Bachelor Project

Czech
Technical
University
in Prague

F3

Faculty of Electrical Engineering
Department of Computer Graphics and Interaction

Real-Time Ray Tracing in Unreal Engine

Vojtěch Vavera

Supervisor: doc. Ing. Jiří Bittner, Ph.D.
Field of study: Open Informatics
Subfield: Computer Games and Graphics
August 2020

ČVUT ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Vavera** Jméno: **Vojtěch** Osobní číslo: **474642**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačové grafiky a interakce**

Studijní program: **Otevřená informatika**

Studijní obor: **Počítačové hry a grafika**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Sledování paprsku v réalném čase v Unreal Enginu

Název bakalářské práce anglicky:

Real-Time Ray Tracing in Unreal Engine

Pokyny pro vypracování:

Zmapujte dostupná rozhraní pro zobrazování pomocí metody sledování paprsků v reálném čase. Nastudujte a popište možnosti efektů dosažitelných pomocí metod sledování paprsku v reálném čase dostupných v Unreal Enginu.
Vytvořte technologickou demonstrační aplikaci využívající sledování paprsků v Unreal Enginu a proveďte důkladné testy kvality a rychlosti zobrazování. Součástí demonstrační aplikace bude hratelné demo, jehož cílem je předvést schopnosti a omezení sledování paprsku v realném čase v herním nasazení.
Vyhodnoťte limity implementace z hlediska velikosti scény, možnosti jejich dynamických změn a složitosti zobrazovaných efektů. Proveďte základní uživatelský test vytvořeného dema, který vyhodnotí subjektivní vnímání efektů simulovaných sledováním paprsků ve srovnání se standardním zobrazovacím řetězcem.

Seznam doporučené literatury:

[1] Tomas Akenine-Moller et al. Real-Time Rendering (4th edition). CRC Press, 2018.
[2] Haines et al. Ray Tracing Gems, Apress, 2019.
[3] UE4 documentation, Ray-Tracing.

Jméno a pracoviště vedoucí(ho) bakalářské práce:

doc. Ing. Jiří Bittner, Ph.D., Katedra počítačové grafiky a interakce

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **11.02.2020** Termín odevzdání bakalářské práce: **14.08.2020**

Platnost zadání bakalářské práce: **30.09.2021**

doc. Ing. Jiří Bittner, Ph.D. podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Mgr. Petr Páta, Ph.D. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací.
Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

.

Datum převzetí zadání Podpis studenta

© VUT v Praze, Design: VUT v Praze, VIC

Acknowledgements

I would like to thank my supervisor, Dr. Jiri Bittner, for providing friendly guidance and important feedback throughout this project.

Declaration

I hereby declare that the present bachelor's thesis was composed by myself and that the work contained herein is my own. I also confirm that I have only used the specified resources.

Prague, 14. August 2020

Abstract

The purpose of this paper is to test, evaluate, and demonstrate the capabilities of Unreal Engine's implementation of ray tracing. The reader is introduced to the topic of real-time ray tracing, as well as the ray traced effects that are available in Unreal Engine. As part of the evaluation and testing of several sample scenes, tips and techniques on how to optimize the scenarios are given. Throughout the paper, we discuss the advantages and disadvantages of using ray tracing in games while giving performance figures to support the claims.

Keywords: real-time ray tracing, unreal engine, ray tracing in videogames, ray tracing

Supervisor: doc. Ing. Jiří Bittner, Ph.D.

Abstrakt

Cílem této práce je otestovat, vyhodnotit a přednést schopnosti implementace ray tracingu v Unreal Enginu. Čtenáři jsou představeny základní principy technologie Sledování Paprsků v Reálném Čase, spolu s efekty založenými na této technologii, jež jsou v Unreal Enginu dostupné. V návaznosti na testování několika různých scénářů jsou vysvětleny a doporučeny postupy pro optimalizaci scén, spolu s výhodami a nevýhodami používání ray tracingu.

Klíčová slova: sledování paprsků v reálném čase, unreal engine, ray tracing ve videohrách, ray tracing

Překlad názvu: Sledování paprsků v reálném čase v Unreal Enginu

Contents

1 Introduction **1**

2 What is Ray Tracing **3**

2.1 Basics of Ray Tracing 3

2.2 Ray Tracing APIs 5

2.2.1 NVIDIA OptiX 5

2.2.2 DirectX DXR 6

2.2.3 Vulkan Ray Tracing Extension 6

3 Ray Tracing in Unreal Engine **9**

3.1 Setting Up Ray Tracing 9

3.1.1 System Requirements 9

3.1.2 Configuring a New Project . . 10

3.2 Available Effects and their Settings 11

3.2.1 Shadows 12

3.2.2 Reflections 12

3.2.3 Translucency 15

3.2.4 Ambient Occlusion 16

3.2.5 Global Illumination 17

4 Benchmarks **19**

4.1 Testing Environment 19

4.2 Benchmark 1 - Room 20

4.3 Benchmark 2 - Balls 23

4.4 Benchmark 3 - Shore 25

4.5 Non-RTX Performance 27

4.6 Scene Scalability Testing 28

4.7 Unreal Engine Version Comparison 31

5 Performance Optimization **33**

5.1 Deciding Which Ray Tracing Effects to Utilize 33

5.2 Denoising vs Samples per Pixel . 34

5.3 Other Ray Tracing Settings 36

6 Unreal Engine Ray Tracing Demo **39**

6.1 Controls 39

6.2 Quality Settings 41

6.3 Quality Presets User Testing . . . 43

7 Conclusion **51**

A Bibliography **53**

Figures

2.1 An image showing an example of how a ray could traverse a scene. Source: nvidia.com. 4

2.2 Metro Exodus uses ray traced global illumination. The difference is especially visible in indoor scenes. Source: gamestar.de. 6

2.3 Ray tracing in Wolfenstein: Youngblood is handled by the Vulkan API. Notice the difference in reflections on the walls. Source: khronos.org. 7

3.1 NVIDIA's list of GPUs currently supporting DXR (excluding the newer, similiarly capable, SUPER series of RTX cards). Source: nvidia.com. 10

3.2 Properly set DirectX 12 launch parameter for the target .exe. 11

3.3 Comparison between traditional hard dynamic shadows and ray traced soft shadows. Ray traced shadows provide realistic smooth transition between lit and shadow areas. 12

3.4 The screen space reflection on the right side of this image lacks the ability to reflect objects outside of the camera's view frustum. 13

3.5 Comparison of ray traced reflections having: no shadows, hard shadows, and area shadows. 13

3.6 Small number of bounces between reflective objects can result in them appearing to be black. 14

3.7 The settings panel for controlling ray traced reflections. 14

3.8 The settings panel for controlling ray traced translucency. 15

3.9 Various refraction settings; Index of Refraction for this material is set to 1. 15

3.10 A comparison of several settings for the material's refraction index. 16

3.11 Ray traced ambient occlusion is more accurate and pronounced, adds depth to the scene. 16

3.12 The difference between ray traced and standard ambient occlusion is more visible when we hide all but the ambient occlusion layer. 17

3.13 The settings panel for controlling ray traced ambient occlusion. 17

3.14 This picture shows that the Final Gather method is susceptible to artifacts. However, these are not as prominent in well lit areas and scenarios. Both Final Gather and Brute Force use a single bounce and 8 samples in this image. 18

3.15 The settings panel for controlling ray traced global illumination. 18

4.1 Test bench hardware specifications. 20

4.2 Benchmark 1 scene using the HIGH preset. 20

4.3 Turning all the available ray traced effects on decreases performance severely, even on the RTX 2070S. . 21

4.4 Benchmark 1 test results for the four quality presets and both GPUs. 21

4.5 Comparison between the MEDIUM and HIGH presets of benchmark 1. 22

4.6 Comparison between the LOW and HIGH presets of benchmark 1. 22

4.7 Benchmark 2 using the HIGH preset. 23

4.8 Comparison between the OFF and HIGH presets of benchmark 2. 23

4.9 Benchmark 2 test results for the four quality presets and both GPUs. 24

4.10 Comparison between all the presets of benchmark 2. 24

4.11 Benchmark 3 using the HIGH preset. 25

4.12 Comparison between the OFF and HIGH presets of benchmark 3. 25

4.13 Benchmark 3 test results for the four quality presets and both GPUs. 26

4.14 Comparison between all the presets of benchmark 3. 27

4.15 This figure shows a the scene of the first test, with 1 and 48 sections on each side. 28

4.16 Final graph showing the ray tracing as well as rasterization data. 29

4.17 A figure showing what the ball test scene looked like. 30

4.18 Final graph for the balls test, both rendering techniques seem to have linear complexity. 30

4.19 This is the final table containing all the scalability testing data. 31

4.20 Comparison between the 4.23 and the 4.24 versions of Unreal Engine, using benchmark 2. 32

5.1 Comparison between the OFF and HIGH presets of benchmark 3. 34

5.2 Comparison between a denoised image and an image with higher sample count. 35

5.3 Visual artefacts introduced by the denoiser. 35

5.4 Demonstration of the Reflection Capture Fallback reflection option. 36

5.5 The RayTracingQualitySwitchReplace node in action. 37

6.1 Table of controls and their key bindings. 39

6.2 The menu, giving the options to play one of the two scenes, tweak resolution settings, and exit the demo. 40

6.3 The first puzzle players encounter, the goal is to navigate through a mirror maze. 40

6.4 The Lobby, where the player spawns upon launching the game; the benchmarks and quality settings can be controlled here. 41

6.5 This table contains detailed settings and their values for each of the quality presets. 42

6.6 Second part of the lobby, with interactable labels of all the available ray tracing settings. 43

6.7 A table of settings for each of the presets available in the playable demo. 44

6.8 A comparison of the presets and their effect on the mirror maze puzzle scene. 45

6.9 A comparison of the presets and their effect on the global illumination focused maze scene. 46

6.10 The first question of the survey. 46

6.11 The second question of the survey. 47

6.12 The third question of the survey. 47

6.13 The fourth question of the survey. 48

6.14 The fifth question of the survey. 48

6.15 The sixth question of the survey. 48

6.16 The seventh question of the survey. 49

Tables

Introduction

The goal of this project is to test the recently added ray tracing capabilities of the Unreal Engine. Ray tracing is an increasingly popular rendering method that promises accurate simulation of optical effects, based on real-world physical laws. After the launch of NVIDIA's RTX GPUs, many game developers have started adopting the ray tracing technology, providing players with new and improved visuals, based on ray traced effects, such as ray traced shadows, reflections, or global illumination.

Epic Games have been working on implementing ray tracing features into their Unreal Engine and released it to the public with the 4.22 version. In the first part of this paper, readers are introduced to the topic of ray tracing, focusing on real-time ray tracing in modern games. The introduction contains a quick summary of available ray tracing APIs, as well as basic concepts of creating a ray traced image.

To give context on how to set up a ray tracing project in Unreal Engine, we show the procedure along with system requirements. Before diving into the testing part, all the available ray traced effects are explained, along with their settings and comparison of different options. The benchmarks that are evaluated in later chapters consist of several different scenes, each aimed to demonstrate and test a different effect. The performance evaluation is supported by optimization tips and techniques available in Unreal Engine to improve performance.

The thesis also includes a ray tracing demo application which was created in Unreal Engine and contains the benchmarking environment, as well as a gameplay demo. The application can serve as a means of testing the reader's hardware, or simply to experience the ray traced effects first hand.

Chapter 2

What is Ray Tracing

Ray tracing is a graphics rendering technique used to simulate physically-based light properties, to achieve believable reflections, refractions, shadows, and indirect lighting. The first use of ray tracing dates back to 1968 when Appel [HAM19] used it to render images and has been improved massively since. Ray tracing is widely used to render high-quality 3D models, cinematic visual effects (VFX) and even animated feature films.

During the last two years, there has been growing interest in the use of real-time ray tracing, to enhance the visual quality of video games. However, the technology available to the consumers never fulfilled the requirements, to output ray traced images 30-60 times per second at a reasonable resolution. NVIDIA changed that, by releasing their RTX 20-series GPUs, which promises enough resources to tackle ray tracing in video games. Many developers have since been able to implement ray traced effects into their games, and titles such as Battlefield V and Shadow of the Tomb Raider, have become known for their ray tracing enhanced graphics.

2.1 Basics of Ray Tracing

To produce a ray traced image, we use rays to gather information about the scene's lighting. A ray consists of its origin in space, and the direction it is facing. We use ray casting, to shoot rays along their direction, to see whether they hit something. However, ray casting on its own does not suffice as a tool

render the image with shadows, reflections and other shading effects.

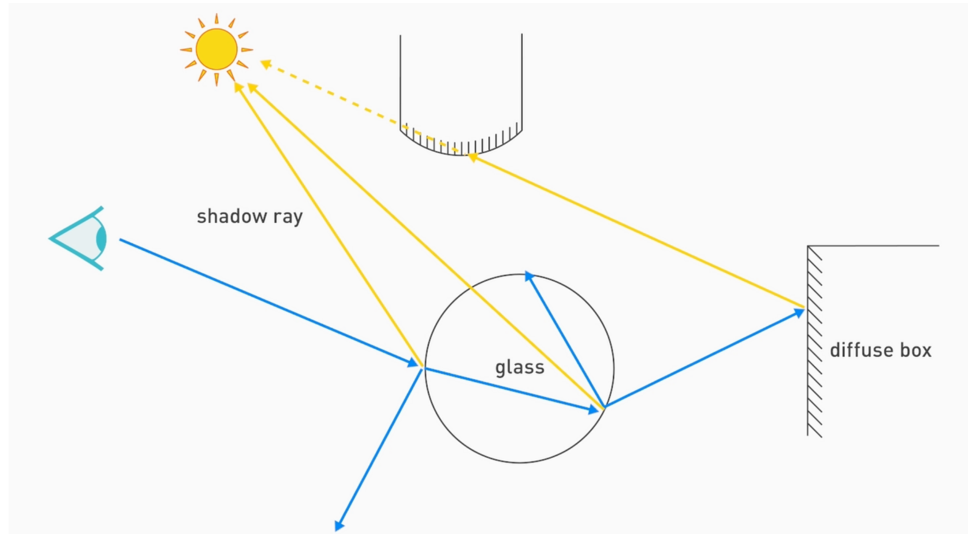

Figure 2.1: An image showing an example of how a ray could traverse a scene. Source: nvidia.com.

To create a ray-traced image, we start by shooting rays from individual pixels of the camera's image, into the scene. Whenever these primary rays hit a surface, based on what they hit, they are reflected, refracted, or shot in the direction of a light source, to determine shadows. The rules for determining the direction of the child rays are based on how lighting works in the real world. The are many different methods to determine the direction in which to shoot the secondary rays, which in turn give us the final colour for each pixel.

The rendering equation [Kaj86], shown below, is a stepping stone in the creation of a ray-traced image; it describes how each point in the scene is affected by light when viewed from a specific direction.

$$L_o(X, \hat{\omega}_o) = L_e(X, \hat{\omega}_o) + \int_{S^2} L_i(X, \hat{\omega}_o) f_X(\hat{\omega}_i, \hat{\omega}_o) \cos\theta_i \, d\hat{\omega}_i$$

Solving the equation, we get radiance L_o, i.e. the total amount of incoming and emitted light at point X when viewing it from direction $\hat{\omega}_o$. To explain the equation: L_e is the amount of emitted light from point X itself, and the integral over a hemisphere oriented around the surface normal collects all the incoming light that is visible to the hemisphere. L_i is the incoming light weighted by the cosine of the angle between the surface normal and

the incoming light direction, and also weighted by f_X, the Bi-directional Reflectance Function (BRDF), which describes how light is reflected on an opaque surface.[HAM19]

Using rays to solve the equation, we start by casting primary rays into the scene. At each intersection point, we look at how much light is emitted from the intersected object and add to the total incoming light. The incoming light is recursively collected by casting additional rays into many different directions, as described by the surface of the integral in the rendering equation. Adding the emitted and incoming light, we now have the complete light information for any given point in the scene.

The rendering equation is a complex problem, and there are many different techniques for solving it [TR12]. We briefly described how ray tracing is used together with the rendering equation to simulate physically-based lighting. More detailed discussion and other techniques are out-of-scope of this thesis.

2.2 Ray Tracing APIs

Ray tracing APIs form a layer between the GPU and the application that utilizes ray tracing, providing the developers with basic functionality, such as ray data structures, scene representation structures, and data operations. There are several APIs available, each with different advantages.

2.2.1 NVIDIA OptiX

Before the launch of RTX cards, NVIDIA created their OptiX API, which is based on their CUDA programming model. OptiX provides a high-level API for ray tracing support, as well as other GPU-accelerated calculations. In OptiX version 5.0, NVIDIA introduced an AI-accelerated denoiser. Paired with ray tracing, the denoiser allows the use of lower sample counts, resulting in faster rendering of high-quality images. OptiX is used in many professional applications, including the Arnold renderer by Autodesk. The Unity game engine uses OptiX's AI denoiser to produce high-quality lightmaps, and recently Blender has adopted OptiX to accelerate their Cycles renderer with CUDA GPUs.[OPX]

2.2.2 DirectX DXR

DirectX 12 DXR API is a ray tracing API made by Microsoft, fit mostly for the Windows 10 operating system. For a GPU to access the DXR features, it needs to be at a DirectX 12 hardware feature level 12_1 or above. This restriction limits the selection of GPUs to NVIDIA's Pascal, Turing, and Volta family chips. Despite its limitations, the DXR API is starting to get widely adopted by video game developers, who utilize it to introduce believable ray tracing effects into their games and engines. DXR can be seen in games like Battlefield V, Metro Exodus, or Call of Duty: Modern Warfare, but also in the Unity game engine, where it enables ray tracing support.

Figure 2.2: Metro Exodus uses ray traced global illumination. The difference is especially visible in indoor scenes. Source: gamestar.de.

2.2.3 Vulkan Ray Tracing Extension

In early 2020, Khronos Group released a new iteration of their ray tracing extension to the Vulkan API. The new Vulkan Ray Tracing Extension provides a cross-platform and multi-vendor framework, making it the most universal of all APIs listed in this chapter. Vulkan-based ray tracing can be seen in action in Wolfenstein: Youngblood, where it handles ray traced reflections.[RTV]

Figure 2.3: Ray tracing in Wolfenstein: Youngblood is handled by the Vulkan API. Notice the difference in reflections on the walls. Source: khronos.org.

Chapter 3

Ray Tracing in Unreal Engine

Unreal Engine 4 is one of the early real-time ray tracing adopters in the game engine industry. Ray tracing features are available to its users since version 4.22, as a beta feature set. Unreal's implementation of ray tracing is based on the DirectX 12 DXR (DirectX Ray Tracing) API, which has been integrated into the engine. Effects that are currently available include ray-traced shadows, reflections, global illumination, ambient occlusion, and translucency.

3.1 Setting Up Ray Tracing

3.1.1 System Requirements

To be able to use the ray tracing features, there are several requirements to be met. Unreal Engine version 4.22 or later has to be installed on a Windows 10 (Build 1809 or later) system, which includes DirectX 12 out of the box. DirectX 12 feature level 12_1, which is needed for DXR API, is currently supported only by a small selection of GPUs, specifically only GPUs manufactured by NVIDIA (see figure 3.1). According to NVIDIA's post about DXR support, all Turing RTX graphics cards, along with some Pascal, Volta, and Turing graphics cards support DXR.

BASIC RT EFFECTS LOW RAY COUNT		COMPLEX RT EFFECTS MULTIPLE RT EFFECTS HIGH RAY COUNT
PASCAL	**TURING**	**TURING RTX**
TITAN XP	GTX 1660 TI	TITAN RTX
TITAN X	GTX 1660	RTX 2080 Ti
GTX 1080 TI		RTX 2080
GTX 1080	**VOLTA**	RTX 2070
GTX 1070 TI	TITAN V	RTX 2060
GTX 1070		
GTX 1060 6GB		

Figure 3.1: NVIDIA's list of GPUs currently supporting DXR (excluding the newer, similiarly capable, SUPER series of RTX cards). Source: nvidia.com.

AMD has been very slow with releasing updates about their cards supporting ray tracing. Cards like the Radeon VII and 5700 XT, although they should have enough computing power to run some ray tracing enabled games, do not come with drivers which would enable the DXR support feature. However, it is rumoured, that AMD could release ray tracing enabled cards, based on their RDNA2 7nm+ architecture, in 2020/21. [AMD]

3.1.2 Configuring a New Project

With a newly created or migrated UE 4.22 project, to enable ray tracing, we need to open Project Settings, and under Platforms > Windows, the Default RHI should be set to DirectX 12. Then under Engine > Rendering tick the ray tracing attribute. If the Support Compute Skincache option has not been enabled for the current project, a prompt pops up, asking us to enable this option, which is required for ray tracing to work. Now the engine needs to be restarted. At this point, before relaunching the engine, make sure that the editor launches in DirectX 12 mode, by creating a shortcut for the Unreal Engine Editor, and setting a -dx12 parameter in the target field (see figure 3.2).

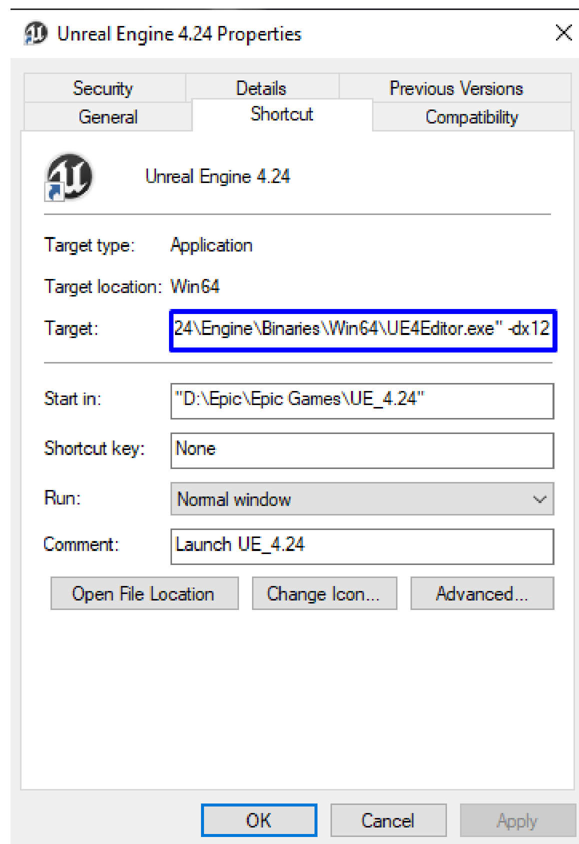

Figure 3.2: Properly set DirectX 12 launch parameter for the target .exe.

3.2 Available Effects and their Settings

As mentioned in the previous chapter, Unreal Engine currently supports several ray tracing effects. These include ray traced shadows, reflections, global illumination, ambient occlusion, and translucency. To give control over these effects, UE4 exposes the controls via its Post Process Volume object. To set up the Post Process Volume for a scene is as simple as dragging it into the current level from the class browser. The Post Process Volume can then be resized to influence a specified area or made global by setting the Infinite Extent (Unbound) parameter to true.

3.2.1 Shadows

Contrary to other ray tracing effects in UE4, light shadows are set separately, per light source. Each light has a Cast ray tracing Shadows option in its Details panel, which turns the ray traced shadows on/off. Samples Per Pixel can also be adjusted on each light separately, to control the quality of the shadow. See figure 3.3 for comparison between a default Shadow Map and ray traced shadow.

Figure 3.3: Comparison between traditional hard dynamic shadows and ray traced soft shadows. Ray traced shadows provide realistic smooth transition between lit and shadow areas.

To control the sharpness of the shadow for Soft Area Shadows, we use the Source Angle (Radius) parameter of the light. Additionally, much like in real life, the greater the distance between the shadow caster and the shadow, the softer the shadow becomes.

3.2.2 Reflections

Ray traced reflections are one of the more noticeable ray tracing effects, as they allow reflection of both dynamic objects and objects that are out of the camera's view frustum. To engage ray traced reflections, we switch the Type of reflections in the Post Process Volume to ray tracing.

Figure 3.4: The screen space reflection on the right side of this image lacks the ability to reflect objects outside of the camera's view frustum.

There are several modifiable parameters of the ray traced reflection (see figure 3.7), which allow for great control over the performance cost of the effect. Shadows selection allows choosing between Hard Shadows, Area Shadows and shadows completely Disabled, see figure 3.5 for comparison.

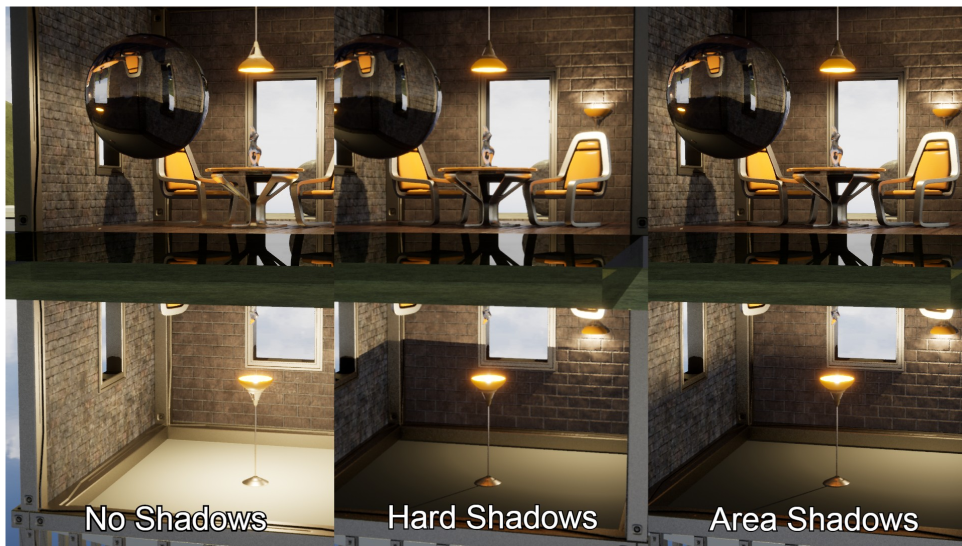

Figure 3.5: Comparison of ray traced reflections having: no shadows, hard shadows, and area shadows.

Max Roughness sets the maximum roughness for materials to receive ray traced reflections, Screen Space Reflection methods are used otherwise. Max

Bounces limits the number of times a single ray can bounce off of surfaces enabled for ray traced reflections by the Max Roughness setting. The number of bounces is especially important when dealing with multiple highly reflective objects placed close to each other, as an insufficient number of bounces can result in black gaps instead of reflected objects (see figure 3.6).

Figure 3.6: Small number of bounces between reflective objects can result in them appearing to be black.

To include translucent objects in the reflections, we can toggle the Include Translucent Objects variable. Samples Per Pixel control, how many times a ray is cast for each pixel. Higher sample count results in a clearer image, at the expense of performance.

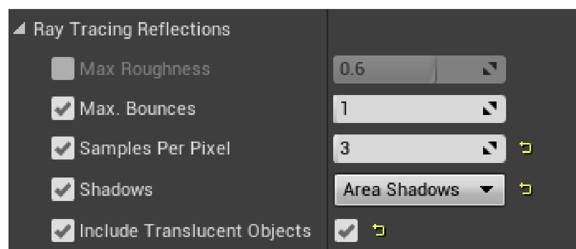

Figure 3.7: The settings panel for controlling ray traced reflections.

3.2.3 Translucency

Ray traced translucency is mainly used to render objects such as glass or water with accurate reflections and refractions of incoming light. In UE4 it is enabled in the Post Process Volume and has several modifiable parameters (see figure 3.8).

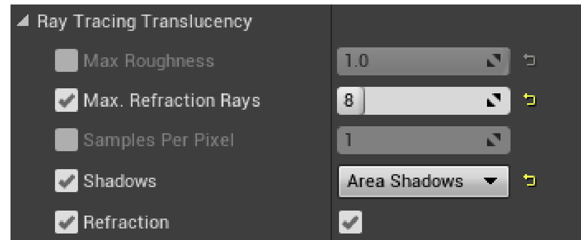

Figure 3.8: The settings panel for controlling ray traced translucency.

Similarly to ray traced reflections, we can choose the reflected shadow type, maximum roughness cutoff, and samples per pixel. When modifying the parameters of ray traced translucency, we need to pay attention to the number of bounces we allow each ray to make. When the Refraction variable is set to true, the translucent object starts to refract the incoming light. For the ray to exit the refracting object, we need at least three bounces in total; otherwise, the object appears black (see figure 3.9). The number of bounces for the refraction ray is set via the Max Refraction Rays parameter.

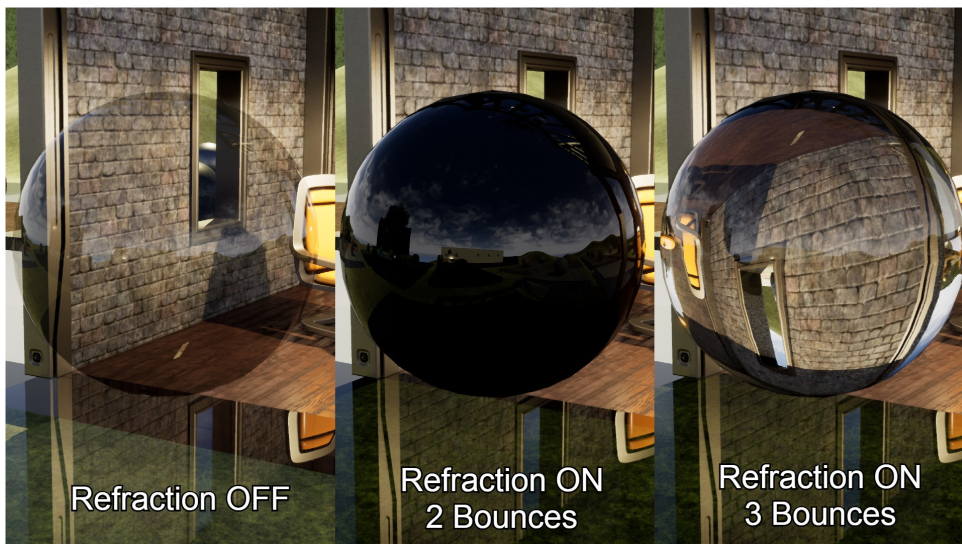

Figure 3.9: Various refraction settings; Index of Refraction for this material is set to 1.

To describe how the translucent object refracts light, we use the Index of Refraction (IOR), which in UE4 corresponds to the specular component of the object's material. The refraction index describes how fast light travels in a given material, which is used to determine the angle of the refracted ray entering the material. See figure 3.10 for comparison of several Index of Refraction values.

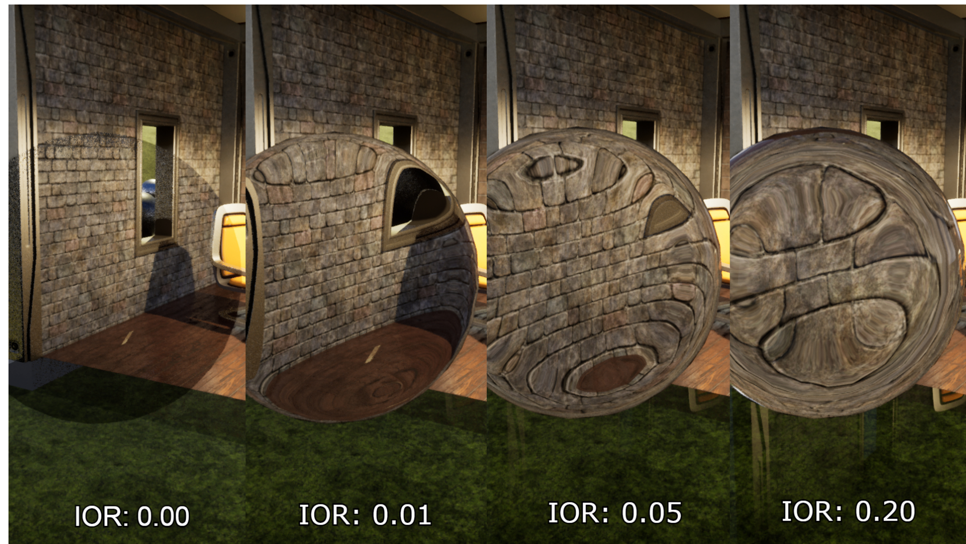

Figure 3.10: A comparison of several settings for the material's refraction index.

3.2.4 Ambient Occlusion

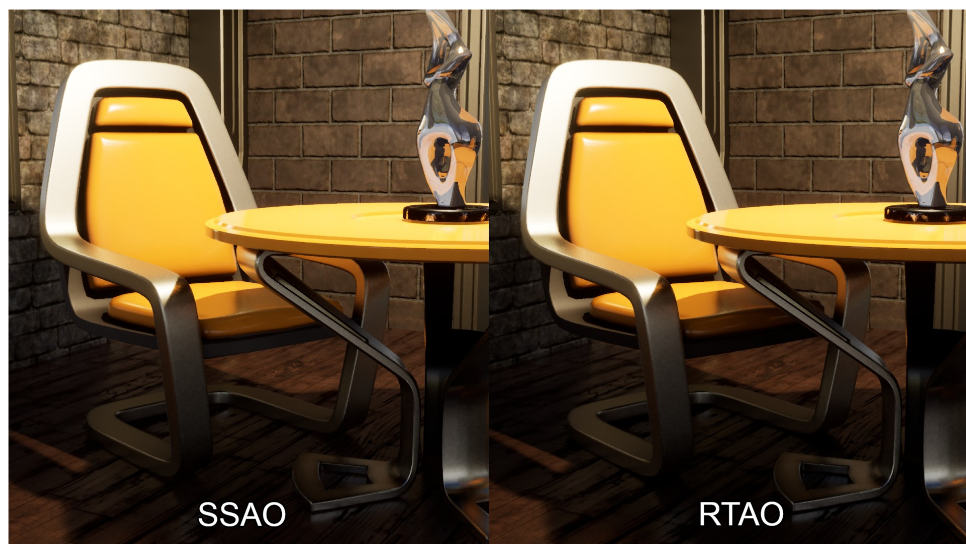

Figure 3.11: Ray traced ambient occlusion is more accurate and pronounced, adds depth to the scene.

Figure 3.12: The difference between ray traced and standard ambient occlusion is more visible when we hide all but the ambient occlusion layer.

Ray traced ambient occlusion achieves a more natural-looking scene with physically correct ambient occlusion. The behaviour of ray traced ambient occlusion can be controlled with the Intensity and Radius parameters in the Post Process Volume, where the Samples per Pixel count can be adjusted as well (see figure 3.13). Radius determines the area around the occluded object that is taken into consideration when deciding the amount of ambient light being blocked.

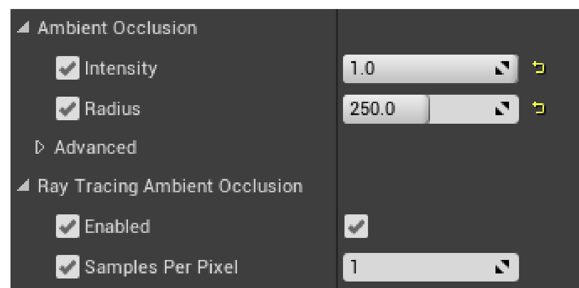

Figure 3.13: The settings panel for controlling ray traced ambient occlusion.

3.2.5 Global Illumination

Ray traced global illumination calculates how the light bounces off of objects and into the scene in real-time, to form indirect lighting. In UE4 (version 4.24 and later), there are currently two methods for calculating ray traced global

illumination. While the Brute Force method is based on the Path Tracing algorithm and is computationally expensive, the new Final Gather method has been developed to give similar results, while reducing the performance hit. The latter of the two is currently limited to a single bounce, and is, in some cases, susceptible to ghosting and artefacts. However, it gives back the much-needed performance at minor costs in image quality (see figure 3.14 for comparison of the two methods).

Figure 3.14: This picture shows that the Final Gather method is susceptible to artifacts. However, these are not as prominent in well lit areas and scenarios. Both Final Gather and Brute Force use a single bounce and 8 samples in this image.

The Post Process Volume allows us to set what global illumination method we want to use, the maximum number of bounces for each ray, and how many ray samples to use for each pixel. In UE4 version 4.24, it is recommended by the developers to use Max Bounces of 1 and Samples per Pixel of 8, to get the best results when using Final Gather.

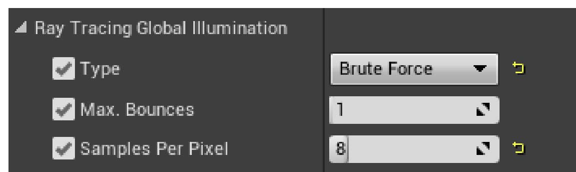

Figure 3.15: The settings panel for controlling ray traced global illumination.

Chapter 4

Benchmarks

In this chapter, we look at several different benchmarks and scenarios of ray tracing use in Unreal Engine. Each scenario was constructed to target and test a different set of ray traced effects. The results can serve as an indicator of what levels of performance can be expected, when developing a game in Unreal Engine, using real-time ray tracing. During this chapter we test several quality presets (OFF, LOW, MEDIUM, HIGH), which are described in detail in Chapter 6. The presets were created to allow simple change of multiple settings at once, to mimic graphics quality settings in video games. Please note, that the following chapter's purpose is not to give tips on how to optimize scenes that use ray traced effects; for performance optimization tips and techniques, see Chapter 5.

4.1 Testing Environment

The tests aim to show what the performance of ray tracing in Unreal Engine looks like and try to answer whether an older card, not designed with ray tracing in mind, can be used to achieve acceptable results. The two cards tested were the NVIDIA GTX 1080 and the NVIDIA RTX 2070 SUPER, both running at stock clock speeds. The rest of the hardware components stayed the same for both GPUs, and are listed in figure 4.1.

CPU:	AMD Ryzen 2700X @4.0Ghz
Motherboard:	ASUS ROG Strix X470-F Gaming
RAM:	16 GB DDR4, 3200Mhz
OS:	Win 10, Build 17763
GPU1:	NVIDIA RTX 2070 Super
GPU2:	NVIDIA GTX 1080

Figure 4.1: Test bench hardware specifications.

When it comes to testing GPU performance, one of the important factors is the driver; especially so for ray tracing, where data structures and other driver-related software play a significant role. Both GPUs were tested with the latest Game Ready driver from NVIDIA - 441.87 (released January 6, 2020).

4.2 Benchmark 1 - Room

Figure 4.2: Benchmark 1 scene using the HIGH preset.

The first benchmark, mostly based on the freely available Starter Map from UE4 example project, takes place in an indoor scene with multiple light sources, translucent objects and semi-reflective surfaces (see figure 4.2). Generally speaking, indoor scenes are more demanding when using ray tracing; this is

because of how light bounces off of walls back into the scene, creating more rays with each reflection/refraction.

Figure 4.3: Turning all the available ray traced effects on decreases performance severely, even on the RTX 2070S.

Figure 4.3 compares what the scene looks like with ray tracing turned completely OFF to what it looks like set to the HIGH preset. We can see that the ray traced reflections on the metal parts of the table as well as on the translucent statue look much more realistic. The ray traced shadows are also much more accurate and add depth to the scene, which is essential when rendering indoor environments. However, looking at the performance of the HIGH preset (see figure 4.4), we are looking at framerates of around 7 FPS, which is very low for any user interaction, and would require a more powerful RTX GPU.

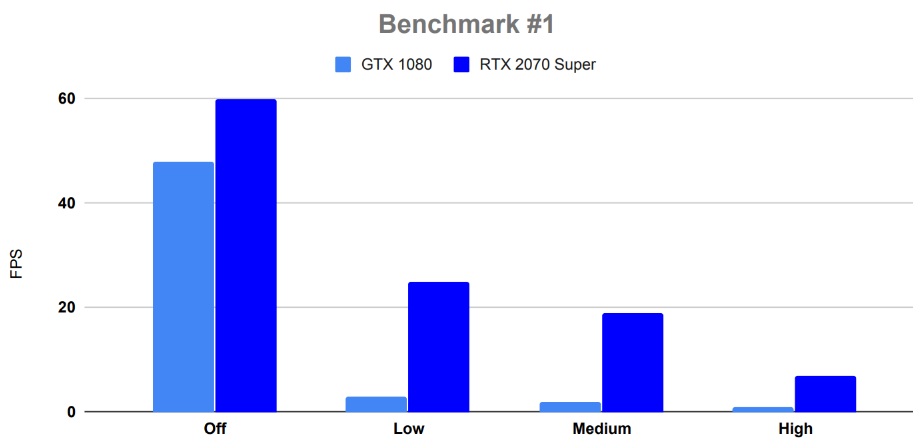

Figure 4.4: Benchmark 1 test results for the four quality presets and both GPUs.

A more sensible approach to enhancing this scene with ray tracing would be to dial down the number of bounces for the reflections and samples per each pixel. This option is represented by the MEDIUM preset, and is displayed in figure 4.5. Comparing the MEDIUM and HIGH presets, we get a slight, but

an almost unnoticeable decrease in visual quality. However, the framerate has improved drastically, and the scene now runs at 19 FPS, using the MEDIUM preset.

Figure 4.5: Comparison between the MEDIUM and HIGH presets of benchmark 1.

To increase the framerate even further, we can disable some of the less noticeable effects; the LOW preset does not utilize ray traced global illumination and does not use refractions and shadows for translucent objects. While the framerate increase from 19 to 25 FPS is noticeable, we lose a lot of visual fidelity and detail (see figure 4.6). Notice how the shadows and some parts of the metallic objects are much darker, because of the low number of bounces and lack of ray traced global illumination.

Figure 4.6: Comparison between the LOW and HIGH presets of benchmark 1.

Comparing the LOW, MEDIUM, and HIGH preset images, the HIGH preset makes sense only if we are targeting users with high-end GPUs, and are focused on storytelling rather than on fast-paced action gameplay. The LOW and MEDIUM presets, on the other hand, can be used even on mid-tier GPUs, such as the RTX 2070. To achieve higher framerates, we would need to optimize the scene better, but this is also likely to improve with future updates of the engine and GPU drivers.

4.3 Benchmark 2 - Balls

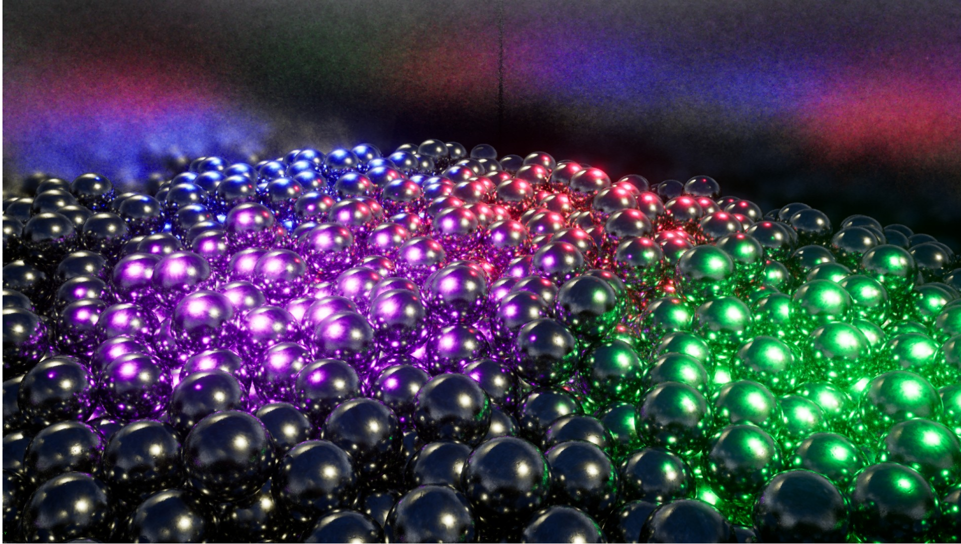

Figure 4.7: Benchmark 2 using the HIGH preset.

While the first benchmark took advantage of all the ray tracing effects available, the second benchmark is heavily focused on reflections. It consists of roughly two thousand shiny metallic balls placed in a metallic cube container. The balls are lit by the skylight (the sky), directional light (the sun) and four additional coloured lights (see figure 4.7), to give off nice reflections.

Figure 4.8: Comparison between the OFF and HIGH presets of benchmark 2.

Looking at the comparison between OFF and HIGH presets, we see that the ray traced reflections make the steel material appear realistic, giving it the shininess metallic materials need. Looking at the performance (see figure 4.9), there is once again a massive drop in performance, although, at 10 FPS using the HIGH preset, it is slightly higher than the first benchmark.

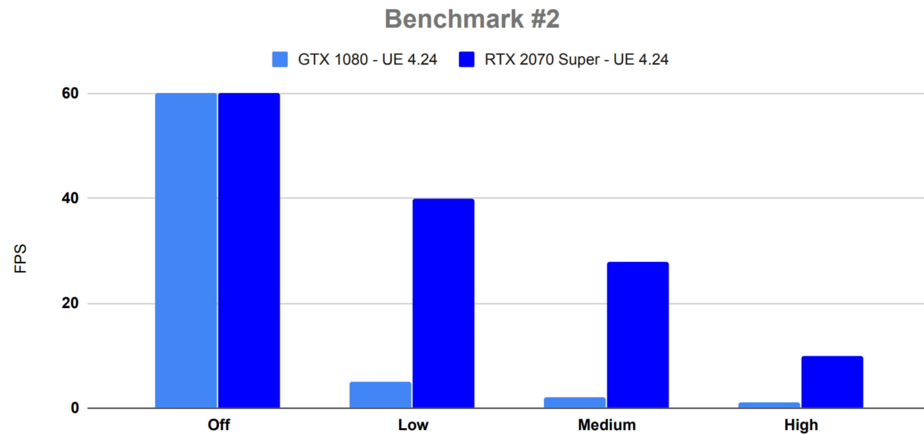

Figure 4.9: Benchmark 2 test results for the four quality presets and both GPUs.

However, a problem arises when we try to use the MEDIUM, and the LOW presets. For this type of scene, where we have lots of reflective objects, the number of max bounces is very crucial. As we discussed earlier, the lower the number of max bounces, the darker the objects appear. Looking at figure 4.10, we see that by turning the quality preset to MEDIUM and LOW, the walls become much darker, and the shininess of the balls decreases.

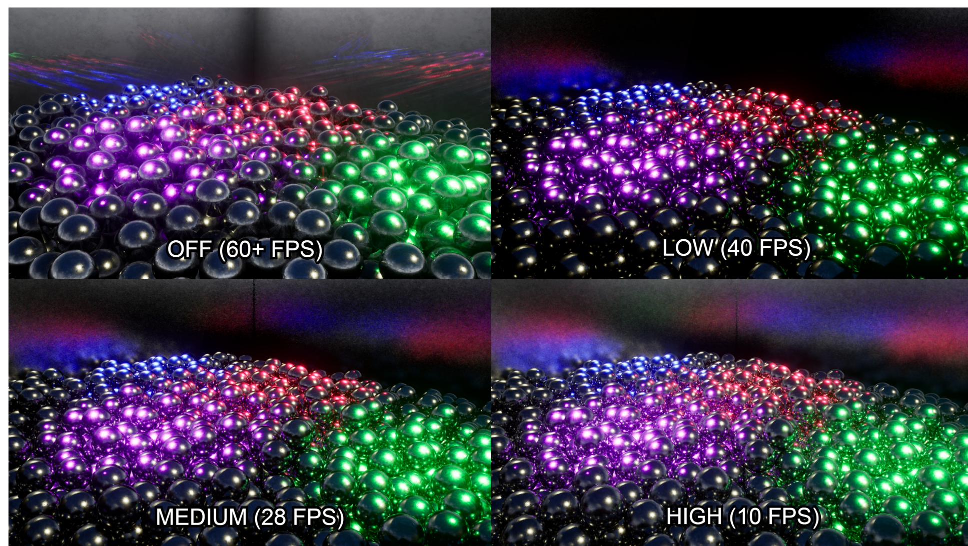

Figure 4.10: Comparison between all the presets of benchmark 2.

To brighten up the scene and try to trick the viewer's eye into thinking it sees a high number of bounces, we can use the Reflection Captures as the last bounce of the reflection. This method is further discussed, along with an

example, in Chapter 5.

4.4 Benchmark 3 - Shore

Figure 4.11: Benchmark 3 using the HIGH preset.

So far, we have tested how Unreal Engine's implementation of ray tracing performs in an indoor scene, and in a synthetic environment focused on testing reflections. In the third benchmark, we look at a complex outdoor scene, with detailed geometry. This benchmark aims to represent the typical environment of a modern AAA title and includes high-poly models of trees, vegetation, cliffs and rock structures. The scene also contains a body of water to add a reflective element.

Figure 4.12: Comparison between the OFF and HIGH presets of benchmark 3.

Comparing the OFF and HIGH presets (see figure 4.12), at first glance,

the differences may appear very subtle. However, upon closer inspection, we see the detail of the ray traced shadows, the reflection in the water, and the natural-looking colour of the leaves. The ray traced translucency is able to display more of the leaves correctly, and the ray traced shadows give the tree crown more depth. When we look at the performance (see figure 4.13), we notice that the overall performance is, yet again, better than the previous benchmarks. The performance is more consistent across the presets than the previous ones, because the scene does not contain many reflective surfaces, and since it is outdoors, many rays end up hitting the sky sphere.

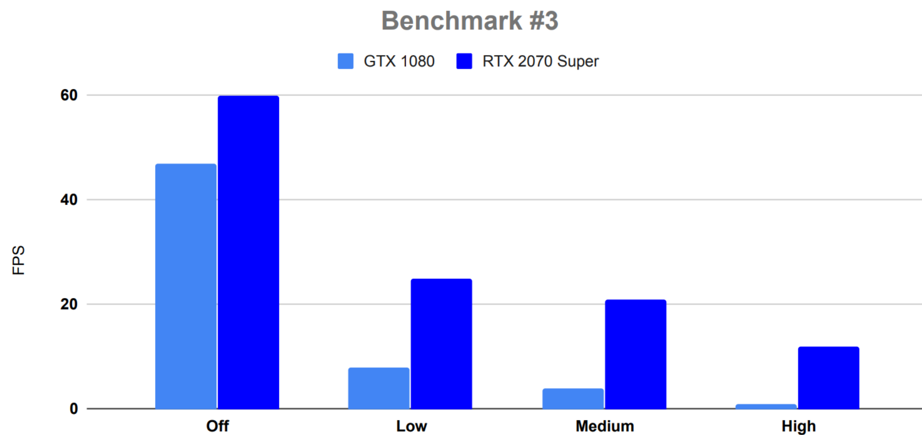

Figure 4.13: Benchmark 3 test results for the four quality presets and both GPUs.

That said, dropping the quality settings down (see figure 4.14), we see that the lack of translucency shadows on the LOW preset makes this preset unusable, due to the way the leaves are displayed. The MEDIUM preset, although it has a lower number of samples, looks very similar to the HIGH preset, while providing a boost in performance. Overall, because the scene does not have very reflective surfaces, nor does it have dark spots, it allows the number of samples per pixel to be dropped while retaining the visual fidelity.

Figure 4.14: Comparison between all the presets of benchmark 3.

However, if we were to use this scene in a game, where the player would not be able to spend much time exploring the details, such as individual leaves of the trees, the OFF preset without any ray tracing would probably be the best option. The difference in performance between turning the ray tracing on and off for this particular scene is far too great to justify the minimal difference in visual quality.

4.5 Non-RTX Performance

As part of the testing process, all three benchmarks were tested on the NVIDIA GTX 1080, to see, what kind of framerates it would deliver. Since the GTX 1080 supports DXR, running the tests required no additional setup, and ray tracing worked instantly. However, looking at the results (see figures 4.4, 4.9, 4.13) we see, that, at least with current ray tracing algorithms and data structures, the older Pascal-based GPU is not very capable. The demo was unplayable, and we encountered frequent freezes.

The results that we observed were to be expected, as the GTX 1080 has a peak throughput of just below 1 Gigarays/s, while the RTX 2070 Super is rated for peak throughput of 7 Gigarays/s, thanks to its RT (Ray Tracing) cores. Now, the metric of Gigarays per second can not be used to precisely calculate the difference in performance we should expect to receive. NVIDIA mentions these numbers as rough estimates for when the card is operating in ideal conditions processing synthetic datasets, that usually do not represent typical workloads, such as games or rendering.

In our tests though, we see that the RTX 2070 Super performed anywhere between 3-8 times better than the GTX 1080. The increase in performance goes to show, that while the ray tracing capabilities of the latest RTX Series cards are better by a significant margin compared to older-gen GPUs, it is still not enough for RayTraced-only real-time rendering.

4.6 Scene Scalability Testing

In benchmark no. 3, where we tested ray tracing performance in an open environment scene, models of the trees were intentionally chosen to contain a high number of polygons. This was done to simulate crowded scenes of modern video game titles. In this chapter, we look closely at how ray tracing performs when scaling the number of objects in our scene, compared to rasterization.

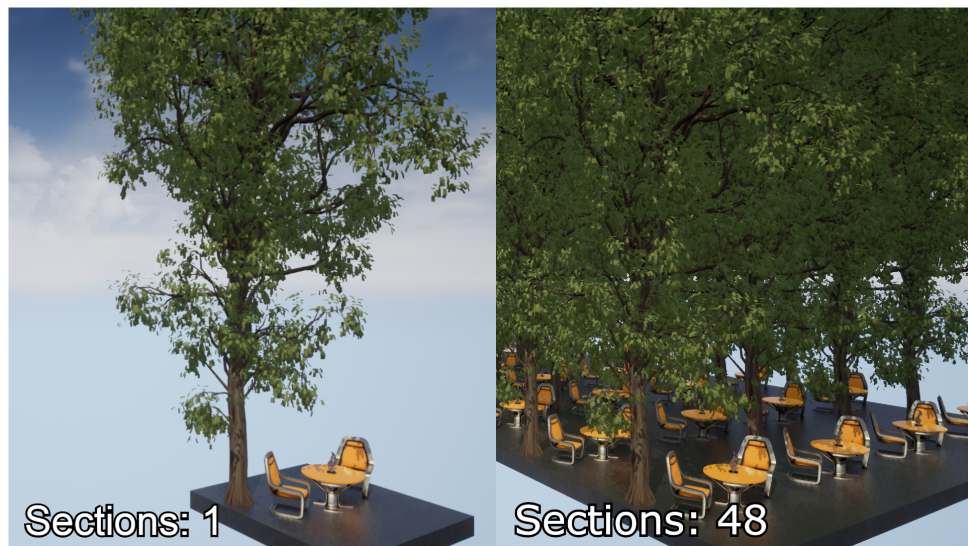

Figure 4.15: This figure shows a the scene of the first test, with 1 and 48 sections on each side.

For our first test, we took the high-poly tree model, with some other translucent and reflective assets, and formed a section (see fig. 4.15). During the first test, we took measurements with an increasingly higher number of section. Based on the results, we can plot a performance graph, with GPU frame times on the Y-axis and scene triangle count on the X-axis.

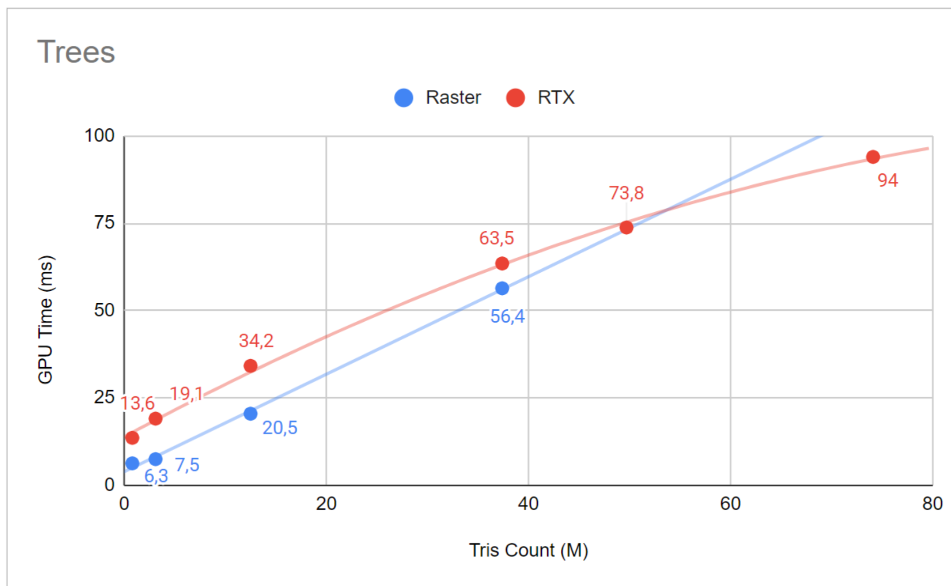

Figure 4.16: Final graph showing the ray tracing as well as rasterization data.

Looking at the graph in figure 4.16, we see that, as expected, the ray tracing performance was worse overall, especially with lower triangle counts; however, ray tracing seems to scale better than rasterization, with close to logarithmic complexity. The theoretical logarithmic complexity of ray tracing is based on its search algorithms in scene acceleration trees, compared with linear complexity of rasterization. Note that rasterization can be further optimized to reduce the complexity by using levels of detail (LODs) and occlusion culling in demanding scenes.

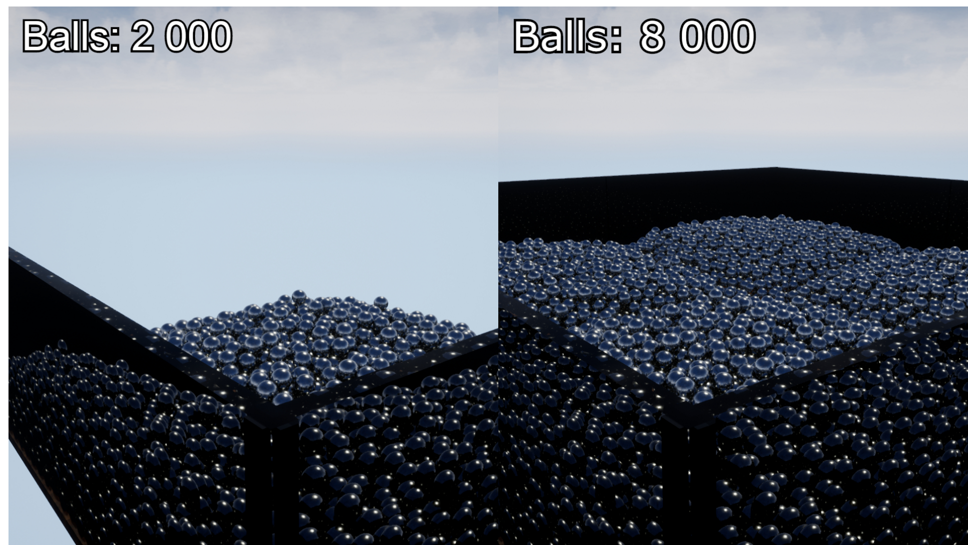

Figure 4.17: A figure showing what the ball test scene looked like.

For our second test of ray tracing scalability, we used the shiny balls from benchmark no. 2. During this test, we dropped roughly 2 000, 4 000, 6 000, and finally 8 000 balls into a container using physical simulation (see figure 4.17), and measured GPU draw times, after the balls settled. The graph in figure 4.18 plots the results.

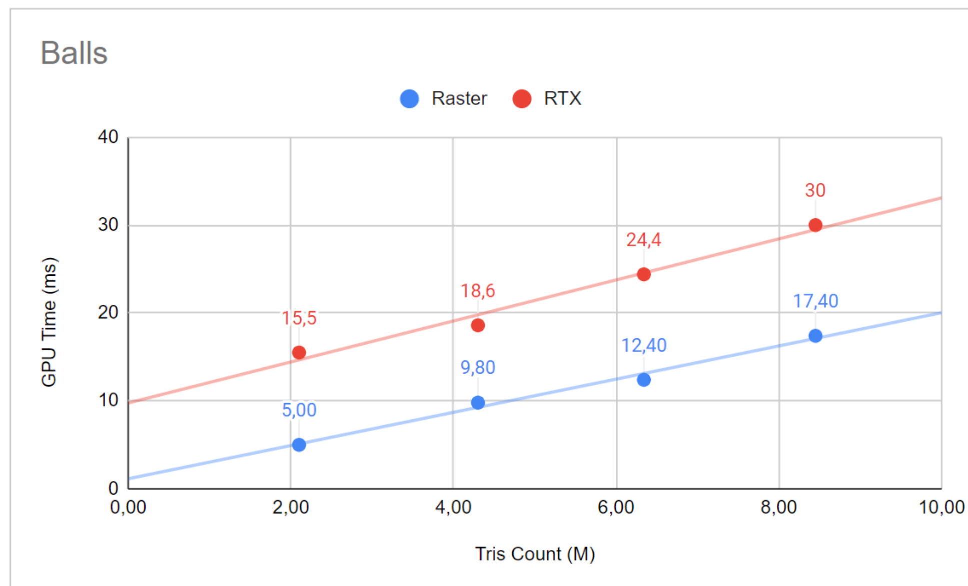

Figure 4.18: Final graph for the balls test, both rendering techniques seem to have linear complexity.

The graph has the same layout as the one in the first test - GPU draw times on Y-axis, triangle count on the X-axis. However, the results seem quite different. This time, both rendering methods seem to have a linear complexity. This can be explained by the fact that the scene consists purely of reflective surfaces. Real-time ray traced reflections are a costly effect, and we can see from the measured values that they do not scale very well. What was more interesting, however, was the fact that the draw times were fairly consistent even when the balls were affected by physics, falling down into the container.

	Balls 1x	Balls 2x	Balls 3x	Balls 4x	Trees 1x	Trees 4x	Trees 16x	Trees 48x	Trees 64x	Trees 96x
Raster										
GPU (ms)	5	9.8	12.4	17.4	6.3	7.5	20.5	56.4	driver crash	driver crash
Total (ms)	20	35	47	62	8.3	8.3	20.7	56.5	driver crash	driver crash
RayTraced										
GPU (ms)	15.5	18.6	24.4	30	13.6	19.1	34.2	63.5	73.8	94
RTGI	0.6	0.7	0.9	1.2	3.3	6.9	10	16.2	18.1	19.3
RTRefl	4.8	5.2	6.5	7.7	3.8	3.9	4.4	5	5.8	6.3
Total (ms)	20	35	48	63	14	19.2	34.5	63.7	76	96.5
Tris:	2.1M	4.3M	6.34M	8.45M	0.8M	3.1M	12.5M	37.4M	49.7	74.1M

Figure 4.19: This is the final table containing all the scalability testing data.

Finally, figure 4.19 shows a table of all measured values, including triangle counts, GPU draw times, as well as separate reflection and global illumination call times. Note that during the testing of the tree section scene, rasterization failed to display the two highest triangle counts, with Unreal Engine crashing each time we tried to display the scene. The fact that ray tracing was able to display more triangles than rasterization only proves that it scales better with scene complexity, at least when it comes to the number of triangles.

4.7 Unreal Engine Version Comparison

Since the addition of ray tracing into Unreal Engine has been made quite recently (version 4.22 was the first to ship with ray tracing in Spring 2019), Epic Games has put a vast amount of effort into optimizing the newly added feature. They have also been adding new features such as the Final Gather algorithm for better optimized ray traced global illumination. As part of the testing, we have tested the second benchmark with both the 4.23 and the latest 4.24 version of Unreal Engine, to see, whether there are measurable improvements in performance.

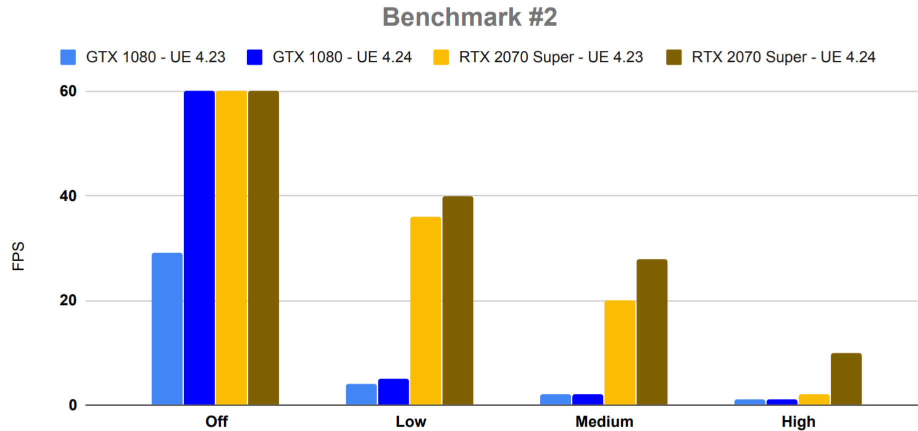

Figure 4.20: Comparison between the 4.23 and the 4.24 versions of Unreal Engine, using benchmark 2.

Figure 4.20 presents the results of the testing. We can see that Unreal Engine version 4.24 is performing consistently better than the older 4.23. (Please note that the MEDIUM preset in 4.23 uses the Brute Force global illumination method - single bounce and sample, as the Final Gather was added in the 4.24 version of Unreal Engine). Some of the improvements in framerates are quite significant. One of the issues with the 4.22 and 4.23 versions was that even though all the ray tracing effects were turned off, the fact that ray tracing was enabled for the project resulted in some ray tracing draw calls. These draw calls caused the performance, especially on the GTX 1080, to be significantly reduced.

Chapter 5

Performance Optimization

In this chapter, we talk about optimizing scenes which use ray traced effects. When building a ray traced scene, it is essential to take into consideration the amount of detail the player is likely to notice and balance that information with the amount of performance we want to sacrifice for the added visual quality. Later in the chapter, we also mention some additional settings that are available via the Unreal Engine console and allow developers to fine-tune the detail vs performance balance.

5.1 Deciding Which Ray Tracing Effects to Utilize

Ray traced effects are computationally demanding; we explained that in Chapter 4, where we focus on testing the performance of ray tracing in Unreal Engine. When we look at modern games that utilize ray tracing to enhance their visual quality, we see that even AAA titles use only a couple, often a single, ray traced effects. To give some examples: the latest Call of Duty: Modern Warfare, uses ray tracing only for local light shadows. Battlefield V uses ray tracing to calculate the reflections of some shiny surfaces, and Metro: Exodus, uses ray traced global illumination.

The reason we do not see many ray traced effects coupled together, is mostly performance. The penalty for using ray tracing, may it only be for reflections or shadows alone, is often as drastic as 20-30% of overall performance, even when using the latest RTX GPUs. Framerate drops that big, are hardly justifiable, given modern methods of approximating results of ray traced algorithms are close to reality.

Figure 5.1: Comparison between the OFF and HIGH presets of benchmark 3.

If we look at the comparison between the OFF and HIGH presets for the third benchmark (see figure 5.1), the difference does not become apparent, until we inspect the images next to each other. The scene simply is not fit for ray tracing, as its design does not benefit from ray tracing advantages.

5.2 Denoising vs Samples per Pixel

Denoising relies on dedicated algorithms to clear up the image to reduce the number of sample per pixel needed for a clear image. In Unreal Engine, denoising can be turned on individually for each ray traced effect, by using the corresponding command in the editor console. Looking at figure 5.2, we notice that the legs of the chair and table look less distorted and less noisy when using a lower number of samples per pixel with the denoiser. The denoised option also performs much better.

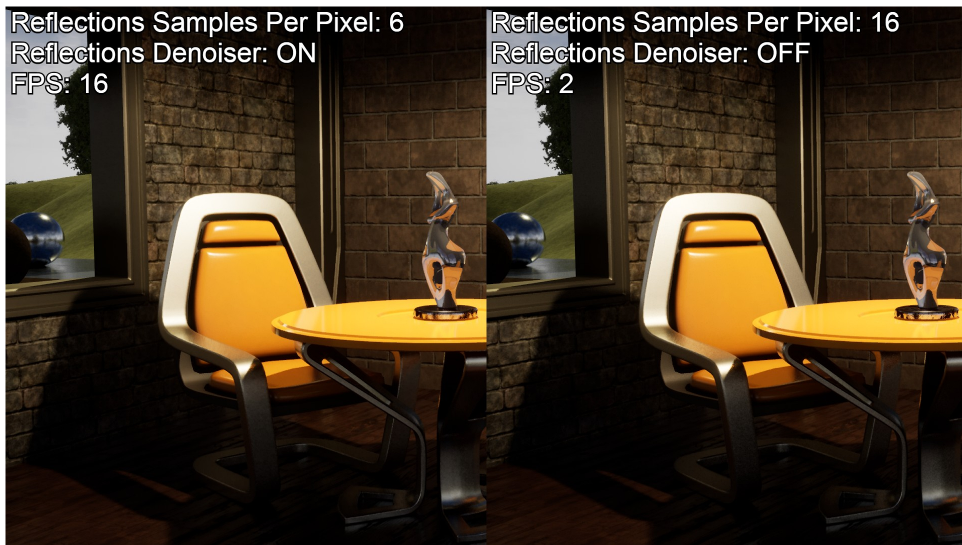

Figure 5.2: Comparison between a denoised image and an image with higher sample count.

That said, using the denoiser does not always guarantee the best results. Figure 5.3 shows one such example, where the denoiser generates visual artefacts when applied on the ray traced global illumination. The performance does still improve when using the denoiser, but might not be worth the degradation in visual quality.

Figure 5.3: Visual artefacts introduced by the denoiser.

5.3 Other Ray Tracing Settings

Aside from the settings available in the Post Process Volume and each Light actor, Unreal Engine allows additional adjustment of the ray tracing settings via its console commands. [UER]

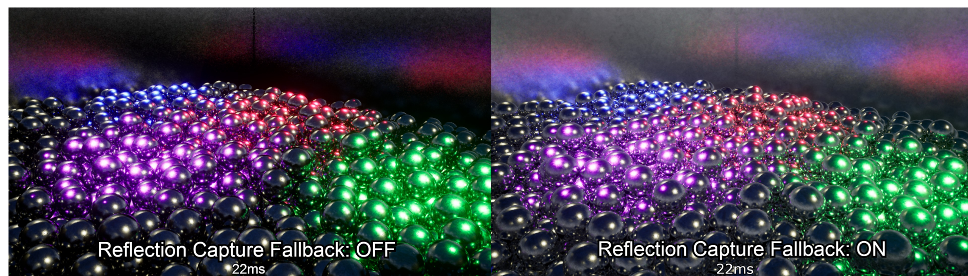

Figure 5.4: Demonstration of the Reflection Capture Fallback reflection option.

The Reflection Capture Fallback option sets the last bounce of each reflection ray to be evaluated from the Reflection Capture Spheres, which results in an image that retains most of the qualities of the ray traced reflections, but solves the problem of dark objects (see figure 5.4). This option could be used to reduce the overall number of reflection bounces, to increase performance, as the fallback has little to no performance cost.

Another way to increase performance is to adjust the distance each ray can travel before being dismissed. The MaxRayDistance can be set for ray traced skylights, reflections, translucency and global illumination, using the editor console. It can help in scenes where we do not require the ray traced effects to be affected by the whole surroundings or can get away with a slightly less precise result.

Unreal Engine also provides a useful tool to optimize materials for use with ray tracing. Their blueprint tool for creating materials features a RayTracingQualitySwitchReplace node (see figure 5.5), which outputs one of two predefined inputs, based on whether ray tracing for that material is enabled or disabled.

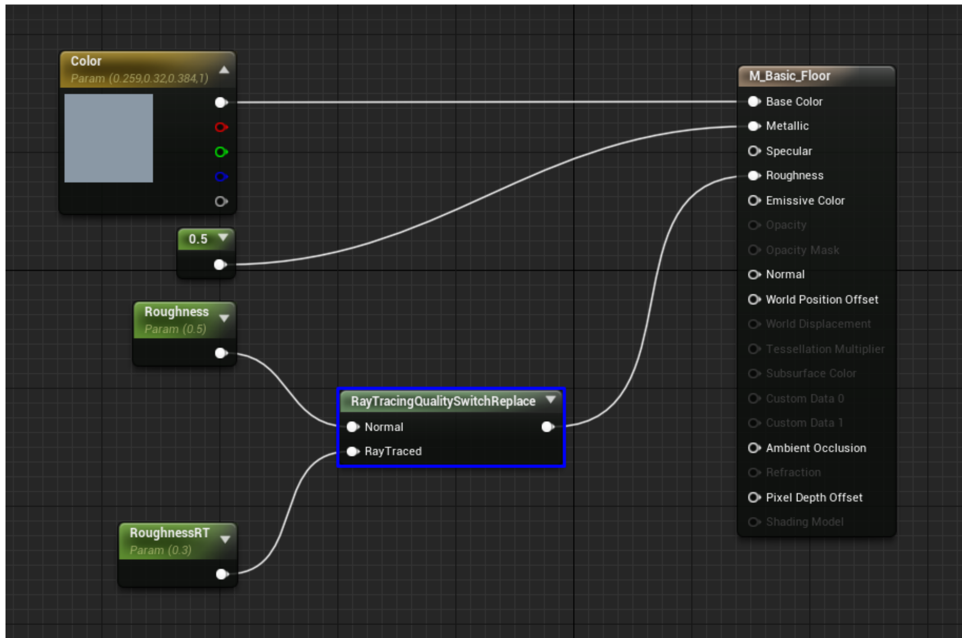

Figure 5.5: The RayTracingQualitySwitchReplace node in action.

Chapter 6

Unreal Engine Ray Tracing Demo

This chapter covers the functionality and controls of the ray tracing demo application, which can be downloaded from the project's website: `http://ueraytracing.dcgi.fel.cvut.cz/`. The purpose of this application is for the reader to be able to test the ray tracing capabilities of their system and to give perspective on how the ray traced effects look and behave when navigating the scene.

6.1 Controls

Movement:	W,A,S,D
Jump:	Spacebar
Camera Movement:	Mouse
Interact:	E
Quality Settings:	U (off, only benchmark scene)
	I (low)
	O (medium)
	P (high)
	L (custom, only benchmark scene)
Menu	Esc

Figure 6.1: Table of controls and their key bindings.

Figure 6.1 shows the control bindings for the application, which are similar to any modern First Person video game. After launching the demo, users are greeted with a menu, allowing the change of resolution, as well as giving them the option of two play modes (see figure 6.2).

Figure 6.2: The menu, giving the options to play one of the two scenes, tweak resolution settings, and exit the demo.

The first button - Play - starts an FPS-maze style game, where the player needs to navigate three areas to get to the finish. While on the quest of completing the three puzzles, users can switch between three quality settings (Low, Medium, High), which affect visuals and performance. For more information about the quality settings for this portion of the demo, see chapter 6.2 and 6.3.

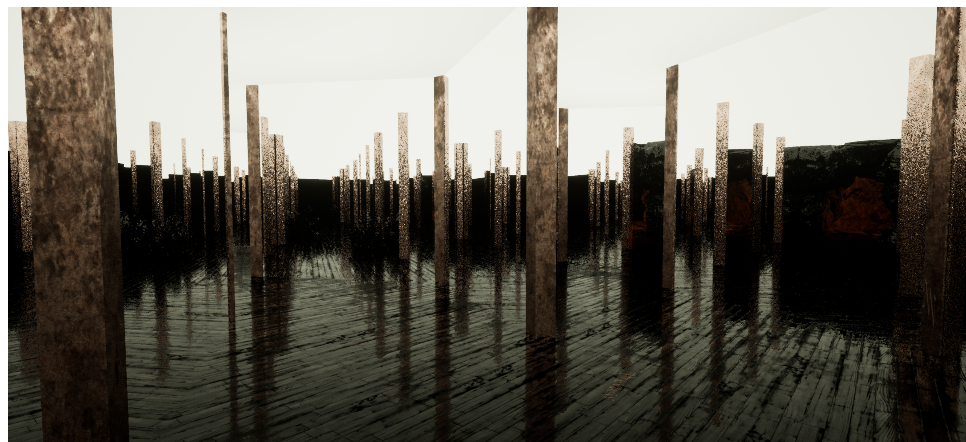

Figure 6.3: The first puzzle players encounter, the goal is to navigate through a mirror maze.

The second button - Benchmark - loads up a scene which we used as an environment for most of the testing conducted in chapter 4. Here the users can toggle between four different quality presets using either the assigned keys or by interacting with their respective buttons in the Lobby.

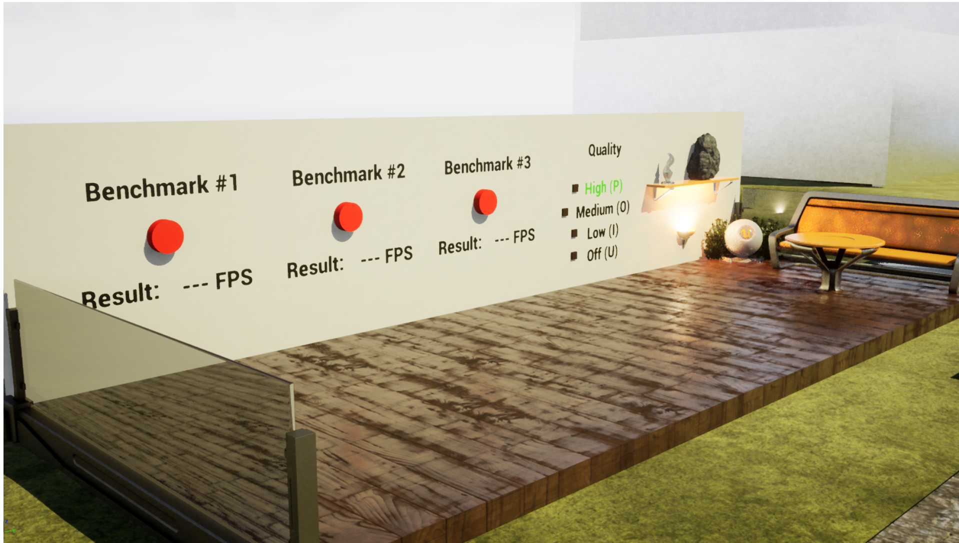

Figure 6.4: The Lobby, where the player spawns upon launching the game; the benchmarks and quality settings can be controlled here.

The Lobby (see figure 6.4) also contains buttons to initiate each of the three benchmarks described in Chapter 4, which can launched using the 'E' key. Every time a benchmark is finished, the average FPS recorded is displayed under the button which was used to start that benchmark. Aside from the Lobby, the scene consists of two locations - the Room and the Shore, which can be discovered and explored by the player freely.

6.2 Quality Settings

The demo app features five quality presets, which alter the ray tracing quality settings. Figure 6.5 lists every setting with its assigned value, that is influenced by the presets. Please note, that, as described in Chapter 5, Unreal Engine allows for additional tweaking of the ray tracing settings such as ray distance, denoising settings, and reflection capture fallback. At the moment, these settings are not influenced by the quality presets, although this may change in future updates of the ray tracing demo application.

		Off	Low	Medium	High	Custom
Ambient Occlusion:						
	Enabled	N	N	Y	Y	?
	Samples Per Pixel	-	-	1	3	?
Global Illumination:						
	Enabled (Type)	N	N	Final Gather	Brute Force	?
	Max. Bounces	-	-	1	3	?
	Samples Per Pixel	-	-	8	3	?
Reflections:						
	Type	Screen Space	Ray Tracing	Ray Tracing	Ray Tracing	?
	Max. Rougness	-	0.6	0.6	0.6	?
	Max. Bounces	-	1	2	5	?
	Samples Per Pixel	-	1	1	3	?
	Shadows (Type)	-	Hard Shadows	Hard Shadows	Area Shadows	?
Translucency:						
	Type	Raster	Ray Tracing	Ray Tracing	Ray Tracing	?
	Max. Roughness	-	0.6	0.6	0.6	?
	Max. Refractions	-	-	3	5	?
	Samples Per Pixel	-	1	1	3	?
	Shadows (Type)	-	N	Hard Shadows	Area Shadows	?
	Refraction	-	N	Y	Y	?
Shadows:						
	Type	Shadow Map	Ray Tracing	Ray Tracing	Ray Tracing	?
	Samples Per Pixel	-	1	1	1	?

Figure 6.5: This table contains detailed settings and their values for each of the quality presets.

While all five options are available in the benchmarking scene, only the Low, Medium, and High presets can be toggled in the puzzle game part. This effectively means that at the moment, there is no option to turn ray tracing completely off while playing the puzzle game scene. Furthermore, the benchmarking scene provides users with the Custom preset. The Custom preset allows for individual tweaking and testing of each setting and is controlled by interacting (E) with corresponding labels located on a wall adjacent to the Lobby, shown in figure 6.6.

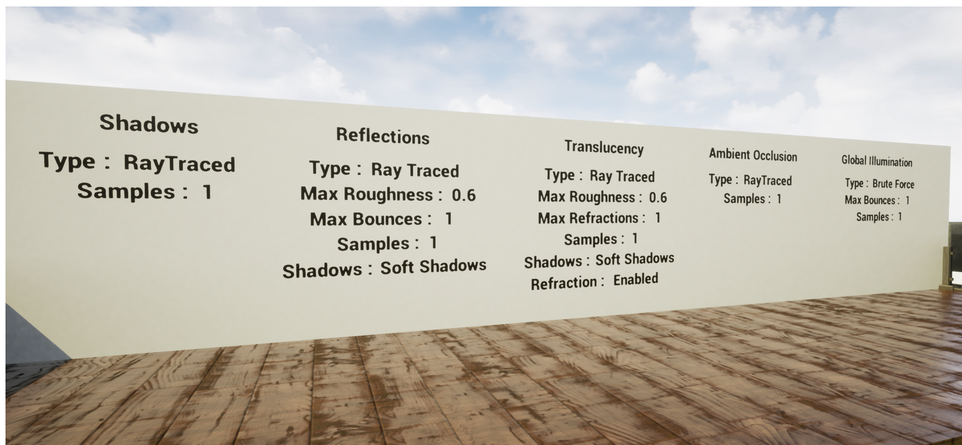

Figure 6.6: Second part of the lobby, with interactable labels of all the available ray tracing settings.

6.3 Quality Presets User Testing

In this section, we test how well we have designed the quality presets for the playable demo. As mentioned in the previous section, there are three quality presets in the gameplay part - LOW, MEDIUM, HIGH. The goal is to test whether the presets are well balanced in terms of visual quality and performance ratio. Figure 6.7 shows a table of all three presets with their specific settings.

		Low	Medium	High
Ambient Occlusion:				
	Enabled	N	Y	Y
	Samples Per Pixel	-	1	3
Global Illumination:				
	Enabled (Type)	Final Gather	Brute Force	Brute Force
	Max. Bounces	1	1	2
	Samples Per Pixel	8	2	3
Reflections:				
	Type	Ray Tracing	Ray Tracing	Ray Tracing
	Max. Rougness	0.6	0.6	0.6
	Max. Bounces	2	4	5
	Samples Per Pixel	1	1	3
	Shadows (Type)	Hard Shadows	Hard Shadows	Area Shadows
Translucency:				
	Type	Ray Tracing	Ray Tracing	Ray Tracing
	Max. Roughness	0.6	0.6	0.6
	Max. Refractions	-	3	5
	Samples Per Pixel	1	1	3
	Shadows (Type)	N	Hard Shadows	Area Shadows
	Refraction	N	Y	Y
Shadows:				
	Type	Ray Tracing	Ray Tracing	Ray Tracing
	Samples Per Pixel	1	1	1

Figure 6.7: A table of settings for each of the presets available in the playable demo.

Before we look at user opinions, let us look at two examples of the presets in action. The first puzzle, which is a mirror maze (see figure 6.8), is heavily dependant on the quality of reflections; it is especially influenced by the number of reflection ray bounces, which determine how 'far' we see into each mirror. It seems that the LOW preset does not provide much in terms of visual quality, while the MEDIUM preset provides acceptable frame times with minimum visual difference to the HIGH preset.

Figure 6.8: A comparison of the presets and their effect on the mirror maze puzzle scene.

Another example shows the last puzzle of the demo, with a focus on global illumination. Looking at figure 6.9, we see there is minimal difference between the LOW and MEDIUM presets, both in visual quality and performance. The effect that is mostly affecting the performance of this scene is global illumination. Looking back at the preset settings table (figure 6.7), the LOW preset is using the Final Gather method and the MEDIUM preset uses the Brute Force method. Comparing these single-bounce results with the two bounces when using the HIGH preset, the visual difference is noticeable, mainly because we are looking at an indoor scene only lit by several lights.

Figure 6.9: A comparison of the presets and their effect on the global illumination focused maze scene.

During the testing, we asked a dozen users to play through the gameplay demo, at the end of which was a button linked to a short survey. The survey mainly focused on finding out what the users thought about the distribution of the presets - whether they were balanced in terms of visual quality vs performance.

What GPU did you use to test the demo?

Short-answer text

Figure 6.10: The first question of the survey.

The first question (see figure 6.10) was a simple check for the GPU used to test demo. This question served mostly the purpose of filtering out answers with non-RTX GPUs, as they would potentially be evaluated separately and differently. None of the answers submitted contained a non-RTX GPU.

Which quality seting did you use most of the time?

- Low
- Medium
- High

Figure 6.11: The second question of the survey.

The second question of the survey (see figure 6.11), asked the users which preset they played with most of the time. Paired with the next questions, which ask how each of the presets performed and felt, we wanted to find out whether the users prefered performance over visual quality, or the other way round. From the twelve answers we collected, eight users preferred the MEDIUM preset, while the rest were split between LOW and HIGH.

Do you think the quality settings (low, medium, high) correspond with the performance and visual quality change in-game? E.g.: Low settings = good performance, bad visuals

- Absolutely yes
- Yes
- Not sure
- No
- Absolutely no

Figure 6.12: The third question of the survey.

The third question (see figure 6.12) is by far the most controversial one. We asked whether the users thought that the presets were equally spaced - whether they provided equal steps in visual quality vs performance ratio. While half of the testers agreed with the preset spacing, the other half was mixed between 'Not Sure' and 'No'. Based on answers from other questions in the survey, it became obvious that the performance between the MEDIUM and HIGH presets was very high, especially when compared with the little to no performance difference between the LOW and MEDIUM presets.

The following three questions each discuss one individual quality preset. We asked the users how the game felt, how it looked, how was the performance. We wanted to gather additional data for each of the presets so that it would be possible to analyze our settings choices for individual presets further.

Briefly describe your gameplay experience while playing on Low settings. (How smooth did the framerate feel, how did the game look, etc.)

Short-answer text

Figure 6.13: The fourth question of the survey.

The fourth question (see figure 6.13) asks about the user experience when playing with the LOW preset. As expected, many users complained that the game looked ugly; some even thought that rasterization would look better. On the other hand, performance seemed decent across all users.

Briefly describe your gameplay experience while playing on Medium settings. (How smooth did the framerate feel, how did the game look, etc.)

Long-answer text

Figure 6.14: The fifth question of the survey.

The fifth question (see figure 6.14) discusses the MEDIUM preset. This seemed to be the sweet spot for most of the players, confirming the outcome of the second question (Which preset did you use most of the time?). As reported by the users, while playing on MEDIUM, the game felt comparably smooth to the LOW preset, while providing a significant increase in performance.

Briefly describe your gameplay experience while playing on High settings. (How smooth did the framerate feel, how did the game look, etc.)

Long-answer text

Figure 6.15: The sixth question of the survey.

The sixth question (see figure 6.15) asks about the HIGH preset. The users reported that the game was very choppy, which eliminated the increase in visual quality. Based on the answers, it seems that we failed to design the HIGH preset so that it would even be considered as an option when playing the game.

Other comments

Long-answer text

Figure 6.16: The seventh question of the survey.

The final seventh question (see figure 6.16) allowed the users to add any additional message or piece of information. While trying the various presets, two users noticed that the difficulty of the second puzzle - the mirror maze - was heavily influenced by the preset they chose. This is one of the problems we encountered when designing the demo, especially with ray traced reflections, as the number of bounces significantly influences what the player sees. In this case, when playing on the LOW preset, two reflection bounces give very little information for the player to navigate the maze (see figure 6.8).

Overall the survey proved very useful to analyze the created presets. While the presets provide some control over the performance, they are not very well balanced and require further changes. Optimization of the whole demo is also needed, as there are still options we can tweak to balance the performance.

Conclusion

This thesis analyzed the ray tracing capabilities of Unreal Engine and showcased them on playable demos, which are available for the reader to try. We summarized the basics of ray tracing and explained why the technology is becoming increasingly more popular amongst game developers. In the last part, we conducted a short user survey, to test the created quality presets, and to evaluate the visual quality vs performance ratio of the demo.

We have experimented with many options, settings, and setups regarding Ray Tracing in Unreal Engine. It became apparent that while some scenes do benefit from the new technology, others are not great candidates, as the Ray Tracing effects provide little to no visual improvement at a high computational cost. Comparing the GTX 1080 and RTX 2070 Super performance, non-RTX GPUs do not have enough computing power to produce playable framerates, even with most of the ray traced effects disabled.

This thesis can serve as an introduction to Unreal Engine's Ray Tracing capabilities. It can help readers in setting up a basic project of their own while giving them valuable insight on how to improve the scene's performance. Moving forward, we would like to iterate on the tests, observing the impact of individual settings on the engine's performance. The included demo also requires more work, as there is plenty of room for optimization, to improve the framerates across all quality settings.

Appendix A

Bibliography

[AMD] *Amd rdna 2*, https://www.pcgamer.com/amd-rdna2-release-date-big-navi-specs-price-performance/, Accessed: 2020-01-12.

[AMT18] Hoffman Naty Akenine-Moller Tomas, Haines Eric, *Real-time rendering*, CRC Press, 2018.

[App68] Arthur Appel, *Some techniques for shading machine renderings of solids*, AFIPS '68, 1968.

[HAM19] Eric Haines and Tomas Akenine-Möller (eds.), *Ray tracing gems*, Apress, 2019, http://raytracinggems.com.

[Kaj86] James T. Kajiya, *The rendering equation*, ACM SIGGRAPH Computer Graphics, 1986.

[OPX] *Nvidia optix ray tracing engine*, https://developer.nvidia.com/optix, Accessed: 2020-08-05.

[Rec07] Meinrad Recheis, *Realtime ray tracing*, Vienna University of Technology, 2007, https://www.cg.tuwien.ac.at/courses/Seminar/SS2007/RealtimeRayTracing-Recheis.pdf.

[REE] *Ray tracing essentials part 6: The rendering equation*, https://news.developer.nvidia.com/ray-tracing-essentials-part-6-the-rendering-equation/, Accessed: 2020-08-03.

[RTV] *Ray tracing in vulkan*, https://www.khronos.org/blog/ray-tracing-in-vulkan, Accessed: 2020-08-04.

[TR12] Thorsten Grosch Jan Kautz Tobias Ritschel, Carsten Dachsbacher, *The state of the art in interactive global illumination*, Computer Graphics Forum, 2012.

[UER] *Real-time ray tracing in unreal engine*, `https://docs.unrealengine.com/en-US/Engine/Rendering/RayTracing/index.html`, Accessed: 2020-01-15.

[VKR] *Introduction to real-time ray tracing with vulkan*, `https://devblogs.nvidia.com/vulkan-raytracing/`, Accessed: 2020-01-15.

[Whi80] Turner Whitted, *An improved illumination model for shaded display*, Communications of the ACM, 1980.