Rubik's Cube

Tyler Campbell

Abstract

My project is a Rubik's Cube that can be controlled using both key inputs and the mouse. The cubes can be scrambled and solved. There are buttons to undo a move and reset the cube back to the original state. There are 4 options for the speed of turning the cube (slow, normal, fast, and very fast). There are 4 options for the size of the cube (2x2, 3x3, 4x4, and 5x5). There is a button that shows and explains the controls for the cube. Each color of the cube can be changed independently and the colors can be reset to the default colors. There is a PLL/OLL trainer, the algorithms that solve the last layer. The trainer shows the algorithms and how they are done. It can be used to learn and to practice. Lastly, there is an example solve, which shows the 4 steps in the CFOP method.

Introduction

Goal

The initial goal of the project was to build an interactive Rubik's Cube. I was inspired to create this because speedcubing is a hobby for me. To begin I wanted to create a functioning cube that could be moved with keyboard inputs, be able to randomly scramble, and look visually close to a cube. In addition I wanted to have different cube sizes, a solver, and be able to move the cube with the mouse. After getting most of the core implementations done, I created a trainer and example solve that can help both beginners and experts alike. The audience of the cube varies from people that just want to mess around with the cube, to beginners learning to solve, and to experts that want to practice.

Previous Work

There are multiple interactive cubes online that have different approaches. From different controls, visuals and frameworks used. A common theme among all of the cubes online is that the cubes do not look realistic. Due to this, I really wanted to focus on making the cube look good through graphics. Additionally, many of the cubes can only be controlled by the keyboard.

Approach

My approach was to use Threejs for the cube, bootstrap for the UI, and the plain html/css/js combo for the rest.

Methodology

The main steps of implementation:

1. The Cube (Visual and different sizes)
2. Movement with Keys
3. Movement with Mouse
4. Extra Features

Cube

The first part of the implementation is the cube itself. The entire cube consists of 27 smaller cubes. To start I used BoxGeometry and BasicMeshMaterial for each Mesh that represented a small cube. The middle cube was placed at (0,0,0) and the distance between the other cubes is ⅓. I then appropriately scaled each cube such that there is no space between the cubes. Next part was to correctly color each face of the cube. This was done using the position and normals of the face of the small cubes to determine which faces were exposed then colors appropriately. After finishing the initial implementation I wanted to make the cube look better. To do this I switched to make the smaller cubes rounded and instead of coloring the cubes, I added a rounded plane to represent the stickers on the cube. To assist with the geometry I used RoundedBoxGeometry and RoundedPlaneGeometry and I switched to a Phong Material. The cube sizes and positions are the same as before. When creating each cube the exposed faces are assigned to the cube. These faces (which are numbered 0-5) are then used to set the position and rotation of each sticker. Code below:

```
sticker.position.set(
   1/6 * [ -1, 1, 0, 0, 0, 0 ][ faces[i] ],
   1/6 * [ 0, 0, -1, 1, 0, 0 ][ faces[i] ],
   1/6 * [ 0, 0, 0, 0, -1, 1 ][ faces[i] ]);
sticker.rotation.set(
   Math.PI / 2 * [ 0, 0, 1, -1, 0, 0 ][ faces[i] ],
   Math.PI / 2 * [ -1, 1, 0, 0, 2, 0 ][ faces[i] ],
   0);
```

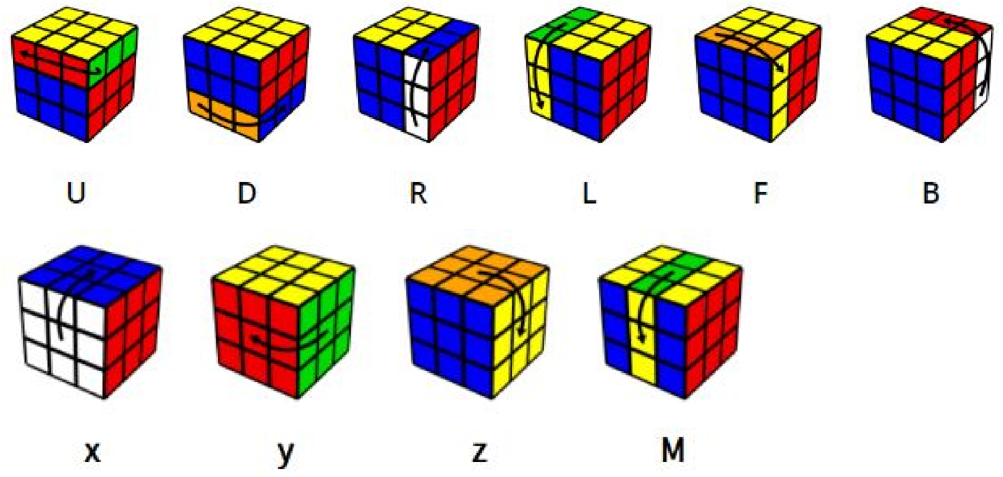
After the position and rotation is set the stickers are scaled to fit the cube face and assigned as a child to the cube. Next step was to allow different sized cubes. For different size cubes the method is the same however loop through the appropriate amount (the length) for each x,y,z position. Thus the 2x2 has 8 smaller cubes, 4x4 has 64 and the 5x5 has 125. The initial offset position in each direction is given by:

```
const offset = this.size % 2 != 0
? 0 - Math.floor(this.size / 2)
: 0.5 - this.size / 2
```

Where this.size is either 2,3,4, or 5. The cube spacing for each cube is still ⅓ for all sizes.

Key Movement

The next thing to implement was the turning of the cube. To do this the smaller cubes that needed to be rotated with the associated move is added to a pivot object. The pivot is then rotated about the axis in a speed and direction until it reaches PI/2. After the move is complete the cubes are removed from the pivot. Each move is represented by a direction (positive or negative), axis (x, y or z), and the layers that need to be turned. The move is determined by standard cube notation. Each cube specific move is determined by a certain character where an uppercase character is rotated in the clockwise direction. For the user moves could be inputted by using the keys that are associated with the moves. To execute a move all moves are added to an array, and the move is executed one by one. To make sure 2 faces don't rotate at the same time, a new move cannot be added unless the cube is static.



The controls of the cube follow standard rubik's cube notation where the faces (front, back, up, down, left, right, etc) can be turned either clockwise (uppercase) or counterclockwise (lowercase).

Mouse Movement

Next step was to add support for moving the cube with the mouse. This is done by adding a listener for mousedown and mouseup. The x and y coordinates of mouse down are converted to normalized device coordinates to see if the mousedown action intersects with the cube. If it intersects the cube the position of the cube and normal of the intersection are used to determine what layer and face needs to be rotated. Then the difference between the x and y of the mousedown and mouseup events is used to determine the direction of the rotation. If there is no intersection, the difference determines the direction of a cube rotation (the entire cube). Depending on the intersection and difference the character representing that move is added to the array of moves and then is performed. For the bigger cubes (4x4 and 5x5) special moves needed to be added to support moving the inner layers.
An example: clicking the furthest right layer on the front face means to rotate the right layer. Then if the mouseup Y is less than mousedown Y (the end is less than the start) then rotate taht layer clockwise or a R move.

Features

After getting the basic functionality and visuals of the cube done, different features were added.
Scramble: Randomly generates possible moves depending on the cube size and adds to the array of moves to perform. It is optimized not to perform a move then undo that move. Each possible move for the current size cube is added to a string and random characters are selected from that string (which represent the move). The new move is only added if it does not undo the previous move added.
Solve: As the cube is moved, the inverse of each move is added to the front of an array. This array when performed solves the cube. The array is reduced as moves are added (i.e. R then r cancel each other out, thus to solve no move is needed) and before the solve (i.e. R R R R moves cancel out, thus to solve no move is needed) to eliminate some repeated moves or unnecessary moves. (This undos all previous moves with some optimization)
Undo: This performs the first move in the array mentioned in solved. This will reverse the last moves made by the user.
Reset: Resets the cube to its default state by removing the old cube and creating a new one. (This does not reset any color changes)
Speed: The speed of rotations has a slow, normal, fast, and very fast setting. This corresponds to the rotation speed of a move. When rotating the rotation of the pivot is increased or decreased (depending on direction) by this rotation speed.
Size: Changes the size of the cube (described above).
Controls: Shows the user what key inputs correspond to each type of move if they do not know cube notation. The pictures can also be clicked to move the cube.
Colors: Each color picker changes the color of that set of stickers. The colors of the stickers are represented by an array. When a color changes the array is updated and the colors of the stickers are updated. The index of the array that the sticker's color should come from is stored in the userData of the sticker. Resetting the colors restores the colors to the default cube colors without resetting the entire cube.
PLL/OLL Trainer: The last layer of a cube is solved with algorithms. This feature helps visualize and practice these algorithms. It first does the reverse of the algorithm. Then using OrbitControls the cube rotates to help the user visualize the cube when this algorithm needs to be performed. The algorithm is put on the screen to show the user the correct moves. The algorithm is then performed to solve the cube. More special moves need to be added in order to complete this. These moves were moving the right and middle together and the front and middle layer together. All of the algorithms are hard coded to each picture.
Example: The example solve is a way for beginners to see the steps of solving a cube. The scramble is predetermined. I then solved the cube and wrote down the moves for each step, then hard coded each step's moves. The example follows the popular CFOP method.
UI: The UI is made using bootstrap and has basic js listeners for each button.

Discussion and Results

Overall I believe I took the best approach and I believe I succeeded. I achieved all of my main goals and multiple stretch goals. There are some features that I implemented but did not make the final version of the project. One being I added different backgrounds, but I did not include any of them because the rendering of turns was not as fluid and did not look as good with no background. Another feature was extending the sticker colors of the unseen faces out such that all 6 sides colors could be viewed. I left this out because when using a real cube you can not see all of the faces. So this was left out to make it more realistic for the user. There are also some future features that would have been implemented with more time and are the next steps in the project. The first being a general solver that solves the cube using the CFOP method (same as in the example). This would decrease the overall time to solve and would help people learning how to solve a cube visualize the actual solve process. A possible alternative is to create an algorithm that can always solve the cube within 20 moves (which is the highest possible move to solve any scramble of the cube). Next, I want to have better and more responsive movement with the mouse. Such that the cube moves with the mouse. For example, when turning a face and the mouse is moved half a turn the cube will move that same half. Lastly, I want to make different shapes of rubik's puzzle such as a pyraminx which is made of small tetrahedrons. I started both the CFOP solver and pyraminx, but stopped because I didn't think I would be able to finish in time so I put my focus to other features. For testing, I had friends, both speedcubers and non-cubers, test out the demo. They reported any bugs, such as a face not moving correctly and input on what I could do better. From testing, I got the idea to add movement to the middle layers of the bigger cubes. I also got feedback on mouse controls which I later tweaked to make it more accurate and user friendly.

Conclusion

I really enjoyed this project (along with the class as a whole). It was very fun to implement something that I am passionate about and can use in the future. I was surprised by and proud of the end product. I learned Three Js and strengthened my javascript by completing the project. The most useful skill being understanding event listeners and effectively debugging javascript code. I can use these in future projects for front-end development.

Works Cited

Rounded Cubes: https://github.com/pailhead/three-rounded-box
Color Picker: http://jscolor.com
Rotating cube:
https://stackoverflow.com/questions/20089098/three-js-adding-and-removing-children-of-rotated-objects
Cube Notation and Algorithms: https://jperm.net
UI: https://getbootstrap.com/docs/4.4/components