

**Czech University of Life Sciences Prague**

**Faculty of Economics and Management**

*Department of Information Engineering*



**Bachelor Thesis**

**MODERN MEANS OF CREATING 3D GRAPHICS  
APPLICATIONS**

**Author: Nzube Victor OKOYE**

© 2015 CULS Prague

## BACHELOR THESIS ASSIGNMENT

Nzube Victor Okoye

Informatics

Thesis title

**Modern means of creating 3D graphics applications**

---

### Objectives of thesis

The aim of this thesis is to describe possibilities of programming 3D applications. The main goal is to analyze and describe available technologies and to demonstrate in practice how a selected technology can be used to implement 3D applications.

### Methodology

Methodology of this thesis is based on analysis and study of various information sources. Based on the synthesis of the gained knowledge a description of available 3D programming technologies will be created. One of the technologies will be chosen and described in more detail and also used for practical demonstration by creating an example application.

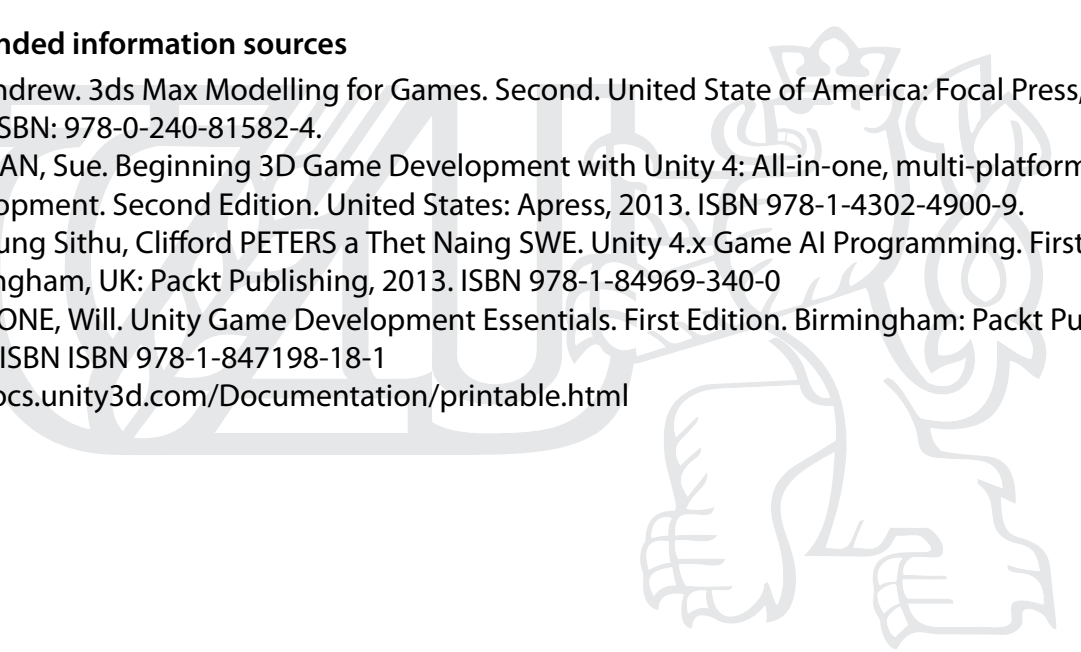
## The proposed extent of the thesis

35-50 pages

---

### Recommended information sources

GAHAN, Andrew. 3ds Max Modelling for Games. Second. United State of America: Focal Press, 2011. ISBN ISBN: 978-0-240-81582-4.

1. BLACKMAN, Sue. Beginning 3D Game Development with Unity 4: All-in-one, multi-platform game development. Second Edition. United States: Apress, 2013. ISBN 978-1-4302-4900-9.
  2. KYAW, Aung Sithu, Clifford PETERS a Thet Naing SWE. Unity 4.x Game AI Programming. First Edition. Birmingham, UK: Packt Publishing, 2013. ISBN 978-1-84969-340-0
  3. GOLDSTONE, Will. Unity Game Development Essentials. First Edition. Birmingham: Packt Publishing, 2009. ISBN ISBN 978-1-847198-18-1
  4. <http://docs.unity3d.com/Documentation/printable.html>
- 

---

### Expected date of thesis defence

2015/06 (June)

### The Bachelor Thesis Supervisor

Ing. Jiří Brožek, Ph.D.

Electronic approval: 10. 11. 2014

**Ing. Martin Pelikán, Ph.D.**

Head of department

Electronic approval: 10. 11. 2014

**Ing. Martin Pelikán, Ph.D.**

Dean

Prague on 04. 03. 2015

## **Declaration**

I declare that I have worked on my bachelor thesis titled “Modern Means of Creating 3D Graphics Applications” by myself and I have used only the sources mentioned at the end of the thesis.

In Prague, .....

.....

Nzube Victor Okoye

## **Acknowledgements**

Big thanks goes to Ing. Jiří Brožek, Ph.D for his advices and for taking the time to supervise my bachelor thesis.

Thanks to Ing. Martin Kozák for the continued help throughout the period of my studies.

# **Modern Means of Creating 3D Graphics Applications**

Moderní prostředky pro tvorbu 3D grafických aplikací

# **Modern Means of Creating 3D Graphics Applications**

## **Summary**

Creating 3D graphics used to be a daunting task and personal computers did not have the processing capacity for handling it; but with the progress in computer architecture- software and hardware, this has become much easier. This thesis deals with 3D graphics creation starting from its basic definition to the underlying technology and finally the current available means for creating 3D graphics application.

The first part of the thesis deals with the well known application programming interface. The second part highlights the system software available to users today for creating 3D graphics. The third part is a practical demonstration of 3D graphics applications that can be created using these modern system software.

## **Keywords**

3D graphics, 3D model, Unity Game Engine, Game development, C#, Artificial Intelligence, 3ds Max

# Moderní prostředky pro tvorbu 3D grafických aplikací

## Souhrn

Tvorba 3D grafiky bývala dříve náročným úkolem, protože osobní počítače neměly dostatečný výpočetní a grafický výkon. Vývoj výpočetní techniky a softwarových aplikací práci s 3D grafikou značně zjednodušil. Tato práce se zabývá tvorbou 3D grafiky od úplných základů a definic, až po současné programovací techniky a moderní nástroje pro tvorbu 3D aplikací.

První část pojednává o nejrozšířenějších systémových rozhraních (API) pro práci s grafickým hardware. Druhá část představuje nejpoužívanější aplikace pro vytváření 3D grafiky. Ve třetí části se zaměříme na praktickou ukázkou modelování 3D grafiky za použití moderního software.

## Klíčová slova

3D grafika, 3D model, Unity Game Engine, vývoj her, C#, umělá inteligence, 3Ds MAX



# Table of Contents

Modern Means of Creating 3D Graphics Applications .....	1
Moderní prostředky pro tvorbu 3D grafických aplikací .....	1
Modern Means of Creating 3D Graphics Applications .....	2
Summary .....	2
Keywords .....	2
Moderní prostředky pro tvorbu 3D grafických aplikací .....	3
Souhrn .....	3
Klíčová slova .....	3
Table of Contents .....	4
1. Introduction .....	7
2. Objectives and Methodology .....	8
2.1 Objectives .....	8
2.2 Overview of the Thesis .....	8
2.3 Methodology .....	9
3. Literature Review .....	10
3.1.1 What is 3D? .....	10
3.1.2 What is 3D Computer Graphics? .....	10
3.2 Application Programming Interface .....	11
3.2.1 Definition of Application Programming Interface .....	11
3.2.2 3D Graphics Application Programming Interface .....	12
3.3 Modern Means and Frameworks .....	26
3.3.1 Overview .....	26
3.3.2 Frameworks for Creating 3D Models .....	26
3.3.3 Frameworks for creating 3D games .....	34

4.	Design and Implementation .....	39
4.1	Game One .....	40
	Game Description .....	40
	Implementation .....	40
	Setting up the Project and Player in Unity.....	41
	Build Platform.....	41
	Setting up Game Environment and Elements .....	42
	Setting up the Player Movement.....	43
	Creating Weapon for the Spaceship and Enemy .....	44
	Wrapping up the build .....	46
	Build and Deployment .....	48
4.2	Game Two.....	48
	Game Description .....	48
	Implementation .....	49
	Build and Deployment .....	50
4.3	Simulation of Game Environment .....	51
	Terrain Creation and Painting.....	51
	Adding Sky Feature and Fog .....	53
	Adding a Lake.....	54
	Character Controller .....	54
	Build and Deployment .....	54
5.	Conclusion .....	55
5.1	Discussion.....	55
5.2	Conclusion .....	55
	Bibliography .....	57
	List of Figures .....	60

List of Tables .....61

# 1. Introduction

Computer graphics is a broad field; to understand it, you need information from perception, physics, mathematics and engineering. (Hughes, et al., 2014)

In computer graphics, objects exist only in the memory of the computer. They have no physical form they are just long lists of numbers and mathematical formulas and little electrons running around inside the computer. (Giambruno, 2002)

The term computer graphics is believed to have been coined by William Fetter, who was a graphic designer for Boeing Aircraft Co. in 1960. He used the term to describe what he was doing in Boeing at that time. ([www.osu.edu](http://www.osu.edu)) This simply suggests that computer graphics pre-dates 1960. But due to the processing capacity of the existing computers at that time, computer graphics was mostly done in two dimensions which are easily referred to 2D graphics.

The development of more advanced computers with higher processing capacity has led to innovations in the field of computer graphics which are based on various academic researches. As a result, there are various developments and implementations of computer algorithms for making computer graphics that are more realistic; and it has kept evolving closer to what is seen in real life.

Depiction of such realistic images with length, width and depth is what is called three dimensional graphics which is easily referred to as 3D graphics.

Nowadays, 3D graphics has a broad range of uses in all sorts of fields including: film making, multimedia, Games, Design, Web, Virtual Reality, Research and Engineering etc.

## **2. Objectives and Methodology**

### **2.1 Objectives**

The goal of this thesis is to discuss and subsequently implement 3D graphics applications using the modern 3D graphics frameworks available to users today.

The most striking aspect of graphics in our everyday lives is the 3D imagery being used in video games and special effects in the entertainment industry and advertisements. (Giambruno, 2002)

Based on this assertion, implementation of the modern 3D graphics application for this work will be done using a video game. This is because the concept of video game design highlights all the important aspects of modern 3D graphics frameworks which include: modelling, rendering, mapping, texturing, lighting, camera, animation and to some extent scripting and/or coding.

### **2.2 Overview of the Thesis**

The thesis is divided into the following:

Chapter 3- Literature review and theoretical background of 3D graphics will be discussed with emphasis on various application programming interface- APIs.

Also, modern frameworks, that is, system software for making 3D graphic applications for the processes involved in the design of video games will be discussed.

Chapter 4- Implementation of 3D graphics application will be done using one of the 3D frameworks discussed in the previous chapter. This implementation will be done by creating a simple video game. The creation process will highlight all the important aspects of modern 3D graphics applications.

Chapter 5- “Wraps up” the thesis by discussing limitations of existing framework that was used.

## **2.3 Methodology**

The thesis is in two parts: theoretical part which is made of desk research including: summary, synthesis and collation of relevant literature and resources.

The second part being the practical part is based on the use of 3D models and meshes either self-authored or downloaded under free licence to design a video game. It will buttress the simplicity of a once complicated process using the modern tools and frameworks described in the theoretical part.

## **3. Literature Review**

Computer graphics is an amazing technology success story. The basic ideas, representations, algorithms and hardware approaches were forged in the 1960's and 1970's and developed to maturity in the following two decades. By the mid 90's computer graphics techniques were reasonably mature, but their impact was still somewhat restricted to "high end" applications such as scientific visualization on super-computers, and expensive flight simulators. (Gortler, 2012)

The previous decade has finally seen the mass commodification of graphics. Every modern personal computer is capable of producing high quality computer generated images, mostly in the form of video games, special effects and virtual-life environments. The entire animation industry has been transformed from its high-end (e.g., Pixar films) down to daily children's television. For live-action movies, the field of special effects has been completely revolutionized; viewers nowadays don't flinch when they see the most amazing computer generated special effects, it is simply expected and very realistic. (Gortler, 2012)

This chapter will focus on the application programming interface and technologies that laid the foundation for creating modern 3D graphics applications. But first, key terminologies that will be used throughout this work will be defined.

### **3.1.1 What is 3D?**

3D stands for three-dimensional, that means that things have three dimensions: namely width, height, and depth. If you look around the room, everything you see is three dimensional- the chair, the desk, the wall, everything. (Giambruno, 2002)

### **3.1.2 What is 3D Computer Graphics?**

3D computer graphics are works of graphic art that were created with the aid of digital computers and specialized 3D software. In general, the term may also refer to the process of creating such graphics, or the field of study of 3D computer graphic techniques and its related technology. (Hees , 2006)

The two definitions of 2D and 3D computer graphics above pose an important question: if everything in the room is three-dimensional, then how come the 3D graphics we have seen in the computer screens are flat? Basically, 3D computer graphics should be referred to as “two-dimensional representation of three dimensional objects”. 2D, means two dimensional, or having only width and height, but no depth; in other word, flat. (Giambruno, 2002)

Our day-to-day interactions with home computers, cell phones, movies, games etc., are based on computer graphics. Perhaps they are less visible in part because they are more successful: The best interfaces are the ones not noticed. It is tempting to say that “2D graphics” is simpler and that 3D graphics is just a more complicated version of the same thing. But many of the issues in 2D graphics like how best to display images on a screen made of a rectangular grid of light-emitting elements, for instance, or how to construct effective and powerful interfaces are just as difficult as those found in making pictures of three-dimensional scenes but with the addition of perspective and much more mathematical input and consideration. (Hughes, et al., 2014)

## **3.2 Application Programming Interface**

### **3.2.1 Definition of Application Programming Interface**

An application programming interface- API is a software program that facilitates interaction with other software programs. APIs allows programmers to interact with an application using a collection of callable functions. The goal of APIs is to allow programmers to write programs that will not cease to function if the underlying system is upgraded. (Cory, Janssen: [www.techopedia.com](http://www.techopedia.com))

API is language dependent or independent:

**Language Dependent:** This means it is only available by using the syntax and elements of a particular language, making it more convenient to use.

**Language Independent:** This means the API is written to be called from several programming languages. (Cory, Janssen: [www.techopedia.com](http://www.techopedia.com))



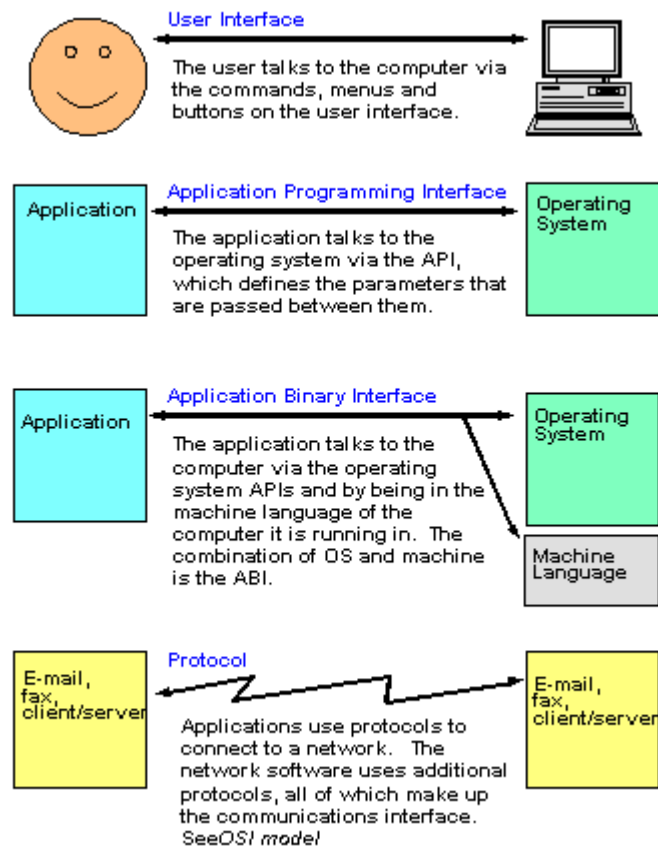


Figure 1- A Comparative Description of an API

(Source: <http://www.pcmag.com/encyclopedia/term/37856/api>)

### 3.2.2 3D Graphics Application Programming Interface

The introduction of an industry-standard 3D application programming interface to mass-market operating systems such as Microsoft Windows and the Macintosh OS X has some exciting repercussions. With hardware acceleration and fast personal computer microprocessors now commonplace, 3D graphics have become standard components of consumer and business applications, not only of games and scientific applications. (Wright , et al., 2010) The APIs has become a vital part of graphic also for the manufacturers as they have provided a way for programmers to access the hardware in an abstract way, while still taking advantage of the special hardware of different graphic cards from different manufacturers. (Hees , 2006)

These 3D APIs Include:

- OpenGL
- Direct3D
- X3D
- Mantle
- RenderMan
- Glide
- WebGL

Among all the APIs listed above, OpenGL and Direct3D are the most common in computers and to some extent, they set the current industry standard. This thesis therefore will focus mainly on OpenGL and Direct3D in detail and then, will briefly highlight the other APIs listed.

### 3.2.2.1 OpenGL



Figure 2- OpenGL Logo

(Source: <https://www.khronos.org/news/logos>)

#### **Overview**

OpenGL which stands for “Open Graphics Library” is an API (Application Programming Interface) to graphics hardware. The API consists of a set of several hundred procedures and functions that allow a programmer to specify the shader programs, objects, and operations involved in producing high-quality graphical images, specifically colour images of three-dimensional objects. (Segal, et al., 2015)

The above definition of OpenGL from the official specification is generic. In order to understand it better, a direct definition from different points of view is necessary.

As such, from a programmer’s point of view, OpenGL is a set of commands that allow the specification of shader programs or shaders, data used by shaders, and state controlling

aspects of OpenGL outside the scope of shaders. Typically the data represent geometry in two or three dimensions and texture images, while the shaders control the geometric processing, rasterization of geometry and the lighting and shading of fragments generated by rasterization, resulting in rendering geometry into the frame buffer. (Segal, et al., 2015)

To the implementor, OpenGL is a set of commands that control the operation of the graphic processing unit- GPU. Modern GPUs accelerate almost all OpenGL operations, storing data and frame buffer images in GPU memory and executing shaders in dedicated GPU processors. However, OpenGL may be implemented on less capable GPUs, or even without a GPU, by moving some or all operations into the host CPU. The implementor's task is to provide a software library on the CPU which implements the OpenGL API, while dividing the work for each OpenGL command between the CPU and the graphics hardware as appropriate for the capabilities of the GPU. (Segal, et al., 2015)

In summary, it is a 3D graphics and modeling library that is highly portable and very fast. (Wright , et al., 2010) Despite its usefulness, it is merely a software library for accessing features in graphics hardware. (Shreiner, et al., 2013)

OpenGL has a wide range of uses including:

- Computer-aided design- CAD
- Film special effects and animation
- Architectural application
- Virtual reality
- Scientific visualization
- Information visualization
- Flight simulation
- Video game development

## **History**

OpenGL was developed by Silicon Graphics Incorporated.

Silicon Graphics Incorporated, SGI specializes in the creation of specialized graphics hardware and software. By the early 1990's, SGI had become the world leader in 3D graphics, and its programming API, IrisGL, had become a defacto industry standard, overshadowing the open-standards-based PHIGS. (Hees , 2006) While Silicon Graphics Inc.'s IrisGL was hugely successful, key market players like Sun, Hewlett-Packard and IBM introduced a number of good 3D hardware that were supported by their own proprietary extensions to PHIGS.

By the early 1990's, 3D graphics hardware technology was fairly well understood by a large number of competitors and was no longer a discriminating factor in computer systems purchases. (Hees , 2006) Silicon Graphics Inc. thereafter sought to develop an open standard for the market but making IrisGL the standard was seemingly impossible because not only was IrisGL for graphics, it also included other APIs like windowing, keyboard and mouse.

The result of Silicon Graphic Inc.'s open standard is the OpenGL.

OpenGL standardised access to hardware, and pushed the development responsibility of hardware interface programs, sometimes called device drivers, to hardware manufacturers and delegated windowing functions to the underlying operating system. With so many different kinds of graphic hardware, getting them all to speak the same language in this way had a remarkable impact by giving software developers a higher level platform for 3D-software development. (Hees , 2006)

In 1992, SGI led the creation of the OpenGL architectural review board- OpenGL ARB, the group of companies that would maintain and expand the OpenGL specification for years to come. As of April 2006, the voting members of the OpenGL ARB include 3D hardware manufacturers: Silicon Graphics Inc., 3Dlabs, ATI Technologies, NVIDIA and Intel, and computer manufacturers: IBM, Apple Computer, Dell, and Sun Microsystems. Microsoft, one of the founding members, left in March 2003. Aside from these corporations, each year many other companies are invited to be part of the OpenGL ARB for one-year term. With so many companies involved with such a diverse set of interests,

OpenGL has become very much a general-purpose API with a wide range of capabilities. (Hees , 2006)

According to current plan, control of OpenGL has been passed to the Khronos Group which started around the end of 2006. This was done in order to improve the marketing of OpenGL, and to remove barriers between the development of OpenGL and OpenGL ES which stands for OpenGL for Embedded Systems also known as GLES. (Hees , 2006)

### **OpenGL Extension**

OpenGL allows vendor innovation through its extension mechanism. This mechanism works in two ways. First, vendors can add new functions to the OpenGL API that developers can use. Second, new tokens can be added that will be recognized by existing OpenGL functions. Making use of new tokens is simply a matter of adding a vendor-supplied header file to your project. Vendors must register their extensions with the OpenGL Working Group- a subset of the Khronos Group, thus keeping one vendor from using a value used by someone else. Conveniently, there is a standard header file *glxext.h* that includes these extensions. (Wright , et al., 2010)

With the OpenGL extension mechanism, gone are the days when games would be recompiled for a specific graphics card.

The extension name helps to identify which vendor created and supports a given extension. Each extension has a three-letter prefix that identifies the source of the extension. The table below provides a sampling of extension identifiers:

<b>Prefix</b>	<b>Vendor</b>
SGL_	Silicon Graphics
ATI_	ATI Technologies
AMD_	Advanced Micro Devices
NV_	NVIDIA
IBM_	IBM
WGL_	Microsoft
EXT_	Cross-Vendor
ARB_	ARB Approved

**Table 1- A Sampling of OpenGL Extension Prefixes (Wright , et al., 2010)**

### 3.2.2.2 Direct3D



**Figure 3- Microsoft Logo**

(Source: <https://www.microsoft.com>)

#### **Overview**

Direct3D is part of Microsoft's DirectX.

DirectX is an SDK software development kit composed of a collection of COM libraries. These libraries provide a variety of functions and classes to make the process of developing games and multimedia applications simpler for the developer. More specifically, DirectX is a collection of smaller APIs which includes Direct3D, DirectShow, DirectInput, DirectSound and DirectPlay. (Thorn, 2005)

#### **What is Direct3D?**

As the name implies, Direct3D is a 3D API. That is, it contains many commands for 3D rendering, but contains few commands for rendering 2D graphics. ([www.microsoft.com](http://www.microsoft.com))

According to another official definition from Microsoft, Direct3D is a low-level API that you can use to draw triangles, lines, or points per frame, or to start highly parallel operations on the GPU. Direct3D in essence:

- Hides different GPU implementations behind a coherent abstraction.
- Is designed to drive a separate graphics-specific processor. Newer GPUs have hundreds or thousands of parallel processors.
- Emphasizes parallel processing. You set up a bunch of rendering or compute state and then start an operation. You don't wait for immediate feedback from the operation. You do not mix CPU and GPU operations. ([www.microsoft.com](http://www.microsoft.com))

In summary, it is the component of the DirectX API dedicated to exposing 3D graphics hardware to programmers on Microsoft platforms including PC- Windows 95 and above, Xbox and Xbox360 console systems, and mobile devices. It is a native API allowing you to create not only 3D graphics for games, scientific and general applications, but also to

utilize the underlying hardware for General-purpose computing on graphics processing units (GPGPU). (Stenning, 2014)

## **History**

The history of Direct3D is rather a brief one, because prior to its first version, Microsoft had the DirectX in place but in February 1995, Microsoft bought a company called RenderMorphics which was started in 1992 by Servan Keondjian. Before the company was bought by Microsoft, it was a 3D graphics API development company and their API was called Reality Lab, which was used in medical imaging and CAD software. Two versions of this API were released.

By buying the company, they brought bringing Servan Keondjian on board to implement a 3D graphics engine for Windows 95. This resulted in the first version of Direct3D. (Hees , 2006)

From then on, future releases of DirectX offered new versions of the Direct3D with visible improvements. The current version is the Direct3D 11 which has more features including: multithreading, compute shader, dynamic shader linking, tessellation etc.

## **Other DirectX Related Tools**

### **D3DX**

This is a library of tools designed to perform common mathematical calculations and several more complicated tasks, such as compiling or assembling shaders used for 3D graphic programming. It also includes several classes that simplify the use of 3D-models and, for example, particle systems. D3DX is provided as a dynamic link library (DLL). D3DX comes with DirectX. (Hees , 2006)

### **DXUT**

This is also called the sample framework.

DXUT is a layer built on top of the Direct3D API. The framework is designed to help the programmer spend less time with mundane tasks, such as creating a window, creating a device, processing Windows messages and handling device events. (Hees , 2006)

### 3.2.2.3 X3D

X3D is a royalty-free open standard file format and run-time architecture to represent and communicate 3D scenes and objects using XML. It is an ISO ratified standard that provides a system for the storage, retrieval and playback of real time graphics content embedded in applications, all within an open architecture to support a wide array of domains and user scenarios. It has a rich set of componentized features that can be tailored for use in engineering and scientific visualization, CAD and architecture, medical visualization, training and simulation, multimedia, entertainment, education, and more. ([www.web3d.org](http://www.web3d.org))

Furthermore, the development of real-time communication of 3D data across all applications and network applications has evolved from its beginnings as the Virtual Reality Modeling Language (VRML) to the considerably more mature and refined X3D standards. ([www.web3d.org](http://www.web3d.org))

#### **X3D Features**

**XML Integrated:** It is cross-platform, usable with web services, distributed networks, and inter-application model transfer

**Componentized:** It allows lightweight core 3D run-time delivery engine

**Extensible:** It allows components to be added to extend functionality for vertical market applications and services

**Profiled:** It has standardized sets of extensions to meet specific application needs

**Evolutionary:** It is easy to update and preserves VRML97 content as X3D

**Broadcast/Embedded Application Ready:** It runs from mobile phones to supercomputers

**Real-Time:** Its graphics are high quality, real-time, interactive, and include audio and video as well as 3D data.

**Well-Specified:** It makes it easier to build conformant, consistent and bug-free implementations ([www.web3d.org](http://www.web3d.org))



### 3.2.2.4 Mantle

Mantle is a graphics API that was developed by AMD in 2013.

It is a low-level API targeted at 3D video games. It was designed as an alternative to Direct3D and OpenGL, mainly for personal computers but the Graphics Processing Unit of PlayStation 4 and Xbox one are also supported.

According to AMD, from a game developer's point of view, creating games for the PC has never been especially efficient. With so many combinations of hardware possible in a personal computer, it's not practical to create specialized programming for every possible configuration. What Mantle does instead is write simplified code that gets translated on-the-fly into something the computer can work with. (www.amd.com)

#### What Mantle Does

- It lets applications speak directly to the Graphics Core Next Architecture
- A Graphics Processing Unit- GPU or Accelerated Processing Unit- APU enabled with the Graphics Core Next architecture (www.amd.com)

### 3.2.2.5 RenderMan

RenderMan Interface Specification is a rendering API. What this means in practice is that RenderMan defines how animation and modeling software should talk to rendering software. (Stephenson, 2007) The figure below describes the how the RenderMan API works.

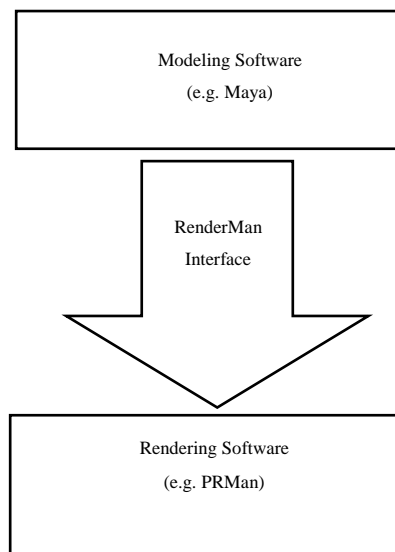


Figure 4- A Rendering API (Stephenson, 2007)

It was developed in 1988 by Pixar Animation Studios as an open API for 3D scenes and digital photorealistic images.

RenderMan was developed out of a concern by Pixar that a wide range of users as possible should be able to make use of their system, and hence they entered into discussion with other major graphics companies. The result of these negotiations was the publication of the RenderMan standard. (Stephenson, 2007)

### **3.2.2.6 Glide**

Glide was developed by 3dfx Interactive as a 3D graphics API for their product Voodoo Graphics 3D accelerator card.

Glide is not a full featured graphics API such as OpenGL. It does not provide high level 3D graphics operations such as transformations, display list management, or light source shading rather, Glide specifically implements only those operations that are natively supported by the Voodoo Graphics hardware. In general, Glide does not implement any functions that do not directly access a Voodoo Graphics subsystem's memory or registers. (3Dfx Interactive, Inc, 1997)

### **3.2.2.7 WebGL**

WebGL is a royalty-free, cross-platform API that brings OpenGL ES 2.0 to the web as a 3D drawing context within HTML, exposed as low-level Document Object Model interfaces. It uses the OpenGL shading language- GLSL ES, and can be cleanly combined with other web content that is layered on top or underneath the 3D content. It is ideally suited for dynamic 3D web applications in the JavaScript programming language, and will be fully integrated in leading web browsers. (Parisi, 2012)

WebGL is developed and maintained by the Khronos Group, the standards body that also governs OpenGL. (Parisi, 2012) The first version of WebGL was released in March 2011.

#### **Main Features of WebGL**

- It is accessed exclusively through a set of JavaScript programming interfaces
- It is based on OpenGL ES 2.0
- It combines with other web content
- It is built for dynamic web applications

- It is cross-platform
- And like all open web specifications, WebGL is free to use (Parisi, 2012)

### **3.2.2.8 Comparison of OpenGL and Direct3D** [Section is referenced from (Hees , 2006)]

OpenGL and Direct3D are the two main competing standards in the 3D graphics market; both API are very viable alternatives for implementing 3D graphics. But usually, OpenGL is often compared to DirectX. DirectX is a family of game technology APIs from Microsoft that includes Direct3D. (Wright , et al., 2010)

Direct3D is a proprietary standard owned by Microsoft and is used extensively on the Windows platform for games, and variants of Direct3D are used on their Xbox gaming console platform and some Windows mobile devices. In the early days of Direct3D, the API was very difficult to use, substantially lacking in features compared to OpenGL, and suffered from some inherent software inefficiencies. (Wright , et al., 2010) Presently, Direct3D is a usable and well-documented API that is quite popular among game programmers interested solely in Microsoft platforms.

OpenGL on the other hand, is quite popular among Windows game developers and is the overwhelming choice for software developers making nongame 3D applications such as the visual simulation industry, content creation tools, scientific visualization, business graphics etc. On the non-Microsoft platforms, such as the Mac OSX or iPhone, Linux- not just desktop, as majority of handheld smart phone devices use a UNIX variant, or Sony and Nintendo gaming devices, OpenGL or an OpenGL-like API is the de facto standard. (Wright , et al., 2010)

The main differences between OpenGL and Direct3D can be summarised under the following heading:

#### **Portability**

Direct3D is only implemented on Microsoft's Windows family of operating systems, including the embedded versions. Several partially functional ports of the Direct3D API have been implemented by Wine, a project to port common Windows APIs to Linux, but this work is difficult due to the dependency of DirectX as a whole on many other components of the Windows operating systems. Microsoft once started development on a port to Apple Computer's Mac OS, but later abandoned this project.

OpenGL, on the other hand, has implementations available across a wide variety of platforms including Microsoft Windows, Linux, UNIX-based systems, Mac OS X and game consoles by Nintendo and Sony, such as the PlayStation 3 and Wii. With the exception of Windows, all operating systems that allow for hardware-accelerated 3D graphics have chosen OpenGL as the primary API. Even more operating systems have OpenGL software renderers.

Secondly, Direct3D is built upon Microsoft's COM. The use of this framework means that C++ code is a little unusual. The plus side of using COM is that you can use the same API in any COM-aware language. These include Visual Basic and Visual Basic Script etc.

OpenGL on the other hand is a specification based on the C programming language. It is built around the concept of a finite state machine, though more recent OpenGL versions have transformed it into much more of an object based system. Though the specification is built on C, it can be implemented in other languages as well.

Finally, Direct3D is designed to be a 3D hardware interface. The feature set of Direct3D is derived from the feature set of what hardware provides.

OpenGL, on the other hand, is designed to be a 3D rendering system that may be hardware accelerated. As such, its feature set is derived from that which users find useful.

### **Performance**

OpenGL was formerly considered to be lagging behind in terms of performance. This assertion was based on the very high cost of graphics accelerators in that consumer market were using software renderers implemented by Microsoft for both Direct3D and OpenGL. Microsoft had marketed Direct3D as faster based on in-house performance comparisons of these two software libraries. The performance deficit was blamed on the rigorous specification and conformance required of OpenGL.

With the introduction of CosmoGL in 1996 by Silicon Graphics Incorporated at the Special Interest Group on Computer Graphics conference- SIGGRAPH, the perception was changed. Silicon Graphics Incorporated challenged Microsoft with CosmoGL, their own optimized Windows software implementation of OpenGL which in various demos matched or exceeded the performance of Direct3D.

For Silicon Graphics Inc., this was an important milestone as it showed that OpenGL's poor software rendering performance was due to Microsoft's inferior implementation, and not to design flaws in OpenGL itself.

On the other hand, Direct3D 9 and below have a particular disadvantage with regard to performance. Drawing a vertex array in Direct3D requires that the CPU switch to kernel-mode and call the graphics driver immediately. While in OpenGL, this flaw does not exist mainly because an OpenGL driver has portions that run in user-mode, can perform marshalling activities to limit the number of kernel-mode switches and batch numerous calls in one kernel-mode switch.

In effect, the number of vertex array drawing calls in a Direct3D application is limited to the speed of the CPU, as switching to kernel-mode is a fairly slow and CPU intensive operation.

Direct3D 10 and above allows portions of drivers to run in user-mode, thus allowing Direct3D 10 and above applications to overcome this performance limitation.

Aside from this, Direct3D and OpenGL applications have no significant performance differences.

### **Structure**

OpenGL was originally designed for then powerful SGI workstations. It included a number of features that are only useful for workstation applications. Today, workstations and consumer machines use the same architectures and operating systems, and so modern incarnations of the OpenGL standard still include these features, although only special workstation-class video cards accelerate them. OpenGL is designed as a feature rich API regardless of hardware support. The specification often drives the implementation of hardware acceleration for these features. For example, when the GeForce 256 graphics card came out in 1999, games like Quake III Arena could already benefit from its acceleration of Transform & Lighting, because the API was designed to provide this feature.

On the other hand, Direct3D developers had to wait for the next version of Direct3D to be released and rewrite their games to use the new API before they could take advantage of hardware-based transform and lighting.

## **Extensions**

The OpenGL extension mechanism is probably the most heavily disputed difference between the two APIs.

OpenGL includes a mechanism where any driver can advertise its own extensions to the API, thus introducing new functionality such as blend modes, new ways of transferring data to the GPU, or different texture wrapping parameters. This allows new functionality to be exposed quickly, but can lead to confusion if different vendors implement similar extensions with different APIs. Many of these extensions are periodically standardized by the OpenGL Architecture Review Board, and some are made a core part of future OpenGL revisions.

On the other hand, Direct3D is specified by one vendor- Microsoft, leading to a more consistent API, but denying access to vendor-specific features.

## **Users**

OpenGL has always seen more use in the professional graphics market than DirectX while DirectX is used mostly for computer games.

At one point many professional graphics cards only supported OpenGL, however, nowadays all the major professional card manufacturers like: NVidia, ATI Technologies and Matrox support both OpenGL and Direct3D on Microsoft Windows.

The reason for OpenGL's advantage in the professional market is partly historical. Many professional graphics applications for example, Softimage 3D and Alias PowerAnimator were originally written in IrisGL for high-end Silicon Graphics Incorporated workstations then ported to OpenGL. Even long after Silicon Graphics Incorporated no longer dominated the market, many professional graphics cards only supported OpenGL.

Finally, the choice between OpenGL and Direct3D usually boils down to liking or disliking Direct3D's object-oriented COM (Component Object Model) methodology versus OpenGL's state machine abstraction. (Wright , et al., 2010) In the end, which ever API is chosen, the desired functionalities are essentially the same. (Horst, 2009)

## **3.3 Modern Means and Frameworks**

This subchapter is focused on the modern system software available to end users for the creating of 3D graphics applications. It deals with the system software for making models and meshes for video games and also software for designing video games because by focusing on these two subjects, the key aspects of modern 3D graphics applications will be properly covered.

The frameworks that will be discussed in this subchapter are divided into two sections:

- Frameworks for creating 3D models *and*
- Frameworks for creating 3D games

### **3.3.1 Overview**

The term framework here refers to system software or computer program.

3D computer graphics software refers to programs used to create 3D computer-generated imagery- CGI (Giambruno, 2002), Computer-aided design- CAD, basic 3D geometric objects and models, animations and artificial intelligent 3D animations.

What these 3D graphics system software does is that it takes advantage of the application programming interface- APIs discussed in the previous section and various low-level programming language to build computer programs that are either for single-platform or in most case cross-platform thereby presenting users a simplified means to create 3D graphics applications with little or no programming knowledge. Some of the 3D graphics software still presents users the means to build more complex 3D graphics application with high level of interactivity and artificial intelligence through scripting, coding and various extensions.

### **3.3.2 Frameworks for Creating 3D Models**

Since the development of computers with higher processing capability and more powerful graphics card, a lot of new proprietary 3D system software has surfaced in the software market. Some of which are more robust than others. This part of the thesis highlights the key system software for creating 3D models and meshes. The main system software includes:

### 3.3.2.1 Autodesk 3ds Max

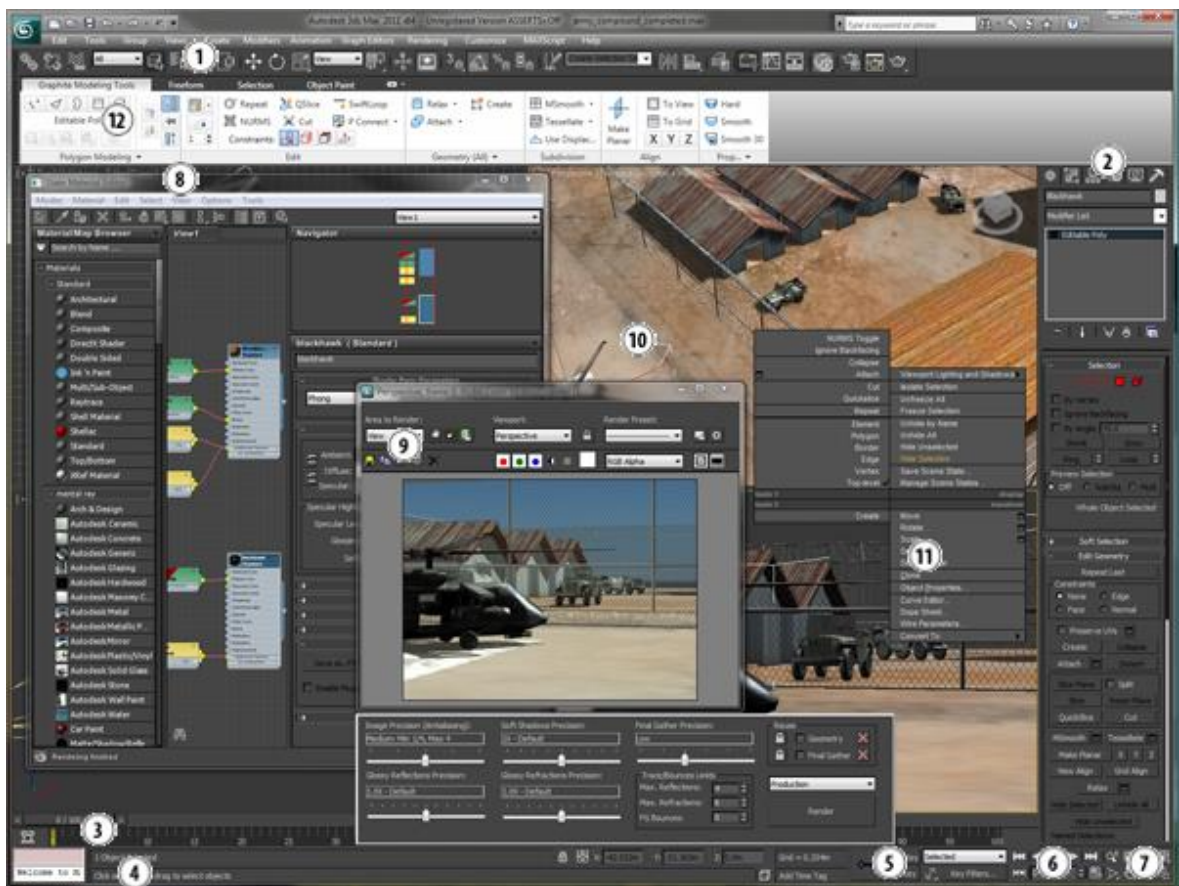


Figure 5- User Interface for Autodesk 3ds Max 2014 (www.autodesk.com)

#### Overview

Autodesk 3ds Max is a 3D modeling software that provides a comprehensive modeling, animation, simulation, and rendering solution for games, film, and motion graphics artists. 3ds Max delivers efficient new tools, accelerated performance, and streamlined workflows to help increase overall productivity for working with complex, high-resolution assets. (www.autodesk.com)

#### Developer

3ds Max is developed by Autodesk Media and Entertainment with its headquarters in Montreal, Quebec Canada. It is one of the key market players in the animation, visual effects and 3D graphics software industry.



## **Version History**

The original version of 3ds Max was created by Gary Yost in 1990 for the MS Dos Platform as 3D Studio. In 1996, the first version for the Microsoft Windows was released with a different name 3D studio Max. Following the purchase of the product in 1999 by Autodesk, there have been a few other renaming and re-branding of the software.

It was first renamed Discreet 3ds max in 2000 and in 2005, with the release of version 8, it was again renamed Autodesk 3ds Max 8. The current naming pattern which has the year of release as a suffix instead of the version number was adopted in 2008. The current version of the software is Autodesk 3ds Max 2015.

## **Supported Platform**

3ds Max is available only on the Microsoft Windows operating system platform. It was available for both 32 bit and 64 bit PC until 2014. Since 2014, 3ds Max is only available for 64 bit PCs.

## **Licencing**

3ds Max is a Trialware meaning that it is a commercial product but available for free limited time trial period after which the user is expected to buy a commercial licence or the right to use the product is discontinued. Autodesk also offers along with the trial version, a 3-year free licence for students.

### 3.3.2.2 Autodesk Maya

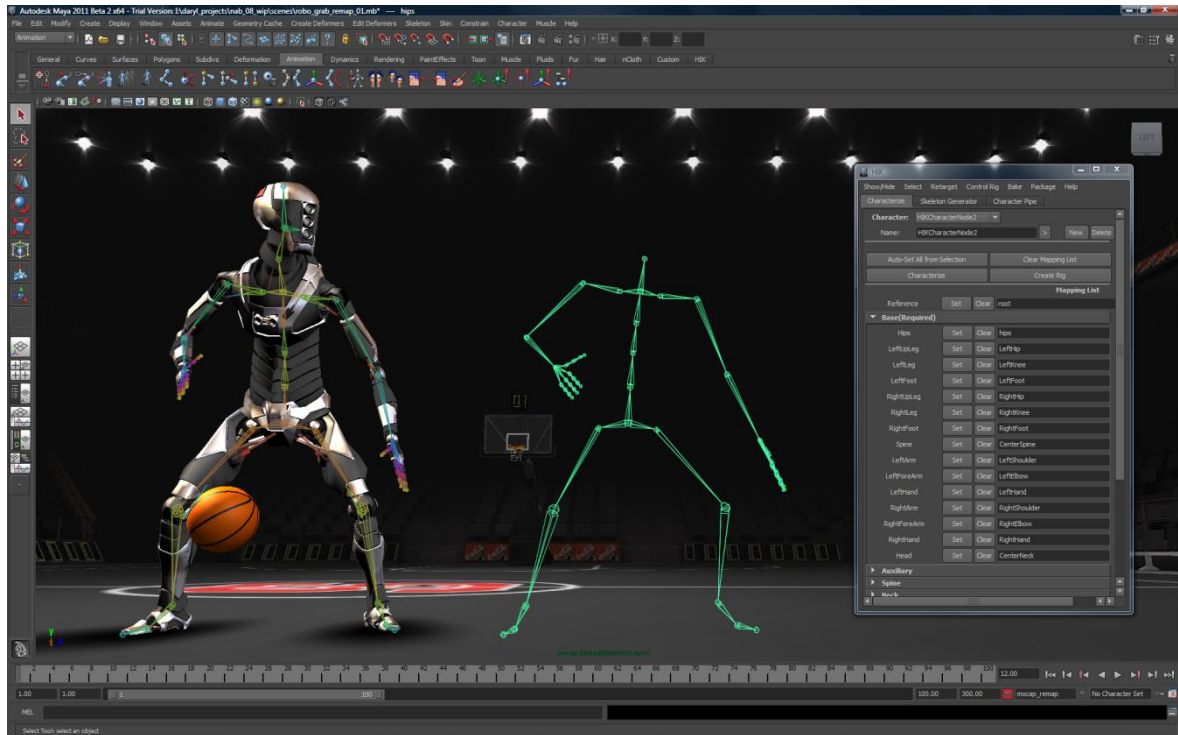


Figure 6- Autodesk Maya User Interface (www.autodesk.com)

#### Overview

Autodesk Maya is an integrated 3D modeling, animation, visual effects, and rendering solution. Maya addresses the challenges faced by everyone from artists handling an entire project alone or with a small team to chief technology officers managing a complex production pipeline. From pre-visualizing feature films before a single frame is shot to producing spectacular finished effects in games or stereoscopic projects. (www.alternativeto.net)

Maya provides creative feature set on a highly extensible production platform and a high-end character and effects toolsets along with increased productivity for modeling, texturing, and shader creation tasks. (www.autodesk.com)

## Developer

Maya was first developed by Alias Systems Corporation which was headquartered in Toronto Canada but as its name suggests, it is currently owned and developed by Autodesk Inc. after it bought it in 2005.

## Version History

The first version of Autodesk Maya was released with the dominant product name Maya in 1998. It was written in C++, MEL- Maya Embedded Language and Python. Since its purchase by Autodesk in 2005, Autodesk has been releasing a yearly version of the software. The current version is Autodesk Maya 2015.

## Supported Platforms

Unlike 3ds Max, Maya is available on multiple platforms including Microsoft Windows, Apple's OS X, Linux- Red Hat Enterprise Linux, Fedora and CentOS.

## Licencing

Autodesk Maya has the same licencing as the 3ds Max, it is a commercial software that is also available as a Trialware and as a free software for students with a 3-year licence.

### 3.3.2.3 Blender

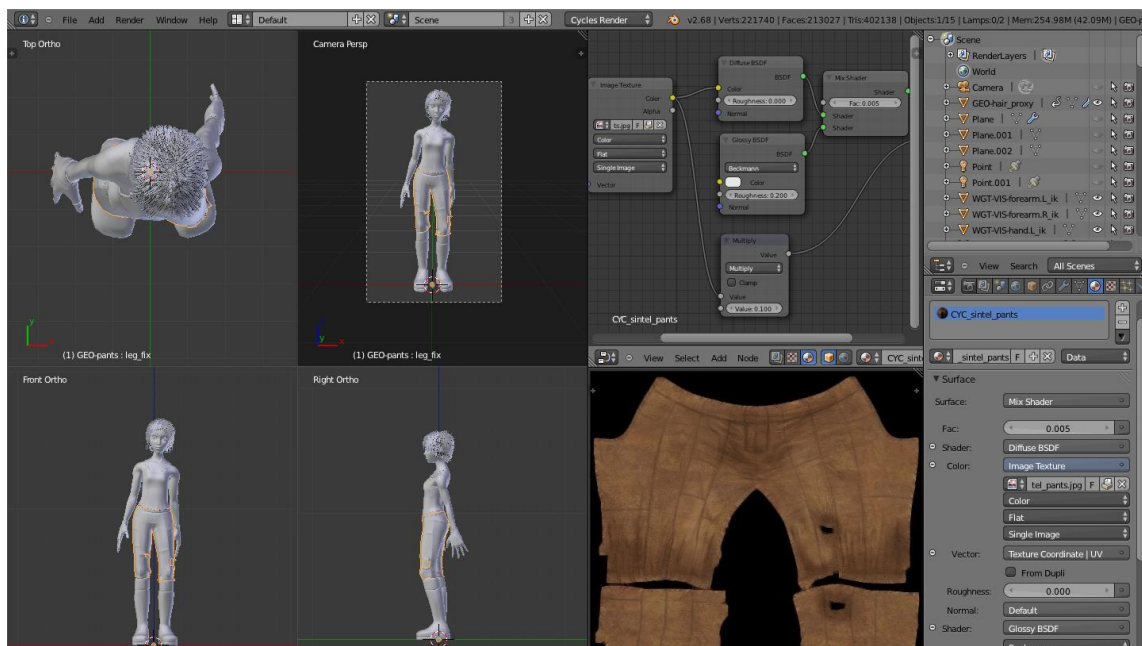


Figure 7- Blender User Interface ([www.blender.org](http://www.blender.org))

## **Overview**

Blender is a free and open source 3D animation software suite. It supports the entirety of the 3D pipeline- modeling, rigging, animation, simulation, rendering, compositing and motion tracking, even video editing and game creation. Advanced users employ Blender's API for Python scripting to customize the application and write specialized tools; often these are included in Blender's future releases. ([www.blender.org](http://www.blender.org))

## **Developer**

Blender is developed by Blender foundation, an independent non-profit public benefit corporation in Netherlands.

## **Version History**

The first version of Blender was released in 1995 by a Dutch animation studio called Neo Geo and Not a Number Technologies- NaN as a commercial software. But in 2002, NaN went bankrupt and consequently, Ton Roosendaal, the main developer of Blender made it an open source project. Since then, Blender has remained open source and the company now runs as a non-profit. The current version is 2.73a which was released on January 2015.

## **Supported Platforms**

Blender runs on multiple platforms including Microsoft Windows, Mac OS X and Linux operating system.

## **Licencing**

Blender is a freeware meaning that is available to use for any purpose for free under a GNU General Public Licence.

### 3.3.2.4 Cinema 4D

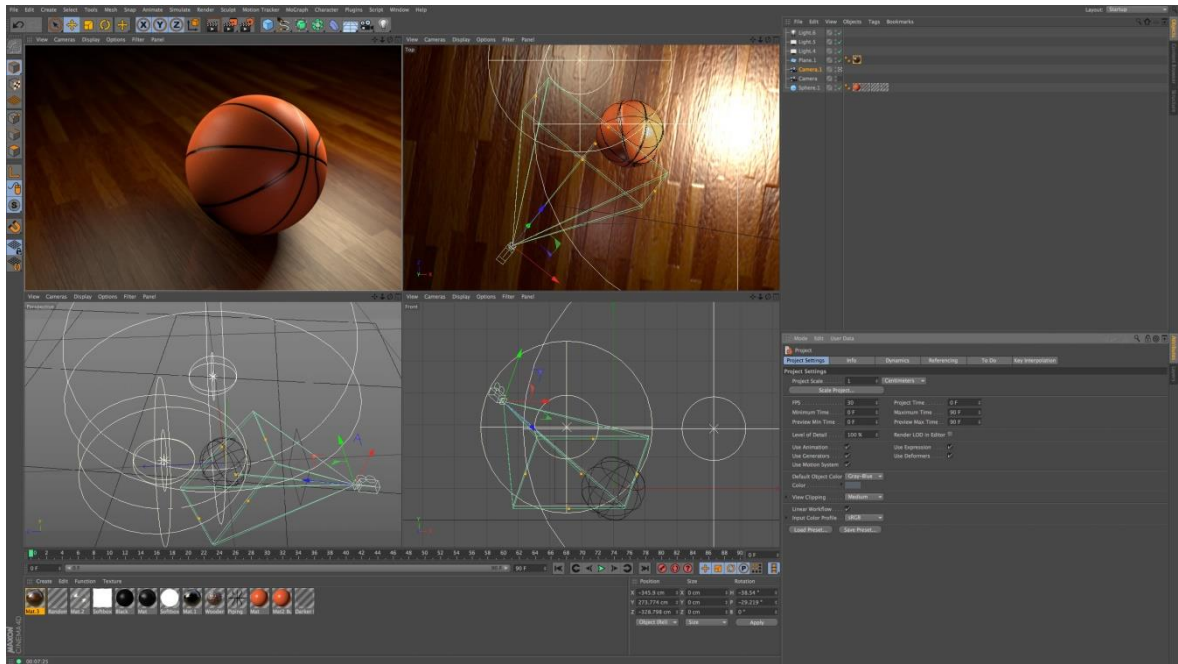


Figure 8- Cinema 4D User Interface ([www.maxon.net](http://www.maxon.net))

#### Overview

Cinema 4D is high-end 3D graphics software. It is used for animation, modeling, sculpting and rendering. It is particular popular for the high quality of 3D graphics it can produce. In a very competitive 3D graphics software market, it has remained one of the preferred tools for 3D artist both in film and game studios.

#### Developer

Cinema 4D was originally developed by Amiga computers and it was exclusively available on the Amiga personal computers. Starting from the fourth version which was released in 1996, Maxon Computer GmbH which is headquartered in Germany started developing it.

#### Version History

The first version of Cinema 4D was released in 1991 with the name FastRay solely for Amiga Personal computers. In 1993, with the release of the new version, it was renamed Cinema 4D still for Amiga platform.

The first release on other platforms started in 1996. Since then, it has continued to improve with each new release supporting the latest industry standards. Its most recent release was in 2014 that is, Cinema 4D R16.

### **Supported Platforms**

Cinema 4D is available for Microsoft Windows and Mac OS X users.

### **Licencing**

Cinema 4D is a commercial software but a free to use version with less features is available for download for students with an 18 months licence.

### **3.3.2.5 Other Frameworks for Creating 3D Models**

There are other 3D software that are also used in the industry but mostly alongside these main ones that have been highlighted. Some of them are summarized in the table below:

<b>Software</b>	<b>Developer</b>	<b>Platform</b>	<b>Licence</b>
Vue	e-on software	Mac OS Microsoft Windows	Commercial
Houdini	Side Effects Software Inc.	Mac OS X Microsoft Windows Linux	Commercial
Autodesk Softimage	Autodesk Inc.	Microsoft Windows Linux	Commercial
Cheetah 3d	MW3D- Solutions	Mac OS X	Commercial

**Table 2- Other 3D Modeling System Software**

(Source: self-authored)

There are some other modern frameworks that help game designers and developers to easily generate terrains and other game environments. These engines makes it possible to quickly simulate an environment without having to model and render meshes that appears on the terrains individually; rather it simulates the environment from built-in models. Some of them include:

- Blended Cities
- Suicidator City Generator
- Skyscraper
- ghostTown
- CityEngine
- Building Generator

### **3.3.3 Frameworks for creating 3D games**

Game engines are the nuts and bolts that sit behind the scenes of every video game. From the artwork right down to the mathematics that decide every frame on screen, the engine makes the decisions. Starting out with rendering- the method of displaying graphics on screen, and integrating a control method and a set of rules for the game to follow- the engine is what a developer builds to "house" the game. Modern 3D game engines are a deluge of meticulously written code, and as such, once used for their intended purpose which is the production of a game they are made for, these engines are often sold, modified, and reused. (Goldstone, 2009)

Due to the level of complexity and cost of such commercial game engines, the game development industry is a difficult area of interest for potential fresh talent to break into, without studying programming languages such as C++ extensively. Modern console and computer games are built around C++ as it is currently the most efficient language in terms of computational speed, and as such, the structure and commands of commercial games engines require thousands upon thousands of such lines of code to function. For many new potential developers, the steep learning curve required to pick up programming languages such as C++, or the engines that utilize it, is simply too great a task to attempt. Without completing degree-level studies in programming or computer animation, it is difficult for many enthusiasts to get started in learning the concepts, methods, and design principles involved in game production. (Goldstone, 2009)

But using the modern framework that is discussed in this subchapter, the complexity of learning and developing games has become a lot easier. The modern frameworks that are discussed in this chapter include:

- Unity
- CryEngine
- Unreal Engine

### 3.3.3.1 Unity

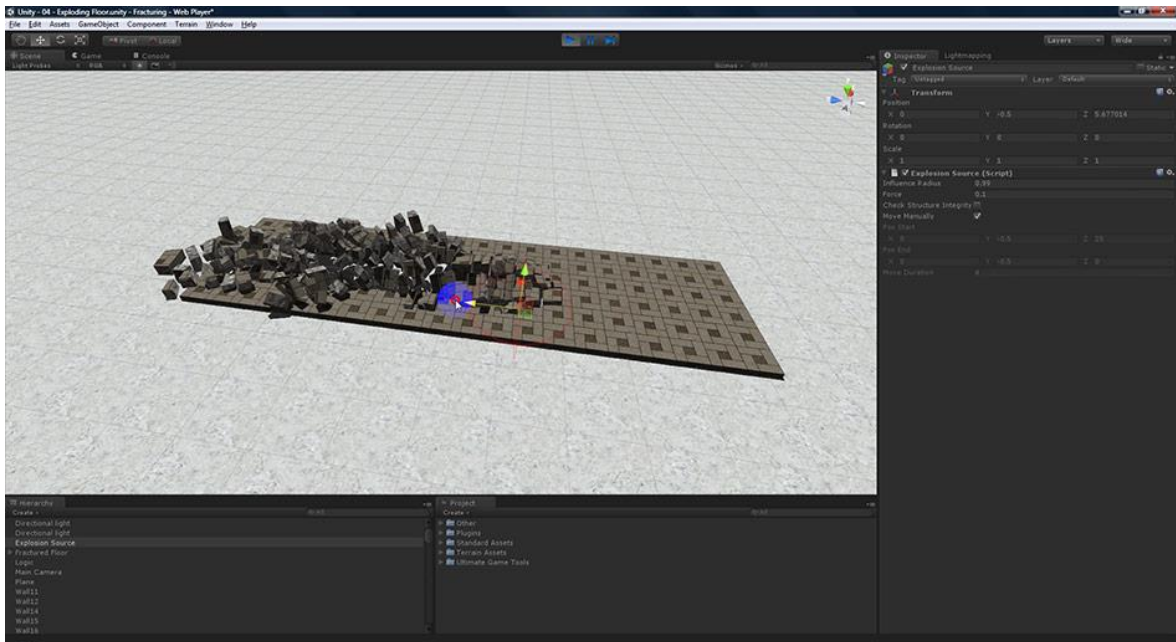


Figure 9- Unity Editor (Unity3D Forum, 2012)

Unity is a 3D game authoring tool for Mac and PC. (Goldstone, 2009) It is a cross-platform 3D and 2D game development system developed by a company called Unity Technologies (originally named Over the Edge). Unity is cross-platform in the sense that the Unity Editor, the game creation and editing tool that is the centerpiece of Unity, runs on OS X and Windows. (Chu, 2013)

There are a million and one reasons why unity stands out from other modern game engine frameworks. From simple features like built-in graphics, sound, and physics engines to more complex feature like Unity Unity's MonoDevelop.

Unity's MonoDevelop is based on the open source C++ library Mono; by using this library, Unity implements just –in-time-compilation (JIT) of code. By using JIT compilation, Unity can take advantage of high-speed compilation, whereby the code you will write for Unity is compiled to Mono just before it is executed. This is crucial for games that must execute code at specific moments during runtime. (Goldstone, 2009)



In addition to the Mono library, Unity also takes advantage of other software libraries in its functionality, such as Nvidia's PhysX physics engine, OpenGL, and DirectX for 3D rendering and OpenAL for audio. All these libraries are built into the application, so game designers and developers not need to worry about learning how to use them individually. (Goldstone, 2009) This makes working with unity seamless.

Furthermore, the market for multi-platform games- especially casual games for iPhone and Android is extremely popular at the moment, and Unity's commitment to cross-platform delivery is well proven. Originally a Mac-based authoring application that could publish to Mac and Windows, Unity unveiled its Windows version in the spring of 2009. As expected, it has opened opportunities for PC-based developers and artists. Since that time, Unity has continued to add support for iPhone, Android, iPad, Windows Phone, Blackberry, Tizen, Windows store app, Mac, Linux/Stream OS, Web player, Wii, Xbox 360, Xbox One, Android TV, Samsung Smart TV, Oculus Rift, Gear VR, PlayStation and PlayStation Vista. (Blackman, 2013)

Finally, Unity provides an excellent entry point into game development, balancing features and functionality with price point. The free version of Unity allows people to experiment, learn, develop, and sell games before committing any of their hard-earned cash. Unity's very affordable offering either a one-time licence or a monthly subscription payment model. Its feature-packed Pro version is royalty free, allowing people to make and sell games with the very low overhead essential to the casual games market. (Blackman, 2013)

### 3.3.3.2 CryEngine

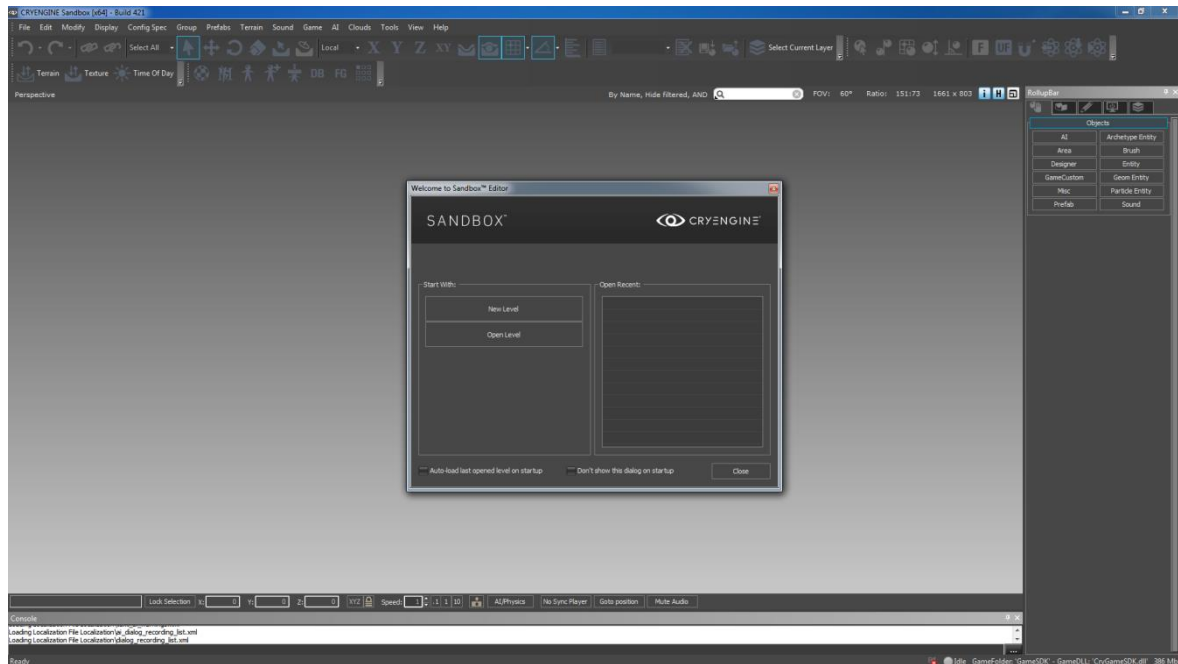


Figure 10- Sandbox Editor (Schulz, et al., 2015)

CryEngine is an advanced development solution for the creation of blockbuster games, movies, high-quality simulations, and interactive applications. The third iteration of Crytek's proprietary engine CryEngine 3 is the only all-in-one game development solution for the PC, Xbox 360, and PlayStation. (www.crytek.com) The main limitation of the engine is that the deployment module of the game in as much as it covers the main game play platforms; its reach is still not so extensive.

CryEngine offers a few advanced features like the Live Create which enables developers to test their games on multiple platforms simultaneously while working with a single editor. With this, developers can see the result of how the game play is not just from the editor but on the platform while still developing.

The licencing for CryEngine is for a monthly subscription fee but they also offer a full licence for a price offered directly by the sales team of Crytek.

### 3.3.3.3 Unreal Engine



Figure 11- Unreal Engine Level Editor ([www.unrealengine.com](http://www.unrealengine.com))

Unreal Engine is a system that organizes a game developer's assets which includes: characters, artwork, props, weapons, music, sound effects, voiceovers, etc. into a visually stunning interactive environment. But not just any visually stunning environment; one that behaves just the way you want it to- as a game, in other words. (Busby, et al., 2009)

The Unreal Engine contains several components. Each can work independently, but they are kept humming along in harmony by a central core engine. These components include: graphics engine, sound engine, physics engine, input manager, network infrastructure and UnrealScript interpreter.

Unreal Engine was developed in 1998 by Epic Games to much accolade. It has consistently improved its framework to the current Unreal Engine 4 which is considered to be highly powerful engine next to Unity by some and to others, better than Unity. In essence, it is one of the key market players in the 3D games industry.

The licencing model of the current Unreal Engine is basically a 5% royalty on revenue per quarter per game. This gives the user access to all the features of the engine.

## 4.Design and Implementation

This chapter, details the design and implementation of two video games as an illustration of the modern means and tools of creating 3D graphics applications discussed in the previous chapter.

For this thesis, Unity game engine will be used for the game development. Unity game engine is used in this thesis because in comparison with Unreal Engine and CryEngine, Unity offers a free personal development version to users for both learning and commercial purposes and also there are a lot of learning resources on the Unity website. The game is based on extensive study of learning resources and examples on Unity study repository. The table below shows the cost comparison between the three games development engines discussed in the previous chapter.

Game Engine	Licence Version	Cost
Unity	Unity Personal	Free both for personal and commercial use (for commercial use, revenue exceeding \$100,000 requires and upgrade to Professional Version)
	Unity Professional	\$75/ month subscription or \$1500 for full licence
	Unity iOS and Android Add-on	\$75/month subscription or \$1500 for full licence for each platform
	Enterprise Solution	Special pricing and arrangement
Unreal Engine	Unreal Engine 4	5% royalty per game per quarter
CryEngine	CryEngine 3	9.90 USD/EUR /month subscription or full licence for an agreed amount

**Table 3- Cost Comparison of Game Engines**

(Source: self-authored)

## 4.1 Game One

### Game Description

The first game is a 3D arcade type game. It comprises of 3D model spaceship fighting in an outer space against asteroids which are coming at it. The spaceship is the main player and is controlled by the player. The player can navigate the spaceship to the right, left, up and down in the 3D space. The goal is to shoot at the asteroids as they fall from the top of the screen towards it. Each asteroid that the player destroys adds some point to the player's score which is displayed on the screen. The more asteroids the player destroys, the more the number that falls from the top of the screen. The game ends when there is a collision between the spaceship and an on-coming asteroid.

The 3D models used in the game are downloaded from internet repository or Unity asset store under free licence.

### Implementation

Before implementing the game, a diagrammatic description of the Unity editor interface is essential because it will be referenced severally in this part of the thesis.



Figure 12- Diagrammatic Description of Unity Interface ([www.unity.com](http://www.unity.com))

## Setting up the Project and Player in Unity

In order to start creating the game a new project has to be created in a directory. This is done from the file menu. As part of the process, unity displays a dialog box with standard 3D assets that are part of the unity game engine; some of the assets include: Character Controllers, Terrain, Skybox, Scripts etc. These standard assets helps to speed up the process of game development as the game designer can easily drag and drop them on the scene as needed without having to deal with making every single asset in external 3D graphics software.

The first focus is to set up the main player which is the spaceship. This is imported into the assets folder of the game and placed on the scene by dragging it to the hierarchy or directly on the viewport. Then a physics component called Rigidbody is then applied to the spaceship.

According to unity documentation, adding a Rigidbody component to an object will put its motion under the control of Unity's physics engine. Even without adding any code, a Rigidbody object will be pulled downward by gravity and will react to collisions with incoming objects if the right Collider component is also present. (www.unity.com) Because this game is assumed to be in the space, we need to remove gravity setting from the physics component. This is done by simply unchecking a box in the Rigidbody setting.

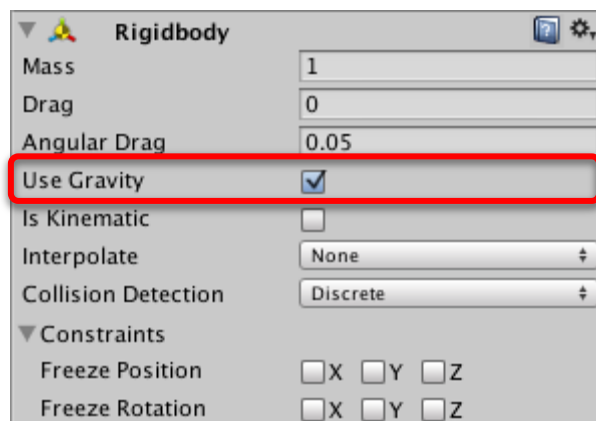


Figure 13- Unity Rigidbody Setting (www.unity.com)

## Build Platform

In order to properly set up the game components, the deployment platform has to be set. This is important as the placement of the other game component depends on the resolution

of the screen which varies among different platforms. For this game, the build platform is the Unity Web Player. This means that the game will be played on any web browser locally or on the internet if it is hosted online. The build setting is accessible from the file menu on the toolbar. The screen resolution for web platform is set to 400 by 680. All other settings are left as default

## Setting up Game Environment and Elements

The main game elements to consider are the camera and the lighting. The game does not need multiple cameras as it is designed to be an arcade style game. The main camera is set to the orthographic view and placed horizontally above the spaceship. This gives the 3D environment a simplified view, as the camera does not probe into the depth of the models in the game.

Since the game is set in arcade style, a simple background will be sufficient for the game play. To set the background, quad, which is one of the built in game components, is selected and a simple texture which is made in 2D texturing software is applied to the quad. Finally the game needs to be lit so that the models will be visible when the game is played and the ideal light for this game is a directional light. Directional light is an evenly scene lightening source. Its intensity and brightness depends on the rotation and not the distance placement as such, it lights all parts of the scene that it is facing evenly depending on the intensity. Three directional lights are placed on the scene so as to light the spaceship evenly from all three directions.

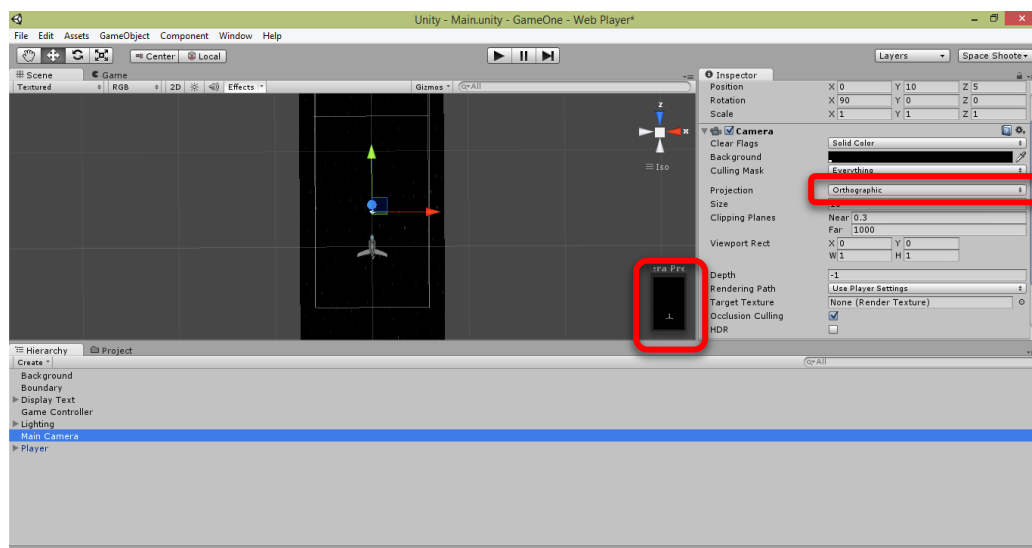


Figure 14- Camera set in Orthographic View  
(Source: self-authored)

## Setting up the Player Movement

To set the player movement, there is need for programming, this is not to implement artificial intelligence rather, the player's input from the physical game controller e.g. mouse, keyboard, joystick etc. needs to be read and understood by the spaceship which represents the player in the game.

Unity supports three programming languages JavaScript, Boo and C#. For this thesis, we will be programming in C#. Unity has its own c# library which makes programming for game objects a lot easier. It also comes with MonoDevelop, an integrated development environment that is supplied to games developers with Unity.

The code for the player movement is:

```
[System.Serializable]
public class Boundary
{
    public float xMin, xMax, zMin, zMax;
}
public class PlayerController : MonoBehaviour {
    public float speed;
    public float tilt;
    public Boundary boundary;
{
    float moveHorizontal = Input.GetAxis ("Horizontal");
    float moveVertical = Input.GetAxis ("Vertical");
    Vector3 movement = new Vector3(moveHorizontal, 0.0f,
moveVertical);
    rigidbody.velocity = movement * speed;
    rigidbody.position = new Vector3(
        Mathf.Clamp
(rigidbody.position.x,boundary.xMin,boundary.xMax),
        0.0f,
        Mathf.Clamp (rigidbody.position.z,
boundary.zMin,boundary.zMax));
    rigidbody.rotation = Quaternion.Euler(0.0f, 0.0f,
rigidbody.velocity.x * -tilt);
}
}
```

The code for the movement is in three parts: the first part controls the boundary which confines the spaceship to a particular area in the 3D space within the screen and camera view. The second part receives the user input from the controller, in this case a keyboard for both horizontal and vertical movements. The third part is for tilting the spaceship; this gives it a more natural feel as it can tilt to the left or to the right as it moves. The public variables defined in each case allow for easy manipulation of values from the editor.



## Creating Weapon for the Spaceship and Enemy

To create weapon and enemies, a built-in asset type in Unity game engine called prefab is used. Prefab acts as a template from which you can create new object instances in the scene; any edits made to a prefab asset are immediately reflected in all instances produced from it but you can also override components and settings for each instance individually. ([www.unity.com](http://www.unity.com))

To create the weapon (shots fired by the spaceship) and enemy (the asteroid), models for both shots and asteroid are placed as child object in an empty game object which is created on the scene. In the asset folder a prefabs are created. The empty game objects for the shots and the asteroid are dragged to their respective prefabs which can now be instantiated multiple things on the scene during game play from single game object created. Each prefab has capsule collider which is set to trigger.

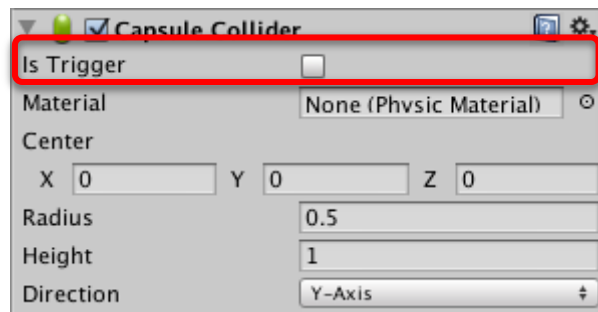


Figure 15- Unity Capsule Collider Setting ([www.unity.com](http://www.unity.com))

The code below is to fire shots from the spaceship and destruction of the shots as it leaves the scene which is also written also in C#

```
using UnityEngine;
using System.Collections;
public class Mover : MonoBehaviour {
    public float speed;
    // Use this for initialization
    void Start () {
        rigidbody.velocity = transform.forward * speed;
    }
}

public GameObject shot;
public Transform shotSpawn;
public float fireRate;
private float nextFire;
void Update ()
```

```

        {
            nextFire = Time.time * fireRate;
            if (Input.GetButton("Fire1") && Time.time > nextFire)
            {
                Instantiate(shot, shotSpawn.position,
shotSpawn.rotation);
            }
        }
public class destroyByBoundary : MonoBehaviour {

    void OnTriggerExit(Collider other) {
        Destroy(other.gameObject);
    }

}

```

In order to make the asteroid part of an exciting gameplay, a few considerations have to be made: first, the asteroid has to enter the scene in some kind of movement that suggest that it has been in motion. Secondly, the prefabs have to enter the scene not in so many numbers and not too few to keep the game a bit challenging and the more asteroids destroyed by the spaceship, the greater the number of asteroids entering the scene. The code for this implementation is:

```

using UnityEngine;
using System.Collections;
public class RandomRotator : MonoBehaviour {
    public float tumble;
    void Start () {

        rigidbody.angularVelocity =
Random.insideUnitSphere*tumble;
    }
}
public class GameController : MonoBehaviour {

    public GameObject harzard;
    public Vector3 spawnValues;
    public int harzardCount;
    public float spawnWait;
    public float startWait;
    public float waveWait;
    IEnumerator SpawnWaves()
    {
        yield return new WaitForSeconds (startWait);
        while(true) {
            for(int i = 0; i < harzardCount; i++)
            {
                Vector3 spawnPosition = new
Vector3(Random.Range (-spawnValues.x,spawnValues.x), spawnValues.y,
spawnValues.z);
                Quaternion spawnRotation =
Quaternion.identity;
                Instantiate (harzard,spawnPosition,
spawnRotation);
            }
            yield return new WaitForSeconds (spawnWait);}
    }
}

```

## Wrapping up the build

For a more interesting gameplay, some effects are needed in the game. For example: audio, graphic user interface to display dialog with the player and a mechanism to restart the game when the spaceship is destroyed.

Unity supports a number of audio data types which include: MPEG, Ogg Vorbis, WAV and AIFF.

To add an audio file to the game, we add an audio component and define the sound. From the settings panel of the audio source we can make a few adjustments to the settings of the audio file.

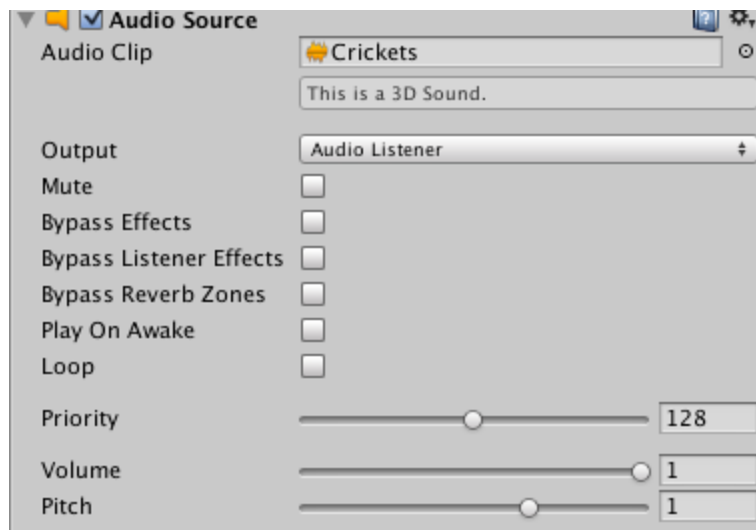


Figure 16- Audio Source Setting in Unity ([www.unity.com](http://www.unity.com))

To add sound effect like sounds of shots fired for the spaceship, the audio file is dragged to the prefab and a line of code is added to the code for shooting the weapon.

```
audio.Play();
```

This code plays the sound each time a shot is fired by the spaceship.

Finally a graphic user interface- GUI is added to display a score for the player; the score is increased by a value for each asteroid that the player destroys. Also a GUI is added to display the text "Game Over" when the spaceship is destroyed and finally another GUI to display "Press 'R' to Restart" which restarts the game. The code below takes care of the

displaying the correct value for the score and displaying the GUI at the right time.

```
public GUIText scoreText;
    public GUIText restartText;
    public GUIText gameOverText;

    private int score;
    private bool restart;
    private bool gameOver;

    void Start () {
        gameOver = false;
        restart = false;
        restartText.text = "";
        gameOverText.text = "";
        score = 0;
        UpdateScore();
void Update()
    {
        if (restart)
        {
            if(Input.GetKeyDown (KeyCode.R))
            {
                Application.LoadLevel
(Application.loadedLevel);
            }
        }
        if (gameOver)
        {
            restartText.text = "Press 'R' to Restart";
            restart = true;
            break;
        }
    }
public void AddScore (int newScoreValue)
{
    score += newScoreValue;
    UpdateScore();
}
void UpdateScore()
{
    scoreText.text = "Score: " + score;
}
public void GameOver()
{
    gameOverText.text = "Game Over";
    gameOver = true;
}
}
```

## Build and Deployment

To build and deploy a game in Unity is as easy as a few clicks. For the first game, the build platform is the web player. As such, the game can be played on a web browser on a local machine or on the internet if it is uploaded to a web server.

From the file menu, select the build and setting option. With the target platform set to web player, click build. This builds the game with the build setting specified by the game developer and the game is ready to be played. To play the game, the unity web player plugin is needed; it is available on Unity website. The development graphics, source code and the deployed game can be found on the CD attached to this thesis.

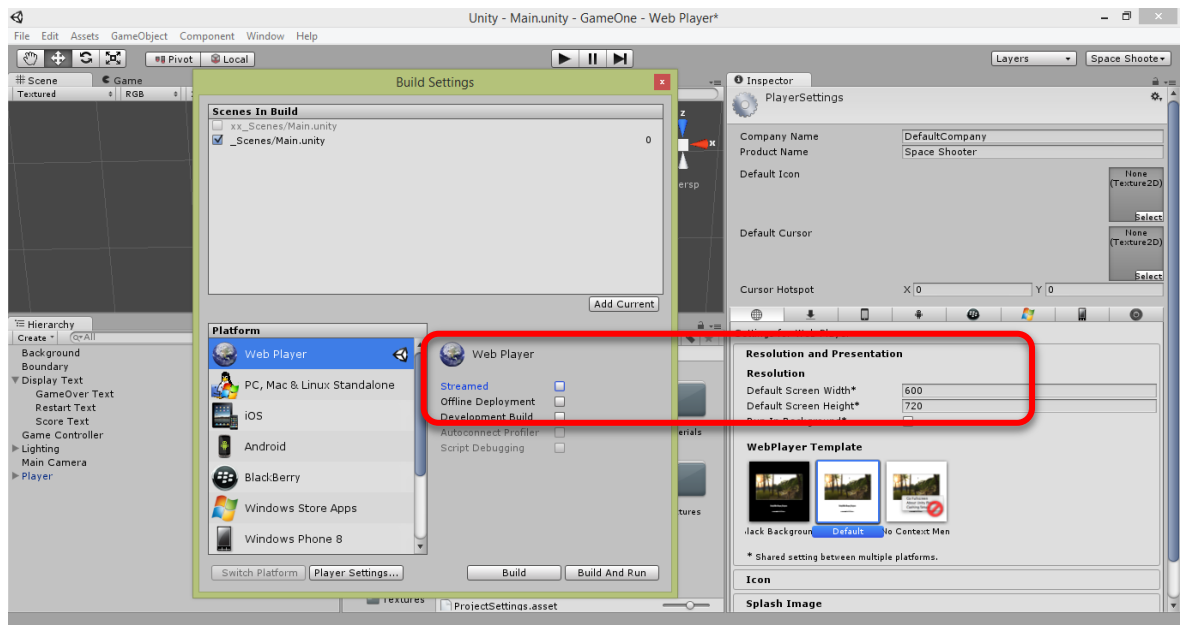


Figure 17- Build Setting in Unity Editor

(Source: self-authored)

## 4.2 Game Two

### Game Description

Game one is an arcade style game which does not show the depth of the 3D models used in creating the game. For this second game, the same assets used in the first game are used again with a few setting adjustments which are intended to show the three dimensional details of the model.

## Implementation

The camera in the first game is set to orthographic view. This gives the player a 2D dimensional view of the 3D scene. To change the view we need to add perspective, in Unity, adding perspective is as simple as a single click. This sets the whole view of the game in three dimension scene.

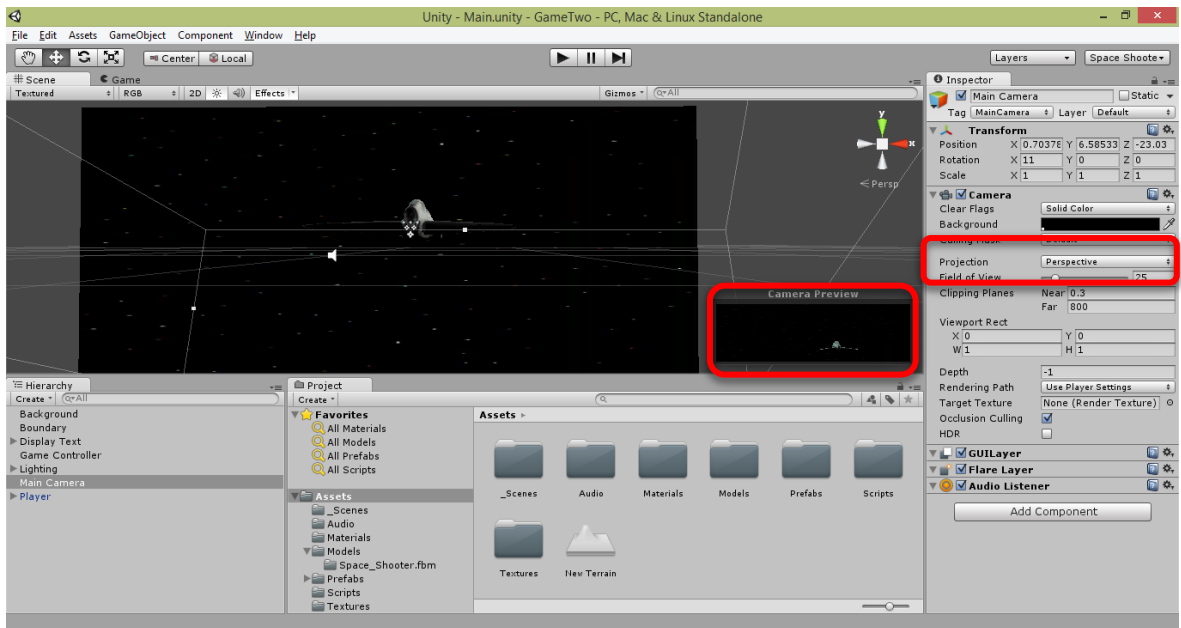


Figure 18- Camera set in Perspective View  
(Source: Self-authored)

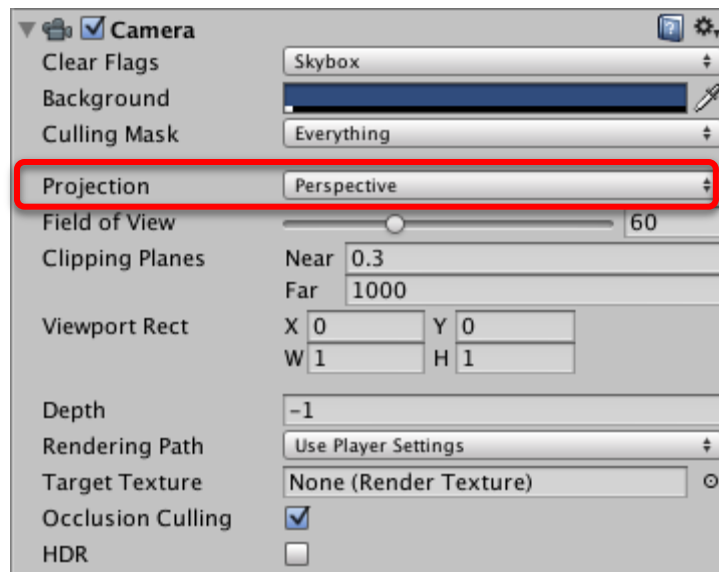


Figure 19- Camera Setting in Unity (www.unity.com)

The background is also rotated and set on the y-axis as opposed to the x-axis in the first game. The camera is then placed behind the spaceship. In summary, the game has a static background on the y-axis and the camera set on perspective view, and projecting into the z-axis behind the player.

It is the same game, same style and look, but by changing the camera setting and placing behind the player suddenly it looks like something entirely different.

Other settings remain the same.

## Build and Deployment

The game will be deployed to play on PC; this is to show the different build capability of the Unity game engine. This is done from the build and setting option from the file menu also. The platform is switched to standalone. Standalone build option deploys to PC, Mac and Linux platforms; select a destination directory and click build.

Unity creates an executable file in the directory which when clicked shows a dialog box displaying the game control and screen resolution select menu and the game is ready to be played. The development graphics, source code and the deployed game can also be found on the CD attached to this thesis.

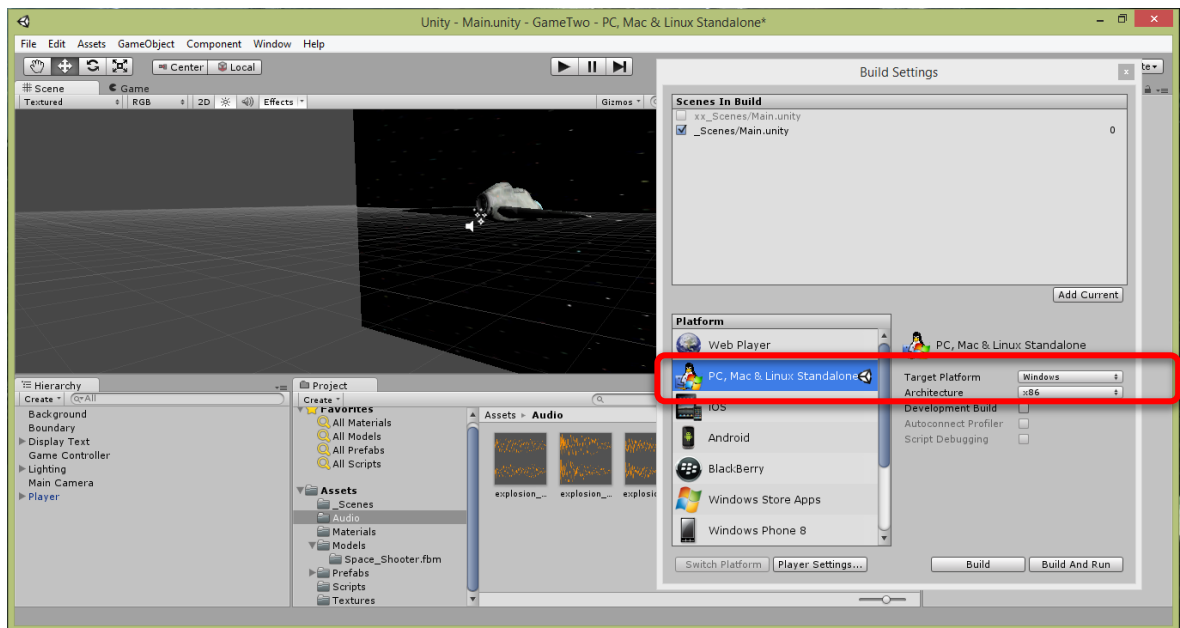


Figure 20- Build and Deployment on PC

(Source: self-authored)

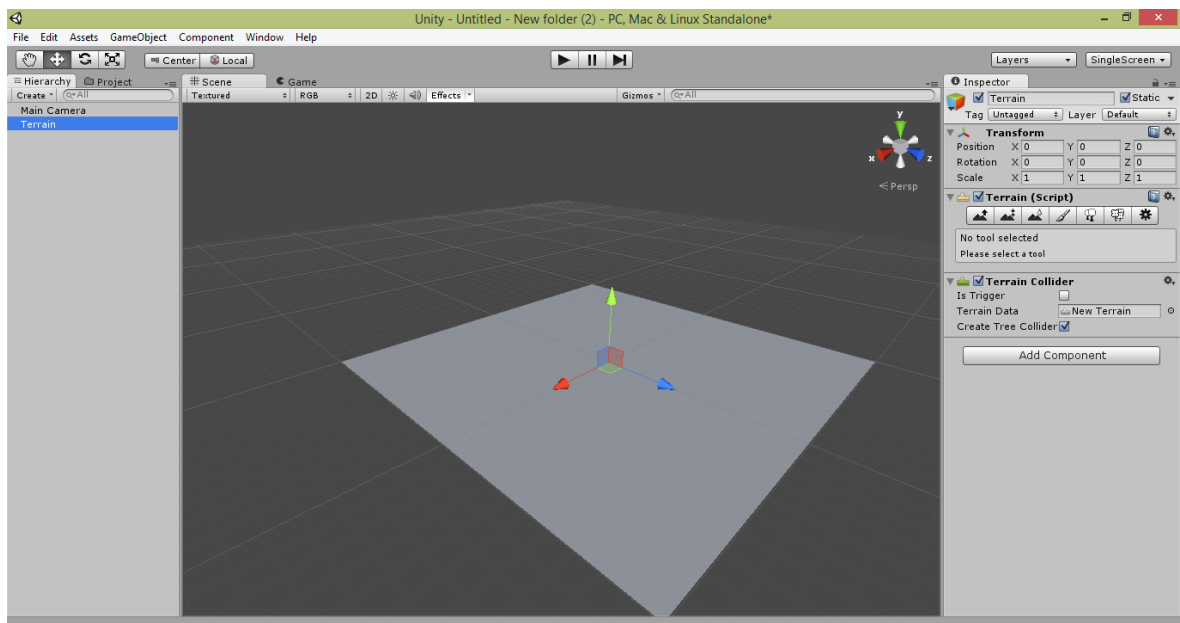
## 4.3 Simulation of Game Environment

This implementation describes the creation of 3D game environment with 3D graphics components supplied within Unity.

### Terrain Creation and Painting

Creating a terrain in Unity is as simple as selecting the terrain object from the GameObject menu on the toolbar. This creates an empty terrain with collider which makes the terrain to serve as a platform for the game by preventing the player from falling through it.

The terrain can be painted with elevations- Mountain, depths- valley, trees, grasses and textures. This is done by importing the terrain and tree standard assets into the game development directory. These assets are built in features that are supplied with Unity. From the terrain setting in the inspector panel, the terrain details are painted.



**Figure 21- Empty Terrain in Unity**

(Source: self-authored)



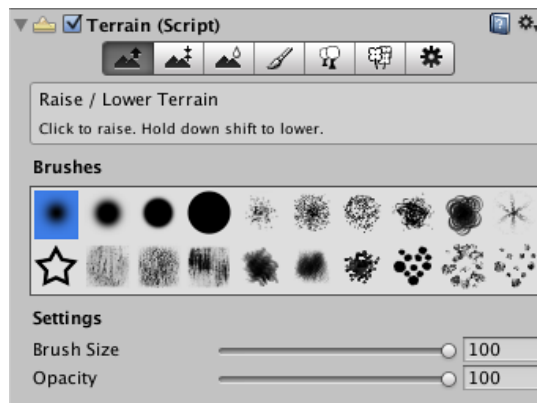


Figure 22- Terrain Setting in Unity (www.unity.com)

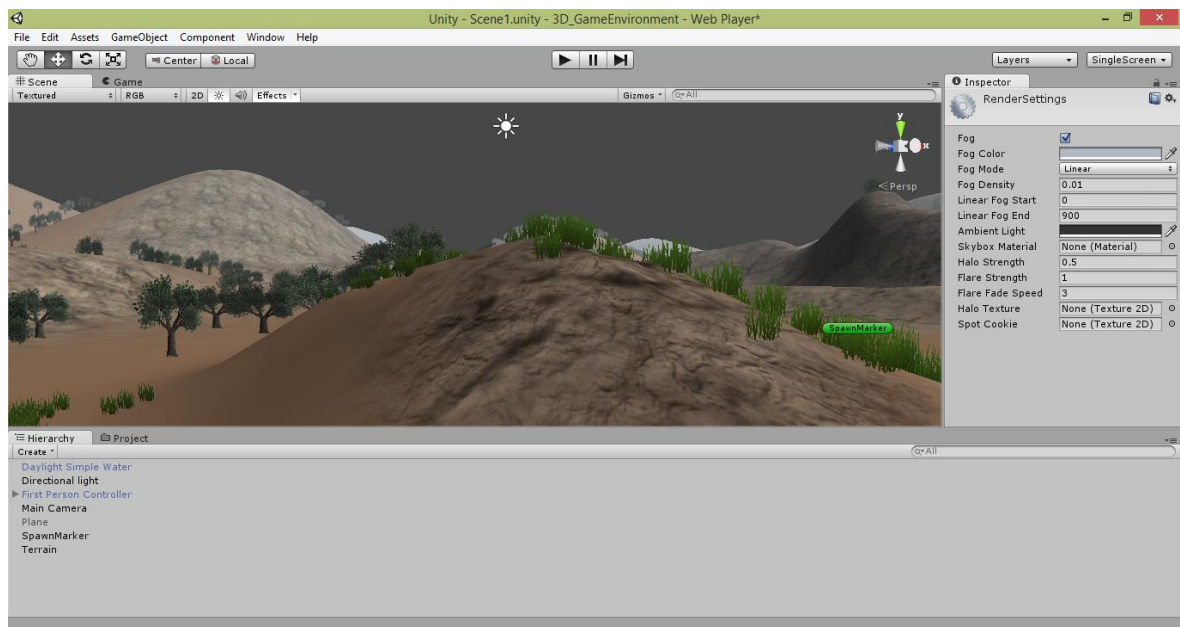


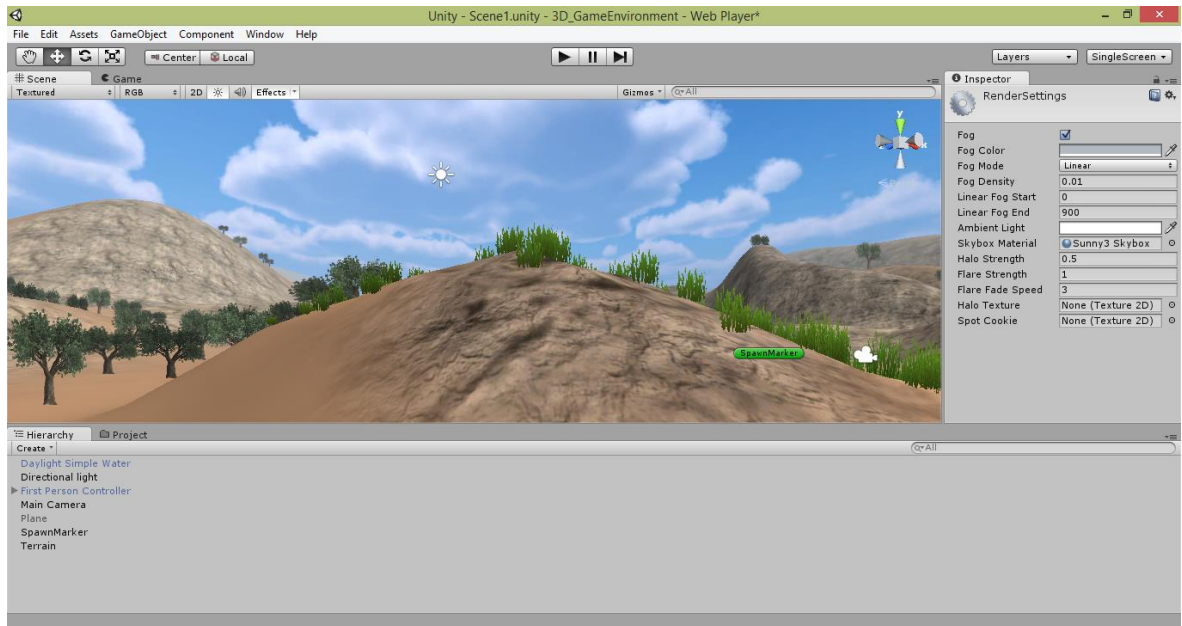
Figure 23- Terrain with Mountains, Trees and Grass

(Source: self-authored)

## Adding Sky Feature and Fog

The addition of sky rendering is another feature that can easily be implemented within Unity. To do this, we import the Skybox standard asset into the game environment directory. From the render setting on the Edit Menu, we choose from a selection of skybox material that Unity has. This is then applied above the terrain into a horizon depicting the sky.

Addition of fog to the game environment adds the feeling of a never ending 3D space not minding the limited size of the terrain. This makes the game environment to seem like an endless world as it fades into a horizon.



**Figure 24- Terrain with Skybox Rendering**  
(Source: self-authored)

## Adding a Lake

A basic water feature is added to this game environment by simply dragging and dropping it to scene. The setting of this water can be adjusted from the settings on the inspector panel.

## Character Controller

In order to explore our game environment in 3D, we add a first person character controller. This acts as the main actor in a first person game. It comes with camera component which makes the game to appear as though it is seen from the controller's view point. The character controller is able to navigate the game environment by walking, running or jumping depending on the control by the game player.

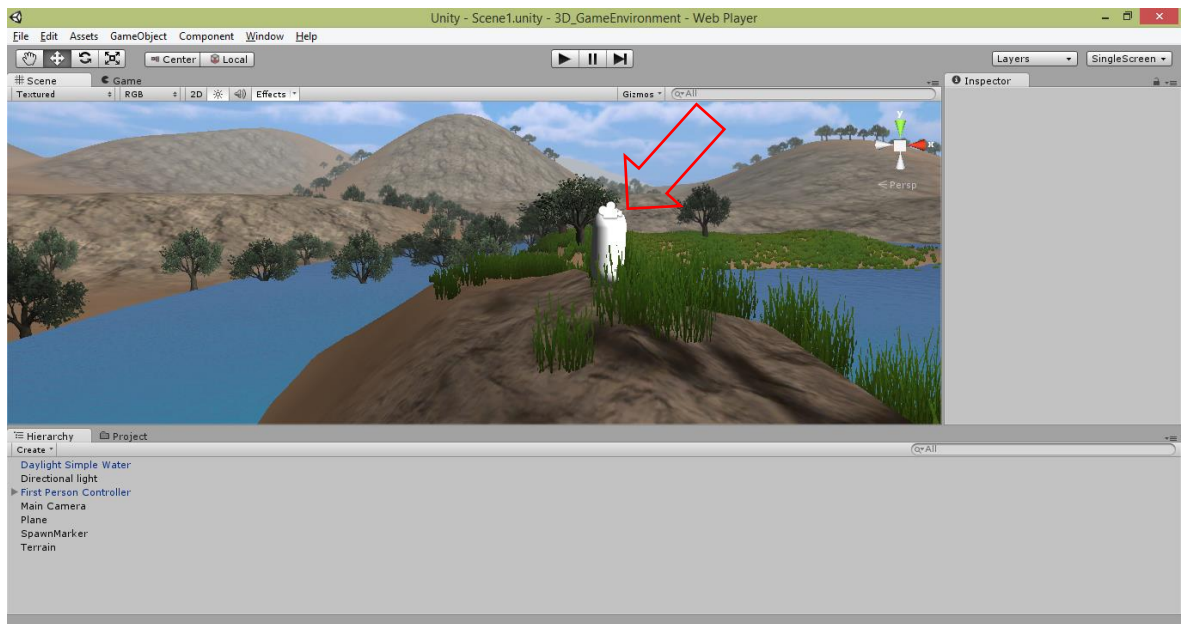


Figure 25- Game with First Person Character Controller

(Source: self-authored)

## Build and Deployment

The game environment is deployed to a web player just like in the first game that was previously described. All the development graphics and the deployed game can be found on the CD attached to this thesis.

## **5. Conclusion**

### **5.1 Discussion**

Having used Unity as a modern tool to successfully create two examples of 3D graphics application, it would be correct to assert that Unity is a very robust framework with a lot of features which gives so much possibility to game developers to create a perfect imaginary world for an excellent game play.

The multiple platform deployment architecture of the Unity framework is far reaching but the process is still not yet entirely seamless as the game developer is expected to do a few setup configurations outside the engine to get this feature to work on some platform.

Finally, the concept of virtual reality is now in practice, the new release of Unity- Unity 5 is built with the capability to develop for Virtual Reality Devices like Oculus Rift and Gear VR but the actual experience for players though immersive still needs to be enhanced for a more realistic game play. The limitation is not on the engine alone but also on the devices that it is built for.

### **5.2 Conclusion**

The first objective of this thesis was to describe the existing technologies for creating 3D graphics application. This was accomplished in the first part of the third chapter. This was done by studying extensively, different literatures and reviewing them through secondary research techniques.

The second goal was to highlight all the modern tools- system software or computer programs available for easy creation of 3D graphics. This was also done in the second part of the third chapter.

My final goal was to describe how using one of the modern system software mentioned in the theoretical part to demonstrate how modern system software can make the creation of 3D graphics application easy. This was done using two examples as shown in the fourth chapter. Finally, a 3D game environment was simulated and deployed. This was done so as to further illustrate the 3D creation tools available within the game engine that was used for the design and implementation.

In conclusion, as a result of this thesis, it is possible to say that the tools available to designers, developers, programmers and any user that works with 3D graphics makes for a very easy workflow and their value as they continue to progress in line with new technological innovations cannot be under-estimated.

# Bibliography

**3Dfx Interactive, Inc. 1997.** Glide 2.2 Programming Guide. *gamers.org*. [Online] 8 March 1997. <http://www.gamers.org/dEngine/xf3D/glide/glidepgm.htm>.

**Blackman, Sue. 2013.** *Beginning 3D Game Development with Unity 4*. New York : Apress, 2013. 978-1430248996.

**Busby, Jason, Parrish, Zak and Wilson, Jeff. 2009.** *Mastering Unreal Technology, Volume I: Introduction to Level Design with Unreal Engine 3*. Carmel : Sams Publishing, 2009. 978-0672329913.

**Chu, Philip. 2013.** *Learn Unity 4 for iOS Game Development*. New York : Apress, 2013. 978-1430248750.

**Cory, Janssen: www.techopedia.com.** What is an Application Programming Interface? *www.techopedia.com*. [Online] <http://www.techopedia.com/definition/24407/application-programming-interface-api>.

**Giambruno, Mark. 2002.** *3D Graphics and Animation*. Oaks, CA : New Riders Publishing, 2002. 978-0735712430.

**Goldstone, Will. 2009.** *Unity Game Development Essentials*. Birmingham : Packt Publishing, 2009. 978-1-847198-18-1.

**Gortler, Steven J. 2012.** *Foundations of 3D Computer Graphics*. Cambridge, MA : The MIT Press, 2012. 978-0262017350.

**Hees , H. 2006.** *3D Computer Graphics*. Mainz : Pedia Press, 2006. uidrpqgpwhpwilvl.

**Horst, Zachary. 2009.** *3-Dimensional Video Games Engines*. Villanova, PA : s.n., 2009.

**Hughes, John F. , et al. 2014.** *Computer Graphics: Principles and Practice*. Third. Willard : Addison-Wesley Professional, 2014. 978-0321399526.

**Parisi, Tony. 2012.** *WebGL Up and Running*. Sebastopol, CA : O'Reilly Media Inc, 2012. 978-1-449-32357-8.

**Schulz, Nicolas and Johnson, Adam. 2015.** CRYENGINE Manual. *dosc.cryengine.com*. [Online] 11 January 2015. <http://docs.cryengine.com/display/SDKDOC2/Home>.

**Segal, Mark and Akeley, Kurt. 2015.** *OpenGL 4.5 Core Profile*. s.l. : The Khronos Group Inc, 2015.

**Shreiner, Dave, et al. 2013.** *OpenGL Programming Guide*. Upper Saddle River, NJ : Addison-Wesley Professional, 2013. 978-0321773036.

**Stenning, Justin. 2014.** *Direct3D Rendering Cookbook*. Birmingham : Packt Publishing, 2014. 978-1-84969-710-1.

**Stephenson, Ian. 2007.** *Essential RenderMan*. London : Springer, 2007. 978-1-84628-344-4.

**Thorn, Alan. 2005.** *DirectX 9 Graphics: The Definitive Guide to Direct3D*. Plano : Wordware Publishing, Inc, 2005. 978-1-55622-229-7.

**Unity3D Forum. 2012.** Editor Extension for Unity 3D. *www.forum.unity3d.com*. [Online] 10 October 2012. <http://forum.unity3d.com/threads/ultimate-fracturing-destruction-editor-extension-for-unity-3d.184170/>.

**Wright , Richard S. , et al. 2010.** *OpenGL SuperBible*. Boston, MA : Addison-Wesley Professional, 2010. 978-0321712615.

**www.alternativeto.net.** Autodesk Maya. *Alternativeto.net*. [Online] Alternativeto.net. <http://alternativeto.net/software/maya/>.

**www.amd.com.** AMD's Revolutionary Mantle. *AMD*. [Online] <http://www.amd.com/en-us/innovations/software-technologies/technologies-gaming/mantle#overview>.

**www.autodesk.com.** 3D Animation Software, Computer Animation Software | Maya. *Autodesk*. [Online] <http://www.autodesk.com/products/maya/overview>.

—. 3d Modeling and Rendering Software. *Autodesk*. [Online] <http://www.autodesk.com/products/3ds-max/overview>.

**www.blender.org.** About. *www.blender.org*. [Online] <http://www.blender.org/about>.

—. Features. *blender.org*. [Online] <http://www.blender.org/features>.

**www.crytek.com.** CryEngine Overview. *www.crytek.com*. [Online] <http://www.crytek.com/cryengine/cryengine3/overview>.

**www.maxon.net.** Cinema 4D Studio. *Maxon.net*. [Online] CINEMA 4D Studio.  
<http://www.maxon.net/products/cinema-4d-studio/who-should-use-it.html>.

**www.microsoft.com.** DirectX SDK Readme. *Microsoft*. [Online]  
<http://msdn.microsoft.com/directx/sdk/readmepage>.

—. Getting Started with Direct3D. *Microsoft*. [Online] [https://msdn.microsoft.com/en-us/library/windows/desktop/hh769064\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/windows/desktop/hh769064(v=vs.85).aspx).

**www.osu.edu.** Section 2: The emergence of computer graphics technology. *The Ohio State University Web site*. [Online] <https://design.osu.edu/carlson/history/lesson2.html>.

**www.pcmag.com.** API Definition. *www.pcmag.com*. [Online]  
<http://www.pcmag.com/encyclopedia/term/37856/api>.

**www.unity.com.** Unity Manual. *www.doc.unity3d.com*. [Online] Unity.  
<http://docs.unity3d.com/Manual/LearningtheInterface.html>.

—. Unity Scripting API. *docs.unity3d.com*. [Online] Unity.  
<http://docs.unity3d.com/ScriptReference>.

**www.unrealengine.com.** Level Editor. *Unreal Engine*. [Online]  
<https://docs.unrealengine.com/latest/INT/Engine/UI/LevelEditor/index.html>.

**www.web3d.org.** What is X3D. *Web3D Consortium*. [Online]  
<http://www.web3d.org/x3d/what-x3d>.



# List of Figures

FIGURE 1- A COMPARATIVE DESCRIPTION OF AN API.....	12
FIGURE 2- OPENGL LOGO .....	13
FIGURE 3- MICROSOFT LOGO.....	17
FIGURE 4- A RENDERING API (STEPHENSON, 2007).....	20
FIGURE 5- USER INTERFACE FOR AUTODESK 3DS MAX 2014 (WWW.AUTODESK.COM).....	27
FIGURE 6- AUTODESK MAYA USER INTERFACE (WWW.AUTODESK.COM) .....	29
FIGURE 7- BLENDER USER INTERFACE (WWW.BLENDER.ORG).....	30
FIGURE 8- CINEMA 4D USER INTERFACE (WWW.MAXON.NET) .....	32
FIGURE 9- UNITY EDITOR (UNITY3D FORUM, 2012) .....	35
FIGURE 10- SANDBOX EDITOR (SCHULZ, ET AL., 2015) .....	37
FIGURE 11- UNREAL ENGINE LEVEL EDITOR (WWW.UNREALENGINE.COM).....	38
FIGURE 12- DIAGRAMMATIC DESCRIPTION OF UNITY INTERFACE (WWW.UNITY.COM).....	40
FIGURE 13- UNITY RIGIDBODY SETTING (WWW.UNITY.COM) .....	41
FIGURE 14- CAMERA SET IN ORTHOGRAPHIC VIEW .....	42
FIGURE 15- UNITY CAPSULE COLLIDER SETTING (WWW.UNITY.COM).....	44
FIGURE 16- AUDIO SOURCE SETTING IN UNITY (WWW.UNITY.COM).....	46
FIGURE 17- BUILD SETTING IN UNITY EDITOR.....	48
FIGURE 18- CAMERA SET IN PERSPECTIVE VIEW .....	49
FIGURE 19- CAMERA SETTING IN UNITY (WWW.UNITY.COM) .....	49
FIGURE 20- BUILD AND DEPLOYMENT ON PC.....	50
FIGURE 21- EMPTY TERRAIN IN UNITY .....	51
FIGURE 22- TERRAIN SETTING IN UNITY (WWW.UNITY.COM) .....	52
FIGURE 23- TERRAIN WITH MOUNTAINS, TREES AND GRASS.....	52
FIGURE 24- TERRAIN WITH SKYBOX RENDERING .....	53
FIGURE 25- GAME WITH FIRST PERSON CHARACTER CONTROLLER.....	54

# List of Tables

TABLE 1- A SAMPLING OF OPENGL EXTENSION PREFIXES (WRIGHT , ET AL., 2010).....	16
TABLE 2- OTHER 3D MODELING SYSTEM SOFTWARE.....	33
TABLE 3- COST COMPARISON OF GAME ENGINES.....	39