Physically Plausible Weather Visualization

Gisle Nes

June 2008

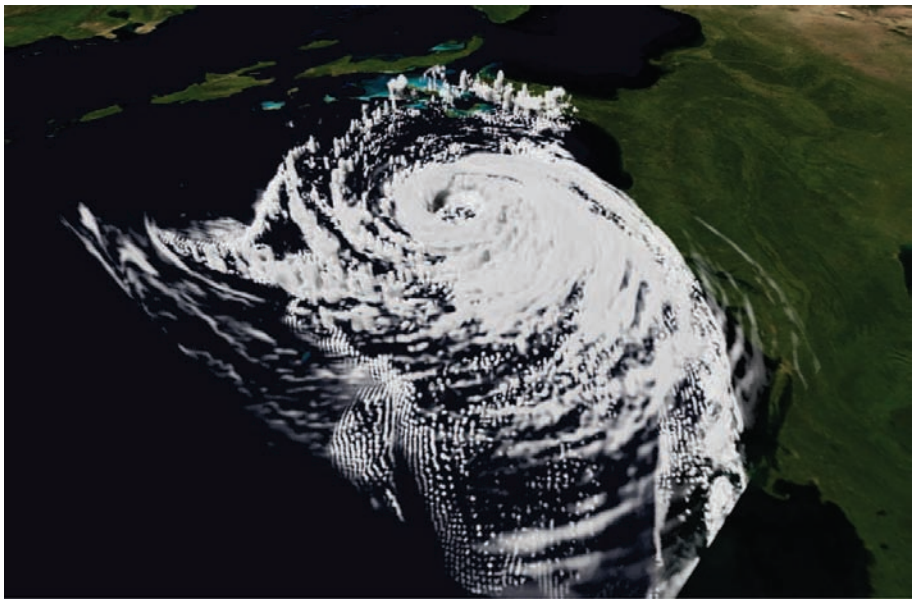
Master Degree Thesis

Department of Informatics
University of Bergen

Master Degree Thesis

Physically Plausible Weather Visualization

by Gisle Nes
June 2008



supervised by Ivan Viola

Visualization Group
Department of Informatics
University of Bergen

http://www.ii.uib.no/vis/teaching/thesis/2008-nes/

Abstract

Physically plausible visualizations of weather-phenomena such as hurricanes or cloud structures are difficult to achieve at interactive frame rates. The visual properties of clouds depend on how light is scattered through the media, as well as the amount of light that is scattered towards the viewer. The fact that all particles can scatter light in all directions makes the light calculation very complex.

Visualizations of weather phenomena are desirable for several applications, but this project has been carried out mainly with television broadcasters in mind. A big interest for television broadcasters is weather visualizations during the hurricane season. Because of the large interest for hurricane visualizations, the main scope of this thesis is to visualize simulated hurricane data sets.

This thesis presents a weather visualization system that utilizes knowledge about the physics of clouds in order to render physically plausible weather visualizations. The interaction between light and clouds is imitated by considering self shadowing and multiple forward scattering. The speed and parallelism of modern graphics hardware enables visualization of large weather data sets at interactive frame rates.

Contents

1 Introduction

Since the dawn of time people have had a special fascination for clouds. These amazing features that form the backdrop of our world can seem large and massive, yet light and peaceful as they drift silently across the sky. From the early days clouds has been seen as messengers for changing weather, and people have had a massive interest in trying to reveal their many secrets. Forming endlessly changing shapes, clouds have been to great inspiration and fascination due to their visual appearances. But not only their visual appearances make these phenomena so special. Being one of the main participants in the water cycle on earth, they are of tremendous importance for all life. Bringing water and life on one hand, and storms and destruction on the other, there is a clear contrast in the nature of clouds. In some situations they act as beautiful assets to the sky, while in others they bring horrendous disasters. A single scenario can even possess different properties at the same time when the context is changed. When observing a tropical storm from the earth, the massive destruction and overwhelming power is evident, while the same disaster can look peaceful and spectacular from the atmosphere. This is an aspect that makes people both respect and admire these fascinating phenomena.

The physical and visual complexity of clouds, and their uncontrollable nature has made them an interesting subject in science branches such as meteorology, physics, art and computer graphics. The physical complexity lies in their ever changing formations and patterns, while the visual complexity is caused by their gaseous nature. As Luke Howard so strikingly put it [22], their physics is *the sport of winds*, and the study of them is *deemed an useless pursuit of shadows*. What lies in this is that to fully understand the physics of clouds, one must predict the unpredictable nature of winds, and to study the visual properties an endless amount interactions between light and the media must be considered.

Based on this fascination, the main goal of this thesis arise - to capture the stunning beauty and dramatic appearance of clouds in a physically inspired manner.

1.1 Overview

This work presents a weather visualization system that renders physically inspired visualizations of meteorological data, at interactive frame rates. The intricate nature of clouds and other weather phenomena is hard to imitate in a physically plausible manner. In this work, approximations to the real world are implemented in order to achieve sufficient physical plausibility from what todays computer hardware can offer. The main scope of this thesis will be visualization of tropical cyclones, known as *hurricanes*. This choice has been made since hurricanes have both a beautiful and dramatic appearance, as well as the interest around these phenomenas is very high. A high interest leads to high availability of the required data, and a large target group. For broadcasters, hurricanes have a very high value as news features, and their application coincide with the goals of this thesis.

Cloud rendering can be seen as a essential problem in computer graphics. Rendering of semi-transparent matter is a highly important concern for many different applications, and due to its gaseous nature, clouds can be seen as an extreme case of such. Many other primitives like smoke, hair, or even skin does also to some extent possess optical properties similar to the ones of clouds. Parts of the theory presented in this thesis will therefore also apply to other visualization fields, such as medical visualization and seismic visualization.

The implementation of the proposed weather visualization system relies heavily on graphics hardware. With the large amounts of data that represent the hurricanes, and the high complexity in light calculation, the performance of modern graphics hardware is exploited in order to achieve interactive frame rates with high quality renderings. Figure 1.1 shows a rendering of hurricane Isabel, which was the costliest and deadliest hurricane that hit the United States in 2003. Figure 1.2 shows hurricane Katrina that struck the United States in late August 2005. Katrina was known as the costliest hurricane in the history of the United States.

1.2 Problem Statements

In *direct volume rendering* (DVR), illumination is an important aspect. The way lighting is considered affects the result in a great deal, and the lighting method that is used should be a concern when aiming to visualize a volume. The most simple way to imitate lighting in DVR is to use a *unshaded* approach. With such an approach, all surfaces will emit light similarly, without any concern on reflection angles or shadowing. This method may be desirable for many illustrative purposes, but it does not yield photo-realistic results. In polygonal computer graphics, it has been common to use the normal vector of each surface to calculate the reflection angle of light. In direct volume rendering, there are no given explicit surfaces with normal vectors. Therefore, the normal vector can be approximated by a normalized gradient vector. This will give *shaded* images using DVR.

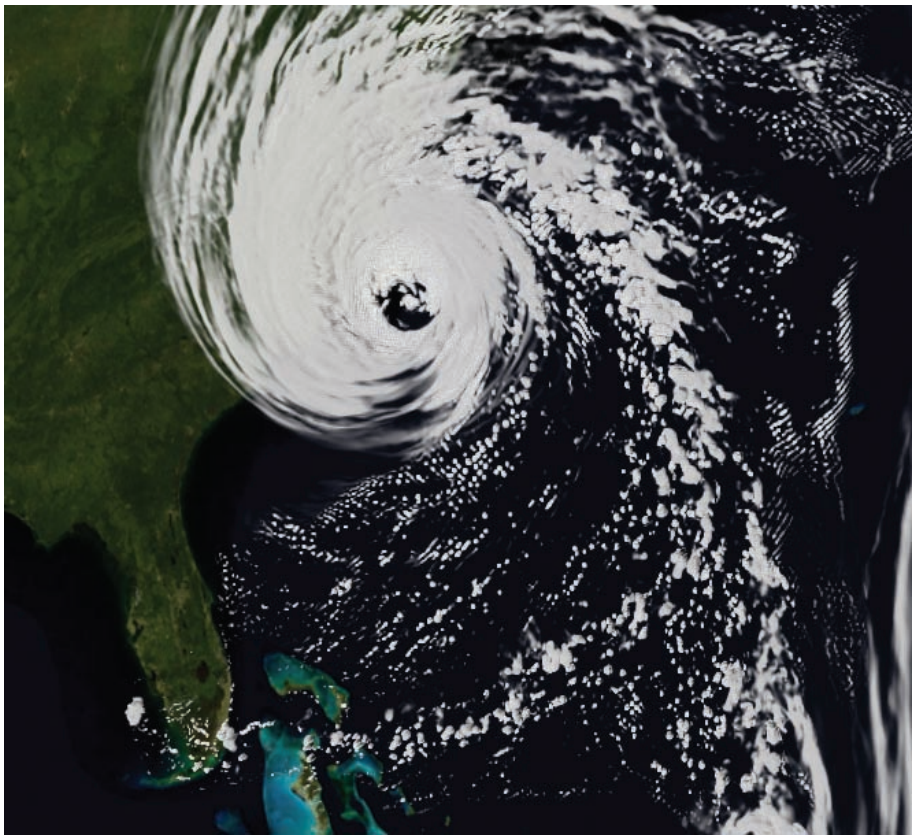


Figure 1.1: *Image rendered with the proposed weather visualization system, showing hurricane Isabel from the atmosphere.*

When it comes to gaseous matter, shaded DVR will not give visually accurate results. Without any solid surfaces, the reflectivity is decided by the interaction between light and particles in the matter. This implies that the light can be scattered in any direction, not only a fixed angle from the surface, and thus it can also be scattered multiple times through the volume. Another aspect that also appears when working with gaseous and translucent matter is the effect of partial occlusion. Partial occlusion occurs when a part of the volume is occluded by matter that is not fully opaque. This can be illustrated by the effect of sunglasses, they partially occlude the light that is directed towards the eye. When multiple scattering and partial occlusion are to be considered in DVR, the entire light propagation through the matter has to be calculated. Since clouds and hurricanes that are within the scope of this thesis can be considered gaseous matter, an approach that account for these factors needs to be implemented.

Figure 1.2: *Image rendered with the proposed weather visualization system, showing hurricane Katrina.*

The main goal of this thesis is to render physically plausible weather visualizations. In order to achieve this, the following statements are aimed to be solved:

- approximate the light transport in clouds in a physically plausible manner
- approximate the light scattering from clouds towards the viewer
- embed the volume in a realistic context
- visualize the change of the phenomena over time
- render the results at interactive frame rates

This thesis is based on previous work that has been done in cloud physics, as well as techniques that has been used for shadowing and illumination in computer graphics. The main goal is to extend the previously introduced deep shadow maps technique [34, 17] to utilize selected fundamental work that has been done in cloud physics [18, 45, 39].

1.3 Thesis Organization

The organization of this thesis is as follows. Chapter 2 will introduce some of the fundamentals of physics and light transport in clouds. This is in a large part based on work done by Nelson Max [35], Nishita et al. [39], and Riley et al. [45]. Next, Chapter 3 gives an introduction to previous work that has been done in the field of computer graphics and volume rendering. This will give an idea of how computer graphics can be utilized to visualize the fundamental light and cloud theory from Chapter 2. Chapter

4 will give a high level overview over how the weather visualization system presented in this thesis is built. This will briefly describe the steps that are taken in the proposed weather visualization system. Chapter 5 will shed further light on each step in the system, and describe the techniques in more detail. The described steps includes data acquisition, light transport calculation, direct volume rendering of clouds, and embedding in context. Chapter 6 will present the results and outcome of this thesis. This includes visual results, as well as a review of the performance that was achieved with the proposed system. Chapter 7 draws a summary of the work, and outlines further perspectives of the discussed field.

2
Fundamentals

As clouds and weather in general are complex physical phenomena, it is important to understand some of the underlying theory to be able to render physically plausible visualizations. In cloud rendering, there are two main aspects to consider. One is the dynamics and modeling of the cloud, which is important in the acquisition of cloud data. There are many ways to acquire cloud data, depending on the application in which it is to be used. Section 2.1 will give an introduction to selected methods that are widely used for cloud data acquisition or generation.

The other fundamental aspect is the light propagation in clouds, which covers the physics on how the cloud is illuminated. A lot of work has previously been done in this field. Section 2.2 will give an introduction to selected fundamental work that is required to understand the light propagation in clouds.

2.1 Weather models

The first step to consider in the weather visualization pipeline is the data acquisition and data representation of the clouds. There are many ways to represent cloud models, and they all have their applications. Depending on both the acquisition of the data and the final rendering, some methods are more suitable than others. For example, in some applications, such as news or weather forecasting, one may want to render actual clouds based on real weather data. In other applications like in the movie or game industry, the goal is to render realistic looking clouds that are not necessarily from a real scenario.

A common method for generating weather data is numerical weather prediction. By using gathered data from the current atmospheric situation as input to computer programs, it is possible to simulate and predict the coming weather. Because of the tremendous amounts of data and the number of different factors, the simulations require large amounts of computation power. These simulations are usually executed on some of the most powerful supercomputers in the world. The forecasts are calculated using mathematical equations for the atmospheric physics and dynamics. Since the equations are non-linear, simplifications have to be done. The WRF model [49] is one of the

widely used methods, and the one that is used as a basis for this thesis. WRF is a mesoscale lattice-based model, and the data sets are stored as voxel volumes. Being mesoscale means that the physical scale ranges from 5 kilometers to several hundred kilometers. This makes WRF a suitable model for hurricane data sets. A dataset consist of multiple variables at every voxel. The hydrometeor variables, that are important for this thesis, are stored as mass ratios. $\mathrm{H}_f = \frac{M_h}{Mass_{air}}$, where H_f is the mass ratio in the hydrometeor field f, M_h is the total mass of hydrometeors and M_{air} is the total mass of air at the current sample. Other variables are often also available in the data sets, such as temperature, pressure and wind speed.

Another widely used source for weather forecast data is Doppler radar data. This is a method that utilize Doppler radars to measure properties like reflectivity, wind velocity and turbulence. In order to get the optical extinction from the reflectivity variable, it is necessary to find a relationship between the reflectivity values and the particle concentration. This has previously been done using Ferrier's calculations [13].

For other applications such as games or movies, it is necessary to generate random clouds rather than reproducing a real scenario. Procedural noise has been widely used to generate cloud models. Using random noise, one can use different operations to generate random but continuous density [41, 32]. By filling up the volume with the generated density, one can get nice looking cloud models. Ebert et al. has worked extensively on cloud modeling using procedural noise [11, 9, 10].

2.2 Light Propagation in Clouds

In volume rendering, what we want to simulate is the interaction between light and the rendered matter. This interaction is what is called the volumetric light transport. The way a matter interacts with light depends on different properties of the matter. Gaseous materials interacts greatly in the volumetric light transport, and is what we call a *participating media*. A participating media is a media that scatter, absorb or emit light. In the case of this thesis the considered matter will be clouds, which consist of gaseous materials. Therefore, to be able to render physically plausible weather visualizations, it is important to understand the way clouds participate in the volumetric light transport. This section will describe important processes that takes place in the interaction between light and cloud particles. It will also introduce mathematical equations that in the end will result in an equation for the total light transport.

This section about light propagation in clouds is a compilation of fundamental referenced work [24, 35, 45].

2.2.1 Scattering and Absorption

When a photon of light hits a particle there are two ways in which the light and the particle can interact. Either, the particle can *absorb* the light, or the light can be *scattered* from the particle. When light gets absorbed, it means that the energy of the light is transformed to another form. The most natural case is that the light is transformed to

heat when it hits a particle. An easy way of noticing this phenomena is by walking on a dark pavement on a warm, sunny day. The other interaction that can happen between light and a particle is as mentioned, scattering. This means that the light hits a particle, and the light is then deviated from its original trajectory. When a light beam goes through a volume, parts of the light will hit particles in the volume and get scattered or absorbed. This means that the total amount of energy in the light beam will decrease as the light goes through the volume, and the light is attenuated. This attenuation is what we call the light *extinction*, and is given the name β. The amount of light that does not hit any particles on the way does not interact with the medium and is described as *transmitted* light.

2.2.2 Optical Properties

When discussing the volumetric light transport, we are considering spatial positions. The spatial position P is a three component vector (x,y,z) in the volume-space. In a volume, the interaction between light and the particles is affected by a number of factors. First of all, the number of particles at the current spatial position is important. This factor, the *material number density* η, is the concentration of particles at this position. The unit of this variable is given as inverse length m^{-1}, hence it gives the number of particles per unit length. This factor can be collected a number of ways, this will be further introduced in Section 2.2.3. The extinction cross section σ_{ext} is the next factor. It is given as the sum of the absorption cross section σ_{abs} and the scattering cross section σ_{sca}. These factors give the fraction of light incident to the particle that is absorbed or scattered respectively. The last factor deciding the interaction is the phase function. This is the function that gives the amount of scattered light in every direction, as described in Section 2.2.5.

In volumes with high particle concentrations, it is very time consuming to consider the scattering and absorption for every single particle. Instead we use coefficients that hold for all particles in a given subset of the volume. We define the *scattering coefficient* β_{sca} as $\sigma_{sca}\eta$ and the *absorption coefficient* β_{abs} as $\sigma_{abs}\eta$. Similarly the *extinction coefficient* β_{ext} is defined as:

$$\beta_{ext} = \sigma_{ext}\eta \tag{2.1}$$

and represents the average extinction in the medium, caused by scattering and absorption. These coefficients give the probability of a particle getting scattered, absorbed or extinct respectively, for one unit length.

For ice and cloud, the *single scattering albedo* ϖ is assumed to be approximately 1 [26, 15]. From equations 2.2 and 2.3 we can then see that the extinction cross section is dominated by the scattering cross section in ice and cloud mediums.

$$\varpi = 1 - \frac{\sigma_{abs}}{\sigma_{ext}} \tag{2.2}$$

$$\sigma_{ext} = \sigma_{abs} + \sigma_{sca} \tag{2.3}$$

For hydrometeor particles that are substantially larger than the wavelength of light, the scattering cross section is assumed to be twice the geometrical particle cross section [33, 4]. The extinction coefficient for particles with radius R_p can therefore be defined as:

$$\sigma_{sca} = 2\pi R_p^2 \tag{2.4}$$

2.2.3 Particle concentration

In meteorological data sets the data is usually stored as mass ratios, the mass of hydrometeor particles over the mass of air (as described in Section 2.1). The usual hydrometeor fields are cloud, rain, ice, snow, graupel and vapor. In order to calculate the light propagation equations, we need to convert these mass ratios into particle concentration η. The particle concentration is described as the number of particles per volume of the considered field. Since we assume the volume of the air V_{air} to be substantially larger than the total volume of particles, the total volume is approximated to the volume of the air. The total mass of all particles in a given field M_t is [45]:

$$M_t = V_{air}\rho_{air}H \tag{2.5}$$

where ρ_{air} is the air density and H is the particle/air mass ratio. Further, we define the mass of a single particle M_p in the field:

$$M_p = V_{prt}\rho_{air} \tag{2.6}$$

where V_{prt} is the volume of a single particle. Knowing the total mass of all particles in the field and the mass of one single particle, we can find how many individual particles are in the field. This is done by dividing the total mass by the mass of a single particle. We want η which is the particle concentration per unit volume. This is calculated by again dividing the number of particles in the field by the volume of the field:

$$\eta = \frac{M_v}{V_{air}M_p} \tag{2.7}$$

2.2.4 Single and Multiple Scattering

Based on the scattering coefficient defined in Section 2.2.2 we know the mean path between each scattering event, β_{sca}^{-1}. In certain media, this mean path can be longer than the physical size of the volume. If this is the case, it implies that a photon can be scattered at most once on its way through the volume. This is what we call single scattering. A media in which light scattering can be approximated by single scattering

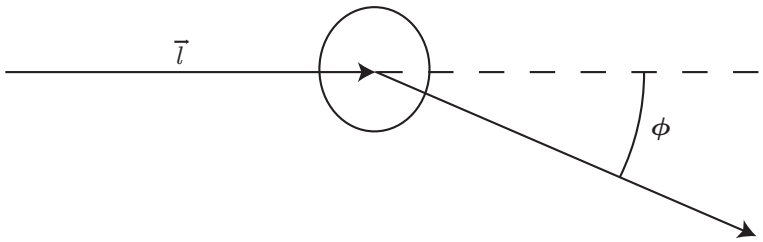


Figure 2.1: *The phase function gives the amount of outgoing light to any direction based on ϕ, the angle between the inward and outward direction.*

is called *optically thin*. An optically thin medium is a medium that is either physically thin or very transparent. This can be seen in clear air and steam over a cup of coffee. When the mean path between scattering events is smaller than the physical size of the volume, we have what is called multiple scattering. This means that each photon is scattered multiple times on its way through the volume which is the case in media that is *optically thick*. This can be seen particularly in clouds, where nearly all photons are leaving the volume after a number of scatterings. This causes the bright white appearance of clouds.

2.2.5 Phase Functions

When light is scattered from a particle, it may be scattered in any outward direction. Based on certain properties of the particles like size, shape and refractive index, the light is distributed unequally around the particle. A *phase function* is a function that describes the light distribution of the scattered light. Based on the inward direction of the light $\vec{l}$ the phase function gives the amount of light scattered in any outward direction $\vec{l}\,'$. An angle ϕ is used as input to the function, to identify the outward direction. This effect is illustrated in Figure 2.1. This function gives an average of the distribution that considers the most important features.

Rayleigh- and *Mie*-scattering [48, 36] are the two most common scattering phase functions used. The scattering from very small particles can be approximated well with Rayleigh scattering, developed by Lord Rayleigh. Scattering from small particles is wavelength dependent, this means that different wavelengths are distributed differently. This is what gives effects like the blueness of the sky, sunsets and also rainbows. The Rayleigh scattering in a hydrometeor field P_f is given by the following equation:

$$P_f(\phi) = \frac{3}{4}\frac{(1+\cos^2\phi)}{\lambda^4} \tag{2.8}$$

Here, ϕ is the angle between the inward and outward directions, and λ is the wavelength of the incoming light.

With particles that are substantially larger than the wavelength of light, we need another method for calculating scattering. Mie scattering is a model that gives good

approximations in such situations. This is a much more complicated function than Rayleigh scattering, and it is very computationally expensive to calculate. Because of the complexity of the function, approximations are usually used. A popular approximation to Mie scattering was developed by Henyey and Greenstein [21]. With the anisotropy factor g, the Henyey Greenstein approximation can be expressed as:

$$P_f(\phi) = \frac{1}{4\pi}\frac{1-g^2}{(1-2g\cos\phi+g^2)^{3/2}} \tag{2.9}$$

Since clouds consist of different types of particles, we often deal with more than one single hydrometeor field in the data set. The phase function depends on particle properties that may vary for the different hydrometeor fields. Therefore we need to use different phase functions for different hydrometeor fields. The final value from the phase function at a voxel P_v is then found by calculating the relative concentration of each field, weighted by the extinction coefficient of the field [26].

$$P_v(\phi) = \frac{\sum_{Fields} P_f(\phi)\beta_f^{sca}}{\sum_{Fields}\beta_f^{sca}} \tag{2.10}$$

where β_f^{sca} is the scattering coefficient of the hydrometeor field f, as described in Section 2.2.2.

2.2.6 Optical Depth

Optical Depth τ [35] can be described as the transparency of a volume sample. This is a function of the extinction coefficient β_{ext} of the hydrometeor field and the length of the sample. One way to imagine the effect of optical depth is to think of fog. When looking into fog, an object immediately in front of the eye has an optical depth of zero. As the object moves further away from the eye, the optical depth will increase. At a certain depth the object will no longer be visible because of the amount of fog between the object and the eye. This means that the optical depth is very large. The optical depth for one hydrometeor field τ_f is found by integrating the extinction coefficient at the current point in space over the length of the sample:

$$\tau_f = \int_{\vec{s}}^{\vec{w}} \beta_{ex}^{f}(\vec{s})ds \tag{2.11}$$

Clouds consist of many different types of hydrometeor particles. As explained in Section 2.2.2, the different hydrometeor fields will extinct the light differently. To be able to render all hydrometeor fields we will therefore need to define the optical depth in terms of all the different fields, τ_{af}. This is done by modifying Equation 2.11 to use the sum of all hydrometeor fields [26]. Numerically this can be approximated as:

$$\tau_{af} = (\sum \beta_{ext}^{field}(\vec{s}))\Delta s \tag{2.12}$$

The optical depth can be expressed as the light attenuation between two points $\vec{s}$ and $\vec{w}$. We can define the light attenuation, or *transmittance* T in terms of the optical depth τ, if we parameterize τ to take a point-parameter:

$$T(\vec{s},\vec{w}) = e^{-\int_{\vec{s}}^{\vec{w}} \tau(t)dt} \tag{2.13}$$

2.2.7 Light Transport

Until now, the different interactions that can happen between light and the participating media has been introduced. To be able to make use of this knowledge, it is important to know how to combine these different interactions. This way we will be able to calculate the total light transport through the medium. The factors that affect the light transport is in-scattering, out-scattering and transmittance. Figure 2.2 illustrates how these factors affect the light transport.

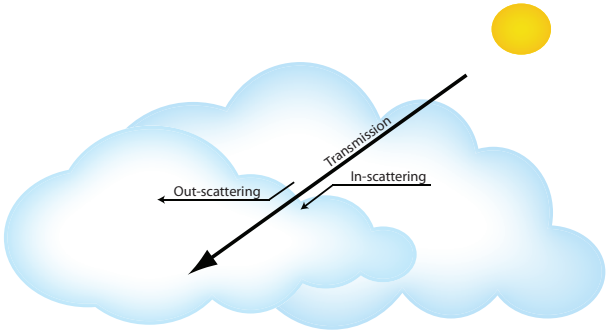


Figure 2.2: *Illustration of the light transport. In-scattering is light entering the path, out scattering is the light leaving the path, and transmittance is the light along the path that does not interact with particles.*

When light goes through a participating media, a series of interactions will happen. From one point in space the light intensity can increase, or the light can be attenuated. If the only interactions that happen are absorption and out-scattering, the light will be attenuated. But if the amount of in-scattering is larger than the sum of out-scattering and absorption, the light will be intensified at this point. This section will present the mathematics involved when the interactions scattering, absorption and transmittance plays together. As a result, an equation that represents the total light transport through the media will be given.

First of all we start with the transmittance of light through the medium. As previously explained, the transmittance is affected by the scattering coefficient β_{sca} and the absorption coefficient β_{abs}. Scattering and absorption coefficients could be combined as the extinction coefficient β_{ext}. The change of light intensity dI at point $\vec{x}$ over

distance ds in direction $\vec{w}$ can be expressed as:

$$\frac{dI(\vec{x},\vec{w})}{ds} = -\beta_{ext} I(\vec{x},\vec{w}) \tag{2.14}$$

which is the basis for the equation for transmittance, Equation 2.13.

When it comes to the in-scattering things get more complex. Now we have to consider the scattering from a range of points, and the direction of the scattering is also important. This is where the phase function comes into play. To consider all forward scattering to the current area we need to integrate over the entire sphere of incoming directions. The change of intensity dI because of in-scattering over the distance ds in direction $\vec{w}$ can then be expressed as:

$$\frac{dI(\vec{x},\vec{w})}{ds} = \beta_{sca} \int_{4\pi} P(\vec{x},\vec{w},\vec{w}') I(\vec{x},\vec{w}') d\vec{w}' \tag{2.15}$$

where $\vec{w}'$ is the incoming direction of the light, and P is the phase function that calculates the amount of light scattered to a given direction. Since this equation integrates over the entire sphere, it is very computationally expensive. Because of this, simplifications are usually done to approximate this equation.

Now we have equations for the absorption, out-scattering and in-scattering of light that passes through the media. These equations can be combined to give total change of intensity of light going over distance ds in a participating media:

$$\frac{dI(\vec{x},\vec{w})}{ds} = -\beta_{ext} I(\vec{x},\vec{w}) + \beta_{sca} \int_{4\pi} P(\vec{x},\vec{w},\vec{w}') I(\vec{x},\vec{w}') d\vec{w}' \tag{2.16}$$

As Nelson Max [35] pointed out, the extinction term can now be moved to the left side and be multiplied by $e^{\int_0^s \tau(t)dt}$ to get an equation for the light intensity of a ray at a specified point. By integrating the resulting equation over the ray between t=0 and t=$\vec{x}$, we can get the exiting light intensity at t=D. This is what Max' equation does:

$$\begin{aligned} I(\vec{x},\vec{w}) = {} & I(0,\vec{w}) T(0,\vec{x}) + \\ & \int_0^{\vec{x}} T(\vec{s},\vec{x}) \beta(\vec{s}) \int_{4\pi} P(\vec{s},\vec{w},\vec{w}') I(\vec{s},\vec{w}') d\vec{w}' d\vec{s} \end{aligned} \tag{2.17}$$

where I(0,$\vec{w}$) is the light intensity at 0 in direction $\vec{w}$, and T is the transmittance as defined in Equation 2.13. The first part in the equation represents the light directly in front of the eye as seen through the media, and is called the *extinction term*. This is because the term describes the extinction of light that goes from the back of the volume to the eye. In a fully transparent media, this would be equal to the light that is behind the media. The second part of the term represents the in-scattering. This is what is added from the participating media. When multiple scattering happens, light is reflected many times and finally some of it may reach the eye. This will result in a stronger light intensity as opposed to the extinction, and it causes an illumination of the

media. In clouds this is clearly visible since most of the visible color is illumination caused by multiple scattering. The extinction term and the in-scattering can be seen in Figure 2.2.

From Equation 2.17, we can see that by calculating the light intensity at a point $\vec{p}$, we also need to know the light intensity at multiple other points because of the scattering. Therefore this is a recursive problem. Because of the many variations that arise because of the double integral in the recurrence relation, this is a very computationally heavy problem. That explains the need for simplifications that will be further introduced.

3
Related Work

In the previous chapter, fundamental methods and mathematics regarding the physics of clouds was introduced. If we want to use this knowledge in the context of computer graphics, we also need to know some basics in the field of volume rendering. Volume rendering is an important part of computer graphics, and it deals closely with what has been explained about light propagation in the previous chapter. In volume rendering in general, what we want to approximate is the interaction of light with the participating medium. This makes previous work in volume rendering also very relevant to weather rendering, and certain elements can easily be mapped to the physics that was introduced in the previous chapter.

This chapter will give an introduction to some of the work that has been done in fields related to weather rendering. Basic rendering techniques for volumes will be described, as well as techniques for shadowing in semi-transparent media, and some approximations that has been done to the rendering equation introduced in Section 2.2.7.

3.1 Volume Representation

Voxels are very commonly used to represent volumetric data sets. The term voxel is a portmanteau of volumetric and pixel. As a pixel represents an area of a 2D image, a voxel represents an area of a 3D volume. A voxel volume is comprised of an equally spaced grid of voxels. Each voxel is addressed by a (x,y,z) coordinate. This can easily be compared to an RGB-cube where the coordinates go from 0 to 1 in each direction. Voxel volumes have the advantage that they easily can be rendered back to front and front to back. The overhead for each value is also small, since no information about the position of each value needs to be stored. This is an advantage for large detailed data sets, since it is quick to look up the values from the coordinates. For certain data sets this is also a disadvantage, since data is stored for the entire volume. This includes all empty space, and positions that could easily be calculated with equations in other models. Therefore, voxel volumes often contain relatively large amounts of data that is redundant. Voxel volumes are often used for medical scans like CT, MR

and ultrasound, but also weather data is commonly stored as voxel volumes [31, 53, 55, 5, 18]. For weather models, voxel volumes are mostly used for physically simulated data. Because of the structure of the voxel volumes, they are well suited to contain large and unstructured models that cannot easily be generated on the fly. As early as in 1984 Kajiya and Von Herzen [25] simulated clouds and stored them as voxel volumes.

Particle systems are simple models that store objects as a collection of spherical particles. Each particle can have different properties such as coordinates, color, texture and radius. Particle systems can be automatically generated, for instance where particles are emitted from a source. The position and other parameters can then be calculated based on the time each particle has lived, and on factors like gravity and applied force. Particle systems can also be made by hand using modeling tools. Particle systems are well suited for small-scale volumes, but for large scaled volumes with high detail they will require a very high number of particles. This may in many cases lead to a large number of approximately equal particles. This can be avoided using other methods like voxel volumes or metaballs. William Reeves introduced a method to model fuzzy volumes like gas and smoke using particle systems [44]. Later, Harris and Lastra has used particle systems to render static clouds [20].

Metaballs are n-dimensional objects that are defined as implicit functions. They were invented by Jim Blinn [2]. There are multiple approaches to render these objects, including Ray Tracing and splatting. By creating a number of metaballs that are placed in the shape of a cloud, one can add additional detail by generating more metaballs at the surface of the original ones [39].

3.2 Direct Volume Rendering

The term *direct volume rendering* is used for rendering techniques where we directly map every spatial sample to a color. Data sets used for direct volume rendering are usually discretely sampled voxel volumes, as described in the previous section. The color at every sample may be affected by multiple factors. In the simplest case, different density values in the volume could be mapped directly to different colors. Other possibilities are to calculate the color based on multiple values like density of different materials and lighting. The mapping between data values and the final optical property is done using a *transfer function*. This can be either a pre-calculated table or a piecewise linear function.

This section will introduce some of the most common methods used for direct volume rendering. It will also describe a GPU-accelerated method that will be used as the volume rendering technique in this thesis.

3.2.1 DVR Methods

There are multiple ways of rendering voxelized volumes. The most widely used direct volume rendering method is ray casting. In this technique, the renderer will cast a ray through the volume at every pixel of the rendered image. In a *front-to-back* approach

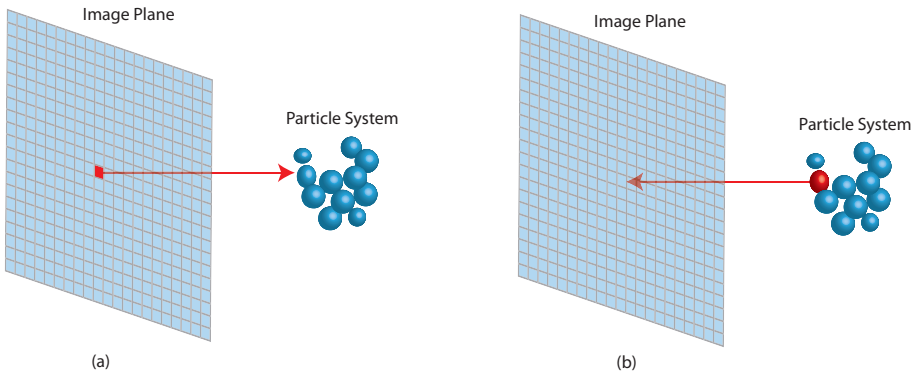


Figure 3.1: *The difference between image-order and object-order rendering. a) A pixel is rendered, and the appropriate volumetric elements are found. b) A volumetric element is rendered, and the appropriate pixels are updated.*

the rays are initiated at the eye point, and it is sampled at regular intervals through the volume. At every step through the volume, a value is fetched from the data set. This value is interpolated based on the current position of the ray in the volume. A transfer function is then used to map this value to a visual property. Then we update the current pixel, merging it with the color we got from the transfer function. We can also use ray casting with a *back-to-front* approach where the ray is initiated at the back of the volume, and is casted to the eye. This will however remove the possibility to use a optimization technique called *early ray termination*. Early ray termination is a technique that will stop the ray once the pixel is fully opaque, thus it will often require less samples for each ray. This is especially efficient in volumes where large parts are occluded, such that only a small part of the volume need to be traversed. In very translucent volumes however the effect is not as evident. Ray casting is an image-order rendering approach. This means that for each rendered pixel, the appropriate volumetric values are rendered. Figure 3.1 illustrates the difference between image-order and object-order rendering..

Ray casting is a relatively expensive rendering technique. Therefore there are other rendering techniques developed that trades the quality and simplicity of ray casting with performance. One such method is Shear warp factorization, developed by Cameron and Undrill, and described by Philippe Lacroute and Marc Levoy [30]. In this method an off-screen buffer is kept, which is axis aligned with the nearest face of the volume. This image buffer has a fixed pixel to voxel scale. The volume is then rendered to the image buffer slice for slice using fixed scaling and blending factors. When all slices of the volume has been rendered, the image is warped into the original orientation and scale. Shear warp is a relatively fast software implementation for direct volume rendering. The image quality however is potentially not as good as the quality of ray casting. Shear warp factorization is an object-order rendering approach. This means that for each volumetric value, the appropriate pixels are updated (see Figure

3.1.

Another technique that trades quality for speed is *splatting* [53]. This is a method that iterates all volume elements in a back to front order, and every element is splatted on to the viewing plane like snowballs on a wall. This results in the nearest objects overdrawing the ones further away, similarly to the painter's algorithm. The properties like color and transparency of these splats vary diametrically in a Gaussian manner. This means that the center of the splats are most intense, and it dissolves out towards the edges. The main computational cost in this technique is that every volumetric element has to be ordered by distance from the eye for each rendered frame. There exists variations of this technique, where the splats may have other properties. Harris and Lastra has used splatting of particle systems for their real-time cloud rendering approach [20]. Splatting is an object-order rendering approach (see Figure 3.1).

3.2.2 GPU-Accelerated Ray Caster

As explained in the previous section, ray casting is a simple method for direct volume rendering, that also gives very good results. However, the performance of the method was a problem that lead to the development of other techniques. In the later years, this problem seem to have become less significant, due to the recent graphics hardware that has become available. Modern graphics hardware has become very advanced, and the possibilities of making shader programs has revolutionized the ways of graphical programming. The modern GPUs are optimized for high throughput of vertices and fragments, and the memory latency is very low. The hardware is cheap, easily available, and widely supported by hardware architectures and operating systems. Modern graphics hardware also supports high level shading languages. This makes GPU-programming ideal for computation that can be done with textures as input and output [16].

Since graphics hardware supports handling of 2D and 3D textures, it is reasonable to store volumetric data as textures. 2D and 3D textures are arrays of voxels. One texture can store up to 4 variables for each voxel, originally as the RGBA color channels. Volumetric data sets are often stored as a cubic grid of values. This makes it easy to map the volumetric values to the voxels of a 3D-texture, or a set of 2D-textures.

The earliest GPU-accelerated ray caster [46, 29, 43] was a multi-pass approach. The CPU would initiate the ray traversal, and the GPU would calculate the fragments for every increment. Since more recent graphics hardware supports *for*-loops in the shaders, volume rendering can now be done in one single pass. For every fragment that is rendered, the fragment shader will loop through the volume in the direction from the eye. For every step a value will be fetched from the 3D texture. The transfer function will map this value to an optical property which is compiled with the previously accumulated optical property of the fragment [12]. When the fragment is fully opaque, i.e. the alpha channel is 1, the loop is terminated as nothing more will be visible along that ray. This optimization technique is called *early ray termination*.

The coordinates of the volume data set is mapped between [0,1] along all three

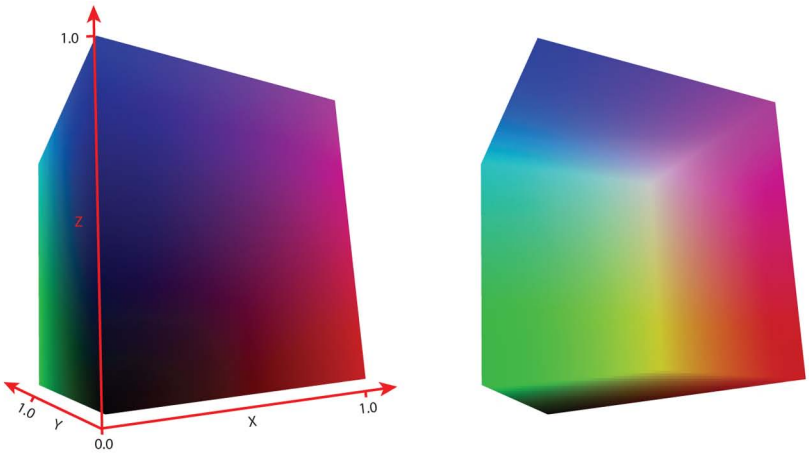


Figure 3.2: *Color cubes that are used to get the coordinates in the voxelized volume. The coordinates of the front faces on the left, and the back faces on the right. The RGB vector represent the corresponding (x,y,z) coordinates in the volume. The direction vector is found by subtracting the back vector from the front vector.*

axes. In order to cast rays through the volume, the fragment shader will need to know the entering and exit points of the volume for each fragment. This is done by pre-rendering color maps for the front and the back faces of the volume. A color map will in the simplest case be an RGB cube with the same size and transformation as the volume. The RGB axes are mapped to the (x,y,z) axes of the volume, with values between [0,1]. Color cubes are illustrated in Figure 3.2. With these color maps as input to the fragment shader, we have the entering and exit vectors of the volume at every fragment. By subtracting the back by the front vector, we also have the distance vector for the ray. This distance vector is then divided into sample steps along the ray.

For simplicity, what has been described until now is the use of cubic color maps. However, Scharsach et al. introduced a more sophisticated way of creating the color maps [47]. Based on the data values that we are interested in, this method will generate a bounding object of the volume, built up by small boxes, instead of one single cube. This way the color maps will skip large parts of the volume that is empty, a technique often referred to as *empty-space skipping*. This is shown in Figure 3.3.

3.3 Light Propagation in Volumes

One of the earliest approaches to rendering of light scattering in volumes was done by Blinn [2]. As he needed to render the rings of Saturn, he came up with an approximate lighting method for cloudy and dusty surfaces. The approach was simplified in the way that he assumed that the primary scattering effect was reflection from single particles. In computer graphics this has been a common assumption, but it is only realistic for

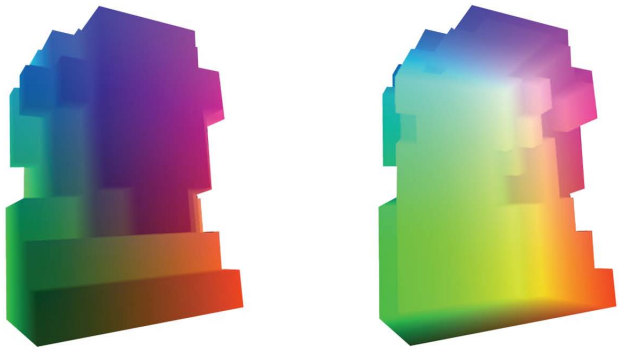


Figure 3.3: *Color shapes that are used to get the coordinates in a voxelized volume. This is an optimization of the color cubes in Figure 3.2. The method removes empty space from the color maps and improves the rendering performance.*

volumes that has a low single scattering albedo [3]. The equation that was introduced in Section 2.2.7 does also consider multiple scattering, but the double integral equation is too computationally expensive for a real-time application. Because of this there have been many approximations for multiple scattering. Harris [19] categorized the approximation techniques into the five categories spherical harmonics, finite element methods, discrete ordinates, Monte Carlo integration and line integral methods. In this section, a brief introduction will be given to the categories that are most relevant to the scope of this thesis.

3.3.1 Finite Element Methods

One way of simplifying the light transport equation is to use a *Finite Element Method*. In this method an unknown function is divided into a finite number of elements. These elements are often polynomials which together approximate the unknown function, and they can be solved numerically [6]. The lighting technique *radiosity* uses a similar approach to calculate diffuse reflection among surfaces. An integral equation will characterize the amount of light reflected from the surfaces. In the radiosity approach all surfaces are divided into smaller elements called patches. For each pair of patches, a *view factor* is calculated. The view factor is a number that states how well two patches can ”see” each other. If the distance between two patches is high, or they are oriented at oblique angles to each other, the view factor is low. Then the view factors are used as coefficients for a linearized form of the rendering equation.

Nishita et al. developed a finite element method for approximating global illumination in clouds [39]. This method accounts for multiple anisotropic scattering. It divides the unknown function into several simple functions that can be solved numerically, and can therefore be considered as a finite element method. In this case the volume is divided into voxels, and the light radiance between each voxel is calculated. Com-

pared to Equation 2.17 presented in Section 2.2.7 two important assumptions are made. The most important assumption is that the phase function of water droplets is highly anisotropic. Forward scattering accounts for most of the scattering, and contributes most to the light propagation. This makes calculation a lot simpler, if we assume that backward scattering can be neglected. The result of this is that Nishita used a reference pattern of all voxels that contributes to the illumination of every point. The same pattern can be used at all positions if the light source is assumed to be at an infinite distance. The second important assumption made by Nishita is that only a low number of scattering steps needs to be considered. In his method scattering is only calculated up to an order of three steps.

3.3.2 Line Integral Methods

When approximating the rendering equation, we could first consider the simplest case where there is no scattering involved. To be able to calculate the light extinction through the volume, we could trace the light propagation from the light source through the volume. By integrating the light intensity along each ray of light, we could approximate the light attenuation along that line. This is why Harris describes these methods as *line integral methods*. Since this method goes through the volume once in one direction, we can immediately see the problems in calculating multiple scattering. Since the rays we follow go only one direction, we will never be able to include the backward-scattered light to the calculation. Fortunately, in cloud rendering this can be avoided. As explained in Section 2.2.5, particles with a radius substantially larger than the wavelength of light has a phase function which favors forward scattering. Because of this one approximation is to totally neglect the backward-scattered light, and focus only on the forward scattered light around the line we are integrating. This technique has been used by Harris and Lastra on clouds represented as particle systems [20]. Later, Harris extended it to also work with voxel volumes [18].

Kniss et al. [27] introduced a method to render translucent media with both absorption and multiple forward scattering considered. This technique divided the light calculation into several slices along the half angle between the viewer and the light source, each step accounting for one step in the line integral. Half angle slicing will be described more throughly in Section 3.4.1. Riley et al. [45] presented a method for visualization of multi field weather data-sets that was based on both Nishita's approach [39] and Kniss' half angle slicing. This way they could integrate the light attenuation and multiple forward scattering with a line integral method, while also using Nishita's voxel pattern for scattering as described in Section 3.3.1.

3.4 Shadowing and Illumination in Volume Rendering

In the real world, illumination is what makes it possible for humans to visually perceive things. Everything that is visible to the human eye is affected by illumination. Because of this, shadowing and illumination is very important in computer graphics in order to

render images which aim to imitate the real world. In the previous sections, different methods that can be used to render volumes have been introduced, as well as methods that can be used to approximate physically plausible light transport in volumes. This section will focus on work that has been done that combines volume rendering with light transport approximations.

3.4.1 Shadowing Methods

When it comes to illumination in computer graphics, we deal with approximations of the real world that will suffice in the current application. The real world is far too complex to recreate, therefore we need to accept simplifications to achieve the desired results. One of the important things to consider when it comes to illumination in computer graphics is the effect of shadowing. In polygonal graphics this is seen clearly, as every polygon occludes parts of the scene from the light source either fully or partially. This is the main basis for the shadowing techniques known as *shadow volumes* [8] and *shadow mapping* [54]. Shadow volumes represent the geometry of the parts of the scene that are occluded from a light source, and hence is in shadow. This way the scene is divided in two: the areas that are occluded from the light source, and the areas that are lit. Shadow maps stores the depth of the scene as seen from the light source. These values are later checked against every point that is rendered, to find if the point is in shadow or not. Both these methods deal with binary checks of shadows, they can only decide whether a point is shadowed or not, and the result is what is called *hard shadows*. In polygonal rendering this simplification can be useful, but in volume rendering we would usually need to consider partial occlusion as the media is usually semi-transparent. To be able to render semi-transparent media with proper shadowing, we need to know the amount of light that is occluded at every point. This would give what we call *soft shadows*.

The theory of shadowing is closely related to visibility algorithms in computer graphics. Calculating the visibility of a point from the eye position is very similar to calculating the amount of light that reaches a point from the light source. In volume rendering, we usually deal with media that partially occlude light. This means that we would need to calculate the partial light occlusion at every point through the volume, and it makes the problem a lot more complex than if we only need to do binary decisions. There are a few common ways to achieve this. One solution is to keep an additional 3D *attenuation volume* [1, 40, 25], where the light attenuation is stored for every sample from the light source. Another way to solve this shadowing problem is to use *half angle slicing* [27, 28].

Kajiya and Von Herzen [25] used a separate 3D volume to store shadow data in addition to the rendered density volume. Behrens and Ratering [1] used a similar approach in their GPU-accelerated shadowing technique. They exploited hardware texture mapping when generating their 3D shadow texture to achieve better performance. Their shadowed volume was rendered using texture based volume rendering [5]. Nulkar and Mueller [40] also used 3D volumes for the shadow data. They used

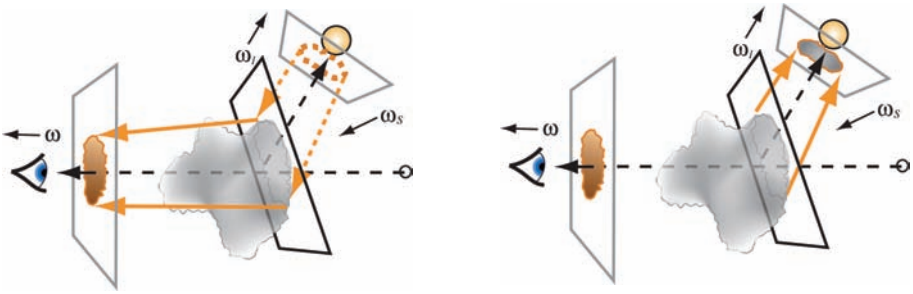


Figure 3.4: *The two passes that are done at every slice in half angle slicing: a) volume is rendered to eye buffer, b) light buffer is updated. Illustrations from Kniss et al. [28].*

splatting both for the shadow computation and the volume rendering. All these approaches generate the shadow volume in a pre-processing step. The shadow volume only needs to be generated once until the light position or the volume changes. Using a separate 3D volume for the shadows is a quite flexible approach, as almost any volume rendering technique can be used for the rendering. It will also give good performance for situations where the light position is static, as the shadowing only has to be calculated once. The downside of using a separate shadow volume is that it takes large amounts of memory to get good quality results. Also Deep Shadow Maps [34] uses a separate 3D texture for the shadow data, but this method uses less memory due to the way the shadow map is built. This method will be described further in Section 3.4.2.

Another approach for shadowing in volume rendering is half angle slicing [27]. This is a method that achieves per-pixel accurate shadows, without the use of a separate shadow volume. In this approach, three buffers are used. The volume is sliced along the half angle between the light source and the eye position. This way, it is possible to traverse the entire volume with an increasing distance from the light source and a decreasing distance from the eye. Three 2D buffers are used, two for the light intensity(*current* and *next*) and one for the eye-rendering. The slicing is divided into two passes, one for the illumination and one for the eye-rendering. In the illumination pass, the *next* and *current* buffers are swapped, and the light intensity is rendered to the *next* buffer. This is done based on the illumination now stored in the *current* buffer and the data fetched from the volume. In the next pass, the same volume data is rendered to the eye buffer, blended with the illumination from the *current* buffer. During the blending, the illumination buffer is sampled multiple times at jittered positions, and the average value is used. In this way a blurring of the forward scattering is achieved. Figure 3.4 illustrates how each step in the half angle slicing is done, and Figure 3.5 shows two images rendered with this technique. Zhang and Crawfis [56] used a similar half angle-approach in their work that was based on the splatting technique from Nulkar and Mueller [40]. Later, Zhang and Crawfis extended this work to also enable soft shadowing [57].

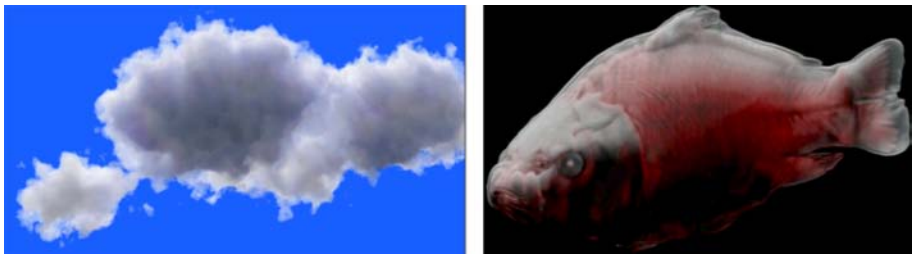


Figure 3.5: *Two semi-transparent volumes rendered using half angle slicing. Images from Kniss et al. [27].*

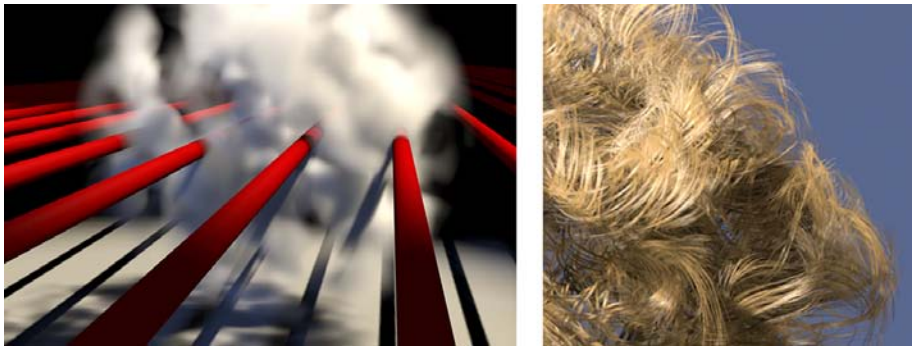


Figure 3.6: *Left: Smoke rendered using Deep Shadow Maps. Right: Fine hair rendered using Deep Shadow Maps. Images from Lokovic and Veach [34].*

3.4.2 Shadow Mapping and Deep Shadow Maps

As described in the previous section, the first approaches that dealt with shadowing in computer graphics used a very limiting simplification. These methods would consider every point in the scene to be either illuminated or shadowed. One of the first approaches that used this binary approach was introduced by Lance Williams [54]. His approach was called Shadow Maps, and is a popular shadowing method. In this method the scene is first rendered from the position of the light source. What is really rendered from this angle is the depth of everything that is seen. These depth values are stored in what is called a *shadow map*. When the shadow map is generated the scene is rendered again, this time from the eye position. By knowing the modelview matrix that was used when rendering the shadow map, we can convert any point P in eye coordinates to the corresponding point P‘ in the shadow map coordinates. This enables us to check the z coordinate of every point that is rendered from the eye position against the corresponding shadow map value. If the z coordinate is higher than the depth value we know that the point is occluded, hence it should be shadowed.

For scenes that deal with fully opaque objects and surfaces regular shadow maps may give sufficiently good results. Unfortunately, this is rarely the case. When we

deal with participating media such as clouds, this becomes very clear. Rather than just knowing whether a point is shadowed or not, we need to know *how* illuminated it is. The fact that this value relies on far more sophisticated factors than just the result of a depth test makes it obvious that other methods has to be used. One method that is an extension of the shadow mapping technique is called Deep Shadow Maps. This method was introduced by Lokovic and Veach [34]. Originally it was developed to give high quality shadows when rendering primitives like fur, hair and smoke. Figure 3.6 shows two images rendered from the original deep shadow map implementation. The basic idea of this technique is to extend the idea of shadow mapping to contain more data than just the depth at where an occluding object is. Instead they store an array of values that represent the intensity of the light along a ray through the scene. This can be done using an approach quite similar to ray casting. As a ray go through the volume a pair of values, light intensity and current depth, is stored for every sample where there is sufficient change. This way the illumination through the entire volume is stored in a deep shadow map. After the deep shadow map is generated, the scene is rendered from the eye. Now every point that is rendered can be checked against the corresponding point in the deep shadow map to get the light intensity. Since the deep shadow map only contains intensity information at the samples where the intensity changes sufficiently, the intensity value cannot be fetched directly from the deep shadow map. A search has to be done through the array of samples, and the intensity value is interpolated between the samples that match the depth.

Hadwiger et al. introduced a method to exploit modern graphics hardware with Deep Shadow Maps [17]. This is done by first rendering multiple slices of the shadow map using multiple render targets. Next, the slices are copied into a 3D texture that is later used together with a GPU-accelerated ray caster for the final image synthesis.

4
Weather Visualization Pipeline

In this thesis, the goal is to introduce a system for physically plausible interactive weather rendering, with focus on simulated hurricanes. The applications it is targeted at are mainly news and weather forecasts for television broadcasters. In order to give the user ability to quickly change the view or different properties, the system should be able to give interactive frame rates. The use for such a system could easily be seen in other areas as well, like for meteorologists or in the game and movie industry. During the process of this thesis, I have been in contact with VizRT, a company that delivers graphics software for television broadcasters. This has helped steering the work in a direction that is desirable for their use.

This section presents a high level overview of the system, and in the following chapter the separate steps will be described more in detail.

4.1 Meteorological Data

One important aspect to consider for a weather visualization system is the type of meteorological data that should be used. The generation of meteorological data sets is a large research field in itself. In general there are two main principles that can be followed, either one can aim to recreate an actual happening that has occurred, or one could create an imaginary happening. In many situations one would like to generate clouds or other weather phenomena that will look good in a given environment. For example in movies or games, the scene will often be the main concern, and the meteorological data should be made accordingly. This will require a manual design of the clouds. In other situations, one would want to give a visualization of an actual weather phenomena that has happened. For example during the hurricane season, it would be very desirable to give images that represent the current situation. This would require another approach in the data generation, usually solved by different weather models that generate the data sets.

Within the scope of this thesis, it is most reasonable to use data sets that are simulations of actual happenings. For such purposes, the most common weather model is the *Weather Research and Forecasting model* [49] that was described in Section 2.1. The

WRF model is a numerical weather prediction model. This means that it generates a forecast based on the atmospheric state in the beginning of the simulation. The atmospheric state is then simulated over time, and the output is given as numerous separate time steps. Since the visual elements in the atmosphere that we aim to visualize in this thesis consist of different types of particles, the data sets are also divided into multiple parts for each time step. In the WRF model, the most common hydrometeor fields that are considered are cloud, rain, ice, snow, graupel, and vapor.

In weather phenomena like hurricanes, there are also a lot more information than just the volumetric data itself that could be interesting. This makes it important to choose the right data format that can store certain meta data. NetCDF [50] is a data format specifically created for meteorological data. Data sets stored as NetCDF can contain a unlimited number of chosen variables, as well as additional attributes. This way, attributes such as longitude and latitude, the physical scale and other important features can be easily accessed. This helps in automating the process of scaling the weather phenomena correctly, and positioning it into a world context.

In the implementation of a volume renderer, it is common to represent the volumes as three dimensional textures, since textures are natively supported by graphics hardware. This makes us able to access the volumetric data directly from a shader program, which is run directly on the graphics processor. A 3D texture can keep from one to four data values at each voxel. This will however limit the precision of each value to 8 bits. Because of this precision limitation, and the fact that the meteorological data sets may contain more than four values at each point, the proposed system uses a separate 3D texture for each hydrometeor field in the data set.

4.2 Shadowing and Illumination

In order to approximate photo realistic images based on a data set, we will first have to calculate the illumination through the participating media that the data sets represent. In this thesis a Deep Shadow Maps (DSM) approach [34] has been chosen for the calculation of shadowing and light propagation. A DSM approach gives the benefits that we can use a simple rendering algorithm like ray casting when the light calculation is done. DSM is also a view independent method, which means that we can freely change the camera position without having to recalculate the light for every frame. In a hurricane visualization this can be suitable, as it could be interesting to investigate every time step from multiple viewing angles. Rendering techniques like ray casting can also give a much better result without increasing the rendering time, unlike for example the half angle slicing technique by Kniss et al. [27].

As explained in Section 3.4.2, the DSM approach uses a separate 3D volume that stores the shadow information. This is done by first rendering the scene from the light source, storing the calculated illumination values in a new 3D structure. A shader program that runs on the graphics processor is not able to update arbitrary voxels in a 3D texture. Therefore the operation has to be done in multiple passes, by rendering

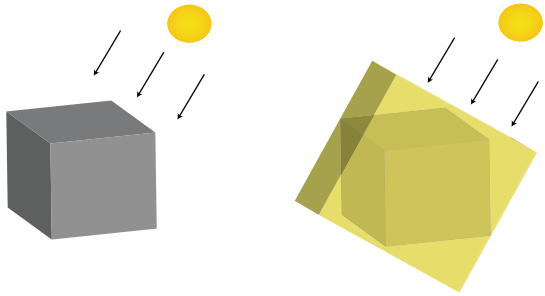


Figure 4.1: *a) The volume depicted as a gray box. b) The DSM depicted in yellow acts as a bounding box around the volume.*

slices of the final 3D structure in each pass. The light values for these slices are made using a modified version of ray casting that was explained in Section 3.2.2. In each pass a ray caster is initiated where every ray starts where the same ray from the previous pass stopped. When a ray has traversed densities in the volume that change the volume illumination, the ray is stopped and the updated illumination value is rendered together with the depth value. This way, we get a data structure where the actual volume is shuffled to the front slices, and all the empty space is left in the latter slices. When we later want to look up the illumination of a point (x,y,z), a search for the depth z has to be done through the array of slices at the point (x,y). Figure 4.1 illustrates how a DSM is mapped as a bounding box around the volume. The volume is depicted in gray, and the DSM in yellow. As can be seen in the figure, the volume and the DSM will have two different coordinate systems. Therefore a mapping has to be done when we want to do a look-up in the DSM while rendering. Section 4.3 will shed light on how this is done.

For the light calculation at each point, this thesis is based on the work by Riley et al. [45]. Riley et al. use Nishita's work [39] as a basis for rendering weather data consisting of multiple hydrometeor fields.

4.3 Rendering

After the light is calculated and stored in a 3D texture, the rendering can be started. This is done using traditional GPU-accelerated ray casting, as described in Section 3.2.2. At this point, all the required hydrometeor fields from the data set are available as separate 3D textures, as well as a 3D texture containing the DSM. Since the DSM is in another coordinate system than the volume, we have to transform the volume space coordinates to shadow map space in order to get the light intensity value. As the casted rays go through the volume, the light attenuation is accumulated so we know the opacity at the current point. This is done by calculating the optical depth of each sample

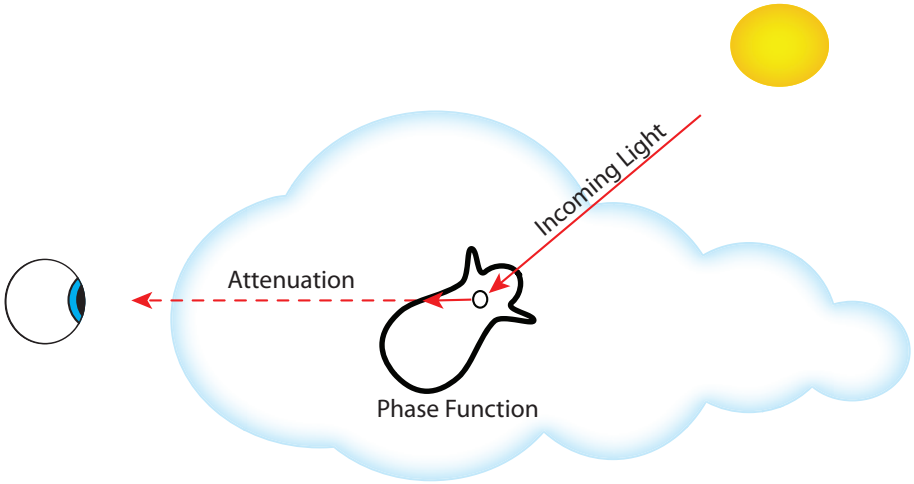


Figure 4.2: *An illustration of the rendering step. The phase function gives the amount of scattered light that is directed towards the viewer. The volume between the eye and the point will attenuate the light intensity. The light intensity is fetched from the pre calculated deep shadow map.*

as described in Section 2.2.6. Next, we have to find out how much light is scattered outwards from the current sample. This is done by considering all the hydrometeor fields and calculating the scattering coefficient as in Section 2.2.2. Then we need to know the amount of scattered light that is directed towards the viewer. This is where the phase function is used. A phase function gives the fraction of the out-scattered light that is scattered towards the viewer, based on a given angle between the viewer and the light source. When we know the amount of light that is scattered towards the viewer and the optical depth between the viewer and the current sample, we can update the color of the current pixel. The ray will be stopped when the optical depth is high, since the following samples will be occluded. Figure 4.2 illustrates the rendering step.

4.4 Embedding into Environment

One of the goals for this thesis is to render near photo-realistic visualizations of weather phenomena like hurricanes. This does not only include to impose the light propagation and visual properties of the hurricanes, it also implies that the hurricanes should be rendered in a realistic setting. For broadcasters it would make sense to integrate this system in a map framework, with satellite photos or similar as a background. In this system, satellite photos from NASA's high resolution image sets called *blue marble* [37] is used. These images span from longitude -180 to 180, and latitude -90 to 90. Based on this information and the meta-data in the WRF data sets, the visualizations can easily be positioned correctly on such maps.

Figure 4.3: *A rendering of hurricane Isabel approaching Florida. Mapped according longitude and latitude values to a Blue Marble satellite photo.*

The scale of the weather data sets are often not symmetrically mapped to the physical scale. For example if the physical distance in x direction is slightly different from the y distance, the data set might still have the same number of voxels in both directions. Also, due to the curvature of the earth, the physical distance on the north edge usually differs from the south edge even though the data set is cubic with equal edges. The flexibility of the ray casting technique helps us take care of these issues. If we want to change the scale of the volume that is rendered in any direction, all that is needed to do is to render the bounding box at the correct scale to the color maps.

Figure 4.3 shows a rendering of the hurricane Isabel approaching northeast Florida. Isabel was a major tropical cyclone that struck Caribbean and southeast United States in 2003.

4.5 Algorithm

The proposed weather visualization system consists of two parts: first the surrounding environment is rendered, and then the weather phenomenon is rendered over the environment. When the weather volume is rendered, blending is enabled and the background will shine through the transparent areas of the volume.

Since the deep light map technique will pre-generate light information, the weather rendering algorithm has two main passes. First, the light propagation in the volume is gathered, and stored in a deep light map data structure. Second, the volume is rendered using a direct volume rendering method, using both the volume data and the light

information from the DLM. Algorithm 1 shows pseudocode of the weather rendering algorithm. The final rendering is positioned onto the previously rendered environment, based on the longitude and latitude positions of the volume.

Algorithm 1 Pseudocode for weather visualization system

```
drawEnvironment()

if time step changed then
    importTimestep(time)
end if

if light-position or volume changed then
    Attach data sets
    generateDeepLightMap()
end if

Attach data sets
Attach deep light map
renderVolume()
```

5
Methodology

The previous chapter gave a brief introduction to the main aspects of this thesis. This chapter will go into details on the techniques that have been used. Although the main scope of the thesis is visualization of hurricanes, it will to some extent refer to the subject as cloud visualization. This is because the theory regarding cloud visualization is very similar to that of hurricane visualization. Therefore, clouds are used as simplified examples when discussing certain subjects.

Hurricanes are very large in physical scale. Thus, in order to capture enough detail, a vast amount of data is needed to represent the hurricanes. When multiple variables are stored at each sample for a hurricane that spans over several thousand kilometers, it is not unusual for one time step to occupy more than 500 megabytes of raw data. When we also know that one data set can contain hourly time steps for several days, it becomes obvious that we face an array of challenges when handling all this data. The fact that we also aim to perform the visualizations with interactive frame rates will further complicate our task. Section 5.1 will discuss these challenges and describe how they are solved in the scope of this thesis.

Weather phenomena in general are very complex to imitate in computer graphics. The gaseous nature of clouds imposes the rendering system to employ advanced rendering equations. The absence of solid surfaces makes the light calculation intricate, since there are no normals that decide the shade of the matter. All visible properties of clouds depend on which areas of the cloud is illuminated, and how the light is reflected from each respective area towards the viewer. Thus we can say that the problem is divided into two parts: the light transport through the media, and the scattering toward the eye. Section 5.2 will give an introduction to the rendering equation used, and the light transport equation. Section 5.3 will in turn describe the way the light transport equation can be solved for the entire volume using a technique called Deep Light Maps. Section 5.4 will focus on how the rendering equation is solved, incorporating the previously calculated light transport values. In Section 5.5 the rendered volume will be embedded into the appropriate context, by using satellite images of the world with the aim to make the scene as photo realistic as possible. Figure 5.1 shows the entire flow of the proposed weather visualization system.

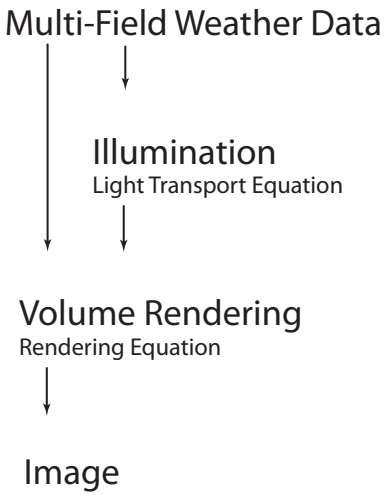


Figure 5.1: *Weather Visualization Flow*

5.1 Meteorological Data

A hurricane consist of many different types of hydrometeor particles. The most common hydrometeor particles are water in different conditions, ice, water and vapor. The particles are of various sizes and shapes. The most common hydrometeor fields in storm or hurricane data sets are rain, cloud, ice, snow, graupel and vapor. Figure 5.2 shows the distribution of the different hydrometeor fields in the hurricane Isabel.

5.1.1 Data Acquisition

A common model for simulation of the atmospheric state is the *Weather Research and Forecasting model* [49], usually referred to as the WRF model. The WRF model is a numerical weather prediction model, which simulates the atmospheric state with respect to certain conditions. A starting state is used as input to the model, and advanced calculations are done to simulate the forthcoming state. These simulations are usually done on large supercomputers due to their complexity. The outcome of the WRF simulations are divided into multiple variables. The most common variables are particle/air mass ratios for rain, cloud, snow, ice and graupel, and other variables such as wind velocity, temperature and pressure. These mass ratios have usually very low values since the amount of air is larger than the amount of hydrometeor particles.Therefore the values are stored as floating point numbers less than 1. The simulated data sets are often stored in a file format called NetCDF [50]. This is a self-describing format, which means that the layout of the file is described in a file header. NetCDF allows a unlimited number of free variables in one file, each variable of potentially different data type. The file format can also store other meta data such as longitude and latitude

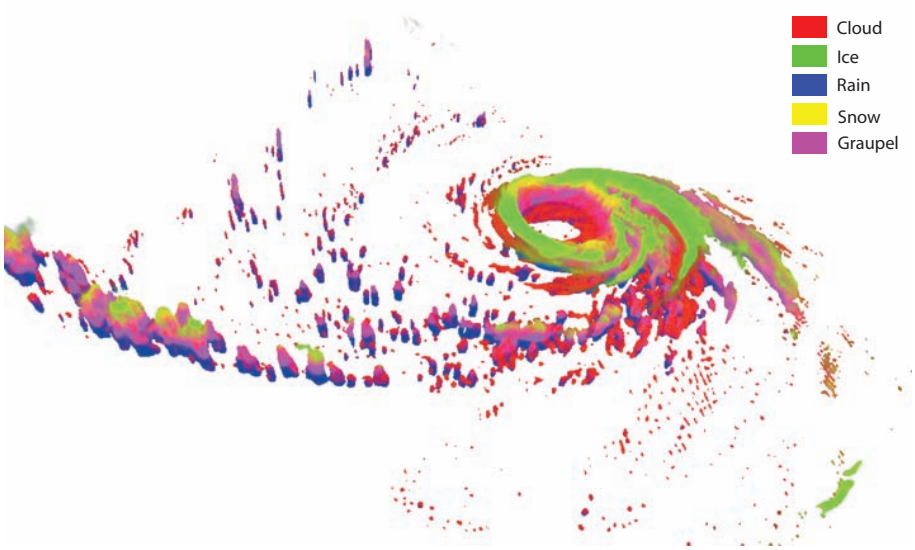


Figure 5.2: *Colored hydrometeor fields of hurricane Isabel*

positions and physical size of the simulation.

An important aspect when developing a system for weather visualization is the availability of the required data sets. The WRF model is widely used by atmospheric research institution. It is principally developed by large organizations like the National Center for Atmospheric Research (NCAR) [38], the National Centers for Environmental Prediction (NCEP), the Forecast Systems Laboratory (FSL) and the Air Force Weather Agency (AFWA) among others [49]. This shows that there is a broad support for the WRF model, and it is said to be a next-generation mesoscale numerical weather prediction system [49]. The NetCDF format is developed by NCAR, and they are also hosting large amounts of weather data in the NetCDF format [38].

The data sets used for illustration purposes in this thesis are the Isabel data set from the IEEE Visualization 2004 Contest [23] and the Katrina data set from the NCAR database [38]. The Isabel data set is based on the WRF model, and stored in a *brick-of-floats* format. The dimensions of the data set are 500x500x100 voxels organized on a rectilinear lattice, and each voxel is a 32 bit floating point number. Hence, each variable is 100 megabytes of raw data. The data set consist of 48 time steps, with one simulated hour between each time step. There are 6 available hydrometeor fields, cloud, graupel, ice, rain, snow and vapor. The Katrina data set is also based on the WRF model, and stored in the NetCDF format. The dimensions are 315x309x34 and each voxel is a floating point number, resulting in approximately 13 megabytes for each variable. The Katrina data set has three available hydrometeor fields, vapor, rain and cloud.

5.1.2 Data Management

The first challenge that arise in our hurricane visualization system is to handle the vast amount of data. As one time step can consist of several hundred megabytes, there are many obstacles to overcome when we aim to render the visualization with interactive frame rates. First of all, reading those amounts of data into memory will take a certain amount of time. This is hard to come by, since the limitation lies in the hardware and operating system and the proposed system aims to be runnable on consumer hardware. In some cases, there are 50 time steps that should be read. With the original data size of 500 megabytes per time step, this would total to 25 gigabytes. Therefore it is clear that pre-loading all the data would be impossible with the amount of memory in computers today. The only pre-loading that can be done is to keep a buffer with the next and previous time steps, that is continuously updated. But even if a buffer is used, it would be desirable to cut down on the actual data size to maximize the performance.

As described in Section 5.1.1, the data values are often stored as 32 bit floating point numbers. One possibility is to store the data sets using the 16 bit unsigned short data type rather than 32 bit floats. This would halve the amount of data, but it would also limit the precision of each value to a large extent. In the graphics hardware, unsigned short values are mapped between 0.0 and 1.0, where the maximum unsigned short 65536 is mapped to 1.0. From Section 5.1.1, we know that particle/air mass ratios in the data sets usually have very low values. In most cases, the values are below 0.03. Therefore, it is clear that a lot of precision is wasted if we use the entire 0 to 1 range, since only a fraction of the actual range is needed. To improve the precision using the unsigned short data type, we can scale up each value by a known factor, such that all values in the data set is below 1, but as high as possible. To take the Isabel data set as an example, all particle/mass ratios are below the value 0.03. In this case we can safely multiply each value by 32, since 0.03*32 = 0.96. This way we would get more out of the available range of the unsigned shorts. When we read back the values later, they are divided by the same factor, in the example case 32. Figure 5.3 shows the difference between a un-scaled unsigned short data set and one up-scaled with a factor of 32. It can clearly be seen that the un-scaled data set suffers from bad precision.

When the data is read to the main memory, each variable is stored as an array. Now the concern is to represent the data such that it is feasible to process it in a reasonable time. Modern graphics hardware are specially designed to deliver a high throughput of data. They can fetch multiple values from a texture in one clock cycle, and separate calculations can be done in parallel. To exploit the calculation performance of the modern graphics hardware in the processing of meteorological data, the data sets are bound as 3D textures. The textures, that are natively supported in the graphics hardware, can have the same dimensions as the data array in the main memory. Then, the next step would be to upload the data to the video memory. With the amounts of data we are working with, also this will require a certain amount of time. By utilizing pixel buffer objects (PBO), it is possible to achieve a transfer rate at between 1 and 2 gigabytes per second. This means that it could take up to one fourth of a second to upload a data set

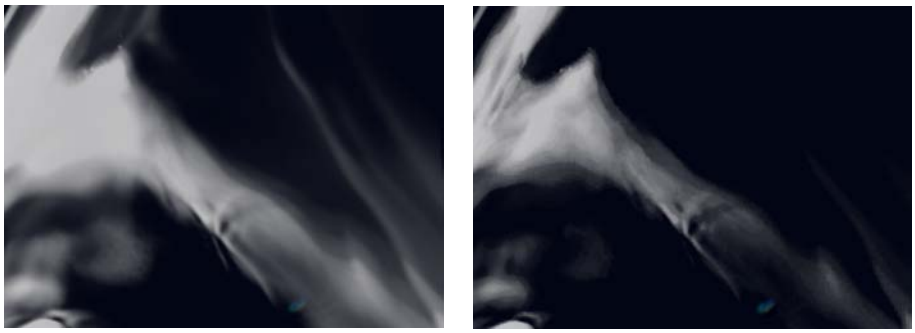


Figure 5.3: *Cloud details from the build-up of hurricane Isabel viewed from the atmosphere. Rendered a) using scaled unsigned shorts and b) using un-scaled unsigned shorts. In a), better precision is achieved using the same number of bits per voxel.*

of 250 megabytes. Although this is a noticeable delay, it will not be a bottleneck that renders it impossible to achieve interactive frame rates.

In Section 5.6 a technique will be introduced, that exploit knowledge about the light calculation presented in the following section in order to generate smaller customized data sets.

5.2 Light Propagation in Clouds

The previous chapter presented a way to represent meteorological data sets, and how to manage the data sets in computer programming. The result was a number of voxelized grids, containing particle/air mass ratios for the different hydrometeor fields. In Section 2.2, basic theory for light propagation in clouds was introduced. In order to use this theory to render the voxelized data sets, we have to derive simplified equations that can be solved numerically.

In computer graphics, it is usual to only consider single scattering for light. Usually, this would be the scattering towards the eye, since this is what would make the object visible. But to fully implement the volume rendering equation that was introduced in Section 2.2.7, multiple scattering has to be considered. A multiple scattering model is far more physically accurate than a single scattering model, but it would have to account for light scattering in all directions. This makes multiple scattering models very complicated and computationally expensive. Fortunately, as Nishita et al. has pointed out, the contribution of most of the out-scattering directions is insignificant to cloud rendering [39]. In fact, the far most dominant scattering direction lies within a small angle around the forward direction. This is why multiple scattering often is simplified to *multiple forward scattering* in cloud rendering. The light model in this thesis exploits this simplification, as well as it is simplified further to only consider

single scattering towards the eye.

The implementation in this thesis is based on a double pass method. First, the light intensity in the direction from the light source is calculated for the entire volume with multiple forward scattering and absorption. The result of this pass is then used while rendering the volume from the eye. This section will first introduce the volume rendering equation used in the second pass in Section 5.2.2. Then, the light transport model that results in the light intensity variable used in the volume rendering equation will be presented in Section 5.2.3.

5.2.1 Light Extinction

The previous section introduced the type of data sets that are used in the proposed weather visualization system. From that chapter we learned that the data sets consist of mass ratios, where each sample is given as:

$$H_f(S) = \frac{M_h(S)}{M_{air}(S)} \tag{5.1}$$

where $H_f(S)$ is the mass ratio for the hydrometeor field f in the sample S, $M_h(S)$ is the total mass of hydrometeors in S, and $M_{air}(S)$ is the total mass of the air in S. In order to render physically inspired images based on the weather data, we need a way to map the mass ratios to optical properties. The optical properties of clouds are mainly affected by light extinction and scattering. Therefore, the extinction coefficient is first calculated from the mass ratios. This will make the basis for the weather rendering system, since calculation of important properties like scattering and opacity is dependent on the extinction coefficient. The first step in the calculation of the extinction coefficient will then be to find the particle concentration. The particle concentration is given by the following equation (see Section 2.2.3):

$$\eta_f(S) = \frac{\rho_{air} H_f(S)}{V^f_{prt} \rho_f} \tag{5.2}$$

Here $\eta_f(S)$ is the particle concentration in hydrometeor field f and sample S, ρ_{air} is the density of the air, $H_f(S)$ is the mass ratio from the data set, V^f_{prt} is the volume of a single particle in the hydrometeor field, and ρ_f is the density of individual particles in the hydrometeor field. Now, when we have the particle concentration of the field, we can find the extinction coefficient. The extinction coefficient $\beta^f_{ext}(S)$ for the field f at sample S is the product of the particle concentration $\eta_f(S)$ and the extinction cross section σ_f of the current field. As explained in Section 2.2.2, σ_f is assumed to be twice the average geometrical particle cross section in field f. For simplicity, each particle is assumed to be spherical. The particle cross sections are then defined according to Pruppacher and Klett [42] in Table 5.1. Now the equations 2.1 and 5.2 are combined,

Hydrometeor	Radius, mm
Cloud	0.01
Ice	1
Rain	1
Snow	2
Graupel	2.5

Table 5.1: *Spherical particle cross sections.*

resulting in the following equation:

$$\beta_{ext}^{f}(S) = \frac{3\rho_{air}(S)H_f(S)}{2\rho_f R_f} \tag{5.3}$$

where R_f is the average particle radius in the field f, from Table 5.1. Now the total extinction coefficient $\beta_{ext}(S)$ at sample S is the sum of the extinction coefficients for each field:

$$\beta_{ext}(S) = \sum_{Fields} \beta_{ext}^{f}(S) \tag{5.4}$$

5.2.2 Volume Rendering Equation

Now we know how to find the extinction coefficient from the mass ratios given in the data sets. In this system, all further calculations will be based on the extinction coefficient to some extent. First of all, it is an important factor for the transmittance T in the volume. The transmittance, or transparency, is based on the optical depth τ, which again is a product of the extinction coefficient and the length of the sample (see Section 2.2.6). Also, when we are dealing with clouds, the scattering coefficient β_{sca} is closely related to the extinction coefficient. In fact, when we assume that the single scattering albedo in clouds is 1, it means that all of the extinct light is scattered (see Section 2.2.2). Thus, we can say that $\beta_{sca} \approx \beta_{ext}$.

The next step is to define the equation that in the end will give the final color of a pixel. The volume rendering equation will approximate the total light transport to the eye, and is defined as [39, 35]:

$$\begin{aligned} L(\vec{w}) &= T(0,\vec{w})L_{bg} + \\ &\int_0^{\vec{w}} T(\vec{s},\vec{w})\beta_{sca}(\vec{s}) \int_{4\pi} P(\psi(\Omega))L_l(\vec{s},\Omega)d\Omega d\vec{s} \end{aligned} \tag{5.5}$$

Here, $T(\vec{s},\vec{w})$ is the light attenuation between points $\vec{s}$ and $\vec{w}$. The light attenuation is based on the optical depth, which was introduced in Section 2.2.6. $T(\vec{s},\vec{w})$ is pre-

viously given in Equation 2.13. L_{bg} is the light behind the volume. $P(\psi(\Omega))$ is the scattering phase function, and $L_l(\vec{s}, \Omega)$ is the light at point $\vec{s}$ in direction Ω.

To understand what Equation 5.5 means, we can divide the equation into two parts. The first part is $T(0, \vec{w})L_{bg}$. Here T will be the total transmittance through the volume. This will then give the amount of light from the background that is visible through the media. The second part of the equation is the light that is scattered from the media towards the eye. The outer integral will integrate over a ray through the volume. T is the transmittance, or transparency, from the front of the volume to the current sample. The inner integral will integrate over all directions, to calculate the amount of light that is scattered towards the eye.

When implementing the Equation 5.5, we first divide the light path from 0 to $\vec{x}$ into N discrete steps. We also need to make certain simplifications to make the equation computationally possible. The most important simplification is that we will only consider multiple forward scattering, so in the eye direction only single scattering is considered. The final equation can be written numerically as:

$$L(\vec{w}) = T(0, \vec{w})L_{bg} + \sum_{\vec{i}=1}^{\vec{w}} (1 - T(\vec{i} - 1, \vec{i}))P(\psi(\Omega))L_l(\vec{i}, \Omega) \qquad (5.6)$$

where P is the scattering phase function and $\psi(\Omega)$ is the angle between the light source and the view point. $L_l(\vec{w}, \Omega)$ is the incoming light at point $\vec{w}$ directed from the light source. The function L covers the calculation of multiple forward scattering, and will be introduced in the following section.

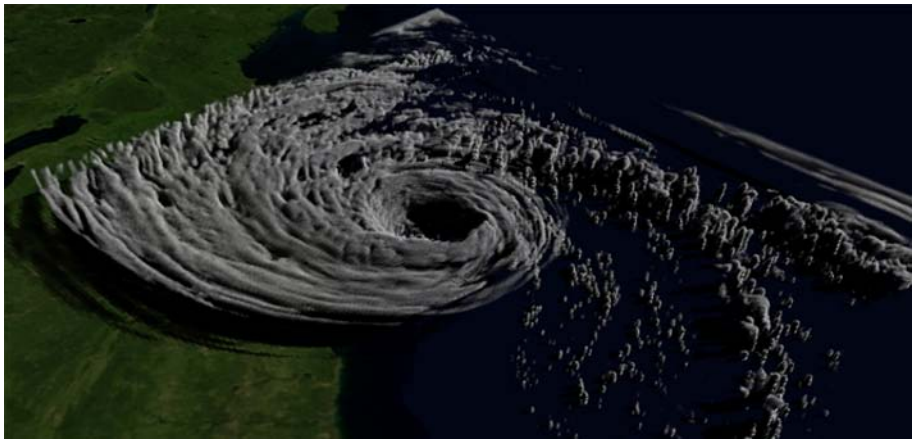


Figure 5.4: *Hurricane rendering where all out-scattering is considered to be lost. We can see that only the surface of the hurricane is lit, since scattered light is considered to be extinct.*

5.2.3 Light Transport Model

Light transport calculations in a volume determine the extinction of light due to scattering and absorption by the particles. The high single-scattering albedo in clouds make the absorption negligible, while the out-scattering is quite high. If we use a low-albedo model like the one Blinn introduced [2], all the out-scattered light would be considered lost. Figure 5.4 shows a hurricane rendered with this model. We can easily see that only the surfaces are lit, and the light does not reach as far into the clouds as it should.

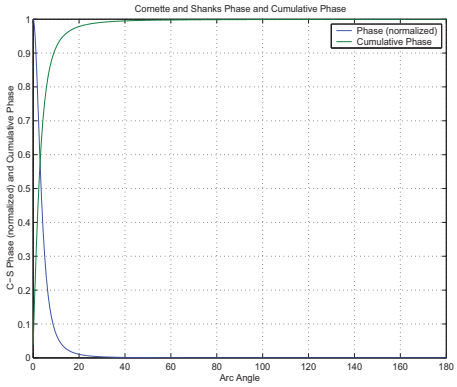


Figure 5.5: *The Cornette and Shanks normalized Phase and Cumulative Phase. Illustration from Riley et al. [45].*

As previously discussed, scattering in clouds is predominantly forward scattering. If we consider a phase function like the Cornette and Shanks Phase given in Figure 5.5, this becomes very clear. According to this phase function, 90 percent of the scattered light is scattered within 10 degrees. When we know this, we can develop an approximation to calculate $L_l(\vec{i}, \Omega)$ in Equation 5.6. In our model, we consider three possible states for the light after leaving a sample. It is either unscattered, forward-scattered or out-scattered. The unscattered light is light that does not strike any particles, and is not participating with the media. Forward-scattered light is the light that strikes a particle, but is scattered in a forward direction. The out-scattered light is the fraction of light that is scattered in any direction except forward, and in the proposed model is considered to be extinct.

In the proposed model, an angle θ is defined. All light scattered within this angle from the light direction is considered to be forward scattered. When we have this information, the light intensity at a point in the light direction L_l as used in Equation 5.5 can be given [39, 45]:

$$L_l(s, \Phi_{lt}) = L_{lt}T(s, lt) + \int_s^{lt} T(s, s')\beta_{sca}(s') \int_\theta P(\psi(\Omega))L_l(s', \Phi_{lt} + \Omega)d\Omega ds' \tag{5.7}$$

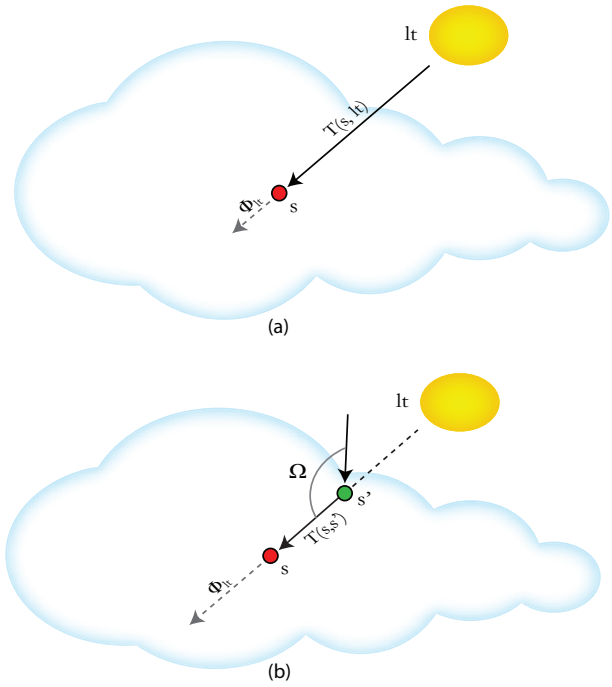


Figure 5.6: *a) The first term of Equation 5.7 is illustrated. This is accounts for the light attenuation, or shadowing, through the volume. The point s is depicted as a red circle. b) The second part of Equation 5.7 accounts for the inward scattered light. The point s is depicted as a red circle, and a point s′ between s and lt is depicted as green.*

where s is a point in space, Φ_{lt} is the direction of the light from the light source and L_{lt} is the light intensity at the light source. $T(s,lt)$ is the transmittance between the light source lt and the point s. In this equation, the first term is the unscattered light that is transmitted through the volume. The second term is the light that is scattered from particles within the θ arc angle. For simplicity, θ is assumed to be small enough to make the angle $\psi(\Omega)$ constant over the arc. This way we can treat the in-scattered light as directional light. Equation 5.7 is illustrated in Figure 5.6.

In the proposed system, the angle θ is divided into two regions, $\frac{\theta}{4}$ for the center region and $\frac{\theta}{2}$ for the peripheral region. This is illustrated in Figure 5.7. The peripheral light is an average of four equally spaced samples around the ray, illustrated by the red arrows. The Cumulative Phase $CP(\theta)$ is the integrated phase function, and is used to get the phase for the regions. With these simplifications we can calculate the light value L_l numerically. We define L_{in}^{ctr} as the incoming light in the center region and L_{in}^{per} as the incoming light in the peripheral region. T is the transmittance. For simplicity, the center transmittance is also used for the peripheral region. First we can calculate the

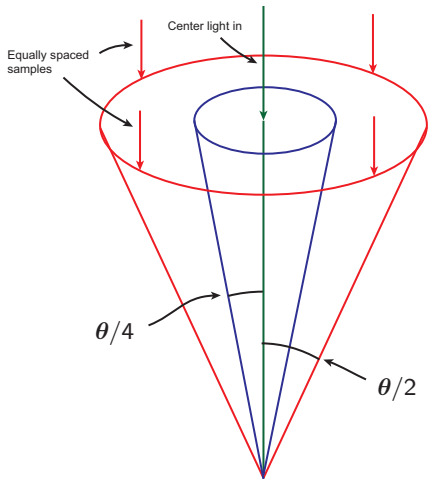


Figure 5.7: *Illustration of how the phase regions are divided. Transmitted light in green, light scattered from central particles in blue, and light scattered from peripheral particles in red.*

transmitted light:

$$L_{tr} = L_{in}^{ctr} T \tag{5.8}$$

Then, we calculate the in-scattered light from the center region:

$$L_{ctr} = L_{in}^{ctr}(1-T)CP(\frac{\theta}{4}) \tag{5.9}$$

Here, the term $(1-T)$ represents the amount of scattered light, since all light that is not transmitted is scattered. Last, the in-scattered light from the peripheral region is calculated:

$$L_{per} = L_{in}^{per}(1-T)(CP(\frac{\theta}{2})-CP(\frac{\theta}{4})) \tag{5.10}$$

Then, we finally calculate the total light transport:

$$L_l(\vec{s},\Phi_{lt}) = L_{tr}+L_{ctr}+L_{per} \tag{5.11}$$

5.2.4 Phase Functions

The way light scatters from a particle is largely dependent on the particles shape and size. Scattering from small particles can be well approximated with the Rayleigh scattering model [48]. As the particles increase in size, they are better approximated with the Mie scattering model [4]. When particles have a radius that is substantially larger than the wavelength of visible light, the forward scattering becomes dominant. This

is captured by the Mie scattering model. The larger particles also changes the scattering behavior such that it becomes less wavelength dependent. This is why rainbows is more visible in situations where there are small particles than in thicker clouds. Also other properties of the particles change the scattering behavior. Examinations done by Wendling et al. [52] discovered that cloud water has a higher forward scattering than ice particles, and ice particles has more side and back scattering. To capture these differences in weather rendering, different phase functions has to be used when dealing with different particle types. One phase function should be applied to ice particles like snow, ice and graupel, while another phase function should be applied to water particles like cloud and rain. Figure 5.8 shows phase functions for ice and water particles, based on calculations by Wendling et al.

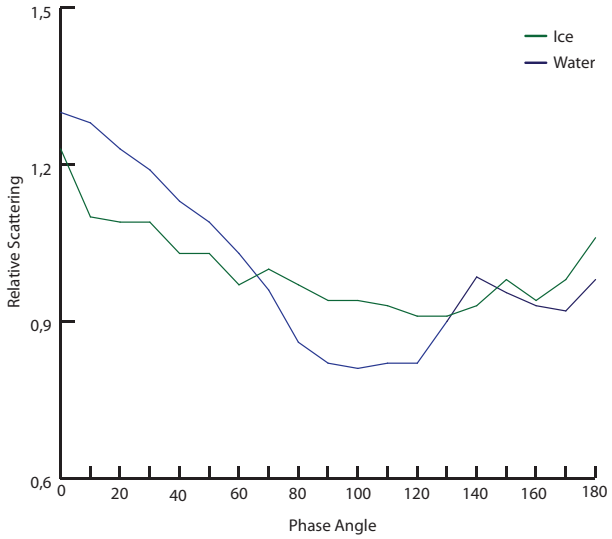


Figure 5.8: *Phase functions for ice and water particles, based on Wendling et al. [52]. Ice particles have more back and side scattering, while water particles favor forward scattering.*

Due to the limitations of imitating sunlight with an RGB value, we need different phase functions when considering light transport and eye scattering. Because of the intensity of sunlight, it could be partly occluded and still look bright white as an RGB color. Therefore, when we consider eye-scattering, the phase function is normalized over its dynamic range, and the peak around zero degrees is truncated (see Figure 5.8). However, when considering the light transport we are interested in the actual amount of the original light that is scattered. Since the scattering in clouds is dominated by forward scattering, we then need to consider the peak around zero degrees (see Figure 5.5).

In the volume rendering equation (Equation 5.5), we need a scattering phase function to consider all possible angles between the eye and the light source. For this purpose, two different phase functions based on values calculated by Wendling et al. [52]

are used. One phase function is applied to ice particles, and one is applied to cloud water (see Figure 5.8). Since we deal with multiple hydrometeor fields at the same time, it is also necessary to average the phase functions based on the relative amount of particles in each hydrometeor field [45]. The phase for all hydrometeor fields at a voxel $P_v(\theta)$ and scattering angle θ is given as:

$$P_v(\theta) = \frac{\sum_{Fields} P_f(\theta)\beta^f_{sca}}{\sum_{Fields} \beta^f_{sca}} \tag{5.12}$$

where β^f_{sca} is the scattering coefficient for the hydrometeor field f at the current position.

In the light transport equation (Equation 5.7), we use another phase function since only forward scattering is considered. For this the Cornette and Shanks Phase model [7] is used. In the numerically written form of the light transport equation (Equation 5.11), the Cornette and Shanks Cumulative Phase model is used directly for finding $P(\Omega)$. Figure 5.5 illustrates the Cornette and Shanks Phase, and the Cumulative Phase.

5.3 Deep Light Maps

Now, the light transport theory that makes the basis for weather rendering has been introduced. To use this theory to render the volumes introduced in Section 5.1, we need an algorithm for volumetric light calculation. This section presents the illumination algorithm used in this thesis. This algorithm is based on the deep shadow map method previously presented by Lokovic and Veach [34], and further improved by Hadwiger et al. [17]. The deep shadow map method used GPU-based ray casting to generate a separate volume containing shadow information. The term *deep* shadow maps is used because in contrast to regular shadow maps, this technique adds a depth dimension to the maps - which makes it is possible to consider partly occlusion. In this thesis the technique will be used to consider multiple forward scattering in addition to light attenuation. Therefore it is more suitable to use the term *deep light map* (DLM).

To generate a volumetric light map, we need to render the light information into a 3D texture. Ideally, the light texture should have the same dimension and orientation as the rendered volume, and a ray caster would update this light texture. Then we would have two 3D textures, where every point in the volume is mapped to the same point in the light texture. Unfortunately, this is intractable with the current graphics hardware. The only value a fragment shader can change is the output value of a fragment, it cannot write to textures at arbitrary places. Because of this, the light map algorithm will have to be a bit more complex. Instead of having a light map that is axis-aligned to the volume, the z-axis of the light map will be aligned along the light direction.

When generating a separate light map for the illumination data, we need a substantial amount of additional memory. Since the light texture in most cases is not axis-aligned with the volume texture, it will be at least the size of the volume texture.

To achieve per-pixel accuracy in the light map, we would therefore need at least the same amount of memory for the light map as for the volume. Because of this, techniques from the deep shadow map implementations by Lokovic and Veach [34], and Hadwiger et al. [17] is used to decrease the amount of needed memory.

The approach that is used is implemented as a multi-pass GPU-accelerated ray caster. It also benefits from techniques like empty space skipping and early ray termination, as described in Section 3.2.2.

5.3.1 Definition

A deep light map is a map of the volumetric illumination as seen from the eye. The light information is represented as a function, the *visibility function* [34], over the depth. A deep light map is similar to a deep shadow map, except the objective is turned around. Instead of only considering what occludes light in the volume, we are also interested in the sources of illumination through the volume. In a deep light map, we deal with one light source that will be the main source of illumination. But if we break it down to small pieces, all particles that scatter light will also act as light sources for the particles around them. This is the main difference between a deep light map and a deep shadow map. In a deep shadow map, the light intensity along a ray from the light source has two options. It will either be attenuated while going through the volume, or it will not interact with any particles and stay unchanged. In a deep light map, the intensity of the same ray could both increase or decrease as it goes through the volume, due to the possibility of in-scattered light. Figure 5.9 shows the difference between a deep light map visibility function and a deep shadow map visibility function.

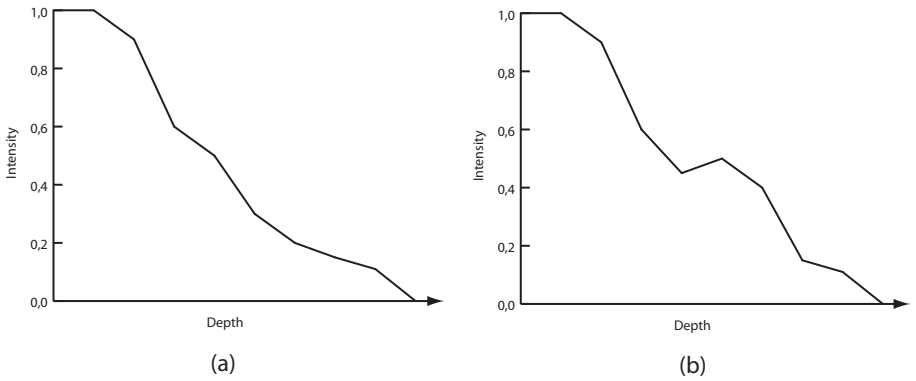


Figure 5.9: *a) A visibility function in a deep shadow map. No in-scattering is considered, and the light is attenuated all the way through the volume. b) A visibility function in a deep light map. Here we also consider in-scattered light, and the intensity of a ray can both increase or decrease on its way through the volume.*

5.3.2 Algorithm

Generating the deep light map is one of the two main passes in the weather visualization algorithm (see Algorithm 1). This pass will again be divided into two steps. First, we calculate the self-shadowing and forward scattering in the volume. This is done by sending rays through the volume, and the light attenuation is accumulated and stored along each ray. For each sample, the forward scattering from the previous sample is also calculated. This results in a visibility function for each of the rays, consisting of an array of control points called *nodes*. Now the visibility function will represent the self-shadowing and central forward scattering in the volume. As explained in Section 5.2.3, we could also calculate the forward scattering from more peripheral angles. This has to be done by generating a separate *scatter volume*, and the deep light map must then be updated to include the peripheral scattering. In the end, the volume will be rendered with a traditional volume renderer from the view point. This last step will be described in Section 5.4.

The self-shadowing and central forward scattering calculation is done using a GPU-accelerated ray caster approach. As the rays go through the volume, the algorithm updates the visibility function at the current (x,y) position. At each sample, the light attenuation T and scattering coefficient β_{sca} is calculated from the mass ratio fetched from the data sets. Then the previous light intensity value is multiplied by the light attenuation value, to get the light intensity after the current sample. The light attenuation and forward scattering is calculated based on Equation 5.11. If there is a change in the light intensity since the previous sample, the visibility function should be updated. The visibility function is updated by adding a depth/intensity pair, to the current array of nodes. When the ray caster is done, we have the light intensity of the entire volume represented as linear splines. Since the visibility function is only updated when the light intensity changes, all of the light information is stored in the first slices of our light map, and all the empty space is in the latter slices. Figure 5.10 illustrates the light intensity through a volume, and the corresponding DLM structure.

Until now, we have only considered the self shadowing and central forward scattering in our deep light map. Since the ray caster will traverse one ray at a time in a depth first approach, it is impossible to calculate the in-scattering from peripheral samples in this pass. The peripheral in-scattering depends on the light intensity of neighboring nodes, which might not yet be calculated. Therefore another pass is introduced to update the light map with peripheral scattering information. This process will be further described in Section 5.3.3.

When a fragment program is run on the graphics processor, like in a GPU-accelerated ray caster, the only output we can get is a color and opacity value for each pixel. Therefore we need a method to represent the values in the deep light map accordingly. This is done by utilizing all the four channels in an RGBA texture. As previously explained, the deep light map stores the visibility function as an array of depth/intensity pairs. Since the texture can store four values, we store two such nodes at each fragment (see Table 5.2). Therefore, the deep shadow map is constructed using a 3D texture, with

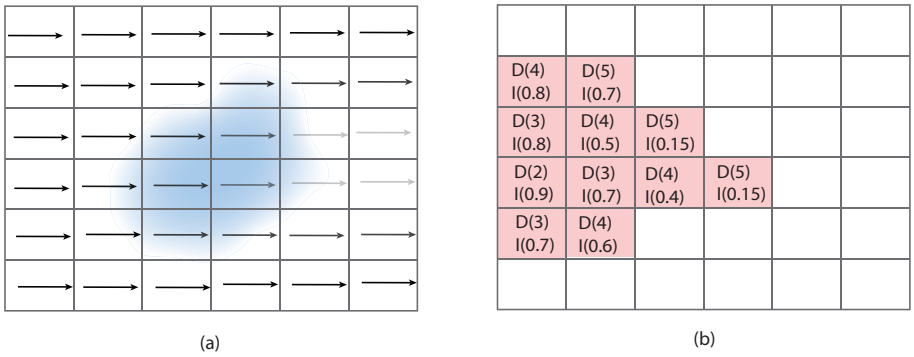


Figure 5.10: *a) Arrows illustrating the light intensity through a volume. The opacity of the arrows decrease as the light is occluded by the matter. b) The same scenario represented as a deep light map. Each node keeps values for intensity and depth. A node is only added for samples where the light intensity changes, therefore all data is stored in the first slices of the light map.*

Node$_i$	R: $depth_i$	G: $intensity_i$
Node$_{i+1}$	B: $depth_{i+1}$	A: $intensity_{i+1}$

Table 5.2: *Two nodes are stored as one RGBA value. Each node represents a control point for the visibility function at the current pixel.*

two control points for the visibility function at each voxel.

To fetch the intensity value at a specific point (x,y,z), we use the x and y coordinates to find the matching visibility function. Then, a search is done through the control points that are stored as nodes. An interpolated value is returned based on the two nodes in front and back of the desired z value.

5.3.3 Light Calculation

For each sample along a ray, the light intensity in the direction from the light source $\overrightarrow{l}$ is calculated. This light intensity is based on the particle/air mass ratios H_s, and the incoming light at the current sample L_{in}. The mass ratios are fetched from one or more attached data sets, depending on the hydrometeor fields that are rendered. From the mass ratios we can calculate the transmittance of the sample T (see Equation 2.13), and by also knowing the light intensity from the previous sample L_{in}^{ctr} we find the transmitted light L_{tr}. The central forward scattering L_{ctr} is calculated from L_{in}^{ctr} and the cumulative phase CP. With the light transport equation given in Equation 5.7, these variables are used to calculate the total light propagation in the volume. But by investigating the light transport equation, it can also be seen that we need a third currently unknown variable, the peripheral light L_{per}. This value should be calculated based on the average light intensity of four equally spaced samples around the ray, L_{in}^{per}.

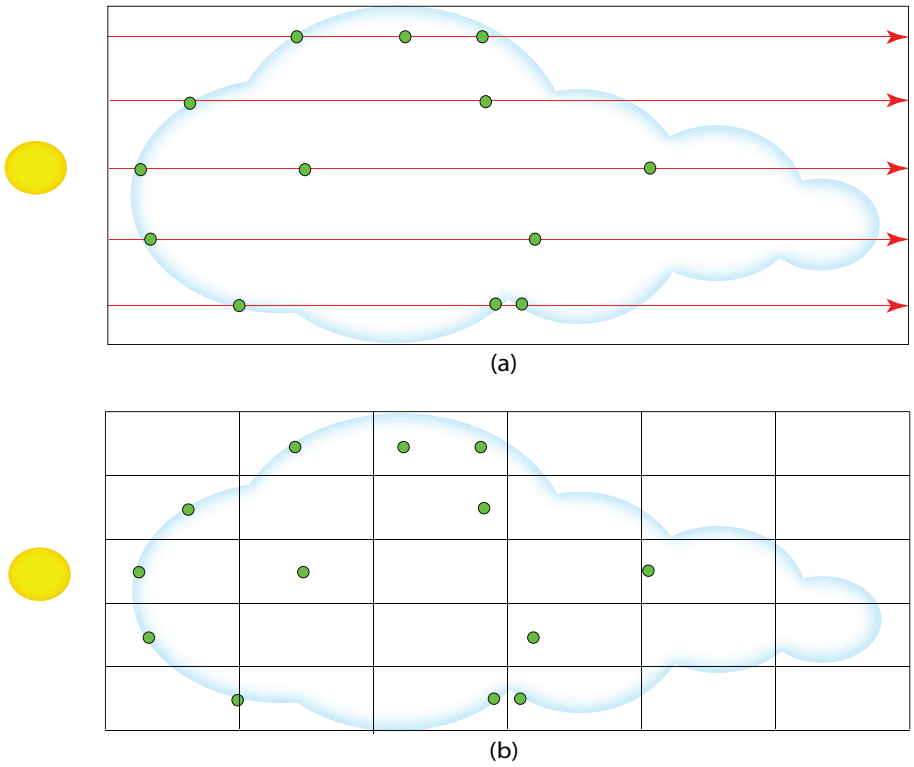


Figure 5.11: *The yellow sphere to the left illustrates the light source. In a) the irregular sampling in the DLM is illustrated by green dots along the red lines, and in b) the low resolution scatter-map is illustrated as a grid over the DLM-nodes*

Due to the depth-first nature of ray casting, these four samples might not be calculated yet. Therefore, we will have to only calculate the light attenuation and central forward scattering in the first pass, with the following stripped down equation:

$$L_l = L_{tr} + L_{ctr} = L_{in}^{ctr} T + L_{in}^{ctr} (1-T) CP(\tfrac{\theta}{4}) \tag{5.13}$$

where θ is the specified forward scattering angle.

To also account for the peripheral in-scattering, an additional pass is required. In the proposed weather visualization system, this is done by using an additional low-resolution scatter-map. The scatter map is a lattice-based volume with the same dimensions and orientation as the DLM. Scattered light is smoother than direct light, since it is randomly distributed to all directions with respect to the phase function. Therefore, a low resolution map is sufficient to create a scattering effect. The scatter-map is generated by slicing the volume in the direction from the light source. The previously rendered DLM is sampled using linear interpolation to get the light intensity for each voxel in the scatter-map. In light intensity values from the DLM, the central scattering

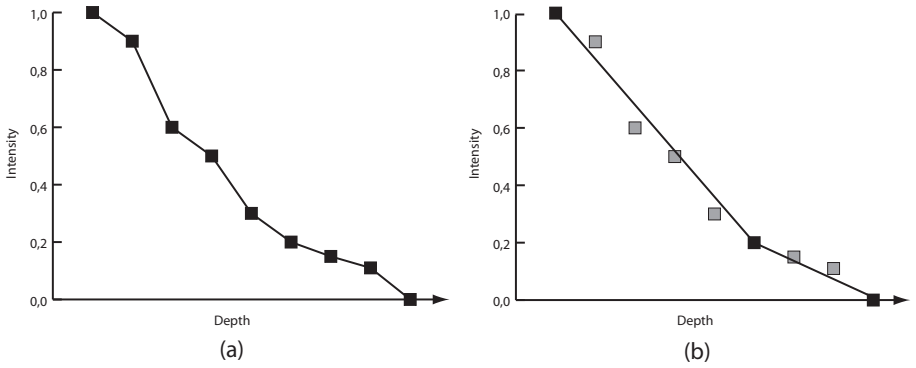


Figure 5.12: *a) Visibility function stored with 9 nodes. b) The same visibility function compressed, using only 3 nodes. All values in between will be linearly interpolated.*

L_{ctr} is already accounted for. Therefore, for each rendered slice in the scatter-map, values from the previous slice is used to calculate the peripheral scattering L_{per} only. This way the scatter-map is built using a greedy algorithm, where each slice depends on the calculations from the previous slice. When the scatter-map is calculated, all nodes in the DLM is updated such that the peripheral scattering is calculated from the scatter map. Figure 5.11 shows how the scatter-map is mapped over the volume. Figure 5.11a shows the irregularly sampled DLM, and Figure 5.11b shows the lattice-based scatter-map over the DLM.

5.3.4 Compression

When storing visibility functions for the deep light map, it might be redundant to store all nodes that differ in light intensity. Therefore, the compression algorithm from Lokovic and Veach [34] is employed. If multiple subsequent nodes in the function have approximately the same gradient, we might as well just store the first and the last of the nodes, since all values in between will be linearly interpolated. This is illustrated in Figure 5.12. By using compression, we achieve similar results using noticeably less memory. This also reduces the cost of doing lookups in the deep light map, as there will be fewer nodes to search through.

The user can specify a value ϵ that represents the maximum allowed deviation from the original path. From this value, we calculate the error bounds that subsequent nodes can lie within. What we want to achieve with this algorithm, is to find the longest possible segment of nodes that lies within the specified error bounds. This resembles trying to pass through as many hoops as possible in one stroke, in a game of croquet. The first sample along the ray will automatically be added as the first node. When the next sample is reached, we need to calculate an upper and lower bound for the allowed deviation. This can be done by calculating the gradient from the previous node to the maximum and minimum allowed values at the current depth. If we have the intensity at the current sample I_c, the intensity at the previous node I_p, the depth of the current

Algorithm 2 Pseudo-code for deep light map compression

```
previousNode = first sample

for samples 2 → do
  if second sample then
    upperBounds = thisUpperBounds
    lowerBounds = thisLowerBounds

  else if current.intensity >upperBounds OR current.intensity <lowerBounds then
    previousNode = previous sample

  else
    upperBounds = min(thisUpperBounds, upperBounds)
    lowerBounds = max(thisLowerBounds, lowerBounds)
  end if
end for
```

sample D_c and the depth of the previous sample D_p, we can get the upper bound UB and the lower bound LB:

$$UB = \frac{I_c+\epsilon-I_p}{D_c-D_p} \tag{5.14}$$

$$LB = \frac{I_c-\epsilon-I_p}{D_c-D_p} \tag{5.15}$$

where ϵ is the allowed deviation. Now all samples are checked against the allowed upper and lower bounds. This is done by calculating the maximum and minimum allowed intensity values at the current depth from the gradients. The maximum allowed value I_{max} and minimum allowed value I_{min} is found:

$$I_{max} = (D_c - D_p)UB + I_p \tag{5.16}$$

$$I_{min} = (D_c - D_p)LB + I_p \tag{5.17}$$

If the current intensity lies between I_{max} and I_{min}, the current sample will be added to the segment. Then the allowed bounds from the previous node to the current sample is calculated. We will now have two maximum and minimum bounds, one that holds for the current sample and one that is for the entire segment. The bounds for the entire segment is then updated by taking the maximum of the lower bounds and the minimum of the upper bounds. If the intensity at the sampled lies outside the error bounds of the segment, the previous sample will be added as a new node. The algorithm then starts over again with a new previous node as the root.

This algorithm is illustrated in Figure 5.13 and in Algorithm 2.

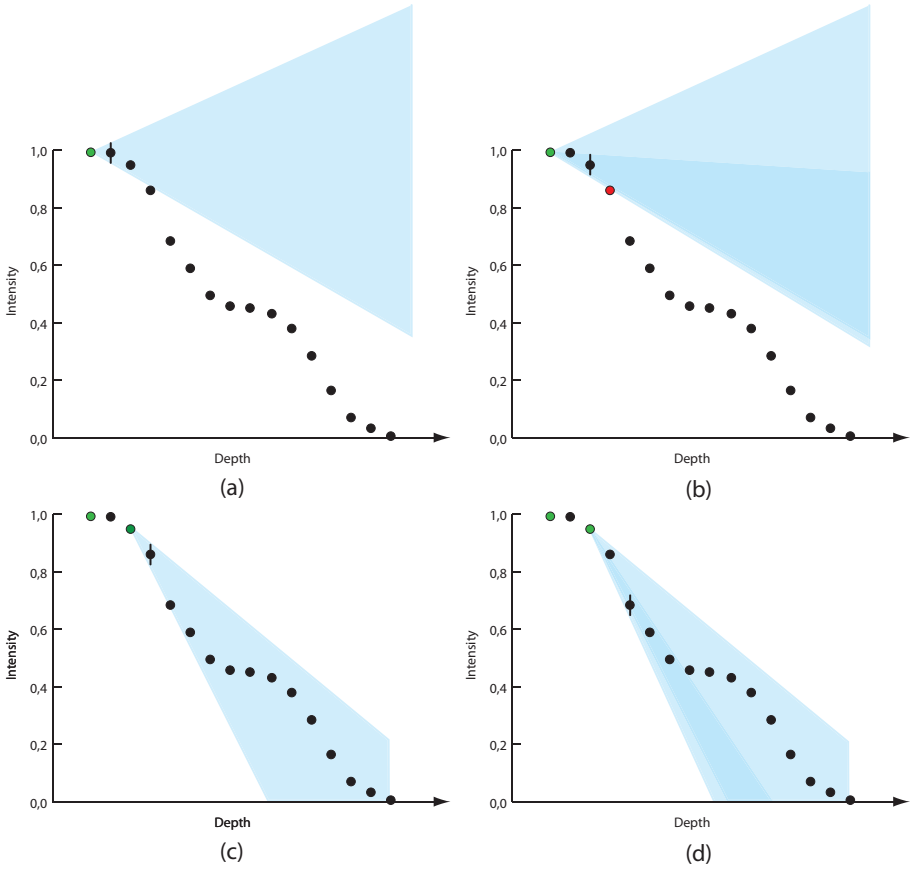


Figure 5.13: *a) First step of the compression algorithm. The first sample is always added as a node, and is depicted in green. Intersection bounds are calculated for the next sample. b) Intersection bounds are calculated for the third sample. Now we can see that the fourth sample, depicted in red, will be out of bounds. The last sample within the bounds are added as a new node. c) and d) The process is started again with the new node depicted as green.*

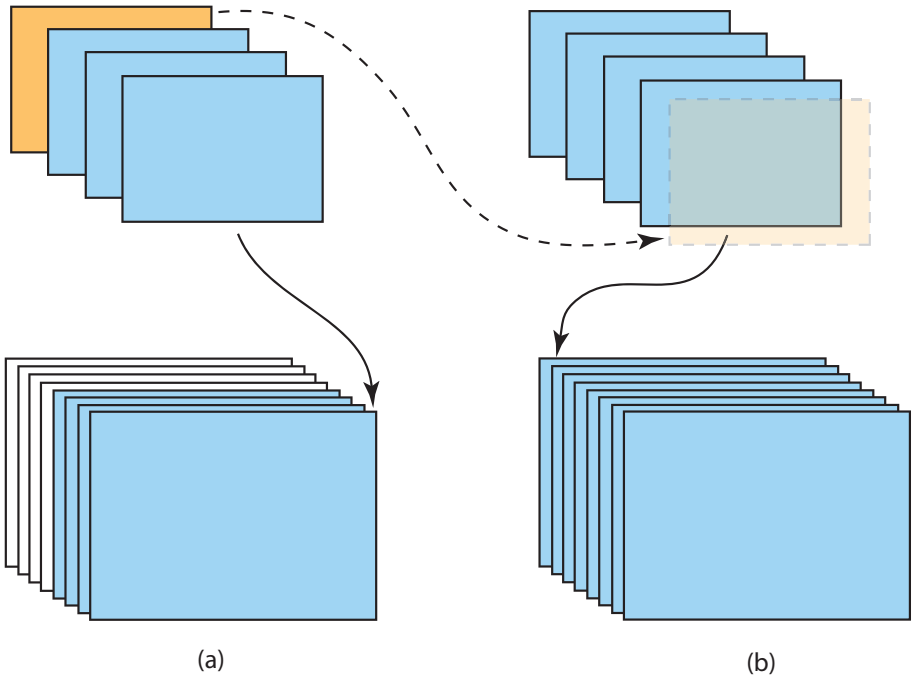


Figure 5.14: *a) The first step in the generation of a deep light map with 8 slices. The first four slices are rendered and copied into the 3D texture. b) The last slice from the previous pass is used as input, to give the ray caster the current depth of each ray. The rendered slices are copied into the 3D texture.*

5.3.5 Deep Light Map Data Structure

The original deep shadow map implementation by Lokovic and Veach [34] used a software approach for the deep shadow map generation. Here, each visibility function was represented as a linked list with control points. Hadwiger et al. [17] used a texture based approach for their GPU-accelerated implementation similar to what is employed in this thesis. In this method, depth layers of a 3D texture are used to store the arrays of nodes. Each node will correspond with one point in space. From a vector (x,y,z), the x and y coordinates will represent the x and y coordinates of the texture. Then, the z coordinate will be represented by the depth value of the nodes. Values that are in-between nodes will be linearly interpolated.

As explained in the introduction to this section, a fragment program that is run on the graphics processor cannot update arbitrary voxels in a 3D texture. Until now, it has also been stated that the generation of a deep light map is done using a ray caster that stores nodes to a 3D texture along each ray. These two statements do clearly not comply with each other. Because of this, the ray caster has to be implemented in an untraditional way. By using multiple passes, the ray caster will have to render slices that are copied into the 3D texture between each pass. Modern graphics hardware has a

Algorithm 3 Pseudo-code for a single deep light map pass

```
passNumber ← Input
previousSlice ← Input

if passNumber == 0 then
   depth = 0
   intensity = 1
else
   depth = previousSlice.depth
   intensity = previousSlice.intensity
end if

glFragColor[0] = vec4(findNextNode(), findNextNode())
glFragColor[1] = vec4(findNextNode(), findNextNode())
glFragColor[2] = vec4(findNextNode(), findNextNode())
glFragColor[3] = vec4(findNextNode(), findNextNode())
```

feature that enables them to render to multiple render targets. By exploiting this feature, we are able to render up to four slices for each pass with current graphics hardware. For each pass, the ray caster needs to know where each ray should start. This is done by using the last of the previous rendered slices as an input texture, since the rendered textures contain a depth value. Figure 5.14 illustrates how a DLM structure is built by rendering four slices in each pass.

In each slice, we are able to store two depth nodes. This is because we use RGBA textures which store four values for each pixel, and each node has two values. Therefore, eight nodes can be stored in each rendering pass if we render to four render targets. Pseudo-code for the algorithm can be seen in Algorithm 3.

5.4 Direct Volume Rendering of Clouds

The previous section introduced a method to generate a separate volume containing light information. In this section, a method to combine the light information with the rendering equation in a direct volume renderer will be described.

5.4.1 Rendering Procedure

In Section 5.2.2, the volume rendering equation that is used in this thesis was introduced. This equation, given in Equation 5.5, is used to get the final color at one rendered pixel. What the equation does, is to integrate over a ray that goes through the volume, to capture the light that is scattered towards the eye from every position along the ray. This equation is very computationally expensive, as it contains a double integral that will consider the light scattering between all possible positions. Equation 5.6 simplifies the volume rendering equation, such that it easily can be solved numerically. What this equation does is to divide the ray going through the volume into several steps, and in this pass only the light attenuation is considered. Since scattering in clouds is dominated by forward scattering, this method only considers multiple scattering in the light transport pass.

The simplification that divides each ray into several discrete steps is quite similar to what is done in a regular ray caster (see Section 3.2). Therefore, a GPU-accelerated ray caster is used for direct volume rendering in this thesis. The GPU-accelerated ray caster will send rays through the volume, and each ray will be sampled at regular intervals. At each sample, the mass ratios from the data sets are fetched, as well as the light intensity value from the deep light map. To fetch a value from the deep light map, we have to know the current coordinates in the deep light map coordinate system. How this is done will be explained in Section 5.4.2. When we have the mass ratios and light intensity, the light attenuation is calculated at each sample. This is done using Equation 2.13. This way we can always keep an accumulated value of the light attenuation from the eye to the current sample. In addition to the light attenuation, we also calculate the scattering coefficient for the sample. This was described in Section 2.2.2. Having the scattering coefficient, a phase function is used to find the amount of light that is scattered towards the eye. By knowing both the total light attenuation from the eye to the sample, and also the amount of light scattered from the sample toward the eye, we can update the current pixel with the light that reaches the eye from the current sample. When the ray reaches the end of the volume, the height coordinate is checked to find if this point lies on the surface of earth. If it does, we should add a shadow value if we want the land to be shadowed by the cloud or hurricane. The shadow is added by first checking the light intensity, and then adding the amount of shadow to the alpha channel.

Figure 5.15 shows four states of the ray caster. First, the ray is initiated at the closest point of the volume. Second, the ray is sampled through the volume, and the light scattered towards the eye is calculated. Third, the ray reaches the end of the

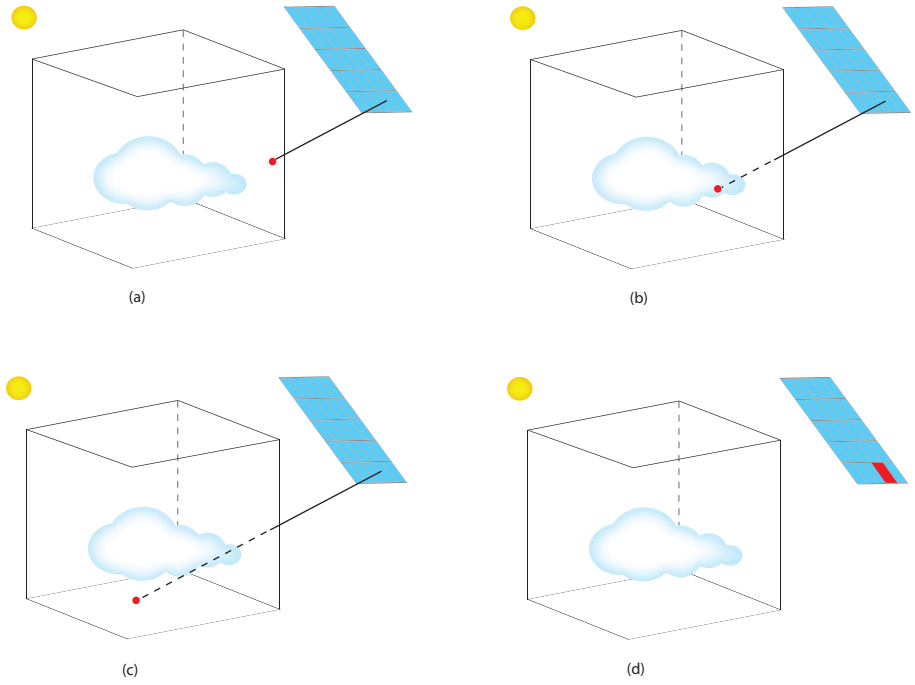


Figure 5.15: *a) A ray is initiated at the red dot, starting at the closest point of the volume. b) The dashed line traverses the volume. For each sample, light intensity is fetched from the deep light map, and light attenuation is calculated. c) The ray reaches the end of the volume. If the point is on the surface of earth, and it is shadowed, the shadow value is added to the alpha channel. d) The pixel is updated with the new color.*

volume. If the point lies in shadow and is in the surface of earth, we add shadow. In the end, the pixel is updated with the new color.

5.4.2 Mapping to Deep Light Map Coordinates

As was explained in Section 5.3, the deep light map will have a different coordinate system than the volume we are rendering. During the ray casting, it is needed to access the same point in space both in the volume and in the light map. This can be done by doing a matrix multiplication, to get the deep light map coordinates from the volume coordinates. But it would be computationally expensive if this were to be done at every spatial position in the volume. Therefore color maps are used, similarly to the ones used in GPU-accelerated ray casting. When the color maps are drawn for the ray caster, the coordinates of the bounding box are mapped to the coordinate system of the volume, such that each corner represents a corner of an RGB cube. This can be done by specifying the vertices of the cube with coordinates in the range [0,1], such that each corner has the same coordinate as the corresponding volume corner. Then the shaders can specify RGB colors according to the coordinates (x,y,z). The

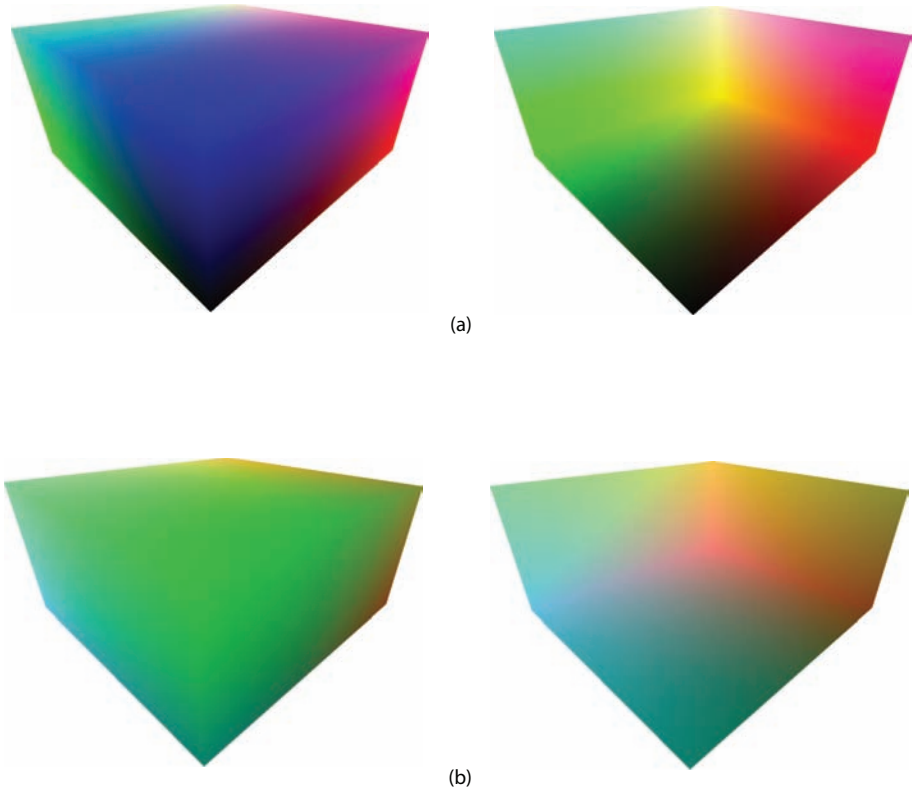


Figure 5.16: *a) Regular color maps for the volume. b) Color maps for the deep light map.*

generation of color maps for the DLM could be done in a similar manner, except the coordinates that represent the RGB colors are mapped to form a fictive RGB cube with the size and orientation of the DLM. This is done by first rendering the same vertices to shape the box. But now we need to transform the (x,y,z) coordinates that give the final color to the DLM coordinate system. The coordinates are first multiplied by the DLM transformation matrix, which is the matrix that can used to move the camera to the light source. After this multiplication the coordinates will be transformed to the correct positions, but they are not yet scaled to fit the size of the DLM. Therefore one more multiplication is needed, this time by a scale matrix that will move the vertices to the [0,1] range of the DLM. Figure 5.16 shows the result of the color-map generation process. The front and back color maps for the volume are shown, together with the corresponding color maps for the DLM.

When we have color maps for both the volume and the deep shadow map, the ray caster knows the starting point and the stopping point in both coordinate systems. Then two position vectors are kept during the ray traversal, one for the current position in the volume and one for the current position in the deep light map. For every step,

each position vector is incremented by the same physical distance in their respective coordinates.

5.4.3 Light Calculation

In the direct volume rendering of clouds, we want to solve the volume rendering equation that was introduced in Section 5.2.2. The equation is given as:

$$L(e) \quad = \quad T(0,e)L_{bg} + \sum_{s=1}^{e} T(e,s)(1-T(s-1,s))P(\psi(\Omega))L_l(s,\Omega) \qquad (5.18)$$

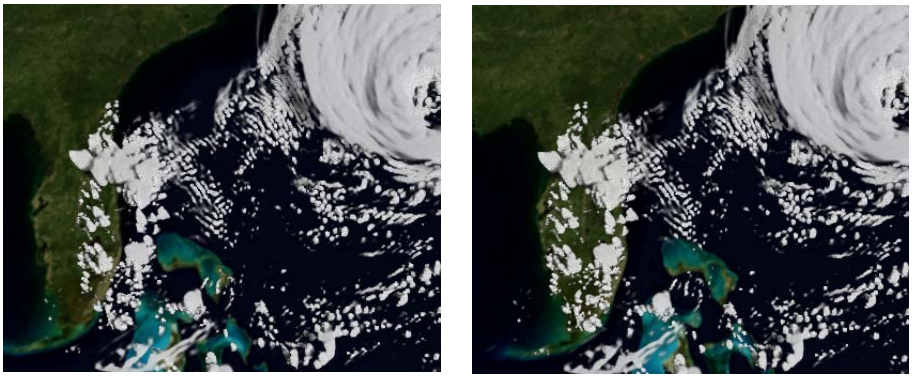
where $T(s,w)$ is the transmittance between samples s and w, L_{bg} is the background light, $P(\psi(\Omega))$ is the phase function for the angle between the ray and the light direction Ω, and $L_l(s,\Omega)$ is the light intensity at sample s in direction Ω.

The first part of the equation is the transmitted light from the background. If a hurricane is seen from above, this part of the equation will be represented by the potentially visible earth beneath the hurricane. In the proposed rendering system, this will be the texture that is rendered below the hurricane in a pre rendering step that is visible through the hurricane. The next part of the equation is a summation of the visible light scattered from all samples along a ray. This part will be taken care of by the ray caster. At each sample along the casted rays, we will calculate the amount of light scattered towards the eye from that sample. The transmittance T is the transparency of the current sample, as introduced in Section 2.2.6. Since we assume the single scattering albedo to be 1, all the light that is not transmitted is scattered. Hence the amount of scattered light is $1-T$. The variable L is the light intensity fetched from the DLM, as explained in the previous section. P is the phase function. It takes the angle between the viewer and the light source $\psi(\Omega)$ as a parameter. The phase function is pre-generated and stored as a one-dimensional texture, where the values are clamped between [0,1]. Therefore, the coordinate for the phase value is $\frac{\psi(\Omega)}{\pi}$.

5.5 Embedding Into Context

Until now, the main concern of this thesis has been illumination and rendering of weather phenomena. This is an important aspect when we aim to approximate photo realistic images of phenomena such as hurricanes, but what is also important is to capture the environment in which the phenomenon resides. Because of the large scale of hurricanes, satellite photos of the earth is a good source for the surrounding environment. In this thesis, the satellite photos from NASA's *Blue Marble* [37] have been used. These are high resolution data sets, that are free to use and redistribute. Section 5.5.1 will describe how a phenomenon can be placed correctly on a satellite photo, based on longitude and latitude values.

The volumetric scale of the phenomenon is another important aspect of the proposed weather visualization system. First of all, the aspect ratio of the axes in the



(a) (b)

Figure 5.17: *Clouds over Florida rendered with a) inaccurate longitude positioning, and b) manually tuned longitude positioning.*

volume may differ from the physical distances of the weather phenomenon. This can easily be solved automatically for the data sets that contain meta data on the physical size. But, in some cases it is also desirable to visualize the phenomenon with a different aspect ratio than in the real world. For example, hurricanes range over a very large area, and the height is relatively small compared to the circumference. Therefore, it may be desirable to exaggerate the height to get a better perception of the phenomenon. Section 5.5.2 will describe how this is done.

To simplify the volume rendering, this thesis has rendered hurricanes on the satellite photos as flat images. Although it is not in the scope of this thesis, Section 7.2 will give some ideas on how this system can be extended to consider the spherical shape of earth.

5.5.1 Placement on Earth

In the proposed system, the surrounding environment is first rendered in a pre-processing step, and the weather phenomenon is then rendered on top of the environment. By doing this, the background will automatically be blended with the phenomenon just as required by the volume rendering equation that was discussed in Section 5.4.3. Due to the high resolution of the blue marble data sets, the satellite photo is split up into several pieces that fit as one texture on the graphics card. These pieces are then rendered as a flat image.

Next, the positioning of the weather phenomenon onto the satellite photo is calculated. The blue marble images span from longitude -180 to 180 degrees, and latitude -90 to 90 degrees. By knowing the longitude and latitude positions of the volume, we can now easily position the volume accordingly. However, some volumes may have inaccurate longitude and latitude values, for example by neglecting the fact that the longitude may change due to the shape of earth. Therefore it is required to be able to fine tune the positioning in such cases. Figure 5.17 shows two images, one where the

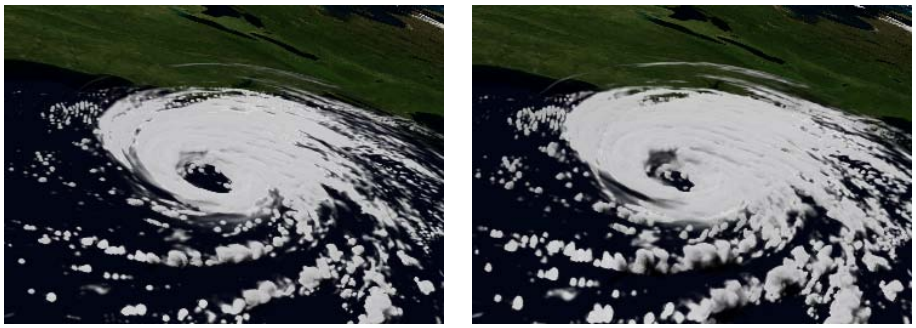


Figure 5.18: *Two images rendered with different height scale. In image b) the height is exaggerated, and the structure can be seen more clearly than in a).*

hurricane Isabel is incorrectly placed according to the satellite photo due to inaccurate longitude values, and one where the longitude value is manually tuned. In this particular data set the inaccuracy can be clearly seen from the clouds following the coast of Florida.

5.5.2 Scaling

In order to achieve the desired visual perception of a weather phenomenon such as a hurricane, it might be necessary to manipulate the result to some extent. To change the aspect ratio of the axes is one way to manipulate the result, such that it gives a better perception. Due to the low distance in height compared to the width and breadth of a hurricane, a visualization with the real physical aspect ratio will look very flat. In the proposed system, one can interactively scale the volume in all three directions to achieve the desired result.

The flexibility of a GPU-accelerated ray caster makes this a trivial extension. A scale-matrix is used to transform the volume while generating the color maps. This way the bounding object in the color maps will have the desired scale, and no other changes are needed to the system.

Figure 5.18 illustrates the effect we can achieve by manipulating the height scale of a hurricane.

5.6 Pre-Calculated Extinction Volumes

In Section 5.1, the WRF data sets that are utilized in the proposed system were introduced. In their original form, these data sets are stored as multiple voxelized volumes where each volume represents one time step of a single hydrometeor field. Since multiple data values are stored for each voxel, a large amount of data has to be considered for every rendered frame. In an interactive visualization system it is crucial to keep the

operations required to render a frame as simple as possible, since the rendering will be repeated continuously. Therefore, a method is introduced that enables pre-calculation of important values in order to simplify the rendering of each frame.

In Section 5.2.1 it was stated that the extinction coefficient β at each voxel is the foundation for all calculations that are done by the visualization system. Therefore, it is possible to base the rendering step on the extinction coefficients rather than the particle/air mass ratios in the original data sets. If the extinction coefficients are calculated in a pre-processing step, it is possible to avoid the same calculation to be repeated every time a frame is rendered. By using pre-calculated extinction coefficients it is also not required to differentiate between all the hydrometeor fields in the data set. In fact, the only property that may vary between different hyderometeor fields after the extinction coefficient is calculated is the scattering phase function. In the proposed system we use two different scattering phase functions, one that accounts for ice particles and one for water particles. By exploiting this knowledge, we can generate custom data sets by combining the extinction coefficients for ice particle fields and water particle fields respectively. This results in a method where only two volumes are needed for every time step, as compared to keeping one volume for every hydrometeor field (usually five fields).

By utilizing this optimization method, there are some obvious advantages. The memory usage will be greatly reduced in situations where there are more than two hydrometeor fields. This reduces the time it takes to load a time step, which in turn improves the time-variation feature of the system. Since only two voxelized volumes are required, the number of texture lookups for each voxel is also reduced, such that the overall frame rate of the system is improved. It is also trivial to extend the system to use this optimization method. When generating the extinction volumes in a pre-processing step, the performance is not crucial. Therefore, it can be done with a batch tool that converts the data sets.

However, there are also downsides of this optimization method. Since the hydrometeor fields are combined to form extinction volumes for ice and water particles, it is impossible to enable or disable single hydrometeor fields after the volumes have been generated. The method also requires a pre-processing step, that stores the data in a non-standard format. Thus, it is not possible to directly import WRF data sets without first converting them with a separate tool.

6 Results

The previous chapter presented the methods that are used for weather visualization in the proposed system. In this chapter, the results that are achieved with this system will be presented. There are several ways to measure the results for a weather visualization system. They can for example be measured by the performance and frame rates that are achieved by the system, the correctness of the light transport, or the visual accuracy of the results. This chapter will therefore be divided into three sections. The first section will discuss the validity of the light calculation, the second section will consider the visual results, and the third section will discuss the performance results. In each section, the relevant problem statements that were introduced in Section 1.2 will be brought up, and the solutions proposed by this thesis will be discussed.

6.1 Illumination

In Section 1.2, the requirements for an illumination algorithm for clouds were discussed. It was pointed out that a global illumination model was needed, that accounts for both shadowing and scattering of light. The method that is presented in this thesis is based on simplifications that were introduced by Nishita et al. [39]. With this method, only light attenuation and multiple forward scattering from a pattern of incident samples is considered. The validity of such a simplification can be discussed, but as pointed out by Nishita et al. and several others [7, 18], the forward scattering in clouds is dominant enough to neglect scattering in other directions. Thus, the results will clearly be approximations to the real world, but simplifications are inevitable in order to maintain computational tractability.

Figure 6.1 shows an illustration of the illumination effects achieved with this visualization system, where a semi transparent sphere is lit, casting shadows to a surface. Figure 6.2 illustrates the forward-scattering in a sphere, where the sphere is occluding the light source similarly to a solar eclipse. The outer edges seems bright from the scattered light, while the center of the sphere is dark since no light scatters all the way through.

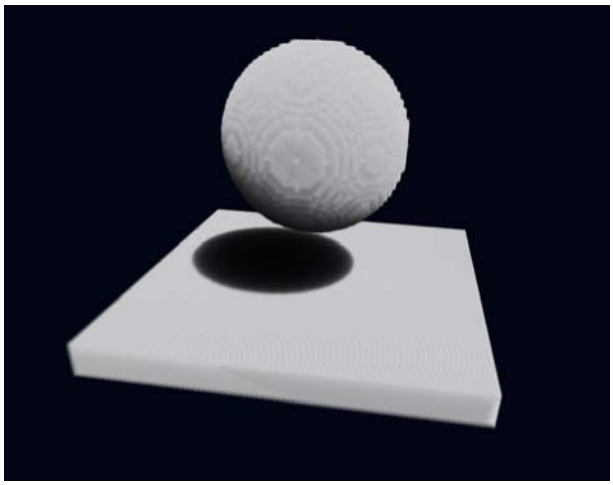


Figure 6.1: *A sphere rendered with the proposed system to illustrate the illumination effect. Soft shadows are seen on the underlying surface, and self-shadowing is visible on the sphere.*

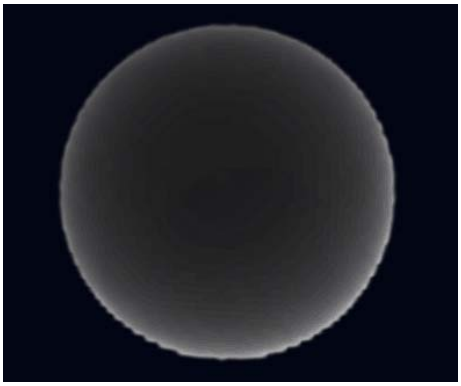


Figure 6.2: *A sphere rendered in front of the light source, similarly to a total solar eclipse. Light is scattered through the media around the outer edges of the sphere.*

6.2 Visual Results

The problem statements in Section 1.2 pointed out that the visual results should imitate the illumination of clouds in a physically plausible manner, which was covered in the previous section. It was also stated that the rendered weather phenomena should be embedded into a realistic context.

In the scope of this thesis, data sets of hurricanes Isabel and Katrina have been used, and images from NASA's Blue Marble [37] have been utilized as the *realistic context*. Thus, both the meteorological data and the environment are based on real scenarios.

In order to evaluate the visual results from the proposed system we consider Figure 6.3 where the hurricane Isabel is rendered at several discrete time steps. Figure 6.3a and b shows the earliest stages of the hurricane build-up, where water vapor and clouds are starting to form the shape of a hurricane. At this stage, the clouds are thin and unstructured. In Figure 6.3c the shape is getting more distinct, and Figure 6.3d,e, and f shows how the hurricane evolves with approximately 10 hours between each frame. In the later time stages we can see cloud structures clearly. For example, we can see patterns of low-altitude cumulus clouds as regularly spaced dots, while the high-altitude cirrus clouds are thin and wispy. Note that the position of the sun is fixed through all time steps. This is to avoid confusion since a light change can give the impression of a different shape, and because the hurricane would not be visible during the night.

As previously stated, to embed the visualized weather phenomena into a realistic context, *Blue Marble* images are used to represent the Earth. For simplicity, the images are rendered as a flat surface in the proposed system. In order to integrate the weather phenomena to the environment in a realistic manner, it is also important that the weather phenomena affects the shading of the earth. This is done by adding shadow to the earth, which enables the weather phenomenon to blend in smoothly with the environment. Figure 6.4 shows an illustration of this, where Figure 6.4a shows hurricane Isabel rendered with the light source approximately 90 degrees towards the viewing angle, and Figure 6.4b rendered with the light source behind the viewer. This illustration also shows the effect of the phase functions, by favoring back-scattering over side-scattering. This can be seen by comparing the brightness of Figure 6.4b to the slightly darker Figure 6.4a.

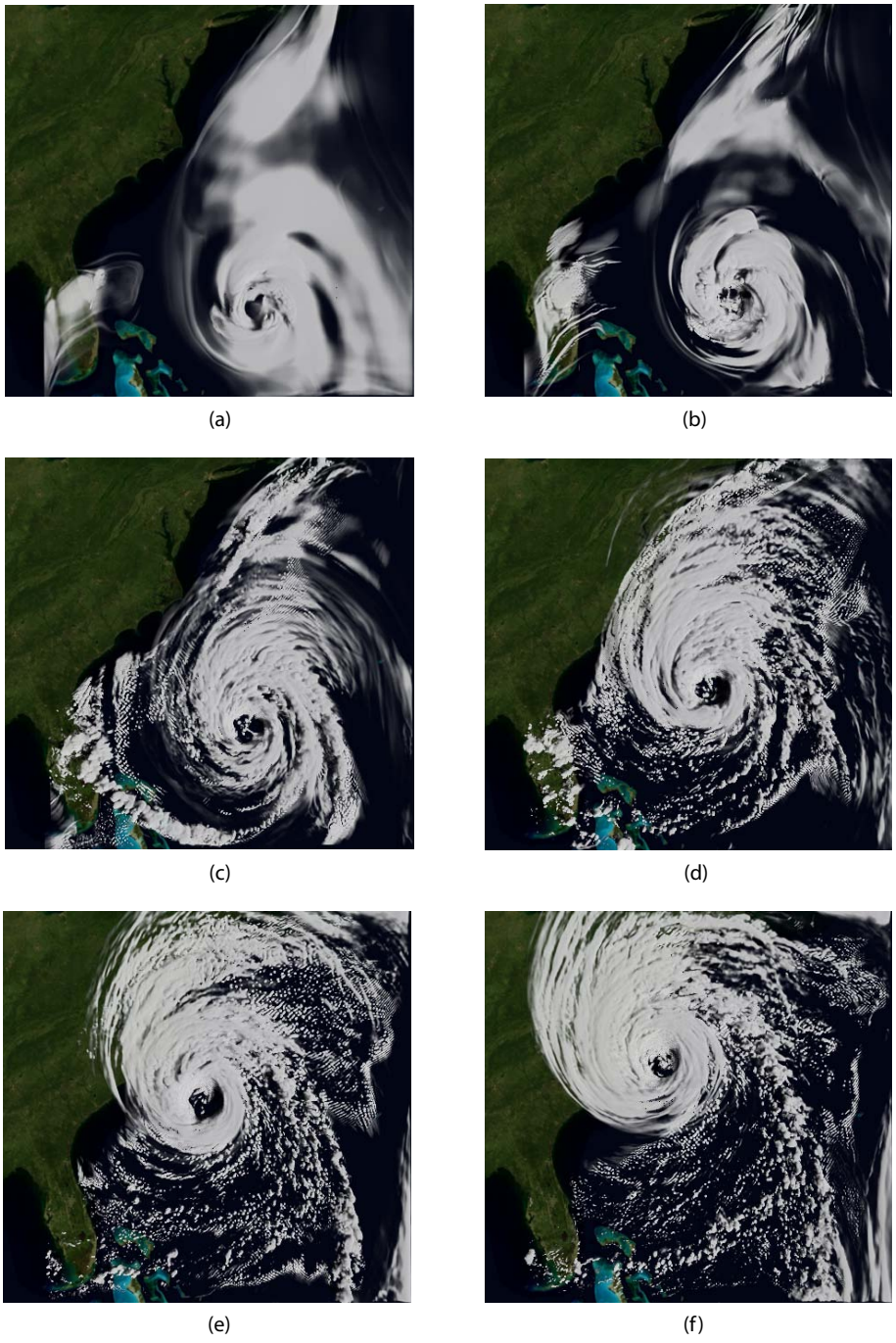
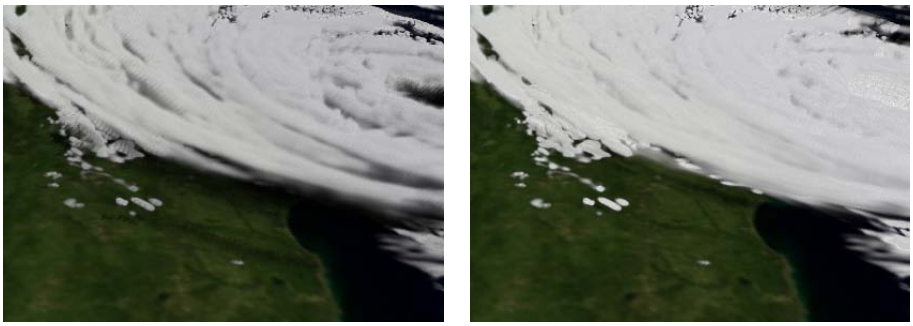


Figure 6.3: *Six separate time steps of the hurricane Isabel data set.*



(a) (b)

Figure 6.4: *Two renderings, showing the same cut with different light source position. In a) the shadows can be seen on the ground, in b) the light source is behind the viewer thus the shadows are less visible. These images also show clearly the effect of the phase functions. More light is back-scattered than side-scattered, therefore b) has a brighter appearance than a).*

6.3 Performance

The last aspect of the proposed weather visualization system that is evaluated is the performance. From Section 1.2, it was stated that the proposed system should be able to render the visualizations at interactive frame rates. The term *interactive frame rates* is not very precise. Therefore, in this thesis we state that a system that gives at least one frame per second can be handled interactively. The benchmarked visualization system was implemented as a plugin to the Volumeshop framework [51].

The system that was used for performance measurements has the following technical specifications:

- Windows XP operating system
- AMD Athlon 64 X2 3800+ CPU
- 2 GB RAM
- GeForce 9600GT, 512 MB video memory

The performance benchmarks has been executed with different properties enabled, to compare the performance results. Table 6.1 gives the performance results using standard WRF data sets, and the pre-computed extinction volumes. The same data set with two different resolutions were tested, the Isabel data set with 500x500x100 voxels (HR), and the Isabel data set reduced to 250x250x50 voxels (LR). Both data sets contain five hydrometeor fields: cloud, ice, rain, graupel, and snow. One time step in the high resolution data set is approximately 500 megabytes of data, while one time step in the low resolution data set is approximately 30 megabytes. The resolution of the rendered images was 512x512 pixels. A deep light map with a resolution of 512x512x12 voxels was used. Frame rates are captured with the benchmarking application *Fraps* [14], and are only approximates to give an idea of the interactivity.

	WRF Volumes (FPS)	Extinction Volumes (FPS)
HR Volume	13	25
HR Volume + Light	5	10
HR Volume + Light + Time	1	2-3
LR Volume	35-40	35-40
LR Volume + Light	12	15
LR Volume + Light + Time	2-3	4-5

Table 6.1: *The table shows that pre-computation of extinction volumes gives a significant improvement with large data sets (HR), while the difference is relatively insignificant when the data sets are smaller (LR). It is also clear that the deep light map generation is considerably time consuming on both high and low resolution data sets.*

From the table it is clear that pre-computation of attenuation volumes gives a significant improvement for large data sets, while for smaller data sets the difference is less notable. When time change is involved, the extinction volumes also requires far less data to be read, which results in better performance compared to standard WRF volumes. Therefore, in situations where it is possible to pre-generate the extinction volumes, this would be desirable in order to achieve full interactivity.

7 Summary

In this thesis, a weather visualization system has been described and implemented. The system includes an advanced illumination model for weather data, and a method to handle multi-variate data at multiple time steps. If we consider the problem statements from Section 1.2, the goals that are fulfilled in this thesis are:

- approximate the light transport in clouds in a physically plausible manner
- approximate the light scattering from clouds towards the viewer
- embed the volume in a realistic context
- visualize the change of the phenomena over time
- render the results at interactive frame rates

In this chapter, the major results of this thesis will be summarized. Section 7.1 will give a conclusion of the work that is covered in this thesis, while possible ideas for future work will be introduced in Section 7.2.

7.1 Conclusion

The proposed visualization system aims at visualizing simulated weather data sets. For this purpose, data sets simulated and stored using the WRF model has been utilized. The WRF model is a model for mesoscale weather phenomena, which means that the phenomena range from five kilometers to several hundred kilometers in physical size.

In the physical size lies one of the main obstacles when developing a visualization system for WRF data sets. Since phenomena such as hurricanes are very large in physical size, the data sets will require a very high resolution to capture a certain amount of detail. This is not only a problem because of the large amounts of data that needs to be handled, the fact that each sample itself will be very large in physical size is also a concern. In many situations, the size of one sample can be several kilometers in each direction. This leads to one of the problems: as the weather data is treated as translucent matter, each sample should not be fully opaque. Therefore, each sample has to be small enough such that some of the light is transmitted through the sample. If not, the

rendering would be similar to rendering of a opaque surface. Therefore, the data set has to be sampled at short intervals to be able to capture the shadow structures.

Another problem with the large physical scale is that large amounts of details are lost. Since each sample is linearly interpolated between voxels, the transitions will be very smooth. Therefore, small variations in light and structure are lost in the visualizations.

Considering the obstacles that lie in the visualization of large scale weather phenomena, the results achieved in this thesis have been very successful. The shadow structures in the clouds are clearly visible and yield a certain realistic appearance. The semi-transparent nature of clouds are also apparent, from the clear soft shadows. The use of Blue Marble images improve the realism, providing a realistic context for the phenomena.

7.2 Future Work

The weather visualization system that has been proposed in this thesis builds a foundation for a variety of future work. This section will present ideas for possible future work, that will extend the functionality or improve the visual results of the proposed system.

In the scope of this thesis, we have considered all light as white. Clearly, this is not the case in the real world. Scattering of light from particles in the atmosphere causes effects that change the color of sunlight. This can be seen during sunset or sunrise, when scattering gives the sunlight a red shade. A weather visualization system could also take this into account, and color the weather phenomena according to the color of the sunlight. Also, the phase functions in the proposed system treat all incoming and outgoing light as white. To improve the realism of the results, the proposed system could be extended to consider colors in the phase functions. This could enable effects such as rainbows and halos in clouds and media with very small particles.

Another extension that would greatly improve the realism of the visualizations is to consider the curvature of the earth in the data sets. Originally, the WRF data sets are represented as cubic volumes. However, by knowing the length of each direction, and the dimensions of the earth, it would be possible to map the data set onto a sphere. This would make it possible to view the earth as a sphere textured with satellite photos, and with the visualization positioned on the earth. In the proposed system, this could be achieved by modifying the way the ray caster color maps are generated, such that the volume is represented as a curved box. The ray casting would also have to be modified such that the volume is accessed with spherical coordinates.

As discussed in Section 7.1, the visualizations may suffer from a low level of details in the data sets. This causes a certain smoothness to the visualizations that can be undesirable, since the real weather phenomena can be a lot more unstructured. In other systems, this has been solved by using texture perturbation [45]. The visualization system proposed in this thesis could also be extended to support texture perturbation.

This would add some random noise to the images, which would imitate the randomness that characterize natural weather phenomena.

In the proposed weather visualization system, a large amount of memory is needed. In addition to the data sets, the system uses an additional volume for the light data. This light-volume uses a simple data structure with a considerable amount of overhead. By implementing more advanced compression techniques, the size of the additional light-volume could be reduced.

Since the proposed system has been carried out mainly with television broadcasters in mind, it has been implemented in an easily portable manner. In cooperation with the broadcast graphics company VizRT, there are plans to implement the system in their OpenGL based graphics framework. With such an implementation, many of the largest television broadcasters are within reach, providing a large audience for the system.

Acknowledgements

I would like to thank the people at VizRT for their interest and the time they invested in the project. Their participation has been a great inspiration through the entire process, from the initial project idea to the finalized thesis.

References

[1] Uwe Behrens and Ralf Ratering. Adding shadows to a texture-based volume renderer. In *IEEE Symposium on Volume Visualization*, pages 39–46, 1998.

[2] James F. Blinn. A generalization of algebraic surface drawing. *ACM Transactions on Graphics*, 1(3):235–256, 1982.

[3] Craig F. Bohren. Multiple scattering of light and some of its observable consequence. *American Journal of Physics*, 55(524–533), 1987.

[4] Craig F. Bohren and Donald R. Huffman. *Scattering of Light by Small Particles*. John Wiley and Sons, 1983.

[5] Brian Cabral, Nancy Cam, and Jim Foran. Accelerated volume rendering and tomographic reconstruction using texture mapping hardware. In *SIGGRAPH'94 Proceedings*, pages 91–98, 1994.

[6] Michael F. Cohen and John R. Wallace. *Radiosity and Realistic Image Synthesis*. Academic Press, Boston, 1993.

[7] William M. Cornette and Joseph G. Shanks. Physically reasonable analytic expression for the single-scattering phase function. *Applied Optics*, 31(16):3152, 1992.

[8] Franklin C. Crow. Shadow algorithms for computer graphics. *Computer Graphics*, 11(2):242–248, 1977.

[9] David S. Ebert. Volumetric modeling with implicit functions: a cloud is born. In *SIGGRAPH'97 Proceedings*, page 245, 1997.

[10] David S. Ebert, F. Kenton Musgrave, Darwyn Peachey, Ken Perlin, and Steven Worley. *Texturing and Modeling, a Procedural Approach*, chapter 19. Morgan Kaufmann, 2002.

[11] David S. Ebert and Richard E. Parent. Rendering and animation of gaseous phenomena by combining fast volume and scanline A-buffer techniques. In *SIGGRAPH'90 Proceedings)*, volume 24, pages 357–366, 1990.

[12] Klaus Engel, Markus Hadwiger, Joe M. Kniss, Aaron Lefohn, Christof Rezk-Salama, and Daniel Weiskopf. Real-time volume graphics. In *Course 28 at ACM SIGGRAPH*, 2004.

[13] Brad Schoenberg Ferrier, Wei-Kuo Tao, and Joanne Simpson. A double-moment multiple-phase four-class bulk ice scheme. *Journal of the Atmospheric Sciences*, 51:249–280, 1994.

[14] Fraps. http://www.fraps.com.

[15] Qiang Fu and Kuo-Nan Liou. Parameterization of the radiative properties of cirrus clouds. *Journal of the Atmospheric Sciences*, 50:2008–2025, 1993.

[16] General-purpose computing on graphics processing units. http://www.gpgpu.org.

[17] Markus Hadwiger, Andrea Kratz, Christian Sigg, and Katja Bühler. Gpu-accelerated deep shadow maps for direct volume rendering. In *GH '06: Proceedings of the 21st ACM SIGGRAPH/Eurographics symposium on Graphics hardware*, pages 49–52, 2006.

[18] Mark J. Harris. Real-time cloud simulation and rendering. Technical Report TR03-040, 2003.

[19] Mark J. Harris. *Real-Time Cloud Simulation and Rendering*. PhD thesis, University of North Carolina, 2003.

[20] Mark J. Harris and Anselmo Lastra. Real-time cloud rendering. *Computer Graphics Forum*, 20(3), 2001.

[21] Louis G. Henyey and Jesse L. Greenstein. Diffuse radiation in the galaxy. *The Astrophysical Journal*, 90:70–83, 1941.

[22] Luke Howard. *On the Modifications of Clouds*. J. Taylor, London, 1804.

[23] IEEE visualization 2004 contest. http://vis.computer.org/vis2004contest/.

[24] James T. Kajiya. The Rendering Equation. In *SIGGRAPH'86 Proceedings)*, volume 20, pages 143–150, 1986.

[25] James T. Kajiya and B.P. Von Herzen. Ray tracing volume densities. *SIGGRAPH*, 84:165–174, 1984.

[26] Jeffrey R. Key, Ping Yang, Bryan A. Baum, and Shaima L. Naisiri. Parameterization of shortwave ice cloud optical properties for various particle habits. *Journal of Geophysical Research (Atmospheres)*, 107:4181–+, 2002.

[27] Joe Kniss, Simon Premoze, Charles Hansen, and David Ebert. Interactive translucent volume rendering and procedural modeling. In *VIS '02: Proceedings*, pages 109–116, 2002.

[28] Joe Kniss, Simon Premoze, Charles Hansen, Peter Shirley, and Allen McPherson. A model for volume lighting and modeling. *IEEE Transactions on Visualization and Computer Graphics 2003*, 9(2):150–162, 2003.

[29] Jens Krüger and Rüdiger Westermann. Acceleration techniques for GPU-based volume rendering. In *IEEE Visualization*, pages 287–292, 2003.

[30] Philippe Lacroute and Marc Levoy. Fast volume rendering using a shear-warp factorization of the viewing transformation. In *SIGGRAPH'94 Proceedings*, pages 451–458, 1994.

[31] Marc Levoy. Display of surfaces from volume data. *IEEE Computer Graphics and Applications*, 8(3):29–37, 1988.

[32] John-Peter Lewis. Algorithms for solid noise synthesis. In *SIGGRAPH'89 Proceedings*, pages 263–270, 1989.

[33] Kuo-Nan Liou, J. L. Lee, Szu-Cheng Ou, Qiang Fu, and Yoshihide Takano. Ice cloud microphysics, radiative transfer and large-scale cloud processes. *Meteorology and Atmospheric Physics*, 46:41–50, 1991.

[34] Tom Lokovic and Eric Veach. Deep shadow maps. In *SIGGRAPH '00 Proceedings*, pages 385–392, 2000.

[35] Nelson Max. Optical models for direct volume rendering. *IEEE Transactions on Visualization and Computer Graphics*, 1(2):99–108, 1995.

[36] Gustav Mie. Bietage zur optik truver medien speziell kolloidaler metallosungen. *Annallen der Physik*, 25(3):377, 1908.

[37] NASA Blue Marble. http://earthobservatory.nasa.gov/Newsroom/BlueMarble/.

[38] The national center for atmospheric research. http://www.ncar.ucar.edu.

[39] Tomoyuki Nishita, Yoshinori Dobashi, and Eihachiro Nakamae. Display of clouds taking into account multiple anisotropic scattering and sky light. *Computer Graphics*, 30(Annual Conference Series):379–386, 1996.

[40] Manjushree Nulkar and Klaus Mueller. Splatting with shadows. In *Volume Graphics*, 2001.

[41] Ken Perlin. An image synthesizer. *Computer Graphics*, 19(3):287–296, 1985.

[42] Hans R. Pruppacher and James D. Klett. *Microphysics of clouds and precipitation*, volume 18. Kluwer Academic, 2 edition, 2000.

[43] Timothy J. Purcell, Ian Buck, William R. Mark, and Pat Hanrahan. Ray tracing on programmable graphics hardware. In *SIGGRAPH'02 Proceedings*, pages 703–712, 2002.

[44] William T. Reeves. Particle systems - a technique for modeling a class of fuzzy objects. *ACM Transactions on Graphics*, 2(2):91–108, 1983.

[45] Kirk Riley, David S. Ebert, Charles D. Hansen, and Jason J. Levit. Visually accurate multi-field weather visualization. In *IEEE Visualization'03 Proceedings*, pages 279–286, 2003.

[46] Stefan Röttger, Stefan Guthe, Daniel Weiskopf, Thomas Ertl, and Wolfgang Straßer. Smart hardware-accelerated volume rendering. In *VisSym*. Eurographics Association, 2003.

[47] Henning Scharsach, Markus Hadwiger, André Neubauer, Stefan Wolfsberger, and Katja Bühler. Perspective isosurface and direct volume rendering for virtual endoscopy applications. In *EuroVis*, pages 315–322. Eurographics Association, 2006.

[48] John W. Strutt. On the light from the sky, its polarization and colour. *Philosophical Magazine*, 41:107–120,274–279, 1871.

[49] UCAR. The weather research & forecasting model. http://www.wrf-model.org.

[50] NetCDF data format. http://www.unidata.ucar.edu/software/netcdf/.

[51] Volumeshop. http://www.volumeshop.org.

[52] Peter Wendling, Renate Wendling, and Helmut K. Weickmann. Scattering of solar radiation by hexagonal ice crystals. *Applied Optics*, 18(15):2663, 1979.

[53] Lee Westover. Footprint evaluation for volume rendering. In *SIGGRAPH'90 Proceedings*, volume 24, pages 367–376, 1990.

[54] Lance Williams. Casting curved shadows on curved surfaces. *SIGGRAPH*, 12(3):270–274, 1978.

[55] Orion Wilson, Allen Van Gelder, and Jane Wilhelms. Direct volume rendeing via 3D textures. Technical report, 1994.

[56] Caixia Zhang and Roger Crawfis. Volumetric shadows using splatting. In *VIS '02 Proceedings*, pages 85–92, 2002.

[57] Caixia Zhang and Roger Crawfis. Shadows and soft shadows with participating media using splatting. *IEEE Transactions on Visualization and Computer Graphics*, 9(2):139–149, 2003.