Specifying Stochastic Objects in a Hierarchical Graphics System

Brian Wyvill, Craig McPheeters, Milan Novacek

Department of Computer Science, University of Calgary.
2500 University Drive N.W.
Calgary, Alberta, Canada, T2N 1N4

Abstract.

Many graphics systems allow a user to describe three dimensional objects with polygon meshes or surface patches. However to achieve realistic scenes for making animated film some objects are better described with stochastic techniques. Some examples would be clouds, fire and mountains. We describe here a hierarchical graphics system consisting of objects which contain geometrical transformations of other objects or primitives. Each object is treated in a consistent fashion whatever the types of primitives that are ultimately called, for example an object may consist of polygon mesh sub-objects and stochastic objects. The system has been designed so that it may easily be extended to include new primitive types, so far a sub-system for generating particles (fire, volcanos etc.) and a sub-system for generating fractal polygons have been implemented Examples are given of the results obtained with this technique.

Overview

In designing a 3D graphics system one of the problems is to decide what classes of primitives will be used for constructing the objects. The end user may wish to define solid objects of a deterministic nature such as a building and also define a cloud which is perhaps best defined in terms of some stochastic function. Graphicsland is a system which has been designed to include different types of primitives and offers a consistent way of handling them. The system stores 3D objects as a hierarchy of geometric transformations, each object may refer to a number of other objects which in turn refer to objects and so on. At the leaves of this tree hierarchy are the primitives which may be geometrical primitives such as polygons or procedures for generating fractal surfaces or particle systems. We have tried to design the system in an extensible way so that other primitives such as beta-spline surfaces may be added later. Figure 1 shows a diagram of the system. It is layered with the data-structure and associated editing and traversal routines as the central core. A user interacts via a control interface and invokes one of a series of specialised user interface modules.

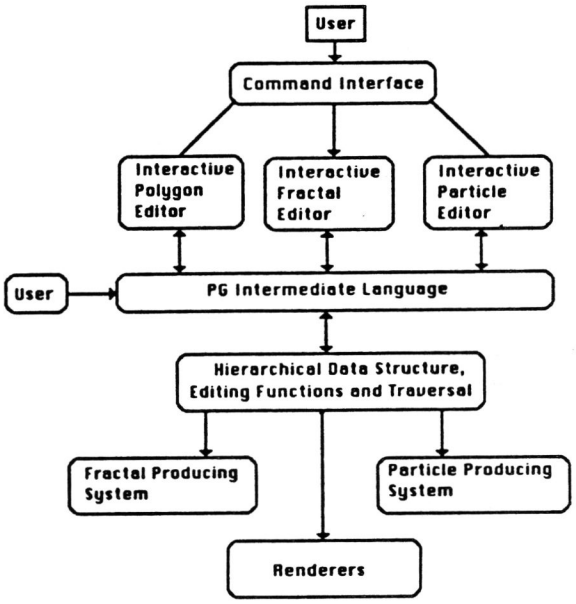


Figure 1 Graphicsland Overview

Each of the user interface modules produces an intermediate command language which is interpreted by the system controlling the object hierarchies. Various graphical editors are being designed, so far only the polygon editor has been implemented with a graphical interface. Currently fractal and particle descriptions can be made via the intermediate command language. The design of this language is such that Graphicsland can be extended by writing special purpose interfaces which will interact with the language layer of the system. For example an interactive fractal mountain package is being written which uses contours drawn by the user. Currently landforms such as mountains may be defined in the intermediate language, by entering specifications for a number of parameters such as height of snow line, density of snow, angle of slope to which the snow will stick, and so on.

This approach also has the advantage that new ideas may be tried out very quickly by implementing commands in the intermediate language. Thus Graphicsland is a research oriented system which provides a test bed for new techniques.

It is useful to compare our approach to other methods. Porter & Duff [Porter 84] describe an image compositor which allows pictures to be composed from a series of images. These images are calculated for a particular viewpoint and can be combined as parts of different pictures reducing the overhead of having to render common parts between like frames. Once the view point is moved the component images would have to be re-rendered. We are providing a facility for composing pictures in object space which has the advantage that a view point can be moved and the correct view rendered. Thus image compositing will have an advantage when there are few camera moves. Object composing has the advantage of working in true 3D as compared to a 2½D approach.

The Hierarchical Data Structure

Figure 2 shows how the object hierarchy is stored as a series of geometric transformations. Each node contains a transformation matrix and a reference to the hierarchy to be transformed. To show a view of the object the structure is traversed as described in [Wyvill 84] in which each new node transforms the current set of coordinates until a primitive is encountered. In the current work the scheme has been generalised in that the polygon primitive has been substituted for a Procedural Image Generator (PIG in the diagram). So far three PIGs have been implemented to generate polygons, particles and fractals. We are currently working on this approach to extend the system to beta-spline surfaces.

The particle sub-system

A particle system based on the work of Reeves described in [Reeves 83] has been implemented with some extensions. For example, Gauss filtering of particles has been included, this has the effect of defocusing the particles which reduces the number required to represent certain special effects. The user has the option of having particles rendered as simple points, or as fuzzy balls. The former case allows very fast rendering on raster devices. Particles are assumed to be pixel sized, and the colour of the affected pixel is simply increased by the particles' own colour intensity. In the latter case a Gauss function is applied as discussed under rendering.

Figure 2 shows the information stored in the object hierarchy node and the PIG node for a particle generator. These specifications differ from Reeves' in that two parameters which control Gauss filtering are associated with each particle generator along with a third parameter which controls the probability that a given particle will become a generator. It has been found that Gauss filtering particles produces some realistic effects with considerably fewer particles than previously used, this is discussed in the section on rendering.

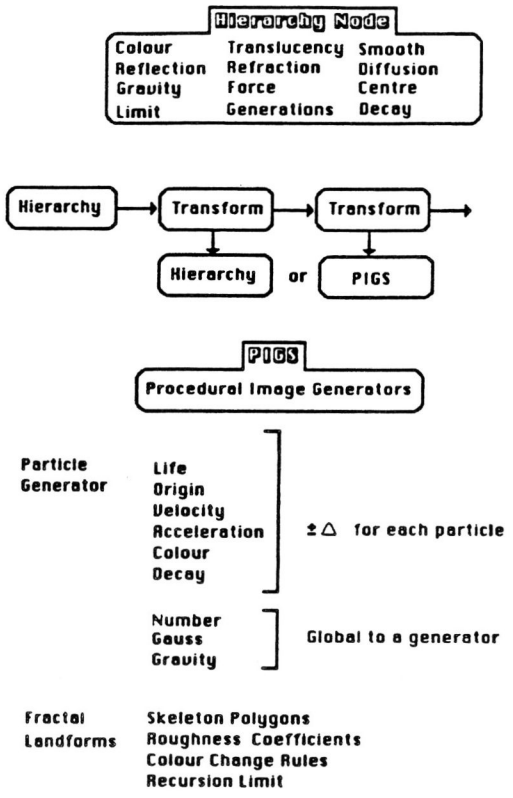


Figure 2 Hierarchical Data Structure

The fractal sub-system

The method for generating fractals in the fractal mountain package is largely based on the work of Mandelbrot [Mandelbrot 83] and Fournier et al [Fournier 82]. A recursive algorithm is employed to subdivide a set of intermediate polygons. After each subdivision the coordinates of the vertices are modified by a random value constrained by a roughness factor. The intermediate polygons determine the gross overall shape of the mountain. They may have been generated by hand or by any other suitable method, for example by a preprocessing program that accepts a set of contour polygons and converts these to a set of tiles covering the requisite surface.

Such a scheme need not be excessively restrictive nor need it require a detailed contour specification in order to yield a pleasing result. The mountains in our examples have each been generated from a single input polygon. A mountain range is simply a number of instances of the one mountain with different geometric transformations applied.

The controlling parameters are summarised in Figure 2. Output from this package is a polygon list in the intermediate command language.

Rendering

The rendering of polygons is well understood, but special attention should be paid to particles. Unlike Reeves system we assume that particles do interact with other surface-based modeling primitives. We have implemented two methods of rendering, Z-buffer and Ray Tracing. In both cases the particles are fully integrated into the algorithm as a primitive.

Several options are available for the interaction of particles with other primitives, for example particles may may act as luminous entities and be added into the image (as in Reeves' system). A weighted replacement option is also available. Note that particles are not proper light sources in that they do not illuminate other objects.

Particles may also be be single points or fuzzy balls, as noted in the particle subsystem description. In the latter case, fuzzy balls are particles with a radius of influence (r) assigned by the user at the definition stage. At the rendering stage a particle is treated as a sphere of radius r, with varying translucency. The edge of the sphere is nearly perfectly translucent, contributing almost nothing to the image, while at the centre the full colour intensity of the particle is added to the pixel value. A negative exponential function is used to provide a smooth transition between the two extremes:

$$f = e^{-\frac{d^2}{\sigma^2}}$$

This gives the fraction of the intensity to add to the image at a distance d from the centre of the particle.

$$\sigma^2$$

is proportional to the radius of influence, r.

In the z-buffer algorithm we first render all polygons, and then do the particles. Particle intensities are added into the frame buffer according to f. In the case of non-transparent particles a weighted partial replacement is done, with the replacement weight again given by f.

Work on raytracing is still in the experimental stage, but in general particles are treated in a similar manner as spheres, and the simple point-particles discussed above are simply assigned a small radius.

Examples

A few examples are shown of combined fractal, particle and polygonal figures. Slides Castle and Martian show two particle systems combined with some objects built from polygons. Slide Road shows some high quality fractal mountains consisting of 128k polygons for each of six mountains. The road and fields are defined by plain polygons. Slide Saturn shows two types of particle rendering, the Gauss filter has been used for the super nova and point rendering for the rings of Saturn. The planet is in fact from polygons. The first two slides represent frames from a short film sequence currently under production to show the application of these techniques to animation.

Conclusion

We have presented a way of combining some different primitives in a 3D animation system. These include particles, fractals and polygons. We are planning to expand this set and to implement new user interfaces to allow animators easy access to these facilities.

Acknowledgements

The JADE project at the University of Calgary has been particularly supportive of our work in distributed graphics. This work and JADE is supported by the Natural Science and Engineering Research Council of Canada. Special mention is also given to Andrew Pearce, who not only built the castle, but also made it explode.

References

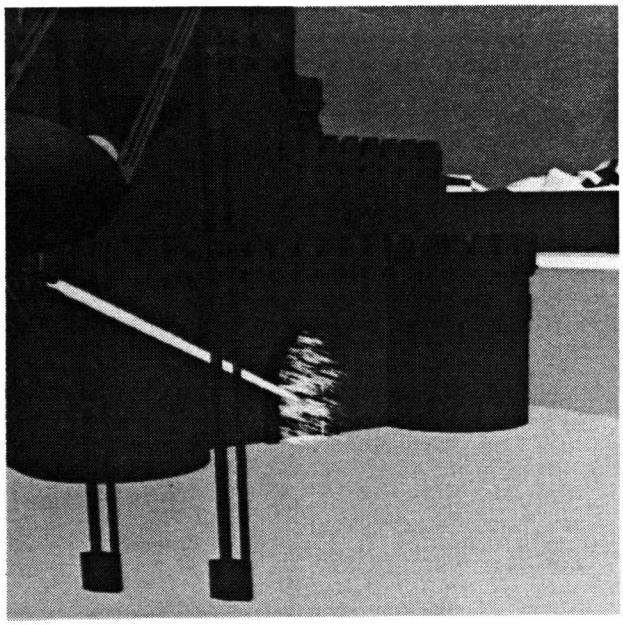
Fournier, A, Fussel, D. and Carpenter, L (June 1982) "Computer Rendering of Stochastic models" Commun. ACM, 25 6, 371-384.

Mandelbrot, Benoit (1983) "The fractal geometry of nature" W.H. Freeman and company.

Porter, T. and Duff, T (July 1984) "Compositing Digital Images" Proc. ACM SIGGRAPH '84, pp. 253-259

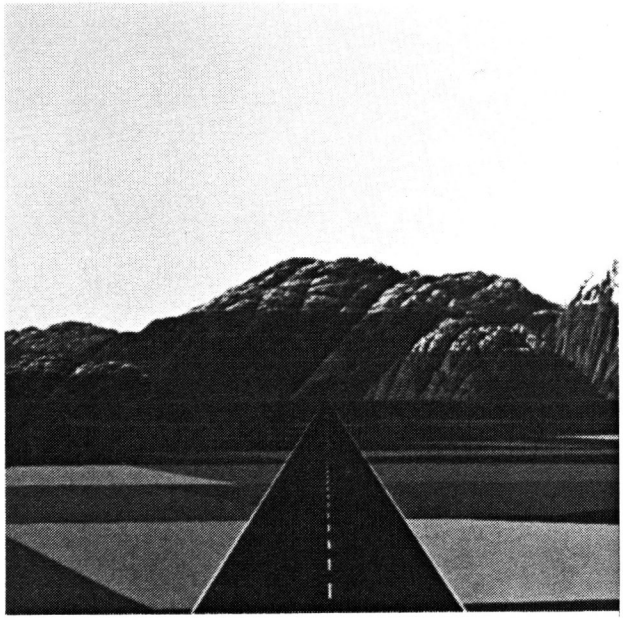
Reeves, William. (Apr 1983) "Particle systems- A technique for modeling a class of fuzzy objects" ACM Transactions on Graphics, 2, 91-108.

Wyvill, B.L.M., Liblong, B, and Hutchinson, N. (June 1984) "Using Recursion to Describe Polygonal Surfaces" Proc. Graphics Interface 84, Ottowa.

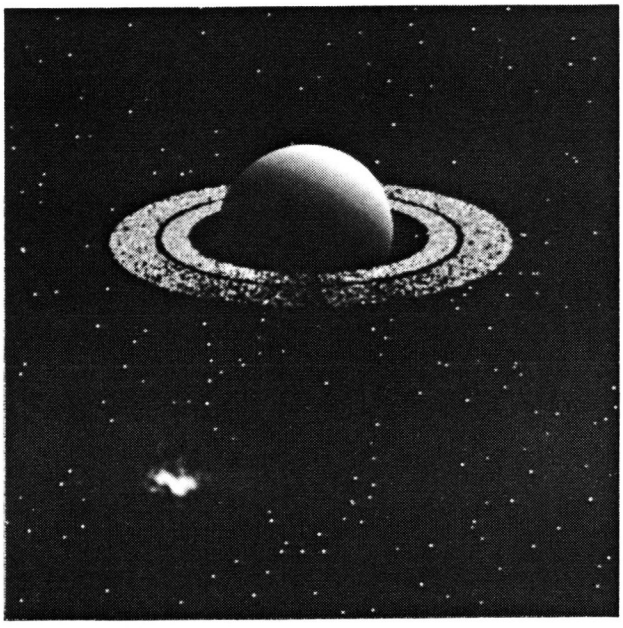


1. Castle

2. Martian



3. Road



4. Saturn

Please note considerable contrast has been
lost on transferring from colour to black and white.