Problem Statement

Write a program to illustrate composite transformations (translation, rotation, scaling) as well as shearing, in graphics window.

Background Theory

In computer graphics, geometric transformations are essential for manipulating shapes and objects within a scene. These transformations modify an object's position, orientation, size, or shape. The most common transformations are translation, rotation, scaling, and shearing

1. TRANSLATION

Translation shifts an object from one location to another in the 2D plane. It does not alter shape, size, or orientation, just position. Its example matrix is:

$$\begin{bmatrix} 1 & 0 & tx \\ 0 & 1 & ty \\ 0 & 0 & 1 \end{bmatrix}$$

where, `tx` and `ty` are the distances to move in x and y directions.

2. ROTATION

Rotation turns an object around a fixed point, typically the origin. It preserves the shape and size but changes the direction it faces. It's useful in animations and object orientation control. Its example matrix (for counterclockwise rotation by θ):

$$\begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

3. SCALING

Scaling resizes an object, i.e. enlarges or shrinks it along x and/or y axes. Uniform scaling keeps the proportions the same; non-uniform changes the aspect ratio. It's example matrix is:

$$\begin{bmatrix} sx & 0 & 0 \\ 0 & sy & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

where `sx` and `sy` are the scale factors for width and height.

4. SHEARING

Shearing skews an object, i.e. it distorts the shape by shifting layers. It's commonly used to simulate 3D perspectives or italic effects in fonts.

X-Shear Matrix:

$$\begin{bmatrix} 1 & shx & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Y-Shear Matrix:

$$\begin{bmatrix} 1 & 0 & 0 \\ shy & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

where `shx` and `shy` determine how much to slant in respective directions.

Algorithm

1. Start
2. Input triangle coordinates (x1,y1), (x2,y2) and (x3,y3).
3. Calculate maximum and minimum height and width and draw axes respectively.
4. Draw the triangle based on input coordinates.
5. Translation
   - Input translation vector (tx, ty)
   - Draw triangle with updated coordinates as:
     - x1+tx, x2+tx, x3+tx, y1-ty, y2-ty, y3-ty for x1,x2,x3,y1,y2,y3
6. Rotation
   - Input the angle by which triangle is to be rotated.
   - Calculated new points as:

     xr1 = centerX + (x1-centerX) * cos(thetaRadian) - (y1-centerY) * sin(thetaRadian);

     xr2 = centerX + (x2-centerX) * cos(thetaRadian) - (y2-centerY) * sin(thetaRadian);

xr3 = centerX + (x3-centerX) * cos(thetaRadian) - (y3-centerY) * sin(thetaRadian);

yr1 = centerY + (x1-centerX) * sin(thetaRadian) + (y1-centerY) * cos(thetaRadian);

yr2 = centerY + (x2-centerX) * sin(thetaRadian) + (y2-centerY) * cos(thetaRadian);

yr3 = centerY + (x3-centerX) * sin(thetaRadian) + (y3-centerY) * cos(thetaRadian);

- Draw new triangle with updated coordinates

.

7. Scaling
   - Input scale factors Sx and Sy.
   - Calculate new coordinates as:

xs1=x1*sx+x1 * (1 - sx);

xs2=x2*sx+x1 * (1 - sx);

xs3=x3*sx+x1 * (1 - sx);

ys1=y1*sy+y1 * (1 - sy);

ys2=y2*sy+y1 * (1 - sy);

ys3=y3*sy+y1 * (1 - sy);

   - Draw triangle with updated coordinates.
8. Shearing
   - Input the values of shx and shy.
   - Calculate new coordinates as:

xsh1 = x1 + shx * (centerY - y1);

ysh1 = y1 + shy * (x1 - centerX);

xsh2 = x2 + shx * (centerY - y2);

ysh2 = y2 + shy * (x2 - centerX);

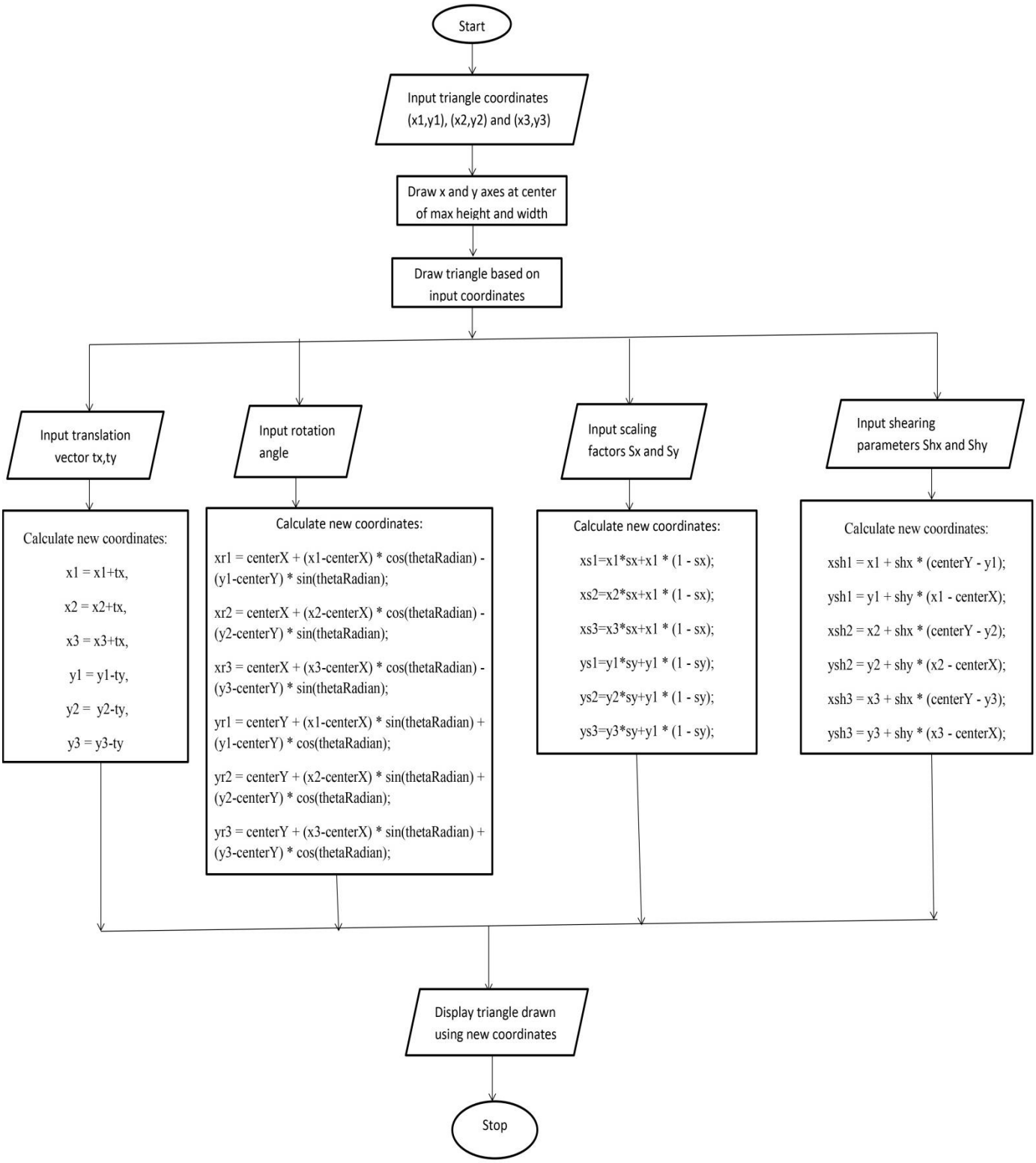
xsh3 = x3 + shx * (centerY - y3);

ysh3 = y3 + shy * (x3 - centerX);

- Draw triangle with updated coordinates.

9. Stop.

Flowchart



Source code

```
#include <iostream>
#include <cstdlib>
#include <ctime>
#include <graphics.h>
#include <cmath>
using namespace std;
int randomColor(){
	srand(time(0));
	return rand() % 6 + 0;
}

void drawTriangle(float x1, float x2, float x3, float y1, float y2, float y3){
	line(x1, y1, x2, y2);
	line(x1, y1, x3, y3);
	line(x2, y2, x3, y3);
}

void drawAxes(float* x1, float* x2, float* x3, float* y1, float* y2, float* y3){
	float maxHeight = getmaxy();
	float maxWidth = getmaxx();
	float centerX = maxWidth/2;
	float centerY = maxHeight/2;
	line(0, centerY, maxWidth, centerY);
	line(centerX,0, centerX, maxHeight);
```

```
	*x1+=centerX;
	*x2+=centerX;
	*x3+=centerX;
	*y1=centerY-*y1;
	*y2=centerY-*y2;
	*y3=centerY-*y3;
}
int main(){
	int gd = DETECT , gm;
	initgraph(&gd, &gm, "");

	int k = 3;
	float x1,x2,x3,y1,y2,y3;
	float tx, ty;
	float maxHeight = getmaxy();
	float maxWidth = getmaxx();
	float centerX = maxWidth/2;
	float centerY = maxHeight/2;
	setlinestyle(SOLID_LINE, 0, 3);

	cout<<"input the coordinates of a triangle: "<<endl;
	cout<<"(x1,y1) ";
	cin>>x1>>y1;
	cout<<"(x2,y2) ";
	cin>>x2>>y2;
```

```
cout<<"(x3,y3) ";
cin>>x3>>y3;

setbkcolor(WHITE);
cleardevice();
setcolor(BLACK);
drawAxes(&x1,&x2,&x3,&y1,&y2,&y3);
drawTriangle(x1,x2,x3,y1,y2,y3);

//for translation
while(k!=0){
        cout<<"enter translation vector: "<<endl;
        cout<<"tx ty --- ";
        cin>>tx>>ty;

        setcolor(RED);
        drawTriangle(x1+tx, x2+tx, x3+tx, y1-ty, y2-ty, y3-ty);
        k--;
}

//for scaling
cout<<"press any key then enter to start scaling: ";
char c;
cin>>c;
setbkcolor(LIGHTBLUE);
```

```
cleardevice();
setcolor(WHITE);
drawAxes(&x1,&x2,&x3,&y1,&y2,&y3);
drawTriangle(x1,x2,x3,y1,y2,y3);
cout<<"----------------------------"<<endl;
k=3;

while(k!=0){
        cout<<"enter the scale:"<<endl;
        cout<<"Sx Sy --- ";
        float sx, sy;
        cin>>sx>>sy;
        float xs1, xs2, xs3, ys1, ys2, ys3;
        xs1=x1*sx+x1 * (1 - sx);
        xs2=x2*sx+x1 * (1 - sx);
        xs3=x3*sx+x1 * (1 - sx);
        ys1=y1*sy+y1 * (1 - sy);
        ys2=y2*sy+y1 * (1 - sy);
        ys3=y3*sy+y1 * (1 - sy);
        setcolor(randomColor());
        setlinestyle(SOLID_LINE, 0, 3);
        drawTriangle(xs1, xs2, xs3, ys1, ys2, ys3);
        k--;
}
```

```
//      //for rotating
        k=3;
        cout<<"press any key then enter to start rotating: ";
        cin>>c;
        setbkcolor(BROWN);
        cleardevice();
        setcolor(WHITE);
        drawAxes(&x1,&x2,&x3,&y1,&y2,&y3);
        drawTriangle(x1,x2,x3,y1,y2,y3);

        float thetaDegrees;
        while(k!=0){
                cout<<"----------------------------"<<endl;
        cout<<"enter the angle to rotate in degrees: "<<endl;
        cout<<"angle -- ";
        cin>>thetaDegrees;
        float thetaRadian = thetaDegrees * (M_PI / 180.0);
        float xr1, xr2, xr3, yr1, yr2, yr3;
        xr1 = centerX + (x1-centerX) * cos(thetaRadian) - (y1-centerY) * sin(thetaRadian);
        xr2 = centerX + (x2-centerX) * cos(thetaRadian) - (y2-centerY) * sin(thetaRadian);
        xr3 = centerX + (x3-centerX) * cos(thetaRadian) - (y3-centerY) * sin(thetaRadian);
        yr1 = centerY + (x1-centerX) * sin(thetaRadian) + (y1-centerY) * cos(thetaRadian);
        yr2 = centerY + (x2-centerX) * sin(thetaRadian) + (y2-centerY) * cos(thetaRadian);
        yr3 = centerY + (x3-centerX) * sin(thetaRadian) + (y3-centerY) * cos(thetaRadian);
        setcolor(randomColor());
```

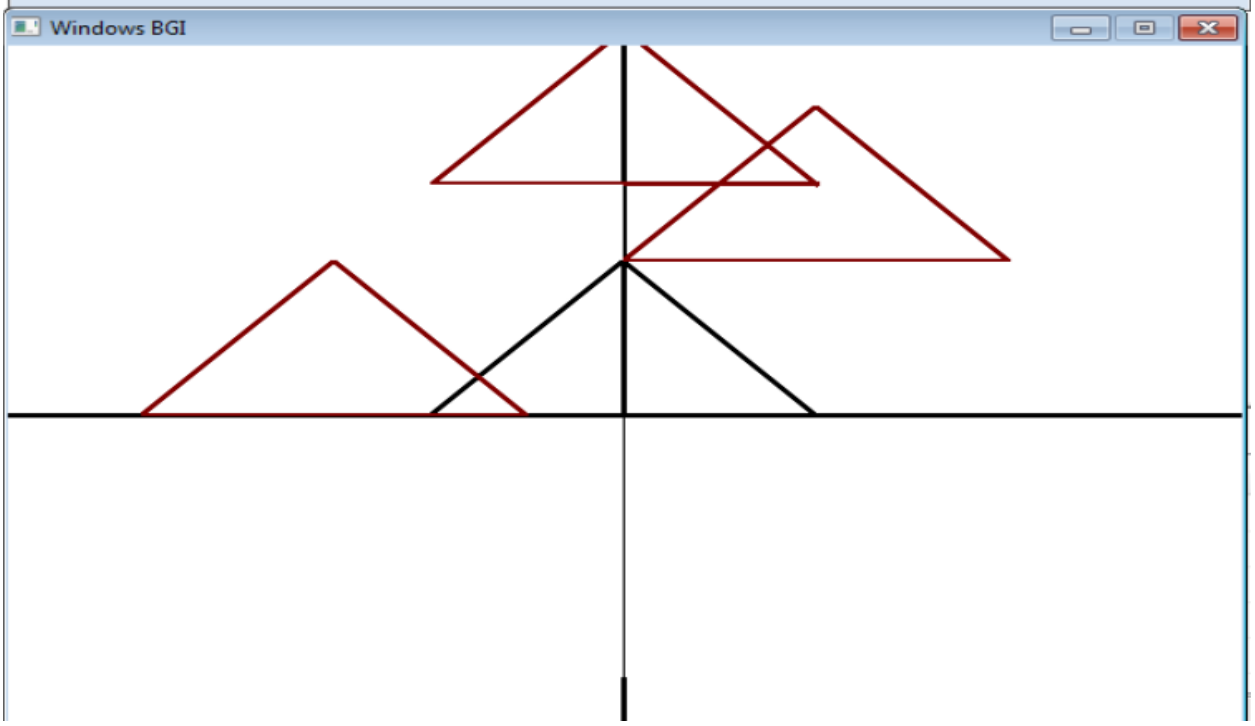
```
drawTriangle(xr1, xr2, xr3, yr1, yr2, yr3);
k--;
}
//for shearing
k=3;
cout<<"press any key then enter to start shearing: ";
cin>>c;
setbkcolor(BROWN);
cleardevice();
setcolor(WHITE);

drawAxes(&x1,&x2,&x3,&y1,&y2,&y3);
drawTriangle(x1,x2,x3,y1,y2,y3);
float shx, shy;
while(k!=0){
        cout<<"enter value of shx and shy: "<<endl;
        cout<<"shx shy -- ";
        cin>>shx>>shy;
        float xsh1, xsh2, xsh3, ysh1, ysh2, ysh3;
        xsh1 = x1 + shx*(centerY-y1);
        ysh1 = y1 + shy*(x1-centerX);
        xsh2 = x2 + shx*(centerY-y2);
        ysh2 = y2 + shy*(x2-centerX);
        xsh3 = x3 + shx*(centerY-y3);
        ysh3 = y3 + shy*(x3-centerX);
```

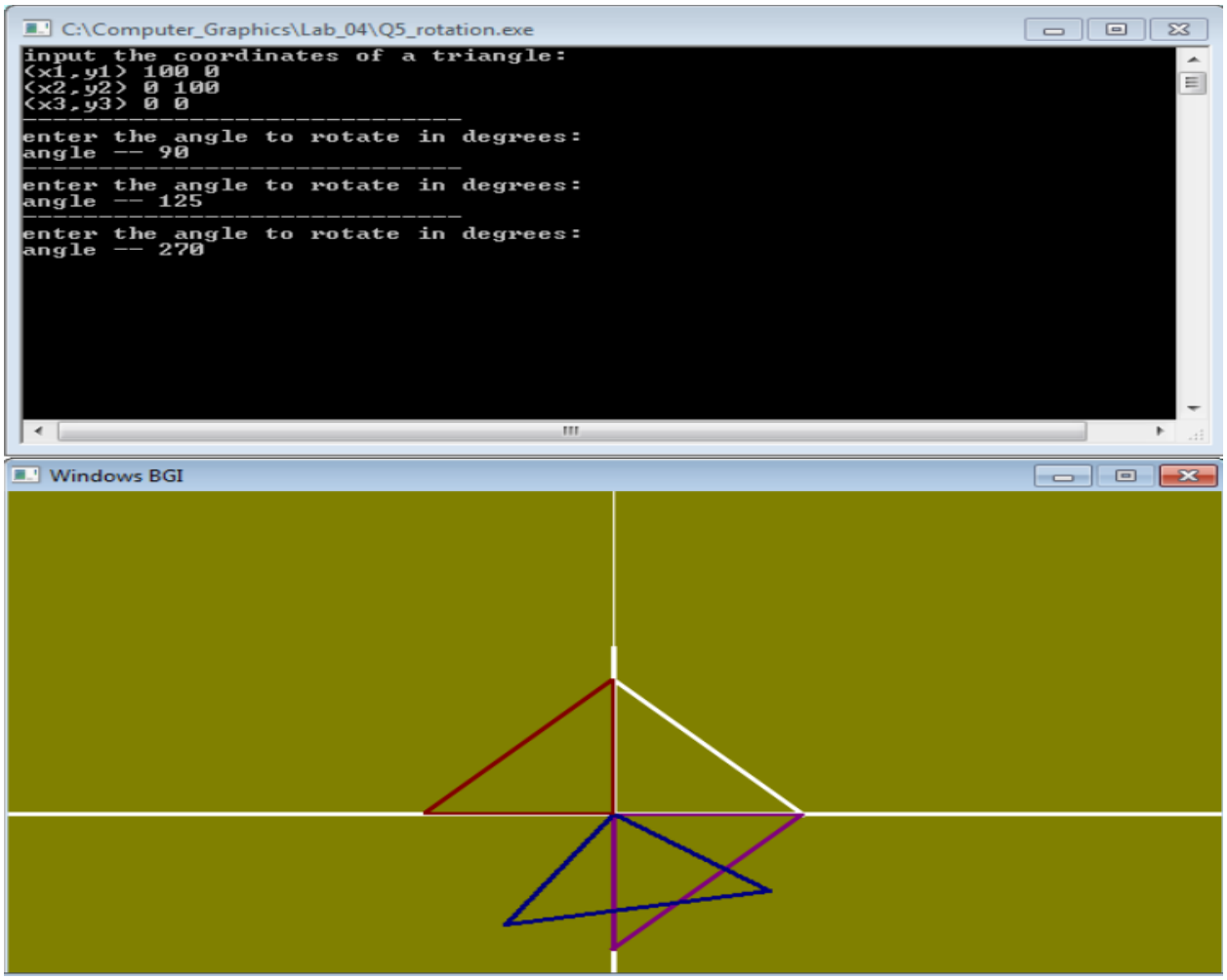
```
            setcolor(randomColor());
            drawTriangle(xsh1, xsh2, xsh3, ysh1, ysh2, ysh3);
            k--;
        }
        getch();
        closegraph();
}
```

Sample Input/Output

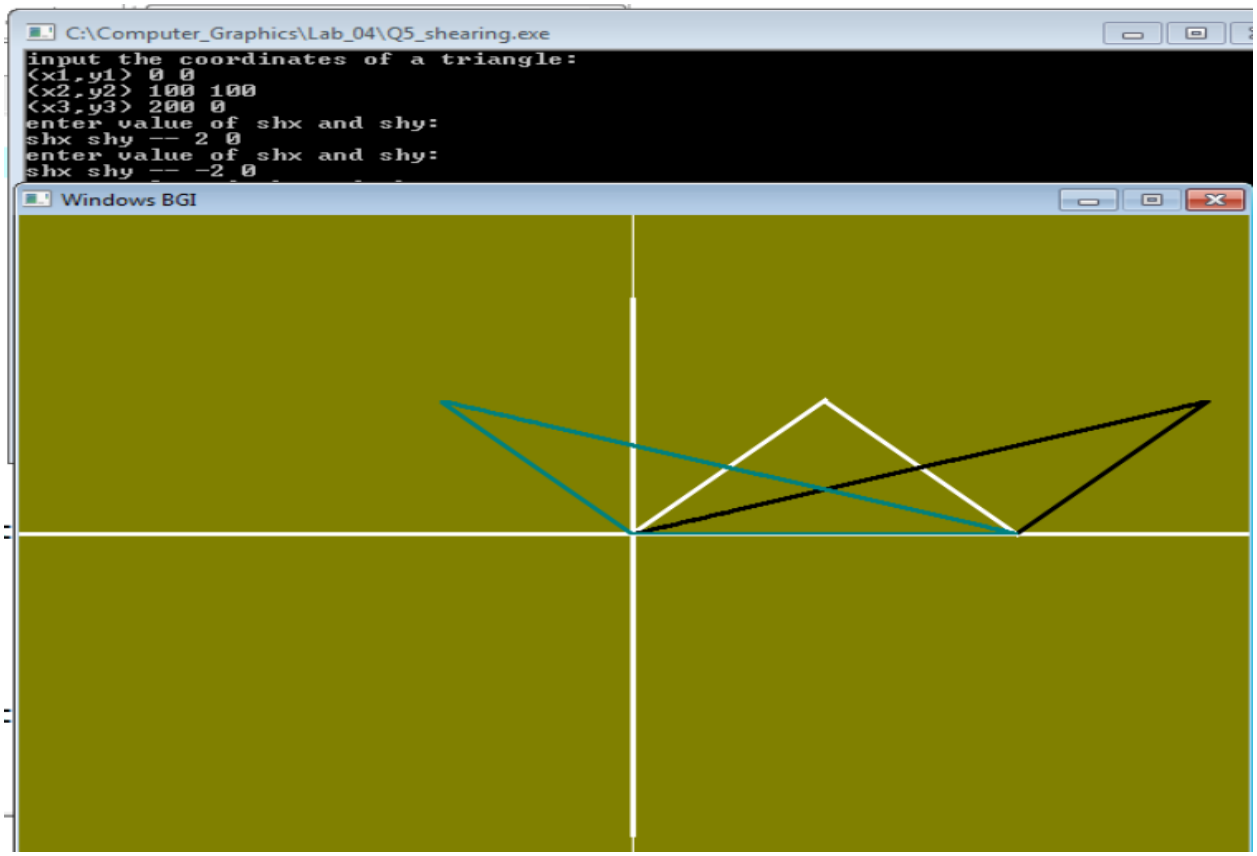
1. Translation



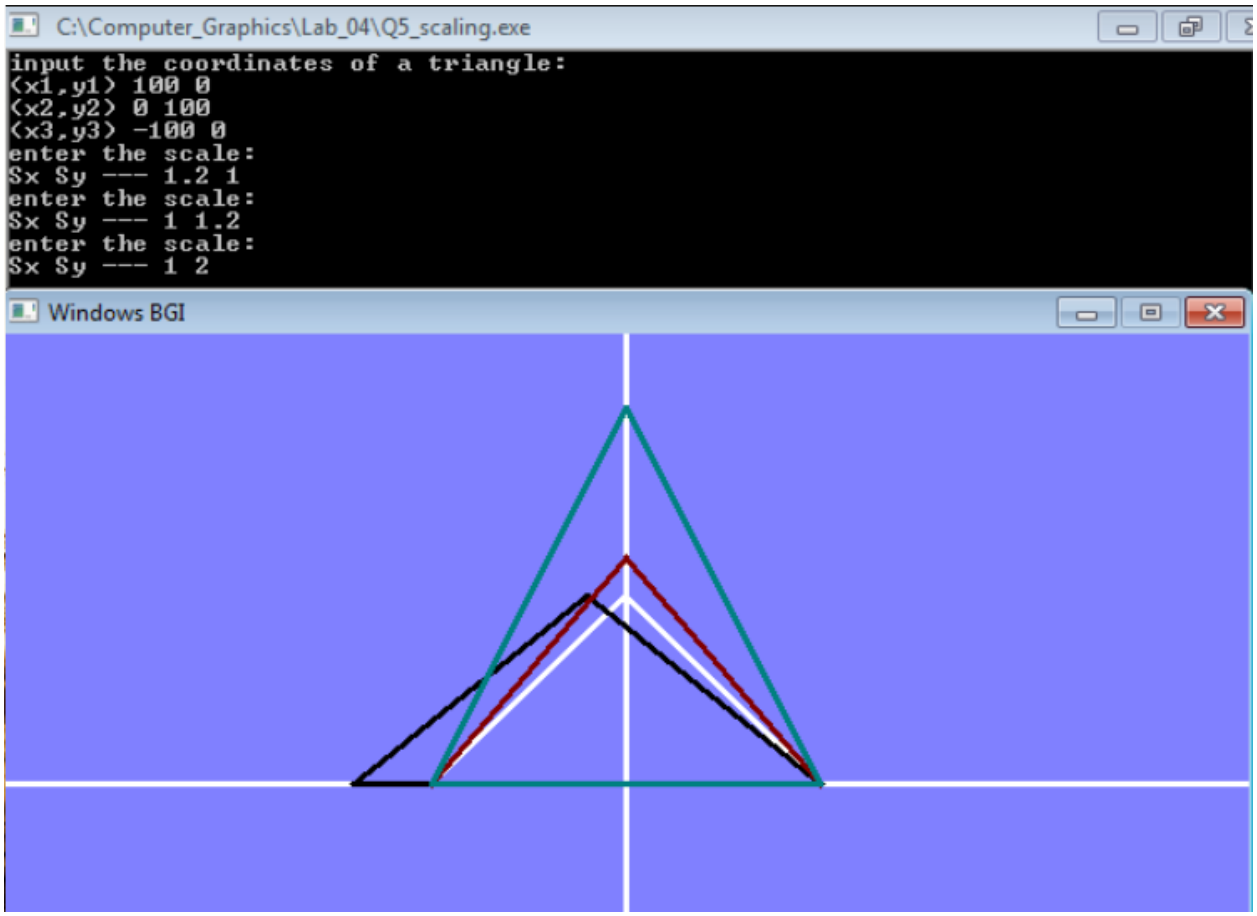
2. Rotation



3. Shearing



4. Scaling



Conclusion

In this lab, we successfully implemented and visualized the fundamental 2D geometric transformations: translation, scaling, rotation, and shearing using C++ and the graphics.h library. By applying these operations on a triangle, we were able to observe the individual effects of each transformation on the shape's position, size, orientation, and geometry. Through translation, the triangle was repositioned to new coordinates by shifting it along the x and y axes. Scaling allowed us to resize the triangle while maintaining its relative proportions, and we observed how scaling with respect to a fixed point preserved the triangle's shape. Rotation was performed around the origin (screen center), demonstrating how the triangle's vertices change based on angular displacement. Lastly, shearing showed how the triangle's shape can be skewed along the x or y direction, distorting the angles while keeping the area characteristics variable.