Adjacency Maze 2.0

Daniel Meyer

Professor Tong Yu

CSE 520, Spring 2019

Objective

The objective for this project was to demonstrate advanced knowledge of OpenGL and GLSL and create a demonstration. My demonstration is an evolution of my OpenGL demonstration. This new version includes the use of GLSL shaders and texturing to provide a much higher level of visual fidelity compared to the previous version.

Abstract

With the addition of GLSL shader as well as textures came many new files for the project as well as a major overhaul to the packaging of the project. The first set of the new files are *.vert* and *.frag* shader files. There are four pairs of these files each being used for a different graphical function: skybox for the skybox cube map textures, squareTex for the floor and ceiling textures, brick for the walls, and lastly objTex for all other maze object textures. Next is the inclusion of Shader.cpp and Cube.cpp with their respective header files. Shader.cpp handles all shader program initialization, creation, and clean up for each of the GLSL shaders used in the project. Cube.cpp handles the creation of a cube map that represents the skybox for the maze. Another major change to the project was the inclusion of OpenGL Extension Wrangler or GLEW which was used in conjunction with the GLSL shaders. Lastly there is also the inclusion of the *imageio* library for PNG file support for texturing use.

Introduction

The base for this project is my Adjacency Maze used last quarter that was originally adapted from Java into C++ and then given a graphical representation with OpenGL. The original graphical representation was very basic with most objects being a solid color and utilizing the lighting model included with OpenGL. To update the maze, I utilized the OpenGL Shading Language (GLSL) to add texturing to the objects and improve the quality of the lighting. Adding shaders to the program led to an increased complexity by magnitudes which made the jump in overall progress much slower as compared to the first iteration of the maze, however with the amount of progress made the end-result betrays the number of systems added. The final product is vastly improved over its first iteration.

Package

AdjacencyMaze2_Final can be divided into three categories: source and header files, shader files, and texture files. The source and header files include both old and new additions. Of the existing files Node.h was reused in its entirety whereas Maze.cpp was brought over as well albeit with heavy modifications. Most of the modifications made to Maze.cpp were in the draw functions whereas all the maze algorithm code was left untouched. In Maze.cpp there are now #include for imageio.h and glew.h as well as a GLEW_STATIC 1 definition. These are added to allow for the use of GLSL shaders as well as texture mapping.

The changes made to the existing draw functions were different based on each type of object being drawn. For the drawFloor() function I replaced the glNormal calls with glTexCoord2f to create the texture coordinates for the corresponding vertices. Another addition to the function is the inclusion of init2DTexture() to initialize the texture map shader and floor textures. The functionality of init2DTexture will be discussed further at a later point in this section. The drawPath() and drawExit() functions received the most changes with code added for texture generation for sphere maps and linear maps. This code was needed as glutSolidSphere() and glutSolidTorus() do not have built-in texture coordinates and thus was necessary to generate the texture coordinates. Next the drawBarriers() function received changes with the barriers no longer being glutSolidCone(), but instead gluCylinder() with a gluDisk() on the top and bottom. There is also the creation of a gluQuadric() to hold gluQuadricTexture() to hold the texture coordinates of the barrier objects. This function also calls init2DTexture() to initialize the shaders and texture for the barriers. Finally the drawWalls() function is changed to call the shaderInit() function initialize the brick shader to apply a brick shader to the solid color maze walls. The two key changes made to Maze.cpp are init2DTexture() and shaderInit(). These two functions are what enable the GLSL shader and texture mapping functionality. Both functions are similar in that they both initialize a shader based on the vertex and fragment shader file name parameters. They do have a few key differences around which shader uniforms are used. Each ahs their own set of uniforms to work on and init2DTexture() also has parameter for the texture file (PNG) to use.

Once the shaders are initialized, depending on the shader being used, different effects will occur. Each of the four pairs of shaders has a different use. The first three shaders, squareTex, objTex, and skybox are all similar in that their core function is to apply a texture. Each handles the textures differently however. The floor and ceiling shader, squareTex, applies a texture without lighting by ignoring lightIntensity (which was calculated in the vertex shader) in the fragment shader and instead simply choosing to apply the texture as it is. The barriers' shader, objTex, applies a texture with appropriate lighting using lightIntensity, calculated in the vertex shader and applied in the fragment shader, applied to the textures. The final texture shader, skybox, doesn't require any form of lighting and thus passes the texture coordinates through the vertex shader and onto the fragment shader. The last of the textures, brick, was supplied by Dr. Tong Yu and utilizes the vertex shader to calculate lightIntesity which is then passed to the fragment shader where shaders functions fract, step, and mix are used to create a brick texture using the fragment shader itself instead of an image.

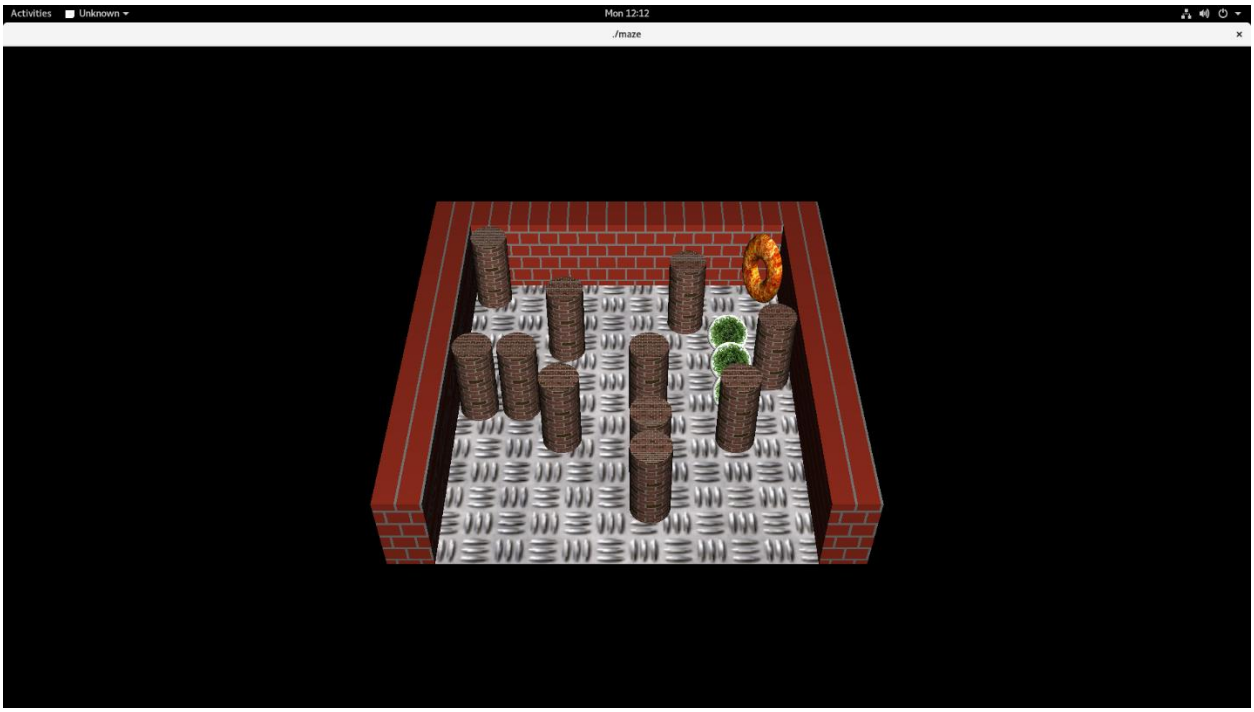
Each object in the maze, with the exception of the walls, pulls from different PNG files in the project folder to use for texture mapping. Each PNG was sourced from the internet via Google Searches and kept under 500kb for performance and storage. I did find with one texture that using an image greater than 1,000kb did lead to some performance issues. One issue I did have with these textures was a flickering caused by using a single buffer instead of double buffering. The solution was to enable double buffering which stabilized the images. To further emphasize performance each image is set to wrap, repeat, and using GL_NEAREST for the mipmapping.

Interface

User interaction is handled through a series of keyboard inputs as follows:

- ‘up arrow’ = rotate up (counter-clockwise about the x-axis)
- ‘down arrow’ = rotate down (clockwise about the x-axis)
- ‘left arrow’ = rotate left (clockwise about the z-axis)
- ‘right arrow’ = rotate right (counter-clockwise about the z-axis)
- ‘w’ = translate north (+ y-axis)
- ‘a’ = translate west (- x-axis)
- ‘s’ = translate south (- y-axis)
- ‘d’ = translate east (+ x-axis)
- ‘e’ = translate up (+ z-axis)
- ‘q’ = translate down (- z-axis)

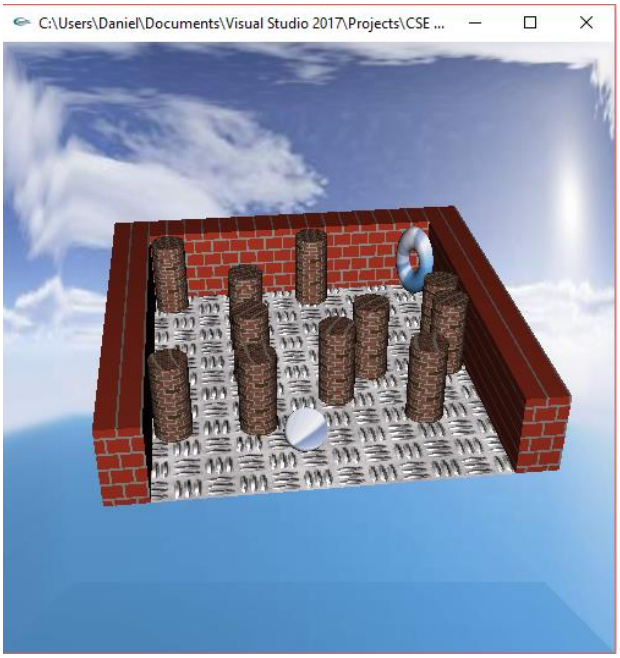
Screenshots



Default View

Zoomed View

Skybox Disabled



Skybox Enabled

Conclusion

The result of version two of this project is vastly superior to version one. In experimenting with the OpenGL Shading Language and importing external data I was able to expand my knowledge of graphics programming. I became more familiar with the mathematics behind graphics programming, specifically the math behind lighting, as well as how data is passed to shaders. Given more time there are several areas I would like to improve upon, many of which were goals I had from the start of the project but was unable to implement due to time constraints. The first is to correct the skybox to work on the school's Linux machines. At home in Visual Studio the skybox worked correctly, however on the Linux machines the transformations were completely incorrect for all maze objects. Next, I would have liked to implement shadowing. This would have added an extra layer of depth and visual fidelity to the maze by "grounding" objects. Lastly, I would have liked to make the exit object be a reflective environment map to add an extra visual element. Overall, I thoroughly enjoyed the experience and was happy to be able to revisit my project from last quarter and update it to what it is now.

References

"The OpenGL Extension Wrangler Library." *GLEW*, glew.sourceforge.net/.

Khronos Group. (n.d.). The Industry's Foundation for High Performance Graphics. Retrieved from https://www.opengl.org/

Khronos Group. (n.d.). The Industry's Foundation for High Performance Graphics. Retrieved from https://www.opengl.org/resources/libraries/glut/

Yu, T. L., Ph. D. (n.d.). CSE 420 Computer Graphics. Retrieved from http://cse.csusb.edu/tongyu/courses/cs420/index.php

Yu, T. L., Ph. D. (n.d.). CSE 520 Computer Graphics. Retrieved from http://cse.csusb.edu/tongyu/courses/cs520/index.php