Experiment No: 6

Experiment Name:

Rotation of a Line, Triangle and Rectangle about a Pivot Point

---

Objective:

The objective of this experiment is to understand and implement rotation transformation in computer graphics by rotating a line, triangle, and rectangle about a given pivot point using OpenGL.

Theory:

Rotation is one of the basic geometric transformations in computer graphics. It changes the orientation of an object by rotating it through a specified angle around a fixed point called the pivot point.

In 2D rotation:

- The object is rotated by an angle θ
- The rotation can be clockwise or anticlockwise
- Rotation about a pivot point requires:
    1. Translating the object to the origin
    2. Applying rotation
    3. Translating it back to the pivot point

Requirements

• Programming Language: C++

• Graphics Library: OpenGL (GLUT)

• Software: CodeBlocks

Procedure / Description of Code:

1. Initialize OpenGL window using GLUT.
2. Set up an orthographic 2D viewing system.
3. Draw a **line, triangle, and rectangle** using OpenGL primitives.
4. Define a **pivot point** for rotation.
5. Translate the object to the pivot point.
6. Apply rotation using glRotatef().
7. Translate the object back to its original position.
8. Use a **timer function** to continuously update the rotation angle.
9. Display the rotated shapes on the screen.

SOURCE CODE:

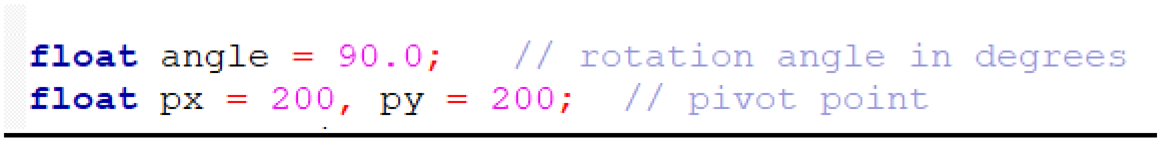
```
#include <windows.h>

#include <GL/glut.h>

#include <math.h>

float angle = 90.0;   // rotation angle in degrees

float px = 200, py = 200;  // pivot point

void rotatePoint(float &x, float &y)

{

   float rad = angle * 3.14159 / 180;

   float tx = x - px;

   float ty = y - py;

   float rx = tx * cos(rad) - ty * sin(rad);

   float ry = tx * sin(rad) + ty * cos(rad);

   x = rx + px;

   y = ry + py;

}
```

```
void drawLine()
{
   float x1 = 100, y1 = 200;
   float x2 = 300, y2 = 200;

   rotatePoint(x1, y1);
   rotatePoint(x2, y2);

   glBegin(GL_LINES);
   glVertex2f(x1, y1);
   glVertex2f(x2, y2);
   glEnd();
}
void drawTriangle()
{
   float x1 = 150, y1 = 250;
   float x2 = 200, y2 = 300;
   float x3 = 250, y3 = 250;

   rotatePoint(x1, y1);
   rotatePoint(x2, y2);
   rotatePoint(x3, y3);
```

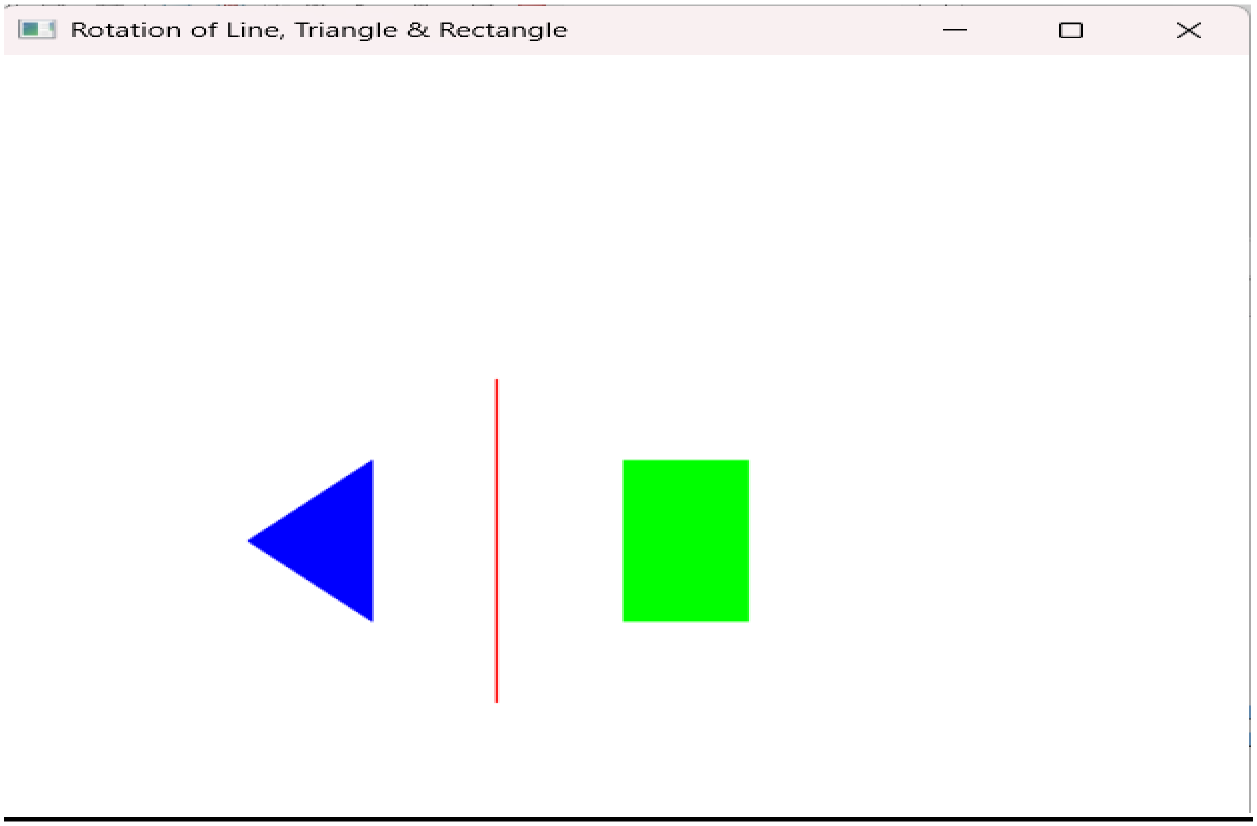
```
    glBegin(GL_TRIANGLES);
    glVertex2f(x1, y1);
    glVertex2f(x2, y2);
    glVertex2f(x3, y3);
    glEnd();
}
void drawRectangle()
{
    float x1 = 150, y1 = 150;
    float x2 = 250, y2 = 150;
    float x3 = 250, y3 = 100;
    float x4 = 150, y4 = 100;
    rotatePoint(x1, y1);
    rotatePoint(x2, y2);
    rotatePoint(x3, y3);
    rotatePoint(x4, y4);
    glBegin(GL_QUADS);
    glVertex2f(x1, y1);
    glVertex2f(x2, y2);
    glVertex2f(x3, y3);
    glVertex2f(x4, y4);
    glEnd();
}
```

```
void display()
{
   glClear(GL_COLOR_BUFFER_BIT);

   glColor3f(1, 0, 0); // Line
   drawLine();

   glColor3f(0, 0, 1); // Triangle
   drawTriangle();

   glColor3f(0, 1, 0); // Rectangle
   drawRectangle();

   glFlush();
}
void init()
{
   glClearColor(1, 1, 1, 1);
   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   gluOrtho2D(0, 500, 0, 500);
}
int main(int argc, char** argv)
```

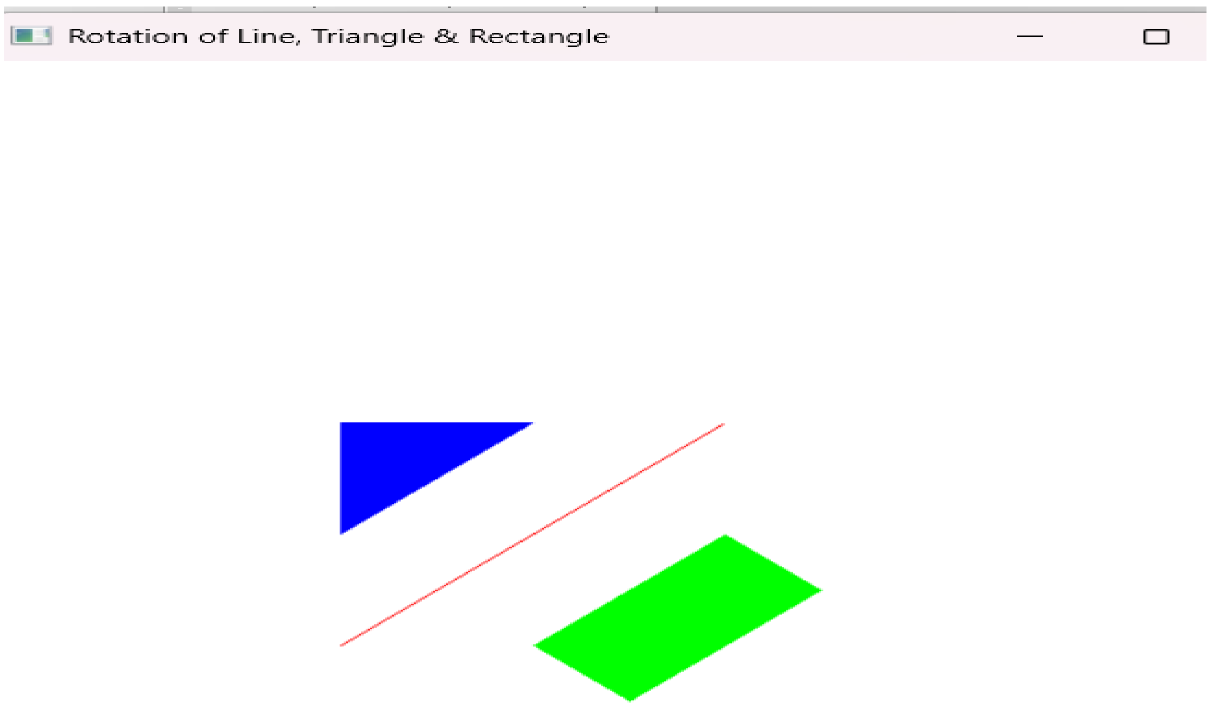
```
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(500, 500);
    glutCreateWindow("Rotation of Line, Triangle & Rectangle");
    init();
    glutDisplayFunc(display);
    glutMainLoop();
    return 0;
}
```



```
float angle = 90.0;    // rotation angle in degrees
float px = 200, py = 200;   // pivot point
```



Rotation about 45 degree:



Conclusion / Discussion:

This experiment demonstrates how **rotation transformation** works in computer graphics. By using translation and rotation functions in OpenGL, objects can be rotated about any pivot point. Continuous rotation makes the visualization more effective and helps in understanding real-time graphics transformations. This concept is widely used in animations, games, and simulations.