GESystem: A MULTI-USER 3D VIRTUAL ENVIRONMENT

Jesse M. Chandler

Problem Report submitted to the
Lane College of Computer Science and Electrical Engineering
West Virginia University
In partial fulfillment of the requirements
For the degree of

Master of Science
In
Computer Science

Frances L. Van Scoy, Ph.D., Chair
Ramana Reddy, Ph.D.
Don McLaughlin, MSCS

Department of Computer Science

Morgantown, West Virginia
2006

Keywords: OpenGL, Virtual Environments

ABSTRACT

Multi-User 3D Virtual Environment

Jesse M. Chandler

Many people have the desire to be able to visit new locations. Unfortunately, constraints of time, limitation of monetary resources, and other problems can arise when planning travel. Therefore, it becomes interesting to computer science to simulate different environments. This program was designed to allow multiple users to experience the same real-time virtual environment via a network connection. In addition to being able to view the same environment, users are also able to view other users within the context of the environment.

Introduction

This project has its origins in computer games. Computer games utilize many concepts that have become interesting areas of study. For example, The West Virginia Virtual Environments Lab has previously used the Torque game engine to create training game to simulate the first response team for the release of biological weapons at a sporting event. Video games are useful for such studies largely due to the ability of games to accurately capture and simulate events and/or environments.

The basic concept of this project is a simulation of a person (or group of people) on a given terrain. It allows multiple users to experience a virtual environment in real-time. Users running the client application connect to a central server which keeps track of other users and their location and orientation within the world. All clients send updated position and heading data to the central server, which then sends the updated data to the other clients

It has applications in other problem areas. For example, such a system could be used to give an accurate representation of State Parks to promote tourism. The system could also be used by landscape architects to model a piece of terrain so that changes to the landscape may be experimented with in a virtual environment before any actual alterations begin. Additionally, the system could be used for forensic applications such as crime scene reconstruction.

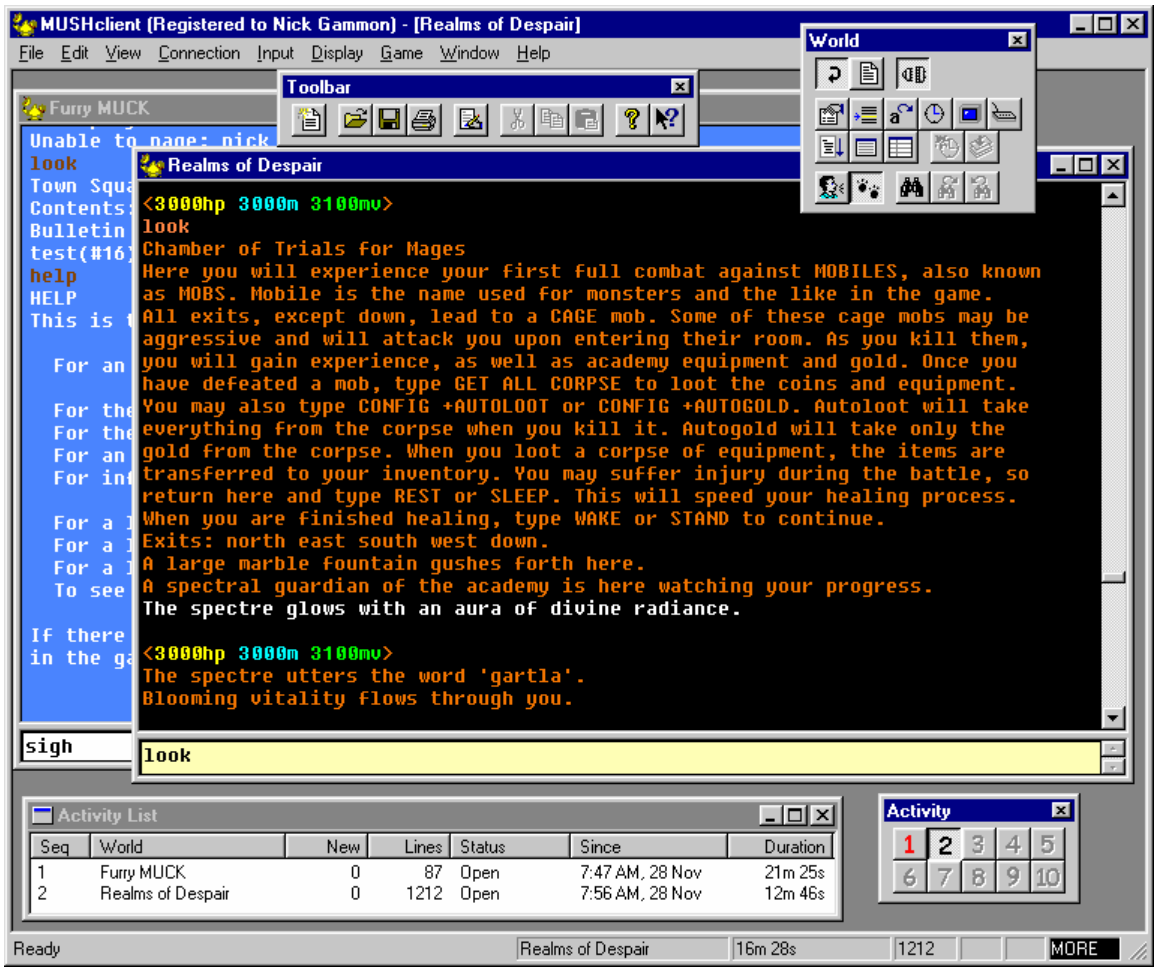
Such systems have been employed in various types of applications. First and foremost (in addition to being the main inspiration for this project) are multiplayer online games.

Brief History of Multiplayer Online Games

MUD

One of the earliest forms of the multiplayer online game is the MUD. MUD stands for Multi-User Dungeon. A MUD is a text-based game in the genre of Dungeons and Dragons. It allows users to create characters based on attributes such as class (warrior, magician, thief, etc) and race (human, elf, dwarf, etc). Each character then has a set of attributes associated with it, often based largely upon the second edition of Dungeons and Dragons.

To play, users telnet to a server and supply login information. Players are then placed into the virtual world where they may interact with other players and "fight" "monsters" to gain experience. As previously stated, MUDs are entirely text-based. In most varieties of MUDs, the player is presented with a room description detailing what the area creator had envisioned, followed by a list of objects, players, and monsters which may be in the same "room." Players send commands such as "get torch" or "kill homunculus" to the server, and the server performs necessary calculations or events to take the desired course of action.



An example of a SMAUG codebase MUD

In order for the MUD to be effective, it must be very detailed in its world description. This gives players a reference point to build a mental image of what the room might look like, as one would do with a novel. Because of this imaginative component, a MUD can provide a very powerful and extremely personal experience for individual players, in addition to being a collective experience shared between groups of players who interact with each other. Unfortunately, this strength can also be a weakness, as it requires a large amount of reading on the player's part. This may cause the player to skip over some of

the details in the room description due to lack of interest, or give up on the game altogether if the player does not posses the imaginative skills necessary to cultivate such a mental image of the simulated world.

While this may seem very low-tech, it is important to note that many MUDs are still in existence today. Additionally, variants of MUDs have been used for other applications. Until recently, the Access Grid project, a system that allows real-time two-way group video conferencing, used a type of MUD as a venue service for other clients to connect and pass messages.

MMORPG

MMORPG stands for Massively Multiplayer Online Role-Playing Game. It can be argued that the MMO is the next generation of the MUD. Similar to the MUD, players connect to a central server, and upon login, are placed into the virtual world where they may interact with other players, fight monsters, and perform other actions within the world. Unlike the MUD, however, MMOs contain a graphical display of the world and other players that exist within it.

A screenshot from Blizzard Interactive's World of Warcraft

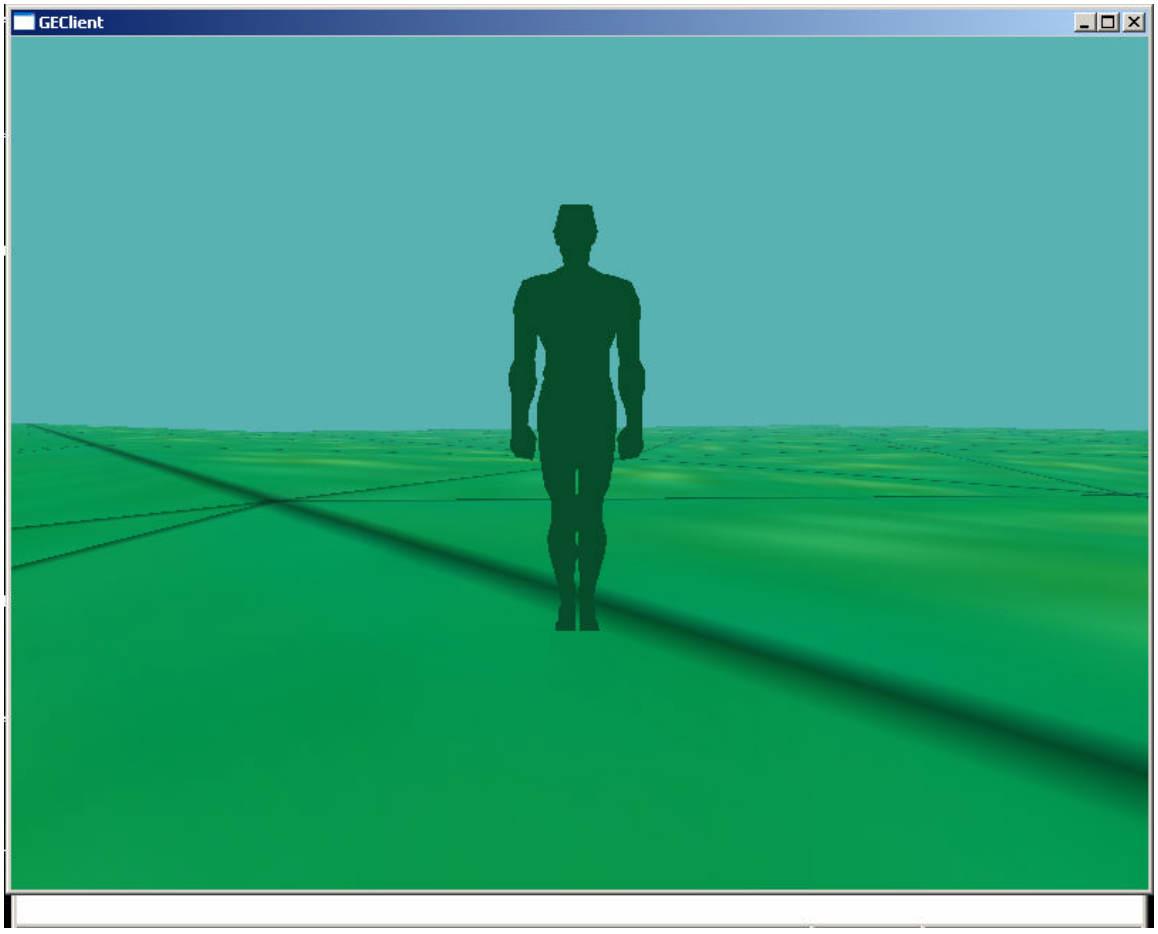
Games of this sort, in some form or another, have existed since around 1985, but did not become popular for another ten years. In the late 1990s, Ultima Online and Everquest gained popularity with many gamers not familiar with the genre, and arguably birthed the present day popularity of the MMO. Today, there are many very popular MMO games. Just to name a few, there are:

- City of Heroes (http://www.cityofheroes.com)
- World of Warcraft (http://www.worldofwarcraft.com)
- Star Wars Galaxies (http://starwarsgalaxies.station.sony.com)
- The Matrix Online (http://thematrixonline.station.sony.com)

All of these game services require the player to purchase the software initially. From there, players then pay additional monthly fees to continue playing. In addition to providing a profit margin for game corporations, these monthly fees are also used to maintain and expand game servers, pay for development of additional in-game content, and pay for development of bug fixes and pay game administrators.

GESystem

This report describes the design and implementation of GESystem (Graphical Environment System). This program incorporates some key elements of the previous types. GESystem is a software package that allows multiple users to simultaneously experience a virtual environment in real-time. It is made up of two components: the GEClient and the GEServer. GEClient is the end-user application which allows the user to connect to a GEServer, load the environment visualization, and view other players who exist within the virtual world. GEServer is the server-side application which manages all user connections and sends out all relevant data to its client connections.



Screenshot of the GEClient

Implementation

The GESystem Multi-User 3d Virtual Environment program client and server applications were written using Microsoft Visual C++ 6.0. Due to the nature of networking components of the project, it was necessary to use multi-threading for this project so as to not interrupt the graphics rendering of the visualization components. Therefore, the native Windows programming API was chosen so that multi-threading and socket connections could be handled natively by the operating system.

OpenGL

This program uses the OpenGL library for rendering the graphic components of the world. OpenGL is an interface to graphics hardware. It is platform-independent, but it does not offer any support for windowing or input. OpenGL only provides functionality to render objects constructed from graphics primitives, such as points, lines, and polygonal figures. OpenGL provides all the necessary operations for arranging objects in a 3d space and calculating the associated information to screen pixels, but little else. It is this simplicity that allows it to be both operating system and hardware independent, and therefore able to run on almost any computing platform (OpenGL Programming Guide, Ch1).

Windows API

Windows API (WinAPI) is the native programming environment for the Microsoft Windows Operating system. It allows programmers a direct interface to certain functionality of Windows. It is through the WinAPI that the input given to the operating

system is handled and the data generated by the OpenGL code may be displayed on the screen.

The Windows API provides a number of services divided into six main categories:

1. Base Services – provides services such as device input and output, error handling, event logging, and processes and threads (as well as numerous others),
2. Common Control Library – is a collection of DLL files that are commonly associated with standard Windows interfaces and commands which give the "look and feel" of a Windows application.
3. Graphics Device Interface – allows an application to create graphical output for devices such as a display or a printer. An application creates a device context and uses options from a set of attributes with it. Attributes include such tools as bitmaps, brushes, clipping, colors, coordinate spaces, and many others.
4. Network Services – allows communication between applications over a network connection. The two network interfaces offered are Network Management, which handles user accounts and network resources, and Windows Networking, which allows for the actual implementation of networking in applications.
5. User Interface – supports defining appearance and behavior. Includes controls for familiar interface options such as, buttons, combo boxes, cursors, dialog boxes, and many others.
6. Windows Shell – supports creating Windows shell extensions. Provides Namespace, a collection of objects of interest such as files and other resources,

and shortcuts, which allow access to other objects located anywhere within the shell's namespace.

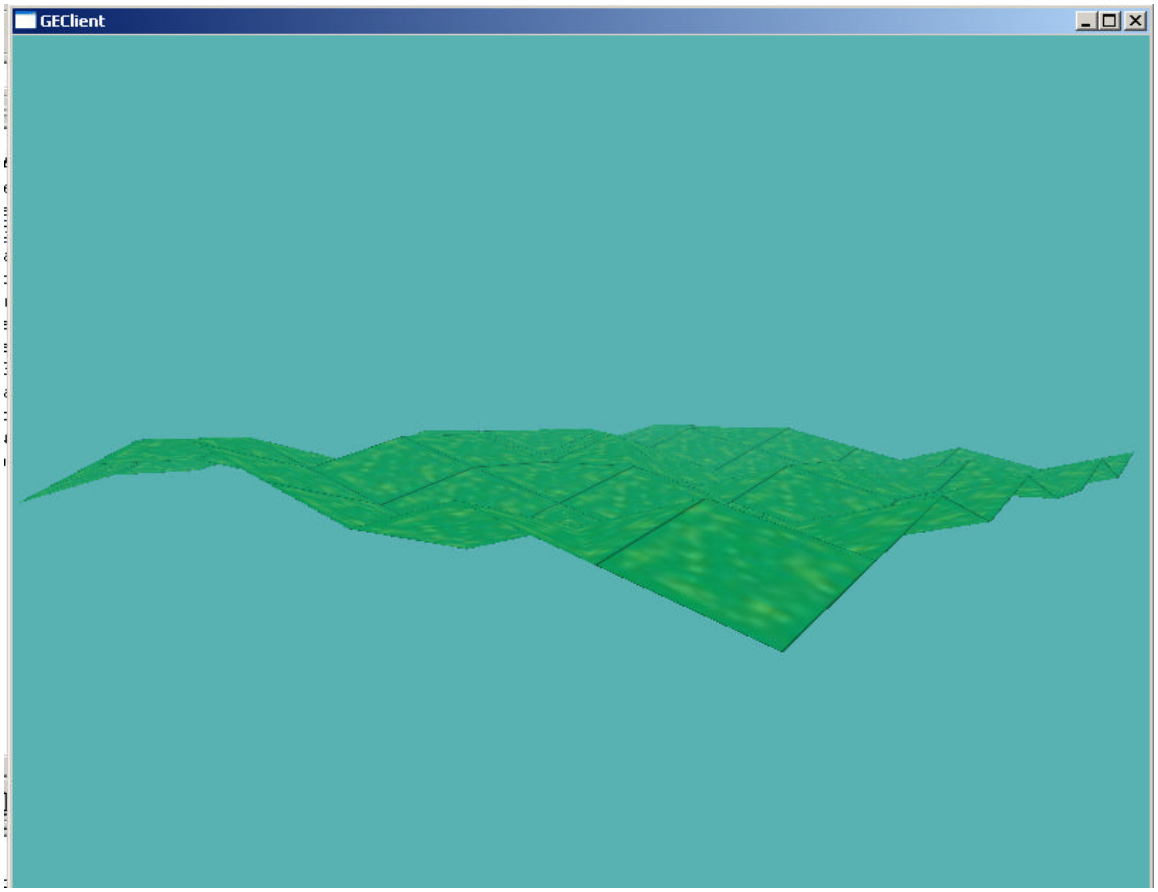
For this application, the main structure of the interface with Windows is similar to many other graphics windowing environments such as GLUT. Initial setup calls are made to the operating system to create the window, specify the window attributes, and begin the message pump to distribute and handle any input or events that are necessary during the runtime of the program. Once the setup is complete, initial OpenGL setup calls are made and the program enters into a render loop where the geometry is repeatedly drawn and updated to reflect changes effected by the user. This loop is executed until the program closes.

Terrain Data

The actual terrain data is stored in a file that is loaded by the client application on startup. It is stored as a Wavefront OBJ file. The terrain was generated using Milkshape3d's Terrain Generator. The user enters a size for the terrain and a number to designate the amount of hills, and the Terrain Generator produces a randomized terrain based on these parameters. The end result is exported to OBJ format (more on Milkshape 3d and OBJ later).

Since the terrain will vary in height, the appropriate height of the user avatar must be determined based on the terrain data. The position of the user is always known for the X-Z plane, but the position on Y-axis is not. Due to the method used to store the terrain

data, this position is not directly accessible; therefore, the data structure must be searched to find the appropriate position in the world. A search method that is well suited to 3d terrain data must be used, as it would not be efficient to employ a linear search through the list of squares to determine the nearest points to the user. For example, given a grid of ten by ten points on the terrain, a total of one hundred data points exist. From these one hundred data points, eighty-one squares are generated. While it may be trivial to employ a linear search of a list of eighty-one elements to determine the closest data points, this example generates a fairly small area when rendered in the virtual environment.



an example 10 x 10 grid rendered

Therefore, it becomes necessary to implement a better method of determining the nearest terrain data points to the user. For this program, a quad-tree data structure is implemented in order to more effectively implement the search.

In this instance, the quad-tree is used to divide the geometry of the terrain into quadrants for searching. The entire map is divided into four quadrants, each with a range of X and Z values in which the squares contained in that quadrant must fall. This process is repeated recursively until the quadrants become actual squares, thus generating a tree structure that is searchable. Where a linear search algorithm is represented in O(n), the quad-tree structure yields a search that is completed in O[log4(n)], greatly reducing the search time of large datasets.

Once the appropriate square is found from the quad-tree, the user data is compared to the points on the square. The height for the user avatar is approximated by the average of the height at the user's coordinates as located on the slope of the line between the two data points on both an X-Y plane and a Y-Z plane. The formula for this can be given by:

$$(pYa + pYb)/2$$

$$pYa = M(pX) + B \text{ , } pYb = M(pZ) + B$$

Where pYa is the height of the user on the X-Y plane and pYb is the height of the user on the Y-Z plane.

Camera

Since this is a 3d environment, it is necessary to move the viewpoint about the world in order to create a more realistic experience for the user. OpenGL treats the viewpoint as a sort of "Camera." The programmer specifies a set of coordinates that designate the position of the camera, the point at which the camera looks toward, and the "up" vector.

For this program, a third-person view was chosen to allow the users to see the avatar that represents them in the virtual environment. This means that the camera must calculate its position and heading based on the position of the user, and not simply rest on the position and heading of the user.

The calculations for the camera are based on the polar coordinate system. The camera maintains a rotational angle (theta) which is separate from the rotation of the user avatar. This angle is used to calculate the camera's orbital rotation about the Y-axis of the of the user avatar. Additionally, it also uses the X and Z coordinates of the user avatar to calculate a heading and distance vector at which to focus the camera. Mathematically, these calculations are given by:

$$H = 30 * \cos(\text{theta}) + X$$

$$F = 30 * \sin(\text{theta}) + Z$$

$$Cx = 4 * \cos(\text{theta}-180) + X$$

$$Cz = 4 * \sin(\text{theta}-180) + Z$$

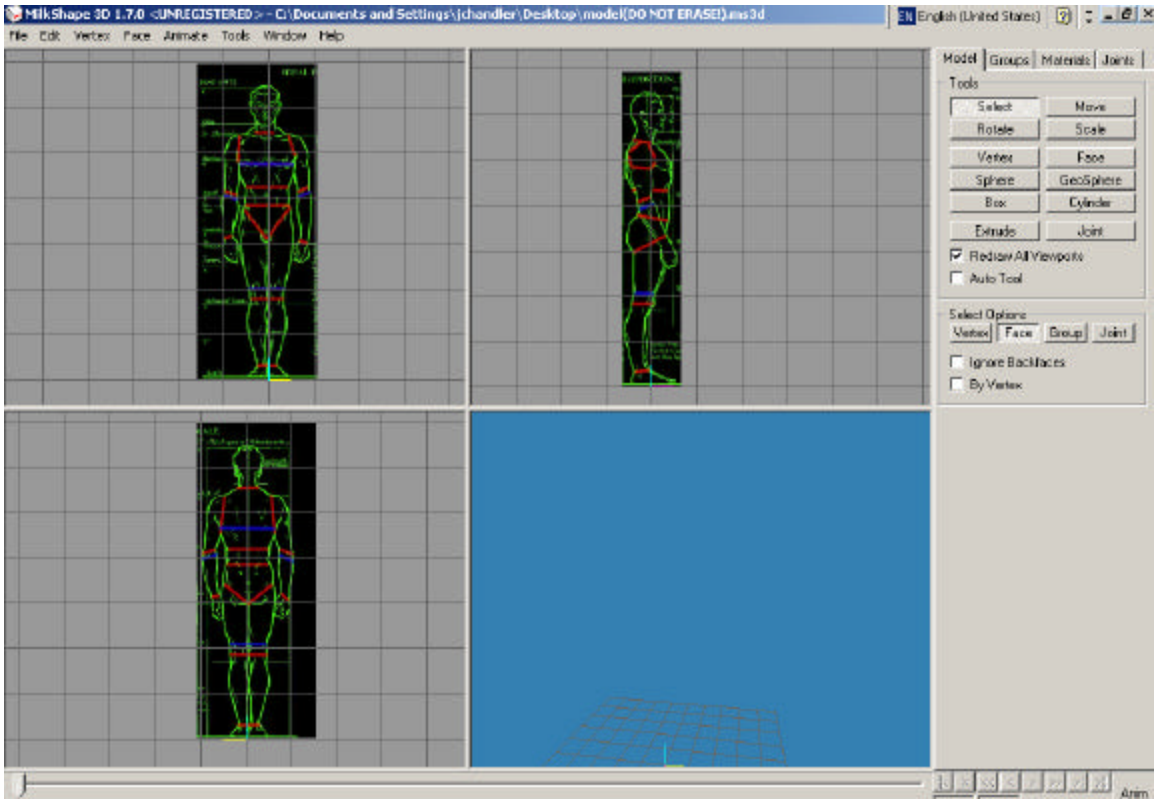
Where H is the horizontal component (or X-axis) of the camera's look-vector, F is the depth component (or Z-axis) of the camera's look-vector, X and Z are the horizontal and

depth components of the user avatar respectively, and Cx and Cz are the horizontal and depth components of the camera position. All constants in the equation are used to offset the camera from the user avatar to achieve a third-person view.

User Avatar

The user's representation in the virtual environment is achieved through a 3d model created in a modeling program. This model was created using Milkshape 3d (http://www.swissquake.ch/chumbalum-soft/ms3d/). Milkshape is a shareware modeling package that incorporates many features of professional level modeling applications, but presented in an easy-to-use format. It is typically designed to work with low-polygon models, which is ideal for this project.

In order to create a realistic-looking human figure for the user avatar, the avatar must be properly proportioned. A common technique for this is to base the model's dimensions on a picture of the object which is being modeled. In this instance, figure drawings of a male human from multiple angles were used as a reference point. This technique is called, "Photograph-Based Modeling."



Screenshot of Milkshape 3d and figure guidelines

From there, vertices are placed in a 3d space by the user. Using the image as a guideline, the user builds the model by hand. The vertices are moved to correspond to actual points on a figure. Once the vertices have been placed, triangle faces are placed by the user on the appropriate vertices.

OBJ File Format

Once the model is created, it is exported into the OBJ file format. The OBJ file format was originally designed for and used by Wavefront's Advanced Visualizer. Since then, it has become a standard file format for many 3d objects. Almost every major modeling package today, both commercial and non, offer support for OBJ files.

An object file format is composed of a data-type identifier, followed by data values offset with slash characters. A grammar for the format can be defined by:

S -> TS

S -> NULL

T -> # G

T -> v Float Float Float

T -> vn Float Float Float

T -> vt Float Float

T -> f D

G -> <string>

D -> Int Int Int

D -> N

D -> X

N -> Int/Int Int/Int Int/Int

X -> Int/Int/Int Int/Int/Int Int/Int/Int

The # production represents a comment and followed by text. The ”v” indicates that the command code is a single vertex, and the following F values are floating point X, Y, and Z coordinates. Each vertex is stored by an index, the first being 1. The “vn” indicates a vertex normal with the following coordinates giving the vector. Vertex normals, like vertices, are stored by index. The “vt” command is a texture coordinate, which is stored

in the same manner. The "f" command indicates a polygonal face. There are several different options for what data may follow. Three integers specify indices for the vertices that make up the three points of the triangular face. Two integers grouped together specify indices for vertex and normals. Finally, three integers grouped together specify indices for vertex, normal, and texture coordinates.

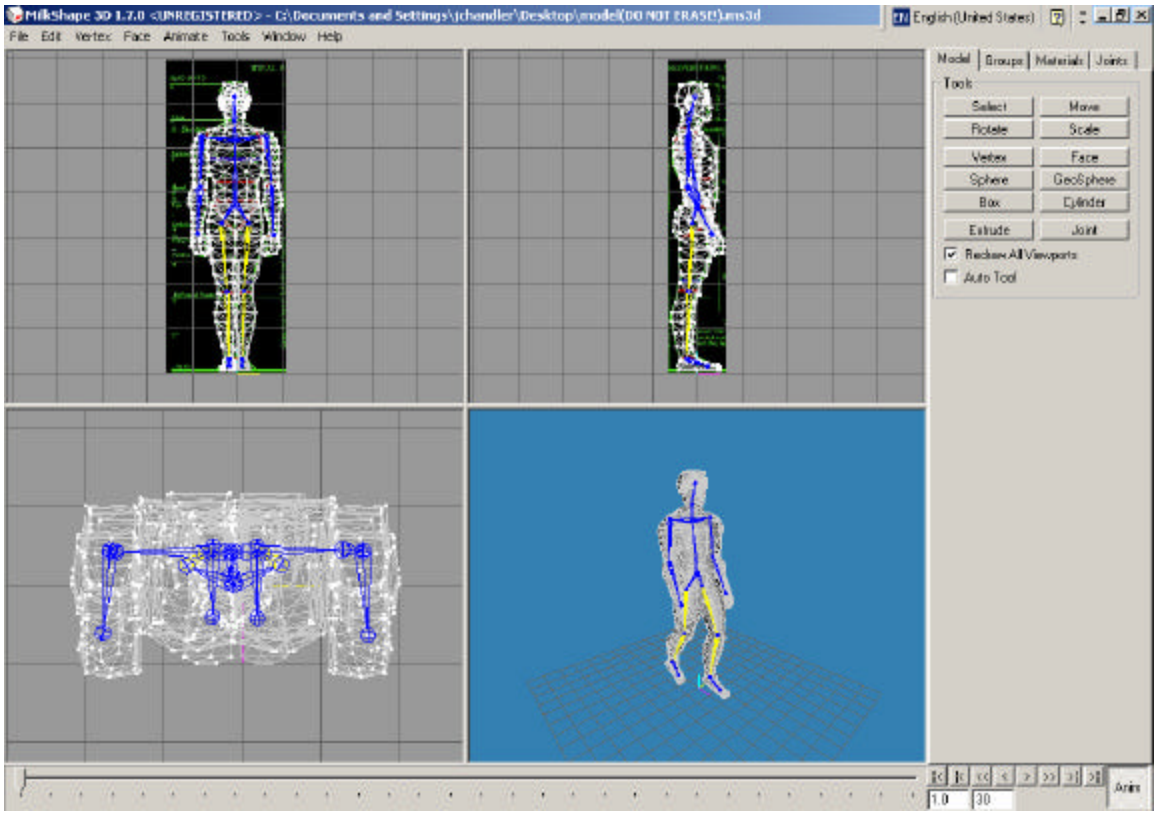
Keyframe Animation

In order to seem more realistic, the user avatar employs a walk animation. The walk animation is achieved through a process called keyframing. The concept of keyframing is similar to techniques used in video compression. In video compression, a single, uncompressed frame is used as an "anchor" point. Subsequent frames between anchor points are interpolated based on the keyframes.

Using the 3d modeling software, a skeleton is created within the model. A skeleton is composed of "joints," which allow for movement and rotation, and "bones," which connect two joints. Like the vertices that form the structure, joints are placed by hand on the 3d model by the uses. Milkshape automatically places a bone between two joints. Once the skeleton is in place, vertices are assigned to the joints. The joints then act as an origin of rotation for the vertices.

Using the skeleton, the model is moved into two positions along the walk cycle. These keyframes are saved and stored to serve as anchor points. To simulate a walk cycle, the

geometry of the model is interpolated over time to move from one keyframe to the next in sequence.



Screenshot of skeletal animation

The OBJ file format, however, does not support keyframe animation directly. Therefore, a workaround has been applied for this instance. The skeletal structure is still used to move the model's geometry, but each keyframe must be exported as a separate OBJ file. Since there are the same number of vertices and triangle faces in each frame, all data except for the new vertex locations may be ignored. From there, each "keyframe" is loaded into the system. A linear interpolation is used to transition smoothly from one "keyframe" to the next. This is achieved by multiplying each keyframe by a scale factor and adding the results. Or:

$$D(XYZ) = scale1 * keyframe1(XYZ) + scale2 * keyframe2(XYZ)$$

scale1 is initially 1.0 and decreases over time

scale2 is initially 0.0 and increases over time

Network – Client side

On the initial startup of the client application, the client sends a TCP connection request to the server on port 6666. If no connection can be established, the program terminates. Once a connection is established, the client application spawns a communication thread which is responsible for all communication to and from the server. This thread sends out the position data of the client's user avatar to the server and receives position data for the user avatars of other clients connected to the server.

Network – GEServer

When the GEServer starts up, it binds a TCP socket to port 6666. It then listens on that port for incoming connections. Upon receiving a connection, a new thread is spawned to handle communication with that client. As a new communication thread is spawned, the thread is assigned a unique ID, and a global list of clients is updated to reflect the new connection. This list of active connections contains the relevant position data for all clients. This data is constantly updated by clients and distributed to all the active connections.

User Interface

Because the user action is limited only to movement about a 3d space, the user interface for the GEClient is very simple. The user uses the arrow keys to move forward, backward, and turn left or right. It is common in many such applications to incorporate the mouse to control movement of the camera, or a combination of mouse and keyboard functions. For the context of this program, this was not incorporated so as to keep the user interface to a bare minimum, therefore ensuring simplicity.

Limitations/Weaknesses

GESystem has some known weakness, but the foremost of which is due to a problem that shall be called "communication misalignment." This problem occurs in GESystem very likely due to an implementation of Windows sockets, Microsoft Foundation Classes (MFC), or some sort of combination between the two. It occurs when successive communication statements fail to achieve the communication explicitly given.

To elaborate on this problem, it first becomes necessary to explain the basic principles of blocking socket communication. Blocking sockets are a standard interface for network communication. Communication occurs with the use of the send and recv function calls. When the sequential execution of statements reaches a send or recv command, execution stalls until has been successfully sent or received respectively. What this means is that for every send or receive in the client application, there must be a corresponding send or receive in the server application. For example:

Client Side	Server Side
send(data1);	recv(data1);
recv(data2);	send(data2);

This is a very simple and straightforward example. It also adequately illustrates the one-to-one relationship between necessary statements on both the client side; in order to maintain flow of the program and prevent deadlock, there must be a corresponding receive at the other end to account for every send. With that in mind, consider the following example:

```
Client Side:                        Server Side:
for(i=0; i<20; i++)                 for(i=0; i<20; i++)
{                                   {
    recv(data[i]);                      send(data[i]);
 }                                   }
```

In this case, the total number of sends and receives matches; unfortunately, this is where communication misalignment begins to occur. There are exactly 20 sends on the server side and exactly 20 receives on the client side. In theory, each statement should wait for the appropriate data communication to take place before moving on to the next statement, but this is not always the case. In most instances, the communication occurs with no problems for an example so trivial as this one. However, consider the modified code as follows:

```
Client Side:                        Server Side:
for(i=0; i<20; i++)                 for(i=0; i<20; i++)
{                                   {
                                        load data from file;
    recv(data[i]);                      send(data);
 }                                   }
```

Again, there are still the same number of sends on the server side as receives on the client side, but this situation will almost always cause communication misalignment. The exact cause of this problem is not known. Some example cases sometimes do not reach even the first send (even though the appropriate data has been read from the file and output to the screen for test purposes), and yet sometimes other example cases will perform a number of iterations before meeting with communication misalignment.

The cause of this problem is still unknown. Due to the definition of the send and recv functions, no other instructions may be executed until communication has occurred. As

most of the functionality of these procedures is abstracted from the programmer, it is difficult to determine exactly how this problem comes about. Several theories include: the implementation of Windows Sockets, the interaction between Windows Sockets and MFC, or possibly even the way in which various hardware instruction level parallelism techniques such as branch prediction and loop unrolling affect execution of code in such a way as to be invisible to both the operating system and the programmer. More research must be conducted on this subject before any conclusions can be reached.

To paraphrase Galileo's supposed last words, "There is no reason why this should occur; and yet it does."

Conclusions and Future Development

GESystem meets the major functionality goals defined. It allows a client application to load and display a virtual environment, connect to a central server, and display other users in the environment. However, at present, GESystem is very primitive. Due to performance issues, many display features, such as lighting, have not been implemented. Additionally, GESystem suffers from problems of scalability and potential network instability due to communication misalignment.

Optional future developments for GESystem are numerous. First and foremost, any instabilities due to communication misalignment in the system must be examined and corrected. Second, the issues of performance and scalability must be addressed. From there, future development will include added support for user/environment interaction, improved visualization elements, and the support for environmental objects to be displayed in addition to just terrain data (such as buildings and other structures).

The potential for expansion of user/environment interaction for GESystem is tremendous. As the system does not currently support any sort of interaction, it is not hindered by any existing functionality that must be used; unfortunately, it does not benefit from any existing functionality either.

One of the first additions to user interaction would most likely be a “chat” functionality that would allow users to communicate with one another in addition to a visual presence. In the event of creating realistic simulations, it could be implemented in such a way as to

prevent other users who are too far away from the speaking user to receive such messages, or perhaps other users who are a distance away are only made aware that a user is speaking, but not aware of the content of the message.

Another possible addition to interaction is interaction with the environment. This is a very broad and complex topic, and would depend on the desired functionality of the developer. Environment interaction could be very simple, such as static response to proximity like removing flowers from the map after a player has "walked on them," or very complex, such as a system of "triggers" that would require a user to perform a certain number of actions in sequence to achieve the desired result.

There are many improvements that could be made to enhance the realism of GESystem's visual component. First and foremost, the user avatar model has not been texture mapped. A realistic "skin" for a user could be created and applied to the model to give the appearance of an actual human, rather than a humanoid figure shaded in grey. A collection of "skins" could even be created and randomly assigned to different users within the system to give the appearance of a diverse crowd and allow users to distinguish between other users visually.

Additionally, there are many other visual effects that could be added to give GESystem a more realistic look. Lighting effects have not been implemented in the system. Lighting and shadowing effects would greatly enhance a user's immersion into the virtual world as their addition would add to the realism of the simulation.

Finally, support for additional objects in the virtual world would greatly help improve the overall realism of the simulated world. At present, the only objects supported in the system are terrain and players. Additional objects could be created and added to suit the needs of the setting to be simulated. For example, if a forest setting is desired, 3d models of trees, plants, and rocks could be created and placed in the world to more closely resemble the desired location. If a more urban setting is desired, model sets might include buildings, vehicles, and other such structures one would expect to find in such an area.

Source Code - Client

winmain.cpp

```
// winmain.cpp – this makes initialization calls to windows to set up our graphics window
/* this code adapted from Beginning OpenGL Game Programming
  by Dave Astle and Kevin Hawkins
  Code Modified by Jesse Chandler
*/

#define WIN32_LEAN_AND_MEAN
#define WIN32_EXTRA_LEAN

#include <windows.h>
#include <gl/gl.h>
#include <gl/glu.h>

#include "CGfxOpenGL.h"

bool exiting = false;
long windowWidth = 800;
long windowHeight = 600;
long windowBits = 32;
bool fullscreen = false;
HDC hDC;

CGfxOpenGL *g_glRender = NULL;

void SetupPixelFormat(HDC hDC)
{
  int pixelFormat;

  PIXELFORMATDESCRIPTOR pfd =
  {
    sizeof(PIXELFORMATDESCRIPTOR),  // size
      1,                    // version
      PFD_SUPPORT_OPENGL |      // OpenGL window
      PFD_DRAW_TO_WINDOW |      // render to window
      PFD_DOUBLEBUFFER,         // support double-buffering
      PFD_TYPE_RGBA,            // color type
      32,                   // prefered color depth
      0, 0, 0, 0, 0, 0,         // color bits (ignored)
      0,                    // no alpha buffer
      0,                    // alpha bits (ignored)
      0,                    // no accumulation buffer
      0, 0, 0, 0,               // accum bits (ignored)
      16,                   // depth buffer
      0,                    // no stencil buffer
      0,                    // no auxiliary buffers
      PFD_MAIN_PLANE,           // main layer
      0,                    // reserved
      0, 0, 0,                // no layer, visible, damage masks
  };

  pixelFormat = ChoosePixelFormat(hDC, &pfd);
```

```
   SetPixelFormat(hDC, pixelFormat, &pfd);
}

LRESULT CALLBACK MainWindowProc(HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM
lParam)
{
   static HDC hDC;
   static HGLRC hRC;
   int height, width;

   // dispatch messages
   switch (uMsg)
    {
   case WM_CREATE:        // window creation
       hDC = GetDC(hWnd);
       SetupPixelFormat(hDC);
       //SetupPalette();
       hRC = wglCreateContext(hDC);
       wglMakeCurrent(hDC, hRC);
       break;

   case WM_DESTROY:          // window destroy
   case WM_QUIT:
   case WM_CLOSE:              // windows is closing

      // deselect rendering context and delete it
       wglMakeCurrent(hDC, NULL);
       wglDeleteContext(hRC);

       // send WM_QUIT to message queue
       PostQuitMessage(0);
       break;

   case WM_SIZE:
       height = HIWORD(lParam);       // retrieve width and height
       width = LOWORD(lParam);

       g_glRender->SetupProjection(width, height);

       break;

   case WM_ACTIVATEAPP:       // activate app
       break;

   case WM_PAINT:            // paint
       PAINTSTRUCT ps;
       BeginPaint(hWnd, &ps);
       EndPaint(hWnd, &ps);
       break;

   case WM_LBUTTONDOWN:       // left mouse button
       break;

   case WM_RBUTTONDOWN:       // right mouse button
       break;
```

```
case WM_MOUSEMOVE:          // mouse movement
   break;

case WM_LBUTTONUP:          // left button release
   break;

case WM_RBUTTONUP:          // right button release
   break;

case WM_KEYUP:
                  int Keys;
   LPARAM kData;
   Keys = (int)wParam;    // virtual-key code
   kData = lParam;          // key data

                  switch(Keys)
                  {
                  case VK_UP:
                           g_glRender->m_isWalking = false;
                           break;
                  default :
                           break;
                  }

   break;

case WM_KEYDOW N:
   int fwKeys;
   LPARAM keyData;
   fwKeys = (int)wParam;    // virtual-key code
   keyData = lParam;          // key data

   switch(fwKeys)
   {
   case VK_ESCAPE:
      PostQuitMessage(0);
      break;
                  case VK_LEFT:
                           g_glRender->player.turnLeft();
                           break;
                  case VK_RIGHT:
                           g_glRender->player.turnRight();
                           break;
                  case VK_UP:
                           g_glRender->m_isWalking = true;
                           g_glRender->player.forwardMove();
                           break;
                  case VK_DOWN:
                           g_glRender->player.backMove();
                           break;

   default:
      break;
   }

   break;
```

```
   default:
      break;
   }
   return DefWindowProc(hWnd, uMsg, wParam, lParam);
}

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int
nShowCmd)
{
   WNDCLASSEX windowClass;    // window class
   HWND       hwnd;           // window handle
   MSG        msg;            // message
   DWORD      dwExStyle;      // Window Extended Style
   DWORD      dwStyle;        // Window Style
   RECT       windowRect;

   g_glRender = new CGfxOpenGL;

   windowRect.left=(long)0;                    // Set Left Value To 0
   windowRect.right=(long)windowWidth; // Set Right Value To Requested Width
   windowRect.top=(long)0;                     // Set Top Value To 0
   windowRect.bottom=(long)windowHeight;   // Set Bottom Value To Requested Height

   // fill out the window class structure
   windowClass.cbSize          = sizeof(WNDCLASSEX);
   windowClass.style           = CS_HREDRAW | CS_VREDRAW;
   windowClass.lpfnWndProc     = MainWindowProc;
   windowClass.cbClsExtra      = 0;
   windowClass.cbWndExtra      = 0;
   windowClass.hInstance       = hInstance;
   windowClass.hIcon           = LoadIcon(NULL, IDI_APPLICATION);  // default icon
   windowClass.hCursor         = LoadCursor(NULL, IDC_ARROW);      // default arrow
   windowClass.hbrBackground   = NULL;                         // don't need background
   windowClass.lpszMenuName    = NULL;                          // no menu
   windowClass.lpszClassName   = "GLClass";
   windowClass.hIconSm         = LoadIcon(NULL, IDI_WINLOGO);      // windows logo small icon

   // register the windows class
   if (!RegisterClassEx(&windowClass))
      return 0;

   if (fullscreen)                         // fullscreen?
   {
      DEVMODE dmScreenSettings;                // device mode
      memset(&dmScreenSettings,0,sizeof(dmScreenSettings));
      dmScreenSettings.dmSize = sizeof(dmScreenSettings);
      dmScreenSettings.dmPelsWidth = windowWidth;        // screen width
      dmScreenSettings.dmPelsHeight = windowHeight;          // screen height
      dmScreenSettings.dmBitsPerPel = windowBits;           // bits per pixel
      dmScreenSettings.dmFields=DM_BITSPERPEL|DM_PELSWIDTH|DM_PELSHEIGHT;

      //
      if (ChangeDisplaySettings(&dmScreenSettings, CDS_FULLSCREEN) !=
DISP_CHANGE_SUCCESSFUL)
      {
```

```
            // setting display mode failed, switch to windowed
            MessageBox(NULL, "Display mode failed", NULL, MB_OK);
            fullscreen = FALSE;
        }
    }

    if (fullscreen)                         // Are We Still In Fullscreen Mode?
    {
        dwExStyle=WS_EX_APPWINDOW;              // Window Extended Style
        dwStyle=WS_POPUP;                   // Windows Style
        ShowCursor(FALSE);                  // Hide Mouse Pointer
    }
    else
    {
        dwExStyle=WS_EX_APPWINDOW | WS_EX_WINDOWEDGE;   // Window Extended Style
        dwStyle=WS_OVERLAPPEDWINDOW;                // Windows Style
    }

    AdjustWindowRectEx(&windowRect, dwStyle, FALSE, dwExStyle);     // Adjust Window To True
Requested Size

    // class registered, so now create our window
    hwnd = CreateWindowEx(NULL,                             // extended style
        "GLClass",                      // class name
        "GEClient",     // app name
        dwStyle | WS_CLIPCHILDREN |
        WS_CLIPSIBLINGS,
        0, 0,                           // x,y coordinate
        windowRect.right - windowRect.left,
        windowRect.bottom - windowRect.top, // width, height
        NULL,                           // handle to parent
        NULL,                           // handle to menu
        hInstance,                      // application instance
        NULL);                          // no extra params

    hDC = GetDC(hwnd);

    // check if window creation failed (hwnd would equal NULL)
    if (!hwnd)
        return 0;

    ShowWindow(hwnd, SW_SHOW);          // display the window
    UpdateWindow(hwnd);                                         // update the window

//          if(!g_glRender->InitConnect())
//                  exiting = true;

    g_glRender->Init();

    while (!exiting)
    {
        g_glRender->Prepare(0.0f);
        g_glRender->Render();
        SwapBuffers(hDC);

        while (PeekMessage (&msg, NULL, 0, 0, PM_NOREMOVE))
```

```
        {
            if (!GetMessage (&msg, NULL, 0, 0))
            {
                exiting = true;
                break;
            }

            TranslateMessage (&msg);
            DispatchMessage (&msg);
        }
    }

    delete g_glRender;

    if (fullscreen)
    {
        ChangeDisplaySettings(NULL,0);          // If So Switch Back To The Desktop
        ShowCursor(TRUE);                       // Show Mouse Pointer
    }

    return (int)msg.wParam;
}
```

CGfxOpenGL.h

```
// CGfxOpenGL – this is the management class.  It sets up openGL, connects to the server, and draws all
//objects
/* this code adapted from Beginning OpenGL Game Programming
  by Dave Astle and Kevin Hawkins
  Code Modified by Jesse Chandler
*/

#ifndef __GL_COMPONENT
#define __GL_COMPONENT

#include<winsock.h>
#include<string>

#include "Player.h"
#include "Room.h"
#include "Other.h"

#define PI 3.14159
#define TWO_PI PI*2.0
#define HALF_PI PI/2.0

using namespace std;

class CGfxOpenGL
{
private:
        int m_windowWidth;
        int m_windowHeight;

        SOCKET m_connectSock;
        float m_angle;
        float m_interp1;
        float m_interp2;

public:
        struct playerData{float x; float y; float z; int theta;};

        CGfxOpenGL();
        virtual ~CGfxOpenGL();

        bool Init();
        bool Shutdown();

        bool InitConnect();

        void SetupProjection(int width, int height);

        void Prepare(float dt);
        void Render();

        parseString(string input , int num);

        getPlayers();
```

```
        CPlayer player;
        COther otherPlayer;
        CRoom *room;

        playerData otherPlayers[2];

        string input[2];

        bool m_isWalking;
};

#endif
```

CGfxOpenGL.cpp

```
#ifdef _WINDOWS
#include <windows.h>
#endif

#include <gl/gl.h>
#include <gl/glu.h>
#include <math.h>
#include "CGfxOpenGL.h"
#include "Player.h"
#include "Room.h"
#include "FreeImage.h"
#include "Other.h"

// disable implic it float-double casting
#pragma warning(disable:4305)


// based on a texture loader by Jason Beverage
int awLoadTexture(const char *fileName)
{
	FIBITMAP *tempImage;
	FREE_IMAGE_FORMAT type;
	int width, height, bpp;
	GLuint result = 0;

	//Attempt to load the file using FreeImage
	tempImage = NULL;
	type = FreeImage_GetFIFFromFilename(fileName);
	tempImage = FreeImage_Load(type, fileName, 0); //FI_DEFAULT(0));
	if (tempImage == NULL)
	{
		MessageBox(NULL , "Texture Screwed up" , "Oops!" , MB_OK);
		return 0;
	}

	//Generate a new texture id for the texture
	glGenTextures(1, &result);
	if (result == 0) return 0;

	//Generate the texture
	glBindTexture(GL_TEXTURE_2D, result);
	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP);
				glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T,
GL_CLAMP);
				glTexParameteri(GL_TEXTURE_2D,
GL_TEXTURE_MAG_FILTER, GL_LINEAR);
				glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
GL_LINEAR);
	width = FreeImage_GetWidth(tempImage);
	height = FreeImage_GetHeight(tempImage);
	bpp = FreeImage_GetBPP(tempImage);

	if (bpp == 24)
	{
```

```
		gluBuild2DMipmaps(GL_TEXTURE_2D, 3, FreeImage_GetWidth(tempImage),
FreeImage_GetHeight(tempImage),
		GL_BGR_EXT, GL_UNSIGNED_BYTE, FreeImage_GetBits(tempImage));
	}
	else if (bpp == 32)
	{

		gluBuild2DMipmaps(GL_TEXTURE_2D, 4, FreeImage_GetWidth(tempImage),
FreeImage_GetHeight(tempImage),
		GL_BGRA_EXT, GL_UNSIGNED_BYTE, FreeImage_GetBits(tempImage));
	}

	//Free the image
	FreeImage_Free(tempImage);
	return result;
}

CGfxOpenGL::parseString(string input , int num)
{
	int i=0;
	string holder;
	while(input.c_str()[i] != ',')
	{
		holder += input.c_str()[i];
		i++;
	}
	this->otherPlayers[num].x = atof(holder.c_str());
	holder.erase();
	i++;

	while(input.c_str()[i] != ',')
	{
		holder += input.c_str()[i];
		i++;
	}
	this->otherPlayers[num].y = atof(holder.c_str());
	holder.erase();
	i++;

	while(input.c_str()[i] != ',')
	{
		holder += input.c_str()[i];
		i++;
	}
	this->otherPlayers[num].z = atof(holder.c_str());
	holder.erase();
	i++;

	while(input.c_str()[i] != '\0')
	{
		holder += input.c_str()[i];
		i++;
	}
	this->otherPlayers[num].theta = atoi(holder.c_str());
```

```
}



CGfxOpenGL::CGfxOpenGL()
{
}

CGfxOpenGL::~CGfxOpenGL()
{
}

// this procedure initiates connection to the server
bool CGfxOpenGL::InitConnect()
{
	WORD sockVersion;
	WSADATA wsaData;

	sockVersion = MAKEWORD(1, 1);


	// Initialize Winsock as before
	WSAStartup(sockVersion, &wsaData);

	this->m_connectSock = socket(AF_INET , SOCK_STREAM , IPPROTO_TCP);


	SOCKADDR_IN sockAddr;

	// Use port 50 and TCP/IP protocol
	sockAddr.sin_port = 50;
	sockAddr.sin_family = AF_INET;

	sockAddr.sin_addr.S_un.S_un_b.s_b1 = 127; // octet
	sockAddr.sin_addr.S_un.S_un_b.s_b2 = 0;   // octet
	sockAddr.sin_addr.S_un.S_un_b.s_b3 = 0;   // octet
	sockAddr.sin_addr.S_un.S_un_b.s_b4 = 1;   // octet

	// now attempt to connect via the socket data member... uh huh huh huh.. member
	if(connect(this->m_connectSock, (SOCKADDR *)(&sockAddr) , sizeof(sockAddr)) != 0)
	{
			MessageBox(NULL , "Connect Failed!" , "Error" , MB_OK);
			return(false);
	}
	else
		return(true);
}

CGfxOpenGL::getPlayers()
{
	struct placeHolder{string xyz[3]; int theta;};

	placeHolder p;

	char recieveText[33];
	SOCKET tempSock = this->m_connectSock;
```

```
	bool stayConnected = true;


	p.xyz[0] = "1.0";
	p.xyz[1] = "1.0";
	p.xyz[2] = "1.0";

	p.theta = 70;

	while(stayConnected)
	{
		send(tempSock , p.xyz[0].c_str() , strlen(p.xyz[0].c_str())+1 , 0); // X
		recv(tempSock , recieveText , 33 , 0);

		send(tempSock , p.xyz[1].c_str() , strlen(p.xyz[1].c_str())+1 , 0); // Y
		recv(tempSock , recieveText , 33 , 0);

		send(tempSock , p.xyz[2].c_str() , strlen(p.xyz[2].c_str())+1 , 0); // Z
		recv(tempSock , recieveText , 33 , 0);

		itoa(p.theta , recieveText , 10);
		send(tempSock , recieveText  , strlen(recieveText)+1 , 0); // theta

		for(int i=0; i<2; i++)
		{
			recv(tempSock , recieveText , 33 , 0);
			this->input[i] = recieveText;
			send(tempSock , "OK" , 3 , 0);
		}

		stayConnected = false;

		if(stayConnected)
			send(tempSock , "OK" , 3 , 0);
	}

	send(tempSock , "END" , 4 , 0);

	closesocket(tempSock);
}

bool CGfxOpenGL::Init()
{
   // clear to black background
    glClearColor(.35 , .7 , .7 , 0);

	// LOAD MODELS AND TEXTURES HERE
	this->player.loadModel("body.obj" , 0);
	this->player.loadModel("walk_1_points.obj" , 1);
	this->player.loadModel("walk_2_points.obj" , 2);
	this->player.current = 0;
	this->player.next = 1;
	this->otherPlayer.current = 0;
	this->otherPlayer.next = 1;
```

```
            this->room = new CRoom;
            this->room->texture = awLoadTexture("grass.jpg");
            this->room->loadFile("small_terrain.obj");
            this->room->root = new CRoom::node;
            this->room->root->x1 = -640;
            this->room->root->x2 = 640;
            this->room->root->z1 = -640;
            this->room->root->z2 = 640;
            this->room->buildQuadTree(this->room->root);


    m_angle = 0.0f;
            m_interp1 = 1.0;
            m_interp2 = 0.0;
            m_isWalking = false;

            // take in input from server
            this->InitConnect();
            this->getPlayers();

            this->parseString(input[0] , 0);
            this->parseString(input[1] , 1);

    return true;
}

bool CGfxOpenGL::Shutdown()
{
    return true;
}

void CGfxOpenGL::SetupProjection(int width, int height)
{
    if (height == 0)                // don't want a divide by zero
    {
        height = 1;
    }

    glViewport(0, 0, width, height);       // reset the viewport to new dimensions
    glMatrixMode(GL_PROJECTION);           // set projection matrix current matrix
    glLoadIdentity();                      // reset projection matrix

    // calculate aspect ratio of window
            gluPerspective(52.0f,(GLfloat)width/(GLfloat)height,.1f,200.0f);

            // setup fogging
            GLfloat fog_color[] = {.35, .7, .7, .1};
            glEnable(GL_FOG);
            glFogi(GL_FOG_MODE , GL_LINEAR);
            glFogfv(GL_FOG_COLOR , fog_color);
            glFogf(GL_FOG_DENSITY , .5);
            glFogf(GL_FOG_START , 1);
            glFogf(GL_FOG_END , 200);
            glHint(GL_PERSPECTIVE_CORRECTION_HINT, GL_DONT_CARE);
```

```
   glMatrixMode(GL_MODELVIEW);            // set modelview matrix
   glLoadIdentity();                      // reset modelview matrix

   m_windowWidth = width;
   m_windowHeight = height;

            glEnable(GL_TEXTURE_2D);
            glEnable(GL_DEPTH_TEST);
            glShadeModel(GL_FLAT);

}

void CGfxOpenGL::Prepare(float dt)
{
   // an idle function. This is where the animation scales are incremented

            if(m_isWalking)
            {
                        // if player is walking, we must increment scales and possibly
                        // change keyframes
                        if(m_interp1 > 0.0)
                        {
                                    m_interp1 -= .05;
                                    m_interp2 += .05;
                        }
                        else
                        {
                                    // signal to change keyframes
                                    if(this->player.current == 0)
                                    {
                                                if(this->player.next == 1)
                                                {
                                                            this->player.current = 1;
                                                            this->player.next = 0;
                                                }
                                                else
                                                {
                                                            this->player.current = 2;
                                                            this->player.next = 0;
                                                }
                                    }
                                    else
                                    {
                                                if(this->player.current == 1)
                                                {
                                                            this->player.next = 2;
                                                }
                                                else
                                                {
                                                            this->player.next = 1;
                                                }
                                                this->player.current = 0;
                                    }

                                    m_interp1 = 1.0;
```

```
                        m_interp2 = 0.0;
                }
        }
        else
        {
                // if player isn't walking, set position to standing
                m_interp1 = 1.0;
                m_interp2 = 0.0;
                this->player.current = 0;
                this->player.next = 1;
        }

}

void CGfxOpenGL::Render()
{

   // clear screen and depth buffer
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
   glLoadIdentity();

        this->player.chaseCam->update();

        this->room->drawRoom();

        this->player.drawPlayer(this->room->searchTree(this->player.posX , this->player.posZ , this->room->root) , m_interp1 , m_interp2);
        this->otherPlayer.drawOther(this->otherPlayers[0].theta , this->otherPlayers[0].y , this->otherPlayers[0].z , 1.0 , 0 , this->otherPlayers[0].theta , this->player.tlist , this->player.vlist);
        this->otherPlayer.drawOther(this->otherPlayers[1].theta , this->otherPlayers[1].y , this->otherPlayers[1].z , 1.0 , 0 , this->otherPlayers[1].theta , this->player.tlist , this->player.vlist);
}
```

CPlayer.h

```
// Player.h: interface for the CPlayer class.
//
//////////////////////////////////////////////////////////////////////

#if !defined(AFX_PLAYER_H__F756986F_2FF3_41CD_A12F_0C7CFE54A726__INCLUDED_)
#define AFX_PLAYER_H__F756986F_2FF3_41CD_A12F_0C7CFE54A726__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#include "Camera.h"
#include <string>
#include <list>

class CPlayer
{
public:
	CPlayer();
	virtual ~CPlayer();

	struct vert{float xyz[3];};
	struct triangle{int xyz[3][3];};



	drawPlayer(float posY , float scale1 , float scale2);

	loadModel(std::string filename , int key);

	forwardMove();

	backMove();

	turnLeft();

	turnRight();

	float posX;
	float posZ;

	int current;
	int next;

	int playerTheta;

	CCamera *chaseCam;

	vert vlist[3][685];
	std::list<triangle> tlist;

private:
```

```
};

#endif // !defined(AFX_PLAYER_H__F756986F_2FF3_41CD_A12F_0C7CFE54A726__INCLUDED_)
```

Player.cpp

```
// Player.cpp: implementation of the CPlayer class.
//
//////////////////////////////////////////////////////////////////////

#include "Player.h"
#include <math.h>
#include <windows.h>
#include <GL\gl.h>
#include <list>
#include <fstream>
#include <string>

#define pi 3.14159

using namespace std;

//////////////////////////////////////////////////////////////////////
// Construction/Destruction
//////////////////////////////////////////////////////////////////////


CPlayer::CPlayer()
{
	this->posX = 0;
	this->posZ = 0;
	this->playerTheta = 270;
	this->chaseCam = new CCamera;
	this->chaseCam->calcHeading(this->posX , this->posZ);

}

CPlayer::~CPlayer()
{

}

CPlayer::drawPlayer(float posY , float scale1 , float scale2)
{
	glPushMatrix();
		glTranslatef(this->posX , posY , this->posZ);
		glRotatef(90 , 0 , 1 , 0);
		glPushMatrix();

			glRotatef(this->playerTheta, 0 , 1 , 0);

			list<triangle>::iterator titer = this->tlist.begin();

			glScalef(.015 , .015 , .015);
			while(titer != this->tlist.end())
			{
				glBegin(GL_TRIANGLES);

					glVertex3f( (scale1*this->vlist[this->current][(*titer).xyz[0][0]].xyz[0] + scale2 * this->vlist[this->next][(*titer).xyz[0][0]].xyz[0]) ,
```

```
                                                                        (scale1*this->vlist[this->current][(*titer).xyz[0][0]].xyz[1] + scale2 * this->vlist[this->next][(*titer).xyz[0][0]].xyz[1]) ,
                                                                        (scale1*this->vlist[this->current][(*titer).xyz[0][0]].xyz[2] + scale2 * this->vlist[this->next][(*titer).xyz[0][0]].xyz[2]));

                                                glVertex3f( (scale1*this->vlist[this->current][(*titer).xyz[1][0]].xyz[0] + scale2 * this->vlist[this->next][(*titer).xyz[1][0]].xyz[0]) ,
                                                                        (scale1*this->vlist[this->current][(*titer).xyz[1][0]].xyz[1] + scale2 *  this->vlist[this->next][(*titer).xyz[1][0]].xyz[1]) ,
                                                                        (scale1*this->vlist[this->current][(*titer).xyz[1][0]].xyz[2] + scale2 * this->vlist[this->next][(*titer).xyz[1][0]].xyz[2]));

                                                glVertex3f( (scale1*this->vlist[this->current][(*titer).xyz[2][0]].xyz[0] + scale2 * this->vlist[this->next][(*titer).xyz[2][0]].xyz[0]) ,
                                                                        (scale1*this->vlist[this->current][(*titer).xyz[2][0]].xyz[1] + scale2 * this->vlist[this->next][(*titer).xyz[2][0]].xyz[1]) ,
                                                                        (scale1*this->vlist[this->current][(*titer).xyz[2][0]].xyz[2] + scale2 * this->vlist[this->next][(*titer).xyz[2][0]].xyz[2]));
                                                titer++;

                                        glEnd();
                                }


                        glPopMatrix();
                glPopMatrix();
}

CPlayer::loadModel(std::string filename , int key)
{
        ifstream file;
        string input , holder;
        int i;
        int count=1;
        int tmp=0;
        vert v;
        triangle t;

        file.open(filename.c_str());
        while(!(file.eof()))
        {
                getline(file , input);

                switch(input.c_str()[0])
                {
                        case '#' :
                                // # indicates comment - no action
                                break;

                        case 'v' :
                                // if it is just a v, that means vertex
                                if(input.c_str()[1] == ' ')
                                {
                                        // so we get the coordinates
                                        // first is the x component
                                        for(i=2; input.c_str()[i] != ' '; i++)
```

```
                {
                        holder += input.c_str()[i];
                }

                v.xyz[0] = atof(holder.c_str());
                holder.erase();
                i++;

                // next the y
                while(input.c_str()[i] != ' ')
                {
                        holder += input.c_str()[i];
                        i++;
                }

                v.xyz[1] = atof(holder.c_str());
                holder.erase();
                i++;

                // finally, the z component
                while(input.c_str()[i] != '\n')
                {
                        holder += input.c_str()[i];
                        i++;
                }

                v.xyz[2] = atof(holder.c_str());
                holder.erase();
                i=0;

                // now insert the vertex into the list
                this->vlist[key][count] = (v);
                count++;

        }
        else
        {
                // if it is vt, that means texture
                if(input.c_str()[1] == 't')
                {
                        // loop to get texture coordinates
                }
                else
                {
                        // if it isn't v or vt, it must be vn
                        // so we loop to get vertex normals
                }
        }
        break;

case 'f' :
        // f indicates that we have a triangle face
        // so a set of loops is needed to get the vert/normal/tex triplets

        // get the first triplet
        for(i=2; input.c_str()[i] != ' '; i++)
```

```
{
	if(input.c_str()[i] != '/')
		holder += input.c_str()[i];
	else
	{
		t.xyz[0][tmp] = atoi(holder.c_str());
		holder.erase();
		tmp++;
	}
}

t.xyz[0][2] = atoi(holder.c_str());
holder.erase();
i++;
tmp=0;

// get the next triplet
while(input.c_str()[i] != ' ')
{
	if(input.c_str()[i] != '/')
		holder += input.c_str()[i];
	else
	{
		t.xyz[1][tmp] = atoi(holder.c_str());
		holder.erase();
		tmp++;
	}

	i++;
}

t.xyz[1][2] = atoi(holder.c_str());
holder.erase();
i++;
tmp = 0;

// finally, the last triplet
while(input.c_str()[i] != ' ')
{
	if(input.c_str()[i] != '/')
		holder += input.c_str()[i];
	else
	{
		t.xyz[2][tmp] = atoi(holder.c_str());
		holder.erase();
		tmp++;
	}

	i++;
}

t.xyz[2][2] = atof(holder.c_str());
holder.erase();
i=0;
tmp = 0;
```

```
                                        // now insert the vertex into the list
                                        this->tlist.push_back(t);
                                        break;

                                case 'g' :
                                        // what do we do with a group??
                                        break;

                                case 's' :
                                        // I don't even know what this is
                                        break;

                                default :
                                        // how did we get here?
                                        break;
                        }

                }

                file.close();

}


// moves forward along the heading using polar coordinates
CPlayer::forwardMove()
{
        this->posX -=.3*cos(this->playerTheta*pi/180);
        this->posZ += .3*sin(this->playerTheta*pi/180);
        this->chaseCam->calcHeading(this->posX , this->posZ);
}



// moves backward along the heading using polar coordinates
CPlayer::backMove()
{
        this->posX += .3*cos(this->playerTheta*pi/180);
        this->posZ -= .3*sin(this->playerTheta*pi/180);
        this->chaseCam->calcHeading(this->posX , this->posZ);
}

CPlayer::turnLeft()
{
        this->playerTheta += 2;
        this->chaseCam->turnLeft();
        this->chaseCam->calcHeading(this->posX , this->posZ);
}

CPlayer::turnRight()
{
        this->playerTheta -= 2;
        this->chaseCam->turnRight();
        this->chaseCam->calcHeading(this->posX , this->posZ);
}
```

CRoom.h

```
// Room.h: interface for the CRoom class.
//
//////////////////////////////////////////////////////////////////////

#if !defined(AFX_ROOM_H__F21BDB86_F26E_4A77_A140_1310EE2D5A9F__INCLUDED_)
#define AFX_ROOM_H__F21BDB86_F26E_4A77_A140_1310EE2D5A9F__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#include<string>
#include<list>

class CRoom
{
public:


	struct point{float xyz[3];};
	struct square{point corner[4];};
	struct tri{int xyz[3][3];};
	struct node{float x1; float x2; float z1; float z2;
				struct node *child0; struct node *child1; struct node *child2; struct node
*child3;
				square *s;};



	CRoom();

	virtual ~CRoom();

	drawRoom();
	loadFile(std::string filename);
	buildQuadTree(node *nodeP);
	float searchTree(float x , float z , node *nodeP);

	int texture;

	node *root;

private:

	point vlist[4097];
	std::list<tri> tlist;



	//

};
```

```
#endif // !defined(AFX_ROOM_H__F21BDB86_F26E_4A77_A140_1310EE2D5A9F__INCLUDED_)
```

CRoom.cpp

```
// Room.cpp: implementation of the CRoom class.
//
//////////////////////////////////////////////////////////////////////

#include "Room.h"
#include <windows.h>
#include <GL\gl.h>
#include <list>
#include <fstream>
#include <string>

using namespace std;

//////////////////////////////////////////////////////////////////////
// Construction/Destruction
//////////////////////////////////////////////////////////////////////

float getHalfwayPoint(float p1 , float p2)
{
	float dist = p1 + p2;

	return(dist/2);
}

CRoom::CRoom()
{

}

CRoom::~CRoom()
{

}


CRoom::drawRoom()
{
	list<tri>::iterator titer = this->tlist.begin();
	glBindTexture(GL_TEXTURE_2D , this->texture);
	while(titer != this->tlist.end())
	{
		glBegin(GL_TRIANGLES);
			glTexCoord2f( 0 , 0); glVertex3fv(this->vlist[(*titer).xyz[0][0]].xyz);
			glTexCoord2f( 0 , 1); glVertex3fv(this->vlist[(*titer).xyz[1][0]].xyz);
			glTexCoord2f(.5 , 1); glVertex3fv(this->vlist[(*titer).xyz[2][0]].xyz);
			titer++;
		glEnd();
	}
}

CRoom::loadFile(std::string filename)
{
	ifstream file;
	string input , holder;
```

```
int i;
int count=1;
int tmp=0;
point v;
tri t;

file.open(filename.c_str());

while(!(file.eof()))
{
	getline(file , input);

	switch(input.c_str()[0])
	{
		case '#' :
			// # indicates comment - no action
			break;

		case 'v' :
			// if it is just a v, that means vertex
			if(input.c_str()[1] == ' ')
			{
				// so we get the coordinates
				// first is the x component
				for(i=2; input.c_str()[i] != ' '; i++)
				{
					holder += input.c_str()[i];
				}

				v.xyz[0] = atof(holder.c_str());
				holder.erase();
				i++;

				// next the y
				while(input.c_str()[i] != ' ')
				{
					holder += input.c_str()[i];
					i++;
				}

				v.xyz[1] = atof(holder.c_str());
				holder.erase();
				i++;

				// finally, the z component
				while(input.c_str()[i] != '\n')
				{
					holder += input.c_str()[i];
					i++;
				}

				v.xyz[2] = atof(holder.c_str());
				holder.erase();
				i=0;

				// now insert the vertex into the list
```

```
                this->vlist[count] = (v);
                count++;
                //cout << "vert!" << endl;

        }
        else
        {
                // if it is vt, that means texture
                if(input.c_str()[1] == 't')
                {
                        // loop to get texture coordinates
                }
                else
                {
                        // if it isn't v or vt, it must be vn
                        // so we loop to get vertex normals
                }
        }
        break;

case 'f' :
        // f indicates that we have a triangle face
        // so a set of loops is needed to get the vert/normal/tex triplets

        // get the first triplet
        for(i=2; input.c_str()[i] != ' '; i++)
        {
                if(input.c_str()[i] != '/')
                        holder += input.c_str()[i];
                else
                {
                        t.xyz[0][tmp] = atoi(holder.c_str());
                        holder.erase();
                        tmp++;
                }
        }

        t.xyz[0][2] = atoi(holder.c_str());
        holder.erase();
        i++;
        tmp=0;

        // get the next triplet
        while(input.c_str()[i] != ' ')
        {
                if(input.c_str()[i] != '/')
                        holder += input.c_str()[i];
                else
                {
                        t.xyz[1][tmp] = atoi(holder.c_str());
                        holder.erase();
                        tmp++;
                }

                i++;
        }
```

```
                                t.xyz[1][2] = atoi(holder.c_str());
                                holder.erase();
                                i++;
                                tmp = 0;

                                // finally, the last triplet
                                while(input.c_str()[i] != ' ')
                                {
                                        if(input.c_str()[i] != '/')
                                                holder += input.c_str()[i];
                                        else
                                        {
                                                t.xyz[2][tmp] = atoi(holder.c_str());
                                                holder.erase();
                                                tmp++;
                                        }

                                        i++;
                                }

                                t.xyz[2][2] = atof(holder.c_str());
                                holder.erase();
                                i=0;
                                tmp = 0;

                                // now insert the vertex into the list
                                this->tlist.push_back(t);
                                //cout << "face!" << endl;

                                break;

                        case 'g' :
                                // what do we do with a group??
                                break;

                        case 's' :
                                // I don't even know what this is
                                break;

                        default :
                                // how did we get here?
                                break;
                }

        }

        file.close();
}

CRoom::buildQuadTree(node *nodeP)
{
        if(nodeP->x1+20 != nodeP->x2)
        {
                nodeP->s = NULL;
```

```
		// we want to split up the node into 4 squares
		// instead of just dividing, it's half the distance to the goal
		nodeP->child0 = new node;
		nodeP->child0->x1 = nodeP->x1;
		nodeP->child0->x2 = getHalfwayPoint(nodeP->x1 , nodeP->x2);
		nodeP->child0->z1 = nodeP->z1;
		nodeP->child0->z2 = getHalfwayPoint(nodeP->z1 , nodeP->z2);

		nodeP->child1 = new node;
		nodeP->child1->x1 = nodeP->x1;
		nodeP->child1->x2 = getHalfwayPoint(nodeP->x1 , nodeP->x2);
		nodeP->child1->z1 = getHalfwayPoint(nodeP->z1 , nodeP->z2);
		nodeP->child1->z2 = nodeP->z2;

		nodeP->child2 = new node;
		nodeP->child2->x1 = getHalfwayPoint(nodeP->x1 , nodeP->x2);
		nodeP->child2->x2 = nodeP->x2;
		nodeP->child2->z1 = nodeP->z1;
		nodeP->child2->z2 = getHalfwayPoint(nodeP->z1 , nodeP->z2);

		nodeP->child3 = new node;
		nodeP->child3->x1 = getHalfwayPoint(nodeP->x1 , nodeP->x2);
		nodeP->child3->x2 = nodeP->x2;
		nodeP->child3->z1 = getHalfwayPoint(nodeP->z1 , nodeP->z2);
		nodeP->child3->z2 = nodeP->z2;

		buildQuadTree(nodeP->child0);
		buildQuadTree(nodeP->child1);
		buildQuadTree(nodeP->child2);
		buildQuadTree(nodeP->child3);

	}
	// else we've got to merge the triangles and place the squares in the leaf nodes
	else
	{
		nodeP->s = new square;

		// can this be more efficient?
		bool fount = false;
		list<tri>::iterator iter = tlist.begin();

		while(iter != tlist.end() && fount == false)
		{
			if(vlist[(*iter).xyz[1][0]].xyz[0] == nodeP->x1 &&
vlist[(*iter).xyz[2][0]].xyz[0] == nodeP->x2)
			{
				// we have found a triangle that fits the X boundaries. now we need to
fit the Z
				if(vlist[(*iter).xyz[0][0]].xyz[2] == nodeP->z1 &&
vlist[(*iter).xyz[1][0]].xyz[2] == nodeP->z2)
				{
					fount = true;
				}
			}

			iter++;
```

```
                }

                if(fount == true)
                {
                        iter--;
                        // now we place the coordinates of the triangle and its neighbor into the square
                        nodeP->s->corner[0].xyz[0] = vlist[(*iter).xyz[0][0]].xyz[0];
                        nodeP->s->corner[0].xyz[1] = vlist[(*iter).xyz[0][0]].xyz[1];
                        nodeP->s->corner[0].xyz[2] = vlist[(*iter).xyz[0][0]].xyz[2];
                        nodeP->s->corner[1].xyz[0] = vlist[(*iter).xyz[1][0]].xyz[0];
                        nodeP->s->corner[1].xyz[1] = vlist[(*iter).xyz[1][0]].xyz[1];
                        nodeP->s->corner[1].xyz[2] = vlist[(*iter).xyz[1][0]].xyz[2];
                        nodeP->s->corner[2].xyz[0] = vlist[(*iter).xyz[2][0]].xyz[0];
                        nodeP->s->corner[2].xy z[1] = vlist[(*iter).xyz[2][0]].xyz[1];
                        nodeP->s->corner[2].xyz[2] = vlist[(*iter).xyz[2][0]].xyz[2];
                        iter++;
                        nodeP->s->corner[3].xyz[0] = vlist[(*iter).xyz[1][0]].xyz[0];
                        nodeP->s->corner[3].xyz[1] = vlist[(*iter).xyz[1][0]].xyz[1];
                        nodeP->s->corner[3].xyz[2] = vlist[(*iter).xyz[1][0]].xyz[2];

                }
                else
                {
                        // put in a dummy square;
                        point dummy = {0 , 0 , 0};
                        nodeP->s->corner[0].xyz[0] = dummy.xyz[0];
                        nodeP->s->corner[1].xyz[0] = dummy.xyz[0];
                        nodeP->s->corner[2].xyz[0] = dummy.xyz[0];
                        nodeP->s->corner[3].xyz[0] = dummy.xyz[0];
                        nodeP->s->corner[0].xyz[1] = dummy.xyz[1];
                        nodeP->s->corner[1].xyz[1] = dummy.xyz[1];
                        nodeP->s->corner[2].xyz[1] = dummy.xyz[1];
                        nodeP->s->corner[3].xyz[1] = dummy.xyz[1];
                        nodeP->s->corner[0].xyz[2] = dummy.xyz[2];
                        nodeP->s->corner[1].xyz[2] = dummy.xyz[2];
                        nodeP->s->corner[2].xyz[2] = dummy.xyz[2];
                        nodeP->s->corner[3].xyz[2] = dummy.xyz[2];
                }

        }

}

float CRoom::searchTree(float x , float z , node *nodeP)
{
        // if there is no square, we're not at the lowest level. So we
        // test some boundaries to see which child node is next on the path
        if(nodeP->s == NULL)
        {
                // if x falls within the bounds of this child
                if(nodeP->child0->x1 <= x && nodeP->child0->x2 >= x)
                {
                        // then it must be on the right half of the square
                        // if it is also inside the z bound of this child
                        if(nodeP->child0->z1 <= z && nodeP->child0->z2 >= z)
                        {
```

```
                        // then it must be on the lower half of the right side
                        return searchTree(x , z , nodeP->child0); // or child zero
                }
                else
                {
                        // since the coords lie on the right half on the upper side
                        return searchTree(x , z , nodeP->child1); // it must be child one
                }
        }
        else
        {
                // if x doesn't fall on the right side, it must be the left
                // so we check the z bounds to see if it falls on the upper half
                if(nodeP->child1->z1 <= z && nodeP->child1->z2 >= z)
                {
                        // if it is the upper half,
                        return searchTree(x , z , nodeP->child3); // it must be child three
                }
                else
                {
                        // otherwise, it must be on the bottom left
                        return searchTree(x , z , nodeP->child2); // or child two
                }
        }

    }
    else
    {
        // if the square wasn't null, we must be at the lowest level
        // so we'll calculate the height based on the square
        square *ret = nodeP->s;
        float height= nodeP->s->corner[0].xyz[1];
        return height;
    }

}
```

COther.h

```
// This class is used to draw other players as received from the server
// Other.h: interface for the COther class.
//
//////////////////////////////////////////////////////////////////////

#if !defined(AFX_OTHER_H__8A9C9B1F_5558_454E_A526_EA6DE2F43468__INCLUDED_)
#define AFX_OTHER_H__8A9C9B1F_5558_454E_A526_EA6DE2F43468__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#include<windows.h>
#include<GL/gl.h>
#include<list>
#include"Player.h"

using namespace std;

class COther
{
public:

	COther();

	virtual ~COther();

	drawOther(float posX , float posY , float posZ , float scale1 , float scale2 ,
				int theta  , std::list<CPlayer::triangle> tlist , CPlayer::vert vlist[3][685]);

	int current;
	int next;

};

#endif // !defined(AFX_OTHER_H__8A9C9B1F_5558_454E_A526_EA6DE2F43468__INCLUDED_)
```

Cother.cpp

```
// Other.cpp: implementation of the COther class.
//
//////////////////////////////////////////////////////////////////////

#include "Other.h"

//////////////////////////////////////////////////////////////////////
// Construction/Destruction
//////////////////////////////////////////////////////////////////////

COther::COther()
{

}

COther::~COther()
{

}

// this implementation is sneaky: we use the geometry from another location to prevent costly
// load times and have to store even more data.
COther::drawOther(float posX , float posY , float posZ , float scale1 , float scale2 ,
                                        int theta  , std::list<CPlayer::triangle> tlist , CPlayer::vert
vlist[3][685])
{
        glPushMatrix();
                glTranslatef(posX , posY , posZ);
                glRotatef(90 , 0 , 1 , 0);
                glPushMatrix();

                        glRotatef(theta, 0 , 1 , 0);

                        list<CPlayer::triangle>::iterator titer = tlist.begin();

                        glScalef(.015 , .015 , .015);
                        while(titer != tlist.end())
                        {
                                glBegin(GL_TRIANGLES);

                                        glVertex3f( (scale1*vlist[this->current][(*titer).xyz[0][0]].xyz[0] + scale2 * vlist[this->next][(*titer).xyz[0][0]].xyz[0]) ,
                                                                (scale1*vlist[this->current][(*titer).xyz[0][0]].xyz[1] + scale2 * vlist[this->next][(*titer).xyz[0][0]].xyz[1]) ,
                                                                (scale1*vlist[this->current][(*titer).xyz[0][0]].xyz[2] + scale2 * vlist[this->next][(*titer).xyz[0][0]].xyz[2]));

                                        glVertex3f( (scale1*vlist[this->current][(*titer).xyz[1][0]].xyz[0] + scale2 * vlist[this->next][(*titer).xyz[1][0]].xyz[0]) ,
                                                                (scale1*vlist[this->current][(*titer).xyz[1][0]].xyz[1] + scale2 * vlist[this->next][(*titer).xyz[1][0]].xyz[1]) ,
                                                                (scale1*vlist[this->current][(*titer).xyz[1][0]].xyz[2] + scale2 * vlist[this->next][(*titer).xyz[1][0]].xyz[2]));
```

```
                                        glVertex3f( (scale1*vlist[this->current][(*titer).xyz[2][0]].xyz[0] + scale2 * vlist[this->next][(*titer).xyz[2][0]].xyz[0]) ,
                                                    (scale1*vlist[this->current][(*titer).xyz[2][0]].xyz[1] + scale2 * vlist[this->next][(*titer).xyz[2][0]].xyz[1]) ,
                                                    (scale1*vlist[this->current][(*titer).xyz[2][0]].xyz[2] + scale2 * vlist[this->next][(*titer).xyz[2][0]].xyz[2]));
                                        titer++;

                                glEnd();
                        }


                glPopMatrix();
        glPopMatrix();
}
```

Source Code – Server

GEServer.h

```
// GEServer.h : main header file for the GESERVER application
//

#if !defined(AFX_GESERVER_H__89421734_113C_4A37_9D08_C02A10EDFFAF__INCLUDED_)
#define AFX_GESERVER_H__89421734_113C_4A37_9D08_C02A10EDFFAF__INCLUDED_

#if _MSC_ VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#ifndef __AFXWIN_H__
	#error include 'stdafx.h' before including this file for PCH
#endif

#include "resource.h"		// main symbols


/////////////////////////////////////////////////////////////////////////////
// CGEServerApp:
// See GEServer.cpp for the implementation of this class
//

class CGEServerApp : public CWinApp
{
public:
	CGEServerApp();

// Overrides
	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CGEServerApp)
	public:
	virtual BOOL InitInstance();
	//}}AFX_VIRTUAL

// Implementation

	//{{AFX_MSG(CGEServerApp)
		// NOTE - the ClassWizard will add and remove member functions here.
		//    DO NOT EDIT what you see in these blocks of generated code !
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
};


/////////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the previous line.

#endif //
!defined(AFX_GESERVER_H__89421734_113C_4A37_9D08_C02A10EDFFAF__INCLUDED_)
```

GEServer.cpp

```
/ GEServer.cpp : Defines the class behaviors for the application.
//

#include "stdafx.h"
#include "GEServer.h"
#include "GEServerDlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CGEServerApp

BEGIN_MESSAGE_MAP(CGEServerApp, CWinApp)
	//{{AFX_MSG_MAP(CGEServerApp)
		// NOTE - the ClassWizard will add and remove mapping macros here.
		//    DO NOT EDIT what you see in these blocks of generated code!
	//}}AFX_MSG
	ON_COMMAND(ID_HELP, CWinApp::OnHelp)
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CGEServerApp construction

CGEServerApp::CGEServerApp()
{
	// TODO: add construction code here,
	// Place all significant initialization in InitInstance
}

/////////////////////////////////////////////////////////////////////////////
// The one and only CGEServerApp object

CGEServerApp theApp;

/////////////////////////////////////////////////////////////////////////////
// CGEServerApp initialization

BOOL CGEServerApp::InitInstance()
{
	// Standard initialization
	// If you are not using these features and wish to reduce the size
	//  of your final executable, you should remove from the following
	//  the specific initialization routines you do not need.

#ifdef _AFXDLL
	Enable3dControls();			// Call this when using MFC in a shared DLL
#else
	Enable3dControlsStatic();  // Call this when linking to MFC statically
#endif
```

```
	CGEServerDlg dlg;
	m_pMainWnd = &dlg;
	int nResponse = dlg.DoModal();
	if (nResponse == IDOK)
	{
		// TODO: Place code here to handle when the dialog is
		//  dismissed with OK
	}
	else if (nResponse == IDCANCEL)
	{
		// TODO: Place code here to handle when the dialog is
		//  dismissed with Cancel
	}

	// Since the dialog has been closed, return FALSE so that we exit the
	//  application, rather than start the application's message pump.
	return FALSE;
}
```

GEServerDlg.h

```
/ GEServer.cpp : Defines the class behaviors for the application.
//

#include "stdafx.h"
#include "GEServer.h"
#include "GEServerDlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CGEServerApp

BEGIN_MESSAGE_MAP(CGEServerApp, CWinApp)
        //{{AFX_MSG_MAP(CGEServerApp)
                // NOTE - the ClassWizard will add and remove mapping macros here.
                //    DO NOT EDIT what you see in these blocks of generated code!
        //}}AFX_MSG
        ON_COMMAND(ID_HELP, CWinApp::OnHelp)
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CGEServerApp construction

CGEServerApp::CGEServerApp()
{
        // TODO: add construction code here,
        // Place all significant initialization in InitInstance
}

/////////////////////////////////////////////////////////////////////////////
// The one and only CGEServerApp object

CGEServerApp theApp;

/////////////////////////////////////////////////////////////////////////////
// CGEServerApp initialization

BOOL CGEServerApp::InitInstance()
{
        // Standard initialization
        // If you are not using these features and wish to reduce the size
        //  of your final executable, you should remove from the following
        //  the specific initialization routines you do not need.

#ifdef _AFXDLL
        Enable3dControls();                     // Call this when using MFC in a shared DLL
#else
        Enable3dControlsStatic();  // Call this when linking to MFC statically
#endif
```

```
	CGEServerDlg dlg;
	m_pMainWnd = &dlg;
	int nResponse = dlg.DoModal();
	if (nResponse == IDOK)
	{
		// TODO: Place code here to handle when the dialog is
		//  dismissed with OK
	}
	else if (nResponse == IDCANCEL)
	{
		// TODO: Place code here to handle when the dialog is
		//  dismissed with Cancel
	}

	// Since the dialog has been closed, return FALSE so that we exit the
	//  application, rather than start the application's message pump.
	return FALSE;
}
```

GEServerDlg.cpp

```
// GEServerDlg.cpp : implementation file
//

#include<winsock.h>

#include "stdafx.h"
#include "GEServer.h"
#include "GEServerDlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

using namespace std;

/////////////////////////////////////////////////////////////////////////////
// CAboutDlg dialog used for App About

class CAboutDlg : public CDialog
{
public:

	CAboutDlg();

// Dialog Data
	//{{AFX_DATA(CAboutDlg)
	enum { IDD = IDD_ABOUTBOX };
	//}}AFX_DATA

	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CAboutDlg)
	protected:
	virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
	//}}AFX_VIRTUAL

// Implementation
protected:
	//{{AFX_MSG(CAboutDlg)
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
	//{{AFX_DATA_INIT(CAboutDlg)
	//}}AFX_DATA_INIT
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CAboutDlg)
```

```
	//}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
	//{{AFX_MSG_MAP(CAboutDlg)
		// No message handlers
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CGEServerDlg dialog

CGEServerDlg::CGEServerDlg(CWnd* pParent /*=NULL*/)
	: CDialog(CGEServerDlg::IDD, pParent)
{
	//{{AFX_DATA_INIT(CGEServerDlg)
		// NOTE: the ClassWizard will add member initialization here
	//}}AFX_DATA_INIT
	// Note that LoadIcon does not require a subsequent DestroyIcon in Win32
	m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
}

void CGEServerDlg::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CGEServerDlg)
		// NOTE: the ClassWizard will add DDX and DDV calls here
	//}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CGEServerDlg, CDialog)
	//{{AFX_MSG_MAP(CGEServerDlg)
	ON_WM_SYSCOMMAND()
	ON_WM_PAINT()
	ON_WM_QUERYDRAGICON()
	ON_BN_CLICKED(IDC_BUTTON_LISTEN, OnButtonListen)
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CGEServerDlg message handlers

BOOL CGEServerDlg::OnInitDialog()
{
	CDialog::OnInitDialog();

	// Add "About..." menu item to system menu.

	// IDM_ABOUTBOX must be in the system command range.
	ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
	ASSERT(IDM_ABOUTBOX < 0xF000);

	CMenu* pSysMenu = GetSystemMenu(FALSE);
	if (pSysMenu != NULL)
	{
		CString strAboutMenu;
```

```
		strAboutMenu.LoadString(IDS_ABOUTBOX);
		if (!strAboutMenu.IsEmpty())
		{
			pSysMenu->AppendMenu(MF_SEPARATOR);
			pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX, strAboutMenu);
		}
	}

	// Set the icon for this dialog.  The framework does this automatically
	//  when the application's main window is not a dialog
	SetIcon(m_hIcon, TRUE);			// Set big icon
	SetIcon(m_hIcon, FALSE);		// Set small icon

	// TODO: Add extra initialization here

	return TRUE;  // return TRUE  unless you set the focus to a control
}

void CGEServerDlg::OnSysCommand(UINT nID, LPARAM lParam)
{
	if ((nID & 0xFFF0) == IDM_ABOUTBOX)
	{
		CAboutDlg dlgAbout;
		dlgAbout.DoModal();
	}
	else
	{
		CDialog::OnSysCommand(nID, lParam);
	}
}

// If you add a minimize button to your dialog, you will need the code below
//  to draw the icon.  For MFC applications using the document/view model,
//  this is automatically done for you by the framework.

void CGEServerDlg::OnPaint()
{
	if (IsIconic())
	{
		CPaintDC dc(this); // device context for painting

		SendMessage(WM_ICONERASEBKGND, (WPARAM) dc.GetSafeHdc(), 0);

		// Center icon in client rectangle
		int cxIcon = GetSystemMetrics(SM_CXICON);
		int cyIcon = GetSystemMetrics(SM_CYICON);
		CRect rect;
		GetClientRect(&rect);
		int x = (rect.Width() - cxIcon + 1) / 2;
		int y = (rect.Height() - cyIcon + 1) / 2;

		// Draw the icon
		dc.DrawIcon(x, y, m_hIcon);
	}
	else
	{
```

```
                CDialog::OnPaint();
        }
}

// The system calls this to obtain the cursor to display while the user drags
//  the minimized window.
HCURSOR CGEServerDlg::OnQueryDragIcon()
{
        return (HCURSOR) m_hIcon;
}

// this is the communication thread procedure
UINT threadProc(LPVOID pParam)
{
        struct passStruct{SOCKET socket; int id; string clients[10];};

        passStruct *data = (passStruct *)pParam;
        int id = data->id;
        int i;

        SOCKET tempSock = data->socket;
        char recieveText[4][33];
        string outText;

        data->clients[0] = "10.0,0.0,30.0,100";
        data->clients[1] = "-50.0,0.0,2.0,66";
        data->clients[2] = "25.0,0.0,60.0,90";


        while(true)
        {
                // recieve position and heading data from the client
                recv(tempSock , recieveText[0] , 33 , 0); // X
                // handle data
                outText += recieveText[0];
                outText += ',';
                send(tempSock , "OK" , 3 , 0);

                recv(tempSock , recieveText[1] , 33 , 0); // Y
                // handle data
                outText += recieveText[1];
                outText += ',';
                send(tempSock , "OK" , 3 , 0);

                recv(tempSock , recieveText[2] , 33 , 0); // Z
                // handle data
                outText += recieveText[2];
                outText += ',';
                send(tempSock , "OK" , 3 , 0);

                recv(tempSock , recieveText[3] , 33 , 0); // theta
                // handle data
                outText += recieveText[3];

                // loop here to send
                for(i=0; i<3; i++)
```

```
                {
                        if(i != id)
                        {
                                send(tempSock , data->clients[i].c_str() , strlen(data-
>clients[i].c_str())+1 , 0);
                                recv(tempSock , recieveText[0] , 33 , 0);
                        }
                }
                recv(tempSock , recieveText[0] , 33 , 0);
                if(strcmp(recieveText[0] , "OK") != 0)
                        break;


        }


        closesocket(tempSock);

        return(0);
}


void CGEServerDlg::OnButtonListen()
{
        // TODO: Add your control notification handler code here
        WORD sockVersion;
        WSADATA wsaData;
        int clientNum =0;
        string clientList[10];

        sockVersion = MAKEWORD(1, 1);


        // Initialize Winsock as before
        WSAStartup(sockVersion, &wsaData);

        this->m_sock = socket(AF_INET , SOCK_STREAM , IPPROTO_TCP);


        int nret;
        // Use a SOCKADDR_IN struct to fill in address information
        SOCKADDR_IN serverInfo;

        serverInfo.sin_family = AF_INET;
        serverInfo.sin_addr.s_addr = INADDR_ANY;
        serverInfo.sin_port = 50;

        nret = bind(m_sock, (LPSOCKADDR)&serverInfo, sizeof(struct sockaddr));

        if (nret == SOCKET_ERROR)
        {
                MessageBox("Bind failed");
                exit(0);
        }
```

```
	// outer loop
	for( ; ;)
	{

		listen(m_sock , 10);

		SOCKET tempSock;
		tempSock = accept(m_sock , NULL , NULL);
		passStruct *data = new passStruct;
		data->id = clientNum;
//		data->clients = clientList;
		data->socket = tempSock;
		// thread procedure call
		AfxBeginThread(threadProc, (LPVOID)data , THREAD_PRIORITY_NORMAL , 0 , 0 ,
NULL);
		clientNum++;

		//closesocket(tempSock);
	}

}
```

StdAfx.h

```
// stdafx.h : include file for standard system include files,
//  or project specific include files that are used frequently, but
//      are changed infrequently
//

#if !defined(AFX_STDAFX_H__0C62862D_4D1E_4F5F_A0BB_440F4211D1ED__INCLUDED_)
#define AFX_STDAFX_H__0C62862D_4D1E_4F5F_A0BB_440F4211D1ED__INCLUDED_
```

```
#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#define VC_EXTRALEAN                    // Exclude rarely-used stuff from Windows headers

#include <afxwin.h>         // MFC core and standard components
#include <afxext.h>         // MFC extensions
#include <afxdtctl.h>               // MFC support for Internet Explorer 4 Common Controls
#ifndef _AFX_NO_AFXCMN_SUPPORT
#include <afxcmn.h>                 // MFC support for Windows Common Controls
#endif // _AFX_NO_AFXCMN_SUPPORT


//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_STDAFX_H__0C62862D_4D1E_4F5F_A0BB_440F4211D1ED__INCLUDED_)
```

Works Referenced

- Overview of the Windows API, Microsoft Press

  http://msdn.microsoft.com/library/default.asp?url=/library/en-us/winprog/winprog/overview_of_the_windows_api.asp

- Woo, Neider, Davis, Schreiner - OpenGL Programming Guide, Chapter 1. Addison Wesley, 2001