COMPUTER GRAPHICS

(UCS 505)

Project:

Rubik's Cube Solver

Submitted By:

Daksh Azad 102003145

Sudiksha Raheja 102003187

GROUP NO. 4

B.E. THIRD YEAR – COE

Submitted To:

Dr. Amrita Kaur

Computer Science and Engineering

Department Thapar Institute of

Engineering and Technology

Patiala – 147001

Table of Contents

1. INTRODUCTION

The Rubik's Cube is a classic puzzle game that has been popular for decades. It consists of a cube with six faces, each composed of nine smaller squares of different colors. The objective of the game is to solve the puzzle by rearranging the smaller squares so that each face of the cube has only one color.

In recent years, there has been growing interest in creating 3D simulations of the Rubik's Cube using computer graphics techniques. These simulations offer several advantages over physical cubes, such as the ability to manipulate the cube with greater ease and speed, as well as the ability to generate infinite variations of the cube.

In this project, we present a 3D simulation of the Rubik's Cube using various concepts of computer graphics. Our goal was to create a visually appealing and interactive simulation that accurately represents the mechanics of the Rubik's Cube.

We used a combination of programming languages and graphics libraries to develop the simulation, including C++, OpenGL, and GLSL. We also implemented several algorithms for cube manipulation, such as the basic layer-by-layer method.

In this report, we will discuss the design and implementation of our Rubik's Cube simulation in detail. We will also evaluate the performance of our simulation and discuss potential future directions for this project.

2. COMPUTER GRAPHICS CONCEPTS USED

1. **3D modeling** - The Rubik's Cube was modeled as a 3D object composed of smaller cubes. Each smaller cube was represented as a vertex in a 3D space, with its position and color determined by its location within the Rubik's Cube. This allowed us to create a realistic and accurate representation of the Rubik's Cube that could be easily manipulated in 3D space.

2. **Transformations** - In order to allow users to rotate all the faces of the cube, change the rotation speed, and rotate the entire cube, we utilized various transformations. This included translation, rotation, and scaling transformations. Translation was used to move the Rubik's Cube to different locations in 3D space. Rotation was used to change the orientation of the Rubik's Cube and its individual faces. Scaling was used to adjust the size of the Rubik's Cube and its individual cubes.

3. **Animation** - Animation involves creating the illusion of motion in a 3D object. In this project, animation is used to make the windmill blades rotate about the x, y, and z axes.

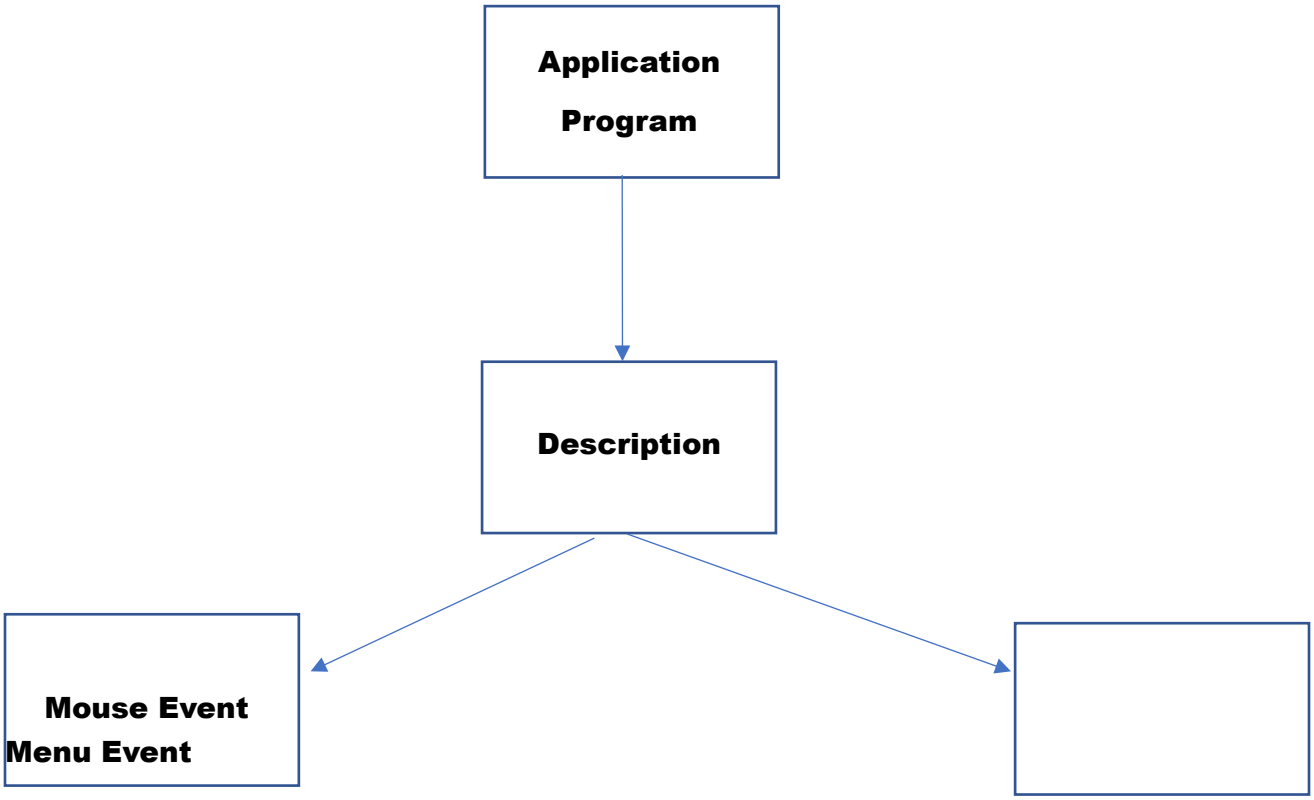
4. **User interaction** - User interaction involves allowing the user to control the windmill animation. In this project, the user can control the speed of the windmill blades using keyboard input.

5. **Texturing** - In order to apply the colors of the Rubik's Cube to the individual cubes, we utilized texturing techniques. We created a texture map that represented the colors of each face of the Rubik's Cube, and applied it to the individual cubes. This allowed us to create a realistic and accurate representation of the Rubik's Cube's color scheme.

Overall, these computer graphics concepts were essential for creating a visually appealing and interactive Rubik's Cube simulation. By utilizing these techniques, we were able to create a realistic and accurate representation of the Rubik's Cube that users can manipulate in real-time.

3. USER DEFINED FUNCTIONS

3(a) DESIGN



Mouse events:

When mouse event occurs, the ASCII code for the corresponding coordinates that generate the event and the location of mouse are returned.

Mouse callback function is:

glutMouseFunc (mouse);

Void mouse (int btn, int state, int x, int y) {

if (button == GLUT_LEFT_BUTTON && state == GLUT_DOWN)

begin = x;

}

Menu Entry:

GLUT provides one additional feature, pop_up menus, which we can use with the mouse to create sophisticated interactive application.

glutCreateMenu ();

glutAddMenuEntry ();

glutAttachMenu (GLUT_RIGHT_BUTTON);

3(b) ALGORITHM

STEP 1: Define vertices for 27 cubes which are used to compose one whole cube known as “Rubiks cube”.

STEP 2: Define colors for each cube of the Rubiks cube to distinguish one cube from the other and as well as color for the speed meter, which is used to control the speed of rotation.

Output( ) function:

STEP 3: This function is used to display the message using the Commands glutBitmapCharacter( ) and glRasterpos*( ).

Polygon( ) function:

STEP 4: Draw a polygon via list of vertices with the line of 3 pixels wide.

Colorcube( ) function:

STEP 5: Map vertices to faces. Hence the use of colorcube function has done 27 times with different prefixes for all the 27 cubes amongst which one with no color, 6 with one color, 12 with two colors and 8 with three colors on it.

Speedmeter( ) function:

STEP 6: This function is used to define vertices for a speedmeter, which is used to control the speed of rotation.

Display( ) function:

STEP 7: Clear the frame buffers and the z-buffer.

STEP 8: Invoke the function speedmeter and output.

STEP 9: Using the variables rotation and inverse the rotation for the faces are defined, where if rotation is one and the inverse flag is zero then the top face of the cube will be rotated in the clockwise direction and incase if rotation is one and also is the inverse flag then the top face of the cube will be rotated in the anti-clockwise direction.

STEP 10: The same procedure that has been specified in the step 9 is adopted with different values for the rotation, such as rotation with the value two is implemented for right three for front, four for left, five for back and six for bottom rotations respectively with the implementation of inverse variable being same.

STEP 11: Invoke the output function and swap the buffers.

Transpose( ) function:

STEP 12: This function is used to define the transpose for all the six faces.

Topc( ) function:

STEP 13: This function is used to assign the values when the operation is rotation of the top face.

Frontc( ) function:

STEP 14: This function is used to assign the values when the operation is rotation of the front face.

Rightc( ) function:

STEP 15: This function is used to assign the values when the operation is rotation of the right face.

Leftc( ) function:

STEP 16: This function is used to assign the values when the operation is rotation of the left face.

Backc( ) function:

STEP 17: This function is used to assign the values when the operation is rotation of the back face.

Bottomc( ) function:

STEP 18: This function is used to assign the values when the operation is rotation of the bottom face.

Spincube( ) function:

STEP 19: This is function is an idle callback which rotates the cube accordingly, if rotation is one and inverse is zero then rotate the top face of the cube by 90 degrees in the clockwise direction and if inverse is one then rotate the same face by 90 degree in the anti-clockwise direction.

STEP 20: The same procedure is implemented for the other faces of the cube with different rotation values.

STEP 21: The cube is redisplayed after the rotation.

Motion( ) function:

STEP 22: This is used to rotate the cube about the selected axis.

Mouse( ) function:

STEP 23: Mouse callback function. This allows user to give input through mouse buttons.

Keyboard( ) function:

STEP 24: use the keys ‘a’,’s’,’d’,’f’,’g’,’h’ to rotate the faces of the accordingly in the clockwise direction and the keys ‘q’,’w’,’e’,’r’,’t’,’y’ to rotate in the anti-clockwise direction.

STEP 25: use the keys ‘1’,’2’,’4’,’5’,’6’,’8’,’9’ are used to move the viewer along the axis.

STEP 26: use the keys ‘m’ and ‘n’ to alter the value of the rotation meter which controls the speed of the rotation.

STEP 27: use the key ‘o’ for the automatic solving of the cube.

Myreshape( ) function:

STEP 28: Define a viewport and set the matrix to projection and modelview matrices.

Mymenu( ) function:

STEP 29: Define the actions corresponding to each entry in the

menu.

Main( ) function:

STEP 30: Both double and z-buffer is enabled. Invoke the start function.

STEP 31: Add entries to the menu and link the menu to the right mouse button.

STEP 32: Stop.

4. CODE

```
#include <windows.h>
#include <GL/glut.h>
#include <string.h>
#include<stdio.h>
void *font = GLUT_BITMAP_TIMES_ROMAN_24;
char defaultMessage[] = "Rotation Speed:";
char *message = defaultMessage;
void
output(int x, int y, char *string)
{
 int len, i;
 glRasterPos2f(x, y);
 len = (int) strlen(string);
 for (i = 0; i < len; i++) {
 glutBitmapCharacter(font, string[i]);
 }
}
static float speed=0.0;
static int top[3][3]={{0,0,0},{0,0,0},{0,0,0}},
right[3][3]={{1,1,1},{1,1,1},{1,1,1}},
front[3][3]={{2,2,2},{2,2,2},{2,2,2}},
back[3][3]={{3,3,3},{3,3,3},{3,3,3}},
bottom[3][3]={{4,4,4},{4,4,4},{4,4,4}},
left[3][3]={{5,5,5},{5,5,5},{5,5,5}},
temp[3][3];
int solve[300];
int count=0;
int solve1=0;
static int rotation=0;
int rotationcomplete=0;
static GLfloat theta=0.0;
static GLint axis=0;
static GLfloat p=0.0,q=0.0,r=0.0;
```

```
GLfloat vertices[][3]={{-1.0,-1.0,-1.0},
 {1.0,-1.0,-1.0},
 {1.0,1.0,-1.0},
 {-1.0,1.0,-1.0}, //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,-3.0,-1.0},
 {1.0,-3.0,-1.0},
 {1.0,-1.0,-1.0},
 {-1.0,-1.0,-1.0}, //bottom center
 {-1.0,-3.0,1.0},
 {1.0,-3.0,1.0},
 {1.0,-1.0,1.0},
 {-1.0,-1.0,1.0},
```

```
void polygon(int a,int b,int c,int d,int e)
{
glColor3f(0,0,0);
glLineWidth(3.0);
glBegin(GL_LINE_LOOP);
glVertex3fv(vertices[b]);
glVertex3fv(vertices[c]);
glVertex3fv(vertices[d]);
glVertex3fv(vertices[e]);
glEnd();
glColor3fv(color[a]);
glBegin(GL_POLYGON);
glVertex3fv(vertices[b]);
glVertex3fv(vertices[c]);
glVertex3fv(vertices[d]);
glVertex3fv(vertices[e]);
glEnd();
}
void colorcube1()
{
polygon(6,0,3,2,1);
polygon(6,2,3,7,6);
polygon(6,0,4,7,3); // center piece
 polygon(6,1,2,6,5);
polygon(6,4,5,6,7);
polygon(6,0,1,5,4);
}
```

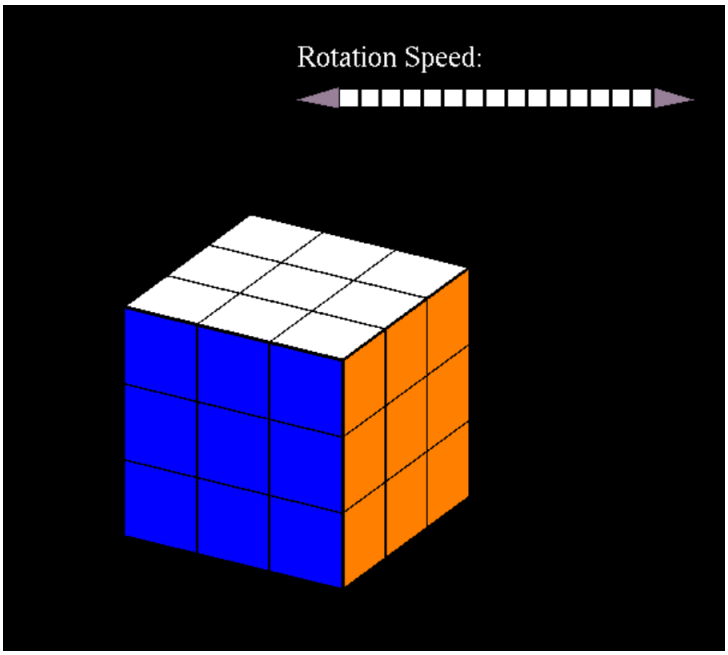
```
void speedmeter()
{
glColor3fv(color[7]);
glBegin(GL_POLYGON);
glVertex3f(0.0,7.2,0.0);
glVertex3f(1.0,7.0,0.0);
glVertex3f(1.0,7.5,0.0);
glEnd();
glPushMatrix();
glTranslatef(1.0,0.0,0.0);
polygon(speedmetercolor[0],216,217,218,219);
glPopMatrix();
```

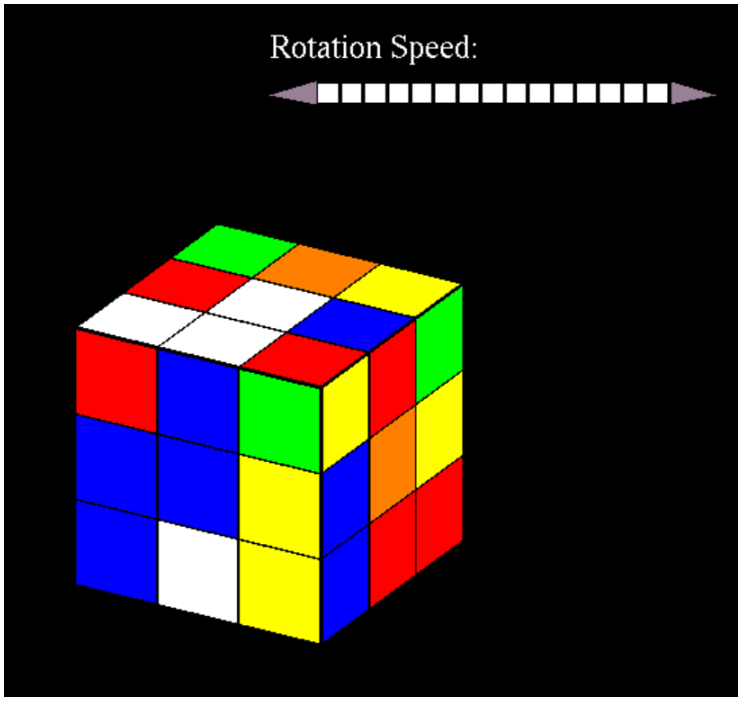
```
void transpose(char a)
{
if(a=='r')
{
int temp;
temp=right[0][0];
right[0][0]=right[2][0];
right[2][0]=right[2][2];
right[2][2]=right[0][2];
right[0][2]=temp;
temp=right[1][0];
 right[1][0]=right[2][1];
right[2][1]=right[1][2];
right[1][2]=right[0][1];
right[0][1]=temp;
}
if(a=='t')
{
int temp;
temp=top[0][0];
top[0][0]=top[2][0];
top[2][0]=top[2][2];
top[2][2]=top[0][2];
top[0][2]=temp;
temp=top[1][0];
 top[1][0]=top[2][1];
top[2][1]=top[1][2];
top[1][2]=top[0][1];
top[0][1]=temp;
}
```

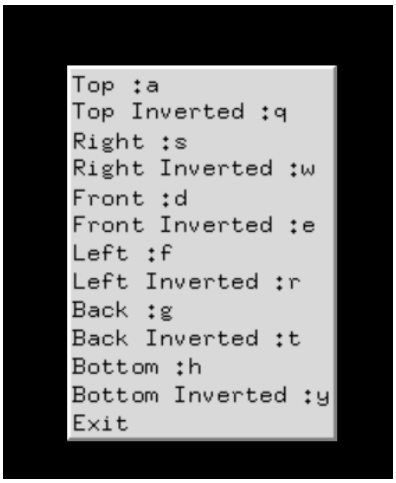
```
void myreshape(int w,int h)
{
glViewport(0,0,w,h);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
if (w <= h)
glOrtho(-10.0,10.0,-10.0*(GLfloat)h/(GLfloat)w, 10.0*(GLfloat)h/(GLfloat)w,-10.0,10.0);
else
glOrtho(-10.0*(GLfloat)w/(GLfloat)h, 10.0*(GLfloat)w/(GLfloat)h,-10.0,10.0,-10.0,10.0);
glMatrixMode(GL_MODELVIEW);
}
```

```
int main(int argc, char** argv)
{
glutInit(&argc, argv);
glutInitDisplayMode (GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
glutInitWindowSize (500, 500);
glutCreateWindow ("RUBIK'S CUBE");
glutReshapeFunc (myreshape);
glutIdleFunc(spincube);
glutMouseFunc(mouse);
glutMotionFunc(motion);
glutCreateMenu(mymenu);
glutAddMenuEntry("Top :a",1);
glutAddMenuEntry("Top Inverted :q",2);
glutAddMenuEntry("Right :s",3);
glutAddMenuEntry("Right Inverted :w",4);
glutAddMenuEntry("Front :d",5);
glutAddMenuEntry("Front Inverted :e",6);
glutAddMenuEntry("Left :f",7);
glutAddMenuEntry("Left Inverted :r",8);
glutAddMenuEntry("Back :g",9);
glutAddMenuEntry("Back Inverted :t",10);
glutAddMenuEntry("Bottom :h",11);
glutAddMenuEntry("Bottom Inverted :y",12);
glutAddMenuEntry("Exit",13);
glutAttachMenu(GLUT_RIGHT_BUTTON);
glutKeyboardFunc(keyboard);
glutDisplayFunc (display);
glEnable(GL_DEPTH_TEST);
glutMainLoop();
//return 0;
}
```

5. OUTPUT / SCREENSHOTS






Top :a
Top Inverted :q
Right :s
Right Inverted :w
Front :d
Front Inverted :e
Left :f
Left Inverted :r
Back :g
Back Inverted :t
Bottom :h
Bottom Inverted :y
Exit


Rotation Speed:

FUTURE POTENTIAL

1. Incorporating machine learning algorithms to generate new and challenging configurations of the Rubik's Cube.

2. Adding multiplayer functionality to allow users to compete against each other to solve the cube.

3. Configuring the program to solve the cube on it’s own as done in real life.