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

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 con rm that I have only used the speci ed 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 e ects 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 gures to support the claims.

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

Supervisor: doc. Ing. Ji°í Bittner, Ph.D.

Abstrakt

Cílem této práce je otestovat, vyhodnotit a p°ednést schopnosti implementace ray tracingu v Unreal Enginu. ƒtená°i jsou p°edstaveny základní principy technologie Sledování Paprsk· v Reálném ƒase, spolu s efekty zaloºenými na této technologii, jeº jsou v Unreal Enginu dostupné. V návaznosti na testování n¥kolika r·zných scéná°· jsou vysv¥tleny a doporu£eny postupy pro optimalizaci scén, spolu s výhodami a nevýhodami pouºívání ray tracingu.

Klí£ová slova: sledování paprsk· v reálném £ase, unreal engine, ray tracing ve videohrách, ray tracing

P°eklad názvu: Sledování paprsk· v reálném £ase v Unreal Enginu

Contents

Figures

Tables

Chapter 1

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 e ects, 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 e ects, such as ray traced shadows, re ections, 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 rst 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 e ects are explained, along with their settings and comparison of di erent options. The benchmarks that are evaluated in later chapters consist of several di erent scenes, each aimed to demonstrate and test a di erent e ect. 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 e ects rst 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 Re ectance Function (BRDF), which describes how light is re ected 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 di erent 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 di erent techniques for solving it [TR12]. We brie y 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 di erent 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, t 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 e ects into their games and engines. DXR can be seen in games like Battle eld 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 di erence 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 re ections.[RTV]