Visualisierungsinstitut der Universität Stuttgart
Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Diplomarbeit Nr. 3299

Zeitabhängige Visualisierung von Lichtechos in partizipierenden Medien

Christoph Bergmann

Studiengang:	Softwaretechnik
Prüfer:	Prof. Dr. Daniel Weiskopf
Betreuer:	Dipl.-Inf. Marco Ament
begonnen am:	15. Februar 2012
beendet am:	16. August 2012
CR-Klassifikation:	I.3.7, J.2

Contents

1 Introduction

A light echo is a delayed reflection of light caused by interstellar gas and dust that was sent out from a star during a short eruption, like e.g. a nova. In most cases, the gas and dust clouds that became visible in the reflection were repelled from the star in earlier eruptions. From the view of an observer the light echo expands in a concentric manner around the star because the length of the indirect path of the scattered light also constantly expands with an increasing distance from the star. Thus the arrival time at the observer is also delayed increasingly.

This phenomenon gained quite some fame in 2002, when the Hubble Space Telescope observed a significant eruption of the star V838 Monocerotis which was followed by a strong light echo that constantly expanded during the following months and years.

Figure 1.1: Light echo of the star V838 Monocerotis as observed by the Hubble Space Telescope in September 2006. Image source: `http://hubblesite.org/explore_astronomy/hubbles_universe_unfiltered/12`

The work of this thesis comprises an implementation of a physically based renderer to simulate light echoes accompanied by the appendant theoretical background. The approach is demonstrated with a variety of data sets.

1.1 Overview of the Document

1. The **Introduction** gives a short overview of the thesis and light echoes in space, which serve as the primary use case for the time-dependent rendering framework developed in this context.
2. The **Fundamentals** present the theory and concepts employed to realize the framework with a focus on the necessary background in computer graphics.
3. Following up on the fundamentals, the Chapter **Implementation** discusses the implementation of the actual framework and also mentions implementation specific details if their impact is significant.
4. **Results** showcases a number of images rendered with the framework. Some simple examples are used to display the various effects in isolation, leading over to more complex images depicting the combination of all effects.
5. Finally **Discussion and Future Work** summarizes the thesis as a whole, also mentioning shortcomings of the methods and the framework that could be addressed in future work.

1.2 Scope of Thesis

The goal of this diploma thesis is to establish a method to illuminate volumetric data via time-dependent light transport respecting the finite speed of light. This also includes participating media with spatially varying refractive indices that lead to both bent light curves as well as a modified speed of light propagation. The desired application for the method is the realistic rendering of astronomical phenomena where light echoes have been observed.

The work on this thesis encompassed the following tasks:

- Establishing an optical model of time-dependent radiative transfer including multiple scattering.
- Interactive modelling of a function exposed to a graphical user interface to simulate the varying brightness of a star over time.
- Developing a Monte Carlo based renderer. The algorithm is based on Photon Mapping. Both photon tracing as well as the radiance estimate have to respect the time dependency of light propagation. This also includes spatially varying refractive indices within the volume which lead to bent light curves and a modified speed of light propagation.

- The frontend and photon tracing is implemented in C++. The renderer is accelerated using a CUDA implementation of the radiance estimate (ray marching and photon gathering). This includes optimization of the Photon Mapping approach to efficiently make use of parallelization, especially with regard to the photon map data structure.
- Devising a variety of volumes to showcase the photon mapping implementation and the extensions to it.

1.3 Light Echoes

A light echo is a phenomenon that can be understood in a way similar to an acoustic echo. Imagine an explosion on a scale that makes the finite speed of light relevant to the observation. One such example would be a supernova. A supernova is a stellar explosion of a star that increases its luminosity up to a factor in the millions through a massive burst of radiation [NAS]. If there is to be an observer of this event in a distance of a few light years, he or she will only be able to actually observe the event at a later point in time with respect to the finite speed of radiative transfer and the distance between the event and the observer.

Figure 1.2: Example of a Supernova: SN 1994D was discovered in 1994 at the Leuschner Observatory. It is located more than 100 million light years away from Earth and can be seen as the bright dot on the lower left. Image source: `http://en.wikipedia.org/wiki/Supernova`

If the star is surrounded by e.g. a nebula, the light of the explosion will also reach it after some time and the light will be scattered by the particles within the nebula in different directions. A fraction of the light will also be scattered towards the observer, who will then also be able to see the scattered light from the supernova. Because of the indirection introduced with the scattering, the total distance of the light path between the supernova and the observer will be significantly increased compared to the light that has been propagated directly from the

supernova to the observer (see Figure 1.3). As a result to this, the viewer will only see the illuminated nebula at a later point in time compared to the observation of the supernova itself. Because of the sheer scale of these events, the perceived delay between the primal observation and the light echo can reach a magnitude of months or even years [hub].

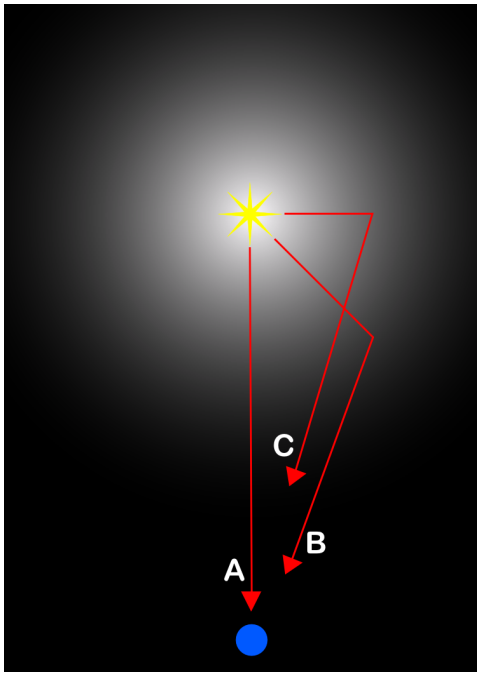


Figure 1.3: Diagram of a light echo. The light was emitted isochronously from the light source but will be observed with time gaps relative to each other. The viewer will first see A, then B and finally C. Image source: `http://en.wikipedia.org/wiki/Light_echo`

Accordingly a light echo is a glimpse into the past. It shows the reflection of an event whose primal room for observation has already passed and thus presents an additional way to gain further insight about it.

The observation of light echoes allows astronomers to analyze the spectrum of supernovae whose first light had reached Earth before the telescope was invented. For example, in 1572, the Danish astronomer Tycho Brahe observed the explosion of supernova SN 1572. In 2008, a scattered light echo of the exact same supernova was obtained more than four centuries after the direct light swept past the Earth [KTU+re].

Another example for such a process was the deduction of the supernova type that created Cassiopeia A, which is the name of the remnant of the supernova that created it.

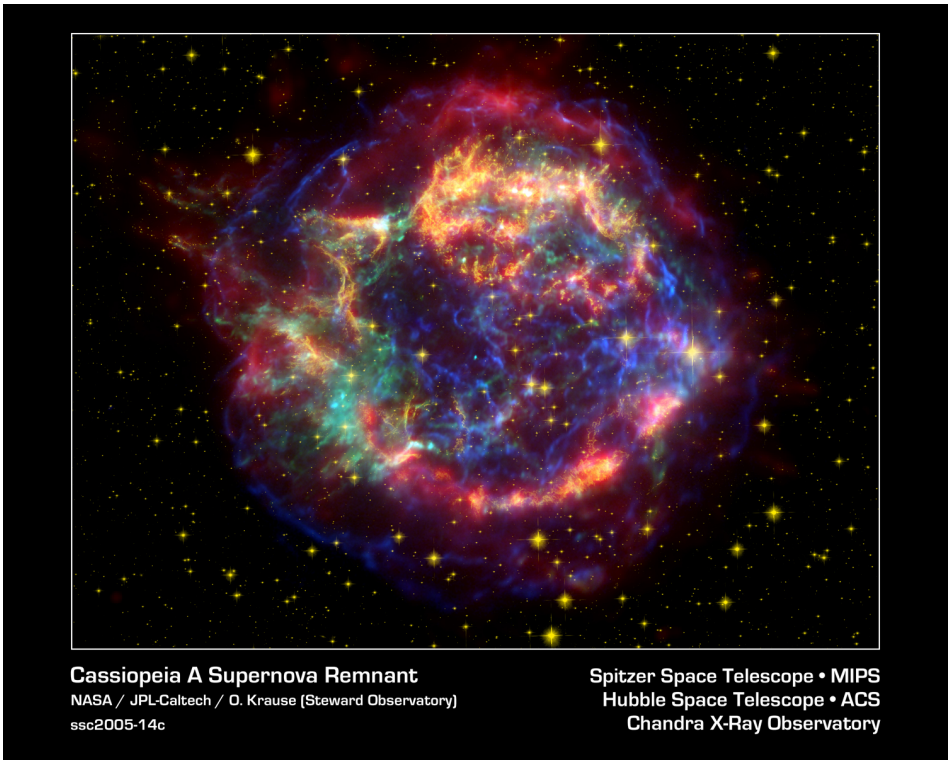


Figure 1.4: False-color image of the supernova remnant Cassiopeia A that was combined from infrared, visible and x-ray data. Please note that this is a visualization of the remnant itself and no depiction of the light echo. Image Source: `http://subarutelescope.org/Pressrelease/2008/05/29/index.html`

Cassiopeia A is located around 9,000 light years from Earth and the supernova could have been observed from Earth around A.D. 1680 [SM]. Over 300 years later the Spitzer Space Telescope captured an image in 2003 while it was testing its optics, that would lead to the insight about the nature of the original supernova: The image yielded infrared signatures that could be associated with a light echo from Cassiopeia A. Further investigation of the spectra of the reflected light allowed to determine the nature of the original supernova [SM].

Another application of light echoes is the spatial reconstruction of the environment of a supernova [Pat05]. So in general light echoes can be used as a large scale, time delayed probing tool to gain more insight about both matter illuminated by it and the origin of the light echo itself. As this thesis is in the field of computer graphics, we basically invert the task: We assume to know about the origin of the light echo and the matter that surrounds it and we would like to develop a tool that renders the light echo that would have been produced by these known prerequisites.

V838 Monocerotis

All information in this section is comprised from [hub].

This section shortly presents another intriguing example of a light echo that highlights the distinction between the expanding light echo and the comparably static nature of the illuminated matter.

Until 2002, V838 Monocerotis was just an unidentified star in a distance of around 20,000 light years from the sun until it gained widespread fame for the observation of one of the most impressive light echoes captured so far.

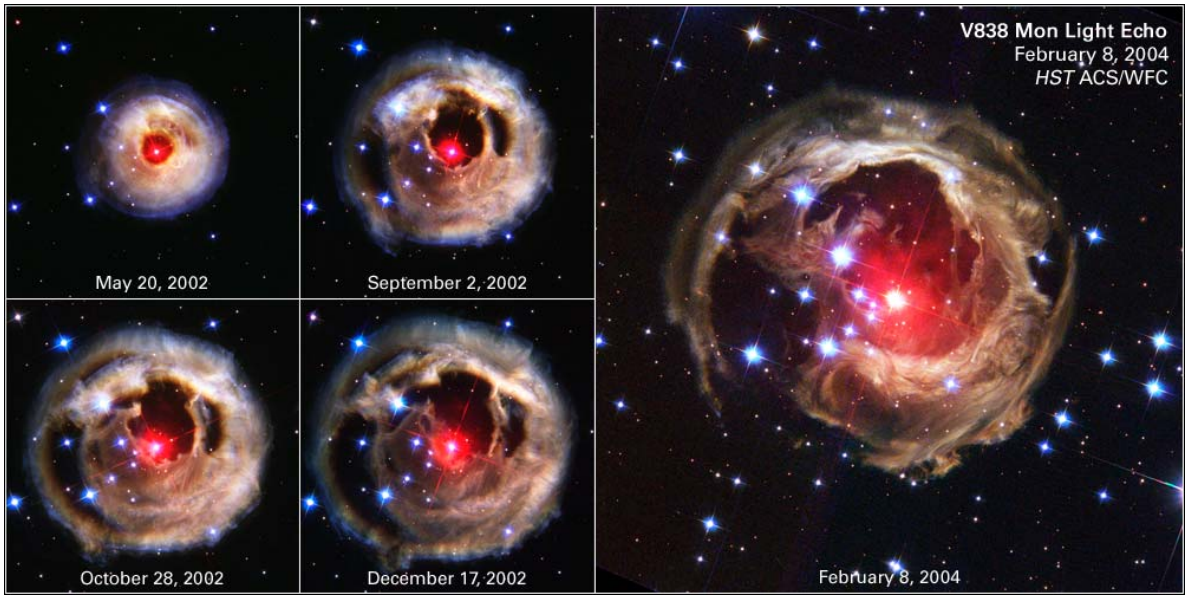


Figure 1.5: V838 Monocerotis. As observed by the Hubble Space Telescope between May 2002 and February 2004. Image source: `http://en.wikipedia.org/wiki/V838_Monocerotis`

As of today the actual event that caused the eruption producing the light echo is not yet determined with certainty. The outburst lasted for several months during which the star flared to 10,000 times its usual brightness but unlike a supernova it did not explosively eject its outer layers [BHL+03].

One very important point in understanding the light echo is the nebula surrounding the star. On first glance the image sequences appear to show a massive outburst of material caused by the eruption. In fact the nebula itself is more or less static and the current assumption is that it originated from one or more outbursts that happened far earlier. So what the images depict are solely the different stages of the light outburst illuminating different parts of the nebula surrounding the star. When comparing the stages of e.g. May and September 2002 in Figure (1.5), the particles of the outer part of the nebula visible in September were also

there in May but just not yet illuminated. The same applies to the September image with respect to the apparent gap of material directly surrounding the red star in the center: As the brightness outburst lasted only for a few months, the inner layers of the nebula are not illuminated anymore at this point whereas the outer layers just came into view.

2 Fundamentals

The theoretical background of this thesis provides information on aspects necessary to understand the light echo rendering technique, as well as the description of the developed framework in the subsequent chapter.

To ensure a consistent notation, the thesis will comply to the notation set forth in the following table of quantities

Symbol	Description
$\Re$	A random number in the range $[0, 1]$.
$\mathbf{x}$	Indicates a position in 3D-Space.
$\vec{\omega}$	Normalized direction.
Ω	Hemisphere of directions.
$L(\mathbf{x}, \vec{\omega}, t)$	Radiance at position $\mathbf{x}$ and direction $\vec{\omega}$ at a time t.
$\Phi(t)$	Flux. In this context time-based and associated with a light source.
c	Speed of Light in vacuum. 299,792,458 m/s.
$n(\mathbf{x})$	Refractive index at position $\mathbf{x}$.
$\nabla n(\mathbf{x})$	Gradient of the scalar field of n at position $\mathbf{x}$.
$\alpha(\mathbf{x})$	Absorption coefficient at position $\mathbf{x}$.
$\sigma(\mathbf{x})$	Scattering coefficient at position $\mathbf{x}$.
$\kappa(\mathbf{x})$	Extinction coefficient at position $\mathbf{x}$. $\kappa(\mathbf{x}) = \alpha(\mathbf{x}) + \sigma(\mathbf{x})$.
$\Lambda(\mathbf{x})$	Albedo. Fraction of energy reflected. $\Lambda(\mathbf{x}) = \sigma(\mathbf{x})/\kappa(\mathbf{x})$.
$\tau(\mathbf{x}', \mathbf{x})$	Transmittance along a line segment from $\mathbf{x}'$ to $\mathbf{x}$.
$f(\mathbf{x}, \vec{\omega}', \vec{\omega})$	Normalized phase function.
g	Henyey-Greenstein phase function parameter.

2.1 Related Work

The original equation of radiative transfer, that is also the base of the volume rendering equation has been described by Chandrasekhar in [Cha60].

Global illumination in participating media using Photon Maps, which provides a solution/approximation to the volume rendering equation has originally been introduced in 1998 by Henrik Wann Jensen [JC98]. The method has been improved in recent years using Photon Beams [JZJ08] and Query Beams [JNJ11].

The extension of Photon Mapping to RGB color channels has been described in [Jen01], although not in detail and partly ambiguous. An implementation of Volumetric Photon Mapping using RGB color channels and even a spectral solution can be found in the open-source renderer Mitsuba [mit].

Nonlinear Photon Mapping that takes into account bent light curves caused by spatially varying refractive indices has been described in [Gut05].

A method to efficiently port the Photon Mapping algorithm to CUDA has been described in [Fle09]. The method is based on the description of a particle simulation by NVIDIA [Gre08]. Although the method is described in the context of Photon Mapping for surfaces, it can be applied to Volumetric Photon Mapping in a straightforward manner.

The novelty of this thesis, the extension of Photon Mapping to include the time dependency of light propagation is inspired by [Pat05]. This publication describes a Monte Carlo Simulation of light echoes. It is not directly suitable to be interpreted as a rendering solution, but provided valuable ideas for our rendering solution.

2.2 Light Transport in Participating Media

In order to render a light echo, the most basic requirement is the ability to render the galactic nebula that is illuminated by a star at its center. In terms of computer graphics, this requirement can be restated as the necessity to simulate light transport in participating media where the star is the emitter of light (simplified as a point light source) and the nebula is the participating medium (simplified as a 3D volume holding a density value per data point).

In our special case, also taking into account the time dependency of light transport is of the essence. As this introduces an additional layer of complexity, we first introduce the background on light transport as it has been commonly used in computer graphics and describe time dependency as an extension. The first formal description of light transport has been introduced by Chandrasekhar [Cha60], stated as the equation of radiative transfer. Here the equation is described using the notation as typically used in computer graphics (see [Jen01]).

Light transport in a participating medium is affected by emission, in-scattering, absorption and out-scattering.

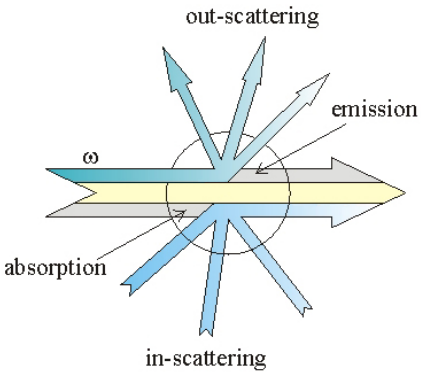


Figure 2.1: Illustration of emission, absorption, in- and out-scattering at a specific position and direction in a participating medium. Image Source: `http://www.cescg.org/CESCG-2004/web/Zdrojewska-Dorota/`

A medium is called homogeneous if the scattering and absorption is constant throughout the entire volume, otherwise it is called non-homogeneous. The change in radiance L at a position $\mathbf{x}$ in direction $\vec{\omega}$ can be written as

$$\begin{aligned}(\vec{\omega}\cdot\nabla)L(\mathbf{x},\vec{\omega}) &= \alpha(\mathbf{x})L_e(\mathbf{x},\vec{\omega}) + \sigma(\mathbf{x})L_i(\mathbf{x},\vec{\omega}) - \alpha(\mathbf{x})L(\mathbf{x},\vec{\omega}) - \sigma(\mathbf{x})L(\mathbf{x},\vec{\omega}) \\ &= \alpha(\mathbf{x})L_e(\mathbf{x},\vec{\omega}) + \sigma(\mathbf{x})L_i(\mathbf{x},\vec{\omega}) - \kappa(\mathbf{x})L(\mathbf{x},\vec{\omega}) \qquad (2.1)\end{aligned}$$

L_e is the emitted radiance, L_i is the in-scattered radiance, σ is the scattering coefficient, α the absorption coefficient and κ is the extinction coefficient. In general the radiance is also dependent on the wavelength, so this equation can be seen as a reduction to a single wavelength.

The in-scattered radiance L_i depends on the radiance L from all directions $\vec{\omega}$ over the sphere Ω

$$L_i(\mathbf{x},\vec{\omega}) = \int_\Omega f(\mathbf{x},\vec{\omega}',\vec{\omega})L(\mathbf{x},\vec{\omega}')d\vec{\omega}' \qquad (2.2)$$

where $f(\mathbf{x},\vec{\omega}',\vec{\omega})$ is the normalized phase function that determines how much light at position $\mathbf{x}$ is scattered from the incident direction $\vec{\omega}'$ into direction $\vec{\omega}$.

Combining Equation (2.1) and (2.2) yields the following integro-differential equation

$$(\vec{\omega}\cdot\nabla)L(\mathbf{x},\vec{\omega}) = \alpha(\mathbf{x})L_e(\mathbf{x},\vec{\omega}) + \sigma(\mathbf{x})\int_\Omega f(\mathbf{x},\vec{\omega}',\vec{\omega})L(\mathbf{x},\vec{\omega})d\omega' - \kappa(\mathbf{x})L(\mathbf{x},\vec{\omega}) \qquad (2.3)$$

Integrating both sides of the equation along a straight path from x_0 to x in direction $\vec{\omega}$ gives

$$
\begin{aligned}
L(\mathbf{x},\vec{\omega}) &= \int_{x_0}^{x} \tau(x^{'},x)\alpha(x^{'})L_e(x^{'},\vec{\omega})dx^{'} \\
&\quad + \int_{x_0}^{x} \tau(x^{'},x)\sigma(x^{'}) \int_{\Omega} f(x^{'},\vec{\omega}^{'},\vec{\omega})L(x^{'},\vec{\omega}^{'})d\omega^{'}dx^{'} \\
&\quad + \tau(x_0,x)L(x_0,\vec{\omega})
\end{aligned} \tag{2.4}
$$

where $\tau(x^{'},x)$ is the transmittance along the line segment from $x^{'}$ to x

$$\tau(x^{'},x) = e^{-\int_{x^{'}}^{x} \kappa(y)dy}$$

The integral $\int_{x^{'}}^{x} \kappa(y)dy$ is also called the optical depth of the medium between $x^{'}$ and x.

Equation (2.4) is the **Volume Rendering Equation**. Solving this equation is the goal of volume rendering techniques that strive to depict a physically based simulation.

Note

For practical applications, rendering an image using the equation usually brakes down to finding a numerical solution/approximation to the equation. A typical ray tracing/ray marching approach identifies a viewing ray for each pixel with the straight path that has been used to derive the equation. We identify x with the position of the camera and need to find the direction ω for each viewing ray and then solve or rather approximate $L(\mathbf{x},\omega)$ for each pixel using an algorithm and finally use a tone mapper to reduce the high-range radiance result to a viewable image.

2.2.1 Transient Volume Rendering Equation

As we would like to capture the time dependency of light propagation, the volume rendering equation is not sufficient to describe this. In this section we show the extended version of the volume rendering equation that takes into account the time dependency. The equations are adapted from [Bun09].

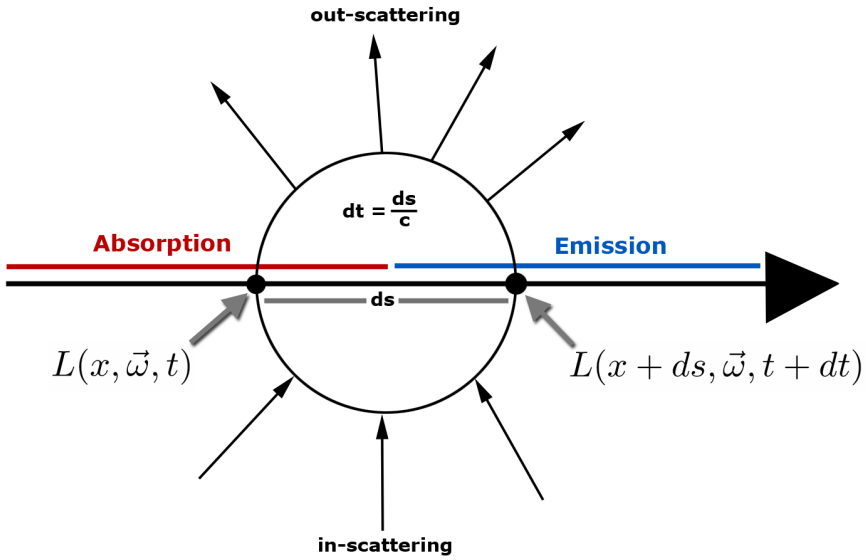


Figure 2.2: Illustration of factors influencing the change of radiance, analog to Figure 2.1. In this case we consider the change of radiance regarding both a differential change in position ds, as well as a differential advance of time dt.

dL is a total derivative, describing the change of a function (in this case L) depending on all parameters (in this case both time and direction).

$$\begin{aligned} dL &= \frac{\partial L}{\partial t}dt + \frac{\partial L}{\partial s}ds \\ &= \frac{\partial L}{\partial t}\frac{ds}{c} + \frac{\partial L}{\partial s}ds \\ &= (\frac{\partial L}{\partial t}\frac{1}{c} + \frac{\partial L}{\partial s})ds \\ \frac{dL}{ds} &= \frac{\partial L}{\partial t}\frac{1}{c} + \frac{\partial L}{\partial s} \end{aligned}$$

The term $\frac{\partial L}{\partial s}$ is a directional derivative that may also be rewritten as a dot product of direction and gradient.

$$\begin{aligned} \frac{dL}{ds} &= \frac{\partial L}{\partial t}\frac{1}{c} + \vec{\omega} \cdot \nabla L \\ \frac{dL}{ds} &= \alpha(\mathbf{x})L_e(\mathbf{x}, \vec{\omega}, t) + \sigma(\mathbf{x}) \int_\Omega f(\mathbf{x}, \vec{\omega}', \vec{\omega})L(\mathbf{x}, \vec{\omega}', t)d\omega' - \kappa(\mathbf{x})L(\mathbf{x}, \vec{\omega}, t) \end{aligned} \tag{2.5}$$

We adapt the equation from integrating along arc length to time:

The differential ds equals the differential time dt multiplied with the modified speed of light with respect to the refractive index: $ds = \frac{c}{n}dt$.

$$\frac{dL}{ds} = \frac{dL}{\frac{c}{n}dt} = \frac{dL}{dt}\frac{n}{c} = \alpha(\mathbf{x})L_e(\mathbf{x},\vec{\omega},t) + \sigma(\mathbf{x})\int_\Omega f(\mathbf{x},\vec{\omega}',\vec{\omega})L(\mathbf{x},\vec{\omega}',t)d\omega' - \kappa(\mathbf{x})L(\mathbf{x},\vec{\omega},t)$$

We define $v(t) = \frac{c}{n}$. Multiplying both sides with $v(t)$ yields

$$\frac{dL}{dt} = v(t)\cdot(\alpha(\mathbf{x})L_e(\mathbf{x},\vec{\omega},t) + \sigma(\mathbf{x})\int_\Omega f(\mathbf{x},\vec{\omega}',\vec{\omega})L(\mathbf{x},\vec{\omega}',t)d\omega' - \kappa(\mathbf{x})L(\mathbf{x},\vec{\omega},t))$$

Now we can integrate along time, using a mapping $\mathbf{x}(t)$ from time to position (written as $\mathbf{x}'(t)$ in the equation to distinguish from the fixed position $\mathbf{x}$). We assume to have valid boundary conditions.

$$\begin{aligned} L(\mathbf{x},\vec{\omega},t) \;=\; & \int_{t_0}^{t}\tau(\mathbf{x}'(t'),\mathbf{x})\alpha(\mathbf{x}'(t'))v(t')L_e(\mathbf{x}'(t'),\vec{\omega},t)dt' \\ & + \int_{t_0}^{t}\tau(\mathbf{x}'(t'),\mathbf{x})\sigma(\mathbf{x}'(t'))v(t')\int_\Omega f(\mathbf{x}'(t'),\vec{\omega}',\vec{\omega})L(\mathbf{x}'(t'),\vec{\omega}',t')d\omega' dt' \\ & + \tau(\mathbf{x}'(t_0),\mathbf{x}'(t))L(\mathbf{x}'(t_0),\vec{\omega},t) \end{aligned} \tag{2.6}$$

The mapping from time to position can be written as follows, assuming a straight line segment:

$$\mathbf{x}(t) = \mathbf{x}(t_0) + \int_{t_0}^{t}\omega\frac{c}{n}dt$$

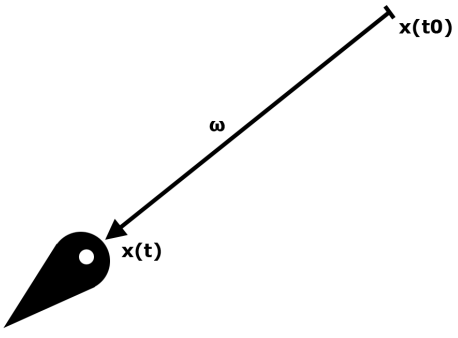


Figure 2.3: Illustration of mapping from time to position.

There is one important addition that we have to forestall at this point: In general $\mathbf{x}(t)$ will be a curve and not a line in our framework, as we will introduce an extension to incorporate refractive media. For more information on this, see Section 2.4.4.

Approximating the transient rendering equation is one of the main goals of this thesis as it will enable to capture the time dependency of light propagation, making it possible to visualize light echoes.

2.3 Scattering and Phase Functions

The way we visually perceive objects in our environment is strongly influenced by the effects of absorption and scattering. As we are working with geometrical optics, we may work under the assumption of discrete light particles (photons) that move along a deterministic trajectory. In this context scattering then denotes the deviation of a photon from its current trajectory caused by an interaction with the environment. This can be caused by e.g. surfaces, particles or density fluctuations.

Basically all things we see are influenced by the effects of scattering. If we look into a mirror, we perceive an almost ideal specular reflection where the light is scattered according to the law of reflection. On the other hand if we imagine a very matte, white, concrete surface we see the effects of diffuse reflections where light from an incident direction is scattered almost uniformly in all directions leaving the surface. From sunlight that is scattered by raindrops to produce a rainbow, yielding the effect of dispersion, to the blue sky showcasing the varying amount of scattering with different wavelengths, all these phenomena are governed by scattering.

Figure 2.4: The way we see things is strongly influenced by the way how light interacts with the environment, especially through means of scattering.

In our case we are especially interested in scattering of light within volumes described by the spatial density of particles within the volume. In general light scattering can be categorized into three domains [BS09]:

- Rayleigh scattering - The particles are small compared to the wavelength of the light.
- Mie scattering - The particles are about the same size as the wavelength of light.
- Geometric scattering - The particles are much larger than the wavelength of light.

For approximating light interaction with a galactic nebula, the category we are most interested in is Mie scattering [Pat05]. Solutions to Mie scattering exist for various types of simplified particle geometries, most notably spheres. Unfortunately these solutions are too expensive and involved for most practical applications ([Goo95], p. 323). As a result to this, simplified phase functions have been developed that can be used in practical applications.

Phase functions

A phase function $f(\mathbf{x}, \vec{\omega}', \vec{\omega})$ describes the angular distribution of light scattered from direction $\vec{\omega}'$ to direction $\vec{\omega}$ at a position $\mathbf{x}$. A phase function is called normalized if the condition

$$\int_{\Omega} f(\mathbf{x}, \vec{\omega}', \vec{\omega}) d\vec{\omega} = 1$$

holds. In this case the phase function defines a probability distribution for scattering from a specific direction into another specified direction ([Jen01], p. 582). This attribute will come in handy for the stochastic approach used in Photon Mapping.

Isotropic phase function

$$f(\mathbf{x}, \vec{\omega}', \vec{\omega}) = \frac{1}{4\pi}$$

The phase function is normalized and states that light is scattered uniformly in any direction. Thus the scattering does not depend on the incoming or outgoing direction. This simple function however is not sufficient to describe the anisotropic scattering by intergalactic dust.

Henyey-Greenstein phase function

$$f(\mathbf{x}, \vec{\omega}', \vec{\omega}) = \frac{1 - g^2}{4\pi(1 + g^2 - 2gcos\Theta)^{1.5}} \tag{2.7}$$

The Henyey-Greenstein (H-G) phase function is the most commonly used phase function to describe scattering by e.g. dust, clouds, oceans or skin [HG41]. It can be influenced by the parameter $g \in [-1, 1]$, is also normalized for the valid range of g and otherwise only depends on the angle Θ between $\vec{\omega}'$ and $\vec{\omega}$. A positive and increasing g value gives an increasing amount of forward scattering, $g = 0$ is isotropic scattering and finally a decreasing negative value of g implies increasing backward scattering. The value assumed to yield the best approximation for galactic dust is $g \approx 0.6$ ([Pat05], p. 1163). The framework uses this function for all calculations involving a phase function.

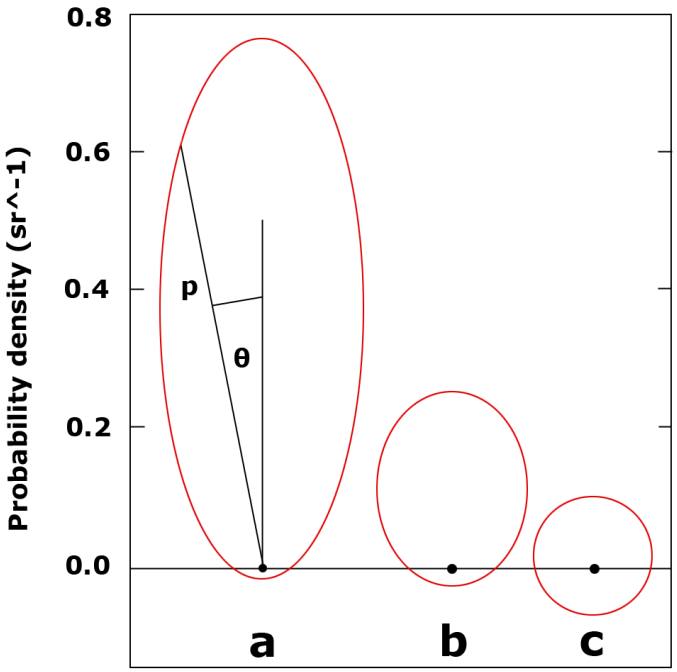


Figure 2.5: Visualization of the Henyey-Greenstein phase function using polar coordinates. For each sample the black dot marks the origin. The phase function angle parameter Θ is used as the polar angle, whereas the phase function result is mapped to the polar radius. This results in the red ellipses. Sample (a) uses g=0.6, sample (b) g=0.35 and (c) uses g=0.04. Image Source: Reconstructed from `http://www.sciencedirect.com/science/article/pii/S0165232X11002667`

Employing the H-G phase function for our Photon Mapping algorithm also requires that we need to determine a new direction for each photon after a scattering event with respect to the phase function. Sampling the phase function can be achieved by inverting it as follows ([Pat05], p. 1167):

$$cos\Theta = \frac{1}{2g}[(1 + g^2 - \frac{(1 - g^2)^2}{(1 + 2g\Re - g)^2}]$$

$cos\Theta$ is the cosine of the scattering angle between an incident direction and the scattered direction, g is the H-G parameter and $\Re$ is a normalized random number. As the scattering angle alone does not determine a unique new direction, the scattering azimuth ϕ is also determined by $\phi = \pi(2\Re - 1)$, a random number in the range of $-\pi \leq \phi \leq \pi$.

With these prerequisites and given a direction $\vec{\omega}_0 = (x_0, y_0, z_0)$, a new sampled direction $\vec{\omega}_1 = (x_1, y_1, z_1)$ can be obtained:

$$
\begin{aligned}
x_1 &= x_0 \cdot cos\Theta - \frac{y_0}{\sqrt{1-z_0^2}} sin\Theta sin\phi + \frac{x_0 z_0}{\sqrt{1-z_0^2}} sin\Theta cos\phi \\
y_1 &= y_0 \cdot cos\Theta - \frac{x_0}{\sqrt{1-z_0^2}} sin\Theta sin\phi + \frac{y_0 z_0}{\sqrt{1-z_0^2}} sin\Theta cos\phi \\
z_1 &= z_0 \cdot cos\Theta - \sqrt{1-z_0^2} sin\Theta sin\phi
\end{aligned} \tag{2.8}
$$

2.4 Volumetric Photon Mapping

Information in this section is compiled from [Jen01] if not noted otherwise.

Volumetric Photon Mapping is a global illumination technique that can be used in scenes with participating media. It has been introduced in 1998 by Henrik Wann Jensen and can be seen as an extension to the Photon Mapping technique for surfaces [Jen96]. Since for our use case the scene only consists of volumes without any surface representations, this section directly introduces Photon Mapping for volumes from scratch, but skips aspects relevant for rendering surfaces. Please note that, while ultimately our goal is to describe a transient Photon Mapping algorithm that captures time-dependent effects, we start with the classical non-transient approach as a basis and introduce transient Photon Mapping as an extension to the original approach later on.

In general photon mapping is a two-pass technique:

- **Photon Tracing Stage** - Photons are emitted from a light source and stored within a dedicated photon map data structure.
- **Photon Gathering Stage** - During the rendering pass based on ray tracing/ray marching, photons are gathered to get a local radiance estimate.

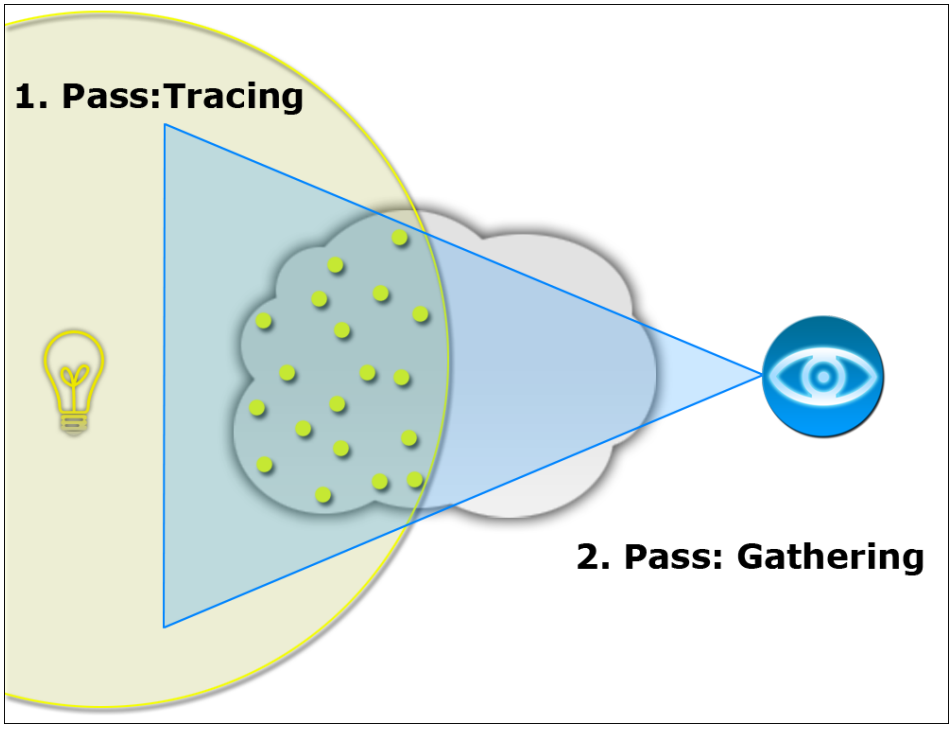


Figure 2.6: Illustration of the two pass approach used in Photon Mapping. Photons are emitted from the light source in the first stage, and will be collected during gathering in the second stage.

2.4.1 Photon Tracing

During the first pass the photon map is constructed by emitting photons from a light source and each time a photon interacts with the volume, it is deposited in the volume photon map.

Caution

Please note that photons in Photon Mapping are not supposed to directly represent their physical counterparts, as the energy transported by one photon depends on both the overall count of photons used for the simulation as well as the flux of the light source. So in Photon Mapping the overall number of photons emitted from a light source is a qualitative term strongly influencing the accuracy of the radiance estimate whereas in reality the number of photons emitted during a time would be a measurement of light power.

The probability density function (PDF) describing the probability of a photon interacting with the participating medium at position $\mathbf{x}$ is

$$F(\mathbf{x}) = 1 - \tau(\mathbf{x_s}, \mathbf{x}) = 1 - e^{-\int_{\mathbf{x_s}}^{\mathbf{x}} \kappa(y)dy}$$

where $\mathbf{x_s}$ is the entry point of the photon in the volume. In order to practically determine the position of interaction for a photon we identify $F(\mathbf{x})$ with a normalized random number $\Re$:

$$\begin{aligned} \Re &= F(\mathbf{x}) \\ \Re &= 1 - \tau(\mathbf{x_s}, \mathbf{x}) \\ \Re &= \tau(\mathbf{x_s}, \mathbf{x}) \\ \Re &= e^{-\int_{\mathbf{x_s}}^{\mathbf{x}} \kappa(y)dy} \\ ln(\Re) &= -\int_{\mathbf{x_s}}^{\mathbf{x}} \kappa(y)dy \\ -ln(\Re) &= \int_{\mathbf{x_s}}^{\mathbf{x}} \kappa(y)dy \end{aligned} \tag{2.9}$$

This means that we can derive a new interaction point, given a starting point $\mathbf{x_s}$ and a direction $\vec{\omega}_s$, by using a normalized random number $\Re$, guessing the optical depth $-ln(\Re)$ at which the interaction will occur and stepping through the volume using ray marching, accumulating optical depth until that value is reached and thus setting the position of interaction.

After the position of interaction is determined, there is more than one option for the next step. One possibility is to use russian roulette using the albedo Λ to determine if the interaction event is either scattering or absorption of the photon. More precise: We take another random number $\Re$. If $\Re \leq \Lambda$ the photon is scattered, otherwise it is absorbed. In this case the power of the photon does not have to be scaled.

We do not use this approach as it cannot easily be combined with the RGB extension. For more information on this, see section 2.4.3. Instead of using russian roulette in conjunction with absorption and scattering, we scale the photon's power with the albedo for each interaction event and only use russian roulette to terminate the path.

There is one last ingredient required for stating a complete tracing algorithm. In our case we assume that the star we would like to simulate is approximated using a point light (a light that shines isotropically, assuming a quadratic distance falloff of the light power Φ). So the starting position for each photon in the simulation can always be set to the position of the light source, but we also need to generate a random set of initial directions for the simulation. In order to generate a normalized random direction $\vec{\omega}_s = (x_s, y_s, z_s)$ using two normalized random numbers $\Re_0$ and $\Re_1$ we can use [Pat05], p. 1167:

$$\begin{aligned} z_s &= 2\Re_0 - 1 \\ x_s &= \sqrt{1 - z_s^2} cos[\pi(2\Re_1 - 1)] \\ y_s &= \sqrt{1 - z_s^2} sin[\pi(2\Re_1 - 1)] \end{aligned} \tag{2.10}$$

Finally we can derive a complete algorithm for tracing and storing a photon in the photon map. The algorithm is listed as it is implemented in the framework, so we use the option to scale the photon's power with the albedo instead of an absorption/scattering decision using russian roulette.

1. Generate a photon at the position of the light source with a random initial direction according to Equation (2.10).
2. Check if there is an intersection with the volume and the initial ray. If so, determine the entry point, otherwise stop tracing the photon.
3. Determine the interaction point within the volume using Equation(2.9). If the interaction point is outside the volume, quit tracing.
4. Multiply the photon's power with the albedo and store it in the photon map.
5. Use russian roulette using the albedo to terminate the path after a number of minimum scattering events.
6. If the decision is to continue with the path, generate a new tracing direction using Equation (2.8) and continue with step (3).

A desired number of photons stored in the photon map is set. The tracing algorithm is placed in a loop and the number of currently stored photons is tracked. New traces are launched until the desired number of photons is reached.

Note

Usually Volumetric Photon Mapping splits the calculation of single scattering and multiple scattering in two separate parts. Single scattering is computed using ray marching whereas multiple scattering is computed via the photon map. In this case the photon tracing algorithm just skips the first interaction by not storing it in the photon map. In our case however, we cannot separate the calculations (see Section 2.4.4) and thus also store the first hit in the photon map.

2.4.2 Photon Gathering

In order to derive a radiance estimate for volumes using photons that represent flux, the relationship between scattered flux Φ and radiance L in a participating medium is used:

$$L(\mathbf{x}, \vec{\omega}) = \frac{d^2\Phi(\mathbf{x}, \vec{\omega})}{\sigma(\mathbf{x}) d\omega dV} \tag{2.11}$$

Combining Equation (2.11) with Equation (2.2) yields

$$
\begin{aligned}
L_{ii}(\mathbf{x},\vec{\omega}) &= \int_{\Omega} f(\mathbf{x},\vec{\omega}',\vec{\omega})L(\mathbf{x},\vec{\omega}')d\vec{\omega}' \\
&= \int_{\Omega} f(\mathbf{x},\vec{\omega}',\vec{\omega})\frac{d^2\Phi(\mathbf{x},\vec{\omega}')}{\sigma(\mathbf{x})d\vec{\omega}'dV}d\vec{\omega}' \\
&= \frac{1}{\sigma(\mathbf{x})}\int_{\Omega} f(\mathbf{x},\vec{\omega}',\vec{\omega})\frac{d^2\Phi(\mathbf{x},\vec{\omega}')}{dV} \\
&\approx \frac{1}{\sigma(\mathbf{x})}\sum_{p=1}^{n} f(\mathbf{x},\vec{\omega}_p',\vec{\omega})\frac{\Delta\Phi_p(\mathbf{x},\vec{\omega}_p')}{\frac{4}{3}\pi r^3}
\end{aligned}
\tag{2.12}
$$

The last step means that the integral is approximated by discretization, replacing the differential volume with the smallest sphere of radius r containing the n nearest photons to the position $\mathbf{x}$. The flux carried by each photon $\Delta\Phi_p$ is determined by dividing the flux of the light source with the number of photons that were *emitted* (as opposed to stored) in the photon tracing stage.

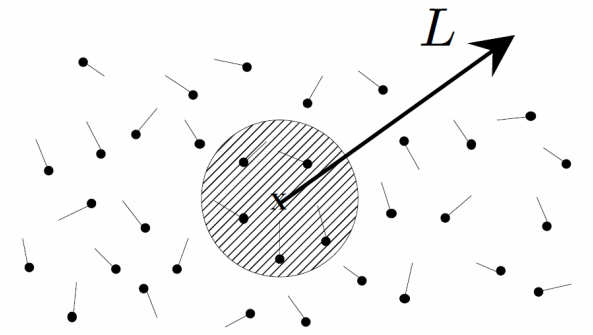


Figure 2.7: Visualization of the photon density estimate. To estimate in-scattered radiance, photons within a sphere of radius r around the current position are taken into account. Source: [JC98]

In general the computation is split into single scattering and multiple scattering, and thus the equation for in-scattered radiance is split accordingly. Please note that in this framework the single scattering term is included in the photon map. Regarding the equation, the only change would be to remove the direct term as all first interactions are also stored in the photon map. This is valid but requires a significantly increased number of photons in the simulation but is unavoidable in this context (see 2.4.4).

$$
L_i(\mathbf{x},\vec{\omega}) = L_{i,d}(\mathbf{x},\vec{\omega}) + \frac{\sigma(\mathbf{x})}{\kappa(\mathbf{x})}L_{i,i}(\mathbf{x},\vec{\omega})
$$

The direct term $L_{i,d}$ is computed using ray marching. More specifically in the case of a point light, the direct in-scattered radiance is computed by attenuating the flux of the light source

with the squared distance from the current position to the light source, and further attenuating it with the transmittance between the current position and the light source. The indirect term $L_{i,i}$ has to be multiplied with the scattering albedo in case absorbed photons are stored in the photon map. In our case this factor drops out of the equation as we already scale the photon's power using the albedo during photon tracing.

To render an image using photon mapping, ray marching is employed to iteratively compute radiance along a ray that passes through the medium. For each step in the volume the radiance from the previous step is attenuated and the emitted and in-scattered radiance at the current position is added:

$$\begin{aligned} L(\mathbf{x_k}, \vec{\omega}) &= \alpha(\mathbf{x_k}) L_e(\mathbf{x_k}, \vec{\omega}) \Delta\mathbf{x_k} \\ &+ \sigma(\mathbf{x_k}) L_i(\mathbf{x_k}, \vec{\omega}) \Delta\mathbf{x_k} \\ &+ e^{-\kappa(\mathbf{x_k})\Delta\mathbf{x_k}} L(\mathbf{x_{k-1}}, \vec{\omega}) \end{aligned} \tag{2.13}$$

with $\Delta\mathbf{x_k} = |\mathbf{x_k} - \mathbf{x_{k-1}}|$ being the step size and $\mathbf{x_0}$ the entry point of the ray in the volume. The step size can be fixed, but various optimizations are possible, e.g. recursively halving the step size if the radiance differs too much, or jittering the starting position for ray marching to reduce banding artifacts.

2.4.3 RGB Extension

In order to introduce more than a single color band (in our case we use RGB channels) to achieve effects like color bleeding, a few adjustments for a practical implementation have to be made. First and foremost data types have to be extended for all values involving or calculating the Radiance L from a single valued type to a three component type. Alongside, the absorption α and scattering σ values are extended to three valued types, representing the color of the volume (or the material of the volume so to speak). Following this, also the extinction coefficient and albedo naturally extend to three valued types.

In general the RGB radiance is calculated like the single valued radiance, just extending the calculation to more than one but separately treated channels. There are some points however that cannot be treated in isolation for each channel:

- **Photon Interaction Position** - During photon tracing the optical depth is accumulated to determine the next point of interaction. As this includes the extinction coefficient that is now three valued, the optical depth is also three valued. Here we track all channels and set the interaction position to the position where the transmittance is reached first on any of the channels.
- **Photon Material Interaction** - Photons have to store a power/weight value in order to be able to capture interactions with colored materials. In this case [Jen01], p. 123 states the options to either scale the scattered photon with the albedo or use russian roulette to decide between scattering or absorption. For colored photons the latter poses

a problem as the individual components of the photon still have to be scaled somehow to capture the material interaction, but neither [Jen01], nor any other literature reviewed for this thesis provided a plausible answer or method for this. We thus choose to discard the russian roulette for an absorption/scattering decision and instead use the first option to multiply with the multi-channeled albedo. This is also the method used in the Mitsuba Renderer [mit].

- **Photon Interaction Type** - In this framework no russian roulette is used to decide between absorption and scattering because of the aforementioned albedo multiplication. Instead we only use russian roulette to terminate the path after a certain number of minimum bounces or when a number of maximum bounces is reached. In-between, the maximum value over all albedo channels is used to determine if the path continues and the weight per channel is divided by the maximum albedo over all channels. This follows the implementation of Mitsuba [mit].

Caution

In general an extension to multiple color channels would yield the effect of dispersion when also taking into account refraction indices. That is, when a scattering event takes place, light waves of different wave lengths would be scattered into different directions according to the wavelength. We ignore this effect as we pack all channels into a single photon and trace a single path for the photon, only adjusting the photon color/power for each interaction, but not separating the directions by e.g. having separate photons for each color channel.

2.4.4 Nonlinear Extension

The method described in this section is adapted from [Gut05].

Another effect that we would like to capture in this framework are bent light curves that are caused by spatially varying refractive indices throughout a volume. A well known phenomenon showcasing this effect is a mirage:

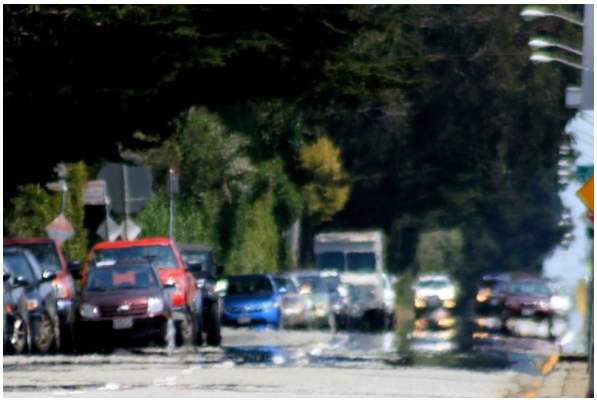


Figure 2.8: A mirage: The air close to the ground is hotter than the air above. As colder air is denser it has a larger refractive index than hot air, thus resulting in the air volume having spatially varying refractive indices. This causes light rays that pass through the air to be bent in the direction of the refractive index gradient, resulting in the distorted image. Image Source: `http://en.wikipedia.org/wiki/Mirage`

In our framework we do not include any type of temperature, but as we still employ a non-homogeneous volume we work under the assumption of spatially varying refractive indices and would like to include this in the Photon Mapping algorithm. Please note that the refraction indices in the framework are directly derived from the density at the current position in the volume using a simple quadratic curve. This is a strongly simplifying heuristic, as a physically plausible calculation of the refractive index would be much more complicated, having to account for the current material, density, temperature and wavelength. Using this heuristic however does not restrict the generality of the method used to calculate the bent light curves, it only biases the refraction indices that are used as a parameter in the calculation.

One restriction using this method is the simplification of using a single refractive index at a position. In reality, the refractive index also varies with the wavelength and thus results in dispersion of the light because it is bent with a varying magnitude according to the wavelength.

To derive a method to bend the light curve, we start with the relationship of the curved light path $l(s)$ and the scalar field $n(\mathbf{x})$ of refractive indices with the ray equation of geometric optics [Sun08]

$$\frac{d}{dl}(n\frac{dl}{ds}) = \nabla n \tag{2.14}$$

Equation (2.14) can be rewritten as a system of first-order differential equations using $v = n\frac{dl}{ds}$:

$$\begin{aligned} \frac{dl}{ds} &= \frac{v}{n} \\ \frac{dv}{ds} &= \nabla n \end{aligned}$$

Now a piecewise-linear approximation of the curve can be acquired using an Euler forward discretization:

$$\begin{aligned} \mathbf{l_{i+1}} &= \mathbf{l_i} + \frac{\Delta s}{n}\mathbf{v_i} \\ \mathbf{v_{i+1}} &= \mathbf{v_i} + \Delta s \nabla n \end{aligned} \tag{2.15}$$

Δs is an arbitrary step size and ∇n is the gradient of the scalar field of refractive indices. Please note that the real step size in the sense of the euclidean distance between two adjacent curve points using this equation is (in general) not Δs and also not constant because it depends on the refractive index and the gradient.

To anticipate one question that might arise with the following section on time dependency: In the framework the time of flight in-between two adjacent curve points is determined by calculating the distance between the current and the next curve point and calculating the required time with respect to the modified speed of light by the refractive index at the current position. Without the nonlinear extension, the time of flight could be calculated directly using Δs but using this extension, the actual non-constant step size has to be taken into account.

The gradient at an arbitrary position within a volume, represented by a voxel grid, can be approximated using e.g. Central Differences [IK]:

$$\nabla n(\mathbf{x}) \approx \begin{pmatrix} \frac{V(\mathbf{x}+\Delta V_x)-V(\mathbf{x}-\Delta V_x)}{2\cdot\|\Delta V_x\|} \\ \frac{V(\mathbf{x}+\Delta V_y)-V(\mathbf{x}-\Delta V_y)}{2\cdot\|\Delta V_y\|} \\ \frac{V(\mathbf{x}+\Delta V_z)-V(\mathbf{x}-\Delta V_z)}{2\cdot\|\Delta V_z\|} \end{pmatrix} \tag{2.16}$$

Where V($\mathbf{x}$) is the value of a volume V at position $\mathbf{x}$ and the ΔV_x,ΔV_y,ΔV_z vectors represent the distances between two voxels of the volume in all three directions (assuming the grid is equidistant).

Now we have the tools to calculate curved light paths, but how do we employ it in volumetric photon mapping?

First, a problem has to be addressed that cannot be solved easily. Assuming a typical Volumetric Photon Mapping implementation, the single scattering term would be calculated by sampling the light source using ray marching from the current position to the light source, giving us the transmittance, incident light direction and distance from the light source. Under the assumption of bent light curves, this information cannot be gained using ray marching, as the light curve that would pass the current position is unknown. In fact it is not only unknown, but ambiguous as the following Figure 2.9 demonstrates:

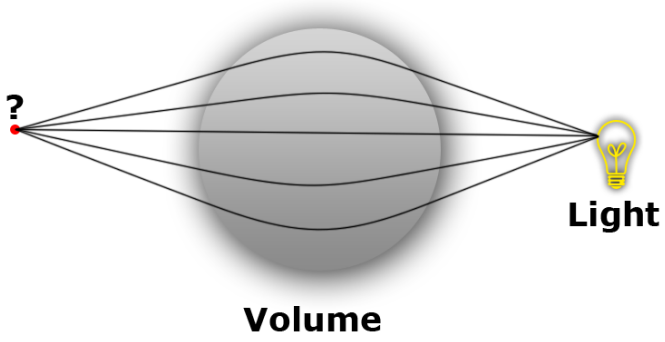


Figure 2.9: At the position marked as the red dot, the incident light direction is highly ambiguous because light is bent to this position from multiple possible paths. Please note that this does not depict multiple-scattering, but light that is exclusively bent because of spatially varying refractive indices. The result at the focus point cannot only be treated like a single scattering event.

So if we decide to ray march from the current sampling position towards the light source, we do not know the direction where the ray would pass by the position of the light source. Conversely, ray marching from the light source yields the same problem. In this framework we implemented and tested an approach to pre-calculate a single scattering grid by tracing bent light curves from the light source, splatting the necessary values to the grid using an interpolation scheme and looking up the saved grid values during rendering. This works fairly well for volumes with only very small variations in the scalar field of refractive indices. But this does not solve the general case, as easily seen in Figure 2.9: At the focus point of the light, the incident light direction is ambiguous and thus cannot be properly captured in a grid that expects unique values per data point. As a consequence to this, calculating the single scattering term via ray marching is restrained from and the direct term is also calculated via the photon map. The only changes necessary for this are to skip the secondary ray marching to the light source completely, and also to store the first interaction in the photon map during photon tracing. The downside of this change is that the number of photons required to achieve renderings that are both detailed and noise-free is increased multiple orders of magnitude compared to the split approach.

By including the direct term in the photon map, we meet the prerequisite of only having to forward-trace bent light curves in the photon mapping algorithm:

- **Photon Tracing** - For tracing the path of a single emitted photon, we bend the photon curve in the volume according to Equation (2.15), still accumulating optical depth until we reach the next interaction point. We directly couple the step size for accumulating optical depth with the step size for calculating the next position on the bent photon curve. The new direction after scattering is still determined using the phase function and the same steps are then repeated for finding the next point of interaction.

- **Photon Gathering** - Here we directly apply Equation (2.15) to the ray marching per pixel that is already in place. Also the ray marching step size is used for the step size parameter Δs in Equation (2.15). So the actual step size is determined by the distance between the current position and the next position that is calculated using Equation (2.15).

This captures both the effect of bent viewing rays, as well as the bent photon curves, which in turn will allow us to capture effects such as volumetric caustics.

2.4.5 Time Dependency Extension

This section introduces the main novelty of this thesis. As we want to visualize the propagation of light echoes over time, the distance scales we work on are rather light years than meters. Taking into account the finite speed of light propagation, the time dependency of light propagation is essential to properly display a light echo. Traditionally CG lighting methods ignore the finite speed of light as it simply does not have a notable visual impact on *earthly* distances. In this case, we present an approximation to the transient rendering equation (see Equation 2.6).

Please note that by definition, if we talk about the *current time* in this context, this always refers to the time of the observer, or in terms of rendering, the time at the current position of the camera.

Another very different but also very interesting use case for the time dependency extension is the work of Prof. Ramesh Raskar at the *MIT Media Lab*[1]. In December 2011 his lab received a lot of attention with the publication of an ultrafast imaging system that is able to reconstruct and visualize sequences of how light propagates through liquids and objects of a scale encountered in everyday life [Mar].

[1] http://web.media.mit.edu/~raskar/

(a) **(b)**

Figure 2.10: Left: A single frame captured within two-trillionths of a second shows how the light propagates along objects. Image Source: Camera Culture Group, MIT Media Lab. Right: A comparable scene built with Maya and rendered with our framework. For more images from this sequence, see Chapter Results.

As the time dependency extension will treat parameters like the current time and the speed of light as arbitrary values, our framework will also allow to render sequences similar to the sequences presented by the Raskar group (in the sense that the framework renders the sequence, as opposed to the sequence being captured and reconstructed with a high-speed camera).

The only changes required in the framework to visualize light propagation on small scales are to revert the size of the volumes used from extends in the range of light years to a centimeter or meter object size and then to reduce the playback speed until light propagation becomes visible. Thus we also present renderings depicting light propagation on small scale objects in Section 4.3. For the rest of this section however, the descriptions are still matched to the light echo use case.

The extension of Volumetric Photon Mapping to take into account the finite speed of light propagation is relatively straightforward, since we impose three important restrictions:

- The light source itself is an isotropic point light. Thus we do not need to store the initial direction and initial position of photons during emission as it would be required for e.g. an area light source.
- The light source position is static. The only thing allowed to change over time is the light flux. This still meets our requirements as we regard the star's position as constant and emulate the supernova as a short but strong variation of the light's flux over time.
- The volume itself as well as its position in space is static. This also complies with observations of real light echoes: Although real supernovae certainly dash a significant amount of matter into space, the main contribution to the visual impression of the light echo stems from now illuminated intergalactic dust and nebulae that have surrounded

the star for a long time and whose positional dynamic during the timeframe of a light echo is negligible.

This allows for a very important simplification. The distribution of photons in the photon map does not change over time. The original Photon Mapping algorithm also deviates from a physical model in the sense that photons represent a certain amount of energy thus abstracting the fact that real light flux is governed by the amount of photons passing a certain position per unit time. As we only want to change the light flux over time, photons do not have to be re-emitted over time because of this simplification and additionally because both the light source's position as well as the volume's position are fixed, the photon map can be precomputed for the entire simulation.

The most important change is that photons cannot store the flux they carry, because this is unknown at emission time and changes during runtime. Traditional Photon Mapping assigns the flux a single photon carries by assigning each photon the flux of the light source divided by the number of photons that have been emitted during the tracing stage. Assuming an RGB color model, the stored flux along a path of stored photons in the photon map may change because of interactions with the volume/material. We have to adapt this approach to enable a dynamic integration of light flux depending on the current time.

Light flux

The light flux is represented by a per-channel flux curve $\Phi_l(t)$. The lookup parameter t of the curve is in the local time of the light source.

Time-dependent Photon Tracing

The photon is changed to not store the flux it carries anymore, but the time of flight (the time passed since the emission from the light source up to the time of the photon interaction) and a weight for each photon. The time difference since the emission is needed for the photon gathering stage to be able to compute a lookup time for the flux the photon carries at a specific point in time. The weight is needed to still be able to reflect interactions of the photon with a material. Usually in Photon Mapping a material interaction would be achieved by changing the flux of the stored photon, but since the flux is not known at this point, we store a weight per channel. Photons emitted from the light source always start with a weight of 1.0 that is subsequently multiplied with the albedo for each scattering event according to the description in Section 2.4.3. During rendering we reconstruct the flux of each photon by multiplying the photon's weight with the dynamically queried flux at the current time for the photon.

Storing the time since emission along with the photon is straightforward. The most important thing is to pay attention to the spatially varying refractive indices as the speed of light varies with the refractive index. The photon tracing now keeps track of the time as follows:

1. Photon tracing starts with the emission at the light source at time $t_0 = 0$.

2. If the curve between the starting point and the first interaction point contains a segment outside the volume, determine the length of this line and assume a refractive index of n = 1.0, thus directly using the vacuum speed of light c for calculating the photon travel time for this part.

3. When calculating a new position $\mathbf{l_{i+1}}$ of the nonlinear photon tracing algorithm within the volume according to Equation (2.15), calculate the euclidian distance d_{step} between $\mathbf{l_i}$ and $\mathbf{l_{i+1}}$.

4. The time stored for the photon at $\mathbf{l_{i+1}}$ is

$$t_{i+1} = t_i + \frac{d_{step} \cdot n(l_i)}{c}$$

   This means that we assume the refractive index n at position $\mathbf{l_i}$ to be constant between $\mathbf{l_i}$ and $\mathbf{l_{i+1}}$. This will modify the speed of light in the volume to c/n for this segment and calculate the time difference accordingly.

5. Accumulate and store the time for each stored photon. Reset the time to zero each time a new photon is emitted.

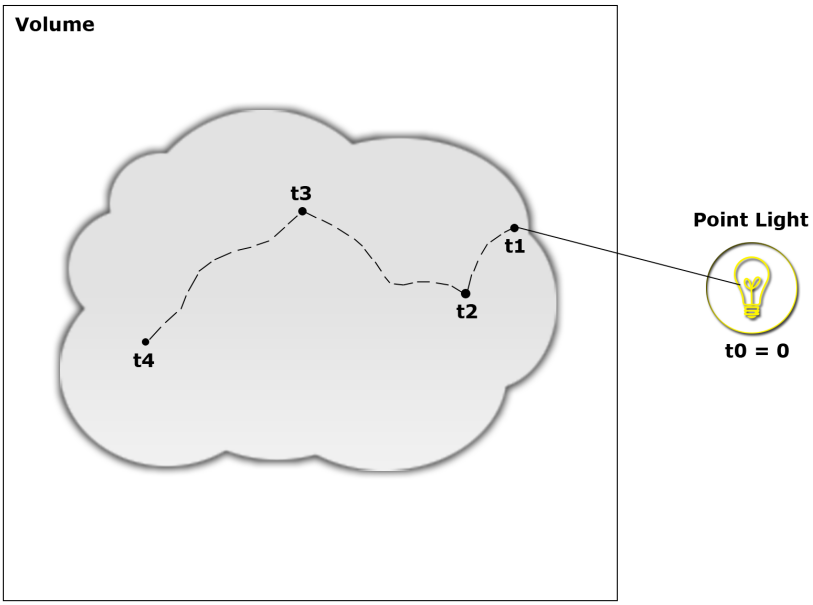


Figure 2.11: Illustration of a light path during time-dependent photon tracing. The solid line is the path of the photon when it is not yet affected by spatially varying refractive indices. The dashed curve shows the bent path of the photon. The dots represent the photons stored at scattering events on this path along with the time since emission (t1 to t4).

This approach provides the basis to collect all information necessary during photon gathering to render the scene at any point in time.

Time-dependent Photon Gathering

In general the photon gathering algorithm stays the same. The equation for the in-scattered radiance is modified only slightly compared to the original Equation (2.12):

$$L_i(\mathbf{x}, \vec{\omega}, t) \approx \frac{1}{\sigma(\mathbf{x})} \sum_{p=1}^{n} f(\mathbf{x}, \vec{\omega}_p^{'}, \vec{\omega}) \frac{\Phi_{l,p}(t_{lookup}(t))}{\frac{4}{3}\pi r^3} \tag{2.17}$$

The only change is the time parameter of the in-scattered radiance that is passed to $\Phi_{l,p}(t_{lookup}(t))$, representing the time-dependent flux of the photon. As $\Phi_l(t^{'})$ is defined to be in the local time of the light source, and t is defined to be in the local time of the observer, $t_{lookup}(t)$ is used convert the observer time to the time of the light source and include the required time of flight:

$$t_{lookup}(t) = t - (t_r + t_d + t_p)$$

- t - is the current time of the observer.
- t_r - is the time required for the light to travel from the observer to the current position in the viewing-ray ray marching algorithm. t_r just has to be tracked during ray marching, also taking into account the modified speed of light. This can be done using the same approach as proposed for photon tracing.
- t_d - is the time required to travel the distance from the current ray marching position to the stored photon position. We use the refractive index at the viewing-ray ray marching position to modify the speed of light for the time calculation.
- t_p - is the time since emission that is stored in the photon, so the time required for the light to travel from the light source to the photon.

So we take the current time at the observer, subtract the time required for the light to travel from the light source to the observer and thus get the local time at the light source, which enables us to make a lookup at the light source for the flux that has been emitted at this point in time.

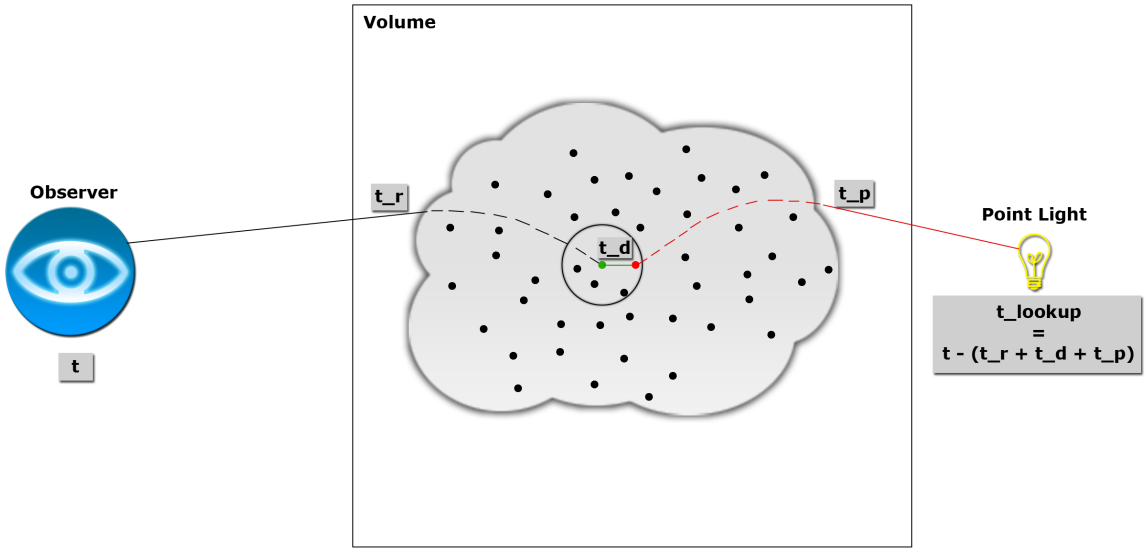


Figure 2.12: Illustration of time-dependent photon gathering. The black path shows a bent viewing ray, the green line the query of a single photon during the nearest neighbor search and the red path shows the path the photon originated from.

Dimensions

As already noted, the dimensions that are usually observed in the context of light echoes are rather light years than meters and years instead of seconds. Because we would like to use realistic values, light years are the base distance unit and years are the base time unit in this framework. Values are converted back to meters or seconds if required.

So in itself, the dimensions used do not demand for a special extension but enforce that some calculations in the framework have to be done using double precision to prevent numerical issues. As the volume's extends can reach a magnitude of light years, scattering and absorption coefficients are very low. On the other hand, the light flux values are very large since the light source is supposed to represent a star. For example, the sun has a power of about 10^{26} Watts [Wil04].

2.4.6 Materials Extension

In order to be able to construct additional interesting sample scenes for our framework, we also use an extension to include different materials within a volume. In our context we define a material as a mapping from volume density at a position $\mathbf{x}$ within the volume to the RGB scattering coefficient σ, the RGB absorption coefficient α, the g parameter of the H-G phase function and the refractive index n. This allows us to model materials with a different color, refractive index and scattering behavior.

We realize the assignment of the material at a position within the volume through an additional material volume that is devised from the export pipeline we use to construct our sample scenes:

1. **Maya** - Used to construct a scene using standard polygon models.
2. **OBJ Export** - The scene is exported using Maya's built-in exporter for the OBJ format.
3. **Voxelizer** - We use the application Voxelizer[2] to convert the OBJ surface based scene to a voxelized representation including a distance field. For each scene, this export has to be done per object in the scene. The application requires selecting a seed point for the conversion. If the scene consists of multiple objects, only the object that holds the seed point will be exported at a time. We exploit this behavior to later on merge multiple volumes to a single density volume and a volume of material IDs.
4. **Voxelizer Converter** - This converter application was written in-house by Marco Ament and converts volumes in the Voxelizer format to a raw density format. Here we use the distance field to construct a smooth density increase for the boundaries of the voxelized objects. This helps to significantly reduce artifacts that would appear in conjunction with the nonlinear ray marching and a volume that was constructed from a surface using a simple binary inside/outside density with respect to the object.
5. **Volume Merger** - The volume merger is a component of our framework. It takes an arbitrary number of exported density volumes as input and constructs a unified density volume and a material volume. The density volume is a simple additively combined density volume of all input volumes. The material volume is constructed by assigning an ID to each input volume and writing the ID to the material volume for every voxel where the density of the input volume is larger than zero. As we do not support blending of materials, this means that the objects (areas with a density greater than zero) should be spatially separated from each other, so that there are no overlaps. In case this happens anyway, existing IDs will be overwritten by the subsequent input volume.
6. **Material Lookup** - During rendering, for each sampling position the material values will be determined by a lookup in the material volume at the current position. The lookup supplies the material ID at the current position, and the material ID is used to determine the material parameters during runtime. Since we do not support blended materials or multiple materials at the same position, the lookup in the material volume does not use interpolation but a simple nearest neighbor lookup.

The material extension does not require any adaptation of the aforementioned extensions. It is integrated simply by including the lookup of the material parameters with respect to the current position within the volume instead of using a set of globally defined material parameters for the volume.

[2] http://techhouse.brown.edu/~dmorris/projects/voxelizer/

2.4.7 Conclusion

Chapter 2.4 presented the necessary background to understand the Volumetric Photon Mapping method and the extensions for time-dependent light propagation, nonlinear effects caused by spatially varying refractive indices, the extension to colored materials and multiple materials within a single volume. Please note that the time dependency extension does not capture relativistic effects. For a comprehensive examination of this field of study, see [Wei01].

Details on data structures and the implementation have not been discussed so far, but this will be caught up in the description of the framework in Chapter 3.

In the end the decision to use Volumetric Photon Mapping to visualize light echos was based on the following points:

- Direct integration of multiple scattering that is especially desirable in conjunction with the time dependency to show delayed light bounce effects in the volume.
- Direct integration of nonhomogeneous volumes with anisotropic scattering, as this reflects the composition of galactic dust and nebulae.
- The possibility to include nonlinear effects caused by spatially varying refraction indices. This however is at the cost of having to include single scattering in the photon map and thus having to increase the number of photons in the simulation multiple orders of magnitude.
- Straightforward integration of time dependency by storing the time of flight for each photon in the photon map.
- Parallelization of the gathering stage by using a grid approach to gather photons. This enables porting the algorithm to CUDA in an efficient manner. For more information on this, see Chapter 3.

2.5 Beam Radiance and Photon Beams

Classical Volumetric Photon Mapping as described in [JC98] and [Jen01] has basically been superseded by the more recent and more efficient Photon Beam and Photon Query methods described in [JZJ08] and [JNJ11]. This section briefly argues why these methods were not an ideal candidate to use as a basis for the extensions desired for this framework.

Beam Query

This approach changes the photon gathering process. Classical Volumetric Photon Mapping collects photons at discrete sampling positions along a viewing ray within a sphere around the current sampling position. The beam query method changes this approach to collect all photons along a viewing ray at once, if their distance to the viewing ray is close enough.

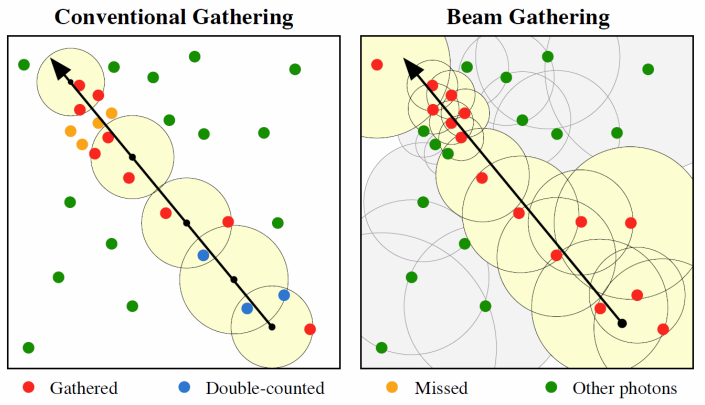


Figure 2.13: The beam query method collects all photons along a viewing ray, eliminating issues with missed or double-counted photons in classical Volumetric Photon Mapping. Image source: [JZJ08]

This imposes the requirement to collect all photons along a ray, which is solved using a Bounding-Box-Hierarchy, which would have to be efficiently parallelized and ported to CUDA. The set of photons surrounding a discrete location within a specific radius can be ported to CUDA more easily and efficiently using a grid (see Section 3.2 Photon Tracing). Also in our context it is unavoidable to ray march along the viewing ray as the ray is bent because of the spatially varying refractive indices and the need to track the time offset, because the power of each photon has to be determined dynamically with the time dependency extension. This invalidates the suggested ray - bounding-box hierarchy intersection traversal to collect the photons and poses the question on how to efficiently gather all photons along a nonlinear curve in a way that does not degenerate to the classical discrete sampling approach.

Photon Beams

While the Beam Query approach only affects the photon gathering stage, Photon Beams extend both the Photon Tracing and Photon Gathering stage. The idea is to store photon beams instead of photon points during photon tracing.

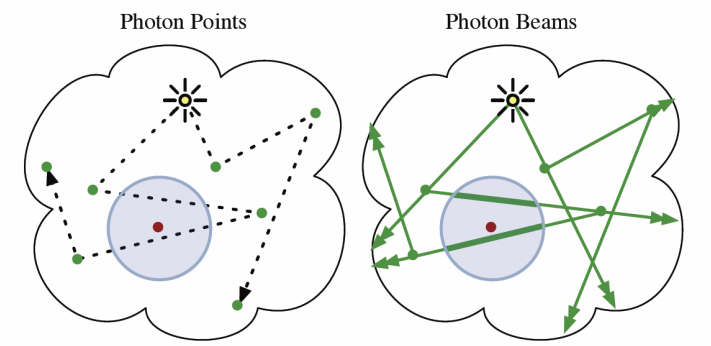


Figure 2.14: The photon beam method stores the full trajectory of a photon and increases the quality of radiance estimation. E.g. a photon query within the blue circle would yield a better result using photon beams. Image source: [JNJ11]

In conjunction with spatially varying refractive indices, this also poses the problem that a photon beam cannot be represented by a ray as the trajectory is possibly curved, again posing the question on how to store photon beams in a way that does not degenerate to storing samples and thus effectively reverting to classical photon tracing. Also, an efficient way of querying the nonlinear curves would have to be found. For each point on a photon beam there would have to be a way to determine the time since emission from the light source for the time dependency extension, which is also nonlinear because the spatially varying refractive indices change the speed of light along the trajectory.

Both methods can be used in conjunction, but we also discard this option as this would require solving the combined issues of both approaches. So in the end the decision not to use the newer beam approaches as a basis to integrate all aforementioned extensions is primarily based on the number of additional issues introduced, where developing a solution is not feasible within the timeframe and scope of this thesis.

2.6 CUDA

All information comprised in this section is contained in [Cor].

The framework developed during the work on this thesis uses CUDA to accelerate the rendering of the Photon Mapping simulation. In order to understand the possibilities as well as the restrictions that inherently come with this approach, this section gives a short introduction to general purpose computation on GPUs and CUDA in general.

2.6.1 Introduction

Nowadays Graphics Processing Units (GPUs) are not only a means to accelerate traditional rasterization based image synthesis but have evolved into a massively parallel many-core

processor that can be used for a large variety of applications. It is especially well suited to problems that can be solved with data-parallel computations. The general term used for computing on the GPU is *GPGPU* (General Purpose GPU).

CUDA is a computing architecture for GPUs that was introduced by NVIDIA in 2006. It features its own dedicated language, called *CUDA C* but also allows the usage of other languages such as OpenCL or DirectCompute. CUDA C is a superset of the C programming language that features a number of extensions to address some necessities associated with GPU programming.

At the very core of CUDA are *Kernels*. On first glance a kernel is nothing but a C function that is tagged with a declaration specifier to mark it as a kernel. A kernel differs from a regular function because it will be executed on the GPU as opposed to the CPU and it will be executed N-times in parallel on N different CUDA threads. The number of threads that will be used in parallel can be specified when the kernel is invoked.

The execution pattern for kernels with respect to threads however is more complex than just stating a single number to be used as a thread count:

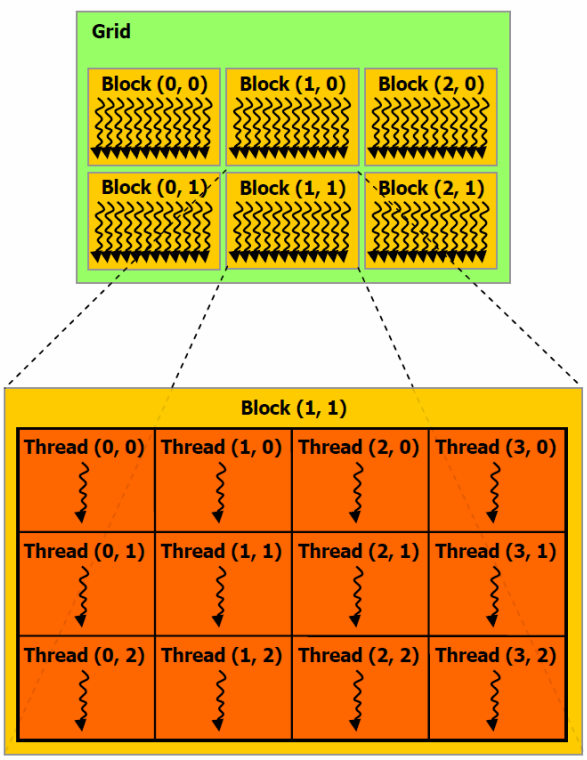


Figure 2.15: CUDA threads are organized in blocks which in turn are associated with a grid. Image Source: [Cor].

In general each thread resides in what is called a *Thread Block*. A block groups threads in one, two or three dimensions. Each thread has a unique thread ID within the block. It is important to note that the dimensionality of threads within a block has no firsthand geometric meaning and can be imagined just like the dimensionality of an array. This however gives the opportunity to organize threads in a way that matches the dimensionality of the problem that is due to be solved and can then of course have a sensible geometric meaning, depending on the problem. For an algorithm that takes a 2D picture as input and manipulates each pixel, an obvious layout is to organize threads in 2D blocks and let each thread manipulate one pixel, but in the end the layout is up to the programmer.

One important hardware restriction for blocks is that one block is always executed on a single CUDA core. One core may execute multiple blocks, but the execution of one block cannot be shared among multiple cores. On top of this there is another layer of abstraction called *Grids*. Blocks are also organized in means of grids. The relationship between blocks and grids is very similar to that of threads and blocks: Blocks are organized in grids in either one, two or three dimensions and are given a unique ID within the grid. The hardware restriction for a grid is that a single grid always has to be executed on a single GPU and the grid is the uppermost structure of thread organization in CUDA.

The reason why threads are embedded in two layers of abstraction are scalability and the possibility for both the programmer and the GPU hardware to optimize the execution of kernels to efficiently use the processing power and memory of the GPU with respect to the problem that should be solved. In order to properly understand the limitations for thread organization, the next section presents the memory hierarchy of CUDA. In general the efficient use of GPU memory, as well as the use of efficient GPU memory, is key to achieve high performance with CUDA.

2.6.2 Memory Hierarchy

CUDA provides access to a number of different memory spaces that can be used inside a kernel.

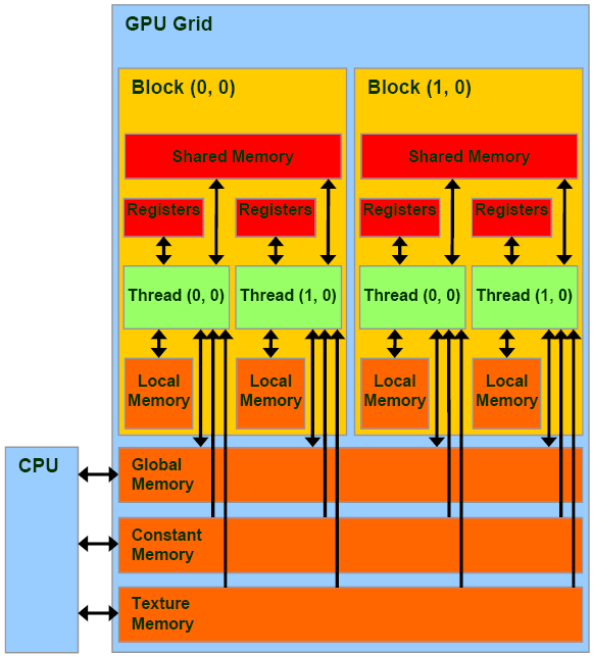


Figure 2.16: Schematics of different memory types in CUDA. Image source: `http://www.drdobbs.com/parallel/215900921`

Global Memory

This equates to the VRAM of the GPU and can be accessed from all threads. The amount of VRAM is comparably large, but it is also the slowest form of memory available. A typical VRAM size on current hardware is about one GB. The typical access time on current hardware, assuming the access is uncached, is a a few hundred cycles. Please note that the memory access pattern is also an important additional factor with regard to the access time. Coherent access is faster than random access of memory.

Constant Memory

Constant memory is also global memory and is read-only but provides a caching mechanism that speeds up parallel memory access of the same location from parallel threads. This type of memory is suitable for e.g. variables that are read often but do not have to be changed.

Texture Memory

Texture Memory is also global memory, is also read-only but is able to use the GPU's texture caching mechanisms. This also enables the use of hardware filtering capabilities, e.g. bilinear or trilinear hardware filtering. With regard to caching and a 3D volume, organizing memory in form of a space-filling curve is advantageous when accessing neighboring voxels, but has the downside of imprecise interpolation.

Local Memory

This type of memory implies that it is local in the scope of each thread. One important thing to note is that this is not a dedicated memory component on the GPU but in the end also just equates to an automatically managed part of the global memory and thus should be avoided for performance critical parts of a kernel if possible. Local memory is e.g. used when the amount of local data in the thread is too large to be kept in faster types of memory.

Shared Memory

Shared Memory is memory that is local to a specific block. Shared Memory is in fact a dedicated memory component that is available per multiprocessor and provides very fast memory access with a latency of only a few cycles. A typical size of shared memory on current hardware is 48 KB per multiprocessor. Shared memory is suitable for e.g. a user-managed cache.

Registers

Registers are used to store kernel parameters and local variables per thread. Registers are as fast as shared memory and on current hardware a typical size is 32K 32-bit registers per multiprocessor.

2.6.3 Conclusion

CUDA offers an easy and flexible way to make use of parallelization. The first step in porting an algorithm to CUDA is to find a sensible way of matching the problem to the offered dimensionalities of threads and blocks. Regarding the parallel execution of the algorithm itself, it is important to note that the execution performance is not only restricted by raw computation power, the pure amount of instructions in the kernel or synchronization points, but also by the memory bandwidth.

The challenge in implementing efficient algorithms for CUDA is often directly associated with being able to use a fast type of memory. This implies that shared memory should be used as often as possible but unfortunately many problems require processing quantities that rule out the usage of shared memory. When having to fall back on the different types of global memory, striving for caching efficiency is the next important step. This implies that if possible, algorithms should be optimized for coherent memory access patterns to minimize the probability of cache misses.

As it will turn out, the CUDA version of the Volumetric Photon Mapping is primarily restricted by memory bandwidth.

3 Implementation

This chapter describes the framework developed during this thesis. It is not an exhaustive documentation of the system design, but tries to focus on the aspects most critical and relevant for the practical implementation. The framework implements Volumetric Photon Mapping including the RGB, nonlinear, materials and time dependency extension.

3.1 Overview

The framework has been developed using C++ for the backend and photon tracing. The GUI is realized using Qt. The rendering itself including photon gathering is implemented in CUDA. OpenGL is used to display the final rendered result, employing CUDA - OpenGL interoperability.

3.1.1 Frontend

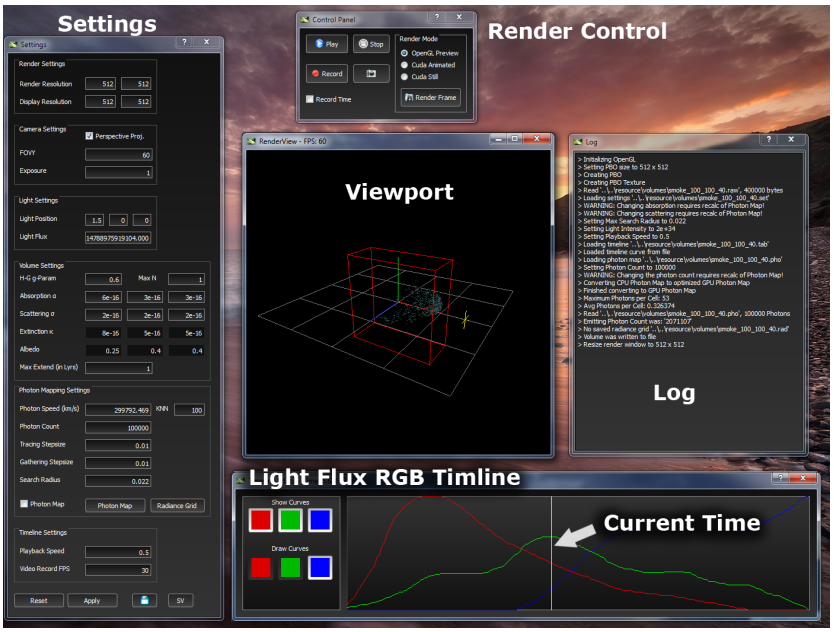


Figure 3.1: Screenshot of the framework frontend.

The GUI offers the viewport and various options. It was developed using C++ and Qt. The only component of interest with regard to the photon mapping algorithm is the light flux timeline:

The timeline controls the light flux over time, separated in RGB color channels. Internally the timeline is stored as 256 entry, three byte valued lookup table that is uploaded as a 1D-Texture to the kernel. The time range covered by the timeline is adjustable and is only used in the kernel for time-dependent flux lookup. The dynamic range of the table is steered by an adjustable base flux value that is multiplied with each color channel (who in turn feature a value range of [0,1]) to get the light's flux value during lookup.

3.1.2 Backend

The following diagram depicts the stages involved in rendering an image in the framework. *Photon Tracing* and *Photon Gathering* are explained in more detail in the subsequent sections.

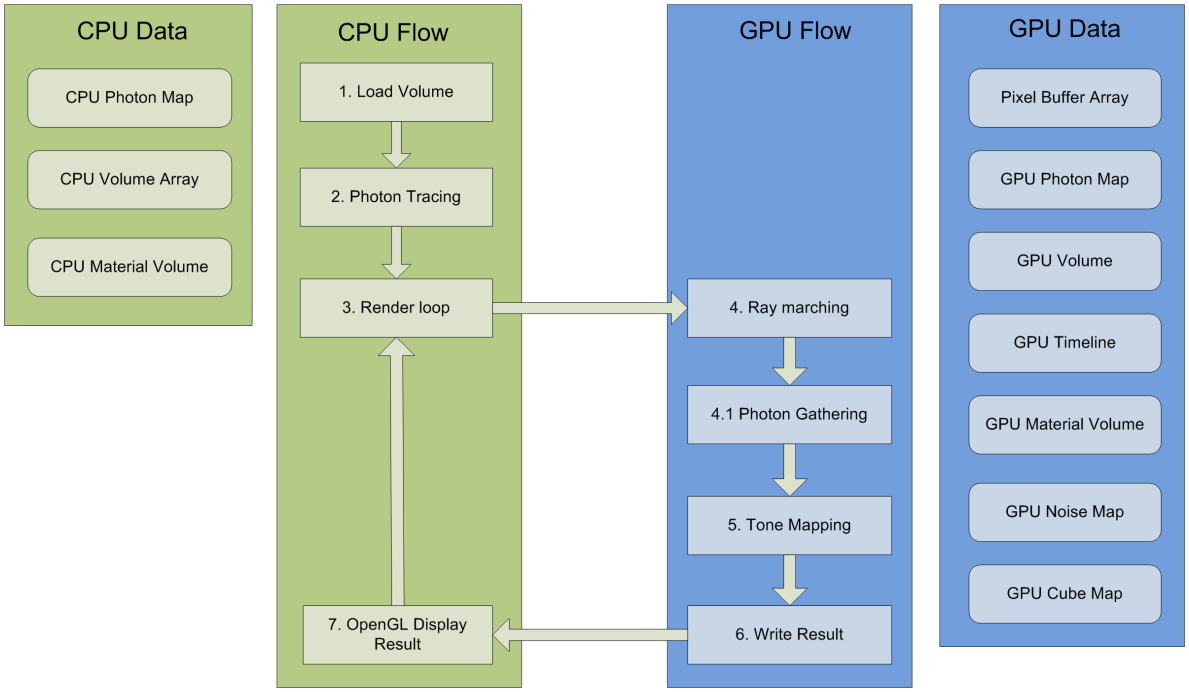


Figure 3.2: Coarse overview of execution workflow in the framework.

1. **Load Volume** - The volume is loaded from a (RAW) file into main memory as an array. In case a material volume exists for the density volume, it is also loaded from file.
2. **Photon Tracing** - The Photon Map is calculated in a preprocessing step. The tracing itself is run entirely on the CPU but afterward the *CPU Photon Map* is converted to a format compatible for GPU usage. If the photon map was already created in a previous session, it will be loaded from file.

3. **Render loop** - Before the rendering is launched, there is an update check for all data required on the GPU. If the data is not yet available e.g. when rendering the first frame, or updated during runtime, the data will be (re)uploaded to the GPU. Afterward the CUDA Kernel is called to render the image.
    - Pixel Buffer Object (PBO) - This is the main object for CUDA-OpenGL interoperability. On the CUDA side it is used as an array with write-access. On the OpenGL side the PBO is bound to a 2D-Texture for rendering. The PBO has the size of the framebuffer and will be created once during runtime and is then updated per frame.
    - GPU Volume - The CPU Volume Array is uploaded and bound to the GPU as a 3D-Texture, using hardware tri-linear interpolation.
    - GPU Material Volume - The CPU Material Volume (if it exists) is uploaded and bound to the GPU as a 3D-Texture, using nearest neighbor sampling.
    - GPU Timeline - The light flux timeline curve in the GUI is uploaded and bound as a 1D-Texture, using hardware linear interpolation during lookup.
    - GPU Photon Map - During Photon Tracing the GPU version of the photon map is prepared and will now be uploaded. More on the data structure, see Section 3.2.
    - GPU Noise Map - The noise map is created on the CPU side and uploaded as a 2D-Texture. It is used to offset eye-rays to reduce artifacts.
    - GPU Cube Map - This is the cubemap texture that is used for the skybox. It is load from six image files representing the six sides of the skybox cube.
4. **Ray marching** - The CUDA kernel is executed with one thread per rendered pixel, identifying each thread with a viewing ray for ray marching. The viewing ray is offset using the noise map texture. After the last step of ray marching, there is a lookup in the cubemap texture using the latest ray marching direction to provide a background to the scene.

    4.1 **Photon Gathering** - During ray marching photons are collected from the GPU photon map. For a detailed description, see Section 3.3.
5. **Tone Mapping** - As the result of ray marching yields a radiance value per color channel, the value needs to be reduced to a dynamic range that can be displayed before the value is stored.
6. **Write Result** - Each thread writes the final color value to the PBO at the respective pixel's designated memory position.
7. **OpenGL Display Result** - After the kernel returns from execution, OpenGL is used to display the final result. The PBO is bound as a 2D texture resource and OpenGL renders a fullscreen quad using the texture without lighting to display the result. If in time-execution mode, the current time is advanced before calling the render loop again.

Scene Elements

A scene within the framework consists of the following entities:

- **Camera** - there is exactly one camera that can be placed in the scene, representing the observer of the light echo. The position and direction of the camera can be modified between successive frames without repeating any preprocessing steps.
- **Light source** - there is exactly one point light source in the scene. Its position can be modified before preprocessing and calculation of the photon map. If the position of the light source is supposed to be changed, the photon map has to be recalculated.
- **Volume** - there is exactly one volume in the scene. The volume's position is always fixed at the origin, but its extends in the scene can be adjusted. This also requires a recalculation of the photon map.
- **Skybox** - to enhance the visuals, a background to the scene can be provided. This is realized in form of a skybox.

3.2 Photon Tracing

Chapter 2.4.1 on Photon Tracing in the fundamentals already outlined the photon tracing technique, but left out implementation specific details like the photon map data structure. In this section we focus on the photon map itself, especially with regard to the CUDA implementation.

As the photon map itself is decoupled from the scene volume/geometry, it is not inherently bound to any specific data structure. But of course for the sake of efficiency and render times, the usage of an optimized data structure with regard to nearest neighbor search is of the essence. As for our use case the photon map can be precomputed for the entire simulation, the most important point related to the data structure is the nearest neighbor lookup efficiency during photon gathering in the CUDA kernel.

Photon Mapping was introduced as a CPU based, non-parallelized technique, employing a KD-Tree as the data structure for the photon map [JC98]. This however is not optimal for a parallelized GPU implementation as the measurements in [Fle09] indicate. We adopt the proposal from [Fle09] to use a grid as the photon map data structure. This technique in turn was originally described for CUDA in [Gre08] in the context of a particle system, but can be adapted for photons in a photon map, allowing for a simple space partitioning that is suitable for GPUs.

The following description of the grid implementation is based on [Fle09] but has a few simplifications, as in our framework the photon tracing stage is still non-parallelized and CPU based. This allows to collect photons in a simple list during tracing and then create the optimized photon map grid from it later on.

The data structure to represent the photon map is composed of a table of all photons from the photon tracing stage and a hash table that allows efficient access of the photons in the photon

table. The photon table is implemented as a fixed size array. Its size is determined by the number of photons that are stored during the photon tracing stage. The hash table is also a fixed size array, holding a position and an offset for the photon map per entry, as each entry in the hash table represents a cell within the grid.

The size of the hash table equals the number of cells within the grid. The photons in the photon table are ordered by cells, so all photons within a specific cell are located next to each other in terms of the memory layout. The hash table entry for the respective cell holds the starting index for this cell's photons in the photon table and the number of photons in this cell, which can be used as an offset for the index to retrieve all the photons for the respective cell. The entries within the hash table are also ordered so that the index represents the linear cell ID. This allows to access the correct entry in the hash table directly by calculating the linear cell ID from the lookup position in 3D space. A typical size for the grid when using 10 millions photons, that still allows queries with a sufficient search radius, is around 200-300k cells in the grid.

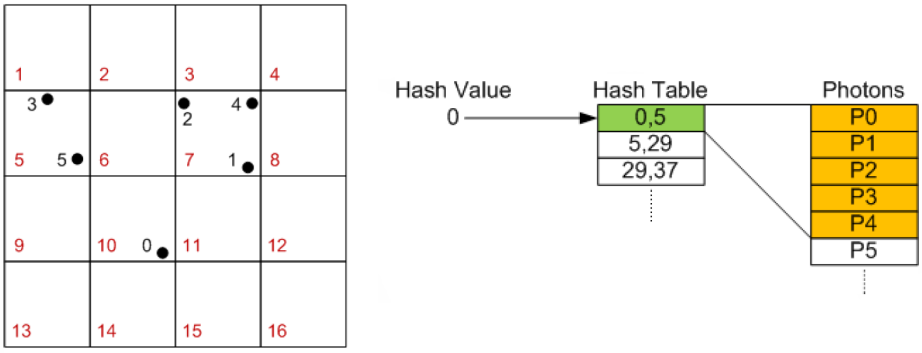


Figure 3.3: Photons are stored in a grid that consists of a table containing all photons and a hash table that stores access information for the photon table : Image source [Fle09].

The grid is uniform and its world space size and position are matched to the volume. The only parameter of the grid is a maximum search radius that is used to construct it. The same search radius will be used during photon gathering. The search radius is used as the side length of a single cell in the grid. The number of cells is calculated accordingly.

Calculating the linear cell ID from a world space position *pos* within the grid can be done by using the size of the grid *gridSize*, the search radius r used to create it and the position's local space coordinate within the grid, *posLocal*:

$$posGrid = \lfloor posLocal/r \rfloor$$

$$linearCellID = posGrid_x + gridSize_x \cdot posGrid_y + posGrid_z \cdot gridSize_x \cdot gridSize_y \quad (3.1)$$

The creation of the photon map now works as follows:

1. During the CPU based photon gathering stage the photon map data structure is a simple array where photons are just deposited sequentially during tracing.
2. After gathering, the two arrays containing the photon table and the hash table are created and initialized as empty.
3. The photon array from the tracing stage is analyzed by iterating over it and for each photon the cell ID is determined using Equation (3.1). The photon count value for this cell ID is increased by one in the hash table for this entry.
4. Iterate over the hash table and determine the starting positions by summing up the just calculated photon count per cell and store the current sum in the loop as the memory offset position.
5. Fill the photon table by placing the photons from the unordered photon array to their correct position in the photon table. This is achieved with an additional offset counter per cell, so that already placed photons will not be overwritten by newly inserted ones.

This approach to calculate the grid is not optimized for efficiency but simplicity as it is only required for preprocessing and not during runtime and it is also not of great importance with respect to the time dependency extension.

To make the grid accessible to the CUDA kernel, the photon table and the hash table are uploaded to the GPU as a set of 1D-Textures. This enables the use of the texture cache as we would like to profit from coherent memory access patterns. The texture access is set to not use interpolation and value normalization, thus we can use texture fetches as simple array lookups but powered by the GPU's caching capabilities.

Regarding the extensions used for Photon Mapping, there are no involved adaptations necessary to use them in conjunction with the grid-based approach:

- **RGB extension** - This only changes the struct used to represent a photon by storing a weight value for each color channel per photon.
- **Materials extension** - Only changes the way lookups have to be done in the photon tracing stage, but not the storage of photons.
- **Nonlinear extension** - This has no effect on the grid technique as it only affects photon positions and directions during the tracing stage.
- **Time dependency extension** - This only changes the struct used to represent a photon by including the time since the emission.

With the photon map grid being precalculated and prepared, the next step is to render the image which is done entirely on the GPU in the CUDA kernel.

3.3 Photon Gathering

This section takes a look at some of the implementation specifics of the actual CUDA rendering pass and the extensions used. The following description explains the steps taken to render a complete frame.

3.3.1 CPU Side Initialization

Before the CUDA render kernel is launched, all data is uploaded and bound to the GPU in case of the first frame rendered. This encompasses

- Volume (3D-Texture)
- Flux timeline table (1D-Texture)
- Photon Map: Photon Table (1D-Textures) and Hash Table (1D-Texture)
- Noise Map (2D-Texture)
- Skybox (Cubemap-Texture)
- Render Target (Pixel Buffer Object)

3.3.2 Kernel Launch

The CPU side launches the kernel with a two dimensional thread/block combination, matched to the resolution of the render target. This means that each thread is matched to render a specific pixel in the render target. All subsequent descriptions are *per thread* as each thread calculates a single pixel in the final image. As the calculation of a single pixel can be done independently from neighboring pixels, the description of the execution of a single thread is sufficient to describe the algorithm as a whole.

3.3.3 Eye Ray Generation

The generation of a viewing ray is the first step in the kernel and is achieved via the thread's ID and depends on the setting to either use a parallel or a perspective projection. The thread knows the position of the pixel to render via the x and y thread/block-ID parameter.

For a parallel projection, the ray direction is always the viewing direction of the camera that is passed as a parameter. The ray start position is the position of the camera with an offset according to the pixel position.

For a perspective projection, the starting position of the ray is always the position of the camera and the ray direction is reconstructed geometrically: The kernel receives two parameters holding a horizontal and vertical difference vector between two pixels and computes the position

on a virtual plane in front of the camera. An alternative to this approach would be to generate the direction via the inverse view-projection matrix.

3.3.4 Volume Intersection Test

The viewing ray is tested for intersection with the bounding box of the volume, returning the first intersection point in case the volume is hit. If the volume is not hit the radiance calculation in the volume is skipped, but computation still continues with the skybox lookup and tone mapping.

Advance time to ray marching start position:
This step is required in case the viewing ray intersects the volume but the camera is located outside the volume. The distance from the camera position to the volume entry point is calculated and the time required for the light to travel the distance, assuming a vacuum refractive index, is saved for further calculations.

3.3.5 Noise Map

To reduce artifacts the framework uses a framebuffer sized noise texture to generate offsets for the CUDA ray marching start positions. The noise texture is generated on the CPU using a random number generator and is then uploaded to the GPU as a 2D-Texture. The kernel uses the fetched texture value to offset the starting position of ray marching at the first intersection with the volume. The offset is

$$\mathbf{Startpos} = \mathbf{IntersectionPos} + \mathbf{CamDir} \cdot \mathit{RaymarchingStepsize} \cdot \mathit{NoiseLookupValue}.$$

Thus the offset distance varies between zero and the length of a single ray marching step.

3.3.6 Ray marching Loop

After the starting position and direction for ray marching are known, the ray marching loop steps through the volume, summing up in-scattered radiance while also tracking time and summed up attenuation along the view path.

1. Look up the density in the volume at the current position. This is required to calculate the scattering and absorption coefficients, as well as the refractive index at the current position.
2. Collect in-scattered radiance at both the current position and time from the photon map and add it to the tracked radiance, also taking into account the attenuation along the viewing path so far.
3. Multiply attenuation for the current step with the tracked attenuation factor to update it for the next step.

4. Advance the ray marching position with respect to the refractive indices, using Equation (2.15) for bent rays.
5. Advance the time tracker with respect to the distance between the current and the next ray marching step position and the current refractive index.
6. Check if the next ray marching position is still within the volume. If not, quit the marching loop. There is also a hardcoded iteration limit, because with the bent viewing rays there is the possibility of an infinite loop if the viewing ray goes in circles if the gradient of the scalar field of refractive indices allows it.

Ray marching Loop - Photon Map Lookup:
This is a closer look at Step (2) of the aforementioned steps. It is the most important part of the kernel and also computationally the most expensive. The estimate of the in-scattered radiance at the current position is achieved by a nearest neighbor search around the current position in the photon map. It is governed by three parameters: the lookup position, a maximum search radius and a maximum number of collected photons. The lookup position is determined by the ray marching and needs no further investigation. The maximum search radius and the maximum number of photons collected per query have to be set by the user before the kernel is launched. The maximum search radius in fact even has to be set before the photon map is precalculated, as the photon map grid layout is determined using the maximum search radius.

To explain this part, we first revisit Equation (2.17) that governs the collection of photons,

$$L_i(x, \vec{\omega}, t) \approx \frac{1}{\sigma(x)} \sum_{p=1}^{knn} f(x, \vec{\omega}'_p, \vec{\omega}) \frac{\Delta\Phi_{l,p}(t_{lookup}(t))}{\frac{4}{3}\pi r^3}$$

where $\mathbf{x}$ is the current ray marching position, $\vec{\omega}$ the current viewing ray direction. knn is the number of photons that should be collected around the current position for the radiance estimate and r is the minimum radius of a sphere that contains the n-nearest photons around the current position. So theoretically, knn is the only parameter, as r could be calculated after having collected the knn-nearest photons. Practically we also have to restrict the search radius to look for the knn-nearest photons, as otherwise it would not be possible to profit from the grid structure. Please note that this is no restriction in comparison with typical CPU Photon Mapping implementations as the KD-Tree structure also employs a maximum search radius. So how does the collection of the knn-nearest photons work in detail?

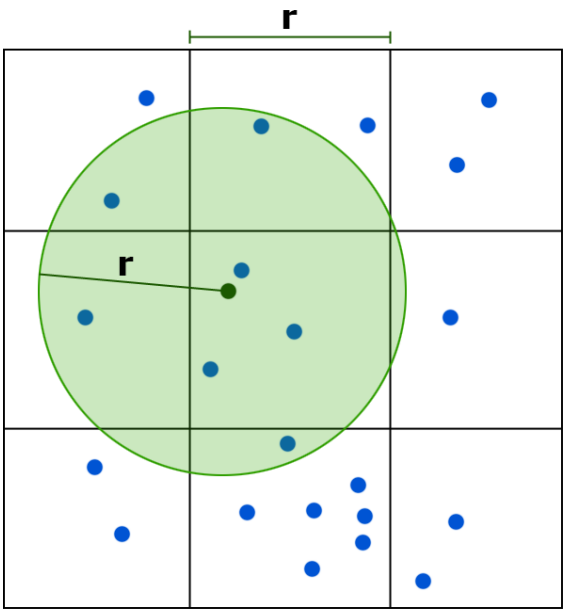


Figure 3.4: Collection of the knn-nearest photons in the photon map grid. Blue dots are the photons in the grid, the green dot is the current ray marching position and the green area represents the circular search area restricted by the maximum search radius.

The procedure used is adapted from [Fle09]. The kernel holds an array of size knn to hold the yet to be determined knn nearest photons. First, the linear cell ID of the cell enclosing the current lookup position is calculated using Equation (3.1). Using the cell ID, the hash table is accessed to retrieve the memory locations and count of all photons contained in this cell in the photon table. The knn-closest photons of this cell that are still within the search radius are inserted into the array. If there are less than knn-photons, all will be inserted as long as they are within the maximum search radius. The same procedure is repeated for all 26 cells neighboring the current cell, inserting photons into the array if their distance is closer than the furthermost photon in the array and also still within the maximum search radius. This guarantees the coverage of the search radius in all cases.

In case at least knn nearest photons can be found within the search radius, the radius used for the radius division in Equation (2.17) is the distance between the current position and the furthermost of the knn closest photons. In case of areas with only a sparse occurrence of photons, there is a problem however: The search radius might not even contain knn photons. On first thought, one might be tempted to just use the radius of the furthermost of the less-than-knn closest photons, but this introduces grave artifacts as the photon's flux influence is highly biased by the distance weighting. We use a heuristic to reduce this kind of artifact: After photon collection, the ratio of found photons to the desired number of photons is used to interpolate the division radius between the outermost found photon and the maximum search radius.

After the knn nearest photons have been collected, their contribution to the in-scattered radiance is calculated using the dynamic time dependent flux lookup described in Section 2.4.5.

In practice, the maximum search radius, the number of nearest photons collected, and the ratio between the two are the dominant factors with regard to the computation time and resulting image quality. The number of nearest photons can be imagined like the size of a blur kernel. Using more photons will produce less noise and smoother results. But in that case the maximum search radius will very likely have to be extended as there will be more areas where not enough photons can be found within the search radius, yielding the aforementioned problems. The visual quality is also of course strongly influenced by the number of photons in the photon map. Using many photons per query might also overblur the result if the photon map does not contain a huge amount of photons. This problem cannot be solved easily as even with a huge number of photons in the map, the spatial distributions of the photons in the map can be highly non-homogeneous.

Another problem that is unique to the time dependency extension is *time aliasing*. The first problem arises with the design of the light source flux timeline: Assuming a typical timeline representing a supernova explosion, being flat most of the time and containing one single short burst of illumination, there can be a lot of photons with high spatial proximity, that still have a low temporal proximity. As photons are collected for spatial proximity, the knn closest photons might not all be of importance with respect to the flux timeline, as their time since the emission might vary strongly because their path through the volume was very different because of multiple scattering or the bent photon curves. The other issue has to be seen in conjunction with the search radius: The greater the distance between the current position and the photon, the higher the temporal error of the flux lookup, as the compensation for the time lookup does take into account the travel distance of a straight line between the two, but this does not account for the different directions of the view path and the photon path.

3.3.7 Skybox

To enhance the visuals of the renderings, the framework uses a skybox to provide a background for the rendered volumes. The skybox can be seen as a texture mapped box that is infinitely far away from the viewer. This is realized in CUDA by uploading the background images as a *Cubemap Texture* to CUDA. This allows for simple directional lookups using the *texCubemap* command that takes a direction as an input parameter and returns the fetched color value from the virtual six sided cube.

The cubemap is integrated into the ray marching algorithm when the ray marching is at the last step per viewing ray. The last direction when the viewing ray left the volume (or the initial direction when it did not intersect the volume) is used as the direction for the cubemap lookup. The returned RGB color value is mapped to linear space and multiplied with an exposure factor and then interpreted as the input radiance value for the last ray marching step. This way it is directly integrated into the ray marching/photon gathering stage and does not have to be treated separately by tone mapping.

The skybox is worth mentioning because it also enhances the visual feedback on the bent viewing rays caused by spatially varying refractive indices within the volume. As the viewing ray direction when leaving the volume will be deflected relative to the direction before entering the volume, the background appears distorted and increases the visibility of the effect.

3.3.8 Tone Mapping

As the result of the ray marching and photon gathering is a radiance value per color channel, we need a means to reduce the high dynamic range so that the radiance values can be mapped to a viewable image (technically speaking to byte-valued RGB channels). This process is called *Tone Mapping*. An algorithm used to reduce the dynamic range of images is called a *Tone Mapping Operator*. Operators are classified as either *global* or *local*. Global operators can be applied per pixel without taking into account neighboring pixels, thus they are usually simpler and faster. Local operators are also allowed to rely on the luminance information from neighboring pixels, thus allowing to adapt the tone mapping to local features of the image. This can yield visually better results but usually comes at the cost of higher complexity and less performance.

Popular tone mapping techniques are for example Reinhard Tone Mapping [RSS02] and Filmic Tone Mapping [Hab10]. As the focus of this framework is on the photon mapping and time dependency extension, the tone mapping operator in the CUDA kernel is a simple operator using an exposure adjustment and gamma correction:

$$channelcolor = clamp((radiance \cdot exposure)^{\frac{1}{gamma}})$$

In the implementation the gamma value is fixed to 2.2 and the exposure value is an arbitrary factor that can be set in the GUI.

3.4 Summary

The Implementation chapter presented the framework developed during this thesis, summarizing the most important aspects with regard to the implementation, referencing the fundamentals as required. The chapter also uncovered a number of issues with the technique and the reason for a strong influence of the render parameters on the quality of the results.

4 Results

This chapter presents a number of sample images rendered using the CUDA Volumetric Photon Mapping framework developed during this thesis.

The following list contains the volumes that were used to render the samples and their source in case they were not self-provided using the Maya export pipeline.

- **Smoke** - Source: `http://www.pbrt.org`. Copyright: Duc Nguyen and Ron Fedkiw.
- **Foot** - Source: `http://www.volvis.org/`. Copyright: Philips Research, Hamburg, Germany.
- **Bonsai Tree** - Source: `http://www.volvis.org/`. Copyright: S. Roettger, VIS, University of Stuttgart.
- **Supernova** - Source: `http://vis.cs.ucdavis.edu/VisFiles/pages/supernova.php`. The data set is made available by Dr. John Blondin at the North Carolina State University through US Department of Energy's SciDAC Institute for Ultrascale Visualization.
- **Stanford Bunny** - `http://www-graphics.stanford.edu/data/3Dscanrep/`. Copyright: Stanford University Computer Graphics Laboratory.
- **Stanford Dragon** - `http://www-graphics.stanford.edu/data/3Dscanrep/`. Copyright: Stanford University Computer Graphics Laboratory.
- **Stanford Lucy** - `http://www-graphics.stanford.edu/data/3Dscanrep/`. Copyright: Stanford University Computer Graphics Laboratory.

The following skyboxes/cubemaps were used to enhance the samples

- **Milky Way** - Source: `http://www.eso.org/public/images/eso0932a/`. Copyright: ESO/S. Brunier.
- **Grace Cathedral** - Source: `http://gl.ict.usc.edu/Data/HighResProbes/`. Copyright: Paul Debevec.
- **Cloudy Sky** - Source: `http://www.custommapmakers.org/skyboxes/zips/`. Copyright: Jockum Skoglund.
- **Saint Lazarus Church** - Source: `http://www.humus.name/Textures/SaintLazarusChurch.zip`. Copyright: Emil Persson.

The image annotations use the following notation:

- **Photons** - Number of photons used for the photon map.

- **KNN** - Number of photons per nearest neighbor search query.
- **g** - Henyey-Greenstein parameter.
- **n** - Maximum refractive index (at the highest density). If not noted, no spatially varying refractive indices are used.

The first images depict simple samples that do not yet make use of the extensions implemented. The subsequent samples show the nonlinear and time dependency extensions in isolation and finally the last section showcases the original use case of the framework: Renderings of galactic light echoes that make use of all the extensions in conjunction. All sample images were rendered on a Core 2 Quad Q6600, 4 GB of main memory and an NVIDIA GeForce GTX 560 graphics card. The default render resolution is 512x512, if not noted otherwise. The images below are sometimes cropped from the original resolution to better fit the page layout if needed.

4.1 No extensions

This section shows a few rendered samples using the CUDA Volumetric Photon Mapping Framework without any of the extensions. Please note that there are no isolated samples for the *RGB* and *Materials* extensions, they are implicitly used after this section.

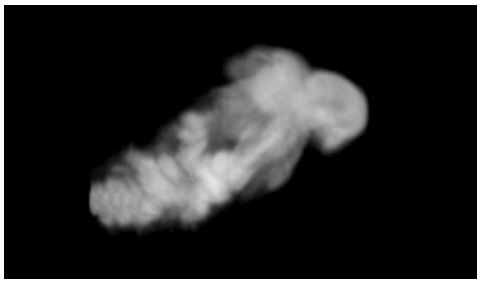


Figure 4.1: Volumetric smoke. Photons: 10 Mio, KNN: 200, g: 0.1, Render time: 4.63s

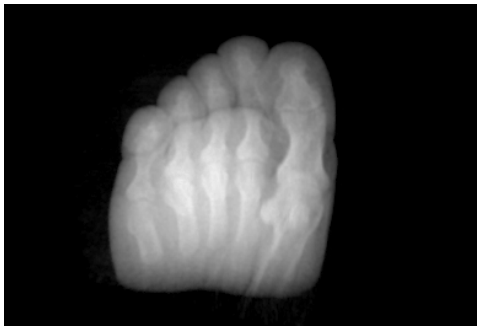
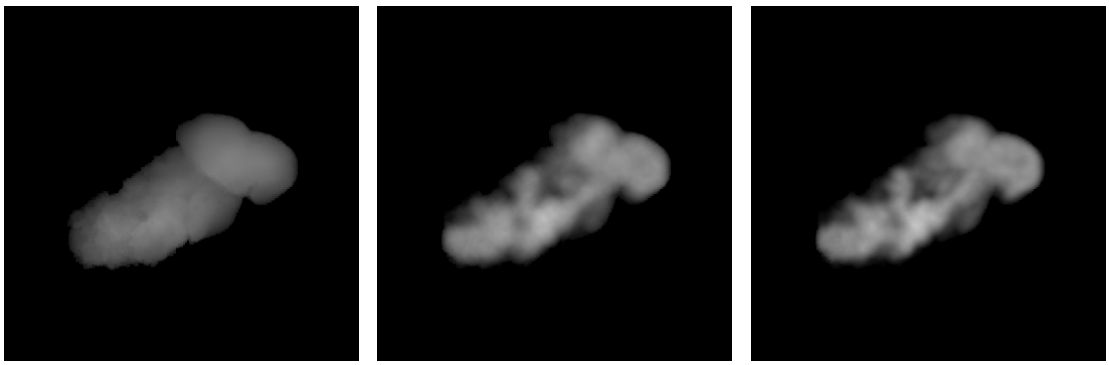
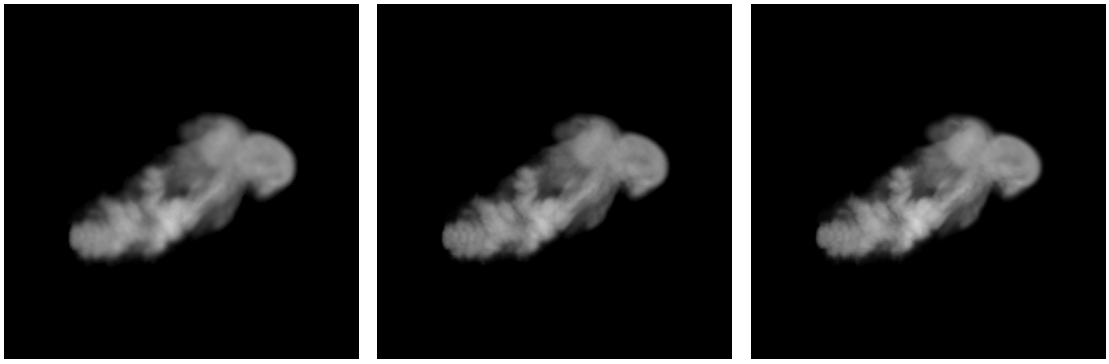


Figure 4.2: Foot CT. Photons: 10 Mio, KNN: 200, g: 0.1, Render time: 20.79s

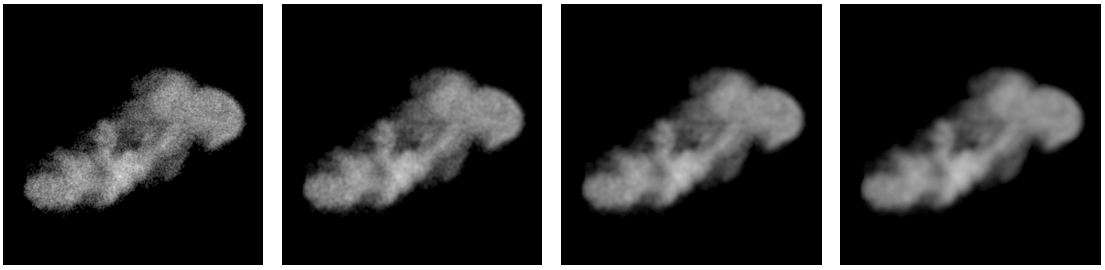


(a) 1K P. 0.9s **(b)** 10K P. 2.7s **(c)** 100K P. 4.8s



(d) 1 Mio P. 24.7s **(e)** 10 Mio P. 46.5s **(f)** Reference

Figure 4.3: Comparison of the same volume rendered with an increasing amount of photons. KNN 200, g 0.1. The figures' labels mention the amount of photons and the render time. The figures show that about 10 million photons are required to get about the same amount of detail as the single scattering ray marching reference solution yields.



(a) 200 Cells, 0.01s **(b)** 100 Cells, 0.2s **(c)** 67 Cells, 0.8s **(d)** 50 Cells, 2.5s

Figure 4.4: Comparison of different search radii, showing the reduction of noise but also exponentially increasing render times. KNN 200, 100K Photons, g 0.1. The labels mention the number of grid cells along the horizontal direction (since a cell's sidelength matches the maximum search radius).

4.2 Nonlinear Extension

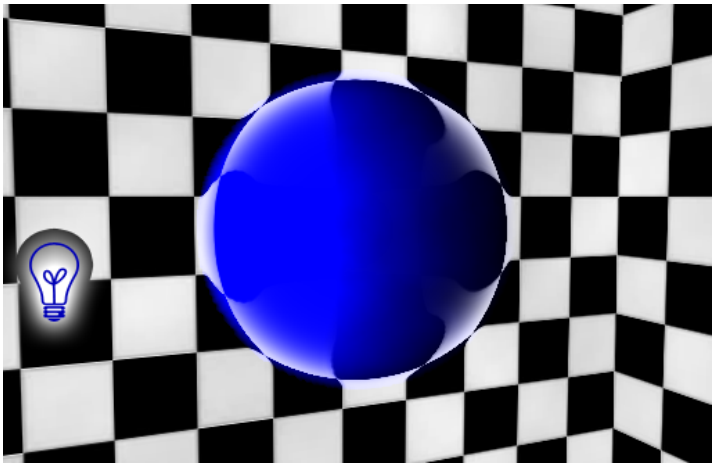
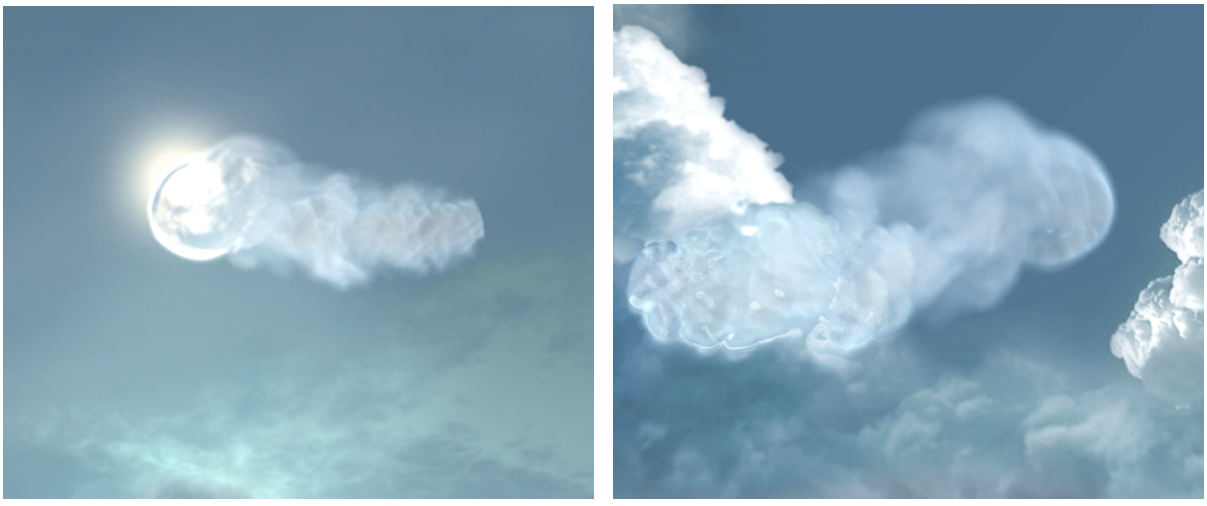


Figure 4.5: Sphere sample. In the outer shell of the sphere, the density linearly falls off from the maximum value at the core to zero at the outer bounds. This causes spatially varying refractive indices that will bend the light. The checkerboard skybox background helps to visualize the bent light curves. Photons: 1 Mio, KNN: 200, g: 0.5, n:1.02, Render time: 93.6s.

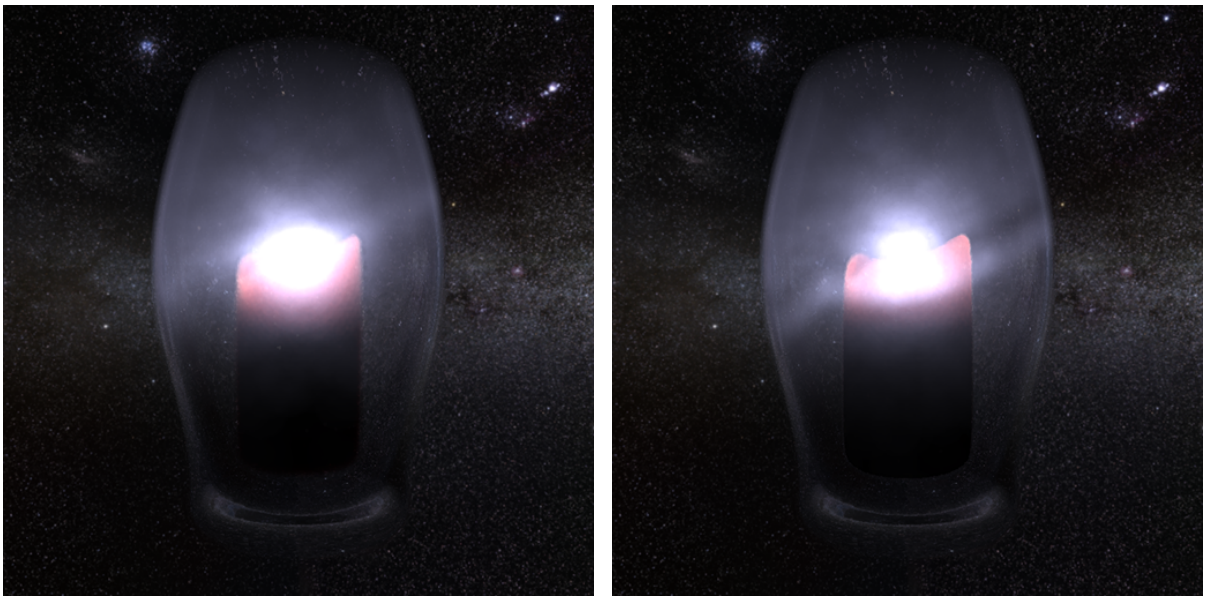


(a) Render time: 56.2s. **(b)** Render time: 71.6s.

Figure 4.6: Refractive cloud. This is the smoke dataset rendered with a sky background. Photons: 10 Mio, KNN: 200, g: 0.1, n:1.03.

(a) Render time: 361s. **(b)** Render time: 1551s.

Figure 4.7: Rendering of a vase composed of three different materials with different refraction indices and absorption coefficients. The image depicts caustics within the volume. 1 Mio photons.

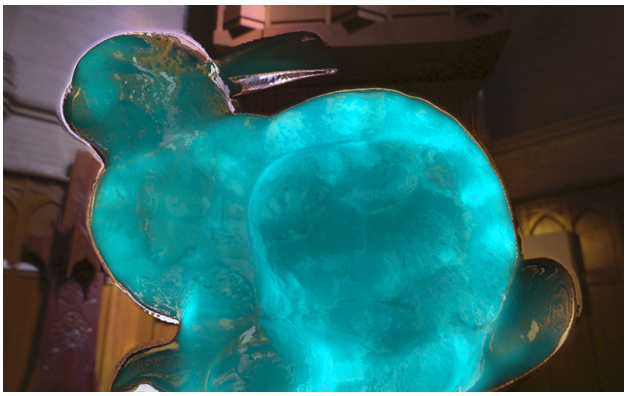


(a) Candle n=1. Render time: 364s. **(b)** Candle n=1.458. Render time: 418s.

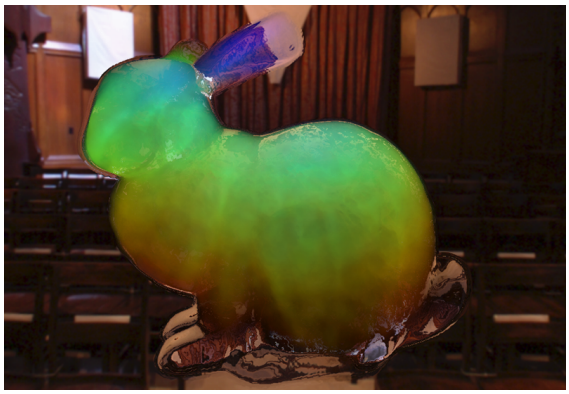
Figure 4.8: Candle confined in a vase. 5 Mio Photons, KNN 200, Render resolution: 768x768. The comparison shows the importance of the nonlinear photon emission. While the glassy vase effect is solely the result of the bent viewing rays, the caustic streaks on the vase are a result of the nonlinear photon emission when also using spatially varying refractive indices for the candle.

(a) Render time: 386s.

(b) Render time: 213s.



(c) Render time: 394s.



(d) Render time: 305s.

Figure 4.9: Rendering of a volume of the Stanford Bunny. The density within the object has been modified using Perlin Noise, which leads to volumetric caustics within the volume because of the resulting spatially varying refractive indices. 1 Mio Photons, KNN 200, Render resolution 1024x1024.

4.3 Time Dependency Extension

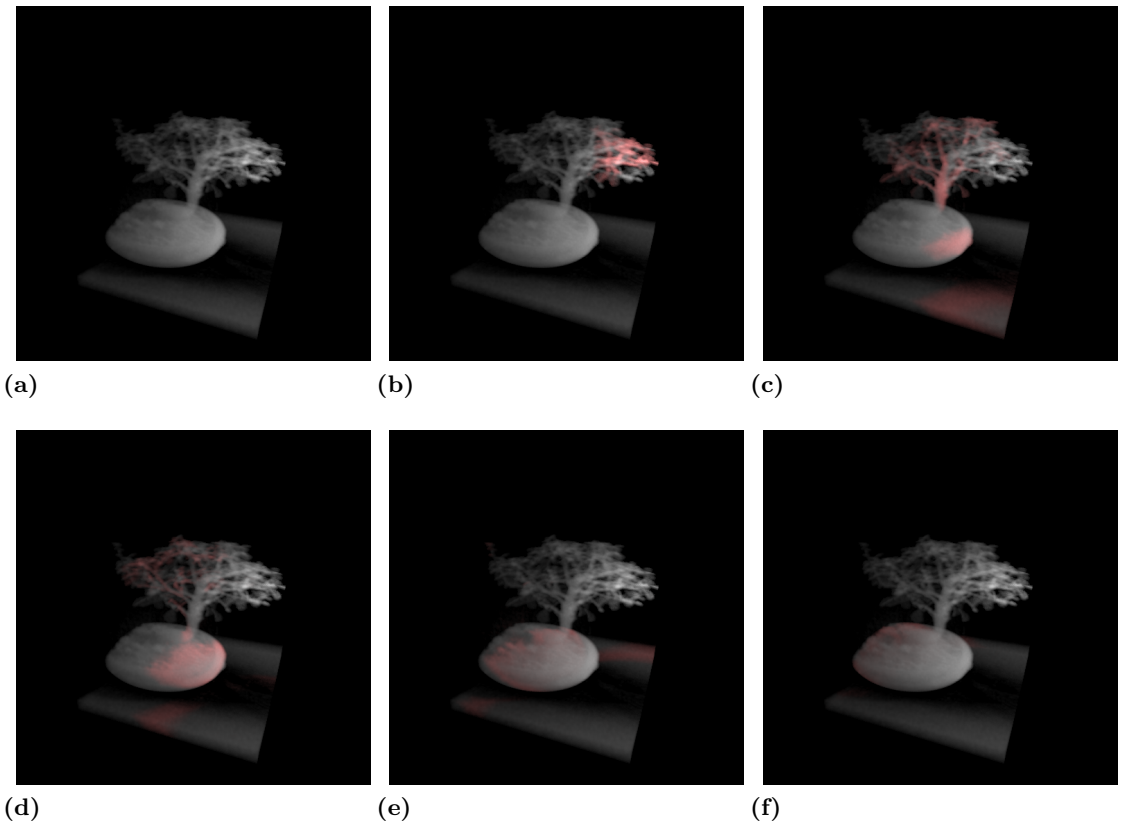


Figure 4.10: Light propagation at different points in time along a tree volume. The sidelength of the volume is two meters. Photons: 5 Mio, KNN: 200, g: 0.1.

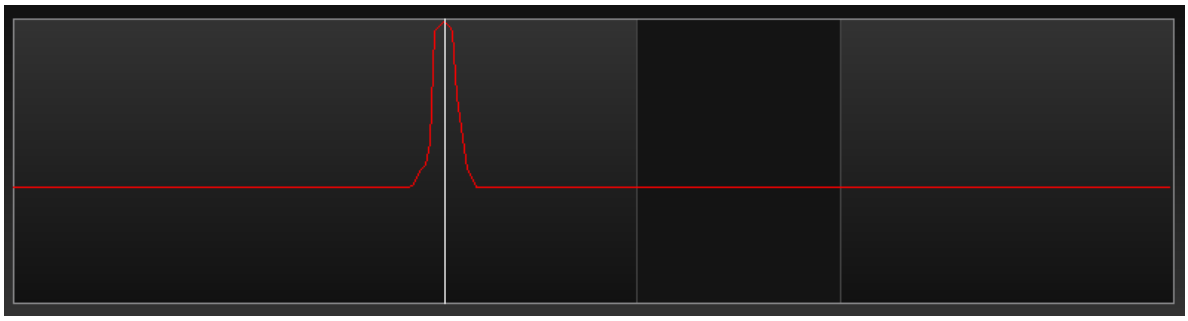
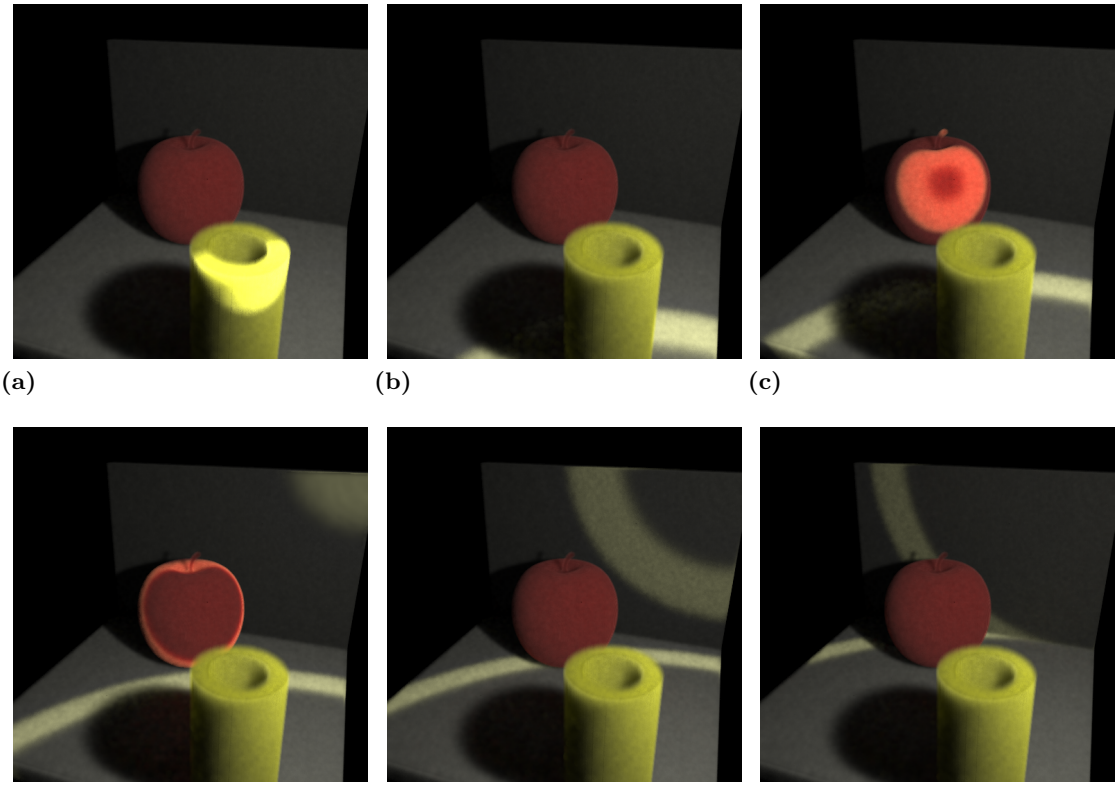


Figure 4.11: Light flux timeline of the tree example. The light emits a short red burst with only a duration of about 1e-9s. The black area shows the time range from which the images were rendered. The sequence encompasses around 6e-9s.



(a) (b) (c)

(d) (e) (f)

Figure 4.12: This sample has been built analog to the MIT Media Lab recording of light propagation. Photons: 10 Mio, KNN: 500, Render time: 404s per frame

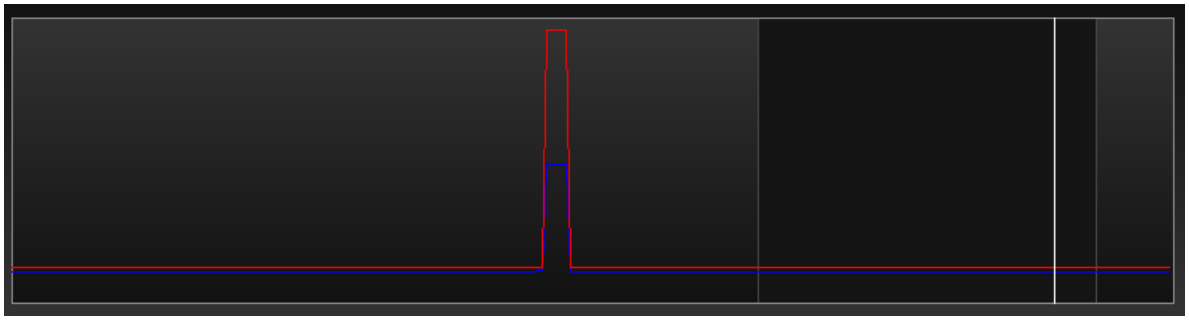
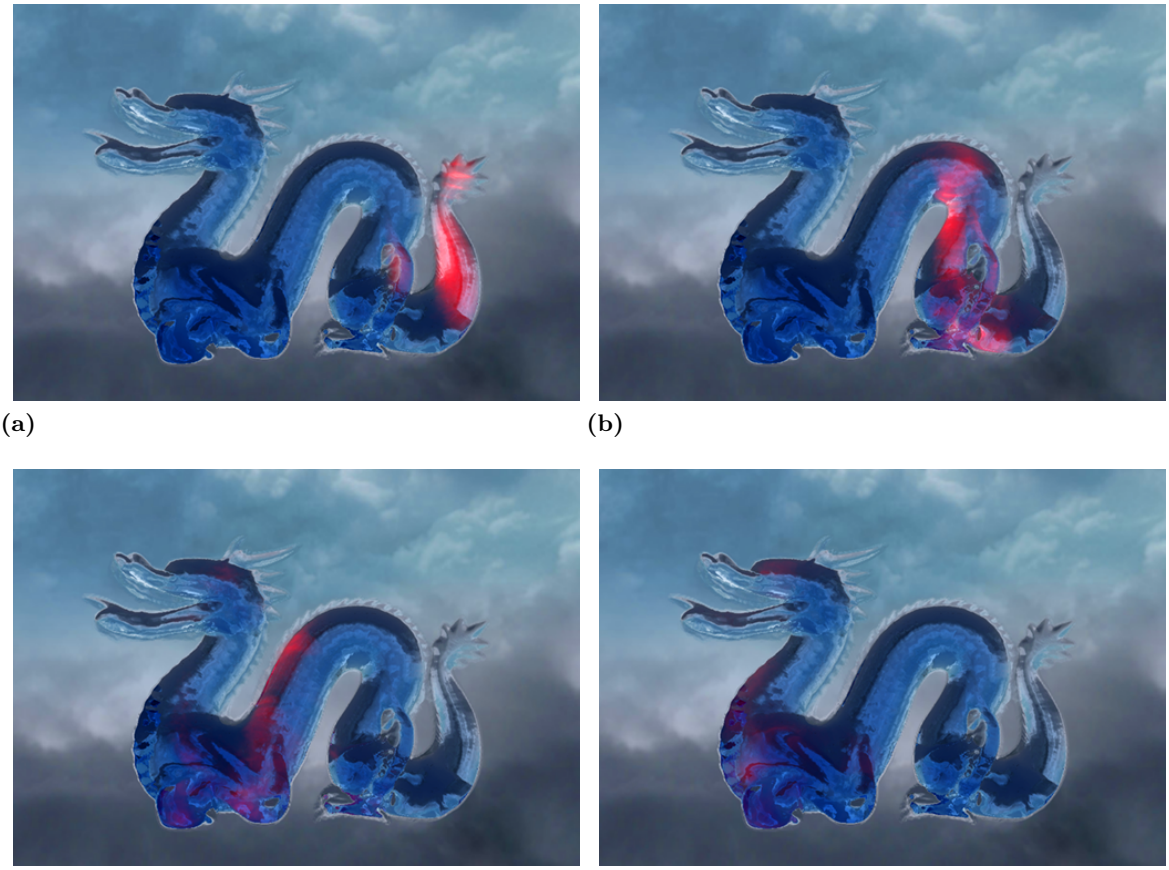


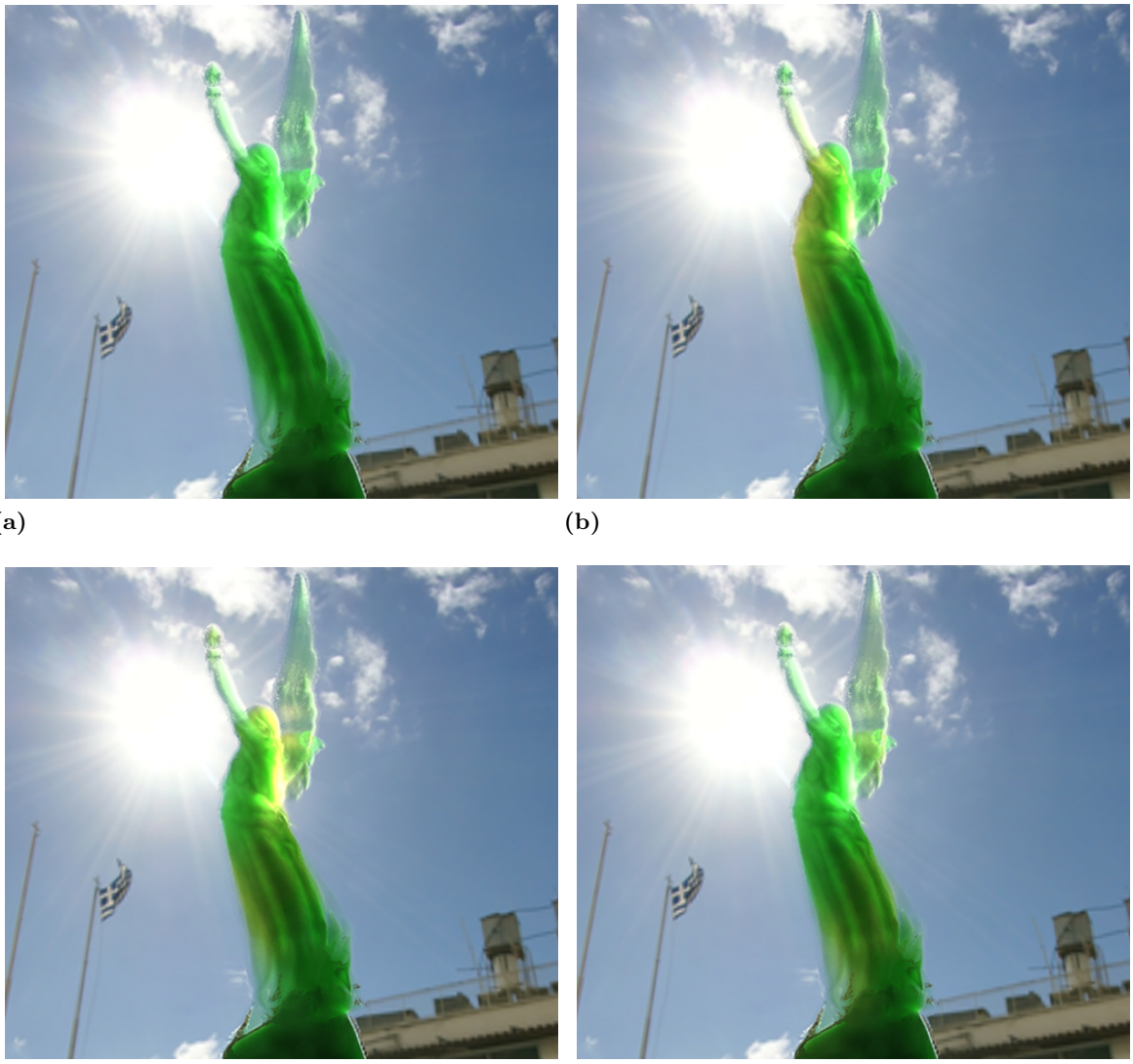
Figure 4.13: Light flux timeline of the apple example. The black area shows the time range for the complete passage of the light burst. The white line marks the point at which image (f) was rendered.



(a) **(b)**

(c) **(d)**

Figure 4.14: Rendering of a volume of the Stanford Dragon. A short red light burst propagates through the dragon. 5 Mio Photons, KNN 200, n=1.4, Render resolution 1024x1024. Render time: 21s per Frame

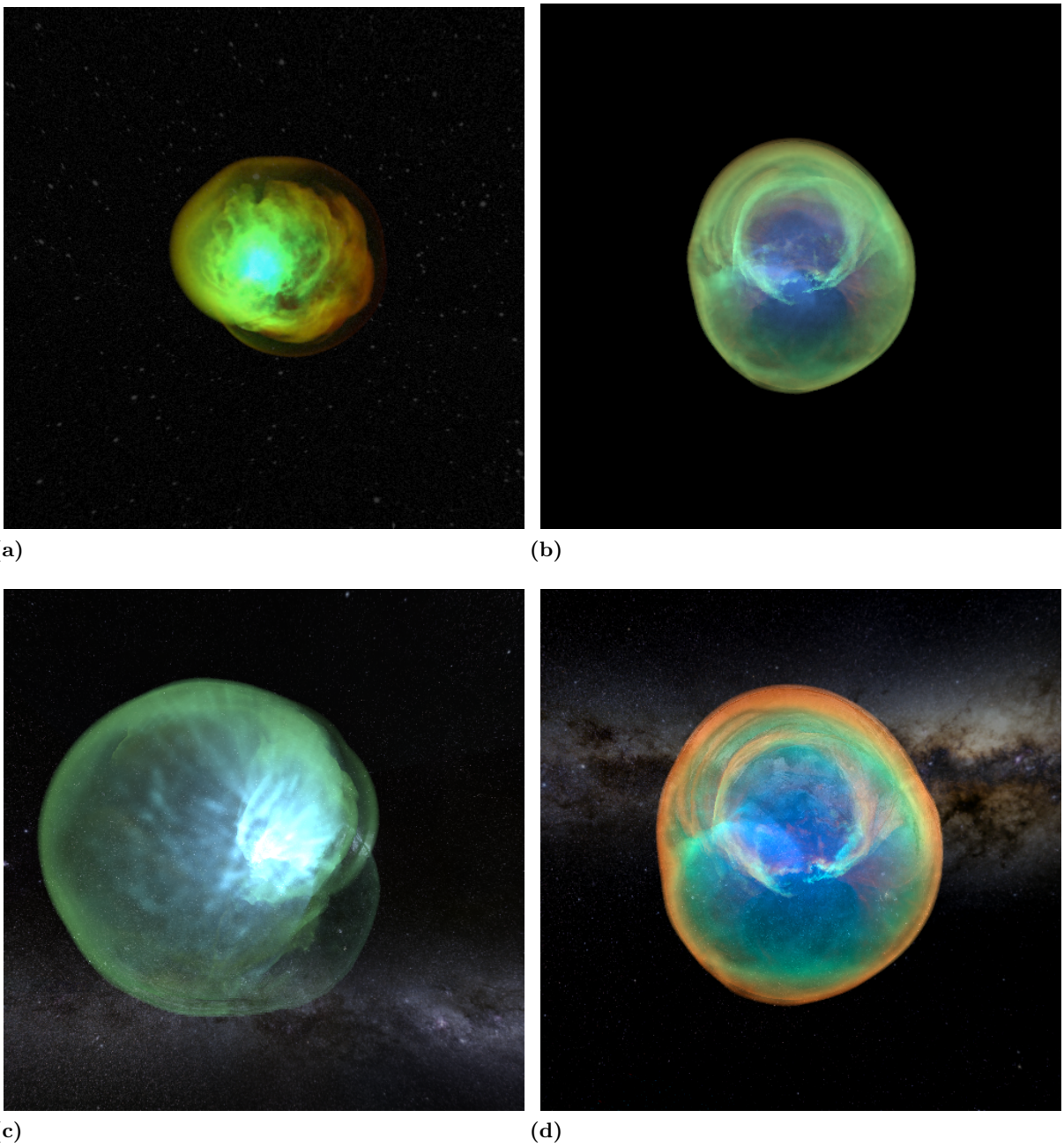


(a) **(b)**

(c) **(d)**

Figure 4.15: Rendering of a volume of Stanford Lucy. A subtle yellowish light burst propagates towards the viewer. 100K Photons, KNN 200, n=1.2, Render resolution 1024x1024. Render time: 30s per Frame

4.4 Supernova Sample



(a) **(b)**

(c) **(d)**

Figure 4.16: Different renderings of the supernova volume using different RGB flux curves for the light source, different camera positions and different material parameters.

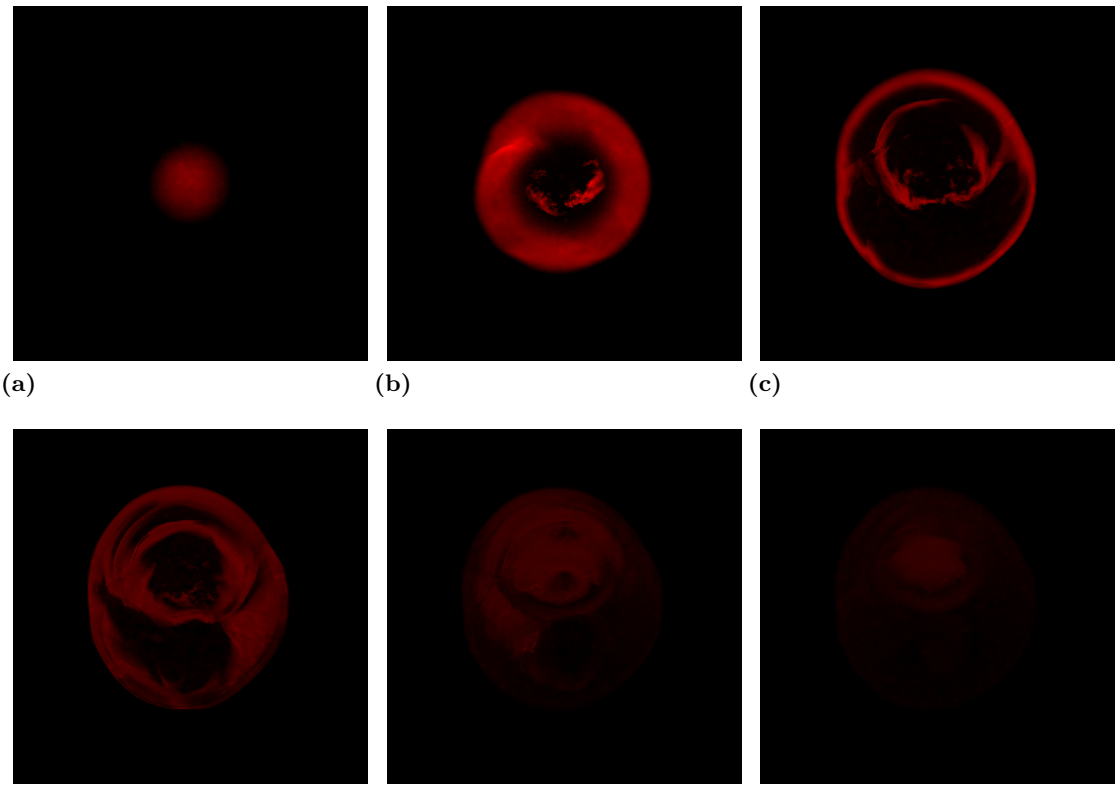


Figure 4.17: A large scale light echo of a single red flash from a light source at the center of the supernova volume. Photons: 10 Mio, KNN: 200, g: 0.6., n:1.01, Render time: 48s per Frame

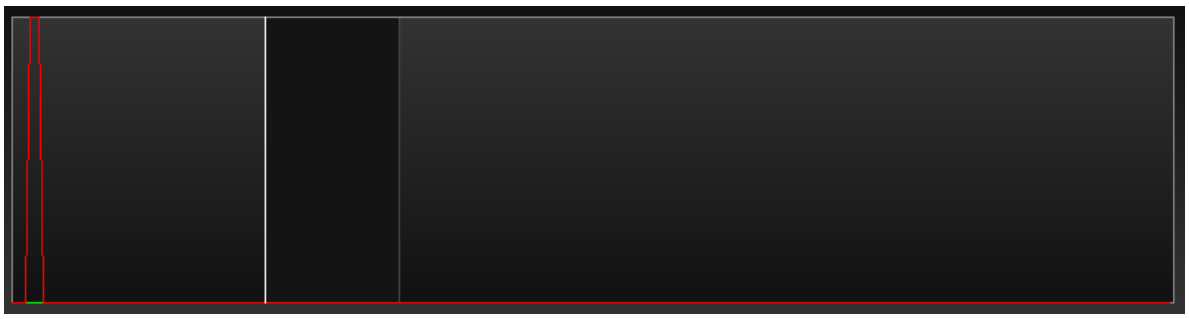
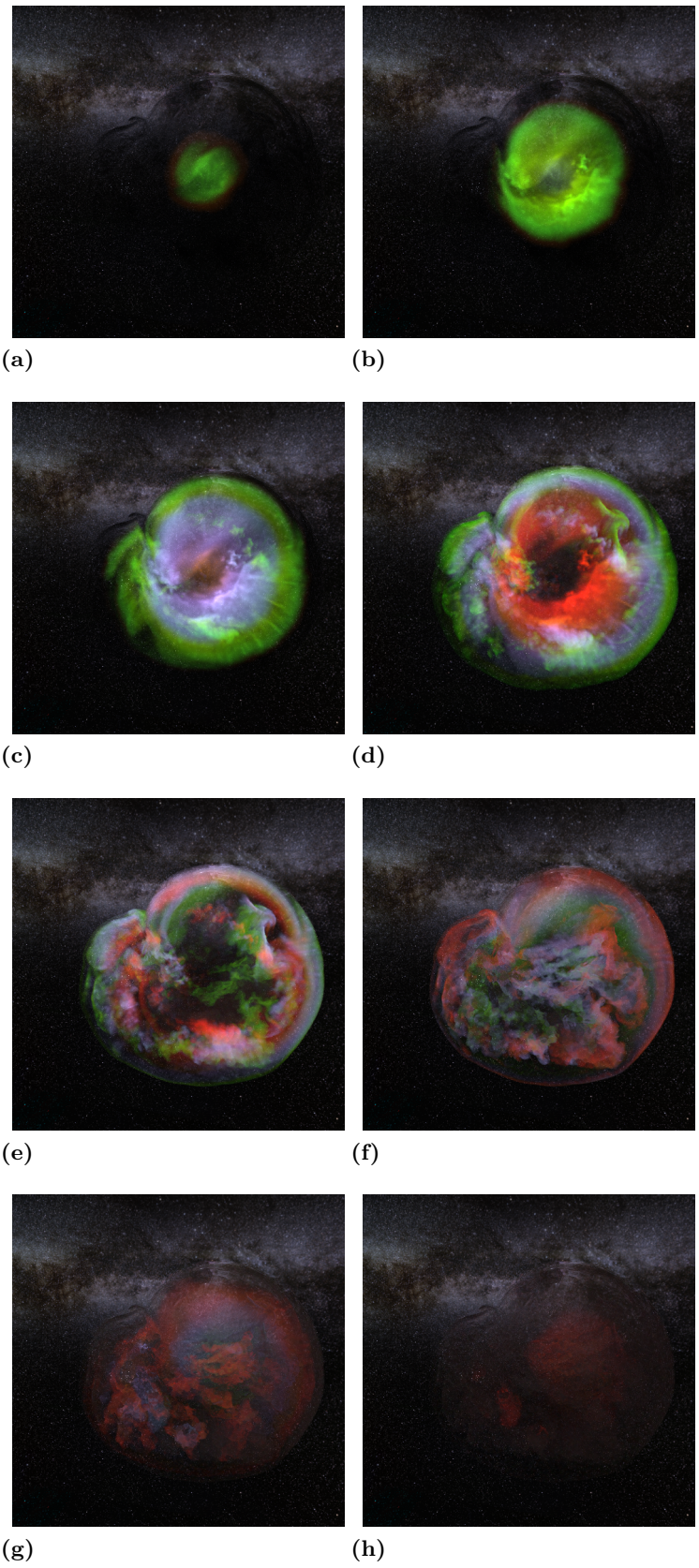


Figure 4.18: Light flux timeline of the single red flash example. The black area shows the time range of the sequence.



(a) **(b)**

(c) **(d)**

(e) **(f)**

(g) **(h)**

Figure 4.19: A large scale light echo with a varying RGB light flux curve over time. Photons: 10 Mio, KNN: 200, g: 0.6., n:1.01, Render time: 56s per frame

5 Discussion and Future Work

This thesis presented the theory and implementation of a framework to simulate time-dependent light propagation in participating media. The intended use is the visualization of intergalactic light echoes, but the framework is also capable of capturing light propagation on small scale objects.

The framework is based on volumetric photon mapping and extended to incorporate multiple color channels, multiple materials per volume and nonlinear effects caused by spatially varying refractive indices. Also, the photon gathering stage is ported to CUDA to accelerate rendering. While these extensions are implementations of techniques already known and published, we introduced the time dependency extension as a novel extension to photon mapping that respects the finite speed of light propagation. As this is combined with the aforementioned extensions, these also require some minor modifications to make all extensions work in combination.

For the remainder of this chapter, we take a look at possible future work and improvements to our framework.

5.1 Multiple Lights

The framework always uses a single point light source. This is sufficient for our primary use case to visualize light echoes where the point light is supposed to represent a star and it also proved sufficient for our samples to demonstrate the extensions to photon mapping. An extension to multiple lights could be implemented in a simple manner: The photon data structure is extended with an attribute to hold a light source ID. The RGB light flux timeline is made a component of the light source, so that each light may have its own timeline.

For the tracing stage, we still use a single photon map and execute a tracing pass for each light. So each light tracing pass stores its photons in the photon map, using the ID of the current light source.

The photon gathering stage mostly stays the same as before. The most important change is to read the light ID of all photons collected and use it to select the proper light flux timeline that is associated with this photon.

5.2 Dynamic Volume

So far we always assumed the volume to be static over time, so the density values are not time-dependent. In this section we shortly outline a possible extension as future work that would enable the usage and integration of volumes that change over time.

Suppose we have multiple time steps of a volume. We would like to use this dynamic volume in conjunction with our framework and all the extensions implemented. How could this be achieved?

One change that applies to both the photon tracing stage, as well as the photon gathering stage is the way lookups of the material/volume parameters at a position within the volume have to be done. Each lookup is done at a specific point in time and space and each volume is associated with a point in time for which it is valid. To get the volume parameters at a specific time, we need to determine the two volumes where the lookup time is in-between the time associated with the volumes. We read the volume parameters from both volumes at the specified position and use linear interpolation between the two values.

Dynamic Volume - Photon Tracing

The photon tracing has to undergo a significant modification. As of now, photons only stored their time of flight and could be emitted independent of the absolute time at which the volume was supposed to be rendered because of the static nature of the volume. With the volume now changing over time, we have to store absolute time values for photons, because the trajectory of a photon during emission will be influenced by the changing volume. As a consequence to this, photons have to be emitted over time using time steps Δt. The photons emitted may be stored within a single large photon map over all emission time steps. This significantly increases the storage requirements as we need to compute enough time steps to provide enough temporal data to cover a full eye-ray traversal of the volume during the photon gathering stage.

So the photon tracing could be modified in the following way:

1. Start photon emission from the light source at time $t = t_0$.
2. For each photon (until the desired number of photons for this time step is reached).
   a) Set the tracking time to t.
   b) During tracing, keep track of the absolute time, starting at time t. To read all necessary parameters at the current position, use the adapted lookup method as described above.
   c) Store the photon at scattering events as usual. The only change is that the absolute time value is stored instead of the relative time of flight.
3. Advance the emission starting time to $t = t + \Delta t$.
4. If more time steps should be computed, goto Step 2.

The number of time steps computed is the factor of how much the storage requirements of the photon map increases. Currently a photon map containing 10 million photons requires about 400-500 MB. Assuming we would need to calculate 1000 time steps, the photon map would occupy around 0.5 TB, making it impossible to store the complete map in RAM or VRAM. This would imply that some sort of streaming mechanism would be required to access the photon map.

Dynamic Volume - Photon Gathering

The adaptation of the photon gathering stage itself is not too extensive: The lookup of the volume parameters also would have to use the adapted lookup method as described before, employing absolute time values.

The most interesting part here would be the changed gathering of photons at the current sampling position. Photons would have to be collected with respect to both spatial as well as temporal proximity. We would need to restrict the sampling to a finite time interval, analog to the sphere that is used to restrict the spatial search. This will make the variance also depend on the temporal resolution of the photon data. Here future work could include research on data structures that allow efficient retrieval with respect to both space and time.

The final necessary change is to also associate the RGB flux timeline with absolute time values as the photons now store an absolute time value. This means that we keep the ability to dynamically change the light flux timeline without having to recompute the photon map.

In general this extension would then also require to carefully set the initial time values for photon emission, the flux timeline and photon gathering/ray marching, as otherwise the ray marching might e.g. traverse the volume at a time where no proper data (photons with a matching or at least temporal close timestamp) is available.

5.3 Various Improvements

Finally the following list presents a number of additional possible improvements and future work

- Parallelization/CUDA port of the Photon Tracing stage. A way to achieve this is described in [Fle09].
- Enhanced performance for the Photon Gathering stage in the CUDA kernel by improving the sorting of the KNN photons.
- Enhanced quality of nonlinear curve steps with higher order methods like e.g. Runge-Kutta.
- Enhanced ray marching via e.g. Adaptive Ray marching to enhance performance and better resolving of small details.

- Research on material parameters. Currently absorption/scattering coefficients and refractive indices are simply user-controlled values or simple heuristics without a proper physical background. In order to enhance the validity of the visualization of light echoes, this should be adapted to values that are backed by research on the actual material compositions of e.g. intergalactic nebulae.
- Research on how the presented extensions could be included in the new Photon Beam [JNJ11] and Query Beam [JZJ08] approaches.
- An additional extension to spectral rendering that also allows to capture the effect of dispersion.
- Finding a solution to the shadow ray problem of nonlinear photon mapping as this currently forces the inclusion of directly scattered photons in the photon map.
- Research on including relativistic effects in the framework.

Some of these issues could be addressed straightforward with techniques that are already available and were just not realized because of the time-constraints of this thesis. Others pose problems that are more general and also generally unsolved at this point and thus are truly to be categorized as future work.

Bibliography

[BHL+03] H. E. Bond, A. Henden, Z. G. Levay, N. Panagia, W. B. Sparks, S. Starrfield, R. M. Wagner, R. L. M. Corradi, U. Munari. An Energetic Stellar Outburst Accompanied by Circumstellar Light Echoes. *Nature*, 422:405–408, 2003. (Cited on page 10)

[BS09] A. Bracher, B.-M. Sinnhuber. An Introduction to Remote Sensing - Chapter Atmospheric Radiative Transfer, 2009. (Cited on page 18)

[Bun09] L. Buntemeyer. 3D Radiative Transfer in Radial Velocity Fields, 2009. (Cited on page 15)

[Cha60] S. Chandrasekhar. *Radiative Transfer*. Dover Pubn Inc, 1960. (Cited on pages 12 and 13)

[Cor] N. Corporation. CUDA C Programming Guide. (Cited on pages 40 and 41)

[Fle09] M. Fleisz. Photon Mapping on the GPU. 2009. (Cited on pages 13, 48, 49, 54 and 72)

[Goo95] R. M. Goody. *Atmospheric Radiation: Theoretical Basis*. 1995. (Cited on page 19)

[Gre08] S. Green. CUDA Particles. 2008. (Cited on pages 13 and 48)

[Gut05] D. Gutierrez. Non-Linear Volume Photon Mapping. *Eurographics Digital Library*, 2005. (Cited on pages 13 and 27)

[Hab10] J. Hable. Uncharted 2: HDR Lighting. Game Developers Conference, 2010. URL `http://filmicgames.com/Downloads/GDC_2010/Uncharted2-Hdr-Lighting.pptx`. (Cited on page 56)

[HG41] L. Henyey, J. Greenstein. Diffuse Radiation in the Galaxy. *Astrophysics Journal*, 93:70 – 83, 1941. (Cited on page 19)

[hub] Hubble's Universe Unfiltered: Show 12: A Flash of Brilliance. URL `http://hubblesite.org/explore_astronomy/hubbles_universe_unfiltered/12`. (Cited on pages 8 and 10)

[IK] M. Ikits, J. Kniss. GPU Gems - Chapter 39. Volume Rendering Techniques. URL `http://http.developer.nvidia.com/GPUGems/gpugems_ch39.html`. (Cited on page 29)

[JC98] H. W. Jensen, P. H. Christensen. Efficient Simulation of Light Transport in Scenes with Participating Media using Photon Maps. 1998. (Cited on pages 12, 25, 38 and 48)

[Jen96] H. W. Jensen. Global Illumination using Photon Maps. 1996. (Cited on page 21)

[Jen01] H. W. Jensen. *Realistic Image Synthesis Using Photon Mapping*. AK Peters, 2001. (Cited on pages 13, 19, 21, 26, 27 and 38)

[JNJ11] W. Jarosz, D. Nowrouzezahrai, H. W. Jensen. A Comprehensive Theory of Volumetric Radiance Estimation using Photon Points and Beams. *ACM Transactions on Graphics*, 2011. (Cited on pages 12, 38, 40 and 73)

[JZJ08] W. Jarosz, M. Zwicker, H. W. Jensen. The Beam Radiance Estimate for Volumetric Photon Mapping. *Proceedings of Eurographics 2008*, 2008. (Cited on pages 12, 38, 39 and 73)

[KTU+re] O. Krause, M. Tanaka, T. Usuda, T. Hattori, M. Goto, S. Birkmann, K. Nomoto. Tycho Brahes 1572 Supernova as a Standard Type Ia Explosion Revealed from its Light Echo Spectrum. *2008*, 456(7222):617 – 619, Nature. (Cited on page 8)

[Mar] J. Markoff. Speed of Light Lingers in Face of New Camera. The New York Times. URL `http://www.nytimes.com/2011/12/13/science/speed-of-light-lingers-in-face-of-mit-media-lab-camera.html`. (Cited on page 31)

[mit] Mitsuba Renderer. URL `http://www.mitsuba-renderer.org/`. (Cited on pages 13 and 27)

[NAS] NASA. Universe 101 - The Life and Death of Stars. URL `http://map.gsfc.nasa.gov/universe/rel_stars.html`. (Cited on page 7)

[Pat05] F. Patat. Reflections on Reflexions - I. Light echoes in Type Ia supernovae. *Monthly Notices of the Royal Astronomical Society*, 357(4):1161–1177, 2005. (Cited on pages 9, 13, 19, 20 and 23)

[RSS02] E. Reinhard, M. Stark, P. Shirley. Photographic Tone Reproduction for Digital Images. *ACM Transactions on Graphics*, 2002. (Cited on page 56)

[SM] G. Stemp-Morlock. Light Echo Helps Solve Supernova Mystery. National Geographic News. URL `http://news.nationalgeographic.com/news/2008/05/080529-supernova-mystery.html`. (Cited on page 9)

[Sun08] X. Sun. Interactive Relighting of Dynamic Refractive Objects. 2008. (Cited on page 28)

[Wei01] D. Weiskopf. *Visualization of Four-Dimensional Spacetimes*. Ph.D. thesis, Eberhard-Karls-Universitaet Tuebingen, 2001. (Cited on page 38)

[Wil04] D. R. Williams. NASA Sun Fact Sheet, 2004. URL `http://nssdc.gsfc.nasa.gov/planetary/factsheet/sunfact.html`. (Cited on page 36)

All links were last followed on August 12, 2012.

Erklärung

Hiermit versichere ich, diese Arbeit selbständig verfasst und nur die angegebenen Quellen benutzt zu haben.

(Christoph Bergmann)