

REAL-TIME RENDERING OF PHYSICALLY-BASED CLOUD SIMULATIONS  
FOR UNIVERSITY UNDERGRADUATE  
RESEARCH FELLOWS

A Senior Honors Thesis

by

KEVIN MICHAEL WALKINGTON

Submitted to the Office of Honors Programs  
& Academic Scholarships  
Texas A&M University  
in partial fulfillment of the requirements of the

UNIVERSITY UNDERGRADUATE  
RESEARCH FELLOWS

April 2004

Major: Computer Science

REAL-TIME RENDERING OF PHYSICALLY-BASED CLOUD SIMULATIONS  
FOR UNIVERSITY UNDERGRADUATE  
RESEARCH FELLOWS

A Senior Honors Thesis


by

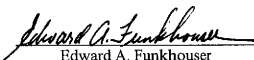
KEVIN MICHAEL WALKINGTON

Submitted to the Office of Honors Programs  
& Academic Scholarships  
Texas A&M University  
in partial fulfillment of the requirements of the

UNIVERSITY UNDERGRADUATE  
RESEARCH FELLOWS

Approved as to style and content by:

  
John Keyser  
(Fellows Advisor)

  
Edward A. Funkhouser  
(Executive Director)

April 2004

Major: Computer Science

**ABSTRACT**

Real-Time Rendering of Physically-Based Cloud Simulations  
for University Undergraduate Research Fellows. (April 2004)

Kevin Michael Walkington  
Department of Computer Science  
Texas A&M University

Fellows Advisor: Dr. John Keyser  
Department of Computer Science

Computers today employ simulations of physical phenomena such as wind and fire and other physical properties in many common applications, including programs meant for training and entertainment. We focus particularly on the realistic simulation of cloud formation and existence on current commercially-available computers. One of the challenges associated with this simulation is its display onto a computer screen, often referred to as rendering. We will present a brief overview of existing cloud rendering techniques and compare their effectiveness to rendering a simulation as it occurs. We will then suggest our rendering method which relies upon the use of three-dimensional textures and modified Gaussian transfer functions for the self-shadowing properties associated with clouds. We will analyze these results, focusing on frame rates and visual appearance, and then conclude by suggesting further work on this topic.

## ACKNOWLEDGMENTS

I would like to extend my greatest thanks to my advisor, John Keyser, for his support and guidance on this project. I would also like to thank Derek Overby and everyone in the Computer Graphics Research Group for their ideas and suggestions. I must also thank my peers, and especially my fellow Undergraduate Research Fellows, who have been working on similar projects and theses for their support. I give thanks to my friends who have been there in my time of need. And finally, I thank my parents, my sister, and the rest of my family for their continuing beliefs in my abilities, their affection, and everything they have done and continue to do to encourage my progress through life.

## TABLE OF CONTENTS

|  | Page |
|--|------|
| ABSTRACT .....   | iii  |
| ACKNOWLEDGMENTS .....                                    | iv   |
| TABLE OF CONTENTS .....                                  | v    |
| LIST OF FIGURES .....                                    | vii  |
| LIST OF TABLES .....                                     | viii |
| SECTION  |      |
| 1 INTRODUCTION .....                                     | 1    |
| 2 PREVIOUS WORK .....                                    | 4    |
| 2.1 Gardner’s Texture-Mapped Ellipsoids .....            | 4    |
| 2.2 Shells and Metaballs .....                           | 5    |
| 2.3 SkyWorks .....                                       | 7    |
| 2.4 A Noise Method for User-Defined Clouds .....         | 8    |
| 3 REAL-WORLD CLOUDS .....                                | 10   |
| 3.1 Non-Solid Formations .....                           | 10   |
| 3.2 Shading .....  | 12   |
| 3.3 Lighting .....                                       | 12   |
| 4 RESOURCE BASIS .....                                   | 14   |
| 4.1 Data Storage .....                                   | 14   |
| 4.2 Gaussian Transfer Functions and Self-Shadowing ..... | 16   |
| 4.3 Levels-of-Detail .....                               | 20   |
| 5 IMPLEMENTATION .....                                   | 23   |
| 5.1 A Cloud Rendering Program .....                      | 23   |
| 5.2 Implementation Issues .....                          | 25   |

|  | Page |
|--|------|
| 6 RESULTS.....                         | 27   |
| 7 FUTURE WORK .....                    | 31   |
| 7.1 Voxel Processing.....              | 31   |
| 7.2 Alternative LOD Schemas .....      | 33   |
| 7.3 Shading Languages .....            | 34   |
| 7.4 Extensions to Current System ..... | 35   |
| 8 SUMMARY AND CONCLUSIONS.....         | 37   |
| REFERENCES.....                        | 38   |
| APPENDIX A .....                       | 40   |
| APPENDIX B .....                       | 41   |
| VITA .....                             | 43   |

**LIST OF FIGURES**

| <b>FIGURE</b> |  | <b>Page</b> |
|---------------|--|-------------|
| 1             | Cloud constructed by ellipsoids .....                          | 4           |
| 2             | Description of Dobashi sphere method .....                     | 5           |
| 3             | Final rendered clouds using Dobashi sphere method .....        | 6           |
| 4             | SkyWorks scene .....   | 7           |
| 5             | Noise-based cloud .....  | 8           |
| 6             | Cumulous clouds .....  | 11          |
| 7             | Light scattering through cloud .....                           | 13          |
| 8             | Shading model applied at varying light distances .....         | 20          |
| 9             | Screenshot of cloud rendering framework .....                  | 27          |
| 10            | Initial cloud rendering .....                                  | 28          |
| 11            | Final rendered image with LOD model applied .....              | 30          |
| 12            | Billboard impostors .....                                      | 34          |
| 13            | Shading language used for shadow-mapping .....                 | 35          |
| 14            | Self-shadowing with light positioned along positive axes ..... | 41          |
| 15            | LOD model applied to rendering system .....                    | 41          |
| 16            | Underside of cloud with respect to light source .....          | 42          |

**LIST OF TABLES**

| <b>TABLE</b> |   | <b>Page</b> |
|--------------|---|-------------|
| 1            | Memory growth per cloud volume texture.....               | 15          |
| 2            | Mathematical symbols and variables used in equations..... | 17          |
| 3            | Frame rates at varying slices through volume texture..... | 29          |



## 1 INTRODUCTION

A major area of on-going research within the field of computer graphics is the simulation<sup>1</sup> of real-life phenomena and the rendering involved. Examples of these phenomena include water, plant life, fire, etc. There are typically two distinct forms of rendering these simulations. One method is to produce the most realistic and best fitting model to the data provided by the simulation. This type forgoes speed to produce highly accurate and detailed renderings of the phenomena being simulated. The other type is just the opposite, trying to produce a visually acceptable simulation in real-time. Visually acceptable means to that with a certain degree of error in the picture produced, the image appears to represent the data accordingly with minimal artifacts, particularly when close to the viewer if using an LOD, or level-of-detail, approach. Real-time implies that the images created are being displayed at a rate that is consistent with the rate at which the human eye can accept rapid projection of images such that no jagged motion can be seen – typically this is at least 24 to 32 fps (frames, or images, per second).

Cloud rendering, in particular, has seen little advancement in the simulation area. For the most part, cloud rendering is thought of as a process that can be added in by hand after the majority of the rendering of a computer-generated scene has been done. This commonly occurs though either the use of skyboxes, polygonal forms that surround the

---

This thesis follows the style and format of *IEEE Transactions on Visualization and Computer Graphics*

<sup>1</sup> Here we refer to simulation as a model existing in a computer-generated world that behaves similarly to a real-life counterpart based upon some true-life model or collection of data

upper part of a computer-generated world that simply have textures applied to the faces to give the impression of a sky, or other post-rendering techniques -- in the worst case merely "painting" in the clouds. Other techniques, such as the SkyWorks project [9,10], involve actual rendering of spheres with various light models in conjunction with projection of textures at the appropriate time and positioning to fool the viewer that clouds as actual objects in three dimensional space are fully being rendered. The downfall of both of these procedures is that rendering follows no physically-based model so while the images may appear accurate and acceptable to the eye, they are merely forced representations lacking a real-world structure that simulations supply.

Recently here at Texas A&M University, Derek Overby researched for his Master's Thesis a physically-based simulation of cloud formation [13]. This simulation reacts to temperature, volume, pressure differences and water vapor parameters to generate clouds in a simulated environment. The simulation, however, is lacking in key areas. In particular, the clouds rendered do not appear as clouds but conglomerations of gray and white. The simulation also can only generate these images at the rate of roughly 1 frame per second on voxelized grids of size 25x50x25. Generation of real-time images that appear acceptable as clouds to the viewer will require converting the data given by either this simulation or another existing one to another format that is easier and quicker to render.

Having a real-time, realistic looking, physically-based simulation of clouds could prove to be an essential aid to various applications of graphics. Weather forecasting and simulation programs are obvious beneficiaries to this type of research. Another common

application is flight simulation. Realistic rendering of clouds in real-time that are not programmed can bring a whole new level of detail to these types of simulation. Furthermore, advancement of rendering simulations of this type could bring advancements to other areas of graphical representation, bringing computer-generated images that much closer to actually looking realistic.

In this document we will first examine in Section 2 some of the previous work conducted on the topic of rendering clouds in general and also simulations of clouds. In Section 3 we will take a closer look at clouds in the real world and the complexities associated with rendering certain elements visible in the natural world on the computer. For Section 4 we discuss Gaussian transfer functions, volumetric textures, and other resources we plan to use in our model. We will discuss our implementation of a rendering program for clouds and implementation issues in Section 5. We will display our results and discuss them in Section 6. We will explain fully several options for future work in Section 7. And in Section 8 we will finally present a summary and our conclusions.

## 2 PREVIOUS WORK

In this section we look at previous work that has been performed in the field of computer graphics on rendering clouds. In Section 2.1 we focus on the work of Elinas [6] with his using a modified form of Gardner's texture-mapped ellipsoids [8]. We look at the work of Dobashi et al. and their shell method in Section 2.2 [3]. Following this in Section 2.3 look at the contributions of the SkyWorks project [9,10]. Finally, we look at a recent study of real-time animation of clouds in Section 2.4.



Figure 1: Cloud constructed by ellipsoids. Image taken from [6].

### 2.1 Gardner's Texture-Mapped Ellipsoids

Looking at clouds, one can quickly notice that there is a tendency for the clouds to appear as a collection of curved surfaces. Moreover, commonly the densest part of the cloud will appear in the center. It is for this reason that Gardner in his 1985 study [8] developed a method for rendering series of ellipsoids texture-mapped in such a way that the very center is the most opaque by appearance and that noise added to the outer edges

causes the details on the fine edges of clouds. Elinas [6] in his paper made use of this method along with a few slight modifications and came up with clouds which appear in Figure 1.

While the output does appear like clouds, they lack the shading to properly define them as such. Moreover, the cloud represented is supposed to be of the stratus form but the ellipsoid approach prevents the correct appearance from showing up.

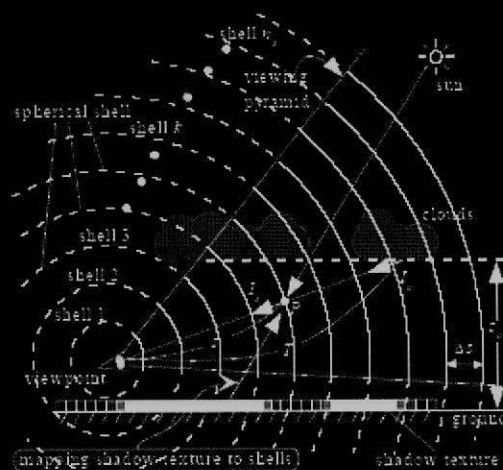


Figure 2: Description of Dobashi sphere method. Image taken from [4].

## 2.2 Shells and Metaballs

Dobashi et al. in a fairly recent paper developed a method for rendering clouds and several effects that are generated by clouds. The method they proposed uses a series of concentric spheres positioned around the viewpoint of a scene to texture-map a portion of the clouds onto as seen in Figure 2. This allows effects like shafts of light which occur because of scattering and absorption by particles in the atmosphere to be

displayed fairly easily [4]. Some final rendered scenes depicting some of the effects and the clouds using this method can be seen in Figure 3.



Figure 3: Final rendered clouds using Dobashi sphere method. Images taken from [4].

Despite the nice appearance, this method used does not provide real-time results. If examined closely, the method requires a conversion to a data structure known as metaballs. Metaballs are essentially ball-like data structures specified at a particular point that encompass a region of uniformity of a larger data structure, in this case a cloud formation.

While metaballs can be used effectively to render clouds, they do not serve as effective and useful data structures for storage of simulation data. The reasoning behind this results from how the dynamics of cloud formation change so quickly and do not always fit into the model of elliptical formations. As clouds form, condensation may rise in various parts of the cloud at different times causing those sections to appear more opaque. Over time, the area between these original parts may come together forming more uniform sections. The result is a constantly and drastically changing distribution

of condensation densities which cannot be easily calculated and transformed into a distribution amongst multiple metaballs, especially if the data containing condensation levels is in a volumetric grid like our simulation.

### 2.3 SkyWorks

SkyWorks is an ever-changing real-time cloud rendering environment developed by Mark Harris and others at the University of North Carolina at Chapel Hill [9,10]. The environment specializes in pre-computed virtual worlds and makes good use of impostors, images which are displayed in place of the real objects being rendered to increase frame rate. An image of SkyWorks can be seen in Figure 4.

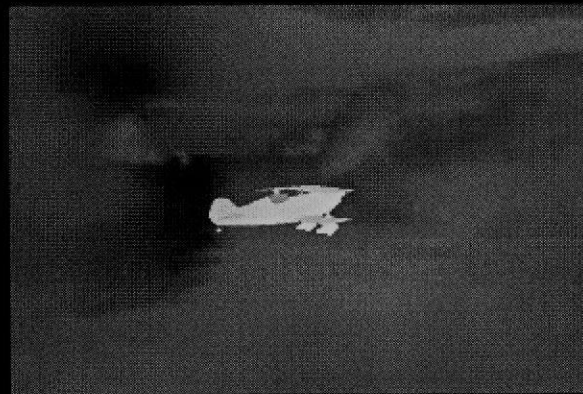


Figure 4: SkyWorks scene. Image taken from [9].

At first glance, the model used in SkyWorks would seem very similar to the model we propose here, although there are a few key differences. First and foremost, in SkyWorks, the scene receives a full computation of the lighting and shading only once per runtime while receiving updates during the runtime process. While it is not stated

the format of “particles” in Harris’ paper, the format of our clouds is geared specifically towards simulation. Lastly, there are minute differences in how we calculate the shading giving our implementation the ability to calculate the shading without relying on the position of the viewer or having to perform multiple render passes on each cloud.

#### 2.4 A Noise Method for User-Defined Clouds

We discuss the effects of Schpok et al. [14] last as the rendering methods do provide some background insight, but we will not dwell on it for long as the method is applied to a model of rendering clouds which is not based upon physical data, rather upon noise data. Noise data refers to that the data which makes up and determines a cloud is not real or based in the physical world. Instead the data used here is made up of Perlin noise and various different types of clouds are defined by operations on this noise data.



Figure 5: Noise-based cloud. Image taken from [14].



By using the graphics hardware, the user can manually defined particular constants to be passed in to the rendering system, and the output is a nice approximation to a cloud of the type defined by the constants. An example of the output of this system can be seen in Figure 5.

This work just goes to show that methods exists to render clouds in real-time which are not based upon physical data and do simplify the rendering problem for people who wish to construct a cloud from scratch. For additional information regarding these methods should reference work by David S. Ebert who specializes in volumetric visualization and procedural techniques [5].

### 3 REAL-WORLD CLOUDS

To more accurately model clouds, we must base a system upon the real world. Hence, we look at the complexities associated with the appearance of clouds in the real world in this section. We focus on the nature of clouds as non-solid formations in Section 3.1. After that, we discuss the shading of clouds in Section 3.2 – in particular the self-shadowing nature of cloud formations. And in Section 3.3, we examine light scattering

#### 3.1 Non-Solid Formations

Clouds by their very nature are not solid formations. They are vaporous objects which gradually change form over time. The forms can be anything from long thin clouds, stratus formations, to large puffy forms, cumulous formations. And on the boundaries, minute fine detail blurs the edges of what appears to be a visually solid formation. As seen in Figure 6, the cumulous clouds shown have very unique shapes and seem to blur together to form larger less-continuous clouds with the fine detail on the edges.

The lack of rigid structure making up the clouds causes much concern for someone attempting to reconstruct them in a computer system. Take a cube for example. In order to reconstruct it, all that is required is a central position, the width of one side, and a normal of one of the faces. From there, the location of the 8 vertices and, in fact, every point on the surface can be determined. Even for a curved structure such

as an ellipse there exist various simple methods to calculate and determine structure. Merely suggest a central point, radii in each direction, and an orientation of the figure and an ellipse can be rendered. Clouds can vary in structure along any given particle within the surrounding volume ruling out a true model using solid sides.



Figure 6: Cumulous clouds.

As a result, the volume in which a cloud exists often is either approximated by solid forms, as described in Section 2.1, or is expressed in terms of other forms. Such forms include particles, textures, and direct buffer manipulation. As described earlier, the solid form approximations fail when they try to fully and accurately represent the data, while the other forms, for the most part, overcome data representation problems sometimes at the cost of complexity to render. More will be discussed on some of these other forms later.

### 3.2 Shading

As can be seen in the clouds in Figure 6, clouds are not always of one color. This may be an obvious notion; however, when trying to render these formations on the computer, every minute detail needs to be addressed to maintain accuracy. In fact, a large majority of the appearance of clouds comes as a result of their self-shadowing property. Self-shadowing refers to how, despite clouds being thought of as continuous forms, they are composed of individual vapor particles which block some light from penetrating the entire form. The high number of vapor particles and low density causes significant shading to affect the appearance.

To accurately model this self-shadowing effect, a method for determining the degree of light penetration at particular points within the cloud and the percentage of light let through the cloud needs to exist. We will discuss a method we have used for this later.

### 3.3 Lighting

In respect to lighting of clouds, the shading is a result of light diffusion through the cloud as discussed above. What we have not discussed with respect to light diffusion is the scattering effect that occurs which illuminates a greater portion of the cloud.

Light scattering refers to simply the reflection of a ray of light from a source. Since clouds are formations resulting from condensation of water vapors in the

atmosphere [4], it makes sense that these vapors will cause some disruption of a ray of light entering into the formation. See Figure 7 for an example of scattering.

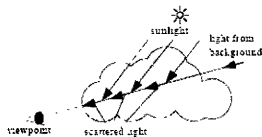


Figure 7: Light scattering through cloud. Image taken from [4].

When determining scattering in a cloud on a computer, we must consider the degree of complexity of the scattering. Typically, there are two forms: single and multiple scattering. Single scattering refers to the effect of dealing with only a single scatter of a light ray from a source to a viewer, while multiple scattering includes multiple scattering from a single source [13]. In the real world, there exist an infinite number of reflections which occur, so for a computer, modeling this exactly would not be practical. Moreover, a single scattering would handle only partially the effect, so we need to choose a medium in between. For this, we will employ an approximation to be discussed in Section 5.

## 4 RESOURCE BASIS

We discuss in this section the mathematical and computational basis for our implementation. This refers to data storage for the clouds in Section 4.1 and the functions and equations used in lighting and shading in Section 4.2. We finish this section with a brief discussion on levels-of-detail and one particular method we have implemented in Section 4.3. In each subsection we discuss our reasons for the choices of whichever bases we choose.

### 4.1 Data Storage

In Section 2 we mentioned briefly the choice of data storage for particular implementations and began highlighting their advantages or disadvantages. We will expand upon that discussion here.

Many of the previous methods made use of noise data to construct clouds while others relied on particles. In our implementation, we assume that the data will be coming from a grid-like structure in space, often referred to as a volumetric grid. This assumption is based upon the fact that the simulation we have to work with is in this format. For shading and other parts of the algorithm we develop a class which defines a particular point within the grid, referred to as a voxel or volume element [7].

Our assumption of the arrangement of the data requires that we design our rendering algorithm around it to remove the need for conversion which would add an

extra costly step. For this, a simple and effective tool exists to aid us: volumetric textures.

Volumetric textures have been around since the mid-eighties and have seen wide-spread use in the medical imaging fields. They are similar to 2D textures but have extra information specifying the depth field, or the extra third-dimension. Essentially, volumetric textures allow us to specify the color and opacities of individual points within a volume rather than trying to specify the details of the volume or part of the volume as a whole. It now becomes self-evident that volumetric textures can make rendering volumes that lack continuity or strongly defined boundaries easy, such as clouds.

| Size of Cloud Volume | Storage Memory Required |
|----------------------|-------------------------|
| $32^3$               | 128 KB                  |
| $64^3$               | 1024 KB = 1 MB          |
| $128^3$              | 8192 KB = 8 MB          |
| $256^3$              | 64 MB                   |

Table 1: Memory growth per cloud volume texture.

The question now is why have volumetric textures not been used extensively in rendering clouds or in any other real-time rendering context? The answer is simple: often times, specifying colors and opacities at individual points within a large grid become costly in terms of memory. If one assumes that a single byte is used for each color and one byte for the opacity component (red, blue, green, and alpha respectively or 4 bytes), the memory growth in Table 1 appears. Years ago this would not have been

acceptable; however, with recent hardware improvements and the reduced cost of memory, it is not uncommon to see computers with over a gigabyte of main memory and 128 MB or more of memory of the video card, the location where a volumetric texture would reside.

Typically, volumetric textures have also been considered to be quite slow for their use. Recent improvements in the technology on video cards and the greater use of procedural noise textures have improved the speed of using volumetric textures. As can be seen, the use of volumetric textures is not without its costs but we believe that now they can be thought of as a viable use for consumer applications and hardware.

We will make use of the volumetric textures to define the cloud volume as it will be displayed. One might be concerned of updating the volume to reflect changes to the clouds or their appearance. We have found that with direct manipulation of the cloud texture, we can actually perform quick updates. It should be noted that we are not concerned with updating the appearance of the cloud on every step but rather whenever the simulation updates which may be every 4 or 5 seconds or more greatly reducing the need for expediency.

#### **4.2 Gaussian Transfer Functions and Self-Shadowing**

In order for our rendering process to display clouds that appear to be real, a test condition to be discussed later, they must be shaded as if they are being viewed in the real world. As we have decided upon volumetric textures for the volume, we examine



lighting and shading work by Joe Kniss [10,11], which can be directly applied toward our rendering method with some slight modification.

To add self-shadowing to our basic rendering method, which consists only of rendering a single volumetric texture at this moment, we will use the Gaussian transfer functions in [11] used on semi-transparent volumes modified ever so slightly to reduce overall number of computations.

Transfer functions are simply mathematical functions that apply some color or intensity values to particular ranges of numerical data. Gaussian transfer functions vary slightly in that there is an assumption on the initial range of the data and that there is a preferred range that the function will map data to. In particular we are concerned with the one-dimensional transfer function associated with intensity of light.

|                  |   |
|------------------|---|
| $I_K$            | Light intensity at volume element K                           |
| $B_K$            | Blur at volume element K                                      |
| $\rho'_K$        | Density line integral at volume element K                     |
| $\alpha_K$       | Opacity at volume element K                                   |
| $\theta$         | Angle between light vector and difference vector              |
| $\mathbf{K}$     | Real-world volume element coordinates expressed as vector     |
| $\mathbf{d}$     | Difference vector, originating at K going toward light source |
| $\ell$           | Light vector, real-world coordinates as vector form           |
| $\alpha_{avg}$   | Average opacities of nearby volume element opacities          |
| $\alpha_{accum}$ | Accumulated opacities   |

Table 2: Mathematical symbols and variables used in equations.

In Table 2, we define the necessary variables to make sense of the Gaussian intensity transfer functions. The intensity is a measurement of how much a portion of the color of the light source (red, green, or blue) is applied to the color of the current volume element. This measurement is split up into two parts as seen in equation 1. The direction portion refers to the amount of light that is directly applied based upon the incident light vector. The indirect portion refers to the light applied from the scattering nature of clouds.

$$I_K = \underbrace{\left(1 - e^{-\alpha_{avg} \rho'_K}\right)}_{direct} + \underbrace{\left(1 - e^{-\alpha_{avg} \rho'_K}\right)}_{indirect} \cdot B_K \quad (1)$$

The K subscript of many of the equations refers to implies that a particular equation must be applied for each voxel at the K'th step along the light ray through the volume. We use this notation rather than integrals to directly refer to when discussing our implementation in the next section.

$$B_K = \|\mathbf{d}\| \cdot \tan\left(\frac{3}{2}\theta\right) \quad (2)$$

The blur shown in equation 2 is a variance applied almost directly from [11] to offset the amount of indirect light such that there is only a slight alteration to the

appearance of the cloud. It is calculated from the angle between the light source vector, or its location in vector form, and the vector from the current voxel to the light source.

$$\alpha_{accum} = \left( 1 - \frac{3}{8} \left( 1 - \frac{\sqrt{\pi}}{2} \right) \cdot \alpha_K \right) \cdot \alpha_{avg} \quad (3)$$

For each volume elements, we keep an accumulation of alpha terms, or rather a weighted accumulation of the amount of light. This accumulation can be seen in equation 3, which determines the amount of light that has passed through the volume element. The numbers in front of the current alpha value are again offsets to lower the amount of the current alpha term contribution to the complete alpha accumulation. We have found the given numbers to produce adequate shading which can be seen in our final results.

In calculating the accumulation and the intensities we also use averages as it considers that each contributing nearby volume element will scatter some light to the current voxel, as represented by  $\alpha_{avg}$ . And since we can assume no decay in the ray from the light source itself, we can apply a portion of the fragments of light to the total light the current voxel receives.

If one refers to [11] there are noticeably slight differences in the variable used. Rather than using the view vector to perform many of the lighting calculations, we

instead base the vectors off of the voxel location in space and how it relates to an adjusted light source vector.

In our system, we perform all shading operations based upon the concept of the volume being centered on the origin. In a system where the cloud is not based at the origin, we can simply apply the appropriate translation to convert it to be so. We have also found that our equations are useful under the assumptions that the light source is a distance anywhere from 25 to 75 units away (based relatively to a cloud consisting of  $32 \times 32 \times 32$  voxels spanning a space of 2 units per side). As can be seen in Figure 8, the light source moves further and further away beginning at the left image until artifacts start to appear in the shading in the final image on the right.

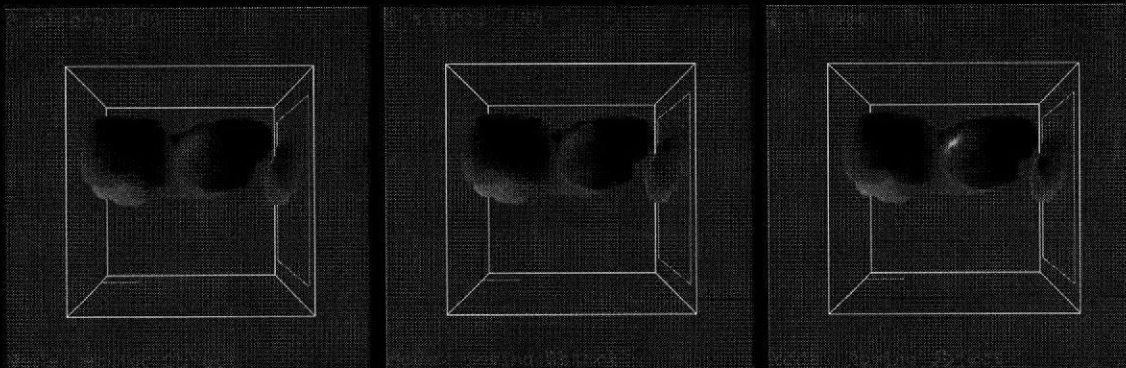


Figure 8: Shading model applied at varying light distances. Left: Distance of 25 units. Middle: Distance of 75 units. Right: Distance of 100 units.

### 4.3 Levels-of-Detail

Recently, computers have become capable of displaying enormous scenes which appear very complex and up close are visually enticing. This achievement comes as a

result of using what are often referred to as levels-of-detail, or LOD. Levels-of-detail refers to changing the complexity of the object, effect, or scene being rendered based upon distance from the viewer.

The concept comes from the observation that when viewing objects up close our eyes notice a larger amount of detail on the objects, however, when viewed far away, the objects may appear simply as specks, or on a computer screen, a single pixel. It should become readily apparent that implementing any sort of LOD schema into a rendering program would prove to be beneficial.

For our implementation we have added an effective yet simple LOD schema which allows for large improvements in frame rates while not sacrificing visual quality. The schema we have chosen is based upon our use of volumetric textures and involves adjusting the number of view-plane slices through the volume.

The basic principle of this schema concerns itself with how volumetric textures are rendered. In order to view them, planes must be sliced through the volume to be texture mapped to. These slices are treated exactly the same as any partially transparent polygons in any particular 3D rendering API. As such, the slices have a certain degree of transparency and as more slices are placed behind one another, the more blending of these slices occur decreasing frame rates (as more calculations are involved to determine the final color to be rendered for a particular pixel).

To cut down on the number of calculations, thus improving the frame rate, a volume can be rendered using fewer slices. The downside from lowering the number of slices comes when each volumetric texture coordinate does not map to a slice, or rather

there are fewer slices cutting through the volume than are required to accurately display each portion of the given texture map. Up close, such artifacts may not be noticeable, however at a distance; we can find that the volume of fewer slices does not noticeably change appearance enabling the frame rates to go higher without a loss in visual quality. We have in fact found that for a 32x32x32 cloud volume, as few as 20 slices can accurately display a cloud at a reasonably far distance without any loss in quality. Up close anywhere from 170 to 200 slices may be required to display all of the voxel data of the cloud texture.

## 5 IMPLEMENTATION

In Section 4 we discussed many of the topics we would use in our implementation of a cloud rendering program. In this section, we will actually focus on the implementation itself. Section 5.1 deals specifically with details of the implementation and the application of material covered in Section 4. And Section 5.2 covers particular implementation issues relevant to the results discussed in Section 6.

### 5.1 A Cloud Rendering Program

We present here a method for rendering a cloud on current commercially-available computer hardware. The method described here is essentially a breakdown of the single cloud renderer from start to finish. We will discuss the details of particular parts of the rendering process as they occur. It should be noted that this rendering context only takes into account rendering one cloud at a time focused on a fixed location. The rendering context also uses data from a file to define the cloud rather than having the simulation running at the same time. We will describe the reasons for these constraints later in Section 5.2.

Initially, we must first load the cloud from the file into our system. We have a data structure set up to represent each voxel. As data is loaded, the location of the light source is given and many of the voxel specific calculations not dealing with the accumulation or the actual calculation of light intensities are performed. These

calculations include constructing vectors, finding the blur angle, and finding  $\rho'$  as described in Section 4.2

Once the data is loaded we must apply our shadowing model. Pseudocode for how we perform this is given in Appendix A. At this point, we have performed step 1 in the algorithm and must now go to step 2, or determining the list of the nearby voxels. The reasoning behind doing this is for the scattering effect of light through clouds. If a voxel is thought of as a region of space of a small size, it is theoretically possible to have light coming in from any direction. We must then consider all surrounding voxels, or rather those which share vertices with the current voxel. This implies that there are 26 possible voxels which can contribute light. We only add voxels to the near-voxel list which have distances to the light source smaller than the current voxel, and for only one light source this can be at most 13 (if light comes in to the voxel directly through an a vertex to the center, a true 45 degree angle).

We next must sort the voxels based upon their distance to the light source. In reality, we do not sort the voxels but rather a list of the indices of the voxels, or where they appear in the volume. We construct this list in the previous step to reduce wastful looping through the volume. For the sorting, we use randomized quicksort as described in [3]. This algorithm runs in  $O(n \log n)$  time. It is possible to achieve even faster run times, even  $O(n)$ , but we wait and discuss this in Section 7.

Once the indices are sorted, we process this list in ascending order. For each voxel we process, we require the use of the average accumulation of alpha terms of the nearby voxels. If there are no nearby voxels which receive light first, we instead use the



intensity of the light itself. The calculated intensity, based upon the calculations performed in step 1, gets applied to the texture red, green, and blue values as a percentage of 255 and the actual light intensity for each of those respective colors. This results in the actual shading of the voxel. Once the shading has been applied, we use the equations in Section 4.2 to store the accumulated alpha term for the voxel and move to the next one. Our shading algorithm works since we are only processing voxels based upon voxels which have been previously processed.

Once the self-shadowing has finished, we actually render the texture. This process is described in full in [15] and demonstrated by a demo source code program provided by ATI [1]. As such, we do not focus on the details. The only addition which can be made to our rendering context is determining the number of slices for our LOD schema; we render other objects normally afterward.

Prior to rendering the cloud texture, we determine the number of slices to render it with. We use a number of slices in multiples of 10 based upon the distance the center of the cloud is from the viewer. This is a simple distance calculation combined with a scaling factor and a multiply using test conditions to prevent the number from becoming too large or too small. Results of implementing this process can be seen in Section 6.

## **5.2 Implementation Issues**

As discussed in Section 5.1, our rendering context only handles a single cloud loaded from a file rather than a simulation and viewed only at the origin. We will discuss the loading from a file first and then the rendering issues second.

The main reason a file is used in place of an actual simulation is because of time constraints. To properly develop a system which would simulate cloud formation and process rendering at the same time would require programming a multithreaded application and spending considerably more time on the project than what was available. We did have at our disposal a simulation which was used to obtain the test cloud data, but addition of this simulation to the rendering context we created would prove to be much more complicated for the time given and for other reasons explained next.

The other large implementation issue is with the volume rendering itself. Currently, our implementation is limited by having the view always focused on the center of the cloud texture. As we have focused primarily on the appearance of the texture, this is alright, however, we would like to be able to extend beyond this capability and currently the mathematics behind the model space transformations do not readily allow this conversion. It is possible, but extending our system to encompass this capability is again dependent upon time and beyond our reach at this point.

## 6 RESULTS

So far we have shown some of the previous work in rendering clouds, examined actual clouds in nature, discussed some of the features our system uses, and discussed our overall implementation in full detail. In this section we will display our results gathered over the time this project has elapsed.

Initially, we began by constructing a system in which to test and monitor cloud growth. It featured file loading of 3ds file formats, camera fly-through, fps monitoring, frustum culling, and early LOD support if only by change of color of boxes representing clouds. In actuality, the LOD support was much greater than that but the appearance in our system only displayed that. A screenshot of this system can be viewed in Figure 9.



Figure 9: Screenshot of cloud rendering framework.

We quickly found ourselves trying to do too many things at once and focused primarily on rendering the cloud itself. We went with volumetric textures knowing in the future a physical simulation would be used in conjunction with our rendering program. Initially, to just have a cloud appear on the screen, we rendered a texture of complete white voxels with the cloud data being applied strictly to the alpha terms. The results of this initial rendering can be seen in Figure 10. Obviously this lacks shading thus making the cloud appear like nothing more than a white blob. This would be where we would head next.

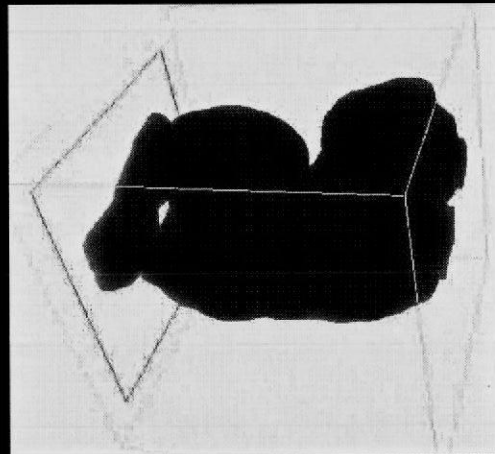


Figure 10: Initial cloud rendering.

For the self-shadowing of clouds, we found the use of the traditional light absorption and emission model [12] to be the most prevalent and more accurate. We apply an augmented form of this model as described in Section 4.2.

With the shading model in place, we began moving toward implementing a system using levels-of-detail. Our slice model proved to be quite effective yielding frame rates as can be seen in Table 3.

| Visible Slices | Approx. Minimum Frame Rate (fps) |
|----------------|----------------------------------|
| 200            | 27                               |
| 170            | 95                               |
| 140            | 200                              |
| 20             | 320                              |

Table 3. Frame rates at varying slices through volume texture.

For testing, we specify two conditions for success. The first condition is that the rendering must occur at interactive rates. We define interactive rates as typically above 15 fps and preferably at least 24 fps. Since our frame rates are well above a single frame per second and generally fall into the range of more than 30 fps, we have not only achieved real-time but also display of the cloud at a rate where motion through the scene appears to be constant and without jitters. This frame rate is possible because of our asymptotic runtime.

The runtime of this renderer is very easy to calculate as the main bottleneck in performance lies within the self-shadowing component. There we must first process the voxels. We perform this operation only once per voxel giving  $O(n)$ . The second step is to sort the voxels, which we have already stated as having  $O(n \log n)$  runtime. Next we build the nearby voxel lists which again occurs once for each voxel, or  $O(n)$ . And

finally we apply the shading calculations for each voxel again giving  $O(n)$ . When combined, the whole process evaluates to  $O(n \log n)$  showing that the sorting of the voxels is the determining factor in how long it takes to display the final cloud.

The second test condition is visual acceptance. This is harder to rate than that of a real-time system but visual inspection by the eye shows we have clouds which appear similar to real clouds as seen in Section 3. A final image is displayed in Figure 11. More images can be seen in Appendix B.

All testing was performed on an Athlon XP 1900+ with 768 MB 333 MHz DDR RAM and an ATI Radeon 9800 XT video card with 256 MB of onboard RAM.

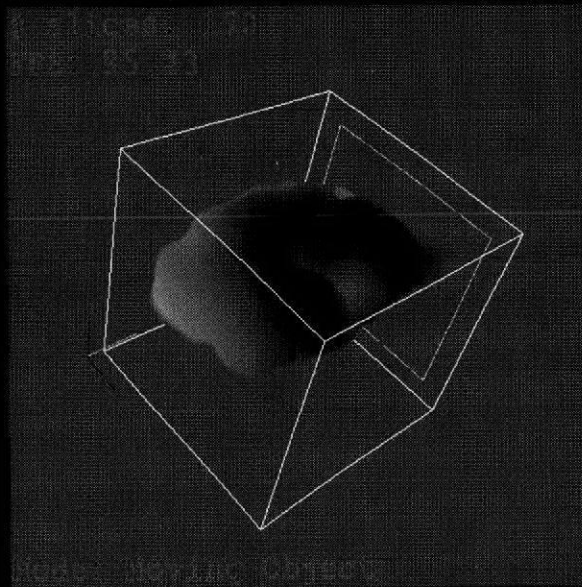


Figure 11: Final rendered image with LOD model applied.

## 7 FUTURE WORK

In this section we describe possibilities for future work on real-time rendering of clouds. Section 7.1 describes how we would apply alternate sorting algorithms or possibly using various other methods for choosing an order to process voxels to shade our clouds. We suggest alternative LOD schemas use to use in Section 7.2 and the complexities associated with implementing these schemas. In Section 7.3 we discuss the future possible use of shading languages to increase rendering speed and quality. Finally in Section 7.4 we discuss more obvious extensions to our rendering system that would give an applicable use to the system.

### 7.1 Voxel Processing

In Section 5.1 we mentioned the possibility of improving the speed of sorting of the voxels to be used for processing, and we there mentioned the possibility of achieving  $O(n)$  runtime. We suggest one possible method for improved sorting and another method which removes sorting altogether.

Given that our cloud is being rendered on a known domain of  $[-1,1]$  along each axis, we know that the real-world coordinates of each voxel are relatively close to one another. Furthermore, we know that the distance for each voxel to the light source must be nearly as close. Given this knowledge, we can determine a point at which the decimal portion of the light distances no longer matters and fix the length. We thus have a series of floating point values all with same length and within a few units of each

other. By this property, we can possibly devise a variation of radix sort [3] to run in  $O(n)$  time. A linear sorting would then allow us to use the same processing method but instead our self-shadowing code portion would run in  $O(n)$  total time versus  $O(n \log n)$  time, a significant improvement in performance.

We can also apply self-shadowing essentially without sorting. This method is spoken of in Kniss's work [12]. The basic principle is that for each light source a plane exists with normal pointing in the same direction as the light vector to be applied to the cloud. In fact, such a plane exists for every point  $t$  along the incident light ray. In the same manner by which slices are cut through the texture to render it, a plane can be used to make a cut through the cloud volume to create a list of voxels to be calculated next. Intuitively, there is loss of precision when using the above method as the voxels on the outer edges of the plane will have their shading calculated by an inaccurate light distance, but the advantage lies in that there is no need for sorting of the voxels removing a costly step, and when the light source lies at an infinitely far distance such as the sun, the loss of precision essentially does not matter. Another error exists in this method in that the light scattering would be dependent upon the voxels processed in the plane prior to the current plane being processed. What this means is that if the light was being determined by nearby voxels which when using this method would lie in the current plane, then this method would not even factor those voxels into the light scattering equations. There is minimal error in this as the light scattering is already an average so it is not of that much concern. Furthermore, adjustments can be made to the light scattering equations to counteract the error induced



## 7.2 Alternative LOD Schemas

In Section 4.3 we discussed the LOD schema we used in our system, however, other schemas exist which can be used instead or on top of our current schema to further increase frame rates requiring only minor alteration to our self-shadowing algorithm. We discuss two such methods in this section to give an idea of the many possibilities.

One LOD schema would be to induce approximations at certain computational step for varying distances. Up close, the system would use the computational methods as described in this paper. At distances, we could possibly treat many of the voxels as if having the same distance to the light source or multiply through by approximated constants instead of performing all of the calculations.

It should become clear from the above paragraph and work mentioned in this paper that another method would be to use smaller volumetric textures or even smaller voxel grids when performing computations. If textures of sizes of  $2^k$  on each side would still be used, lowering the size of  $k$  by 1 would yield an 8 fold speed increase roughly. This would most definitely improve rendering speeds considerably. The difficult part in applying such an improvement relies upon achieving this improvement without noticeably changing the appearance of the cloud from far away. Essentially, several factors used in the computation of the self-shadowing would have to be adjusted to account for the loss of voxel data or texture space that represents the cloud.

Finally, as used in SkyWorks [9,10], several precomputed or possibly even real-time billboard impostors could be used to represent cloud volumes at far distances. See Figure 12 for an example of this.

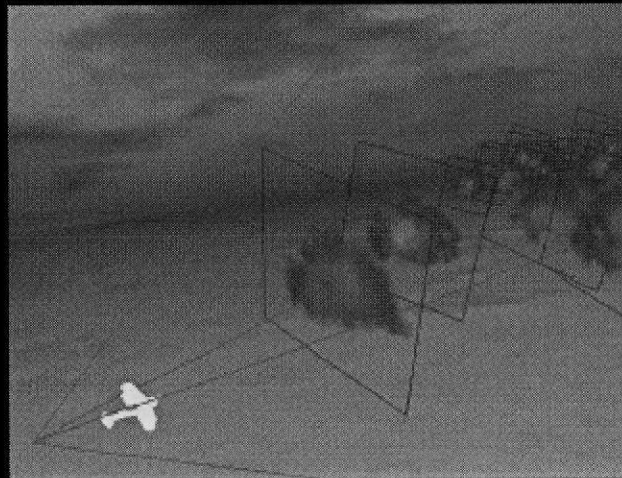


Figure 12: Billboard impostors. Image taken from [9].

### 7.3 Shading Languages

Recently, a trend among graphics programmers is to make use of hardware as much as possible to achieve higher frame rates. Recent hardware employs the use of vertex and pixel shaders to allow the use of small assembly-like coded programs to perform various operations on vertices and pixels. These programs are highly specialized for graphics programming by having built in support for common data structures, such as vectors and matrices, and common procedures, such as linear interpolations and the reordering of components of vectors. Moreover, these programs

can operate on textures and perform numerous lighting calculations as seen in Figure 13 where the shading language program generates texture coordinates for shadow-mapping.

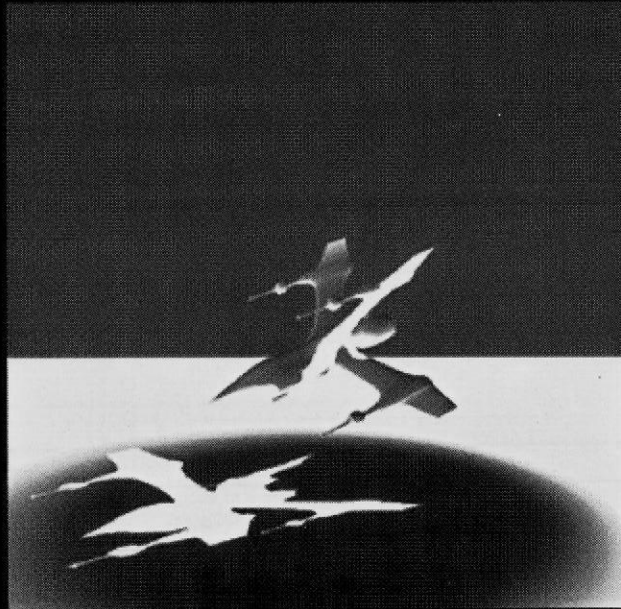


Figure 13: Shading language used for shadow-mapping. Image taken from [2].

Since our system works on volumetric textures and makes heavy use of vector calculations, an obvious extension to our system would be to add shading language support. In fact, it is theoretically possible to perform all or most of our shading calculations on the hardware through the use of shading languages. By doing so, we would leave the CPU to focus on the simulation almost entirely.

#### **7.4 Extensions to Current System**

As we have previously stated, our system is incomplete focusing on rendering a single cloud at the moment. We do have a framework in which to render multiple clouds

but the transition to this framework is not entirely simple. We would like to perform this transition among using other potential improvements to be discussed in the section.

In order to perform the transition and receive the rendering capability of our single cloud system, one possible direction to head in would be to use the method of rendering to a texture. The idea is that for each cloud we would compress the rendered image into a single 2D plane and store that as a texture, complete with alpha values. We would then use this texture much in the same way as billboard impostors [9,10], rendering planes aligned toward the viewer, although there does exist a problem if objects actually are within the cloud itself. We could theoretically use a procedure similar to [9,10] or rather just render that particular cloud as a full volumetric texture as we would be doing if we directly ported volumetric textures to our framework.

Many other extensions to our framework exist that would be nice to apply to receive clouds which are that much more realistic. Such extensions would possibly include: shafts of light, shadow casting by clouds onto real objects or other clouds, having clouds affected by multiple lights or even lightning, and even forms of precipitation. Each of these areas can easily be its own extensive project and research does continue on many of them currently.

## 8 SUMMARY AND CONCLUSIONS

In this paper we have presented a method for real-time rendering of cloud simulations showing our current rendering context which handles a single cloud structured so as to be updated by a simulator. We have discussed previous rendering methods and examined real-world clouds and their complexities as how they apply to rendering on a computer screen. We have discussed at length volumetric textures and operations on them and have shown how certain transfer functions can be applied to achieve a self-shadowing context. Our results have been presented and we have examined the various possibilities for future work.

We conclude that our system, while infant in nature, does show potential for use in a simulation environment and that acceptable results are feasible to obtain in real-time on commercially available computer hardware. It then becomes possible to apply our system to a larger scale framework and use our rendering method in either more useful simulations, such as for basic flight or training, and even possibly in entertainment.

## REFERENCES

- [1] "ATI Developer: Source Code," *ATI Technologies Inc*, <http://www.ati.com/developer/sdk/RADEONSDK/Html/Samples/OpenGL/>, [21 Feb. 2004].
- [2] *Cg Toolkit User's Manual*, ver. 1.1, p. 152, Santa Clara, CA: nVidia Corp., 2002.
- [3] T. Cormen et al., *Introduction to Algorithms*, 2nd ed., Cambridge, MA: MIT Press, 2001.
- [4] Y. Dobashi et al., "A Simple, Efficient Method for Realistic Animation of Clouds," *Proc. of SIGGRAPH '00*, pp.19–28, 2000.
- [5] D.S. Ebert, "Volumetric Modeling with Implicit Functions: A Cloud is Born," *Visual Proc. of SIGGRAPH '97*, p. 147, 1997.
- [6] P. Elinas and W. Sutzerlinger, "Real-time rendering of 3D clouds," *Journal of Graphics Tools*, 5(4), pp. 33–45, 2000.
- [7] J. Foley et al., *Introduction to Computer Graphics*, p. 382, Reading, MA: Addison Wesley, 1994.
- [8] G.Y. Gardner, "Visual Simulation of Clouds," pp. 297–303, *Proc. of SIGGRAPH '85*, 1985.
- [9] M.J. Harris and A. Lastra, "Real-Time Cloud Rendering," *Computer Graphics Forum (Eurographics '01 Proc.)*, 20(3):C–76 – C–84, 2001.
- [10] M.J. Harris, "SkyWorks Cloud Rendering Engine," *Dept. of Computer Science, Univ. of North Carolina*, <http://www.cs.unc.edu/~harrism/SkyWorks/>, [28 Oct. 2003].
- [11] J. Kniss et al., "Gaussian Transfer Functions for Multi-Field Volume Visualization," *Proc. IEEE Visualization 2003*, pp. 497–504, 2003.
- [12] J. Kniss et al., "A Model for Volume Lighting and Modeling," *IEEE Trans. on Visualization and Computer Graphics*, vol. 9, issue 2, pp. 150–162, Apr -June 2003.
- [13] D R Overby, *Interactive Physically-Based Cloud Simulation*, master's thesis, Texas A&M Univ., College Station, TX, 2002.

- [14] J. Schpok et al., "A Real-Time Cloud Modeling, Rendering, and Animation System," *Proc. of 2003 ACM SIGGRAPH/Eurographics Symp. on Computer Animation*, pp. 160-166, 2003
- [15] H. Shen, "Volume Rendering," *Dept. of Computer Science and Eng., Ohio State Univ.*, <http://www.cis.ohio-state.edu/~hwshen/788/VR.ppt>, [15 Feb. 2004].

#### **SUPPLEMENTAL SOURCES CONSULTED**

M. Woo et al., *OpenGL Programming Guide*, 3rd ed., Reading, MA: Addison Wesley, 1999.

**APPENDIX A - SELF-SHADOWING PSEUDOCODE**

1. Calculate data for each voxel (distance to light, relative density approximation, voxel angle, etc.)
2. Create a list of voxels near to each voxel that are closer to the light source.
3. Sort voxels by ascending distance to light source; store indices in array L.
4. for  $i := 0$  to  $\text{size}(L) - 1$ 
  - a. Calculate and store opacity of voxel  $i$ .
  - b. Calculate and store color of voxel  $i$  (using opacity).
  - c. Accumulate opacity terms.



## APPENDIX B – ADDITIONAL IMAGES

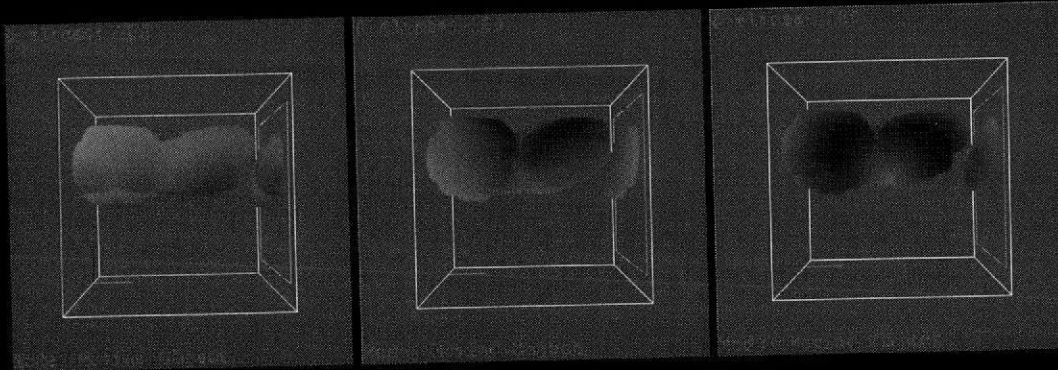


Figure 14: Self-shadowing with light positioned along positive axes. Left: x-axis. Middle: y-axis. Right: z-axis. All views are toward the negative z-axis.

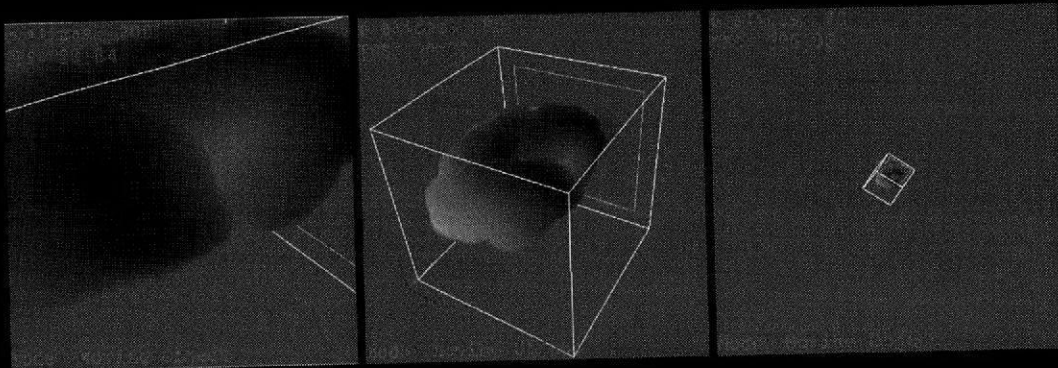


Figure 15: LOD model applied to rendering system. Left: Close-up, 200 slices at approximate minimum 28.84 fps. Middle: Full-screen, 180 slices at 70.92 fps. Right: Far away, 20 slices at approximate maximum 325.92 fps.

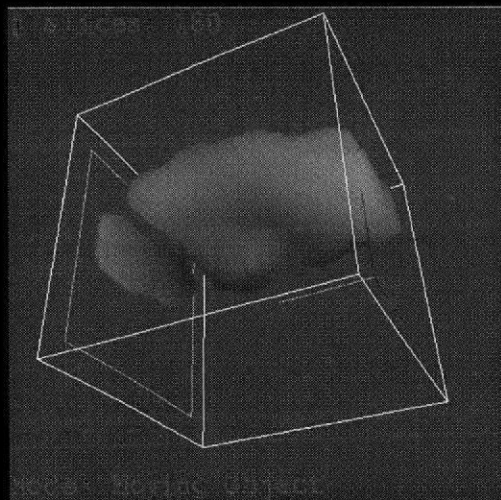


Figure 16: Underside of cloud with respect to light source.

**VITA**

Kevin Michael Walkington  
1306 Rona St.  
Weatherford, TX 76086  
(817) 596-9797

Kevin Michael Walkington is an undergraduate at Texas A&M University pursuing his Bachelor's Degree in Computer Science with a Minor in Mathematics. This degree will be awarded in May of 2004, at which time he will graduate with University Honors, Foundation Honors, and Undergraduate Research Fellows designations. During his undergraduate career he has held the distinction as being a Lechner Scholar, an SBC Scholar, and a member of the Engineering Scholars Program. He has membership in numerous honors societies, including National Society of Collegiate Scholars, Golden Key Society, Phi Eta Sigma, Phi Kappa Phi, and Upsilon Pi Epsilon, and other societies including Percussion Studio. During his research, he worked with a previous Texas A&M student, Derek Overby, and gave presentations to both his peer Fellows and their advisors as well as a presentation at Student Research Week. After graduation, he plans to return in the fall to Texas A&M University to pursue his Master's Degree in Computer Science.