Introduction

This OpenGL program rotates a quadrilateral around a user-defined pivot point by applying a transformation formula.

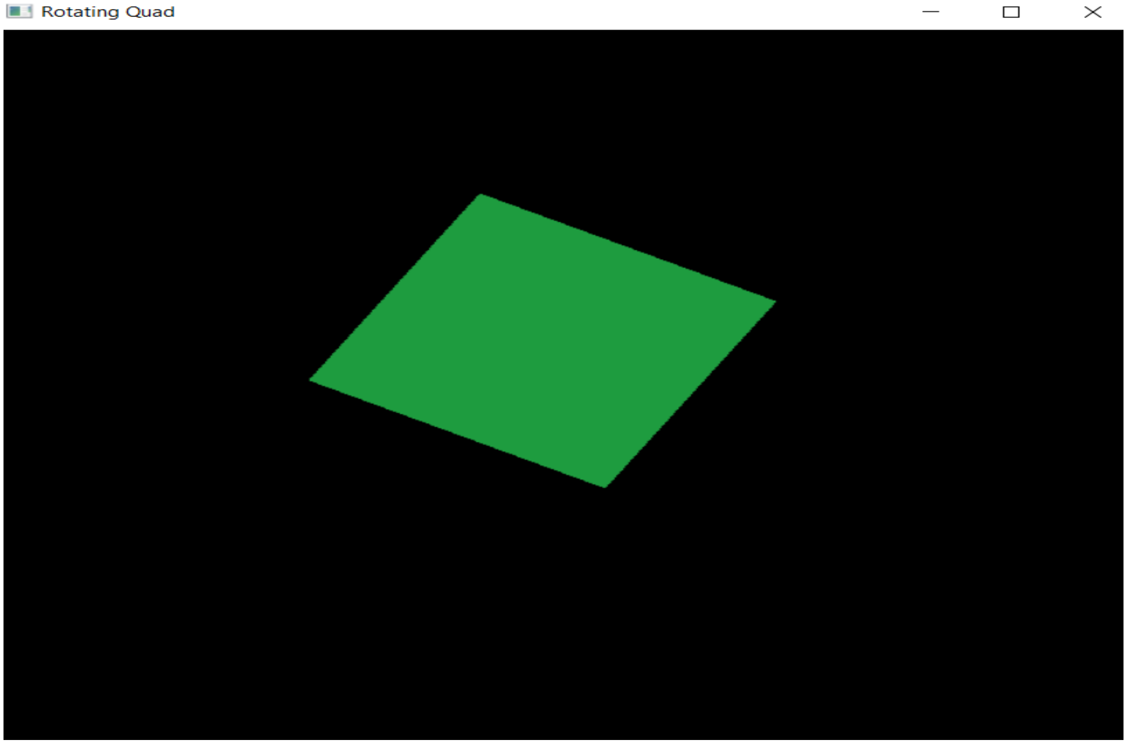
Code

```
#include <GL/glut.h>
#include <math.h>
#include <iostream>


void RotateQuad(float x1, float y1, float x2, float y2, float x3, float y3, float x4, float y4, float px, float py, float degree, int direction) {
    float radian = degree * (3.1416 / 180);
    if (direction == 0) radian = -radian;
    float points[4][2] = {{x1, y1}, {x2, y2}, {x3, y3}, {x4, y4}};
    float rotated[4][2];

    for (int i = 0; i < 4; i++) {
        float tx = points[i][0] - px;
        float ty = points[i][1] - py;

        rotated[i][0] = px + (tx * cos(radian) - ty * sin(radian));
        rotated[i][1] = py + (tx * sin(radian) + ty * cos(radian));
    }
    glBegin(GL_QUADS);
    glColor3ub(30, 156, 63);
    for (int i = 0; i < 4; i++) {
        glVertex2f(rotated[i][0], rotated[i][1]);
    }
    glEnd();
}

void display() {
    glClear(GL_COLOR_BUFFER_BIT);
    RotateQuad(150, 200, 300, 200, 300, 350, 150, 350, 250, 250, 30, 0);
    glFlush();
}

void myInit() {
    glClearColor(0, 0, 0, 1);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(0, 500, 0, 500, -1, 1);
}

int main(int argc, char** argv) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(500, 500);
    glutInitWindowPosition(100, 100);
    glutCreateWindow("Rotating Quad");
    myInit();
    glutDisplayFunc(display);
    glutMainLoop();
    return 0;
}
```

Important Functions & Operations

1. **Rotation Logic**
   - Each vertex (x, y) is translated relative to the pivot (px, py).
   - The rotated coordinates are computed using:
     $x' = px + (x - px)\cos(\theta) - (y - py)\sin(\theta)$
     $y' = py + (x - px)\sin(\theta) + (y - py)\cos(\theta)$
   - The rotated points are then used to redraw the quadrilateral.
2. **Pivot Logic**
   - The pivot (px, py) determines the center of rotation.
   - All vertices are transformed relative to this pivot, ensuring smooth rotation.
3. **RotateQuad() Function**
   - Takes quadrilateral coordinates, pivot, angle, and direction as parameters.
   - Converts degree to radians and adjusts sign based on direction.
   - Computes new coordinates using trigonometric formulas and renders the rotated quadrilateral using glBegin(GL_QUADS).
   - When direction is zero, quad will be rotated clockwise, else (1) anti clockwise.

4. **display() Function**
    - Clears the screen and calls RotateQuad() with specific quadrilateral coordinates, a user-defined pivot, and a rotation angle.
    - Uses glFlush() to ensure immediate rendering of the updated frame.
5. **Example**
    - Rotating the quad 30 degree clockwise when pivot point is (250,250).



Conclusion

This program demonstrates how to rotate a quadrilateral around a user-defined pivot point using OpenGL. By applying trigonometric transformations, the vertices are recalculated and rendered dynamically. This approach is fundamental in computer graphics for handling object transformations efficiently.