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

The topic of this thesis the dynamic insertion of 3D objects, defined in computer assisted design(CAD) Files, into an Unreal Engine program while it is running. Especially important for the project are why this might even a problem and how it can actually be realized. For these purposes an Unreal Engine plug-in was developed which enables such functionality and an additional Unreal Engine program which implements the plug-in and can be used to present and interact with the objects in a multi-user desktop or virtual reality environment.

1.1 Motivation

Virtual reality(VR) is a relatively new field which is constantly seeing a lot of interest and innovation for all the new possibilities in opens up in software development and user interaction. In recent years VR has been used in many companies in various industries such as the engineering, architecture and healthcare and this number keeps on growing. One such company is Inosoft [].
Inosoft is a software development firm in Marburg which was founded in 1993 and has since worked and consulted over two thousand projects for various companies including Viessmann, CSL Behring, Sanofi and many more. They are also very interested in VR and have been working in the field since 2016. Inosoft was also interested in establishing a working relationship with the Phillips University Marburg. As such they reached out with some very interesting projects in the field of VR. Among them was designing and developing a concept to dynamically insert and interact with objects from CAD files in a running Unreal Engine environment.
There are definitely certain scenarios where this could be a very useful tools. As an example, let's take a software Inosoft developed which is used to train workers in a digital production plan while the physical building was being built. This quite a handy tool and has helped quite a bit [**?**]. Slight problems arise when things about the model need to be added or changed. First the changes need to be implemented in Unreal, packaged for standalone use and then redistributed to everyone who needs to use them. It would be a lot simpler if the program could simply open a file and add the new or update objects without ever having to change the version of it.
Another use-case where this could be useful is in collaborative design or presenting 3D models. Instead of having to make the scene and import everything beforehand and distribute this version of the program, simply having a program that can open a file and have the model appear for everyone involved could save a lot of time and effort.
So seeing as there are uses for this technology it makes sense to look into how it could be

done and what the limitations are, as well as looking into why this isn't already officially part of Unreal Engine.

1.2 Goals

The main goal of this work is to develop an efficient and user-friendly plug-in which will make it possible to load 3D objects from the most common CAD formats during the runtime of an Unreal Engine program. Additionally another software will be developed to use the plug-in and allow simple interactions with these objects in a normal desktop window as well as in a virtual reality environment.

Efficiency

The developed plug-in should be capable of handling large amounts of data seeing as the models which can be found in CAD files can be incredibly large, containing thousands or millions of vertices and polygons. If the plug-in were to effect the runtime performance in a significant way, such as causing stutters or freezing the program all together, it would severely worsen the user experience and invalidate the whole point of the program.

Expandability

The field of computer assisted design is very wide and there are countless programs and formats for all the varying use-cases in which it is being used. That is why creating one solution for all of those is incredibly complicated and way out of the scope and possibilities of this project. Instead it is much better to concentrate on creating a simple to use and understand system which can then be further improved upon and adjusted for the concrete cases of clients or projects.

1.3 Thesis Structure

In Chapter 2 the Unreal Engine will be clarified and explained. Seeing as this is both the tool which is being used for development as well as being the software for which the plug-in is being developed, an understanding of how it works and what its limitations are is needed in order to better grasp the project and what problems might arise. It is a rather expansive tool so not everything will be covered, only the more basic aspects and the concrete parts which play a role for this project. Then, in Chapter 3, the plug-in will be analysed, starting of with how the files are parsed and into what sort of form they are transformed in order to be used. After that comes the actual mesh generation mesh, how it is achieved and where extra attention is required. In Chapter 3.3 it will be illustrated in what ways users can interact with the newly created objects, either using mouse and keyboard or a virtual reality headset. In Chapter 4 the developed programs will be presented, evaluated and compared to similar software to see where its strengths and weaknesses are. Lastly in Chapter 5 the reached goals and some possible further projects and improvements will be discussed.

1.4 Related Works

When it comes to this topic there are unfortunately not that many similar works. When it comes generally importing CAD files into Unreal Engine, Datasmith definitely needs to be mentioned.
Datasmith is an official set of tools and plug-ins created by Epic Games, the developers of Unreal Engine, to simplify and streamline the process of importing various CAD formats into the engine []. It is important to note that the main focus of Datasmith is to make the process of transferring a model from a CAD software into the Unreal Engine editor during development smoother and more efficient []. Nonetheless amongst the many features it has it does also contain a plug-in for loading the models in runtime. This plug-in is still in being developed, even upon installation there are clear warnings that the software is still in a beta, so there are some missing features and there also haven't seemingly been many updates to it since the initial research for this project started [].
Outside of Datasmith there are a handful of small plug-ins that can be found which handle this topic, most importantly glTFRuntime [] and Runtime FBX Import []. They were developed by a small team and a single person respectively and are available to be bought in the Unreal Marketplace. The strengths and weaknesses of these tools, as well as Datasmith, will be discussed in further detail later in order better evaluate the programs developed for this project.

2 Unreal Engine

The Unreal Engine is a 3D graphics video game engine, first created for the first person shooter Unreal in 1998 [**?**]. Originally written mostly by Tim Sweeney, the founder of Epic Games, it has since grown an incredible amount and become one of the most popular game engines on the market, only perhaps beaten by Unity []. It has also had many versions since its initial release, first with Unreal Engine 2 in 2002 and then with version 3 in 2006 []. Up until recently Unreal Engine 4, released in 2014, was the latest version but April of 2022 saw the official release of Unreal Engine 5. All of the versions were written in C++ enabling great performance as well as portability, so that the engine is currently supported on a wide range of desktop, console, mobile and even virtual reality platforms [].

In its more than 25 year history the Unreal Engine has been used to create a vast number of incredibly popular and critically acclaimed games such as Fortnite, Hellblade and the Bioshock series, only to name a few. Even though the main use-case has remained video game development, the engine has seen wide adoption in many other industries as well. In film making it can be used to create virtual sets that can be rendered in real time on large LED screens and lighting systems while also tracking around actors and objects using the camera's movement. Epic Games worked with the Industrial Lights and Magic of division Lucasfilm to develop their StageCraft technology [**?**], first used in filming the television show The Mandalorian [**?**]. Outside of these creative fields due to its wast functionalities and ease of use, it has been used to create virtual reality tools to explore building and car designs, as well pharmaceutical drug molecules [**?**].

For the purposes of this project Unreal Engine 4.27 was used and this is the version that will be described unless specified otherwise. Although this technically isn't the latest version and the development of this project started around the same time as version 5 was officially released, there were multiple reason as to why this decision was made. First of all, pretty much any new software release tends to bring with it a number of bugs and quirks which need to be discovered and fixed first. This doesn't always have to be the case but a lot of developers will wait for the software become more ironed out before using it. That is if they even want to use UE5. There are many programs already written in earlier versions of it and not every one of those might truly require the new features UE5 brings with it so the update might not even occur. Also the initial research for the project, which also included learning how Unreal works and how to use it, was done months before the launch.
On the other hand Unreal Engine 4 is a very mature tool which has been used and improved for years now. There are also many sub-versions of it but the decision was

made to use the latest one, 4.27.2, as it should be UE4 at its best and also due to the excellent compatibility between it and earlier version of UE4.

2.1 Unreal Engine Basics

Developing a video game is quite a complicated process and requires various features in order to create a cohesive experience. As Unreal is primarily a game development engine it also has to support many of these functionalities. In total there are more than dozen editors for levels, materials, meshes,physics and user-interface, to just name a few, but for the purposes of this project only a few are of relevance. These are the level, material and blueprint editors.

The level editor is the primary editor where the levels are created and modified by placing, transforming and editing properties of objects. This is also the default screen Unreal shows when creating or opening a project and what that looks like is shown in figure 2.1. As can be seen in the figure, in the centre of the screen is the level itself. Above it is a toolbar for managing project settings, code, plug-ins and as well a play button which can be used to launch the game inside of the editor for testing purposes. On the bottom the content browser which display all of the assets which are part of the project can be found. This includes meshes, materials, code as well as project plug-ins. On the left is a toolbar for placing various built in objects and on the right all of the objects in the current world, as well as details about the currently selected object can be seen. This is also generally the screen where a developer would import any external assets into the engine directly or through one of the engines importer tools.

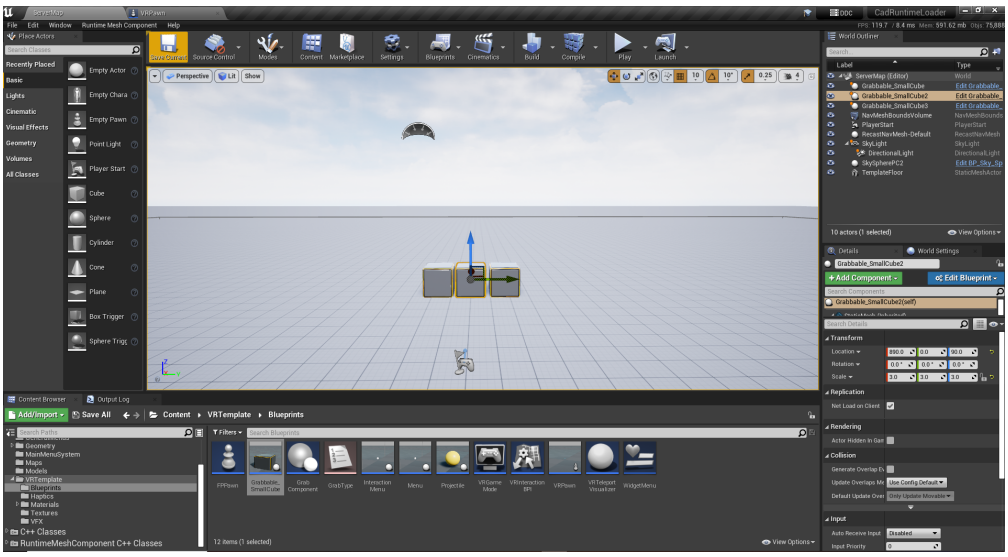

Figure 2.1: Example of what the Level Editor looks like for a project

Actors and Components

In Unreal Engine all of the objects that can be placed inside of a level are called Actors. This includes everything from meshes to particle systems to even the players starting location. This is partially due to Unreal Engines object-oriented nature so having all objects inherit from one base class, in this case Actor, can be quite beneficial.

Pawns and Player Controllers

text

Material Editor

2.2 Blueprints and C++

As already described in 2.1 ...

2.3 Networking

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