



**NITTE**  
(Deemed to be University)

**NMAM INSTITUTE  
OF TECHNOLOGY**

Nitte (DU) established under Section 3 of UGC Act 1956 | Accredited with 'A+' Grade by NAAC

---

# Report on Mini Project

## **RUBIK'S CUBE**

**Course Code : 20CS607**

**Course Name : Computer Graphics**

Semester : VI SEM

Section : C

### **Submitted To:**

Dr. Sannidhan M S

Department of Computer Science and Engineering

### **Submitted By :**

Raksha Kamath - 4NM20CS143

Rudradeep Roy - 4NM20CS147

Sharanya Rao - 4NM20CS163

**Signature of Course Instructor**

## **DECLARATION**

I hereby declare that the entire work embodied in this Mini Project titled, “RUBIKS CUBE” submitted to the Visvesvaraya Technological University, Belagavi has been carried out by us at the Department of CSE, NMAM Institute of Technology, Nitte under the supervision of Dr. Sannidhan M S , Assistant Professor, Department of CSE, NMAM Institute of Technology, Nitte. This thesis has not been submitted in part or full for the award of any diploma or degree of this or any other University.

## **ABSTRACT**

In this project we are rendering a 3D Rubik's cube using OpenGL with the help of C++.

The Rubik's cube consists of a cube with six faces , each made up of nine smaller squares of different colors. The objective of the puzzle is to scramble the cube and then solve it by restoring each face to a single color. The first is to solve the first layer of the cube by matching the edge and the corner pieces , then for the second layer we match the colors of the edge pieces with the colors of the centre pieces on the adjacent layers , for the top layer of the cube we match the four edge pieces with the centre piece and then we position the final layer edge pieces to match the colors of the adjacent center pieces and then finally manipulating the corner pieces of the last layer to match the colors of their adjacent sides as well to get the fully solved Cube.

In the program, there is a feature where the user can rotate the cube along the x-axis , y-axis , z-axis along with free movement of the rubik's cube according to the movement of the mouse pointer where the user is able to see the cube in different angles to observe which color of which edge or corner is where to be able to solve accordingly . There is also an added feature of changing the speed of the rotation of the sides.

As explained , the user needs to be able to manipulate and rotate the sides of the cube according to their own convenience hence the controls are added in the key board for the user to be able to rotate each side of the cube using the glutKeyboardFunc .

There is also another feature added where the user can reshuffle or reset the cube along with a timer and score in place, if the user decides to reshuffle and reset the cube then the timer and score also gets reset

along with it . Once the user starts to solve the cube , the score depends on how much time the user takes to complete the puzzle , initially the score starts with 10000.Once the user starts to solve the score keeps on decreasing as the time keeps increasing.

In this 3d Rubik's Cube project we will be making use of the libraries :  
<GL/glut.h> , <GL/glu.h> , <GL/gl.h> .

## ACKNOWLEDGEMENT

*We would like to give our sincere acknowledgement to everybody responsible for the successful completion of our project titled “ RUBIKS CUBE ”.*

*It gives us great pleasure to acknowledge with thanks the assistance and contribution of many individuals who have been actively involved at various stages of this project to make it a success. Firstly, we are very grateful to this esteemed institute “NMAM Institute of Technology ” for providing us an opportunity for our degree course. We wish to express our whole hearted thanks to our principal DR. NIRANJAN.N.CHIPLUNKAR for providing the modernized lab facilities in our institute. Success does not happen overnight, but hard work and dedication can achieve any goal. We have tried our level best to fulfill the requirements of the project, but I could not have achieved my goal without the able guidance of Dr. SANNIDHAN M S , Assistant Professor. We thank our guide for providing us an opportunity to work under their guidance and for their constant support and encouragement.*

*We thank one and all for helping us during our project .*

## TABLE OF CONTENTS

DECLARATION.....	2
ABSTRACT.....	3
ACKNOWLEDGEMENT.....	5
INTRODUCTION :.....	7
OBJECTIVES :.....	7
Non - Functional Requirements :.....	8
SOFTWARE Requirements :.....	8
HARDWARE Requirements :.....	8
DESIGN.....	9
IMPLEMENTATION.....	20
RESULTS AND DISCUSSIONS :.....	31

## **INTRODUCTION :**

- The Rubik's Cube is a 3D puzzle that has fascinated people for decades . It consists of a cube with six faces , each made up of nine smaller squares of different colors. The objective of the puzzle is to scramble the cube and then solve it by restoring each face to a single color.

## **OBJECTIVES :**

- From a users point of view, the objective of this program is to provide a virtual Rubik's Cube game that can be played on the computer . The program allows users to interact with a 3D representation of the Rubik's Cube , scramble it randomly , and then try to solve it by manipulating the cube's different faces.
- The program also provides a graphical interface that allows users to visualize the different algorithms used to solve the Rubik's Cube. This can be useful for beginners who are learning how to solve the puzzle , as well as for advanced players who are looking to improve their solving speed and accuracy.

## **Non – Functional Requirements :**

### **SOFTWARE Requirements :**

- CodeBlocks IDE or any IDE with a C or C++ compiler
- OpenGL
- Operating System , any OS which has Windows/Mac/Linux to run the program

### **HARDWARE Requirements :**

- Any modern Processor
- 2 GB RAM (min)
- Display with a minimum resolution of 800 x 600 pixels
- Keyboard and mouse
- Graphics Card



# **DESIGN**

glut functions used:

**glRasterPos2f(GLfloat x, GLfloat y)** : OpenGL maintains a 3-D position in window coordinates. This position, called the raster position, is maintained with subpixel accuracy. It is used to position pixel and bitmap write operations. x specifies the x-coordinate for the current raster position and y Specifies the y-coordinate for the current raster position.

**glutBitmapCharacter(void \*font, int character):** glutBitmapCharacter renders a bitmap character using OpenGL.

**glLineWidth(GLfloat width):** glLineWidth — specify the width of rasterized lines.

**glTranslatef(GLfloat x, GLfloat y, GLfloat z):** Calling glTranslatef(x, y, z) will compose the current transformation with a translation that takes the origin to (x,y,z).

**glBegin(GLenum mode), glEnd(void):** Specifies the primitive or primitives that will be created from vertices presented between glBegin and the subsequent glEnd .

**glClear(GLbitfield mask):**Clears the specified buffers to their current clearing values.

**glColor3f(GLfloat red,GLfloat green,GLfloat blue):** it can be used to give each vertex its own color.The GL stores both a current single-valued color index and a current four-valued RGBA color. It specify new red, green, and blue values explicitly and sets the current alpha value to 1.0 (full intensity) implicitly.

**glColor3fv( const GLfloat \*v ):** Specifies a pointer to an array that contains red, green, blue, and (sometimes) alpha values.

**glRotatef(GLfloat angle, GLfloat x,GLfloat y, GLfloat z):** The glRotatef function computes a matrix that performs a counterclockwise rotation of angle degrees about the vector from the origin through the point (x, y, z).

**glPushMatrix( void), void glPopMatrix( void):** The glPushMatrix and glPopMatrix functions push and pop the current matrix stack.

**glFlush(void):** Different OpenGL implementations buffer commands in several different locations, including network buffers and the graphics accelerator itself. The glFlush function empties all these buffers, causing all issued commands to be executed as quickly as they are accepted by the actual rendering engine. Though this execution may not be completed in any particular time period, it does complete in a finite amount of time.

**glViewport(GLint X, GLint Y, GLsizei Width, GLsizei Height):** The glViewport subroutine specifies the affine transformation of X and Y from normalized device coordinates to window coordinates.

**glLoadIdentity(void)** :glLoadIdentity replaces the current matrix with the identity matrix. It is semantically equivalent to calling glLoadMatrix with the identity matrix.v

**GLfloat:** GLfloat is a data type used in OpenGL to represent single-precision floating-point values with a guaranteed minimum range of values. It is typically used to specify coordinates, colors, and other parameters in OpenGL functions.

**glOrtho(GLdouble Left, GLdouble Right, GLdouble Bottom, GLdouble Top, GLdouble Near, GLdouble Far):** The glOrtho subroutine describes a perspective matrix that produces a parallel projection. (Left, Bottom, -Near) and (Right, Top, -Near) specify the points on the near clipping plane that are mapped to the lower left and upper right corners of the window, respectively, assuming that the eye is located at (0, 0, 0).

**glMatrixMode(GLenum cap):** Specifies which matrix stack is the target for subsequent matrix operations. Three values are accepted: GL\_MODELVIEW, GL\_PROJECTION, and GL\_TEXTURE.

**glEnable(GLenum cap) :** The glEnable function enables OpenGL capabilities.

**glutSwapBuffers(void):** Performs a buffer swap on the *layer in use* for the *current window*. Specifically, glutSwapBuffers promotes the contents of the back buffer of the *layer in use* of the *current window* to become the contents of the front buffer. The contents of the back buffer then become undefined

**glutPostRedisplay(void):** Marks the normal plane of *current window* as needing to be redisplayed. The next iteration through `glutMainLoop`, the window's display callback will be called to redisplay the window's normal plane. Multiple calls to `glutPostRedisplay` before the next display callback opportunity generates only a single redisplay callback. `glutPostRedisplay` may be called within a window's display or overlay display callback to re-mark that window for redisplay.

**glutInit(int \*argc, char \*\*argv):** `glutInit` is used to initialize the GLUT library.

**glutInitDisplayMode(unsigned int mode):** The *initial display mode* is used when creating top-level windows, subwindows, and overlays to determine the OpenGL display mode for the to-be-created window or overlay.

**glutInitWindowSize(int width, int height):** Windows created by `glutCreateWindow` will be requested to be created with the current *initial window position* and *size*.

**glutCreateWindow(char \*name):** `glutCreateWindow` creates a top-level window. The name will be provided to the window system as the window's name. The intent is that the window system will label the window with the name.

**glutReshapeFunc(void (\*func)(int width, int height)):** `glutReshapeFunc` sets the reshape callback for the *current window*. The reshape callback is triggered when a window is reshaped.

**glutIdleFunc(void (\*func)(void)):** glutIdleFunc sets the global idle callback to be func so a GLUT program can perform background processing tasks or continuous animation when window system events are not being received.

**glutTimerFunc(unsigned int msec, void (\*func)(int value), value):** glutTimerFunc registers the timer callback func to be triggered in at least msec milliseconds. The value parameter to the timer callback will be the value of the value parameter to glutTimerFunc. Multiple timer callbacks at same or differing times may be registered simultaneously.

**glutMouseFunc(void (\*func)(int button, int state, int x, int y)):** glutMouseFunc sets the mouse callback for the *current window*. When a user presses and releases mouse buttons in the window, each press and each release generates a mouse callback.

**glutMotionFunc(void (\*func)(int x, int y)):** The motion callback for a window is called when the mouse moves within the window while one or more mouse buttons are pressed.

**glutCreateMenu(void (\*func)(int value)):** glutCreateMenu creates a new pop-up menu and returns a unique small integer identifier. The range of allocated identifiers starts at one. The menu identifier range is separate from the window identifier range. Implicitly, the *current menu* is set to the newly created menu.

**glutAddMenuEntry(char \*name, int value):** glutAddMenuEntry adds a menu entry to the bottom of the *current menu*.

**glutAttachMenu(int button):** glutAttachMenu attaches a mouse button for the *current window* to the identifier of the *current menu*.

**glutKeyboardFunc(void (\*func)(unsigned char key, int x, int y)):** glutKeyboardFunc sets the keyboard callback for the *current window*.

**glutDisplayFunc(void (\*func)(void)):** glutDisplayFunc sets the display callback for the *current window*.

**glutMainLoop(void):** glutMainLoop enters the GLUT event processing loop.

### **User-defined functions used:**

**void polygon(int a,int b,int c,int d,int e):** The above code defines a function called "polygon" that takes in five indices of vertices and uses them to draw a filled polygon of a specified color and a outline with a thickness of 3.0 using OpenGL graphics library.

**void colorcube1() to void colorcube27():** The function colorcube1() and so on upto colorcube27() is used to draw a Rubik's cube with each face of the cube colored using polygons. The colors are defined using glColor3fv() and color[] array. The function calls polygon() to draw each face of the cube with the specified colors and vertex indices.

**void speedmeter():** The code defines a function called speedmeter() that draws a speedometer using polygons of different colors. The speedometer consists of a black base and 15 color-coded segments, each with a different color. The

function uses the `polygon()` function to draw each segment, and also draws a black pointer at the top of the speedometer.

**`void display()`:** The display function is responsible for rendering the cube and displaying various text messages on the screen such as the current score, time, and control instructions.

**`void transpose()` :** defines a function `transpose()` that takes a character as input and performs a specific type of operation on one of six 3x3 arrays (top, bottom, left, right, front, or back) depending on the character input.

The operation performed is a transpose of the 3x3 array. It swaps the elements at certain positions in the array to reflect the transpose operation.

Each array is accessed and modified in the same way.

**`void topc()`, `void frontc()`, `void backc()`, `void right()`, `void leftc()`, `void bottomc()`:** implement the clockwise rotation of the Rubik's Cube faces. Each function corresponds to a particular face and updates the values of the cubelets on that face and the adjacent faces. The code uses temporary variables to store the values of the cubelets that need to be updated, and then updates them according to the rotation. The transpose function is called at the beginning of each function to perform the necessary matrix transpose operation.

**`void shuffle_cube()`:** The function `shuffle_cube()` shuffles a Rubik's cube by applying a random sequence of moves.

**void reset\_cube():** The reset\_cube() function sets the initial state of the Rubik's Cube. It initializes the colors of each face of the cube to their default state.

**void spincube():** The spincube() function is responsible for rotating the cube in response to user input. It updates the angle of rotation (theta) and checks if the rotation is complete by checking if the angle has reached 90 degrees. If the rotation is complete, it sets the rotationcomplete flag, stops the idle function and applies the appropriate cube rotation function (topc(), bottomc(), frontc(), backc(), leftc(), rightc()) based on the rotation and inverse flags. These flags are set by the keyboard() function which handles user input. Once the cube has been rotated, the rotation and inverse flags are reset to their default values and the theta angle is reset to zero. Finally, glutPostRedisplay() is called to update the display.

**void motion():** This is a callback function for handling mouse motion events in a GLUT-based program. The function takes two integer arguments x and y representing the current position of the mouse cursor on the screen. The function first checks if the variable moving is true, which indicates that the left mouse button is being held down and the user is dragging the cube. If moving is true, the function calculates the difference between the current mouse position and the previous mouse position (x-beginx and y-beginy, respectively) and adds these differences to the variables q and p, respectively. The variables q and p are used to keep track of the rotation angles of the cube around the x and y axes, respectively. Finally, the function calls glutPostRedisplay() to signal GLUT to redraw the scene with the updated cube rotation angles.



**void mouse():** The mouse() function is a callback function for handling mouse events in OpenGL. It takes four arguments - btn, state, x, and y - which correspond to the button pressed, the state of the button (down or up), and the x and y coordinates of the mouse when the event occurred.

If the middle button is pressed down, the function does nothing.

If the left button is pressed down, the moving variable is set to 1, indicating that the user is dragging the mouse. The current x and y coordinates of the mouse are stored in beginx and beginy, respectively, so that they can be used to calculate the amount of movement when the mouse is dragged. Finally, glutPostRedisplay() is called to redraw the scene with the new position of the cube.

**void keyboard():** A function that handles keyboard inputs for a Rubik's Cube simulation program. It has a series of if statements that check for certain keys and set various parameters depending on the key pressed.

For example, if the 'a' key is pressed and the rotation of the cube is complete, the rotationcomplete flag is set to 0, indicating that a rotation is in progress. The rotation variable is set to 1, indicating that the first layer is being rotated, and the inverse variable is set to 0, indicating that it is not an inverted rotation. The count variable is incremented, and the rotation command is added to the solve array. Similar if statements handle the other possible rotation keys and adjust the rotation, inverse, count, and solve variables accordingly.

Also there are if statements that handle certain other keys. For example, pressing the '2' key increases the p variable by 2.0, while pressing '8' decreases it by 2.0. Pressing 'm' increases the speed variable by 0.3, up to a maximum of 4.2, and pressing 'n' decreases it by 0.3. Pressing 'o' performs an undo operation, if there is a rotation in the solve array to undo.

This function handles the user input for the Rubik's Cube simulation program and update the appropriate variables based on the user's actions.

**void myreshape()** :Handles window resizing. It is called whenever the window is resized and takes the new width and height of the window as input parameters. The function first sets the viewport to cover the entire window using `glViewport()`. Then it switches to the projection matrix using `glMatrixMode(GL_PROJECTION)` and sets it to the identity matrix using `glLoadIdentity()`.

Next, it sets up an orthographic projection using `glOrtho()` depending on the aspect ratio of the window. If the window is wider than it is tall, it uses the width to calculate the left and right planes of the frustum, and if it is taller than it is wide, it uses the height to calculate the top and bottom planes of the frustum. In both cases, it sets the near and far planes of the frustum to -10 and 10, respectively.

Finally, it switches back to the modelview matrix using `glMatrixMode(GL_MODELVIEW)`.

**void mymenu()** :This code defines a function `mymenu` which is used as a callback function for the GLUT menu. The function takes an integer argument `id` which represents the ID of the menu item that was selected by the user.

The function first checks if the rotation of the cube is complete, which is determined by the global variable `rotationcomplete`. If it is not complete, the function does nothing. If it is complete, the function sets `rotationcomplete` to 0 and performs a switch case on the `id` argument. For each case, the function sets the global variables `rotation` and `inverse` based on the selected menu item, which are used to perform a rotation on the cube. It also updates the `solve` array with the selected move and increments the count variable. Finally, if the

rotation is complete, the function sets the `glutIdleFunc` to the `spincube` function, which continuously rotates the cube until the next menu item is selected.

**void timer()** : This is a function called "timer" that uses the GLUT library to repeatedly call itself every 1000 milliseconds (1 second) using the "`glutTimerFunc`" function. Each time it is called, the "`timerValue`" variable is incremented by 1 and "`glutPostRedisplay`" function is called to mark the current window as needing to be redisplayed. This function is typically used to update the screen display at a regular interval, such as for implementing an animation or a timer in a game.

**int main()** : The main function is the starting point of the program and contains all the code that needs to be executed. In this specific program, the main function first prompts the user to enter the difficulty level of the Rubik's Cube they want to solve. It then creates the window for the Rubik's Cube simulation using the GLUT library and sets various functions to handle input events and display updates.

The `myreshape` function is called whenever the window is resized, `spincube` is called continuously to animate the cube, mouse and motion functions handle mouse input, `mymenu` function creates a menu for the user to interact with, and keyboard function handles keyboard input. The `shuffle_cube` function initializes the Rubik's Cube by shuffling its faces randomly. The `display` function is called to draw the Rubik's Cube, and `timer` function increments the timer value used to calculate the time taken to solve the Rubik's Cube. `glutMainLoop()` is called to start the GLUT event processing loop, which waits for input events and redraws the window as necessary. The program continues running until the user chooses to exit by selecting the "Exit" option from the menu.

# IMPLEMENTATION

Some of the features implemented for the rubik's cube using openGL:

1. Difficulty level
2. Menu
3. Rotation controls- X,Y and Z axis
4. Rotation speed: decrease (n) , increase (m)
5. User-controls-Shuffle cube, Reset cube
6. Timer
7. Score

## 1. Difficulty level:

```
int difficulty_level = 0;
void shuffle_cube()
{
    srand(time(NULL));

    int i;
    for (i = 0; i < difficulty_level; i++) {
        int choice = rand() % 5;
        switch (choice) {
            case 0:
                bottomc();
                break;
            case 1:
                backc();
                break;
            case 2:
                leftc();
                break;
            case 3:
                rightc();
                break;
            case 4:
                frontc();
                break;
        }
    }
}
```

This code snippet defines a function called `shuffle_cube` that is used to shuffle the Rubik's cube randomly according to the `difficulty_level` specified. The

function first seeds the random number generator using the current time using the `srand` function.

It then executes a loop that runs `difficulty_level` times. Within the loop, it generates a random number between 0 and 4 using the `rand` function and uses a switch statement to perform a random rotation on the Rubik's cube using one of the five possible rotation functions: `bottomc()`, `backc()`, `leftc()`, `rightc()`, or `frontc()`. These functions perform a rotation on the corresponding side of the Rubik's cube by swapping the colors of the cubes in a particular pattern.

## **2. Menu:**

The menu for operating the features was done using `glutCreateMenu(void (*func)(int value))` and `glutAddMenuEntry(char *name, int value)` in the main function.

`glutCreateMenu(void (*func)(int value))` is a function in the GLUT library that creates a new pop-up menu and sets the callback function to be executed when a menu item is selected. The `func` argument is a pointer to the callback function that takes an integer value as its parameter.

`glutAddMenuEntry(char *name, int value)` is a function in the GLUT library that adds a new item to a previously created pop-up menu. The `name` argument specifies the name of the new item, and the `value` argument specifies the value that will be passed to the callback function when the item is selected. The value can be used to determine which menu item was selected and to perform the appropriate action.

```

glutCreateMenu(mymenu);
shuffle_cube();
glutAddMenuEntry("INSTRUCTIONS",0);
glutAddMenuEntry("-----",0);
glutAddMenuEntry("a: Top",1);
glutAddMenuEntry("q: Top Inverted",2);
glutAddMenuEntry("s: Right",3);
glutAddMenuEntry("w: Right Inverted",4);
glutAddMenuEntry("d: Front",5);
glutAddMenuEntry("e: Front Inverted",6);
glutAddMenuEntry("f: Left",7);
glutAddMenuEntry("r: Left Inverted",8);
glutAddMenuEntry("g: Back",9);
glutAddMenuEntry("t: Back Inverted",10);
glutAddMenuEntry("h: Bottom",11);
glutAddMenuEntry("y: Bottom Inverted",12);
glutAddMenuEntry("o: Reverse move",0);
glutAddMenuEntry("",0);
glutAddMenuEntry("ROTATION CONTROLES:",0);
glutAddMenuEntry("-----",0);
glutAddMenuEntry("4 & 6: X-Axis",0);
glutAddMenuEntry("2 & 8: Y-Axis",0);
glutAddMenuEntry("1 & 9: Z-Axis",0);
glutAddMenuEntry("5: Origin",0);
glutAddMenuEntry("",0);
glutAddMenuEntry("ROTATION SPEED:",0);
glutAddMenuEntry("-----",0);
glutAddMenuEntry("m: Increase",0);
glutAddMenuEntry("n: Decrease",0);
glutAddMenuEntry("",0);
glutAddMenuEntry("USER CONTROLS",0);
glutAddMenuEntry("-----",0);
glutAddMenuEntry("Shuffle_cube",13);

```

We make use of the user-defined function void mymenu(int id) which is a callback function that is called when a menu item is selected in a GLUT menu. The function takes an integer argument "id" that represents the id of the selected menu item. The function then performs a switch-case operation to determine which menu item was selected and performs the appropriate action based on that selection. These actions may include setting rotation flags, calling other functions such as "spincube", "shuffle\_cube", or "reset\_cube", or exiting the program.

### 3. Rotation controls- X,Y and Z axis:

```

case 0: glutIdleFunc(spincube);
        break;

```

In the main function, we've given case 0 for X,Y,Z axis and origin orientation of the cube in `glutAddMenuEntry` function. When the `mymenu` function is called, it executes the code snippet for case 0.

When the user selects menu item 0, the `glutIdleFunc` function is called with `spincube` as its argument. `glutIdleFunc` sets the global idle callback function, which is executed when the event loop is idle. In this case, the `spincube` function is passed as the idle callback, so when the event loop is idle, the `spincube` function will be called repeatedly.

The keyboard function is called when user presses corresponding to X,Y,Z axis and origin, ie; 4 & 6 keys for X axis, 2 & 8 keys for Y axis, 1 & 9 keys for Z axis, and 5 key for origin orientations.

If the user presses the '2', '4', '6', '8', '9', '1', or '5' key, and `rotationcomplete` variable is equal to 1, the function will adjust the values of `p`, `q`, or `r`, which are used to control the position of the cube.

One such code snippet is:

```
if(key=='4'&&rotationcomplete==1)
{
    q=q-2.0;
    glutIdleFunc(spincube);
}
```

The purpose of this code could be to allow the user to trigger a rotation of a cube by pressing the '4' key, but only if a previous rotation has completed. The value of "q" hold the current position of the Rubik's Cube during the rotation animation. They are used to update the position of the cube during the solving process.

Spincube function: This function is responsible for spinning the cube during the animations when the user makes a move. It updates the rotation angle (theta) by adding 0.5 plus the current speed to it. If theta reaches 360.0, it is set back to 0.0.

Once the rotation angle reaches 90.0, rotationcomplete variable is set to 1, and glutIdleFunc is set to NULL, which stops the animation. Then, depending on the rotation and inverse values set in mymenu function, one of the cube's faces is rotated using the corresponding rotation function (topc, rightc, etc.) and the solve array is updated with the current move.

After the rotation is completed, rotation is set back to 0, theta is set back to 0, and glutPostRedisplay is called to redraw the cube.

#### **4. Rotation speed: decrease (n) , increase (m):**

When user presses value m on keyboard, then the speedmeter speed increases;

The below code snippet in the keyboard function says:

static int speedmetercolor[15]={0,0,0,0,0,0,0,0,0,0,0,0,0,0,0}: It is used to store the color values for a speed meter, indicating the rotation speed of the Rubik's Cube.

static int speedmetercount=-1: used as a counter to keep track of the number of speed meter colors filled, indicating the rotation speed of the Rubik's Cube.

When the user presses m on the keyboard, based on the speed value it executes the first if statement or the second if statement in the code snippet.

If speed value is greater than 1.3 and below 2.9 or equal; then add 0.3 to speed value and based on the speedmetercount value as index value of the array speedmetercolor (which holds the color of the speedmeter as array elements),



then assign value 4 to it. This displays the color corresponding to color 4 from the speedmetercolor array thus displaying the corresponding speed to the color.

Similarly if speed value is greater than 2.9 and lesser or equal to 4.2, then increment speed value by 0.3 and assigns 5 value to that array index of speedmetercolor array thus displaying the corresponding speed to the color.

```
if(key=='m' && rotationcomplete==1)
{
    if(speed>1.3)
    {
        if(speed<=2.9)
        {
            speed=speed+0.3;
            speedmetercolor[++speedmetercount]=4;
        }
    }
    glutPostRedisplay();
}
if(key=='m' && rotationcomplete==1)
{
    if(speed>2.9)
    {
        if(speed<=4.2)
        {
            speed=speed+0.3;
            speedmetercolor[++speedmetercount]=5;
        }
    }
    glutPostRedisplay();
}
```

Similarly to the above code snippet when the user presses key n, speedmeter speed decreases and corresponding if statement in the keyboard function is executed and also giving color to the speedmeter when speed decreases.

**Speedmeter function :**

This is a function called speedmeter() which draws a speedometer display on the screen using OpenGL commands.

The function first sets the color to the 7th color in the color array using glColor3fv(). It then draws a polygon with glBegin() and glEnd() functions to create the background of the speedometer display.

The function then proceeds to draw a series of polygons at specific coordinates using glPushMatrix() and glPopMatrix() commands to set the current matrix to a new one, perform transformations on it and then restore the previous one. Each polygon has a specific color which is obtained from the speedmetercolor array at a specific index. The polygon() function is called with the specific color index and four vertex coordinates as arguments to draw each polygon.

The speedmeter() function ends by drawing another polygon for the needle of the speedometer with the same glBegin() and glEnd() commands.

## 5. User-controls- Shuffle cube, Reset cube:

### Shuffle cube:

glutAddMenuEntry("Shuffle cube",13); This code snippet is executed when user presses on the shuffle cube option in the menu. The mymenu function is called and it executes this case 13:

```
case 13 :  
    shuffle_cube();  
    timerValue = 0;  
    glutIdleFunc(spincube);  
    break;
```

The shuffle\_cube function is called to shuffle the cube randomly and a glutIdleFunc for spincube function is called as well.

glutIdleFunc(spincube) is a GLUT function call that sets the idle callback function to spincube.

In GLUT, the idle callback function is called when there are no events to handle, meaning that the application is idle. By setting spincube as the idle callback function, the cube will continuously rotate as long as the application is idle, which gives the illusion of the cube being in motion.

#### Reset cube:

glutAddMenuEntry("Reset cube",14); This code snippet is executed when user presses on the reset cube option in the menu. The mymenu function is called and it executes this case 14:

```
case 14 :  
    reset_cube();  
    timerValue = 0;  
    f=false; // should stop the count value when its called  
    glutIdleFunc(spincube);  
    break;
```

The reset\_cube function is called to reset the cube and a glutIdleFunc for spincube function is called as well.

The reset\_cube() function sets the initial state of the Rubik's Cube. It initializes the colors of each face of the cube to their default state.

It uses a nested for loop to iterate over each of the 3x3 arrays that represent each face of the cube. For each position in the array, it sets the value to the appropriate color. For example, for the right face, which is initially set to all red (1), the loop sets each position in the array to 1. Similarly, the front face is initially set to all green (2), and the back face is initially set to all blue (3).

By calling this function, the cube is restored to its original state, allowing the player to start a new game.

## 6. Timer:

`int timerValue = 0;` This stores the value of a timer used for controlling the rotation speed of the Rubik's Cube.

From the below code snippet we infer:

```
void timer(int value)
{
    timerValue++;
    glutPostRedisplay();
    glutTimerFunc(1000, timer, 0);
}
```

This function `timer` is a callback function that is called repeatedly at a fixed interval by `glutTimerFunc`. It takes an integer parameter `value` which is not used in the function.

Inside the function, a global variable `timerValue` is incremented by 1. Then, `glutPostRedisplay()` is called to update the display and redraw the scene. Finally, `glutTimerFunc(1000, timer, 0)` is called to register the timer function to be called again in 1000 milliseconds (1 second).

This function is used to create a timer in the program that increments a counter by 1 every second and updates the display.

From the below code snippet we infer:

```
char timerString[32];
sprintf(timerString, "Timer: %d", timerValue);
output(-9, 8, timerString);
```

In the display function, the `sprintf` function is used to format the timer value as a string, and then the `output` function is called to render the string on the

screen. The position (-9, 8) is specified, which means the text will be rendered near the top-left corner of the screen.

According to the code snippet below:

```
for (int i = 0; i < 32; i++)  
    glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18, timerString[i]);
```

This code snippet is an alternative implementation of the output() function for drawing the timerString on the screen.

It uses the glutBitmapCharacter() function to draw each character of the string individually, using the GLUT\_BITMAP\_HELVETICA\_18 font.

This implementation says that timerString is a null-terminated string with a maximum length of 32 characters.

Output function: As seen from the code snippet below:

```
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]);  
    }  
}
```

The output function is used for rendering text on the screen using GLUT's built-in bitmap fonts. The function takes in three arguments: x and y specify the position where the text should be rendered, and string is a pointer to the null-terminated string that contains the text to be rendered. The function calculates the length of the string using strlen. For each character in the string, the function calls glutBitmapCharacter to render the character at the current raster position (specified by x and y). The function uses a global variable font

to specify which bitmap font to use for rendering. This variable should be set before calling the function.

## 7. Score:

int score = 100000; used to store the current score of the rubik's cube game.

According to the code snippet below:

```
char scoreString[32];  
sprintf(scoreString, "Score: %d", score - timerValue * 10);  
output(-9, 7, scoreString);
```

This code segment is creating a string that displays the player's score, which is calculated by subtracting ten times the timer value from the score. The sprintf function is used to format the string with the current score value, and then the output function is called to display the string on the screen.

The char variable scoreString is declared to hold the formatted string.

The sprintf function is called with three arguments: scoreString, the format string "Score: %d", and the value to be inserted into the format string (score - timerValue \* 10). This calculates the player's score based on the current score and timerValue values and formats it as a string with the label "Score: ".

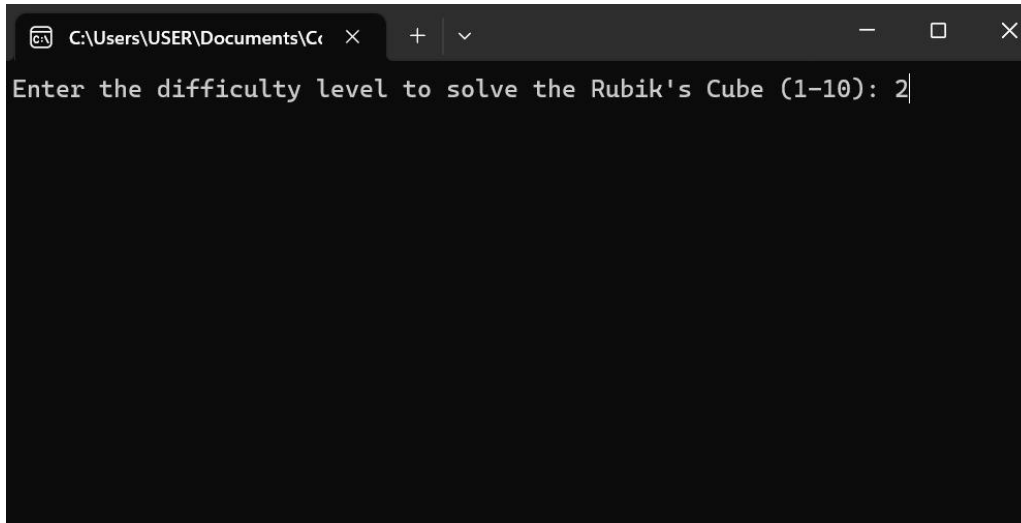
The output function is called with three arguments: -9 and 7, which specify the location on the screen to display the string, and scoreString, which contains the string to display.

This code is displaying the player's score on the screen and updating it based on the timer value.

## RESULTS AND DISCUSSIONS :

- **Running the program :**

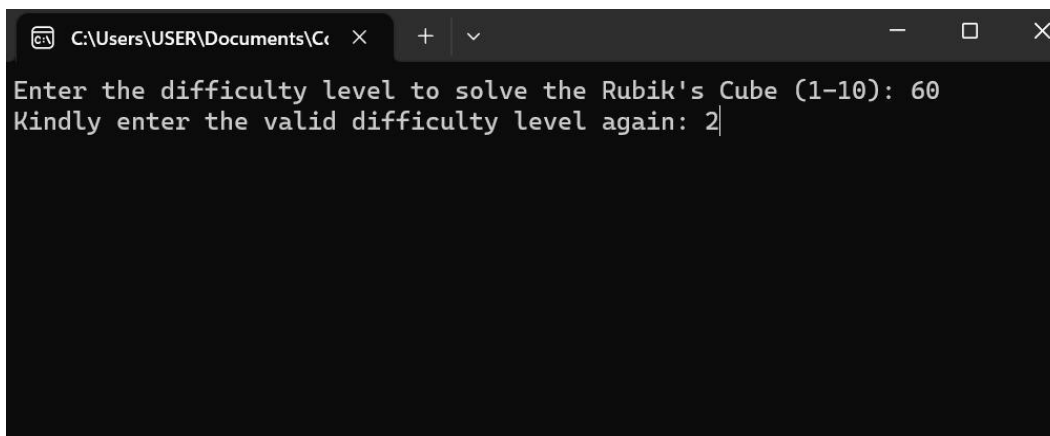
When executing the program , the program will ask the user at which difficulty level the user wants to solve the puzzle in.



```
C:\Users\USER\Documents\Cc > Enter the difficulty level to solve the Rubik's Cube (1-10): 2
```

- **Entering a valid difficulty level :**

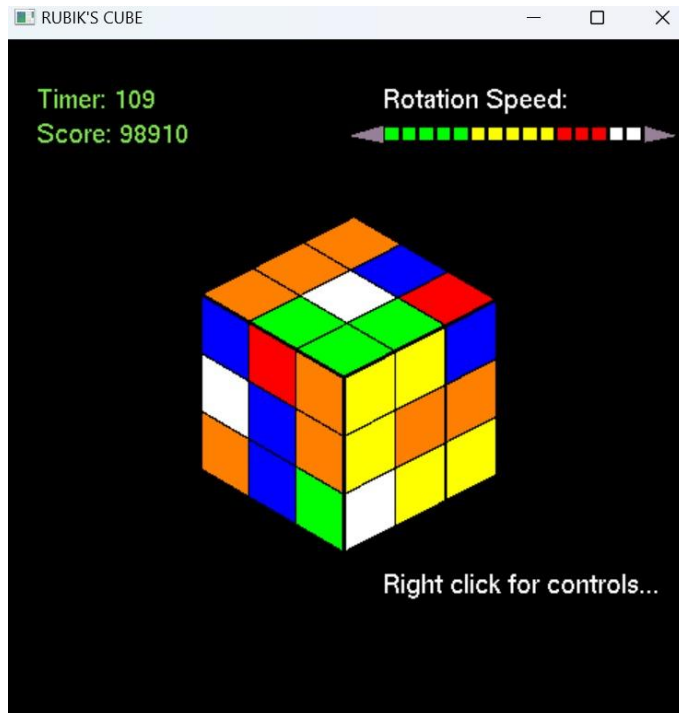
Users will be asked to enter a valid difficulty level if they have entered an input exceeding the difficulty range provided.



```
C:\Users\USER\Documents\Cc > Enter the difficulty level to solve the Rubik's Cube (1-10): 60
Kindly enter the valid difficulty level again: 2
```

- **Adjusting Rotational speed :**

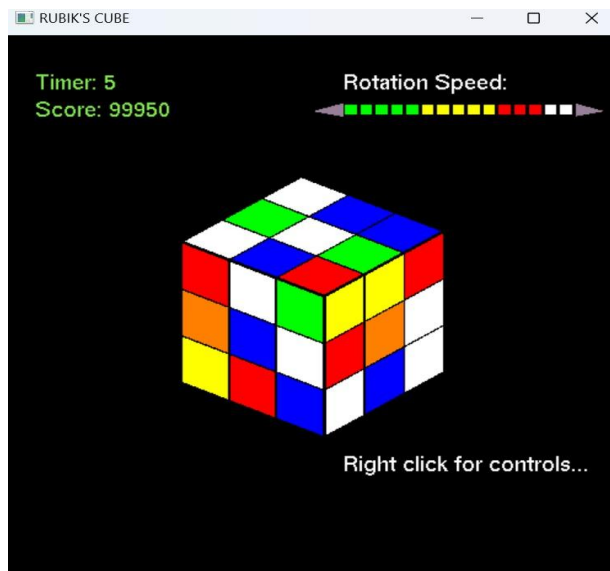
The users can adjust / change the rotational speed of the cube depending on how comfortable they find while solving the cube.





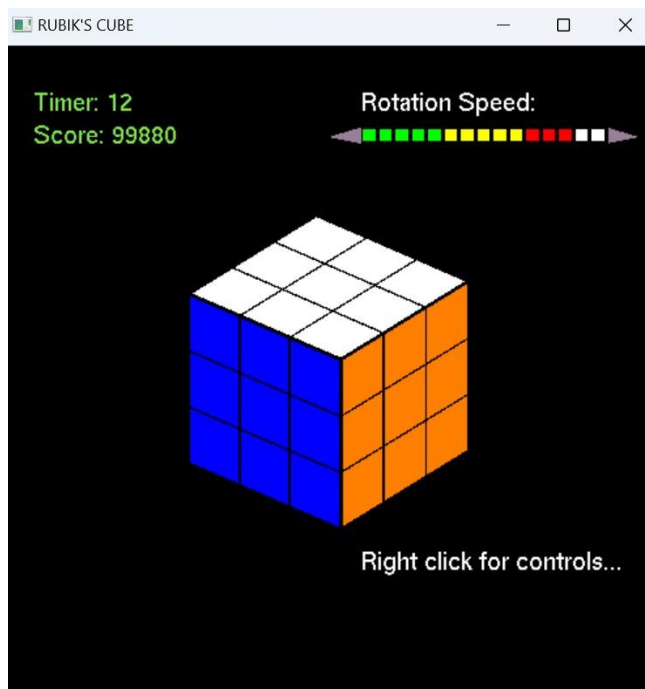
- **Shuffle Cube :**

Users can shuffle the cube in order to start solving .



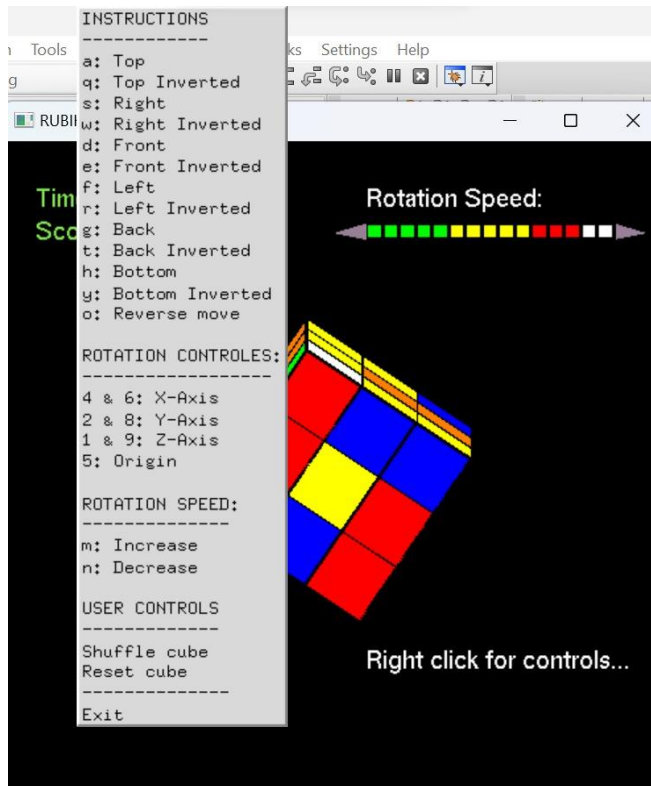
- **Reset Cube :**

Users will be able to reset the cube .



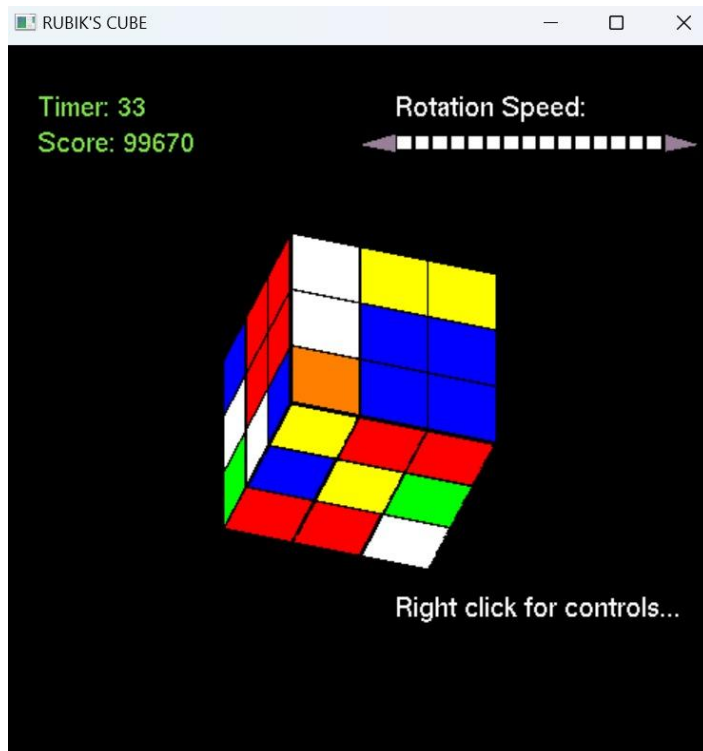
- **Control Instructions :**

Users are able to check a drop down panel where they can view the set of instructions to control , view , rotate the cube accordingly.



- **Timer :**

A timer function has been added which shows the user the time its been taking for them to solve the cube .



- **Score :**

The user can see their score while solving the cube , for every second that passes in the a timer 1 point from the score gets deducted till the user completes solving the cube.

