Particle Based Participating Media Rendering Using Density Octrees

by

Richard Elliot Monette

A Thesis submitted to
the Faculty of Graduate Studies and Research
in partial fulfilment of
the requirements for the degree of
Master of Science
in

Systems and Computer Engineering
Carleton University
Ottawa, Ontario, Canada
September 2011

Library and Archives Canada

Published Heritage Branch

395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque et Archives Canada

Direction du Patrimoine de l'édition

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence
ISBN: 978-0-494-87846-0

Our file Notre référence
ISBN: 978-0-494-87846-0

Abstract

In order for computer generated imagery to recreate the characteristic visual appearance of phenomena such as smoke and fog it is necessary to compute the way in which light interacts with participating media. In this thesis we present a novel technique for computing volumetric single scattering lighting solutions for particle based inhomogeneous participating media data sets. We seek to calculate volumetric lighting solutions for particle based data sets as such data sets have the advantage of being spatially unbounded and relatively unrestricted with regard to memory as compared to uniform grids. In order to perform the calculations which are required for computing such a lighting solution, we introduce a novel octree based data structure. We refer to this new data structure as a density octree. The design of the density octree allows for efficiently computing light attenuation throughout the spatial extent. Using our new algorithm and data structure, we are able to produce high quality output imagery of arbitrary particle data sets in the presence of arbitrary numbers of lights.

Table of Contents

Abstract ii

Table of Contents iii

List of Figures vi

List of Algorithms viii

1 Introduction 1

2 Overview 3

2.1 Participating Media Interaction Processes 4

2.1.1 Absorption 5

2.1.2 Emission 6

2.1.3 Scattering 7

2.1.4 In-Scattering 9

2.2 The Equation of Transfer 11

2.3 Volume Representation 13

2.3.1 Eulerian / Grid Based 14

2.3.2 Lagrangian / Particle Based 14

2.4 Ray Marching 15

2.5 Classic Volume Rendering Techniques 16

2.5.1 Ray Tracing Methods 16
2.5.2 Photon Mapping Methods 17

3 Related Work **21**

3.1 Ray Traversal of Octree Point Clouds on the GPU 21
3.2 Deep Shadow Maps 22
3.3 GPU Deep Shadow Map Techniques 24
3.3.1 Opacity Shadow Maps 24
3.3.2 Deep Opacity Maps 25
3.3.3 Fourier Opacity Mapping 26
3.4 Ray Marching Using Sparse Block Grids 26

4 Particle Based Participating Media Rendering Using Density Octrees **28**

4.1 Single Scattering Volumetric Lighting Calculation 30
4.1.1 Density Octree 30
4.1.2 Lighting Evaluation 32
4.2 Intersection Testing Routines 33
4.3 Temporal Coherence 35
4.4 Solid Geometry Volumetric Shadowing 35
4.5 Performance Enhancements 36
4.5.1 Early Termination 36
4.5.2 Multi-threading Support 37
4.6 Volumetric Lighting Calculation User Parameters 37
4.6.1 Shadowing Density 37
4.6.2 Solid Geometry Shadow Intensity 38
4.6.3 Particle Sub-Sampling 38
4.6.4 Particle Intersection Testing Size 38

4.7 Forward Density Integration using GPU Hardware 39
4.7.1 Motion Blur 40
4.7.2 Depth of Field 40

5 Results **42**
5.1 Performance 43
5.2 Visual Quality 47
5.3 Discussion 52

6 Conclusion **57**

7 Future Work **58**
7.1 Level of Detail 58
7.2 Multiple Scattering 59
7.3 Ambient Occlusion 59
7.4 Particle Replication 59

List of References **60**

List of Figures

1 Image rendered with Fury 2
2 Example of homogeneous and in-homogeneous medium 4
3 Radiance reduction due to absorption 5
4 Emissive light source 6
5 Radiance increase due to emission 7
6 Radiance reduction due to out-scattering 10
7 Radiance increase due to in-scattering 11
8 Two dimensional uniform grid 14
9 Single scattering ray tracing solution 17
10 Multiple scattering photon mapping solution 20
11 Single vs. Multiple Scattering Example 20
12 Deep shadow maps 22
13 Deep shadow map construction 23
14 Opacity shadow maps 25
15 Deep opacity maps 26
16 Particle rendering pipeline 30
17 Loose octrees 31
18 Octree formation 32
19 Density octree intersection 33
20 Volumetric geometry shadows 36

21 GPU Particle Rendering Effects 40
22 Time vs. Number of Particles 44
23 Time vs. Number of Lights 45
24 Image Size vs. Time 46
25 Density Tree Coarseness vs. Time 46
26 Krakatoa vs. New algorithm visual comparison #1 48
27 Krakatoa vs. New algorithm visual comparison #2 49
28 Effects of particle size 50
29 Effects of particle density 51
32 Volumetric Geometry Shadows 54
33 Particles casting shadows on geometry 55

List of Algorithms

1 Particle attenuation calculation pseudocode 33

2 Recursive density octree intersection routine 34

3 Ray-sphere distance routine 35

Chapter 1

Introduction

One of the most compelling aspects of computer graphics is the rendering of so called participating media phenomena such as smoke, fog and clouds. These phenomena are referred to as participating media in that their presence in a virtual scene to be rendered affects the transfer of light between the solid surfaces and the camera. Aesthetically pleasing treatment of these phenomena is critical to achieving high quality computer graphics imagery. This thesis presents a high quality, temporally coherent and computationally efficient technique for rendering such participating media. Specifically, we introduce a new method for computing volumetric lighting solutions for participating media which is modeled by particle data sets, as opposed to traditional grid based representations. We seek to render particle based participating media because a range of modern fluid dynamics computation solutions use particle based data sets, instead of traditional grid based density tracking.

The primary contribution of this thesis is a novel technique which allows for computing single scattering volumetric lighting solutions for in-homogeneous particle based partcipating media data sets consisting of arbitrary numbers of particles for an arbitrary number of lights of various types. The technique presented exhibits logarithmic complexity with respect to the number of particles and linear complexity with regard to the lights being considered. This performance characteristic is

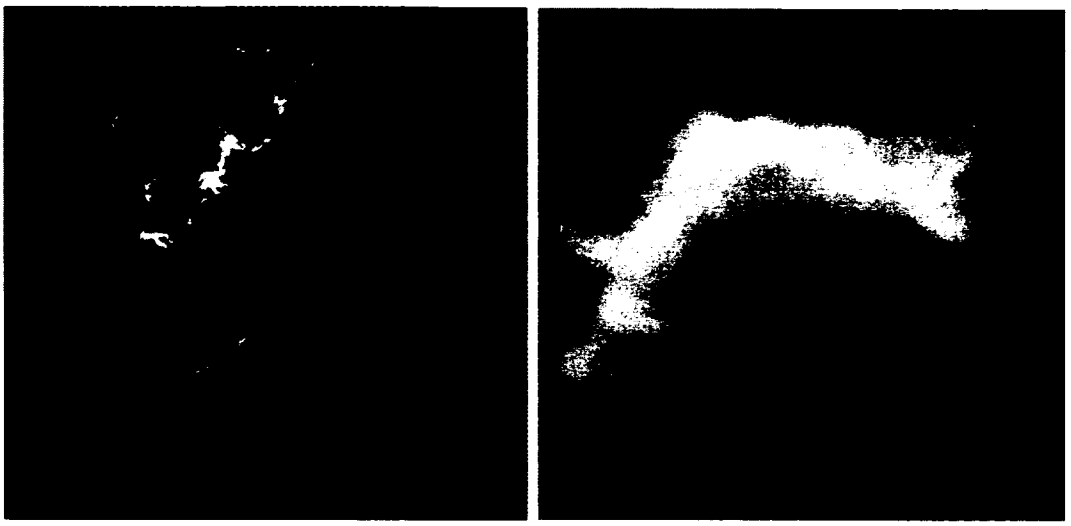


Figure 1: Images rendered using the Exocortex Fury software product which integrates the algorithm described in this thesis.

achieved by forming a density octree which models the particle density throughout the spatial extent. The density octree allows for computationally efficient evaluation of attenuation along each light path between the particles and the lights in the data set. Our work extends that conducted by Knoll, who uses a similar octree method, however only to represent solid geometry. [1] In addition, this thesis describes the computer software system implemented by the author in order to evaluate the new volumetric rendering algorithm. The volumetric lighting software routines developed for this thesis have been included in the Exocortex Fury renderer, a production ready and utilized software product.

In Chapter 2, we begin by establishing a theoretical basis which describes the various types of interactions which light may have with participating media. A survey of relevant research is provided in Chapter 3. In chapter 4, we present a definition of the problem which our technique addresses and describe in detail our proposed technique by which we resolve this problem. In Chapter 5, we present our performance results and a selection of example output images. We conclude with a discussion of directions for future work and conclusions in Chapters 6 and 7, respectively.

Chapter 2

Overview

In general, computer graphics solutions rely upon the assumption that the scenes to be rendered are made up of a collection of solid surfaces existing in a vacuum. This assumption greatly simplifies rendering since, in a vacuum, radiance (the measure of brightness and color of a single ray of light [2]) is constant along any path between the surfaces, lights and virtual cameras. However, this assumption, while beneficial in terms of reducing computational complexity, has the shortcoming of preventing the rendering of effects such as attenuation and scattering of light due to participating media such as fog, smoke and other atmospheric phenomena.

To capture the appearance of such phenomena, we must use those techniques which calculate the interaction of the light with the participating media present in the scene. Participating media is that matter, such as water droplets (in the case of fog) or dust particles (in the case of smoke), which will affect the behavior of light by changing the direction in which it travels and the amount of energy it carries due to processes such as scattering and absorption.

For a complete treatment of following topics we refer to the reader to the comprehensive resource, Physically Based Ray Tracing [3]. A summary of the salient information from that text is presented here for the readers convenience. We first introduce the main processes by which light interacts with participating media. Then

we introduce the equation of transfer which describes how light behaves in participating media. This is followed by a discussion of grid and particle based means of storing volumetric data. We overview the ray marching technique which is a standard method by which volumetric data can be integrated. Finally, we outline the classic ray tracing and photon mapping methods for solving the volumetric lighting problem.

2.1 Participating Media Interaction Processes

Participating media is characterized as being either homogeneous or in-homogeneous in nature. A homogeneous media is uniform throughout the scene whereas an in-homogeneous media spatially varies in properties such as density, color or other attributes.

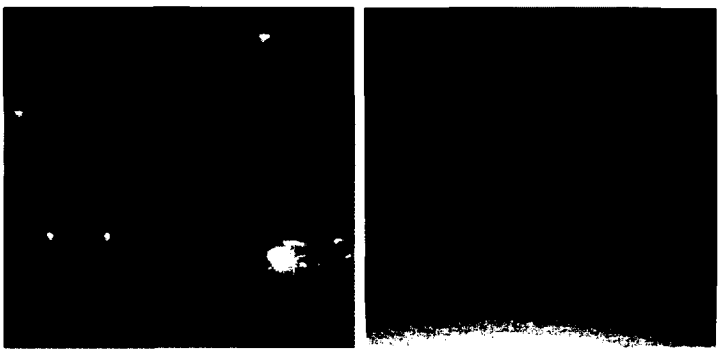


Figure 2: Two scenes, one which is enveloped in a uniform, homogeneous fog and the other of an in-homogeneous smoke plume. Image (left) used with permission. ©Wenzel Jakob Image (right) used with permission. ©Henrik Wann Jensen

Four main processes affect the distribution of radiance in a scene in the presence of some participating media:

- Absorption - reduction in radiance due to the conversion of light to another form of energy such as heat
- Emission - radiance that is added to the environment from luminous particles

- In-Scattering - radiance that interacts with particles, changing direction such that it then travels along the viewing ray to the camera
- Out-Scattering - radiance that interacts with particles, changing direction such that it does not travel along the viewing ray to the camera

Next, we examine each of these processes in detail.

2.1.1 Absorption

Absorption is the process by which radiance interacts with participating media reducing the total transmitted radiance. Absorption is measured using the absorption coefficient σ_a which is the probability that light is absorbed per unit distance traveled in the medium. In the case on an in-homogeneous medium this absorption coefficient may be spatially varying, whereas it will be uniform in a homogeneous medium. If spectrally accurate rendering is required, it is possible to model an absorption coefficient which varies with respect to the various wavelengths of the light.

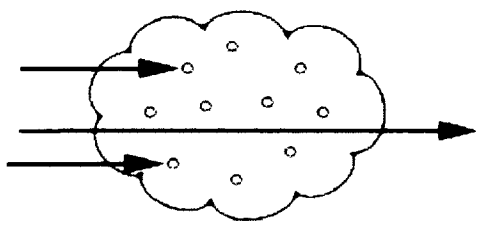


Figure 3: Incident radiance, L_i, is reduced due to absorption resulting in less outgoing radiance.

Given some incident radiance $L_i(p, \omega)$, at position p traveling along vector ω, we find the change in outgoing radiance $L_o(p, \omega)$ after the absorption interaction with the participating media using equation 1:

$$dL_o(p, \omega) = -\sigma_a(p, \omega)L_i(p, -\omega)dt \tag{1}$$

2.1.2 Emission

Emission increases the amount of radiance traveling along some path through a participating medium due to the conversion of energy into visible light by chemical, thermal or other means. For example, consider the glowing hot gas of a torch. The burning gas releases energy, some of which will be in the form of visible light. The emission function, $L_{ve}(p, w)$, describes the change in radiance as a ray moves through a volume of emissive particles.

Figure 4: A burning candle flame is an emissive participating media

The change in radiance per unit of differential distance is:

$$dL = L_{ve}dt \tag{2}$$

If we consider the path from a point in the emissive participating media to the camera to be made up of some N finite steps, then the sum of the the emissive energy will be, where $L_{ve}i$ is the emissive energy at each step:

$$L_{ve}total = \sum_{i=0}^{N} L_{ve}i \tag{3}$$

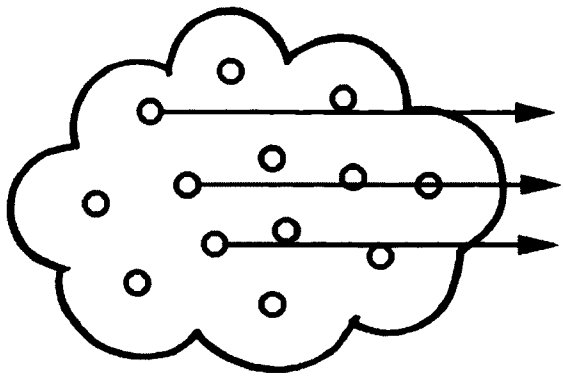


Figure 5: Particles in an emissive participating media emit light energy

2.1.3 Scattering

As a beam of light passes through some participating media, it may intersect with the particles that constitute that media. This interaction can cause the light to change the direction in which it is traveling. The effects of this scattering are two fold. The first outcome is that the total radiance will be reduced due to the out-scattering of light energy. However, it is also possible that light will enter the medium and be scattered such that it increases the total radiance which reaches the virtual camera. This outcome, where the total radiance is increased, is referred to as in-scattering. We examine both in and out-scattering in greater detail in the next two sections.

Out-Scattering

The probability of an out-scattering event occurring per unit distance is given by the scattering coefficient, σ_s. As with the absorption coefficient, the reduction in radiance along a differential length of dt due to out-scattering is given by:

$$dL_o(p,\omega) = -\sigma_s(p,\omega)L_i(p,-\omega)dt \tag{4}$$

Thus, the total reduction in radiance due to absorption and out-scattering is the sum of $\sigma_a + \sigma_s$. This combined effect of absorption and out-scattering is called attenuation or extinction. For convenience, the sum of these two coefficients is denoted by the attenuation coefficient σ_t:

$$\sigma_t(p, \omega) = \sigma_a(p, \omega) + \sigma_s(p, \omega) \tag{5}$$

Given the attenuation coefficient σ_t, the differential equation describing overall attenuation,

$$\frac{dL_o(p, \omega)}{dt} = -\sigma_t(p, \omega) L_i(p, -\omega), \tag{6}$$

can be solved to find the beam transmittance, which gives the fraction of radiance that is transmitted between two points on a ray:

$$T_r(p \rightarrow p') = e^{-\int_d^0 \sigma_t(p+t\omega, \omega) dt} \tag{7}$$

where d is the distance between p and p', σ is the normalized direction vector between them and T_r denotes the beam transmittance between p and p'. Note that the transmittance is always between zero and one, as it represents a normalized percentage of radiance which passes between two points. Thus, if radiance exiting a point p on a surface in a given direction ω is given by $L_o(p, \omega)$, after accounting for extinction, the incident radiance at another point p' in direction $-\omega$ is

$$T_r(p \rightarrow p') L_o(p, \omega) \tag{8}$$

Two useful properties of beam transmittance are that transmittance from a point to itself is one ($T_r(p \rightarrow p) = 1$ and in a vacuum $T_r(p \rightarrow p') = 1$ for all p'. Another important property, which is true in all media, is that transmittance is multiplicative

along points on a ray:

$$T_r(p \to p'') = T_r(p \to p')T_r(p' \to p''), \tag{9}$$

for all points p' between p and p''. This property is important for volume scattering implementations, since it makes it possible to incrementally compute transmittance at many points along a ray by computing the product of each previously computed transmittance with the transmittance for its next segment.

Given then some path beginning at p_0 and travelling for N discrete finite steps, the transmittance would be calculated as follows:

$$T_r(p_0, p_N) = \prod_{i=0}^{N-1} T_r(p_i \to p_{i+1}) \tag{10}$$

The negated exponent in T_r is called the optical thickness between two points. It is denoted by the symbol τ:

$$\tau(p \to p') = \int_0^d \sigma_t(p + t\omega, -\omega)dt \tag{11}$$

In a homogeneous medium, where σ_t is constant, τ is trivially evaluated and yields Beer's law:

$$T_r(p \to p') = e^{-\sigma_t dt} \tag{12}$$

2.1.4 In-Scattering

While out-scattering reduces radiance along a ray due to radiance scattering out in different directions, in-scattering accounts for increased radiance due to radiance scattering in from other directions.

The total radiance added per unit distance due to in-scattering is given by the

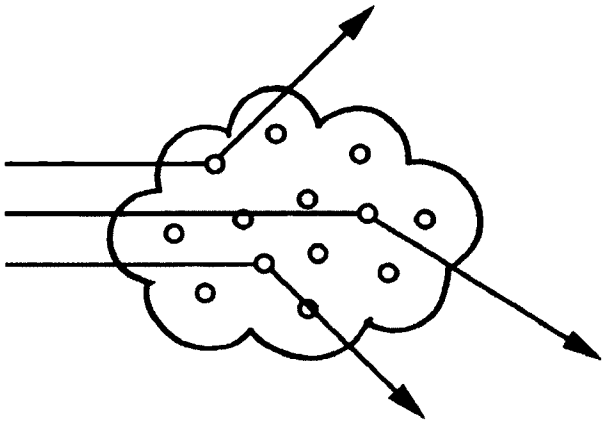


Figure 6: Out-scattering reduces the radiance as lights is scattered to a direction other than along the viewing ray

source term S:

$$dL_o = S(p, \omega)dt \tag{13}$$

which accounts for the increase in radiance from both emission, L_{ve}, and in-scattering. The in-scattering is computed by integrating the unit sphere around the point to sample the incident radiance, L_i, from all directions, *total* L_i.

$$total\ L_i = \int_{S^2} p(p, -\omega' \rightarrow \omega) L_i(p, \omega') d\omega' \tag{14}$$

$$S(p, \omega) = L_{ve}(p, \omega) + \sigma_s(p, \omega)\ total\ L_i \tag{15}$$

This equation states that the total added radiance is the emitted radiance plus all the direct lighting radiance L_i that is being in-scattered from all directions.

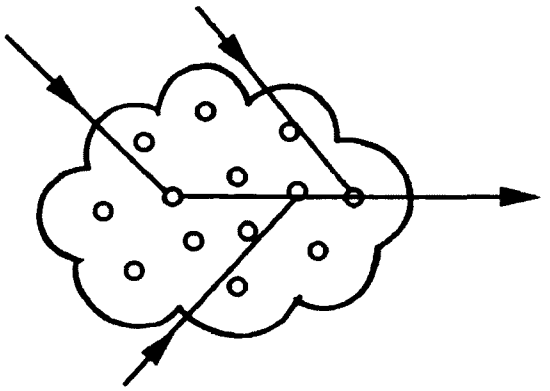


Figure 7: In-scattering increases radiance as light is scattered such that it begins to travel along the orientation of the viewing ray

2.2 The Equation of Transfer

The equation of transfer is the fundamental equation that governs the behavior of light in a medium that absorbs, emits and scatters radiation [4]. It accounts for all of the volume scattering processes, which are; absorption, emission and in- and out-scattering. These factors provide an equation that describes the distribution of radiance in an environment. In fact, the light transport equation is a specialized case of the equation of transfer, which has been simplified by the removal of consideration of participating media and specialized only to scattering from solid surfaces. [5]

The equation of transfer is an integro-differential equation that describes how the radiance along a beam changes at a point in space. It can be transformed into a pure integral equation that describes the effect of participating media from the infinite number of points along a line. It can be derived in a straightforward manner by subtracting the effects of the scattering processes that reduce energy along a beam (absorption and out-scattering) from the processes that increase energy along it (emission and in-scattering).

We recall that the source term gives the change in radiance at a point p in a

particular direction w due to emission and in-scattered light from other points in the medium:

$$S(p,\omega) = L_{ve}(p,\omega) + \sigma_t(p,\omega)L_i(p,-\omega)dt \tag{16}$$

The source term, L_i, accounts for all of the processes by which radiance is added to a ray.

The attenuation coefficient, σ_t, accounts for all processes that reduce radiance at a point: absorption and out-scattering. The differential equation that describes this effect is

$$dL_o(p,\omega) = -\sigma_t(p,\omega)L_i(p,-\omega)dt \tag{17}$$

The overall differential change in radiance at a point p' along a ray is found by adding these two effects together to get the integro-differential form of the equation of transfer (the equation is integro-differential due to the integral over the sphere in the source term):

$$\frac{\theta}{\theta t}L_o(p,\omega) = -\sigma_t(p,\omega)L_i(p,-\omega) + S(p,\omega) \tag{18}$$

With suitable boundary conditions, this equation can be transformed into a purely integral equation. For example, if one was to assume that there are no surfaces in the scene such that the rays are never blocked and have an infinite length, then the integral equation of transfer would be

$$L_i(p,\omega) = \int_0^\infty T_r(p' \to p)S(p',-\omega)dt \tag{19}$$

where $p' = p + \omega$. The meaning of this equation is reasonably intuitive: it states that the radiance arriving at a point, p, from a given direction, ω, is the sum of the

radiance along all points along the ray from the point. The amount of added radiance at each point along the ray that reaches the ray's origin is reduced by the total beam transmittance from the ray's origin to the point.

In a more likely scenario, where there are reflecting and/or emitting surfaces in the scene, rays won't necessarily have infinite length and the surface that a ray hits affects its radiance, adding outgoing radiance from the surface at the point and preventing radiance from points along the ray beyond the intersection from contributing to radiance at the ray's origin. If a ray (p, ω) intersects a surface at some point p_0 a distance t along the ray, then the integral equation of transfer is

$$L_i(p, \omega) = T_r(p_0 \rightarrow p) L_o(p_0, -\omega) + \int_0^t T_r(p' \rightarrow p) S(p', -\omega) dt' \tag{20}$$

where $p_0 = p + t\omega$ is the point on the surface and $p' = p + t'\omega$ are points along the ray.

This equation describes the two effects that contribute to radiance along the ray. First, reflected radiance back along the ray from the surface is given by the term L_o, which gives the emitted radiance and reflected radiance from the surface. This radiance may be attenuated by the participating media; the transmittance from the ray origin to the point p_0 accounts for this. The second term accounts for the added radiance along the ray due to volume scattering and emission, but only up to the point where the ray intersects the surface; points beyond that one don't affect the radiance along the ray.

2.3 Volume Representation

In order to render participating media, its properties must be tracked throughout the spatial extent of the volume in which it resides. The two common means by which this is achieved are the use of Eulerian (grid) and Lagrangian (particle) based methods,

which are outlined in greater detail in the following two sections.

2.3.1 Eulerian / Grid Based

The traditional, and still one of the most popular, methods used to track pressure and velocity in fluid simulations is a uniform grid. [6] [7]

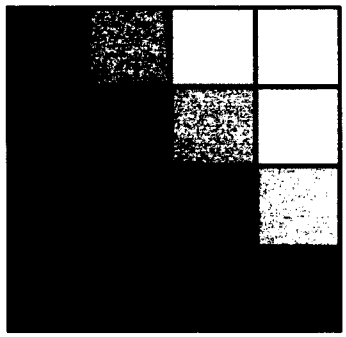


Figure 8: In this example, a two-dimensional grid is being used to track color in each grid cell. Note that because the grid is a of a low resolution, it is possible to see the discontinuity in the color gradient.

A major disadvantage of using a uniform grid for simulation is that memory is required even for those cells in the grid which are empty. As a result, significant amounts of memory may be required in order to get a simulation which captures sufficient detail and spatial extents.

In the case of rendering volumes which are represented by grids it is possible to sample the density for a given cell simply by accessing the memory representing this location. This is a major advantage as this access is typically O(1) (constant time).

2.3.2 Lagrangian / Particle Based

An alternative method for tracking the properties of some participating media throughout the spatial extent of the volume in which it resides is to consider that

participating media is composed of some number of discrete particles. Typically each particle is characterized by its position and velocity. Such approaches include vortex methods [8] [9] [10] [11] and smoothed particle hydrodynamics. [12] [13] [14] The recent popularity of using particle based methods for simulation but the contrasting dearth of particle based rendering techniques is part of our motivation for designing an efficient particle based volumetric lighting pipeline.

Particle based methods have the advantage of not needing to use computer memory in order to store information about zero density grid cells. For example, consider a large room through which a cloud of smoke is traveling. Using a uniform grid, it would be necessary to have memory tracking that there is no smoke present in the vast majority of the spatial extent. Given that computer memory is not unlimited, it is possible that a situation may arise in which it is not possible to use a fine enough uniform grid to resolve the fine details of the smoke without exhausting available memory. In contrast, a particle based system would only need to store the information about the location of the particles that are actually representing the smoke.

It should be noted, that in the case of a uniform volume of participating media it potentially would not be ideal to use a particle representation. However this scenario rarely arises in practice and in the extreme case of a fully homogenous volume, there exist more efficient for rendering such homogenous participating media data sets.

2.4 Ray Marching

Except for cases where participating media is homogeneous and has a uniform isotropic scattering function, the volume rendering function is solved using numerical integration. This integration can be performed using a technique called ray marching [15] [16] [17] which takes steps through the participating media and evaluates each segment. Each segment has a length of Δp and it is assumed that for this length the

incoming light and properties of the participating media are constant. Based upon these assumptions, the radiance for a given segment due to direct illumination can be calculated as:

$$L(p,w) = \sum_{l}^{N} L_l(p,w_l')p(p,w_l',w)\sigma_s(p)\Delta p + e^{-\sigma_t \Delta p} L(p + w\Delta p, w) \tag{21}$$

where N is the number of lights in the scene and $L_l(p,w_l')$ is the incident radiance at p arriving from direction w_l' from a light source. The first term sums the incident radiance contribution from each light in the scene for a segment and second term accounts for that radiance which is coming from the previous segments. For a participating media volume of finite size, ray marching can be used to compute the total radiance from direct illumination by recursively calling the above formula:

$$L_{n+1}(p,w) = \sum_{l}^{N} L_l(p,w_l')p(p,w_l',w)\sigma_s(p)\Delta p + e^{-\sigma_t \Delta p} L_n(p + w\Delta p, w) \tag{22}$$

Here we see the total direct illumination at $L_{n+1}(p,w)$ is the product of the sum of direct illumination from all directions and the scattering coefficient plus the attenuated total direct illumination from the previous ray marching step.

2.5 Classic Volume Rendering Techniques

2.5.1 Ray Tracing Methods

The classic method for rendering with participating media is to extend the traditional ray tracing algorithm to take into account the various absorption, emission, in- and out-scattering effects.

In the typical case of the volume being represented by a grid, ray marching is

used to calculate the attenuation of the radiance from the light passing through the volume to the scattering point as well as the attenuation from that point back to the viewing ray origin.

This ray marching consists of performing a sampling of the grid at regular intervals along the ray and performing a full attenuation evaluation to the light for each point.

This method is only computationally effective for computing single scattering. [18] That is, when light comes from the light source and undergoes only a single scattering interaction (i.e. a change in direction) and then reaches the camera by traveling along the viewing ray back to the camera.

To effectively capture the effects of multiple scattering, a more computationally efficient algorithm such as photon mapping is required [19].

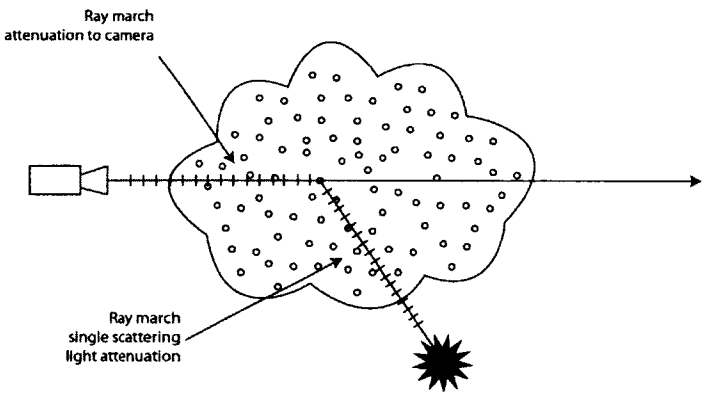


Figure 9: To solve the single scattering component of the lighting equation using ray tracing, ray marching steps are made outward from the camera along the viewing ray. At each step, the amount of light which reaches the point is evaluated using a second ray marching operation towards the light source.

2.5.2 Photon Mapping Methods

Photon mapping is a rendering technique which can be used to efficiently compute global illumination in scenes which consist of both solid surfaces and participating media. The photon mapping algorithm calculates global illumination by emitting

photons into the scene from the light sources and tracing their paths through the scene as the photons interact with the various surfaces and participating media [20]. When a photon collides with a solid surface, the bidirectional reflectance distribution function of the surface is evaluated to determine the outgoing angle and energy of the photon as it continues through the scene. In order to calculate the interaction of a photon with the participating media, the photon is ray-marched through any participating media volumes. At each step, the phase function of the participating media is evaluated to determine if the photon will scatter and change directions and the absorption cross section is used to calculate the likelihood with which the photon will be absorbed (terminating its traversal of the scene).

Photons are stored in a photon map at those points where they interact with the surfaces in the scene. Photons are also stored in a photon map when they interact with participating media. For performance reasons, those photons which represent radiance from diffuse, caustic (those photons which have interacted with reflective/refractive surfaces) and participating media sources are normally stored in their own separate maps. These photon maps are typically implemented using kd-trees, such that efficient nearest neighbor queries can be performed (these queries are used for evaluating the amount of radiance at some point within the scene at render time).

At each point where a photon interacts with the participating media, it will be either absorbed or scattered. The probability of scattering is denoted by the scattering albedo, Λ:

$$\Lambda = \frac{\sigma_s}{\sigma_t} \tag{23}$$

Where σ_s is the scattering coefficient and σ_t is the transmission coefficient. This scattering albedo can be used to scale the power of a photon after it has scattered. However, in many cases this will result in a large number of low power photons which

is computationally inefficient. Instead, Russian roulette [21] can be used to determine if a photon is scattered or absorbed. This can be accomplished by comparing Λ to a random number $\epsilon \in [0, 1]$:

$$\text{Given } \epsilon \in [0,1] = \begin{cases} \epsilon \leq \Lambda & \text{Photon is scattered} \\ \epsilon > \Lambda & \text{Photon is absorbed} \end{cases}$$

In order to evaluate the radiance at some point, on a surface or in a participating media volume, the photon maps are queried. We can estimate the reflected radiance at a surface point using the following equation where the $\frac{1}{\pi r^2}$ term is used to scale the sum of the radiance which is reflected from each photon, Φ_p, as described by the bi-directional reflectance distribution function [22] [23], f_r, at the point on the sampling disk where the photon lies.

$$L_r(x, \vec{w}) = \frac{1}{\pi r^2} \sum_{p=1}^{N} f_r(x, \vec{w_p}, \vec{w}) \Delta\Phi_p(x, \vec{w_p}) \tag{24}$$

To estimate the radiance at some point in a volume, slight modification of the surface radiance estimate equation is required to account for sampling photons within a sphere instead of a disc on a surface. Firstly, the scaling factor now corresponds to the volume of a sphere instead of the surface area of a disk. Secondly, the outgoing radiance now being calculated will be in the specific direction $\vec{w}$.

$$L_o(x, \vec{w}) = \frac{1}{\frac{4}{3}\pi r^2} \sum_{p=1}^{n} p(x, \vec{w_p}', \vec{w}) \Delta\Phi_p(x, \vec{w_p}') \tag{25}$$

Used in conjunction with ray tracing (to solve for single scattering) photon mapping is an effective technique for computing multiple scattering effects.

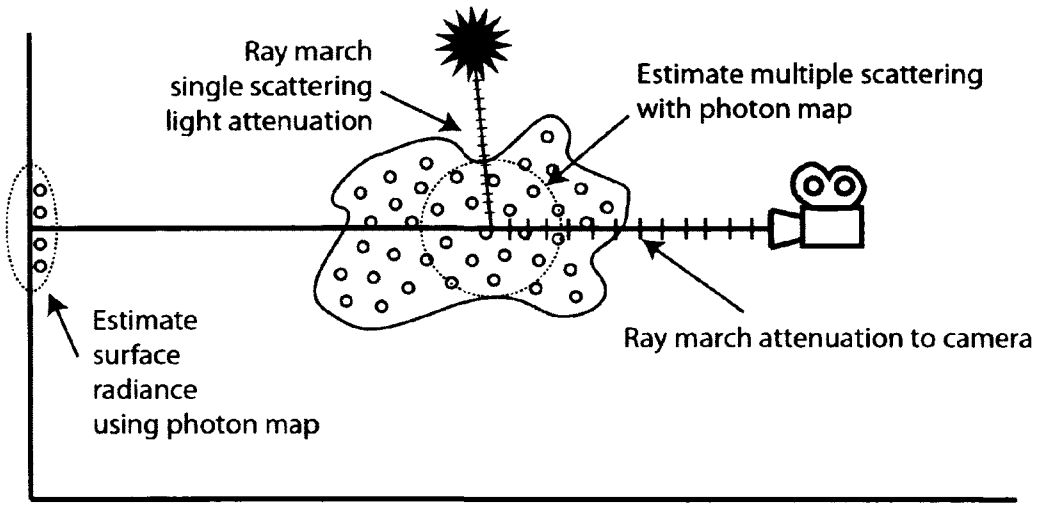


Figure 10: Solving Multiple Scattering using Photon Mapping

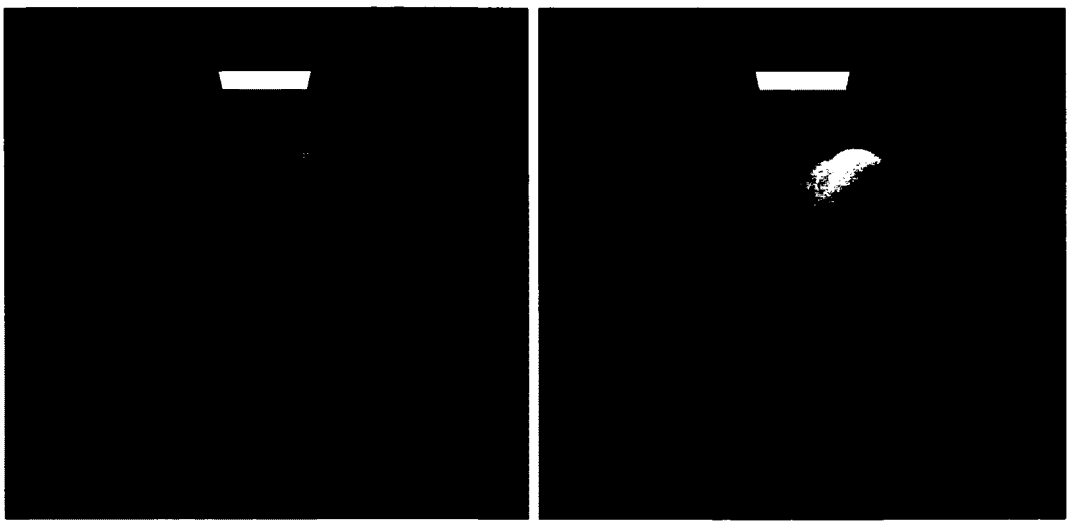


Figure 11: The left image shows the scene with only single scattering computed. The right image adds full multiple scattering effects. Images used with permission. ©Wenzel Jakob

Chapter 3

Related Work

The following chapter presents the literature which is relevant to the problem addressed by this thesis. The literature is grouped into two main sections; those dealing with methods which specifically make use of the octree data structure as an acceleration method and those which follow the pattern of the deep shadow maps technique (elaborated upon below).

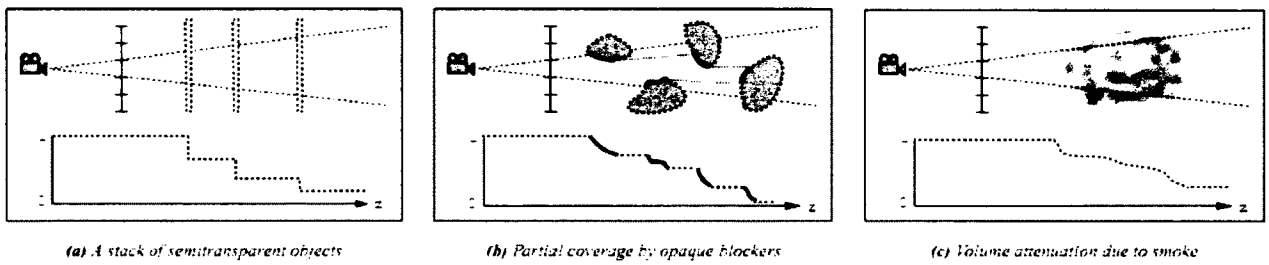
3.1 Ray Traversal of Octree Point Clouds on the GPU

One of the primary means by which the efficiency of traditional ray tracing can be improved is to use spatial subdivision to reduce the number of intersection tests between viewing rays and scene geometry. [24] Traditionally, references to the polygonal geometry which represents the scene is stored in spatial subdivision nodes. However, as the number of spatial subdivision nodes increases, the size of each spatial subdivision node will be no larger than the smallest displayable size of one pixel on the the computer screen. At this point, it is not strictly necessary to actually perform the ray-geometry intersection, as it is now sufficient to simply perform intersection testing with the subdivision nodes in order to determine the location of surfaces. This

observation is exploited by various authors for the specific purpose of computer graphics [25] [26] and as a general purpose density estimate technique. [27] Specifically, our work follows that conducted by Knoll who uses an octree to represent the vertices of solid geometry. [1] We extend his work by adding support for transparent surfaces via a general density tracking octree and implement the previously mentioned volumetric lighting calculations in order to produce a volumetric lighting particle rendering system.

3.2 Deep Shadow Maps

Deep shadow maps [28] are a technique which produces high quality shadows for volumetric media such as hair and smoke. In contrast to traditional shadow maps, which store a single depth per pixel, deep shadow maps store a representation of the transmittance through a pixel at a range of depths.



(a) A stack of semitransparent objects (b) Partial coverage by opaque blockers (c) Volume attenuation due to smoke

Figure 12: Visibility functions shown in two-dimensional cross section. Each diagram shows a beam of light that starts at the shadow camera origin (i.e. the light source) and passes through a single pixel of the deep shadow map, accompanied by that pixels visibility function. (a) The beam's power is reduced as it passes through consecutive semitransparent surfaces. (b) The blockers are opaque, but each covers only part of the pixel's area; the emphasized segments of the function correspond to visible portions of the blockers. (c) Passage through smoke reduces the beam's power in a more continuous manner. Image used with permission. ©Eric Veach and Tom Lokovic

Deep shadow maps have the following advantages over traditional shadow maps

[29]:

- Support for transparent surfaces and volumetric primitives such as smoke
- For the same quality shadows, a smaller shadow map can be used
- Support for mip-mapping [30]

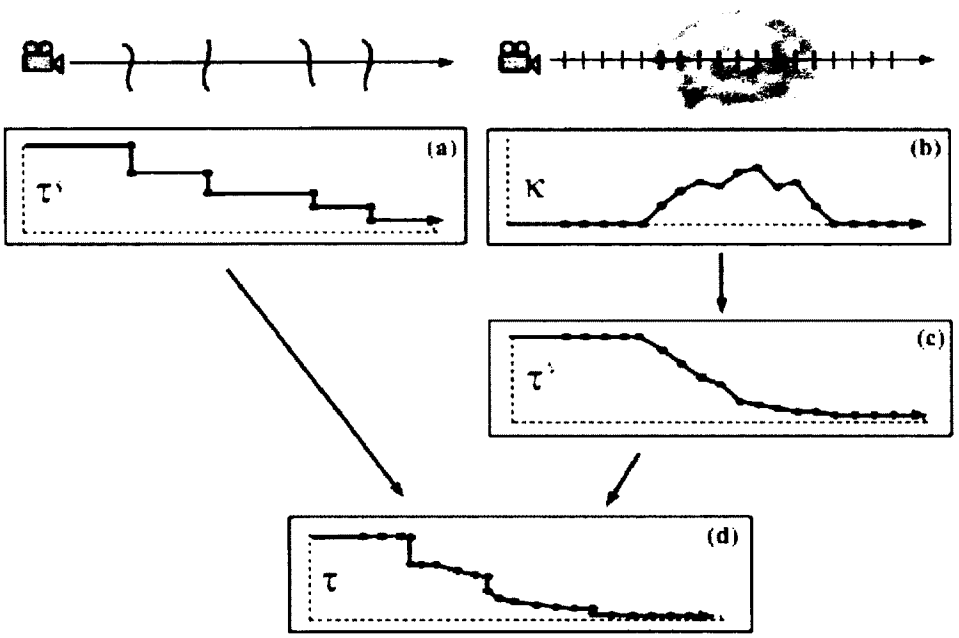


Figure 13: Constructing a transmittance function from a scene which contains solid a volumetric elements. (a) The object intersections along a given ray yield the surface transmittance function τ^s, which has a discontinuity at the depth of each surface. (b) The extinction function κ is obtained by sampling the atmospheric density at regular intervals along the ray. (c) The extinction function is integrated and exponentiated to yield the volume transmittance τ^v. (d) The surface transmittance and the volume transmittance are multiplied to obtain the final transmittance function τ for each ray. Image used with permission. ©Eric Veach and Tom Lokovic

A deep shadow map consists of a rectangular array of pixels in which every pixel stores a visibility function. Consider a ray that starts at the shadow camera origin and passes through the point (x,y) on the image plane. Some fraction of the light

emitted along this ray will be attenuated by surfaces or by volumetric scattering and absorption. The fraction of the light that penetrates to a given depth z is known as the transmittance τ(x,y,z). We refer to τ as a transmittance function when we wish to consider the transmittance at a fixed image point (x,y) as a function of z. This transmittance is typically stored in the form of a linked-list of depth/transmittance pairs per pixel.

3.3 GPU Deep Shadow Map Techniques

The following sections outline several contemporary techniques which make various improvements to the classic deep shadow maps technique and make use of GPU hardware for acceleration.

3.3.1 Opacity Shadow Maps

The deep shadow map algorithm stores a per-pixel piecewise linear approximation of the transmittance function instead of a single depth which results in a more precise shadow computation than traditional depth based shadow maps. However, this compactness and quality is achieved at the expense of significant up-front processing. In the case of animation, where lights or scene geometry updating, these maps can become costly to compute.

Opacity shadow maps [31] seek to calculate the transmittance function for any point in space using an algorithm which can be hardware accelerated. The opacity shadow map algorithm uses a set of parallel opacity maps which are oriented perpendicular to the light's direction. By approximating the transmittance function with discrete planar maps, opacity maps can be efficiently generated using graphics processing unit hardware. On each opacity map, the scene is rendered from the light's perspective, clipped by the map's depth. Instead of storing depth values, each pixel

stores Ω, the line integral of densities along the path from the light to the pixel. The opacity values from adjacent maps are sampled and interpolated, to find the amount of light reaching some point in space, during rendering.

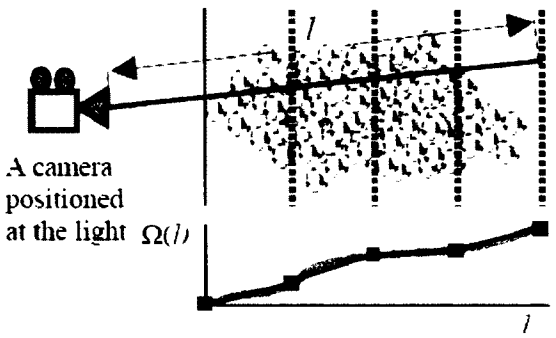


Fig. 1. The opacity function $\Omega(l)$ shown in a solid gray curve is approximated by a set of opacity maps.

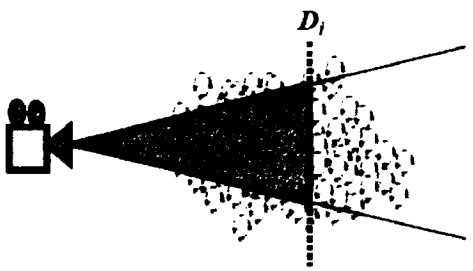


Fig. 2. The volume is rendered on each opacity map, clipped by the map's depth (D_l). The transparent region illustrates the clipped volume.

Figure 14: Opacity shadow maps. Image used with permission. ©Tae-Yong Kim

3.3.2 Deep Opacity Maps

Deep opacity maps [32] extend the opacity shadow map concept to improve visual quality while decreasing the number of mapping layers required. This is accomplished by using both traditional shadow mapping and opacity shadow maps in order to create a better distribution of opacity layers. Instead of using slices at regular intervals, depth information (from the traditional shadow map) is used to create opacity layers that vary in depth from the light source on a per-pixel basis. Since the opacity layers are warped to match the scene geometry, significantly fewer layers are needed to achieve sufficient rendering quality.

(a) Opacity Shadow Maps (b) Deep Opacity Maps

Figure 15: Opacity shadow maps use regularly space planar layers. Deep opacity maps use fewer layers, conforming to the shape of the volume. Image used with permission. ©Cem Yuskel

3.3.3 Fourier Opacity Mapping

While the deep shadow maps algorithm is a suitable off line method for calculating lighting in participating media, it can consume an unbounded amount of memory and is, in the basic form, unsuited to be implemented on graphics hardware. Fourier opacity mapping [33] is similar to opacity shadow maps and deep opacity maps in that it can be readily implemented on graphics hardware. However, instead of using discrete sampling planes (uniform or otherwise warped to scene geometry) Fourier opacity mapping stores transmittance information as a set of Fourier co-efficients which can be used to produce a smooth, continuous transmittance value along any light ray.

3.4 Ray Marching Using Sparse Block Grids

Introduced by Bridson [34], the sparse block grid is a two level data structure which balances the benefits of spatial subdivision against the costs of the potentially deep hierarchy of an octree. A first top level coarse grid is allocated to encompass the scene extents. Then, if some coarse region is necessary (because there is some data

in the region) the second level high resolution grid is allocated. In his thesis Bridson details how the sparse block grid achieves $O(n^{2.25})$ memory utilization as compared to an octree which is $O(n^2)$ and a uniform grid which is $O(n^3)$. While this is slightly higher than the octree, the sparse block grid does maintain the $O(1)$ access time charactertistic as compared to $O(logn)$ for the octree.

Thus, based upon its relatively more efficient memory utilization, the sparse block grid is a popular choice for data storage when using traditional ray marching techniques. However, although using the sparse block grid does ameliorate the memory issue associated with dense grids, traditional ray marching still has its own set of limitations. Notably, interpolations must be used at sample points and further care is required to efficiently distribute samples across the rendering volume.

Chapter 4

Particle Based Participating Media Rendering Using Density Octrees

Our goal is to create an algorithm which can render volumetric lighting effects in a time efficient, physically plausible manner. To be clear, we say those volumetric lighting effects are:

- Light attenuates as a result of interaction with inhomogeneous participating media
- Solid objects block light thereby forming volumetric shadows
- Support for an arbitrary number of lights
- Support for conventional computer graphics light types, such as point and spot lights

Our rendering system models inhomogeneous participating media using particle data sets. We elect to use particles for modeling participating media so that our simulations are spatially unbounded and grid free. By not using a grid based system, we significantly reduce our memory requirements, as compared to a uniform grid system.

The above requirements, with regard to lighting and particle data sets, present a unique rendering challenge. In order to achieve our lighting objectives, we must solve two density integrations for each particle:

1. Problem 1. Find the attenuation of the light reaching the camera from a given particle

2. Problem 2. Find the attenuation of the light reaching a particle from a given light source

We make the observation that use of GPU hardware is an excellent solution to Problem 1. Using the alpha blending functionality built into GPU hardware, we can solve the forward density integration per particle for millions of particles at interactive rates. Even including the computation required to achieve the particle depth sorting which is required to obtain correct alpha blending blending, the forward density integration process is a relatively computationally inexpensive task.

As a result of using the standard GPU pipeline for our forward density integration, it is possible to integrate other GPU based rendering techniques into our volumetric lighting and rendering system. For example, we implement a pre-existing GPU based technique [35] to include motion blur and depth of field into our system. We explain in greater detail the means by which these effects are achieved in the "Forward Density Integration using GPU Hardware" section.

The rendering process begins with the lighting computation. Presently, the lighting calculation is conducted using the CPU for ease of programmatic implementation. Once computed, the lighting data is added to the vertex buffers in memory which represent the particles. Once the lighting data is added to the vertex buffers, they are sent to the GPU for the final rendering.

We begin by explaining the lighting process and then outline the process used to create the final image using the GPU.

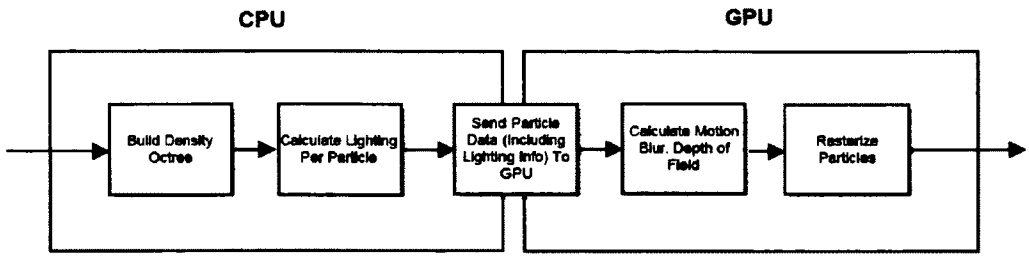


Figure 16: Particle rendering pipeline

4.1 Single Scattering Volumetric Lighting Calculation

We determine how much light reaches a particle from a light source by integrating the density along a ray constructed between the particle and the light source. Performing density integrations on particle based participating media is a challenge as there is no immediately available grid representation of the density upon which a ray marching density integration can be performed. Instead, we use a novel octree based data structure which contains the particles as our density integration acceleration structure. We refer to this data structure as the density octree.

4.1.1 Density Octree

In order to accelerate the volumetric lighting calculation, we use a novel octree construction approach in order to reduce the number of ray-particle intersection tests that are required. The density octree is a loose octree which is created using a subdivide only as needed rule. Our goal in setting up the subdivision rule is to equally balance the amount of time spent performing tree traversal bounding-box intersection and particle intersection tests. If too deep a hierarchy is created then too much time will be spent in traversal, conversely too shallow a tree results unnecessary particle

intersection tests.

Loose Octree

The density octree is created by inserting the constituent particles of our data set into an octree. In the standard manner, the particles are placed into the lowest octree node which they fit. However, in order to ensure that particles are concentrated in the leaves, we consider the particles to be a single point in space, as opposed taking their radius into account when determining which octree node they reside within. This creates a loose octree in which the stored particles actual radius may overlap the boundaries of a given octree node. We ensure this concentration of particles in the leaves for performance reasons. As it is relatively computationally cheap to traverse the tree, as compared to performing ray-particle intersection tests, we wish to minimize the number of potential ray-particle intersection tests that will be required.

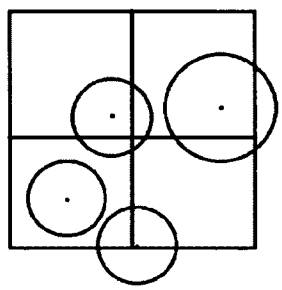


Figure 17: Loose octree allows shapes to overlap cells

Subdivide As Needed Construction

In addition to having the property of being loose, the density octree is also constructed in a subdivide only as needed manner. [36] The density octree begins as a single cell which is then subdivided only when a user selected density is reached, as measured by the number of particles residing in a given cell. Unlike a conventional grid, the

density octree has the desirable characteristic of only requiring memory in those areas of the scene where density information exists in the original particle data set.

In the case of an animated sequence, the density octree is only built once per frame in which particles or other scene objects change their position. The same density octree is used for each light. In this way, the small setup time associated with creating the density octree is amortized across the number of lighting passes required.

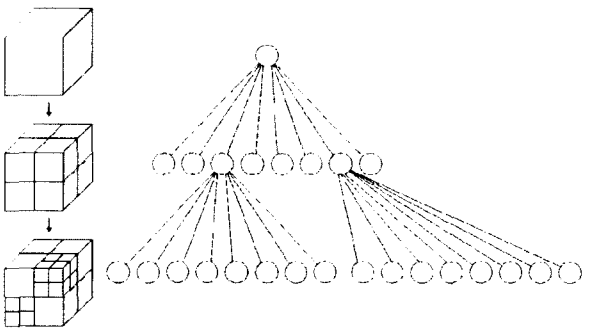


Figure 18: Octree formed through three subdivision iterations

Each density octree node keeps references to the particles that reside within the nodes spatial extent.

4.1.2 Lighting Evaluation

Integration of the density octree is performed via ray casting. A light ray is constructed between some particle and the light source that is currently being evaluated. This light ray is then intersected with the nodes of the octree in a recursive, top-down manner. This analytical ray casting approach achieves computational efficiency in that Ray-Axis Aligned Bounding Box (AABB) tests are only required at the boundaries of the density octree nodes. This is in contrast to ray marching, where it is often the case that multiple steps might be performed within a given uniform grid cell. By performing the intersection tests only as necessary, performance is increased.

At each density octree node that the light ray intersects, the light ray is then intersection tested against those particles residing in that node. When the ray intersects a particle, the density of that particle is added to the accumulated density. Finally, once the total density along the light ray has been accumulated, the total density is used to compute the amount of light reaching the particle.

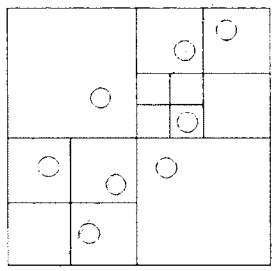


Figure 19: Example of a light ray, shown in red, intersecting a cross section of a density octree

Algorithm 1 Particle attenuation calculation pseudocode

$lightRayDirection \leftarrow lightPosition - particlePosition$
$lightRay \leftarrow Ray(particlePosition, lightRayDirection)$
$totalDensity \leftarrow RecursivelyIntersectDensityOctree(densityTree, lightRay)$
$transmittedRadiance \leftarrow e^{-totalDensity}$
$litParticleColor \leftarrow lightColor * transmittedRadiance$

4.2 Intersection Testing Routines

Strictly speaking, the particles which constitute a point based data set have no size. However, in order to translate them into volumes which can be rendered to pixel on a display, some size must be assigned. While larger sizes reduce the number of particles required to achieved coverage on-screen, they also limit the amount of detail and so some balance regarding particle size must be obtained. We create a logical mapping between the size of the particle used for rasterization and that used for ray-particle

Algorithm 2 Recursive density octree intersection routine

```
if RecursivelyIntersectDensityOctree(lightRay, thisTreeNode) then
  totalDensity ← 0
  for all childTreeNodes do
    totalDensity ← totalDensity + Intersect(lightRay, childTreeNode)
  end for
  for all containedParticles do
    totalDensity ← totalDensity + RaySphereIntersection(lightRay)
  end for
  return totalDensity
end if
return 0
```

intersection tests. We have found that a one to one mapping of particle size in relative units between rasterization size and intersection size is generally optimal. However, this relationship is easily modified and can be configured by end users.

We make the fundamental observation that, although it may be possible to create an alternative method which uses data buckets or a formula driven approach (such a Fourier analysis) to create an interpolation strategy, if exact, temporally coherent results are required then it is necessary to compute a complete and accurate lighting evaluation for each particle being rendered. Our method computes a complete lighting evaluation for each particle which results in higher quality, temporally coherent results.

We use the optimized ray-AABB intersection presented by Williams et al. [37] We use the following optimized ray-sphere intersection [2]. Our ray-sphere intersection routine calculates the distance between the closest point on the line segment and the particle center. Using this distance, we are able to apply a scaling to the particle density based upon the extent to which the light ray intersects the particle radius. By having a gradual relationship based upon distance we eliminate visual glitches that result from otherwise binary intersect/not intersect calculations.

Algorithm 3 Ray-sphere distance routine

```
diff ← sphereCenter − rayOrigin
ft ← dot(rayDirection, rayDirection)
if ft ≤ 0 then
  ft ← 0
else
  ft ← ft/squaredLength(rayDirection)
  diff ← diff − ft ∗ rayDirection
end if
return squaredLength(diff)
```

4.3 Temporal Coherence

In order for our algorithm to be used for computing volumetric lighting solutions for animations we must ensure the results that are coherent between frames. This entails that there is no flickering or visual artifacts between frames of animation. As we are computing an exact solution to the per particle level and scaling our particle density based upon the extent to which a light ray intersects a particle, temporal coherency is an inherent property of our algorithm.

4.4 Solid Geometry Volumetric Shadowing

So far our discussion has pertained primarily to the calculating the interaction of light with participating media. However, we also want to take into account the presence of solid surfaces within the scene. In order to achieve this aim, we build a bounding volume hierarchy spatial data structure on the solid geometry. Our bounding volume hierarchy implementation is based on the code provided by Shirley et al. in his ray tracing book. [38] Prior to conducting a full attenuation calculation using the density octree, the given light ray in question is tested against the solid geometry bounding volume hierarchy. If the light ray intersects a solid surfaces then no light will be reaching that point and it can be concluded that the illumination for that particle is

zero without further computation.

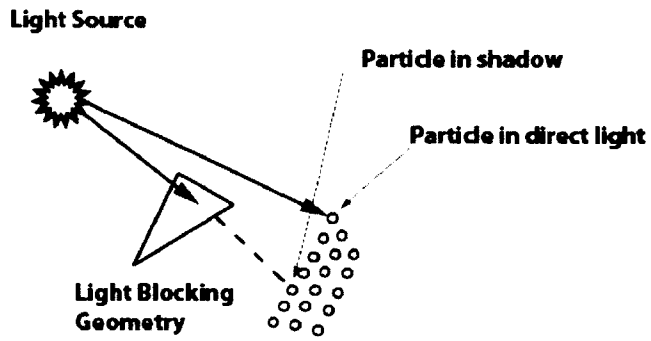


Figure 20: Solid geometry casts shadows on particles

4.5 Performance Enhancements

The primary means by which our volumetric lighting algorithm achieves its computational efficiency is the use of the density octree which ensures that only the minimum of required calculations are performed. However, we also implement our solution using some specific programmatic techniques in order to achieve further performance enhancement. These techniques are detailed in the following sections.

4.5.1 Early Termination

If the attenuation (as calculated by ray-particle intersection tests) has reduced transmission to zero, then there is no reason to continue computation. In such a scenario, the remaining ray-particle intersections can be aborted in order to save computations.

For participating media that is particularly dense, this improvement alone can significantly improve run times, as light will be attenuated to zero well before the total number of potential computations are performed.

4.5.2 Multi-threading Support

In order to take full advantage of modern multi-core processors, we implement our solution using the Intel Threading Building Blocks [39] code library. This library exposes a number of convenient programming constructs that facilitate allowing code to distribute workloads across multiple cores and processors. The primary section of our code that benefits from this optimization is the ray traversal of the density octree. Since the density octree is not modified in the course of rendering a single frame, meaning that only read operations are performed on the memory which represents the tree, it is relatively trivial to parallelize this process. We apply the threading building blocks PARALLEL_FOR construct which automatically handles the assignment of threads to each available core or processor in order to optimize throughput. In this way we are able to better utilize the available computing resources to the extent that the available memory bandwidth will allow.

4.6 Volumetric Lighting Calculation User Parameters

A range of parameters are provided which allow the user to tweak the output of our algorithm. We outline those parameters below.

4.6.1 Shadowing Density

This parameter is a scaling factor which allows the user to control the amount of light which is attenuated at each ray-particle interaction. Increasing this value results in light reaching zero intensity in fewer ray-particle interactions.

4.6.2 Solid Geometry Shadow Intensity

In the case that a particle is shadowed by solid geometry, the user can control the extent to which the particle is shadowed. This allows the user to set the shadow intensity to values greater than zero in order to achieve a more subtle shadowing effect that is closer in appearance to the results which would be achieved with a multiple scattering solution. This is the case because even in a geometry shadowed scene position, other light would be coming from multiple scattering sources resulting in the overall shadowing being greater than zero.

4.6.3 Particle Sub-Sampling

In the case of very large particle data sets, it may not be necessary to include the entire data set in the density octree. Even with some representative subset high quality volumetric lighting solutions can be achieved. To this end, we provide the particle sub-sampling parameter which allows the user to select what percentage of the total particle data set to included in the density octree.

The sub-sampling is performed by taking a uniform random distribution sampling of the input data set. Although this method technically has the deficiency of not adaptively sampling regions of greater density, we experimentally found that while it was possible to create non-uniform sampling it generally took longer to generate such a sampling than to just simply increase the number of samples in the uniform random sampling.

4.6.4 Particle Intersection Testing Size

This parameter lets the user set the size which is used in ray-particle intersection tests. By adjusting this size to be larger, it is more likely that a ray will intersect with the particles resulting in light reaching zero intensity in fewer ray-particle interactions. It

is most effectively used in conjunction with the particle skip step parameter (as detailed above) in order to adjust particle sizes such that the full volume is represented, even when the particle skip step is set such that not all particles are included in the density octree.

4.7 Forward Density Integration using GPU Hardware

In order to determine the amount of light reaching the camera along some viewing ray we must evaluate the out-going radiance from each particle along that ray taking into account the cumulative attenuation due to the proceeding particles along the viewing ray. This cumulative attenuation is calculated using the GPU hardware alpha blending functionality. In order to render correctly alpha blended particles using the GPU, we must sort the vertices that represent those particles. Sorting is performed based upon the vertices distance to the camera. Once the particles are sorted they are uploaded to the GPU in order to be rendered.

A custom geometry shader is used which expands each single vertex into a set of four vertices which represent a quad facing the camera centered around the original vertex position. By varying the size of the generated quad it is possible to have user selectable particle sizes. Furthermore, because UV coordinates for the quad are generated by geometry shader it is possible to map arbitrary particle color and opacity maps to the quad. Having adjustable particles sizes is convenient in that it allows for smoothing to be performed by using larger than a single pixel, low opacity particles.

We further detail the methods by which we produce two characteristic cinematic effects, motion blur and depth of field, which are essential for compositing particle renderings into special effects shots for film and television.

By varying characteristics such as size, opacity falloff towards edges and shape of the quads created in the geometry shader we obtain these effects, as detailed in the following sections. The ease with which our algorithm is added to existing particle rendering solutions is amongst its strengths.

Figure 21: Example of a basic scene being rendered (left), with motion blur (center) and depth of field (right).

4.7.1 Motion Blur

Motion blur results from the rapid movement of the camera or the scene being viewed such that during a single exposure multiple views of the scene occur. Using the geometry shader stage it is possible to efficiently simulate a motion blur effect. This is accomplished by modifying the shape of the quads that are produced. By creating rectangular quads of varying dimensions, where elongation correlates to greater particle speed, a motion blur effect is achieved.

4.7.2 Depth of Field

Depth of field is the distance between the nearest and furthest objects in a scene that appear acceptably in focus in a rendered image. By adjusting the depth of field it is possible to direct the viewers attention to that potion of the scene which is in focus. In a manner similar to that which is used for rendering motion blur, it is possible to use

the geometry shader to create a depth of field effect. This is accomplished by varying the size of the quad based upon its distance from the focal point. Additionally, the drop off curve which controls the particle opacity is modified to simulate the effects of depth of field.

Chapter 5

Results

We assess our new rendering algorithm using a range of quantitative and qualitative metrics. Our quantitative metrics include: processing time required to light a given number of particles for a fixed number of lights, processing time required to light a fixed number of particles for a given number of lights and the amount of processing time required to light a fixed number of particles for a fixed number of lights given an output screen resolution. The purpose of these quantitative experiments to examine the performance characteristics for the primary rendering time factors; the number of particles, the number of lights and the output resolution.

In the average case, the volmetric lighting algorithm presented with logarithmic in terms of complexity with regard to the number of particles in the data set. That is, in order to calculate the lighting for a given particle it is necessary to perform a density octree traversal which is a logarithmic time complexity operation. If multiple lights are in the scene then the lighting solution is computed sequentially for each light and the amount of illumination is aggregated. We have found that the results of our performance evaluations are congruent with the theoretical bounds of our system.

Our qualitative evaluations begin by examining the effects on the output image of varying the user parameters including the particle size and density. We then present results for a selection of data sets which represent real world usage scenarios.

As our point of comparison for both our quantitative and qualitative evaluations, we have selected Krakatoa [40]. Krakatoa is an industry standard particle rendering system produced by Thinkbox Software, which has been used in the production of numerous films and commercials. Krakatoa performs lighting calculations using a deep shadow maps based approach. The Krakatoa lighting method involves creating a two dimensional 'light plane' texture oriented down the directional axis of the given light for which the calculation is being performed. Particles are then projected to this plane and the attenuation at each pixel is updated accordingly as each subsequent write to each pixel is performed.

In each evaluation scenario where we compare to Krakatoa, the same data sets are used in both rendering systems. Krakatoa employs a sophisticated caching mechanism that prevents unnecessary re-computation of values between renderings. However, for our purposes we are attempting to compare un-cached performance. When using cached results, the computation time is essentially zero as the only work required is to read back the saved results. For this reason, our timings are taken with the caching functionality in Krakatoa disabled. All of our tests have been conducted on the same hardware consisting of a standard desktop personal computer with a single Intel i7 eight core processor running at 2.67GHz and 4GB of RAM. Although our system uses the GPU for final display, the CPU is used for the lighting calculation, as does Krakatoa. Therefore, the GPU is not relevant for the performance evaluations we are conducting.

5.1 Performance

Our first performance evaluation explores raw particle throughput in terms of how much time is required to light a given number of particles. We observe that our system is characterized by a logarithmic time increase with respect to the number

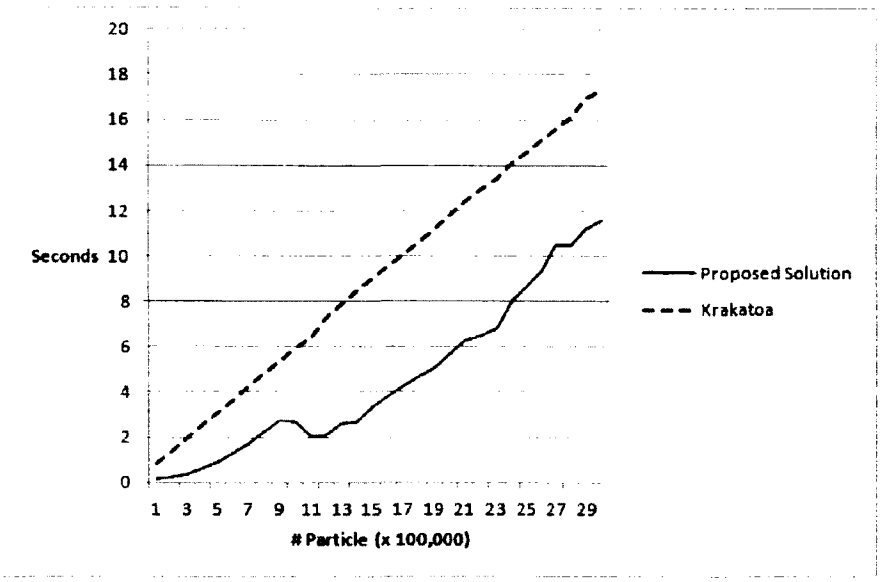


Figure 22: Time required to calculate a lighting solution for a single light given some number of particles.

of particles. This is expected, as our density evaluation are performed on an octree based data structure. Krakatoa exhibits linear behavior as a result of their light pass algorithm. However, in this naive case we are using the complete set of particles in our density tree. Recall that our system is capable of, and only requires, a subset of the input particles to be stored in the density tree in order to get good lighting results. When we automatically adjust the number of particles used in the density evaluation tree such that we only use enough to achieve a visually acceptable lighting result, instead of over defining the density evaluation, our system can also be made to achieve near linear performance. We consider this ability to have automatic or user control over lighting solution qualities to be a strength of our system.

In this performance evaluation we explore the relationship between the time required to achieve a lighting solution and the number of lights in the scene give a fixed number of particles. Both our system and Krakatoa exhibit a linear performance relationship between the number of lights present in the scene and the time required to compute a lighting solution. Although the constant time factor of our

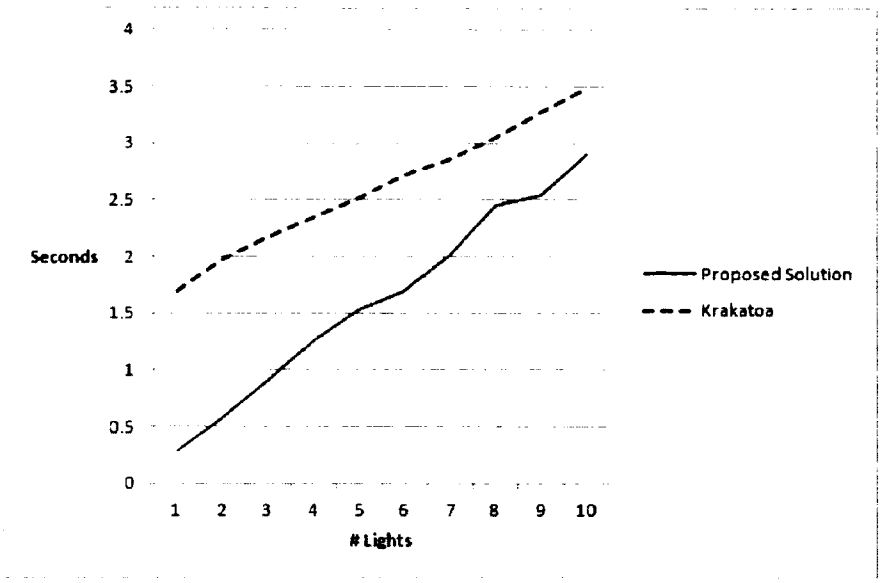


Figure 23: Time required to calculate a lighting solution for a given number of lights on a scene of 250,000 particles.

system is slightly higher, resulting in a slightly greater time increase per light, both systems have the same performance characteristic with regard to increasing number of lights. We attribute our higher constant time factor to differences in programmatic implementation rather than any deficiency in the base algorithms.

Our volumetric lighting, motion blur and depth of field calculations are independent of output image size, as shown in Figure 24. This is because all our lighting computation is performed using the output resolution independent density octree. As such, no additional computation time is required for higher resolution output. In contrast, we observe that Krakatoa appears to exhibit a quadratic increase in time required to render as output frame size increases. Given that the performance of our algorithm is independent of resolution, it is a good choice in situations where modern high resolution frames are required.

Adjusting the number of particles per density tree cell influences render time. We have experimentally determined that for every 1,000,000 particles in the scene a corresponding coarseness of 50 particles per cell is a good rule of thumb, as shown in

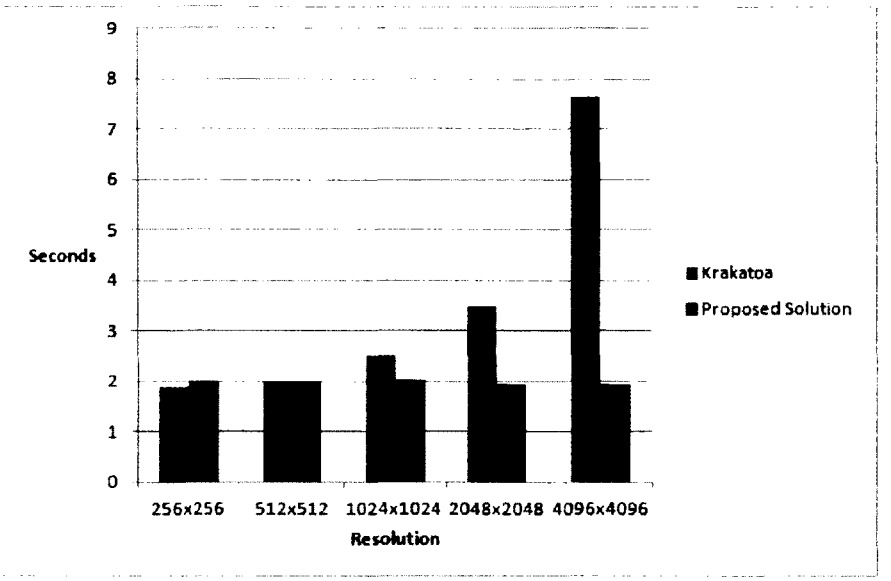


Figure 24: Time required to render at a given output frame size for a scene consisting of 250,000 particles and one light.

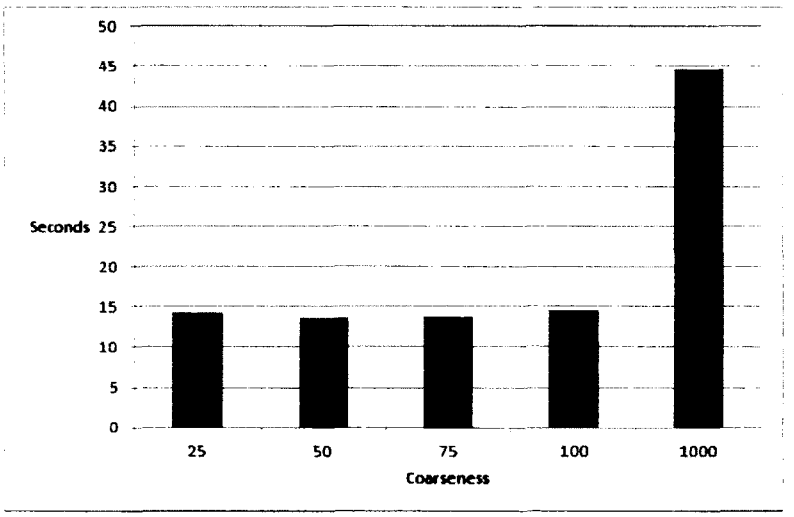


Figure 25: Amount of time required to compute lighting for a single light in a scene of 1,000,000 particles for various density tree coarseness settings.

Figure 25.

5.2 Visual Quality

Although raw performance statistics are an important metric of the utility of a rendering system, ultimately the final criterion is the quality of the output imagery. To this end, we examine several real world scenes and compare the output of our system against Krakatoa. Additionally, we present output results for a range of user configurations using our system.

This smoke plume data set consists of 350,000 particles. Note how Krakatoa is unable to produce a smooth lighting solution with the number of particles provided. In contrast, our system, which calculates the lighting at each particle and is able to scale particles sizes while maintaining a correct lighting solution, is able to produce a smooth lighting solution. Using our algorithm, it is possible to achieve higher quality results using less particles. In practice, requiring less particles to achieve acceptable visual quality means that our system out performs Krakatoa on typical rendering tasks.

Using our lighting algorithm, we are able to assign the lighting data to particles independent of their scale. Strictly speaking, as the particle size is increased the lighting data at the point center is less valid towards the outer edges of the point sprite however we have found that in practice the data extrapolates quite well. As the particle size evaluation shows, our system is able to provide lighting data for a range particle appearances ranging from very detailed, gritty sand type particles to large cloud type particles. In each case the rendering time remains the same as our lighting is decoupled from the output particle rendering, as shown in Figure 28 and Figure 29.

The density scale parameter allows the user to adjust the overall particle density

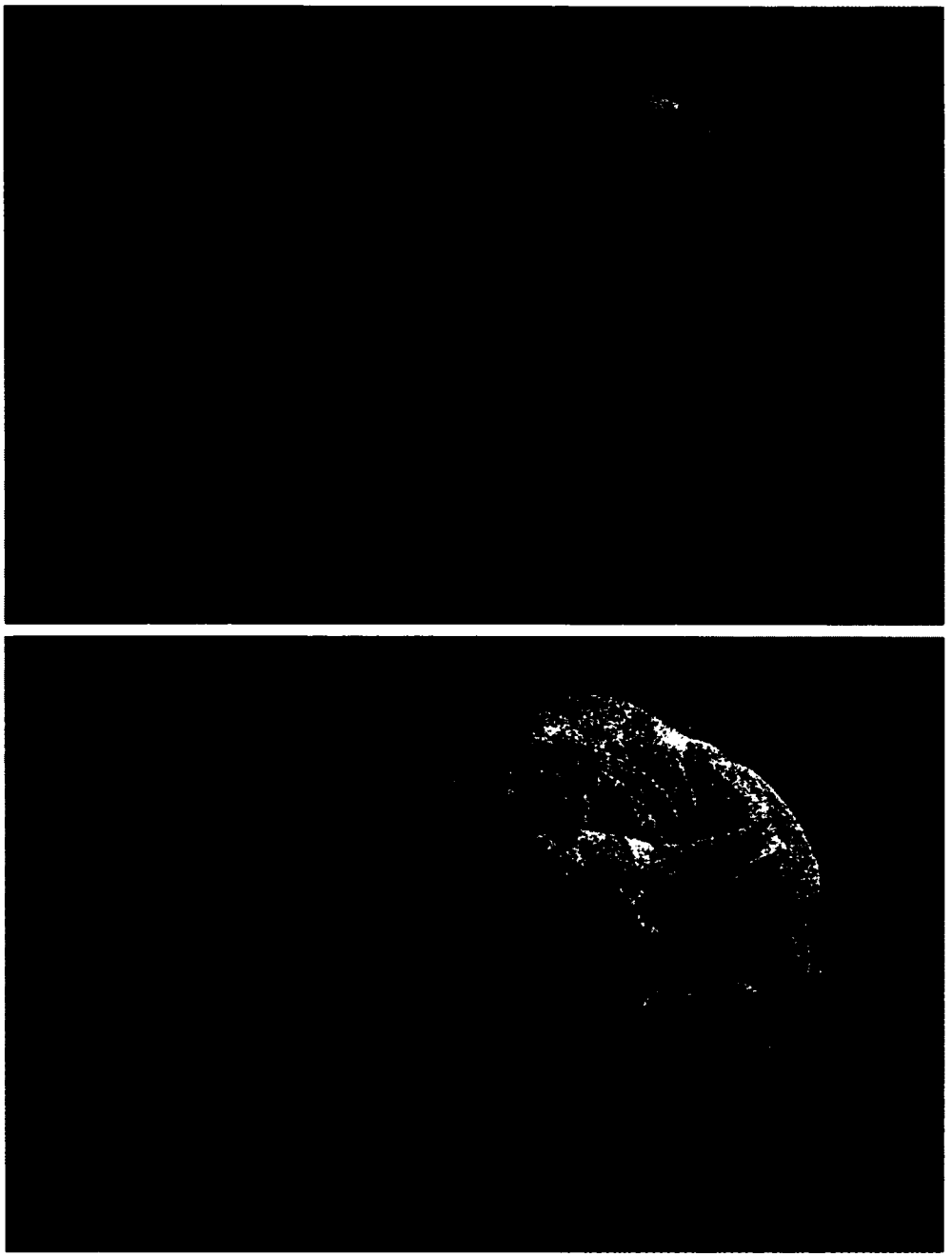


Figure 26: Smoke plume data set (350,000 particles) rendered with our proposed solution (top) and Krakatoa (bottom).

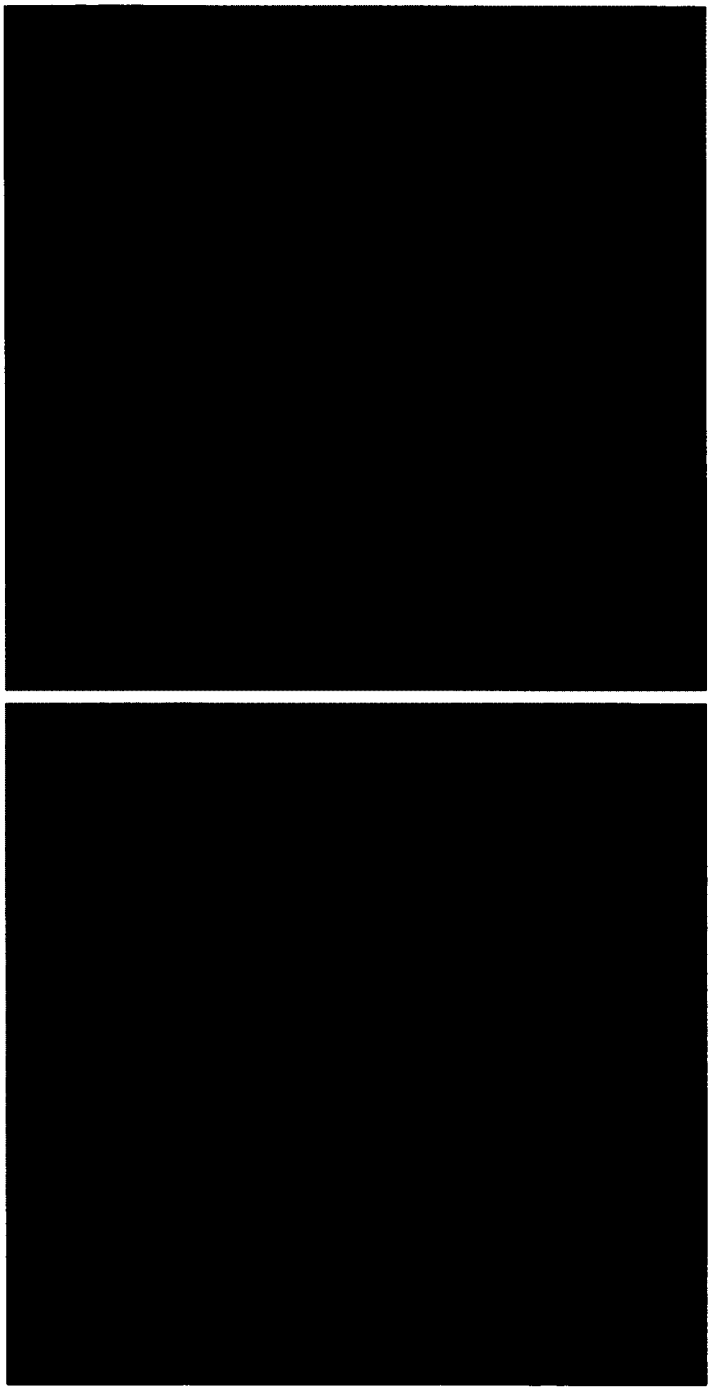


Figure 27: Perlin cloud data set (9,000,000 particles) rendered with our proposed solution (top) and Krakatoa (bottom). Renders took 7.1 and 14.6 seconds, respectively.

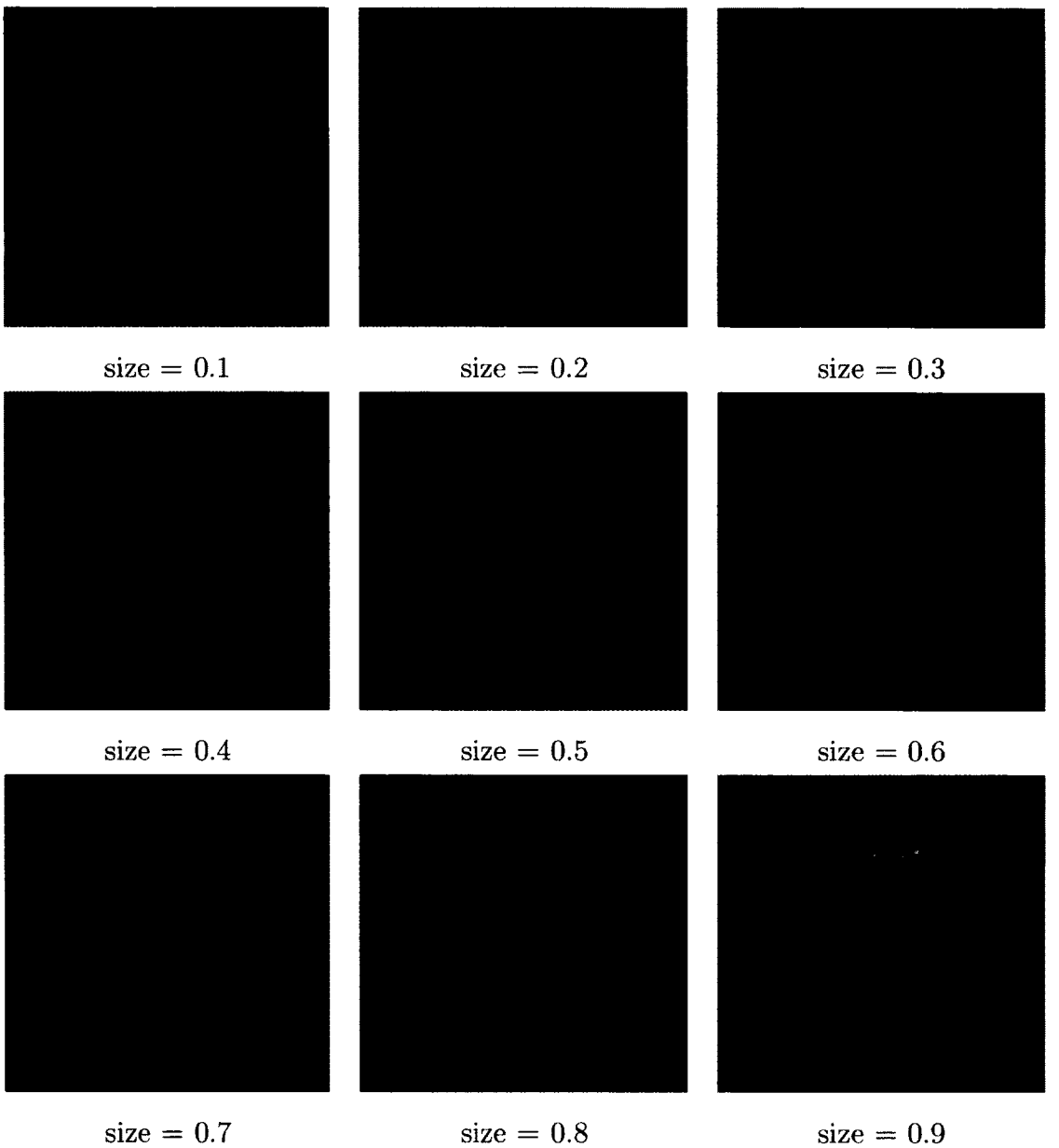


Figure 28: Effects of increasing output particle size on the same lighting solution. In all cases, render time remains constant.

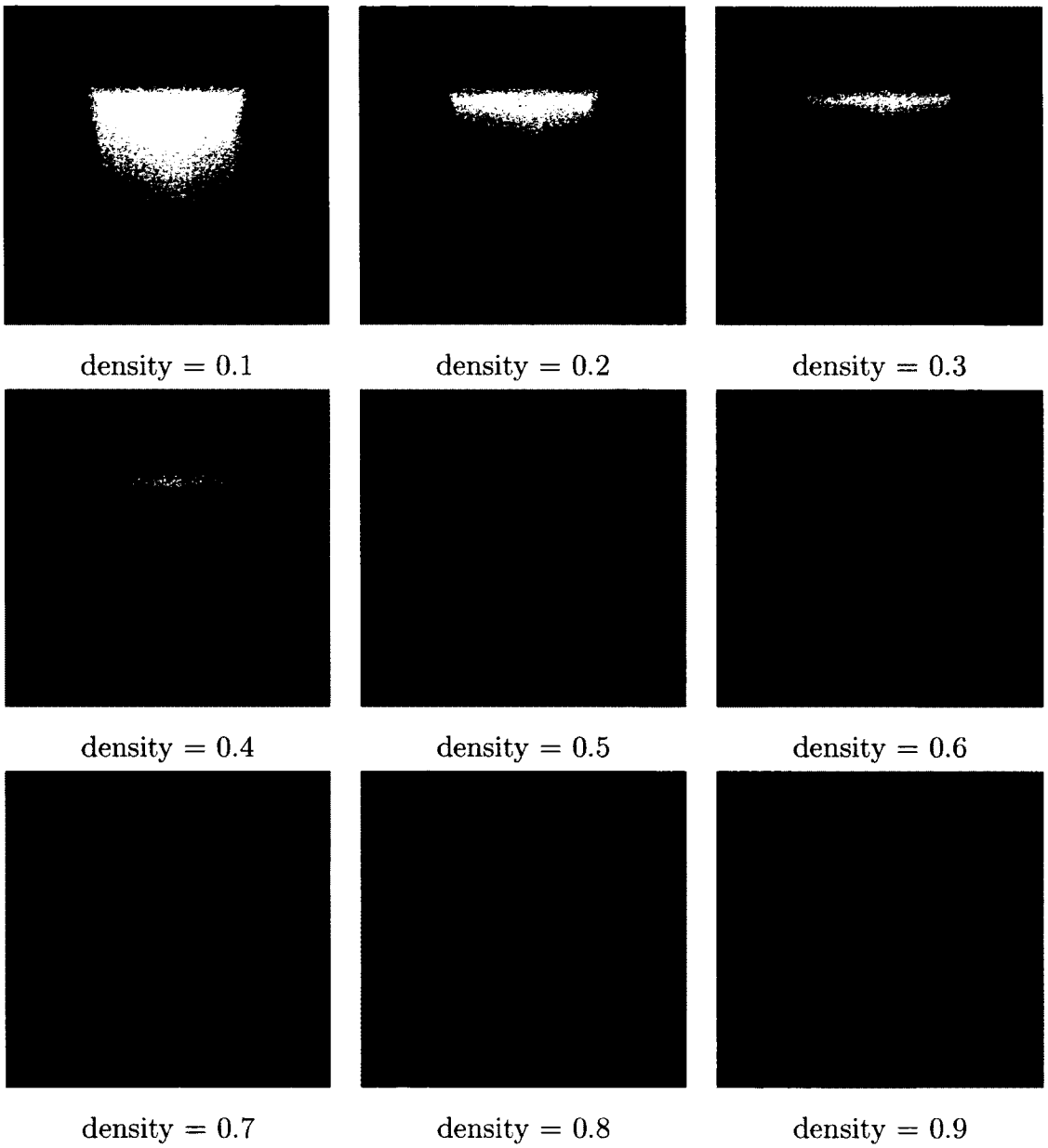


Figure 29: Effects of increasing the particle density. In all cases, render time remains constant.

of a data set. As with particle size, this parameter can be adjusted without affecting the render time, apart from the cases in which modifying density changes the number of particles for which early termination results in a change in calculation time.

In Figure 30, we show the effects of subsampling on the output of our system and Krakatoa. Even with only 10% of the original data set being used in the density octree our system still provides meaningufl, albeit coarse information about the lighting in the scene. Krakatoa on the other hand fails to provide much meaninful lighting data once the input data set is reduced.

Our lighting algorithm integrates seamlessly with the rendering of solid scene geometry. This is accomplished by calculating the intersection of light rays with the scene geometry. Doing so results in the calculation of volumetric shadows, all with no scene modification or special processing required, as shown in Figure 32.

Additionally, our system is able to calculate the shadows that would be cast by participating media onto arbitrary scene geometry, as shown in Figure 33. The shadow map is output as a separate floating point image buffer which can be composited into original frame buffer using standard image compositing tools.

5.3 Discussion

As the results show, our rendering system, based around our novel particle based participating lighting algorithm, is able to achieve high quality output in an efficient and time competitive manner as compared the current industry standard particle rendering tool. As explained in the introduction of this section, in the default configuration our system is logarithmic in time complexity with regard to the number of particles. However, based upon our observation that it is unnecessary to use the entire data set in creating the density octree we can continuously reduce the amount of particles included in a subsampled set such that our run time can be adjusted so that it is

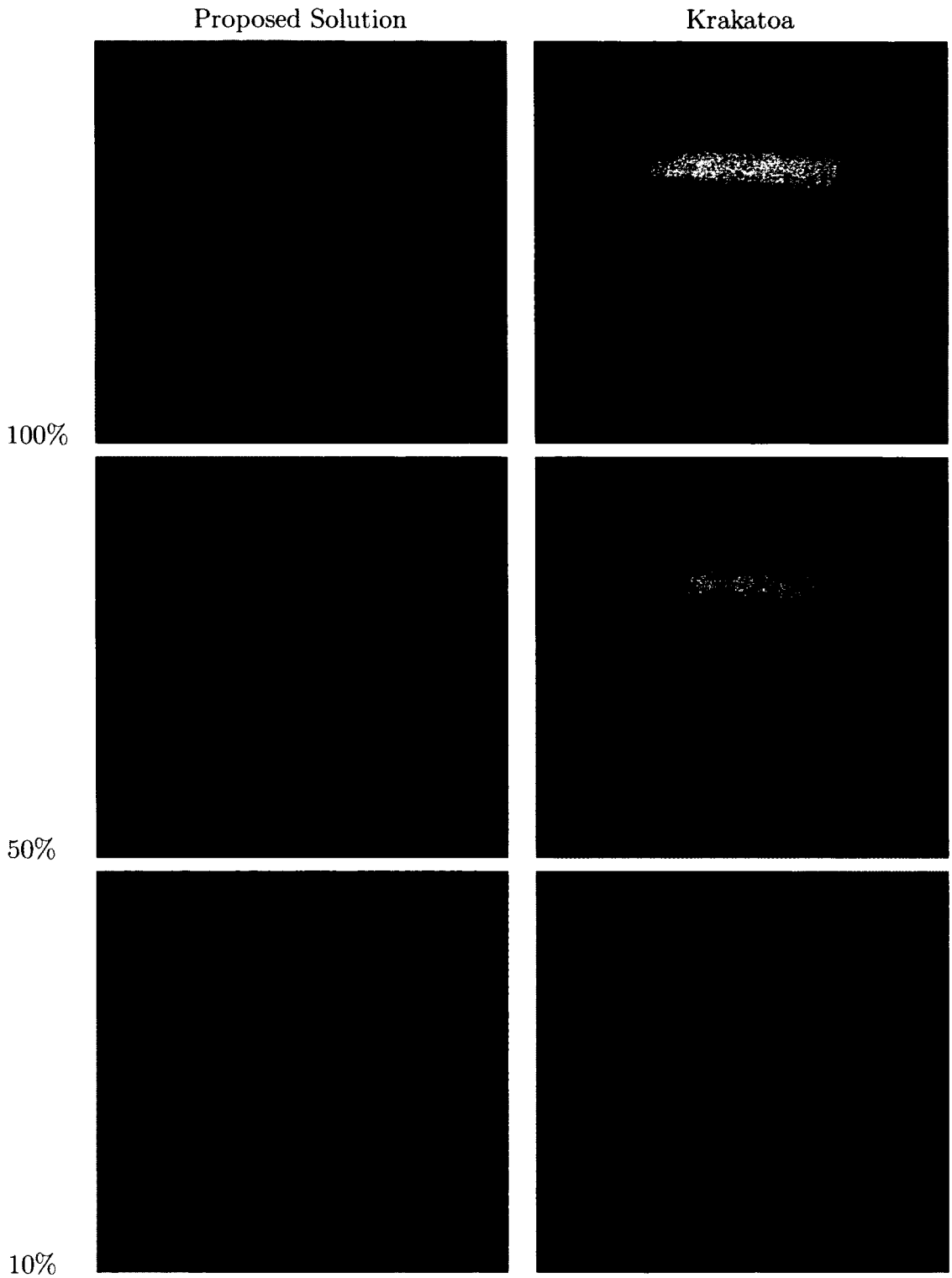


Figure 30: Effects of changing percentage of total particles included in the density tree (3,000,000 particles data set)

	Proposed Solution	Krakatoa
100%	10.4s	17.8s
50%	5.5s	9.0s
10%	3.5s	2.2s

Figure 31: Performance results for various percentage of total particles included in density tree. Krakatoa numbers were obtained by reducing the number of particles in the source data set.

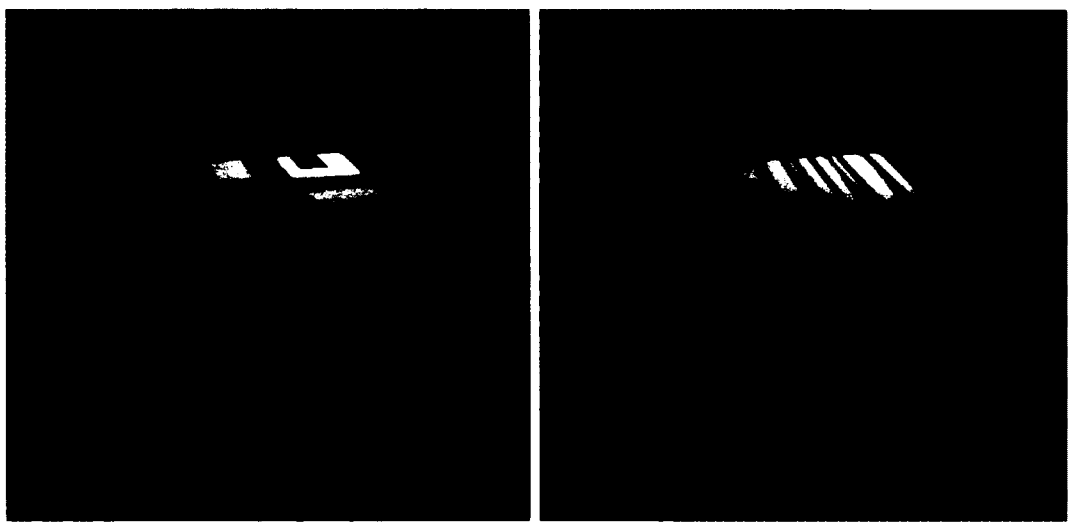


Figure 32: Solid scene geometry casting shadows onto participating media.

Figure 33: Our system is able to calculate the shadows that volumetric media would cast onto scene geometry.

closer to linear in characteristic.

In our qualitative evaluations, we found that our system produces higher quality outputs with less particles than Krakatoa. As a result of requiring less particles for the same output quality, our system achieves comparable or better performance in practical usage scenarios. Requiring less input particles is a major advantage in that it reduces the amount of disk space required to store the data sets. Perhaps even more importantly, because less particles are required for rendering, it is not necessary to simulate as many particles at the outset of the production process. As this simulation step is routinely amongst the most expensive in terms of computation time, our system with its lower particle requirements, is an attractive option for production rendering pipelines.

Furthermore, our system seamlessly integrates the calculation of volumetric particle shadows and the shadows that are cast by participating media onto any arbitrary scene geometry, as shown in Figure 33. As our lighting calculation is independent of

output resolution, there is no additional expense for higher resolution output images. The user parameters of system which control the particle size and density can also be adjusted without affecting computation time.

Based upon the above results, we believe that we have achieved our objectives to produce a computationally efficient particle based participating rendering system.

Chapter 6

Conclusion

We have demonstrated a novel technique for computing volumetric lighting on arbitrary particle data sets. We achieve the goal of computing volumetric lighting solutions by using a density octree, which allows for evaluating the density of the participating media represented by a particle data set. Given this computationally and memory efficient density representation, we can compute the amount of light which reaches each particle in the scene. Our new algorithm exhibits linear time complexity with respect to the number of lights and logarithmic with respect to the number of particles in the data set. This performance is achieved through the use of the density octree data structure presented in this thesis. Using this new technique it is possible to achieve high quality computer generated imagery of various participating phenomena. As shown by the integration of existing motion blur and depth of field techniques, our new volumetric lighting algorithm is suitable for being included into existing particle rendering systems. This ease of integration into existing systems positions our new technique well for implementation into existing visual effects pipelines. In fact, it has already been included in the Exocortex Fury particle rendering software.

Chapter 7

Future Work

The following sections present a selection of areas which we identify as future avenues of development to continue improving our existing system.

7.1 Level of Detail

In regions further from the camera, it is not necessary that the lighting calculations be as accurate since the viewers will not be able to as readily see detail in those areas. Using our algorithm it would be possible to automatically adjust the level of detail by making the percentage of total particles used in ray-particle checks a function of the distance to the camera. For example, consider some density octree node located near the camera which contains one hundred particles. In this case, since the density octree node is located close to the camera we would want the most accurate lighting calculation possible, so all one hundred particles would be ray-sphere tested. However, consider some density octree node which is a significant distance away from the camera. In this case, we might only perform ray-sphere tests for a representative sub-portion of the particles in the density octree node. In this way, full computation would only performed in those regions which it is required.

7.2 Multiple Scattering

At present, our system only calculates the single scattering component of volumetric lighting. A further extension would be to develop a method by which it would be possible to efficiently utilize the spatial density information provided by the density octree in order efficiently calculate the multiple scattering component of volumetric lighting.

7.3 Ambient Occlusion

Given the spatial locality information provided by the density octree it should be possible to create a fast ambient occlusion calculation process. This would be useful for creating various artistic effects as well as fast approximations of multiple scattering when a fully calculated result is not required.

7.4 Particle Replication

One of the issues when working with particle based simulations is that it often takes more time to calculate the particle dyanmics than it does to render those particles. As a result, rendering systems can be under utilized because it is not possible to simulate enough particles to fully load the particle rendering systems. In order to alleviate this problem, it would desirable that a particle rendering system could take a base set of particles which represent the structure, movement and other characteristics of the original simulation and then add additional particles at render time which further enhance the final output.

List of References

[1] A. Knoll. "Ray traversal of octree point clouds on the gpu." In "Interactive Ray Tracing, 2008. RT 2008. IEEE Symposium on," page 182 (2008).

[2] T. Akenine-Möller, E. Haines, and N. Hoffman. *Real-Time Rendering 3rd Edition.* A. K. Peters, Ltd., Natick, MA, USA. ISBN 987-1-56881-424-7 (2008).

[3] M. Pharr and G. Humphreys. *Physically Based Rendering: From Theory to Implementation (The Interactive 3d Technology Series).* Morgan Kaufmann. ISBN 012553180X (2004).

[4] S. Chandrasekhar. *Radiative Transfer.* New York: Dover Publications. Originally published by Oxford University Press, 1950. (1960).

[5] J. Arvo. *Transfer Equations in Global Illumination* (1993).

[6] J. Stam. "Real-time fluid dynamics for games." (2003).

[7] R. Bridson. *Fluid simulation for computer graphics.* A K Peters, Wellesley, Mass. ISBN 9781568813264 (2008).

[8] L. Yaeger, C. Upson, and R. Myers. "Combining physical and visual simulation—creation of the planet jupiter for the film 2010." In "Annual Conference on Computer Graphics," volume 20, pages 85–93 (1986).

[9] M. N. Gamito, P. F. Lopes, and M. R. Gomes. "Two-dimensional simulation of gaseous phenomena using vortex particles." In "In Proceedings of the 6th Eurographics Workshop on Computer Animation and Simulation," pages 3–15. Springer-Verlag (1995).

[10] A. Angelidis and F. Neyret. "Simulation of smoke based on vortex filament primitives." In "Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation," SCA '05, pages 87–96. ACM, New York, NY, USA. ISBN 1-59593-198-8 (2005).

[11] S. I. Park and M. J. Kim. "Vortex fluid for gaseous phenomena." In "Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation," SCA '05, pages 261–270. ACM, New York, NY, USA. ISBN 1-59593-198-8 (2005).

[12] M. Desbrun and M. paule Gascuel. "Smoothed particles: A new paradigm for animating highly deformable bodies." In "In Computer Animation and Simulation 96 (Proceedings of EG Workshop on Animation and Simulation," pages 61–76. Springer-Verlag (1996).

[13] M. Mller, D. Charypar, and M. Gross. "Particle-based fluid simulation for interactive applications." (2003).

[14] S. Premoze, T. Tasdizen, J. Bigler, A. Lefohn, and R. T. Whitaker. "Particle-based simulation of fluids." (2003).

[15] M. Levoy. "Efficient ray tracing of volume data." *ACM Trans. Graph.* **9**, 245–261. ISSN 0730-0301 (1990).

[16] K. Perlin and E. M. Hoffert. "Hypertexture." *SIGGRAPH Comput. Graph.* **23**, 253–262. ISSN 0097-8930 (1989).

[17] D. S. Ebert, F. K. Musgrave, D. Peachey, K. Perlin, and S. Worley. *Texturing and Modeling: A Procedural Approach.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 3rd edition. ISBN 1558608486 (2002).

[18] H. W. Jensen and P. H. Christensen. "Efficient simulation of light transport in scences with participating media using photon maps." In "Proceedings of the 25th annual conference on Computer graphics and interactive techniques," SIGGRAPH '98, pages 311–320. ACM, New York, NY, USA. ISBN 0-89791-999-8 (1998).

[19] H. W. Jensen. *Realistic image synthesis using photon mapping.* A. K. Peters, Ltd., Natick, MA, USA. ISBN 1-56881-147-0 (2001).

[20] H. W. Jensen. "Global illumination using photon maps." In "Proceedings of the eurographics workshop on Rendering techniques '96," pages 21–30. Springer-Verlag, London, UK. ISBN 3-211-82883-4 (1996).

[21] J. Arvo and D. Kirk. "Particle transport and image synthesis." *SIGGRAPH Comput. Graph.* **24**, 63–66. ISSN 0097-8930 (1990).

[22] J. F. Blinn. "Models of light reflection for computer synthesized pictures." *SIGGRAPH Comput. Graph.* **11**, 192–198. ISSN 0097-8930 (1977).

[23] G. J. Ward. "Measuring and modeling anisotropic reflection." *SIGGRAPH Comput. Graph.* **26**, 265–272. ISSN 0097-8930 (1992).

[24] A. S. Glassner. *Space subdivision for fast ray tracing*, pages 160–167. Computer Science Press, Inc., New York, NY, USA. ISBN 0-8186-8854-4 (1988).

[25] J. Gu and S. Wei. "An octree ray casting algorithm based on multi-core cpus." In "Computer Science and Computational Technology, 2008. ISCSCT '08. International Symposium on," volume 2, pages 783 –787 (2008).

[26] W. Song, S. Hua, Z. Ou, H. An, and K. Song. "Octree based representation and volume rendering of three-dimensional medical data sets." In "BioMedical Engineering and Informatics, 2008. BMEI 2008. International Conference on," volume 1, pages 316 –320 (2008).

[27] H. Eberhardt, V. Klumpp, and U. Hanebeck. "Density trees for efficient nonlinear state estimation." In "Information Fusion (FUSION), 2010 13th Conference on," pages 1 –8 (2010).

[28] T. Lokovic and E. Veach. "Deep shadow maps." In "Proceedings of the 27th annual conference on Computer graphics and interactive techniques," SIGGRAPH '00, pages 385–392. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA. ISBN 1-58113-208-5 (2000).

[29] L. Williams. "Casting curved shadows on curved surfaces." *SIGGRAPH Comput. Graph.* **12**, 270–274. ISSN 0097-8930 (1978).

[30] L. Williams. "Pyramidal parametrics." In "Computer Graphics (SIGGRAPH 83 Proceedings," (1983).

[31] T.-Y. Kim and U. Neumann. "Opacity shadow maps." In "In Proceedings of the 12th Eurographics Workshop on Rendering Techniques," pages 177–182. Springer-Verlag (2001).

[32] C. Yuksel and J. Keyser. "Deep opacity maps." *Computer Graphics Forum (Proceedings of EUROGRAPHICS 2008)* **27**(2) (2008).

[33] J. Jansen and L. Bavoil. "Fourier opacity mapping." In "Proceedings of the 2010 ACM SIGGRAPH symposium on Interactive 3D Graphics and Games," I3D '10, pages 165–172. ACM, New York, NY, USA. ISBN 978-1-60558-939-8 (2010).

[34] R. Bridson, S. U. P. in Scientific Computing, and C. Mathematics. *Computational aspects of dynamic surfaces.* Stanford University (2003).

[35] H. E. Madill, J. and B. Houston. "Fury renderer." Exocortex Technologies, Inc. Ottawa, Canada. (2010).

[36] J. Wilhelms and A. Van Gelder. "Octrees for faster isosurface generation." *ACM Trans. Graph.* **11**, 201–227. ISSN 0730-0301 (1992).

[37] A. Williams, S. Barrus, R. K. Morley, and P. Shirley. "An efficient and robust ray-box intersection algorithm." *journal of graphics, gpu, and game tools* **10**(1), 49–54 (2005).

[38] P. Shirley and R. Morley. *Realistic ray tracing.* Ak Peters Series. AK Peters. ISBN 9781568811987 (2003).

[39] J. Reinders. *Intel threading building blocks.* O'Reilly & Associates, Inc., Sebastopol, CA, USA, first edition. ISBN 9780596514808 (2007).

[40] *Thinkbox Software Krakatoa http://www.thinkboxsoftware.com/krakatoa/.*