Comparing Lighting Methods Using GPU Hardware Accelerated Real Time Ray Tracing

Páraic Bradley 180165038

May 2022

MComp Computer Science with Industrial Placement (Games Engineering)

Supervisor: Rich Davison

Word Count: 12636

Table of Contents

Abstract

When implementing ray traced lighting effects it is important to understand how the scenes in which these effects are deployed will affect their visual impact and performance metrics. This paper explores implementation of ray traced reflections, shadows, global illumination and ambient occlusion shaders in order to determine ideal optimisations and combinations of effects in different archetypical graphical scenes. The strengths and weaknesses of each effect is evaluated based on their performance measured against other ray traced effects and rasterised equivalents, as well as their subjective visual improvements. The results of this project determine an ideal combination of ray traced graphical effects in different scene types, including outdoor urban environments, outdoor rural environments and indoor environments.

Declaration

"I declare that this dissertation represents my own work except where otherwise stated."

Acknowledgements

I would like to thank supervisor Rich Davison for his help and guidance throughout the project.

Figures

1.0 Introduction

1.1 Motivation and Context

The technique of ray tracing is inspired by how light travels in real life. The ultimate goal is to simulate how each ray of light would actually behave in order to produce physically accurate lighting techniques. The computations involved in the process are prohibitively computationally expensive. It is because of this that real time ray tracing has been the holy grail for the computer graphics community for many years. **(Purcell, Buck, Mark and Hanrahan, 2005).** With the release of Nvidia's Turing GPU architecture in 2018, ray traced lighting techniques are now practical to achieve in real time in video games. This new wave of graphics cards specialised "RT cores" designed to speed up the computation ray intersections. This has enabled a revolution of interest in using ray tracing to achieve lighting fidelity in real time that would have previously been restricted to offline, pre-calculated rendering.

The technique of ray tracing can be used to simulate a range of graphical effects, including reflections, global illumination, shadowing and ambient occlusion. Not all of these effects are created equal, each one will have its own unique performance costs and visual improvements depending on the scene in which they are deployed. In the 2022 videogame "Dying Light 2", enabling ray traced global illumination alone reduces the average framerate by 66%. Ray traced global illumination can give dramatic visual improvements over traditional approaches, as it allows accurate simulation of how light bounces around an environment even with dynamic light sources and a changing time of day. Prior to ray tracing, it was common for global illumination to be restricted to static scenes with fixed lighting, or to be of a much lower quality in dynamic scenes **(Wald et al., 2002).** Ray traced reflections can help eliminate the frequent distracting artifacts with screen space reflections in common modern approaches **(Macedo, Serpa and Rodrigues, 2018).**

The problem that this project sets out to fix is that there is currently a lack of understanding in the literature base about when and where specific ray traced techniques should be implemented to maximise performance and visuals. Ray traced shadows may provide more fine detail that rasterised approaches, however in a wide open area with a fast moving camera, these improvements are unlikely to be noticed. In situations like this, it may be advisable to simply use traditional rasterised shadows and enjoy the increase in performance. This project sets out to explore custom implementations of four ray tracing techniques, reflections, shadows, global illumination and ambient occlusion and to investigate optimisations to be made for each and how these optimisations affect visuals and performance across a variety of scenes. The ultimate goal of this process is to create a set of guidelines for developers on which ray traced graphical effects, or combinations of effects are ideal in different types of scene.

1.2 Project Aim and Objectives

Aim: To implement and compare hardware accelerated real time ray tracing algorithms for various lighting techniques.

Objectives

1. **Identify four existing ray tracing algorithms for real time lighting techniques from the literature.**

   The four identified ray tracing techniques are: global illumination, reflections, shadows and ambient occlusion. These four techniques were selected to showcase a range of graphical effects which can be achieved using ray tracing. Each of these different techniques may be approached in different ways and offer opportunities to consider various optimisations.

   This objective has been updated slightly since the original project proposal. At that time, only three algorithms had been selected: global illumination, shadows and reflections. During the project the scope was expanded as development proceeded ahead of schedule. Ambient occlusion was added to offer a further point of comparison to the selected techniques and to cover a broader range of the possibilities ray tracing affords.

2. **Implement shaders for these ray traced lighting techniques to run in Nvidia's Falcor Engine.**

   The four shaders are to be implemented using the Slang shader language and will run in Falcor. Falcor includes an array of pre-existing render passes to compare against as a baseline.

   This objective also changed following submission of the project proposal. Early in development the decision was made to pivot from Unreal Engine 4 to Nvidia's own Falcor Engine which they use for their internal ray tracing research and development. Falcor is an ideal software solution for academic projects such as this one, and further reasons on the pivot will be detailed in later chapters.

3. **Identify and implement a range of 3D graphical scenes to highlight the strengths and weaknesses of each lighting method.**

   Each ray tracing algorithm will have its own strengths and weaknesses, and these will be highlighted by comparing them across a range of different scenes. The selected scenes cover indoor, outdoor urban and outdoor rural settings.

   The scenes are from Nvidia's ORCA (Open Research Content Archive). These are professionally made scenes used in industry for testing similar graphical effects and are free to use for research purposes.

This objective was updated over the course of development. The original objective stated that the scenes were to be handmade, however the decision to pivot to ORCA gave higher quality scenes and freed up development time to implement an ambient occlusion shader.

4. **Evaluate each method and combination of methods' performance and visual profile across different 3D graphical scenes.**

   Each method will be compared by its performance cost, measured using the scene's framerate, and by its subjective visual improvements. Consideration is also given to the optimisations each method allows, such as number of rays per pixel, and how these will impact framerate and visuals.

1.3 Structure Outline

This paper will begin with the technical background which will explore how this project's foundations in the literature base.

This is followed by the methodology section which covers what was done during the development of the project and how. The overview will cover the software engineering approach, before moving into explanations of the initial prototyping stages, the technology used, the testing done and the code structure of the project.

The methodology section ends with an explanation of the algorithms used for each of the four ray traced lighting techniques, as well as the temporal accumulation shader.

Next is the results and evaluation section which will detail the performance and visual profiles of each of the four shaders. This will discuss the strengths and weaknesses of each approach.

The evaluation section ends with a reflection on the successes and failures of the software engineering approaches deployed.

The conclusion begins with explaining the outcome of the project, highlighting the ideal ray tracing techniques to be used in each type of graphical scene.

This is followed by an evaluation on whether the aims and objectives were fulfilled, what was learned during the project and a list of future areas of work to be explored.

2.0 Technical Background

Exploring the ray tracing literature base gives a sense of how the technique has evolved, from early iterations where a single frame would be calculated over the course of several minutes **(Purcell, Buck, Mark and Hanrahan, 2005)**, to early real time ray tracing endeavours **(Wald et al., 2002)** and all the way through to where the industry is today, with ray accelerating hardware enabling complex ray tracing techniques in real time.

This project focuses on four ray tracing techniques identified from the literature base. The algorithms used are inspired by the following papers as well as "A Gentle Introduction to DirectX Raytracing" by Chris Wyman, https://cwyman.org/code/dxrTutors/dxr_tutors.md.html.

Ray traced reflections are discussed in the paper "Fast and Realistic Reflections Using Screen Space and GPU Ray Tracing—A Case Study on Rigid and Deformable Body Simulations" (**Macedo, Serpa and Rodrigues, 2018).** This is a relatively recent paper, coming out around the same time as hardware accelerated real time ray tracing was becoming a possibility. It specifies a hybrid approach with both screen space and ray traced reflections combined in order to maximise the trade off of visuals to performance. The specifics of this paper are not necessarily relevant to the general implementation desired for this paper, as it discusses more specifically reflections of deformable bodies.

Ray traced shadows are the topic of the paper "Soft shadow volumes for ray tracing" **(Laine et al., 2005).** It is important to note that this paper is from 2005 and predates modern ray tracing hardware acceleration. This paper focuses on offline rendering, rather than real time, however the fundamental algorithms on how to achieve ray traced shadows are still informative.

The paper "Interactive global illumination using fast ray tracing" **(Wald et al., 2002),** explores the advantages of ray traced global illumination over traditional methods and sets out how to achieve this effect. Ray traced global illumination is capable of capturing how light behaves as it bounces around a scene with dynamic light sources at a level of accuracy not possible using traditional techniques.

Ambient occlusion was identified as an ideal technique after the research period has already subsided and development had begun in earnest. This particular paper explores both ambient occlusion and directional occlusion: "Practical filtering for efficient ray-traced directional occlusion" **(Egan, Durand and Ramamoorthi, 2011).** Again this paper is from 2011, and the techniques specified are not necessarily compatible with this project's approach of

implementing the DirectX12 ray tracing API. Especially, the complex specialised implementation in this paper is outside the scope of the project. However, it served as a fundamental foundation in understanding what ambient occlusion is, and how to achieve the effect using ray tracing.

"Understanding the efficiency of ray traversal on GPUs" **(Aila and Laine, 2009)** is an important paper as it ensures an understanding of the fundamental maths involved in firing rays into the environment and calculating points of intersection. In this project the DirectX12 API abstracts most of these complex calculations away from the developer, so it is easy to forget these important concepts. Reading this paper gave a better appreciation of how ray tracing works at a basic level and encouraged the creation of better, more optimised code.

Exploring the literature base instilled confidence in this project that ray traced lighting techniques can achieve results impossible with rasterised approaches. This research reinforced the notion that providing conclusions about how to intelligently implement ray tracing techniques across different archetypical graphical scenes is a valuable process.

3.0 Methodology

3.1 Overview

The implementation of four shaders proceeded using an agile software development cycle. Working versions were to be developed in weekly sprints as to be ready to be showcased in weekly stand-up meetings with the project supervisor. Weekly deadlines were not always met, and meetings could not occur over the easter break, which covered the majority of development, however this did not hinder the completion of the project and work continued to proceed smoothly.

3.2 Initial Development Stages

At the beginning of the timeline, the research stage ran concurrently with the creation of the initial prototypes. The earliest prototypes did not run in real time on the GPU, but involved frames being pre calculated on the CPU with code written entirely in C++. Before committing to implementing a complex ray tracing technique, such as global illumination, these first tests involved simply tracing a single ray per pixel and checking if it intersected with a sphere to be drawn to the screen. These prototypes coincided with the research so as to nurture an understanding of the process of ray tracing and to gain confidence in using the technique. Doing this initial work was essential in order to be able to estimate the magnitude of work which could be undertaken throughout the development lifecycle and informed how the project would take shape and grow.

Eventually, over the course of multiple sprints, these tests transitioned from C++ code to shaders written in GLSL, enabling simple ray tracing running in real time on the GPU. The knowledge gained from these endeavours served as a base from which to begin to transition to development in earnest of the identified four ray tracing shaders in Unreal Engine 4.

3.3 Technology

The four identified ray tracing algorithms: global illumination, shadows, reflections and ambient occlusion, are implemented in the form of shaders to be run in Falcor. This is an engine developed by Nvidia for research projects such as this for their own internal teams as well as for the wider academic community. These shaders are written in the Slang shader language, which is based on the commonly used HLSL.

Additionally, the shaders are made using the DirectX12 ray tracing API which allows them to leverage the ray tracing accelerating hardware in Nvidia's Ampere GPUs. This results in a vastly more performant product, which is essential considering how computationally intensive ray tracing techniques are. The DirectX12 API and Falcor abstract away the underlying data structures for ray traversal, allowing for greater focus on optimising the shaders in question.

Work pivoted to Falcor and Slang from Unreal Engine 4 and HLSL early in development. There were a number of advantages Falcor held over Unreal Engine, so the decision was made to change. Because this happened early in development, very little work was lost and the transition was relatively smooth and painless. Falcor's system of 'render passes' is fairly straightforward and user friendly, so the process of passing data into and executing a shader is greatly simplified compared to Unreal Engine. Falcor contains numerous pre-existing render passes for comparison against the shaders developed for this project, including path tracers, ray traced techniques and traditional rasterised techniques. Unreal Engine's strong ray tracing options for a baseline comparison was one of its main advantages in the first place, so Falcor handily matches this. Additionally, Falcor supports the use of scenes from ORCA, which eliminates the need for Unreal Engine's scene editor. Falcor supports both Slang and HLSL, but Slang is already backwards compatible with all HLSL code, so any shader code written for Unreal Engine was able to be reused in Falcor.

3.4 Testing

The nature of this project restricted the possibilities for a robust testing framework. The goal when developing these shaders is to retrieve usable performance metrics in only a few specific contexts. It is required that each of these shaders executes across five scenes, and the success of failure of each of these is to be determined by the performance results. Testing and debugging occurred more informally, with visual bugs being stamped out as they appear during shader debugging. Once a shader is loading and exhibiting expected behaviour across all five scenes, development can happily proceed to an evaluation stage. The project certainly would have benefitted from a more comprehensive set list of tests, however the nature of the work being done somewhat prohibited this and robustness was not a high priority for these shaders. The most important outcome of development was to obtain valuable performance data in order to compare the various visual effects and this was achieved.

3.5 Code Structure

Falcor outputs an image using a succession of render passes. These can take inputs and execute shaders to produce some visual effect. The output from this render pass can then be plugged into another pass for further processing or simply output to the screen. Considering

each graphical effect as a consecutive and discrete block allows development to better focus on these individual elements and optimise them in isolation. This system makes development user friendly and relatively more straightforward, but the process of making render passes communicate can be complex. Taking the outputs from various passes and combining them logically and intelligently is often difficult to accomplish. The academic nature of this project focuses on individual shaders and effects, so it was not a necessity to have all shaders communicate. Had this been considered however, the project may have benefitted by having more robust output images showing multiple ray traced effects.

Render Pass projects in Falcor are written in C++ and are used to initialise any external variables which need to be passed into the shaders before executing them. The main class in these projects inherit from a Render Pass base class. The key methods to consider when are the setScene(), reflect() and execute() methods which are overridden from the parent class. SetScene() is used to initialise variables when a scene is loaded for the render pass. Reflect() is used to pass variables into the shader in question. Execute() runs the shader and can contain additional logic to control its behaviour.

The shaders are written in the Slang shader language and extensively implement the DirectX12 ray tracing API. When this API is used, the entry point is always the "ray generation" method. This method is run once per pixel and is typically used to define the logic of when rays should be fired. Each shader will have its own behaviour for how rays should behave and under what circumstances they should be fired. Once fired, a ray will enter into either the anyHit(), closestHit(), or miss() methods. As the name implies, the miss() method handles logic for when a ray does not intersect with any geometry at all, and usually simply means no action needs to be taken for that ray. AnyHit() handles logic for every intersection the ray has. The majority of logic for the four specified shaders however is included in the ClosestHit(), which would correspond to the hit point closest to the ray origin. Imagining the rays as actual real world rays of light, it would be expected that at this first hit point the ray should bounce or be absorbed in some way, so this method contains most of the logic for how rays should behave in these shaders. All four shaders makes use of several methods in a separate common utilities slang file and this enables the reuse of code to improve readability.

3.6 Ray Traced Reflections Algorithm

The first shader to be discussed gives a naïve solution for ray traced reflections which nevertheless creates some visually impressive and detailed results. The reflections shader traces one ray from the observer camera per pixel. In most cases, once this ray intersects with geometry, typically a triangle primitive, it writes the colour of the material at that point to the output image. However, when a ray intersects with a triangle whose roughness value is below a certain threshold, the shader shoots out another ray from the intersection point. This ray

follows the path of the first ray reflected off the hit point (calculated using the face normal at this point) and continues until in intersects with the next primitive. The output colour for the pixel is then calculated to combine both the colour at the reflection point and the termination point of the secondary ray. This technique allows for perfect mirror reflections which capture perfectly detailed images of the environment.

3.7 Ray Traced Shadows Algorithm

The next shader gives accurate ray traced hard shadows. Two approaches for this are implemented all within the one shader, and the two approaches can be swapped between using a checkbox on the Falcor UI. Keeping within the same shader allows the reuse of code and enables quickly switching between the two approaches without the need to load a new shader. These two approaches are included to explore different optimisations that can be made within the ray traced shadows technique and to evaluate their performance and visual profile.

The first approach traces one ray to every light source per pixel on the screen. For a single pixel, a ray is fired from the world position of the geometry at that point. This ray returns a value of 1.0 if it hits the light source and 0.0 if it does not hit light source. The colour to shade this pixel is calculated based on the distance to the light and its intensity. Repeat this for every pixel and the shader will produce a colour to darken this pixel by based on the effects of each light source, this is combined with the colour of the material at that point and sent to the output texture. The final result gives hard shadows which give fine grain detail accurate to the pixel.

The second approach fires a ray to only one randomly chosen light per pixel. The algorithm proceeds as before, but instead of cycling through every light in a scene, one is chosen at random and only that light affects that pixel. This produces an extremely noisy image, but this can be remedied using solutions such as temporal accumulation or a denoiser. This can lead to better performance in scenes with a large number of light sources.

3.8 Ray Traced Global Illumination Algorithm

The global illumination shader simulates how light bounces around an environment. In the same way the shadows shader contained two toggleable effects in one shader, so does the global illumination shader. In order to demonstrate the visual effects of ray traced shadows and direct lighting combined with the indirect lighting of ray traced global illuminations, the ray traced shadows are also called from the global illumination shader. This is done by toggling

a checkbox, so the performance of the global illumination shader can be assessed in isolation. The majority of the ray traced shadows code is reused efficiently with calls to a shared utility class, however some code was duplicated in the ray generation shader. This is one area for improvement.

If performing the global illumination alone, the algorithm works as follows. The ray generation shader cycles through each pixel on screen and the first step considers the lighting at that point. Similarly to the shadows algorithm, the shader samples a random light and computes the illumination at that point based on the distance to the light, the intensity and the diffuse material colour at that pixel. It is at this point that a shadow ray would be fired and taken into consideration, however as this explanation is covering the global illumination shader in isolation, the payload from the shadow ray is omitted from the calculation.

Once a base colour is determined for the pixel, an additional ray is fired to simulate one bounce of indirect light. A random bounce direction is computed and a ray is fired from the world position at that pixel in said direction. Once that ray intersects with a surface, the lighting and shadowing of that intersection point is computed as before, then the colour at that point is returned in the ray payload. The colour of the bounced ray is combined with the base colour of the ray' origin point to find the final output colour for that pixel. This is then written to the output texture.

Due to selecting a random bounce direction for the fired ray, the output image for global illumination is noisy and required reconstruction work through temporal accumulation or denoising, which adds to performance overheads.

3.9 Ray Traced Ambient Occlusion Algorithm

The ambient occlusion shader was added late into development, thanks to work on the other three shaders proceeding faster than expected. This flexibility was afforded by the agile methodology employed on this project. The ambient occlusion algorithm produces an inaccurate estimation of indirect lighting, but can be an convincing effect regardless in certain scenes, as will be discussed in more detail in the evaluation section.

A relatively simple algorithm, the ray generation method considers each pixel and fires a ray from the world position at each pixel. The direction of this ray is computed randomly, as with global illumination. The fired ray has a maximum radius at which intersections will be considered, if a hit occurs outside this radius, it will be discarded and the ray will be considered a miss. If a ray intersects with geometry it returns a value of 0, and if it misses it returns a value

of 1. This value returned is set to be the ambient occlusion colour at this pixel, with 0 for black and 1 for white. This is combined with the diffuse material colour at that point to get the output colour.

Because of the random ray directions, it is likely a pixel will be its ambient occlusion colour from black to white as a new ray is fired each frame. This produces a noisy output which must be reconstructed with temporal accumulation or denoising. Pixels with a lot of geometry nearby will be more likely to register hits and be shaded black on average, while pixels in sparse areas are more likely to register misses and be shaded white on average. As temporal accumulation averages out these results, the output is comparable to shadowing as areas under tables for example, are much darker than on top.

This shader can also be set from the runtime user interface to fire multiple rays per pixel, the ambient occlusion colour is then averaged between each of them to produce a number between 0 and 1. This results in a less noisy image, which can aid with denoising, but comes at the cost of performance.

3.10 Temporal Accumulation Algorithm

The temporal accumulation shader was born of necessity to produce a clean output image from the noisy outputs of the global illumination, shadows and ambient occlusion shaders. The shader itself is very simple, only a few lines to average the current frame and last frame, so most of the relevant logic is handled in the C++ code behind the render pass. By reading Falcor update flags, this code determines is there have been any updates in the scene, such as the camera moving or animations playing. If there is any motion on screen, the accumulation must be reset as otherwise the output will smear. The code also checks if the resolution of the scene has changed by the window being resized, for example. In this scenario the accumulation must be reset as the previous frame will not line up with the current frame, causing artifacts. The accumulation pass may also be toggled, so when it is disabled, the code simply copies the input frame to the output.

4.0 Results and Evaluation

Each shaders' performance is to be compared against the rasterised "Forward Renderer" render pass included in Falcor as standard. This pass is representative of traditional rendering techniques such as rasterised lighting and screen space ambient occlusion. It will server as a baseline to show the percentage performance costs for each ray traced technique and to highlight any visual improvements.

The performance for the forward renderer pass is detailed in the table below. Each test was performed at a resolution of 1920 x 1080 over a 60 second period.

Forward Renderer:

Scene	Arcade	Bistro Exterior	Bistro Interior	Emerald Square	Sun Temple
Average Framerate (fps)	762	258	N/A	90	410

One anomalous result to note is the bistro interior map. This map is unfortunately incompatible with the forward renderer pass due to issues with the material data for the scene and a bug in Falcor's render pass. This means the scene cannot be used for performance comparisons but may still be included to highlight the visual characteristics of the ray tracing techniques in different environments.

The number of triangles for each scene is given below. This will impact the performance of each scene. As the scenes from ORCA are not necessarily developed by the same individuals, the data is not in a consistent format. The number of triangles for the arcade scene is unknown and the details for the sun temple scene only lists the number of vertices.

Scene	Arcade	Bistro Exterior	Bistro Interior	Emerald Square	Sun Temple
Number	unknown	2,832,120 triangles	1,046,609 triangles	10,046,405 triangles	1,641,711 vertices

All ray traced render passes are preceded by a pre existing render pass included in Falcor which populates the G Buffer with data required for the shaders. As this is constant with each of the ray tracing shaders, it will not interfere with performance comparisons between them.

4.1 Evaluating Ray Traced Reflections Pass

The performance for the ray traced reflections pass is detailed in the table below. Each test was performed at a resolution of 1920 x 1080 over a 60 second period.

Scene	Arcade	Bistro Exterior	Bistro Interior	Emerald Square	Sun Temple
Average Framerate (fps)	962	490	620	148	614

These performance metrics give somewhat unexpected results in that this shader yields higher framerates than the baseline forward renderer. Conventional wisdom dictates that the ray traced reflections would give lower framerates than the rasterised render, however this can be explained by noting that this render pass covers reflections only. Although ray tracing is intensive, this particular render pass forgoes lighting, shadows, ambient occlusion and anti-aliasing to focus solely on reflections. Averaged across all scenes, excluding bistro interior for which no baseline performance data exists, the reflections pass saw a 45% increase in framerate. In a typical hybrid rendering approach, this reflections pass would be combined with other passes for rasterised lighting and other effects to create a more complete image, and this would bring the expected decrease in framerates.

The following two screenshots show a comparison between the ray traced reflections shader and the baseline forward renderer on the bistro exterior scene.

Figure 1: Bistro exterior scene with ray traced reflections

Figure 2: Bistro exterior scene with rasterised reflections

The ray traced reflections shader includes impressively detailed mirror reflections, whereas the forward renderer relies on a cubemap image. A cubemap may be convincing in certain contexts from a distance, but here the result is unconvincing. Cubemap are often paired with screen space reflections to capture finer grain detail by copying data from "screen space". One disadvantage this common technique brings is that it cannot show objects which are offscreen in the reflection and this can lead to artifacts and objects disappearing from reflections as the camera moves. The ray tracing shader circumvents these issues as the entire scene is loaded in the internal BVH structure, so it is not reliant on information from the screen space.

Increasing the roughness threshold for the shader can produce unusual effects as unexpected surfaces suddenly become reflective. However increasing the roughness threshold slightly on the sun temple scene showcases the robustness of the shader. The following screenshot depicts distorted reflections on the marble floor, which enables accurate reflections across an uneven surface, as the shader takes into account the face normal at each ray intersection.

Figure 3: Sun temple with ray traced reflections

An area the shader could be improved is nested reflections. Currently the shader has a fixed maximum ray depth of 1, so after one reflection, if a ray hits another reflective surface, it terminates. This is depicted in the following screenshot on emerald square, showing that the shop windows in the reflection are all black, whereas those in the camera's view are all reflective. Increasing the maximum ray depth from 1 to 2 would allow one additional bounce in a reflection but would bring a substantial performance cost. Care must be taken to enforce a maximum ray depth to prevent rays continuing infinitely.

Figure 4: Emerald square with ray traced reflections

This reflection shader takes a rather naïve approach and does not accurately take into account all the various aspects of the material the ray intersects with. Currently the shader only considers roughness and the material's diffuse colour, with no consideration for any specular element or emissive surfaces. Nor is there any way for the reflections to accurately depict the lighting of the scene. The shader could be improved by taking into account the lighting of the scene.

An advanced technique for further development would be transparent surface reflections. Currently in the shader reflective surfaces are treated as mirror surfaces, whereas in real life, for materials like glass some light would reflect off and some would pass directly through it. This can look unusual where all glass surfaces appear as mirrors, especially in the emerald square screenshot. Ray traced transparency reflections would allow light to continue through the glass and reveal anything behind, creating a more photorealistic image.

Additionally, to increase the performance of ray traced reflections, the level of detail in the BVH structure could be reduced and screen space reflections could be used for higher detail close to the camera. This gives the benefit of a higher framerate than pure ray traced reflections and fewer distracting artifacts than pure screen space reflection. This optimisation can be pushed further by having the reflections done at lower than native resolutions.

4.2 Evaluating Ray Traced Shadows Pass

There are two approaches to consider and compare for the ray traced shadows render pass. The first approach takes into consideration every light source in the scene, shooting one ray towards each per pixel. Performance decreases linearly with the number of lights in a scene. The second approach randomly selects one light source and only shoots one ray per pixel. This can dramatically increase performance, but it produces a noisy output, so it will require information to be built up over multiple frames to produce a stable output, by either the temporal accumulation pass or a denoising solution.

The performance for the "all lights per pixel" approach is detailed in the table below. Each test was performed at a resolution of 1920 x 1080 over a 60 second period.

Scene	Arcade	Bistro Exterior	Bistro Interior	Emerald Square	Sun Temple
Average Framerate (fps)	817	487	N/A	135	125

This data immediately highlights how this approach can have performance decrease dramatically as the light count increases. The arcade scene sees a 7% increase in performance versus the forward renderer baseline, for the same reasons discussed in the reflection shader evaluation. This is only a shadows pass, whereas the forward renderer includes a rasterised shadows pass, a lighting pass, ambient occlusion pass, anti-aliasing pass and more. Such a small 7% increase in performance is indicative of how intensive ray tracing techniques can be. The arcade scene includes only a small number of light sources, so the ray traced shadows technique used here is relatively performant. The sun temple scene sees a 70% decrease in performance versus the forward renderer baseline. This is because the sun temple contains many more light sources, and the current approach requires *n* rays to be fired per pixel in a scene with *n* lights, which can result in a massive decrease in performance. The bistro interior scene unfortunately fails to load with the ray traced shadow shader and this is a result of how lights are handled in this scene, there are no traditional lights, only some emissive surfaces, which are not supported by this shader. This is an area for future work.

The "all lights per pixel" approach results in a very stable output image which does not require information to accumulate over multiple frames. Contrastingly, the "single light per pixel" approach requires accumulation, so when the camera is in motion the output image is extremely noisy and unstable and the accumulation is constantly being cleared. In a video game setting, a stable output image is a necessity, as well as high framerates, so in scenes with few light sources the "all lights per pixel" approach may be preferrable. Alternatively, the use of a denoising solution could eliminate the need for temporal accumulation, so the increased framerates of "single light per pixel" method could be enjoyed.

The ray traced shadows shader can choose whether or not to show a material's diffuse colour, or simply output a greyscale image showing shadowed and lit areas. The following two screenshots demonstrate these two options. Going forward the ray traced shadows pass will be highlighted using the greyscale option to place greater emphasis on the shadows.

Figure 5: Arcade with coloured ray traced shadows

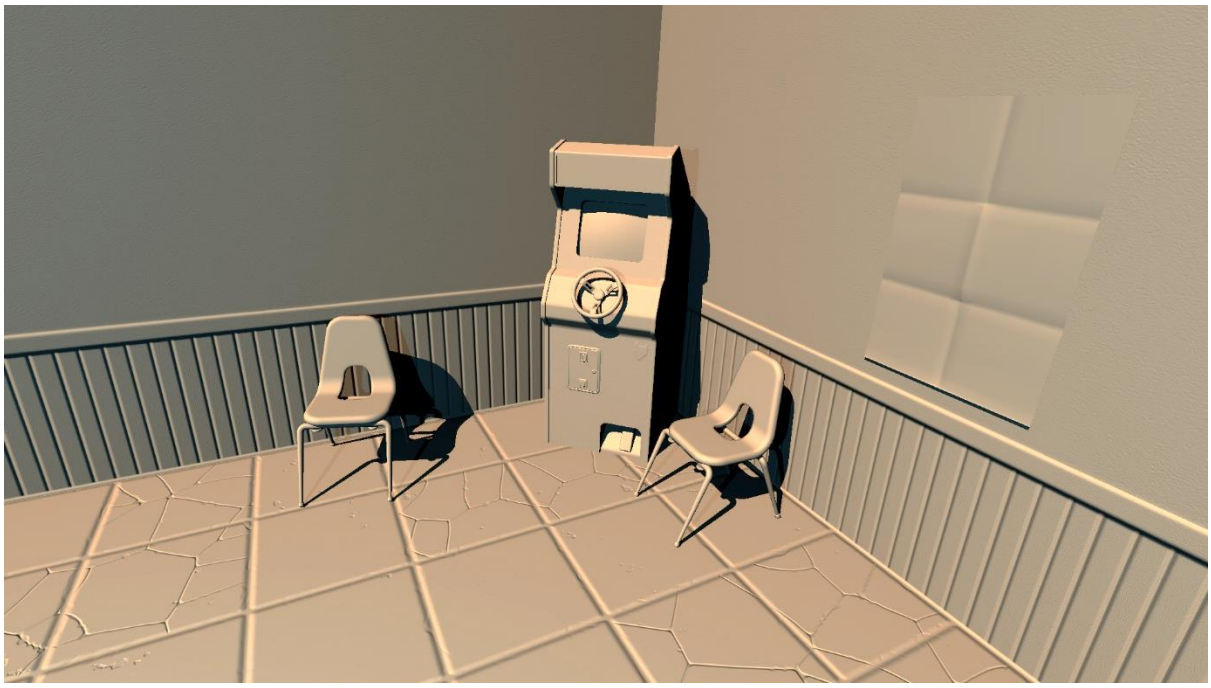

Figure 6: Arcade with greyscale ray traced shadows

Examining the output for the Arcade scene, it is noticed that the shader output produces highly detailed hard shadows. With multiple light sources, some areas are fully shadowed and some are only partially shadowed. This is noted in the previous screenshot underneath the leftmost chair as the intensity of the light sources, as well as their distance impacts how the shadows are drawn. Comparing this to the output of the forward renderer, the two are very similar and any advantages either way are not immediately apparent.

Figure 7: Arcade with rasterised shadows

However, on closer inspection the traditional rasterised approaches to shadowing can exhibit unusual behaviour, as illustrated in the screenshot below.

Figure 8: Arcade with rasterised shadows close up

The rasterised shadows approach appears to rely partly on screen space information, so when the orange chair is no longer in the camera's view, its shadow appears to disappear. The ray traced shadows shader does not exhibit this behaviour and preventing artifacts from screen space techniques is a major advantage in its favour.

Figure 9: Bistro exterior with ray traced shadows

The bistro exterior scene highlights a major problem with implementing solely direct lighting with no indirect component. This scene contains only one light source: the sun. Because of this, everything being hit by this light is fully illuminated and everything shadowed from the sun is completely dark. This problem is fixed by combining the ray traced shadows shader with the ray traced global illumination shader, so light from the sun bounces around the scene and illuminates shadowy area, just as light behaves in the real world. Alternatively, the ray traced shadows pass could be combined with some rasterised indirect lighting approach to achieve a hybrid rendering approach. This would sacrifice the benefits of ray traced global illumination but would boost the framerate. This approach would make sense for game developers creating scenes which would benefit most from ray traced shadows, such as indoor scenes with lots of fine details.

Figure 10: Emerald square with ray traced shadows

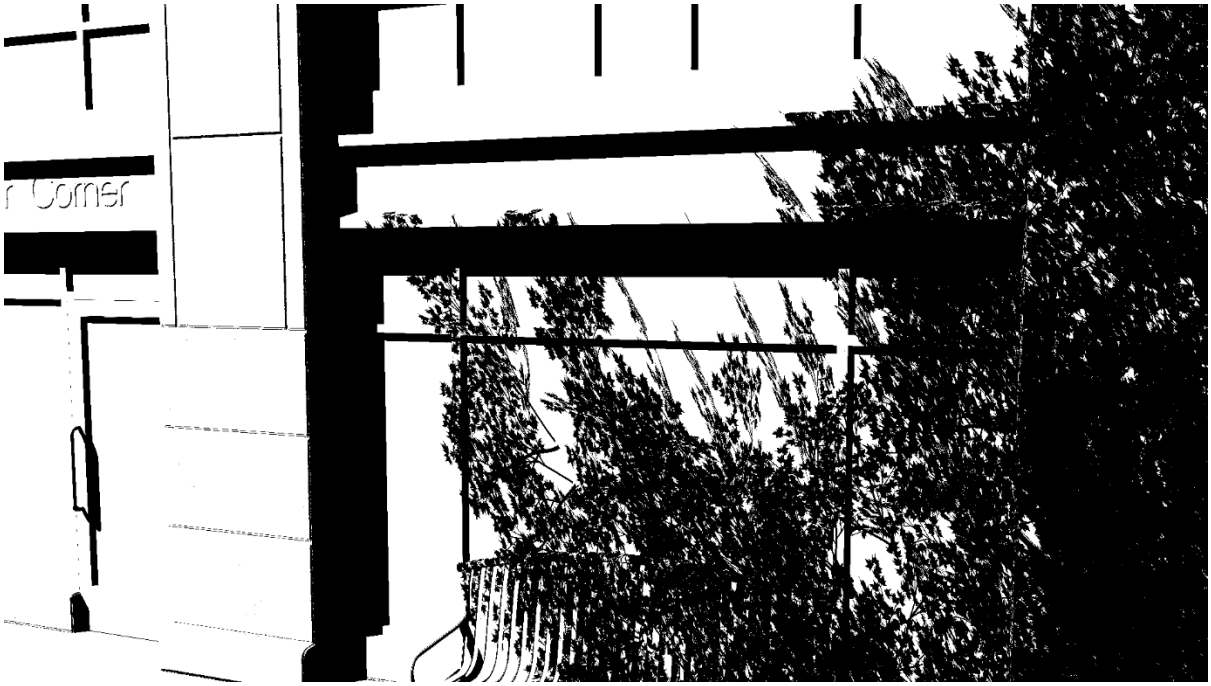

Figure 11: Emerald square with ray traced shadows close up

The emerald square scene highlights several of the strengths and weaknesses of the ray traced shadows shader. Ray traced shadows are able to capture a great amount of fine detail, which is especially highlighted by the foliage shadows in the screenshot above. The shadow cast from a nearby tree captures the detail of every leaf to a pixel accurate degree. Comparing this to the rasterised shadows approach, it can be noted that this level of detail is not present in the forward renderer screenshots.

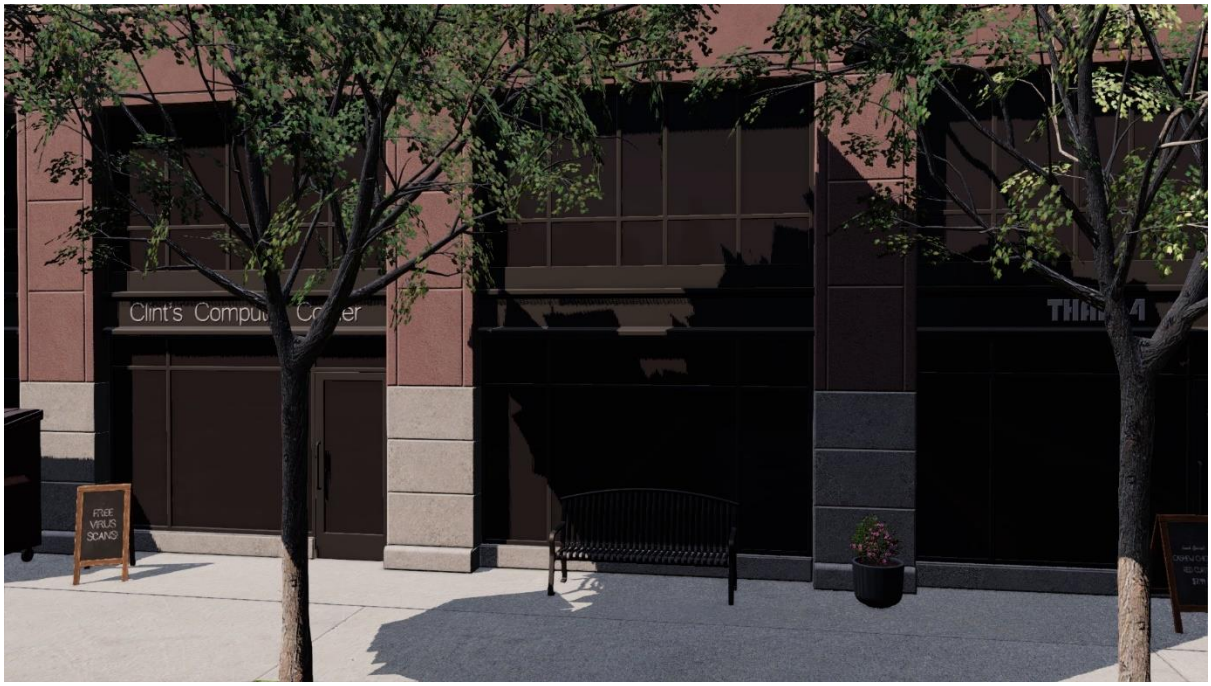

Figure 12: Emerald square with rasterised shadows

The shadows from the tree using traditional rasterised techniques are blocky and low resolution; this shows a clear advantage is visual clarity using ray traced shadows. However, it should be noted that this forward renderer is not necessarily representative of the best possible results using traditional shadow rendering techniques. More complex implementation will achieve more detailed results.

One major disadvantage this shader has is the lack of soft shadows. In real life, shadows are more defined close to the shadow casting object and become blurrier and less and less defined the further away they get. Realistic soft shadows can be achieved using ray tracing and this is a major advantage they hold over traditional approaches. Unfortunately, this feature was cut for time, but would be a high priority for future development.

The second ray traced shadows approach is the “single light per pixel” approach, which does not scale negatively in performance with the number of lights, however, does require data to be accumulated of multiple frames.

The performance for the single light per pixel approach is detailed in the table below. Each test was performed at a resolution of 1920 x 1080 over a 60 second period. These tests are performed with the temporal accumulation pass disabled, so as to measure the performance of the shadows shader alone.

Scene	Arcade	Bistro Exterior	Bistro Interior	Emerald Square	Sun Temple
Average Framerate (fps)	947	486	N/A	135	755

From these results, the bistro exterior and emerald square scenes both perform identically to the previous ray traced shadows approach. This is because both scenes have only one light source, so are not affected by the negative performance scaling from increased light counts. With the single light per pixel approach, the performance of the sun temple scene rockets to 572% of the "all lights per pixel" approach. This is because the sun temple has the highest light count so was the worst affected by the negative performance scaling.

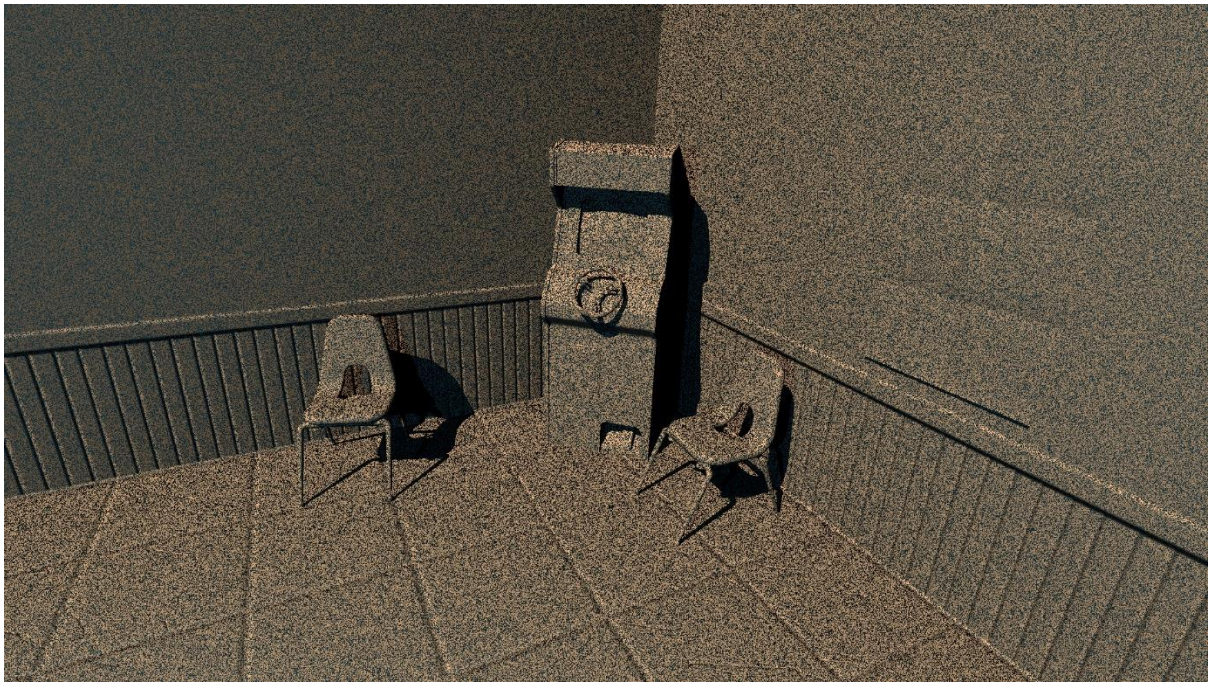

Figure 13: Arcade with noisy ray traced shadows

The noisy output from the single light per pixel approach is shown above. Thankfully this can be remedied with the temporal accumulation approach, which will be detailed in section 4.5.

4.3 Evaluating Ray Traced Global Illumination Pass

The performance for the ray traced global illumination pass is detailed in the table below. Each test was performed at a resolution of 1920 x 1080 over a 60 second period. These tests are performed with the temporal accumulation pass disabled and are done with 1 ray per pixel only.

Scene	Arcade	Bistro Exterior	Bistro Interior	Emerald Square	Sun Temple
Average Framerate (fps)	491	171	N/A	78	331

The bistro interior scene unfortunately fails to load for the same reasons as with the ray traced shadows shader.

Figure 14: Arcade with ray traced global illumination

The global illumination pass alone without temporal accumulation and with only 1 ray per pixel produces a noisy and unstable output image. Performance with the accumulation pass enabled will be considered in section 4.5.

In the arcade scene, the global illumination shader sees a 36% drop in framerate compared to the baseline forward renderer. We see largely insignificant visual improvements for the drop in performance in this scene, however some of the more subtle shading in the area behind the arcade cabinet is obscured by the noisy output and is more visible when using temporal accumulation. In comparison to the pure direct lighting of ray traced shadows, we can see

those obscured areas such as the right side of the arcade cabinet are now lit more realistically, instead of being completely shadowed.

Figure 15: Bistro exterior with ray traced global illumination

Indirect lighting only, with no direct shadowing can look unusual, as is highlighted by the bistro exterior scene. However, this is normal behaviour and the lighting will look much more natural when the ray traced shadow and global illumination shaders are combined in the next section. The front of the café is illuminated directly by the sun, while the left side is obscured. Unlike the ray traced shadows shader, the left side is still visible because it is being illuminated by bounced light being scattered around the environment. The café sign above the veranda picks up a subtle red hue from bounced light reflecting up from below. Subtle realistic effects like this are difficult to achieve with ray traced global illumination. This scene sees a 33% drop in performance from the forward renderer, and this illustrates the heavy performance costs of ray traced global illumination.

Figure 16: Emerald Square with ray traced global illumination

The emerald square scene showcases something of a worst-case scenario for the ray traced global illumination shader. The main central square is mostly flat and sparse with few surfaces for light to reflect off. The subtler indirect light bounces are overpowered by the intense glare of the direct sunlight, so any visual improvements from the global illumination is largely unnoticed, but the performance hit remains. This scene sees a 13% decrease in framerate compared to the baseline. This smaller decrease could be explained by the openness of the scene, there is a higher chance for reflected rays to bounce back up into the sky, which would be registered as a miss, not a hit, so fewer further calculations are required.

Increasing the number of rays per pixel can see a massive performance hit for the global illumination shader and results in a less noisy image overall.

The performance for the ray traced global illumination pass firing 4 rays per pixel is shown below. Each test was performed at a resolution of 1920 x 1080 over a 60 second period. These tests are performed with the temporal accumulation pass disabled.

Scene	Arcade	Bistro Exterior	Bistro Interior	Emerald Square	Sun Temple
Average Framerate (fps)	168	55	N/A	22	115

Figure 17: Arcade with ray traced global illumination 4 rays per pixel

When firing 4 rays per pixel, the output contains significantly less noise, but still too much to be considered a stable image. If using a temporal accumulation technique, this noise reduction is unlikely to impact the final image, as the same work is done after simply sampling 4 different frames. The performance hit for firing more rays per pixel is significant, giving a 66% decrease in framerate averaged across all four scenes. The performance in the emerald square scene averages to 22fps which would not be considered an acceptable interactive framerate in a videogame, dipping well below the accepted 30fps or 60fps standards. Especially considering this is the performance of the global illumination pass alone. The one scenario in which firing additional rays per pixel can be beneficial is when reconstructing the image using a denoiser. With only 1 ray per pixel, there may not be enough information for the denoiser to produce a clean final image, but the increasing to 2 – 4 rays per pixel may be enough to push the image over the edge. This is demonstrated in the following two screenshots, captured using the Optix denoiser.

Figure 18: Arcade with ray traced global illumination 1 ray per pixel denoised

Figure 19: Arcade with ray traced global illumination 4 rays per pixel denoised

Although not captured well in the screenshot, in motion the scene with only 1 ray per pixel contained significant shimmering in motion, and this effect was greatly lessened by increasing the ray count to 4 per pixel.

When paired together, the ray traced shadows and global illumination shaders can provide convincingly accurate lighting in the various scenes.

The performance for the ray traced shadows and global illumination pass together is shown below. Global illumination was calculated firing only one ray per pixel. Each test was performed at a resolution of 1920 x 1080 over a 60 second period. These performance metrics were measured with the temporal accumulation shader disabled; however, all screenshots will be shown with it enabled. This is to better highlight the strengths and weaknesses of the shaders without being obscured by a noisy, unstable image.

Scene	Arcade	Bistro Exterior	Bistro Interior	Emerald Square	Sun Temple
Average Framerate (fps)	490	165	N/A	61	316

The arcade brings an insignificant drop in framerate compared to global illumination alone, within a 1% tolerance of the original result. This is likely explained by the arcade's low light count and simple setting as to a certain extent some shadow calculations are already being done in the global illumination shader, as the illumination of the point hit by the bounced light must be considered.

Figure 20: Bistro exterior with ray traced global illumination and shadows

The bistro exterior scene sees a 4% decrease in framerate. This is partially explained by the same reasons as the arcade scene but not entirely. Much of the performance hit when adding ray tracing effects comes from the creation and managing of the underlying BVH structure, when adding multiple ray tracing effects on top of each other, they reuse this BVH structure. This means that any subsequent added ray traced effects do not add to the render time of a frame the same amount as they would if they were done by themselves. The scene itself produces impressive lighting, with the red bounced light from the veranda being especially noticeable without being obscured by noise. The combination of both direct and indirect lighting results in a more complete image, with the two techniques filling out each other's weaknesses.

Figure 21: Emerald square shopfront with ray traced global illumination and shadows

The emerald square scene shows off the strengths of this lighting technique with this shopfront just off the main square. Notice the bounced yellow light coming from the sign next to the door. The benches and bins on the right side of the building are covered in shadow, yet bounced indirect lighting allows further shadowing behind and beneath them. This effect comes entirely from the indirect component, as the only direct light source in this scene is the sun and these objects are fully occluded from it.

Figure 22: Emerald square shopfront with rasterised render

Looking at the same shopfront using the rasterised forward renderer, the image looks comparatively flat. The details in the shading on the shadowed half of the building are completely missing and there is no indirect lighting component whatsoever. All of these visual improvements come with an actual uptick in performance. The forward renderer achieves an average of 98fps when benchmarking this shopfront and the combined shadows and global illumination shader achieves 100fps. It is important to note however that the ray traced render pass is omitting effects such as ambient occlusion and anti-aliasing, which are included in the forward renderer, so in a full render performance would be lower.

Figure 23: Sun temple with ray traced global illumination and shadows

The sun temple scene showcases impressive shading as light from the brazier bounces around the scene. This scene saw only a 5% decrease in performance compared to global illumination alone. The visual glitch in this screenshot stems from how materials are handled in Falcor, as this sun temple scene was originally designed for Unreal Engine.

The emerald square shopfront and bistro exterior scenes in particular showcase ray traced global illumination as a transformative visual effect. They stand as a best-case scenario for the technique as they are complex scenes with many surfaces for light to bounce around. Detailed bounced lighting achieved by the ray traced global illumination shader is not possible at this same level of quality using rasterised global illumination techniques.

One major area of improvement in the global illumination shader is the need for multiple light bounces. Currently light bounces once upon hitting a surface, then terminates at the next hit point. Multiple bounces would allow more convincingly photorealistic lighting. Additionally, the global illumination shader is currently built on a Lambertian shader model which deals only with matte surfaces and assumes apparent brightness is the same, regardless of the viewing angle. Updating the shader to incorporate a wider range of material properties would be an ideal area for future work.

4.4 Evaluating Ray Traced Ambient Occlusion Pass

The performance for the ray traced ambient occlusion pass is detailed in the table below. Each test was performed at a resolution of 1920 x 1080 over a 60 second period. These tests are performed with an AO radius of 5, 1 ray per pixel and no temporal accumulation. All screenshots will be captured with the temporal accumulation enabled and the scene shading colour disabled to ensure clarity and increase contrast between light and shadow.

Scene	Arcade	Bistro Exterior	Bistro Interior	Emerald Square	Sun Temple
Average Framerate (fps)	705	399	505	163	595

The performance of the ray traced ambient occlusion shader is on average 22.5% faster than the baseline forward renderer when firing only one ray per pixel. In comparisons with the other ray tracing techniques, it can be noticed that ambient occlusion is faster that global illumination on average but slower than reflections. Ray traced ambient occlusion is another technique which results in a noisy output, this can be limited by firing more rays per pixel, temporal accumulation or denoising.

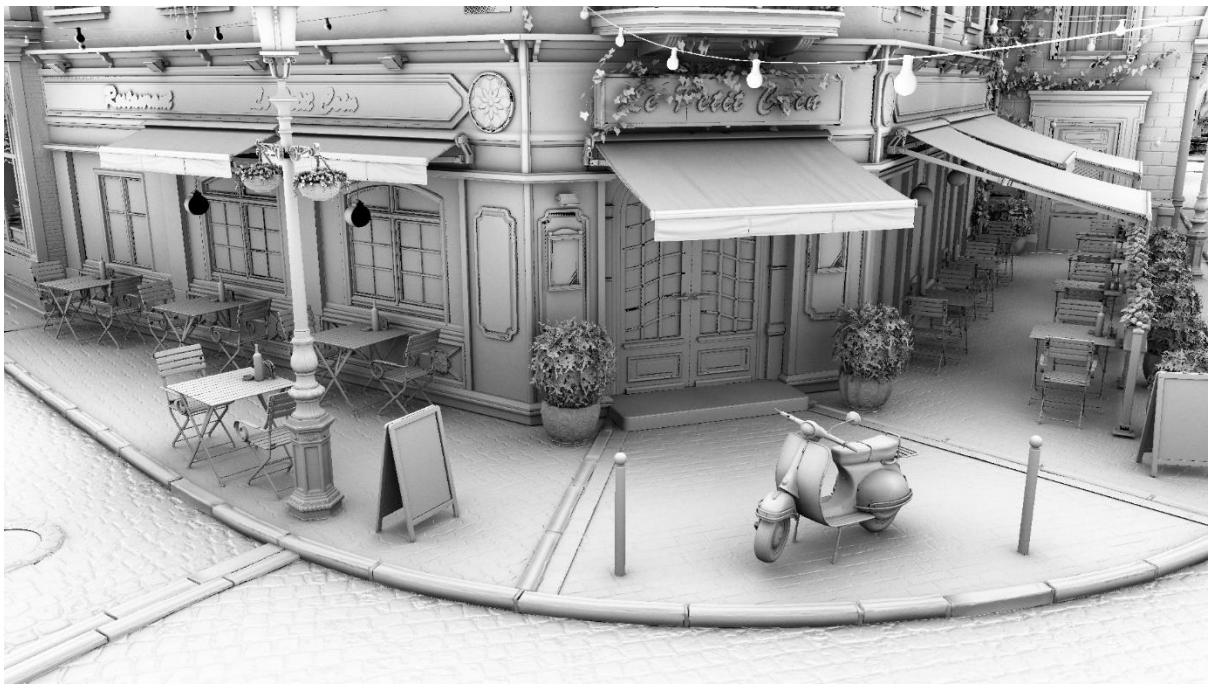

Figure 24: Bistro exterior with ray traced ambient occlusion

Figure 25: Bistro exterior with ray traced ambient occlusion coloured

These two screenshots show the bistro exterior scene with ray traced ambient occlusion, both in greyscale and with colour. Ambient occlusion gives a rough approximation of how exposed an area is to ambient light, it does not depict how light behaves in real life. The effect can be convincing however, with areas underneath tables and points where surfaces meet being shaded a darker colour that their surroundings. The final result adds depth and texture to an image, however it can feel artificial to a certain extent.

Figure 26: Bistro exterior with rasterised render

The forward renderer used a screen space ambient occlusion technique to achieve this effect, so this can be used as a point of comparison for the ray traced ambient occlusion shader. The forward renderer's ambient occlusion implementation is quite lacking when laid alongside the equivalent ray traced technique. The shadowing underneath objects such as the tables and plant pots is extremely subtle, with only some shading where surfaces meet. As with all screen space techniques, this is prone to artifacts as the camera moves. Overall, the ray tracing ambient occlusion technique gives a higher fidelity image at the cost of performance.

Figure 27: Emerald square with ray traced ambient occlusion

The relative open space of the emerald square means many objects are not shadowed as the nearest objects are outside the AO radius. This shader does however provide some pleasant shadowing underneath benches and in the tree foliage. The aggressive shadowing in the grass is a subjective improvement depending on the viewers preferences. The ambient occlusion does provide something of an approximation of grass shadows as it shades the foliage, but many will find it to be overly darkened, considering the scene is in direct sunlight. This is one symptom of ambient occlusion being only an approximation of indirect lighting and will not provide such accurate results as a combination of ray traced shadows and global illumination.

Figure 28: Bistro interior with ray traced ambient occlusion

Figure 29: Sun temple with ray traced ambient occlusion

The sun temple is a scene in which the ray traced ambient occlusion shader excels. The shading is accurate enough to capture the grooves in the tiles, braziers and columns. The areas behind the bannisters and the textures on the statue's wings are all shaded convincingly. Likewise, the bistro interior shows impressive shadowing beneath tables, cutlery and behind the bottles in

the bar. This shows the scalability of the ray tracing approach as it handles such a cluttered scene with many fine details.

The ray traced ambient occlusion shader also allows a scalable number of rays to be fired per pixel. The performance data for 4 rays per pixel are shown below. Like before, each test was performed at a resolution of 1920 x 1080 over a 60 second period, with an AO radius of 5 and no temporal accumulation.

Scene	Arcade	Bistro Exterior	Bistro Interior	Emerald Square	Sun Temple
Average Framerate (fps)	506	250	313	103	389

Increasing from 1 to 4 rays per pixel sees a 33% drop in performance on average across all five scenes. As with global illumination, the only advantage this serves is to create a less noisy image, and while this is off little importance when using temporal accumulation, it can be extremely practical in real world scenarios. Temporal accumulation such as that deployed with these shaders is unlikely to be used in the context of a real video game as it must reset every time the camera moves or the scene updates, i.e., and animation plays. In such a setting it would be more common to use a denoising solution and if the original image is sharper and cleaner, then the output image from the denoiser will be of a higher quality.

4.5 Evaluating Temporal Accumulation Solution

The following table shows the performance for various ray tracing shaders with and without temporal accumulation for various scenes. Each test was performed at a resolution of 1920 x 1080 over a 60 second period. These tests are performed with one 1 ray per pixel where relevant. Average framerate is shown in frames per second. The rows relating to accumulation are highlighted in green.

	Arcade	Bistro Exterior	Bistro Interior	Emerald Square	Sun Temple
Shadows (Single light per pixel) No Accumulation	947	486	N/A	135	755
Shadows (Single light per pixel) With Accumulation	145	129	N/A	122	149
Global Illumination No Accumulation	491	171	N/A	78	331
Global Illumination With Accumulation	147	132	N/A	71	154
Global Illumination + Shadows No Accumulation	490	165	N/A	61	316
Global Illumination + Shadows With Accumulation	142	131	N/A	56	148
Ambient Occlusion No Accumulation	705	399	505	163	595
Ambient Occlusion With Accumulation	153	141	126	136	151

The temporal accumulation shader exhibits unusual behaviour here, as it does not result in an expected percentage decrease in performance. Instead, framerates appear to reach a cap in the 130 – 150 fps range. This can likely be explained by saying that the temporal accumulation shader adds a fixed amount of time to the frame render time which is not impacted by the number of triangles on screen. This would mean that with the accumulation shader enabled, the frame time cannot get low enough to enable framerates in excess of 150fps. Thankfully, these high framerates are not standard in the videogame industry and performance is still well within acceptable ranges. The emerald square scene indicates that once below this maximum fps threshold, the percentage performance cost to the shader is quite low, only costing 5fps with global illumination and shadows enabled.

The following two screenshots demonstrate the effect the temporal accumulation shader has in removing noise from an image.

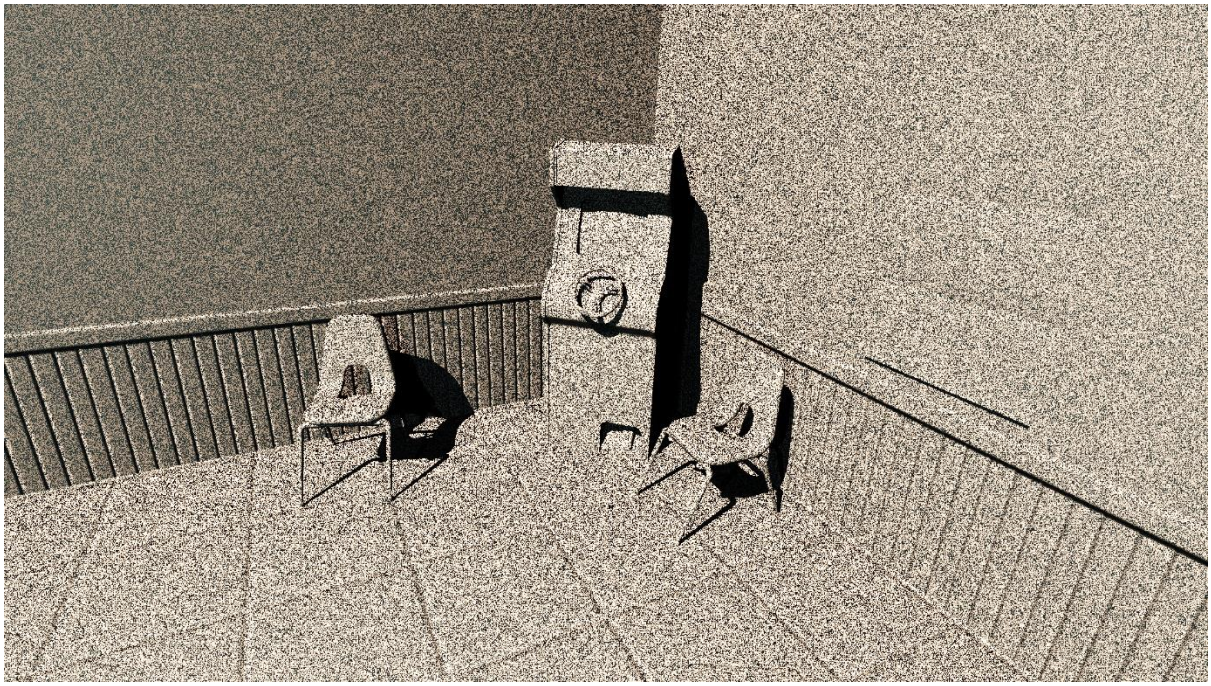

Figure 30: Arcade with ray traced shadows noisy

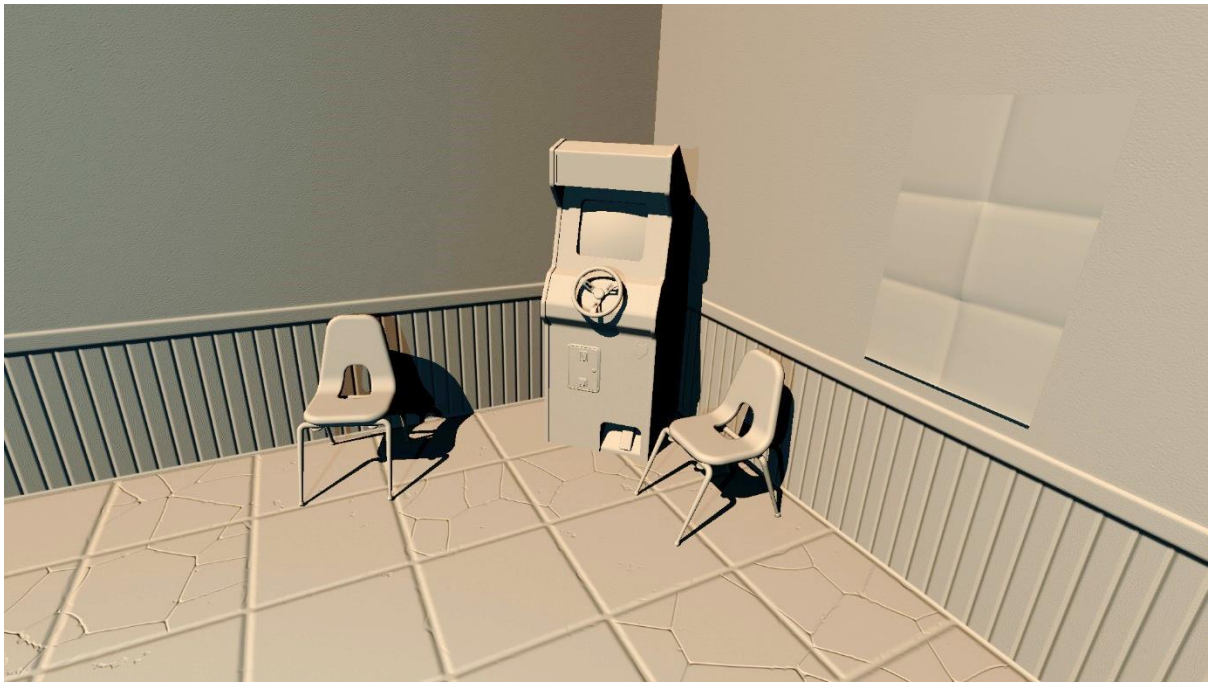

Figure 31: Arcade with ray traced shadows accumulation

The shader results in an incredibly clean final image with all noise completely removed. This is a very successful outcome.

Currently the accumulation shader requires the camera in the scene to be stationary and all objects to be static. Continuing to accumulate data while the camera moved, or while object animations played would result in a smeared image. A denoiser would clean up each frame individually and allow the camera to move freely, however these are less effective at producing perfectly clean output images. There are limits to what a denoiser can achieve as they will produce better results with less noisy images. Denoisers are much more performance intensive, however the benefit of being able to move the camera is a necessity, so in a video game context this would be the preferred approach to cleaning an output image.

For further developments, it would be ideal to investigate the unusual performance characteristics of the accumulation shader and develop a solution which will enable higher framerates.

4.6 Evaluating Software Engineering Approach and Project Plan

Overall, the project was successful, although it was not without its fair share of bumps along the road. The early pivot from Unreal Engine to Falcor caused some major disruption in the short term, however it turned out to be extremely beneficial in the long run. The simplified render pass structure allowed for shaders to be written and executed with relative ease, compared to the process in Unreal Engine. This should not come as a surprise, considering Falcor is created for purposes exactly like this for Nvidia's own internal research.

The project was carried out using an agile software engineering approach and this was a mostly beneficial choice. Soft weekly deadlines in the form of stand-up meetings with the project supervisor were a boon to the project's time management. The necessity to produce a working product at the end of these weekly sprints was a strong motivator to keep a forward momentum on the work being done. Not to say that this always went according to plan, or that there were no problems encountered during development. The majority of implementation took place during the Easter break, during which no stand-up meetings occurred. Occasionally a meeting would be missed during term time. Some more intensive tasks were too significant to produce a minimum viable product to display during a sprint, particularly the first reflections shader during the transition to Falcor and global illumination. These were minor disruptions however in the wider scope of the entire project, and such speed bumps had already been accounted for in the original project timeline. Overall, the agile software engineering approach was a positive experience for the project. The agile approach allowed changes to be made more freely during the development process, such as the Falcor pivot and the addition of an ambient occlusion shader. A more structured approach such as waterfall would have been too rigid for this particular project.

This is the original project timeline included in the proposal.

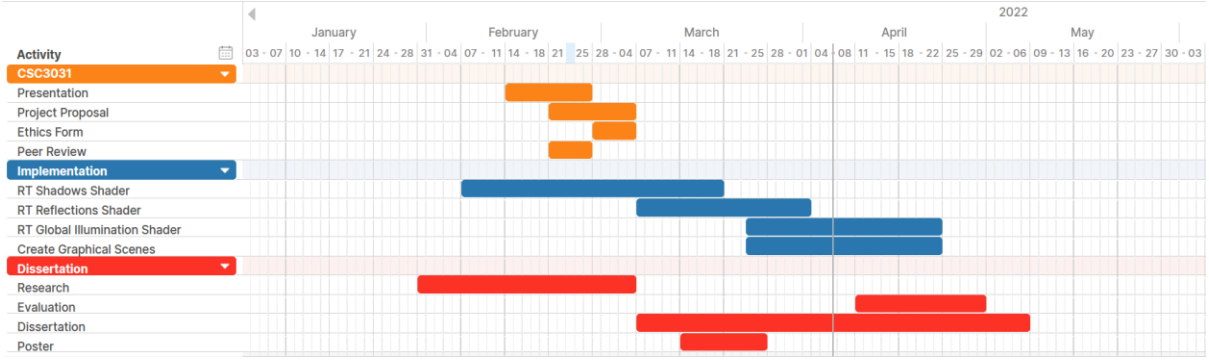

Figure 32: Project plan

Although not perfect, the broad strokes of this plan do accurately represent the reality of how the project progressed. The research phase ran parallel to the development of some of the

early prototypes, although this is shown on the plan as "RT Shadows Shader". The reflections shader was the first to be developed and took the longest amount of time, as it required gaining familiarity with the Falcor tools. This timeline allocates three weeks for each shader type, however in reality the shadows shader was completed in less than one and the global illumination shader in under two. Additionally, the "Create Graphical Scenes" task was cut and the decision was made to instead rely on the ORCA scenes made available for research use by Nvidia. All of this saved time enabled the addition of the ambient occlusion shader to highlight a broader range of possibilities with ray tracing.

Overall, the project plan was stuck to fairly well, besides the minor changes mentioned above. The risk management undertaken in the planning phase allowed development to continue smoothly, without loss of data or downtime from illness. The one week of slack built into the timetable before the deadline proved to be useful as the project write up took longer than expected.

5.0 Conclusion

The ultimate goal of this project was to determine the ideal ray tracing techniques to be implemented in certain situations in order to maximise visual fidelity and minimise performance costs. In reality, this task is incredibly complex as the archetypical scenes selected from ORCA cannot represent every possible scenario. However, based on the findings of the extensive evaluation, the conclusions drawn are as follows. This conclusion will list each of the five scenes and recommend ideal ray traced techniques for each, backed up with evidence from the results of the project.

Arcade: This is a simple indoor scene with few fine details or reflective surfaces. Ray traced reflections would go unnoticed, with performance being wasted. Ray traced shadows are unnecessary to capture the broad details of this scene. By looking at figure 5 showing the ray traced shadows in this scene and figure 7 showing the traditional rasterised shadows, it can be observed that the traditional shadow approach has no trouble keeping pace with ray traced shadows in this scene, while being more performant overall. With such a sparse scene, ray traced ambient occlusion can provide a sufficiently convincing illusion of ambient, indirect lighting. Ambient occlusion's framerate in this scene is 44% higher than global illumination in this scene. As such, ray traced ambient occlusion is the recommended optimal ray tracing technique for sparse, low detail indoor scenes such as this one.

Bistro Exterior: The reflective shop windows of the bistro exterior are a particular showcase for the advantages of ray traced reflections, as seen in figure 1. Additionally, there is only one major light source in this scene: the sun. The narrow streets of this scene provide ample opportunity for indirect light to bounce and illuminate areas hidden from the sun, so this scene proves to be a best-case scenario for ray traced global illumination. Although performance intensive, ray traced global illumination is a transformative effect in terms of the fidelity and believability of a scene. A comparison of its effects in an urban outdoor environment are seen in figures 21 and 22. While the bistro exterior does have some fine detail objects that would benefit from ray traced shadows, priority must be given to the more transformative effects of ray traced reflections and global illumination which are much harder to achieve using rasterised techniques **(Wald et al., 2002)**. Because of this, it is recommended to implement traditional shadowing techniques in outdoor urban scenes such as this one in order to save performance. A combination of ray traced reflections and global illumination is the recommended ideal for scenes similar to the bistro exterior.

Bistro Interior: The indoor cluttered scene of the bistro interior may find it benefits less from the effects of global illumination and that the approximation ambient occlusion gives of indirect lighting is sufficient. This is highlighted in figure 28. While this particular scene unfortunately did not load with ray traced shadows, similar scenes would certainly benefit from

the ability to shadow such fine detail accurately. This ray traced shadow effect is highlighted best in figure 11. Detail at this level is a scene with dynamically moving lights would simply not be possible to achieve efficiently using rasterised techniques. **(Laine et al., 2005)** Specifically in the bistro interior scene, there were a number of reflective surfaces, such as glasses, windows and metal cutlery which could make it a prime candidate for ray traced reflections. The recommendation for cluttered, highly detailed indoor scenes such as the bistro interior is to use ray traced shadows and ambient occlusion. Optionally, ray traced reflections could be added if the scene contained sufficient reflective surfaces to benefit from it.

Emerald Square: The sparse outdoor environment of emerald square proved to be the least affected by implementing many of these ray traced techniques. Figure 16 shows global illumination is largely ineffective as the outdoor expanse has little room for light to bounce around. The indirect lighting element is overpowered by the direct glare of the sun. Without any reflective surfaces at all, ray traced reflections are a nonstarter. It is worth noting that in an outdoor scene with a body of water, ray traced reflections may be welcome, however. Ray traced ambient occlusion overly darkened much of the foliage in the scene and suffered from a lot of the same defects as the global illumination implementation. This was noted in figure 27. The standout effect was ray traced shadows in this scene, as it so effectively captured the fine detail shadows of the foliage in figure 11. The recommended ray traced effect for such sparse outdoor rural settings as this is shadows, with optional reflections depending on the presence of bodies of water.

Sun Temple: Another sparse indoor scene, what is true of the arcade is largely going to be true of the sun temple. Ray traced reflections must be tampered with aggressively to produce effective results in the sun temple, as seen in figure 3. Although reflections in the marble are aesthetically pleasing, the roughness threshold becomes so high that the scene looks less believable. A more robust reflections solution could perhaps better capture reflections on rougher surfaces like marble. Ray traced shadows in this scene do not produce effects which couldn't already be achieved with rasterised shadows. Global illumination is pleasant in the large halls of this scene, however ambient occlusion provides an accurate enough estimation of indirect lighting to be convincing, as seen in figure 29. As with the arcade scene, the recommended effect for sparse indoors scenes such as the sun temple is ray traced ambient occlusion.

5.1 Aims and Objectives

The project was overall successful in fulfilling the aims and objectives, although this will be discussed in more detail as the objectives changed throughout the course of development.

Aim: To implement and compare hardware accelerated real time ray tracing algorithms for various lighting techniques.

This aim was fulfilled to a high standard. Four ray traced effects from the literature were implemented and this covers a wide range of the various lighting techniques which can be achieved with the technique. The evaluation and comparisons drawn between the various ray traced effects and traditional rasterised effects were extensive and produced valuable conclusions. One way this evaluation could have been improved, would be by isolating the various rasterised graphical effects. For example, comparing ray traced shadows to just a rasterised shadow pass's performance would have been more valuable than comparing to an entire rasterised render in the forward render pass.

Objectives

1. **Identify four existing ray tracing algorithms for real time lighting techniques from the literature.**

   This objective changed slightly since the project proposal, however it changed to expand the scope of the project, rather than to restrict it. An additional algorithm from the literature base was implemented and this was a particular strength of the project. This objective was completed successfully.

2. **Implement shaders for these ray traced lighting techniques to run in Nvidia's Falcor Engine.**

   This objective was rewritten to specify Falcor rather than Unreal Engine. This would not qualify as failure to fulfil the objective however, as the essence of implementing the shaders from the first objective remained and was completed. Changing to Falcor from Unreal simply changed *how* the objective was completed. These four shaders could have been improved and made more robust, as is specified in the future work section, but they were implemented to a high standard and are sufficient for the project's current scope and requirements.

3. **Identify and implement a range of 3D graphical scenes to highlight the strengths and weaknesses of each lighting method.**

   This is one objective that the project perhaps failed to meet. The original objective in the proposal specified that three graphical scenes should be created specifically for this project and this was not done. Creating the scenes especially for the project would have meant they could have played more into the strengths and weaknesses of the developed shaders as they were customised specifically for them. Instead, scenes were

selected from ORCA. This was a minor sacrifice to be made as it resulted in obtaining five professional quality scenes which cover a wide range of environments, and also freed up development time to create the ambient occlusion shader. While the original objective was not literally fulfilled, the essence of obtaining a range of graphical scenes was fulfilled. This change in the third objective did not end up ultimately harming the outcome of the project.

4. **Evaluate each method and combination of methods' performance and visual profile across different 3D graphical scenes.**

   This objective was satisfied. The results and evaluation were a massive focus for this project and were carried out in great detail to produce valuable conclusions. The evaluation could have been improved by giving a greater focus to combinations of various ray traced effects.

5.2 What was Learned

This project introduced me to a range of new skills I had no experience with before starting this project. At the earliest stages of research and prototyping I was not confident that creating a ray tracing implementation would be within my abilities, but by engaging with the literature base and familiarising myself with the tools used in this project, I was able to rise the challenge.

I found the agile software engineering approach to be a positive experience, and this would be my preferred approach to future projects. The weekly sprints help with time management, and my only regret is they weren't stuck to more rigorously. Setting weekly deadlines during the easter break in the absence of stand-up meetings would have allowed the project to proceed more smoothly. In future projects I intend to set more personal internal deadlines, as the weekly sprint structure was sorely missed when writing up the final dissertation.

This project allowed me to further my work with shaders. I had some brief experience working with HLSL, but this project allowed me to grow my skills by working extensively in Slang. I expect the skills gained working with Slang will carry over as the language is based on HLSL and the syntax is identical. Working with Falcor was my first experience working with C++ and this was a big part of the reason why getting the very first shader set up was so painful. I am happy to have had the experience, though, as I expect C++ literacy will be a useful skill to have in the future.

5.3 Future Work

Areas for improvement and future work are discussed throughout the research and evaluation section, however they are summarised here below, organised by shader:

Ray Traced Reflections: This shader would benefit from a more robust material system which would allow it to accurately capture reflections on a wider range of rougher surfaces. Support for a hybrid approach of ray traced and screen space reflections could improve performance while minimising any screen space artifacts.

Ray Traced Shadows: One of the biggest advantages of ray traced shadows is soft shadowing. This is when shadows become less defined as they get further from the shadow casting object and this is how light behaves in the real world. This would be a high priority for future development.

Ray Traced Global Illumination: Like with reflections, the global illumination shader would benefit from a more robust material system to enabled more accurate light bounces on a wide range of rougher surfaces. The most valuable area for future development would be enabling multiple light bounces, as this would allow for more accurate indirect lighting.

Ray Traced Ambient Occlusion: As a more straightforward technique, ambient occlusion has less room for future development. One area of concern that should be investigated is how ambient occlusion is affected by foliage and grass. An interesting future development would be to uncover a method to prevent the over-darkening ambient occlusion can exhibit on grass.

Appendix

Glossary

ORCA	Open Research Content Archive – an archive run by Nvidia of graphical scenes free to use in a research context
AO	Ambient occlusion – a graphical effect which provides an approximation of shadowing from ambient lighting
GI	Global Illumination – a graphical effect which simulates how light bounces around an environment
GPU	Graphics Processing Unit
CPU	Central Processing Unit
FPS	Frames per second
HLSL	High level shading language
API	Application Programming Interface

References

1. Purcell, T., Buck, I., Mark, W. and Hanrahan, P., 2005. **Ray tracing on programmable graphics hardware.** *ACM SIGGRAPH 2005 Courses on - SIGGRAPH '05*,.
2. Macedo, D., Serpa, Y. and Rodrigues, M., 2018. **Fast and Realistic Reflections Using Screen Space and GPU Ray Tracing—A Case Study on Rigid and Deformable Body Simulations.** *Computers in Entertainment*, 16(4), pp.1-18.
3. Wald, I., Kollig, T., Benthin, C., Keller, A. and Slusallek, P., 2002. **Interactive global illumination using fast ray tracing.** *EGRW '02: Proceedings of the 13th Eurographics workshop on Rendering*
4. Laine, S., Aila, T., Assarsson, U., Lehtinen, J. and Akenine-Möller, T., 2005. **Soft shadow volumes for ray tracing.** *ACM SIGGRAPH 2005 Papers on - SIGGRAPH '05*,.
5. Sabino, T., Andrade, P., Gonzales Clua, E., Montenegro, A. and Pagliosa, P., 2012. **A Hybrid GPU Rasterized and Ray Traced Rendering Pipeline for Real Time Rendering of Per Pixel Effects.** *Lecture Notes in Computer Science*, pp.292-305.
6. Corso, A., Salvi, M., Kolb, C., Frisvad, J., Lefohn, A. and Luebke, D., 2017. **Interactive stable ray tracing.** *Proceedings of High Performance Graphics*,.
7. Aila, T. and Laine, S., 2009. **Understanding the efficiency of ray traversal on GPUs.** *Proceedings of the 1st ACM conference on High Performance Graphics - HPG '09*,.
8. Egan, K., Durand, F. and Ramamoorthi, R., 2011. **Practical filtering for efficient ray-traced directional occlusion.** *ACM Transactions on Graphics*, 30(6), pp.1-10.
9. **Open Research Content Archive:** https://developer.nvidia.com/orca Used for the bistro interior, bistro exterior, emerald square and sun temple scenes.
10. **A Gentle Introduction To DirectX Raytracing** https://cwyman.org/code/dxrTutors/dxr_tutors.md.html