Particle Systems—A Technique for Modeling a Class of Fuzzy Objects

WILLIAM T. REEVES
Lucasfilm Ltd

This paper introduces particle systems—a method for modeling fuzzy objects such as fire, clouds, and water. Particle systems model an object as a cloud of primitive particles that define its volume. Over a period of time, particles are generated into the system, move and change form within the system, and die from the system. The resulting model is able to represent motion, changes of form, and dynamics that are not possible with classical surface-based representations. The particles can easily be motion blurred, and therefore do not exhibit temporal aliasing or strobing. Stochastic processes are used to generate and control the many particles within a particle system. The application of particle systems to the wall of fire element from the Genesis Demo sequence of the film *Star Trek II: The Wrath of Khan* [10] is presented.

Categories and Subject Descriptors: I.3.3 [**Computer Graphics**]: Picture/Image Generation; I.3.5 [**Computer Graphics**]: Computational Geometry and Object Modeling; I.3.7 [**Computer Graphics**]: Three-Dimensional Graphics and Realism

General Terms: Algorithms, Design

Additional Key Words and Phrases: Motion blur, stochastic modeling, temporal aliasing, dynamic objects

1. INTRODUCTION

Modeling phenomena such as clouds, smoke, water, and fire has proved difficult with the existing techniques of computer image synthesis. These "fuzzy" objects do not have smooth, well-defined, and shiny surfaces; instead their surfaces are irregular, complex, and ill defined. We are interested in their dynamic and fluid changes in shape and appearance. They are not rigid objects nor can their motions be described by the simple affine transformations that are common in computer graphics.

This paper presents a method for the modeling of fuzzy objects that we call particle systems. The representation of particle systems differs in three basic ways from representations normally used in image synthesis. First, an object is represented not by a set of primitive surface elements, such as polygons or patches, that define its boundary, but as clouds of primitive particles that define its volume. Second, a particle system is not a static entity. Its particles change form and move with the passage of time. New particles are "born" and old

Author's address: William T. Reeves, Lucasfilm Ltd, P.O. Box 2009, San Rafael, CA 94912.

Reprinted From **acm Transactions On Graphics**—April 1983—Vol. 2, No. 2

Reprinted with permission from *Computer Graphics* Vol. 17, No. 3, July 1983, 359-376.

particles "die." Third, an object represented by a particle system is not deterministic, since its shape and form are not completely specified. Instead, stochastic processes are used to create and change an object's shape and appearance.

In modeling fuzzy objects, the particle system approach has several important advantages over classical surface-oriented techniques. First, a particle (for now, think of a particle as a point in three-dimensional space) is a much simpler primitive than a polygon, the simplest of the surface representations. Therefore, in the same amount of computation time one can process more of the basic primitives and produce a more complex image. Because a particle is simple, it is also easy to motion-blur. Motion-blurring of fast-moving objects for the removal of temporal aliasing effects has been largely ignored in computer image synthesis to date. A second advantage is that the model definition is procedural and is controlled by random numbers. Therefore, obtaining a highly detailed model does not necessarily require a great deal of human design time as is often the case with existing surface-based systems. Because it is procedural, a particle system can adjust its level of detail to suit a specific set of viewing parameters. As with fractal surfaces [5], zooming in on a particle system can reveal more and more detail. Third, particle systems model objects that are "alive," that is, they change form over a period of time. It is difficult to represent complex dynamics of this form with surface-based modeling techniques.

Modeling objects as collections of particles is not a new idea. Fifteen years ago, the earliest computer video games depicted exploding spaceships with many little glowing dots that filled the screen. Point sources have been used as a graphics data type in many three-dimensional modeling systems (e.g., the early Evans and Sutherland flight simulators), although there are few real references to them in the literature. Roger Wilson at Ohio State [4] used particles to model smoke emerging from a smokestack. There were neither stochastic controls nor dynamics in his model. Alvy Ray Smith and Jim Blinn used particles to model star creation and death in galaxies for the Cosmos series [11]. Alan Norton [9] used particles to generate and display three-dimensional fractal shapes. Jim Blinn [3] discussed light reflection functions for simulating light passing through and being reflected by layers of particles. His technique was used to produce images of the rings of Saturn. Blinn did not address the fuzzy object modeling problem which is the topic of this paper. Volumetric representations have also been proposed as viable alternatives to surface representations. Solid modeling [13] is a form of volumetric representation, as is the work of Norm Badler and Joe O'Rourke on "bubbleman" [2]. The use of stochastic modeling relates our work to the recent advances in fractal modeling [5].

Section 2 decribes the basic framework of particle systems in more detail. Section 3 examines how particle systems were used to produce the fire element in the Genesis Demo sequence from the movie *Star Trek II: The Wrath of Khan* [10]. Section 4 presents several other applications of particle systems, and Section 5 discusses ongoing and future research in this area.

2. BASIC MODEL OF PARTICLE SYSTEMS

A particle system is a collection of many minute particles that together represent a fuzzy object. Over a period of time, particles are generated into a system, move and change from within the system, and die from the system.

Reprinted From **acm Transactions On Graphics**—April 1983—Vol. 2, No. 2

To compute each frame in a motion sequence, the following sequence of steps is performed: (1) new particles are generated into the system, (2) each new particle is assigned its individual attributes, (3) any particles that have existed within the system past their prescribed lifetime are extinguished, (4) the remaining particles are moved and transformed according to their dynamic attributes, and finally (5) an image of the living particles is rendered in a frame buffer. The particle system can be programmed to execute any set of instructions at each step. Because it is procedural, this approach can incorporate any computational model that describes the appearance or dynamics of the object. For example, the motions and transformations of particles could be tied to the solution of a system of partial differential equations, or particle attributes could be assigned on the basis of statistical mechanics. We can, therefore, take advantage of models which have been developed in other scientific or engineering disciplines.

In the research presented here, we use simple stochastic processes as the procedural elements of each step in the generation of a frame. To control the shape, appearance, and dynamics of the particles within a particle system, the model designer has access to a set of parameters. Stochastic processes that randomly select each particle's appearance and movement are constrained by these parameters. In general, each parameter specifies a range in which a particle's value must lie. Normally, a range is specified by providing its mean value and its maximum variance.

The following subsections describe in more detail the basic model for particle systems, and how they are controlled and specified within the software we have written.

2.1 Particle Generation

Particles are generated into a particle system by means of controlled stochastic processes. One process determines the number of particles entering the system during each interval of time, that is, at a given frame. The number of particles generated is important because it strongly influences the density of the fuzzy object.

The model designer can choose to control the number of new particles in one of two ways. In the first method, the designer controls the mean number of particles generated at a frame and its variance. The actual number of particles generated at frame f is

$$NParts_f = MeanParts_f + Rand() \times VarParts_f,$$

where *Rand* is a procedure returning a uniformly distributed random number between -1.0 and $+1.0$, $MeanParts_f$ the mean number of particles, and $VarParts_f$ its variance.

In the second method, the number of new particles depends on the screen size of the object. The model designer controls the mean number of particles generated per unit of screen area and its variance. The procedural particle system can determine the view parameters at a particular frame, calculate the approximate screen area that it covers, and set the number of new particles accordingly. The corresponding equation is

$$NParts_f = (MeanParts_{sa_f} + Rand() \times VarParts_{sa_f}) \times ScreenArea,$$

where $MeanParts_{sa}$ is the mean per screen area, $VarParts_{sa_f}$ its variance, and *ScreenArea* the particle system's screen area. This method controls the level of detail of the particle system and, therefore, the time required to render its image. For example, there is no need to generate 100,000 particles in an object that covers 4 pixels on the screen.

To enable a particle system to grow or shrink in intensity, the designer is able to vary over time the mean number of particles generated per frame (i.e., $MeanParts_f$ and $MeanParts_{sa_f}$ are, as used above, functions of frame number). Currently, we use a simple linear function

$$MeanParts_f = InitialMeanParts + DeltaMeanParts \times (f - f_0)$$

or

$$MeanParts_{sa_f} = InitialMeanParts_{sa} + DeltaMeanParts_{sa} \times (f - f_0),$$

where f is the current frame, f_0 the first frame during which the particle system is alive, *InitialMeanParts* the mean number of particles at this first frame, and *DeltaMeanParts* its rate of change. The variance controls, $VarParts_f$ and $VarParts_{sa_f}$, are currently constant over all frames. More sophisticated quadratic, cubic, or perhaps even stochastic variations in both the mean and variance parameters would be easy to add.

To control the particle generation of a particle system, therefore, the designer specifies f_0 and either the parameters *InitialMeanParts*, *DeltaMeanParts*, and *VarParts*, or the parameters $InitialMeanParts_{sa}$, $DeltaMeanParts_{sa}$, and $VarParts_{sa}$.

2.2 Particle Attributes

For each new particle generated, the particle system must determine values for the following attributes:

(1) initial position,
(2) initial velocity (both speed and direction),
(3) initial size,
(4) initial color,
(5) initial transparency,
(6) shape,
(7) lifetime.

Several parameters of a particle system control the initial position of its particles. A particle system has a position in three-dimensional space that defines its origin. Two angles of rotation about a coordinate system through this origin give it an orientation. A particle system also has a *generation shape* which defines a region about its origin into which newly born particles are randomly placed. Among the generation shapes we have implemented are: a sphere of radius r, a circle of radius r in the x–y plane of its coordinate system, and a rectangle of length l and width w in the x–y plane of its coordinate system. Figure 1 shows a typical particle system with a spherical generation shape. More complicated generation shapes based on the laws of nature or on chaotic attractors [1] have been envisioned but not yet implemented.

Reprinted From **acm Transactions On Graphics**—April 1983—Vol. 2, No. 2

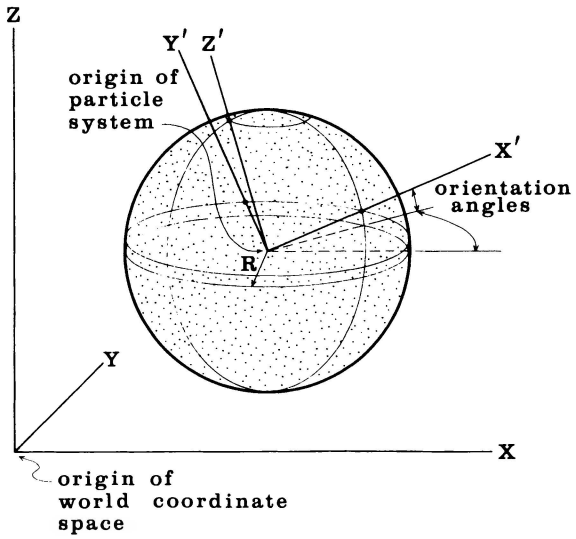

Fig. 1. Typical particle system with spherical generation shape.

The generation shape of a particle system also describes the initial direction in which new particles move. In a spherical generation shape, particles move outward away from the origin of the particle system. In a circular or rectangular shape, particles move upward from the x–y plane, but are allowed to vary from the vertical according to an "ejection" angle, which is another parameter (see Figure 3). The initial speed of a particle is determined by

$$InitialSpeed = MeanSpeed + Rand() \times VarSpeed,$$

where *MeanSpeed* and *VarSpeed* are two other parameters of the particle system, the mean speed and its variance.

To determine a particle's initial color, a particle system is given an average color,[1] and the maximum deviation from that color. Particle transparency and particle size are also determined by mean values and maximum variations. The equations are similar to the one given above for initial speed.

A particle system has a parameter that specifies the shape of each of the particles it generates. The particle shapes implemented so far are spherical, rectangular, and streaked spherical. The latter is used to motion-blur particles—a very important feature when modeling fast-moving objects. We discuss streaking particles in more detail in Sections 2.5 and 3.

The number of possible attribute control parameters and their variants is endless. We are presenting those that we have found to be most useful and interesting.

2.3 Particle Dynamics

Individual particles within a particle system move in three-dimensional space and also change over time in color, transparency, and size.

[1] In more detail, average red, green, and blue values are specified.

To move a particle from one frame to the next is a simple matter of adding its velocity vector to its position vector. To add more complexity, a particle system also uses an acceleration factor to modify the velocity of its particles from frame to frame. With this parameter the model designer can simulate gravity and cause particles to move in parabolic arcs rather than in straight lines.

A particle's color changes over time as prescribed by the rate-of-color-change parameter. The transparency and size of particles are controlled in exactly the same way. In our implementation, these rates of change are global for all particles in a particle system, but one can easily imagine making this parameter stochastic too.

2.4 Particle Extinction

When it is generated, a particle is given a lifetime measured in frames. As each frame is computed, this lifetime is decremented. A particle is killed when its lifetime reaches zero.

Other mechanisms, if enabled, arrange for particles to be killed because they can contribute nothing to the image. If the intensity of a particle, calculated from its color and transparency, drops below a specified threshold, the particle is killed. A particle that moves more than a given distance in a given direction from the origin of its parent particle system may also be killed. This mechanism can be used to clip away particles that stray outside a region of interest.[2]

2.5 Particle Rendering

Once the position and appearance parameters of all particles have been calculated for a frame, the rendering algorithm makes a picture. The general particle-rendering problem is as complicated as the rendering of objects composed of the more common graphical primitives, such as polygons and curved surfaces. Particles can obscure other particles that are behind them in screen depth. They can be transparent and can cast shadows on other particles. Furthermore, particles can coexist in a scene with objects modeled by surface-based primitives, and these objects can intersect with the particles.

In our existing system, two assumptions allow us to simplify the rendering algorithm. First, we assume that particle systems do not intersect with other surface-based modeling primitives, and hence our rendering algorithm need only handle particles. Objects modeled using other techniques are composited together with particle system objects in a postrendering compositing stage. In order for a particle system to intersect or be behind other objects, the rendering system will split the image of a particle system into subimages based on clipping planes defined in the model coordinate space. These subimages are then combined with other images in the compositing stage.

The other simplifying assumption made in our current rendering system is that each particle can be displayed as a point light source.[3] With this assumption,

[2] Note that this clipping is performed in modeling space—to a given plane for example. Clipping to the viewing frustum occurs later in the rendering stage and is discussed below.

[3] Explosions and fire, the two fuzzy objects we have worked with the most, are modeled well with this assumption. Other fuzzy objects, such as clouds and water, are not. Section 5 discusses rendering algorithms for these objects.

Reprinted From **acm Transactions On Graphics**—April 1983—Vol. 2, No. 2

determining hidden surfaces is no longer a problem. Each particle adds a bit of light to the pixels that it covers. A particle behind another particle is not obscured but rather adds more light to the pixels covered. The amount of light added, and its color depend on the particle's transparency and color. Currently, the amount of light added does not depend on the distance between the particle and the viewing position, but that would be an easy extension. The viewing transformation, the particle's size, and its shape determine which pixels are covered. All particle shapes are drawn antialiased in order to prevent temporal aliasing and strobing. Light may be added to a pixel by many particles, so the rendering algorithm clamps the individual red, green, and blue intensities at the maximum intensity value of the frame buffer instead of letting them wrap around.

With this algorithm and assumptions, no sorting of the particles is needed. They are rendered into the frame buffer in whatever order they are generated. Shadows are no longer a problem, since particles do not reflect but emit light.

2.6 Particle Hierarchy

Our system has a mechanism that supports the formation and control of particle system hierarchies. The model designer creates a particle system in which the particles are themselves particle systems. When the parent particle system is transformed, so are all of its descendant particle systems and their particles. The parent particle system's mean color and its variance are used to select the mean color and variance of the offspring particle systems using the same equations presented earlier. The number of new particle systems generated at a frame is based on the parent's particle generation rate. The other parameters of the parent similarly affect those of its children. The data structure used to represent the hierarchy is a tree.

A hierarchy can be used to exert global control on a complicated fuzzy object that is composed of many particle systems. For example, a cloud might be composed of many particle systems, each representing a billowing region of water particles. A parent particle system could group these all together and control the cloud's global movement and appearance as influenced by the wind and terrain.

3. USING PARTICLE SYSTEMS TO MODEL A WALL OF FIRE AND EXPLOSIONS

The Genesis Demo sequence [14] from the movie *Star Trek II: The Wrath of Khan* [10] was generated by the Computer Graphics project of Lucasfilm Ltd. The sequence depicts the transformation of a dead, moonlike planet into a warm, earthlike planet by an experimental device called the Genesis bomb. In a computer-simulated demonstration, the bomb hits the planet's surface and an expanding wall of fire spreads out from the point of impact to eventually "cleanse" the entire planet. The planet's surface begins to buckle, mountains grow, and oceans, vegetation, and an atmosphere form to produce an earthlike environment.

The wall-of-fire element in the Genesis Demo was generated using a two-level hierarchy of particle systems. The top-level system was centered at the impact point of the genesis bomb. It generated particles which were themselves particle systems. Figure 2 illustrates the positions of these second-level particle systems and how they formed expanding concentric rings on the surface of the planet.

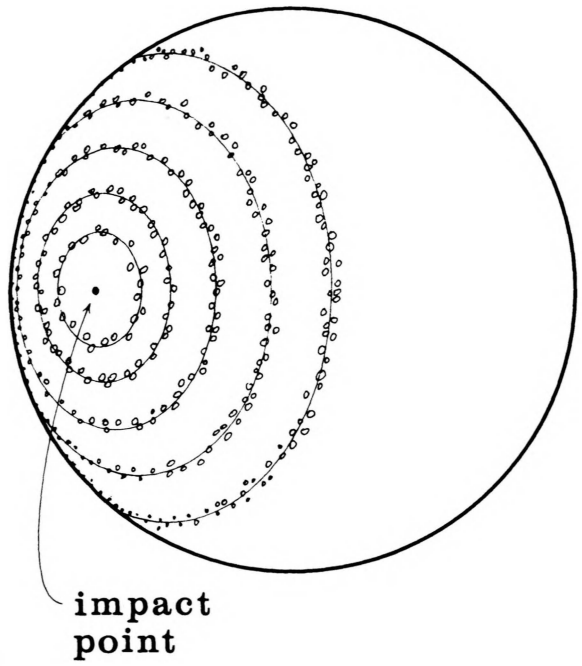

Fig. 2. Distribution of particle systems on the planet's surface.

The number of new particle systems generated in each ring was based on the circumference of the ring and a controlling density parameter. New particle systems were spaced randomly around the ring. Particle systems overlapping others in the same or adjacent rings gave the ring a solid, continuous look.

The second-level particle systems began generating particles at varying times on the basis of their distance from the point of impact. By varying the starting times of the particle systems, the effect of an expanding wall of fire was produced.

The second-level particle systems were modeled to look like explosions. Figure 3 shows an example. The generation shape was a circle on the surface of the planet. Each particle system was oriented so that particles, generated at random positions within the circle, flew upward away from the planet's surface. The initial direction of the particles' movement was constrained by the system's ejection angle to fall within the region bounded by the inverted cone shown in Figure 3. As particles flew upward, the gravity parameter pulled them back down to the planet's surface, giving them a parabolic motion path. The number of particles generated per frame was based on the amount of screen area covered by the particle system.

The individual particle systems were not identical. Their average color and the rates at which the colors changed were inherited from the parent particle system, but varied stochastically. The initial mean velocity, generation circle radius, ejection angle, mean particle size, mean lifetime, mean particle generation rate, and mean particle transparency parameters were also based on their parent's

Reprinted From **acm Transactions On Graphics**—April 1983—Vol. 2, No. 2

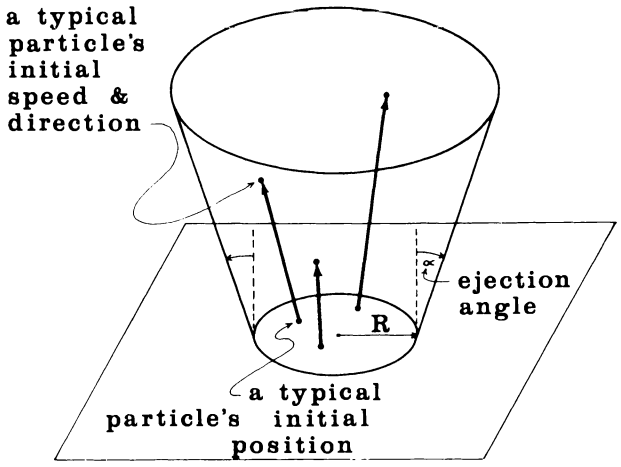

Fig. 3. Form of an explosion-like particle system.

parameters, but varied stochastically. Varying the mean velocity parameter caused the explosions to be of different heights.

All particles generated by the second-level particle systems were predominately red in color with a touch of green. Recall from Section 2.5 that particles are treated as point light sources and that colors are added, not matted, into a pixel. When many particles covered a pixel, as was the case near the center and base of each explosion, the red component was quickly clamped at full intensity and the green component increased to a point where the resulting color was orange and even yellow. Thus, the heart of the explosion had a hot yellow-orange glow which faded off to shades of red elsewhere. Actually, a small blue component caused pixels covered by very many particles to appear white. The rate at which a particle's color changed simulated the cooling of a glowing piece of some hypothetical material. The green and blue components dropped off quickly, and the red followed at a slower rate. Particles were killed when their lifetime expired, when their intensity fell below the minimum intensity parameter, or if they happened to fall below the surface of the planet.

A quickly moving object leaves a blurred image on the retina of the human eye. When a motion picture camera is used to film live action at 24 frames per second, the camera shutter typically remains open for 1/50 of a second. The image captured on a frame is actually an integration of approximately half the motion that occurred between successive frames. An object moving quickly appears blurred in the individual still frames. Computer animation has traditionally imaged scenes as individual instants in time and has ignored motion blur. The resulting motion often exhibits temporal aliasing and strobing effects that are disconcerting to the human eye. Motion blur is a complex topic that is beginning to appear in the literature [7, 12].

The particles in our wall-of-fire element are motion-blurred. Three-dimensional positions are calculated for a particle at the beginning of a frame and about halfway through the frame, and an antialiased straight line is drawn between the

corresponding screen coordinate positions in the frame buffer.[4] Antialiased lines are used to prevent staircasing (moving jaggies) and strobing (popping on and off) effects. To be perfectly correct, screen motion due to movement of the camera should be considered when calculating where to blur a particle. One can also argue that simulating the imperfect temporal sampling of a movie camera is not ideal and that motion blur should really simulate what happens in the human eye. This is a good area for future research.

In the finished sequence, the wall of fire spread over the surface of the planet both in front of and behind the planet's limb (outer edge). The rendering algorithm generated two images per frame—one for all particles between the camera's position and the silhouette plane of the planet, and one for all particles on the other side of this clipping plane. These two elements were composited with the barren moonlike planet element and the stars element in back-to-front order—stars, background fires, planet, and foreground fires.

Because the wall of fire was modeled using many small light-emitting particles, light from the fire should have reflected off the planet's surface. Our current implementation of particle systems does not handle light reflection on surface-based objects. To achieve this effect, Lucasfilm team member Tom Duff added an additional strong local light source above the center of the rings of fire when he rendered the planet's surface. This produced the glow that circles the ring of fire on the planet's surface. (This glow is visible in Figure 5.)

Figure 4 is a frame showing the initial impact of the Genesis bomb. It was generated from one very large particle system and about 20 smaller ones about its base. About 25,000 particles exist in this image. Figure 5 occurs partway through the first half of the sequence. It contains about 200 particle systems and 75,000 particles. Figure 6 shows the ring of fire extending over and beyond the limb of the planet. It is formed from about 200 explosions and 85,000 particles. Figure 7 shows the wall of fire just before it engulfs the camera; in Figure 8 the camera is completely engulfed. Both employ about 400 particle systems and contain over 750,000 particles. The textures in Figure 8 are completely synthetic and yet have a "natural" and highly detailed appearance that is uncommon in most computer graphics images. These images are interesting statically, but they only really come alive on the movie screen. It is interesting to note that this is also the case for many of the best traditional (i.e., non-computer-generated) special effects shots where motion blur is an important factor.

A few points concerning random numbers are of interest from a production point of view. The random number routine we use is based on [6], and generates numbers uniformly in the range [0.0, 1.0]. It is an incremental algorithm based on updating a table of seed values. To checkpoint a production, all that need be saved is this random number table—we do not save all the parameters of 750,000 particles. To restart a computation at frame n, the closest preceding frame p is found that cannot contribute particles to frame n (this is determined from the lifetime parameters of all the active particle systems). Frame $p + 1$'s random number table is then read, and particle generation can begin from there. No

[4] A particle's trajectory is actually parabolic, but the straight-line approximation has so far proved sufficient.

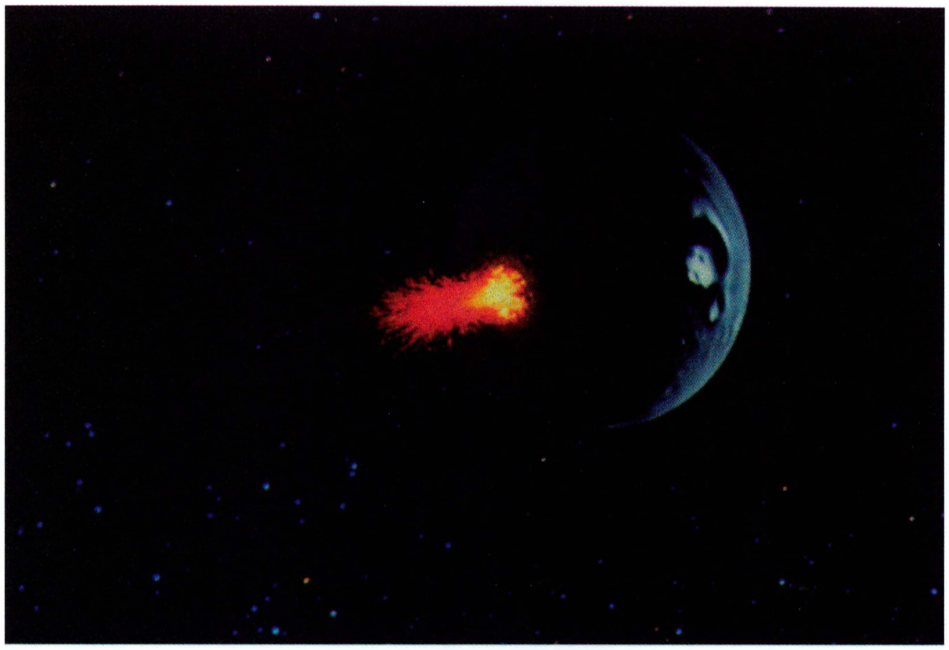

Fig. 4. Initial explosion.

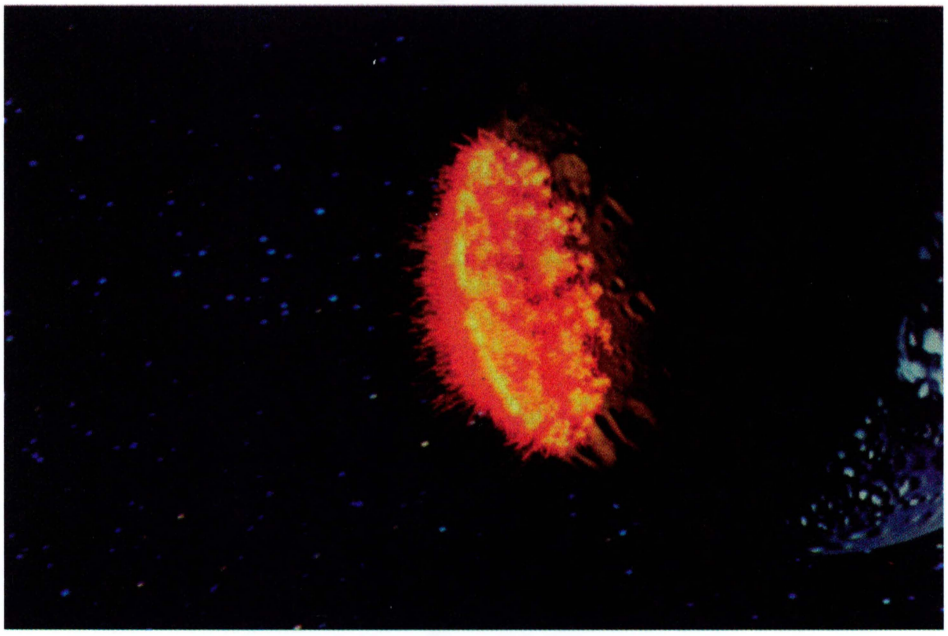

Fig. 5. Expanding wall of fire.

Reprinted From **acm Transactions On Graphics** — April 1983 — Vol. 2, No. 2

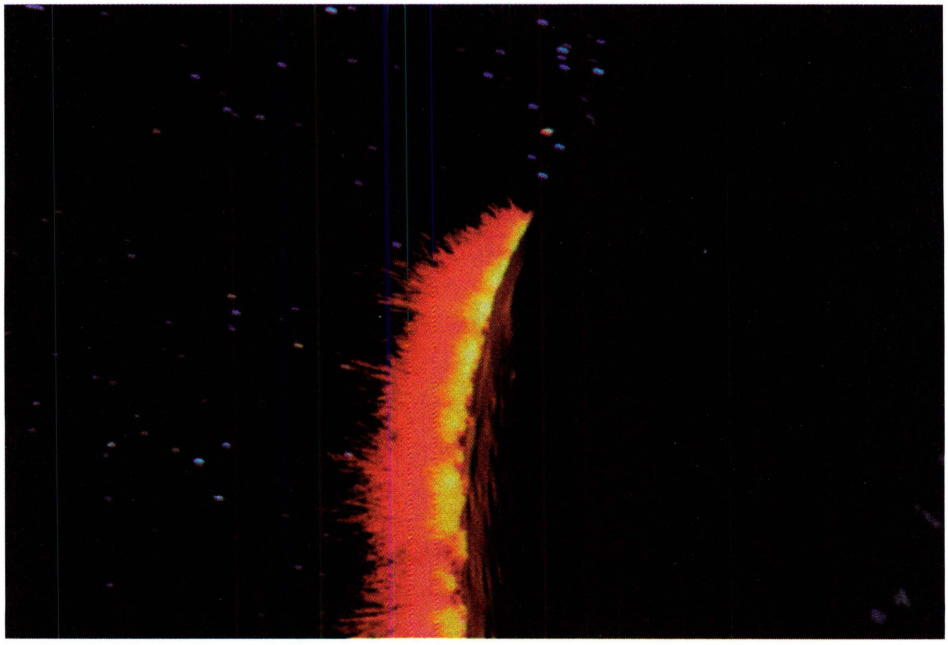

Fig. 6. Wall of fire over limb of planet.

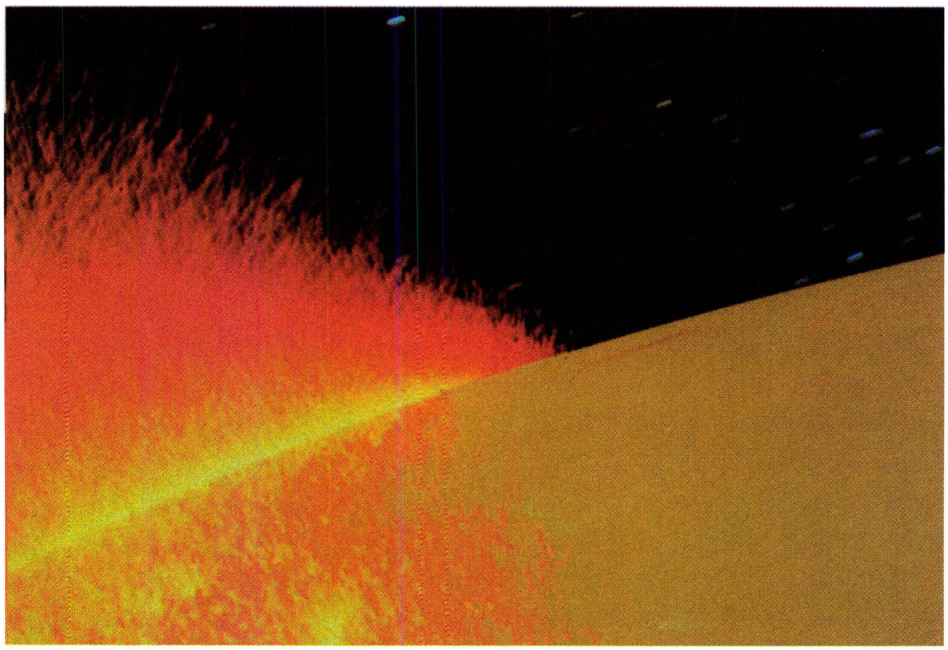

Fig. 7. Wall of fire about to engulf camera.

Reprinted From **acm Transactions On Graphics** — April 1983 — Vol. 2, No. 2

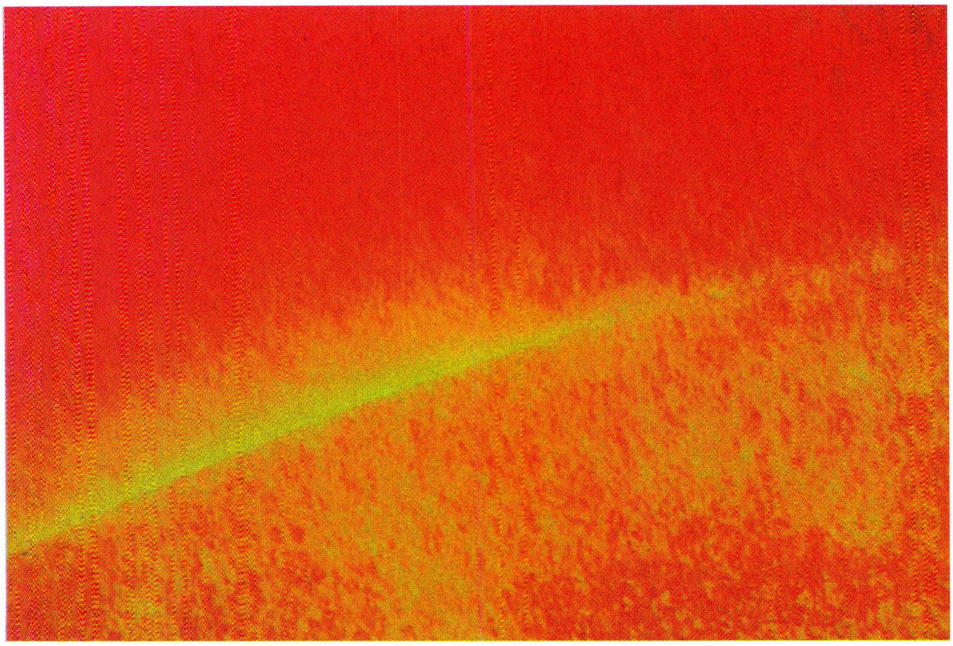

Fig. 8. Wall of fire completely engulfing camera.

particles are drawn until the simulation reaches frame n, so this backing up and restarting usually takes only a few minutes.

Particles moving off screen or being extinguished for any reason do not affect the randomness of other particles. This is because all stochastic decisions concerning a particle are performed when it is generated. After that, its motion is deterministic. If stochastic elements were to be used to perturbate the dynamics of a particle (e.g., to simulate turbulence), more care would have to be taken when checkpointing a frame and killing particles. In that case, it would probably be better to use a more deterministic and reproducible random number generator.

4. OTHER APPLICATIONS OF PARTICLE SYSTEMS

4.1 Fireworks

We are currently using particle systems to model fireworks. The fireworks differ from the Genesis Demo in that the control parameters of the particle systems vary more widely, and streaking is more predominate. Figure 9 shows two red explosions superimposed. One explosion is tall, thin, and near the end of its lifetime, and the other is short, fat, and building up to full steam. Figure 10 shows several green explosions dying off and blue spherical explosion starting up. Figure 11 contains overlapping, multicolored explosions formed with different generation shapes and ejection angles. Again, these images only really come alive when projected at 24 frames per second.

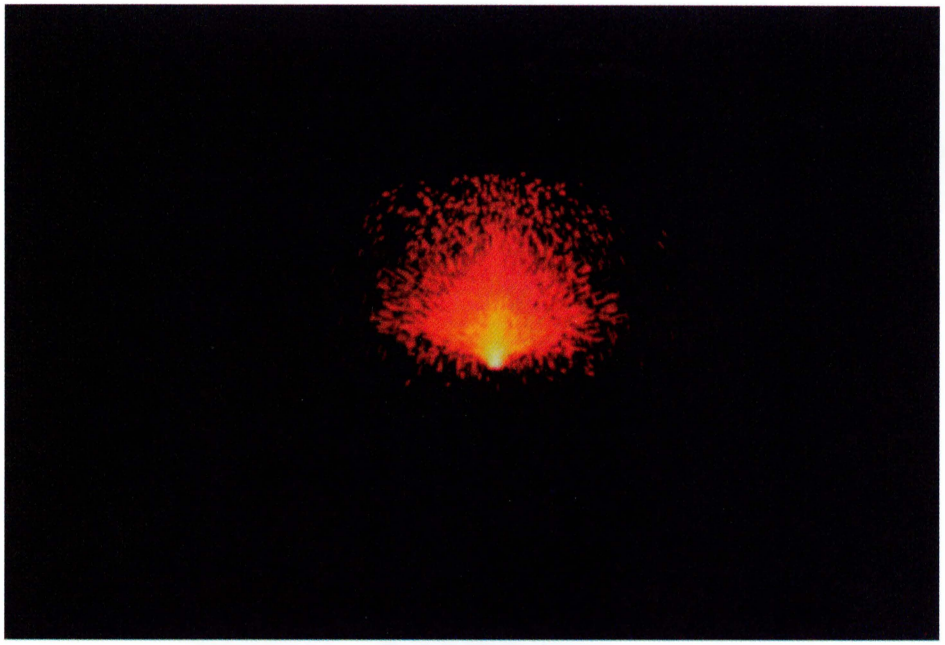

Fig. 9. Two red fireworks.

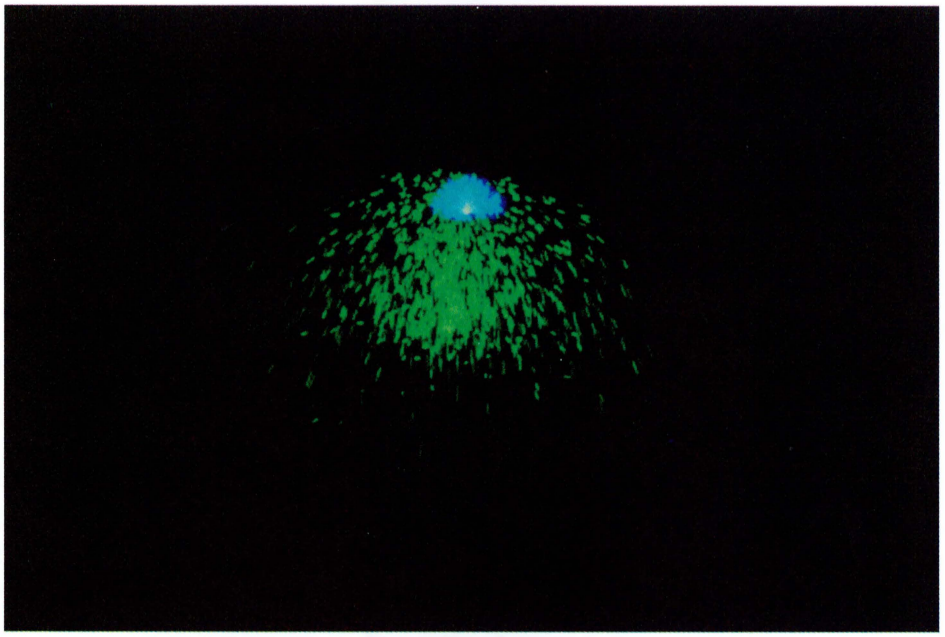

Fig. 10. Green and blue fireworks.

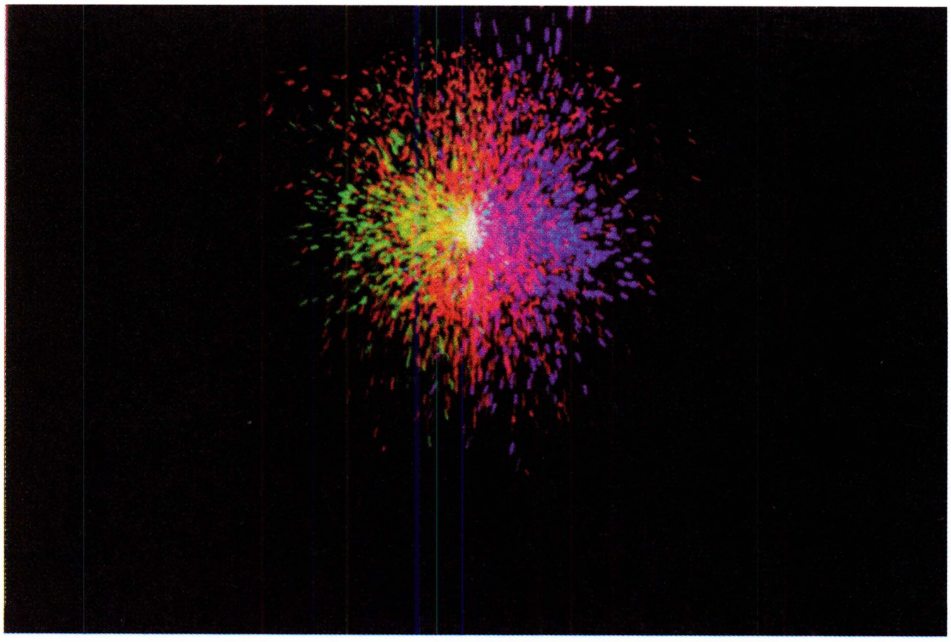

Fig. 11. Multicolored fireworks.

4.2 Line Drawing Explosions

Particle systems are being used to model exploding objects in a computer-simulated tactical display for a scene from the movie *Return of the Jedi* [8]. In this case, the particle systems are implemented on a line-drawing display. In order to simulate motion blur, the particles are drawn as very small straight lines instead of as points. The texturing effects that are evident in the previous examples are lost on a line-drawing display, and yet the motion still looks real and the sequence gives the viewer the impression that something is exploding. This is because the model is dynamic—it moves well.

4.3 Grass

To model grass, we use an explosive type of particle system, similar to that used in the Genesis Effect. Instead of drawing particles as little streaks, the parabolic trajectory of each particle over its entire lifetime is drawn. Thus, the time-domain motion of the particle is used to make a static shape. Grasslike green and dark green colors are assigned to the particles which are shaded on the basis of the scene's light sources. Each particle becomes a simple representation of a blade of grass and the particle system as a whole becomes a clump of grass. Particle systems randomly placed on a surface and overlapping one another are used to model a bed or patch of grass.

Figure 12 is a picture entitled *white.sand* by Alvy Ray Smith of Lucasfilm. The grass elements of this image were generated as described above. The plant

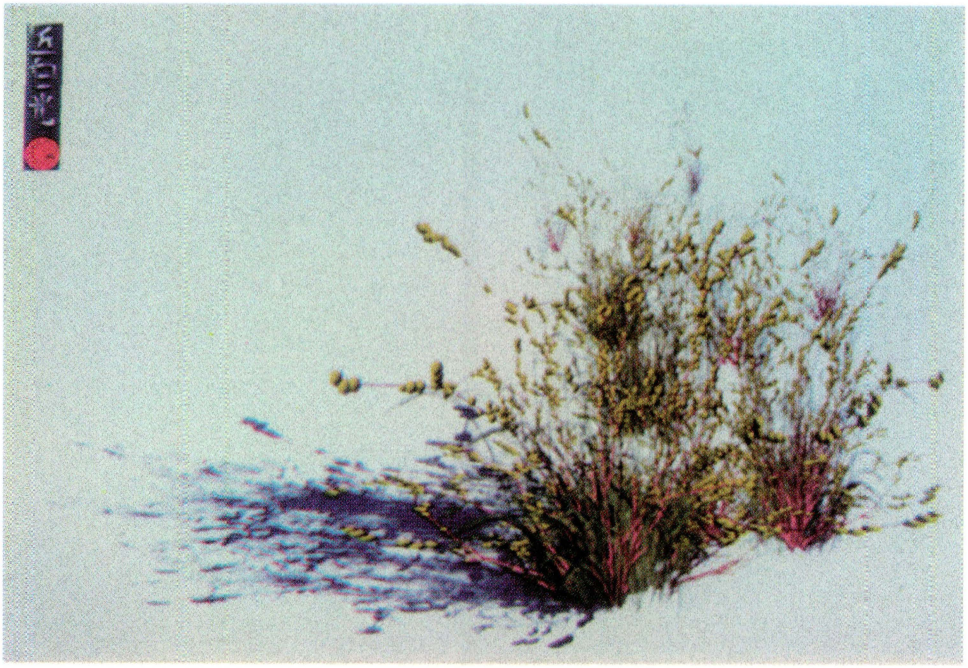

Fig. 12. *white.sand.*

elements were generated using a partially stochastic technique similar to particle systems.

5. ONGOING RESEARCH IN PARTICLE SYSTEMS

A logical extension of this research will be to use particle systems to model fuzzy objects in which the individual particles can not be rendered as point light sources, but must be rendered as individual light-reflecting objects.

To this end, we have begun to investigate the modeling of clouds. Clouds are difficult for several reasons. First, the shape and form of clouds are complex, depending on many factors such as wind direction, temperature, terrain, and humidity. The atmospheric literature abounds with cloud models that are simple in concept but computationally difficult, since most are based on partial differential equations. Second, clouds are difficult because they can throw shadows on themselves. This property is very important in making a cloud look like a cloud. Third, the number of particles needed to model a cloud will be very large. This will require an efficient rendering algorithm.

8. CONCLUSIONS

We have presented particle systems, a method for the modeling of a class of fuzzy objects, and have shown how they were used in making the fire element of the Genesis Demo sequence for the movie *Star Trek II*: *The Wrath of Khan*. Particle systems have been used as a modeling tool for other effects and appear promising for the modeling of phenomena like clouds and smoke.

Reprinted From **acm Transactions On Graphics** — April 1983 — Vol. 2, No. 2

Particles, especially when modeled as point light sources or as streaks of light, have proved efficient to render—they are merely antialiased lines. Because they are so simple, they lend themselves to a hardware or firmware implementation. With a hardware antialiased line-drawing routine, the computation of our wall-of-fire element would have been two to three times faster.

Particle systems are procedural stochastic representations controlled by several global parameters. Stochastic representations are capable of producing minute detail without requiring substantial user design time. The textures in the fire sequence could not have been modeled with other existing methods. Fire images, scanned in from a photograph or painted, could have been texture mapped, but they would still have been static. Another advantage of a procedural representation is its ability to adapt to several different viewing environments. For example, procedural representations can generate only as much detail as is needed in a frame, potentially saving significant amounts of computation time.

Having finally come to grips with spatial aliasing, it is now time for computer image synthesis to being to investigate and solve temporal aliasing problems. The Genesis Demo is the first "big screen" computer-synthesized sequence to include three-dimensional dynamic motion blur. The particles in a particle system can easily be motion-blurred because they are so simple. A great deal of work remains to be done in this area—blurring particles is much easier than blurring curved surface patches.

Particle systems can model objects that explode, flow, splatter, puff up, and billow. These kinds of dynamics have not been produced with surface-based representations. The most important aspect of particle systems is that they move: good dynamics are quite often the key to making objects look real.

7. ACKNOWLEDGMENTS

The author gratefully acknowledges the suggestions and encouragement of all members of the graphics project at Lucasfilm Ltd, especially those who worked on the Genesis Demo sequence: Loren Carpenter, Ed Catmull, Pat Cole, Rob Cook, David DiFrancesco, Tom Duff, Rob Poor, Tom Porter, and Alvy Ray Smith. The crusade for motion blur and antialiasing in computer image synthesis is a goal of the entire graphics project and Lucasfilm as a whole. One of the referees deserves credit for pointing out several extensions and improvements to the motion blurring discussion. Finally, thanks to Ricki Blau for editorial and photographic assistance.

REFERENCES

1. Abraham, R., and Shaw, C. *DYNAMICS—The Geometry of Behavior.* City on the Hill Press, Santa Cruz, Calif., 1981.
2. Badler, N. I., O'Rourke, J., and Toltzis, H. A spherical human body model for visualizing movement. *Proc. IEEE 67,* 10 (Oct. 1979).
3. Blinn, J. F. Light reflection functions for simulation of clouds and dusty surfaces. Proc. SIGGRAPH '82. In *Comput. Gr. 16, 3,* (July 1982), 21-29.
4. Csuri, C., Hackathorn, R., Parent, R., Carlson, W., and Howard, M. Towards an interactive high visual complexity animation system. Proc. SIGGRAPH 79. In *Comput. Gr. 13,* 2 (Aug. 1979), 289-299.
5. Fournier, A., Fussel, D., and Carpenter, L. Computer rendering of stochastic models. *Commun. ACM 25,* 6, (June 1982), 371-384.

6. KNUTH, D. E. *The Art of Computer Programming*, vol. 2. Addison-Wesley, Reading, Mass., (1969), p. 464.
7. KOREIN, J., AND BADLER, N. I. Temporal anti-aliasing in computer generated animation. To appear in Proc. SIGGRAPH '83 (July 1983).
8. LUCASFILM. *Return of the Jedi* (film), May 1983.
9. NORTON, A. Generation and display of geometric fractals in 3-D. Proc. SIGGRAPH '82. In *Comput. Gr. 16*, 3 (July 1982), 61–67.
10. PARAMOUNT. *Star Trek II*: *The Wrath of Khan* (film), June 1982.
11. PBS. *Carl Sagan's Cosmos Series.* (television series), Public Broadcasting System, 1980.
12. POTMESIL, M., AND CHAKRAVARTY, I. Modeling motion blur in computer-generated images. To appear in Proc. SIGGRAPH '83 (July 1983).
13. REQUICHA, A. A. G., AND VOELCKER, H. B. Solid modelling: A historical summary and contemporary assessment. *IEEE Comput. Gr. Appl.* (March 1982).
14. SMITH, A. R., CARPENTER, L., CATMULL, E., COLE, P., COOK, R., POOR, T., PORTER, T. AND REEVES, W. *Genesis Demo Documentary* (film), June 1982, Lucasfilm Ltd.

Received February 1983; revised April 1983; accepted April 1983