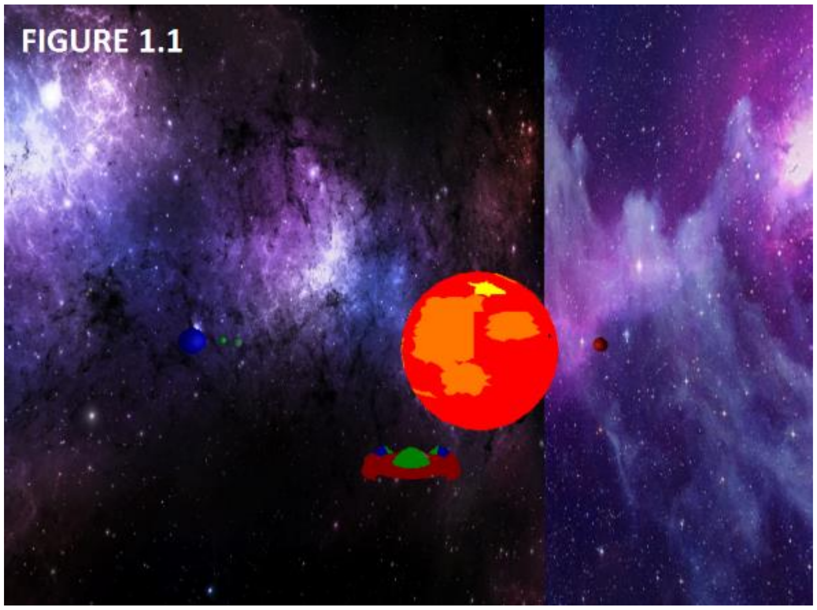
Comp 465 Expansion Pack – OpenGL Tutorials

-By: Jared Fowler 100%

Spring 2015

Introduction:

CSUN's Comp 465, as taught by Professor Mike Barnes, is an introductory graphics course which introduces the basic concepts of openGL, including, but not limited to, the concepts of modeling, model manipulation via matrices, and shaders via glsl. Warbird Simulator, the final project as can be seen in figure 1.1, represents the pinnacle of the material learned in Comp 465. (Granted that most of the hard work is done under the hood!) This was an amazing accomplishment and learning experience for anyone being introduced to graphics for the first time. It does, however, leave much to be desired.



My original project proposal was to study tutorials based on shadow mapping and then apply what I learned to the simulator. After not having directly worked with openGL for the last four months, I began reading tutorials and quickly found myself confused. I quickly realized a couple of things: One, not only did I need a refresher course, but there were techniques in openGL that I had either never seen or never fully understood before. Two, if I'm going to have to learn some new techniques before rendering the shadow maps, why not take the whole simulator to the next level? This is exactly what I did, well, sort of. The projects I've created do not have smart missiles and a cool chasing algorithm, rather, they focus much more upon the rendering of graphics. What do I mean by "projects"? I'm the type of person that learns by hands on application, thus, I created several projects, each one representing what I learned in a specific tutorial(s). These projects build on top of one another. For the vast majority of this report I will discuss what these projects are, the difficulties encountered in each, and any workarounds.

Table of Contents:

Section Name	**Tutorials Used**	**Hours Spent**
Installation, Usage, and Architecture	Professor Barnes' openGL setup with Visual Studio	5
Project 1: Textures and Music	http://www.opengl-tutorial.org/ Tutorials 1-5, 7 http://www.ambiera.com/irrklang/tutorials.html	15
Project 2: Mouse and Keyboard	http://www.opengl-tutorial.org/ Tutorial 6	4
Project 3: Directional Light	http://www.opengl-tutorial.org/ Tutorial 8	2:30
Project 4: IBOs and 2DText	http://www.opengl-tutorial.org/ Tutorials 9, 11	4
Project 4B: Multi-Texturing	http://www.mbsoftworks.sk/ Tutorial 14	12
Project 5: Multi-Models and Camera	Reference: 465 Warbird Simulator	8
Project 5B: Star Field	http://stackoverflow.com/	8
Project 6: Shadow Mapping	http://www.opengl-tutorial.org/ Tutorial 16 OpenGL Programming Guide 4.3	10

Installation, Usage, and Architecture:

The projects were created and used via Visual Studio 2013 Ultimate. You will need to install the glm, glew, and freeGlut libraries. Professor Barne's has a nice tutorial already made up on how to do this. CAUTION: Try to download the versions as indicated in Barnes' tutorial. I spent hours hitting my head against the wall while trying to re-vitalize my 465 project. The problem turned out to be compatibility issues with the new freeGlut version. You may try to use the newest versions at your own discretion.

Upon opening the solution (.sln) file, you'll see in your solution explorer window a list of projects. You can switch between which one you want to build and start up by right clicking on the project and selecting "Set as Startup Project". Each of the projects have four configurations set in the properties option.

1. Properties.C/C++.Preprocessor.ProcessorDefinition :: _CRT_SECURE_NO_WARNINGS
2. Properties.C/C++.Preprocessor.General.AdditionalIncludeDirectories :: ../Common/AUDIO/include
3. Properties.Linker.General.AdditionalLibraryDirectories :: ../Common/AUDIO
4. Properties.Linker.Input.AdditionalDependencies :: freeglut.lib, glew32.lib

There is both common content and individual project content. Common content includes textures, model files, sound files, and common #included header files which I either wrote or borrowed from the openGL tutorials. All of the code which I wrote/modified should contain my name in a header comment. The individual projects have their own MainV<version>.cpp file and shaders. I've attempted to make the projects as modular as possible. The files are written/placed in such a way that only a few lines of code need be added to insert a new model, texture, shader, etc. This modularity increases with each new project version.

Keyboard and Mouse Functionality: (CAUTION: Not all commands work for lower versions.)

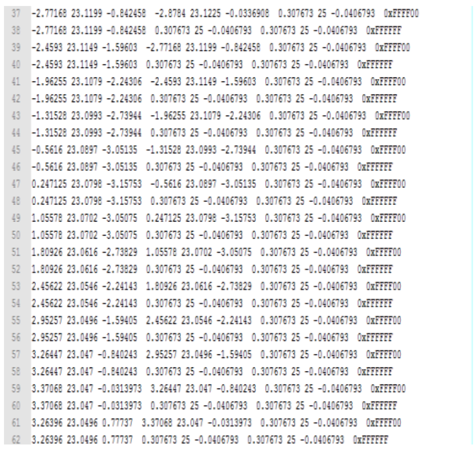
KEY	**FUNCTION**
Mouse	Rotate Camera
Esc	Exit Program
F1	Free/Bind Mouse
F2	FPS MAX on/off
F3	Show/Hide 2D Text

F4	Music off/on
F5	Next Audio
Arrows	Move Camera
V	Change Camera View
F	Change Camera to free look

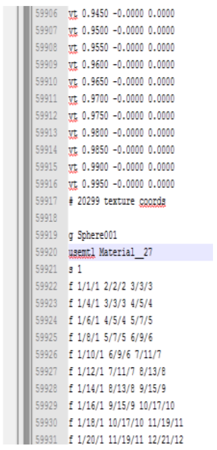
<u>Project 1: Textures and Music</u>

About:

This project's original name was "Simple Textured Earth". I wanted to create a simple sphere with an earth texture mapped to it. The only texture mapping experience I ever had previously was that of the 'sky-box' in Warbird Simulator. This entire project, therefore, was a learning experience. I created a sphere using 3ds-Max, found an earth texture online, converted the texture to .DDS format using and extension I found for GIMP, and finally, mapped the texture to the sphere. The reader should assume, from this point forward, that 3ds-Max and GIMP will be the only tools used for these processes. Following the tutorial, I exported the model as an .obj file. I was both excited and delighted to learn what composes an .obj file. Unlike the .tri file used in 465 where every line represents a triangle and color, the .obj file contains 4

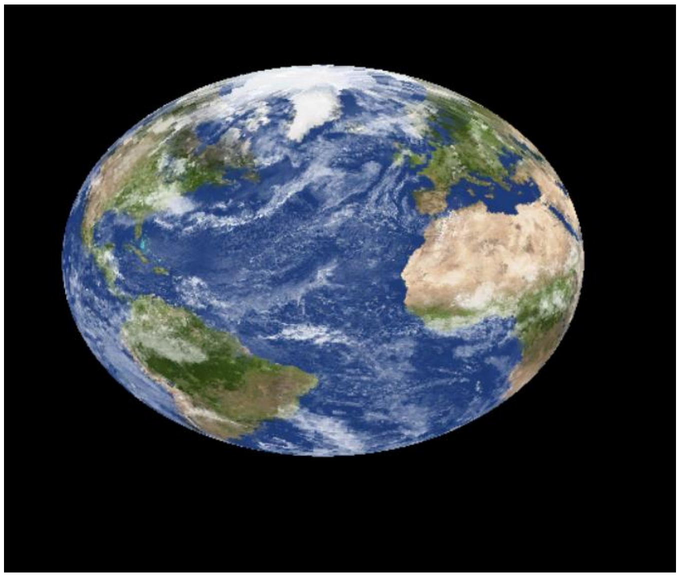


```
-2.77168 23.1199 -0.842458  -2.8784 23.1225 -0.0336908  0.307673 25 -0.0406793  0xFFFF00
-2.77168 23.1199 -0.842458  0.307673 25 -0.0406793  0.307673 25 -0.0406793  0xFFFFFF
-2.4593 23.1149 -1.59603  -2.77168 23.1199 -0.842458  0.307673 25 -0.0406793  0xFFFF00
-2.4593 23.1149 -1.59603  0.307673 25 -0.0406793  0.307673 25 -0.0406793  0xFFFFFF
-1.96255 23.1079 -2.24306  -2.4593 23.1149 -1.59603  0.307673 25 -0.0406793  0xFFFF00
-1.96255 23.1079 -2.24306  0.307673 25 -0.0406793  0.307673 25 -0.0406793  0xFFFFFF
-1.31528 23.0993 -2.73944  -1.96255 23.1079 -2.24306  0.307673 25 -0.0406793  0xFFFF00
-1.31528 23.0993 -2.73944  0.307673 25 -0.0406793  0.307673 25 -0.0406793  0xFFFFFF
-0.5616 23.0897 -3.05135  -1.31528 23.0993 -2.73944  0.307673 25 -0.0406793  0xFFFF00
-0.5616 23.0897 -3.05135  0.307673 25 -0.0406793  0.307673 25 -0.0406793  0xFFFFFF
0.247125 23.0798 -3.15753  -0.5616 23.0897 -3.05135  0.307673 25 -0.0406793  0xFFFF00
0.247125 23.0798 -3.15753  0.307673 25 -0.0406793  0.307673 25 -0.0406793  0xFFFFFF
1.05578 23.0702 -3.05075  0.247125 23.0798 -3.15753  0.307673 25 -0.0406793  0xFFFF00
1.05578 23.0702 -3.05075  0.307673 25 -0.0406793  0.307673 25 -0.0406793  0xFFFFFF
1.80926 23.0616 -2.73829  1.05578 23.0702 -3.05075  0.307673 25 -0.0406793  0xFFFF00
1.80926 23.0616 -2.73829  0.307673 25 -0.0406793  0.307673 25 -0.0406793  0xFFFFFF
2.45622 23.0546 -2.24143  1.80926 23.0616 -2.73829  0.307673 25 -0.0406793  0xFFFF00
2.45622 23.0546 -2.24143  0.307673 25 -0.0406793  0.307673 25 -0.0406793  0xFFFFFF
2.95257 23.0496 -1.59405  2.45622 23.0546 -2.24143  0.307673 25 -0.0406793  0xFFFF00
2.95257 23.0496 -1.59405  0.307673 25 -0.0406793  0.307673 25 -0.0406793  0xFFFFFF
3.26447 23.047 -0.840243  2.95257 23.0496 -1.59405  0.307673 25 -0.0406793  0xFFFF00
3.26447 23.047 -0.840243  0.307673 25 -0.0406793  0.307673 25 -0.0406793  0xFFFFFF
3.37068 23.047 -0.0313973  3.26447 23.047 -0.840243  0.307673 25 -0.0406793  0xFFFF00
3.37068 23.047 -0.0313973  0.307673 25 -0.0406793  0.307673 25 -0.0406793  0xFFFFFF
3.26396 23.0496 0.77737  3.37068 23.047 -0.0313973  0.307673 25 -0.0406793  0xFFFF00
3.26396 23.0496 0.77737  0.307673 25 -0.0406793  0.307673 25 -0.0406793  0xFFFFFF
```



```
vt 0.9450 -0.0000 0.0000
vt 0.9500 -0.0000 0.0000
vt 0.9550 -0.0000 0.0000
vt 0.9600 -0.0000 0.0000
vt 0.9650 -0.0000 0.0000
vt 0.9700 -0.0000 0.0000
vt 0.9750 -0.0000 0.0000
vt 0.9800 -0.0000 0.0000
vt 0.9850 -0.0000 0.0000
vt 0.9900 -0.0000 0.0000
vt 0.9950 -0.0000 0.0000
# 20299 texture coords

g Sphere001
usemtl Material__27
s 1
f 1/1/1 2/2/2 3/3/3
f 1/4/1 3/3/3 4/5/4
f 1/6/1 4/5/4 5/7/5
f 1/8/1 5/7/5 6/9/6
f 1/10/1 6/9/6 7/11/7
f 1/12/1 7/11/7 8/13/8
f 1/14/1 8/13/8 9/15/9
f 1/16/1 9/15/9 10/17/10
f 1/18/1 10/17/10 11/19/11
f 1/20/1 11/19/11 12/21/12
```



sections: v (vertices), vn (normal of vertices), vt (texture map vertices), and f which links the previous three vertex types together. The first line of f, for example, has 1/1/1 2/2/2 3/3/3. Each group of 3 numbers represents a vertex, its normal, and texture map coord. The numeric value is essentially an index of a vertex in one of the lists. (1/20/4 → in list v, the 1st vertex; in list vn, the 20th vertex; in list vt, the 4th vertex)

Adding music to the project was just a bonus. Last semester I wanted to add music to Warbird Simulator but lacked the time and knowledge to do so. I found a nice API called irrklang. Once working, it was super easy to use and certainly improves the project.

Result:

The earth rotates. Camera is stationary. The shaders used are absolutely basic, essentially calculating position based upon translation and rotation, and passing the texture to the fragment shader.

Errors/Work-Arounds:

There were three major issues which caused much grief and lost time. First, the openGL tutorial recommended using a texture compressing tool called "The Compressionator" to compress textures to .DDS format. Using this tool proved faulty and textures were not rendered at all. I downloaded an extension for GIMP which fixed this issue. Second, a simple openGL misconfiguration. "GL_ELEMENT_BUFFER" versus "GL_ARRAY_BUFFER". This error also prevented the texture from rendering. Third, irrklang x64 versus x32. I downloaded the 64-bit version of the library and could not get it to work. After spending an hour or so working with the linker properties, I tried the 32-bit version and everything worked great.

<u>Project 2: Mouse and Keyboard</u>

About:

Keyboard events were a simple carry over from Warbird Simulator. Adding mouse events was a new learning experience. The concepts were easy enough, but how the events are timed and handled turned out very interesting. The freeGlut event listeners are constantly listening for mouse and keyboard events. Instead of directly calling a function which handles the event, a function is called which puts the event onto a queue. (std::vector, so technically a stack.) The update function will then call another function which will take these events and handle them.

Result:

We now have a free view camera which can move forward/backwards and side-to-side with the keyboard, and can be rotated with the mouse.

Errors/Work Arounds:

Mouse events were being triggered without the mouse being moved. This was not an error with the code, rather, it was an error with my external mouse. Upon unplugging it and only using my laptop mousepad, this problem went away.

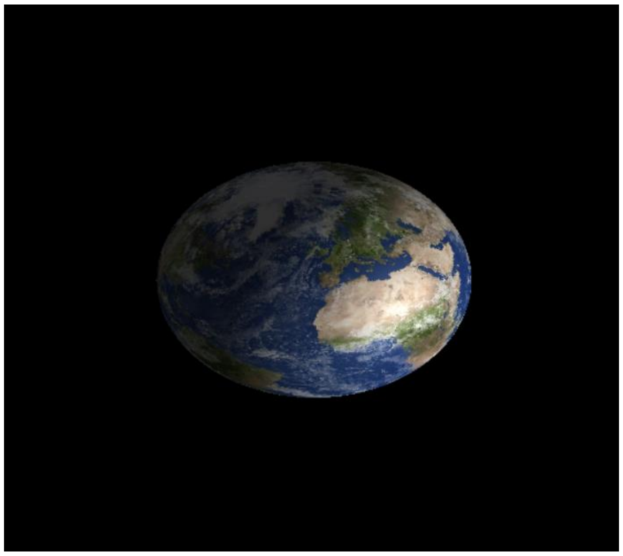
<u>**Project 3: Directional Light**</u>

About:

This project was relatively quick to implement. I utilized the shaders provided by the openGL tutorial. Lighting in these shaders is done via the Phong model, the same model which I attempted to use in Warbird Simulator. Modifications were made to the code to pass in different matrices to the shaders, other than the normal MVP, such as View and ModelView matrices, and light position.

Result:

The shaders worked beautifully!



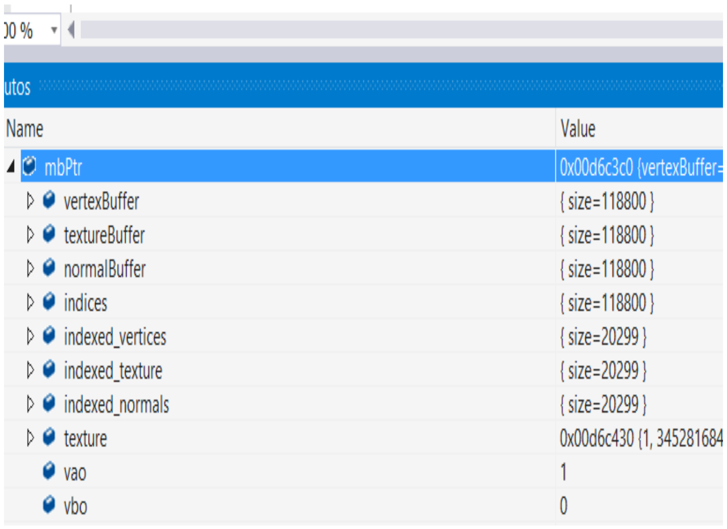
Errors/Work Arounds:

Shader light functionality was modified to work with far away objects. The light intensity value was increased a thousand-fold times or so to obtain the scene shown here. In a later project the light positions and intensity are handled outside the shader and are passed into the shader as a uniform vec4.

Project 4: IBOs and 2D Text

About:

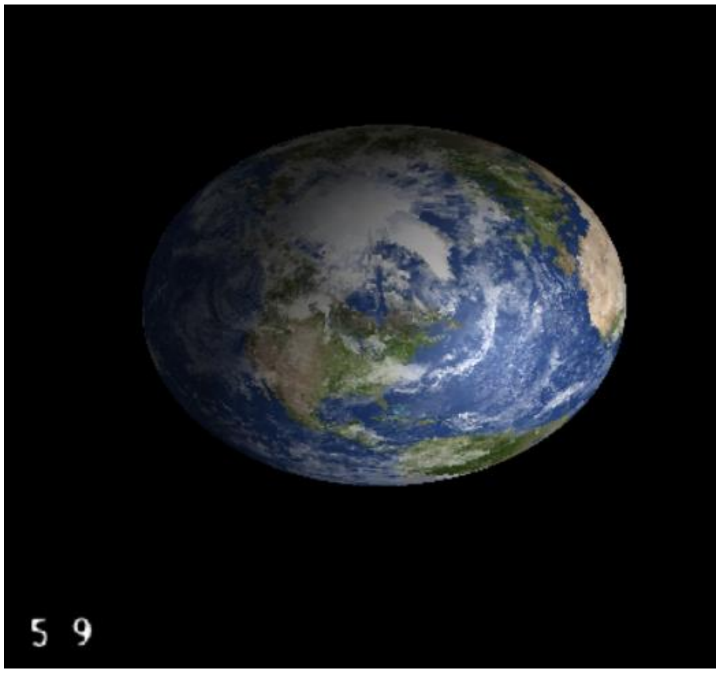
Index buffers are used to save resource space on the graphics card. This concept was only introduced and slightly used in Warbird Simulator. I modified my project to utilize index buffers for all the models. The openGL tutorial provided a 2D Texture API which I use to display the FPS.



```
Name                    Value
◢ mbPtr                 0x00d6c3c0 {vertexBuffer=
  ▷ vertexBuffer        { size=118800 }
  ▷ textureBuffer       { size=118800 }
  ▷ normalBuffer        { size=118800 }
  ▷ indices             { size=118800 }
  ▷ indexed_vertices    { size=20299 }
  ▷ indexed_texture     { size=20299 }
  ▷ indexed_normals     { size=20299 }
  ▷ texture             0x00d6c430 {1, 345281684
    vao                 1
    vbo                 0
```

Result:

We can see the size of the different vertex buffers in the Visual Studio Debugger for a single model, as presented in this image. Without indexing, we would have a total of (118800 * 3 = 356,400) vertices. With indexing, we reduce this number down to (118800 + 20299 * 3 = 179,697) vertices. This is about a 50% decrease in resources used for this **single** model. The 2D Texture API was easy to configure and use! I decided to also take the opportunity to manage the Updates/Second, which directly effects the FPS. Every 1000ms the program looks at the number of draws since the last check. If the number is below 60 it decreases the time between updates, else, it increases the time. I also added functionality to allow maximum FPS. This was done by setting the glutIdle function handler to the draw function. This does not result in a



continuous set of unique frames, rather, it gives us an idea of the rendering power of the CPU and GPU. With this setting active on my laptop, the scene runs about 800 FPS.

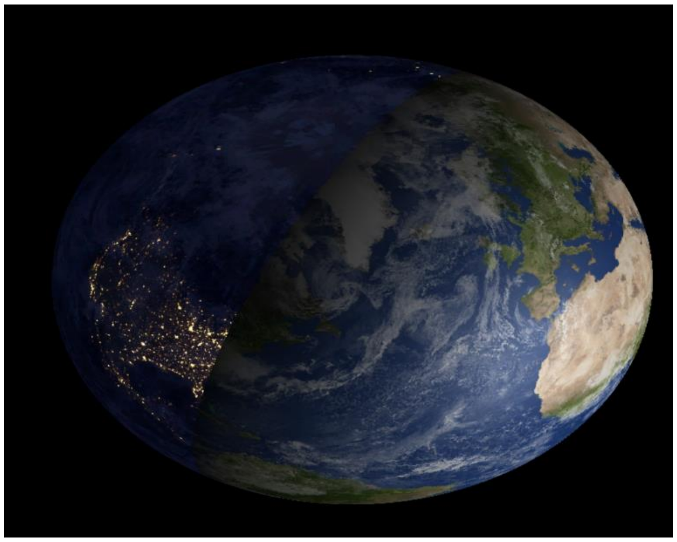
Errors/ Work Arounds:

None.

<u>Project 4B: Multi-Texturing</u>

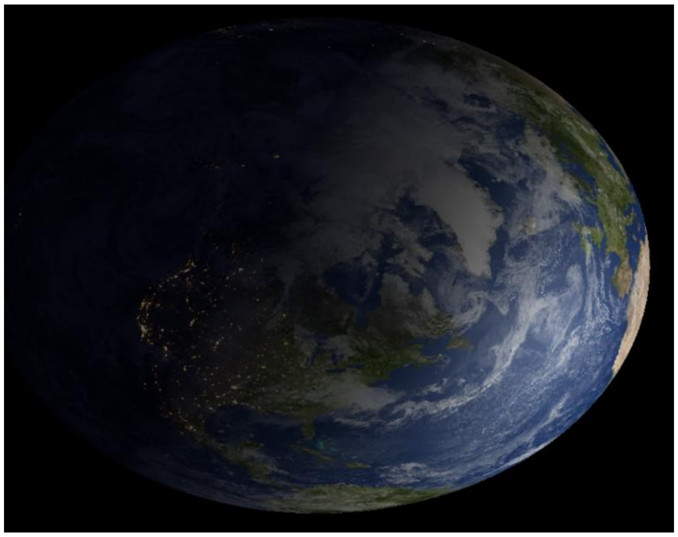
About:

This project contains, in my opinion, the neatest addition to the entire solution. While admiring my handy-work up to this point I conjured up a neat idea: Add city lights to the dark side of the earth! I found the appropriate textures and spent the next several hours trying to load them both into the fragment shader. (openGL apparently supports up to 32 textures to be loaded into the same shader at a time.)



Result:

I made simple modifications to the shaders and got the result as shown to the upper-right. This worked out nicely, but the sudden transition from night to day wasn't realistic. To get around this, I used a form of interpolation within the fragment shader. Given a fragments current color on the night side, find out what color it will be on the day side. The difference of these two color vectors is then interpolated via percentage of how close it is to a specific day longitude. This allows a smooth transition between night and day with the city lights slowly fading in/out!



Errors/Work Arounds:

Loading two textures into the shaders proved more difficult than it should have had been. My procedure was correct, but a simple index error in a for-loop caused the same texture to be loaded twice! I, of course, mistook this to mean that the second texture wasn't recognized by the shader. Everything was smooth sailing after this error was corrected.

Project 5: Multi-Models and Cameras

About:

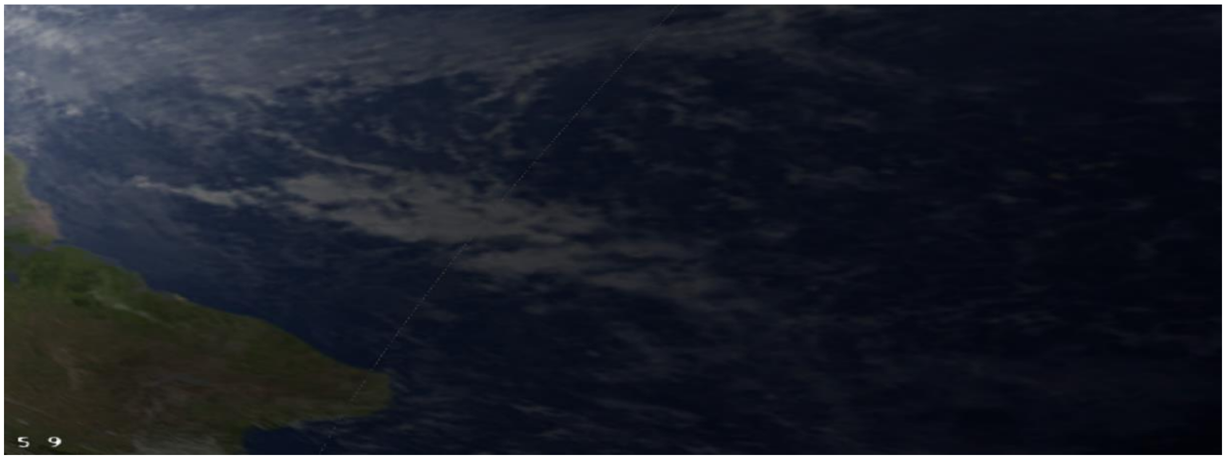
The Sun, the Moon, and a cubesat were added to the scene. Warbird Simulation classes were used as a template to create the Objec3D and CelestialSolid3D classes used in this solution. I did some research about orbit and rotation speeds and scaled the values appropriately. I considered making a camera class, but ultimately decided to hard code the different camera views. These cameras are set to either chase an object or it can be set in free-view mode. These camera views are in no-wise stationary static.

Result:

As seen in image.

Errors/Work Arounds:

After scaling the distances from earth to moon, and earth to sun, a white streak appeared upon the surface of the earth. I spent a few hours trying to figure out what was causing this, but the

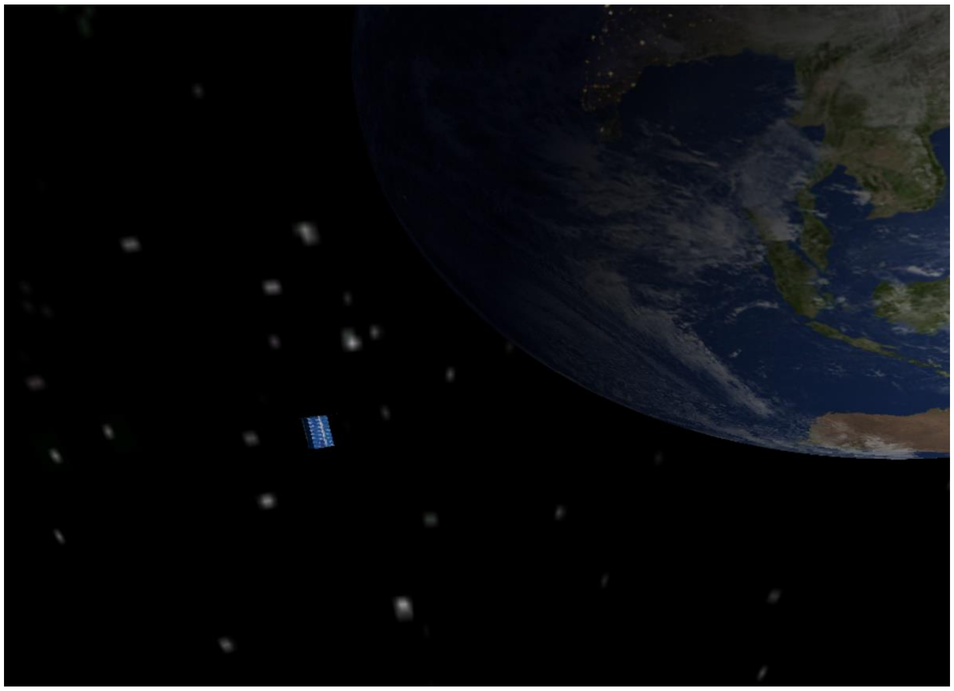


problem was eventually resolved by lowering the interpolation space, as mentioned in project 4B, from the value 0.2 to 0.1. (These values are dependent upon the cos(angle) between the vertices' normal and the light source.)

<u>Project 5B: Star Field</u>

About:

Warbird Simulator is "contained" within a box, with each face of the box having a different texture. I wanted to add a background texture to the scene, but unlike the simulator, I didn't want visible edges. I debated a while between implementing a sky-box or a sky-sphere. The sky-box would require much less resources than the sphere based upon the minimal number of vertices needed to generate it. The sky-sphere, on the other hand, would more easily avoid visible edges and seems. I ended up creating a sky-sphere. I read a "tutorial" on stackOverflow which gave the proper way to render a sky-box, which I applied to my sky-sphere. The sphere is drawn before anything else, the depth-buffer is turned off, and only camera rotation is taken into account. This allows the sky-sphere to be drawn behind all the other objects, and no matter where the view camera travels we will always remain in the center of the sphere.



Result:

As seen in image.

Errors/Work Arounds:

The initial star field texture was somewhat blurry, probably due to over-stretching the image. To fix this, I modified the background's fragment shader to dim the texture's color. I later changed my approach and just used GIMP to decrease the brightness and increase the contrast of the texture.

Project 6: Shadow Mapping

About:

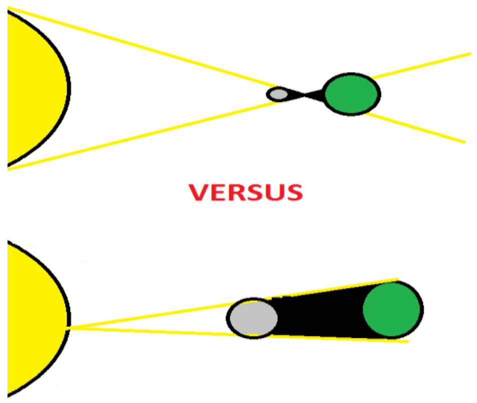
The final add-on to this project is shadow mapping. The goal is to allow the moon to cast a shadow unto the earth, and vice-versa. Shadow mapping is done by rendering the scene twice. The first time we render it from the point of perspective of the light's position. The information stored in the depth buffer will indicate which fragments are closer than others. This information can then be used to map shadows in the fragment shader when we render the scene regularly for the second render pass.

Result:

This was certainly a more challenging effect to produce than the previous effects already added. I was successfully able to generate a shadow map and map it to the earth. There is, however, a major problem which I will discuss in the next section. As expected, the max FPS setting only reached about 50% of what it would reach before. (Due to the double rendering cycle.)

Errors/Work Arounds:

Turns out that in order to get shadow mapping to work properly with my project, I'd probably have to render the scene more times than twice, or I'd have to try a different technique all together. If you run the program you'll notice that the projected shadow s much too big. I tried fixing this via the viewing frustum, but I then came to an understanding of why this was happening. The light's source is set to be the center of the sun. This is unrealistic, as in reality the light source is the entire sun, in particular, the top and bottom visible edges from earth. The moon's shadow should be projected in an elongated x-like manner, but in this case it was projected in a v-like manner. I thought of some possible fixes, including: a.) Triple-pass render with two light sources, one on top of sun and one on bottom. b.) Pass in Moon's position and determine if it's between fragment and sun location. If it is, draw a shadow depending on a minimum angle requirement.



I did not take the time to render either of these approaches. Maybe next time.

Summary:

The long hours have certainly paid off. Going through the tutorials and applying the material has been a great learning experience and has brought a sense of personal accomplishment. I encourage the reader to more fully study out the actual code as only a select few of the many learning experiences have been shared in this report.

References:

"11.) Multitexturing - OpenGL 3.3 - Tutorials - Megabyte Softworks." *11.) Multitexturing - OpenGL 3.3 - Tutorials - Megabyte Softworks*. N.p., n.d. Web. 16 Apr. 2015.

"How to Proper Position Skybox Camera Using OpenGL." *C++*. N.p., n.d. Web. 16 Apr. 2015.

"Opengl-tutorial.org | Tutorials for Modern OpenGL (3.3+)." *Opengltutorialorg*. N.p., n.d. Web. 16 Apr. 2015.

"Shadow Mapping." - *VRwiki*. N.p., n.d. Web. 16 Apr. 2015.

Shreiner, Dave. *OpenGL Programming Guide: The Official Guide to Learning OpenGL, Versions 4.3*. Boston, Mass: Addison-Wesley, 2011. Print.