Cubix

General Rubik’s Cube Graphics Project

Motivation

- Applied what I learned from Graphics Honors class
- Improve skills in C++, OpenGL, Git
- I wanted to implement my own solver, even if others exist online

Main Goals

Render Cube

Solving Rubik’s Cube

Milestones

Render Cube

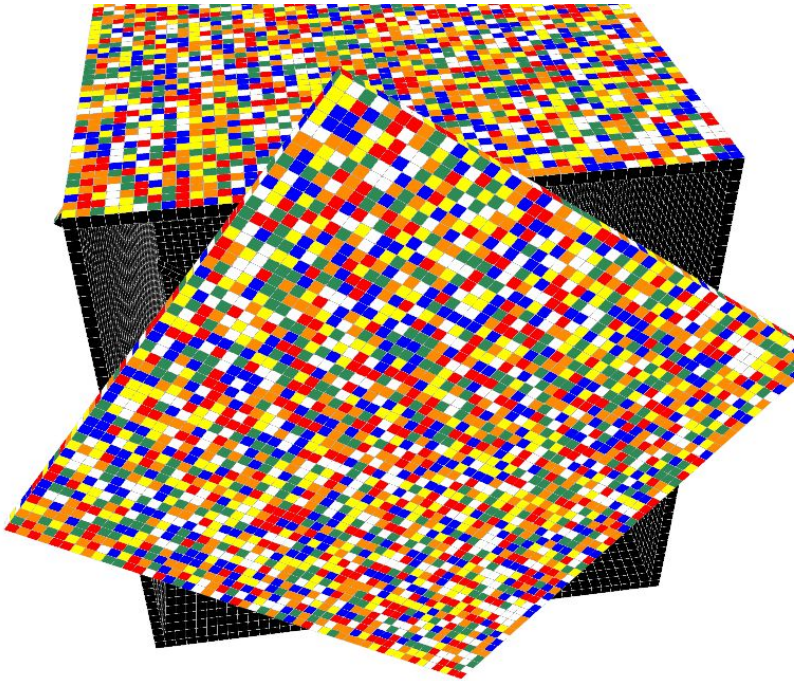
- Positions of Vertices and Faces
- Implementing Turning
- Coloring of faces

Solving Rubik’s Cube

- Solving centers
- Solving edges
- Solving 3x3x3
- Pipeline from solving to rendering

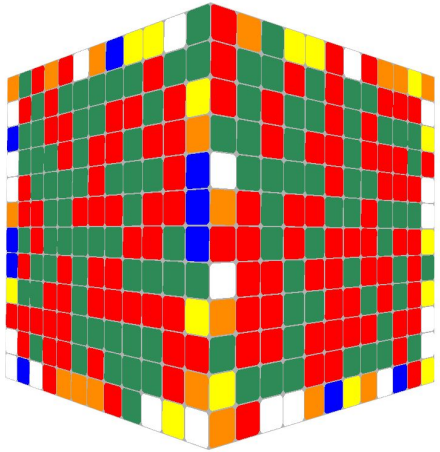
Design of Rubik's Cube

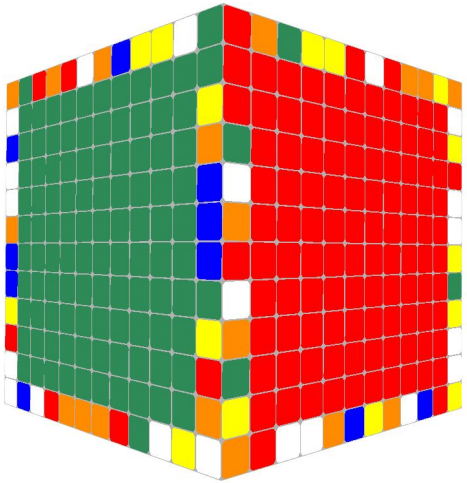
- Hollow
- Render each small cube
- Bits to determine orientation



Solving Center Pieces

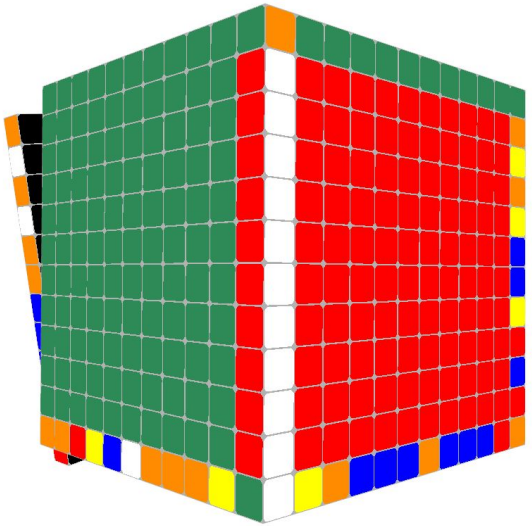
- 8-step process to swap between adjacent faces





Solving Edges

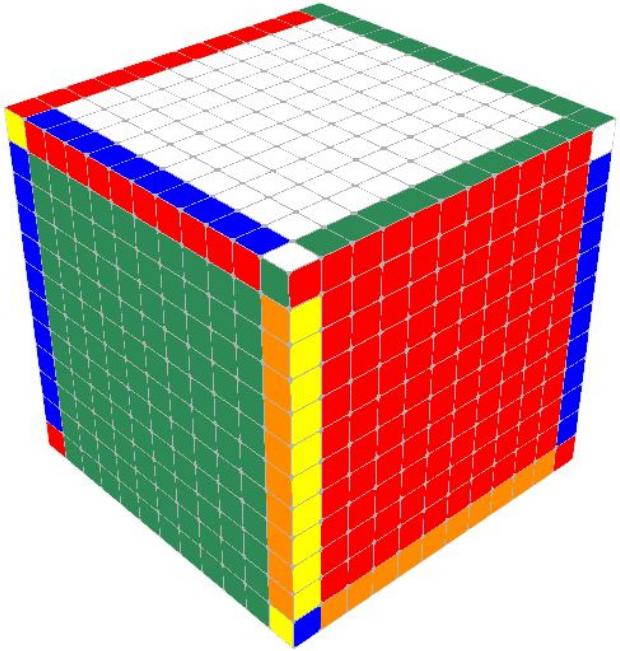
- Different casework depending on where desired cubie is
- Swapping algorithm for last 4 edges



Solve like 3x3x3

- Parity of N

```
void Solver::solve3x3x3() {
    solveCross();
    solveCorners4();
    solveSecondLayer();
    solveLastCross();
    solveLastEdges();
    solveLastCornerPosition();
    solveLastCornerOrientation();
}
```

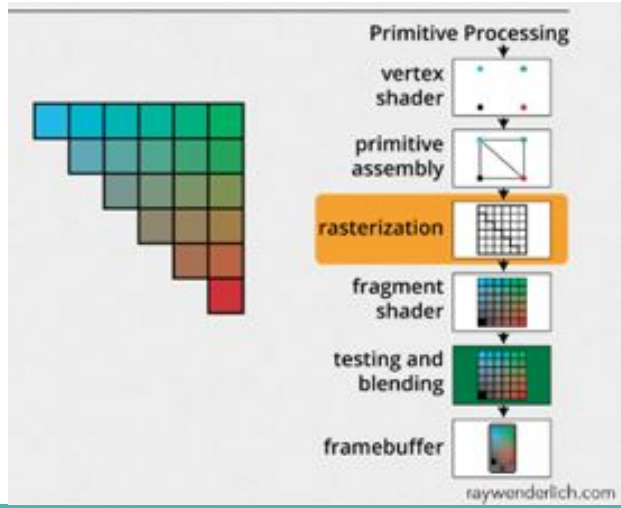


Code Structure

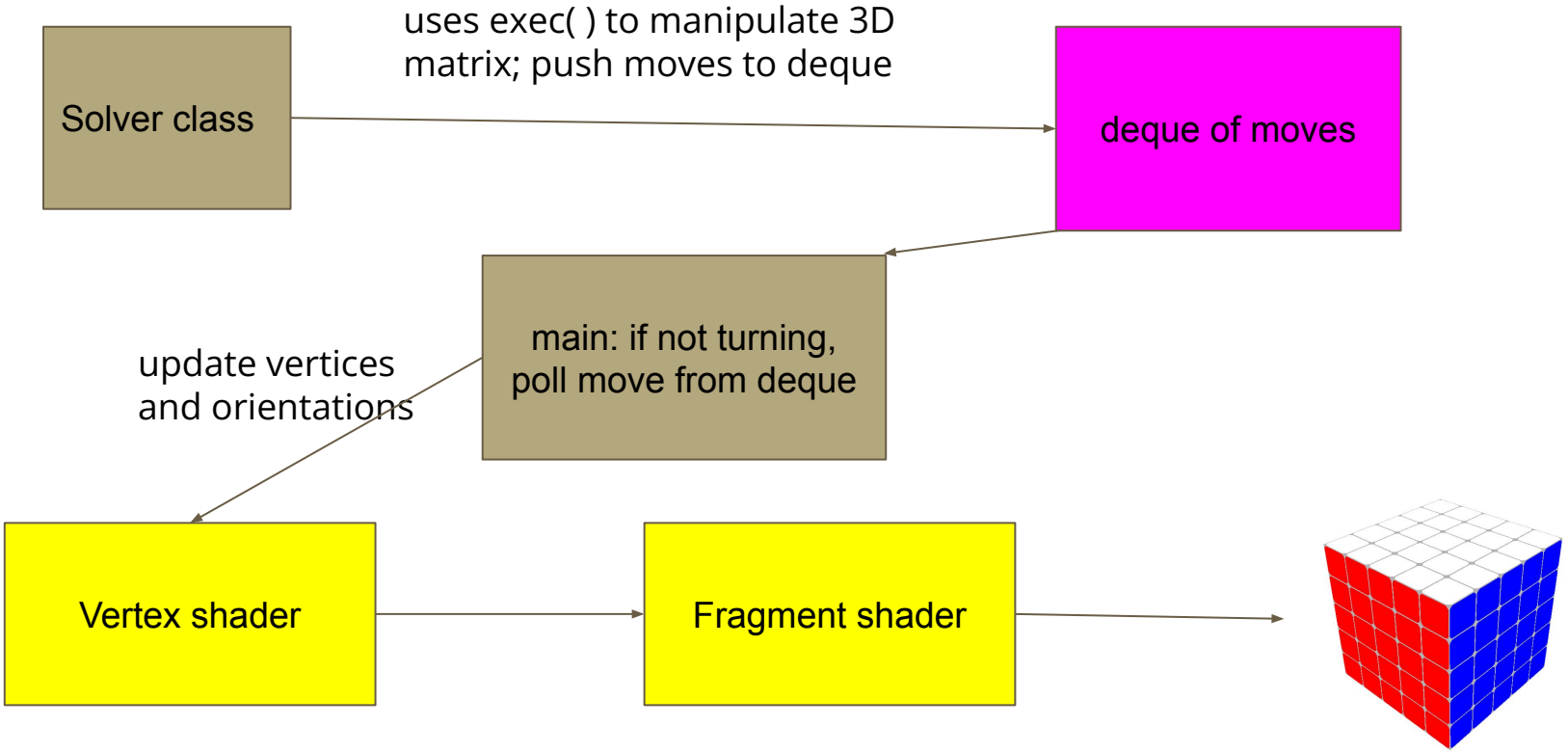
- *config.h* - constants, including cubeWidth
- *solver.cc* - Methods to solve center faces, edge pieces, and corner pieces
- *gui.cc* - Keyboard mapping for user-interaction
- *shaders/cube.vert* - Vertex Shader
- *shaders/cube.frag* - Fragment Shader
- *main.cc* - Rendering loop that updates cubies' positions and orientations

OpenGL Rendering

- Specify vertices and faces in main
- Shaders are programs in GPU that render shapes
  - Vertex shader
  - Fragment shader
- Transforms from world coordinates to screen coordinates

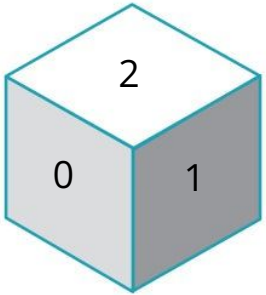


Structure


Solver class
uses exec( ) to manipulate 3D matrix; push moves to deque
deque of moves
main: if not turning, poll move from deque
update vertices and orientations
Vertex shader
Fragment shader

Structure

- Internal representation of cube



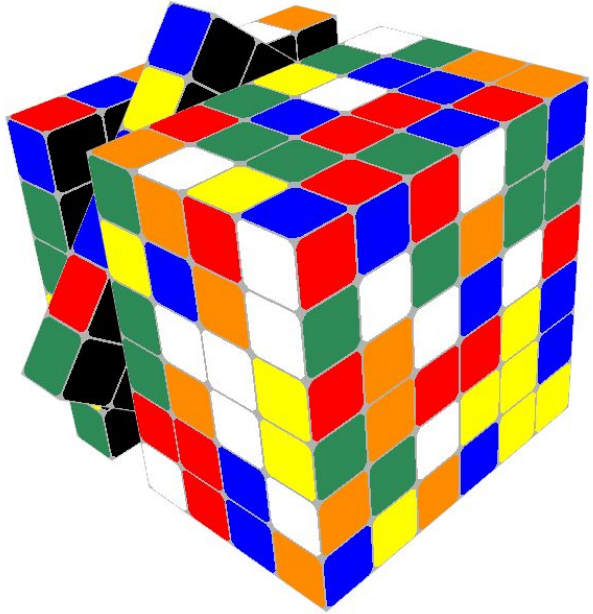

```
int faces[6][cubeWidth][cubeWidth]; // internal representation
// 0 1 2 3 4 5
// front, right, top, bottom, left, back
// green, red, white, yellow, orange, blue

std::deque<glm::ivec3>* dequePtr; // pointer to the deque of moves to fill
```

Structure

- Internal representation of cube
- Turn represented by 3 ints

```
void exec(int face, int layer, int qt);
```



Structure

- Internal representation of cube
- Turn represented by 3 ints
- If graphical representation turning, update theta & vertices; check if it has rotated enough, and if so update cubies orientation
- If graphical representation not turning, poll the next move from deque and start turning

Turning

- Maintain which small cubes are currently rotating
- Time since rotation start and rotation speed
- Vertex shader applies rotation matrix
- Update final positions and orientations of vertices

```
vec4 tmp = vertex_position;
gl_Position = projection * view * mat * tmp;
```

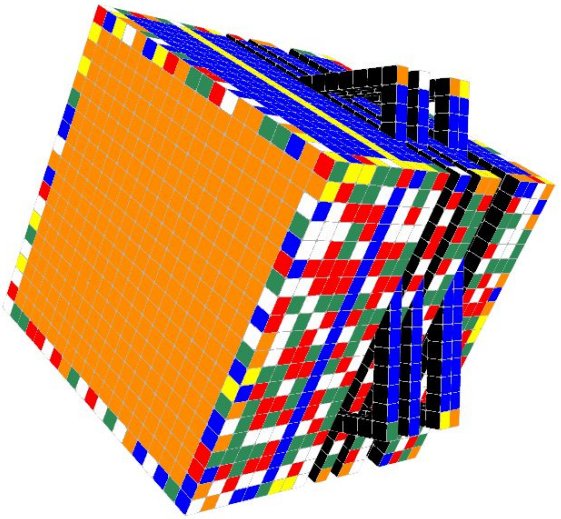
Demo

Feedback from Professor

“Cool! I ran it on my machine for a 71x71x71 cube and it worked! ...I'm pretty impressed!”

Challenges

- Design Choices
- Corner-cases of solving cube
- Optimization: render + solving



Further Work

- Multithreading
- Illusion that inside is not hollow
- User clicks-and-drags layers to scramble the cube
- User can input a JSON file
- Users can try to manually solve the cube themselves

Questions?