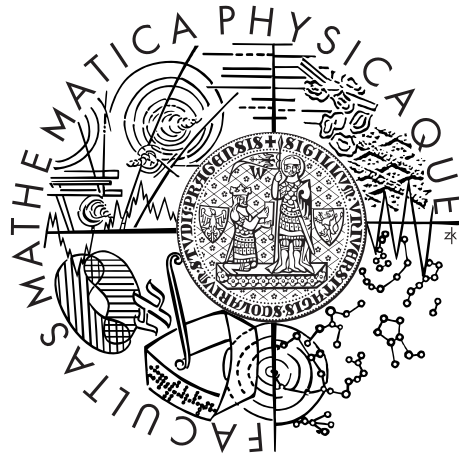


Charles University in Prague
Faculty of Mathematics and Physics

MASTER THESIS



Oskár Elek

Physically-based Cloud Rendering on GPU

Department of Software and Computer Science Education

Supervisor of the master thesis: doc. Dr. Alexander Wilkie

Study programme: computer science

Specialization: software systems

Prague 2011

Here I would like to thank Alexander Wilkie for the valuable discussions and general advice. My family, especially my mother Eva, for their support in my studies. My girlfriend Paula for helping me overcome the stress associated with the work.

I declare that I carried out this master thesis independently, and only with the cited sources, literature and other professional sources.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular the fact that the Charles University in Prague has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 paragraph 1 of the Copyright Act.

In date

Author signature

Název práce: Fyzikální Vykreslování Oblaků na GPU
Autor: Oskár Elek
Katedra: Kabinet software a výuky informatiky
Vedoucí diplomové práce: doc. Dr. Alexander Wilkie

Abstrakt: Optická simulace participujících medií je zajímavý a taky důležitý problém, který ale nemá žádné jednoduché řešení. Mezi participujícími médii lze navíc oblaky, díky jejich pro simulaci složitým vlastnostem, chápat jako obzvláště náročný případ. Cílem této práce je navrhnout řešení tohoto problému a to navíc takové, které by tuto simulaci provádělo interaktivně. Hlavními kritérii při navrhování této metody byly její fyzikální věrnost a maximální využití některých výhodných vlastností oblaků, které by nám pomohly vyvážit jejich složitou podstatu. Ve výsledku je námi navrhovaná metoda postavená na algoritmu fotonových map, kterou ale zásadním způsobem modifikujeme tak, aby bylo dosaženo její interaktivity a časové koherence. Tomuto napomáhá i fakt, že jsme se při návrhu snažili, aby naši techniku bylo možné implementovat na současných GPU, jejichž masivně paralelní výpočetní výkon jsme chtěli využít. Prototyp naší metody jsme implementovali v aplikaci, která je schopná interaktivně vykreslovat (zatím pouze) jeden oblak. Naše diskuze se tedy především zabývá tím, jak tento prototyp naší metody zlepšit natolik, aby jej bylo možné použít v různých praktických aplikacích v průmyslu.

Klíčová slova: fyzikální vykreslování oblaků, rozptyl světla, participující média, photon mapping, programování GPU

Title: Physically-based Cloud Rendering on GPU
Author: Oskár Elek
Department: Department of Software and Computer Science Education
Supervisor: doc. Dr. Alexander Wilkie

Abstract: The rendering of participating media is an interesting and important problem without a simple solution. Yet even among the wide variety of participating media the clouds stand out as an especially difficult case, because of their properties that make their simulation even harder. The work presented in this thesis attempts to provide a solution to this problem, and moreover, to make the proposed method to work in interactive rendering speeds. The main design criteria in designing this method were its physical plausibility and maximal utilization of specific cloud properties which would help to balance the complex nature of clouds. As a result the proposed method builds on the well known photon mapping algorithm, but modifies it in several ways to obtain interactive and temporarily coherent results. This is further helped by designing the method in such a way which allows its implementation on contemporary GPUs, taking advantage of their massively parallel sheer computational power. We implement a prototype of the method in an application that renders a single realistic cloud in interactive framerates, and discuss possible extensions of the proposed technique that would allow its use in various practical industrial applications.

Keywords: physically-based cloud rendering, light scattering, participating media, photon mapping, GPU programming

Contents

1	Introduction	3
1.1	Light scattering	4
1.2	Motivation	5
1.3	Thesis goals	6
1.4	Organization	7
2	Physical and mathematical background	8
2.1	Participating media	8
2.2	Physics of light transport	15
2.3	Spherical harmonics	19
3	Related Work	21
3.1	Photon mapping	21
3.2	Non-interactive methods	22
3.3	Interactive methods	24
4	Method overview	27
4.1	Assumptions and rationale	27
4.2	Brief algorithm description	31
5	Detailed method description	35
5.1	(P) Gradient computation	35
5.2	(P) Cloud volume sampling	36
5.3	(P) kD-tree construction	36
5.4	(P) Peak densities computation	40
5.5	(P) VPCs and billboards generation	41
5.6	(P) Photon map initialization	42
5.7	(R) Photon tracing	43
5.8	(R) Illumination reconstruction	44
5.9	(R) Cloud visualization	45
6	Method implementation	48
6.1	Environment and libraries	48
6.2	Implementation design	49
6.3	Evaluation	51
7	Conclusion	55
7.1	Fulfilment of the thesis goals	55
7.2	Discussion	56
A	CD contents	63
B	AtmoVision short user reference	64
B.1	System requirements	64
B.2	Installation	64
B.3	Usage	64

1. Introduction

The desire of human beings to graphically reproduce the surrounding reality accompanies us from the formation of first simple human societies. There is no surprise to this, as the majority of information perceived by our senses is visual. As a consequence, although there have always existed various abstract and symbolic directions in visual arts, the main emphasis have always been on realistic depiction of reality — starting with cave paintings of human and animal figures and other common objects, through realistic architectural paintings during antiquity, up to renaissance painters. The reproduction techniques naturally became more complicated and elaborate, as the time went — for instance, painters during the Dutch Golden Age era strived not only to realistically reproduce proportions of the objects of their interest, but also to create realistic impressions of light conditions in the scene.

Up to this point, all visual arts had one thing in common: the reproduction process always consisted of a painter creating their piece using some sort of colourant, based on their own impressions. This radically changed with the advent of photography in the early 19th century. Photography removed the subjective element from the reproduction process — the painter — producing a truly objective image, so similar to the very perception of humans. Combined with the relatively fast acquisition process, photography soon became the way to document our reality via newspaper reports and such. And in spite of this, photography still remained an art; it just transformed the artistic focus from creation of the painting itself into inventive preparation of the scene to be photographed and illumination in it.

However, it soon became apparent that the advantage of objectivity is also one of the weaknesses of photography; anything that is supposed to be on a photograph has to be physically present in the photographed scene. Needless to say, this can be very difficult or costly sometimes, if not completely impossible (such as taking photographs of surfaces of distant planets, acquisition of environments with extreme temperatures etc.).

The solution to this limitation came much later with the birth of a new science by the half of the 20th century — computer science — which concerns itself with the recently invented digital computers. During the early decades of their existence, digital computers did not have sufficient performance for massive physical simulations. Nevertheless the available computational power grew exponentially and by the 1960s and 1970s the computer graphics field was born, along with one of its most important branches, *rendering* [7]. Rendering is a process of creating synthetic images or sequences of images based on a mathematical description of the rendered object or scene. Rendering changed the paradigm of the previous few millennia — the reproduced object did not have to physically exist, nor was it a figment of artist’s imagination, but the basis for the rendered image resides in the memory of a computer instead. This of course allows for arbitrary changes of rendered objects within the limitations of a particular rendering system.

Although rendering concerns itself with generation of any kind of imagery, the important for us in this context is its main sub-field, physically-based rendering. As the name suggests its focus lies on synthesis of images based on the existing laws of physics. In most cases physically-based rendering draws from the laws of geometrical optics, which is the highest abstraction above quantum optics [4]. The main reason

for this is that geometrical optics maps fairly well onto the existing computer architectures, as its description of light propagation is limited to infinitesimally thin rays of energy and as that it seldom contains continuous quantities, which as we all know are particularly troublesome for contemporary computers.

The main focus of physically-based rendering is computation of the *rendering equation* [35] (abbreviated RE hereinafter; see Section 2.2 for the exact formulation of both surface and volumetric RE). RE is a Fredholm integro-differential equation which evaluates intensity of light coming from some direction in a scene, given the description of light sources, surfaces and geometric relations in the scene. The main difficulty lies in the recursive nature of the equation — if we want to find the amount of light energy coming from some direction, we must first calculate the amount of light energy that bounces from the point we are looking at, from all directions visible from this point. In theory, the number of bounces any given light ray can undergo is infinite; and although in reality this is practically impossible, this number is still high enough to employ researchers trying to find a universal solution for this equation for three decades, yet still without a convincing success. Of course, many algorithms for solving RE exist, however none of them is universally usable in all conditions, which limits the artists who in the end want to employ RE to obtain photorealistic images.

1.1 Light scattering

One of the main foci of physically-based rendering is visual simulation of natural phenomena, as opposed to rendering of artificial objects and materials produced by human. Natural phenomena include many kinds of processes in nature and within the field of physically-based rendering they are often loosely categorized by the modelling or simulation method which is most suitable for a given phenomenon. Yet they have one thing in common: most natural phenomena exhibit a way of interacting with light called *light scattering*.

Light scattering is a physical process caused by *participating media*. Particles of such a medium can influence light passing through it by diverting it into another direction, allowing it to reach places it originally would not reach. An attentive reader can correctly notice that light scattering is a very broad term; almost any interaction of light with environment can be considered as scattering. Therefore to avoid confusion, in rendering we often regard light scattering as being caused by media which are at least partly translucent or transparent. This narrows down the selection a bit, but still includes vast amount of organic and inorganic materials — organic tissues, minerals, plastic materials, liquids, gases, and many more others. In all these media, light scattering causes light to behave in many distinctive ways, causing all sorts of visual features. These features are generally very important to simulate, if one wants to realistically reproduce appearance of such media.

The main difficulty of participating media is that even the rendering equation, however complicated, is not sufficient for simulating them. The reason lies behind the formulation of RE — it describes light interactions on discrete material boundaries only, assuming that the space between these surfaces is filled with vacuum. As we will see in Section 2.2, we can overcome this shortcoming by extending RE with another term, which accumulates scattered light along a path between two examined surface points. However, this extension complicates the matter even more — the interaction of light is not limited to discrete surfaces, but it can now happen at an arbitrary point in space!

The work described in this thesis concerns itself with the visual simulation of one such object type, the clouds. Clouds are generally composed of water in all its three phase states, and from additional minor compounds. All of these substances are participating media, and as we will see in Section 2.1, most of them are particularly difficult to simulate. We will present a method which was specifically designed to overcome these difficulties by exploiting some other specific properties of clouds. Thanks to this, not only the presented method is able to simulate light behaviour in clouds on a physical basis, but it can do so in interactive frame rates.

1.2 Motivation

Rendering of any kind of outdoor environments inherently requires taking sky and clouds into account. Even if not directly visible, they are responsible for many important lighting characteristics of our living environment. In his previous work ([15, 16]) the author dealt with the problem of interactive visual simulation of planetary atmospheres, which enabled real-time applications to include photorealistic physically-based skies. Unfortunately these skies were cloudless, because the proposed method simply did not take clouds into account in the simulation. The work described in this thesis attempts to remedy this deficiency.

In addition to the general rendering algorithms capable of handling participating media, several methods for non-interactive rendering of clouds exist ([36, 19, 54, 44]). However, these methods are too computationally demanding to be applicable in interactive environments. Yet there are numerous interactive and real-time applications that could benefit from realistic physically-based clouds:

- **Flight simulations**

Visual fidelity of training simulations for pilots is their second most important aspect, right after the correctness of flight physics simulation. This is a very demanding application, because the dynamism of the involved environments does not allow for usual compromises, such as usage of cloud impostors or even static skyboxes.

- **Computer games**

Many computer games take place in outdoor environments, making them natural targets for real-time cloud rendering methods. The problem here is basically the same as with flight simulators; the dynamic gaming environment imposes many limitations on the potentially useful cloud rendering algorithms.

- **Scientific visualizations**

Meteorological visualizations can naturally take advantage of interactive visual simulations of clouds. In addition, all other visualization techniques that deal with volumetric participating media can put such method (or its parts) in use.

- **Architectural rendering**

Practically all architectural visualizations include outdoor environments. Of course, the techniques used in architectural rendering are by far not interactive. On the other hand, realistic interactive visualization techniques could be

used for previewing the final image, making setting of scene parameters much quicker and easier.

Naturally, there are numerous methods that attempt to solve the problem of interactive cloud rendering. However, as you will see in Section 3.3, each of these approaches has certain drawbacks (e.g. too limiting assumptions or non-physical *ad hoc* lighting computation). This makes us think that the problem of interactive physically-based cloud rendering is far from being solved. Therefore in this work we propose a solution to this problem that not only removes some of the previous methods' limitations, but also does this in a physically plausible manner.

The natural question in this point would be, why there is need for a specialized method for cloud rendering; can't one just use any general method for rendering of volumetric media? Unfortunately it is not that simple. First, as we previously mentioned, for every general method there are pathological cases where the method performs very poorly or even fails to produce the solution at all. Second and mainly, clouds themselves are a very difficult medium to handle; they are costly to store, they exhibit very high amount of anisotropic scattering orders, they can have a huge variety of shapes (making them difficult to localize), and so on. On the plus side, any method capable of handling clouds well will be applicable to many other types of participating media as well.

1.3 Thesis goals

The main objectives of this work were the following:

1. Thorough examination of the existing techniques for rendering participating media with focus on interactive cloud rendering methods. Identification of main limitations of these methods and utilization of this knowledge in the subsequent steps.
2. Design of the prototype of a novel interactive method for physically-based cloud rendering. The main imperative here is that the design should take into account all important cloud media properties: high albedo, strong scattering anisotropy, high average amount of light scattering orders, spatial medium density fluctuations. The method also has to build on realistic assumptions, such as slow light conditions changes and also slow medium shape changes (disallowing any precomputations). The usefulness of such technique is immediately obvious in applications with slow environment changes, for instance soaring simulations as well as other applications mentioned in Section 1.2. As such, it is acceptable for the method to be biased.
3. Proof of feasibility of the proposed method prototype by its implementation. The resulting application should allow its user to freely navigate around at least a single cloud in its natural environment, the sky. It is assumed the implementation will take advantage of the high parallel computational power of contemporary GPUs to speed up the simulation, as well as to visualize its results.

4. Description of the proposed method prototype in a thesis text. Discussion about shortcomings of the method and suggestions how to optimize and extend it in the author's future work to be usable in practical applications.

1.4 Organization

The text of this thesis is logically divided into chapters in a way that allows even a reader with modest mathematical knowledge to understand the topic of specialized participating media rendering, even if their knowledge of the relevant physics and computer-graphics techniques is minimal.

We start by introducing the directly relevant physical laws and mathematical concepts in Chapter 2. Then we proceed to the related works from the field of rendering in Chapter 3.

The first look on the clouds specifically will be taken in Chapter 4. Here we introduce the most significant properties of clouds, along with a few observations that will help us find compromises allowing us to accomplish our main objective. After that we finally give the first condensed description of our method prototype. We will refer to this description in the subsequent chapters.

Chapter 5 then describes the previously outlined method steps in much greater detail, but mostly from the theoretical point of view. This is then complemented by Chapter 6, providing more practical and low-level perspective on our approach. After the technical evaluation at the end of the chapter we finally conclude our findings, and also elaborate on the future prospects of the proposed technique, in Chapter 7.

The figures in each chapter are organized in one of two ways. Achromatic figures and pictures where the colour does not play an important role are incorporated directly into the text. On the other hand colourful pictures and photographs can be found at the end of each respective chapter, printed in colour.

The text contains a sizeable amount of symbols that denote various physical and other quantities. Appendix C lists and describes the majority of these symbols in a concise table.

2. Physical and mathematical background

The following sections describe the physical and mathematical basis that we will refer to through the rest of the text. Section 2.1 describes participating media and their important characteristics. Section 2.2 introduces the *volume rendering equation*, compares it to the already mentioned surface rendering equation, and introduces a few important physical laws that describe transfer of electromagnetic energy through participating environments. Finally, Section 2.3 briefly introduces the concept of spherical harmonic functions, which is used by our method for representation of local cloud illumination.

2.1 Participating media

The notion of participating media in computer graphics has already been introduced in Section 1.1. Let us now look at some important properties of participating media that influence the way how they interact with light (for an exhaustive physical description of light scattering please refer to the book by van de Hulst [56]).

Types of interaction Although light scattering is a wave phenomenon (it manifests the wave properties of light), it is quite helpful to look at the process of light interaction with participating media from the abstracted point of view of a single photon. A photon passing through the volume of a particular participating medium can interact with this medium in two ways: it can either be absorbed or scattered. In addition to that a new photon can be emitted. These three event types can happen at any point in the medium, from the macroscopic point of view.

- **Absorption** If a photon passing through a medium is absorbed, its energy is transformed into another form, such as kinetic energy of the medium particle. Absorption decreases the intensity of light passing through a medium; for example, water absorbs light with longer wavelengths better, causing any light passing through a volume of water to become blue after some time (or rather after some travelled distance).

The strength of absorption of a medium is guided by the *absorption coefficient* σ_a [m^{-1}]. Absorption coefficient describes the mean free path of a photon in a medium; for instance in a homogeneous substance with $\sigma_a = 0.5\text{m}^{-1}$ the mean distance a photon travels until it is absorbed is 2m.

- **Scattering** If a photon is scattered its energy is diverted into another direction (this direction depends on the medium *phase function*, which will be described later). For instance, our atmosphere scatters the short blue wavelengths better, diverting more blue photons into the direction of ground observer, making the sky appear blue as well.

Similarly to absorption, scattering is guided by the *scattering coefficient* σ_s [m^{-1}]. Its meaning is equivalent to the σ_a as well.

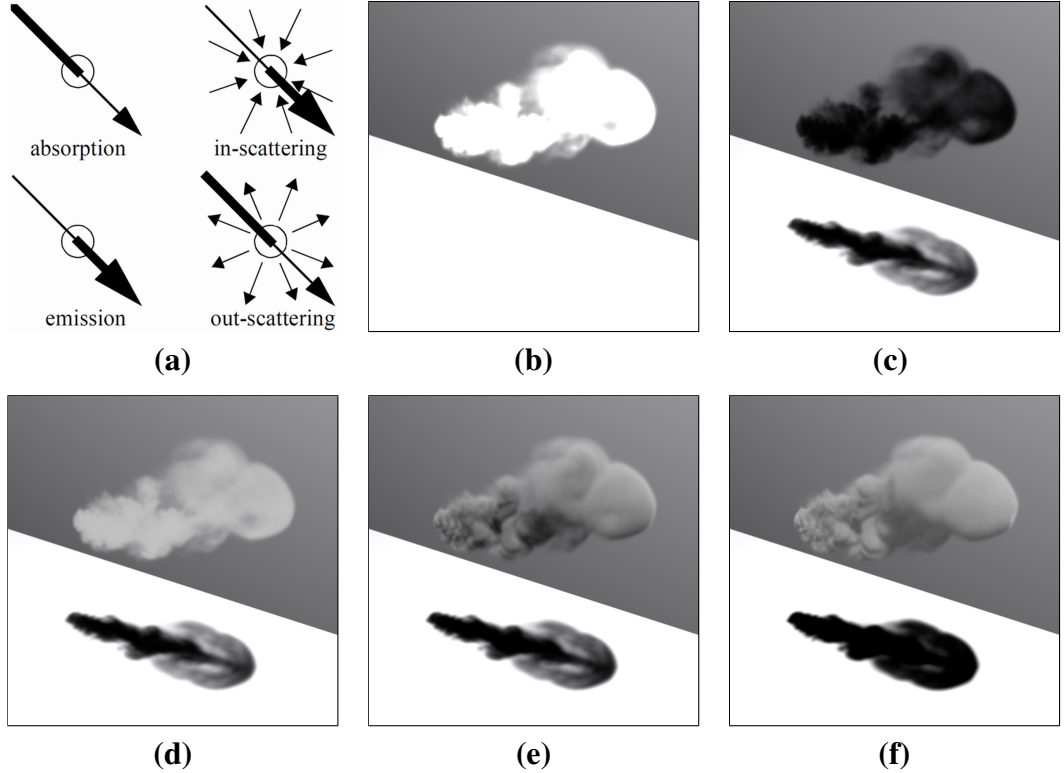


Figure 2.1: Interaction types in participating media. (a) schematically shows how these interactions increase or decrease intensity of light after each respective event type. The remaining images show the effects of these interactions in a smoke cloud dataset: emission alone (b), absorption alone (c), emission and absorption (d), scattering (e) and finally all three combined (f).

- **Emission** Finally, a medium can emit photons from within its volume, for example as a consequence of blackbody radiation. Emission is controlled by the *emission coefficient* σ_e [m^{-1}]. The meaning of σ_e is a little different than previously; in a medium with a given σ_e , on average σ_e photons are emitted per 1m long path through the medium.

Please note that further in this text we will not describe volumetric emission. This is mainly because we do not need to deal with emissive media in our work, and furthermore, the inclusion of emission into simulation is usually trivial, as it is a constant of a spatially-varying diffuse term (which is simply added to the resulting light intensity) in practically all cases.

- **Extinction** Extinction is not a distinct type of interaction; rather it is a combined effect of absorption and scattering. Thanks to the linearity of these two interactions one can simply define extinction as an event when a photon is either absorbed or scattered, decreasing the intensity of a beam of light passing through a medium by the photon's energy. Consequently we define the *extinction coefficient* σ_t [m^{-1}] as $\sigma_t = \sigma_a + \sigma_s$.

Furthermore, based on σ_t , we define scattering efficiency or the *scattering albedo* α as

$$\alpha = \frac{\sigma_s}{\sigma_a + \sigma_s} = \frac{\sigma_s}{\sigma_t}. \quad (2.1)$$

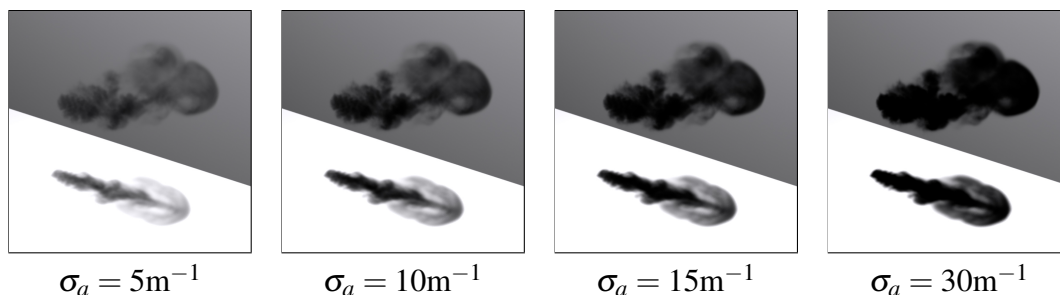


Figure 2.2: Effects of absorption strength increase on medium appearance.

The meaning of scattering albedo is the following: if a photon hits a medium particle with albedo α , it gets scattered with the probability of α and absorbed with the complementary probability, $1 - \alpha$. The higher the albedo of a participating medium is, the more difficult is simulation of the medium. This is because photons will propagate through a high- α medium for much longer, so the simulation will spend much more time tracing photons or rays though the medium volume, until they are culled away (by e.g. Russian roulette or fixed energy thresholding).

The albedo of clouds is the main reason why they are so difficult to simulate. Except for some cases most clouds have $\alpha = 1$. It means that any photon will ‘bounce around’ in a cloud until it files out of its volume. In combination with the relatively large spatial extent of most clouds, photons or light rays can easily scatter tens or even hundreds of times until they get out of the cloud volume [6]. This is indeed very computationally intensive.

Figure 2.1 illustrates these interactions. Notice the qualitative properties they produce. A purely emissive object does not cast a shadow, since there is no mechanism that would attenuate light. On the other hand absorption just attenuates light, so the cloud appears black (this is a behaviour similar to e.g. coal dust). However, none of these two gives good information about spatial relations and mass distribution in the cloud — only scattering does, simply because it is the only interaction that redirects light. Also notice that the combined effect of absorption and scattering creates darker shadow than just absorption alone. For this reason the extinction coefficient σ_t contains both σ_a and σ_s ; the only difference is that while absorbed light is ‘lost’, scattered light just changes its direction and ultimately, a portion of it also reaches the observer at a side.

As we will see in Section 2.2 the relation between the coefficients and the increase or decrease of light intensity is not linear, but exponential instead. The effect of increasing value of σ_a is shown in Figure 2.2; arguably, the cloud with $\sigma_a = 30\text{m}^{-1}$ is less than six times ‘darker’ than the one with $\sigma_a = 5\text{m}^{-1}$ (please bear in mind that the conversion from HDR to LDR might have distorted these relations).

Fortunately, if an object contains multiple substances, each with its own values of the described coefficients, these can simply be added together and the resulting values be used during simulation. Therefore different media types do not interfere with each other. One needs to treat multiple mixed media separately only if they have different phase functions (see below).

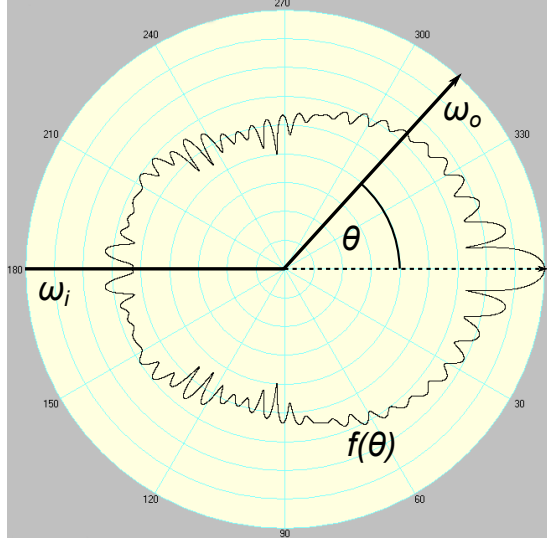


Figure 2.3: Polar plot of a phase function $f(\theta)$.

The coefficients σ_a and σ_s are derived from the absorption and scattering particle cross-sections C_a [m²] and C_s [m²]. C_a and C_s are proportional to the projected area of a single medium particle towards the incoming light direction. They express the amount of absorbed and scattered light by such a particle. To obtain σ_a and σ_s of a medium one simply has to know C_a and C_s of a single particle in the medium and the medium particle density η [m⁻³]:

$$\sigma_a[\text{m}^{-1}] = C_a[\text{m}^2] \cdot \eta[\text{m}^{-3}] \quad \sigma_s[\text{m}^{-1}] = C_s[\text{m}^2] \cdot \eta[\text{m}^{-3}]$$

A trivial consequence of these relations is that the absorption and scattering coefficients scale linearly with local medium density. We can therefore represent participating media by storing their maximal σ_a and σ_s and a normalized density field. Not only this strategy can decrease memory requirements (we can potentially need less bits per density sample), but also allows us to change the medium absorption and scattering potential independently by changing only the values of σ_a and σ_s (instead of altering the entire medium dataset).

Phase function So far we have only described how often scattering (and absorption and emission) events in a participating medium occur, yet we do not know what happens after such event takes place. As already mentioned above the light scattering is a wave phenomenon. If a light wave meets a scattering particle the particle will start to oscillate, creating a new ‘scattered’ light wave. The amplitude of this newly induced wave is given by the spherical *amplitude function* $F(\theta, \phi)$. Since we would like to work in the realm of discrete light rays and particles within geometrical optics, we will instead utilize the *phase function* $f(\theta, \phi)$, which is derived from F (please refer to [56] for details). Please also note that both F and f depend on the incident light wavelength λ , which from now on we will assume for all other quantities and functions, if not explicitly stated otherwise.

A phase function $f(\theta, \phi)$ is a non-negative function that depends on the outgoing scattering direction $\vec{\omega}_o$ with spherical coordinates $[\theta, \phi]$ (with respect to the incident direction $\vec{\omega}_i$). Moreover, the spherical integral of f equals to 1, which makes it a proper probability density function (abbreviated PDF); we will utilize this fact later.

Also, for spherical scattering particles the values of $f(\theta, \phi)$ are isotropic in respect to the azimuth ϕ . Since we will always limit ourselves to spherical scatterers we will drop the parameter ϕ from the notation of f ; hence from now on we will denote phase function as $f(\vec{\omega}_i, \vec{\omega}_o)$ or $f(\theta)$, where θ is the scattering angle between $\vec{\omega}_i$ and $\vec{\omega}_o$. Figure 2.3 depicts this relation.

$f(\theta)$ can be used in two ways in a rendering system that uses geometrical optics for light simulation. The first, direct use allows us to take $\vec{\omega}_i$ and $\vec{\omega}_o$ at a scattering point and use the corresponding value of f to scale the intensity of the light ray or particle that corresponds to $\vec{\omega}_o$. The second, more interesting use enables us to generate a new outgoing scattering direction $\vec{\omega}_o$ from the incident direction $\vec{\omega}_i$ with the probability given by f ; importance of rejection sampling can be used for this. Such approach saves us from scaling down the newly generated light ray or light particle intensity — the method itself ensures that the light energy carried by the generated samples will be angularly distributed according to $f(\theta)$. And thanks to f being a valid PDF, both us these techniques conserve energy.

The main issue with f is that it is generally very hard to evaluate. The values of both the amplitude function and the phase function are the result of a complex oscillating behaviour of the scattering particle induced by the incident light wave. In the great majority of configurations (meaning shape of the scatterer and the light wave state) f does not have a closed analytical form and therefore quite involved numerical methods have to be used to evaluate it. To avoid this we can make use of the two existing approximations [56], each building on a set of assumptions that hold quite well for certain media types.

- **Rayleigh theory**

Rayleigh theory describes light scattering by particles which are very small in comparison to the incident light wavelength. Written formally, the diameter d of a particle has to conform to

$$d \ll \frac{\lambda}{2\pi}$$

which holds quite well for particles with sizes ranging from individual atoms to small organic molecules. This theory has been described by John William Strutt, The Lord Rayleigh, in 1871. It states that the phase function for non-polarized light under such conditions is

$$f_{\text{Rayleigh}} = \frac{3}{4}(1 + \cos^2 \theta). \quad (2.2)$$

The plot of f_{Rayleigh} is shown in Figure 2.4 (left).

Another important property of Rayleigh scattering is that its scattering coefficient is inversely proportional to λ^4 , which means the shorter wavelengths are scattered much more than the longer ones. For this reason the Rayleigh scattering is chromatic and is responsible for many well known scattering phenomena, the most prominent being the sky colour. Also, as opposed to Mie scattering Rayleigh scattering yields a strongly polarized light, mainly in scattering angles around 90 degrees.

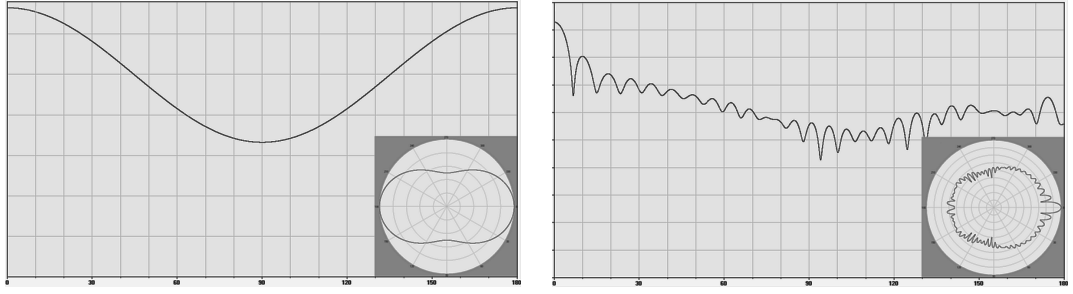


Figure 2.4: Rayleigh phase function (left, linear scale, $d = 20\text{nm}$, $\lambda = 450\text{nm}$) and an example of Mie phase function (right, \log_{10} -scale, $d = 4\mu\text{m}$, $\lambda = 450\text{nm}$) for $\theta \in \langle 0, \pi \rangle$ shown in Cartesian and polar coordinates. *The plots have been generated by the MiePlot software (<http://www.philiplaven.com/mieplot.htm>).*

- **Mie theory**

Mie theory has been published by Gustav Mie in 1908 [42]. Rather than a proper physical theory it is a solution of Maxwell’s equations for electromagnetic waves interacting with perfectly spherical dielectric particles of an arbitrary size.

There is no single Mie phase function; rather, for every particle diameter there exists a unique corresponding phase function. Figure 2.4 (right) shows an example of such function. One can immediately observe that it has quite a complicated shape, although it is band-limited. However, please note that the presented plot is in \log_{10} -scale — this means that the largest forward lobe is over 10 times stronger than the second one immediately next to it, and therefore the great majority of light energy is scattered under very small angles. This kind of behaviour is universal for Mie scattering: the larger the scattering particle is, the more light is scattered into forward directions.

Although the theory is limited to perfectly spherical dielectric particles, it is still very useful. For instance, most of the cloud mass is constituted by tiny water droplets, which generally have spherical shapes. Later in this section we will show how to approximate Mie phase functions, since once again there is no analytical formula that would express them.

Let us also remark that there is no sharp transition from Rayleigh to Mie scattering. Instead, Rayleigh scattering is a limit case of Mie scattering for very small particles. When decreasing the size of an examined spherical particle, its phase function eventually converges to f_{Rayleigh} . This means that Mie scattering is, in fact, also λ -dependent, but the differences of the Mie phase function values for different wavelengths are quickly disappearing as the scatterer size approaches the visible light wavelengths. Because of this, Mie scattering is usually considered to be an achromatic phenomenon.

To work with Mie scattering we need to evaluate its phase function somehow. There are two options to do that; either by using precomputed tabulated values of the rigorous phase functions, or by utilizing an analytical approximation. We prefer the latter variant, because it is technically simpler and more portable, which are very useful properties within the GPU environment.

One of the best established approximations to Mie phase functions is the Henyey-

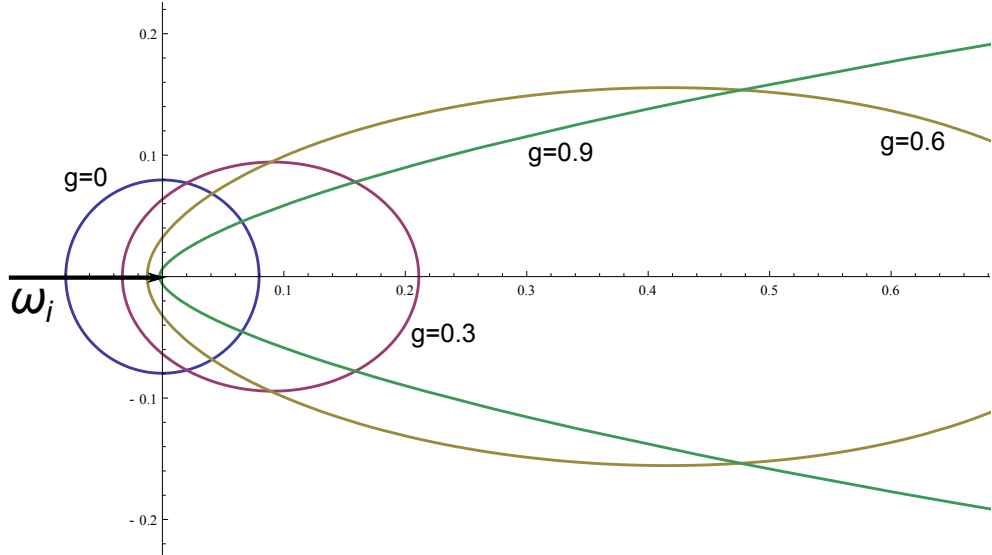


Figure 2.5: Linear polar plots of f_{HG} for $g \in \{0, 0.3, 0.6, 0.9\}$ with regard to θ . For negative values of g the function is just mirrored around the y -axis. Also note that for $g = 0$ the function is isotropic.

Greenstein function [25]. It is a function of two scalar parameters defined as

$$f_{\text{HG}}(\theta, g) = \frac{1}{4\pi} \cdot \frac{1 - g^2}{(1 + g^2 - 2g \cos \theta)^{3/2}} \quad (2.3)$$

where θ is the scattering angle and $g \in (-1, 1)$ is the scattering asymmetry coefficient. The value of g gives the function its shape — a positive g will cause forward scattering and vice versa. Based on g we also classify scattering as isotropic (if $g = 0$) and anisotropic (for all $g \neq 0$, which is the most common situation). Figure 2.5 shows graphs of f_{HG} for a few different values of g .

One of the nice properties of g is that it is not just an *ad hoc* parameter, but it can be obtained from an arbitrary phase function. For a phase function $f(\theta)$ g is defined as

$$g = \int_{\Omega_{4\pi}} f(\theta) \cos \theta \, d\vec{\omega}_o. \quad (2.4)$$

Therefore g is a more general concept and is not tied to the Henyey-Greenstein function. For instance g is used in the similarity theory [61] that under some specific conditions allows usage of a scaled isotropic scattering instead of an anisotropic one, which can potentially be a useful optimization.

The last useful property of f_{HG} we would like to mention is the straightforward possibility to importance-sample it. The scattering angle θ proportional to the PDF defined by $f_{\text{HG}}(\theta, g)$ is obtained as

$$\cos \theta = \frac{1}{|2g|} \left(1 + g^2 - \left(\frac{1 - g^2}{1 - g + 2g\xi} \right)^2 \right) \quad (2.5)$$

where ξ is a uniformly distributed unit random variable.

2.2 Physics of light transport

Section 2.1 describes how participating media interact with light locally. Now we will introduce laws and principles that describe this behaviour on a global level.

Beer-Lambert-Bouguer law We already know how to quantify the mean free path of a photon in a participating medium. However, this does not tell us how much will a pencil of light be attenuated if it travels a certain distance in a medium. This attenuation is described by the Beer-Lambert-Bouguer law: let I be the intensity (without regarding its exact radiometric quantity for now) of a light pencil that enters a participating medium with extinction coefficient of σ_t . Its intensity I' after passing through the volume of this medium is then

$$I' = I \cdot e^{-\tau(l)} \quad \tau(l) = \int_l \sigma_t(s) ds \quad (2.6)$$

where l is the path of the light pencil through the medium. We need the function τ to account for extinction coefficient changes in non-homogeneous media. In homogeneous media it is simply defined as

$$\tau(l) = \sigma_t \cdot \ell(l) \quad (2.7)$$

where $\ell(l)$ is the length of the path l .

The reason why the Beer-Lambert-Bouguer law is exponential can intuitively be understood as follows. The process of attenuation of light in a medium by absorption and scattering is a continuous decay process in the limit case. As such it is naturally described by the exponential function, which describes all continuous growth and decay processes. So, if the attenuation rate of light in a homogeneous medium is 100% (i.e. $\sigma_t = 1 \text{m}^{-1}$), then the light travelling 1m long path in this medium will be attenuated by the factor of $e^{-\sigma_t} = \frac{1}{e}$. This is however exactly what we expect from a continuous unit decay process along a unit path, since

$$\sigma_t = 1 \Rightarrow \lim_{n \rightarrow \infty} \left(1 - \frac{\sigma_t}{n}\right)^n = \frac{1}{e}.$$

Woodcock tracking In algorithms that trace rays (or particles) through the volume of a participating medium, we generally need to perform two basic operations in each tracing step for a ray: first, we need to find the point where the ray interacts with the medium, and second (if the interaction was scattering), we must find the new scattering direction. The second task is usually solved by importance-sampling the medium phase function, as we already mentioned. As for the first one, we can utilize the principle of exponential extinction described by the Beer-Lambert-Bouguer law.

The simplest case are homogeneous media. Here the extinction is governed by the simple linear function defined by the Equation 2.7. Therefore to find the distance to the next extinction event d_{event} one simply evaluates the following equation:

$$d_{\text{event}} = -\frac{\ln \xi}{\sigma_t} \quad (2.8)$$

where σ_t is the extinction coefficient of the homogeneous medium and ξ is a unit random variable.

However, we generally have to deal with heterogeneous media. Here we must integrate along the examined light path to find the next attenuation event, since we do not know the medium composition along this path in advance. Therefore to find the distance d_{event} we must evaluate the implicit equation derived from the Equation 2.6:

$$\int_0^{d_{\text{event}}} \sigma_t(\vec{x}_0 + t \cdot \vec{\omega}) dt = -\ln(1 - \xi). \quad (2.9)$$

This equation has to be evaluated numerically. The most straightforward method is to use ray-marching; the algorithm steps through the medium (either using constant-sized steps or random steps to avoid aliasing) and accumulates the value of the τ function (which is basically the left side of the Equation 2.9). This is done until the accumulated value of τ exceeds the value on the right side of the Equation 2.9, or until the stepping does not leave the medium boundary.

Input:

$\vec{x}_0, \vec{\omega}$: ray with origin \vec{x}_0 and direction $\vec{\omega}$

σ_T : majorant extinction coefficient

(d_{\min}, d_{\max}) : interval of the ray to account for

rand(): unit random numbers generator

Result:

d_{event} : distance on the ray where the scattering event occurs

begin

$d_{\text{event}} = d_{\min} - \ln(1 - \text{rand}()) / \sigma_T$

while $d_{\text{event}} \leq d_{\max} \wedge \sigma_t(\vec{x}_0 + d_{\text{event}} \cdot \vec{\omega}) / \sigma_T < \text{rand}()$ **do**

$d_{\text{event}} = d_{\text{event}} - \ln(1 - \text{rand}()) / \sigma_T$

end

return d_{event}

end

Algorithm 1: The Woodcock tracking algorithm.

Ray-marching is a conceptually simple algorithm, but unfortunately it is biased; the bias of course arises from the choice of the step size. The better technique to use is Woodcock tracking (Algorithm 1) which was originally proposed by Woodcock et al. [60] for neutron tracing in nuclear reactions, and introduced into computer graphics by Raab et al. [49]. The idea behind Woodcock tracking is quite simple: first, the so-called *majorant extinction coefficient* σ_T is computed as the maximal extinction coefficient throughout the entire medium volume. Then, the algorithm performs randomized ray-marching through the medium, but treats it as a homogeneous medium with extinction coefficient σ_T . In each tracking step the algorithm then looks-up the real σ_t at the current position $\vec{x}_0 + d_{\text{event}} \cdot \vec{\omega}$ and accepts this event as a real extinction event with the probability $\sigma_t(\vec{x}_0 + d_{\text{event}} \cdot \vec{\omega}) / \sigma_T$; otherwise it discards the event as virtual and continues. After generating a real scattering event the decision if it is absorption or scattering is decided based on the medium albedo α . This method was proven by Coleman [10] to generate correct and unbiased mean free path values.

Volume rendering equation In the Introduction (Chapter 1) we briefly described the *surface rendering equation* (abbreviated RE), why it is generally hard to solve, and also what makes transition to the *volume rendering equation* (abbreviated VRE) even

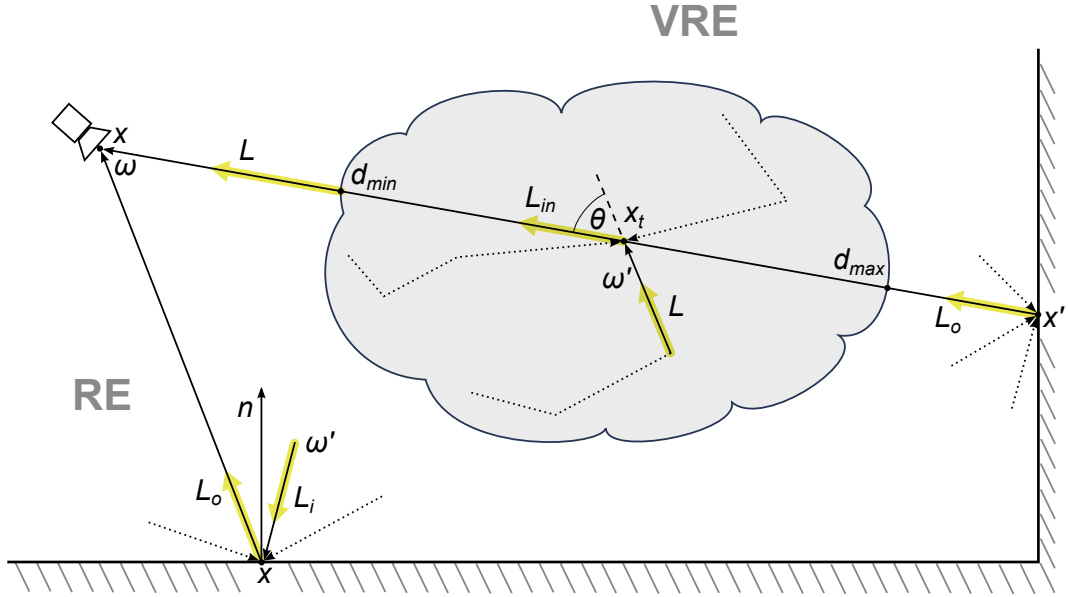


Figure 2.6: Scheme depicting the Equations 2.10 (the part marked ‘RE’), 2.11 and 2.12 (the part marked ‘VRE’).

more difficult. Having defined all we need previously in this chapter let us now look at these equations in more detail.

The surface rendering equation (more precisely, its hemispherical formulation) describes the radiance L_o leaving a surface at the point \vec{x} in the direction $\vec{\omega}$. The formulation is very similar to original Kajiya’s original formulation [35], except that Kajiya’s version describes integration over all surfaces visible from \vec{x} , while the hemispherical formulation integrates over the directions above \vec{x} :

$$L_o(\vec{x}, \vec{\omega}) = L_e(\vec{x}, \vec{\omega}) + \int_{\Omega_{2\pi}} r(\vec{x}, \vec{\omega}', \vec{\omega}) L_i(\vec{x}, \vec{\omega}') (-\vec{\omega}' \cdot \vec{n}) d\vec{\omega}'. \quad (2.10)$$

Here, $L_e(\vec{x}, \vec{\omega})$ is the emitted radiance from \vec{x} into $\vec{\omega}$ and $L_i(\vec{x}, \vec{\omega}')$ is the incident radiance at \vec{x} coming from $\vec{\omega}'$. Figure 2.6 depicts these relations. The function r is the *bi-directional reflectance distribution function* of the surface (abbreviated BRDF; the established notation of BRDF is ‘ f ’, but we will use ‘ r ’ to avoid confusion with the notation of phase function). In short, BRDF is a 4-dimensional function that expresses the fraction of light that a surface reflects; for a thorough descriptions of this function please refer to Dutré et al. [14] or Pharr and Humphreys [47], for instance.

The difficulty of the Equation 2.10 lies in its recursive nature. This is because $L_i(\vec{x}, \vec{\omega}') = L_o(\vec{x}', \vec{\omega}')$, where \vec{x}' is the nearest intersection of the ray $-\vec{\omega}'$ with the scene.

One of the largest limitations of RE is that it is built on the assumption that the scene consists of discrete surfaces separated by vacuum. This is of course almost never entirely true, although there are situations where this simplification does not produce any visible differences from the correct solution (for example the amount of air in most interiors is not sufficient for producing any scattering-related phenomena).

However, to account for the interactions described earlier in this chapter it is necessary to reformulate RE. The directional formulation of VRE defines the radiance L at any point \vec{x} in the scene from the direction $\vec{\omega}$ as

$$L(\vec{x}, \vec{\omega}) = \int_{d_{\min}}^{d_{\max}} T(d_{\min} \leftrightarrow t) \sigma_s(\vec{x}_t) L_{\text{in}}(\vec{x}_t, \vec{\omega}) dt + T(d_{\min} \leftrightarrow d_{\max}) L_o(\vec{x}', \vec{\omega}) \quad (2.11)$$

$$L_{\text{in}}(\vec{x}, \vec{\omega}) = \int_{\Omega_{4\pi}} f(\vec{x}, \vec{\omega}', \vec{\omega}) L(\vec{x}, \vec{\omega}') d\vec{\omega}' \quad (2.12)$$

$$T(t_1 \leftrightarrow t_2) = e^{-\tau(t_1 \leftrightarrow t_2)} \quad \tau(t_1 \leftrightarrow t_2) = \int_{t_1}^{t_2} \sigma_t(\vec{x}_t) dt \quad (2.13)$$

where d_{\min} and d_{\max} are the distances along the ray $\vec{\omega}$ to the nearest and farthest point in the participating medium, respectively. The notation of parametrized points is straightforward: $\vec{x}_u = \vec{x} - u \cdot \vec{\omega}$ where u is the distance parameter associated with the ray originating at \vec{x} with the direction $\vec{\omega}$ (the negative sign arises from the fact that we are moving along the negative side of $\vec{\omega}$). $L_o(\vec{x}', \vec{\omega})$ is the radiance leaving \vec{x}' in the direction $\vec{\omega}$ as defined by the Equation 2.10. $L_{\text{in}}(\vec{x}, \vec{\omega})$ is the in-scattered radiance at \vec{x} into the direction $\vec{\omega}$; it is the equivalent of the whole main integral in the Equation 2.10. See Figure 2.6 for a schematic depiction of the situation. T is the transmittance function — it is basically the Beer-Lambert-Bouguer law as defined in the Equation 2.6, just using a slightly changed notation. Notice also the additional positional parameter \vec{x} in the phase function call (Equation 2.12); it allows f to vary with position in the medium (for instance to incorporate local changes of the scattering anisotropy due to different particle composition).

Looking at the formulations of RE and VRE one can immediately see the similarities they share. In addition to the already mentioned equivalence of L_{in} with the hemispherical integral in the Equation 2.10, the BRDF in the Equation 2.10 and the phase function in the Equation 2.12 also basically have the same purpose (and except their different domains they are mathematically equivalent as well). Finally, same as RE, VRE is recursive as well — the calculation of the in-scattered radiance L_{in} requires evaluation of the radiance L from all directions, which again corresponds to the Equation 2.11.

On the other hand, the complexity of VRE is considerably higher. Looking at the Equation 2.10 it is obvious that each recursion level of RE adds evaluation of the 2-dimensional hemispherical integral over \vec{x} . At the same time, each level of recursion of VRE adds evaluation of a 3-dimensional integral (the linear integral in the Equation 2.11 plus the spherical integral in the Equation 2.12). In addition to that, calculation of each recursion level in VRE requires linear integration to obtain T (which would another integration dimension if not for independence of calculation of T and L_{in}). And finally, each recursion level also calls RE itself (from which we can see that if in the entire scene $\sigma_t = 0$, VRE collapses to RE). From all this we can see that the volumetric rendering problem is much more difficult than the rendering of surfaces, which is necessary to have in mind when designing an algorithmic solution to a particular instance of this problem.

As the last thing in this section we would like to include a different formulation of VRE, this time in its differential energy transport variant. This formulation will be more suitable for forward tracing algorithms, such as photon mapping. Since the behaviour of light in participating media can be regarded as a continuous process, we

can define the differential change of the radiance L of a pencil of light due to scattering and absorption as

$$\frac{dL(\vec{x}, \vec{\omega})}{d\vec{x}} = -\sigma_t L(\vec{x}, \vec{\omega}) + \sigma_s \int_{\Omega_{4\pi}} f(\vec{x}, \vec{\omega}', \vec{\omega}) L(\vec{x}, \vec{\omega}') d\vec{\omega}'. \quad (2.14)$$

The notation stays the same as in the directional formulation of VRE, and also the schematic depiction by Figure 2.6 remains suitable for this formulation.

2.3 Spherical harmonics

Spherical harmonic functions or in short *spherical harmonics* are a specialized mathematical concept that allows representation of a function by a series of coefficients and basis functions, similar to the Fourier transform. The difference between them is that the Fourier transform works with functions defined on \mathbb{R}^n , while spherical harmonics have 2-dimensional spherical domain.

Spherical harmonics have been described in many publications on mathematics and mathematical physics. In computer graphics spherical harmonics have been used for instance in works by Kautz et al. ([38, 53]) for arbitrary BRDF rendering in combination with environment mapping and for precomputed radiance transfer. In addition, several computer-graphics-oriented technical reports on this topic exist, for instance [21, 51, 52]. Therefore we will not delve into too much detail in this topic; we will just describe the basics necessary for our work and an interested reader can easily obtain additional details in the aforementioned sources.

Projection and reconstruction Let us assume we have defined an infinite series of orthonormal basis functions $B_1(x), B_2(x), \dots$ and a real function $F(x)$ we would like to approximate by functions B_1, \dots, B_n for some n . The function F can be approximated by \hat{F} as

$$\hat{F}(x) = \sum_{i=1}^n c_i \cdot B_i(x). \quad (2.15)$$

Also, if $n \rightarrow \infty \Rightarrow \hat{F} \rightarrow F$. We can represent the approximated function \hat{F} by storing the coefficients c_1, \dots, c_n , then evaluate it by computing the Equation 2.15 at x ; this process is called *reconstruction*. Of course, the more coefficients we store, the closer will \hat{F} be to F (in terms of the maximum-difference error).

The coefficients c_i are obtained by the inverse process called *projection* by integrating the product of F and the basis functions over the entire support of F :

$$c_i = \int F(x) B_i(x) dx. \quad (2.16)$$

Intuitively, the Equation 2.16 calculates how similar F is to a particular basis function.

Real spherical harmonics General spherical harmonics are complex 2-dimensional functions defined over a unit 3-dimensional spherical domain. Since we do not need their imaginary parts we will only be interested in their real versions. The real spherical

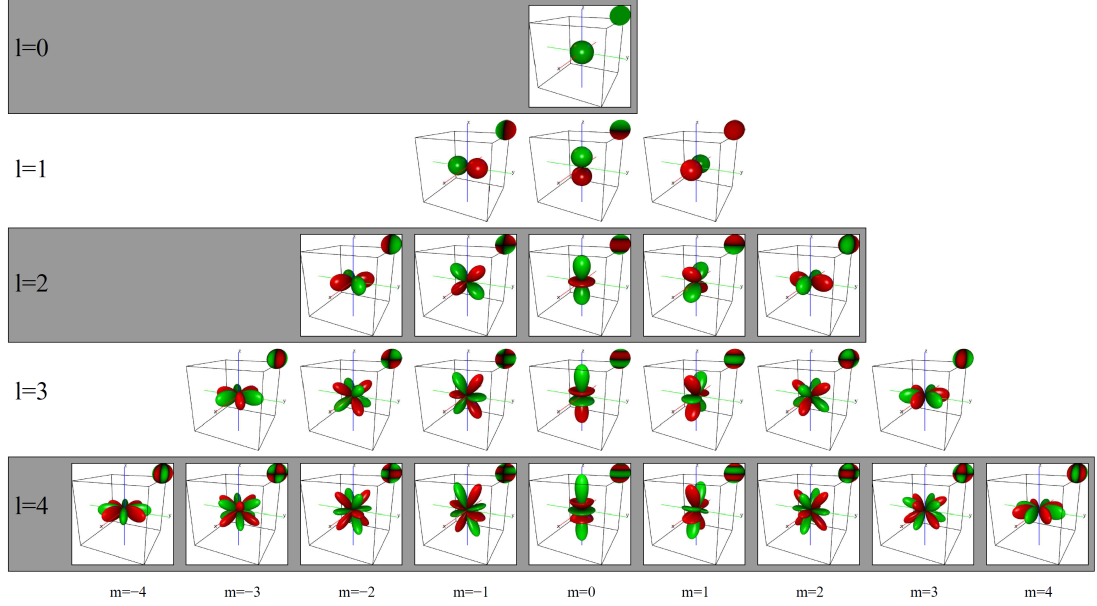


Figure 2.7: The first 5 bands of the real spherical harmonics. The positive values are indicated by green colour, while the negative ones are red. *Image from Schönefeld [51].*

harmonics $y_l^m(\theta, \phi)$ are defined as

$$y_l^m(\theta, \phi) = \begin{cases} \sqrt{2}N_l^m \cos(m\phi)P_l^m(\cos\theta) & m > 0 \\ \sqrt{2}N_l^{-m} \sin(-m\phi)P_l^{-m}(\cos\theta) & m < 0 \\ N_l^0 P_l^0(\cos\theta) & m = 0 \end{cases} \quad (2.17)$$

where

- $l \in \mathbb{N}_0$ is the band index and $-l \leq m \leq l$
- θ and ϕ are the zenith angle and the azimuth of the sample, respectively
- N_l^m are the corresponding normalization coefficients
- P_l^m are the associated Legendre polynomials

The real spherical harmonics for $l = 0, \dots, 4$ are shown in Figure 2.7. The definition of N_l^m and P_l^m can be found e.g. in [21], along with a numerically reliable evaluation scheme of the associated Legendre polynomials and a C++ implementations of the above defined.

Utilization of the real spherical harmonics is very similar to any other basis functions. We can define

$$y_i = y_l^m \\ i = l(l+1) + m$$

and then use the Equations 2.16 and 2.15 to represent our function of interest by putting $B_i = y_i$. The integration in the Equation 2.16 naturally becomes spherical, and is usually evaluated by numerical sampling. Calculation and storage of Λ^2 coefficients is required to approximate a function to the precision of the first Λ SH bands.

3. Related Work

Chapter 2 presented the theoretical concepts relevant for our work. This chapter provides an overview of some of the existing works from the field of computer graphics that concern themselves with simulation of global illumination effects in participating media, especially in clouds.

In general, these methods can be categorized as non-interactive and interactive. Non-interactive methods aim to generate high-quality images in the cost of high computational costs — it is not unusual that rendering of a single image takes minutes or even tens of minutes. On the other hand, interactive methods must be able to generate an image in hundreds of milliseconds or faster, so they can react on user’s input. For this they must always do some compromises in quality or complexity of the simulation they perform. In addition, interactive methods tend to be more specialized, so they can perform optimizations suitable for a certain situation or phenomenon they attempt to simulate.

In recent years, we can observe that non-interactive and interactive techniques for rendering global illumination on surfaces slowly converge. The first such method was *instant radiosity* introduced in 1997 by Alexander Keller [39]; since then the method received much attention in the rendering community. Also, attempts to make the photon mapping on surfaces interactive were made, for instance in works of Purcell et al. [48] or Airieau et al. [2].

Unfortunately in rendering of global illumination in participating media this convergence is much slower. The main reason is the complexity difference between RE and VRE described in Section 2.2. Another reason is that memory representation of participating media is much more costly than representation of surfaces, because of the higher dimensionality of volumetric data.

3.1 Photon mapping

Photon mapping (abbreviated PM) was introduced by Henrik Wann Jensen in 1996 in his PhD thesis [31]. Many papers have been written on this topic (for example [30, 12, 28, 24, 23]) and also PM has been summarized in Jensen’s book [32].

Photon mapping is a bidirectional ray-based algorithm which consists of two phases:

- **Photon tracing**

The radiative power of all light sources in the scene is equally distributed among n virtual particles called *photons*. These photons are then traced through the rendered scene using ray tracing. If a photon hits a non-specular surface its energy is stored in so-called *photon map*, which is a 3D data structure (most often a kD-tree) independent of the scene geometry.

- **Rendering**

The scene is then rendered using regular ray tracing. When a ray hits a non-specular surface the energy of photons in the vicinity of the hit point is assigned to the ray, contributing to the final colour of the pixel corresponding to the ray. There are two ways how to reconstruct the energy of photons near the hit point. The *histogram method* divides the photon map into bins; when a photon is being

stored into the map is assigned into one of the bins. During reconstruction, the energy of the nearest bins is interpolated, instead of treating individual photons separately. On the other hand the *density estimation method* finds the nearest photons directly (either using N-nearest-neighbours search, or it searches in a fixed radius around the hit point). Then it sums their intensities and divides this sum by the area through which the averaging has been performed, producing an estimate.

PM is a robust algorithm, working very well in most common settings. This can especially be said about its progressive improvements [24, 23]. It is also generally faster than most of the available alternatives, thanks to the caching mechanism of photon map. On the other hand PM is a (consistently) biased algorithm; there are several types of bias in PM, although practically all of them arise from the reconstruction stage.

3.2 Non-interactive methods

Not many non-interactive methods exist that would be specialized on a particular type of situation or phenomenon. This is most probably because the reason for development of specialized methods in general are possible optimizations applicable in a given situation. However, very rarely these optimizations improve the speed of a particular technique by an order of magnitude or more; such improvements are most often in the realm of tens of percent. As such, they are not a priority of non-interactive methods (e.g. it is not so critical if generation of an image takes 10 or 15 minutes). But since clouds are an especially difficult case even amongst participating media (as we will see in Chapter 4) some non-interactive methods for their simulation exist — the works of Gardner [19] and Nishita et al. [44] are examples of these.

As for the general available methods, most of them can be extended to take participating media into account. The extension mostly have to tackle the problem of substituting VRE instead of RE — how difficult this is depends of course on the particular algorithm.

Pure ray-based methods Practically all classical ray-based methods can be extended to handle participating media: ray tracing [36], path tracing [49], bidirectional path tracing[41], Metropolis light transport [46]. Since they already have the capability to evaluate (hemi)spherical integrals, the main necessary addition is the ability to sample interaction events with participating media, which is not so difficult to do.

On the other hand, their computational costs are very high, as already their original variants are rather slow. As for their robustness, they generally retain the properties of their original versions. For instance, path tracing is quite suitable for rendering media with low-to-moderate albedo and low scattering anisotropy, but in highly reflective and anisotropic media path tracing produces a lot of noise. Unfortunately, clouds are a textbook example of such medium, and therefore neither path tracing nor any of the remaining aforementioned methods are particularly suitable for their simulation.

Photon mapping The extension of PM to participating media has been described by Jensen and Christensen [33] in the general case and by Jensen et al. [34] for rendering of sub-surface light scattering. It has also been summarized in the Jensen's book [32].

An example of a GPU-accelerated technique for cloud rendering based on PM is the work of Cha et al. [8].

Photon mapping is a more suitable candidate for extension to participating media than the purely ray-based methods, since the increase of the problem dimensionality is not so troublesome for it thanks to the light caching mechanism. It is also more natural due to the photon map being natively a three-dimensional structure (although in practice a different map is used for surfaces than for volumetric media).

Extending PM to participating media does not only add the need to sample the mean free path of traced photons. The reconstruction step gets more complicated as well, since we have to account for all photons along the reconstruction ray, not only around its intersection point with the scene. The conventional way to do this is to choose a number of samples along the gathering ray and search for photons within spherical neighbourhoods around these points. This is not very accurate though: some photons might be missed, while some others can be counted twice. A much better volumetric photon gathering technique is the beam radiance estimate by Jarosz et al. [29]. This approach searches for photons in a cylindrical space around the gathering ray and produces results with much higher quality in shorter rendering times.

Volumetric PM can handle much wider range of participating media, mainly environments with high albedo are not so problematic for it, as for, say, path tracing. However, media with very strong scattering anisotropy can still cause problems, mainly if a low number of photons is used — the probability for the observer to be in the direction of the narrow phase function lobe quickly decreases with the increasing anisotropy. This can produce significant amounts of noise.

Other methods Radiance caching for participating media has been presented by Jarosz et al. [27]. The algorithm is not dissimilar to photon mapping, as it also utilizes a light power caching mechanism, as the name suggests. The difference is in the manner how this is done: radiance caching uses a set of caches computed explicitly by e.g. path tracing. The placement of these caches is guided by local gradients of the scene illumination, and the spherical radiance distribution in these caches is stored in the form of spherical harmonics. Outgoing radiance anywhere in the medium is then obtained by interpolation from the nearby cache points.

Volumetric radiance caching is actually the only technique capable of achieving results comparable to volumetric photon mapping that we know about.

Finite element methods have been used to render participating media as well. Conceptually they are equivalent to surface radiosity methods. Their adaptation for participating media divides the medium volume into discrete segments; the energy transport is then computed between all pairs of these segments, until energy equilibrium is reached.

Originally proposed by Chandrasekhar in 1960 [9] as the *discrete ordinates method* (DOM) this approach has been used for the simulation of general energy transport. Later already within computer graphics improvements have been proposed [20, 18]. Also, a very similar approach have been used by Haber et al. [22] for specialized simulation of radiative transfer in the Earth’s atmosphere.

In general, finite element methods are slower than ray-based methods, mainly because in environments with highly anisotropic scattering the discretization of the simulation space needs to be very fine. Hence they also usually consume more memory.

On the other hand they are usually more robust and exhibit less noise than ray-based methods (albeit sometimes in the cost of excessive interpolation and light smearing artefacts). However, the main advantage of these techniques is that once the energy equilibrium in the scene is computed, it stays valid until the scene configuration changes, which might be very beneficial in some usage scenarios.

3.3 Interactive methods

In contrast to non-interactive methods where there are only few specialized methods for cloud rendering, among interactive and real-time techniques the specialized ones dominate, for the reasons mentioned above. In this section we will therefore concern ourselves solely with the specialized methods for cloud rendering. And since interactive methods use a wider variety of cloud representations than the non-interactive ones (where participating media are almost always represented by 3D density fields) we will categorize them according to this criterion.

Billboard-based techniques Billboards are simple textured camera-facing quads which are widely used in real-time rendering systems for representation of various, mostly semi-transparent, particles. Naturally, there are numerous techniques that use them for representation of clouds.

One of the classical billboard-based real-time techniques is the work of Wang [58]. The technique is suitable for game studios workflows — an artist creates the basic shape of the cloud by hand in a modelling software, then the algorithm randomly fills this shape with cloud particles. The cloud illumination is purely empirical — the colour of the particles is guided by colour palettes and by an *ad hoc* lighting model. The technique has been used in the Microsoft Flight Simulator 2004 game. Numerous approaches similar to this exist, such as the work of Wenzel [59], which has been used in all versions of Cryengine.

A much more theoretically sound work by Szirmay-Kalos et al. [55] describes the so-called *illumination networks*. The idea of this work is to perform a GPU-accelerated iterative lighting simulation on the cloud particles represented by billboards. The core of the technique is a Monte-Carlo approach that uses a set of fixed directions to transfer the light energy in the cloud. Each simulation step takes the current energy state of the scene and ‘pushes’ this energy into these fixed directions. Finally, after a sufficient amount of iterations the cloud energy state converges to its equilibrium. However, the main drawback of such approach is that after a change of the lighting conditions in the scene or of the cloud shape the entire simulation has to be re-run from the beginning, which is unfortunately very impractical for the potential target applications (such as computer games).

Slice-based techniques Slice-based approaches use a set of parallel slices of geometry to sample the scattering medium volume. These slices can be axis-aligned, view-direction-aligned, or then can have different alignment for some specific reasons. In either way they are textured by fetching density and transparency values from the medium volume (mostly represented by a 3D texture on the graphics hardware). Finally, they are rendered by utilizing the standard alpha-blending functionality of GPUs.

Slice-based approaches originate from volumetric visualization techniques, where the auxiliary geometry is used to visualize a volumetric dataset after application of a transfer function. As such there is no intrinsic way to compute energy transfer on such medium representation. This was changed by the work of Kniss et al. [40] who presented their method called *half angle slicing*. An application of this method to cloud rendering was published by Riley et al. [50].

The method, as the name suggests, aligns the auxiliary slicing geometry with the half vector between view and light directions in the scene. The algorithm then runs in two passes: in the first pass it calculates forward light scattering between adjacent geometry slices, starting at the one closest to the light source. In the second pass the slices are rendered in front-to-back order from the camera position, accumulating the values of the transmittance function and of course of the cloud colour.

Despite being an *ad hoc* approach (it is centred around the capabilities of contemporary GPUs, not around what a proper light simulation in participating media requires) it produces nicely looking results in situations where forward scattering dominates. However, the main drawback of the method is it supports illumination by only a single light (because of the alignment requirement). Because of this limitation Riley et al. in their extension to clouds [50] had to design an approximate model for taking sky light into account (otherwise the results would look too artificially and isolated).

Other techniques The remaining techniques work directly on the volumetric medium dataset, or they use a hybrid representation that cannot be simply categorized.

A hybrid method that utilizes a combination of a geometrical mesh and a deformation hypertexture for cloud representation was presented by Bouthors et al. [6] (as a continuation of some of the authors' previous work [5]). The illumination simulation relies on a huge dataset containing an analysis of multiple scattering behaviour in plane-parallel volumetric slabs. This dataset has then been compressed into a set of empirical functions by least square optimization. During rendering, these functions are used to find the most probable light paths for a particular point on the rendered cloud. Although this method produces very impressively-looking results we find some of its assumptions weak and also some of the algorithm steps not very transparent.

Just recently, a method by Engelhardt et al. [17] was published. Their work extends the aforementioned instant radiosity method [39] to participating media. The idea behind instant radiosity is that global illumination is approximated by tracing so-called *virtual point lights* (VPLs), which are shot from the light sources, through the scene (similar to photon tracing). These VPLs are then stored on surfaces in the scene and are used to calculate local illumination at a shaded surface point. The extension to participating media is quite intuitive — similar to volumetric photon mapping the VPLs are also stored anywhere in the scene, not only on surfaces. Ray marching is used for their visualization — but at each sample only a randomly chosen subset of VPLs is used to compute the light contribution, since evaluation of all VPLs would be very costly.

The main contribution of this work is, however, an optimized bias compensation: all instant radiosity methods suffer from clamping bias, which arises from the fact that light contribution from VPLs must sometimes be clamped to avoid oversaturated singularities near the locations where VPLs are stored. Engelhardt et al. propose an

approximate bias compensation which enables the method to run in interactive frame-rates (while normally computation of the compensation is quite costly). And in spite of being approximative, the technique produces very nice-looking results which are in most cases visually indistinguishable from reference path tracing solutions.

Another interesting approach was presented by Kaplanyan and Dachsbacher [37]. It is a finite elements method similar to the works described in Section 3.2. However, by using approximative solutions for some of the simulation steps, using a low-resolution simulation grid with only a few iterations per frame, and implementing the core of the method on GPU the authors were able to achieve very good performance. The method has been used for approximative solutions of higher-order global illumination on both surfaces and volumes in the CryENGINE[®] real-time game engine, which has been used for example in the Crysis 2 game.

4. Method overview

This chapter takes a more detailed look at clouds specifically, discussing their properties and hard points. We will also formulate the assumptions we built our approach on. Then we will finally introduce our method by presenting its workflow and briefly discussing each of its steps.

4.1 Assumptions and rationale

So far we have described the physics of participating media and energy transport in them in general. Let us now look at some specific properties of clouds in more detail, so we can identify the difficult ones, but also determine which of them can be used to our advantage when designing our method.

Albedo and density Clouds in most cases have practically 100% albedo, meaning they exhibit very little absorption (except when soaked with large amounts of rainwater, such as nimbostratus clouds — see Figure 4.2 (h)). Effectively this means that if a photon enters a cloud, it will be scattering inside until it flies out. Moreover, clouds are relatively dense. Their absolute density is not so high, but after taking into account their spatial extent, the situation changes. Let us take an example: according to Bouthors et al. [6] a typical scattering coefficient in cumuli is $\sigma_s = 0.05\text{m}^{-1}$; this corresponds to a mean free path of 20m. In a cloud with the diameter of 1km a photon will scatter several tens of times before leaving the cloud. In reality this number can go even to hundreds scattering events per photon. This is indeed a lot — from the programmatic point of view it means that we must simulate very high numbers of scattering events, without a possibility the simulated photon will get absorbed, thus saving us some computational effort.

Scattering anisotropy Since clouds mostly consist of tiny water droplets (which are however much larger than the wavelengths of visible light), the scattering in clouds is strongly forward and therefore highly anisotropic. This in terms of the corresponding phase function causes it to have a major forward lobe, preferring very small scattering angles.

A typical cloud phase function is shown in Figure 4.1. The rigorous Mie phase function has a number of interesting features, but since the plot is in logarithmic scale, the real energy they carry is about 4–5 orders of magnitude smaller than the energy represented by the main forward lobe. As such they will not be a main point of interest of us, and therefore we will use the Henyey-Greenstein approximation instead, as it approximates the main forward lobe fairly well.

However, such strong forward scattering causes some complications on its own. It allows light to penetrate deeper into the cloud volume; if clouds would scatter light isotropically, a part of the incident light would be scattered back, leaving the cloud sooner and thus decreasing computational costs of the simulation.

Another complications arise when strong forward scattering is combined with such high average number of orders as we are dealing here with. The effect of this is that in spite of the local scattering being strongly forward, the overall scattering behaviour becomes more and more isotropic with the increasing number

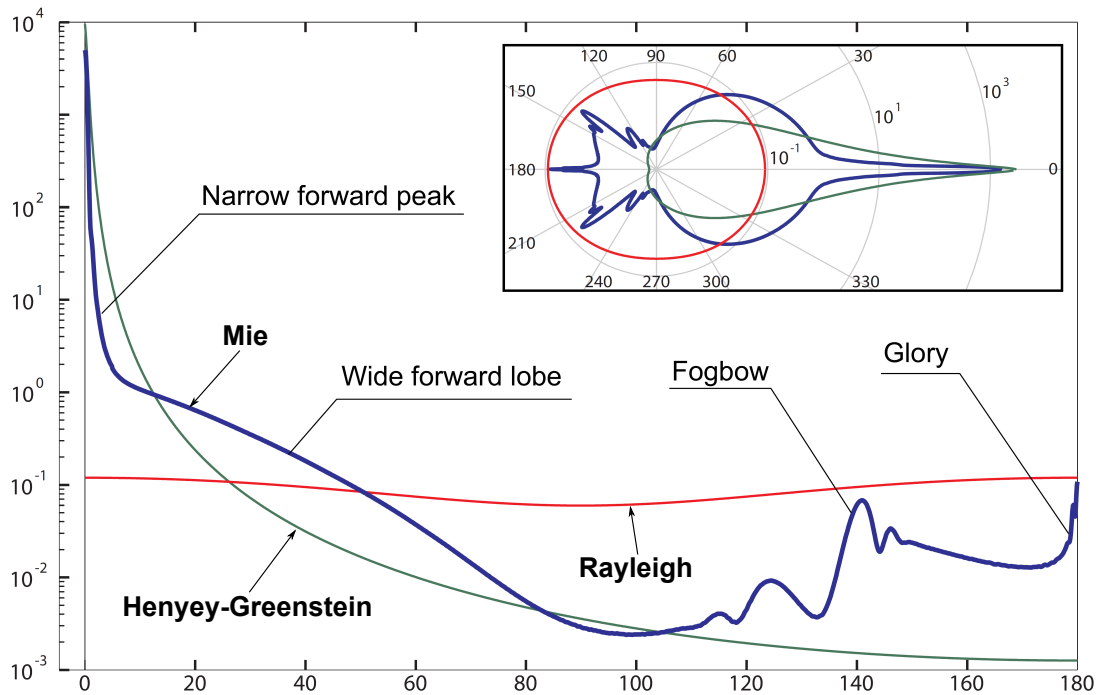


Figure 4.1: Example of a phase function typical for cumulus clouds (labelled ‘Mie’). The other two are the Heney-Greenstein approximation for $g = 0.99$ and the Rayleigh phase function (see Section 2.1). Notice the \log_{10} -scale. *Image from Bouthors et al. [6].*

of bounces a photon undergoes in the cloud (the resulting angular distribution in fact corresponds to repeated convolution of the cloud phase function with itself, see e.g. [50]). This causes the very common effect that while the cloud edges are very bright when looking against the Sun (colloquially called ‘silver lining’), the inner and rear parts of the cloud look isotropic and uniform. That is because the photons that reached these parts have almost random directions after such high amount of scattering events they have experienced (and also because the light that enters the cloud directly from the Sun is actually weaker here, than the relatively isotropic light that originates in atmospheric scattering). This effect is nicely visible on the photographs (a), (b), (c) and (d) in Figure 4.2 (for instance, the upper-right cloud in picture (d) is shadowed by another one not visible on the photograph. It can be observed that while the directly lit clouds behind it have much sharper lighting, the shadowed cloud has more isotropic appearance with less contrast mainly on its top edge).

Again, this is a problematic behaviour to simulate. The high amount of average scattering orders does not allow us to use simplifying approaches, such as considering only single- or double-scattering. On the other hand, since the scattering is so anisotropic, we cannot use approximations for very dense media — for instance, according to Jensen et al. [34], their diffusion approximation does not work well in combination with strong forward scattering.

Morphology ¹ Finally, clouds also can have a wide variety of complex shapes (see for example pictures (e), (f) and (g) in Figure 4.2). This makes their representation in memory difficult — each cloud type would require a different representation

¹By ‘morphology’ we mean the spatial distribution of a cloud mass in a particular point in time.

that suits it most (e.g. the hybrid approach by Bouthors et al. [6] can handle only very compact cloud types, such as cumuli). Hence in general we have to use 3D density fields to represent clouds, which is of course very memory-intensive. That in consequence does not allow us to effectively work with very large clouds without a significant loss of details.

The above analysis indicates that if we do not want to resort to completely empirical models, it is really necessary to stick to the rigorous physical and mathematical concepts, or at least their sufficiently close approximations. Therefore, we can only do compromises on the higher, computational level. Let us now look at some properties of clouds that would allow us to do so.

Illumination In rendering, arguably the most influential aspect that any algorithmic designing is centred around is lighting. In our case we have two options:

- During the day the main light source is without any doubt the Sun. The secondary source is the ambient light scattered in the atmosphere, but that directly depends on the Sun as well. This is very well visible in Figure 4.2 (e) — the right side of the cloud is illuminated by direct reddish sunlight, while the rest is blueish because of the atmospheric scattered light. No other light sources come into consideration, at least under any normal circumstances.
- At night there are few possible sources: the Moon (again plus the (negligible amount of) light originating in the atmospheric scattering) and in urbanized areas the illumination from street lights. There may be others, for instance an aeroplane signal lights, but we can safely consider these comparatively rare.

We are not by any means trying to convey an opinion that the clouds are illuminated by a single light source (as many interactive methods tend to assume). Our point is that, except some rare occasions, clouds are naturally illuminated by low-frequency light sources, both in spatial and in temporal sense. In other words, all of the described sources move very slowly or not at all, and neither of them illuminates clouds locally.

This stays valid for occlusion as well — clouds are practically always occluded only by other clouds, which never move very rapidly across the sky (relatively to their size).

This finding is essential for our method, as we will describe later. We cannot assume completely stationary light sources, which excludes any kind of static illumination precomputation scheme. However, it allows us to safely assume that no abrupt change of lighting conditions will occur, which in turn will permit us to consider some tolerance on the illumination update speed.

Morphology An observation very similar to the previous one can be made for clouds morphology. Again, we cannot assume the cloud shape remains static. However, convection responsible for morphology changes in all clouds is a relatively slow process; translating this into the language of interactive rendering, the cloud shape does not change every frame, or at least not so much it would completely invalidate the previous light simulation results.

These two observations are at the same time the main assumptions of our method design. They lead us to the first design decision, which is usage of photon mapping as the core algorithm for illumination simulation.

Photon mapping has been built on the observation that in order for a rendering algorithm to be robust and efficient it needs to be bidirectional [32]. That means it has to consider both the lighting and the observer in a scene, and it has to be able to effectively establish a connection between them, allowing transfer of light energy from the light sources to the observer. PM itself does this in the two passes described in Section 3.1.

Bidirectionality is especially important in scenes with a lot of glossy surfaces and anisotropically-scattering participating media, which is exactly our case. This is because in such situations the purely forward or backward algorithms have problems reaching their target scene elements — the backward algorithms based on the recursive evaluation of the Equations 2.10 and 2.11 (such as path tracing and stochastic ray tracing) have in some cases problem to reach the light sources in the scene, and *vice versa*.

This makes PM suitable for the rendering of clouds, but it is still not the reason why it also benefits from the observations we made. This reason is the light caching property of photon mapping. In the first pass PM creates a photon map of the simulated scene; this photon map represents the light energy distribution for a particular scene state. The scene state contains positions and specific properties of the objects in the scene, so if any object or light source in the scene moves, the photon map has to be built anew. However, position of the observer is not a part of the scene state, so if the observer moves only the second pass needs to be repeated.

The **main design premise** of our approach method the above observations of slowly-changing scene state and the light caching feature of photon mapping. The latter allows us to spread the computation of the cloud photon map across multiple frames, so only a fraction of all photons needs to be traced in each frame. We call these portions of photons *photon generations* and label them G_i . Each generation contains N_G photons and there is a fixed number of p generations, which means the total number of photons in the photon map is simply $N_T = p \cdot N_G$. Therefore the total flux of the photons which will constitute the photon map Φ_T is equally divided between these generations.

The point of this approach emerges when we look at the observations we have made. We start with an empty photon map M and a scene in some state, and in each frame we add the generation G_i (where $i = 0, \dots, p - 1$) containing N_G photons into M (by tracing them through the scene). After p frames M is updated and so is the illumination in the scene. Now if the scene state changes (e.g. the Sun moves), we should correctly discard M and build it from scratch. What we do instead is that we discard only the oldest generation G_0 , removing the photons associated with it from M , and add a new generation G_p into M by tracing its photons through the scene in the new state. The next frame removes G_1 from M and adds G_{p+1} into it, and so on.

In this approach M behaves as a queue which contains the p most recent generations of photons. Naturally, some of these generations correspond to different (older) scene states than the current one. But thanks to the assumptions of slowly changing scene state the deviation of the energy state of the scene represented by M will be small and ideally unnoticeable.

This approach has multiple advantages. First, it allows us to distribute the simula-

tion computational costs between multiple frames. Second, a change of the scene state does not invalidate the entire photon map, it just deprecates it, initiating an iterative update. Third, our approach has a very good temporal coherence (which is often a problem with this class of methods). This all means that even though the computation of the entire photon map is hardly an interactive operation, spreading it across multiple frames can possibly make the per-frame update interactive, and if the assumptions will hold the resulting cloud illumination will be indistinguishable from the correct result.

Naturally, this is hardly a new idea. Similar progressive concepts such as *adaptive spatio-temporal sampling* and *frameless rendering* have been investigated before [3, 43, 45, 11], quite recently even in combination with photon mapping [2]. We are however not aware of any method that would apply these ideas to the rendering of participating media or clouds specifically.

4.2 Brief algorithm description

Section 4.1 described the specific conditions we have to deal with in cloud rendering, and the assumptions we derived from them. This section builds on those assumptions and presents the main algorithmic steps of our method, to create a coherent picture of the method without going too much into details. These steps will then be described in Chapters 5 and 6 from an abstract and an implementation point of view, respectively.

Cloud representation We use a hybrid approach to represent clouds in our method. It utilizes both 3D scalar density fields and billboards, and combines them to take advantages of both of them.

We start with a 3D cloud density field — it can be modelled by an artist, or it might be a result of an actual convection simulation. In either way we do not want to impose any precomputation requirements, so the cloud shape can even be simulated interactively. How to do this is beyond the scope of this work (please refer for instance to [13, 6] for an overview of physically-based techniques for cloud animation). The only requirement is that the cloud density data reside in GPU memory, so they are accessible to the simulation.

The primary purpose of the density data is the illumination computation. Photons are traced through these volumetric data using the techniques described in Chapter 2. However, we do not directly use the density field to visualize the cloud they represent; instead, we generate a set of billboards based on the data, and these billboards are then used to render the cloud.

The reasons we use billboards to visualize clouds are two-fold. First, they map well to the common design of most contemporary interactive rendering engines. Second, they provide a way to discretize the cloud volume, which we will need later in the algorithm.

Basic algorithm steps The work of our algorithm in the temporal sense is centred around two kinds of frames with different granularities: **animation frame** and **image frame**. An animation frame is a time period where the algorithm works with the same density data. An image frame is what is commonly referred to as ‘frame’ in interactive methods — a single rendered image. Naturally, animation frames change less often

than image frames — this is a common technique to decrease computational intensity of the cloud animation by amortizing the computation of the next animation frame across several image frames. In addition, it is fully in agreement with our assumption of slowly changing cloud morphology.

The workflow of the algorithm within a single **animation frame** consists of the following steps:

1. Preparation

The billboards for the cloud visualization are generated from the input cloud density field $V_\rho = V(\rho)$. Other necessary data structures are prepared as well. This phase is done just once per animation frame.

(a) Gradient computation (Section 5.1)

Based on V_ρ the algorithm computes the corresponding gradient magnitude field $V_g = V(|\nabla\rho|)$ using finite differences method. V_g has the same resolution as V_ρ . Alternatively, it is possible to use the local variance field of V_ρ (within some radius), if it produces better results in the subsequent steps.

(b) Cloud sampling (Section 5.2)

By means of rejection sampling a set of 3D point samples is generated with the distribution given by V_g . The corresponding values of V_ρ at these points are stored with them.

(c) kD-tree building (Section 5.3)

A three-dimensional kD-tree is built around the point samples generated in Step 1b. The spatial median criterion is used to separate the samples.

(d) Per-leaf peak densities computation (Section 5.4)

For each leaf of the kD-tree the maximal value of V_ρ is found. These values correspond to local majorant coefficients σ_T within each of the respective volumes spanned by the kD-tree leaves.

(e) VPCs and billboards generation (Section 5.5)

Inside each of the kD-tree leaves with $\sigma_T > 0$ the algorithm places a new billboard and its corresponding *virtual photon collector*, VPC. These VPCs serve as imaginary bins to which scattered photons are assigned in Step 2a.

(f) Photon map initialization (Section 5.6)

A circular queue representing the cloud photon map M is created. Members of this queue are linear arrays of spherical harmonics coefficients, which are used to represent the spherical illumination distribution in each of the VPCs without the need to explicitly store traced photons. The queue holds p such arrays to represent the entire update cycle of M .

2. Rendering

After Phase 1 is finished, the algorithm starts periodical rendering of **image frames** from the prepared data, until a new animation frame is started.

(a) Photon tracing (Section 5.7)

At the beginning of each image frame, N_G photons are traced through the cloud and compressed into the next free array of SH coefficients in M . This is done for the current scene state, which consists of the currently active animation frame and the positions of all light sources.

(b) **Illumination reconstruction** (Section 5.8)

M now contains the p most recent photon generations represented by the arrays of SH coefficients. More precisely, each of the arrays contains a set of Λ^2 SH coefficients for each cloud VPC (where Λ is the number of used SH bands). Only at this point the observer position is taken into account — for each VPC its illumination colour is reconstructed precisely from this position. This is done by summing the corresponding per-VPC SH coefficients and evaluating the Equation 2.15.

(c) **Cloud visualization** (Section 5.9)

After reconstructing the directional illumination colour for each VPC, these colours are assigned to the corresponding billboards, which are then sent to GPU for rasterization.

The biggest drawback of the presented method is currently its inability to make fluent and efficient transition from one animation frame into another. The only possibility to to this at the moment is a rough equivalent of the double buffering technique: while rendering the current animation frame, the preparatory Phase 1 has to be computed for the next animation frame in background while still rendering the current animation frame. When the next animation frame is prepared the transition between them can be performed by weighted blending spread across several image frames. Please refer to Section 7.2 for additional details.



(a)



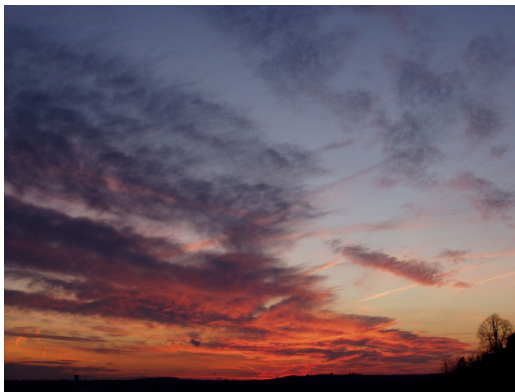
(b)



(c)



(d)



(e)



(f)



(g)



(h)

Figure 4.2: Real cloud photographs.

5. Detailed method description

This chapter provides a detailed breakdown of the algorithm outlined in Section 4.2; we will also be referring directly to its steps. Each of the steps is thoroughly described, as well as interconnections between them. The section names are prefixed by either ‘(P)’ or ‘(R)’, depending on if they belong to the Preparation or the Rendering phase, respectively.

5.1 (P) Gradient computation

In this step the algorithm calculates the local gradient magnitude field V_g from the input density field V_ρ ¹. V_g , as well as V_ρ , is stored in GPU memory and has the same resolution as V_ρ .

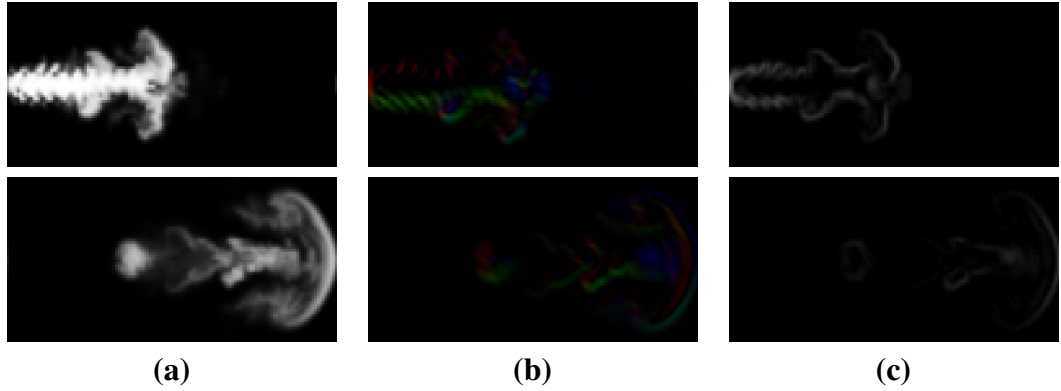


Figure 5.1: Two slices of the density and gradient fields corresponding to the smoke cloud dataset from Figures 2.1 and 2.2. Column (b) shows the colour-coded gradient vectors and column (c) the corresponding normalized local variances of the density field from column (a).

Gradient of a function of three variables $h(x, y, z)$ is a vector of its first partial derivatives:

$$\nabla h = \left(\frac{\partial h}{\partial x}, \frac{\partial h}{\partial y}, \frac{\partial h}{\partial z} \right).$$

Using central finite differences method it can be approximated as

$$\nabla h \approx \begin{pmatrix} \frac{h(x + \varepsilon, y, z) + h(x - \varepsilon, y, z)}{2}, \\ \frac{h(x, y + \varepsilon, z) + h(x, y - \varepsilon, z)}{2}, \\ \frac{h(x, y, z + \varepsilon) + h(x, y, z - \varepsilon)}{2} \end{pmatrix}, \quad (5.1)$$

¹ V_g can alternatively contain local variance values of ρ within some fixed radii around the sample points. The decision which option to choose depends on the subjective quality of the results produced by both choices in the subsequent steps

where ε is a small constant that defines the approximation radius. Its choice depends on the frequency of h — generally, the higher frequencies h contains, the smaller should the value of ε be.

The results of application of the Equation 5.1 on a density field are shown in Figure 5.1. Note that since $\nabla\rho$ detects the locations where the density function ρ changes rapidly, it behaves as an edge detector. For instance, the density field in the first row of Figure 5.1 is homogeneous from inside, so the gradient (and also the local variance) values are very small there. As we will explain later this is exactly the desired result of this algorithm step.

The second operation we perform during this step is finding the global suprema of V_ρ and V_g . We will need these values in the subsequent steps.

5.2 (P) Cloud volume sampling

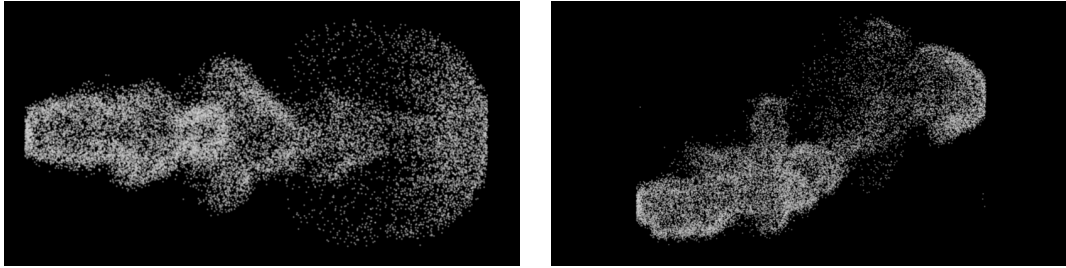


Figure 5.2: Side and top views of the point-sampled smoke cloud dataset using 20000 samples.

Next, we need to generate a set of point samples with the distribution of the gradient magnitude field V_g produced by Step 1a. For this we regard V_g as probability distribution function and use rejection sampling to produce the point samples.

Note that rejection sampling does not need the sampled function to be a proper probability density function with unit integral. The only requirement is that the values of the sampled function and the generated random numbers used for the sampling are from the same interval. We ensure this by normalizing the values of V_g by dividing them by their supremum obtained in the previous step. Although we could normalize V_g to be a proper probability density function (by dividing its values by the integral over V_g), the only effect this would have would be radically decreased efficiency of the rejection sampling in most cases.

In addition to generating the samples themselves we also store the normalized values of V_ρ at their respective positions with them. These values will be utilized during the kD-tree building in Step 1c.

5.3 (P) kD-tree construction

In this step the algorithm builds a three-dimensional kD-tree from the point samples set generated by Step 1b. There are three reasons why we do this:

1. Placement of cloud VPCs and billboards distributed according to the cloud gradient magnitude field V_g .

2. Creation of a data structure to locate nearest VPCs during the photon storing operations in Step 2a.
3. Creation of a partitioning structure to separate locations in cloud with similar values of V_ρ .

VPCs and billboards placement If we would need to perform this step only, we could simply use the locations generated by Step 1b (and use low discrepancy series to generate them to avoid clustering). However, because of Reasons 2 and 3, we cannot do that directly.

Our motivation for billboards (and VPCs, since there is a 1:1 correspondence between them) to be distributed according to V_g , and not V_ρ (as one could intuitively expect) is the following: naturally, the distribution of matter in clouds is normally higher at cloud cores and decreases towards its outer parts. Distributing billboards and VPCs according to a cloud V_ρ would mean that the majority of them would be placed at the dense cloud core. However, this is not optimal at all — what we really need is to distribute them according to the gradient of the cloud illumination, so we can more finely sample the areas where it changes more dynamically.

Of course, we do not know in advance how a cloud will be illuminated. What we know, however, is that in general the illumination in a cloud will vary much more at its edges (Figure 4.2 (a)–(d) illustrates this very well). Hence we want to place more samples at the cloud edges, which is exactly what sampling of the cloud gradient magnitude field will produce. Figure 5.2 shows an example of such sampling.

Locating the nearest VPC kD-tree is a common and effective data structure for spatial nearest-neighbour searching (at least in lower-dimensional spaces). Moreover, our situation will be even simpler than the general NN-search case — as will be described in Section 5.5, we place VPCs and their associated billboards into the centres of the constructed kD-tree leaves. That means finding the VPC which is closest to a particular location in space corresponds to finding the tree leaf containing the location.

Separation of similar-density regions The Woodcock tracking algorithm (Algorithm 1 in Section 2.2), which we use for mean free path sampling, needs the majorant extinction coefficient σ_T for its work. Naturally, σ_T corresponds to the supremum of V_ρ , except for scaling by a cloud particle cross-section.

However, Yue et al. [62] in their work shown that Woodcock tracking has very high performance penalty in highly heterogeneous media. This is because the algorithm treats the simulated medium as homogeneous with a uniform extinction coefficient σ_T (which is the reason why the algorithm is unbiased). In participating media where most of the medium has significantly smaller extinction coefficient than σ_T on average, most of the events generated by the algorithm are rejected as virtual, which can severely reduce performance of the tracking procedure.

To solve this problem the authors proposed to partition the medium volume onto regions with small variability of the medium density using kD-tree, and then use the local majorant extinction coefficient in each of these regions to perform Woodcock tracking. This effectively minimizes the amount of virtual events, while still keeping the algorithm unbiased. When properly formulated this problem transforms to finding of the largest empty hyper-rectangle [1] in the density function ρ .

Although this approach is able to decrease the rendering costs significantly, the time needed to construct the partitioning kD-tree is in order of seconds at least, which is of course unacceptable for interactive applications.

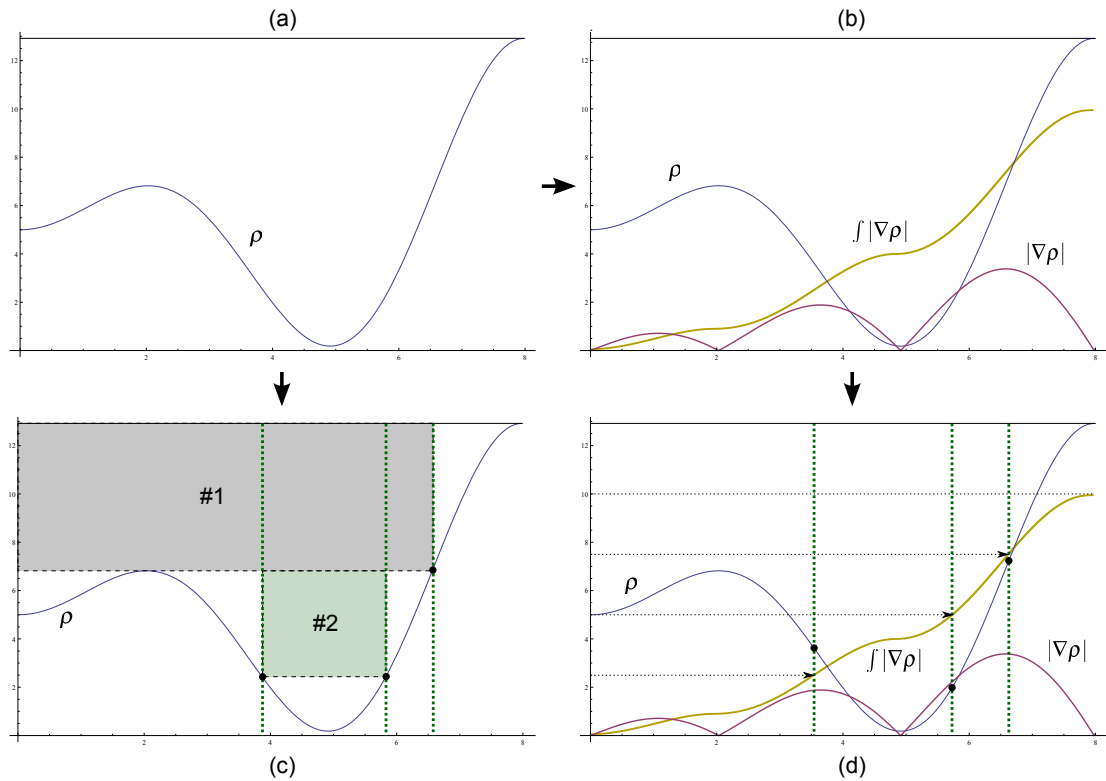


Figure 5.3: Our approximate solution of the largest empty rectangle partitioning in 1D. Image (a) shows an example of a density function defined analytically as $\rho = x \sin(x) + 5$. Image (c) shows the partitioning after 2 iterations of the largest empty rectangle search, yielding three splitting planes (thick vertical dashed lines). Our approach is shown in images (b) and (d); image (b) contains plots of the gradient magnitude function $|\nabla\rho|$ and its cumulative function $\int |\nabla\rho|$. Image (d) shows that the splitting planes are obtained by recursively sampling the function $(\int |\nabla\rho|)^{-1}$, always in halves of the sampling intervals — for three splitting planes this means sampling at 50%, 25% and 75% of the original sampling interval (which is $\langle 0, 10 \rangle$ in the example). Of course, we do not directly sample $(\int |\nabla\rho|)^{-1}$ in our method (in higher than one-dimensional space this would not even make sense); instead, we generate many samples of $|\nabla\rho|$ in Step 1b and during the kD-tree build we always divide the current samples in halves using the spatial median criterion on the dimension where the discrepancy of density values is currently highest. Comparing images (c) and (d) it is apparent that our proposed approach produces splitting planes which are very close to the original largest empty rectangle solution.

Our solution to this issue lies in the observation that partitioning the space according to $|\nabla\rho|$ yields very similar partitioning structure as partitioning according to the largest empty rectangles. This is depicted in Figure 5.3. Intuitively, the largest empty rectangle partitioning creates regions with the lowest possible range of density values, which is desired, since the space ‘above’ the graph of ρ (see Figure 5.3 (a)) represents the rejected virtual extinction events. However, the partitioning according to sampled $|\nabla\rho|$ encloses the regions where $|\nabla\rho|$ is large and therefore creates regions with low variance of ρ as a side effect. In the end, this is precisely the desired behaviour.

As a result, the time needed to build the kD-tree by our approach is at least 1–2 orders of magnitude smaller than the times Yue et al. report for similar scenes. In

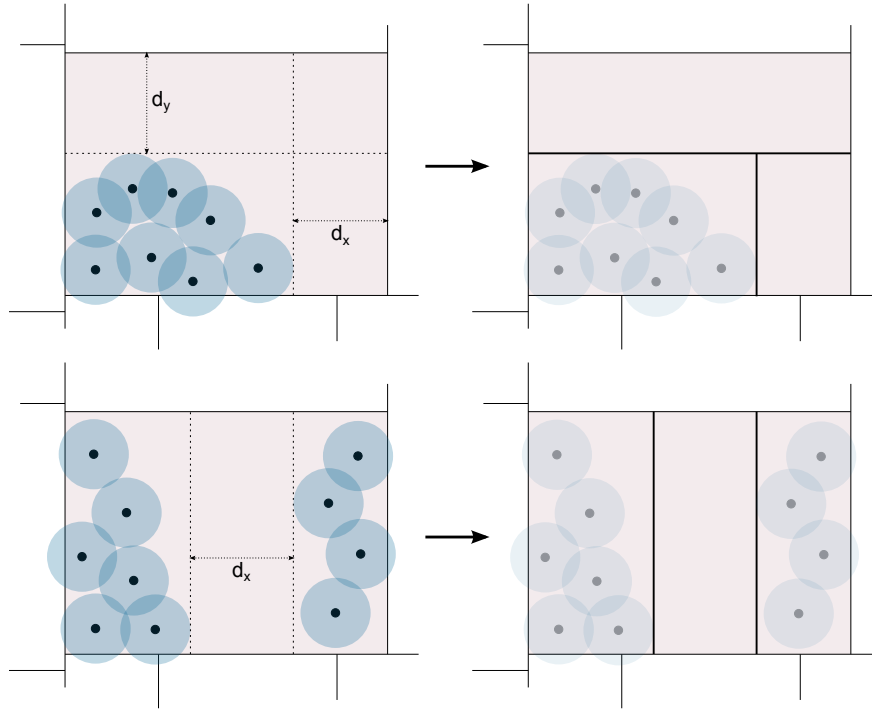


Figure 5.4: Finalization of a kD-tree node — in case there is too much leftover space the algorithm culls it to better encapsulate the point samples.

addition, although our method is only an approximation, it is more correct in certain sense. The reason why we think this is that Yue et al. don't solve the problem of the 4-dimensional largest empty hyper-rectangle, since this is infeasible even for non-interactive applications. Instead, they solve the 2D largest empty rectangle problem for every axis separately (at each kD-tree subdivision step) and choose the best of the three solutions. On the other hand, our approach, despite being approximative, works natively in 3D (because the samples of $|\nabla\rho|$ are three-dimensional).

As for the building of the kD-tree itself, we use a custom building procedure working in $O(n \log n)$ time, very similar to the standard algorithm by Wald and Havran [57]. First, the algorithm sorts the gradient magnitude samples into three separate lists, according to their x, y and z coordinate, respectively. After each recursive subdivision along one of the dimensions, the point lists corresponding to the other two dimensions are rearranged in accordance with the relative position from the splitting plane.

As has already been indicated, we use our own partitioning criterion. This is because the commonly used *surface area heuristic* (SAH) does not make any sense in our situation (SAH is tied to the probability of hitting the geometry contained within a node — first, this is undefined for points, and second, we are not building the tree to compute intersections with anything). Instead, we put the splitting plane at the spatial median of the currently examined point set. And since we want to separate regions with high density variance, we select such splitting dimension out of the three, that minimizes the density variation in the newly-created sub-nodes. Ideally, we should sweep through the examined points to find the optimal splitting plane position, but this would be too costly and would not make the resulting partitioning much better.

In similar spirit we designed the termination criterion. Yue et al. [62] derive it from the formulation of the largest empty rectangle problem by approximating the costs of

traversing a subdivided and non-subdivided node (basically they calculate if the subdivision removes a large enough empty rectangle that it pays off to traverse one more tree node). However, we are not solving the largest empty rectangle problem directly, and also approximating the costs of the additional traversal step on GPU would be complicated (because of the difficult predictability of the memory access patterns). Instead we use a user-provided constant threshold on the density values range within a node — is this range is below the threshold, we terminate the subdivision process for the node.

However, when we terminate the subdivision process, we finalize the current node by additional heuristic operations. First, each of the point samples is assigned a constant radius of influence (chosen by the user). Then, based on this radii, we look if there is not too much space uncovered by the samples somewhere within the node. If there exists such space we further subdivide the node until the samples inside it are encapsulated well enough. This process is illustrated by Figure 5.4. The main reason to use such heuristics is that sometimes the termination criterion can succeed even for clustered or too distant point samples (if they bear similar densities), which then causes too large VPCs and billboards generated for such nodes in Step 5.5.

The results of our kD-tree construction on a simple ellipsoid dataset and the already presented smoke cloud dataset are shown in Figure 5.5.

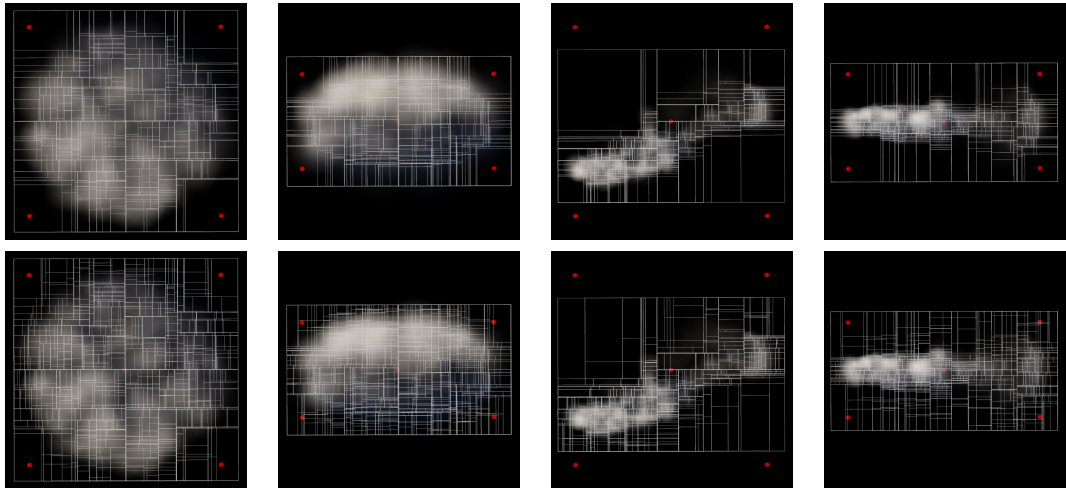


Figure 5.5: kD-tree visualization without the heuristic finalization (top row) and with it (bottom row). From left to right: ellipsoid dataset (top and side views) and the smoke cloud dataset (top and side views). The relative density threshold for the termination criterion was set to 0.25. Please note that the apparent tree complexity is the result of multiple subdivision levels being overlaid along the view direction. For the same reason the finalized versions appear to have some nodes subdivided also in the interior regions of the datasets, even though they do not.

5.4 (P) Peak densities computation

In Section 5.3 we described our kD-tree construction and the usage of the tree to partition the cloud volume for Woodcock tracking. What is still missing, however, are the values of the per-leaf local majorant extinction coefficients, which the algorithm requires. Although they could be approximated by taking the maximal value from the

densities stored with the point samples contained inside each leaf, this could of course lead to their underestimation.

Therefore we compute these values by traversing V_ρ for each leaf separately and finding the maximal value of the density at each respective leaf. In addition, we also find for each leaf the percentage of the cloud volume with non-zero density values (simply by counting the percentage of texels with non-zero densities, as we represent V_ρ by a 3D texture on GPU). We will need these values later during the Rendering phase. The results of the per-leaf peak densities computation are shown in Figure 5.6.

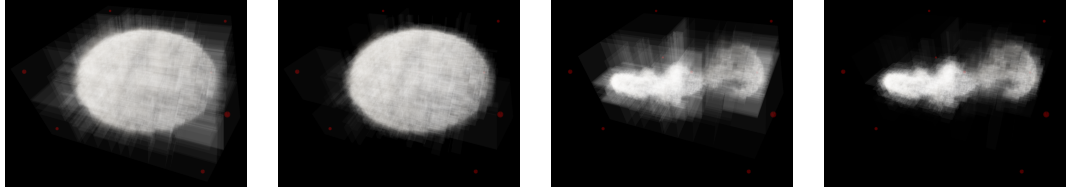


Figure 5.6: Visualization of the per-leaf densities in the ellipsoid (non-finalized versus finalized) and the smoke cloud (non-finalized versus finalized) datasets. Notice that despite the finalization of the kD-trees yields more nodes, it also produces better partitioning in terms of encapsulating similar-density regions.

5.5 (P) VPCs and billboards generation

We can now place the virtual photon collectors, along with their corresponding billboards, into the leaves of the tree constructed in Step 1c. VPCs and billboards each have two main geometrical properties that need to be set: position and radius.

The position of each VPC should naturally be in the centre of its corresponding kD-tree leaf's *axis-aligned bounding box* (AABB), since during the photon tracing stage (Section 5.7) it will serve as the nearest-neighbour photons accumulator for the leaf. However, since kD-tree is a structure based on rectilinear partitioning cells, billboards placed this way exhibit a visible regular distribution during the rendering. To break this regularity, we place each VPC into the centroid of the gradient point samples encapsulated by its corresponding leaf. This is of course an *ad hoc* solution, but it also results in VPCs and billboards being placed closer to regions where the cloud matter is located, which is a plausible effect in the end.

The radii of the generated VPCs and billboards should reflect the fact that a billboard is actually a substitute for a certain volume of the medium it represents. Since we treat billboards as spherical, the radius r of some VPC-billboard pair should be

$$r = \sqrt[3]{\left(\frac{3}{4\pi}abc\right)}$$

where a , b and c are the side lengths of the corresponding leaf's AABB. However, such an approach creates a significant amount of holes in the medium, because fundamentally any rectilinear body has larger half-diagonal than the radius of the sphere with the same volume. To solve this we simply double the radii of the generated billboards, so they somewhat overlap. However, this introduces another issue. Since each billboard represents a portion of the medium volume, it also has assigned a transmittance value proportional to this volume. This transmittance is later converted into the billboard

opacity and used during the Rendering phase to alpha-blend the rendered billboards together. To compensate for this increased volume, we later downscale the billboards transmittance values by the factor of 8, which is exactly by how much we increased the volumes represented by the billboards.

The resulting generated billboards and VPCs are shown in Figure 5.7. Note that we do not generate them for leaves with very small local majorant extinction coefficient, nor for the leaves which do not contain any point samples (because of the finalization step).

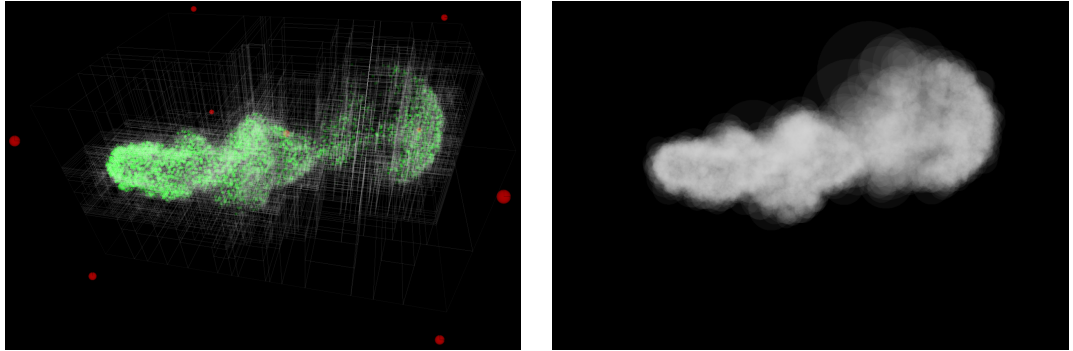


Figure 5.7: VPCs and billboards generated in the kd-tree visualized by Figure 5.6. The left image shows the generated VPCs inside the tree, and the right image the corresponding constant-coloured transparent billboards with already adjusted radii.

5.6 (P) Photon map initialization

As the final step of the Preparation phase the algorithm initializes the data structure which will represent the photon map M . As briefly indicated in Step 1f we use the spherical harmonic functions (defined in Section 2.3) to represent angular outgoing photon radiance at all VPCs.

There are three reasons that motivated us to use spherical harmonics to represent the photon map in our method:

1. The most costly operation in photon mapping is generally the illumination reconstruction. This is especially true for participating media, since it is necessary to accumulate photon contributions along the entire path from the observer, not just at its scene intersection. Usage of SH, in combination with the histogram reconstruction method, speeds up this process immensely. That is because all it takes to evaluate radiance coming from a VPC is to evaluate the Equation 2.15 for the direction to observer using the set of SH coefficients associated with the VPC.
2. In most cases at least a semi-dynamic data structure is needed to represent the photon map (e.g. even though we know how many photons we want to shoot, we do not know in advance how many times will each photon scatter in the scene). As we want to perform photon tracing on GPU, this is a problematic requirement for us, because possibilities of dynamic memory allocation on GPU are very limited. Therefore having a fixed amount of SH coefficients to represent the photon map is surely very beneficial in this situation.

3. As outlined in Chapter 4 our photon shooting scheme amortizes the costs of this operation across multiple image frames by dividing the total amount of photons into p generations. Then, during the reconstruction, these generations have to be combined together to produce the reconstructed estimate. Since SH coefficients combine linearly in a meaningful way, all that is needed for the reconstruction is to sum the corresponding coefficient of all generations G_i .

We represent M by a circular queue of p arrays of spherical harmonics coefficients. The memory requirements of this representation can be expressed as follows:

$$|M| = (p + 2) \times (\#VPCs \times 3\Lambda^2)$$

where

- p is the number of photon generations
- $\#VPCs$ is amount of VPCs generated by Step 1e
- Λ is number of SH bands to use; ‘3’ because we need a separate coefficient set for each of the RGB components

The reason why we need $(p + 2)$ arrays instead of just p is simple. To avoid the necessity to sum all generations of SH coefficients after every photon map update we take advantage of the linearity property of the SH-representation. For p generations of photons we need an additional SH coefficients array to maintain the sum of all G_i and yet another array to represent the generation which is about to be discarded — this last generation has to be subtracted from the summed array before a new photon generation is shot. If we would have exactly p arrays to represent the photons population, immediately after subtracting the oldest generation from the summed array this would logically contain energy of just last $(p - 1)$ photon generations.

As soon as we have the VPCs, the billboards and the photon map prepared the Rendering phase can begin by shooting the first generation of photons G_0 . Naturally, the photon map does not contain the entire flux present in the cloud Φ_T immediately. That is so only after the first p image frames. This is a part of the main current drawback of our method mentioned in Section 4.2. Section 7.2 describes some ideas how this could be solved in future.

5.7 (R) Photon tracing

After all prerequisites are accomplished the algorithm can finally start to produce image frames, until the next animation frame begins. Each image frame begins by tracing N_G photons through the rendered scene in its current state. We can divide this procedure into three phases.

Photon shooting First, N_G photons is emitted from the light sources in the scene. Each light source should emit a portion of the N_G photons proportional to its radiant power. We will focus on the illumination during the day since the rendering of clouds at night is considerably less important and also similar to the daily situation.

Direct illumination by the Sun is handled by shooting photons from the current Sun direction at the rendered cloud. The exact location where a photon enters the cloud is

determined by uniformly sampling the plane perpendicular to the Sun direction. The colour of the illumination the photons carry of course varies throughout the day, which is particularly visible during sunset or sunrise. We obtain this colour by attenuating the sunlight colour by the atmospheric extinction for the current light path from the Sun through the atmosphere.

As for the ambient light around the cloud we should correctly use the current environment map to shoot photons from the cloud surroundings. Since it would be quite costly to generate such map we use a simple approximation instead. We assign all ambient photons a constant colour, which however varies throughout the day. This colour represents the scattered atmospheric light which arrives at the cloud location.

We obtain the atmospheric attenuation values as well as the daytime-dependent ambient light colours from two precomputed textures. Computation of these textures is described in the author’s paper [16].

Photon tracking After leaving a scene light source the photon is traced through the medium using the techniques described in Chapter 2. The algorithm first generates a new scattering location by sampling the photon free path using the enhanced Woodcock tracking algorithm [62]. Then a new scattering direction is generated using the Equation 2.5. These two steps are repeated until the photon is discarded for leaving the cloud volume (since practically no absorption occurs in clouds).

Photon storing At each new scattering event the photon is added to M by projecting its flux and direction to the SH basis (using the Equation 2.16) and adding the values of the obtained SH coefficients to the nearest VPC.

However, we do not store the photon using its incident direction $\vec{\omega}_i$, but its scattered direction $\vec{\omega}_o$. By using $\vec{\omega}_i$ in general each VPC would store incident flux. To reconstruct the outgoing radiance in Step 2b for a VPC we would have to convolute the angular distribution of its incident flux with the phase function spatially aligned with the relative observer position in regard to the VPC (similar to the work of Kautz et al. [38]). This would however be quite a costly operation, because in terms of spherical harmonics this would mean that the phase function would have to be projected into the SH basis and the resulting coefficients rotated to align with the incident illumination function. Hence, by storing photons using $\vec{\omega}_o$ instead, we already obtain the outgoing flux at the VPC, so the reconstruction step just needs to directly evaluate it using the relative observer position.

5.8 (R) Illumination reconstruction

Every time the photon map is updated or the relative position between the rendered cloud and the observer changes, it is necessary to update the colours of the cloud billboards.

First, the photon map itself needs to be updated. The photon tracing procedure in Step 2a updated the newest photon generation G_i for some $i \geq p$. To update the photon map the SH coefficients array corresponding to G_i has to be added to the summed SH coefficients array. At the same time the oldest, already deprecated generation G_{i-p} has to be subtracted from the accumulation array.

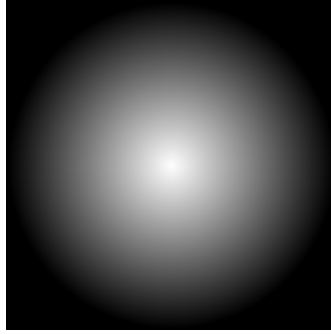


Figure 5.8: Radial gradient texture.

After the photon map is up-to-date we can proceed to the illumination reconstruction. Thanks to the decision to store the outgoing radiant flux at VPCs (see Section 5.7) the reconstruction just has to directly evaluate the Equation 2.15 for each VPC. Then, after dividing the obtained values by the medium volumes at the respective VPCs, we can immediately assign the resulting colours to the corresponding billboards.

5.9 (R) Cloud visualization

Having the billboards' colours updated by the previous step, we can directly render them on the graphics hardware. They are alpha-blended together using the values of the transmittance function properly scaled by the average density and volume of the imaginary region they represent. To ensure correct alpha-blending the billboards are sorted according to their observer distance and rendered in the back-to-front fashion.

As for the billboards themselves, we render them as textured semi-transparent radial shapes, using quads to represent them geometrically. We texture them by simple radial gradient textures (as shown in Figure 5.8), which represent the projection of an achromatic semi-transparent homogeneous sphere, that the billboards stand for. However, this texture just modulates the billboards opacity, not their colour — that is solely provided by the simulation.

Some resulting images of our method are shown in Figures 5.9 and 5.10. Despite the fact that both of these datasets are morphologically quite complex, the approximation by billboards still works fairly well. The important photon map parameter values were $N_T = 10.65M$, $p = 50$, $\Lambda = 5$. The obtained framerates were 4 FPS during the illumination update and 15–20 FPS with fully updated photon map on NVidia GeForce GTX485M GPU. Some of the images were over- or under-saturated to adapt to the very high dynamic range of cloud illumination (despite our renderer already uses a simple tone mapping operator). Notice that our method is able to account for the typical global illumination effects in clouds, such as volumetric shadows, varying global scattering anisotropy, or silver edges.

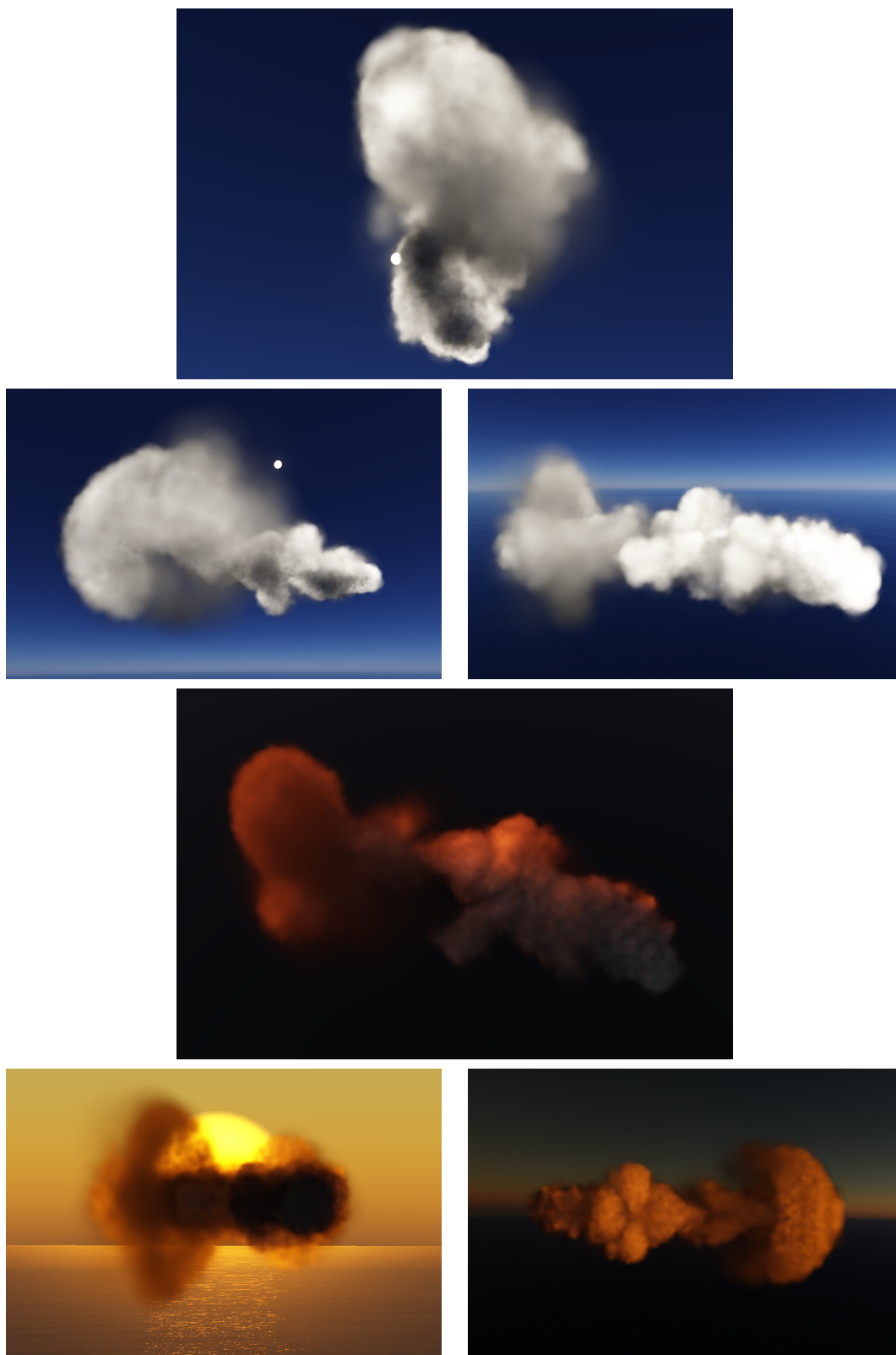


Figure 5.9: Screenshots of the smoke cloud dataset (a part of the extended scenes for the PBRT renderer, <http://www.pbrt.org>) during the day and in the evening.

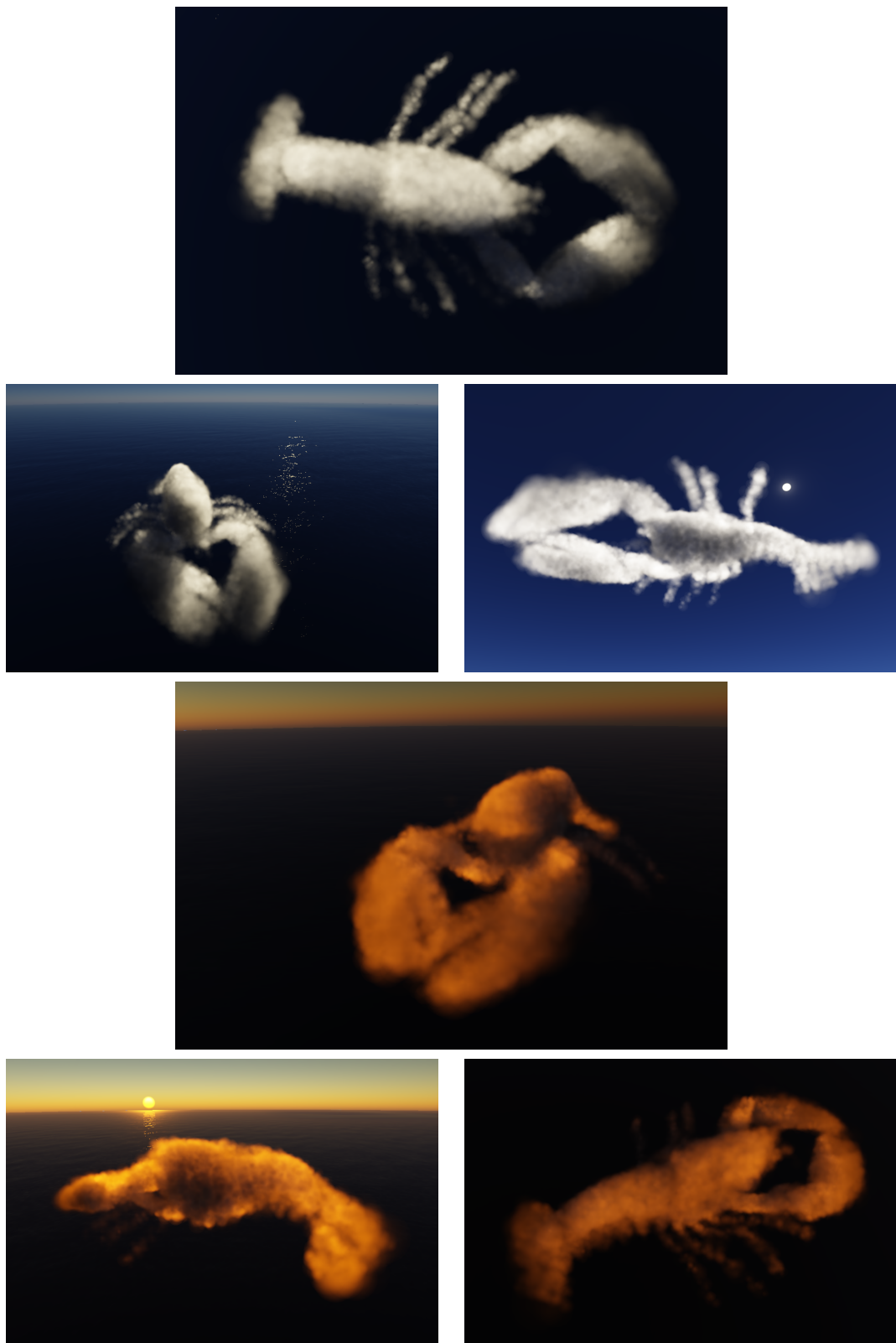


Figure 5.10: Screenshots of the lobster dataset (from the Stefan Roettger's online volume library, <http://www9.informatik.uni-erlangen.de/External/vollib>) during the day and in the evening.

6. Method implementation

In previous chapters we described the high-level theoretical concepts behind our work. This chapter descends to a lower level by describing the programmatic environment we use (Section 6.1) and the actual implementation of the presented technique (Section 6.2). We conclude this chapter by a series of performance tests, which will demonstrate the ability of our method to run at interactive framerates (Section 6.3).

6.1 Environment and libraries

Languages For the main application we use the C++ language. We chose C++ because it is naturally tied to the development of interactive graphics applications and high-performance applications in general. As a consequence, most important graphical libraries and frameworks are natively written in C++.

The parallel code that runs on GPU is written in CC (CUDA C), which is a combination of C and C++ modified for use in GPU programming.

Our shader language of choice is HLSL, which is used in the rendering front-end for defining the way how the rendered entities are visualized. HLSL is a minor modification of the Cg language, which in turn is C modified for graphical GPU programming. HLSL was chosen over Cg because it is tied to the DirectX 3D graphics library we use.

Libraries and frameworks

DirectX (v9.0c) 3D graphics SDK and API. Mainly takes care of communicating with GPU in graphics-related tasks and allocating graphics-related resources.

CUDA (v3.2) An SDK and API for parallel general-purpose computations on GPU. The acronym stands for *Compute Unified Device Architecture*. We use it for launching parallel GPU programs from the host application and for allocation of the related non-graphical resources on GPU.

Win32 API The low-level GUI API for Windows operating systems. Our application does not actually contain any GUI elements, so WinAPI takes care of only some basic tasks, such as the rendering window management and system messages handling.

Boost (v1.46) Probably the largest general-purpose library package for C++. We use only a fraction of its modules, such as timers, random numbers generation, or lexical casting.

TinyXML A small library that provides pull-parsing of XML files. We use it for configuration file loading.

Operating system We have developed our implementation on Windows 7 Professional x64 OS. It should be able to run on any Windows OS, as long as it supports the library versions we use and the underlying machine fulfils the hardware requirements. The application does not run on other systems, because the DirectX library we use is tied to the Windows systems. As for the hardware requirements of AtmoVision please refer to Appendix B.

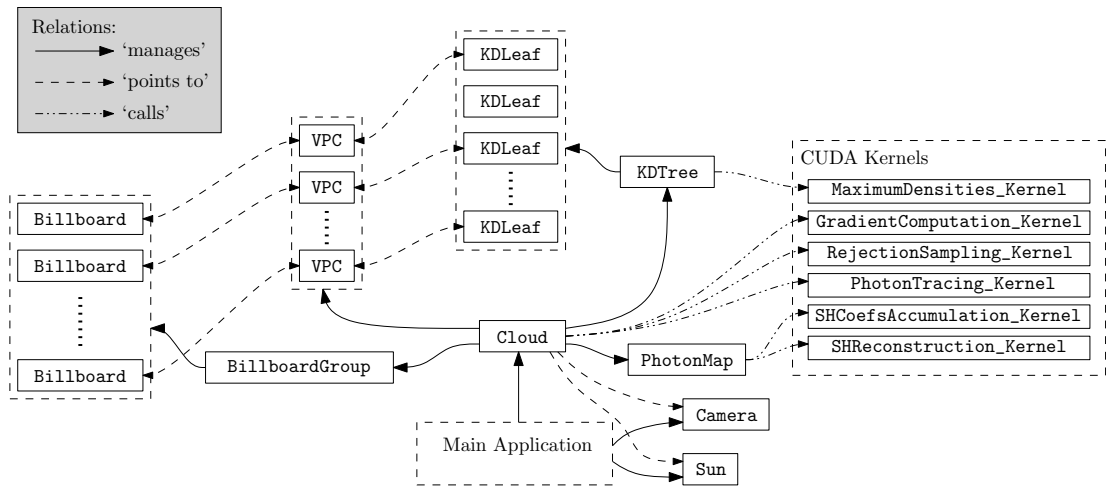


Figure 6.1: AtmoVision class diagram, depicting the more notable classes responsible for cloud rendering.

6.2 Implementation design

The application that implements our method is named AtmoVision. From the user’s point of view it is a very simple demonstrative application that does not contain any GUI elements. It is not intended for practical usage.

The class design of AtmoVision is shown in Figure 6.1. It loosely corresponds to the algorithm sketched in Section 4.2 and described in detail by Chapter 5. The rest of this section will describe this correspondence and the relations between the classes.

There are two types of code in AtmoVision — host code and device code. This terminology comes from the distinction the CUDA SDK makes, and its meaning is that the host code is executed by CPU, while the device code runs in parallel on GPU. In practice all C++ code is host code; the device code is a set of separately compiled CC functions linked externally to the host code. These functions are called *kernels* and are launched by CPU. When a kernel is launched, CPU creates through device driver a *grid* of parallel GPU threads, which then execute the kernel code. How many threads will be launched in the grid is decided prior to the launch by the programmer. It is important to have in mind that these threads are very lightweight in comparison to CPU threads — launching a grid with e.g. 10000 threads is perfectly feasible in most cases.

The difference between host and device code is not only the processing unit that executes them — they also use different memory spaces. While host code naturally works with the computer’s physical memory, device code uses the on-die GPU memory. Therefore, the standard workflow when working with device kernels is as follows:

1. allocate a desired amount of device memory
2. copy the working data from host to device memory
3. launch the device kernel
4. copy the processed data from device memory back to host memory
5. free the allocated device memory

Of course, this does not mean that at every kernel launch there has to be data transfer from CPU to GPU memory and back after the launch completes. The device memory

can be allocated once and then pointers to it may normally exist in host code and passed to kernel launches whenever needed.

Host code Let us now look at the class substructure of AtmoVision responsible for cloud rendering, which is depicted in Figure 6.1. The ‘Main Application’ designates the rest of the AtmoVision structure, which is however not relevant for us now. Instead, we will focus on the `Cloud` class, which understandably represents a single cloud instance. It further manages the following classes:

- `KDTree`
The `KDTree` class represents the kD-tree data structure created in Step 1c. The constructed `KDTree` keeps its leaves (represented by the `KDLeaf` class) in a linear list for easier iteration through them. Each `KDLeaf` holds pointer to one instance of `VPC` class, except the ones that do not fulfil the criteria described at the end of Section 5.5.
- `VPC`
The *virtual photon collectors* generated by Step 1e are represented by this class. Each `VPC` points to one instance `KDLeaf` class that it was generated from (but not the other way around). Also, each `VPC` points (indirectly through an index maintained by `BillboardGroup`) to an instance of the `Billboard` class, which represents it during visualization in Step 2c. The `Cloud` class holds the list of all `VPCs` for a given animation frame.
- `BillboardGroup`
The `BillboardGroup` class manages all operations related to billboards — their creation and destruction, camera-relative sorting, and so on. It also takes care of their rendering in Step 2c — every time the camera position changes the `BillboardGroup` class refills the geometry buffers that hold their vertex data, so that the billboards face the camera all the time.
- `PhotonMap`
Photon map is created in Step 5.6. As described in Section 5.6 it internally is a circular queue of p arrays of SH coefficients. However, these arrays reside in the device memory, since they are updated and reconstructed by device kernels.

Other than that, there are also the `Camera` and `Sun` classes managed by the main application. However, the `Cloud` class holds pointers to them, so it can recognize when the position of either of them changes, to update the respective dependent entities.

Device code The part of AtmoVision code executed by GPU¹ consists of the kernels grouped under the ‘CUDA Kernels’ header in Figure 6.1, which also shows which respective host class calls them. Let us look at the purpose of each of them.

- `MaximumDensities_Kernel` computes the per-leaf local majorant extinction coefficients in Step 1d. The grid size is the same as the number of leaves the current kD-tree has, and each thread computes the local σ_T for a single leaf.

¹We are of course referring to the non-graphical code. Management of shaders is the responsibility of each class that visualizes something. The work with shaders is a lot different and somewhat simpler than with general computation routines on GPU — shaders use JIT compilation and are called implicitly when an associated rendering batch is submitted through DirectX calls.

- `GradientComputation_Kernel` computes the gradient field V_g in Step 1a. A 2D grid with the dimensions of $x \times y$ is launched (where x and y are the first two dimensions of V_ρ) and each thread computes the values of V_g along an entire z -column.
- `RejectionSampling_Kernel` performs rejection sampling of V_g in Step 1b. The grid size equals the number of desired point samples and each thread produces one sample.
- `PhotonTracing_Kernel` traces N_G photons through the cloud density field in Step 2a. The launched grid size is N_G/N_t , and each thread shoots and traces N_t photons.
- `SHCoefsAccumulation_Kernel` accumulates the array of SH coefficients in Step 2b. The grid size is the same as the number of VPCs and every thread accumulates the complete SH coefficients set of one VPC.
- `SHReconstruction_Kernel` performs the observer-dependent illumination reconstruction in Step 2b. Again, the grid size equals the VPCs amount and each thread reconstructs the illumination for a single VPC.

As can be seen we successfully defer most of the computationally-demanding operations to GPU. The only exception is the kD-tree construction in Step 1c, as this is not an easily parallelizable procedure. However, we recently became aware of such a technique by Zhou et al. [63], so in future we plan to examine and implement it.

6.3 Evaluation

Having both theoretical and practical part of our method described we can now analyse its results and performance under various key settings. Our testing PC configuration was:

- Intel Core i7-2630QM CPU @ 2GHz
- 16GB DDR3 physical memory
- NVidia GeForce GTX 485M GPU with 2GB GDDR5 on-die memory
- Windows 7 Professional x64 OS

The testing scenario we considered was as follows. We started by rendering a reference case with a fixed settings of the key simulation parameters. We measured the times necessary to complete each of the more time-consuming preparation Steps 1a–1d (Steps 1e and 1f have almost negligible costs compared to the other four). We also measured the average time necessary to trace the per-frame photon generations. After that, we rendered a series of test cases, varying the values of the considered key parameters in each of them and comparing the measured time values against the reference case. By this we can see the influence of the important simulation parameters on the simulation speed, and also on the quality of rendered images. All these measurements were performed with all other scene elements disabled to avoid interference with the cloud rendering measurements. Also, we use global Woodcock tracking in Step 2a (that is, we use the original approach with a single global σ_T instead of using the local per-leaf values). The purpose of this is the comparison between the global and the local technique on GPU.

The scene we used for all measurements is shown in Figure 6.2. All test cases were rendered in the resolution of 1920 by 1080. The simulation parameters we considered for testing and the corresponding test cases were:

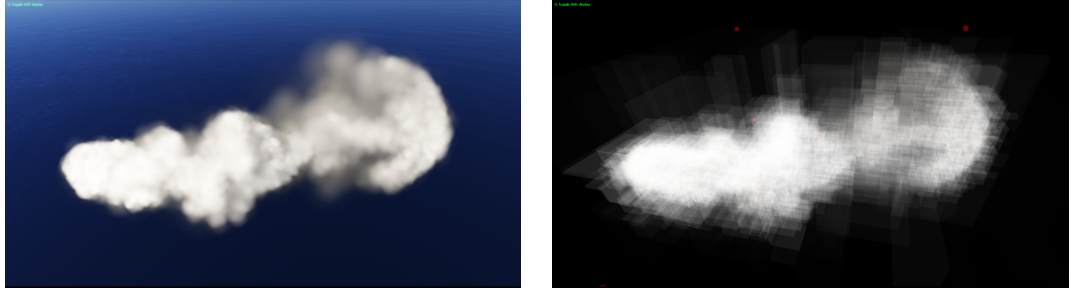


Figure 6.2: Reference cloud and its kD-tree.

- #PS — the number of point samples used for sampling V_g in Step 1b. Considered values: **5k** (test case A1), **20k** (reference test case), **50k** (test case A2)
- N_T ; considered values: **1.5M** (case B1), **5M** (reference case), **10M** (B2) (however, the actual values used by the simulation may be somewhat different, since they are aligned with the grid size)
- p ; considered values: **20** (case C1), **50** (reference case), **200** (case C2)
- Λ ; considered values: **2** (case D1), **4** (reference case), **6** (case D2)
- local Woodcock tracking, using #PS of **20k** (case E1) and **100k** (case E2)

The results of the measurements are visualized in Figure 6.3 and summarized in the following table:

	#PS	N_T	p	Λ	#VPCs	t_{1b}	t'_{1c}	t''_{1c}	t_{1d}	t_Σ	t_{2a}	t'_{2a}
Ref	20k	5.73M	50	4	8668	92	24	265	35	421	132	1152
A1	5k	5.73M	50	4	2787	35	5	68	47	160	120	1047
A2	50k	5.73M	50	4	17729	205	74	674	37	995	135	1178
B1	20k	1.64M	50	4	8668	92	25	268	34	424	60	1829
B2	20k	10.6M	50	4	8668	93	25	270	35	428	212	995
C1	20k	5.24M	20	4	8668	92	24	267	34	423	251	958
C2	20k	6.55M	200	4	8668	92	25	269	34	425	60	1832
D1	20k	5.73M	50	2	8668	94	24	268	35	426	87	759
D2	20k	5.73M	50	6	8668	92	25	268	34	425	221	1928
E1	20k	5.73M	50	4	8668	94	25	267	35	427	174	1518
E2	100k	5.73M	50	4	28448	404	168	1362	39	1978	214	1867

- #VPCs is the number of VPCs generated as a result of using #PS gradient magnitude samples
- t_{1a} [ms] (not listed in the table) is the time spent in Step 1a. Its value in all measurements was 5 milliseconds (since the resolution of V_p was the same in all test cases)
- t_{1b} [ms] is the time spent rejection-sampling V_g in Step 1b
- t'_{1c} [ms] is the time spent on sorting the gradient magnitude samples in Step 1c
- t''_{1c} [ms] is the time spent on building the kD-tree from the sorted point samples in Step 1c
- t_{1d} [ms] is the time spent on computing the local majorant extinction coefficients in Step 1d
- t_Σ [ms] is the sum of t_{1a} to t_{1d}
- t_{2a} [ms] is an average time it took to trace N_G photons in Step 2a

- t'_{2a} [ms] is perhaps the most important quantity to look at. It expresses the average time required to trace 1M photons throughout all photon generations and can be regarded as an efficiency indicator for the corresponding simulation parameters. Mathematically it can be expressed as

$$t'_{2a} = p \cdot t_{2a} \cdot \frac{10^6}{N_T}$$

The measurements results provide us with a certain insight into the inner workings of our method. We can draw several conclusions from them, albeit none of them is entirely unexpected.

- First of all, we can see from t'_{2a} that the efficiency of photon tracing grows with increasing number of photons shot per generation (e.g. in cases B2 and C1). This is understandable, because the photon shooting has a certain constant setup overhead, so the less photons are shot, the higher relative share on the step cost will this setup stage have. Moreover, with increased amount of work in one kernel launch the GPU scheduler has more options how to optimize the grid execution.
- What may at first seem surprising is the fact that the local version of Woodcock tracking (case E1) is slower on the same simulation settings than the global version. To explain this it is important to realize that the costs of memory accesses on GPU are much higher than on CPU, when compared against the costs of arithmetic instructions. The local Woodcock tracking variant saves time by performing less (superfluous) arithmetic operations, but adds the necessity to traverse the partitioning kD-tree. In our reference case the corresponding kD-tree occupied roughly 0.5MB of GPU memory, which is not that much, but it is still too much to fit into the shared memory on GPU (which serves as a user-managed cache). As a consequence the tree must reside in the global GPU memory, which has very high access times, and this is what causes the measured times. This hypothesis is supported by the measurement case E2, where the used settings resulted in roughly 3.3 times higher amount of VPCs and hence kD-tree leaves. So even though the kD-tree partitions the cloud density field more finely, the added traversal costs are simply higher.
- The costs of the entire Preparation phase are directly proportional to #PS and do not depend on the remaining simulation settings. Out of these the most costly operation is the kD-tree construction, which we will aim to solve in future.
- The desired precision of the angular illumination approximation by spherical harmonic functions influences the rendering times quite significantly (test cases D1 and D2). This has two reasons — not only the evaluation of SH themselves adds additional computational costs, but also the increased amount of stored coefficients causes more reads and writes from an to the global memory, which as we already mentioned is the main bottleneck also in the photon tracking phase.

Overall, we can conclude this section by saying that our method is able to maintain interactive speeds for the majority of reasonable simulation settings, even if the illumination is updated in each image frame. On the other hand, the preparation times at each animation frame are too high to ensure interactive animation speeds. In Section 7.2 we elaborate how this could be solved.

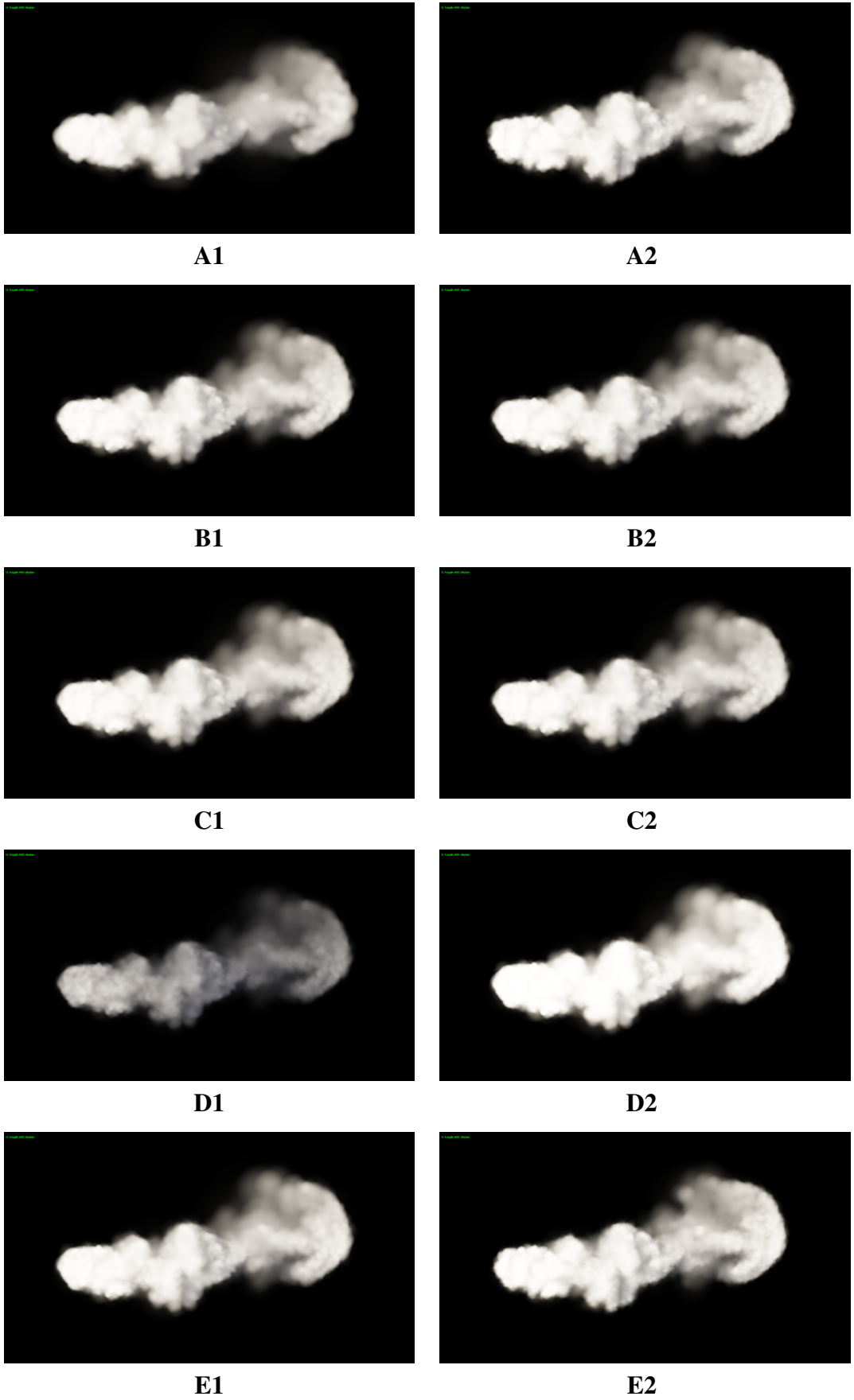


Figure 6.3: Rendered images corresponding to the measured test cases.

7. Conclusion

The work presented in this thesis aims to tackle the difficult problem of interactive simulation of global illumination in clouds. Our aim from the beginning was to build our approach on the valid physical laws and stick to them until it is really necessary to resort to approximative or even completely empirical solutions because of the limitations the environment of interactive rendering imposes. For the same reason we tried to formulate a set of realistic assumptions and observations valid for our target phenomenon, and hold to them when figuring out the solutions to encountered problems and obstacles.

Of course, the resulting method prototype we propose is in no way complete, nor it is ready for industrial application. Nevertheless, we successfully solved at least a part of the problems associated with this topic. By this, we have showed that the contemporary technical possibilities already allow usage of physically-correct methods even in interactive environment, which has until recently been reserved for the domain of non-interactive approaches. Therefore the end of this chapter provides some insights how to continue on this work and develop it to the point where it is feasible for practical applications.

7.1 Fulfilment of the thesis goals

In Section 1.3 we stated the main objectives of our work. Let us now discuss how much this effort was successful.

1. We have examined a substantial volume of works not only from the rendering field, but also publications regarding the physics of participating media. This naturally deepened our understanding of the physical nature of light scattering. This has been concisely summarized in Chapter 2. As for the existing approaches for rendering participating media we have reviewed the ones known to us in Chapter 3. Although this overview surely cannot be claimed as complete, it is sufficiently exhaustive to give reader not familiar with this group of methods a solid understanding of the main problems associated with participating media rendering.
2. We have proposed a method prototype that is able to interactively simulate most of the described phenomena associated with clouds. Of course, the method in its current state has some weaknesses, the most apparent being its inefficiency of cloud animation. However, there is no principal issue that would forbid future development of the method to a point where it efficiently copes with all possible cloud properties; some suggestions how to do that will be discussed in the next section. Overall, the presented approach fulfils our initial expectations.
3. Our implementation described in Chapter 6 contains all features from Chapter 5. It utilizes the features of the state-of-the-art consumer GPUs, which enables the method to run interactively.

7.2 Discussion

Since we now have implemented a working prototype of our method, the next step is to consider various ways of extending it. This section presents some ideas in this direction we would like to explore, but the time constraints we had on this work did not yet allow us to.

Overall optimizations First of all our implementation is not entirely optimal. Despite all important algorithms we employ are implemented with the optimal asymptotic complexity, we feel there are still many options for quantitative optimization.

For instance, when looking at the evaluation of the preparation stage it is immediately visible that the most expensive operation is the kD-tree construction in Step 1c. First of all, as already mentioned we would like to explore the possibilities of parallel kD-tree build on GPU. Yet even on CPU our implementation is not entirely optimized. For example at each subdivision step it is necessary to perform the re-sorting of the point samples for the newly-created child nodes (as briefly described in Section 5.3). Currently our implementation uses auxiliary lists for this operation, instead of reordering the points in-place.

Another potentially large space for optimizations are the GPU CUDA kernels. First, we have not really performed any thorough quantitative analysis of the kernel launches parametrizations, which could influence their performance significantly. Second, the main part of any GPU program optimization is its memory access pattern, yet we have not properly examined these either. The most notable example is how the partitioning kD-tree is currently stored — since it resides in global memory (due to its too large size), its traversal is much slower as if it would reside in texture or even shared memory.

Putting all together we think that even by just optimizing the existing code, without any approximations or simplifications, we would roughly double the speeds at which the simulation works now.

Enhancements There are numerous possible enhancements we would like to test. For instance, one of the current issues of the method is its relative high illumination variance, which can be seen under some lighting conditions as varying colours of the neighbouring VPCs. This is of course a consequence of the histogram reconstruction method. Using some other reconstruction method, such as photon splatting, could significantly reduce the variance and thus decrease the amount of photons necessary for smooth illumination transitions.

Next, despite the billboard representation maps well to the common designs of current real-time rendering pipelines, it is difficult to produce higher-frequency details with it. Ray-marching along the view rays, plus possibly a procedural enrichment, is necessary to produce such details. We have actually implemented a special-purpose ray-marching in AtmoVision — instead of producing a single ray-marched image of the rendered cloud, we render each kD-tree leaf into a small billboard texture, then align these billboards in projection space and composite them together, which produces a seamless image. The problem arises when each billboard has a different colour denoted by the corresponding VPCs — after that the billboards do not fit together anymore and produce a very blocky-looking image, see Figure 7.1. Naturally we would

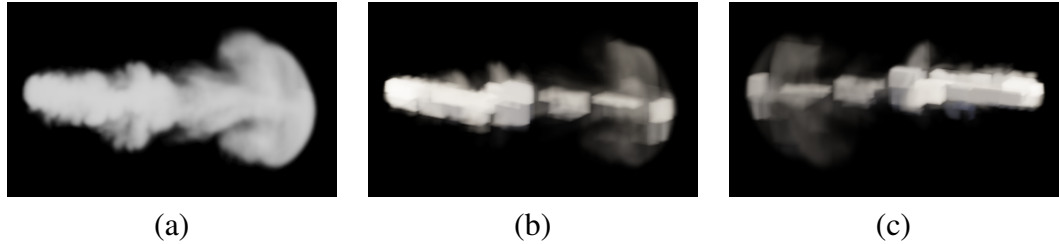


Figure 7.1: The smoke cloud dataset rendered by ray-marching using 189 separately updated billboards. The image (a) just accumulates transparency along the view rays, while (b) and (c) also use the per-VPC illumination values. The blockiness in the latter case is apparent immediately.

like to find a way to combine these two approaches, so we can divide the cloud volume into discrete regions, but still create fluent transitions between them.

Practical extensions To be used in an applied environment the usability of our approach has to be increased. First of all, we have not yet elaborated any multi-resolution scheme for our method. That is of course unacceptable for any practical uses, as it allows just one or two clouds rendered at a time. What comes to mind in this direction is that the cloud VPCs and billboards could be placed in a hierarchy (most probably defined by the kD-tree) and from a distance the nearby smaller VPCs would just hierarchically collapse together.

Another issue is the already mentioned animation fluency problem. Not only using the method as it is now means that for each new animation frame the entire preparation phase needs to be performed. In addition also the photon map has to be rebuilt, since it is fixed to the particular VPCs distribution, which naturally changes between the animation frames. Hence we would like to explore possibilities how to transform one photon map into another without noticing. We think it should be possible to sample the nearby old VPCs at the location of a newly created VPC, and then interpolate their illumination. Amortization of this process across multiple image frames should not be problematic either, if needed.

Experimental extensions The current SH representation of the per-VPC angular illumination distribution works fairly well, but as we saw in Section 6.3 it is quite costly. Moreover it does not capture the highly-anisotropic silver lines phenomenon very well (or only when a higher number of SH bands is used). However, we observed (and not only us, see e.g. [26]) that in most places in a cloud the illumination comes predominantly from one major direction or from directions close to it. In our opinion it should be feasible to represent the illumination encoded into the asymmetry coefficient g centred around the light direction, for instance. The encoding could simply be done by evaluating the Equation 2.4. Then during the reconstruction phase this coefficient could be used as the parameter for some general phase function approximation, such as f_{HG} .

Finally, we think that there is also a need for a visual adaptation model in clouds. This is because the illumination dynamic range in clouds is much higher than in other environments (such as the sky), which results in the cloud often being either over- or under-saturated.

Bibliography

- [1] Alok Aggarwal and Subhash Suri. Fast algorithms for computing the largest empty rectangle. In *Proceedings of the third annual symposium on Computational geometry*, SCG '87, pages 278–290, 1987.
- [2] Boris Airieau, Daniel Meneveaux, Flavien Bridault, and Philippe Blasi. Photon streaming for interactive global illumination in dynamic scenes. *Visual Computer*, 27:229–240, March 2011.
- [3] Larry Bergman, Henry Fuchs, Eric Grant, and Susan Spach. Image rendering by adaptive refinement. *SIGGRAPH Computer Graphics*, 20:29–37, August 1986.
- [4] Margaret F. Born and Emil Wolf. *Principles of optics*. Cambridge University Press, 7th (extended) edition, 1999. ISBN 0521642221.
- [5] Antoine Bouthors, Fabrice Neyret, and Sylvain Lefebvre. Real-time realistic illumination and shading of stratiform clouds. In *Eurographics Workshop on Natural Phenomena*, 2006.
- [6] Antoine Bouthors, Fabrice Neyret, Nelson Max, Eric Bruneton, and Cyril Crassin. Interactive multiple anisotropic scattering in clouds. In *Proceedings of the 2008 symposium on Interactive 3D graphics and games*, I3D '08, pages 173–182, 2008.
- [7] Wayne Carlson. A critical history of computer graphics and animation. <http://design.osu.edu/carlson/history/lessons.html>, 2003. Version from July 18th 2011.
- [8] Deukhyun Cha, Sungjin Son, and Insung Ihm. Gpu-assisted high quality particle rendering. *Computer Graphics Forum*, 28(4):1247–1255, 2009.
- [9] Subrahmanyan Chandrasekhar. *Radiative transfer*. Dover Publications, 1960. ISBN 0486605906.
- [10] W. Coleman. Mathematical verification of a certain Monte Carlo sampling technique and applications of the technique to radiation transport problems. In *Nuclear Science and Engineering* 32, pages 76–81, 1968.
- [11] Abhinav Dayal, Cliff Woolley, Benjamin Watson, and David Luebke. Adaptive frameless rendering. In *ACM SIGGRAPH 2005 Courses*, SIGGRAPH '05, 2005.
- [12] Kirill Dmitriev, Stefan Brabec, Karol Myszkowski, and Hans-Peter Seidel. Interactive global illumination using selective photon tracing. In *Proceedings of the 13th Eurographics workshop on Rendering*, EGRW '02, pages 25–36, 2002.
- [13] Yoshinori Dobashi, Kazufumi Kaneda, Hideo Yamashita, Tsuyoshi Okita, and Tomoyuki Nishita. A simple, efficient method for realistic animation of clouds. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '00, pages 19–28, 2000.

- [14] Philip Dutré, Philippe Bekaert, and Kavita Bala. *Advanced global illumination, 2nd edition*. A K Peters/CRC Press, 2006. ISBN 1568811772.
- [15] Oskar Elek. Rendering planetary atmospheres in real-time. Bachelor thesis, Charles University Prague, 2008.
- [16] Oskar Elek and Petr Knoch. Real-time spectral scattering in large-scale natural participating media. In *Proceedings of Spring Conference on Computer Graphics 2010*, pages 83–90. Comenius University Bratislava, 2010.
- [17] Thomas Engelhardt, Jan Novak, and Carsten Dachsbacher. Instant multiple scattering for interactive rendering of heterogeneous participating media. Technical report, KIT - Karlsruhe Institut of Technology, December 2010.
- [18] Raanan Fattal. Participating media illumination using light propagation maps. *ACM Transactions on Graphics*, 28:7:1–7:11, February 2009.
- [19] Geoffrey Y. Gardner. Visual simulation of clouds. In *Proceedings of the 12th annual conference on Computer graphics and interactive techniques, SIGGRAPH '85*, pages 297–304, 1985.
- [20] Robert Geist, Karl Rasche, James Westall, and Robert Schalkoff. Lattice-Boltzmann lighting. In *Rendering Techniques 2004 (Proceedings of the Eurographics Symposium on Rendering)*, pages 355–362, 2004.
- [21] Robin Green. Spherical harmonic lighting: the gritty details. <http://www.research.scea.com/gdc2003/spherical-harmonic-lighting.pdf>, 2003. Version from July 20th 2011.
- [22] Jörg Haber, Marcus Magnor, and Hans-Peter Seidel. Physically-based simulation of twilight phenomena. *ACM Transactions on Graphics*, 24:1353–1373, October 2005.
- [23] Toshiya Hachisuka and Henrik Wann Jensen. Stochastic progressive photon mapping. *ACM Transaction on Graphics*, 28:141:1–141:8, December 2009.
- [24] Toshiya Hachisuka, Shinji Ogaki, and Henrik Wann Jensen. Progressive photon mapping. *ACM Transaction on Graphics*, 27:130:1–130:8, December 2008.
- [25] L. G. Henyey and J. L. Greenstein. Diffuse radiation in the Galaxy. *Astrophysical Journal*, 93(1):70–83, 1941.
- [26] Ivo Ihrke, Gernot Ziegler, Art Tevs, Christian Theobalt, Marcus Magnor, and Hans-Peter Seidel. Eikonal rendering: efficient light transport in refractive objects. *ACM Transactions on Graphics*, 26, July 2007.
- [27] Wojciech Jarosz, Craig Donner, Matthias Zwicker, and Henrik Wann Jensen. Radiance caching for participating media. *ACM Transaction on Graphics*, 27:7:1–7:11, March 2008.
- [28] Wojciech Jarosz, Henrik Wann Jensen, and Craig Donner. Advanced global illumination using photon mapping. In *ACM SIGGRAPH 2008 classes, SIGGRAPH '08*, pages 2:1–2:112, 2008.

- [29] Wojciech Jarosz, Matthias Zwicker, and Henrik Wann Jensen. The beam radiance estimate for volumetric photon mapping. In *ACM SIGGRAPH 2008 classes*, SIGGRAPH '08, pages 3:1–3:112, 2008.
- [30] Henrik Wann Jensen. Global illumination using photon maps. In *Proceedings of the eurographics workshop on Rendering techniques '96*, pages 21–30, 1996.
- [31] Henrik Wann Jensen. *The photon map in global illumination*. PhD thesis, Technical University of Denmark, September 1996.
- [32] Henrik Wann Jensen. *Realistic image synthesis using photon mapping*. A. K. Peters, Ltd., 2001. ISBN 1568811470.
- [33] Henrik Wann Jensen and Per H. Christensen. Efficient simulation of light transport in scenes with participating media using photon maps. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '98, pages 311–320, 1998.
- [34] Henrik Wann Jensen, Stephen R. Marschner, Marc Levoy, and Pat Hanrahan. A practical model for subsurface light transport. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '01, pages 511–518, 2001.
- [35] James T. Kajiya. The rendering equation. In *Proceedings of the 13th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '86, pages 143–150, 1986.
- [36] James T. Kajiya and Brian P. Von Herzen. Ray tracing volume densities. In *Proceedings of the 11th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '84, pages 165–174, 1984.
- [37] Anton Kaplanyan and Carsten Dachsbacher. Cascaded light propagation volumes for real-time indirect illumination. In *Proceedings of the 2010 ACM SIGGRAPH symposium on Interactive 3D Graphics and Games*, I3D '10, pages 99–107, 2010.
- [38] Jan Kautz, Peter-Pike Sloan, and John Snyder. Fast arbitrary BRDF shading for low-frequency lighting using spherical harmonics. In *Proceedings of the 13th Eurographics Workshop on Rendering*, 2002.
- [39] Alexander Keller. Instant radiosity. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '97, pages 49–56, 1997.
- [40] Joe Kniss, Simon Premoze, Charles Hansen, and David Ebert. Interactive translucent volume rendering and procedural modeling. In *Proceedings of the conference on Visualization '02*, VIS '02, pages 109–116, 2002.
- [41] Eric P. Lafortune and Yves D. Willems. Rendering participating media with bidirectional path tracing. In *Proceedings of the Eurographics workshop on Rendering techniques '96*, pages 91–100, 1996.

- [42] Gustav Mie. Beiträge zur optik trüber medien, speziell kolloidaler metallösungen. *Annalen der Physik*, 330(3):377–445, 1908.
- [43] Don P. Mitchell. Generating antialiased images at low sampling densities. *SIGGRAPH Computer Graphics*, 21:65–72, August 1987.
- [44] Tomoyuki Nishita, Yoshinori Dobashi, and Eihachiro Nakamae. Display of clouds taking into account multiple anisotropic scattering and sky light. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, SIGGRAPH '96, pages 379–386, 1996.
- [45] James S. Painter and Kenneth R. Sloan. Antialiased ray tracing by adaptive progressive refinement. *SIGGRAPH Computer Graphics*, 23:281–288, July 1989.
- [46] Mark Pauly, Thomas Kollig, and Alexander Keller. Metropolis light transport for participating media. In *Proceedings of the Eurographics Workshop on Rendering Techniques 2000*, pages 11–22, 2000.
- [47] Matt Pharr and Greg Humphreys. *Physically based rendering: from theory to implementation, 2nd edition*. Morgan Kaufmann, 2010.
- [48] Timothy J. Purcell, Craig Donner, Mike Cammarano, Henrik Wann Jensen, and Pat Hanrahan. Photon mapping on programmable graphics hardware. In *ACM SIGGRAPH 2005 Courses*, SIGGRAPH '05, 2005.
- [49] Mathias Raab, Daniel Seibert, and Alexander Keller. Unbiased global illumination with participating media. In *Proceedings of Monte Carlo and Quasi-Monte Carlo Methods 2006*, pages 591–606, 2006.
- [50] Kirk Riley, David S. Ebert, Martin Kraus, Jerry Tessendorf, and Charles D. Hansen. Efficient rendering of atmospheric phenomena. In *Rendering Techniques '04 (Proceedings of the Eurographics Symposium on Rendering)*, pages 374–386, 2004.
- [51] Volker Schönefeld. Spherical harmonics. http://heim.c-otto.de/~volker/prosem_paper.pdf, 2005. Version from July 20th 2011.
- [52] Peter-Pike Sloan. Stupid spherical harmonics (SH) tricks. Game Developers Conference 2008, February 2008. <http://www.ppsloan.org/publications/>. Version from July 20th 2011.
- [53] Peter-Pike Sloan, Jan Kautz, and John Snyder. Precomputed radiance transfer for real-time rendering in dynamic, low-frequency lighting environments. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 527–536, 2002.
- [54] Joe Stam. Multiple scattering as a diffusion process. In *Eurographics Rendering Workshop*, pages 41–50, 1995.
- [55] László Szirmay-Kalos, Mateu Sbert, and Tamás Umenhoffer. Real-time multiple scattering in participating media with illumination networks. In *Proceedings of the Eurographics Symposium on Rendering*, 2005.

- [56] Hendrik C. van de Hulst. *Light scattering by small particles*. Dover Publications, 1981. ISBN 0486642283.
- [57] Ingo Wald and Vlastimil Havran. On building fast kd-trees for ray tracing, and on doing that in $O(N \log N)$. In *Proceedings of IEEE Symposium on Interactive Ray Tracing 2006*, pages 61–69, 2006.
- [58] Niniane Wang. Realistic and fast cloud rendering. In *Journal of Graphics, GPU & Game Tools*, 2003.
- [59] Carsten Wenzel. Real-time atmospheric effects in games. In *ACM SIGGRAPH 2006 Courses, SIGGRAPH '06*, pages 113–128, 2006.
- [60] E. Woodcock, T. Murphy, P. Hemmings, and T. Longworth. Techniques used in the gem code for Monte Carlo neutronics calculations in reactors and other systems of complex geometry. In *Proceedings of Conference on the Application of Computing Methods to Reactor Problems*, pages 557–579, 1965.
- [61] Douglas R. Wyman, Michael S. Patterson, and Brian C. Wilson. Similarity relations for the interaction parameters in radiation transport. *Applied Optics*, 28(24):5243–5249, 1989.
- [62] Yonghao Yue, Kei Iwasaki, Bing-Yu Chen, Yoshinori Dobashi, and Tomoyuki Nishita. Unbiased, adaptive stochastic sampling for rendering inhomogeneous participating media. *ACM Transactions on Graphics*, 29:177:1–177:8, December 2010.
- [63] Kun Zhou, Qiming Hou, Rui Wang, and Baining Guo. Real-time KD-tree construction on graphics hardware. Technical report, Microsoft Research, April 2008.

A. CD contents

The contents of the accompanying CD are organized as follows:

- /AtmoVision — the demonstrating application installation (.zip file)
- /Misc — miscellaneous multimedial and other data
- /Sources
 - /AtmoVision — complete source code of the demonstrating application and the encapsulating Visual Studio 2008 project
 - /Thesis — complete \LaTeX source code of the thesis including the image data
- /Thesis — the thesis .pdf file

B. AtmoVision short user reference

B.1 System requirements

The recommended system configuration for running AtmoVision is the following:

- CPU: x86 or x64 @ 1.5 GHz
- Physical memory: 1.5GB RAM
- GPU: NVidia GeForce GTX 4xx or newer with CUDA Compute Capability 2.0 or higher
- HDD: 500MB free space (optionally with writing rights for the output .log file)
- OS: Windows 7 or Vista

It is very well possible that AtmoVision will run on older systems as well, but there is no guarantee it will perform as reported in Section 6.3 or run at all.

B.2 Installation

The installation procedure of AtmoVision is very simple:

- Unpack either the `AtmoVision.zip` or the `AtmoVision.7z` archive located in the `/Atmovision` directory on the accompanying CD to a target directory.
- Check the `AtmoVision/Resources/Dbs.xml` configuration file if the settings are suitable for the installation machine.
- Launch the application either with `AtmoVision/AtmoVision.exe` or one of the `AtmoVision/AtmoVision_XYZ.exe` where `XYZ` denotes suffix indicating the executable compilation options.

B.3 Usage

AtmoVision consists of just a single window without any GUI elements. All actions are performed simply with keyboard and mouse. Pressing F1 at any time toggles a textual tooltip with all keyboard and mouse controls listed.

The usage of AtmoVision is very simple. Upon launching the application the camera is positioned at the Earth orbit. Pressing F3 moves the camera right next to the cloud. The cloud at that moment will be in the middle of the Preparation phase (see Section 4.2) stopped between Steps 1b and 1c. Pressing Space will resume the simulation and start the cloud Rendering phase. Some of the more useful controls are:

- F1 displays AtmoVision controls
- F3 positions the camera near the cloud
- Space starts the simulation
- B toggles the cloud debug visualization
- PgUp/PgDown move the Sun
- W/S/A/D/Q/Z control the camera movement
- Left mouse button plus mouse movement pans the camera
- Middle mouse button plus mouse movement controls the camera view direction

- Right mouse button plus mouse movement revolves the camera around the cloud
- Mouse wheel moves the camera forward and backwards
- Ctrl+P saves a screenshot in the application directory
- Esc quits AtmoVision

C. Symbols and notation

The following tables presents all important symbols and abbreviations used throughout the text. The symbols ordered analogously to their order of appearance in the text, and grouped semantically.

Symbol	Physical dimension	Description
a / \vec{a}		scalar / vector quantity
λ	m	wavelength of light
σ_a	m^{-1}	absorption coefficient
σ_s	m^{-1}	scattering coefficient
σ_t	m^{-1}	extinction coefficient
σ_T	m^{-1}	majorant extinction coefficient
σ_e	m^{-1}	emission coefficient
α		scattering albedo
η	m^{-3}	medium particle density
F		medium amplitude function
f	sr^{-1}	medium phase function
r	sr^{-1}	surface BRDF
θ	rad	scattering or zenith angle
ϕ	rad	azimuthal angle
g		scattering asymmetry coefficient
\vec{x}	[m,m,m]	point in Cartesian space
$\vec{\omega}$	[m,m,m]	ray direction
\vec{n}	[m,m,m]	surface normal
ξ		unit random variable
τ		extinction function
l		light path in a medium
d, t, u	m	ray distance parameter
L	$W \cdot m^{-2} \cdot sr^{-1}$	radiance
T		transmittance function
B_i		general basis function
c_i		general basis coefficient
y_l^m		real spherical harmonic function
P_l^m		Legendre polynomial
N_l^m		SH normalization coefficient
Λ		number of SH bands used for function approximation
M		photon map
G_i		i -th photon generation
p		number of concurrent photon generations / photon map update period
N_T		total number of photons in photon map
N_G		number of photons in one generation
N_t		number of photons shot by one CUDA thread during photon update
Φ_T	W	total radiant flux present in a scene

Symbol	Physical dimension	Description
V / \vec{V}		scalar / vector field
ρ		(cloud) relative density function
V_ρ		(cloud) density field
V_g		(cloud) gradient magnitude field

Abbreviation	Meaning
AABB	Axis-Aligned Bounding Box
BRDF	Bidirectional Reflectance Distribution Function
DOM	Discrete Ordinates Method
FPS	Frames Per Second
HDR	High Dynamic Range
LDR	Low Dynamic Range
NN	Nearest Neighbour
(P)	Preparation
PDF	Probability Density Function
PM	Photon Mapping
(R)	Rendering
RE	Rendering Equation
SAH	Surface Area Heuristic
SH	Spherical Harmonics
VPC	Virtual Photon Collector
VPL	Virtual Point Light
VRE	Volume Rendering Equation