Computer Science 780 Project Report

Game Engines for Interactive Collaborative Biomedical Visualizations

Andrew Gits

Supervised by:

Burkhard Wuensche

Semester One 2005
Department of Computer Science
University of Auckland

Table of Contents

1 Abstract

The overall aim of this project was to investigate the use of existing computer game technology for collaborative biomedical purposes. This involved implementing a simple visualization tool into a 3D game engine. Various game engines were researched, and reviewed based on their features and flaws. Three critical factors were taken into consideration when reviewing the engines – rendering ability, network capability, and ease of use. Commercial engines were preferred over open-source engines, as they are widely obtainable, and often have superior compatibility and stability than the open-source alternatives.

The tool development involved three main steps – loading the dataset into the game engine, visualizing the dataset, and extending the multiplayer features to utilize them for collaborative visualization. The engine chosen was Quake 3, as this satisfied the criteria as best possible, and there was a huge amount of resources existing for Quake 3 modifications, making the task more achievable in the given time-frame. The resultant tool, while slightly lacking in functionality, demonstrates the ability to use the Quake 3 engine for visualizing a dataset. Additionally, it allows several users to connect to a server, and visualize the same information, provided they have the dataset on their computer.

2 Introduction

As biomedical technology advances, so does the size and complexity of the data sets. As a result, it becomes much harder to utilize the information obtained. Visualization of these data sets is fundamental for exploring, analyzing, and understanding the biomedical data, so it is essential to find (or develop) a tool that will allow this.

A variety of commercial visualization applications exist, however most of them are expensive and difficult to operate. Furthermore, they are usually not designed for collaboration and interactivity in the exploration of data sets. Another drawback is that all popular visualization tools use different data formats. This makes it difficult for researchers to exchange data sets and results unless they all use the same software.

This project examines the use of game engines as a visualization tool for biomedical data sets. A huge amount of research goes into the design of game engines, and they often contain the most efficient, well tested implementations of algorithms from all areas of Computer Science. Additionally, game engines are designed to allow multi-player games over a network, and are typically easy and intuitive to use.

For these reasons, this project investigates the ability to adapt game engines for biomedical visualization. Re-use of existing code and functions would mean minimal work would be required to complete the modifications needed. This would result in a free, easy-to-use tool which allows researchers world-wide to collaborate and explore biomedical data sets.

3 Background

3.1 Computer Game Graphics

Computer game engines can be categorized as either 2D or 3D. While many 2D engines do in fact utilize 3D APIs to draw content, they are of little use in a medical visualization since they are designed for 2D graphics only. There are two main APIs created for 3D graphics – OpenGL and DirectX [4, 5]. Most games support one of these, and some engines occasionally support both.

OpenGL is a cross-language cross-platform API developed by SGI. The interface consists of numerous functions which are used to draw 2D and 3D content. OpenGL focuses solely on Computer Graphics, and the implementations available on various platforms are very efficient, making it very popular with the Computer Game Industry.

DirectX is a collection of APIs developed by Microsoft, and includes support for sound, networking, graphics, and video. At the current time, DirectX is only available on Windows and, although quite different, Xbox. Direct3D deals with the 3D component of DirectX, and is roughly equal with OpenGL in terms of speed and functionality.

One other aspect that should be mentioned is hardware. The majority of modern video cards support hardware acceleration of both OpenGL and Direct3D. As such, almost all current generation 3D game engines require one of these video cards to run at an acceptable frame rate. The video card used for this project has an nVidia GeForce 4 Ti [6] chipset, which performs sufficiently well even considering it is several years old.

An issue that might arise is that game developers cut corners in order to keep rendering fast, so as to retain a high frame-rate for real-time interaction [2]. While these shortcuts may make the game more enjoyable, this can not be replicated for medical visualizations. Biomedical data provided to users to visualize and explore must be as accurate as possible, otherwise it becomes almost useless.

3.2 Computer Games Architecture

A popular trend when designing computer game engines is to separate the game into three modular, parts as shown in the diagram below.

The first part (the .exe) contains the low level "engine" code. This is where tasks such as rendering, physics and memory management are performed. It handles all the functions utilizing hardware, such as sound cards and I/O devices. The engine code also links all the APIs, such as OpenGL, into the game.

The second part is higher level, and abstracts away from hardware specific code. The main purpose of this part is to manage the game logic. This "game code" defines how the game itself plays. The implementation can vary – sometimes a scripting language such as Lua [7] is used, and other times the game code is written as a library and linked into the engine.

The final part is the game content, which contains all of the media the game uses. It may contain anything from images, to sounds, to 3D data. Many developers incorporate their own proprietary formats for data, in particular 3D data, which has been optimized to for use with the engine.

The main benefit of utilizing this type of game architecture is flexibility. It enables the developer to open-source the game code, letting users create modifications – anything from a small weapon addition, to an entire game – without the need to access the engine source code.

3.3 Existing Computer Game Engines

A variety of existing game engines were looked at, ranging from free open source engines, to commercial engines used in actual off-the-shelf games. There were several important factors that were taken into consideration when choosing an engine. The most significant, as expected from a 3D visualization, is the rendering power. A fast efficient render system was desired, without needing ridiculous hardware specifications. Another crucial factor is which 3D data format the engine uses – different formats vary in difficulty to implement. The final main feature required is network support, enabling 2 or more users to connect to a central server. This is vital for any sort of collaboration implementation.

The decision between using an older engine or a newer one was difficult. While older engines generally have been patched to fix bugs and are a lot more stable, newer engines have the advantage of utilizing the most recent graphics hardware, and thus have much more rendering power.

3.3.1 Open Source Engines

The number of Open Source engines available at present is ever-changing, with a constant flow of new engines replacing older engines. However, a few have stood the test of time. Several of these engines are older commercial engines, open sourced simply because of their age. The remaining are amateur engines, being created by a community of programmers, and are continually being developed with improvements being made every day.

OGRE [8]

OGRE (Object-Oriented Graphics Rendering Engine) is an established amateur 3D engine, supporting both OpenGL and Direct3D on a number of platforms. It also supports a number of 3D data formats, and rendering modes. It is quite advanced, however it is only a graphics engine, and lacks any kind of networking, or other non-graphic features. Thus, it will not be suitable for this project.

Irrlicht [9]

Irrlicht is another amateur engine which supports both OpenGL and Direct3D. It also features its own software engine, which is an advantage for computers lacking adequate 3D specific graphics hardware. It is written in C++ although it is also available for .NET languages. However the lack of network functionality means this is not suitable.

The Nebula Device [10]

The Nebula Device is essentially a C++ based rendering engine, and features 'shader' functionality – programs which run on the video card chip. It is scriptable, but only currently supports DirectX 9. Like the previous open-source engines, it lacks networking support, so is not suitable.

Quake 2 [13]

Quake 2 is an older commercial game and engine, produced by id software. It is written in C, not C++, however it supports OpenGL, and has a reasonably sized amateur community continually developing it. It uses a proprietary MD2 format for 3D data, and there are numerous constraints on the format's graphical complexity. It does have network support however, as well as GUI functionality.

3.3.2 Commercial Engines

Whilst Commercial Engines can cost an incredible amount to license [3], in some cases they can be used for free. This is achieved by modifying the "game code" portion of the engine, which is often open-sourced to encourage amateurs to create their own add-ons to the game.

Cipher [11]

Cipher is a 3D game engine based mainly on the Quake 3 Arena architecture. It supports OpenGL, and includes many of the features found in complete game engines. It has networking support, as well as a 2D GUI system, which make it an attractive engine. While Cipher isn't actually being used in any existing commercial games, it still costs $100 to license, and unfortunately there is no way to access the game code without licensing the engine.

Quake 3 [13]

Quake 3 is one of the most mainstream game engines in the game industry at present. It is OpenGL based, and features network support, efficient hardware rendering, and an extremely easy-to-use GUI system. The engine has been around for several years now, and is extremely stable, and many resources exist for modifying and extending the engine.

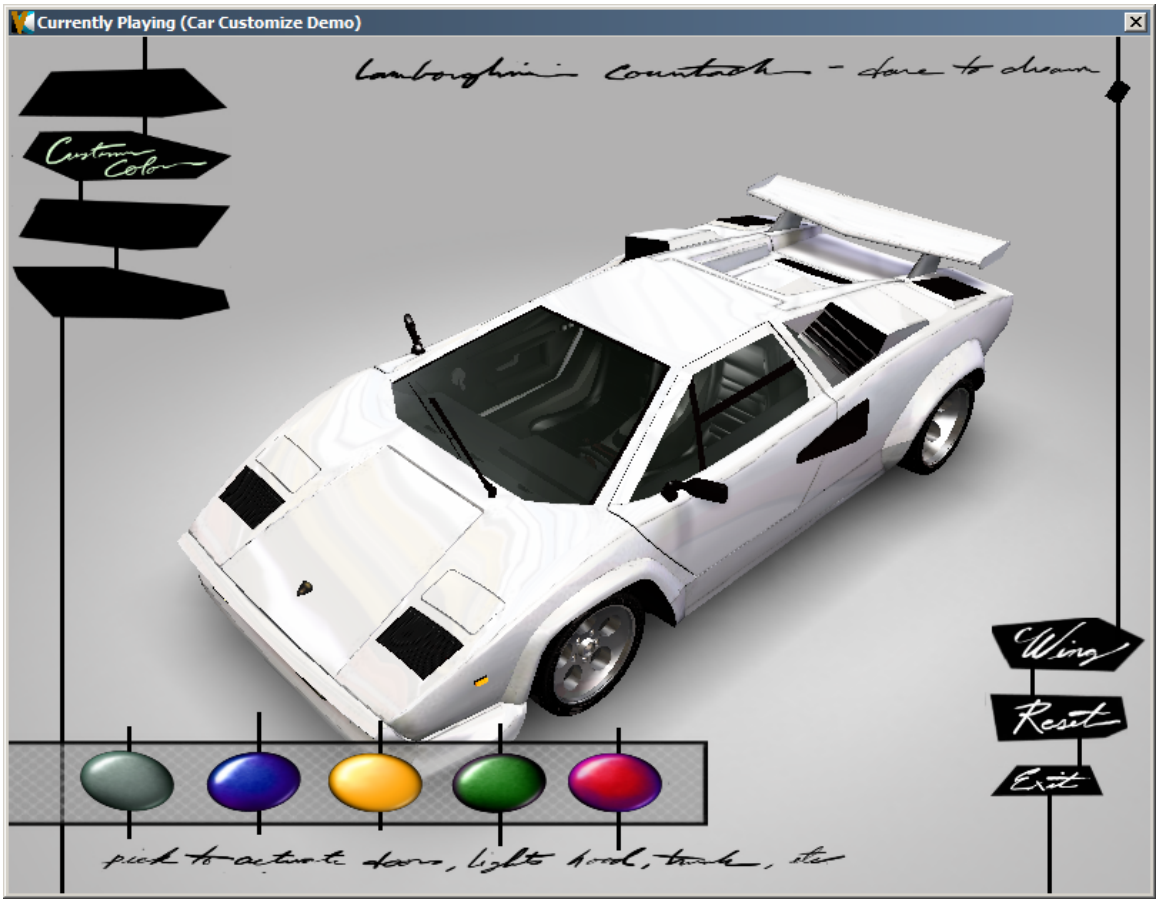
Unreal Engine 2 [12]

Unreal Engine 2 is another complete solution engine. It features OpenGL and Direct3D support, plus thoroughly tested networking capabilities, and ease of use. However, it has its own proprietary scripting language, known as Unreal-Script, which makes the process of modifying the engine slightly more complex.

Doom 3 [13]

The Doom 3 engine is one of the most advanced game engines available on the market. It allows rendering extremely complex 3D data, and has a fully complete lighting and shadowing system, at the expense of having massive hardware requirements. However, the other features are similar to that of the other engines, and there is no special advantage in using the Doom 3 engine.

All the above commercial engines are quite similar in feature-set, the only differences being the licensing involved in using/modifying the engine. Another engine that should be considered is Deep Creator [14]. While this isn't truly a game engine, it is being marketed as a "*powerful authoring application to create interactive 3D environments, assemblies and objects*". It doesn't have the networking features present in these commercial game engines, but there is a lot more room for customization and the uniqueness of Deep Creator is in its ability to create specific interfaces which allow the user to visualize and interact with the content. The application is versatile enough that almost any type of application can be created from it [1], and it has detailed documentation on every aspect of usage.



The figure above shows one sample of a Deep Creator application – in this example, the user can customize a car with different colours, and rotate the car in full 3D to see it from different points of view.

3.4 Existing game-based visualizations

There have been several attempts at using game engines for non-game purposes. Some of the more successful projects have been investigated below.

Unrealty: Application of a 3D Game Engine to Enhance the Design, Visualization and Presentation of Commercial Real Estate [15]

This paper discusses the various benefits and issues around the idea of using a game engine as a tool for visualizing Real Estate. It discusses features that exist in the Unreal game engine which can be utilized to create realistic representations of houses (and other types of building). These representations can be used by Real Estate agents to allow Clients to visually explore property without the need to actually visit the site.

UnrealTriage: A Game-Based Simulation for Emergency Response [16]

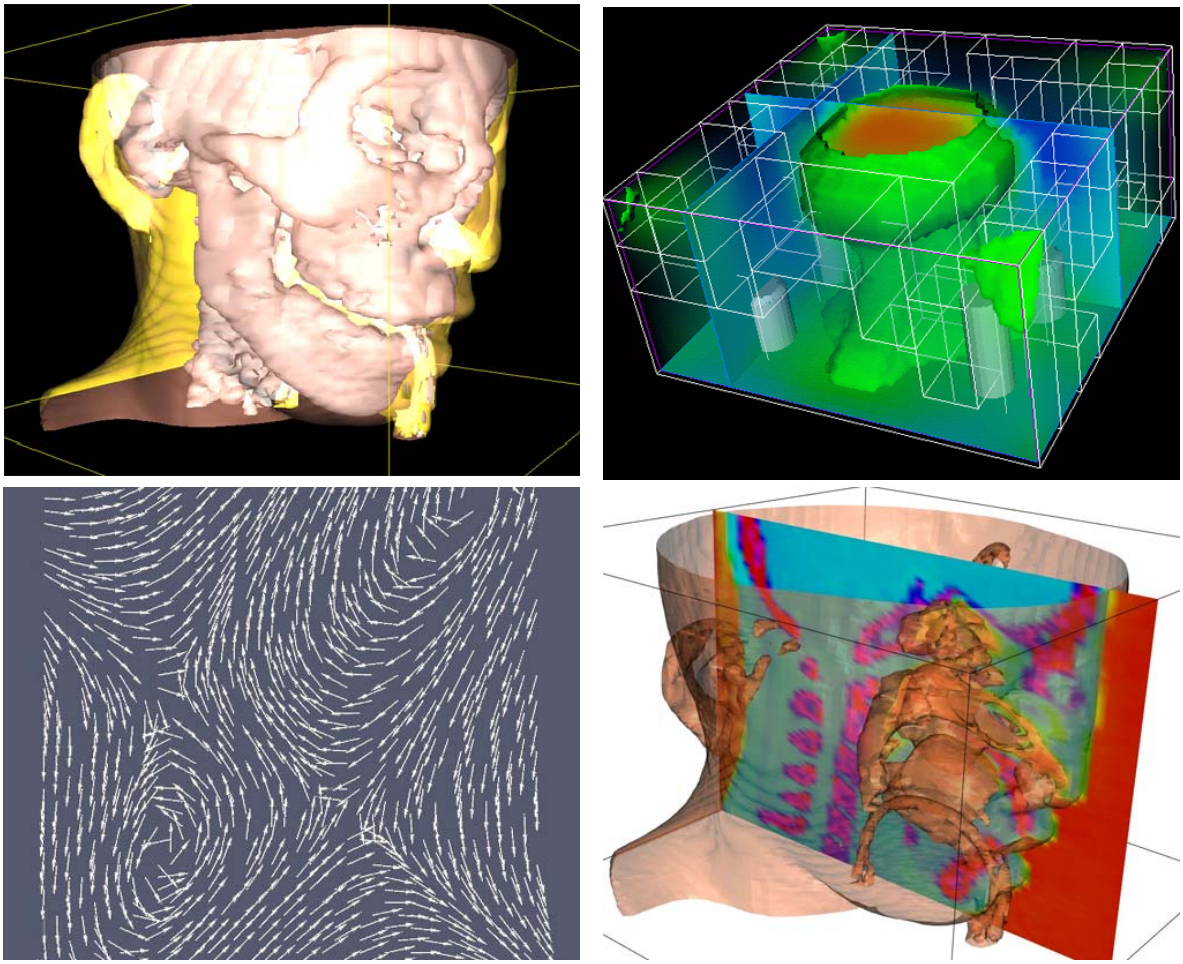
UnrealTriage is a "simulation of a mass casualty incident" based on the Unreal engine. The aim of the simulation is to engage emergency response players in a small airplane crash with 30 casualties. The players must suppress fires and locate and categorize the casualties into varying levels of urgency.

3.5 Medical Visualizations

Biomedical Visualizations can be a huge benefit in the biomedical industry. They can be used to teach medical students things such as bone, organ, or tissue identification, and can even be used as a first look at the steps involved in performing complex medical procedures [1].

3.5.1 VTK – Visualization Toolkit

The Visualization Toolkit (VTK) [17] is an open-source set of C++ class libraries organized into 'kits', containing functionality for 3D computer graphics, image processing, and visualization. It is used by researchers to build applications for biomedical, scientific and information visualization. It is not a stand-alone application, but rather it can be used as an addition to existing applications, enabling developers to build end-user products [26].



4 The Visualization Tool

4.1 Design

The design of the tool was split into three parts. The first part discusses the requirements of the visualization tool in detail. The second section talks about the Quake 3 engine itself – features that can be utilized, as well as constraints that need to be considered. The final section describes what modifications are required to achieve the tool requirements.

4.1.1 The Requirements

The main goal is to create a visualization tool with the ability to visualize a biomedical dataset, and allow multiple users on separate computers to collaborate and explore the visualization.

Visualization

- Load Complex Dataset – *the test data for this project was a 50,000 polygon dataset generated by an implementation of the marching cubes algorithm.*
- Display Dataset in real-time, from any perspective – *this would allow the user to move around the visualization, and explore it from different angles.*
- Display Graphical Glyphs – *this involves the addition of elements such as arrows and streamlines to enhance the visualization.*

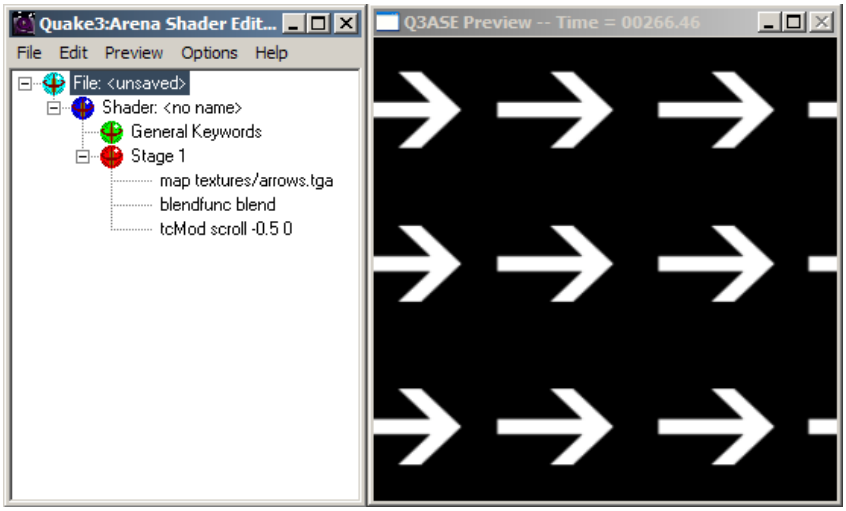
Collaboration

- Allow multiple users to connect to a visualization server – *ideally all the users would be able to connect and visualize the dataset located on that server.*
- Ability for users to all view the same dataset – *the tool should offer the ability for users to simultaneously view a dataset, rather than having to take turns.*
- Enable communication between users whilst viewing dataset – *some sort of communication, most likely text based, would help user collaboration.*

4.1.2 The Engine

For this project, Quake 3 was chosen as the game engine to be modified. There are several main reasons for this, as listed above. But there are also some additional reasons. Quake 3 was created by id software, and was programmed by John Carmack, who is one of the top game engine programmers in the gaming industry. The engine has been available long enough such that any bugs or instabilities that it may have possessed are non-existent, as a result of numerous patches.

An extremely useful feature existing in Quake 3 that wasn't mentioned before is its extensive 'shader' system. Although having the same name, these have no similarity with the shaders which are used with programmable graphics cards. The shaders in Quake 3 allow the creation and application of dynamic materials in-game. There are numerous effects, including 2D translation, alpha blending, and environment mapping, all of which are configurable by editing a .shader script file. This will come in handy when implementing graphical glyphs in the visualization. An added bonus is that several tools exist to visually create these shader files [25] (as shown in the figure below).



Another benefit of the Quake 3 engine is the 'console'. This is an area in-game which allows the user to input commands, or adjust variables. Commands can be useful functions such as 'screenshot', allowing the user to capture the current screen, whereas variables can have instant feedback in the engine – setting 'r_showtris' to 1 allows us to see every triangle drawn on screen for example.

One minor setback is the lack of official documentation regarding the Quake 3 engine. There are numerous unofficial specifications available on the internet [18, 19, 20], but these are not always 100% accurate, and may be outdated, or just blatantly incorrect. This is also a disadvantage when it comes to converting the dataset to the Quake 3 model format (MD3) as only one unofficial specification exists [18], and that document is not 100% correct.

The idea behind the implementation of this tool was to modify the chosen game engine as little as possible. Thus, existing features of the Quake 3 engine shall be used, and extended if necessary, to satisfy the requirements above.

4.1.3 The Modifications

As the source code for the Quake 3 engine has not been publicly released, the tool will need to be implemented using only the game code. The game code is freely available under a 'limited use' license, and there are dozens of websites which contain tutorials and information for extending it [21].

There are two methods of implementing the game code into Quake 3. The first involves scrapping the existing code, and writing a visualization specific library from scratch. The second method involves making minor additions to the current game-code to suit our needs. For this project, one of the aims was to modify the game engine as little as possible, thus learning the difficulty involved in implementing the visualization, so the second method was chosen.

The first step would involve loading the dataset into the engine. Since the engine does all the 3D data loading internally, the game code does not have the ability to load arbitrary 3D data into the game. Thus, Quake 3's existing code will be used, and the data will be converted into Quake 3 MD3 format prior to loading it into the game.

Unfortunately, because of the way Quake 3 deals with content during multi-player, each of the clients would need the dataset on their computer if they were to view it on an online server. However this isn't really an issue, since the dataset file size is small, and would be quick to download on most internet connections.

The only other modification required would be that of the physics engine – a visualization tool does not need gravity. A user will want to fly around the dataset, with 6 degrees of freedom. Although the game code doesn't handle the math behind the physics engine, it does have the ability to adjust user velocity and gravity, allowing us to create a 'weight-less' environment for the users to view the dataset in.

4.2 Implementation

The following sections detail the steps taken to implement the tool.

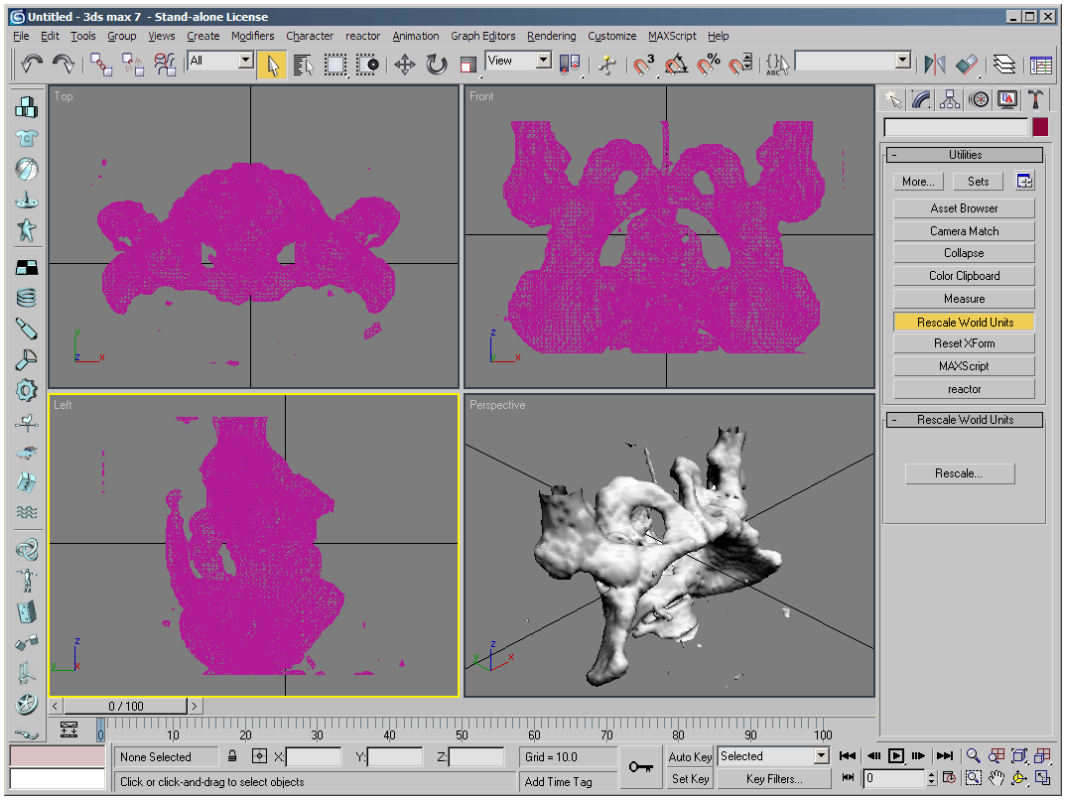
4.1.1 Preprocessing

The implementation of the tool begins with processing the data. Because of issues discussed in the design, the dataset will need to be converted to MD3 format prior to any modifications of the game code. Thus, several 3D tools were investigated. Data was provided from a 'marching cubes' based application, and exported as an ASCII based 3D Studio format (.ASC), largely because of the simplicity in writing an exporter.

At this point, it was discovered that 3D Studio Max 7 [22] did not support directly importing .ASC files in. So, a simple 3D application named 'Milkshape 3D' [23] was purchased. This tool allowed the conversion between many formats, in particular for game engines, so it allowed the conversion of .ASC format into .3DS format, and the conversion of the mesh to a Quake 3 MD3, thus allowing the direct import into the game engine.

However, one of the constraints of the MD3 data format (based on the unofficial specifications) is that the maximum number of vertices allowed in a surface is 4096. The converted mesh contained around 27,000 vertices, making it unsuitable for use in Quake 3. In hope of an error in the MD3 document, the dataset was tested anyway. It resulted in Quake 3 throwing an error, stating the mesh had "more than 1000 vertices". The MD3 format supports multiple meshes inside the one file, so the idea was to split a complex mesh into a series of simple meshes, and export it as a single MD3 file.

At this point, the mesh was imported back into 3D Studio Max, and split up, each piece containing less than 1000 vertices. It should be noted that the unofficial specifications also claim the maximum number of surfaces to be 32, but as this mesh had less than 28,000 vertices, only 28 surfaces were used. Whether this 32 surface limit is accurate or not remains unknown.



Shown above is the dataset loaded in 3D Studio Max. Although it is not possible to distinguish, the mesh is actually split into 28 pieces. The idea was that the mesh should still be identical once split up, and represent the original mesh when viewed in Quake 3. Once the splitting process was completed, the mesh was exported back into Milkshape 3D, and then converted to an MD3 file. Upon retesting, Quake 3 loaded the dataset successfully, and displayed it on the screen.

An optimal solution for this time-consuming process would be the creation of an automated tool to convert from an existing popular 3D format to the Quake 3 MD3 format. However, because of the lack of MD3 documentation and the time constraints, this tool was not created.

4.1.2 Game-code

The next step was to modify the Quake 3 game-code to suit the requirements. One of the main aims was to modify the game engine as little as possible, so steps were taken to maximize the use of existing code.

The first task involved adding an 'entity' into Quake 3, which represented the dataset. An 'entity' is simply a dynamic object in the Quake 3 game-world, with different properties depending on what the entity represents. Every object in Quake 3 is an entity, including players, weapons, rockets, and anything dynamic.

The simplest possible way to create an entity to represent the dataset was to modify the grenade launcher function. Essentially, the 'grenade' properties were adjusted to suit an entity that would eventually become the dataset. Any kind of explosions or game-play related code (such as gravity) was disabled. Then, adjustments were made to the client side, to allow for non-standard Quake 3 content to be used, and set the 'grenade' model to that of the dataset.

This was an important step, as Quake 3 has built-in protection against cheaters, in the form of content-authentication. Essentially, when a user connects to a server, the server verifies all the game content. If any data is not standard, the client is rejected from the server. Thus, by setting sv_pure to 0, we are telling Quake 3 to allow custom content.

The above figure displays the results of modifying the grenade entity to create a 'dataset' entity.

Once these small modifications had been made, the basic framework for displaying a dataset was complete. The next step was to alter the state of the users. In the original Quake 3 game, players were all affected by gravity. As it happens, Quake 3 also allows for 'spectators' to watch a game in progress, thus allowing the anti-gravity type of control that is necessary in visualization. So with a few minor modifications to the server-side game code, the user can now move freely in 6 dimensions, allowing them to view the dataset from any point.

Another function necessary for visualization is the need for dynamic visualizations – the use of graphical glyphs to represent additional information. This was implemented using the Quake 3 shader system, detailed in the design section. The shader created was quite simple, consisting of an arrow texture containing an alpha channel, and the script configured to scroll the texture along one axis. Nevertheless, the result is impressive, and shows how Quake 3 shaders could be used for much more advanced visualization tasks.

4.3 Usage

4.3.1 Installation

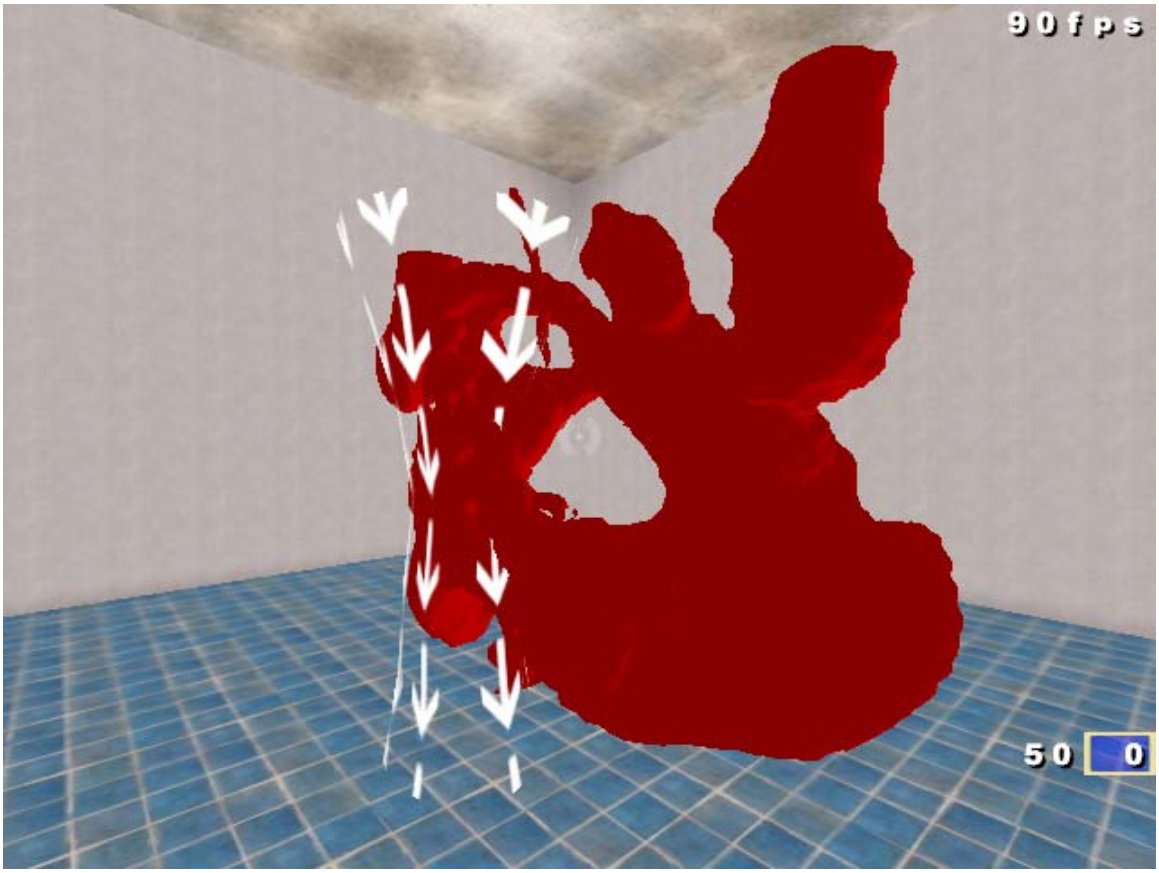
To use this tool, you will need a working installation of the full version of Quake 3 Arena. You must also have the zip file of this implementation, named "q3vis.zip". Since this visualization is based on the Quake 3 engine, the files must be extracted to the Quake 3 Arena directory. The directory names must be preserved when extracting the archive, so that the tool will function properly. Once the archive has been extracted, there will be a file named "vistool.bat" located in the main Quake 3 Arena directory, which will begin the visualization tool when executed.

4.3.2 Visualization

To use the tool to visualize the dataset, simply run the "vistool.bat" file. This will launch Quake 3, and automatically load the modification. Once the game has loaded, the user will drop into a simple room. At this point, the user should move to a position in the level which isn't near a wall, so the dataset can be placed with space to move all around it.

To move around in the game-world, the simplest method is to use the W, A, S, and D keys. These correspond to the arrow keys (Up, Left, Down, and Right respectively) and allow movement forwards/backwards and left/right. The Q and Z keys can be used to move vertically up and down. Finally, the mouse can be used to rotate the view around (similar to the way one would move their head around).

The user can activate the console by pressing the [`] key (normally located to the left of the 1 key). The basic dataset that was converted for this project can be loaded by typing "/vis_load" (without speech-marks) and pressing enter.

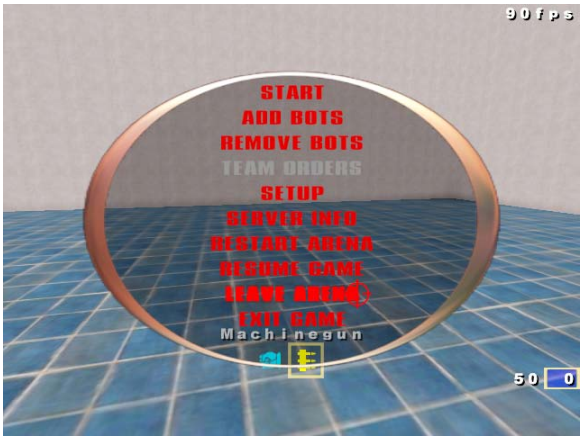


Once the dataset has been loaded, the user is free to fly around and observe it from different angles. The figure above shows what the screen should look like once this point has been reached. Although it is not possible to see from an image, the arrows are actually animated, and scroll down in the direction they are pointing.

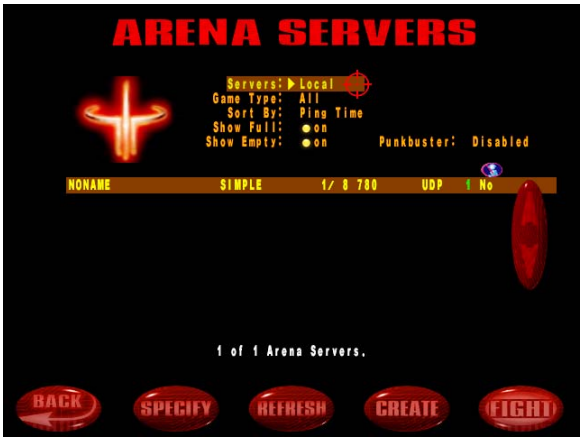
4.3.3 Collaboration

There are two main methods of collaboration for the visualization tool. Both require one computer to be the server, and the others to connect to it.

The first method is simpler, but takes a little longer. Once the first computer has been set up, any additional users who wish to connect to the server (the first computer) must press [Esc] once the modification has launched, and click “Leave Arena” as shown below-left.



Once the user has entered the main menu (above-right), they need to click Multiplayer. This will take them to the multi-player menu. The user must verify they are browsing ‘Local’ servers (shown below-left) providing they are on a local area network with the server. Once they find the server they wish to connect to, they click the ‘Fight’ button (below-right).



The second method is slightly more complex, but much quicker. It is also the easier solution for connecting through the internet. First, the IP address of the server machine has to be known. Then, any additional users wanting to join simply activate the console by pressing [`] and then typing "/connect IP" where IP is the IP address of the server machine. The user should connect after a few seconds, and the drop into the game-world along with any other users currently on the server. Any datasets present in the level should be visible instantly.

Chatting is quite simple too. The first step is to set an appropriate user name. This is done by activating the console, and typing "/name" followed by the desired user name. At this point, the console can be deactivated, and the user can send a message to the other users on the server at any time by pressing the 'T' key. Pushing this shows a line at the top of the screen saying "Say: ". From this point, the user simply types in a message, and upon pressing the enter key, the message is sent to all other users on the server.

5 Evaluation

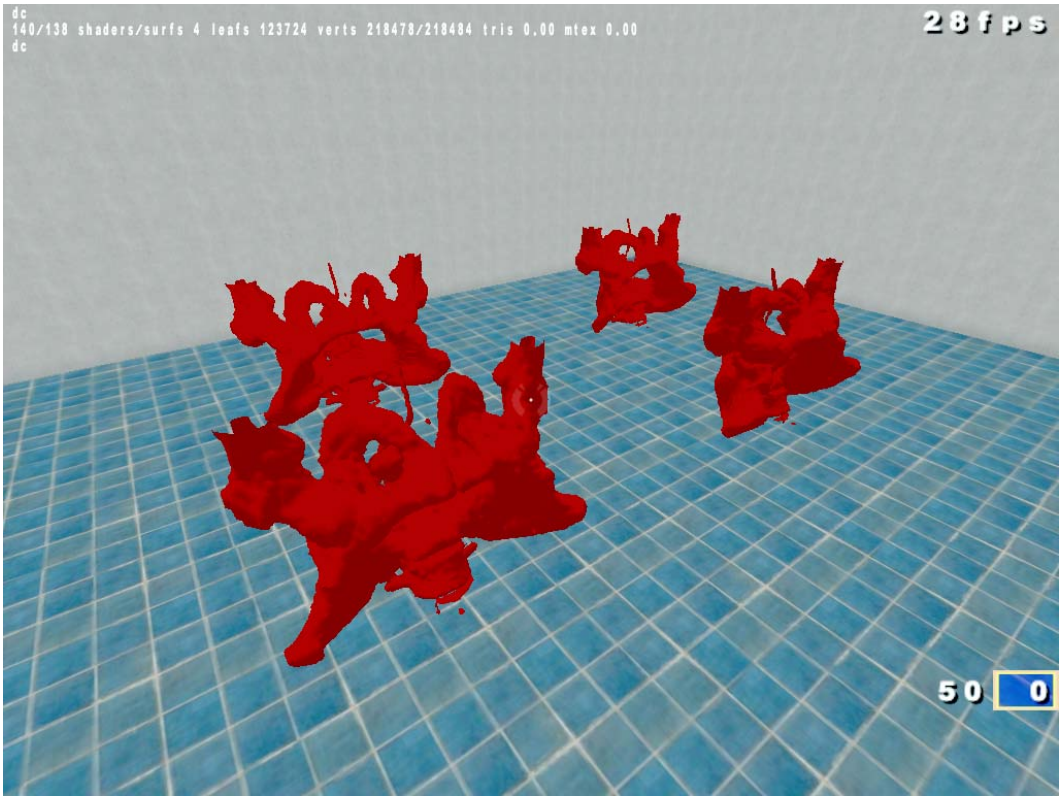
5.1 The Tool

The tool is quite simple in design, which is a result of minimizing modifications to the Quake 3 game-code, and still achieving basic visualization. It meets the requirements of loading an arbitrary dataset, and being able to visualize it. It also allows for the use of custom animation effects on materials, which is more of a proof-of-concept for the tool's capability of representing graphical icons. Unfortunately, many of the features in a normal visualization tool do not exist in this implementation. This is due in part because of time constraints, but also because of the limitations of the Quake 3 engine.

The collaboration portion of the tool worked with very little modification to the existing game-code. This is one of the many benefits of using a game-engine to implement the visualization tool, as the code is already complete, working, and tested. The addition of the visualization of the dataset ties in with no issues to the Quake 3 multiplayer code. It allows any users (who have previously acquired and installed the dataset) to connect to a server and collaborate with other users.

5.2 The Quake 3 Engine

The Quake 3 engine is ideal for rendering complex datasets. A small test was performed using the dataset from the project, to test the speed of the engine when rendering a considerable amount of polygons. A number of data sets were loaded into the visualization tool simultaneously, and using Quake 3's built-in performance statistics, results were acquired.



As you can see from the figure above, the engine can handle 4 datasets quite easily, which comes to a total of 218,484 triangles. And the frame-rate is still around 30fps, which is easily enough for visualization of a dataset. As a point of reference, Quake 3 ran the original dataset at around 90fps. This means the engine can easily render between 4 and 6 million triangles per second on a medium specification machine, which is more than enough for most visualizations.

However, the engine contains numerous limitations. These are all a result of some functionality that is built into the engine code, and since the source code is not yet publicly available, there was no way to resolve these issues. With access to this code, one could implement loading of an arbitrary 3D file format, possibly a well-known format used in other biomedical visualization tools.

Another side effect of using the Quake 3 engine is the time taken to learn the inner-workings of the game-code. With no official documentation available, it was an involved process, and only community work could be relied upon for reaching the desired goals.

6 Conclusion

Game engines can – and should – be used as a base for a collaborative visualization tool for biomedical data sets. With the amount of research and work put into game engines, they serve as a perfect framework that has already been tested and optimized to the extreme. With the addition of multi-player functionality, the visualization tool can be easily extended as a collaborative tool. Moreover, game engines in general have very intuitive interfaces, meaning very little modification would be needed if support was being added to create an easy-to-use interactive visualization tool.

In general, it would be better to use an engine which is fully open-sourced, although (as mentioned before) the Quake 3 engine will become open-source in the near future, allowing any further work being done on this topic to utilize the Quake 3 engine to the full extent. The only remaining issue would be the lack of documentation on the Quake 3 engine. This could perhaps be resolved by the use of a different engine – one of the selling points of the Cipher engine is its extensive documentation [24] – although this would require purchasing licenses and such, and may not be a feasible solution. However, the rendering power and network capabilities of the Quake 3 engine in particular outweigh the flaws, and if more time was spent on the development of the tool, it could have a great deal more functionality added, resulting in better performance as a true visualization tool.

7 Future Work

There are countless possibilities for extensions on this topic. Some of them are listed below, split into two sections – visualization, and collaboration.

7.1 Visualization

- Automated conversion

  The creation of a tool which automatically converts from any desired format to the Quake 3 MD3 format would reduce the time needed for preprocessing.
- Ability to enable/disable specific visualizations

  Certain datasets may have more than one visualization mode, and a visualization tool should have the ability to switch between modes.
- Client side visualizations

  A user may want to view the visualization in a different mode, and should be able to, without affecting all the other users connected to the server.
- Ability to customize visualizations in-game

  A simple interface on the client side could allow modification of the visualizations in-game, allowing more flexibility on the user-end.
- Proper GUI designed for visualizations

  A custom created user interface, allowing users a full range of options for loading datasets, changing visualizations, and other functions.
- Ability to view datasets in alternative ways (rotate set, rather than user)

  The user should be able to use the mouse to rotate the dataset, rather than having to manually move around the dataset.
- Per-polygon detection (if possible) so users can tag individual triangles

  This is a hard-coded limitation with the Quake 3 engine; however the inevitable release of the engine source code will make this possible.
- Stereo-vision (viewing datasets in actual 3D)

  The Quake 3 engine actually has built-in support for Red-Blue stereo separation to create 3D. This could be extended for visualizations.

7.2 Collaboration

- Better text chat support

  The current Quake 3 system is only designed for minimal chatting in-game. This could be extended to allow public/private conversations and such.

- Voice-Over-IP support

  Being able to discuss certain aspects of datasets by actually talking would be a huge benefit for users collaborating from various parts of the world.

- Display where current users are in the game-world

  A simple entity showing users position, as well as their name hovering above them, would allow users to follow each other and interact.

- Possibly webcam support, to discuss with other users

  An extension of the Voice-Over-IP would be to implement web camera support, so users could see and talk to each other regarding the datasets.

- List of currently connected users, along with their location

  This could help in distinguishing different users, by having an active list of who they are and where they are connecting from (around the world)

- Ability to interact with datasets (point out problems, etc)

  An extension of the visualization problem – it would be useful to be able to point out features on the dataset and have other users see the same thing.

8 References

1. Blackman, *S. Serious Games...and Less!* Contributions to ACM Siggraph Computer Graphics Volume 39 Issue 1 (February 2005), pp. 12-16.
2. Rhyne, T. *Computer Games and Scientific Visualization.* Communications of the ACM Volume 45 Number 7 (July 2002), pp. 40-44.
3. Stang, B. *Game Engines Features and Possibilities* (2003) http://www.student.dtu.dk/~s958608/GameEngines2.pdf
4. OpenGL – http://www.opengl.org/
5. DirectX – http://www.microsoft.com/windows/directx/default.aspx
6. GeForce 4 Ti - http://www.nvidia.com/page/geforce4ti.html
7. Lua - http://www.lua.org/
8. OGRE - http://www.ogre3d.org/
9. Irrlicht - http://irrlicht.sourceforge.net/
10 The Nebula Device - http://www.radonlabs.de/nebula.html
11. Cipher Engine - http://www.cipherengine.com/
12. Unreal Engine 2 - http://udn.epicgames.com/Two/UnrealEngine2Runtime
13. ID Software - http://www.idsoftware.com/
14. Deep Creator - http://www.righthemisphere.com/products/dcreator/
15. Unrealty - http://www.unrealty.net/vsmm99/
16. UnrealTriage - http://www.ists.dartmouth.edu/library/58.pdf
17. VTK - http://public.kitware.com/VTK/
18. Quake 3 MD3 unofficial specifications - http://www.icculus.org/homepages/phaethon/q3a/formats/md3format.html
19. Quake 3 Structs - http://linux.ucla.edu/~phaethon/q3mc/immutable1.html
20. Quake 3 Unofficial Shader Manual - http://qeradiant.com/manual/Q3AShader_Manual/index.htm
21. Code3Arena - http://www.planetquake.com/code3arena/
22. Autodesk 3DS Max - http://www.discreet.com/3dsmax/
23. Milkshape 3D – http://www.swissquake.ch/chumbalum-soft/
24. Cipher Documentation - http://www.cipherengine.com/sampledoc.php
25. Q3ASE - http://www.bpeers.com/software/q3ase/
26. VTK Review - http://www.crd.ge.com/~lorensen/ExtremeTesting/ExtremeTesting.htm
27. The power of 3D in biomedical visualization – http://vfxworld.com/?sa=adv&code=57c5ed8a&atype=articles&id=2437&page=1