9

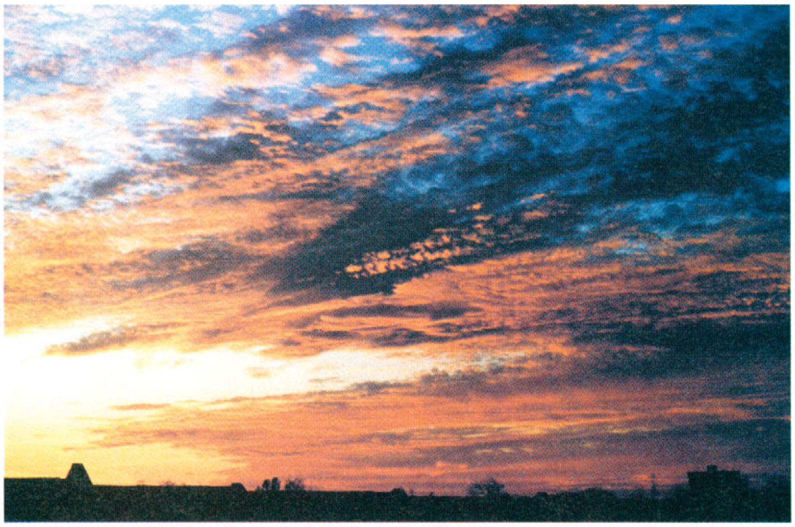
VOLUMETRIC CLOUD MODELING WITH IMPLICIT FUNCTIONS

DAVID S. EBERT

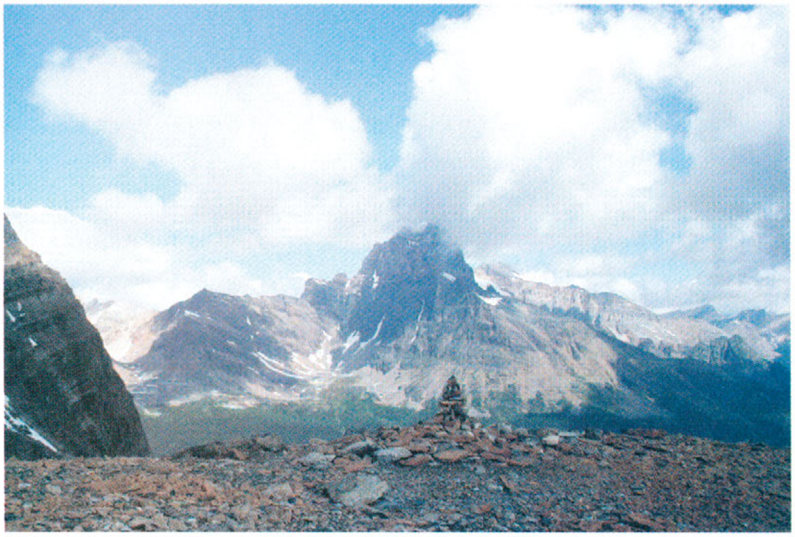
Modeling clouds is a very difficult task because of their complex, amorphous structure and because even an untrained eye can judge the realism of a cloud model. Their ubiquitous nature makes them an important modeling and animation task. Several examples of the wide-ranging beauty of clouds can be seen in Figure 9.1. This chapter provides an introduction to the important visual and physical characteristics of clouds and important rendering issues for clouds. An overview of previous approaches to cloud modeling and animation, as well as recent approaches to interactive cloud modeling, is presented next. Finally, this chapter describes my volumetric procedural approach for cloud modeling and animation that allows easy, natural specification and animation of the clouds, provides the flexibility to include as much physics or art as desired into the model, unburdens the user from detailed geometry specification, and produces realistic volumetric cloud models. This technique combines the flexibility of volumetric procedural modeling with the smooth blending and ease of control of primitive-based implicit functions (meta-balls, blobs) to create a powerful new modeling technique. This technique also demonstrates the advantages of primitive-based implicit functions for modeling semitransparent volumetric objects. An exploration of how this cloud modeling approach can be adapted for real-time rendering can be found in Chapter 10.

CLOUD BASICS

Clouds are made of visible ice crystals and/or water droplets suspended in air, depending on the altitude of the cloud and, hence, the air temperature. Clouds are formed by air rising, cooling the water vapor in the air to its saturation point, and condensing onto small particles in the atmosphere. The visible condensed water vapor forms a cloud (University of Illinois 2002). Cloud shape varies based on the process that forces the air to rise or bubble up, the height at which the clouds form, and various other conditions (University of Illinois 2002). These air-lifting forces include



(a)



(b)

FIGURE 9.1 Several example photographs of real clouds. Copyright © 2002 David S. Ebert.

(c)

(d)

convection, convergence, lifting along frontal boundaries, lifting due to mountains (called *orographic lifting*), and Kelvin-Helmholtz shearing. Kelvin-Helmholtz shearing (billows) occurs when there is vertical wind shear and produces clouds that look similar to water waves or ripples. Several sources present a very nice introduction to clouds and their identification (University of Illinois 2002; Tricker 1970; Cotton and Anthes 1989; Houze 1993).

When considering cloud altitude, clouds formed above 20,000 feet (e.g., cirrus, cirrostratus) are often thin and wispy in appearance and composed primarily of ice crystals because of the cold temperatures. Clouds formed between 6500 feet and 23,000 feet (e.g., altocumulus) are primarily formed of water droplets and have the appearance of collections of small puffy clouds, sometimes in waves. Clouds formed below 6500 feet (e.g., stratus, stratocumulus) are, again, primarily comprised of water droplets and have the appearance of large clouds in layers. The most characteristic cloud type is the puffy cumulus cloud. Cumulus clouds are normally formed by convection or frontal lifting and can vary from having little vertical height to forming huge vertical towers (cumulonimbus) created by very strong convection.

The visual appearance of clouds not only is useful for creating more natural images and animations but is also very important for weather forecasters. Weather spotters are trained to look for key components of the visual appearance of clouds, enabling them to determine information on potential storms that cannot be collected from weather radar and other measurements. Clouds have several easily identifiable visual characteristics that must be modeled to produce accurate images and animations. First of all, clouds have a volumetrically varying amorphous structure with detail at many different scales. Second, cloud formation often results from swirling, bubbling, turbulent processes that produce the characteristic cloud patterns and their evolution over time. Finally, they have several illumination/shading characteristics that must be accurately rendered to obtain convincing images. Clouds are a three-dimensional medium of small ice and water droplets that absorb, scatter, and reflect light.

Illumination models for clouds are classified as low-albedo and high-albedo models. A low-albedo reflectance model assumes that secondary scattering effects are negligible, while a high-albedo illumination model calculates the secondary and higher-order scattering effects. For optically thick clouds, such as cumulus, stratus, and cumulonimbus, secondary scattering effects are significant, and high-albedo illumination models (e.g., Blinn 1982a; Kajiya and Von Herzen 1984; Rushmeier and Torrance 1987; Max 1994; Nishita, Nakamae, and Dobashi 1996) should be used. Detailed descriptions of implementing a low-albedo illumination algorithm can be found in several sources (Kajiya and Von Herzen 1984; Ebert and Parent 1990).

Simulation of wavelength-dependent scattering is also important to create correct atmospheric dispersion effects for sunrise and sunset scenes (see Figures 9.4 and 9.5 for example renderings of clouds with sunset illumination). Self-shadowing of clouds and cloud shadowing on landscapes are also important to create realistic images of cloud scenes and landscapes. Correct cloud shadowing requires volumetric shadowing techniques to create accurate images, which can be very expensive when volumetric ray tracing is used. As mentioned in Chapter 7, a much faster alternative is to use volumetric shadow tables (Kajiya and Von Herzen 1984; Ebert and Parent 1990) or hardware-based 3D texture slicing (Kniss, Kindlmann, and Hansen 2002).

SURFACE-BASED CLOUD MODELING APPROACHES

Modeling clouds in computer graphics has been a challenge for nearly 20 years (Dungan 1979), and major advances in cloud modeling still warrant presentation in the SIGGRAPH Papers Program (e.g., Dobashi et al. 2000). Many previous approaches have used semitransparent surfaces to produce convincing images of clouds. Voss (1983) introduced the idea of using fractal synthesis of parallel plane models to produce images of clouds seen from a distance. Gardner (1984, 1985, 1990) produces very impressive images and animations of clouds by using Fourier synthesis[1] to control the transparency of large, hollow ellipsoids. In his approach, large collections of ellipsoids define the general shape of the clouds, while the Fourier synthesis procedurally generates the transparency texture and creates the cloud detail. Kluyskens (2002) uses a similar approach to produce clouds in Alias/Wavefront's Maya animation system. He uses randomized, overlapping spheres to define the general cloud shape. A solid-texture "cloud texture" is then used to color the cloud and to control the transparency of the spheres. Finally, the transparency of the spheres near their edges is increased so that the defining spherical shape is not noticeable. Gardner's approach has been extended to real-time cloud rendering by using OpenGL extensions and adapting the transparency calculations for easier hardware implementation using subtracting blending modes (Elinas and Stürzlinger 2000).

Even simpler approaches to clouds are often used for interactive applications, such as games. One of the most common techniques is to create a 2D cloud texture that is texture mapped onto the sky dome of the scene. These textures are commonly generated using noise-based textures, and a real-time approach to generating procedural cloud textures using multitexturing and hardware blending can be found in

1. In this case, sums of cosines with different amplitudes and periods control the textured transparency.

Pallister (2002). The use of billboards and imposters[2] allows more interaction with the clouds in the scene. As with most other approaches using polygonal surfaces to approximate clouds, a Gaussian or exponential falloff of the transparency toward the edge of the cloud is needed so that the defining shape cannot be seen.

VOLUMETRIC CLOUD MODELS

Although surface-based techniques can produce realistic images of clouds viewed from a distance, these cloud models are hollow and do not allow the user to seamlessly enter, travel through, and inspect the interior of the cloud model. To capture the three-dimensional structure of a cloud, volumetric density-based models must be used. Kajiya and Von Herzen (1984) produced the first volumetric cloud model in computer graphics, but the results are not photorealistic. Stam and Fiume (1995) and Foster and Metaxas (1997) have produced convincing volumetric models of smoke and steam, but have not done substantial work on modeling clouds.

Neyret (1997) has recently produced some preliminary results of a convective cloud model based on general physical characteristics, such as bubbling and convection processes. This model may be promising for simulating convective clouds; however, it currently uses surfaces (large particles) to model the cloud structure. Extending this approach to volumetric cloud modeling and animation should produce very convincing images and animations.

Several researchers have combined volume modeling of clouds with surface-based imposter rendering. Mark Harris has recently extended the imposter technique to generate real-time flythroughs of complex, static cloudscapes (Harris and Lastra 2001; Harris 2002). Spherical particles are used to generate volumetric density distributions for clouds (approximately 200 particles per cloud), and a multiple scattering illumination model is precomputed for quicker shading of the particles. The particles are then rendered using splatting (Westover 1990) to create the imposters for the clouds. To allow the viewer to correctly fly through the environment and the actual clouds, the imposters need to be updated frequently. Dobashi et al. (2000) also have used imposters to allow quick rendering of clouds that are modeled using a partially physics-based cellular automata approach. However, the preintegration of the imposters for rendering limits the performance of this approach. As previously

2. An *imposter* is a texture-mapped polygon with the precomputed rendering of an object (color and alpha) mapped onto simple geometry to speed rendering. The texture-mapped image needs to be updated as the view changes to reflect the appropriate image of viewing the object from a new viewpoint.

mentioned, both of these approaches are actually volumetric cloud models, but use texture-mapped imposters for speed in rendering.

Particle systems (Reeves 1983) are commonly used to simulate volumetric gases, such as smoke, with very convincing results and provide easy animation control. The difficulty with using particle systems for cloud modeling is the massive number of particles that are necessary to simulate realistic clouds.

Several authors have used the idea of volume-rendered implicit functions for volumetric cloud modeling (Bloomenthal et al. 1997). Nishita has used volume-rendered implicits as a basic cloud model in his work on multiple scattering illumination models; however, this work has concentrated on illumination effects and not on realistic modeling of the cloud geometry (Nishita, Nakamae, and Dobashi 1996). Stam has also used volumetric blobbies to create his models of smoke and clouds (Stam and Fiume 1991, 1993, 1995). I have used volumetric implicits combined with particle systems and procedural detail to simulate the formation and geometry of volumetric clouds (Ebert 1997; Ebert and Bedwell 1998). This approach uses implicits to provide a natural way of specifying and animating the global structure of the cloud, while using more traditional procedural techniques to model the detailed structure. The implicits are controlled by a modified particle system that incorporates simple simulations of cloud formation dynamics, as described later in this chapter.

A VOLUMETRIC CLOUD MODELING SYSTEM

In developing this new cloud modeling and animation system, I have chosen to build upon the recent work in advanced modeling techniques and volumetric procedural modeling. As mentioned in Chapter 1, many advanced geometric modeling techniques, such as fractals (Peitgen, Jürgens, and Saupe 1992), implicit surfaces (Blinn 1982b; Wyvill, McPheeters, and Wyvill 1986; Nishimura et al. 1985), grammar-based modeling (Smith 1984; Prusinkiewicz and Lindenmayer 1990), and volumetric procedural models/hypertextures (Perlin 1985; Ebert, Musgrave, et al. 1994) use procedural abstraction of detail to allow the designer to control and animate objects at a high level. Their inherent procedural nature provides flexibility, data amplification, abstraction of detail, and ease of parametric control. When modeling complex volumetric phenomena, such as clouds, this abstraction of detail and data amplification are necessary to make the modeling and animation tractable. It would be impractical for an animator to specify and control the detailed three-dimensional density of a cloud model. This system does not use a physics-based approach because it is computationally prohibitive and nonintuitive to use for many animators and modelers. Setting and animating correct physics parameters for dew point, particulate distributions, temperature and pressure gradients, and so forth is a time-consuming, detailed task. This model was developed to allow the modeler and animator to work at a much higher level and doesn't limit the animator by the laws of physics.

Volumetric procedural models have all of the advantages of procedural techniques and are a natural choice for cloud modeling because they are the most flexible, advanced modeling technique. Since a procedure is evaluated to determine the object's density, any advanced modeling technique, simple physics simulation, mathematical function, or artistic algorithm can be included in the model.

The volumetric cloud model uses a two-level model: the cloud macrostructure and the cloud microstructure. Implicit functions and turbulent volume densities model these, respectively. The basic structure of the cloud model combines these two components to determine the final density of the cloud.

Procedural `turbulence` and `noise` functions create the cloud's microstructure, in a manner similar to the `basic_gas` function (see Chapter 7). This allows the procedural simulation of natural detail to the level needed. Simple mathematical functions are added to allow shaping of the density distributions and control over the sharpness of the density falloff.

Implicit functions were chosen to model the cloud macrostructure because of their ease of specification and smoothly blending density distributions. The user simply specifies the location, type, and weight of the implicit primitives to create the overall cloud shape. Any implicit primitive, including spheres, cylinders, ellipsoids, and skeletal implicits, can be used to model the cloud macrostructure. Since these are volume rendered as a semitransparent medium, the whole volumetric field function is being rendered. In contrast, implicit surface modeling only uses a small range of values of the field to create the objects. The implicit density functions are primitive-based density functions: they are defined by summed, weighted, parameterized, primitive implicit surfaces. A simple example of the implicit formulation of a sphere centered at the point *center* with radius *r* is the following:

$$F(x,y,z):(x - \text{center}.x)^2 + (y - \text{center}.y)^2 + (z - \text{center}.z)^2 - r^2 = 0$$

The real power of implicit functions is the smooth blending of the density fields from separate primitive sources. I use Wyvill's standard cubic function (Wyvill, McPheeters, and Wyvill 1986) as the density (blending) function for the implicit primitives:

$$F_{\text{cub}}(r) = -\frac{4}{9}\frac{r^6}{R^6} + \frac{17}{9}\frac{r^4}{R^4} - \frac{22}{9}\frac{r^2}{R^2} + 1$$

In the preceding equation, r is the distance from the primitive. This density function is a cubic in the distance squared, and its value ranges from 1 when $r = 0$ (within the primitive) to 0 at $r = R$. This density function has several advantages. First, its value drops off quickly to zero (at the distance R), reducing the number of primitives that must be considered in creating the final surface. Second, it has zero derivatives at $r = 0$ and $r = R$ and is symmetrical about the contour value 0.5, providing for smooth blends between primitives. The final implicit density value is then the weighted sum of the density field values of each primitive:

$$\text{Density}_{\text{implicit}}(p) = \sum_i \left(w_i F_{\text{cub}_i}(p - q)\right)$$

where w_i is the weight of the ith primitive and q is the closest point on element i from p.

To create nonsolid implicit primitives, the location of the point is procedurally altered before the evaluation of the blending functions. This alteration can be the product of the procedure and the implicit function and/or a warping of the implicit space.

These techniques are combined into a simple cloud model:

```
volumetric_procedural_implicit_function(pnt, blend%, pixel_size)
  perturbed_point = procedurally alter pnt
                    using noise and turbulence
  density1 = implicit_function(perturbed_point)
  density2 = turbulence(pnt, pixel_size)
  blend = blend% * density1 +(1 - blend%) * density2
  density = shape resulting density based on user controls for
            wispiness and denseness (e.g., use pow and
            exponential function)
  return(density)
```

The density from the implicit primitives is combined with a pure turbulence-based density using a user-specified `blend%` (60% to 80% gives good results). The blending of the two densities allows the creation of clouds that range from entirely determined by the implicit function density to entirely determined by the procedural turbulence function. When the clouds are completely determined by the implicit functions, they will tend to look more like cotton balls. The addition of the procedural alteration and turbulence is what gives them their naturalistic look.

VOLUMETRIC CLOUD RENDERING

This chapter focuses on the use of these techniques for modeling and animating realistic clouds. The volume rendering of the clouds is not discussed in detail. For a description of the volume-rendering system that was used to make my images of clouds in this book, please see Chapter 7. Any volume-rendering system can be used with these volumetric cloud procedures; however, to get realistic effects, the system should accumulate densities using atmospheric attenuation, and a physics-based illumination algorithm should be used. For accurate images of cumulus clouds, a high-albedo illumination algorithm (e.g., Max 1994; Nishita, Nakamae, and Dobashi 1996) is needed. Chapter 10 also shows how these techniques can be rendered at interactive rates using 3D texture mapping hardware.

Cumulus Cloud Models

Cumulus clouds are very common in nature and can be easily simulated using spherical or elliptical implicit primitives. Figure 9.2 shows the type of result that can be achieved by using nine implicit spheres to model a cumulus cloud. The animator/modeler simply positions the implicit spheres to produce the general cloud structure. Procedural modification then alters the density distribution to create the detailed wisps. The algorithm used to create the clouds in Figures 9.2 and 9.3 is the following:

```
void cumulus(xyz_td pnt, float *density, float *parms, xyz_td
             pnt_w, vol_td vol)
{
  float new_turbulence();     // my turbulence function
  float peachey_noise();      // Darwyn Peachey's noise function
  float metaball_evaluate();  // function for evaluating
                              // meta-ball primitives
  float mdens,                // meta-ball density value
       turb, turb_amount      // turbulence amount
       peach;                 // Peachey noise value
  xyz_td path;                // path for swirling the point
  extern int frame_num;
  static int ncalcd = 1;
  static float sin_theta_cloud, cos_theta_cloud, theta,
        path_x, path_y, path_z, scalar_x, scalar_y, scalar_z;
```

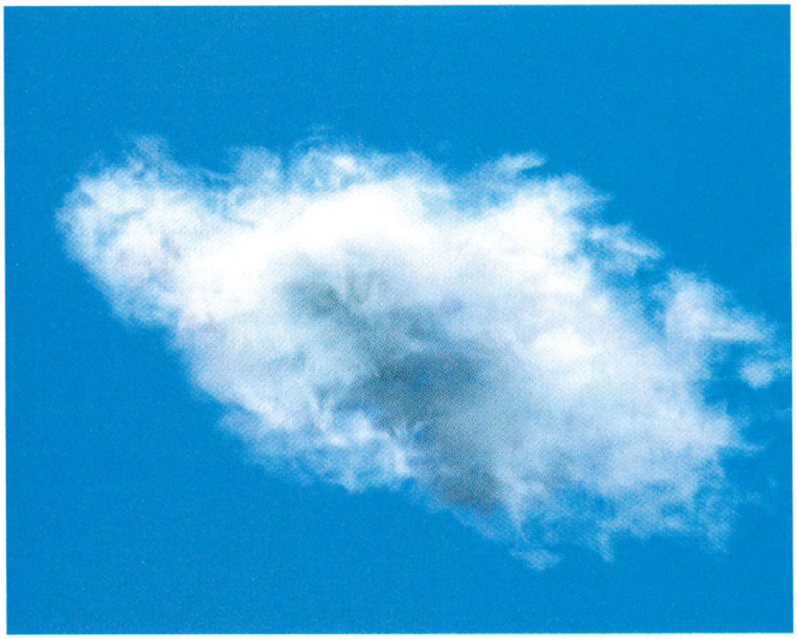


FIGURE 9.2 An example cumulus cloud. Copyright © 1997 David S. Ebert.

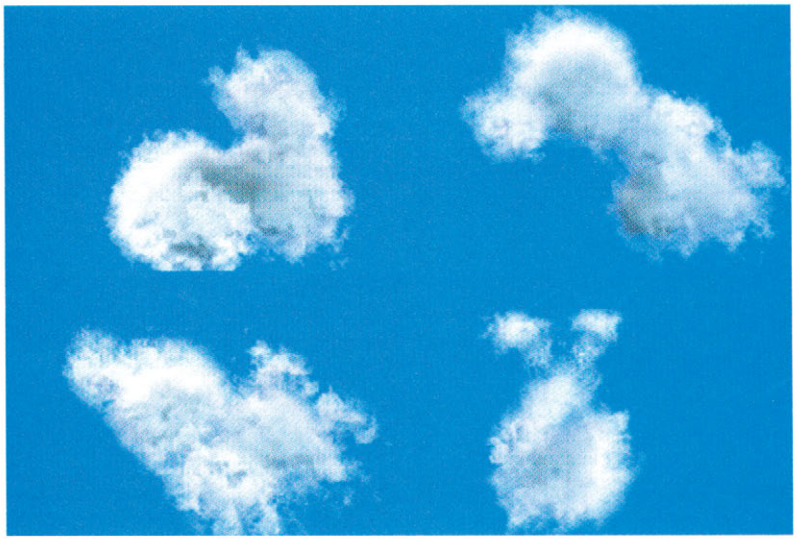


FIGURE 9.3 Example cloud creatures. Copyright © 1998 David S. Ebert.

```
  // calculate values that only depend on the frame number
  // once per frame
  if(ncalcd)
    {
     ncalcd = 0;
     // create gentle swirling in the cloud
    theta = (frame_num%600)*.01047196; // swirling effect
    cos_theta_cloud = cos(theta);
    sin_theta_cloud = sin(theta);
    path_x = sin_theta_cloud*.005*frame_num;
    path_y = .01215*(float)frame_num;
    path_z = sin_theta_cloud*.0035*frame_num;
    scalar_x = ( . 5+(float)frame_num*0.010);
    scalar_z = (float)frame_num*.0073;
    }
  // Add some noise to the point's location
  peach = peachey_noise(pnt); // Use Darwyn Peachey's noise
  pnt.x -= path_x - peach*scalar_x;
  pnt.y = pnt.y - path_y +.5*peach;
  pnt.z += path_z - peach*scalar_z;
  // Perturb the location of the point before evaluating the
  // implicit primitives.
  turb = fast_turbulence(pnt);
  turb_amount = parms[4]*turb;
  pnt_w.x += turb_amount;
  pnt_w.y -= turb_amount;
  pnt_w.z += turb_amount;
  mdens = (float)metaball_evaluate((double)pnt_w.x,
          (double)pnt_w.y, (double)pnt_w.z, (vol.metaball));
  *density = parms[1]*(parms[3]*mdens + (1.0 -
                     parms[3])*turb*mdens);
  *density = pow(*density,(double)parms[2]);
}
```

`Parms[3]` is the blending function value between implicit (meta-ball) density and the product of the turbulence density and the implicit density. This method of blending ensures that the entire cloud density is a product of the implicit field values, preventing cloud pieces from occurring outside the defining primitives. Using a large `parms[3]` generates clouds that are mainly defined by their implicit primitives and are, therefore, "smoother" and less turbulent. `Parms[1]` is a density scaling factor, `parms[2]` is the exponent for the `pow( )` function, and `parms[4]` controls the amount of turbulence to use in displacing the point before evaluation of the implicit primitives. For good images of cumulus clouds, useful values are the following: 0.2 < `parms[1]` < 0.4, `parms[2]` = 0.5, `parms[3]` = 0.4, and `parms[4]` = 0.7.

Cirrus and Stratus Clouds

Cirrus clouds differ greatly from cumulus clouds in their density, thickness, and falloff. In general, cirrus clouds are thinner, less dense, and wispier. These effects can be created by altering the parameters to the `cumulus` cloud procedure and also by changing the implicit primitives. The density value parameter for a cirrus cloud is normally chosen as a smaller value and the exponent is chosen larger, producing larger areas of no clouds and a greater number of individual clouds. To create cirrus clouds, the user can simply specify the global shape (envelope) of the clouds with a few implicit primitives or specify implicit primitives to determine the location and shape of each cloud. In the former case, the shape of each cloud is mainly controlled by the volumetric procedural function and turbulence simulation, as opposed to cumulus clouds where the implicit functions are the main shape control. It is also useful to modulate the densities along the direction of the jet stream to produce more natural wisps. For instance, the user can specify a predominant direction of wind flow and use a turbulent version of this vector to control the densities as follows:

```
void Cirrus(xyz_td pnt, float *density, float *parms, xyz_td
           pnt_w, vol_td vol, xyz_td jet_stream)
{
 float new_turbulence();     // my turbulence function
 float peachey_noise();      // Darwyn Peachey's noise function
 float metaball_evaluate();  // function for evaluating the
                             // meta-ball primitives
 float mdens,                // meta-ball density value
    turb,turb_amount         // turbulence amount
    peach;                   // Peachey noise value
    xyz_td path;             // path for swirling the point
 extern int frame_num;
 static int ncalcd = 1;
 static float sin_theta_cloud, cos_theta_cloud, theta,
      path_x, path_y, path_z, scalar_x, scalar_y, scalar_z;
 // calculate values that only depend on the frame number
 // once per frame
if(ncalcd)
   {
     ncalcd = 0;
     //create gentle swirling in the cloud
     theta = (frame_num%600)*.01047196; // swirling effect
     cos_theta_cloud = cos(theta);
     sin_theta_cloud = sin(theta);
     path_x = sin_theta_cloud*.005*frame_num;
```

```
      path_y = .01215*(float)frame_num;
      path_z = sin_theta_cloud*.0035*frame_num;
      scalar_x = (.5+(float)frame_num*0.010);
      scalar_z = (float)frame_num*.0073;
    }
  // Add some noise to the point's location
  peach = peachey_noise(pnt); // Use Darwyn Peachey's noise
  pnt.x -= path_x - peach*scalar_x;
  pnt.y = pnt.y - path_y +.5*peach;
  pnt.z += path_z - peach*scalar_z;
  // Perturb the location of the point before evaluating the
  // implicit primitives.
  turb = fast_turbulence(pnt);
  turb_amount = parms[4]*turb;
  pnt_w.x += turb_amount;
  pnt_w.y -= turb_amount;
  pnt_w.z += turb_amount; // make the jet stream turbulent
  jet_stream.x += .2*turb;
  jet_stream.y += .3*turb;
  jet_stream.z += .25*turb;
  // warp point along the jet stream vector
  pnt_w = warp(jet_stream, pnt_w);
  mdens = (float)metaball_evaluate((double)pnt_w.x,
          (double)pnt_w.y, (double)pnt_w.z, (vol.metaball));
  *density = parms[1]*(parms[3]*mdens + (1.0 - parms[3])*
                                        turb*mdens);
  *density = pow(*density,(double)parms[2]);
    }
```

Examples of cirrus cloud formations created using these techniques can be seen in Figures 9.4 and 9.5. Figure 9.5 shows a higher cirrostratus layer created by a large elliptical primitive and a few individual lower cirrus clouds created with cylindrical primitives.

Stratus clouds can also be modeled by using a few implicits to create the global shape or extent of the stratus layer, while using volumetric procedural functions to define the detailed structure of all of the clouds within this layer. Stratus cloud layers are normally thicker and less wispy, as compared with cirrus clouds. Adjusting the size of the turbulent space (smaller/fewer wisps), using a smaller exponent value (creates more of a cloud layer effect), and increasing the density of the cloud can create stratus clouds. Using simple mathematical functions to shape the densities can create some of the more interesting stratus effects, such as a mackerel sky. The mackerel stratus cloud layer in Figure 9.6 was created by modulating the densities with turbulent sine waves in the x and y directions.

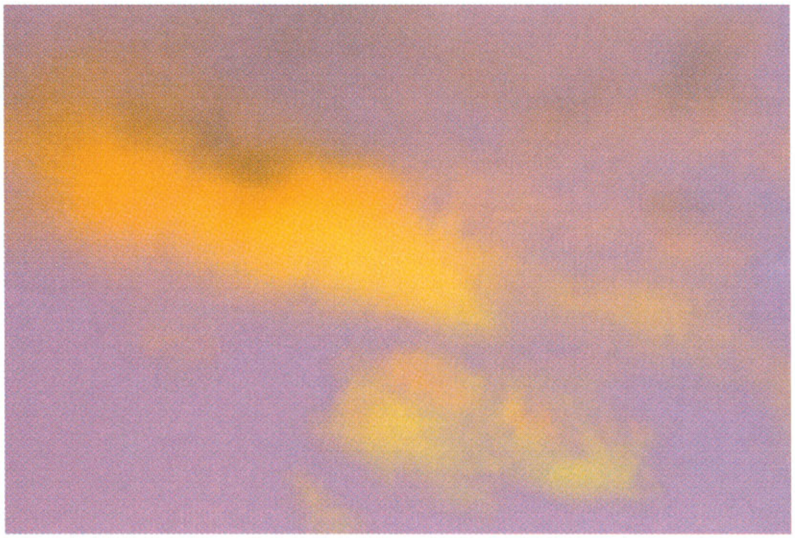


FIGURE 9.4 Cirrus clouds. Copyright © 1998 David S. Ebert.

FIGURE 9.5 Another example of cirrus and cirrostratus clouds. Copyright © 1998 David S. Ebert.

FIGURE 9.6 A mackerel stratus layer. Copyright © 1998 David S. Ebert.

Cloud Creatures

The combination of implicit functions with volumetric procedural models provides an easy-to-use system for creating realistic clouds, artistic clouds, and cloud creatures. Some examples of cloud creatures created using a simple graphical user interface (GUI) to position nine implicit spheres can be seen in Figure 9.3. They were designed in less than 15 minutes each, and a straw poll shows that viewers have seen many different objects in them (similar to real cloud shapes). Currently, the simple GUI only allows access to a small portion of the system. The rest of the controls are available through a text-based interface. Allowing the user access to more of the controls, implicit primitives, and parameters of the full cloud modeling system can create more complex shapes, time-based deformations, and animations. These cloud creatures are easily designed and animated by controlling the implicit primitives and procedural parameters. The implicit primitives blend and deform smoothly, allowing the specification and animation of skeletal structures, and provide an intuitive interface to modeling amorphous volumetric creatures.

User Specification and Control

Since the system uses implicit primitives for the cloud macrostructure, the user creates the general cloud structure by specifying the location, type, and weight of each implicit primitive. For the image in Figure 9.2, nine implicit spheres were positioned to create the cumulus cloud. Figure 9.3 shows the wide range of cloud shapes and creatures that can be created by simply adjusting the location of each primitive and the overall density of the model through a simple GUI. The use of implicit primitives makes this a much more natural interface than with traditional procedural techniques. Each of the cloud models in this chapter was created in less than 30 minutes of design time.

The user of the system also specifies a density scaling factor, a power exponent for the density distribution (controls amount of wispiness), any warping procedures to apply to the cloud, and the name of the volumetric procedural function so that special effects can be programmed into the system.

ANIMATING VOLUMETRIC PROCEDURAL CLOUDS

The volumetric cloud models described earlier produce nice still images of clouds and also clouds that gently evolve over time. The models can be animated using the procedural animation techniques in Chapter 8 or by animating the implicit primitives. Procedural animation is the most flexible and powerful technique since any deformation, warp, or physical simulation can be added to the procedure. An animator using key frames or dynamics simulations can animate the implicit primitives. Several examples of applying these two animation techniques for various effects are described below.

Procedural Animation

Both the implicit primitives and the procedural cloud space can be animated algorithmically. One of the most useful forms of implicit primitive animation is warping. A time-varying warp function can be used to gradually warp the shape of the cloud over time to simulate the formation of clouds, their movement, and their deformation by wind and other forces. Cloud formations are usually altered based on the jet stream. To simulate this effect, all that is needed is to warp the primitives along a vector representing the jet stream. This can be done by warping the points before evaluating the implicit functions. The amount of warping can be controlled by the wind velocity or gradually added in over time to simulate the initial cloud development. Implicits can be warped along the jet stream as follows:

```
perturb_pnt = procedurally alter pnt using noise and turbulence
height = relative height of perturb_pnt
vector = jet_stream + turbulence(pnt)
perturb_pnt = warp(perturb_pnt, vector, height)
density1 = implicit_function(perturbed_pnt)
. . .
```

To get more natural effects, it is useful to alter each point by a small amount of turbulence before warping it. Several frames from an animation of a cumulus cloud warping along the jet stream can be seen in Figure 9.7. To create this effect, ease-in and ease-out based on the frame number were used to animate the warp amount. The implicit primitives' locations do not move in this animation, but the warping function animates the space to move and distort the cloud along the jet stream vector. Other warping functions to simulate squash and stretch (Bloomenthal et al. 1997) and other effects can also be used. Instead of a single vector and velocity, a vector field is input into the program to define more complex weather patterns. The current system allows the specification of vector flow tables and tables of functional primitives (attractors, vortices) to control the motion and deformation of the clouds. Stam (1995) used this procedural warping technique successfully in animating gases.

Implicit Primitive Animation

The implicit primitives can be animated in the same manner as implicit surfaces: each primitive's location, parameters (e.g., radii), and weight can be animated over time. This provides an easy-to-use, high-level animation interface for cloud animation. This technique was used in the animation "A Cloud Is Born" (Ebert et al. 1997), showing the birth of a cumulus cloud followed by a flythrough of it. Several stills from the formation sequence can be seen in Figure 9.8. For this animation, the centers of the implicit spheres were moved over time to simulate three separate cloud elements merging and growing into a full cumulus cloud. The radii of the spheres were also increased over time. Finally, to create animation in the detailed cloud

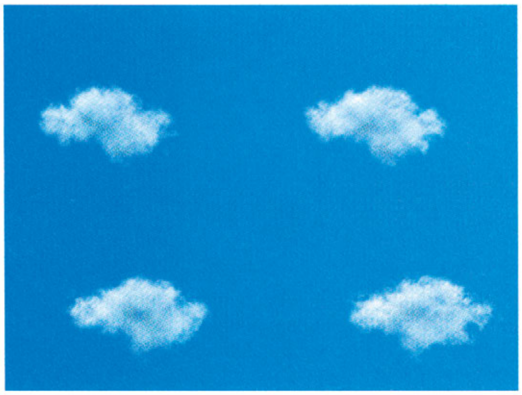


FIGURE 9.7 Cloud warping along the jet stream. Copyright © 1998 David S. Ebert.

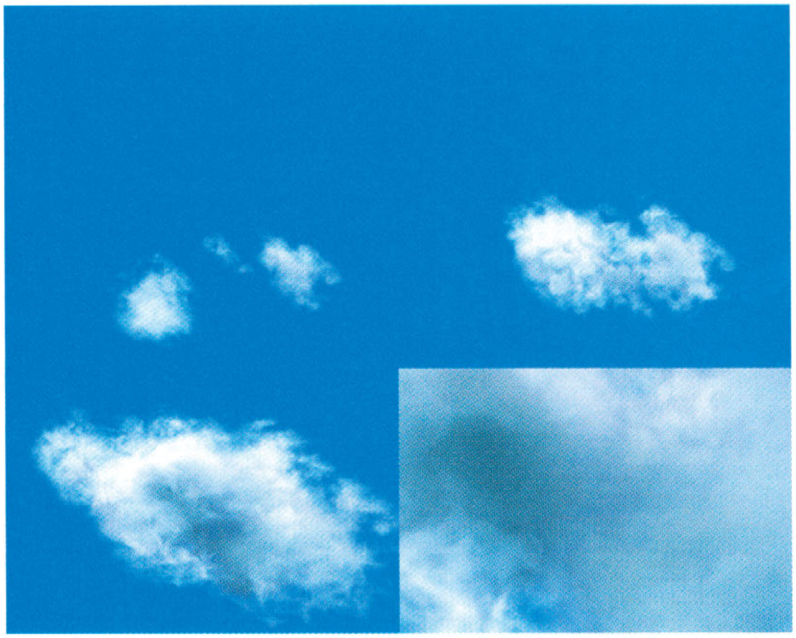


FIGURE 9.8 Several stills from "A Cloud Is Born" showing the formation of the cloud. Copyright © 1998 David S. Ebert.

structure, each point was moved along a turbulent path over time before evaluation of the `turbulence` function, as illustrated in the `cumulus` procedure.

A powerful animation tool for volumetric procedural implicit functions is the use of dynamics and physics-based simulations to control the movement of the implicits and the deformation of space. Since the implicits are modeling the macrostructure of the cloud while procedural techniques are modeling the microstructure, fewer primitives are needed to achieve complex cloud models. Dynamics simulations can be applied to the clouds by using particle system techniques, with each particle representing a volumetric implicit primitive. The smooth blending and procedurally generated detail allow complex results with less than a few hundred primitives, a factor of 100–1000 less than needed with traditional particle systems. I have implemented a simple particle system for volumetric procedural implicit particles. The user specifies a few initial implicit primitives and dynamics information, such as speed, initial velocity, force function, and lifetime, and the system generates the location, number, size, and type of implicit for each frame. In our initial tests, it took less than one minute to generate and animate the implicit particles for 200 frames.

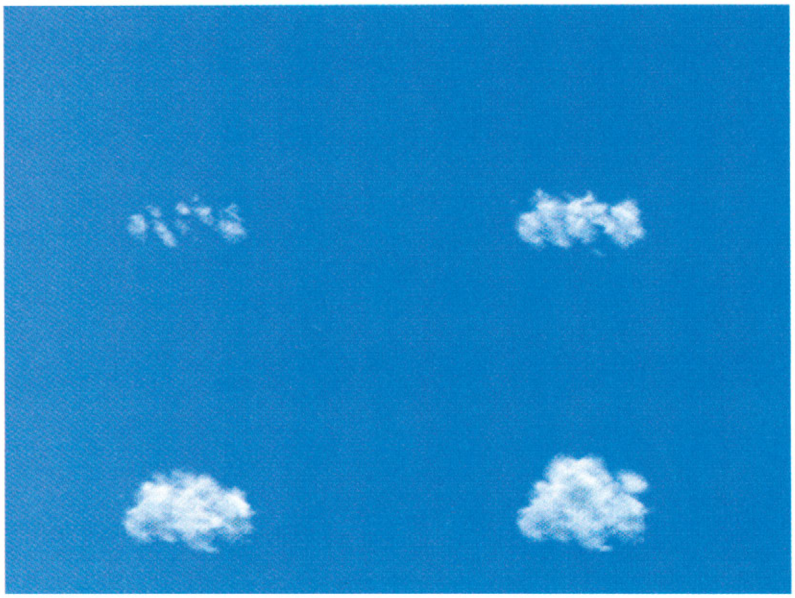


FIGURE 9.9 Volumetric procedural implicit particle system formation of a cumulus cloud. Copyright © 1998 David S. Ebert.

Unlike traditional particle systems, cloud implicit particles never die—they just become dormant.

Cumulus clouds created through this volumetric procedural implicit particle system can be seen in Figure 9.9. The stills in Figure 9.9 show a cloud created by an upward turbulent force. The number of children created from a particle was also controlled by the turbulence of the particle's location. For the animations in this figure, the initial number of implicit primitives was 12, and the final number was approximately 50.

The animation and formation of cirrus and stratus clouds can also be controlled by the use of a volumetric procedural implicit particle system. For the formation of a large area of cirrus or cirrostratus clouds, the particle system can randomly seed space and then use turbulence to grow the clouds from the creation of new implicit primitives, as can be seen in Figure 9.10. The cirrostratus layer in this image contains 150 implicit primitives that were generated from the user specifying 5 seed primitives.

To control the dynamics of the cloud particle system, any commercial particle animation program can also be used. A useful approach for cloud dynamics is to use

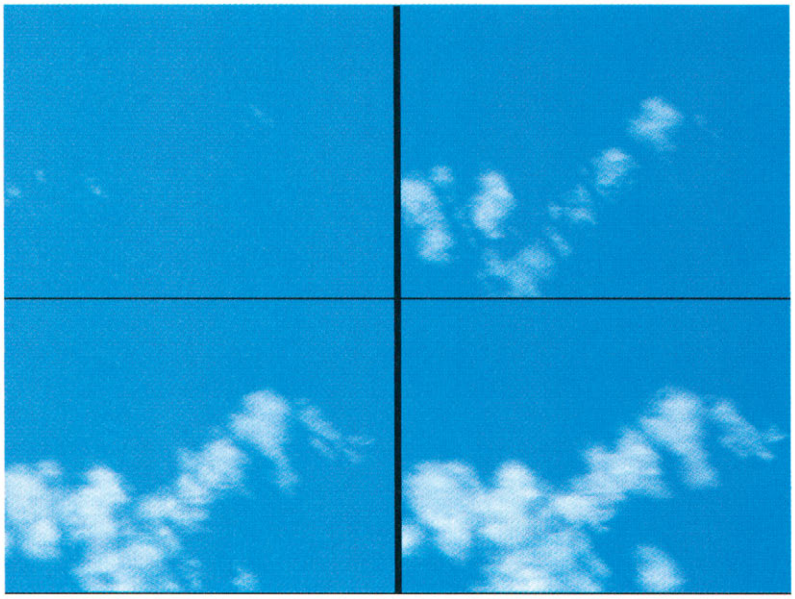


FIGURE 9.10 Formation of a cirrostratus cloud layer using volumetric procedural implicit particles. Copyright © 1998 David S. Ebert.

qualitative dynamics—simple simulations of the observed properties and formation of clouds. The underlying physical forces that create a wide range of cloud formations are extremely complex to simulate, computationally expensive, and very restrictive. The incorporation of simple, parameterized rules that simulate observable cloud behavior will produce a powerful cloud animation system.

INTERACTIVITY AND CLOUDS

Creating models of clouds that can be rendered and animated at interactive rates is an exciting challenge, which is becoming tractable with the latest increases in graphics hardware speed and programmability.

Simple Interactive Cloud Models

There are several physical processes that govern the formation of clouds. Simple visual simulations of these techniques with particle systems can be used to produce more convincing cloud formation animations. As mentioned previously, Neyret

(1997) suggests that the following physical processes are important to cloud formation simulations: Rayleigh-Taylor instability, bubbles, temperature rate variation, Kelvin-Helmholtz instability, vortices, and Bernard cells. His model also takes into account phenomena at various scales, bubble generation, and cloud evolution.

Results from an implementation of these techniques by Ruchigartha using MEL scripts in Maya can be found at *www.ece.purdue.edu/~ebertd/ruchi1*. An example of the output from her system and the GUI cloud control can be seen in Figure 9.11.

Rendering Clouds in Commercial Packages

The main component needed to effectively render clouds is volume-rendering support in your renderer. Volumetric shadows, low- or high-albedo illumination, and correct atmospheric attenuation are needed to produce realistic-looking clouds. An example of a volume-rendering plug-in for Maya that can be used to create volume-rendered clouds can be found on the Maya Conductor CD from 1999 and on the HighEnd3D Web page, *www.highend3d.com/maya/plugins*. The plug-in, volumeGas, by Marlin Rowley and Vlad Korolov, implements a simplified version of my volume renderer (which was used to produce the images in these chapters). A resulting image from this plug-in can be seen in Figure 9.12.

CONCLUSION

The goal of these last three chapters has been to describe several techniques used to create realistic images and animations of gases and fluids, as well as provide insight into the development of these techniques. These chapters have shown a useful approach to modeling gases, as well as animation techniques for procedural modeling. To aid in reproducing and expanding the results presented here, all of the images are accompanied by detailed descriptions of the procedures used to create them. This gives you not only the opportunity to reproduce the results but also the challenge to expand upon the techniques presented. These chapters also provide insight into the procedural design approach that I use and will, hopefully, inspire you to explore and expand procedural modeling and animation techniques. The next chapter extends this work to discuss issues with real-time procedural techniques and hardware acceleration of these techniques.