

Faculty of Science, Technology and Arts

**Department of Computing
Project (Technical Computing)
[55-604708]
2019/20**

Author:	Thomas Clark
Student ID:	B6015511
Year Submitted:	2020
Supervisor:	Tom Sampson
Second Marker:	Joe Saunders
Degree Course:	BSc Computer Science for Games
Title of Project:	An Investigation into the Application of Real-Time Ray Tracing on Rendered Material Surfaces in Games

Confidentiality Required?

NO ☒

YES ☐

An Investigation into the Application of Real-Time Ray Tracing on Rendered Material Surfaces in Games

Thomas Clark

Sheffield Hallam University

BSc Computer Science for Games

ACKNOWLEDGMENTS

I would like to thank my friends and family for their support throughout University and allowing me to have a memorable period of my life.

I would also like to thank all the staff on my Computer Science for Games course, as they have all been great. Additionally a thank you to my project supervisor Tom Sampson for his feedback and ideas with the project.

ABSTRACT

The purpose of this project is to investigate the current state of real-time ray tracing in games, as it is an up and coming technique that game developers will begin to regularly utilise as we move into the next generation.

It will include a deliverable utilising the techniques, showing where the hardware currently sits in the market and the benefits it provides in terms of visual improvements, in comparison to what currently exists as a common technique for rendering inn games.

CONTENTS

Acknowledgments	3
Abstract	4
Contents	5
1. Introduction	7
1.1 Project Overview.....	7
1.2 Project Aims.....	7
1.3 Deliverables	8
2. Information Review	8
2.1 History of Ray Tracing & Rendering	8
2.2 Ray Tracing Techniques	9
2.2.1 Shadows.....	9
2.2.2 Global Illumination	9
2.3 Existing Hardware and Software.....	10
2.4 Problems moving forward	11
3. Design Approach	12
3.1 Introduction	12
3.2 Software	12
3.2.1 Engine	12
3.2.2 Other Software	12
3.3 Hardware	13
3.4 Assets	13
3.4.1 Building a scene.....	13
3.4.2 Models	13
3.4.3 Materials and Textures	14
4. Development & Implementation	15
4.1 Setting up the Environment.....	15
4.1.1 High Definition Rendering Pipeline	15
4.1.2 HDRP + DirectX Ray Tracing	16
4.1.3 Forward & Deferred Rendering.....	18
4.2 Ray Tracing implementations	19
4.2.1 Ray Traced Ambient Occlusion.....	19
4.2.2 Ray Traced Global Illumination	21
4.2.3 Ray Traced Shadows (+ contact shadows)	23
4.2.4 Ray Traced Reflections.....	27

4.2.5	Ray Traced Refractions	29
4.2.6	Recursive Rendering	31
4.3	Scene Management	33
4.3.1	Materials and Textures	33
4.3.2	The effects of Ray Traced Shadows, Reflections and Refractions on the scene	33
4.3.3	Ray Tracing Effects in Real Time	34
4.3.4	Ease of Use Implementations	36
5.	Comparisons	38
5.1	Visual Improvements	38
5.1.1	Rasterization	38
5.1.2	Real-Time Ray Tracing	40
5.1.3	The Differences	42
5.2	Statistics and Optimisations	43
5.2.1	Frame Rates	43
5.2.2	Hardware Usage	44
6.	Critical Evaluation	46
6.1	Overview	46
6.2	Specification	46
6.3	Process Improvements	46
6.4	Hardware and Optimisations	47
6.5	Modifications & General Issues	48
6.6	Personal Reflection	49
6.7	Future Work	50
	References	51
	Bibliography	53
	Appendices	55

1. INTRODUCTION

1.1 PROJECT OVERVIEW

The ability to improve the virtual realism in Games is an ever-growing challenge that Game developers are facing in order to create engaging worlds and environments. Recently there have been developments in both hardware and software to provide some of the most resource intensive technologies of computer graphics to your screen but doing so with as little overhead as possible in terms of performance. The latest and possibly greatest advancement involves Real-Time Ray Tracing.

Ray Tracing is a technique often used in games, however doing this in real time is a completely different matter, due to the huge impact on performance when re-calculating all the relevant values for shading, lighting, colours, reflections, shadows and more. The technique reverses the actual lighting coming from the light itself, as it is about firing a 'ray' from the position of the camera for every pixel in the viewport. This ray is then extended into the scene until it hits an object, which in turns creates a ray towards the light. The material of the object and normal of the pixel determines how this ray is sent back towards the light. The application of this in real-time, as mentioned, is resource intensive therefore being selective on what implements real-time ray tracing on the scene and what doesn't is important. Surface materials are a good option, as most of the time it involves flat or large objects, while getting the most out of the variation of lighting depending on the material, therefore showcasing its ability, and making a more obviously better-looking game.

This project will investigate this, by researching and implementing the applications of real-time ray tracing to several types of material surface within Unity Engine with a view into the feasibility of using this technique compared to some other popular techniques. With this in mind, this report will look into the hardware that is now available to allow us to use and improve the performance when using real-time ray tracing and also look into how and why real-time ray tracing is being used more in our games to see what the future looks like with this technique.

1.2 PROJECT AIMS

The investigation into the implementation of real-time ray tracing as part of this project, should include technical details within this report of the application in Unity and the techniques of ray tracing, as well as a scene created within Unity Engine itself, designed in a way that showcases the different effects of real-time ray tracing on various material surfaces, such as transparent surfaces, shiny surfaces and non-shiny surfaces. As this is a real-time application, it needs to show how the changes of the environment affect the shadows, reflections, lighting, and refractions of the surfaces. This means having moving objects and changing lighting as a part of the scene. Whether all of this is visually and worth implementing into games is simply a matter of perspective, therefore this project will include images comparing all of the ray tracing effects to build a picture of what has the most effect, and real world statistics, which look at comparisons of the application with and without real-time ray tracing.

1.3 DELIVERABLES

The scene for this project will be created using Unity Engine's new Ray Tracing engine, however there are various techniques to the way in which real-time ray tracing can be handled. Unity's Ray Tracing using HDRP contains the following effects: Ray Traced Ambient Occlusion, Ray Traced Contact Shadows, Ray Traced Global Illumination, Ray Traced Reflections, Ray Traced Shadows and Recursive Ray Tracing (Unity, 2019). All of these may be used in the final scene; however, this project will primarily focus on the effects of Reflections, Refractions and Shadows on Surface Materials. Global Illumination and Ambient Occlusion will also be used here to provide further Real-Time ray tracing effects alongside deferred rendering, which tends to be a more efficient method. By the end, the deliverable will consist of several Unity builds of the same scene, but with different rendering aspects modified to provide comparisons of the Real-Time Ray Tracing effects.

2. INFORMATION REVIEW

2.1 HISTORY OF RAY TRACING & RENDERING

The technique of ray tracing has in fact been around for many years, but in a real-time implementation it just simply was not practical due to how long it took to render each frame. In the cinematography field ray tracing has been used regularly, especially in animation such as Cars (Pixar, 2006) and Monsters University (Pixar, 2013), however even pre-rendering using ray tracing is a large task, as Christensen et. al (2006) suggests in a paper on Ray Tracing for the Movie Cars – “Rendering this image (a frame) used 111 million diffuse rays, 37 million specular rays and 26 million shadow rays”, following this shortly after with “if the ray differentials are ignored and no caching is done, the render time is 15 hours 45 minutes”. These sort of render times can be reduced using various algorithms, such as pixel averaging (Behmanesh, Pourbahrami and Gholizadeh, 2012), parallel programming and the techniques Per mentions regarding ignoring ray differentials and caching. However, to render in a real-time environment, it realistically requires at least 30 frames per second - a large reduction from 15 hours.

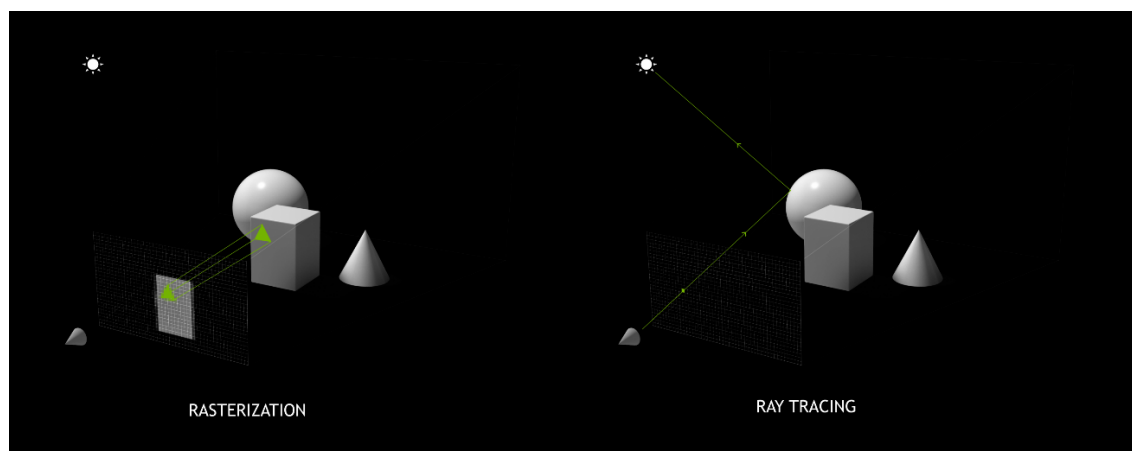


Figure 2.1 Rasterization v Ray Tracing (Stitch, 2018)

Instead, for years games that use real-time computer graphics have utilized rasterization techniques, using diffuse, ambient, and specular lighting (such as Blinn-Phong) and have prospered because of it. The difference with rasterization being that it projects the vertices in the world into a 2D view (which is what you see on the screen) and uses processes such as vertex shading combined with the lights in the scene. Caulfield (2018) states in his article titled “What’s the Difference Between Ray Tracing and Rasterization” for Nvidia – “it’s fast and the results have gotten very good, even if it’s still not always as good as what ray tracing can do.” This suggests that there are some applications that simply do not require the use of ray tracing and gamers are happy with the results without it. In most cases, people will always prefer performance over better graphics, in order to have a more enjoyable gameplay experience overall. Figure 1.1 shows a basic comparison of the two technologies.

2.2 RAY TRACING TECHNIQUES

2.2.1 Shadows

One of the largest problems computer graphics face is the computation of the shadows on the scene, as Haines and Akenine-Möller explain (Ray Tracing Gems, 2019). In rasterization methods, this can sometimes involve having a second “layer/buffer” of an object, which is then transformed appropriately based on the light position, then combined with the original vertex data, to provide the resulting shadow. There are other methods, such as shadow mapping, but when it comes to shadows, real-time ray tracing provides a clearly greater output and “avoids many problems inherent in rasterized shadows” as Haines and Akenine-Möller (Ray Tracing Gems, 2019) continue. As recognised with global illumination, real-time ray tracing continues to have devastating performance problems as the number of rays rise. They explain in the book, that to help with this problem, the computations should focus on the image regions where shadows appear, particularly on the shadow boundaries. They follow up suggesting a practical method for implementing ray traced shadows, involves using multiple methods – the standard rasterization pipeline for primary ray visibility and ray tracing just for the visibility of the light sources.

This implementation then provides a partial improvement in the graphics with the real-time ray tracing with local illumination, while lowering the performance costs by using rasterization in some areas where it can maybe afford to do so more. This will not produce as good of a view as using global illumination but is likely to produce better results than shadow mapping and offline ray tracing, as they suggest. On page 177 of the book, it provides images with a comparison of shadow mapping to the real-time ray tracing and rasterization technique, pointing out the softer edges produced in the shadow mapping technique and slightly lower detail including some potentially missing shadows. The combination of the two rendering techniques, begs the question of what else can be combined with real-time ray tracing, to reduce the costs while keeping as good of a graphical look to our games?

2.2.2 Global Illumination

Despite rasterization being a good option, ray tracing is inevitably more realistic, as it considers multiple objects in the surroundings when the global illumination model technique is used with it. The difference with local and global illumination for ray tracing, is that after the initial ray from the camera hits an object, local illumination then simply fires a single ray into every light source to determine how each light affects that point, therefore isn’t taking into account other objects in the scene. Global illumination means the ray coming out of the

intersection of the object not only goes towards all the light's sources, but also multiple other rays are also shot out in various other directions which hit other objects, accumulating to create a more environment realistic look. The number of rays that are distributed after the ray has hit an object can be defined, however the more rays means the more processing power it will take, having a larger impact on the frame rate. The effects of this can be dependent on the surface material of the object the ray is hitting.

The surface material of the object can determine how the new ray is distributed, as a material such as glass or Perspex causes refraction, alternating the course of the ray through the object (instead of away from it), while a shiny metal surface would send the ray back towards where it came from. Local and Global illumination may also be known as direct and indirect illumination and can be combined (with ray tracing) to create great looking scenes. Figure 1.2 shows the comparison of ray traced global illumination, compared to it without – a similar concept to the goal of this project. The standout difference between the images is the change in lighting within the shadowed areas.



Figure 2.2 (Left) Ray Traced Global Illumination disabled, (right) Ray Traced Global Illumination enabled. Both within Unity. (Unity, 2019)

2.3 EXISTING HARDWARE AND SOFTWARE

Computing all the light and colour values from the rays during run-time takes up so much computing power, but in 2018 Nvidia introduced some architecture that, as John Ballard (2018) writes, “Changed the Future of Gaming”. Nvidia’s RTX Graphical Processing Units using their new Turing technology (DXR) include a new “Ray Tracing Core”, as a part of its architecture alongside the normal rasterizer and various other elements. This new ray tracing core introduces a new ray tracing pipeline, which is optimised specifically for ray tracing and in turn, boosts the performance and ability to use real-time ray tracing. However, it isn’t just the architecture that is involved in the ray tracing pipeline, as the software must also support this, including using the relevant algorithms to make the most out of the hardware. This information is explained within various development blogs released by Nvidia, one of which being a video series titled Practical Real Time Ray Tracing with RTX (Lefrancois and Gautron, 2018).

After months of the RTX cards being released, new products begun to emerge that really started to make use of this new technology and the CEO of Nvidia, Jen-Hsun Huang, acknowledged how important this has been in the market, saying RTX is a “Home Run, I look forward to upgrading hundreds of millions of PC Gamers” (Palumbo, 2019). For these products to be created, there needs to be some software behind them, and various game engines and API’s have introduced support for Real-Time ray tracing since the release of the RTX cards. Some of these are DirectX12 (using DXR, or DirectX Ray Tracing), Vulkan API, Unreal Engine and Unity. Unity ray

tracing is built on top of the High Definition Render Pipeline (HDRP) and requires Nvidia RTX or Turing hardware with an appropriate version of Windows 10 (Martin, 2019).

2.4 PROBLEMS MOVING FORWARD

It is worth noting that despite all the advancements with real-time ray tracing in the industry within the past two years, it is far from being a common implementation in games. One of the most popular videos showcasing real-time ray tracing is the Star Wars: Reflections demo (Unreal Engine, 2018). Gordon (2019) mentions this demo was originally created using four Tesla V100 GPUs, worth approximately £50,000, which is far from affordable. The post then goes on to talk about how just five months after, it was played using a single RTX Quadro GPU which is still around £5,600, but an improvement. This goes to show that provided the hardware (and even software) continues to develop and improve, the next few years can be considerably more fruitful for real-time ray tracing in the market. Nvidia in fact do seem to be advancing the technology, by also introducing the DirectX Ray Tracing capability on some of its older but powerful cards, that do not have ray tracing cores. The post finishes to discuss what ray tracing will be like on the new PlayStation 5, as it was revealed that both new generation consoles (PlayStation 5 and Xbox Series X) will have ray tracing capabilities. How much of ray tracing they use in real-time is still an open question, however it paves the way for the future.

3. DESIGN APPROACH

3.1 INTRODUCTION

The approach of this project will focus on both the performance of Real-Time ray tracing in the modern day, using the new RTX cards created by Nvidia and the Software available, in the form of Unity Engine, and the visual improvements that real-time ray tracing brings in a dynamic environment. Judging the value based on these factors makes sense, since this is the same way that consumers judge whether they believe the technology is worth it. The improvement in visual effects may have serious impacts on the performance, allowing judgements to be made on the benefits at the current time and how it can be improved in the future.

3.2 SOFTWARE

3.2.1 Engine

When deciding how to carry out my investigation into real-time ray tracing, a familiar tool should be used that is well designed for implementing the techniques. There are three main options here: DirectX12, Unity3D (Unity, 2020) and Unreal Engine (Epic Games, 2020). DirectX would allow more flexibility and control into what exactly is implemented in relation to ray tracing, however would require considerably more work that is probably out of scope for this project, while with Unreal Engine I didn't have much experience with, despite its ray tracing abilities being available for a longer time and being well known for high quality graphics. Unity3D is an engine I have used for multiple projects in the past, therefore implementing the scene would not be as much of a problem.

Unity's High Definition Rendering Pipeline is something I was new to and a relatively new introduction to the Unity engine, so I would still need to understand some new methods – such as post processing (including the ray tracing capabilities) moving to use 'volumes' which are added to scenes, instead of part of the baking process. There is a risk to using something which is new to an engine, in that there will be plenty of bugs or maybe not as optimised functionality, however during the development of this project the Unity version with the DirectX Ray Tracing Support came out of the beta phase, therefore it was a good choice.

3.2.2 Other Software

Various other software was used in the development of this project, including Microsoft word from Microsoft office online (Microsoft, 2020) for this report, the open source editing software GIMP (GIMP, 2020) and another open source texture creation software called Materialize (Bounding Box Software, 2020). Part of being able to show a clear difference in the use of different graphical processing techniques is the materials that go with it and the aim of this project is about showing the effects on certain material surfaces, primarily reflective, meaning good metallic surfaces and non-flat surfaces would provide good results. GIMP and Materialize helped me do this for materials that I simply could not find online or via the Unity asset store.

Source control provides the opportunity to keep track of my project and create the various versions required for the comparisons. I often utilise bitbucket (Atlassian, 2020) due to the user-friendly interface for the repository, but also because it is a good size for the project sitting at 2GB. Sourcetree (Atlassian, 2020), also syncs nicely alongside it due to them both being made by Atlassian. Sourcetree is also the source control method I have used the most, so I was familiar

with how it works and what I needed to do to keep track of the project. The use of sourcetree also acts as a backup in case I happened to lose the project or I wanted to roll back a part of the implementation.

3.3 HARDWARE

Hardware is very important for this project since the only way it is possible is due to the recent advancements in Graphical Processing Units which use ray tracing dedicated technology and utilize the ray tracing features within DirectX 12. The Unity documentation for real time ray tracing using the HDRP and DXR, suggests the hardware currently able to handle the Real-time ray tracing features which contain a 'ray tracing core' are the Nvidia RTX Series cards, however as Adrian Willings writes (Willings, 2019), several other graphics cards using the Pascal and Turing architecture are also feasible to utilise ray tracing features primarily due to their high performance and driver modifications. For this project I decided to choose the RTX 2060 – which is one of the lower spec GPU's, however it is also the cheapest and I was not prepared to spend the money on a much higher end card. This of course will lower the capabilities of the project however it should still produce a valid comparison of real-time ray tracing.

3.4 ASSETS

3.4.1 Building a scene

When choosing the assets for the project, I needed to think about what objects and materials will best show off shadows, reflections and refractions in a real-time ray traced environment. This meant using things like metal, marble and glass for reflections, rounded glass objects or Perspex for refractions and be set in an environment that contains sufficient natural and unnatural lighting to produce realistic shadows. I went through a few ideas, including a warehouse and city environment however struggled to find the assets for them or would be far too much work for this project. I decided to go for inside of a modernised garage, where the cars, mirrors and garage door would give me the desired reflections I want, the windows and some other objects would give me some refractions and lastly the large windows combined with indoor lighting would provide realistic shadows.

3.4.2 Models

The next step was acquiring the 3D assets, which involves either creating them using 3D modelling software, building them within Unity using the engine or additional packages and/or finding them online through various sites such as TurboSquid (TurboSquid, 2020). Using another 3D Modelling software could prove challenging and might take longer than it should without any prior knowledge on how it works, so my best options here were to build it within Unity and to find any assets online. TurboSquid provided some of the assets including the cars, the robot character that the camera is attached to and the bicycle. A lot of the other assets were taken from Unity default projects or the unity asset store, while the building itself was created using the Unity ProBuilder extension, which is a popular extension for more intricate modelling from within Unity.

3.4.3 Materials and Textures

Without any materials these assets would not provide any benefits to the scene when trying to show off the real-time ray tracing effects. I knew that acquiring the right materials might be a difficult task and would be a lot of creating my own materials from within Unity. Thankfully, Unity has various asset packages and default projects that contain an abundance of materials to be used, so this helped considerably. For the materials that weren't directly available to me either via Unity or online searches, I needed a relatively easy method to create the correct textures to build up the material, such as diffuse, normal and mask maps, therefore, as mentioned in the software section of this report, Materialize and GIMP served me well. To gather the base textures for this was a relatively quick google image search and there was plenty of options to choose from, to aide me in creating assets that were as realistic as possible within Unity.

4. DEVELOPMENT & IMPLEMENTATION

4.1 SETTING UP THE ENVIRONMENT

4.1.1 High Definition Rendering Pipeline

Background Information & Decision Making

The Unity High Definition Rendering pipeline is a tool made to focus on improved visual quality for physical based rendering and to provide unified and coherent lighting with features independent of the rendering path (Lagarde, 2020). It is a new addition to the Unity engine, with it coming out of the preview stages at the start of 2020 and allows high-end PCs and consoles to access greater overall visual graphics. As of creating this project, the latest stable version for the 2019 version of Unity is 7.3.1, so this will be used. Some of the primary additions the pipeline brings include new shaders, such as the 'lit' and 'layeredlit' shader, plus a decal shader to allow smaller textures to be placed on surfaces of materials. It also includes some major lighting and camera improvements, such as the addition of physical light units to give scenes a realistic look and High Dynamic Range Imaging Skyboxes.

The pipeline also allows the usage of post processing features such as Ambient Occlusion and Screen Space Reflections using volumes, to give realistic shadowing and reflections. Some Games which will be using Unity's HDRP include Oddworld: Soulstorm and System Shock 3, which were both announced at Unity's GDC keynote (Barton, 2019) and provide some truly impactful graphics when compared to any standard rendering pipeline Unity game. These are just some of the many additions and examples with the HDRP, however the most important upgrade relevant to this project is the connection to DirectX12 and DirectX Ray Tracing.

Implementation Processes

To create the project the Unity Hub (Unity, 2020) was downloaded and new project created using the latest 2019 Unity version. This is because the 2020 versions of Unity are not yet out of alpha and beta stages and would likely produce bugs that are unlikely to be resolved by the end of this project. Additionally, when creating the project the High Definition Rendering Pipeline starting template should be selected, as this is required for the Ray Tracing functionality. This ensures all the relevant packages and settings are automatically added to the project, however, initially this project was going to use 7.2.1, but due to bugs it had to be upgraded.

Upgrading the project from HDRP version 7.2.1 to version 7.3.1 is accomplished by accessing the package manager in the Window menu, finding the HDRP and HDRP Config packages and either updating immediately from the current version if one is available, or selecting the advanced dropdown, enabling preview packages and then selecting the dropdown on the HDRP packages and installing one of the other versions. Some of the older 2019 versions of Unity do not have access to version 7.3 of the HDRP, therefore ensuring the Unity version is up to date is important.

At the top there is an 'install configuration editable package' button, which installs the HDRP configuration locally. When upgrading to newer versions of the rendering pipeline, if this is not updated then your HDRP version may not update properly.

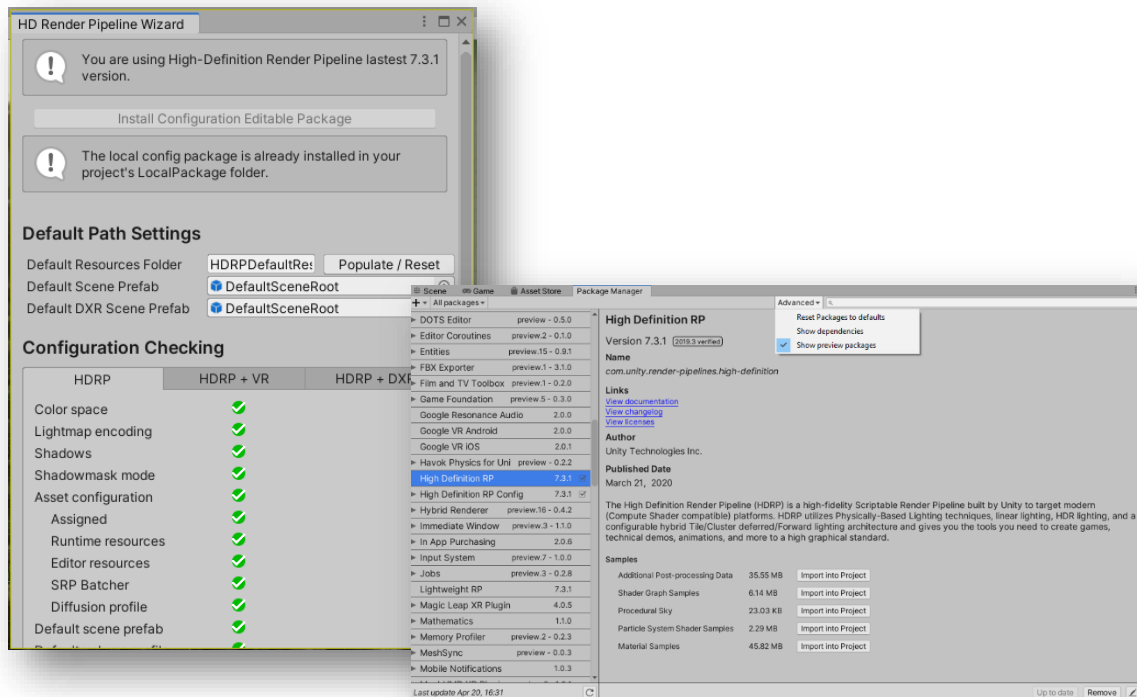


Figure 4.1 HDRP Wizard and Package Installation

4.1.2 HDRP + DirectX Ray Tracing

Background Information & Decision Making

DirectX Ray Tracing is almost standard for current ray tracing architecture solutions, due to the platform agnosticism, therefore it makes sense for Unity to connect their High Definition Rendering Pipeline to that (Peddie, 2019). With the vast visual improvements that the HDRP brings, it only gets better with the introduction of ray tracing, especially in real time. However, currently this combination doesn't fully rely on solely ray tracing, but is a hybrid of rasterization and ray tracing in order to provide an efficient solution, while the hardware isn't quite at the affordable capabilities of regular real-time ray tracing usage (Benyoub, 2020).

Implementation Processes

In order to use the ray tracing features, the HDRP with DirectX is required, meaning the project must be upgraded via a HD Render Pipeline Wizard built into Unity, under Window / Render Pipeline. The Wizard allows easy implementation of the rendering pipeline that contains ray tracing, using a list of various settings, assets and APIs that are required, such as DirectX12 itself and disabling static batching. The wizard allows you to simply click 'Fix All' (or 'fix' for individual configuration changes), and then the project is almost ready to go. In the early stages of the project it was not necessary to upgrade any project materials either to high definition materials or to the latest version of the HDRP, however as will be explained in the materials and textures section, this became necessary. From this point, Real-time ray tracing can be achieved within Unity using the High Definition Rendering Pipeline with DirectX Ray Tracing.

Effects are introduced using the volume system in HDRP projects; therefore a volume was created for all the ray tracing functionalities. The volume can be set to global, meaning affecting everything in the scene, or can be limited to a defined area, potentially allowing

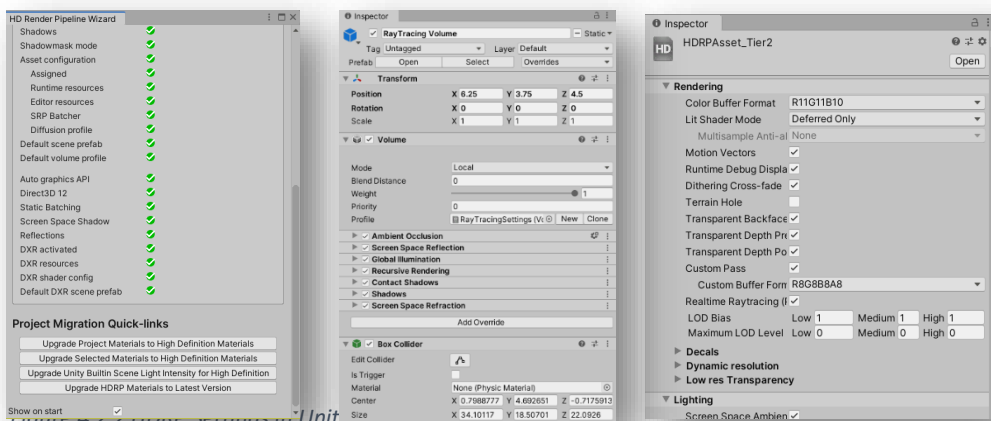
certain aspects of a game to inhibit greater graphics than others. More importantly, this could be attached to a player to limit the range of the ray tracing effects, relevant to the camera position. In this case, the volume is used around the whole of the garage scene, since there isn't anything on the outside that needs rendering in a lower quality, however ensuring the effects are limited within the necessary parts of the world space.



Figure 4.2.1 Volume component in unity

6 of the ray tracing additions DXR brings are: Ray-Traced Ambient Occlusion, Ray Traced Shadows, Ray Traced Contact Shadows, Ray Traced Global Illumination, Ray Traced Reflections and Recursive Ray Tracing. The following sections will talk about these settings and what they do, but primarily which this project utilises, how they were implemented and the effects of it:

- To enable real-time ray tracing, ensure it is turned on for the camera in the Project Settings / HDRP Default Settings / Ray Tracing.
- Of course ray tracing can be pre-computed as well, so ensure this is working in real-time as well, which can be enabled via the HDRP Asset / Rendering / Realtime Ray Tracing.
- Unity contains two tiers of ray tracing, with Tier 2 allowing considerably more ray tracing options for higher quality graphics, in the event you have the hardware to be able to run it. For this scene, Tier 2 allows access to all the ray tracing functionality available. This can be changed in the HDRP asset / Rendering / Ray Tracing Tier.



4.1.3 Forward & Deferred Rendering

Background Information & Decision Making

Forward rendering is the more computationally expensive of the two types, with the method also often restricting the number of dynamic lights in the scene. This is a linear method where each geometry is passed down the pipeline one at a time, then broken down into vertices which are transformed and split into pixels which have final rendering processing applied, until the final image is produced, meaning this is mostly done all in one pass (Owens, 2013).

Deferred shading is based on the idea to defer or postpone most of the heavy rendering (such as lighting) to a later stage (De Vries, 2014). The technique has two passes, a first pass that gets most of the geometry data and stores it in the G-Buffer, followed by a second pass which gets most of the lighting data based on the original data stored in the G-Buffer. This is faster because it only calculates a pixel once in the lighting pass and allows considerably more lights to be active.

Unity also contains an option for using both deferred and forward rendering on a per camera and reflection probe basis; they give an example of using forward mode for a planar reflection probe but deferred for the camera. This allows greater quality reflections while not limiting the lights in the scene and improving camera performance.

Because of the computational costs and use of lighting in the scene to further the example of ray tracing effects, it didn't make sense to use the forward rendering method, therefore for the real-time ray tracing implementation the deferred rendering method was used. This also makes sense with a graphically intensive technique such as real time ray tracing, where saving resources is important.

Implementation Processes

Unity makes modifying the rendering mode easy, by putting a lot of general rendering settings inside of the HDRP Asset. The setting is under Rendering / Lit Shader Mode inside of the HDRP Asset and for deferred rendering should be set to Deferred Only.

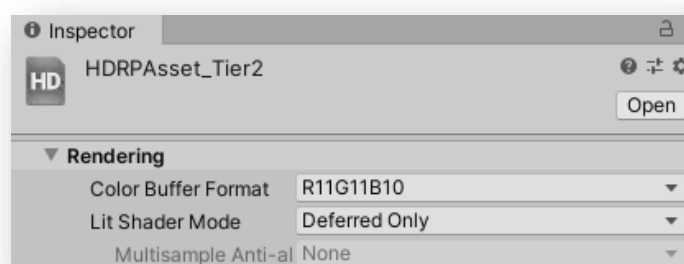


Figure 4.3 HDRP Shader rendering mode in the HDRP Asset

4.2 RAY TRACING IMPLEMENTATIONS

4.2.1 Ray Traced Ambient Occlusion

Background Information & Decision Making

Adding the ray traced ambient occlusion is one of the primary additions to getting more realistic lighting and shadows. As one of the three main areas of this project is shadows, it made sense to add this as it heightens the light contrasts in corners and shadowed areas slightly more, due to the concept of ambient occlusion being calculating just how bright certain parts of the screen should be based on geometry (Stewart, 2020).

Implementation Processes

Similarly to most of the ray tracing features, they are enabled through the rasterization volume overrides. The following steps were taken to implement this feature:

- In the HDRP Asset, enable screen space ambient occlusion under Lighting / Screen Space Ambient Occlusion and ensure it is enabled in the Project Settings / HDRP Default Settings / Lighting / Screen Space Ambient Occlusion.
- In the Ray Tracing Volume, the Ambient Occlusion volume override was added, via Lighting / Ambient Occlusion.
- **Ray Tracing** was enabled on the volume to switch it from the rasterization method.
- The **intensity** should be realistic, as setting this value too high can have a negative effect on the lighting in the scene, making appear too dark. 1 seemed like a sensible value here.
- It makes sense for the **Ray length** to encapsulate the whole scene in this scenario, however in larger scenes the value here can be reduced to a sensible value based on view distances and rendering of objects further away from the camera. A value of 40 was large enough for this implementation.
- By default the ambient occlusion has a lot of noise, and this can be reduced with a combination of the **sample count** (the number of rays per pixel used to create the effect) and the **denoise** option. Using the denoise option allows a lower sample count, saving resources. A sample count of 30 in conjunction with a denoise radius of 1, provided good quality ambient occlusion for this scene, removing a lot of noise.



4.4.1 Ray Tracing Ambient Occlusion Volume Override

Effect on the Scene



Figure 4.4.2 Ray Traced Ambient occlusion enabled



Figure 4.4.3 Ray Traced Ambient Occlusion Disabled

As the images show, ray traced ambient occlusion provides greater lighting contrasts on the scene, while making colours more realistic and not as dull, such as the red on the car and the vibrancy of the overhead lights.

4.2.2 Ray Traced Global Illumination

Background Information & Decision Making

Ray Traced Global Illumination adds considerably more realism to the lighting and shadows similarly to that of the ambient occlusion, however global illumination uses up more resources. It involves bouncing light rays in random directions from the point of contact (pixel) on an object, often referred to as indirect lighting. Colour bleeding is a good example of this, where the light hitting one object causes another object to inherit some of that colour.

Implementation Processes

This implementation can have large effects on computational resources, so should be treated with care. It was implemented as follows:

- The Global illumination volume override was added to the ray tracing volume under lighting.
- **Ray Tracing** was enabled to switch from rasterized global illumination.
- Similarly to the other overrides, the **Ray Length** was changed to 40 to encapsulate the whole scene.
- The **sample count** here can have an exponential effect on performance, so in order to utilise global illumination while keeping the performance reasonable, this was set to a value of 1. Anything higher showed a clear reduction in frame rate, as more rays were being fired per pixel.
- **Bounce** count relates to the number of times a light ray bounces from an object, which with global illumination also has serious performance impacts. Simply setting this to 1 in this scenario (similarly to the sample count) allowed me to utilise global illumination but not have too large of a performance impact for the hardware available.
- Because of the lower sample count, but also just with global illumination in general because of the number of pixels being calculated, the denoise option was essential – however unity provides various denoise options for global illumination to help with performance and quality. The half resolution denoise evaluates the denoise process at half the resolution, therefore reducing quality but improving performance. Combined with the second denoiser pass creates much cleaner lighting. Both were enabled as it provides a mix of quality and performance for the global illumination. The default values for the denoise radius were fine for this implementation, however if further noise is experienced, these can be increased.

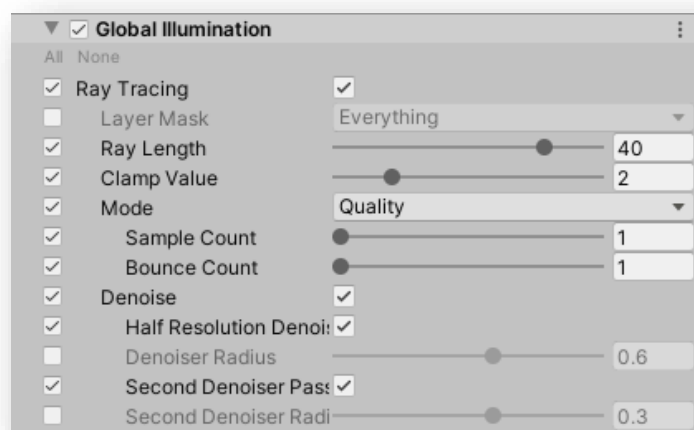


Figure 4.5.1 Ray Tracing Global Illumination Volume Override

Effect on the Scene



Figure 4.5.2 Ray Traced Global Illumination Enabled



Figure 4.5.3 Ray Traced Global Illumination Disabled

The images show the slight improvements with this scene that global illumination brings, primarily the effects of the directional lighting, with the edges of the windows and back wall being slightly brighter and containing more of the directional lights from the outside than the indoor point lights.

4.2.3 Ray Traced Shadows (+ contact shadows)

Background Information

Shadows are a major factor in realistic lighting and ray traced shadows with contact shadows highlight the improvements in the scene. By default unity has various options for shadows, such as shadow mapping, through baked lighting and with screen space shadows, however for the ray tracing implementation it extends the screen space shadows post processing volume to use rays.

Rays work well with shadows, as they can identify generally if a pixel needs to be darker based on the point of impact and angle to the light(s), but additionally if a ray never actually reaches a pixel due to objects intercepting it, it can identify that there is likely going to be a shadow there. This is closer to how light rays work in real life, so it only makes sense that it will make the scene more realistic, compared to some rasterization methods of creating a second shadow layer and skewing the image. Contact shadows add further shadow depth, by adding more detailed harder shadows from things like edges to other objects behind.

Implementation Processes

Ray Traced shadows and Ray Traced Contact Shadows were implemented in Unity in the following way:

- First, ensure the lighting values are relatively correct and do not make the directional lights too bright for the point lights on the ceiling lights inside the scene. A realistic directional light value here is about 3.14 Lux.
- Shadows are generated from the lights themselves, so it only makes sense that the relevant settings are on the lights, but also means not all lights have to contain the ray tracing effects, allowing optimisations. However, ray traced shadows only work with directional lights, spot lights, points lights and rectangle lights.
 - Inside of Shadows / Shadow Map in these lights, they contain a **Screen Space Shadows** option and a **Ray Traced Shadows** option which were enabled to active the ray traced shadows.
 - As this is about showing off ray tracing in real-time, the **update mode** should be set to Every frame, so that it can recalculate the shadows in the effect something is moving.
 - The **Sample Count** improved the noise on the edge of the shadows, therefore it made sense to set this to 4, with the **Denoise** combined with the **Denoise Radius** parameters removing the hard edges from the shadow, giving it a more realistic look. Enabling and setting a radius of 10 gave it enough of an 'edge blur' without overdoing it and reduced a lot of the noise.

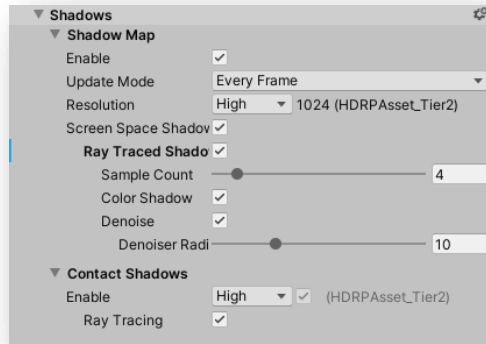


Figure 4.6.1 Ray Traced Shadows settings

- The lights also have **contact shadows** settings, however this can be used in conjunction with regular rasterization methods, so enable the **ray tracing** option, but also set the **quality** to high so that the shadows are good quality (for higher performance this would be set lower).
- Additionally, the contact shadows have a volume override inside my ray tracing volume, inside of the shadowing section.
 - After **enabling** contact shadows, the **length** and **distance scale factor** control how long the contact shadow ray is (in meters) and the scale of the contact shadow in relation to the camera distance away from it. A value of 1 for the length gave the full effect of the contact shadows, while 0.5 for the scale factor optimised the shadows when at a distance.
 - **Max distance** controls how far away the camera would be before the contact shadows no longer render, with the **fade distance** the distance where the shadow starts to fade away from view. As these are small details, it didn't make sense to be able to see these from the other side of the garage, so a value of 20 for max distance is about right for performance and quality with the fade distance at a value of 5.
 - The contact shadows should be as visible as possible and as they are only small this adds a lot of effect and aligns with generally how they would realistically look. Setting the **opacity** to 1 achieved this.
 - The **quality** setting is fine to be set to high, as these are only small details and provide a lot of visual benefits with slightly lower resource usage.

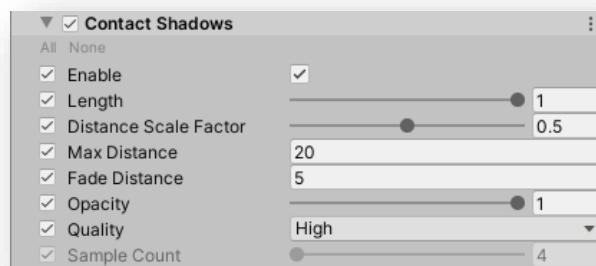


Figure 4.6.2 Ray Traced Contact Shadows volume override

Effect on the Scene



Figure 4.6.3 Ray Traced Shadows Enabled (on all lights)



Figure 4.6.4 Ray Traced Shadows Disabled (on all lights)

As the images show, with ray traced shadows enabled, there is much more definition to the shadows, such as the plants shadow. Additionally, ray traced shadows can reduce the chance of bugs with the shadows, as indicated with the bicycle. There is a strange shadow coming from the handlebars that was generated by the screen space shadows shadowmask.

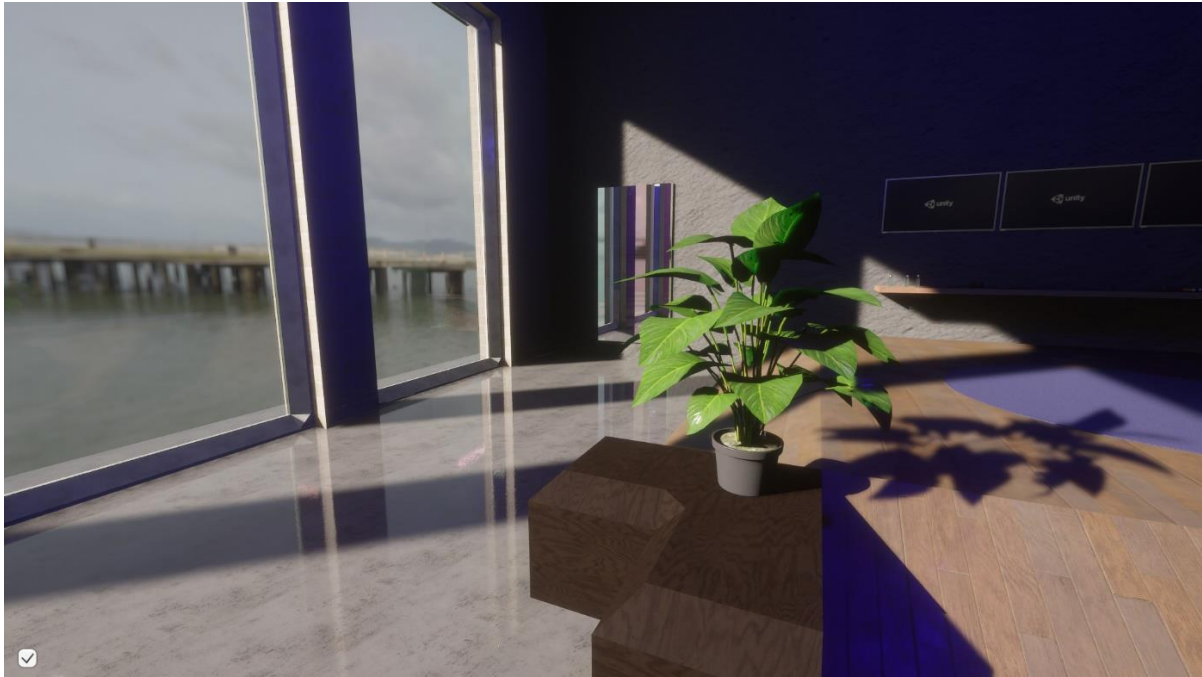


Figure 4.6.5 Ray Traced Contact Shadows Enabled



Figure 4.6.6 Ray Traced Contact Shadows Disabled

The images show the harder edges within the normal shadows, that are created by contact shadows. The plant pot and edge of the table have harder shadows as the lights has hit them more directly. Also, the edges of the windows where light has hit, contains more of an accurate shadow from the window frame, thanks to the ray traced contact shadows.

4.2.4 Ray Traced Reflections

Background Information & Decision Making

Ray Traced Reflections remove the need for reflection probes, as rays bounce off reflective materials in order to provide the relevant reflection. The feature is enabled via the rasterized version of reflections without probes, called Screen Space Reflections (SSR), therefore enabling ray tracing on this volume override adds additional parameters, for example the number of bounces can be increased before it reaches the light source in order to potentially show reflections inside of reflections. The quality of the reflections using this method are good, but not quite as good as recursive rendering. This makes it suitable for objects that are not as reflective, such as the matte colour of the red car and the red garage door.

Implementation Processes

In order to enable this feature, the following steps were taken:

- Ensured that screen space reflections are enabled inside of the HDRP Asset under Lighting / Reflections and inside Project Settings / HDRP Default Settings / Lighting / Screen Space Reflections.
- Inside of the Ray Tracing Post Processing volume, add the volume override / Lighting / Screen Space Reflection.
- Enabled **Ray Tracing** on this override, and changed some of the other parameters:
 - **Reflect sky** makes the skybox show in the reflections, this was enabled.
 - The **ray length** was changed to 40, this means the reflections encapsulate the whole scene and the ray does not cut off halfway through the building. In some scenarios this would be useful and would save resources in the event you cannot see a certain distance away due to fog or blur.
 - The **clamp value** limits the pre-exposed pixels being reflected. This can affect the noise and quality of the reflections. A value of 1.8 provided quality reflections where necessary, however as these are objects that are not very reflective, it did not make sense to have this value too high.
 - The Sample count is the number of reflective rays that are sent per pixel in the frame. This significantly reduces the noise but is quite resource intensive as you scale upwards. A value of 5 here combined with the denoise parameter gave a good level of quality.
 - The denoise option goes a long way to ensuring the reflection is not fragmented and using in conjunction with the sample count allows less rays to be sent out without losing too much quality.

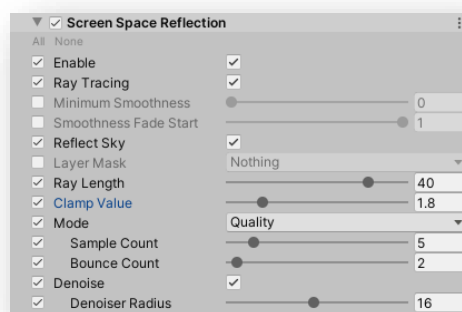


Figure 4.7.1 Ray Traced Reflection volume override

Effect on the Scene



Figure 4.7.2 Ray Traced Reflections Enabled



Figure 4.7.3 Ray Traced Reflections Disabled

The main comparison with the feature in this image is effect on the garage door. You can see in the first image where it is enabled, that it reflects the inside of the garage, while with it disabled you only see the skybox. Using reflection probes here could change this, as will be discussed later in the report.

4.2.5 Ray Traced Refractions

Background Information & Decision Making

Refractions are the rays that essentially travel through objects such as glass and Perspex and can refract at different angles in order to show objects through them differently to what they would look like normally. This makes the light passing through things such as the glass bottles and windows look slightly different, more so with the bottles as they are rounded objects. However, ray traced refractions are only possible in Unity using the recursive rendering functionality, which is discussed in the following chapter. Without the recursive rendering override, refraction is accomplished using screen space refraction.

Implementation Processes

Whether using ray tracing or screen space refraction, individual materials have their own refraction properties that need to be specified and are made available once the material is set to a Surface Type of Transparent:

- The refraction model determines a shape that the object will refract, such as a glass ball with have the sphere refraction model causing a curved refraction. In our case, objects such as the bottles and car windows will have a sphere refraction model, however the glass windows will have a thing refraction windows.
- The index of refraction relates to the amount the object refracts, so the very rounded bottles will have a slightly higher index of refraction such as 1.6, while the car windows would be much lower such as 1.05. Generally, this affects the magnification of the objects on the other side of the transparent object.

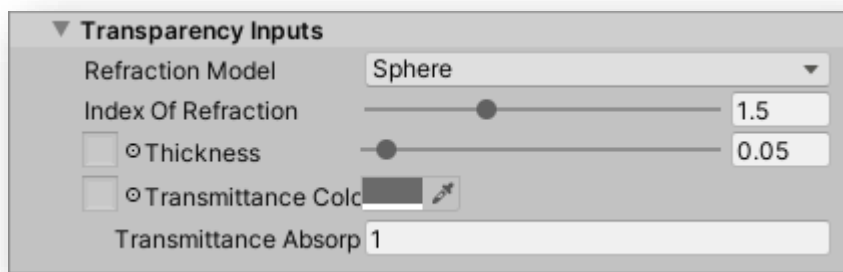


Figure 4.8.1 Materials Refraction settings

Effect on the Scene



Figure 4.8.2 Screen Space Refractions



Figure 4.8.3 Ray Traced Refractions using Recursive Rendering

The Images show the refraction effect on water bottles, where you can clearly see the light refracting through in the second image, and providing a distorted view of what is behind, as well as some specular lighting being returned to the camera. The first image provides a refracted view, but you cannot see the water inside at all, because of the way it handles the refraction, making it almost look like a metal reflection.

4.2.6 Recursive Rendering

Background Information & Decision Making

Recursive rendering accentuates the effects of the reflections and refractions by making objects cast reflection and refractions rays which carry on hitting other surfaces. The difference between this and ray traced reflections and refractions and their bounce parameter, is that it is a replacement pipeline for rendering meshes (Unity, 2020), hence why setting an object to use the ray tracing rendering mode causes the object to not render at all when recursive ray tracing is enabled.

This technique seemed to have the best effect when it came to reflection and refraction, therefore, to get the best comparison, it meant using it on noticeable objects, such as those with metal and glass materials. The problem with that, however, is that it provides a significant performance cost.

Implementation Processes

To enable this feature the following process took place:

- Add the Recursive Rendering volume override to my ray tracing volume and **enable** the functionality in the parameters.
- The **Layer Mask** allows certain types of objects to only use the recursive rendering effects, opening optimisation opportunities. For example, if some objects are on a layer named liquid, but the liquid layer was not on the layer mask here, it would not be considered by the recursive ray. For this, simply making everything be considered improves the quality.
- The **max depth** parameter adds the number of recursions of ray's on objects, allowing things such as more reflections inside of reflections. As this project is trying to show off as much potential with reflections as possible with the resources available, setting this value to 5 allows the scene to capture all the reflections that have been implemented. A value of 1 means the ray bounces away from the mesh to the light, leaving just a reflection of the skybox, while 5 provides the reflection from the car on the reflection from the other mirror, through the reflection of a mirror. Scaling this value up increases resource usage, so setting it reasonably based on the scene is important for efficiency.
- Similarly to most other ray tracing volume overrides, the **ray length** determines the distance the ray goes, meaning if the reflection or refraction is too far away, it will not bounce off the surface.

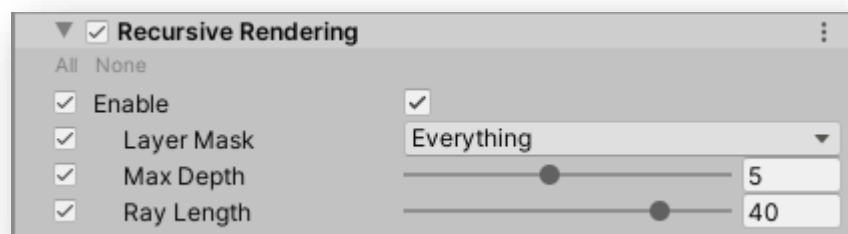


Figure 4.9.1 Recursive Rendering volume override

Effect on the Scene



Figure 4.9.2 Recursive Rendering Enabled



Figure 4.9.3 Recursive Rendering Disabled and materials modified to use Ray Traced Reflections

The images provide a clear improvement with the recursive rendering affect, with the windows have a clearer reflection back into the room, and the metal on the car providing a much higher quality and realistic view. The mirror in the back has a better view of the scenes lighting too.

4.3 SCENE MANAGEMENT

4.3.1 Materials and Textures

One of the largest factors when creating the scene and trying to not only show off the real-time ray tracing effects for shadows, reflections and refractions the best, but also ensure the scene actually looks good, is the materials and textures that I use. The HDRP Lit shader can create very good-looking materials without too much work, but also is the primary unity shader for the HDRP and making the most of the HDRP with DirectX Ray Tracing effects. I acquired the necessary materials from a range of places:

- The Default unity projects contained some useful materials, so the 'SmallRoomRayTracing' file from the Unity GitHub site was downloaded, as well as the default HDRP sample scene.
- Seamless textures were acquired (such as the brick wall texture and marble floor) online, then Materialize utilised, to make various texture maps, such as the normal, metallic and smoothness maps, to put into a mask map using GIMP. This adds depth and accuracy to the material.
- The car models and robot model were all acquired from TurboSquid and mostly came with materials. This allowed me to easily add useful materials to the models, however some didn't work or were not completely transferrable to the HDRP lit shader, therefore I had to source them using some of these other methods, such as creating my own material or using the Unity material packages.
- Quite a few of the materials were sourced from Unity's PBR materials pack from their GitHub, however a lot of these wouldn't work with the HDRP and DirectX Ray Tracing, therefore essentially copying the shader variables and textures to a Lit Shader version is necessary.

4.3.2 The effects of Ray Traced Shadows, Reflections and Refractions on the scene

The combination of all three techniques provides a new and higher quality effect to the scene, but balancing which ones provide the greater improvement is important. In the scene, which uses things such as cars and a marble flooring and plenty of transparent objects, reflections and refraction provide a lot of benefit, while shadows aren't quite as useful, since it is indoor so there isn't loads of objects affecting various shadows and shadow depth. This means slightly more resources were dedicated towards the use of ray traced reflections and recursive rendering, than that of ray traced shadows and contact shadows.

Of course the materials being used are also a major factor in best showing the effects of real-time ray tracing, so for implementing this scene it made sense to do something that contains reflective objects such as metals, plenty of refractive objects and objects that light can travel through for shadows using glass, and some not so reflective objects so the shadows of objects can rest on, such as the brick, concrete and wood. As mentioned in the approach section of this report, there was many ways of obtaining the various materials, using tools such as materialize and also thanks to the ease of use with unity's lit shaders, making producing as realistic a scene as possible with little design and artistic knowledge, considerably easier.

Achieving a lot of the effects and ensuring the settings of all of the ray tracing volume overrides, HDRP settings and material settings required a lot of trial and error, slightly altering values until it felt like it produced the desired affect without using too much computational resources, however help was obtained from the Unity documentation (Unity, 2020), in order to get as close as possible to what is expected from a Unity Real-Time ray tracing scene.



Figure 4.10 Scene view

4.3.3 Ray Tracing Effects in Real Time

Of course this project is focussed towards using ray tracing in real-time, therefore various alterations were made to the scene to include moving objects, which requires the various ray tracing effects to be re-calculated. Thankfully, small changes in object movement don't necessarily require the whole camera space to be re-generated, allowing an improvement in performance, however if the camera itself moves, then all of the ray's need to be calculated and effects altered, as looking at shadows, reflections and lighting from different perspectives, causes different views.

The following was added to the scene to show off the real-time ray tracing in effect:

- The lights on the ceiling swing side to side – the way the lighting hits the lights when they move requires recalculation of lighting rays such as global illumination and ambient occlusion. Reflective surfaces also must re-calculate the pixels being shown.
- The car vibrates as if it is running – Small but fast motion, such as vibrations, can be hard on the hardware, since it is constantly having to re-calculate the rays being affecting by the moving object. This shows a degree of speed with modern hardware and real-time ray tracing.
- The robot model the camera is attached to – Adding this robot allowed a reflection, on objects that utilised ray traced reflections and recursive rendering, further showing the effects of these in real-time. It also means shadows must be recalculated when the robot moves.
- The ceiling lights change colours – This was added for effect to show that the lighting being traced towards other objects changes.

- The Audi car headlights – Added to accentuate the shadows and contact shadows on an object, in this case the car in front. They flicker on and off to show the real-time effect of shadows.
- Large windows, the cars, and the mirrors – These are quite a major part of the scene, as they both use recursive rendering to provide quality reflections and refractions in real-time. All the objects that are moving or changing colour in the scene can be viewed in these, meaning rays are re-calculated to produce accurate reflections and refractions.

```

public class ColourCycler : MonoBehaviour
{
    Light lt;

    public float duration = 1.0f;
    public Color col1;
    public Color col2;

    // Start is called before the first frame update
    void Start()
    {
        lt = GetComponent<Light>();
    }

    // Update is called once per frame
    void Update()
    {
        float t = Mathf.PingPong(Time.time, duration) / duration;
        lt.color = Color.Lerp(col1, col2, t);
    }
}

```

```

public class LightSwing : MonoBehaviour
{
    public float delta = 1.0f;
    public float speed = 2.0f;
    public float direction = -1;
    private Quaternion startPos;

    // Start is called before the first frame update
    void Start()
    {
        startPos = transform.rotation;
    }

    // Update is called once per frame
    void Update()
    {
        Quaternion a = startPos;
        a.x += direction * (delta * Mathf.Sin((Time.time * speed)));
        transform.rotation = a;
    }
}

```

```

public class Vibration : MonoBehaviour
{
    public float delta = 1.0f;
    public float speed = 2.0f;
    public float direction = -1;
    private Vector3 startPos;

    // Start is called before the first frame update
    void Start()
    {
        startPos = transform.position;
    }

    // Update is called once per frame
    void Update()
    {
        Vector3 a = startPos;
        a.y += direction * (delta * Mathf.Sin((Time.time * speed)));
        transform.position = a;
    }
}

```

Figure 4.11.1 Code snippets of movement implementations

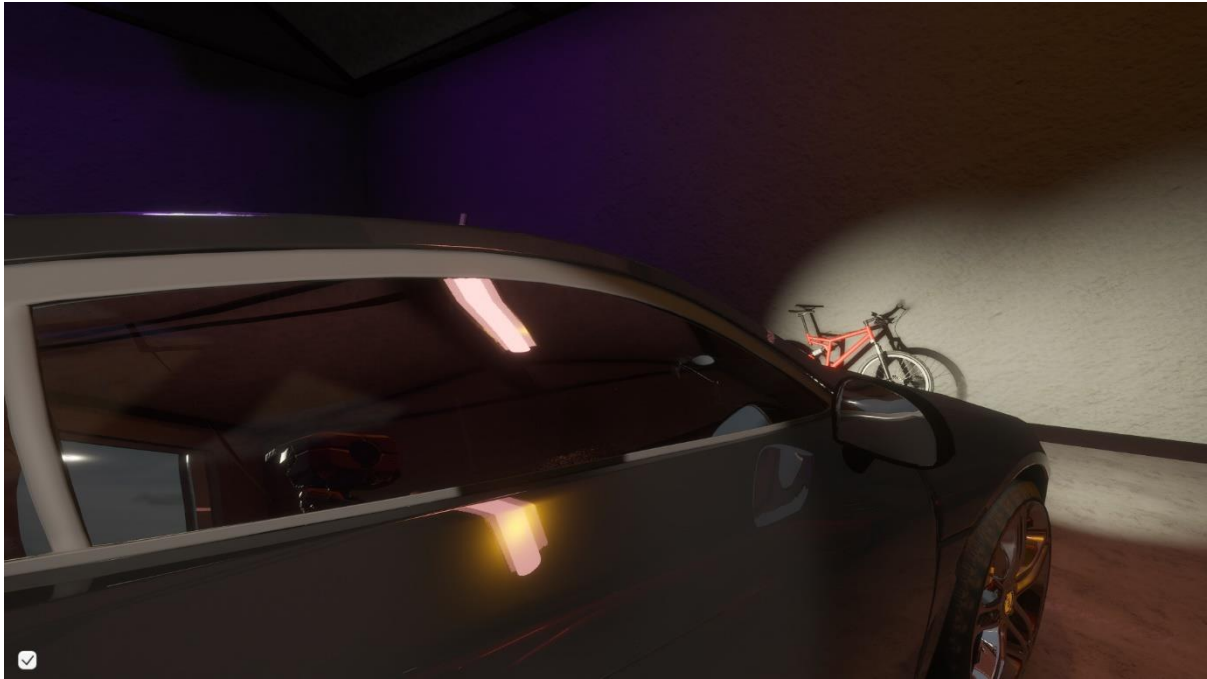


Figure 4.11.2 Showing the robot in the reflection of the car, showing how that the ray traced reflections are working in real-time

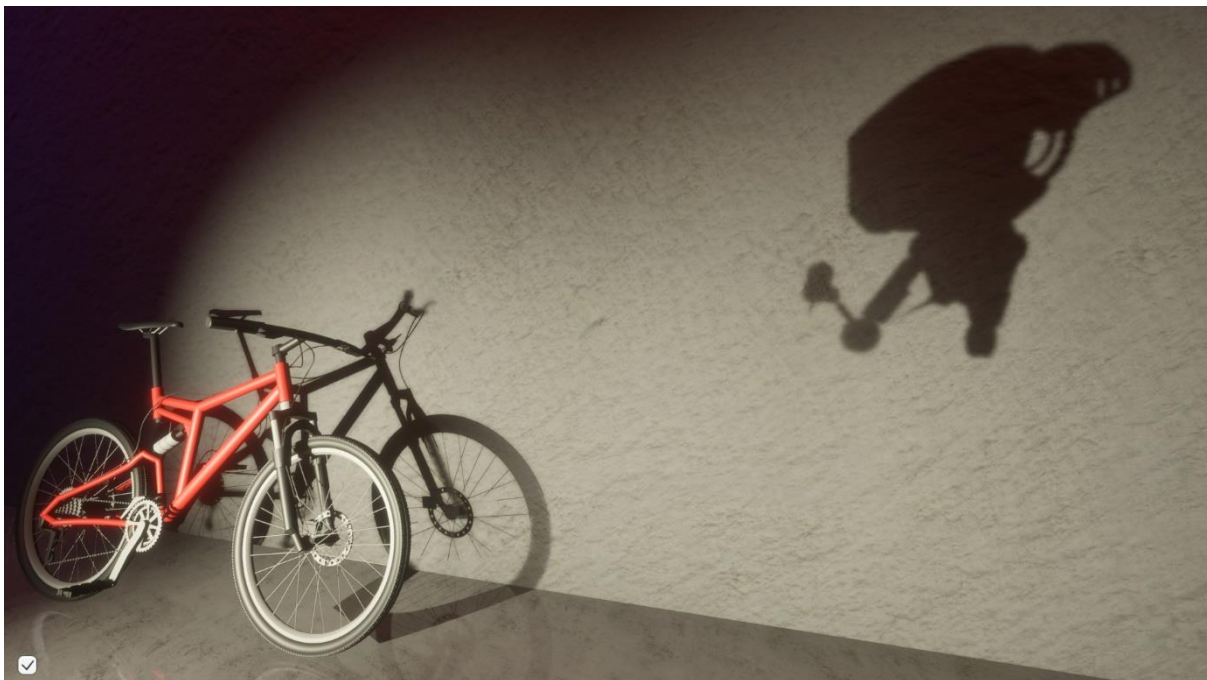


Figure 4.11.3 Showing the robots shadow against the wall, showing the ray traced shadows working in real-time

4.3.4 Ease of Use Implementations

In order to quickly toggle the ray tracing effects, to make things considerably easier when taking the images used in this report, a menu was implemented with several toggles for some of the effects on the ray tracing volume. It also included a toggle for hiding the menu itself, so that it was not in the way of the images. However, this menu does not include to toggle all the ray tracing features implemented, since some do not solely rely on the ray tracing volume and instead require changes on other objects such as materials and lights. This menu helped deal with low framerates experienced when moving about the scene, such as for taking some of

the images. An FPS Counter script also helped to monitor the frames per second during runtime, in preparation for comparisons (Hampson, 2020).

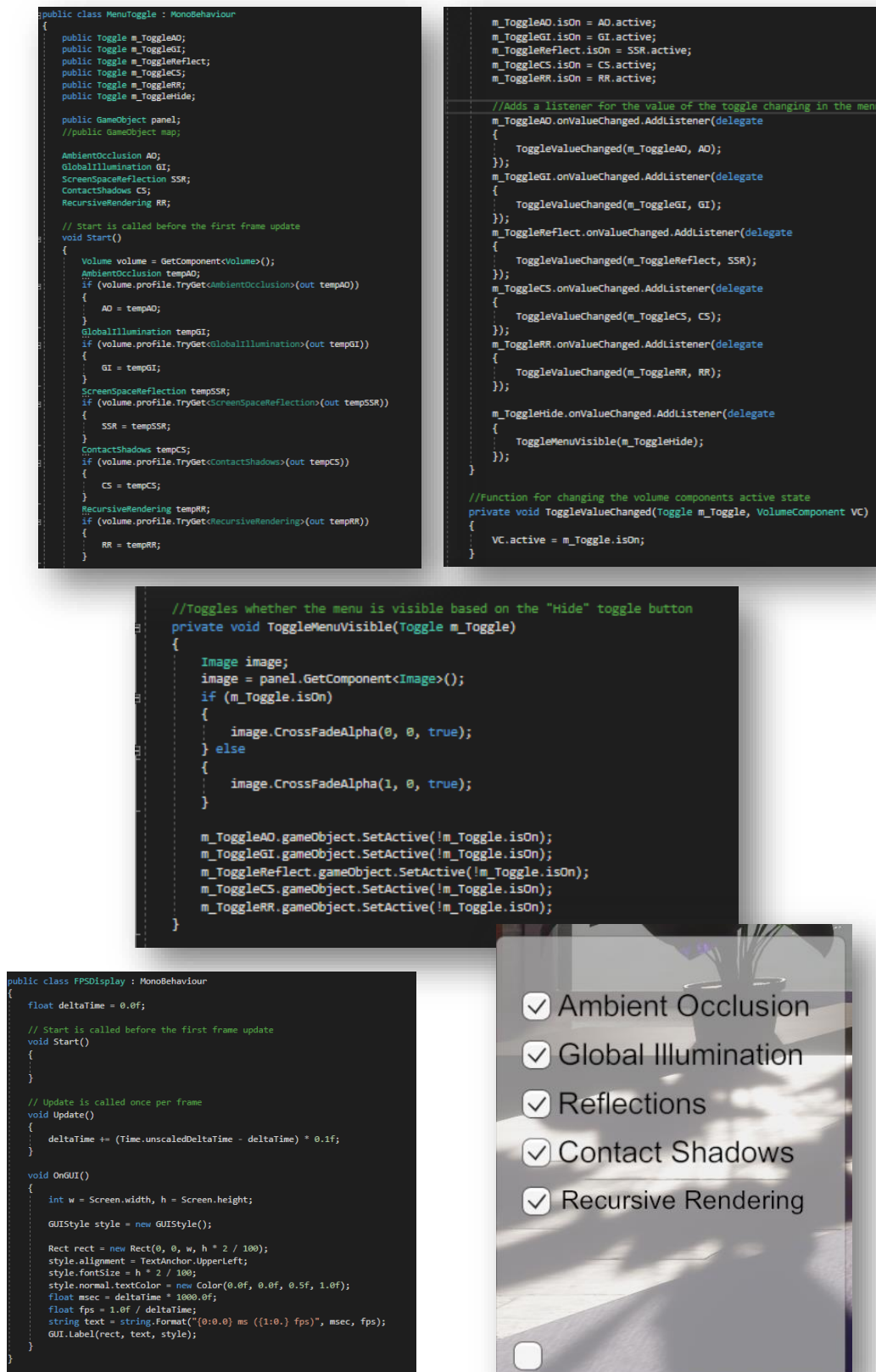


Figure 4.12 Code snippets and image of menu

5. COMPARISONS

5.1 VISUAL IMPROVEMENTS

In order to show the true visual benefits of real time ray tracing techniques, they are best compared to some of the more common rasterization techniques that the High Definition Rendering Pipeline uses when ray tracing is not enabled. In order to show this, the project was originally created from a version that did not implement any of the DirectX Ray Tracing configuration, but instead multiple instances of the project was made with the help of source control and further developed with the different rendering techniques.

5.1.1 Rasterization

To compare the rasterization methods, some of the Screen Space methods were used, such as Screen Space Shadows, Screen Space Reflections and Screen Space Refraction. Along with this, global illumination and ambient occlusion were used through lightmaps and light probes. As well as the Screen Space additions, Reflection probes and Planar reflection probes were added, as screen space reflection does not fully capture the reflections on its own but provides closer detail reflections. The reflection probes were placed around both cars, and another encapsulating the whole garage, while slightly more resource costly planar reflection probes were used on the floor, windows, and mirrors. The following images were taken from the scene using these methods:



Figure 5.1.1 Rasterization Image 1



Figure 5.1.2 Rasterization Image 2

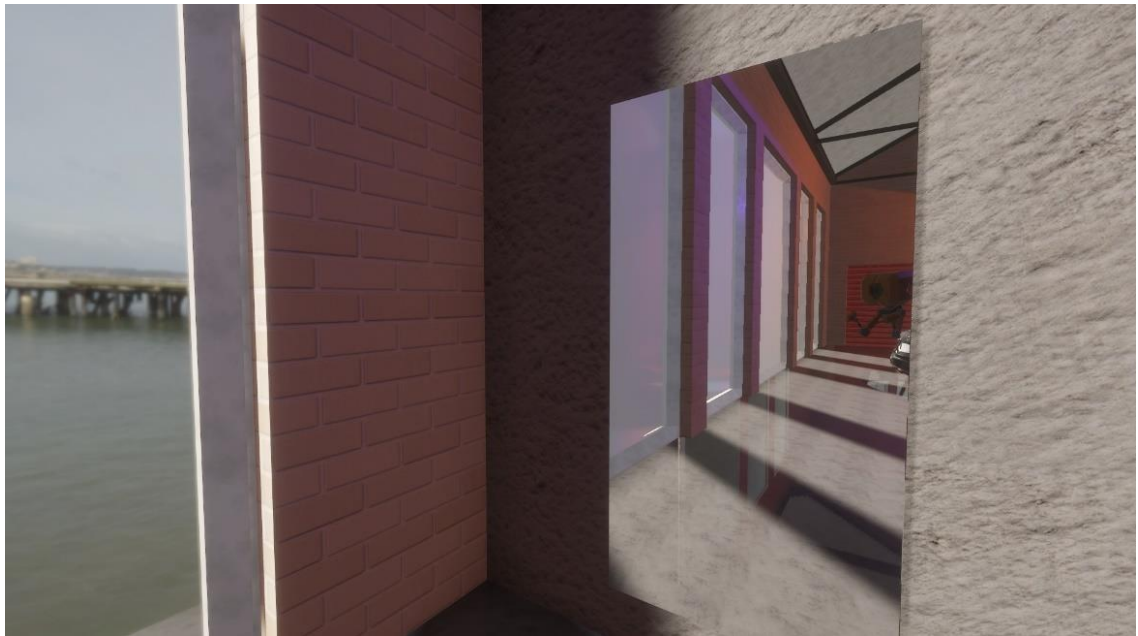


Figure 5.1.3 Rasterization Image 3



Figure 5.1.4 Rasterization Image 4

5.1.2 Real-Time Ray Tracing

The real-time ray tracing images here were taken from the same as the implementation section of this report:

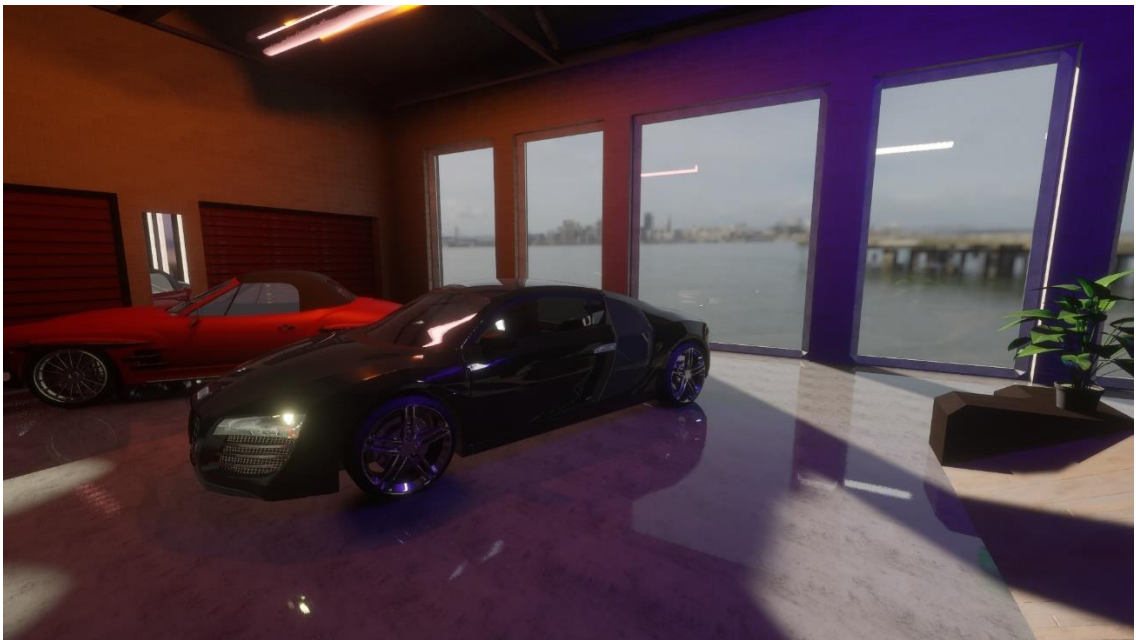


Figure 5.2.1 Ray Tracing Image 1



Figure 5.2.2 Ray Tracing Image 2



Figure 5.2.3 Ray Tracing Image 3



Figure 5.2.4 Ray Tracing Image 4

5.1.3 The Differences

Quality

The primary difference between the set of images is the quality, where the ray tracing implementation is considerably more realistic, however this does not mean that the rasterized implementation is not bad, by any means, with it still quite clearly showing high quality reflections, shadows and lighting.

However, the quality with the screen space and probes images is mostly thanks to the planar reflection probes. This can be seen in Rasterization Images 3 and 4, with the mirror using planar reflection probes and providing an accurate reflection, while the garage door that is simply inside of the regular reflection probes, lacks in some quality, with the reflection probes often having issues with depth, such as with the windows on the right side of the garage door reflection. The red car should be more of a matte colour, like in the ray traced image 2, but the rasterization has almost too much of a reflection without changing the material. Comparing this to the ray traced views and there is a clear quality improvement.

Reflection probes themselves pick up transparency, however planar reflection probes, as seen in rasterization image 3, currently does not support reflecting transparent objects, something which ray tracing handles well, thanks to reflected rays continuing to pass through the object with refraction.

The lighting and shadows in the rasterized images are not the same as in the ray tracing view, however the light probes improve this and add more depth to the global illumination. The shadows themselves are good quality, however in some case lack definition and accuracy when compared to the ray tracing version.

Performance

The performance differences here are huge, with the ray traced implementation being very slow to load, while the rasterized view is considerably smoother in performance. The statistics behind this will be discussed further in the next section. Nonetheless, choosing between the two, or a combination of the two, ensures optimal quality and performance.

Implementing

Implementing the two are relatively simple thanks to Unity, however the rasterization method requires a lot of smaller changes, such as positioning the light and reflection probes in the scene as best as possible. Also ensuring any of the screen space settings in the volume component are set correctly, while the ray tracing implementation consists of modifying volume settings and materials and the positioning is not really an issue.

5.2 STATISTICS AND OPTIMISATIONS

Real time ray tracing has serious impacts on the performance, therefore understanding how much you can implement while keeping the game playable with reasonable hardware is important. Here I will be reviewing frame rates and system performance with various configurations of the real-time ray tracing features, such as how many objects are ray traced, the individual ray tracing volume settings and the effects of each ray tracing type.

5.2.1 Frame Rates

This section will contain a series of screenshots with some frame rates taken during runtime of the various builds for this project. Of course, a lot more optimisations can be made in order to further increase the fps, however these are the values for the current implementations.

Quality Real-Time Ray Tracing

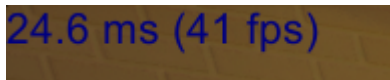
The higher quality real-time ray tracing version that mostly makes up this report, generally stayed at around 3 frames-per-second when moving throughout the scene:



328.7 ms (3 fps)

Rasterized version

The Rasterized version stayed at around 41 frames-per-second when moving throughout the scene:



24.6 ms (41 fps)

Ray Tracing Without Recursive Rendering

Without Recursive rendering, an increase in the bounce count of the ray traced reflections, and removing ray traced shadows on the ceiling lights, does not actually provide too much benefit in terms of performance, with values of around 5 fps:



213.4 ms (5 fps)

Ray Tracing without Ray Traced Global Illumination and Ray Traced Ambient Occlusion

This example is the fps with global illumination and ambient occlusion disabled, two potentially unnecessary additions – depending on the type of game being made. Additionally the global illumination effects are slightly covered using light probes:



112.0 ms (9 fps)

5.2.2 Hardware Usage

This section will show how real-time ray tracing builds effect the system, further showing how costly it is on computational resources. It uses AORUS ENGINE's monitoring tool, that came with the RTX 2060 used, and from windows task manager.

Graphical Processing Unit

As we are rendering graphics here, it makes sense that the GPU is going to give us the best results when reviewing effects on the hardware. Note that the images here are taken with the GPU automatically scanning when it should boost its performance:

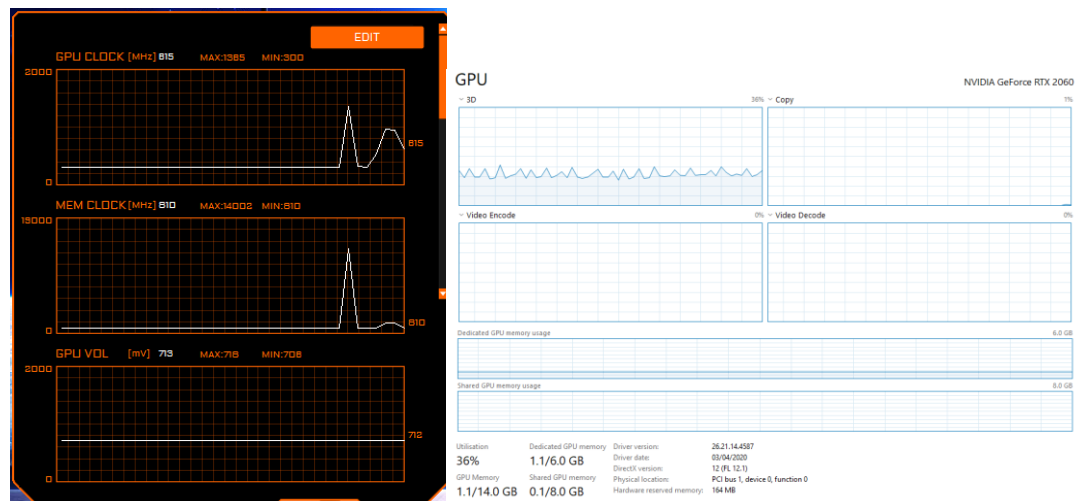


Figure 5.3.1 Baseline GPU levels at a relatively idle level

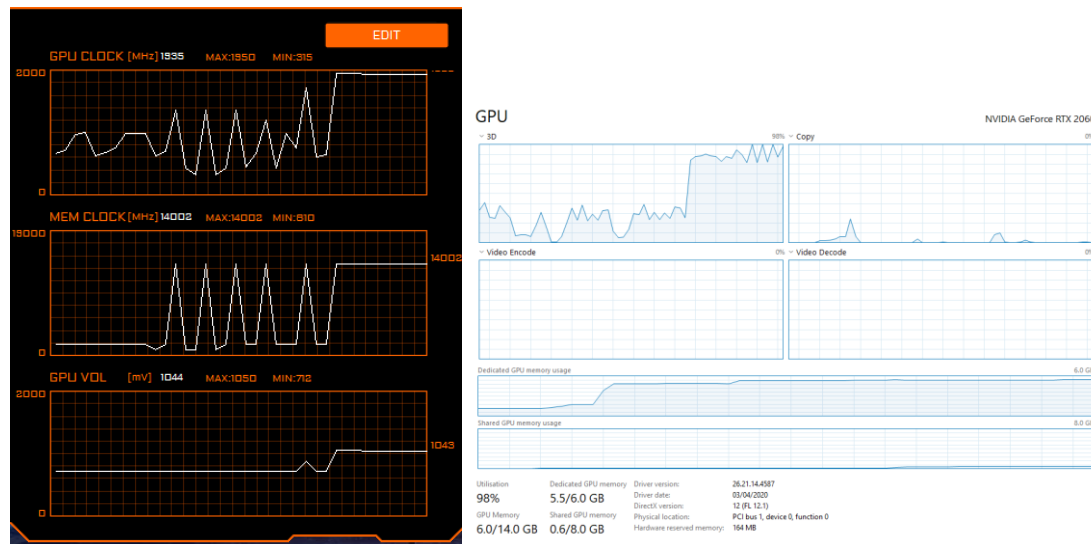


Figure 5.3.2 Runtime statistics of the highest quality ray tracing build primarily used in this report, the GPU shows an incredible increase in speed and usage of all of its resources.

Central Processing Unity

The CPU is primarily used in the initial rendering of the scene, therefore during runtime, does not have as much of an impact on graphics in this situation, however games with much larger codebases and require constant rendering based on view distances, can have impacts on the CPU:

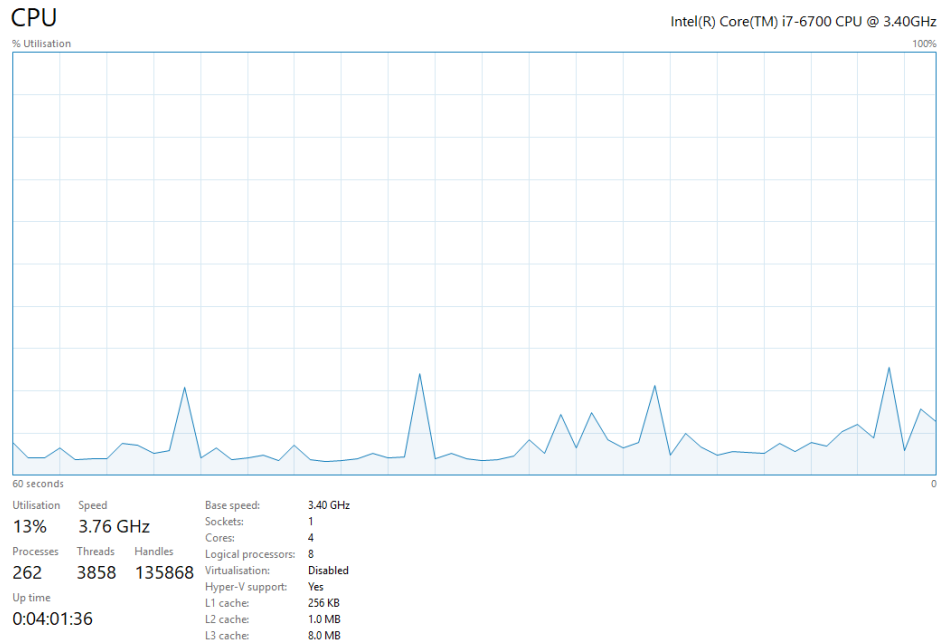


Figure 5.4.1 An idle state of the CPU

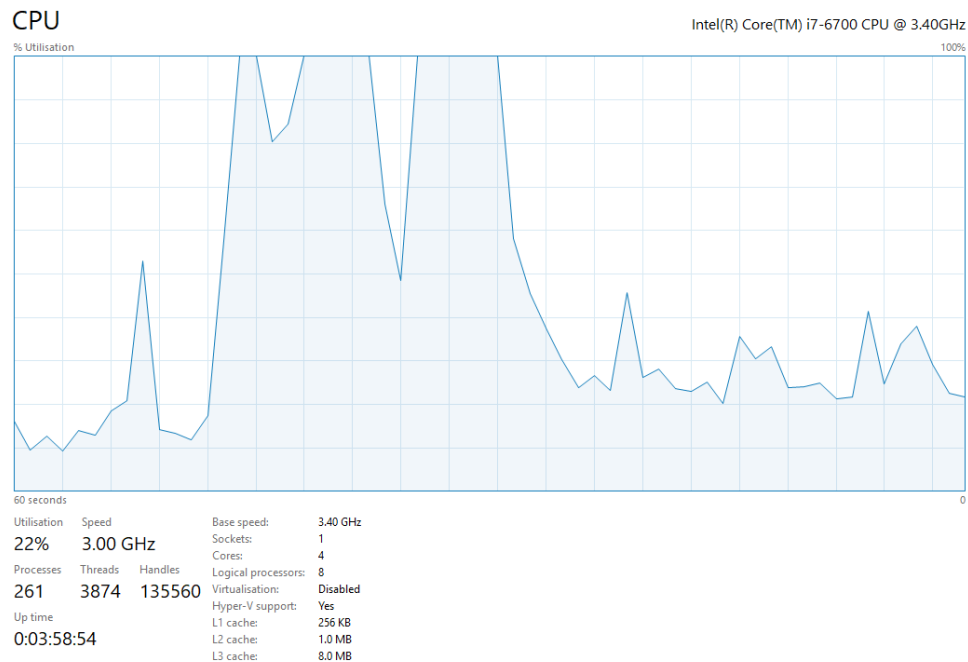


Figure 5.4.2 The CPU during the start-up and into runtime of the real-time ray tracing implementation

6. CRITICAL EVALUATION

6.1 OVERVIEW

This chapter will discuss several ways in which the project went well, as well as improvements that could have been made and areas that may not have been fully considered during the development of the project, despite hitting most of the aims for the project. It will be a reflection on both my own work and to how it compares to that of the original specification for the project, but also what could happen in the future with real-time ray tracing and the optimisations that could have been made to the project to further improve it and show off the benefits and capabilities with ray tracing and the hardware support that is currently available for it.

6.2 SPECIFICATION

The project mostly stayed true to the specification, however throughout researching the topic and understanding what it was I wanted to focus on, it did change slightly. The main change was looking into the mathematics behind ray tracing, which did not really fit in with the context of the report. My primary aim was to provide a comparison of the different features within real-time ray tracing using Unity, not to go into as much depth as looking into the mathematics behind it. Another change involved adding an extra layer of comparison in the form of the probe's version, which used reflection probes and light probes. This change helps to solidify the benefits of the real-time ray tracing effects, by using another common technique used within Unity to improve graphics.

Part of the specification was having a schedule of development for both the scene and report. This schedule also changed drastically throughout the scene deliverable section of the project, as discovering how real-time ray tracing is implemented into Unity caused tasks to be flipped, in the sense of turning on the ray tracing had to be done at the end instead of the start. In the end I was on track and it did not affect the overall time.

6.3 PROCESS IMPROVEMENTS

The overall process of the project has gone well, however there could have been some improvements in order to provide a better comparison of ray tracing effects and how the project was developed.

My personal organisation of the project development involved using Trello, which really helped in understanding my timetable for deadlines throughout the project and the guidelines for when I'd like parts of the scene to be completed, therefore I am happy with my decision to utilise this tool. During the development process I kept my Unity project file up to date with source control, however this process was not as smooth as I would have liked. My main issue was partially my own doing, as my repository exceeded the maximum file size due to incorrect use of the git ignore file (it was in the wrong place). I resolved this by creating a new repository and ensuring the git ignore was in the correct place, which allows the project to be under the 2GB limit. After that, not using Git Large File storage for my assets, meant the repository filled up again and I could no longer push my changes, therefore another repository was made where Git LFS was installed. My project initially used 3 different versions – a regular version without any graphical enhancements, a version with light and reflection probes and a version with the

ray tracing features, so I figured the best way to handle this would be by using branches in source control. This worked well, to a degree, when I created the scene without any probes or ray tracing and then created the branch and modified the others, however I further modified the original version and simply merging over the scene into the other branches would have removed some of the features exclusive to the ray tracing and probes branches, therefore the process of copying scripts and objects was longer than it could have been, if I had finished the scene fully without any of the graphical modifications. This is also changed, as the benefits of the version without any graphical enhancements didn't really fit in as a valid comparison, therefore I opted to simply alter some of the ray tracing features to provide more of a performance friendly ray tracing version.

One of my biggest challenges with the project was obtaining the assets and materials for the scene, as creating all the assets would have taken time, but also, I do not have much experience in 3D Modelling and creating high definition textures. My methods around creating the scene, involved doing some general simple shapes, such as the building, using ProBuilder and sourcing the rest from (mostly) TurboSquid. The materials were tricky, due to the difference between Unity using DXR, which required compatible shaders (primarily the built in lit shader), this meant that some of the package shaders I could acquire from the Unity asset store didn't really work, so it was a long process to go through each one and manually convert them to use the lit shader. Creating my own textures had to be straight forward and would not require too much effort, as I have little experience in creating textures. Thankfully, I found a handy tool named materialize, which allowed me to create diffuse, normal, metallic, smoothness and height maps from simply a single texture image that I found online. Of course this had to be seamless, so I had to take that into account as well. The last challenge was converting materials some into to have a mask map for the lit shader, so using GIMP with a plugin and some research made this possible very quickly.

6.4 HARDWARE AND OPTIMISATIONS

In order to do this project, it required specialist hardware that initially I did not have outside of the university labs. Real-Time ray tracing in Unity is only possible using the Nvidia RTX cards, therefore I would have to spend a lot of time in University or purchase one myself. I soon realised towards the beginning of development on the project that my best option here was to acquire one myself so that I generally had better access and more time could be spent on the project. There were three options, RTX 2060, RTX 2070 or RTX 2080 ranging in price from roughly £300 to £800. The most feasible option for me was the lower cost RTX 2060, as it meant I could do the project without spending more than I would have liked. This does however mean there are improvements that could have been made to the project provided a higher spec GPU was used, as greater performance and capabilities could be acquired at better framerates, for example increasing the number of bounces the ray takes.

However, this does not mean the lower spec graphics card is not capable of more, as various optimisations can be used to make the most of the architecture and being smart about what objects utilise the real-time ray tracing feature. I found throughout the project that applying ray tracing to all the materials was having serious impacts on the frame rate, therefore I needed to prioritise objects that show off the features the best. As my project is about Shadows, Reflections and Refractions, this meant reflective and refractive objects should have this, but not all - in order to show some differences. This meant objects such as one of the cars, the mirrors and the glass windows would be good choices. However the car is a large object, so

applying the ray traced reflections to this is very resource intensive, and it shows when the scene is run. Similarly I wanted to show the effects of number of bounces, so you can see the reflection of the windows in the reflection of the mirrors for example, but increasing this number also means there are more calculations to be made to determine the resulting pixel. Another large factor is the use of global illumination on the scene; however I feel this is necessary to show more accuracy with the shadows.

All of these decision are relevant to what is being produced and in this case it isn't really a large amount of graphics being shown, compared to an open world, therefore I tried to get the most out of the hardware; but clearly more optimisations can be made, as the scene in the real-time ray traced environment has a low framerate. This is a place I could have improved, as I did not do much research with optimisations, just the relatively obvious ones.

6.5 MODIFICATIONS & GENERAL ISSUES

Several modifications were made during this project, that primarily came as I researched more and more into the effects of ray tracing, how to apply them and what I wanted the project and report to focus on.

My first major modification came during the development process of the scene, for several reasons. The original plan was to enable the ray tracing effects within unity, then develop the scene there and disable the features for the comparison of the effects. I shortly discovered that it was not as simple as enabling the ray tracing effects, but instead you must upgrade the whole project to DirectX12, which has the DirectX Ray Tracing capabilities. You still must enable the ray tracing features after the upgrade; however this introduced a lot more functionality which could affect the performance and even graphical processing compared to that of DirectX11 which Unity uses by default. This could affect the results, therefore my process involved enabling DirectX Raytracing after the scene was (mostly) complete. Another reason why I had to do this, was due to a bug with one of the Unity extensions I used in building the scene. I discovered an issue regarding vertex selection with the ProBuilder package, specifically after the DirectX12 upgrade, which would crash the program. Due to this, I submitted a bug report and the Unity team responded saying they was able to replicate the issue, as the following image shows.

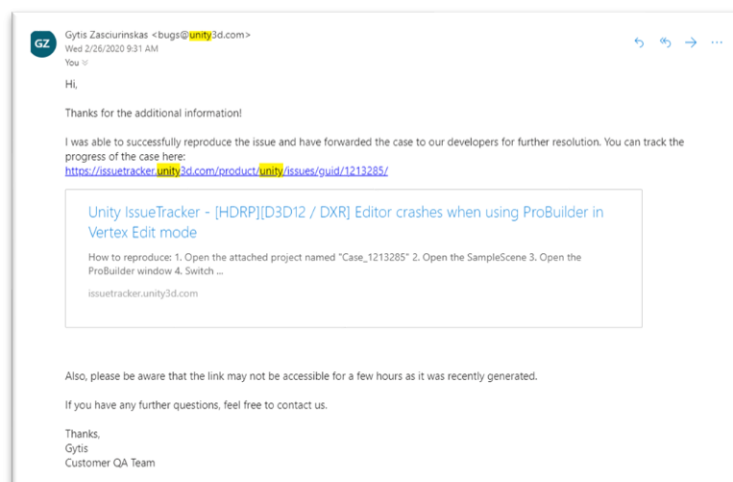


Figure 6.1 Unity Bug Report

Another issue I had to query the Unity forums for, was with a shadowing issue in the ray traced scene, which is still apparent and a problem. It was never made clear what the issue is, as a response for the issue was not made to the post in time:

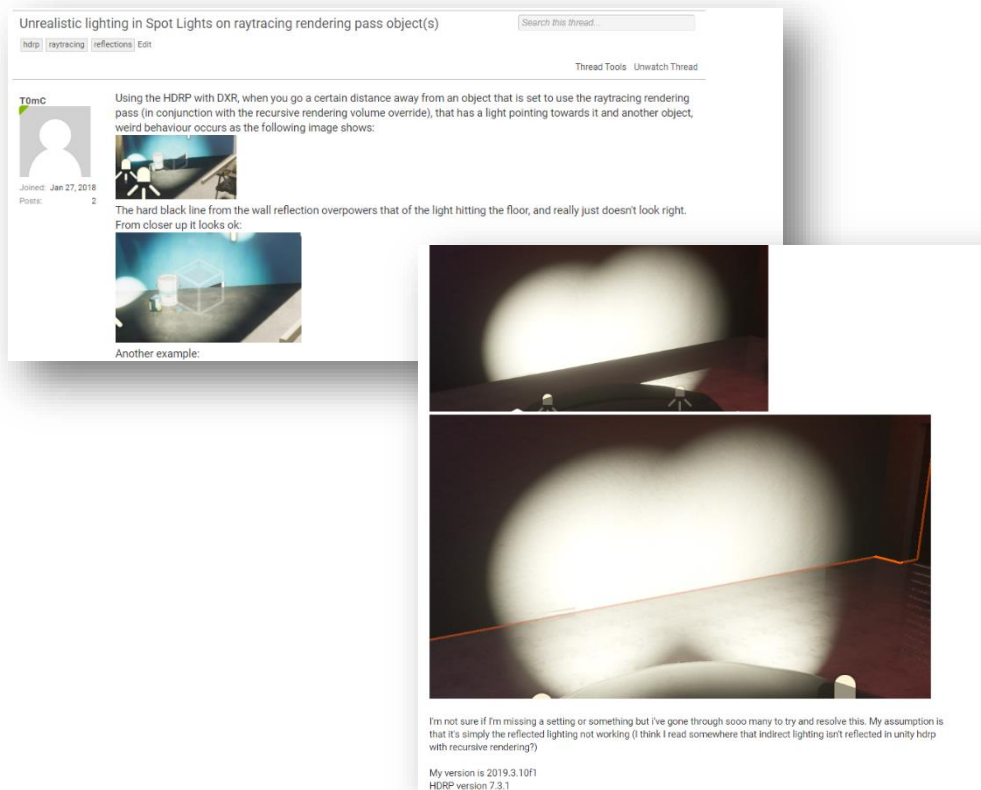


Figure 6.2 Unity Forum Post

6.6 PERSONAL REFLECTION

Overall I believe I have managed this project well and generally been well on time for the various deadlines, which is not easy when I have so many other pieces of work due. My original decision to use Trello was very useful and really helped with getting the right attitude towards making progress. I was open to the suggestions and ideas from my project supervisor and this made me more confident in what I was producing, in order to obtain a mark I can be proud of. There is one part of this project I believe I could've done better, which was during the start of the year I left the scene development later than I would have liked, which was a combination of not being at my computer due to the Christmas break, delaying in getting the correct hardware in order to actually be able to work on the scene from home and finally due to general after holiday lack of motivation. As I started to develop in smaller chunks however, I motivated myself to focus more and create the final scene how I envisioned it from the start. There was one moment in the project where I hit somewhat of a mental block when thinking about how I would acquire the relevant assets for the scene, but I committed myself to having to do a lot more work in researching and creating the assets rather than trying to find them all online, since this was proving difficult. In the end I think the final product looks good considering my little design experience.

6.7 FUTURE WORK

Real-time Ray Tracing using the current advances in Graphical hardware has been almost revolutionary, however it is still quite a way off efficient and larger use in games. Looking towards the future the architectures and hardware available can only adapt to improve the performance, with the market bolstered by the introduction of ray tracing in the next generation of game consoles. As for this project, it would likely be improved in the future by updates to the Unity engine, more specifically the integration of DirectX Ray Tracing in the High Definition Rendering Pipeline, as it is a relatively new technology added to the engine. Furthermore the scene I have created is not very large, and applying this to a much larger scene, maybe with some natural additions such as grass and trees, could show off a more realistic use of the technology in games.

There was an additional setting that could have been added to the real-time ray tracing implementation for this project, which involved the use of Path Tracing. This feature is one of the most resource intensive techniques and is currently not a feasible addition to ray tracing due to the amount of noise it adds to the scene and resources it requires, at least without a very expensive piece of hardware. Due to this, path tracing is more regularly used in offline rendering. The idea of path tracing is rays are distributed with each pixel in camera space and upon intersection with an object, the ray is then randomly distributed hundreds of times in various directions, where it then hits another surface, repeats the same process, until it reaches the maximum number of bounces and exits the scene, or it hits a light source. The sample values (shading values for each pixel) is then averaged to provide the resulting pixel colour (Dusterwald, 2016). This feature could make the scene look considerably better, and fix any issues produced by, for example, indirect lighting with shadows in reflections, however it is not currently at a place with affordable hardware for now to be integrated.

REFERENCES

Pixar (2006). Cars – Disney / Pixar - <https://www.pixar.com/feature-films/cars>

Pixar (2013). Monsters University – Disney / Pixar - <https://www.pixar.com/feature-films/monsters-university>

Per H. Christensen et al. (2006). Ray Tracing for the Movie ‘Cars’
<https://graphics.pixar.com/library/RayTracingCars/paper.pdf>

Behmanesh, Ali Asghar & Pourbahrami, Shahin & Gholizadeh, Behrouz (2012). International Journal of Computer Graphics & Animation (IJCGA) Vol 2, No.4 – Reducing Render Time In Ray Tracing By Pixel Averaging <http://airccse.org/journal/ijcga/papers/2412ijcga02.pdf>

Stitch, Martin (2018). Nvidia Corporation – Real-Time Ray Tracing with Nvidia RTX <http://on-demand.gputechconf.com/gtc-eu/2018/pdf/e8527-real-time-ray-tracing-with-nvidia-rtx.pdf>

Caulfield, Brian (2018). Nvidia Corporation – What’s the Difference Between Ray Tracing and Rasterization? <https://blogs.nvidia.com/blog/2018/03/19/whats-difference-between-ray-tracing-rasterization/>

Unity Technologies (2019). Unity Technologies – Manual High Definition RP | 7.1.6 / Manual / Ray Tracing / Effects and Volume Overrides / Ray-Traced Global Illumination
<https://docs.unity3d.com/Packages/com.unity.render-pipelines.high-definition@7.1/manual/Ray-Traced-Global-Illumination.html>

Haines, Eric & Akenine-Möller, Tomas et al. (2019). Ray Tracing Gems – High-Quality and Real-Time Rendering with DXR and other APIs.

Ballard, John (2018) The Motley Fool – NVIDIA Just changed the future of Gaming.
<https://www.fool.com/investing/2018/08/26/nvidia-just-changed-the-future-of-gaming.aspx>

Lefrancois, Martin-Karl & Gautron, Pascal (2018). Nvidia Corporation (Developer Blog) – Video Series: Practical Real-Time Ray Tracing With RTX <https://devblogs.nvidia.com/practical-real-time-ray-tracing-rtx/>

Martin, Edward (2019) Unity Technologies. Ray Tracing – What does it mean to you?
<https://blogs.unity3d.com/2019/04/26/ray-tracing-what-does-it-mean-to-you/>

Unity Technologies (2019). Unity Technologies Manual – High Definition RP | 7.3.1 / Manual / Ray Tracing / Getting Started with Ray Tracing <https://docs.unity3d.com/Packages/com.unity.render-pipelines.high-definition@7.3/manual/Ray-Tracing-Getting-Started.html>

Unreal Engine (2018) Reflections Real-Time Ray Tracing Demo | Project Spotlight | Unreal Engine
<https://www.youtube.com/watch?v=J3ue35ago3Y>

Gordon, Whitson (2019) IGN - PS5: What is Ray Tracing, and Should You Care? (Amazing lighting, but will it be worth it?) <https://uk.ign.com/articles/2019/10/10/what-is-ray-tracing-and-should-you-care>

Palumbo, Alessio (2019) WCCFtech – NVIDIA CEO: RTX Is a Home Run; I Look Forward to Upgrading Hundreds of Millions of PC Gamers. <https://wccfttech.com/nvidia-ceo-rtx-is-a-home-run-i-look-forward-to-upgrading-hundreds-of-millions-of-pc-gamers/>

Willings, A. (2019). What is ray tracing and what hardware and games support it? Retrieved from <https://www.pocket-lint.com/games/news/nvidia/148279-what-is-ray-tracing-and-what-hardware-and-games-support-it>

Lagarde, S. (2018). The high definition render pipeline: Focused on visual quality - unity technologies blog. Retrieved from <https://blogs.unity3d.com/2018/03/16/the-high-definition-render-pipeline-focused-on-visual-quality/>

Barton, S. (2019, -06-17T13:33:37+00:00). In the pipeline: Unity's HDRP brings next-gen graphics to the world's most-used game engine. Retrieved from <https://www.mcvuk.com/development-news/in-the-pipeline-unitys-hdrp-brings-next-gen-graphics-to-the-worlds-most-used-game-engine/>

Peddie, J. (2019, -12-06T13:59:16+00:00). Unity HDRP and all those rays. Retrieved from <https://gfxspeak.com/2019/12/06/unity-hdrp-those/>

Benyoub, A. (2020). High definition render pipeline real-time ray tracing is now in preview - unity technologies blog. Retrieved from <https://blogs.unity3d.com/2020/03/06/high-definition-render-pipeline-real-time-ray-tracing-is-now-in-preview/>

Lagarde, S. (2020). HDRP: Out of preview in 2019.3 - unity technologies blog. Retrieved from <https://blogs.unity3d.com/2020/02/24/hdrp-out-of-preview-in-2019-3/>

Owens, B. Forward rendering vs. deferred rendering. Retrieved from <https://gamedevelopment.tutsplus.com/articles/forward-rendering-vs-deferred-rendering--gamedev-12342>

De Vries, J. (2014). Deferred shading. Retrieved from <https://learnopengl.com/Advanced-Lighting/Deferred-Shading>

Stewart, S. (2020). What is ambient occlusion? Retrieved from <https://www.gamingscan.com/what-is-ambient-occlusion/>

Hampson, D. FramesPerSecond - unify community wiki. Retrieved from https://wiki.unity3d.com/index.php/FramesPerSecond?_ga=2.238278021.42919712.1586816516-2083943077.1586621079

Dusterwald, S. (2016, -07-04T12:03:03+00:00). Path tracing vs ray tracing. Retrieved from <https://www.dusterwald.com/2016/07/path-tracing-vs-ray-tracing/>

BIBLIOGRAPHY

Technologies, Unity real-time development platform | 3D, 2D VR & AR visualizations. Retrieved from <https://unity.com/>

Epic Games.Unreal engine | the most powerful real-time 3D creation platform. Retrieved from <https://www.unrealengine.com/en-US/>

Microsoft.Office 365 | microsoft office. Retrieved from <https://www.office.com/>

Gimp. Retrieved from <https://www.gimp.org/>

Bounding Box Software.Bounding box software - materialize. Retrieved from <https://boundingboxsoftware.com/materialize/>

Atlassian.Bitbucket | the git solution for professional teams. Retrieved from <https://bitbucket.org/product>

Atlassian.Sourcetree | free git GUI for mac and windows. Retrieved from <https://www.sourcetreeapp.com>

TurboSquid.3D models for professionals :: TurboSquid. Retrieved from <https://www.turbosquid.com/>

Unity.Download unity! Retrieved from <https://unity3d.com/get-unity/download>

Unity Technologies.High definition render pipeline wizard | high definition RP | 7.3.1. Retrieved from <https://docs.unity3d.com/Packages/com.unity.render-pipelines.high-definition@7.3/manual/Render-Pipeline-Wizard.html>

Tjan, R. (2018). Spotlight team best practices: Making believable visuals in unity - unity technologies blog. Retrieved from <https://blogs.unity3d.com/2018/03/09/spotlight-team-best-practices-making-believable-visuals-in-unity/>

Unity Technologies.Ray-traced ambient occlusion | high definition RP | 7.3.1. Retrieved from <https://docs.unity3d.com/Packages/com.unity.render-pipelines.high-definition@7.3/manual/Ray-Traced-Ambient-Occlusion.html>

Unity Technologies.Ray-traced global illumination | high definition RP | 7.3.1. Retrieved from <https://docs.unity3d.com/Packages/com.unity.render-pipelines.high-definition@7.3/manual/Ray-Traced-Global-Illumination.html>

Unity Technologies.Ray-traced shadows | high definition RP | 7.3.1. Retrieved from <https://docs.unity3d.com/Packages/com.unity.render-pipelines.high-definition@7.3/manual/Ray-Traced-Shadows.html>

Unity Technologies.Ray-traced contact shadows | high definition RP | 7.3.1. Retrieved from <https://docs.unity3d.com/Packages/com.unity.render-pipelines.high-definition@7.3/manual/Ray-Traced-Contact-Shadows.html>

Unity Technologies.Ray-traced reflections | high definition RP | 7.3.1. Retrieved from <https://docs.unity3d.com/Packages/com.unity.render-pipelines.high-definition@7.3/manual/Ray-Traced-Reflections.html>

Unity Technologies. Recursive rendering | high definition RP | 7.3.1. Retrieved from <https://docs.unity3d.com/Packages/com.unity.render-pipelines.high-definition@7.3/manual/Ray-Tracing-Recursive-Rendering.html>

Unity Technologies. SmallOfficeRayTracing. Retrieved from <https://github.com/Unity-Technologies/SmallOfficeRayTracing>

Desmonster. (2016). *Duesen bayern mystar 190 SL (car in scene)*.
<https://www.turbosquid.com/FullPreview/Index.cfm/ID/1062796>:

ioannespavlvs12. (2018). *3D audi R8 3D model model (car in scene)*.
<https://www.turbosquid.com/3d-models/3d-audi-r8-model-1283969>:

fernandoh2m. (2016). *Yellow robot (part of the camera in scene)*.
<https://www.turbosquid.com/FullPreview/Index.cfm/ID/1006610>:

petoboso. (2009). *Bicycle (bicycle in scene)*.
<https://www.turbosquid.com/FullPreview/Index.cfm/ID/503602>:

APPENDICES

APPENDIX A

PROJECT SPECIFICATION - Project (Technical Computing) 2019/20

Student:	Thomas Clark
Date:	21/10/19
Supervisor:	Thomas Sampson
Degree Course:	Computer Science for Games
Title of Project:	An Investigation into the Application of Real-Time Ray Tracing on Rendered Material Surfaces in Games

Elaboration

Real-Time ray tracing is a technique of which Game developers have been aware of for some years in order to improve the graphics to a more realistic look. Unfortunately, the resources haven't quite been available for more regular use, however due to the sudden rise in more appropriate hardware and a higher demand for more visually powerful games, it is starting to come into fruition. The Aim of this project is to really show off the benefits that real-time ray tracing can have on graphics, more specifically on material surfaces, and explore why we are only just starting to see this technology appear in our games. It will also give a deeper understanding of the process behind the implementation and where it is most appropriate to use such a powerful graphical technology. To do so, I will be looking into the hardware and software statistics behind resource usage with and without real-time ray tracing, to help understand where it is most appropriate. Furthermore, I will produce a scene using Unity Engine that has various real-time ray tracing elements on various material surfaces, in order to show where it can be most beneficial. Determining whether something is actually better or not is based on perspective, therefore as part of the critical evaluation I will also be getting video game users views on the scene that has been created, where they answer some questions or provide ratings to prove how useful the technology actually is and whether they would like to see it grow into something that is regularly used in the video games that they play. Upon completion of this project, I should have a greater knowledge of real-time ray tracing, which is likely to be carried into the future and be very useful to take into the gaming industry.

Project Aims

This project is intended to accomplish the following:

- Learn how real-time ray tracing is developed on material surfaces inside of a game engine.
- Determine the requirements and decisions that must be made to implement real-time ray tracing in a successful and appropriate manner.
- Learn how real-time ray tracing is achieved mathematically.
- Produce a scene involving real-time ray tracing on material surfaces.
- Evaluate where real-time ray tracing currently is in the gaming industry and the possibilities of growth in the future.
- Compare the benefits of real-time ray tracing to other methods of graphical shading.

Project deliverable(s)

There will be two parts to this project:

- A scene developed using Unity Engine which displays the visual effects real-time ray tracing has on material surfaces. It will contain various real-time ray traced elements as well as normally shaded elements to really show the comparison.
- A report which details how real-time ray tracing is used both inside and outside of game development and the decisions developers must take to use it appropriately, provides an explanation of the maths behind real-time ray tracing using the previously mentioned scene as an example, how real-time ray tracing is accomplished from within Unity Engine and lastly a comparison of common shading/graphical techniques to that of real-time ray tracing.

Action plan

Find Project Supervisor	11/10/2019
First Supervisor Meeting	16/10/2019
Create Project planning board <ul style="list-style-type: none">- Use Trello to manage project and track time	25/10/2019
Project Specification and Ethics Form	25/10/2019
Setup version control for scene deliverable	15/11/2019
Create Unity Project file	15/11/2019
Research & Identify resources useful for gathering information <ul style="list-style-type: none">- Looking for books, websites, articles, guides and blogs	22/11/2019
Information Review	06/12/2019
Determine and acquire/create relevant assets for scene <ul style="list-style-type: none">- Objects that will use real-time ray tracing- The materials to be used and any effects on them- Any background environments and objects- Non real-time ray traced surfaces that will show some comparison	13/12/2019
Positioned scene in final form, with or without real-time ray tracing on some surfaces	17/01/2020
Create questionnaire / feedback form on scene for use in critical evaluation	31/01/2020
Investigate method for mathematically calculating a real-time ray traced polygon <ul style="list-style-type: none">- Researching how it works- Understanding fully how it works to be able to put into the report	07/02/2020
Provisional Contents Page <ul style="list-style-type: none">- Determine the sections that will go into the report	21/02/2020
Real-time ray tracing applied to most (if not all) of the objects on the scene <ul style="list-style-type: none">- A mostly finished scene deliverable- Should be ready for comparisons	21/02/2020
Testing to ensure Unity scene is working as it should under various conditions (to ensure accurate results are gathered when other users provide feedback)	21/02/2020
Acquired all feedback on scene and real-time ray tracing in general, in preparation for critical evaluation	28/02/2020

Review performance statistics of real-time ray tracing to understand the decisions that need to be made to implement it effectively	06/03/2020
Comparison of Real-time ray tracing to that of other shading methods in Games, using examples from deliverable scene <ul style="list-style-type: none"> - Lighting, Reflection and Refraction - Shadows and Self-Shadowing 	06/03/2020
Draft Critical Evaluation	27/03/2020
Sections of draft Report	27/03/2020
Body of Project Report	22/04/2020
Project Report submitted via Turnitin	23/04/2020
Project Deliverable	24/04/2020
Demonstration of Work	12/05/2020

BCS Code of Conduct

I confirm that I have successfully completed the BCS code of conduct on-line test with a mark of 70% or above. This is a condition of completing the Project (Technical Computing) module.

Signature: T.Clark

Publication of Work

I confirm that I understand the "Guidance on Publication Procedures" as described on the Bb site for the module.

Signature: T.Clark

GDPR

I confirm that I will use the "Participant Information Sheet" as a basis for any survey, questionnaire or participant testing materials. This form is available on the Bb site for the module.

Signature: T.Clark

Ethics

Complete the SHUREC 7 (research ethics checklist for students) form below. If you think that your project may include ethical issues that need resolving (working with vulnerable people, testing procedures etc.) then discuss this with your supervisor as soon as possible and comment further here.

Both you and your supervisor need to sign the completed SHUREC 7 form.

Please contact the project co-ordinator if further advice is needed.

APPENDIX B

RESEARCH ETHICS CHECKLIST FOR STUDENTS (SHUREC 7)

This form is designed to help students and their supervisors to complete an ethical scrutiny of proposed research. The SHU [Research Ethics Policy](#) should be consulted before completing the form.

Answering the questions below will help you decide whether your proposed research requires ethical review by a Designated Research Ethics Working Group.

The final responsibility for ensuring that ethical research practices are followed rests with the supervisor for student research.

Note that students and staff are responsible for making suitable arrangements for keeping data secure and, if relevant, for keeping the identity of participants anonymous. They are also responsible for following SHU guidelines about data encryption and research data management.

The form also enables the University and Faculty to keep a record confirming that research conducted has been subjected to ethical scrutiny.

For student projects, the form may be completed by the student and the supervisor and/or module leader (as applicable). In all cases, it should be counter-signed by the supervisor and/or module leader, and kept as a record showing that ethical scrutiny has occurred. Students should retain a copy for inclusion in their research projects, and staff should keep a copy in the student file.

Please note if it may be necessary to conduct a health and safety risk assessment for the proposed research. Further information can be obtained from the Faculty Safety Co-ordinator.

General Details

Name of student	Thomas Clark
SHU email address	B6015511@my.shu.ac.uk
Course or qualification (student)	Computer Science for Games
Name of supervisor	Thomas Sampson
email address	thomasreds@hotmail.com

Title of proposed research	An Investigation into the Application of Real-Time Ray Tracing on Rendered Material Surfaces in Games
Proposed start date	11/10/19
Proposed end date	12/05/19
Brief outline of research to include, rationale & aims (250-500 words).	<p>Real-Time ray tracing is a technique of which Game developers have been aware of for some years in order to improve the graphics to a more realistic look. Unfortunately, the resources haven't quite been available for more regular use, however due to the sudden rise in more appropriate hardware and a higher demand for more visually powerful games, it is starting to come into fruition.</p> <p>The Aim of this project is to really show off the benefits that real-time ray tracing can have on graphics, more specifically on material surfaces, and explore why we are only just starting to see this technology appear in our games. It will also give a deeper understanding of the process behind the implementation and where it is most appropriate to use such a powerful graphical technology.</p> <p>To do so, I will be looking into the hardware and software statistics behind resource usage with and without real-time ray tracing, to help understand where it is most appropriate. Furthermore, I will produce a scene using Unity Engine that has various real-time ray tracing elements on various material surfaces, in order to show where it can be most beneficial.</p> <p>Upon completion of this project, I should have a greater knowledge of real-time ray tracing, which is likely to be carried into the future and be very useful to take into the gaming industry.</p> <p>This project is intended to accomplish the following:</p> <ul style="list-style-type: none"> • Learn how real-time ray tracing is developed on material surfaces inside of a game engine. • Determine the requirements and decisions that must be made to implement real-time ray tracing in a successful and appropriate manner. • Learn how real-time ray tracing is achieved mathematically. • Produce a scene involving real-time ray tracing on material surfaces. • Evaluate where real-time ray tracing currently is in the gaming industry and the possibilities of growth in the future. • Compare the benefits of real-time ray tracing to other methods of graphical shading.
Where data is collected from individuals, outline the nature of data, details of anonymisation, storage and disposal procedures if required (250-500 words).	<p>There may be data acquired for the calculation values of effective and relevant real-time ray tracing, of which may come from any research areas, however this data may not necessarily be unique to an individual, but instead a general guideline. There also may be values relevant for lighting and textures to that of the graphics within the scene of the Unity Engine deliverable. All other values would be generated as appropriate, but mostly random or determined by myself.</p> <p>There won't be any personal/private data that would be stored, but some values may be used in the report and game engine, though</p>

	<p>would therefore remain anonymous in the event it is unethical to do otherwise. Any quotes or views from individuals will also be anonymous (unless a reference is required). In the event personal/private data is required for the duration of the project, it shall be removed and replaced with alternative/dummy data before it is finished.</p> <p>Data may be gathered for the critical evaluation, as views from users will be required to determine whether real-time ray tracing technology is seen as an overall beneficial technology to use in video games. Anything that is deemed personal or private will be removed and/or made anonymous; this may be done by getting overall statistics based on the answers/ratings.</p>
--	--

1. Health Related Research Involving the NHS or Social Care / Community Care or the Criminal Justice Service or with research participants unable to provide informed consent

Question	Yes/No
<p>1. Does the research involve?</p> <ul style="list-style-type: none"> • Patients recruited because of their past or present use of the NHS or Social Care • Relatives/carers of patients recruited because of their past or present use of the NHS or Social Care • Access to data, organs or other bodily material of past or present NHS patients • Foetal material and IVF involving NHS patients • The recently dead in NHS premises • Prisoners or others within the criminal justice system recruited for health-related research* • Police, court officials, prisoners or others within the criminal justice system* • Participants who are unable to provide informed consent due to their 	No
<p>2. Is this a research project as opposed to service evaluation or audit?</p> <p><i>For NHS definitions please see the following website</i></p> <p>http://www.hra.nhs.uk/documents/2013/09/defining-research.pdf</p>	No

If you have answered **YES** to questions **1 & 2** then you **must** seek the appropriate external approvals from the NHS, Social Care or the National Offender Management Service (NOMS) under their independent Research Governance schemes. Further information is provided below.

NHS <https://www.myresearchproject.org.uk/Signin.aspx>

* All prison projects also need National Offender Management Service (NOMS) Approval and Governor's Approval and may need Ministry of Justice approval. Further guidance at:

<http://www.hra.nhs.uk/research-community/applying-for-approvals/national-offender-management-service-noms/>

NB FRECs provide Independent Scientific Review for NHS or SC research and initial scrutiny for ethics applications as required for university sponsorship of the research. Applicants can use the NHS proforma and submit this initially to their FREC.

2. Research with Human Participants

Question	Yes/No
Does the research involve human participants? This includes surveys, questionnaires, observing behaviour etc.	Yes

Question	Yes/No
1. <i>Note If YES, then please answer questions 2 to 10 If NO, please go to Section 3</i>	
2. Will any of the participants be vulnerable? <i>Note: Vulnerable' people include children and young people, people with learning disabilities, people who may be limited by age or sickness, etc. See definition on website</i>	No
3. Are drugs, placebos or other substances (e.g. food substances, vitamins) to be administered to the study participants or will the study involve invasive, intrusive or potentially harmful procedures of any kind?	No
4. Will tissue samples (including blood) be obtained from participants?	No
5. Is pain or more than mild discomfort likely to result from the study?	No
6. Will the study involve prolonged or repetitive testing?	No
7. Is there any reasonable and foreseeable risk of physical or emotional harm to any of the participants? <i>Note: Harm may be caused by distressing or intrusive interview questions, uncomfortable procedures involving the participant, invasion of privacy, topics relating to highly personal information, topics relating to illegal activity, etc.</i>	No
8. Will anyone be taking part without giving their informed consent?	No
9. Is it covert research? <i>Note: 'Covert research' refers to research that is conducted without the knowledge of participants.</i>	No
10. Will the research output allow identification of any individual who has not given their express consent to be identified?	No

If you answered **YES only** to question **1**, the checklist should be saved and any course procedures for submission followed. If you have answered **YES** to any of the other questions you are **required** to submit a SHUREC8A (or 8B) to the FREC. If you answered **YES** to question **8** and participants cannot provide informed consent due to their incapacity you must obtain the appropriate approvals from the NHS research governance system. Your supervisor will advise.

3. Research in Organisations

Question	Yes/No
1. Will the research involve working with/within an organisation (e.g. school, business, charity, museum, government department, international agency, etc.)?	No
2. If you answered YES to question 1, do you have granted access to conduct the research? <i>If YES, students please show evidence to your supervisor. PI should retain safely.</i>	

<p>3. If you answered NO to question 2, is it because:</p> <p>A. you have not yet asked</p> <p>B. you have asked and not yet received an answer</p> <p>C. you have asked and been refused access.</p> <p><i>Note: You will only be able to start the research when you have been granted access.</i></p>	
--	--

4. Research with Products and Artefacts

Question	Yes/No
1. Will the research involve working with copyrighted documents, films, broadcasts, photographs, artworks, designs, products, programmes, databases, networks, processes, existing datasets or secure data?	Yes
<p>2. If you answered YES to question 1, are the materials you intend to use in the public domain?</p> <p><i>Notes: 'In the public domain' does not mean the same thing as 'publicly accessible'.</i></p> <ul style="list-style-type: none"> Information which is 'in the public domain' is no longer protected by copyright (i.e. copyright has either expired or been waived) and can be used without permission. Information which is 'publicly accessible' (e.g. TV broadcasts, websites, artworks, newspapers) is available for anyone to consult/view. It is still protected by copyright even if there is no copyright notice. In UK law, copyright protection is automatic and does not require a copyright statement, although it is always good practice to provide one. It is necessary to check the terms and conditions of use to find out exactly how the material may be reused etc. <p><i>If you answered YES to question 1, be aware that you may need to consider other ethics codes. For example, when conducting Internet research, consult the code of the Association of Internet Researchers; for educational research, consult the Code of Ethics of the British</i></p>	yes
<p>3. If you answered NO to question 2, do you have explicit permission to use these materials as data?</p> <p><i>If YES, please show evidence to your supervisor.</i></p>	
<p>4. If you answered NO to question 3, is it because:</p> <p>A. you have not yet asked permission</p> <p>B. you have asked and not yet received and answer</p> <p>C. you have asked and been refused access.</p>	A/B/C

Adherence to SHU policy and procedures

Personal statement

I can confirm that:	
<input type="checkbox"/> I have read the Sheffield Hallam University Research Ethics Policy and Procedures	
Student	
Name: Thomas Clark	Date: 22/10/19
Signature: T. Clark	
Supervisor or other person giving ethical sign-off	
I can confirm that completion of this form has not identified the need for ethical approval by the FREC or an NHS, Social Care or other external REC. The research will not commence until any approvals required under Sections 3 & 4 have been received.	
Name: Thomas Sampson	Date: 24/10/2019
Signature: T.Sampson	