# CS 488 Project Proposal

Ryan Loftus

July 8, 2024

## 1 Purpose

The purpose of this project is to extend the existing CS 488 raytracer to have more realistic and complex lighting effects as well as performance improvements. The specific improvements and their implementation are described in this proposal.

## 2 Topics

- Path Tracing
- Fresnel Reflection
- Microfacet Models
- Alpha Blending
- IBL for Lambertian
- SAH kd-Tree

## 3 Statement

This project will build upon the existing cs488 raytracer by adding path tracing, parallelization, Fresnel reflection, microfacet models, alpha blending, and image-based lighting for Lambertian materials.

Path tracing involves tracing light from each pixel as would be done by a raytracer, except that we trace each pixel multiple times, and include a degree of randomness in the angle at which light travels. We then take the probability weighted average of the sample to get the value of a given pixel. Path traced images are more physically realistic and improve upon raytracing with soft shadows, color bleeding, and global illumination. Because path tracing computes each pixel value independently, the path tracing algorithm can be parallelized by partitioning the image into segments and processing those segments in parallel.

Fresnel reflection uses Fresnel equations to determine how much light is reflected and how much light is refracted by a glass surface. Fresnel reflection is used to create more realistic lighting effects with glass surfaces.

Microfacet models are physically-based models for surfaces that reflect light. For example, a flat mirror and a rough metal surface reflect light very differently. The Cook-Torrance model approximates the distribution of orientations of microfacets and is used to simulate the glossiness of the reflective surface.

Alpha blending is used to render translucent surfaces. The alpha value determines the degree of transparency of the surface. Alpha blending will allow the raytracer to properly render translucent surfaces.

Image-based lighting maps light from an environment image onto the objects in the scene. IBL for Lambertian materials uses the light from an environment map image to light Lambertian surfaces, creating more realistic lighting effects for Lambertian materials.

A Surface-Area Heuristic kd-tree uses the Surface-Area Heuristic to construct a kd-tree of the objects in a scene for faster raytracing. The Surface-Area Heuristic BVH is already implemented, and can be improved upon by using a kd-tree instead which performs faster on average.

## 4 Technical Outline

### 4.1 Path Tracing

For path tracing, we use the following high level algorithm to sample rays for a given pixel:

```
PathTracePixel(pixel, scene, maxDepth):
    color = (0, 0, 0)
    for sample in 1 to numSamples:
        cx, cy = getPixelCenter(pixel)
        xOffset = generateRandomNumberBetween(0, 1)
        yOffset = generateRandomNumberBetween(0, 1)
        ray = generateRayThroughScreenAt(cx + xOffset, cy + yOffset)
        color += traceRay(ray, scene, maxDepth)
    color /= numSamples
    return color
```

Because path tracing iterates over each pixel independently, we can parallelize path tracing by partitioning the image and processing each segment in the partition in parallel. High level pseudocode for this approach is given below:

```
PathTraceSegment(segment, imageBuffer):
    for px in segment:
        imageBuffer[px] = PathTracePixel(px)

PathTraceImage(imageBuffer, imageSize, numThreads):
    imagePartition[1..numThreads] = partition(imageSize, numThreads)
    threads[1..numThreads] = [null..null]
    for i from 1 to numThreads:
        threads[i] = new Thread(PathTraceSegment, imagePartition[i], imageBuffer)
    for i from 1 to numThreads:
        threads[i].join()
    return imageBuffer
```

The number of threads used will be the number of cores available on the machine. This can be found using `std::thread::hardware_concurrency()`.

### 4.2 Fresnel Reflection

When light hits glass, we use the Fresnel equations:

$$\rho_s = \frac{\eta_1 \cos(\theta_i) - \eta_2 \cos(\theta_o)}{\eta_1 \cos(\theta_i) + \eta_2 \cos(\theta_o)}$$

$$\rho_t = \frac{\eta_1 \cos(\theta_o) - \eta_2 \cos(\theta_i)}{\eta_1 \cos(\theta_o) + \eta_2 \cos(\theta_i)}$$

$$F(\theta_o, \theta_i) = \frac{1}{2}(\rho_s^2 + \rho_t^2)$$

where $\theta_i$ is the angle between the normal and the incoming ray and $\theta_o$ is the angle between the normal and the refracted ray, $\eta_1$ is the index of refraction of the material the ray is exiting and $\eta_2$ is the index of refraction of the material the ray is entering.

In path tracing, $R = F(\theta_o, \theta_i)$ gives the probability that the ray is reflected and $(1 - R)$ gives the probability that the ray is refracted.

## 4.3 Microfacet Models

We will use the Cook-Torrance model described by Dunn and Wood to compute the reflectance:

$$r = k_a + \sum_{i=0}^{n} l_c * (\vec{n} \cdot \vec{l}) * (d * r_d + s * r_s)$$

where $k_a$ is the ambient reflectance, $\vec{l}$ is the light ray vector, $\vec{v}$ is the view direction vector, $l_c$ is the light contribution from each light source, $n$ is the number of light sources, $r_d$ is the diffuse reflectance, $r_s$ is the specular reflectance, and $s, d$ are constant scaling factors for the specular and diffuse reflectance defined such that $s + d = 1$.

$$r_d = k_d$$

where $k_d$ is the diffuse reflectance coefficient of the material.

$$r_s = \frac{D * G * F}{4 * (\vec{n} \cdot \vec{l}) * (\vec{n} \cdot \vec{v})}$$

where $D, G, F$ are the Normali Distrubtion Function, Geometric Attenuation Function, and Fresnel Function respectively. We've already defined the Fresnel function. We use:

$$D = \frac{1}{\pi\alpha^2}(\vec{h} \cdot \vec{n})^{\frac{2}{\alpha^2}-2}$$

where $\alpha$ is the gloss coefficient and $\vec{h}$ is the half-vector. And,

$$G = \min\left\{1, \frac{2(\vec{h} \cdot \vec{n})(\vec{n} \cdot \vec{v})}{\vec{v} \cdot \vec{h}}, \frac{2(\vec{h} \cdot \vec{n})(\vec{n} \cdot \vec{l})}{\vec{v} \cdot \vec{h}}\right\}$$

Using this model to compute reflectance in the shading function will allow a glossiness parameter that determines the behaviour of metal surfaces.

## 4.4 Alpha Blending

The final pixel value, $p$, for a ray, $r$, should be computed as

$$p = \alpha_{s_1} c_{s_1} + (1 - \alpha)\text{next\_surface}(r')$$

where $s_1$ is the first surface hit by $r$, $\alpha_{s_1}$ is the alpha value for $s_1$, and $c_{s_1}$ is the color of $s_1$. $r'$ is the ray obtained by altering the ray $r$ by moving its start point just past the surface $s_1$. This allows the ray to continue to hit another surface. We continue this until either $(1 - \alpha_{s_1}) = 0$ or a pre-defined depth limit is reached.

## 4.5 IBL for Lambertian

Image-based lighting for Lambertian materials will be implemented by adding a second loop in the shade function for Lambertian materials that runs after the loop over all point light sources. The second loop runs some pre-determined number of times $N$ and iterates over $N$ randomly generated rays, tracing each ray, and if the ray does not hit any surface, we add the IBL quantity to a running sum. We then take the average of the IBL quantities by dividing the running sum by $N$ at the end.

## 4.6 SAH kd-Tree

A kd-tree is a binary tree that splits a space at each level. The root node is the space that contains all objects, the interior nodes are subspaces, and the leaf nodes are a set of objects. The pseudocode given by Wald and Havran for building a kd-tree is given below (note that searching the kd-tree will be very similar to searching the BVH that is already implemented, so the searching algorithm is not described in pseudocode here):

```
RecBuild(triangles, space):
    if triangles.size <= 1:
        return new LeafNode(triangles)
    p = FindPlane(triangles, space)
    if cost when splitting on p >= cost without splitting:
        return new LeafNode(triangles)
    (SL, SR) = split space with p
    (TL, TR) = split triangles by p
    NL = RecBuild(TL, SL)
    NR = RecBuild(TR, SR)
    return new interior node(p, NL, NR)

BuildKDTree(triangles):
    space = AABB(triangles)
    return RecBuild(triangles, space)
```

When building the kd-tree, we use the Surface-Area Heuristic to decide whether to make a subspace a leaf node or split it into two more subspaces. The Surface-Area Heuristic minimizes:

$$C = C_b + \frac{SurfaceArea(Box_{child})}{SurfaceArea(Box_{parent})} N_o C_o$$

where $C$ is the cost of searching through the space, $C_b$ is the cost of ray-box intersection, $C_o$ is the cost of ray-object intersection, $N_o$ is the number of objects in the space, and $\frac{SurfaceArea(Box_{child})}{SurfaceArea(Box_{parent})}$ estimates the probability that a ray going through the parent also goes through the child.

In addition to the SAH, another optimization is to prefer splits where one half is empty. Define:

$$\lambda(p) = \begin{cases} 0.8 & |T_L| = 0 \vee |T_R| = 0 \\ 1 & \text{otherwise} \end{cases}$$

Then, we multiply whatever cost is computed by SAH by $\lambda(p)$ to get a new cost which prefers splits where one half is empty.

Finally, we suppose our $SAH$ function takes into account the triangles on the plane and considers which side triangles on the plane should be put into, returning the cost after putting triangles on the plane into the optimal side and the side that is optimal. For example, $SAH$ might return $(5, LEFT)$.

To build the kd-tree, we use algorithm presented by Wald and Havran:

```
PerfectSplits(triangles, space):
    B = clip triangles to space to consider perfect splits
    S = empty set
    for k from 1 to 3:
        S = S union (k, B_{min,k}) union (k, B_{max,k})
    return S

Classify(triangles, SL, SR, p):
    TL = TR = Tp
    for all t in triangles:
        if t lies in plane p and Area(p intersect space) > 0:
            Tp = Tp union t
        else:
            if Area(t union (SL - p)) > 0:
                TL = TL union t
            if Area(t union (SR - p)) > 0:
                TR = TR union t
    return (TL, TR, Tp)
```

```
FindPlane(triangles, space):
    for all t in triangles:
        minCost, bestP, bestPside = (infty, NULL, NULL)
        for all p in PerfectSplits(triangles, space):
            (SL, SR) = split space with p
            (TL, TR, Tp) = Classify(triangles, SL, SR, p)
            (cost, pside) = SAH(space, p, |TL|, |TR|, |Tp|)
            if cost < minCost:
                (minCost, bestP, bestPside) = (cost, p, pside)
    (TL, TR, Tp) = Classify(triangles, SL, SR, p)
    if bestPside == LEFT:
        return (bestP, TL union Tp, TR)
    else:
        return (bestP, TL, TR union Tp)
```

## 5 Objectives

### 5.1 Soft Shadows and Color Bleeding Using Path Tracing

The goal of this objective is to render images with more realistic shadows and surfaces. This will be accomplished by the path tracing described above. The shadows will be "softer," meaning they will be lighter close to the edges like shadows appear in real life. Similarly, color bleeding (also accomplished by path tracing) will make the edges between different colored surfaces less sharp, creating more realistic scenes.

### 5.2 Global Illumination Using Path Tracing

The goal of this objective is to achieve realistic lighting by simulating the interaction of light with surfaces, including multiple light bounces and indirect illumination. This will be accomplished by implementing path tracing.

### 5.3 Faster Path Tracing Using Parallelization

The performance improvement attained through parallelization will depend on the number of cores available on the machine being used. Suppose the image size is constant, there are $n$ CPU cores are available, and the runtime of the path tracing algorithm without parallelization is $R$. Then, the goal is to achieve a runtime that is close to $\frac{R}{n}$. The overhead involved in multithreading makes it impossible to achieve exactly $\frac{R}{n}$ runtime, but we expect to be close to this ideal runtime. For example, if $R = 100$ time units, a program that takes 55 time units with $n = 2$ and 40 time units with $n = 3$ would be a success.

### 5.4 Realistic Glass Using Fresnel Reflection

The goal of this objective is to have more realistic glass materials by implementing Fresnel reflection which ensures some light is reflected and some light is refracted when a ray hits glass, which is aligned with how light reflects/refracts off of glass materials in real life.

### 5.5 Realistic Metals Using Microfacet Models

The goal of this objective is to support a glossiness parameter for metals. A lower glossiness should correspond to a smoother surface that reflects light like a mirror and a higher glossiness should correspond to a rougher surface that reflects light like a rough/scratched piece of sheet metal.

### 5.6 Translucent Materials Using Alpha Blending

The goal of this objective is to support translucent materials so that objects behind a translucent material are visible and colored according to the color of the translucent material in front. If multiple translucent materials are in front of one another, the result should be colored accordingly, up to a configurable (but finite) level of depth.

### 5.7 IBL for Lambertian Materials

The goal of this objective is to have more realistic ambient lighting and more cohesive scenes by diffusing light from the environment map onto Lambertian materials. Currently, only the background and specular reflection make use of IBL, so the lighting is not consistent because IBL does not affect all materials as it would in the real world. Implementing IBL for Lambertian materials will fix this.

### 5.8 Faster Raytracing Using SAH kd-Tree

According to Rivera-Alvarado and Zamora-Madrigal, the kd-tree performs better than the BVH for raytracing use cases. They observed an average runtime increase of 34% when using kd-trees compared to BVH. The goal of this objective is to improve the runtime of the raytracing algorithm by $\sim 34\%$ by implementing a Surface-Area Heuristic kd-tree.

## 6 Previously Implemented Objectives

The following extra objectives were achieved as part of previous assignments:

- Faster raytracing using an SAH BVH acceleration data structure
- Faster particle simulation using Barnes-Hut

## 7 Bibliography

**Citations:**
Hanrahan, P. (2001). *Monte Carlo path tracing.* SIGGRAPH 2001 Course 29: Monte Carlo Ray Tracing.

Pharr, M., Jakob, W., & Humphreys, G. (2018) *Physically Based Rendering: From Theory To Implementation.* Chapter 14.5.

**Use in Project:**
Reference material for math, algorithms, and concepts involved in path tracing.

**Citations:**
Smith, A. R., & Blinn, J. F. (1996, August). Blue screen matting. In Proceedings of the 23rd annual conference on Computer graphics and interactive techniques (pp. 259-268).
Horie, T. (2002). *Alpha Blending Tutorial.* Phat Code. www.phatcode.net/articles.php?id=233.

**Use in Project:**
Both texts will be used as references for the math and implementation of alpha blending.

**Citation:**
Ladefoged, K. S., & Madsen, C. B. (2019, October). *Spatially-varying diffuse reflectance capture using irradiance map rendering for image-based modeling applications.* In 2019 IEEE International Symposium on Mixed and Augmented Reality (ISMAR) (pp. 46-54). IEEE.

**Use in Project:**
This paper will act as a secondary reference for IBL, where the primary resource will be the shading lecture.

**Citations:**
Walter, B., Marschner, S. R., Li, H., & Torrance, K. E. (2007). *Microfacet Models for Refraction through Rough Surfaces.* Rendering techniques, 2007, 18th.

Dunn, I. & Wood, Z. *Graphics Compendium.* Chapter 47.

Karis, B. (2013). *Specular BRDF Reference.*

Cook, R. L., & Torrance, K. E. (1982). *A reflectance model for computer graphics. ACM Transactions on Graphics* (ToG), 1(1), 7-24.

**Use in Project**:
These texts explain microfacet models and specifically the Cook-Torrance model which will be used for this project. Texts will be used to understand the various parameters and their purpose as well as the reflectance equations that will be used.

**Citations:**
Wald, I., & Havran, V. (2006, September). *On building fast kd-trees for ray tracing, and on doing that in O (N log N).* In 2006 IEEE Symposium on Interactive Ray Tracing (pp. 61-69). IEEE.

Rivera-Alvarado, E., & Zamora-Madrigal, J. (2023). *An evaluation of Kd-Trees vs Bounding Volume Hierarchy (BVH) acceleration structures in modern CPU architectures.* Revista Tecnología en Marcha, 36(2), 86-98.

**Use in Project:**
The Wald & Havran text describes an algorithm for building kd-trees that perform well and the Rivera-Alvarado & Zamora-Madrigal text provides evidence for why the kd-tree implementation will be an improvement over the current BVH implementation.