

AIN SHAMS UNIVERSITY
FACULTY OF COMPUTER & INFORMATION SCIENCES
DEPARTMENT OF SCIENTIFIC COMPUTING



Performance Optimization of Image Rendering on Programmable Graphics Hardware

A Thesis Submitted to the Department of Scientific Computing, Faculty of Computer and Information Sciences, Ain Shams University, in Partial Fulfilment of the Requirements for the Master Degree of Computer and Information Sciences

By

Ahmed Osman Aiad

Supervised by

Prof. Dr. Mohammed Essam Khalifa

Dean of Faculty of Computer and Information Sciences,
Ain Shams University

Dr. Ahmed Hamed Kaboudan

Computer Science Department,
Higher Institute of Computer Science
And Information Technology

Dr. Safwat Helmy Hamad

Scientific Computing Department,
Faculty of Computer and Information Sciences,
Ain Shams University

Cairo, 2010

ACKNOWLEDGMENTS

I am deeply indebted to, Prof. Mohammed Essam Khalifa, for his endless help, and support during last year. His advice and insight have been invaluable throughout my entire time of research. Our meetings gave me academic guidance and provided the milestones that helped to develop my research skill.

I want to express my deep gratitude to Dr. Ahmed Kaboudan for his Significant and academic support. His continuous contact with me gave me a really strong push toward achieving my research goals.

I would like to extend my thanks to Dr. Safwat Hamad for his continuous encouragement, understanding and support. He stood by me throughout all the difficulties I faced during the last year. He pushed me to fill in that one last detail to elevate the level of my thinking and my work. I also admire and respect his integrity both as a researcher and as a person

I won't forget to also thank Prof. Dr. Ahmed Gabr and Dr. Ahmed alabbassy, who gave me a lot of time and effort. I cannot possibly thank them enough.

Finally I am really grateful to my wife for her deep caring and very strong encouragement that helped me in finishing this work. Furthermore, this thesis would have never been possible without the support of my family, I am deeply grateful for my father and my mother.

ABSTRACT

Computer graphics have become commonplace in our daily lives. Nearly every modern high budget movie has some sort of digital special effects shot. Even small budget movies and local television advertisements tend to utilize some graphics technology. There are even several cartoon shows that are completely animated hand rendered using a computer. Yet aside from a small handful of high production cost movies, even non-experts can tell when computer rendered special effects are used.

Several factors contribute to effects shot, including modeling, animation, and rendering. If we focus just on rendering the answer is simple: time. When realistic imagery is required for example when adding computer graphics effects into a live action film the computational cost of generating the image increases. This cost increase prevents many films from achieving the visual fidelity necessary for seamless effects integration. It also prevents applications like games, which require fast rendering times, from having realistic imagery.

Ray tracing is an image synthesis technique which simulates the interaction of light with surfaces. Most high-quality, photorealistic renderings are generated by global illumination techniques built on top of ray tracing. Real-time ray tracing has been a goal of the graphics community for many years. Unfortunately, ray tracing is a very expensive operation. VLSI technology has just reached the point where the computational capability of a single chip is sufficient for real-time ray tracing. Supercomputers and clusters of PCs have only recently been able to demonstrate interactive ray tracing and global illumination applications. In this dissertation we show how a ray tracer can be written as a stream program. We present a stream processor abstraction for the modern programmable graphics processor (GPU) | allowing the GPU to execute stream programs. We describe an implementation of our streaming ray tracer on the GPU and provide an analysis of the bandwidth and computational requirements of our implementation. In addition, we use our ray tracer to evaluate simulated GPUs with enhanced program execution models. We also present an implementation and

evaluation of global illumination with photon mapping on the GPU as an extension of our ray tracing system. Finally, we examine hardware trends that favor the streaming model of computation. Our results show that a GPU-based streaming ray tracer has the potential to outperform CPU-based algorithms without requiring fundamentally new hardware, helping to bridge the current gap between realistic and interactive rendering.

TABLE OF CONTENTS

Abstract	ii
Table of Contents	iv
List of Figures	vii
List of Tables	ix
Chapter 1: Introduction	1
1.1 Motivation and problem Statement	2
1.2 Research Objectives	3
1.3 Thesis Overview	3
Chapter 2: Background Overview	5
2.1 Graphics Rendering Pipeline	6
2.1.1 Application Stage	6
2.1.2 Geometry Stage	7
2.1.3 Rasterizer Stage	8
2.2 The Graphics Proceesing Unit	10
2.3 GPU Pipeline Overview	11
2.3.1 The Vertex Shader	11
2.3.2 The Geometry Shader	12
2.3.3 The Pixel Shader	12
2.3.4 The Merging Stage	13
2.3.5 Effects	13
2.4 Introduction to Stream Programming	13

Chapter 3: Literature Survey.....	16
3.1 Introduction	17
3.1.1 Rendering	19
3.2 Real Time Rendering Techniques	20
3.2.1 Scale Line Rendering and Rasterisation	20
3.2.2 Ray Casting	21
3.3 Non Real Time Rendering Techniques	22
3.3.1 Radiosity	23
3.3.2 Photon Mapping	25
3.3.3 Ray Tracer	25
3.4 Reflection and Shading Models	26
3.4.1 Reflection	27
 Chapter 4 : Programming Graphics Processing	 28
4.1 The Programmable Fragment Processor as a Stream Processor...	29
4.2 Streams	30
4.3 Flow Control	32
4.4 Texture Memory as Memory	33
 Chapter 5: Stream Ray Tracer Design and Acceleration.....	 36
5.1 Introduction To Ray Tracer	37
5.1.1 Simple Ray Tracing Implementation	38
5.2 Data Structure.....	39
5.2.1 Creating the Uniform Grid	40
5.2.2 Traversing The Uniform Grid	41
5.3 Removing Recursion.....	44
5.3.1 Reflection	44
5.3.2 Refraction	45

5.4 Storing The 3D Scene	45
5.5 Kernels For Ray Tracer	46
5.5.1 Primary Ray Generator	46
5.5.2 Voxel Pre-calculation	47
5.5.3 Ray Traverser	47
5.5.4 Ray Intersector	48
5.5.5 Load Balancing Traversing And Intersection Loop Intersector	49
5.6 Ray Tracer Acceleration	50
5.6.1 Object Intersection Buffer	51
5.6.2 Shadow Mapping	53
5.7 OIB Implementation	53
5.7.1 Ray Casting on the CPU	54
5.7.2 Ray Casting on GPU	55
5.8 Ray Tracer with OIB	55
 Chapter 6: Photon Mapping on GPU	 57
6.1 Introduction	58
6.2 Photon Mapping On GPU.....	60
6.2.1 Photon Tracing	60
6.2.2 Constucting The Photon Map Data Structure	61
6.2.3 The Radiance Estimate	66
6.2.4 Rendering	67
 Chapter 7: Performance Evaluation & Analysis.....	 68
7.1 Introduction	69
7.2 Ray Tracer without Acceleration	70
7.3 Accelerated Ray Tracer.....	74
7.3.1 Ray Casting	74
7.4 Photon Mapping.....	77
 Conclusions And Future work	 80
 Bibliography	 82

LIST OF FIGURES

Figure 2.1	The geometry stage subdivided into a pipeline of functional stages	7
Figure 2.2	Orthographic, parallel and perspective projection	8
Figure 2.3	The rasterizer stage subdivided into a pipeline of functional stages	9
Figure 2.4	GPU implementation of the rendering pipeline	11
Figure 2.5	The stream programming model	14
Figure 3.1	An image created by using POV-Ray	18
Figure 3.2	An example of a ray-traced image	23
Figure 4.1	The programmable fragment processor as a stream processor	32
Figure 4.2	Multiple kernel invocations via multi-pass rendering	33
Figure 4.3	Dependent texture lookup as a pointer dereference. A simple four triangle mesh can be stored in two textures	36
Figure 5.1	Ray Tracing Basic Concept	39
Figure 5.2	Spatial Data Structure(Uniform Grid)	42
Figure 5.3	In the uniform grid, the triangle is referenced every cell containing a part of the triangle	42
Figure 5.4	Scene converted into Uniform Grid	43
Figure 5.5	Traversing Uniform Grid using a 3D-DDA Algorithm	45
Figure 5.6	Ray Path	46
Figure 5.7	Storing the uniform grid structure in textures	48
Figure 5.8	Kernel Sequence Diagram of the Simple Ray Tracer	48
Figure 5.9	Primary Ray Generator	48
Figure 5.10	Voxel Pre-calculation	49
Figure 5.11	Ray Traverser	50
Figure 5.12	Ray Intersector	50
Figure 5.13	Example Traverse/Intersection Loop	51
Figure 5.14	The OIB data structure contains the object indices (right) as well as the interpolated (u,v)-coordinates of the visible triangles (left)	53
Figure 5.15	Two triangles with the same z-value $z = 0.5$ (left) and two overlapping triangles with almost the same z-value where the depth test fails (right)	54
Figure 5.16	Artefacts without multisampling and interpolation, and no visible artefacts with 3×3 multisampling and interpolation	55
Figure 5.17	Interleaved rendering leads to higher frame rates than normal rendering, but also to a blurred image	57
Figure 5.18	The animated robots scene rendered with reflections by our hybrid ray tracer.	58
Figure 6.1	Photon mapping system flow	62
Figure 6.2	Stages in a bitonic sort of eight elements	64
Figure 6.3	code for the bitonic merge sort fragment program	65
Figure 6.4	Resultant textures for a grid-based photon map generated by bitonic	65
Figure 6.5	Building the photon map with stencil routing	67

Figure 6.6	Computing the radiance estimate with the kNN-grid	69
Figure 7.1	Application Model	72
Figure 7.2	Scene A	74
Figure 7.3	Scene B	74
Figure 7.4	Scene C	74
Figure 7.5	Scene B ray traced	75
Figure 7.6	Scene C ray traced	75
Figure 7.7	Render Time for Different GPUs	75
Figure 7.8	Render Time: Different Grid Size	76
Figure 7.9	GPU vs. CPU: 1 Iteration	76
Figure 7.10	CPU vs. GPU: Different Iterations	76
Figure 7.11	The almost constant frame rate	79
Figure 7.12	Test scene renderings	81

LIST OF TABLES

Table 5.1	Variables	46
Table 5.2	Reflection Iteration	47
Table 5.3	Ray traversing state	50
Table 7.1	Demo Scenes	73
Table 7.2	Accelerated ray tracer bench mark scenes and their number of triangles	77
Table 7.3	Rendering performances (fps) measured with and without shadow rendering	78
Table 7.4	Rendering performances (fps) measured with and without multisampling	78
Table 7.5	Frames per second of a traditional ray tracer using a kd-tree hybrid ray tracer	79
Table 7.6	GPU render times in seconds for the scenes	82

INTRODUCTION

- 1.1 Motivation and Problem Statement**
 - 1.2 Research Objectives**
 - 1.3 Thesis Overview**
-
-

CHAPTER 1

Introduction

1.1 Motivation and problem Statement

Computer graphics have become commonplace in our daily lives. Nearly every modern high budget movie has some sort of digital special effects shot. Even small budget movies and local television advertisements tend to utilize some graphics technology. There are even several cartoon shows that are completely animated, and rendered using a computer. Yet aside from a small handful of high production cost movies, even non-experts can tell when computer rendered special effects are used. Why are computer rendered images easy to pick out? Several factors contribute to an effects shot, including modelling, animation, and rendering. If we focus just on rendering the answer is simple: time. When realistic imagery is required for example when adding computer graphics effects into a live action film the computational cost of generating the image are increased. This cost increase prevents many films from achieving the visual fidelity necessary for seamless effects integration. It also prevents applications like games, that require fast rendering times, from having realistic imagery. Most of the high-quality, photorealistic renderings made today rely on a rendering technique called ray tracing. Ray tracing simulates the interaction of light with surfaces, a process which is very computationally expensive. When the rendering does not have to happen interactively, such as a special effects shot for a movie, it is simply a matter of compute time and resources to perform the simulations necessary to generate realistic images. When rendering speed matters, image quality is often traded off for increased speed. This trade off point varies from production to production and plays a large role in determining its achievable level of realism. We are interested in bridging the gap between realistic and interactive graphics. State of the art ray tracers today can render scenes at several frames per second on a supercomputer or on a cluster of high-end PCs. Unfortunately games, CAD, and other single-user applications, cannot rely on every person having a supercomputer or cluster to run them on. Instead, they exploit commodity graphics processors (GPUs) and render images with as high of quality as possible. Graphics processors have improved significantly in the past several years both in terms of speed and in terms of the type of shading they support. Most recently, they have exposed a fairly general programming environment which allows developers the freedom to write very realistic looking shader. In this thesis we explore the reformulation of high quality rendering algorithms for interactive rendering. We show that ray tracing can be expressed as a streaming computation, and that high performance processors like GPUs are well suited for supporting streaming computations. We can abstract the GPU as a general stream processor. The combination of this reformulation and abstraction allows us to implement high quality rendering algorithms at interactive rates on commodity programmable graphics hardware. Furthermore, we will argue that ray tracing is most naturally and efficiently expressed in the stream programming model.

1.2 Research Objectives

This thesis makes several contributions to the areas of computer graphics and graphics hardware design: We present a streaming formulation for ray tracing. Streaming is a natural way to express ray tracing. Modern high performance computing hardware is well suited for the stream programming model. The stream programming model helps to organize the ray tracing computation optimally to take advantage of modern hardware trends. We have developed a stream processor abstraction for the programmable fragment engine found in modern graphics processors. This abstraction helps us to focus on our algorithm rather than on the details of the underlying graphics hardware. More broadly, this abstraction can support a wide variety of computations not previously thought of as being mappable to a graphics processor. We have also developed an abstraction for the GPU memory system. We show that dependent texturing allows texture memory to be used as a read only general purpose memory. We can navigate complex data structures stored in texture memory via dependent texture lookups. As with our stream processor abstraction, our memory abstraction has proved useful to a wide variety of algorithms. We have implemented and evaluated a prototype of our streaming ray tracer on current programmable graphics hardware. Our implementation runs at comparable speeds to a fast CPU-based implementation even though current GPUs lack a few key stream processor capabilities. The GPU-based implementation validates our ray tracing decomposition and GPU abstractions. We have extended our ray tracer to support photon mapping. Our implementation solves the sorting and searching problems common to many global illumination algorithms in the streaming framework. This implementation is an initial step in demonstrating that the streaming model is valid for advanced global illumination computations as well as simple ray tracing.

We analyze how other ray tracing systems are optimized to take advantage of fundamental hardware trends. We show that these optimizations take advantage of the same trends that the streaming approach does, only in a much more cumbersome way than is exposed by the stream programming model. We conclude that high performance ray tracing is most naturally expressed in the stream programming model. In this thesis, we are primarily concerned with studying static scenes { scenes in which the light sources or eye position can move but scene geometry is fixed. Our analysis does not account for any scene pre-processing times. Efficient ray tracing of dynamic scenes is an active research area in the ray tracing community, and is beyond the scope of this thesis.

1.3 Thesis Overview

This thesis is organized in seven chapters including this one. Their contents are described briefly in the following lines:

- **Chapter 2 Background Overview:** in this chapter we give a brief review of some of the subjects that are related to the field of our research.
- **Chapter 3- Literature Survey:** this chapter provides an overview on rendering techniques; both real time and non real time techniques, while focusing on the ray tracer considered in this research.
- **Chapter 4- Programmable Graphics Processor Abstractions::** this chapter provides an overview on abstract the programmable fragment engine in current GPUs as a stream processor

- **Chapter 5- *Stream Ray Tracer Design And Acceleration*:** this chapter presents algorithms for decompose ray tracer on graphics hardware and how to accelerate this algorithms
- **Chapter 6- *Photon mapping on GPU*:** this chapter presents algorithms for decompose photon mapping on graphics hardware and how to map this algorithms
- **Chapter 7- *Performance Analysis & Evaluation*:** in this chapter, the performance of the proposed algorithms was tested and compared over a number of case studies representing a combination of implement ray tracer and photon mapping on CPU and GPU .