Creative Technologies Project: CPU Path Tracer

**Alex St John Feetham**[alex2.feetham@live.uwe.ac.uk](mailto:alex2.feetham@live.uwe.ac.uk)  
Supervisor: Thomas Bashford-Rodgers

Department of Computer Science and Creative Technology

University of the West of England  
Coldharbour Lane  
Bristol BS16 1QY

**A picture containing wall, indoor

Description automatically generated**

**Abstract**

The aim of this project is to find an efficient solution to implement a path tracing system to run on a computer’s central processing unit. The project will be implemented using Visual Studio and written in C++. The project successfully implements a solution for path tracing on the cpu.

**Keywords**: Path Tracing; Ray Tracing; Bounding Volume Hierarchy; PBR Materials;

**How to access the project**

All of the project files can be found here:

<https://github.com/alexfeetham98/CTP_17016942>

It is possible that there may be an issue with the SFML libraries not correctly linking once the project is downloaded. This is due to it being unable to find the libraries as it is looking for the wrong file path as it will be looking for the C:// drive of my computer. If this happens, please consult this YouTube tutorial to re-setup the file paths. All the things needed are there, the project just doesn’t know how to find them.

<https://www.youtube.com/watch?v=YfMQyOw1zik>

**1. Introduction**

A path tracer is a computer graphics system which renders three-dimensional scenes which are faithful to reality. To accomplish this, rays are fired from a single viewpoint (camera) into the scene. When a ray intersects with an object in the scene, it is either reflected or refracted depending on the material properties of the object. The ray continues to bounce around the scene, collecting colour information from each intersection, until the ray intersects with a light source. When this happens, the system averages all the values calculated from all the paths that were traced in order to get the final pixel colour value (Dusterwald, 2016). This project aimed to demonstrate how a path tracer could be implemented on a CPU.

**Key Deliverables**

* Implement a path tracer using C++
* Implement virtual cameras/scenes
* Implement sampling (anti-aliasing)
* Implement different materials
* Implement lighting
* Implement a BVH

The idea for the project came after attempting to build a ray tracing program for another module. That project showed the subject area to very interesting and this project has arisen as a result.

In ray tracing, a ray is fired out from the camera into the scene and is traced until it intersects with a solid object in the scene. From here, a ray is cast to each of the light sources in the scene to calculate the illumination and the surface shading is calculated for the intersection point of the 2 rays. Then if the object is transparent, the ray is sent out at an angle to simulate refraction, or the ray is sent back in the opposite direction is the object is reflective (Dusterwald, 2016).

A path tracer builds upon this concept. In a path tracer, rays are distributed randomly within each pixel in camera space and at each intersection with an object a new reflection ray, pointing in a random direction, is generated. Once a ray has finished bouncing around the scene, sample value is calculated based on the objects the ray bounced against and the sample value is added to the average for the source pixel (Öqvist, 2015). The more samples per pixel, the higher the quality of image that will be produced. If the number of samples per pixel (SPP) was say 200 SPP, then it would merely produce an image that resembles a bunch of dots on screen. However, if the number of samples per pixel was more in the thousands, a real image would start to be produced.

**2. Practice**

The project was set up in the Visual Studio IDE (Microsoft Corporation, 2019) and written using the C++ programming language. To display the outputted image from the program, the Simple and Fast Multimedia Library (SFML) was chosen. SFML provides a simple interface to the various components of your computer (Gomila, 2014). SFML was used to handle the creation of a window and display the calculated image from the main program. If not for this, a window would need to be crated from scratch using the Direct3D API which was unnecessary for this project, so SFML was used to make things simpler.

**2.1 Ray Class**

All ray/path tracers require a ray class, and a computation of what colour is seen by the ray. A ray can be expressed by the following equation:

P is the point where the ray intersects an object, A is the origin of the ray, B is the direction of the ray, and t is time. This is illustrated by **Figure 1**.

A picture containing text, clock, watch

Description automatically generated

Figure 1 Visualisation of the equation for a ray (Shirley, 2020)

**Figure 2** shows the result of the one of the first implementations of a ray class. If the ray intersected with the sphere object then it returned the colour at the intersect point, else it missed so returned the background colour.

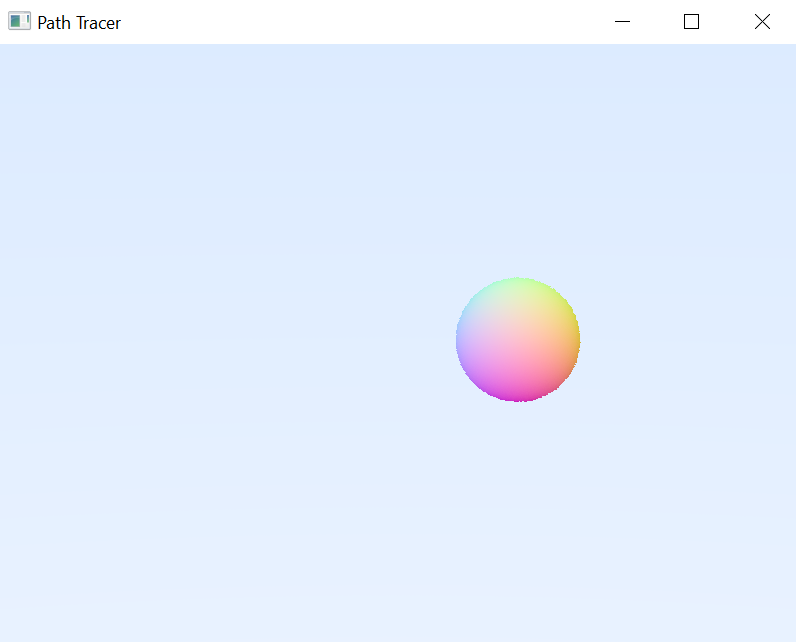


Figure 2 Ray traced sphere

**2.2 Sampling (Anti-Aliasing)**

Anti-aliasing is another important feature that a path tracer should include. Image anti-aliasing is the smoothing of edges and colours in digital images and fonts. It makes edges appear less jagged and helps blend colours in a natural-looking way (Christensson, 2019). It does this by taking multiple samples per pixel to calculate a better level of accuracy and make the image appear smoother. The following code demonstrates how this was implemented:

for(int j = image\_height - 1; j >= 0; j--) {

for(int i = 0; i < image\_width; i++)

{

colour col(0, 0, 0);

for (int s = 0; s < num\_samples; s++)

{

float u = float(i /

float(image\_width));

float v = float(j /

float(image\_height));

Ray r = cam.get\_ray(u, v);

col += ray\_colour(r, background,

world, sceneDepth);

}

col /= float(ns);

col = Vector3(sqrt(col[0]),

sqrt(col[1]), sqrt(col[2]));

The effect sampling has is very noticeable, even with a relatively small number of samples per pixel. **Figure 2** had only 1 sample per pixel, causing the image to appear noisy and for the edges of the sphere to appear jagged. On the other hand, **Figure 3** has 100 samples per pixel. The image is a lot clearer, with smooth transitions over the colour gradient and smooth edges around the sphere. This makes the image far more realistic looking. Further examples of sampling can be seen in Appendix A.

Chart, bubble chart

Description automatically generated

Figure 3 Ray traced sphere

**2.3 Virtual Camera**

(Shirley, 2020) presents a relatively simple camera class which has been successfully implemented into the project. The camera allows for the movement and modification of the viewport. The camera class uses a vector3 for its position in 3D space and as well as for the position it is looking at. Additionally, it has modifiable aspect ratio, aperture, field of view, and focusing distance. This camera acts as the origin point for all of the rays which are fired into the scene.

**2.4 Material Class**

**2.4.1 Lambertian (diffuse)**

Diffuse objects that don’t emit light merely take on the colour of their surroundings, but they modulate that with their own intrinsic colour. Light that reflects off a diffuse surface has its direction randomized (Shirley, 2020). To accomplish this in the project, the point the ray intersects an object, along with the ray itself are passed into the material class. A random vector is generated based on the normal of the point of the intersection. This new random vector becomes the new directional vector of the ray. Each time the ray intersects and object, the colour at that point is returned and added to the previous colour. This simulates how light causes colours to reflect in each other. **Figure 4** illustrates this effect as you can see the green and red colours from the walls are slightly visible in the shadows of the two cubes.

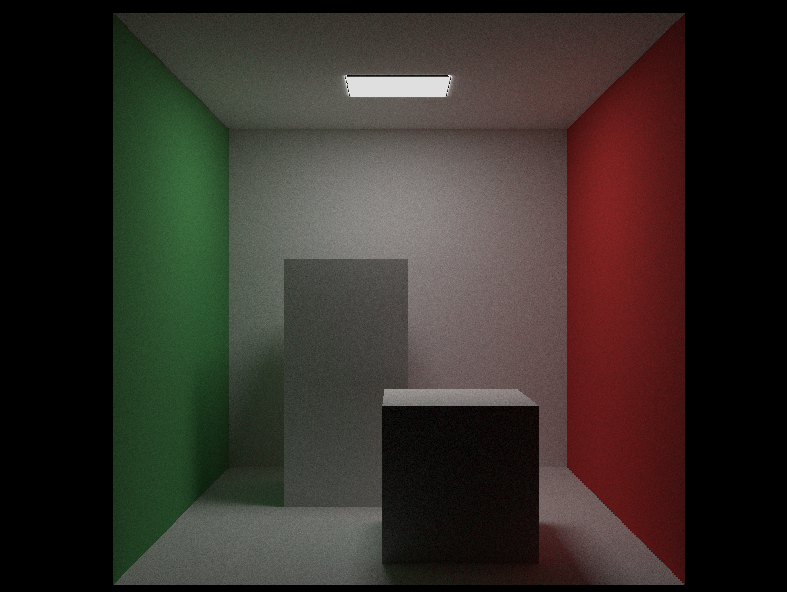


Figure 4 Cornell Box – 5,000 samples per pixel

**2.4.2 Metal (Reflection)**

Whereas a lambertian material partially absorbs light, a metal material should reflect it. As such, if a ray meets an object with a metal material, it should bounce off in the opposite direction. **Figure 5** illustrates the equation for reflectance:

A picture containing text, watch

Description automatically generated

Figure 5 Diagram of reflectance (Shirley, 2020)



Figure 6 Cornell Box - 5,000 samples per pixel

**Figure 6** shows the effect of the metal material when applied to the back wall of the Cornell Box. Here is has been set to have a maximum refractive index so in essence it acts as a mirror.

**2.5 Lighting**

Lighting was successfully implemented by creating an emissive material. If the ray intersects with the light source, then it does not scatter. It terminates at that point and returns the colour value of the light (white). As previously stated, all of the returned colour values from the intersect points of each ray are averaged together. The effect of the light is created by raising this average as it adds the maximum colour value (white) to the average.

**2.6 Bounding Volume Hierarchy**

A BVH was implemented to reduce the amount of redundant ray paths. A BVH works by subdividing (bounding) the primitives with a scene to create a hierarchy. Primitives are stored in the leaves and each node stores a bounding box of the primitives in the nodes beneath it (Pharr & Humphreys, 2010). This means that the first check for a ray is to see if it has hit one of these bounding boxes. If it hasn’t, then it means that the ray won’t be intersecting any objects, thus it can just return the background colour.

**3. Outcomes**

(Christensson, 2019) defines sampling as the smoothing of edges and colours in digital images. Sampling was one of the core deliverables of this path tracer project. Appendix A illustrates multiple instances of the same image but with a variable number of samples per pixel. **Table 1** shows the samples per pixel versus the time taken for the program to finish execution of the Cornell Box scene. It is clear to see that there is a linear relationship between the samples per pixel and time taken to render. This has been visualised by the scatter graph at the end of Appendix A which has a linear trendline. It is worth noting that this program is running on my personal desktop which utilises an AMD Ryzen 2700X CPU.

|  |  |
| --- | --- |
| Samples Per Pixel | Time (minutes) |
| 5 | 3.3 |
| 25 | 4.1 |
| 100 | 5.5 |
| 500 | 12.7 |
| 1000 | 22.4 |
| 5000 (metal back) | 87.2 |
| 5000 (lambert back) | 93.8 |
| 10000 | 186.5 |

Table 1

**4. Limitations and Future Improvements**

Initially, the program was to utilise the ray-triangle intersect algorithm proposed by (Möller & Trumbore, 1997). This was partially implemented however there were some complications. If a ray intersected with a triangle which ran directly along the x axis, then it would not detect the collision and return the correct colour. **Figure 7** shows a Cornell Box, rendered using (Möller & Trumbore, 1997) ray triangle intersection algorithm. As you can see, there is no colour for the back of the box, and it has been set such that two triangles form the back side and have the same colour as the top and bottom of the box (Appendix B).

If the project were continued further, then I would endeavour to create a more bespoke system and implement a wider array of features such as global illumination and loading of custom 3D models as was initially proposed for the project. Furthermore, multi-threading could have been implemented to further increase the efficiency of the program.

A picture containing funnel chart

Description automatically generated

Figure 7 Result of (Möller & Trumbore, 1997) ray triangle intersection implementation

**5. Conclusion**

The project achieved an implementation of a path tracing system with BVHs, materials, lighting, and sampling. The program can produce high quality, realistic, ray traced images (Appendix A). Direction for future improvements of the program has been identified should the program be carried forward.

# **6. References**

Christensson, P., 2019. *Anti-Aliasing Definition.* [Online]   
Available at: https://techterms.com/definition/anti-aliasing  
[Accessed 16 01 2021].

Dusterwald, S., 2016. *Path Tracing vs Ray Tracing.* [Online]   
Available at: https://www.dusterwald.com/2016/07/path-tracing-vs-ray-tracing/  
[Accessed 11 November 2020].

Gomila, L., 2014. *Simple and Fast Multimedia Library.* [Online]   
Available at: https://www.sfml-dev.org/index.php  
[Accessed 18 October 2020].

Microsoft Corporation, 2019. *Visual Studio IDE.* [Online]   
Available at: https://visualstudio.microsoft.com/  
[Accessed July 2021].

Möller, T. & Trumbore, B., 1997. Fast, Minimum Storage Ray-Triangle Intersection. *Journal of Graphics Tools,* 2(1), pp. 21-28.

Öqvist, J., 2015. *Path Tracing.* [Online]   
Available at: https://chunky.llbit.se/path\_tracing.html  
[Accessed 11 November 2020].

Pharr, M. & Humphreys, G., 2010. *Physically Based Rendering: From Theory to Implementation.* Second Edition ed. Burlington, Massachusetts: Morgan Kaufmann Publishers.

Shirley, P., 2020. *Ray Tracing In One Weekend.* v3.2.2 ed. s.l.:s.n.

# **7. Bibliography**

Bala, K., 2012. *Ray Tracing (Shading and Sampling).* Ithaca: Cornell University.

Bentley, J. L., 1975. Multidimensional Binary Search Trees Used for Associative Searching. *Communications of the ACM,* 18(9), pp. 509-517.

Christensson, P., 2019. *Anti-Aliasing Definition.* [Online]   
Available at: https://techterms.com/definition/anti-aliasing  
[Accessed 16 01 2021].

Dusterwald, S., 2016. *Path Tracing vs Ray Tracing.* [Online]   
Available at: https://www.dusterwald.com/2016/07/path-tracing-vs-ray-tracing/  
[Accessed 11 November 2020].

Gomila, L., 2014. *Simple and Fast Multimedia Library.* [Online]   
Available at: https://www.sfml-dev.org/index.php  
[Accessed 18 October 2020].

Herout, A. & Havel, J., 2009. Yet Faster Ray-Triangle Intersection (Using SSE4). *IEEE Transactions on Visualization and Computer Graphics,* 07 07, 16(3), pp. 434-438.

Hollasch, L. W., Coulter, D. & Bazan, N., 2017. *Rendering Pipeline.* [Online]   
Available at: https://docs.microsoft.com/en-us/windows-hardware/drivers/display/rendering-pipeline

Immel, D. S., Cohen, M. F. & Greenberg, D. P., 1986. A Radiosity Method for Non-Diffuse Environments. *ACM SIGGRAPH Computer Graphics,* 20(4), pp. 133-142.

Kajiya, J. T., 1986. The Rendering Equation. *ACM SIGGRAPH Computer Graphics,* 20(4), pp. 143-150.

Matusik, W., 2012. *Global Illumination and Monte Carlo.* [Online]   
Available at: https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-837-computer-graphics-fall-2012/lecture-notes/MIT6\_837F12\_Lec18.pdf  
[Accessed June 2021].

Metropolis, N. & Ulam, S., 2012. The Monte Carlo Method. *Journal of the American Statistical Association,* 44(247), pp. 335-341.

Microsoft Corporation, 2019. *Visual Studio IDE.* [Online]   
Available at: https://visualstudio.microsoft.com/  
[Accessed July 2021].

Möller, T. & Trumbore, B., 1997. Fast, Minimum Storage Ray-Triangle Intersection. *Journal of Graphics Tools,* 2(1), pp. 21-28.

Öqvist, J., 2015. *Path Tracing.* [Online]   
Available at: https://chunky.llbit.se/path\_tracing.html  
[Accessed 11 November 2020].

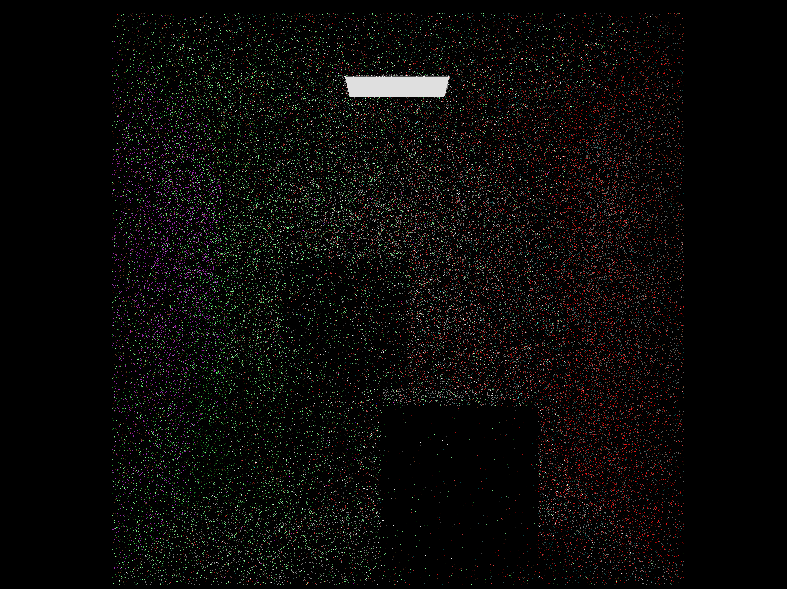
Pharr, M. & Humphreys, G., 2010. *Physically Based Rendering: From Theory to Implementation.* Second Edition ed. Burlington, Massachusetts: Morgan Kaufmann Publishers.

Shevtsov, M., Soupikov, A. & Kapustin, A., 2007. *Ray-Triangle Intersection Algorithm for Modern CPU Architectures,* Moscow: Intel Corporation.

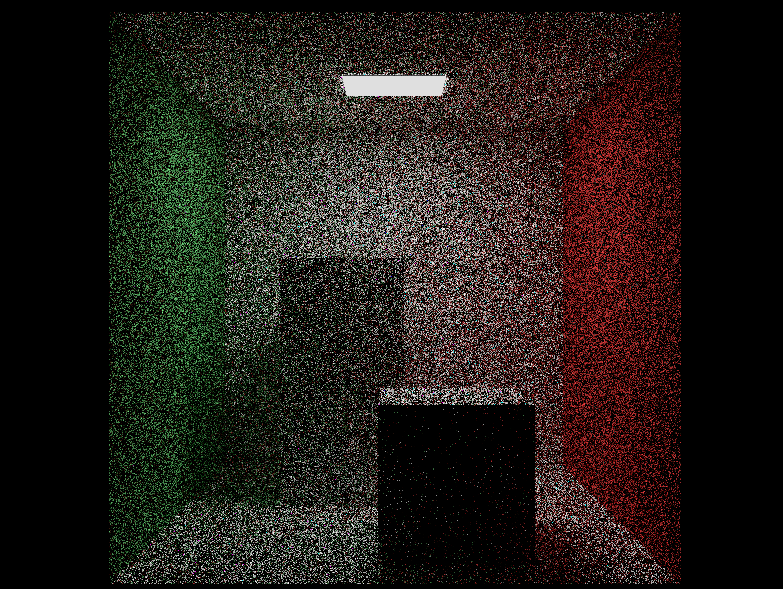
Shirley, P., 2020. *Ray Tracing In One Weekend.* v3.2.2 ed. s.l.:s.n.

Shirley, P., 2020. *Ray Tracing: The Next Week.* v3.2.3 ed. s.l.:s.n.

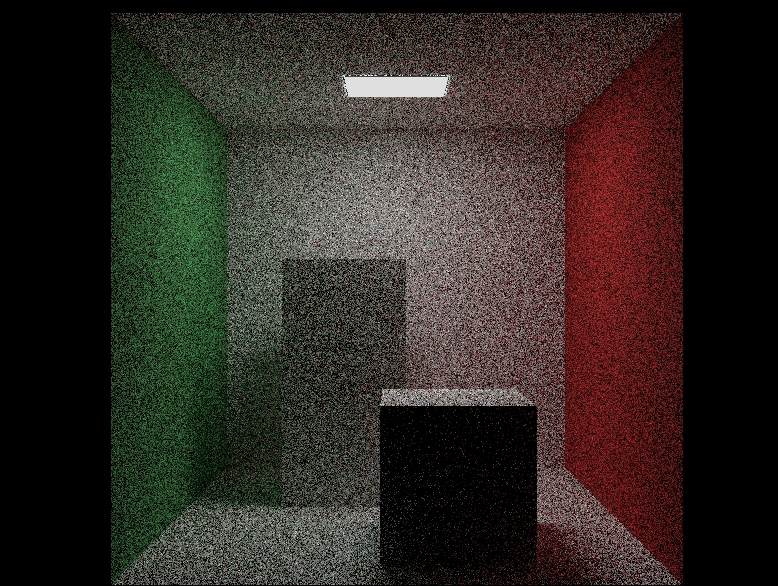
**Appendix A: Path Traced Images Produced**



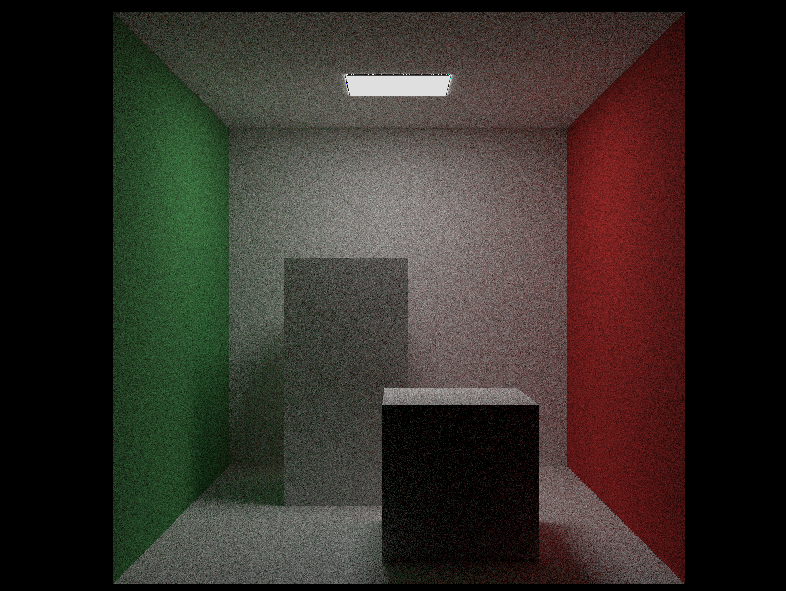
1 Sample Per Pixel



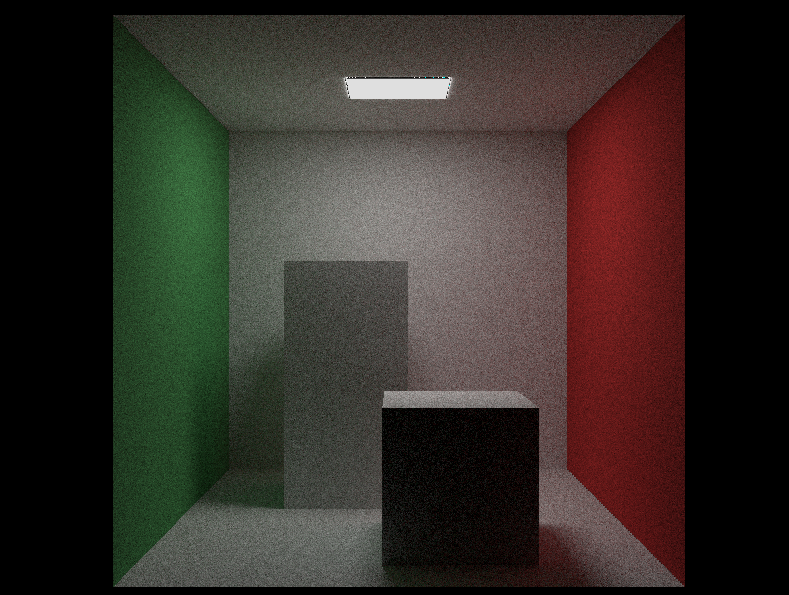
5 Samples Per Pixel



25 Samples Per Pixel



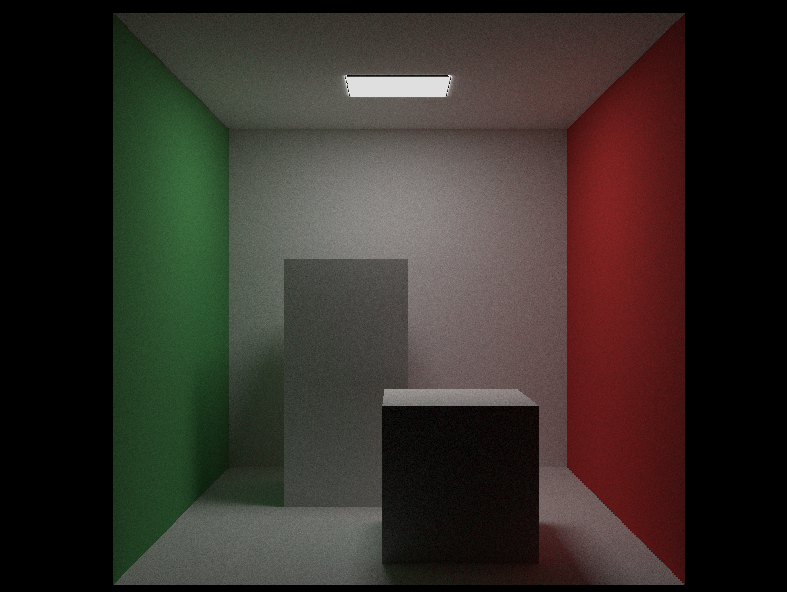
100 Samples Per Pixel



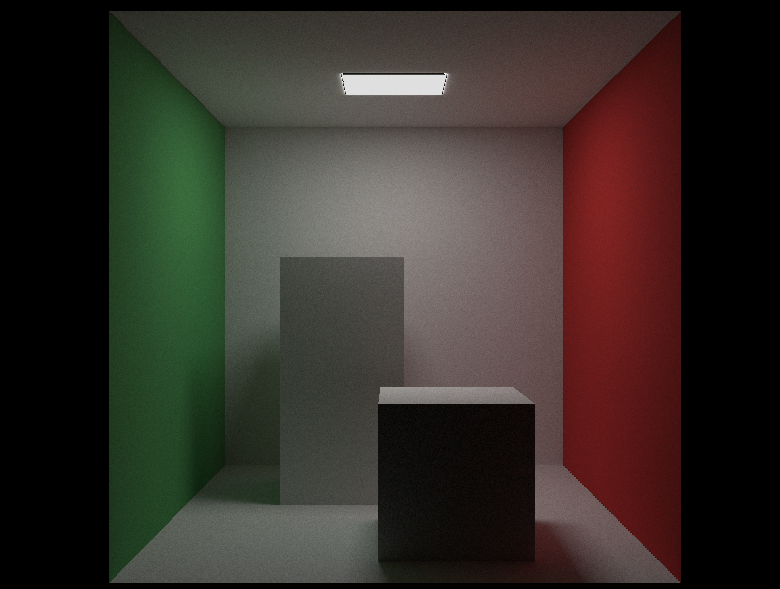
1000 Samples Per Pixel



5000 Samples Per Pixel – Reflective Back Wall



5000 Samples Per Pixel



10000 Samples Per Pixel

**Appendix B: Triangle Coordinates**

Text

Description automatically generated

**Appendix C: Project Timeline**

|  |  |
| --- | --- |
| October | Final proposal to be submitted by (12/11/2020)  Conduct research on different subject areas and narrow down project choices  Make choice of project and begin research |
| November | Project decided and meeting with supervisor, begin writing proposal first draft  Finalise proposal ready for submission  \*lost a week due to contraction of Covid-19, proposal submission extended until 19/11/2020  Submit Final Proposal (19/11/2020) |
| December | Implement SFML into C++ project  Construct Vector Class  Begin Research Report |
| January | Finalise Research Report  Research Report (14/01/2021)  Construct Spatial Data Structures for Geometry |
| February | Project Demo (05/02/2021)  Begin working on Ray Triangle Intersection  Implement Assimp Library |
| March | Implement metal PBR  Implement glass PBR |
| April | Implement Sampling |
| May | Implement BVHs |
| June | Implement lighting |
| July | Write final report |
| August | Research Report (03/08/2021)  Artefact (03/08/2021)  Final Report (03/08/2021) |