Game Engines

Features and possibilities

By Bendik Stang 2003

Abstract

This is a project in two parts and was made at the institute of Informatics and Mathematical Modeling at the Technical University of Denmark.

The first part demonstrates that a game engine is a great tool to make a virtual reality presentation. It is argued herein that a game engine is better, cheaper and easier to use than most virtual reality tools available when the project was started.

To demonstrate this a presentation of the new student building, 'Oticon salen' is visualized, based on the information given by the architects and combined with a mockup of the existing building 101 at the Technical University of Denmark. The building was actually built between the time the visualization was made and the time the report was handed in.

The second part of the project shows that the selected game engine can be modified. The argument for using scientific virtual reality tools over a game engine is that the user can implement any changes and or modifications that are required for a certain presentation. In short, virtual reality tools are extremely flexible and since the game engines have been made with one purpose only, they are often deemed less flexible. Given the full source code that is available from the chosen game engine the user is free to implement any changes he/she seems fit.

In the second part I demonstrate how to improve the game engine with some new features. To modify the game engine a new grass impostor system was developed. The grass is fully integrated in the in-game editor and allows for real time customization. The result is a virtual reality environment with a visual grass quality that is higher than anything I have seen on the marked today.

Index

Preface

After having studied Virtual Reality and computer graphics for a few years, I began looking into Game Engines as an alternative to make Virtual Reality presentations, and later on Game Engines to make games.

Figure 1: "The future of computer games is not about reinventing the wheel; it's about improving the engine." – Tim Sweeney, Epic Games, technical mastermind behind the Unreal engine.

Virtual Reality has been the standard term for graphical computer representations in 3D in the science and architecture for a long time.
It has been associated with very high cost and required a higher educational level in informatics. The equipment used has primarily been SGI super computers; needless to say the equipment has been very expensive.

In the academic world there is often a certain level of snobbery, and computer games have been considered unserious by many.

Figure 2: Will you need to spend over US$ 1million?

Figure 3: Will less then US$1000 suffice?

In recent years the commercial game engines have become so advanced that they in most cases outperform the virtual reality engines in all aspects except stereo vision and 3D tracking. Still, I have not yet seen anyone use a Game Engine to create a good Virtual Reality presentation to this day.

As a student, I am trying to learn what I think is useful for my future career. With the current trends on the marked today, getting jobs within any industry is not going to be easy. With this in mind, I am focusing on options that would be ideal for start up or low cost projects. Part of what I am trying to look at in this report is how to get the most out of your investment if you are going to work with some kind of interactive 3D graphics.

I mostly enjoy learning, and sometimes reinventing the wheel is very educational, other times it turns out to be a waste of time and dangerous for motivation. When is it worth it, and when is it better to learn from others, and move on? This question becomes even more important when there are investors that are waiting for results.

I believe there are various applications for a game engine, both for science, visualization, education and entertainment. What tools or engines are available and what would be the best choice to the given task?

What is available at the marked today, how do I find it, and how do I get it?

Once an engine is found, what to do? How much work will be needed? What skills are needed?

While this report has been created several new game engines are under production with even more advanced rendering techniques and physics than the ones mentioned herein. This only strengthens my argument about game engines vs. virtual reality engines.

I began this project with the intent to create an Architectural VR presentation of a new building that was to be built at my university. The work was to be done at the Virtual Reality Center at the Technical University of Denmark. As many fellow students I know, my group began to create our own VR engine, using C++ OpenGL and Performer.

After a while the 'VR-engine' programmers found the work too difficult and tedious. I was mainly focusing on the VR concept and on the 3D graphics at the time, and was lost without them. For a over a year I thought I would have to abandon the project, and get on with something else, but other things brought my attention to the features of game engines, and I began to investigate.

The original project was partly about building a virtual reality system from scratch. Since this path was abandoned the project became a bit shallow and so by improving the chosen game engine I hope to compensate for that.

I will now try to show that I have learned something during this process.

I would like to thank Bent Dalgaard Larsen and Andreas Bærentzen at IMM@DTU who has been most patient and helpful through this project.

I would also like to thank Melvyn May who made the first attempt to create foliage in Torque game engine and whom has answered all of my emails.

I would like to thank the people at Garage Games for releasing a quality game engine like Torque at such a price that independent game developers and poor students like me can afford it.

This project has been carried out under the supervision of Associated professor Niels Jørgen Christensen at IMM at the Technical University of Denmark.

-Bendik Stang Sept 2003

Figure 4: New Student Union @ The Technical University of Denmark

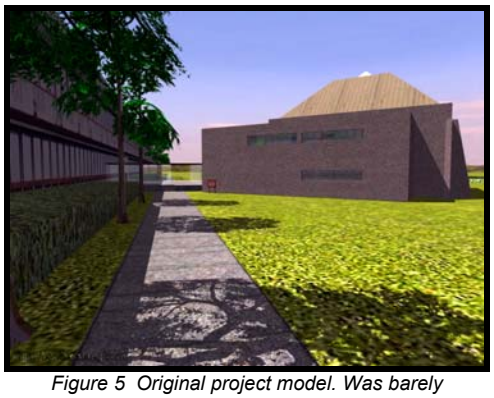


Figure 5 Original project model. Was barely implemented in OpenGL - performer before the project was abandoned

…"A game engine is a very broad thing," says Tim Sweeney, co-founder of Epic Games. "You want to have the world's best rendering, but you also need to have physics, collisions … everything that a game needs."

But the exponential growth of computing power is giving developers more options than ever before, and it's allowing games to expand in new directions. As we examine the future of game engines, we'll see how graphics aren't the only area where computing resources will impact game development. More powerful hardware means more realistic environments, more interactive environments, better opponent AI, and physics that accurately simulate the real world. Technology that'll make today's simple, static game environments look primitive!

-Engines And Engineering
By Steven L. Kent @ GameSpy.com| Oct. 31, 2002

1 Introduction

Choosing an engine
First we will look at Virtual Reality Engines and Computer Game Engines, and compare them in terms of technology, features and price. This work was a bit obscured by the fact that some producers of both game engines and virtual reality engines would not tell me the price of their product to be quoted in this report.

However after doing some extensive searching on the internet and contacting several of the leading game developers in the industry, a decent set of data was acquired that allows for an interesting comparison. From these data a game engine can be chosen and used to create a presentation.

Using the engine
Once a game engine has been selected, a demonstration of the game engine features will be presented showing how to manipulate the virtual reality environment within the system. The game engine features will only be briefly demonstrated in this project.

Next the implementation of the Oticon building will be shown step by step. The process includes the creation of the 3D model, creation of the needed textures, then combining them and placing it all into the virtual world.

Improving the Engine
At the end of the report I will look at the implementation of some openGL code in the chosen Game Engine. Torque from Garage Games will be used as the example engine in which I do the actual implementation.

The use of billboard impostors is well known. The method is quite effective when used on symmetric objects seen at a distance, but the suspension of disbelief is broken the minute the billboards get close to the camera. In the chosen game engine there was no grass but the texture of the terrain as seen on figure 7. After the implementation of the fxGrassReplicator code the landscape is very different (see figure 8).

Figure 6: There is no vegetation code in the original Torque engine.

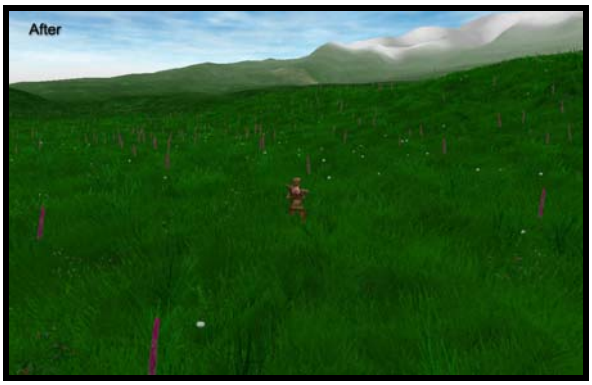


Figure 7: The grass is made of a replication system for billboards (flowers) and grass elements.

The resulting grassy environment is of a more detailed quality than anything seen in another game engine on the marked…..At least for now.

2 What is a game engine?

A game engine is an integrated collection of various computer code objects that together run the video game. These modules include:

- A graphics module for 2D or 3D
- A physics module
- A collision detection module
- An input/output module
- A sound module
- An artificial intelligence module
- A network module
- A database module
- A Graphical User Interface module (GUI)

Different games will have some or all of the above modules. The code that makes up the various modules can be very complex. Creating an industry leading game engine is a huge task, which requires large amounts of time and resources.

Many of the modules contain highly advanced algorithms that in some cases have been developed for military applications, science, medicine or special effects for the film industry. So far the contributing industries have been slow to use the game industry in return.

2.1 Isometric engines

These engines used to be 2D engines. They are currently in a transition face where they are partly 2D and partly 3D and very soon they will be fully 3D. The reason for using fully 3D is that it enables some rather stunning lighting & shadow effects over the 2D sprites. With the new 3D hardware higher visual performance can be obtained with 3D.

Figure 8: DiabloII isometric game engine. (by Blizzard Entertainment.)

These engines are commonly used in strategy games, and were common in the role play games (RPG) genre. There are still relatively new titles on the marked that are based on this kind of engines.

Figure 9: Baldur's Gate 2 isometric engine (by BioWare)

Typical Isometric games include: Baldur's Gate 2, Diablo 2 and Warlords Battle Cry 2.

Isometric engines are not a good alternative for VR.

2.2 3D FPS engines

(First Person Shooter)

This is probably the biggest genre in the gaming industry, and they all have one ting in common; they push the 3D engines to their limits.

Figure 10: Old Doom1 FPS engine (December 10, 1993 by ID software)

Figure 11: New Doom3 FPS engine (not yet released – by ID Software)

As illustrated the quality of the game engines have been improved rather dramatically over the last decade. This is a logical evolution as there are so much money in the gaming industry, and the customers tend to buy the best looking games.

FPS games are always amongst the most sold games.

Typical FPS games include: Doom, Quake, Half Life, Counter Strike, Unreal, Duke Nuke'm, DeusEx, Halo, Wolfenstein, Medal of Honor, Serious Sam, Spec Ops, Dessert Storm and Hitman to mention a few.

The FPS engine can make a great alternative to a VR engine.

It will normally include tools to provide a smooth and easy workflow to create the art and interaction elements. Most of the common 3D modeling applications such as 3DSMax, Maya and Softimage all have exporters that work with the most common of these engines.
This will be explained and demonstrated later in this report.

The state of the art 3D engine today includes features like pixel shaders, bump mapping, cubic environment mapping, full 3d animations with animation blending, physics simulation, particle simulation, cloth simulation, liquid simulation, rag doll simulation and many other things hardly ever seen in a 2D engine.

2.3 MMOG Engines

The difference in a regular game engine and a Massive Multiplayer Online Game Engine (mmog) is based on the network code, and data management. The mmog's normally include large databases, and have a distributed network of servers to handle the huge amount of users active at the same time. Since bandwidth is quite expensive network traffic optimization is one of the most important parts of a good mmog engine. Compression and careful selection of the data transmitted over the internet can save huge amounts of money.

All of the new mmog engines are fully integrated with a state of the art 3D engine comparable with the FPS type engine. So if the application needs a very large amount of simultaneous users, a **mmog engines could be useful for specific types of VR presentations. A v**irtual art gallery is one thing that comes to mind, where people could meet and show each other their work.

Ultima Online, Ever Quest, Asheron's Call and Anarchy Online are examples of massive multiplayer online games.

3 Alternative Game Engine uses

This is what initially led me into looking at game engines. In order to make an architectural VR-presentation I began to study game engine features in comparison to expensive VR tools. The conclusion was that I would be able to sit on my own home PC and develop and test the presentation. Where I was used to fight with other students and faculty to get some time on the supercomputer, I now had the tools to work on this where, and when I wanted, and best of all; The results would be available to most people with a PC, where my previous work was only available to the privileged few with access to a SGI supercomputer.

In my previous work with Virtual Reality presentations I had the experience of working with the Danish Design School (University of Design) on a very artistic virtual reality application. During this process it became obvious that Virtual Reality has a great potential as a new medium for the future Arts. Much is yet there to be discovered.

There are hardcore demo groups that makes fantastic 3D demos where they do all the programming work them selves. This could be the emergence of the new art form. Not yet appreciated by most people.

So in my opinion Game Engines could be used for science, education, entertainment and art.

3.1 VR engine vs. Game Engine

If we look at the development of the game engines and compare that to the development of

the VR-engines there is one major difference. The game engine has and will be created for mainstream personal computers and console platforms, where as the VR engine up till now has been created for a high end system like a SGI super computer. Until right after the turn of the century, the high-end VR-systems outperformed the game systems by being capable of handling several orders of magnitude more polygons, textures and fill rates. The VR input system was and still is, quite a bit more advanced than the average home computer. 3D tracking devices and advanced audio video input are very expensive and fragile, and probably will be for some time.
Still, it was almost disturbing to see the supercomputers immense power fade in comparison to the mainstream graphics cards during 2000 – 2002. Within those two years a US$ 1 million system was outperformed by a US$ 200.- graphics card in many ways. Of course the supercomputer still had other valuable features, like lots of RAM and multiple CPUs, but their days of superior graphics were gone.

The way the graphics processors develop makes it unwise to invest huge amounts of money on high-end graphics processors as a new and much better will make the investment obsolete in less than nine months.

So where does all this lead in relation to the difference between a VR engine and a Game Engine?
The gaming marked is always craving the best visual effects and computer art. This has caused the gaming industry to develop game engines that gets the very maximum out of the available hardware. The competition has been tight, and optimization and quality has been vital for the sales. Huge amount of development money have been put into the development of the various game engines available today.

The VR industry being focused on high-end systems have had a much smaller marked. In many ways a more lucrative marked. When the computer system cost more than a million dollars, the software was possible to sell for tens of thousands of dollars, and there was hardly any competition. This has changed, and prices on relevant software have dropped so rapidly that it is hard to fathom. Examples like Maya Unlimited 4.0 went from approximately USD$ 15,000.- to less than 4,000.- in one month. Same happened to Softimage and other 3D modeling and animation applications.

Such overpricing and lack of competition as well as the enormous computer capabilities may have caused the development of the Virtual Reality Tools to fall behind the accelerating development of competitive commercial game engines for the main stream computers.

Still this is all good news for those that have not invested in the high end systems and software. We are now able to buy better and cheaper tools and hardware and it is even becoming easier to use.

For the low cost development of a VR presentation or computer game, there are some public license engines that are being developed. One would be Crystal Space. The earlier versions of the Quake and Unreal engines are also available for non-commercial uses.

Close to this is a commercial engine that has proven its worth with the game Tribes. The engine was produced by Sierra and sold to Garage Games in 2001. The engine is available at $100.- per developer license. Since the engine is so cheap improvements are being worked on by many independent developers, thus it resembles the development style of an open source environment.

The more expensive engines are available from between $10,000.- to $250,000.- and more and better engines are on their way that will probably come at even higher prices; Doom3 and HalfLife2 to mention two.

The Virtual Reality tools like MultiGen Creator, cost $100.000 little over a year a go.

Price: (These prices were quoted in January 1998.) A Vega Multi-Process development license costs $12,117.16, with $2,154.29 for annual support and $2,420.20 for a run-time license.

3.2 Cost vs. Benefit

In my experience there is little doubt. The multi million dollars worth of VR hardware and software cannot compete with some of the alternative game engines. So when it comes to price vs. Performance, Game Engines win by far. When it comes to ease of implementation – as in getting your idea visualized and presentable, Game Engines win by far.

Considering both production time and cost of software it becomes very clear to me.

4 Choosing an engine

Let's say you have a 3D presentation you would like to make and you would like to do so using a game engine to speed up the process. Which game engine is the most appropriate?

This is a very complex question, and one that I cannot answer fully. However, I have spent a lot of time trying to find an appropriate engine and in doing so I found something that fit my needs.

4.1 Define needs

First, before anything else one has to define the uses for this engine. In my case I was looking for something that I could use at my university to make VR presentations. But since I was going to spend a substantial amount of time familiarizing myself with the chosen engine, I might as well choose one that I would be able to use once I graduated.

For my project I needed an engine and support tools that would enable me to:

- Create a model of a building and apply textures.
- Export the model and textures to the game engine.
- Create a Graphical User Interface that would allow others to easily figure out how to get through my presentation.
- Import an avatar into the project that could be used as a demonstration model and allow the user to use the avatar as a reference for a human.
- Allow me to make the presentation look good and professional.
- Include easy to use shadows and lightmaps. No extra programming necessary would be optimal.
- Allow me to create interactive objects with relative ease.
- A forum or support group that could answer questions and provide help when necessary.
- Allow me to add new code to the engine using C++ and openGL.
- Support sound elements in the 3D structure.
- Preferably run on Windows, Linux, Unix and Macintosh.

Due to time restriction and the extreme prices, any VR system was completely out of the question. If there were no limits to the budget, I might have looked at VEGA from MultiGen but this is not a viable tool in my opinion if I am to use my experience after I graduate.

I spent some time asking around at the various game developer sites on the internet.
The International Game Developers Association – IGDA.org was one of the first places that I went and got a lot of helpful information, as well as Gamasutra.com.

Here are the game engines I have looked at; some more in depth than others.

- Unreal - $10,000
- Quake2 - $10,000
- Quake3 - $250,000
- Torque - $100
- 3D Game Studio – $80
- Genesis - $10,000
- Lithtech - $75,000
- Crystal Space - Free
- Power Render - 5,500

After filtering out the most expensive, and those that were buggy and lacking features I was left with Torque and 3d Game Studio. I chose Torque as I found it the most viable solution for future commercial uses. At $100, or the free student license Torque had all the features needed to make a good solid virtual reality presentation.
It had export tools for several 3D modeling applications, and most importantly, a very easy learning curve.

There are a few other engines that I have not looked into. These are:

- OpenSceneGraph
- Xengine
- NeoEngine
- OpenApp

As new game engines keeps appearing I had to stop the game engine research at one point, choose an engine and move on. I was first aware of the four above mentioned engines at the end of this project and so they have not been compared with the others.

4.2 Overview over engines:

Game Engines	Dark Basic Pro	Quake 1	Unreal	Halflife	Genesis	Nebula	Quake 2	Game Studio	Quake 3	Lichtech 2	Vulpine	**Torque**	Crystal Space	Power Render
Culling system	BSP	BSP	BSP	BSP	BSP	-	BSP	BSP	BSP	Portal	Portal	**BSP**	BSP	Yes
Mipmap	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	**Yes**	Yes	Yes
LOD	-	-	Discr.	-	Discr.	Discr.		Discr.	Discr.	Cont.	Cont.	**Discr.**	-	Cont.
Environment map	Cubic	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	**Yes**	Yes	Yes
Lightmaps	Yes	Yes	Yes	Yes	Yes	-	Yes	Yes	Yes	Yes	Yes	**Yes**	Yes	Yes
Dynamic shadows	Yes	-	Yes	-	Yes	-	-	Yes	-	Yes	Yes	**Yes**	Yes	Yes
Mesh interpolation	-	-	Yes	-	-	Yes	Yes	Yes	Yes	Yes	Yes	**Yes**	-	Yes
Terrain	Yes	-	Yes	-	-	Yes	-	Yes	-	Yes	Yes	**Deform.**	Yes	Deform.
Particle system	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	**Yes**	-	Yes
Mirrors	Yes	-	Yes	-	Yes	-	Yes	Dev	Yes	Yes	Yes	**Dev**	Yes	Yes
Curved surfaces	-	-	Yes	-	-	-	-	-	Yes	Yes	Yes	**-**	Yes	Yes
Shaders	Yes	-	-	-	-	Yes	-	-	Yes	Yes	Yes	**Dev**	-	Yes
Bone animation	Yes	-	Yes	Yes	Yes	Yes	-	-	-	Yes	Yes	**Yes**	-	Yes
Multiplayer	Yes	Yes	Yes	Yes	-	Yes	Yes	Yes	Yes	Yes	Yes	**Yes**	-	Yes
Multisession	-	-	Yes	-	-	-	Yes	Dev	Yes	Yes	Yes	**Yes**	-	Yes
Pysics Engine	-	-	-	-	-	-	-	-	-	Yes	Yes	**Dev**	-	?
Scripting language	-	Basic	C	Basic	C	C		TLC	C++	Java	C++	**C/pyth.**	phyton	C++
Price	$100	GPL	$10,000	?	$10,000		$10,000	$80	$250,000	$75,000		**$100**	GNU	$5,500

The Halflife engine was not for sale and I was not able to obtain a price on Nebula and Vulpine.

5 The Torque Engine

After having considered various engines the choice fell on Torque from GarageGames. The engine had all the features needed. It was cheap and even free for me as a student. The developer community was very alive and most helpful answering all my questions. All this left me comfortable with my choice.

I'm now going to look at the different parts of the Torque game engine and shortly comment the different features, and how they can be used to make an interactive presentation.

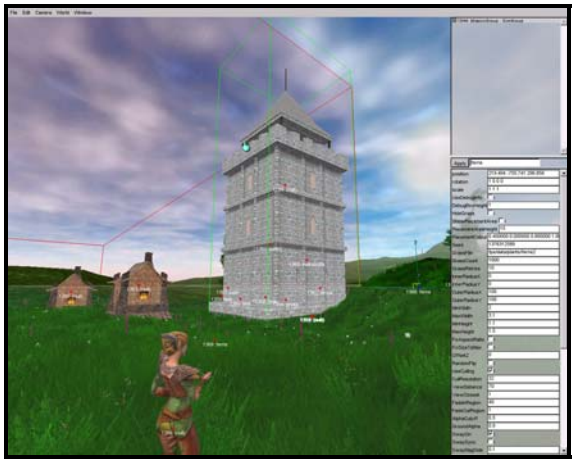


Figure 12: Torque ingame editing tools

5.1 Documentation

At the Garage Games webpage there are three main sections that can be used for reference:

1. **The Torque Documentation pages**. Here you will find information on all the features of the engine and how to install the SDK, make it compile, create contents, scripts etc. An over view is in the Appendix B.

2. **The Torque Resource library**. This contains lots of good tutorials, new export tools, and other helpful tips to get you started.

3. **The Torque Developer Forum**. When all other fails, this is the place to go. Unless the question is rude or plain out silly it will normally get answered in a few hours sometime in only a few minutes.

5.2 Scripting

This is a feature that I have not yet looked at much. It is based on a C++ like syntax and allows you to access almost all game logic. This would be the tool to use to create interactive features in the presentation. Things like doors, triggers, switches, interactive objects, animations, light effects, particles and more can be created or modified using the scripting language. More information on this is available at the garage games website.

5.3 Ingame Editor

Once the SDK has compiled, you can open the demo application. This will put you in control of an avatar in a basic test world with some test objects.

By hitting F1 you can learn how to access the three in-game editors and their tools:

GUI Editor

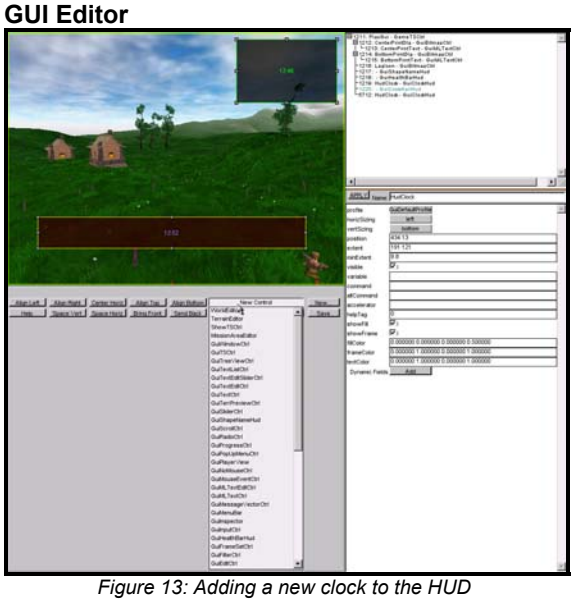


Figure 13: Adding a new clock to the HUD

This editor will allow you to modify or create your own Graphical User Interface (GUI). You can easily include things like a Head up Display (HUD) with compass, target crosshairs, satellite view or other goodies.

You can also create dialog windows that will display text and other information. Like a help menu, login menu, etc.

Mission Editor

The mission editor enables realtime manipulation of most objects in the game. It allows you to add new models such as buildings, trees and plants. You can control the light settings, sky textures, fog thickness and color and much more.

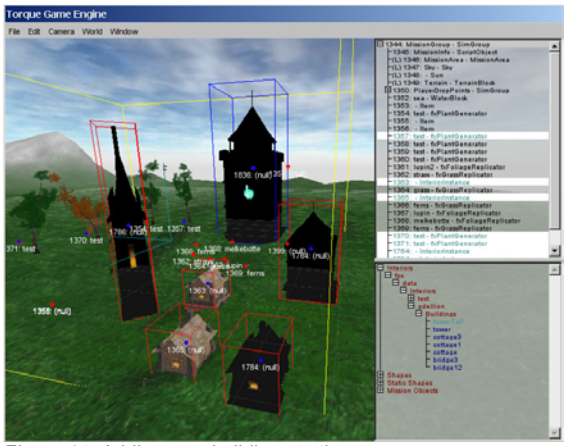


Figure 14: Adding new buildings to the scene.

As you add the new buildings that you have created in e.g. WorldCraft, they appear unlit. That is, their lightmaps has not been yet generated.

You can move the objects around in the scene and rotate and scale them as you like. Duplication of objects is done like in a word processor by the copy paste functions that is used in most programs today.

Once the buildings are in the desired position you can regenerate all lightmaps both on the terrain and on the buildings.

The buildings will cast shadow on each other and on the terrain.

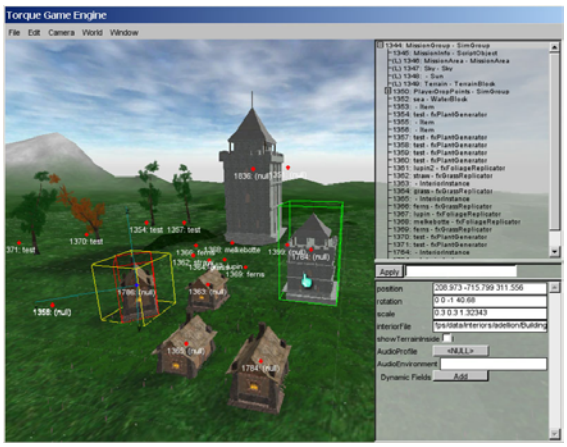


Figure 15: Lightmaps are regenerated (a building is scaled down)

Terrain Editor

This is another useful feature that allows realtime manipulation of the terrain height map.

There are several brushes that can be used for the manipulation, such as excavate, add dirt, adjust height, smooth and flatten. You can set the brush size to many sizes from 1 to 25 terrain-texels in diameter. The Brush can be both smooth and hard for all the features. On the illustration below the mountain was 'pulled up' using an ''adjust height' tool.

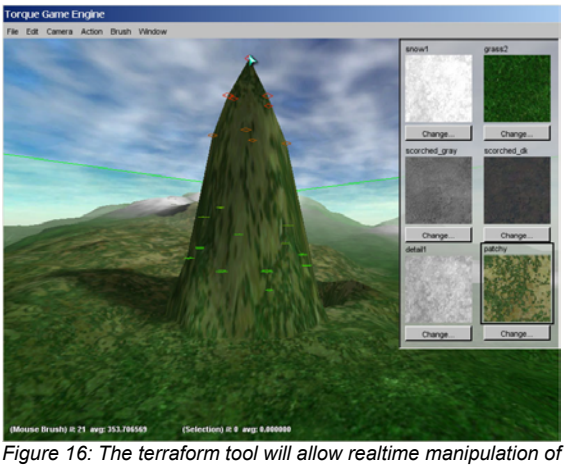


Figure 16: The terraform tool will allow realtime manipulation of the height map.

Once the desired manipulations have been done, a new lightmap must be generated before the mountain casts shadow on the terrain.

Another important tool is the terrain paint tool that will enable you to use the terrain bock as a canvas. You can load six textures, which you can either manually paint on the terrain, or use some of the many filters. The filters can apply textures depending on the height or slope of the terrain. Other random filters are also available.

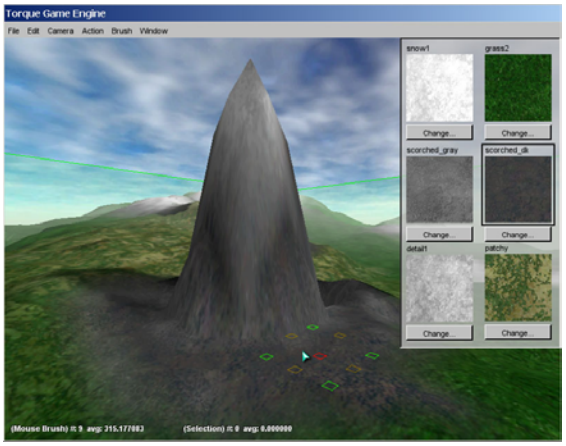


Figure 17: The terrain paint tool will allow realtime texturing of the terrain.

6 Creating a visualization

There are two types of objects supported by the Torque engine.

- DTS (Dynamic shapes and items without shadow)
- DIF (Static shapes and interior objects with light maps.)

The DTS objects can be made in 3D studio Max, or LightWave, and can then be exported to the Torque engine. Models created in any other application can be saved as VRML and imported in 3D studio Max.

The DIF objects can be created using LightWave, WorldCraft or Quark. The programs can create a model in the MAP format, which is the old Quake2 format. The MAP format can then be converted to DIF by using the "map2dif" application.

In this project I will only use the DIF format. This is mainly because all my models will need a light map, and I would like to show that this presentation can be made using cheap tools or freeware. WorldCraft is freeware.

As mentioned in the preface, I began this project a intending to use a VR system to create a VR presentation. The project crashed as the other students in the project group gave up.

In this chapter I will demonstrate how to make the feature model, and get it into the game engine.

Figure 18: This is a rendered preview of the model that was to be used in the project the scene is rendered in Maya 2001.

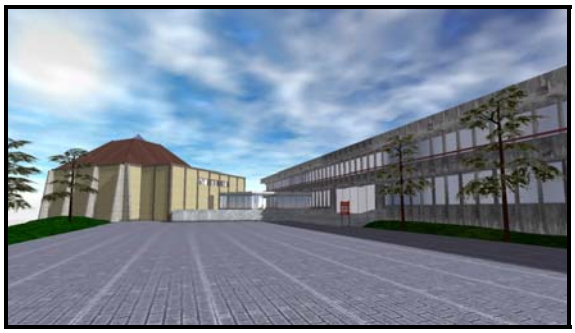


Figure 19: This is a realtime snapshot of the scene from the final presentation 2002.

6.1 Acquiring the model data

When the project was started, the building was not yet built. So we acquired the architectural blueprints from the architect office.

A very important notice here is that the constructional accuracy given on blue prints contains an accuracy that is relevant for the construction engineers, and **not** relevant for the graphical presentation. Do not waste time with details that cannot be seen with the bare eye on a monitor!

The new building was to be connected to building 101 at the campus. I could not get hold of the blueprints for this building and was forced to do my own inaccurate measurements. This was mostly done by estimation or counting footsteps along the side of the building.

One experience that I learned from this project is that one should always take a lot of photographs of the relevant site and its surroundings. I am sure this is obvious to architects. The images will serve great for model reference, but also give great inspiration and material when making the relevant textures.

6.2 *Acquiring the relevant textures*

To create virtual reality presentation that looks realistic, textures are of vital importance. Good texture work will make the model look just as good as if it was built by several orders of magnitude more polygons. Reducing the number of polygons will also increase the performance.

The existing facades were photographed and processed for texturing. The process consists of cleaning the image, cropping it and making it tileable. I would recommend doing this on a clouded day and not to use flash.

In order to make the images tileable an image manipulation program such as Photoshop is very useful. Other cheaper alternative image manipulation programs could be Gimp or Photoshop Pro. The image can then be offset so that all the borders of the image appear in the middle of the image. By using the 'stamp' tool one can then smoothly copy parts of the picture to overlap other non tileable areas. Below is and example of a before and after the tileable process.

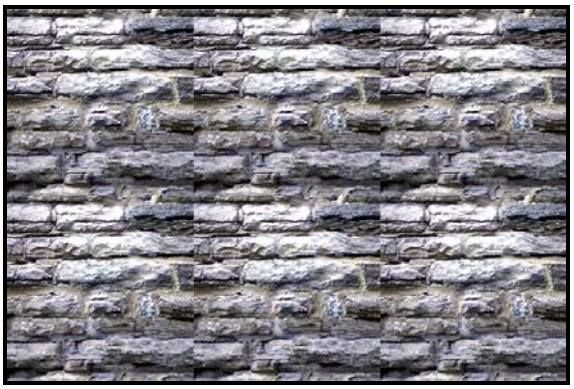


Figure 20: An unprocessed picture of a brick wall is used as a texture. The six instances of the texture becomes quite visible.

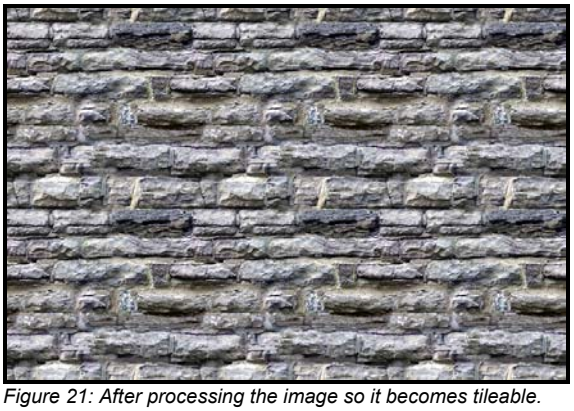


Figure 21: After processing the image so it becomes tileable.

Several tutorials on the exact method can be found on the internet. Here is one example: http://www.computerarts.co.uk/tutorials/type/tutorial.asp?id=28122

If you are using a camera to take images for textures, there are a few things you should keep in mind. First you should try to do it on a cloudy day. Direct sunlight causes a lot of secular light on the image that will look bad on a texture. Try also to get far away from the source before photographing if there is a zoom - use it. Next, it is important to position the camera along the normal of the face that you are photographing. This will make the work in Photoshop a lot easier, as the perspective distortion will be minimal.

Work with high resolution images and scale them down to the desired texture size after all the processing has been done.

To avoid repetitive patterns a high pass filter can be applied.

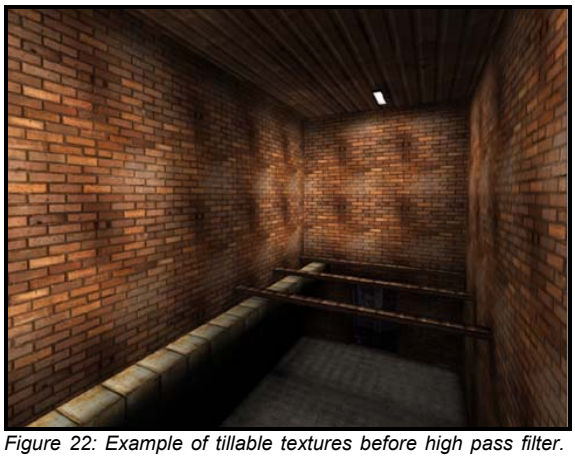


Figure 22: Example of tillable textures before high pass filter. (Picture courtesy of www.gamasutra.com)

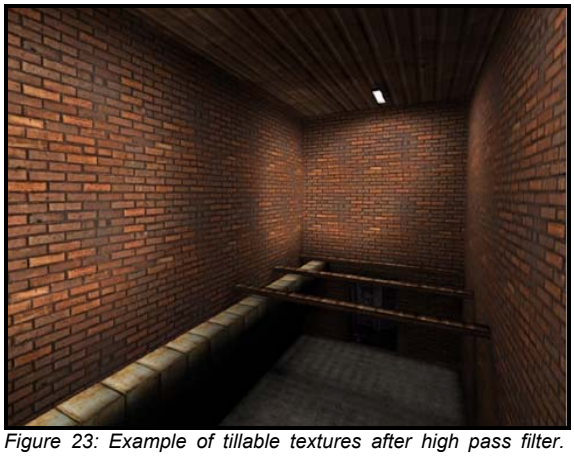


Figure 23: Example of tillable textures after high pass filter. (Picture courtesy of www.gamasutra.com)

Other textures can be found on the internet. Places like 3Dcafe.com have some for free. In general it is a good idea to keep ones textures in a size that is of factor 2. e.g. 32x32 32x64 128x256 etc. Most new graphics cards and game engines will handle 512x512 textures. Going above this is ok if you are sure that your system supports it. Using textures other that by a factor of 2 could cause lower performance.

6.3 Textures

Using photographs as texture material or making the texture from scratch?

In the original 3D model, I used a processed image of the original façade as a texture. This looked flat and fake. It was a simple way to make the building 101 with very simple geometry.

Looking over my old material as I was about to make the new model for the Torque visualization, it became clear that there had to be a better way to make the texture.

Figure 24: The original 101 façade texture

In order to make a good set of textures from this image, one has to look for tileable elements. The window is one such element. It repeats again and again. Looking at the concrete pieces of the wall, one can see that this also tiles, but that there is a variation of the dirt from tile to tile, and the tiles on top of the building are less dirty than the one in the story below.

To create a texture of the concrete tiles, I used the following concrete texture.

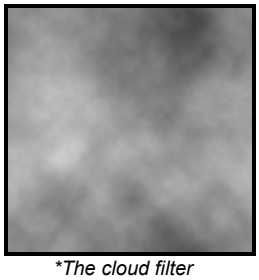
Figure 25: 256x256 Jpeg concrete texture. (High passed and tileable.)

Using this alone would give a very clean look as if the building was brand new. Obviously some dirt needs to be applied.

This is what I did to the image in Photoshop:

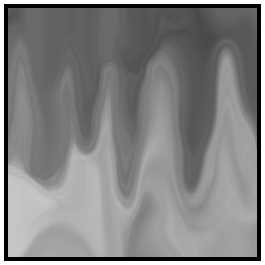
- Added and mixed another texture. (see below)
- Adjust hue/saturation until the color matched that of the lightest elements of the façade.
- Applied a new layer with a cloud filter* on. *This creates a black and white random cloud pattern.*
- Used the filter 'liquefy' on the cloud layer to create the dirt patterns.
- Applied the 'dirt' pattern as a 'hard light' to the other images.

Here is how the dirt texture was made.



**The cloud filter*

⇨



The cloud filter after the 'liquefy' process.

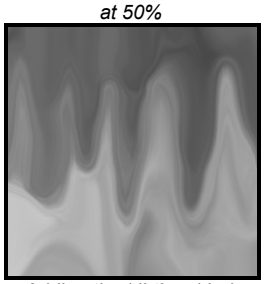
Here is how the final texture was created.

One tileable concrete texture

+

Mixed with another texture at 50%

+



Adding the 'dirt' as 'dark light'

=

Resulting Texture

By creating a few different 'dirt' textures in different strengths of black and white I created three light textures and three dark ones for the upper and lower stories.

Example of the upper(L) and lower(R) story texture

And finally a picture of the resulting façade:

Figure 26: Corner of building 101 - snapshot from realtime environment

7 Using WorldCraft 3.3

First I will need a freeware application called WorldCraft or Hammer. The program is commonly used as a level editor tool in for most of the popular game engines on the marked. It is very simple, but more than adequate for constructing buildings and other non organic models.

In appendix B all the Documentation for WC is available as well as some simple tutorials to get started.

7.1 Modeling in WorldCraft

In the original project, when the real Student Union was not yet built, making a VR presentation had a genuine purpose. As the project has been severely delayed this purpose is some what lost. I have chosen to create the original concept to demonstrate the ease in which I can do this. It is still a time consuming job, but in less time I'll get better results.

To better understand the process of making models in WorldCraft for a game engine a few useful pieces of information is worth keeping in mind during the modeling work.

- Limited texture memory. *The amount of available texture memory depends on the hardware. A good general rule is to keep the textures at a maximum of 512x512. In this report all textures are 256x256 or smaller.*
- Limited geometry capabilities. *If you have a very complex or large shape, it will have to be divided into several smaller pieces, and then assembled in the game engine.*
- Level of detail. *Depending on the amount of models in your scene, this is often the key factor to getting a good frame rate.*
- Effects of the shadow maps. *This is maybe one of the more interesting things to keep in mind. A flat surface will have the same shadow map all over. If the surface has some elements that stick out or in, then these elements will cause a shadow effect on the other surface. The shadow map makes a huge difference in visual appearance. Make the models so they take advantage of this!*

7.2 Creating Building 101

Without the real blueprints over the building, creating a VR duplicate is hard. Looking at the purpose of this VR presentation, there is no need for any high accuracy, as the building 101 is there as a reference to the new S-house. *Therefore the models herein were created based on data from images, and from memory.*

Since building 101 is such a large building, it had to be divided into several smaller pieces. By finding parts of the building that are repetitive it is possible to copy-paste elements of the building to create the full building.

For simplicity I split the building up into

1. Façade 2 story
2. Façade 3 story
3. Façade 2 story with entrance south
4. Façade 2 story with entrance north
5. Roof
6. Façade corner 3 story
7. Façade corner 2 story
8. Façade corner 1 story
9. Façade 1 story for roof.

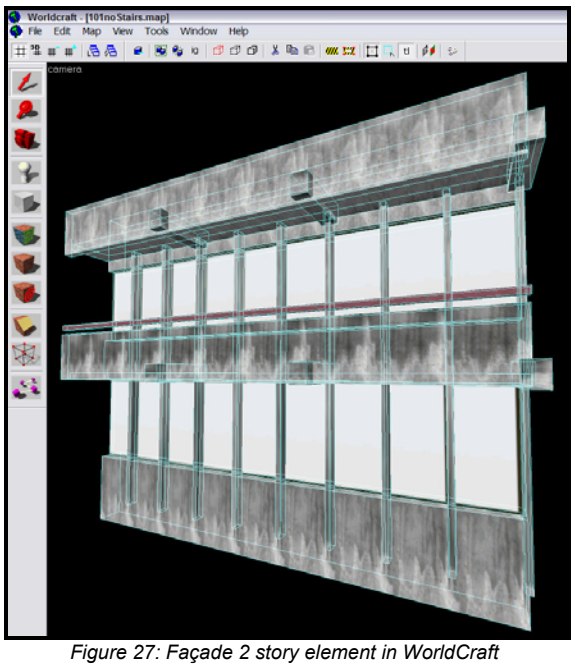


Figure 27: Façade 2 story element in WorldCraft

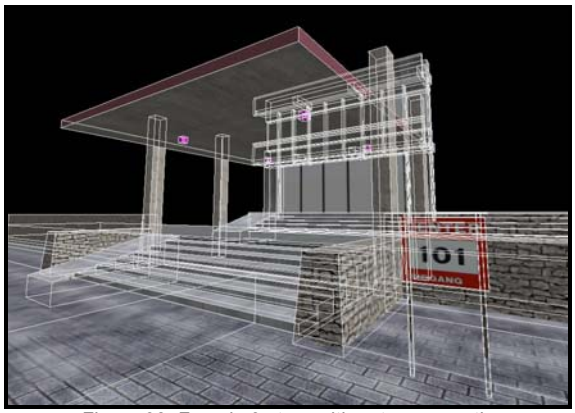


Figure 28: Façade 2 story with entrance south

Adding the parts together in Torque is easily done, but the accuracy can be a bit tricky. In the Torque editor preferences it is possible to set the accuracy of movement. This will allow you to move the next piece so that it fits exactly next to the other. By copying the coordinates of one building element and pasting them into a newly inserted element, you only need to adjust the alignment along one axis.

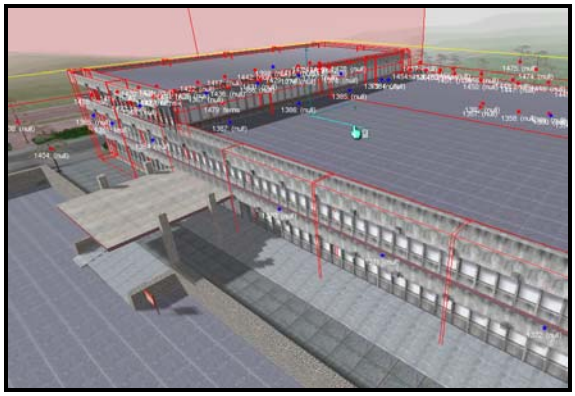


Figure 29: Front façade of 101 after each piece has been added to the scene.

7.3 Creating the new S-house.

Since the object of this presentation has been somewhat changed, I have chosen to simplify the work on this part of the visualization a bit. There is no basement in the model, and some of the interior rooms have been left out. No furniture or other interior decoration has been added.

I used WorldCraft just as when making building 101. Where I split 101 up into several pieces, I only split the new S-house into two pieces. This was to minimize the artifacts of the interior lights. One piece contains the glass corridor linking the S-house with building 101, and the other is the entire S-house.

Adding extra features - transparency
In order to get transparency on the windows I had to implement a new piece of code. Unfortunately at writing time, there is a bug in this, causing the alpha blending to do mysterious things some times. I have kept in the demo as I expect there to be a fix to this problem very soon, and it does illustrate the feature of transparent windows.

I am not going to show how this was implemented in code. The method is the same as I will show in the chapter about implementing code, where I use the example of adding the new grass elements.

Every week new features are added to the game engine by the people using it. Each added feature comes with an instruction on how to install it.

Adding extra features – environment map
While adding transparency I also enabled the environment map effect on the windows of building 101. As shown on this illustration the windows have become more window-like than in the previous illustration on page 13.

This is done by opening the script file:
/fps/data/init.cs

By adding the line:
```
addMaterialMapping("windowUp","environment:
fps/data/skies/day_0002 0.5" );
```

The texture “windowUp.png” will get the static spherical environment texture “day_0002.png” applied with the blending factor of 50%.

Figure 30: Spherical environment map has been added to the windows.

By creating light nodes in WorldCraft the inside of the S-house easily gets a ‘nice’ look from the variation in the lightmaps.

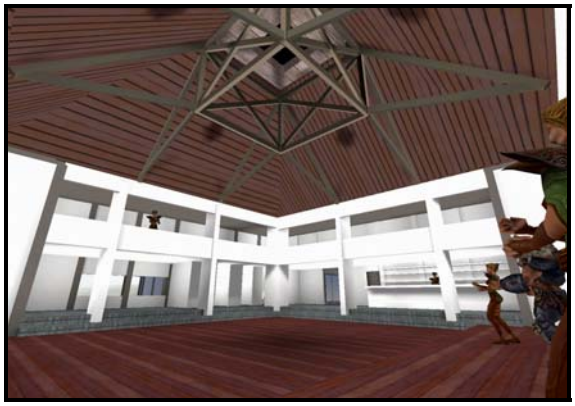


Figure 31: Interior lights. Avatars for size reference – by Realm Wars

8 Improving the Game Engine

This turned out to be a lot larger task than first imagined. One ting is to create a piece of openGL code, and another thing is to get familiar with 200 000 lines of source code and implement ones own code in it. The time it took to get familiar with Torque source code is in my opinion worth it. With this experience I can now use the game engine to present any kind of openGL code that I might make in the future.

Part of the original plan was to allow the VR user to test out various interior decorations. The students were to be able to use the VR presentation as a kind of doll house to figure out how they wanted the new Student Union to be decorated internally.

This interior decoration presentation would only require me to do some additional 3D artwork. This is something that I enjoy, but it would not be very educational for me. Instead I have chosen to demonstrate how it is possible to create new features or special graphics/effects using C++, OpenGL and the Torque Game Engine (TGE)

8.1 Game engine use for educational purposes.

Performer
Many students working with computer graphics know how to create 3D graphics using openGL and C++, but due to time limits, they hardly ever get a chance to use these skills in a project where they can put the resulting 3D graphics into an appropriate scene. The following demonstration is meant to show how this can be done.

Examples of projects where time restrains prohibit the result to be put into context:

Create a terrain optimization algorithm.
Standard result would be a terrain triangle mesh in wire frame and maybe even textured polygons.
Optimal results: Terrain implemented in VR world, multiple terrain textures, shadows, world content such as plants, trees, buildings and people. Demonstrating the generated terrain put in actual use. To show the performance of the terrain all the other elements could be toggled off.

Create soft shadows for realtime use.
The result would be a special scene where there was some limited objects demonstrating the effects of the used algorithms. The optimal result would be a full VR world where the algorithms were implemented to demonstrate not only that they work, but how this would work with a full realtime system and the impact of the improvements in visual quality.

Create a new radiosity rendering method for realtime.
Typical result would be the standard Cornell box, where the rendering algorithms were used in realtime.

Optimal result would be to get this into a full word where all lightmaps were created using this method. A cornell box should only be one room in this world.

My point here is that by using a game engine in the first place, the resulting presentation could be so much more, without putting much more than a fraction extra work into it. And each project that was completed this way would add quality to the engine that could be used either commercially or submitted as open source for a lot of people to enjoy.

Counter arguments to this would be that the game engine would impose restrictions to how the algorithms would have to be made. However since all the code would be in C++ and open GL it would be a rather easy job to take it out of the engine and make it run in a separate application.

However writing the code in a separate application first and then later integrating it with a game engine would cause double work, as the original code was not structured to work with the engine. In perspective most students that complete their education and continue to work with computer graphics will most likely be in a work environment where they have to create code that will have to work with an already existing code. Thus writing code for a game engine as an educational process will not only give the students the extra, and almost free, insight in a game engine, but it will also teach them something about writing source code that is part of a larger program. So the students would get more experience out of almost the same amount of work. The university on the other hand could encourage the students to commit the results as open source, and so the game engine would increase its quality. This would ensure that future students would find the game engine attractive and enjoy working on improving it or using it for other scientific/educational uses.

In order to create any object in the TGE engine certain basic setup code is needed. After getting the TGE SDK I was worried that I would have to

spend a lot of time getting to know the code before I could begin to produce anything useful.

I wrote a post on the TGE forum asking for assistance on this subject, and within a few days Melvyn May had made me a code snippet that would allow me to begin working right away.

The code snippet was very well documented and can be used as part of an educational exercise where the students could perform an otherwise regular openGL exercise in C++. It would surely make the exercise a lot more fun!

8.2 Getting to know the structure

I would suggest starting with some simple exercises using only openGL. Creating a rotating cube, applying vertex colors and textures would be a good start. Get the Melvin May's fxObject from the resources at GarageGames. Implement it and read the documentation carefully. This will teach you how to write openGL code that works with the Torque Game Engine (TGE)

There are a lot of other resources at Garage Games that can teach you more about things like scripting, rendering effects, pixel shaders etc.

More detailed knowledge of the structures of the game engine is also available in the TGE documentation.

8.3 Using online resources

There is always the GarageGames forum, where most questions are answered quickly. If you have a chat client like mIRC you can always login to the GarageGames chat and talk to the others in the community.

9 Creating fxGrassReplicator

I have spent a lot of time modeling plants and foliage in 3D modeling applications. The work is tedious and implementing the resulting objects in a VR world is a big job that very often gives a repetitive results.

Creating billboards is a well known method in VR for generation of forests and foliage in nature scenes. For the demonstration purpose in this project I will try to improve the visual quality of the grass.

9.1 Previous work on visualizing grass

One of the latest action games uses the billboard impostors to create grass. As mentioned it looks good from afar and terrible when the billboards rotate around your feet as you walk through the grass. The result is even worse as they tend to use mostly one texture for grass in any given scene.

Figure 32: Soldier of Fortune by Double Helix 2002. Note how the grass all look the same. It becomes very obvious when seen in realtime.

Another approach used by Acherons Call 2 a new massive multiplayer online roleplaying game (MMORPG) is to use quads with a texture and an alpha map, but not to allow them to rotate to face the camera.

The game only uses grass in enclosed areas or in patches if placed in the open. I presume the grass is quite demanding on the hardware. I do not know if the use any kind of level of detail.

Figure 33: Asherons Call 2 by Microsoft Game Studios. The grass elements consists of one textured quad that is placed at a 90 degree angle on the terrain. The grass does not rotate to face the camera.

The method used in Asherons Call looks decent from a low perspective. However once you move the camera up a little higher and look down, the quads with the grass texture become almost invisible. This gives the illusion that the grass directly under you suddenly disappears. (See Figure 36)

9.2 My approach

When you move around in a world full of billboards the illusion of being in a real world will quickly break as you see all the plants rotate around their Z-axis to face you/ the camera. In order to avoid this it is imperative that the billboards are far away from the camera, and replaced with something else when viewed up close.

The distance check is not very time consuming, but if you were to create billboards with grass on, the amount of elements to apply a distance check would be substantial. It is probably in the order of 200 000 or more. Some optimization could be done, but what kind of replacement for the billboards would be used up front?

After looking at various methods for drawing grass in some of the newer games on the marked, I decided to give it a try.

I first got Melvin May's code for the fxFoliageReplicator. The code enabled the placement of several alpha textured billboards and was created to allow the placement of trees and bushes.

Used for grass, the result looked reasonable from a low perspective, but moving the camera up and

looking down the polygon faces were very obvious. (See figure 33-34)

Figure 34: Typical 'billboarded' grass. All the grass elements face the camera. This works ok for still images, but once you begin to move around it becomes obvious that the grass elements are rotating.

Figure 35: Looking at the same scene from above. The grass elements are all aligned.

Tilting the polygons a little (20-30 degrees) made it look a lot better from above, but now the grass polygons became rather obvious when walking in the grass. (See illustration below)

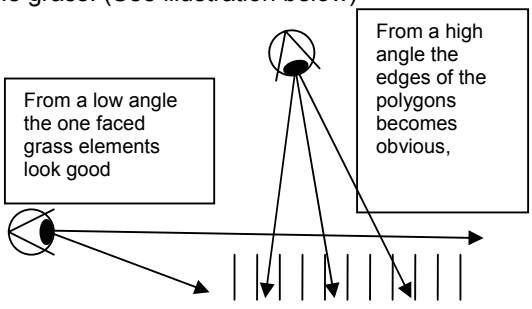


Figure 36: The grass disappearing from a high perspective.

Since it is not mandatory that the grass elements only consist of one polygon, I tested grass elements consist of two quads placed in a V-shape.

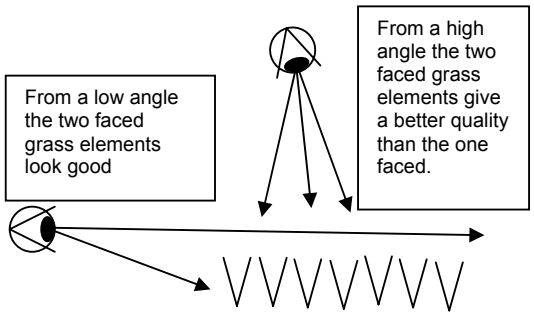


Figure 37: Here the grass is still visible at a high perspective.

The two faced grass elements looked fine from the lower angles. Tilting the camera down, one of the two faces could get aligned with the direction of view, but the sister polygon would always be in an angle, thus keeping up the appearance of grass.

Figure 38: Double quads in a v-shape textured with the same texture, added vertex coloring and removed the billboard effect of aligning the faces to the camera.

Now the grass began to look reasonable from both a high perspective and a low perspective.

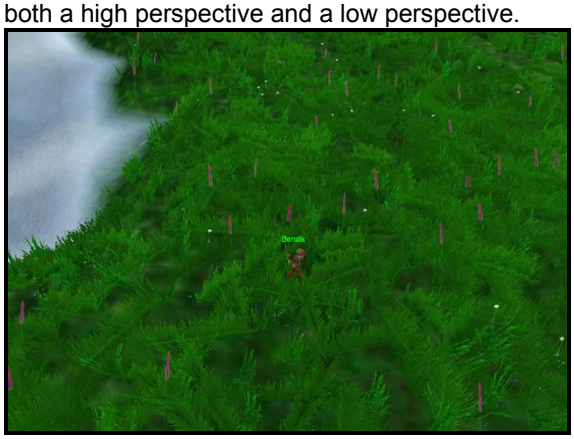


Figure 39: The high angle visual quality is not as good as the lower angle, but I found this result a lot better than the original method.

9.2.1 Even grass distribution

Circular placement area
Once I had made the code for the actual grass elements, I began looking into a way to distribute the grass evenly over a given area.

In the original code the grass was distributed by random in a given circular area. Position is given by polar coordinates R and ω where both are generated as random numbers.

Distributing the grass elements by using polar coordinates where the position is given by (R,ω) result in a higher density in the center of the circle and a lower density at the edge of the circle when R is a random number $\in$[0,OuterRadius] (see Figure 37).

More even distribution.
By dividing the circle into several circle-rings, like a dart board, the distribution could be applied more evenly. The total density of the circle was calculated, and then each ring was assigned the appropriate amount of elements. By dividing the circle into ten rings, the distribution was better but not good enough(see Figure 38). Finally at 100 subdivisions the distribution was good (Fig. 39).

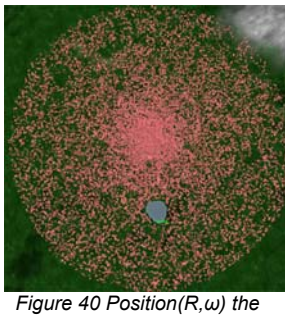


Figure 40 Position(R,ω) the original distribution algorithm.

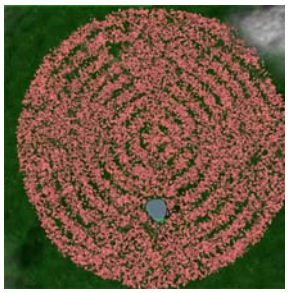


Figure 41 Position(R,ω) where the circle is subdivided in 10 rings and distribution by density.

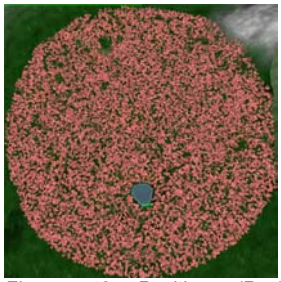


Figure 42 Position (R,ω) where the subdivision is set to 100

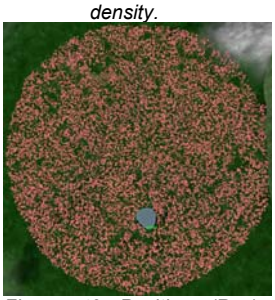


Figure 43 Position (R,ω) where R is the square root of a random number between 0 and 1.

Figure 39 and 40 have a slight difference in the hue. This is due to a difference in the alpha blending which is irrelevant in the distribution evaluation.

However I soon became aware of a much more elegant way to distribute the elements over the circular area. Instead of subdividing the circle into 100 rings a similar result could be obtained by merely using the square root of a number between 0 and 1 and then multiplying this with the desired radius in the circle.

To sum up the methods:
In all four examples ω is a random number between 0 and 2π, and where r is a random number between 0 and the desired radius and R is the outer radius of the circle.

Figure 37:
Position X = r * R * cos(ω)
Position Y = r * R * sin(ω)

Figure 38:
The total density of the total area πR^2 is calculated. The circle πR^2 is then divided into 10 rings. The elements are divided into the 10 rings based on the density and the area of the ring, and then distributed into each ring according to:
Position X = r * R * cos(ω)
Position Y = r * R * sin(ω)

Figure 39: Exactly the same as in Figure 38 only distributed over 100 rings instead of 10.

Figure 40:
Position X = ($\sqrt{}$r) * R * cos(ω)
Position Y = ($\sqrt{}$r) * R * sin(ω)

The visual result of the algorithms used in both figure 39 and 40 looks identical. But since the algorithm used in figure 40 is both quicker and more elegant I have removed the other.

Squared placement area

Making squared placement areas required the given area to be sub-divided into smaller cells.

To allow for a high level of customization two variables from the editor were reused as they were only useful in case of a circular placement area.

For the square placement area the two variables were reused to define the number of rows in both x axis and y axis.

Since the placement algorithm uses a loop to go through all the elements, a modulus for each row was used. See sample code below.

Now it is only to go through each subdivision and place a grass element until all elements have been placed. If all the grass elements were oriented the same way, the result would look much like a wheat field with rows.
(See figure 44-45)

```
if (!mFieldData.mIsSquare){
        // Calculate a random offset for the CIRCULAR distribution system.

        float radius = RandomGen.randF();
        Angle   = RandomGen.randF(0, M_2PI);

        // Calcualte the new position.
        GrassPosition.x += mFieldData.mOuterRadiusY * sqrt(radius) * mCos(Angle);
        GrassPosition.y += mFieldData.mOuterRadiusX * sqrt(radius) * mSin(Angle);
        }
        else{
        // Calculate a random offset for the SQUARE distribution system.
        float dX, dY, mX, mY;
        int n, N, mXn, mYn;

        //n = mCurrentGrassCount + (101 - RelocationRetry); // current iteration
        n = idx; // Iterator
        mX = (float)mFieldData.mOuterRadiusX; // lenght of X
        mY = (float)mFieldData.mOuterRadiusY; //length of Y
        mXn = mFieldData.mInnerRadiusX;  // number of divisions along X
        mYn = mFieldData.mInnerRadiusY;  // number of divisions along Y
        dX = (float)mX/(float)mXn; // distance between the X divisions
        dY = (float)mY/(float)mYn; // distance between the Y divisions

        if (mFieldData.mIsRandom)
            {
            GrassPosition.x += RandomGen.randF(-mX, mX);
            GrassPosition.y += RandomGen.randF(-mY, mY);
            }
        else
            {
            // The position of the area center plus…(-1.0f * mX/2) is the offset
            // to allow the center of the placement area to be the center of the group object.
            // (1 + n%mXn) will go from 1 to the number of rows in the X direction
            // (1 + n/mXn)%nYm will go from 1 to the number of rows in the Y direction
            GrassPosition.x += (-1.0f * mX/2) + ((float)(1 + n%mXn )) * dX;
            GrassPosition.y += (-1.0f * mY/2) + ((float)(1 + (n/mXn)%mYn)) * dY;
            }
        }
        if (mFieldData.mIsRandom)
            {
            GrassDirection.x = GrassPosition.x + mFieldData.mMaxWidth * mCos(Angle2);
            GrassDirection.y = GrassPosition.y + mFieldData.mMaxWidth * mSin(Angle2);
            }
        else
            {
            GrassDirection.x = GrassPosition.x+mFieldData.mMaxWidth*mCos(mFieldData.mRotateAngle);
            GrassDirection.y = GrassPosition.y+mFieldData.mMaxWidth*mSin(mFieldData.mRotateAngle);
            }
}
```

In order to make the grass placement more random two more things were done.

First the orientation was set to a random direction. Secondly instead of placing the grass element in the exact grid point, the grass element would be placed randomly within a radius of the given grid point. (See figure 46)

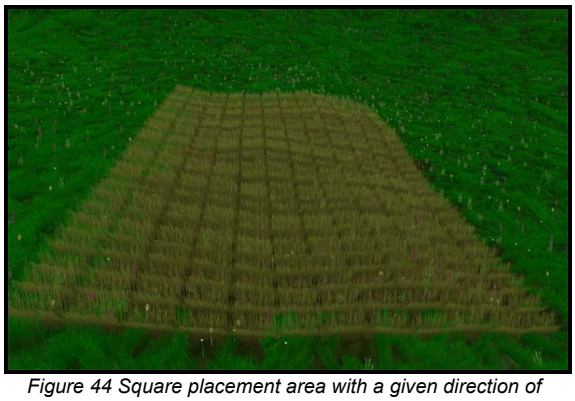


Figure 44 Square placement area with a given direction of each element.

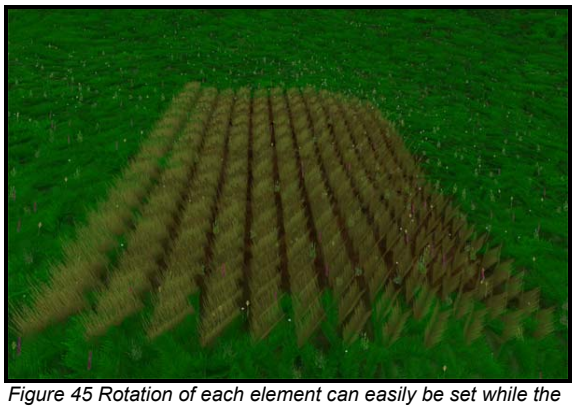


Figure 45 Rotation of each element can easily be set while the presentation is running..

Figure 46 Square placement area with a random direction of each element.

9.2.2 Alpha blending problems

Due to the amount of grass elements back to front sorting is not a viable solution. If a sorting algorithm was used all the elements would have to be resorted every frame. With e.g. 200 000 grass elements this would quickly lower the frame rate.

With out any sorting the best solution is normally the (GL_ALPHA_TEST) where the alpha value is a binary value given by the texture.

This would result in elements looking like this:

Figure 47: No sorting and GL_ALPHA_TEST enabled.

The upside to this method is that the blending works nice with the avatars running through. The downside is that the edge between the ground and the textured polygon becomes quite sharp.

The alternative where the edge between the ground and the polygon becomes less obvious is when the two lower vertices in the polygon becomes partly transparent. But in order to get this feature to work properly, GL_ALPHA_TEST must be disabled and the result looks like this.

Figure 48: No sorting and GL_ALPHA_TEST disabled

9.2.3 Vertex coloring for flexible effects.

Vertex coloring is a cheap and easy way to get more variation in the output graphics.

By enabling vertex coloring a grayscale texture can be used instead of a colored texture. This gives a nice flexibility to the grass elements.
By manipulating the vertex colors, the grass can change color from light green in the spring to dark gray in the winter.

So instead of making several textures for the various seasons' one texture can be used.
If there are multiple colors on the texture vertex coloring might not be a good option, so objects like flowers would not use this feature.

By assigning different colors to the top and bottom of the polygon, a nice effect is created. Normally a darker color would be appropriate at the bottom and a lighter one on the top.
Here is an exaggerated example.

Figure 49: Vertex color example. different colors at the top and bottom.

9.2.4 Culling algorithm used

The culling mechanism was done by Melvin May. It encapsulates the entire area of the grass with a quad tree. The quad tree is made up of boxes that can be checked for culling. The smallest set of boxes will have a set of grass elements attached.

More information on this is given in the source code.

9.2.5 Animation of vertices and lights

This is a very simple animation where the two top vertices of each grass element will follow a sinus function. A time element is included to decide the speed of the animation and a variable is set to control the magnitude of the animation.

Upon drawing the polygon a light value is multiplied with the vertex colors, giving the illusion of variances of shade over time. This effect tries to copy the effects of the variance of light due to the movement of the grass in the wind.

9.2.6 Collision tests

The original code created the billboards by doing one collision test with the terrain, and then drawing the billboard in that point.

With the wide grass elements this method gave some ugly artifacts. In curved terrain the grass polygons would stick out, and the illusion of grass was quickly lost.

The first attempt to fix this was done by rotating the grass around the terrain normal. This worked fine in some instances where the grass was placed horizontal in a hill, but was not aligned with the curvature of the hill.

The solution was only partly successful, as the problem remained where there was a change in the terrain curvature. The solution also removed the random element in the rotation of the polygons. Now they would all follow the height curves of the terrain.

One viable solution was to make one collision test at each corner of the polygon. This way the random rotation of the polygons was preserved. There would still be problems where the curvature of the terrain changed greatly.

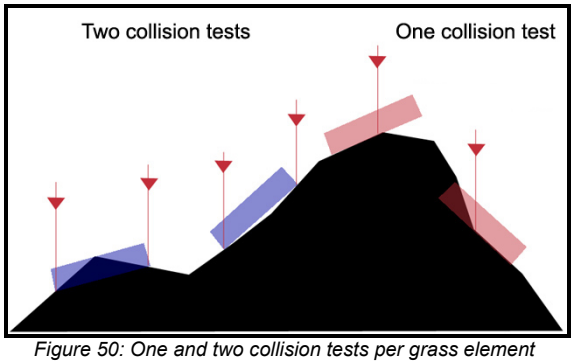


Figure 50: One and two collision tests per grass element

The remaining problems were dealt with by only allowing the grass to grow at a given maximum angle of curvature.

The chosen solution was not perfect. It gave a lot better visual quality, but as seen on fig. 39 it eliminated polygons sticking out into the open.

By testing what kind of object the collision is done with, it is possible to allow the grass to only grow on terrain, water, interior objects or static objects, or any combination of these.

Uses for this could be to make crass covered roofs or flower beds and water plants.

9.2.7 LOD and popping

To increase performance I implemented a LOD system. The system could remove every other, second or third grass element, as the distance increased. This would have worked nicely if it was possible to blend the alpha value of each grass element in and out.

As stated in 8.3.2 the alpha check was done by GL_ALPHA_TEST, and results in a binary value of the alpha of the grass. So either the grass is visible or it is not.

Without this feature it was impossible to avoid severe popping, after several tests the LOD was removed from the system until I came up with the idea of letting the grass element polygons grow up from the ground instead of fade in.

Two variables define the distance from the camera to where the grass elements are visible. This test is done whether LOD is implemented or not. By using grow up/down instead of alpha blend in/out in combination with max/min viewing distance; LOD could now effectively be integrated.

Instead of using a LOD system within the fxGrassReplicator object, e.g. 4 instances of fxGrassReplicator could be used to create 4 levels of detail. By leaving the level of detail out of the

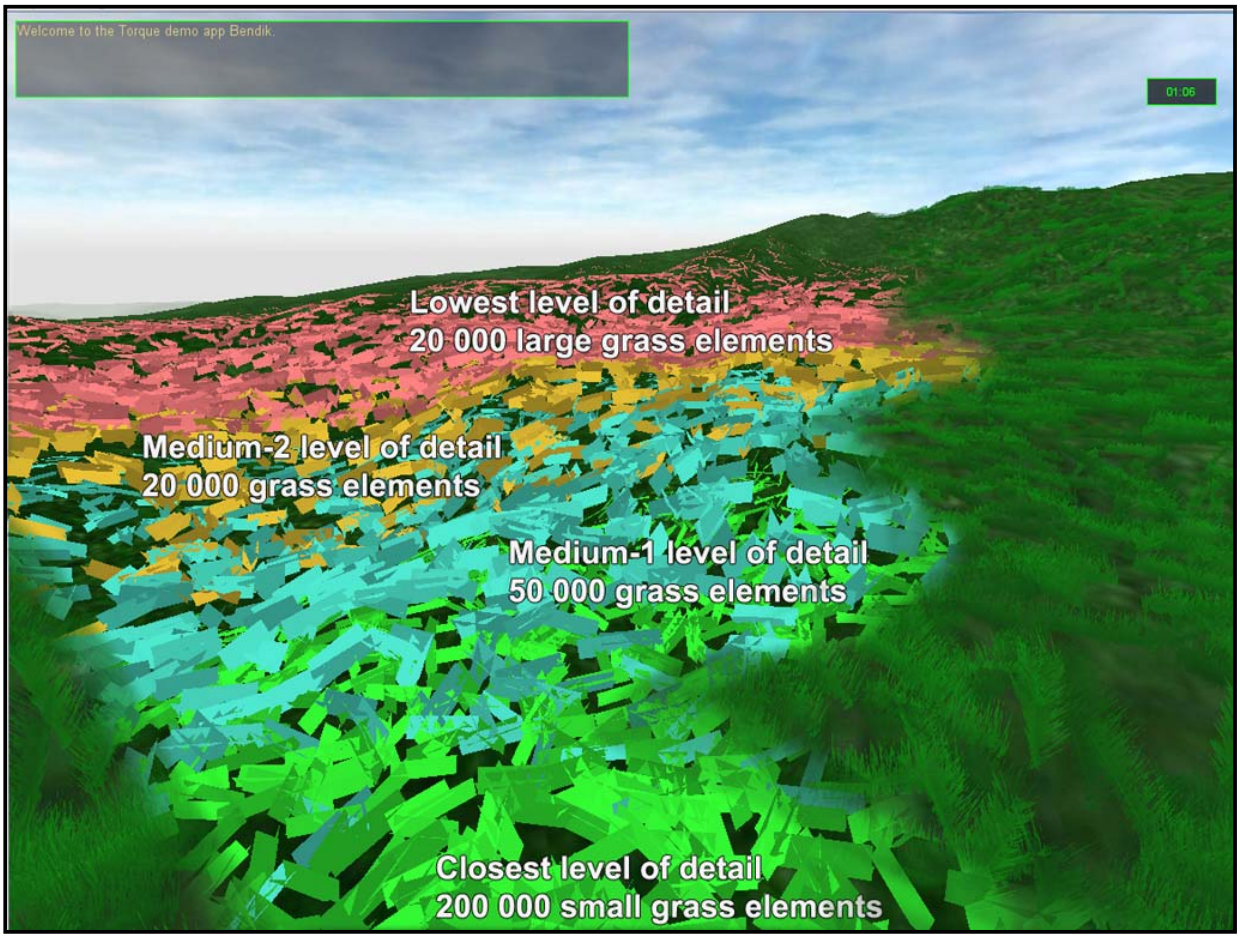


Figure 51: Level of detail visualized.

actual code, but enabling the use of each instance as one set of detail, great optimization was possible.

Figure 51 is a blend between two screen shots. It shows the grass displayed as grass and the colored elements shows how the level of detail works in this scene. The lowest level of detail (pink) has twice as large grass elements as the closest level of detail.

9.2.9 Conclusion

Using camera aligned impostors optimizes the display of each billboard element. However it does not look well when the camera is moving through a scene since the rotations of the billboards are very obvious.

With the amount of grass needed to create large green fields it puts a heavy strain on the computer hardware. With my hardware from late 2001 it runs at around 10-15 frames per second.

Comparing the results with what I have seen elsewhere on the marked today, I'm surprised that the visual quality of my grass is so high in comparison. I'm most pleased with the result, and believe that with a little further adjustment of the level of detail as well as a little better computer hardware, my code could be used in virtual reality presentations and games when grassy fields are needed.

The grass is not perfect like I dreamed of but, while testing various ways of making grass I came to understand that you must accept some kind of compromise due to hardware restrictions. It is optimal if one has a limited control over the camera. This way the grass can be tweaked to either look perfect from a low or a high angle.

With this code I believe I have improved the game engine a bit.

9 Conclusion

Game engines vs. Virtual Reality engines.
I started out looking for a good way to make a virtual reality presentation. In my opinion I have found one.

Comparing a game engine to a vr-engine I found little conceptual differences. The game engines seem to have more visual effects and an easier workflow than the vr-engines. The game engines are also generally speaking cheaper.

Since the marked of Game Engines change so rapidly, today's best engine might not be tomorrow's best engine. The Torque engine might not be the best engine for all uses, but it has a lot of people working on improvements. This indicates a chance that Torque will be up-to-date for some time to come. With the very low cost of use, I still feel it was a great choice for this project.

Further more I believe that the days of Virtual Reality tools are over. The quality and usability of game engines exceeds that of the best Virtual Reality tools. During the time I have spent on this project new and even better game engines have appeared on the marked. But, the Virtual Reality marked has not died, and so the Game Engines will take over for the Virtual Reality engines.

With game optimized multiplayer code it is now possible to create a presentation, and host a server online. Many people can then log on, and experience not only the e.g. architectural presentation of a new building, but also the interaction of lots of people within this world.

As the game engines and their tools mature, it is becoming easier and easier to make presentations. Still, making the models and the textures for the presentation requires a fair amount of work, and I cannot see how this could be done a lot easier. Better modeling tools can only help the work process, but in the end the model must be built.
If the architects that design the building do so in a format that can be converted to one that goes right into the game engine, lots of work can be saved.

Looking at the models I made myself in this project, the work process, and the implementation the result far surpasses that of the original project that was abandoned. Of course I have gained valuable skills in between, and so a direct comparison is not completely fair.

Comparing the work process and the expenses of the equipment and software, I see little doubt. Using a Game Engine is both quicker, better and far cheaper.

Implementing new code to a Game engine
The second part of this project was about implementing new code to improve the chosen game engine. I found this part of the project very educating and fun.

I successfully created fxGrassReplicator to allow lifelike swaying grass to be easily placed over a desired area.

The code improved the existing method by removing the rotation of the billboards to face the camera. This was a vital improvement in an environment where the camera is moving close to the ground. By using two collision checks to place the billboards it was possible to place the grass in an uneven terrain without any visual. And using two billboards instead of one allowed for variances in the camera pitch and still keeping the visual quality reasonable.

Obtaining the relevant information needed to complete this task was mostly easy. At the Garage Games forum I often found quicker service than that of a teacher at the University, as people would answer posts around the clock.

Writing a 3D computer graphics program can be a huge task. Getting a good looking result is often a tedious and time consuming process. Doing all this within one university project is normally not possible due to time restrictions. Compared to what is on the marked today I think the results of the grass code that I have implemented far surpasses what I have seen anywhere else. Unfortunately it is still demanding a lot of CPU time and will have to be tweaked and optimized further or one can wait half a year and get a better computer.

10 Reference

Sherman Chin Lit Kong – *3D Binary Space Partitioning (Game Engine Focus)* - University Of Portsmouth.

Microsoft OpenGL Reference pages.
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/opengl/glfunc01_9u3y.asp

fxFoliageReplicator by Melvyn May
http://www.garagegames.com/index.php?sec=mg&mod=resource&page=view&qid=3057

fxRenderObject by Melvyn May
http://www.garagegames.com/index.php?sec=mg&mod=resource&page=view&qid=3217

3D Game engine list
http://cg.cs.tu-berlin.de/~ki/engines.html

SIGGRAPH - Commercial Game Engines
http://www.siggraph.org/cgi-bin/cgi/idCatResults.html&CategoryID=23
and
http://www.siggraph.org/cgi-bin/cgi/idECatResults.html&CategoryID=23

Engines And Engineering
What to expect in the future of PC games.
By Steven L. Kent | Oct. 31, 2002
http://www.gamespy.com/futureofgaming/engines/

International Game Developers Association
Game Engines listing
http://www.reanimation-studios.com/igda/gd_engine.shtml

Gobal Illumination Compendium by Philip Dutré.
http://www.cs.kuleuven.ac.be/~phil/GI/TotalCompendium.pdf