g y_b - x_b y_g \\ &= 0.313(0.588 - 0.07) - 0.329(0.268 - 0.15) + 0.268(0.07) \\ &\quad - 0.15(0.588) = 0.0539 \end{aligned}$$

и

$$C_r = \frac{0.0539}{D}\left(\frac{Y_w}{y_w}\right) = \frac{0.0539}{0.215}\left(\frac{1}{0.329}\right) = 0.762$$

Аналогично получаем $C_g = 1.114$ и $C_b = 1.164$. По координатам цветности рассчитаем координаты цвета $XYZ$:

$$X = x\frac{Y}{y} = 0.25\frac{10}{0.2} = 12.5$$

$$Z = (1 - x - y)\frac{Y}{y} = (1 - 0.25 - 0.2)\frac{10}{0.2} = 27.5$$

Преобразование, заданное уравнением (5.22), имеет вид:

$$[R] = [C''][X']$$

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 2.739 & -1.145 & -0.424 \\ -1.119 & 2.029 & 0.033 \\ 0.138 & -0.333 & 1.105 \end{bmatrix} \begin{bmatrix} 12.5 \\ 10.0 \\ 27.5 \end{bmatrix} = \begin{bmatrix} 11.133 \\ 7.209 \\ 28.772 \end{bmatrix}$$

Преобразование координат RGB в координаты цветности МКО выполняется аналогично.

**Пример 5.12.** Преобразование основных цветов RGB в систему МКО

Преобразуем цвет с координатами RGB (255,0,0), т. е. красный цвет максимальной интенсивности на экране монитора, в координаты цветного МКО. Основные цвета телеэкрана и опорный белый цвет такие же, как в примере 5.11, поэтому $D$, $C_r$, $C_g$, $C_b$ равны соответствующим значениям из предыдущего примера. Применяя (5.21), получаем

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} 0.478 & 0.299 & 0.175 \\ 0.263 & 0.655 & 0.081 \\ 0.020 & 0.160 & 0.908 \end{bmatrix} \begin{bmatrix} 255 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 121.94 \\ 67.19 \\ 5.05 \end{bmatrix}$$

Координаты цветности равны

$$x = \frac{X}{X+Y+Z} = \frac{121.94}{121.94+67.19+5.05} = \frac{121.94}{194.18} = 0.628$$

$$y = \frac{Y}{X+Y+Z} = \frac{67.19}{194.18} = 0.346$$

$$Y = 67.19$$

Это координаты цветности основного красного цвета монитора (см. табл. 5.4).

Для того чтобы использовать систему RGB как стандарт в цветном телевидении, сигнал должен лежать в полосе от 0 до 6 МГц и быть совместимым со стандартным черно-белым телевидением. В 1953 г. Национальный комитет по телевизионным системам (NTSC) принял в качестве стандарта цветовую систему YIQ, основанную на модели МКО $XYZ$. Из-за ограничений на ширину полосы пропускания яркость определяется одной координатой $Y$. Сигнал $Y$ занимает основную часть полосы частот (0—4 МГц), причем в нем пропорции красного, зеленого и синего основных цветов NTSC выбраны так, что он соответствует кривой спектральной чувствительности глаза. В сигнале $Y$ содержится информация о яркости, поэтому в черно-белом телевидении используется только эта координата. В качестве опорного белого цвета в системе NTSC раньше использовался стандартный источник $C$ МКО. Сейчас же для этой цели обычно применяется стандартный источник $D_{6500}$ МКО [5-50]. Различие между ними невелико.

Для того чтобы передавать цвет, т. е. тон и насыщенность, при помощи более узкой полосы частот, учитываются некоторые особенности зрительного восприятия. В частности, чем меньше предмет, тем хуже различаются его цвет, а объекты, меньшие определенного размера, кажутся черно-белыми. Если же объект меньше некоторого минимального предела, то его цвет вообще не воспри-

нимается. В системе $YIQ$ информация о тоне и насыщенности цвета представляется с помощью линейных комбинаций разностей красного, зеленого и синего цветов и значения $Y$. Координата цвета $I$ (синфазный сигнал) соответствует цветам от оранжевого до голубого, т. е. «теплым» тонам, $Q$ (интегрированный сигнал) — от зеленого до пурпурного, т. е. всем остальным. Координата $I$ занимает полосу частот примерно 1.5 МГц, а $Q$ — только 0.6 МГц. Преобразование из RGB в YIQ имеет вид

$$\begin{bmatrix} Y \\ I \\ Q \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ 0.596 & -0.274 & -0.322 \\ 0.211 & -0.522 & 0.311 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

а из YIQ в RGB —

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1 & 0.956 & 0.623 \\ 1 & -0.272 & -0.648 \\ 1 & -1.105 & 0.705 \end{bmatrix} \begin{bmatrix} Y \\ I \\ Q \end{bmatrix}$$

Для того чтобы преобразовать в YIQ координаты $XYZ$ МКО или обратно, нужно объединить эти уравнения с уравнениями (5.22) и (5.25).

Как и система МКО, цветовые пространства RGB и CMY трехмерны и условно изображаются в виде куба (рис. 5.60).

Началом координат в цветовом кубе RGB служит черный цвет,

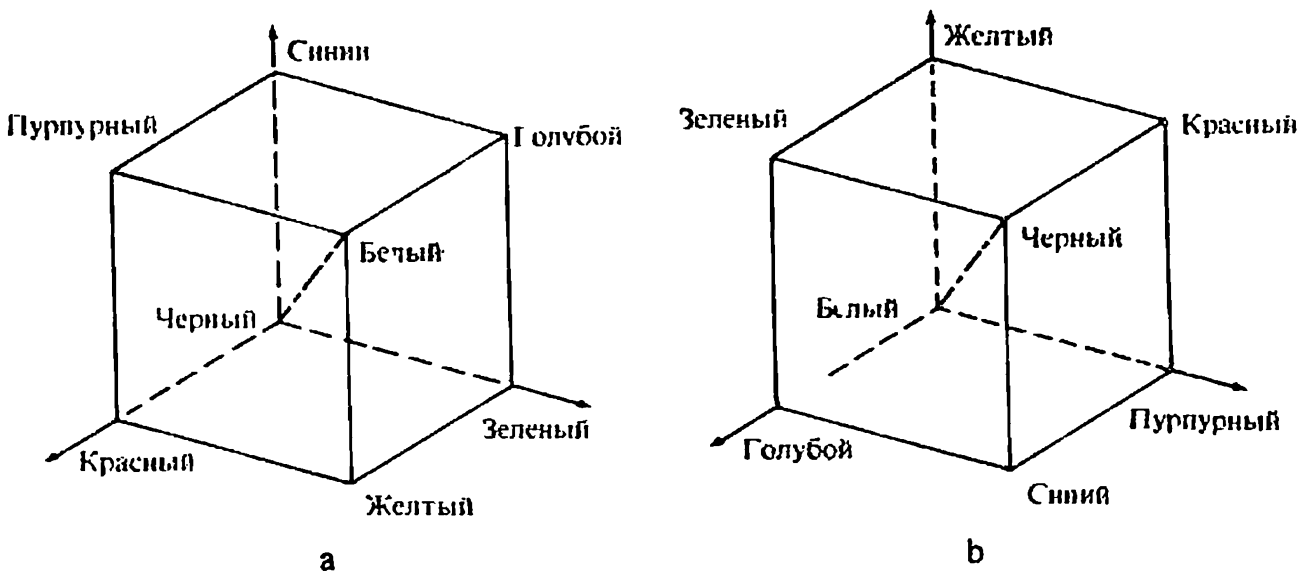



Рис. 5.60. Цветовые кубы: (a) RGB, (b) CMY.

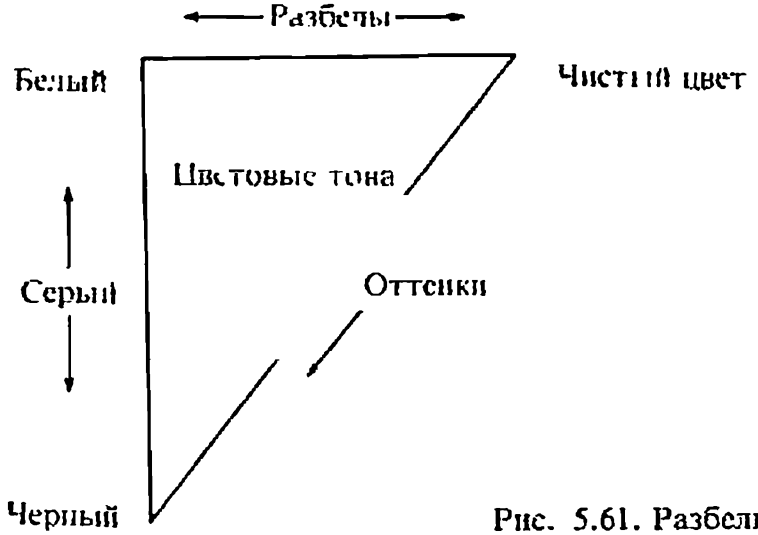



Рис. 5.61. Разбелы, оттенки и тона чистого цвета.

а в CMY — белый. Ахроматические, т. е. серые цвета в обеих моделях расположены по диагонали от черного до белого, а дополнительные цвета лежат в противоположных вершинах. Преобразование между пространствами RGB и CMY выражается следующим образом:

$$[R\ G\ B] = [1\ 1\ 1] - [C\ M\ Y].$$

Однако описывать субъективное восприятие цвета людьми в этих системах неудобно. Например, как в обозначениях МКО, RGB или CMY задать пастельный красновато-оранжевый цвет (см. рис. 5.58)? Художники характеризуют цвет с помощью таких понятий, как разбелы, оттенки, тона. Разбелы получают, добавляя в чистый цвет белый, оттенки — черный, тона — добавляя обе эти краски. Это удобно изобразить на треугольнике (рис. 5.61). Таким образом представляется один цвет, а если собрать треугольники для всех чистых цветов вокруг центральной черно-белой оси, то можно построить трехмерную модель субъективного представления цвета. На этом основана цветовая система Оствальда [5-51].

Смит [5-52] предложил построить модель субъективного восприятия в виде объемного тела HSV (цветовой тон, насыщенность, светлота). Если цветовой куб RGB (рис. 5.60, а) спроецировать на плоскость вдоль черно-белой диагонали, получается шестиугольник с основными и дополнительными цветами в вершинах. При снижении насыщенности или чистоты основных цветов размер и возможный цветовой охват куба RGB уменьшается, поэтому соответствующая шестиугольная проекция также будет меньше. Если проекции куба RGB и его подкубов собрать вдоль главной диагонали, представляющей количество света или светлоту цвета от черного (=0) до белого (=1), то получится объемный шестигранный конус

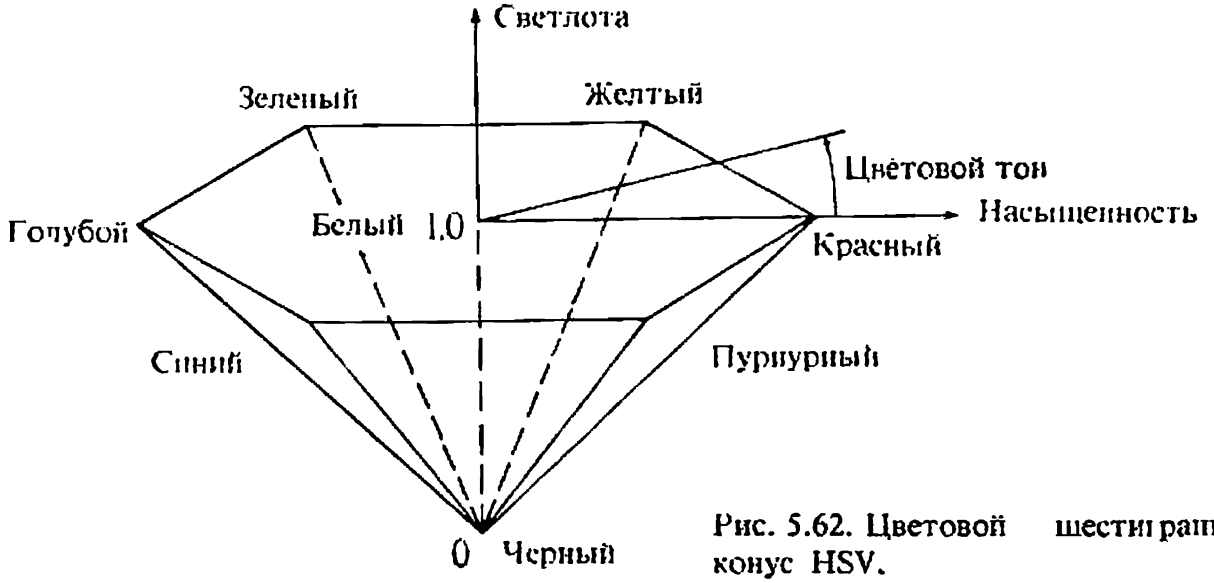



Рис. 5.62. Цветовой шестигранный конус HSV.

модели HSV (рис. 5.62). Интенсивность вдоль его оси возрастает от 0 в вершине до 1 на верхней грани, где она максимальна для всех цветов. Насыщенность определяется расстоянием от оси, а тон — углом (0°—360°), отсчитываемым от красного цвета. Для того чтобы на рисунке красный был в начале отсчета, проекция цветового куба была повернута на 120° против часовой стрелки. Насыщенность меняется от 0 на оси до 1 на границе шестиугольника. Отметим, что насыщенность зависит от цветового охвата, т. е. от расстояния от оси до границы для каждого $V$. При $S = 1$ цвета или их дополнения полностью насыщены. Ненулевая линейная комбинация трех основных цветов не может быть полностью насыщена. Если $S = 0$, то тон $H$ неопределен, г. е. на центральной оси находятся ахроматические, серые цвета.

Модель HSV соответствует тому, как составляют цвета художники. Чистым пигментам отвечают значения $V = 1$, $S = 1$; разбелам — цвета с увеличенным содержанием белого, т. е. с меньшим $S$; оттенкам — цвета с уменьшенным $V$, которые получаются при добавлении черного. Тон изменяется при уменьшении как $V$, так и $S$.

Преобразование цветового пространства HSV в RGB выполняется непосредственно с помощью геометрических соотношений между цветовым шестигранным конусом и кубом. Следующий алгоритм заимствован у Смита [5-52]:

### *Алгоритм преобразования HSV в RGB*

H — *цветовой тон (0—360°), 0° — красный*
S — *насыщенность (0—1)*
V — *светлота (0—1)*

```
RGB — красный, зеленый, синий; основные цвета (0—1)
Floor — функция выделяющая целую часть числа
  проверка ахроматического случая
  if S = 0 then
    if H = Неопределенность then
      R = V
      G = V
      B = V
    else
      если H определено, то ошибка
    end if
  else
    хроматический случай
    if H = 360 then
      H = 0
    else
      H = H/60
    end if
    I = Floor (H)
    F = H − I
    M = V * (1 − S)
    N = V * (1 − S * F)
    K = V * (1 − S * (1 − F))
    (R, G, B) = (V, K, M) означает R = V  G = K, B = M и
    т. д.
    if I = 1 then (R, G, B) = (V, K, M)
    if I = 2 then (R, G, B) = (N, V, M)
    if I = 3 then (R, G, B) = (M, V, K)
    if I = 4 then (R, G, B) = (M, N, V)
    if I = 5 then (R, G, B) = (K, M, V)
    if I = 6 then (R, G, B) = (V, M, N)
  end if
  finish
```

Преобразование из цветового пространства RGB в HSV проводится по данному ниже алгоритму, также взятому у Смита:

***Алгоритм преобразования RGB в HSV***

```
RGB — красный, зеленый, синий; основные цвета (0—1)
H — цветовой тон (0—360°), 0° — красный
S — насыщенность (0—1)
```

```
V — светлота (0—1)
Max — функция определения максимума
Min — функция определения минимума
   определение светлоты
   V = Max (R, G, B)
   определение насыщенности
   Temp = Min (R, G, B)
   if V = 0 then
      S = 0
   else
      S = (V – Temp)/V
   end if
   определение цветового тона
   if S = 0 then
      H = Неопределенность
   else
      Cr = (V – R)/(V – Temp)
      Cg = (V – G)/(V – Temp)
      Cb = (V – B)/(V – Temp)
      цвет между желтым и пурпурным
      if R = V then H = Cb – Cg
      цвет между голубым и желтым
      if G = V then H = 2 + Cr – Cb
      цвет между пурпурным и голубым
      if B = V then H = 4 + Cg – Cr
      перевод в градусы
      H = 60 * H
      приведение к положительным величинам
      if H < 0 then H = H + 360
   end if
   finish
```

Джоблов и Гринберг [5-53] рассматривают другое представление пространства HSV, в котором применяется не шестигранный конус, а цилиндр.

Цветовая модель HLS (цветовой тон, светлота, насыщенность) в виде двойного шестигранного конуса является расширением одиночного конуса HLS. Так как модель HLS применяется для самосвстящихся предметов, светлота здесь обозначает яркость, определенную в начале этого раздела. В модели HLS цветовой куб RGB проецируется так, что получается двойной шестигранный конус со

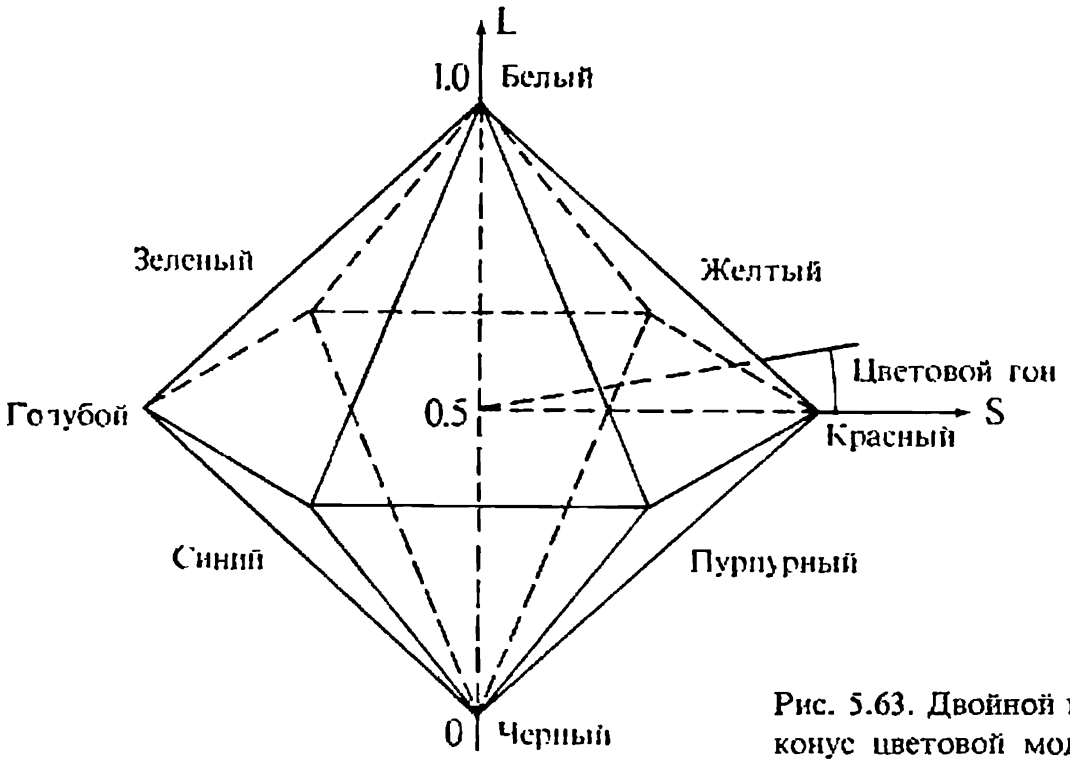



Рис. 5.63. Двойной шестигранный конус цветовой модели HLS.

светлотой по оси от 0 (черный) в одной вершине до 1 (белый) во второй (рис. 5.63). Как и в модели HSV, насыщенность определяется радиальным расстоянием от центральной оси. Полностью насыщенные основные цвета и их дополнения расположены при $S = 1$, а при $S = 0$ $H$ неопределено.

Алгоритм перевода HLS в RGB взят из работ [5-54] и [5-55]

***Алгоритм преобразования HLS в RGB***

```
H — цветовой тон (0—360°)  0° — красный
L — светлота (0—1)
S — насыщенность (0—1)
RGB — красный, зеленый, синий; основные цвета (0—1)
if L ⩽ 0.5 then
   M2 = L * (1 + S)
   else
      M2 = L + S − L * S
   end if
   M1 = 2 * L − M2
   проверка на нулевую насыщенность
   if S = 0 then
      if H = Неопределенность
         R = L
         G = L
         B = L
```

```
    else
      Ошибка вследствие неверных данных
    end if
  else
    расчет координат RGB
    call RGB (H + 120, M1, M2; Value)
    R = Value
    call RGB (H, M1, M2; Value)
    G = Value
    call RGB (H - 120, M1, M2; Value)
    B = Value
  end if
  finish
```

*подпрограмма расчета координат RGB*

```
subroutine RGB (H, M1, M2; Value)
H — цветовой тон (0—360°) 0° — красный
  приведение цветового тона и заданного диапазона
  if H < 0 then H = H + 360
  if H > 360 then H = H - 360
  определение координат
  if H < 60 then Value = M1 + (M2 - M1) * H/60
  if H ⩾ 60 and H < 180 then Value = M2
  if H ⩾ 180 and H < 240 then Value = M1 + (M2 - M1) *
  * (240 - H)/60
  if H ⩾ 240 and H ⩽ 360 then Value = M1
  return
```

Алгоритм преобразования RGB в HLS

*Алгоритм преобразования RGB в HLS*

```
RGB — красный, зеленый, синий; основные цвета (0—1)
H — цветовой тон (0—360°), 0° — красный
L — светлота (0—1)
S — насыщенность (0—1)
Max — функция определения максимума
Min— функция определения минимума
  определение светлоты
  M1 = Max (R, G, B)
  M2 = Min (R, G, B)
  L = (M1 + M2)/2
```

```
определение насыщенности
ахроматический случай
if M1 = M2 then
   S = 0
   H = Неопределенность
else
   хроматический случай
   if L ⩽ 0.5 then
      S = (M1 – M2)/(M1 + M2)
   else
      S = (M1 – M2)/(2 – M1 – M2)
   end if
   определение цветового тона
   Cr = (M1 – R)/(M1 – M2)
   Cg = (M1 – G)/(M1 – M2)
   Cb = (M1 – B)/(M1 – M2)
   if R = M1 then H = Cb – Cg
   if G = M1 then H = 2 + Cr – Cb
   if B = M1 then H = 4 + Cg – Cr
   H = 60 * H
   if H < 0 then H = H + 360
end if
finish
```

Цилиндрическое представление используется также в цветовой системе Манселла [5-56], основанной на наборе образцов цвета. Система Манселла — это стандарт восприятия. В этой системе цвет определяется цветовым тоном, насыщенностью (чистотой) и светлотой (количеством света). На центральной оси цилиндра находятся значения интенсивности от черного на нижней грани до белого — на верхней. При увеличении радиального расстояния от оси возрастает насыщенность, или чистота, цвета. Цветовой тон определяется углом, как показано на рис. 5.64. Главное преимущество системы Манселла, благодаря которому она широко применяется в промышленности, состоит в том, что одинаковые приращения насыщенности, тона и интенсивности вызывают ощущения одинаковых изменений при восприятии. В цилиндре есть незаполненные места. Существует преобразование субъективного представления цвета в системе Манселла в основные цвета МКО (см., например, [5-57]). Мейер и Гринберг [5-58] успешно пользовались цветами Манселла при выводе на цветной дисплей. При этом основные цве-

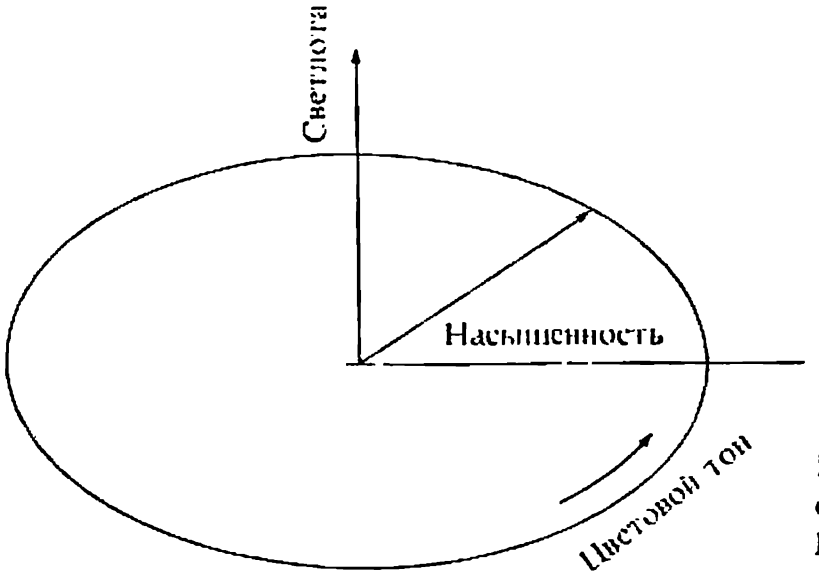



Рис. 5.64. Концептуальное представление системы цветов Манселла.

та МКО служили промежуточным стандартным цветовым пространством: сначала координаты Манселла преобразуются к основным цветам МКО *XYZ*, а затем переводятся в координаты RGB для цветного монитора. Этим способом Мейер и Гринберг получили некоторые цвета Манселла, которые раньше были известны только как экстраполяция существующих образцов.

Отсюда видно практическое значение стандартного цветового пространства МКО. В машинной графике оно особенно важно при создании или моделировании красок для репродукций на основе существующих промышленных красителей. Примером может служить выбор и изображение цветов на шевроле Камаро, см. вклейку 1. Если цвет краски выбирается из цветов монитора то не нужно разрабатывать спецификацию для производства красителей. Это делается преобразованием RGB координат монитора в основные цвета МКО. Изготовитель переводит их в свои характеристики, например, цветовой тон, насыщенность и светлоту по Манселлу. Если же нужно оценить окраску готовой машины, то спецификация красителя переводится в систему МКО, а затем в RGB для вывода на экран. Пространство МКО применяется также во многих других задачах.

Если предполагается, что зависимость между значениями, полученными из предыдущих цветовых моделей и напряжением на электронных пушках телевизионного монитора, является линейной, то для получения качественного изображения на экране необходимо произвести калибровку.

Интенсивность свечения экрана монитора пропорциональна напряжению, поданному на электронную пушку:

$$I = \text{const}(V)^{\gamma}.$$

Для вывода интенсивности $I_k$ напряжение должно быть

$$V_k = \left(\frac{I_k}{\text{const}}\right)^{1/\gamma}.$$

Кэтмул [5-59] описывает подробную процедуру расчета константы и $\gamma$. Из экспериментов известно, что $1 \leqslant \gamma \leqslant 4$, а для цветного монитора обычно лежит в пределах 2.3—2.8. Результаты калибровки используются как значения в таблице цветов.

Такая процедура, называемая гамма-коррекцией, калибрует только интенсивность экрана. Для калибровки цвета необходимо также знать цветности МКО красного, зеленого и синего люминофоров экрана. Коуэн [5-60] рассматривает процедуру настройки цветного телемонитора с учетом как гамма-коррекции, так и цветности люминофоров.

При закраске цветных изображений методом Гуро или Фонга (см. разд. 5.5. и 5.6) могут получиться самые неожиданные результаты. Они зависят от цветовой модели, которая используется при интерполяции, и модели вывода на экран. Если одну из них можно перевести в другую с помощью аффинного преобразования (прямая переходит в прямую), то на экране будет правильное изображение, а если такого преобразования нет, то возможно появление нарушений визуальной непрерывности. Как показано выше, между цветовыми моделями МКО, RGB, CMY и YIQ есть аффинное преобразование, а между этими моделями и системами HSV или HLS — нет. Нечто похожее происходит при наложении цветных прозрачных поверхностей, например в алгоритме удаления невидимых поверхностей.

Наконец, существует проблема цветовой гармонии, т. е. подбора радующих глаз цветов. Этой теме посвящено множество работ, знакомство с которыми лучше всего начать с книг Маркуса [5-61] или Джадда и Вышецки [5-45]. Один из основных принципов — упорядоченный подбор цветов, например вдоль определенной траектории или в одной плоскости цветовой модели. Лучше всего выбирать цвета, дающие одинаковое различие ощущений. Гармоничные цвета часто заимствуются из природы, например последовательность оттенков зеленого. Также можно выбирать цвета одинаковой насыщенности или тона, т. е. более или менее похожие.

## 5.16. ЛИТЕРАТУРА


5-1 Cornsweet, T.N., *Visual Perception*, Academic Press, New York, 1970.

5-2 Bui-Tuong, Phong, "Illumination for Computer Generated Images," doctoral thesis, University of Utah, 1973. Also as Comp. Sci. Dept. Rep. UTEC-CSc-73-129, NTIS ADA 008 786. A condensed version is given in *CACM*, Vol. 18, pp. 311–317, 1975.

5-3 Gouraud, H., "Computer Display of Curved Surfaces," doctoral thesis, University of Utah, 1971. Also as Comp. Sci. Dept. Rep. UTEC-CSc-71-113 and NTIS AD 762 018. A condensed version is given in *IEEE Trans. C-20*, pp. 623–628, 1971.

5-4 Duff, T., "Smooth Shaded Renderings of Polyhedral Objects on Raster Displays," *Computer Graphics*, Vol. 13, pp. 270–275, 1979 (*Proc. SIGGRAPH 79*).

5-5 Warn, David R., "Lighting Controls for Synthetic Images," *Computer Graphics*, Vol. 17, pp. 13–21, 1983 (*Proc. SIGGRAPH 83*).

5-6 Torrance, K.F., and Sparrow, E.M., "Theory for Off-Specular Reflection from Roughened Surfaces," *Journal of the Optical Society of America*, Vol. 57, pp. 1105–1114, 1967.

5-7 Blinn, James F., "Models of Light Reflection for Computer Synthesized Pictures," *Computer Graphics*, Vol. 11, pp. 192–198, 1977 (*Proc. SIGGRAPH 77*).

5-8 Cook, Robert L., "A Reflection Model for Realistic Image Synthesis," master's thesis, Cornell University, 1982.

5-9 Cook, Robert L., and Torrance, K.E., "A Reflectance Model for Computer Graphics," *ACM Trans. on Graphics*, Vol. 1, pp. 7–24, 1982.

5-10 Beckmann, P., and Spizzichino, A., *Scattering of Electromagnetic Waves from Rough Surfaces*, MacMillan, New York, 1963, pp. 1–33, 70–98.

5-11 Purdue University, *Thermophysical Properties of Matter*, Vol. 7: *Thermal Radiative Properties of Metals*, Vol. 8: *Thermal Radiative Properties of Nonmetallic Solids*, Vol. 9: *Thermal Radiative Properties of Coatings*, Plenum, New York, 1970.

5-12 Newell, M.E., Newell, R.G., and Sancha, T.L., "A Solution to the Hidden Surface Problem," Proc. ACM Annual Conf., Boston, August 1972, pp. 443–450.

5-13 Kay, Douglas Scott, "Transparency, Refraction and Ray Tracing for Computer Synthesized Images," master's thesis, Cornell University, 1979.

5-14 Kay, Douglas Scott, and Greenberg, Donald, "Transparency for Computer Synthe- sized Images," *Computer Graphics*, Vol. 13, pp. 158–164, 1979 (*Proc. SIGGRAPH 79*).

5-15 Myers, Allen J., "An Efficient Visible Surface Program," Rep. to NSF, Div. of Math. and Comp. Sci., Computer Graphics Res. Group, Ohio State University, July 1975.

5-16 Appel, Arthur, "Some Techniques for Shading Machine Rendering of Solids," *SJCC 1968*, Thompson Books, Washington, D.C., pp. 37–45.

5-17 Bouknight, Jack, "A Procedure for Generation of Three-dimensional Half-toned Computer Graphics Presentations," *CACM*, Vol. 13, pp. 527–536, 1970.

5-18 Kelley, Karl C., "A Computer Graphics Program for the Generation of Half-tone Images with Shadows," master's thesis, University of Illinois, 1970.

5-19 Bouknight, Jack, and Kelley, Karl C., "An Algorithm for Producing Half-tone Computer Graphics Presentations with Shadows and Movable Light Sources," *SJCC 1970*, AFIPS Press, Montvale, N.J. pp. 1–10.

5-20 Williams, Lance, "Casting Curved Shadows on Curved Surfaces," *Computer*

*Graphics*, Vol. 12, pp. 270–274, 1978 (*Proc. SIGGRAPH 78*).
5-21 Atherton, Peter R., "Polygon Shadow Generation with Application to Solar Rights," master's thesis, Cornell University, 1978
5-22 Atherton, Peter R., Weiler, Kevin, and Greenberg, Donald. "Polygon Shadow Generation," *Computer Graphics*, Vol 12, pp. 275–281, 1978 (*Proc. SIGGRAPH 78*).
5-23 Catmull, Edwin, "A Subdivision Algorithm for Computer Display of Curved Surfaces," doctoral thesis, University of Utah, 1974. Also as UTEC-CSc-74-133, NTIS A004968.
5-24 Blinn, James F., and Newell, Martin, E., "Texture and Reflection in Computer Generated Images." *CACM*, Vol. 19, pp. 542–547, 1976.
5-25 Blinn, James F., "Simulation of Wrinkled Surfaces," *Computer Graphics*, Vol. 12, pp. 286–292, 1978 (*Proc. SIGGRAPH 78*).
5-26 Carpenter, Loren C., "Computer Rendering of Fractal Curves and Surfaces," pp. 1–8, suppl. to *Proc. SIGGRAPH 80*, August 1980.
5-27 Fournier, Alain, and Fussell, Don. "Stochastic Modeling in Computer Graphics," pp. 9–15, suppl. to *Proc. SIGGRAPH 80*, August 1980
5-28 Mandelbrot, B., *Fractals: Form, Chance, and Dimension*, W. H. Freeman, San Francisco, 1977.
5-29 Kajiya, James T., "New Technique for Ray Tracing Procedurally Defined Objects," *Computer Graphics*, Vol. 17, pp. 91–102, 1983 (*Proc. SIGGRAPH 83*). Also in *ACM Trans. on Graphics*, Vol. 2, pp. 161–181, 1983.
5-30 Whitted, Turner, "An Improved Illumination Model for Shaded Display," *CACM*, Vol. 23, pp. 343–349, 1980.
5-31 Potmesil, M., and Chakravarty, I., "A Lens and Aperture Camera Model for Synthetic Image Generation," *Computer Graphics*, Vol. 15, pp. 297–305, 1981 (*Proc SIGGRAPH 81*).
5-32 Potmesil, M., and Chakravarty, I., "Synthetic Image Generation with a Lens and Aperture Camera Model," *ACM Trans. on Graphics*, Vol. 1, pp. 85–108, 1982.
5-33 Barr, Alan H., private communication.
5-34 Hall, Roy A., "A Methodology for Realistic Image Synthesis," master's thesis, Cornell University, 1983.
5-35 Hall, Roy A., and Greenberg, Donald. "A Testbed for Realistic Image Synthesis," *IEEE Computer Graphics and Applications*, Vol 3, pp. 10–20, 1983.
5-36 Moravec, Hans P., "3D Graphics and the Wave Theory," *Computer Graphics*, Vol. 15, pp. 289–296, 1981 (*Proc. SIGGRAPH 81*).
5-37 Korein, J., and Badler, V.R., "Temporal Anti-Aliasing in Computer Generated Animation," *Computer Graphics*, Vol. 17, pp. 377–388, 1983 (*Proc. SIGGRAPH 83*).
5-38 Potmesil, M., and Chakravarty, I., "Modeling Motion Blur in Computer-Generated Images," *Computer Graphics*, Vol. 17, pp. 389–399, 1983 (*Proc. SIGGRAPH 83*).
5-39 Reeves, William T., "Particle Systems—A Technique for Modeling a Class of Fuzzy Objects," *Computer Graphics*, Vol. 17, pp. 359–376, 1983 (*Proc. SIGGRAPH 83*), and *ACM Trans. on Graphics*, Vol. 2, pp. 91–108, 1983.
5-40 Blinn, James F., "Light Reflection Functions for Simulation of Clouds and Dusty Surfaces," *Computer Graphics*, Vol. 16, pp. 21–29, 1982 (*Proc. SIGGRAPH 82*).
5-41 Dungan, W. "A Terrain and Cloud Computer Image Generation Model," *Computer Graphics*, Vol. 13, pp 143–150, 1979 (*Proc. SIGGRAPH 79*).

5-42 Marshall, R., Wilson, R., and Carlson, Wayne, "Procedural Models for Generating Three-dimensional Terrain," *Compute Graphics*, Vol. 14, pp. 154–162, 1980 (*Proc SIGGRAPH 80*).
5-43 Csuri, C.A., "Panel: The Simulation of Natural Phenomena," *Comput r Graphics*, Vol. 17, pp. 137–139, 1983 (*Proc. SIGGR APH 83*).
5-44 Wyszecki, G., and Stiles, W.S., *Color Science*, Wiley, New York, 1967.
5-45 Judd, D.B., and Wyszecki, G., *Color in Business, Science and Industry*, Wiley, New York, 1975.
5-46 Hunt, R.W.G., *The Reproduction of Color*, 3d ed., Wiley, New York, 1975.
5-47 Hunter, Richard S., *The Measurement of Appearance*, Wiley, New York, 1975.
5-48 Meyer, Gary W., "Colorimetry and Computer Graphics," Program of Computer Graphics, Report Number 83-1, Cornell University, April 1983.
5-49 Judd, Deane B., "Colorimetry," National Bureau of Standards Circular 478, 1950. Updated in Nimerof, I., "Colorimetry," NBS monograph 104, 1968.
5-50 Pritchard, D.H., "U'S Color Television Fundamentals— A Review," *IEEE Trans. on Consumer Electronics*, Vol. CE-23, pp. 467–478, 1977.
5-51 Ostwald, N., *Colour Science*, Vols. I and II, Wimsor & Winsor, London, 1931.
5-52 Smith, Alvey Ray, "Color Gamut Transformation Pairs," *Computer Graphics*, Vol. 12, pp. 12–19, 1978 (*Proc. SIGGRAPH 78*).
5-53 Joblove, George H., and Greenberg, Donald, "Color Spaces for Computer Graphics," *Computer Graphics*, Vol 12, pp. 20-25, 1978 (*Proc. SIGGRAPH 78*).
5-54 "Status Report of the Graphics Standards Committee," *Computer Graphics*, Vol. 13, August 1979.
5-55 *Raster Graphics Handbook*, Conrac Division, Conrac Corporation, 600 N. Rimsdale Ave., Covina, California 91722.
5-56 Munsell, A.H., *A Color Notation*, 9th ed., Munsell Color Company, Baltimore, 1941. The latest *Book of Color* is available from Munsell Color Company, 2441 North Calvert Street, Baltimore, Maryland 21218.
5-57 Keegan, H.J., Rheinboldt, W.C., Schleter, J.C., Menard, J.P., and Judd, D.B., "Digital Reduction of Spectrophotometric Data to Munsell Renotations," *Journal of the Optical Society of America*, Vol. 48, p. 863, 1958.
5-58 Meyer, Gary W., and Greenberg, Donald, "Perceptual Color Spaces for Computer Graphics," *Computer Graphics*, Vol. 14, pp. 254–261, 1980 (*Proc. SIGGRAPH 80*).
5-59 Catmull, Edwin, "Tutorial on Compensation Tables," *Computer Graphics*, Vol. 13, pp. 1–7, 1979 (*Proc. SIGGRAPH 79*).
5-60 Cowan, William B., "An Inexpensive Scheme for Calibration of a Colour Monitor in Terms of CIF Standard Coordinates", *Computer Graphics*, Vol. 17, pp. 315–321, 1983 (*Proc. SIGGRAPH 83*).
5-61 Marcus, Aaron, "Color—A Tool for Computer Graphics Communication," *Close-up*, Vol. 13, pp. 1–9, August 1982.

## ЛИТЕРАТУРА НА РУССКОМ ЯЗЫКЕ

[5-45] Джад Д., Вишецки Дж. Цвет в науке и технике . — М.: Мир, 1978.

# Псевдокод

Описываемый псевдокод предназначен для обеспечения понимания и реализации представленных в тексте алгоритмов. Он не задумывался как точный, синтаксически корректный и полный язык. Элементы псевдокода заимствованы из нескольких общеизвестных языков программирования: Бейсика, Фортрана, Паскаля и т. д. В псевдокоде содержатся структурные конструкции, например **if-then-else** и **while**. Для удобства в язык включен оператор безусловного перехода **go to**. Оператор цикла **for-next** взят из Бейсика. В язык входят также подпрограммные модули. Функции и специальные подпрограммы, например **Min, Max, Push, Pop,** определены в алгоритмах индивидуально. Назначение таких процедур, как **Draw** и **Plot**, понятно из названия.

Коротко приведем здесь общие соглашения, используемые при описании алгоритмов. Все ключевые слова набраны строчными буквами жирным шрифтом. Все инструкции в теле операторов **if-then-else, while** или цикла **for-next** выделены при помощи отступа. Комментарии набраны курсивом и располагаются с таким же отступом, как и инструкции, к которым они относятся. Многолитерные имена переменных начинаются с прописной буквы, остальные литеры — строчные. Однолитерные имена могут быть представлены как прописной, так и строчной буквой. Функции выделены жирным шрифтом и начинаются с прописной буквы. Далее эти соглашения описываются подробно.

## А.1. Комментарии

Комментарии набраны курсивом. Операторы и относящиеся к ним комментарии расположены с одинаковым отступом. В начале алгоритма помещаются комментарии, кратко описывающие назначение алгоритма и определяющие используемые переменные.

## А.2. Константы

Все константы считаются десятичными числами, если в комментариях не оговорено иное. Например, все числа 9, −3, 6.732, $1 \cdot 10^{-9}$, −5.83 являются константами.

## А.3. Переменные

Переменная — это имя, используемое для хранения значения, которое может изменяться. Длинные имена переменных начинаются с прописной буквы, остальные ли-

теры в имени — строчные, если только использование прописных букв не поможет согласовать записи в алгоритме и основной части книги. Однолитерные имена могут быть как строчными, так и прописными буквами. Для большей наглядности могут использоваться литеры с индексами, например, Flag, $P_2$, x, y.

## A.4. Массивы

Массив — имя для индексированной совокупности величин. Для имен массивов действуют те же соглашения, что и для переменных. Обращение ко всему массиву осуществляется по его имени, а к индивидуальным элементам — по имени массива, за которым в скобках следует список индексов. В качестве примера приведем Window, Window(1, 3).

## A.5. Оператор присваивания

Знак «равно» используется для присваивания переменной, расположенной в левой части оператора, значения выражения — из правой части.

## A.6. Арифметические выражения

Обычные арифметические операции: умножение, деление, сложение и вычитание обозначаются знаками •, /, «+», «−».

## A.7. Логические операции и операции отношения

Логические операции **and** и **or**, как это здесь показано, выделяются жирным шрифтом. Операции отношения «равно», «не равно», «меньше», «больше», «меньше или равно», «больше или равно» обозначаются знаками =, ≠, <, >, ≤, ≥ соответственно. Эти операции используются для проверки условий. Результатом проверки является логическое значение «истина» или «ложь».

## A.8. Оператор finish

Оператор **finish** используется для обозначения конца работы алгоритма.

## A.9. Операторы while и end while

Операторы, расположенные внутри блока **while** = **end while**, циклически выполняются до тех пор, пока истинно некоторое условие. Значение условия проверяется в самом начале блока. Когда условие перестает быть истинным, выполнение программы продолжается с оператора, следующего за **end while**. Все операторы в блоке располагаются с отступом. Общая форма оператора такова:

```
while (условие)
    [операторы, которые необходимо выполнить]
end while
```

*Пример*

```
i = 0
while (i < 5)
    x = x + 5
```

```
    i = i + 1
end while
finish
```

## A.10. Оператор if-then

Оператор **if-then** используется в зависимости от истинности некоторого условия для изменения последовательности выполнения программы либо для присваивания переменной другого значения.

Если аргументом **then** служит номер оператора и условие истинно, то управление передается на оператор с данным номером. В противном случае выполняется следующий по порядку оператор. Номера операторов являются их метками.

Если аргументом **then** служит оператор присваивания и условие истинно, то присваивание выполняется. В противном случае присваивание не выполняется, а управление передается сразу на следующий оператор. Общая форма оператора такова:

**if** (условие) **then** (номер оператора)

**if** (условие) **then** (оператор присваивания)

*Примеры*

```
if (i < 10) then 3
if (i < 10) then x = x + 1
```

## A.11. Операторы if-then-else и end if

Оператор **if-then-else** используется для выбора в зависимости от истинности некоторого условия одного из двух блоков операторов. Для обозначения конца блока оператора **if-then-else** используется оператор **end if**. В этом операторе не подразумевается никакого повторения, выполняется лишь один из двух блоков. После этого выполнение программы продолжается с оператора, идущего следом за **end if**. Все операторы внутри блока **if-then-else** располагаются с отступом. Общая форма оператора

```
if (условие) then
    [операторы, выполняемые, если условие истинно]
else
    [операторы, выполняемые, если условие ложно]
end if
```

*Пример*

```
if (i > 0) then
    x = x + 1
else
    x = x - 1
end if
```

Если оператор **if-then-else** записывается на одной строке, то оператор **end if** опускается. Заметим, что когда опускается **else** и вторая группа операторов, то у нас получается блочный оператор.

## A.12. Оператор for-next

Циклическое управление можно реализовать с помощью оператора **for-next** аналогично оператору **while**. Циклическое выполнение операторов, расположенных внутри тела цикла **for-next**, осуществляется до тех пор, пока значение переменной цикла находится в заданных пределах. Все операторы внутри блока **for-next** располагаются с отступом. Общая форма оператора такова:

**for** (пер__цикла) = (нач__знач) **to** (кон__знач) **step** (шаг__цикла)
    [*операторы, которые надо выполнить*]
**next** (пер__цикла)

Если отсутствует шаг__цикла, то он полагается равным единице. Допустимы отрицательные значения шага. В качестве начального, конечного значений и шага цикла могут выступать переменные.

*Пример*

```
for x = 1 to n step a
    y = y + x
next x
```

## A.13. Оператор go to

Оператор **go to** вызывает безусловный переход на оператор, задаваемый аргументом. Общая форма оператора такова:

**go to** (номер оператора)

Номера операторов являются метками, они располагаются у самого левого края оператора.

## A.14. Подпрограммные модули

Подпрограмма — отдельный программный модуль, который вызывается с помощью оператора **call**. Начало подпрограммы определяется оператором **subroutine**. Выход из подпрограммного модуля обозначается оператором **return**. После выхода из подпрограммы управление передается на оператор из вызывающей программы, следующей за оператором **call**. Оператор **subroutine** содержит список входных и список выходных переменных. Связь между вызывающей программой и подпрограммой осуществляется только через эти переменные, все другие переменные в подпрограммном модуле являются локальными. Общая форма операторов **call**, **subroutine** и **return** такова:

**call** имя (входные переменные; выходные переменные)
**subroutine** имя (входные переменные; выходные переменные)
**return**

Списки входных и выходных переменных для операторов **call** и **subroutine** разделяются точкой с запятой и должны соответствовать друг другу. Первая буква имени подпрограммы — прописная, все остальные — строчные. Пример подпрограммы:

```
subroutine Проверка (x, y; Flag)
if x < y then
```

```
    Flag = 0
else
    Flag = 1
end If
return
```

## A.15. Функции

В алгоритмах, описанных в данной книге, определены разнообразные функции. Имена функций выделяются жирным шрифтом и начинаются с прописной буквы. В качестве примера приведем функцию

**Max** ($x_1$, $x_2$),

которая возвращает большее из значений $x_1$ и $x_2$.

# Задачи

Так как изучение машинной графики немыслимо без ее практического использования, здесь приводится ряд задач. Если в вашем распоряжении имеется растровое графическое устройство, то рекомендуется для уменьшения вычислительной сложности и улучшения наглядности использовать растр размером 32 × 32. Если есть устройство с большим разрешением, то в качестве единицы надо использовать группу пикселов. Если пиксел не квадратный, то следует подобрать группу пикселов, как можно более близкую к квадрату. Если в наличии имеется векторный дисплей, то сетка 32 × 32 изображается так, как это показано на рис. 4.43, с. Активация пиксела обозначается помещенной на нем цифрой или штриховкой. Предлагаемые задачи группируются по главам.

## К главе 2

2.1. Используя ЦДА (разд. 2.2) и алгоритм Брезенхема (разд. 2.5), напишите программу для вычерчивания отрезков из одной произвольной точки в другую на псевдорастре 32 × 32. Используйте псевдобуфер кадра в виде одномерного массива сначала для хранения изображения, а затем для вывода его из псевдобуфера на экран. Демонстрационный текст должен состоять по крайней мере из 16 отрезков с началом в центре окружности и концами, равномерно расположенным по окружности. Обеспечьте возможность устанавливать центр окружности в произвольную точку растра. Визуально сравните результаты. Напечатайте список активированных пикселов для отрезка из (0, 0) в (−8, −3) для обоих алгоритмов. Как воздействует на результат инициализация? Сравните вычислительную эффективность двух алгоритмов путем хронометрирования разложения в растр 100 случайно выбранных отрезков.

2.2. Окружность, разложенную в растр, можно получить с помощью алгоритма Брезенхема для генерации окружности, описанного в разд. 2.6. Ее можно также сгенерировать путем разложения в растр ребер вписанного многоугольника с помощью алгоритма Брезенхема для отрезка. Напишите программу, реализующую оба метода, и разложите в растр окружность радиуса $R = 15$ на растре 32 × 32. Сравните результаты для вписанных многоугольников с 4, 8, 16, 32, 64 и 128 сторонами для алгоритма генерации окружности. Используйте псевдобуфер кадра в виде одномер-

ного массива сначала для хранения изображения, а затем для вывода его из псевдобуфера на дисплей. Предусмотрите выдачу списка растровых точек, используя формат «строка-колонка» для каждого алгоритма в предположении, что начало координат (0, 0) находится в левом нижнем углу. Результаты сравните визуально и численно.

2.3. Пусть задан многоугольник с внешней частью, определяемой точками (4,4), (4,26), (20,26), (28,18), (21,4), (21,8), (10,8) и (10,4) и внутренним отверстием, описываемым точками (10,12), (10,20), (17,20), (21,16) и (21,12) на растре 32 × 32. Напишите программу, использующую простой алгоритм с упорядоченным списком ребер, который описан в разд. 2.18, для развертки и изображения сплошной области, лежащей внутри многоугольника. Предполагается, что начало координат (0, 0) расположено в левом нижнем углу растра. Выведите список заполненных пикселов в формате «строка-колонка» в порядке сканирования сверху вниз и слева направо.

2.4. Для многоугольника из задачи 2.3 напишите программу, реализующую более эффективный алгоритм с упорядоченным списком ребер, описанный в разд. 2.19. Преобразуйте в растровую форму и изобразите сплошную область внутри многоугольника. Используйте список активных ребер. Для реализации групповой сортировки по $y$ используйте связный список. Напечатайте содержимое списка активных ребер для строки 18 в растре 32 × 32. Напечатайте в порядке сканирования изображаемые пикселы и содержимое связанного списка.

2.5. Напишите программы, реализующие алгоритмы заполнения по ребрам и с перегородкой, описанные в разд. 2.20, для развертки сплошной области, расположенной внутри *внешней* границы многоугольника из задачи 2.3. Используйте псевдобуфер кадра в виде двумерного массива для хранения изображения, и для последующего вывода после развертки каждого ребра. Сравните результаты. Сравните вычисленную эффективность и эффективность ввода вывода двух алгоритмов. Можно ли с помощью этих алгоритмов правильно выполнить растровую развертку всего многоугольника, включая внутреннее отверстие?

2.6. Напишите программу, реализующую алгоритм заполнения по ребрам и флагу, описанный в разд. 2.21, для растровой развертки сплошной области, расположенной внутри *внешней* границы многоугольника из задачи 2.3. Используйте псевдобуфер кадра в виде двумерного массива для хранения изображения и для последующего вывода псевдобуфера кадра на дисплей. Изобразите содержимое буфера кадра после определения контура и после завершения развертки. Сравните с результатами задачи 2.5. Можно ли с помощью данного алгоритма правильно преобразовать в растровую форму весь многоугольник, включая и внутреннее отверстие? Если да, то модифицируйте, если это требуется, программу для обработки всего многоугольника. Если нет, то почему?

2.7. Напишите программу, реализующую простой гранично-заполняющий алгоритм с затравкой для заполнения внутренней части многоугольника из задачи 2.3. Обеспечьте выдачу списка граничных пикселов. Сгенерируйте и выдайте на печать список заполненных пикселов, если затравка — пиксел (14,20). Обеспечьте возможность просмотра содержимого стека в любой момент. Какова максимальная глубина стека?

2.8. Выполните задачу 2.7 для построчного алгоритма с затравкой, описанного в разд. 2.24. Сравните результаты.

2.9. Используя конфигурации ячеек 2 × 2, показанные на рис. 2.62, разработайте программу для вывода восьми полутонов «серого» слева направо в растре 32 × 32. Повторите результат для растра размера 64 × 64 и 128 × 128 и сравните результаты. Добавьте к растру 128 × 128 упорядоченное возмущение и сравните результаты.

## К главе 3

3.1. Реализуйте отсечение отрезков по двумерному прямоугольному окну с помощью простого алгоритма из разд. 3.1, алгоритма Сазерленда — Коэна из разд. 3.2 и алгоритма разбиения средней точкой из разд. 3.3. Сравните эффективность работы этих алгоритмов. Алгоритмы должны с первого захода определять и изображать полностью видимые отрезки, а также определять и отвергать полностью невидимые отрезки.

3.2. Напишите программу, реализующую двумерную версию алгоритма отсечения отрезков Кируса — Бека как внутренней, так и внешней областью произвольного выпуклого полигонального окна. Этот алгоритм должен определять и отвергать невыпуклые отсекающие многогранники. Для частного случая прямоугольника сравните полученные результаты с результатами задачи 3.1. Изменяйте число сторон в отсекающем прямоугольнике и фиксируйте зависимость времени работы алгоритма от числа сторон. Какая получается зависимость?

3.3. Обобщите задачу 3.2 на случай произвольного трехмерного выпуклого полигонального отсекающего множества (см. разд. 3.11).

3.4. Напишите программу, реализующую алгоритм Сазерленда — Ходжмена для отсечения многоугольников, который был описан в разд. 3.16. Ограничьтесь случаем произвольных многоугольников, отсекаемых прямоугольными окнами. Изображайте результирующий многоугольник после каждого этапа отсечения. В качестве теста возьмите многоугольник с вершинами (−4,2), (8,14), (8,2), (12,6), (12,−2), (4,−2), (4,6), (0,2), отсекаемый окном (0,10,0,10).

3.5. Обобщите задачу 3.4 на случай произвольных выпуклых окон.

3.6. Произведите отсечение плоского многоугольника, заданного вершинами $P_1(-0.4,0.4,0)$, $P_2(0.1,0.1,0)$, $P_3(0.3,0.3,0)$, $P_4(0.2,0,0)$, $P_5(0.3,-0.2,0)$, $P_6(0.1,-0.1,0)$, $P_7(-0.4,-0.4,0)$, $P_8(-0.2,0,0)$ и повернутого на 45° вокруг оси $x$ относительно цилиндра, ось которого совпадает с осью $z$, радиус равен 0.3, а экстремальные аппликаты равны ±0.3. Выполните эту задачу, используя алгоритм отсечения многоугольников Сазерленда — Ходжмена. Воспользуйтесь также алгоритмом Кируса — Бека отсечения отрезков для вычисления видимости концевых точек отрезков и точек пересечения отрезков с поверхностью. Не забудьте, что цилиндр имеет торцы. Этот цилиндр нужно задать вписанным многогранником с 32 сторонами. Изобразите цилиндр и отсеченный им многоугольник, используя подходящее видовое преобразование. Распечатайте список вершин многоугольника после отсечения.

3.7. Воспользуйтесь данными из упр. 3.6 и измените алгоритм Сазерленда — Ходжмена так, чтобы отсечь многоугольник относительно внешней области цилиндра. Изобразите цилиндр и отсеченный многоугольник, применив подходящее видовое преобразование. Распечатайте список вершин многоугольника после отсечения.

3.8. Видоизмените алгоритмы, созданные в задачах 3.6 и 3.7, для отсечения одного цилиндра другим так, чтобы имелась возможность применить классические тео-

ретико-множественные операции объединения и пересечения. Это упражнение имеет отношение к моделированию формы сплошных тел.

3.9. Напишите программу, реализующую алгоритм Вейлера — Азертона отсечения относительно невыпуклых многоугольников, который был описан в разд. 3.17. Распечатайте списки входных и выходных точек пересечения. Распечатайте списки вершин результата отсечения и отсекателя. В качестве теста возьмите обрабатываемый многоугольник с внешней границей (0,0), (20,0), (20, −20), (0, −20) и внутренней дырой (7, −13), (13, −13), (13, −7), (7, −7) и отсекающий многоугольник с внешней границей (−10, −10), (−10,10), (10,10), (10, −10) и внутренней дырой (−5, −5), (5,25), (5,5), (25,5). Обратите внимание на рис. 3.34, с. Изобразите исходные отсекающий и обрабатываемый многоугольники, а также получившийся отсеченный многоугольник.

## К главе 4

4.1. Напишите программу для ЭВМ, которая использует метод плавающего горизонта, описанный в разд. 4.2. Программа должна удалять невидимые линии для поверхности, описанной функцией:

$$F(x,z) = 8\cos(1.2R)/(R+1) \qquad R = \sqrt{x^2+z^2} \qquad -2\pi \le x,z < 2\pi$$

Точка наблюдения расположена в бесконечности на положительной полуоси $z$; оценка повернута на 25° вокруг оси $x$, а затем на 15° вокруг оси $y$.

4.2. Используя метод Робертса, удалите вручную невидимые линии из сцены, описанной ниже. Сцена изображается посредством преобразования диметрии; наблюдатель расположен в бесконечности на положительной полуоси $z$. Преобразование диметрии, за исключением проецирования на плоскость $z = 0$, задается в [1-1] матрицей 4 × 4, описывающей это преобразование в однородных координатах:

$$[T] = \begin{bmatrix} 0.92582 & 0.13363 & -0.35355 & 0 \\ 0 & 0.92541 & 0.35355 & 0 \\ 0.37796 & -0.32732 & 0.86603 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Обратной для этой матрицы преобразования является ее транспозиция. Сцена состоит из куба и прямоугольного параллелепипеда, заданных координатами вершин:

| куб | | | параллелепипед | | |
|---|---|---|---|---|---|
| 3 | 1 | 11 | 1 | 2 | 7 |
| 6 | 1 | 11 | 10 | 2 | 7 |
| 6 | 4 | 11 | 10 | 3 | 7 |
| 3 | 4 | 11 | 1 | 3 | 7 |
| 3 | 1 | 8 | 1 | 2 | 1 |
| 6 | 1 | 8 | 10 | 2 | 1 |
| 6 | 4 | 8 | 10 | 3 | 1 |
| 3 | 4 | 8 | 1 | 3 | 1 |

4.3. Напишите программу, реализующую алгоритм Робертса. Используйте оценку из задачи 4.2 как контрольный пример.

Решите задачи 4.4—4.11, используя в качестве контрольной основную сцену, описанную в примере 4.19, и ее модификацию. Основная контрольная сцена состоит из треугольника, который протыкает прямоугольник изнутри. Используется растр, состоящий из 32 × 32 элементов. Если нужно, то можно использовать двумерный массив для имитации буфера кадра. Координаты угловых точек прямоугольника таковы: $P_1(10,5,10)$, $P_2(10,25,10)$, $P_3(25,25,10)$, $P_4(25,5,10)$, а вершин треугольника: $P_5(15,15,15)$, $P_6(30,10,5)$, $P_7(25,25,5)$. В измененной сцене треугольник не протыкает прямоугольник, а $P_4$ заменяется на $P_5(15,15,5)$.

4.4. Напишите программу, реализующую основную версию алгоритма Варнока, которая описана в разд. 4.4. Изобразите результат для обоих описанных выше контрольных сцен. Изобразите каждое окно или подокно, которое образуется в процессе обработки сцены.

4.5. Повысьте эффективность алгоритма из задачи 4.4 путем создания более сложного теста непересечения. Включите в алгоритм, кроме того, возможность распознавать единственный охватывающий, единственный внутренний и единственный пересекающий многоугольники. Включите в алгоритм сортировку по приоритету глубины. Добавьте списковую структуру данных для реализации выигрыша от использования информации, полученной на более ранней стадии решения. Включите в алгоритм средства устранения эффекта ступенчатости (разд. 2.26). Для проверки и сравнения этих алгоритмов воспользуйтесь более сложными сценами.

4.6. Напишите программу, реализующую алгоритм Вейлера — Азертона (см. разд. 4.5), используя в качестве контрольных примеров описанные выше сцены.

4.7. Реализуйте алгоритм $z$-буфера (см. разд. 4.7). В качестве контрольных примеров используйте две описанные выше сцены. После обработки каждого многоугольника изобразите содержимое буфера кадра и $z$-буфера. Что получится, если ограничить точность значений $z$ в буфере соответственно до 32, 16, 8, 4 бит?

4.8. Напишите программу, реализующую описанный в разд. 4.8 алгоритм Ньюэла — Ньюэла — Санча, использующий список приоритетов. Добавьте к вышеописанным контрольным сценам ромб с вершинами $P_8(15,20,20)$, $P_9(20,25,20)$, $P_{10}(25,20,20)$, $P_{11}(20,15,20)$. Изображайте содержимое $z$-буфера после обработки каждого многоугольника.

4.9. Реализуйте описанный в разд. 4.10 алгоритм построчного сканирования, использующий $z$-буфер. В качестве контрольных примеров возьмите те две сцены, которые были описаны выше. Результат изображайте построчно, строка за строкой. Предусмотрите возможность визуализации для каждой сканирующей строки списков активных многоугольников и ребер.

4.10. Реализуйте интервальный алгоритм построчного сканирования (Уоткинса), который описан в разд. 4.11. В качестве контрольных примеров используйте те две сцены, которые были указаны выше. Результат изображайте построчно, строка за строкой. Предусмотрите возможность визуализации для каждой сканирующей строки списка активных ребер и стека пересечений.

4.11. Реализуйте описанный в разд. 4.13 алгоритм трассировки лучей для случая непрозрачных поверхностей. Для проверки работы программы воспользуйтесь двумя описанными выше сценами. Считайте, что наблюдатель находится в бесконеч-

ности на положительной полуоси $z$. Изображайте результат пиксел за пикселом, по мере их формирования. Предусмотрите возможность визуализации при обработке каждого пиксела списка активных объектов. Повысьте эффективность этого алгоритма путем вычисления прямоугольной оболочки всей сцены и проецирования ее на картинную плоскость. Все пикселы, находящиеся за пределами области проекции, можно не обрабатывать. Сравните результаты работы алгоритма до и после этого изменения.

## К главе 5

Следующие задания обычно выполняются на растровом дисплее с 16 или более уровнями интенсивности или цветами. Чем больше уровней или цветов, тем лучше выглядят результаты. Для того чтобы выполнить задания на векторном дисплее, нужно перевести двоичное представление из $n$-разрядного буфера кадра в десятичное число и занести его в соответствующее место растра.

5.1. Рассмотрим $n$-гранное полигональное представление непрозрачного цилиндра радиуса $R$ с осью, перпендикулярной направлению наблюдения. Напишите программу, изображающую цилиндр радиуса $R = 15$, используя простую модель освещения, описываемую уравнением (5.1), и любой подходящий алгоритм удаления невидимых поверхностей для $n = 8, 16, 32$. Взгляните на рис. 5.3. Единственный источник света и наблюдатель расположены в бесконечности на положительной полуоси $z$. Поверните цилиндр на 90° вокруг оси $z$ и сравните результаты.

5.2. Закрасьте цилиндр из задачи 5.1 методом Гуро (разд. 5.5), используя простую модель освещения. Сравните результаты. Введите в модель освещения зеркальное отражение (см. ур. 5.7). Измените параметр $n$.

5.3. Закрасьте цилиндр из задачи 5.1 методом Фонга (разд. 5.6). Сравните результат с изображениями из задачи 5.1 и 5.2. Добавьте в модель зеркальное отражение (см. уравнение (5.7)). Сравните результаты, в частности форму зеркальных бликов, с закраской Гуро.

5.4. Для сцены из четырехугольника и треугольника (задачи 4.4—4.11) напишите программу, изображающую прозрачный треугольник (см. разд. 5.9) и непрозрачный четырехугольник методом Ньюэла — Ньюэла — Санча (см. разд. 4.8) или с помощью интервального алгоритма построчного сканирования (см. разд. 4.11). Преломление не учитывается. Интенсивность в местах наложения поверхностей рассчитывается как линейная комбинация видимой прозрачной поверхности и непрозрачной грани, лежащей за ней. Каковы будут результаты, если изменить коэффициент прозрачности? Сделайте прямоугольник прозрачным, а треугольник непрозрачным и сравните изображения.

5.5. Внесите в интервальный алгоритм построчного сканирования из задачи 4.10 построение теней. Пусть и треугольник, и четырехугольник непрозрачны. Наблюдатель расположен в бесконечности на положительной полуоси $z$. Один точечный источник находится в точке $x = 30$, $y = 40$. Как построить тень от прозрачного треугольника?

5.6. Внесите построение теней в алгоритм выделения видимых непрозрачных поверхностей трассировки лучей из задачи 4.11. Пусть треугольник и четырехугольник непрозрачны, наблюдатель находится в бесконечности на положительной полуоси $z$, а единственный источник — в точке $x = 30$, $y = 40$. Как построить тень от прозрачного треугольника?

5.7. Напишите программу нанесения узора, показанную на рис. 5.34, на октант сферы методом разбиения (см. разд. 5.11). С помощью наиболее подходящего алгоритма удаления невидимых граней изобразите результат на растре 32 × 32. Сравните его с результатом на растре 64 × 64.

5.8. Постройте глобальную модель освещения с алгоритмом трассировки лучей, описанную в разд. 5.12 (рис. 5.42—5.44). Для проверки программы используйте простую сцену из примера 5.9.

5.9. Напишите программу, рисующую на экране разноцветные квадраты разного размера. Измените программу так, чтобы цвет текущего квадрата можно было прибавлять, вычитать и заменять на цвет предыдущего. Проверьте с помощью этой программы аддитивную цветовую систему (см. разд. 5.15), например красный + синий = пурпурный. Исследуйте эффект одновременного контраста, изображая маленький пурпурный квадрат внутри ярко-красного и ярко-синего квадратов.

# ОГЛАВЛЕНИЕ

Предисловие редакторов перевода ..... 5
Предисловие к русскому изданию ..... 7
Предисловие ..... 9

ГЛАВА 1. ВВЕДЕНИЕ В МАШИННУЮ ГРАФИКУ ..... 13

1.1. Обзор машинной графики ..... 13
1.2. Типы графических устройств ..... 15
1.3. Графические дисплеи на запоминающей трубке ..... 16
1.4. Векторные графические дисплеи с регенерацией изображения .. 18
1.5. Растровые графические дисплеи с регенерацией изображения .. 23
1.6. Устройство электронно-лучевой трубки ..... 30
1.7. Устройство цветной растровой ЭЛТ ..... 31
1.8. Системы с телевизионным растром ..... 33
1.9. Диалоговые устройства ..... 36
1.10. Резюме ..... 46
1.11. Литература ..... 46

ГЛАВА 2. РАСТРОВАЯ ГРАФИКА ..... 48

2.1. Алгоритмы вычерчивания отрезков ..... 48
2.2. Цифровой дифференциальный анализатор ..... 50
2.3. Алгоритм Брезенхема ..... 54
2.4. Целочисленный алгоритм Брезенхема ..... 59
2.5. Общий алгоритм Брезенхема ..... 60
2.6. Алгоритм Брезенхема для генерации окружности ..... 63
2.7. Растровая развертка — способ генерация изображения ..... 73
2.8. Растровая развертка в реальном времени ..... 73
2.9. Групповое кодирование ..... 80
2.10. Клеточное кодирование ..... 83
2.11. Буферы кадра ..... 85
2.12. Адресация растра ..... 87
2.13. Изображение отрезков ..... 89
2.14. Изображение литер ..... 91
2.15. Растровая развертка сплошных областей ..... 92
2.16. Заполнение многоугольников ..... 93
2.17. Растровая развертка многоугольников ..... 94
2.18. Простой алгоритм с упорядоченным списком ребер ..... 97

2.19. Более эффективные алгоритмы с упорядоченным списком ребер ..... 99
2.20. Алгоритм заполнения по ребрам ..... 105
2.21. Алгоритм со списком ребер и флагом ..... 107
2.22. Алгоритмы заполнения с затравкой ..... 110
2.23. Простой алгоритм заполнения с затравкой ..... 111
2.24. Построчный алгоритм заполнения с затравкой ..... 114
2.25. Основы методов устранения ступенчатости ..... 119
2.26. Простой метод устранения лестничного эффекта ..... 123
2.27. Свертка и устранение ступенчатости ..... 127
2.28. Аппроксимация полутонами ..... 131
2.29. Литература ..... 139

ГЛАВА 3. ОТСЕЧЕНИЕ ..... 142

3.1. Двумерное отсечение ..... 143
3.2. Алгоритм отсечения Сазерленда — Коэна, основанный на разбиении отрезка ..... 153
3.3. Алгоритм разбиения средней точкой ..... 158
3.4. Обобщение: отсечение двумерного отрезка выпуклым окном ..... 166
3.5. Алгоритм Кируса — Бека ..... 170
3.6. Внутреннее и внешнее отсечение ..... 181
3.7. Определение факта выпуклости многоугольника и вычисление его внутренних нормалей ..... 182
3.8. Разбиение невыпуклых многоугольников ..... 187
3.9. Трехмерное отсечение ..... 188
3.10. Трехмерный алгоритм разбиения средней точкой ..... 192
3.11. Трехмерный алгоритм Кируса — Бека ..... 194
3.12. Отсечение в однородных координатах ..... 198
3.13. Определение выпуклости трехмерного тела и вычисление внутренних нормалей к его граням ..... 201
3.14. Разрезание невыпуклых тел ..... 203
3.15. Отсечение многоугольников ..... 206
3.16. Последовательное отсечение многоугольника — алгоритм Сазерленда — Ходжмана ..... 207
3.17. Невыпуклые отсекающие области — алгоритм Вейлера — Азертона ..... 220
3.18. Отсечение литер ..... 227
3.19. Литература ..... 228

ГЛАВА 4. УДАЛЕНИЕ НЕВИДИМЫХ ЛИНИЙ И ПОВЕРХНОСТЕЙ .. 230

4.1. Введение ..... 230
4.2. Алгоритм плавающего горизонта ..... 233
4.3. Алгоритм Робертса ..... 250
4.4. Алгоритм Варнока ..... 290
4.5. Алгоритм Вейлера — Азертона ..... 315
4.6. Алгоритм разбиения криволинейных поверхностей ..... 320
4.7. Алгоритм, использующий z-буфер ..... 321
4.8. Алгоритмы, использующие список приоритетов ..... 329

4.9. Алгоритмы построчного сканирования ..... 338
4.10. Алгоритм построчного сканирования, использующий z-буфер ..... 339
4.11. Интервальный алгоритм построчного сканирования ..... 345
4.12. Алгоритмы построчного сканирования для криволинейных поверхностей ..... 355
4.13. Алгоритм определения видимых поверхностей путем трассировки лучей ..... 360
4.14. Резюме ..... 373
4.15. Литература ..... 374

ГЛАВА 5. ПОСТРОЕНИЕ РЕАЛИСТИЧЕСКИХ ИЗОБРАЖЕНИЙ .... 377

5.1. Введение ..... 377
5.2. Простая модель освещения ..... 380
5.3. Определение нормали к поверхности ..... 386
5.4. Определение вектора отражения ..... 388
5.5. Закраска методом Гуро ..... 391
5.6. Закраска Фонга ..... 394
5.7. Простая модель освещения со специальными эффектами ..... 399
5.8. Более полная модель освещения ..... 401
5.9. Прозрачность ..... 410
5.10. Тени ..... 416
5.11. Фактура ..... 427
5.12. Глобальная модель освещения с трассировкой лучей ..... 437
5.13. Более полная глобальная модель освещения с трассировкой лучей ..... 454
5.14. Направления современных исследований ..... 457
5.15. Цвет ..... 458
5.16. Литература ..... 487
Приложение А. Псевдокод ..... 490
Приложение В. Задачи ..... 495

---

Учебное издание

Дэвид Роджерс

АЛГОРИТМИЧЕСКИЕ ОСНОВЫ МАШИННОЙ ГРАФИКИ

Заведующий редакцией чл.-корр АН СССР В. И. Арнольд
Зам. зав. редакцией А.С. Попов
Ст. научн. редактор И. А. Маховая. Мл. научн. редактор Л. А. Королева
Художник В. А. Медников. Художественный редактор В. И. Шаповалов
Технические редакторы Н. И. Борисова, Т.А. Чистякова
Корректоры Т. Е. Луганова, М. Г. Орлова
ИБ № 6679
Подписано к печати 27.10.89. Формат 60×90¹/₁₆. Бумага офсетная № 1. Гарнитура таймс. Печать офсетная. Объем 16,00. Усл. печ. л. 32,00 в т.ч. 0,50 п.л. вкл. Усл. кр.-отт. 65,00. Уч.-изд. л. 33,33. Изд. № 1/6187. Тираж 20000 экз. Зак.1378. Цена 2 р. 80 к.

ИЗДАТЕЛЬСТВО «МИР»
В/О «Совэкспорткнига» Государственного комитета СССР по печати.
129820, ГСП, Москва И-110, 1-й Рижский пер., 2.
Набрано в Межиздательском фотонаборном центре издательства «Мир»

Можайский полиграфкомбинат В/О «Совэкспорткнига»
Государственного комитета СССР по печати.
143200, Можайск, ул. Мира, 93.

---