Advanced Technologies Report - Raytracer

Ricardo Heath
15000805

University of the West of England

July 17, 2019

The task is to create a CPU raytracer that supports reflections, refractions, shadows, and PBR materials. The solution was to implement this using the Rust programming language and create a multithreaded monte carlo raytracer that uses a BVH for spatial partitioning of the scene

1 Introduction

Raytracing is a slower but potentially more accurate method of rendering graphical objects on a computer. Rasterizers, such as those used in the games industry that take vertex positions and convert them to pixels, are not as accurate in modeling real lighting due to only attempting to approximate global illumination and the effects that indirect light rays have on a scene. Some rasterizers get close using PBR shaders and advanced lighting, but they sacrifice visual fidelity for performance. The raytracer created for this prototype attempts to properly simulate global illumination via indirect light bouncing.

2 Related Work

The ebook by Shirley, 2016 explores the implementation of a BRDF-incorrect monte carlo raytracer with the standard raytracing features such as materials. The article *An Overview of the Ray-Tracing Rendering Technique* and other associated articles describe the creation of a simple ray-tracer using BVH for spatial partitioning and geometric intersection tests. The ebook by Pharr, Jakob, and Humphreys, 2004 discusses physically correct rendering including the main rendering equation for indirect lighting and the monte carlo estimator. This project uses their work to create a prototype suitable for solving the task.

3 Method

3.1 Raycasting

Raycasting is done by constructing a directional vector and seeing if there are any intersections along that vector with anything in the scene. The scene is composed of multiple shapes with varying materials. Rays are cast for each pixel of the output image in order to calculate colour based on the intersections or lack thereof. A position along the ray is calculated as follows

$$pos = origin + (t \cdot direction) \tag{1}$$

A ray is often used to represent a light ray in the same way that light in real life will bounce around surfaces until hitting your eyeballs.

3.1.1 Samples Per Pixel

The number of rays cast per pixel determines the noise of the output image. Increasing the number of samples leads to longer execution times with diminishing returns in noise quality. Stratifying the samples can be done to ensure that the location of the samples within the pixel do not randomly converge on certain areas and are instead distributed somewhat evenly.

Multiple samples per pixel results in anti-aliasing along shape edges due to the samples of a pixel being averaged together. Some of the samples will intersect the shape and some will not, leading to smoother edges.

3.1.2 Parallelization

Raycasting is embarrassingly parallelizable as each pixel can be computed independently of every other pixel. The workload is split up in to chunks and processed on different threads until all the pixels of that chunk are complete.

3.1.3 Recursion

When a ray is cast in to the scene it will bounce upon intersection. This bounce generates a new ray which will be cast in to the scene again. A max recursion depth of 100 is used to prevent running out of stack space. Russian Roulette could be used to solve this problem by stopping the recursive bouncing based on probabilities. It's important not to stop a sample too early as it will introduce noise so a minimum threshold of 1 or 2 bounces is used.

3.1.4 Epsilon

The smallest unit that two floating point values can differ by is referred to as the Epsilon. Epsilon should be added to the origin of each successive bounce as to ensure that the next raycast is not internal to the shape.

3.1.5 Monte Carlo

Global Illumination's rendering equation is as follows

$$L_o(x,\omega_o) = L_e(x,\omega_o) + \int_\Omega L_i(x,\omega_i) f_r(x,\omega_o,\omega_i)\cos\theta d\omega_i \quad (2)$$

Monte carlo uses an estimator to solve the rendering equation, that equation is as follows

$$V = \frac{1}{N}\sum_{i=0}^{N}\frac{f(x)}{pdf(x)} \quad (3)$$

The bouncing that a ray exhibits allows a single ray cast from the screen to collect multiple colour values along its bounce path. These values are averaged together to form a single colour value. This averaging is a simple and inaccurate version of the monte carlo estimator.

3.2 Camera

The origin of the camera is $(0,0,0)$ with the y-axis going up and the x-axis going right. z-axis going in to the screen is negative. Two vector offsets are used, one horizontal and one vertical. The direction of the ray can be described with the equation

$$bl + (u \cdot horizontal) + (v \cdot vertical) \quad (4)$$

Where u and v are values between 0 and 1 representing the height and width of the current pixel in relaton to the screen resolution.

3.3 Shapes

Intersection tests are used to determine if a ray collided with the shape based on their geometric terms.

3.3.1 Sphere

A sphere has a position and radius. A spheres boundaries are calculated as follows

$$x^2 + y^2 + z^2 = radius^2 \quad (5)$$

where if the left-hand is equivalent to the right hand then the position is on the boundary of that sphere.

This can be simplified to this where p is the position of the point as a vector and c is a vector of the center of the sphere

$$dot((p-c),(p-c)) = radius^2 \quad (6)$$

If our ray at time t is on the boundary of the sphere then an intersection has occurred.

3.3.2 Triangle

Ray-triangle intersection algorithm using Barycentric coordinates is used.

Two edges of the triangle are calculated and then the normals are calculated by multiplying the edges together.

$$normal = \begin{pmatrix} e1_y \cdot e2_z \\ e1_z \cdot e2_x \\ e1_x \cdot e2_y \end{pmatrix} - \begin{pmatrix} e1_z \cdot e2_y \\ e1_x \cdot e2_z \\ e1_y \cdot e2_x \end{pmatrix} \quad (7)$$

the magnitude of the normals is then calculated to normalize the normals

$$\frac{normal}{\sqrt{n_x^2 + n_y^2 + n_z^2}} \quad (8)$$

The distance from the origin to the triangle is found and the dot of the normals and the ray is done to get the denominator. the numerator is the inverse of the normals dot ray origin. The intersection is then the numerator over the denominator and the intersection vector is $start + (dir * t)$. Normals are calculated for every edge and the determinant is calculated and checked to see if it is inside our outside the triangle. If the intersection for all edges is inside the triangle then the intersection is valid.

3.4 BVH

Bounding volume hiearchies are used as a spatial partition to improve the runtime performane of the prototype through intersection testing. BVH's are a tree structure, often implemented as a binary tree as in the case of this project, in which every leaf node of the tree is wrapped in a bounding volume, in this case an axis-aligned minimum bounding box. These bounding

boxes are contained within larger bounding boxes, the larger boxes being the parents of the smaller, such that the root node is a bounding box that encompasses the entire tree. If an intersection test is invalid for the bounding box of the leaf node then none of its child nodes could possibly intersect, reducing the amount of tests performed.

3.4.1 AABB

Axis-aligned minimum bounding box is a box volume in which the edges of the box are parrelel to the axis of the world, in other words the box is not rotated. AABB intersection is performed per axis.

$$t0 = min(\frac{(minbox_a - origin_a)}{direction_a}, \frac{maxbox_a - origin_a}{direction_a}) \tag{9}$$

$$t1 = max(\frac{(minbox_a - origin_a)}{direction_a}, \frac{maxbox_a - origin_a}{direction_a}) \tag{10}$$

Where the a subscript element is one of the position axis (x, y, or z). minbox is the smaller values of the AABB positions while maxbox is the larger values of the AABB positions.

When t1 is less than t0 then the ray is outside the bounds of the AABB for any of the axes then no intersection has occured, otherwise it has occured.

3.5 Materials

Materials are applied to shapes to define their interaction with rays.

3.5.1 Diffuse

Diffuse materials are a simple material that have albedo properties and absorption properties. Albedo determines the colour of the material while absorption determines how much of the colour is received by the ray. Rays that hit a diffuse material are then randomised for the bounce by using a random unit sphere. A random point is picked from the unit sphere that is tangent to the intersection and then a ray is sent from that intersection to the random point. The random point in the unit sphere is calculated by rejecting any points that lie outside of the area of that unit sphere.

3.5.2 Metallic

Metallic materials are a potential mirror material if the fuzzyness of the metal is near 0. As fuzzyness approaches 0 the clarity of the metallic materials reflection increases. The reflected ray is as follows

$$v - 2dot(v, n) \cdot n \tag{11}$$

Where v is the normalized ray direction and n is the normal of the intersection plus the fuzzyness factor

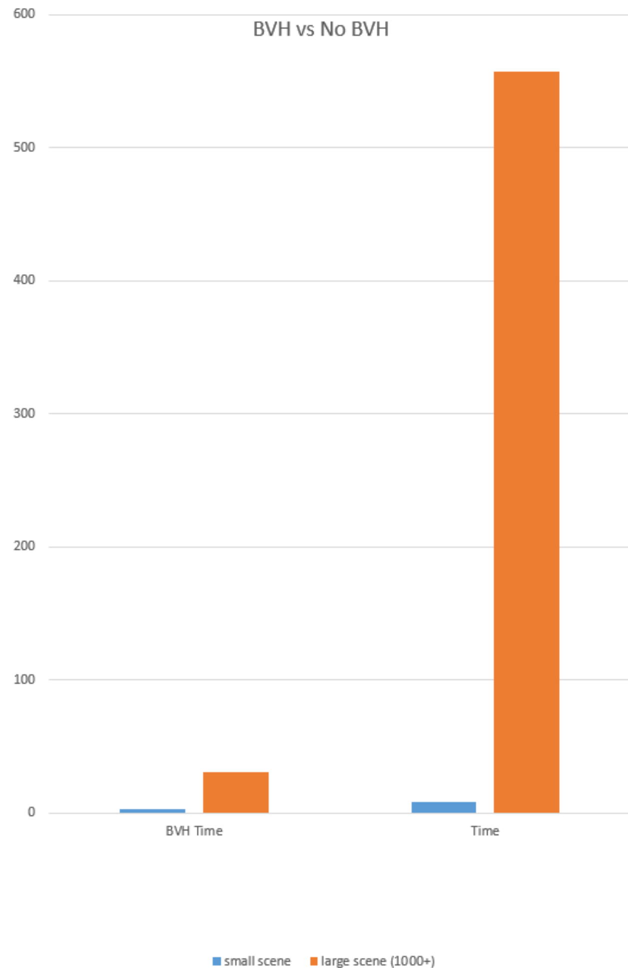

Figure 1: *BVH vs no BVH in different scenes*

which is defined as $fuzz * pos$ where pos is a random position in a unit sphere.

3.6 Gamma

Gamma correction should be used to ensure that the brightness of the colour values matches our expectations. Image viewers in software assume images are inherently gamma coloured, so despite our colour being accurate the viewer adjusts them such that they become inaccurate. Gamma 2 is simplified to

$$colour = 255 * \sqrt{colour} \tag{12}$$

while also adjusting the range of pixel values to match the 8-bit range expected of each colour component.

4 Evaluation

When the BVH for the project is used there is a very noticeable improvement in performance as can be seen in Figure 1. Two scenes were tested, one with roughly a dozen objects and one with over a thousand. The BVH in the small scene is 3.5 times quicker, while in the big scene it is 18.5 times quicker. This is inline with the expected n(log(n)) Big-O time of a BVH rather than a linear speedup.

5 Conclusion

There are a number of issues with the project in that the task was not fully accomplished. There is a lack of sufficient materials, lighting, and shadows. There's also the possibility of fixing the cube shape present in the prototype, the normals, and adding texturing. Global illumination, while present, is not BRDF correct and the lack probability distribution functions results in the diffuse material being slightly emissive.

Bibliography

An Overview of the Ray-Tracing Rendering Technique. URL: `http://www.scratchapixel.com/lessons/3d-basic-rendering/ray-tracing-overview`.

Pharr, Matt, Wenzel Jakob, and Greg Humphreys (2004). *Physically Based Rendering: From Theory to Implementation*. URL: `http://www.pbr-book.org/3ed-2018/contents.html`.

Shirley, Pete (2016). *Ray Tracing in One Weekend*.