The Synthesis and Rendering of Eroded Fractal Terrains

F. Kenton Musgrave†, Craig E. Kolb† and Robert S. Mace‡

†Yale University Department of Mathematics
Box 2155 Yale Station
New Haven, Connecticut 06520

‡Silicon Graphics, Inc.
2011 N. Shoreline Boulevard
Mountain View, California 94039

Abstract

In standard fractal terrain models based on fractional Brownian motion the statistical character of the surface is, by design, the same everywhere. A new approach to the synthesis of fractal terrain height fields is presented which, in contrast to previous techniques, features locally independent control of the frequencies composing the surface, and thus local control of fractal dimension and other statistical characteristics. The new technique, termed *noise synthesis*, is intermediate in difficulty of implementation, between simple stochastic subdivision and Fourier filtering or generalized stochastic subdivision, and does not suffer the drawbacks of creases or periodicity. Varying the local crossover scale of fractal character or the fractal dimension with altitude or other functions yields more realistic first approximations to eroded landscapes. A simple physical erosion model is then suggested which simulates hydraulic and thermal erosion processes to create global stream/valley networks and talus slopes. Finally, an efficient ray tracing algorithm for general height fields, of which most fractal terrains are a subset, is presented.

CR Categories and Subject Descriptors: I.3.3 [**Computer Graphics**]: Picture/Image Generation, I.3.7 [**Computer Graphics**]: Three-Dimensional Graphics and Realism

General Terms: Algorithms, Graphics

Additional Keywords and Phrases: Fractal, terrain models, stochastic subdivision, fractional Brownian motion, fractal dimension, lacunarity, crossover scale, erosion models, height fields, ray tracing.

1. INTRODUCTION

At first glance, fractal terrains are convincing forgeries of natural mountain terrains. Closer scrutiny, however, reveals an unnatural character in these surfaces as representations of nature. One problem is the fact that if one turns the average "fractal mountain" upside-down and looks at the other side of the surface, it looks (statistically) identical to the "top" side. This is almost never the case in nature, where depressions in the landscape fill up with all manner of detritus, thus acquiring smoother surfaces over the ages of geologic time.

The origin of fractal landscapes in computer graphics is this: some time ago, B. B. Mandelbrot perceived an analogy between a record of Brownian motion over time, and the skyline of jagged mountain peaks.[15] He reasoned that if this process were extended to two dimensions the resulting "Brownian surface" might provide a visual approximation to mountains in nature. He then found it necessary to generalize from Brownian motion to a fractional Brownian surface. Some of Mandelbrot's earliest computer graphics images were of such surfaces.[14] Voss later used Mandelbrot's fractional Brownian surfaces to create some very convincing forgeries of nature.[28] New terrain synthesis methods have since been proposed by Fournier et al.,[4] Miller,[19] and Lewis.[12]

Most fractal terrain surfaces are related to fractional Brownian motion (*fBm*), and can be called more loosely $1/f^{\beta}$ surfaces. Fractal terrains in general have no global erosion features inherently due to isotropy and stationarity, and practically due to the difficulty in implementation and computation of such global processes, which require global communication.

In this paper, we present a flexible approach to the generation of fractal terrain models of varying smoothness and asymmetry in a first-pass surface specification stage, and suggest a second-pass global, physical erosion process for height fields which generates both local and global erosion features through a simplified simulation of natural erosion processes. The terrain generation method features arbitrary local control of *fractal dimension* and *crossover scale*, neither of which was sought in previous methods. It also features arbitrary *lacunarity**, which is not available in common subdivision algorithms. Terrain patches can be compactly represented as height fields; we will also describe a very efficient algorithm for ray tracing regular** height fields. This rendering algorithm can be characterized as the definition of height fields as a new type of primitive (thus joining spheres, planes, polygons, etc.) in the ray tracing paradigm.

2. OTHER WORK

Creating synthetic images of fractal terrains usually involves two distinct procedures: *modelling* and *rendering*. It is well-known that rendering fractal terrains is generally more time-consuming than modelling them, especially if one chooses to ray trace the scene. In our scheme for imaging eroded synthetic terrains, we subdivide modelling into two steps: *terrain generation* and *erosion simulation*. As the latter step is an attempt to simulate the actions of the elements on the landscape over geologic time, it is not surprising to find that it competes with rendering, in the time required to create realistic results.

---

* *Crossover scale* and *lacunarity* will be defined in section 3.1.

** We define a *regular height field* as a two-dimensional array of altitude values where the distance is constant between all rows and between all columns of altitude values.

As all three steps are essential to creating realistic images, we present new results in each area.

First let us review work by other researchers in the fields of fractal terrain synthesis, erosion modelling, and ray tracing of terrain models.

2.1. Fractal Terrain Modelling

Most fractal terrain models have been based on one of five approaches: Poisson faulting,[15, 28] Fourier filtering,[15, 28, 18] midpoint displacement,[4, 12, 19, 25] successive random additions,[28] and summing band-limited noises.[7, 8, 19] The approach presented here is of the last type, which we will refer to as the *noise synthesis* method. We will now briefly review these techniques (for a more detailed review, see Voss and Saupe[25] or Musgrave, Kolb, and Mandelbrot[21]).

The original terrain generation technique employed by Mandelbrot[15] was generation of fBm using Poisson faulting. The Poisson faulting technique involves applying Gaussian random displacements (faults, or step functions) to a plane or sphere at Poisson distributed intervals. The net result is a Brownian surface. This approach has been employed by Mandelbrot to create fractal coastlines and Voss to create fractal planets.[15] It has the advantage of being suitable for use on spheres for creation of planetoids. Its primary drawback is the $O(n^3)$ time complexity of the algorithm.

Midpoint displacement methods are standard in fractal geometry, and were introduced as a fast terrain generation technique by Fournier, Fussell, and Carpenter.[4] We classify the various midpoint displacement techniques by locality of reference: *wireframe* midpoint displacement, *tile* midpoint displacement, *generalized stochastic subdivision*, and *unnested** subdivision. Wireframe subdivision is used only in the well-known triangle subdivision scheme[4] and involves the interpolation between two points in the subdivision process. Tile midpoint displacement involves the interpolation of three or more non-collinear points; it is used in the "diamond-square" scheme of Miller,[19] the square scheme of Fournier et al.,[4] and the hexagon subdivision of Mandelbrot and Musgrave.[25] Generalized stochastic subdivision[12] interpolates several local points, constrained by an autocorrelation function. Miller [19] also proposed an unnested "square-square" subdivision, which is not strictly a *midpoint* subdivision scheme.

Wireframe and tile midpoint displacement methods are generally efficient and easy to implement, but have fixed lacunarity and are nonstationary due to nesting (for illustrations of the resulting artifacts, see Miller[19]). Generalized stochastic subdivision and unnested subdivision schemes are stationary; the former is flexible but not particularly easy to implement, while the latter features fixed lacunarity and is very simple to implement. Note that all midpoint displacement techniques produce true fractal surfaces[23] but simply have an incorrect statistical character to qualify as fractional Brownian motion.[16]

Successive random additions is a flexible unnested subdivision scheme. If points determined in previous stages of subdivision are re-used, they are first displaced by addition of a random variable with an appropriate distribution. Previous points need not be re-used; new grid points to be displaced can be determined from the previous level of subdivision by linear or nonlinear interpolation. Successive random additions features continuously variable level of detail, which is useful for zooms in animation, and arbitrary lacunarity. The lacunarity λ depends on the change of resolution at successive generations; time complexity of the algorithm is a function of λ and the final resolution R. The successive random additions algorithm is easy to implement.

Fourier filtering generates fBm by taking the Fourier transform of a two-dimensional Gaussian *white noise*, multiplying it in frequency space with an appropriate filter, and interpreting the inverse Fourier transform of the product as a height field.[28] Alternatively, one can simply choose the coefficients of the discrete Fourier transform, subject to the proper constraints, and interpret the inverse Fourier transform as above.[25] Advantages of this approach include the availability of arbitrary lacunarity and precise control of global frequency content. Disadvantages include periodicity of the final surface, which can require that substantial portions of the computed height field patch be discarded, the $O(n\log n)$ time complexity of the FFT algorithm, the level of complexity of implementation, and lack of local control of detail.

What we call *noise synthesis* can be described as the iterative addition of signals with tightly band-limited frequencies, each of which has a randomly varying, or *noisy*, amplitude. Noise synthetic surfaces have been used by Miller,[19] Gardner[7] and Saupe.[8] Miller has used Perlin's procedural "$1/f$ noise"[20] (which is actually $1/f^3$ noise) as a *displacement map*[3] to add detail to the (otherwise straight) edges of polygons tessellating a Brownian surface of similar spectral content. Gardner has introduced a noise function, based on a "poor man's Fourier series"[6] (a variation of the Mandelbrot-Weierstrass function) and interpreted it as a height field. The quantization of altitude values of the height field yields terraced land, such as mesas. Our approach differs from Gardner's in that we exercise local control over frequency content based on the amplitude of existing signal and other functions. The Perlin noise function is notably more isotropic than Gardner's noise function, and is not periodic; Gardner's terrains and textures suffer visible artifacts due to these factors. In addition, the use of Gardner's noise function requires that one subjectively determine critical values for a number of constants. Driven by table lookups, the Gardner noise function is much faster than the Perlin function.

Noise synthesis is a *functional-based modelling* technique. Each point is determined procedurally, independently of its neighbors; no global computation is required. Point-evaluation is a distinguishing property of the noise synthesis method for generating random fractals.

Recently, a parallel and independent research effort by Saupe has developed an approach to noise synthesis similar to that presented here. Saupe's work features an emphasis on mathematical foundations, while the authors' emphasizes applications. For a thorough mathematical treatment of the issues of noise synthesis which is complimentary to this work, see Saupe.[8]

2.2. Erosion Models

The issue of the symmetry of fBm has been addressed by Mandelbrot and Voss[15, 28] through the use of non-linear scaling in a post-processing step, and by Mandelbrot[25] through the use of random variables with non-Gaussian distributions in the displacement process. These approaches yield peaks which are more jagged and valleys which are smoother, but still lack global erosion features. A global river system, created algorithmically at terrain-generation time, has been demonstrated by Mandelbrot and Musgrave,[25] with less-than-satisfactory results (see plate 10).

Kelley et al.[11] have used hydrology data to derive a system for the generation of *stream network* drainage patterns which are subsequently used to determine the topography of a terrain surface. This approach features, by its construction, the global dependence necessary for realistic hydraulic erosion patterns, and has a strong basis in measurements of real physical

* *Unnested* here means that successive levels of subdivision retain no points from previous levels.

systems. This approach to modelling hydraulic erosion is relatively efficient; what it lacks is the detail of a fractal surface. While the stream network may be fractal, the "surface under tension" used for the terrain surface is not, and cannot be readily made so without disturbing the drainage basins and stream paths.

A simple hydraulic erosion simulation is proposed here in which water is dropped on each vertex in a fractal height field and allowed to run off the landscape, eroding and depositing material at different locations as a function of the sediment load of water passing over each vertex. It features the global communication necessary to create global features, but is slow despite the $O(n)$ time complexity. We also present a global model for simulation of what we refer to as *thermal weathering*. While hydraulic erosion creates valleys and drainage networks, thermal weathering wears down steep slopes and creates talus slopes at their feet. The thermal weathering simulation can create realistic results in much less computing time than the hydraulic erosion simulation, and is also $O(n)$ in time complexity. Both models are discussed in section 4.

2.3. Ray Tracing Fractal Terrains

Efficient ray tracing of fractal terrains has been addressed by Kajiya,[9,10] Bouville,[2] Miller,[19] and Mastin et al.[18] Kajiya and Miller propose procedural fractal terrain models to save memory and achieve adaptive level of detail; Miller proposes a parallel scheme for rendering terrains which is not specific to ray tracing. Mastin et al. propose a quadtree spatial decomposition for the ray tracing of height fields.

The ray tracing schemes of Kajiya and Bouville rely on invariance of the horizontal position of computed vertices of the terrain height field under subsequent iteration of the midpoint subdivision process used to generate the surface, in order to ensure that the surface is within predictable bounds for ray/surface intersection test culling. This requires that the subdivision scheme *nests*, else the bounding volumes become ill-defined. (Note that nested subdivision schemes suffer most from the *creasing problem*.) The nesting requirement cannot always be met in iterative subdivision schemes, as when subdividing non-nesting polygons such as hexagons (see Mandelbrot[25]). Such limitations led the authors to develop an efficient ray tracing scheme which is not necessarily procedural but is general to all regular height fields. This new method can be implemented hierarchically as an n^2 tree, and in that is similar to the quadtree approach of Mastin et al. It uses a DDA to traverse a spatial subdivision data structure, and in that is reminiscent of the 3DDDA *ARTS* algorithm of Fujimoto.[5] Our "grid tracing"[22] scheme will be described in section 5.

3. TERRAIN SYNTHESIS

3.1. Fractal Dimension, Fractional Brownian Motion, Crossover Scale, and Lacunarity

We now give a very brief description of some of the mathematical terminology associated with the generation of fractal terrains. For greater depth, see Peitgen and Saupe.[25] For this discussion, we define D_f as the *fractal dimension* of the surface, D_E as the *Euclidean dimension* of the surface, and H as the *Holder exponent*. (Note that previous authors have sometimes erroneously referred to H as the fractal dimension.) For terrain models D_E=2 and D_f=3–H.

The fractal dimension D_f, Euclidean dimension D_E, Holder exponent H, and the *spectral exponent* β of $1/f^{\beta}$ noise and of fBm are related by

$$D_f = D_E+1-H = D_E+\frac{3-\beta}{2}. \qquad (1)$$

It follows that $\beta = 1+2H$ and $H = (\beta-1)/2$. Since $D_E = 2$, for our purposes, $D_f = 3-H = (7-\beta)/2$. H is constrained to the interval [0,1] and β to [1,3]; outside this range we do not have formally fractal behavior.

Fractional Brownian motion in one dimension is a *stochastic process* $X(t)$ with a *power spectrum** $S(f)$ scaling with f as

$$S(f) \sim 1/f^{\beta}$$

where β again is in the interval [1,3]. FBm is itself not a stationary process, but its increments $I(t, \Delta t) = X(t + \Delta t)-X(t)$ are; that is, the expected value of $I(t, \Delta t)$ is zero for all t and Δt and the variance σ^2 of $I(t, \Delta t)$ does not depend on t. In the special case of Brownian motion, $H = 0.5$ and σ^2 varies as Δt^{2H}. Thus for $H = 0.5$ increments are uncorrelated; for H>0.5 (as in fractal terrains, where H is approximately equal to 0.8) increments are positively correlated; for H<0.5 they are negatively correlated (corresponding to a *very* rough surface). In more than one dimension fBm is a *random field* $X(x,y,...)$ with X on any straight line being a $1/f^{\beta}$ noise.

Discretized fractional Brownian motion is a stochastic process $X(t)$ with a discrete power spectrum such that each spectral line has the energy

$$\int_{f_i}^{\lambda f_i} S(f)df$$

where λ is the logarithmic spacing of the lines. Many fBm surfaces used in computer graphics are discretized fBm's.

A fractal surface changes in character depending on whether it is observed from nearby or from far away. From far away it appears flat or smooth (as the Earth seen from space). The transition from "nearby" to "far away" appearances occurs at the *crossover scale* which is the scale where vertical and horizontal displacements are equal. Thus, for a mountain range rising within one kilometer from sea level to peaks which are one kilometer high, the crossover scale is one kilometer. The crossover scale is not to be confused with the upper and lower frequency cutoffs for a band-limited fBm.

Lacunarity generally refers to gaps in fractals;[15] in this instance it refers to the gap between frequencies composing the discretized fBm of the fractal terrain. Thus when iteratively adding the frequencies composing the discretized fBm, if the frequency f_i added at stage i is a multiple λ of f_{i-1},

$$f_i = \lambda f_{i-1},$$

λ is the spatial lacunarity of the fBm. While spatial lacunarity affects the texture of the fBm, this effect is usually only noticeable for $\lambda > 2$. Therefore we usually let $\lambda = 2$, as lower values involve more computation for a given frequency range of fBm and higher values effect the surface appearance.

3.2. Noise Function

Noise synthetic terrain generation is accomplished by the addition of successive frequencies of tightly band-limited noises. The source of the noise we use is a version of the Perlin[26] noise function. The ideal noise function for our purposes would be *monochromatic* (i.e., single-frequency), *stationary* (invariant under translation), and *isotropic* (invariant under rotation). The Perlin function supplies a band-limited signal of random amplitude variation. It is stationary and nearly isotropic.**

---

* The *power spectrum* $S(f)$ is the power $P(f)$ of the signal at frequency f, or the the mean squared power over interval Δf centered at f, divided by Δf: $S(f) = P(f)^2/\Delta f$.

** It is geometrically impossible to reconcile the three criteria of mono-

The Perlin noise function $N{:}R^n{\to}R$ is implemented as a set of random gradient values defined at integer points of a lattice or grid in space (of dimension n = 1, 2, or 3) which are interpolated by a cubic function. At lattice points in space (points in space with integer coordinates), the value of the function is zero (a *zero crossing*) and its rate of change is the gradient value associated with that lattice point. The first derivative of the function is interpolated at non-integer points using the cubic function $3x^2 - 2x^3$, which features second derivative continuity and zero rate of change at the end points, where x=0 and x=1. Since the gradient might be, for instance, increasing at two consecutive lattice points i and i+1, there may also be a zero crossing at a point between i and i+1 (see figure 1). This gives rise to frequencies in the noise function higher than the primary frequency, which is defined by the spacing of the integer lattice.

Figure 1 One-dimensional trace of noise function.

The noise function can be modified to have an arbitrary, non-zero value at the lattice points. This increases the variance of the function, but adds low frequency components to the signal which cannot be controlled or subsequently removed.[12] For an analysis of the spectral characteristics of such a noise function see Saupe.[8]

The Perlin noise function $N(\vec{p})$ outputs a signal with a fixed lower frequency f. To generate a signal of lowest frequency uf, one can perform a scalar multiplication $u\vec{p}$ of the coordinate vectors supplied to N. This has the effect of moving the reference points in the noise lattice, producing the desired frequency shift in the output of N. We will see this practice used below.

3.3. Frequency Control

Given the noise function, how do we use it to generate more realistic terrains? Subjective observation of natural landscapes reveals that in certain types of mountain ranges there is a marked change in the statistics of the surface as one moves from the foothills to higher peaks. The foothills are more rounded, while the higher mountains are more jagged. Sometimes, as in the eastern slope of the Sierra Nevada, the entire mountain range rises in a relatively short distance from a nearly flat plain. This change of character can be characterized as a change of fractal dimension D_f, crossover scale, or both.

Using the noise synthesis technique we can easily devise terrain models with such features by modulating the power spectrum of the surface as a function of horizontal position and/or vertical altitude. We give some examples below.

Given a (Perlin) noise function $N{:}R^n{\to}R$ with Gaussian distribution in the interval [−1,1], we can generate fBm as follows: For our lowest frequency offset a_0 we have

$$a_0 = N(\vec{p}_0)$$

where $\vec{p}_0$ is the initial object space coordinate vector of the height field position being calculated. Iterating (discretized) fBm at lacunarity λ>1 requires that, at iteration n, the frequency added is proportional to $(f_0\lambda^n)^{-0.5\beta}$. Setting the lowest frequency f_0=1 gives a frequency increment at iteration n of $\lambda^{-0.5\beta n}$. Thus for higher frequencies added at iteration i>0 we have

$$a_i = N(\vec{p}_{i-1}\lambda)\omega^i$$

where $\omega = \lambda^{-0.5\beta}$ (a constant) and $\vec{p}_i = \vec{p}_{i-1}\lambda$. Note that for a two-dimensional noise function $N{:}R^2{\to}R$, we have $\vec{p}_i = \vec{p}_0\lambda^i$. For $N{:}R^3{\to}R$, we may have $\vec{p}_i \neq \vec{p}_0\lambda^i$ due to vertical displacement.

For the purposes of terrain modelling we will introduce a number of parameters to the formulae for fBm. We may wish to translate N by a constant c_t so that it is, for instance, always or nearly always positive. We may also wish to scale N by a factor c_s to reduce or expand its range. In the patches shown in plates 3 and 11, we insert the lowest frequency first:

$$a_0 = (N(\vec{p}_0) + c_t)\ c_s + c_0$$

where a_0 is the initial height of a point in the height field and c_0 is an offset constant which determines the zero value or "sea level" of the terrain. We have for the altitude a_i at stage i>0:

$$a_i = a_{i-1} + a_{i-1}\ (N(\vec{p}_i) + c_t)\ c_s\ \omega^i$$

The procedure used in plates 3 and 11 generates a surface in which the power of the high frequencies is linearly proportional to the (previous) altitude of the surface. This amounts to modulating crossover scale with altitude. In plates 3 and 11, as in the other illustrations of noise-synthetic terrain patches, we set λ = 2. (The rainbow in plate 11 is from Musgrave.[24])

In plate 4 the crossover scale varies with altitude and horizontal position. Here we have

$$a_0 = F(x)\ (N(\vec{p}_0) + c_t)\ c_s + c_0$$

with $F(x) = \min(2x, 2-2x)$, assuming that x varies from 0 to 1. To give the ridge a more natural path than that of a straight line, we add some $1/f^3$ noise to x before calculating $F(x)$. The contribution of higher frequencies is again scaled as

$$a_i = a_{i-1} + a_{i-1}\ (N(\vec{p}_i) + c_t)\ c_s\ \omega^i$$

Fractal dimension can be modulated as easily as the crossover scale by scaling β or H in successive generations. Plate 5 shows a patch which is planar on the left, to space filling on the right (modulo the upper and lower frequency cutoffs, which are approximately at 7 and 1/128 times the patch size, respectively). In this case, we have $\beta = 1/x$ (corresponding to $H = 1/x - 3/2$), and x in the interval (0,1].

In plate 6, we linearly change fractal dimension D_f from 2 (H=1, β=3) to 3 (H=0, β=1) on the right. Note that this is not the same as going from planar ($1/f^\infty$) to filling all of 3 space ($1/f^0$), as in plate 5.

It is interesting to note that experience indicates that modulation of crossover scale is more effective than modulation of fractal dimension for modelling realistic looking terrain. That changing crossover scale alone should have such a dramatic effect is not surprising, for as B. B. Mandelbrot has pointed out,[17] the fractal dimension of the Himalayas is approximately the same as that of the runway at the JFK airport; what is true is simply that the crossover scale of the latter is on the order of millimeters while that of the former is on the order of kilometers.

---

chromaticity, stationarity, and isotropy in a multidimensional Perlin noise function. If the primary (lowest) frequency of an n-dimensional Perlin function is f along the axes of the grid upon which it is defined, then the frequency is $\sqrt{n}f$ along the diagonal of that grid.

It is readily apparent that the global lacunarity λ is subject to precise user control in the noise synthesis scheme. Computational cost for a surface is a function of the number of frequencies used. Thus surfaces generated with small lacunarity will be more expensive to compute than those with large lacunarity. Cost can be reduced by omitting high frequencies when their contribution drops below an arbitrary threshold.

With the noise synthesis method, one may exercise local control over lacunarity. This can be accomplished by displacing the initial coordinate p_0 supplied to the noise function by a vector valued noise function $\vec{N}$ (e. g., Perlin's "Dnoise()"[26]). The effects of such local change of lacunarity are shown in plate 1b, where we modulate intensity on the image plane as

$$intensity = N(\vec{p}_0 + \vec{N}(\vec{p}_0)).$$

(Plate 1a shows the similarly interpreted output of noise $N(\vec{p}_0)$.) Note that local change of lacunarity interferes with the precise local control of frequency. While it is not obvious that this local modulation of lacunarity is particularly useful for terrain synthesis, it may prove useful for the synthesis of other textures, such as clouds, smoke, or flames.

Plates 11 and 12 are details of 100 x 100 patches similar to that in plate 3. Note that the triangles, which are obscured by bump mapping, are quite large in comparison with the overall image. By including only relatively low frequencies in the terrain and leaving high-frequency details to a texture map, we can achieve realistic results using very small height fields.

4. PHYSICAL EROSION MODEL

We have divided erosive processes into two categories: *hydraulic erosion* and *thermal weathering*. Hydraulic erosion is that caused by running water. What we term "thermal weathering" subsumes the non-hydraulic processes which cause rock to flake off steep inclines and form talus slopes at the base. In this section we will illuminate the two erosion simulation algorithms.

4.1. Hydraulic Erosion

The hydraulic erosion model involves depositing water ("rain") on vertices of the height field and allowing the water and sediment suspended in the water to move to any lower neighboring vertices. The erosive power of a given amount of water is a function of its volume and the amount of sediment already carried in the water.

The hydraulic erosion model is implemented by associating with each vertex v at time t an altitude a_t^v, a volume of water w_t^v, and an amount of sediment s_t^v suspended in the water. At each timestep, we pass excess water and suspended sediment from v to each neighboring vertex u. The amount of water passed, Δw, is defined as:

$$\Delta w = min(w_t^v, (w_t^v + a_t^v) - (w_t^u + a_t^u))$$

If Δw is less than or equal to zero, we simply allow a fraction of the sediment suspended in the water at v to be *deposited* at v:

$$a_{t+1}^v = a_t^v + K_d s_t^v$$
$$s_{t+1}^v = (1 - K_d) s_t^v$$

Otherwise, we set

$$w_{t+1}^v = w_t^v - \Delta w$$
$$w_{t+1}^u = w_t^u + \Delta w$$
$$c_s = K_c \Delta w$$

Here, c_s is the *sediment capacity* of Δw. When passing sediment from v to u, we remove at most this amount of sediment from s_t^v and add it to s_{t+1}^u. If c_s is greater than s_t^v, a fraction of the difference is subtracted from a_t^v and is added to s_{t+1}^u, which constitutes the erosion of soil from v. Finally, we allow a fraction of the sediment remaining at v to be deposited as above. Thus, if $s_t^v \geq c_s$, we set

$$s_{t+1}^u = s_t^u + c_s$$
$$a_{t+1}^v = a_t^v + K_d(s_t^v - c_s)$$
$$s_{t+1}^v = (1 - K_d)(s_t^v - c_s)$$

Otherwise,

$$s_{t+1}^u = s_t^u + s_t^v + K_s(c_s - s_t^v)$$
$$a_{t+1}^v = a_t^v - K_s(c_s - s_t^v)$$
$$s_{t+1}^v = 0$$

The constants K_c, K_d, and K_s are, respectively, the *sediment capacity constant*, the *deposition constant* and the *soil softness constant*. K_c specifies the maximum amount of sediment which may be suspended in a unit of water. K_s specifies the softness of the soil and is used to control the rate at which soil is converted to sediment. K_d specifies the rate at which suspended sediment settles out of a unit of water and is added to the altitude of a vertex.

Through the above process, water and, more importantly, soil from higher points on the landscape are transported to and deposited in lower areas. This movement constitutes the communication necessary for modelling the global process of erosion. In a full two-dimensional implementation, one must take care to distribute water and sediment to all neighboring lower vertices in amounts proportional to their respective differences in overall elevation.

Although this model is *ad hoc*, the resulting landscapes bear reasonable resemblance to natural erosion patterns. Further research will concentrate on constructing a more sophisticated, physically accurate model.

Plates 2 and 9 show a terrain patch before and after 2000 time steps of hydraulic and thermal erosion. The erosion simulation required approximately 4 hours of CPU time on a Silicon Graphics Iris 4D/70 workstation. The uneroded patch shows a good first approximation to an eroded landscape with a central stream bed. The uneroded patch was created by weighting the addition of always positive noise values by the distance d of the point from the diagonal of the patch, which diagonal is also "higher" at the far end. The stream bed is made non-linear by the addition of $1/f^3$ noise to the distance d. In this simulation, $K_c = 5.0$, $K_d = 0.1$, and $K_s = 0.3$. Note the gullys, confluences, and alluvial fans that have appeared in the eroded patch, which is rendered as a dry wash, i.e., without water present.

The distribution of rainfall on landscapes in nature is strongly influenced by *adiabatics*, or the behavior of moisture-laden air as it rises and descends in the atmosphere. As air rises, it cools and the relative humidity rises. When the relative humidity exceeds one hundred percent, clouds form; when the clouds become dense, precipitation occurs. Wind blowing over mountains raises air as it passes over the mountains, thus precipitation is much greater at the top and downwind of, mountain peaks. It is easy to include a rough approximation of adiabatic effects in our erosion model by making precipitation a linear function of altitude. This has a significant effect on the erosion patterns produced.

In our use of the hydraulic erosion model, we have simply allowed a fixed amount of rain (approximately one one thousandth of the height of the vertex) to fall at regular intervals (approximately every sixty to one hundred time steps). Mandelbrot and Wallis[13] have pointed out that records of flooding of the Nile river show a $1/f$ noise distribution, i.e., large floods happen with low frequency. Such a noisy distribution in the

rainfall would constitute a more realistic simulation of nature; it is probable that it would have a long-term effect on the erosion features created. This is an idea yet to be explored.

4.2. Thermal Weathering

The other erosion process we model is thermal weathering, which is a catch-all term for any process that knocks material loose, which material then falls down to pile up at the bottom of an incline. The thermal weathering process creates talus slopes of uniform angle. Thermal weathering is a kind of relaxation process and is both simple to implement and fast. At each time step t+1, we compare the difference between the altitude a_t^v at the previous time step t of each vertex v and its neighbors u to the (global) constant talus angle T. If the computed slope is greater than the talus angle, we then move some fixed percentage c_t of the difference onto the neighbor.

$$a_{t+1}^u = \begin{cases} a_t^v - a_t^u > T: & a_t^u + c_t(a_t^v - a_t^u - T) \\ a_t^v - a_t^u \le T: & a_t^u \end{cases}$$

With care taken to assure the equitable distribution of talus material to all neighboring vertices, the slope to the neighboring vertices asymptotically approaches the talus angle.

Plates 7 and 8 show a patch created with non-uniform lacunarity before and after slumping or thermal weathering. This process has created a rough approximation of sand dunes.

4.3. Discussion of Erosion Models

An interesting extension would be to account for the differing hardnesses of bedrock, silt, and talus. This can be accomplished by adding appropriate fields to the vertex data structure and making the simplifying assumption that silt lies on top of talus, which in turn lies on top of bedrock. Another simple extension will be to add strata of differing hardnesses to the bedrock, as is commonly seen in sedimentary rock. This can be accomplished through the use of a Perlin texture (as in Plate 13) or more efficiently by table lookup of a vertically perturbed one-dimensional array of hardnesses. Finally, it would be desirable to render the water flowing on the landscape. To do this realistically represents a major challenge and will be the aim of future research.

5. RAY TRACING HEIGHT FIELDS

Having created eroded fractal height fields, we now need to render them quickly and realistically. We present a fast ray tracing technique for height fields, termed *grid tracing*.[22]

The basic idea is this: A two-dimensional array of altitude values is traversed in an arbitrary direction by a ray, through the use of a modified *DDA* (*Digital Differential Analyzer*) algorithm. The array is thought of as composing a *grid* of small square *cells* corresponding to the pixels being illuminated by a DDA algorithm. Each cell has associated with it the height field altitudes at the four corners of the cell. As the ray traverses the array of cells, the altitude of the ray at each cell is compared to the four altitudes associated with the cell. Ray/surface intersection tests need only be performed when the altitude span of the ray over the extent of the cell intersects the interval of altitudes defined by the lowest and highest of the four altitudes associated with the cell. For a ray traveling **above** the surface, the condition can be stated:

$$min(ray_{z_{near}}, ray_{z_{far}}) \le max(h_{i,j}, h_{i,j+1}, h_{i+1,j}, h_{i+1,j+1}) \qquad (2)$$

where $ray_{z_{[near,far]}}$ represents the altitudes of the end points of the ray segment within the cell and the $h_{m,n}$ represent the altitudes of the height field H at the four corners of the i,j^{th} cell. As the surface of the terrain within a cell can be represented by exactly two triangles which split the square diagonally, the ray/surface intersection test consists of two ray/triangle intersection tests. These tests are greatly simplified by the shape and orientation of the triangles. Only rays grazing past the surface will fail the intersection tests; most rays will incline directly into the surface at the first cell where surface intersection is tested. Note that the calculation in expression (2) can be simplified by creating an auxiliary array of values for each cell which are equal to the right hand side of (2).

Advantages of this algorithm are manifold. First, only the height field need be calculated and stored as the model. The actual polygon descriptions (i.e., the plane equations of the triangles) need only be calculated when an intersection test is performed. This can save both time and space in the creation of the terrain model, as polygons which are not visible in the rendering are never fully described. Second, the grid traversal can be accomplished with the use of a modified Bresenham DDA[27] algorithm. The Bresenham algorithm is a highly optimized, fast algorithm which uses only floating point addition* in determining the height of the ray and the next cell along the path of the ray. Third, the algorithm is general. Any two-dimensional array of scalar data (i.e., an image) may be interpreted as a height field and ray traced with this algorithm. Fourth, the algorithm performs ray/object intersections in $O(\sqrt{k})$ time with a small constant multiplier, where k is the number of cells in the grid.

Furthermore, the grid tracing algorithm can be made hierarchical and, for fractal terrains which are not post-processed as by an erosion simulation, procedural. The hierarchy is created by having each cell contain another grid rather than two triangles. Each altitude associated with the cell is then equal to the height of the bounding box of the subgrid. Hierarchical decomposition is desirable for ray tracing large grids, as implementation as a hierarchy of n x n grids (an n^2 tree) reduces the time complexity to $O(\log_n \sqrt{k})$. In renderings of a portion of a 1217 x 1217 grid, grid traversal time was reduced by approximately 50% with a 16^2 tree implementation. Plate 10 is such a rendering, and at one ray per pixel at 1280 x 1024 resolution, it required less than one half of an hour of CPU time times eight CPUs using a parallel ray tracing scheme[22] on an Encore Multimax computer, under the C-Linda parallel programming language.[1]

Grid tracing of procedural terrain models enables adaptive level of detail and efficient memory usage in that no unnecessary height field values need be computed. In the procedural implementation, height field values are calculated and stored when a cell is first traversed by a ray. Measuring divergence of *primary* (first-generation) rays at the far end of a cell determines whether a cell needs to be decomposed into a subgrid; such divergence is a linear function of the distance the ray has traveled. This simple metric breaks down in the face of the bane of ray tracing: rays reflected and refracted by curved or bump-mapped surfaces.

Grid tracing is a memory-bound algorithm. While the grid can be traversed at great speed, page faults generated when a grid is too large to fit into main memory severely compromise the speed of a grid tracing program. A 1000 x 1000 grid composed of two byte integer altitude values comprises two megabytes of memory. If one elects to store the plane equation data for triangles when it is calculated for ray/surface intersection

* The best-known Bresenham DDA is an integer algorithm. The version used for our purposes is not the integer Bresenham DDA, but rather a slightly less optimal floating point version. A simpler alternate scheme could use an ordinary integer DDA for the traversal, but would need to check one cell to either side of the ray path for possible intersections due to imprecision in tracking that path.

tests, the memory usage increases steadily as the image is rendered unless active memory management is implemented. We have found it beneficial to store a small number of triangles in a cache organized as a linked list of triangle data stored in most-recently-used-first order.

Note that height fields tessellated by equilateral triangles, as opposed to right triangles, can be ray traced just as efficiently. A right triangle can be transformed into an equilateral triangle with a skewing and a scaling transformation such as $x = x+y/2$ and $y = y\sqrt{3}/2$. The inverse of the product of these transformations can be applied to the ray, whereupon the ray can traverse a rectilinear grid in "grid space". This is useful because an equilateral triangle tessellation of a surface requires less stored data per unit area of surface and is, upon rendering, often more aesthetically appealing than a right triangle tessellation.

6. CONCLUSION

We have demonstrated a new method for creating fractal terrains which gives a first approximation to erosion features, at terrain generation time. As opposed to previous methods, noise synthesis allows local control over frequencies comprising the surface. We have also suggested two effective erosion algorithms which simulate hydraulic erosion by flowing water and thermal weathering, which chips away steep inclines and forms talus slopes. The hydraulic erosion simulation requires a significant amount of computer time to create extensive, deep drainage systems, while the thermal weathering can be quite fast. Finally, we have described a ray tracing algorithm for height fields which is efficient enough to allow the ray tracing of detailed terrains in a reasonable amount of computer time.

Acknowledgements

The work of Ken Musgrave and Craig Kolb was accomplished in Benoit Mandelbrot's project at Yale University as a part of our ongoing quest for more beautiful and realistic forgeries of nature, using the latest results in fractal geometry. We are deeply grateful to Benoit Mandelbrot, and to Dietmar Saupe, for their comments, corrections, and inspiration in this endeavor. The work at Yale was funded, in part, through the Office of Naval Research contract N00014-88-K-0217, and development was carried out with the invaluable assistance of Nick Carriero, Rob Bjornson, and David Gelerenter of the Yale Computer Science Department, using C-Linda[1] on the Encore Multimax. That system is supported, in part, by NSF grants DCR-8601920 and DCR-8657615. All the authors, particularly Rob Mace, are indebted to Silicon Graphics, Inc. for their generous support.

References

1. Ahuja, S., N. Carriero, and D. Gelerenter, ''Linda and Friends,'' *IEEE Computer*, August, 1986.
2. Bouville, Christian, ''Bounding Ellipsoids for Ray-Fractal Intersection,'' *Computer Graphics*, vol. 19, no. 3, pp. 45-52, July 1985.
3. Cook, Robert L., ''Shade Trees,'' *Computer Graphics*, vol. 18, no. 3, pp. 223-230, July, 1984.
4. Fournier, Alain, D. Fussell, and L. Carpenter, ''Computer Rendering of Stochastic Models,'' *Communications of the ACM*, vol. 25, pp. 371-384, 1982.
5. Fujimoto, A., T. Tanaka, and K. Iwata, ''ARTS: Accelerated Ray Tracing System,'' *IEEE Computer Graphics and Applications*, vol. 6, no. 4, pp. 16-26, April, 1986.
6. Gardner, Geoffrey Y., ''Visual Simulation of Clouds,'' *Computer Graphics*, vol. 19, no. 3, pp. 297-303, July, 1985.
7. Gardner, Geoffrey Y., *Functional Modelling (SIGGRAPH course notes)*, Atlanta, 1988.
8. Jurgens, H., Dietmar Saupe (eds.), and Dietmar Saupe, ''Point Evaluation of Multi-Variable Random Fractals,'' *Visualisierung in Mathematik und Naturissenschaft - Bremer Computergraphik Tage 1988*, Springer-Verlag, Heidelberg, 1989.
9. Kajiya, James T., ''New Techniques for Ray Tracing Procedurally Defined Objects,'' *Computer Graphics*, vol. 17, no. 3, July, 1983.
10. Kajiya, James T., ''New Techniques for Ray Tracing Procedurally Defined Objects,'' *Transactions on Graphics*, vol. 2, no. 3, pp. 161-181, July, 1983.
11. Kelley, Alex D., M. C. Malin, and G. M. Nielson, ''Terrain Simulation Using a Model of Stream Erosion,'' *Computer Graphics*, vol. 22, no. 4, pp. 263-268, August, 1988.
12. Lewis, J. P., ''Generalized Stochastic Subdivision,'' *ACM Transactions on Graphics*, vol. 6, no. 3, pp. 167-190, July, 1987.
13. Mandelbrot, Benoit B. and J. R. Wallis, ''Some Long-Run Properties of the Geophysical Records,'' *Water Resources Research 5*, pp. 321-340, 1969.
14. Mandelbrot, Benoit B., ''Stochastic Models for the Earth's Relief, the Shape and the Fractal Dimension of the Coastlines, and the Number-Area Rule for Islands,'' *Proceedings of the National Academy of Sciences (USA)*, vol. 72, pp. 3825-3828, 1975.
15. Mandelbrot, Benoit B., *The Fractal Geometry of Nature*, W. H. Freeman and Co., New York, 1982.
16. Mandelbrot, Benoit B., ''Comment on Computer Rendering of Stochastic Models,'' *Communications of the ACM*, vol. 25, no. 8, pp. 581-583, 1982.
17. Mandelbrot, Benoit B., *personal communications*, 1988.
18. Mastin, Gary A., P. A. Watterberg, and J. F. Mareda, ''Fourier Synthesis of Ocean Waves,'' *IEEE Computer Graphics and Applications*, vol. 7, no. 3, pp. 16-23, March, 1987.
19. Miller, Gavin S. P., ''The Definition and Rendering of Terrain Maps,'' *Computer Graphics*, vol. 20, no. 4, pp. 39-48, 1986.
20. Miller, Gavin S. P., *personal communications*, 1988.
21. Musgrave, F. Kenton, Craig E. Kolb, and B. B. Mandelbrot, *A Survey of Terrain Synthesis Techniques*, to appear.
22. Musgrave, F. Kenton, ''Grid Tracing: Fast Ray Tracing for Height Fields,'' *Yale Dept. of Computer Science Research Report RR-639*, July 1988.
23. Musgrave, F. Kenton and B. B. Mandelbrot, ''About the Cover,'' *IEEE Computer Graphics and Applications*, vol. 9, no. 1, January, 1989.
24. Musgrave, F. Kenton, ''Prisms and Rainbows: a Dispersion Model for Computer Graphics,'' *Proceedings of the Graphics Interface '89 - Vision Interface '89*, London, Canada, June, 1989.
25. Peitgen, H. O. and Dietmar Saupe (eds.), *The Science of Fractal Images*, Springer-Verlag, New York, 1988.
26. Perlin, Ken, ''An Image Synthesizer,'' *Computer Graphics*, vol. 19, no. 3, pp. 287-296, July, 1985.

27. Rogers, D. F., *Procedural Elements for Computer Graphics,* Mc Graw Hill, New York, 1985.

28. Voss, Richard F., *Random Fractal Forgeries,* Springer-Verlag, Berlin, 1985. (in Fundamental Algorithms for Computer Graphics, R. A. Earnshaw, ed.)