A Bi-Directional Path Tracing system for rendering high quality computer images and scenes

Nathan Butt
Department of Computer Science and Creative Technology
University of the West of England
Coldharbour Lane, Bristol
Nathan2.Butt@live.uwe.ac.uk
Student ID. 16013327

Figure 1: An example of PBR based rendering technique (Image curtosy of the Blender Foundation.) [2]

Abstract
Physically Based rendering defines a series of techniques which are designed to result in renders that emulate the environment as closely as possible. One class of PBR is global illumination algorithms which are derived from the ray-tracing family of algorithms. This report seeks to illustrate the research that has been conducted into implementations of one of the most advanced approaches to GI, that being Bi-Directional Path Tracing, in order to consider what steps will need to be taken to perform such an implementation.

Author Keywords
Rendering; Ray-tracing; Light Transport; Monte-Carlo Rendering;

UWE Games Technology, 12/2018, Bristol, UK.

Introduction

Bi-Directional path-tracing (BDPT) is a global illumination algorithm which combines sampled paths generated from both the viewer and the light source in order to estimate the radiance of all points in the scene. BDPT was first proposed by Eric LaFortune in 1994 [5] and was then further iterated upon by Eric Veach in 1997 [11] via the proposition of new sampling methods. BDPT and other raytracing algorithms are defined primarily by their approach to computing light transport.

Whilst all methods share common features, BDPT utilises Monte Carlo sampling in order to simulate the light transport within a 3D space which in turn allow for the production of high-quality images. This report will be reviewing research centred on the implementation of a BDPT rendering system and how this method differs from pre-existing raytracing implementations. The report will also detail information about how such a system can be implemented. With the intended goal of said research to provide the theory necessary to create a correct implementation.

Research Methods

The primary goal of this project is to create an implementation of BDPT that is capable of rendering arbitrary 3D scenes. These scenes would be comprised of different 3D models with different forms of light sources and surfaces. Therefore the research will be focused on answering several key research questions. These are:

1. What are the various methods that are utilised for light transport simulation?
2. What are the fundamental elements for image synthesis common to all algorithms?
3. What are the best algorithms for building the system according to a set specification?
4. How can these be implemented efficiently in a productive manner?

For this research, a selection of secondary research sources have been consulted involving both established journals such as ACM SIGGRAPH along with various books on the topic such as Physically Based Rendering From Theory to Implementation (PBRT). In addition to this, progress will be made on the implementation of algorithms arising from the research conducted both during the implementation and as part of the researched engaged for this report.

Research and Implementation

Basic Image Synthesis

A BDPT system is a ray-tracing based rendering method, so several collections of algorithms are required in order to resolve issues regarding image synthesis that is the creation of images within a 3D scene. This function is fulfilled by intersection algorithms which test for intersection with primitives.

Considering the systems need to be capable of rendering 3D models and objects, it is required that the system is capable of intersecting triangles as well as primitives such as boxes for rendering optimisation.

Intersection

In computer graphics Rays are defined by this function. Rays have an Origin (O) and a normalised direction (D) with magnitude (t)[7, 3].

$$R(t) = O + tD \tag{1}$$

Intersection of triangles can be performed utilising the Moller-Trumbore Algorithm [7]. This algorithm utilises the barycentric coordinate system which defines points relative to the origin triangle which when summed should equal one.

$$w + u + v = 1 \tag{2}$$

The algorithm equates the definition of a point along the ray to the definition of a barycentric point on a triangle. Allowing this equation to be derived which is then solved utilising scalier triple product for the derived set of matrices, providing all barycentric coordinates (u, v, w) as well as the magnitude of the ray.

Whilst a geometric solution could have been considered, this method was comparatively simple to implement whilst achieving similar results [7]. Hence, this algorithm will be used to perform intersections with objects in the scene.

Acceleration
Another consideration which is virtually compulsory for a raytracing system is a scene acceleration system as not doing so will result in $O(n)$ complexity for rendering which will be problematic in the case of complex environments [3, 9]. This is addressed via utilising a special data structure which subdivides the space into a grid or similar structure, allowing many redundant intersection tests to be avoided [9].

These methods involve the creation of bounding volumes (BV) which is a volume that encompasses an object or collection in objects, in this case a bounding box due to its computational simplicity. These BVs are then used to partition the space into various segments reducing the number of intersection tests per frame [10]. These objects form part of a BV-hierarchy (BVH) which in turn can be organised utilising various tree structures [1]. It is these hierarchies which are then recursively traversed with a node being inspected if an intersection is detected.

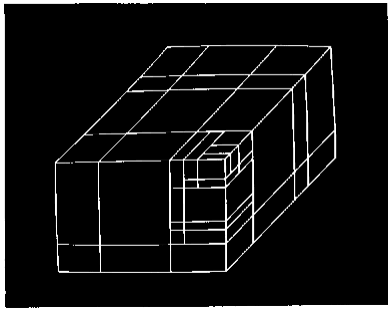

Figure 2: A visualisation of a hierarchy of BVs of unequal subdivision. [10]

Whilst these bounding volumes could be any form of 3D primitive, boxes are most often utilised outside of tight fitting boxes. For the boxes, Ray-AABB intersection tests are utilised to check for collisions and these are performed utilising the slab test. This is a straightforward comparison test which prevents excessive overhead in the system.

```
bool isIntersecting(Vector3 p0, Vector3 p1, Ray ray)
{
    Vector3 t0 = (p0 - rayOrigin) * Ray.invDir;
        Vector3 t1 = (p1 - rayOrigin) * Ray.invDir;
        Vector3 minVector = min(t0, t1)
        Vector3 maxVector = max(t0, t1)

        return msximumComponent(minVector)
<= minimumComponent(maxVector)
}
```

Figure 3: The simple SIMD slab test implementation [6]

Octrees and BSP trees however whilst effective do not guarantee the most efficient splits, as these algorithms partition there spaces equally. Therefore, tailoring the BVH to the space is often the best approach. This is done via utilising Surface Area Heuristic (SAH) [9].

$$C_s = t_t + (P_L \sum_{i=1}^{N_L} t_i(l_i)) + (P_R \sum_{i=1}^{N_R} t_i(r_i)) \qquad (3)$$

SAH utilises geometric probability in order to perform splits which reduce rendering costs by determining which split parallel to a select axis results in a lower test cost.

This produces a BVH which is tailored for the space in an optimised way partitioning away empty space as well as reducing the field of triangles to test. The result is a significant reduction in the count of rendering tests which in turn reduces rendering times.

Light Transport

This property deals with how a rendering model determines the radiance within a scene and the propagation of light within a scene. Path-tracing models such as BDPT account for indirect sources of light rather than direct sources of light such as the case in recursive raytracing. Hence the reason these are classed as Global illumination algorithms.

These models resolve the problem of illumination via the application of what is known as the rendering equation which is based upon modelling radiosity [4]. This equation defines radiance at a point on a surface in a 3D scene.

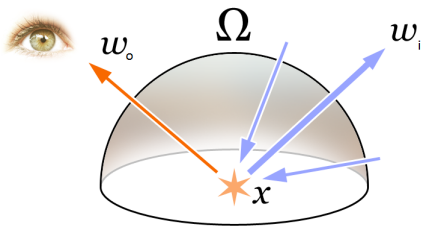

Figure 4: An illustration of the rendering equation in differential solid angle form [8]

$$L_o(x, \omega_o) = L_e(x, \omega_o) + \int_\Omega L_i(x, \omega_i) f_r(x, \omega_o, \omega_i) \cos\theta d\omega_i \qquad (4)$$

This Statement: $f_r(x, \omega_o, \omega_i)$

Represents the BRDF or Bi-Reflectance Distribution function which describes the behaviour of light upon contact with a surface. [1]

Initially however this equation is expressed with the integral being expressed in the form of a solid angle. However, path-tracing utilises a different form of the equation where radiance is expressed in terms integrating over surface area as seen below [4, 11]:

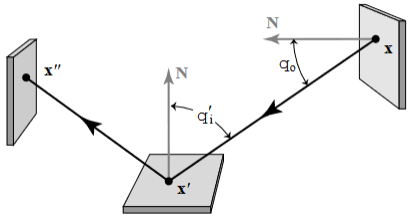

Figure 5: An illustration of the rendering equation. As you can see the base form integrates over the union of surface areas in the scene. [11]

$$L_o(x' \to x'') = L_e(x' \to x'') + \int_S L_i(x \to x') f_r(x \to x' \to x'') G(x \leftrightarrow x') dA(x) \quad (5)$$

This is known as the 3-point form of the rendering equation and it encompasses all surfaces in the scene with (S) representing a union of all surfaces/polygons in the scene. This Equation contains an additional term called the geometry term which is used to convert the differential solid angle to a differential surface area. This term is partially resolved via a shadow ray represented here using the Visibility function.

$$G(x \leftrightarrow x') = V(x \leftrightarrow x') \frac{\cos\theta \cos\theta''}{||x - x''||^2} \quad (6)$$

[11]

In order to produce an image, this equation would then be solved via the utilisation of Monte-Carlo sampling for the derived area form of the equation.

BDPT and Path Integral

Traditionally, this problem would be resolved in path-tracing by applying an unweighted Monte-Carlo solution. However as was pointed out by LaFortune and later by Veach, said methods do not produce perfectly ideal results. According to LaFortune this was largely a result of the fact that present algorithms were not able to consider both eye points and light sources equally [5, 11].

The aspect of the problem that BDPT seeks to resolve which was first proposed by Lafortune and later enhanced by Veach. This method involves tracing paths from both the light source and the camera (sometimes referred to as eye). Each vertex on both paths are connected to all other vertices on the other path via shadow rays in order to efficiently compute a wide range of lighting effects.

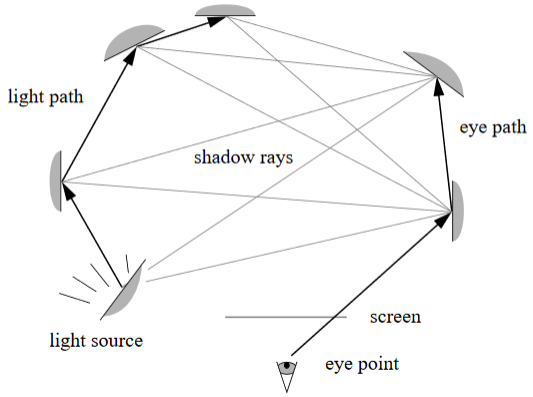

Figure 6: An illustration of how paths between vertex surfaces are generated in BDPT [5]

However, from this its possible to derive a new integral for the light path between the eye and the light source. This is known as the path integral which is defined as follows:

$$I = \int_{\Omega_k} f(\bar{x}) d\mu \quad (7)$$

where

$$S = \bigcup_{i=1}^{\infty} \Omega_k \tag{8}$$

for all paths of a finite length [11].

This was derived by Veach and integrates over all possible finite length paths (represented by Ω_k) that could connect a light source to a camera. This integral is comprised of a path function and a product measure. [11]

The path function is a derived form of the rendering equation which takes a set of paths between several points of length k. This path function is derived from the differential surface area form of the rendering equation and accounts for all paths.

$$f(\bar{x}) = L_e(x_0 \rightarrow x_1)G(x_0 \leftrightarrow x_1) \\ (\prod_{i=1}^{L} f_r(x_{i-1} \rightarrow x_i \rightarrow x_{i+1})G(x_i \leftrightarrow x_{i+1}))* \\ W_e(x_{l-1}, x_l) \tag{9}$$

Where $\bar{x}$ represents the set of paths generated on a particular point connecting both the light source and the view source.

$$\bar{x} = x_0, x_1, x_2 ... x_k$$

Hence the problem is now only comprised of a single integral which can then be evaluated to give the radiance of the surface point. This integral whilst too complex to be solved deterministically, can be solved utilising Monte-Carlo sampling in a similar manner to pre-existing methods such as path-tracing.

Sampling
Monte-Carlo integration is a numerical method derived from statistics which can be used to solve a variety of complex integrals making it very useful in computing radiance. In Monte-Carlo the expected value of an integral is computed as the following:

$$E[I] = \frac{1}{N} \sum_{i=1}^{N} \frac{f(X_i)}{pdf(X_i)} \tag{10}$$

[9]

This method involves computing the average value of a large number of random samples of the path integral divided by the probability density function (pdf) in area measure of generating that path. The result is an estimator whose values will equal the correct radiance values on the camera as the number of random samples tends to infinity.

It is desirable that an estimator have the lowest variance possible. Variance is the degree of an estimators error from the true value of an integral. Therefore a lower variance is desired as this results in a higher quality image.

In order to reduce variance when sampling there are several strategies that can be used. In path-tracing and related methods a technique called importance sampling is used. Importance sampling exploits the structure of the Monte-Carlo estimator and makes some modifications to the pdf function. Importance sampling simply exploits the

notion that if the pdf of a value x is proportional to the path function or in essence:

$$pdf(x) \propto f(x)$$

$$pdf(x) = cf(x)$$

As seen here this results in a constant value (c) across all samples, hence the variance is zero or close to zero [9]. Importance sampling is reflected in the BRDF/shading model for a surface where pdfs are proportional to the BRDF of a surface. This results in a rough approximation of the sampling behaviour for the said point on a surface.

Lafortune utilised this concept in his BDPT implementation where pdfs for path surfaces account for the BRDF of the surface [5].

However, one significant limitation with this method is that the algorithm only considers single sampling strategies. As Veach points out in his thesis, the different strategies used can result in different sampling outcomes, each of which has some degree of variance causing noise.

This is resolved by Veach via the utilisation of an algorithm called Multiple Importance Sampling which simply considers importance samples from all possible sampling strategies and incorporates them into a weighting function [11].

$$E[I] = \frac{1}{N} \sum_{i=1}^{N} \frac{w(X_i) f(X_i)}{pdf(X_i)} \tag{11}$$

This weighting function computes the weighted contributions of the complete collection of light paths across all sampling strategies. By weighting these multiple samples it enables all contributions from all light sources to be accounted for to the correct extent. This decreases the variance in the importance sampling model resulting in less noisy images [11].

The key to this algorithm is choosing a weighting function which is able to ensure the lowest variance possible via providing weighting values which ensure the lowest elements possible. The solution that Veach proposes is known as the balance heuristic.

$$W(x) = \frac{n_i p_i(x)}{\Sigma_k n_k p_k(x)} \tag{12}$$

[11]

The balance heuristic simply accesses a balance of probabilities between the current light path and the sum of probabilities of all possible paths. This produces a weight value which is then applied to the estimator, resulting in a minimal variance estimator for a wide variety of path integrands.

Implementation

To implement the system it requires that the two algorithms that BDPT is derived from are implemented first. One of these being the path-tracing algorithm which generates light paths from the camera to the light source connecting each surface vertex, and the second being light-tracing which is similar to path-tracing, except paths are generated from a light source rather than the camera. Upon implementing these two algorithms the two key systems along with their core components will be in place to allow for the bi-directional path tracer to be implemented.

In addition to this several methods are required, namely many of the fundamental algorithms mentioned previously that allows rays to be generated and intersection tests with objects as well as several acceleration objects.

Conclusion

As can be seen from the research, extensive work is needed in order to ensure that the system being created is able to function correctly and produce the intended sets of images. Currently the implementation of some of the fundamental acceleration structures has already occurred. Considering the research that has been performed the next step of the project will be to finalise the implementation of a Path-Tracer. Doing so will put into place the groundwork as far as sampling methods and probability density is concerned, allowing the light-tracing subsystem to then be implemented.

References

[1] Akenine-Moller, T., Haines, E., and Hoffman, N. *Real-Time Rendering*, 3rd ed. A. K. Peters, Ltd., Natick, MA, USA, 2008.

[2] Foundation, B. Cycles open source production rendering.

[3] Glassner, A. S. *An introduction to ray tracing*. Elsevier, 1989.

[4] Kajiya, J. T. The rendering equation. *SIGGRAPH Comput. Graph. 20*, 4 (Aug. 1986), 143–150.

[5] Lafortune, E. P., and Willems, Y. D. Bi-directional path tracing.

[6] Majercik, A., Crassin, C., Shirley, P., and McGuire, M. A ray-box intersection algorithm and efficient dynamic voxel rendering. *Journal of Computer Graphics Techniques (JCGT) 7*, 3 (September 2018), 66–81.

[7] Möller, T., and Trumbore, B. Fast, minimum storage ray/triangle intersection. In *ACM SIGGRAPH 2005 Courses*, ACM (2005), 7.

[8] N/A. Rendering equation, Apr 2018.

[9] Pharr, M., and Humphreys, G. *Physically Based Rendering: From Theory to Implementation (The*

Interactive 3d Technology Series). Morgan Kaufmann, 2004.

[10] Rubin, S. M., and Whitted, T. A 3-dimensional representation for fast rendering of complex scenes. *SIGGRAPH Comput. Graph. 14*, 3 (July 1980), 110–116.

[11] Veach, E. *Robust monte carlo methods for light transport simulation*. PhD thesis, Stanford University, 1997.