Philipps-Universität Marburg

Fachbereich 12 - Mathematik und Informatik

Master Thesis

Dynamic Insertion of 3D Objects from CAD Files into Unreal Engine

Matija Miskovic

Septermber 2022

Supervisor:
Prof. Dr. Thorsten Thormählen

Research Group Graphics and Multimedia Programming

Declaration of Originality

I, Matija Miskovic (Computer Science Student at Philipps-University Marburg, Matrikelnummer 3139015), versichere an Eides statt, dass ich die vorliegende Diplomarbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe. Die hier vorliegende Diplomarbeit wurde weder in ihrer jetzigen noch in einer ähnlichen Form einer Prüfungskommission vorgelegt.

Marburg, 17. Dezember 2009

Max Mustermann

Abstract

Viele der in der Computergrafik verwendeten 3D-Modelle werden mit Hilfe der Dreiecksnetze repräsentiert. ... (max. 1 Seite)

Abstract

text text text text text text text text text text text text text text text text text text text text text text (exakte englische Übersetzung der deutschen Kurzfassung)

Table of Contents

1 Introduction

Diese Diplomarbeit beschäftigt sich mit den Parallel View-Dependent Compressed Progressive Meshes und deren Umsetzung in die vom Grafikkartenhersteller NVIDIA entwickelte parallele Programmiersprache CUDA. Dazu gehört die Entwicklung einer für die parallele Verarbeitung geeignete effiziente Datenstruktur, sowie eine effiziente Datenverwaltung.

1.1 Motivation

Die aktuelle Entwicklung der Multimediaindustrie versucht zunehmend die Simulation von virtuellen Welten realistisch darzustellen. Die Ansprüche der Anwender werden mit der Zeit immer größer und dementsprechend die generierte virtuelle Realität immer komplexer. So eine Entwicklung ist unweigerlich mit der Steigerung der erforderlichen Rechenleistung verbunden, da die simulierten Objekte aus Millionen von Polygonen bestehen können und in Echtzeit dargestellt werden müssen. Im Laufe der Jahre sind viele verschiedene Verfahren entwickelt worden, die das Ziel hatten, die komplexen Objekte mit einem vertretbaren Qualitätsverlust in Echtzeit darzustellen. Der mit Abstand beste Ansatz, um den Kompromiss zwischen Qualität und Geschwindigkeit zu finden, ist die View-Dependent Progressive Meshes. Einer der Vorteile dieser Herangehensweise ist, dass dieses Verfahren hochgradig parallelisierbar ist, so dass sich mit einer geeigneter Programmiersprache und Hardware eine beachtliche Effizienzsteigerung erzielen lässt.
Die von NVIDIA entwickelte parallele Programmiersprache CUDA setzt auf den aktuellen Trend der GPGPUs und ermöglicht es mit einer kostengünstigen Grafikkarte, die in fast jeden Desktoprechner vorhanden ist, Programme effizient zu parallelisieren. Aus diesem Grund ist CUDA für das Parallelisieren von View-Dependent Progressive Meshing besonders geeignet.

1.2 Goals

Das Ziel dieser Arbeit ist die Entwicklung einer effizienten parallelen Implementierung von komprimierten View-Dependent Progressive Meshes in CUDA, welche in der Lage ist, Objekte die aus mehreren Millionen von Polygonen bestehen können, in Echtzeit zu verarbeiten.

Echtzeit

Das entwickelte Programm soll selbst sehr große Polygonnetze effizient verarbeiten können. Die Eingaben des Benutzers für die Translation und Rotation des Objekts sollen in

Echtzeit umgesetzt werden. Die durchschnittliche Laufzeit des Programms pro Frame soll höchstens drei Mal soviel Zeit wie das Rendering des gegebenen Modells benötigen, um eine Echtzeitdarstellung des Modells zu ermöglichen. Dabei können die Modelle aus mehreren Millionen von Dreiecken bestehen.

Kosten

Das Programm sollte mit der normalen, kostengünstigen Privatanwender-Hardware laufen, sodass für die Ausführung keine Spezialrechner benötigt werden. Die einzige Vorrausetzung an das System ist eine NVIDIA-Grafikkarte die CUDA 1.1 unterstützt. Diese ist aber in den meisten Desktoprechnern vorhanden oder kann kostengünstig nachgerüstet werden.

1.3 Thesis Structure

Im ersten Abschnitt des Kapitels **??** soll zunächst die Bedeutung der Grafikkarte als Berechnungseinheit verdeutlicht werden. Dann soll im zweiten Abschnitt die Hard- und Softwarearchitektur der Programmiersprache CUDA beschrieben werden, sowie einige Vorschläge zu Codeoptimierung diskutiert, bevor im Kapitel **??** ein Überblick über die wichtigsten Verfahren zur Echtzeitdarstellung komplexer Objekte geben wird. An dieser Stelle werden auch das View-Dependent Progressive Meshing, sowie einige Simplifizierungstechniken genauer erläutern. Kapitel **??** beschäftigt sich mit der Theorie des im Rahmen dieser Diplomarbeit entwickelten Algorithmus. Dabei sollen die Datenstrukturen, die Kompression, sowie die einzelnen Schritte des Algorithmus genauer erläutert werden. Die Implementierung des Algorithmus in CUDA wird im Kapitel 5 besprochen, dabei sollen die benutzten Bibliotheken, sowie die CUDA-spezifische Umsetzung des Programms beschrieben werden. Anschließend werden im Kapitel 6 die durchgeführten Tests und deren Ergebnisse dokumentiert und diskutiert, sowie im Kapitel 7 ein Ausblick auf weiterführende Arbeiten gegeben.

1.4 Related Works

Im Themenbereich der Progressive Meshes und View-Dependent Progressive Meshes gab es schon am Ende des letzten Jahrzehnts einige Veröffentlichungen [Hop96, Hop97]. Diese waren zwar eine gute und notwendige Weiterentwicklung vom klassischen LOD-Algorithmus, ermöglichten aber nicht eine effiziente Echtzeitdarstellung von großen Modellen. In [Hop98] wurde schließlich ein Versuch unternommen eine effizientere Datenstrucktur zu entwickeln, um den Speicherverbrauch zu optimieren und bessere Geschwindigkeit zu erreichen. Diese effizientere Datenstruktur brachte zwar einige Verbesserungen, ermöglichte aber dennoch keine Echtzeitdarstellung von großen Modellen. Seit dem gab es eine Reihe von Verfahren, die das Ziel hatten eine effiziente Echtzeitdarstellung von großen Modellen zu ermöglichen. Einige von diesen Verfahren nutzten Multi-Triangulationen [DFMP98], andere Versuchten die View-Dependent Progressive Meshes weiterzuentwickeln [Paj01,

PD04,ESV99]. Doch keins dieser Verfahren konnte die Anforderungen vollständig erfüllen. Eine erst kürzlich veröffentlichte Arbeit [HSH09] machte endlich einen Schritt in die richtige Richtung. Die in dieser Arbeit implementierte GPU-Variante von parallelen View-Dependent Progressive Meshes ermöglichte eine akzeptable Echtzeitdarstellung von großen Modellen. Diese braucht durchschnittlich das dreifache der Zeit, die für das Rendering des Modells benötigen wird und lässt somit einen großen Spielraum für die Optimierung offen.

2 Unreal Engine

In order to understand the aim of this project and what sort of problems might arise, we first need to have a basic understanding of the software for which it is being developed for and which is also being used to develop it. In this case that is the Unreal Engine.

The Unreal Engine is a 3D graphics video game engine, first created for the first person shooter Unreal in 1998.

2.1 Unreal Engine Basics

2.1.1 Actors and Components

2.1.2 Pawns and Player Controllers

text

2.2 C++ and Blueprints

As already described in 2.1 ...

2.3 Unreal Engine Networking

text

3 Dynamic 3D Object Insertion

In diesem Kapitel soll das eigene Verfahren beschrieben werden. Es geht dabei nicht nur darum zu beschreiben was gemacht wurde, sondern ebenfalls darum zu begründen, weshalb bestimmte Design-Entscheidungen getroffen wurden. Das Kapitel sollte nicht „Eigenes Verfahren“ heißen, sondern etwas mit dem Titel der Arbeit zu tun haben.

3.1 File Formats and Parsing

3.1.1 File Sharing

3.2 Runtime Mesh Generation

3.2.1 Runtime Mesh Component

3.2.2 Material Generation

3.3 Object Interaction

3.3.1 Mouse and Keyboard

3.3.2 Virtual Reality

3.4 Plug-in Installation and Usage

4 Results and Evaluation

In diesem Kapitel sollen die Ergebnisse dieser Diplomarbeit diskutiert werden.

4.1 CAD Runtime Loader

4.2 Comparison to related works

4.3 Shortcomings and possible Improvements

5 Conclusion

In diesem Kapitel sollen zunächst die erreichten Ziele diskutiert und abschließend ein Ausblick auf mögliche, weiterführende Arbeiten gegeben werden.

Bibliography

[CEI01] Carl Erikson D. M., III W. V. B.: Hlods for faster display of large static and dynamic environments. *University of North Carolina at Chapel Hill* (2001).

[Cha08] Charpentier F.: *Nvidia Cuda: Das Ende der CPU?* Technical report, Tom's Hardware, Jun 2008. http://www.tomshardware.com/de/CUDA-Nvidia-CPU-GPU,testberichte-240065.html (20.08.2009).

[Cla76] Clark J. H.: Hierarchical geometric models for visible surface algorithms. *Communications of the ACM 19*, 10 (Oct 1976), 547–554. http://design.osu.edu/carlson/history/PDFs/clark-vis-surface.pdf (09.09.2009).

[DFMP98] De Floriani L., Magillo P., Puppo E.: Efficient implementation of multi-triangulations. In *VIS '98: Proceedings of the conference on Visualization '98* (Los Alamitos, CA, USA, 1998), IEEE Computer Society Press, pp. 43–50.

[Eck99] Eckert M.: *Von-Neuman Architektur.* Technical report, TecChannel, sep 1999. http://www.tecchannel.de/server/prozessoren/401364/so_funktioniert_ein_prozessor (19.08.2009).

[Eis06] Eiserle M.: Progressive techniken in der computergrafik. *Universität Rostock* (2006). http://vcg.informatik.uni-rostock.de/assets/publications/theses_sem/SA_Eiserle2006.pdf (15.09.2009).

[ESV99] El-Sana J., Varshney A.: Generalized view-dependent simplification. *Computer Graphics Forum 18*, 3 (1999), 83–94.

[Fos95] Foster I.: *Designing and Building Parallel Programs.* Addison Wesley Pub Co Inc, Reading, MA, USA, 1995.

[Har08] Harris M.: *Parallel Prefix Sum (Scan) with CUDA.* Programming guide, NVIDIA, 2008.

[Hop96] Hoppe H.: Progressive meshes. In *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1996), ACM, pp. 99–108.

[Hop97] Hoppe H.: View-dependent refinement of progressive meshes. In *SIGGRAPH '97: Proceedings of the 24th annual conference on Computer*

graphics and interactive techniques (New York, NY, USA, 1997), ACM Press/Addison-Wesley Publishing Co., pp. 189–198.

[Hop98] Hoppe H.: Efficient implementation of progressive meshes. *Computers & Graphics 22*, 1 (1998), 27–36.

[HSH09] Hu L., Sander P. V., Hoppe H.: Parallel view-dependent refinement of progressive meshes. In *I3D '09: Proceedings of the 2009 symposium on Interactive 3D graphics and games* (New York, NY, USA, 2009), ACM, pp. 169–176.

[JNS08] John Nickölls Ian Buck M. G., Skadron K.: *Scalable Parallel Programming*. Programming guide, UNIVERSITY OF VIRGINIA, 2008.

[Lit08] Littschwager T.: *Neue Grafik-Generation: Alle Details*. Technical report, Chip, Sep 2008. http://www.chip.de/artikel/Neue-Grafik-Generation-Alle-Details_32708718.html (19.08.2009).

[Lue01] Luebke D. P.: A developer's survey of polygonal simplification algorithms. *IEEE Computer Graphics and Applications 21*, 3 (May/Jun 2001), 24–35. http://www.cs.virginia.edu/~luebke/publications/pdf/cg+a.2001.pdf (11.09.2009).

[MB00] Mrohs Bernd C. R.: Progressive meshes - eine einführung. *test* (Jul 2000). http://www.mrohs.com/publications/Bernd%20Mrohs,%20Christian%20Raeck%20-%20Progressive%20Meshes.pdf(29.07.09).

[MGK03] Michael Guthe P. B., Klein R.: Efficient view-dependent out-of-core visualization. *University of Bonn, Institute of Computer Science II* (2003). http://www.uni-marburg.de/fb12/informatik/homepages/guthe/files/guthe-2003-efficient (12.09.2009).

[Nah02] Nahmias J.-D.: Real-time massive model rendering. *University College London* (Sep 2002). http://www.cs.ucl.ac.uk/research/equator/papers/Documents2002/Jean-Daniel_Nahmias/Massive_Model_Rendering.htm (29.07.09).

[NVI07] NVIDIA: *NVIDIA CUDA Compute Unified Device Architecture*. Programming Guide Version 1.0, NVIDIA Corporation, Santa Clara, CA, USA, 2007.

[NVI08] NVIDIA: *NVIDIA GeForce GTX 295*, 2008. http://www.nvidia.de/object/product_geforce_gtx_295_de.html (02.10.2009).

[NVI09] NVIDIA: *OpenCL Programming Guide for the CUDA Architecture*. Programming Guide Version 2.3, NVIDIA Corporation, Santa Clara, CA, USA, 2009.

[OLG*05] Owens J. D., Luebke D., Govindaraju N., Harris M., Krüger J., Lefohn A. E., Purcell T. J.: A survey of general-purpose computation on graphics hardware. In *Eurographics 2005, State of the Art Reports* (Aug 2005), pp. 21–51.

[Paj01] Pajarola R.: Fastmesh: Efficient view-dependent meshing. *Computer Graphics and Applications, Pacific Conference on 0* (2001), 0022.

[PD04] Pajarola R., DeCoro C.: Efficient implementation of real-time view-dependent multiresolution meshing. *IEEE Transactions on Visualization and Computer Graphics 10*, 3 (2004), 353–368.

[Rißß99] Risska V.: *Test: Intel Core i7 920, 940 und 965 Extreme Edition.* Technical report, Computerbase, Sep 1999. http://www.computerbase.de/artikel/hardware/prozessoren/2008/test_intel_core_i7_920_940_965_extreme_edition/ (20.08.2009).

[Tro01] Troger C.: Levels of detail. *Institute of Computer Graphics and Algorithms Vienna University of Technology* (2001). http://www.cg.tuwien.ac.at/courses/Seminar/SS2001/lod/troger_paper.pdf (09.09.2009).

[WIK] WIKIPEDIA: *Octree.* http://de.wikipedia.org/wiki/Octree (29.07.2009).

List of Abbreviations

ALU Arithmetic Logic Unit
BTF Bidirektionalen Textur Funktion
CPU Central Processing Unit
CU Control Unit
CUDA Compute Unified Device Architecture
FLOPs Floating Point Operations Per Second
FPU Floating Point Unit
GPGPU General Purpose Compution on Graphics Processing Unit
GPU Graphics Processing Unit
HLOD Hierarchische Level of Detail
IFS Indexed-Face-Set
LOD Level of Detail
MIMD Multiple Instruction Multiple Data
OpenCL Open Computing Language
OpenGL Open Graphics Library
PCAM Partitionierung Kommunikation Agglomeration Mapping
PM Progressive Meshes
SFU Spezial Funktion Units
SIMD Single Instruction Multiple Data
SIMT Single Instruction Multiple Threads
SLI Scalable Link Interface
SP Streaming-Prozessoren
SM Streaming-Multiprozessoren
TPC Textur Prozessor Clustern
VBO Vertex Buffer Object

List of Figures

List of Tables

List of Algorithms

Listings