17

QAEB RENDERING FOR PROCEDURAL MODELS

F. KENTON MUSGRAVE

INTRODUCTION

This chapter and the next present some pretty technical discussions. In the previous three chapters, I have tried to keep the discussion at a level where the technically minded artist might want to follow along. Now we're plunging into stuff that is probably only of interest to mathematically minded programmers. So a word of warning: You might just want to read the introductions, which are written at a fairly conversational level, or maybe even skip this chapter and the next entirely. These chapters were written originally as technical papers, so most of the verbiage is in the dry and dense idiom of scientific writing.

In the previous chapter we developed some procedural functions designed to serve as realistic terrain models. When interpreted as height fields, these terrain models are pretty good. But, as we pointed out, height fields are usually precomputed and stored at a fixed *post spacing*. That is, the function is sampled at points on a regular grid, as at the points where lines intersect on graph paper. This has three undesirable side effects: First, we have to store the height field data in files that can become rather large (although this is less an issue as memory and disk space become ever cheaper). Second, the discrete values in the height field must be interpolated, usually linearly, leading to unnatural artifacts such as a terrain composed of triangles—try finding *that* in nature! Third, and the most serious problem in my view, we have a fixed level of detail given by the post spacing. If we get too close to our terrain model, it becomes locally flat and boring. If we view it at too great a distance, we will get aliasing, due to the triangles being smaller than the Nyquist limit, or we have to resort to adaptive level of detail (LOD) schemes.

As demonstrated in earlier chapters, we can build terrain functions that are both continuous (i.e., with a well-defined value everywhere across the surface, not just at certain predetermined sample points) and band-limited so that the features in the model can be kept at or near the Nyquist limit, whatever that limit may be locally.

Thus we can generate models that have ideal appearance everywhere, even though such a model must be view dependent. This is because the appropriate level of detail at any given point is a function of its distance from the view point, due to the perspective projection, as well as the synthetic camera's field of view and resolution. What we require is a rendering algorithm that can take advantage of the flexibility of the procedural approach.

A few years ago I was teaching a class for which the first edition of this book was the text. To illustrate to the class just how simply the procedural approach can generate piles and piles of visual detail, I designed such an algorithm. Again, the goal was maximal simplicity in the algorithm, period. Accordingly, I expected it to be *really* slow. It came as a considerable surprise when it turned out to be only *very* slow, not glacial. (That is, it took on the order of a minute to create an image, when I was expecting days.) I gave this algorithm the wonderfully turgid name *quasi-analytic error-bounded ray tracing*, or *QAEB tracing* for short. To balance the scales of pretense, I pronounce the acronym QAEB whimsically "kweeb" (to rhyme with "dweeb," of course).

QAEB tracing was originally applied to height fields. A discussion with John Hart led me to the realization that, aside from the speedup scheme described later that applies only to height fields, QAEB tracing is actually a general rendering scheme for point-evaluated *implicit* models. Implicit models are isosurfaces of fields, for example, surfaces where some function F defined over three-space is equal to zero. (We'll see examples of exactly this in our cloud models toward the end of this chapter.) So QAEB tracing is actually a pretty powerful rendering method—slow but *really* simple. I should point out that QAEB tracing is simply raymarching but with a variable step size and an implied use of band-limiting in the procedural model to provide adaptive level of detail. But the pertinent point is this: it's simple to implement and it makes cool pictures.

Without further ado, let's launch into the pithy text of the technical paper on QAEB tracing, replete with the turgid use of the royal "we."

QAEB TRACING

We present a numerical method called *QAEB tracing* for ray-tracing procedural height field functions as displacement maps. The method accommodates continuous adaptive level of detail in the height field. Spatial error in ray-surface intersections is bounded by an error specified in screen space. We describe a speedup scheme general to incremental height field ray-tracing methods and a method for rendering

hypertextures, for example, clouds. The QAEB algorithm is simple and surprisingly fast.

One capability distinguishing scanline rendering from ray tracing is the capability of rendering *displacement maps* (Cook 1984). We describe a method for ray tracing a subclass of displacement maps, *height fields*. Height field rendering is important for visualizing terrain data sets, as in various defense-related simulation applications. Adaptive level of detail is desirable in such renderings, and this new method accommodates that simply. Quasi-analytic error-bounded ray tracing is a method for rendering general implicit functions, that is, continuous functions of n variables $F : R^n \rightarrow R$. We will show applications for height fields, where $n = 2$, and hypertextures, where $n = 3$. We call it "quasi-analytic" because it yields a numerical approximation of an analytic ray-surface intersection. This approximation is bounded by an error specified in screen space. When F is a procedural fractal function, the world space frequency content of the terrain model may be linked to its projected screen space Nyquist limit, to accommodate adaptive level of detail without aliasing. QAEB-traced images can be superior, due to their adaptive level of detail and the nonpolygonal character of the rendered height field primitive. As the QAEB algorithm is isolated in the ray-surface intersection routine, QAEB-traced objects amount to new primitives that may be added to the standard inventory of ray-traced geometric primitives such as spheres and polygons. Development of the QAEB algorithm was originally motivated by the desire to render landscapes with adaptive level of detail. We thus describe the algorithm in the context of rendering height fields.

A striking aspect of the QAEB algorithm is its simplicity. It was expected to be slow, due to its profligate character. That is, it requires a very large number of evaluations of the implicit function. In practice it is surprisingly fast, probably due to its simplicity: the entire code can fit in cache on a contemporary microprocessor, yielding near-optimal performance. A desirable feature is that spatial precision (and error) is linked directly to the spatial sampling rate. No greater precision is calculated than is needed, potentially saving CPU cycles and time.

PROBLEM STATEMENT

Generally, the only moving part in a terrain animation is the camera. Freedom of movement of the camera with perspective projection requires adaptive LOD in the terrain model. LOD can be readily accommodated with procedural fractal terrain models. Fractals are scaling (Mandelbrot 1982), thus detail is potentially unlimited.

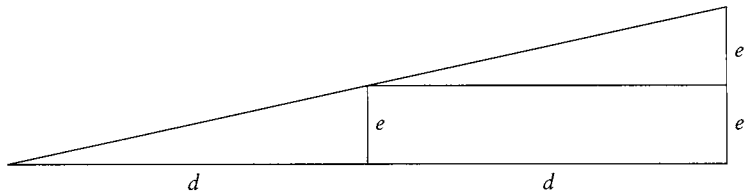


FIGURE 17.1 Feature span e varies linearly with distance d: at base length d, altitude is e; at base length $2d$, altitude is $2e$. For isosceles triangles, reflect about the base.

Procedural fractals are inherently band-limited and can have parameterized band limits, as shown in previous chapters. An argument based on similar triangles (see Figure 17.1) shows that, in a perspective projection, feature span (not area) in screen space varies linearly with distance. We can use this knowledge to roll off high frequencies in the model to keep them at or below the Nyquist limit of the projected basis function (e.g., the Perlin *noise* function) in screen space. Octaves in the spectral construction of the fractal may be related to distance d as

$$octaves = -\log_2(\tan(fov/res)) - \log_2(d) - 3.5$$

where *fov* is the lateral field of view, *res* is the number of samples across that field, and 3.5 is a constant relating our chosen Perlin *noise* basis function to the Nyquist frequency.

PRIOR ART

Procedural ray tracing of synthetic fractal terrains with adaptive level of detail has been addressed by Kajiya (1983a, 1983b) and Bouville (1985). These methods are based on polygon subdivision fractal terrains. They require data structure overhead, for example, quadtrees, to organize the numerous polygon primitives. Polygon-based methods can suffer artifacts due to discontinuities in levels of detail. This gives rise to the nontrivial problem of detecting and sealing "cracks" between polygons of different sizes, where adaptive level of detail dictates a change in local polygon size. The QAEB approach is nonpolygonal and features continuous adaptive level of detail, and thus does not suffer from this complication. Procedural fractal functions accommodating continuous frequency change for such adaptive level of detail are described in the previous chapter.

The speedup scheme for rendering height fields was originally proposed by Anderson (1982) and Coquillart (1984). Perlin described a raymarching scheme (Perlin and Hoffert 1989) for hypertextures without LOD.

THE QAEB ALGORITHM

We now describe the QAEB method in the context of the simplest case, rendering height fields.

QAEB tracing is predicated on the following assumptions:

1. A user-specified error ε in the ray-surface intersection and surface normal is acceptable.
2. The function F being rendered is a height field, that is, a continuous function $F : R^2 \rightarrow R$ with a well-defined, globally invariant "up" direction $\vec{u}_F$.
3. F is a point-evaluated function, for example, a procedural texture, that can be efficiently evaluated at any point (x, y).
4. Near and far clipping planes are acceptable.

The algorithm is this: Starting at the near clipping plane, we march away from the view point in error-sized increments until we either exceed the far clipping plane or cross the height field surface, in which case we return an approximate intersection point and surface normal. The marching increment, or *stride* Δ, is exactly equal to ε in the virtual screen's world space dimensions, at the virtual screen's distance from the view point. The size of Δ varies linearly with distance, as shown in Figure 17.1. At each step we evaluate the height field function F and compare it against the ray's altitude relative to $\vec{u}_F$. (Note that this constitutes rendering an implicit function with the isosurface $F(x,y)$ = surface elevation.) This is profligate in evaluations of F, leading to two conclusions: the algorithm is expected to be slow, and F should be as efficient as possible.

C code implementing the QAEB algorithm is presented in Appendix A.

ERROR IN THE ALGORITHM

The error is defined to be the difference between the analytic intersection of the ray and F, and the intersection determined by the QAEB approximation. The error ε is specified in screen space in terms of sample spacing, for example, one pixel. Both the

stride and the error vary in world space proportional to εd, where d is the distance from the view point. The value of ε actually specifies three somewhat independent errors along three axes: the vertical, lateral, and depth axes of the (horizontal) screen. The lateral error is exactly ε. The vertical error (the error associated with vertical perturbation of the sampling ray) is indeterminate, due to the chances of hitting or missing a ridge profile. Hitting or missing the top of a ridge can result in vastly different intersection points in world space; this problem is intractable. The depth error is more interesting. To ensure that depth error corresponds to ε, it should be proportional to the slope F' of F. For F with F' discontinuous at local maxima and maximum slope $(F_{max})'_{max}$ at such maxima, the stride Δ should vary as $1/(F_{max})'_{max}$ to ensure meeting the specified error.[1] If F yields low, smooth terrain, Δ may be increased. Note that *fractional Brownian motion* (fBm), upon which most synthetic terrain models are based, is self-affine (Voss 1988). That is, for fBm, local slope can increase as higher spatial frequencies are added. Thus the correct stride may change with the dictates of LOD on terrain frequency content. (Note that these matters notwithstanding, we have never found it necessary to use anything other than the simple stride length εd in our applications for image synthesis of nonpathological models—in other words, anything we ever wanted to render nicely.[2])

NEAR AND FAR CLIPPING PLANES

Stride Δ varies linearly with distance d from the view point. It follows that at distance $d = 0$, $\Delta = 0$. Therefore, raymarching must begin at a near clipping distance $d_0 > 0$. Features closer to the view point than d_0 will not be visible. Conversely, a greater value of d_0 implies a larger initial stride. Thus the value of d_0 can significantly impact rendering time.

---

1. This statement is actually incorrect; the truth is more subtle. The depth error, and thus the stride, should be linked to the minimum slope occurring at local maxima of F. That is, if local maxima may be very narrow (corresponding to sharp peaks or ridges in the terrain), the stride must be small enough to capture them within the specified error. However, if maxima of F may be steep on one side only, the error corresponds to the minimum $(F_{max})'_{min}$ of the two slopes on either side of the local maximum, as $1/(F_{max})'_{min}$.

2. Manuel Gamito has used interval arithmetic to ensure analytic ray-surface intersections in the QAEB scheme and ray-domain distortion methods (Barr 1986) to render nonheight field terrains as seen in Figure 17.2. Werner Benger (*www.photon.at/ ~werner/Light.html*) has developed more approximate marching methods to speed up the QAEB rendering process, while sacrificing accuracy in the ray-surface intersection.

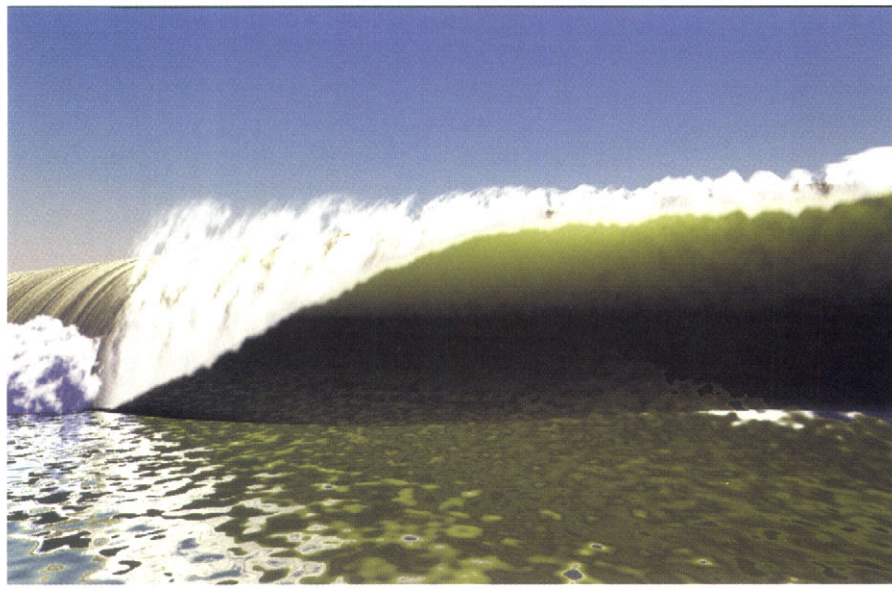


FIGURE 17.2 A QAEB-traced non-height-field model of a wave, modeled and rendered by Manuel Gamito. The spray and foam are QAEB hypertextures.

A far clipping plane[3] $d_f < \infty$ is necessary to terminate computation. Beyond d_f the model is not visible. Smaller values of d_f imply a shorter raymarch. Choosing the distances for the clipping planes involves the kind of trade-off between time and realism typical in computer graphics.

CALCULATING THE INTERSECTION POINT AND SURFACE NORMAL

Ray-surface intersection is indicated when the ray altitude crosses adjacent evaluations z_{i-1} and z_i of F. As these evaluation points are exactly ε apart, either can serve as the approximate intersection point, both values being guaranteed to be within ε. Alternatively, we may use the intersection of the ray and the line between z_{i-1} and z_i. This may yield a slightly more accurate solution at a cost of a few more operations—a cost that will generally be overwhelmed by the cost of the march to the intersection.

3. We refer to the near and far clipping distances as "clipping planes," although they are implemented as distances from the view point and are thus more like "clipping spheres."

The surface normal is constructed via the cross product of two vectors between three samples of F. The first vector $\vec{v}_d$ is from z_{i-1} to z_i. The second vector $\vec{v}_l$ is from z_{i-1} to a third sample z_l of F taken at a distance equal to the current stride Δ_i in a lateral direction, that is, perpendicular to the plane containing the ray and $\vec{u}_F$. The normalized cross product $\|\vec{v}_d \times \vec{v}_l\|$ serves as our surface normal approximation.

C code implementing this scheme appears in Appendix B.

ANTIALIASING

Antialiasing may be accomplished with ordinary supersampling methods. Supersampling implies sampling at a higher spatial frequency; this in turn implies a smaller error ε. In adaptive supersampling, this may require that the samples that indicate supersampling be recomputed with the implied smaller ε to ensure meaningful results. Uniform supersampling simply implies a smaller ε throughout, as ε should generally be equal to the screen space distance between adjacent samples.

To the extent that F represents an uncorrelated random function (all reasonable terrain models being in fact highly correlated), the model is self-jittering. That is, samples automatically have a stochastic character, even if equidistant in screen space. This is a property inherent in the random fractal model, not the QAEB rendering method.

A SPEEDUP SCHEME FOR HEIGHT FIELDS

Assuming that (1) the scene is rendered bottom to top relative to $\vec{u}_F$ and (2) $\vec{u}_F$ coincides with the screen "up" vector $\vec{u}_s$, we may employ a simple optimization to great benefit. We keep an array A_d of depth values of size equal to the number of lateral samples in the screen. A_d is initialized to d_0 and updated to the distance d_i from the view point of the last intersection in the corresponding column. Subsequent rays in the same column, but higher relative to $\vec{u}_F$, may begin marching at d_i rather than d_0. This speeds up rendering enormously, particularly for horizontal views of great depth. The second assumption may be dispensed with by indexing A_d along an axis $\vec{v}_F$ horizontal relative to $\vec{u}_F$ and in the plane of the virtual screen. The span w_a of the array indices along $\vec{v}_F$ is equal to the extent of the projection of the rotated screen onto $\vec{v}_F$ (see Figure 17.3). The number of buckets in A_d is $\lceil w_A/\varepsilon \rceil$. A given screen sample is projected onto $\vec{v}_F$ to determine its bucket in A_d. This bucket is within the specified error ε, laterally. The projection of a point p on the screen onto $\vec{v}_F$ is

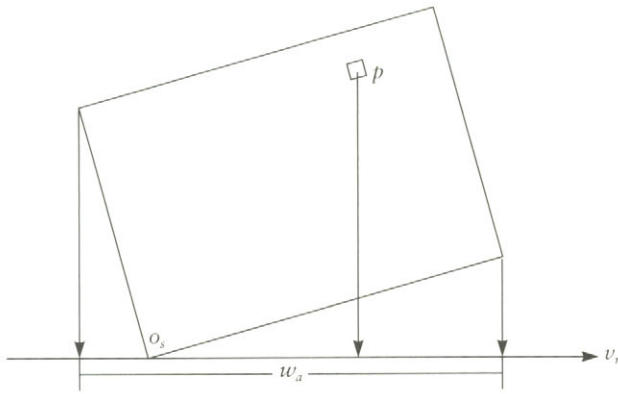


FIGURE 17.3 Projection of rotated screen and sample onto horizontal axis.

$\vec{v}_F\lceil(p-o_S)\bullet\vec{v}_F\rceil$, where o_s is the lower-left corner of the screen relative to $\vec{u}_F$ and $\vec{v}_F$. This speedup scheme is complicated in cases where the screen contains the point where $\vec{u}_F$ and the view direction are parallel. Such cases may be dealt with by keeping two depth arrays, one for each of the two marching directions opposite relative to $\vec{u}_F$.

SHADOWS, REFLECTION, AND REFRACTION

In rendering with adaptive level of detail, correct shadows depend on correct projected shadow feature size. In the QAEB scheme with a given ε, that size is equal to the stride Δ_i at the intersection point where the shadow ray is spawned. Features must retain this size in the shadow projection. For a light source at infinity, the feature size does not vary with the projection. Thus the shadow ray stride is constant and equal to Δ_i. For a light source not at infinity, for example, a point light source, feature size changes linearly with distance, as shown earlier. It follows that shadow rays start with stride proportional to Δ_i, and the stride goes to zero as the distance to the light source d_l goes to zero. Stride then varies with Δ_i and d_l as $\Delta_i d_l/(d_l + 1)$.

Reflection and refraction cannot be handled correctly in the QAEB scheme, as the divergence of adjacent rays is affected arbitrarily in such (specular) transport, and our assumptions about the geometric error break down. This may not be significant, as this scheme was developed to render fractal models of natural phenomena, such as landscapes and clouds, that usually feature neither type of specular

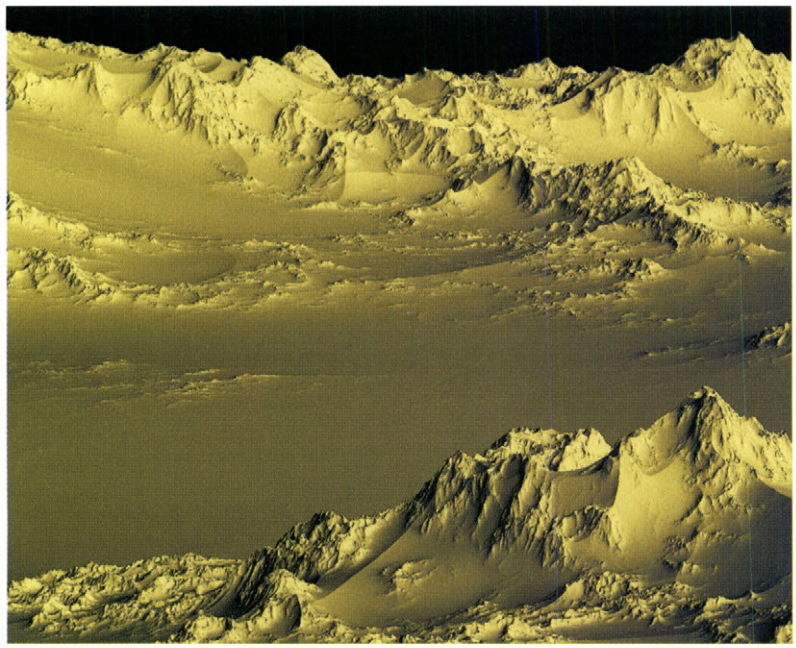


FIGURE 17.4 The very first QAEB-traced terrain. The terrain model is the "ridged multifractal" function described in Chapter 16. Copyright © F. Kenton Musgrave.

transport. Experience indicates that reflective and refractive stochastic surfaces generally create visually confusing images, at any rate.

PERFORMANCE

The QAEB-traced terrain in Figure 17.4 was rendered in 2 minutes, 41 seconds on a 150 MHz R4400, at 640 × 480 (NTSC video) resolution. Near and far clipping planes are at 0.01 and 100.0, respectively. Figure 17.5 is at the same resolution, with shadows from a directional light source (i.e., a light source at infinity) at an elevation of ~30° above horizontal. Rendering time was 17 minutes, 57 seconds. Figure 17.6 was rendered at a film recorder resolution of 2000 × 1333, without shadows and with atmospherics, in 62 minutes. Near and far clipping planes are at 0.4 and 100.0, respectively. All images were rendered at one ray per pixel with an ε of one pixel.

This performance is far better than was expected before testing. Expectations were low because of the profligate evaluations of the procedural height field

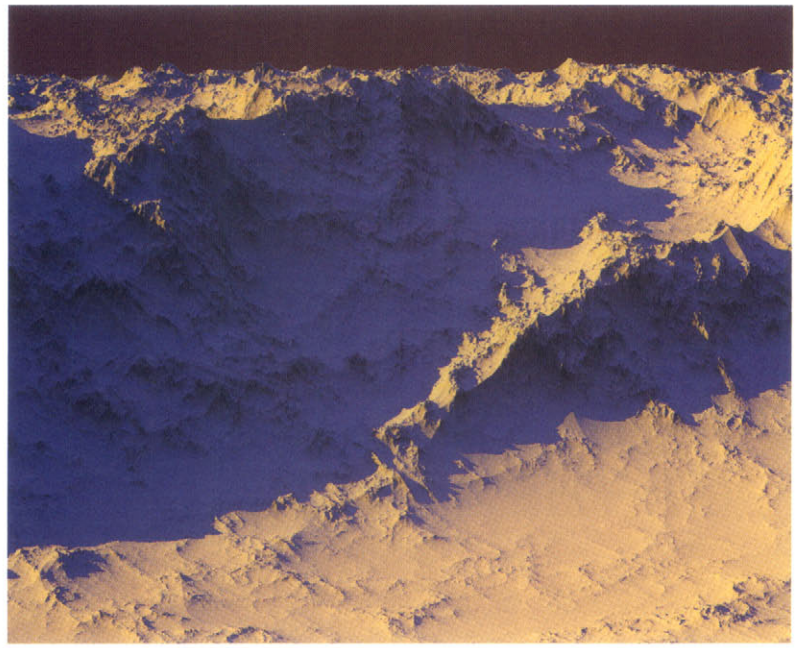


FIGURE 17.5 The first QAEB rendering demonstrating shadows from a light source at infinity. Copyright © F. Kenton Musgrave.

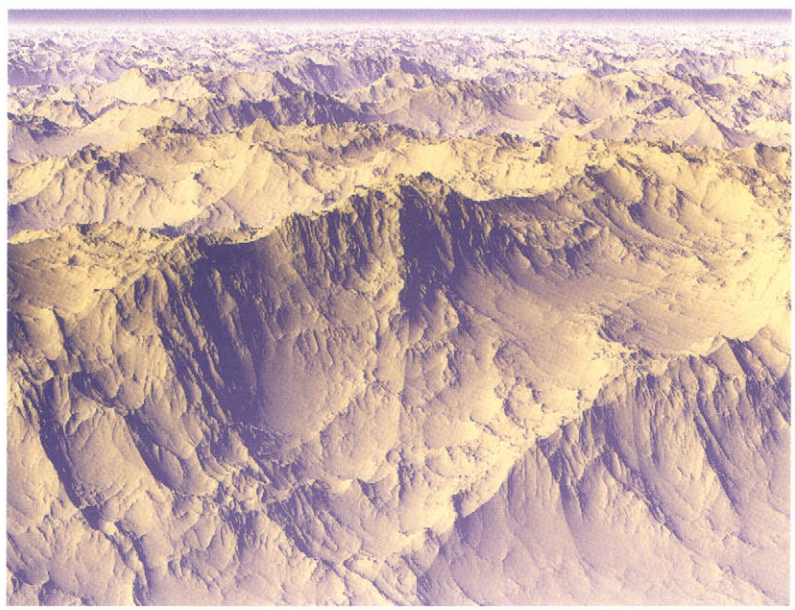


FIGURE 17.6 High resolution and great depth, with atmosphere.

function, which uses on the order of 10^3 floating-point operations per evaluation. QAEB tracing time is dominated by evaluations of the Perlin *noise* function in our implementation. Rendering time is directly impacted by the computational complexity of F, the size of the stride, the screen resolution, the number of screen samples, the distances to the near and far clipping planes, the angle of the view direction relative to $\vec{u}_F$, the use of shadows, and, in that case, the number of light sources and their angles to $\vec{u}_F$ and the slope considerations at local maxima described earlier.

QAEB-TRACED HYPERTEXTURES

The QAEB scheme is readily applicable, without the previous speedup scheme, to volume rendering of procedural hypertextures (see Figure 17.7).

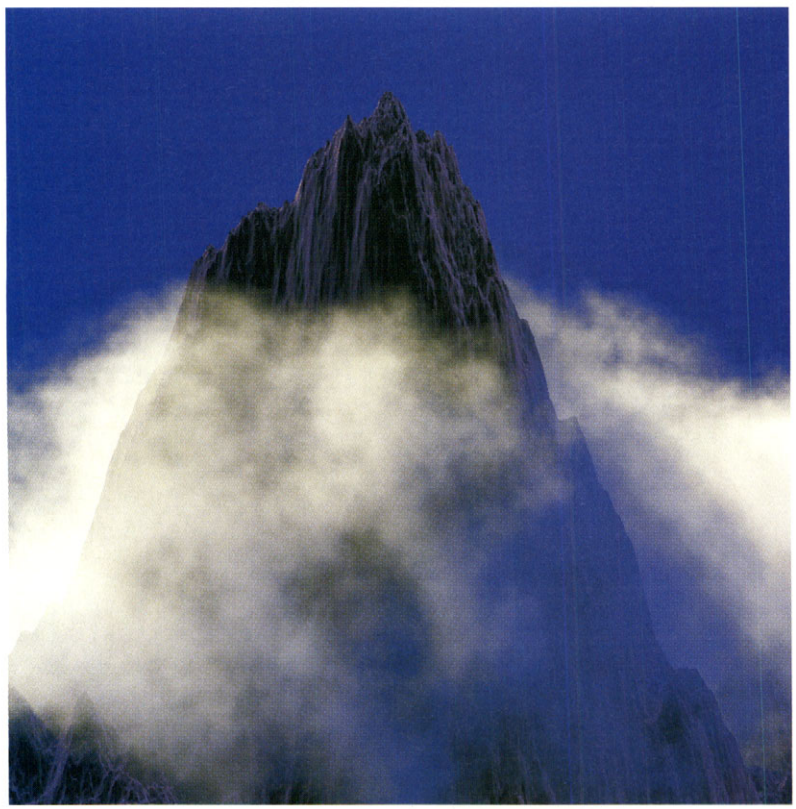


FIGURE 17.7 QAEB-traced scene with volumetric shadowing.

Clouds

Rather than checking ray height versus F along the raymarch, you can interpret values of $F > 0$ as density, with values of $F < 0$ being zero density or clear air. This corresponds to raymarching a *hypertexture* (see Chapter 12). Accumulating the density according to Beer's law (see Chapter 18) and computing lighting with a self-occluding, single-scattering model can yield nice results.[4] Applying a realistic high-albedo, anisotropic multiple-scattering illumination model (Max 1994) yields substantially better results, as seen in Figures 17.8 and 17.9.[5]

Our hypertexture cloud model has two parts: a procedural fractal function, usually either vanilla fBm or Perlin's "turbulence" (fBm constructed using the absolute value of Perlin *noise* as the basis), and a simple weighting function that modulates the fractal, making the fractal cloud a more or less distinct blob situated in clear air. The weighting function is necessary to ensure that fractal clouds do not completely permeate all of space, as the fractal functions are statistically homogeneous. This weighting function can be as simple as $F = 1.0 - d$, where d is the distance from a central point, or something more complex to shape the cloud, as seen in Figure 17.9 and Figure 17.8(a), where the cloud bottom is rather flat. (The shaping function for these clouds is included in the C code on this book's Web site, *www.mkp.com/tm3*.)

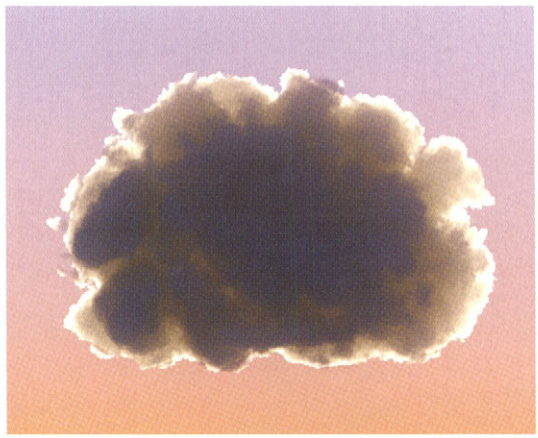
In the simplest single-scattering illumination model, local illumination is attenuated by accumulating density along a shadow ray shot toward the light source. Naively, one such ray must be sent per sample, accounting for most of the cost in the rendering. In practice, the frequency of such illumination samples can be decreased, and their stride length may be increased, up to the point of objectionable artifacts. (For details on how, see the code on this book's Web site.) Storing precomputed illumination values in a voxel grid can speed rendering at the cost of increased memory use.

Jittering samples within the interval of the stride (Cook 1984) also allows greater stride length in primary and shadow rays, with graceful degradation in image quality, as seen in Figure 17.8(c). Nonjittered samples will lead to conspicuous quantization artifacts at large stride lengths, as illustrated in Figure 17.8(b).

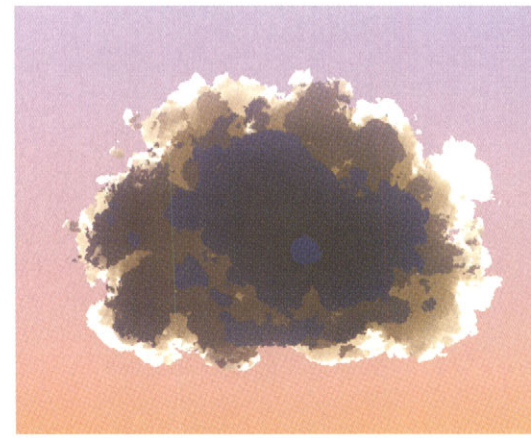
---

4. Note that in this application, QAEB tracing is a slower but simpler and more accurate equivalent of the gas rendering methods in Chapter 9.

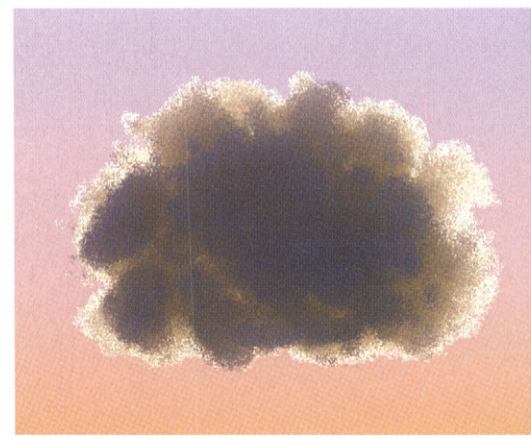
5. Unfortunately, treatment of such physical illumination models are beyond the scope of this book. Their development is the current Ph.D. research topic of my student Sang Yoon Lee at the time of this writing. For more on his work, see *http://student.seas.gwu.edu/~sylee/*.



(a)



(b)



(c)

FIGURE 17.8 A QAEB-traced cloud: (a) rendered with a stride of two pixels; (b) quantization artifacts from long stride with regular sampling; (c) same long stride, with jittered sampling. Copyright © F. Kenton Musgrvae.

Simple QAEB-traced fBm clouds with single-scattering illumination were used to create the Mickey Mouse and Goofy characters and a rather detailed cruise ship model in a television commercial for Disney cruise lines produced during my tenure at Digital Domain. Only spherical cloud primitives were used, along with animation of the hypertexture spaces and densities. These simple models yielded some striking and novel animation effects. For a preliminary cloud character animation test from this project, see *www.kenmusgrave.com/animations.html.*

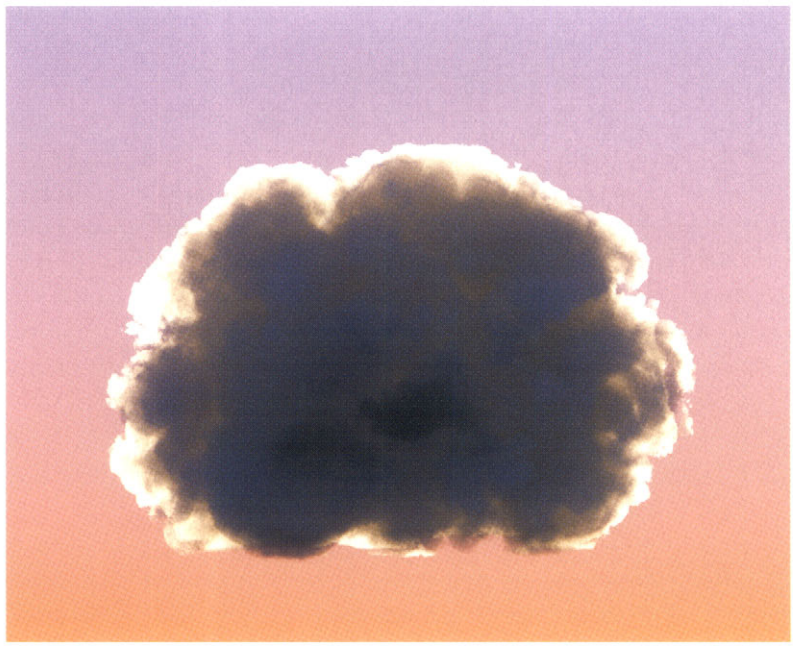


FIGURE 17.9. A QAEB-traced hypertexture cloud, with a radiosity solution for high-albedo anisotropic multiple scattering, by Sang Yoon Lee. The radiosity model is by Nelson Max.

BILLOWING CLOUDS, PYROCLASTIC FLOWS, AND FIREBALLS

An ontogenetic model of billowing can yield realistic dynamic models of fireballs, billowing smoke, and growing cumulus clouds. The distinctive behavior we call "billowing" results from rapid advection in a fluid medium, causing turbulent flow in three dimensions. Accurate modeling of such turbulent flow is a problem of notorious computational difficulty, which only recently yielded to solutions practical for the field of image synthesis (Fedkiw 2001). For entertainment applications, empirical accuracy is not the goal; visual verisimilitude is, and visual novelty might prove even more useful.

The film *Dante's Peak* required effects simulating the fast-moving, highly destructive cloud of hot volcanic ash known as a pyroclastic flow. At the time, I had already developed the realistic hypertexture cloud model just described. That model was originally designed to serve as a testbed for the difficult problem of modeling the anisotropic, high-albedo, multiple-scattering illumination required for truly realistic

rendering of clouds, as seen in Figure 17.9. That cloud model can be extended to model billowing in a relatively simple way.

Our first attempt consisted of a swarm of cloudlets animated in the Alias Dynamation particle system package. This approach yielded good results when the animation consisted simply of sweeping the swarm of cloudlet fields through a static texture space. "Popping" of high frequencies is inevitable in such a scheme because the higher-frequency details change faster than those of lower frequency as the cloud front advances through texture space. The result was an explosive quality in the advancing cloud, which seemed appropriate. However, the director, citing footage of real volcanic clouds, requested a dynamic, billowing quality wherein the cloud appears to turn inside out as it evolves along its forward direction. This leads to the solution presented here.

A single cloudlet can be made to appear to billow by scaling the domain of the fractal function, relative to the "forward" direction of the billowing. The scaling is of the angle a given sample makes with that axis with time (see Figure 17.10). The result of this domain distortion is that, as the sample point is rotated toward the forward direction in magnitude proportional to the angle, cloud features appear to rotate toward a singularity opposite the "forward" direction.

Features get stretched longitudinally with time in this scheme. To ameliorate objectionable artifacts arising from this, the distorted, "old" texture is rolled off with time and replaced with a "young," undistorted texture that is in turn rolled off as it ages. This proceeds cyclically, yielding a cyclic model. This cyclic nature is disguised, visually, by growth in size of the cloudlet with time. For details on how this is accomplished, see the code on this book's Web site (*www.mkp.com/tm3*).

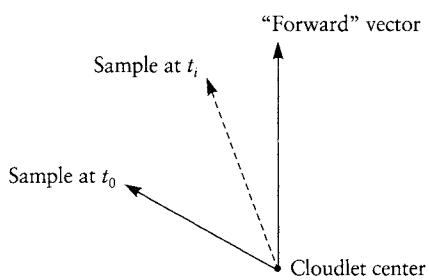


FIGURE 17.10 Rotation of samples with time in the billowing scheme.

Fireballs

Fireballs may be simulated by animating a color map that varies from white through yellow, orange, and red to black, as in the "flame" shader in Chapter 15. This map is indexed by radius (hotter toward the center) and time (cooling to black with age). The map color is used as the cloud texture color. Fireballs are more efficient to render than clouds because they are self-luminous and require no illumination or shadow calculations. Preliminary models of such CG pyrotechnics were developed for a bid for certain special effects scenes in the film *Air Force One*.[6]

Psychedelic Clouds

As noted in Chapter 15, a vector-valued fBm can be interpreted as an RGB color vector and used to color a cloud to get a fantastical coloration, as seen in Figure 17.11. The animation *Comet Leary*[7] was accomplished with this model, by sweeping the hypertexture through five spherical weighting fields of successively decreasing density. Though not a convincing simulation of turbulent flow, it is a simple model yielding a visually appealing effect. The GIT scheme described in Chapter 15 could be used to obtain greater control over stochastic coloring.

CONCLUSION

We have demonstrated a numerical algorithm for ray-tracing implicit procedural functions with adaptive level of detail. It is simple, surprisingly fast, and general to all continuous functions $F : R^n \rightarrow R$. It features a screen space geometric error bounded by a user-specified value. The described speedup scheme, which is important to good performance in rendering height fields, is general to all incremental height field ray-tracing schemes. The stride we employ is the most conservative possible. It may be possible to use more sophisticated methods, such as Lipschitz conditions (Standler and Hart 1994; Worley and Hart 1996) to speed rendering by extending the average stride length. The demonstrated speed of this algorithm is surprising. This speed is conjectured to be linked to the algorithm's simplicity, which allows it to reside entirely in cache memory for near-optimum microprocessor performance.

---

6. For an early animation test, see *www.kenmusgrave.com/animations.html*.
7. Also available at *www.kenmusgrave.com/animations.html*.

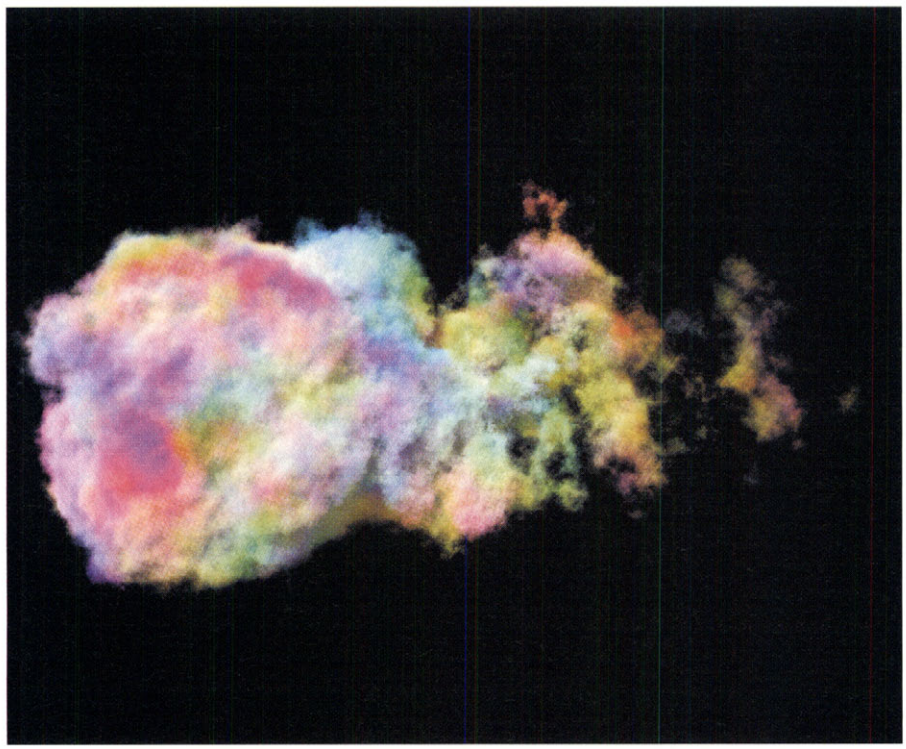


FIGURE 17.11 *Comet Leary* is a whimsical rendering of Timothy's final return to Earth. It is a QAEB cloud with coloring by a vector-valued fBm. Copyright © F. Kenton Musgrave.

We have adapted the method to render measured height field data sets to add adaptive level of detail. For this application, we simply constrain the fractal function F by the measured data to render such data sets with added stochastic detail. The lowest frequency in the added fractal detail should be close to that of the post spacing in the measured data set.

We have also extended the method to procedural hypertextures, with a corresponding increase in rendering time. The results have been cloud models of unprecedented realism, some promising synthetic pyrotechnic effects, and some gratuitous psychedelia.