Frame Buffer Algorithms for Stochastic Models

Alain Fournier
Tom Milligan

Computer Systems Research Institute
Department of Computer Science, University of Toronto
Toronto, Ontario, M5S 1A4

ABSTRACT

Stochastic modelling is a technique that allows shifting the complexity of the picture away from the modelling database while rendering a class of "natural" looking objects. When implementing this technique within traditional rendering systems, however, one has still to deal with a very large number of geometric and display primitives.

We present here a test system where most of the database expansion is done at the display level. A simple display processor, acting directly on the frame buffer, receives sparse 3-D geometric data, generates dense 2-D stochastic arrays, and determine their colour, shade and visibility to write the final image on the frame buffer.

The system allowed us to explore the issues of reentrant stochastic subdivision, priority and visibility determination, quick shading techniques and filtering at the frame buffer level with a bit-slice display processor. Examples of the images produced and performance statistics are given for the system.

RESUME

Le modelage stochastique est une technique qui permet de repousser la complexité de l'image loin des données du modelage tout en représentant une classe d'objets qui apparaissent "naturels". Cependant, quand cette technique est réalisée avec des systèmes de représentation traditionels, on a encore affaire à un très grand nombre de primitives géomètriques et d'affichage.

Nous présentons ici un système expérimental où la plus grande partie de l'expension des données est faite au niveau de l'affichage. Un processeur graphique simple, agissant directement au niveau de la mémoire d'image, reçoit des données géomètriques dispersées en trois dimensions produit des tableaux de valeurs stochastiques denses en deux dimensions, et déterminent leur couleurs, intensités et visibilité pour écrire l'image finale en mémoire.

Le système nous a permis d'explorer les problèmes associés à la subdivision stochastique réentrante, à la détermination de la priorité et de la visibilité, aux techniques rapides de détermination des teintes au niveau de la mémoire d'image avec un processeur d'affichage en tranches. Nous donnons aussi des examples des images produites avec le système, accompagnées de leurs statistiques.

KEYWORDS: stochastic modelling, terrain modelling, frame buffer algorithms, adaptive subdivision.

1. Motivations

The complexity of a picture can be measured by the number of basic elements (or *primitives*) necessary to display it. This complexity can be introduced at the modelling level or delayed until the display level. At the modelling level complexity may manifest itself as a large number of primitives or the use of higher degree surfaces, stochastic models [Four82], or particle systems [Reev83]. Since these are normally broken down into simpler geometric objects such as line segments or polygons, geometric and display operations still have to be performed on a large number of primitives. As an example, the landscape image in Figure 1 is composed of 16 bicubic patches subdivided into about 2000 triangles each. Roughly 30 minutes of a VAX 11/780 processor time were required to model and display the scene.

If we want to postpone the "data base amplification" step as long as possible, algorithms must be developed to perform operations as close to the frame buffer as possi-

ble. Processes like clipping, 3-D to 2-D mapping, filtering, visibility and shading computations should be performed with simple, fast, dedicated processors. An example of this strategy is the well known technique of *texture mapping* [Catm75, Blin78, Will83]. Instead of being described by dense data in 3-D, objects are described by sparse data in 3-D and dense data in 2-D in the form of 2-D texture arrays. Then a complex image is obtained from a sparse database, but even more importantly the geometric transformations have only been applied to sparse data. In the case of texture mapping, the 2-D mapping is facilitated by the fact that the position of the surface is not affected by the mapping, and therefore the shading (except in the case of *bump mapping* [Blin78, Haru84]) and visibility are not modified.

In [Four82] we developed the techniques for stochastic modelling as applied to computer graphics, and in particular the reentrant subdivision techniques that allowed to generate approximations to *fractional Brownian motion*. This process, introduced by Mandelbrot [Mand68, Mand82], permits realistic approximation of earth terrain. In [Pipe84] we described the *STINT*, a hardware board designed to generate a stochastic array of values using our subdivision algorithm. We demonstrated that the subdivision algorithm is simple enough to be implemented easily in hardware, and that such an implementation yields a two order of magnitude improvement in the speed of generation of stochastic data over a general purpose processor. The other important point is that the data is next to the display processor (*i. e.* in its address space) and the speed of the generation of the stochastic data has to be matched to the speed at which this processor can use the data.

This paper will describe a group of techniques to generate complex images in a display processor operating directly on the frame buffer. These techniques include the generation of the stochastic data defining implicitly the complex model, using either the STINT board or a firmware simulation, the mapping of that data to create the screen projection of a stochastic surface, the shading computations, the visible surface determination and possible filtering techniques.

This represents a heavy burden for a processor normally dedicated to more mundane tasks, such as line drawing or area filling. This is further complicated by two types of constraints. In most cases in a graphics system, simple primitives are used to describe the objects and must obey constraints at their boundaries. Then the geometric transformations and the rest of the rendering process operate piecewise on these primitives (such as polygons or parametric patches). Clearly then, the dense data generation and display algorithms which operate "close to the frame buffer" must be prepared to deal with and respect the constraints imposed on them. These can be called the *external coherence* constraints. Since most of the data is generated at the frame buffer level, and only as needed, as the objects appear and disappear, move farther and closer, more or less data is actually generated. We have to make sure that in every circumstance the picture appears to be a correct reconstruction of a virtual object. In a traditional system, the details are permanently stored in the data base, and the problem is only of correctly sampling and filtering it. In our system, we have to be careful to respect these *internal coherence* constraints as we generate the dense data "on the fly".

An intelligent division of labour between a general purpose host processor and one or more special purpose display processors greatly improves the speed of the display process. The host processor does the work which requires complex operations such as 3-D floating point calculations. Special purpose processor(s) perform the simple repetitive tasks typical of display processing based on dense 2-D data. The processors are inexpensive and very fast in their limited repertoire. Such a trend is now fairly obvious, and merely one more turn of the *wheel of reincarnation* [Myer68]. It is interesting in this respect to point out that we have seen in the past processors dedicated to raster operation and display tasks (more recently [Levi84]), geometric tasks [Clar82] and modelling tasks [Pipe84]. We are in this paper exploring the issues dealing with the design of a display processor to which some of the modelling and geometric tasks have been transferred.

2. The Experimental System

We have developed an experimental display system which generates terrain scenes from a very sparse 3-D database consisting of a network of square patches. Each patch is "painted" with a terrain texture generated from an array of stochastic data. The software runs on a VAX 11/780 host processor with an Adage 3000 graphics system. Resident on the Adage are two special purpose processors. One (the STINT) is custom built to generate the stochastic data [Pipe84]. The second is a commercial bit-slice processor which is part of the raster system. For added flexibility, we have also implemented a STINT simulation in microcode, which allows quick exploration of alternatives for the algorithms or the architecture of the STINT. All the microcode has been written using a high level compiler, and we have incorporated macros that allow compilation of the same code into C, to be run on the host processor if necessary.

The terrain scene is initially modelled as a single square patch. This patch is recursively subdivided into smaller squares which are of a suitable size for painting. At the same time the totally invisible subpatches are eliminated, using a ***extent*** based on an estimate of the final height of the fully formed surface. The painter module asks for a dense data array using the stochastic interpolation algorithm and then maps it to the screen space definition of the patch. Colour, height and lighting information are added to the image during the process and clipping and visible surface determination are performed. The entire system runs on the special processors with the exception of the geometric subdivision itself.

3. The Subdivision

The display system passes the single square "world" patch through a perspective projection and determines its size and location in screen coordinates. If it is too large to be processed as an entity it is subdivided into four smaller patches and the process is repeated. The criterion for size is based on the maximum size of the stochastic arrays generated by the system. In the current implementation, it is 129x129. Therefore a patch is subdivided if any of its sides has a "Manhattan" length greater than 129 screen units (pixels). This continues until the subpatches are completely invisible or until they are the appropriate size for painting. Subdivision proceeds in an order which supports the visible surface determination algorithm used by the patch painter, that is the subpatches are generated in a front to back priority ordering, determined from the eye position.

A patch is considered visible if any part of it could appear on the display. This is not a clear cut operation since the planar patches represent only the base area over which the stochastic surface will be drawn (*cf* Figure 2). For this purpose patches are considered to be three dimensional blocks whose vertical dimension is the highest value expected on the stochastic surface. This is similar to the well known technique of "boxing" objects for quick intersection tests, and more specifically to the use of "cheesecake" extents [Kaji83] in ray tracing stochastic surfaces. The height of the extent box is a function of the scaling factor to be used to map the stochastic data, and is deterministically bounded by the size of the entry in the lookup table used for the Gaussian increments and the size of the elements in the STINT array [Pipe84]. Subpatches which are of the right size, but partially outside the window are clipped during the painting process. This means that only a small fraction of the total painting effort will be expended on invisible portions of the patches.

Subdivision performed in this manner guarantees that the work done in data generation and mapping will be closely matched to the size of the final image on the screen. Figure 3 illustrates the subdivision of a patch under a perspective view with a large, but not uncommon, foreshortening. The top of Figure 3 shows the patch boundaries resulting from the subdivision. At the bottom of the figure are the patch boundaries in screen space. Note that the bottom corners have been subdivided less than their neighbours because they have been clipped out.

The modelling and the geometric modules of the graphic system have then to deal only with subpatches. With the sizes used in our implementation (a 512x512 screen and 129x129 stochastic data arrays) this translates into about 20 to 200 geometric primitives (the subpatches) and 1 modelling

primitive (the initial "world" patch) instead of about 50,000 geometric primitives (assuming polygons covering about 5 pixels each, which is conservative for an detailed image). As we will see later, the figures would be even larger if stochastic modelling were not used.

4. Dense Data Generation

The stochastic data is generated using the recursive interpolation algorithm described in [Four82], and whose hardware implementation is described in [Pipe84]. The algorithm generates approximations of fractional Brownian motion. It ensures external or boundary coherence by setting the initial seeds of the pseudo-random number generators and it ensures internal coherence by tying the subsequent seeds to the values of the points used in the computation of the new points. It should be noted that in a software implementation, it is better to tie the seeds to the indices of the point being computed. Then changing parameters, such as the dimension, which influence all the values, will not change totally the macroscopic features of the patch. This will make possible animations where only the dimension changes, for instance.

It must be stressed that purely recursive subdivision is not correct, since the process we want to approximate is not Markovian. This means that the knowledge of the boundary of a subpatch is not sufficient to compute its interior. Enough information has to be carried over from the neighbours to insure the required coherence between levels of interpolation. Figure 4 illustrates the reentrant process. Note that the boundary elements of the new array are not usable, since they cannot be computed from all their neighbours. The elements of the array that can safely be used are the ones at a distance from the boundaries equal to or greater than the distance between the points reentered.

This property will automatically limit the level of subdivision that can be safely used, if we insist on computing the subdivision in only one step from the "world" patch. In fact, since we use a 129x129 array, anything past the 12th or 13th level of subdivision (7 for the "world" patch and 5 or 6 for the subpatch) will introduce noticeable discontinuities between subpatches. There are two ways to overcome that limit: one is to use a bigger "world" patch. This means more memory, but is easily obtained. A 1M byte memory would easily accommodate a 513x513 array of 16 bit elements. This would allow about 16 levels, that is an amplification of 9 decimal orders of magnitude from the original square. The other way is to carry the subdivision in steps, carrying from step to step enough array elements to exactly compute the interpolation of the needed points. This is what the original software does. Here the limit (beside the additional time) is in the precision of the array elements. The values of the elements adjacent at some level will tend to be equal, and the standard deviation will eventually go to 0. So in both cases there is a practical limit to the "amplification" effect. But to keep the limit in perspective, in the present case it means that the fully evolved world has more than 100 million data points (13 levels, corresponding to a 2^{13} linear amplification, or 2^{26} data points).

5. The Display Operations

The challenge of implementing the rendering algorithms in a very simple processor can largely be overcome with a few clever programming techniques. A notable example is the calculation of complex arithmetic functions using inverted lookup into precomputed tables. It is even possible in some cases to avoid the search of the inverted index and move incrementally through it.

It is often expedient to resort to approximation. Careful choice of the approximation leads to results which are good in all but the most extreme cases. For example, the strips used to render the surface are actually not vertical in screen space, due to the perspective projection. Fortunately, the direction can be safely approximated as vertical with a large saving in computation. Otherwise the visibility algorithm would be more complex [Ande82].

The quadrilaterals passed to the painter are modelled as bilinear parametric patches. These patches represent the base area over which the surface will be drawn. Each data point is mapped first to a location in the patch and then to the surface by drawing a vertical strip originating from the patch (*cf*

Figure 2). The length of the strip is determined from the value of the data point and scaled by a factor provided by the perspective system. Only the visible portion of the strip is drawn. It is clipped to fit the display window and then visible surface determination is performed.

Note that **all** data points are mapped to the base patch. In a conventional texture mapping the data array would be sampled to match the actual size of the patch. In this case that is incorrect; the size and shape of the base patch can be quite different from that of the surface which is constructed over it.

The bilinear mapping of the data array to the patch can be viewed as a linear interpolation of the two screen space coordinates of the patch over the data array. This is useful since there is another parameter which must be interpolated over the patch. The length of the vertical strips drawn to represent the surface must be scaled as in any perspective projection by the inverse of the distance from the eye.

To allow simple visible surface determination, the patch is always filled from front to back (*i.e.* from the bottom of the screen up) and the strips are constrained to fit into vertical columns on the screen. It is then sufficient to record the highest point reached thus far in each column. As the fill process proceeds up the screen, only the portions of strips above the highest already seen are visible. It is crucial that this process execute quickly, even though it places strict ordering requirements on the preceding steps. Both patches and fill line traces must be performed in the correct order. Whenever data is mapped to the screen (a patch or a data point) the lowest item must be done first.

The strips are coloured using a simple height to colour lookup table. The first colour mapping scheme we used was designed to match the traditional colours of a geography atlas. Since the pictures illustrating this paper are grey, there was no need to choose them otherwise. We have, however, experimented with more "realistic" colour schemes.

The shading model used is based on Lambert's law:

$$I = I_i \; k \; (\mathbf{L} \cdot \mathbf{N})$$

The intensity of light reflected from a surface is determined by the angle of incidence of the light rays and the surface normal. The calculation involves the normalized dot product of the incident light vector and the surface normal. The surface normal is calculated as the cross product of two tangent vectors. These are approximated using the vertical displacement between adjacent data points and a scale factor related to the interpoint variance. This scaling compensates for the decrease in variance and corresponding lighter shading which accompany increasing depth in the stochastic interpolation.

6. Examples and Performance

The most exhaustive way to test a modelling and display system is to use it to create an animation. We produced a one minute "flyby and look around" animation, where the "world" was expanded several million fold. The dimension (see [Mand82] for the definition of the dimension) was 2.1.

Figures 5 to 7 give an idea of the images produced. Figure 5 is an orthographic projection of the whole world. The original square (4 data points) now contains 512x512 points, that is to say 16 129x129 arrays. The time to compute it was 36 seconds. Figure 6 is a view of the world from the south. One can recognize the islands in the bay, and the point at the tip of the southern cape. The level of subdivision ranges from 9 to 11. The whole scene has 54 subpatches and took 61 seconds to compute. Figure 7 is a view in roughly the same direction but much closer to the cape. Compare the point now to its size in Figure 5. The time to compute that image was 160 seconds, and the level ranges from 9 to 13.

It should be noted that 30 to 40% of the bit-slice time in the figures given is spent simulating the STINT algorithm. For example, in the last picture, out of the 160 seconds, 60 seconds were spent simulating the STINT interpolation, 50 seconds were spent in the 3-D to 2-D mapping, and about 50 seconds in the display operations. The time spent in the host processor was about 2 seconds, totally overlapped with the bit-slice time.

7. Enhancements

There are of course many improvements that can be made to the display system. We already discussed how to expand further the levels of subdivision. Another straightforward improvement is to use a more sophisticated shading model. In order not to pay too high a price in performance, the best is to use the same lookup table techniques already used in other computations. The use of lookup tables for shading in the frame buffer has already been reported [Evan84] and our situation is similar, since we have a normal and we want to compute quickly the intensity. From the differences in elevation between points and from the precision needed, we can establish that 3 to 4 bits for each normal vector component is enough, giving a 512 to 4K entry lookup table for the intensity. Separate specular and diffuse reflection tables would be used. The same inverted lookup techniques already reported can be used to provide more precision.

Although the current system makes no attempt at anti-aliasing, *supersampling* can be used without too much additional effort. The display operations on the current system operate at pixel resolution. We can compute the location, width and length of the vertical strips used to represent the surface at subpixel resolution and make the columns used for visible surface determination of subpixel width. It ~~would~~ not be necessary to generate more data than are already being used. The ratio of data points to patch pixels is currently between one and two. The subpixel strips corresponding to one data point will typically total between one half and one pixel across and may cover portions of more than one pixel both horizontally and vertically. The height of the strips is already computed to the necessary precision.

Figure 8 illustrates a portion of the frame buffer a few pixels across. The display process is using a 4 by 4 subpixel resolution. In this example data points are being mapped to three consecutive strips. The pixel contribution of the strips can be computed 'on the fly' using the information that is already being recorded for visible surface determination. The frame buffer itself can be used for pixel construction. The only additional storage required is that needed to extend visible surface determination to subpixel resolution. The result ~~will be~~ equivalent to supersampling to 4x4 subpixels.

8. Conclusions

The system as it stands permits one more step towards the display of some visually complex objects in a reasonable time. It is also a case study of processor specialization within a display system. We have shown with it most of the advantages of the adaptive subdivision technique to generate surfaces, and some of its limitations for data amplification due to the limited sizes of the arrays and the limited precision of the numbers.

The gain in speed from a traditional modelling and rendering system is about two orders of magnitude, but we are still far from real time. The modelling operations and the geometric transformations represent only 20,000 operations (100 subpatches at 200 per subpatch), which can be easily executed in real time by a 1Mips processor. The display operations, on the other hand, represent up to 8×10^8 microinstructions in the bit-slice. Thus it would take the equivalent of about 500 pairs of 50ns processors to accomplish the same task in real time, if the work is fairly distributed among them.

Our experimental system can only model objects which are suited to a dense 2-D data representation. Traditional dense 3-D modelling is appropriate for a large class of (typically man-made) objects. Techniques must be developed to merge these two classes of objects in a single system. Fortunately this is often quite easy. In a flight simulator, for instance, the separation between terrain and cloud cover on one hand and airfields and buildings on the other is quite clear. Among the problems we will investigate next are the mapping of the stochastic data over surfaces more complex than bilinear patches, and the visibility problems introduced by merging them with polyhedral objects.

Acknowledgements

We gratefully acknowledge the support of the Natural Sciences and Engineering Research Council and of the Communication Research Centre of the Department of Communications.

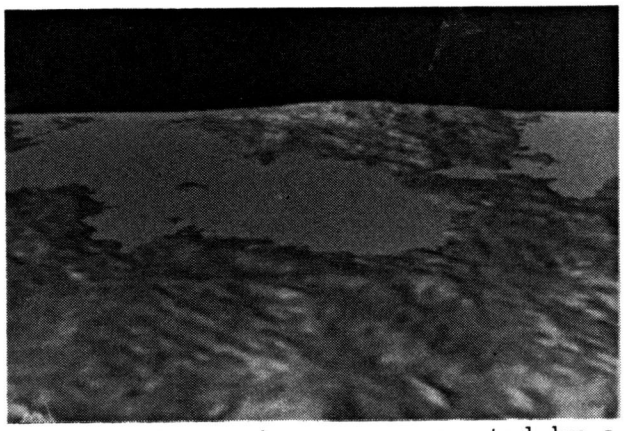

Figure 1. A terrain scene generated by a traditional modelling system. Comprises about 32,000 triangles.

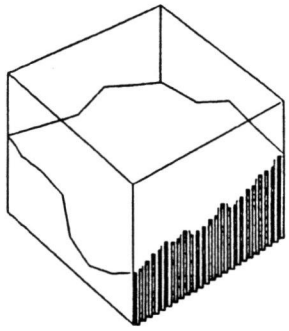

Figure 2. The stochastic surface is drawn over a planar base area using a series of vertical strips. The parallelepiped is the surface extent for the purpose of clipping.

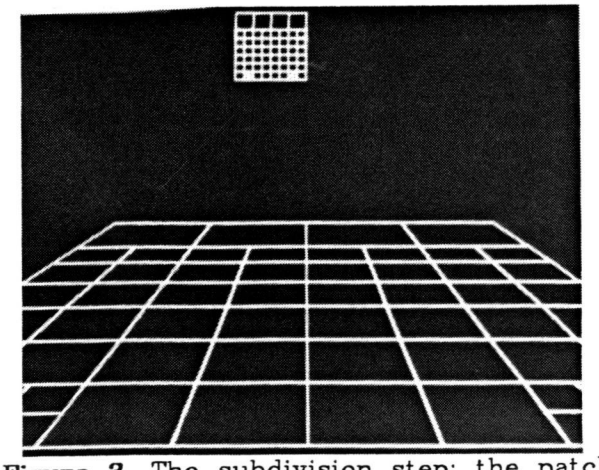

Figure 3. The subdivision step: the patch boundaries resulting from subdivision of the space.

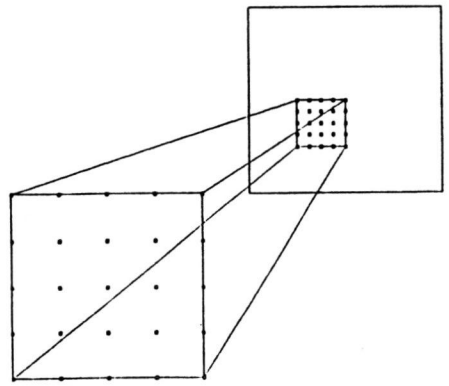

Figure 4. Stochastic interpolator re-entry. All data points available from the initial patch are included as precomputed levels in the new interpolation.

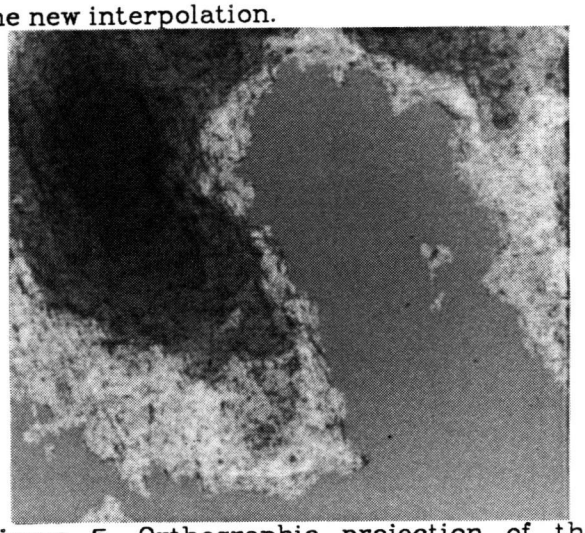

Figure 5. Orthographic projection of the "world" patch. Contains 16 subpatches.

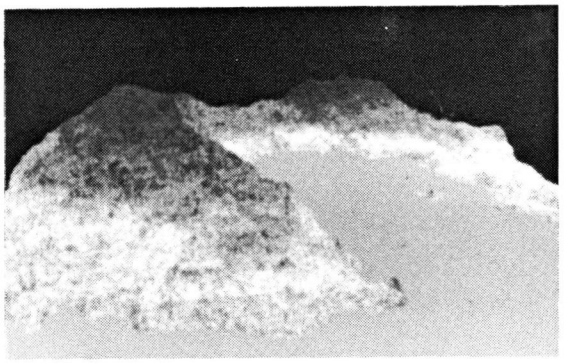

Figure 6. A view from the south. Contains 54 subpatches, and levels from 9 to 11.

Figure 7. A closer view. Note the point in the center of the picture. Contains 120 subpatches, and levels from 9 to 13.

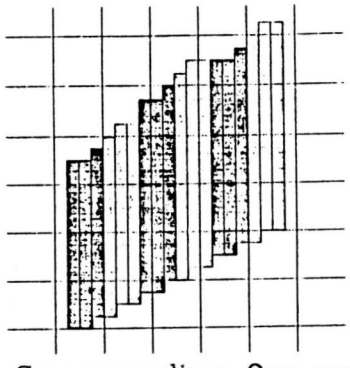

Figure 8. Supersampling: One row of data points mapped to the surface. Subpixel resolution is 4 by 4. Data points mapped to three consecutive strips.

References

[Ande82] Anderson. D. P. "Hidden Line Elimination in Projected Grid Surfaces", ACM Transactions on Graphics, *1*, 4, (October 82), 274-288.

[Blin78] Blinn, J. F. "Simulation of Wrinkled Surfaces", in *Proceedings of SIGGRAPH '78*, also published as Comput. Graphics, *12*, 3, (Aug 1978), 286-292.

[Catm75] Catmull, E., "Computer display of curved surfaces", in *Proc. IEEE Conference on Computer Graphics, Pattern Recognition and Data Structure*. (May 1975).

[Clar82] Clark, J. H. "The Geometry Engine: A VLSI Geometry System for Graphics", in *Proceedings of SIGGRAPH 82*, also published as Comput. Graphics, *16*, 3, (Aug 82), 127-133.

[Evan84] Evans, K. B. "Realtime Lighting Manipulation in Color via Lookup Tables", *Proceedings of Graphics Interface '84*, (May 1984), 173-177.

[Four82] Fournier, A., Fussell, D. and Carpenter, L. "Computer Rendering of Stochastic Models", Communications of the ACM, *25*, 6, (June 1982), 371-384.

[Haru84] Haruyama, S. and Barsky, B. A. "Using Stochastic Modelling for Texture Generation", IEEE Computer Graphics and Applications, *4*, 3, 7-19.

[Kaji83] Kajiya, J. T., "Ray Tracing Procedurally Defined Objects", in *Proceedings of SIGGRAPH 83*, also published as Comput. Graphics, *17*, 3, (July 83), 91-102.

[Levi84] Levinthal A. and Porter, T., "Chap- A SIMD Graphics Processor", in *Proceedings of SIGGRAPH 84*, also published as Comput. Graphics, *18*, 3, 77-82.

[Mand68] Mandelbrot, B. B. and Van Ness, J. W. "Fractional Brownian motions, fractional noises and applications", SIAM Review, *10*, 4, (Oct 1968), 422-437.

[Mand82] Mandelbrot, B. B. *The Fractal Geometry of Nature*. Freeman, (1982).

[Myer68] Myer, T. H. and Sutherland, I. E., "On the Design of Display Processors", Communications of the ACM, *11*, 6, (June 68), 410-414.

[Pipe84] Piper, T. S. and Fournier, A. "A Hardware Stochastic Interpolator for Raster Displays", in *Proceedings of SIGGRAPH 84*, also published as Comput. Graphics, *18*, 3, (July 84), 83-91.

[Reev83] Reeves, W. T., "Particle Systems-A Technique for Modeling a Class of Fuzzy Objects", Transactions on Graphics, *2*, 2, (April 83), 91-108.

[Will83] Williams, L. "Pyramidal Parametrics", in *Proceedings of SIGGRAPH 83*, also published as Comput. Graphics, *17*, 3, (July 83), 1-11.