**Computer Graphics – Final Project**

Due Wednesday December 5th, 2012

Eric Arndt

Matthew Farkas

Michael Yahner

**Intro**

We decided for our final project to make game engine. Our final project is a 3D game engine implemented using C#. The framework version is .net 4.0 and using DirectX 9 via a wrapper in C# called Slimdx.

**Functionality**

Shape Manipulations: The three main shape manipulations we learned in class were implemented including Translation, Rotation, and Scaling. Users can define two main types of shapes: Triangles and Squares which can be added, removed, and modified within the scene. Color may also be applied to shapes.

3D Models: – The engine can load in 3D models. Though the only models accepted are Microsoft’s legacy “.x” model. These models cannot be animated (skinned).

Lighting: Two different types of lights were implemented that we learned in class including Point and Directional light. Lights can be added, removed, and modified within the scene.

Physics: Life-like gravity was implemented using object translations and positioning of the terrain. As objects move farther away from its starting location, they appear to be under a realistic gravity force by accelerating.

Wireframe: The ability to convert a scene of objects into wireframe was implemented.

**How to Use:**

Please view the README in Computer Graphics Term Project/Graphics Project Runnable-Executable folder. There are explicit understandable and long instructions in there. There are also a different set of instructions in the Computer Graphics Term Project/Graphics Project Source Code folder. That one explains how to start and run the source code.

**What we each did**

**Eric Arndt:**

**Matthew Farkas:** My primary responsibility was to create most of the user-interface (GUI). The main layout was derived from existing game engines like Unity 3D, to include a main display panel shown in the middle of the screen surrounded with controls on the left and the right and a menu items bar. Descriptions of main GUI controls are as follows:

Mouse events controls for camera movement: Buttons were created to translate or rotate the camera along the x, y and z axis. In addition, mouse events were created to pan the scene by clicking and dragging within the panel and zooming in and out with the mouse wheel. When the camera is moved, labels are updated to indicate its current position.

Environment Checkboxes: A series of checkboxes were created and wired events to turn gravity on/off, turn wireframe on/off, turn global lights on/off, and to clear the current scene.

Object Input: An area for objects was created to keep track of all the user-defined objects. Menu bar items were created to easily add Squares and Triangles. For each shape that is added, a new entry would be created in the Object list drop down list. A textbox and button were created to rename objects so that they can easily be identified.

Object translations, rotation, and scaling input controls: Three areas were created to perform the basic shape translations, rotation, and scaling along the x, y, and z axis. Upon key-pressing the enter key, the shape will be positioned as indicated. Validation was added throughout to ensure a number was passed into each textbox. In addition, a color picker was created to color objects.

Light controls: An area was created, similar to the object area, to keep track of all user-defined lights. Menu bar items were created to easily add Point and Directional lights. For each light that is added, a new entry would be created in the Light list drop down list. In addition, text boxes were created with validation to manipulate the location and direction of the lights.

Function to select and load files for meshes: An open file dialog was created, that specifically loads .x files to be used as meshes for objects.

**Michael Yahner:**

I designed and built the entire graphics engine portion. This includes the Mesh, Light, and Terrain. This also includes setting up the graphics device, creating the custom vertex for the fixed function pipeline, the camera class, and the renderer class.

The device class just sets up parameters for the device and then instantiates the device. Along with this class is the camera class that sets up a 3D camera with the near and far plane. This class also allows you to move around in your 3D world along with rotating it. The custom vertex class which defines custom vertex structures for telling the graphics device what data it is receiving. The last class that accompanies that device is the renderer class. This class is for displaying everything that is created on the device’s buffer.

Meshes are just a collection of vertices in the buffer and indices which save on the amount of buffers you need. The Mesh class holds the ability to load an .x mesh and create the two stated meshes from earlier. It also holds the information about where they are located, rotated, scaled and what color they are. The light class holds the information about creating two different types of lights and where they are or pointing to. The terrain randomly creates itself every time you run it. It creates a random width and height no larger than 50 units and no less than 2 units. The terrain is created along the x and z access with the height being on the y access.