# Multi-Platform Multi-User OpenGL Environment

December 9, 2006

The University Of Reading
Department of Computer Science
School of Systems Engineering
3/CS/7N - Computer Science Project
Final Project Report
Multi-Platform Multi-User OpenGL Environment
Supervisor:Dr James A.D.W. Anderson
Second Reader: Dr Rachel McCrindle
Author: Jonathan Waller

**Abstract**

The aim of this project was to create a multi-platform, multi-user 3D virtual environment. An environment that can be personalised, explored, and used for conversations in many languages.

The project uses OpenGL to create a graphically interesting world. Light colour and intensity changes throughout the day, stars become visible at night. The landscape is seemingly infinite in size and can hold many millions of objects.

The project has designed to be multi-platform and fully support the Unicode character set.

The main drive behind this project was the desire to remove boundaries from users, allowing all to enjoy a 3D virtual environment. By fully supporting Unicode, conversations can contain a much wider range of characters, thus enabling native communication in practically all of the world's languages. Furthermore, by designing the software to be multi-platform, users of non Windows operating systems gain access to 3D interaction not previously available.

Many creative people yearn to demonstrate their artistic talents, so personalisation was a key feature of this project. Allowing users to explore the world and come upon undiscovered areas that have been adapted by another user, allows a much stronger connection to the medium than instant messaging may have. The intention was to create an engaging and enjoyable experience for all users.

The aims of this project have been realised. The environment that can be personalised and explored. Unicode chat has been implemented, and tested in a variety of languages.

# Contents

# 1 Introduction and Specification

## 1.1 Key terms

Throughout this document a few unfamiliar terms have been mentioned. To clear up possible confusion, the terms are defined here.

- Jasper - Refers to the implementation of the client and server, used as the online name for this project.
- UniServer - The server part of Jasper.
- UniApplet - The client part of Jasper. Can be run as an web-based applet or standalone if required.
- Instance - An individual item in the world, each has a unique identifier. Each instance is associated with a file to define its 3D shape. Many instances can have the same file. (A forest of identical trees for example)

## 1.2 Primary objectives of the project:

- Multi-user 3D environment

  It is a complaint of some that chat rooms and instant messaging are a "cold" medium, as users do not receive the same kind of emotional response as they have when communicating with the person face-to-face. When this is not possible, allowing many users to explore a shared world and meet each other helps to build community and provides a stimulating atmosphere. Users will be able to adapt their surroundings to their desires and these changes will be kept even when they leave the world.

- Multi-platform environment

  In the main, access to 3D environments have been the preserve of the Windows user. Enabling the client program to run on multiple operating systems, will allow users of Windows, Linux, Unix, Mac and Solaris, for example, to explore the environment and communicate with each other.

- Unicode character set support

  At the time of writing the author can find no web based communications tool that supports Unicode characters. It is possible to download specialised instant messaging software with support for Unicode characters, but some corporate and academic networks do not allow their users to install software on their computers.

## 1.3 Secondary objectives of the project:

- User customisation

Allowing users to modify the world they inhibit allows a more interesting environment to arise. Many people are interested in showing their creativity and will create special structures to display to others. This interest can be seen in the construction of DAAP, "The first world of virtual sculpture on line", where undergraduate 3D artists display their project work in a virtual environment.

- Improve realism of world.

  By using a variety of effects the virtual world can become more realistic. Example techniques could include lighting and terrain effects. Lighting effects and sky colour can be based on the client's local time.

- Web based

  It is important to allow the project to be web based, as some users cannot install software on the computer they are using. To this end it has been decided to create a Java applet than can run through a web browser.

## 1.4 External Review

### 1.4.1 Similar projects

The projects show below demonstrate some of the functionality in Jasper, such as a 3D environment or Unicode support.

- MSN Messenger

  MSN Messenger is an popular instant messaging tool which allows users to chat to one another. No 3D environment is supported. MSN Messenger supports Unicode characters, but only works on Windows and Mac. It also has to be installed with Administrator privileges.

- Activeworlds

  Activeworlds is a client that allows access to a collection of 3D environments. Facilities exist to import objects into these environments and to talk with others. The client only runs on Windows systems. Unicode support in Activeworlds was only added after Jasper was started, but this support seems unreliable and the author has had limited success with non-roman character input.

- VRML Homesteads

  VRML Homesteads are VRML scenes that are connected by hyperlinks. The scenes can be explored but are single user.

### 1.4.2 Interest to the computing community

Jasper fills a niche which the projects above do not. As well as supporting more platforms, such as Linux and Unix, Jasper allows the reliable use of Unicode in conversations.

Another item of possible interest to the computing community is the "Goo" algorithm. The "Goo" algorithm generates the landscape tiles used in the creation of the vast continuous landscape seen in the environment. This algorithm runs quickly, and creates natural looking features. Depending on the level of recursion, the output detail can be tuned to the desired amount. This algorithm is of my own design.

### 1.5 General statement of the problem

The project must be able to support multiple users in the same online environment. It must run on more than one system architecture and must support Unicode characters, to allow conversation in many languages. The secondary requirements for this project are the ability of the user to modify the world, to allow the project to run through a web browser and to improve the realism of the world with graphical effects.

Jasper was created to allow people on many operating systems to use the same 3D environment. The aim of this project is entertainment and communication. To the author's knowledge there are no multi-platform applications of this type that support Unicode characters. Jasper was created to fill this gap.

A common problem is that applications have been customised for one character set, and therefore will incorrectly render text in another character set. This is the case with software such as ICQ Lite Chinese Simplified, which due to encoding methods makes it impossible to show characters outside of Chinese simplified encoding, or to mix two non-roman character sets (such as Arabic and Korean). This is an essential feature for environments where many languages may be used.

## 2 Analysis and Design

### 2.1 Why Java was chosen

One of the key reasons was that programs written in Java can easy run on multiple physical platforms through a virtual machine. Another reason was that Java can be used to create applets, which run through a web page.

Other languages that can be used to create multi platform applications would have to be re-compiled for the new architecture. The programs created in this way would have to be downloaded, and not be web based, which was one of the secondary requirements of this project.

.NET would not need to recompiled, as it has a similar structure to Java's "byte-code", and could be run on many platforms, due to the open source .NET implementation "Mono". However connecting .NET to native OpenGL libraries, would limit it to one platform only. (Unlike OpenGL for Java, which is a multi platform wrapper)

## 2.2 Why OpenGL was chosen

A graphical framework was needed as the author did not want to render the scene in software, as this would be very slow. The two most widely used graphical frameworks are DirectX and OpenGL. OpenGL is cross platform, however DirectX only runs on Windows systems. Finding the "OpenGL for Java" classes finalised the author's decision to use OpenGL.

## 2.3 Applet Classes:

**CmdThread** Manages thread that connects to the server. Deals with user commands and commands sent from the server. Periodically informs the server of the new avatar position. Forwards messages to the InstanceMngr, Server and LogBox.

**Converter** The abstract base class for all the file converters. Contains helper functions for converting between types, generating normals and copying arrays. Contains model arrays that can be accessed in a uniform way, independent of file type.

**ConverterASC** Allows 3D Studio Max ASCII model files to be converted to a generic model format.

**ConverterMTL** Allows Alias|Wavefront material files to be converted to a generic material format.

**ConverterOBJ** Allows Alias|Wavefront Object model files to be converted to a generic model format.

**ConverterPC** Allows JOF Point cloud model files to be converted to a generic model format.

**ConverterPNG** Allows Portable Network Graphic image files to be converted to a generic texture format.

**ConverterTGA** Allows Truevision Targa image files to be converted to a generic texture format.

**FileObj** Contains arrays holding information about a file in a generic model, material or image format. Contains a render() procedure to render the file to the OpenGL environment.

**FileMngr** Manages all the FileObj objects. Requests Download and Conversion of required files. Acts as a cache for downloaded files of all types. Returns a FileObj when given a URL address.

**GlobalDate** Keeps track of server time, used when creating the sky and lighting colours.

**InstanceObj** Contains information about the location and rotation of an object, its Unique Identifier and the URL of the file it is rendered with. Contains a render() procedure which calls the render procedure of its associated FileObj.

**InstanceMngr** Manages all InstanceObj objects. Allows addition and deletion of instances. Responsible for deleting instances of objects that are outside the client's view distance.

**JTGATextureLoader** Inherited from GL4Java's TGATextureLoader. Allows the loading of textures from an InputStream, not a URL. This is required as all file transfers are proxied though the server.

**LandscapeMngr** Responsible for all processing related to the landscape. This includes creating required landscape tiles as the user walks around the world, blending tiles together and applying colour to the landscape based on the landscape features.

**LogBox** Inherited from JTextArea. A multi line text area extended to support line wrapping and Unicode fonts.

**ObjConvertThread** Manages a queue of objects to be converted. Selects and runs an appropriate converter to convert a file into its generic form.

**ObjDownloadThread** Manages a queue of objects to be downloaded. Downloads requested files. Limits files to under 500KB.

**OpenGLCanvas** Stores the user's position and rotation in the world. Manages time-based lighting. Generates and draws the radar and sky dome. Receives keyboard input for user movement. Calls rendering routines in the Instance objects and Landscape object.

**ProtocolStringHelper** De-constructs and constructs protocol compliant strings. Contains functions to return part of a protocol string such as the command or the sender. Allows easy construction of protocol compliant strings that can be sent to the server.

**TextEnterBox** Inherited from JTextField. Single line text box extended to support Unicode fonts. When Enter is pressed, the input is sent and the text box cleared.

**UniApplet** Main applet class. Generates the user interface, and sends all user input to GLCanvas. Allows the applet to run within a web page or as a standalone Java application.

## 2.4 Server Classes:

**ProtocolStringHelper** De-constructs and constructs protocol compliant strings. Contains functions to return part of a protocol string such as the command or the sender. Allows easy construction of protocol compliant strings that can be sent to the clients.

**InstanceSObj** Data only object (no methods) to hold location, rotation, Unique Identifier and URL of each model instance in the world. Used within the WorldDB class.

**sCmdThread** There is one sCmdThread for each client connected to the server. This class processes the commands and chat messages sent from its client and sends commands and chat messages to the client when required. Changes to the world near to the user are sent to the client.

**serverCmd** This class manages messages (chat and command) that are sent to and from the clients. This class listens for new connections and creates sCmdThread threads to deal with them. This class broadcasts chat messages to all other clients. This class initialises the WorldDB object.

**serverObj** This class manages the file proxy system. This class listens for new connections and creates sObjThread threads to deal with them.

**sObjThread** This thread allows the proxy of files. The client requests a file by its URL, and this class will either download and send the file to the client or inform the client that the file cannot be downloaded.

**timeOutCheck** Sends periodic messages to check whether the client is still connected, which the client is expected to reply to. When there is no response the client is given 3 warnings before being disconnected.

**WorldDB** Stores every instance in the world (e.g avatars, huts, trees). Allows objects to be added and deleted from the world. Initialises the world to contain 40 trees, two houses and a signpost. Contains the sendObjectsInSightOf() procedure, which informs the client of all instances that are close to the user.

**UniServer** Starts serverCmd and serverObj.

# 3 Development and implementation

## 3.1 Development method.

The project was developed using the evolutionary prototyping model. This involved creating a working prototype as soon as possible, and then improving it in steps, each cycle having a running application to test. The tree like structure of dependencies in the client restricted this slightly, but gave a much stronger structure to build the rest of the application around.

## 3.2 Command Infrastructure

The project consists of two parts, a server (UniServer) and a client (UniApplet). Both are written in Java. The server listens on two ports for incoming connections. The client is a Java applet loaded in a web page, when the page is loaded the client connects to the server. Many clients can be connected to the server at the same time, so that many users can co-habit the same 3D environment.

There is well defined, extensible, human readable protocol that the clients and server must adhere to. Version problems are very unlikely due to the client being a Java applet and being re-downloaded each time it is run.

### 3.2.1 Server Operation

The server listens on two ports for incoming connections, one "data" port and one "command" port.

**Data Port:** Due to security restrictions for applets, applets can only open connections to the web server that they were downloaded from. This means that an applet cannot directly download a file if it is not stored on the same web server. The data port solves this problem by creating a simple proxy so that the client applet can request a file from UniServer, and UniServer will download the file and send the data to the applet that requested it. This connection is used to download all the files that the applet needs, 3D models and images for example.

**Command Port:** A central server stores all the information about the world, including the locations of objects within that world. When a client connects to the server, information about the closest objects to that player is sent to the client.

When the applet first loads, it will connect to the command port on UniServer. UniServer will then send a list of all the objects close to the player's location, object information is in the form:

*obj UID x y z rx ry rz URL*

Each instance of a 3D model has a UID (unique identifier), this allows the client and server to keep track of all the instances of each object. The UID is guaranteed to be globally unique.

Some objects (such as the sky dome) are always the same relative distance from the player. These objects are sent in the same way, but use *wobj* as a specifier.

Should an object move in the scene an *obj* line is created and sent to all connected clients who are close to that object. The client sends *obj* lines to the server to inform it that the avatar (or another object under client control) has moved.

### 3.2.2 Client Operation

All of the rendering is performed in OpenGL in the client program. This is achieved by using the "OpenGL for Java" classes.

**3D model rendering:** When the client first connects it receives a list of near objects from the server in *obj* line form (shown above).

This line updates (or adds to) the contents of the instance list, depending on the UID. If the associated 3D model file is not in the File Manager Cache, It is then downloaded and converted. All objects within the instance list are then rendered to the screen. Objects that move far away from the avatar are removed from the instance list.[1]

This instance list is then rendered to the screen using the associated position and rotation data.

## 3.3 Graphical effects

### 3.3.1 Time of Day Lighting

When the client connects to the server it is sent the server's local time. This ensures that all clients have an identical time which is used when creating the sky and lighting colours.

### 3.3.2 Dynamic sky colour gradient

The midnight sky gradient consists of a deep blue at the horizon, fading to black directly above the viewer. The noon sky gradient consists of a light blue at the horizon, fading to a mid blue. The ratio of day to night colours is not a linear relationship but follows a cosine wave, meaning that the greatest changes in sky colour will occur around dawn and dusk, around noon and midnight, the change will be more gradual.[2]

### 3.3.3 Starfield at night

The star filed is random, but based on a fixed seed, so that all clients will get the same star field. The stars are clearest at midnight (server time), but fade

[1] See figure 6 in pictorial appendix for a data flow diagram of the 3D object download and conversion process.

[2] This relationship is demonstrated in Figure 5 of the pictorial appendix.

into the sky gradient gradually, becoming indistinguishable at 6am and 6pm, and throughout the "daytime" (6am to 6pm). Stars closer to the horizon blend to the horizon colour.

## 3.4 Landscape features

### 3.4.1 Infinite Landscape Generation

Tiles are generated when they are within 40 meters of the user. As can be seen from Figure 1 in the Pictorial Appendix, as a user walks into an area requiring a new tile, the new tile will be generated, blended with existing tiles and rendered.

Tiles are generated by the client as required. Each tile is generated by a seed that is dependent on its global coordinates. This means that when a tile at a particular location is generated concurrently on two clients, the shape of the landscape tiles will be the same.

As the seed can be calculated by the client, as it knows the avatar's location, no landscape data has to be sent to or from the server for consistent landscapes across all clients.

The Goo algorithm creates independent tiles. To ensure that the tiles knit together at their edges, a blending technique has been implemented.

### 3.4.2 Landscape Tile Blending

To create a smooth landscape from the generated tiles, the tiles have to be blended together. There are two 3x3 grids, making 18 landscape tiles that are generated.[3] The second plane of tiles is offset by half a tile width in the x and z axis, so that the middle of the top 9 tiles overlay the top left corners of the lower 9 tiles. To generate the height at a particular point on the landscape, the two heights at that point on the landscape are added together, with a bias on which is closer to the centre of a tile.

To calculate a point that falls halfway between the middle of the tiles, half of the height will be derived from one tile and half from another. The advantage of this is when the sampling point approaches the edge of an overlapping tile, the influence of that tile becomes zero, and the middle of the other tile is used, so that the areas where the tiles would not bind together are of no influence, and the landscape is smooth.

To demonstrate this, please regard Figure 4 in the pictorial appendix. The X is closer to the centre of tile A than tile B. The calculated height at point X, will be biased toward the height at X on tile A, with B having a lesser influence. As X approaches the edge of tile B, the height of B has less and less of an influence on the output height. The following equation describes this relationship:

```
Height=(H(A)*Distance to centre of B)+(H(B)*Distance to centre of A)
```

[3] See Figure 3 in the pictorial appendix for a diagram of this.

Where H(A) is the height at X on tile A. The distances to the "centre of A" and to the "centre of B" are normalised, so that they sum to 1.

### 3.4.3 Goo algorithm

Tiles used for the creation of the vast landscape are generated with the "Goo" algorithm. One such tile is shown in Figure 2 of the pictorial appendix. The "Goo" algorithm takes a seed as an input. Given the same seed, the same tile will be generated.

The Goo algorithm seeks to create a natural looking landscape shape over a square tile. Goo generates the heights of each of the verticies of the tile. Other stages of landscape generation, such as blending and colouration are added later.

**The parameters given to the goo algorithm are:**

- Dimensions of tile to be generated (top, left, width, height)
- Minimum and Maximum heights of the landscape
- Seed for random number generator

It works as follows:

- 1) 4 random numbers within the height range are generated, then applied to the 4 corners of the given area
- 2) If the area to be generated is 2x2 then end this function
- 3) Of the corner's heights, find the lowest and highest in the area.
- 4) Generate a random number between these ranges and apply it to the following points:

  i) Middle 4

  ii) Two on left middle and 2 on right middle

  iii) Two on top middle and 2 on bottom middle

  [At this point the area is blank apart from 12 coloured points]
- 5) Call 2 onward as a function, recursively, giving the top left quarter as the new area
- 6) Call 2 onward as a function, recursively, giving the top right quarter as the new area
- 7) Call 2 onward as a function, recursively, giving the bottom left quarter as the new area
- 8) Call 2 onward as a function, recursively, giving the bottom right quarter as the new area

This algorithm is recursive. Changing line 2 to a higher area will decrease the resolution of the landscape, and speed up the algorithm.

### 3.4.4 Dynamic land colour

A uniform green landscape would be uninteresting. The landscape colour is calculated from the gradient at each surface Each point's colour is calculated separately. To find the gradient at a point on the landscape, two neighboring points in the x and z directions are found, and the change in y for both of the points is recorded. Both of the gradients are calculated and averaged. This gradient value is then used to calculate the ratio between the two landscape colours. On surfaces with a gradient of 0 (flat ground), the surface is coloured a dark green. As the gradient increases the ratio of dark green to light brown changes so that vertical slopes, such as cliffs, have a light brown colour. The point gains a mixture of these two colours and OpenGL blends between these.

## 3.5 File Converters

### 3.5.1 ASC, PC and OBJ Model import

Converters have been added to convert ASC, PC and OBJ files into a generic model form, which can be read by the application.

- ASC files are 3D Studio Max ASCII files. The format stores verticies, face and face colour information (RGBA).
- JOF Point cloud (PC) files are a simple format to store the verticies of an object. No material or face information is stored. In the environment PC models have been used to create explosion effects and a transparent-looking tunnel.
- OBJ files are Alias|Wavefront Object files. The files store faces and verticies, as well as material information (diffuse, specular, ambient, emission). Materials can be assigned textures. This model format is useful when you wish to have more control over the material of your object; causing it to glow or become more shiny for example.

### 3.5.2 Extensible Converter Architecture

All converter classes are inherited from the abstract Converter class. This provides arrays to store model, image or material details in a generic form. It also contains the abstract function doConversion().

New converter types that have been inherited from the Converter class must implement doConversion(). This takes the raw data from the file, and converts it into a generic model, image or material form that can be read in a consistent way for all file types.

This allows programmers to create a converter any kind of image, 3D model or other file type, provided they abide by the simple interface.

A prerequisites system is in place to ensure that all objects are processed in the correct order. For example, OBJ files may store material information in a separate file, linked to from within the OBJ file. The 3D model cannot

be converted into an array representation until the materials are known, so the model processing is halted, the material file is downloaded and processed and then the model file can be processed.

## 3.6 Specification of programs

There are two programs that make up Jasper. UniApplet and UniServer.

### 3.6.1 UniApplet

UniApplet is the client which runs either as an applet or as a standalone application. It connects to the server, and downloads Instances close to it in the world. Any associated files that it does not have in cache are downloaded, and converted into a generic form for their type (model, image or material file). The interface then renders all of the Instances in the scene, by referring to their associated model files.

The user can walk around the environment, with new landscape tiles being generated if they enter a new area. The user's location is periodically updated so the server can inform all the other clients of this user's new location.

Users can select and add Instances to the environment, with an associated object. These can be saved to the server, for all users to see.

### 3.6.2 UniServer

UniServer is the server part of Jasper and runs on a computer permanently connected to the Internet. The server keeps track of all instances in the world and informs users if there is an instance change within their field of view. The server will broadcast all chat messages sent from a user. As a separate service the server operates a proxy for files. By sending a URL to the open port, the file will be sent back back the server, or an error if the file cannot be found.

## 3.7 Other Information

### 3.7.1 Technical difficulties experienced

The default version of the VM which comes with Internet explorer, does not support Swing, which was used to draw the components on the user interface. This was solved by using a Third party Swing.Jar file from BioCoRE.

### 3.7.2 Data structures employed

Vectors are used to store lists of objects. This technique is used in FileMngr to hold a collection of FileObj objects, containing information about files that have been downloaded. Vectors are also used on the server and the client to store all of the world instances.

3D models are converted to a generic form and stored within the arrays of a FileObj.

## 3.8 Data input and Formats

### 3.8.1 PC

JOF Point cloud files are a simple format to specify the points of an object. It was initially used as a simple converter class tester, but many users have found it useful for explosion effects.

The file format is as follows:

```
256
X:1.027030 Y:0.123684 Z:-0.634111
X:0.945148 Y:0.535328 Z:-0.634111
X:0.711971 Y:0.884303 Z:-0.634111
X:0.362996 Y:1.117480 Z:-0.634111
...
```

The first line states how many lines to read, one line per point. And then each subsequent line states the X, Y and Z coordinate of a point. The size the point is drawn on screen is fixed at 5 pixels.

### 3.8.2 ASC

The 3D Studio Max ASCII file, is a human readable format which stores vertex, face and colour information as well as smoothing. The format is shown below:

```
Named object: "Sphere"
Tri-mesh, Vertices: 467 Faces: 938
Vertex list
Vertex 0: X:-0.0633293 Y:0.123308 Z:3.78605
Vertex 1: X:-0.0633293 Y:0.123308 Z:0.0
<Many lines removed>
Vertex 466: X:-3.47911 Y:-1.46055 Z:2.17891
Face list
Face 0: A:2 B:3 C:0 AB:1 BC:1 CA:1
Material:"r10g255b3a0"
Smoothing: 1
Face 1: A:3 B:4 C:0 AB:1 BC:1 CA:1
Material:"r10g255b3a0"
Smoothing: 1
```

Each face is assigned a colour RGBA, which allow the surface to be transparent. Many submodels may exist in one file.

The "Smoothing" field is ignored. All objects in the environment are drawn with smoothed colours but non-smoothed normals.

### 3.8.3 OBJ

Below, the OBJ format is described.

Comments are signified by a line starting with a hash symbol. If an MTL file is associated with the OBJ file then this will be specified with "mtllib". Lines starting with v, vt and f signify verticies, texture verticies and faces respectively. "usemtl" refers to a material change. Material information is found in the MTL file.

A sample OBJ file is shown below.

```
# Wavefront OBJ file
mtllib 7g_hut.mtl
# object MBconv
g MBconv
v -2.64874 -0.0815319 0.732645
v -2.64874 -0.0780487 0.725964
v -2.64874 -0.0780487 0.736369
<Many lines removed>
# 847 verticies
vt 2.92053 2.6726 vt 3 2.6726 vt 2.92053 2.91641
<Many lines removed>
# 703 texture verticies
usemtl MAT_ACDEAC
f 834/1 840/2 833/3
f 832/4 833/3 839/5
f 833/3 840/2 839/5
<Many lines removed>
# 1641 faces
```

### 3.8.4 MTL Files

OBJ files may specify a MTL file, which is used to store materials values and texture filenames associated with the object.

A MTL file has the following form:

```
newmtl MAT_AAE6B4
Ka 0.117647 0.117647 0.117647
Kd 0.752941 0.752941 0.752941
Ks 0.752941 0.752941 0.752941
illum 1.91371e-305
Ns 8
newmtl MAT_ACDEAC
Ka 0.0823529 0.0823529 0.0823529
Kd 0.823529 0.823529 0.823529
Ks 0.0823529 0.0823529 0.0823529
illum 1.91371e-305 Ns 8
map_Kd brick.png
```

Ka, Kd and Ks specify the ambient, diffuse and specular material for that object. map_Kd allows a texture file to be specified.

#### 3.8.5 TGA + PNG

TGA and PNG are open image formats. Both are reasonably complex formats. All PNG and TGA processing is performed by the OpenGL for Java libraries, so it is not a requirement of this project to understand and re-document the image formats. Please see Bibliography for links to specification information.

### 3.9 Use of Tools, Libraries and Existing code.

#### 3.9.1 Java Profiler

A Java profiler was used to examine the running of the applet, and enabled the detection of bottle necks and inefficiencies in the source code. A function to calculate the height of each point on the landscape tile was being called many thousands of times a second. To fix this the landscape heights were precalculated and stored in an array. The internal representation of 3D models were stored as a string containing a list of commands to be carried out to create the object. This was very extensible but led to many thousands of string comparisons every second. This was remedied by storing the model format in arrays, to hold verticies, normals materials, texture indexes and a final faces array that points to 3 indexes in the vertex, normal and material arrays.

#### 3.9.2 Java Integrated Development Environments (IDEs)

Java was used as the programming language for the server and the client. The IDE used was initially Microsoft Visual J++ 6.0 but was then switched to IBM's Eclipse Platform as it allowed simultaneous debugging of the server and client, plugins (such as a profiler) and had a more understandable multi thread debugging interface.

#### 3.9.3 Reimplementation of functions

Some functions (such as String.replaceAll) were not supported on the virtual machine installed with Internet Explorer. These functions were recreated, so a new version of the VM would not be required.

#### 3.9.4 Third party Libraries.

**OpenGL for Java** OpenGL for Java is a third party library created by Jausoft. This library a creates a wrapping of the underlying system's OpenGL capabilities, and presents a Java interface which is used to render the environment.

**TCBG's Swing.Jar** TCBG [4] have created a swing.jar file for use with their BioCoRE application. This

[4] The Theoretical and Computational Biophysics Group at the University of Illinois at Urbana-Champaign

jar file was used to allow the use of Swing components in the applet without requiring an updated version of the VM. This is very similar to the one from Sun, with the advantage that the TCBG jar file could be linked to as a class archive from the applet, and did not need to be copied to the computer's local drive.

# 4 Testing and Quality

Testing was performed in three stages.

- Module testing - This project was developed using the evolutionary prototyping method. After each evolution the new features and classes were tested thoroughly. The functions and procedures of the classes were investigated to try to find errors. This was performed by creating a set of test harnesses for each of the classes. Some programs external to the project were used to test the operation of sockets in networking classes.
- Speed and Load testing - Although the project does not need to handle a extreme number of users, it was interesting to discover the number of users that could connect before the system became unusable.
- User testing - An important part of any testing, is how the end user interacts with the software and any changes they wish made to the software. At the end of development, a number of users tried the software, their familiarity with computers was varied. Feedback was be requested from all participants. High level features of the program such as multi-user support are also discussed.

## 4.1 Module testing

Due to the tree like structure of dependencies in the Applet[5] and Server[6], module testing was made simpler as the classes that had no dependencies, such as the ProtocolHelper class, could be tested individually. In the event of an incorrect result the source code of the class was examined to find the source of the error and edited to fix it.

Once these classes were shown to be correct the classes one step up the tree could be tested, because their dependencies were shown to be correct, and thus all errors that occurred would have occurred in the class being tested.

An important dependency to remember is that of the client/server system, some functions output involves sending data out, or acting on data that has been received. As well as creating test harnesses for class testing, telnet was used to test some features of the server, by creating sample protocol lines for it to process. A simple Java server was constructed to test the socket output of the client's CmdThread and ObjDownloadThread.

[5] See Figure 7 in the pictorial appendix
[6] See Figure 8 in the pictorial appendix

The applet has been tested for data that would be expected to be sent. For computer to computer communication, the test inputs included extreme and unusual but valid data, for example the system time was never negative and protocol strings were correctly formatted. However, for User Interface functions, the testing was much broader and included inputs that might be seen as malicious.

Although this is good method for ensuring that the program is bug free, it does not cover some important issues, such as how the server will cope under high load due to many users.

Module testing was an iterative process, as some classes presented errors when given unusual data, and so were fixed and retested.

## 4.2 Speed:

It is was important that the applet did not run too slowly with many items in view. Although there was no user noticeable slow down, the author re-implemented the rendering routine to an array based system increase the frame refresh rate. The applet has been tested on a 400MHz machine with 20 objects on the screen, and ran smoothly. Highly detailed objects (over 500KB) will not be downloaded.

## 4.3 Load testing

Setting up the server and simultaneously connecting it from 12 computers did not cause any noticeable slowdown, and the server was not under high load. Running many applets on the same computer did cause the applets to become slow, but the applets perform much more processing than the server and it not expected that more than one applet will be simultaneously run on the same computer.

The preference for performing processing on the client is so that the server load is as light as possible. This allows a much higher number of users before the server becomes unresponsive. Data transfer is small because the client is sent location data, and does all the processing itself.

## 4.4 User Testing

User Interface testing is part of module testing, separately however it is important also to see how users adapt to the User Interface, changing the design to make the application easier to use may be required. Module testing alone does not test ease of use.

### 4.4.1 Nickname selection:

When a user first loads the applet they are presented with "/nick" written in the input box. Typing their chosen nickname after this and pressing enter sends

a request to the server to set their nickname to this new value. A variety of nicknames were tested:

| Chosen nickname | Result | Correct result? |
|---|---|---|
| No text | Refused | Yes |
| Three spaces | Refused | Yes |
| "Jonathan" | Valid | Yes |
| "A B C D E" | Valid | Yes |
| "Sakura" written in Japanese Hiragana | Valid | Yes |
| "Zhongwen" written in Chinese Hanzi | Valid | Yes |
| 50 character nickname in Roman characters | Valid | Yes, not ideal. |
| Nickname with Katakana and Roman characters | Valid | Yes |

**Result Meanings:**

- Refused - The nickname input was invalid. The new nickname is not used, and the user is sent the message:

```
* You are currently a Guest. Sign in to chat.
* To sign in type "/nick <your-nickname>"
```

- Valid - The nickname is valid and is set as the user's nickname. They are sent this message: (Followed by the new nickname)

```
* Your nickname is now:
```

Currently multiple users may choose the same nickname, which could cause confusion. A dummy function is used to check for duplicates, but could be easily extended. This is shown below.

```
boolean nickInUse(String nick) {
  return false;
}
```

If this function returns TRUE, then an "0" is added to the end of the nickname. In the event of a collision this number is incremented until a unused nickname is found.

### 4.4.2 Multi user

Multiple users can exist in the same environment at the same time, position and rotation of each user in the scene is reflected in other client's displays by the movement of an avatar. In the current version of the program other users are shown as rabbits, mirroring their user's location. Users can type messages to each other. Users are identified by a nickname which they choose when they first load the applet. Objects that have been placed and saved in the environment by a user, can be seen by all other users.

### 4.4.3 Multi Platform + Multi Language

The program was primarily developed and tested on the Windows operating system with Internet Explorer, as this is the most common combination of software that the applet will be used with. The applet has also been tested with Linux .

The client is written in Java and so may run on any compatible virtual machine. The use of the GL4Java native classes restricts the platforms marginally, but GL4Java will run on Win32, Macintosh, Linux, Unix and Solaris systems. The disadvantage of not not being to able to run the client on less widely used operating systems such as BeOS and HP-UX is outweighed by the convenience and speed of OpenGL.

The server runs identically on Windows and Linux systems, as would be expected on any system with a Java virtual machine. The applet was tested in a selection of popular browsers; Mozilla Firebird/Firefox, Internet Explorer, Opera and Netscape.

As mentioned before, when connected to the server, users may send text to each other. This text may use any Unicode characters. Unicode text input has been tested on Win32 with Microsoft's Japanese Input Method Editor (MS-IME2000). This makes it possible to send Japanese text containing Hiragana, Katakana or Kanji from one user to another. Other languages that require Unicode, such as Korean, Hindi or Arabic can be input in the standard operating system way (Either with a different keyboard layout or with an IME). Linux Unicode input can be achieved through kinput2. As shown above Unicode nicknames are also supported.

Windows will correctly display Unicode text if a compatible font is available. Linux correctly displays Unicode text provided that the "Dialog font" for Java is a Unicode font, this is easy to set and should be enabled as default. If no Unicode fonts exist on a system, then Unicode text cannot be displayed. However, Unicode fonts have been shipped with Windows since Windows 98, and also come with recent versions of Office. Other Unicode fonts such as "Bitstream Cyberbit" are also available, and can be used on Linux and Windows. "Arial Unicode MS" and "MS Gothic" are both fonts created by Microsoft which support different "planes" of Unicode characters. "MS Gothic" is a fixed width font, and is used primarily because it is simpler to do line wrapping.

### 4.4.4 Object buttons testing

Three buttons control the addition of objects into the environment, "Create Object", "Change Object" and "Save Object".

- Create Object - Makes a signpost a fixed distance in front of the user. This shows where the new object will be placed. The object is drawn a fixed distance in front of the user, and thus can be moved to the desired location.

- Change Object - Changes the object's model file. (To a tree or house, for example)
- Save Object - Saves the object to the server, and attaches it to fixed place on the ground.

The standard order is to click the "Create" button, choosing a new location, clicking the "Change" button to choose a new object, and then clicking save after making a final decision. The buttons can be clicked in any order, doing as one would expect. Clicking save when there is no object on the screen does not change the environment. You may walk around after choosing a new object, or you may skip choosing an object and just save a signpost to the environment.

#### 4.4.5 User interface:

It was important that the interface of the application was easy to use for many types of users. Careful consideration was taken when creating the user interface, to ensure that it was intuitive, and did not rely too heavily on English for actions to be carried out. Although the text of the buttons is in English, the text is simple and the user will work from left to right when adding a world object. The users withe English not a first language, seemed to use the interface without difficulty. Also the user manual can also be translated into other languages.

### 4.5 User Acceptance Testing

To ensure quality of the final product, the applet was made accessible though the authors website, and some users were asked for their opinions on it.

Concerns that were brought up were as follows:

**"The landscape seems to change colour if you turn quickly"** This was because the land position was updated before the sun was moved, and so the sun took one frame to catch up to the correct position. The rendering order was changed and this problem was solved.

**"There are only trees close to the start point"** To make the environment more interesting, the initial world is given 40 pine trees, two houses and a signpost in fixed locations. More trees can be added by users.

**"There's nothing on the signpost"** The signposts that can be added to the scene are only used for decoration. Though using them to give information to the user seems a good idea, this feature shall be added later.

**"The radar is not clear"** This remark was aimed at the grey relief map drawn under the radar. Radar was changed to be transparent and only show coloured points at objects in the scene.

**"The points on the radar are too small"** The points were doubled in size, given more contrasting colours and a command was added to allow the user to manually set the point size.

**Feedback** Many users commented on the stars during the night, and one user liked the environment because it was "Less linear".

When asked if people would just play in the 3D environment and not use the chat, one user said "It's most amusing with chat, you can make up scenes with friends". One student remarked that it was "annoying", that the multi player games that he played forced him to write "in English letters", referring to the act of transliterating the pronunciation of foreign phrases into roman characters.

The environment has been running online for several months, only being reset when the server is changed to a new version. Despite no online documentation, the users have picked up the interface quickly and the online environment now holds many areas of interest, such as a dinosaur farm, a long tunnel and a small tank battle.

## 4.6 User Interface Feedback

Due to its simplicity, the user interface was quickly understood and used without difficulty. The comments made on the radar were the only concerns. Commands to change the radar point size and to import model files not in the quick object list, are accessed though commands in the chat window. There are three buttons that allow the user to create, select and save objects into the scene. The user can walk around the scene to choose the desired position and rotation of the new object. This procedure has strong visual feedback, and users had no difficulty creating complex scenes with multiple objects. After selecting a wrong object to place in the scene, one user quickly remedied their actions without help by clicking again on the object selection button and continued building his scene. Thus showing the intuitive nature of the interface.

The applet was tested by a variety of users chosen for their native language and varying familiarity with computer software. The native language of participants included Korean, Japanese and English. The ages ranged from a year 6 primary school student, to adult users.

Inputting text in non-roman characters (Japanese for example) is possible by using the standard input methods (IME for Windows, kinput2 for Linux). No new techniques need to be learned.

## 4.7 Problems found during testing:

The radar was brought up as topic of concern. The original design of the radar was a grey level relief of the surrounding landscape, showing the peaks and valleys in the local area. On user commented that is was sometimes difficult to see clearly

An other user suggested that the points on the radar were made larger as to be clearer to the user.

The radar was modified to remove the grey relief map, use more contrasting colours, and the points were made larger. A command was added so that user could choose the size of the radar points.

# 5 Summary, Costings and Critique

## 5.1 Source code length

At the start of this project the size of the final program was estimated to be 6500 lines of code, spread throughout all the programs created.

**The final program had the following lines of code:**

- Applet: 3711
- Server: 682
- Total: 4393

The applet is significantly larger in size than the server as all landscape generation, environment rendering and model parsing is performed by the applet. The server manages object locations and forwards on chat messages to the clients. This preference for performing processing on the client is so that the server is under a much lighter load when a large number of users are in the world.

One of the main reasons for this discrepancy in sizes, was the author's inexperience in larger programs of this type, OpenGL had been used in small programs of 200 to 300 lines, but it was not known how this would map to more graphically demanding projects such as this one. Also, a few features that were envisaged at the outset, were not chosen for inclusion into the final program. These included complex features such as collision detection and a large collection of 3d model file parsers to allow many different types of models to be imported.

## 5.2 Extent to which the aims of the project been fulfilled

Primary objectives of the project:

- Multi-user 3D environment

  This aim has been fulfilled. It is possible to connect multiple clients to the same server, position and rotation of one user around the scene is reflected in other client's displays by the movement of an avatar. Users can request a "nickname" to be used within the environment and then may send text to each other.

- Allow the client program to be multi-platform

  This aim has been fulfilled. The client is written in Java and so may run on any compatible virtual machine. The use of the GL4Java native classes restricts the platforms marginally, but GL4Java will run on Win32, Macintosh, Linux, Unix and Solaris systems. The disadvantage of not not being to able to run the client on less widely used operating systems such as BeOS and HP-UX is outweighed by the convenience and speed of OpenGL.

- Enable communication with the full Unicode character set

  This aim has been fulfilled. As mentioned before, when connected to the server, users may send text to each other. This text may use any Unicode characters. Unicode text input has been tested on Win32 with Microsoft's Japanese Input System (MS-IME2000). This makes it possible to send correct Japanese text containing Hiragana, Katakana or Kanji from one user to another. Other languages that require Unicode, such as Korean, Mandarin and Cantonese will also be input in a similar way. Unicode nicknames are also supported.

Secondary objectives of the project:

- User customisation

  This aim has been fulfilled. Users are able to add models to the world and create scenes with standard objects or objects of their own design. Models can be arbitrarily placed anywhere in the landscape, and once saved, are shown to all other users. A system is in place to ensure that clients are only sent information about instances that are relevant to them, and so the server can hold many millions of instances when the client only needs to render a very small number in a particular area.

- Improve realism of world.

  This aim has been fulfilled. Fixed sky lighting has been added to the world, and object surfaces are lit correctly by using face normals. All face colours in RGBA form have been converted to materials.

- Web based.

  This aim has been fulfilled. The client has been designed to run as a web based applet or as a standalone application. The problems caused by Internet Explorer's VM have been solved, and the applet has been tested in many browsers.

## 5.3 Problems and difficulties

**Allowing Unicode support within Internet Explorer without requiring an upgrade of the standard VM.** Java has two main windowing methods, AWT and Swing, Swing was built to replace AWT.

As far the as the author can see, it is impossible use a Unicode font in AWT without editing files on each computer that uses the client. Currently the default Java fonts are not Unicode compliant. Swing is needed to use a Unicode font. Internet Explorer uses version 1.1 of the Java VM, and thus does not have Swing.

This was solved by using a 3rd party swing implementation (BioCOrE), which was linked to by the applet.

**Java Security features.** Java's security model caused a variety of problems for the development of this application.

- Applets cannot connect to a server when they load. This was worked around by creating a Thread that dealt with all commands sent to and from the server. Strangely, this did not have the same restriction, even when the thread was created when the applet was started.
- An applet can only connect to the server of the computer that it was downloaded from. This means that if the user wishes to download a 3D model file from an arbitrary place on the web, they cannot. This restriction was solved by building a simple proxy into the server which allow the client to request files, through the server, from anywhere on the Internet.

## 5.4 Main accomplishments to date

- Creating a modular framework so that different converter classes can be easily added so the functionality of the applet can be extended. Created a variety of converters to import all kinds of file types into the environment.
- Creating a download queueing system that only downloads one file at a time, but can understand dependencies required needed for the creation of there objects and can reorganise the queue accordingly.
- Creating a landscape generation algorithm using a seed-based system. Integrating this with the gradient colouration and blending algorithms.
- Allowed the project to support Unicode on many platforms, Ability to send and receive chat messages written in any language.
- Allowing many users to inhabit the same scene. Users can see and talk to one another.
- Creating a variety of environment effects to improve the realism of the scene.
- Infinite landscape generation. Clients will generate the same landscape without needing to send landscape data from the server.
- Allowing the user to change their environment with the import of 3D models.

## 5.5 Main conclusions and findings of project

### 5.5.1 User interest in virtual environments

Having placed the applet on the web, I found the user uptake and involvement to be much higher than I expected. It seems that multiple user environments are very popular, and the 3D graphics or Unicode support has helped to draw users to the project.

### 5.5.2 How I would modify my methods if doing this project again.

I would have added land and item ownership at an early stage. To "tack on" a land or item ownership system at this stage of development, would require a lot of programming. I would have added object and land ownership from the early stages, so that this would be integrated into the whole and the code would remain neat.

I would have created a wide selection of converters for many different types, so that users could import a much wider range of objects.

However, overall, the author is very pleased with the development of the application, and was helped by the class structure, designed before the implementation was started, as this helped to bind the project together.

## 5.6 Further work

This project is ongoing, and due to the surge in public interest the author has decided to continue the project. Possible additions to the project are as follows. Not all of them may be implemented, but suggest what could be done.

- Create a much more developed item infrastructure. Allow users to have an "inventory" in which items can be stored.
- Allow goods to be created, bought and sold. Add wealth in the form of a currency or purely by a barter system.
- Users may be allowed to modify the terrain in certain areas, to further customise the world. Users could add a mountain range close to their house or create a valley to enhance the appearance of their area.
- Create an ownership system of land. A problem with the current system is that users can place inappropriate objects in an area that another user has spent a lot of time customising. Allowing a particular user to "own" a portion of land and only have rights to put items in that area, would protect it from this kind of vandalism.
- Allow signposts to inform the users of information. Include click-able items with context sensitive actions, clicking could take an item, transport you to a new area, read a signpost or load the web page of the landowner.

- To add to the realism of the world, better atmospheric effects will be added. These will include items such as snow, mist, fog and reflection effects.
- Allow the user to change their avatar, this could be based on wealth to give users an incentive to trade. Users may enjoy the opportunity to become an Alien or a Velociraptor!
- The author plans to create a VRML converter, as this is very commonly used format and would greatly improve the scope of the program. VRML supports texturing, materials, animation, prototyping, text and a few other features.
- The ability not to walk through apparently solid objects would add to the realism of the scene. Collision detection and object obstruction will be implemented. There are a variety of different general purpose collision detection algorithms that exist, where the most accurate are quite computationally expensive. The author has devised an algorithm that takes advantage of knowledge about this game and creates a collection of bounding boxes around elements of each object, the advantage of this algorithm is that is can be run quickly and is not as crude as simply bounding the object in X,Y and Z planes.

# 6 Bibliography

**VRML97 Specification, ISO/IEC 14772-1:1997**

- Complete VRML Specification
- http://www.web3d.org/Specifications/VRML97/

**The Annotated VRML97 Reference Manual**

- A comprehensive VRML description, useful as quick reference.
- http://www.web3d.org/resources/vrml_ref_manual/Book.html

**Virtual Worlds as Architectural Space An exploration By Veronica Zidarich**

- An essay on 3D online worlds, community building and virtual design.
- www.foundation-langlois.org/e/activites/zidarich/zidarich.pdf

**Active Worlds**

- Large online community. Features 3D worlds which users can participate in and add to.
- http://www.activeworlds.com/

**VRML Worlds Gallery**

- Similar to VRML Homesteads, a collection of VRML worlds that can be explored.
- http://www.avatara.com/vrml/

**Kerin Spaink**

- Fascinating Essay on Net Art and Virtual Homesteads
- http://www.spaink.net/english/ArsPrix97.html

**BioCoRE Setting up your browser to work with Swing**

- 3rd-party Swing implementation used to allow Swing components on Internet Explorer's Java VM
- http://www.ks.uiuc.edu/Research/biocore/swing_setup.shtml

**Extensible Java Profiler**

- A profiler for Java - Created as a plugin for the Eclipse IDE
- http://ejp.sourceforge.net/

**DAAP: The first world of virtual sculpture on line**

- A 3D virtual environment where undergraduate artists display their project work.
- http://www.cerhas.uc.edu/dwoodham/daap2.htm

**PNG (Portable Network Graphics) Specification**

- Specification of PNG image files
- http://www.w3.org/TR/REC-png-multi.html

**TGA (TARGA) Specification**

- Specification of TGA image files
- http://astronomy.swin.edu.au/~pbourke/dataformats/tga/

**The Programmer's File Format Collection**

- Collection of specifications for many file formats
- http://www.wotsit.org/

**Mono Project**

- Open Source implementation of .NET Development Framework
- http://www.go-mono.com/