Czech Technical University in Prague
Faculty of Electrical Engineering

Masterer Thesis

Progressive Computation of Global Illumination

Prague, 2012 **Author: Bc. Zdeněk Glazer**

Aknowledgements

I would like to thank to Hanka Pokorná, who has always supported me. And also to Ing. Jaroslav Sloup for supervision of this thesis.

Declaration

I hereby declare that I have completed this thesis independently and that I have listed all the literature and publications used.

Prague, 10. 11. 2012 ________________ ______________________

Signature

Abstract

Global illumination always played a key role in realistic computer generated image synthesis. Many algorithms have been developed to compute realistic images in reasonable time, though rendering volumetric phenomenas such as smoke including volumetric caustics and complex glossy reflection and refraction paths is still consuming task. Recently some new light caching approaches and progressive rendering techniques were discovered. This thesis aims to test their accuracy and feasibility on various scenes containing heterogenous participating media.

Abstrakt

Globální osvětlení vždy hrálo klíčovou roli při realistické syntéze obrazu. Existuje mnoho algoritmů které dokáží realistický obraz vypočítat v rozumném čase, ale výpočet globálního osvětlení u scén obsahujících opticky aktivní prostředí, jako je například kouř je stále časově velice náročný. V nedávné době se objevily nové postupy předvýpočtu světelných cest ve scéně a nové progresivní metody výpočtu obrazu.V této diplomové práci chceme prověřit jejich přesnost a rychlost na různých scénách obsahujicích nehomogenní opticky aktivní prostředí.

Vložit zadani prace

Contents

List of Figures

List of Tables

Chapter 1

Introduction

People have always been curious about new forms of visualization and illusion of reality. In the last decade computer generated imagery has reached a state, when majority of the public can't distinguish real photographs from entirely synthesized images. Thanks to this advancements we can enjoy otherwise impossible film shoots, imaginary worlds, visualizations of non existing buildings and much more. All this become possible, because ways how to represent objects and simulate light transport using computer algorithms have been found. The other important factor is ever growing computation power of these devices, which enables us visualize highly detailed datasets with complex materials and challenging lighting scenarios.

Even though many things are possible to simulate and visualize efficiently these days volumetric phenomenons such as clouds, smoke and other optically active medias are still real challenge even for high-end computers. This is all caused by the fact that the data we try to visualized are volumetric and that light passing through can be absorbed, scattered or even emitted in different wavelength (fig 1.1). This means that we can no longer assume that visibility of an object can be determined simply by determining if it is hidden behind other object or not.

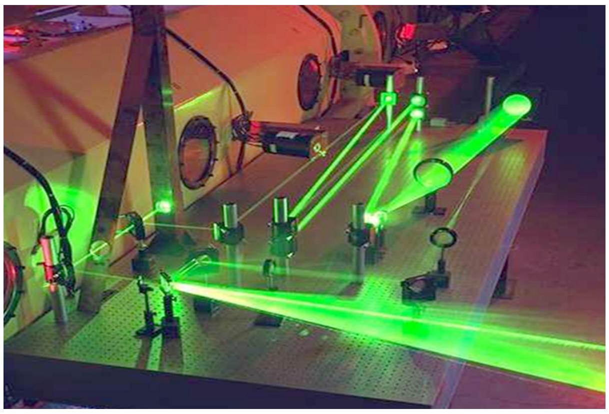

Figure 1.3: Laser beam being refracted and reflected multiple times. The beam is visible thanks to interaction with tiny particles floating in the air, which diffract part of the light energy to the camera chip.Source [NAS].

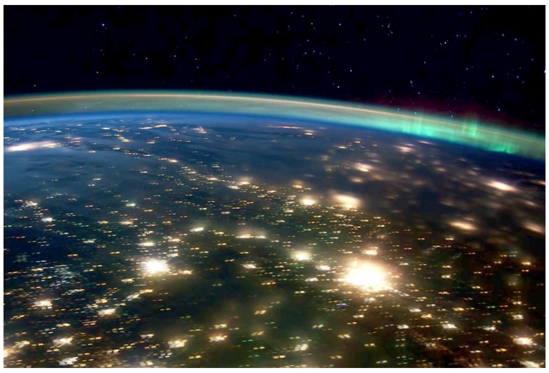

Figure 1.1: Aurora borealis one of the most fascinating volumetric phenomenons on earth. This photo also exhibits atmospheric light scattering and city glow at night. Source [SAL].

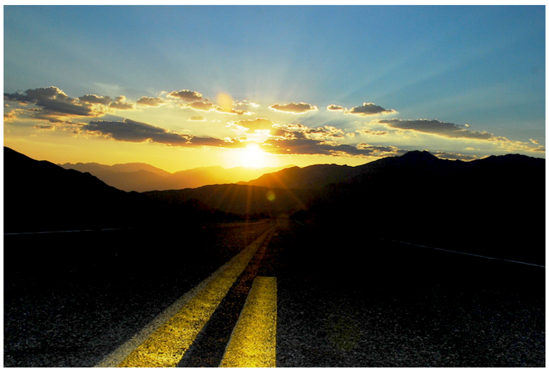

Figure 1.2: Crepuscular rays also known as good rays are nice example of volumetric shadowing effect, which can be quite believably approximated using only single light scattering. Autor [PN].

In terms of feasible render times only direct illumination component can be computed. This approach can give us satisfying results such as those on the figure (fig 1.2). Unfortunately many cool looking effects such as volumetric caustics, which can be seen on the figure (fig 1.3) are impossible to get using this approach.

Figure 1.4: Image after first progressive iteration on the left. After ten iterations in the middle and on the right final quality image after twenty iterations.

This thesis focuses on methods, which can render anything with physical accuracy and without restrictions on the rendered scene. The other very important bonus of these methods is their progressive nature. In the first passes the corse illumination of the scene is computed and it's progressively refined (fig 1.4). This for example means, that artist defining lighting mood of the visual effect shots can iterate faster. They have got something to show to the director, which results in quick turnarounds and substantial cost and time savings.

Our aim is to implement these progressive Monte Carlo methods in the presence of heterogenous, optically active media and test them on scenes with varying complexity.

1.1 Thesis structure

This Thesis consists of eight main chapters. In the first chapter we have introduced you to the problem. In the second chapter the nifty theoretical details behind realistic image synthesis are exposed and fundamental terms such as BRDF, phase function and rendering equation defined and explained.

The third chapter is an overview of current strategies and algorithms used to visualize the volumetric phenomenons and their pros and cons listed. The fourth chapter leads us to the core of this thesis, the progressive methods of computation the global illumination in the presence of volumetrically active phenomenons.

In the fifth chapter we choose some of the methods mentioned in preceding chapter and further analyze them and propose a way how to and why to use them. In the sixth chapter we present our solution and describe it's internal structure. In the seventh chapter our results and test are shown. And finally eight chapter concludes the information gained and results obtained during this thesis creation.

Chapter 2

Fundamentals of realistic image synthesis

In this chapter we briefly revise fundamental theory behind realistic computer generated imagery. Our aim is to synthesize (render) image of a virtual scene. To do this properly, with physical accuracy in mind, we have to find a suitable physical model of light.

If we would like to be absolutely correct, we would have to use quantum optics model. This would lead us to tracing individual photons and interaction with matter on an atomic level, which is completely computationally unfeasible. Nevertheless when it comes to computer graphics even a simplified ray optics model can handle majority of the phenomena we are interested in. This model assumes that light travels in straight lines, at infinite speed and the only things that might happen to it are emission, absorption, reflection, and transmission. Using this model we will be unable to simulate phenomena such as diffraction (for example chromatic aberration), polarization and other similar effects depending on light wavelengths and polarity.

When the suitable light model is chosen, we should define some numerical quantities we can compute with. In the next section we will start with radiometry quantities and inspect problem of light interaction with surfaces. This gives us enough fundaments to formulate rendering equation. We continue with light interaction with participating media, which leads us to extended rendering equation. Algorithms mentioned in the next chapters are based on this theory and they present various methods solving this complex equation.

2.1 Radiometry

Radiometry is set of techniques for measuring different kinds of electromagnetic radiation such as light. We could have used a photometry, which on the other hand measures the response of a human eye to visible light. This basically corresponds to measurements retrieved using radiometry and weighted with an eye response function. For the purposes of this thesis we will use radiometric quantities, because corresponding photometric quantities can be easily derived.

2.1.1 Radiometric quantities

$$\phi = \frac{dQ}{dt} \quad [W = J.s^{-1}] \tag{2.1}$$

Radiant flux , also known as radiant power, denoted as ϕ. This quantity corresponds to the light power mentioned on the light bulbs in Watts. Where Q [J] denotes radiant energy, which describes how much energy (amount of photons times their energy) is presented in a given area at a point in time. This leads us to flux definition as amount of radiant energy going through a location over time , alternatively speaking how fast is the change of the amount of the photons in a specified location.

$$E(x) = \frac{d\phi(x)}{dA} \quad [W.m^{-2}] \tag{2.2}$$

Irradiance , also known as flux density, denoted as *E(x).* The irradiance at a point x on the surface *S,* can be expressed as an amount of incident flux to differential area at a given time.

$$L(x,\omega) = \frac{d\phi(x)}{cos\theta dAd\omega} \quad [W.m^{-2}.sr^{-1}] \tag{2.3}$$

Radiance is probably the most important quantity. Denoted as $L(x,\omega)$, where x is a point of interest and ω is a differential angle coming in a given direction, also known as solid angle.Radiance expresses how much flux is coming from a differential direction $d\omega$ onto a differential surface dA. The $cos\theta$ factor compensates the shortening amount of irradiance on the surface with growing θ[1], while the level of illumination is maintained. This is the quantity, which has to be gathered for every pixel in the rendered image.

2.1.2 Relationships between quantities

In order to render our image we have to compute radiance coming from the scene through every pixel of the virtual camera sensor. From the table 2.1 you can see that radiance units are Watts

[1]The meaning of θ is $cos(\theta) = (\vec{n}.\vec{\omega})$, where $\vec{n}$ is surface normal at a point of interest (x) and $\vec{\omega}$ is a direction vector we want to compute radiance for.

per square meter per steradian[2]. According to the units of the remaining radiometry quantities we should be able to derive them using integration.

Radiant energy	Q	[J]
Radiant flux	ϕ	[W]
Irradiance	E	$[W.m^{-2}]$
Radiance	L	$[W.m^{-2}.sr^{-1}]$

Table 2.1: Table summary of the radiometric quantities.

If we integrate incoming radiance over a hemisphere denoted as Ω in a given point x we get equation 2.4. Notice a $-\omega$ in the equation, this is caused by the fact that Ω usually refers to hemisphere consisting of outgoing solid angles. But when computing irradiance we are interested in the incoming radiance.

$$E(x) = \int_{\Omega} L(x, -\vec{\omega}) * cos(\theta) d\vec{\omega} \tag{2.4}$$

To get flux from radiance, we will have to perform two integration steps. We will have to integrate radiance over a hemisphere in every differential surface, which belongs to the surface we are interested in. This is precisely written in the equation 2.5.

$$\phi = \int_{A} \int_{\Omega} L(x, \vec{\omega}) * cos(\theta) d\vec{\omega} dA \tag{2.5}$$

[2]Steradian is unit of a solid angle.

Figure 2.1: This is an example of the same material (aluminum metal) with different surface texture. From left to right brushed, polished and surface after abrasive blasting. Below each image you can see corresponding schematic BRDF function

2.2 Surface interaction

Using ray optics model, light can interact with object in only four ways. It can be created by the object, which is referred as an emission. On the other way it can be partially or fully absorbed, reflected or refracted in the same time.

As was mentioned in the beginning of this chapter, it's almost impossible to simulate these light interactions in large scenes using light interaction on a molecular level. Currently the most used scene representation is probably an object boundary representation[3]. Unfortunately the microscopic structure of the object plays a huge role in the light interaction as can be seen on the fig 2.1 and would again result in an immense complexity if represented using triangles. To cope with this problem we usually model the corse object shapes using the boundary representation and we call the microscopic behavior "the material behavior". To model the material behavior we use the BRDF function.

2.2.1 BRDF

The *bidirectional reflectance distribution function* describes what is the probability that an incoming light from a given incoming direction will reflect to a given outcoming direction from

[3]Consisting of a finite set of triangles, quadrangles and other n-gons.

the given surface. As stated before we can define the BRDF function as:

$$f(\omega_i, x, \omega_o) = \frac{dL_o(x, \omega_o)}{dE(x, \omega_i)} = \frac{L(x, \omega_o)}{L_i(x, \omega_i) * cos(\theta_i) d\omega_i} \tag{2.6}$$

To formulate the BRDF function more precisely we should define it's properties:

1. **Function domain:**
   For a given surface point x the BRDF is four dimensional function. Two dimensions for the incoming direction and two for the outgoing direction.

2. **Value range:**
   BRDF can gain any positive value. For example using constant BRDF functions we can model diffuse materials.

3. **Reciprocity:**
   The BRDF value stays the same if we interchange incoming and outgoing direction. This is a the very fundamental property many global illumination algorithms rely on. Thanks to this behavior we can gather or shoot light energy using the same BRDF functions and all light interactions remain the same.

4. **Linearity:**
   It's a linear function, thus it is not dependent on irradiance from other directions. To get total reflected radiance in the given direction ω_o and given point we get an equation 2.7.

$$L_o(\vec{\omega}_o) = \int_{\Omega} L_i(\vec{\omega}_i) * f_r(\vec{\omega}_i, \vec{\omega}_O) * cos(\theta) d\vec{\omega} \tag{2.7}$$

5. **Energy conservation:**
   Total reflected energy to all directions is smaller or equal to the total irradiance in any given point. This ensures that an object can't reflect more light than it receives.

2.2.2 Rendering equation

We have successfully defined the quantities we want to compute and models we want to use for the light surface interaction. What we want to achieve is to distribute the light energy in the scene and compute the energy fraction which reaches our image pixels.

To describe the energetic equilibrial state in the scene[4] we use rendering equation 2.8. As you can see it closely resembles the equation 2.7. The main difference between these two equations is the fact that the unknown radiance [5] is on both sides of the equation and that instead of the local radiance $L(x, \omega_o)$ we want to trace a ray in a direction ω_i to the scene and compute the radiance in closest ray surface intersection $L(r(x, \omega_i), -\omega_i)$.

$$L(x, \omega_o) = \int_\Omega L(r(x, \omega_i), -\omega_i) * f(\omega_i, x, \omega_o) * cos(\theta_i) d\omega_i \tag{2.8}$$

$$\overbrace{L(x, \omega_o)}^{\text{outgoing}} = \underbrace{L_e(x, \omega_o)}_{\text{emited}} + \underbrace{\int_\Omega L(r(x, \omega_i), -\omega_i) * f(\omega_i, x, \omega_o) * cos(\theta_i) d\omega_i}_{\text{reflected}} \tag{2.9}$$

The equation 2.8 can deal with light reflection, refraction and absorption. To consider the light emitting objects in our scene we have to introduce an emitted radiance [6] into the equation. Which result in eq 2.9. To get an actual image of our scene $L(x, \omega_o)$ (outgoing radiance) has to be computed for every image pixel.

[4] This refers to the light distribution in the scene.

[5] Both $L(r(x, \omega_i), -\omega_i)$ and $L(x, \omega_o)$ are unknown.

[6] $L_e(x, \omega_o)$

2.3 Volume interaction

By now we have assumed that radiance is conserved along its ray path between surfaces. It's time to consider participating media interaction in our computation. To do this we should define some properties, which can be used to model and describe different types of volumetric phonomenons.

κ_a $[m^-1]$ - Absorption coefficient is equal to the probability that a photon gets absorbed in volumetric media, per unit of distance along it's flight direction. In other words what is the probability that the photon will hit molecule and gets absorbed[7]. This leads to radiance reduction.

κ_s $[m^-1]$ - Scattering coefficient is equal to probability that the photon will by scattered[8], per unit of distance along it's flight direction. This again leads to radiance reduction in the original photon flow direction, but also to radiance propagation into other directions.

κ_t $[m^-1]$ - Extinction coefficient is a summation of the two preceding properties. It equals to probability that the photon will collide with the particles in the media[9], which gives us following relationship: $\kappa_t = \kappa_a + \kappa_s$.

We distinguish four different types of interaction events, which have got a lot in common with previously defined coefficients:

1. **Absorption:**
   $\boldsymbol{\kappa_a}$ gives us the probability of absorption. To evaluate an effect to the incident radiance we simply subtract proportional part of it as follows:
   $$dL_a(x, \omega_0) = -\kappa_a(x) * L(x, \omega_0)dx \tag{2.10}$$

2. **Volume emission:**
   To simulate any light emitting volumetric phenomenons like fire or aurora, we have to add some energy to the ray going through this media. The emission contribution to the radiance equals to:
   $$dL(x, \omega_0) = \kappa_a(x) * L_e(x, \omega_0)dx \tag{2.11}$$

3. **Out-scattering:**
   The radiance going through the media can get scatter, thus the original energy flow might

[7]Absorbed equals to transformation of photon energy to other kinds of energy, such as heat radiation.

[8]Scattering is process when radiation is forced to deviate from it's straight path to other directions.

[9]The probability that it gets scattered or absorbed.

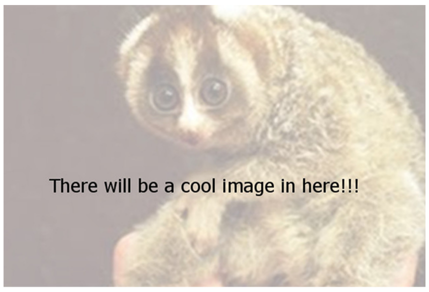

Figure 2.2: This fig can be helpful to understand how each of the absorption, emission, in and out-scattering events affects the radiance.

participate into many new directions. The probability that this will happen is described by $\boldsymbol{\kappa_s}$, so the radiance loss can be describeded as:

$$dL(x, \omega_0) = -\kappa_s(x) * L(x, \omega_0)dx \tag{2.12}$$

4. **In-scattering:**
   To sum the total radiance in a given point in the media we have to cope with the consequences of out-scattering from the surrounding media. To do it we have to integrate incoming radiance to a point in the space from all possible directions and weight it with the probability that the scattering from these directions will happen $\boldsymbol{\kappa_s}$. This leads us to eq. 2.13. The $p(x, \omega_0, \omega_i)$ is phase function, which will be described further in the next section 2.3.1.

$$dL_i(x, \omega_0) = \int_\Omega L(x, \omega_i)p(x, \omega_0, \omega_i)d\omega_i \tag{2.13}$$

For better understanding take a look on the picture fig2.2. Since all the above equations considered only a differential volume we will have to extend it in section 2.4.

2.3.1 Phase functions

To model a light interaction on a molecular level, we use phase functions[10], which can be used to model different volumetric media types. It describes the probability of scattering light from incoming direction ω_i to the outgoing direction ω_o.

Phase function has got these useful properties:

1. **Function domain:**
   Is the same as in the case of the BRDF function. For the given point in the space it is four dimensional function. Two dimensions for incoming and to for the outgoing direction.

2. **Value range:**
   Greater than zero and it's very useful to normalize phase function values so that:
   $$\int_{\Omega 4\pi} p(\omega_i, x, \omega_i) = 1 \qquad (2.14)$$
   This way we can be sure that the energy will be conserved.

3. **Reciprocity:**
   The phase function value stays the same if we interchange incoming and outgoing direction.

4. **Conservation:**
   The incoming radiance is always greater or equal to the sum of the distributed energy into all directions.

The phase function depends heavily on the ratio between the light wavelength and particle size of the volumetric media. Since we are interested mainly in the visible light spectra, we can narrow the dependency on the particle size. Fig 2.3 shows different phase function approximations, depending on the growing particle size. As you can see it is extremely difficult to describe this complex behavior globally in one analytical model function. Thus multiple models are used in practice, each one is suitable only for some media types.

2.3.1.1 Heney-Greenstein phase function

This phase function can nicely approximate both forward and backward light scattering behaviors, modifying only one variable g. The function is defined as:

$$HG(\theta) = \frac{1 - g^2}{4\pi(1 + g^2 - 2gcos(\theta))^{frac32}} \qquad (2.15)$$

[10] Analogically to the BRDF functions, which model light surface interaction on microscopical level and simulate different surface materials.

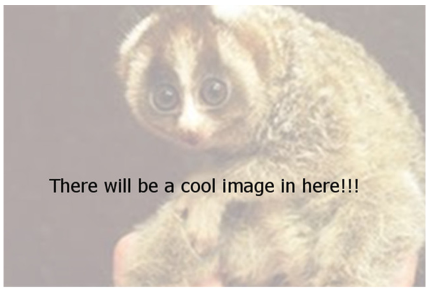

Figure 2.3: Phase function dependency with fixed light wavelength on growing particle sizes. More details about the functions are in the sections 2.3.1.2,2.3.1.2 ,2.3.1.3 .

Thanks to it's flexibility and low computational complexity it's widely used to simulate dust, smoke even water light behavior. The fig 2.4 demonstrates how is the ratio between forward and backward scattering dependent on the g value, while fig 2.5 shows a 3D shape of forward scattering phase function.

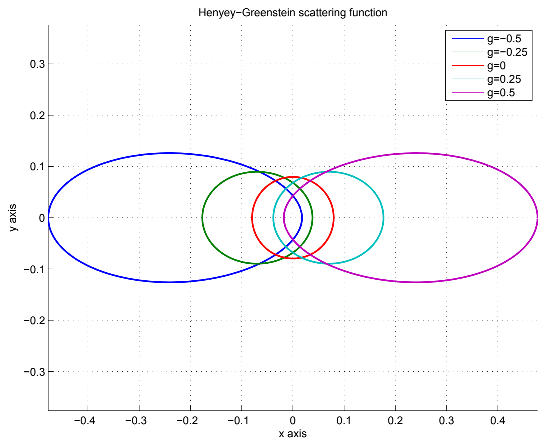

Figure 2.4: Plot of Heney-Greenstein phase function with different g coefficients. For g<0 the function represents backscattering media, for g=0 isometric scattering and for g>0 growing forward scattering tendencies.

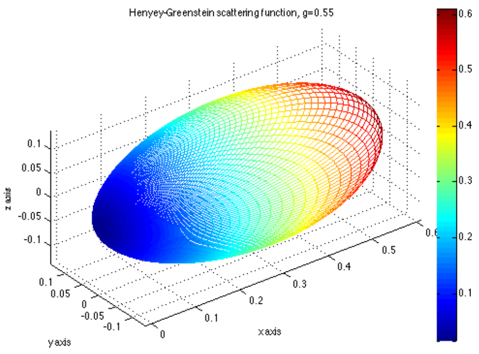

Figure 2.5: 3D plot of Heney-Greenstein phase function with scattering coefficient set to 0.55, which results in moderate forward scattering.

2.3.1.2 Rayleigh Scattering

Rayleigh scattering models interaction between electromagnetic radiation and particles, which should be much smaller than the radiation wavelength. Since it can model different wavelengths, polarizations and particle sizes, it is a widely used in scientific computation and computer graphics. It's an ideal model for light atmosphere interaction, because it describes the scattering coefficient dependency on light wavelength as eq. 2.16. The effects it has on the atmosphere appearance can be seen on fig 2.6.

$$\kappa_s = \frac{1}{\lambda^4} \tag{2.16}$$

The original equation 2.17 is really general, I_0 is incoming light intensity, σ is light wavelength, n is refractive index of particles, R is distance to particle and d is the diameter of the particle.

$$I = I_0 \frac{1 + cos^2(\theta))}{2R^2} * \left(\frac{2\pi}{\lambda}\right)^4 * \left(\frac{n^2-1}{n^2+2}\right)^2 * \left(\frac{d}{2}\right)^6 \tag{2.17}$$

Since many of these coefficients, can be considered constant in the media we can put them into scattering coefficient σ_s. Resulting polar plot is in the fig 2.7 and the phase function is following:

$$R(\theta) = \frac{3}{16\pi}(1 + cos^2(\theta)); \quad \sigma_s = \frac{2\pi^5 * d^6}{\lambda^4 * 3} * \left(\frac{n^2-1}{n^2+2}\right)^2 \tag{2.18}$$

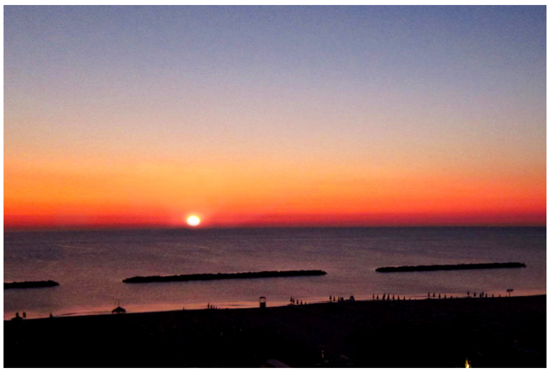

Figure 2.6: Because shorter wavelengths are scattered more, the red color light spectra dominates on the horizon, where the light path through the atmosphere is longest.

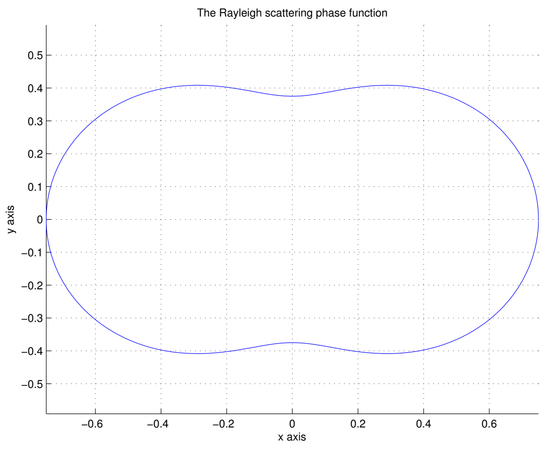

Figure 2.7: 2D polar plot of function Rayleigh scattering function.

2.3.1.3 MIE scattering

Probably the most elaborate model of all mentioned. It's based on the theory of Lorenz–Mie solution to Maxwell's equations. It is used to simulate light interaction with particles, which size is approximately equal to the light wavelength. These properties make it an ideal candidate for light water droplet interaction simulations such as cloud, mist, rain lighting. Since it is wavelength dependent it can be used to simulate effects such as rainbows or coronas.

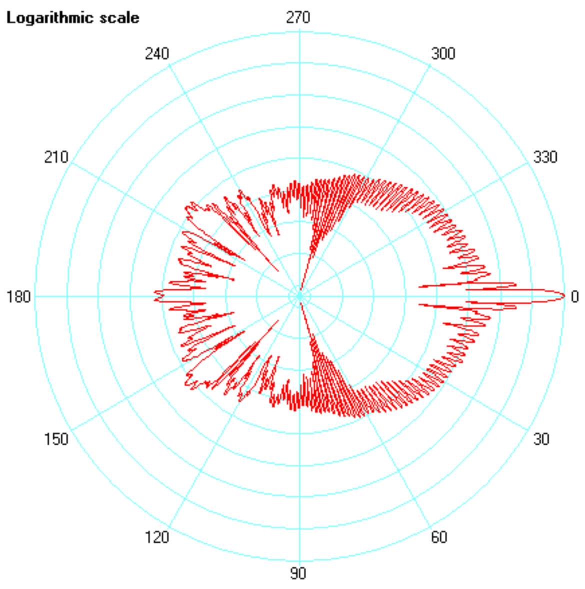

Figure 2.8: Polar diagram of scattering of red light (wavelength 0.65 μm, perpendicular polarization) from a water droplet of r = 10 μ. Source [Lav].

Unfortunately it's quite computation costly [11]. It's usually evaluated once for the whole θ domain[12] in the pre computation phase and stored to lookup table. Sample polar plot of the function can be seen on the fig 2.8. Notice the major forward peak, which carries almost 50% of the incoming energy[13], this property makes it even more challenging for an efficient importance sampling techniques, because the side scattering might be easily omitted, leading to loss of rainbow like effects.

[11] Approximation method's do exist and will be presented in the chapter 6.

[12] We usually assume symmetric behavior along an incoming light direction, which results in $< 0; \pi >$ interval for θ.

[13] Even though it might not be absolutely evident, from the logarithmic scale polar plot.

2.4 Volume and surface interaction

In order to properly simulate light transport in the scenes containing both surfaces and volumetrically active media, few adjustments to the rendering equation have to be made.

First of all, the participating media can add radiance to the pixel, because of both emission and in-scattering. On the other hand the surface radiance contribution can be attenuated, because of the media absorption and out-scattering. To accommodate these effects into rendering equation, we have to extend the volume interaction theory from differential volume slice (sec. 2.3) to brother volume section.

2.4.1 Radiance losses

The loss of radiance caused by out-scattering and absorption is very often called transmittance. The equation for differential volume looks like this:

$$dL(x,\omega) = -\kappa_a(x) * L(x,\omega)dx - \kappa_s(x) * L(x,\omega)dx = -\kappa_t(x) * L(x,\omega)dx \tag{2.19}$$

To integrate overall loss over a longer segments of the media we have to compute its **optical thickness** as fallows:

$$\int_{x_0}^{x} \kappa_t(x_v)dv \tag{2.20}$$

where the x_0 and x are the end points of the media section we are interested in. According to the *Beer–Lambert law* [Bou29] we can now evaluate the total transmittance between two points in the media as follows:

$$\tau(x_0, x) = e^{-\int_{x_0}^{x} \kappa_t(v)dv}; \tag{2.21}$$

Now we can express the overall radiance loss as:

$$L(x,\omega) = L(x_0,\omega)\tau(x_0,x) \tag{2.22}$$

2.4.2 Radiance gains

In the terms of radiance gains the equations are quite straight forward. The only thing we have to do is to weight the gain with appropriate probability and attenuate it with corresponding transmittance. This way we get following equation, which express the radiance gain over a given volumetric path segment:

$$L(x,\omega) = \int_{x_0}^{x} \underbrace{\tau(x_0,x_v)\kappa_a(v)L_e(x,\omega)}_{\text{attenuated emitted radiance}} + \underbrace{\tau(x_0,x_v)\kappa_s(v)L_i(x_v,\omega)}_{\text{attenuated in-scattered radiance}} \; dv \tag{2.23}$$

2.4.3 Extended rendering equation

Now we have all the necessary elements to put together an extended rendering equation for rendering both surfaces and volumetrically active media. Eq. 2.24 is known as radiative transfer equation [Cha60]. By extending this equation from differential volume to the fully integral form we get eq. 2.25.

$$dL(x,\omega) = \overbrace{\underbrace{-\kappa_a L(x,\omega)}_{absorption}\;\underbrace{-\kappa_s L(x,\omega)}_{out-scattering}}^{\text{dissipation loss}}\overbrace{\underbrace{+\kappa_a L_e(x,\omega)}_{emission}\underbrace{+\kappa_s L_i(x,\omega)}_{in-scattering}}^{\text{gain}} dx \tag{2.24}$$

$$L(x,\omega) = \overbrace{\underbrace{L(x_0,\omega)}_{\text{surface radiance}} \; * \; \underbrace{\tau(x_0,x)}_{\text{transmittance}}}^{\text{attenuated surface radiance}} + \overbrace{\int_{x_0}^{x} \underbrace{\tau(x_0,x_v)\kappa_a(v)L_e(x,\omega)}_{\text{attenuated emitted radiance}} + \underbrace{\tau(x_0,x_v)\kappa_s(v)L_i(x_v,\omega)}_{\text{attenuated inscattered radiance}} \; dv}^{\text{accumulation of radiance gain}} \tag{2.25}$$

2.5 Volume representation

We have got many options for volume representation in computer graphics. Rather than representing individual particles[14] we usually use larger volumes of space using quantization or some kind of bounding volume representation.

- **Particle representation** This representation is widely used in games and movies, for phenomenons like sand, dust. The main advantage is that this representation fits nicely into rasterization pipeline, where each point can be rendered out as a semitransparent point or patch.

  Though it is not very suitable representation for raytracing, because it is really hard to ray-trace something infinitely small without aliasing artifacts. For raytracing one can estimate a volume density based on the neighboring particles. To perform efficiently neighborhood queries an acceleration structure would have to be build upon these points.

- **Bounding volume representation** We can use any type of boundary representations and turn them into volumetrically active objects. The easiest way is to use parametric primitives such as boxes, spheres or cylinders. But efficient algorithms for mesh geometry exist too. This way we can represent just the boundary of the volume quite efficiently and fill it with any type of 3D texture[15] or value to alter it's optical characteristics[16] to get desired results. See fig 2.9 for examples.

- **Grid representation** To have an absolute control over each voxel[19] values we can use some kind of grid representation. For example fluid simulations or MRI[20] data come in a form of regularly spaced 3D grids. To over come a cubic space complexity of this representation more evolved structures, such as octree can be used[21].

[14]Even if we want to render large amounts of "particles" we usually might choose to render them as a point cloud where every particle has got one pixel size, this representation is not based on any physical prerequisites, though might serve well in many scenarios.

[15]Be it parametric or classical image stack texture.

[16]Characteristics like absorption, density etc. Discussed in section **??**

[19]Also known as *volumetric pixel* is a volumetric analogy of pixel in 2D. We can imagine it as an element of regular 3D grid.

[20]MRI stands for *magnetic resonance imagery* very often used in radiology for noninvasive interior medial examination.

[21]Lately a new industry standard format [Ani] for efficient storage and access of sparse volumetric datasets has been established

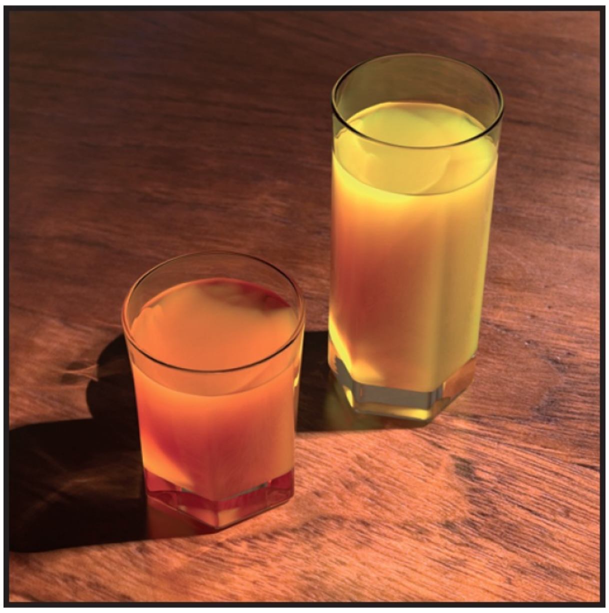

Figure 2.9: Left image is glass with juice[17], the volume is bounded by triangle mesh, source [NNDJ12b]. On the right we can se a simple procedural box filled with heterogenous fog[18]

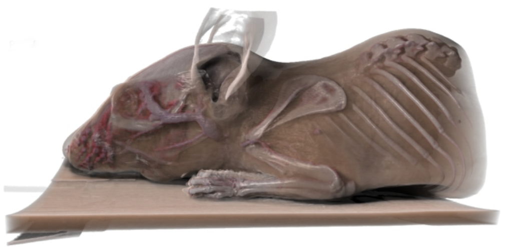

Figure 2.10: Example of a volumetric MRI data set consisting of an image stack. Rendered using Voreen [RDRS10].

Figure 2.11: Smoke simulation with a simulation grid overlay, each grid cell represents one voxel with different density[1].

[1]For the smooth looking results linear interpolation of neighboring values is used.

Chapter 3

Common solutions

In this chapter we will try to summarize the most common techniques for solving the rendering equation in the presence of participating media. Many methods are used today and it's way beyond the scope of this thesis to do a proper summary of them, so only widely used methods related to the context of this thesis were picked.

3.1 Rasterization

Rather than talking about game solutions, where is the main goal to render everything real time and physical accuracy of lighting is less important than plausible smooth graphics, we will have a look on rasterzation used in off-line rendering.

Rasterization comes as a nature choice, when it comes to particle rendering. It's key benefit is that the whole scene geometry doesn't have to fit into the RAM [1], in the cost of linear rendering complexity. Because list of primitives, be it points, lines or triangles, is rasterized one by one in the frame-buffer.

[1]Computer main random access memory.

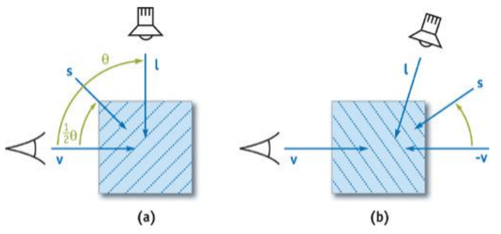

Figure 3.1: Illustration of the half axis angle determination for single particle sorting pass. Source [NVI].

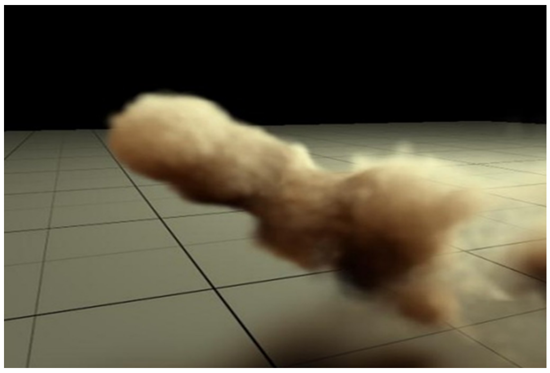

Figure 3.2: This image has been rendered using multiple shadow maps for particle direct illumination on gpu. Source [NVI].

Almost all the algorithms widely used rely on the same kind of preprocessing stage. The particles should be ideally sorted from the furtherest to the closest, and rasterized in this order, so that the alpha blending works correctly. Which is quite costly, considering we should sort millions of particles twice. Once for the camera view and once for the shadow map generation light view. To partially overcome this issue a half-angle axis sorting can be used, where we sort particles along a half angle axis between the camera and the light only once as can be seen on fig. 3.1.

Also shadow map containing only a nearest depth is practically useless when used with millions of partially transparent particles. To cope with this we can either rasterize the particles in the common layer batches for camera view and light view to sample progressively reducing transmittance as used in nVidia demo fig. 3.2.

Or we can eventually use deep shadow maps [LV00], where is the occlusion function of depth[2] instead of a single value per pixel in the case of regular shadow maps. This approach allows pre-filtering[3] which leads to suppression of both depth and aliasing artifacts commonly seen in shadow map renderings.

[2]The function is usually represented as lookup table containing [depth,transmittance] tuples.

[3]Object can occlude only partial part of the pixel, thus adding only appropriate amount of occlusion in incident depth.

Image courtesy of Digital Domain
Image courtesy of Warner Bros. / Gradient Effects
Image courtesy of Paramount Pictures and Prime Focus VFX

Figure 3.3: Examples of rasterization Krakatoa particle render used in feature film productions [Thi] .

Even tough only single scattering volume interaction can be modeled this way, it's very popular method widely used in the VFX industry, because it can handle very efficiently even billions of particles as can be seen on fig. 3.3.

3.2 Raytracing

Another very popular set of methods for solving rendering equation is based on ray-tracing. Instead of rasterizing (drawing) directly visible elements, concept of ray scene interaction is used, where set of rays originating either in camera or in light are shoot into the scene space to gather light information. Which is way more flexible than rasterization, in the terms of complex light scene interaction and logarithmic cost of visibility computation[4].

The flexibility unfortunately goes in hand with an additional cost of acceleration structure build and usage for a ray primitive intersection. That's why these algorithms are so memory demanding. Because majority of the scene has to be loaded in the RAM for efficient visibility calculation.

In the following subsections the key ray-tracing concepts and methods are briefly overviewed.

3.2.1 Visibility evaluation

The key aspect of ray-tracing volumes is the radiance integration between two points in space. Lets consider only transmittance change along one ray. In homogeneous media a simple analytic solution exists. Since κ_t is constant we just evaluate the $\tau(x_0, x) = e^{-\kappa_t * |x_0 \to x|}$.

In order to compute transmittance in non-homogeneous media according the eq. 2.21, we

[4]Not all the elements of the scene have to be usually intersected. And the complexity of ray element intersection in the present of volume partitioning acceleration structure such as KD or BVH tree is logarithmic.

Figure 3.4: Examples of unbiased ray-tracing methods, from left *path tracing, light tracing, bidirectional path tracing* . The principle is pictured on the top raw and example renderings are in the bottom one.

can use ray-marching technique. Which evaluates the transmittance coefficient of the media $\kappa_t(x_v)$, in a point $x_v = \vec{r} * v$, where $\vec{r}$ is a ray direction and v is some chosen step size. Ideally we should choose very small v, which would inevitably lead to render time increase.

More recently, another technique has become popular for sampling distances along a ray in an inhomogeneous medium. The idea comes from the neutron transport community in the 60s and has various names (delta tracking, pseudo-scattering); we'll call it Woodcock tracking in effort to promote original paper.

pbrt single scattering can support procedural volumes and any type of lighting

3.2.2 Unbiased methods

Path tracing, light tracing fig. 3.4 Plus and cons - very slow no caching re-computation of visibility factor

3.2.3 Biased methods

[Jar08] irradiance caching, Photon tracing, final gather ... Plus and cons

Chapter 4

Consistant progressive methods

what does it mean to be progressive

4.1 Stochastic progressive photon mapping

[]

4.2 Beam mapping

[JNT+11] Hybrid solution cpu gpu rasterization..

4.3 Virtual point lights

[Kel97] [HKWB09]

4.4 Virtual ray lights

[NNDJ12b]

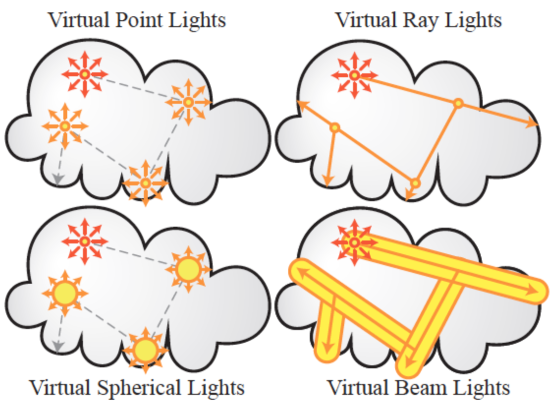

Figure 4.1: During the preprocessing stage photon paths are shoot into the participating media. This image compares the methods we can store and evaluate cached radiance in the form of virtual lights. This figure demonstrates the analogy of virtual spherical lights to virtual point lights with virtual beam lights to virtual ray lights. Source [NNDJ12a].

4.5 Virtual beam lights

As you can se on the image fig 4.1 [NNDJ12a]

Chapter 5

Proposed solution

I have chosen these methods and why... How to do it scheme of the progressive raytracing renderer.

Even though it offers all the basic functions necessary for raytracing,we had to decide what extensions well be needed. This chapter contains description of major software architecture decisions and feature implementations we have made.

Figure 5.1: Proposed progressive rendering system architecture.

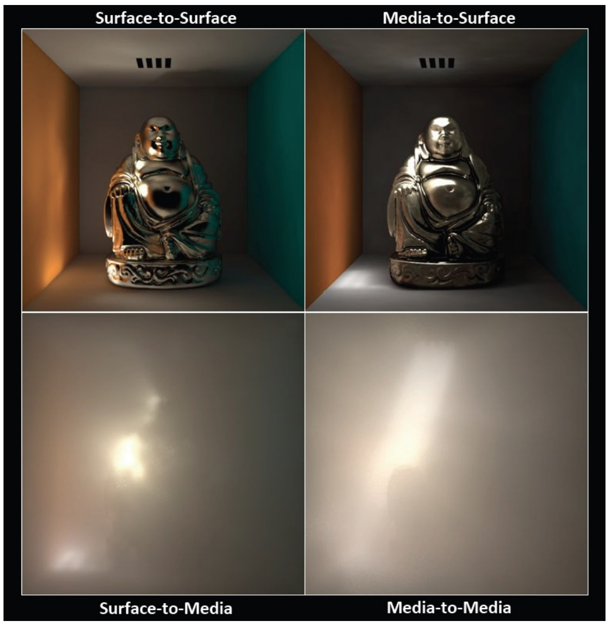

Figure 5.2: For better understanding of the ways how participating media interact with the light and surfaces, we can divide the complex interaction into four simpler subroutines. Source [NNDJ12a].

5.1 Progressive rendering system architecture

5.1.1 Light transport paths

Different light transport paths surface surface, surface media, media surface, media media fig. 5.2.

5.2 Ground truth

light tracer - dual algorithm to path tracer. It is able to capture caustic effects. Maybe path tracer too??

co tu ma byt pouzil jsem tuhle a tuhle metodu proc odkazat se na predchozi kapitolu a zminit zmeny ktere jsem provedl a proc.

Co jsem orezal co rozsiril.

A dojit k blokovemu schematu co budu muset imlementovat. Mozna zminit i PBRT.

Chapter 6

Implementation

All the selected algorithms are based on Monte Carlo ray tracing methods, thus we would have to implement, all the ray casting methods, scene loaders and acceleration structures from the ground up. An efficient and easily extensible implementation of the ray tracing core would take us probably much longer than the implementation of the algorithms we want to investigate.

Thus we have deiced to implement the algorithms into PBRT[1], which we use as a core framework for the raytracing. It's a full featured ray tracer, containing all we need to implement and test selected algorithms.

6.1 Rendering system architecture

Lets see how well we will be able to implement proposed architecture from chapter 5 into the PBRT framework. On the fig 6.1 we can see an architecture of the ray tracer which uses photon rendering. As you can se we have to make few changes to effectively implement progressive renderer suggested on fig 5.1.

The main difference is the lack of progressive rendering stage which reuses scene acceleration structures for photon shutting and rendering passes. And also a shared frame buffer for final image rendering. So we have implemented new renderer into PBRT with the following structure fig 6.2.

[1]Physically Based Rendering toolkit source [PH04]

Figure 6.1: A diagram of PBRT photon casting renderer.

Figure 6.2: A diagram of new progressive render implementation using PBRT.

Figure 6.3: Two distinct 3D functions were implemented to represent various heterogenous media. Multiple octaves of 3D Perlin noise on left and custom step function on right.

6.2 Volume representation

To test selected algorithms we have to decide which volume representations we will use to model different kinds of volumetric phenomenons.

- **Bounding volumes** We have chosen to implement a bounding volume representation, using a volume box. To model the heterogenous density inside the volume box, we use multiple octaves of 3D Perlin noise function and custom step function. This this way we have a memory friendly, easily expandable and adjustable volume representation we can use to test out selected algorithms. To see some examples of the functions used, have a look at fig 6.3. As you can see 3D Perlin noise is very useful for modeling heterogenous fog environments. While the step function will help us in the debugging stage, where we will test the correctness of photon volume multiple scattering behavior.

- **Unified grid** To test out real world production scenarios, we have decided to use PBRT internal unified grid representation, where we can directly load any 3D fluid simulation. Have a look at 6.4, as you can see this way we can nicely represent smoke plumes.

6.3 Brdf

What brdf will be used, importance sampling implementaion

Figure 6.4: PBRT 3D uniform grid structure used to hold fluid simulation data.

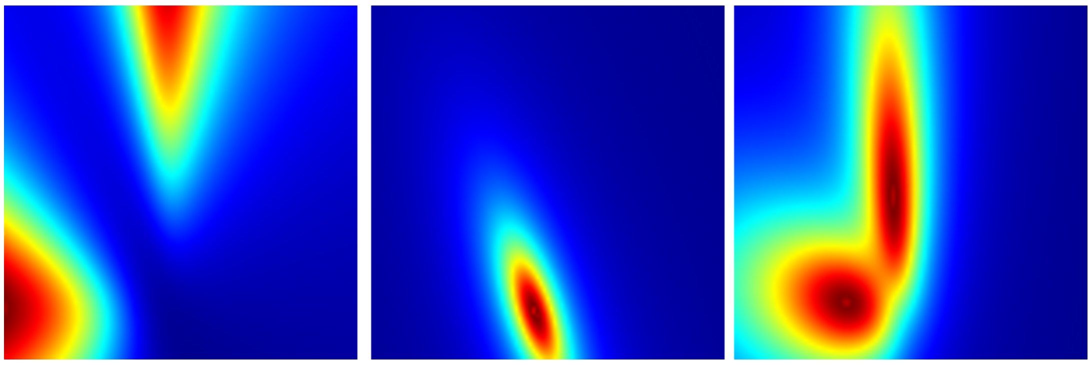

Figure 6.5: Different functions have been evaluated in discreet point pairs (from 0 to 1 parametric range) on two randomly placed rays.
On the left image only contribution using ray moderately forward scattering (g=0.75) Heney-Greenstein phase functions has been evaluated. In the middle inverse squared contribution and on the right contribution using both phase function and inverse sqared distance.

6.4 Phase funcitons

What phase func will be used and imp sampl implem

V tehle kapitole muzu i ukazat ty veci z matlabu ze funguji a jak. Klidne popsat, i postup implemntace pres jednoduzsi az po slozitejsi metody.

Chapter 7

Results

Mely by tu byt tabulky a srovnani metod a ruzne parametry.

Chapter 8

Conclusion

this method was implemented and results are great

8.0.1 Future work

Bibliography

[Ani] DreamWorks Animation. Opensource format for efficient storage of sparse volumetric data.@ONLINE. URL: http://www.openvdb.org/.

[Bou29] P. Bouguer. *Essai d'optique sur la gradation de la lumière.* 1729. URL: http://books.google.co.uk/books?id=JNkTAAAAQAAJ.

[Cha60] S. Chandrasekhar. *Radiative transfer.* Dover Books on Physics Series. Dover Publications, Incorporated, 1960. URL: http://books.google.cz/books?id=CK3HDRwCT5YC.

[HKWB09] Miloš Hašan, Jaroslav Křivánek, Bruce Walter, and Kavita Bala. Virtual spherical lights for many-light rendering of glossy scenes. *ACM Trans. Graph.*, 28(5):143:1–143:6, December 2009. URL: http://doi.acm.org/10.1145/1618452.1618489, doi:10.1145/1618452.1618489.

[Jar08] Wojciech Jarosz. *Efficient Monte Carlo Methods for Light Transport in Scattering Media.* PhD thesis, UC San Diego, September 2008.

[JNT$^+$11] Wojciech Jarosz, Derek Nowrouzezahrai, Robert Thomas, Peter-Pike Sloan, and Matthias Zwicker. Progressive photon beams. *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH Asia 2011)*, 30(6), December 2011.

[Kel97] Alexander Keller. Instant radiosity, 1997.

[Lav] Philip Laven. Mie plot software illustration image. URL: http://www.philiplaven.com/mieplot.htm.

[LV00] Tom Lokovic and Eric Veach. Deep shadow maps. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '00, pages 385–392, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co. URL: http://dx.doi.org/10.1145/344779.344958, doi:10.1145/344779.344958.

[NAS] NASA. Laser optics. URL: http://gimp-savvy.com/.

[NNDJ12a] Jan Novák, Derek Nowrouzezahrai, Carsten Dachsbacher, and Wojciech Jarosz. Progressive virtual beam lights. *Computer Graphics Forum (Proceedings of EGSR 2012)*, 31(4), June 2012.

[NNDJ12b] Jan Novák, Derek Nowrouzezahrai, Carsten Dachsbacher, and Wojciech Jarosz. Virtual ray lights for rendering scenes with participating media. *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH 2012)*, 31(4), July 2012.

[NVI] Simon Green NVIDIA. Nvidia cuda sdk example, smoke particle demo. URL: `http://docs.nvidia.com/cuda/cuda-samples/`.

[PH04] Matt Pharr and Greg Humphreys. *Physically Based Rendering: From Theory to Implementation*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2004.

[PN] Petra Glazerova Petr Nejtek. Crepuscular rays.

[RDRS10] Timo Ropinski, Christian Döring, and Christof Rezk Salama. Advanced Volume Illumination with Unconstrained Light Source Positioning. *IEEE Computer Graphics and Applications*, 2010.

[SAL] Image Science and NASA Johnson Space Center Analysis Laboratory. Aurora borealis from iss. URL: `eol.jsc.nasa.gov`.

[Thi] Thinkbox. Thinkbox software's production-proven volumetric particle rendering@ONLINE. URL: `http://www.thinkboxsoftware.com/krakatoa/`.

Appendinx A

Appendix

abreviations + test scene images

Appendinx B

DVD Content

galerie obrazku test scen