Efficiently Rendering Gobs and Gobs of Particles

Jerry Tessendorf*

Cinesite Digital Studios

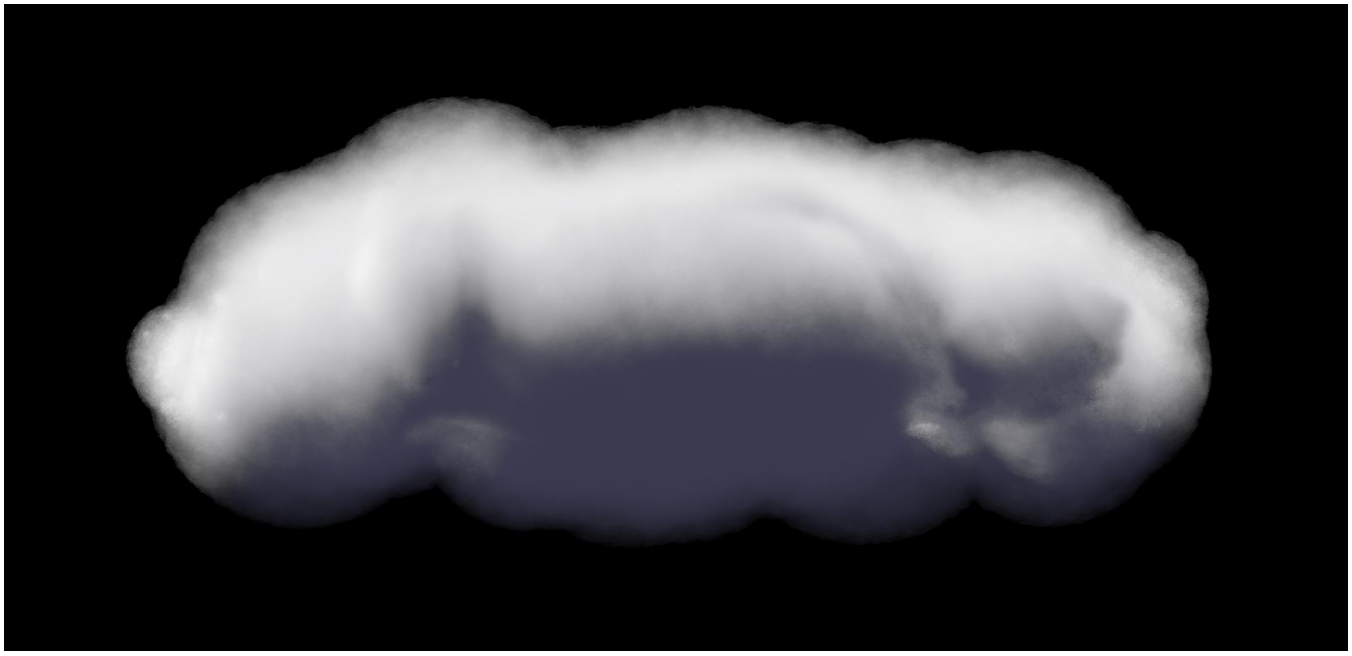


Figure 1: Cloud-like distribution with 1,167,396,675 particles, rendered using volumetric self-shadowing and the other techniques described in the text.

Abstract

Particles systems play a very strong role in entertainment computer graphics because of their versatility and easy connection to natural phenomena. Some applications would benefit from the simulation and visualization of millions to billions (10^6 to 10^9) of particles as a single system. A direct attempt to model and render such large numbers of particles with commercial off the shelf graphics tools presents a difficult challenge because they are not designed to efficiently handle this extreme problem. By specializing the rendering algorithms to the unique conditions specific to this problem, software has been developed and deployed which is able to render unlimited numbers of particles in very low amounts of RAM. This approach also generates volumetric lighting and opacity, including self-shadowing, making the particle renderer an efficient volume renderer. The algorithms combine several channels of alpha and depth data with techniques to transform the hiding and shadow map problems to a compositing operation. There are similarities to the "Deep Shadow Map" concept[1] of Lokovic and Veach, but we rely on a statistical rationale to reduce the data load even further. When augmented with a semi-procedural method of generating particle data from guide particles and "emission algorithms", complex visulizations can be generated with little or no disk space and very little RAM. Several examples are provided, including one which exceeds 10^9 particles in the image.

1 Introduction

A great many natural phenomena of interest in computer graphics have a physical description in terms of particles and their interactions. Even fluid dynamics, which is a description of the motion of a continuum material, is derivable physically and theoretically from the bulk behavior of a great many very tiny particles[6, 2, 3]. The branches of physics called Statistical and Condensed Matter Physics explain large scale systemss in terms of underlying discrete quantities and interactions. This is reflected in computer graphics by the popular and successful techniques of using dynamic particle systems to model, animate, and/or render stars, rainfall, spray[4], hair, cloth[5], fire[7], smoke, "energy beams", and many more effects that may not on first blush have a particle nature. Usually however, rendering of particles is augmented with sprites, blobs, and textures in order to fill out fine detail that is not explicitly modeled with particles.

In this paper we present our system for modeling, animating, and rendering very large numbers of particles. What is meant by "very large" is that there should be no limit to the number of par-

*Cinesite Digital Studios, 1017 N. Las Palmas Ave., Los Angeles, CA 90038, email: jerry@cinesite.com

ticles rendered. In practice, we have achieved film-quality high resolution renders of as many as 10^9 particles in a single frame using very modest computing resources, with the ability to render even larger numbers bounded only by the limit of cpu time allotted for the render. The intent is to render very large numbers of sub-resolution particles which model both fine detail and bulk structural properties, in effect building a macroscopic object from its microscopic bits. For example, the image below the title of this paper contains 290 million particles and exhibits volume-rendering-style self-shadowing of the distribution of the particles, while requiring only 22 MB of RAM to hold the image data, shadow maps, particles and code in the rendering process. The render time of 43 hours on a Xeon 900 MHz processor includes the time needed to generate the shadow maps, which is nearly 50% of the total render time.

At the heart of our system is a custom-made particle renderer which efficiently renders anti-aliased, motion-blurred spherical particles with surface shading. The particles are modeled at render time as true spheres, not points, which occlude each other. To handle occlusion of one particle by another, Lokovic and Veach introduced the concept of *deep shadow maps*[1], in which the alpha channel is generalized into a curve of alpha as a function of depth. We have adopted instead a statistically-oriented approach of computing several channels of the statistical properties of alpha and depth. By computing these statistics on the fly during the render, we can construct an effective spatio/statistical model of particle density which controls volumetric opacity as a statistical function of depth. Using this process to create shadow maps for lights, rendered images exhibit realistic volumetric self-shadowing behavior.

In order to render large numbers of particles, we have developed a mechanism to efficiently feed particles to the renderer. Storage of 10^9 particles in RAM or on disk is technically feasible but limits its practical applications. The alternative we have chosen in this system combines *guide particles*, which are animated traditionally, with *child particles* which are spawned procedurally from the guide particles. The guide particles feed controling parameters to the procedural algorithm for spawning children, so that arbitrarily fine detail can be built into the distribution of particles while generating them in large quantities. We have implemented the proceduralism in *particle emission shaders* to flexibly design the procedure while fully coupling the guide particle attributes with algorithms and child particles.

2 Rendering Spherical Particles

Even though we think of the particles as very small spheres, in fact smaller than the camera can resolve, it is still important to accurately compute the contribution each sphere can make to the image. Without such an accurate calculation, errors can appear in a moving scene as shimmering or aliasing artifacts, and even in still images as unexpected holes in the density of the image. So the first step in rendering many small particles is to accomplish an accurate render of individual spheres. As a side benefit, the renderer performs well when a sphere becomes resolvable as well.

An examination of figure 2 shows clearly that the image plane silhouette of a 3D sphere is not always a circular disk, and is also not an ellipse. The image plane silhouette is a quadratic function of position on the image plane, and the shape is more accurately described as a slice through a 3D ellipsoid. In this section, an implicit formula is given for the silhouette shape. The key quantities that control the shape are the position of the center of the sphere in the image plane, and the ratio of the sphere radius to its distance from the camera.

This analysis gives us three important algorithms for rendering large quantities of spherical particles:

1. an algorithm for testing whether an image plane point is inside the sphere silhouette or not.
2. an algorithm for computing the point on the disk edge that lies on the line between an inside point and an outside point.
3. an algorithm for mapping a point on the image plane inside the disk to the two corresponding points on the 3D sphere.

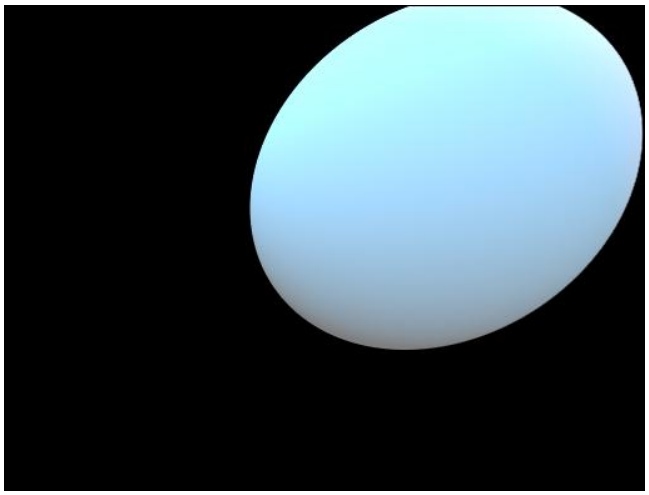


Figure 2: Image of a sphere demonstrating that the silhouette of a sphere is not circular or elliptical in general.

The first two results provide a method for doing clipping in the image plane, and the third one allows us to track back to points on the sphere from the image plane, without doing an explicit ray trace. Inspiration for these algorithms comes from the basic properties of ray-sphere interaction described in reference [8].

In the special case that the sphere is aligned on the axis of the camera view direction, the image plane shape is a circular disk, with radius $\epsilon_0/\sqrt{1-\epsilon_0^2}$, where the dimensionless variable ϵ_0

$$\epsilon_0 = \frac{R}{|\vec{\mathbf{r}}_S - \vec{\mathbf{r}}_C|} \,, \tag{1}$$

is less than one. Here, R is the radius of the 3D sphere, $\vec{\mathbf{r}}_S$ is the 3D position of the sphere, and $\vec{\mathbf{r}}_C$ is the 3D position of the camera.

2.1 Imaging a Sphere

The basic imaging equation maps points $\vec{\mathbf{r}}$ in 3D space to points $\vec{\mathbf{x}}$ on the image plane. Using the camera position $\vec{\mathbf{r}}_C$ and the camera pointing direction $\hat{\mathbf{n}}$, the imaging equation for the perfect camera is

$$\vec{\mathbf{x}} = \frac{\vec{\mathbf{r}} - \vec{\mathbf{r}}_C}{\hat{\mathbf{n}} \cdot (\vec{\mathbf{r}} - \vec{\mathbf{r}}_C)} - \hat{\mathbf{n}} \tag{2}$$

It is easy to verify that, although $\vec{\mathbf{x}}$ is written as a point in 3D space, the set of such points, generated as $\vec{\mathbf{r}}$ varies through the 3D space in front of the camera, all lie on a plane that has $\hat{\mathbf{n}}$ as its normal. Also, the units of $\vec{\mathbf{x}}$ are in dimensionless "tangent" units.

The points on the surface of the sphere have the representation

$$\vec{\mathbf{r}}_S + R\hat{\phi} \,, \tag{3}$$

where the unit vector $\hat{\phi}$ points in all directions. The imaging equation for the points on the surface is now

$$\vec{\mathbf{x}}(\hat{\phi}) = \frac{\vec{\mathbf{r}}_S - \vec{\mathbf{r}}_C + R\hat{\phi}}{\hat{\mathbf{n}} \cdot (\vec{\mathbf{r}}_S - \vec{\mathbf{r}}_C + R\hat{\phi})} - \hat{\mathbf{n}} \tag{4}$$

This is not a 1:1 mapping, but some additional effort below yields an explicit equation for the silhouette and the 3 algorithms of interest.

2.2 Mapping a Sphere to the Image Plane

The points $\vec{\mathbf{x}}(\hat{\phi})$ inside the sphere footprint in the image plane have the form $\vec{\mathbf{x}}(\hat{\phi}) = \vec{\mathbf{x}}_S + \delta\vec{\mathbf{x}}(\hat{\phi})$, with

$$\delta\vec{\mathbf{x}}(\hat{\phi}) = \epsilon \frac{\mathbf{M}\cdot\hat{\phi}}{1+\epsilon\hat{\mathbf{n}}\cdot\hat{\phi}} \tag{5}$$

and $\epsilon = R/\hat{\mathbf{n}}\cdot(\vec{\mathbf{r}}_0 - \vec{\mathbf{r}}_C) < 1$. The matrix $\mathbf{M}$ is a projection operator

$$\mathbf{M} = \mathbf{1} - \frac{\hat{\mathbf{r}}\hat{\mathbf{n}}}{\hat{\mathbf{r}}\cdot\hat{\mathbf{n}}} \tag{6}$$

with $\hat{\mathbf{r}} = (\vec{\mathbf{r}}_0 - \vec{\mathbf{r}}_C)/|\vec{\mathbf{r}}_0 - \vec{\mathbf{r}}_C|$.

2.3 Mapping from the Image Plane Disk to the Sphere

With a little more manipulation, equation 5 provides an algorithm for mapping a point $\delta\vec{\mathbf{x}}$ on the image plane back to the two corresponding points on the 3D sphere. That algorithm is constructed in this section. Some properties of the algorithm will serve in the other algorithms of interest.

To find points on the sphere, we must find $\hat{\phi}$ for any image plane point $\delta\vec{\mathbf{x}}$. The explicit expression for $\hat{\phi}$ is

$$\hat{\phi} = g\,(\delta\vec{\mathbf{x}} + \vec{\mathbf{z}}) - \frac{1}{\epsilon}\vec{\mathbf{z}} \tag{7}$$

with

$$\vec{\mathbf{z}} = \frac{\hat{\mathbf{r}}}{\hat{\mathbf{r}}\cdot\hat{\mathbf{n}}}\ . \tag{8}$$

There are two solutions for g, corresponding to the two points on the sphere that are intersected by a ray from the camera. The solutions are

$$g_\pm = \frac{\vec{\mathbf{z}}\cdot(\delta\vec{\mathbf{x}}+\vec{\mathbf{z}}) \pm \left\{(\vec{\mathbf{z}}\cdot(\delta\vec{\mathbf{x}}+\vec{\mathbf{z}}))^2 - \left(z^2-\epsilon^2\right)|\delta\vec{\mathbf{x}}+\vec{\mathbf{z}}|^2\right\}^{1/2}}{\epsilon\,|\delta\vec{\mathbf{x}}+\vec{\mathbf{z}}|^2} \tag{9}$$

2.4 The Silhouette as a Slice of a 3D Ellipsoid

Note that for g to exist as a real number, the quantity inside the square root in equation 9 must be positive. Points in the image plane which have a negative value for the radical do not map to the 3D sphere. So the silhouette of the disk is the collection of points which make the radical zero. Setting it to zero and rearranging terms a little, we can write the equation for the silhouette as

$$\xi(\delta\vec{\mathbf{x}}) \equiv (\delta\vec{\mathbf{x}}+\vec{\mathbf{z}})\cdot\mathbf{Q}\cdot(\delta\vec{\mathbf{x}}+\vec{\mathbf{z}}) = 0\ , \tag{10}$$

with the matrix $\mathbf{Q}$ being

$$\mathbf{Q} = \mathbf{1} - \frac{\vec{\mathbf{z}}\vec{\mathbf{z}}}{z^2-\epsilon^2} \tag{11}$$

Because $\vec{\mathbf{z}}$ and $\mathbf{Q}$ are functions only of $\hat{\mathbf{r}}$ and $\hat{\mathbf{n}}$, $\xi(\delta\vec{\mathbf{x}})$ is a quadratic function of $\delta\vec{\mathbf{x}}$ that corresponds to a 3D ellipsoid centered in 3D space at $-\vec{\mathbf{z}}$. Since the vectors $\delta\vec{\mathbf{x}}$ always lie in the image plane, the silhouette is the shape obtained by slicing the 3D ellipsoid with a plane co-located and co-oriented with the image plane.

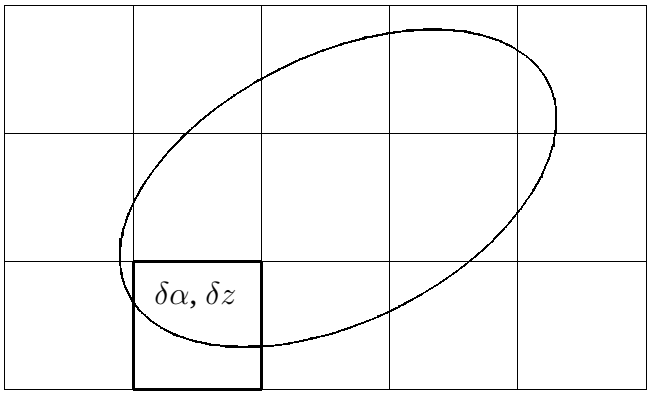


Figure 3: Outline of a sphere projected onto the image plane.

2.5 Testing for Inside/Outside Status

The algorithm to test for whether a point in the image plane in inside the disk or not is based on the sign of the function in equation 10. Writing this quantity as

$$T(\delta\vec{\mathbf{x}}) = |\delta\vec{\mathbf{x}}+\vec{\mathbf{z}}|^2 - \frac{|\vec{\mathbf{z}}\cdot(\delta\vec{\mathbf{x}}+\vec{\mathbf{z}})|^2}{z^2-\epsilon^2} \tag{12}$$

the test for inside/outside conditions is

$$\begin{array}{ll} T(\delta\vec{\mathbf{x}}) & > 0 \quad \text{Outside} \\ T(\delta\vec{\mathbf{x}}) & \leq 0 \quad \text{Inside} \end{array}$$

2.6 Intersection of a Line and the Silhouette

We consider the following problem, related to clipping polygons with the disk: Suppose the image plane point $\delta\vec{\mathbf{x}}_0$ lies in the disk ($T(\delta\vec{\mathbf{x}}_0)$ tested negative), and $\delta\vec{\mathbf{x}}_1$ lies outside the disk. What is the point along the line segment between these two points that lies on the edge of the disk?

To solve this, the points on the line segment are parameterized as $\delta\vec{\mathbf{x}}(t) = \delta\vec{\mathbf{x}}_0 + t\delta^2\vec{\mathbf{x}}$, with $\delta^2\vec{\mathbf{x}} = \delta\vec{\mathbf{x}}_1 - \delta\vec{\mathbf{x}}_0$. The points on the line segment are the collection for which $0 \leq t \leq 1$. Anything outside that range is not in the line segment. Putting this expression for $\delta\vec{\mathbf{x}}(t)$ into ξ, the result is a quadratic equation for t. Because $\delta\vec{\mathbf{x}}_0$ is inside the disk, we know that one of the solutions of this quadratic equation is position and one is negative. We only need the positive solution.

2.7 Alpha and Depth

Using the three algorithms built in this section, the silhouette of the sphere in the image plane can be evaluated to compute the fraction $\delta\alpha$ the sphere occupies in each pixel, as well as the average distance δz of the sphere from the image plane for the portion of the sphere in each pixel. Figure 3 illustrates the situation. Picking a pixel that includes the boundary of the sphere silhouette, the calculation of the occupied area requires identifying the vertices v_i. The number of vertices depends on the details of the intersection of the pixel with the silhouette, and can be between two and eight. The locations of the vertices are computed using the algorithm of section 2.6.

In some simple situations, the fraction $\delta\alpha$ can be analytically computed, e.g. when the sphere lies entirely inside one pixel, or completely occupies a pixel. In more complex conditions, it is adequate to tesselate the silhouette within a pixel and compute the area from the tesselation.

3 Large Numbers of Particles

In this section we extend the ability to accurately render a single sphere into a procedure for rendering large (in fact unlimited) numbers of particles. The goal is to render the particles with as little RAM as possible. Typically the particles will be small in size, and frequently unresolvable. Fine structural detail and bulk object shape in an image arises from the cumulative affect of many tiny particles.

One approach to rendering large numbers of unresolved particles is to treat them as if they are point-like. In that case the image of each particle is a point spread function presumably related to the camera optics or some other criteria. But while they are tiny, we do not want to assume that the particles are point-like for several reasons: (1) the volumetric nature of a cloud of particles requires some description of the tiny but not zero size of the particles; (2) as a particle approaches the camera in an animation, its size may become resolvable.

The approach we adopt here combines sphere rendering algorithms with a statistically-motivated scheme to track the distribution of alpha and depth. Updating the alpha and depth channels appropriately for each new particle, the procedure keeps in memory only one sphere at a time, without sorting the spheres or explicitly knowing the location of the spheres relative to each other. This transforms the rendering of spherical particles to a constant memory-size problem, with the heap size dominated by the storage of the image plane data.

The tradeoff for the memory efficiency of this algorithm is that the conditions underlying the statistical motivation are sometimes invalid. In this case, the rendered image structure can depend on the order in which particles are rendered. While we have seen such order-dependence in some scenes, it appears that in many applications this does not harm image quality.

3.1 Rendering and Statistics

Concentrating on a particular pixel, the N small particles that are imaged in that pixel (each particle either is wholly contained or overlaps with the pixel of interest) are characterized by three quantities: the fraction $\delta\alpha_k$ ($k = 1, \ldots, N$) of the pixel that the particle occupies (irrespective of possible occlusions by other particles), the distance δz_k of the relevant portion of the particle from the camera, and the intensity δi_k that the particle may contribute to the pixel.

In a "deterministic" world, in which complete information is know for all of the particles at any time, the approach to rendering these particles might be

1. For each particle, compute the fraction of the pixel that contains the *visible* portion of the particle $\delta\alpha_k^V$, taking into account occulsion by all other particles. Of course, $\delta\alpha_k^V \leq \delta\alpha_k$, and the calculation of $\delta\alpha_k^V$ is much more complex and time consuming that the calculation of $\delta\alpha_k$ and δz_k.

2. Compute the intensity and alpha for the pixel as

$$i = \sum_{k=1}^{N} \delta i_k \, \delta\alpha_k^V \tag{13}$$

$$\alpha = \sum_{k=1}^{N} \delta\alpha_k^V \tag{14}$$

The depth data δz_k does not explicitly enter this computation, but it is implicitly used in the process of sorting the particles to determine each one's visible area.

In the statistical rendering approach, we accept some loss of image precision because the visible alphas $\delta\alpha_k^V$ are not computed. Instead, using only the per-particle values, a "best estimate" of the contribution of each particle is made. The process is iterative. To perform the iteration, we augment the set (α, i) with iterative estimates of the mean distance z and the mean square distance $z2$. Initial values for these quantities in a pixel are set as $i_0 = \alpha_0 = z_0 = z2_0 = 0$, with the subscript 0 denoting the value before particles are rendered. The iterative procedure updates the pixel data when there is a statistically reasonable chance that a particle is visible relative to the other particles that have been rendered up to that point. After N particles have been rendered, the average distance to the collection of particles in the pixel is $z_N + (1 - \alpha_N) z_{\max}$ and the statistical thickness of the collection of particles is $2\sqrt{z2_N - (z_N)^2}$. The distance $z_{\max}$ is the far-plane cutoff distance for the render.

Suppose N particles have been rendered in this scheme, generating pixel data α_N, z_N, i_N, and $z2_N$. Particle $N + 1$ has data $\delta\alpha_{N+1}$, δz_{N+1}, and δi_{N+1}. The first step is to determine whether at least some fraction of this particle is visible to the pixel. The decision is a two-step tree:

1. Is $\alpha_N < 1$? If so, then there is some chance that the particle can be seen in the portion of the pixel's area that has not been filled, and the pixel data should be updated with the particle's data. If $\alpha_N = 1$, then the second test is employed.

2. Since $\alpha_N = 1$, is $\delta z_{N+1} \leq z_N$? If yes, then the particle is located closer to the camera than at least some of the particles that have contributed to the pixel data. In this case, the pixel data should be updated with the particle data. If $\delta z_{N+1} > z_N$, the particle is located behind most of the visible particles in the pixel, and the pixel data is not updated.

For updating the pixel data, the particle data is weighted by its relative size in the image plane:

$$i_{N+1} = i_N(1 - \delta\alpha_{N+1}) + \delta i_{N+1} \, \delta\alpha_{N+1} \tag{15}$$

$$\alpha_{N+1} = \alpha_N(1 - \delta\alpha_{N+1}) + \delta\alpha_{N+1} \tag{16}$$

$$z_{N+1} = z_N(1 - \delta\alpha_{N+1}) + \delta z_{N+1} \, \delta\alpha_{N+1} \tag{17}$$

$$z2_{N+1} = z2_N(1 - \delta\alpha_{N+1}) + (\delta z_{N+1})^2 \, \delta\alpha_{N+1} \tag{18}$$

This iterative procedure clearly does not generate an accurate image of many types of objects for many rendering problems. However, there are two extremes of circumstance in which it works very well. First, the statistical case of many small particles, for which this scheme was designed. Suppose there is a large collection of particles which contribute to the rendered radiance in a pixel, the particles all being small compared to the size of a pixel. Also, with a large collection of particles, their positions can be thought of them as random. For two small particles with pixel areas $\delta\alpha_1$ and $\delta\alpha_2$ randomly located in the pixel, the most likely amount of area particle 1 will contribute is $(1-\delta\alpha_2)\delta\alpha_1$. This expression combines the area of particle 1, $\delta\alpha_1$, with the probability that particle 2 does not hide particle 1. Second, the case of large particles that occupy several pixels. For a pixel fully occupied by the sphere, $\delta\alpha_{N+1} = 1$, and the update steps 15 – 18 are the correct steps for replacing all previous information with that of the sphere.

The conditions under which this scheme does not work is the middle ground between those two: when a particle size is comparable to the size of the pixel. In this case, it is important to isolate which parts of a pixel the particle occupies in addition to the area fraction it occupies. This failure is easy to detect during rendering and looks qualitatively like aliasing. Once detected, it can be corrected using traditional antialiasing methods, for example pixel supersampling can reduce the error to an acceptable level.

Figure 4 shows an example rendering of over 470 million particles. For this simple case, the particles are flat shaded and there is

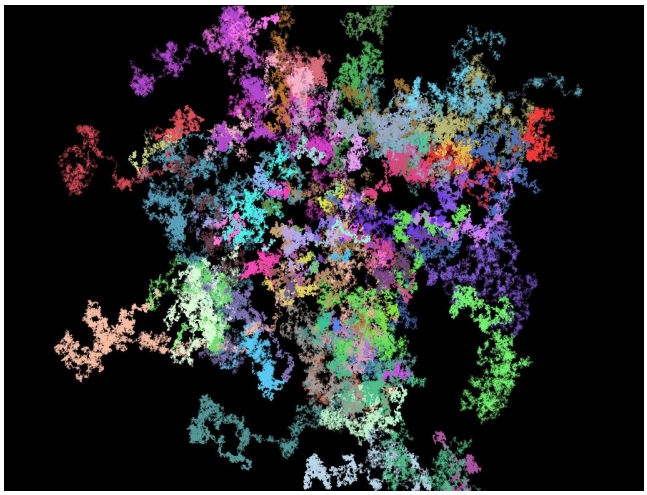


Figure 4: An image rendered using 473,750,116 particles. The algorithm for particle emission (see section 5 is a random walk with partial coherence in the random step. Rendered in approximately 64 hours on an MIPS R12000 195MHz, this 1024x778 image, required less than 30 MB of RAM.

no lighting. Each particle inherited the color of the guide particle it was spawned from (see section 5). The individual particles occupy an area of approximately 1% of the pixel area, depending on their distance from the camera. Coherent structure and fine detail are built up from the procedural details of emitting such a huge number of particles.

4 Statistical Thickness and Shadow Maps

As with any other rendered object, the visual appearance of a group of particles is strongly affected by the lighting and shadowing of the particles. For large numbers of particles distributed in some volume, shadowing must include the self-shadowing of some particles (partially) occluding others, which normally is attributed in a volume renderer to variations in density within the medium[9, 10]. Volumetric lighting and shadowing can be achieved with large numbers of particles, using the alpha and depth channels introduced in the previous section. However the amount of memory/disk space needed for this particle approach is dramatically less than that needed for both traditional volumetric rendering and deep shadow maps. In this section we demonstrate an algorithm for this, and illustrate it with a case in which fine detail is achieved in the presence of volumetric shadows, but with very little RAM used in the render.

As with more traditional techniques, shadowing here is accomplished by generation of a shadow map associated with each light that casts a shadow. Whereas traditional shadow maps consist of a depth map for the distance from the light objects in the scene, we use here the average alpha α, average distance z, and mean squared distance $z2$ from the light to the particles, computed as described in the previous section, for a camera that is co-located with the light. The light intensity at any distance Z from the light is $L = L_{\text{full}}\ t(Z) + L_{\text{shadow}}\ (1 - t(Z))$, where L_{full} is the full intensity of the light, L_{shadow} is the intensity of the light when fully shadowed, and $t(Z)$ is the transmissivity of the volumetric medium:

$$t(Z) = \begin{cases} 1 & Z < z - z_{rms} \\ \frac{Z - z + z_{rms}}{2 z_{rms}}(1-\alpha) & z - z_{rms} \le Z \le z + z_{rms} \\ (1-\alpha) & Z > z + z_{rms} \end{cases} \tag{19}$$

Here $z_{rms} = \sqrt{z2 - z^2}$ is the root mean square thickness of the volume of particles, from the point of view of the light.

Figure 4 illustrates the visual impact of this style of volumetric shadowing. In the top image the particles are all equally lit by a light which is located behind and above the collection of particles. The bottom image is the identical render, with the self-shadowing algorithm in equation 19 applied. The sense of bulk, depth, and thickness is much more pronounced with self-shadowing. There is detail in the shadowing that reflects the detail in the distribution of particles. The additional computational requirement to achieve this lighting is to render a three channel image of the particle distribution from the location of the light, the three channels being α, z, and $z2$. The size of the image plane needed to accomplish this quality is smaller than typically needed for shadowing by α alone because the additional channels provide more information. The deep shadow map technique enjoys a similar benefit. The images in figure 4, which were rendered at 2048×1556, used a shadow map plane of 1024×1024, with total RAM requirements of about 160 MB to hold image data, particle data, shadow data, and executable (no image plane tiling was used).

5 Particle Emission

The effort of rendering 10^9 particles can be complicated by hardware issues as well as rendering software design. The position and color of 10^9 particles would occupy about 15 GB of disk space or more, and disk access times could dominate total rendering time. The alternative we have chosen is to implement a shader-like language for building particle emission algorithms to generate particles at render time. This approach is based on "guide particles", which are a smaller set of particles whose behavior is animated and controled with traditional techniques. The attributes of each guide particle are combined with a particle emission procedural algorithm to drive the generation of many additional "child" particles at render time.

This combination of explicit modeling and proceduralism is very flexible. For example, the particles rendered in figure 4 are entirely procedural - only one guide particle was used to identify the location and orientation of the bulk particles. Figure 5 demonstrates another mixture of explicit particle modeling coupled with procedural particle emission. The 99 guide particles were modeled with variety of colors and emitting from a common source. Each image contains 10 million particles total, with the child particles following correlated random walks away from the guide particles. Varying the correlation coefficient from a small value (less correlated) to a value that approaches 1 (highly correlated) increases the structural organization of the collection of particles. As can be seen in figure 5, lower correlation coefficients produce particle distributions that appear to be cloud-like or nebula-like in appearance, while very high correlations produce string-like structures. Figure 6 demonstrates random walks on very long paths, while varying the correlation along the length of the path. Over 10^9 particles were rendered in this image, which required approximately 60 MB of RAM to render. Both string-like and nebula-like structures are present.

Figure 7 illustrates another procedural method for generating particles. In this image, 5 million particles were generated on a simulated water surface, propagated in time under the influence of gravity, then rendered with motion blur. Particle generation and propagation occured on the fly during rendering. For locations randomly picked on the surface, an algorithm incorporating the local

wave action was used to determine if a particle should be emitted. As is visible on the foreground wave, this procedure is capable of resolving emission from individual small waves riding on a larger wave.

6 Conclusions

We have shown in this paper that large numbers of particles can be efficiently rendered on modest hardware. The ability to accomplish renders with realistic lighting and shading opens several avenues of graphics applications, including fluid and volumetric visualization. Particle rendering in this fashion appears to be more efficient than traditional voxel-based ray marching methods for volume rendering, while delivering greater detail in the final imagery, especially in the detail of the self-shadowing. Our particular method of using the first two statistical moments of depth is a model of the distribution of particles along the line of sight, and certainly it does not represent the exact distribution. Enhancements to this statistical approach, such as computing higher order statistics of depth and alpha, may yield better results for some applications.

References

[1] Tom Lokovic and Eric Veach, "Deep Shadow Maps", ACM Siggraph Proceddings, 2000.

[2] Nick Foster and Dimitris Metaxas, "Modeling the Motion of a Hot, Turbulent Gas," Computer Graphics Proceedings, Annual Conference Series, 1997.

[3] Ronald Fedkiw, Jos Stam, and Henrik Wann Jensen, "Visual Simulation of Smoke," Computer Graphics, ACM Siggraph Proceedings, 2001.

[4] Kevin H. Martin, "Building a Better Borg," *Cinefex*, **76**, 1999.

[5] David E. Breen, Donald H. House, and Michael J. Wozny, "Predicting the Drape of Woven Cloth Using Interacting Particles," ACM Siggraph Proceedings, 1994.

[6] A Pumir, M. Chertkov, and B. Shraiman, "Lagrangian Tetrads: Geometry and Dynamics" Program on Physics of Hydrodynamic Turbulence (Jan 31 - Jun 30, 2000), Institute for Theoretical Physics, `http://online.itp.ucsb.edu/online/hydrot_c00/pumir`

[7] F. Battaglia, K. McGrattan, R Rehm, and H. Baum, "Simulating Fire Whirls," National Institute of Standards and Technology, NISTIR 6341, July 1999.

[8] Jeff Hultquist, "Intersection of a Ray with a Sphere," *Graphics Gems*, ed. Andrew S. Glassner, Academic Press, 1990.

[9] David S. Ebert, "Procedural Modeling of Gases," *Texturing and Modeling, A Procedural Approach*, Academic Press, 1994.

[10] "Volumetrics in Jig", `http://www.steamboat-software.com/Support/VolumeObjects`

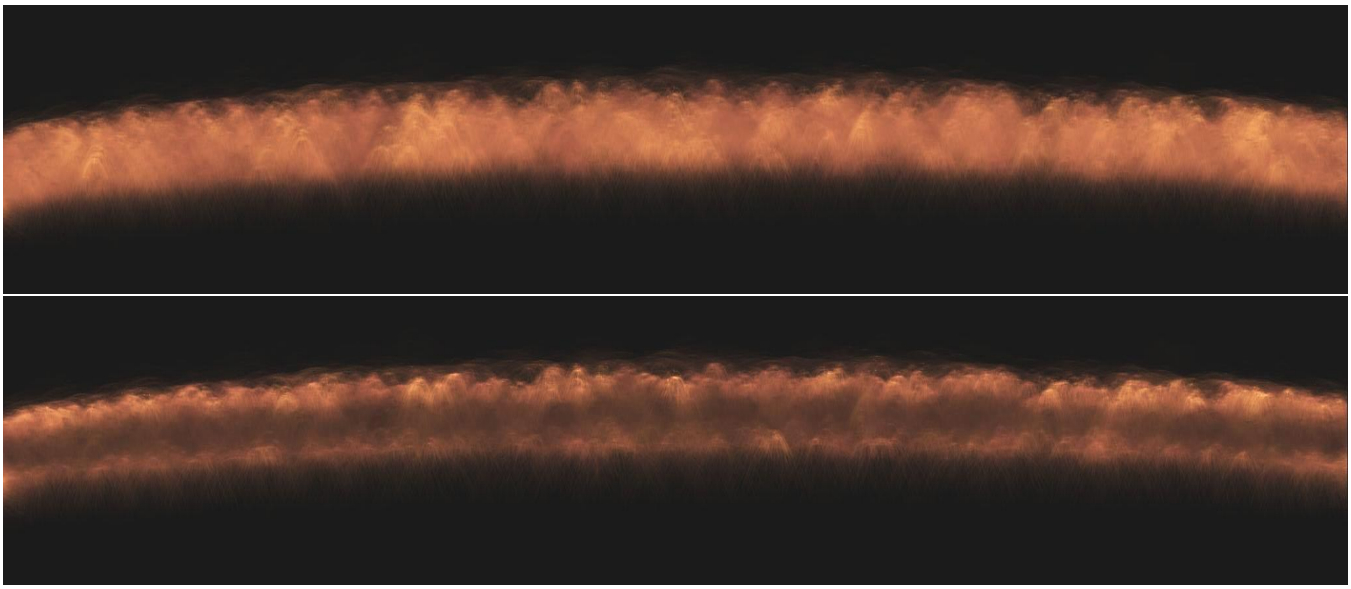
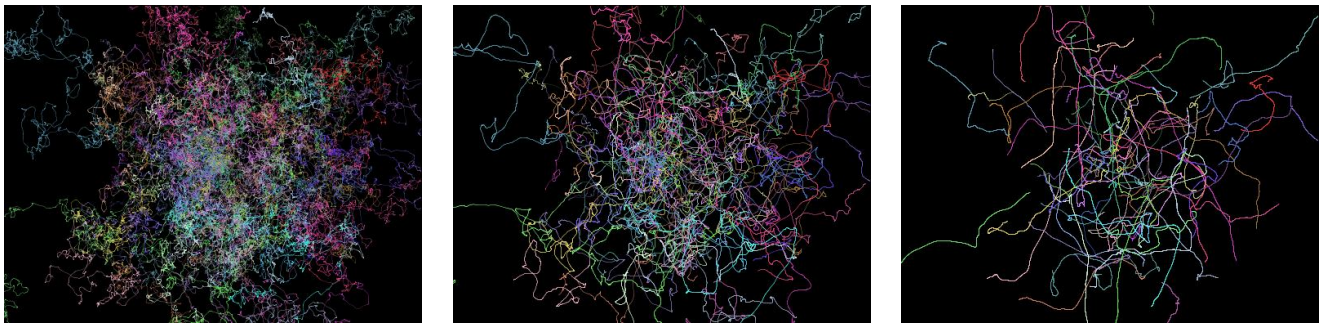


Figure 5: Top: An image rendered using over 4 million particles, with motion blur used to create extended structures. The particles are backlit, but without self-shadowing. Bottom: the same situation with self-shadowing.



(a) (b) (c)

Figure 6: Correlated random walks with various correlations. (a) Correlation is 0.99; (b) 0.999; (c) 0.9999. Each image contains 10 million particles.

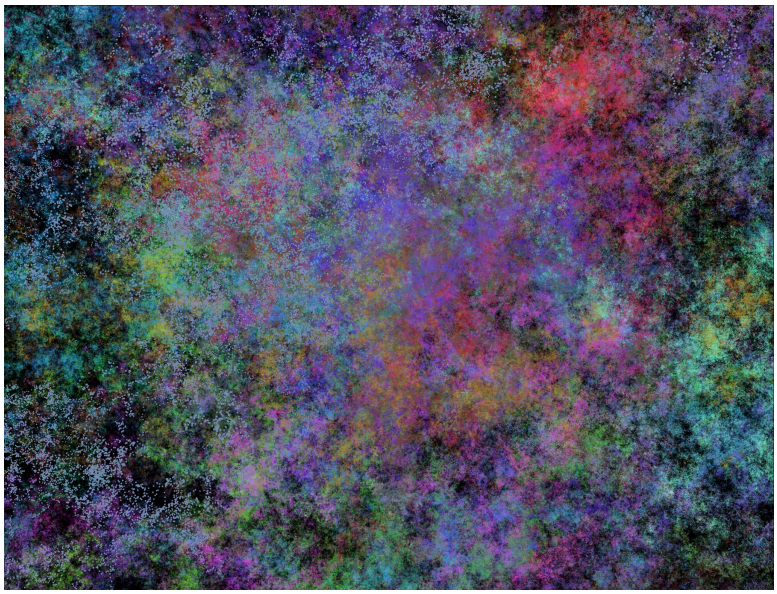


Figure 7: High resolution rendering of particles following random walks. In this image each guide particle was the starting point of a random walk for which the correlation of steps increased from near zero at the start of the walk to 1 at the end of the walk. There are 99 guide particles, and each walk is 10,300,000 particles long, for a total of 1,019,700,000 particles rendered in the image.

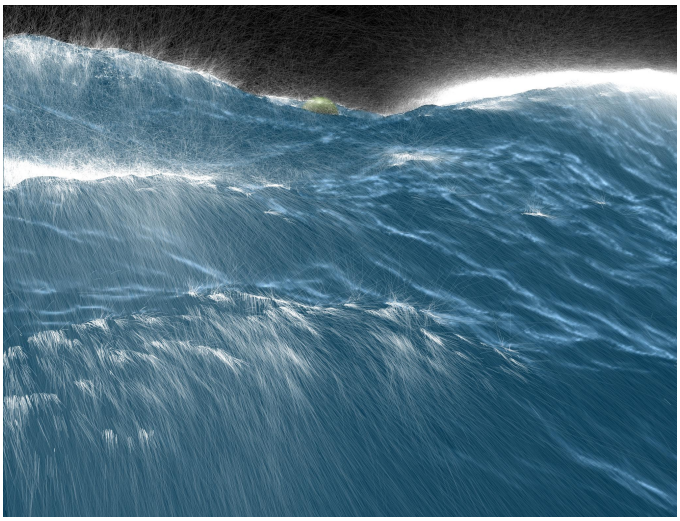


Figure 8: Simple use of large numbers of particles to create an environmentally rich effect. Particles are emitted from a cg water surface and propagate through space under the influence of gravity. Approximately 5 million particles were simulated for this image, although some of them have fallen below the water surface interface. While the number of particles is not extremely large, all of the particles were generated at render time, one at at time.