An Approach to Point Based Approximate Color Bleeding with Volumes

Christopher J. Gibson and Zoë J. Wood

Abstract. 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 explicit extension to the commonly used point-based color bleeding technique which allows fast, believable in- and out-scattering effects building on existing data structures and paradigms. The proposed method achieves results comparable to that of existing Monte Carlo integration methods, that is realistic looking renders of scenes which include volume data elements, obtaining render speeds between 10 and 36 times faster while keeping memory overhead under 5%.

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 [1] algorithms. In their basic form, these algorithms tend to limit or omit the lighting contribution from volumetric data or participating media within the scenes. This paper presents an explicit extension to the point-based color bleeding (PCB) algorithm and data representation tuned to volumes, *light-voxel (or lvoxel)* to address the need to represent participating media which leverages a point cloud representation of a scene.

The proposed method achieves results comparable to those produced with Monte Carlo ray tracing [2], that is, images that include color bleeding from volume elements but with drastically reduced run times, speeding up renders by around 10 to 36 times. Figure 3 illustrates a comparison of our algorithm and Monte Carlo ray traced results.

1.1 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 [3]).

In contrast to polygonal models, in volumes, as light passes through a participating media, light will become absorbed based on its absorption probability density σ_a, scatter based on a scatter probability density σ_s or simply pass through the volume. Both *absorption* and *scatter-out* involve the reduction of energy through a volume, lowering its transmittance. This can be combined into the following representation: $\sigma_t(\mathrm{p}, w) = \sigma_a(\mathrm{p}, w) + \sigma_s(\mathrm{p}, w)$. Then, integrating the transmittion of the volume over a ray gives us equation (1.1): $T_r(\mathrm{p} \rightarrow \mathrm{p}') = e^{-\int_0^d \sigma_t(p+tw,w)dt}$.

Additionally, in volumes, the probability that light may scatter from direction w to w' is described using a distribution function or *phase function*, which describes the angular distribution of light scattered by particles in a volume. 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 [4].

Finally, while σ_s may reduce the energy of a ray passing through a volume, radiance from other rays may contribute to the original ray's radiance, called *scatter-in*. After we guarantee that the distribution is normalized, to make sure that the phase function actually defines a proper probability distribution for a particular direction, we can integrate the total radiance scatter based on the normalized phase function $phase(w \rightarrow w')$ over all directions w' to get our total scatter in a direction w: $S(\mathrm{p}, w) = L_{\mathrm{ve}}(\mathrm{p}, w) + \sigma_{\mathrm{S}}(\mathrm{p}, w) \int_{\mathbb{S}^2} phase(\mathrm{p}, -w' \rightarrow w) L_i(\mathrm{p}, w') \mathrm{d}w'$. The volume emission coefficient, $L_{\mathrm{ve}}(\mathrm{p}, w)$, is not discussed here.

2 Related Work

Global Illumination: Global illumination is an important problem in computer graphics, with numerous successful algorithmic solutions, including, photon mapping [5], radiosity [6] and Monte Carlo sampling techniques for ray tracing [2]. The most closely related methods to this work are those that sample the scene and use a two stage approach to model direct and indirect illumination. Of particular relevance is Point-Based Approximate Color Bleeding [1], 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 [7], but without algorithmic details, such as 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 volume representation in 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. Jensen in [8] 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.

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][10]. Volume data representations often include an efficient multi-resolution data representation [11][12]. When dealing with multi-resolution volume octree datastructures, removing occluded nodes from being tested can drastically increase performance [13]. 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. Although the idea of using these techniques in volumes is not new, utilizing them to guarentee correct volume integration in the point cloud used in PCB is.

3 Algorithm

We present an algorithm which is an extension to the point cloud techniques described in [14] and [1], specifically building off the point-based color bleeding 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, PCB methods subdivide the world into small representational segments, called surfels [1], 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 method is to include volumetric representations into PCB methods in a fast and coherent way while keeping memory overhead and computational complexity to a minimum. In the existing algorithm [1], 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. 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 volume) and store the direct lighting (or relevant lighting properties) 2) gather indirect lighting and 3) model the scatter-out and scatter-in properties of volumetric lighting.

3.1 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.

We sample the opaque geometry as 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. Rays are cast from this logical camera just as we would ray trace a normal scene, with surfels generated where those rays intersect with the scene. 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. In our tests, one lvoxel for every $2^3 - 4^3$ voxels achieved good results.

Caching the direct light contribution at each lvoxel by testing the transmittance (equation 1.1) to each light source saves us from re-computing light calculations during sampling in sections 3.3 and 3.4 [15].

3.2 Gathering Light

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. This decision 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.

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 $L(\mathrm{p} \leftarrow w)$ 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. How radiance and transmittance are handled with lvoxels will be explained in the next section.

3.3 Adding Scatter-Out

Modifications to the previously mentioned irradiance sampling technique for scatter-out effects with volumes are few. The most significant 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.

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.

In addition to traversing the tree front-to-back, we also keep track of the incoming radiance and current transmittance. Both of these values are modified according to the equations described in Section 1.1 taking into account each lvoxel's scatter and absorption coefficients.

3.4 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 sparser dataset. 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. Each of these samples gather light as described in Section 3.2.

4 Results

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. All 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 render times of each. Our algorithm is able to achieve a very small image difference and an increase in render time efficiency.

The scene tested involved a 60,000 triangle Sponza Atrium model including only vertex and normal information for simplicity. CT scan datasets of a human

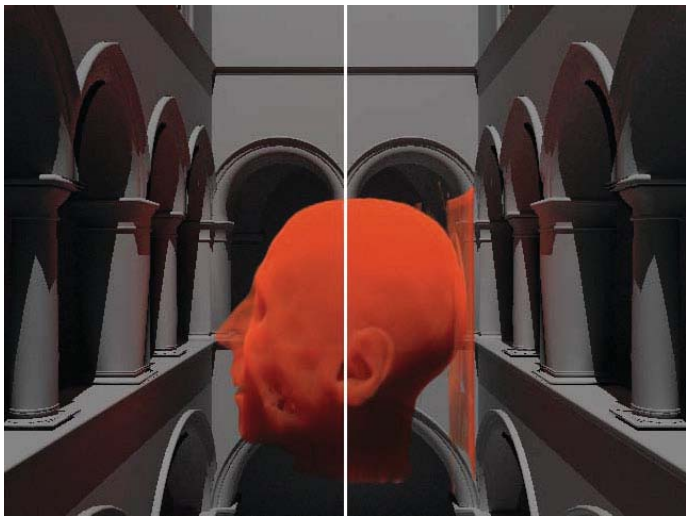
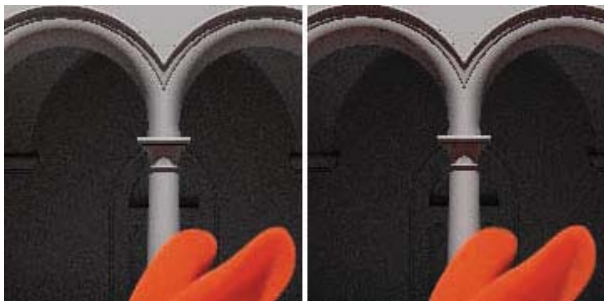


Fig. 1. CT Head scene comparison of the PCB extension (left) and traditional Monte Carlo results (right)

(a)



(b)

Fig. 2. (a) Zoomed image showing traditional PCB (left) and PCB with extension (right.) and (b) Zoomed image showing PCB extension (left) and Monte Carlo (right.)

head and the Stanford Bunny were used in order to test scatter in/out contributions involving complex participating media. Both Figures 3 and 1 show the CT data in Sponza Atrium rendered with traditional Monte Carlo scattering and our extended PCB. These resulting renders are very similar, and a closer look in Figure 2(b), exemplifies the great similarity between the two images. However, there are some small artifacts present in the image rendered with the point cloud representation, and the indirect lighting is slightly darker overall. Further results that show proper color bleeding can be seen in Figure 4. Close up comparison of traditional PCB versus extended PCB is shown in Figure 2(b).

4.1 Data Comparison

Table 1 shows render times, image differences and memory overhead for the CT scan datasets in the Sponza Atrium with varying sized volumes and for three different rendering methods: Monte Carlo without PCB, traditional PCB and finally our extended PCB algorithm.

Table 1. Data for the Sponza Scene with the two different CT datasets

Scene: CT Bunny in Sponza	Render Time (s)	Image Delta	Memory Overhead
64^3 resolution volume			
Monte Carlo w/o PCB	3229 sec	NONE	NONE
Traditional PCB	348 sec	5.8%	466.3 MB (4.780%)
Extended PCB	433 sec	2.1%	466.7 MB (4.786%)
128^3 resolution volume			
Monte Carlo w/o PCB	3297 sec	NONE	NONE
Traditional PCB	348 sec	5.6%	466.3 MB (4.780%)
Extended PCB	402 sec	2.4%	467.5 MB (4.783%)
512^3 resolution volume			
Monte Carlo w/o PCB	3674 sec	NONE	NONE
Traditional PCB	348 sec	9.6%	466.3 MB (4.780%)
Extended PCB	417 sec	3.8%	466.4 MB (4.785%)
Scene:CT head in Sponza	Render Time (s)	Image Delta	Memory Overhead
64^3 resolution volume			
Monte Carlo w/o PCB	10150 sec	NONE	NONE
Traditional PCB	348 sec	14.2%	466.3 MB (4.780%)
Extended PCB	756 sec	3.7%	468.0 MB (4.800%)
128^3 resolution volume			
Monte Carlo w/o PCB	15811 sec	NONE	NONE
Traditional PCB	348 sec	14.4%	466.3 MB (4.780%)
Extended PCB	755 sec	4.2%	467.3 MB (4.790%)
256^3 resolution volume			
Monte Carlo w/o PCB	31373 sec	NONE	NONE
Traditional PCB	348 sec	14.2%	466.3 MB (4.780%)
Extended PCB	864 sec	4.3%	467.1 MB (4.790%)

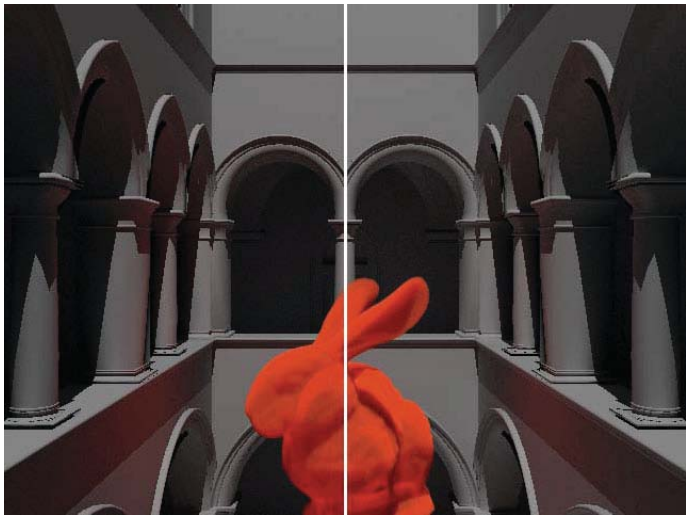


Fig. 3. Bunny Scene comparison of the PCB extension (left) and traditional Monte Carlo results (right)

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 (see Table 1), 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 the point cloud would be significantly simpler than the entire scene and can be moved to another system for out-of-core evaluation.

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 speedup upwards of 36 times that of the Monte Carlo renders. Compared to traditional PCB runs, we found the increase in overall runtime to be well worth the improvements to the renders we achieved. Even on sparse octrees without volumes, our *front to back* octree traversal 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: To objectively compare rendering results, we used a perceptual image difference program called pdiff to identify how close any two renders matched. The results, which ranged from 2.1% and 4.3%, are shown in Table 1.

Figure 2(a) 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 results are comparable.

4.2 Conclusion

We have presented an extension to the PCB algorithm by [1] 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. We obtained render speeds up to 36 times faster than that of pure Monte Carlo renders with a memory overhead between 2 to 5 MB with an image difference of less than 5% across all tests.

For more information on this subject, please refer to [16].

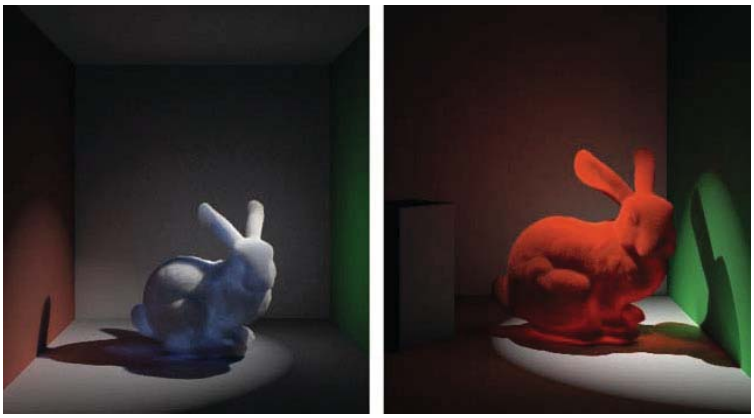


Fig. 4. Additional examples of clear color bleeding using our algorithm

4.3 Future Work

In our tests, we focused on isotropic phase functions in our volumes, while many volumes could have unique scatter functions. We could simulate more complex surface scattering functions by creating spherical harmonic representations of the radiance at any specific point in the volume. Another fruitful addition to our current method would be including 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 a sample point (often present in traditional PCB algorithms). Finally, as mentioned by Christensen [1], 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.

Acknowledgements. A special thanks to Patrick Kelly from DreamWorks Animations for his consistent help and support. Our brainstorming sessions were invaluable, and always left me full of new ideas as well as helping me hone the subject for this project.

References

1. Christensen, P.H.: Point-based approximate color bleeding (2008)
2. Robert, L., Cook, T.P., Carpenter, L.: Distributed ray tracing. In: Proceedings of the 11th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 1984, pp. 165–174. ACM, New York (1984)
3. Matt Pharr, G.H.: Physically Based Rendering: From Theory to Implementation, 2nd edn. Morgan Kaufmann, San Francisco (2010)
4. Cerezo, E., Perez, F., Pueyo, X., Seron, F.J., Sillion, F.X.: A survey on participating media rendering techniques, vol. 21, pp. 303–328 (2005)
5. Jensen, H.W.: Realistic Image Synthesis Using Photon Mapping. A.K. Peters, Ltd., Natick (2009)
6. Dorsey, J.: Radiosity and global illumination. The Visual Computer 11, 397–398 (1995), doi:10.1007/BF01909880
7. Christensen, P.H.: Point-based global illumination for movie production. In: SIGGRAPH 2010 (2010)
8. Jensen, H.W., Christensen, P.H.: Effcient simulation of light transport in scenes with participating media using photon maps. In: Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 1998, pp. 311–320. ACM, New York (1998)
9. Levoy, M.: Display of surfaces from volume data (1988)
10. Kajiya, J.T., Herzen, B.P.V.: Ray tracing volume densities (1984)
11. Westermann, R.: A multiresolution framework for volume rendering. In: Symposium on Volume Visualization, pp. 51–58. ACM Press, New York (1994)
12. Levoy, M.: Effcient ray tracing of volume data. ACM Transactions on Graphics 9, 245–261 (1990)
13. Guthe, S., Strasser, W.: Advanced techniques for high-quality multi-resolution volume rendering, vol. 28, pp. 51–58 (2004)
14. Tabellion, E., Lamorlette, A.: An approximate global illumination system for computer generated films, vol. 23, pp. 469–476 (2004)
15. Wrenninge, M., Bin Zafar, N.: Volumetric methods in visual effects (2010)
16. Gibson, C., Wood, Z.: Point based color bleeding with volumes. Technical report, California Polytechnic State University (2011), `http://digitalcommons.calpoly.edu/theses/533`