CLOUD MODELING AND RENDERING

by

Daniel James Perry

A Senior Honors Thesis Submitted to the Faculty of
the University of Utah
in Partial Fulfillment of the Requirements for the

Honors Degree of Bachelor of Science

in

Computer Science

APPROVED:

Peter Shirley Supervisor	Martin Berzins Director, School of Computing
Peter Shirley Departmental Honors Advisor	Martha S. Bradley Director, Honors Program

April 5, 2007

ABSTRACT

A central challenge of computer graphics is rendering realistic natural scenes. A challenging part of natural scenes are aesthetically pleasing rendering of clouds and other gases. We describe current methods of rendering realistic and non-realistic clouds and gases. We then discuss an implementation of these methods. Finally we present recommendations for successfully rendering clouds and other gases in both realistic and non-realistic settings.

For Mom and Dad.

TABLE OF CONTENTS

Abstract **ii**

1 Introduction **1**

2 Current Methods **3**

2.1 Overview 3

2.2 2D Textures 5

2.3 3D Procedural Texture Methods 8

2.4 Simulation Methods 10

2.5 Volume Rendering 11

2.6 Alternate Rendering Methods 12

2.7 Summary 13

3 Gas Rendering Implementation **14**

3.1 Introduction 14

3.2 Implementation 15

3.2.1 Concepts 15

3.2.2 Implementation 16

3.3 Conclusions 18

4 Cloud Rendering Implementation **19**

4.1 Introduction 19

4.2 Implementation 20

4.3 Conclusions 23

5 Conclusions **24**

5.1 Conclusions 24

5.2 Future work 25

Chapter 1

Introduction

Misty forest glens. Cloudscapes. Foggy river banks. Some of the most rich and beautiful scenes depicted in art and today in computer graphics, are natural scenes. For ages artists strove to capture, and to an extent have captured, these picturesque scenes that inspire and awe. More recently, a newer breed of artists, those facilitating lighting approximations and geometric modeling instead of the traditional canvas and paint brush, are also striving to capture scenes of beauty. Indeed one of the central challenges of computer graphics is to reproduce on screen in a convincing way, what one has seen in real life. An important part of many natural scenes are the sometimes central, and other times seemingly unnoticed appearance of gas and clouds. In a scene like a sunset, they take center stage. In others, they take seemingly unnoticed roles. And yet in the latter, when absent, they take with them an important element of realism. Clouds and gas are an important part of many a natural scene.

Equally important are indoor and other unnatural scenes. In this realm instead of cloudscapes we find the need to capture steam, smoke and other gaseous elements. A steaming shower or a cup of hot cocoa without the steam wouldn't be the same. Neither would a fire without smoke or a car exhaust without the visible evidence of its function. Gas provides important visual clues in many situations.

Clouds and gas can be one of the more challenging elements to both model and

render. They present difficulty because they are not rigid forms, nor do they maintain a solid surface. Instead, clouds and gases have a very irregular shape. This makes it difficult to approximately represent them using traditional polygonal models. They are truly volumetric elements of a scene, because their shading can only really be calculated using a true volumetric model.

Another difficulty is that we are familiar with how clouds and gases look. Because we encounter them daily in their many forms, we are familiar with what a gas, and specific instances of gases, are supposed to look like. From a rendering perspective, this means that if we don't get enough features correct, the gas will not be convincing to the human eye.

There has already been a lot of work done in this area, both directly and indirectly. Gaseous elements in natural scenes have attributes that are not completely unique in their shape and shading difficulties. In fact most of these difficulties are shared with the more general problems like volume rendering, hyper-textures, and solid textures. Researchers exploring cloud modeling and rendering have found incredible applications of techniques like solid textures and hyper-textures to the problem of producing natural looking elements in a cloud model.

Taking into account these different approaches, we have summarized and commented on the current methods used at this time for realistic gas and cloud modeling and rendering. We have also selected an approach to modeling and rendering realistic clouds and gas, and implemented it. We present our results and recommendations from the implementation, and our conclusions.

Chapter 2

Current Methods

2.1 Overview

Because of the motivation to create convincing cloud scenes, or at least aesthetically pleasing images, there has been a lot of work done in improving cloud rendering. There has also been a lot of work done in work paralleling cloud rendering - like volume rendering. In considering cloud modeling and rendering, we will first review methods of representing clouds as two-dimensional objects, and then as full three-dimensional. Looking at three dimensional modeling, we can divide methodologies into two broad categories: simulation and procedural. As concerning cloud rendering of three-dimensional models, we also step back and consider methods of more general volume rendering.

Because of needs for interactive speeds and limits of previous hardware, some research has gone into presenting clouds and gases in two dimensions. While these methods have merit in speed, they have problems with realism. We examine briefly some work in this area.

More recent methods attempt to use full three-dimensional modeling and rendering in order to obtain more realistic and pleasing results. Again, modeling is generally done by either simulation or procedural methods, and rendering is done using more

general volume rendering techniques.

In simulation, we attempt to model the physical characteristics of the cloud, along with its interaction with its surroundings. Accurate simulations have the ability to create very realistic cloud models. However, because of all the physical simulation going on behind the scenes, cloud simulation is generally very computationally intensive. They are also difficult to control to obtain an desired artistic look and feel. In essence you get a very good result at a very high cost and physics-based controls.

Using procedural methods provides different benefits and problems. The major benefit of procedural methods over simulation is their comparative simplicity. Not only more simple, but if designed in the right way, a procedural texture gives far more control over the model. The artist is not restricted by a physical model, or even to describing the cloud in physical terms. Instead the artist can describe the cloud in physical dimensions and fine tune other attributes, such as thickness, color, whispiness, etc. Procedural methods use a simplified function that describes only the visual attributes of the cloud, instead of physically modeling the cloud or gas. The function doesn't necessarily model a real cloud, but rather provides density and other values needed to create a visually convincing image. The difficulty of procedural methods lie mostly in creating a function that accurately models a cloud. The realism of the cloud is also restricted by the model chosen.[7]

Whether we use simulation, procedural, or any other method of modeling gas and clouds, we also need to be able to render the model. We turn to current methods of volume rendering in general and what specific subset of volume rendering is used to render clouds and gas in scenes. Because volume rendering is used in so many other fields this area has matured greatly, with more recently even interactive volume rendering development.

2.2 2D Textures

One of the simpler approaches to cloud rendering is in two-dimensional textures. Normally this approach is taken when the clouds are to be no more than a texture in the background of the scene, with no interaction with other elements in the scene. In this situation, because detail isn't quite as important for background images, the focus is on cloud rendering, and not on modeling. There is no real model of the cloud, as in 3D approaches we will discuss later, instead we concern ourselves only with producing a texture that resembles a cloud at a distance.

2D textures can really be generally classified into two broad catagories, "image textures" and "procedural textures". In a general sense, image textures involve storing the texture information, and then looking it up when using the texture. Procedural textures consist of using a function to produce the color needed at any point on the fly. While at the extremes they are different, they also overlap in many areas. For example we can do some preprocessing on an image texture, like anti-aliasing - but that doesn't necessarily make it a procedural texture. And we can store some initial information to be used in a procedural texture, like a random number lattice, but that doesn't make it into an image texture. Here we take a very general and safe distinction: where most of the creativity which emerges in the final image texture occurs, that's what we'll call it. For example, if we take a picture of a cloud and do simple anti-aliasing processing on it, that is considered an image texture. If I have a simple noise lattice, but my procedure takes that stored information and turns it into something that looks like a cloud, then we consider that a procedural texture. Our examples will be easily classified into these two broad categories, with the majority of our discussion on the latter group.

Using an image texture can give satisfactory results. However, using an image texture carries with it the same danger of any image texturing approach, that of possible incorrect shadowing/coloring for the scene, lack of detail at close inspection, and

a checkerboard look with the repetition of an image. Most of the difficulty associated with using image textures can be overcome, with tools like level of detail textures, blending, and selecting the right kind of images and image placement. Image texturing is a pretty standard feature of any rendering system, and can usually be added without too much difficulty in rare case that it is not included. For this reason image texturing can be attractive since relatively little work is needed to get a satisfactory result. Because image texturing is such a standard feature, and discussed in detail elsewhere, we refrain from further explanation on this topic.

Easily related to 2D image texturing is that of 2D procedural texturing. Like image texturing, in our context of creating cloud images, procedural texturing is limited to providing background texture and color, and doesn't really lend itself much to interaction with 3D objects within the scene. Procedural texturing is also a feature included in many graphics systems; because it naturally helps in understanding 3D procedural texturing, we discuss it here briefly.

Even procedural textures limited to two dimensions is a very broad category. They can be used in many ways. From the very simple, like a checkered floor tile, to the more complex, like the swirling texture of a marble table. The trick with procedural textures is figuring out how to make the procedure produce the colors you would like.

For a simple example, consider creating a texture for a checkered floor. We would need something that would alternate our tile colors in a regular n by n grid. Here is one method:

```
Color TileTexture( float s , float t ){
  int ss = (int)n*s, tt = (int)n*t;
  if( (ss+tt)%2 == 0 ) return color1;
  return color2;
}
```

This would produce a texture like that in figure 2.1.

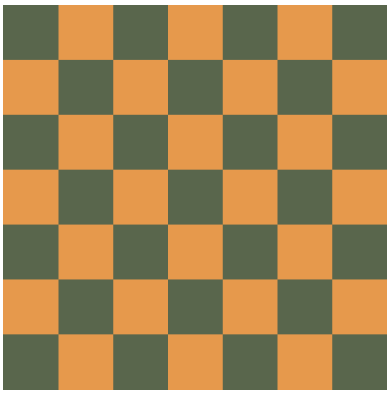

Figure 2.1: Result of a simple procedure for a checkered texture.

This gives us an idea of how a procedural texture can be constructed. One can imagine the limitless possibilities as well as the hours of tweaking to arrive at the texture one desires.

In 1985 Perlin produced a ground breaking paper on creating random-looking textures that could be both consistent and controlled [13]. Using his approach we can now create a texture that looks random, yet can be consistent across multiple frames and allows for some control of the randomness. These properties are very useful when creating any type of natural looking image.

During that same year Gardner [4] published a paper showing another approach to create natural looking procedural textures. By combining sine functions with different phases, amplitudes, and frequencies, he was able to create realistic looking clouds. Peachey in [1] gives a simplified example of Gardners approach as a RenderMan shader. This example demonstrates how using procedural textures one can achieve somewhat complex results. We refer the reader to Peachey's example for implementation details. Our results for his example are shown below in figure 2.2.

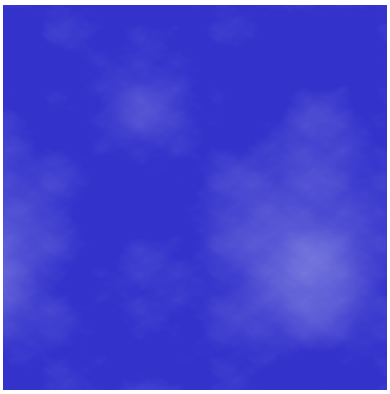

Figure 2.2: More complex 2D texture of a cloud.

As the example shows 2D textures can be used to produce images of varying complexity. Some benefits of a 2D procedural texture are that the procedure can normally handle any level of detail without any more work. If designed in the right way, the texture can also eliminate problems like the "tile look" where it is not desired, and having the texture change over the extent of the space - for example, have the cloud texture change the look of the cloud in different "tilings". This adds to the improved visual quality of the image.

These approaches to producing clouds are fine for background images. However, in order to render clouds that can interact with other parts of the scene (cast shadows, etc.) we move on to 3D modeling and rendering of clouds.

2.3 3D Procedural Texture Methods

Building on our discussion about 2D proceduarl textures, we can use functions to, at runtime, calculate the model of the cloud, rather that having it precomputed and stored. This is also different from a simulation, because rather than having some state that is continually modified over time by a set of functions, we simply evaluate a set of procedures that return a value - the same value every time they are evaluated with the same arguments (not random).

For example, in the implementation discussed later, we use procedural methods to compute the density of a gas at any point in 3-dimensional space. In other words we have some function density(x, y, z) that returns a density value for some point. The density of the cloud is not already precomputed, nor do we modify this state over time as in a simulation. (Although we could modify the procedure to be a function of time as well - density(x, y, z, t) - to allow for animation).

This provides us with some advantages. A table of density (or whatever we are modeling) does not have to be maintained. This eliminates memory usage and access time. Also, evaluation allows us arbitrary levels of detail at runtime. If defined properly we also have no limitation of the size of the procedure entity (we can define it for all 3d floats), and can be done without noticeable repetition of values.

This approach of course has some disadvantages. In general it's not always possible or easy to fit a 3-dimensional problem to a procedure (although gases do). Redundant computation can prevail if care is not taken. Procedures aren't always intuitive and can become complex or unwieldy.

These advantages and disadvantages are drawn from the more general comparison of procedural textures to using texture images.[1]

There has been a lot of work in procedural texturing in general, but also specifically for cloud rendering. Ebert has championed the use of procedural texturing in cloud rendering for sometime now[1],[7],[3] and [2]. Musgrave uses procedural texturing to create entire scenes, including clouds [1]. In the more general sense, procedural textures and methods have been used for innumerable things, and have played a big role in animation[1],[13],[12], and [14].

This approach to realistic cloud rendering has very good results. This paper is targeting more the artistic look of a cloud instead of a physically accurate representation of a cloud. Ebert, as cited above, had great results in developing a system

of rendering clouds using procedural textures. These same techniques have been adapted by many others, some examples are [9],[7].

2.4 Simulation Methods

Simulation methods attempt to use physically based equations to approximate density at a given point. Generally differential equations model how a grid of particles or voxels affect each other. This grid of voxels are given some starting state, and then at each iteration of the simulation the state of each voxel changes with relation to those surrounding it, according to the equations chosen to model the cloud. A straightforward example of this is given in [6].

This approach has its roots in mathematical modeling. Generally differential equations are setup to model the interaction of air particles as per real life. This mathematical model, already approximating the real event, is then approximated on computer hardware. The resulting simulation can be quite accurate results. Some of the earlier approaches to this problem were done by [8].

Although more accurate, because of the resources needed to accommodate more precise simulations this approach can be much more expensive in terms of computation. Further, if the creation of clouds is purely for artistic merit other approaches are more suitable[7]. However with increase in availability and lower in price of compute power, this is becoming less of a problem, and modified simulations can now run in real time[6].

2.5 Volume Rendering

Using either type of cloud modeling will use some type of volume rendering. In the both procedural texture modeling and simulation modeling, volume rendering is preferred to produce visually appealing and realistic results. The interaction of light with a gaseous volume is generally difficult to approximate without some sort of volume rendering. Again, [8], is one of the first to publish in this area, using ray casting for volume rendering. [3],[1] uses ray casting as well to volume render the densities modeled by procedural textures, to produce more realistic results.

Because volume rendering is useful in many ways, a lot people have spent time developing its effectiveness. Fields like scientific visualization, requiring visualization tools to be able to examine large sets of data, have developed volume rendering into a very useful tool. There are many different approaches to volume rendering, several compared and summarized in [11]. Ray tracing has been the traditional approach to volume rendering [8]. While it can be very costly, ray tracing has a very simple and direct adaptation to volume rendering. The basic algorithm is as follows. In order to approximate realistic lighting equations for volumes, the volume is split into voxels. The ray is then cast from the eye point (or the other way, it doesn't matter in the end) for each pixel or sample, and is stepped along each voxel. At each voxel an opacity and color are calculated or looked up, and then that voxel's contribution (as a function of the color and opacity) is added to the final color of that pixel. If we are starting from the eye then a running opacity contribution is also taken into account (to take into account dissipation of light as we consider voxels further and further from the eye). This basic algorithm can be improved upon in several ways. For example [10] presents some approaches to speed up this method by using spatial coherency (bounding volume like optimizations) and early ray termination.

While ray tracing is a very simple approach, and the approach taken in this paper, there are many other methods that have been developed to render a volume. An-

other popular approach is volume splatting. Volume splatting is more akin to z-buffer approaches to rendering. Instead of tracing out the voxels with a ray, each voxel is "splatted" into the viewing plane. This approach allows for much better performance with some reduction in accuracy.

As with ray tracing, volume splatting has been extended and improved by many researchers. [17] extends splatting by fixing some of the anti-aliasing problems from which the algorithm suffers. [15] extends the method by approximating the volume cells with hardware renderable transparent triangles.

While there are several other methods to volume rendering, ray tracing and volume splatting seem to be the dominant approaches to volume rendering. While any method of rendering can be used with any method of modeling, in the implementations presented in this paper we use ray tracing for volume rendering. A future exercise might be to use volume splatting in place of ray tracing.

Finally, while traditionally volume rendering has been done "offline" because of the compute-intensive nature of it, with the advent of programmable graphics hardware, some have been able to obtain real-time volume rendering on commodity machines[9].

2.6 Alternate Rendering Methods

Other methods of displaying realistic clouds have been explored as well. One exceptional example is [5], where researchers were able to create realistic looking clouds at interactive rates using a particle system and billboards. Their goal in development was to have realistic looking clouds at a computation cost viable for a computer game. [16] takes a similar approach to another level. Their approach is what is used for realistic cloud rendering in Microsoft's Flight Simulator. Their implementation boasts the ability to render a larger variety of cloud types and layering to create a more realisitic scene. This paper does not take this approach to realistic clouds.

We instead focus on 3D procedural approaches to rendering clouds and gases, more akin to [3].

2.7 Summary

As discussed previously, both simulation and procedural texture methods of modeling clouds and gases have their respective advantages and disadvantages. Because our motivation in modeling clouds is more in terms of artistically usefully and visually pleasing, instead of accurate representation, we have decided to follow the route of procedural methods in this paper. We also use ray tracing to volume render this model, for its simplicity.

Chapter 3

Gas Rendering Implementation

3.1 Introduction

Being able to model gases in a visually pleasing way is a necessary challenge. Gases, and gas-like, objects present themselves in many scenes. For example, steam from a hot drink, mist in the morning, haze or smog on a city (not quite clouds), smoke from a gun or fire, emphasized stench of an ogre's feet, and so on. To be able to render gases in a visually pleasing way can dramatically improve the details of any computer generated scene.

Modeling gas in a procedure, as described previously, is generally done using a three-dimensional volume density function. A volume density function is basically the adaption of solid texturing [13] for use in gas modeling. The volume density function maps three-dimensional points to a volume's density, aka $vdf : \Re^3 \rightarrow \Re$. The density is then used, along with opacity to render the volume of gas. This is done using normal volume rendering methods.[1]

3.2 Implementation

3.2.1 Concepts

Ebert [1] provides an incredible introduction and explanation of this topic in his book. In essence, we create a solid texture function, using noise and turbulence functions. Once we have created a function with the attributes we would like, we then shape the volume density function into what we would like it to look like.

For example, imagine we need a scene where ugly green steam is coming off the foot of an ogre. To create an effect using this methods, we would first create a solid texture function using noise and turbulence that created the type of "swirls" and other features that we would like. In rendering the procedural texture we would use a general volume rendering algorithm, using density as a cue for opacity, and then a user selected color for the visible steam; in this case, probably a repulsive green.

Once the general features of the gas volume are developed, we can then start to "shape" the volume into what we would like. Without shaping, what we have is a block (or whatever shape we are rendering it in) of steam, with edges and all. By shaping the block, we can make it appear more like a gas. Ebert [1] discusses two ways of shaping the volume.

The first is to get rid of the surrounding shape. We do this by gradually dissipating the density of the volume at the edges of the shape. This way it doesn't look like a glass case of steam, but a floating cloud of steam.

The second method suggested is to gradually and randomly lower the density of the steam as it moves upward (or in any other direction). This gives the appearance of the steam slowly dissipating as it moves away from the source.

3.2.2 Implementation

In implementation, we generally followed Ebert's[1] direction.

As a macro structure we use a cube. Then we implement a hyper-texture-like[12] surface on the cube. This texture basically volume renders the points inside the cube, using a ray-cast step through method. At each point in the cube a hyper-texture-like density function is evaluated and then contributed to the volume rendering.

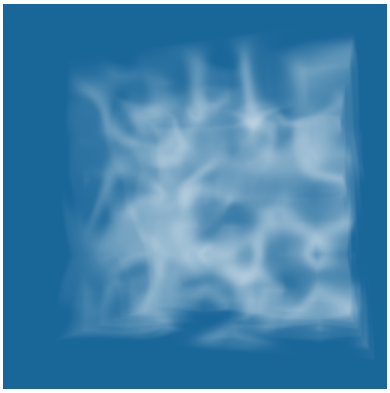

Figure 3.1: A basic cube of gas.

The trick to giving the volume its gaseous look is the density function. The density function needs to be able to look like it was really swirled and mixed with the air around it. Enter noise and turbulence functions [1],[13]. Noise and turbulence functions are tools that allow procedural textures to look more "natural" or at least unrepetitive and unpredictable.

As Ebert describes, the density function can be constructed by creating a simple, small lattice of noise values (our lattice was 64 by 64 by 64, as suggested in [1]). This lattice is then trilinearly interpolated at each sampled point. This is then fed into a "standard Perlin turbulence function". The turbulence value returned is then raised to a specified power (this differs depending on how thick you'd like gas to look).

Further modification of the turbulence value creates a more realistic density function. For example, using a sine function to create veins (as in Perlin noise influenced marble

textures [13]). These different modifications can be tweaked and altered to change how the gas looks.

Once the density function is working as desired, further improvements to the "gas block" can be made. As described in the concept section above, Ebert suggests two modifications for a steam-like gas. First, we drop off the density of the steam exponentially towards the edges, so it doesn't look like a block of steam.

A second improvement is to reduce the density of the steam as it rises vertically, thus giving the effect that the steam is slowly mixing with the surrounding air.

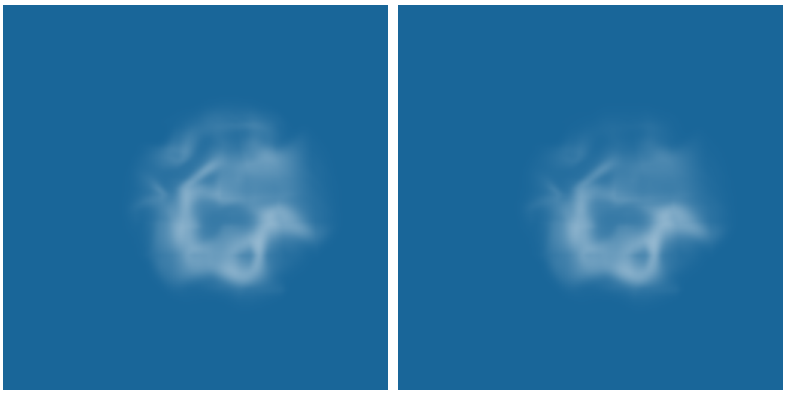

Figure 3.2: Edges are now not as visible (left), and steam dissipates as it rises(right).

By combining the strengths of hyper-textures, volume rendering, and Perlin noise, we are able to make a convincing steam.

Figure 3.3: Steam rises from a piping hot cup of cocoa.

We also concern ourselves with the shadow cast by the steam texture. Using the macro shape of the box would be very unrealistic (a box shadow for a gaseous body). Instead, for each shadow ray intersecting the box of steam, we calculate the density of the light penetrating the steam and multiply that by the light color which is added to the ambient term. This results in a shadow more appropriate for the body of gas.

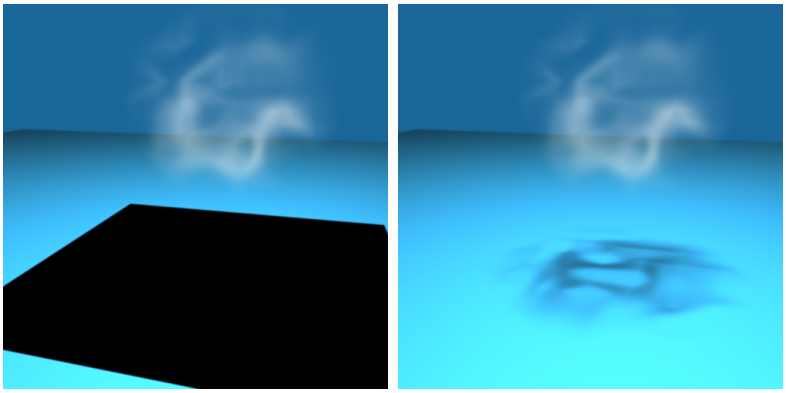

Figure 3.4: A shadow cast from the steam, using bounding volume (left), and using density of gas(right).

3.3 Conclusions

Implementing a procedural gas model and rendering system has provided more insight into the problem of rendering realistic gas. A procedural model has some good benefits. The greatest benefit is tweak-ability. Because it shares so much with the tools developed by Perlin, one can tweak specific arguments to get specific visual differences. For example, changing the power function argument can make the gas appear more thick. Changing the maximum density value also affects the visual appearance of the steam, and modifying the sine function changes the swirling effect on the gas.

Chapter 4

Cloud Rendering Implementation

4.1 Introduction

Cloud rendering is an important part of any outdoor scene. A cloudless sky can be less than convincing. There are innumerable ways to create a cloud, from the very simple, like a two dimensional image texture or procedure texture, to a full simulation, mimicking the real physical cloud.

But of course, the simple methods are too simplistic, and many times the more accurate methods are too complex. Texturing the picture of a cloud across a sky may suffice for very high clouds, with little movement in animation, but because they are just textured images they will remain stationary and any lighting will normally be incorrect, which may or may not be obvious.

A simple two dimensional procedural texture would improve upon the situation, because at view time calculations could make lighting realistic and even animation possible. However, if anything is going to interact with the clouds, or if we are going to be moving around the clouds, again, this will not due. Except at a distance, the clouds will look planar and unrealistic - more like looking at clouds through a window, than being in the scene with the clouds.

A three dimensional model of the cloud will clear up the remaining complaints -

that of immersiveness and interaction. Not only that, but for realistic lighting, the cloud can be shaded as a three dimensional volume.

As discussed earlier, a complex three dimensional simulation of the cloud would provide an ideal model of the cloud. But again, this would introduce complex and expensive simulation computation, not to mention the restriction on creativity introduced by physical simulation mechanisms.

Finally, there is the ability to use procedural volume textures to imitate the look of a cloud. While not physically correct, they would provide a way to produce truly three dimensional volumes of clouds. And, because they are built using standard animation tools like Perlin noise, it is easier to use arguments as more intuitive controls for the look of the cloud.

For these reasons, in this paper we have implemented and explore more the last option, that of creating clouds from procedural textures and volume rendering them.

4.2 Implementation

For this implementation, we again follow Ebert's explanation[1] with some diversion for simplicity. The general idea is very similar to the gas rendering described above. We essentially start with a volume texture of noise, and shape it into a cloud shape. We make sure to create very intuitive controls for the cloud shape. As Ebert describes, we create controls both for the "macro shape", or general shape of the cloud, as well as for the "micro shape", or smaller details - like density, whispiness, etc.[1]

Our implementation of the gas was limited in it's macro shape, essentially to a sphere or cube. This was admissible, since most steam and smoke is contained within one small area, and it generally dissipates outside that small volumetric area. Not so with clouds, they can expand across an entire sky, and don't really dissipate outside of a specific volume. This requires a more general method for determining the macro

shape of the cloud. So, in this implementation we specify a general bounding shape (such as a sphere or cube), and then inside of this bounding shape, we specify an additional macro shape, that actually determines the boundary of the cloud. We specify the inner macro shape using an implicit equation.

This provides an important advantage. We don't need to calculate the intersection of the volume rendering ray with the inner macro shape. We do have to evaluate each point in the volume rendering steps, but we have to calculate the density at these points anyway. This means we can use complex shapes without having to have a potentially expensive intersection algorithm; the shape only needs to have an implicit definition. This opens up the kinds of shapes and objects we can use to define the general shape of the cloud.

Figure 4.1: Two clouds side by side.

In his paper[3] and book[1], Ebert also adapts a way to speed up cloud shadowing by creating a shadow table first introduced by Kajiya[8]. This eliminates redundant shadow calculation by creating a table of shadow values for each frame. In order to keep our implementation simple, we instead recalculate every shadow evaluation. Even without the shadow table, our implementation runs at acceptable speeds, and gives encouragement at speedups available by implementing this feature in future work.

Shadow casting by the cloud is handled the same way as the gas. If the cloud is between any other objects in scene and the light, the amount of light that would get through at that point is calculated (the density is calculated in the same way a ray cast volume rendering ray would), multiplied by the light color and added to the ambient term. This way the shadow is neither completely occlusive, nor non-existent. It appropriately lets through enough of the light term to look like a cloud shadow.

Internal shadowing is handled in a very similar way. Every point evaluated within the cloud volume is first shadow tested against other objects in scene, then an inner shadow calculation is done, so that points further within the volume don't get as much contribution from the light source as those on the outside. This provides some good effect. We also increase the amount of ambient contribution to make up for the amount of reflected light within the cloud. Although not strictly the lighting model suggested by Ebert, this model gives satisfactory results.

Figure 4.2: Cloud without internal shading.

Figure 4.3: Cloud with internal shading.

4.3 Conclusions

The method as described works to both model and render a descent cloud. Some of the advantages of using the procedural texture approach to modeling the cloud, are the ease at modifying the visual appearance of the cloud. With this model we can thin out the cloud, making it look more like gas or a thick cloud, we can change the major shape of the cloud as desired, and all of this without having to really delve into the physical representation of a cloud.

As described, the rendering process runs at acceptable speeds. Modifications such as using the shadow table described by [8] and [1], and further reducing redundant shadowing and density calculations would definitely improve the performance of the implementation. Even without these optimizations, the runtime is acceptable. This is very encouraging, and if combined with recent work on real time volume rendering [9], these methods could be used in even real time applications.

Chapter 5

Conclusions

5.1 Conclusions

Through out this paper, we have examined various methods of rendering realistic gas and clouds. Generally these have been considered as methods involving simulations and methods using procedural textures to create realistic models and renderings of clouds.

After an examination of the various methods currently in you use, we've decided that procedural texture methods have an advantage over realistic simulations of clouds, generally in two areas. The first is in the artistic utility of procedural methods, as described by [1]. The second is the simplicity of the implementations. While procedural textures can become complex in their own right, they require no deep domain knowledge of cloud simulation. They instead use a more broad tool-set of procedural textures, one with which computer animators and artist should already be somewhat familiar.

An implementation of both gas and cloud modeling and rendering using a procedural approach has supported these arguments. Not only this, but without implementing some of the optimizations described by various researchers in [1] we have been able to model and render gas and clouds at acceptable rates. This is very encouraging for

further developments along this approach.

Considering all these factors, we recommend this procedural texture approach. While approaches using simulation have been able to reach real time speeds, notably in [6], they still lack the control made available by procedural textures. Being able to run them in real time also requires a lot of simplification to the simulation. Further, with work done in real time volume rendering, such as [9], procedural methods like those implemented in this paper would then be very ideal for artistic animation of clouds and gases.

5.2 Future work

The methods used in this paper have given very satisfactory results at reasonable speeds. Improvements to the implementation would include using the shadow tables described in [1]. We've also taken a very simplistic lighting model for the clouds, taking into account internal reflection by increasing the contribution by the ambient term. Another improvement would be to use a more accurate lighting model for the clouds. Some are suggested in both [1] and [9].

Another approach that would be useful to explore would be the use of volume splatting or one of its derivatives and volume textures, instead of ray tracing. Such an approach may also contribute to the speed of the methods explored in this paper.

Finally, the approach taken by [5] in using both particle systems and bill-boarding gave them very good results. They are able to interactively render beautiful cloud scenery and interact with it. Combining those methods with the procedural texture methods implemented in this paper would be a worth while project. This would result in renderings done at speeds acceptable for game play (and other game computation), but with the functional and controlling elegance provided by procedural texturing.

Bibliography

[1] Darwyn Peachy Ken Perlin David S. Ebert, F. Kenton Musgrave and Steven Worley. *Texturing and Modeling: A Procedural Approach*. Morgan Kaufmann Publishers, San Francisc, CA, USA, 2003.

[2] Oliver Deusen, David S. Ebert, Ron Fedkiw, F. Kenton Musgrave, Przemyslaw Prusinkiewicz, Doug Roble, Jos Stam, and Jerry Tessendorf. The elements of nature: interactive and realistic techniques. In *GRAPH '04: Proceedings of the conference on SIGGRAPH 2004 course notes*, page 32, New York, NY, USA, 2004. ACM Press.

[3] David S. Ebert and Richard E. Parent. Rendering and animation of gaseous phenomena by combining fast volume and scanline a-buffer techniques. *Computer Graphics*, 24(4):357–366, 1990.

[4] Geoffrey Y. Gardner. Visual simulation of clouds. In *SIGGRAPH '85: Proceedings of the 12th annual conference on Computer graphics and interactive techniques*, pages 297–304, New York, NY, USA, 1985. ACM Press.

[5] Mark J. Harris. Real-time cloud rendering. *Computer Graphics Forum*, 20(3):76–85, 2001.

[6] Mark J. Harris, William V. Baxter, Thorsten Scheuermann, and Anselmo Lastra. Simulation of cloud dynamics on graphics hardware. In *HWWS '03: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*,

pages 92–101, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association.

[7] David S. Ebert Joshua Schpok, Joseph Simons and Charles Hansen. A real-time cloud modeling, rendering, and animation system. *Eurographics/SIGGRAPH Symposium on Computer Animation*, pages 160–166, 2003.

[8] James T. Kajiya and Brian P Von Herzen. Ray tracing volume densities. In *SIGGRAPH '84: Proceedings of the 11th annual conference on Computer graphics and interactive techniques*, pages 165–174, New York, NY, USA, 1984. ACM Press.

[9] Joe Kniss, Simon Premoze, Charles Hansen, and David Ebert. Interactive translucent volume rendering and procedural modeling. In *VIS '02: Proceedings of the conference on Visualization '02*, Washington, DC, USA, 2002. IEEE Computer Society.

[10] Marc Levoy. Efficient ray tracing of volume data. *ACM Trans. Graph.*, 9(3):245–261, 1990.

[11] Michael Meibner, Jian Huang, Dirk Bartz, Klaus Mueller, and Roger Crawfis. A practical evaluation of popular volume rendering algorithms. In *VVS '00: Proceedings of the 2000 IEEE symposium on Volume visualization*, pages 81–90, New York, NY, USA, 2000. ACM Press.

[12] K. Perlin and E. M. Hoffert. Hypertexture. In *SIGGRAPH '89: Proceedings of the 16th annual conference on Computer graphics and interactive techniques*, pages 253–262, New York, NY, USA, 1989. ACM Press.

[13] Ken Perlin. An image synthesizer. In *SIGGRAPH '85: Proceedings of the 12th annual conference on Computer graphics and interactive techniques*, pages 287–296, New York, NY, USA, 1985. ACM Press.

[14] Ken Perlin. Improving noise. In *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 681–682, New York, NY, USA, 2002. ACM Press.

[15] Peter Shirley and Allan Tuchman. A polygonal approximation to direct scalar volume rendering. In *VVS '90: Proceedings of the 1990 workshop on Volume visualization*, pages 63–70, New York, NY, USA, 1990. ACM Press.

[16] Niniane Wang. Realistic and fast cloud rendering in computer games. In *SIGGRAPH '03: ACM SIGGRAPH 2003 Sketches & Applications*, pages 1–1, New York, NY, USA, 2003. ACM Press.

[17] Matthias Zwicker, Hanspeter Pfister, Jeroen van Baar, and Markus Gross. Ewa volume splatting. In *VIS '01: Proceedings of the conference on Visualization '01*, pages 29–36, Washington, DC, USA, 2001. IEEE Computer Society.

Name of Candidate:	Daniel James Perry
Birth Date:	9 March 1983
Birth Place:	South Jordan, UT
Address:	1481 Midas Creek Drive, South Jordan, UT