Solving the Rubik Cube Puzzle

Pax Kulbis

12/14/16

Abstract

The original Rubik Cube is a Three by Three puzzle in which each side of the Cube is a different color. The objective of the puzzle is to reach this solved state through a series of rotations after the Cube has been scrambled. The goal of this project is to create an interactive Three by Three Rubik Cube in the 3rd dimension written in Javascript. Another focus of the project is to create a function which solves the Cube.

1 Introduction

Due to its simple structure but extremely complicated nature, the Rubik Cube has been a favorite puzzle for mathematicians and problem solvers. Many have attempted to create algorithms which more efficiently solve the Cube. Also known as *The Magic Cube,* it was originally created in Budapest by Hungarian sculptor and professor Erno Rubik in 1974 to model 3D geometry[1]. Today, the Rubik Cube is used in speed competitions, to represent and solve real and theoretical mathematical problems, and simply just for fun! The original puzzle was designed as a Three by Three Cube with 6 different colored faces, but now there are several different imitations and variations of the Cube such as the Two by Two, Four by Four, Five by Five, etc., the Gear Cube, the Rubik's 360, and so on.

1.1 Definitions

- **Face**: A whole side of the Cube, composed of nine facelets. In the Cube's solved state, the face is completely one color.
- **Facelet**: The smallest part of a Cube. It is composed of one color.
- **Cube**: The whole Rubik Cube.
- **Cubelet**: One physical piece of the Cube. There are three types: centers which are composed of one facelet, edges which are composed of 2 facelets, and corners which are composed of 3 facelets. (In case of the center, cubelet and facelet referes to the same thing)

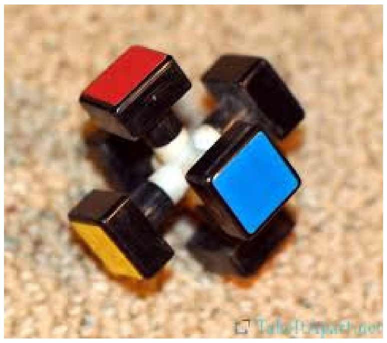
- **Rotation**: A quarter turn (90 degree) of one of the faces.
- **Composition of the puzzle**: The current location and orientation of all of the cubelets of the puzzle.
- **Layer**: All of the cubelets which make a face. Can also be thought of as a face of the Cube and all adjacent facelets.
- **Location**: The place of a cubelet on the Cube (For example, the corner of the red, green, and white faces).
- **Orinetation**: The way in which a cubelet is placed within it's location. (If the cubelet is in the location described previously, possible orientations are white facelet on white face, white facelet on red face, white facelet on green face)

2 About the Cube

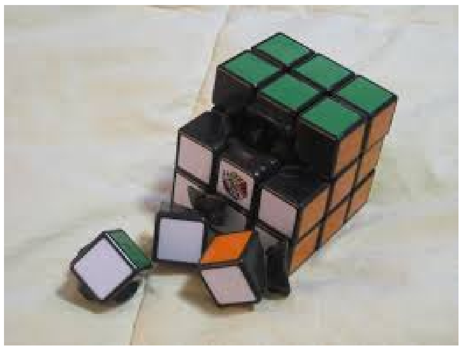
2.1 Structure of Three by Three Cube

The Three by Three Rubik Cube is a Cube where each face is composed of 9 facelets of the same color. Upon first glance, it may seem that each facelet can move independently, but if the Cube is taken apart we see that it is really only composed of 26 pieces. There are 6 center cubelets, 12 edge cubelets, and 8 corner cubeletes. The center cubelets do not change positions, they are the frame of the Cube. All of the other cubelets move and rotate about this frame. Some conclusions that can be drawn from this observation is that since the center cubelets are part of the frame of the Cube, they will never move. Therefore, each side can be referred to as that sides center color. Since each facelet of the edge and corner pieces are connected to another facelet, both facelets must be taken into account when solving for the Cube. Another way of saying this is that we must consider the whole cubelet before making a move, not just the facelet. If we solve one whole face of the Cube, but the adjacent facelets do not match with their side, then in reality we have not done any useful work. We will have to move these cubelets later so that all facelets match, undoing the work that we put in to solve that face. The solved Cube should be thought of as all of the cubelets being in the correct location and correct orientation. This is one of the hardest concepts to understand about the Cube. A rotation will change the position of the cubelet that we are focused on, but it will also change 7 other cubelets.

Figure 1: Disassembled Three by Three



(a) Center cubelets cannot change location, only orientation

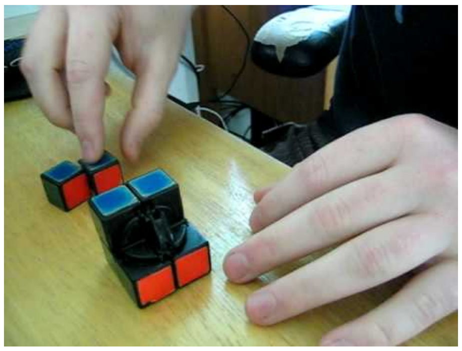


(b) Corner and Edge cubelets are held together by a frame

2.2 Structure of Two by Two Cube

The Two by Two Rubik Cube is similar to the design of the Three by Three, but has some important differences. To start there are only 4 facelets on each side and 8 total cubelets. Each cubelet is a corner piece, so it has 3 facelets. By removing the edge cubelets and center cubelets, the puzzle is less restricted in its movements. There is no rigid frame in the Two by Two puzzles, so it is harder to keep track of which face should be which color in the solved cube. The frame rotates with the Cube, so it is not constant and cannot help the user. Even though each cubelet can move to any location, the solved Cube will still always have the same colored faces next to each other. On a standard Cube, if white is the top and yellow is the bottom, then blue will be on the right of red, orange will be on the right of blue, green will be on the right of orange. (People who solve the 2 by 2 simply memorize where each color should go.) This is true because as previously mentioned, each cubelet will have the same facelets in the same orientation relative to the virtual frame. Rotations change the composition of the Cube, not of individual cubelets.

Figure 2: A partially disassembled Two by Two Cube



The frame is hidden and unmarked, unlike the Three by Three.

2.3 Possible Positions

Since the Cube is composed of cubelets not facelets, we will consider the different types of cubelets to calculate the possible permutations of the Cube. Each cubelet has a location and an orientation. The location is its place in the Cube (i.e. the corner of the red, green and white faces), and the orientation is determined by the face that the facelets are in (i.e. white facelet on the green side). Only one facelet is required to determine the orientation of the cubelet. The other two facelets will always be in the same orientation in relation to the third facelet since the composition of the cubelet doesn't change. Both location and orientation are used to determine the number of possible positions, but the different types of cubelets and a concrete frame place some restrictions upon the total amount of possible positions, as shown in the following equation:

$(8! \times 3^7)(\frac{12!}{2} \times 2^{11}) = 43,252,003,274,489,856,000$

By breaking this equation down, we can more fully understand what is going on.

- $8!$: This represents all of the possible locations of the corner cubelets. Since there are 8 locations, and every piece can be at every location, there are 8 factorial possibilities. This equates to 40,320 possibilities.

- 3^7: This represents all of the possible orientations of the corner cubelets. Each cubelet has 3 possible orientations, which gives the base. It may be expected that the base should be raised to the 8th power as there are eight corner cubelets. However, since a rotation of a side moves 4 corners at a time, the corner's orientation is not completely independent of each other. If 7 corner cubelets' orientations are known, then there is only 1 possible orientation for the last piece. This section equates to 2,178 possibilities.

- $\frac{12!}{2}$: This represents all of the possible locations of the edge pieces. There are 12 locations where edge pieces can reside giving 12 factorial. It is divided by 2 due to the reason that a rotation changes the location of 4 edge pieces. This didn't apply for the corner pieces because corner cubelets can be affected by 3 different rotations, while edge cubelets will only be affected by 2 different rotations. The corner cubelets have more freedom. This section equates to 239,500,800 possibilities.

- $2l^{11}$: This represents all of the possible orientations of the edge cubelets. Each edge cubelet has 2 possible orientations which gives the base. It is raised to the 11th power and not the 12th for the same reason as the corner cubelet pieces. This section equates to 2048.

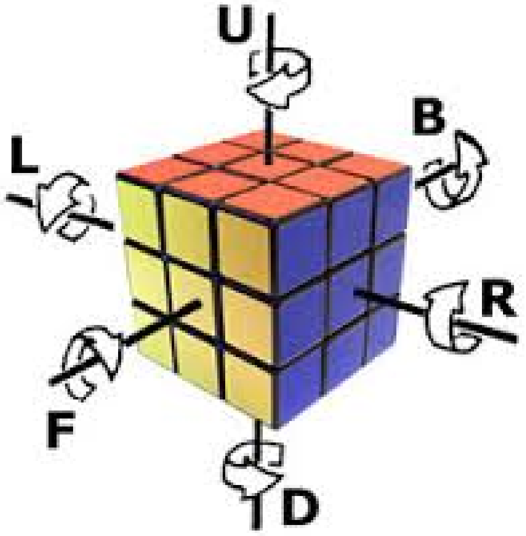
- 1: This represents the location and orientation of all of the center pieces. As explained earlier, the center pieces are part of the frame of the Cube, and so only have one location. Furthermore, since the center facelet looks the same in any of its four orientations, we do not have to account for it. This equates to 1 position.

By multiplying all of these together, we get over 43 quintillion, the amount of possible permutations for a standard Three by Three Rubik Cube reached only through rotations. It is important to note that if positions that can be reached by disassembling and reassembling the Cube are included, there would be no restrictions. In a similar fashion, if the orientation of the center pieces matter, than the product would be multiplied by 4^6: 4 possible orientations for 6 center cubelets.

2.4 Singmaster Notation

Singmaster notation is the standard representation for rotations of any cubic Rubik Cube (as opposed to Rubik puzzles with more or less than 6 sides)[3]. This notation was created by David Singmaster's[1] In this documentation, the different faces are characterized as one of the following: front(F), back(B), left(L), right(R), up(U), or down(D). Since the center cubelets of each face on a Three by Three Cube don't ever move, it is also appropriate to denote the rotations in terms of their center color. However, this doesn't apply to a Two by Two Cube, so the standard F, B, L, etc. is used. Each letter represents a clockwise turn (while looking at the face), and a mathematical prime mark (') after the letter represents a counter-clockwise turn. Each letter indicates a rotation of the coinciding layer around the appropriate x, y, z axis.

Figure 3: Direction of rotations in Singmaster notation



Yellow is the front side in this case.

[1]A retired professor of mathematics at London South Bank University, he is most famous for his Cube notation, his solution to the Rubik Cube and his personal collection of mechanical puzzles and brain teasers.

3 Algorithms

99 percent of people who can solve the Rubik Cube puzzle use a combination of algorithms and adjustments to set up the next algorithm. In terms of the Rubik Cube, algorithms are used to change the location and/or the orientation of a cubelet without messing up the rest of the puzzle. (Algorithms do change other cubeletes as well, but not ones that are important to the user at the time.) An algorithm is composed of a precise order of rotations. In order for the algorithm to work correctly, certain preconditions must be met. Preconditions are specific to each algorithm and require a cubelet, or several cubelets, to be in the correct location and/or orientation. In between algorithms, adjustments are made by the user to meet these preconditions. There are several different series of algorithms, called methods, used for solving the Rubik Cube. These methods range in complexity and are used for different reasons.

3.1 Layer Method

The Layer Method is also known as the Beginner's Method as it is one of the easiest methods to memorize and execute. Jessica Fridrich[2] is given credit for this method as she made it popular by posting it online in 1997[2]. (The Beginner's Method is actually a simplified version of Fridrich's method.) The layer method solves the Cube, as expected, by layer. A layer is defined by the cubelets, not faces. With the Three by Three Cube, the bottom and top layers consist 4 corner cubelets, 4 edge cubelets, and 1 center cubelet, while the middle layer has 4 center cubelets and 4 edge cubelets. The Layer Method has 7 steps, each of which has a, or a group of algorithms, to choose from to complete the goal from the step. These algorithms may be, and usually are, used more than once in each step.

3.2 Thistlethwaith's Method

Morwen Thistlewaith's [3] Method was designed to solve the Cube in as few moves as possible, at most 52 to be precise[4]. [4] His algorithm works so efficiently because it splits the Cube into groups of increasing restrictions.

- G_0 = [L, R, F, B, U, D] This includes all possible positions of the Cube.
- G_1 = [L, R, F, B, U2, D2] Positions that can be reached (from the solved state) with quarter turns of the L, R, F, and B, but double turns of U and D.

[2] A professor at Binghamton University, who specializes in data hiding applications in digital imagery.

[3] A mathematician and professor at the University of London, Thistlewaith is an adament problem solver. He has a huge collection of problem solving books and has come up with his own puzzle, the Thistlethwaite's loop, in addition to his contributions to the Rubik Cube puzzle.

[4] Currently, it is has been proved that the Rubik Cube can be solved from any possible position in 20 moves. This is known as the *God's Number* of the Three by Three Rubiks Cube, as it would take a being of much higher intelligence to know exactly which moves to perform

- G_2 = [L, R, F2, B2, U2, D2] Positions that can be reached with quarter turns of L and R, but double turns of F, B, U, D.
- G_3 = [L2, R2, F2, B2, U2, D2] Positions that can be reached with double turns of each side.
- G_4 = [1] The solved Cube.

To transition from one group to another a certain algorithm based on the position must be used. Since this method is so efficient with its rotations, it has thousands of algorithms to choose from to meet the exact composition of the Cube. Thistlethwaite's Method was originally written on a computer due to the quantity of data. It is impossible for a human to memorize all of these algorithms and different cases, so there have been adaptations made to the original method so that it would be more human friendly, while still maintaining the properties of his group theory. This Human Thistlethwaite Method is widely used in professional speed solving competitions due to its efficiency.

3.3 My Solve Algorithm

As humans, our minds are able to analyze a situation and pick out the important parts. Computers, on the other had, must have specific instructions for specific situations. Since there are so many possible positions of the Cube, creating a general solution for every possible position is out of the scope of the author's coding knowledge. Therefore, a more simple algorithm was created which solves a Cube that has been moved up to 2 times from it's solved state. This algorithm was created through experimentation and observations of the composition of the Cube after rotating it two different ways. The first thing that was created was an algorithm that solves the Cube after one rotation. After one rotation, 4 faces of the Cube will not have the same color throughout, while the other 2 faces will be complete. One of the completed faces is then rotated until it has reached its original position (i.e. four rotations) or if the Cube is solved. (To check if the Cube is solved, the 2D array representing the Cube is compared with a 2D representation of the solved Cube. If each element in the arrays is the same, then the Cube is completed. There is a *checkCube* function for this.) If the Cube is not yet solved, the same process is done on the other face that has not been altered by the original rotation. Although it may not be the most efficient way, this will always result in a solved Cube. In the program, this is named the *solveOneMove* function. To account for the second rotation, there is more variability. There are three different types of situations: the same face is rotated two times, opposite faces rotated, and any other combination of rotations. To determine which situation has occurred, count the amount of corners that are not in their solved state.

- For the first situation, there are four corner cubelets in an incorrect position. There are four corner cubelets to a face. If only one face was rotated, then these four corners are wrong (wrong meaning that the cubelet is not in its solved location or orientation).

To solve the Cube from this situation, only the *solveOneMove* function must be called since it is implemented in a way that it is able to solve the Cube.

- For the second situation, there are eight corner cubelets in an incorrect position. Opposite faces share no corner cubelets. If opposite faces are both rotated, then all 8 faces are wrong. To solve this situation, a similar process to the solveOneMove function is used. First, a face which is complete is located. It is then rotated until that layer is correct. (To check if the layer is solved, the 2D array representing the Cube is compared with a 2D representation of the solved layer. If each element in the arrays is the same, then the layer is completed. There is a *checkLayer* function for this.) This process solves four of the corners and leaves four remaining incorrect corners. The *solveOneMove* function is then called to solve the rest of the Cube.

- For the third situation, there are six corner cubelets in an incorrect position. The original rotation results in four incorrect corners. An additional rotation (as long as it is not of the same or opposite face) will result in only 2 more total incorrect corners since that face already has 2 incorrect corners (due to the original rotation). To solve this situation, we must simplify the Cube to a position where only four corners are incorrect so that the *solveOneMove* function can be applied. First, the checkLayer function is used to locate a layer which has 2 correct cubelets. This will result in two possible faces. The opposite face is then rotatated until there are only four incorrect corners or until it has reached its original position (four rotations). If it has not reached the point with four incorrect corners, then the same process is repeated on the other possible face. Once there are four incorrect corners, then the *solveOneMove* function can be applied. There is a special case within this algorithm where there are 4 incorrect corners, but the Cube is not in a composition in which the solveOneMove function will correctly work. To safeguard against this case, a method which makes sure that there 2 solved faces before going on was added.

4 Implementation and Design

This project is written mainly in Javascript. However, Three JS and HTML5/Canvas were also used to implement the graphics and animations of the Cube. The Cube is represented by a 2D array which holds the colors of the individual facelets of the Cube. Whenever a function is called that changes the composition of the Cube, the colors inside of the 2D array are changed. The representation of the Cube is then updated with the new changed composition. To draw a facelet onto the canvas, it takes the color from the 2D array and draws a facelet of that color onto the canvas. This new facelet is then stored in an identical position of another 2D array that represents the display of the array. Throughout the execution of the program, the 2D array of facelets periodically calls upon the 2D array of colors to update the Cube.

4.1 Chrome Console

As part of it's browser, Google Chrome has included a way to interact with it's window in real time. "The Chrome Developer Tools (DevTools for short) are a set of web authoring and debugging tools built into Google Chrome. The DevTools provide web developers deep access into the internals of the browser and their web application"[5]. This console will be used to call commands for the Interactive Rubik Cube. To access the chrome console from the Interactive Cube, simply right click on the window (with the Cube) and select **Inspect**. On the right side of your current window, the DevTools panel should open. Open up the **Console** tab, which is located in the upper toolbar of this panel. (There are other ways to access the console that work just as well.) To call a function, simply type in the desired command in the text box and click **Enter**.

4.2 Functions

In the project, all functions work almost the same way for the Two by Two and Three by Three Cubes. The only difference is that when the functions are dealing with the Three by Three Cube, they must account for edge and center pieces as well as all the elements of the Two by Two.

4.2.1 Scramble Function

Scrambling someone else's Rubik Cube is sometimes more fun than actually solving it (especially for those who can't solve it); however, sometimes it becomes repetitive. There is a scramble method which performs 25 rotations consecutively (25 is an arbitrary number). This method is implemented by going through a for loop 25 times, performing a random rotation each time. To call this function, type *scramble()* into the console.

4.2.2 Solve Function

The solve function was described earlier. To call it, type in *solveTwoMoves()* into the console.

4.2.3 Animate

This function is an internal function of the program that animates the rotations when called. The parameters of this function are face, direction, and start time. The face and direction (clockwise or counterclockwise) are taken from the queue, and start time is taken from when animate starts rendering the current animation. Every couple of milliseconds, the time is linearly interpolated to find an angle of displacement for the layer that is being rotated. The layer is then rotated that angle on the appropriate plane and is redrawn. This recurs until the rotation is complete. The user never has to call this function; as an internal function it is called automatically as the program runs.

4.2.4 Rotation Functions

Each of the rotations has their own function and consists of several different parts.

- Rotation added to queue: The desired rotation is added to the queue in the form of an object with two properties: face and direction. These objects of the queue are eventually sent to the animate function which displays the rotation. This step allows for several rotations to be called at a time, without animating them all at once.
- Each layer is rotated: This has two parts: the rotation of the face and the rotation of the adjacent pieces. This is accomplished by changing the color each facelet, and then updating the 2D array with colors in new places. Since the Cube is implemented with 2D arrays, each rotation needs unique code. This code was created through experimentation and observation of the Cube.
- Cube is redrawn: Once the animation has finished and the Cube has been updated with the correct colors after the rotation, the Cube is redrawn onto the canvas.

To rotate a face, simply type in the corresponding uppercase letter followed by parentheses in the console (e.g. *F()*, *L()*, etc.). Counter clockwise rotations have the same implementation, but with their own unique code for rotating the layer. To call them, type in the corresponding uppercase letter, cc (for counter clockwise), and then parentheses (e.g. *Fcc()*, *Lcc()*, etc.).

4.2.5 Design

This set of functions was created entirely for recreational purposes. When called, a series of rotations is performed that ends up creating an aesthetically pleasing pattern in the cube. There are two such functions: *design()* and *design1()*

5 Timeline

- Weeks 0-8: This served as an intro to the class in which we explored new programs with prof. Ando. We also learned the basics of Javascript
- Weeks 9-12: In these weeks, I worked on creating the RTICA for my Two by Two cube, including a geometric and visual representation of the cube, rotation functions, and the solve function.
- Week 13: Seeing as creating the solve function was out of the scope of my coding knowledge, I switch tactics and worked on my *solveTwoMoves()* function. I also worked on my website.
- Thanksgiving Break: I created a Three by Three RTICA. I also worked on my website and narrative.
- Weeks 14-16: I created the animations of the cube as well as completed my narrative and website.

6 Future Extensions

With more time, resources, and/or knowledge, the following developments could be made to improve the project.

- A solve function that would work on all possible positions (using the Layer Method and/or Thistlethwaite's Method).
- The project could be expanded to include larger Cubes, such as the Four by Four, Five by Five, etc.
- A more user-friendly interface. There could be different buttons used for the different rotations.
- Instructional tutorial videos that would help beginners learn the different methods.
- A way to zoom in and out through scrolling.
- A more graphically aesthetic Cube, complete with shading, better spacing between the pieces, etc.
- The possibility of choosing preferred colors (or even images) for the different sides of the Cube.
- Make the code more efficient (Such as the solve two layer function)

References

[1] `https://en.wikipedia.org/w/index.php?title=Rubik_Cube&redirect=no`

[2] `https://en.wikipedia.org/wiki/Jessica_Fridrich`

[3] `https://en.wikipedia.org/wiki/David_Singmaster`

[4] Scherphuis, Jaap. *Jaap's Puzzle Page* Thistlethwaite's 52-move Algorithm. N.p., n.d. Web. 05 Dec. 2016.

[5] Meggin Kearney, Kayce Basques, *Chrome DevTools Overview.* Chrome DevTools Overview - Google Chrome. Google, n.d. Web. 05 Dec. 2016.