Solving Radiance Transport as an Ordinary Differential Equation

by

Chris Gonterman

A thesis submitted in conformity with the requirements
for the degree of Master of Science
Graduate Department of Computer Science
University of Toronto

Abstract

Solving Radiance Transport as an Ordinary Differential Equation

Chris Gonterman

Master of Science

Graduate Department of Computer Science

University of Toronto

2010

We introduce an alternative to Monte-Carlo techniques for solving radiance transport problems for participating media. We use a reformulation of the volume rendering equation from its standard integro-differential form to a purely differential form. We then leverage the large body of work in numerical methods for solving differential equations by framing and analyzing the problem as a differential equation. To our knowledge, this is the first application of such techniques in the area of photorealistic rendering of volumes based on ray optics.

Contents

Chapter 1

Introduction

Image synthesis poses a rich problem space in computer graphics. Accurately portraying a scene incorporates the techniques and knowledge of many fields, including but not limited to software engineering, physics, mathematics, and computer science. In particular, visualizing complex effects such as global lighting and volumetric phenomena requires a significant amount of computation. This thesis addresses the problem of rendering images with participating media and introduces a new technique for rendering these effects.

As producing high quality images is very time consuming, a critical goal in rendering research is fine-tuning the trade off between computational resources and visual fidelity. Approaches to this fall into two key categories: real-time and offline. Real-time algorithms are characterized by maximizing image fidelity under stringent fixed time requirements, whereas offline algorithms place a higher emphasis on the visual accuracy of the final image. The entertainment industry presents many high profile applications of these approaches: video games are a prime example of real-time techniques, while movie special effects exemplify offline techniques.

An elusive, challenging and important phenomenon that requires visual realism is that of participating media such as smoke, steam and dust. These effects are described by the physical process of radiative transfer. The Radiative Transfer Equation (RTE, see

Section 2.1) which governs this process yields closed form solutions in only the simplest of cases, so accurately rendering its effects often requires lengthy numerical calculation. This complexity has forced all but the most advanced real-time techniques to resort to approximating the effects of smoke with particle systems and simple billboards (textured rectangles). On the other hand, offline methods are able to directly compute a solution to the RTE using numerical techniques. A common approach to solving the RTE is Monte Carlo integration, as with other rendering problems; however the necessity of Monte Carlo methods is not clear.

Fortunately, due to the nature of the RTE, Monte Carlo integration is not the only available method. This thesis explores the application of non-Monte Carlo techniques to solving the RTE in a ray tracing setting. In particular, we lever the body of work in numerical methods for solving Ordinary Differential Equations (ODEs). Numerical methods for differential equations (NMDE) have been studied for over a century. We can incorporate this knowledge to gain a better understanding of the efficacy of various techniques for solving the RTE. We expect this will add a degree of control over the accuracy of the solution. Unless analytic or empirical data is available, controlling the accuracy of the RTE solution, and thus the final image quality, is something that can only be done indirectly with Monte Carlo integration by varying the number of samples used. Tuning the sample count for a desired image quality is a matter of trial and error.

Chapter 2 introduces the RTE in the context of image synthesis, and discusses previous techniques for rendering images with it. Chapter 3 describes the basics of ODE formulation and solution, and presents a variety of solvers that our algorithm will utilize. Chapter 4 discusses the details of applying ODE solvers to the RTE in a ray tracer, as well as the specifics of our implementation. Chapter 5 presents results from experiments with our renderer and existing techniques. In chapter 6, we conclude that the technique presented by this thesis provides a viable alternative to Monte Carlo methods, and we discuss possible extensions for future work.

Chapter 2

Volume Rendering

Ray tracing produces images by simulating the transport of light energy within a scene using the geometric concept of rays interacting with objects in the scene. This algorithm has been the basis of most photorealistic image synthesis algorithms since it was proposed by Turner Whitted in 1980[20]. The physical basis for this approach draws on the laws of geometric optics to determine the paths of the rays, as well as the law of conservation of energy to describe the interaction of the rays with the objects they intersect, producing the color of the light seen along each ray. The laws of radiative transfer predict the transfer of light along those paths between surface interactions. It is this last circumstance in which participating media comes into play.

Light transport yields two key equations that characterize how light is scattered among interactions within the scene. The first deals with boundary conditions at surface interactions, and is known as the Rendering Equation. The second models light scattering and transport between surface interactions; it is called the Radiative Transfer Equation. As the effects of participating media are confined to radiative transfer, it is this second equation in which we are interested. We will explore the Radiative Transfer Equation and how it models the effects of participating media in Section 2.1. Section 2.2 discusses key advances in volume rendering research related to participating media.

2.1 The Volume Rendering Equation

The fundamental tool for visualizing participating media is the Radiative Transfer Equation (RTE)[4]. Graphics researchers have come to know it as the Volume Rendering Equation. It describes the propagation of electromagnetic radiation in a medium, taking into account emission, absorption and scattering. Given the intensity of light at the boundary of the participating medium, the equation uniquely specifies the light field for the volume. The following integro-differential formulation describes the light radiance along a ray:

$$\nabla_\omega L(\mathbf{x},\omega) + \sigma_t(\mathbf{x})L(\mathbf{x},\omega) = E(\mathbf{x},\omega) + \sigma_s(\mathbf{x})\int_{S^2} p(\mathbf{x},\omega,\omega_i)L(\mathbf{x},\omega_i)d\omega_i \tag{2.1}$$

where $L(\mathbf{x},\omega)$ is the radiance of the light at the point $\mathbf{x}$ in the direction ω, σ_t and σ_s are the extinction and scattering coefficients, respectively, E is the medium's emissivity, S^2 is the set of directions on the unit sphere, and p is the medium's phase function. The phase function will be described in detail later.

Each of the terms admits a simple physical analog. The first term, $\nabla_\omega L(\mathbf{x},\omega)$, is a directional derivative along ω of the light field at the point $\mathbf{x}$. In other words, it gives us the change in the light field in the direction of the light ray, with respect to direction ω. The next term, $\sigma_t(\mathbf{x})L(\mathbf{x},\omega)$, tells us what proportion of the light is absorbed and thus removed by the medium. This accounts for absorption of light within the medium along a given direction, as well as when the light is scattered away from the direction of the ray, called *out-scattering*. The next term is $E(\mathbf{x},\omega)$, the amount of light spontaneously generated by the medium.

The last term is the most complicated part of the equation. It describes the amount of light scattered towards the direction of the ray, called *in-scattering*. Given a unit quantity of light coming in from the direction ω_i, the phase function, $p(\mathbf{x},\omega,\omega_i)$, tells us how much light is scattered in the direction ω. After accumulating the light scattered from all directions, the scattering coefficient, $\sigma_s(x)$, modulates this. The scattering coefficient

is also referred to as the *scattering albedo*. The in-scattering term combined with the emission term are together called the *source term*, as they describe the source of any additional light along the ray.

We now see that equation 2.1 can be intuitively described as the following:

$$\text{(change in light along } \omega) + \text{absorption} = \text{emission} + \text{in-scattering}$$

$$\text{(change in light along } \omega) = \text{emission} - \text{absorption} + \text{in-scattering}$$

Now that we have an equation describing how light rays interact with participating media, we need to know how to solve it. Due to the complexity of the in-scattering term, it is impossible to derive an analytical solution for all but the simplest of cases. Furthermore, the light field $L(\mathbf{x}, \omega)$ is a five dimensional function. Rendering algorithms only require its value at a few particular locations and directions. Thus, we can reduce the dimensionality of the problem by parameterizing equation 2.1 by a specific ray $\mathbf{x}(t) = \mathbf{x}_0 + t\omega$:

$$L'(t) = E(t) - \sigma_t(t)L(t) + \sigma_s(t)\int_{S^2} p(t, \omega_i)L(\mathbf{x}, \omega_i)d\omega_i \tag{2.2}$$

Typically, we will have traced a ray from a surface intersection because we need to know the light incident on the intersection in a specific direction. We use this ray to parameterize the equation, and then use whatever method desired to solve for the incident light.

While our approach uses Eq. 2.2 directly, existing techniques require a pure integral equation, as opposed to an integro-differential equation. This can be achieved by integrating Eq. 2.2:

$$L(t) = \int_0^t \left[T(0,t')\big(E(t')dt' + \sigma_s(t')S(t')\big)\right] dt' + T(0,t)L(0), \tag{2.3}$$

where the in-scattering term $S(t)$ is defined as:

$$S(t) = \int_{S^2} p(t, \omega_i)L(\mathbf{x}(t), \omega_i)d\omega_i,$$

and the transmittance $T(a, b)$, which describes how much a unit of light is attenuated when traveling from a to b, is

$$T(a, b) = e^{-\int_a^b \sigma_t(t')dt'}.$$

Eq. 2.3 is often referred to as the *volume rendering equation*.

This description of scattering phenomena is unavoidably recursive, as the source term includes the contribution of the function $L(\mathbf{x}, \omega_i)$ itself. This recursion is the primary source of complexity in generating images with the RTE. An obvious way to mitigate this is to limit the amount of recursion, effectively limiting the number of scattering events that take place. This is a reasonable approximation to use when the scattering albedo σ_s is low, as each scattering event reduces the total energy of the ray significantly. If we model the source term by considering only direct paths to light sources, ignoring the effects of the source term along those paths, we have a *single scattering* model. The name owes to the fact that the scattering events modeled by the source term are handled only a single time per light path. Single scattering is sufficient for variety of effects, such as sunbeams, also known as crepuscular rays. The results section contains images of such effects, e.g. Figure 5.3, rendered with single scattering. In a *multiple scattering* model, the full effects of the recursive nature of the source term are considered. Correctly considering multiple scattering has the effect of brightening the medium, as light is allowed to diffuse throughout via scattering.

2.1.1 Phase Functions

The phase function $p(\mathbf{x}, \omega, \omega_i)$ is analogous to the Bidirectional Reflectance Distribution Function (BRDF) used in surface lighting calculations. Both define the proportion of outgoing light at a scattering event as a function of incoming radiance. For all BRDFs $f(\omega_i, \omega_o)$, the following holds true:

$$\forall \omega_i \int_{S^2} f(\omega_i, \omega_o) d\omega_o \leq 1.$$

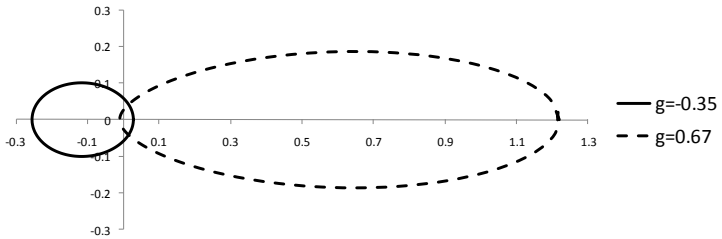

Figure 2.1: Plot of two Henyey-Greenstein functions with differing g. Functions with negative g (solid line) primarily backscatter light, whereas those with positive g show forward scattering.

Interpreted physically, this asserts that the BRDF conserves energy. When a surface absorbs some amount of light, this integral will be less than one. A similar statement holds for phase functions, except that since absorption is characterized by the σ_a term, the integral must evaluate to exactly one:

$$\forall\omega \int_{S^2} p(\mathbf{x}, \omega, \omega_i) d\omega_i = 1$$

Most phase functions depend only on a single angle θ between the incoming and outgoing and outgoing directions ω and ω_i. In this case, the participating medium is called *isotropic*, and the phase function is denoted by simply $p(\theta)$, where directly forward is defined as $\theta = 0$, while $\theta = \pi$ defines the backward direction. If there is additional dependence on the direction ω, the medium is *anisotropic*. It is useful to define a parameter g that describes the preferred scattering direction of the function:

$$g = \frac{1}{4\pi} \int_{S^2} p(\theta) \cos\theta d\theta,$$

where θ is the angle between the incoming and outgoing directions ω and ω_i. This formula is chosen to describe the average value of the product of the phase function and the cosine of the scattering angle ω[17]. Positive values of g correspond to forward scattering. If $g = 0$, there is no preferred direction, and, in a confusing overloading of the term, the

phase function is called *isotropic*. This is distinguished from the previous usage of the term to describe the dependence on the direction ω, where the medium itself is called isotropic. Most phase functions are, however, anisotropic. Figure 2.1 provides examples of phase functions with different values of g to show how the shape of the function relates to g. As $\theta = 0$ corresponds to the forward direction, the shape of the plot shows the distribution of scattered light as a ray travels to the right.

There are a few common phase functions that are used, some for physical accuracy and some for efficiency. The simplest phase function is isotropic:

$$p(\theta) = \frac{1}{4\pi}. \tag{2.4}$$

The most common parameterized phase function is the Henyey-Greenstein function[6]. It was introduced to empirically describe the scattering of light by the interstellar medium, and has since been used to describe scattering in additional media such as clouds and water.

$$p(\theta) = \frac{1}{4\pi} \frac{1 - g^2}{(1 + g^2 - 2g\cos\theta)^{1.5}}, \tag{2.5}$$

where $g \in \,]-1, 1[$ is a parameter exactly corresponding to the previously mentioned preferred scattering direction. When the efficiency of computing the fractional exponent is a concern, the following approximation introduced by Schlick may be used[1]:

$$p(\theta) = \frac{1}{4\pi} \frac{1 - k^2}{(1 + k\cos\theta)^2} \tag{2.6}$$

Again, $k \in \,]-1, 1[$ controls the preferred scattering direction. Figure 2.2 shows how close the Schlick function approximates the Henyey-Greenstein function for corresponding values of g and k. Both of these functions describe distributions that are easy to sample, which is useful for Monte-Carlo techniques as well as photon mapping, which are described later.

Several analytical phase functions have been derived for situations in which physical accuracy is a primary concern. For instance, Rayleigh scattering describes the scattering

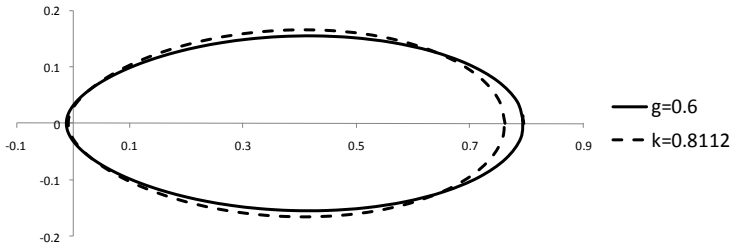

Figure 2.2: Comparison of Schlick and Henyey-Greenstein phase function for corresponding parameters

of light by spheres much smaller than the wavelength of the light[17]. The formula has a wavelength dependency that causes shorter wavelengths to scatter more. This type of scattering occurs in the Earth's atmosphere, which contributes to the blue color of the sky. When larger particles are involved, Mie theory[15] may be used to describe the scattering, which is unfortunately prohibitively complex for many applications. Often, weighted combinations of Henyey-Greenstein functions are used to approximate these real-world models.

2.2 Survey of Rendering Algorithms

Rendering algorithms for participating media all have their roots in radiative transport theory. The Radiative Transfer Equation described above was introduced by Chandrasekhar in 1960 with his book *Radiative Transfer*[4]. The theory has been used in a variety of fields, such as astrophysics and atmospheric sciences. As graphics is concerned primarily with the visual fidelity of generated images, the model may be simplified in favor of less computationally intensive approaches or those that present more intuitive control over the simulation. As such, most rendering techniques rely on phenomenological simplifications of the model presented by Chandrasekhar.

Early attempts to render the effects of participating media were limited to simple

directly observed phenomena, such as color desaturation when light travels through the atmosphere for an extended period. The first application of models based on radiative transfer to the synthesis of images with participating media was in a paper by Blinn in 1982[2]. This paper modeled participating media as a flat layer of suspended particles of uniform radius. Blinn then introduced a single scattering lighting model, simulating a beam of light entering the layer, attenuating due to absorption, and scattering at most once, directly into the viewing direction. The technique was used to render the rings of Saturn using empirical values for media properties derived from Voyager I measurements.

Blinn's derivation assumed a single light source with simple media geometry, and, importantly, assumed a low scattering albedo such that the effects of multiple scattering were negligible. Additional work was done to extend the model to more complex geometry, such as by Max[13], but restrictions on the lighting still existed. Kajiya and Von Herzen constructed two new models that did not restrict the placement and number of lights, allowing for integration with ray tracing techniques[11], one of which was suitable for multiple scattering. Both approaches represented media as a three dimensional grid of density data, allowing for more general volumes than Blinn's original technique. Their single scattering technique consisted of two steps. First, the light from each of n light sources is sent through the volume and stored in n three dimensional arrays. This is done by calculating a line integral from each point in the array to each light source, ignoring scattering events. In the second step, the image is rendered by evaluating the RTE as a line integral along the camera ray, using the light array in place of the in-scattering term.

Kajiya and Von Herzen's second technique is used when the scattering albedo is high, where multiple scattering effects are appreciable. Two approximations are used: the first is a perturbation expansion of the RTE on the scattering albedo, and the second is the usage of band limited spherical harmonics to represent the light field. These yield a series of equations defined on the three dimensional grid of the participating media which can be solved using relaxation methods. Methods using band-limited spherical harmonics in

this manner are termed P_N-methods, with N corresponding to the highest band used.

Kajiya and Von Herzen's technique did not model all volume/surface interactions. While participating media cast shadows on surfaces, the reverse was not true. Rushmeier and Torrance took a different approach to calculating light transport based on the radiosity global illumination algorithms[18]. Their technique is known as a *zonal method*, because the media is divided up into zones of uniform density. A particular zone contributes a well-defined proportion of light to any other zone or surface element, akin to radiosity's form factors. Computing these factors yields a set of simultaneous equations describing the intensity of the zones and surface in terms of the other elements. This particular formulation requires the assumption of an isotropic phase functions, but it has been extended to allow anisotropy by using the *Discrete Ordinates* method, discretizing the directions as well as the zones[14].

Each of these techniques relies on a memory-intensive discretization of the volume. Monte Carlo techniques, such as those based on Kajiya's path tracing approach[10], alleviate this memory issue, and they also allow for more general volume geometries. These approaches evaluate the rendering equations in their integral form using a Monte Carlo estimator, a technique for numerically solving any definite integral. Given an integral $\int_a^b g(x)dx$, the Monte Carlo estimator tells us to take N samples of $g(x)$ according to some distribution $p(x)$, and sum the samples according to the following formula to get an accurate approximation to the value of the integral:

$$G_N = \frac{1}{N}\sum_{i=1}^{N}\frac{g(x_i)}{p(x_i)}. \tag{2.7}$$

G_N is guaranteed to converge to the value of the integral as the number of samples N grows arbitrarily large. Due to the nondeterministic nature of the Monte Carlo estimator, it is important to use a sufficient number of samples, as well as tailored sample distributions, in order to increase accuracy and convergence in practice. A thorough explanation of the Monte Carlo estimator can be found in PBRT[17].

Since the Monte-Carlo estimator is suited for general purpose complex integral calcu-

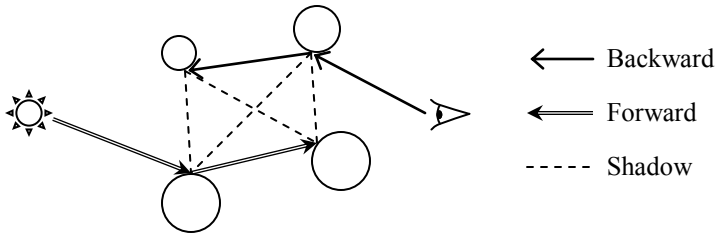

Figure 2.3: Bidirectional path tracing follows rays forward from the light as well as backwards from the camera. Shadow rays connect all the points in the different paths.

lations, it can incorporated directly into ray tracing computations. As mentioned above, this approach was utilized by Kajiya to solve the Rendering Equation for surfaces. It has also been used for participating media, for example in Lafortune's adaptation of bidirectional path tracing to participating media[12]. In particular, the bidirectional path tracing algorithm simultaneously traces paths from a light source (forward tracing) and the camera (backward tracing), connecting the paths' scattering events via shadow rays in order to compute shading. When handling participating media, each path is allowed to randomly scatter within the medium according to a probability computed via the optical thickness of the medium along the path. By sampling enough paths, the technique is guaranteed to converge to the correct solution, but it requires considerable amounts of time, a general trait of Monte Carlo algorithms.

Global illumination and caustics are difficult to capture both with and without participating media. Jensen introduced photon mapping to render these effects selectively and efficiently, first without participating media[8], and subsequently to include participating media[9]. This is a two-pass algorithm, separating forward tracing from backward tracing. The first pass traces a number of paths starting from light sources, depositing photons into a data structure when the path interacts with surfaces or media. This data structure, the photon map, provides a characterization of the light in the scene. In the second pass, rays are traced from the camera, and the photon map is used to provide

lighting information at surface and volume shading calculations. Volume interactions are calculated by evaluating the RTE using a Monte Carlo technique called *ray marching*, described in more detail in Chapter 4.

2.3 Summary

We introduced the Radiative Transfer Equation, Eq. 2.1, and detailed its effect on light transport. We explored key papers in the area of rendering participating media. Various techniques, including finite element techniques, emerged to solve approximations of the RTE efficiently. Monte Carlo based ray tracing approaches were developed to handle more general cases, and have been popular choices for rendering complex global illumination effects. The techniques introduced in this thesis can be inserted into ray tracing algorithms in the same place as these Monte Carlo solvers. As our technique solves the RTE as an ODE directly, the next chapter describes the tools necessary to do so.

Chapter 3

Ordinary Differential Equations

Differential equations are mathematical equations that explore a function through its derivatives. They provide a fundamental tool for modeling systems that have been used for centuries in a wide variety of fields. As a simple example, consider the motion of a ball thrown into the air, ignoring air resistance. This can be described by the equation $\ddot{y} = -g$, which relates the acceleration of the ball, $\ddot{y}$, to the force of gravity. If we want to know the position of the ball at a particular time t, we can easily integrate the equation twice to find $y(t)$ as a function of the initial velocity and position of the ball. Many interesting differential equations don't admit solutions quite so easily.

As we mentioned in Chapter 2, the Radiative Transfer Equation is expressed as a differential equation. While existing rendering techniques transform this equation into an integral form for application of Monte Carlo integration techniques, our technique solves the RTE directly as a differential equation in the form of an *Initial Value Problem* (IVP). This chapter introduces the techniques used to solve IVPs such as those that arise during rendering with the RTE. Section 3.1 discusses IVPs in general, while Sections 3.2 and 3.3 explore a variety of specific techniques for solving them numerically.

3.1 Initial Value Problems

Solving a differential equation entails finding a function that satisfies the equation. It is not immediately clear that this is a well-posed problem: uniqueness is not guaranteed by the differential equation alone. Further conditions may be necessary to guarantee the uniqueness of the solution. For instance, the equation $y' = m$ describes the slope of a line, but there are infinitely many lines $y = mx + b$ that satisfy this equation. The y-intercept b serves to specify a unique solution in this case.

While it is difficult to determine if an arbitrary differential equation has a unique solution, there are techniques for specific kinds of problems. One such example is that of *Initial Value Problems* (IVPs), in which the problem is specified by an explicit first-order ODE together with the value of the solution to the function at a single point (the initial conditions), denoted by

$$y' = f(x, y), \quad y(x_0) = y_0.$$

This kind of problem arises when the ODE describes the dynamics of a system, and the given value of the function determines the initial state of the system. The RTE, together with the initial radiance of a ray, naturally form an IVP.

Solving IVPs can sometimes be as simple as integrating the function $f(x, y)$ and setting the constant of integration to match the initial conditions. Often, it is too difficult, if not impossible, to analytically integrate f. This is the case for the RTE, namely Eq. 2.2, as the source term is too complex. Numerical techniques for computing approximate solutions to IVPs have existed for centuries, with a particular emphasis in the 20th century given to creating efficient and stable solvers. Most existing methods that are interesting for our problem use a stepwise approach, whereby an approximation of $f(x, y)$ is computed within a window around a known value, then the approximation is integrated for the length of the step size, h, and the process is repeated for the domain of the desired solution. These are often classified based on the amount of error they introduce. It is

useful to distinguish between *local* and *global error*, respectively the error introduced by a single step and then error accumulated over the entire process. This classification is called the *order* of the solver: an order p solver introduces $O(h^{p+1})$ local error at each step, which corresponds to $O(h^p)$ global error. Some *adaptive* solvers also provide methods for adjusting the step size based on computed errors to maintain a desired accuracy.

Given the value of $f(x, y)$ at $(x, y) = (x_0, y_0)$, and a step size of h, solvers are distinguished by the approximation of $f(x, y)$ used to calculate the next step $y_1 = y(x_0 + h)$. In general, all approaches follow the following structure for computing a step:

$$y_{i+1} = y_i + h_{i+1}\Phi(f, x_i, y_{i+1}, y_i, \cdots, y_{i-k+1}), \tag{3.1}$$

where Φ represents the solver, and k denotes the number of previous steps used by the formula. If $k > 1$, i.e. a *multi-step* solver, the solver needs to keep track of previous values. For many solvers, Φ does not depend on y_{i+1}, which allows the formula to be *explicit*, as opposed to *implicit*. Implicit solvers are more computationally intensive, but have greater numerical stability. This is useful for solving *stiff* problems, a loosely defined term that describes IVPs that are more susceptible to numerical instability.

The RTE does not exhibit stiffness, since the terms do not vary rapidly, so we will focus on discussing explicit methods. The next two sections give an overview of various explicit single step and multi-step methods. Our approach to solving the RTE incorporates both of these styles of solvers.

3.2 Single Step Methods

3.2.1 Euler's Method

The simplest solver is known as *Euler's Method*. It works by constructing a linear approximation to $y(x)$ at $x = x_i$, using the derivative value as given by the ODE: $y'(x_i) = f(x_i, y(x_i)) \approx f(x_i, y_i)$. The derivative is recomputed at every step. This

yields the following formula for computing the value of $y(x)$ at the next step:

$$y_{i+1} = y_i + hf(x_i, y_i).$$

Determining the order of a solver usually can be done by comparing the Taylor expansion of $y(x)$ around $x = x_i$ to an expansion of y_{i+1} given by Eq. 3.1. For Euler, this is simple enough that we'll show the error explicitly. The Taylor expansion is

$$y(x_i + h) = y_i + hy'(x_i) + \frac{1}{2}h^2y''(x_i) + O(h^3).$$

Since $y'(x_i) = f(x_i, y_i)$, we see that Euler's Method is based on the Taylor expansion up to the h^2 term. This means the error introduced by one step of Euler's Method is $O(h^2)$. In other words, Euler's Method is an order 1 solver.

3.2.2 Runge-Kutta

Runge-Kutta methods are a class of methods that achieve higher accuracy per step at the cost of additional samples of the function $f(x, y)$. This is done by approximating $y(x)$ along the interval $[x_i, x_{i+1}]$, and then computing $f(x, y)$ at a new point within that interval, using the additional data to construct a more accurate approximation to $y(x)$ for the entire interval. This strategy can be continued with additional data points for increased accuracy. The total number of times this is done per step is denoted by s, yielding an s-stage Runge-Kutta formula. Once all the samples of $f(x, y)$ are collected, they are combined with various weighting schemes to produces the value y_{i+1}:

$$y_{i+1} = y_i + h_{i+1}(\omega_1 k_1 + \omega_2 k_2 + \cdots + \omega_s k_s), \tag{3.2}$$

where k_j is the j-th sample of f, and ω_i is the weight for that sample.

The sampling strategy for Runge-Kutta methods is determined by the following formula for computing the j-th sample:

$$k_j = f(x_i + h_{i+1}\alpha_j, y_i + h_{i+1}\sum_{r=1}^{s}\beta_{jr}k_r). \tag{3.3}$$

The parameter α_j describes the x position of the sample as a proportion of the interval. The y position of the sample is determined by a weighted average of linear approximations to $y(x)$, each approximation corresponding to a separate sample. β_{jr} determines the weight of the r-th sample in the average.

The α, β and ω parameters can be tuned to achieve ease of implementation or error efficiency. They are the defining characteristic of a particular Runge-Kutta method, and are usually described in a *tableau*:

$$\begin{array}{c|cccc} \alpha_1 & \beta_{11} & \beta_{21} & \cdots & \beta_{s1} \\ \alpha_2 & \beta_{12} & \beta_{22} & \cdots & \beta_{s2} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \alpha_s & \beta_{1s} & \beta_{2s} & \cdots & \beta_{ss} \\ \hline & \omega_1 & \omega_2 & \cdots & \omega_s \end{array}$$

Note that values of β_{jr} for $r \geq j$ correspond to the weights of the current and future stages. In the tableau, these are the values on the diagonal and to the right. If any of these values are non-zero, then Eq. 3.3 becomes an implicit equation. In the worst case, all the samples become coupled, and the entire system needs to be solved at once. If $\beta_{jr} = 0$ for $r > j$, but the diagonal is allowed to be non-zero, the formula is called semi-implicit, as the samples can be determined by sequentially solving smaller systems of uncoupled implicit equations. As we will be employing explicit methods, we will only discuss formulas in which $\beta_{jr} = 0$ for $r \geq j$. By far the most common explicit Runge-Kutta method is a 4-th order, 4-stage method called *RK4*. It is so common that it is known as *the* Runge-Kutta method.

Adaptive Runge-Kutta

Adaptive Runge-Kutta methods adjust the step size at every step to satisfy a user-specified tolerance condition. In order to know when to change the step size, we need to be able to compute some error estimate at each step. The simplest way to do this is

to apply an additional higher-order formula simultaneously, with the difference between the results giving an estimate of the local error. Such an additional formula is called a *companion formula*. Clearly, evaluating an additional formula can be costly. Luckily, there are some tableaus that, with only an additional set of weights ω_i, yield two valid formulas of order p and $p+1$. Such $(p, p+1)$ *formula pairs* are described by adding an extra row to the bottom of the tableau. Our solver includes three adaptive solvers. Two are $(4, 5)$ formula pairs, the Cash-Karp[3] and Fehlberg[5] methods, and the last is a $(5, 6)$ formula pair implemented by Hull et al.[7].

Armed with an error estimate, we would like to be able to adjust the step size to be as large as possible while still maintaining a given error tolerance. Given a user-specified tolerance parameter τ and the current step size h_i, we determine that a step is acceptable if its error est_i satisfies the following constraint:

$$|est_i| < h_i\tau.$$

We are looking for a multiplicative factor γ such that if $h_{i+1} = \gamma h_i$, the tolerance constraint is met for at the next step

$$|est_{i+1}| \approx h_{i+1}\tau.$$

Letting z_i be the true solution for the local IVP at the i-th step, the estimate z_i given by a $(p, p+1)$ formula pair satisfies

$$y_i = z_i + ch_i^{p+1} + O(h^{p+2}),$$

where c is a value dependent on the companion formula. This yields

$$|est_i| = ch_i^{p+1} + O(h^{p+2}).$$

Substituting this into our desired equation for γ, we find

$$h_{i+2}\tau > |est_{i+1}| \approx c(\gamma h_i)^{p+1} = \gamma^{p+1}|est_i|$$

Solving for γ, we find

$$\gamma = \left(\frac{\tau h_i}{|est_i|} \right)^{1/p}$$

If we were to use this value directly, we would expect to be always on the very edge of accepting a solution. To deal with additional error, e.g. from floating point arithmetic, our heuristic for choosing the next step size use a "safety factor" of .9:

$$h_{i+2} = .9 \left(\frac{\tau h_i}{|est_i|} \right)^{1/p} h_{i+1}. \tag{3.4}$$

If a step is rejected, this formula is also used to adjust the step size for the current step. This scheme is a well-known heuristic for adaptive step size techniques.

3.3 Multi-step Methods

The most common type of multi-step formula is known as a *Linear Multistep Formula* (LMF). Just as Runge-Kutta methods take a linear combination of new samples of the derivative, LMFs use a linear combination of previous values of the derivative, as well as previous y values.

$$y_{i+1} = \sum_{j=1}^{k} \alpha_j y_{i+1-j} + h \sum_{j=0}^{k} \beta_j y'_{i+1-j} \tag{3.5}$$

If β_0 is non-zero, then the LMF is implicit. As with the single-step solvers, the implicit solvers incur an additional computational cost to achieve greater numerical stability. We focus on the simplest of the LMFs, the Adams-Bashforth methods, which are explicit and have $\alpha_1 = 1$ and $\alpha_j = 0$ for $j \neq 1$:

$$y_{i+1} = y_i + h \sum_{j=1}^{k} \beta_j y'_{i+1-j} \tag{3.6}$$

In essence, this method works by constructing a polynomial that interpolates the derivative y' through the last k data points, then directly integrating that polynomial. The coefficients β_j are picked to accomplish this. These coefficients can be either precalculated or constructed on the fly, which allows variable stepsize and variable order. For an

s-step method, this yields an order s formula. We chose to use a fixed step size, 4-step Adams-Bashforth method:

$$\beta_j = \left(\frac{55}{24}, -\frac{59}{24}, \frac{37}{24}, -\frac{3}{8}\right)$$

A significant advantage of Adams-Bashforth methods lies in that, regardless of the order of the method, the function f is only evaluated once per step. In contrast, an order p Runge-Kutta method requires a minimum of p evaluations per step. Thus, in situations in which f is costly to compute, the Adams-Bashforth method may yield significant computational benefit.

As multi-step methods require data from previous steps, the solver must be boot-strapped. If the solver is variable-order, it may compute the appropriate β_j values for however many steps it has available. Otherwise, a different solver can be used for the first s steps. In our implementation, we used RK4 for the first four steps to boostrap the multi-step method.

3.4 Summary

This chapter introduced the basic theory behind ordinary differential equations and initial value problems. We described a variety of solvers that can be used for general purpose IVPs, as well as the specific choices that form the basis of our method. These techniques perform well for non-stiff problems and are designed to converge on to a good solution while maintaining low computational cost if the solution is sufficiently smooth. While there some discontinuities, notably at scattering events, the ray-parameterized RTE is mostly smooth, which makes these techniques suitable for solving the ray-parameterized RTE. With these tools, we need only fit them into a rendering framework to produce images with participating media. The next chapter discusses the details of implementing these ODE techniques in a ray tracer.

Chapter 4

Implementation

4.1 PBRT

Our software is built as a plug-in to the open-source rendering software bundled with the textbook Physically Based Rendering[17]. The software, known as PBRT, is a fully functional ray tracer, designed with a multitude of extension points to allow researchers to try new rendering techniques without having to write an entire rendering system from scratch. A full description of the software literally fills a textbook, so the discussion here is limited to the parts that affect the rendering of participating media.

When a ray travels between intersections, PBRT passes the ray to a `VolumeIntegrator` class to calculate the ray's interaction with the medium. The volume integrator's `Li()` method calculates the extent of the ray's travel through the volume, if any, and returns the amount of light added to the ray by the medium. An additional method `Transmittance()` is called to determine the proportion T_r of the ray's original light that remains at the other side of the medium. Thus, given a ray with radiance L_0 interacting with the medium, the exiting radiance is the sum of $L_0 T_r$ and the result of calling `Li()`. This models equation 2.3.

PBRT comes with two volume integrator implementations: `emission`, which only

calculates light emitted from the medium, and `single`, which computes emission, absorption, and single-scattering effects. Both of these use ray marching to numerically integrate the volume rendering equation, Eq. 2.3, as parameterized by the ray given. This uses the Monte Carlo estimator, as defined in Eq. 2.7, which means it computes a weighted sum of samples of the integrand at a number of points. These samples are computed by dividing the ray's extent in the medium is divided into segments of uniform length Δx. A sample is drawn from each segment sequentially, and the integrand is computed from that sample point. The results are weighted and summed accordingly. There are some considerations to be taken with the sample within segments. If the initial point is always used, the resulting images exhibit aliasing artifacts. If a random point is used, it is likely that two adjacent segments will have nearby sample points, which leads to higher variance in the image. A compromise is to use a uniform offset into each segment that is selected randomly for each call to `Li()`. Since the samples along the ray are chosen sequentially, this makes the algorithm "march" along the ray.

PBRT represents participating medium itself with the `VolumeRegion` class. This is where the medium's physical boundary is stored, as well as spatially varying properties such as $\sigma_t(\mathbf{x})$, $\sigma_s(\mathbf{x})$, $E(\mathbf{x}, \omega)$, and $p(\mathbf{x}, \omega, \omega_i)$. These attributes are retrieved via method calls, allowing the backing data to be either analytical or sampled. PBRT supplies implementations for homogeneous media, exponential fog (suitable for the atmosphere), and a grid of sampled data. The grid's data is trilinearly interpolated to provide C^0 continuity within the medium.

4.2 Our Contributions

We produced two plug-ins that implement the `VolumeIntegrator` interface using ODE techniques, one with support for single scattering and another one for multiple scattering. As PBRT does not contain a ray-marching multiple scattering integrator, we created an

additional plugin which supports this. We implemented six different ODE solvers to be used by the integrators. Table 4.1 shows the characteristics of the solvers we chose to use. Each solver was written as a separate class inheriting from an `ODESolver` interface. This allowed us to write the volume integration methods without worrying about the details of the ODE techniques.

Name	Order	Step-Size	Multi-step
`euler`	1	Fixed	Single
`rk4`	4	Fixed	Single
`adams`	4	Fixed	Multi
`dverk`	5	Adaptive	Single
`cashkarp`	4	Adaptive	Single
`fehlberg`	4	Adaptive	Single

Table 4.1: ODE Solver Characteristics

The solvers are written such that they can approximate the solution of any first-order ODE. Each `ODESolver` supports two methods, `Solve()` and `Step()`. The solvers are supplied with the ODE by means of a function pointer that evaluates $y' = F(x, y)$. Given the solution value at point (x_i, y_i), `Step()` computes exactly one step of the solver at the given step size h, yielding (x_{i+1}, y_{i+1}), where $x_{i+1} = x_i + h$. Starting from a given point (x_0, y_0), `Solve()` uses `Step()` repeatedly to find the solution to the ODE at desired point x_f, computing intermediate approximations along a grid with equal spacing h. When using adaptive solvers, `Step()` provides an indication of the amount of error, which `Solve()` uses to adjust the step size according to equation 3.4. In this case, `Solve()` does not necessarily compute intermediate approximations that are equally spaced.

`Li()` computes the amount of light exiting the medium along a given ray due to scattering and emission. This suggests an IVP to solve. The ODE specification is given

by the ray parameterization of the RTE in equation 2.2, which is implemented in a callback function that computes $L'(t)$. We need the initial value and the solution domain. Since PBRT has a separate calculation for the attenuation of incoming light due to the medium (the `Transmittance()` method), the initial radiance value is zero. The solution domain is given by intersecting the ray with the boundary of the medium as given by the `VolumeRegion`. Our integrators create one of the six `ODESolver`s based on a user parameter, then supply this information to the `Solve()` method to find the radiance exiting the media along the ray. An important consideration is that the initial point given to `Solve()` needs to be offset by a different random amount for each ray along the ray direction, a process called jittering. Omitting jitter leads to undesirable banding in the final image, where the color in the image exhibits step functions that should not appear. Similar banding would happen in PBRT's ray marching approach if the sample offset into each ray segment is uniform across different rays.

4.2.1 Multiple Scattering

As mentioned in Chapter 2, handling multiple scattering requires considerable computation. Our single scattering integrator mitigates this by approximating $L(\mathbf{x}, \omega_i)$ in the source term with only direct light sources. Computing $L(\mathbf{x}, \omega_i)$ directly via ODE techniques would be prohibitively expensive due to recursion. We decided to incorporate Jensen's photon mapping techniques to simulate multiple scattering[9].

As mentioned in section 2.2, photon mapping is a two-phase technique to accelerate the calculation of indirect illumination. The first phase shoots photons from light sources and traces them through the scene, depositing them in a photon map at each interaction to keep track of illumination throughout the scene. The second phase, the usual rendering process, uses photons stored in the map to estimate indirect illumination at any desired location. We will describe each phase in more detail.

PBRT already includes an implementation of photon mapping for indirect illumina-

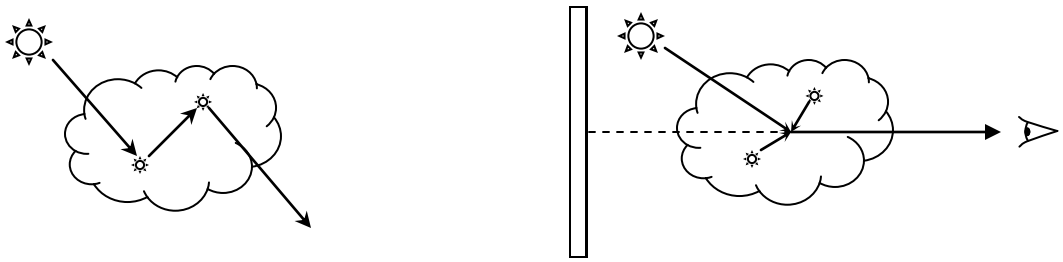

Figure 4.1: (Left) During the first pass, photons are placed in the volume according to rays traced from the light source. (Right) The second pass uses these photons to model the contribution of multiple scattering.

tion of surfaces called `exphotonmap`. Their integrator maintains two photon maps, caustic and indirect, for photons from specular and non-specular paths respectively. We started with this code, and created a `PhotonMap` class also maintains a volume map, and can be shared between both surface and volume integrators. When a photon travels through the medium, our class attempts to record photons within the volume map. After the photon exits the medium, photons will be placed in the surface maps, caustic and indirect.

The `PhotonMap` class has a `Construct()` method that is called during PBRT's pre-processing phase. This method repeatedly traces paths from light sources until either the specified number of photons has been reached or, after tracing 500,000 paths, the number of stored photons is less than a thousandth of the traced paths. For each path, a random light source and direction are sampled. After finding the nearest surface in the chosen direction, we check for interaction with participating medium before reaching the surface. As specified in [9], the cumulative density function describing the probability of a photon interacting with the medium at point x is

$$F(x) = 1 - \tau(x_s, x) = 1 - e^{-\int_{x_s}^{x} \sigma_t(s)ds},$$

where x_s is the point the photon enters the medium, τ is the optical transmittance, and

σ_t is the attenuation coefficient. Thus, a photon at a location x is determined to have interacted with the medium if, for a random parameter $\xi \in [0, 1]$, we find that $\xi < F(x)$. Equivalently, this is true when $\int_{x_s}^{x} \sigma_t(s)ds > -\ln(\xi)$. In other words, we can find a photon interaction location by sampling $-\ln(\xi)$ and solving $\int_{x_s}^{x} \sigma_t(s)ds > -\ln(\xi)$ for x. Practically, this is done by marching along the ray starting at x_s, accumulating $\sigma_t(s)$ until it surpasses $-\ln(\xi)$ or we've exited the medium. In the former case, a photon is recorded in the volume map and a new path direction is sampled according to the phase function. In the latter, the ray has exited the medium and intersected the surface, so a photon is stored in one of the two surface maps, and a new direction is picked according to the BRDF. No more than ten interactions are allowed per path, and Russian roulette is also used to randomly terminate the path tracing at any interaction.

When the desired number of photons is reached, they are formed into kd-trees to assist in the next phase. During the rendering process, whenever the volume integrator needs to know the incoming light, i.e. for the source term, it calls the photon map's `VolumeLi` method. The formula to compute the radiance at a point x is given in [9]:

$$L_i(\mathbf{x}, \omega) = \frac{1}{\sigma_s} \sum_{p=1}^{n} f(\mathbf{x}, \omega_p, \omega) \frac{\Delta\Phi_p(\mathbf{x}, \omega_p)}{\frac{4}{3}\pi r^3}.$$

Φ refers to electromagnetic flux, so $\Delta\Phi_p(\mathbf{x}, \omega_p)$ denotes the radiance contributed by the photon p in the direction ω_p, which is stored in the photon data structure. The photon map looks for the n nearest photons to compute the sum, the furthest of which is used to find the radius of the containing volume r. On note, the use of the n-nearest neighbors is non-linear and introduces bias to Monte Carlo estimators. As in [9], we have the option of not storing volume photons from direct lighting, i.e. the first interaction in a path. This corresponds to separating the multiple and single scattering. In this case, the single scattering term is computed directly as previously, and then added to the light received from the photon map.

4.3 Summary

We introduced the PBRT ray tracing framework upon which we built our renderer. We discussed the Monte Carlo implementation it uses for participating media calculations, and described the additional code we added to support our ODE techniques. Additionally, we described the volume photon mapping algorithm we used to support multiple scattering effects. The next chapter highlights the experiments we conducted with our renderer and the existing system built into PBRT.

Chapter 5

Results

We will compare the performance and behavior of six integrators on various scenes containing participating media with different volumetric properties and surrounding geometry. Each of these integrators is based on a solver described in Sections 3.2 and 3.3. We also experiment with the effects of single and multiple scattering on image quality and rendering time. Finally, we will compare our integration technique against the traditional Monte-Carlo style ray marching integrator built into PBRT. Ground truth renderings were generated using a Monte Carlo ray marching integrator with a very small step size and a large number of samples per pixel. For convenience, the figures are all contained at the end of the chapter.

PBRT contains functionality for tracking statistics both overall and within individual components. This gives us the ability to track both the total time for the rendering process, as well as the time spent only within the ODE solver. Timing at such fine level incurs significant overhead, so only utilized integrator-specific timings for our first experiment. Other times given are total rendering times. Images were computed on two computers, a 2.54 GHz Intel Core 2 Duo and a 3.8 GHz Intel Xeon. Ground truth renderings were computed using a Monte Carlo ray marching integrator. In addition to raising the number of samples per pixel, we selected a step size calculated to let the

rendering take approximately three days.

Each solver has its advantages and disadvantages. Our first experiment shows the effectiveness of using higher order for Runge-Kutta solvers. Figure 5.1 shows a scene rendered with a first-order Euler solver and a fourth-order Runge Kutta solver, as well as a ground truth comparison. These were timed so that the integrators had a time budget of 120 seconds. We can see that the Euler solver exhibits more variance, especially in the close-up of the sphere, while the image produced by the Runge-Kutta is somewhat darker.

Our next experiment compares the multi-step Adams-Bashforth solver against a Runge-Kutta technique, which are both of order 4, as seen in Figure 5.2. The advantage of only requiring one function evaluation per step is clear. Both techniques exhibit similar noise in the ceiling and the lower right box. While some of this can be fixed by increasing the number of camera rays per pixel, as we found was necessary to do for the ground truth, there is a marked decrease in variance for the multi-step algorithm in the highlighted regions. Furthermore, the multi-step algorithm completes an order of magnitude faster.

Figure 5.3 shows the results of our last integrator comparison. Similar to the previous experiment, we pitted a multi-step solver against a single-step Runge-Kutta solver, but this time the single-step solver used adaptive step sizes. Again, the multi-step solver outperforms in both time and quality. The image of the Runge-Kutta solver is much darker. Explicit Runge-Kutta techniques will undershoot the true value of an exponentially decaying function. The absorption component of the RTE exhibits this decaying behavior, which may account for the darkness of the Runge-Kutta image. Furthermore, the artifacts within the shafts of light in the Adams-Bashforth image may be a result of insufficient step size causing the algorithm to miss the true boundaries of the light shafts. The difference in step size between the multistep solver and the ground truth solution is a factor of 50.

The next comparison shows the effects of multiple scattering on the final image. Figure 5.4 shows a scene rendered with and without multiple scattering. As this is primarily to highlight the visual differences between single and multiple scattering rather than the choice of solver, we only show the results of the Euler integration, and omit a ground truth rendering. Multiple scattering enables visualization of volume caustics, as well as contributing to scene brightness by considering light paths that a single scattering integrator does not.

We also compared the performance of single and multistep solvers against the Monte Carlo integrator in a scene with smoke from a volumetric data source. As described in Chapter 4, this data is linearly interpolated. Each rendering used the same step size of 0.05 world units. A tolerance of 0.01 was used for the adaptive integrator (See Section 3.2.2 for information on how tolerance is handled). The only difference is the choice of integrator. We selected a fairly large step size in order to accentuate the errors that arise from each technique when using insufficient step sizes. Figure 5.5 shows the results of this test. At the quality setting used, the ODE techniques appear to overestimate the light within the smoke, and there are artifacts within the front lobe of the smoke with the Adams multistep method. Otherwise, the time required for all the techniques is of the same order.

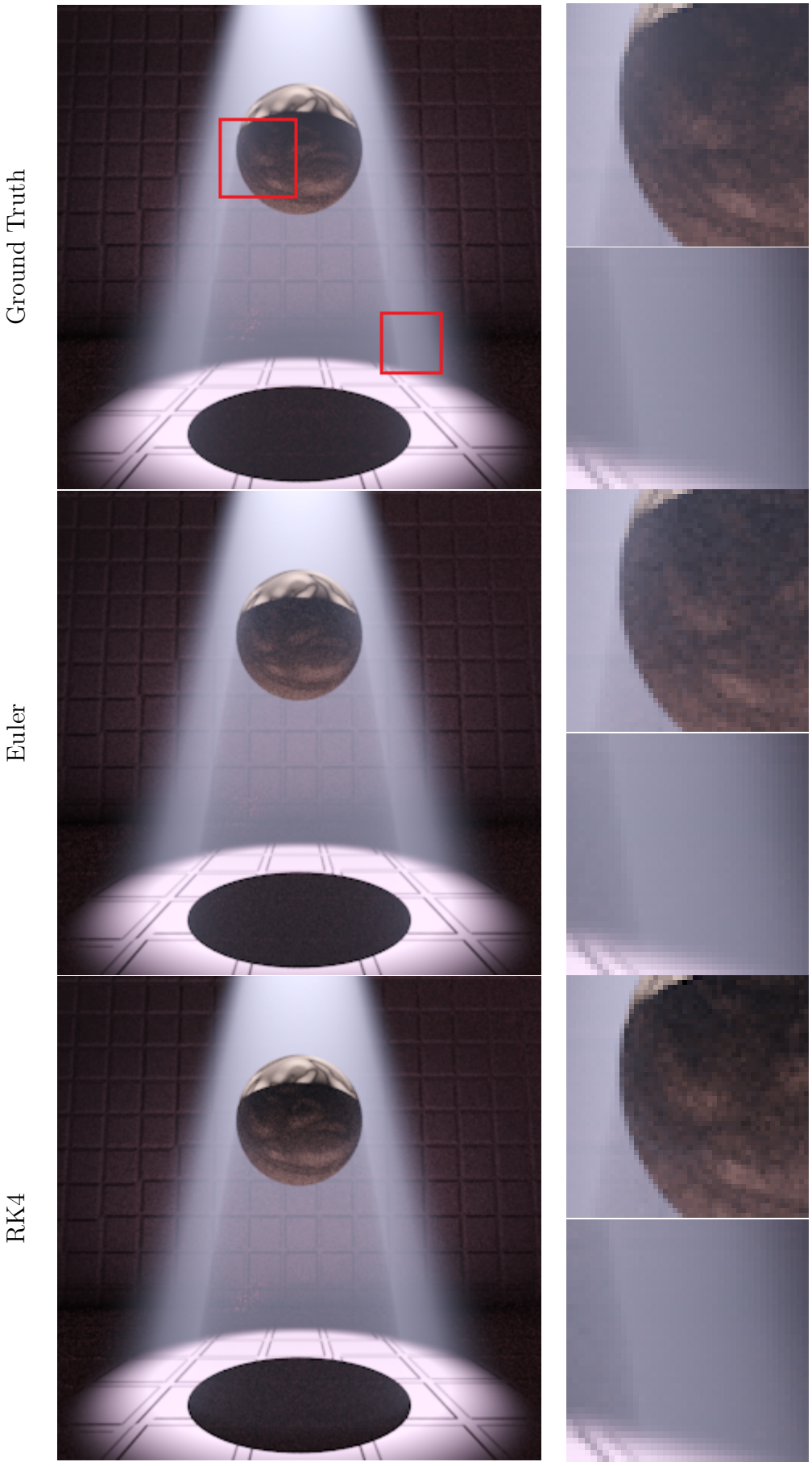

Figure 5.1: Effects of order on a solution with fixed computational budget (120 seconds integrator time, 2.54 Ghz CPU)

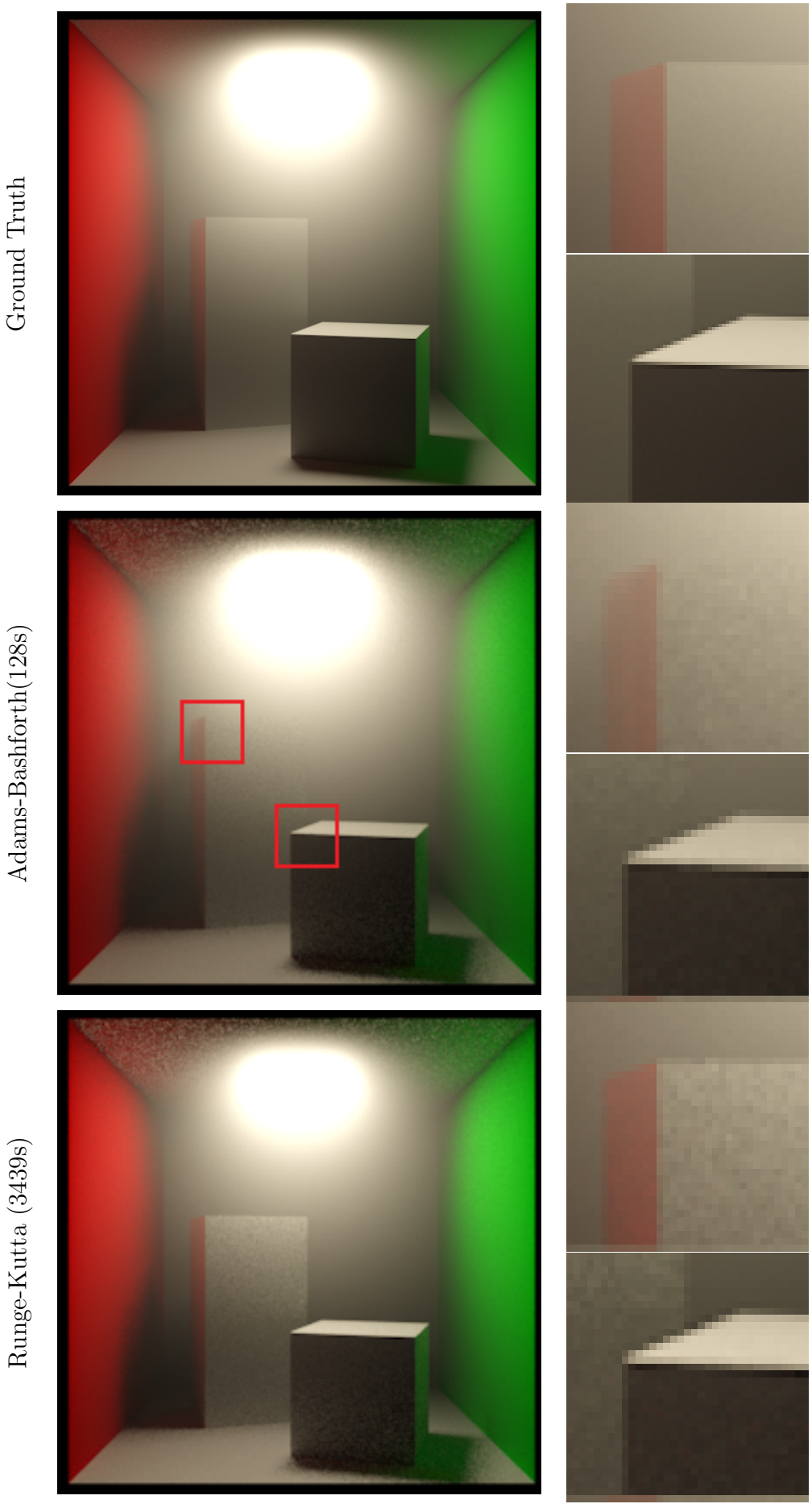

Figure 5.2: One and multi-step algorithms (2.54 GHz).

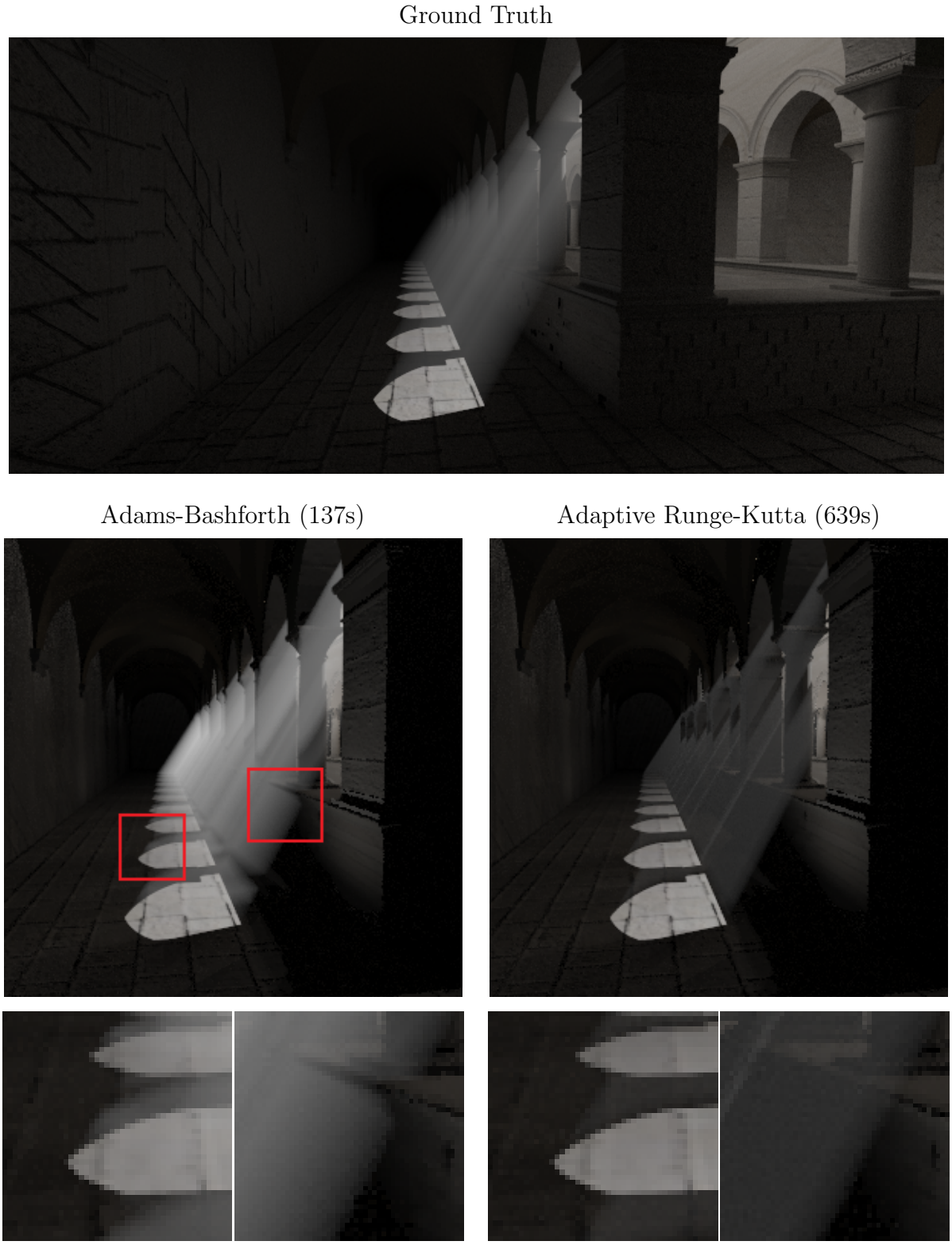

Figure 5.3: Comparison of adaptive step-size Runge-Kutta and constant step-size Adams-Bashforth solvers (2.54 GHz).

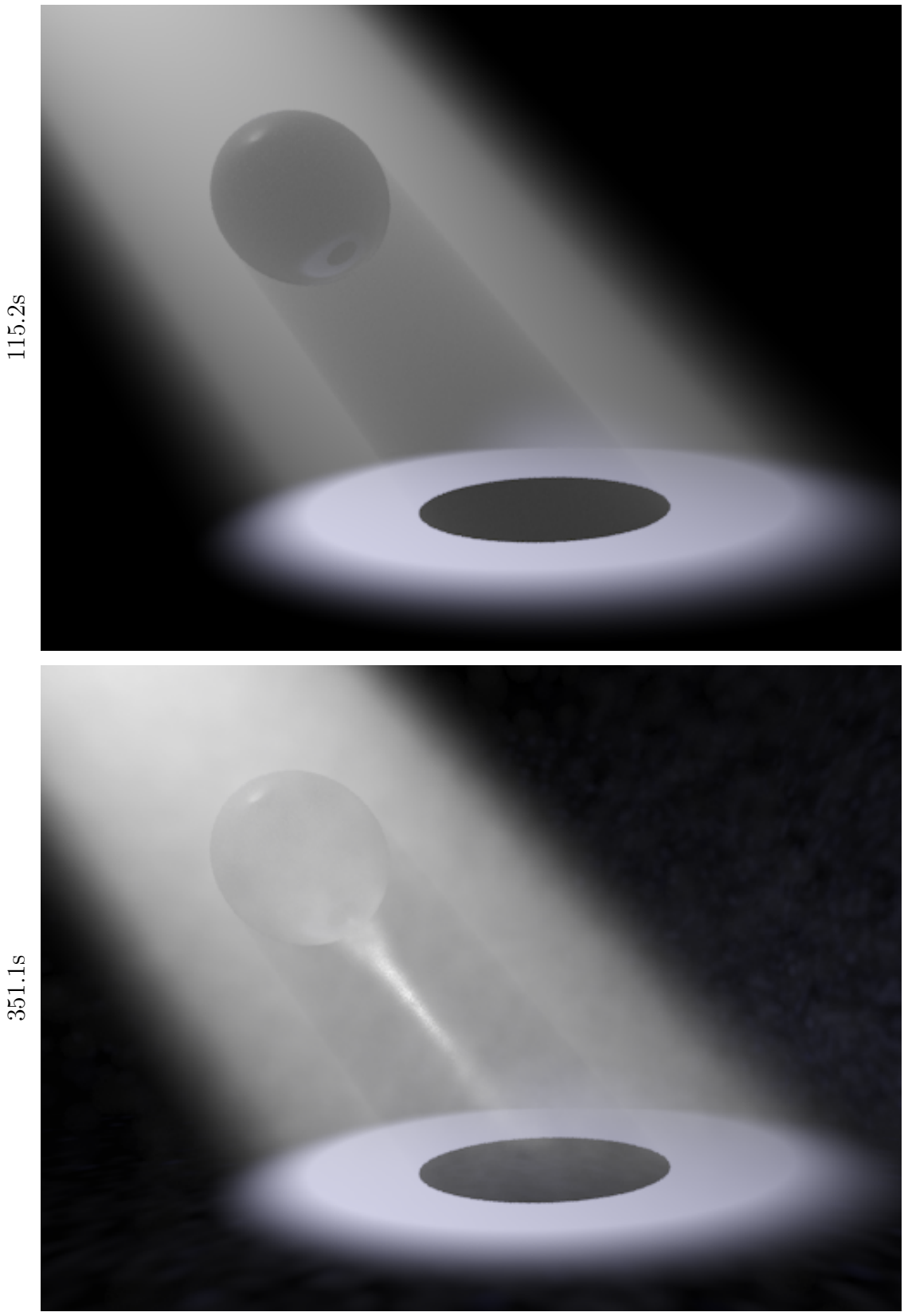

Figure 5.4: Spotfog scene exhibiting scattering effects (3.8 GHz). (Top) Rendered by Euler integration with single scattering only. (Bottom) Rendered by Euler integration with multiple scattering.

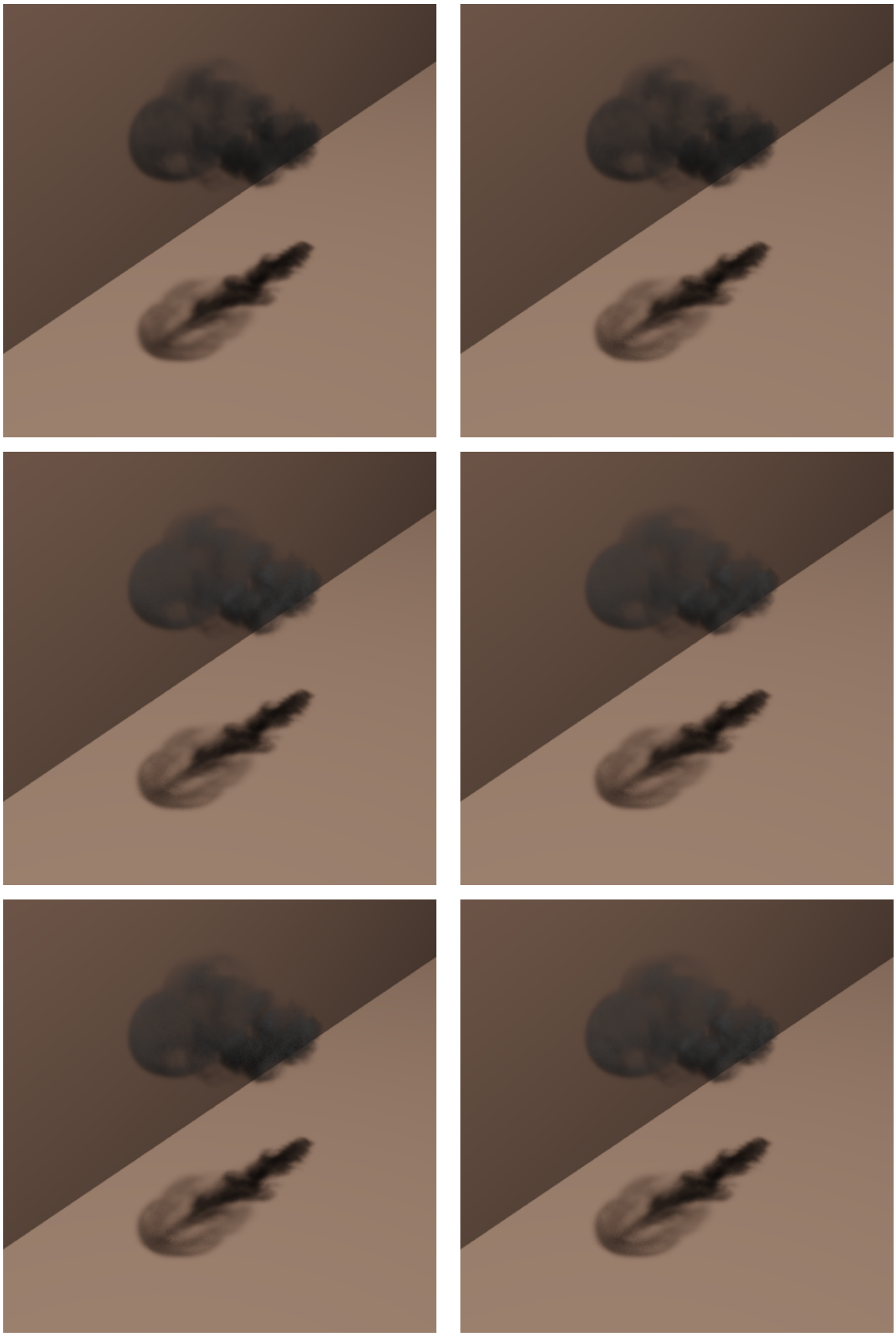

Figure 5.5: Smoke data set rendered with many techniques (3.8 GHz). In order from the top left: Ground Truth, Monte Carlo (58.9s), Euler (50.3s), RK4 (89.6s), Cash-Karp (87.6s), and Adams (51.2s).

Chapter 6

Conclusions and Future Work

6.1 Conclusions

We introduced a new method of solving the Radiative Transport Equation for the purpose of rendering participating media. Traditional ray tracers use Monte Carlo integration to perform this computation, as the technique is suited to a wide number of problems encountered during ray tracing. Instead, we maintain the RTE in differential form, allowing us to leverage a large body of research in efficient methods for solving ordinary differential equations. We built a system utilizing these techniques as a module for the open source ray tracing software PBRT, allowing us, again, to draw upon existing techniques for improving rendering efficiency and fidelity. Maintaining their modular design, our system makes implementing additional ODE solvers that plug into our RTE solver a simple task.

We tested a variety of solvers, including both single and multistep solvers as well as adaptive and fixed step size solvers. We found that Euler methods are comparable to existing Monte Carlo techniques, sometimes outperforming the Monte Carlo techniques by a small amount. Higher order Runge-Kutta techniques and multistep Adams methods afford a moderate performance gain as well. However, there are drawbacks in that these

high order techniques are may have additional aliasing errors at low quality settings.

6.2 Future Work

There has been work on improving Monte Carlo techniques for ray tracing, known as Metropolis Light Transport, or MLT[19]. The technique uses the path integral formulation of light transport, using the Metropolis sampling algorithm to form new paths by permuting existing ones. This helps overcome the problem with traditional Monte Carlo ray tracing that paths which contribute to the image brightness can be difficult to find in complex scenes. While MLT was initially developed without consideration for participating media, it has been adapted to incorporate participating media[16]. It would be interesting to compare the results of the refined Monte Carlo techniques to those of our own.

Among the solvers we tested, we found that the multistep solvers had the best performance, but there remain other solvers that can be applied to the same problem. Implicit solvers or other symplectic integrators may be able to better deal with the energy dissipation noticed in figure 5.3, but we do not expect the extra overhead to be a sufficient gain. We would like to experiment with adaptive multistep solvers. In this case, there are both variable step size and variable order methods available. Additionally, there are noted discontinuities in the RTE at scattering events and volume phenomena boundaries, such as spotlights. We could taylor the ODE solver to detect these discontinuities and handle them appropriately.

Participating media is an important part of movie special effects. Another area to explore is the application of ODE techniques to animations, beyond just static images. As seen in our results, the solvers each have different noise characteristics. It would be interesting to see how the noise appears visually in sequences of animation frames.

6.3 Closing Thoughts

This thesis shows that ODE techniques present a viable alternative to Monte Carlo techniques for solving the RTE in a ray tracing setting. Having identified areas where efficiency has been improved, and some where results can still be refined, we hope this will contribute to further work in increased efficiency in rendering participating media.

Bibliography

[1] Philippe Blasi, Bertrand Le Saëc, and Christophe Schlick. A rendering algorithm for discrete volume density objects. *Computer Graphics Forum*, 12(3):201–210, 1993.

[2] James F. Blinn. Light reflection functions for simulation of clouds and dusty surfaces. *SIGGRAPH Comput. Graph.*, 16(3):21–29, 1982.

[3] J. R. Cash and Alan H. Karp. A variable order runge-kutta method for initial value problems with rapidly varying right-hand sides. *ACM Trans. Math. Softw.*, 16(3):201–222, 1990.

[4] S Chandrasekhar. *Radiative Transfer*. Dover Publications, NY, 1960.

[5] Fehlberg E. Low-order classical Runge-Kutta formulas with step size control and their application to some heat transfer problems. Technical Report 315, NASA, 1969.

[6] L. G. Henyey and J. L. Greenstein. Diffuse radiation in the galaxy. *Astrophysical Journal*, 93:70–83, January 1941.

[7] T.E. Hull, W.H. Enright, and K.R. Jackson. Users guide for DVERK - a subroutine for solving non-stiff ODEs. Technical Report 100, University of Toronto, 1976.

[8] Henrik Wann Jensen. Global illumination using photon maps. In *Proceedings of the eurographics workshop on Rendering techniques '96*, pages 21–30, London, UK, 1996. Springer-Verlag.

[9] Henrik Wann Jensen and Per H. Christensen. Efficient simulation of light transport in scences with participating media using photon maps. In *SIGGRAPH '98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 311–320, New York, NY, USA, 1998. ACM.

[10] James T. Kajiya. The rendering equation. *SIGGRAPH Comput. Graph.*, 20(4):143–150, 1986.

[11] James T. Kajiya and Brian P Von Herzen. Ray tracing volume densities. *SIGGRAPH Comput. Graph.*, 18(3):165–174, 1984.

[12] 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, London, UK, 1996. Springer-Verlag.

[13] Nelson Max. Light diffusion through clouds and haze. *Comput. Vision Graph. Image Process.*, 33(3):280–292, 1986.

[14] Nelson Max. Efficient light propagation for multiple anisotropic volume scattering. In *Proceedings of the 5th Eurographics Workshop on Rendering*, pages 87–104, 1994.

[15] Gustav Mie. Beiträge zur optik trüber medien, speziell kolloidaler metallösungen. *Annalen der Physik*, 330:377–445, 1908.

[16] 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, London, UK, 2000. Springer-Verlag.

[17] Matt Pharr and Greg Humphreys. *Physically Based Rendering from Theory to Implementation*. Morgan Kaufmann, 2004.

[18] Holly E. Rushmeier and Kenneth E. Torrance. The zonal method for calculating light intensities in the presence of a participating medium. In *SIGGRAPH '87:*

Proceedings of the 14th annual conference on Computer graphics and interactive techniques, pages 293–302, New York, NY, USA, 1987. ACM.

[19] Eric Veach and Leonidas J. Guibas. Metropolis light transport. In *SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 65–76, New York, NY, USA, 1997. ACM Press/Addison-Wesley Publishing Co.

[20] Turner Whitted. An improved illumination model for shaded display. *Commun. ACM*, 23(6):343–349, 1980.