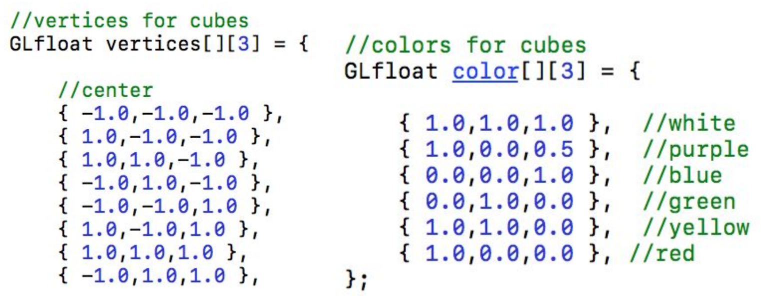
Rubik’s Cube Using OpenGL and C++

I. Motivation

This project encompasses many important computer graphics concepts, including a 3D object composed of sub-objects, coloring, rotations with hierarchical structure, mouse and keyboard control, and animation. This knowledge has been applied to create a Rubik’s cube that has movement and behavior almost identical to its real world counterpart.

II. System Design

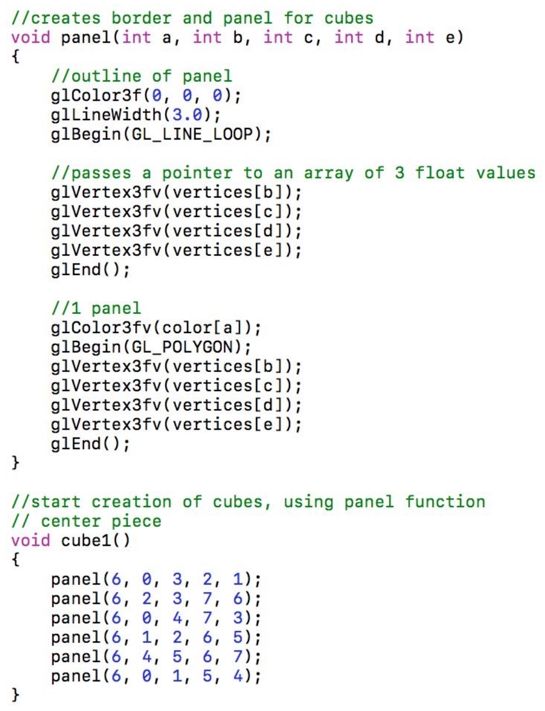
Using OpenGL, GLUI, and C++ in MS Visual Studio, we have created a 3D Rubik’s cube with realistic rotating behavior. The cube is composed of sub-cubes, with a maximum of 27 cubes visible at any time. Also, their respective positions and colors are stored in arrays. The rotation of the entire cube is controlled by clicking and dragging with the mouse, while individual slice rotations are controlled by keyboard input. Rotations are essentially “color-swapping”. Additionally, the speed of slice rotations can be increased or decreased, the camera can zoom in and out, and the cube can be solved with a keyboard command. All keyboard control instructions are displayed on the screen with the Rubik’s cube.



```
//vertices for cubes
GLfloat vertices[][3] = {

    //center
    { -1.0,-1.0,-1.0 },
    { 1.0,-1.0,-1.0 },
    { 1.0,1.0,-1.0 },
    { -1.0,1.0,-1.0 },
    { -1.0,-1.0,1.0 },
    { 1.0,-1.0,1.0 },
    { 1.0,1.0,1.0 },
    { -1.0,1.0,1.0 },
```

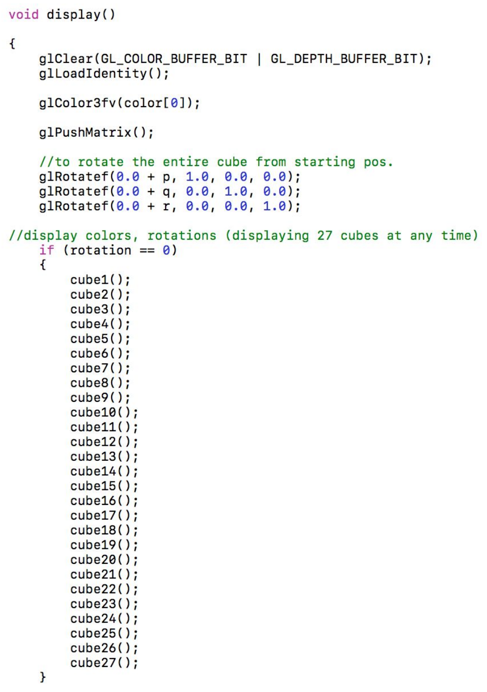
```
//colors for cubes
GLfloat color[][3] = {

    { 1.0,1.0,1.0 },  //white
    { 1.0,0.0,0.5 },  //purple
    { 0.0,0.0,1.0 },  //blue
    { 0.0,1.0,0.0 },  //green
    { 1.0,1.0,0.0 },  //yellow
    { 1.0,0.0,0.0 }, //red
};
```

Figure 1. *Storing vertices and colors in arrays*



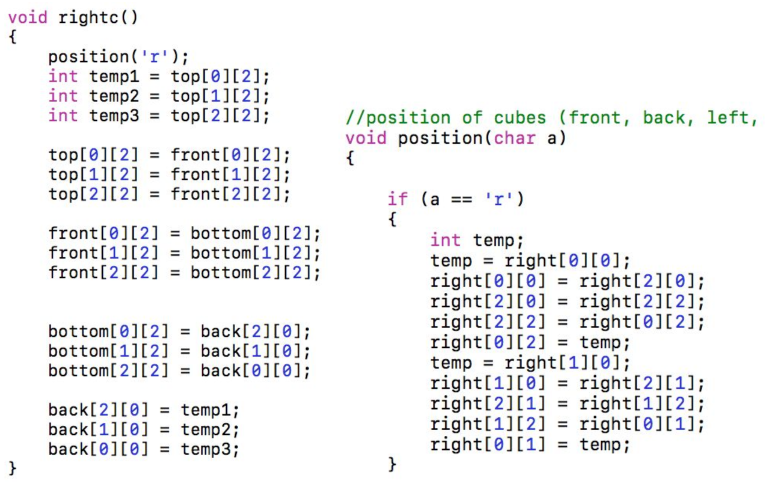
```
//creates border and panel for cubes
void panel(int a, int b, int c, int d, int e)
{
    //outline of panel
    glColor3f(0, 0, 0);
    glLineWidth(3.0);
    glBegin(GL_LINE_LOOP);

    //passes a pointer to an array of 3 float values
    glVertex3fv(vertices[b]);
    glVertex3fv(vertices[c]);
    glVertex3fv(vertices[d]);
    glVertex3fv(vertices[e]);
    glEnd();

    //1 panel
    glColor3fv(color[a]);
    glBegin(GL_POLYGON);
    glVertex3fv(vertices[b]);
    glVertex3fv(vertices[c]);
    glVertex3fv(vertices[d]);
    glVertex3fv(vertices[e]);
    glEnd();
}

//start creation of cubes, using panel function
// center piece
void cube1()
{
    panel(6, 0, 3, 2, 1);
    panel(6, 2, 3, 7, 6);
    panel(6, 0, 4, 7, 3);
    panel(6, 1, 2, 6, 5);
    panel(6, 4, 5, 6, 7);
    panel(6, 0, 1, 5, 4);
}
```

Figure 2. *Functions to draw cubes (uses GL_LINE_LOOP for border/outline and GL_POLYGON for cubes)*



```
void display()

{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glLoadIdentity();

    glColor3fv(color[0]);

    glPushMatrix();

    //to rotate the entire cube from starting pos.
    glRotatef(0.0 + p, 1.0, 0.0, 0.0);
    glRotatef(0.0 + q, 0.0, 1.0, 0.0);
    glRotatef(0.0 + r, 0.0, 0.0, 1.0);

//display colors, rotations (displaying 27 cubes at any time)
    if (rotation == 0)
    {
        cube1();
        cube2();
        cube3();
        cube4();
        cube5();
        cube6();
        cube7();
        cube8();
        cube9();
        cube10();
        cube11();
        cube12();
        cube13();
        cube14();
        cube15();
        cube16();
        cube17();
        cube18();
        cube19();
        cube20();
        cube21();
        cube22();
        cube23();
        cube24();
        cube25();
        cube26();
        cube27();
    }
```

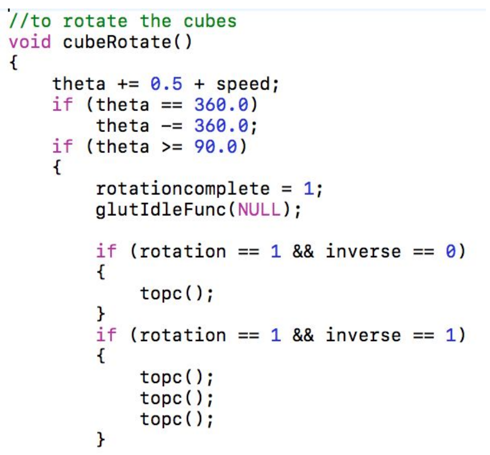
Figure 3. *Display function to display correct cubes and rotate cube*



```c
void rightc()
{
    position('r');
    int temp1 = top[0][2];
    int temp2 = top[1][2];
    int temp3 = top[2][2];

    top[0][2] = front[0][2];
    top[1][2] = front[1][2];
    top[2][2] = front[2][2];

    front[0][2] = bottom[0][2];
    front[1][2] = bottom[1][2];
    front[2][2] = bottom[2][2];


    bottom[0][2] = back[2][0];
    bottom[1][2] = back[1][0];
    bottom[2][2] = back[0][0];

    back[2][0] = temp1;
    back[1][0] = temp2;
    back[0][0] = temp3;
}
```

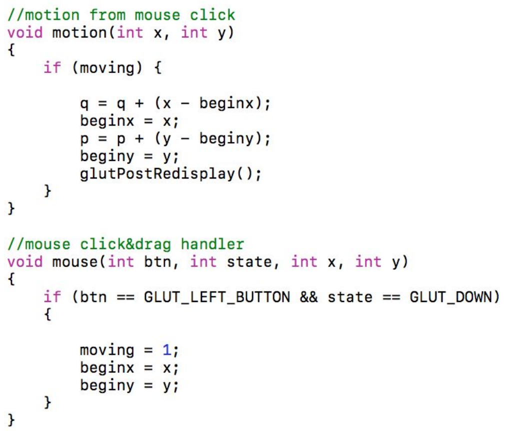
```c
//position of cubes (front, back, left,
void position(char a)
{

    if (a == 'r')
    {
        int temp;
        temp = right[0][0];
        right[0][0] = right[2][0];
        right[2][0] = right[2][2];
        right[2][2] = right[0][2];
        right[0][2] = temp;
        temp = right[1][0];
        right[1][0] = right[2][1];
        right[2][1] = right[1][2];
        right[1][2] = right[0][1];
        right[0][1] = temp;
    }
```

Figure 4. *Function for position of cubes/faces using arrays to swap data*



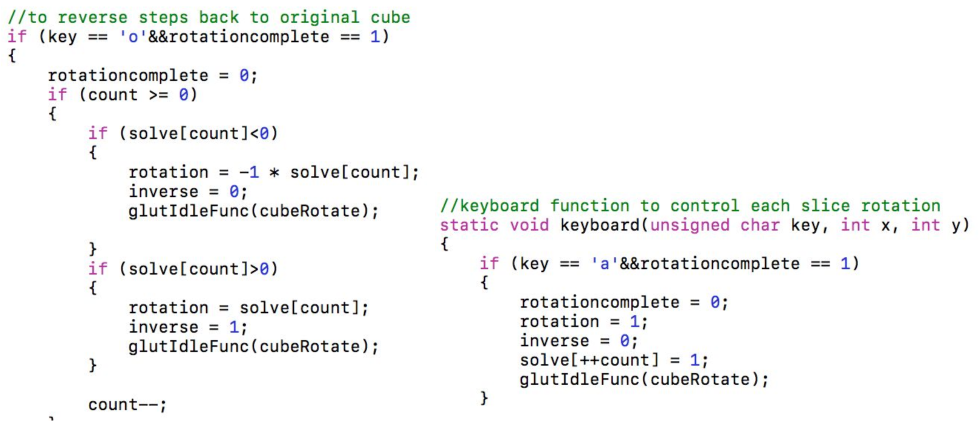
```c
//to rotate the cubes
void cubeRotate()
{
    theta += 0.5 + speed;
    if (theta == 360.0)
        theta -= 360.0;
    if (theta >= 90.0)
    {
        rotationcomplete = 1;
        glutIdleFunc(NULL);

        if (rotation == 1 && inverse == 0)
        {
            topc();
        }
        if (rotation == 1 && inverse == 1)
        {
            topc();
            topc();
            topc();
        }
```

Figure 5. *Function calls to rotate slices (wrap-around at 360 degrees).*



```c
//motion from mouse click
void motion(int x, int y)
{
    if (moving) {

        q = q + (x - beginx);
        beginx = x;
        p = p + (y - beginy);
        beginy = y;
        glutPostRedisplay();
    }
}

//mouse click&drag handler
void mouse(int btn, int state, int x, int y)
{
    if (btn == GLUT_LEFT_BUTTON && state == GLUT_DOWN)
    {

        moving = 1;
        beginx = x;
        beginy = y;
    }
}
```

Figure 6. *Function to detect mouse click and drag for cube rotation control*

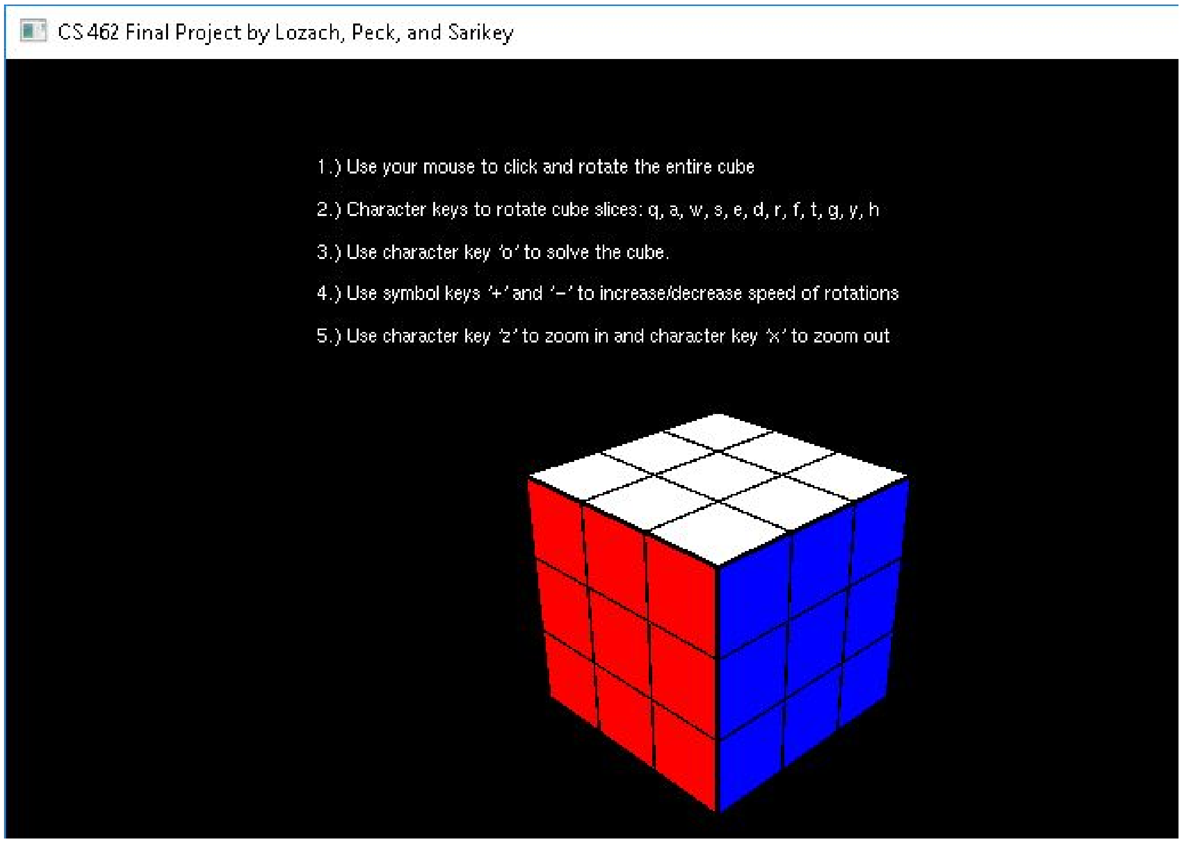


```
//to reverse steps back to original cube
if (key == 'o'&&rotationcomplete == 1)
{
    rotationcomplete = 0;
    if (count >= 0)
    {
        if (solve[count]<0)
        {
            rotation = -1 * solve[count];
            inverse = 0;
            glutIdleFunc(cubeRotate);

        }
        if (solve[count]>0)
        {
            rotation = solve[count];
            inverse = 1;
            glutIdleFunc(cubeRotate);
        }

        count--;
```

```
//keyboard function to control each slice rotation
static void keyboard(unsigned char key, int x, int y)
{
    if (key == 'a'&&rotationcomplete == 1)
    {
        rotationcomplete = 0;
        rotation = 1;
        inverse = 0;
        solve[++count] = 1;
        glutIdleFunc(cubeRotate);
    }
```

Figure 7. *Functions for keyboard input controls*

III. Final Output



CS 462 Final Project by Lozach, Peck, and Sarikey

1.) Use your mouse to click and rotate the entire cube

2.) Character keys to rotate cube slices: q, a, w, s, e, d, r, f, t, g, y, h

3.) Use character key 'o' to solve the cube.

4.) Use symbol keys '+' and '−' to increase/decrease speed of rotations

5.) Use character key 'z' to zoom in and character key 'x' to zoom out

Figure 8. *Completed Rubik's cube with instructions for user*