

## Fast Generation and Surface Structuring Methods for Terrain and Other Natural Phenomena

Eng-Kiat Koh  
Information Technology Institute  
Republic of Singapore

D. D. Hearn\*  
Department of Computer Science and  
National Center for Supercomputing Applications  
University of Illinois at Urbana-Champaign, USA

### Abstract

Fractal geometry has proven to be a powerful tool for modeling natural phenomena. Using discrete approximations to fractional Brownian motion over a finite grid plane, computer graphics terrain-rendering algorithms are able to generate highly realistic topographical displays. Similar procedures can be applied to model other natural phenomena, such as clouds and water. Two important considerations in these algorithms are computational efficiency and the ability to control macroscopic surface features. Here we introduce a technique for structuring surface features so as to conform to a specified "elevation" envelope. We also present methods for implementing this technique using a recursive random midpoint-displacement procedure.

**Key Words:** Fractal Geometry, Fractional Brownian Motion, Random Midpoint Displacements, Terrain and Natural Phenomena Modeling, Control Surface, Control Elevations, Terrain Envelope

### 1. Introduction

Graphical simulation of natural phenomena involves a tradeoff between realism and speed of computation, since accurate modeling of natural scenes requires complex object descriptions. Real-time flight simulators currently use simplified object models [11], sacrificing some realism to attain the required processing speed. On the other hand, animation sequences for feature films or for television commercials often require more realistic displays of natural phenomena and so must tolerate longer image-generation times. But in many applications, a compromise solution is desirable that allows both realism and expeditious image generation.

There are two general approaches to modeling terrain and other natural phenomena. Object geometry can be described with Euclidean constructs, or we can set up a procedural description to model more accurately the irregularities that are characteristic of natural objects. Using Euclidean geometry, we can model each object

---

\*

Department of Computer Science; 1304 W. Springfield Avenue; Urbana, IL 61801; USA.  
Email: hearn@cs.uiuc.edu Fax: 217-333-3501.

in a scene as a set of one or more equations, usually polynomials representing quadric or bicubic spline surfaces. Procedural descriptions of an object are typically based on the methods of fractal geometry, which allow an object to be modeled with "infinite" detail.

Euclidean geometry methods can reduce computational requirements while providing sufficient realism for some applications by approximating surfaces of natural objects with standard geometric shapes. Combinations of plane and quadric surfaces, for example, have been used to model the macroscopic surface features of terrain, trees, and clouds [4]. Surface detail is then added using texture mapping to generate an "impressionistic" display of the scene. This allows scenes to be generated fairly quickly, and reasonably realistic terrain displays of rolling hills and gentle mountain ranges are possible. However, the level of detail occurring in natural phenomena cannot be adequately described with the methods of Euclidean geometry. One would be hard pressed to generate satisfactory displays of the Alps or the Rocky Mountains using quadric surfaces overlaid with texture maps.

Fractal geometry methods [9, 13] produce the most realistic rocky terrain displays, but at higher computational cost. Terrain variations are now modeled as approximations to the  $1/f$  noise commonly observed in a wide range of natural phenomena. A useful mathematical model for describing such variations is *fractional Brownian motion* [9, 10], an extension of the concept of ordinary Brownian motion. Surface variations can be described as a type of random walk, and realistic terrain features are produced by generating finite samples of fractional Brownian motion above a ground plane. Computations in this procedure can be reduced by approximating the fractional Brownian motion with a midpoint displacement approach [1, 3].

As a means for controlling the surface contours of terrain and other natural phenomena, a surface envelope can be specified to constrain generated elevations above the ground plane. Here, we present a fast and simple method for specifying surface structure by superimposing a polygon envelope on the elevation calculations and structuring the algorithms for parallel execution [6]. A more complex and slower procedure for imposing surface constraints is to combine fractal methods with spline functions [12]. Another approach that has been suggested is to fit a given set of elevation data points with triangular patches, then apply a random midpoint displacement to the edges of those triangles larger than a preset value [2].

In the following section, we review procedures for generating fractal surfaces. Then we discuss efficient methods for modifying these procedures to control surface structure using a polygonal envelope.

## 2. Generating Fractional Brownian Motion

To generate terrain elevations as fractional Brownian motion (fBm), we first set up a finite reference (ground) plane, usually square, referenced as the  $xy$  plane. Elevations  $\mathbf{z} = \mathbf{z}(\mathbf{x}, \mathbf{y})$  are then calculated as functions of grid positions  $(\mathbf{x}, \mathbf{y})$  over this ground plane. A fractional Brownian motion for elevation variations is then defined as a Gaussian distribution over the increments  $\Delta\mathbf{z}$ , with a variance proportional to a power of the ground-plane distance between the two elevation positions [13]:

$$\langle |\Delta\mathbf{z}|^2 \rangle \propto [(\Delta x)^2 + (\Delta y)^2]^H \quad (1)$$

Here, the variance  $\langle |\Delta z|^2 \rangle$  represents an average value for the elevation differences between the two ground-plane positions, and  $0 < H < 1$ . In Fig 1 the ground-plane distance  $d = \{(\Delta x)^2 + (\Delta y)^2\}^{1/2}$  is shown for two elevation points.

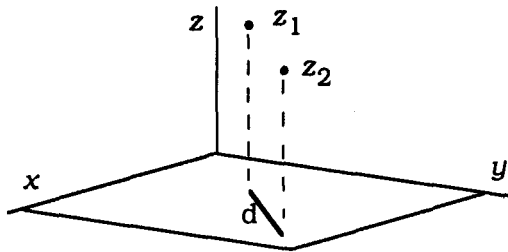


Figure 1. Ground plane distance between two elevation points.

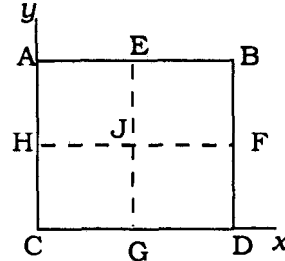


Figure 2, Subdividing the ground plane.

Parameter  $H$  determines the roughness, or fragmentation, of the surface and is related to the fractal dimension. The value  $H = 1/2$  yields ordinary Brownian motion, where variance is directly proportional to ground-plane distance. Fractal dimension  $D$  of the generated terrain surface is determined from parameter  $H$  as

$$D = 3 - H \tag{2}$$

The larger the value of  $D$ , the rougher and more fragmented the surface appears. Values in the neighborhood  $D = 2.2$  ( $H = 0.8$ ) generate very realistic earth mountains, while higher values of  $D$  can be used to create extraterrestrial landscapes.

Fractional Brownian motion also has the property that its spectral density (power spectrum) is inversely proportional to a power of frequency. For the two-dimensional elevation function  $z$ , we have:

$$|Z(f_x, f_y)|^2 \propto \frac{1}{(f_x^2 + f_y^2)^{1+H}} \tag{3}$$

where  $Z(f_x, f_y)$  is the Fourier transform of the elevation  $z(x, y)$ . Thus fBm produces elevation variations that exhibit  $1/f$  noise.

There are several methods that have been used to approximate fBm on a finite ground-plane grid. An accurate approximation of fBm can be obtained by representing  $z(x, y)$  as a discrete inverse Fourier transform and making use of the relationship in (3). A faster, but less accurate method, is to calculate elevations as random displacements in a recursive midpoint subdivision process.

### Discrete Fourier Transform Method

On the reference plane, an  $n$  by  $n$  square grid of  $(x, y)$  coordinate positions is established, with

$$x_i = i\delta, \quad y_j = j\delta \quad \text{for } i, j = 0, 1, 2, \dots, n-1 \tag{4}$$

where  $\delta$  is the grid step size for both coordinates. Elevations at the grid positions can then be denoted as

$$z_{ij} = z(i\delta, j\delta) \quad (5)$$

We obtain the sample path for the elevations by requiring that its power spectrum satisfy relation (3). Thus, we determine appropriate values for the Fourier transform  $Z_{kj}$  and obtain the required elevations with the discrete inverse Fourier transform:

$$z_{ij} = \sum_{k,m=0}^{n-1} Z_{km} e^{i2\pi(f_k x_i + f_m y_j)} \quad (6)$$

where the spatial frequencies are

$$f_k = \frac{k}{n\delta}, \quad f_m = \frac{m}{n\delta} \quad (7)$$

The appropriate values for  $Z_{km}$  can be obtained by generating uniformly distributed random numbers (white noise) and filtering with a transfer function satisfying the proportionality in (3). That is, we generate pseudorandom complex numbers and multiply by the factor

$$\frac{1}{(f_k^2 + f_m^2)^{(1+H)/2}} \propto \frac{1}{(k^2 + m^2)^{(1+H)/2}} \quad (8)$$

Equation (6) is evaluated by setting up the coordinate grid so that  $n$  is a multiple of 2 and applying the fast Fourier transform (FFT) algorithm. Using the FFT algorithm reduces the amount of computation, but the number of multiplications and additions required to evaluate each summation in Eq. (6) is still proportional to  $n \log_2 n$ . This is quite expensive for most applications; however, this method has produced some of the most realistic terrain displays [9, 13]. Some other difficulties with this approach are that all of the elevations must be calculated at the same time, surface detail is not easily varied across the grid, and it is not easy to extend the terrain beyond the grid without recalculating a completely new grid of elevations.

Rendering the terrain model is accomplished by connecting the coordinates of the elevation points so as to form a set of triangular polygons. Individual polygons are shaded according to the lighting effects set for the scene, the type of terrain features to be modeled (rocky, snowcapped, etc.), and the orientation of the viewing plane.

Although elevations are generated randomly, some postprocessing can be performed to adjust relative terrain heights. This is accomplished by raising each calculated elevation value to a power  $p$ . For  $p > 1$ , valley regions (where  $z < 1$ ) are flattened and mountain peaks ( $z > 1$ ) are elongated. For  $p < 1$ , the reverse is obtained: peaks are flattened and valleys become steeper.

### ***Random Midpoint-Displacement Method***

This procedure simplifies the calculation of elevations by using a recursive subdivision approach [1, 3, 7, 8, 13]. It eliminates the Fourier series calculations, and involves only successive averaging and the addition of a Gaussian random variable to obtain each elevation value.

We begin the procedure by assigning an elevation value to each of the four corners of the rectangular ground plane. Then the original boundaries of the ground plane are divided in half to obtain five new grid points (E, F, G, H, and J), as shown in Fig. 2. Elevations for these new grid positions are calculated as averages of the original elevations plus a random value. The process is repeated for each new subgrid (e.g., AEJH in Fig. 2). and subdivision continues to a desired level of recursion.

An *average elevation* can be defined for each new grid point using elevations of the nearest neighbors of that point. For example, average elevations for points E and F in Fig. 2 can be defined as  $(z_A + z_B)/2$  and  $(z_B + z_D)/2$ , respectively. The final elevation for these points is then obtained by adding a random value:

$$\begin{aligned} z_E &= (z_A + z_B)/2 + r \\ z_F &= (z_B + z_D)/2 + r \end{aligned} \tag{9}$$

where  $r$  is generated from a Gaussian distribution with zero mean and appropriate variance. Average elevation for the grid center (point J) could be determined from the elevations of points E and G, or points H and F. Alternatively, we could calculate the elevation of this center point as the average of the corner elevations plus a random value:

$$z_J = (z_A + z_B + z_C + z_D)/4 + r \tag{10}$$

Other schemes are possible for calculating the average elevations of the grid points generated at each subdivision.

To obtain an approximation to fBm, we can require that the variance of the Gaussian distribution satisfy Eq. (1). That is, the variance should be proportional to the grid separations raised to the  $H$  power. The process can be simplified by selecting  $r$  as a table lookup from a Gaussian distribution with zero mean and a variance of 1. Then  $r$  can be multiplied by the grid separation times a surface "roughness" factor.

Triangular surface patches can be formed as the elevations are generated. At each level of recursion, the triangles are subdivided into successively smaller planar patches. The final scene is then displayed by shading the patches according to the type of terrain features to be modeled and the positions of the light sources and viewing plane.

Calculating elevations as recursive midpoint displacements does not preserve all the properties of fBm. In particular, this process is not stationary. That is, its statistical properties are not constant across the grid. This leads to visible creases across terrain surfaces that have an artificial, papier-mache look. However, it is possible to smooth this effect by applying random variations throughout the grid. Also, a major advantage of the random midpoint displacement method is that the number of operations necessary to calculate  $n$  grid elevations is of order  $n$ .

### 3. Structuring Terrain Features

The elevations generated with the previous two algorithms are random variables that depend only on the separation between grid points, the fractal dimension, and the parameters of the Gaussian distribution. In the random midpoint displacement method, for example, successively decreasing random elevation changes are produced since the

distance between grid points is halved at each step. Adjustments could be made to superimpose a desired structure on the terrain features by applying various postprocessing methods. For instance, all elevations could be raised to a power chosen to heighten or lessen the contrast between peaks and valleys. Alternatively, local adjustments could be made to produce other effects, such as enhancing selected peaks or creating a new valley. A more efficient method for structuring terrain features is to impose the desired restrictions on the Gaussian distribution function so that generated elevations automatically conform to a prespecified structure.

### ***Modified Gaussian Distribution Function***

Terrain features can be controlled in the fBm algorithms by setting the parameters for the Gaussian distribution according to the type of terrain features to be modeled. In this way, elevations can be directly generated that are within a given tolerance of a predefined *terrain envelope*, which specifies the required terrain features. From this envelope, we determine a set of *control elevations* for the grid points on the ground plane. The terrain envelope is user defined and could be specified in several ways (as discussed in the next section).

We can develop an efficient structured terrain generator by modifying the random midpoint displacement method so that the mean ( $\mu$ ) and standard deviation ( $\sigma$ ) for the Gaussian distribution are calculated as functions of the control elevations. That is, we use the specified control elevation at each grid point to determine values for  $\mu$  and  $\sigma$  at that point. A generated elevation value will then lie near the terrain envelope, with a tolerance determined by the standard deviation. There are a number of ways we could set up the calculations for these parameters. A straightforward method that produces good results is simply to set  $\mu$  and  $\sigma$  proportional to the difference between the calculated average elevation and the predefined control elevation at that point. The random number generated from the Gaussian distribution thus acts as an adjustment to ensure that the elevation at a grid point is close to the control elevation, within limits set by the value of  $\sigma$ . We could also set the variance of this distribution proportional to grid separations times a surface roughness factor.

Calculation of the Gaussian parameters can be illustrated with Fig. 2. Initially, elevations for the corner positions (A, B, C, and D) are set to the specified control elevations, so that these points lie on the terrain envelope. At the first subdivision level, elevations are calculated as in Eq. (9), except that the mean and standard deviation are now functions of the control elevation at each position. For example, values of these parameters for grid position F can be calculated as

$$\begin{aligned}\mu_F &= z_{C_F} - (z_B + z_D) / 2 \\ \sigma_F &= s |\mu_F|\end{aligned}\tag{11}$$

where  $z_{C_F}$  is the control elevation for position F, and  $0 < s < 1$  is a predetermined scaling factor. The Gaussian variable  $r$  generated from this distribution will have a value "close" to the difference between the control elevation and the average elevation at each position. Small values for  $s$  (say,  $s < 0.1$ ) produce tighter conformity to the terrain envelope, while larger values of  $s$  provide for greater elevation fluctuations. Gaussian variables with mean  $\mu$  and standard deviation  $\sigma$  are obtained from standard lookup tables.

### ***Specifying the Surface Envelope***

There are a number of ways one might specify a polygon surface envelope. For example, the control elevations could be determined from a given set of contour lines in the ground plane or from a set of control surfaces specified above the ground plane. For specified contours, the contour lines can be connected in various ways to generate a polygonal shell over the ground plane.

To design special terrain features, we could specify a set of control positions above the ground plane to define the control polygon vertices. Grid intersection points in the ground plane can then be projected up to intersect the specified polygon control surfaces. Figure 3 illustrates a terrain envelope constructed with intersecting pyramids that could be used to define a series of mountain peaks.

When overlapping objects are used to specify the surface envelope (Fig. 3), it is possible for more than one control surface to overlay a particular grid position. We adopt the convention that the control elevation for that position is determined by the highest control surface at that point.

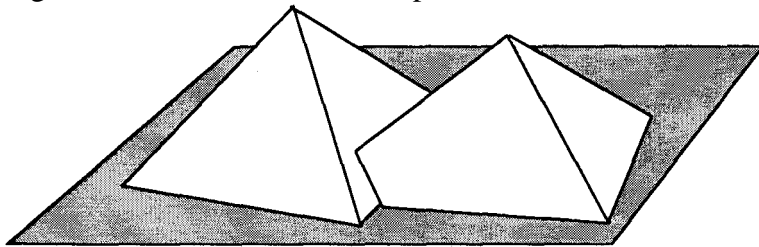


Figure 3. A terrain envelope over a ground plane.

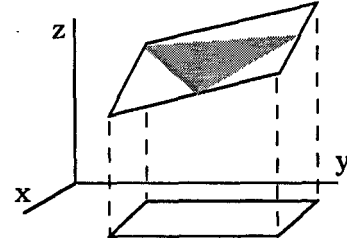


Figure 4. Bounding rectangle of control surface projected to ground plane.

### ***Calculating Control Elevations***

With the surface envelope specified as a set of control polygons, we can efficiently determine grid-point control elevations by projecting the control surfaces onto the ground plane and using a "scan-line" approach to identify those grid positions that lie within the surface boundaries. With this approach, we incrementally calculate both the grid intersection positions on the control surface boundaries and the elevations over the control surfaces. We assume that all control surfaces are specified (or constructed) as triangles.

To determine the control elevation for each grid point in the ground plane, we can make use of methods similar to those applied in the depth-buffer (z-buffer) hidden-surface algorithm [5]. An "elevation buffer" is used to store current elevation values at each step of the process. Initially, each grid point is assigned the ground elevation value 0. Control surfaces are processed one at a time to determine values for the control elevations over the surfaces. As each control elevation corresponding to a grid point is calculated for a particular surface, the calculated elevation is compared to the current contents of the elevation buffer. If the buffer value is smaller, the elevation for that grid point is updated to the newly calculated value. After all surfaces have been processed, the elevation buffer contains the final control elevations for the grid points.

Each triangular control surface is processed by first calculating plane parameters A, B, C, and D from the coordinate values of the three polygon vertices, then determining control elevations for that surface as a function of these parameters. For

any ground-plane position  $(x, y)$ , the corresponding elevation in the plane of the control triangle is calculated as

$$z = (-Ax - By - D) / C \tag{12}$$

Once we have determined the control elevation  $z_{ij}$  corresponding to grid position  $(x_i, y_j)$  on the ground plane, the control elevations of adjacent row and column positions in the grid are obtained incrementally:

$$z_{i+1,j} = z_{ij} - \Delta x (A / C) \tag{13}$$

$$z_{i,j+1} = z_{ij} - \Delta y (B / C)$$

with  $\Delta x$  and  $\Delta y$  as the grid spacing in the x and y directions.

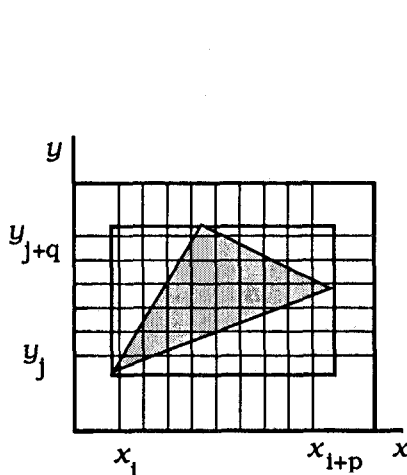


Figure 5. Grid rows and columns overlapping the ground plane projection of a control surface.

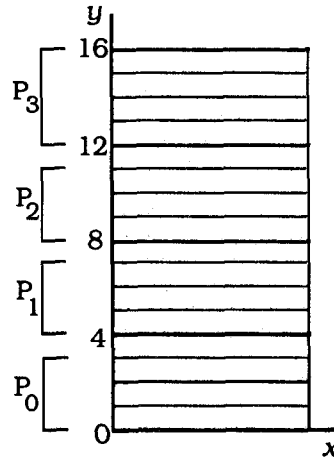


Figure 6. Assignment of 17 grid rows to 4 processors to calculate control elevations.

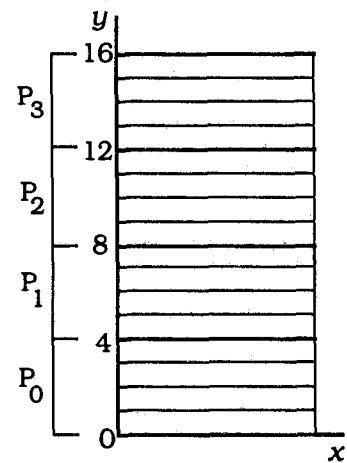


Figure 7. Assignment of 17 grid rows to 4 processors to calculate average midpoint elevations.

The grid rows and columns affected by a particular control surface are those that overlap the bounding rectangle of the surface (Figs. 4 and 5). These grid positions can be processed with procedures similar to those used in scan-line area-filling algorithms [5]. For each y "scan line" inside the bounding rectangle, the x intersections with the edges of the control surface are obtained from the edge parameters. Thus for  $y_j$  in Fig. 5, an edge intersection is calculated as

$$x_j = (y_j - b) / m \tag{14}$$

where  $m$  is the slope of the edge,  $b$  denotes the y-intercept value for that edge, and  $x_j$  is a coordinate value that may not coincide with a grid intersection  $x$  value. Intersections along this edge for successive y values are calculated incrementally:

$$x_{j+1} = x_j + \Delta y / m \tag{15}$$

For the example in Fig. 5, these incremental calculations are carried out for  $y_j$  through  $y_{j+q}$ .

Grid positions affected by a control surface are those that lie between the two intersections calculated for each  $y$  "scan line" by Eqs. (14) and (15). Control elevations for these grid points are determined with Eqs. (12) and (13). These operations are performed for each control surface, and the current maximum control elevations are stored in the grid array as each surface is processed.

#### 4. Implementation on Parallel Vector Architectures

Further speedups can be obtained in the algorithms for structuring and generating terrain by optimizing them for vectorization and concurrency on systems that support such operations. These enhancements can make substantial improvements in efficiency, since the terrain algorithms are primarily based on vector calculations and there are several ways that parallelism can be exploited. Calculations over a regular grid (instead of triangular patches) make it easier to vectorize the code, since height values can be obtained as simple vector additions.

##### *Vectorization Methods*

Effective vectorization methods require that the number of array elements to be processed should be either a multiple of the vector register length or very large compared to the register length. In the latter case, the cost for processing the extra elements is not significant compared to the total computation time. We can satisfy the first condition by setting the size of the ground-plane grid so that vector lengths are multiples of the register length. For very large grids, we also satisfy the second condition.

Vector operations occur in several places in the terrain modeling algorithm. These operations include the control elevation calculations in Eq. (13), evaluation of the Gaussian parameters for grid positions in Eq. (11), determination of the random variable  $r$ , and the final evaluation of grid elevations in Eq. (9).

Control elevation calculations can be optimized for vectorization by carrying out the operations in Eq. (13) for all rows and columns within the bounding rectangle of the control surface (Fig. 5). That is, we evaluate  $z_{km}$  for grid positions  $(x_k, y_m)$  with  $k = i, i+1, \dots, i+p$  and  $m = j, j+1, \dots, j+q$ . The vectors can be specified across either the  $y$  rows or the  $x$  columns, depending on whether the width of the rectangle is greater than the height, or vice versa. Obtaining control elevations for all points within the bounding rectangle simplifies the calculations, since the plane equation (12) need be evaluated at only one grid point (rather than once for every "scan line" at the starting edge intersection position). All other calculations require only the single subtraction of Eq. (13). Once the boundary intersection points have been determined from Eq. (15), the control elevations corresponding to grid points inside the control surface can be processed against the elevation buffer, as described in Section 3. Since only two edge intersections are needed for each "scan line", these calculations are too few to be vectorized. Instead, we can tack the edge intersection values (Eq. (15)) onto the end of the control elevation vector.

After the control elevations for all grid positions have been determined, we can calculate the final grid elevations. We optimize these calculations for vectorization by setting up the vectors on an  $(n+1)$  by  $(n+1)$  grid. Also, we set  $n$  to be a power of 2 so that

the grid can be successively divided by 2 an integral number of times to carry out the subdivision process. Initially, we assign control elevations to the four corners of the grid. The remaining  $n-1$  positions along the top and bottom of the grid are assigned elevations as calculated by the random midpoint displacement procedure using parameters from Eqs. (11) and the table lookup for  $r$ . Grid elevations along the midline of the grid can then be computed as a vector operation, using the grid elevations along the top and bottom edges of the grid. This vector operation can then be repeated recursively for each half (top and bottom) of the grid and continues until all  $n-1$  lines between the bottom edge and the top edge of the grid have been processed.

### Concurrency Optimizations

There are several ways that we can effectively take advantage of concurrent processing for the two classes of objects we must process: the set of grid points and the set of control surfaces that define the elevation envelope. A direct approach for processing the triangular control surfaces is to assign each surface to an individual processor. For large grids, this would impose considerable memory requirements, since each processor would need a separate copy of the two-dimensional grid array. Assuming shared memory architectures, an alternate scheme is to divide the ground plane into  $p$  regions, one for each available processor. This can be done by assigning each processor an equal (or nearly equal) number of  $y$  rows. This may not distribute the load equitably among the processors, but without advance knowledge about the distribution of control surfaces over the ground plane, this scheme is probably as good as any other.

Suppose we have an  $(n+1)$  by  $(n+1)$  grid. Each processor can be assigned  $n/p$  distinct rows (or columns) of the grid, with one processor assigned an extra row. An individual processor would then be responsible for calculating control elevations over that portion of each control surface that overlapped the rows assigned to that processor. If both  $n$  and  $p$  are powers of 2, say  $n = 2^m$  and  $p = 2^k$ , we would assign  $2^{m-k}$  rows to each of  $p-1$  processors, and  $2^{m-k}+1$  rows would be allotted to the last processor. Thus, the first processor would be assigned rows 0 through  $2^{m-k}-1$ , the second would get rows  $2^{m-k}$  through  $2^{m-k+1}-1$ , and so forth, with the last processor allotted rows  $2^m-2^{m-k}$  through  $2^m$ . Figure 6 illustrates this assignment scheme with four processors and a 17 by 17 ground-plane grid.

To calculate average elevations for grid midpoint positions, we modify slightly the allocation scheme above. Each processor needs two subdivision boundary rows for these calculations. With  $p = 2^k$  processors and  $2^m+1$  rows and columns in the grid, we now assign  $2^{m-k}+1$  rows to each processor, where the boundary rows for the subdivisions are at indices 0,  $2^{m-k}$ ,  $2^{m-k+1}$ ,  $\dots$ ,  $2^m$ . Rows 0 through  $2^{m-k}$  are assigned to the first processor, rows  $2^{m-k}$  through  $2^{m-k+1}$  go to the second processor, and so on up to the last processor, which is assigned rows  $2^m-2^{m-k}$  through  $2^m$ . This allocation of rows to the processors is illustrated in Fig. 7 for  $m = 4$  (17 by 17 grid) and  $k = 2$  (four processors). Elevations along the subdivision boundary rows are obtained by first assigning control elevations to the four corners of the ground-plane grid. Then elevations along the top and bottom grid boundaries ( $y_0, y_n$ ) are calculated concurrently, using the midpoint methods of Section 3. With elevations along these boundaries established, parallel calculation of the midpoint elevations along the subdivision boundaries can be carried out. Finally, each processor is assigned a pair of

subdivision boundary rows, and the elevations within each subdivision are calculated in parallel by the  $2^k$  processors.

A somewhat faster method for calculating elevation averages is to first assign control elevations along two boundaries of the ground plane, at intervals determined by the number of processors to be used. For example, in Fig. 7 we can assign control elevations to positions 0, 4, 8, 12, and 16 along the left and right edges of the ground plane. Elevations at grid points along each of these rows can then be calculated in parallel by assigning a processor to each row (with one processor assigned two rows). With these boundary rows established, parallel processing for the remaining rows is carried out as described above. The difference between the two approaches is that elevations along the boundary rows are now determined from the control elevations at each end of the row, instead of averaging from the corner elevations.

## 5. Results

The algorithms described in the previous sections were implemented on an Alliant FX/8, a shared memory system that can be configured with up to eight parallel processors (computational elements). Each processor supports vector operations with a vector register of length 32. A vector instruction on this system can be executed two to four times faster than the equivalent sequential operations. Therefore, ground-plane grid dimensions were chosen to be multiples of 32 for the scenes generated on this system.

An example terrain structure is shown in Fig. 8\*. The mountains were formed using a control envelope consisting of two intersecting pyramids (Fig. 3), each containing four triangular surfaces. A 33 by 33 ground plane grid was chosen for this scene, and a scaling factor of 0.1 was used to calculate the variance of the Gaussian distribution (Eq. 11).

The same procedure was used to construct a cloud pattern (Fig. 9)\*and water waves (Fig. 10)\*above a ground plane. A composite picture of mountains, clouds, and water is shown in Fig. 11.\* The components of this scene were rendered and displayed in color on a Pixar Image Computer with Gouraud shading applied to the triangular polygon surfaces. A single light source at the position of the viewer was used to illuminate the Scene.

We can expect the performance of the optimized algorithms to improve as the size of the ground-plane grid increases, since vector lengths are then increased. More time is thus spent in processing compared to the time needed to setup the vector registers and synchronize the parallel processors. Timing runs with sixty four control surfaces were carried out with various grid sizes. The terrain-generating algorithms employing vectorization and concurrency on the Alliant ran approximately 7 to 14 times faster than the corresponding sequential implementation as the grid size increased from 33 by 33 to 1025 by 1025. Times to generate these scenes with the optimized algorithms varied from about 0.2 seconds to 18.2 seconds. These timing runs indicate that real-time generation of such structured terrain scenes is possible with moderate grid sizes.

## 6. Summary

We have presented an algorithm for quickly modeling and displaying common types of natural phenomena: terrain, clouds, and water. This method consists of a random midpoint displacement procedure modified with a specified control surface that allows a designer to structure surface details. The control surface is formed with triangular sections that are used to restrict the elevation displacements above a ground plane.

\* See page C-472 for Figures 8 and 9 and page C-473 for Figures 10 and 11.

## Acknowledgements

This research was supported in part by the National Science Foundation under grant NSF MIP-8410110, by the U.S. Department of Energy under grant DOE DE FG02-85ER25001, and by the US. Air Force under grant AFOSR-F49620-86-C-0136.

We are also indebted to Allan Tuchman and to Peter Shirley for their help and advice during the course of this study.

## References

- [1] Carpenter, L. C., "Computer Rendering of Fractal Curves and Surfaces". SIGGRAPH '80 Proceedings Computer Graphics (July 1980), Vol.14, No. 3, pp. 9-15.
- [2] Falcidieno. B. and C. Pienovi, "Natural Surface Approximation by Constrained Stochastic Interpolation", Computer Aided Design (April 1990), Vol. 22, No. 3, pp. 167-172.
- [3] Fournier, A., D. Fussel, and L. Carpenter, "Computer Rendering of Stochastic Models", Communications of the ACM (June 1982), Vol. 25, No. 6, pp. 371-384.
- [4] Gardner, G. Y., "Simulation of Natural Scenes Using Textured Quadric Surfaces", SIGGRAPH '84 Proceedings, Computer Graphics (July 1984). Vol. 18, No. 3, pp. 11-20.
- [5] Hearn, D. and M. P. Baker, *Computer Graphics*, Prentice-Hall, Englewood Cliffs, New Jersey, 1986.
- [6] Koh, E-K and Hearn, D. D., "Generating and Structuring Terrain Features on Parallel Vector Architectures", Department of Computer Science, Report 1435, University of Illinois at Urbana-Champaign, May 1988.
- [7] Lewis, J. P., "Generalized Stochastic Subdivision", ACM Transactions on Graphics (July 1987). Vol. 6, No. 3, pp. 167-190.
- [8] Miller, G. S. P., "The Definition and Rendering of Terrain Maps", SIGGRAPH '86 Proceedings, Computer Graphics (August 1986). Vol. 20, NO. 4, pp. 39-48.
- [9] Mandelbrot, B. B., *The Fractal Geometry of Nature*, Freeman Press, New York, 1983.
- [10] Mandelbrot, B. B. and J. W. Wallis, "Fractional Brownian Motions, Fractional Noises, and Applications", SIAM Review (1969). Vol. 10, pp. 422-437.
- [11] Schacter, B. J., ed., *Computer Image Generation*, Wiley-Interscience, New York, 1983.
- [12] Szeliski, R. and D. Terzopoulos, "From Splines to Fractals", SIGGRAPH '89 Proceedings, Computer Graphics (July 1989), Vol. 23, No. 3, pp. 51-60.
- [13] Voss, R. F., "Random Fractal Forgeries", *Fundamental Algorithms for Computer Graphics*, ed. R. A. Earnshaw, Springer-Verlag, Berlin, 1985, pp. 805-835.