Approximate and Probabilistic Algorithms for Shading and Rendering Structured Particle Systems

William T. Reeves

Computer Research and Development,
Lucasfilm Ltd

Ricki Blau

Computer Science Division, Dept. of Electrical Engineering and Computer Science,
University of California, Berkeley

Abstract

Detail enhances the visual richness and realism of computer-generated images. Our stochastic modelling approach, called *particle systems*, builds complex pictures from sets of simple, volume-filling primitives. For example, structured particle systems have been used to generate trees and a grass-covered forest floor. Particle systems can produce so much irregular, three-dimensional detail that exact shading and visible surface calculations become infeasible. We describe approximate and probabilistic algorithms for shading and the visible surface problem. Because particle systems algorithms generate richly-detailed images, it is hard to detect any deviation from an exact rendering. Recent work in stochastic modelling also enables us to model complex motions with random variation, such as a field of grass blowing in the breeze. We analyze the performance of our current algorithms to understand the costs of our stochastic modelling approach.

CR Categories and Subject Descriptors: I.3.3 [**Computer Graphics**]: Picture/Image Generation - Display algorithms; I.3.5 [**Computer Graphics**]: Computational Geometry and Object Modelling - Curve, surface, solid, and object representations - Modelling packages; I.3.7 [**Computer Graphics**]: Three-Dimensional Graphics and Realism - Animation - Colour, shading, shadowing, and texture

General Terms: Design, Algorithms, Performance Analysis

Key Words: stochastic modelling, approximation

1. Introduction

Natural, as opposed to man-made, objects exhibit an immense variety of irregular shapes and random variation in their detail. To represent this variety in synthetic images, we would like to have models that are not entirely deterministic. The use of *stochastic* models has recently been advanced as an approach to creating naturalistic detail in computer-generated images [5]. The idea of "data amplification" is fundamental to stochastic modelling algorithms, as well as to other classes of algorithms described by Smith [13]. A simple data base specifies the general characteristics of the modelled object, and detail is generated algorithmically to describe the object fully. A complicated image can be generated from a small data base, and the amount of detail can vary with the displayed size of the object.

The best-known examples of stochastic modelling are the *fractal* algorithms of Fournier, Fussell, and Carpenter [5], inspired by the mathematics of Mandelbrot [9], and the particle systems described by Reeves [12]. Particle systems represent objects as clouds of primitive particles that occupy their volumes, rather than using more classical surface-based representations such as polygons, patches, and quadric surfaces. A particle system is not a static entity, as its particles can move and change form with the passage of time. The position, orientation, attributes, and dynamics of each particle are defined by a set of constrained stochastic processes.

Particle systems are important for three reasons. First, because a particle is much simpler than most graphical primitives, many more can be drawn in a given amount of computation time. Hence, a more complex and detailed image can be generated. Second, particle systems are both procedural and stochastic. By employing data amplification techniques, they require less human design time than conventional modelling methods. Third, particle systems can be used to model objects that change form over a period of time. In our experience, it is more difficult to represent complex dynamics of this form with surface-based modelling techniques.

This paper presents several new results in particle systems that were left as future research in our previous paper [12]. These results were used in the film *The Adventures of André and Wally B.* [7] to generate three-dimensional background images of a forest and of grass covering its floor. An example is shown in Figure 1.

Our new algorithms generate more sophisticated particle systems with greater internal structure. The strength of these stochastic modelling algorithms is their ability to transform a small set of simple constraints into a complete description of complex objects. The problem is that they create so much irregular, three-dimensional detail that exact visible surface and shading calculations become infeasible. Our solution is to exploit the visual complexity of the models by adopting approximate and probabilistic algorithms. The rich detail in the images tends to mask deviations from an exact rendering. We also present some recent work in stochastic modelling that enables us to model more complex motions, such as a field of grass blowing in the breeze. Finally, we analyze the performance of our current algorithms and discuss the potential gains of hardware support for rendering particle systems.

Several other researchers have modelled and rendered images of trees and vegetation. Brooks et. al. [3] at MAGI extended a combinatorial geometry system to model simple trees. Marshall et. al. [10] from Ohio State University used a

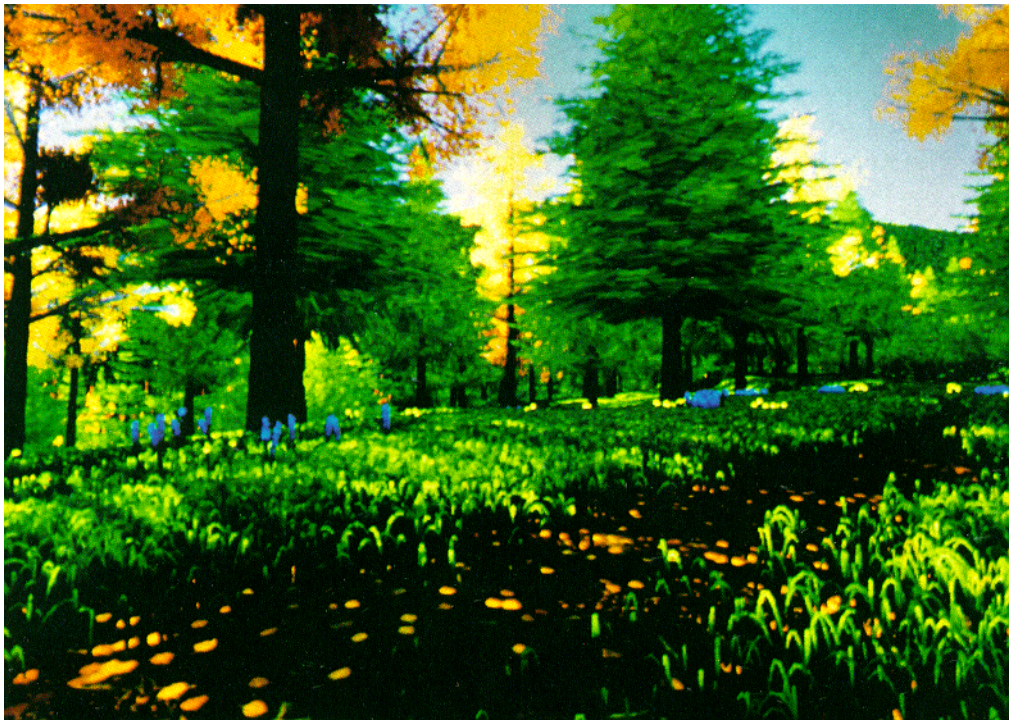


Figure 1. Forest Scene from *The Adventures of André and Wally B.*

procedural technique to generate polygonal models of trees. Both of these early efforts produced deterministic models of individual trees, whereas our algorithms create large groups of trees with stochastic variation.

More recently, Aono and Kunii [1] developed geometric models that are based on botanical descriptions of trees and that exhibit accurate branching structures. Their research emphasizes issues in modelling; our work is more strongly oriented towards creating complex, coloured images. Gardner [6] has simulated scenes of trees and terrain using textured quadric surfaces; his approach is more efficient than ours, but it creates objects that are less highly detailed. Smith [13] described a modelling technique called *graftals* that represents plants using Lindenmayer systems. Bloomenthal [2] has created remarkably realistic images of trees, based on finely-detailed surface-oriented representations. Both Smith and Bloomenthal emphasize the detailed structure of individual plants, whereas we take a more global view of a forest environment.

2. Structured Particle Systems

Particle systems were first used to model a wall of fire in the Genesis sequence from the film *Star Trek II: The Wrath of Khan* [11]. The fire is modelled as particles of light that move in three-space. Each particle system resembles a miniature volcano from which many particles explode upwards, eventually falling to the planet's surface due to the pull of gravity. A frame from this sequence, such as Figure 2, displays the trajectories of the visible particles during the time that the imaginary camera shutter is open to record the frame. The trajectories are approximated by a sequence of straight line segments, one per frame. Thus, a motion-blurred line is drawn to represent each particle.

Associated with each fire particle is a set of parameters that describe its position and characteristics: the location from which it erupted, its initial velocity, and attributes such as colour and size. At generation time, the parameters are assigned initial values, drawn from a random distribution. Each particle is generated and transformed through time independent of all other particles.

The particle systems that model natural phenomena such as trees and grass are more structured, and, consequently, the particles are not independent. Each tree is drawn as a set of line segments, and possibly small circles, that constitutes its branches and leaves. Many complex relationships exist among the particles representing the branches and leaves of a tree, as together they must form a cohesive three-dimensional object. The rest of this section describes new techniques and types of controls for generating *structured* particle systems.

2.1 Trees

The first step in modelling a forest scene is to populate the forest with individual trees, creating a tree data base that contains the location and type of each tree. The specifications in this pre-computed data base are subsequently used to generate and render the trees, frame-by-frame, during a second stage of the computation.

2.1.1 Creating the Forest

Sometimes the scene designer wishes to place each tree exactly, perhaps to simulate some real environment. The designer positioned the trees in Figure 3 with an interactive model editor to ensure that they fit into the scene with other, separately-computed elements.

When a vast number of trees are to be modelled, as in Figure 4, it is more practical to generate their locations procedurally with special-purpose programs. Our tree placement program requires four types of input: a grid size that controls the spacing between trees, a parameter that specifies the minimum distance between any pair of trees, one or more regions on the horizontal plane that will be filled with trees, and a terrain map that provides the elevation of points on the plane. The program creates at most one tree per grid point, generating random displacements independently in the x and y dimensions to offset the actual location of the tree from the given point. Should parts of the forest become more crowded than the minimum spacing parameter allows, grid points are left empty.

The layout of a landscape can be controlled even when the placement algorithm is stochastic. For example, texture maps can direct the program to leave the terrain bare where meadows, streams, or other open areas exist. An even more sophisticated placement algorithm could be based on empirical forestry models and consider factors such as elevation, water drainage, and sunlight to determine the density of growth.

Tree type selection is performed at the same time as tree placement. Each tree is assigned a type (e.g., maple, spruce, or aspen) either interactively or procedurally. The stochastic methods used for Figure 4 distribute the deciduous trees more densely in the valleys and evergreen trees more densely on the hills. The probability that a tree is evergreen increases with the elevation; we use a random number to choose among the possible tree types.

2.1.2 Generating a Tree

The particle systems program processes the trees in the data base serially; it generates a complete model for each tree and then renders the model. Starting with the main trunk, the algorithm constructs the tree by recursively generating subbranches. The data structure for the model is a tree in which each node describes a branch segment.

Before invoking the recursive branch generation procedure, the algorithm stochastically assigns a set of initial characteristics and dimensions. Some of these dimensions are illustrated in Figure 5.

The values for these parameters are randomly drawn from distributions associated with the tree's type. For example, the following equation determines the height of a tree:

$$Height = MeanHeight + Rand() \times DeltaHeight$$

where Rand() is a procedure returning a pseudo-random number uniformly distributed between -1.0 and +1.0. *MeanHeight* and *DeltaHeight* are specified for the tree type; they represent the mean height and the maximum difference from the mean. If the height is distributed uniformly between a and b, then *MeanHeight* is $\frac{a+b}{2}$, and *DeltaHeight* is $\frac{|b-a|}{2}$. In Figure 3, the deciduous trees have a *MeanHeight* of 60 and a *DeltaHeight* of 12; the evergreen trees also have a *MeanHeight* of 60, but a *DeltaHeight* of 16.

One parameter may depend on another. For example, the global *width* controls the breadth of the primary branches; this parameter is stochastically set to a fraction of the tree's height according to the equation:

$$Width = Height \times (MeanWidth + Rand() \times DeltaWidth)$$

MeanWidth is 0.6 for the deciduous trees, and 0.5 for the evergreens. *DeltaWidth* is 0.05 for the deciduous trees, and 0.25 for the evergreens.

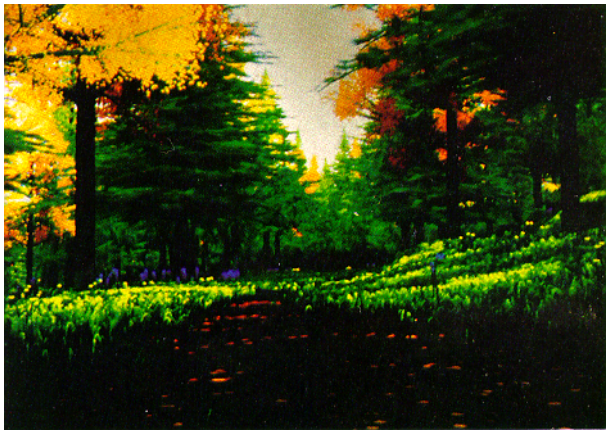


Figure 3. André's Forest

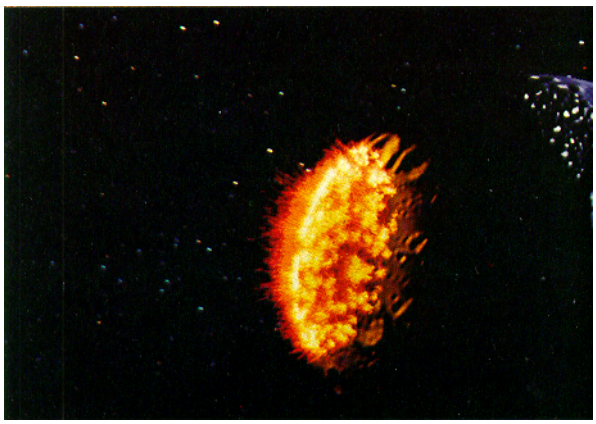


Figure 2. Frame from *Star Trek II: The Wrath of Khan*

Figure 4. Tree-covered Hills at Sunrise

The relationship between parameters need not be linear. For example, branches are tapered by decreasing the thickness as the distance d from the base of the branch increases, according to the equation:

$$thickness = thickness_b \times \sqrt{\frac{length - d}{length}}$$

where *length* is the total length of the branch and $thickness_b$ is the thickness at its base.

The height of the lowest branch is stochastically set to a fraction of the tree's height. The distance between two sub-branches is drawn from a distribution that depends on the tree type and the thickness of the parent branch. Another control indicates whether branches occur singly or in whorls.

The parameters of the branch length distribution depend on the dimensions of the tree, as illustrated in Figure 5. An approximate bounding volume for the tree is computed from its height and width; its shape varies with the tree type, conical for evergreens and elliptical for deciduous trees. For each branch, we select a *branching angle* from a distribution associated with the tree type. We next compute *MeanLength*, the distance to the surface of the bounding volume from the position where the branch meets the trunk. The actual length of any branch is taken from a distribution centered on *MeanLength*.

A recursive algorithm generates sub-branches. We sample a distribution to obtain the ratio between the diameters of the parent and each sub-branch. The sub-branch inherits many parameters from its parent, but some controls are adjusted to the dimensions of the child. For example, sub-branches are spaced more closely together as the branch thickness decreases. The recursion stops either when a branch reaches a minimum thickness or at a specified maximum depth of recursion.

Characteristically, aspen trees, such as in Figure 3, have forked branches, but our evergreens do not. For each species, we specify a probability that a branch forks. A parent-child relationship exists between a branch and its sub-branch, but two forks of a branch have a sibling relationship and share probability distributions.

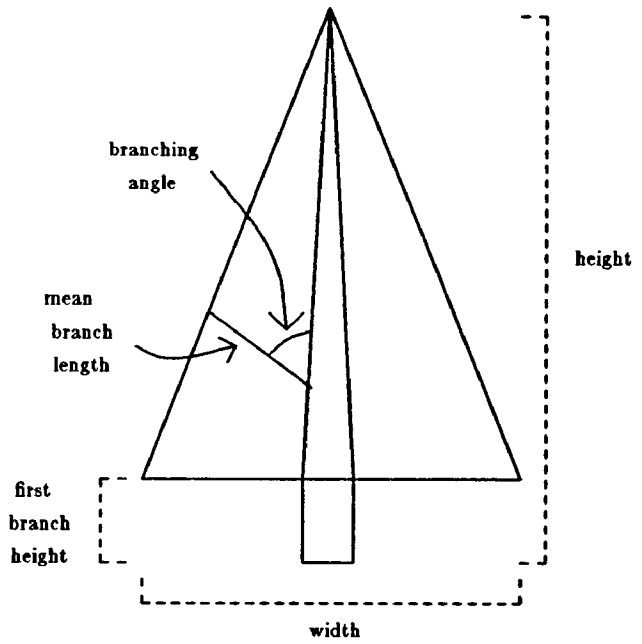


Figure 5. Dimensions Used in Tree Model

The colours in the image vary from tree to tree. The exact shades are offset by random amounts from mean values characteristic of the tree type. Two tables of colours are established for each tree, and the thickness of the branch determines which table is used. Colours for the trunk and main branches are taken from one table, and colours for the leaves from the other. Alternatively, one table may be used for old wood and the other for new.

The branch-generation algorithm produces trees with a regular structure, unlike real trees that have been affected by their environment and natural disasters. We simulate these effects by post-processing the three-dimensional description of the tree. Separate algorithms bend tree branches to simulate the effects of gravity, prevailing winds, and prevailing sunlight direction. Another algorithm randomly warps branches to simulate a form of catastrophe theory.

Finally leaves or needles are added to the branches that have no sub-branches. The stochastic parameters that control the placement and characteristics of leaves are: shape, orientation, spacing, density, colour, and location.

All of the random variables for the images in this paper were drawn from uniform distributions, as described above. In tuning the parameters, we concentrated more on visual results than on actual botanical data. Other, more accurate, distributions should be explored in the future.

Particle systems can be combined with elements computed using other techniques. For example, the tree trunks in Figure 3 were modelled as truncated, solid cones and rendered with conventional texture-mapping techniques.

Approximately 1.1 million particles compose the trees in Figure 3. Nearly sixty megabytes of binary data were generated from only twenty-one thousand bytes of input, resulting in a data amplification factor of over three thousand. The expansion factor is, naturally, less for scenes in which each object occupies only a small area of the screen. The input for the trees in Figure 4 was expanded by only a factor of four to generate fourteen megabytes of data; however, the input data base itself was generated procedurally from a much smaller specification.

2.2 Grass

The images of grass result from an extension of the work reported earlier by Reeves [12]. Clumps of grass are scattered randomly over the input terrain. A texture map, with a bird's-eye view of the terrain, optionally specifies the locations of bare spots or different types of grasses. Some stochastic parameters specify global parameters for an entire clump of grass: its position, orientation, area, density, and type.

Each clump contains many separate blades of grass. Both the structure of the clumps and the geometry of the individual blades are simpler than the models used for trees. Stochastic processes determine the number of blades within the clump and the characteristics for each blade: position, height, thickness, curvature, orientation, and colour. Short, straight-line particles approximate each blade's parabolic arc. Stochastic bends and kinks added to some blades of grass enhance the realism of the image. Simple flowers are created by adding yellow or blue particles to some blades of grass.

Eighteen thousand clumps of grass are visible in Figure 3. A total of 733,887 particles were drawn to render the grass.

3. Shading Models for Particle Systems

The fire particle systems require only simple shading calculations, because each particle is modelled as an independent light source[12]. Each fire particle is stochastically assigned an initial colour that changes over time according to a simple linear relationship that simulates cooling. The tree and grass particle systems reflect rather than emit light; they consequently require a more sophisticated shading model with ambient, diffuse, and specular shading components. For more realistic rendering, the shading algorithm also provides self-shadowing, external shadows, and coloured light sources.

A single tree may be composed of over one million independent particles. It would be a formidable task to shade each particle exactly, calculating whether it is in shadow and determining if it should be highlighted. In fact, unless the camera points directly at the sun from within the tree, it is almost impossible to tell whether or not any one leaf should be in shadow. Our solution is to use a probabilistic shading model for both the trees and the grass. For example, the particle's position and orientation determine the probability that it is in shadow. We calculate this probability and then use a random number to decide whether or not to render the particle as if it were in shadow.

3.1 Trees

Trees are self-shadowing, as branches and leaves of the tree shadow other parts of the same tree. In a forest, a tree is also shadowed externally by neighbouring trees. Our shading functions provide ambient, diffuse, and specular components and also approximate both forms of shadowing.

Highlights occur where the tree's branches or leaves are exposed directly to sunlight. This condition is most likely to exist close to the outer edge of the tree in the direction of the sun. Accordingly, the diffuse shading component for a particle varies with the distance into the tree from the light source, d_d, as shown in Figure 6. The diffuse component drops off exponentially as d_d increases according to the following equation:

$$D = e^{-\alpha d_d}$$

The parameter α controls the rate of the exponential dropoff. Random highlights are added by stochastically turning on a specular component whenever d_d is small and the cosine of the angle between the light direction and the branch direction is close to zero.

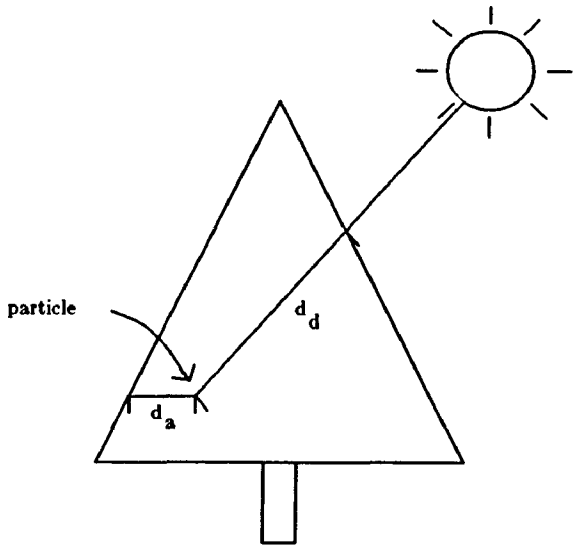


Figure 6. Distances Used by Tree Shading Algorithm

Self-shadowing is simulated primarily by controlling the ambient shading component. The ambient component for a particle drops off exponentially as the distance into the tree, d_a, increases. This distance, as shown in Figure 6, is independent of the position of the light source and represents the shortest distance from the particle to a point on the tree's bounding volume in a direction parallel to the ground. Another parameter, $A_{\min}$, sets a minimum for the ambient component and guarantees that there is some light even in the deepest interior of the tree. The ambient component equation is therefore:

$$A = \max(e^{-\beta d_a}, A_{\min})$$

A different problem is to add external shadows, those cast by other trees. Again, we use an approximation technique because exact computation of the shadows would be very expensive. The locations and heights of any trees positioned between a specified tree and the light source define a plane that skims the top of neighbouring trees and passes through the light source. An example is shown in Figure 7. Particles above this plane are in full sunlight, so the specular, diffuse, and ambient shading components all contribute to the shading calculation. The probability that sunlight reaches other parts of the tree decreases for particles located below the plane. If a particle is more than a specified distance below the plane, only the ambient shading component is used. For particles lying in between, a random number is selected to decide if the diffuse and specular components contribute to the shading.

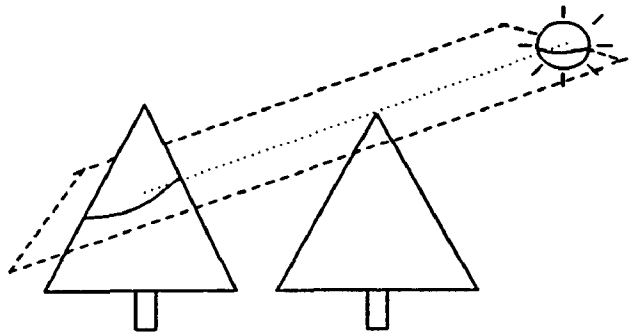


Figure 7. External Shadow Plane

To heighten the visual effect of the images in *André and Wally B.*, we used different colours for different types of light. A yellowish tinge to the diffuse and specular components produced an early morning sunrise warmth, and a bluish tinge to the ambient component of the light gave us a look inspired by the landscape artistry of Maxfield Parrish [8]. Figure 8 shows trees shaded with the techniques described in this section.

3.2 Grass

A similar stochastic algorithm shades the grasses. The contributions of both the diffuse and ambient components depend on the distance from the top of the clump of grass to the particle in question, decreasing exponentially as the depth increases. The difference between the two is that the diffuse component drops off much more quickly than the ambient component. As with trees, each lighting component can have a different colour.

The strongest visual effects are due to the shading function that casts tree shadows onto the grass. The principle idea behind our algorithm is a form of ray casting. We view the particle from the light source to see if it is visible through the trees. If the particle is not visible, it is in shadow. A simple and effective device for implementing this idea is the *shadow mask*. Our method is similar to a technique used by Lance Williams [15].

To create the shadow mask for a scene, we first compute an orthographic image of the trees from the direction of the light source. We then extract the silhouette information from this image and form a texture map. Before shading each particle of grass, we transform its position by this same orthographic transformation, effectively calculating a view of the particle from the light source. The calculated screen coordinates index into the shadow mask texture map to obtain a value that indicates how much the particle is in shadow. In the final colour computation, this value determines how much of the diffuse lighting component to use. If the particle is completely in shadow, only the ambient shading component is used. A single, pre-computed shadow mask is used for all frames in which the positions of the trees and of the light source remain constant. The shadow mask texture map in Figure 9 was used to shade the grass element in Figure 10.

The shadow mask technique effectively represents the shadows of trees on the grass because the trees are always between the grass and the sun. In general, the shadow mask technique works only if the objects casting the shadow are completely between the light source and the surface on which the shadow is cast. This limitation arises because the texture map that stores the shadow mask is only two-dimensional. Three-dimensional texture maps that store both depth and coverage information could solve this problem, but we have not explored this area.

4. Visible Surface Determination

Our previous research essentially ignored the visible surface problem with particle systems. The fires of *Star Trek II*, as shown in Figure 2, were composed of light-emitting particles. When several particles overlapped in a pixel, their colours were simply added together. With light-reflecting particles, such as those of the trees and grass, one particle can obscure another by being in front of it with respect to the selected camera position. We must now solve the visible surface problem.

The following sections describe two slightly different visible surface algorithms, one for the trees and one for the grass.

4.1 Trees

Consider a forest scene containing many trees. As we have seen, our stochastic generation algorithms usually attain a significant amount of data amplification. It is not feasible to generate particles for all trees in an image and then perform a traditional visible surface algorithm on them. Instead, we employ a painter's algorithm approach. We first sort all trees in the scene into a back-to-front order with respect to screen depth. Then, for each tree in turn, we generate a stochastic model, shade it, and render it into the image on top of any trees that have been previously rendered. As soon as a tree has been rendered, its data is discarded.

The accuracy of this back-to-front approach relies on the assumption that the bounding volumes of the trees do not intersect. While this assumption is inaccurate for forests in general, interesting images can be made with tree databases that conform to this restriction. The grass visible surface algorithm of the next section removes this non-intersecting restriction but is slightly more expensive.

Rendering each individual tree is not trivial either, as some branches of the tree obscure others. Within a tree we also apply a form of painter's algorithm that depends on a bucket sort. The bounding volume of the tree defines a set of buckets that are indexed by the eye space z distance of the particle — that is, by its depth into the scene. As a particle is generated, it is transformed into two-dimensional screen space and inserted into the bucket list corresponding to its average eye space z distance. After all particles have been generated, they are drawn in back-to-front bucket order on top of any particles that have already been rendered into the image. All particles are drawn as small, antialiased circles or short, antialiased straight line segments.

This bucket sort is another inexpensive but successful approximation. A completely accurate comparison sort of all the particles in a scene would require $O(n \log n)$ operations, where n is the number of particles. The approximate algorithm is linear in the number of particles, assuming that the trees have been sorted previously. Therefore, it requires only $O(m \log m + n)$ operations, where m, the number of trees, is much smaller than n. If we skipped the preliminary sort of the trees, a bucket sort of all particles could be accomplished with $O(n)$ operations, but the memory costs would be much greater. For our forest scenes, n typically ranges between 1.0 and 1.7 million. In contrast, m is usually less than 10,000, and, in close-up scenes such as Figure 11, it is only 195. We have never noticed any anomalies attributable to the bucket sort in any of our static or dynamic images. We commonly use about 2000 buckets. Even for a large tree, spanning fifty feet of depth, each bucket would cover a small area, about 0.3 inches. Since the images are so complex to start with, any imperfections are very difficult to detect as long as they are consistent from frame to frame.

4.2 Grass

We cannot create the appearance of a continuous carpet of grass without allowing clumps of grass to intersect. The visible surface algorithm for grass is more sophisticated than for trees because it permits intersecting clumps. The algorithm sorts the clumps of grass back-to-front and then calculates the bounding box of each clump in eye space. The clumps are generated in the back-to-front order, and the particles are entered into the bucket sort list. The difference is that the bucket list is not flushed and drawn at the end of each clump as it is at the end of each tree. Instead, only some of the buckets are drawn at the end of each clump. A bucket is drawn only if none of the bounding boxes from the remaining clumps overlaps it in eye-space z distance. Because the clumps are sorted, it is trivial to test for this condition.

Whenever the rear-most buckets are drawn, the range of the entire bucket list slides forward in eye-space z depth. Any undrawn particles remaining in the list may need to be reassigned to different buckets. This algorithm is more expensive than the non-intersecting tree algorithm because of two additional tasks – checking for overlapping buckets and reassigning particles when the range of the bucket list changes.

5. Complex Particle System Dynamics

The particle dynamics of the fires [11] were simple — each particle independently followed a parabolic trajectory. In the film *The Adventures of André and Wally B.*, the tree and grass elements were still. Since then, we have added realistic dynamics to depict a field of tall grasses blowing in the breeze. Stochastic models represent two phenomena, gusts of wind and the motion of wind-blown blades of grass.

The first step is to model wind. The scene designer specifies the terrain, a prevailing wind direction, and the average wind speed. A program generates wave fronts of particles that travel across the terrain. Each particle represents a small, localized gust of wind. The waves of particles display random variation, moving roughly parallel to the specified wind direction. New waves arrive at a rate that matches a specified wind gust frequency.

A *wind map* is a two-dimensional, top-down view of the terrain that specifies the wind intensity at discrete grid points. Letting the waves of particles move through time, we build a wind map for each frame of the animated sequence. The intensity of the wind at a particular location and time is determined by the number of wind gust particles close to the grid point.

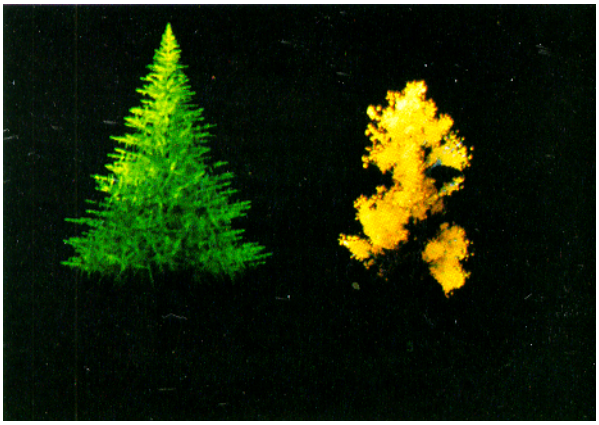


Figure 8. Shaded Trees

Figure 11. Close-up Forest Scene

Figure 9. Shadow Mask

Figure 12. Still from Film of Blowing Grass

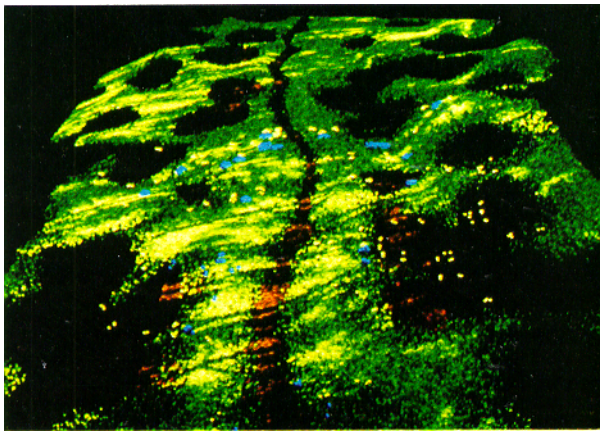


Figure 10. Grass Element Shadowed by Shadow Mask

Stochastic processes also model the reaction of the blade of grass as it is hit by a gust of wind. The simulated motion has the following key features:

- The blade bends around an axis that is perpendicular to the wind direction and passes through the blade's base.
- The amount of bending depends on the intensity of the gust of wind.
- The amount of bending increases with the distance from the ground, so that the blade is stiffest at its base.
- Over time and in the absence of other gusts, the blade returns to its original position following a damped sinusoidal motion called a *bending function*.
- The effects of successive gusts of wind are additive. At each frame, a bending function is generated and remembered for subsequent frames. The effective bending function at a given frame is the sum of all active bending functions.

The use of a two-dimensional wind map limits the motion of the wind to a plane. However, the algorithm achieves a three-dimensional look by tying each blade of grass to the ground and varying the degree of bending over the length of the blade.

Stochastic processes add further variation to the motion. For example, all blades do not bend exactly perpendicular to the wind direction, and blades close together use slightly different wind gust intensities. A blade of grass is bent by transforming the particles that represent it, taking care to keep them connected.

We can experiment with stochastic wind functions and wind maps, watching the wind patterns flow in real time on our vector display system. After adjusting the controls interactively, we can re-compute the motion in a few minutes to inspect the results. A short animated film of blowing grass has been computed. Figure 12 is one frame from the film.

6. Performance of Particle Systems Algorithms

In order to characterize the performance of particle systems algorithms, we measured the execution of four programs that produced different types of stochastic elements. All programs were run under Berkeley 4.2 UNIX† on an essentially idle VAX 11/750 with floating point accelerator and 4Mb of memory. The programs were written in C, except for some assembly-language routines for mathematical functions and matrix manipulation. We computed the elements twice, once storing the images in the virtual memory of the user process, as reported in Table 1, and once using a frame buffer with microcode line-drawing routines (which we call an *enhanced* frame buffer). All images were computed at 512 by 512 resolution.

Execution profiling explains where the cpu time is spent in the four programs. Table 2 shows the distribution of the user cpu time among three major phases of computation, described below:

1. generate,
2. shade and render, and
3. draw particles

The unaccounted time was consumed by global initialization, argument parsing, and sorting the data base. We also omit the time spent drawing trunks on the Near Trees, because they were not modelled by particle systems.

6.1 Generate

The details of model generation depend on the type of object being modelled. For example, one-fifth of the tree generation time is spent in the sin and cos routines, computing branch vectors based on the angle a branch makes with the trunk or parent branch. Simpler stochastic processes are used to generate the Fire element.

Model generation uses random numbers extensively. The incremental pseudo-random number generator in our math library takes about twenty-seven minutes to compute the twenty million random numbers used for the trees in Figure 4. In contrast, inline code accessing a pre-computed table of a few hundred random numbers takes less than four minutes. The table-driven approach creates satisfactory visual diversity and can provide, at approximately equal cost, random numbers drawn from any type of distribution. The visual success of this optimization has been observed by others [5,13].

6.2 Shade and Render

The second phase performs shading and perspective calculations, clips particles, and sorts them into screen-depth buckets. Perspective and clipping routines typically take between five and ten percent of the program's total running time. Shading also consumes about five to ten percent of the compute time for the trees and grass. Less time was spent shading the Far Trees than the Near Trees because simpler lighting models were used and external shadows were omitted. The least amount of time was spent on the shade and render phase for the Fire element, because it has the simplest shading models and requires no visible-surface calculations.

Floating point matrix and mathematical library routines accounted for about one-fifth of the total user time. These routines are called during both of the first two phases, but not in the final phase which uses only screen coordinates. Additional floating point operations are performed by in-line code in procedures outside the math and matrix libraries. For example, we examined a procedure that accounts for one-third of the tree generation time. More than three-fourths of the instructions in its inner loop were floating point instructions.

	Far Trees *Figure 4*	Near Trees *Figure 9*	Grass *Figure 9*	Fires *Figure 2*
number of frames	1	1	1	11
user cpu time (hh:mm:ss)	5:31:18	10:27:27	4:44:31	52:50
memory (Mb)	4.95	10.17	–	5.04
lines drawn (1000's)	592	1067	416	268
lines per frame (1000's)	592	1067	416	24

TABLE 1. Overall measurements for particle systems programs

	without frame buffer				with enhanced frame buffer			
	Far Trees	Near Trees	Grass	Fires	Far Trees	Near Trees	Grass	Fires
generate	32.5	11.4	13.8	28.3	47.6	18.6	25.0	37.9
shade and render	25.1	28.4	32.3	23.6	30.3	45.1	57.6	30.1
draw particles	33.2	49.9	51.6	41.0	14.8	19.6	13.2	23.6
total	90.8	89.7	97.7	92.9	92.7	83.3	95.8	91.6

TABLE 2. Percent of user time spent in each phase of computation

† UNIX is a trademark of Bell Laboratories.

6.3 Draw Particles

The final phase takes as input the description of particle primitives in two-dimensional screen coordinates and draws the primitives into the frame buffer. Our enhanced frame buffers have microcode routines for drawing anti-aliased lines and circles. The host provides the x and y coordinates for line endpoints, the width of the line, and its colour; the frame buffer then determines which pixels to modify. When a virtual frame buffer is used, both line-drawing calculations and address computation are performed in software, and drawing particles is the most expensive phase of the computation. As Tables 2 and 3 show, the frame buffer successfully takes over much of this work, leaving the host cpu to spend most of its time generating, shading, and rendering the model. Furthermore, the real frame buffer provides physical memory for the entire image and reduces the operating system costs for virtual memory management.

	Without frame buffer	With enhanced frame buffer	ratio with / without
Far Trees	5:31:18	4:06:05	.74
Near Trees	10:27:27	6:13:52	.60
Grass	4:44:31	2:34:42	.54
Fire	52:50	41:28	.78

TABLE 3. User time with and without a real frame buffer

An important advantage of the virtual frame buffer is its ability to store data of an arbitrary precision. The particle systems algorithm builds up a picture by repeatedly adding very small amounts of colour to a pixel. Experience indicates that eight bits per primary colour, while sufficient for displaying a finished image, are inadequate for computing some images. A real frame buffer with twelve to sixteen bits per colour and functions for drawing anti-aliased particle primitives would provide valuable hardware support for rendering particle systems.

Table 2 shows that Near Trees consumed proportionally more particle drawing time than Far Trees. Because the Near Trees particles appear larger on the screen, the line drawing calculations for each particle take longer.

6.4 Summary

Notably, no single phase of the computation dominates. Theoretical results obtained by Fournier show that the cost to compute a stochastic parametric surface by a fractal subdivision algorithm is linear in the number of points to be displayed [5]. He argues that if the subdivision algorithm is implemented efficiently, the time required to generate the model should be less than the time to transform, shade, and display it. From our measurements, we similarly conclude that the cost of creating an image with particle systems is not specifically due to the expense of generating the stochastic model, but is more generally explained by the inherent need to process a great deal of complexity in all phases of image creation. Even when we offload most of the costs of drawing primitives to the frame buffer, more time is typically required to manage, draw, and render the model than to generate it.

It is difficult to compare our measurements with the costs of creating synthetic images using traditional models. Images as complex as our forest scenes have rarely been modelled without using stochastic, or other algorithmic, methods to generate detail. When they have, the modelling effort must typically be measured in human design time. The costs of shading, visible surface calculations, and rendering are also difficult to compare with previous work, because few other measurements are available. Crow [4] and Whitted and Weimer [14] have published some measurements for pictures using conventional models, but their images were much simpler than ours. An interesting area for future work is the measurement and performance analysis of more general-purpose modelling and rendering software. We are currently investigating this area.

7. Conclusions

We have demonstrated that particle systems are able to model complex and structured objects. Simple primitives, given a set of relationships that bind them into a cohesive whole, can be used to produce complex models with extensive and varied detail. These relationships must specify the constraints by which millions of particles are dependent on one another. Because our new particle systems are more structured and dependent, we have developed more sophisticated algorithms to model their dynamics.

Particle systems were first used to model an amorphous phenomenon, fire, and it seemed natural to use a volume-filling representation. The tree and grass images demonstrate that volume-filling representations, such as particle systems, can effectively model solid objects. Such objects are conventionally modelled by surface-based techniques or solid geometry.

Procedural modelling techniques can be used to create models with more detail than a human designer could ever specify, and stochastic approaches can provide a rich variety of detail. Unfortunately, it is infeasible to compute exact solutions to the visible surface and shading problems for the enormous amount of detail that we can generate. Our algorithms are based on the belief that exact solutions are not always necessary in scenes with great visual complexity. We described an approximate painter's algorithm for visible surface determination and introduced probabilistic approaches to shading. Shadow masks, implemented as texture maps, simplified the task of adding shadows to the image.

Performance analysis shows that the computation is distributed relatively evenly among the three major phases of our particle systems algorithms: model generation, shading and rendering, and particle drawing. In particular, the cost of generating complex, structured models does not dominate the computation, as one might expect. Instead, the expense arises from the need to process a vast amount of three-dimensional detail throughout all phases of the computation in order to create the visual richness that we desire.

8. Acknowledgements

We would like to thank the members of the Computer Graphics Project of Lucasfilm Ltd for forming a stimulating and enjoyable working environment. John Lasseter contributed significantly to the visual design of the forest scenes and provided welcome encouragement. Eben Ostby developed special compositing software for the forest backgrounds. The shading algorithms profited from our discussions with Rob Cook. The work of the second author was supported in part by a State of California Microelectronics fellowship.

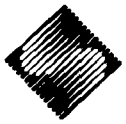
9. Bibliography

[1] Aono, M. and T. L. Kunii, Botanical tree image generation, *IEEE Computer Graphics and Applications 4*, 5 (May 1984), 10-34.

[2] Bloomenthal, J., Modeling natural trees with space curves, *SIGGRAPH 85, Computer Graphics 19*, 3 (July 1985).

[3] Brooks, J., R. Murarka, D. Onuoha, F. Rahn and H. Steingurg, An extension of the combinatorial geometry technique for modeling vegetation and terrain features, Contract report 159 for USA Ballistic Research Laboratories, Mathematical Applications Group, Inc., June 1974.

[4] Crow, F. C., A more flexible image generation environment, *SIGGRAPH 82, Computer Graphics 16*, 3 (July 1982), 9-18.



[5] Fournier, A., D. Fussell and L. Carpenter, Computer rendering of stochastic models, *Comm. ACM 25*, 6 (June 1982), 371-384.

[6] Gardner, G. Y., Simulation of natural scenes using textured quadric surfaces, *SIGGRAPH 84, Computer Graphics 18*, 3 (July 1984), 11-20.

[7] Lucasfilm Ltd, *The Adventures of André and Wally B.*, (film), Aug. 1984.

[8] Ludwig, C., *Maxfield Parrish*, Watson-Guptill, New York, 1973.

[9] Mandelbrot, B. B., *Fractals: Form, chance and dimension*, Freeman, San Francisco, 1977.

[10] Marshall, R., R. Wilson and W. Carlson, Procedure models for generating three-dimensional terrain, *SIGGRAPH 80, Computer Graphics 14*, 3 (July 1980), 154-162.

[11] Paramount, Genesis Demo from *Star Trek II: The Wrath of Khan*, in *SIGGRAPH Video Review Number 11*, June 1982.

[12] Reeves, W. T., Particle systems—A technique for modelling a class of fuzzy objects, *SIGGRAPH 83, Computer Graphics 17*, 3 (July 1983), 359-376.

[13] Smith, A. R., Plants, fractals, and formal languages, *SIGGRAPH 84, Computer Graphics 18*, 3 (July 1984), 1-10.

[14] Whitted, T. and D. M. Weimer, A software testbed for the development of 3d raster graphics systems, *Transactions on Graphics 1*, 1 (Jan. 1982), 43-58.

[15] Williams, L., Casting curved shadows on curved surfaces, *SIGGRAPH 78, Computer Graphics 12*, 3 (Aug. 1978), 270-274.