COMP371 Final Project Report

Liam-Thomas Flynn - 40034877

Sharon Chee Yin Ho - 40044575

Anissa Kouki - 40032516

Mélina Phan - 40094159

Keven Abellard - 40008652

This particular project was interesting to us because the simplicity of a Rubik's cube allowed us to focus on textures, animation, shadows, etc., without spending time making an extremely complex model or world. The other suggested projects required changing the camera more, or creating completely new complex models, and we felt that spending time on those things would take away from what we really wanted to learn more in-depth about.

Our objective going into this project was to further develop our familiarity with things seen in previous assignments. For example, hierarchical modelling, animation, and texture mapping. We would also expand our knowledge of OpenGL by working with different libraries.

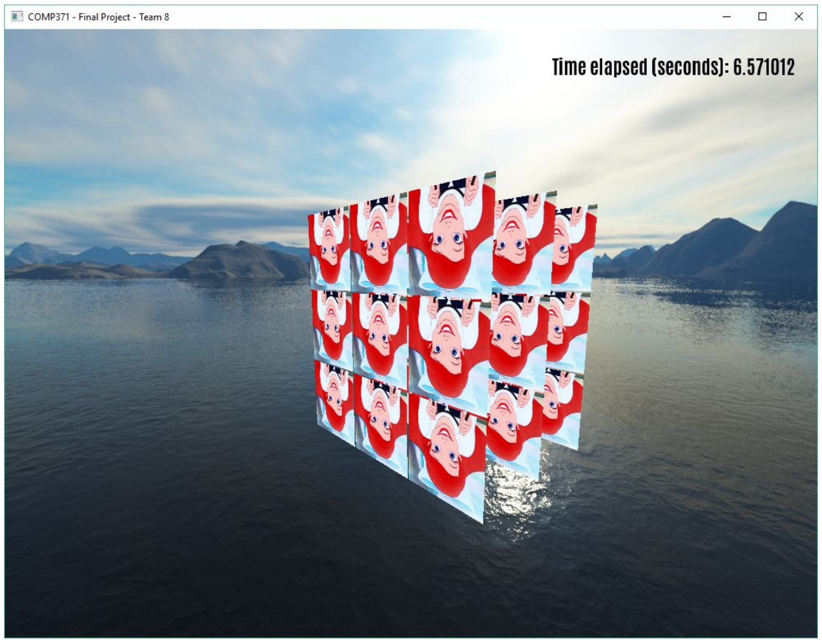
This project explicitly asked for the addition of a timer, which we considered to be an interesting topic to work with since we haven't explicitly used third party libraries for our previous assignments. Because we were going to play around with libraries, we also decided to implement background music and sound effects in our project, as proposed in the project instructions. With this functionality, our original concept was to include a "hint" button that would help the user identify what the theme for a particular face was in order to help them solve the Rubik's cube puzzle. We initially planned on having 6 different textures, each their own theme, and each texture split into 9 smaller components, so they would cover each face of the Rubik's cube entirely. Thus, when shuffled, it would be difficult for the user to tell what the original picture is. The initial "hint" key concept to go with this would be a sound and texture toggle, it would play the hint sound corresponding to a particular face, and show the user what that solved face should look like. We also wanted to have a feature that would be able to tell when the Rubik's cube is solved. The concepts we ended up successfully implementing will be covered later in this report.

Since there were still a few functionalities we could not fix for previous assignments, we really wanted to focus on those and make this final project extra special. We planned on fixing up a few bugs like implementing shadows correctly and having smooth animations for every move. Because none of us were able to create a functional skybox in Quiz 2, we wanted to make sure to have one for this project.

There were several things that led to the accomplishment of most of our objectives. For one, throughout the first and second programming assignments, we were able to develop a sense of the team and how to best work together when tasks are generally separated. We decided to meet often, split work into smaller chunks, check in with each other often through chat when not available for calls, and merge our branches more often. Meeting and checking in with each other often meant that we were able to switch tasks around and ask for help frequently. Splitting work into smaller tasks lightened the burden on each individual team member. Finally, merging often saved us the trouble of having a bunch of conflicts to resolve on the last day, as well as making sure that tasks started later in the project would have a base that was more complete and functional than if the task had branched off of some other non-master branch.

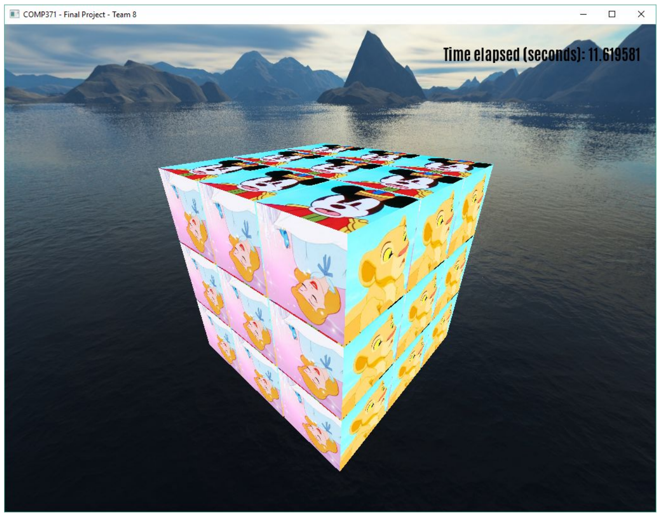
Another thing we did was reuse any and all resources we could from the first two programming assignments. A lot of the work done in the past could fairly easily be repurposed, fixed, or cleaned up for this project, and doing so saved us a lot of trouble. Especially at the beginning of the project, when starting from scratch can make things seem very daunting.

Every feature required a lot of research and practice, some more than others. At first, passing a vector or an array of texture ids proved to be troublesome. There was a lot of time spent on reading OpenGL documentation to understand what each thing does and how they work. Once we had a better grasp of shaders and textures, we tackled texturing the sides of each cubie individually. After some more research, we decided to set up an array of “first” UV coordinates that contains the first texture coordinates of each side of the cubies. That way, we were able to use glDrawArraysInstanced to draw different textures per side.


COMP371 - Final Project - Team 8
Time elapsed (seconds): 6.571012

First set of vertices displayed with their corresponding texture

When all the vertices are drawn, the Rubik's cube is complete:


COMP371 - Final Project - Team 8
Time elapsed (seconds): 11.619581

A complete Rubik's cube

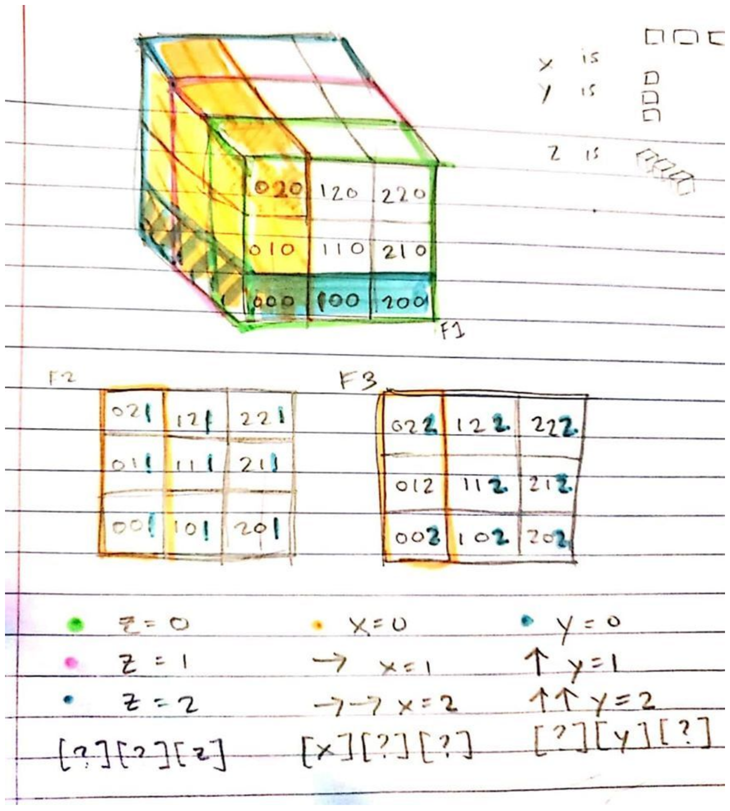
In other words, each side has a specific texture bound to it and it is printed during the setup of the Rubik's cube.

One of the project requirements was to implement lighting and shadows. For the lighting, we decided to implement the light using the Phong lighting model. While we discussed (and implemented) having the light follow the player (like a flashlight), we decided to keep it fixed, to better display the resulting shadows.

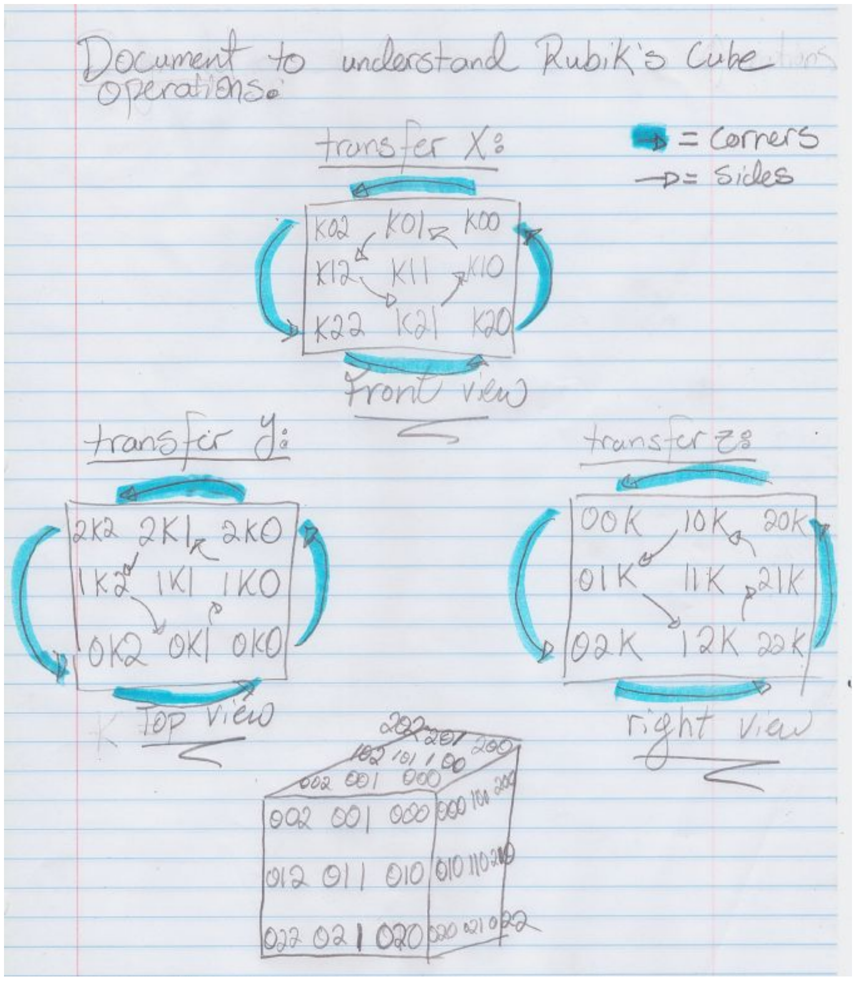
For the shadow themselves, we chose to implement them using the same technique in the previous assignment, shadow mapping using two passes. We used the tutorial from the LearnOpenGL website (https://learnopengl.com/Advanced-Lighting/Shadows/Shadow-Mapping) to assist us in this endeavor.

While we built on what we have learned during the previous assignment, we were not successfully able to implement proper shadows during the last project. Diving back into it for this project, we realized that the mistake was in the shadow mapping (first pass) of our implementation of the algorithm. The mistake was that we did not generate the shadow map correctly, and we didn't activate the correct texture once it was time to go through the second pass and render the actual scene. Once we fixed that mistake, we were able to generate the scene successfully in two passes and generate a working shadow, which albeit is a bit too large for the cube casting the shadow.

For ensuring a functional Rubik's cube, our transformations had to be perfect, or else the Rubik's cube would no longer be solvable. To have a better grasp of Rubik's cube transformations, we had to draw a diagram of what each transformation does.



Rubik's cube and understanding of how the 3D array works



Rubik's transformations per side (x, y, or z = 0, 1, or 2)

Every time we make a move, whichever side we decide to move has its cubies rotated clockwise (x, z) or counterclockwise (y) by 90 degrees. We then make sure that the program knows where each cubie is now situated at. The diagrams were also helpful for other teammates to understand these transformations.

For the skybox, we simply re-used the code from LearnOpenGL on Cube mapping (https://learnopengl.com/Advanced-OpenGL/Cubemaps). This allowed us to create a big box that surrounds our Rubik’s cube with different textures mapped to each side. The textures were cropped in a way to give the box the appearance of a seamless sky.

An issue we had with our animations in previous assignments was ensuring that the user couldn’t interrupt them. To give our animations time to smoothly complete, we disabled keyboard inputs for the duration of them. This was accomplished through the use of the *glfwPollEvents* function in our game loop. For our Rubik’s cube, this was key to making sure that our shuffle method would properly animate each and every turn of the shuffle, before letting the user issue any further commands.

Beyond the librairies we started with, there were two more that we made use of for this project: irrKlang, and freetype. IrrKlang is a library used to incorporate sounds and music into the project. Its primary use was the play2D method which plays sounds at the file path we specify; and this either only once, like our sound effects, or in a loop, like our background music. The freetype library is one that allows us to display the time elapsed. Without this library, while we are able to keep track of the time, we are unable to render text that would display this data.

The objectives we set for ourselves that we were unable to meet for the deadline were texture spanning and checking if the Rubik’s cube was solved. The achievement of both of these objectives depended on how we came to get our texture mapping to work in the first place. The texture spanning objective would require a single texture or image, stretched to cover 9 faces on a single side of the Rubik’s cube. We found that the logic we used to map a single texture so it would tile over 9 faces of a cube’s side was not something that could be easily modified to accommodate for a single larger texture split into 9 smaller representations.

Similarly, the logic we used for deciding which textures would be mapped to what faces would not adapt to checking if the correct textures were all side-by-side on their current face. One alternative solution we thought of would take advantage of the 3D matrix we used to keep track of our transformations. We would simply compare the current state of the matrix to the state of a *solved* Rubik's cube. Unfortunately, this would end up either requiring 24 solution matrices, or requiring a function that could traverse our transformation matrix face by face and check the position of the cubes. Both of these methods, unfortunately, came up on the last day of work, and thus were unable to be added to the project.

Throughout this report, we have gone over several of the topics where we learned more, compared to previous assignments. These are, namely, the following: how to handle animations, using shaders and textures together, making use of documentation (both in and outside of the code itself), how to implement a skybox, as well as choosing third party libraries and how to take advantage of them to improve our final result. One additional topic we would like to note is the choice of themes for each cube. We found that choosing themes that we were already invested in made the task of finding textures and sounds for each cube much more enjoyable, and kept us motivated through what might have otherwise been incredibly dull.

Below are the references and resources that we made use of for our project.

Code References:

- Framework: Labs & PA1/PA2
- Skybox: https://learnopengl.com/Advanced-OpenGL/Cubemaps
- Camera: https://learnopengl.com/Getting-started/Camera
- Shadows and Lights:

  https://learnopengl.com/Advanced-Lighting/Shadows/Shadow-Mapping
- Text Rendering: https://learnopengl.com/In-Practice/Text-Rendering

Libraries:

- Audio and Sounds: https://www.ambiera.com/irrklang/
- Text: https://www.freetype.org/