CHAPTER

Fluids: Liquids and Gases 8

Among the most difficult graphical objects to model and animate are those that are not defined by a static, rigid, topologically simple structure. Many of these complex forms are found in nature. They present especially difficult challenges for those intent on controlling their motion. In some cases, special models of a specific phenomenon will suffice. We begin the chapter presenting special models for water, clouds, and fire that approximate their behavior under certain conditions. These models identify salient characteristics of the phenomena and attempt to model those characteristics explicitly. Such approaches are useful for a specific appearance or motion, but for a more robust model, a more rigorous scientific approach is needed. As far as a computational science is concerned, all of the phenomena mentioned earlier fall under the classification of *fluids* and computing their motion is called *computational fluid dynamics* (CFD). The basics of CFD are given at the end of this chapter.

Many of the time-varying models described in this chapter represent work that is state of the art. It is not the objective here to cover all aspects of recent research. The basic models are covered, with only brief reference to the more advanced algorithms.

8.1 Specific fluid models

Fire, smoke, and clouds are gaseous phenomena that have no well-defined surface to model. They are inherently volumetric models, although surface-based techniques have been applied with limited success. For example, water when relatively still has a well-defined surface; however, water changes its shape as it moves. In the case of ocean waves, features on the water's surface move, but the water itself does not travel. The simple surface topology can become arbitrarily complex when the water becomes turbulent. Splashing, foaming, and breaking waves are complex processes best modeled by particle systems and volumetric techniques, but these techniques are inefficient in nonturbulent situations. In addition, water can travel from one place to another, form streams, split into separate pools, and collect again. In modeling these phenomena for purposes of computer graphics, programmers often make simplifying assumptions in order to limit the computational complexity and model only those features of the physical processes that are visually important in the specific situation of interest.

8.1.1 Models of water

Water presents a particular challenge for computer animation because its appearance and motion take various forms [5] [7] [17] [20]. Water can be modeled as a still, rigid-looking surface to which ripples can be added as display attributes by perturbing the surface normal using bump mapping [1].

Alternatively, water can be modeled as a smoothly rolling height field in which time-varying ripples are incorporated into the geometry of the surface [12]. In ocean waves, it is assumed that there is no transport of water even though the waves travel along the surface in forms that vary from sinusoidal to cycloidal [6] [15].[1] Breaking, foaming, and splashing of the waves can be added on top of the base model in a postprocessing step [6] [15]. The transport of water from one location to another adds more computational complexity to the modeling problem [10].

Still waters and small-amplitude waves

The simplest way to model water is merely to assign the color blue to anything below a given height. If the y-axis is "up," then color any pixel blue (with, for example, an illumination model that uses a consistent normal) in which the world space coordinate of the corresponding visible surface has a y-value less than some given constant. This creates the illusion of still water at a consistent "sea level." It is sufficient for placid lakes and puddles of standing water. Equivalently, a flat blue plane perpendicular to the y-axis and at the height of the water can be used to represent the water's surface. These models, of course, do not produce any animation of the water.

Normal vector perturbation (the approach employed in bump mapping) can be used to simulate the appearance of small amplitude waves on an otherwise still body of water. To perturb the normal, one or more simple sinusoidal functions are used to modify the direction of the surface's normal vector. The functions are parameterized in terms of a single variable, usually relating to distance from a source point. It is not necessarily the case that the wave starts with zero amplitude at the source. When standing waves in a large body of water are modeled, each function usually has a constant amplitude. The wave crests can be linear, in which case all the waves generated by a particular function travel in a uniform direction, or the wave crests can radiate from a single user-specified or randomly generated source point. Linear wave crests tend to form self-replicating patterns when viewed from a distance. For a different effect, radially symmetrical functions that help to break up these global patterns can be used. Radial functions also simulate the effect of a thrown pebble or raindrop hitting the water (Figure 8.1). The time-varying height for a point at which the wave begins at time zero is a function of the amplitude and wavelength of the wave (Figure 8.2). Combining the two, Figure 8.3 shows the height of a point at some distance d from the start of the wave. This is a two-dimensional function relative to a point at which the function is zero at time zero. This function can be rotated and translated so that it is positioned and oriented appropriately in world space. Once the height function for a given point is defined, the normal to the point at any instance in time can be determined by computing the tangent vector and forming the vector perpendicular to it. These vectors should then be oriented in world space, so the plane they define contains the direction that the wave is traveling.

Superimposing multiple sinusoidal functions of different amplitude and with various source points (in the radial case) or directions (in the linear case) can generate interesting patterns of overlapping ripples. Typically, the higher the frequency of the wave component, the lower the amplitude. Notice that these do not change the geometry of the surface used to represent the water (e.g., a flat blue plane) but are used only to change the shading properties. Also notice that it must be a time-varying function that propagates the wave along the surface.

The same approach used to calculate wave normals can be used to modify the height of the surface (e.g., [11]). A mesh of points can be used to model the surface of the water and the heights of the

[1]A *cycloid* is the curve traced out by a point on the perimeter of a rolling disk.

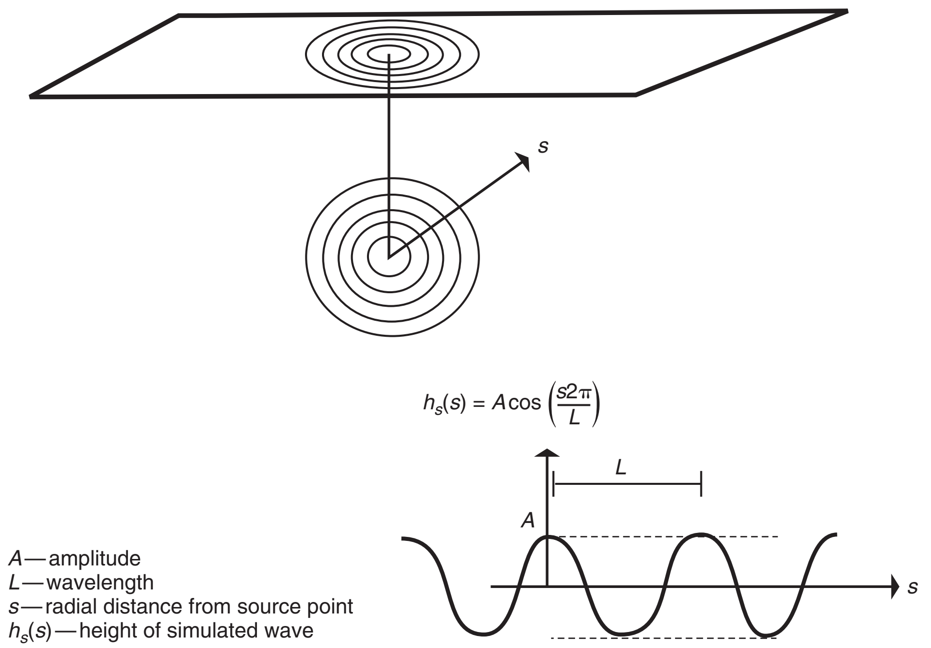

s
$h_s(s) = A\cos\left(\frac{s2\pi}{L}\right)$
L
A
s
A—amplitude
L—wavelength
s—radial distance from source point
$h_s(s)$—height of simulated wave

FIGURE 8.1

Radially symmetric standing wave.

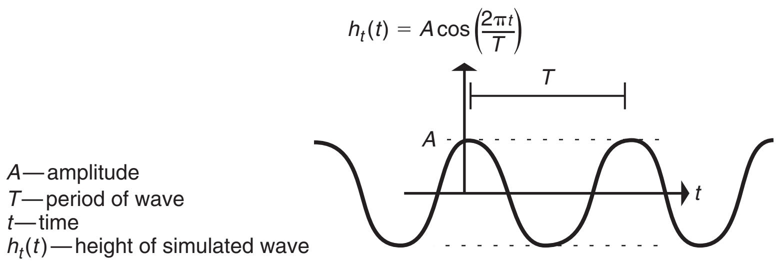

$h_t(t) = A\cos\left(\frac{2\pi t}{T}\right)$
T
A
t
A—amplitude
T—period of wave
t—time
$h_t(t)$—height of simulated wave

FIGURE 8.2

Time-varying height of a stationary point.

individual points can be controlled by the overlapping sinusoidal functions. Either a faceted surface with smooth shading can be used or the points can be the control points of a higher order surface such as a B-spline surface. The points must be sufficiently dense to sample the height function accurately enough for rendering. Calculating the normals without changing the actual surface creates the illusion of waves on the surface of the water. However, whenever the water meets a protruding surface, like a rock, the lack of surface displacement will be evident. See Figures 8.4 and 8.5 for a comparison between a water surface modeled by normal vector perturbation and a water surface modeled by a height field. An option to reduce the density of mesh points required is to use only the low-frequency, high-amplitude functions to control the height of the surface points and to include the high-frequency, low-amplitude functions to calculate the normals (Figure 8.6).

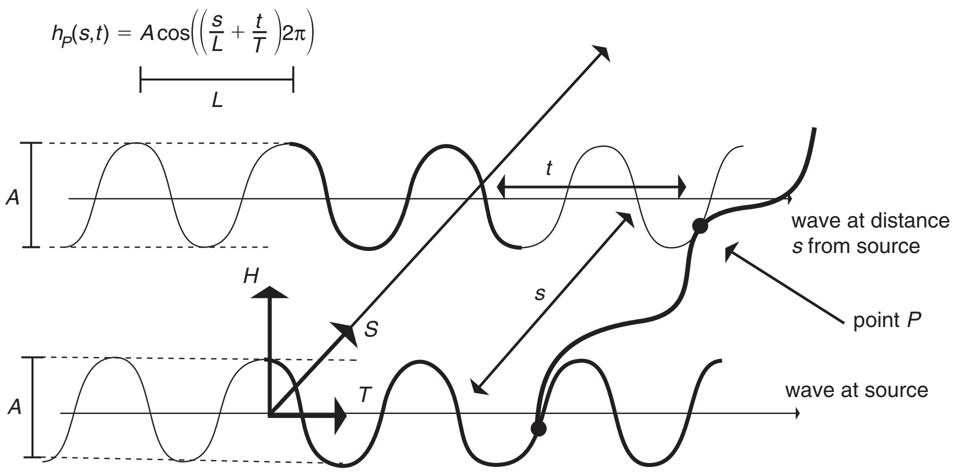

$h_P(s,t) = A\cos\left(\left(\frac{s}{L}+\frac{t}{T}\right)2\pi\right)$
L
A
t
wave at distance s from source
H
s
point P
S
T
A
wave at source

FIGURE 8.3

Time-varying function at point *P*.

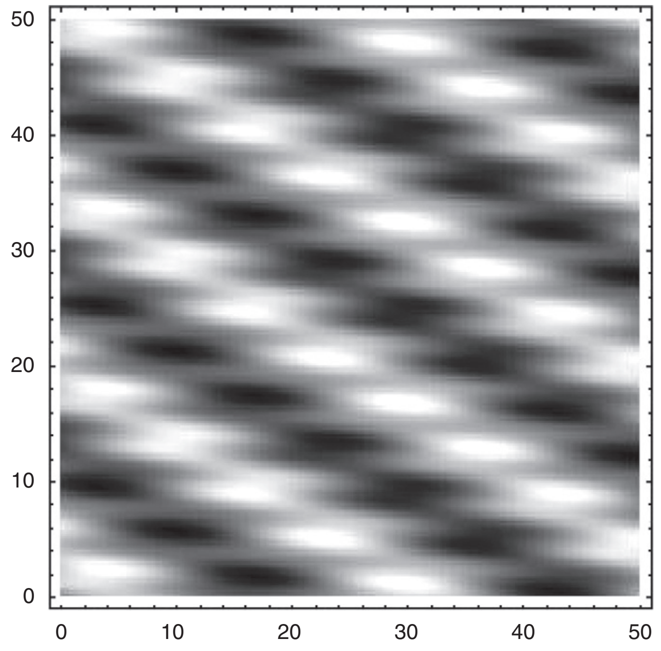


FIGURE 8.4

Superimposed linear waves of various amplitudes and frequencies.

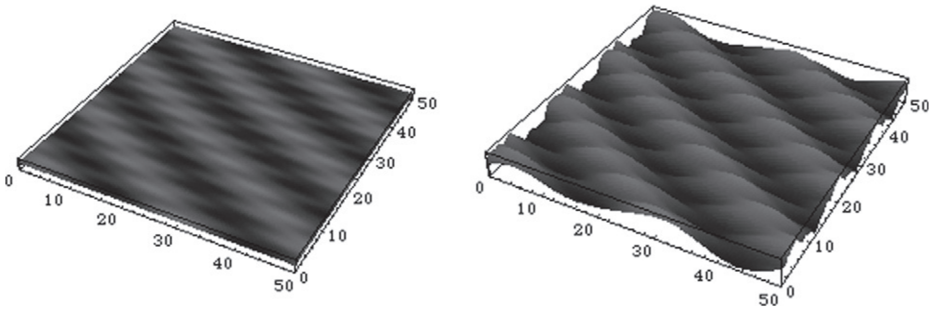
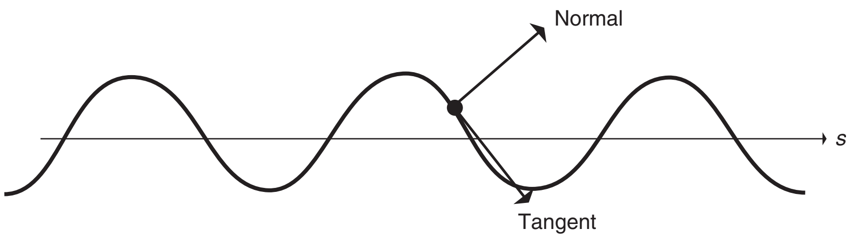


FIGURE 8.5

Normal vector displacement versus height displacement.



$$h_P(s,t) = A\cos\left(\left(\frac{s}{L}+\frac{t}{T}\right)2\pi\right)$$

$$\frac{d}{ds}h_p(s,t) = -\left(A\frac{2\pi}{L}\sin\left(\left(\frac{s}{L}+\frac{t}{T}\right)2\pi\right)\right)$$

$$\text{Tangent} = \left(1, \frac{d}{ds}h_p(s,t), 0\right)$$

$$\text{Normal} = \left(-\left(\frac{d}{ds}h_p(s,t)\right) 1, 0\right)$$

FIGURE 8.6

Normal vector for two-dimensional wave function.

The anatomy of waves

A more sophisticated model must be used to model waves with greater realism, one that incorporates more of the physical effects that produce their appearance and behavior. Waves come in various frequencies, from tidal waves to capillary waves, which are created by wind passing over the surface of the water. The waves collectively called wind waves are those of most interest for visual effects. The sinusoidal form of a simple wave has already been described and is reviewed here in a more appropriate form for the equations that follow. In Equation 8.1, the function $f(s, t)$ describes the amplitude of the wave in which s is the distance from the source point, t is a point in time, A is the maximum amplitude, C is the propagation speed, and L is the wavelength. The period of the wave, T, is the time it takes for one complete wave to pass a given point. The wavelength, period, and speed are related by the equation $C = L/T$.

$$f(s,t) = \cos^{-1}\left(\frac{2\pi(s - Ct)}{L}\right) \tag{8.1}$$

The motion of the wave is different from the motion of the water. The wave travels linearly across the surface of the water, while a particle of water moves in nearly a circular orbit (Figure 8.7). While riding the crest of the wave, the particle will move in the direction of the wave. As the wave passes and the particle drops into the trough between waves, it will travel in the reverse direction. The steepness, S, of the wave is represented by the term H/L where H is defined as twice the amplitude.

Waves with a small steepness value have basically a sinusoidal shape. As the steepness value increases, the shape of the wave gradually changes into a sharply crested peak with flatter troughs. Mathematically, the shape approaches that of a cycloid.

In an idealized wave, there is no net transport of water. The particle of water completes one orbit in the time it takes for one complete cycle of the wave to pass. The average orbital speed of a particle of water is given by the circumference of the orbit, pH, divided by the time it takes to complete the orbit, T (Eq. 8.2).

$$Q_{\text{ave}} = \frac{\pi H}{T} = \frac{\pi HC}{L} = \pi SC \tag{8.2}$$

If the orbital speed, Q, of the water at the crest exceeds the speed of the wave, C, then the water will spill over the wave, resulting in a breaking wave. Because the average speed, Q, increases as the steepness, S, of the wave increases, this limits the steepness of a nonbreaking wave. The observed steepness of ocean waves, as reported by Peachey [15] is between 0.5 and 1.0.

A common simplification of the full CFD simulation of ocean waves is called the Airy model, and it relates the depth of the water, d, the propagation speed, C, and the wavelength of the wave, L (Eq. 8.3).

$$\begin{aligned} C &= \sqrt{\frac{g}{\kappa}\tanh(\kappa d)} = \sqrt{\frac{gL}{2\pi} \times \tanh\left(\frac{2\pi d}{L}\right)} \\ L &= CT \end{aligned} \tag{8.3}$$

In Equation 8.3, g is the acceleration of a body due to gravity at sea level, 9.81 m/sec^2, and $k = 2\pi/L$ is the spatial equivalent of wave frequency. As the depth of the water increases, the function $\tanh(kd)$ tends toward one, so C approaches $gL/2\pi$. As the depth decreases and approaches zero, $\tanh(kd)$ approaches kd, so C approaches $\sqrt{gd}$. Peachey suggests using *deep* to mean $d > L/4$ and *shallow* to mean $d < L/20$.

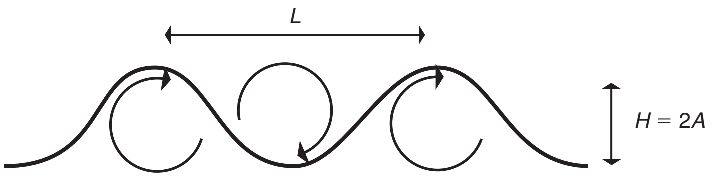


FIGURE 8.7

Circular paths of particles of water subjected to waves.

As a wave approaches the shoreline at an angle, the part of the wave that approaches first will slow down as it encounters a shallower area. The wave will progressively slow down along its length as more of it encounters the shallow area. This will tend to straighten out the wave and is called *wave refraction*.

Interestingly, even though speed (C) and wavelength (L) of the wave are reduced as the wave enters shallow water, the period, T, of the wave remains the same and the amplitude, A (and, equivalently, H), remains the same or increases. As a result, the orbital speed, Q (Eq. 8.2), of the water remains the same. Because orbital speed remains the same as the speed of the wave decreases, waves tend to break as they approach the shoreline because the speed of the water exceeds the speed of the wave. The breaking of a wave means that water particles break off from the wave surface as they are "thrown forward" beyond the front of the wave.

Modeling ocean waves

The description of modeling ocean waves presented here follows Peachey [15]. The ocean surface is represented as a height field, $y=f(x, z, t)$, where (x, z) defines the two-dimensional ground plane, t is time, and y is the height. The wave function f is a sum of various waveforms at different amplitudes (Eq. 8.4).

$$f(x,z,t) = \sum_{i=1}^{n} A_i W_i(x,z,t) \tag{8.4}$$

The wave function, W_i, is formed as the composition of two functions: a wave profile, w_i, and a phase function, $\theta_i(x, z, t)$, according to Equation 8.5. This allows the description of the wave profile and phase function to be addressed separately.

$$W_i(x,z,t) = \omega_i(\text{fraction}[\theta_i(x,z,t)]) \tag{8.5}$$

Each waveform is described by its period, amplitude, source point, and direction. It is convenient to define each waveform, actually each phase function, as a linear rather than radial wave and to orient it so the wave is perpendicular to the x-axis and originates at the source point. The phase function is then a function only of the x-coordinate and can then be rotated and translated into position in world space.

Equation 8.6 gives the time dependence of the phase function. Thus, if the phase function is known for all points x (assuming the alignment of the waveform along the x-axis), then the phase function can be easily computed at any time at any position. If the depth of water is constant, the Airy model states that the wavelength and speed are also constant. In this case, the aligned phase function is given in Equation 8.7.

$$\theta_i(x,y,t) = \theta_i(x,y,t_0) - \frac{t-t_0}{T_i} \tag{8.6}$$

$$\theta_i(x,z,t) = \frac{x_i}{L_i} \tag{8.7}$$

However, if the depth of the water is variable, then L_i is a function of depth and u_i is the integral of the depth-dependent phase-change function from the origin to the point of interest (Eq. 8.8). Numerical integration can be used to produce phase values at predetermined grid points. These grid points can be used to interpolate values within grid cells. Peachey [15] successfully uses bilinear interpolation to accomplish this.

$$\theta_i(x,z,t) = \int_0^x \theta_i'(u,z,t)du \tag{8.8}$$

The wave profile function, *wi*, is a single-value periodic function of the fraction of the phase function (Eq. 8.5) so that $w_i(u)$ is defined for $0.0 \leq u \leq 1.0$. The values of the wave profile function range over the interval $[-1, 1]$. The wave profile function is designed so that its value is one at both ends of the interval (Eq. 8.9).

$$\omega_i(0,0) = \omega_i(1.0) = 1.0 \tag{8.9}$$

Linear interpolation can be used to model the changing profile of the wave according to steepness. Steepness (H/L) can be used to blend between a sinusoidal function (Eq. 8.1) and a cycloid-like function (Eq. 8.10) designed to resemble a sharp-crested wave profile. In addition, wave asymmetry is introduced as a function of the depth of the water to simulate effects observed in waves as they approach a coastline. The asymmetry interpolant, k, is defined as the ratio between the water depth, d, and deep-water wavelength, L_i (see Eq. 8.11). When k is large, the wave profile is handled with no further modification. When k is small, u is raised to a power in order to shift its value toward the low end of the range between zero and one. This has the effect of stretching out the back of the wave and steepening the front of the wave as it approaches the shore.

$$\omega_i(u) = 8|u - 1/2|^2 - 1 \tag{8.10}$$

$$\begin{aligned} L_i^{\text{deep}} &= \frac{gT_i^2}{2\pi} \\ k &= \frac{d}{L_i^{\text{deep}}} \end{aligned} \tag{8.11}$$

As the wave enters very shallow water, the amplitudes of the various wave components are reduced so the overall amplitude of the wave is kept from exceeding the depth of the water.

Spray and foam resulting from breaking waves and waves hitting obstacles can be simulated using a stochastic but controlled (e.g., Gaussian distribution) particle system. When the speed of the water, Q_{average}, exceeds the speed of the wave, C, then water spray leaves the surface of the wave and is thrown forward. Equation 8.2 indicates that this condition happens when $\pi S > 1.0$ or, equivalently, $S > 1.0/\pi$. Breaking waves are observed with steepness values less than this (around 0.1), which indicates that the water probably does not travel at a uniform orbital speed. Instead, the speed of the water at the top of the orbit is faster than at other points in the orbit. Thus, a user-specified spray-threshold steepness value can be used to trigger the particle system. The number of particles generated is based on the difference between the calculated wave steepness and the spray-threshold steepness.

For a wave hitting an obstacle, a particle system can be used to generate spray in the direction of reflection based on the incoming direction of the wave and the normal of the obstacle surface. A small number of particles are generated just before the moment of impact, are increased to a maximum number at the point of impact, and are then decreased as the wave passes the obstacle. As always, stochastic perturbation should be used to control both speed and direction.

For a more complete treatment of modeling height-field displacement-mapped surface for ocean waves using a fast Fourier transform description, including modeling and rendering underwater environmental effects, the interested reader is directed to the course notes by J. Tessendorf [19].

Finding its way downhill

One of the assumptions used to model ocean waves is that there is no transport of water. However, in many situations, such as a stream of water running downhill, it is useful to model how water travels from one location to another. In situations in which the water can be considered a height field and the motion is assumed to be uniform through a vertical column of water, the vertical component of the velocity can be ignored. In such cases, differential equations can be used to simulate a wide range of convincing motion [10]. The Navier-Stokes equations (which describe flow through a volume) can be simplified to model the flow.

To develop the equations in two dimensions, the user parameterizes functions that are in terms of distance x. Let $z=h(x)$ be the height of the water and $z=b(x)$ be the height of the ground at location x. The height of the water is $d(x)=h(x)-b(x)$. If one assumes that motion is uniform through a vertical column of water and that $v(x)$ is the velocity of a vertical column of water, then the shallow-water equations are as shown in Equations 8.12 and 8.13, where g is the gravitational acceleration (see Figure 8.8). Equation 8.12 considers the change in velocity of the water and relates its acceleration, the difference in adjacent velocities, and the acceleration due to gravity when adjacent columns of water are at different heights. Equation 8.13 considers the transport of water by relating the temporal change in the height of the vertical column of water with the spatial change in the amount of water moving.

$$\frac{\partial v}{\partial t} + v\frac{\partial v}{\partial x} + g\frac{\partial h}{\partial x} = 0 \tag{8.12}$$

$$\frac{\partial d}{\partial t} + \frac{\partial}{\partial x}(vd) = 0 \tag{8.13}$$

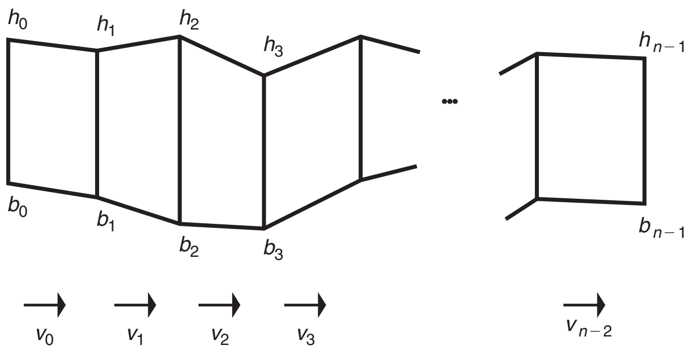


FIGURE 8.8

Discrete two-dimensional representation of height field with water surface h, ground b, and horizontal water velocity v.

These equations can be further simplified if the assumptions of small fluid velocity and slowly varying depth are used. The former assumption eliminates the second term of Equation 8.12, while the latter assumption implies that the term d can be removed from inside the derivative in Equation 8.13. These simplifications result in Equations 8.14 and 8.15.

$$\frac{\partial v}{\partial t} + g\frac{\partial v}{\partial x} = 0 \tag{8.14}$$

$$\frac{\partial d}{\partial t} + d\frac{\partial v}{\partial x} = 0 \tag{8.15}$$

Differentiating Equation 8.14 with respect to x and Equation 8.15 with respect to t and substituting for the cross-derivatives results in Equation 8.16. This is the one-dimensional wave equation with a wave velocity $\sqrt{gd}$. As Kass and Miller [10] note, this degree of simplification is probably not accurate enough for engineering applications.

$$\frac{\partial^2 h}{\partial t^2} + gd\frac{\partial^2 h}{\partial x^2} \tag{8.16}$$

This partial differential equation is solved using finite differences. The discretization, as used by Kass and Miller [10], is set up as in Figure 8.8, with samples of v positioned halfway between the samples of h. The authors report a stable discretization, resulting in Equations 8.17 and 8.18. Putting these two equations together results in Equation 8.19, which is the discrete version of Equation 8.16.

$$\frac{\partial h_i}{\partial t} = \left(\frac{d_{i-1} + d_i}{2\Delta x}\right)\Big|_{v_{i-1}} - \left(\frac{d_i + d_{i+1}}{2\Delta x}\right)v_i \tag{8.17}$$

$$\frac{\partial v_i}{\partial t} = \frac{(-g)(h_{i+1} - h_i)}{\Delta x} \tag{8.18}$$

$$\frac{\partial^2 h_i}{\partial t^2} = -g\left(\frac{(d_{i-1} + d_i)}{2(\Delta x)^2}\right)(h_i - h_{i-1}) + g\left(\frac{(d_i + d_{i+1})}{2(\Delta x)^2}\right)(h_{i+1} - h_i) \tag{8.19}$$

Equation 8.19 states the relationship of the height of the water surface to the height's acceleration in time. This could be solved by using values of h_i to compute the left-hand side and then using this value to update the next time step. As Kass and Miller [10] report, however, this approach diverges quickly because of the sample spacing.

A first-order implicit numerical integration technique is used to provide a stable solution to Equation 8.19. Numerical integration uses current sample values to approximate derivatives. Explicit methods use approximated derivatives to update the current samples to their new values. Implicit integration techniques find the value whose derivative matches the discrete approximation of the current samples. Implicit techniques typically require more computation per step, but, because they are less likely to diverge significantly from the correct values, larger time steps can be taken, thus producing an overall savings.

Kass and Miller [10] find that a first-order implicit method (Eqs. 8.20 and 8.21) is sufficient for this application. Using these equations to solve for $h(n)$ and substituting Equation 8.19 for the second derivative ($\ddot{h}(n)$) results in Equation 8.22.

$$\dot{h}(n) = \frac{h(n) - h(n-1)}{\Delta t} \tag{8.20}$$

$$\ddot{h}(n) = \frac{\dot{h}(n) - \dot{h}(n-1)}{\Delta t} \tag{8.21}$$

$$\begin{aligned} h_i(n) = {} & 2h_i(n-1) - h_i(n-2) \\ & -g(\Delta t)^2 \frac{d_{i-1} + d_i}{2 \cdot \Delta x^2}(h_i(n) - h_{i-1}(n)) \\ & +g(\Delta t)^2 \frac{d_{i-1} + d_i}{2 \cdot \Delta x^2}(h_{i+1}(n) - h_i(n)) \end{aligned} \tag{8.22}$$

Assuming d is constant during the iteration, the next value of h can be calculated from previous values with the symmetric tridiagonal linear system represented by Equations 8.23–8.25.

$$Ah_i(n) = 2h_i(n-1) + h_i(n-2) \tag{8.23}$$

$$A = \begin{bmatrix} e_0 & f_0 & 0 & 0 & 0 & 0 & 0 \\ f_0 & e_0 & f_i & 0 & 0 & 0 & 0 \\ 0 & f_i & e_2 & \ldots & 0 & 0 & 0 \\ 0 & 0 & \ldots & \ldots & \ldots & 0 & 0 \\ 0 & 0 & 0 & \ldots & e_{n-3} & f_{n-3} & 0 \\ 0 & 0 & 0 & 0 & f_{n-3} & e_{n-2} & f_{n-2} \\ 0 & 0 & 0 & 0 & 0 & f_{n-2} & e_{n-1} \end{bmatrix} \tag{8.24}$$

$$e_0 = 1 + g(\Delta t)^2 \left(\frac{d_0 + d_1}{2(\Delta x)^2} \right)$$

$$e_i = 1 + g(\Delta t)^2 \left(\frac{d_{i-1} + 2d_1 + d_{i+1}}{2(\Delta x)^2} \right) (0 < i < n-1)$$

$$e_{n-1} = 1 + g(\Delta t)^2 \left(\frac{d_{n-2} + d_{n-1}}{2(\Delta x)^2} \right)$$

$$f_i = -\left(g(\Delta t)^2 \left(\frac{d_i + d_{i+1}}{2(\Delta x)^2} \right) \right) \tag{8.25}$$

To simulate a viscous fluid, Equation 8.23 can be modified to incorporate a parameter that controls the viscosity, thus producing Equation 8.26. The parameter τ ranges between 0 and 1. When $\tau = 0$, Equation 8.26 reduces to Equation 8.3.

$$Ah_i(n) = h_i(n-1) + (1-\tau)(h_i(n-1) - h_i(n-2)) \tag{8.26}$$

```
Specify h(t=0), h(t=1), and b
for j=2 . . . n
    to simulate sinks and sources, modify appropriate h values
    calculate d from h(j-1) and b; if hi<bi, then di=0
    use Equation 8.10 (Equation 8.13) to calculate h(j) from h(j-1) and
      h(j-2)
    adjust the values of h to conserve volume (as discussed above)
    if hi<bi, set hi(j) and hi−1(j) to bi−ε
```

FIGURE 8.9

Two-dimensional algorithm for water transport.

Volume preservation can be compromised when $h_i < b_i$. To compensate for this, search for the connected areas of water ($h_j < b_j$ for $j = i,\ i+1,\ \ldots,\ i+n$) at each iteration and compute the volume. If the volume changes from the last iteration. then distribute the difference among the elements of that connected region. The algorithm for the two-dimensional case is shown in Figure 8.9.

Extending the algorithm to the three-dimensional case considers a two-dimensional height field. The computations are done by decoupling the two dimensions of the field. Each iteration is decomposed into two subiterations, one in the *x*-direction and one in the *y*-direction.

Summary

Animating all of the aspects of water is a difficult task because of water's ability not only to change shape but also to change its topology over time. Great strides have been made in animating individual aspects of water such as standing waves, ocean waves, spray, and flowing water. But an efficient approach to an integrated model of water remains a challenge.

8.1.2 Modeling and animating clouds

Most any outdoor daytime scene includes some type of clouds as a visual element. Clouds can form a backdrop for activity in the foreground, in which case they may be slow moving or even still. Clouds can be used as an area of activity allowing planes to fly around or through them or, as fog, cars to drive through them. Clouds viewed from afar have certain characteristics that are much different from clouds that are viewed close up. From a distance, they exhibit high level opaque appearance features that transparently disappear upon inspection at a closer distance. Cloud formations can inspire, impart calm, be ominous and foreboding, or produce magnificent sunsets. Distant clouds can have a puffy opaque appearance, often resembling familiar shapes of, for example, various animals or other common objects. Other clouds have a layered or fibrous appearance. While some cloud formations viewed from a distance might appear as a stationary backdrop or slow moving rigid formations, storm clouds can

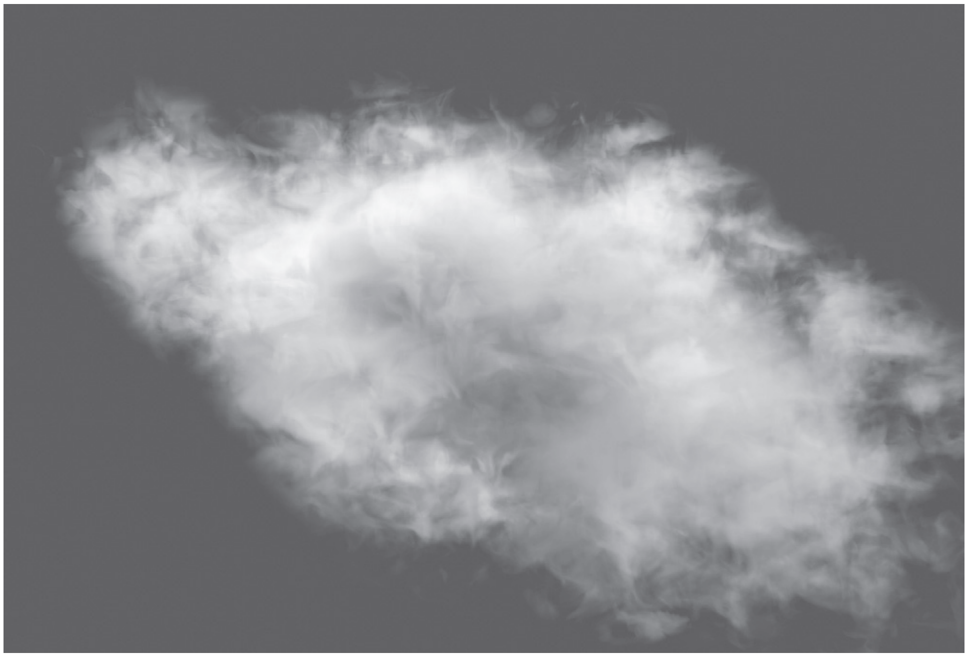


FIGURE 8.10

An example of cirrus and cirrostratus clouds at sunset. (Copyright 1998 David S. Ebert.)

form relatively quickly and can appear threatening in a manner of minutes. Complex atmospheric processes form clouds. These processes can be modeled and animated to show the formation of storms, hurricanes, and inclement weather (see Figures 8.10 and 8.11).

Clouds are naturally occurring atmospheric phenomena. They are composed of visible water droplets or ice crystals suspended in air, and are formed by moisture droplets adhering to dust particles. When moisture is added to the air and/or the temperature of the air decreases, clouds can form. In either case, moisture in the air condenses into droplets because the air cannot transparently hold that amount of water at that temperature. Clouds occur at a variety of altitudes. Some clouds can be found at 60,000 feet while others can be as low as ground level. Although typically one thinks of clouds as high in the sky, fog phenomena are just very low-level clouds. In addition, a single cloud formation can start near the surface and extend upward as much as 10,000 feet.

Anatomy of clouds and cloud formation

Clouds are categorized according to their altitude, their general shape, and whether they carry moisture (e.g., see [23]). The naming convention combines a standard set of terms that describe the altitude, shape, and moisture of the cloud.

FIGURE 8.11

An example of a cumulus cloud. (Copyright 1997 David S. Ebert.)

Altitude	
Cirrus/cirro	High clouds (by itself, it refers to high level fibrous clouds)
Altus/alto	Middle clouds
Otherwise	Low-level clouds
Shape	
Cumulus/cumulo	Puffy (Latin for “stack”)
Stratus/strato	Layer (Latin for “sheet")
Moisture	
Nimubs/nimbo	Water bearing

Certain types of clouds can typically be found at certain elevations. For example, high-level clouds are cirrus, cirrostratus, and cirrocumulus; mid-level clouds are altocumulus, altostratus, and nimbostratus; low-level clouds are cumulonimbus, stratocumulus, stratus, and cumulus. Clouds typified by vertical development are fair weather cumulus and cumulonimbus. Other cloud names less frequently encountered include contrails, billow clouds, mammatus, orographic, and pileus clouds.

Clouds are formed by low-level processes such as particles in the air, turbulence, wind shear, and temperature gradients combined with a significant moisture content. The basic moisture cycle consists of the following: transpiration $\rightarrow$ evaporation $\rightarrow$ condensation $\rightarrow$ precipitation $\rightarrow$ and back to transpiration.

The processes that form clouds consist of adding moisture to the air or cooling the air. In either case the moisture content becomes higher than what can be handled transparently by the air. As a result, water droplets form and cling to dust particles in the air. The cooling of an air mass is often due to an air mass rising in altitude; air tends to cool as it rises and loses pressure. The cause for this rise is usually due to something forcing the air mass upward: the traveling air mass encounters an incline like a mountain slope; the air mass travels up and over a colder, denser air mass (a warm front); and a cold, dense air mass forces its way underneath the warmer air mass and raises the warm air mass (a cold front). The air mass temperature can also be cooled by the cooling of the earth's surface, often creating fog or other low-level clouds. Adding moisture to the air mass is often accomplished by water evaporating at the earth's surface underneath the air mass.

Cloud models in CG

Although modeling clouds as a single, static background painting can be effective in some animations, limited cloud animation can be done cost effectively. This can just be a texture mapped quadrilateral. Simply moving (i.e., panning) the background provides some degree of animation. Most animation effects can be introduced by using an animated texture map, essentially a movie, of clouds for the background. Additionally, using multiple, semitransparent background images, representing clouds at varying heights, and moving them relative to one another, simulating parallax, can produce the impression of a deep three-dimensional space at minimal cost. In some situations, it is more appropriate to use a spherical shell instead of a planar surface, although care must be taken at the horizon to avoid unrealistic transitions between sky and ground. To display a fog effect, a simple linear distance-based attenuation of object shading can be used. However, the animation of fog swirls requires an approach similar to CFD discussed earlier in this chapter.

For more control and added visual realism, appropriately shaded geometric primitives can be used as cloud building blocks. Two-dimensional primitives can be used for distant clouds while three-dimensional primitives give a greater sense of distance and depth when viewed from various angles and are appropriate for more immersive environments. The primitives can be moved relative to one another and scaled up or down, providing easy control for an animator. While not physically based, this approach can produce effective visuals if the primitives are carefully designed.

The use of such building blocks was pioneered in 1984 by Geoffrey Gardner [8][9] and the general approach is still an attractive way to model clouds. Gardner's technique uses a sky plane that is parallel to the ground plane and some distance above it. Ellipsoids are used as the cloud building blocks and are positioned by the user relative to the sky plane. The ellipsoid surface is textured (as described in the following paragraphs) and then mapped onto a sky plane. Gardner also uses these textured ellipsoids to build full three-dimensional cloud models, which allows for a full three-dimensional effect as the viewing angle changes relative to the ellipsoids. Notice, however, that the texturing function is not defined on the interior of the ellipsoid shells, so the visuals associated with flying into a cloud are not supported. In addition to clouds, Gardner also used this approach, with suitable modification to the texture function, to model other environmental features such as trees and hills as in Figure 8.12.

The surface texturing function is based on the sum of two weighted sums of sine waves, defined to produce a pseudorandom cloud-like texture. The two sets of sine waves are orthogonal to each other. In each set, each frequency is twice the frequency of its predecessor and has an amplitude of one over radical two times its predecessor. In addition, the phase shift of each frequency is based on its position in the other dimension. A user-defined threshold is set so that

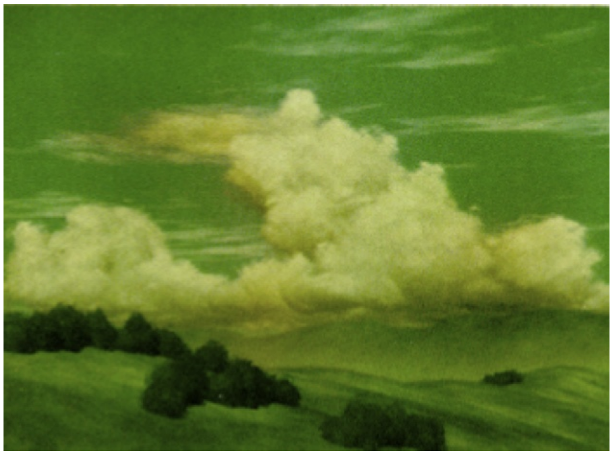


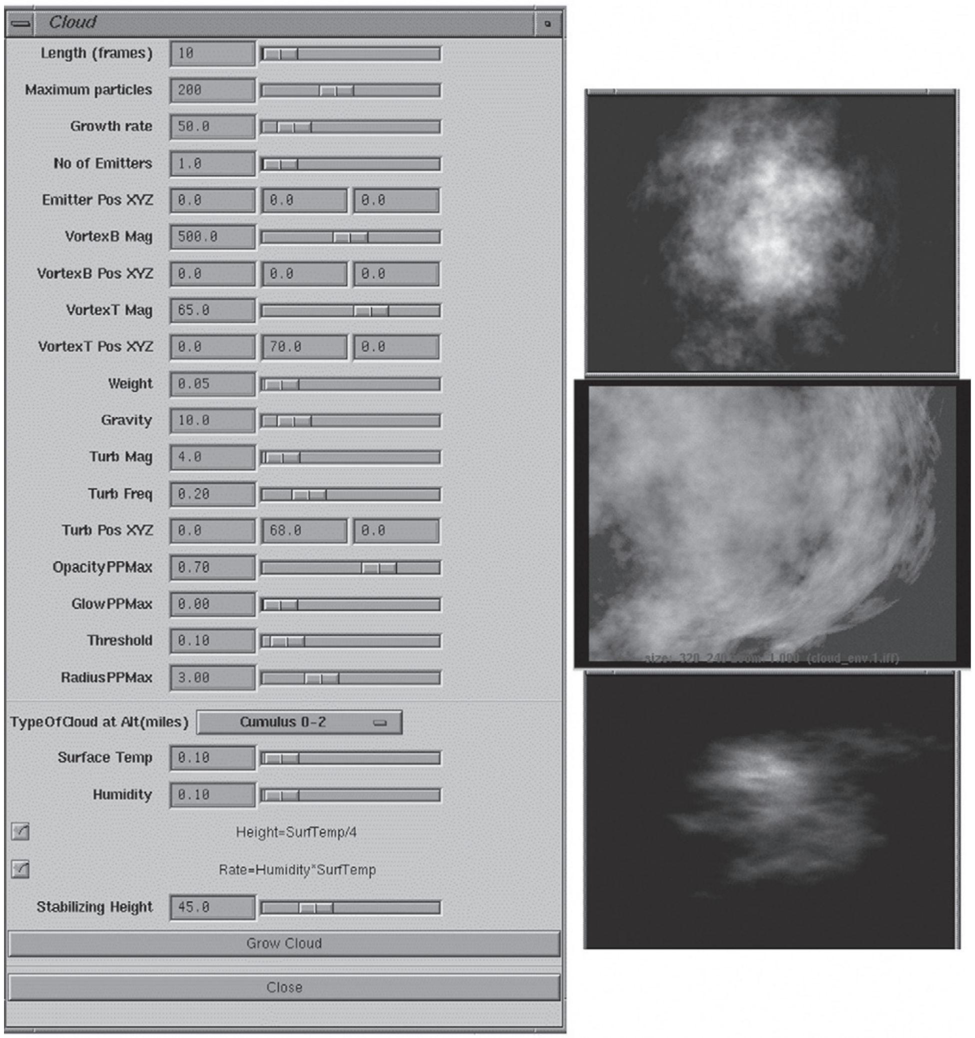
FIGURE 8.12

Landscape constructed using multiple partially transparent ellipsoids. (© 1985 ACM, Gardner, G., *Visual Simulation of Clouds*. Computer Graphics. Proceedings of SIGGRAPH 85, 19(3). San Francisco, CA., August 1985. With permission.)

any lesser texture value is considered totally transparent. In order to remove any discontinuities that might arise at the ellipsoid boundary, a monotonically decreasing attenuation function is defined to have a value of one at the middle of the ellipsoid and zero at the boundary. This is used to scale the texture function so that final function values scale down to zero as they approach the limit of the ellipsoid. This creates a cloud primitive that is generally in the shape of an ellipsoid, but whose texture feathers away close to the boundary, thus avoiding any noticeable discontinuities in the shading.

Surface-based textures, such as those used by Gardner, are sufficient for clouds viewed from a distance and represented on a sky plane or viewed from a distance, but to accommodate close viewing and even immersion in the clouds, such as that encountered in current flight simulators and games, a volumetric approach must be taken. Volumetric approaches define a density value for any point in space. Volumes are rendered by considering the entire line of sight through the volume. This makes realistic shading more difficult because it depends on the accumulated shading of each point along the line of sight inside the cloud. Each point along the light of sight has a certain density that controls its opacity/transparency. Each of these points along the line of sight reflects the light that gets to it from a light source. Direct illumination from the light source is light that filters through the cloud on a line from the light source directly to the point in question. Indirect illumination from the light source is light that comes from the light source by way of reflections off of nearby points in the cloud. This in turn means that the illumination of nearby points must be known in order to know how much light they reflect. Usually only one or two levels of such reflections are considered to make shading computations tractable. Mathematically, the shading is an integration along the line of sight and illumination is integration along lines from each light source. Indirect lighting is integration from all directions. For most applications, these integrals can be computed using appropriate discrete approximations or, in some cases, ignored altogether. In any case, the illumination is computationally expensive and not a topic here. The interested reader is directed to the book *Texturing and Modeling: A Procedural Approach*, edited by David Ebert [3].

The same building block approach to cloud formation can be taken with the volumetric approach, as with the ellipsoid shells of Gardner. See Figure 8.13 for an example of work by David Ebert. At the



Length (frames)	10		
Maximum particles	200		
Growth rate	50.0		
No of Emitters	1.0		
Emitter Pos XYZ	0.0	0.0	0.0
VortexB Mag	500.0		
VortexB Pos XYZ	0.0	0.0	0.0
VortexT Mag	65.0		
VortexT Pos XYZ	0.0	70.0	0.0
Weight	0.05		
Gravity	10.0		
Turb Mag	4.0		
Turb Freq	0.20		
Turb Pos XYZ	0.0	68.0	0.0
OpacityPPMax	0.70		
GlowPPMax	0.00		
Threshold	0.10		
RadiusPPMax	3.00		
TypeOfCloud at Alt(miles)	Cumulus 0-2		
Surface Temp	0.10		
Humidity	0.10		
✓	Height=SurfTemp/4		
✓	Rate=Humidity*SurfTemp		
Stabilizing Height	45.0		

A GUI used to control cloud formation B Example clouds

FIGURE 8.13

(A) An example of cloud dynamics GUI and (B) example images created in Maya^{TM}. (© 1999 Rchi Gartha.)

heart of Ebert's volumetric approach are implicitly defined density functions that provide high-level organization of the cloud, combined with turbulence functions that provide the fine detail.

The implicit function used here is really a sum of individual spherical implicit functions. Each of these spherical functions maps a point in space to a density based on the point's distance from the sphere's center and a density function. One property of the density function is that it is one at its center and monotonically decreases to zero at its outermost edge, which is called its *radius of influence*.

In addition, it has a slope of one at its radius of influence, allowing it to blend smoothly with other primitives. A commonly used density function is a sixth degree polynomial in terms of distance that the point is from the center, but a third degree polynomial in terms of distance-squared, thus avoiding the cost of the square root of the distance calculation (Eq. 8.27). The implicit function for the cloud is a weighted sum of these individual implicit functions (Eq. 8.28). See Chapter 12.1 for more discussion on implicit functions and surfaces. The use of these implicit density functions provides the macrostructure for the cloud formation, allowing an animator to easily form the cloud with simple high-level controls.

$$F(r) = -\frac{4}{9}\frac{r^6}{R^6} + \frac{17}{9}\frac{r^4}{R^4} - \frac{22}{9}\frac{r^2}{R^2} + 1 \tag{8.27}$$

$$D(p) = \sum_i w_i F(|p - q|) \tag{8.28}$$

To introduce some pseudorandomness into the density values, the animator can procedurally alter the location of points before evaluating the density function. This perturbation of the points should be continuous with limited high-frequency components in order to avoid aliasing artifacts in the result. The perturbation should also avoid unwanted patterns in the final values.

The microstructure of the clouds is provided by procedural noise and turbulence-based density function (see Appendix B.6). This is blended with the macrostructure density to produce the final values for the cloud density. Various cloud types can be modeled by modifying the primitive shapes, their relative positioning, the density parameters, and the turbulence parameters. See Figures 8.10 and 8.11 for examples.

The building-block approach allows the clouds to be animated either at the macro or the micro level. Cloud migration can be modeled by user-supplied scripts or simplified particle system physics can be used. Internal cloud dynamics can be modeled by animating the texture function parameters, in the case of Gardner-type clouds, or turbulence parameters in the case of Ebert.

8.1.3 Modeling and animating fire

Fire is a particularly difficult and computationally intensive process to model. It has all the complexities of smoke and clouds and the added complexity of very active internal processes that produce light and motion and create rapidly varying display attributes. Fire exhibits temporally and spatially transient features at a variety of granularies. The underlying process is that of combustion—a rapid chemical process that releases heat (i.e., is *exothermic*) and light accompanied by flame. A common example is that of a wood fire in which the hydrocarbon atoms of the wood fuel join with oxygen atoms to form water vapor, carbon monoxide, and carbon dioxide. As a consequence of the heated gases rising quickly, air is sucked into the combustion area, creating turbulence.

Recently, impressive advances have been made in the modeling of fire. At one extreme, the most realistic approaches require sophisticated techniques from CFD and are difficult to control (e.g., [18]). Work has also been performed in simulating the development and spread of fire for purposes of tracking its movement in an environment (e.g., [2] [16]). These models tend to be only global, extrinsic representations of the fire's movement and less concerned with the intrinsic motion of the fire itself. Falling somewhere between these two extremes, procedural image generation and particle systems provide visually effective, yet computationally attractive, approaches to fire.

Procedurally generated image

A particularly simple way to generate the appearance of fire is to procedurally generate a two-dimensional image by coloring pixels suggestive of flames and smoke. The procedure iterates through pixels of an image buffer and sets the palette indices based on the indices of the surrounding pixels [14]. Modifying the pixels top to bottom allows the imagery to progress up the buffer. By using multiple buffers and the alpha (transparency) channel, a limited three-dimensional effect can be achieved. In Figure 8.14 (Color Plate 3), a color palette filled with hues from red to yellow is used to hold RGB values. The bottom row of the image buffer is randomly initialized with color indices.

Particle system approach

One of the first and most popularly viewed examples of computer-generated fire appears in the movie *Star Trek II: The Wrath of Khan* [13]. In the sequence referred to as the *genesis effect*, an expanding wall of fire spreads out over the surface of the planet from a single point of impact. The simulation is not a completely convincing model of fire, although the sequence is effective in the movie. The model uses a two-level hierarchy of particles. The first level of particles is located at the point of impact to simulate the initial blast (Figure 8.15); the second level consists of concentric rings of particles, timed to progress from the central point outward, forming the wall of fire and explosions.

Each ring of second-level hierarchy consists of a number of individual particle systems that are positioned on the ring and overlap to form a continuous ring. The individual particle systems are

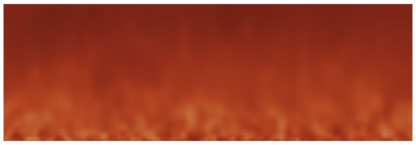


FIGURE 8.14

Image representing fire generated using pixel averaging operations.

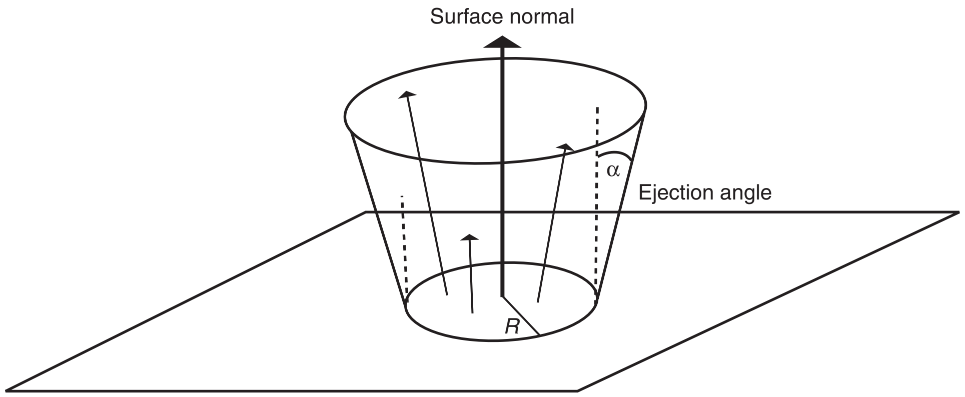


FIGURE 8.15

Explosion-like particle system.

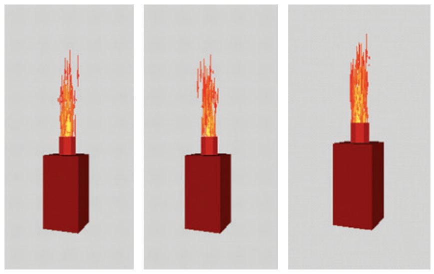


FIGURE 8.16

Images from a particle system simulation of fire.

modeled to look like explosions. The particles in each one of these particle systems are oriented to fly up and away from the surface of the planet. The initial position for a particle is randomly chosen from the circular base of the particle system. The initial direction of travel for each particle is constrained to deviate less than the ejection angle away from the surface normal. Figure 8.16 (Color Plate 4) shows a simple particle system fire.

Other approaches

Various other approaches have been used in animations with varying levels of success. Two-dimensional animated texture maps have been used to create the effect of the upward movement of burning gas, but such models are effective only when viewed from a specific direction. Using a two-dimensional multiple planes approach adds some depth to the fire, but viewing directions are still limited. Stam and Fiume [18] present advection-diffusion equations to evolve both density and temperature fields. The user controls the simulation by specifying a wind field. The results are effective, but the foundation mathematics are complicated and the model is difficult to control.

8.1.4 Summary

Modeling and animating amorphous phenomena is difficult. Gases are constantly changing shape and lack even a definable surface. Volume graphics hold the most promise for modeling and animating gas, but currently they have computational drawbacks that make such approaches of limited use for animation. A useful and visually accurate model of fire remains the subject of research.

8.2 Computational fluid dynamics

Gases and liquids are referred to collectively as *fluids*. The study of methods to compute their behavior is called CFD.

Fluids are composed of molecules that are constantly moving. These molecules are constantly colliding with one another and with other objects. The molecular motion transports measurable amounts of fluid properties throughout the fluid. These properties include such things as density, temperature, and momentum. In CFD, the assumption is that the fluid is a continuous medium in the sense that these properties are well-defined at infinitely small points of the fluid and that the property values are smoothly varying throughout the fluid. This is called the *continuity assumption*.

Modeling gaseous phenomena (smoke, clouds, and fire) is particularly challenging because of their ethereal nature. Gas has no definite geometry, no definite topology. Gas is usually treated as *compressible*, meaning that density is spatially variable and computing the changes in density is part of the computational cost. Liquids are usually treated as *incompressible*, which means the density of the material remains constant. In fact, the equations in Section 8.1.1 were derived from the CFD equations.

In a *steady-state flow*, the motion attributes (e.g., velocity and acceleration) at any point in space are constant. Particles traveling through a steady-state flow can be tracked similarly to how a space curve can be traced out when the derivatives are known. *Vortices*, circular swirls of material, are important features in fluid dynamics. In steady-state flow, vortices are attributes of space and are time invariant. In time-varying flows, particles that carry a non-zero vortex strength can travel through the environment and can be used to modify the acceleration of other particles in the system by incorporating a distance-based force.

8.2.1 General approaches to modeling fluids

There are three approaches to modeling gas: grid-based methods (*Eulerian formulations*), particle-based methods (*Lagrangian formulations*), and hybrid methods. The approaches are illustrated here in two dimensions, but the extension to three dimensions should be obvious.

Grid-based method

The grid-based method decomposes space into individual cells, and the flow of the gas into and out of each cell is calculated (Figure 8.17). In this way, the density of gas in each cell is updated from time step to time step. The density in each cell is used to determine the visibility and illumination of the gas during rendering. Attributes of the gas within a cell, such as velocity, acceleration, and density, can be used to track the gas as it travels from cell to cell.

The flow out of the cell can be computed based on the cell velocity, the size of the cell, and the cell density. The flow into a cell is determined by distributing the densities out of adjacent cells. External forces, such as wind and obstacles, are used to modify the acceleration of the particles within a cell.

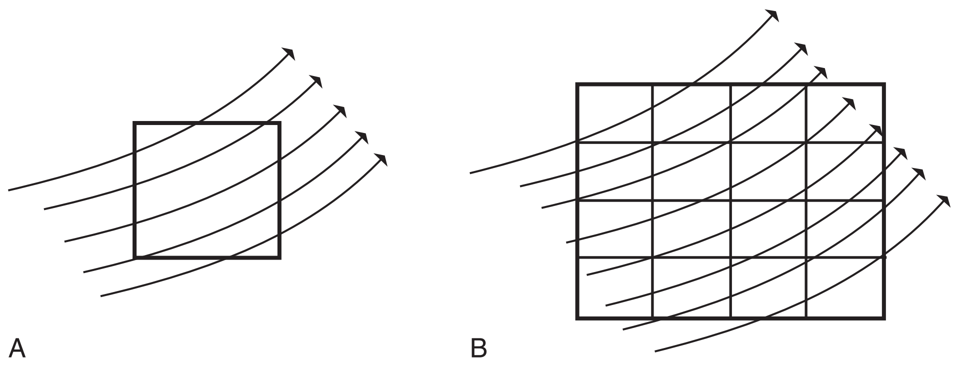


FIGURE 8.17

Grid-based method. (A) Gas flowing through an individual cell. (B) Grid of cells.

The rendering phase uses standard volumetric graphics techniques to produce an image based on the densities projected onto the image plane. Illumination of a cell from a light source through the volume must also be incorporated into the display procedure.

The disadvantage of this approach is that if a static data structure for the cellular decomposition is used, the extent of the interesting space must be identified before the simulation takes place in order to initialize the cells that will be needed during the simulation of the gas phenomena. Alternatively, a dynamic data structure that adapts to the traversal of the gas through space could be used, but this increases the complexity of the simulation.

Particle-based method

In the particle-based method, particles or globs of gas are tracked as they progress through space, often with a standard particle system approach (Figure 8.18). The particles can be rendered individually, or they can be rendered as spheres of gas with a given density. The advantage of this technique is that it is similar to rigid body dynamics and therefore the equations are relatively simple and familiar. The equations can be simplified if the rotational dynamics are ignored. In addition, there are no restrictions imposed by the simulation setup as to where the gas may travel. Particles are assigned masses, and external forces can be easily incorporated by updating particle accelerations and, subsequently, velocities. The disadvantage of this approach is that a large number of particles are needed to simulate a dense expansive gas and that to display the surface of a liquid, the boundary between and fluid and nonfluid must be reconstructed from the particle samples. Recent techniques generalize on the idea of using rigidly defined particles to include a particle that represents a distribution of fluid mass in the surrounding area.

Hybrid method

Some models of gas trace particles through a spatial grid. Particles are passed from cell to cell as they traverse the interesting space (Figure 8.19). The display attributes of individual cells are determined by the number and type of particles contained in the cell at the time of display. The particles are used to carry and distribute attributes through the grid, and then the grid is used to produce the display.

8.2.2 CFD equations

In developing the CFD calculations, the volume occupied by a small (*differential*) fluid element is analyzed. Differential equations are created by describing what is happening at this element. (Figure 8.20).

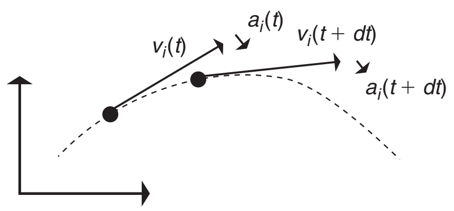


FIGURE 8.18

Particle-based method.

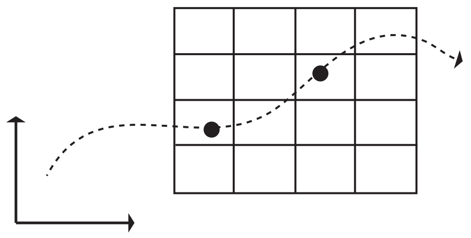


FIGURE 8.19

Hybrid method.

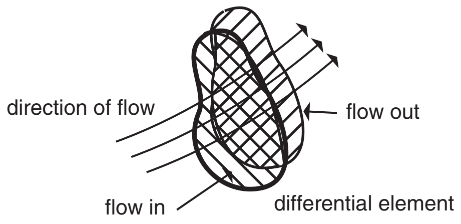


FIGURE 8.20

Differential element used in Navier-Stokes.

The full CFD equations consist of equations that state, in a closed system, the following:

- Mass is conserved.
- Momentum is conserved.
- Energy is conserved.

When expressed as equations, these are the *Navier-Stokes* (NS) equations. The NS equations are nonlinear differential equations. There are various simplifications that can be made in order to make the NS equations more tractable. First of all, for most all graphics applications, energy conservation is ignored. Another common simplification used in graphics is to assume the fluid is nonviscous. These assumptions result in what are called *Euler equations* [12]. By ignoring energy conservation and viscosity, the Euler equations describing the conservation of mass and conservation of momentum can be readily constructed for a given flow field. When dealing with liquids (as opposed to a gas), another useful assumption is that the fluid is incompressible, which means that its density does not change.

Conservation of mass

Conservation of mass is expressed by the *continuity equation* [4]. The underlying assumption of the conservation of mass is that mass is neither created nor destroyed inside of the fluid. The fluid flows from location to location and, if compressible, can become more dense or less dense at various locations. To determine how the total mass inside of an element is changing, all of the sources and sinks in the element have to be added up. The *divergence* of a flow field describes the strength of the source or sink at a location; the divergence for a field, F = (Fx, Fy, Fz), is $div\, F = \nabla \cdot F = \frac{\partial F_x}{\partial x} + \frac{\partial F_y}{\partial y} + \frac{\partial F_z}{\partial z}$. If no mass is created or destroyed inside the element, then it is reasonable that the total divergence within the element must be equal to the flow differences across the element at the x, y, and z boundaries. This is stated by the *Divergence Theorem*: the integral of the flow field's divergence over the volume of the element is the same as the integral of the flow field over the element's boundary. So, instead of worrying about all the sources and sinks inside of the element, all that has to be computed is the flow at the boundaries.

For now, for a given three-dimensional element, consider only flow in the x-direction. To make the discussion a bit more concrete, assume that the fluid is flowing left to right, as x increases, so that the mass flows into a element on the left, at x, and out of the element on the right, at $x + dx$. In the following equations, ρ is density, p is pressure, A is the area of the face perpendicular to the x-direction, V is the volume of the element, and the element dimensions are dx by dy by dz. To compute the mass flowing across a surface, multiply the density of the fluid times area of the surface times the velocity component normal to the surface, v_x in this case. The conservation of mass equation states that the rate at which mass changes is equal to difference between mass flowing into the element and the mass flowing out of the element (Eq. 8.29). The notation for a value at a given location is $< value >(_{\text{location}}$ or, when appropriate, $< value >(_{\text{location surface}}$.

$$-\frac{\partial(\rho V)}{\partial t} = (\rho v_x A)|_{x+dx} - (\rho v_x A)|_x \tag{8.29}$$

Replace the volume, V, and area, A, by their definitions in terms of the element dimensions (Eq. 8.30).

$$-\frac{d(\rho dxdydz)}{dt} = (\rho v_x dydz)|_{x+dx} - (\rho v_x dydz)|_x \tag{8.30}$$

Divide through by $dxdydz$ (Eq. 8.31).

$$-\frac{\partial \rho}{\partial t} = \frac{\rho v_x}{dx}|_{x+dx} - \frac{\rho v_x}{dx}|_x \tag{8.31}$$

Replace the finite differences by their corresponding differential forms (Eq. 8.32).

$$-\frac{\partial \rho}{\partial t} = \frac{\partial(\rho v_x)}{\partial x} \tag{8.32}$$

Finally, include flow in all three directions (Eq. 8.33).

$$-\frac{\partial \rho}{\partial t} = \frac{\partial(\rho v_x)}{\partial x} + \frac{\partial(\rho v_y)}{\partial y} + \frac{\partial(\rho v_z)}{\partial z} \tag{8.33}$$

If ∇ is used as the gradient operator, $\nabla F = \left(\frac{\partial F}{\partial x}, \frac{\partial F}{\partial y}, \frac{\partial F}{\partial z}\right)$ then a common notation is to define the divergence operator as in Equation 8.34.

$$\nabla \cdot \mathrm{F} = \frac{\partial F_x}{\partial x} + \frac{\partial F_y}{\partial y} + \frac{\partial F_z}{\partial z} \tag{8.34}$$

Using the divergence operator in Equation 8.33 produces Equation 8.35.

$$-\frac{\partial \rho}{\partial t} = \nabla \cdot (\rho v) \tag{8.35}$$

If the fluid is incompressible, then the density does not change. This means that $\frac{\partial \rho}{\partial t}$ is zero and the constant density can be divided out, resulting in Equation 8.36.

$$0 = \nabla \cdot \mathrm{v} \tag{8.36}$$

Conservation of momentum

The momentum of an object is its mass times its velocity. The momentum of a fluid is its density times the volume of the fluid times the average velocity of the fluid.

A change in momentum $\frac{d(mv)}{dt}$ must be induced by a force (i.e., $f = ma = m\frac{dv}{dt}$). The force in the element of a flow field is either a global force, such as viscosity and gravity, or a change in pressure across the element, $\frac{dp}{dx}$. As noted previously, effects of viscosity are commonly ignored in order to simplify the equations; we will also ignore gravity for now.

For an element, the change in momentum is the change of momentum inside the element plus the difference between the momentum entering the cell and the momentum leaving the element. The mass inside the cell is ρV and, thus, the momentum inside the element is $\rho V v$. The rate of change of momentum inside the element is $\frac{\partial (pVv)}{\partial t}$.

Given a three-dimensional element, consider flow only in the x-direction. The mass flowing out one surface is density × area of the surface times velocity in the x-direction, v_x. The momentum out one surface is the mass flowing out the surface times its velocity, $\rho\, A v_x v$. The force on the element in the x-direction is the pressure difference across the element from x to $x+dx$. The force is equal to the negative of the change in momentum Equation 8.37.

$$-\left(p|^{A}_{x+dx} - p|^{A}_{x}\right) = \frac{\partial(\rho V v)}{\partial t} + \left((\rho v_x A)v|_{x+dx} - (\rho v_x A)v|_{x}\right) \tag{8.37}$$

Following the steps above in developing the conservation of mass equation by replacing the area, A, with its definition in terms of element dimensions, $dydz$, replacing the volume, V, with $dxdydz$, dividing through by $dxdydz$, and putting everything in differential form gives Equation 8.38.

$$-\frac{\partial p}{\partial x} = \frac{\partial(\rho v_x \mathrm{v})}{\partial x} + \frac{\partial(\rho \mathrm{v})}{\partial t} \tag{8.38}$$

Now, considering flow in all three directions, and separating out momentum only in the x direction, produces Equation 8.39.

$$-\frac{\partial p}{\partial t} = \frac{\partial\left(\rho v_x^2\right)}{\partial x} + \frac{\partial\left(\rho v_x v_y\right)}{\partial y} + \frac{\partial(\rho v_x v_z)}{\partial z} + \frac{\partial(\rho v_x)}{\partial t} \tag{8.39}$$

If desired, viscosity and other forces can be added to the pressure differential force on the left-hand side of the equation. Similar equations can be derived for the y and z directions.

8.2.3 Grid-based approach

The basic process for computing fluid dynamics using a grid-based (Eulerian) approach is to do the following:

- Discretize the fluid continuum by constructing a grid of cells that, collectively, cover the space occupied by the fluid and at each node (cell), the fluid variables (e.g., density and velocity) are approximated and used to initialize the grid cells.
- Create the discrete equations using the approximate values at the element.
- Numerically solve the system of equations to compute new values at the cells using a Newton-like method for the large, sparse system of equations at each time step.

Before solving these equations, boundary conditions must be set up to handle the cells at the limits of the domain. For example, the *Dirichlet boundary condition*, which sets solution values at the boundary cells, is commonly used. The CFD equations, set up for each cell of the grid, produce a sparse set of matrix equations. There are various ways to solve such systems. For example, LU and Conjugate Gradient solvers are often used. Efficient, accurate methods of solving symmetric and asymmetric sparse matrices, such as these, are the topic of ongoing research. Thankfully, in computer animation believability is more of a concern than accuracy, so approximate techniques typically suffice.

Stable fluids

Stam presents a method that solves the full NS equations that is easy to implement and allows real-time user interaction [17]. Although too inaccurate for engineering, it is useful in computer graphics applications. In particular, the procedures are designed to be stable more than accurate while still producing interesting visuals. The procedures operate on a velocity vector field and density field. The velocity field moves the density values around.

Density update

Density values are moved through the density grid assuming a fixed velocity field over the time step. The density equation (Eq. 8.40) states that density (ρ) changes over time according to density sources (*s*), the diffusion of the density at some specified rate (*k*), and advection of the density according to the velocity field (**u**).

$$-\frac{\partial \rho}{\partial t} = s = k\nabla^2 \rho - (\mathrm{u} \cdot \nabla)\rho \tag{8.40}$$

In implementing stable fluids, density is kept in an array of scalars. The first term, *s*, is an array of density sources set by user interaction or by the environment. These are used to initialize the new density grid at each time step. The second term diffuses the density from each cell to its adjacent cells. The diffusion of the grid can be solved by relaxation. The third term advects the density according to the velocity field.

To diffuse, each cell is updated by a weighted average of its four neighbors added to the original value. Each neighboring value is multiplied by a where $a = dt*diff*N*N$, where *diff* is a user-supplied parameter, and N is the resolution of the grid. This sum is added to the cell's original value and then this sum is divided by $1 + 4*a$.

To advect, the position is interpolated backward according to the velocity field. In the simplest case, the velocity vector at the particle position is accessed and linearly stepping backward according to the selected time step, making sure to clamp within the grid boundaries. The density is then bilinearly interpolated from nearby values and this interpolated value is used as the density of the position.

The velocity update

The change of the velocity over time is given by Equation 8.41 and is due to external forces (*f*), diffusion of the velocities, and self-advection.

$$-\frac{\partial u}{\partial t} = f + \mu \nabla^2 \mathrm{u} - (\mathrm{u} \cdot \nabla)\mathrm{u} \tag{8.41}$$

The velocity is kept in an array of vectors. The first term, *f*, adds in external forces from the user or environment. The second term allows for the diffusion of the velocity, and the third term allows the velocity to be carried along by the velocity field.

These equations are implemented by the sequence: add force, advect, diffuse, and project. The projection step is to adjust the density so that mass is preserved.

The simulation

The simulation is conducted by updating the contributions from density forces and external forces, then stepping the velocity field a step in time, taking the density field a step in time, then drawing the density.

8.2.4 Particle-based approaches including smoothed particle hydrodynamics

As previously noted, particle-based approaches have the advantage of being intuitive, easy to program, and fast to execute—at the expense of accuracy. However, for computer graphics and animation, accuracy is a common sacrifice. The basic idea is to model the fluid by a mass of particles. A simple particle system can be used, but the rigid nature of standard particles means that a massive amount of particles has to be used in order to create the impression of a fluid. In cases of modeling a gas or a spray of liquid this can be effective, but in cases where it is useful to represent a liquid's surface (e.g., for display) this approach is problematic.

In order to give a better impression of a cohesive fluid, spherical implicit surfaces, often referred to as *metaballs* (see Chapter 12.1), can be used in place of rigid particles. Because metaballs allow for the reconstruction of smooth intersections between particles, this approach can produce a gooey-like appearance similar to crunchy peanut butter. Metaballs have the ability to generate organic, blobby shapes but fall short of being able to model liquids effectively. While this is an improvement over rigid particles, is still takes a very large number of particles to produce a smooth surface. In addition, metaballs lack any physical characteristics that make them suitable for effectively representing liquids.

For a more principled approach and one that produces a smoother surface, smooth particle hydrodynamics (SPH), developed for astrophysics, does a better and faster job of rendering a liquid from a collection of particles. SPH can be viewed as an extension of the metaball approach (Figure 8.21). The SPH particle is similar to that in a typical particle system and is represented by its position and velocity. An SPH particle, similar to an implicit surface particle, has a density function that contributes to a field value over its area of influence. The difference is that each SPH particle has a mass and density and each SPH particle also carries and distributes its mass over an area. The SPH particle has a kernel function that plays the role of a metaball's density function. The main difference

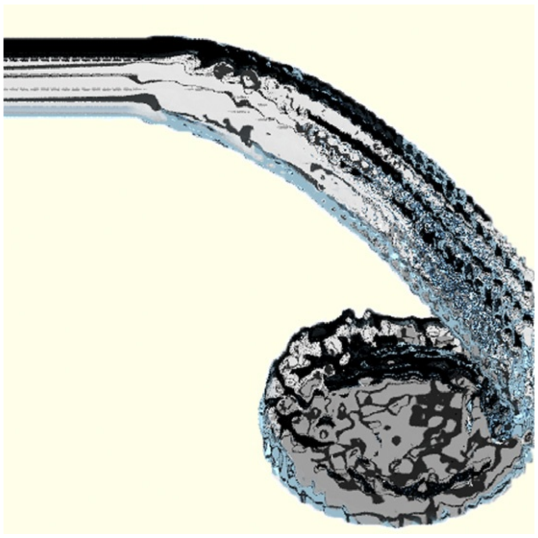


FIGURE 8.21

Water simulation using sSPH.

(Image courtesy of Di Cao.)

between an SPH particle and an implicitly defined sphere is that one of the kernel's parameters is its support distance, which is a function of its mass. It should be noted that there are different ways to model particles and compute values in SPH and that the material here follows the work of Muller et al. [21][22]. The interested reader can refer to these papers for more details.

In implementing the fluid dynamics using SPH particles, the main concerns are about conserving mass and updating the fluid particle representation over time due to the forces on the particles.

An additional practical consideration is how to display the fluid—how to reconstruct the fluid from the particle definition at any one point in time for display—and, especially for liquid, how to identify its surface. Similar to a standard particle system, at any one point in time a particle has a position and a velocity and is subject to various forces. The forces considered are those captured in the CFD equations—the pressure gradient and viscosity of the fluid, as well as environmental forces such as gravity. In addition, in the case of liquids, a surface tension force can be modeled.

An SPH particle carries an additional property of mass. This mass is distributed through space according to a smoothing kernel function. In fact, any property of the fluid can be distributed through space using a smoothing function. A property, s, at any given location, r, can be computed using Equation 8.42. In particular, the density can be computed from Equation 8.42, resulting in Equation 8.44. The gradient of a property and the Laplacian of a property, useful in computer pressure gradient and viscosity, can be computed using Equation 8.43.

$$s(r) = \sum_j m_j \frac{s_j}{\rho_j} W_h \left(r - r_j\right) \tag{8.42}$$

$$\begin{aligned} \nabla s(r) &= \sum_j m_j \frac{s_j}{\rho_j} \nabla W_h(r - r_j) \\ \nabla^2 s(r) &= \sum_j m_j \frac{s_j}{\rho_j} \nabla^2 W_h(r - r_j) \end{aligned} \tag{8.43}$$

$$\rho(r) = \sum_j m_j W_h(r - r_j) \rho_s(r) = \sum_j m_j W_h(r - r_j) \tag{8.44}$$

The use of particles guarantees that mass is conserved as long as the mass of a particle does not change and as long as particles are not created or destroyed. Updating the fluid according to CFD principles is effected by migration of particles through space due to the forces modeled by the CFD equations. First, the density at a location is computed using Equation 8.43.

Then the pressure is computed using the ideal gas state equation $p = k(\rho - \rho_0)$, where ρ_0 is the rest density for water and k is a constant in the range of 100 to 1000. Forces derived directly by SPH are not guaranteed to be symmetric, so symmetric versions of the force equations are then computed (Eq. 8.45):

$$\begin{aligned} f_{pressure}(r_i) &= -\sum_j M_j \frac{p_i + p_j}{2\rho_j} \nabla W(r_i - r_{j,} h) \\ f_{gravity}(r_i) &= \rho_i \mathrm{g} \\ f_{viscosity}(r_i) &= \mu \sum_j m_j \frac{v_j - v_i}{\rho_j} \nabla^2 W(r_i - r_{j,} h) \end{aligned} \tag{8.45}$$

Quantities in the particle field can be computed at any given location by the use of a radial, symmetric, smoothing kernel. Smoothing kernels with a finite area of support are typically used for computational efficiency. Smoothing kernels should integrate to 1, which is referred to as *normalized*. Different smoothing kernels can be used when computing various values as long as the smoothing kernel has the basic properties of being normalized and symmetric. As examples, the kernels used in the work being followed here are a general purpose kernel (Eq. 8.46), a kernel for pressure to avoid a zero gradient at the center (Eq. 8.47), and a kernel for the viscosity (Eq. 8.48) and its corresponding Laplacian (Eq. 8.49). In this work, a Leap Frog integrator is used to compute the accelerations from the forces.

$$W_{poly6}(r, h) = \frac{315}{64\pi h^9} \begin{cases} (h^2 - r^2) & 0 \le r \le hb \\ 0 & otherwise \end{cases} \tag{8.46}$$

$$W_{spiky}(r, h) = \frac{15}{\pi h^6} \begin{cases} (h - r)^3 & 0 \le r \le h \\ 0 & otherwise \end{cases} \tag{8.47}$$

$$W_{viscosity}(r, h) = \frac{15}{2\pi h^3} \begin{cases} -\frac{r^3}{2h^3} + \frac{r^2}{h^2} + \frac{h}{2r} - 1 & 0 \le r \le h \\ 0 & otherwise \end{cases} \tag{8.48}$$

$$\nabla^2 W_{viscosity}(r, h) = \frac{45}{\pi h^6}(h - r) \tag{8.49}$$

8.3 Chapter summary

Most of the techniques discussed in this chapter are still the subject of research efforts. Water, smoke, clouds, and fire share an amorphous nature, which makes them difficult to model and animate. Approaches that incorporate a greater amount of physics have been developed recently for these phenomena. As processing power becomes cheaper, techniques such as CFD become more practical (and more desirable) tools for animating water and gas, but convenient controls for such models have yet to be developed.

References

[1] Blinn J. Simulation of Wrinkled Surfaces. In: Computer Graphics. Proceedings of SIGGRAPH 78, 12(3). Atlanta, Ga.; August 1978. p. 286–92.

[2] Bukowski R, Sequin C. Interactive Simulation of Fire in Virtual Building Environments. In: Whitted T, editor. Computer Graphics. Proceedings of SIGGRAPH 97, Annual Conference Series. Los Angeles, Calif.: Addison-Wesley; August 1997. p. 35–44. ISBN 0-89791-896-7.

[3] Ebert D, Musgrave K, Peachey D, Perlin K, Worley S. Texturing and Modeling: A Procedural Approach. Cambridge, Massachusetts: AP Professional; 1998.

[4] Engineers Edge. Continuity Equation—Fluid Flow, http://www.engineersedge.com/fluid_flow/continuity_equation.htm**; August 2005.**

[5] Foster N, Metaxas D. Realistic Animation of Liquids. Graphical Models and Image-Processing, September 1996;58(5):471–83. Academic Press.

[6] Fournier A, Reeves W. A Simple Model of Ocean Waves. In: Evans DC, Athay RJ, editors. Computer Graphics. Proceedings of SIGGRAPH 86, 20(4). Dallas, Tex.; August 1986. p. 75–84.

[7] Fournier P, Habibi A, Poulin P. Simulating the Flow of Liquid Droplets. In: Booth K, Fournier A, editors. Graphics Interface '98. June 1998. p. 133–42. ISBN 0-9695338-6-1.

[8] Gardner G. Simulation of Natural Scenes Using Textured Quadric Surfaces. In: Computer Graphics. Proceedings of SIGGRAPH 84, 18(3). Minneapolis, Minn.; July 1984. p. 11–20.

[9] Gardner G. Visual Simulation of Clouds. In: Barsky BA, editor. Computer Graphics. Proceedings of SIGGRAPH 85, 19(3). San Francisco, Calif.; August 1985. p. 297–303.

[10] Kass M, Miller G. Rapid, Stable Fluid Dynamics for Computer Graphics. In: Baskett F, editor. Computer Graphics. Proceedings of SIGGRAPH 90, 24(4). Dallas, Tex.; August 1990. p. 49–57. ISBN 0-201-50933-4.

[11] Max N. Vectorized Procedural Models for Natural Terrains: Waves and Islands in the Sunset. In: Computer Graphics. Proceedings of SIGGRAPH 81, 15(3). Dallas, Tex.; August 1981. p. 317–24.

[12] NASA. Euler Equations, http://www.grc.nasa.gov/WWW/K-12/airplane/eulereqs.html**; August, 2005.**

[13] Paramount. Star Trek II: The Wrath of Khan (film). June 1982.

[14] Parusel A. Simple Fire Effect, GameDev.net; January, 2007. http://www.gamedev.net/reference/articles/article222.asp.

[15] Peachey D. Modeling Waves and Surf. In: Evans DC, Athay RJ, editors. Computer Graphics. Proceedings of SIGGRAPH 86, 20(4). Dallas, Tex.; August 1986. p. 65–74.

[16] Rushmeier H, Hamins A, Choi M. Volume Rendering of Pool Fire Data. IEEE Comput Graph Appl July 1995;15(4):62–7.

[17] Stam J. Stable Fluids. In: Rockwood A, editor. Computer Graphics. Proceedings of SIGGRAPH 99, Annual Conference Series. Los Angeles, Calif.: Addison-Wesley Longman; August 1999. p. 121–8. ISBN 0-20148-560-5.

[18] Stam J, Fiume E. Depicting Fire and Other Gaseous Phenomena Using Diffusion Processes. In: Computer Graphics. Proceedings of SIGGRAPH 95), In Annual Conference Series. Los Angeles, Calif: ACM SIGGRAPH; August 1995. p. 129–36.

[19] Tessendorf J. Simulating Ocean Water. In: SIGGRAPH 2002 Course Notes #9 (Simulating Nature: Realistic and Interactive Techniques). ACM Press; 2002.

[20] Weimer H, Warren J. Subdivision Schemes for Fluid Flow. In: Rockwood A, editor. Computer Graphics. Proceedings of SIGGRAPH 99, Annual Conference Series. Los Angeles, Calif.: Addison-Wesley Longman; August 1999. p. 111–20. ISBN 0-20148-560-5.

[21] Muller M, Charypar D, Gross M. Particle-based Fluid Simulation for Interactive Applications. In: Eurographics/SIGGRAPH Symposium on Computer Animation. 2003.

[22] Muller M, Schirm S, Teschner M. Interactive Blood Simulation for Virtual Surgery Based on Smoothed Particle Hydrodynamics. Technology and Health Care April 19, 2004;12(1/2004):25–31.

[23] WeatherOnline Ltd. Weather Facts: Cloud Types, www.weatheronline.co.uk/reports/wxfacts/Cloud-types.html**; Nov. 2011.**