

©Copyright 2013 DigiPen Institute of Technology and DigiPen (USA) Corporation. All rights reserved.

Rendering Techniques for Cloud Volume Densities with Shadows

BY
Nicolas Giuliatti
Bachelor of Science, Real-Time Interactive Simulation
DigiPen Institute of Technology, May 2008

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science
in the graduate studies program
of DigiPen Institute Of Technology
Redmond, Washington
United States of America

Fall
2013

Thesis Advisor: Dr. Garry Herron

DIGIPEN INSTITUTE OF TECHNOLOGY
GRADUATE STUDY PROGRAM
DEFENSE OF THESIS

THE UNDERSIGNED VERIFY THAT THE FINAL ORAL DEFENSE OF THE
MASTER OF SCIENCE THESIS OF NICOLAS GIULIETTI

HAS BEEN SUCCESSFULLY COMPLETED ON NOVEMBER 25, 2013

TITLE OF THESIS: RENDERING TECHNIQUES FOR CLOUD VOLUME
DENSITIES WITH SHADOWS

MAJOR FIELD OF STUDY: COMPUTER SCIENCE.

COMMITTEE:

Gary Herron, Chair

Jason Hanson

Pushpak Karnick

Xin Li

APPROVED:

Dmitri Volper date
Graduate Study Program Director

Xin Li date
Dean of Faculty

Dmitri Volper date
Department Chair of Computer Science

Claude Comair date
President

The material presented within this document does not necessarily reflect the opinion of the Committee, the Graduate Study Program, or DigiPen Institute of Technology.

INSTITUTE OF DIGIPEN INSTITUTE OF TECHNOLOGY
PROGRAM OF MASTER'S DEGREE
THESIS APPROVAL

DATE: 11/25/2013

BASED ON THE CANDIDATE'S SUCCESSFUL ORAL DEFENSE, IT IS
RECOMMENDED THAT THE THESIS PREPARED BY
NICOLAS GIULIETTI

ENTITLED
RENDERING TECHNIQUES FOR CLOUD VOLUME DENSITIES WITH
SHADOWS

BE ACCEPTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE
DEGREE OF MASTER OF COMPUTER SCIENCE FROM THE PROGRAM OF
MASTER'S DEGREE AT DIGIPEN INSTITUTE OF TECHNOLOGY.

Gary Herron
Thesis Advisory Committee Chair

Dmitri Volper
Director of Graduate Study Program

Xin Li
Dean of Faculty

The material presented within this document does not necessarily reflect the opinion of the Committee, the Graduate Study Program, or DigiPen Institute of Technology.

Table of Contents

Abstract.....	1
Chapter 1: Introduction.....	2
1.1 Goal.....	3
1.2 Organization.....	3
Chapter 2: Volume Rendering Equation.....	4
Chapter 3: Survey.....	14
3.1 Techniques Used for Modeling Clouds.....	14
3.1.1 Introduction.....	14
3.1.2 Particle System.....	14
3.1.2.1 Modeling Clouds.....	15
3.1.3 Volume Texture.....	15
3.1.3.1 Modeling Clouds.....	16
3.1.4 Mesh.....	17
3.1.4.1 Marching Cubes.....	17
3.1.4.2 Modeling Clouds.....	21
3.2 Techniques Used for Rendering Clouds.....	22
3.2.1 Introduction.....	22
3.2.2 Particle System.....	22
3.2.2.1 Billboarding.....	23
3.2.2.1.1 Screen-Aligned Billboarding.....	24
3.2.2.2 Rendering Clouds Using a Particle System.....	25
3.2.2.2.1 Shader Constant Instancing.....	25
3.2.2.3 Spherical Billboards.....	25
3.2.3 Volume Ray Tracing.....	28
3.2.3.1 Rendering Clouds Using Volume Ray Tracing.....	28
3.2.3.1.1 Transfer Function.....	29
3.2.4 Fog Polygon Volumes.....	29
3.2.4.1 Rendering Clouds Using Fog Polygon Volumes.....	30
3.3 Techniques Used for Rendering Cloud Shadow.....	32
3.3.1 Introduction.....	32
3.3.2 Surface Shadows.....	33
3.3.2.1 Traditional Depth Shadow Map.....	33
3.3.3 Volumetric Shadows.....	34
3.3.3.1 Deep Shadow Map.....	35
3.3.3.2 Opacity Shadow Map.....	38
3.3.3.3 Half-Angle Slicing.....	39
3.3.3.4 Deep Opacity Map.....	42
3.3.3.5 Fourier Opacity Map.....	44
Chapter 4: Results.....	45
4.1 Platform.....	46
4.2 Rendering Clouds of Smoke with Shadows.....	46
4.2.1 Speed Results.....	46
4.2.2 Accuracy Results.....	48

4.2.2.1 Images.....	48
4.2.2.2 Accuracy Comparison.....	51
4.2.3 Rendering Clouds of Smoke with Shadows Analysis.....	57
Chapter 5: Conclusion and Future Work.....	59
References.....	61

Abstract

The purpose of this research is to analyze and compare different approaches to cloud volume (smoke, steam, and dust) rendering with shadows for computer graphics in order to determine levels of accuracy and performance for each method. Accuracy is a measure of realism, and performance is a measure of speed for a fixed amount of memory. Traditional surface rendering techniques are not appropriate for rendering of semi-transparent cloud volumes. Instead, techniques that consider the volumetric qualities of clouds should be used. Also, modeling of clouds is generally different from that of modeling surfaces, because clouds have inner densities that surfaces do not have. Shadows are an important part of the appearance of clouds. They allow for more accurate representations of clouds by adding essential visual cues. However, shadow techniques may have artifacts that impact overall accuracy or realism of an image.

In order to determine levels of accuracy and performance for methods of rendering clouds with shadows, the author surveyed the field of rendering techniques for cloud volumes with shadows, implemented several techniques, and analyzed different approaches. Accuracy and performance results of all implementations were compared with each other.

Chapter 1: Introduction

For the purposes of this paper, we use the word “cloud” to mean any volumetric cloud density such as a cloud of smoke, steam, or dust. The visual appearance of a cloud can be quite interesting. It can take on many shapes and sizes, and have various densities. Some clouds are semi-transparent, while other more dense clouds are more opaque. The interaction of light and clouds is of special interest, because it is what gives the clouds their characteristic look. An important characteristic of the look of clouds is the appearance of shadows in them, and around them. Shadows give us visual information that allows use to make judgments about the clouds, and the space that they are in. For example, they can give us an ideal about how far the clouds are from other objects. They can tell us a little about how thick or dense parts of the clouds are. If there is a light in the area of the clouds, shadows can sometimes give us an ideal about where the light is coming from.

In computer graphics, there are several popular ways to represent clouds. Choosing one over another can significantly change the look of the final image that is drawn. One representation may be particularly less accurate than another for drawing clouds. In addition, the performance gained by choosing one representation over another can be large. The memory usage between different representations for a desired level of accuracy varies too. Perhaps one representation is particularly suited for rendering one type of cloud better than the others. Where one representation may fail, another may excel. In fact, some representations were developed as a result of the shortcomings of another. Cloud rendering is a combination of a modeling technique, a volume rendering technique, and a volume shadowing technique.

1.1 Goal

The aim of this research is to analyze many of the techniques, and determine levels of accuracy and performance for drawing an image of clouds using vertex and pixel shaders based on results produced by implementations and experiments. For the purposes of this paper, we define accuracy as a measure of realism, and performance as a measure of speed for a fixed amount of memory. To be clear, the goal is to determine levels of accuracy and performance for a single combination of a modeling technique, a rendering technique, and a shadowing technique. A side effect of the research is determining the levels of accuracy and performance of cloud modeling, rendering, and shadowing techniques separately. The research should show which techniques are more suitable for different situations including desired accuracy, performance, and cloud appearance. In addition, we aim to organize and layout the progression of the field of cloud volume graphics with shadows for reference.

1.2 Organization

The research is organized by first developing several volume rendering equations that are referenced in every volume rendering technique that follows. There are three root sections in chapter 3. The first section starts with how clouds are modeled using different modeling techniques. The next section is an examination of various techniques used to render the clouds modeled in the previous section, and the final section of chapter 3 deals with how cloud volume rendering is extended with shadows. The order in which each technique was originally developed was preserved, so that the research shows how the field progressed. Chapter 4 lists results from the experiments performed with our implementations, compares the results, and explains the results. In addition, it explains how we compare accuracy or realism for different shadow techniques.

Chapter 2: Volume Rendering Equation

This entire chapter consists of an explanation and derivation of several useful volume rendering equations. Some of the equations are used throughout the entire paper. All of the material in this chapter came from chapter 1 of the book called *Real-Time Volume Graphics*. [EHKRW06]

The physics behind volume rendering relies on geometric optics. Light is assumed to travel along a straight line unless interaction between light and participating medium takes place. The interaction between light and participating medium can be described by emission, absorption, and scattering. Emission occurs when a material emits light. In reality, hot gases emit light by converting heat into radiative energy. Absorption occurs when light travels through a material that converts the light into heat, so that light energy is reduced. Scattering occurs when a material changes the direction of light propagation. If the wavelength (or energy of photons) is not changed by scattering, the process is called elastic scattering. Conversely, inelastic scattering affects the wavelength, and will not be considered in this paper.

Radiance I is a measure of energy of light which is affected by emission, scattering, and absorption. Radiance is defined as:

$$I = \frac{dQ}{dA_{\perp} d\Omega dt}$$

where Q is radiative energy, A_{\perp} is a unit of area projected along the direction of the light, Ω is solid angle, and t represents time. $A_{\perp} = A \cos(\vartheta)$, if ϑ is the angle between the light direction and the normal vector on the surface of A .

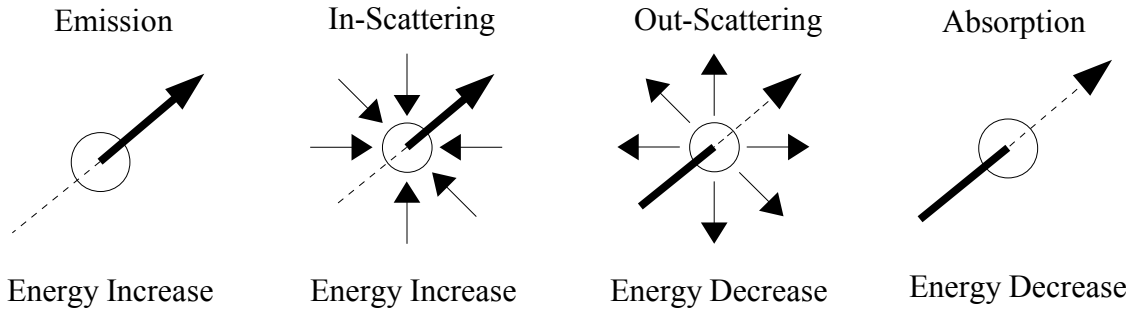


Figure 1: Interactions between light and participating media that affect the radiance along the ray.

By combining the absorption, emission, and scattering effects, the following equation for the transfer of light is obtained:

$$\omega \cdot \nabla_x I(X, \omega) = -\chi I(X, \omega) + \eta$$

The term $\omega \cdot \nabla_x$ is the dot product of the light direction ω and the gradient of radiance I with respect to position x . The dot product describes the directional derivative taken along the light direction. The term χ is the total absorption coefficient or attenuation, and the quantity η is the total emission.

The total absorption coefficient χ consists of true absorption k , and scattering coefficient σ . Both k and σ represent energy loss. The total absorption coefficient can be written as

$$\chi = k + \sigma$$

The total emission coefficient η consists of source term q , and scattering term j . Both q and j represent energy gain. The total emission coefficient can be written as

$$\eta = q + j$$

Note that χ, η, k, σ, q , and j depend on position x , and direction ω along a light ray. Terms k, σ , and q are optical material properties assigned by transfer functions, a physical model of gas, or simply noise. The scattering term j , however, needs

to be indirectly computed from material properties by considering all possible contributions of all incoming light directions. Scattering term j can be written as

$$j(x, \omega) = \frac{1}{4\pi} \int_{\text{Sphere}} \sigma(x, \omega') p(x, \omega', \omega) I(x, \omega') d\Omega'$$

For the scattering term j , contributions from incident light $I(x, \omega')$ are accumulated by integrating over all directions ω' . The contributions are weighted by the scattering coefficient σ and phase function p , which describes the chance that light is scattered from the original direction ω' to new direction ω . We assume that the phase function is normalized according to

$$\frac{1}{4\pi} \int_{\text{Sphere}} p(x, \omega', \omega) d\Omega' = 1$$

The factor $\frac{1}{4\pi}$ is used to cancel the factor of 4π that is picked up by integrating a unit function over a whole sphere.

By combining emission, absorption, in-scattering, and out-scattering the complete equation for the transfer of light is obtained:

$$\omega \cdot \nabla_x I(x, \omega) = -(k(x, \omega) + \sigma(x, \omega)) I(x, \omega) + q(x, \omega) + \int_{\text{Sphere}} \sigma(x, \omega') p(x, \omega', \omega) I(x, \omega') d\Omega'$$

Because the solution of the complete equation of transport of light is computationally expensive, simplified models are often used. One or more terms in the complete equation is removed or simplified for efficiency. The following are some common optical models:

Absorption Only. The volume is assumed to consist of cold, perfectly black material that may absorb incident light. No light is emitted or scattered.

Emission Only. The volume is assumed to be gas that only emits light but is completely transparent. No absorption or scattering occurs.

Emission-Absorption Model. The gas can emit light and absorb incident light. However, scattering and indirect illumination are neglected.

Single Scattering and Shadowing. This model includes single scattering of light that comes from an external light source (i.e. not from the volume). Shadows are modeled by taking into account the attenuation of light that is incident from an external light source.

Multiple Scattering. Here, the goal is to evaluate the complete illumination model volumes.

In order to represent a gas, a particle system may be used with the goal being to accurately represent the complete volume rendering equation. A particle system is just a large number of quadrilaterals. Commonly, each quadrilateral has a number of attributes such as color, texture, and an alpha value. With only the ability to set color, texture coordinates, and alpha for each vertex of a quadrilateral before rendering, we often represent a volume using the Emission-Absorption Model. The Emission-Absorption Model does not represent scattering and shadowing, but it is rather simple and efficient.

In order to write the Emission-Absorption Model volume rendering equation using the same notation above, we set the scattering terms in the complete volume rendering equation to zero and get:

$$\omega \cdot \nabla_x I(x, \omega) = -k(x, \omega) I(x, \omega) + q(x, \omega)$$

If only a single ray of light is considered, we can rewrite the Emission-Absorption Model volume rendering equation as

$$\frac{dI(s)}{ds} = -k(s) I(s) + q(s)$$

where positions are described by the length parameter s .

The form of the Emission-Absorption equation above can be solved for radiance by integrating along the direction of light flow from the starting point $s=s_0$ to the ending point $s=D$, resulting in the following volume-rendering integral

$$\text{(Equation 1.0)} \quad I(D) = I_0 e^{-\int_{s_0}^D k(t) dt} + \int_{s_0}^D q(s) e^{-\int_s^D k(t) dt} ds$$

The term I_0 represents the light entering the volume from the background. $I(D)$ is the radiance leaving the volume at $s=D$ and finally reaching the camera. The first term describes the light from the background attenuated by the volume. The second term represents the integral contribution of the source terms attenuated by the participating medium along the remaining distances to the camera. The term

$$\tau(s_1, s_2) = \int_{s_1}^{s_2} k(t) dt$$

is defined as the optical depth between positions s_1 and s_2 . The optical depth is a measure of how long light may travel before it is absorbed. In other words, optical depth indicates the typical length of light propagation before scattering occurs. Small values of optical depth mean that the medium is rather transparent, and high values of optical depth are associated with a more opaque material. The corresponding transparency for a material between $s=s_1$ and $s=s_2$ is

$$\text{(Equation 1.1)} \quad T(s_1, s_2) = e^{-\tau(s_1, s_2)} = e^{-\int_{s_1}^{s_2} k(t) dt}$$

With this definition of transparency, we obtain a slightly different version of the volume-rendering integral

$$\text{(Equation 1.2)} \quad I(D) = I_0 T(s_0, D) + \int_{s_0}^D q(s) T(s, D) ds$$

The volume rendering equation still neglects scattering. One way to incorporate single scattering is to compute the gradient of the scalar field, and use it as a normal for

local illumination like Phong or Blinn-Phong. The gradient serves as a normal, because the gradient is identical to the normal of the isosurface through the respective point in space. Therefore, volume shading can produce an effect similar to an illuminated isosurface. Local illumination is included in the volume-rendering integral by extending the source term to

$$q_{extended}(s, \omega) = q_{emission}(s, \omega) + q_{illum}(s, \omega)$$

Sometimes, optical properties are derived from a density model of the participating medium. In this description, ρ represents the density of the material and all optical properties are weighted by the density. For example, the total absorption coefficient χ is replaced by χ' according to

$$\chi = \chi' \rho$$

There are similar substitutions for the true absorption coefficient, $k = k' \rho$, the scattering coefficient, $\sigma = \sigma' \rho$, and the true emission term $q = q' \rho$.

The volume-rendering integral cannot be evaluated analytically. Instead, numerical methods are applied to find an approximation as close to the solution as possible. A common approximation splits the integration domain into n intervals. The intervals are described by locations $s_0 < s_1 < \dots < s_{n-1} < s_n$, where s_0 is the starting point of the integration domain and $s_n = D$ is the endpoint. Please note that the intervals do not necessary have equal lengths.

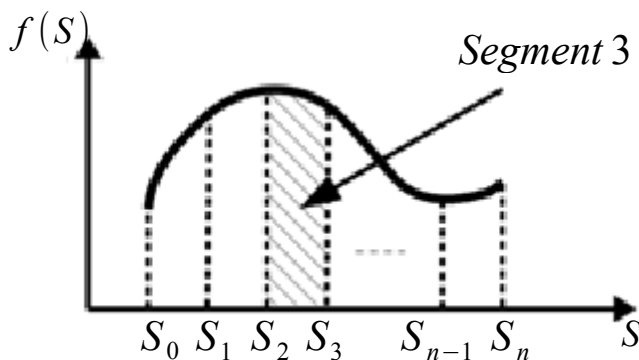


Figure 2: Partitioning of the integration domain into several intervals. The intervals are described by locations $s_0 < s_1 < \dots < s_{n-1} < s_n$. The i th interval or segment is $[s_{i-1}, s_i]$. The hatched box indicates the integration result for the third segment.

Considering the light transport within the i th interval $[s_{i-1}, s_i]$ (with $0 < i \leq n$) we can obtain the radiance at s_i according to.

$$I(s_i) = I(s_{i-1})T(s_{i-1}, s_i) + \int_{s_{i-1}}^{s_i} q(s)T(s, s_i)ds$$

We introduce an new notation for the transparency T and color contribution c (i.e., the radiance contribution) of the i th interval:

$$T_i = T(s_{i-1}, s_i), \quad c_i = \int_{s_{i-1}}^{s_i} q(s)T(s, s_i)ds$$

The hatched box in the figure above illustrates the result of integration over one interval. The radiance at the endpoint of the volume is then given by

$$I(D) = I(s_n) = I(s_{n-1})T_n + C_n = (I(s_{n-2})T_{n-1} + C_{n-1})T_n + C_n = \dots$$

Which can be written as

(Equation 1.3)
$$I(D) = \sum_{i=0}^n c_i \prod_{j=i+1}^n T_j, \text{ with } c_0 = I(s_0)$$

Often, transparency T_i is replaced by opacity $\alpha_i = 1 - T_i$.

Now that the integration domain is segmented into n discrete intervals, and the summations and multiplications can be computed, the last thing that we need for volume-rendering is the evaluation of transparency and color contributions of the intervals. A rather common approach approximates the volume-rendering integral by a Riemann sum over n equidistant segments of length $\Delta x = (D - s_0)/n$. Here the function to be integrated is approximated by a piecewise-constant function. The integral over a single interval corresponds to the area of the rectangle defined by the function value evaluated at a sampling point and by the sampling width.

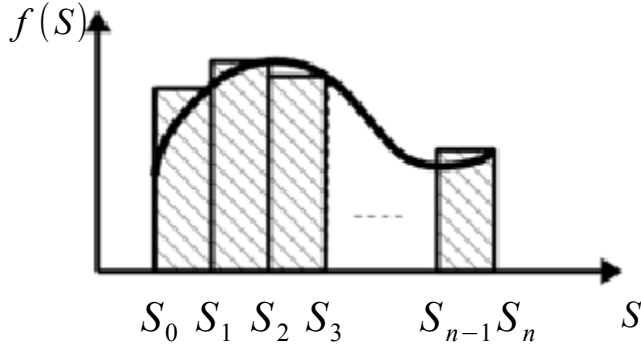


Figure 3: Approximation of an integral by a Riemann sum.

In this approximation, the transparency of the i th segment is

$$T_i \approx e^{-k(s_i)\Delta x}$$

and the color contribution for the i th segment is

$$c_i \approx q(s_i) \Delta x .$$

There are two compositing schemes used to evaluate radiance in **Equation 1.3** iteratively: front-to-back compositing and back-to-front compositing. Front-to-back compositing is sometimes called the *under* operator, while back-to-front compositing is sometimes called the *over* operator. The front-to-back compositing is applied when the viewing rays are traversed from the eye point into the volume. Let C represent a color, typically given as a three channel RGB (red, green, blue) color value. The front-to-back iteration equations are:

$$\begin{aligned} \hat{C}_i &= \hat{C}_{i+1} + T_{i+1} C_i, \\ \hat{T}_i &= T_{i+1} (1 - \alpha_i), \end{aligned}$$

with the initialization

$$\begin{aligned} \hat{C}_n &= C_n, \\ \hat{T}_n &= 1 - \alpha_n \end{aligned}$$

The results of the current iteration step are \hat{C}_i and \hat{T}_i . \hat{C}_{i+1} and \hat{T}_{i+1} are the accumulated results of the previous computations. The source term C_i and the opacity

α_i are given by the transfer function or originate from a physical model of gas. The iteration first starts at the sampling position $i = n$ (closest to the camera) and ends at $i = 0$ (at the backside of the volume).

By renaming the variables according to $C_{dst} = \hat{C}_j$ (with $j = i, i + 1$), $\alpha_{dst} = 1 - \hat{T}_j$ (with $j = i, i + 1$), and $\alpha_{src} = \alpha_i$, front-to-back compositing can be rewritten in its most common form:

$$\text{(Equation 1.4)} \quad \begin{aligned} C_{dst} &\leftarrow C_{dst} + (1 - \alpha_{dst}) C_{src}, \\ \alpha_{dst} &\leftarrow \alpha_{dst} + (1 - \alpha_{dst}) \alpha_{src} \end{aligned}$$

Variables with subscript *src* (as for “source”) describe quantities introduced as inputs from the optical properties of the data set (e.g., through a transfer function or from a physical model of gas), whereas variables with subscript *dst* (as for “destination”) describe output quantities that hold accumulated colors and opacities. The compositing is repeatedly applied while marching along the ray, updating color C_{dst} and opacity α_{dst} along its way. By reversing the traversal direction, we obtain the back-to-front compositing scheme:

$$\begin{aligned} \hat{C}_i &= \hat{C}_{i-1} (1 - \alpha_i) + C_i, \\ \hat{T}_i &= T_{i-1} (1 - \alpha_i), \end{aligned}$$

with the initialization

$$\begin{aligned} \hat{C}_0 &= C_0, \\ \hat{T}_0 &= 1 - \alpha_0 \end{aligned}$$

The iteration starts at $i = 0$ and ends at $i = n$. Note that the accumulated transparency

\hat{T}_i is not needed to compute the color contribution \hat{C}_i , so it is omitted. We rewrite the back-to-front scheme like we did with the front-to-back scheme:

$$\text{(Equation 1.5)} \quad C_{dst} \leftarrow (1 - \alpha_{src}) C_{dst} + C_{src}$$

Note that there is no iterative update of opacity needed because accumulated opacity (or transparency) is not required to determine the color contribution.

In addition to the above compositing schemes, alternative approaches are sometimes used. For example, maximum intensity projection (MIP) and x-ray or weighted sum projections are often applied in medical imaging applications. MIP is computed according to the compositing equation:

$$C_{dst} \leftarrow \max(C_{dst}, C_{src})$$

The final result is the maximum color contribution along a ray. This compositing scheme is independent of the traversal order; i.e., it may be applied in a back-to-front, a front-to-back, or any other order. The main application for MIP is virtual angiography—the display of vessel structures in medical scans.

Chapter 3: Survey

3.1 Techniques Used for Modeling Clouds

3.1.1 Introduction

How clouds are modeled can have a large affect on efficiency, and the accuracy of the final cloud image. All of the techniques in this section are modeling techniques that an artist might use in order to model clouds. Some representations are more suitable for clouds than others. The techniques discussed in this chapter are not about how to render clouds. This entire chapter is devoted to how each technique can be used to model clouds, what the goals of each technique are, and how each technique differs from the others. All techniques may use the word “volume” in order to describe what the final result of the modeling process is, but some use the word more loosely than others for reasons that will be made clear.

3.1.2 Particle System

A particle system is defined as a method for modeling fuzzy objects such as fire, smoke, and water. Particle systems model an object as a cloud of primitives that define its volume.[R83] Particles systems were introduced after surface rendering techniques, because with particles an object is represented not by a set of primitive surface elements, such as polygons or patches, that define its boundary, but as clouds of primitive particles that define its volume. Simply by varying the number of particles in a region, a particle system can represent a volume of varying density. Another benefit that the particle system representation has over surface polygons is that an object represented by a particle system is not deterministic, since its shape and form are not completely specified. Instead,

stochastic processes are used to create an object's appearance. A particle (for now, think of a particle as a point in three-dimensional space) is a much simpler primitive than a polygon, the simplest of the surface representations. Therefore, in the same amount of computation time one can process more of the basic primitive and produce a more complex image.[R83] The model definition is procedural, so it can be controlled by random numbers. Therefore, obtaining a highly detailed model does not necessary require a great deal of human design time as is often the case with existing surface-based systems.[R83]

3.1.2.1 Modeling Clouds

We model clouds by simply choosing how many particles we want to use. Then, the position of each particle in 3D space is chosen by computing a random number, or by using a physics simulation. The size of each particle can be randomized too, or it can be controlled by a physics simulation. Each particle is assigned a color that is used during rendering, and each particle is given a transparency value that represents how much we can see through the particle. Describing how complicated clouds are modeled using a particle system is beyond the scope of this paper.

3.1.3 Volume Texture

Originally, a texture was the source of pattern for mapping patterns, pictures, or texture onto 3D surfaces.[C74] [BN76] The performance and quality of texture mapping has improved with techniques that are still used today. For example, fixed-rate texture compression directly attacks memory and bandwidth problems, and caching concerns. [MHH08] By having hardware decode compressed textures on the fly, a texture can require less texture memory and so increase the effective cache size.[MHH08] At least as

significant, such textures are more efficient to use, as they consume less memory bandwidth when accessed.[MHH08] Another improvement to textures is how they are sampled in order to reduce aliasing. Aliasing is an unwanted artifact that occurs when a signal being sampled is too low of a frequency.[MHH08] In order to reduce aliasing, several filter techniques have been introduced. A filter allows us to reconstruct the signal that may be represented by less data than the original signal.[MHH08]

Textures are not limited to two dimensions. In fact, they can have a third dimension. Another name for a 3D texture is volume texture. It makes sense to use a volume texture in order to model clouds, because techniques that we use for 2D textures, can be extended to 3D textures. Texture compression techniques can be applied to cube or volume textures.[MHH08] For an entire book that deals with 3D image processing see [TY09]. The book explains a number of 3D filters that can be applied to textures.

Using a 3D texture has the advantage of being able to show all important aspects of a data image in a single data set.[W07] In other words, a 3D texture can represent solid surface and low density features in one texture. Neither a particle volume nor a polygonal mesh surface has the ability to represent both solid surfaces and low density features using the same representation. For now, we can think about a 3D texture as being a multidimensional grid where each grid element is an integer. A volume texture has the same advantage over a traditional polygonal mesh representation in that the model definition can be procedural, so it can be controlled by random numbers.

3.1.3.1 Modeling Clouds

Clouds are modeled using 3D noise and/or a physics simulation, and the result is stored in a 3D texture. The 3D texture stores a multidimensional array of density or opacity values. Both noise and physics simulations are not discussed in this paper. For details on one type of noise that can be used to model clouds see [G05].

3.1.4 Mesh

A polygon is a geometric shape that is made up of three or more points.[S08] The more points that are used, the more complex a shape can look.[S08] Triangles are three-point polygons whose three edges are used to connect each of the points that make up the shape of the primitive.[S08] Triangles are the most common type of primitive used in 3D video games.[S08] Triangles are so common because they have a number of useful properties.[R10] They are the simplest primitive that describes a surface in space, it is simple to linearly interpolate values across them, and they can be used to construct a number of higher order primitives.[R10] When we group a number of triangles together, we call it a mesh. The mesh structure predates the introduction of particles and volume textures. Advancements have made it an attractive representation for modeling clouds. One advancement allows a mesh to be procedurally generated like a particle volume, or a volume texture.

3.1.4.1 Marching Cubes

Marching Cubes (MC) is a algorithm used for creating a constant density triangular mesh surface.[LC87] A mesh is generated from density evaluations, or from simple inside and outside binary tests performed for every cube in a 3D array of cubes (voxel buffer). The algorithm determines how a surface intersects a single cube in the voxel buffer, then moves (or marches) to the next cube.

To find the surface intersection in a cube, we assign a one to a cube's vertex if a data value at the vertex exceeds (or equals) the value of the surface we are constructing. These vertices are inside (or on) the surface. Cube vertices with values below the surface receive a zero and are outside the surface. The surface intersects those cube edges where one vertex is outside the surface (one) and the other is inside the surface (zero). With this assumption, we determine the topology of a surface within the cube finding the location

of the intersection later. One disadvantage of MC is that making a binary decision about whether a surface intersects a cube or not exhibits false positives (spurious surfaces) or false negatives (erroneous holes in surfaces), particularly in the presence of small or poorly defined features.[L88]

The 2D case of the algorithm is called Marching Squares (MS). Marching Squares is easier to describe and is analogous to MC. The following is pseudo-code for MS applied to the ellipse in the figure below:

- 1 For every point in the grid
 - 1.1 Set the point's status to inside or outside the ellipse (from the formula for an ellipse)
- 2 For every square in the grid
 - 2.1 For every line segment that has a point outside and point inside the figure
 - 2.1.1 Add a point in the list t placed in the middle between the two points
- 3 Draw sequential lines between the points found in list t

Figure 4: Pseudo-code for Marching Squares applied to an ellipse. For 2.1.1, we could compute the intersection point using linear interpolation instead of taking the midpoint. Pseudo-code provided by [L03].

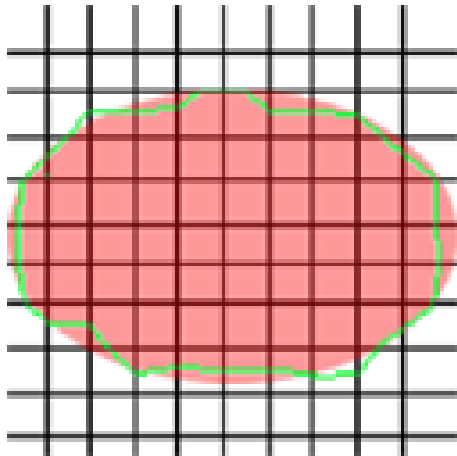


Figure 5: Rendering of an ellipse using Marching Squares taking the midpoint of each square edge to be the intersection. The orange ellipse is the shape that we wish to generate. The green line is the output from Marching Squares. Image provided by [L03].

Extending MS to MC, we note that there are eight vertices of a cube, and they can be in only two states (inside and outside). Thus, there are $2^8=256$ ways a surface can intersect with a cube. By enumerating these 256 cases, a code of inside and outside cases for each vertex of a cube can be mapped to edges of triangles used to determine the surface. In order to map from such a code to edges of triangles, we must also enumerate vertices and edges. One such enumeration follows:

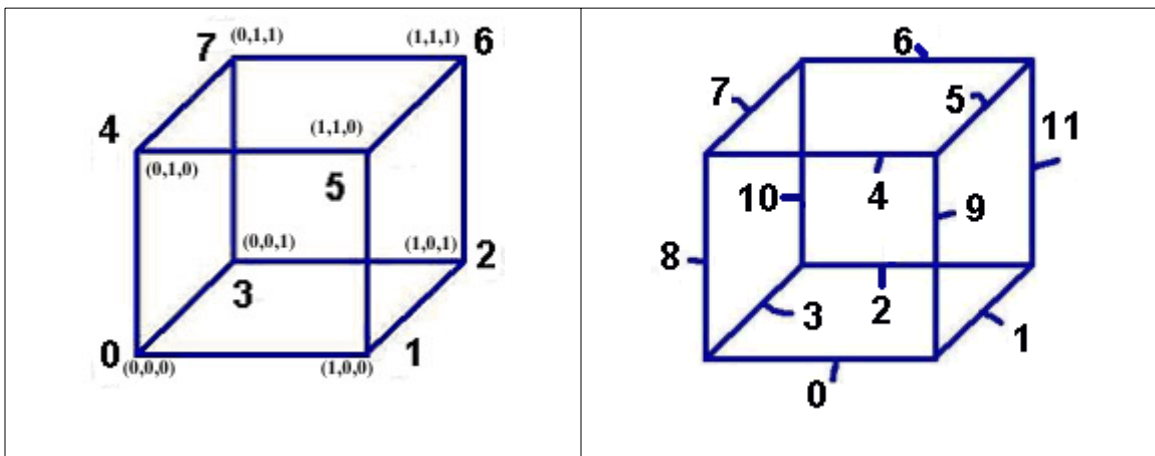


Figure 6: (Left) shows one possible vertex enumeration for a cube. (Right) shows one possible edge enumeration of a cube. Image provided by [L03].

In order to visualize MC, for the sake of explanation, we can assume that the intersection can be computed as a midpoint between cube edges, much like we did with MS. The following sequence of images shows the ideal behind MC.

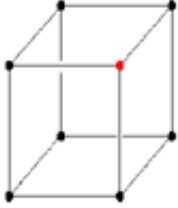
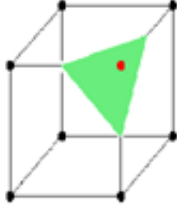
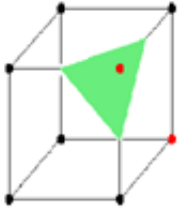
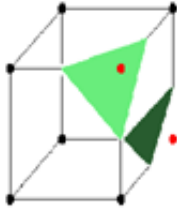
	<p>1) One point is outside of the the object we want to draw and the remaining seven are inside.</p>		<p>2) Generated a triangle that is formed between the centers on all three lines that now have one point on the inside and one on the outside of the model.</p>
	<p>3) Another point in the same cube is outside the model.</p>		<p>4) Generate another triangle, this time on the other side of the cube. This single cube has contributed with two triangles for the finished object. After processing all cubes, we get an object quite close to what we want.</p>

Figure 7: Marching Cubes applied to a single cube. For the sake of simplicity, we assume that the intersection of the surface and each cube edge is half-way along each edge. Image provided by [L03].

The following pseudo-code summarizes the Marching Cubes algorithm:

- 1 For every cube in the voxel buffer space
 - 1.1 Calculate an index by comparing the eight density values at the cube vertices with the surface constant
 - 1.2 Using the index, look up the list of edges from a precalculated table
 - 1.3 Using the densities at each edge vertex, find the surface edge intersection using linear interpolation
 - 1.4 Output the triangle vertices

Figure 8: Pseudo-code that summarizes the Marching Cubes algorithm. Pseudo-code provided by [L03].

3.1.4.2 Modeling Clouds

A closed cloud-shaped polygonal mesh can be built using Marching Cubes with meta-balls. With meta-balls, we determine if a point from a 3D grid is inside or outside of our surface using the following equation:

$$e = \frac{mass}{(distance)^2} \quad [J01]$$

The distance in the denominator represents the distance between a point mass and the grid point. We have a number of moving points with mass. The position and the mass of each ball is set using random numbers or a physics simulation. We sum e for all points with mass when considering a single grid point. Once we have e for a grid point, we test if the grid point is outside or inside of a surface by determining if e is greater than or less than a threshold called *Level* that we define. The different combinations of inside and outside states for a single voxel's points determine which triangles, if any, are rendered for the voxel. Each voxel point above the threshold causes a bit to be set in an integer. The integer is used to index into a predefined list of triangles defined as edges. We linearly interpolate between voxel points along an edge in order to find the actual vertex of a surface triangle. The interpolate t that we use for the interpolation is calculated by:

$$t = \frac{Level - e_1}{e_2 - e_1} \quad [J01]$$

where e_1 and e_2 are the sums of e for the voxel edge endpoints. All voxels that intersect the surface are evaluated. The vertices are used to set a vertex buffer, and the edge indices are used to set an index buffer.

3.2 Techniques Used for Rendering Clouds

3.2.1 *Introduction*

Once clouds have been modeled using any one of the techniques discussed in section 3.1, they can be rendered using one of the techniques found in this section. More advantages and disadvantages of each volume representation can be found here, but they are associated with efficiency and accuracy of techniques used to draw them. Choosing which cloud representation to use can be based solely on how each is rendered, because some rendering techniques are less efficient than others, and some are more accurate than others. In this section, we examine the differences between cloud rendering techniques in order to determine relative levels of accuracy and performance between different combinations of cloud modeling and rendering techniques.

One similarity between all of the rendering techniques discussed here, is that all of them use some solution to the volume rendering equation (**Equation 1.0**) explained in chapter 2. **Equation 1.0** is for approximating emission and absorption of light for a volume.[EHKRW06]

3.2.2 *Particle System*

The general principal of rendering a particle volume is as complicated as rendering traditional polygon mesh.[R83] Particles can obscure other particles that are behind them in screen depth.[R83] They can be transparent and cast shadow on other particles. In addition, particles can coexist in a scene with objects modeled by surface based primitives, and these objects can intersect with particles.[R83] In [R83], the assumption is made that each particle can be displayed as a point light source.[R83] Such an assumption means that the volume only approximates volume rendering using

emission. For volumes with emission and absorption such as clouds, we use the solution of the iterative form of the volume-rendering equation (**Equation 1.4** or **Equation 1.5**). By using equations 1.4 or 1.5, we must sort particles by screen depth which might be a time consuming operation with a large amount of particles. GPU texture blending can be used in order to render particle clouds. However, since what is rendered is a set of blended textured polygons, performance drops when the view comes close to the clouds and semi-transparent polygons fill the entire screen.[M01] In order to describe how clouds are rendered using a particle system, we first describe how the points computed in section 3.1 are used in order to define the geometry that is rendered. Later, we show how to extend particles systems in order to remove artifacts caused by using them in a scene with terrain.

3.2.2.1 *Billboarding*

Billboarding is the act of orienting a textured polygon based on the view direction. [MHH08] The texture used for each polygon, when representing clouds, looks like a small part of a cloud. The polygon itself is called a billboard or a particle, and it is usually a quadrilateral. As the view changes, the orientation of the polygon changes. In order to perform billboarding, we use three mutually orthogonal vectors. A point in the world defines the center of the billboard while the orientation is defined by the three vectors. The center points are the positions of each particle that we set with random numbers or a physics simulation.

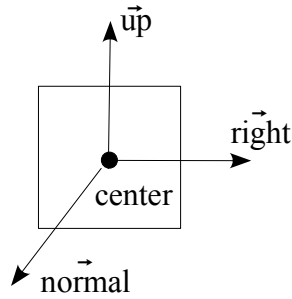


Figure 9: The normal, up, and right vectors used for billboarding along with the polygon and its center. Image derived from [MHH08].

3.2.2.1.1 Screen-Aligned Billboarding

For screen-aligned billboarding, the negative of the normal vector of the billboard is always in the same direction of the view of the camera.[MHH08] In order to draw the polygon, we must calculate the positions of the vertices of the polygon in world space. One way to calculate the positions is to displace from the center position in the direction of the up and right vectors. In order to get the scale right, we must have a variable that controls the size of the billboard as well. The up vector is simply the up vector from the camera view matrix. The camera up vector is a vector that defines the up direction for the camera. The right vector is simply the right vector from the camera view matrix. The camera right vector is a vector that defines the right direction for the camera. If the up and right directions are of unit length, the four vertices that we want are:

$$\begin{aligned}
 v0 &= center + \vec{up} + \vec{right} & v1 &= center - \vec{up} + \vec{right} \\
 v2 &= center - \vec{up} - \vec{right} & v3 &= center + \vec{up} - \vec{right}
 \end{aligned}$$

3.2.2.2 *Rendering Clouds Using a Particle System*

In order to draw clouds, we use textured screen-aligned billboards positioned randomly, or positioned using a physics simulation. We use either equation 1.4 or 1.5, so that the clouds are drawn with emission and absorption. In order to use equations 1.4 and 1.5, the billboards are sorted by screen depth. The next section discusses a technique that can be used in order to accelerate particle rendering.

3.2.2.2.1 Shader Constant Instancing

Shader constant instancing is a way of rendering billboards or particles with the intent to render them quickly using the GPU. One advantage of shader constant instancing is that we do not have to read and write to a vertex buffer when we want to draw particles. We set a small vertex buffer once with local space positions of a quadrilateral. World positions are calculated in the vertex shader through the use of an array of vectors that store the center position and size of each quadrilateral. For example, the vertex buffer and the index buffer may hold 64 quadrilaterals. The array of positions passed to the vertex shader holds 64 positions too. In addition, the colors for all particles are passed as an array of 64 vectors. For each draw call, 64 particles are drawn. Billboarding is performed in the vertex shader.[C05]

3.2.2.3 *Spherical Billboards*

Spherical Billboards (SB) is a particle system technique used to solve billboard clipping artifacts by calculating the real path length a light ray travels inside a given particle, because the length determines the opacity value to be used during rendering. Like many particle systems, quadrilaterals primitives are sent through the graphics

pipeline. However, each particle is assigned a radius that is used during rendering to give it the appearance of a sphere volume when it comes into contact with opaque objects in the scene.[USS06]

To find out where opaque objects are during particle rendering, the opaque objects are drawn first, and the resulting depth buffer storing camera space z coordinates is saved in a texture. Next, the particles are sorted so that they can be rendered in the order of furthest from the camera to closest to the camera. Particles are rendered as quadrilaterals perpendicular to the viewing axis of the camera coordinate system. The particles are rendered with depth testing disabled to eliminate incorrect object-billboard clipping. For each pixel of a rendered particle, the interval that the ray travels inside of the particle sphere is computed using the saved depth values of opaque objects along with the particle position in camera space, the shaded billboard point in camera space, the particle radius, the screen coordinates of the shaded point, and the camera's front clipping plane distance. [USS06]

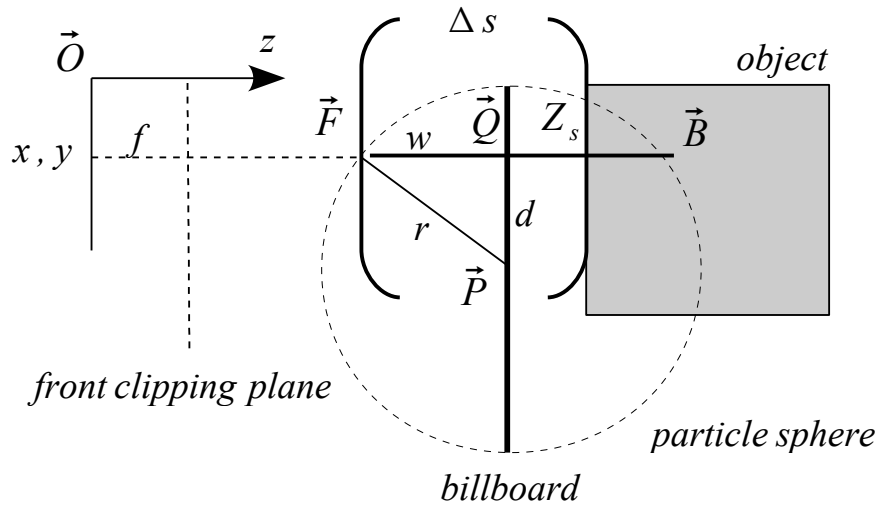


Figure 10: Computation of length Δs the ray segment travels inside a particle sphere of radius r and of center \vec{P} . Image derived from [USS06].

Let $\vec{P} = (x_p, y_p, z_p)$ be the center of the particle in camera space,

$\vec{Q} = (x_q, y_q, z_q)$ be the pixel of the billboard being rendered, and r be the radius of the sphere. Assuming an orthographic projection, that is $z_q = z_p$, the distance between the ray and the particle is

$$[USS06] \quad d = \sqrt{(x_q - x_p)^2 + (y_q - y_p)^2}$$

The closest and the farthest points of the particle sphere on the ray from the camera are \vec{F} and \vec{B} , respectively. The distances of these points from the camera can be obtained as

$$[USS06] \quad |\vec{F}| \approx z_p - w, |\vec{B}| \approx z_p + w$$

Where,
$$w = \sqrt{r^2 - d^2}$$

Thus,
$$\Delta s = \min(Z_s, |\vec{B}|) - \max(f, |\vec{F}|) \quad [USS06]$$

Assuming the density is homogeneous inside particle j , we can calculate the opacity for each pixel which equals the decrease in radiance caused by extinction (i.e. the sum of absorption and out scattering). Using Beer's Law we have

$$[USS06] \quad \alpha_j = 1 - e^{-\tau_j \Delta s_j}$$

where τ is the density. However, in order to avoid artifacts, we modify the opacity function to the following:

$$[USS06] \quad \alpha_j \approx 1 - e^{-\tau_j (1 - d/r_j) \Delta s_j}$$

Additive blending is used to blend multiple particles into the scene buffer.

3.2.3 *Volume Ray Tracing*

Volume Ray Tracing (VRT) is a technique that can be used in order to draw clouds from a volume texture. Although this method has high intrinsic computational cost, when rendering a sufficiently large data-set, ray tracing should be competitive because its low time complexity ultimately overcomes its large time constant.[PPLSHS99] This crossover will happen sooner on a multiple CPU computer because of ray tracing's high degree of intrinsic parallelism.[PPLSHS99] In addition, ray casting algorithms enjoy speedup advantages from several different optimization techniques.[DKCBK98] The first is called space leaping, or skipping over areas of insignificant opacity, and the second is called early ray termination.[DKCBK98] In the next section, we explain how the volume texture clouds built from noise or a physics simulation are rendered using VRT.

3.2.3.1 *Rendering Clouds Using Volume Ray Tracing*

Rendering clouds using VRT is different from rendering clouds using a particle system in that no sorting of primitives is required. Instead, we trace a ray through a volume to compute the color for that pixel.[PPLSHS99] Samples of the the volume are taken along the ray at equally spaced intervals.[L90] As each ray is marched through the volume, scalar values are mapped to optical properties through the use of a transfer function which results in a RGBA value that includes the corresponding emission and absorption for the current sample.[H09] For a short description of transfer functions, see the section below. Samples (color and opacity) are composited from front-to-back.[L90] The front-to-back compositing equation is **Equation 1.5**.

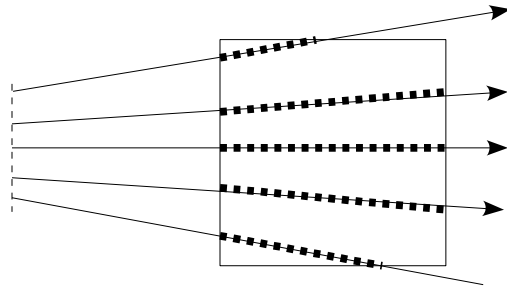


Figure 11: The volume is sampled at equally spaced intervals along rays from the eye.

3.2.3.1.1 Transfer Function

A transfer function is used to “view” a certain part of the volume. For example, there may be a skin layer and a bone layer in a volume. A transfer function could be designed to just look at the skin, just the bone, or both. Typically, values in a 3D volume texture are in the range from 0 to 255. One transfer function may consider values in the range from 40 to 60 to be skin and color them tan, and consider values in the range from 80 to 250 to be bone and color them white.[H09]

3.2.4 Fog Polygon Volumes

A technique that can be used in order to draw mesh clouds is called Fog Polygon Volumes (FPVs). FPVs is a convenient and flexible technique for rendering ordinary polygon objects of any shape as thick volumes of light-scattering or light-absorbing material.[04] The motivation for developing the algorithm was to render a patchy fog in real-time first-person simulators in which the viewer can move through the fog.[M01] The result is a true volumetric rendering of ordinary polygon objects.[04] A large cloud is more efficiently rendered using FPVs than a particle system, because a particle system is a method that blends slices, and the cloud can cover a large area of the scene making the

number of slices too large.[M01] In addition, unlike a particle system, the performance does not drop when the view comes close to the object.[M01] This technique has the advantage of not being time consuming like ray tracing is.[M01] However, the results often do not look as realistic as those produced by traditional techniques, mainly due to a low polygon count of gas boundaries and visible discontinuities in the derivative of the fog intensity along sharp boundary edges.[M01] In the next section, we describe how to render the mesh clouds built in section 3.1 using FPVs.

3.2.4.1 Rendering Clouds Using Fog Polygon Volumes

Before rendering clouds using FPVs, we assume that the gas inside of the polygon boundary is of constant density.[M01] For each pixel, we can determine the distance a ray associated with this pixel traveled through the gas by subtracting the distance between the front facing and back facing triangles from a view point.[M01] The resulting per-pixel distance is converted to the attenuation factor and used to blend the scene color with a uniform fog color.[M01] FPVs use vertex and pixel shaders in multi-pass rendering to generate a measure of object thickness at each pixel.[04] In order to render the volumes in a scene with opaque objects, we start by rendering the scene of opaque objects storing color in an ordinary back buffer, and view space depth in another texture.[04] Next, we render the depths of all of the back faces of the volumes into a texture with additive blending, and we render the depths of all of the front faces of the volumes into another texture with additive blending again.[04] The two textures with cumulative depths for back and front facing triangles and the texture with view space depths for opaque objects in the scene are used to calculate object thickness for each pixel.[04] The thickness is used to calculate absorption or scattering in approximations where the total amount of light for a pixel is a function of only thickness.[04] One approximation for absorption can be done using Beer's Law assuming the density τ is homogeneous inside the volume. [MHH08][USS06] Therefore, given the thickness Δs_j for volume j , the opacity, which

equals the decrease in radiance caused by extinction (i.e. the sum of absorption and out scattering), is

$$[\text{USS06}] \quad \alpha_j = 1 - e^{-\tau_j \Delta s_j}$$

Another way to think about how the thickness can be used to compute radiance is to use **Equation 1.2** with the interval between s_0 and D being the thickness computed using FPVs. While everything else in the equation remains constant.

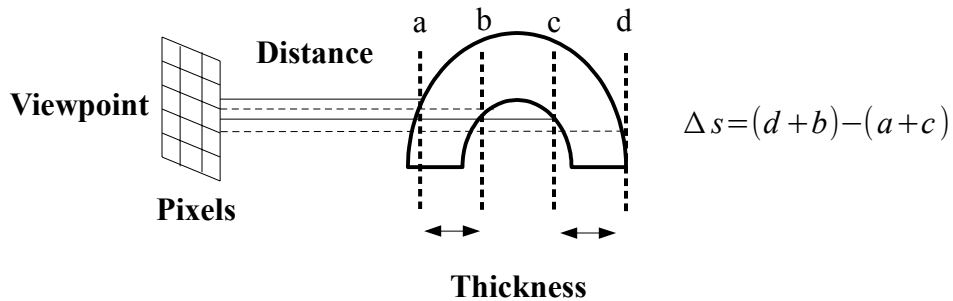


Figure 12: Computing thickness. Front face depths are a and c , and back face depths are b and d . Image derived from [04].

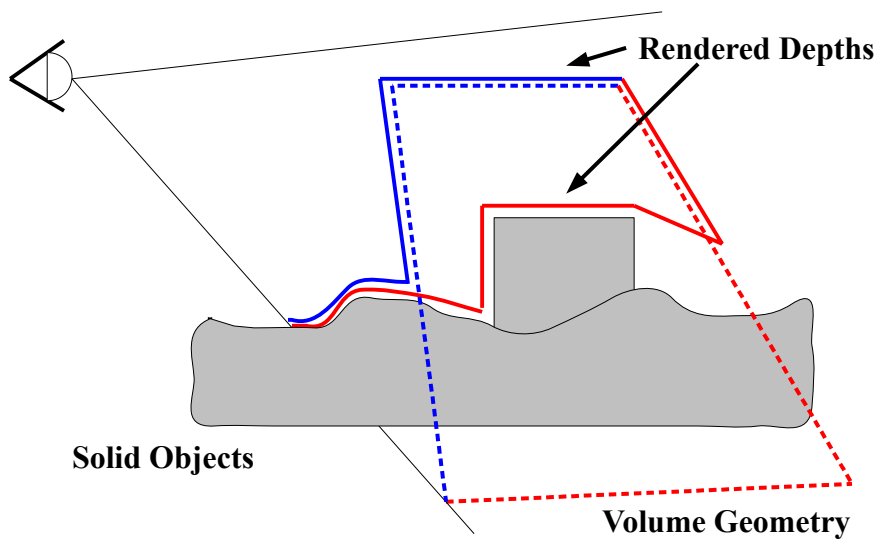


Figure 13: Handling opaque objects intersecting the volume objects. The volume geometry is shown with a dotted line. A pixel shader compares the volume object depth to the solid object depth read from a texture and outputs the lesser depth value. This results

in depth information being taken from the geometry shown with solid lines. The depth comparison clamps the occluded volume object pixels to the nearest solid object depth, effectively limiting the volume object thickness to the proper amount for all intersection cases. Image derived from [04].

3.3 Techniques Used for Rendering Cloud Shadow

3.3.1 *Introduction*

Clouds rendered without shadows insufficiently capture the look of real clouds. We haven't finished determining the levels of accuracy and performance of cloud rendering techniques until we examine them in combination with cloud shadowing techniques. For our research to be thorough, we analyze techniques that self-shadow clouds, and techniques that cast shadows on nearby terrain, for any given light position. Some techniques used to render a shadow for clouds are more appropriately applied to a specific cloud volume rendering technique. Picking a technique used for rendering the shadow of clouds can be based on what technique is used for rendering the clouds themselves. The opposite can be true too. In other words, the decision about which technique to use for rendering clouds, can be based on what technique is chosen for rendering the shadow of the clouds. Moreover, each shadow rendering technique has its own strengths and weaknesses, and shadow rendering is so important for a good final image.

Our preference is for cloud shadowing techniques that are fast, accurate, and memory efficient, but we examine the strengths and weakness of each technique in order to build a better picture about which technique is appropriate for each situation. Cloud shadows that are volumetric are preferred to ones that are not. Surface shadows are shadows that are not volumetric. For more information about volumetric shadows and surface shadows, see the sections below.

3.3.2 Surface Shadows

The shadow technique discussed in this section is used to compute shadowing for surfaces. It is not volumetric, so it will not produce accurate shadows for clouds. It builds a texture map from a single depth value computed from a light. Traditional Depth Shadow Maps (TDSMs) suffer from aliasing issues.[W78][DL06] We describe TDSMs in the section that follows.

3.3.2.1 Traditional Depth Shadow Map

A Traditional Depth Shadow Map [W78] (TDSM) is built by rendering the depth of all possible light occluders from the view of the light. When rendering a surface that may be in shadow, every sample from the view of the eye is used to calculate the depth of the sample from the light, and the depth is compared with the corresponding sample in the shadow map. If the depth in the shadow map is less than the depth computed for the sample from the eye, the sample is in shadow, so it is colored darker.

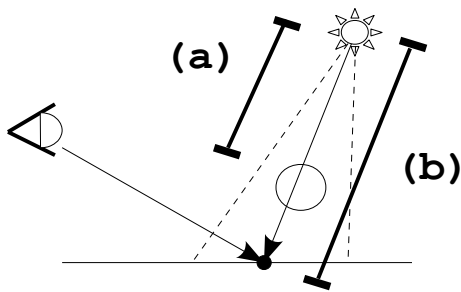


Figure 14: The distance (a) from the light is less than the distance (b) from the light, so the point is in shadow.

3.3.3 *Volumetric Shadows*

All of the shadow mapping techniques in this section are volumetric shadow mapping techniques that can be used to render the shadow of clouds. A volumetric shadow is defined as the amount of light reaching a point in a volume along a light ray traced between the volume point and the light source.[JB09] We examine five techniques that can be used for rendering a volumetric shadow for clouds. The five techniques are Deep Shadow Maps (DSMs)[LV00], Opacity Shadow Maps (OSMs)[KN01], Half-Angle Slicing (HAS)[KPHSM03], Deep Opacity Maps (DOMs)[YK08], and Fourier Opacity Maps (FOMs)[JB09]. The next few sections are devoted to describing DSMs, OSMs, HAS, DOMs, and FOMs, analyzing the progression or origins of each, and comparing them to each other. Levels of accuracy and performance of techniques used to shadow clouds are described in this section. Some techniques used to shadow clouds may appear more desirable than the others, because of distinctions between the techniques. When combined with the accuracy and performance of modeling techniques described in section 3.1, and cloud rendering techniques described in section 3.2, this section helps show levels of accuracy and performance of techniques used to render clouds with shadows.

DSMs were the first to be introduced. DSMs improve upon Traditional Depth Shadow Maps, by requiring lower resolution textures when used to capture fine details of hair, fur, and smoke.[LV00] In addition, DSMs can handle volumetric effects like clouds, and Traditional Depth Shadow Maps cannot.[LV00] DSMs are a fitting solution for offline applications.[JB09] In other words, it requires a significant amount of data initialization time.[KN01] DSMs are compact and of high quality, but when the volume changes in time with respect to the light, the generation cost can cancel out the computational benefit of the algorithm.[KN01] An unbounded amount of memory is required for capturing the primitives.[JB09] Bounded memory DSM techniques exist, but they they introduce artifacts.[HKSB06]

OSMs is an approximation to DSMs.[JB09] Unlike a DSM, an OSM can be

generated efficiently on the GPU using a bounded amount of memory[KN01][JB09] An OSM produces layering artifacts.[JB09] However, the layering artifacts can be softened and reduced at the cost of some efficiency.[JB09] In regard to rendering particle clouds with shadows, an OSM requires all of the particles to be sorted twice.

HAS is another approach to rendering volumetric shadows. HAS uses less memory than DSMs.[JB09] HAS has a bounded memory requirement, unlike DSMs. HAS can be expensive, because it requires multiple geometry passes and render-target switches.[JB09] In regard to rendering particle clouds with shadows, HAS requires all of the particles to be sorted only once, which is an improvement upon OSMs.

The introduction of DOMs focused on rendering hair, but the method is applicable for rendering other semi-transparent objects.[YK08] DOMs combine traditional depth shadow mapping [W78] and OSMs [KN01] to give a better distribution of opacity layers. [YK08] Layering artifacts that are apparent in OSMs, unless a very high number of slices are used, can be avoided with a DOMs.[YK08] Moreover, far fewer layers are necessary to generate high quality shadows.[YK08]

FOMs are another approach to rendering a volumetric shadow. FOMs improve upon slice-based methods like OSMs, by avoiding layering artifacts.[JB09] Instead, FOMs have a loss of high-frequency detail, and have some amount of ringing which is a different type of artifact.[JB09] However, FOMs cope well with outliers (in this context, an outlier is a primitive that is apart from the main group).[JB09] FOMs cannot be used with opaque volumes.[JB09] In regard to rendering particle clouds with shadows, FOMs require all of the particles to be sorted only once, which is the same as HAS, and it is an improvement upon OSMs.

3.3.3.1 Deep Shadow Map

A DSM [LV00] is a rectangular array of pixels in which a visibility function is

stored for every pixel. Intuitively, a visibility function is defined by considering a beam of light that moves from the light into a scene. The function value at a given depth is simply the fraction of the beam's initial power that penetrates to that depth. The power of light is attenuated as it passes through material giving lower visibility at greater depth. Two functions are combined in order to create the final transmittance function for a single pixel. The two functions are called surface transmittance and volume transmittance. [LV00]

The surface transmittance τ^s function is a combination of opaque object coverage and semitransparent object coverage. Opaque object coverage is determined by the amount of opaque object surface considered for a single pixel. The image below on the left represents opaque object coverage and a corresponding visibility function. Semitransparent object coverage is determined by the amount of opacity and thickness of semitransparent objects considered for a single pixel sample. Starting with a transparency of 1, we multiple a value of $1 - \alpha$ for each surface hit. Each surface hit lowers surface transmittance for a pixel for a single depth. The image below on the right represents a stack of semitransparent objects and a corresponding visibility function. [LV00]

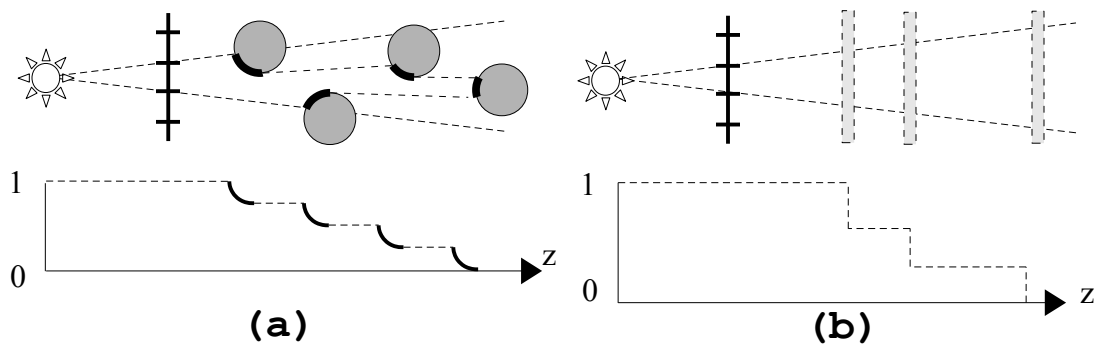


Figure 15: Visibility functions. Each diagram shows a beam of light that starts at the shadow camera origin (i.e the light source) and passes through a single pixel of the deep shadow map, accompanied by the pixel's visibility function. **(a)** The blockers are opaque, but each covers only part of the pixel's area; the emphasized segments of the function correspond to visible portions of the blockers. **(b)** The beam's power is reduced as it passes through consecutive semitransparent surfaces. Image derived from [LV00].

The volume transmittance τ^v function is generated using an atmospheric density field. An atmospheric density field can be generated using noise, like Simplex Noise, or a physics simulation can be used, and the density field can be stored in a voxel buffer or a 3D texture. The atmospheric density field is used to generate extinction values which are a measure of light attenuation for points inside of the medium. The extinction values themselves are computed by sampling and accumulating atmospheric density from a point in the medium to the light. The extinction function is denoted by κ . Other names for extinction are absorption and attenuation. An image of volume attenuation due to a cloud, and a corresponding visibility function appears below. With a measure of extinction for any point in the medium, we can use an approximation of Beer's Law to compute the final value transmittance for a linear segment as

[LV00]
$$\tau^v(z) = e^{-(z_{i+1}^v - z_i^v)(k_{i+1} - k_i)/2}$$

where z_i^v is the depth, and k_i is extinction for sample i along a viewing ray.

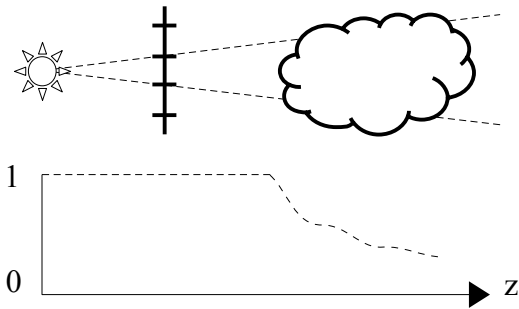


Figure 16: Visibility function for volume attenuation due to a cloud. The diagram shows a beam of light that starts at the shadow camera origin (i.e the light source) and passes through a single pixel of the deep shadow map, accompanied by the pixel's visibility function. Image derived from [LV00].

The final transmittance is the product of the surface transmittance and the volume transmittance. It is a piece-wise linear function of many segments, so we compress it. The final visibility for a single pixel is a weighted combination of the transmittance function

τ at nearby sample points.[LV00]

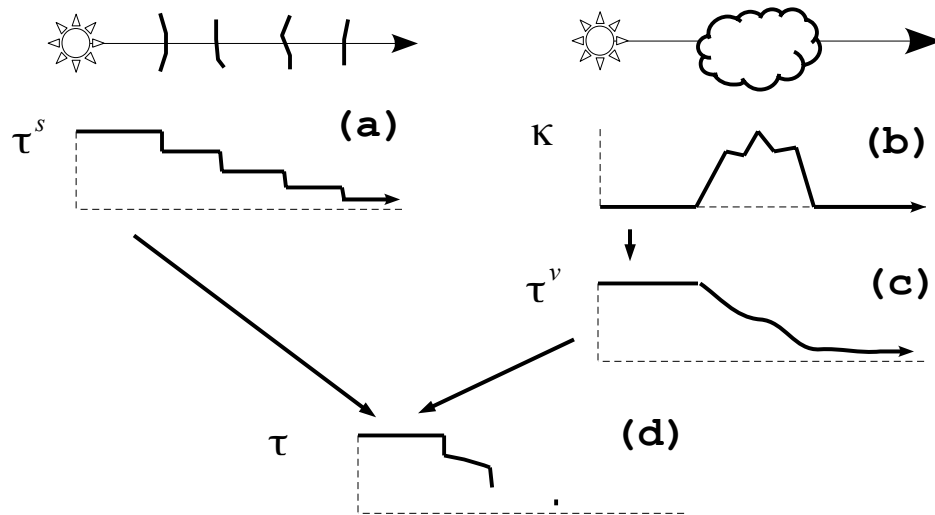


Figure 17: Constructing a transmission function. **(a)** The object intersections along a given ray yield the surface transmittance function τ^s , which has a discontinuity at the depth of each surface. **(b)** The extinction function κ is obtained by sampling the atmospheric density at regular intervals along the ray. **(c)** The extinction function is integrated and exponentiated to yield the volume transmittance τ^v . The surface transmittance and volume transmittance are multiplied to obtain the final transmittance function τ for each ray. Image derived from [LV00].

3.3.3.2 Opacity Shadow Map

Opacity shadow maps (OSMs) use a set of parallel opacity maps oriented perpendicular to a light's direction.[KN01] The opacity maps are embedded in the scene. On each opacity map, the scene is rendered from the light's point of view, clipped by map's depth.[KN01] The rendering order of the primitives is from the nearest to the light to the furthest from the light which requires sorting. Rendered primitives can be points, lines, and polygons. Each primitive contributes its associated alpha valued for a single pixel. Each pixel in the map stores an alpha value that approximates the opacity relative to the light at the pixel's positions. The opacity values from adjacent maps are sampled

and linearly interpolated at the position of each shadow computation. The interpolated opacity Ω is used to calculate transmittance τ using Beer's Law

$$[\text{KN01}] \quad \tau = e^{-\Omega}$$

The final shadow Φ is computed from the transmittance as

$$[\text{KN01}] \quad \Phi = 1 - \tau$$

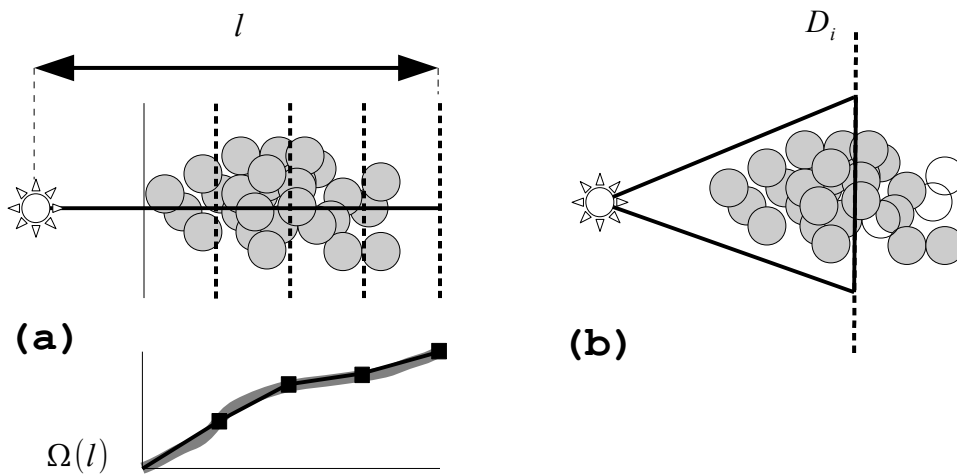


Figure 18: (a) The opacity function $\Omega(l)$ shown in solid gray curve is approximated by a set of opacity maps. (b) The volume is rendered on each opacity map, clipped by the map's depth D_i . The region outside of the triangle is clipped. Image derived from [KN01].

3.3.3.3 Half-Angle Slicing

Half-Angle Slicing (HAS) is a technique that is used in order to render volumetric shadows for volume rendering without needing to sort primitives twice like we would if we were using an opacity shadow map. In addition, with HAS we use less texture memory than we would if we were using an opacity shadow map.[G08]

As light from a source moves through a semi-transparent volume, the material of the volume attenuates the light, which produces volumetric shadows. With HAS, the amount of light attenuated from the light's point of view is accumulated using a pixel buffer. Slices of a volume can be thought of as billboards which are commonly aligned perpendicular to the view direction. However, we render the volume as a series of slices perpendicular to a half-angle vector \vec{s} in order to increase efficiency. The half-angle vector is a vector halfway between the view and the light direction. We slice along the half-angle vector, so that the sorting of the slices is roughly the same for rendering from both the eye and the light point of view. Therefore, we can accumulate the shadowing from the light at the same time as we are blending the slices to form the final image. [FIKLH04]

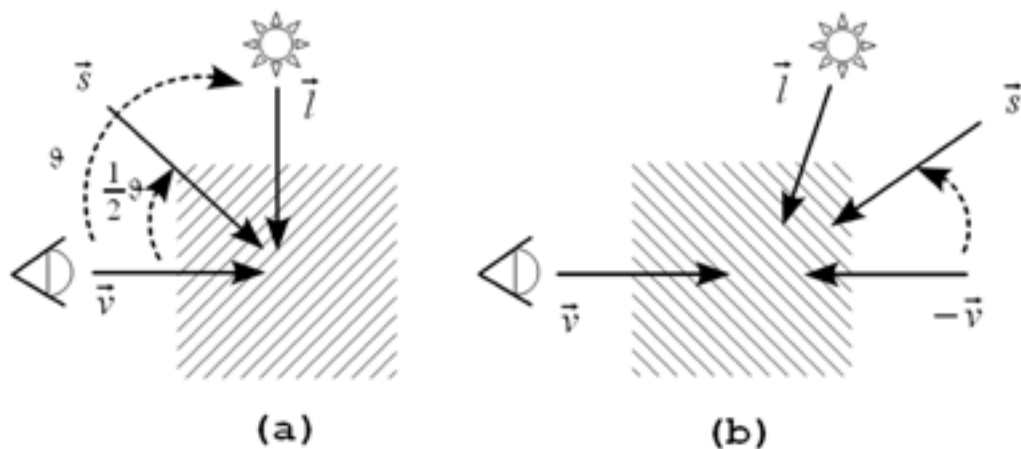


Figure 19: (a) The dot product of the light and view directions is positive. The slice is set halfway between the light and view directions. The volume is rendered front-to-back for the eye using the Under operator (Equation 1.4). (b) The dot product of the light and view directions is negative. Slice along the vector halfway between the light and the inverted view directions. Render the volume back-to-front for the eye using the Over operator (Equation 1.5). Image derived from [FIKLH04].

The direction of rendering depends on the orientation of the light relative to the viewer. We want to always render from front-to-back from the lights point of view, so that we can accumulate the shadow correctly.[FIKLH04]

When the viewer is pointing in approximately the same direction as the light (a), we render the slices from front-to-back from the eye's point of view. When the camera is pointing toward the light (b), we negate the view vector, and render the slices from back-to-front from the eye's point of view.[FIKLN04]

The amount of light arriving at a particular slice is equal to one minus the accumulated opacity of the previously rendered slices. Each slice is first rendered from the eye's point of view, using the results of the previous pass rendered from the light's point of view, which are used to modulate the brightness of samples in the current slice. The same slice is then rendered from the light's point of view to calculate the intensity of light arriving at the next slice.[FIKLN04]

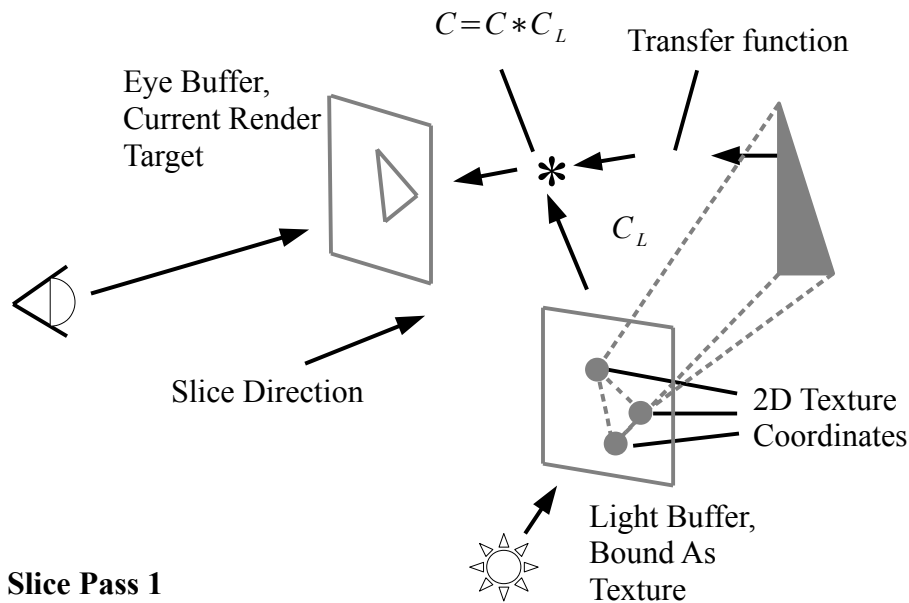


Figure 20: Two-pass volumetric shadow **Slice Pass 1.**

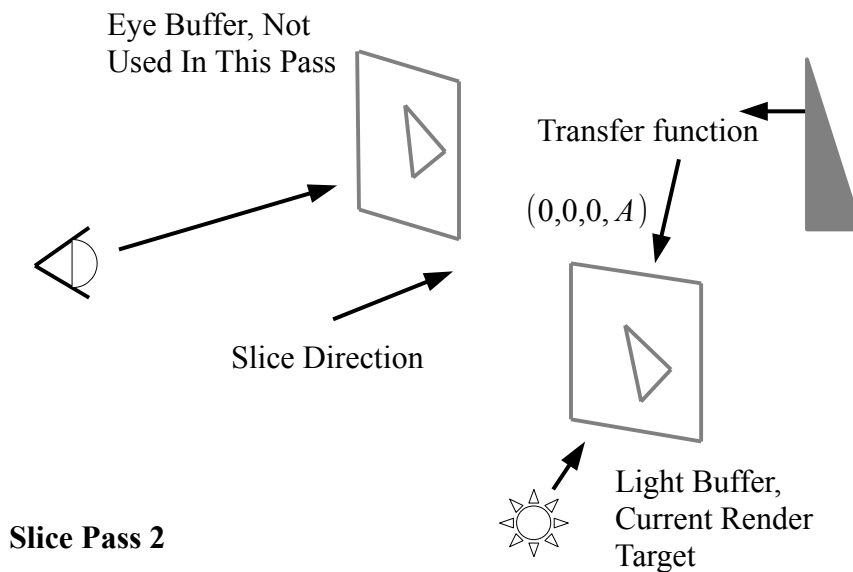


Figure 21: Two-pass volumetric shadow **Slice Pass 2**. Image derived from [FIKLH04].

3.3.3.4 Deep Opacity Map

A Deep Opacity Map (DOM) is an extension of the concept of OSMs.[YK08] It combines TDSMs and OSMs to give a better distribution of opacity layers.[YK08] First, the geometry is rendered from the view of the light, recording the depth values in a shadow map.[YK08] Next, an opacity map similar to an OSM from the view of the light is generated.[YK08] The novelty of the algorithm lies in the way the opacity maps are distributed.[YK08] Instead of using regular slices of the geometry in between two planes normal to the light direction, the depth information in the shadow map is used to create layers that vary in depth from the light source on a pixel-by-pixel basis.[YK08] When creating the DOM, the shadow map gives us the depth at which the geometry starts. Starting from the depth at which the geometry starts, the geometry is divided into a small number of layers by offsetting from the starting depth. The spacing does not have to be

uniform. The final shape of the separators between layers is not planar but it is similar to the shape of the geometry.[YK08]

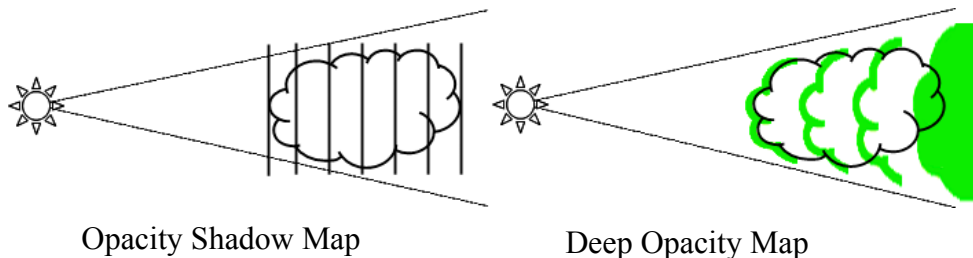


Figure 22: Opacity shadow maps use regular spaced planar layers. Deep opacity maps use fewer layers, conforming to the shape of the object.

The advantage of using a DOM over an OSM is that by shaping the opacity layers, visual layering artifacts are eliminated and interpolation between layers occurs within the volume, thus hiding possible inaccuracies.[YK08] This allows high quality results with far fewer layers.[YK08] One disadvantage of using a small number of layers is that it can be more difficult to ensure all points in the volume are assigned to a layer. [YK08] Points can lie beyond the last layer. Points beyond the end of the last layer do not correspond to any layer.[YK08] When implementing DOM, one has a choice to ignore such points, include them in the last layer, or ensure that last layer lies beyond the volume.[YK08] Ignoring such points means that they will not cast shadows.[YK08] Including them in the last layer means that they cast shadows on themselves.[YK08] Ensuring that the last layer lies beyond the volume can be done by either increasing the layer sizes or the number of layers.[YK08] While the last option might seem “ideal,” it can lead to unnecessary computational cost, since the light intensity beyond a certain point in the volume is expected to vanish.[YK08] For reasonable results, [YK08] recommends mapping the points to the last layer.

3.3.3.5 Fourier Opacity Map

A Fourier Opacity Map (FOM) reformulates and approximates variable absorption $\sigma(z)$ of a ray of light traveling through a translucent medium using a Fourier series for each pixel in light space. Given absorption, we use Beer's Law in order to compute transmittance $T(d)$. The generalized form of transmittance that is approximated is

$$T(d) = e^{-\int_{z=0}^{z=d} \sigma(z) dz} \quad [\text{JB09}]$$

The Fourier series approximation that we use to reconstruct the transmittance function is

$$\begin{aligned} T(d) = e^{-\int_{z=0}^{z=d} \sigma(z) dz} &\approx \frac{a'_0}{2} d + \sum_{k=1}^{k=n} \frac{a'_k}{2\pi k} \sin(2\pi k d) \\ &+ \sum_{k=1}^{k=n} \frac{b'_k}{2\pi k} (1 - \cos(2\pi k d)) \end{aligned} \quad [\text{JB09}]$$

where the Fourier coefficients a'_k and b'_k are

$$\begin{aligned} a'_k &= -2 \sum_i \ln(1 - \alpha_i) \cos(2\pi k d_i) \\ b'_k &= -2 \sum_i \ln(1 - \alpha_i) \sin(2\pi k d_i) \end{aligned} \quad [\text{JB09}]$$

For a given light ray i , the Fourier coefficients can be computed exactly in a single rasterization pass by additive blending without sorting the primitives. Each primitive contributes to the accumulation of Fourier coefficients for a single pixel. We store the first n coefficients in a FOM. The coefficients are used to reconstruct the transmittance function which is used during rendering to apply shadowing. [JB09]

Using a FOM instead of an opacity shadow map has the advantage that we only need to sort the particles once. In addition, opacity shadow maps have slice artifacts that a FOM will not have. However, a FOM may have an artifact known as ringing if not enough coefficients are used. [JB09]

Chapter 4: Results

We used several of the techniques from chapter 3 in order to render clouds of smoke with shadows using vertex and pixel shaders. The techniques are OSMs from section 3.3.3.2, DOMs from section 3.3.3.4, and FOMs from section 3.3.3.5. We render smoke using particle systems with three different volumetric shadow techniques in order to analyze actual levels of accuracy and performance for drawing clouds of smoke with shadows.

Accuracy and performance is measured for a fixed amount of memory, so that a comparison can be made between image and speed results for differing techniques. In terms of performance, we desire fast rendering speed and low memory usage, and the results can be directly compared, because actual numeric quantities are reported. In terms of accuracy, we use a large fixed amount of memory for slices or coefficients to establish a highly accurate baseline image that is nearly the same for all shadow techniques. Using smaller fixed amounts of memory reduces accuracy and increases artifacts. The difference between a baseline image and one produced using a lower fixed amount of memory shows inaccuracies that can be compared to other techniques.

First, the platform that was used in order to perform all of the speed measurements is described. The results and analysis of the techniques used to render clouds of smoke with shadows are found in section 4.2. Images produced by each implementation can be found in section 4.2.2.1.

4.1 Platform

Processor: Intel(R) Core(TM) i3-2100 CPU @ 3.10 GHz (2 CPUs), 3.10 GHz

Installed memory (RAM): 6.00 GB

DirectX version: DirectX 11

Card name: NVIDIA GeForce GTX 550 Ti

Display memory: 3787 MB (Dedicated Memory: 992 MB)

4.2 Rendering Clouds of Smoke with Shadows

Performance is perhaps the most important consideration when choosing a technique to render clouds of smoke. Smoke and shadow rendering performance is largely influential on total performance, and it is what is analyzed in this section. For each smoke rendering technique with shadows, we use the name of the shadow technique to identify it, because each shadow technique is distinct. The attributes that make a smoke rendering technique with shadows more desirable than another is fast rendering speed, low memory usage, and high accuracy or realism.

4.2.1 Speed Results

A number of time measurements were taken for each shadow technique using varying fixed amounts of memory. The number of render target texture components (r,g,b,a) used for each test was 16, 8, and 6. Note that DOMs use one render target texture component for depths from the light, so there is one less slice than OSMs for each test. The final speed that we compare is the sum of the time in milliseconds that it took to build the shadow information, and the time that it took in milliseconds to render the

smoke with shadows. Time measurements are averaged over 100 samples. Particle sorting time is included too.

In regard to Fourier Opacity Mapping, we varied the number of coefficients used to reconstruct the transmittance function. Increasing the number of coefficients increases accuracy by decreasing ringing. In regard to Opacity Shadow Maps and Deep Opacity Maps, we varied the number of slices used to compute the shadow map. Increasing slices increases accuracy. Adding slices to OSMs and DOMs, or coefficients to FOMs improves the ability of each technique to draw thin features of smoke more correctly. The resolution of the screen remained at a constant 1024x768 for all of the tests. The shadow map resolution remained at 512x512 as well. The number of particles that were used in all tests was fixed to 2500, and the position of the camera and light relative to the smoke remained constant. In addition, the same smoke volume was used for all measurements. The results are listed in the table below.

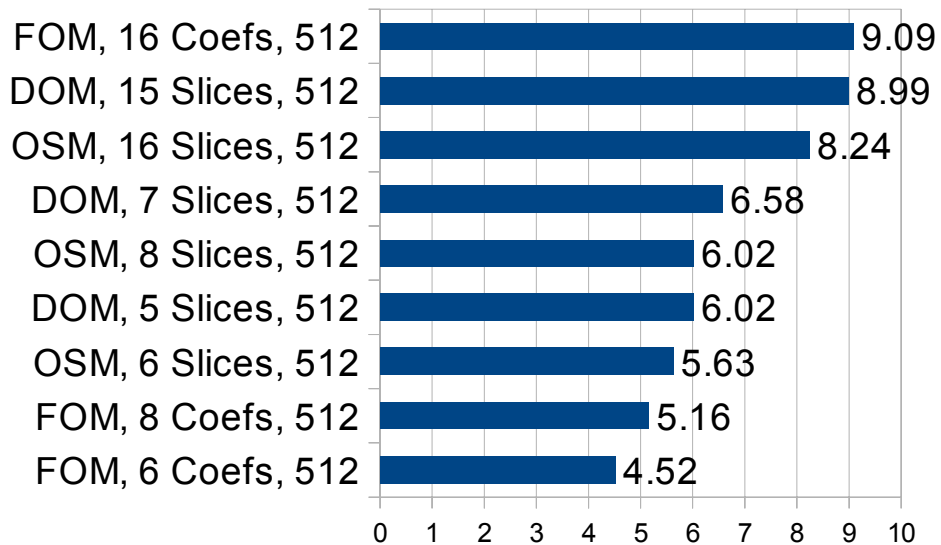


Figure 25: Millisecond timings for all 3 implementations sorted from fastest to slowest. The time that it took to build the shadow information plus the time that it took to render the smoke with shadows is the number shown. Particle sorting time is included, and time measurements were averaged over 100 samples. The IDirect3DQuery9 interface was used

to perform all time measurements. Each sort of all of the particles took 0.96 milliseconds.

4.2.2 Accuracy Results

We show a few of the resulting images of clouds of smoke rendered with our implementations in the section below. Section 4.2.2.2 shows more detailed accuracy results. All of the images shown are generated using a single spotlight.

4.2.2.1 Images

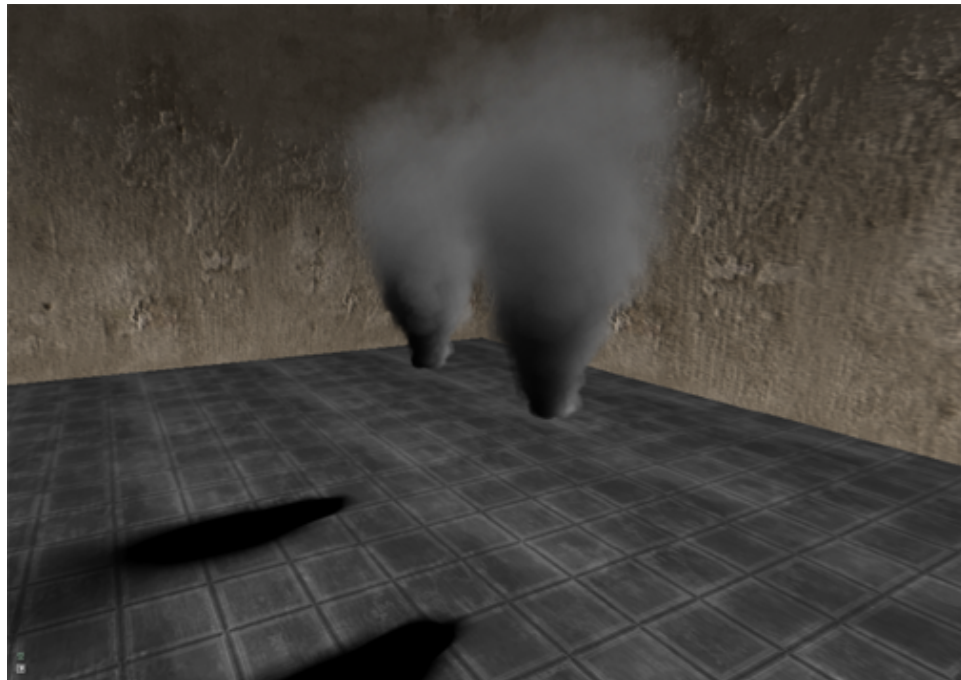


Image 1: Smoke with an Opacity Shadow Map

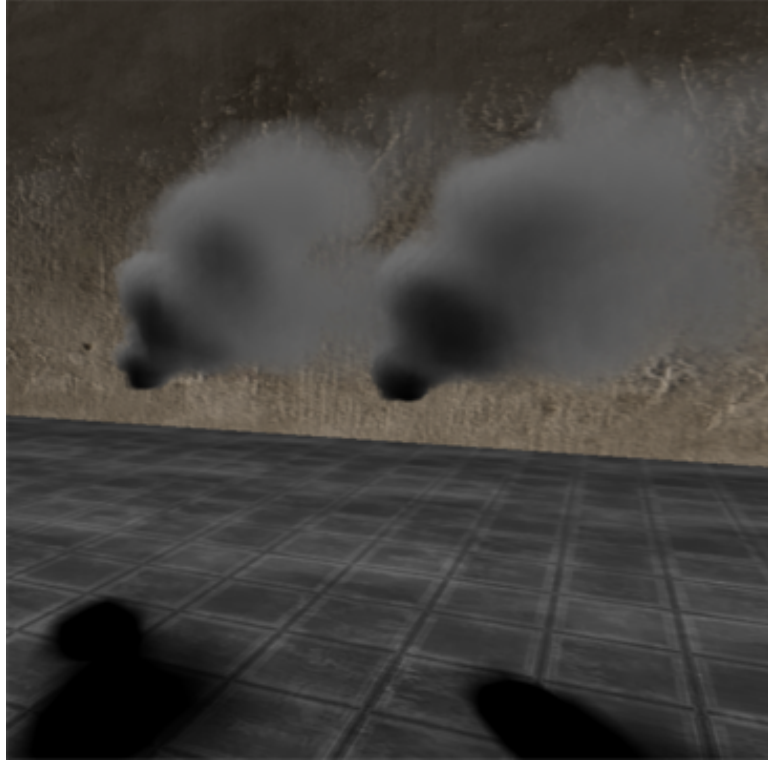


Image 2: Smoke with an Opacity Shadow Map after a wind effect

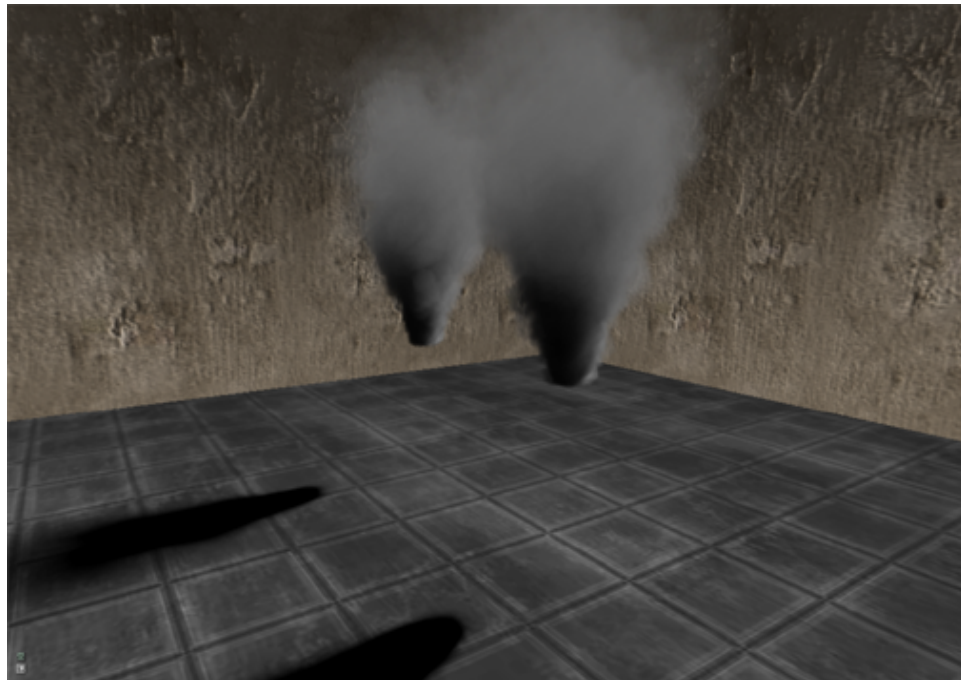


Image 3: Smoke with a Deep Opacity Map

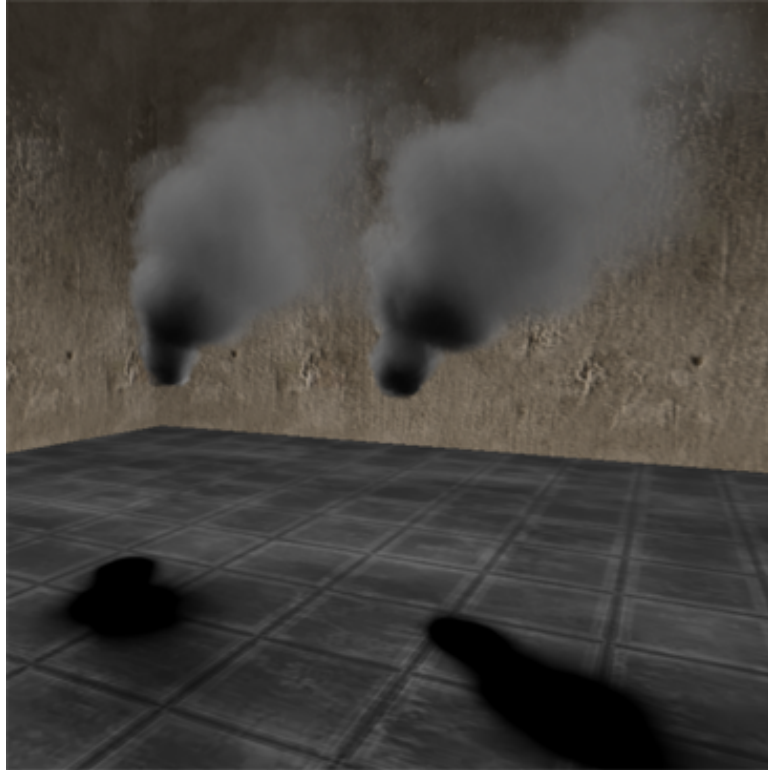


Image 4: Smoke with a Deep Opacity Map after a wind effect

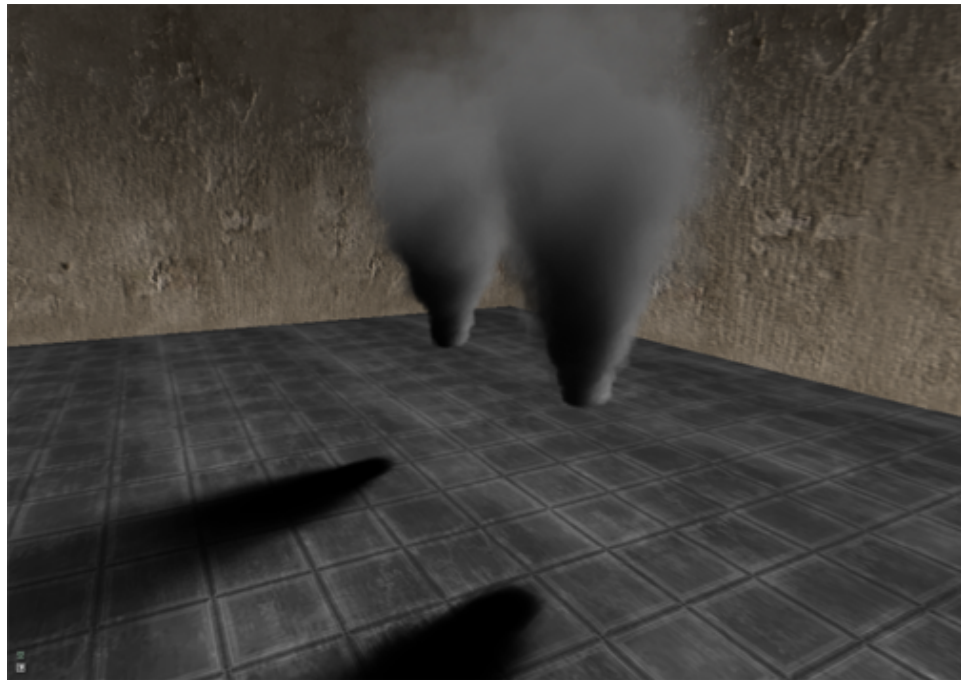


Image 5: Smoke with a Fourier Opacity Map

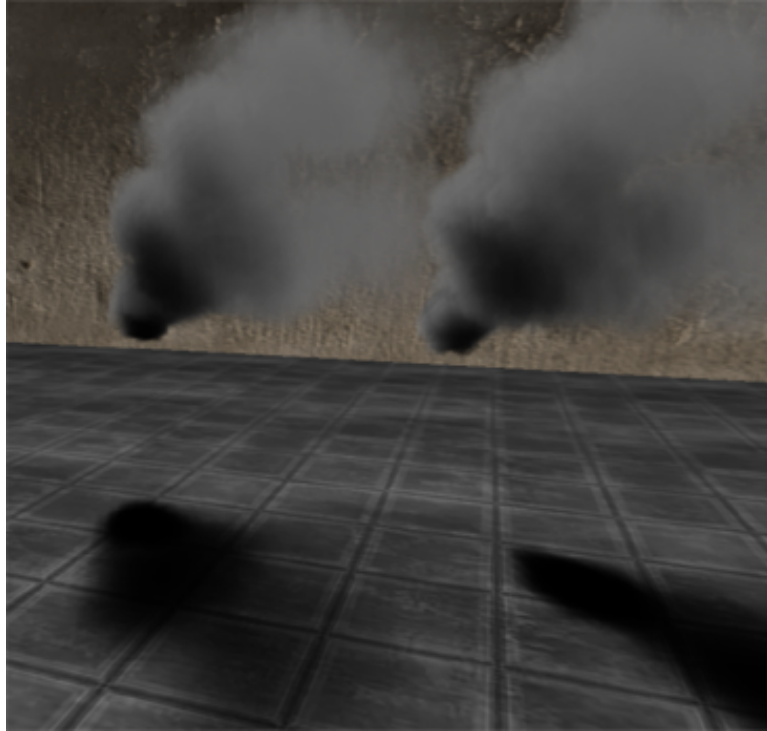


Image 6: Smoke with a Fourier Opacity Map after a wind effect

4.2.2.2 Accuracy Comparison

The number of render target texture components (r,g,b,a) used for the baseline images was 24. The difference between baseline images for all techniques is below, followed by the difference between baseline images and images with lower numbers of components including 16, 8, and 6.



Image 7: 24 slice OSM baseline



Image 10: Image 7 and Image 8 difference enhanced by 4x



Image 8: 23 slice DOM baseline



Image 11: Image 7 and Image 9 difference enhanced by 4x



Image 9: 24 coefficient FOM baseline



Image 12: Image 8 and Image 9 difference enhanced by 4x



Image 13: 24 slice OSM baseline



Image 14: 16 slice OSM



Image 15: 8 slice OSM



Image 16: 6 slice OSM

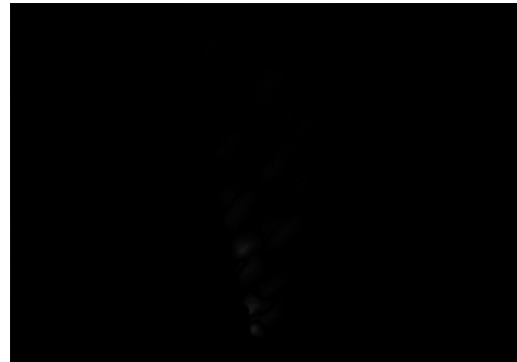


Image 17: Image 13 and Image 14 difference enhanced by 4x

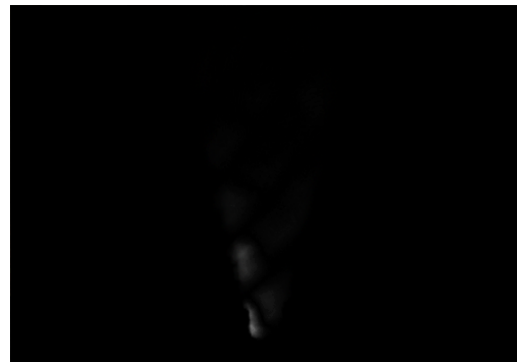


Image 18: Image 13 and Image 15 difference enhanced by 4x

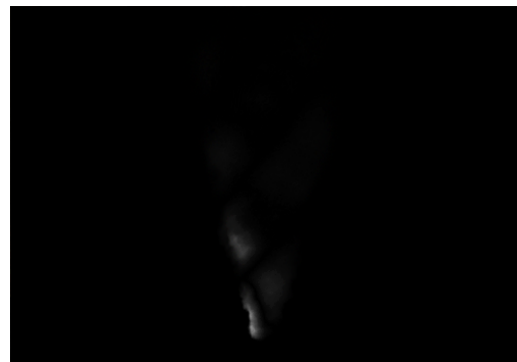


Image 19: Image 13 and Image 16 difference enhanced by 4x



Image 20: 23 slice DOM baseline



Image 21: 15 slice DOM



Image 22: 7 slice DOM



Image 23: 5 slice DOM



Image 24: Image 20 and Image 21 difference enhanced by 4x



Image 25: Image 20 and Image 22 difference enhanced by 4x



Image 26: Image 20 and Image 23 difference enhanced by 4x



Image 27: 24 coefficient FOM baseline



Image 28: 16 coefficient FOM



Image 29: 8 coefficient FOM



Image 30: 6 coefficient FOM

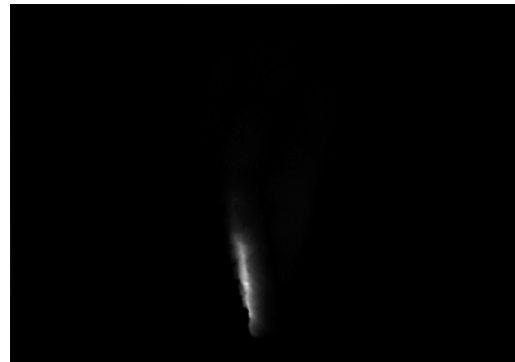


Image 31: Image 27 and Image 28 difference enhanced by 4x

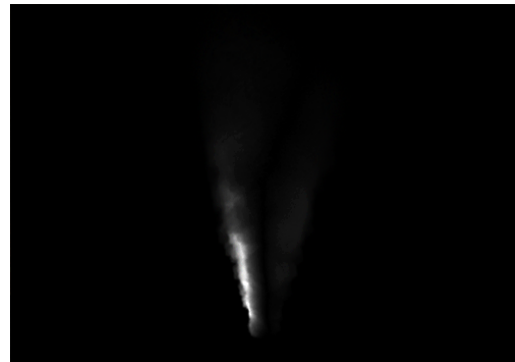


Image 32: Image 27 and Image 29 difference enhanced by 4x

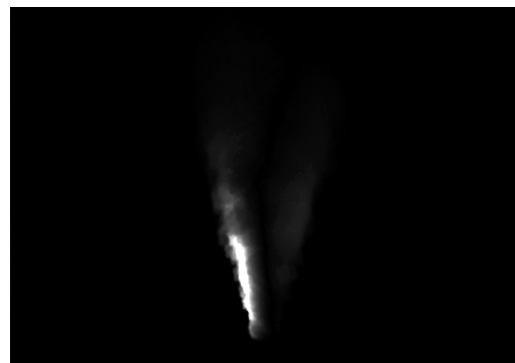


Image 33: Image 27 and Image 30 difference enhanced by 4x

The baseline images are considered to be ground truth images, because they use the most slices or coefficients to compute the shadow. The same volume and scene conditions were used in order to render all of the baseline images for all of the techniques. In addition, the same fixed amount of memory was used. The baseline images are similar to each other, but some differences show when we compute the difference between baseline images from two different techniques, as we have done for images 10, 11, and 12 above. For this case, it appears that OSMs and DOMs have represented the shadow most accurately, because the highest quality baseline images were almost identical. In addition, there was almost no difference between images 13 and 14, as well as images 20 and 21. All of the techniques become more accurate when larger amounts off memory are used for slices or coefficients. By taking the difference between a baseline image, and an image produced using a lower fixed amount of memory, we validate that the accuracy error has to be at least the amount shown in the resulting image difference. Moreover, the closer the baseline image is to the exact image, the better the bound on error is. Consistent with image results of hair from [YK08], DOMs appears to be more accurate than OSMs for low fixed amounts of memory, because images 18 and 19 are more noticeable than images 25 and 26. On the other hand, FOMs appeared to perform most poorly in terms of accuracy. The baseline image for FOMs was greatly different from the baseline images of OSMs and DOMs, and the difference between the baseline image (*Image 27*) and images of lower fixed amounts of memory (*Image 28, 29, and 30*) were large. Inaccuracies of FOMs consistent to those shown here can be found in [SVLL10], and are attributed to ringing and a less-than-optimal depth bounds. Inaccuracies with OSMs and DOMs are attributed to slicing artifacts. Note that the scene used to produce our accuracy results does not show additional artifacts that may be caused by outliers.

4.2.3 *Rendering Clouds of Smoke with Shadows Analysis*

The techniques have differences that may have led some to perform better or worse than others. All of the methods had competitive rendering speed for the varying fixed amounts of memory used. One reason for FOMs rendering slightly more quickly than the others with low fixed amounts of memory is that particles do not have to be sorted when building a FOM, but they need to be sorted when building an OSM or a DOM. With larger fixed amounts of memory and more accuracy, rendering with FOMs slowed down significantly and became the slowest of all. Building an OSM is simply additive blending of opacity, and rendering is simply linear interpolation. OSMs only slightly outperforms DOMs in rendering speed, because they are only slightly different techniques. On the other hand, building a FOM is additive blending of Fourier coefficients with trigonometric functions, and reconstructing the transmittance function for rendering is the evaluation of more trigonometric functions.

Looking at accuracy, FOMs performed most poorly, because the differences between its baseline image and each image using a lower fixed amount of memory was large. The accuracy of OSMs and DOMs was higher with lower fixed amounts of memory, and baseline images of the techniques were similar to each other. They both had slicing artifacts with lower fixed amounts of memory, but the artifacts of DOMs appeared to be significantly less in magnitude, because the slices conform to the shape of the smoke. On the other hand, FOMs had ringing artifacts. With the same fixed amount of memory used for the baseline images, FOMs appeared to be furthest from matching the baseline images of the other techniques.

OSMs and DOMs performed the best in terms of memory usage and rendering speed with larger fixed amounts of memory. With nearly the same number of slices, OSMs is faster and uses less memory. However, DOMs uses less slices for comparable accuracy. There are ways of adding accuracy or reducing artifacts. With slicing methods like OSMs and DOMs, we can add more slices, position them without uniformity, or use

cubic spline interpolation instead of linear interpolation. With FOMs, we could certainly add more coefficients used to represent the transmittance function. However, combating accuracy issues in such a way comes at the cost of reduced performance in speed and memory usage.

Chapter 5: Conclusion and Future Work

We achieved our goal of determining levels of accuracy and performance of methods used for rendering clouds with shadows using vertex and pixel shaders. We implemented techniques used to render clouds with shadows, produced results for our implementations, and analyzed the results. Before the techniques were implemented, we did a survey of many techniques specifically used for modeling clouds, rendering clouds, and rendering cloud shadows. The survey gave us the knowledge that we needed for our own implementations, so that we could generate our own results.

We have highlighted strengths and weaknesses of many of the techniques used to model and render clouds with shadows, and we organized the progression of the field of cloud volume graphics with shadows for reference. It is not known what future changes to hardware might make a technique that is researched here improve in some aspect. An improvement could amount to a more desirable technique in the long run. However, this research makes choosing the appropriate method for rendering clouds with shadows easier.

One possible improvement would be to use cubic spline interpolation for sampling opacity from slice-based techniques like OSMs and DOMs in order to attempt to smooth slice artifacts. Another improvement would be to use a mesh or some other approximation of clouds in order to render the front-most depths from the light when building a DOM. The goal would be to improve cloud shape approximation, and it could make building a DOM more efficient and smooth out artifacts caused by using the particles themselves. Another possible improvement would be to handle opaque polygons in the volumetric shadow maps. All of our shadow maps are computed using a spot light. An omnidirectional light using environment mapped shadows might be possible and worthwhile. Another improvement would be to handle intersections of clouds with opaque terrain polygons without artifacts using Spherical Billboards discussed in section 3.2.2.3. The use of OpenCL, Nvidia's CUDA, or some of the techniques found in [JB11] would increase efficiency. One possible way to increase the speed of OSMs and DOMs

might be to use the half-angle described in section 3.3.3.3 in order to combine HAS with OSMs or DOMs for the purpose of performing only one sort for the light pass and the view pass combined. Sorting a large number of particles can be costly in terms of speed, so sorting the particles once, instead of twice, could increase efficiency.

References

- [EHKRW06] Klaus Engel, Markus Hadwiger, Joe Kniss, Christof Rezk-Salama, Daniel Weiskopf, "Real-Time Volume Graphics." A K Peters, 2006
- [R83] William T. Reeves, "Particle Systems-A Technique for Modeling a Class of Fuzzy Objects", 1983
- [C74] Edwin Earl Catmull, "A Subdivision Algorithm for Computer Display of Curved Surfaces", 1974.<www.pixartouchbook.com/storage/catmull_thesis.pdf>
- [BN76] James F. Blinn, Martin E. Newell, "Texture and Reflection in Computer Generated Images", ACM, 1976
- [MHH08] Tomas Akenine-Möller, Eric Haines, Naty Hoffman, "Real-Time Rendering." A K Peters, 2008
- [TY09] Junichiro Toriwaki, Hiroyuki Yoshida, "Fundamentals of Three-dimensional Digital Image Processing." Springer, 2009
- [W07] Daniel Weiskopf, "GPU-Based Interactive Visualization Techniques." Springer, 2007
- [G05] Stefan Gustavson, "Simplex Noise Demystified", 2005.<www.itn.liu.se/~stegu/simplexnoise/simplexnoise.pdf>
- [S08] Allen Sherrod, "Game Graphics Programming." Cengage Course Technology, 2008
- [R10] Steve Rabin, "Introduction to Game Development 2010.", 2010
- [LC87] William E. Lorensen, Harvey E. Cline, "Marching Cubes: A High Resolution 3D Surface Construction Algorithm", ACM, 1987
- [L88] Marc Levoy, University of North Carolina, "Display of Surfaces from Volume Data", ACM, 1988
- [L03] Mats Lindh, "Marching Cubes", 2003.<<http://www.ia.hiof.no/~borres/cgraph/explain/marching/p-march.html>>
- [J01] Andreas Jönsson, "Fast Metaballs", 2001.<<http://www.angelcode.com/dev/metaballs/metaballs.html>>

- [M01] Radomir Mech, "Hardware-Accelerated Real-Time Rendering of Gaseous Phenomena", Journal of Graphics Tools, 2001
- [C05] Francesco Carucci, "GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation." Addison-Wesley Professional, 3 (Inside Geometry Instancing), 2005
- [USS06] Tamas Umenhoffer, Laszlo Szirmay-Kalos, Gabor Szijarto, Budapest University of Technology, "Spherical Billboards and their Application to Rendering Explosions", ACM, 2006
- [PPLSHS99] Steven Parker, Michael Parker, Yarden Livnat, Peter-Pike Sloan, Charles Hansen, Peter Shirley, "Interactive Ray Tracing for Volume Visualization", ACM, 1999
- [DKCBK98] Frank Dachille, Kevin Kreeger, Baoquan Chen, Ingmar Bitter, Arie Kaufmany, "High-Quality Volume Rendering Using Texture Mapping Hardware", Siggraph, 98
- [L90] Marc Levoy, University of North Carolina,, "Efficient Ray Tracing of Volume Data", ACM, 1990
- [H09] Kyle Hayward, "Volume Rendering 102: Transfer Functions ", 2009.<<http://graphicsrunner.blogspot.com/2009/01/volume-rendering-102-transfer-functions.html>>
- [O4] Nvidia, "SDK White Paper : Fog Polygon Volumes (Rendering Objects as Thick Volumes)", 2004
- [W78] Lance Williams, New York Institute of Technology, "Casting Curved Shadows on Curved Surfaces", Computer Graphics (SIGGRAPH '78 Proceedings), 1978
- [DL06] William Donnelly, Andrew Lauritzen ,University of Waterloo, "Variance Shadow Maps", ACM, 2006
- [JB09] Jon Jansen, Louis Bavoil, "Fourier Opacity Mapping", ACM, 2009
- [LV00] Tom Lokovic, Eric Veach, "Deep Shadow Maps", ACM, 2000
- [KN01] Tae-Young Kim, Ulrich Neumann, University of Southern California, "Opacity Shadow Maps", ACM, 2001

- [KPHSM03] Joe Kniss, Simon Premoze, Charles Hansen, Peter Shirley, Allen McPherson, "A Model for Volume Lighting and Modeling", ACM, 2003
- [YK08] Cem Yuksel, John Keyser, "Deep Opacity Maps", EUROGRAPHICS, 2008
- [HKSB06] Markus Hadwiger, Andrea Kratz, Christian Sigg, Katja Buhler, "GPU-Accelerated Deep Shadow Maps for Direct Volume Rendering", Proceedings of Graphics Hardware, 2006
- [G08] Simon Green, "Volumetric Particle Shadows", 2008
- [FIKLN04] Randima Fernando, Milan Ikits, Joe Kniss, Aaron Lefohn, Charles Hansen, "Volume Rendering Techniques." Pearson Education, 39, 2004
- [SVLL10] Marco Salvi, Kiril Vidimce, Andrew Lauritzen, Aaron Lefohn, "Adaptive Volumetric Shadow Maps", ACM, 2010
- [JB11] Jon Jansen, Louis Bavoil, "Fast Rendering of Opacity Mapped Particles Using DirectX 11 Tessellation and Mixed Resolutions", 2011