A Puzzling Project: Computerized Rubik's Cube Generator and Manipulator

Jacob Anderson

December 13, 2016

1 Abstract

My project was to design a 3 dimensional Rubik's Cube that can be randomized by the computer and manipulated by the user. In addition, the program is able to generate cubes of varying sizes, eg. 2x2, 3x3, 5x5, etc. Lastly, the program has some ability to solve the Rubik's Cube by itself. I used JavaScript to write my program in, so that users can manipulate it from any web browser without needing to download any special software.

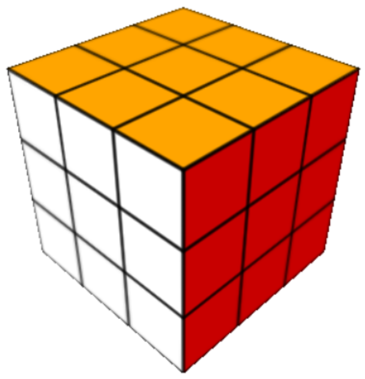

Figure 1: A Rubik's Cube

2 Background

There are two main reasons why I chose to create a virtual Rubik's Cube for my project. First, I have always been interested in puzzles, and the Rubik's Cube is a very well known puzzle that is recognizable for almost anybody. Second, since I already had experience in coding, I was seeking a challenge, and I had noticed that other students in the past have attempted to create virtual Rubik's Cubes, and to my knowledge have been completed in JavaScript. In addition, none have been attempted in sizes other than the standard 3x3. Therefore, I can build from the past Rubik's Cube projects while adapting them to

JavaScript and adding the additional functionality of different sized cubes.

3 About the Rubik's Cube

The Rubik's Cube is a 3D combination puzzle invented by Erno Rubik, a Hungarian sculptor and professor of architecture, in 1974 [1]. The traditional (3x3) Rubik's Cube consists of six faces, each one covered in nine stickers of a different color. An internal mechanism allows for individual faces to be rotated without the Cube falling apart. The goal of the puzzle is to take a *scrambled* Cube (i.e one in which the colors have been mixed up) and perform a series of rotations until each face is returned to having only one color. Since the invention of the traditional Rubik's Cube, many variations have been made; the most notable being Rubik's Cubes of different sizes, such as 2x2, 4x4, 5x5, etc. The traditional way to solve a Rubik's Cube is by holding it so that three faces are shown: the *left* face, the *right* face, and the *top* face, as shown in Figure 1. I have designed my program to maintain this positioning at all times.

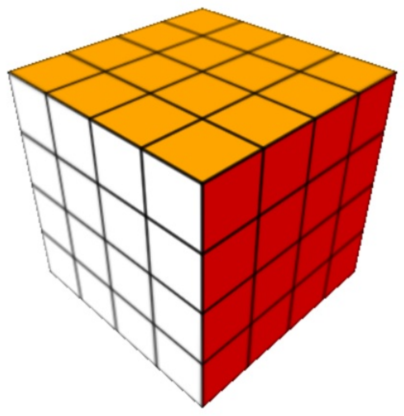

Figure 2: A 4x4 Rubik's Cube

4 Cube Generation

The Rubik's Cube is generated using three.js, which itself uses WebGL. It is comprised of a 3-dimensional matrix of *cublet* objects (the number of which is determined by the size of the cube). For convenience, each cublet is identical; all of the cublets have all six colors, because if the program works correctly, the colors of the internal faces will never be seen, so they do not matter. Each cublet is sized at one unit in each dimension. In Cubes of odd numbered size, the center cublet is centered at the origin, and each other cublet is centered at locations one unit apart; for example, (0,1,1) or (-1,0,1). For Cubes of even numbered size, in order to keep the Cube centered at the origin, each cublet is located every half unit; for example, (0.5,0.5,-0.5) would be the location of a cublet. This way, the origin is the intersection point of the central four cublets. Lastly, ln order for Rubik's Cubes of all sizes to not appear too large or too small to manipulate, I programmed the camera to zoom in or out based on the size of the Cube.

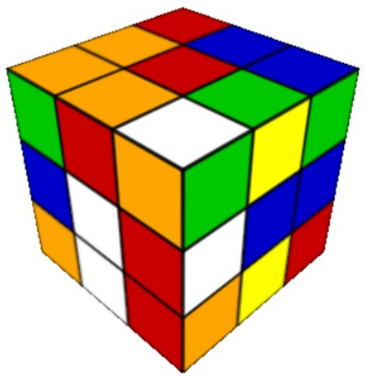

Figure 3: A randomized Rubik's Cube

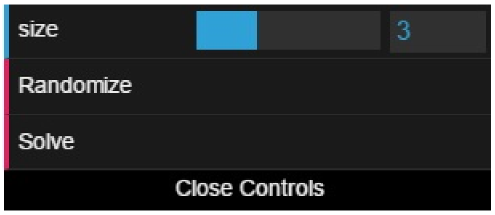

Figure 4: My user interface, implementing dat.GUI

5 User Manipulation

I have programmed four different methods of manipulating the Rubik's Cube. The first method is manually rotating sections of the Cube; this is done by clicking and holding the desired section with the mouse and moving it in the desired direction. In order to accomplish this in the program, I use the *raycaster* function, which creates a ray between the camera and the mouse, and then it finds every object that intersects said ray. I then use the intersecting face of the closest intersected object to determine which face of the Cube will be rotated, and I use the direction of movement of the mouse to determine the axis around which that face will rotate and in which direction.
Secondly, the user can press the arrow keys on the keyboard to rotate the entire Cube 90 degrees in the desired direction. The reason I designed it to rotate a full 90 degrees is to maintain the traditional positioning of the Rubik's Cube.
The third method is a slider where the user can adjust the size, and a new Cube of that size will be generated. The final method is two buttons the user can click on, labelled *Randomize* and *Auto-Solve*, which cause the program to generate a new randomized Cube and to solve the Cube by itself, respectively. In order to implement the slider and buttons, I am using dat.GUI, a lightweight graphical interface for changing variables in JavaScript [2].

6 Randomization and Solving

In order to randomize the Cube, the program conducts a series of random rotations to shuffle the cube. In order to achieve a solved Cube, my program does not actually solve the Cube in the same way a human would, because this would be extremely difficult to program. Instead, it simply generates a new Rubik's Cube of the current size in the solved position.

7 Obstacles

Unfortunately, my project currently is not fully operational, due to a few issues in the program. First, once the user has begun rotating the faces of the Cube, those faces begin to act different from the unrotated faces, wherein they no longer rotate properly. I have not yet found the root cause of this issue. Second, once the user has changed the size of the Cube, it can no longer be manipulated with the mouse. I believe this is relating to poor compatibility between dat.GUI and the *animate* function in three.js, but I have not yet been able to repair this either. If given more time, fixing these issues would be my next task.

References

[1] https://en.wikipedia.org/wiki/Rubik's_Cube.

[2] https://github.com/dataarts/dat.gui.