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

**Research Report**

**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. This research report aims to explore methodologies surrounding ray-triangle intersection, ray intersection acceleration, and image/pixel sampling in hopes to find information to aid in the construction of an efficient path tracing system. Path tracing uses a lot of computing power and appears to be the way forward in terms of realistic computer graphics; thus an efficient solution is required for this problem.

**Keywords**: Path Tracing; Bounding Volume Hierarchy; KD-Tree; Ray-Triangle Intersection; Ray Intersection Acceleration; Anti-Aliasing; Sampling;

**1. Introduction**

Simply put, 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).

3 dimensional models are comprised of series of polygons, most commonly triangles. As such, ray-triangle intersection is a critical part of this research as it will determine whether a ray has intersected an object in the scene as it will detect if a ray has collided with any of the polygons which constitute the 3D model in the scene. (Möller & Trumbore, 1997) present a “clean algorithm for determining whether a ray intersects a triangle”.

Further research into ray intersection has led to researching into bounding volume hierarchies (BVH) and K-Dimensional trees (KD-trees) both of which are a form of ray intersection acceleration. Both methods are detailed at length by (Pharr & Humphreys, 2010). Similarly, (Shirley, 2020) also uses BVHs in his implementation of a ray tracing system. This shows a trend towards the implementation of BVHs over kd-trees which requires more research to determine which is best suited to this project.

As (Dusterwald, 2016) states, thousands or even hundreds of thousands of rays are fired into a scene in path tracing. This leads to increased computing times while the paths of each ray are calculated. This research aims to create a path tracing system with an optimum efficiency. To accomplish this, Möller & Trumbore’s ray-triangle intersection algorithm could be combined with the use of a ray intersection acceleration methodology such as a BVH or KD-tree as discussed by (Pharr & Humphreys, 2010).

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.

Research Questions/Objectives:

* Can a path tracer be implemented on a Central Processing Unit (CPU)?
* What are bounding volume hierarchies?
* What are KD-Trees?
* Can BVHs or KD-Trees be used in conjunction with ray-triangle intersection to improve efficiency?
* What is sampling?

**2. Research Methods**

Due to the ongoing coronavirus pandemic, research has been almost completely limited to secondary research from home via the internet. The main approach was to use the Google Scholar search engine to find academic research papers around the subject of path tracing, as well as researching for the subsystems that comprise such a system. Most of the data found in these papers was quantitative.

Many recent papers use ray-triangle intersection methods however, they base this research on older materials which use the methods researched by (Möller & Trumbore, 1997). This shows that it is critical to have a fundamental understanding rooted in the ray-triangle intersection algorithm by (Möller & Trumbore, 1997) in order to build upon this with my own research and experimentation with this implementation. This also proves that this paper is a qualitative source of information as it is widely renown.

A small amount of primary research was conducted using the videogame Minecraft to demonstrate the effects of sampling/anti-aliasing and can be seen in figures x and y.

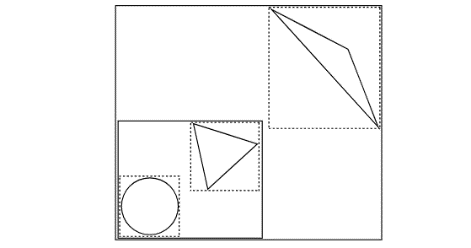
**3. Research Findings**

One of the most important aspects of a path tracing system is the tracing of the rays. Research into ray-triangle intersection has clearly shown that the algorithm (**Equation 1**) devised by (Möller & Trumbore, 1997, p.22) is superior, with numerous other papers referencing this work. (Möller & Trumbore, 1997, p.22) state that the memory savings are significant for triangle meshes, ranging from about 25 to 50 percent. From this statement it can be inferred that less memory would be required by using this algorithm. Alternatively, the remaining, larger percentage of free memory good be used for other processes.

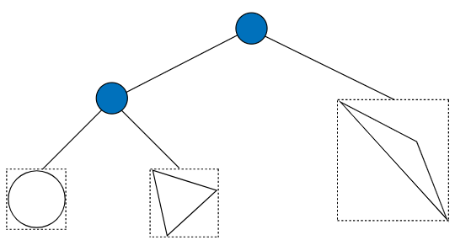
**Equation 1**

Initial research into ray intersection acceleration methods led to the discovery of bounding volume hierarchies and KD-trees. The subsequent research into both methods resulted in finding the book Physically Based Rendering by Pharr & Humphreys. Sections 4.3 and 4.4 present very clear understandings of the methods used to implement both BVHs and KD-trees respectively.

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). **Figure 1** and **Figure 2** represent a visualisation of how a BVH would be work. As a ray traverses through the tree, any time it doesn’t intersect a node’s bounds, the subtree beneath that node can be skipped as it knows that there is no primitive there and is being fired into empty space.



**Figure 1:** A small collection of primitives, with bounding boxes shown by dashed lines (Pharr & Humphreys, 2010).



**Figure 2:** The corresponding bounding volume hierarchy to **Figure 1**, from (Pharr & Humphreys, 2010)

Similarly, a BVH is also implemented in a ray tracing program by (Shirley, 2020). Shirley states that for any BVH, the code should always be in the form of:

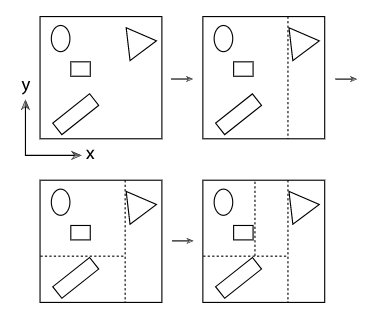
if (ray hits bounding object)

return whether ray hits bounded objects

else

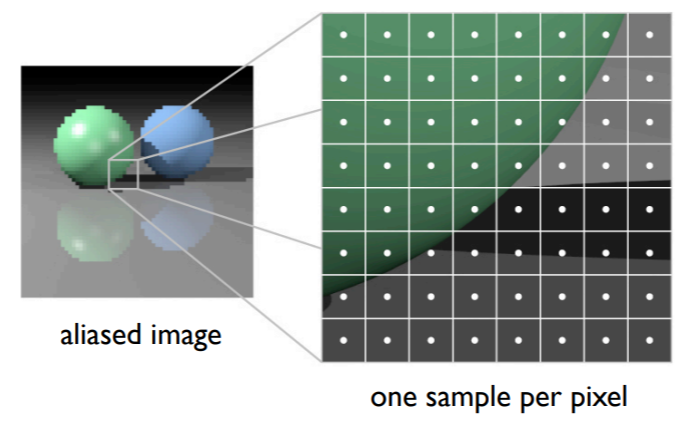
return false

KD-trees subdivide a scene with planes. Similarly to BVHs, it starts with a bounding box which encompasses the entire scene. If the number of primitives in the bounding box exceeds a defined threshold, the box is split in half by a plane. This continues until there are no sections which exceed this threshold. Primitives which overlap two areas are associated with both areas, whereas a BVH will assign the primitive to just one group (Pharr & Humphreys, 2010).

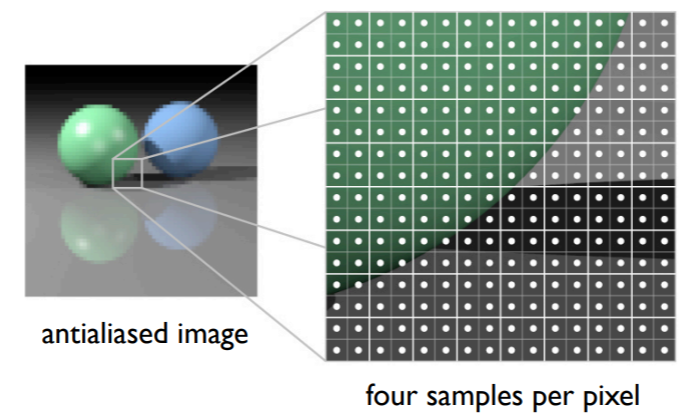


**Figure 3:** Representation of how a KD-tree would recursively split the bounding box of a scene (Pharr & Humphreys, 2010).

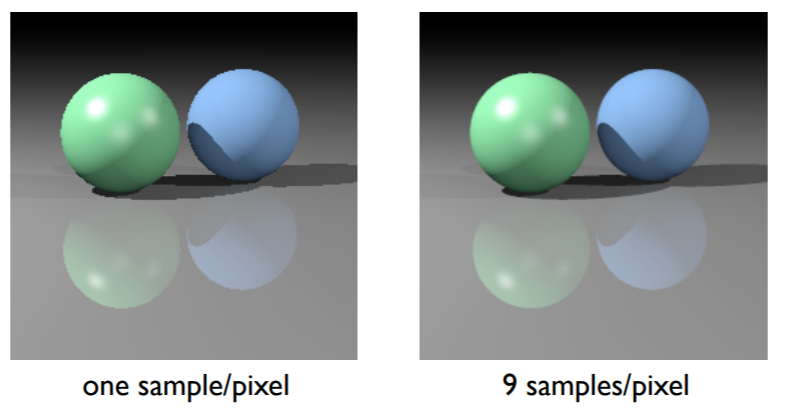
Initial research into sampling quickly led to anti-aliasing. Research for sampling was returning results not related to computer graphics rendering whereas anti-aliasing was much more specific. (Christensson, 2019) defines it as the smoothing of edges and colours in digital images. It helps blend colours in a natural-looking way. This is achieved by taking multiple samples per pixel to increase the accuracy of the value returned by the rays. **Figures 4**, **5**, and **6** from (Bala, 2012) support this.



**Figure 4**

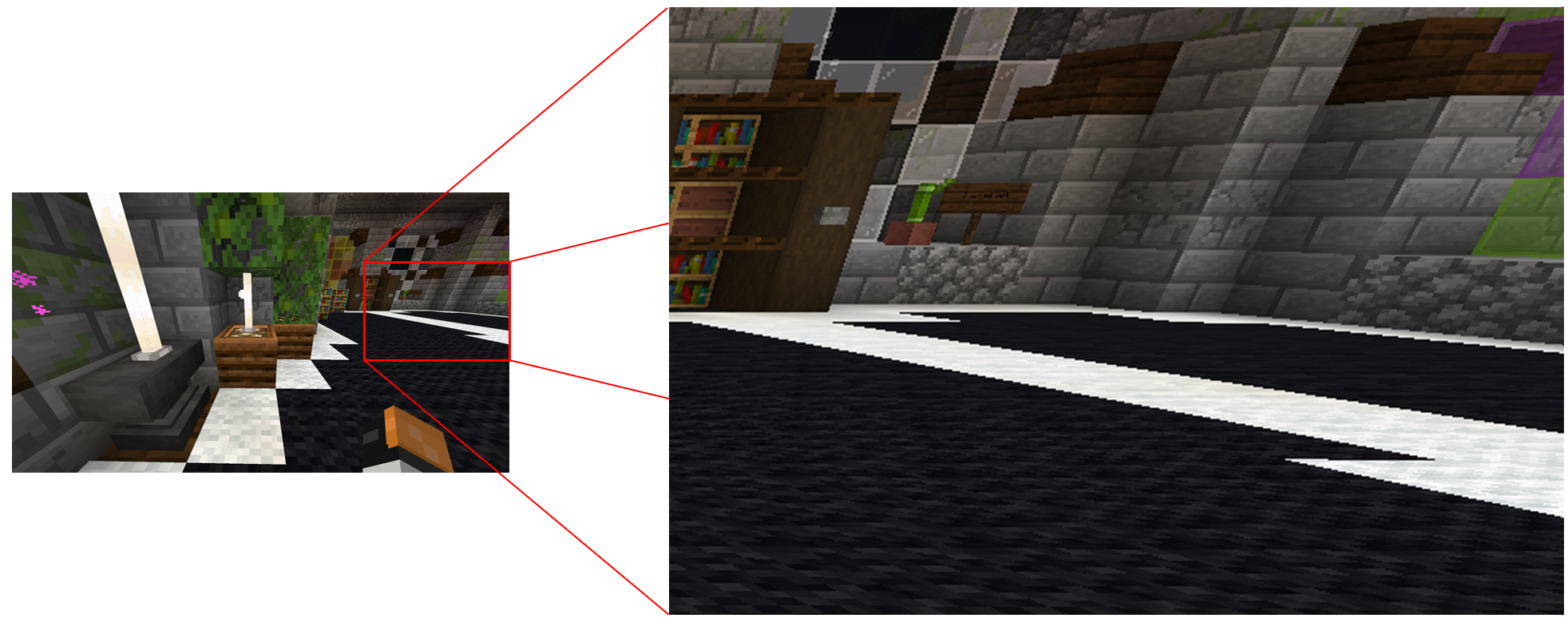


**Figure 5**

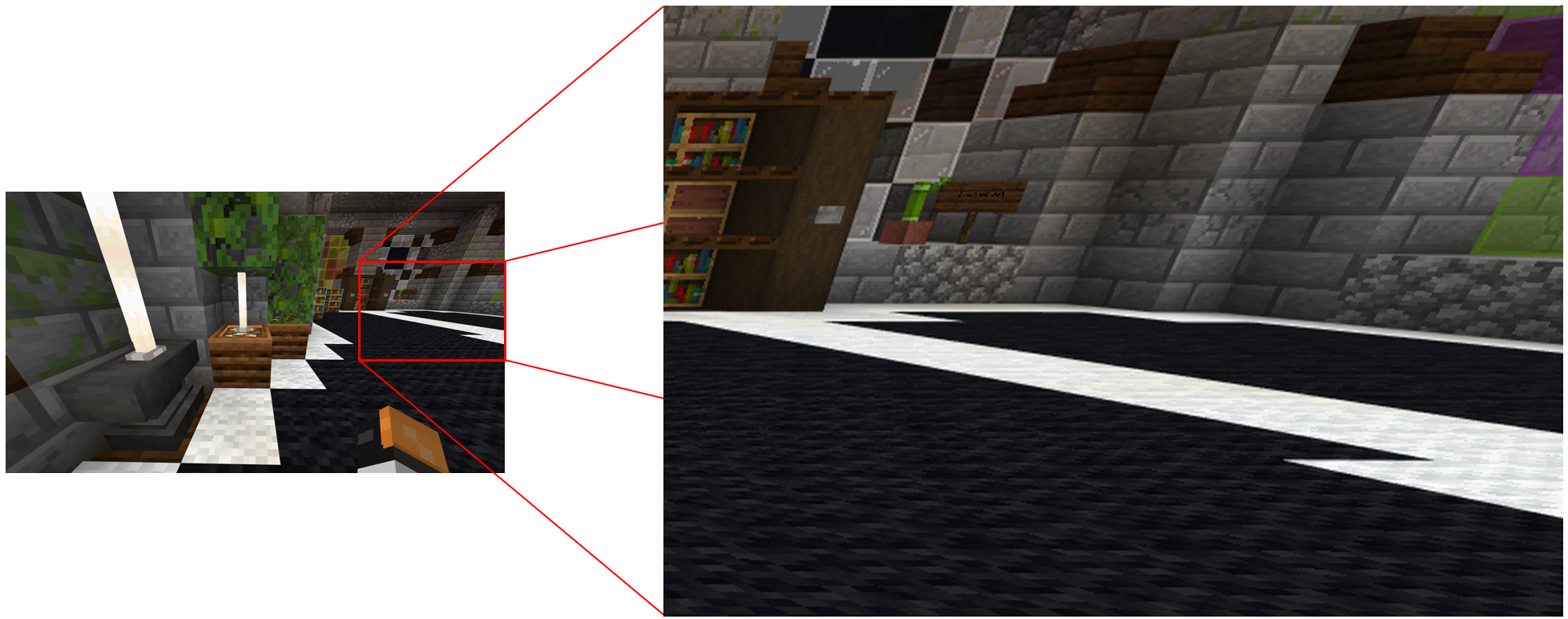


**Figure 6**

Additionally, primary research was conducted into anti-aliasing within Minecraft. **Figure 7** and **Figure 8** show the affects that anti-aliasing has within Minecraft.



**Figure 7:** Screenshot captured at 3840x2160 from Minecraft on Xbox Series X with anti-aliasing set to 1.



**Figure 8:** Screenshot captured at 3840x2160 from Minecraft on Xbox Series X with anti-aliasing set to 16.

These screenshots support (Bala, 2012) by showing a clear difference between the same scene being rendered with only 1 sample per pixel versus 16 samples per pixel. In **Figure 7**, the edges where the white and black carpets meet are jagged and frayed. Whereas in **Figure 8**, they appear straight and uniform. Minecraft on Xbox Series X caps the anti-aliasing at a maximum of 16 samples per second. Furthermore, Minecraft is a simple game with simple graphics, if this great of an effect can be created with a maximum of 16 samples per pixel, then this is clearly what my path tracing program should strive for.

**4. Conclusion and Recommendations**

Based upon this research it is evident that the ray-triangle intersection algorithm from (Möller & Trumbore, 1997) could be implemented in conjunction with a BVH as outlined by (Pharr & Humphreys, 2010) and (Shirley, 2020). As a BVH will not assign primitives to multiple bounding boxes, the path tracing system will be more efficient as some rays will be able to ignore a primitive if they initially collided with an overlapping bounding box which the primitive was not assigned to. The aim of this project is to create an efficient system, thus a BVH will be more optimum than a KD-tree for ray intersection acceleration.

I have also concluded that the path tracing system should not need any more than 16 samples per pixel as this is shown to be more than enough in a game which has low graphics.

**5. References**

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

Available at:

<https://www.cs.cornell.edu/courses/cs4620/2012fa/lectures/36raytracing.pdf>

[Last Accessed 20 January 2021].

Christensson, P., 2019. *Anti-Aliasing Definition.* [Online]   
Available at: <https://techterms.com/definition/anti-aliasing>   
[Last Accessed 18 January 2021].

Dusterwald, S., 2016. *Path Tracing vs Ray Tracing.* [Online]   
Available at: <https://www.dusterwald.com/2016/07/path-tracing-vs-ray-tracing/>   
[Last Accessed 16 January 2021].

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

[Last Accessed 20 January 2021].

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

Available at:

<http://www.pbr-book.org/>

[Last Accessed 20 January 2021].

Shirley, P., 2020. *Ray Tracing: The Next Week.* v3.2.3 [Online]

Available at:

<https://raytracing.github.io/books/RayTracingTheNextWeek.html>

[Last Accessed 19 January 2021]

**6. Bibliography**

Bentley, J. L., 1975. Multidimensional Binary Search Trees Used for Associative Searching. *Communications of the ACM,* 18(9), pp. 509-517, doi:10.1145/361002.361007

[Last Accessed 20/01/2021]

Gomila, L., 2014. *Simple and Fast Multimedia Library.* [Online]   
Available at:

<https://www.sfml-dev.org/index.php>   
[Last 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.

[Last Accessed 18 January 2021]

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.

[Last Accessed 10 November 2020]

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

[Last Accessed 10 November 2020]

Öqvist, J., 2015. *Path Tracing.* [Online]   
Available at: <https://chunky.llbit.se/path_tracing.html>   
[Last Accessed 11 November 2020].

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

Available at:

<https://www.graphicon.ru/html/2007/proceedings/Papers/Paper_46.pdf>

[Last Accessed 18 January 2021]

Shirley, P., 2020. *Ray Tracing In One Weekend.* v3.2.2 [Online]

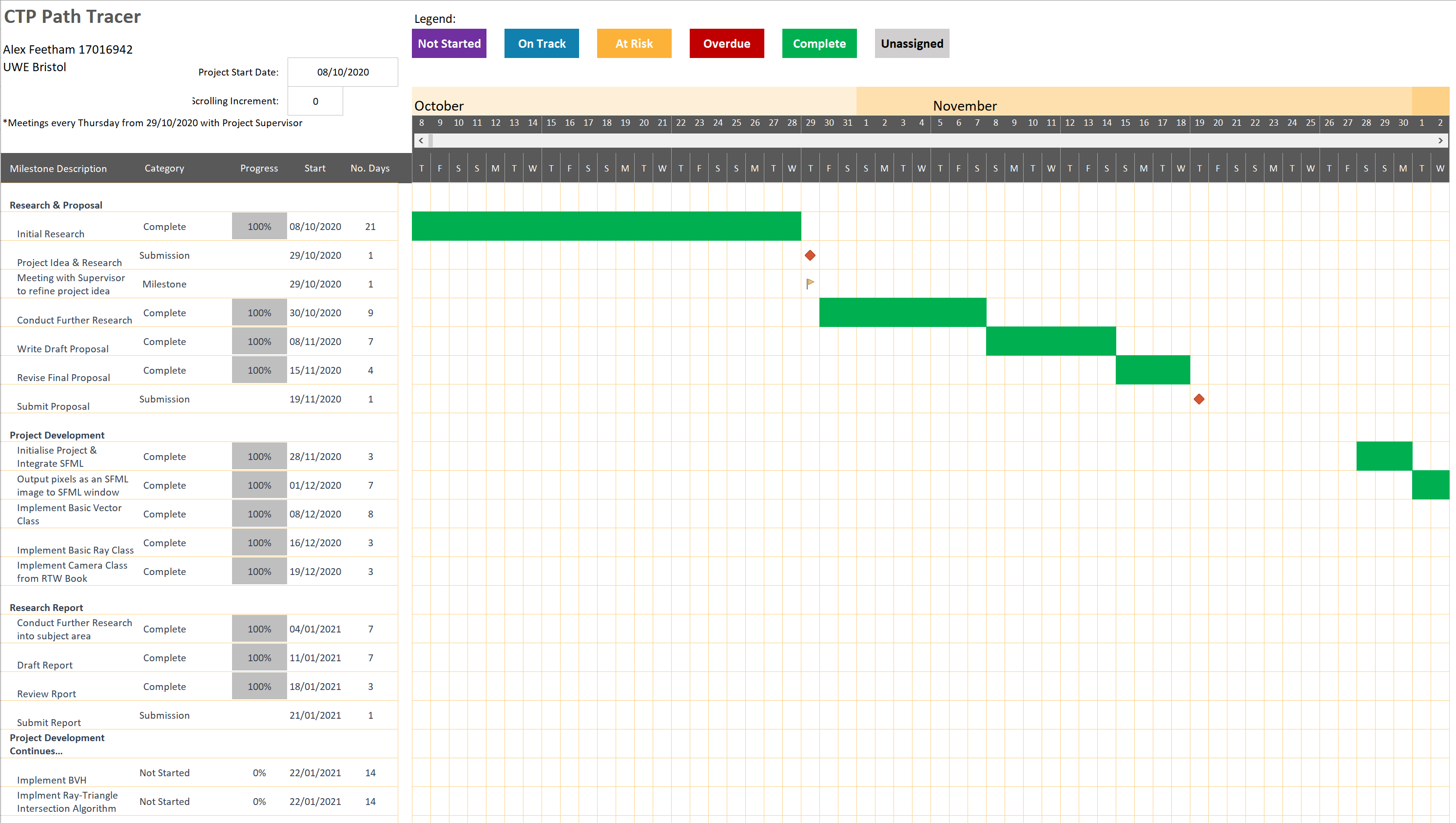
Available at:

<https://raytracing.github.io/books/RayTracingInOneWeekend.html>

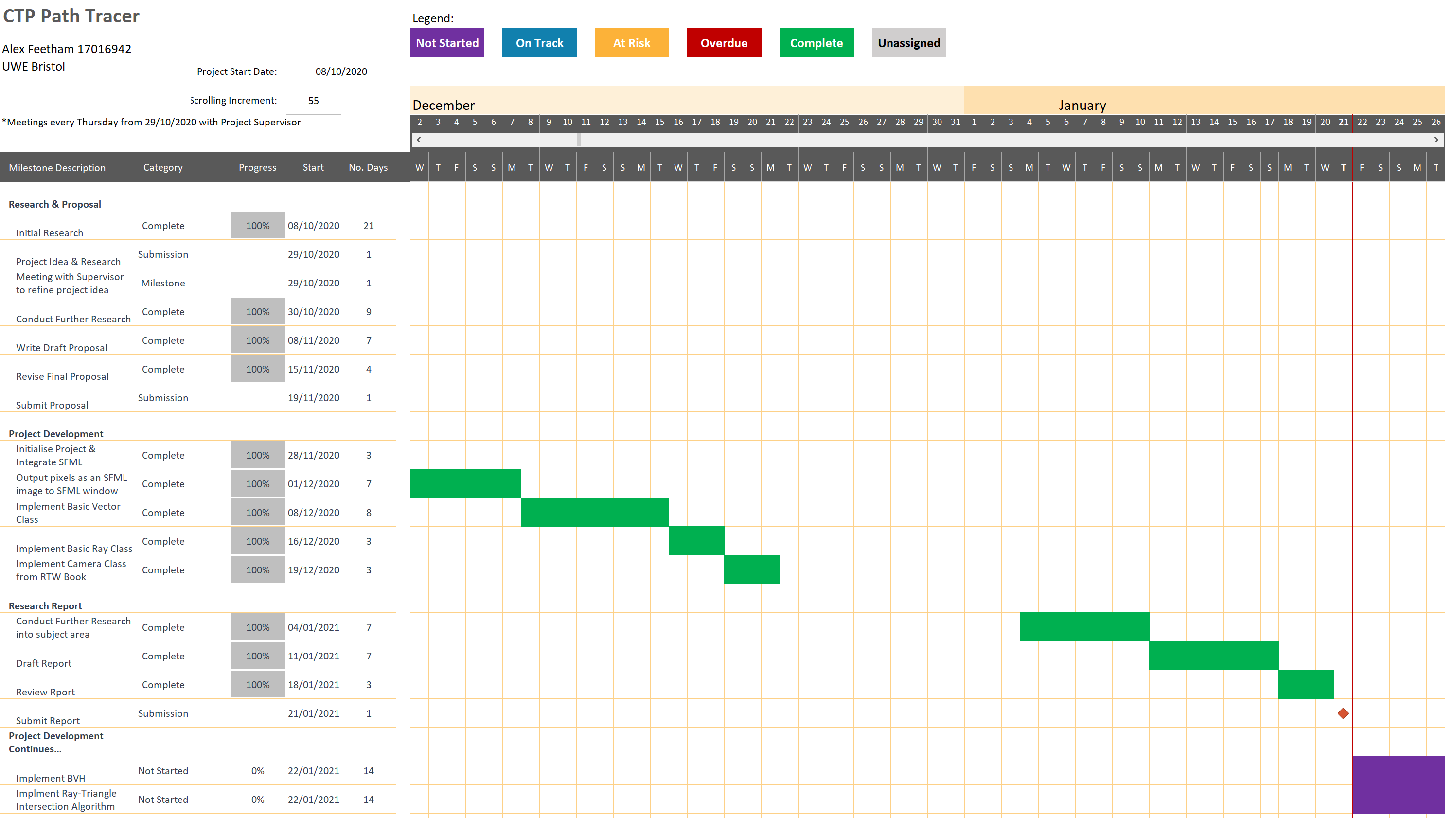
[Last Accessed 2 December 2020]

**Appendix A: Project Log**

To monitor progress throught the project I have decided to implement a gantt chart by modifying a template to suit my needs. Thus far, the project has followed a linear development path. However, as the project progresses and more tasks arise and begin to be implemented, more items of the chart will begin to overlap. The excel file will be included alongside this document.



**Appendix A – Figure 1:** Gantt Chart section between 08/10/2020 and 02/12/2020



**Appendix A – Figure 2:** Gantt Chart section between 02/12/2020 and 26/01/2021