Hybrid Rasterization / Ray Casting for Volumetric Lighting

Richard Monette

June 1, 2011

Contents

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. Esthetically pleasing treatment of these phenomena is critical to achieving high quality computer graphics imagery. This thesis presents a high quality, 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. Our technique is capable of computing volumetric lighting for particle data sets consisting of millions of particle for an arbitrary number of lights of various types and exhibits linear algorithmic complexity.

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 the relevant research is provided in Chapter 3. We then present a detailed definition of the problem which our technique addresses, followed by the implementation details, results and conclusion.

2 Theoretical Overview

In general, computer graphics solutions make 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[1]) 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 and smoke or atmospheric effects.

To capture such effects, 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 part 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.

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.

2.1 Participating Media Interaction Processes

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 density 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. Typically, this value also varies with respect to the frequency of the light (i.e. higher wavelengths interact more with the participating media resulting in a quicker absorption).

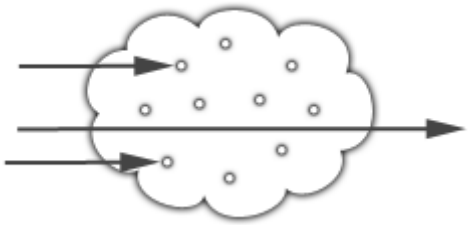


Figure 1: Incident radiance is reduced due to absorption resulting in less outgoing radiance.

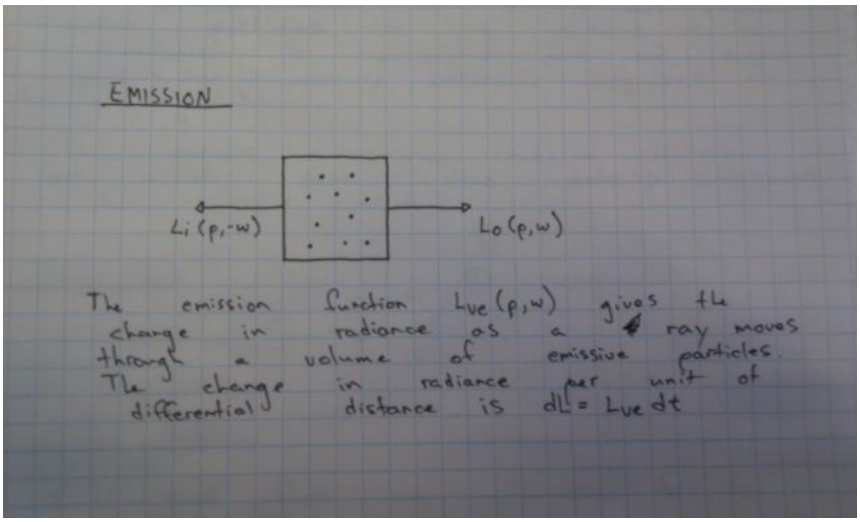
Given some incident radiance $L_i(p, \omega)$, at position p traveling along vector ω, we find the change in outgoing radiance $L_o(p, \omega)$ after the absorption interaction with the participating media using the following equation:

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

2.1.2 Emission

Emission increases the amount of radiance traveling along some path through a participating media 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.



2.1.3 Out-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 to the out-scattering of light energy. However, it is also possible that light will enter the medium and be scattered such that it is added to that which reaches the virtual camera. This outcome, where light is added, is referred to as in-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:

$$dL_o(p,\omega) = -\sigma_s(p,\omega)L_i(p,-\omega)dt \tag{2}$$

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:

$$\sigma_t(p,\omega) = \sigma_a(p,\omega) + \sigma_{p,\omega}(p,\omega) \tag{3}$$

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

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

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

$$T_r(p \to p') = e^{-\int_d^0 \sigma_t(p+t\omega,\omega)dt} \tag{5}$$

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. Thus, if exitant radiance from a point p on a surface in a given direction σ is given by $L_0(p,\sigma)$, after accounting for extinction, the incident radiance at another point p' in direction $-\sigma$ is

$$T_r(p \to p')L_o(p,\sigma) \tag{6}$$

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

$$T_r(p \to p'') = T_r(p \to p')T_r(p' \to p''), \tag{7}$$

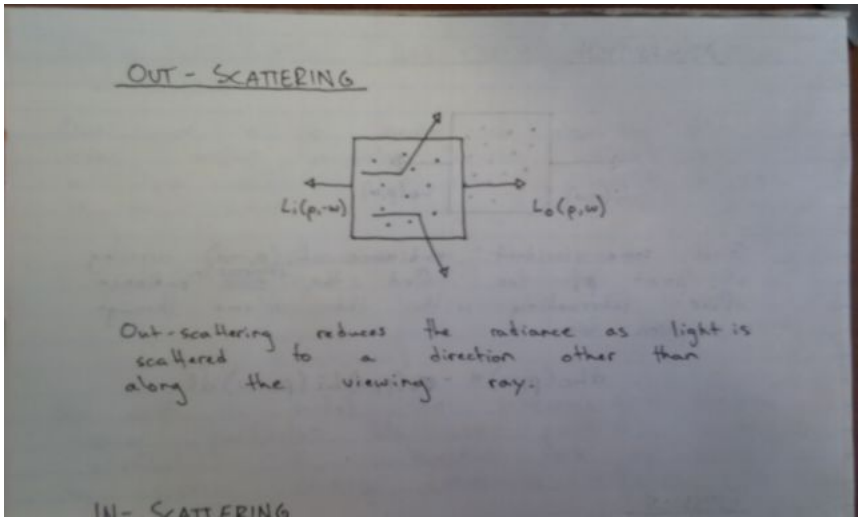
for all points p' between p and p''. This property is important for volume scattering implementations, since it makes it possible to incrementally compute transmittance at many points along a ray by computing the product of each previously computed transmittance with the transmittance for its next segment.

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

$$\tau(p \to p') = \int_0^d \sigma_t(p + t\omega, -\omega)dt \tag{8}$$

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

$$T_r(p \to p') = e^{-\sigma_t d} \tag{9}$$



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.

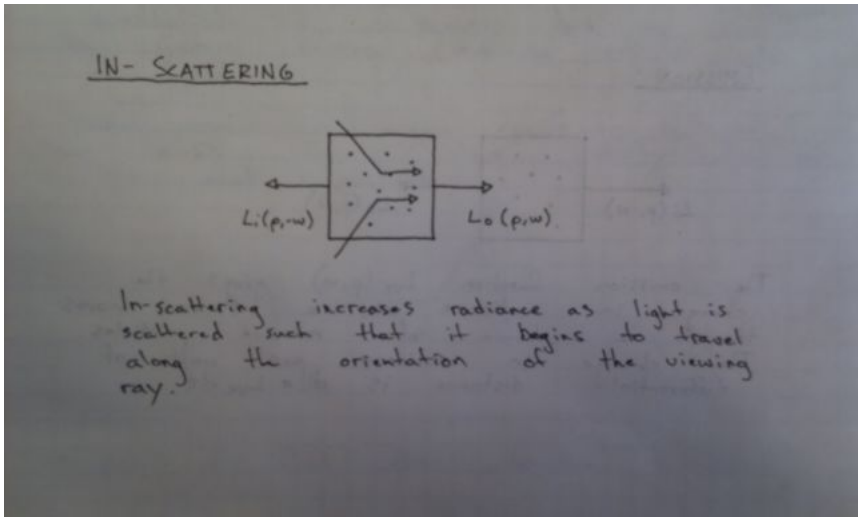
The total radiance added per unit distance due to in-scattering is given by the source term S:

$$dL_o = S(p, \omega)dt \tag{10}$$

which accounts for the increase in radiance from both emission and in-scattering:

$$S(p, \omega) = L_{ve}(p, \omega) + \sigma_s(p, \omega) \int_{S^2} p(p, -\omega' \to \omega) L_i(p, \omega')d\omega' \tag{11}$$

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.



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.[2]

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:

$$S(p,\omega) = L_{ve}(p,\omega) + \sigma_t(p,\omega)L_i(p,-\omega)dt \tag{12}$$

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

The attenuation coefficient, $\sigma_t(p,\omega)$, accounts for all processes that reduce radiance at a point: absorption and out-scattering. The differential equation that describes this effect is

$$dL_o(p,\omega) = -\sigma_t(p,\omega)L_i(p,-\omega)dt \tag{13}$$

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):

$$\frac{\theta}{\theta t} L_o(p, \sigma) = -\sigma_t(p, \sigma) L_i(p, -\sigma) + S(p, \sigma) \tag{14}$$

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

$$L_i(p, \omega) = \int_0^\infty T_r(p' \to p) S(p', -\omega) dt \tag{15}$$

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

$$L_i(p, \omega) = T_r(p_0 \to p) L_o(p_0, -\omega) + \int_0^t T_r(p' \to p) S(p', -\omega) dt' \tag{16}$$

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 beam 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.

2.3.1 Eulerian / Grid Based

Traditionally fluid simulations have been conducted using a grid to track density.

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 to be composed of a number of discrete particles. Typically each particle will be characterized by its position and velocity.

Particle based methods have the advantage of not needing to use computer memory in order to store information about where there is no density. 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 very possible that it would not be possible to use a fine enough grid to resolve the fine details of the smoke without surpassing 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.

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[7] which takes steps through the participating media and evaluates each segment. Each segment has a length of Δx 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:

$$L(x,w) = \sum_{l}^{N} L_l(x,w_l')p(x,w_l',w)\sigma_s(x)\Delta x + e^{-\sigma_t \Delta x}L(x+w\Delta x,w) \tag{17}$$

where N is the number of lights in the scene and L_l is the radiance from each light source. The first term sums the radiance contribution from 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:

$$L_{n+1}(x,w) = \sum_{l}^{N} L_l(x,w'_l)p(x,w'_l,w)\sigma_s(x)\Delta x + e^{-\sigma_t \Delta x} L_n(x + w\Delta x, w) \tag{18}$$

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. 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.

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. 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).

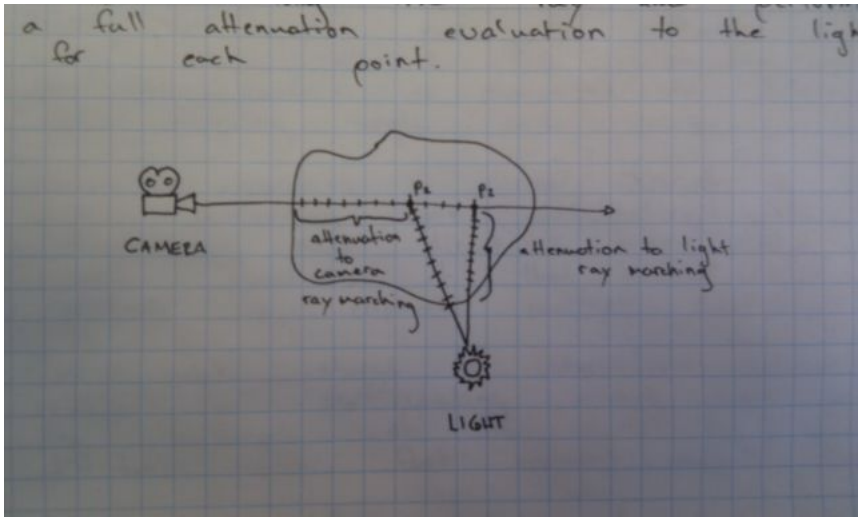


Figure 2: Solving Single Scattering with Ray Tracing

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:

$$\Lambda = \frac{\sigma_s}{\sigma_t} \tag{19}$$

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, this will in many cases result in a large number of low power photons which is computationally inefficient. Instead, Russian roulette[3] 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:

$$L_r(x,w) = \frac{1}{\pi r^2}\sum_{p=1}^{N} f_r(x,w_p,w)\Delta\Phi_p(x,w_p) \tag{20}$$

To estimate the radiance at some point in a volume a slight modification is required to account for sampling photons within a sphere instead of a disc on a surface:

$$(w\cdot\bigtriangledown)L_o(x,w) = \sum_{p=1}^{n} p(x,w_p,w)\frac{\Delta\Phi_p(x,w_p)}{\frac{4}{3}\pi r^2} \tag{21}$$

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

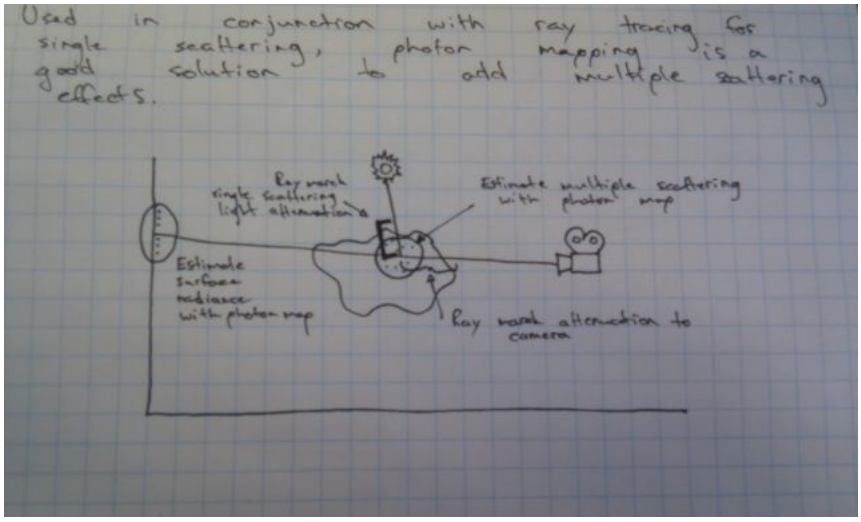


Figure 3: Solving Multiple Scattering using Photon Mapping

3 Related Work

3.1 Deep Shadow Maps

Deep shadow maps [8] 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 stores a representation of the transmittance through a pixel at a range of depths.

Deep shadow maps have the following advantages over traditional shadow maps:

- 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

A deep shadow map is 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 stored in the form of a linked-list of depth/transmittance pairs per pixel.

3.2 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 [6] 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.

3.3 Deep Opacity Maps

Deep opacity maps [10] 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.

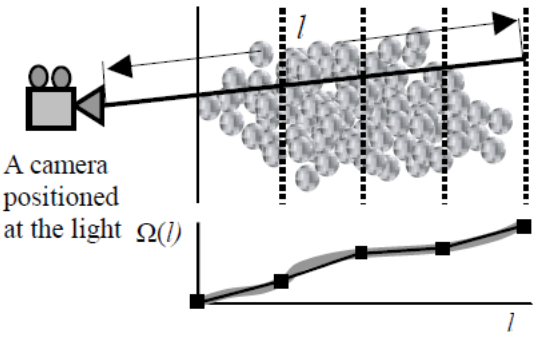


Fig. 1. The opacity function $\Omega(l)$ shown in a solid gray curve is approximated by a set of opacity maps.

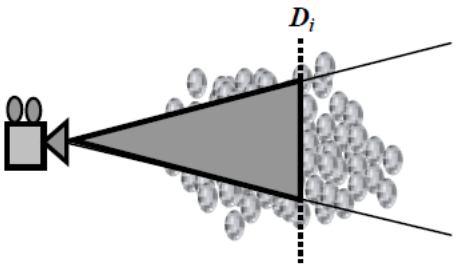
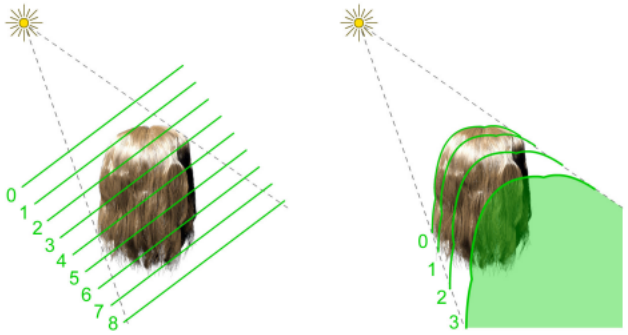


Fig. 2. The volume is rendered on each opacity map, clipped by the map's depth (D_i). The transparent region illustrates the clipped volume.

Figure 4: Opacity Shadow Maps



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

Figure 2: *Opacity shadow maps use regularly spaced planar layers. Our deep opacity maps use fewer layers, conforming to the shape of the hair model.*

Figure 5: Deep Opacity Maps

3.4 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 [5] 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.

4 The Particle Based Participating Media Rendering Problem

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 forming volumetric shadows
- Support for an arbitrary number of lights
- Support for all conventional computer graphics light types (points, spot, directional)

Our rendering technique models inhomogeneous participating media using particle data sets. Particles are used such that our simulation is spatially unbounded and not limited by system memory (as otherwise would be the case for grid based systems).

These requirements present a unique rendering challenge. We must solve two density integrations for each particle:

1. Find the attenuation of the light reaching the camera from a given particle
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 alpha blending we can solve this density integration per particle for millions of particles at interactive rates. Even with depth sorting (required to obtain correct blending) this is a relative non issue.

Furthermore, we have a pre-existing solution[9] to solve motion blur and depth of field. This involves using the geometry shader stage of the GPU pipeline to create point sprites. By varying the characteristics of these sprites we obtain these effects, as detailed in the following pertinent sections.

The rendering process begins with the lighting computation. The lighting data is computed using the CPU for ease of implementation reasons. The data is added to the vertex buffers which represent the particles and then 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.

4.1 Volumetric Lighting Calculation

In order to determine how much light reaches a particle from light source we must integrate the density along the light ray between that particle and light source. This presents a challenge for our rendering system, in that because we are modeling the participating media there is no grid which we can perform a ray marching density integration upon. Instead, we use a special formed octree as out density integrator.

4.1.1 Density Octree

In order to satisfy the requirement of a spatial data structure which can represent the density of our particle data sets we elect to use an octree.

The density octree is formed by inserting the constituent particles of our data set, subdividing octree cells when a user selected density is achieved. The octree has the useful property of only allocating memory in those areas where there is greater detail in the original particle data set.

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 in turn 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.

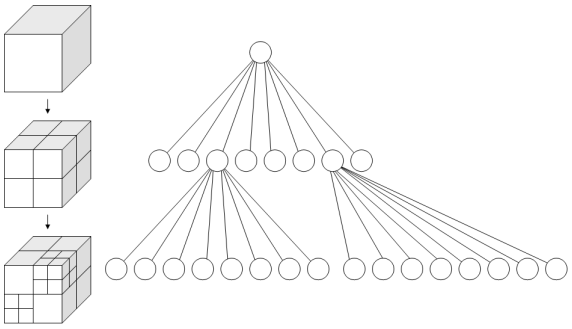


Figure 6: Octree formed through three subdivisons

Each octree node stores the number of particle which reside within the nodes spatial extent.

4.1.2 Density Octree Smoothing

<SAY SOMETHING

4.1.3 Lighting Evaluation

Integration of the density octree is performed via ray casting. A ray is constructed between some particle and the light source that is currently being evaluated. This ray is then intersected with the nodes of the octree in a top-down manner. This ray casting approach is advantageous

in that ray-AABB tests are only required only for those areas of density defined by the density octree. 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 these density evaluations only as necessary, computer memory is accessed less often resulting in better performance.

Although it would seem necessary to perform intersection testing between the light-particle ray and each particle, we experimentally determined that this is not the case. In fact, performing such exact testing actually results in over-defining the transmittance signal resulting in visually objectionable high frequency noise.

Furthermore, we have found that allowing the user to blend between a uniform density across the entire tree and the per cell density allows a great deal of flexibility in terms of tuning the appearance in terms of details vs. smoothness.

<INSERT OVER DEFINED EXAMPLE GRAPH>

We determine the attenuation by ray casting, counting the number of ray-particle interactions and calculating the attenuation accordingly.

$$particle\ illumination = e^{-ray\ casting\ density\ *\ density\ scaling\ factor} \tag{22}$$

for all particles **do**
  construct a ray from the particle to the light
  for all particles (or a subset) **do**
    check ray - particle intersection
    if intersection **then**
      attenuate light energy
    end if
    if *light energy* ≤ 0 **then**
      break ray-particle intersection checking loop
    end if
  end for
end for

Figure 7: Pseudo-code for naive ray-particle testing procedure.

4.2 Ray-Particle Intersection Testing

Fundamentally speaking, the particle 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 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, it is necessary to compute a complete and accurate lighting evaluation for each particle being rendered.

4.2.1 Intersection Algorithm (Ray-Sphere)

We use the following optimized ray-sphere intersection[1]:

4.2.2 Intersection Algorithm (Ray-AABB)

We use the following optimized ray-AABB intersection:
<INSERT CITATION FOR SHIRLEY INTERSECTION ALGORITHM>

4.3 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 BVH) on the solid geometry. Prior to conducting a full attenuation calculation using the density octree, the given particle-light ray in question is tested against the solid geometry BVH. If the ray intersects some 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 computational expense.
<INSERT DIAGRAM SHOWING AN EXAMPLE OF THIS>

4.4 Forward Density Integration using GPU Hardware

In order to determine the amount of light reaching the camera from some particle we must evaluate the out going radiance from each point taking into account the attenuation due to those particles which occlude the particle which we are evaluating. 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 centred 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.

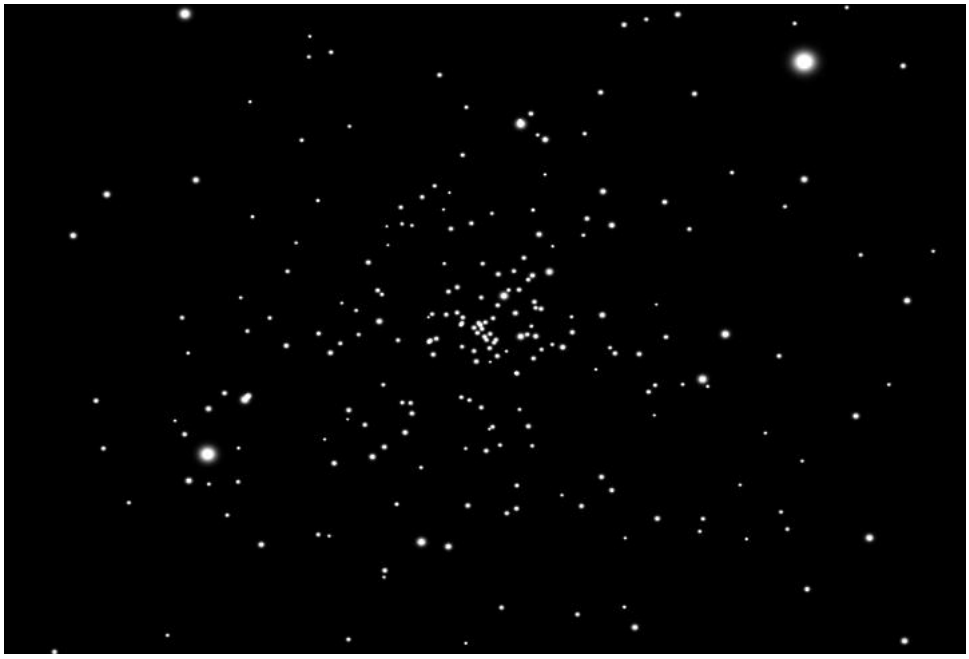


Figure 8: Example of a simple GPU particle rendering

4.4.1 Motion Blur

Motion blur is the blur in an image which 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 create such 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 relates to greater particle speed, a motion blur effect is achieved.

4.4.2 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 dropoff curve which controls the particle opacity is modified

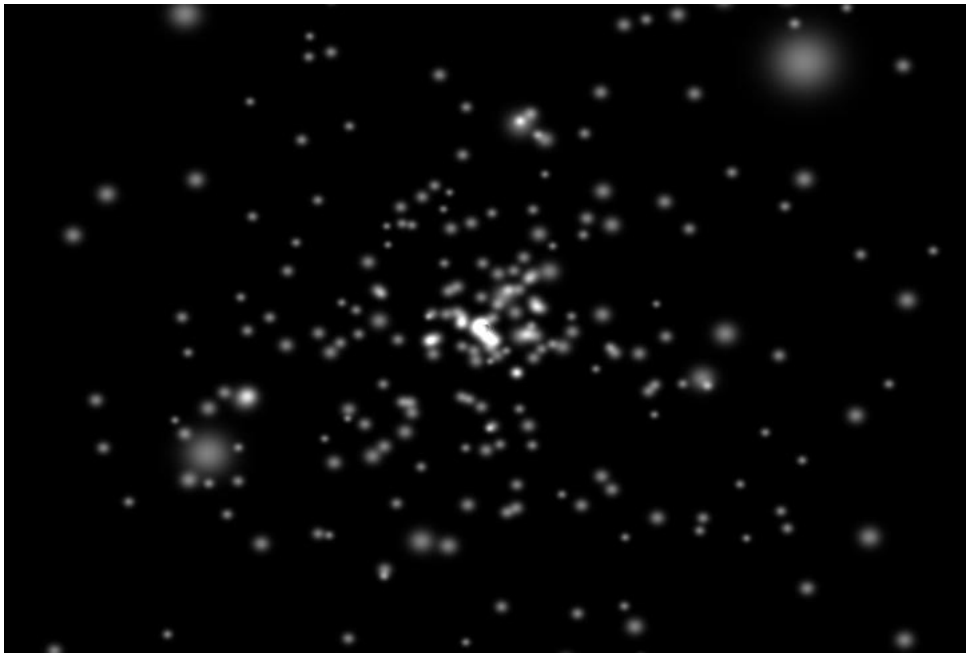


Figure 9: Example of rendering larger particles using the geometry shader scaling procedure

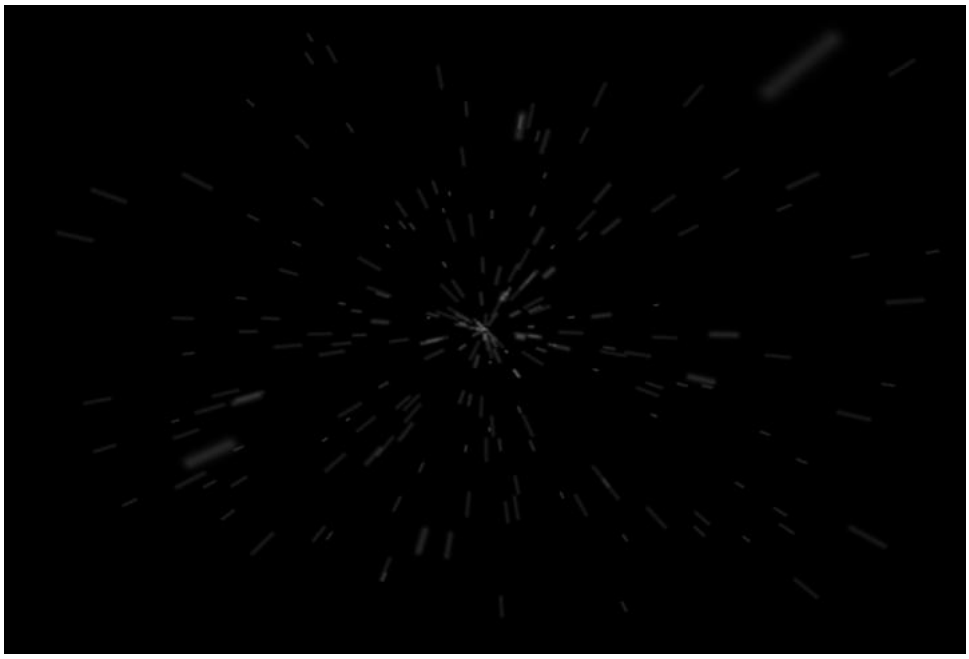


Figure 10: Example of the motion blur effect created using the geometry shader

to simulate the effects of depth of field.

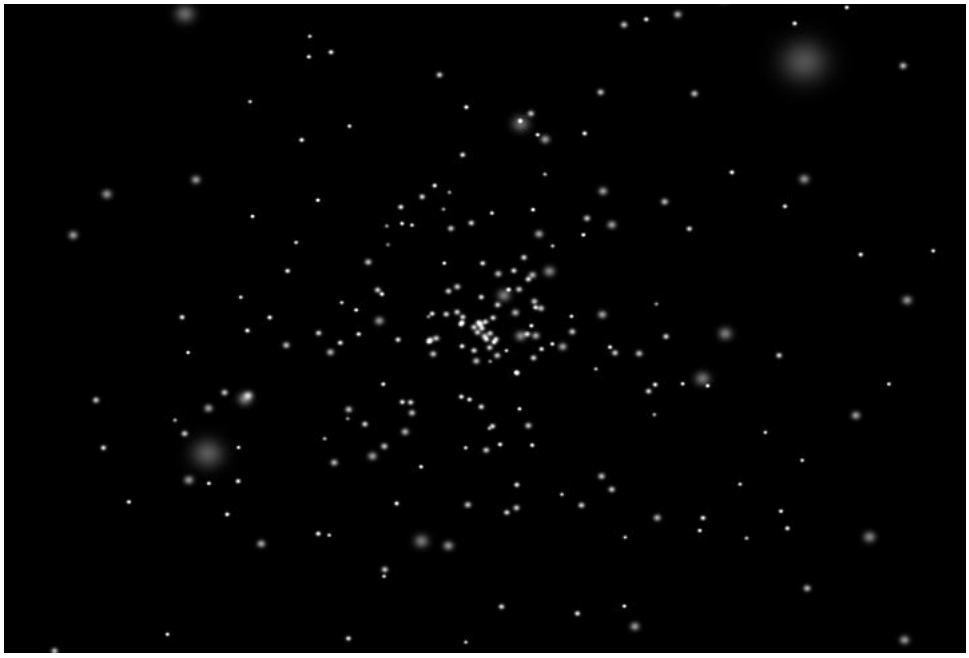


Figure 11: Example of the motion blur effect created using the geometry shader. Note how particles in the foreground appear out of focus.

4.5 Algorithmic Efficiency

In the naive formulation, our algorithm would be expected to run in $O(n^2)$ time. However, we can improve this expected run time in several ways:

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 characterized by a significant density, this alone improvement alone can significantly improve run times, as light will be attenuated to zero well before the total number of computations is performed.

2. Spatial Tree Structures: Ray casting can be optimized with any tree structure that would commonly be used for traditional ray tracing such as an octree or kd-tree.

   It is yet to be seen if the time required to build a tree on the particle data set (which will be changing per frame) out weighs the time saved by using such a structure.

3. Level of Detail: In regions further from the camera, it is not necessary that the lighting calculations be as accurate, as viewers will not be able to see detail in those areas. Using our algorithm it is easy to adjust the level of detail used by making the percentage of total

particles used in ray-particle checks a function of the distance to the camera. In this way, full details are only computed as necessary.

4. Parallelism: Because each particle calculation is independent, our algorithm exhibits a high degree of parallelism which can be exploited on multi-core or GPU hardware.

With these optimizations, notably the use of a tree based data structure, it is possible to reduce the expected run-time to $O(n\ log\ n)$.

Thus we have an algorithm which calculates lighting on a per particle basis with negligible memory overhead and has several computational optimizations.

4.6 Spatial Partitioning Tree

4.7 Multi-threading Support

In order to take full advantage of modern multi-core processors, we implement our solution using the Intel Threading Building Blocks (TBB) code library. This library exposes a number of convenient programming constructs that facilitate allowing code to scale 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 TBB 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 bandwidth will allow.

5 Performance Evaluation

We evaluate the performance of the above described system against the following metrics. Our point of comparison is Krakatoa, an industry standard particle rendering system produced by Thinkbox Software, which has been used in the production of numerous commercial films and commercials.

5.1 Image Quality vs. Particle Count

Note how Krakatoa requires significantly more particles to achieve as smooth a lighting solution. As a result of requiring more particles, Krakatoa takes more time to produce the same output quality.

5.2 Time vs. Particle Count

Using our lighting algorithm, time increases linearly with the number of particles in the scene.

5.3 Time vs. Number of Lights

Using our system, the time increase associated with adding more lights is linear.

5.4 Time vs. Image Size

Our lighting, motion blur and depth of field calculations are independent of ouput image size. As such, no more computation time is required for higher resolution output.

5.5 Example Images

6 Conclusion

6.1 Conclusions

6.2 Summary of Contributions

6.3 Future Research

We have identified the following areas as potential avenues for future research.

6.3.1 Cache Efficient Octree

At the core of our system is the octree data structure which is used to create the density evaluator. As such, any improvements that can be made to the efficiency of the octree will directly also improve our algorithm run times. One such improvement can come in the form of a cache efficient octree. On modern central processing units, where memory latency is hidden using special on processor caches, significant speed improvements can be obtained by specifically designing data structures which take advantage of such caches. We would be interested to revisit our implementation and determine whether additional performance could be achieved by using such specially designed octree implementations.

6.3.2 GPU Implementation of Octree

References

[1] Tomas Akenine-Möller, Eric Haines, and Natty Hoffman. *Real-Time Rendering 3rd Edition*. A. K. Peters, Ltd., Natick, MA, USA, 2008.

[2] James Arvo. *Transfer Equations in Global Illumination*. 1993.

[3] James Arvo and David Kirk. Particle transport and image synthesis. *SIGGRAPH Comput. Graph.*, 24:63–66, September 1990.

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

[5] Jon Jansen and Louis Bavoil. Fourier opacity mapping. In *Proceedings of the 2010 ACM SIGGRAPH symposium on Interactive 3D Graphics and Games*, I3D '10, pages 165–172, New York, NY, USA, 2010. ACM.

[6] Tae-Yong Kim and Ulrich Neumann. Opacity shadow maps. In *In Proceedings of the 12th Eurographics Workshop on Rendering Techniques*, pages 177–182. Springer-Verlag, 2001.

[7] Marc Levoy. Efficient ray tracing of volume data. *ACM Trans. Graph.*, 9:245–261, July 1990.

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

[9] Hahn E. Madill, J. and B. Houston. Fury renderer. Exocortex Technologies, Inc. Ottawa, Canada., 2010.

[10] Cem Yuksel and John Keyser. Deep opacity maps. *Computer Graphics Forum (Proceedings of EUROGRAPHICS 2008)*, 27(2), 2008.