Simulating Natural Phenomena for Computer Graphics

Ronald Fedkiw, Stanford University

ABSTRACT
Level set methods have gained popularity in a number of research areas from computational physics to computer vision to computer graphics. They provide for a robust representation of geometry that can be dynamically evolved by solving partial differential equations in a manner that has been fine tuned by a community of researchers over a number of years. Traditionally, simulation of natural phenomena has been the focus of the computational physics community, but more recently computer graphics researchers have become interested in these methods in part because water, smoke and fire are exciting elements in high demand for special effects in feature films. Although computer graphics is the "inverse problem" to computer vision and this chapter is genuinely more concerned with synthesis than acquisition, understanding the application of level set methods to the physics problems that motivated their invention should enable computer vision researchers to gain a deeper understanding and appreciation of these methods and the related underlying mathematics.

1 Introduction

Consider the digital representation of a real word object. For example, suppose one starts with a set of data points such as those shown in figure 1 (left) which were obtained from an MRI image of a rat brain. In [29], variational and partial differential equation (PDE) methods were proposed to solve this problem in the context of level set methods. The authors first constructed a surface energy for the variational formulation and then obtained the gradient flow of the energy functional. This was recast in a level set framework and used to"shrink wrap" an implicit surface around the data points as shown in figure 1 (right) reminiscent of snakes or active contours [16]. The virtue of the approach proposed in [29] is that one can blindly obtain acceptable results on difficult problems. However, this same blind application of the mathematics can sometimes lead to various difficulties such as algorithms that are too slow or those that do not perform well in certain situations. One reason to become well-versed in the PDE and level set technology is to overcome the difficulties commonly encountered when first devising methods of this type. For example, although the PDE based

method in [29] was found to be impractical due to the costly solution of the nonlinear parabolic PDE proposed in that paper, later consideration in [28] showed that one could obtain a similar solution by solving a more efficient hyperbolic PDE, or by using an even more efficient marching type method. Thus, before abandoning a classical PDE approach, one should carefully explore for options that, for example, can lead to orders of magnitude improvements in efficiency. This emphasizes the importance of a thorough understanding of PDE and level set methods to the modern day computer vision researcher.

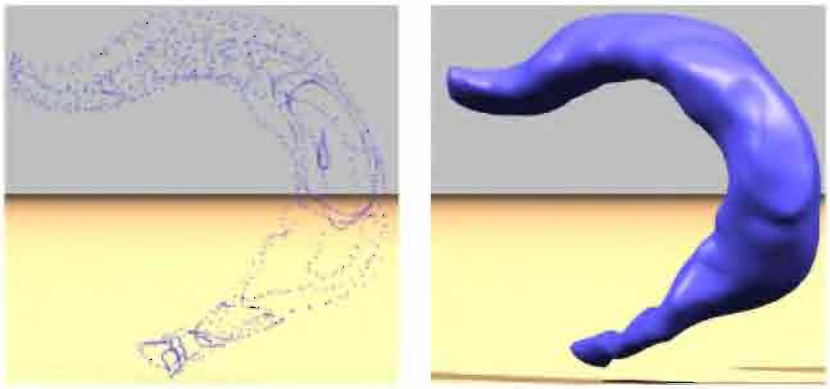

FIGURE 1. **Sample data points acquired from an MRI image of a rat brain (left), and a reconstructed level set representation (right).**

Although there is a rich field of computational fluid dynamics (CFD) in which many of the modern day algorithms for solving PDE's were and are still being developed, we instead present this chapter in the context of simulation of natural phenomena for computer graphics (CG) applications since the computer vision researcher will most likely already be familiar with, or even publish in, this area. However, we stress that the advanced computer vision researcher might wish to familiarize themselves with some of the underlying mathematical principles advocated in CFD, especially if the goal is to make clever observations such as those in [2] linking the Navier-Stokes equations for fluid flow to image inpainting.

We start with a general discussion of the equations of fluid flow discussing these equations in the context of smoke simulations. There we point out that one cannot always solve the equations exactly due limited computational resources. This is important to keep in mind when considering real-time or interactive-time applications. Then we discuss one particular way of modifying the equations in order to obtain the desired effect. Of course, the general problem of constructing equations for computational modeling drives a large portion of computer vision research. Next, we move on to two phase flows such as those involving liquids. Here, level set methods thrive both in the computational physics and the computer graphics communities. The characteristically smooth liquid surfaces that are both

physically accurate and visually pleasing are difficult to obtain with other interface tracking methods. Simulation of water is a fairly difficult problem demanding sophisticated methods for evolving surfaces highlighting the real potential of level set methods. Finally we move on to fire simulation which in some sense is still a partially unsolved problem in both graphics and physics (in physics one refers to combustion). In the context of fire simulation, we use the modeling tools discussed for smoke simulation, the advanced level set methods discussed for water simulation, and introduce new methods for modeling boundary conditions at discontinuities.

2 Smoke

The modeling of natural phenomena such as smoke remains a challenging problem in computer graphics. This is not surprising since the motion of gases is highly complex and turbulent. Visual smoke models have many obvious applications in the CG industry including special effects and interactive games. Ideally, a good CG smoke model should both be easy to use and produce highly realistic results. Obviously, the modeling of smoke and gases is of importance to other engineering fields as well. Only recently have researchers in computer graphics started to excavate the abundant CFD literature for algorithms that can be adopted and modified for CG applications. Unfortunately, most CG smoke models are either too slow or suffer from too much numerical dissipation. In [8], Fedkiw et al. proposed a method that exploits physics unique to smoke in order to design a numerical method that is both fast and efficient on the relatively coarse grids traditionally used in CG applications (as compared to the much finer grids used in the CFD literature).

In [8], the smoke's velocity was modeled with the inviscid incompressible Euler equations which are more appropriate for gas modeling and less computationally intensive than the full viscous Navier-Stokes equations. The advection part of these equations was solved using a semi-Lagrangian integration approach which is very popular in the atmospheric sciences community for modeling large scale flows dominated by constant advection where large time steps are desired, see e.g. [24] for a review. This was followed by a pressure-Poisson equation guaranteeing that the model is stable for any choice of the time step. One of the main contributions of [8] was to reduce the numerical dissipation inherent to semi-Lagrangian schemes by using a technique from the CFD literature known as "vorticity confinement" [25] to model the small scale rolling features characteristic of smoke that are usually absent on coarse grid simulations. The basic idea is to inject the energy lost due to numerical dissipation back into the fluid using a forcing term. This force is designed specifically to increase the vorticity of the flow, which visually keeps the smoke alive over time. This forcing term

is consistent with the Euler equations in the sense that it disappears as the number of grid cells is increased. In CFD this technique was applied to the numerical computation of complex turbulent flow fields around helicopters where it is not possible to add enough grid points to accurately resolve the flow field. The computation of the force only adds a small computational overhead, and consequently the resulting simulations are almost as fast as the one's obtained from the basic "stable fluids" semi-Lagrangian algorithm in [22] without the unappealing numerical dissipation. A sample calculation is shown in figure 2.

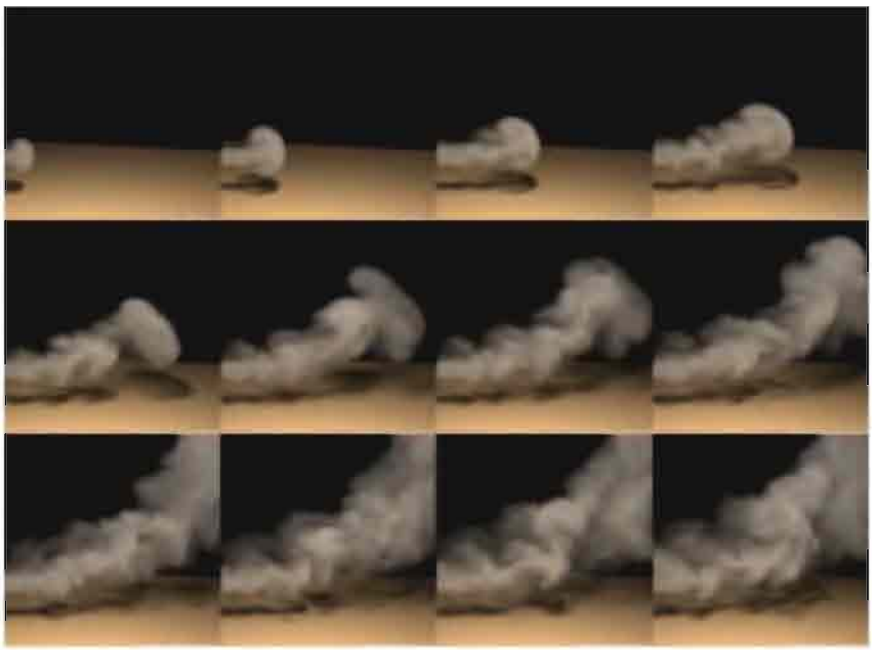

FIGURE 2. **A warm smoke plume injected from left to right rises under the effects of buoyancy. Note the high level of detailed rolling effects present in the figures.**

For simplicity, assume that our gases can be modeled as inviscid, incompressible, constant density fluids. The effects of viscosity are negligible in gases especially on coarse grids where numerical dissipation dominates physical viscosity and molecular diffusion. When the smoke's velocity is well below the speed of sound, the compressibility effects are negligible as well, and the assumption of incompressibility greatly simplifies the numerical methods. Consequently, the equations that model the smoke's velocity, $\vec{V} = (u, v, w)$, are given by the incompressible Euler equations [17]

$$\nabla \cdot \vec{V} = 0 \tag{1.1}$$

$$\frac{\partial \vec{V}}{\partial t} = -(\vec{V} \cdot \nabla)\vec{V} - \nabla p + \vec{f}. \tag{1.2}$$

These two equations state that the velocity should conserve both mass (equation 1.1) and momentum (equation 1.2). p is the pressure of the gas and $\vec{f}$ accounts for external forces. The constant density of the fluid has been arbitrarily rescaled to unity. As in [10, 11, 22], [8] solved these equations in two steps. First an intermediate velocity field $\vec{V}^*$ is computed solving equation 1.2 over a time step $\triangle t$ without the pressure term

$$\frac{\vec{V}^* - \vec{V}}{\triangle t} = -(\vec{V} \cdot \nabla)\vec{V} + \vec{f}. \tag{1.3}$$

After this the velocity field $\vec{V}^*$ is forced to be incompressible using a projection method [4]. This is equivalent to computing the pressure from the following Poisson equation

$$\nabla^2 p = \frac{1}{\triangle t} \nabla \cdot \vec{V}^* \tag{1.4}$$

with pure Neumann boundary condition, i.e., $\frac{\partial p}{\partial \vec{N}} = 0$ at a boundary point with normal $\vec{N}$. Note that it is also straightforward to impose Dirichlet boundary conditions where the pressure is specified directly as opposed to specifying its normal derivative. The intermediate velocity field is then made incompressible by subtracting the gradient of the pressure from it

$$\vec{V} = \vec{V}^* - \triangle t \nabla p. \tag{1.5}$$

Besides evolving the velocity field, additional equations for the evolution of both the temperature T and the smoke's density ρ are needed. Assuming that these two scalar quantities are simply advected according to the smoke's velocity leads to

$$\frac{\partial T}{\partial t} = -(\vec{V} \cdot \nabla)T, \tag{1.6}$$

and

$$\frac{\partial \rho}{\partial t} = -(\vec{V} \cdot \nabla)\rho. \tag{1.7}$$

Both the density and the temperature affect the fluid's velocity. Heavy smoke tends to fall downwards due to gravity while hot gases tend to rise due to buoyancy. A simple model that accounts for these effects can be constructed by defining external forces that are directly proportional to the density and the temperature

$$\vec{f}_{buoy} = -\alpha\rho\vec{z} + \beta(T - T_{amb})\vec{z}, \tag{1.8}$$

where $\vec{z} = (0, 0, 1)$ points in the upward vertical direction, T_{amb} is the ambient temperature of the air and α and β are two positive constants

with appropriate units. Note that when $\rho = 0$ and $T = T_{amb}$, this force is identically zero.

Usually smoke and air mixtures contain velocity fields with large spatial deviations accompanied by a significant amount of rotational and turbulent structure on a variety of scales. Nonphysical numerical dissipation damps out these interesting flow features, and the goal of the approach in [8] is to add them back on the coarse grid. One way of adding them back would be to create a random or pseudo-random small scale perturbation of the flow field using either a heuristic or physically based model. For example, one could generate a divergence free velocity field using a Kolmogorov spectrum and add this to the computed flow field to represent the missing small scale structure (see [23] for some CG applications of the Kolmogorov spectrum). While this provides small scale detail to the flow, it does not place the small scale details in the physically correct locations within the flow field where they are missing. Instead, the details are added in a haphazard fashion and the smoke can appear to be "alive", rolling and curling in a nonphysical fashion. The key to realistic animation of smoke is to make it look like a passive natural phenomena as opposed to a "living" creature made out of smoke.

The method proposed in [8] looks for the locations within the flow field where small scale features should be generated and adds the small scale features in these locations in a physically based fashion that promotes the passive rolling of smoke that gives it a realistic turbulent look on a coarse CG grid. With unlimited computing power, any consistent numerical method could be used to obtain acceptable results simply by increasing the number of grid points until the desired limiting behavior is observed. However, in practice, computational resources are limited, grids are fairly coarse (even coarser in CG than in CFD), and the discrete difference equations may not be asymptotically close enough to the continuous equations for a particular simulation to behave in the desired physically correct fashion. The key idea is to design a consistent numerical method that behaves in an interesting and physically plausible fashion on a coarse grid. In general, this is very difficult to do, but luckily a vorticity confinement method was recently invented by Steinhoff, see e.g. [25], for the numerical computation of complex turbulent flow fields around helicopters where it is not possible to add enough grid points to accurately resolve the flow.

The first step in generating the small scale detail is to identify where it comes from. In incompressible flow, the vorticity

$$\vec{\omega} = \nabla \times \vec{V} \tag{1.9}$$

provides the small scale structure. Each small piece of vorticity can be thought of as a paddle wheel trying to spin the flow field in a particular direction. Artificial numerical dissipation damps out the effect of these paddle wheels, and the key idea is to simply add it back. First normalized

vorticity location vectors

$$\vec{N} = \frac{\vec{\eta}}{|\vec{\eta}|}, \qquad (\vec{\eta} = \nabla|\vec{\omega}|) \tag{1.10}$$

that point from lower vorticity concentrations to higher vorticity concentrations are computed. Then the magnitude and direction of the paddle wheel force is computed as

$$\vec{f}_{conf} = \epsilon \triangle x(\vec{N} \times \vec{\omega}) \tag{1.11}$$

where $\epsilon > 0$ is used to control the amount of small scale detail added back into the flow field and the dependence on the spatial discretization $\triangle x$ guarantees that as the mesh is refined the physically correct solution is still obtained.

In [8], a finite volume spatial discretization was used to numerically solve the equations. The computational domain was divided into identical voxels with the temperature, the smoke's density and the external forces defined at the center of each voxel while the velocity is defined on the appropriate voxel faces. This staggered MAC grid arrangement of the velocity field is standard for incompressible flows, see e.g. [13]. To handle boundaries immersed in the fluid all voxels that intersect an object are tagged as being occupied, and all occupied voxel cell faces have their velocity set to that of the object. Similarly, the temperature at the center of the occupied voxels is set to the object's temperature. Consequently an animator can create many interesting effects by simply moving or heating up an object. The smoke's density is of course equal to zero inside the object. However, to avoid a sudden drop-off of the density near the object's boundary, the density at boundary voxels is set equal to the density of the closest unoccupied voxel.

The fluid velocity is updated in three steps. First, the advection term in equation 1.3 is solved for using a semi-Lagrangian scheme. The semi-Lagrangian algorithm builds a new grid of velocities from the ones already computed by backward tracing the midpoints of each voxel face through the velocity field. New velocities are then interpolated at these points and their values are transferred to the face cells they originated from. It is possible that the point ends up in one of the occupied voxels, and in this case one simply clips the path against the object's boundary. This guarantees that the point always lies in the unoccupied fluid. Simple linear interpolation is easy to implement and combined with the vorticity confinement force gives satisfactory results. Next, the force fields are added to the velocity grid. These include user supplied fields, the buoyancy force defined by equation 1.8 and the confinement force defined by equation 1.11. This is done by simply multiplying each force by the time step and adding it to the velocity. Finally, the velocity field is made incompressible to conserve mass. As stated above, this involves the solution of a Poisson equation for the pressure (equation 1.4). The discretization of this equation results in a

sparse linear system of equations. Free Neumann boundary conditions can be applied at the occupied voxels by setting the normal pressure gradient equal to zero at the occupied boundary faces. The system of equations is symmetric, and the most natural linear solver in this case is the conjugate gradient method which is easy to implement and has much better convergence properties than simple relaxation methods. An incomplete Choleski preconditioner is recommended to improve convergence. These techniques are quite standard and the reader is referred to [12] for more details. In practice, only about 20 iterations of this solver gives visually acceptable results, although the solution is not usually converged at this point. After the pressure is computed, its gradient is subtracted from the velocity, as mentioned above. After the velocity is updated, both the temperature and the smoke's density are solved for, again using the semi-Lagrangian scheme.

3 Water

Water surrounds us in our everyday lives. Given the ubiquity of water and our constant interaction with it, the animation and rendering of water poses one of the greatest challenges in computer graphics. The difficulty of this challenge was underscored recently through the use of water effects in a variety of motion pictures including the recent feature film "Shrek" where water, mud, beer and milk effects were seen. In response to a question concerning what was the single hardest shot in "Shrek", DreamWorks SKG principal and producer of "Shrek", Jeffrey Katzenberg, stated, *It's the pouring of milk into a glass.* [14]. This illustrates the need for photorealistic simulation and rendering of water (and other liquids such as milk), especially in the case of complex, three dimensional behavior as seen when water is poured into a glass as in figure 3. A key to achieving this goal is the visually accurate treatment of the surface separating the water from the air. The behavior of this surface provides the visual impression that water is being seen. If the numerical simulation method used to model this surface is not robust enough to capture the essence of water, then the effect is ruined for the viewer.

In [9], Foster and Fedkiw noted that the level set method suffered from excessive numerical dissipation on coarse grids resulting in a visually disturbing nonphysical loss of water volume during the course of the simulation. They realized that this volume loss could be alleviated in part by introducing marker particles near the interface and by using these marker particles to correct errors in the level set function in regions of high curvature. Later, in [5], the authors worked to make the approach advocated in [9] computational accurate enough to be applied to flow physics problems. Notably, both of these papers were inspired by the control problems under consideration by the second and fourth authors in [5]. The approach ad-

vocated in [5] can be thought of as a "thickened" front tracking approach to modeling the surface and was called the"particle level set method". The improvements in [5] (over [9]) were achieved by focusing on modeling the surface as opposed to the liquid volume itself as was done in [9].

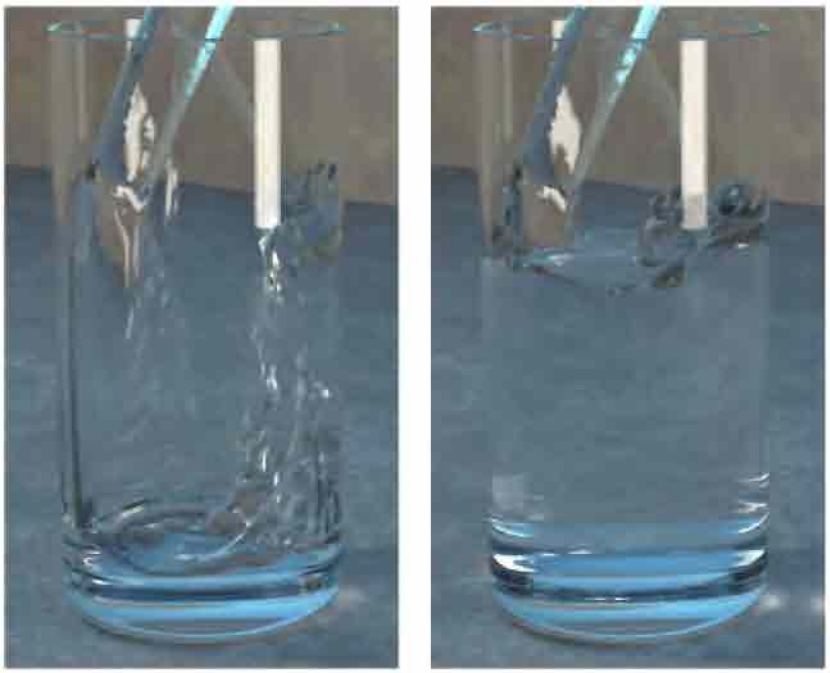

FIGURE 3. **Water being poured into a glass (55x120x55 grid cells).**

This shift in philosophy away from volume modeling and towards surface modeling is the key idea behind [6] resulting in better photorealistic behavior of the water surface. For example, [6] proposed a new treatment of the velocity at the surface in order to obtain more visually realistic water surface behavior. The motion of both the massless marker particles and the implicit function representing the surface is dependent upon the velocities contained on the underlying computational mesh. By extrapolating velocities across the water surface and into the region occupied by the air, more accurate and better visual results were obtained. In the limit as the computational grid is refined, the resulting surface condition is identical to the traditional approach of making the velocity divergence free, but it gives more visually appealing and physically plausible results on coarse grids. Furthermore, this velocity extrapolation procedure allows a degree of control to be applied to the behavior of the water surface. For example, one can add dampening and/or churning effects forcing it to quiet down or splash up faster than would be allowed by a straightforward physical simulation.

The liquid volume was simulated as one side of an isocontour of an im-

plicit level set function, ϕ. The surface of the water is defined by the $\phi = 0$ isocontour with $\phi < 0$ representing the water and $\phi > 0$ representing the air. By using an implicit function representation of the liquid volume, one obtains a smooth, temporally coherent liquid surface. [9] rejected the use of particles alone to represent the liquid surface because it is difficult to a calculate a visually desirable smooth liquid surface from the discrete particles alone. The implicit surface is dynamically evolved in space and time according to the underlying liquid velocity $\vec{V}$. The appropriate equation to do this is

$$\phi_t + \vec{V} \cdot \nabla\phi = 0. \tag{1.12}$$

A level set only approach to modeling the surface will not yield realistic surface behavior due to an excessive amount of volume loss on coarse grids. A seminal advance of [9] in creating realistic liquids for computer animation is the hybridization of the visually pleasing smooth implicit function modeling of the liquid volume with particles that can maintain the liquid volume on coarse grids. The inclusion of particles provides a way for capturing the liveliness of a real liquid with spray and splashing effects. Curvature was used as an indicator for allowing particles to influence the implicit surface representation of the water. This is a natural choice since small droplets of water have very high curvature and dynamic implicit surfaces have difficulty resolving features with sharp corners.

Figure 4 (left) shows the initial data for a notched disk that was rotated for one rigid body rotation about the point (50,50). Figure 4 (right) shows the result obtained using a level set only approach where both the higher and lower corners of the disk are erroneously shaved off causing both loss of visual features and an artificially viscous look for a liquid. This numerical result was obtained using a highly accurate fifth order WENO discretization of equation 1.12 (see e.g. [15, 21]). Figure 5 (left) shows the result obtained with the method from [9]. The particles inside the disk do not allow the implicit surface to cross over them and help to preserve the two corners near the bottom. However, there is little they can do to stop the implicit surface from drifting away from them near the top corners. This represents loss of air or bubble volume as the method erroneously gains liquid volume. This is not desirable since many complex water motions such as wave phenomenon are due in part to differing heights of water columns adjacent to each other. Loss of air in a water column reduces the pressure forces from neighboring columns destroying many of the dynamic splashing effects as well as the overall visually stimulating liveliness of the liquid. While the hybrid liquid volume model of [9] attempts to maintain the volume of the liquid accurately, it fails to model the air or more generally the opposite side of the liquid surface. [6] shifts the focus away from maintaining a liquid volume towards maintaining the liquid surface itself. An immediate result of this approach is that it leads to symmetry in particle placement. That is, particles are placed on both sides of the surface and used to maintain

an accurate representation of the surface itself regardless of what may be on one side or the other. The particles are not meant to track volume, they are intended to correct errors in the surface representation by the implicit level set function. See [5] for more details. Figure 5 (right) shows that this new method correctly computes the rigid body rotation for the notched disk preserving both the water and the air volumes so that more realistic water motion can be obtained.

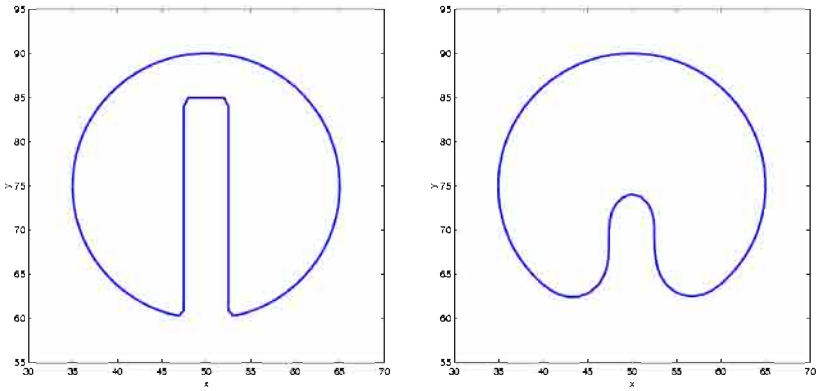

FIGURE 4. **Initial data (left) and solution after one rotation (right) using the level set only method.**

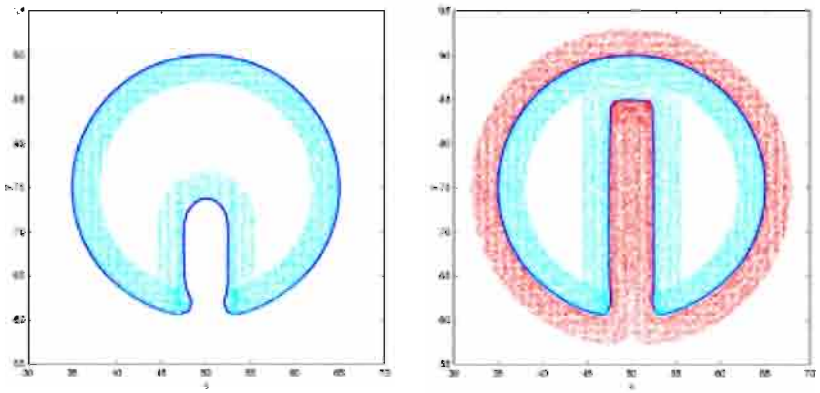

FIGURE 5. **Solution after one rotation using particles on the inside only (left) and using particles on both sides (right).**

In the particle level set method, two sets of particles are randomly placed in a "thickened" surface region (e.g. three grid cells on each side of the surface) with *positive* particles in the $\phi > 0$ region and *negative* particles in the $\phi < 0$ region. There is no need to place particles far away from the surface since the sign of the level set function readily identifies these regions gaining a large computational savings. The number of particles placed in each cell is an adjustable parameter that can be used to control the amount of resolution, e.g. 64 particles per cell works well in three spatial dimensions.

Each particle possesses a radius, r_p, which is constrained to take a minimum and maximum value based upon the size of the underlying computational cells used in the simulation. A minimum radius of $.1\min(\triangle x, \triangle y, \triangle z)$ and maximum radius of $.5\min(\triangle x, \triangle y, \triangle z)$ appear to work well. The radius of a particle changes dynamically throughout the simulation, since a particle's location relative to the surface changes. The radius is set according to:

$$r_p = \begin{cases} r_{max} & \text{if } s_p\phi(\vec{x}_p) > r_{max} \\ s_p\phi(\vec{x}_p) & \text{if } r_{min} \leq s_p\phi(\vec{x}_p) \leq r_{max} \\ r_{min} & \text{if } s_p\phi(\vec{x}_p) < r_{min} \end{cases}, \tag{1.13}$$

where s_p is the sign of the particle (+1 for positive particles and -1 for negative particles). This radius adjustment keeps the boundary of the spherical particle tangent to the surface whenever possible. This fact combined with the overlapping nature of the particle spheres allows for an enhanced reconstruction of the liquid surface.

The marker particles and the implicit function are separately integrated forward in time using a forward Euler time integration scheme. The implicit function is integrated forward using equation 1.12, while the particles are passively advected with the flow using $d\vec{x}_p/dt = \vec{V}_p$, where $\vec{V}_p$ is the fluid velocity interpolated to the particle position $\vec{x}_p$.

The main role of the particles is to detect when the implicit surface has suffered inaccuracies due to the coarseness of the computational grid in regions with sharp features. Particles that are on the wrong side of the interface by more than their radius, as determined by a locally interpolated value of ϕ at the particle position $\vec{x}_p$, are considered to have *escaped* their side of the interface. This indicates potential errors in the implicit surface representation of the interface. In smooth, well resolved regions of the interface, our implicit level set surface representation is highly accurate and particles do not drift a non-trivial distance across the interface. We associate a spherical implicit function, designated ϕ_p, with each particle p whose size is determined by the particle radius, i.e.

$$\phi_p(\vec{x}) = s_p(r_p - |\vec{x} - \vec{x}_p|). \tag{1.14}$$

Any difference in ϕ from ϕ_p indicates potential errors in the implicit function representation of the surface. That is, the implicit version of the surface and the particle version of the surface disagree. Escaped positive particles are used to rebuild the $\phi > 0$ region and escaped negative particles are used to rebuild the $\phi < 0$ region. The reconstruction of the implicit surface occurs locally within the cell that each escaped particle currently occupies. Using equation 1.14, the ϕ_p values of escaped particles are calculated for the eight grid points on the boundary of the cell containing the particle. This value is compared to the current value of ϕ for each grid point and the smaller value (in magnitude) is taken, which is the value closest to the $\phi = 0$ isocontour defining the surface. This is done for all escaped positive

and all escaped negative particles. The result is an improved representation of the surface of the liquid. This error correction method is applied after every computational step in which ϕ has been modified in some way. This occurs when ϕ is integrated forward in time and when the implicit function is reinitialized to a signed distance function using

$$\phi_\tau = -S(\phi_{\tau=0})(|\nabla\phi| - 1), \tag{1.15}$$

where τ is a fictitious time and $S(\phi)$ is a smoothed signed distance function given by

$$S(\phi) = \frac{\phi}{\sqrt{\phi^2 + (\triangle x)^2}}. \tag{1.16}$$

More details on this reinitialization equation are given in [26].

In complex flows, a liquid interface can be stretched and torn in a dynamic fashion. The use of only an initial seeding of particles will not capture these effects well, as regions will form that lack a sufficient number of particles to adequately perform the error correction step. Periodically, e.g. every 20 frames, particles are randomly reseeded about the "thickened" interface to avoid this dilemma. This is done by randomly placing particles near the interface, and then using geometric information contained within the level set function (e.g. the direction of the shortest possible path to the surface is given by $\vec{N} = \nabla\phi/|\nabla\phi|$) to move the particles to their respective domains, $\phi > 0$ or $\phi < 0$. The goal of this reseeding step is to preserve the initial particle resolution of the interface, e.g. 64 particles per cell. Thus, if a given cell has too few or too many particles, some can be added or deleted respectively.

Although the Navier-Stokes equations can be used to find the velocity within the liquid volume, boundary conditions are needed for the velocity on the air side near the free surface. These boundary condition velocities are used in updating the Navier-Stokes equations, moving the surface, and moving the particles placed near the surface. The velocity at the free surface of the water can be determined through the usual enforcement of the conservation of mass (volume) constraint, $\nabla \cdot \vec{V} = 0$. This equation allows one to determine the velocities on all the faces of computational cells that contain the $\phi = 0$ isocontour. Unfortunately, the procedure for doing this is not unique when more than one face of a cell needs a boundary condition velocity. A variety of methods have been proposed, e.g. see [3] and [11]. [6] proposed a different approach altogether, the extrapolation of the velocity across the surface into the surrounding air. As the computational grid is refined, this method is equivalent to the standard one, but it gives a smoother and more visually pleasing motion of the surface on coarser (practical sized) grids. [6] extrapolated the velocity out a few grid cells into the air, obtaining boundary condition velocities in a band of cells on the air side of the surface. This allows one to use higher order accurate methods

and to obtain better results when moving the implicit surface using equation 1.12, and also provides velocities for updating the position of particles on the air side of the surface. Velocity extrapolation also assists in the implementation of the semi-Lagrangian "stable fluid" method, since there are times when characteristics generated by this approach look back across the interface (a number of cells) into the air region for valid velocities.

The equation modeling this extrapolation for the x component of the velocity, u, is given by

$$\frac{\partial u}{\partial \tau} = -\vec{N} \cdot \nabla u, \tag{1.17}$$

where $\vec{N}$ is the unit normal vector perpendicular to the implicit surface and τ is fictitious time. A similar equation holds for the v and w components of the velocity field. Fast methods exist for solving this equation in $O(n \log n)$ time, where n is the number of grid points that one needs to extrapolate over, in our case a five grid cell thick band on the air side of the interface. The fast method is based upon the observation that information in equation 1.17 propagates in only one direction away from the surface. This implies that one does not have to revisit previously computed values of $\vec{u}_{ext}$ (the extrapolated velocity) if the calculation is preformed in the correct order. The order is determined by the value of ϕ allowing an $O(n \log n)$ sorting of the points before beginning the calculation. The value of u itself is determined by enforcing the condition at steady state, namely $\nabla\phi \cdot \nabla u = 0$ where the derivatives are determined using previously calculated values of ϕ and u. From this scalar equation, a new value of u can be determined, and then one can proceed to the next point corresponding to the next smallest value of ϕ, etc. Further details of this method are discussed in [1].

This velocity extrapolation method enabled [6] to apply a newly devised method for controlling the nature of the surface motion. This was done simply by modifying the extrapolated velocities on the air side of the surface. For example, to model wind-blown water as a result of air drag, one takes a convex combination of the extrapolated velocities with a pre-determined wind velocity field

$$\vec{V} = (1-\alpha)\vec{V}_{ext} + \alpha\vec{V}_{wind}, \tag{1.18}$$

where $\vec{V}_{ext}$ is the extrapolated velocity, $\vec{V}_{wind}$ a desired air-like velocity, and $0 \leq \alpha \leq 1$ the mixing constant. This can be applied throughout the surface or only locally in select portions of the computational domain as desired. Note that setting $\vec{V}_{wind} = 0$ forces churning water to settle down faster with the fastest settling resulting from $\alpha = 1$.

4 Fire

The modeling of natural phenomena such as fire and flames remains a challenging problem in computer graphics. Although simulations of fluid behavior are in demand for special effects, fire effects are especially in demand due to the dangerous nature of this phenomenon. Fire simulations are also of interest for virtual reality effects, for example to help train fire fighters or to determine proper placement of exit signs in smoke filled rooms (i.e. so they can be seen). Combustion processes can be loosely classified into two rather distinct types of phenomena: detonations and deflagrations. In both of these processes, chemical reactions convert fuel into hot gaseous products. Deflagrations are low speed events such as the fire and flames we address in this section, while detonations are high speed events such as explosions where shock waves and other compressible effects are important. As low speed events, deflagrations can be modeled using the equations for incompressible flow (as opposed to those for compressible flow). Furthermore, since viscous effects are small, the incompressible inviscid Euler equations can be used.

An important, often neglected aspect of fire and flame modeling concerns the expansion of the fuel as it reacts to form hot gaseous products. This expansion is the reason for the visual fullness observed in many flames and is partly responsible for the visual turbulence as well. Since the incompressible equations do not account for expansion, [19] proposed a simple thin flame model for capturing these effects. This was accomplished by using an implicit surface to represent the reaction zone where the gaseous fuel is converted into hot gaseous products. Although real reaction zones have a nonzero (but small) thickness, the thin flame approximation works well for visual modeling and has been used by scientists as well, see for example [18] which first proposed this methodology. [19] used a dynamic implicit surface to track the reaction zone where the gaseous fuel is converted into hot gaseous products. Then both the gaseous fuel and the hot gaseous products were separately modeled using independent sets of incompressible flow equations. Finally, these incompressible flow equations were updated together in a coupled fashion using the fact that both mass and momentum must be conserved as the gas reacts at the interface. While this gives rather pleasant looking laminar (smooth) flames, a vorticity confinement term was included to model the larger scale turbulent flame structures that are difficult to capture on the relatively coarse grids.

There are three basic visual phenomena associated with flames. The first of these is the blue or bluish-green core seen in many flames. These colors are emission lines from intermediate chemical species, such as carbon radicals, produced during the chemical reaction. In the thin flame model, this thin blue core is located adjacent to the implicit surface. Therefore, in order to track this blue core, one needs to track the movement of the implicit surface. The second visual phenomenon is the blackbody radia-

tion emitted by the hot gaseous products, in particular the carbon soot. This is characterized by the yellowish-orange color familiarly associated with fire. In order to model this with visual accuracy one needs to track the temperatures associated with a flame. If the fuel is solid or liquid, the first step is the heating of the solid until it undergoes a phase change to the gaseous state. (Obviously, for gas fuels, we start in this gaseous state.) Then the gas heats up until it reaches its ignition temperature corresponding to the implicit surface and the beginning of the thin blue core region. The temperature continues to increase as the reaction proceeds reaching a maximum before radiative cooling and mixing effects cause the temperature to decrease. As the temperature decreases, the blackbody radiation falls off until the yellowish-orange color is no longer visible. The third and final visual effect addressed is the smoke or soot that is apparent in some flames after the temperature cools to the point where the blackbody radiation is no longer visible. This effect is modeled by carrying along a density variable in a fashion similar to the temperature.

In [19] the implicit surface separates the gaseous fuel from the hot gaseous products and surrounding air. Consider for example the injection of gaseous fuel from a cylindrically shaped tube. If the fuel were not burning, then the implicit surface would simply move at the same velocity as the gaseous fuel being injected. However, when the fuel is reacting, the implicit surface moves at the velocity of the unreacted fuel plus a flame speed S that indicates how fast fuel is being converted into gaseous products. S indicates how fast the unreacted gaseous fuel is crossing over the implicit surface turning into hot gaseous products. The approximate surface area of the blue core, A_S, can be estimated with the following equation

$$v_f A_f = S A_S, \tag{1.19}$$

where v_f is the speed the fuel is injected across the injection surface with area A_f, e.g. A_f is the cross section of the cylindrical tube. This equation results from canceling out the density in the equation for conservation of mass. The left hand side is the fuel being injected into the region bounded by the implicit surface, and the right hand side is the fuel leaving this region crossing over the implicit surface as it turns into gaseous products. From this equation, we see that injecting more (less) gas is equivalent to increasing (decreasing) v_f resulting in a larger (smaller) blue core. Similarly, increasing (decreasing) the reaction speed S results in a smaller (larger) blue core. While one can turn the velocity up or down on a cylindrical jet, the reaction speed S is a property of the fuel. For example, S is approximately $.44m/s$ for a propane fuel that has been suitably premixed with oxidizer [27]. (Note that while this thin flame approximation is fairly accurate for premixed flames, diffusion flames behave somewhat differently.)

In order to get the proper visual look for flames, it is important to track individual elements of the flow and follow them through their temperature histories. This is particularly difficult because the gas expands as it

undergoes reaction from fuel to hot gaseous products. This expansion is important to model since it changes the trajectories of the gas and the subsequent look and feel of the flame as individual elements go through their temperature profile. Individual elements do not go straight up as they pass through the reaction front, but instead turn outward due to the effects of expansion. It is difficult to obtain visually full turbulent flames without modeling this expansion effect. In fact, many practitioners resort to a number of low level hacks (and lots of random numbers) in an attempt to sculpt this behavior. In contrast, [19] obtained this behavior by modeling the expansion directly.

[19] used one set of incompressible flow equations to model the fuel with density ρ_f and a separate set of incompressible flow equations to model the hot gaseous products and surrounding airflow with density ρ_h. This requires a model for coupling these two sets of incompressible flow equations together across the interface in a manner that models the expansion that takes place across the reaction front. Given that mass and momentum are conserved one can derive the following equations for the coupling across the thin flame front:

$$\rho_h(V_h - D) = \rho_f(V_f - D), \tag{1.20}$$

$$\rho_h(V_h - D)^2 + p_h = \rho_f(V_f - D)^2 + p_f, \tag{1.21}$$

where V_f and V_h are the normal velocities of the fuel and the hot gaseous products, and p_f and p_h are their pressures. Here, $D = V_f - S$ is the speed of the implicit surface in the normal direction. These equations indicate that both the velocity and the pressure are discontinuous across the flame front. Thus, one needs to exercise caution when taking derivatives of these quantities as is required when solving the incompressible flow equations. (Note that the tangential velocities are continuous across the flame front.)

When considering solid fuels, there are two expansions that need to be accounted for. Besides the expansion across the flame front, a similar expansion takes place when the solid is converted to a gas. However, S is usually relatively small for this reaction (most solids burn slowly in a visual sense), so we can use the boundary of the solid fuel as the reaction front. Since [19] did not model the pressure in solids, only equation 1.20 applies. This equation can be rewritten as

$$\rho_f(V_f - D) = \rho_s(V_s - D), \tag{1.22}$$

where ρ_s and V_s are the density and the normal velocity of the solid fuel. Substituting $D = V_s - S$ and solving for V_f gives

$$V_f = V_s + (\rho_s/\rho_f - 1)\, S \tag{1.23}$$

indicating that the gasified solid fuel moves at the velocity of the solid fuel plus a correction that accounts for the expansion. [19] modeled this phase

change by injecting gas out of the solid fuel at the appropriate velocity. This can be used to set arbitrary shaped solid objects on fire as long as they can be voxelized with a suitable surface normal assigned to each voxel indicating the direction of gaseous injection. The ability to inject (or not inject) gaseous fuel out of individual voxels on the surface of a complex solid object allows one to animate objects catching on fire, burn different parts of an object at different rates or not at all (by using spatially varying injection velocities), and extinguish solid fuels simply by turning off the injection velocity. Figure 6 shows a ball catching on fire as it passes through a flame.

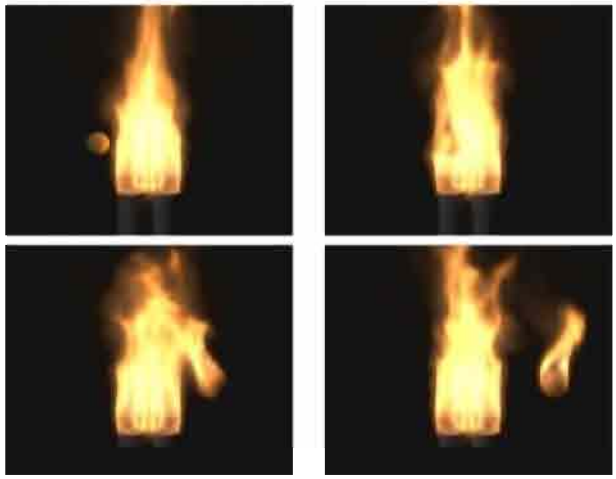

FIGURE 6. **A fiammable ball passes through a gas fiame and catches on fire.**

[19] tracked the reaction zone (blue core) using the level set method to track the moving implicit surface with ϕ positive in the region of space filled with fuel, negative elsewhere and zero at the reaction zone. The implicit surface moves with velocity $\vec{W} = \vec{V_f} + S\vec{N}$ where $\vec{V_f}$ is the velocity of the gaseous fuel and the $S\vec{N}$ term governs the conversion of fuel into gaseous products. Standard averaging of voxel face values is used to define $\vec{V_f}$ at the voxel centers. The motion of the implicit surface is defined through $\phi_t = -\vec{W} \cdot \nabla\phi$.

When solving the two sets of incompressible Euler equations, the equations for the velocity

$$\vec{V_t} = -\left(\vec{V} \cdot \nabla\right)\vec{V} - \nabla p/\rho + \vec{f} \tag{1.24}$$

are solved for in two parts. First, this equation is used to compute an intermediate velocity $\vec{V}^*$ ignoring the pressure term, and then pressure (correction) term is added using

$$\vec{V} = \vec{V}^* - \triangle t \nabla p/\rho. \tag{1.25}$$

The key idea to this splitting method is illustrated by taking the divergence of equation 1.25 to obtain

$$\nabla \cdot \vec{V} = \nabla \cdot \vec{V}^* - \triangle t \nabla \cdot (\nabla p/\rho) \tag{1.26}$$

and then realizing that we want $\nabla \cdot \vec{V} = 0$ to enforce mass conservation. Thus the left hand side of equation 1.26 should vanish leaving a Poisson equation of the form

$$\nabla \cdot (\nabla p/\rho) = \nabla \cdot \vec{V}^*/\triangle t \tag{1.27}$$

that can be solved to find the pressure needed for updating equation 1.25.

Since there are two sets of incompressible flow equations, we need to address the semi-Lagrangian update when a characteristic traced back from one set of incompressible flow equations crosses the implicit surface and queries the velocities from the other set of incompressible flow equations. Since the normal velocity is discontinuous across the interface, the straightforward approach fails. Instead, we need to use the balance equation 1.20 for conservation of mass to correctly interpolate a velocity. Suppose we are solving for the hot gaseous products and we interpolate across the interface into a region where a velocity from the gaseous fuel might incorrectly be used. Instead of using this value, we compute a ghost value as follows. First, we compute the normal velocity of the fuel, $V_f = \vec{V}_f \cdot \vec{N}$. Then we use the balance equation 1.20 to find a ghost value for V_h^G as

$$V_h^G = V_f + (\rho_f/\rho_h - 1)\, S. \tag{1.28}$$

Since the tangential velocities are continuous across the implicit surface, we combine this new normal velocity with the existing tangential velocity to obtain

$$\vec{V}_h^G = V_h^G \vec{N} + \vec{V}_f - (\vec{V}_f \cdot \vec{N})\vec{N} \tag{1.29}$$

as a ghost value for the velocity of the hot gaseous products in the region where only the fuel is defined. This ghost velocity can then be used to correctly carry out the semi-Lagrangian update. Since both $\vec{N}$ and $\vec{V}_f$ are defined throughout the region occupied by the fuel, and ρ_f, ρ_h and S are known constants, a ghost cell value for the hot gaseous products, $\vec{V}_h^G$, can be found anywhere in the fuel region (even quite far from the interface) by simply algebraically evaluating the right hand side of equation 1.29. [20] showed that this ghost fluid method, invented in [7], could be used to compute physically accurate engineering simulations of deflagrations.

After computing the intermediate velocity $\vec{V}^*$ for both sets of incompressible flow equations, equation 1.27 is solved for the pressure and finally equation 1.25 is used to find the new velocity field. Equation 1.27 is solved by assembling and solving a linear system of equations for the pressure as discussed in more detail in [9]. Once again, caution needs to be exercised

here since the pressure is discontinuous across the interface. Using the ghost fluid method and equation 1.21, one can obtain and solve a slightly modified linear system incorporating this jump in pressure. We refer the reader to [20] for explicit details and a demonstration of the physical accuracy of this approach in the context of deflagration waves.

The temperature profile has great effect on how we visually perceive flames. Thus, we need a way to track individual fluid elements as they cross over the blue core and rise upward due to buoyancy. That is, we need to know how much time has elapsed since a fluid element has passed through the blue core so that we can assign an appropriate temperature to it. This is easily accomplished using a reaction coordinate variable Y governed by the equation

$$Y_t = -\left(\vec{V} \cdot \nabla\right) Y - k, \tag{1.30}$$

where k is a positive constant which we take to be 1 (larger or smaller values can be used to get a good numerical variation of Y in the flame). Ignoring the convection term, $Y_t = -1$ can be solved exactly to obtain $Y(t) = -t + Y(0)$. If we set $Y(0) = 1$ in the region of space occupied by the gaseous fuel and solve equation 1.30 for Y, then the local value of $1 - Y$ is equal to the total time elapsed since a fluid element crossed over the blue reaction core. We can now use the values of Y to assign temperature values to the flow. The animator can sculpt both the temperature rise and the temperature falloff for a flame based on Y. However, for the temperature falloff, there is a physically correct, viable (i.e. computationally cheap) alternative. For the values of Y in the temperature falloff region, we simply solve

$$T_t = -\left(\vec{V} \cdot \nabla\right) T - c_T \left(\frac{T - T_{air}}{T_{max} - T_{air}}\right)^4 \tag{1.31}$$

which is derived from conservation of energy. This equation is solved by first using the semi-Lagrangian stable fluids method to solve for the convection term. Then the fourth power term is analytically integrated to cool down the flame at a rate governed by the cooling constant c_T. Similar to the temperature, the animator can sculpt a density curve for smoke and soot formation. The density should start low and increase as the reaction proceeds. In the temperature falloff region, the animator can switch from the density curve to a physically correct equation

$$\rho_t = -\left(\vec{V} \cdot \nabla\right) \rho \tag{1.32}$$

that can (once again) be solved using the semi-Lagrangian stable fluids method.

5 References

[1] Adalsteinsson, D. and Sethian, J., *The Fast Construction of Extension Velocities in Level Set Methods*, J. Comput. Phys. 148, 2-22 (1999).

[2] Bertalmio, M., Bertozzi, A. and Sapiro, G., *Navier-Stokes, Fluid Dynamics, and Image and Video Inpainting*, Proc. of the Int. Conf. on Comput. Vision and Pattern Recognition, IEEE, Kauai, HI, v. I, pp. 355-362 (2001).

[3] Chen, S., Johnson, D. and Raad, P., *Velocity Boundary Conditions for the Simulation of Free Surface Fluid Flow*, J. Comput. Phys. 116, 262-276 (1995).

[4] Chorin, A., *A Numerical Method for Solving Incompressible Viscous Flow Problems*, J. Comput. Phys. 2, 12-26 (1967).

[5] Enright, D., Fedkiw, R., Ferziger, J. and Mitchell, I. *A Hybrid Particle Level Set Method for Improved Interface Capturing*, J. Comput. Phys. (in review).

[6] Enright, D., Marschner, S. and Fedkiw, R., *Animation and Rendering of Complex Water Surfaces*, Siggraph 2002 Annual Conference (in press).

[7] Fedkiw, R., Aslam, T., Merriman, B. and Osher, S., *A Non-Oscillatory Eulerian Approach to Interfaces in Multimaterial Flows (The Ghost Fluid Method)*, J. Comput. Phys. 152, 457-492 (1999).

[8] Fedkiw, R., Stam, J. and Jensen, H., *Visual Simulation of Smoke*, Siggraph 2001 Annual Conference, 23-30 (2001).

[9] Foster, N. and Fedkiw, R., *Practical Animation of Liquids*, Siggraph 2001 Annual Conference, 15-22 (2001).

[10] Foster, N. and Metaxas, D., *Modeling the Motion of a Hot, Turbulent Gas*, Siggraph 1997 Annual Conference, 181-188 (1997).

[11] Foster, N. and Metaxas, D., *Realistic Animation of Liquids*, Graphical Models and Image Processing 58, 471-483 (1996).

[12] Golub, G. and Van Loan, C., *Matrix Computations*, The Johns Hopkins University Press, Baltimore, 1989.

[13] Harlow, F. and Welch, J., *Numerical Calculation of Time-Dependent Viscous Incompressible Flow of Fluid with a Free Surface*, The Physics of Fluids 8, 2182-2189 (1965).

[14] Hiltzik, M. and Pham, A., *Synthetic Actors Guild*, Los Angeles Times, May 8, 2001, natl. ed. : A1+, 2001.

[15] Jiang, G.-S. and Peng, D., *Weighted ENO Schemes for Hamilton Jacobi Equations*, SIAM J. Sci.Comput. 21, 2126-2143 (2000).

[16] Kass, M., Witkin, A. and Terzopoulos, D., *Snakes: Active Contour Models*, Int. J. of Comp. Vision 1, 321-331 (1988).

[17] Landau, L. and Lifshitz, E., *Fluid Mechanics*, Butterworth Heinemann, 1959.

[18] Markstein, G., *Nonsteady Flame Propagation*, Pergamon Press, Oxford (1964).

[19] Nguyen, D., Fedkiw, R. and Jensen, H., *Physically Based Modeling and Animation of Fire*, Siggraph 2002 Annual Conference (in press).

[20] Nguyen, D., Fedkiw, R. and Kang, M., *A Boundary Condition Capturing Method for Incompressible Flame Discontinuities*, J. Comput. Phys. 172, 71-98 (2001).

[21] Osher, S. and Fedkiw, R., *Level Set Methods and Dynamic Implicit Surfaces*, Springer Verlag, NY 2002.

[22] Stam, J., *Stable Fluids*, Siggraph 19999 Annunal Conference, 121-128 (1999).

[23] Stam, J. and Fiume, E., *Turbulent Wind Fields for Gaseous Phenomena*, Siggraph 1993 Annual Conference, 369-376 (1993).

[24] Staniforth, A. and Cote, J., *Semi-Lagrangian Integration Schemes for Atmospheric Models - A Review*, Monthly Weather Review 119, 2206-2223 (1991).

[25] Steinhoff, J. and Underhill, D., *Modification of the Euler Equations for "Vorticity Confinement": Application to the Computation of Interacting Vortex Rings*, Physics of Fluids 6, 2738-2744 (1994).

[26] Sussman, M., Smereka, P. and Osher, S., *A Level Set Approach for Computing Solutions to Incompressible Two-Phase Flow*, J. Comput. Phys. 114, 146-159 (1994).

[27] Turns, S., *An Introduction to Combustion*, McGraw-Hill Inc., 1996.

[28] Zhao, H.-K., Osher, S. and Fedkiw, R., *Fast Surface Reconstruction Using the Level Set Method*, 1st IEEE Workshop on Variational and Level Set Methods, 8th ICCV, Vancouver, 194-202, 2001.

[29] Zhao, H.-K., Osher, S., Merriman B. and Kang, M., *Implicit and Nonparametric Shape Reconstruction from Unorganized Data Using a Variational Level Set Method*, Comput. Vision and Image Understanding 80, 295-314 (2000).