Creating a 3D Game With Textures and Lighting

John Fitzsimmons

I. Abstract

The objective is to create a simple, 3D first person shooter using OpenGL and C++. To acheive this goal, I will need to implement textures, lighting, 3Dshapes, polygons, and basic C++ structures.

II. Introduction

The problem of creating a 3D first person shooter from scratch is a large task. To accomplish this task, I am breaking down the problem into simple blocks, such as getting a camera to work so that I can view the world from a person's perspective and move around. The reason I chose this project is because computer games interest me immensly and creating one of my own would give me some experience and pleasure, even if it is very primitive.

My project will probably be finished very near the end of the year as there will always be another upgrade to add. The maximum scope of this program would be to have multiple users playing on the same computer or over a network and models of the users and different objects. Multiple weapons would be available to pick up and different levels to play. However, the multiple user aspect nor the models will be accomplishable by the end of the year.

III. Theory and Discusion and Workplan

I will go over the two main features that I have completed so far: the camera and collisions. The camera is the user's point of view. Using keyboard input to change velocity, the user moves around. The largest task was create a realistic system to use mouse input to rotate the camera up, down, and side to side. To do this, horizontal and vertical movement is calculated in pixels for each iteration and used to modify two angles. One for up and down and the other for side to side. Using both of these angles, I calculate a unit vector in the opposite direction the user is looking to input into an openGL function. The distance the user looks up is the sine of the angle up. The maximum distance forward is the cosine of this angle. The forward distance is further broken up into x and z components. The x component is the max distance multiplied by the sine of the angle to the side and the y component is the max distance multiplied by the cosine of that angle. As a result, my code looks like such with a unit vector of length 2:

```
temp=2*cos(angleDown);
centerX=eyeX-temp*sin(angleLeft);//puts center point 2 units
centerZ=eyeZ-temp*cos(angleLeft);//in front of the eye
centerY=eyeY-2*sin(angleDown);
```

An up vector is also generated very similar to this to make the camera right side up.

The other large chunk of code is collisions. My collisions are very simple. When a projectile hits an object, it stops and moves back a little (so it doesn't get stuck in the object). The acceleration due to gravity then pulls the object down. If it hits the ground or the top of an object it just stops and stays.

After a certain amount of time, the projectile will be deleted. That way, there won't be hundreds of projectiles bogging down on processing power. To detect collisions, I check to see if the projectiles is inside the bounds of the object plus the radius of the projectile. This currently only works for rectangular prisms or flat rectangles. Circles and spheres are also workable with this algorithm. To do more complicated objects, multiple smaller, undrawn objects will have to be used to approximate the larger shape and acheive accurate collisions.

Another large piece of the program will be textures. I will have to load in a texture for each different type of surface I am going to use. To do this I must read a bitmap image into memory. I do this by first reading in the bitmap header which contains information about the file, and then I read in 4 values for every pixel. Each value is the size of a byte, so I read these values into an array of unsigned chars. This array is then plugged into a texture function (glTexImage2D) which stores the image in video memory. I then delete the array from memory. For some reason, the read in textures are not staying in video memory, even though I have debuged and found I interpreted the bitmap correctly. Textures will only be accomplished if time permits because I have decided to push back fixing the problem until later.

The other main aspect of collisions are the actual objects. I have created a "class" object and I have a pointer list of these objects. When a destroyable object gets hit enough, it will be destroyed and I will take it out of the list so it isn't drawn and doesn't use processing power. Currently, I am working on a system to read in objects and put them in the list. When I finish, I will be able to load in an environment from a text file, allowing for multiple levels. After accomplishing muliple text input levels, I will work on a HUD (heads up display) which will show life, time, and score. I will also add different weapons that cause different damages that the player can pick up in the level. Lighting will be added last, such as a muzzle flash effect and lighting throughout the level.

My basic system of operations is to code, test, and debug. I do this for every single aspect of the project. When I added the camera code, I had to fix alot of errors. For example, I found out on the opengl.org site that the "center" values determine a point a unit away from the camera in the direction it is looking rather than a unit vector. The "up" values actually do determine a unit vector in the "up" direction.

Only I am working on this project. I need no money and therefore have no budget. The only equipment I am using is a computer, implementing OpenGl and C++. I may also use a graphical modeler later on. I have mainly used the OpenGL website to get all of my information for my project. However, as I am now getting into some more complicated aspects (not just camera and drawing basic objects), I will need more and more sources. I have begun to research texturing, which I have researched outside the opengl.org site. I will have to research elsewhere for most of the remaining tasks as well. Tasks to be completed include adding textures, lighting, additional objects, and a more

complicated environment.

Sources:

OpenGL - The Industry Standard for High Performance Graphics. http://www.opengl.org/. January 27, 2005.

Spacesimulator.net - OpenGL Texture mapping. http://www.spacesimulator.net/tut3_texturemapping.htm January 26, 2005.

OpenGL Texture Mapping : An Introduction. http://www.gmonline.demon.co.uk/cscene/CS8/CS8-02.html. January 23, 2005.