Comp 371- Computer Graphics Project: Rubik's Cube

Concordia University Dept. of Computer Science & Software Engineering

Date: Sunday August 20th, 2020

"We certify that this submission is the original work of members of the group and meets the Faculty's Expectations of Originality"

Signed and Submitted by:

Noor Alali (26797180)
Umaid Malik (27576110)
Vishal Senewiratne (40003989)

Introduction

At the start, our team was formed very late compared to others. We also started communicating with each other a few weeks after the project was posted. However, we were able to establish a central means of communication and used Github for version control and source code management. Most of us had to learn opengl from the lectures provided and from “learnopengl.com” [1]. Setting up the project and linking it to the opengl libraries was the first hurdle that we had to go through. However, with the help of Youtube and “learnopengl.com” [1], we were able to accomplish this in a timely manner.

During the first iterations, we were able to complete the tasks with good timing, but we did not assign tasks to each other. Therefore, we had some trouble knowing what was left to do and who did what. The second part of the project was better because we started by assigning tasks to each person. We had some trouble implementing some features, but we had better communication. The final iteration was tough because we were brought down to a team of 3 people. We tried assigning tasks, but the bulk of the work was done by Umaid, Vishal worked on the report and Noor helped with some tasks.

Methodology

Starting with the first iteration of the project, we had to build models using the third letter of our first name and fourth letter of our student identification number. Therefore, we started by following the setup shown on “learnopengl” [1]. In order to get the window and an OpenGL context to render in, we used the GLFW library. The

window object was created, and its dimensions were given along with the name of the window.

```
glfwInit();
glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR, 3);
glfwWindowHint(GLFW_CONTEXT_VERSION_MINOR, 3);
glfwWindowHint(GLFW_OPENGL_PROFILE, GLFW_OPENGL_CORE_PROFILE);
//glfwWindowHint(GLFW_OPENGL_FORWARD_COMPAT, GL_TURE);
glfwWindowHint(GL_DEPTH_TEST, 24);

GLFWwindow* window = glfwCreateWindow(1024, 768, "COMP 371 - Team 6 - Assignment 3", NULL, NULL);
```

After this the unit cube was built by using triangles and following the graphics pipeline. Thirty-six vertices and twelve triangles are stored in a vertex array that helps create the unit cube. Our vertex shader and fragment shader are both in main at this point. A different function is made to compile and link the shader programs.

```
// link shaders
GLuint shaderProgram;
shaderProgram = glCreateProgram();

glAttachShader(shaderProgram, vertexShader);
glAttachShader(shaderProgram, fragmentShader);
glLinkProgram(shaderProgram);
```

In order to create each model, we set initial model parameters and then used the appropriate matrix transformation to get the shape of it. After each model was created, we reset the world matrix, rotation matrix and model scaling matrix to the identity matrix to avoid unwanted transformations for the next model.

The second iteration was focused on lighting and texture mapping. We had to enclose the models in a sphere, but we had trouble implementing this. For texture mapping, we had to load the uv coordinates into the buffer, load the textures from

files and put it in buffers and get the textures into the fragment shader. Details about this process are mentioned below.

In relation to texture mapping, we use glGenTextures to get the buffer id, then we bind the buffer. We use GL_LINEAR in order to achieve a smoother pattern.

```
textureID = 0;
glGenTextures(1, &textureID);
assert(textureID != 0);

glBindTexture(GL_TEXTURE_2D, textureID);

glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);


int width, height, nrChannels;
unsigned char* data = stbi_load(textureFilePath, &width, &height, &nrChannels, 0);
```

In the vertex shader, we have two variables, aUV, which is a texture coordinate from OpenGL and vertexUV, which is an interpolated version that is going to the fragment shader. We simply do an assignment from one to the other.

```
vertexUV = aUV;
```

In the fragment shader, we declare a uniform sampler2D that takes in the texture and the texture coordinate, which is also multiplied by a vec4 that determines the intensity of the light. If there is no texture applied, only the lighting is taken into consideration for the pixel.

```
if (drawTexture)
        FragResult = texture(texture_1, vertexUV) * vec4(phongModel, 1.0);
else
        FragResult = vec4(phongModel, 1.0);
```

The final iteration of this project included different options that we had to pick from. After carefully reviewing each topic, we decided to go ahead with the Rubik’s cube

because it seemed simple enough to implement and we would easily be able to make improvisations to it. It also seemed the most feasible given that we have been brought down to a team of three people.

We started off by building the Rubik's cube itself. It is composed of 26 individual cubes which are texture mapped accordingly. The middle cube of the Rubik's cube was not added because it is not visible at any time.

```
// cube 1
partMatrix = cube_rotation_X_3 * cube_rotation_Y_1 * cube_rotation_Z_1 * translate(glm::mat4(1.0f), glm::vec3(0.05f, 0.05f, 0.05f));
worldMatrix = worldOrientationMatrix * modelTranslationMatrix * modelScalingMatrix * modelRotationMatrix * partMatrix;
texturedShaderProgram.setMat4("worldMatrix", worldMatrix);

glBindTexture(GL_TEXTURE_2D, shape_A_9.getTextureID());
texturedShaderProgram.setInt("texture_1", 0);
glDrawArrays(GL_TRIANGLES, 30, 6);

glBindTexture(GL_TEXTURE_2D, color_3.getTextureID());
texturedShaderProgram.setInt("texture_1", 0);
glDrawArrays(GL_TRIANGLES, 12, 6);

glBindTexture(GL_TEXTURE_2D, science_1.getTextureID());
texturedShaderProgram.setInt("texture_1", 0);
glDrawArrays(GL_TRIANGLES, 6, 6);
```

Then a 5 digit timer was created that runs in seconds from 0-99999 in order to keep track of time.

```
// draw timer
modelTranslationMatrix = translate(glm::mat4(1.0f), glm::vec3(0.1f, 4.0f, 2.0f));
worldMatrix = worldOrientationMatrix;
texturedShaderProgram.setMat4("worldMatrix", worldMatrix);

int time[5];
time[0] = glfwGetTime();
time[1] = glfwGetTime() / 10.0f;
time[2] = glfwGetTime() / 100.0f;
time[3] = glfwGetTime() / 1000.0f;
time[4] = glfwGetTime() / 10000.0f;
```

A skybox has been added as well, where we have 6 texture locations for each side of the box. The source of these textures were free to use, as mentioned.

```
// draw skybox
drawSkybox(texturedShaderProgram, skybox_front, skybox_right, skybox_back, skybox_left, skybox_bottom, skybox_top);
```

Design and Implementation

All of the tasks are implemented in one main file. Therefore, we shall separate segments of code in this file and highlight their main functions.

MAIN{

Setup window

Compile and link shaders

Load textures

Create vertices and colors

Setup texture and framebuffer for creating shadow map

Setup grid lines and axis lines

Initialize initial model parameters

RENDER LOOP{

Render shadows

Call model drawing functions with initial model parameters and textures

Draw light cube with textures

Draw Rubik’s cube with textures

Render scene

Draw timer

Draw skybox

}

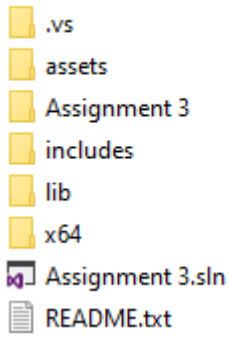
```
}
```

Process all inputs

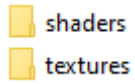
Model drawing functions with textures

Draw grid lines and axis lines

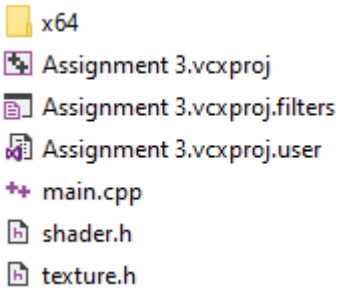
In terms of the directory of the project, each iteration has a separate folder that is similar. Therefore, we shall just display the directories of the last part of this project. We have folders for the assets, the project, all our ‘includes’ files, “lib” files, sln file and the “README.txt” file.



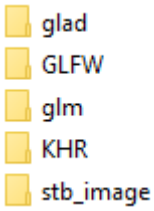
The assets folder include a shader folder and a texture folder. The shader folder contains vertex and fragment shaders for textures and shadows. The texture folder contains all the image files that are used to texturize models in the program.



The “Assignment 3” folder contains all the project files, including the main.cpp and header files created within the project.



The “includes” folder contains different libraries that we have used for OpenGL compatibility.



The “lib” folder only contains the glfw3.lib file, which must be linked to the project to use GLFW.

Deployment

Once the program has been compiled and rendered, you will observe a grid plane with 5 models in a skybox that is textured with the night sky and far off mountains. There is a Rubik’s cube floating a bit higher up as well. In order to move the camera, while holding the right mouse button, use the ‘W’, ‘S’, ‘A’, ‘D’ keys and then in order to move in the positive and negative Y direction, use the spacebar and ‘Z’ accordingly. ‘SHIFT’ will increase the movement speed.

In order to control the models, you have to select one of them with either ‘1’, ‘2’, ‘3’, ‘4’, or ‘5’ and then you can use the same directional inputs as the camera without holding the right mouse button. Press ‘R’ to reset model to initial position, ‘SHIFT’ + ‘R’ to reset model to initial size, ‘U” to upscale model, ‘J’ to downscale model, ‘T’ or

‘Y’ to rotate model in ‘x’ direction, ‘G’ or ‘H’ to rotate model in ‘y’ direction, ‘B’ or ‘N’ to rotate model in ‘z’ direction

The Rubik’s cube is controlled by layers. To rotate the first layer, use ‘Q’ or ‘A’ for the ‘x’ direction, ‘R’ or ‘F’ for the ‘y’ direction and ‘U’ or ‘J’ for the ‘z’ direction. For the second layer rotation, use ‘W’ or ‘S’ for the ‘x’ direction, ‘T’ or ‘G’ for the ‘y’ direction and ‘I’ or ‘K’ for the ‘z’ direction. Finally, for the third layer, use ‘E’ or ‘D’ for theIn ‘x’ direction, ‘Y’ or ‘H’ for the ‘y’ direction and ‘O’ or ‘L’ for the ‘z’ direction

To reset the timer, input ‘0’. ‘X’ will toggle the textures, ‘E’ will display a perspective projection, ‘V’ will display an orthographic projection, ‘B’ will render triangles, ‘N’ will render line, ‘M’ will render points, hold ‘C’ to disable backface culling, ‘HOME’ will return you to the initial position and mouse scroll will zoom in and out.

Conclusion

In conclusion, this project came with a lot of trials that we had to overcome. Some of us have never programmed in C++ and have not touched any OpenGL. There was certainly a big learning curve in the beginning, but after that we were able to figure things out slowly but surely with different sources on the internet and with the class lectures. Being a team of 3 people has also affected our efficiency. It has taught us about time management, code optimization, communication, and resourcefulness.

Most of the tasks were able to be completed in time for submissions. However, we did have trouble implementing the spheres around the models and rotating the Rubik's cube. We also have some functions that take a lot of overhead, for example, the skybox that constantly changes position as we move to keep the same background distance. To improve on our design, we should have modularized our code into different classes and subclasses to improve clarity. On the other hand, our program has good movement and model scaling. The timer has also been implemented concisely with a few lines of code.

Overall, this has been a good learning experience in computer graphics for our team, we should be able to create better graphics programs efficiently with our newfound knowledge.

References

[1] Joey de Vries. Learn OpenGL, June 2014. https://learnopengl.com/.

[2] http://www.opengl-tutorial.org/

Textures:

Rubik’s cube faces:

Hollow_knight_art.png

[3] Ari Gibson et al. *Hollow Knight*, February 2017. http://www.hollowknight.com/

color.png
[4] Made using GIMP

shape_A.png
[5] Troyka. Spiral movement and rotation. Design elements, Stock vector. August 2011.https://www.colourbox.com/vector/spiral-movement-and-rotation-design-elements-vector-2269950

shape_B.png
[6] Troyka.Design elements set. Decorative patterns in circle shape. Vector art, Stock vector. January 2014. https://www.colourbox.com/vector/patterns-in-circle-shape-design-elements-set-vector-8609309

Number.png
[7]
https://stock.adobe.com/ca/search?filters%5Bcontent_type%3Aphoto%5D=1&filters%5Bcontent_type%3Aillustration%5D=1&filters%5Bcontent_type%3Azip_vector%5D=1&filters%5Bcontent_type%3Avideo%5D=1&filters%5Bcontent_type%3Atemplate%5D=1&filters%5Bcontent_type%3A3d%5D=1&filters%5Binclude_stock_enterprise%5D=0&filters%5Bis_editorial%5D=0&filters%5Bfree_collection%5D=0&filters%5Bcontent_type%3Aimage%5D=1&k=digital+clock+font&order=relevance&safe_search=1&search_page=1&get_facets=0

shape_C.png
[8]
https://www.wallpaperflare.com/blue-geometric-shape-wallpaper-abstraction-dark-geometry-figure-wallpaper-qthqg

Science.png
[9]

James Thew. Science Vector Icon Set. A collection of gold science themed line icons including a atom, chemistry symbols and equipment. *Alamy*, August 2015. https://www.alamy.com/stock-photo-science-vector-icon-set-a-collection-of-gold-science-themed-line-icons-87960526.html

Skybox.png
[10]
https://www.cleanpng.com/png-cube-mapping-night-sky-star-counter-strike-1-6-sky-5778000/download-png.html