Masaryk University
Faculty of Informatics

Real-time rendering of volumetric clouds

Master's Thesis

Juraj Pálenik

Brno, Spring 2016

Masaryk University
Faculty of Informatics

Real-time rendering of volumetric clouds

Master's Thesis

Juraj Pálenik

Brno, Spring 2016

Declaration

Hereby I declare that this paper is my original authorial work, which I have worked out on my own. All sources, references, and literature used or excerpted during elaboration of this work are properly cited and listed in complete reference to the due source.

Juraj Pálenik

Advisor: Mgr. Jiří Chmelík, Ph.D.

Acknowledgement

I thank my advisor Mgr. Jiří Chmelík, Ph.D. for his great help with my thesis. I would also like to thank the members of Bohemia Interactive co., namely Ing. Jan Zelený, Bc. Dan Doležel, Peter Benýšek, Ing. Ondřej Martinák, for their patience and priceless insights.

Abstract

This thesis describes the principles of real-time rendering of translucent volumetric data and researches a method for real-time rendering of dynamic volumetric clouds shaded with dynamic light for purposes of an interactive flight simulation. The described techniques are implemented in a demonstration Unity game engine project.

Keywords

clouds, real-time rendering, volumetric rendering, dynamic lighting of volume, raymarching, single scattering, multiple forward scattering, Unity game engine

Contents

List of Figures

Introduction

Clouds are an indispensable part of our everyday landscape. The graphic interactive applications that strive to reproduce realistic environments are bound to include clouds in all their majesty. This includes dynamic shapes, physically based lightning, volumetric behaviour. Rendering volumetric dynamic clouds with dynamic lightning in real-time programs has only recently been made possible, but not for a dynamic camera inside of the volume of clouds.

The goal of this thesis is to exploit the possibility of rendering volumetric clouds in a general setting, where both camera and dynamic objects can be placed inside of the volume of clouds and the lightning of the scene is computed dynamically. Such solution would allow for use in real-time flight simulations, or could be used for rendering of low altitude clouds, fog and mist. The demand for such work was voiced by Bohemia Interactive company and during the implementation I was also consulting with the company's developers.

Several approaches for rendering volumetric clouds has been already researched and published with various render-time/quality trade-offs. Among the last successful approaches is the work of the programmers from Guerrilla Games company, who presented their results on the SIGGRAPH 2015 Advances in Real-Time rendering in Games. The solution utilized a raymarching algorithm with shading of the clouds using single scattering approximation. Their presentation titled 'The Real-time Volumetric Cloudscapes of Horizon: Zero Dawn' [1], did not, for obvious reasons, include particular implementation of this method. A master's thesis inspired by this work, was already done on Masaryk University by Mgr. Michal Pavelka [2]. In his work as well as in the game HORIZON, the clouds are located in a bounding volume inaccessible to the camera.

In this thesis a rigorous formal approach to the topic has been adapted, exploring the principles of light scattering and it's computer simulation. The first chapter features a brief overview of known cloud modelling methods. The second chapter contains an explanation of

light propagation through the clouds and lists several ways to approach the simulation of the phenomenon. The third chapter explains the raymarching algorithm that is the core part of the implementation, with couple of modifications proposed by various authors. The fourth chapter considers further improvements and modification of the usual raymarching algorithm, subject to this particular problem. The fifth and final chapter details the peculiarities of implementation and the results of my method.

1 Clouds

For the purposes of this work, a cloud is a volume of scattering precipitating medium, which forms various shapes in the skies. Their visual properties strongly depend on the light interaction inside of this volume. In the following chapter is a brief description of their physical properties and a list of standard ways to model the clouds.

1.1 Physical properties

The clouds are formed from water droplets suspended in the air. The conditions for forming of the water droplets are sufficient humidity, condensation temperature and the presence of dust particles called condensation centres.

An interesting observation is that the extinction of the light in the cloud does not depend on the distribution of the sizes of the water droplets. The light extinction is also slightly higher for longer wavelengths of visible light and thermal radiation. [3]

For the modelling of clouds, only the cloud density is therefore important. From the thermodynamics theory of gases and the physical model of the atmospehre, the cloud creation mechanism can be simplified into several useful axioms. The exact wording was taken from the Guerilla games persentation [1]. Further information regarding cloud physics can be found for example in the books [4, 5].

- Density increases at lower temperatures
- Temperature decreases over altitude
- High densities precipitate as rain or snow
- Wind direction varies over altitude
- Clouds rise with heat from the earth
- Dense regions make round shapes as they rise
- Light regions diffuse like fog
- Atmospheric turbulence further distorts clouds

1.2 Modelling

In this section are listed some of the most notable methods for modelling cloud shapes. Historically, the approach of Geoffrey Y. Gardner was very successful, who in 1985 used procedurally textured quadratic surfaces to model different types of clouds. These were rendered using surface based lightning model. Gardner positioned himself as he says '... we have adopted the strategy of the impressionist painters: to represent the essence of a scene as efficiently as possible.' ([6]) Regarding the previous work of his time he says 'In general, these efforts attempted to model cloud physics accurately and therefore involved fairly rigorous mathematics.' ([6])

Since I acknowledge that using artistic methods may prove more efficient than complicated simulations, my work is situated to be able to render any cloud data provided in volumetric texture, be it of artistic or mathematical origin.

Implicit volumes

Kniss et al. proposed that volumetric clouds can be modelled using implicit volume and a perturbation texture. [7] The idea is to define a volumetric data of the cloud density using analytic expression, and shift every value in pseudo-random direction. This is accomplished by querying the analytic function in a volume positions distorted by an offset generated by 3D vector noise texture.

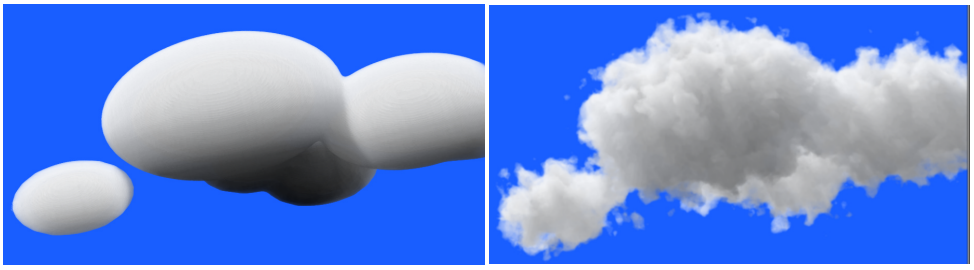

Figure 1.1: Implicit volume perturbation cloud, implicit surface on the left and perturbed result on the right, the picture was taken from [8].

Particle clouds

The idea of particle based clouds is to build up the shape from smaller uniform puffs. This approach was used also by Mark J. Harris in 2001 and is described in his work [9]. Egor Yusov made a recent (2014) reimplementation of the method with unique particle blending aimed for more realistic shading [10]. A bachelor thesis at Masaryk University using particle approach was made by Jakub Křoupal [11], and a master's thesis by Vít Kučera at 'Vysoké Učení Technické, Brno' [12]. All the implementations used precomputed lightning to achieve real-time performance.

Polygon clouds

In 2008 Antoine Bouthors et al. tought that '... volume rendering cannot be achieved with enough resolution for sharp clouds such as cumulus' ([13]). Thus they used a polygon based model together with Hypertexture introduced by Ken Perlin and Eric M. Hoffert [14]. The cloud bounding volume is defined by manually created polygon model, which is then combined with three dimensional noise texture.

Voxel clouds

A straightforward but memory intensive approach is to store the density values of the cloud into a voxel grid. The particular data can be obtained by cell automata simulation introduced by Dobashi et al. [15], or fluid dynamics computation used by Kajiya et al. [16], or even by meteorological data from cloud measurements. A good trade-off between a large memory footprint and the discrete resolution of the grid is essential.

Procedural clouds

The book of David S. Ebert et al., 'Texturing and modeling: a procedural approach' gives a good insight into the topic. The clouds are generated using noise functions, such as Perlin noise, or Worley noise, combined with fractal Brownian motion technique (fBm) [17]. I have used a fBm Perlin noise precomputed into three dimensional texture, reusing the data and the approach from Pavelka's thesis [2].

2 Shading

As opposed to traditional surface based objects, clouds do not have a well defined boundary. The cloud shapes and colours are caused by light interactions inside the volume. In order to visualize this effect it is necessary to understand the light propagation through the volume.

A very thorough explanation is to be found in chapter 12 of the book 'Thermal Radiation Heat Transfer' [18]. A good explanation concerning volume rendering can be found in 'Real-Time Volume Graphics' [8]. In the following chapter is the basic information as I understand the topic, however some formulations may appear similar to the above mentioned publications.

2.1 Volume light integral

In general, the interaction between light and a participating medium can be described by three phenomena – absorption, emission and scattering. The light is assumed to be travelling through free space unless a water droplet, or a dust particle is found in its way, upon which some of the following happens.

Absorption The light energy is consumed by the medium and no longer accounts for the visual effect. This can be caused by black body particles at low temperatures, that absorb all the light and emit it back in different wavelength, such that is not perceived by human eye. (Also known as thermal emission, or heat.)

Emission The light is spontaneously emitted by the matter due to Planck's black body radiation, for example ash particles at very high temperatures contained in smoke during explosions.

Scattering The incoming light is redirected onto a different path. In general, the wavelength of the incoming light may change during the scattering, but in order to be able to separate the simulation for different wavelengths this possibility is neglected.

The listed effects are described mathematically by computing the intensity of light at a given point in space in a given direction, denoted

by $L(x, \vec{\omega})$.[1] Considering an elementary cylindrical volume around the light direction $\vec{\omega}$. (See figure 2.1.)

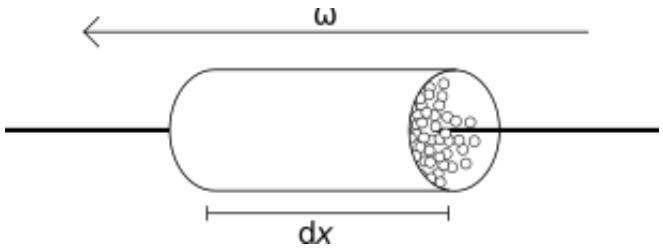

Figure 2.1: The elementary cylindrical volume filled with participating medium

The light intensity will drop from one end of the volume to the other by combination of absorption and scattering. It will be enhanced by amount of light emitted by the volume element, and the intensity of light passing form a different direction, scattered in direction $\vec{\omega}$. This can be described using differential equation

$$\frac{\partial L(x, \vec{\omega})}{\partial x} = -(\alpha(x) + \sigma(x))L(x, \vec{\omega}) + L_e(x, \vec{\omega}) + L_i(x, \vec{\omega}), \qquad (2.1)$$

where α describes absorption, σ describes scattering, $L_e(x, \vec{\omega})$ is the amount of light emitted in point x in direction $\vec{\omega}$, $L_i(x, \vec{\omega})$ is the amount of light incoming from the surrounding volume scattered in direction $\vec{\omega}$.

The notation $L(x, \vec{\omega})$ describes the light travelling in a straight line with $\vec{\omega}$ being unit direction vector and x being the parameter defining distance from a fixed point $\vec{x}_0$. This allows writing the directional derivative along $\vec{\omega}$ as $\frac{\partial L}{\partial x}$. (See Figure 2.1.)

To render the image, the light intensity incoming from the direction $\vec{\omega}$, reaching the camera screen positioned at $\vec{x}_a = a\vec{\omega} + \vec{x}_0$, has to be computed. (See Figure 2.2.) The boundary condition for the linear

1. The quantity in consideration is correctly called radiance and is measured in $[\mathrm{W} \cdot \mathrm{sr}^{-1} \cdot \mathrm{m}^{-2}]$. Radiance is a quantity which does not change its value when the light is travelling through free space. In this work a more intuitive name 'intensity' is used, or simply 'light'.

first order partial differential equation will be the intensity of light in direction $\vec{\omega}$ at the end of the ray $\vec{x}_b = b\vec{\omega} + \vec{x}_0$. Denoting $L(b, \vec{\omega}) = L_b(\vec{\omega})$, the backgroud light.[2] Solving the Equation (2.1) by integration factor gives

$$L(a, \vec{\omega}) = L_b\, e^{-\tau(b,a)} + \int_b^a e^{-\tau(x,a)} (L_e(x, \vec{\omega}) + L_i(x, \vec{\omega}))\, \mathrm{d}x, \qquad (2.2)$$

where $\tau(t_1, t_2) = \int_{t_1}^{t_2} [\sigma(t) + \alpha(t)]\, \mathrm{d}t$. The Equation (2.2) is called Volume-Light integral.

The useful information is that the amount of light gets exponentially reduced with $\tau(x) = \tau(x, a)$, sometimes also called the optical depth or thickness. The emission represented by L_e can be as simple as $\epsilon\rho(x)$, a constant factor modified by the volume density, or can have more complex form, as described in section 5.1 of SHADERX^5 [19].

The next step is to deal with the complexity hidden in the function L_i. The definition of L_i says it is the amount of light incoming from the surrounding volume scattered in direction $\vec{\omega}$. Denoting $I(x, \vec{\omega}, \vec{\omega}')$, the intensity of incoming light from direction $\vec{\omega}'$ at the point $\vec{x} = \vec{x}_0 + x\vec{\omega}$.

$$L_i(x, \vec{\omega}) = \frac{1}{4\pi} \oint \sigma(x, \vec{\omega}) p(x, \vec{\omega}, \vec{\omega}') I(x, \vec{\omega}, \vec{\omega}')\, \mathrm{d}\Omega', \qquad (2.3)$$

integrating all the incoming light I from every direction $\vec{\omega}'$ multiplied by the probability of scattering event σ and a phase function p. Introducing the phase function as a distribution of how much light is redirected towards which direction being the goal here, since it is a well documented property of the material.

2. $\vec{x}_b$ may be position outside the volume where the light intensity equals to the one produced by the light source, or at the surface of the object where surface shading can be computed. Either way it is the light that would have been seen without the presence of the clouds.

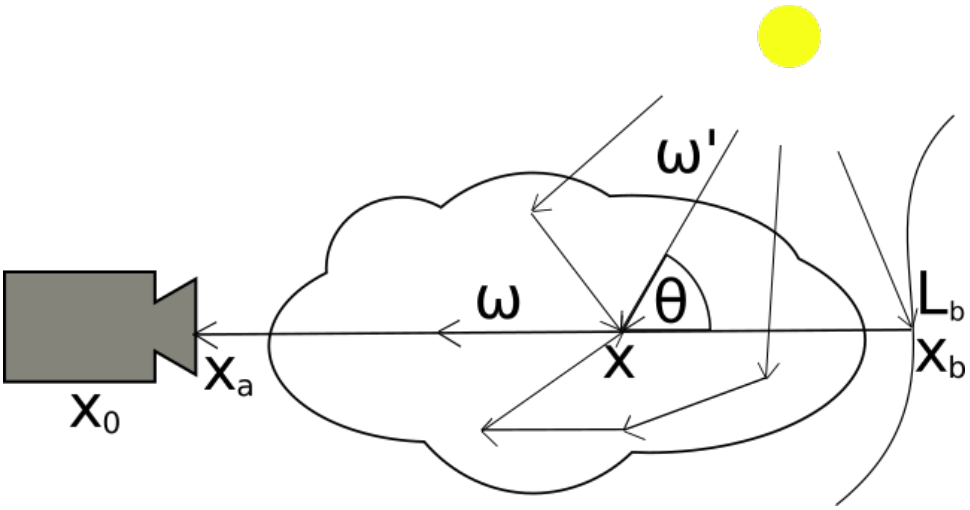

Figure 2.2: Light propagation through volume, x_0 is the position of the camera, $\vec{x}_a$ is the position where the light intensity incoming from direction $\vec{\omega}$ is to be computed, x is the variable from Equation (2.1), L_i would be all the light incoming from all possible directions $\vec{\omega}'$, L_b is the background light incoming from the position b in a straight line.

2.2 Phase function

The phase function is 'gears and wheels' of the light scattering and can be compared to bidirectional reflective distribution function (BRDF) for surface shading. The function describes the amount of energy being distributed towards different scattering vectors, or equivalently the amount of energy being received under different impact angles.

The important part is separating the scattering mechanism (phase function) from the intensity of light being scattered (I) and from the amount of scattering happening at given position ($\sigma(\vec{x})$), because the phase function of different materials can be studied and modelled by various physical theories. Most of the phase functions are axially symmetric, so only the angle between the impact vector and outgoing vector ($\cos(\theta) = \vec{\omega} \cdot \vec{\omega}'$) is enough to describe the phase function. Some of the phase functions relevant to the cloud rendering are described below, the plots can be found in Figure 2.3.

Isotropic phase function. The trivial case, where the light scatters uniformly towards all directions. Very useful for analytical solutions, since the phase function factor can be wholesomely omitted from the Equation (2.3).

Rayleigh. Rayleigh scattering is the process where light scatters on the molecules or on particles much smaller than the wavelength of the light. The first to describe this effect was John William Strutt, 3rd Baron Rayleigh, in 1871. The corresponding phase function is

$$p_R(\theta) = k\frac{1 - \cos^2(\theta)}{\lambda^4}, \tag{2.4}$$

where k is a constant factor, λ is a wavelength and θ is the angle between the incident and outgoing light direction $\cos(\theta) = \vec{\omega} \cdot \vec{\omega}'$. Since the phase function is wavelength dependent it is responsible for the blue color of the sky. There are much more pronounced effects than molecule scattering in clouds. This function was used by Harris er al. [9]

Henyey–Greenstein. In 1941 Henyey and Greenstein published an astronomical article [20] concerning light scattering on dust particles in intergalactic observations. The phase function was de-

rived based on the mechanism of light scattering on objects of size much bigger than the wavelength of the visible light, reading

$$p_{HG}(g,\theta) = \frac{1}{4\pi}\frac{1-g^2}{(1+g^2-2g\cos(\theta)^{3/2}}, \tag{2.5}$$

where g is a parameter and θ is again the angle between the incident and outgoing light direction $\cos(\theta) = \vec{\omega}\cdot\vec{\omega}'$.

Normalised Henyey–Greenstein. In the book 'Real-Time Volume Graphics', chapter 6.2, the author suggests using 'renormalized' Henyey-Greenstein phase function for single scattering approximations

$$p'_{HG}(g,\theta) = \frac{p_{HG}(g,\theta)}{p_{HG}(g,0)} = \frac{(1-g)^3}{(1+g^2-2g\cos(\theta)^{3/2}}, \tag{2.6}$$

which does not integrate to unity over the 4π steradians, but its value is restricted to interval $[0,1]$ for every direction.

Mie. Gustav Mie and Ludvig Lorentz derived the theory for electromagnetic wave scattering on the particles of size similar to the wavelength of the incident light. This is so far the most accurate approximation for the light scattering in clouds and it is also computationally the most intense. A great help for computing the scattering distribution is MiePlot[3]. The plotting of the function can be found in Figure 2.3.

3. A computer program for scattering of light from a sphere using Mie theory & the Debye series, available online at `http://www.philiplaven.com/mieplot.htm`.

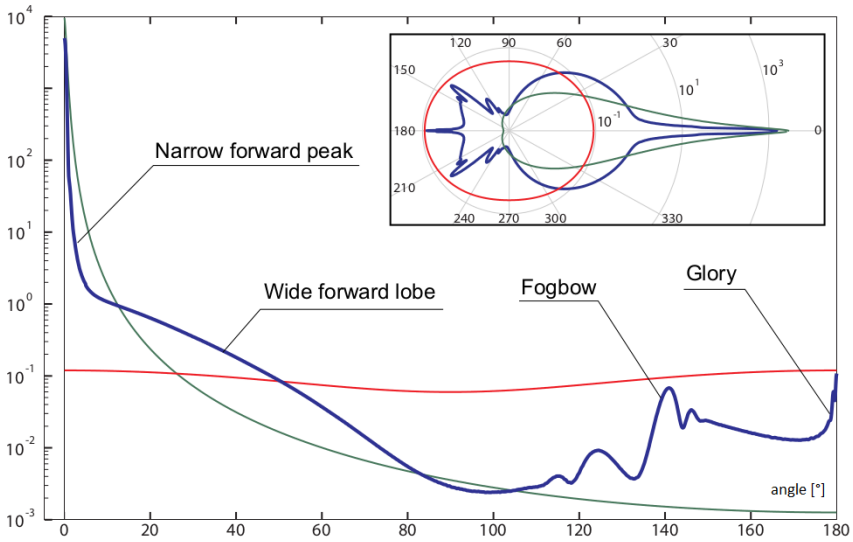

Figure 2.3: Plotting of phase functions taken from [13]. Log plots (inset: polar log plots) of commonly used phase functions. Red: Rayleigh. Green: Henyey-Greenstein with g = 0.99. Blue: Mie.

2.3 Solving the integral

In the first section of this chapter was derived the theory of light propagation through clouds. The complexity of the obtained Volume-Light integral allows only for numerical solutions. Further restriction is that the simulation has to be computed in real-time to abide with dynamic lightning of the scene. This section briefly describes the various approaches of approximating the Equation (2.2) by the relevant authors.

An accurate way to solve the volume light integral is considered to be Monte Carlo path tracing. This, however, is far from real-time rendering. A significant success has been achieved by Anoine Bouthors et al. by case studying the results of Monte Carlo integration and deriving heuristics to solve the multiple scattering in interactive frame rates (10 fps). Harris et al. have made use of multiple forward scattering in pre-computation of light distribution and Klaus Engel et al. have made use of multiple forward scattering in 'half angle slicing'. The programmers from Guerilla Games have fine-tuned the visual results of the single scattering approximation. In the following list these approaches are commented and my approach is established.

Bouthors et al. The most advanced physical simulation of light transportation in real-time. In the case study they have concluded that up to 30 scattering events are of anisotropic character and are necessary for realistic simulation of the light propagation [13]. The results (Figure 2.4) are astonishing, but the price is paid in performance. (Only 10 fps.) The data model is based on polygon clouds with hypertexture, which is not useful for fast prototyping of clouds and most importantly the light propagation is computed surface to surface, hence an obstacle for dynamic camera inside the volume of clouds.

Harris et al. Mark J. Harris and Anselmo Lastra implemented particle clouds system with realistic shading, using algorithm that approximates multiple forward scattering in a preprocess, and first order anisotropic scattering at runtime. The method is suitable for a flight simulator and produces high frame rates. The associated math is explained in section 3.2. The only drawback of this approach is the statically precomputed light.

Engel et al. In the book 'Real-Time Volume Graphics', Chapter 6, [8] a volume rendering method called 'Half angle slicing', or 'Ping-Pong' method is described. It uses a technique similar to multiple forward scattering by Harris et al. The light is computed in lock step with the color, for a cutting plane travelling through the volume at half angle between the camera and the light. The light intensities in the volume are evaluated only once, making the theoretical running complexity in $\mathcal{O}(n)$. They also obtained great results making the phase function a function of the density through the parameter g in Henyey-Greenstein phase function, such as $g = 1 - \alpha$, where α is opacity of the material. (Figure 2.6) The method however locks the shade resolution with the color resolution, which does not scale well.

Guerrila Games An example of raymarching algorithm fine-tuned to perfection. The significant part of the success is due to a unique modelling technique, which dynamically creates realistic shapes. The shading integral is approximated by sampling 6 steps towards the sun. The multiple scattering is mimicked by jittering the light steps in a cone. The Beer's law is also modified by a 'Powder' function named after powder sugar, to account for multiple scattering in arbitrary direction. The clouds are rendered in a bounding volume far and above the player. The rendering time was cut down to 2 ms. [1]

From the listed approaches I have chosen the raymarch computation, which I would like to equip with some analytical approach. The goal is to improve the method to be able to fly through the clouds.

Figure 2.4: The results of multiple scattering by Bouthors et al., the picture was taken from [13].

Figure 2.5: The results of single scattering raymarching by Guerilla games. The clouds from the game Horizon: Zero dawn, the picture was taken from their presentation [1].

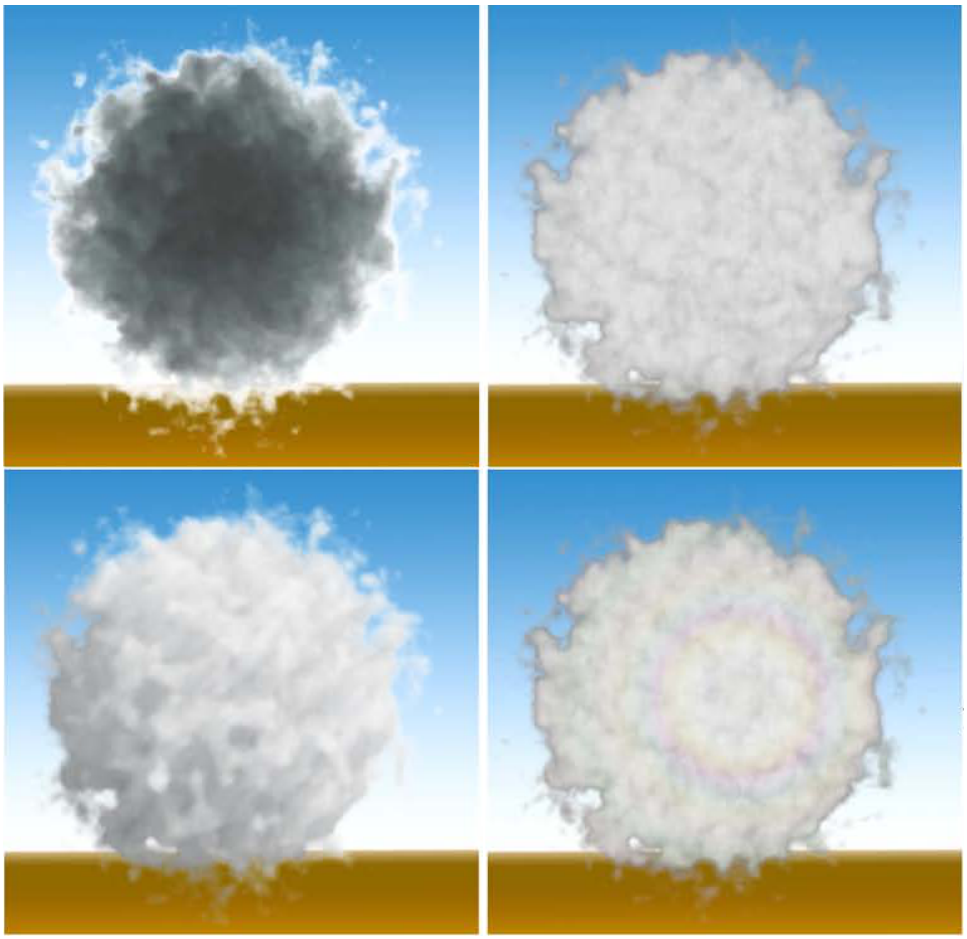

Figure 2.6: Silver lining and the dark edges simulated by inhomogeneous phase function, from the book Real-Time Volume Rendering, [8]. All the pictures were shaded using Henyey-Greenstein phase function with $g = 1 - \alpha$ of the volume element. Top left – cloud lit from behind, top right – lit from the front, bottom left – lit from above at 45°, bottom right – addition of Mie phase function creating the glory effect.[5]

5. The image was taken from `http://www.cg.informatik.uni-siegen.de/data/Tutorials/EG2006/RTVG07_GlobalIllumination.pdf`

3 Raymarching

The general idea of raymarching is to directly visualize the volumetric data. The final image is composed by approximating the volume light integral from Chapter 2 for rays going from the camera through a projection screen into the frustum volume. The prerequisite for the raymarch is the ability to evaluate volume properties at given positions and to execute the computation in alpha pass, as a post process, to be able to blend the clouds into an existing scene. The following chapter describes the raymarching algorithm and its subsequent use and modifications.

3.1 Single scattering

The goal of this section is to give an insight into the simplest raymarching algorithm, especially the math behind, so the complexity cna be build later. The objective is to evaluate the volume integral from Equation (2.2). In order to do so a mapping between the volume data and the coefficients in Equations (2.1) and (2.3) has to be defined.

Density $\rho(\vec{x})$**.** First of all, the only volumetric data is going to be cloud density $\rho(\vec{x})$ at a given position $\vec{x}$. All of the remaining parameters are going to be derived from the density. This means that the cloud will be made of uniform 'cloud material'.

Emission $L_e(\vec{x})$**.** The cloud will not be emitting any light, so the emission coefficient is identically zero. $L_e(\vec{x}) \equiv 0$

Absorption $\alpha(\vec{x})$**.** The absorption coefficient is also going to be zero $\alpha(\vec{x}) \equiv 0$.

Scattering $\sigma(\vec{x})$ Scattering coefficient is going to be proportional to the density $\sigma(\vec{x}) = \sigma\rho(\vec{x})$.[1]

1. In order to keep the number of constants low, a customary abuse of notation is introduced. The $\sigma(\cdot)$ on the left is a function, while the σ on the right is a constant multiplied by the density function $\rho(\cdot)$.

Incident light $L(a, \vec{\omega})$**.** The incident light L_i is going to be evaluated for single scattering event as

$$L_i(x, \vec{\omega}) = \sigma\rho(x)p(\vec{\omega} \cdot \vec{\omega}_{sun})e^{-\int_{\varsigma(s)} \sigma\rho(s)\,\mathrm{d}s}, \tag{3.1}$$

where $\varsigma(s)$ is a curve connecting point $\vec{x} = x_0 + x\vec{\omega}$ to the sun.

Plugging the parameters into the Equation (2.2) results in

$$\begin{aligned} L(a, \vec{\omega}) &= L_b\, e^{-\tau(b,a)} + \\ &\int_a^b e^{-\tau(x,a)}\sigma\rho(x, \vec{\omega})p(\vec{\omega} \cdot \vec{\omega}_{sun})e^{-\int_{\varsigma(s)} \sigma\rho(s)\,\mathrm{d}s}\,\mathrm{d}x, \end{aligned} \tag{3.2}$$

remembering from Section (2.1) that $\tau(t_1, t_2) = \int_{t_1}^{t_2} \sigma\rho(t)\,\mathrm{d}t$. Choosing the simplest form of approximating the integrals become multiple Riemann sums. (One for every integral)

$$L = L_b e^{-T_n} + \sigma p \Delta x \sum_{i=0}^{N-1} e^{-T_i}\rho_i e^{-T'_i}, \tag{3.3}$$

where $T_i = \sigma\Delta x \sum_{j=0}^{i} \rho_j$ and $T'_i = \sigma\Delta s \sum_{j=1}^{k} \rho(\vec{x}_i + j\Delta s \cdot \vec{\omega}_{sun})$.

The factor $e^{-T'_i}$ represents a subroutine that evaluates the incident lightning at the point $\vec{x}_i$ taking k steps towards light, accumulating optical thickness and plugging it into an exponential function, reading

$$\mathcal{I}(\vec{x}_i) = \mathcal{I}_i = e^{-\sigma\Delta s \sum_{j=1}^{k} \rho(\vec{x}_i + j\Delta s \cdot \vec{\omega}_{sun})}. \tag{3.4}$$

In order to rewrite the rest of the Equation (3.3) in more programmer friendly manner a variable $\mathcal{E}_n = e^{-T_n}$ is introduced to represent the exponential function inside the main sum of Equation (3.3). (In code is this variable also called extinction factor, since it reduces the intensity of light propagating through cloud.)

$$\mathcal{E}_n = e^{-\sigma\Delta x \sum_{j=0}^{n} \rho_j} = \prod_{j=0}^{n} e^{-\sigma\rho_j\Delta x}, \tag{3.5}$$

which yields recurrent formulation

$$\begin{aligned} \mathcal{E}_0 &= 1, \\ \mathcal{E}_n &= e^{-\sigma\rho_n\Delta x}\,\mathcal{E}_{n-1}. \end{aligned} \tag{3.6}$$

After substituting the variables back into the Equation (3.3) only a single sum remains. Denoting the value $\mathcal{C}_n = \sigma p \Delta x \sum_{i=0}^{n} \mathcal{E}_i \rho_i \mathcal{I}_i$, which easily reads as another recurrent equation

$$\begin{aligned} \mathcal{C}_0 &= 0, \\ \mathcal{C}_n &= \mathcal{C}_{n-1} + \sigma p \rho_n \mathcal{E}_n \mathcal{I}_n \Delta x. \end{aligned} \tag{3.7}$$

A remark should be made towards the meaning of $\mathcal{E}_n$. Observing that it is the direct result of the Equation (2.1) and that it expresses the portion of light intensity in the given distance from camera as

$$\mathcal{E}(x) = e^{-\int_0^x \sigma \rho(s)\, \mathrm{d}s}.$$

This is also known as the Lambert-Beer law [18]. In other words it is the transparency of the volume between camera and the position $\vec{x}_n$. Opacity and transparency are related as $\alpha = 1 - \mathcal{E}$. The Equation (3.3) then reads $L = (1 - \alpha) L_b + \mathcal{C}_N$, which can be recognized as blending the cloud colour with background colour using cloud alpha.

The result of this derivation is summarised in Algorithm 1

An interesting remark is that the first to use this method to render clouds were Kajiya et al. in 1984 [16]. The computation used precomputed light intensities stored in an array, spherical harmonics to address the phase function and they used a fluid dynamics simulation to model the cloud shapes at resolution of 10 by 10 by 20. The work was titled 'Ray tracing volume densities' and the resulting image was done on an IBM4341 at 512 X 512 resolution, with CPU computation time of 2 hours.

Algorithm 1: Raymarch

input : ray origin, ray direction, frame buffer, σ, p, Δx
output: color of the pixel

```
T ← 1 ;          // The accumulating variable for transparency
color ← 0 ;              // The accumulating variable for color
for no. of steps do
    den ← query the density at current position;
    ext ← e^(−σ×den×Δx) ;                          // due eq. (3.6)
    I ← evaluate incident lightning I(x_i);
    T ← T × ext;
    col ← σ × p × den × I × T × Δx ;               // due eq. (3.7)
    color ←color + col;
    move the position by Δx;
return T × backgroundColor + color;
```

3.2 Multiple forward scattering

In this section the cloud shading used by Harris et al. [9] and my analysis and adaptation of the method is explained.

A very interesting proposition has been made by Harris et al. They observed that Equation (2.2) can be also used for computing the incident light $\mathcal{I}(\vec{x}_i)$, since a dummy camera could be positioned into a point in the cloud and aimed towards the sun. The light intensity captured by this dummy camera would be then scattered towards the real camera creating the final image. (See Figure 3.1.)

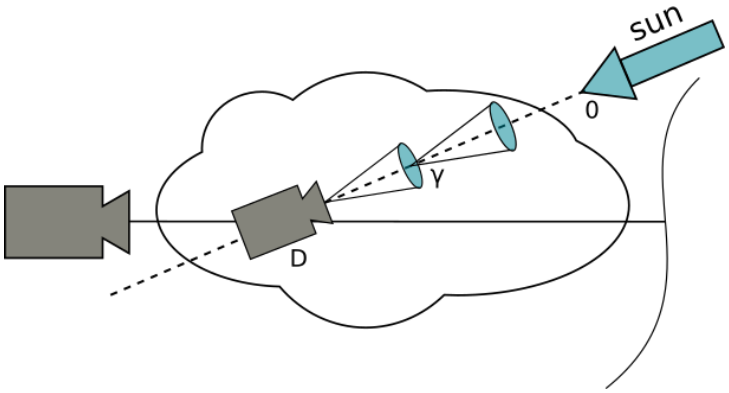

Figure 3.1: The idea of multiple forward scattering. The light is accumulated from a small angle γ around the light direction. The light intensity in blue discs is considered to be constant.

Harris et al. used a slight variation of the Equation (2.2), where the light is followed from the origin (sun) towards the dummy camera. The dummy camera is at distance D from the sun and is looking directly towards sun. The emission and absorption coefficients are again zero. Denoting the integral $\mathcal{I}(D, \vec{\omega}_{sun})$, to represent the incident light.

$$\mathcal{I}(D, \vec{\omega}_{sun}) = \mathcal{I}_0 e^{-\int_0^D \sigma\rho(s)\,\mathrm{d}s} + \int_0^D L_i(x, \vec{\omega}_{sun}) e^{-\int_x^D \sigma\rho(s)\,\mathrm{d}s}\,\mathrm{d}x, \quad (3.8)$$

where L_i represents the gathered light from all directions as defined in Equation (2.3).

Disregarding the other directions, integrating only in a small angle γ around the light direction $\vec{\omega}_{sun}$, treating the incident light from the small angle as constant. Equation (2.3) for compactness.

$$L_i(x,\vec{\omega}) = \frac{1}{4\pi}\oint \sigma(x,\vec{\omega})p(x,\vec{\omega},\vec{\omega}')I(x,\vec{\omega},\vec{\omega}')\,\mathrm{d}\Omega'.$$

Since everything in this integral, except the phase function, is considered to be constant with respect to the small angle γ, the integration is quite simple.

$$L_i(x,\vec{\omega}_{sun}) = \frac{\gamma}{4\pi}\sigma p'\rho(x,\vec{\omega}_{sun})\mathcal{I}(x,\vec{\omega}_{sun}), \tag{3.9}$$

where p' is integrated phase function in a small angle γ around the forward direction (See Figure 2.3, note the forward peak.) This is very important, because after substituting the L_i into the Equation (3.8) the $\mathcal{I}$ is evaluated for the same angle on both sides.

$$\begin{aligned}\mathcal{I}(D,\vec{\omega}_{sun}) &= \mathcal{I}_0 e^{-\int_0^D \sigma\rho(s)\,\mathrm{d}s} + \\ \int_0^D \frac{\gamma}{4\pi}\sigma p'\rho(x,\vec{\omega}_{sun})\mathcal{I}(x,\vec{\omega}_{sun})e^{-\int_x^D \sigma\rho(s)\,\mathrm{d}s}\,\mathrm{d}x.\end{aligned} \tag{3.10}$$

Using once again the Riemann sum to solve the integral. The first integral will be approximated with the right Riemann sum, the second by the left Riemann sum and the last one again with right Riemann sum. Choosing the left and right sums makes no theoretical difference, since there should be infinitely many intervals, but it allows for simple formula in the end.

$$\mathcal{I}_N = \mathcal{I}_0 e^{-\sum_{i=1}^{N}\sigma\rho_i\Delta x} + \sum_{i=0}^{N-1}\frac{\gamma\sigma p'}{4\pi}\rho_i\mathcal{I}_i\Delta x e^{-\sum_{j=i+1}^{N}\sigma\rho_j\Delta x},$$

Introducing some useful substitutions and making use of multiplications of exponentials results in

$$\mathcal{I}_N = \mathcal{I}_0\prod_{i=1}^{N}e^{-\eta_i} + \sum_{i=0}^{N-1}g_i\prod_{j=i+1}^{N}e^{-\eta_j}, \tag{3.11}$$

where $\eta_i = \sigma\rho_i\Delta x$ and $g_i = \frac{\gamma}{4\pi}p'\eta_i\mathcal{I}_i$. Now explicitly writing the last (n-th) part from both expressions and taking out the common factor.

Also changing the N to n, since the previous equation was meant to represent the whole integral from 0 to D, now it represents the intermediate results instead.

$$\begin{aligned} \mathcal{I}_n &= e^{-\eta_n} \left(\mathcal{I}_0 \prod_{i=1}^{n-1} e^{-\eta_i} + \sum_{i=0}^{n-2} g_i \prod_{j=i+1}^{n-1} e^{-\eta_j} \right) + g_{n-1} = \\ &= e^{-\eta_n} \mathcal{I}_{n-1} + g_{n-1} = \left(e^{\eta_n} + \frac{\gamma}{4\pi} p' \eta_{n-1} \right) \mathcal{I}_{n-1}. \end{aligned} \tag{3.12}$$

Observing that the expression in the huge parentheses is actually $\mathcal{I}_{n-1}$, yields a recurrence equation.

Harris et al. used this in precomputed light for particle generated clouds, averaging the values of $\mathcal{I}_n$ for neighbouring pixels, thus achieving a little blurring of shades.

Analysis and adaptation of the method

Direct use of multiple scattering for Algorithm 1 would be ineffective, since the values of computed light would not be shared between pixels (rays), nor between the frames, and the subroutine $\mathcal{I}$ (from Equation (3.4)) would get too slow for real-time rendering. Instead, the effect of multiple scattering is analysed and simulated by other means.

Plotting the amount of light integrated by the multiple scattering and comparing it against the single scattering yields an interesting outcome. In the chart at the page 26 (Figure 3.2) there are three plots of different light integration approaches.

The values in the chart represent the intensities of light at respective positions for a single ray traversing the sphere of density inversely proportional to the radius. $\rho(t) = \{1 - |t-1|, t \in (0,2);\ 0 \text{ otherwise}\}$. (Black line in the figure.)

200 steps were evaluated inside of the cloud volume. $\Delta x = 0,01$. The blue line represents the amount of light using multiple scattering as by Equation (3.12) with parameters $\sigma = 3$, $\gamma p' = 0,7 \times 4\pi$. The yellow line was computed using Equation (3.4), with the same parameters. The orange line was computed using again the Equation (3.4), but with hand picked $\sigma = 0,9$ to mimic the multiple forward scattering as closely as possible.

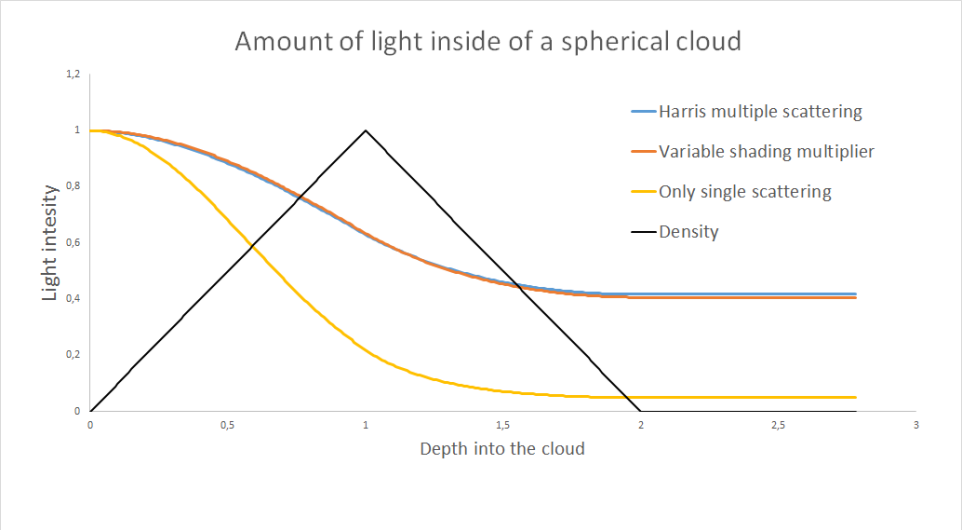

Figure 3.2: Plotting of the effect of the multiple forward scattering compared with single scattering and single scattering with user defined sigma for shading.

Based on this analysis, I have concluded that the multiple forward scattering computation results in significantly more light being propagated through cloud, as can be also seen on the pictures from article by Harris et al. [9] (See Figure 3.3), but also that the results should not be too different from the case where the clouds are shaded using single scattering with coefficient of scattering being smaller for the shading than for the light accumulation. Rather than working out the direct relation between the shading coefficient for every phase function I let the user interactively modify the parameter σ' changing the amount of incident light inside of the volume of cloud. This is combined with taking a fuzzy path towards the sun rather than a straight one, which results in blurring of the shades. The path followed is contained in cone with axis aimed towards the sun and the aperture[2] of 40 degrees.

2. Angle at the apex

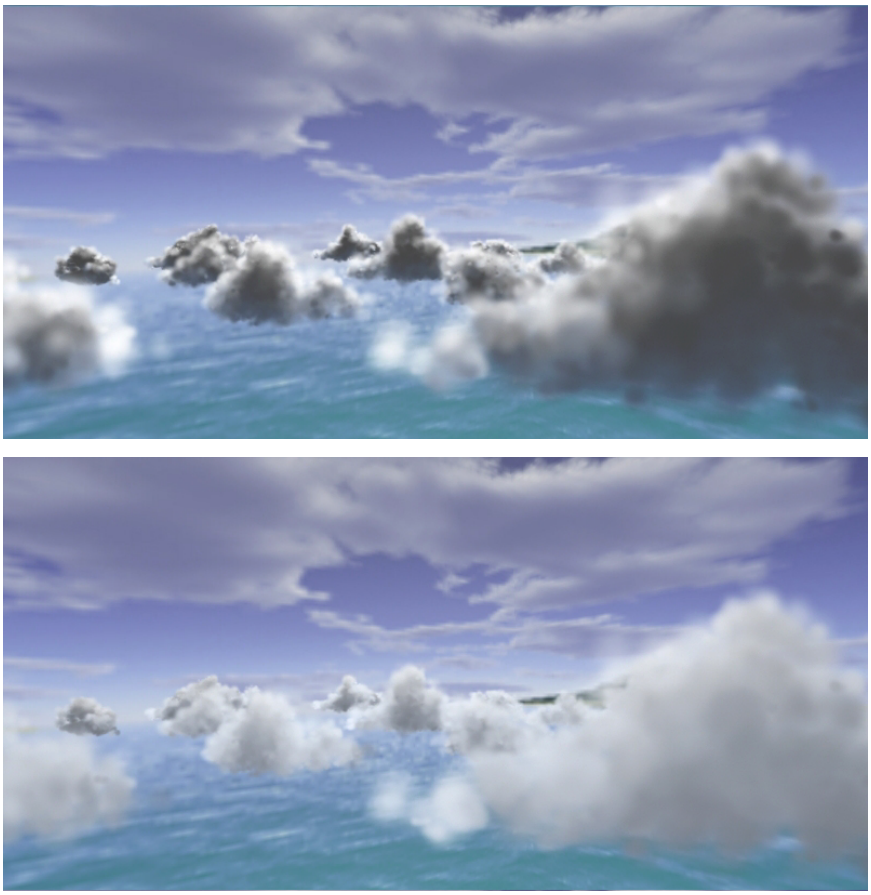

Figure 3.3: Single scattering (above) compared to multiple forward scattering (below), the picture was taken from Harris et al. [9].

3.3 General multiple scattering

The complexity of the general multiple scattering does not allow for a rigorous solution in a real-time simulation, the next section therefore contains only heuristics that attempt to improve the visual effect.

Powder function

The idea was taken from [1] and the reasoning is that the general case of multiple scattering will cause the cloud to 'collect' light inside the volume. Their *ansatz* was that this would be described by 'Powder' effect

$$\mathcal{P}(\tau) = 1 - e^{-k\tau}, \tag{3.13}$$

where τ is the optical thickness and k is a parameter. Supposedly, the Beer's law should be modified by multiplying with 'Powder' function. The new shading model would then read

$$\mathcal{I}_{BP}(\tau) = e^{-\sigma'\tau}\left(1 - e^{-k\tau}\right). \tag{3.14}$$

The effect is somewhat similar to the slight shading of directly illuminated sides and can bring out more details when viewing direction approaches light direction. The authors originally proposed that this dependency on view and light angle (θ) should be made explicit, but they were not specific on the way how to achieve this. I propose the formula $(\cos(\theta) + 1)/2$, so the modified equation reads

$$\mathcal{I}_{BP}(\tau) = e^{-\sigma'\tau}\left[1 - \frac{\cos(\theta) + 1}{2}\, e^{-k\tau}\right]. \tag{3.15}$$

Emission

After implementing the above mentioned approaches, even thought the clouds appear to be properly shaded from outside, upon entering the cloud volume, the colour would be unnaturally dark. In order to counter this I have implemented another heuristic. The inspiration was the traditional Phong lightning model – putting in some artificial light. The goal was to brighten mostly the dense regions so the obvious choice was to introduce emission in form $L_e(x) = \epsilon\rho(x)$, where ϵ is a constant.

3.4 Step size

The goal of this section is establishing the Δx in Equation (3.3). The idea of adjusting the step size was propesed in several publications [2, 8], but none provided the following computations. The tutorial on Production volume rendering [21] contains a mention of frustum shaped buffer, but the dimensions of respective voxels are not discussed.

Optimal step size

The first requirement is to allow the camera to be positioned inside of the clouds, therefore the raymarch has to start at the near clipping plane of the camera. In order to achieve uniform resolution of the final image, cubes that will occupy exactly one pixel at the screen have to be sampled. Let $h \cdot n$ be the side of a square of a single pixel un-projected on the near clipping plane, where n in the distance of the clipping plane. Casting lines through the corners of such square will cut a small frustum of the rendered scene. (See Figure 3.4.)

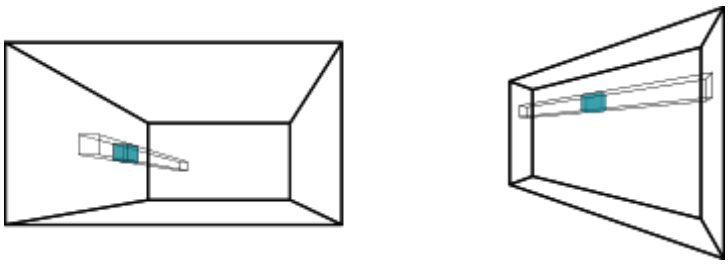

Figure 3.4: Frustum shaped voxels, the blue cube represent one voxel adapted to the frustum size.

A side of a square obtained by slicing this small frustum by a plane parallel to the clipping plane in the distance z will be $a(z) = h \cdot z$. Cutting the frustum into cubes means that the distances between the cutting planes would be equal to the sides of the squares. This results in making the steps of size $h \cdot z$.

$$\begin{aligned} z_0 &= n, \\ z_{k+1} &= z_k + h \cdot z_k, \\ z_k &= (1+h)^k \, z_0 \end{aligned}$$

The dimension h can be estimated from the height resolution (H) and the field of view angle (FOVY = α) as

$$h = \frac{2\tan(\frac{\alpha}{2})}{H}.$$

the number of steps to cover the whole space is

$$f = (1+h)^k\, n,$$
$$\frac{f}{n} = (1+h)^k,$$
$$k = \frac{\log(\frac{f}{n})}{\log(1+h)}.$$

Putting in the numbers, $n = 1$, $f = 1000$, $\alpha = 60°$, $H = 900$ px, the required number of steps is approximately 5388. I expect such performance to be achievable in the future, unfortunately as of today (May 27, 2016), the possible step counts would range from 64 to 256 samples depending on the particular machine.

The approach to keep the exponentially distributed steps[3] and tweaking the parameter h so that smaller number of steps cover the whole frustum results in

$$h' = \left(\frac{f}{n}\right)^{\frac{1}{k}} - 1. \tag{3.16}$$

Trying in the $n = 1$, $f = 1000$, $k = 255$ gives $h' = 0,027459$. The size of the last step would be $h'f/(1+h') = 26,73$, compared to the first step $h'n = 0,027$. This means 1000 times more details near the camera as opposed to the far clipping plane.[4] Although this is analytically correct, there is no way such step would work.

The reason why placing the clouds far and away in a bounding volume is such a great help, is that the theoretical step size ratio between the place in the middle of the scene (half of the distance to the far clipping plane) and the far clipping plane is only 2. The theoretical number of steps between the position $n = 500$ and $f = 1000$ with the same display resolution as above is only 540.

---

3. Solving the equation as $a(z) = \frac{\mathrm{d}z}{\mathrm{d}t} = h\, z$ yields $f(t) = Ce^{ht}$
4. Since the step size is directly proportionate to the distance, the ratio is inherent to the formula.

The practical solution

The approach I have opted for is very similar to the optimal solution, but utilizes more interactivity; picking an initial step size Δz_0 for the beginning and multiplying it by a small factor q every step. This describes a geometric series

$$z_k = n + \Delta z_0 + q\Delta z_0 + q^2\Delta z_0 + \cdots + q^{k-1}\Delta z_0,$$

and the sum of the series is

$$z_k = \frac{q^k - 1}{q - 1}\Delta z_0 + n. \tag{3.17}$$

Both Δz_0 and q are user modifiable parameters. This is very easily combined with the level of detail.

Level of detail

Since the Nyquist–Shannon sampling theorem says that the sampling frequency has to be at least twice the highest frequency in the sampled data, with the enlarging step the frequencies in the data have to fade. This can be achieved by couple of ways. Firstly, for procedurally generated clouds using fractal Brownian motion, the number of octaves would be adjusted accordingly to the sampling frequency. Secondly, for static data stored in a texture enabling the Mipmapping and sampling the texture's appropriate Mip level containing the less detailed data.

4 Occlusion

The ideas in this chapter are based on experiments with the algorithm and simple mathematical calculus. The possibility to adapt the step size to render just the portion of volume before the obstacle was first voiced by Bc. Dan Doležel.

To successfully incorporate the clouds into the scene, occlusion with other geometry has to be solved. The trivial approach is to compute the eye space z coordinate, compare it against the depth buffer and terminate the raymarch once it reaches too far from the camera. This however makes visible cutting panes, that are moving against the underlying surface, making them very hard to miss. (See Figure 4.1.) The solution is to integrate the whole volume as close to the surface of the occluding geometry as possible. This means adapting the step size accordingly to the values in depth buffer, so the ray ends just before hitting the surface.

$$\Delta\vec{x}' = \frac{d - n - b}{N|\Delta\vec{x}_z|}\Delta\vec{x}, \tag{4.1}$$

where $|\Delta\vec{x}_z|$ is the size of z component of the step in camera space, d is the distance in camera space reconstructed from depth buffer, n is the distance of near cutting plane (or the distance of dummy projection geometry on which is the volume rendered), N is the number of steps, b is bias, small number to end the raymarch slightly before hitting the geometry rather than being deprived of the final step due to numerical error. For sampling due to Equation (3.17) I adjust the Δz_0 parameter

$$\Delta z_0' = (d - n - b)\frac{q - 1}{q^N - 1}. \tag{4.2}$$

Including this into the implementation seemed flawless, until the camera appeared inside of a cloud, in a place with high enough density to be considered opaque. Even though the Algorithm 1 accounts for the changes in the step size, the difference between integrated colours was big enough to create image negative of the scene, where the shapes of occlusion geometry could clearly be seen. (See Figure 4.2.) An attempt to counter these visual artefacts was to enhance the precision of Riemann sum from Equation (3.3) by choosing a better numerical integration method.

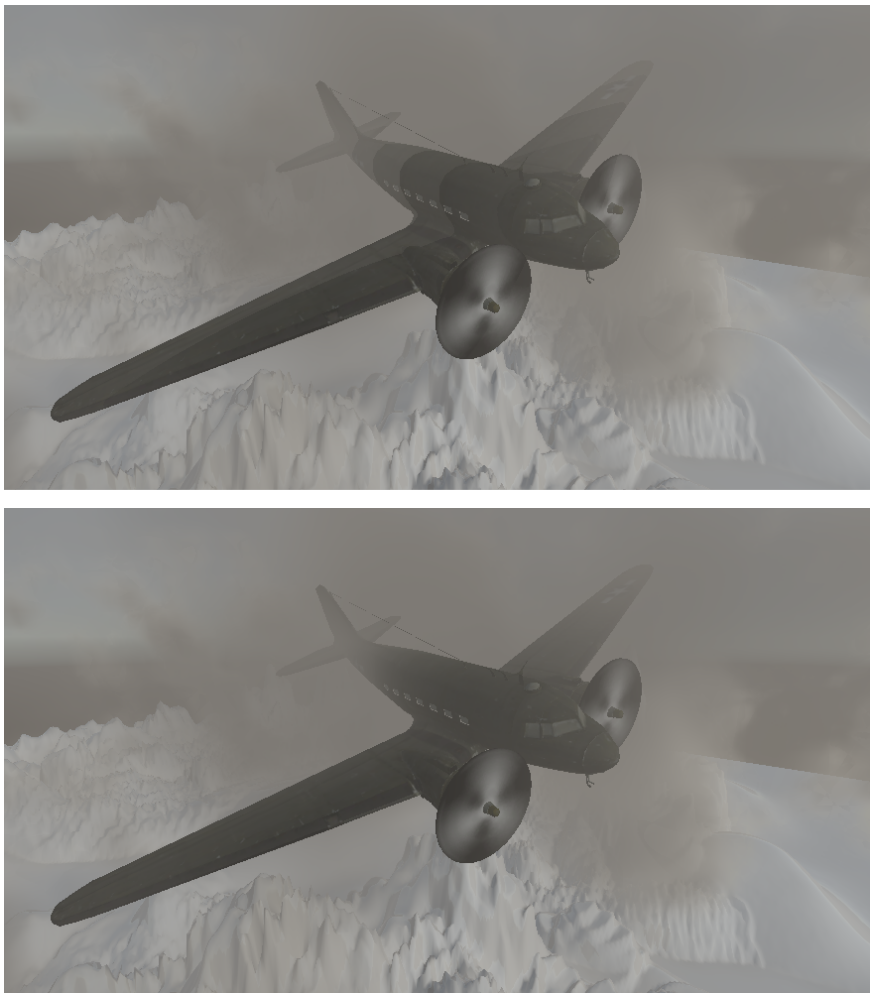

Figure 4.1: Geometry inside the volume of clouds computed using fixed step size (above) as opposed to adaptive step size (below). The plane model was downloaded from Unity Asset store.

4.1 Numerical integration

The numerical integration theory was taken from Chapter 10 from the book 'Numerical Methods' [22].

The first improvement from Riemann sum is the trapezoid rule. The general formula for evaluating definite integral using the trapezoid rule is

$$\int_a^b f(x)\,\mathrm{d}x \approx \sum_{i=1}^{n}(x_i - x_{i-1})\frac{f(x_i) + f(x_{i-1})}{2} \tag{4.3}$$

Using the equidistant spacing this reduces to

$$\int_a^b f(x)\,\mathrm{d}x \approx \left[\frac{f_0}{2} + f_1 + f_2 + \cdots + f_{n-1} + \frac{f_n}{2}\right] \tag{4.4}$$

Evaluating the integral by summing the halves using Equation (4.3) to get simple sum as shown in Equation (4.4), seems like twice as much work for no significant improvement. Another improvement is the composite Simpson's rule, reading

$$\int_a^b f(x)\,\mathrm{d}x \approx \sum_{i=1}^{n}\frac{x_i - x_{i-1}}{6}\left[f(x_{i-1}) + 4f\left(\frac{x_i + x_{i-1}}{2}\right) + f(x_i)\right], \tag{4.5}$$

which reduces to

$$\int_a^b f(x)\,\mathrm{d}x \approx \frac{h}{3}\left[\frac{f_0}{2} + 2f_1 + f_2 + \cdots + f_{2n-2} + 2f_{2n-1} + \frac{f_{2n}}{2}\right]. \tag{4.6}$$

and a $\frac{3}{8}$ Simpson's rule

$$\begin{aligned}\int_a^b f(x)\,\mathrm{d}x \approx \frac{3h}{8}\big[&f_0 + 3f_1 + 3f_2 + 2f_3 + 3f_4 + 3f_5 + \ldots \\ &\cdots + 2f_{3n-3} + 3f_{3n-2} + 3f_{3n-1} + f_{3n}\big].\end{aligned} \tag{4.7}$$

Since in Section 3.4 is shown that the length of the step should enlarge as it get further away from the camera, it is necessary to check how this affects the integration method. Using the Equation (4.5) and computing the integral piecewise would be a solution, but in order to reduce the number of instructions mapping the result directly to the Equation (4.6) is preferred.

Theorem 1. *The Simpson's rule can be applied by summing the sampled function in a for loop, using appropriate coefficients multiplied by step size.*

Proof. Suppose that desired sampling of the function $f(x)$ can be parametrized by injective continuous function $x = \varphi(t)$, such that t can be divided into uniform grid yielding the sampling points in x. This can be solved by performing a substitution and then using the Simpson's rule.

$$\begin{aligned}\int_a^b f(x)\,\mathrm{d}x = \left|\begin{matrix} x = \varphi(t) \\ \mathrm{d}x = \varphi'(t)\,\mathrm{d}t \end{matrix}\right| = \int_{\varphi^{-1}(a)}^{\varphi^{-1}(b)} f(\varphi(t))\varphi'(t)\,\mathrm{d}t \approx \\ \approx \frac{3\Delta t}{8}\left[f(\varphi(t_0))\varphi'(t_0) + 3f(\varphi(t_1))\varphi'(t_1) + \right. \\ \left. + 3f(\varphi(t_2))\varphi'(t_2) + 2f(\varphi(t_3))\varphi'(t_3) + \dots\right] = \\ = \frac{3}{8}\left[f(x_0)\Delta x_0 + 3f(x_1)\Delta x_1 + 3f(x_2)\Delta x_2 + 2f(x_3)\Delta x_3 \dots\right],\end{aligned} \tag{4.8}$$

using the fact, that $\varphi(t_i) = x_i$, and that

$$\mathrm{d}x = \varphi'(t)\,\mathrm{d}t \Rightarrow \Delta x(t_i) = \varphi'(t_i)\Delta t. \qquad \square$$

Using this to approximate the integral from Equation (3.2), first writing the equation

$$\begin{aligned}&L(a,\vec{\omega}) = L_b\, e^{-\tau(b,a)} + \\ &\int_b^a e^{-\tau(x,a)}\sigma\rho(x,\vec{\omega})p(\vec{\omega}\cdot\vec{\omega}_{sun})e^{-\int_{\varsigma(s)}\sigma\rho(s)\,\mathrm{d}s}\,\mathrm{d}x,\end{aligned}$$

where $\tau(t_1,t_2) = \int_{t_1}^{t_2}\sigma\rho(t)\,\mathrm{d}t$.

Now observe that to solve the outer integral from b to a the inner integral has to be evaluated for every sampling point of the outer one. Therefore it is useful to approximate the inner integral by the trapezoid rule and the outer integral by the Simpson's rule. The resulting color $\mathcal{C}_N$ will be expressed by function $f(x)$ to be substituted into the Simpson's law. The function $f(x)$ reads

$$f(x_n) = \sigma p\rho(x_n)\mathcal{I}_n \exp\left[-\sum_{i=1}^{n}\sigma'\frac{\Delta x_n}{2}(\rho(x_{i-1}) + \rho(x_i))\right]. \tag{4.9}$$

Introducing once again the $\mathcal{E}$ as

$$\begin{aligned} \mathcal{E}_0 &= 1, \\ \mathcal{E}_n &= \mathcal{E}_{n-1} \exp\left[-\sigma' \frac{\Delta x_n}{2}(\rho(x_{n-1}) + \rho(x_n))\right], \end{aligned} \tag{4.10}$$

and $\mathcal{C}_n$

$$\begin{aligned} \mathcal{C}_0 &= 0, \\ \mathcal{C}_n &= \mathcal{C}_{n-1} + \varsigma_n \sigma p \rho(x_n) \Delta x_n \mathcal{I}_n \mathcal{E}_n, \end{aligned} \tag{4.11}$$

where ς_n will assign correct constant according to Simpson's rule.

The observation is that the factor $\mathcal{E}_n$ is likely going to approach zero somewhere inside of the cloud, hence putting the correct factor ς_n on the last sampling position is not necessary. However it is crucial starting the integration no sooner than the raymarch will hit the cloud, otherwise even bigger integration errors than by using the Riemann sum would be introduced. The comparison can be seen in Figure 4.2.

4.2 Analytical solution

An analytical solution to the volume light integral can be found for special conditions. I have not found this solution anywhere in the literature, so I suppose that the other authors did not see any particular use for it.

Setting the phase function and incident lightning to 1, and solving the integral by substitution yields

$$\int_0^L \sigma\rho(x) e^{-\int_0^x \sigma\rho(t)\,\mathrm{d}t}\,\mathrm{d}x = \begin{vmatrix} u = \int_0^x \sigma\rho(t)\,\mathrm{d}t \\ \mathrm{d}u = \sigma\rho(x)\,\mathrm{d}x \end{vmatrix} = 1 - e^{-\int_0^L \sigma\rho(x)\,\mathrm{d}x}, \tag{4.12}$$

where the limits of the integration are from 0 (the near clipping plane) to L, (the far clipping plane or the distance of the occluding geometry).[1] This means that the colour of clouds with very dull phase function and homogeneous ambient lightning is equal to $1 - e^{-\tau(x)}$, where

1. The integration here is in different order than in the rest of the equations. The name front to back, and back to front is sometimes used. This one is front to back as opposed to the others in this thesis.

$\tau(x) = \int_0^x \sigma\rho(t)\,\mathrm{d}t$ is the non-negative optical thickness. It is easy to see that the result of the integral is bounded between 0 and 1. Observing that the function $\mathcal{I}$ discussed in Chapter 2, is for the single scattering of the form $\mathcal{I} = e^{-x}$, which is also bounded and that the factor p being of the form $p = p(\vec{\omega}, \vec{\omega}_{sun})$ can be considered constant for given ray. The color $\mathcal{C}$ will then be $\mathcal{C}(\vec{\omega}) = p(\vec{\omega})\zeta(\vec{\omega})$, where ζ is the result of the integration

$$\zeta(\vec{\omega}) = \int_0^L \mathcal{I}(x)\sigma\rho(x)e^{-\int_0^x \sigma\rho(t)\,\mathrm{d}t}\,\mathrm{d}x,$$

which on the condition that the density and scattering are bounded between 0 and 1, means that the whole integral is bounded between 0 and 1.[2] Enforcing the bounding criteria in the shading model gives an easy, but powerful criteria for checking the error. Should it happen during the numerical computation that the sum will be greater than 1, it is the sign of numerical error (probably caused by too long step) and the result should be clamped. Although this technique is virtually ubiquitous, especially in computer graphics, I have derived the explanation and formulated conditions for successful use.

2. For the proof of this think about the fact, that the integrand in the Equation (4.12) is non-negative on the domain of integration.

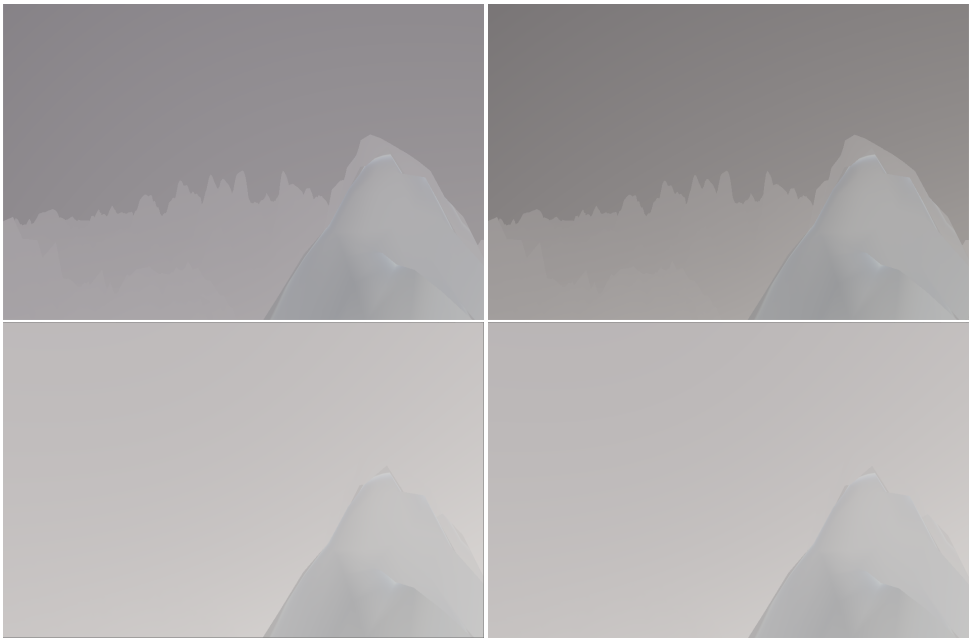

Figure 4.2: Comparison of numerical integration methods for the same parameters, top left Riemann integration, top right trapezoid rule, bottom left simpson 3/8 rule, bottom right simpson rule.

5 Implementation

The implementation was done in the Unity game engine as a custom material (shader), which is supposed to be applied to a plane geometry located before the camera.[1] All the rendering parameters are customizable during the game mode (except for the number of steps). Some of the rendering methods have to be changed by macros in the code. The shader is tagged as a transparency shader, in order to be rendered after the rest of the geometry in the scene. Combining the clouds with other transparent geometry was not tested. Using the clouds in an another project is as easy as putting the dummy geometry into the scene and attaching correct material and supporting scripts. Brief overview of the implementation details and the measurement of the performance is examined in the following sections.

5.1 Code

The shader programs in Unity are written in declarative language called ShaderLab[2], which encompasses all the 'bureaucracy' around the shader use. The particular vertex shader, fragment shader, ..., are located in this file as functions and are written in HLSL/Cg shading language. The compatibility with OpenGL should be provided by Unity. The extent of the compatibility was not examined in this work. The minimal working example of a vertex and fragment shader programs written in Unity ShaderLab is in Listing A.1.

I have to mention that the implementation is not thoroughly optimized and is meant as a proof of concept, not a standalone application. The goal was to implement a demonstration application that would allow for researching the method, not to implement a final solution. The Unity Game Engine was chosen for fast prototyping, not as a target platform.

---

1. It can be applied to any geometry working as a window into the skies.
2. `http://docs.unity3d.com/Manual/SL-Shader.html`

CPU computation

On the cpu side there are several routines located in five C# scripts, located in folder `Assets/Scripts`.

CameraScipt. Controls the camera movement. To control the camera the left mouse button has to be pressed. The view can be rotated by moving the mouse and the position can be moved by "W", "S", "A", "D" characters. To enable the controls the CameraScript has to be attached to MainCamera object.

DepthBufferSwitch. Enables the use of the depth buffer for custom shaders. Has to be attached to MainCamera in order to compute occlusion in scene.

ShaderValuesCpu. Rotates the cone of vectors positioned in 5, 10, 15 and 20 degrees angle from the axis into the direction of light in the world space coordinates. These are used for the fuzzy sampling of light. The script should be attached to the cloud's proxy geometry.

PerlinLoader. Takes one argument, the name of a 3D texture located in folder `textures`, to be loaded into the shader as a variable. The format of the texture is the one Pavelka used in his work [2]. Raw byte stream with three 32 bit integers defining dimensions followed by one float for every voxel. The order of the axes is irrelevant, since it is supposed to be a noise texture. The script should be attached to the cloud's proxy geometry.

WorleyLoader. Works the same as PerlinLoader, but stores the texture into different variable. Should also be attached to the cloud's proxy geometry.

Vertex shader

In vertex shader the position in all the coordinate systems is computed, namely the SV position, View space position, Screen space position, World space position and also the world space ray direction is computed as difference between the vertex position and camera position.

Fragment shader

All the work described throughout the thesis is done in the fragment shader.

1. Initialize the fuzzy path for the light by storing the respective step of the light path in an array of vectors. (Mathematical vectors of type `float3`.)

2. Initialize the variables – light direction, world space position, world space ray direction, eye space step size, eye space distance reconstructed from depth buffer.

3. If the Bounding Box parameter is set to YES, adjust the ray origin, to start the raymarch at the border of the bounding volume and adjust the step size to traverse the whole bounding volume.

4. If position of the ray origin is set to be locked to grid, move the position to closest grid point. ($\vec{x}_0 := \lfloor \vec{x}_0 \cdot f \rfloor / f$)

5. If the additive noise for position of the ray origin is enabled, move the ray origin in direction of ray direction by amount defined by a noise texture. ($\vec{x}_0 := \vec{x}_0 + \text{noise}(x_s) \cdot \vec{\omega}$)

6. If adaptive steps are set to YES (SmartStep switch), adjust the length of the ray due to occlusion as described in Chapter 4.

7. Do raymarching described at length in Chapter 3. The changing of respective integration methods is available by setting macros in the beginning of the shader code.

Shader parameters

The accesible properties of the shader that can be controlled from the Unity GUI, also during play mode. The ordering and names are the same as those on the GUI panel.

Noise. Adds noise to starting positions of the ray, the number specifies the magnitude. The noise is applied only if the density at current position is zero. The intention is to reduce stripes artefacts in the clouds.

Color. The color of the clouds. (The color of the light that will not be attenuated, for correct shading and no colouring the black color has to be set.)

Ambient color. The color of the ambient (emitted) light. For no emission the value of black colour should be set.

Frequency Perlin. Frequency of the Perlin data.

Frequency Worley. Frequency of the Worley data.

Frequency base Forces the ray origins to be aligned to a grid , the technique can be utilized to reduce 'boiling' of the high level frequencies of the cloud details, however, it is better to reduce the step size to suppress the boiling instead. Value less than one will create tiling alias. Value zero disables the effect.

Frequency. Frequency common to both Perlin and Worley data.

Blend. The ratio of mixing the Perlin vs. Worley data. 1 is Perlin, 0 is Worley.

Gain Perlin. Value multiplier for the Perlin data.

Gain Worley. Value multiplier for the Worley data.

Smart step. This enables the adaptive size of step due to collision of ray with a geometry in scene as described in Chapter 4.

Bias. The bias from the Chapter 4

Step size. Initial step size of the raymarch. This value will be modified due to occlusion (Smart step) and bounding volume options.

Step Q. The ratio to multiply the step size every step.

Brightness. Value multiplier for the final colour.

Density. Value multiplier for the density of the clouds.

Powder. The variable in powder function discussed in 3.3. The desired effect depends on the right size of the Shade step.

Shade. The multiplier from Section 3.3.

Shade Step. The size of the first step for the lightning subroutine from Chapter 3.

Shade Step Q. The quotient to multiply the length of the step size every step of the lightning subroutine. (This only affects one subroutine, the next one starts again with first step of size Shade Step.)

Shade step count. Number of steps to be taken in the lightning subroutine.

Shade jitter. The fuzzy path for light integration discussed in 3.3.

Greenstein. Greenstein phase function described in Section 2.2. The effect will be turned of for g = 0.

Rayleigh. Monochromatic Rayleigh phase function defined in Section 2.2. Set to zero to turn off.

Parameters A, B. Parameters for thresholding the noise texture, same as in Pavelka's work [2]. Value less than A from the noise texture is considered to be density 0, value more than B is considered to be density 1. In between these values the density is smoothly interpolated.

Alpha correction. A power of resulting alpha of the clouds. $\alpha := \alpha^k$

Alpha peeling. Will discard cloud pixels with alpha less than specified value.

Dimensions. Self explanatory values for dimensions of the cloud volume. The clouds are bounded between two spheres in the altitude given by the parameters.

Bounding box. A switch for enabling the clouds to be rendered with the information about the bounding box. Value greater than 0 will cause the bounding box to be used. The smooth entering of the volume is left as a future work.

Speed. The speed of the movement of the clouds. The clouds tend to disappear after a while for an unknown reason.

5.2 Measurement

The measurements were done in Unity 5.3.1 on a machine with the following specifications:

Windows 8.1 Pro, 64-bit Operating system
Intel(R) Core(TM) i7 CPU 920 @ 2.67 Ghz

GeForce GTX 780 Ti, DirectX11
Driver version 255.60
Direct3D API version 11.2
Total available graphics memory: 7167 MB
Memeory bandwidth: 336.00 GB/s

Memory

The memory requirements of the method are equal to those of storing the two noise textures. The maximal resolution tested was two textures at 256^3 voxels. The textures were in format RGBA32, taking 16 bytes for every voxel. $2 \times 16 \times 256^3 \doteq 540$ MB. This had no noticeable impact on the performance (See Figure B.1). It is posiible to cut down the memory requirement by storing different octaves of the noise in respective RGBA channels in a smaller texture and combining them during the computation as was done in the game HORIZON: Zero Dawn [1].

Time

Several measurements were done to determine the render time dependency of various parameters. For the measurements the Unity profiler was used, which introduced significant overhead, causing that the measured values do not correspond with the actual performance and can be viewed only relative to each other.

Integration method. The four different methods were measured for constant conditions of Perlin texture size 256^3, Worley texture size 128^3, 256 steps of raymarching, 4 steps of shading, display resolution of 1906x987. The results (See Figure B.2) show, that there is no relevant time dependency for the various methods.

Powder method variants. The measurement for variants of powder method shows, that there is no render time difference between the variants. (See Figure B.3.) The variants were measured at constant conditions of Perlin texture size 256^3, Worley texture size 128^3, 256 steps of raymarching, 4 steps of shading, display resolution of 1906x987, using Simpson's integration method.

Number of steps. The measurement confirms the theoretical linear render time dependency on the number of steps, while measuring at constant conditions of Perlin texture size 256^3, Worley texture size 128^3, 4 steps of shading, display resolution of 1906x987, using Simpson's integration method. (See Figure B.4.)

Resolution. The measurement shows the expected increase in render time on the increase in resolution, measuring at constant conditions of Perlin texture size 256^3, Worley texture size 128^3, 256 steps of raymarching, 4 steps of shading, using Simpson's integration method. (See Figure B.5.)

Coverage. The Figure B.6 shows strong dependency of render time on the amount of clouds in the scene, measured with constant conditions of Perlin texture size 256^3, Worley texture size 128^3, 256 steps of raymarching, 4 steps of shading, display resolution of 1906x987, using Simpson's integration method. (See Figure B.6.)

The most interesting of these measurements is that the render time strongly depends on the amount of clouds in the scene. This includes both the cloud coverage settings and the dependency on the viewing angle, determining the amount of skies being observed. The bottom line being that the not optimized implementation runs on good enough hardware in FullHD (1920x1080) resolution at real-time frame rates.

5.3 Visual results

The Figure 5.1 shows the view from distance, the Figure 5.2 shows an example of a flight through the clouds.

Figure 5.1: The rendered image of clouds from distance

Figure 5.2: The rendered image of flight through the clouds

Conclusion

The goal of this thesis was to describe the existing methods for real-time rendering of volumetric clouds and research the possibility of real-time rendering of dynamic clouds with dynamic camera inside the volume of clouds.

The problematic of light interaction with the clouds and other general translucent volumetric objects was described at length, the known solutions were listed and several approaches for rendering the phenomenon were explained. I have chosen and adapted an existing method to abide with the new requirements. As a great success I consider the adaptation of better numerical integration, which I have not seen implemented before in this field. Secondly the analysis of the effect of multiple forward scattering, although more thorough study should follow. Last but not least the theoretical work on step size and the occlusion, connected with level of detail. I have explained the limitations of the method and implemented a demonstration application that can be used for further experiments and improvements.

Among the improvements that should be forthcoming is definitely implementing realistic shadows, especially shadows cast by the dynamic objects inside the volume of clouds onto the clouds volume. Other easier improvements may be atmospheric effects such as god rays and distance blurring. The application is prepared to visualize any kind of data, but a dedicated weather system with dynamic modelling of evolving clouds would increase realism significantly.

Bibliography

[1] Andrew Schneider. *The Real-time Volumetric Cloudscapes of Horizon: Zero Dawn*. 2015. URL: http://bit.ly/1XRf4OV (visited on 05/11/2016).

[2] Michal Pavelka. 'Realistické vykreslování mraků v reálném čase'. Master's Thesis. Masarykova univerzita, Fakulta informatiky, Brno, 2016. URL: http://is.muni.cz/th/373851/fi_m/ (visited on 05/08/2016).

[3] A. Kokhanovsky. 'Optical properties of terrestrial clouds'. en. In: *Earth-Science Reviews* 64.3-4 (Feb. 2004), pp. 189–241. ISSN: 00128252. URL: http://www.patarnott.com/satsens/pdf/opticalPropertiesCloudsReview.pdf (visited on 05/18/2016).

[4] Daniela Řezáčková et al. *Fyzika oblaků a srážek*. Praha: Academia, 2007. ISBN: 978-80-200-1505-1.

[5] Roger Clausse and Léopold Facy. *The Clouds*. New York: Grove Press, 1961.

[6] Geoffrey Y. Gardner. 'Visual simulation of clouds'. In: *ACM Siggraph Computer Graphics*. Vol. 19. ACM, 1985, pp. 297–304. URL: http://dl.acm.org/citation.cfm?id=325248 (visited on 05/18/2016).

[7] Joe Kniss et al. 'Interactive translucent volume rendering and procedural modeling'. In: *Visualization, 2002. VIS 2002. IEEE*. IEEE. 2002, pp. 109–116.

[8] Klaus Engel et al. *Real-Time Volume Graphics*. Wellesley, MA: A K Peters, Ltd., 2006. ISBN: 978-1-56881-266-3.

[9] Mark J. Harris and Anselmo Lastra. 'Real-time cloud rendering'. In: *Computer Graphics Forum*. Vol. 20. 2001, pp. 76–85. URL: http://www.markmark.net/PDFs/RTClouds_HarrisEG2001.pdf (visited on 05/17/2016).

[10] E Yusov. 'High-Performance Rendering of Realistic Cumulus Clouds Using Pre-computed Lighting'. In: *High-Performance Graphics 2014, Lyon, France*. IEEE. 2014, pp. 127–136.

[11] Jakub Křoupal. *Generování a vykreslování mraků*. Bachelor's thesis. 2008. URL: http://is.muni.cz/th/173165/fi_b/ (visited on 05/17/2016).

[12] Vít Kučera. 'Zobrazování animovaných oblaků v reálném čase'. Master's Thesis. Vysoké učení technické v Brně, Fakulta informačních technologií, 2007. URL: http://hdl.handle.net/11012/54024 (visited on 05/17/2016).

[13] Antoine Bouthors et al. 'Interactive multiple anisotropic scattering in clouds'. In: *Proceedings of the 2008 symposium on Interactive 3D graphics and games*. ACM, 2008, pp. 173–182. URL: http://www-evasion.imag.fr/Publications/2008/BNMBC08/clouds.pdf (visited on 05/17/2016).

[14] Ken Perlin and E. M. Hoffert. 'Hypertexture'. In: *SIGGRAPH Comput. Graph.* 23.3 (July 1989), pp. 253–262. ISSN: 0097-8930. URL: http://doi.acm.org/10.1145/74334.74359 (visited on 05/17/2016).

[15] Yoshinori Dobashi et al. 'A simple, efficient method for realistic animation of clouds'. In: *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*. ACM Press/Addison-Wesley Publishing Co., 2000, pp. 19–28. URL: http://dl.acm.org/citation.cfm?id=344795 (visited on 05/19/2016).

[16] James T. Kajiya and Brian P. Von Herzen. 'Ray tracing volume densities'. In: *ACM SIGGRAPH Computer Graphics*. Vol. 18. ACM, 1984, pp. 165–174. URL: http://dl.acm.org/citation.cfm?id=808594 (visited on 05/18/2016).

[17] David S Ebert. *Texturing and modeling: a procedural approach*. Third edtition. AP Professional, 2007. ISBN: 0-12-228760-6.

[18] Robert Siegel and John R. Howell. *Thermal Radiation Heat Transfer*. Third edtition. Hemisphere Publishing Corporation, 1992. ISBN: 0-89116-271-2.

[19] Wolfgang F Engel. *ShaderX5: advanced rendering techniques*. Third edtition. Boston MA: Charles River Media, 2007. ISBN: 0122287606.

[20] Louis G Henyey and Jesse Leonard Greenstein. 'Diffuse radiation in the galaxy'. In: *The Astrophysical Journal* 93 (1941), pp. 70–83.

[21] Magnus Wrenninge and Nafees Bin Zafar. *Production Volume Rendering. Fundamentals*. Oct. 2011. URL: http://bit.ly/1XRer7W (visited on 05/11/2016).

[22] Anne Greenbaum and Timothy P. Chartier. *Numerical methods: design, analysis, and computer implenentation of algorithms*. Princeton University Press, 2012. ISBN: 978-0-691-15122-9.

A Listings

```
Shader "MyShader" {
  Properties  {
    _MainTex ("Texture", 2D) = "white" {}
  }
  SubShader {
    Pass {
      CGPROGRAM
      #pragma vertex vert
      #pragma fragment frag

      #include "UnityCG.cginc"

      struct appdata {
        float4 vertex : POSITION;
        float2 uv : TEXCOORD0;
      };

      struct v2f {
        float2 uv : TEXCOORD0;
        float4 vertex : SV_POSITION;
      };

      sampler2D _MainTex;

      v2f vert (appdata v) {
        v2f o;
        o.vertex = mul(UNITY_MATRIX_MVP, v.vertex);
        o.uv = v.uv;
        return o;
      }

      float4 frag (v2f i) : COLOR {
        return tex2D(_MainTex, i.uv);
      }
      ENDCG
    }
  }
}
```

Listing A.1: A minimal working example of shader written in Unity ShaderLab. Applies a specified texture to an object it is attached to.

B Measurement charts

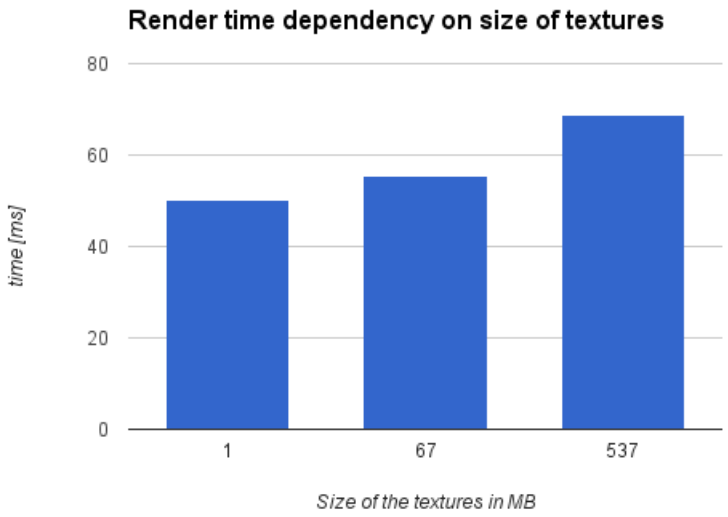

Figure B.1: Render time dependency on the size of the texture data.

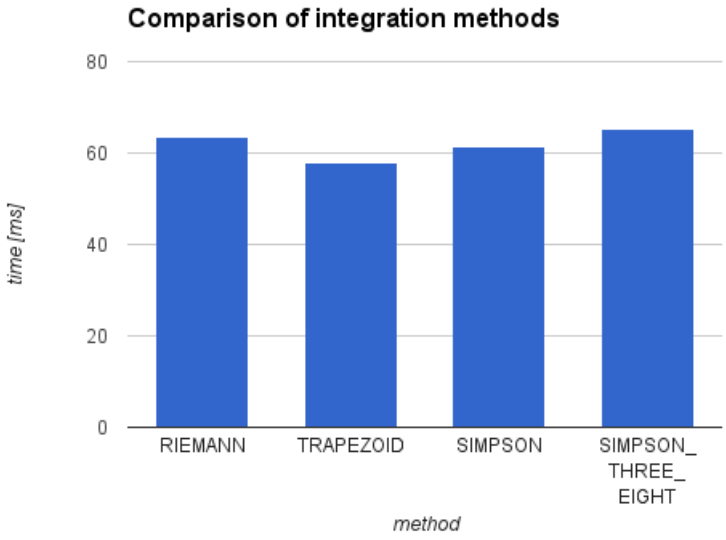

Figure B.2: Render time comparison of numerical integration methods.

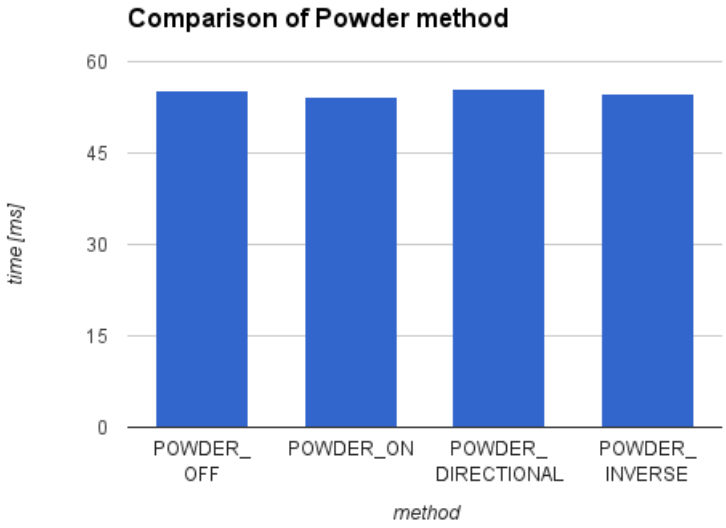

Figure B.3: Render time comparison of 'Powder' method variants.

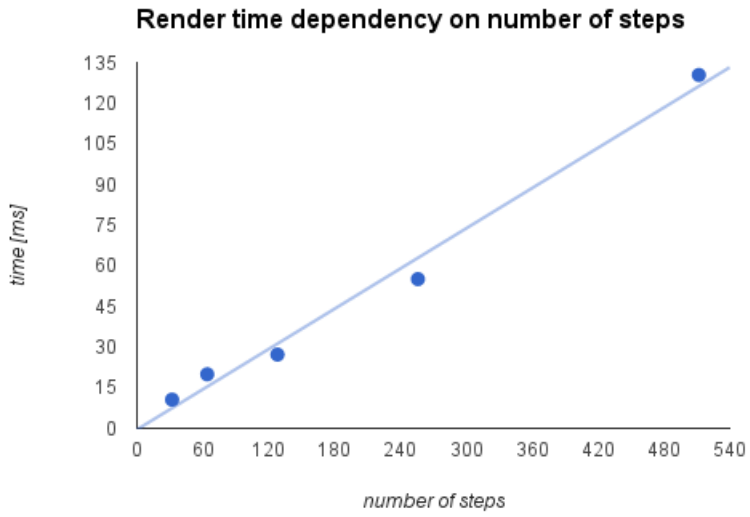

Figure B.4: The render time dependency on number of steps of the raymarch.

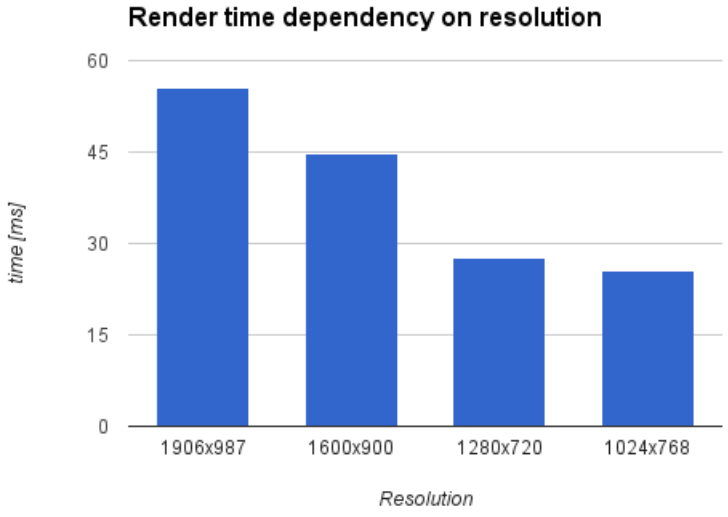

Figure B.5: Render time dependency on the target resolution.

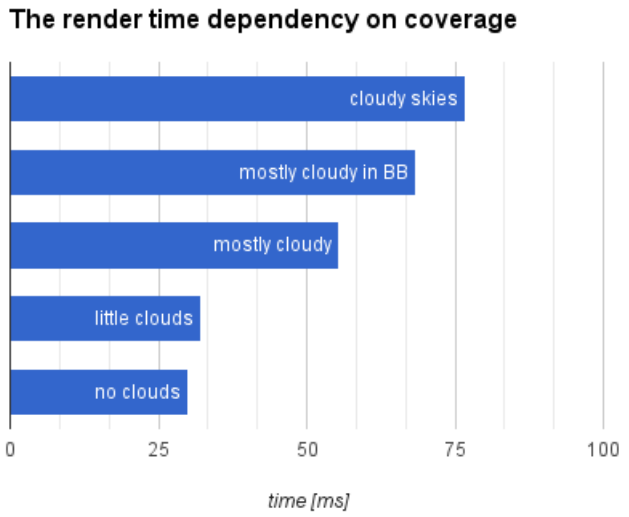

Figure B.6: Render time dependency on the amount of clouds in scene.

C Working with Unity

In order to run the code associated with this thesis, Unity game engine is required[1]. The file archive in information system of the Masaryk University[2] contains a Unity project, which can be imported into Unity by extracting the archive → opening Unity → `Open`, and navigating to the main folder of the extracted project (`VolumetricClouds`). After opening the project in Unity a demonstration scene is located in project structure in folder Scenes. (If project structure is not visible in the layout, `Window` → `Project`, or pressing Ctrl+5, will open the project structure.) By pressing play arrow in the middle of the toolbar above the realtime rendering of cloud shold be happening in game window. To adjust the parameters of the shader, the game object `Clouds` has to be selected in scene hierarchy. The parameters will then appear in the Inspector tab. (To open any of the mentioned, if needed, the dropdown menu `Window` will help.) The Figure C.1 show the Unity at work. The archive contains also a demonstration video.

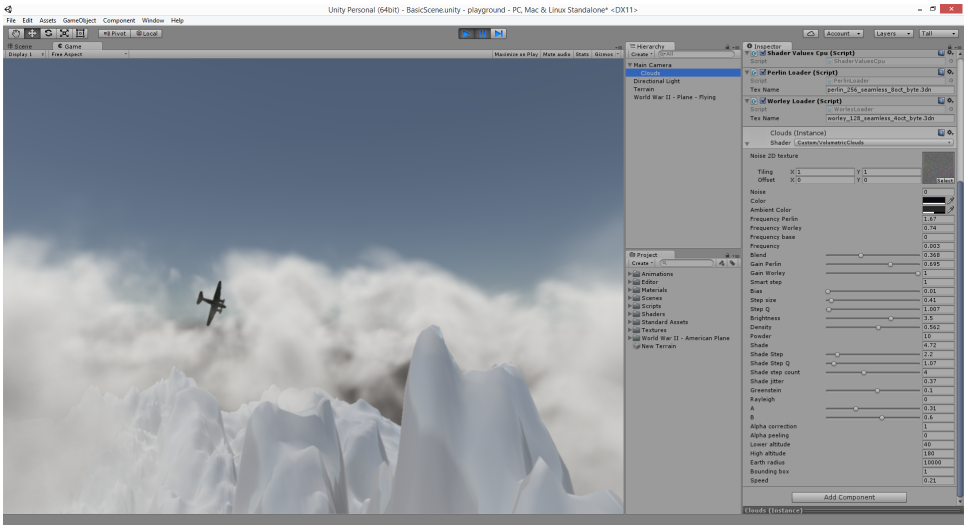

Figure C.1: Preview of the implementation in use in Unity game engine. The controls for the parameters are visible on the right.

1. Unity game engine is available online at `https://unity3d.com/get-unity`
2. The archive is available online at `https://is.muni.cz/th/396277/fi_m/`