

Interactive Rubik's Cube with Texture Mapping and Shading

Jonathan Muckell

December 6, 2006

1 Project Summary

1. The Rubik's Cube (27 individual cubes has been drawn on the screen)
2. The entire Rubik's cube can be easily rotated
3. The user can select any of the 9 faces of the cube and rotate that face in any direction
4. A timer has been implemented that starts at 0:00
5. Texture mapping has been added to each cube face
6. Texture mapping has been added to the background
7. Two light sources have been placed at infinity, the properties of the light have been set
8. The material properties of the cube have been set
9. Shading has been implemented by using the normal from each cube face.

2 Directions

2.1 Rotate the Cube

1. CLICK and DRAG with mouse button. The Rubik's cube rotates on the screen

Implementation: The cube appears to rotate around a fixed axis. In the actual implementation, the observer is moved in a spherical shape around the origin. To the user, the cube appears to be rotating because the background always remains fixed.

2.2 Rotate a Face

1. PRESS any number from 1-9. Select the appropriate number that corresponds to the face you desire to rotate

2. PRESS ENTER to rotate the face in the positive direction OR PRESS - SPACE BAR to rotate the cube in the opposite direction

Implementation: A single set of curved arrows was written. The set of arrows is rotated and translated into the appropriate position. The arrows are drawn using line loops and a triangle. Also the coordinates are fixed in space and appear to be rotating because the observes position moves in a spherical shape. The 9 cubes that make up the face are rotated by directly manipulating their homogeneous coordinates, see below.

```
//X axis Rotation - Initializing Homogeneous Coordinates
if( k == X_LEFT || k == X_MIDDLE || k == X_RIGHT){
    m.l[0][0]=1;
    m.l[1][1]=cos(angle);
    m.l[1][2]=-(DIRECTION)*sin(angle);
    m.l[2][1]=(DIRECTION)*sin(angle);
    m.l[2][2]=cos(angle);
    //break;
}
```

2.3 Change the Cube's Texture

1. PRESS 't' for texture - This changes the surface of every face in the rubiks cube to be one of 3 different textures.

2.4 Change Background

1. PRESS 'b' for background - This changes the background to be one of three different bitmap images.

2.5 Randomize

1. PRESS 'r' - 20 random faces are rotated.

Implementation: The program randomly chooses on of the 9 possible faces to rotate and a random direction. By calling RCube.MakeMove(move, direction); twenty times.

2.6 Rotation Reversal

1. PRESS 's' for solve. This simple reverses all previous moves. It would be interesting to incorporate an algorithm quickly solves the cube in the fewest number of moves.
2. PRESS 'u' for undo. This undoes the previous move.

3 Lighting and Shading

There are two light sources positioned at infinity. The final color and brightness of each pixel on the cube is determined by the diffuse component of the light, the diffuse material property of the material, a small amount of white emission, and the shading. The shading is calculated by the cube's surface normal. Only the diffuse component of the light and material were used because it seems to produce the image that has the most identifiable colors for each cube face. This is important because the user needs to be able to clearly distinguish the colors in order to try and solve the cube.

The shading is determined by the normal from each cube's surface. Flat shading is used, instead of Phong shading, this is because flat shading is not only quicker, but better looking in this particular project. This is because with Phong shading the colors on the cube could become difficult to distinguish from each other. Although Flat shading does not look as realistic, it made the colors much easier to observe and the cube easier to manage.

```
void Draw()
{
    glColor3d(r,g,b);
    glEnable(GL_TEXTURE_2D);

    glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);
    glBindTexture(GL_TEXTURE_2D, textureName[texture]);

    diffuse[0] = r; diffuse[1] = g; diffuse[2] = b; diffuse[3] = 1.0;
    glMaterialfv(GL_FRONT_AND_BACK, GL_DIFFUSE, diffuse);

    glEnable(GL_LIGHTING);

    glBegin(GL_QUADS);
        GLfloat norm[3];
        cross(v[0],v[1],v[2],norm);
        glNormal3fv(norm);
        glVertex3d(v[0].x, v[0].y, v[0].z);    glTexCoord2f(0.0, 0.0);
        glVertex3d(v[1].x, v[1].y, v[1].z);    glTexCoord2f(0.0, 1.0);
        glVertex3d(v[2].x, v[2].y, v[2].z);    glTexCoord2f(1.0, 1.0);
        glVertex3d(v[3].x, v[3].y, v[3].z);    glTexCoord2f(1.0, 0.0);
    glEnd();

    glDisable(GL_LIGHTING);
    glDisable(GL_TEXTURE_2D);
}
```

Figure 1: The code segment implements all the texture, lighting, and shading for the Rubik's cube. This draws a single QUAD for one side of one cube.



Figure 2: Notice the shading on the left face. Having all the colors on the same face shade together gives the appearance of a shadow and still allows the user to easily identify the difference in the colors.

4 Texture Mapping

There are 3 different textures for the cube and 3 different background textures. The numerous cube and background textures account for the long initial run time. Any 24-bit bitmap image can be displayed as the texture or background. The image quality settings are set for the best quality since the performance penalty is manageable.

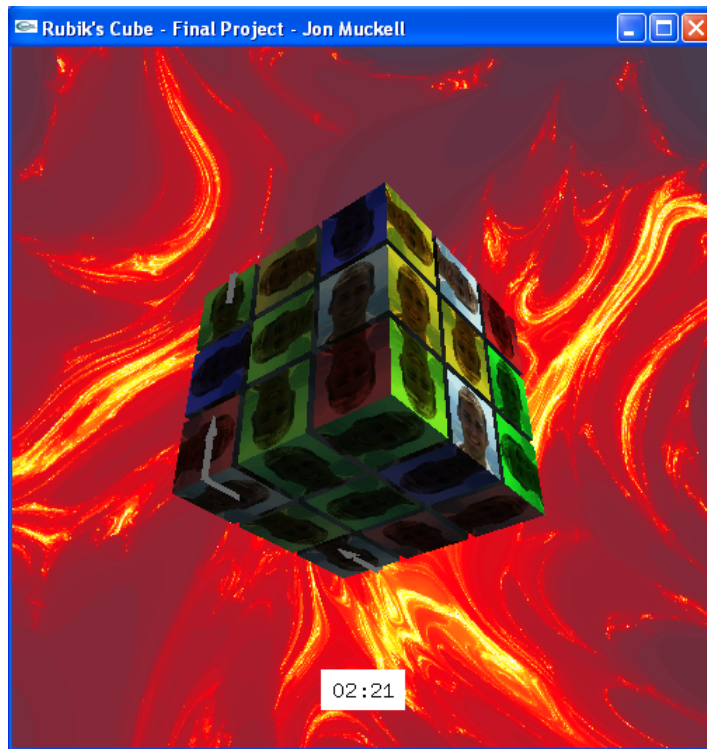


Figure 3: Any 24-bit bitmap can be easily set as the background or the cube texture.

5 Positioning and Moving the Observer

The observer can look at the cube from any possible angle, but is not able to flip the cube. This is done by moving the location of the observer in a nice smooth spherical manner. The observer is placed back at the initial starting position to draw the timer on the screen. This causes the timer to remain in a constant location while the cube is rotated.

```
glLoadIdentity();
GLfloat EYEX= 10*cos(EYE_A)*sin(EYE_B);
GLfloat EYFY = -10*cos(EYE_A)*cos(EYE_B);
GLfloat EYEZ = 10*sin(EYE_A);
gluLookAt (EYEX,EYFY,EYEZ,0,0,0,0,0,1);
```

6 Conclusion

I choose to code an interactive Rubik's cube because it allows a fun and flexible way to show off numerous graphics concepts. Matrix manipulation was needed for the rotation and translation of the cube's faces, as well as, moving of the

arrows. The faces are easily rotated when using a number pad, since each row of three numbers corresponds to the three arrow locations in that dimension. For implementing the timer, careful usage of the idle callback was needed. Understanding of how OpenGL handles lighting and shading was necessary in order to get the desired results.

My goal for this project was to implement a Rubik's cube similar to the conventional design (27 mini-cubes each with six different color faces) and combine this design with more advanced graphics concepts such as shading and texture mapping. These additional features allow the conventional Rubik's cube to be presented, in my opinion, an artistic fashion. Many of the software classes used in this project should be easy to modify and reuse for future graphics projects.