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

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

Achieving realistic or believable global illumination in scenes with participating media is expensive. Light interacts with the particles of a volume, creating complex radiance patterns. This paper introduces an extension to the commonly used point-based color bleeding technique which allows fast, believable in- and out-scattering 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.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The ability to render scenes with realistic lighting is desirable for many entertainment application settings such as film. Great results have been achieved for large scenes using point based color bleeding [2] algorithms. These algorithms, however, tend to limit or omit entirely the lighting contribution from volumetric data or participating media within the scenes. 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.

The proposed method achieves results comparable to those produced with Monte Carlo ray tracing but with drastically reduced run times, speeding up renders by around 10 times. Figure 1.1 illustrates a comparison of our algorithm and Monte Carlo ray traced results.

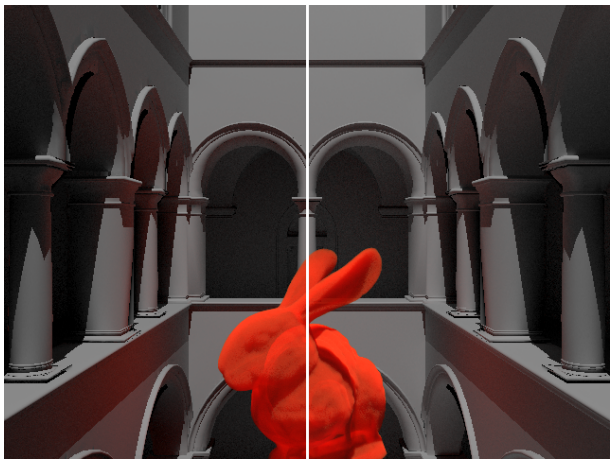

Figure 1.1: Comparison of the PCB extension (left) and traditional Monte Carlo results (right.)

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 [11]).

2.1 Radiance

Irradiance is the change of flux (radiant power) over an area, denoted by $E = \frac{d\Phi}{dA}$ [12]. Another way to look at this problem involves the relationsip 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 $L(\mathrm{p}, w)$ given p is the point on a surface and

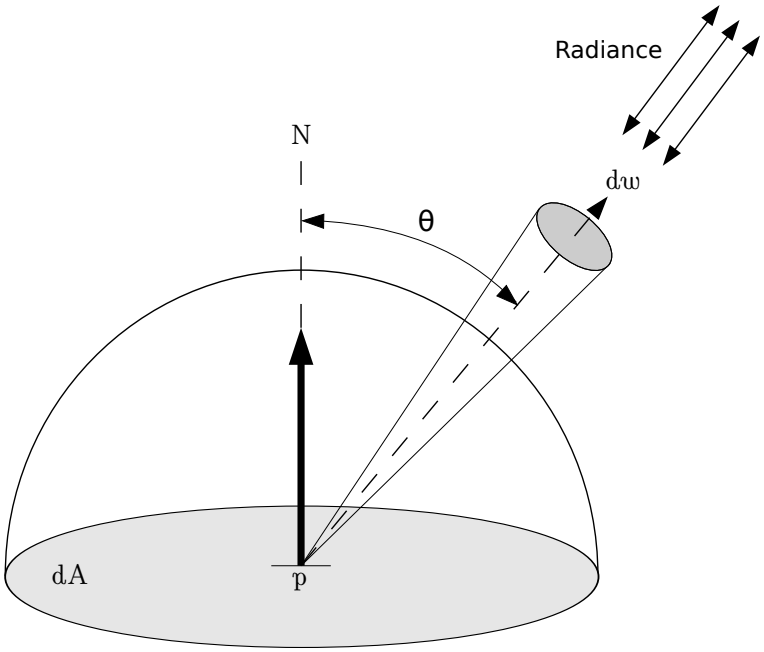

Figure 2.1: Evaluation of radiance at a point on an opaque surface. Only the hemisphere around the surface normal is considered. Incoming radiance is measured and scaled by its solid angle.

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:

$$L = \frac{d^2\,\Phi}{dwdA^{\perp}} \tag{2.1}$$

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:

$$E = \int L(\mathrm{p} \leftarrow w)\cos\theta dw. \tag{2.2}$$

$L(\text{p} \rightarrow w)$ represents the radiance leaving point p and $L(\text{p} \leftarrow w)$ represents incoming radiance given a direction w. Note that radiance along a straight path is invariant. For example:

$$L(x \rightarrow y) = L(y \rightarrow x). \tag{2.3}$$

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 Volume Lighting

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/extintion (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.

2.2.1 Absorption:

As light passes through a participating media, light will become absorbed based on its absorption probability density σ_a. As stated earlier in Equation 2.3, it is known that radiance along a straight path is invariant, which allows us to

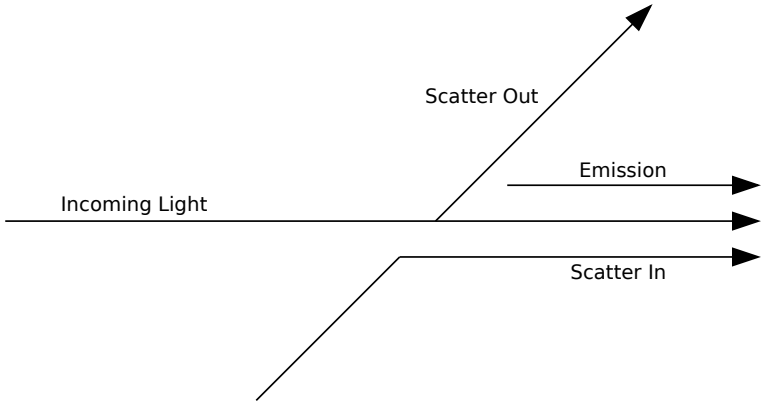

Figure 2.2: Light scatter properties vary based on the participating media.

estimate the amount of light absorbed or scattered given point p and direction w with the following:

$$e^{-\int_0^d \sigma_a(p+tw,w)dt}, \tag{2.4}$$

where σ_a represents the probability density that light will be absorbed over a distance $\mathrm{d}t$.

2.2.2 Scatter Out:

In addition to being absorbed by the medium, light can be scattered based on a scatter probability density σ_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:

$$\mathrm{d}L_o(\mathrm{p}, w) = -\sigma_s(\mathrm{p}, w)L_i(\mathrm{p}, -w)\mathrm{d}t. \tag{2.5}$$

2.2.3 Transmittance:

Both (2.4) and (2.5) 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:

$$\sigma_t(\mathrm{p}, w) = \sigma_a(\mathrm{p}, w) + \sigma_s(\mathrm{p}, w). \tag{2.6}$$

Then, integrating the transmittion of the volume from (2.6) over a ray gives us the following:

$$T_r(\mathrm{p} \to \mathrm{p}') = e^{-\int_0^d \sigma(p+tw,w)dt}. \tag{2.7}$$

2.2.4 Phase Functions

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 $phase(w \to w')$. The probability that light may scatter from direction w to w' is described using this function. This distribution is visualized in Figure 4.1. 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].

2.2.5 Scatter In

Although σ_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

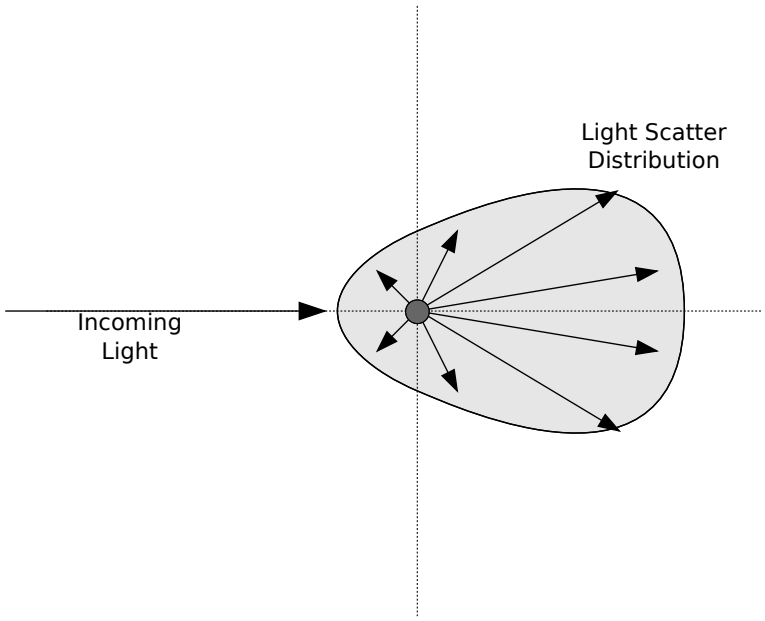

Figure 2.3: Visual representation of a phase function around a scatter point. The area represents the distribution of the scattered light about a sphere.

Figure 4.1) may contribute to the original ray's radiance. Before we can integrate incoming radiance, we must take into account the *source term* $\mathbb{S}$, or total added radiance per unit distance, where the following constraint must hold true:

$$\int_{\mathbb{S}^2} phase(w \to w')\mathrm{d}w' = 1. \tag{2.8}$$

This normalization makes sure that the phase function actually defines the probability distribution for a particular direction.

Finally, we can integrate the total radiance scatter based on the normalized phase function $phase(w \to w')$ over all directions w' to get our total scatter in a direction w:

$$S(\mathrm{p}, w) = L_{\mathrm{ve}}(\mathrm{p}, w) + \sigma_{\mathrm{s}}(\mathrm{p}, w) \int_{\mathbb{S}^2} phase(\mathrm{p}, -w' \to w) L_i(\mathrm{p}, w')\mathrm{d}w'.$$

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

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 [16] which 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. Of particular relevance to this work is that done in Point-Based Approximate Color Bleeding [2], which describes the process of sampling the scene in order to create a point cloud representation, 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.

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 rendering of scenes which contain volume data. A number of approaches have been developed in order to render volume data [9][8]. Volume data representations often include an efficient multi-resolution data representation [14][10]. When dealing with multi-resolution volume octree datastructures, removing occluded nodes from being tested can drastically increase performance [5]. Our algorithm 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.

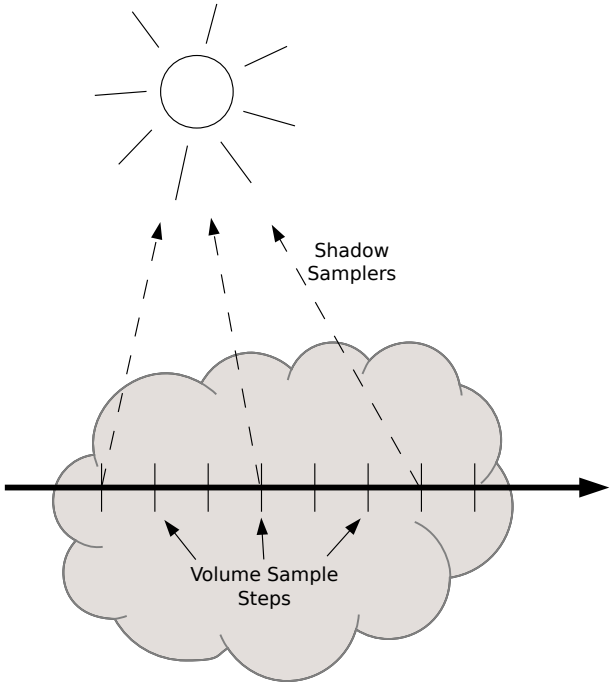

Figure 3.1: Stepping allows for estimation of the integral through the entire volume. Shadow rays are cast intermittently to estimate the direct lighting contribution.

Chapter 4

Algorithm

We present an algorithm which is an extension to the point cloud techniques described in [13] 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.

In general, these methods subdivide the world into small representational segments, called surfels in [2], which are stored in a large point cloud, representing the scene. 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.

The goal of our proposed method is to include volumetric representations into a global illumination algorithm in a fast and coherent way. 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*.

In general, our algorithm must 1) sample the scene geometry (including the participating media within the scene) and store the direct lighting (or relevant lighting properties) within an acceleration structure for fast evaluation, 2) gather indirect lighting during regular ray casts using scene geometry and 3) model the scatter-out and scatter-in properties of volumetric lighting during the indirect lighting gather stage.

4.1 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.

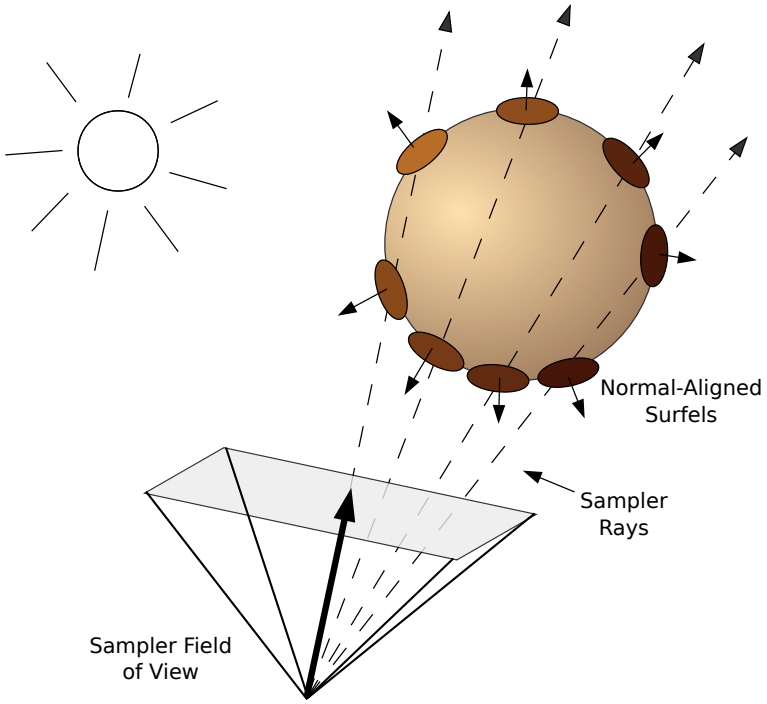

Figure 4.1: Rays are cast from a special camera during the surfel sample phase. Each time the ray intersects with geometry a surfel is created.

We sample the opaque geometry in surfels, which are computed using a perspective viewing volume slightly larger then the current viewing frustum, with a sampling rate two times that of the desired pixel resolution. 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 (2.7) to each light source saves us from re-computing light calculations during sampling in sections 4.3.2 and 4.3.3 [15].

4.2 Gathering Light

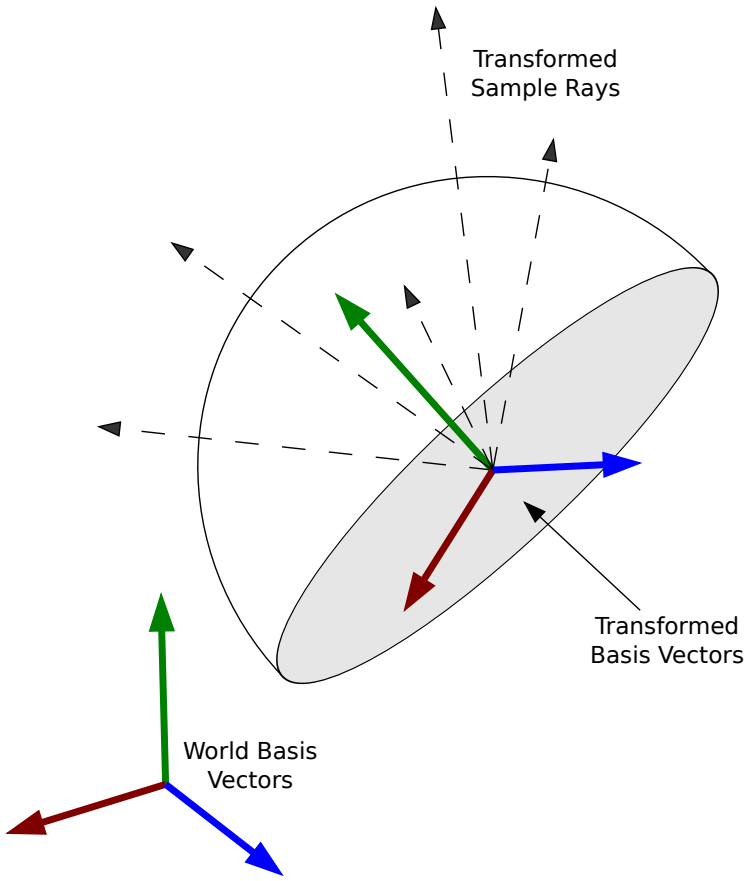

Figure 4.2: Basis vectors are generated based on the surface normal in order to transform samples on a hemisphere to test surrounding radiance.

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 as seen in Figure 4.2. 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.

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. Each sample cast out from p evaluates $L(\text{p} \leftarrow w)$ (as shown in Figure 4.3,) 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* $\mathbb{S}$ (2.8) for the given phase function.

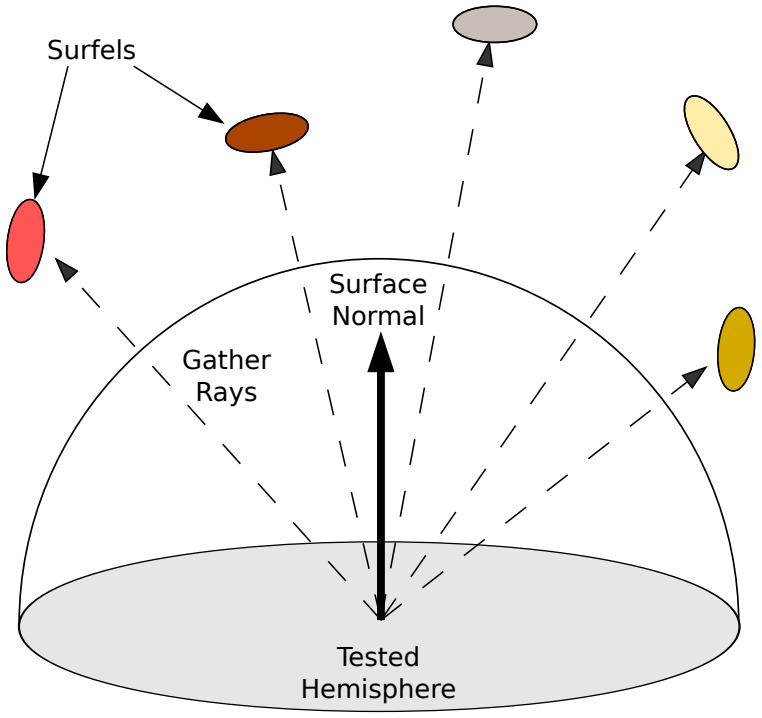

Figure 4.3: Gather rays are cast into the point-cloud, returning the estimated radiance coming from a given direction. The radiance is then scaled based on the solid angle of that sample cast (based on sample count.)

4.3 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.

4.3.1 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.

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 (σ_s) and absorption (σ_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 4.4.

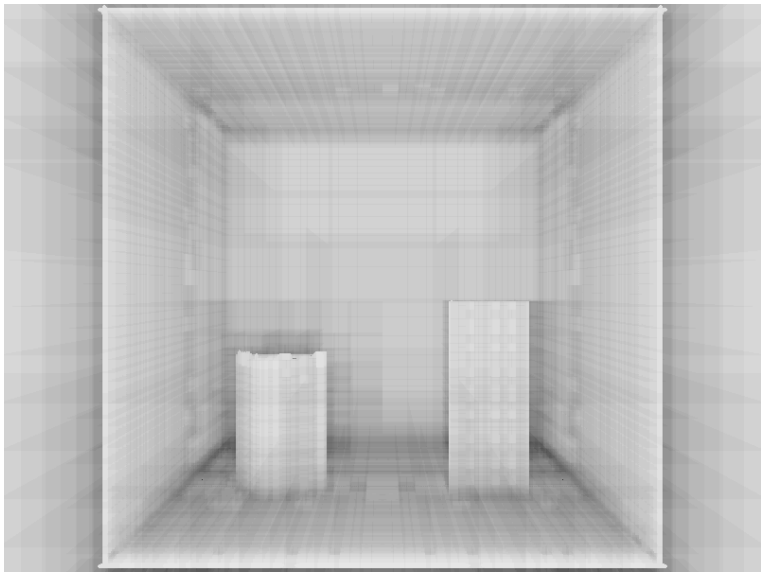

Figure 4.4: Illustrates the octree traversal algorithm testing efficiency. White represents no tests, while black represents the most. Closer objects evaluate faster, and all data in the point-cloud behind them are occluded.

4.3.2 Adding Scatter-Out

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.

4.3.3 Adding Scatter-In

After adding lvoxels to our octree structure and evaluation algorithm, the only modifications necessary for scatter-in are to the volume rendering equation.

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 4.5. This helps us replace expensive $S(\mathrm{p}, \mathrm{w})$ 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.

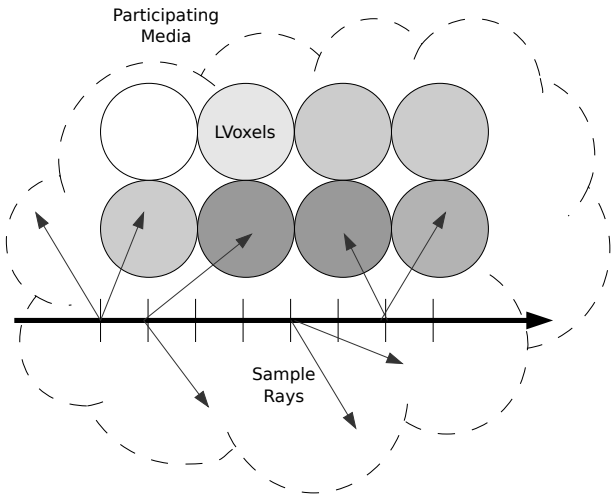

Figure 4.5: Sample rays are cast during volume traversal, allowing for decent estimates of lighting contribution at each point.

Chapter 5

Results

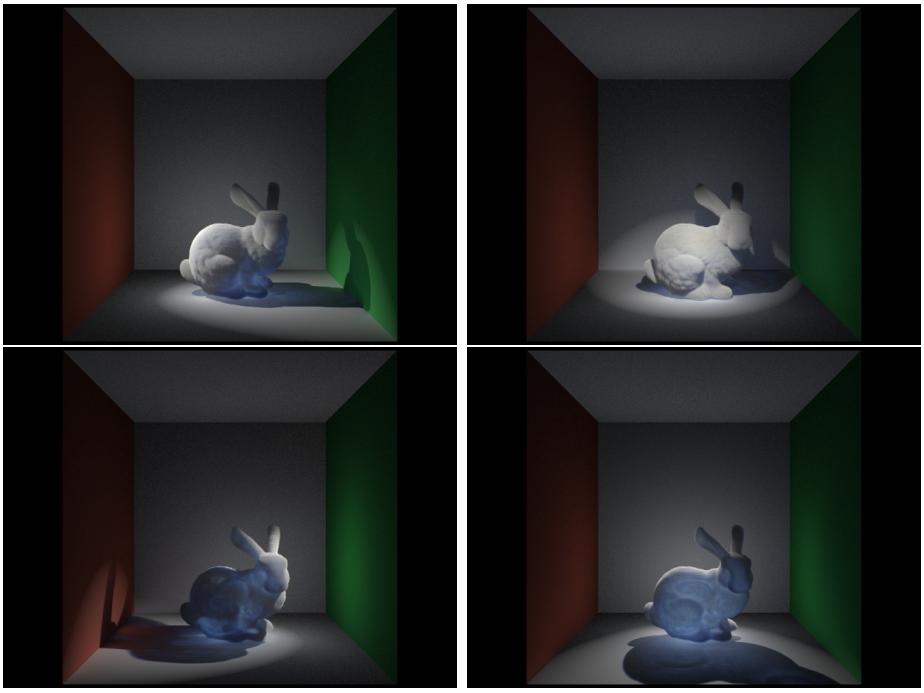

Figure 5.1: A test scene showing a light's interaction with a volume changing depending on the direction and position of the light.

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.

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 1.1 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 5.3.

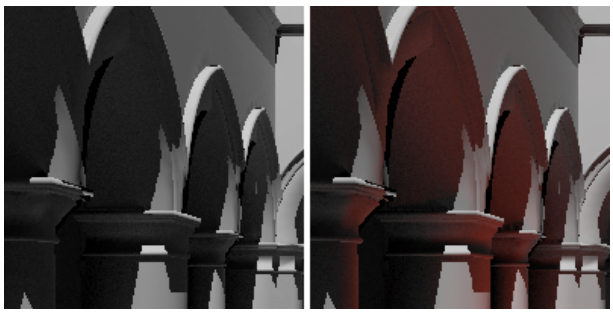

Figure 5.2: Zoomed image showing traditional PCB (left) and PCB with extension (right.) Note the visible color bleeding with our method.

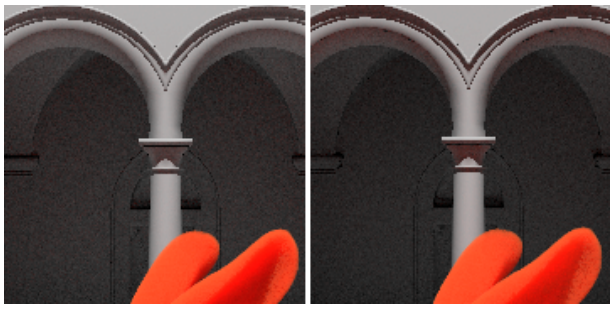

Figure 5.3: Zoomed image showing PCB extension (left) and Monte Carlo (right.)

5.1 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%)

5.2 Analysis

5.2.1 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.

5.2.2 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

Figure 5.2 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.

5.3 Conclusion

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.

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 representations 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.

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] Marc Levoy. Display of surfaces from volume data, 1988.

[10] Marc Levoy. Efficient ray tracing of volume data. *ACM Transactions on Graphics*, 9:245–261, 1990.

[11] Greg Humphreys Matt Pharr. *Physically Based Rendering: From Theory to Implementation*. Morgan Kaufmann, 2 edition, 2010.

[12] Philippe Bekaert Philip Dutre, Kavita Bala. *Advanced Global Illumination*. A K Peters, 2006.

[13] Eric Tabellion and Arnauld Lamorlette. An approximate global illumination system for computer generated films. 23:469–476, 2004.

[14] Rdiger Westermann. A multiresolution framework for volume rendering. In *SYMPOSIUM ON VOLUME VISUALIZATION*, pages 51–58. ACM Press, 1994.

[15] Magnus Wrenninge and Nafees Bin Zafar. Volumetric methods in visual effects. 2010.

[16] R Xu and S.N Pattanaik. *A Novel Monte Carlo Noise Reduction Operator*, volume 25. Computer Graphics and Applications, IEEE, 2005.

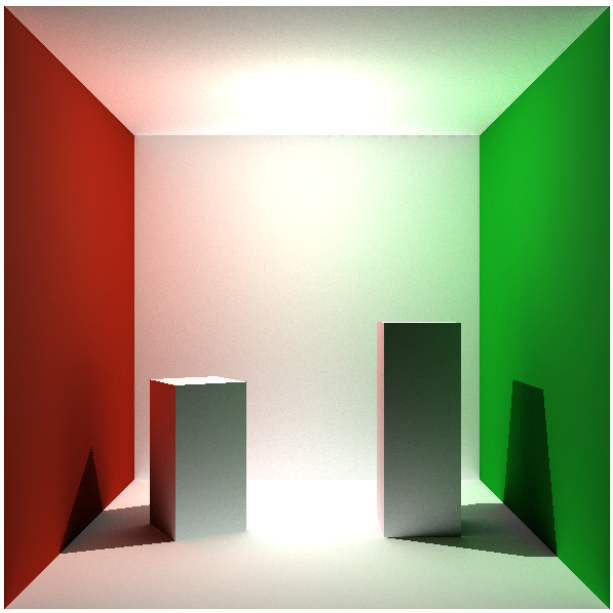

Figure 6.1: Image exemplifying clear out-scattering from Stanford bunny volume.

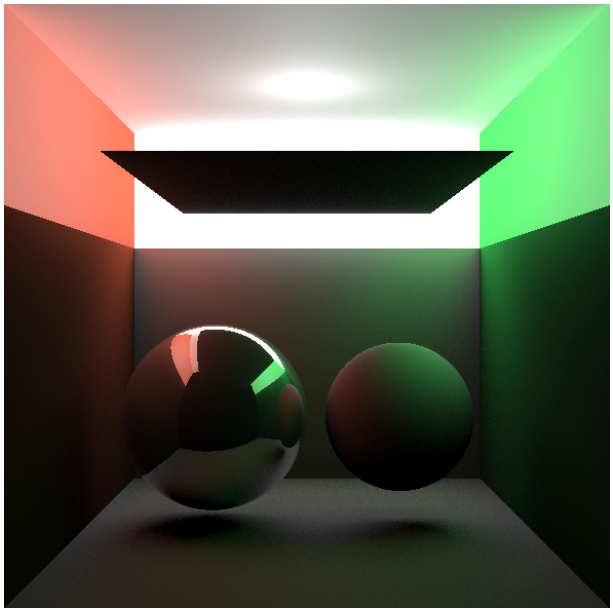

Figure 6.2: Image exemplifying clear out-scattering from Stanford bunny volume.

Figure 6.3: Image exemplifying clear out-scattering from Stanford bunny volume.

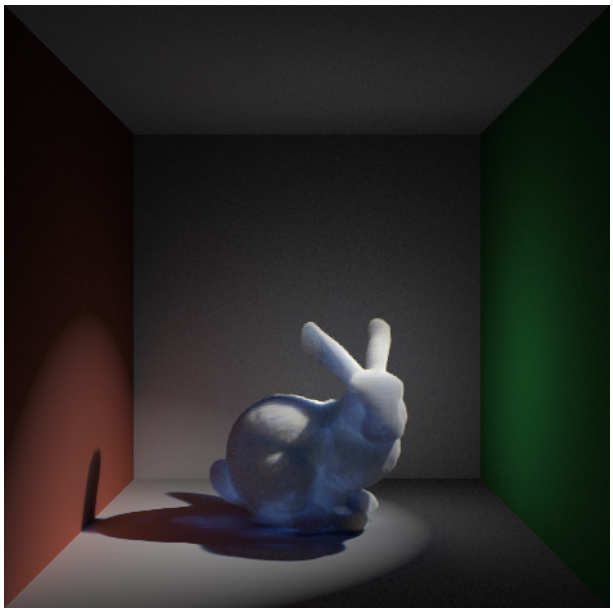

Figure 6.4: Image exemplifying clear color bleeding next to the red wall in the bunny's shadow and correct transmittance through the bunny's hollow form.

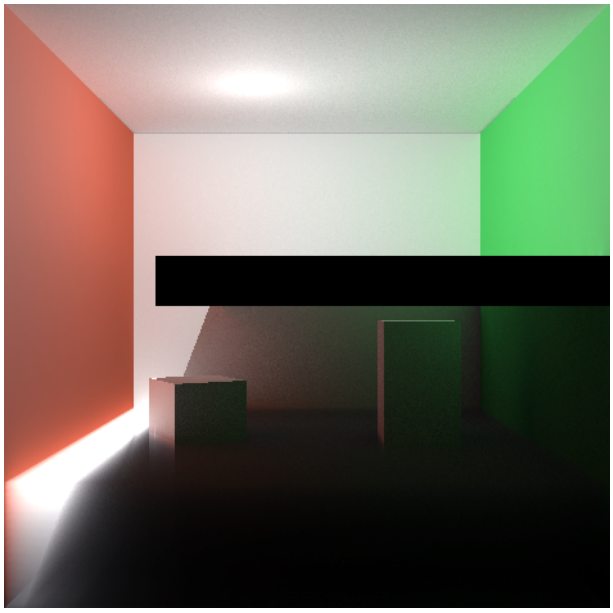

Figure 6.5: The black occluding geometry in the center stops all but the light to the left to enter below. In and out scattering on the volume and walls is evident.