

POINT-BASED COLOR BLEEDING WITH VOLUMES

A Thesis

Presented to

the Faculty of California Polytechnic State University

San Luis Obispo

In Partial Fulfillment

of the Requirements for the Degree

Master of Science in Computer Science

by

Christopher Gibson

June 2011

© 2011

Christopher Gibson

ALL RIGHTS RESERVED

## COMMITTEE MEMBERSHIP

TITLE: Point-Based Color Bleeding With Volumes

AUTHOR: Christopher Gibson

DATE SUBMITTED: June 2011

COMMITTEE CHAIR: Zoë Wood, Ph.D.

COMMITTEE MEMBER: Aaron Keen, Ph.D.

COMMITTEE MEMBER: Chris Lupo, Ph.D.

## Abstract

### Point-Based Color Bleeding With Volumes

Christopher Gibson

The interaction of light in our world is immensely complex, but with modern computers and advanced rendering algorithms, we are beginning to reach the point where photorealistic renders are truly difficult to separate from real photographs. Achieving realistic or believable global illumination in scenes with participating media is exponentially expensive compared to our traditional polygonal methods. Light interacts with the particles of a volume, creating complex radiance patterns.

In this thesis, we introduce an extension to the commonly used point-based color bleeding (PCB) technique, implementing volume scatter contributions. With the addition of this PCB algorithm extension, we are able to render fast, believable in- and out-scattering while building on existing data structures and paradigms. The proposed method achieves results comparable to those produced with Monte Carlo ray tracing but with drastically reduced run times, speeding up renders by nearly a factor of 10.

We present results that are both comparable to existing Monte Carlo integration methods while also achieving much faster run-times with minimal memory usage.

# Contents

<b>List of Tables</b>	vii
<b>List of Figures</b>	viii
<b>1 Introduction</b>	1
1.1 Graphics & Light . . . . .	1
1.2 Global Illumination . . . . .	1
1.3 Color Bleeding Techniques . . . . .	2
1.4 Our Contribution . . . . .	3
<b>2 Background</b>	5
2.1 Radiance . . . . .	5
2.2 BRDF and the BSRDF . . . . .	7
2.3 Volume Lighting . . . . .	8
2.3.1 Absorption: . . . . .	9
2.3.2 Scatter Out: . . . . .	9
2.3.3 Transmittance: . . . . .	9
2.3.4 Phase Functions . . . . .	10
2.3.5 Scatter In . . . . .	11
2.4 Monte Carlo Integration . . . . .	12
<b>3 Related Work</b>	13
3.1 Global Illumination . . . . .	13
3.1.1 PCB . . . . .	13
3.1.2 Photon Mapping . . . . .	14
3.2 Volume Rendering . . . . .	14

3.2.1	Seminal Work . . . . .	15
3.2.2	Multi-Resolution Volumes . . . . .	16
3.2.3	Occlusion Techniques . . . . .	17
3.2.4	Volume Lighting . . . . .	17
<b>4</b>	<b>PCB Extension Algorithm</b>	<b>19</b>
4.1	Point Based Color Bleeding . . . . .	19
4.2	Extension Overview . . . . .	20
4.3	Sampling the Scene . . . . .	21
4.3.1	Surfel Sampling . . . . .	22
4.3.2	LVoxel Sampling . . . . .	23
4.4	Gathering Light . . . . .	23
4.4.1	Point-Cloud Ray Casting . . . . .	23
4.4.2	Hemisphere Sampling . . . . .	24
4.5	Integrating Volume Data . . . . .	26
4.5.1	Data-Structure Modifications . . . . .	26
4.5.2	Octree Traversal . . . . .	27
4.5.3	Acquiring Scatter-Out Contributions . . . . .	28
4.5.4	Acquiring Scatter-In Contributions . . . . .	28
<b>5</b>	<b>Results</b>	<b>31</b>
5.1	Environment . . . . .	32
5.2	Test Scene . . . . .	33
5.3	Data Comparison . . . . .	33
5.4	Analysis . . . . .	34
5.4.1	Memory . . . . .	34
5.4.2	Speed . . . . .	34
5.5	Conclusion . . . . .	36
<b>6</b>	<b>Future Work</b>	<b>38</b>
	<b>Bibliography</b>	<b>40</b>

# List of Tables

# List of Figures

1.1	A simple Cornell box scene showing simple global illumination. . . . .	2
1.2	Comparison of the PCB extension (left) and traditional Monte Carlo results (right.) . . . . .	4
2.1	Evaluation of radiance at a point on an opaque surface. Only the hemisphere around the surface normal is considered. Incoming radiance is measured and scaled by its solid angle. . . . .	6
2.2	Light scatter properties vary based on the participating media. . . . .	8
2.3	Visual representation of a phase function around a scatter point. The area represents the distribution of the scattered light about a sphere. . . . .	10
2.4	Comparison between a volume with (right) and without (left) in-scattering contribution. . . . .	12
3.1	Global illumination example achieved via photon mapping. Source: <a href="http://graphics.ucsd.edu/~henrik/papers/photon_map/">http://graphics.ucsd.edu/~henrik/papers/photon_map/</a> . . . . .	15
3.2	Volume renders from [10] showing CT scan volume data visualized in three-dimensions, complete with realistic lighting. . . . .	16
4.1	An example of a point-cloud scene where the geometry had been sampled, a disc representation replacing the geometry. The radii of the discs in this image are reduced to better exemplify their presence. . . . .	20
4.2	Rays are cast from a special camera during the surfel sample phase. Each time the ray intersects with geometry a surfel is created. . . . .	22
4.3	Basis vectors are generated based on the surface normal in order to transform samples on a hemisphere to test surrounding radiance.	24

4.4	Gather rays are cast into the point-cloud, returning the estimated radiance coming from a given direction. The radiance is then scaled based on the solid angle of that sample cast (based on sample count.) . . . . .	25
4.5	Illustrates the octree traversal algorithm testing efficiency. Lighter shades represent less tests, while darker shades represents the most. Closer objects evaluate faster, and all data in the point-cloud behind them are occluded. . . . .	27
4.6	Sample rays are cast during volume traversal, allowing for decent estimates of lighting contribution at each point. . . . .	29
4.7	Stepping allows for estimation of the integral through the entire volume. Shadow rays are cast intermittently to estimate the direct lighting contribution. . . . .	30
5.1	Sponza Atrium mesh and Stanford bunny volume rendered using the PCB Extension algorithm. . . . .	31
5.2	A test scene showing a light's interaction with a volume changing depending on the direction and position of the light. . . . .	32
5.3	Real-time visualization of the Sponza Atrium mesh in MeshLab. .	33
5.4	Zoomed image showing PCB extension (left) and Monte Carlo (right.) . . . . .	35
5.5	Zoomed image showing traditional PCB (left) and PCB with extension (right.) Note the visible color bleeding with our method. .	35
6.1	Example of point-based color bleeding without the volume extension algorithm. . . . .	42
6.2	Example of a scene almost entirely in shadow, showing indirect lighting in play. . . . .	43
6.3	The Sponza Atrium with the Stanford volumetric bunny. In-/Out-scattering is evident on the volume and on the surrounding atrium walls. . . . .	43
6.4	Image exemplifying clear out-scattering from Stanford bunny volume. . . . .	44
6.5	Image exemplifying clear color bleeding next to the red wall in the bunny's shadow and correct transmittance through the bunny's hollow form. . . . .	44

6.6	The black occluding geometry in the center stops all but the light to the left to enter below. In and out scattering on the volume and walls is evident. . . . .	45
6.7	Illustrates how a light may act when placed within a hollow volumetric object. The bunny is shown as slightly brighter, scattering light about the scene (especially underneath it, where there are holes in the volume.) . . . . .	45
6.8	Shows how CT scan data can be used to visualize scanned objects like a human face. Subsurface scattering and transmittance through thin materials is evident. . . . .	46

# Chapter 1

## Introduction

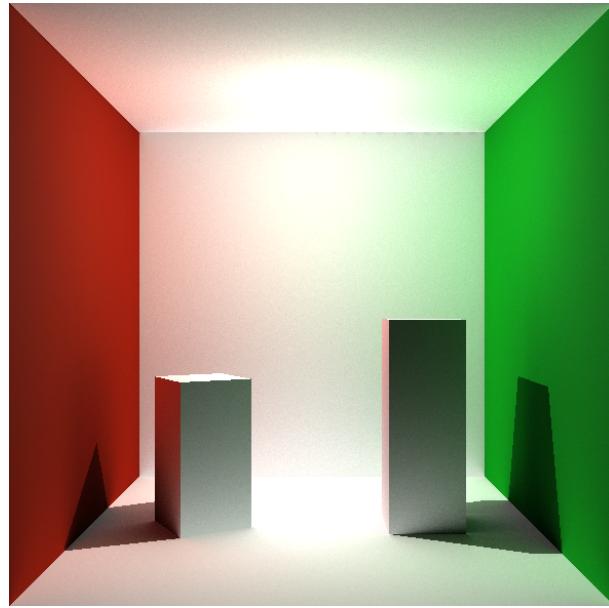
### 1.1 Graphics & Light

At its core, computer graphics is the visualization of light and its interaction in an abstract world, be this a simulation or a virtual world created by artists. The ability to render scenes with realistic lighting is desirable for many entertainment application settings such as film. Even in non-photorealistic renders, the interaction of light is of paramount importance.

### 1.2 Global Illumination

The effort to accurately evaluate the radiometric quantities within a virtual scene, especially in non-realtime systems, may be referred to collectively as *global illumination* [15]. In order to achieve greater realism, scenes must rely on more than simple direct-lighting algorithms and move to more complex systems to better evaluate light interaction (Figure 1.1.) This field of study has lent itself

to the breathtaking visual effects in movies, advertisements, television shows and other artistic mediums. The line between real and fake is getting blurrier every year as the lighting calculations become more and more exact. As we have pushed the boundaries of our graphical capabilities, we also increase our computational complexity exponentially.



**Figure 1.1:** A simple Cornell box scene showing simple global illumination.

### 1.3 Color Bleeding Techniques

Great results have been achieved for lighting complex scenes using point based color bleeding [2] algorithms, where a point-cloud representation of a scene's direct lighting is computed for the purpose of efficiently calculating a representation of the scene's radiometric properties surrounding any given point. Per H. Christensen's color bleeding algorithm (or similar implementations) is already in place in many production companies and used in many feature films.

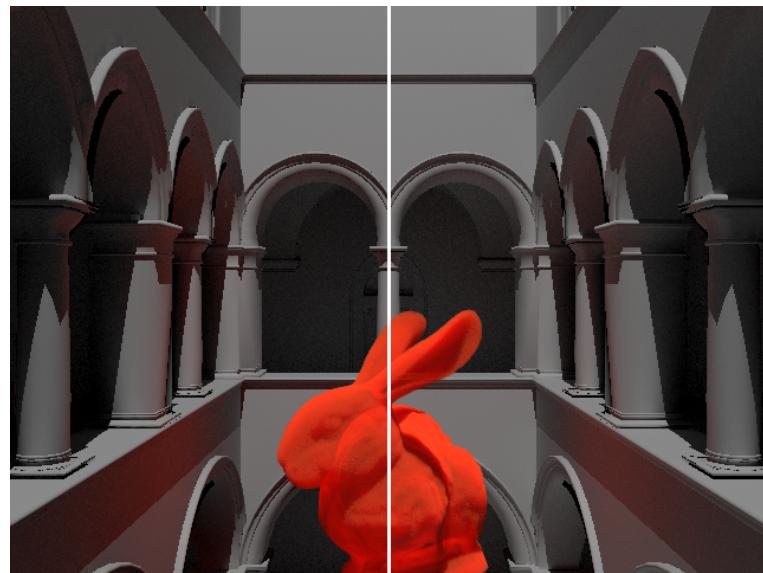
These algorithms, however, tend to limit or omit entirely the lighting contribution from volumetric data or participating media within the scenes. Due to the highly complex nature of the domain, coupled with the computational complexity involved therin, many algorithms choose to disregard this portion of the lighting algorithm, often leaving the programmer or artist to fake the volume's contribution in other ways.

## 1.4 Our Contribution

This paper presents an algorithm to address this missing component in the point based color bleeding algorithm. Specifically, we propose the addition of a data representation tuned to volumes, *light-voxel* (*or lvoxel*) to address the need to represent participating media to an existing global illumination algorithm which leverages a point cloud representation of a scene. Over the course of this paper, we will:

1. Discuss the surrounding subjects of light, volume rendering and global illumination,
2. List and describe the seminal and related works,
3. Describe how our algorithm meets our requirements and
4. Analyze the results of our implementation.

Our method achieves results comparable to those produced with Monte Carlo ray tracing but with drastically reduced run times, speeding up renders by a factor of 10. Figure 1.2 clearly illustrates a close comparison of our algorithm and Monte Carlo ray traced results.



**Figure 1.2:** Comparison of the PCB extension (left) and traditional Monte Carlo results (right.)

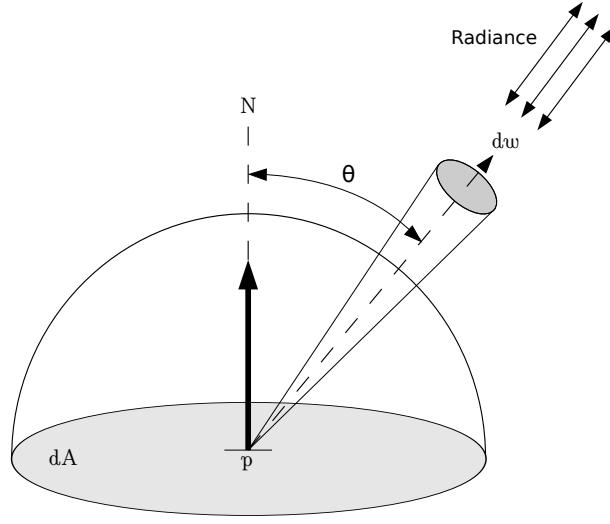
# Chapter 2

## Background

The goal of the proposed method is to include volumetric representations into a global illumination algorithm in a fast and coherent way. One of the unique features of participating media is that they must be represented with a more complex data-structure than solid geometric objects which are usually polygonalized in most rendering processes. Light interacts with the particles of a volume, creating complex radiance patterns (increasing the necessary computational complexity exponentially.) In particular the most fundamental concepts are presented here, (based off of [12]).

### 2.1 Radiance

Irradiance is the change of flux (radian power) over an area, denoted by  $E = \frac{d\Phi}{dA}$  [13]. Another way to look at this problem involves the relationship between the surface and surrounding radiometric quantities. Radiance helps us evaluate how much power enters or leaves any given point. The definition of radiance leaving a surface can be denoted  $L(p, w)$  given  $p$  is the point on a surface and



**Figure 2.1:** Evaluation of radiance at a point on an opaque surface. Only the hemisphere around the surface normal is considered. Incoming radiance is measured and scaled by its solid angle.

$w$  is the direction we are evaluating. The flux is projected upon the area of the surface  $dA$  based on the solid angle  $dw$ , which gives us Figure 2.1 and the following equation:

$$L = \frac{d^2 \Phi}{dwdA^\perp} \quad (2.1)$$

Inversely, incoming radiance, or irradiance, is evaluated by the strength of the light coming from any given direction  $w$ . This can be represented by the following:

$$E = \int L(p \leftarrow w) \cos\theta dw. \quad (2.2)$$

$L(p \rightarrow w)$  represents the radiance leaving point  $p$  and  $L(p \leftarrow w)$  represents incoming radiance given a direction  $w$ . Note that radiance along a straight path

is invariant. For example:

$$L(x \rightarrow y) = L(y \rightarrow x). \quad (2.3)$$

Measuring the incoming radiance at any given point  $p$  in all directions is the key to the graphics lighting equation, and a crucial element in how a rendered scene looks and feels.

## 2.2 BRDF and the BSRDF

The *bidirectional reflectance distribution function* (or BRDF) is a function that gives us a formal method of describing the reflected radiance from a surface given an incident radiance from another light source or emissive surface [?]. This simplification helps us with quick evaluations in scenes with simple lighting (such as point light sources) and other direct-lighting techniques. The BRDF can be generalized into the following equation, based off of Equation 2.2. If the incident direction  $w_i$  represents the differential cone of incoming directions, the differential radiance at point  $p$  is represented by:

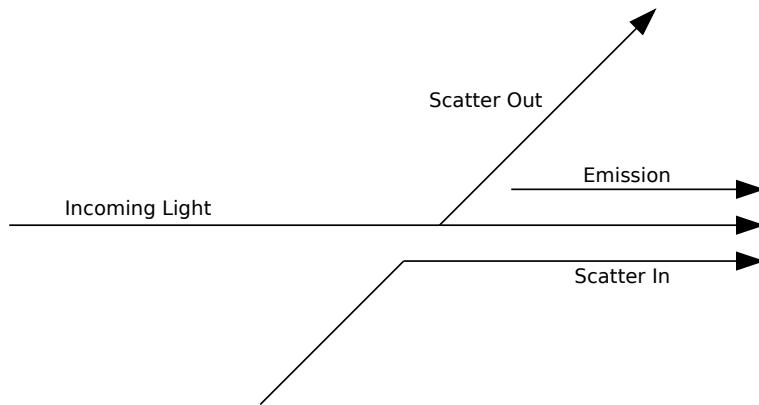
$$dE(p, w_i) = L_i(p, w_i) \cos\theta_i dw_i. \quad (2.4)$$

Many materials such as skin, marble and plastic do not simply reflect the incoming light energy, but also transmit it through the surface, a process called *subsurface scattering*. Given the obvious shortfalls of the BRDF in this instance, the BSSRDF (*bidirectional scattering-surface reflectance distribution function*) takes into account a surface's scatter properties. This generalized function describes the ratio of "exitant" differential radius from incoming point and direction

$p_i, w_i$  to an outgoing point and direction  $p_o, w_o$ . This turns a one-dimensional reflectance equation into a two-dimensional scatter equation, significantly increasing its complexity.

## 2.3 Volume Lighting

The BSSRDF describes the complexities of light traveling and scattering within complex surfaces similarly to that of volumes. Volumes follow very similar behaviors to opaque surfaces in terms of radiance, except on a particle-level. Participating media like smoke or fog is made up of particles which cause the scatter (i.e. clouds) and absorption/extinction (i.e. smoke,) behaviors that would be extremely computationally expensive to model and simulate on any scale. Therefore, such behaviors are modeled in terms of Transmittion, emission, Scatter in and Scatter out like in Figure 2.2. We are able to identify the probability that light will be absorbed, scattered and/or transmitted through any point in a participating medium by identifying their probability density functions, as described in the following sections.



**Figure 2.2:** Light scatter properties vary based on the participating media.

### 2.3.1 Absorption:

As light passes through a participating media, light will become absorbed based on its absorption probability density  $\sigma_a$ . As stated earlier in Equation 2.3, it is known that radiance along a straight path is invariant, which allows us to estimate the amount of light absorbed or scattered given point  $p$  and direction  $w$  with the following:

$$e^{-\int_0^d \sigma_a(p+tw, w) dt}, \quad (2.5)$$

where  $\sigma_a$  represents the probability density that light will be absorbed over a distance  $dt$ .

### 2.3.2 Scatter Out:

In addition to being absorbed by the medium, light can be scattered based on a scatter probability density  $\sigma_s$ . As light is scattered and thus redirected, the amount of energy passing through the density in direction  $w$  is reduced. We can model the scatter equation through the following equation:

$$dL_o(p, w) = -\sigma_s(p, w)L_i(p, -w)dt. \quad (2.6)$$

$dL_o$  represents the outgoing radiance given point  $p$  and direction  $w$ .

### 2.3.3 Transmittance:

Both (2.5) and (2.6) involve the reduction of energy through a volume, reducing how much energy passes through (also known as its transmittance.) The

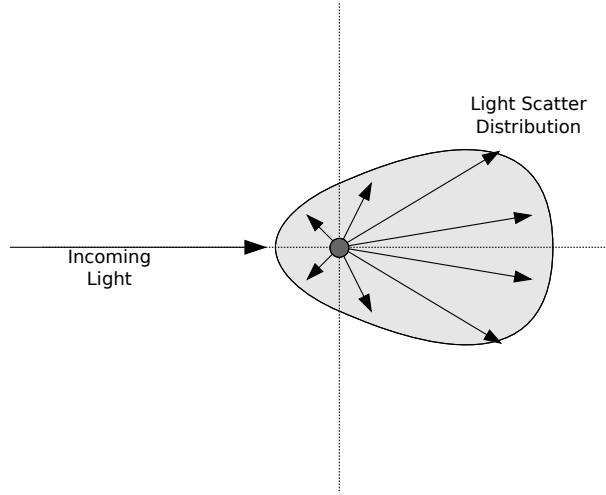
two can be combined into the following overarching representation:

$$\sigma_t(p, w) = \sigma_a(p, w) + \sigma_s(p, w). \quad (2.7)$$

Equation 2.7 gives us transmittance ( $\sigma_t$ ) at point  $p$  and direction  $w$ . Using this representation, we can integrate over a ray passing through the volume in order to evaluate the resulting radiance transmittion:

$$T_r(p \rightarrow p') = e^{-\int_0^d \sigma(p+tw, w) dt}. \quad (2.8)$$

### 2.3.4 Phase Functions



**Figure 2.3:** Visual representation of a phase function around a scatter point. The area represents the distribution of the scattered light about a sphere.

When dealing with particles in volumes that may scatter light, a distribution function or *phase function* describes the angular distribution of light scattered, described as  $\text{phase}(w \rightarrow w')$ . The probability that light may scatter from direction  $w$  to  $w'$  is described using this function. This distribution is visualized in

Figure 2.3. All tests in this paper were rendered using one of the simplest phase functions, known as the isotropic or *constant* phase function which represents the BRDF analog for participating media [1].

### 2.3.5 Scatter In

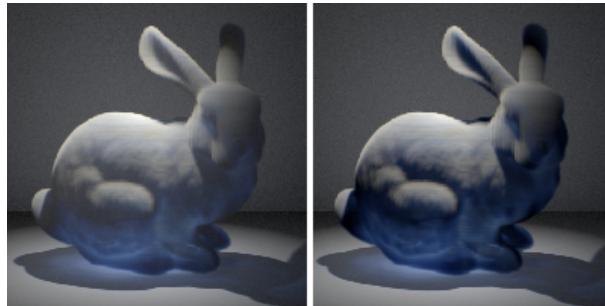
Although  $\sigma_s$  may reduce the energy of a ray passing through a volume, radiance from other rays (scattered by their respective phase functions as seen in Figure 2.3) may contribute to the original ray's radiance. This allows for radiance emitted from surrounding participating media and geometry to contribute to any given point within the volume, as exemplified in Figure 5.3. Before we can integrate incoming radiance, we must take into account the *source term*  $S$ , or total added radiance per unit distance, where the following constraint must hold true:

$$\int_{\mathbb{S}^2} \text{phase}(w \rightarrow w') dw' = 1. \quad (2.9)$$

This normalization makes sure that the phase function accurately defines the probability distribution for a particular direction. Given a summation of all outgoing radiance along every direction  $w'$ , we should be left with the total incoming radiance from direction  $w$ .

Finally, assuming conditions are met for Equation 2.9, we can integrate the total radiance scatter based on the normalized phase function  $\text{phase}(w \rightarrow w')$  over all directions  $w'$  to get our total scatter in a direction  $w$ :

$$S(p, w) = L_{ve}(p, w) + \sigma_s(p, w) \int_{\mathbb{S}^2} \text{phase}(p, -w' \rightarrow w) L_i(p, w') dw'.$$



**Figure 2.4:** Comparison between a volume with (right) and without (left) in-scattering contribution.

$L_{ve}(p, w)$  represents the emission coefficient of a volume and is not discussed in this paper.

## 2.4 Monte Carlo Integration

Monte Carlo methods have many applications in estimating complex systems through the use of random numbers and sampling schemes. One of the most useful technique is Monte Carlo integration, which estimates the integral of an arbitrary function through sampling discrete values defined over a specified domain [13]. For this reason, Monte Carlo has become *integral* in the field of computer graphics, where it may manifest itself in evaluating incoming radiance over a surface, estimating light scatter, or randomly sampling area lights in order to get soft shadows.

Many of the elements listed above can be (and are) estimated using this technique. In order to get good results, however, many hundreds of samples may be necessary. The cost of sampling using this method may still be cost-prohibitive, at which point the problem lies in the sample method itself, not how the samples are used or generated.

# Chapter 3

## Related Work

### 3.1 Global Illumination

Global illumination is an important problem in computer graphics, that numerous successful algorithms involving photon mapping [6], radiosity [4] and Monte Carlo sampling techniques [18] try to mitigate or overcome in a reasonable timeframe. The most closely related methods are those that sample the scene and use a two phase approach to model direct illumination and indirect illumination.

#### 3.1.1 PCB

Of particular relevance to this work is that done in Point-Based Approximate Color Bleeding developed by Per Christensen [2]. This method thoroughly samples all of the geometry in a scene and creates a point cloud representation of the direct lighting at each point., used to evaluate the incoming radiance surrounding a given point on a surface.

As recently as 2010, discussion of approximating volume scattering using point

clouds was mentioned in [3], but did not offer how *back-to-front or front-to-back rasterization* would be achieved with the current rasterization method (handled by our octree traversal method) or how scatter, extinction and absorption would be managed within the three-dimensional volume representation inside the point cloud.

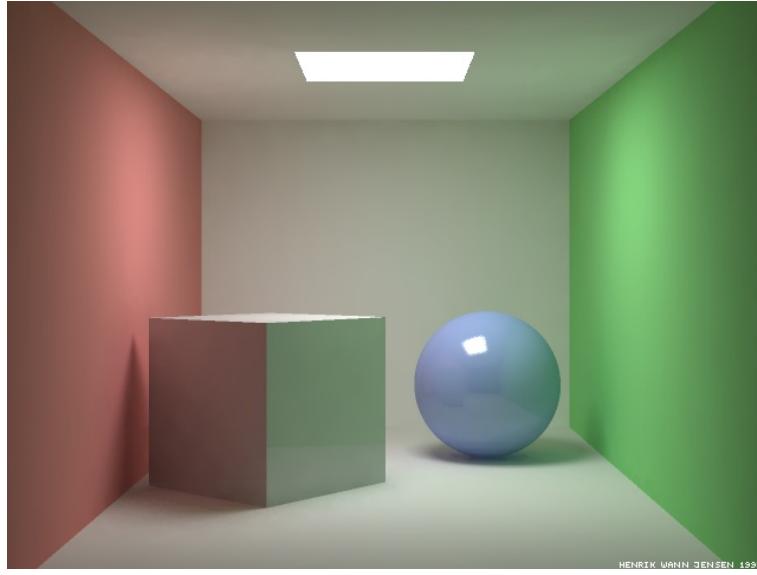
### 3.1.2 Photon Mapping

Other closely related work includes attempts at simulating light scatter and absorption properties of participating media through existing photon mapping techniques, which have shown promise in the past. In [7], Jensen describes a process where photons participate and become stored inside the volume itself for later gathers during volume integration.

While this technique is shown to work, it primarily focuses on caustic effects in volumes and the generated photon map. Our storage method does not require data to be stored in the volume, but in a separate, more lightweight data-structure better suited for out-of-core rendering.

## 3.2 Volume Rendering

This paper is focused on the lighting and rendering of scenes which contain volume data. A number of approaches have been developed in order to represent volume data through computer visualizations or renders [10][8].



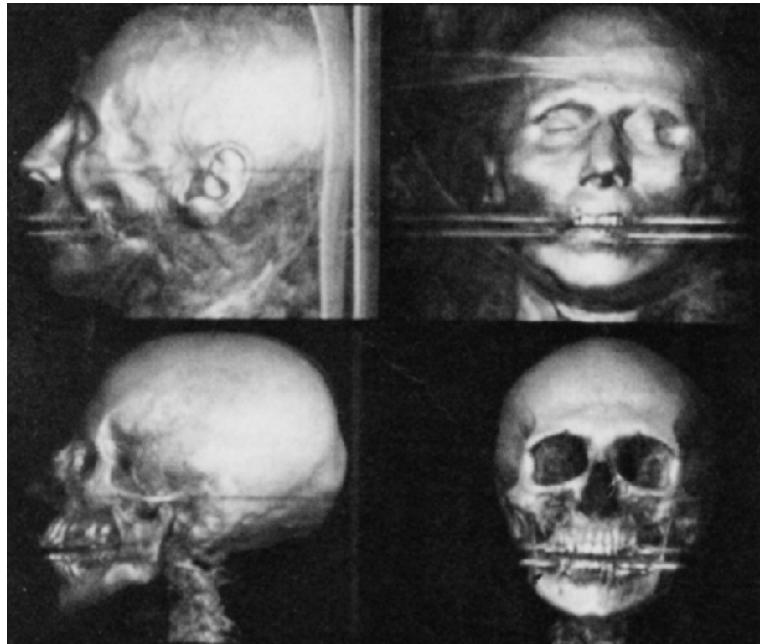
**Figure 3.1:** Global illumination example achieved via photon mapping.  
Source: [http://graphics.ucsd.edu/~henrik/papers/photon\\_map/](http://graphics.ucsd.edu/~henrik/papers/photon_map/)

### 3.2.1 Seminal Work

Some of the most basic volume rendering techniques are described in [10]. Before the time of the paper's creation, many of the accepted methods of volume visualization involved generating polygonal representations of the volumes by sampling the opacities and comparing them to a selected isovalue to determine whether or not the voxel is designated as "inside" the volume or "outside." A polygonal mesh is then constructed based on this differentiating boundary. Unfortunately, the algorithm defined above fell prey to spurious surfaces and holes caused by a limited sample range (since the polygonal method suffered from having to make a binary decision. Either a ray intersected the volume or it did not.)

In response to this, Levoy proposed a process of testing against two arrays (one with opacities and one with colors) that represent the voxel datastructure. If a voxel was intersected, the algorithm would interpolate its color and opacity

to those surrounding it based on the nature of the intersection. This allowed for transparent volumes, and also allowed for opacities to be separated from voxel color, allowing for some important pre-processing to be done on the data such as volume classification, where the opacity of a volume determined on an iso range. This in turn allowed the renderer to focus attention to specific densities in scans (useful for medical imaging.)



**Figure 3.2:** Volume renders from [10] showing CT scan volume data visualized in three-dimensions, complete with realistic lighting.

### 3.2.2 Multi-Resolution Volumes

Because of the complexity of volume data (both through data representation and computation,) volume rendering algorithms often implement efficient multi-resolution data representation [16]. [11] describes a hierarchical method of managing a volume data set in order to remove unnecessary "Empty" cells and to reduce the amount of intersection tests done on voxels (or "Cells", as the higher

level nodes are referred to.) This algorithm introduced the use of octrees as an efficient means of dividing and containing the volume data.

Our algorithm implements a sparse octree data-structure for both the volumes in the scene and the point-cloud used for the indirect lighting equation.

### 3.2.3 Occlusion Techniques

Another method involves estimating what nodes in a volume octree are definitely occluded based on surrounding node densities. [5] describes building a two-dimensional occlusion map and filling it in over a series of iterations, removing occluded nodes from being tested each iteration. This method has been shown to drastically increase rendering performance by removing as much as 30% per iteration.

Like many seminal volume rendering algorithms, our implementation takes advantage of a multi-resolution, view-independant octree datastructure in order to handle a large amount of complex lighting and volume data, skipping material occluded by opaque geometry cached in the data-structure in the form of surfels.

### 3.2.4 Volume Lighting

As our computational power has improved, we have been able to tackle problems in lighting that we could not have overcome in the past. [9] builds upon Levoy's volume rendering method by implementing shadowing by sending shadow rays toward each light for each sample within the volume in order to estimate the amount of extinction between the point and the light. Indirect lighting is also employed (though only forward-scattering due to the incremental nature of their

algorithm.)

[19] attempts to handle multiple-scattering and volume shadows in scenes that sport mixed polygonal and volumetric data. The paper describes light scatter representations similar to Equations 2.5, 2.6, 2.7, where light is able to scatter in and out from the sampling ray. The algorithm then handles volume shadows caused by polygonal mesh data by constructing a series of shadow buffers, evaluating the volume shadow as a texture at each slice.

While our volume lighting equation takes into account volume scatter properties, we do not evaluate shadows in a separate pass. Instead, our algorithm not only evaluate transmittance between arbitrary sample points and the scene lights (giving us believable direct lighting,) but we simulate scatter in properties by casting monte-carlo samples out into our point-cloud to evaluate scene and volume radiance. Additionally, our point cloud is not constrained by our traversal method, so all forms of scatter are supported. Finally, the algorithm described also does not handle scatter out contributions to the scene, where the volume data may contribute to the rest of the scene’s lighting.

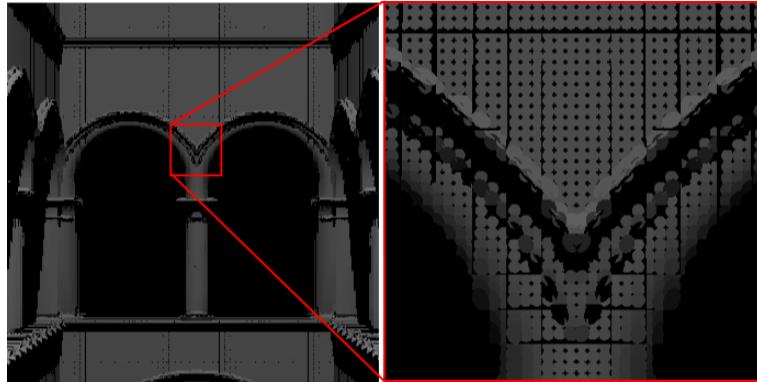
# Chapter 4

## PCB Extension Algorithm

We present an algorithm which is an extension to the point cloud techniques described in [14] and [2], specifically building off the point-based color bleeding (PCB) technique by Christensen. The modifications involve evaluating light scatter and absorption properties at discrete points in the volume and adding them to the point cloud. Using a front-to-back traversal method, we can correctly and quickly approximate the *light-volume* representation's contribution to a scene's indirect lighting evaluation.

### 4.1 Point Based Color Bleeding

In general, the color-bleeding algorithm subdivide the world into small representational segments, called surfels in [2], which are stored in a large point cloud, representing the scene (See Figure 4.1.) Surfels are used to model direct illumination, and are then used in a later phase to compute indirect lighting and color bleeding in an efficient manner. This method is split up into three stages:



**Figure 4.1:** An example of a point-cloud scene where the geometry had been sampled, a disc representation replacing the geometry. The radii of the discs in this image are reduced to better exemplify their presence.

1. Sample the scene and save a discrete representation of the surfaces along with direct lighting in a point cloud
2. Perform normal ray tracing on the scene geometry
3. Replace ambient estimates with a gather stage, sampling the scene around a point to gather the indirect lighting component

The goal of our proposed method is to include volumetric representations into a global illumination algorithm in a fast and coherent way similar to how surfels are represented in the point cloud.

## 4.2 Extension Overview

In the existing algorithms [2], surfels represent opaque materials within the point cloud, thus to incorporate a representation of volumetric data, an additional data representation was necessary to handle the scatter and absorption properties of participating media. In general, our data representation closely follows the

model of surfels, in that we choose to sample the volume at discrete locations and store a finite representation of the lighting at those discrete locations, but with modifications to handle the special attributes of lighting in transparent media. In keeping with the naming conventions established, we call our discrete sampling of lighting elements for a volume: *lvoxels*.

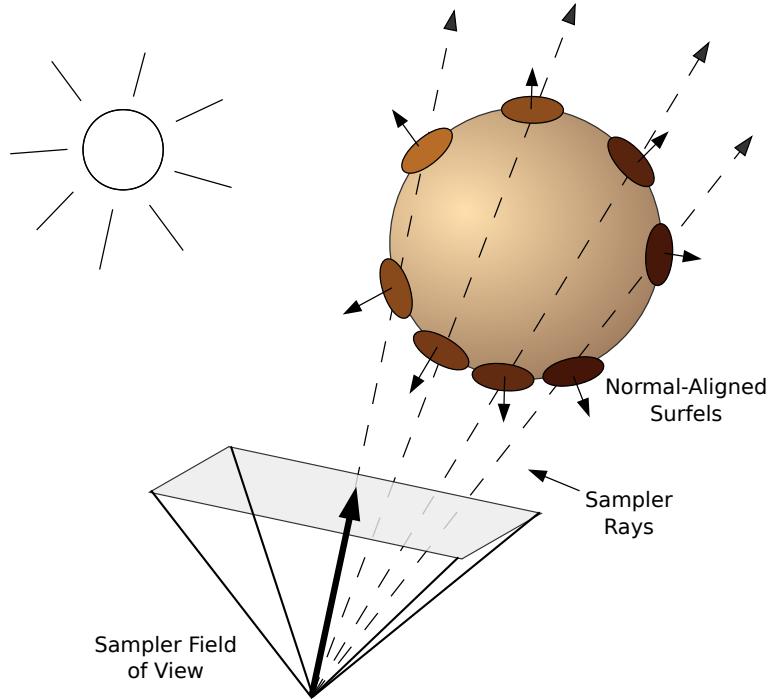
As a quick review, our algorithm must do the following:

1. Sample the scene geometry and store the direct lighting (or relevant lighting properties) within an acceleration structure for fast evaluation
2. Sample the participating media and evaluate scatter, absorbtion and direct lighting at each discrete point
3. Identify points of interest during regular ray casts using scene geometry
4. Orient a set of hemispherical samples along the normals of raycast surfaces and cast the rays into the point-cloud
5. Model the scatter-out and scatter-in properties of volumetric lighting during the indirect lighting gather stage.

### 4.3 Sampling the Scene

The goal of this stage of the algorithm is to sample the scene geometry (including the volume) and store the direct lighting in a finite data representation to be used later for global illumination lighting effects. As all of our finite data represents the direct lighting of some small portion of a surface or element in a three-dimensional scene, we refer to the union of all finite lighting samples as a “point cloud”. This point cloud is stored in an octree representation for efficient

access to all data elements, surfels and lvoxels. Surfels differ from lvoxels only in that surfels represent a flat, solid geometry while lvoxels represent a transparent, volumetric medium. Both have radii and position so both can be placed within the same point cloud.



**Figure 4.2:** Rays are cast from a special camera during the surfel sample phase. Each time the ray intersects with geometry a surfel is created.

### 4.3.1 Surfel Sampling

We sample the opaque geometry in surfels, which are computed using an abstract sampling camera with a field of view slightly larger than the current viewing frustum, with a sampling rate two times that of the desired pixel resolution. Rays are cast from the sampling camera and intersections with geometry mark sample points as seen in Figure 4.2, giving the scene a view-dependant, thorough sample set.

### 4.3.2 Lvoxel Sampling

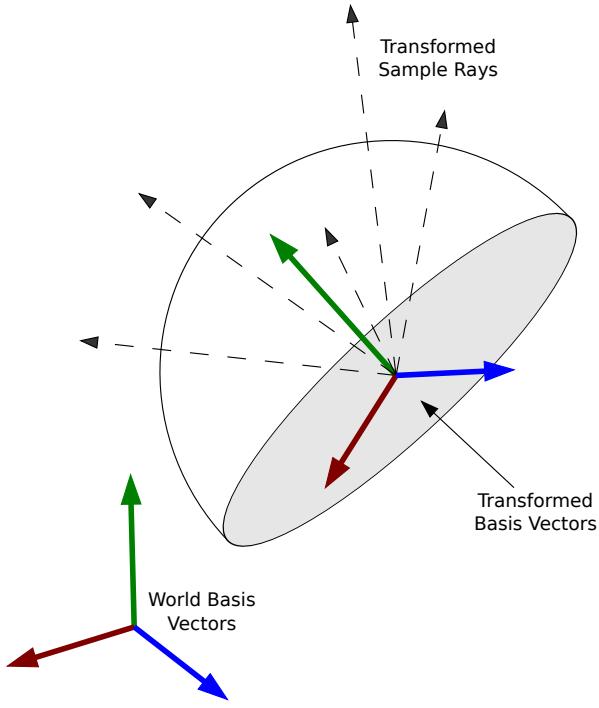
Lvoxels are generated by marching over the entire domain of the volume by a specific, preset interval, sampling scatter and absorption coefficients in order to get an average throughout the area an lvoxel will occupy. Typically this involves eight to sixteen absorption and scatter samples per lvoxel. These values, as well as the radius of the lvoxels, may differ depending on the complexity and raw resolution of the volume.

Caching the direct light contribution at each lvoxel by testing the transmittance (2.8) to each light source saves us from re-computing light calculations during sampling in sections 4.5.3 and 4.5.4 [17].

## 4.4 Gathering Light

### 4.4.1 Point-Cloud Ray Casting

Next, our algorithm uses a gather stage similar to the one in PCB, which calculates the irradiance at a point on a surface, given the radiance of the scene around it. Unlike PCB, which uses a software rasterization method, we chose to evaluate irradiance by raycasting into the point-cloud around a hemisphere oriented along the surface's normal. The decision to cast out of a hemisphere rather than using a software rasterization technique as was adopted in previous PCB implementations was made to simplify the tests which compare traditional Monte Carlo sampling methods to the extended PCB algorithm, but also to simplify evaluation of the transparent lvoxels within the octree.

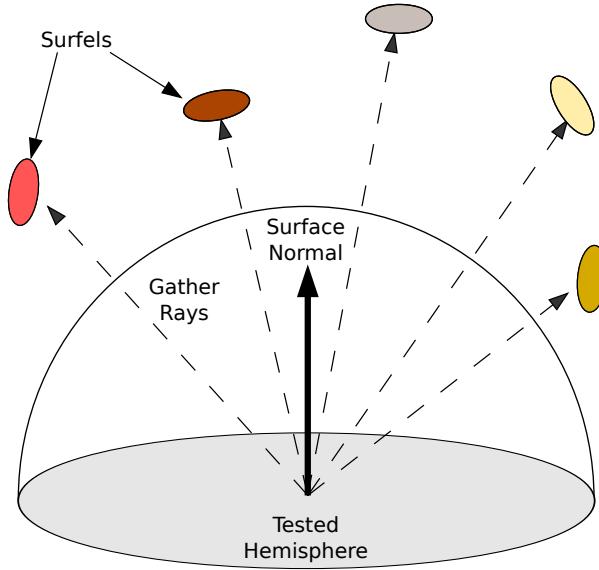


**Figure 4.3:** Basis vectors are generated based on the surface normal in order to transform samples on a hemisphere to test surrounding radiance.

#### 4.4.2 Hemisphere Sampling

In order to approximate the integral of incoming light at point  $p$  on the surface, we sample across a hemisphere oriented along the surface's normal  $N$  at  $p$  as seen in Figure 4.3. Each sample cast out from  $p$  evaluates  $L(p \leftarrow w)$  (as shown in Figure 4.4,) which is then multiplied by  $w \cdot N$  in order to represent  $\cos\theta$ . In order to obtain good results, 128-256 samples are typically necessary to combat noise caused by the samples. The resulting irradiance from the weighted sum of the samples is normalized by multiplying the *source term*  $\mathbb{S}$  (2.9) for the given phase function.

It is also necessary to consider the sampling method just as important as the evaluation of those samples. Generating purely random rays leads to clumping and high levels of noise, so a stratified sampling method was chosen, subdividing



**Figure 4.4:** Gather rays are cast into the point-cloud, returning the estimated radiance coming from a given direction. The radiance is then scaled based on the solid angle of that sample cast (based on sample count.)

the sample space equally into a two-dimensional grid and gittering within the grid. This helps us avoid clumping issues, and guarantees an even distribution over the entire domain. In order to map this two dimensional domain over our hemisphere, we chose the following mapping code:

```

1 Vec3 sampleToHCoord(float us, float ts) {
2     const float r = sqrt(1. - us);
3     const float theta = 2 * PI * ts;
4     const float x = r * cosf(theta);
5     const float y = r * sinf(theta);
6     return Vec3(x, y, sqrt(us));
7 }
```

Note that  $us$  represents the Z value of the hemispherical sample (with the

normal naturally placed down the Z plane.)  $\sqrt{us}$  represents a diminishing curve, allowing fewer samples near the top of the hemisphere, avoiding a common clumping behavior and keeping the distribution of samples even.

## 4.5 Integrating Volume Data

In order for *lvoxels* to contribute meaningfully to our scene during the light gather stage, we must make some architectural modifications to the algorithm in order to handle 1) more than one sample type in our octree and 2) the ability to handle non-opaque samples. Both of these required simple changes in the octree data-structure as well as modification of the traversal algorithm used.

### 4.5.1 Data-Structure Modifications

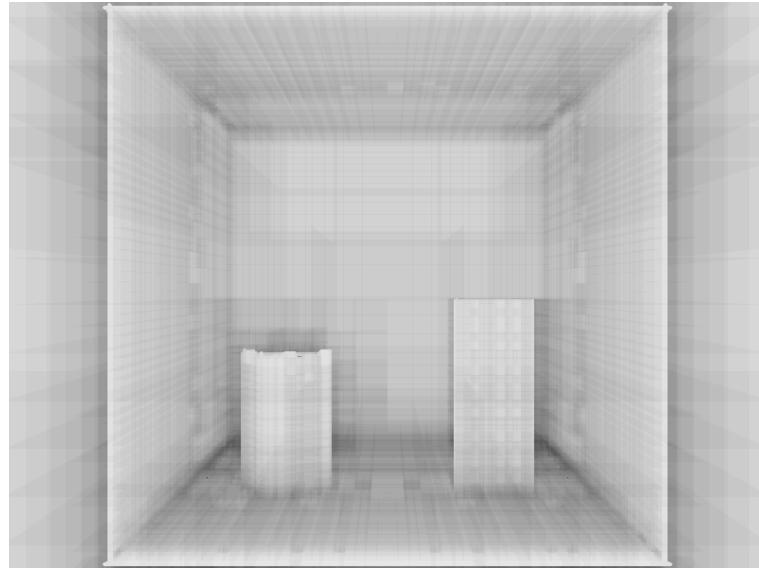
Modifications to the previously mentioned irradiance sampling technique in order to allow scatter-out effects with volumes are few. The biggest changes are to the point cloud octree and its traversal. Specifically, when computing lighting, we must account for the fact that when an element of the point cloud is hit, it may be transparent. In the standard algorithm, absorption and transmittance would not be taken into account and the traversal would stop at the first lvoxel encountered.

Therefore, our algorithm must fulfill the following requirements: 1) The algorithm must ensure that the *lvoxels* are placed in the same octree datastructure as the *surfels*, 2) Our algorithm must keep track of the current Transmittance in order to determine the contribution of all samples encountered and 3) We must traverse the leaf nodes of the scene from front-to-back in order to integrate

transparent sample contributions correctly.

### 4.5.2 Octree Traversal

In order to properly evaluate transparent and opaque surfaces within the point cloud, we made changes to node-level octree traversal. Each branch traverses its children from closest to farthest, guaranteeing that closer leaf nodes are evaluated first. Leaf nodes then use the pre-evaluated scatter ( $\sigma_s$ ) and absorption ( $\sigma_t$ ) coefficients for each lvoxel to appropriately alter the sample ray's transmittance, and continue with the traversal, with each hit contributing to the final resulting radiance value. Once a surfel is hit, there is no need to continue traversing the octree as seen in Figure 4.5.



**Figure 4.5:** Illustrates the octree traversal algorithm testing efficiency. Lighter shades represent less tests, while darker shades represents the most. Closer objects evaluate faster, and all data in the point-cloud behind them are occluded.

### 4.5.3 Acquiring Scatter-Out Contributions

Once the changes to the point-cloud data-structure have been made, we are able to 1) guarantee correct evaluation of the transparent surfaces through front-to-back octree traversal and 2) stop evaluating leaf nodes once we have hit an opaque surfel, reducing the overall sample count. Now that *lvoxels* are supported, we simply sample the scene as we have with regular PCB. The modified traversal algorithm already takes care of transmittance through any transparent media so no further changes are necessary.

### 4.5.4 Acquiring Scatter-In Contributions

After adding lvoxels to our octree structure and evaluation algorithm, the only modifications necessary for scatter-in are to the volume rendering equation. As an overview, volume integration will step through the volume, at each point it will:

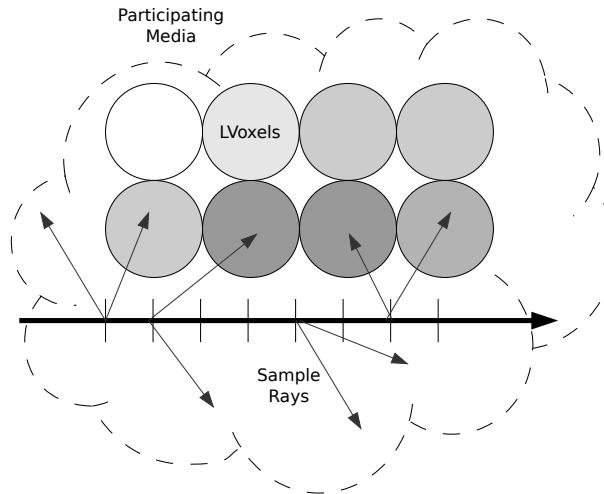
1. Update the current transmittance by the scatter and absorbtion terms at the given point
2. Cast shadow rays to estimate the direct light (and in cases of non-uniform scatter, apply a phase function)
3. Sample scatter contribution by sending rays out into the point cloud
4. Add direct illumination contribution and indirect illumination to the current incoming radiance

Recall that to model lighting for a volume, in-scattering requires integrating over all directions. Casting Monte Carlo sample rays through the volume and

into the scene would be computationally expensive. Instead, for each sample we send out rays into the point cloud, iterating through a much less dense dataset like in Figure 4.6.

This method helps us replace expensive  $S(p, w)$  evaluations with traversals into the octree. The two main differences between sampling scattered light within a volume and evaluating the irradiance on a surface are 1) the distribution function, which is based on the volume's phase function, and 2) the samples are distributed over a sphere rather than a hemisphere.

These scatter samples are distributed throughout the volume marching process typically taken while rendering volumes as seen in Figure 4.7. The ray generates a series of sample points through the volume and iterates through, gathering light from direct lighting and indirect sampling as well as updating the ray's transmittance (which identifies how much light passes through the volume from behind.)



**Figure 4.6:** Sample rays are cast during volume traversal, allowing for decent estimates of lighting contribution at each point.

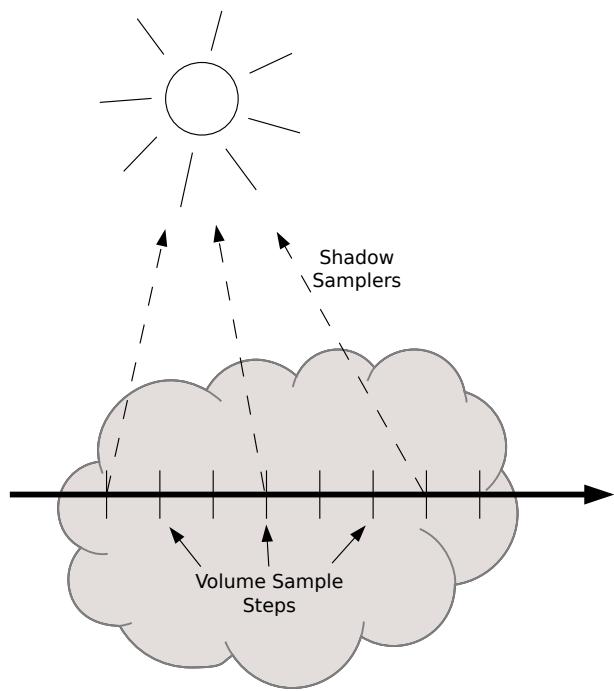
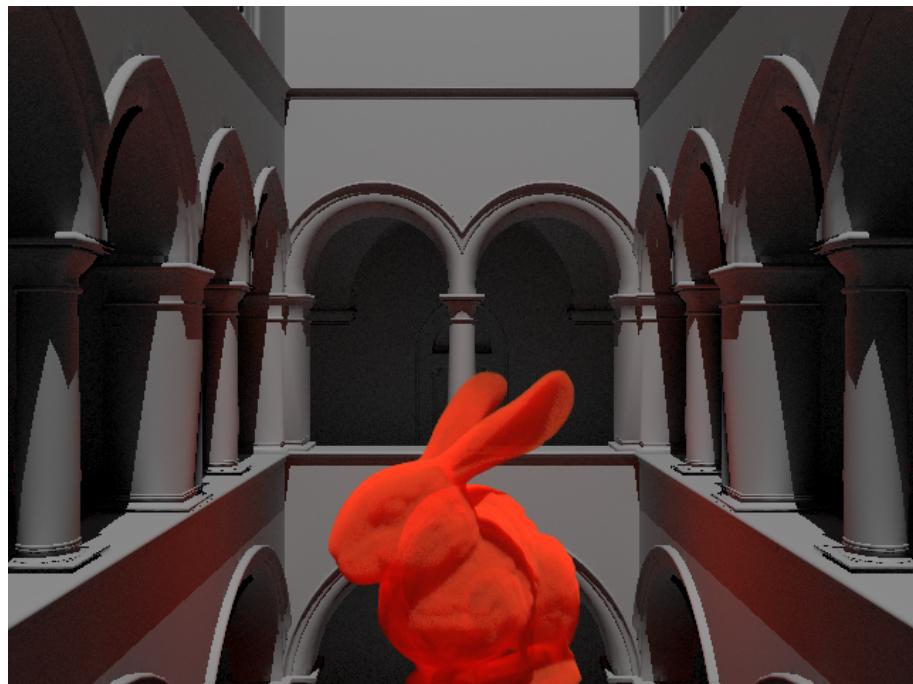


Figure 4.7: Stepping allows for estimation of the integral through the entire volume. Shadow rays are cast intermittently to estimate the direct lighting contribution.

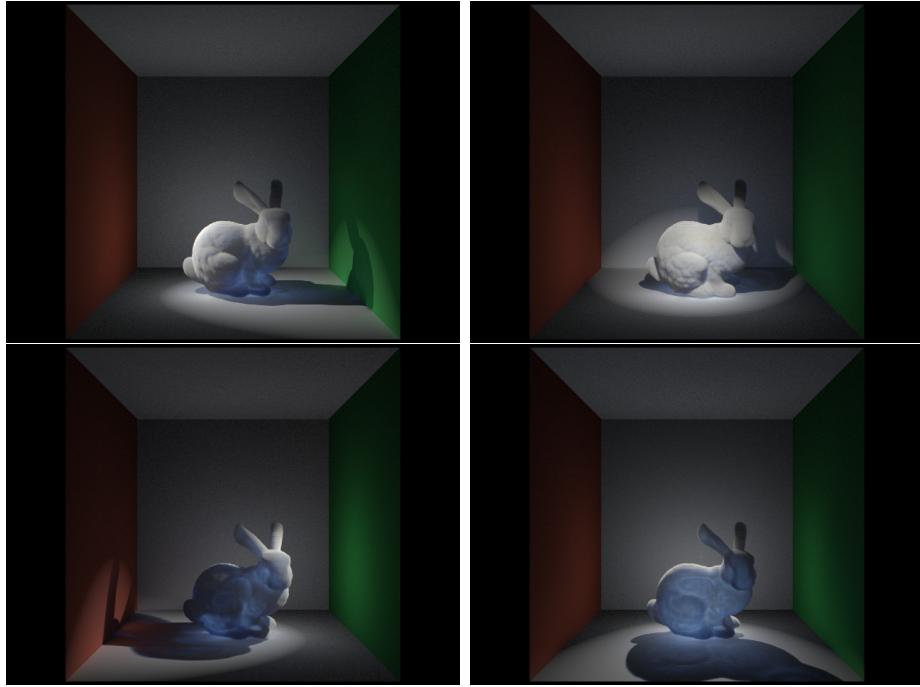
# Chapter 5

## Results

This section will discuss the testing environment and test scenario used to compare traditional Monte Carlo gather methods with the results that we were able to achieve using our PCB Extension algorithm.



**Figure 5.1:** Sponza Atrium mesh and Stanford bunny volume rendered using the PCB Extension algorithm.

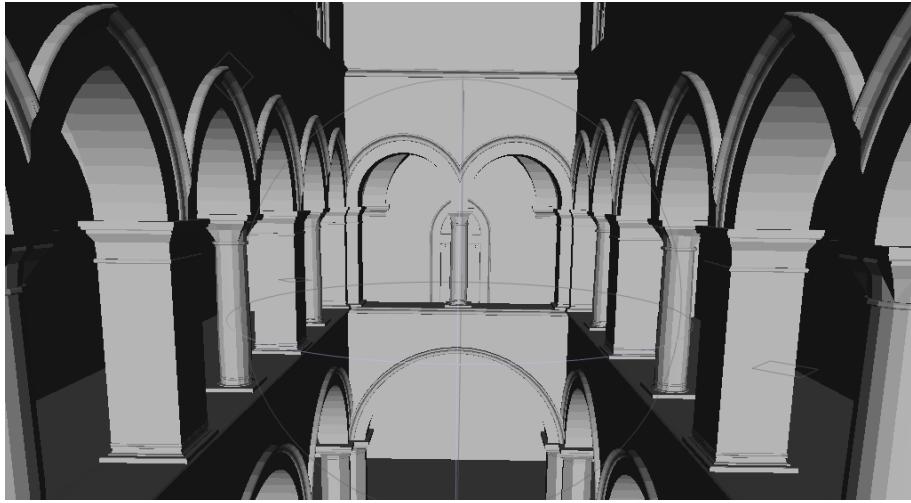


**Figure 5.2:** A test scene showing a light’s interaction with a volume changing depending on the direction and position of the light.

## 5.1 Environment

Our algorithm is able to achieve realistic lighting effects for scenes that include volumetric elements using our lvoxel representation with a point-based color bleeding approach to global illumination. The following test cases were run on a commodity-class Intel i5 3 GHz machine with 4 Gb of RAM. Because of the disparity between academic-level versus production-class ray tracer implementations, we tested and compared our results against a naive implementation of Monte Carlo global illumination not using the point cloud representation. We then compared the resulting images and the time it took to render each. Our algorithm is able to achieve a small difference between images and an increase in efficiency measured in time to render.

## 5.2 Test Scene



**Figure 5.3:** Real-time visualization of the Sponza Atrium mesh in MeshLab.

The scene tested involved a 60,000 triangle Sponza Atrium including only vertex and normal information for simplicity. The CT scan data of the Stanford Bunny was used in order to test scatter in/out contributions by complex participating media. Figure 1.2 shows the bunny and Sponza Atrium showing traditional Monte Carlo scattering. At first glance these two images are very similar, however there are a number of small artifacts present in the image rendered with the point cloud representation, and the indirect lighting is slightly darker overall. A closer look at the two results exemplifies the great similarity between the two images, as shown in Figure 5.4.

## 5.3 Data Comparison

Scene	Render Time (s)	Image Delta	Memory Overhead
Monte Carlo w/o PCB	3351 sec	NONE	NONE
Traditional PCB	???? sec	11.0%	390 Mb (4.0%)
Extended PCB	0397 sec	4.8%	395 Mb (4.1%)

## 5.4 Analysis

Our analysis involves comparing 1) the overall render time 2) the perceived image delta between the images and 3) the memory overhead used by the point-cloud data.

### 5.4.1 Memory

When using traditional PCB, the real benefit to its surfel representation is shown in more complex scenes. In the Sponza Atrium, the scene generated over 2.5 million surfels for a 60,000 triangle scene. Adding volume data to the scene does not add an objectionable amount of data to the point cloud, but for scenes with large volumes the costs could quickly add up without some form of multi-resolution light caching. In this regard, adding yet another representation of the volumes may be expensive, but not prohibitively so. Additionally, larger scenes would benefit from this representation, as it would be significantly simpler than the entire scene and can be moved to another system for out-of-core evaluation.

### 5.4.2 Speed

Even without volume integration, Monte Carlo integration without a lighting representation like PCB is prohibitively slow for even the simplest scenes. Adding a point cloud representation gave us an impressive speedup. That speedup was compounded even more when volume scattering was added into the tests, showing a factor of eight speedup for our test scenes.

Even on sparse octrees without volumes, our *front to back* octree traversal method operates at an efficiency of  $O \log n$  for each node traversal while skipping

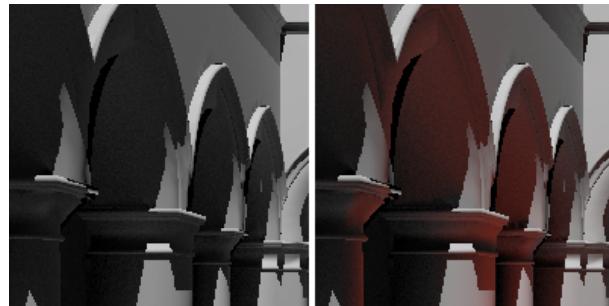
nodes occluded by surfels, leading to an average performance increase of over 18%.

## Image Quality

In order to objectively compare the pure Monte Carlo render to the PCB extension render, we used a perceptual image difference program called pdiff and ran the pair of images through in order to identify how close the two images were to each other. The results showed a clear difference (where pixels in one image clearly differend from another) of 4.8%.



**Figure 5.4:** Zoomed image showing PCB extension (left) and Monte Carlo (right.)



**Figure 5.5:** Zoomed image showing traditional PCB (left) and PCB with extension (right.) Note the visible color bleeding with our method.

Figure 5.5 compares the non-PCB Monte Carlo image with that of the traditional PCB renders, showing the clear lack of proper in-/out-scattering. With

the extended algorithm, however, the scenes look nearly identical.

We would like to note that there are a number of small artifacts in the PCB renders due to imprecision and incorrect surfel collisions. It is important to note that, as past papers will attest, such issues are easily overcome and our artifacts are more due to implementation and time constraints than limits on the algorithm itself.

## 5.5 Conclusion

In this paper, we discussed the necessity for proper global illumination approximations in renders, listed a number of algorithms that have attempted to do this but have fallen short specifically in volume scatter contributions, and presented an extension to the PCB algorithm by [2] which handles both scatter-in and scatter-out contributions. The addition of the lvoxel paradigm to the already successful point based color bleeding algorithm is shown to be a cost effective method of approximating and evaluating complex scatter functions based on participating media. The speedups are clear, and the memory footprint is easily manageable. The ability to evaluate the irradiance at a point in the scene by using only the point cloud representation is a clear win for out-of-core renderers.

Computer graphics, be it photo-realistic or artistically leaning, relies heavily on the paradigms established in the physics of light in the real world. Global illumination is just one of many areas of focus trying to better represent that light and its complex interactions in the abstract worlds we choose to bring to screen. One can only imagine the leaps and bounds that computer graphics is destined to experience in the following years, but inevitably the obstacles boil down to the same subset of problems. How to manipulate light and, by extension,

color in order to make the viewer experience a story or emotion.

# Chapter 6

## Future Work

As mentioned in Christensen’s point based color bleeding article, surfels can be modified to “gather” light recursively from their position in the point cloud, allowing for simulated multi-bounce lighting. This would require only a small change to the current algorithm, and would apply to volumes as well to allow very realistic scatter approximations in participating media.

In our tests, all participating media scatters light equally in all directions. This is rarely the case, as volumes tend to have unique scatter functions. We can simulate more complex surface scattering functions by creating spherical harmonic representations of the radiance at any specific point in the volume. Our current implementation supports such an approach, but remains untested.

Typical implementations of the PCB algorithm include rougher estimations (usually in the form of a series of spherical harmonic coefficients) at higher levels in the octree, to be evaluated depending on that node’s solid angle to our sample point. Due to time constraints, we did not implement full multi-resolution repre-

sentations of each node. Including LVoxel data in that representation would be a trivial process.

Our ray tracer runs a number of threads to split the image into multiple parts in order to achieve simple parallelism. Before the threads are created, however, we generate surfels and lvoxels sequentially. Due to the nature of our octree implementation, we cannot add elements and still be thread safe, but this would not be a large obstacle. Scenes like the sponza atrium would run a number of times faster if we were to parallelize our implementation more effectively.

Although for our purposes raycasting the scene to sample for surfels worked well, it is most certainly not an optimal algorithm. Sampling the scene in a more geometry-aware fashion would lead to fewer samples and better results. Subdividing the scene into smaller polygons and sampling the scene (say, one or two surfels per micro-polygon) would increase our coverage while decreasing unnecessary overlap. We found that the simpler approach worked best for us, given our time frame, however it can most certainly be improved.

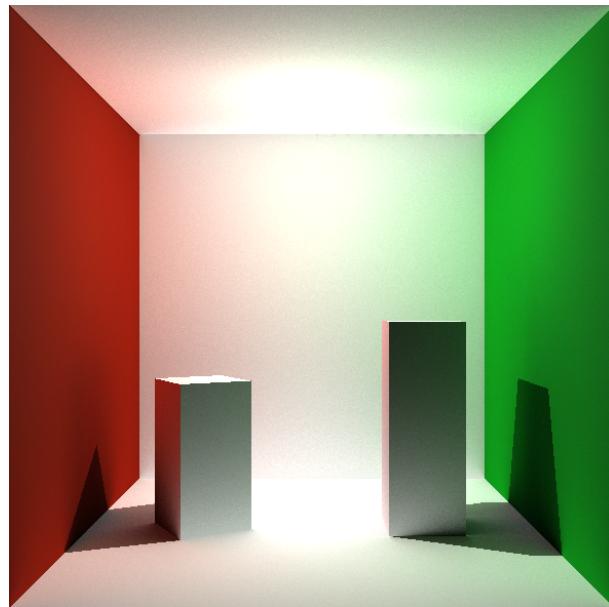
Because the color-bleeding effect in PCB focuses primarily on the point-cloud data, we are offered a unique opportunity to consider offloading the entire octree structure out-of-core in order to outsource the (still computationally complex) algorithm onto other machines, if not to on-board GPUs. Taking advantage of a graphics processor’s fast math and hardware rasterization would allow for much faster indirect-lighting evaluations.

# Bibliography

- [1] Eva Cerezo, Frederic Perez, Xavier Pueyo, Francisco J. Seron, and Francois X. Sillion. A survey on participating media rendering techniques. 21:303–328, 2005.
- [2] Per H. Christensen. Point-based approximate color bleeding. 2008.
- [3] Per H. Christensen. Point-based global illumination for movie production. SIGGRAPH '10, 2010.
- [4] Julie Dorsey. Radiosity and global illumination. *The Visual Computer*, 11:397–398, 1995.
- [5] S Guthe and W Strasser. Advanced techniques for hhigh-quality multi-resolution volume rendering. 28:51–58, 2004.
- [6] Henrik Wann Jensen. *Realistic Image Synthesis Using Photon Mapping*. A. K. Peters, Ltd., Natick, MA, USA, 2009.
- [7] Henrik Wann Jensen and Per 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, 1998.

- [8] James T. Kajiya and Brian P. Von Herzen. Ray tracing volume densities, 1984.
- [9] Joe Kniss, Charles Hansen, Peter Shirley, and Allen Mcpherson. A model for volume lighting and modeling. *IEEE Transactions on Visualization and Computer Graphics*, 9:150–162, 2003.
- [10] Marc Levoy. Display of surfaces from volume data, 1988.
- [11] Marc Levoy. Efficient ray tracing of volume data. *ACM Transactions on Graphics*, 9:245–261, 1990.
- [12] Greg Humphreys Matt Pharr. *Physically Based Rendering: From Theory to Implementation*. Morgan Kaufmann, 2 edition, 2010.
- [13] Philippe Bekaert Philip Dutre, Kavita Bala. *Advanced Global Illumination*. A K Peters, 2006.
- [14] Eric Tabellion and Arnauld Lamorlette. An approximate global illumination system for computer generated films. 23:469–476, 2004.
- [15] James M. Van Verth and Lars M. Bishop. *Essential Mathematics for Games and Interactive Applications*. Morgan Kaufmann, Burlington, MA, USA, 2008.
- [16] Rdiger Westermann. A multiresolution framework for volume rendering. In *SYMPORIUM ON VOLUME VISUALIZATION*, pages 51–58. ACM Press, 1994.
- [17] Magnus Wrenninge and Nafees Bin Zafar. Volumetric methods in visual effects. 2010.

- [18] R Xu and S.N Pattanaik. *A Novel Monte Carlo Noise Reduction Operator*, volume 25. Computer Graphics and Applications, IEEE, 2005.
- [19] Caixia Zhang and et al. light propagation for mixed polygonal and volumetric data, 2005.



**Figure 6.1:** Example of point-based color bleeding without the volume extension algorithm.

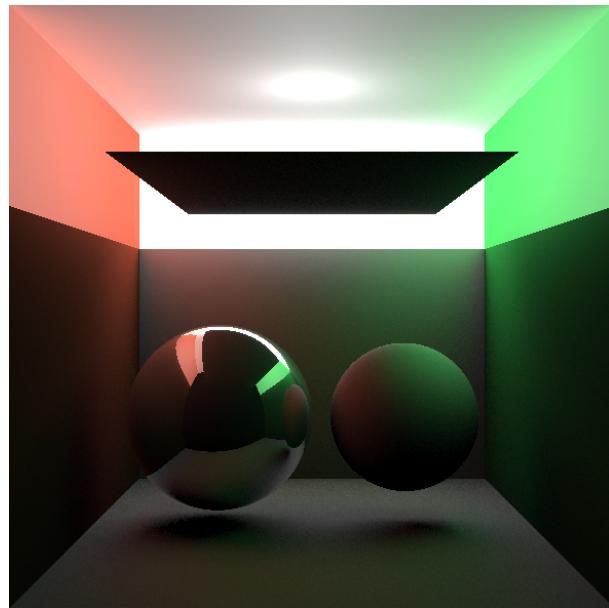


Figure 6.2: Example of a scene almost entirely in shadow, showing indirect lighting in play.



Figure 6.3: The Sponza Atrium with the Stanford volumetric bunny. In-/Out-scattering is evident on the volume and on the surrounding atrium walls.



Figure 6.4: Image exemplifying clear out-scattering from Stanford bunny volume.

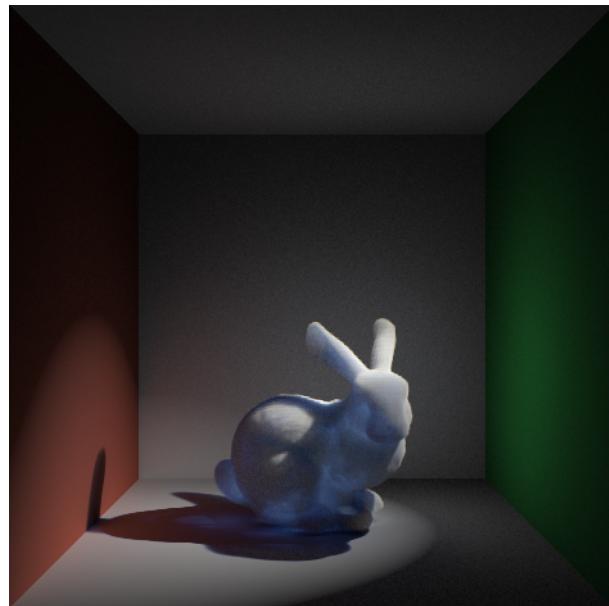


Figure 6.5: Image exemplifying clear color bleeding next to the red wall in the bunny's shadow and correct transmittance through the bunny's hollow form.

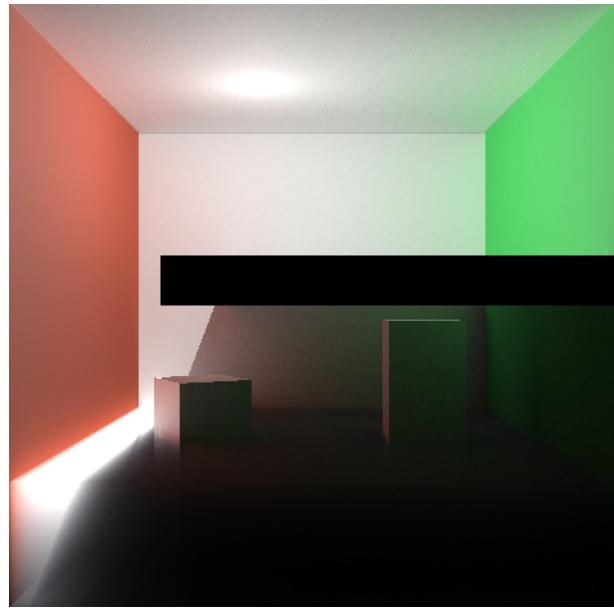


Figure 6.6: The black occluding geometry in the center stops all but the light to the left to enter below. In and out scattering on the volume and walls is evident.

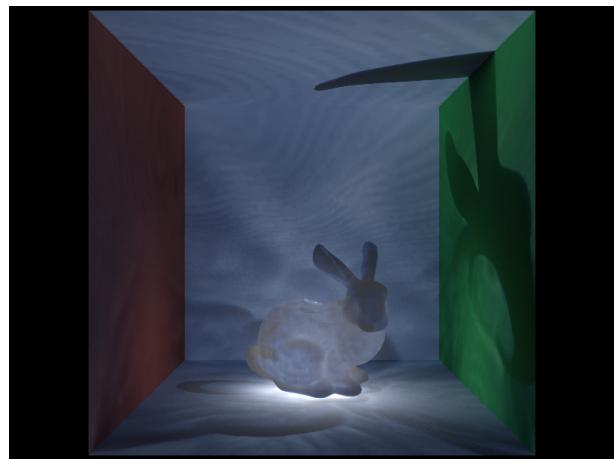


Figure 6.7: Illustrates how a light may act when placed within a hollow volumetric object. The bunny is shown as slightly brighter, scattering light about the scene (especially underneath it, where there are holes in the volume.)



**Figure 6.8:** Shows how CT scan data can be used to visualize scanned objects like a human face. Subsurface scattering and transmittance through thin materials is evident.