ABSTRACT

This paper details the intricacies of the components of a 3D engine and explores the use of these engines in the world today. The paper discusses what you will find in the code of an engine, as well as the two types of code in the marketplace today. Some of the specifics of graphics are covered, and an overview of the different genres of games is provided. The paper finishes up with a look at two of the most popular 3D game engines and a summary of my proposed future work.

KEY WORDS

3D, game engines, video games

1. INTRODUCTION

Video games have become an ubiquitous presence in houses, universities, companies and virtually any other place where people and electronic devices are brought together all across the world. For the player, how the game looks, feels and plays is of the utmost importance. Meanwhile, for the game designer and programmer, the most important part of a video game is the game engine - the computer application or program component that can be reused over and over in order to assist with the programming and developing of games. The most complex form of game development engine is the 3D engine, which is the focus of this research. Through the use of a 3D engine, game developers save money, time, and resources, and are able to more rapidly produce sophisticated games of higher quality. The inherent flexibility of such game engines means that thousands of different looks and playing behaviors can be produced using the same engine, and it is the underlying technologies and approaches that make that flexibility possible that are the subjects of this paper.

2. CONTENTS OF A 3D GAME ENGINE

Generally speaking, a 3D game design engine first provides two primary repositories for creative materials used in the game being designed: game logic and game art. The game engine is primarily an executable software application that provides an environment for the development and test running of game logic, and for the incorporation of corresponding game art, to ultimately produce a playable game [1]. The game logic contains program code that controls game play behaviors such as dialogue, player movement and boundaries, weapon behavior and capabilities, and the myriad other elements that combine to make a computer game. The game logic, in turn, tells the game art what it needs to access [1]. For example, if you walk outside a building (the game logic will know up to what point is inside) the graphics will inevitably change. Planning this schematic would look something like an algorithmic tree with “World” as the top node, which then splits off to “Indoors,” “Outdoors,” etc. [1].

Although it may not seem like the game logic deals with graphics and 3D elements, it certainly does. Most of the math that is required to display the complex world comes from the logic portion of the engine.

2.1 GRAPHIC API’S

Game engines link up with Graphic Application Programming Interfaces, or API’s, to more powerfully interact with the game software. API’s contain specific commands for different graphic rendering strategies, and thus, greatly help the graphic designer [2]. As you can see from the graph below, the graphics engine, on level 1, makes use of the Game API on level 2. This is the layer which defines the high-level characteristics of a game engine [2].

![Figure 1: Game Engine Hierarchy][2]
2.1.1 DIRECT 3D

One of the two most highly recognized and used Graphic API’s is Microsoft’s Direct3D, part of the DirectX family of API’s. Microsoft consistently works on updates to provide new capabilities and functions for developers. This leads to the constant improvement of graphics we see through the years. Designed mainly as a 3D hardware interface for the Windows operating system, Direct 3D has been criticized for its proprietary nature [3]. Though, in recent years, it has become a very clean API to use and is widely accepted across the industry. The version currently being used by most new games is Direct 3D 10, and this version is explored in detail in Blythe’s paper [3].

2.1.2 OPENGL

As aforementioned, the main competitor to Direct 3D in the Graphical API market is OpenGL. As opposed to the hardware focuses of Direct3D, OpenGL is a 3D rendering system that makes use of hardware if it is present. It is known to be easy to use, though it has the drawback of being entirely in C [2]. In the end, which ever API is chosen, the desired functionalities are essentially the same.

3. OPEN-SOURCE VS CLOSED-SOURCE

There are two main variations of code for 3D game engines – open, or closed, source. Open-source engines refer to the software that is available for anyone to use, usually found online somewhere. An open-source engine can range from something made by an amateur in a basement, to code of a once-popular game that the developers have now decided to reveal [4]. A major advantage of working with open-source engines is that you have a huge amount of manpower behind the code, since anyone across the web can step through, work, and provide feedback with it. Possible disadvantages are obvious, such as working with bugged code [4]. In the end, you get what you pay for.

Closed-source code, on the other hand, is usually much more innovative and powerful. Not revealed to the public because the engine usually contains new technology and is almost always fully functional, these are the engines of games such as Left 4 Dead, or Crysis [4].

4. EVOLUTION OF GRAPHICS

Similar to any other technology in the world today, graphics in 3D engines have progressed rapidly. Early games such as Doom used very simple, flat textural 2.5-dimensional graphics to build characters and the world. This refers to a sort of pseudo 3D look, in which the graphics are meant to look 3D, but are actually only on two planes [5]. All of the graphics were software-rendered, meaning operating system and game software did all of the calculations and processing [5].

As time progressed, hardware-accelerated graphics started to appear. GPU chips, or graphical processing units which sat in the motherboard of the computer, hit the market and provided games with more robust graphics because of the faster processing speed. Soon, games like Doom 3 used per-pixel shading instead of shading all the pixels as a whole. Where we are now, games like Crysis are almost photorealistic [5].
Another technique used in graphic design across the industry is “bump mapping.” This refers to when a certain texture is applied to a simple shape or graphic that makes it look as if it has more depth and character [2]. This is clearly evident in the following two pictures.

Figure 5 – Bump mapping on the right ball

5. DIFFERENT GAME GENRE’S

The video games in the market today focus on many different styles of play. The three most popular and mainstream, however, are the First-Person Shooter, the Real-Time Strategy, and the Role-playing game. Each of these game types offers a different outlook upon graphics engines and their development.

5.1 FIRST-PERSON SHOOTER

Most popular games are from this genre. They consist of a view from the eyes of the main character of the game who is usually wielding some type of weapon. The focus of the graphics is on the weapons and other NPC’s, or non-player characters [4]. Some of the unique graphical features in an FPS include blurring of vision when changing view, and particle effects of weapons that must be highly detailed [4]. For example, a good game will have an interesting difference between what emits from a plasma gun versus a lightning gun.

Figure 6 – First-Person Shooter

5.2 REAL-TIME STRATEGY

Real-Time Strategy games allow the user to take control of a certain amount of units over a small or large map. The user typically has a view from above a 2D, or more recently, 3D map. Up until a few years ago, almost every RTS used 2D units, and the player was not able to zoom in to watch the action [5]. However, these game engines have advanced so much that you can zoom so far into the action that an RTS can sometimes be confused for an FPS.

Figure 7 – StarCraft - 1998

5.3 ROLE-PLAYING

Role-playing games attempt to turn the user into an in-game character who makes fictional decisions. These games usually employ a third-person view and the graphical focus is on the characters armor, and other characters, either evil or friendly [4]. One example is the online game World of Warcraft, a game not known for its particularly leading-edge graphics, but the art design has won it many awards.

Figure 8 – World of Warcraft

6.0 POPULAR GAME ENGINES

There are two particular 3d game engines that have been around for a while and have produced numerous popular games. One is the Unreal Engine, used in the whole Unreal series, and the other is the Source Engine, used by the company Valve to create classics like Half-Life 2.

6.1 SOURCE ENGINE

Valve’s engine uses the latest version of Direct3D and the Havok Physics engine which employs a top-of-the-line dynamic collision system. The engine is most known for its facial animation system which utilizes a professor’s
studies of face movement [6]. Steam uses C++ for programming and is known as a very dynamic engine, changing every few months to incorporate new technology [6]. Figures 9 and 10 below, of two different games, demonstrate the range in power of the source engine.

6.2 UNREAL ENGINE

The Unreal Engine, another FPS engine, has been around since the late ‘90’s, using an ancient Glide API during its originating years. Switching over to Direct3D since, Unreal has made its presence known in the industry with Natural Motion, a type of character animation capture used in animated movies, and Nvidia Physx, or physics powered by Nvidia GPU’s [7].

7. PROPOSED WORK

Considering my father to be an architect, I spoke with the lead 3D computer design employee in his company. He is constantly creating graphical representations on the computer for the structures and interiors that my father designs. Additionally, he used to work for Acclaim video games, and has access to certain small game engines. Therefore, we came up with the idea of creating a 3D environment from an existing engine. Hopefully, with his help, I can modify either a video game engine or an existing model which he has already created, and experiment with it to change it into something more of my taste. I will be working directly with an engine, and therefore my knowledge and scope of them will increase. I plan to show my version of an interactive 3D environment at a small annual tech show at my public library in January.

The work I complete on this project will be significant for a few reasons. First, the knowledge I gain from direct work with a game engine is second to none. Although I learned plenty of the theory behind video games and their engines through this research, this project will provide me with tangible work and results. Second, I can hope to teach others a little something about the creation of graphics and video games at the tech show. Lastly, if I am successful enough, my Uncle, who is a computer science professor at a local college, will demonstrate my project in class. If I make it to this point, only the sky is the limit.

I was first introduced to this idea of an interactive environment when I was searching through research papers. I came across tons of scholarly papers in which people set out to do the same thing I plan to do. For example, in Kot, Wuensche, Grundy, and Hosking’s paper, these students successfully integrate source code files as monsters in a game that you can interact with [4]. Each monster is a different source file, and you can view that file simply by walking up to the monster. They come to the conclusions that reusing a game engine for the development of an information visualization tool saves a lot of time, and that modifying an existing game is much simpler than writing new source code to work with a graphics engine [4]. In the picture on the left, Figure 12, if you walk into the object on the right, which represents a large .c file, you will see the source code visible in the picture to the right, Figure 13 [4].

In addition, I found another paper in which the authors create an archaeological site from a game engine. They discuss the different facets of different types of engines, and which one fit their goals the best. Once again, they harp on the effectiveness and economic advantage to using an engine instead of starting from scratch [2].

Since I will be working with a professional on the project, I am not certain of every aspect that the project will entail, but we will be using C and C++ programming languages, and AutoCAD (graphical design program used by many architects) will be involved. Although most of my courses in Villanova have not focused on the nature of this project, it will probably be difficult at first. However, with the knowledge of C programming, as well as the help of a professional from the game and architecture industry, I hope to learn quickly. Also to my benefit, I had a week-long tutorial on computer graphics this past summer.
Although my schedule will be somewhat tight, I hope to learn, work, and be prepared to present my project by the January 30th tech show. I will spend a week when I first get home for Christmas familiarizing myself with the programs involved, and hope to work quickly from there. My schedule will hopefully look something like this:

<table>
<thead>
<tr>
<th>Week</th>
<th>Task</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Familiarize with programs/techniques</td>
</tr>
<tr>
<td>2-3</td>
<td>Organize and begin plan with Professional</td>
</tr>
<tr>
<td>3-4</td>
<td>Complete bulk of work</td>
</tr>
<tr>
<td>5-6</td>
<td>Optimize project and present</td>
</tr>
</tbody>
</table>

8. CONCLUSIONS

Clearly the video game industry is very intricate, and lying beneath the whole industry is the formation and use of 3D game engines. These engines have simple underlying structures but can produce extremely powerful graphics. There are several genres of games, and each has its own unique ideas and focuses in design. As seen above, one graphics engine can produce radically different looking and performing games. Without these engines, games would take enormous amounts of time to produce, and we would not see as much variation across the different titles as we do now. Hopefully through my proposed project, I will learn even more about the software that powers thousands of 3D video games.

9. REFERENCES


