Concordia University Department of Computer Science and Software Engineering

**COMP371: Computer Graphics**

**Design Document for Team 3:**

Members:

Antoine Dufour-Blain, 21308130, antoine.dufourblain@gmail.com

Kennedy Ballard, 40132850, kennedyjballard@gmail.com

Samuel Lambert-Senécal, 40112310, s\_lambert-senecal@hotmail.fr

Ana Lucio, 40032548, catalan.lucio@hotmail.com

Thomas Rao, 40097357, [thomas.rao@outlook.com](mailto:thomas.rao@outlook.com)

**Rubik’s cube**

1. **Introduction**

The following document outlines the progress and requirements for the final assignment submitted for the course COMP 371 Computer Graphics, to dr. Serguei A. Mokhov and Jonathan Llewellyn. Its current version was completed on April 28, 2021. All team members provided substantial contribution to this project and its previous iterations (i.e., Project Assignments 1 & 2) as evidence by the progress on the BitBucket Repository.

1. **Concept**

The assignment is a playable “Rubik’s Cube” puzzle coded in C++ using openGL and associated libraries, such as GLEW, GLM, and GLFW. The code incorporates a rudimentary approach to gesture tracking, namely via Microsoft Kinect, as outlined below. It extends upon the OIGestureTracker class of the OpenISS library. The freetype library was used in order to instantiate a timer to keep track of the user’s completion speed.

1. **Requirements**

All required dependencies are included in the file, including any additional libraries, which can be found in the “ThirdParty” folder. In order to build the executable and run the program, and up to date version of “CMake” is necessary. It can be obtained at:

<https://cmake.org/download/>

The build instructions are then as follows:

On Command Line:

1. Download and Install CMake

2. Open a terminal

3. Run cmake:

cd <source\_folder>

cmake -S . -B <build\_folder>

cmake --build <build\_folder> --target install

CMake GUI:

1. Download and Install CMake

2. Open the CMake GUI

3. Set the `Where is the source code:` field to the source folder

4. Set the `Where to build the binaries:` field to different location (e.g.: source\_folder/build)

5. Click Configure

6. Click Generate

Windows:

Once that is created you can build the \*\*ALL\_BUILD\*\* project followed by building

the \*\*INSTALL\*\* project.

Linux:

In the terminal:

cd <build\_folder>

make install

Once built and installed, all the relevant files will be in the \*\*dist\*\* folder at the top level of the source folder.

Linux/macOS:

In the terminal type:

./comp371-w21-03

Windows:

Click on the \*\*Project\_Team\_3.exe\*\* from the File Explorer.

You should then be running the program as intended.

**4.Commands and inputs**

Keyboard keys for controlling the cube:

* Q and E rotate the top row of the cube left right respectively.
* C and B rotate the middle row of the cube left right respectively.
* A and D rotate the bottom row of the cube to the left right respectively.
* W and S rotate the left column of the cube up and down respectively.
* G and V rotate the middle column of the cube up and down respectively.
* R and F rotate the right column of the cube up and down respectively.
* N toggles on and off the OIDynamicNullGestureTracker inputs.
* 1,2,3,4,5 and 6 change the position of the camera to face a specific side of the cube.
* K scrambles the cube and starts the timer and stops the automatic inputs.
* Left shift and P pauses the timer.
* Left and right arrow positions camera to the left and right of the cube by a small amount.
* Up and down arrow positions camera to the right side of the cube by a small amount.

**5. Architecture and Techniques**

**5.1 OIDynamicNullGestureTracker**

The OIDynamicNullGestureTracker is an extension of the OINullGestureTracker class of the OpenISS library, which itself is an extension of the OIGestureTracker class. It overrides no methods from OINullGestureTracker class and has no additional members. It does, however, override the following virtual methods from OIGestureTracker: getHands, and getGestures. These methods are intended to return a vector of hand movements, in the case of the Dynamic null tracker, they operate as follows:

getHands returns a vector containing a single “handData” object which is randomized to either be off the screen (via the “HAND\_IS\_LOST” OIHandState) or be on the screen (via the “HAND\_IS\_TRACKING” OIHandState). If the hand is on the screen, its position is then randomized with x and y coordinates between -30 and 30. The z value of the hand never changes for these purposes.

getGestures returns a vector containing a single “gestureData” object which is always a completed “GESTURE\_CLICK” gesture. Its position is randomized as above on the x and y axes from -30 to 30. This could be extended to produce different gestures, however in order to maintain portability between NiTE Gesture Tracking libraries which detect only 3 different gestures, and NuiTrack Gesture Tracking libraries which offer a far more robust variety of gesture, we have opted to rely on the position of the gesture being performed rather than the nature of the specific gesture itself.

It is intended that the “handData” returned by getHands represents the left hand of the player, whereas the “gestureData” returned by getGestures represents their right hand. In our program, if the left hand is off the screen and a push is performed with the right hand, it will rotate the entire cube in the cardinal direction in which the gesture was located, as defined by the lines y=x and y=-x. If the hand is in the upper quadrant, the cube will rotate so that what was previously its bottom face will become its front face, and what was previously its front face will now be on top. The same is true of the 4 other cardinal directions. If the “left hand” is present when a push is performed, it will instead rotate one of the 4 “slices” of the cube based on the current front facing side. It will analyze the position of the left hand in the same way outlined for the position above in order to decide which slice to rotate, then will rotate it clockwise or counter clockwise based on the position of the “click” gesture, a vertical rotation will look at the y position of the click gesture and a horizontal rotation will look at its x position.

**5.2 Rubiks and Cubelet**

The Rubiks class is the object which contains the 27 “Cubelets” object forming the rubiks cube. These contain a variety of structs to maintain their data including the animation of the cubelet based on input from the user. When an input is requested, the “animation” struct is instantiated to count the time until the animation is completed. It will then render based on the information within this struct while using the computer clock as a timer. It will not respond to additional requests while the animation is ongoing. It will then confirm that it is in the current position. The Cubelets are all rendered with different images on each side, using multiple themes as requested, specifically cats, dogs, houses, colours, sports, and math.

**5.3 Timer**

A timer appears on the screen while the user is attempting to solve the cube. It starts when the cube is scrambled and can be stopped manually by using Lshift+P. It uses the freetype library and the arial font. It uses the RenderText function to render the timer in a headsup display that is available to the player.

**6. Challenges and Difficulties**

No one in our team had much or any C++ experience prior to taking this course. It has proven extremely challenging to simply understand the form of the language. Our initial attempts involved writing Java code in C++ which works some of the time but not always. Organizing our compilation units in implementation and declaration files was also something we had no experience with at the start of this process. The simple act of importing and linking libraries which is easy to dismiss as trivial for an experienced C++ programmer cost us days and many headaches.

Due to the remote nature of the class and the ongoing international health crisis at the time of the completion of this project, we never met each other in person. This also caused unforeseen organizational issues simply because as teammates we could have had disagreements about form and structure for the project, which is to say nothing on the emotional and psychological impact of the confinement imposed by the circumstances of the last year.

It proved to be challenging to adapt our code from the previous iterations of the project to fit the requirements of this one. A lot of refactoring was done in order to better fit CPP standards of which we were unaware. We also did not know how to utilize cmake as a building tool before the class and had to do a lot to make it work. Organizing the architecture of the cube and cubelets to perform the animations required was also as extremely draining and time-consuming process.

**7. Conclusion**

This project forced us to experiment with new software with which we were unfamiliar. We hope you will find it satisfactory as a demonstration of our accomplishments in this course.