Richa Mohan
15-112
Section J

RUBE-OIDS:
The Rubik's Cube-oid Generator and Solve

THE PROBLEM:

Creating and Solving a Rubik's cube seemed like an interesting problem from the perspective of programming and that is why I chose to make a rubik's cube/cuboid generator and solver for my term project.

Creating the Cube-oid: Though there are a lot of program that show you the steps to solve a rubik's cube, most of them don't allow you to choose the dimensions, see the formation of the cube-oid and shuffle it yourself which can all be a lot of fun from the perspective of a user. Programs like ruwix.com which have some great features like showing the user the steps to solve a rubuk's cube, doesn't allow the user to do any of these things and hence, I wished to add them to my program.

Rotations: Rotations was the most challenging and hence, the most interesting problem in this program. The orientation of the boxes on each face of the cube is different, hence obtaining the desired row or column for rotation can be a hard task, especially if you wish to show the rotations for the user. While showing the 3D rotation is inherent in many programs, it is missing from many and once again doesn't necessarily extend to different dimensions of the cube and is hence an interesting problem addressed in my program. Hence, this program allows users to see the 3D rotations while shuffling and solving the cube-oids.

Solving the Cube-oid: A rubik's cube is something most people I know cannot solve, but surely wish they could. Many a times when I've unsuccessfully tried to solve a Rubik's cube, I've wished that there was a way I could keep track of everything I did to shuffle it and simply go back those steps. This is exactly what this program does and thus helps the user solve the cube by providing the steps, or solves the cube for the user, as even simply looking at the cube being solved can be an interesting experience.

Rubik's Cuboid: One thing I couldn't find on any of the existing rubik's cube generating and solving websites was rubik's cubiods, that is, extending the idea of rubik's cubes to cuboids. This provides users with an interesting and new perspective of looking at rubik's cubes as some of the rules to solve one would be different, for example, one would have to rotate the cuboid by a 180 degree angle every time.

THE SOLUTION:

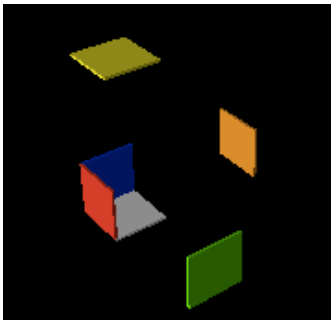
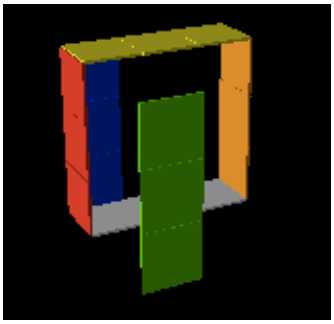
The cube is a sub-class of cuboid as every cube is also a cuboid, hence I shall explain all the steps in terms of the cuboid

Creating the Cube-oid:

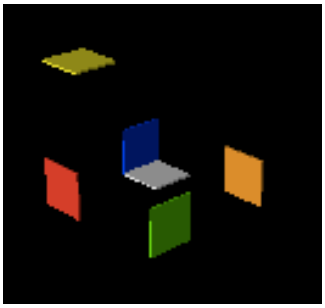
Every cuboid has the usual features, the length, width and height. Based on the dimensions chosen by the user, the length (top to bottom) is divided into rows, the breath (left to right) into columns and the height into hs. Using the number of rows/cols/hs the face and the distance form the center; we obtain the boxes of each individual face.
Each box is approximately of size 1, so if the breadth is divided into 3, then we will have 3 columns of boxes with their starting points as [-1,0,1] and if divided into 4, then the starting point would be [-1.5,-.5,.5,1.5], since (0,0,0) is the center of the screen. By obtaining this list we get the 6 faces of the cube-oid

There were two possible ways to make the faces of the cubes:

1. **Parallel Faces Method**: The first box of each face is parallel to its parallel face: that means that the first red box is parallel to the first orange box, and so is the second one…so on and so forth. This means that obtaining the first col of the front and back faces would require obtaining the same indeces from the list of two faces. This is shown in the following pictures:

2. **Same Starting Point Method:** The first box of each face is at the same starting location, so if the first box of the front face is at the bottom left corner, the first box of every face will be at it's bottom right corner:



While I ran back and forth with both the ways, eventually I chose to use method 1 since it made obtaining the rows and columns for rotation less complicated.

Rotations:

The entire design of the program kind of revolves around the rotations. Hence, this was the most important as well as challenging part of my project.

As mentioned above, I chose the parallel faces method because it made it easier to obtain the faces of rotation. The rotation faces are of 3 different types:

1.Front Horizontal Rotations:

The rotation consists of obtaining a row from the front, back, right and left faces.

Each face has the
2 5 8
1 4 7
0 3 6
orientation of indecs, so in order to obtin any row, I had to get [0,3,6] for the bottom row, which was done using a for loop from the starting index of the row to the end of the face with steps equal to the length of the face.

2.Front Vertical Rotations:

The rotation consists of obtaining the came column from the front, back, top and bottom faces. This was done by obtaining the column respective to the keyPressed using which the starting index for that column was calculated for example, for the first column in this case the starting index would be 1 and then we obtain the column by going from the starting index to starting index+ width of the face.

3. Side Vertical Rotations:

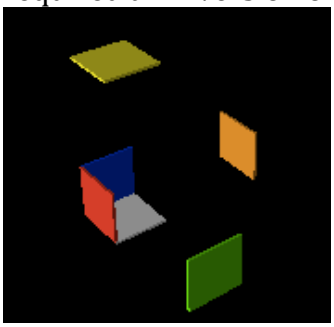
This rotation consists of obtaining a column of the right and left faces and a row of the top and bottom faces. For the side face, that is the right and left faces, the method to obtain the boxes of rotation is the same as that of the one in used in the front vertical rotations and the method to obtain the top and bottom boxes is similar to that of the front horizontal rotation. This is so because the rotation involves columns from the right and left faces and rows from the top and bottom faces.
→Hence , using these methods, the respective rotation frames were obtained.

Rotation Frames: Once the boxes for rotation were obtained, to show the rotation, they were put into a frame, a feature of vpython, which allowed me to be able to show the actual rotation of the cube-oid.
However, a problem created by this was that rotating objects in a frame doesn't actually change the attributes of the individual objects and neither can one delete a frame once it has been established, hence, I had to delete and redraw all the boxes by updating the colors of the individual boxes in the entire cube-oid.

Updating Color Lists:

This required changing the order of the colors in a list that was later used to recreate all the boxes for all the faces. The parallel faces method created complications here because for example, if one were to rotate the bottom row by

-pi/2, the front row becomes the left row(front red row would replace blue left row). This means that the first red box shown below would be the left corner most box of the left row, which is not the first, but the last box of the left row. Hence, this required an inversion of the boxes of the rows for some of the rotations.



Rotating Side Frame:
Using the Parallel Faces Method made the rotation of side frames easier as each side face rotation changed the order of the indeces similarly and could hence be generalized.

Two types of rotations for cubes:

1. When the angle of rotation is pi/2: if the original matrix of indeces of the side
   2 5 8 8 7 6
   is 1 4 7 after a pi/2 rotation it become 5 4 3
   0 3 6 2 1 0
2. When the angle of rotation is –pi/2: The original matrix of indeces changes to
   0 1 2
   3 4 5
   6 7 8

Rotation for cubioid:

- Since the rotations are a 180 degree rotations the side matrix of indeces
  6 3 0
  changes to 7 4 1, that is, it inverts.
  8 5 2

These changes were implemented by coming up with formulas to change the order of the indeces.

Shuffling the Cube:
This function obtains a random keyPress from the possible key Presses and a random angle from the available choices and implements the rotation. The user has the ability to start and stop the shuffling as and how it wishes to do so.

Solving the Cube:

The user can obviously try to solve the rubix cube on it's own, along with that, he also has the following two options

1. Computerized solution: On pressing the solve button, the computer starts solving the cube while showing the user each rotation that it makes to reach the solution.
2. Step-by-step solution: The "show solution" button shows the user, in steps, the keys that it has to press to reach the solution.

Solution: The solution to the rubik's cube-oids is obtained by keeping a track of all the keys that were pressed, that is, the steps taken to shuffle the cube. Then the solution key is the reverse of the keys pressed along with reversed uppercase and lowercase letters as the angle of an uppercase rotation is the opposite of the angle of a lower case rotation. For example: if original list of keysPressed = ['a', 'c', 'F', 'g', 'L'], the solution list will look like ['l', 'G', 'f', 'C', 'A']

Rubik's Cuboid:

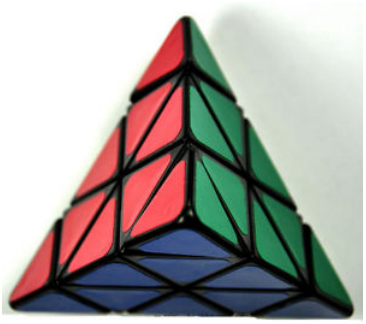
While some of the features were generalizable for the cube and cuboid, some had to be different. The following are things that are special to the cuboid:

- 180-degree rotation angles – since the dimensions of each set of parallel faces is different, you can't make 90 degree rotations.
- The different angle of rotation meant different changes in the faces, hence the algorithm to change the faces for cubes and cuboids is different.
- Different rotation of side face – when one of the corner rows/cols/hs are moved, the face on the side also moves with it. Since the rotations are of a 180 degrees that means that all the boxes on the side face flip, the first in the list becomes the last and the last becomes the first.
- There would be a difference in the improved solution since pressing the same key twice would get the cube back to it's original state along with pressing the same upper and lowercase key consecutively (which is the case in the normal cubes)

FUTURE PLANS:

Here are a few features that I would like to implement later to improve my project:

1. Extending the idea to different shapes: while my project currently extends the idea of a rubik's cube to rubik's cuboids I would love to also make rubik's pyramids (called pyranmix), and perhaps some 3D hexagonal shapes.



2. Create a controls feature that would allow the user to enter any dimensions for the cube-oids that he or she wishes to.

3. The possibility of diagonal rotations in rubik's cibe's and cuboinds which will look something like the following: