2011-06-24

Rendering Realistic Cloud Effects for Computer Generated Films

Cory A. Reimschussel
Brigham Young University - Provo

Rendering Realistic Cloud Effects for Computer Generated Production Films

Cory Reimschussel

A thesis submitted to the faculty of
Brigham Young University
in partial fulfillment of the requirements for the degree of

Master of Science

Michael Jones, Chair
Parris Egbert
Mark Clement

Department of Computer Science

Brigham Young University

August 2011

ABSTRACT

Rendering Realistic Cloud Effects for Computer Generated Production Films

Cory Reimschussel
Department of Computer Science, BYU
Master of Science

This work addresses the problem of rendering clouds. The task of rendering clouds is important to film and video game directors who want to use clouds to further the story or create a specific atmosphere for the audience. While there has been significant progress in this area, other solutions to this problem are inadequate because they focus on speed instead of accuracy, or focus only on a few specific properties of rendered clouds while ignoring others. Another common shortcoming with other methods is that they are not integrated into existing rendering pipelines. We propose a solution to this problem based on creating a point cloud to represent the cloud volume, then calculating light scattering events between the points. The key insight is blending isotropic and anisotropic scattering events to mimic realistic light scattering of anisotropic participating media. Rendered images are visually plausible representations of how light interacts with clouds.

Table of Contents

List of Tables

List of Figures

v

Chapter 1: Introduction

Figure 1 Clouds rendered using our method

We address the problem of rending clouds. Clouds are difficult to render because they are composed of water vapor droplets which scatter light as the light passes through the volume of the cloud. The problem is even more difficult because water scatters light with different intensities in different directions, also known as anisotropy—and in some cases this anisotropy results in visually significant effects.

Film directors use cloud effects to tell stories. Clouds can be used to quickly set the scene in a film or game. Directors use the ominous thunderhead, the glowing sunset or the lazy cumulous

floating through the summer sky to evoke a feeling in the audience. For live action film, a director might have to wait several days to shoot clouds in the right atmospheric conditions. CG film directors have the advantage that the artist can create and completely control the clouds in a scene or shot. The artist does not have to wait for the ideal weather conditions but does have to create the clouds using a computer.

Simulating clouds in a computer is a complex task involving many facets. There are many solutions to the different aspects of generating computer clouds, each one focusing on a different piece of the solution and none offering a complete solution. The basic tasks include modeling, animation, shading, lighting, and rendering. Each of these aspects must be considered when creating cloud effects for CG films.

While existing rendering approaches, described later, provide good solutions to the problem they address, these methods are inadequate in that they are not physically accurate, or are not easily integrated into a production pipeline. Our work seeks to provide a method that resolves these shortcomings.

Wavelength dependent global illumination algorithms for anisotropic participating media could be used to render clouds (Max 2004). However, this is prohibitively expensive so specialized methods have been developed to reproduce illumination effects for clouds but at a lower computational cost. One of the more significant problems in rendering clouds is the choice of phase function. The phase function describes how light is scattered as it passes through a material. Clouds scatter light anisotropically, so it is important to use a phase function that mimics this behavior. Some methods ignore the phase function altogether (Wang 2003) and instead focus on cloud shape and transparency. Others chose to use the Rayleigh scattering function (Harris 2001, Harris 2003) for its simplicity. However, while it is easier to compute and

is accurate for smaller particles, like aerosols, the Rayleigh function is inaccurate for larger particles like water droplets in clouds. Light scattering through water droplets can be calculated using the Mie function but the Mie function is more complex to compute. Some works use a simplified version of the Mie function, such as (Trembilski 2002), which focuses on the strong forward scattering component. However, this omits features like the glory and fogbow.

The method described in (Bouthors 2008) does incorporate a pre-computed Mie function and is able to reproduce realistic cloud features by matching cloud shapes to slabs and using the pre-computed data for the slabs to approximate light scattering through the cloud shape. While this method is more physically accurate than other methods, (Bouthors 2008) still suffers because it is not easily integrated into an existing rendering pipeline as it was created specifically to run in real-time on a GPU. (Bouthors 2008) can also be difficult to work with because it produces multiple passes where each pass represents an order of scattering events. Each pass must be individually adjusted and combined with the other passes before the final image is produced.

Contribution

We present a physically-based solution for rendering CG clouds which achieves many realistic effects, including glory, fogbow and silver lining effects at a lower computational cost than true global illumination for clouds. We rely on scattering events between a sparse set of points within the cloud geometry to simulate global illumination effects. The key insight which drives this work is that while individual scattering events between water droplets in a cloud are anisotropic according to the Mie function, the sum of these anisotropic events is isotropic. In general, water droplets that receive light from one direction will scatter that light anisotropically. But if a water droplet receives light from many directions the net effect of light scattered from that droplet will be isotropic. These insights lead to a new method that simulates global illumination in clouds.

3

We track scattering events received at a point. If the point receives light from primarily one direction then we calculate light scattered from that point anisotropically. However, if the point receives light from many different directions then we calculate light scattered from that point isotropically.

Figure 2 illustrates this process. In the upper left is an image of the anisotropic Mie scattering function. In the image in the upper right the Mie scattering function has been modified by being summed with another Mie scattering function, oriented differently. This process continues with the addition of two more Mie scattering functions, all oriented differently, until the final image in the lower right that appears very similar to an isotropic scattering function.

Thesis Statement

Blending isotropic and anisotropic phase functions in a point-to-point approximation of light scattering within clouds results in believable images of clouds that can include the glory, fogbow and silver lining.
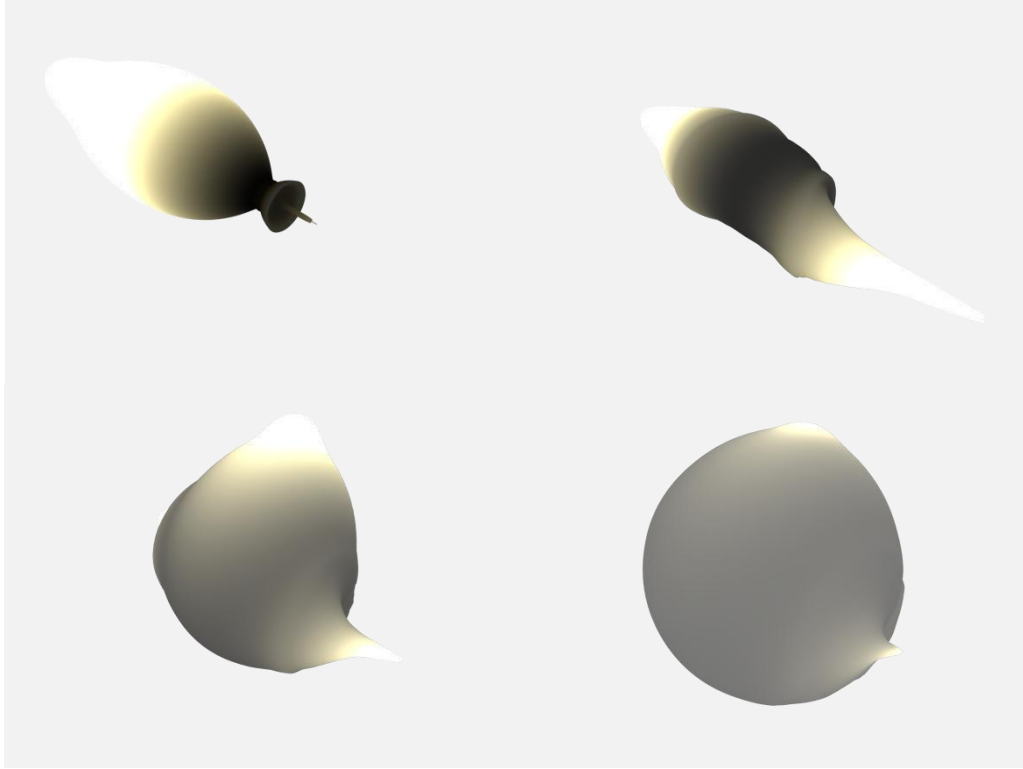
Figure 2 Conceptual visualization of the Mie scattering function

As scattering events are summed, the overall effect becomes more isotropic

Our method works by first taking a geometric volume and creating a collection of points inside that volume. We then perform multiple passes through the point cloud to simulate light scattering between points within the volume. We make use of the Mie function and the distance between any two points to attenuate the light scattering between those two points. For each point in the cloud, we also calculate a ratio of outgoing light scattered anisotropically, via the Mie function, and light scattered isotropically. This ratio preserves anisotropic scattering where it may be needed and gracefully switches to isotropic scattering in other places. The ratio allows the net effect of multiple anisotropic scattering events in different directions from a single point to be modeled using isotropic scattering. Our unique use of this ratio reduces rendering time while still reproducing specific realistic cloud effects. Once the simulation has finished, ray marching, along with the Mie function and the anisotropic/isotropic scattering ratio variable, creates a final

image from the light scattered, calculated and stored in the point cloud. Our work has been developed within the Maya/Renderman rendering pipeline. Our algorithm handles all standard geometry that can be integrated into the Maya/Renderman pipeline.

Figure 1 demonstrates the visual plausibility of the resulting clouds. In the image on the lower left, note the silver lining around the cloud which results from strong forward scattering on the model edge. In the image on the lower right notice the strong contrast between the lit and unlit sides of the cloud.

Chapter 2: Related Work

Modeling

Modeling is the process of creating 2D or 3D geometry that represents the general shape and position of the object in the computer simulated world. For modeling clouds, one of the easiest and quickest solutions is to create the geometry by hand using conventional geometry representations. Compared to geometry for characters or other more complicated objects, cloud geometry is simple to construct, and can be made from basic shapes. Modeling the clouds by hand using conventional geometry gives the artist more control over shape and position, but can be harder to animate. Using 3D geometry with volumetric shaders can also increase render time. In a production film it is important to find a compromise between output quality and render time.

Instead of trying to simulate the physics, most other approaches to rendering clouds focus on quickly obtaining a reasonable image. While results-based rendering methods can produce reasonable images, they often lack specific features that give added realism.

One of the most common techniques to render clouds is to use 2D billboards (Wang 2003). The billboards are usually layered to simulate 3D effects. Clouds rendered as billboards are significantly faster than full 3d clouds, but visual problems tend to increase as the distance from the cloud decreases. Animating the cloud billboards can also pose problems, and calculating effects like sub-surface scattering is difficult. This technique is commonly used to represent large numbers of clouds rendered a long distance from the camera.

Clouds can also be represented using particle systems. While it is possible to render the cloud particles as points, it is more common to render the particles as spheres/ellipsoids or metaballs (blobby surfaces) (Elinas 2000). An advantage of this approach is that it is easier to simulate

7

forces like wind or turbulence acting on the clouds. Representing clouds as a blobby surface using metaballs may be a very useful approach because it combines the benefits of particle simulation with the blobby surface. Since the blobby surface is rendered as a single piece of geometry, this method lends itself to the advanced rendering techniques along with the relative ease of particle simulation.

One particularly effective solution is to render particles as billboards, thus gaining the benefits of both particle based methods and billboard rendering. The combined approach works well for simulation and animation, but is not as useful for advanced rendering techniques like sub-surface scattering. It is also harder to model specific cloud shapes and sizes using particles.

One final approach is to model clouds using fluid simulations or volumetric representations (McGuire 2006). While this approach may be more accurate for animation and shading, modeling clouds with this technique is implicitly more difficult. Defining cloud shapes and sizes in a fluid simulation is much harder than creating and shaping a geometric mesh or even than creating clusters of particles, in part due to the complexity of the simulation system. But even disregarding complexity, fluid simulations can be undesirable because of the enormous computational requirements.

Animation

Ease of use is one of the main considerations when deciding how to model clouds. Another key consideration is what animation is needed for the clouds. If the director only needs the cloud to float across the sky, then the artist could simply key-frame standard geometry. However, key framing is difficult in some cases, particularly when multiple characters or objects interact. Animation requirements must be taken into account when modeling clouds.

8

The simplest and most standard animation method is key-framing. As noted above there are situations when key-framing clouds can be sufficient. But in general clouds are too complex to be simulated completely by key-framing. And while key-framing would work for regular geometry, it does not work for particle or fluid simulations or other dynamic simulations.

Because clouds are difficult to animate manually using key-framing, simulation tends to be the preferred method for representing clouds, and indeed much of the work on CG clouds is focused on real-time simulation. Cloud simulation may be as simple as attaching dynamic forces to particle systems, or may be as complex as full 3D fluid simulation. While it is harder to directly control cloud shape, size and motion with simulation techniques, it is easier for the artist to focus on general movements and effects without having to worry about all of the details. And in general, simulations tend to produce the most realistic results.

Another interesting approach to cloud motion is to animate the cloud shader or texture. This approach allows the artist to manipulate the intra-cloud motion and change small details within the cloud without changing the general cloud shape. Animating the shader or texture is an interesting solution because it mimics the natural behavior of clouds – clouds do not statically float across the sky, but are ever changing and molding into different shapes. Of particular interest is (Schpok 2003) which includes a texture animation method with interesting results.

Rendering

While working on modeling and animation or simulation, the artist must also keep in mind how the cloud is to be shaded or textured. The most common approach to shading clouds is to use fractal patterns (Wang 2003). Using fractals for surface and volumetric shading mimics the wispy effect of natural clouds. Fractals are used almost exclusively for shading clouds, and the

9

results are excellent as long as the scene calls for wispy clouds. In addition to the fractal textures, other techniques add to the look and feel of CG clouds.

By their nature clouds exhibit strong scattering effects. It is possible to simulate many of these scattering effects using techniques like multiple forward scattering, anisotropic scattering and Mie scattering. Many researchers have tried to reproduce and simulate natural cloud scattering effects. (Bouthors 2008) describes a particularly successful approach for approximating scattering effects for CG clouds. While calculating scattering effects in general generates more realistic and pleasing results, it is not always possible to implement such calculations. Ray traced scattering can be computationally cost prohibitive. Also, it may be difficult to calculate sub-surface scattering when the clouds are represented as a set of billboards instead of as a volume.

In situations where it is not possible to fully calculate scattering effects there are other techniques that approximate light scattering without the complex computation. One such technique is to use a deep shadow map. The map is computed from one or more light sources, but instead of darkening areas normally in shadow, the map is set to illuminate areas that are not in direct light. While not as precise as computing scattering effects, the deep shadow map technique can produce plausible results. Deep shadow maps with particle systems and fractal shaders were used in the BYU student film "Kites" (Kites 2009).

Chapter 3: Methods

Varying the ratio of light scattered isotropically and anisotropically, using a precomputed Mie scattering function, together with a point cloud representation of the cloud volume and ray marching algorithms results in a cloud rendering algorithm that produces visually plausible global rendering effects. The next section highlights key features of this new rendering method. The sections that follow describe the method in detail.

Features

Our method is unique because, while most other methods focus on either speed or physical accuracy, our method focuses on both. Our method reproduces some of the specific features of clouds most often ignored by other methods, such as fogbows, the glory and the silver lining, while still making use of speedups, such as pre-computing and reusing data that reduce computation time.

The key feature of this method is how it uses the Mie scattering function. The Mie function describes how light scatters through water particles, and is very anisotropic. The Mie function is what gives clouds some of their defining visual features like the silver lining effect, the glory and the fogbow. However most other methods do not make use of the Mie scattering function because it is very complex to compute. We use a pre-computed version of the function stored as a texture map that gives us the additional effects of the Mie function without having to compute it for individual scattering events.

The most physically accurate way to calculate scattering in clouds is to use path tracing with Monte-Carlo integration. While accurate, path tracing so many scattering events is very computationally time consuming. Instead of trying to calculate all of those scattering events, we

simplify the process by assuming that the light scattering at any one point can be more or less isotropic. Near the lit surface of the cloud we expect most of the scattering events to be very anisotropic because most of the light is coming from one direction. In the interior of the cloud we would expect that the combined effect of the individual Mie scattering events is isotropic because light is scattering from all directions. We mimic this effect by using a ratio value that controls how much light is scattered isotropically and anisotropically. This allows scattering events at a point to become more isotropic as that point receives light from more directions.

Another of the most important features of our method is reuse. Once light scattering in a cloud for a specific incoming light angle has been computed, the results can be reused for any view angle without any re-computation. Another speedup is that this method can be very easily parallelized – any one scattering event in a pass of the algorithm is independent of any other and can be calculated in any order.

Data Structures

Scattering data is stored in three data structures: the point cloud, the scattering function and a brickmap. A brickmap is a 3D data structure developed by Pixar to store data and is the functional equivalent of the 2D texture map. The point cloud stores the intensity and direction of both incident and exident light at points within the cloud volume. The scattering function is stored as a texture map. The brickmap is created using the output point cloud and stores final scattering data used in rendering. Each data structure is described in more detail in the following sections.

12

Point Cloud

The irradiance in a cloud created by light scattering in the cloud is stored as a point cloud. The point cloud is essentially a set of data points with a position in space and a set of data for each point. Table 1 lists the data that are stored in the point cloud. Details on how the point cloud is created are given later.

Table 1 Data Variables

Variable Name	Data
P_i	Stores the coordinates of each point in world space
L_i	A normalized vector that stores the prominent light direction Used when calculating the Phase Angle for the Mie function
$Lsum_i$	Vector that stores the summed direction of the prominent light vector for each point that contributes light to the current point
Ci_i	Stores the input or initial light percentage at the point At the end of each pass the calculated light contribution from other points is added to this value
Co_i	During the algorithm stores the summed light contribution from other points In the end is used to store the final or output light percentage
D_i	Density value (between 0.0 and 1.0) for that point Note: Varying the density adds noise to the results

| R_i | Value that stores the ratio (between 0.0 and 1.0) of Mie scattering to isotropic scattering |
| | Initially set to 1.0 (complete Mie scattering) |

Scattering Function

The Mie function describes how light is scattered by water droplets in a cloud. Given an incident and exidant light direction, the Mie function returns the intensity of the light scattered in the exidant direction as a percentage of the intensity of the light coming in from the incident direction. Mie scattering is both anisotropic and wavelength dependent. We used the Mie function computed in (Bouthors 2008), stored in the form of three texture maps, one for red, green and blue values. The texture maps were created by reading in the raw scattering data, converting it to an image, then converting the image into a texture. Lookup in the Mie scattering data is done by scaling and shifting the angle between incoming and outgoing light, also called the phase angle, to correctly fit the bounds of the texture map.

The Mie scattering function is depicted in Figure 3. In the image on the left the horizontal axis represents the difference between the incoming light angle and the scattering light angle, with the left side of the graph being 0° (forward scattering) and the right side of the graph 180° (backward scattering). The red, green, and blue lines represent the scattering values for the respective light components. The image on the right is a polar plot of the same data, oriented so that the incoming light angle is up.

The differences in scattered light intensity account for many of the lighting effects observed in clouds. The strong forward lobe accounts for the semi-transparent appearance of clouds lit from

behind. Small differences in intensity between the red, green and blue scattering functions result in rainbow-like glory and fogbow effects.
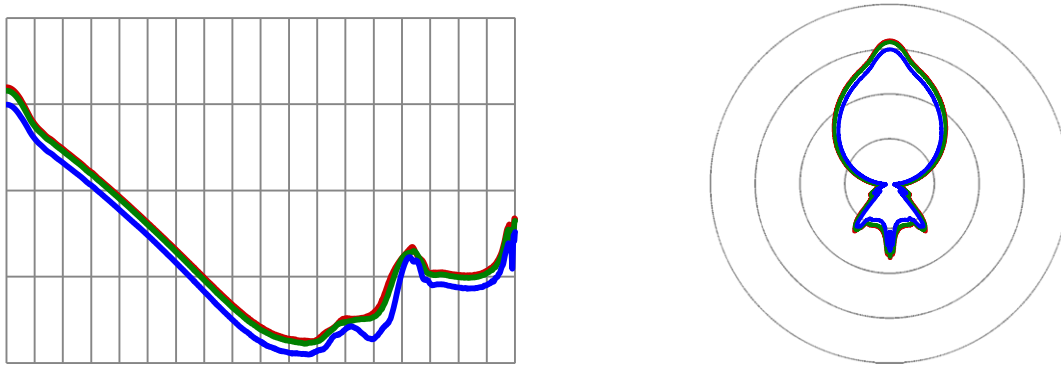

Figure 3 Mie scattering function plots

Mie Scattering Function as Cartesian and polar plot. For the graph on the left the vertical axis represents scattering intensity, while the horizontal axis represents the phase angle. The graph on the right is a polar plot of the same data, with the incoming light angle as up.

Brickmap

Before the final rendering, the output point cloud, together with the simulated scattering data, are converted to a brickmap. A brickmap is a Pixar-specific implementation of a 3D texture map. The brickmap generally requires less storage space and leads to lower lookup times when compared to point clouds.

Rendering Algorithm

The rendering algorithm is a three-step process. The first step is generating a point cloud that will represent the cloud volume. The point cloud stores all of the intermediate and final data. The second step is simulating how light scatters through the point cloud. The final step is to render the results from the simulation pass. Figure 4 shows snapshots from this process for a cloud shaped like a bunny. Each step is explained in more detail in the following sections.
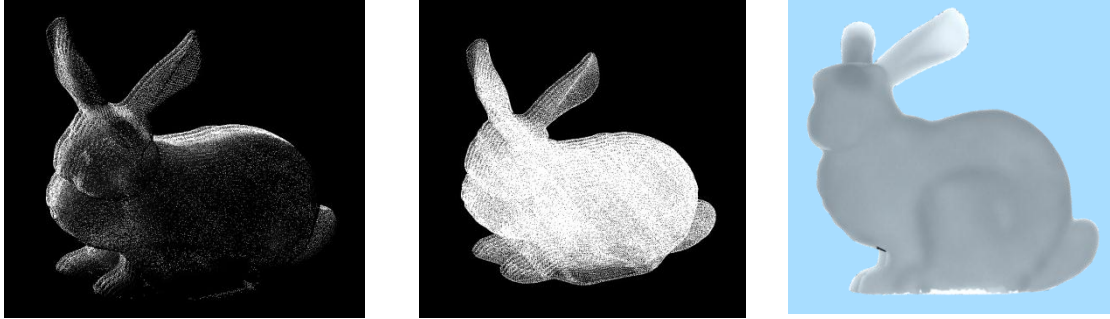
15

Figure 4 Steps in our cloud renderer

Three steps in rendering a cloud shaped like a bunny. First create the point cloud. Then simulate light scattering through the point cloud. Finally render an image using the two point clouds

Generating the point cloud

The point cloud is generated using a standard ray-marching algorithm, adding points to the point cloud at each step. For each point, a color representing the attenuated light Ci_i, is calculated and stored, along with a vector, L_i, representing the prominent light direction, a Mie to isotropic scattering ratio R_i, and a density value D_i. Multiplying the density by a fractal pattern adds noise to the cloud. The image on the left side of Figure 4 shows the resulting point cloud. Points are colored with their initial value. At this point, only direct illumination from the light source is recorded in the point cloud and only the back edge of the bunny is illuminated directly by the light source. The other points remain dark until multiple scattering passes are computed.

Figure 5 depicts the creation of two points at different depths relative to the cloud surface and the calculation of their initial color and prominent light direction. For each point, the initial light contribution from the sun attenuated for distance, a prominent light direction, a Mie/isotropic ratio and a density are stored. The point on the left is closer to the surface, so less of the light is attenuated. The point on the right is farther from the surface so some of the light from the source is attenuated before it reaches the point. The Mie distribution is oriented parallel to the incoming light direction.
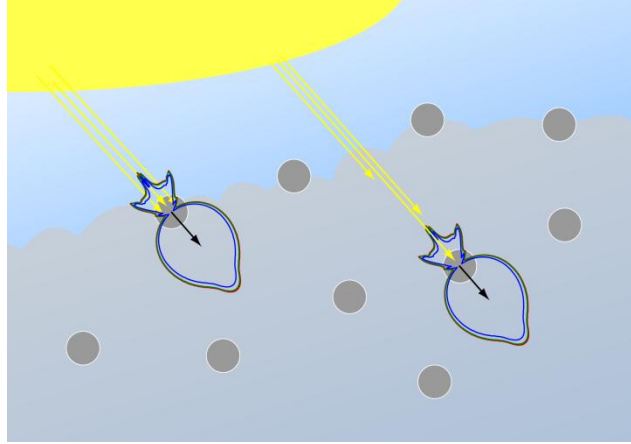
16

Figure 5 Creation and initialization of two points in a cloud.

The point nearest the surface receives more light initially.

Simulating light scattering

Scattering of light is simulated using a main function and a worker function.

Main Function

After the point cloud and Mie data have been read in the main function enters a loop. Each loop is one pass of the algorithm. For each pass, the main algorithm initializes a set of worker threads then waits for the threads to complete the work. Once each thread has finished, the main function will update the point cloud for the next pass by updating the ratio value R_i for each point. R_i represents the ratio between purely Mie scattering (1.0) and purely isotropic scattering (0.0). R_i is updated using the vector L_{sum}, which represents the summed intensity and direction of scattering events over one pass of the algorithm. R_i is updated using the equation:

$$R_i = R_i - R_i \cdot e^{-|L_{sum}|} \hspace{3cm} \textbf{Equation 1}$$

The justification for this approach is that each point starts with a prominent light direction - all of the light scattering through that point comes from the sun, and therefore one direction. But as the

point receives scattering events from other points the light scattering through the point becomes more isotropic - not that the individual scattering events are isotropic, but that since there are so many Mie scattering events coming from all directions, the overall effect is isotropic. The same simplification is made in other work (Haber 2005) involving Mie scattering in the atmosphere although we smoothly vary from Mie to isotropic scattering rather than switching to isotropic scattering after the first bounce as in (Habel 2005). If the point is receiving scattering events in primarily one direction then the summed length of L_{sum} grows, and R_i stays closer to 1.0 (anisotropic scattering). If however the point receives scattering events from all directions, the scattering vectors summed in L_{sum} will cancel each other out, L_{sum} becomes shorter and R_i will move toward 0.0 (isotropic scattering).

Figure 6 represents this transformation, with the Mie function depicted in the upper left corner, and the isotropic function depicted in the lower right. As L_{sum} approaches zero, the scattering function transforms into the isotropic function shown in the lower right.
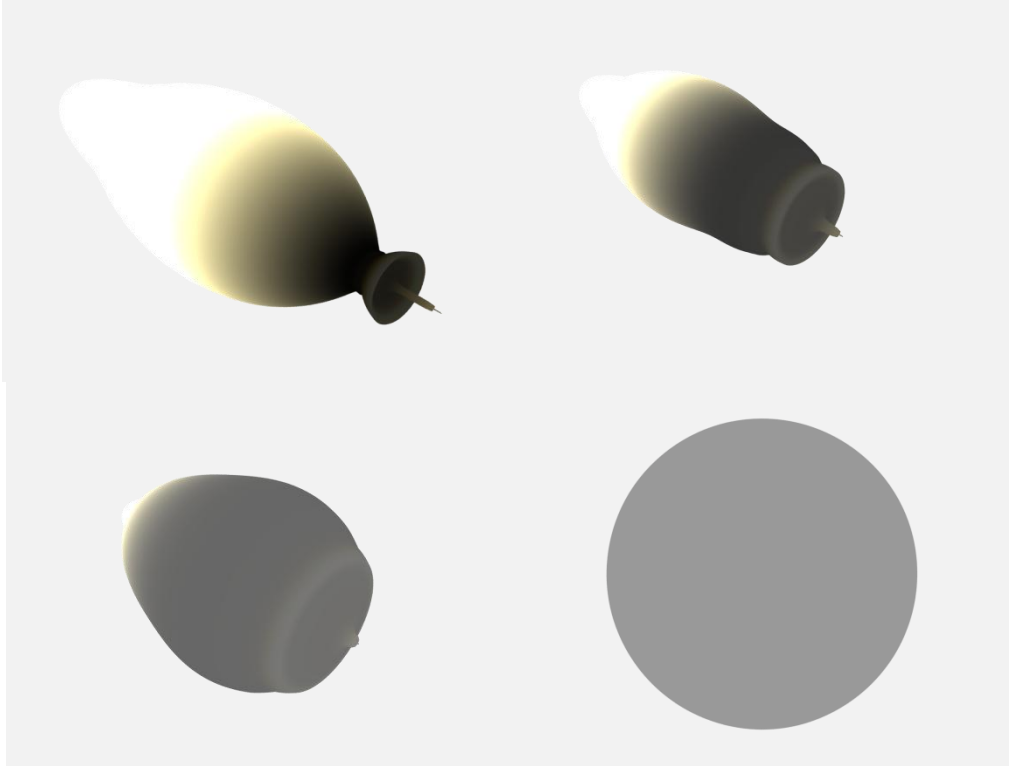
Figure 6 Mie to Isotropic Scattering

After R_i has been updated, L_{sum} is normalized and copied to L_i. The light contribution stored in C_o is added into C_i, and C_o is reset to zero.

After the main loop is finished the data is written out to an output point cloud. At this point each component of the output color C_o is thresholded to fall within the range 0 to 1.

$$C_{o(red)} = 1.0 - e^{-C_{i(red)}}$$

$$C_{o(green)} = 1.0 - e^{-C_{i(green)}}$$

Equation 2

$$C_{o(blue)} = 1.0 - e^{-C_{i(blue)}}$$

Worker Function

The main part of the scattering algorithm is $O(n^2)$ where n is the number of points, and is calculated using two nested loops. We limit the range of point-to-point scattering. This does not improve the $O(n^2)$ asymptotic bound but does reduce the number of scattering events that need to be computed. One of the major speedups of this algorithm is to limit the scattering calculations to and from a point to a local neighborhood of points. At the start of each interior loop, the worker thread will first calculate the distance between the two points. If the distance is greater than a threshold it skips any further calculations and moves on to the next iteration. This speedup does not significantly affect the results because as the distance between two points increases, the attenuation between the two points based on distance makes the light scattered between the two points become negligible. This is similar to the use of a smoothing kernel with limited support radius in smoothed particle hydrodynamics (Muller 2003). The worker will do the same with the light attenuation value for the distance between the two points. If the attenuation due to distance A_d between the two points P_{src} and P_{dst} is below a certain threshold the scattering event is viewed as negligible and is ignored.

The attenuation due to distance is calculated using the distance between the two points, the average of the densities D_i of the two points, and the extinction coefficient σ_e as defined in (Bouthors 2008). The extinction function includes extinction of light due to the water molecules within the cloud volume. The extinction coefficient σ_e effectively scales the exponential function, controlling how fast the light falls off due to distance.

$$A_d = \ e^{-|P_{src} - P_{dst}| \cdot (D_{src} + D_{dst})/2 \ \cdot \sigma_e} \qquad\qquad \textbf{Equation 3}$$

After calculating the attenuation due to distance A_d between the two points, we calculate the attenuation due to Mie scattering A_m. A_m is calculated using the Mie function and the phase

angle Θ, or the angle between prominent light direction L_i and the normalized vector from P_{src} to P_{dst} ($\frac{P_{src}-P_{dst}}{|P_{src}-P_{dst}|}$). We use the phase angle thus obtained as a lookup to the Mie function to get A_m.

Finally we calculate the light scattering event from P_{src} to P_{dst} as follows:

$$Co_{dst} = A_d \cdot (A_m \cdot R_{src} + (1 - R_{src}) \cdot Ci_{src} \qquad \textbf{Equation 4}$$

$$L_{sum\ dst} = A_d \cdot (A_m \cdot R_{src} + (1 - R_{src}) \cdot L_{src} \qquad \textbf{Equation 5}$$

where Co_{dst} is the light scattered from P_{src} to P_{dst}, and is calculated by attenuating the incoming light by the ratio of isotropic and anisotropic light and the distance between the two points. $L_{sum\ dst}$ is the summed prominent light direction of the destination point and is calculated similarly to Co_{dst}. At this point, calculate the opposite scattering event from P_{dst} to P_{src} using the same process. A_d can be reused, but A_m is recalculated.

Figure 7 illustrates the worker algorithm in three different arrangements of source scattering points for a single target point. Each point has a prominent light direction L_i shown as a black vector, and a Mie/isotropic function shown as an oriented Mie scattering plot. The point receiving light also includes a light blue vector that represents $Lsum_i$ and which is updated after each scattering event. In the first scattering event, (a), the target is located far away from and to the side of the source. In this case, the source contributes little light to the source due to attenuation and directional Mie scattering. In (b) the source and target are closer but the target lies outside the region of strong forward scattering for the source. In this case, the source's contribution, represented by a small light blue vector, is added to the contribution from (a). The source in (c) is close to the target and the target lies in the direction of strong forward scattering. The contribution from this source is added to the contributions of the previous two.

21

Similar scattering events are computed and included for other points near the target but not shown in the figure. The final scattered light vectors are shown in (d). The direction of the sum is the new prominent direction of the target. The length of the sum is used to set the scattering ratio for the target. If the sum of the vectors is long, then the target receives light primarily from one direction and Mie scattering is stronger than isotropic scattering at the target. If the sum of the vectors is short, then the target may be receiving light from many directions and isotropic scattering is stronger than Mie scattering at the target. The rationale for this decision is that simultaneous Mie scattering in many directions is nearly isotropic. The process is repeated for each set of points in the range. If the thread completes work for the given range, it will ask for a new range until the whole pass has been completed. At this point the threads exit and control returns to the main function. For the next pass the threads are respawned.
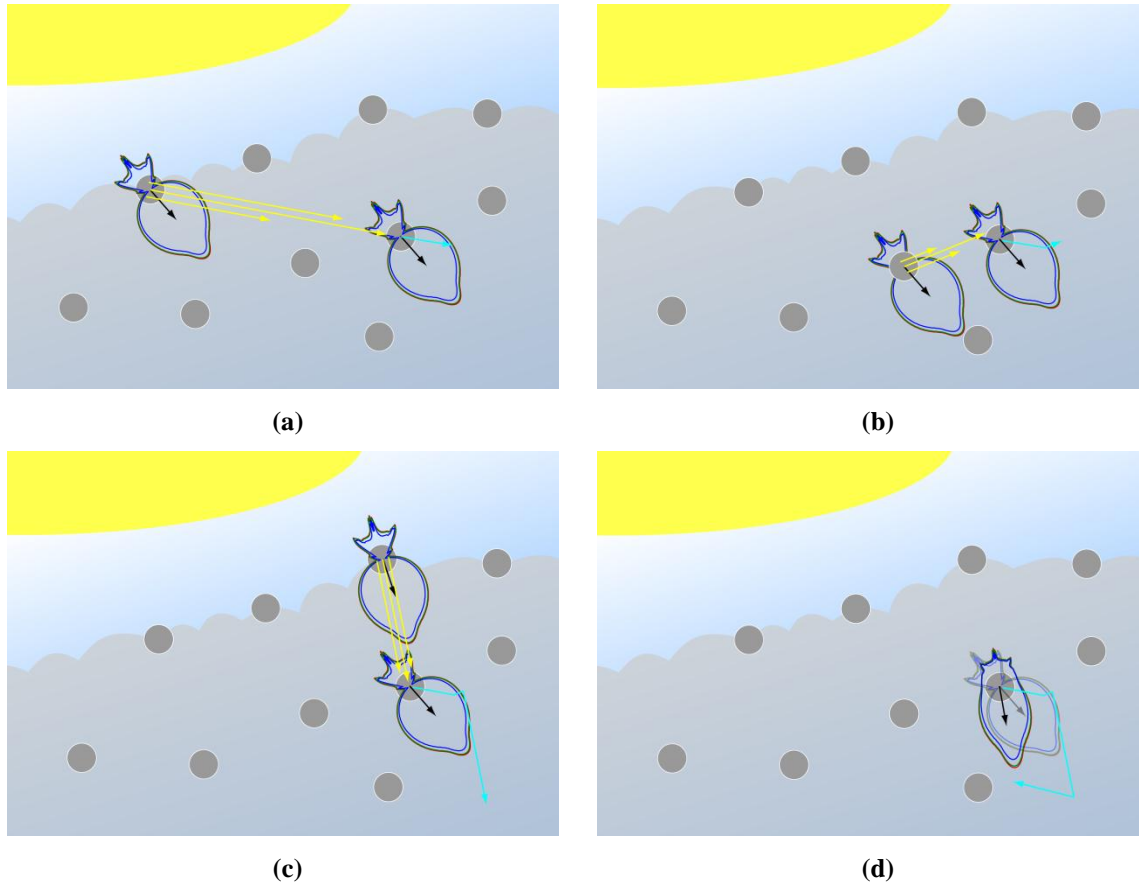
Figure 7 Accumulation of light scattered from three points to a nearby point.

Final Rendering

After computing and storing light scattering within the cloud volume, the illumination data in the point cloud is converted to a brickmap for final rendering. Final rendering is based on a ray marching algorithm. The ray marcher steps through the volume of the geometry, sampling data from the brickmap and the initial point cloud at each step. From the brickmap we sample the output light value Co_i along with the prominent light vector L_i, the density D_i, and the Mie to isotropic ratio R_i. From the input point cloud we sample the initial light value Ci_i, so that we do not have to recalculate the attenuated light contribution from the sun.

Using this data we first compute attenuation due to Mie scattering A_m using L_i and the incident vector L_o,

23

$$\Theta = \cos^{-1} L_i \cdot L_o$$

<div align="right">**Equation 6**</div>

$$A_m = Mie(\Theta)$$

and the attenuation due to distance A_d using ℓ_i, the distance from the surface to the current point P_i, and the extention coefficient σ_e.

$$A_d = e^{-\ell_i \cdot \sigma_e}$$

<div align="right">**Equation 7**</div>

From these we calculate total luminance Λ_i at the point using A_m, A_d, and a constant that scales the isotropic scattering A_{iso}. This equation attenuates a light value by the Mie function and an isotropic value as determined by the ratio R_i, adding the initial contribution from the sun Ci_i, and attenuating the sum of the values by the distance from the surface in the incident direction. By doing this we make sure to include contributions from the sunlight and from light scattered through the cloud.

$$\Lambda_i = ([A_m \cdot R_i + A_{iso} \cdot (1 - R_i)] \cdot Co_i + A_m \cdot Ci_i) \cdot A_d$$

<div align="right">**Equation 8**</div>

The volume covered by each step is summed using trapezoidal integration. To this final result we add an ambient term representing light scattered through the cloud from the sky. The final results are written out to an image file.

Chapter 4: Results

As is the practice in graphics papers on rendering in general, and on rendering clouds specifically, we subjectively demonstrate the quality of our algorithm by including rendered images. We also objectively demonstrate the quality of the algorithm by showing cloud-specific effects like the glory, fogbow, strong contrast between the lit and unlit sides of the cloud and the silver lining.

Although our algorithm is independent of the geometric model, we chose to use the Stanford bunny geometry as our model. The thin ears and the thicker body allow us to test and view different cloud features without switching between models. Although the original geometry is modeled using many polygons, this did not significantly affect computation times because we render from a point cloud. The algorithm was also used to render other geometry such as the Utah teapot and a Maya particle system rendered as a blobby-surface. First we give some images that illustrate the different components of the algorithm then we give images that demonstrate cloud-specific rendering effects.

The unique and defining feature of this method is how it makes use of the Mie function. We used a ratio value R_i that would control how isotropic the scattering was at a point. Figure 8 shows a screenshot of the bunny point cloud colored by R_i. The darker values represent isotropic scattering, while lighter values represent anisotropic scattering. In this view, the light is coming from the right side to the left. As expected the lit side is darker – more isotropic – while the unlit side is lighter – more anisotropic. This is because while the points on the lit side receive both forward scattering and back scattering events, the points on the unlit side receive only forward scattering events. The combination of forward and backward scattering events on the lit side makes the overall effect more isotropic. The lighter (anisotropic) band around the middle of the

25

bunny also fits expectations. The Mie function is strongly forward scattering with some back scattering. The side scattering is very weak. Thus the points around the middle remain more anisotropic because the only scattering events they receive are from the side and are significantly weaker. Other variations are due to differences in the density – less dense particles make for stronger scattering events.
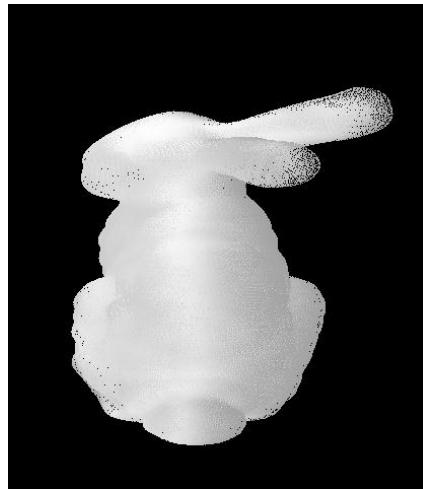

Figure 8 The values of R_i after the scattering passes have completed.

Darker values are more isotropic, while lighter values are more anisotropic. Notice the strong anisotropic (Mie) scattering around the middle.

We can verify that the scattering process results in visually significant effects by viewing renders based on point clouds that have been through varying numbers of passes of the scattering algorithm. Figure 9 shows renderings of a single scene in which the number of scattering passes increases. Increasing the number of scattering passes increases the distribution of light through the cloud.
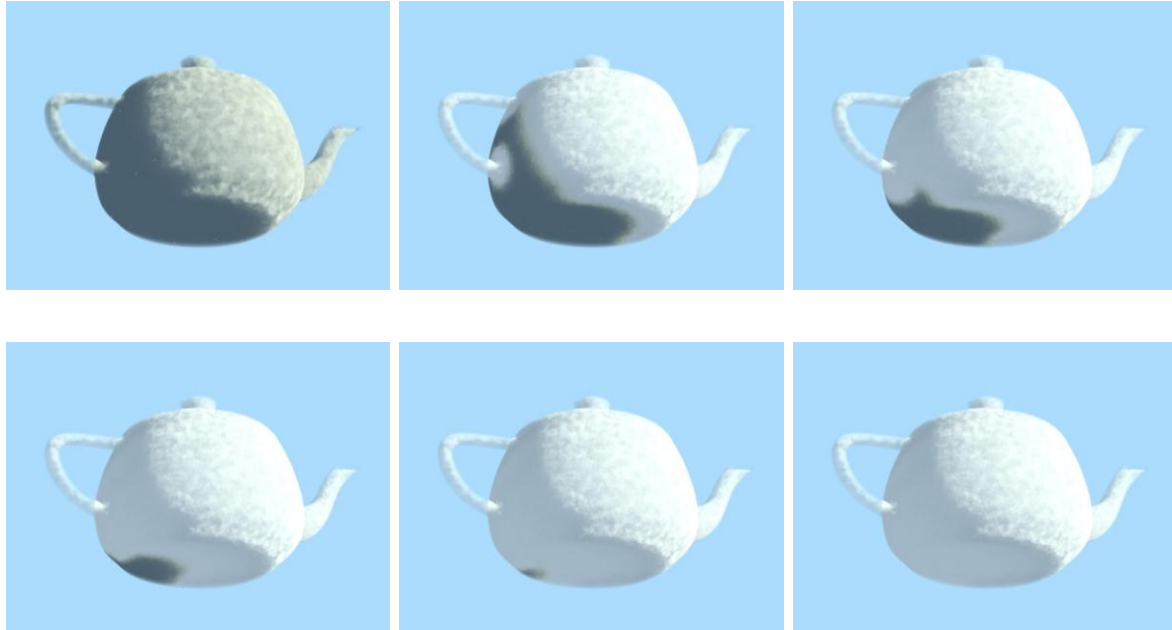
Figure 9 Renderings of the scattering process.

The top left image was rendered using a point cloud before the scattering simulation had been run. The bottom right image was created using a point cloud created after all of the scattering passes had been computed. The middle images form the intermediate steps.

Figure 10 shows the bunny, teapot and blobby surface models rendered to mimic sunset and night lighting. As long as the incoming light angle remained constant, the view angle and the input light color could change without having to recompute scattering in the point cloud. The teapot and blobby-surface point clouds were modified by moving the input points using a fractal pattern, producing a softer look to the clouds.

Figure 11 (a) illustrates the glory effect generated using wavelength dependent anisotropic scattering using the Mie function. The effect is due to strong Mie scattering at the lit surface of the cloud and occurs when the view angle is oriented in the same direction as the incoming light.

Figure 11 (b) shows the silver lining effect, which is due to thinner and less dense areas of the cloud. The bright ears in the image are an example of thinner areas, while the lighter spot in the

lower right is due to a less dense volume of the cloud. Also interesting is Figure 11 (c) which shows the strong contrast between the lit side of the cloud and the unlit side.

Figure 11 (d) was created by using the same input settings as the bunny cloud, but with different geometry. The geometry was created by taking a blobby-surface, defined by a particle system, and converting that surface into a polygonal piece of geometry.
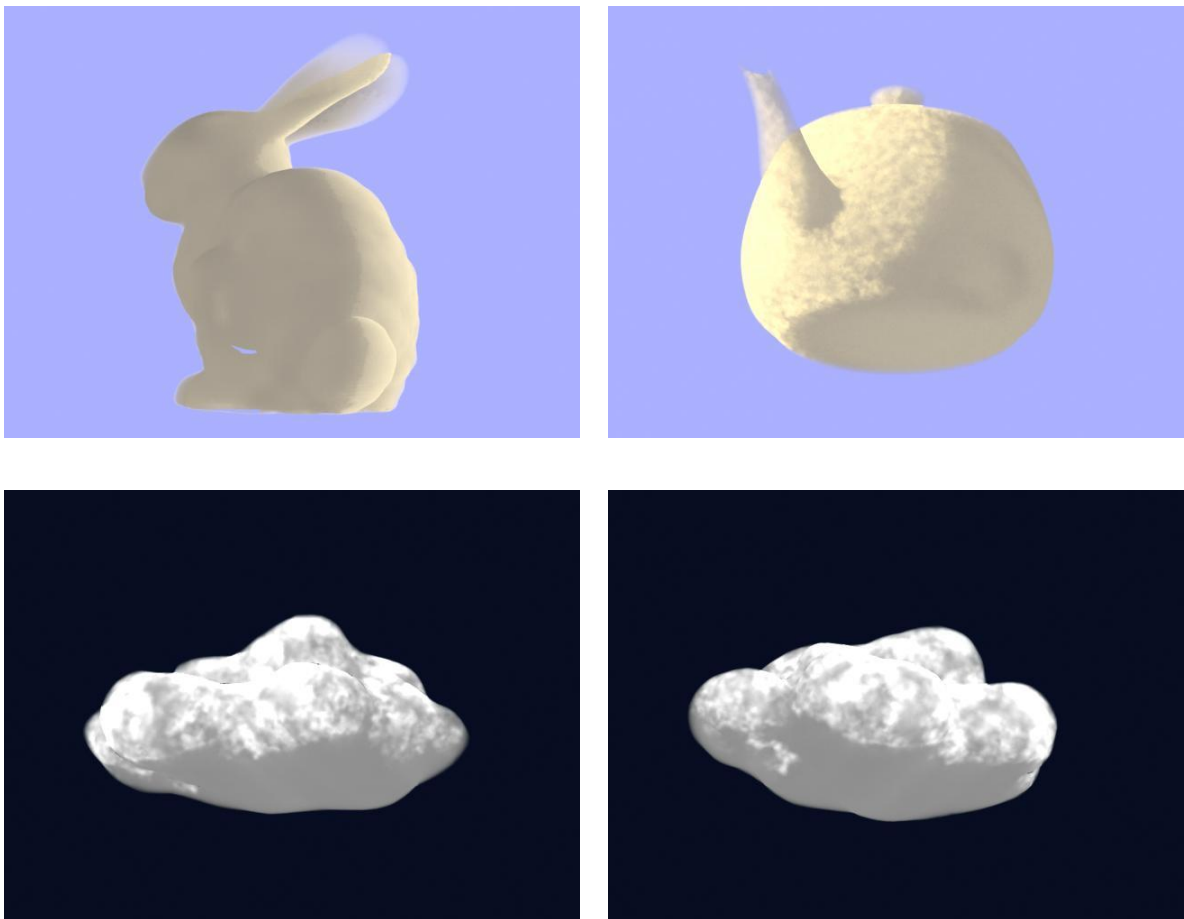
Figure 10 Clouds rendered with different lighting colors

www.manaraa.com
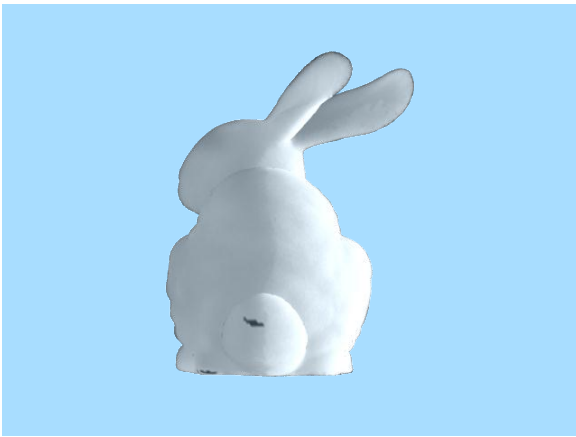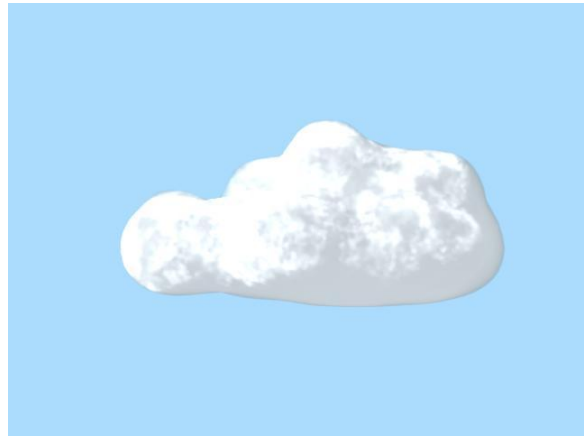
Figure 11 Clouds rendered with different features

Rendering of the clouds showing off some of the cloud features including the glory (a), the silver lining (b) and the strong contrast between lit and unlit sides of the cloud (c). Also included is a rendering with the same input settings for different geometry (d), and clouds with different input (e), (f).

Chapter 5: Analysis / Discussion

The main contribution of this work is an algorithm for simulating light scattering through a volume of water vapor droplets. While there is room for improvement in the implementation, the ideas of simulating light scattering events using a combination of the Mie function with isotropic scattering are unique and reproduce realistic cloud phenomena. The primary weakness of the algorithm is the final ray marching pass that accumulates results from the point cloud to produce the final image. Tuning the algorithm proved to be very difficult due to how this final pass was implemented. Other problems encountered in this phase include dealing with holes in the geometry (notice the artifacts in the bunny geometry), concave and overlapping surfaces (noticeable in the bunny and teapot clouds in Figure 10), and transparency. One of the major features of clouds is fuzzy edges, which feature was not fully integrated into this work due to time constraints and difficulty working with the final pass.

As is common for volume rendering algorithms (Apodaca 2000) our method can be very sensitive to the input parameters. A small change in the input can greatly affect the results. The variables that have the greatest effect on the output are the extinction coefficient σ_e, the input light values, the scalars for the Mie scattering contribution, the isotropic scattering contribution and the actual size of the geometry (distance between points and number of points). While it is possible to tune some of the variables independently, like the scalars for the Mie and isotropic scattering contributions (see Figure 12), other variables – the input light values, the extinction coefficient, and number of points in the point cloud – are interdependent and must be tuned simultaneously. Simultaneously tuning interdependent variables requires a long period of trial and error to obtain the desired results. While some parameter tweaking is expected with volumetric rendering, the process is complicated by the problem that one of the parameters (the

30

density of the point cloud) which controls render times also significantly affects the final image. Therefore the algorithm can only be fully tuned with the quality set high (and the render times longer). In practice we experienced render times of 2-10 minutes per frame. For a full quality final render this is acceptable, but for tuning this becomes very tedious.
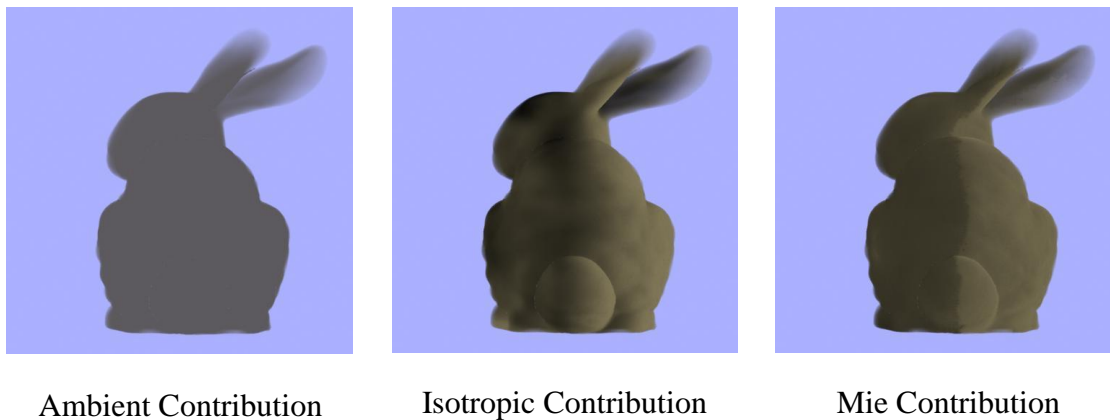

| Ambient Contribution | Isotropic Contribution | Mie Contribution |

Figure 12 The different contributions to the bunny cloud rendered separately

One consolation in the tuning process is that geometries of roughly the same size can be tuned the same way to produce roughly the same results. As an example, the input parameter settings for the bunny cloud image were used as a starting point for the teapot cloud and even the blobby surface cloud. There were some changes to the input parameters, but the changes were small compared to tuning the algorithm from scratch.

Although tuning variables can be time consuming the advantage is that more variables give the technical artist more control and flexibility. Having the ability to control light contributions from Mie scattering, isotropic scattering, ambient light, cloud density, and incoming light color and intensity means that the algorithm can be adapted for many different types of effects, as can be seen in the results above.

While it does have its drawbacks, our implementation of the final rendering pass does have its benefits, the greatest of which is that it is fully integrated into production 3d software. If needed

31

our method could be adapted into a Maya/Renderman production pipeline with minimal modifications. This also gives the benefit of being able to easily work with all types of geometry and lights. While we were implementing this method we were able to focus on the algorithm itself instead of having to deal with implementing point clouds or hand-generating geometry. Another benefit is that any type of geometry that can be used with Maya can be adapted to work with this method by converting it to an acceptable form. For this project we worked mainly with polygons, but we were easily able to use a particle system to generate new geometry when needed.

Another difficulty encountered with this algorithm is that in nature clouds have a very high dynamic range. The final pass of the algorithm proved so difficult to implement and tune because it needed to compress a large range of light values into a range of 0 to 1 expected by the renderer.

Future Work

The area of this algorithm that has the most potential for improvement is the final pass of the algorithm that generates the final image. For what it does it could be made much more efficient and much less temperamental to changes in input parameters. Another cloud feature that would really benefit this algorithm is the addition of fuzzy, semitransparent edges. Fuzzy edges can be a major feature of clouds, so adding them into this method would go a long way towards increasing realism.

One avenue of future research is to find a way to speed up the middle part of the algorithm where the scattering is calculated. The time requirement to calculate the scattering for a single cloud is one of the major drawbacks to this method being adopted for industrial use. It might also be interesting to see if some of the ideas from this method could be adapted to a much faster

32

billboard based rendering algorithm. Another possible area of research would be to integrate this algorithm with a sky rendering algorithm.

References

Apodaca, Anthony A, and Larry Gritz. *Advanced Renderman: Creating CGI for Motion Pictures.* San Diego, CA, California: Academic Press, 2000.

Bouthors, Antoine, Fabrice Neyret, Nelson Max, Eric Bruneton, and Cyril Crassin. "Interactive Multiple Anisotropic Scattering in Clouds." *Proceedings of the 2008 Symposium on Interactive 3D Graphics and Games.* New York, NY: ACM, 2008. 173-182.

Dobashi, Yoshinori, Kazufumi Kaneda, Hideo Yamashita, Tsuyoshi Okita, and Tomoyuki Nishita. "A Simple, Efficient Method for Realistic Animation of Clouds." *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques.* New York, NY: ACM Press/Addison-Wesley Publishing Co., 2000. 19-28.

Elinas, Pantelis, and Wolfgang Sturzlinger. "Real-Time Rendering of 3D Clouds." *Journal of Graphics Tools* (A. K. Peters, Ltd.) 5, no. 4 (October 2000). 33-45.

Haber, Jorg, Marcus Magnor, and Hans-Peter Seidel. "Physically-Based Simulation of Twilight Phenomena." *ACM Transactions of Graphics* (ACM) 24, no. 4 (October 2005). 1353-1373.

Harris, Mark J. "Real-Time Cloud Simulation and Rendering." *Doctoral Dissertation.* The University of North Carolina at Chapel Hill, 2003.

Harris, Mark J., and Anselmo Lastra. "Real-Time Cloud Rendering." *Computer Graphics Forum.* Blackwell Publishing, 2001. 76-84.

Henry, Jed. "Kites." Animated film produced by Brigham Young University Center for Animation. 2009.

Liao, Horng-Shyang, Jung-Hong Chuang, and Cheng-Chung Lin. "Efficient Rendering of Dynamic Clouds." *Proceedings of the 2004 ACM SIGGRAPH International Conference on Virtual Reality Continuum and its Applications in Industry.* New York, NY: ACM, 2004. 19-25.

Max, Nelson, Greg Schussmanq, Ryo Miyazaki, Kei Iwasaki, and Tomuyuki Nishita. "Diffusion and Multiple Anisotropic Scattering for Global Illumination in Clouds." *Journal of the Winter School of Computer Graphics (UNION Agency – Science Press)*, 2004. 2-6.

McGuire, Morgan, and Andi Fein. "Real-Time Rendering of Cartoon Smoke and Clouds." *Proceedings of the 4th International Symposium on Non-Photorealistic Animation and Rendering.* New York, NY: ACM, 2006. 21-26.

Muller, Matthias, David Charypar, and Markus Gross. "Particle-Based Fluid Simulation for Interactive Applications." *Proceedings of the 2003 ACM SIGGRAPH/Eurographics*

Symposium on Computer Animation. Aire-la-Ville, Switzerland: Eurographics Association, 2003. 154-159.

Perlin, Ken, and Eric M. Hoffert. "Hypertexture." *Proceedings of the 16th Annual Conference on Computer Graphics and Interactive Techniques.* New York, NY: ACM, 1989. 253-262.

Roden, Timothy, and Ian Parberry. "Clouds and Stars: Efficient Real-Time Procedural Sky Rendering Using 3D Hardware." *Proceedings of the 2005 ACM SIGCHI International Conference on Advances in Computer Entertainment.* New York, NY: ACM, 2005. 434-437.

Schpok, Joshua, Joseph Simons, David S. Ebert, and Charles Hansen. "A Real-Time Cloud Modeling, Rendering, and Animation System." *Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation.* Aire-la-Ville, Switzerland: Eurographics Association, 2003. 160-166.

Trembilski, Andrzej, and Andreas Brossler. "Surface-Based Efficient Cloud Visualisation for Animation Applications." *Proceedings of the 10th International Conference in Central Europe on Computer Graphics.* Darmstadt, Germany: Visualization and Computer Vision, 2002. 453-460.

Wang, Niniane; Microsoft Corporation;. "Realistic and Fast Cloud Rendering." *Journal of Graphics, GPU, & Game Tools.* (A. K. Peters, Ltd.) 9, no. 3 (2004). 21-40.

Appendix: Implementation Details

The implementation details for this project can be divided into three sections: tools, code, and rendering.

Tools

The results were generated on a quad-core Intel processor running the 64-bit version of Windows 7 Enterprise. This project's main algorithm was coded in C. The clouds were modeled and rendered using Autodesk Maya 2008 (64-bit) with the RenderMan for Maya plugin version 3.0.2 (64-bit). Videos were created using Adobe After Effects CS4.

Code

The code for this project was set up as a C++ application in Visual Studio. The project was set to compile with multi-threading enabled. In order to read and write to the point cloud files, additional Pixar RenderMan libraries were referenced by the linker:

- Program Files\Pixar\RenderManProServer-15.0\lib

- Program Files\Pixar\RenderManStudio-2.0.2-maya2010\rmantree\lib

- Program Files\Pixar\RenderManStudio-2.0.2-maya2010\lib

The actual library needed for the point clouds is called libprman.lib, which was listed as an additional dependency. Since we used the 64-bit version of the RenderMan software the Visual Studio project must be set to compile to a 64 bit platform.

After the project has been set up writing the code is straightforward. All that was needed to access the point cloud library functions was to include the file "pointcloud.h", which is located in the above referenced locations. Refer to the RenderMan documentation for information about the point cloud functions.

36

The input for the executable was set to be two strings, one the location of the input point cloud, and the other the location of the output point cloud. Later on in the development we added the ability to output additional intermediate point cloud files.

Depending on how the Pixar library is referenced, it may be necessary to include a copy of libprman.lib in the same directory as the cloud project executable.
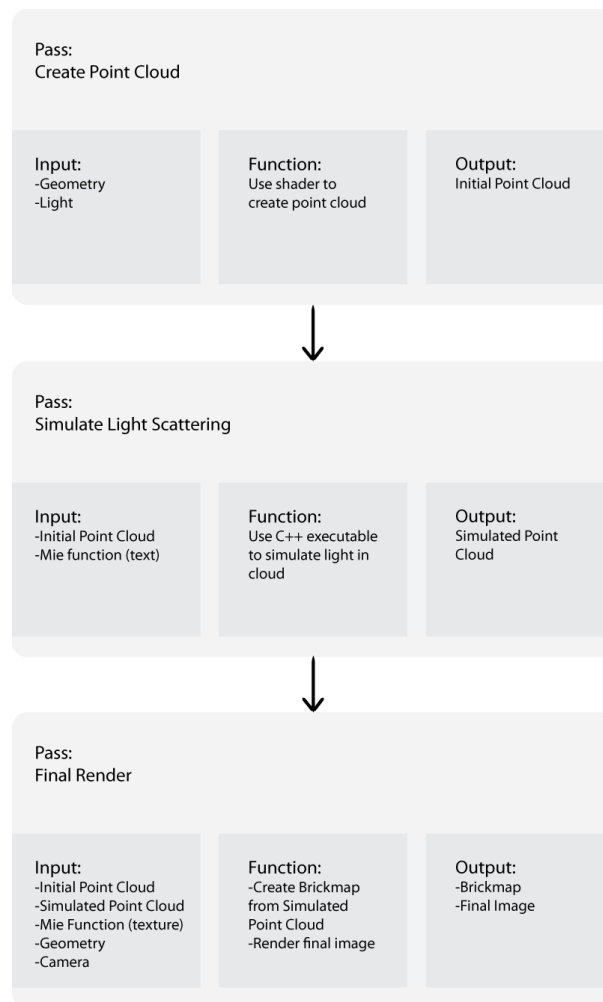
Rendering

Pass:
Create Point Cloud

Input:
-Geometry
-Light

Function:
Use shader to
create point cloud

Output:
Initial Point Cloud

Pass:
Simulate Light Scattering

Input:
-Initial Point Cloud
-Mie function (text)

Function:
Use C++ executable
to simulate light in
cloud

Output:
Simulated Point
Cloud

Pass:
Final Render

Input:
-Initial Point Cloud
-Simulated Point Cloud
-Mie Function (texture)
-Geometry
-Camera

Function:
-Create Brickmap
from Simulated
Point Cloud
-Render final image

Output:
-Brickmap
-Final Image

Figure 13 Algorithm Setup

The rendering portion of this project was implemented as a set of passes in Maya with two RenderMan shaders. The files were set up in the following folder structure:

37

- [project root]
    - data – contains the output point clouds and brickmaps
    - renderman
        - [scene name]
            - data – contains the point cloud and brickmap files generated by the Maya/RenderMan passes, generated automatically
            - images – contains the output images from the final pass, generated automatically
            - rib – contains the scene rib files created by Maya/Renderman, generated automatically
            - shaders – contains compiled scene shaders, generated automatically
        - …
    - rmantex – contains the pre-generated texture that represents the Mie function generated from the data produced by (Bouthors 2008), must be copied to this folder before rendering
    - scenes – contains the saved Maya scene file

With the "RenderMan for Maya" plugin installed Maya will automatically generate the above file structure. It remains only to copy the texture files into the correct location, and occasionally to copy the intermediate output files from the C++ executable.

Both RenderMan shaders were written using a simple text editor, and then compiled into RenderMan ".slo" shaders using the "shader.exe" tool that comes with the Pixar software. We imported these into Maya by creating two new RenderMan shader nodes, and pointing them to

the two ".slo" files. When the RenderMan shader nodes are created Maya automatically created "Shading Group" nodes, which will be modified later in the setup.

The next step in the setup process was to create a new series of RenderMan passes. The passes were modified to perform the three steps of our algorithm – create the point cloud, simulate light scattering, and render the final image. This version of RenderMan for Maya comes with a collection of predefined passes. For our project we created a new "SSMakeBrickmap" pass. By default this pass is used for rendering general sub-surface scattering effects, but we were able to adapt it to our purposes. The SSMakeBrickmap pass is composed of three parts, and each part can be configured separately. The first sub-pass is used to create a point cloud from the scene data and by default was named "rmanSSRenderPass". The screenshot in Figure 14 shows the modifications we made to the pass. In this screen the "sunShape" is a link to the main light representing the sun, and "readPointCloudSG" is a link to the shading group for the final pass shader, which causes only the geometry being shaded by the cloud shaders to be used when generating the point cloud. The field "input_cloud" is used to name the initial point cloud, which is generated by this pass and stored in the renderman/[scene name]/data directory. The "Caching Behavior" can be used to control when the pass will be run. If it is set to "Compute" then the pass is always computed. If set to "Reuse" then the pass will not run, and the previously computed data will be used. The "Caching Behavior" applies to all of the passes.
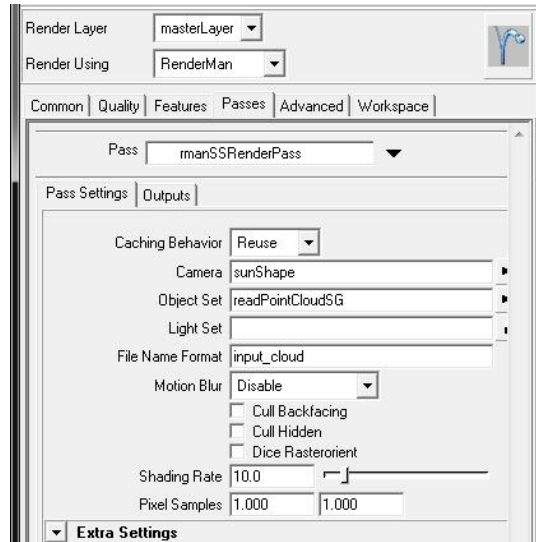
Figure 14 rmanSSRenderPass settings

The next subpass was named by default "rmanSSDifusePass". Normally it is used to compute

sub-surface scattering on the previously generated point cloud. For our project we adapted it to

call the light-scattering executable, as can be seen in Figure 15. The "Command" field was set to

be:

"…\cloud.exe" [passinfo rmanSSRenderPass filename] [passinfo this filename]

The command has the full path to the cloud.exe executable (not shown here). The "passinfo"

commands return the filenames for the "rmanSSRenderPass" and the current pass – which

become the input and output point cloud filenames for the light-scattering executable. Since we

changed the default command for this pass we were able to safely ignore the other pass settings
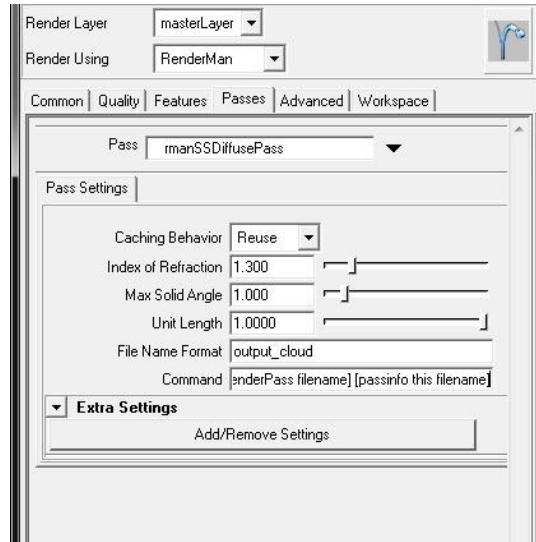
that no longer applied.

Figure 15 rmanSSDiffusePass settings

The final sub-pass was named "rmanSSMakeBrickmapPass", and accordingly is used to create

the final brickmap file using the output from the light-scattering executable. For this pass the

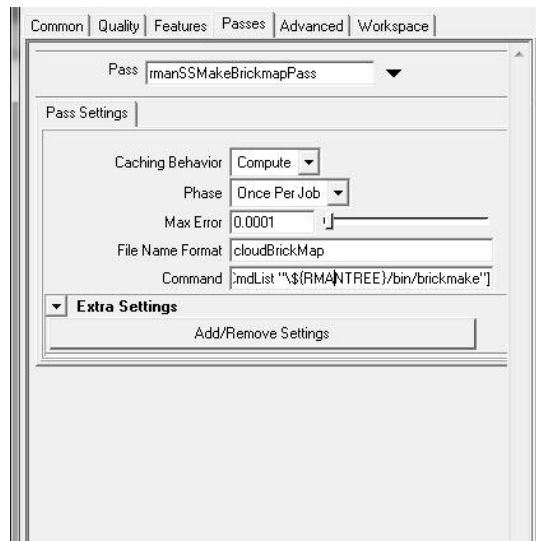default settings were adequate, as can be seen in Figure 16.

Figure 16 rmanSSMakeBrickmapPass settings

In order for the RenderMan renderer to properly create the initial point cloud file, it must have a

reference to the data variables. This was accomplished by adding a "Ri Injection Point". Under

"Default RiOptions MEL" we added the following lines, separated by semicolons:

- RiDisplayChannel "vector _L"

- RiDisplayChannel "color _i"

- RiDisplayChannel "color _o"

- RiDisplayChannel "float _d"

- RiDisplayChannel "float _r"

The final step in configuring Maya/RenderMan was to configure the shading groups for the custom shaders created earlier. We named the two shaders and the corresponding shading groups according to their functionality: "createPointCloud" and "readPointCloud", the former running during the first pass of the algorithm and the latter running during the last pass. No changes were needed for the "createPointCloud" shading group, but to the second we added an "Adaptor Controller", as can be seen in Figure 17. We set the adaptor to use the "createPointCloud" shading group whenever the "pass_id" matched "rmanSSRenderPass". This adapter causes the renderer to use the "createPointCloud" shader when rendering the first pass that creates the point cloud, and use the "readPointCloud" shader on the final pass.

With these settings it is trivial to add new geometry to the scene – simply attach the "readPointCloud" shading group to the geometry. When the scene is rendered the first pass will use the "createPointCloud" shader with the light representing the sun as the camera to generate a point cloud of the geometry. Next the light scattering executable will be run on the new point cloud. Finally, the output from the light scattering executable is converted into a brickmap and passed to the readPointCloud shader to create the final image.
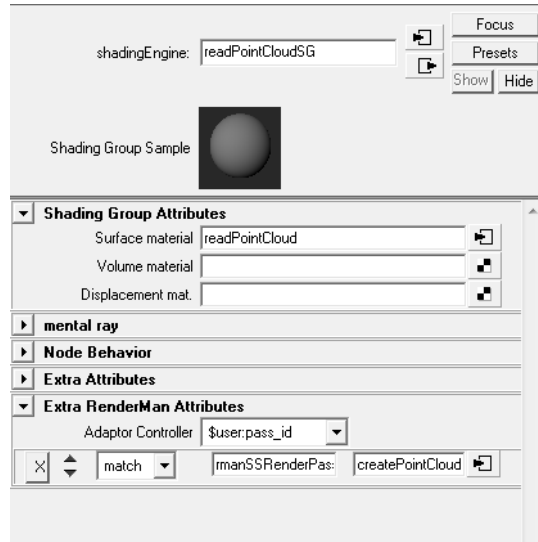
Figure 17 readPointCloudSG settings

The following tables list the inputs and outputs to the two shaders and the C++ executable.

Table 2 C++ Executable Input and Output

Input	String representing the path and filename to the input point cloud
	String representing the path and filename to the output point cloud
Output	The output point cloud
	Optional: point cloud files representing the different passes computed in the algorithm

43

Table 3 Shader createPointCloud Input and Output

Input	String "filename" representing the path and filename for the initial point cloud For our project set to: "[passinfo rmanSSRenderPass filename]"
	String "displaychannels" representing the names of the variables to be stored in the initial point cloud For our project set to: "_L,_i,_o,_d,_r"
	Float "scale" that adjusts the overall scale of the coordinate system
	Float "stepsize" used to control the step size of the ray marching algorithm
	Float "sigma_e" represents the variable σ_e
Output	Initial point cloud

Table 4 Shader readPointCloud Input and Output

Input	String "input_filename" which stores the path and filename of the input point cloud
	String "output_filename" which stores the path and filename of the output brickmap
	String "cloudDataPath" which stores the path to the Mie texture files
	Float "sigma_e" represents the variable σ_e
	Float "stepsize" used to control the step size of the ray marching algorithm
	Float "scale" that adjusts the overall scale of the coordinate system
	Float "mie_scale" that adjusts the Mie scattering contribution in the final image
	Float "iso_scale" that adjusts the isotropic contribution to the final image
	Color "sky" that represents the ambient light contribution to the final image
Output	Final image