Enhancing Ray Tracing Through Bi-Directional Techniques And Binary Space Partitioning

Jishuang Chen, Jiashang Cao, Seoyoun Kim, Kevin Hopkins

Abstract

This study implements a bi-directional ray tracing methodology as an advanced alternative to traditional approaches for enhanced specular reflections. Additionally, binary space partitioning trees are implemented and evaluated for optimization in render time. The limitations of conventional models such as the Phong model are discussed, and comparisons are drawn against established methods including Whitted's ray tracing model and Cook et al.'s distributed ray tracing. With the utilization of various programming languages, systems, libraries, and APIs, this methodology provides promising improvements compared to a default implementation.

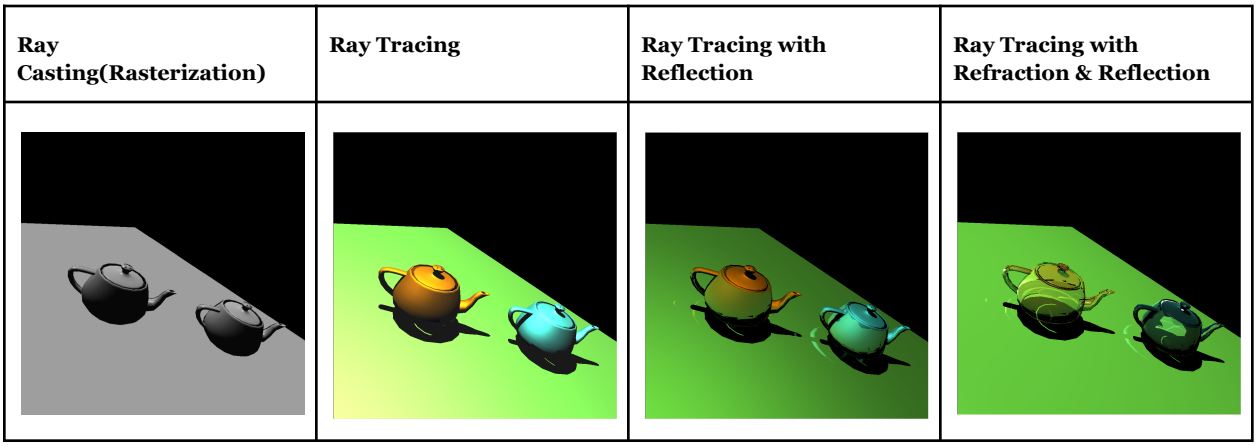

Figure 1: The raytracer exhibits much more complexity than ray casting, showcasing detailed shadows and realistic reflections and refractions.

1. Background

The common use of the Phong model has been fundamental in computer graphics for the determination of color values in a 3D scene. Despite being a widely accepted technique, it offers room for improvement with respect to accounting for realistic specular reflections. Other methods like Whitted's ray tracing model and Cook et al.'s distributed ray tracing provide a valuable home base in their attempt to improve the shortcomings of the Phong model. However, these too tend to falter when tasked with managing complex reflecting surfaces.
The obvious and intuitive model of tracing light rays from a light source to the viewer is wasteful because only a few of these rays reach the viewer. The ray tracing algorithm efficiently traces rays in the opposite direction: from the viewer to the light source.

We adopt Binary Space Partitioning (BSP) trees, a technique with origins in 1969 military flight simulation research and later honed for graphic rendering in a pivotal 1980 paper from Henry Fuchs of UNC. BSP outstrips traditional algorithms such as Painter's, Warnock's, and extensive Z-buffer checks by strategically partitioning space and employing a tree structure for swift occlusion culling. This approach effectively addresses the visibility challenge in complex scenes, particularly those with intricate reflections. Leveraging the BSP principles that revolutionized rendering in John Carmack's development of *DOOM*, our implementation

demonstrates marked improvements in computational efficiency. It surpasses older methods by eliminating the need for exhaustive surface sorting and per-pixel depth comparisons, thereby streamlining the rendering of still images with detailed light dynamics. The proven performance of BSP in our study underscores its continued significance and adaptability in advanced graphical computations.

2. Methodology

Bi-directional ray tracing has been deployed as an enhanced approach to ray tracing techniques. A noticeable advantage of this technique is its ability to handle shadows, indirect illumination, material refraction features and provide faster speed. The implementation of the BSDF illumination model [1] allows for the handling of indirect illumination. Material refraction features were achieved by applying the Fresnel term [2] combined with Snell's Law.

Additionally, a binary tree structure can iteratively segment space and the subset of polygons within that space such that only the most necessary of sections and polygons are eligible to be drawn by the render. A tree traversal will identify these sections and these sections will be assessed at various parts of the rendering process. This algorithm can be used to optimize the efficiency of occlusion culling and limit intersection and depth tests.

3. Implementation

The programming languages used include C++ for their speed of execution, precise control over graphics hardware, and versatility. Challenges encountered included managing recursive function calls for multiple reflecting surfaces, which were tackled using a depth limit. The program emitted a light from coordinate (0, 0, -1) through each pixel in the screen, and followed each light's reflection or refraction from object to object until it either 1) reached a light source or 2) reached a predefined maximum recursion depth (maxDepth). We tested the program with an .asc file we created with a Python script that modified the original homework teapot .asc file.

$$I = I_a + k_d \sum_{j=1}^{j=ls} (\bar{N} \cdot \bar{L}_j) + k_s \sum_{j=1}^{j=ls} (\bar{N} \cdot \bar{L}'_j)^n,$$

where

I = the reflected intensity,
I_a = reflection due to ambient light,
k_d = diffuse reflection constant,
$\bar{N}$ = unit surface normal,
$\bar{L}_j$ = the vector in the direction of the jth light source,
k_s = the specular reflection coefficient,
$\bar{L}'_j$ = the vector in the direction halfway between the viewer and the jth light source,
n = an exponent that depends on the glossiness of the surface.

Equation 1: Formula of the Phong Model [1]

$$I = I_a + k_d \sum_{j=1}^{j=ls} (\bar{N} \cdot \bar{L}_j) + k_s S + k_t T,$$

where

S = the intensity of light incident from the reflective direction

k_T = the refractive coefficient

T = the intensity of light from the refractive direction

Equation 2: Formula of the BSDF Model [1]

$$n_1 \sin\theta_\mathrm{i} = n_2 \sin\theta_\mathrm{t}$$

Equation 3: Snell's Law

3.1 Rendering Process

In our rendering process, we used the Phong illumination model for local illumination and recursive (reflect and refract) lights for global illumination. Then we integrated these two using the formula from the Bidirectional Scatter Distribution Function (Equation 2) for global illumination. This approach allows for realistic simulation of object surface color, reflective color and refractive light projection.

For each emitted light, we computed two lights. The local illumination part was in charge of ambient light, diffuse light, and specular light. The other part was global illumination, which included reflective and reflective light. These two lights had their own

recursive function that traced the light to its corresponding direction.

In the local illumination part, we defined the point where the ray and triangle collide as the incident point. This part was turned off when all the rays from the light source to the incident point were blocked by some other triangle. The triangle normal was interpolated from all three vertices to this point. We assigned specific diffuse (Kd) and specular (Ks) values to each vertex in the .asc file, like the teapot in our scene, to accurately depict how light interacts with their surfaces. These parameters are crucial in determining the light scattering characteristics of the object. The spec power and eye direction are given, and the reflection direction can be computed from normal and incident light direction. Once we get all of these parameters, we can get the ambient, diffuse, and specular light correspondingly (Equation 1).

In the global illumination part, we need to compute reflective light and refractive. To get these lights, we need to retrieve light intensity from recursive lights from their corresponding directions using light emitting. We are reusing the reflective direction from the local illumination part, which every material will reflect in the same way. The implementation employs Snell's Law (Equation 3) for refraction direction, crucial for transparent materials. For a real life simulation, we also consider double refraction due to material thickness. The reflectance nomenclature declared in Nicodemus' 1977 paper [3] informed us on how to declare rigid definitions for reflective surfaces, particularly bi-directional reflectance, which aided in synthesizing a natural concept of reflectance into rigid, programmed rules.

Additionally, Fresnel's equations, approximated using Schlick's approximation [2], which are used by the BRDF method, determine the reflection and refraction ratio based on the angle of incidence, enhancing realism in materials like glass or water. The model accounts for total internal reflection, a key factor in rendering reflective surfaces such as glass, where light is entirely reflected under specific angles and conditions. Once we get the reflection and refraction ratio, we will multiply it with its corresponding light intensities.

With both illumination parts, we can combine our results using the BSDF model (Equation 2). Note that BSDF is a superset of BRDF (Bidirectional Reflectance Distribution Function). We will treat the clamped sum of reflective light and specular light as the total specular light S. The $K_T T$ will be refractive light. After we plug in ambient light and diffuse light in the formula we will get our total illumination for this ray.

3.2 Binary Space Partitioning

Overall, we used a BSP tree to only do depth checks selectively when traversing through the tree. When an object is already fully occluded in the tree traversal, we do not need to check anything further down that tree. This greatly cuts down on excessive depth checks, optimizing the render time. The scene was drawn in an order that respects spatial relationships, with the closest drawn first. These partitions were repurposed for backface culling.

Our implementation of binary space partitioning was centered around developing comprehensive BSPNode and BSPTree data structures. The BSPTree class is tasked with constructing the BSP tree by dividing the scene into spatial partitions. This division is performed by recursively creating BSPNode instances, each delineating a specific spatial segment. This was the novel concept in Schumacher's 1969 paper that we referenced [5]; however, this implementation was based around a complicated position matrix. Instead, this paper was expanded upon and this matrix approach was simplified out [4] in Fuchs' 1980 paper, which we also referenced heavily. This paper formed our partitioning protocol. The partitioning of these nodes, guided by the choosePartitionPlane method, is based on the average position and orientation of polygons, ensuring a balanced distribution within the tree. This balance is critical for optimizing ray traversal and for efficiently managing visibility checks.

As rays are cast into the scene, they traverse the BSP tree, guided by a traversal function in the BSPTree class. This traversal assists occlusion culling, whereby entire sections of the scene are quickly dismissed if they do not intersect with the ray. This process is informed by the ray's interaction with the

partition planes of the nodes, substantially reducing unnecessary depth checks.

When a ray reaches a leaf node, which contains the actual geometry, intersection tests are conducted in this confined area. This localized approach, in contrast to brute force methods, markedly decreases the number of intersection tests, thereby expediting the rendering process.

Furthermore, backface culling in the BSP model is efficiently handled due to the inherent structure of the tree. By evaluating the orientation of polygons relative to the viewer within each partition, polygons facing away from the viewer can be quickly excluded from rendering.

In our implementation, individual polygons are grouped into partitions rather than grouping whole objects. This granularity allows for more precise control and culling, which is especially beneficial in densely populated scenes.

4. Results and Discussion

We observed the enhancement of specular reflections, as well as a faster processing speed due to the employment of the BSP tree. These results underscore the improved performance of the bi-directional ray tracing methodology compared to traditional techniques.

4.1 Unique Findings

A striking observation was the markedly enhanced realism of reflected images. This was due to the bi-directional ray tracing's ability to accurately simulate light propagation.

4.2 Comparison to Ray Casting

We compared our ray tracing framework to our previous ray casting framework. Ray casting, while capable of rendering scenarios limited to direct illumination, does not offer the nuances of complex lighting. Our refined ray tracing model, enriched with the Bidirectional Scattering Distribution Function (BSDF) and the incorporation of Fresnel terms, not only captures the subtlety of detailed shadows but also handles indirect illumination. The BSDF component allows for a comprehensive treatment of light interaction with surfaces, encompassing a spectrum of effects from diffuse to glossy reflections, based on the material properties. The Fresnel terms add a layer of realism by modulating reflectivity as a function of the viewing angle, thereby enhancing the depiction of materials in a way that aligns with their real-world optical characteristics. This combination of BSDF and Fresnel terms in our ray tracing paradigm yields a superior visual fidelity that shows both the direct and indirect lighting, as well as the realistic reflective and refractive qualities of materials within complex 3D scenes.

4.3 BSP Optimization

Importantly, the ray tracer produced visually identical results with and without the BSP tree implementation toggled on. The algorithm's effect is not visual, it is chiefly identified in the render time optimization.

# of teapots	Ray Tracing Avg. Time	Ray Tracing w/ BSP Avg. Time	Improvement
2	31.62s	29.64s	6.6%
3	47.43s	42.89s	10.5%
4	65.78s	56.12s	17.2%
5	86.95s	71.33s	21.8%

Figure 2: The BSP algorithm shows a distinct improvement in efficiency as more objects and thus more occlusion exists in the scene.

When a python script was used to generate more .asc files with progressively more teapots, each located at a fixed offset from each other, we were able to observe a marked improvement in the benefit of the BSP algorithm. As the scene became more cluttered with objects and thus more polygons, this meant more occlusion, intersection, backface culling, and generally would strain a standard ray tracer with more depth checks. As expected, the more polygons the scene had, the more performance improvement

the BSP algorithm showed over the standard ray tracer.

4.4 Potential Improvements

In our model, we were calculating the indirect illumination (reflect and refract light) and the direct illumination (diffuse and specular light). Thus, our model can count as a semi-global illumination model. To make a more realistic image, we can change the ray rendering to a complete global illumination. For instance, in the last two image instances in figure 1, there is a highlight on the green surface. The highlight color is rendered by the reflected light from the edge of the teapot. By making a global illumination to this scenario, there will be an additional teapot color that projects into the green plane, which makes the image more realistic.

Similar to ray casting, ray tracing is sampling the color information on a continuous space. In this case, the resulting image might have less details and disintegrated edges. We could possibly use the Monte Carlo method to make multiple samples within each pixel. By making a small random offset sample within each pixel, we could produce a more refined result. The Monte Carlo method can also apply to recursive lights for more realistic lights. However, if we are trying to use this method, we need to pay attention to the run time efficiency issue because each unique sample needs a corresponding render thus the sample amount is proportional to the computing time.

Binary space partitioning is often used in real-time graphics applications like video games and flight simulators. Because these settings are moving, dynamic, and incorporate the viewer and their moving environment, these applications provide a much greater benefit to using an algorithm like BSP that reduces overdrawing. Otherwise, in producing a still render of two teapots, the BSP trees provide minimal improvement, but as the teapots increase the improvement increases. To harness this, we could perhaps enable our ray tracer to process scenes that move in some way, or create an environment in an .asc file that is much more rich in detail.

5. Conclusion

The bi-directional ray tracing methodology represents a significant advancement in simulating enhanced specular reflections. The potential use of this technique signifies a forward movement in the realm of computer graphics and 3D scene representation. The research anticipates a future direction that further refines this technique to handle ever complex graphics scenarios.

References

[1] Turner Whitted. 1980. An improved illumination model for shaded display. Commun. ACM 23, 6 (June 1980), 343–349.
Retrieved from:
https://dl.acm.org/citation.cfm?id=358882

[2] Schlick, C. 1994. An Inexpensive BRDF Model for Physically-based Rendering. Computer Graphics Forum. 13 (3): 233–246.
Retrieved from:
https://wiki.jmonkeyengine.org/docs/3.4/tutorials/_attachments/Schlick94.pdf

[3] Nicodemus, F. E., Richmond, J. C., Hsia, J. J., Ginsberg, I. W., & Limperis, T. 1977. Geometrical considerations and nomenclature for reflectance.
Retrieved from:
https://www.semanticscholar.org/paper/Geometrical-considerations-and-nomenclature-for-Nicodemus-Richmond/0709f019c495a97b2a0369ded5ec18fe52e8672c#paper-topics

[4] Fuchs H, Kedem Z and Naylor B. On visible surface generation by a priori tree structures. Proceedings of the 7th annual conference on Computer graphics and interactive techniques
https://dl.acm.org/doi/abs/10.1145/800250.807481

[5] Schumaker, R.A., Brand, P., Gilliland, M. and Sharp, W. "Study for Applying Computer-Generated Images to Visual Simulation," AFHRL-TR-69-14, U.S. Air Force Human Resources Laboratory (1969)
https://apps.dtic.mil/sti/citations/AD0700375