4-1-2011

Adopting Game Technology for Architectural Visualization

Scott A. Schroeder
Purdue University, scottsc@alumni.purdue.edu

Graduate School ETD Form 9
(Revised 12/07)

PURDUE UNIVERSITY
GRADUATE SCHOOL
Thesis/Dissertation Acceptance

This is to certify that the thesis/dissertation prepared

By Scott A. Schroeder

Entitled
Adopting Game Technology for Architectural Visualization

For the degree of Master of Science

Is approved by the final examining committee:

Clark A. Cory
Chair

Phillip S. Dunston

Nicoletta Adamo-Villiani

To the best of my knowledge and as understood by the student in the *Research Integrity and Copyright Disclaimer (Graduate School Form 20)*, this thesis/dissertation adheres to the provisions of Purdue University's "Policy on Integrity in Research" and the use of copyrighted material.

Approved by Major Professor(s): Clark A. Cory

Approved by: James L. Mohler 04/19/2011
Head of the Graduate Program Date

Graduate School Form 20
(Revised 9/10)

PURDUE UNIVERSITY
GRADUATE SCHOOL

Research Integrity and Copyright Disclaimer

Title of Thesis/Dissertation:

Adopting Game Technology for Architectural Visualization

For the degree of Master of Science

I certify that in the preparation of this thesis, I have observed the provisions of *Purdue University Executive Memorandum No. C-22,* September 6, 1991, *Policy on Integrity in Research.**

Further, I certify that this work is free of plagiarism and all materials appearing in this thesis/dissertation have been properly quoted and attributed.

I certify that all copyrighted material incorporated into this thesis/dissertation is in compliance with the United States' copyright law and that I have received written permission from the copyright owners for my use of their work, which is beyond the scope of the law. I agree to indemnify and save harmless Purdue University from any and all claims that may be asserted or that may arise from any copyright violation.

Scott A. Schroeder

Printed Name and Signature of Candidate

04/15/2011

Date (month/day/year)

*Located at http://www.purdue.edu/policies/pages/teach_res_outreach/c_22.html

ADOPTING GAME TECHNOLOGY FOR ARCHITECTURAL VISUALIZATION

A Thesis

Submitted to the Faculty

of

Purdue University

by

Scott A. Schroeder

In Partial Fulfillment of the

Requirements for the Degree

of

Master of Science

May 2011

Purdue University

West Lafayette, Indiana

To my family and friends, thank you for the tremendous support during this time.

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Schroeder, Scott A. M.S., Purdue University, May 2011. Adopting Game Technology for Architectural Visualization. Major Professor: Clark A. Cory.

Current methods to display a new home in the architectural visualization industry involve long render times and hundreds of frames that require rendering. Many times, these virtual tours that are produced are slow, methodical, and limit the viewer's perspective of the home. This research looks into using computer game engines to display the virtual tour in real time, thus removing the long render time requirements and limited viewer perspective.

CHAPTER 1. INTRODUCTION

This chapter introduces the motivation behind this research. Included in this chapter is the primary research question as well as several secondary research questions. Also important to this chapter are the scope of the research, assumptions, and limitations.

1.1. Problem Statement

Can current generation game engines such as *Unreal* or *Unity* be leveraged by the architectural visualization industry to create completely interactive, user controlled virtual tours?

1.2. Research Question

Are computer game engines the next step for architectural virtual tours either replacing or in conjunction with pre-rendered still imagery and video?

1.2.1. Secondary

Of the game engines available to the public, which are best suited to create the virtual tour?

1.3. Scope

The research will focus around the various topics related to adopting game engine technology for architectural visualization (arch viz). The topics

include previous research done using game engines, the game engines themselves, and how the user feels connected to the virtual environment. Concurrent with the research, a prototype virtual tour will be created inside the selected game engine(s).

1.4. Significance

This research will expand the technology used by the arch viz industry. The results will add to the types of deliverables an arch viz company can offer its clients as well as the type of deliverable used in mass marketing of a home, condo, or other type of development such as resorts, casinos, parks, and recreational facilities.

1.5. Definitions

AAA game - usually refers to a video game in terms of its large budget, large development team, and a massive PR campaign to sell the game. This does not always refer to a high quality game (Juuso, 2009).

Culling - is where surfaces or objects are removed from the rendering process if they are not visible to the camera's point of view (Futuremark, 2010, p.2).

Indie game - usually refers to a video game that has a much smaller budget, small development staff and virtually no PR campaign to sell the game. This does not always refer to a low quality game (Juuso, 2009).

Lightmap - is a texture map that is used to create lighting effects on top of a base texture (Futuremark, 2010, p.3).

Lightmapping - is the process of multiplying the base texture with the light map (Futuremark, 2010, p.3).

1.6. Assumptions

The following assumptions are being made:

- Access to *Unity Pro* will be provided through the Purdue University Envision Center.
- The author's computer is similar to a workstation in industry when used to estimate and compare render times and performance benchmarks.

1.7. Limitations

The study is being conducted with the following limitations:

- The primary testing environment for the virtual tour will be through either a web browser or through an executable file.
- The experience of presence is subjective to the user.

1.8. Delimitations

The following delimitations are acknowledged:

- Ability to put the virtual tour inside an actual sales center is not feasible at this time.

1.9. Chapter Summary

This chapter introduced the research contained within this thesis, outlining the key research questions and variables. Additionally this chapter noted the limitations and delimitations of the chosen scope, and its contribution to the body of knowledge by explaining the significance of the research.

CHAPTER 2. LITERATURE REVIEW

2.1. Introduction

The goal of this research is to start to change the perception noted by Dr. Hudson-Smith of the Centre for Advanced Spatial Analysis at the University College London. Dr Hudson-Smith stated that when referring to game engines in architecture, "It's a niche field. Indeed, it is viewed with suspicion by many, and it is a struggle at times to be taken seriously" (Varney, 2007). Whether creating an architectural tour using pre-rendered stills or movies or for export to a game engine, the process is virtually the same to create the base models. Imagine a union between architectural visualization (arch viz) and today's game engines. Doing so could enable a potential home buyer to take a virtual tour of the exact home they wish to buy, all without ever leaving the sales office or even the comfort of their own home. Imagine the cost savings to a homebuilder if they do not have to build and maintain a physical model home. By combining gaming and arch viz, this is attainable.

Previous research has shown that using game engines to create virtual experiences such as creating virtual museums (Lepouras & Vassilakis, 2004) as well as safely educating laboratory technicians on the dangers of lab accidents (Bell & Folger, 2003) can be extremely effective. In an article written by Allen Varney (2007) discussing the process of recreating London in the *Oblivion* game engine, Varney alluded to the fact that there is a growing movement among some architects to use the photo realistic power of game engines for visualization purposes. That need, combined with the game engines that are readily available on the market today, is what created the interest for furthering this research.

2.2. Game Engines

In the research by Bell and Folger (2003), Lepouras and Vassilakis (2004), and Smith and Trenholme (2008) into the best way to implement virtual reality with a mainstream audience and affordable cost, all came to the same conclusion, that to use video game engines was the most practical approach. While there are many game engines on the market that are readily available, research by Smith and Trenholme (2008) shows that first-person shooter (FPS) game engines generally have more robust features for modifications. There are certainly many more engines to choose from that can be looked at, but the three engines that will be discussed in this research are *CryENGINE 3*, *Unity*, and *Unreal Development Kit* (UDK). All three engines have well documented support as well as large user-based forums that provide additional support and knowledge. All three game engines also easily accept models exported from Autodesk's *3ds Max*, either in the FBX format or in a format specific to the game engine itself.

2.2.1. CryENGINE 3

Crytek's *CryENGINE 3* can produce some of the highest quality graphics of all of the engines available on the market today. *CryENGINE 3* has the ability to compute real-time dynamic global illumination (GI), which means that the engine does not have to pre-compute the GI bounces. All of the GI can be done in real-time on both static (non-moving) and dynamic (moving) objects. The engine also has a unique deferred lighting solution that allows for large numbers of lights to be rendered efficiently. *CryENGINE 3* has full support for high dynamic range (HDR) lighting which increases the range of colors being rendered, thus boosting the realism of the images (CryENGINE 3 Visuals, 2011). Within CryENGINE 3 there is a daylight system that realistically simulates the lighting changes during any time of day, and includes transitions in lighting effects from dawn to night (CryENGINE 3 Sandbox, 2011). To increase the

speed in which visuals are being displayed on the screen, CryENGINE 3 has been optimized to support multi-core CPU's (CryENGINE Performance, 2011).

Since system requirements for *CryENGINE 3* could not be specifically located, the requirements for the game *Crysis 2* will be used instead. A valid assumption can be made that these requirements are similar as *Crysis 2* is one of the first games to showcase *CryENGINE 3*. For the purposes of this research, the requirements listed are the highly recommended requirements and not necessarily the minimum requirements. This is because in the case of arch viz, the end result is displayed on a computer that is custom built to the specifications needed. Another reason for using the highly recommended settings is that these settings allow the engine to use all of its ground breaking technology. If this engine is invested in, the end result should be the best possible. The highly recommended settings are as follows (Visionary, 2011):

- Windows 7 64-bit operating system
- Intel i7 series CPU at least 3GHz or faster
- 4GB System RAM or higher
- NVidia or ATI series graphics card that is DirectX 11 capable and at least 1.8 GB of video memory on the graphics card

However, the quality of this engine comes at a cost of both computing power needed to display the created content and in terms of the cost of licensing the engine itself. When Crytek was contacted to provide an estimate of the different game engine costs, Kathrin Seigmund (personal communication, October 1, 2010) who is a representative from Crytek, could not give an estimate of the licensing fees as they are not public information at this time.

For companies not wishing to have full control over the source code and who are looking for an engine more for visualization purposes, Crytek does offer other license types such as one for serious games and one for cinematics. These licenses are outside the use of a typical game development and offer various levels of access to the source code of the game engine (CryENGINE 3 Licensing, 2011).

2.2.2. Unity

Unity is perhaps one of the more evenly balanced engines as far as cost combined with quality and is readily accessible to any user. *Unity* has fairly basic CPU requirements and video card requirements, meaning that any computer built within the last few years should be able to easily handle *Unity* created applications. *Unity* can also run on a variety of operating systems from Windows to Mac OS (Unity System Requirements, n.d.). However, the more complex the scene being rendered on screen, the higher the system requirements will be.

From the *Unity* website one can download a working version for free and start creating content. In fact, a user can create content for sale with this free version of *Unity*. However, as noted in the end user license agreement (EULA), if a company or person makes in excess of $100,000 in one year with a product created by Unity they must purchase a *Unity Pro* license (Unity EULA, 2010).

The key differences between the *Unity* and *Unity Pro* licenses that would be applicable to an arch viz firm would be that the free version of *Unity* does not have full Umbra object culling support, static mesh combining at render time, full-screen post-processing effects and real time shadows. The *Unity* free version also has a splash screen as well as a watermark on the screen (Unity License Comparisons, 2010). For any company or person looking to use *Unity* in full-scale production, it is best to purchase *Unity Pro* for the onetime cost of $1,500 per license (Unity Store, 2010). Though, it should be noted that a company or group of individuals cannot mix *Unity* and *Unity Pro* licenses (Unity EULA, 2010). This means that if a company were to use *Unity Pro*, every artist who develops content on the *Unity* engine must be using a *Unity Pro* license. A company cannot have a group of artists working on the free version, and then only purchase one license of *Unity Pro* to finalize the lighting, and effects and to compile the final deliverable.

Previously, *Unity* did not include any light mapping engine as *Unreal* does, so a user would have to pre-render a light map out of *3ds Max* or whichever software they are using. However, with the release of *Unity 3 Pro*, the *Beast* light

mapping software will be included at the same $1,500 price quoted above (Unity Store, 2010).

One advantage that *Unity* has over both the *CryENGINE* and *Unreal Development Kit* is that Unity offers additional software to easily create web deployable content. In conjunction to this, Unity also offers an easily installed plug-in on the user's web browser, which allows the content to be played over a website. So, in effect, a potential homebuyer could tour their prospective home from the comfort of their own home or while on their smart phone.

2.2.3. Unreal Development Kit

The *Unreal Development Kit* (UDK) is a fully functional version of the *Unreal Engine 3* that is available for anyone to download from Epic. UDK offers an easy to use, fast rendering, and near photo real GI rendering solution with its *Lightmass* software. Also supported in UDK is the ability to use HDR lighting to further enhance the quality of the lighting solution. UDK is able to run in a 64-bit Windows environment as well as multi-core CPU’s, which gives the end user greater computing power and memory (UDK Features, 2011).

Similar to *CryENGINE 3*, UDK has minimum system requirements but recommended system requirements will be used instead. These requirements will allow for the best quality visuals from the engine to be displayed properly. The recommended system requirements are as follows (UE3, 2010).

- Windows 7 64-bit operating system
- Any multi-core CPU at 2 GHz or faster
- 4 GB system RAM or higher
- NVIDIA 8000 series or ATI equivilant and higher graphics card with at least 512 Mb of video memory.

UDK can be used for free if issued for nonprofit or educational use. However, for most arch viz firms, they would fit into the $99 royalty bearing license fee that Epic offers would apply. A company would pay $99 for each UDK

license. For the first $50,000 earned using UDK, the company would pay no royalties to Epic. After the initial $50,000 earned, the company would then pay Epic a 25% royalty fee on any subsequent income (UDK EULA, 2011). Though Epic recommends that if a company has earned or plans to earn over $250,000 to contact Epic and inquire about purchasing licenses of UDK that would not require royalty fees.

2.2.4. Game Engine Comparison

Table 2.1 compares the hardware requirements for all three of the previously mentioned game engines. When *Unity* is listed as N/A, for not available, that means that the specifications are far too broad to list. As noted in Section 2.2.2, *Unity* has a wide range of systems it can be configured to run on. *Unity* also can run on a Mac, however since both *CryENGINE* and UDK cannot run on a Mac, only the Windows operating system is listed for *Unity*.

Table 2.1.

Comparison of Game Engine System Requirements

	CryEngine 3	Unity	UDK
Operating System	Windows 7 64-bit	Windows XP SP2	Windows 7 64-bit
CPU	Intel i7 3 GHz	N/A	Multi-core at 2 GHz
Memory (RAM)	3 GB	N/A	3 GB
Video (RAM)	1.8 GB	N/A	512 MB
DirectX Support	DirectX 9 and 11	DirectX 9	DirectX 9 and 11

2.3. User Experience

Virtual home tours are meant to be accepted by the buyer in lieu of an actual tour of a home, so it is important to look at the factors that will help the user experience the virtual tour in order to make them feel as if they are actually

walking through their future home. Important elements in providing a compelling virtual tour are presence, navigation, and display.

2.3.1. Presence

Presence is defined as the subjective experience of being there, and is a psychological phenomenon that resides in the perceptions of the user (Bostan, 2009). The feeling of presence, facilitated by virtual reality, can lead the user to think what they are experiencing is real. This is important because people tend to remember things they experience firsthand over things they simply read (Kolb, 1984). Previous research and development of an interactive training simulator to help prevent laboratory accidents using the *Half-Life* game engine showed the concept of using game engines to help users experience situations that may not be practical or safe to experience in the real world (Bell & Folger, 2003). While touring a home in the real world is practical, and there are limited safety concerns, this research can apply the concept provided by Bell and Folger to the financial feasibility of virtual homes versus physical model homes.

Given the current economic recession and the impact it has had on the housing industry, many builders are looking to reduce costs. Tom Doucette, owner of Doucette Communities in Arizona, talks about how in one of these communities they are down to a single model home. Doucette also talks about how his communities are looking at using virtual reality as a supplement to the model home (Sullivan, 2010).

To expand on the concept of presence, one can look to the research done by Carrie Heeter (1992) who defined three primary areas of presence: personal presence, social presence, and environmental presence. While social and environmental presences focus on interacting with other people and the surrounding environment respectively, for the virtual home tour our focus is on personal presence which can be defined as the user’s experience inside the virtual world (Bostan, 2009; Heeter, 1992). With the available quality of the game

engines mentioned in the previous section, allowing the user to feel a significant degree of presence within the virtual world is possible.

2.3.2. Navigation

One of the basic ways a user can feel presence is by not having to worry about a complex navigation system. A widely adapted user navigation layout in most first-person games is the WASD and mouse configuration. WASD is controlled as follows: W and S control the user's forward/back movement and S and D control the users' left/right movement. The mouse is used to look around the environment, with the left mouse button controlling the user's interactions with objects. In some cases, the middle mouse wheel can be used to scroll through on-screen options (Clarke & Duimering, 2006).

2.3.3. Display

The quality of how the environment is displayed is an important contribution to how a user feels presence (Bostan, 2009). From a technical standpoint, there needs to be a computer configured to run the virtual environment smoothly. If any of the engines are invested in, there needs to be an equal investment into a computer than can properly display the virtual tour. With all of the game engines mentioned, they provide exact specifications for hardware and system requirements as shown in Table 2.1, so there should not be any issues with poor quality display. *Unity* and *UDK* can run smoothly on a fairly basic configured computer, while *CryENGINE* requires a much more robust computer to be able to run smoothly. The monitor being used for display should be large enough in order for any user to be able to clearly see the tour. With many large screen TVs supporting computer input; this may be the best option.

2.3.4. Information Display

While display includes how well the environment looks on screen, it also includes how information is displayed within the virtual tour. Users can often get annoyed or discouraged if information is displayed poorly on screen. This is known as *visual obstruction* and it is when information displayed on the screen gets in the way of what the user wants to view, thusly hindering the overall tour experience (Clarke & Duimering, 2006). Based on user feedback gathered by Clarke and Duimering (2006) there are several design factors to consider when displaying information on screen. They are as follows:

- Use of translucent information boxes.
- Use of voice recordings rather than text boxes on screen.
- Preference for information displayed at the bottom of the screen rather than on the top, sides, or center of the screen.

2.3.5. Visual Discomfort

Visual discomfort is one of the largest challenges in viewing virtual reality. Typically at home or even a sales office there will be smaller audiences with shorter distances between the viewers and the screen (Blondé, Doyen, & Borel, 2010). The lighting of the area where the display is located must also be considered. If the area is brightly lit, then the pupils of the human eye contract resulting in increased depth of focus in the eye (Blondé, et. al., 2010). All of these factors can lead to vergence vs. accommodation decoupling, which in basic terms means 3D objects are not portrayed correctly to the screen. This phenomenon is often cited as a primary cause of visual discomfort and fatigue (Blondé, et. al., 2010).

Visual discomfort, or simulator sickness, is important to understand as it is a feeling that is similar to motion sickness that one would experience from being on a boat, car, or plane. Often symptoms are as mild as nausea, but can also lead to vomiting. While the reasons why this happens are still relatively unknown,

researchers think that the brain thinks that it is hallucinating. Since hallucinations are often a sign of poison in the body, the brain tells the body to purge (Johnson, 2009). While there are no concrete numbers regarding how many people could suffer from simulator sickness. A study has been done by the United States Army in which the researchers found that almost half of the pilots who train on simulators felt the effects of simulator sickness, with the most frequent symptom being nausea (Johnson, 2005). Simulator sickness induced nausea and in extreme cases, vomiting, obviously would not be an ideal occurance for a sales center. These effects should not be a deterrent to use virtual reality in a sales center, but instead used to educate and research the proper configuration to minimize the discomfort.

2.4. Prototype Performance Requirements

Once the virtual tour prototype is completed, it will need to be evaluated based on criteria that directly affect the overall performance of the virtual tour. According to Claypool, Claypool, and Damaa (2006) the rates and resolution of frames rendered in a game directly influence the game's playability and enjoyment. Further research by Claypool and Claypool (2009) suggest that the rate the frames are rendered is more important than the resolution at which the frames are rendered. Data collected by Claypool et al. (2006) showed that there was a significant increase in player performance at 30 frames per second (FPS) over lower frame rates of 15, 7, and 3 FPS respectively. However, when the frame rate increased from 30 FPS to 60 FPS, the performance was only slightly improved.

Looking at the video game industry for examples, there seems to be split decisions on 60 FPS and 30 FPS as a standard. In a May 2010 interview about the upcoming game *Call of Duty: Black Ops*, studio head Mark Lamia of Treyarch mentions that this game will run at 60 FPS (GameTrailers, 2010). Another widely anticipated game, *Gears of War 3,* will have a target of 30 FPS. Cliff Bleszinski who is the design director for Epic Games talks about why *Gears of War 3* is

targeting 30 FPS, “We’d rather put extra visuals on the screen than have 60 FPS” (Gaskill, 2010). What Bleszinski is referring to is opting to not have a faster frame rate at the expense of lower quality visuals displayed on the screen. Data collected by Claypool et al. (2006) shows that in user perception, there isn’t much difference between 30 and 60 FPS, but there are noticeable differences at speeds lower than 30 FPS.

Since these virtual tours will present a complex scene in terms of polygon counts of the geometry to display, how the game engine processes the geometry will impact the frame rate. One way game engines process complex scenes is only rendering the geometry that is visible to the camera's field of view (Mulloni, Nadakutti, & Chittaro, 2007). The software controlling what geometry is being rendered based on its visibility to the camera is known as culling (Futuremark, 2010). An example is if the virtual tour's camera is on the first floor of a home, the second floor geometry would not be processed by the game engine since it is not currently being seen by the user.

2.5. Chapter Summary

This chapter summarized existing research on using available game engines as a low cost solution for virtual reality. Three of the latest state of the art game engines were reviewed and compared, as well as providing basic cost analysis of each engine. Additionally, since the product being created is destined for use outside an academic environment, licensing issues were also addressed. The concept of presence, and how the user experiences, navigates, and views the virtual tour were reviewed as well as ways to enhance the user's overall experience. Finally, there was research presented on the phenomenon of simulator sickness which can have adverse effects on a user's experience. The next chapter will introduce the methodology that will be used to test the virtual tour.

CHAPTER 3. METHODOLOGY

This chapter provides a summary of the prototype creation, research framework, and analysis methods used in this study.

3.1. Prototype Creation Methodology

The process to create the virtual tour was set up to resemble a typical production pipeline that is used in industry. The primary steps in this process are as follows:

- Design CAD received, typically in *AutoCAD* format.
- CAD imported into *Autodesk 3DS Max* to create the 3D space.
- Texturing and lighting applied to the 3D space.
- 3D space exported from *3ds Max* using the FBX format.

3.1.1. Virtual Tour House

The house that will be used for the virtual tour is representative of a typical single family home. The total square footage of the house is 1,264 square feet, which includes 3 bedrooms and 2 bathrooms. The layout of the home is shown in Figure 3.1.

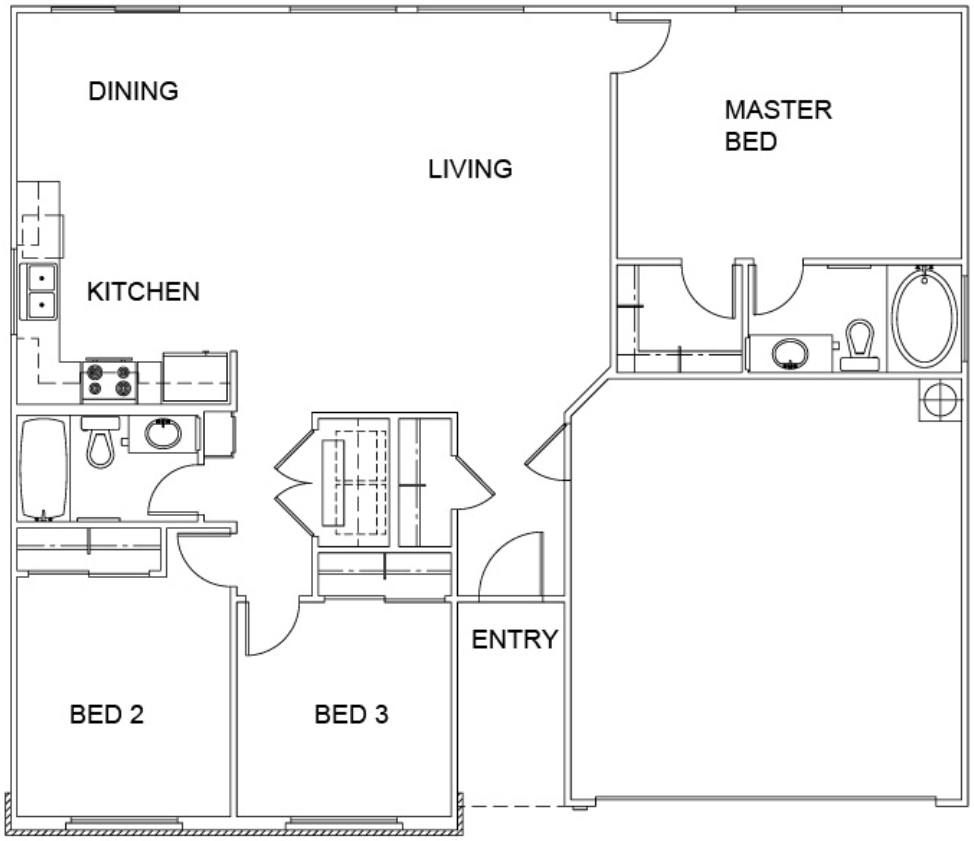

Figure 3.1. The layout of the home that will be used in the virtual tour.

3.2. Choosing the Game Engine

The game engines presented in Section 2.2 each have strengths that are suited to use in creation of the virtual tour. However, both the *Unity* engine and UDK engine have the right balance of quality and affordability. As shown in Section 2.2.2, a person can download functional versions of UDK and *Unity* for free and start to create profitable applications. Therefore, prototypes will be developed using these two game engines.

3.2.1. Game Engine Testing Environment

The computing environment that will be used to evaluate the engines will be similar to that found in industry. The specifications of the chosen computer are as follows:

- Intel i7 Quad Core 2.93 MHz
- 8 gigabytes of DDR3 SDRAM at 1066Mhz
- ATI Radeon HD 4850 video card
- Windows 7 Professional
- 21” LCD monitor with a resolution of 1400x900

3.3. Measures of Success

The primary measures of success will be the engine’s ability to import objects created from 3ds Max efficiently and without loss of information to the objects. How the engines handle materials and lighting will also be evaluated, as well as the general ease of use of the engines. Another measure of success is how the engine performs in displaying the virtual tour. Therefore the FPS will be measured. As cited in Section 2.4 the target values are between 30 and 60 FPS.

3.4. Chapter Summary

This chapter has focused on the steps to create the prototype, the analysis of the game engines, and the comparative analysis of the engines.

CHAPTER 4. PRESENTATION OF DATA

This chapter will cover the results of creating the virtual tour prototype. Topics that will be covered in this chapter include creating content to be imported into the game engines, an in depth analysis of the game engines, and finally the performance results of the virtual tour prototype.

4.1. Content Creation

To create content for a virtual tour, be it for use in an animation or use inside a game engine, the process is virtually the same. The artist always should be aware of creating geometry that adheres to the standards set forth by their production department. While each company or individual artist will differ on specific modeling standards, there are a few universally accepted practices. One major standard is the use of quad sided faces over triangles as quad sided faces are easily translated by the software and provide an optimal surface to perform mesh smoothing (Dillon, 2008).

4.1.1. Textures

Similar to creating 3d models, texture creation is virtually the same process for animation or game engines. There are standards that artists should follow; primarily they should create textures whose dimensions are a power of 2. This means that the texture dimensions should be 256, 512, 1024, or 2048. The reason behind using textures that have dimensions that are a power of 2 is that most graphics cards and rendering engines work with these sizes more efficiently (Birn, 2006). When you are dealing with showing textures in real time, being able

to efficiently handle them is absolutely critical. During the process of creating textures for this project it was found to be easiest to create textures that were square in size, meaning that both dimensions are the same such as 512 x 512. In the event that a texture needed to be non-square, it was still important to retain the power of 2 texture size. For example, if you needed a texture for a background image for your scene and it needed to be longer in pixels than it was high in pixels, the texture could be 2048 x 512.

4.1.2. UVW Unwrap for Lightmapping

One important area of content creation that applies specifically to both game engines is the need to create a lightmap channel. This map always resides in the 2nd map channel of any object and must adhere to several universal standards as shown in Figure 4.1 (Lightmapping UVs, 2010).

- The UVs are contained within the [0,1] x [0,1] UVW space
- No UVs overlap each other
- There is sufficient space between the UVs
- The UVs for the mesh share the same relative size, unless higher resolution information is needed for a specific part of the object.

During this process to set up these lightmap channels, it quickly became apparent that this would be a time consuming task since every object in the scene needed to have this. When one thinks of the sheer number of props inside a typical home, this task became daunting. Even with the use of automated scripts and modeler experience, this phase still represented a substantial time investment. However, it should be noted that many of the props can be reused in future projects so the initial invested time is recuperated in the long term.

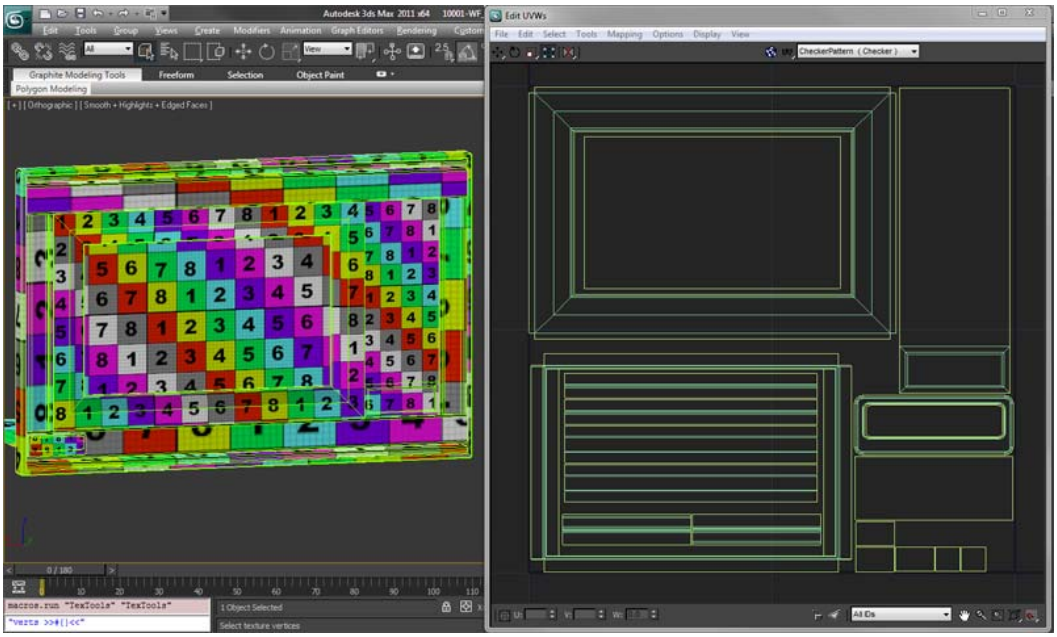

Figure 4.1. Example of a properly unwrapped object for lightmapping. The numbered checkered map shown on the object is there for the artist to check to make sure the mapping is correct.

4.2. Unity Evaluation

Unity was considered a feasible option to develop and create the virtual walk through. A primary reason the Unity engine was chosen was that it can be downloaded for free. A person can also use this free version of Unity to create and distribute applications.

4.2.1. Importing Objects

To import content created in *3ds Max* into *Unity*, it was a simple export and import process using the FBX format. The FBX format is developed by *Autodesk* and allows for near seamless transitions between *Autodesk* products, such as *3ds Max* and *Maya* (Autodesk FBX, 2011). The FBX format is also compatible with third-party applications such as *Unity* and UDK.

There was a difference in scale between *Unity* and *3ds Max*, but this was easily remedied either in the export of the FBX files or inside *Unity* itself. The scale difference came from inside *3ds Max* where the scene was created in feet and inches. This is a general standard for architectural visualization in the United States as most CAD files created are in feet and inches. *3ds Max* can be set to draw in many different scales, so a user is not limited to just feet and inches. However, in difference to *3ds Max*, Unity's scale is set permanently to meters (Vosburg, 2009). This difference is easily remedied in two ways. One way is to set the scale to meters when exporting the FBX file. The FBX export dialog has a section where a user can specify the scale to which to convert the file. Another way is to export the FBX file using the default scale of *3ds Max* and then change the scale of the objects inside *Unity*.

While the exporting process is straight forward, there are some caveats that must be addressed. One of these is the direction of the surface normal inside *3ds Max*. If the normal is inverted, while this may look correct inside *3ds Max* or in a rendered image, depending on your settings, when imported into *Unity* the object will appear to be invisible until you look at the object from the opposite direction (see Figure 4.2). A simple fix for this is to enable the backface cull option inside *3ds Max* which will allow the user to see the object as you would inside *Unity*. If the surface normal is facing the wrong way, the artists can simply flip the normal's direction using tools inside *3ds Max*. Once inside Unity there is no option to flip the normal direction, so the artist must ensure that the normal is correct before starting the export process.

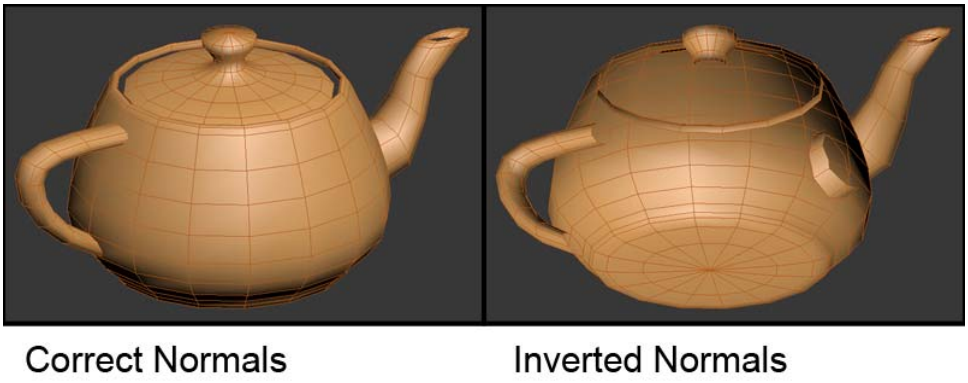

Figure 4.2. Examples of correct and inverted normals.

Another issue that arose during the exporting and importing process is when dealing with the physical scale and pivot point of an object. Many times an object is created either too big or too small, so it is scaled to fit the space. Normally this is fine, as long as the object is kept inside *3ds Max*. However, during the exporting process the scale of the object can be reset. For example, an artist creates a cube that is 10'x10'x10'. Then, upon deciding the cube is too big, the artist scales it down to 5'x5'x5'. As mentioned previously, this is normal practice as long as the object is left inside *3ds Max*. But on occasion during the export process, the box's scale will resize itself back to 10'x10'x10' as the FBX exporting process will remove the scale transform that was applied in *3ds Max*. It should be noted that this does not always occur. If this occurs, there is a simple fix inside *3ds Max*. The artist would select the object and apply a "Reset XForm" modifier. What the Reset XForm modifier does is reset the object's bounding box to the current size of the object. Unfortunately, many times the artist will not know this will occur, especially if they didn't model the object themselves, unless they go through the export process and see a discrepancy with the object's size.

4.2.2. Lighting

During the lighting process of the virtual tour it became apparent that one of the limitations in the free version of *Unity* that was proving to be a hindrance in

the overall workflow was the lack of the light's ability to cast shadows. Only in *Unity Pro* can lights cast any form of shadows. Initially the concept was the render out the light and shadows from *3ds Max* and *Mental Ray* using the render to texture feature, then compare that result with what a person could achieve with *Unity Pro* lighting. However, there seemed to be a discrepancy in what was being rendered and what was actually being saved. What would be rendering on the screen was the correct image in terms of brightness and value of colors. Yet once that image was saved to the hard drive, the brightness and color value was reduced by almost half.

Since *Mental Ray* was being used as the primary rendering system, the lights inside *3ds Max* were *Mental Ray* lights as well as a *Mental Ray* sun and sky system. To use this set up effectively, the artist should set their exposure control to MR Photographic exposure control. However, there appeared to be issues with using MR Photographic exposure control and the render to texture feature as it would display the correct image exposure but not apply that when the image was saved. Any research into this matter was inconclusive, only that others have experienced the same problems. There was no official word from the makers of *3ds Max* as to whether this was an issue they would address in a hotfix or later release.

After research failed to find a viable work around to the problem, the decision was made to only use *Unity Pro* for the duration of the creation process to avoid the issue all together. With the ability to enable shadows on lights inside *Unity Pro* the lighting process became much easier. There are some limitations on which lights can cast shadows depending on which rendering path is being used in *Unity*. The default rendering path is forward rendering. In forward rendering, only directional lights can cast shadows. Spot lights and point lights cannot cast any type of shadow. If the rendering path is changed to deferred lighting, all light types can cast shadows (Unity Rendering Paths, 2010). In order to determine which rendering path was the most effective, a test was run using both types and the average FPS was recorded. The results in Table 4.1 show

that deferred lighting increased the overall average frame rate. Therefore deferred lighting was used for the virtual tour.

Table 4.1.

Comparison of Unity Rendering Paths for Lights Measured by FPS

Rendering Path	Average FPS
Forward rendering	180 FPS
Deferred rendering	240 FPS

4.2.3. Materials

Materials in *Unity* are easy to use and easy to create using the various pre-installed material shaders. However, many of these pre-installed materials work well for only generic needs. There is not a global material shader that works well for any occasion, nor is there a shader that provides a user with a certain level of control that was needed for this virtual tour. After doing some research into material shader creation inside *Unity*, a shader creator was discovered. This was created by one of the members of the *Unity* forums, Stramit (2011), aptly named *Strumpy Shader Editor*. What makes the *Strumpy Shader Editor* so intuitive is that a user can create a custom material shader without having to learn any sort of shader programming language. The *Strumpy Shader Editor* is a graphically based editor where a user simply connects nodes in order to achieve the desired result.

In the case of the virtual tour, three different custom material shaders were created that closely resembled the control given to an artist using the UDK engine. The first shader was a basic shader that allowed the user more control over the lightmaps used to light the material, as well as more control over the specularity and falloff of the highlight on the material. The graph of this shader is shown in Figure 4.3.

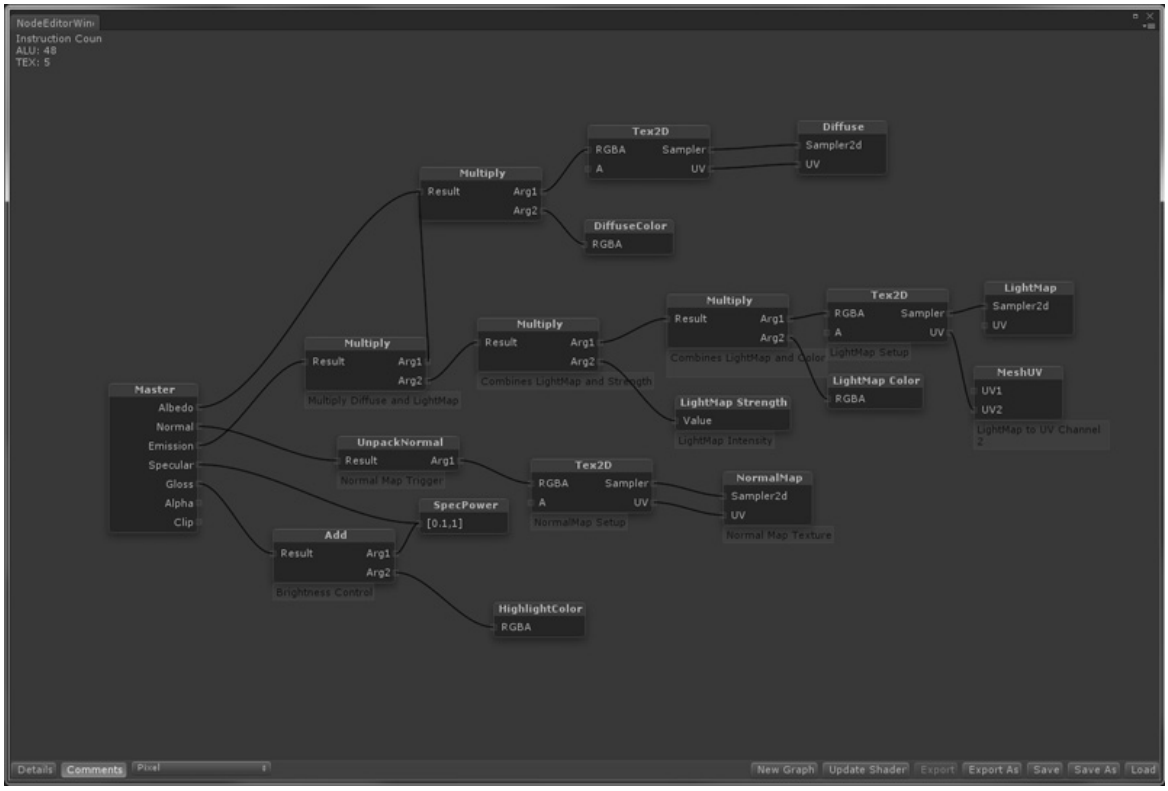

Figure 4.3. The graph of the basic shader inside the *Strumpy Shader Editor.*

The second shader that was created used the first shader as a base and added in the ability for the material to display reflections, for example a chrome material. The reflections were not done in real time in order to keep the virtual tour running smoothly. Instead, reflections were done using a cube map that was rendered from *3ds Max* using *Mental Ray*. Getting the cube map from *3ds Max* was a simple process. All the artist has to do is apply a spherical shader onto the camera in the scene and *Mental Ray* will automatically render out a 360-degree panoramic image that can be easily converted into a cube map inside *Unity* (see Figure 4.4).

Figure 4.4. The reflection map created in *3ds Max* to use inside *Unity*.

The third custom material shader that was created provided the artist the most control of all of the shaders. Not only did the shader provide all of the control of the basic shader, as well as the reflections of the second shader, it also provided the option to mask out certain areas of an object so the reflections would not be shown or would not be as intense. The reason this shader was needed is evident in the kitchen appliances where there are many different types of surfaces from glass, to metal, to chrome. Since these objects were created using one texture sheet (as shown in Figure 4.5), the need to control the reflections was evident. Since the third shader was the most complex (see Figure 4.6), it was also the most computationally intensive shader to run inside the engine so it was only used as needed.

Diffuse Map Reflection Mask Map

Figure 4.5. An example of a diffuse texture sheet and a reflection mask. The white areas will show stronger reflections and the black areas will not show any refection.

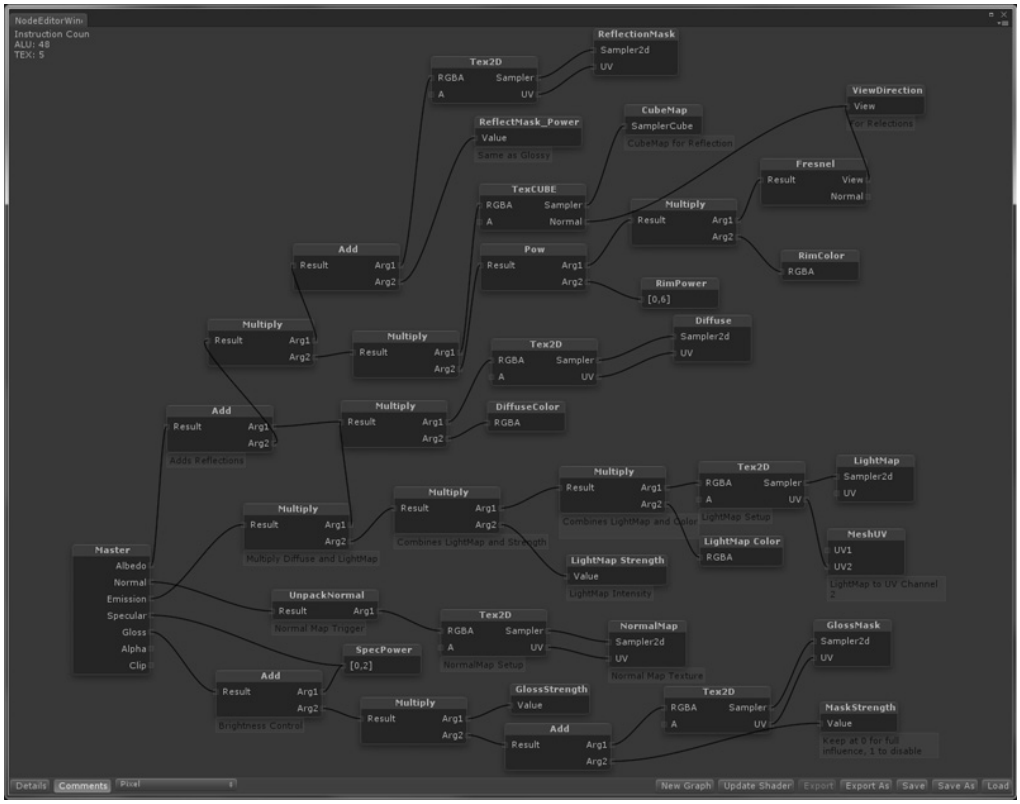

Figure 4.6. The graph of the third and most complex shader

After some experience dealing with the shader, it was found that it would be easier for an object as complex as some of the kitchen appliances to use a multi-sub material that corresponds to the face ID's of the object rather than one complete texture sheet. For example, any face with an ID of 1 would have the chrome material applied to it. Any face with an ID of 2 would have the stainless steel material applied, and so on. Figure 4.7 shows an example of a single mesh with multiple face IDs. Each color represents a new face ID number and this would correspond with the associated material number. For example, the green colored faces are ID number 1, and material 1 is the stainless steel material. The yellow colored faces are ID number 2 and material 2 is the glass material.

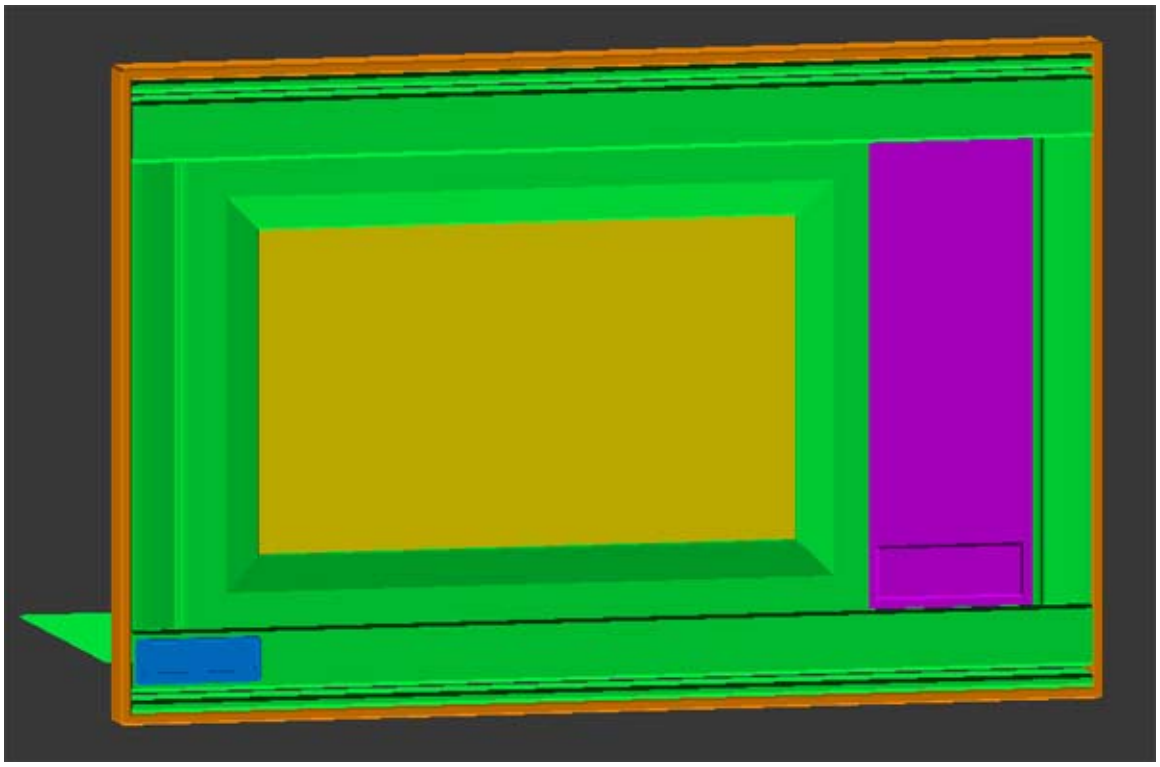

Figure 4.7. An example of a single mesh with multiple face IDs.

4.3. Unreal Development Kit Evaluation

The *Unreal Development Kit* (UDK) was chosen as it is a premiere game engine that is free for anyone to download. Unlike *Unity*, all of the features of UDK are enabled on install. Epic, developer of UDK, provides in-depth documentation and tutorials about all aspects of UDK for free on their website. There are also quite a few training DVDs available from various online schools such as the *Gnomon Workshop* and *Eat 3D* that a user can purchase to further their knowledge of UDK.

4.3.1. Importing Objects

As with *Unity*, UDK can work with the FBX file format. So all of the caveats noted in the *Unity* importing objects review, Section 4.2.1, apply to importing objects into UDK. The only difference for UDK is the scale of the units inside the game engine. While *Unity* was in meters, UDK is closer to centimeters. In fact, 1 unit inside *3ds Max* roughly equals 2 centimeters in UDK. So an artist would apply a scale factor of about 1.75 to an object inside UDK to get it to match the size of the same object in *3ds Max* (Flynt, 2010).

4.3.2. Lighting

Lighting inside UDK is very easy to understand and implement. One important item to include in your scene is creating a lightmass importance volume around your objects. Having a lightmass importance volume is critical to the quality and speed of your lighting solution and, in fact, UDK will warn you that you do not have an importance volume set when you calculate your lighting. The lightmass importance volume controls the areas in which UDK's global illumination engine, *Lightmass*, emits photons. This allows the user to focus the GI solution rather than having the photons bouncing around to infinity inside the game engine (Haines, Wright, & Cornish, n.d.). UDK can also have an object cast light based on the emissive, self illuminated, properties of the light.

In UDK all lights have the ability to cast various types of shadows from static to dynamic shadows. The difference between static and dynamic shadows, in basic terms, is that the static shadows are baked into the lightmap and dynamic shadows are calculated in real time as the engine is running. Since they are running in real time, dynamic shadows are more intensive to display than static shadows. Since the shadows in the virtual tour did not need to move, for example a main character casting a shadow on the ground as they move, all lights were set to cast static shadows.

During the lighting process, an issue with the overall quality of Lightmass started to become apparent. Even setting the light quality to the highest setting and increasing the lightmap resolution to 4096x4096 for an object, there was still a large amount of artifacts seen on the walls and ceilings of the home as shown in Figure 4.8. After doing some research on the Epic Games UDK forum, a solution was found, as shown in Figure 4.9.

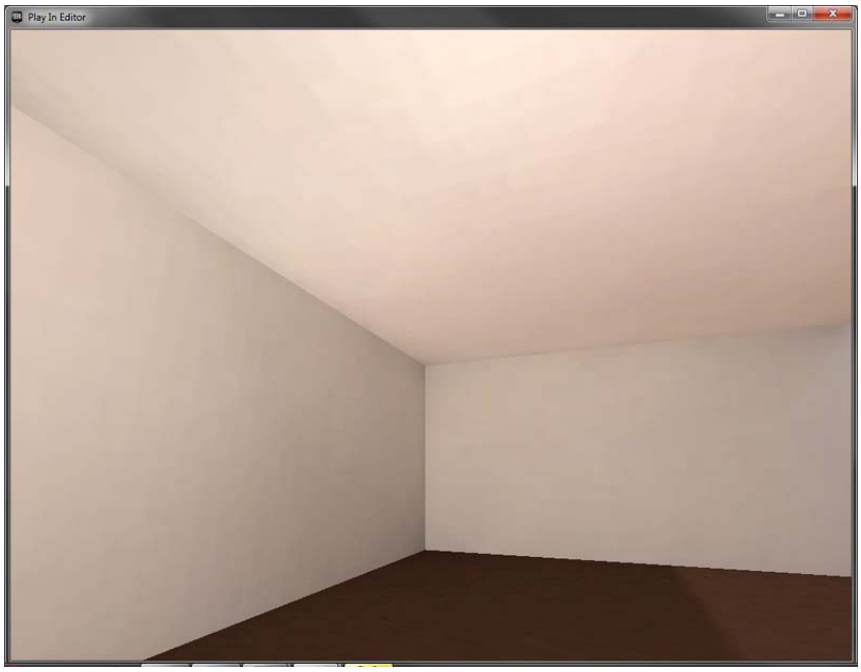

Figure 4.8. An example showing the artifacts created by the lightmap compression. Note the noise on the walls and ceiling.

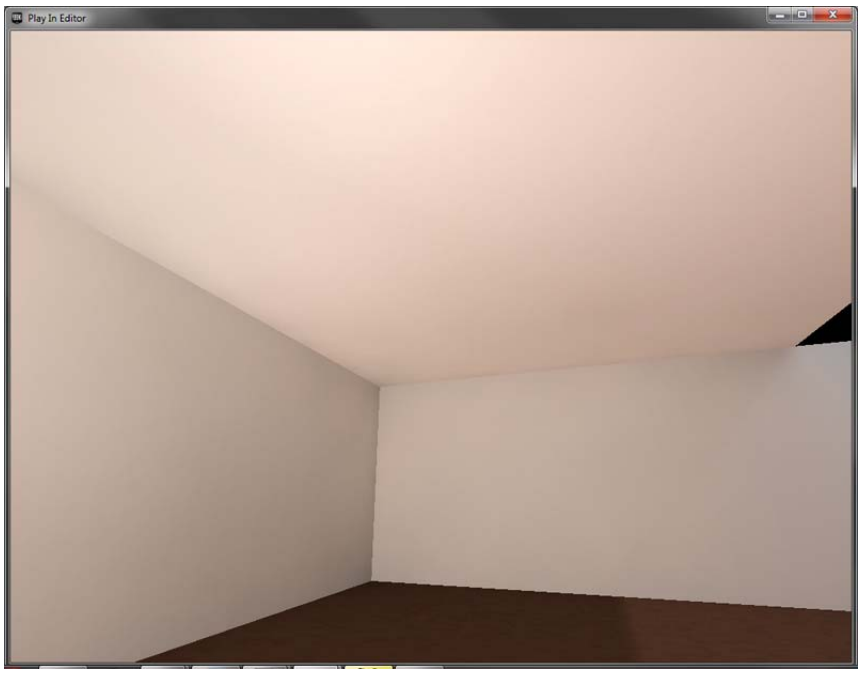

Figure 4.9. An example showing the effect of disabling the lightmap compression. Note the much smoother look on the walls.

What was causing the artifacts was the compression applied to the lightmaps calculated by Lightmass. To resolve this issue, a user simply has to edit the baselightmass.ini file in the folder where UDK is installed and set the lightmap compression from "True" to "False". However, doing so does not come without consequence. Disabling lightmap compression increases the overall file size about 4 times the original size (taz1004, 2010). For example, if the file is six megabytes in size, it will be increased to around twenty-four megabytes if compression is disabled. From various posts on the forums, the only side effect is the file size. There does not seem to be an impact on the performance during display or during rendering, but further research is still needed (taz1004, 2010).

Normally, the compression artifacts are hidden by the textures applied to the surfaces. Many games have materials that have grime, dirt, or something other than just one color applied to it such as a stone or brick material. However, in a home setting, the single color wall is very common as all the wall has is a

paint color applied to it. This single color wall is what magnifies the artifacts as there is nothing to hide them.

4.3.3. Materials

Materials in UDK are a little different than what a person may be used to, but overall creating materials is fairly easy. Given the extensive documentation that Epic has provided on the UDK website, learning the material editor was an easy task. UDK has one shader (see Figure 4.8) that an artist can use to create any sort of material. UDK's materials are more intuitive compared to *Unity* where you are fixed into the pre-installed shaders if you do not write your own custom shader.

It should be noted that UDK does not support images that are in jpeg format. While this may seem odd, it actually follows a fairly logical concept. All of the textures that were imported into UDK were in the Targa format, which includes an alpha channel. This alpha channel can be used to store extra information with the image, such as a bump map or specular map without the need for an extra image. An artist simply loads on Targa texture into the material editor and applies the RGB (red, green, blue) slot to the diffuse slot on the material editor. Then, using the same texture, the artist applies the alpha channel slot to the specular channel as shown in Figure 4.10.

UDK also does not support images that are not dimensioned to a value of power of 2, which was described in Section 4.2.1. UDK will not even allow a user to import the image, and then scale it to the nearest power of 2 size as *Unity* does. This prevents the engine from inadvertently distorting the image as it is scaled to the correct size.

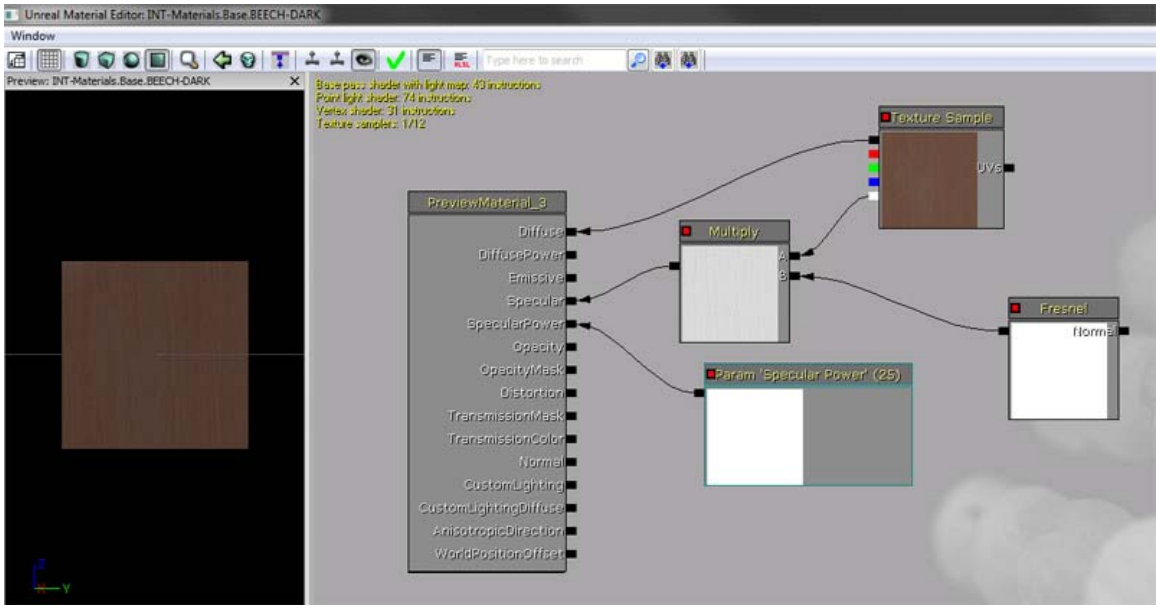

Figure 4.10. An example of the UDK material editor. Note the alpha channel (the white box) of the diffuse texture being used to control the specular level.

4.4. UDK and Unity Performance Results

In the virtual tours created in both UDK and Unity and tested on the computer with the specifications cited in Section 3.2.2, both tours performed beyond what was expected. Both in UDK and Unity, on high quality settings, the frame rate never dropped below 100 FPS. As noted in Section 2.4, many current video games are built to a target of between 30-60 FPS. The tours were tested with a full screen resolution of 1440x900 with dynamic light and shadows enabled on the light used for the sun.

Post-processing effects such as bloom and color correction were also used on the tours. Bloom refers to the soft glow or highlights that brightly lit objects produce. This effect is done to create a softer look and commonly used in film and video games. Both tours used the same geometry and textures to ensure near exact scene composition. Materials were created to be as similar as possible as far as which materials were using specular, normal, and reflection

maps. Neither tour used real-time reflections as these are intensive to render in real-time. If reflections were needed, a cube map reflection was used as shown in Figure 4.4.

Both *Unity* and UDK have diagnostic tools that allow a user to view information about the performance of the content being shown on screen. These tools include a FPS counter that displays the FPS in real-time. Using these tools, data was recorded from the tours and is shown in Tables 4.2 and 4.3. However, during the test in UDK it appeared as if the engine placed a limit on the FPS at no higher than 120 FPS. All throughout the tour during the test the FPS was maxed out at 120, never going above and never dropping below. Which showed that while the tour was performing at higher than 120 FPS, the engine diagnostic tool would only record values of 120 FPS or lower. There were a few posts to the Epic Games online forum about unlocking the FPS, however even after doing the edits suggested the FPS was still maxed out at 120. In actuality, as shown in Section 2.4, as long as the tour is performing between 30-60 FPS, then the result will be perceived as smooth. Anything higher than 60 FPS is nearly impossible for the viewer to notice as shown by Claypool et al. (2006).

In addition to the FPS recorded, the draw time of the frames were also recorded as shown in Table 4.3. Both Unity and UDK list the frame time in the same diagnostic tool that lists the FPS. This measurement is how long it takes the engine to render the frame being displayed on the screen. While FPS is generally a good measure of performance, the frame time is better of an indicator of performance (Dunlop, 2003). However, FPS and frame time are related as frame time is calculated from the FPS. In simple terms, the total number of frames per second is divided by 1,000 milliseconds to calculate the frame time. This means that the lower the FPS, the higher the frame time. The reasoning behind why frame time is a better indicator is that frame time follows a linear path where FPS does not (Dunlop, 2003).

Table 4.2.

Comparison of Frame Rates for Unity and UDK

	Maximum FPS	Lowest FPS	Average FPS
Unity	484	297	340
UDK	120	120	120

Table 4.3.

Comparison of Frame Times in milliseconds for Unity and UDK

	Fastest Time	Slowest Time	Average Time
Unity	2.1 m/s	3.0 m/s	2.5 m/s
UDK	8.3 m/s	8.3 m/s	8.3 m/s

4.5. Chapter Summary

This chapter covered the standards that need to be followed when creating content to import into the game engines, as well as the specifics of creating the lightmap channel. Also included in this chapter were in depth descriptions of how Unity and UDK were used to create the virtual tour. Finally, how the virtual tours created by Unity and UDK performed on the testing computer was discussed.

CHAPTER 5. CONCLUSIONS AND SUMMARY

This chapter provides conclusions based on the results shown in Chapter 4 from the UDK and *Unity* game engines as well as a summary of the thesis.

5.1. Conclusion

Overall, both UDK and *Unity* performed very well in being used to create a virtual tour as well as display the tour. From an “out of the box” standpoint, UDK is the more intuitive solution. A primary example of this is the difference between creating materials in *Unity* and UDK. The pre-installed materials in *Unity* are used for very specific needs, where as UDK has a very global and more robust material editor. UDK also has more professional tutorials available for the new user. This does not mean that *Unity* is lacking in tutorials, but most of that information comes from the users in the *Unity* community. Similar to the materials, the tutorials for *Unity* are very specific and at times are only focused on the exact task at hand. This makes it hard to distill that information into the project that a person in working on, whereas UDK’s tutorials use more global information that a user can then apply to their project.

Ultimately, the choice between UDK and *Unity* would come down to an individual arch viz studio decision. If price were not an issue, UDK would be the engine to choose without a doubt. However, many arch viz studios may not want to invest as much into a game engine, so in this case *Unity* would be the clear choice as *Unity* can deliver an incredibly high quality product for its relatively low cost. If *Unity* is chosen, as shown in Section 4.2.2, the lack of the ability for lights to cast shadows in the free version of *Unity* means that the best choice for an arch viz company would be to purchase *Unity Pro*.

Figure 5.1 shows a comparison between a *3ds Max* render using *Mental Ray* and real-time screenshots from *Unity* and UDK. The render, image A, took approximately 8 minutes to complete using high quality settings that were also optimized for speed as much as possible. The render time for image A includes time to calculate GI. The *Unity* and UDK images, B and C respectively, were extremely close in time taken to render out the lightmaps for the entire scene at about 10 minutes. However, these 10 minutes includes lightmaps for the entire scene and the tour is ready to be displayed in real time. The *3ds Max* render was 8 minutes for a single frame of an animation that was 300 frames long.

To summarize, for the game engines the render time was roughly 10 minutes for both *Unity* and UDK to prepare the scene to be able to tour the house. The *3ds Max* render took 8 minutes for a single image that limits the user's view. If the render were to be used in an animation path, there would be hundreds more frames needed to be rendered therefore increasing the overall render time.

Figure 5.1. Comparisons of three methods used to create imagery for arch viz tours discussed in this research.

5.2. Future Research

This research was focused on setting a foundation for arch viz companies to adapt gaming technology. Further research can be done in the following areas:

- Virtual tours on mobile platforms, such as smart phones or tablet computers.
- Using *Unity*'s ability to create content that is playable in a web browser.
- Using a gaming counsel such as the *Xbox 360* or *Playstation 3* to display the virtual tour.
- Creating functionality into the tour to allow the user to customize finishes, wall colors, appliances, and even furniture styles.
- Rendering video from the game engine. All of the engines allow for real time rendering and how this compares to traditional rendering from *3ds Max* using *Mental Ray*, *Vray*, or similar high quality rendering solution.

5.3. Chapter Summary

This concludes the research into using video game engines to create architectural virtual tours. The primary reason for this research was given in the problem statement and research questions. *Unity* and UDK were given an in-depth analysis as they were used to create a virtual tour. Several conclusions were drawn about the game engines as well as ideas for future research into the application of this technology. The goal of this research was to investigate into expanding the technology used by the arch viz industry as well as provide additional research to the field.

LIST OF REFERENCES

LIST OF REFERENCES

Autodesk FBX. (2011). *Autodesk FBX*. Retrieved from http://usa.autodesk.com/adsk/servlet/pc/index?id=6837478&siteID=123112

Bell, J.T. & Folger, H.S. (2003). Implementing virtual reality laboratory accidents using the Half-Life game engine, WorldUp, and Java3D. *Proceedings of the 2003 American Society for Engineering Education Annual Conference & Exposition.* doi: 10.1.1.86.3654

Blonde, L., Borel, T., Doyen, D. (May, 2010). *3D stereo rendering: New processing & perception challenges.* Retrieved from http://www.3dathome.org/resources-white-papers.aspx

Bostan, B. (2009, February). Requirements analysis of presence: Insights from a RPG game. *ACM Computers in Entertainment*, *7*(1), Article 9. doi: doi.acm.org/10.1145/1486508.1486517

Clarke, D. & Duimering, R.P. (July, 2006). How computer gamers experience the game situation: A behavioral study. *Computers in Entertainment, (CIE)* 4(3). doi: doi.acm.org/10.1145/1146816.1146827

Claypool, M., Claypool, K. (2010). Perspectives, frame rates and resolutions: It's all in the game. *Proceedings of the 4th International Conference on Foundations of Digital Games (ICFDG)*, USA, (pp. 42-49). doi:10.1145/1536513.1536530

Claypool, M., Claypool, K., Damaa, F. (2006). The effects of frame rate and resolution on users playing first person shooter games. *Proceedings of ACM/SPIE Multimedia Computing and Networking (MMCN),* USA, 6071, 607101. doi:10.1.1.98.7665

CryENGINE 3 Licensing. (2011). *Crytek CryENGINE 3.* Retrieved from http://mycryengine.com/index.php?conid=3

CryENGINE 3 Performance. (2011). *Crytek CryENGINE 3*. Retrieved from http://mycryengine.com/index.php?conid=50

CryENGINE 3 Sandbox. (2011). *Crytek CryENGINE 3*. Retrieved from http://mycryengine.com/index.php?conid=53

CryENGINE 3 Visuals. (2011). *Crytek CryENGINE 3*. Retrieved from http://mycryengine.com/index.php?conid=8

Dillon, K. (2008, November 14). 3D mesh topology tip: Quads vs. triangles. Retrieved from http://blendernewbies.blogspot.com/2008/11/3d-mesh-topology-tip-quads-vs-triangles.html

Dunlop, R. (2003, January 22). FPS versus frame time. Retrieved from http://www.mvps.org/directx/articles/fps_versus_frame_time.htm

Haines, S., Wright, D., Cornish, D. (n.d.). Unreal lightmass - Static global illumination for unreal engine 3. Retrieved from http://udn.epicgames.com/Three/Lightmass.html#LightmassImportanceVolume

Flynt. (2010, November 24). Unreal unit. *UDK Central*. Retrieved from http://udkc.info/index.php?title=Unreal_unit

GameTrailers. (2010, May 28). Interview with Treyarch studio head Mark Lamia about the upcoming video game *Call of Duty: Black Ops*. Retrieved from http://www.gametrailers.com/video/e3-2010-call-of/100688

Gaskill, J. (2010, May 6). *Epic sticks with 30FPS for Gears of War 3*. Retrieved from http://g4tv.com/thefeed/blog/post/704506/epic-sticks-with-30fps-for-gears-of-war-3.html

Heeter, C. (1992). Being there: The subjective experience of presence. *Presence: Teloperators and Virtual Environments 1*, 2, 262-271.

Johnson, D. M. (April, 2005). Introduction to and review of simulator sickness research. *U.S. Army Research Institute for the Behavioral and Social Sciences* (Research Report 1832). Retrieved from http://www.hqda.army.mil/ari/pdf/RR%201832.pdf

Johnson, J. (2009). Videogame motion sickness: Tech clinic diagnosis. *Popular Mechanics DIY Tech.* Retrieved from http://www.popularmechanics.com/technology/how-to/4219424

Juuso. (2009, October 24). What are AAA titles? (Updated definition). *Game Producer Blog*. Retrieved from http://www.gameproducer.net/2009/10/24/what-are-aaa-titles-updated-definition/

Lepouras, G., & Vassilakis, C. (2004). Virtual museums for all: Employing game technology for edutainment. *Virtual Reality*, 8(2) doi: 10.1007/s10055-004-0141-1

Lightmapping UVs. (2010, July 1). *Unity*. Retrieved from http://unity3d.com/support/documentation/Manual/LightmappingUV.html

Mulloni, A., Nadalutti, D., & Chittaro, L. (2007). Interactive walkthrough of large 3D models on buildings on mobile devices. *3D technologies for the World Wide Web. Proceedings f the twelfth international conference on 3D web technology, Italy,* 17-25. doi: doi.acm.org/10.1145/1229390.1229393

Smith, S. P., & Trenholme, D. (2008). Computer game engines for developing first-person virtual environments. *Virtual Reality,* 12(3), 181-187. Retrieved from http://dro.dur.ac.uk/5274/

Stramit (2010, August 2). Re: Strumpy Shader Editor. [Online forum comment]. Retrieved from http://forum.unity3d.com/threads/56180-Strumpy-Shader-Editor-4.0a-Massive-Improvements

Sullivan, J. (2010, October 4). Builders explore new approaches to model homes. *Builder Magazine Online.* Retrieved from http://www.builderonline.com/design/builders-explore-new-approaches-to-model-homes.aspx

taz1004 (2010, September 8). Re: Improving Lightmass Quality. [Online forum comment]. Retrieved from http://forums.epicgames.com/showthread.php?t=744190

Visionary. (2011, February 19) Crysis 2 recommended system requirements unveiled. *VR-Zone Technology Beats*. Retrieved from http://vr-zone.com/articles/crysis-2-recommended-system-requirements-unveiled/11255.html

Vosburg, E. (2009, December 28). Re: What is Unity's scale? [Online forum comment]. Retrieved from http://forum.unity3d.com/threads/37833-what-is-Unity-s-scale

Unity End User License Agreement. (2010). *Unity.* Retrieved from http://unity3d.com/unity/unity-end-user-license-3.x

Unity License Comparisons. (2010). *Unity.* Retrieved from http://unity3d.com/unity/licenses

Unity Rendering Paths. (2010). *Unity*. Retrieved from http://unity3d.com/support/documentation/Manual/RenderingPaths.html

Unity System Requirements. (n.d.). *Unity.* Retrieved from http://unity3d.com/unity/system-requirements.html

UDK Features. (2011). Retrieved from http://www.udk.com/features

UE3. (2010). UE3 minimum and recommended specifications. Retrieved from http://udn.epicgames.com/Three/UE3MinSpecs.html

Unreal Development Kit end-user license agreement. (2010). Retrieved from http://www.udk.com/licensing