MASTERS THESIS: USE OF SILHOUETTE EDGES AND VICINITY SHADING IN PARTICLE VISUALIZATION

by

James L. 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 L. 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 L. 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 like 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. I will investigate the use of image based silhouette edges and vicinity shading textures with respect to particle visualization determining if they are appropriate.

To my wife, Laura

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

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

- My family, especially Laura for all the love and support
- My advisor, for helping to provide the vision and direction that helped me get through this.
- Steve, for being such a great boss and mentor.
- Pete, he's the man!
- Christiaan Gribble, for a friendly ear and jointly writting much of the code used for the vicinity shading computation.
- C-SAFE and DOE, for the funds to do this research.
- SCI, for providing an atmosphere that makes collaboration happen.
- others

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)[4]. Objects and material 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 timestep 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.

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 and direct volume rendering. 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 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.

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. The use of a local lighting model such as Phong generated specular highlights can aide in seeing boundaries of particles and the three dimensional structure of individual particles, but can introduce other problems. One such problem is aliasing[1]. 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.

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 with a local model. Using a global operation, such as shadows, can provide information about proximity of structures and help disambiguate relative positions in the data[9]. However, shadows 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 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. Our perceptual system makes use of stereo, object, and color information to distinguish boundaries[9]. While stereo is not necessary to find most boundaries, it can reinforce edge perception found by changes in color 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

Global illumination is a lighting model which takes into consideration the placement of neighboring geometry[5]. 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 efforts to approximate the equation to speed up computation. Greg Ward et al. in their paper used a method of caching the indirect computation to speed up neighboring pixels[13]. 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 el al. used another method of caching illumination for later renderings[3]. They stored indirect illumination at discrete grid locations on a sphere. Object would then using the surface normal lookup and interpolate values of nearby nodes. This approximation provided for faster renderings and slight changes 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.

More recently global illumination has been looked at for visualization of dynamic isosurfaces[14]. 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[7] only computes the visibility of a point with a global light source (such as a sky light source). This happens as a preprocessing phase. The amount of light reaching these points are then stored with the geometry for later use in the rendering stage. Additionally the principle direction of the incoming light can be stored to index into an environment map during rendering.

Vicinity shading is a term coined by James Stewart in his paper dealing with precomputing the uniform diffuse illumination for isosurface volumes[12]. This is ambient occlusion for visualizing isosurfaces generated from volumes. Luminance is computed for a given voxel based on the visibility of the background giving equal weight for all directions. Surface points with more visibility had higher luminance than surfaces which were occluded by other geometry. The precomputed vicinity shading values are then stored in a separate texture volume and interpolated to the extracted isosurface during rendering. His paper showed the utility of including term based on structure of the geometry in the lighting.

Silhouette edges are another method for enhancing shape perception of the scene. These edges are usually rendered in black and follow the view-dependent hull of the object. This helps define macroscopic structure of the geometry. Several methods [11, 2] have been proposed to compute and render these edges for polygonal scenes. One such method uses the dot product of the normal and the view direction as a lookup table into a 1D transfer function. 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.[6] addressed

this issue by computing the curvature of the surface using the values in a volume to unify the edge width.

McCool used silhouette edges indirectly to compute shadows for interactive scenes [8]. 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 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[10]. The basis for the proposed research builds on this framework.

CHAPTER 3

VICINITY SHADING

In order to investigate vicinity shading 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 developed at the University of Utah. 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 vicinity shading information will pertain to spheres, since they will be used to represent particles during visualization.

3.1.1 Texture Mapping

Vicinity shading precomputes values on the surface of the geometry. In the ambient occluding papers discussed earlier, values were stored at the vertices. In Stewart's paper values were stored at voxel locations. Since spheres, however, have no natural vertices or voxels, two dimensional texture maps will be used to represent shading values on the surface. Vicinity shading 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 perhaps??). An advantage to using this mapping is the fast computation of texture coordinates used during rendering.

However, a disadvantage 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 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 be explored.

3.1.2 Ambient value computation

With the mappings established, vicinity values can now be filled in. 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 spheres and the ability to crop particles based on values associated with them. Precomputation time is based on the speed of ray intersections, so having a good acceleration structure is important to making this approach tractable.

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

The vicinity value texture 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. Texels are sampled a user defined number of times using a jittered sampling pattern (Reference or figure ??). 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. The hemisphere sampling takes two jittered values to provide a more uniform sampling pattern while reducing aliasing.

Once the ray direction and origin have been generated a ray is shot into the geometry. If a hit occurred, then the particle is obscured from the background and no luminance is added to the texel. If no intersection occurred 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 background could add perceptual cues. 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 are 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 3.1.

Since the texels which are causing problems are inside other particles they ordinarily do not contribute to the image. Their values can therefor 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 3.2.

The question becomes thus, how is a texel determined to be inside or outside? This determination is done 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. By examining the dot product of the ray's direction and the intersected object's

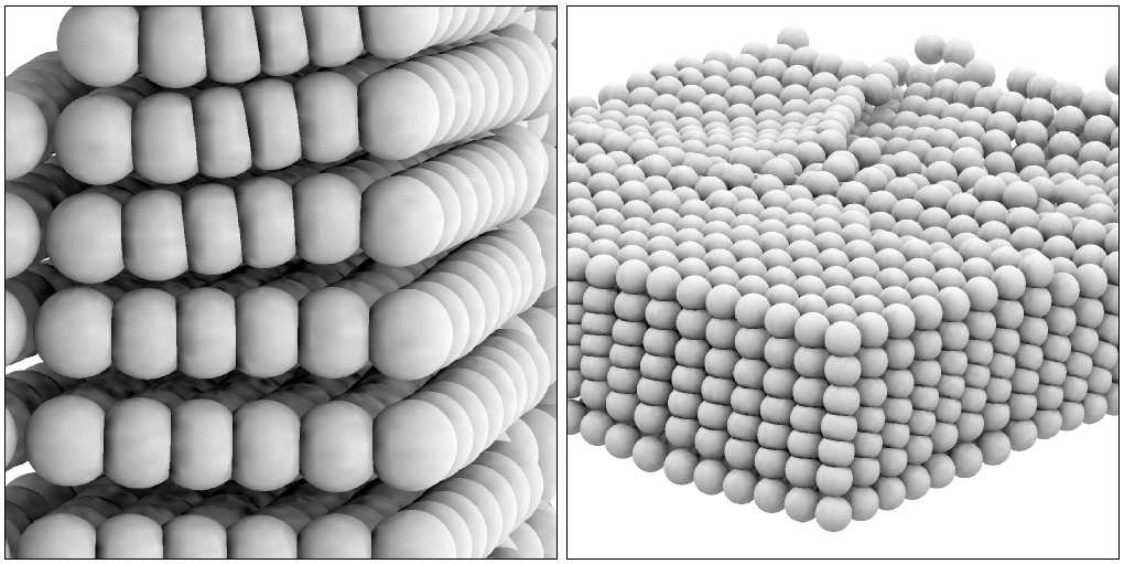

Figure 3.1. Bilinear interpolation can cause artifacts in regions where particles overlap

normal we can determine 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 3.3, 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 certain threshold are considered to be inside and are assigned an average of neighboring pixels who are considered outside.

3.1.4 Final post processing

After all the textures for the data set have been generated, a final post processing phase is done. This involves two steps, gamma correction and quantization to 8 bit integers. Gamma correction with a value of 2 is done to lighten up the darker regions for added perceptual acuity. Quantization is done using the maximum value in the texture set to save space in memory when the textures are needed for rendering (a

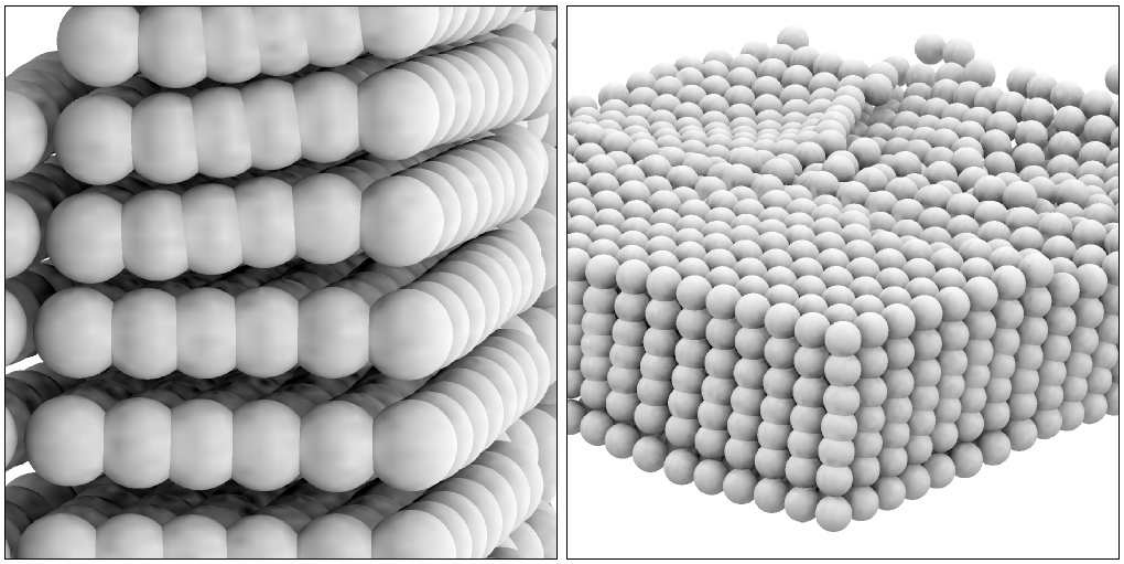

Figure 3.2. By dilating values that are outside to ones inside, interpolation errors can be reduced without introducing new errors.

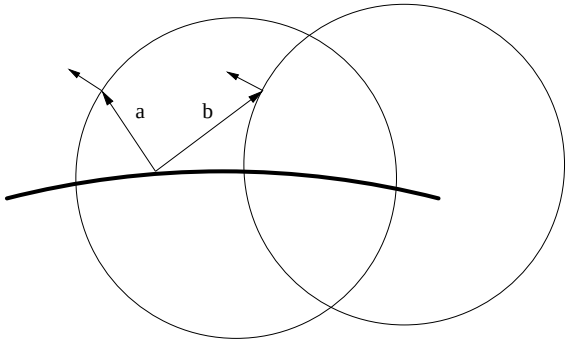

Figure 3.3. Determining if a ray is inside or outside by means of the dot product of the normal and ray direction works for ray a, but not for ray b.

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 this precision and 8 bits per texel is sufficient.

3.2 Render phase

The particle data and texture data is read in. The previously mentioned acceleration structure is built and used to perform ray object intersections. Once

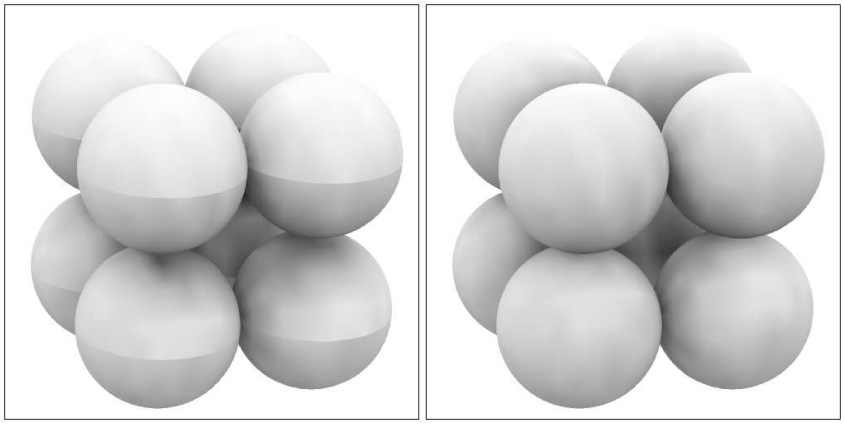

Figure 3.4. By wrapping the textures across the U parameter the seam can be eliminated.

a particle has been intersected its number 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 manor 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 more smooth appearance than nearest neighbors would provide. Texels are wrapped along the latitude direction of the texture, but not the longitudinal. This helps eliminate the seam running down the side as seen in Figure 3.4.

3.3 Analysis

3.3.1 Precomputation Time

While precomputation time is not the focus of this research, a couple notes are in order. Precomputation time is based on the number of total samples in the scene multiplied by the average intersection time. The number of total samples is the product of the number of samples per texel, texels per sphere, and spheres themselves. Table 3.1 lists some approximate times for texture generation.

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.

Name	Number of particles	Total number of samples	Time to generate (min)
Fireball 6	955,000	11,979,520,000	66 min.
Fireball 11	952,755	11,951,358,720	261 min.
Cylinder 6	214,036	2,684,867,584	13 min.
Cylinder 22	212,980	2,671,621,120	25 min.
Bullet 2	569,523	7,144,096,512	?? min.
Bullet 12	??	??	?? min.

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 time 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 Perceptual results

By computing ambient occlusion values, global 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 structure than without it. Figure 3.5 shows an MPM data set with and without the vicinity 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

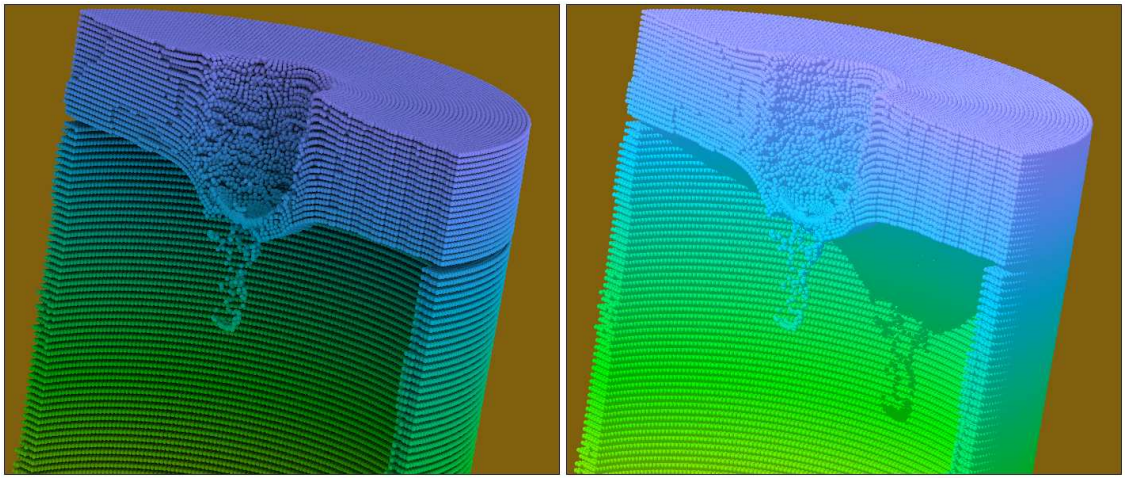

Figure 3.5. The image on the left is without ambient occlusion information. The one on the right includes it.

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 shot 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 scientist 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 vicinity shading is inexorably tied to the physical presence of all the particles, the shading no longer holds. Figure 3.6 illustrates this problem.

Figure 3.6. As the data is cropped, the vicinity shading values are no longer applicable.

However, once a suitable cropping is obtained the ambient occlusion information can be recalculated. Future work could include a method of updating the visible particles during rendering, so the user would be able to continue the inspection of the data.

The use of vicinity shading can greatly augment the perception of structure during visualization and is therefor appropriate. The ability of domain scientists to use it without having to tune rendering parameters is also beneficial. Reduction in precomputation time will be left for future work.

[Inclusion of other images, as I get them].

CHAPTER 4

SILHOUETTES

Create silhouettes using the depth buffer. This required two things storing the depth buffer and then using the depth buffer to create edges.

4.1 Depth buffer from ray tracing

Ray tracing determines the closest object by comparing intersection points along a ray $f(t) = direction * t + origin$. Comparisons are done using values of t, and the object corresponding to the smallest value is t is treated as the closest object. If the direction vector used is normalize, 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 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.

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. However, because the depth buffer is already relatively smooth, convolution of the Gaussian is unnecessary. Therefore, only the Laplacian is needed to look for zero crossings in the second derivative. In my implementation this is done using a 3x3 kernel of the shape found in figure 4.1.

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

Figure 4.1. Laplacian Kernel

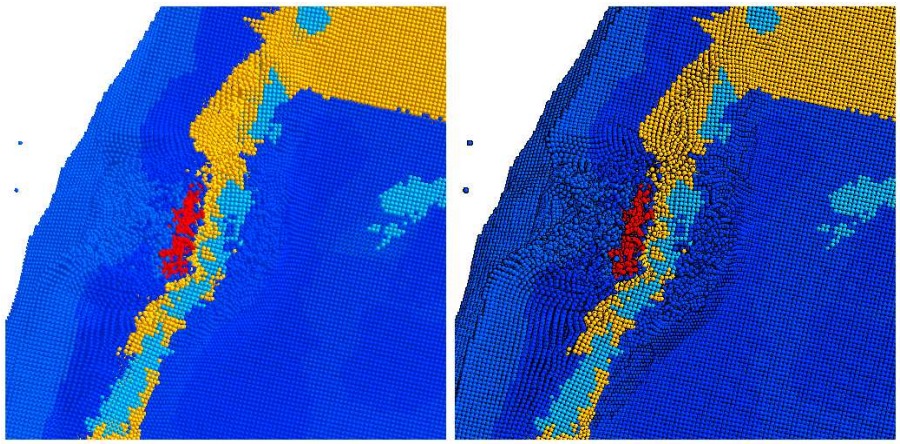

Figure 4.2. Silhouettes over each particle can be useful showing trends in particle placement, especially without shadows

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.

Marking all the silhouette edges in an image can be desirable in some cases as can be seen in Figures 4.2 and 4.3. Here the silhouettes provide a trend in particle placement where lighting or shadows fail.

Other cases this may result in too many silhouettes convoluting the image or generating aliasing. 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

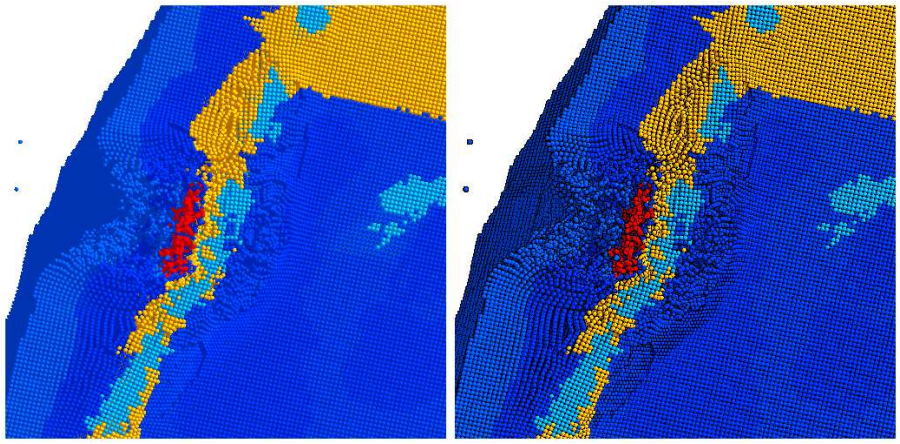

Figure 4.3. The same data and view as Figure 4.2, but with shadows

the magnitude of the Laplacian we can mark edges where the discontinuity is greater than a threshold. By varying this threshold we can selectively show edges corresponding to different degrees of discontinuity. Figure 4.4 shows a series of images where silhouettes are drawn for edges of increasing discontinuity.

The threshold can be used while rotation the object or changing the field of view without needing updating under certain circumstances. When the silhouettes are using the depth threshold, this threshold is only meaningful and useful while the image changes if the relative depth of the particles in the scene remain relatively constant. Conditions which provide this are common. Typical renderings of these particle datasets keep the look at and focal point at the center of the object. A user will usually only rotate the object or change the field of view, keeping the relative depth values meaningful to the depth threshold.

4.3 Analysis

One advantage of the silhouette edges is the precomputation time. Since this is entirely image based, there is no precomputation necessary. However it does require computation while rendering. For our particular implementation edge detection is done in a separate thread from the image rendering, and the computation is fast enough to not be out paced and become the bottle neck during interaction. Edge

detection must be done after the image is finished rendering, because we cannot guarantee neighbor information availability until after all rendering threads are finished computing their pixels. Only a single pass is necessary to do the edge computation.

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.

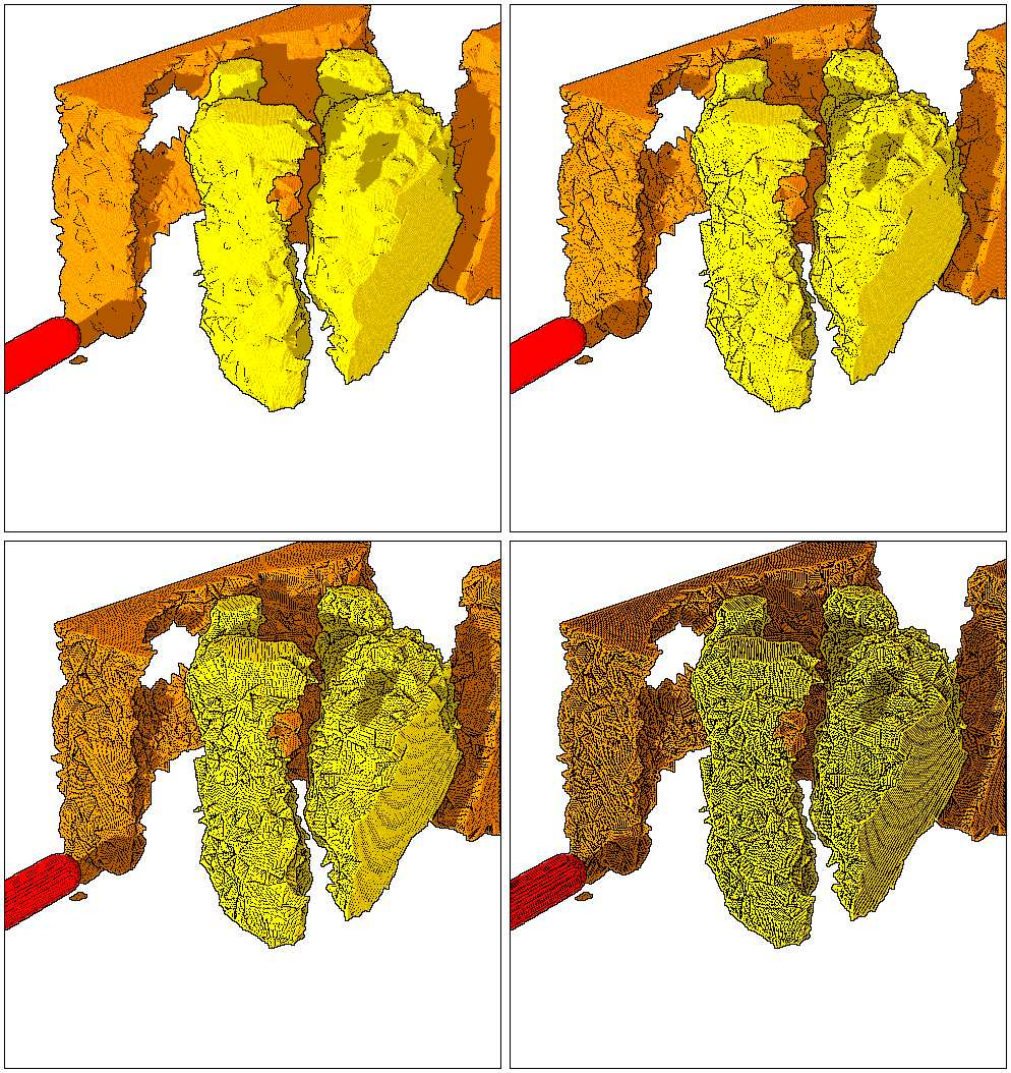

Figure 4.4. By varying the threshold of the Laplacian to be counted as an edge we can change how many edges are displayed.

CHAPTER 5

FUTURE WORK

1. Texture compression
2. How would using lower resolution textures look? What about only one value per sphere?
3. Silhouettes which can be more selective - larger support kernel
4. Different levels of silhouettes (intensity of silhouette is determined by the degree of discontinuity).
5. Ability to do silhouette edges in multi-sampled images.

REFERENCES

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

[2] Bruce Gooch, Peter-Pike J. Sloan, Amy Gooch, Peter Shirley, and Richard Riesenfeld. Interactive techincal illustration. In *ACM Interactive 3D*, April 1999.

[3] Gene Greger, Peter Shirley, Phillip Hubbard, and Donald Greenberg. The irradiance volume. *IEEE Computer Graphics and Applications*, 18(2):32–43, 1998.

[4] J.E. Guilkey and J.A. 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.

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

[6] Gordon Kindlmann, Ross Whitaker, Tolga Tasdizen, and Torsten Möller. Curvature-based transfer functions for direct volume rendering: Methods and applications. In *IEEE Visualization*, October 2003.

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

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

[9] Stephen E. Palmer. *Vision Science*. The MIT Press, Canbridge, Massachusetts, 1999.

[10] Steven Parker, William Martin, Peter-Pike J. Sloan, Peter Shirley, Brian Smits, and Charles Hansen. Interactive ray tracing. In *Symposium on Interactive 3D Graphics*, pages 119–126, 1999.

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

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

[13] Gregory Ward, Francis Rubinstein, and Robert 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.

[14] Chris Wyman, Steven Parker, Peter Shirley, and Charles Hansen. Interactive display of isosurfaces with global illumination. In *Proceedings of IEEE Visualization 2004*, October 2004. Submitted for publication.