USE OF SILHOUETTE EDGES AND AMBIENT OCCLUSION IN PARTICLE VISUALIZATION

by

James Lawrence Bigler

A thesis submitted to the faculty of
The University of Utah
in partial fulfillment of the requirements for the degree of

Masters of Science

in

Computer Science

School of Computing

The University of Utah

August 2004

THE UNIVERSITY OF UTAH GRADUATE SCHOOL

SUPERVISORY COMMITTEE APPROVAL

of a thesis submitted by

James Lawrence Bigler

This thesis has been read by each member of the following supervisory committee and by majority vote has been found to be satisfactory.

Chair: Charles D. Hansen

Steven G. Parker

Peter Shirley

THE UNIVERSITY OF UTAH GRADUATE SCHOOL

FINAL READING APPROVAL

To the Graduate Council of the University of Utah:

I have read the thesis of James Lawrence Bigler in its final form and have found that (1) its format, citations, and bibliographic style are consistent and acceptable; (2) its illustrative materials including figures, tables, and charts are in place; and (3) the final manuscript is satisfactory to the Supervisory Committee and is ready for submission to The Graduate School.

Date	Charles D. Hansen Chair: Supervisory Committee

Approved for the Major Department

Chris R. Johnson
Chair/Director

Approved for the Graduate Council

David S. Chapman
Dean of The Graduate School

ABSTRACT

Particle visualization has traditionally borrowed techniques from grid based methods. Some grid based methods such as shadows and the Phong illumination model are easily applied to the particle renderings which often use iconic shapes such as spheres and ellipsoids to represent the data. While these approaches are appropriate for particle based data, not all grid based visualization methods are suitable for particle visualization. This thesis investigates the use of image based silhouette edges and ambient occlusion textures with respect to particle visualization demonstrating the effectiveness of these techniques.

To my wife, Laura

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

I would like to the thank the following individuals and organizations.

- Laura, my love and my life.
- My family, for all the love and support.
- My advisor, Chuck, for helping to provide the vision and direction that helped me get through this. I would not be here if not for him.
- Steve, for being such a great boss and mentor.
- Pete, he's the man!
- Christiaan Gribble, for a friendly ear and jointly writing much of the code used for the vicinity shading computation.
- Jim Guilkey, for data, feedback, and plenty of sass.
- SCI, for providing an atmosphere that makes collaboration happen.
- C-SAFE and DOE, for the funds to do this research.
- Data sets seen in Figures 3.3, 3.4, 3.7, 3.8, 3.14, 4.5, 4.6, 4.7, 4.8, 5.1 courtesy of Jim Guilkey.
- Data sets seen in Figures 1.1, 1.2, 1.3, 3.9, 3.10, 3.11, 3.12, 3.13, 3.15 courtesy of Biswajit Banerjee.

CHAPTER 1

INTRODUCTION

Computational simulation is a way to explore and understand the world around us. In order to do so, often the interaction of materials with their surroundings must be computed. This calculation is frequently done on grids, however, grids are not well suited for complicated geometry where the grid would have to be refined beyond computational capabilities. Implicit surfaces could be used to deal with the resolution problem, unfortunately the complexity of the equations defining intricate surfaces make their use prohibitive. Instead, structures made up of discrete pieces of material can be represented using particles.

Particle methods for computation have been used to simulate many different phenomenon in scales ranging from the evolution of the universe to the iteration of atoms on a molecular scale. In astronomical simulations, particles can represent large bodies of mass and are often used to compute N-body problems where forces such as gravity can be computed for each piece of matter. Atomic simulations represent individual atoms as particles that interact with the other atoms using such forces as atomic potential and attraction. Simulations in this scale seek to know how individual molecules are formed or how they react with other molecules. As the scale increases, the amount of matter represented by particles can change to accommodate the limits of computation.

One particular application of particles in simulation is the Material Point Method (MPM)[37, 38, 13]. Objects and materials in the simulation are represented with particles. This allows for arbitrary shape, and changes in this shape, as the simulation progresses. At the beginning of each time step, values from the particles are interpolated to a grid. This grid is usually of a coarser resolution than the particles meaning there are more particles in relation to node points. CFD computation is then performed on the grid and values are re-interpolated to the particles and their locations are updated. MPM has advantages over strictly grid based methods in that contact between material does not have to be explicitly computed as in a finite element approach[2].

One way to view the results of particle simulations is to interpolate values onto a regular grid and visualize the data using traditional grid based methods such as isosurfacing[27] and direct volume rendering[26]. However this has several disadvantages. Areas where the particles are sparse can result in incorrect structure reconstruction missing features too fine for the chosen grid resolution. To compensate for this, grids must be refined beyond the resolution of the particles. Also, memory is wasted representing areas that do not need refinement, such as regions without data or where the values are varying smoothly. Representing particle data as a regular grid can also mask the fine structure which may exist in the original form. Certain types of particle data, such as molecular simulations, do not have an analog in grid based representations and thus pose a problem in using this representation.

It is clear there exists a need to view particle data in a way that is both appropriate and informative. What does this mean? Since particle data seeks to represent pieces of a larger whole, the ability to see and interpret the macroscopic structure created by these particles is vital. As well, the ability to view fine structure is important for particle visualization. Appropriate visualizations are those which add to the overall understanding of the data while minimizing erroneous interpretations of it. There are two main goals of particle visualization. The first is the structure which is represented by the actual placement and sizes of the particles. The second are qualitative trends in values associated with the particles such as mass, speed, or stress.

Particle visualization often takes the form of rendering the data as spheres to represent the location and size of the particles used in simulation. Color mapping scalar quantities associated with particles (such as mass, volume, and speed) is an

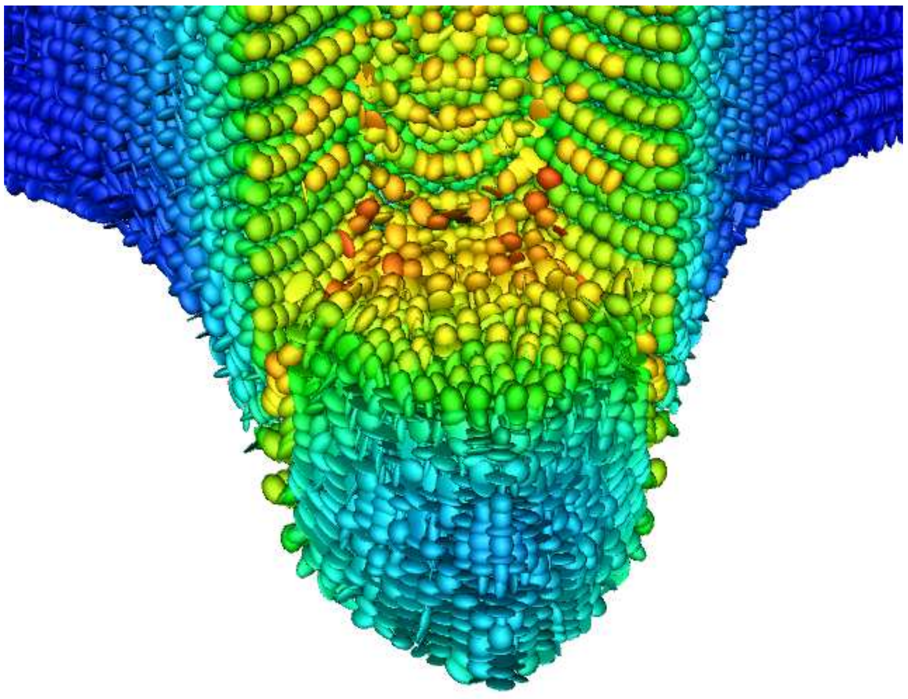

Figure 1.1. Tensor values can be used to deform and rotate spheres representing particles into ellipsoids.

additional method used to obtain a qualitative understanding of the data. Other methods of applying particle values to the dataset include particle deformation and rotation resulting in ellipsoid or super-ellipsoid primitives (Figure 1.1).

Lighting models can be used to provide spatial information. Renderings which make no use of lighting make it difficult to distinguish one particle from another (Figure 1.2). The use of a local lighting model such as Phong generated specular highlights can aid in seeing boundaries of particles and the three dimensional structure of individual particles, but can introduce other problems. One such problem is aliasing[7]. As the particles become near to or smaller than a pixel, features from the lighting model create high frequency changes in the images resulting in aliasing. This is distracting to the perception of the overall structure of the data as seen in Figure 1.3.

Local lighting models also fail to aid in the perception of the spatial relationships of particles with each other. This global information cannot be accurately captured

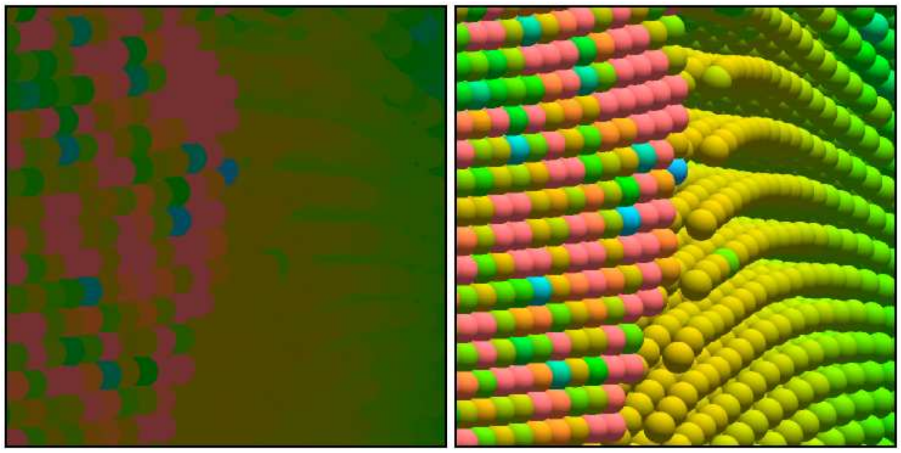

Figure 1.2. Local lighting models can help show particle boundaries. Using the same view and data, the left image had no lighting, the right includes phong shading.

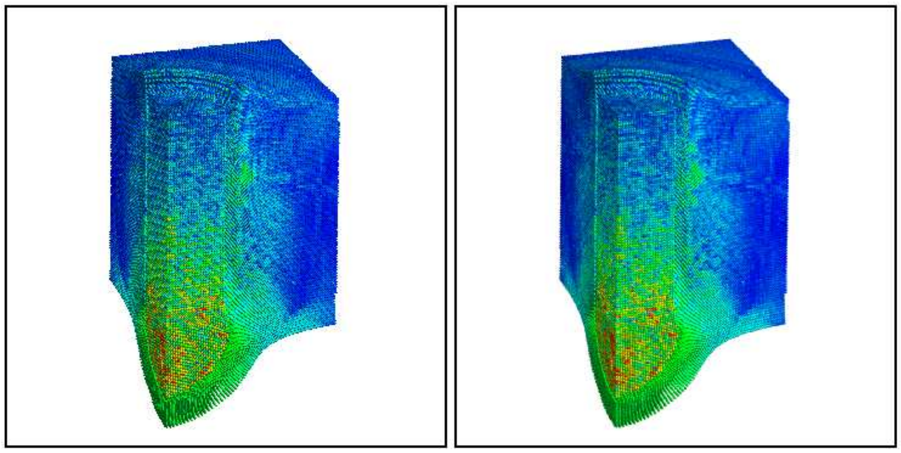

Figure 1.3. Aliasing can have a negative effect on the visualization process. The image on the left uses a single sample per pixel. The image on the right uses four jittered samples per pixel.

with a local model. As seen in Figure 1.4 using a global operation, such as shadows, can provide information about proximity of structures and help disambiguate relative positions of the geometry[45, 32]. Shadows, however, create regions where the ambient term of the shading model becomes dominate. With many local shading models, the ambient term becomes constant in regions of shadow or where the surface faces away from the light source, reducing the spatial cues needed for

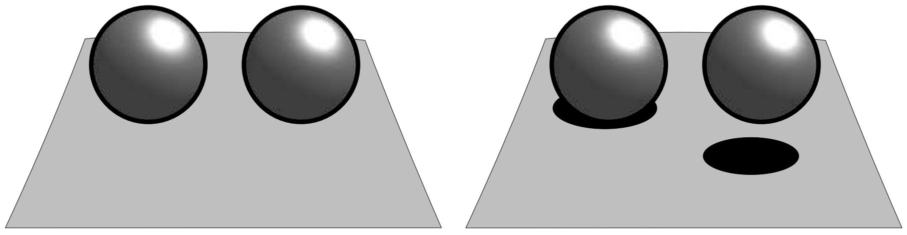

Figure 1.4. Without shadows, it is difficult to determine how far the spheres are from the surface (left). Shadows help disambiguate the spatial relationship between the spheres and the surface (right).

particle visualization. Nonconstant ambient values which take into consideration global positioning become important.

While illumination can help with shape perception, silhouette edges can be used to accentuate a set of view dependent boundaries[9, 11]. Our perceptual system makes use of stereo, object, and color information to distinguish boundaries of materials and objects[32]. While stereo is not necessary to find most boundaries, it can reinforce edge perception found by changes in color thereby creating a stronger sense of discontinuities. Since monoscopic renderings, such as those commonly used for computer displays, do not have stereo, we can augment boundaries of shapes with silhouette edges. Object based silhouette methods attempt to ascribe silhouettes to edges in the geometry. The effect of this method on particle based visualization is to accentuate the edges of each particle. This can lead to aliasing when each particle is only represented by a few pixels. The silhouette feature will be even smaller and distract from the overall representation of the data. Silhouette edges can be made more meaningful if they can provide information about the macroscopic shape represented by groups of particles.

CHAPTER 2

BACKGROUND

The background for this work covers many topics in computer graphics. Discussion of previous work will describe methods for precomputation of global lighting and ambient occlusion. Silhouette edge detection and application will then be discussed. Following this, an overview of the ray tracing rendering framework used for this thesis and related work. A discussion of particle visualization will follow.

Global illumination is a lighting model which takes into consideration the placement of neighboring geometry[18]. In his paper, Kajiya provides the definition of a rendering equation. This integral provides a model of the intensity of light from one point to another composed of the emitted light between these two points and the scattered light from the scene by the distant point all modulated by a geometric term. Simply put, this method of lighting provides not only direct illumination values, but light originating from indirect sources.

Brute force integration using Monte-Carlo integration can be used to evaluate the rendering equation, however, this method is computationally expensive and is slow to converge. There have been several efforts to approximate the equation to speed up computation of a single rendering or multiple views[46, 12, 17]. Greg Ward, *et al.* used a method of caching the indirect computation to speed up neighboring pixels[46]. A ray is shot into the scene. At the intersection point a search is made of previously computed indirect lighting. If found, these values are used rather than computing the indirect lighting. This provided a significant performance boost without introducing many artifacts.

Gene Greger, *et al.* used another method of caching illumination for later rendering[12]. Irradiance was stored at discrete grid locations for varying directions. When querying the grid for irradiance in direction v at point p, v is used as a lookup into the nodes surrounding p. This approximation provided for faster renderings when changing the view and the ability to move small objects in the scene. This method seems good for scenes where the grid is sufficiently resolved for the level of detail of the geometry. In the case of particle data sets generated by MPM simulations, this would not account for the fine structure.

Jensen developed a method called photon mapping to solve the rendering equation[17]. Packets of light, called photons, are emitted from the various light sources in the scene. As the photons intersect surfaces, a record is kept indicating the amount of incoming light the photon contains. Depending on the properties of the material at the intersection point the photon is attenuated and reflected back into the scene. Photon mapping can account for many complex lighting phenomena such as specular reflections, dielectrics, and translucent materials such as milk or marble that require sub-surface scattering. The collection of records made by the photons is called a photon map. Once the photon map has been generated, traditional ray tracing is used to render the scene and generate direct lighting computations including shadows. At the point of intersection, the photon map is queried for nearby records. The results of these records are added to the direct lighting component. One benefit of using a photon map, is the ability to render the scene from any viewpoint without having to recompute the photon map.

More recently global illumination has been looked at for the visualization of dynamic isosurfaces[47]. Irradiance is stored in a volume corresponding to the varying possible isosurfaces contained in the data. When shading the extracted isosurface geometry, irradiance is interpolated from nearby grid points and applied to the surface. To reduce interpolation errors, a grid of higher resolution than the data can be used for the irradiance.

While global illumination refers to the model which incorporates interreflection of light between surfaces, another lighting model called ambient occlusion or obscurances only computes the visibility of a point with a diffuse light source (such as a sky light source)[48, 30, 16, 24]. The motivation arose from the observation that ambient occlusion could give perceptually similar results to global illumination models with a small fraction of the computational cost.

Zhukov, *et al.* first introduced this concept with term obscurances[48]. The scene was first broken into discrete pieces. Given a neighborhood near the patch, the ratio of other patches obscuring the surface was computed. Computation was done as a preprocess with the obscurance values used as an ambient term during rendering. Results could be combined with arbitrary direct illumination, because the obscurance computation was a property of the geometry and not of the lighting conditions. The method was also useful for interactive applications, because the results were view independent. The work of Iones, *et al.* [16] builds on this work and provides additional implementational details about how to discretize the scene, use of illumination maps to store and utilize the results, and comparisons with other global illumination methods.

Hayden Landis detailed how ambient occlusion could be used to add realism to models[24]. Unlike global illumination, this precomputation would allow the model to be placed in any environment and lighting condition. This was achieved by storing the values of the obscurances at the vertices of the models. Additionally the principle direction of least obscurance can be stored to index into an environment map during rendering.

Méndez, *et al.* extended the use of obscurances to ray tracing[30]. Unlike previously mentioned methods, the ambient occlusion values were not precomputed and stored. Instead, the values were calculated during the rendering phase as an alternative to path tracing[18]. This greatly decreased the rendering time as compared to path tracing, as well as eliminated the need to break the scene into patches. The ambient occlusion values would have to be recomputed, however, every time the scene was rendered.

Vicinity shading was a term coined by James Stewart in his paper that dealt with precomputing the uniform diffuse illumination for isosurface volumes[36]. Similar to ambient occlusion for geometry, ambient values were computed for visualizing isosurfaces generated from volumetric data. His method, however, considered

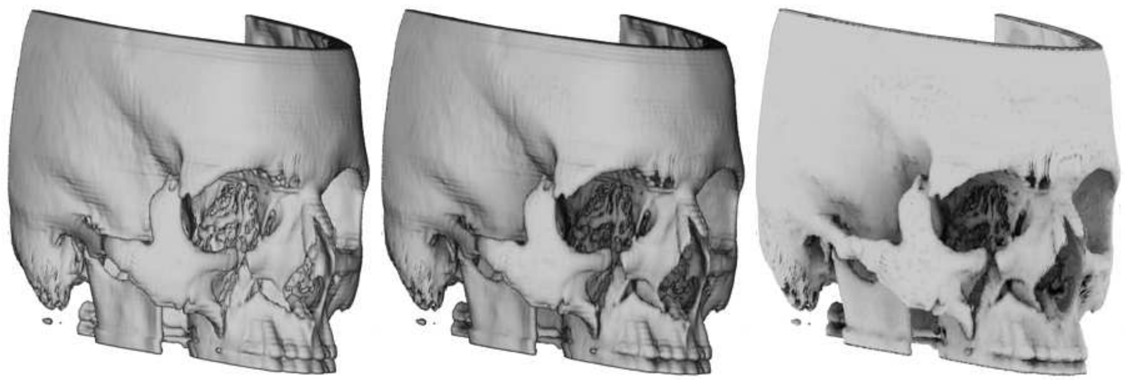

Figure 2.1. An isosurface representing a human skull with a fracture of bones in the right hand jaw. Vicinity shading helps the perception of space between the jaw bones and the rest of the skull. The image on the left has only phong illumination. The image on the right shows the vicinity values. The center image is the combination of the left and right images.*Image taken from [36] and used by author's permission*

only the geometry in the vicinity of the surface when querying for occluders. Luminance was computed for a given voxel based on the presence and distance of occluders within a certain radius. Occluders that were farther away resulted in more luminance than ones that were closer. The precomputed luminance was stored as vicinity shading values in a separate texture volume and interpolated to the extracted isosurface during rendering. His paper showed the utility of including an ambient term based on structure of the geometry in the lighting (see Figure 2.1). His work also presented clever methods to compute the vicinity shading values for all isosurfaces at the same time.

Silhouette edges are another method for enhancing shape perception of the scene[10]. These edges are usually rendered in black and follow the view-dependent hull of the object. This helps define macroscopic structure of the geometry. Saito and Takahashi used image based techniques to combination values from various buffers including the depth buffer to add features such as silhouette edges, hash marked shading, and contour lines[35]. Specifically they used the depth buffer to detect silhouette edges. Gooch, *et al.* use the cosine of the angle between the normal and the view direction as a lookup table into a 1D transfer function to produce silhouette edges[9, 11]. As the dot product nears zero the surface is increasingly colored with the silhouette color. This method does not take into consideration the curvature of the surface and therefore is subject to varying widths of edges. Edges on surfaces which have low curvature will have thicker silhouettes than edges where the curvature is high. Kindlmann, *et al.* addressed this issue by computing the curvature of the surface using the values in a volume to unify the edge width[20].

McCool used silhouette edges indirectly to compute shadows for interactive scenes[29]. His method rendered the scene from the perspective of the light and saved the depth buffer. Using this depth buffer as an image, the method would perform edge detection to find discontinuities. These discontinuities, representing silhouettes, would be used to generate geometry representative of regions that were in shadow. The notable aspect of McCool's algorithm is the use of the depth buffer to generate silhouette edges.

Parker, *et al.* provided a method to interactively view large particle data sets as spheres using interactive ray tracing and clever acceleration structures[33, 34]. The basis for the proposed research builds on this framework. The interactive ray tracer, hereafter referred to as RTRT (Real-Time Ray-Tracer), is built of two software components.

The first implements a traditional ray tracer where rays are shot into the scene where intersections are queried. The closest objects are shaded using an assortment of material properties such as phong shading or specular reflection. Objects in the scene are stored as lists or inside specialized acceleration structures that are designed to reduce the amount of work required to find the closest object[19, 6, 8]. The acceleration structure used for this work utilizes a hierarchical grid with the indices stored at the deepest level. Proper use of grids help reduce the number of ray/sphere intersections a ray must perform[21, 1].

The second component of RTRT provides the user interface and multiprocessor framework. User input, such as pan or zoom, is processed and the according changes are made to the rendering system. Rendered images are displayed in a window the user can view and interact with. Because ray tracing is a per pixel operation, parallelism is achieved by assigning groups of pixels to individual threads running on their own processor. The target architecture of RTRT is large distributed shared memory systems such as SGI's Origin series[33]. Once the rendering threads are finished computing, the result is handed off to the display thread. The display thread displays the result in a window for the user to view.

There has also been work to perform interactive ray tracing on clustered PCs instead of large distributed shared memory systems[4, 5, 39, 40, 41, 42, 43, 44]. Pixels assignments are divided among the processors in various ways and the results communicated back to a process that displays the results. The efforts of using clusters for interactive ray tracing involve load balancing, memory distribution, and communication. Many features of ray tracing such as global illumination and dielectrics have been shown to run within these frameworks.

Spheres are a natural representation of particles in visualization. Krogh, *et al.* used spheres to visualize particle data consisting of millions of spheres[22]. Individual spheres were used to give the domain scientists the ability to see the whole structure and zoom in to inspect fine detail. Using large parallel supercomputers, they were able to reduce the time to render a frame containing five million spheres down to a few seconds.

Particle visualization can also use tensors to modify the shape of the spheres into ellipsoids. Gumhold showed how texture splatting could be used on modern graphics hardware to render large number of ellipsoids representing tensor data interactively[14]. By making use of the programmability of modern graphics cards, depth correction can also be made to account for the overlapping ellipsoids.

This approach extended the work of Guthe, *et al.* who used texture splatting to represent particle flow in a vector field[15]. Objects such as arrows or comet trails are prerendered from different orientations and with varying lighting directions. During rendering the texture corresponding to the desired orientations is chosen and splatted on the screen.

Spheres representing particles have also been used to visualize flow fields[3, 23]. These particle visualizations use spheres to follow lines of flow in vectors fields. The systems used to render them must be able to sustain frame rates as well as interactivity. Computation is also done to define paths for each sphere as they are inserted in the field.

CHAPTER 3

AMBIENT OCCLUSION

Ambient occlusion values are similar to computing the incoming light from diffuse sky[48]. Langer, *et al.* discuss the perceptual ability to recover shape (relative spatial information) from shading on a cloudy day (computing visibility of a uniform diffuse light source)[25]. The ability to perceive shape can therefore be facilitated by the use of ambient occlusion. Previous work has shown ambient occlusion to be appropiate for life like scenes[48, 30, 16, 24]. It is therefore the work of this thesis to determine if ambient occlusion is appropiate for particle visualization.

In order to investigate ambient occlusion for particle data several components have been implemented: the precomputation of values, the storage, and reconstruction during rendering. The reconstruction of the shading values occurs during rendering and is therefore performance critical.

3.1 Data precomputation and storage

The framework for rendering is the interactive ray tracer, RTRT, developed at the University of Utah[33]. Particle visualization is accomplished by representing the particles as spheres because of the efficient ray/sphere intersection routine. Only the center and radius must be stored to perform the intersection test. The discussion of techniques to generate and apply ambient occluding shading information will pertain to spheres, since they will be used to represent particles during visualization.

3.1.1 Texture Mapping

Ambient occlusion values are precomputed and stored on the surface of the geometry. In the ambient occluding papers discussed in chaper 2, values were stored at the vertices or with textures[48, 30, 16, 24]. In Stewart's paper values

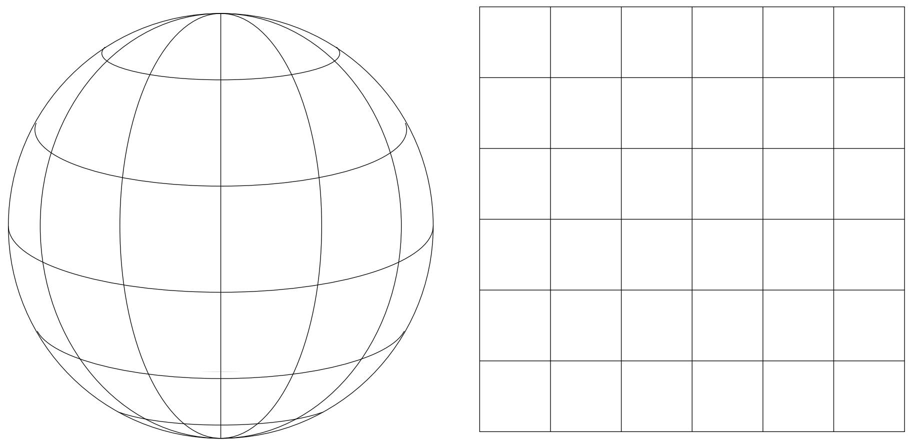

Figure 3.1. To texture a sphere using two a two dimensional mapping, one dimension corresponds to latitude the other to longitude.

were stored at voxel locations[36]. Since spheres, however, have no natural vertices or voxels, two dimensional texture maps will be used to represent shading values on the surface. Ambient values can be precomputed and stored in these texture maps. During rendering the textures will be mapped and interpolated onto the spheres.

To apply the textures the latitude/longitude coordinate mapping commonly used for spheres will be employed (Figure 3.1). An advantage to using this mapping is the fast computation of texture coordinates used during rendering. One disadvantage, however, is that the area of the texel is non uniform. This can result in areas of greater texture refinement where the texels are small and low refinement in areas where the texels are large. In practice, however, this is only noticeable when zooming the view point extremely close to the particles, which is not the common case. There are other mappings of two dimensional textures onto spheres that preserve area, but because texture coordinate generation during rendering of area preserving mappings can be costly, those methods were not explored.

3.1.2 Ambient value computation

With the mappings established, ambient shading values can now be computed. Particle locations are read into memory and the ray intersection acceleration structure is built. This is a modified version of a hierarchical grid with support for only

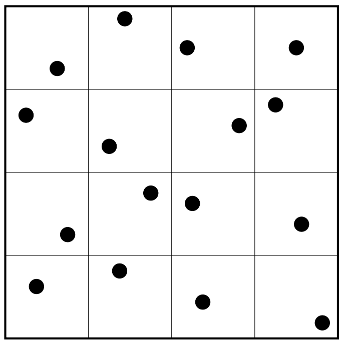

Figure 3.2. Jittered sampling divides the domain into regions and assigns a single point a random location for each cell.

spheres and the ability to crop particles based on values associated with them[33]. Precomputation time is based on the speed of ray intersections, so having a good acceleration structure is critical to making this approach tractable.

Once the acceleration structure is built, computation of the shading values can begin. Ambient value textures are computed on a per sphere basis, so work can be divided among multiple processors working in parallel. Being able to generate these textures in parallel is also important to making the precomputation phase reasonable (hours instead of days).

The texture used to store the shading values is broken up into individual texels. A texture resolution of 16 by 16 texels provides a nice compromise in resolution and computation time and was used for all the images generated in this document. Smaller textures would be faster to compute, but will blur features of the lighting. Larger textures would make the textures smoother, however, it will increase computation time. Texels are sampled a number of times, defined by the user, using a jittered sampling pattern to reduce aliasing on the texture (Figure 3.2). Samples are then mapped onto the surface of the particle creating the origin for a ray.

Ray directions are generated on the hemisphere corresponding to the normal at the surface point with a cosine distribution favoring the top of the hemisphere. Using this distribution prevents having to multiply the resulting luminance by the cosine of the direction and the normal[28, 46]. The hemisphere sampling takes two jittered values to provide faster convergence for Monte Carlo integration[31].

Once the ray direction and origin have been generated, a ray is shot into the geometry. If a hit occurres, then the particle is obscured from the background and no luminance is added to the texel. If no intersection occurres, the texel's value is incremented by the luminance of the background corresponding to the ray.

After the all the samples for a texel have been computed the value is then divided by the number of samples averaging the luminance of the background visible to the texel. This approach provides the ability to use backgrounds with non uniform luminance. Using a background with a light top and dark bottom could add perceptual cues similar to being outside on a sunny day. When the particle's texture is finished being created, the values are dilated and written to disk for later use.

3.1.3 Texture dilation

Dilation of the textures is needed to compensate for errors introduced by bilinear interpolation of the texel values. This error occurs when neighboring texels are inside another particle, and hereafter referred to as being inside. Those texels not inside are conversely referred to as being outside. Texels which are inside another particle are rightly black. Through the process of bilinear interpolation these black texels incorrectly darken the areas next to this boundary. The effect can be seen in Figure 3.3 where the particles touch.

Since the texels which are causing problems are inside other particles, they ordinarily should not contribute to the image. Their values can therefore be replaced by something more meaningful. By dilating values of texels outside to ones inside, these errors can be reduced significantly as seen in Figure 3.4.

The question becomes thus, how is a texel determined to be inside or outside? This determination is performed during the ambient value computation. The geometric scene is comprised solely of spheres whose normals always face outside. When an object is intersected the normal of the intersected object is also computed.

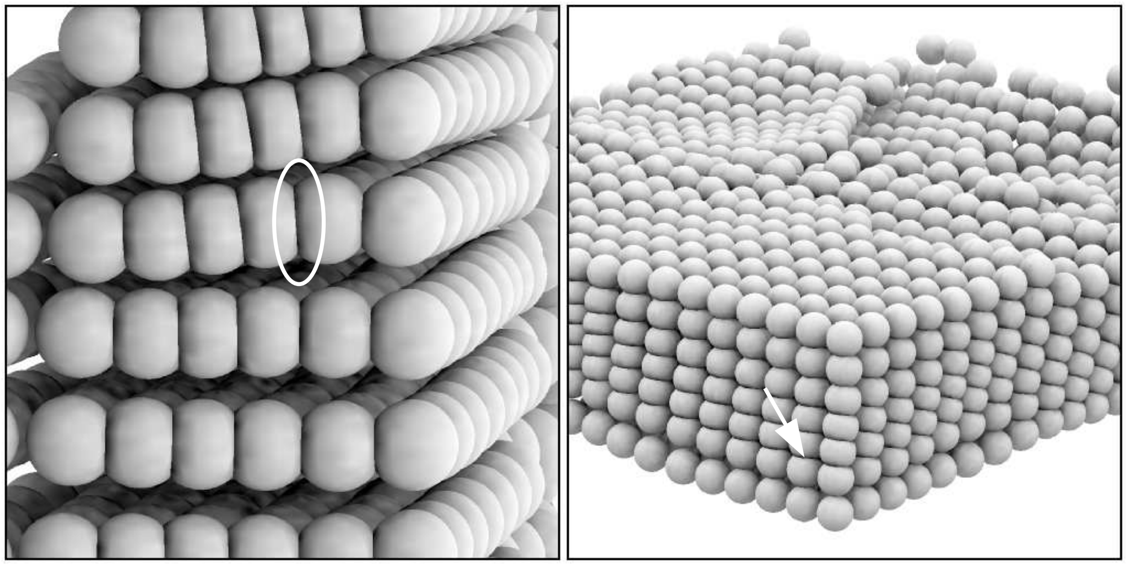

Figure 3.3. Bilinear interpolation can cause artifacts in regions where particles overlap. The dark banding that can be seen between the particles (examples highlighted with an ellipse and arrow) are caused by interpolating values that exists inside other particles with values that are outside.

By examining the angle between the direction of the ray and the normal of the intersected object, it can determined which side of the object was intersected. If the inside was intersected, it can be assumed that the ray originated from inside the sphere. This might not catch all cases as can be seen in Figure 3.5, but in practice some will register and be enough for the thresholding. A count is kept for how many rays from a texel were inside. This value is used later during the dilation phase to determine if a texel is inside or outside.

During a post process phase texels with an inside count above a user defined threshold are considered to be inside and are assigned an average of neighboring texels that are considered outside.

3.1.4 Final post processing

After all the textures for the data set have been generated, a final post processing step is performed. This involves two methods, gamma correction and quantization to eight bit integers. Gamma correction with a value of two is done to lighten up the darker regions for added perceptual acuity. Using the maximum value in

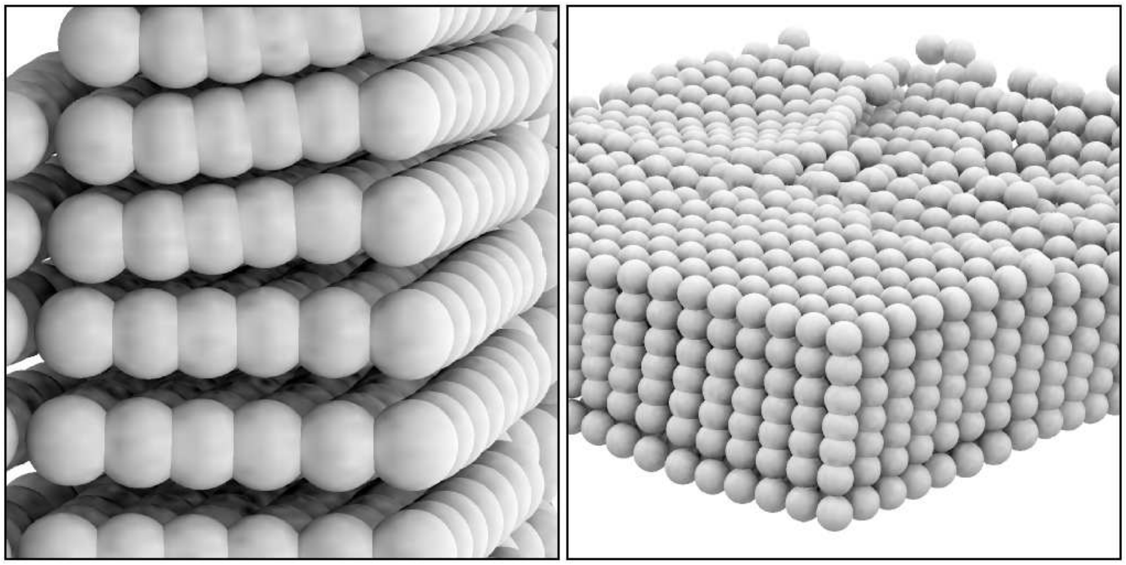

Figure 3.4. By dilating values that are outside to ones inside, interpolation errors can be reduced without introducing new errors.

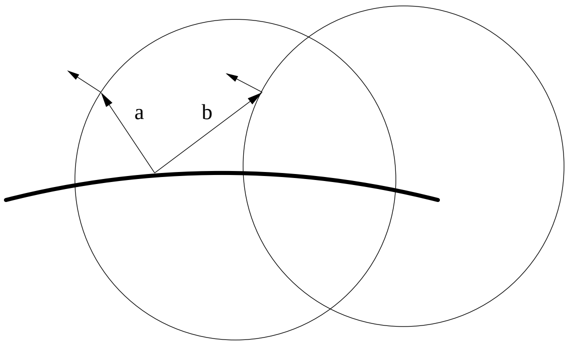

Figure 3.5. Determining if a ray is inside or outside by means of the dot product of the normal and ray direction works for ray a, but not for ray b.

the texture set, quantization is done to save space in memory when the textures are needed for rendering (a cost savings of four fold). Values in the textures are computed and written to disk as floats to maintain a reasonable precision for post processing operations, however, rendering does not require such precision and eight bits per texel is sufficient.

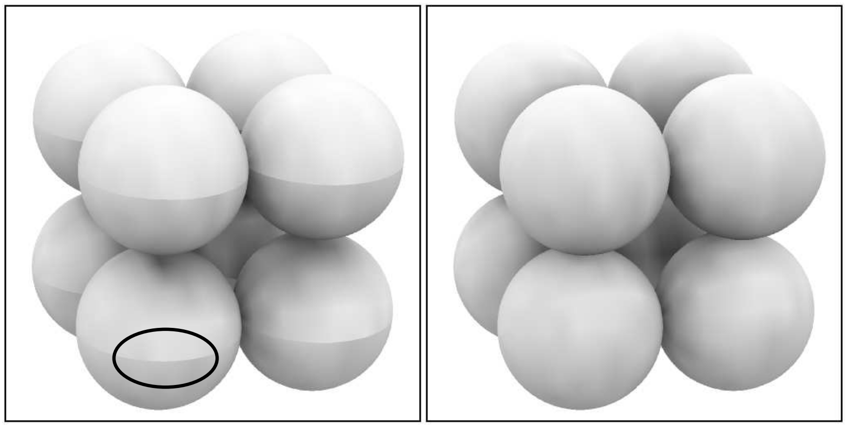

Figure 3.6. By wrapping the textures across the U parameter the seam can be eliminated. One of the seams has been highlighted with an ellipse.

3.2 Render phase

The particle data and texture data are read into memory from disk. The previously described acceleration structure is constructed and used to perform ray object intersections. Once a particle has been intersected its ID is used to index the textures in memory. The textures are in the same order as the particles, so that during this phase the appropriate texture can be used.

The intersection point and center of the sphere is used to calculate UV coordinates in the same manner as during texture generation to ensure the proper correlation. These coordinates are then used to lookup texels. Bilinear interpolation is used to give the textures a smoother appearance than nearest neighbor sampling would provide. Texels are wrapped along the latitude direction of the texture, but not the longitudinal. This helps eliminate the seam running along the side as seen in Figure 3.6.

3.3 Analysis

3.3.1 Precomputation Time

While precomputation time is not the focus of this research, an examination of the computational expense is provided. Precomputation time (T) is:

Table 3.1. Approximate time to generate textures for varying data sets. All times given are for 49 samples per texel with 16x16 (256) texels per sphere (12544 samples per sphere). Computation was done using 20 processors on an SGI Origin 3800 with 600 MHz R14K processors. Images using the textures are also referenced.

Name	Figure	Number of particles	Total number of samples	Time to generate
Fireball 6	3.7	955,000	11,979,520,000	66 min.
Fireball 11	3.8	952,755	11,951,358,720	261 min.
Cylinder 6	3.9	214,036	2,684,867,584	13 min.
Cylinder 22	3.10	212,980	2,671,621,120	25 min.
Bullet 2	3.11	569,523	7,144,096,512	35 min.
Bullet 12	3.12	543,088	6,812,495,872	33 min.
Foam 50	5.1	7,157,720	89,786,439,680	12 hours

$$T = \frac{1}{N_p} \sum_{i}^{N_s} T_i \tag{3.1}$$

$$N_s = \frac{\mathbf{X} samples}{texel} \frac{\mathbf{Y} texels}{sphere} T_s \tag{3.2}$$

where N_p is the number of processors computing in parallel, N_s is the total number of samples, T_i is the time it takes to compute sample i, and T_s is the total number of spheres. Table 3.1 provides experimental results for texture generation.

The difference in rendering times for data sets with relatively equal number of particles is due to the placement of particles. With MPM, particles start with an even distribution. As the simulation progresses, the particles move about creating regions of higher and lower densities. Grid based acceleration structures favor geometry of even distribution (provided the objects are of similar size).

While these precomputation times can be quite expensive, they are reasonable, provided the availability of large parallel hardware. Whether or not the computation cost can be amortized by the benefit to the visualization is not the focus of this research. Precomputation time can be reduced by either creating better performing ray intersection code or by decreasing the number of samples.

3.3.2 Impact on rendering time and memory

In order to make sure we keep the renderings interactive, the ambient occlusion values must be precomputed. Fifty samples per pixel computed during rendering would have too much of a negative impact on frame rates. Even a factor of ten would give an unacceptable reduction in performance. The ambient values are therefore stored in a texture. The impact on memory usage is directly related to the resolution of the textures used. A single texture is stored for each particle. Textures have only a single byte per texel representing the ambient value. If the resolution of a texture is 16x16, as in the images displayed in this thesis, each sphere would need 256 bytes. The memory requirements for a series of data sets can be found in Table 3.2.

Since interactivity is the goal of preprocessing and storing all the textures, the effect on frame rates must also be addressed. In Table 3.3 relative frame rates for various data sets. The views corresponding to the measurments are referenced as figures in the table. Using the ambient occlusion values does increase rendering times, however the impact is only about 10%. This is quite reasonable.

The last column of the table shows the frame rates using the ambient textures, but without the direct lighting computation. The textures provide some variance of lighting over the surface, and can be used without the addition of the direct lighting. Figure 3.15 shows a color mapped data set with and without the addition of direct lighting. The disadvantage of using only the ambient occlusion values would be the inability to use shadows, as this requires adding the direct lighting

Table 3.2. Extra memory required to store the ambient value textures for various data sets.

Name	Figure	Number of particles	Texture memory (Mega Bytes)
Fireball 6	3.7	955,000	233 MB
Cylinder 22	3.10	212,980	52 MB
Bullet 2	3.11	569,523	139 MB
Foam 50	5.1	7157720	1,747 MB

Table 3.3. Relative frame rates for a series of views and data sets are presented. Computation was done using 20 processors on an SGI Origin 3800 with 600 MHz R14K processors. The image rendered was 500x500 pixels except Bullet 2 and Bullet 12 which were 250x500 to match the images.

Name	Figure	Without textures (f/s)	With textures	Textures without direct lighting
Fireball 6	3.7.A	18.96 f/s	17.23 f/s	19.10 f/s
Fireball 6	3.7.C	13.89 f/s	12.70 f/s	14.40 f/s
Fireball 11	3.8.A	12.00 f/s	10.68 f/s	10.86 f/s
Fireball 11	3.8.C	9.09 f/s	8.49 f/s	9.45 f/s
Cylinder 6	3.9.A	16.77 f/s	15.07 f/s	16.47 f/s
Cylinder 6	3.9.B	9.86 f/s	9.23 f/s	10.26 f/s
Cylinder 22	3.10.A	15.22 f/s	14.04 f/s	15.14 f/s
Cylinder 22	3.10.B	8.19 f/s	7.84 f/s	8.35 f/s
Bullet 2	3.11.A	29.90 f/s	26.67 f/s	30.01 f/s
Bullet 2	3.11.C	26.43 f/s	24.50 f/s	27.56 f/s
Bullet 12	3.12.A	32.89 f/s	28.75 f/s	30.50 f/s
Bullet 12	3.12.C	24.62 f/s	22.66 f/s	26.32 f/s

component when a surface is not in shadow. As can be seen in the table, using only the ambient occlusion values can provide frame rates as good as or better than using only direct lighting. In many cases this can be preferable when direct lighting is not needed for shadows.

3.3.3 Perceptual results

By computing ambient occlusion values, global spatial positioning information can be encoded in the texture. This information is unique to each particle and gives a much better impression of the shape of the underlying structure than without it. Figure 3.13 shows an MPM data set with and without the ambient occlusion shading.

One benefit of computing only the ambient term is the ability to combine it with other lighting techniques such as shadows at rendering time. This provides the ability to dynamically change the lighting conditions used for direct lighting conditions to suit the needs of the visualization.

Another option for shadow or other direct lighting generation is to compute them when the textures are generated. This requires selecting a location for the light as well as an intensity that balances the light's contribution with the ambient values in the scene. If the light is made too bright, the ambient values will be too dark. Only the ambient values can be seen if the light is too dark. However, one benefit of computing the shadows as a preprocess is the computation savings of not having to shoot shadow rays during rendering. The savings here depends on the complexity of the geometry, however, since the rendering cost is independent of the content of the texture there is no additional rendering cost to include shadows in the textures.

One of the benefits of visualizing particle data sets with an interactive ray tracer is the ability to crop particles based on their value. This is an invaluable tool for domain scientists who wish to view different aspects of their data. The most common use is to crop the values based on physical location to view the inside structure. Since ambient occlusion shading is inexorably tied to the physical presence of all the particles, the shading no longer holds. Figure 3.14 and 3.15 illustrate this problem. Once a suitable cropping is obtained, however, the ambient occlusion information can be completely recalculated in another preprocessing step.

The use of ambient occlusion shading can greatly augment the perception of structure during visualization and is therefore appropriate. The ability of domain scientists to use it without having to tune rendering parameters is also beneficial. Precomputation time is tractable, however, it can be a deterrent for use of ambient occlusion. Being able to view time varying data in RTRT is a desireable feature, because the number of time steps that can be viewed is limited only by how much memory the system has available. Unfortunately, the memory requirements to store the textures will reduce the number of time steps that are possible to load into memory. The ability to view many multiple time steps iteractively makes viewing time varying data in RTRT desirable. The impact of using the textures on frame rates, however, are minimal and provide the same level of interactivity as without using them.

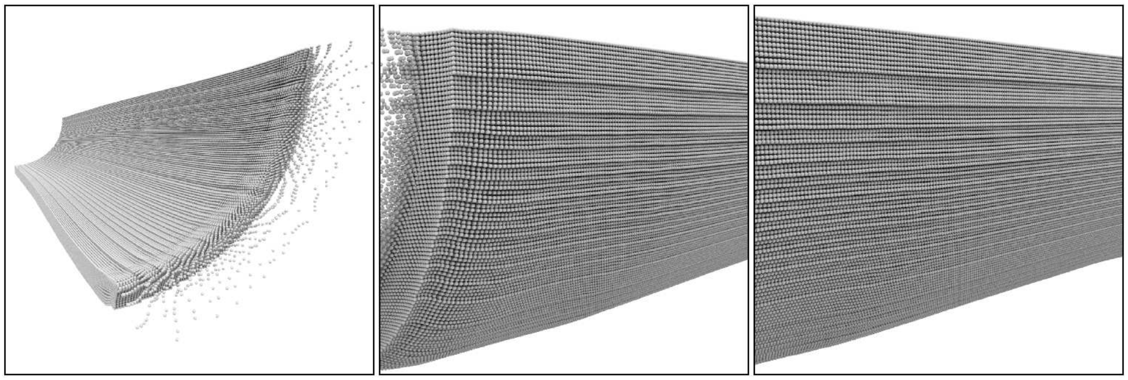

Figure 3.7. Ambient occlusion shading texture values mapped on the Fireball 11 data set. The views are identical to the ones in Figure 3.8. Sub-images are referred to as A, B, and C, left to right.

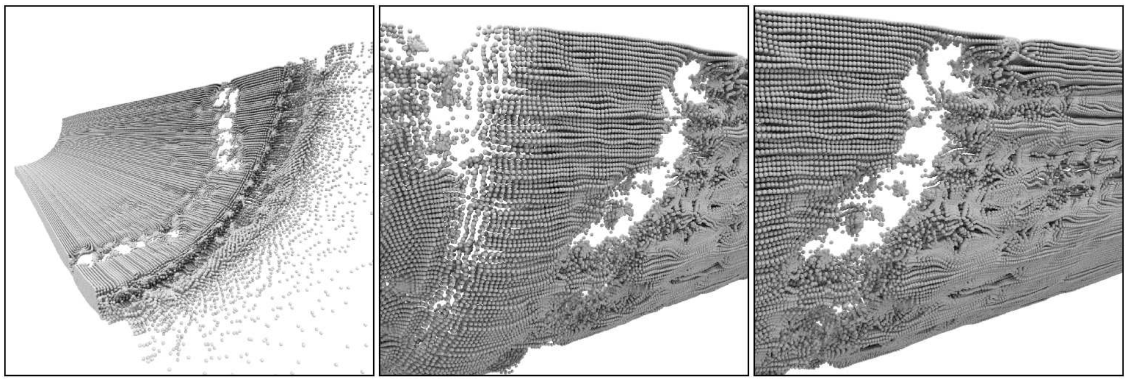

Figure 3.8. Ambient occlusion shading texture values mapped on the Fireball 11 data set. The views are identical to the ones in Figure 3.7. Sub-images are referred to as A, B, and C, left to right.

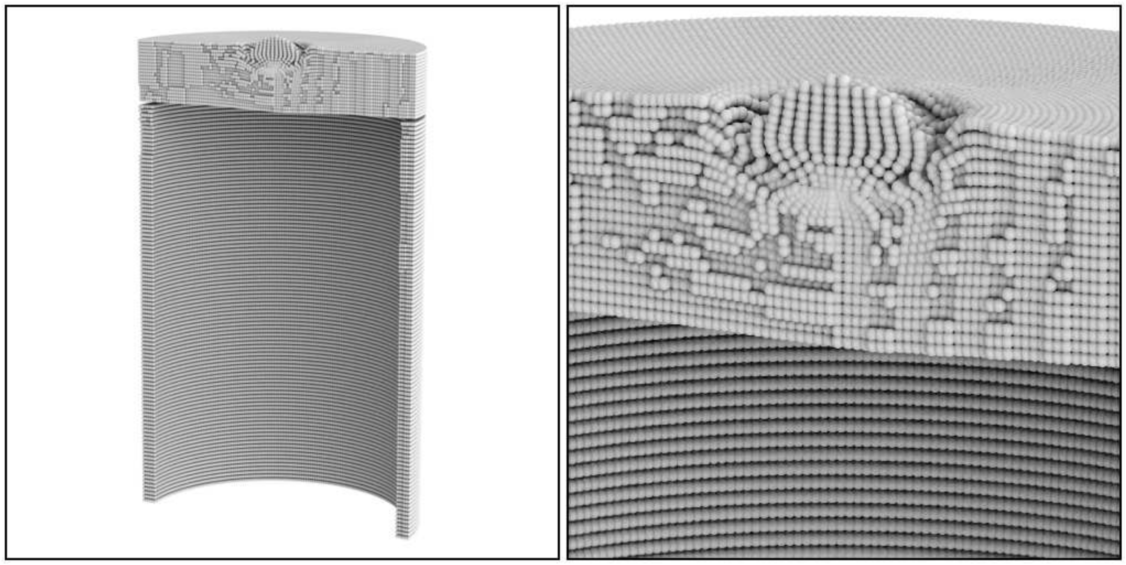

Figure 3.9. Ambient occlusion shading texture values mapped on the Cylinder 06 data set. Notice how the intricate structure is quite visible on the center of the structure. The spheres that protrude further from the front surface are clearly seen to do so. The views are identical to the ones in Figure 3.10. Sub-images are referred to as A (left) and B (right).

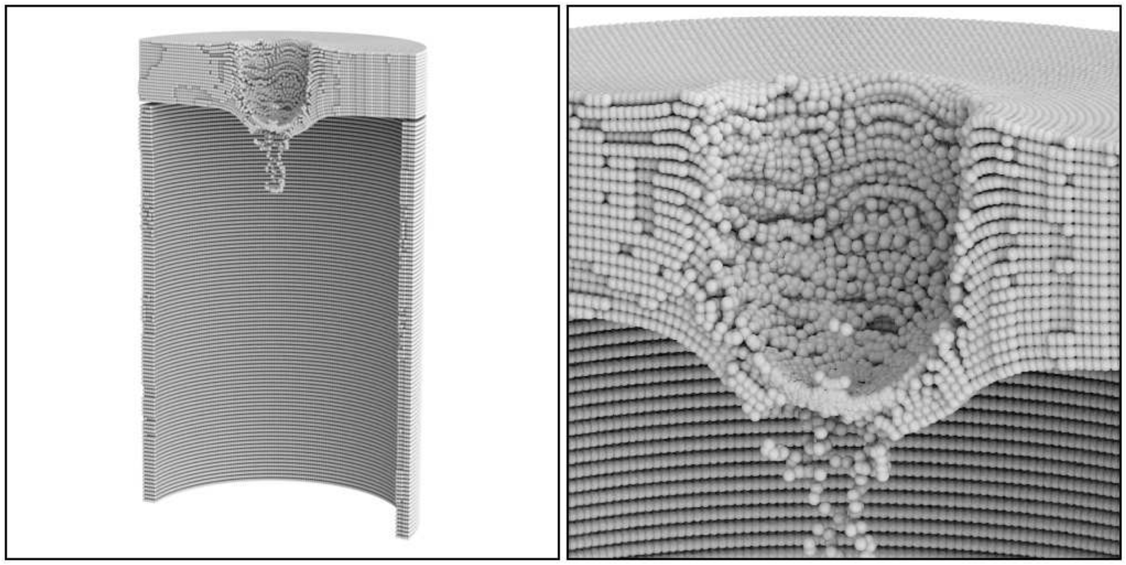

Figure 3.10. Ambient occlusion shading texture values mapped on the Cylinder 06 data set. One thing to note is the ability to see the two spheres standing out in front of the hole. The views are identical to the ones in Figure 3.9. Sub-images are referred to as A (left) and B (right).

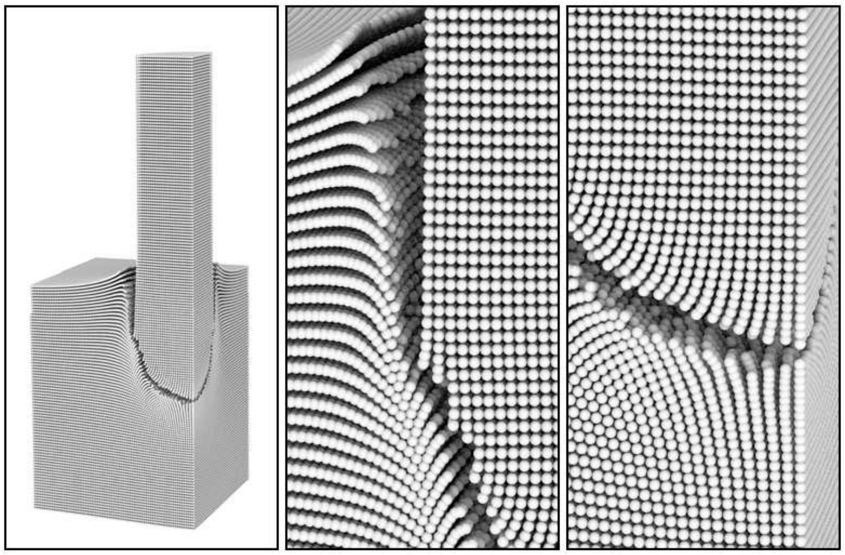

Figure 3.11. Ambient occlusion shading texture values mapped on the Bullet 02 data set. Sub-images are referred to as A, B, and C, left to right. Figure A is provides a view of the whole data set, while B and C show some close ups. The relative positions of the bullet with the material are easily perceived. The views are identical to the ones in Figure 3.12.

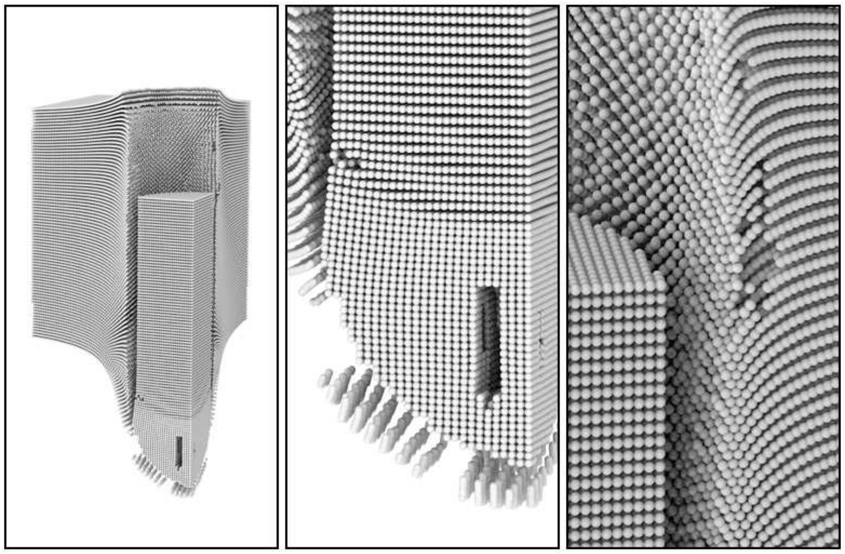

Figure 3.12. Ambient occlusion shading texture values mapped on the Bullet 02 data set. Sub-images are referred to as A, B, and C, left to right. Figure A is provides a view of the whole data set, while B and C show some close ups. As with Figure 3.11, the relative positions of the bullet with the material as well as inside the holes in the objects are easily perceived. The views are identical to the ones in Figure 3.11.

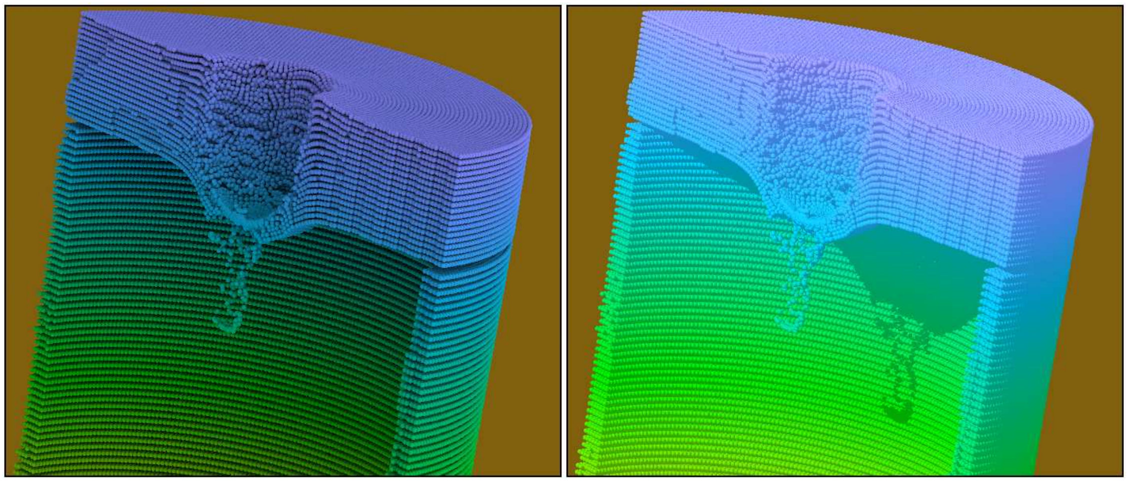

Figure 3.13. The image on the left is without ambient occlusion information. The one on the right includes the ambient shading.

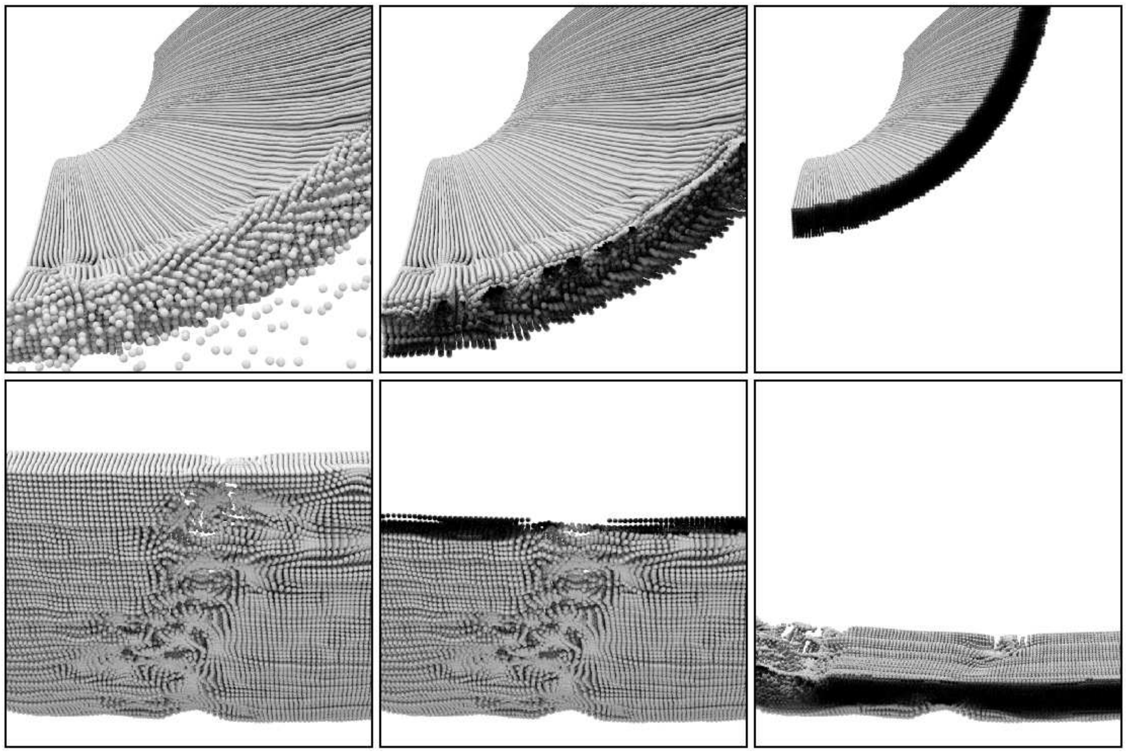

Figure 3.14. As the data is cropped, the ambient occlusion shading values are no longer applicable. The far left images show the uncropped data set. The center and right images show progressive cropping of the data. The internal particles that were not visible before the cropping are erroneously dark.

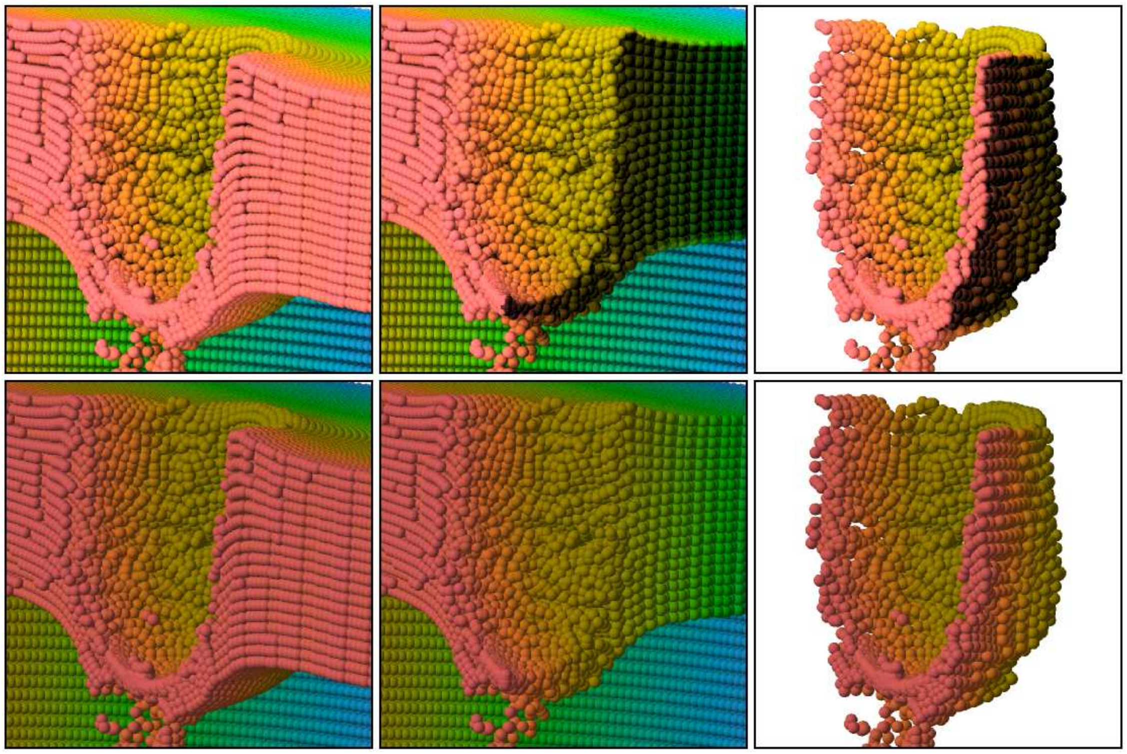

Figure 3.15. The images above show the visualization with the ambient occlusion values combined with direct lighting. The lower images use only the direct lighting contribution. The image on the left is the uncropped data. Images in the center and right hand sides show two different croppings.

CHAPTER 4

SILHOUETTES

Structures found in data sets generated using the material point method are generally represented using groups of particles. Silhouette edges placed on the edges of these structures can help to accentuate them. Since the macroscopic structures are made up of individual spheres, object based methods would need to account for the groups of particles and compute them as a preprocess.

An image based approach could be object agnostic as well as eliminate the preprocessing phase. Using a depth buffer as an image, edge detection can determine where there are discontinuities in depth. These discontinuities correspond to sharp changes in scene's geometry or edges of objects the occlude other geometry. By placing black pixels on the rendered image where discontinuities in the depth buffer exist, silhouette edges of objects will become more visible.

In order to perform image based silhouette edge detection, the depth buffer must be stored, edge detection performed, and black pixels marking these edges drawn on the image prior to displaying the final rendering.

4.1 Depth buffer from ray tracing

Ray tracing determines the closest object by comparing intersection points along a ray:

$$\mathbf{p}(t) = \mathbf{a} + t\vec{\mathbf{b}} \tag{4.1}$$

where $\mathbf{p}(t)$ is the position along the ray in terms of t, $\mathbf{a}$ is the origin of the ray, and $\vec{\mathbf{b}}$ is the direction of the ray. If $\mathbf{a}$ and $\vec{\mathbf{b}}$ are equal then comparisons can done using values of t, and the object corresponding to the smallest value is t is treated

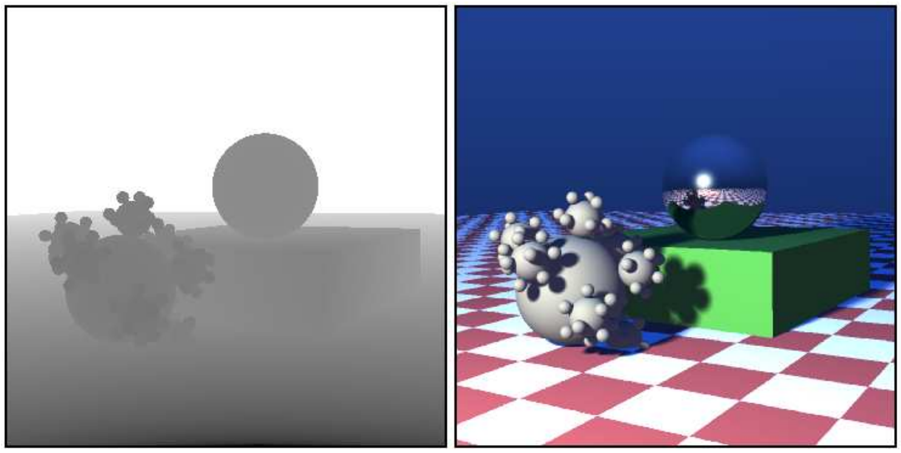

Figure 4.1. The depth buffer and the corresponding image generated using RTRT. The values for the depth buffer have been quantized in such a way as to make the foreground discontinuities more visible.

as the closest object. If the direction vector used is normalized and **a** is the same, the units of t between pixels are uniform and can be used in operations together. The values of t for each pixel can be thought of as a depth value; the collection of all t's as a depth buffer. While the depth buffer does not need to be explicitly stored during the process of generating the ray traced image, the information is already computed as part of the ray tracing process and can be cached for later use. Figure 4.1 shows an example of the depth buffer and image used to generate it.

4.2 Generating silhouette edges from the depth buffer

A popular method of computing edges in images is using the Laplacian of a Gaussian (LoG). This kernel is convolved with the image and edges are marked where there are zero crossings. The Gaussian is used to smooth the image first, because, as a second derivative kernel, the Laplacian is very sensitive to noise (see Figure 4.2). Convolution of the Gaussian is unnecessary, because the depth buffer is already relatively smooth. Therefore, only the Laplacian is needed to look for zero crossings in the second derivative. In the implementation used for this thesis it is done using a 3x3 kernel of the shape found in figure 4.3.

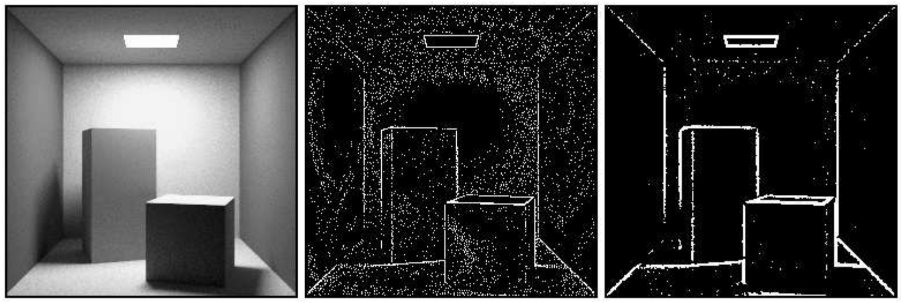

Figure 4.2. Second derivative kernels are very sensitive to noise, and usually require smoothing the input first to produce reasonable results. The image on the left is the original image. The center image is made by thresholding the Laplacian without any smoothing. The right side image smoothes the image using a Gaussian, convolves with the Laplacian, and thresholds the values.

-1	-1	-1
-1	8	-1
-1	-1	-1

Figure 4.3. Laplacian Kernel

Zero crossings are discovered by taking the difference between the center sample and the remaining surrounding samples. Since all depth values are positive, non zero differences indicate a zero crossing or an edge. One thing to take into consideration are double edges produced where there are discontinuities in the first derivative. To prevent these double lines, only differences which are positive are taken into account (see Figure 4.4).

Marking the silhouette edges of all the individual spheres can be desirable in some cases as can be seen in Figures 4.5 and 4.6. Here the silhouettes provide a trend in particle placement where lighting does not show the individual particles well or where the particles are in shadow and the corresponding sphere's color is constant.

In other cases this may result in too many silhouettes cluttering the image or generating aliasing (see Figure 4.7). To reduce the number of silhouettes, we must have a way of marking those edges which are of more importance. For our application the edges of more importance are those with higher discontinuities in depth. Using the magnitude of the Laplacian, edges where the discontinuity is greater than a threshold can be marked. By varying this threshold we can selectively show edges corresponding to different degrees of discontinuity. Figure 4.7 shows a series of images where silhouettes are drawn for edges of decreasing discontinuity.

The threshold can be used while rotating the object or changing the field of view without modification under certain circumstances. These conditions exists in two situations. The first is when the threshold used to determine edges is zero. No view dependent information is used in this case. The second scenario occurs when the relative depths of the objects remains nearly constant. This happens when the either the field of view changes or rotating about a point inside or close to the geometry such as the center. If the eye point moves away or towards the geometry, the threshold will need to be modified, because the relative depth values will change. Typical renderings of these particle datasets keep the view point towards the center of the object. A user will usually only rotate the object or change the field of view and not move the eye point closer or farther away, keeping the relative depth values meaningful to the depth threshold.

In RTRT edge detection is done on a frame boundary, because we cannot guarantee the availability of neighboring information until after all the rendering threads are finished computing their pixels. Only a single pass is necessary to perform the edge computation, as the Laplacian is computed for each pixel and edges assigned based on the threshold.

4.3 Analysis

One advantage of the silhouette edges is the lack of precomputation time. Since the method described in this thesis is entirely image based, no precomputation on the geometry is necessary. It does, hoever, require computation while rendering. For this particular implementation, edge detection is done in a separate thread from the image rendering allowing the computation to be overlapped with that of the rendering threads. This helps reduce the negative impact on frame rates.

Table 4.1. Relative frame rates for a series of views and data sets. Computation was done using 20 processors on an SGI Origin 3800 with 600 MHz R14K processors. The resolution of the images rendered is 500x500 pixels.

Figure	Without Silhouettes (f/s)	With Silhouettes
Figure 4.1.B	17.064 f/s	16.056 f/s
Figure 4.7.C	2.220 f/s	2.179 f/s
Figure 4.7.B	2.220 f/s	2.197 f/s
Figure 4.8.A and 4.8.B	1.155 f/s	1.162 f/s
Figure 4.8.C and 4.8.D	2.683 f/s	2.632 f/s

Table 4.1 shows the relative performance when computing silhouette edges while running. Based on the data in this table, the frame rates drop by only 2-5%. This is a negligible cost.

The impact on memory is based on the resolution of the image. Since the depth buffer is now stored, an extra four bytes per pixel are required to store the depth. The image buffer also uses four bytes per pixel to store rendering results (one byte for red, green, blue, and alpha channels). Windows typically used for rendering contain 40,000 to 360,000 pixels (200x200 to 600x600). At four bytes per pixel, the overhead to store the depth buffer ranges from a hundred kilobytes to two megabytes. On modern distributed shared memory systems, typically containing gigabytes of memory, the memory impact is very small.

Based on the experiments done, the use of silhouette edges is appropriate applied to particle data sets of this nature. Users can make use of it immediately during their renderings, and with the help of the user defined threshold, can have a device to aid in the visualization process.

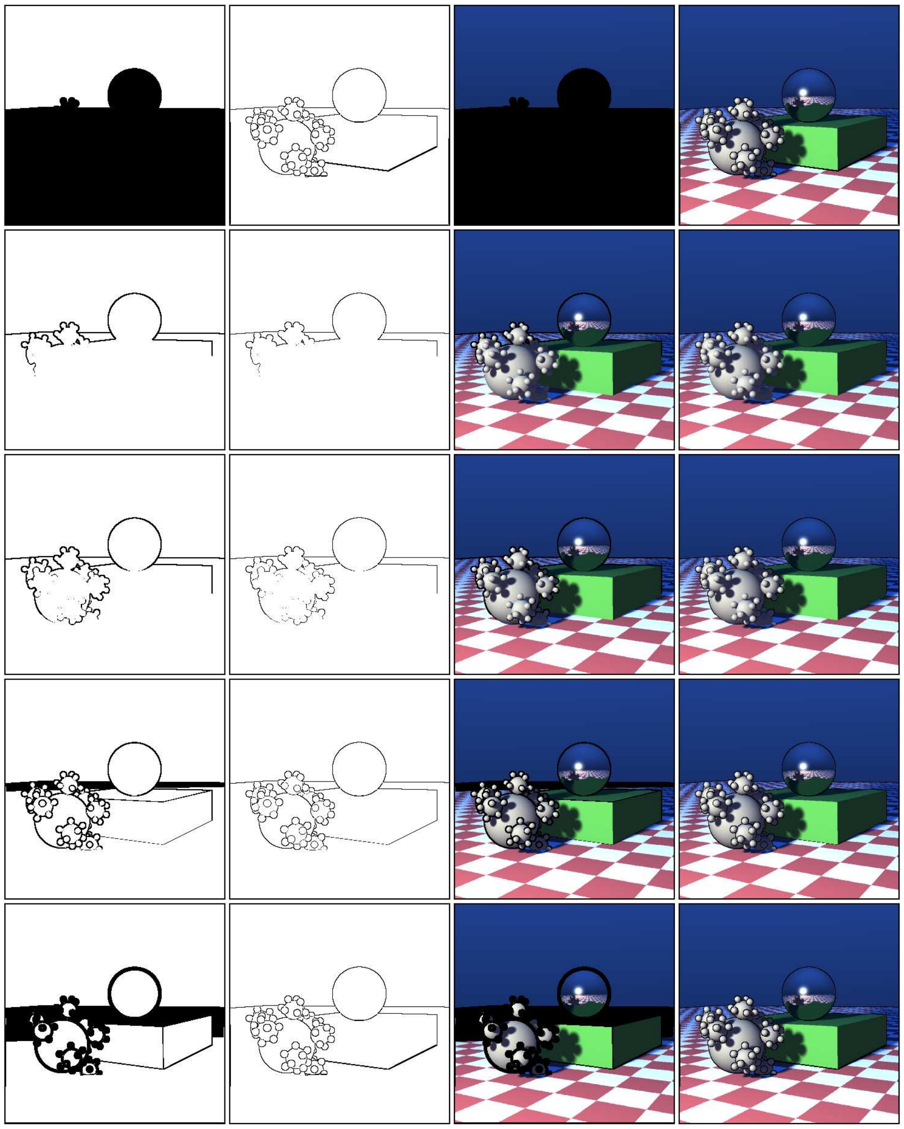

Figure 4.4. The left most column contain images where the edges are assigned by using the absolute value of the Laplacian compared with a threshold. The second column from the left is where only the positive values of the Laplacian are greater than a threshold. The right two columns of images show the edges applied to the rendered image. The top row of images compare the Laplacian to a threshold of 0, while the lower images use progressively smaller and smaller thresholds encompassing more and more edges.

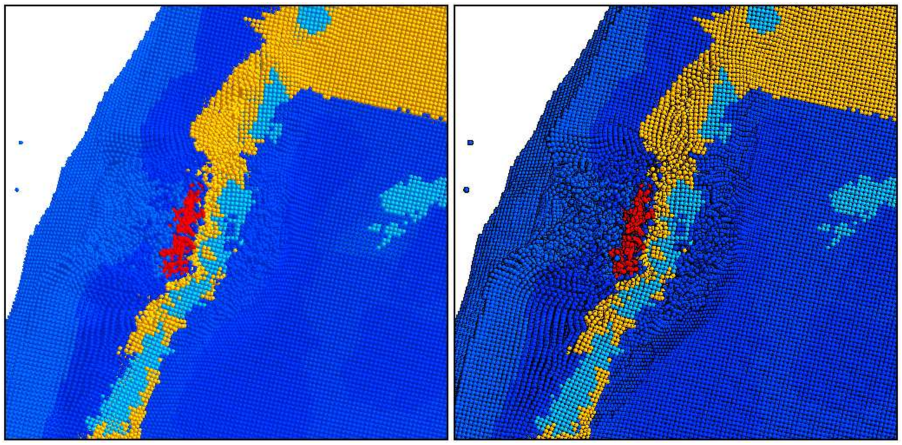

Figure 4.5. Silhouettes over each particle can be useful showing trends in particle placement, especially without shadows

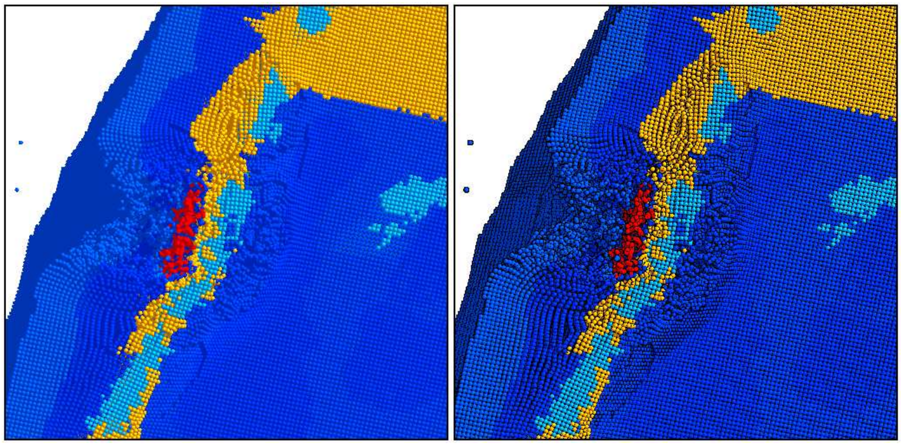

Figure 4.6. The same data and view as Figure 4.5, but with shadows

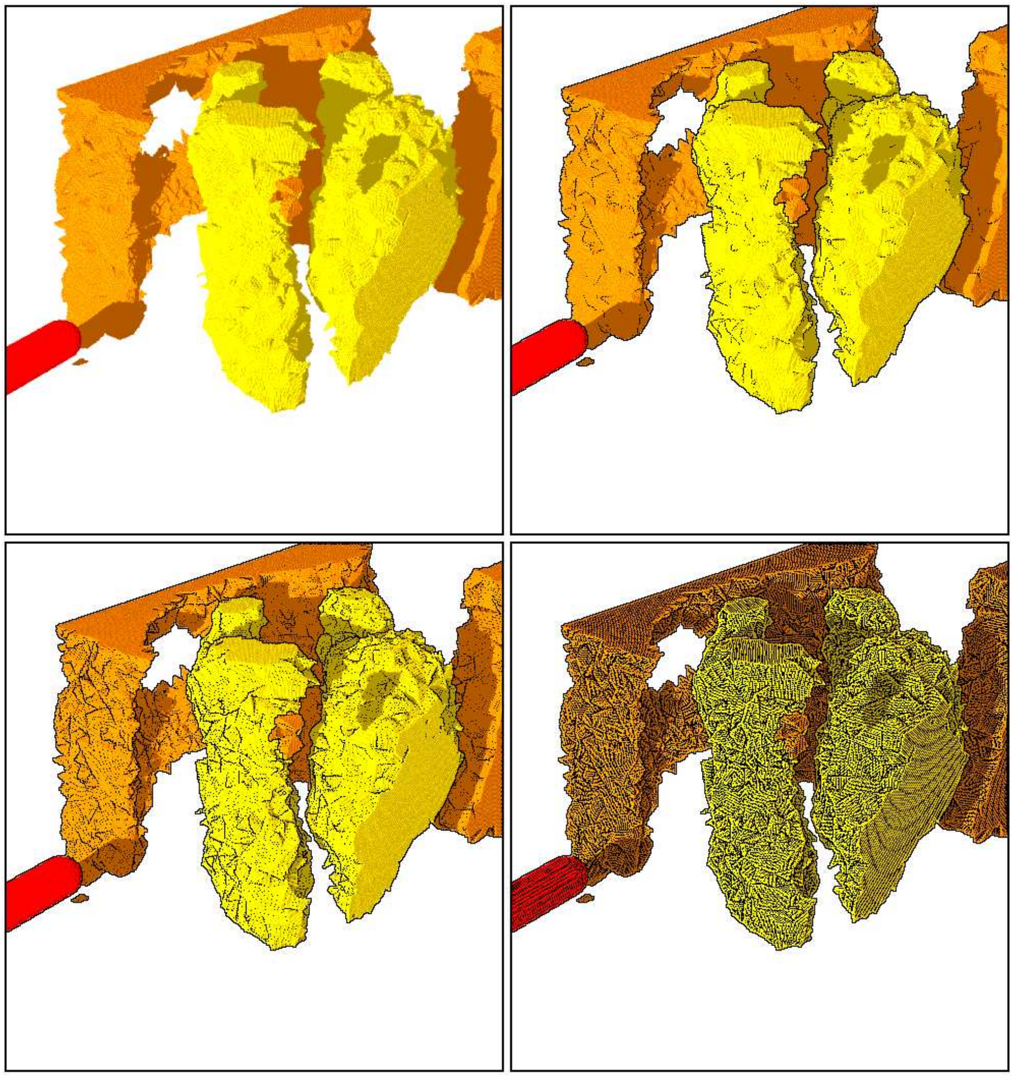

Figure 4.7. By varying the threshold of the Laplacian to be counted as an edge we can change how many edges are displayed. The upper left image shows no silhouettes. The upper right and lower left show progressively more and more edges. The lower right image, shows all the detectable edges.

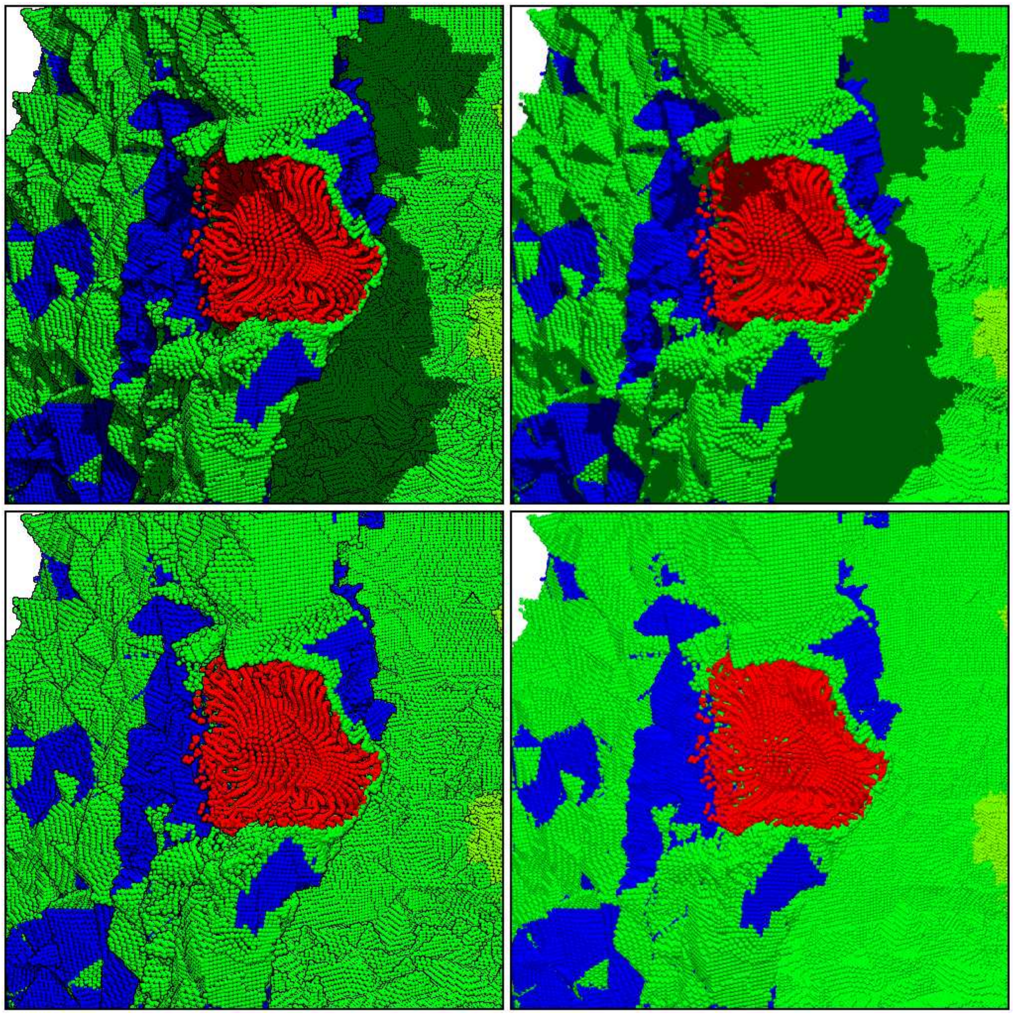

Figure 4.8. These images show the use of silhouette edges with and without shadows. The images are referred to as A (top left), B (top right), C (bottom left), and D (bottom right). Images on the top have shadows, while images on the left have silhouette edges.

CHAPTER 5

CONCLUSIONS AND FUTURE WORK

The goal of this thesis is to ascertain the appropriateness of the use of ambient occlusion and silhouette edges for particle visualization. Appropriateness can be defined as both the affected quality of the visualization and the feasibility of the use of these systems.

5.1 Ambient Occlusion

Ambient occlusion provides global lighting information that can be viewed interactively. This information is purely used for spatial perception. Domain scientists who were consulted stated the visualization with ambient occlusion value textures was of benefit to them, but only if they could be used for time varying data. Since RTRT provides these scientists the ability to view many time steps each containing hundred of thousands to millions of particles interactively, having textures for all the particles in memory would be necessary.

There are two major obstacles to achieving this. First, the amount of memory required to store the textures in memory is prohibitive. Since 256 bytes are required for each particle that itself only takes twelve or more bytes, only about a tenth the number of time steps can be loaded into memory. There may be, however, ways to overcome this that could be investigated for future work. The resolution of the texture could be reduced and the effects analyzed. This work could also be extended to investigate how useful a single ambient value per sphere could be.

Another way to relieve the memory burden is by reducing the number of textures in memory. This could be done several ways. First, the textures could be compressed in such a way as to take advantage of the redundancy in the texture sampling. One such form of redundancy is completely black textures corresponding to many particles that lie inside a group of particles completely obscuring the background. This and other redundancy could perhaps be exploited for significant gains. Second, memory for textures could be demand paged, so only the textures of the visible particles are loading into memory. Third, textures could have a variable sizes, based on the variation in the texture. Textures with lower variation could be represented with fewer texels.

The other factor limiting the usefulness of ambient occlusion textures is the precomputation time. Even utilizing multiple processors running in parallel, precomputation time could take anywhere from 30 minutes per time step to several hours. The foam data set, as seen in Figure 5.1 and consisting of 7.1 million particles, took 12 hours to compute the textures using 20 600 MHz R14K processors on an SGI Origin 3800. While the preprocessing time is acceptable for creating nice visualizations for a small number of time steps, it quickly becomes intractable for large time varying data sets.

The problem of precomputation time could be addressed in several ways. The first is to reduce the number of texels used in a texture, either uniformly or on a texture by texture basis as described above. A second way is to use a smaller radius for occlusion queries. Currently the ray must reach the extents of the geometry without registering a hit before it is deemed to have no occlusions. Perhaps, similar to Stewart's paper, a vicinity query could be computed[36]. This would reduce the radius for occlusion queries and result in earlier ray termination. The radius could be a user defined parameter. The performance benefit and impact on spatial perception would have to investigated as future work. Further work could be done to improve the efficiencies of the ray intersection routine. Since this is the most time consuming portion of the precomputation time, reducing the cost of performing ray intersections would decrease texture generation time. Another way to reduce the precomputation cost would be to only generate textures that are visible by performing view dependent queries. During an interactive phase, texels viewable from the rendering can be flagged for later precomputation. This will reduce the number of generated texels which would not be visible anyway.

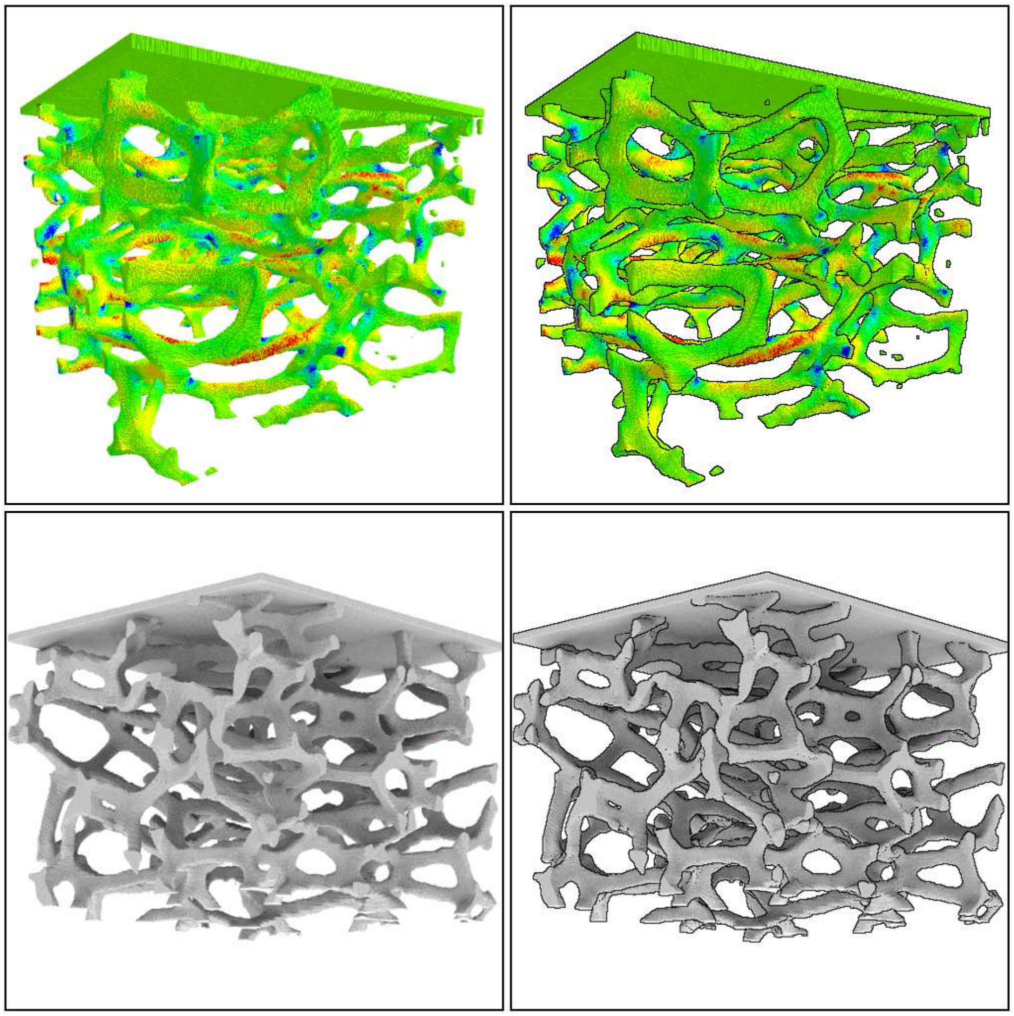

Figure 5.1. This data depicts a foam structure being crushed. The top images are color mapped with direct lighting only. The bottom images show the ambient occlusion values applied to the data set. Images on the right contain silhouette edges, the ones on the left do not.

The domain scientists need to be able color map their data. The use of ambient occlusion textures does not interfere with the ability to color map the particles as can be seen in Figures 3.13 and 3.15. Another feature users took advantage of is the ability to use shadows to accentuate various features. Since the ambient textures do not include direct lighting information, any component of the direct lighting such as the position of the light, its intensity, or color can be changed as need be for the purposes of the visualization.

Another feature domain scientists use when visualizing their particle data sets is the ability to crop out particles based on values associated with the particles. This feature allows users to view only the parts of the data that of are interest. When particles are cropped, the values in the ambient occlusion textures on longer correspond to the visualization being viewed. Users could recompute the texture values with the new cropping, but this can much longer then the investigative process of visualization reducing the ability to explore the data freely. A method to dynamically update the particles could be developed to overcome this issue. If a visibility query is made on the initial data and stored, this data could be used to recognize newly visible particles and dynamically update the shading on those particles and perhaps those in the neighborhood. This would reduce the cost of recomputing the entire thing. Adaptive methods such as these would still be costly and could lead to potential errors when textures do not correctly correspond the viewable geometry. For now, this method is limited to static and fly through visualizations, and not dynamic exploration.

In reference to the quality of the visualization, domain scientists indicated they liked the look of the visualization. However, the precomputation time and memory constraints were deterrents.

5.2 Silhouette edges

The other focus of this thesis was the appropriateness of silhouette edges. The use of silhouettes have several benefits, no precomputation time, little impact on frame rates, intuitive user control for the selection of how many silhouettes to view, and improved visualization of structure.

No precomputation is required to use silhouette edges during visualization. This ability to provide immediate response to the user makes the use of this technique attractive. It also provides users the ability to try out the visualization and determine if they like it. Since there is no cost to experiment with it, they are more likely to use it on a regular basis. There is also only a single control to adjust the density of silhouettes. Increasing the value will increase the number of silhouettes, decreasing the value produces the opposite effect. The results of changing the parameters are also visible while the user is changing them, making the experience more explorative.

When shown to the domain scientists, the utility of this method was readily apparent. One, in fact, began to make immediate use of it even when I was not around. In reference to the Foam data set (see Figure 5.1), he indicated how impressed he was as the structure became visible once silhouettes were used. This is a testimony of the attractiveness of this technique.

There is work to be done to improve this method. Currently the edges of the silhouettes can be jagged. A particle not represented by more than a handful of pixels can appear much larger and blocky. Silhouettes on these are useful, as particles that ordinarily would be missed now have some highlighting emphasis. A method to smooth out these edges, however, would better represent the macro shape of the object the silhouette is accentuating.

Another direction for future work would be to add gray levels to the silhouettes to indicate silhouettes of varying discontinuity. Thickness could also be used to accentuate greater differences in depth.

One final direction of future work would be to look at how to appropriately apply silhouette edges to multi-sampled renderings. Saito and Takahashi average the depth for a given pixel and use that value when computing edges[35]. This, however, will still produce jagged silhouettes. Perhaps the depth for each sample could be stored independently in a higher resolution buffer instead of averaged. Silhouettes would be computed with the depth buffer and applied with a width corresponding to a pixel size with gray levels darkest at the center and lightens at the edges.

5.3 Final comments

Silhouette edges are appropriate for particle visualization. They help aid in the perception of spatial relationships between particles, are easy to use, can be used without precomputation time, and have nominal impact on frame rates and memory consumption.

Ambient occlusion textures are also appropriate for particle visualization, however, there are several limitations that must be addressed as future work before they can more readily be used. These limitations include precomputation time, run time memory usage, and restriction to static data that will not be cropped during rendering. Despite these limitations, there are some positive aspects of using ambient occlusion textures. During run time, there is little impact on frame rates allowing the user the ability to inspect the data interactively. There is also no need to have a user make choices about lighting conditions that could spoil the rendering and require recomputation of the textures. Most importantly, the human perception system can more easily perceive the macro structure of the particles by utilizing an illumination technique that makes use of global position information.

REFERENCES

[1] J. Arvo and D. Kirk. A survey of ray tracing acceleration techniques. In A. S. Glassner, editor, *An Introduction to Ray Tracing*. Academic Press, San Diego, CA, 1989.

[2] S. G. Bardenhagen, J. U. Brackbill, and D. Sulsky. The material-point method for granular mechanics. *Comput. Methods Appl. Mech. Engrg.*, 187:529–541, 2000.

[3] R. Bruckschen, F. Kuester, B. Hamann, and K. I. Joy. Real-time out-of-core visualization of particle traces. In *Proceedings of the IEEE 2001 symposium on parallel and large-data visualization and graphics*, pages 45–50. IEEE Press, 2001.

[4] D. DeMarle, C. Gribble, and S. Parker. Memory-savvy distributed interactive ray tracing. In B. R. Dirk Bartz and H.-W. Shen, editors, *Eurographics Symposium on Parallel Graphics and Visualization*, 2004.

[5] D. DeMarle, S. Parker, M. Hartner, C. Gribble, and C. Hansen. Distributed interactive ray tracing for large volume visualization. In *IEEE Symposium on Parallel Visualization and Graphics*, pages 87–94, October 2003.

[6] A. Fujimoto, T. Tanaka, and K. Iwata. Artscelerated ray-tracing system. *IEEE Comput. Graph. Appl.*, 6(4):16–26, 1986.

[7] A. Glassner. *Principles of Digital Image Synthesis.* Morgan Kaufmann Publishers, San Francisco, California, 1995.

[8] A. S. Glassner. Space subdivision for fast ray tracing. *IEEE Computer Graphics and Applications*, 4(10):32–43, October 1984.

[9] A. Gooch, B. Gooch, P. Shirley, and E. Cohen. A non-photorealistic lighting model for automatic technical illustration. *Proceedings of SIGGRAPH 98*, pages 447–452, July 1998. ISBN 0-89791-999-8. Held in Orlando, Florida.

[10] A. A. Gooch. Interactive non-photorealistic technical illustration. Master's thesis, University of Utah, December 1998.

[11] B. Gooch, P.-P. J. Sloan, A. Gooch, P. Shirley, and R. Riesenfeld. Interactive techincal illustration. In *ACM Interactive 3D*, April 1999.

[12] G. Greger, P. Shirley, P. Hubbard, and D. Greenberg. The irradiance volume. *IEEE Computer Graphics and Applications*, 18(2):32–43, 1998.

[13] J. Guilkey and J. Weiss. An implicit time integration strategy for use with the material point method. In *Proceedings from the First MIT Conference on Computational Fluid and Solid Mechanics*, June 2001.

[14] S. Gumhold. Splatting illuminated ellipsoids with depth correction. In *Proceedings of 8th International Fall Workshop on Vision, Modelling and Visualization 2003*, pages 245–252, November 2003.

[15] S. Guthe, S. Gumhold, and W. Straer. Interactive visualization of volumetric vector fields using texture based particles. *Journal of WSCG*, 10(3), February 2002.

[16] A. Iones, A. Krupkin, M. Sbert, and S. Zhukov. Fast, realistic lighting for video games. *IEEE Computer Graphics and Applications*, 23(4):54–64, May 2003.

[17] H. W. Jensen. *Realistic image synthesis using photon mapping.* A. K. Peters, Ltd., 2001.

[18] J. T. Kajiya. The rendering equation. In *Computer Graphics (Proceedings of SIGGRAPH)*, volume 20(4), pages 143–150, August 1986.

[19] T. L. Kay and J. T. Kajiya. Ray tracing complex scenes. In *Proceedings of the 13th annual conference on Computer graphics and interactive techniques*, pages 269–278. ACM Press, 1986.

[20] G. Kindlmann, R. Whitaker, T. Tasdizen, and T. Möller. Curvature-based transfer functions for direct volume rendering: Methods and applications. In *IEEE Visualization*, October 2003.

[21] K. S. Klimaszewski and T. W. Sederberg. Faster ray tracing using adaptive grids. *IEEE Comput. Graph. Appl.*, 17(1):42–51, 1997.

[22] M. Krogh, J. Painter, and C. Hansen. Parallel sphere rendering. In *EUROGRAPHICS Workshop on Parallel Graphics and Visualization*, Gristol, England, September 1996.

[23] F. Kuester, R. Bruckschen, B. Hamann, and K. I. Joy. Visualization of particle traces in virtual environments. In *Proceedings of the ACM symposium on Virtual reality software and technology*, pages 151–157. ACM Press, 2001.

[24] H. Landis. Production-ready global illumination. *Course 16 notes, SIGGRAPH 2002*, 2002. Available at http://www.renderman.org/RMR/Books/sig02.course16.pdf.gz.

[25] M. Langer and S. W. Zucker. Shape from shading on a cloudy day. *Journal of Optical Society of America*, 11:467–478, 1994.

[26] M. Levoy. Display of surfaces from volume data. *IEEE Comput. Graph. Appl.*, 8(3):29–37, 1988.

[27] W. E. Lorensen and H. E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. In *Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, pages 163–169. ACM Press, 1987.

[28] T. J. V. Malley. A shading method fo rcomputer generated images. Master's thesis, Univeristy of Utah, June 1988.

[29] M. D. McCool. Shadow volume reconstruction from depth maps. In *ACM Transactions on Graphics*, volume 19(1), pages 1–26, January 2001.

[30] A. F. Méndez, M. Sbert, and L. Neumann. Obscurances for ray-tracing. In *EUROGRAPHICS 2003 Poster Presentation*, Granada, Spain, September 2003.

[31] D. P. Mitchell. Consequences of stratified sampling in graphics. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 277–280. ACM Press, 1996.

[32] S. E. Palmer. *Vision Science.* The MIT Press, Canbridge, Massachusetts, 1999.

[33] S. Parker, W. Martin, P.-P. J. Sloan, P. Shirley, B. Smits, and C. Hansen. Interactive ray tracing. In *Symposium on Interactive 3D Graphics*, pages 119–126, 1999.

[34] S. Parker, M. Parker, Y. Livnat, P. Sloan, C. Hansen, and P. Shirley. Interactive ray tracing for volume visualization. *IEEE Transactions on Visualization andComputerGraphics*, 5(3):238–250, July-September 1999.

[35] T. Saito and T. Takahashi. Comprehensible rendering of 3-d shapes. In *Computer Graphics (Proceedings of SIGGRAPH)*, volume 24(4), pages 197–206, August 1990.

[36] A. J. Stewart. Vicinity shading for enhanced perception of volumetric data. In *IEEE Visualization*, October 2003.

[37] D. Sulsky, S. Zhou, and H. L. Schreyer. A particle method for history dependent materials. *Comput. Methods Appl. Mech. Engrg.*, 118:179–196, 1994.

[38] D. Sulsky, S. Zhou, and H. L. Schreyer. Application of a particle-in-cell method to solid mechanics. *Computer Physics Communications*, 87:236–252, 1995.

[39] I. Wald, C. Benthin, A. Dietrich, and P. Slusallek. Interactive Distributed Ray Tracing on Commodity PC Clusters – State of the Art and Practical Applications. *Lecture Notes on Computer Science*, 2790:499–508, 2003. (Proceedings of EuroPar 2003).

[40] I. Wald, C. Benthin, and P. Slusallek. Distributed Interactive Ray Tracing of Dynamic Scenes. In *Proceedings of the IEEE Symposium on Parallel and Large-Data Visualization and Graphics (PVG)*, 2003.

[41] I. Wald, C. Benthin, M. Wagner, and P. Slusallek. Interactive rendering with coherent ray tracing. In A. Chalmers and T.-M. Rhyne, editors, *Computer Graphics Forum (Proceedings of EUROGRAPHICS 2001)*, volume 20, pages 153–164. Blackwell Publishers, Oxford, 2001.

[42] I. Wald, T. J. Purcell, J. Schmittler, C. Benthin, and P. Slusallek. Realtime Ray Tracing and its use for Interactive Global Illumination. In *Eurographics State of the Art Reports*, 2003.

[43] I. Wald and P. Slusallek. State of the art in interactive ray tracing. In *State of the Art Reports, EUROGRAPHICS 2001*, pages 21–42. EUROGRAPHICS, Manchester, United Kingdom, 2001.

[44] I. Wald, P. Slusallek, and C. Benthin. Interactive distributed ray tracing of highly complex models. In S.J.Gortler and K.Myszkowski, editors, *Rendering Techniques 2001 (Proceedings of the 12th EUROGRAPHICS Workshop on Rendering*, pages 277–288. Springer, 2001.

[45] L. C. Wanger, J. A. Ferwerda, and D. P. Greenberg. Perceiving spatial relationships in computer-generated images. *IEEE Comput. Graph. Appl.*, 12(3):44–51, 54–58, 1992.

[46] G. Ward, F. Rubinstein, and R. Clear. A ray tracing solution for diffuse interreflection. In *Proceedings of SIGGRAPH 1988*, Computer Graphics Proceedings, Annual Conference Series, pages 85–92. ACM, ACM Press / ACM SIGGRAPH, 1988.

[47] C. Wyman, S. Parker, P. Shirley, and C. Hansen. Interactive display of isosurfaces with global illumination. In *IEEE Transactions on Visualization and Computer Graphics*, Submitted for publication in.

[48] S. Zhukov, A. Iones, and G. Kronin. An ambient light illumination model. In *Rendering Techniques'98*, pages 45–55. Springer-Wien, NewYork, 1998.