Department of Computer Science and Engineering

Shivanirudh S G, 185001146, Semester VII

27 August 2021

UCS1712 - Graphics and Multimedia Lab

Exercise 6: 2D Composite Transformations and Windowing in C++ using OpenGL

Aim:

To compute the composite transformation matrix for any 2 transformations given as input by the user and applying it on the object.

Code:

```
#ifndef LOPENGL_H
#define LOPENGL_H

#include <GL/freeglut.h>
#include <GL/gl.h>
#include <GL/glu.h>
#include <math.h>
#include <stdio.h>
#include<iostream>
#include<vector>
#include<ctime>
using namespace std;

#endif
```

```
#ifndef LUTIL_H
#define LUTIL_H

#include "Headers.h"

//Screen Constants
const int SCREEN_WIDTH = 640;
const int SCREEN_HEIGHT = 480;
const int SCREEN_FPS = 60;
const int POINT_SIZE=3;
vector<int> X_points;
vector<int> Y_points;

double transform1[3][3];
double transform2[3][3];

vector<pair<double, double>> transforms;

int edge_count;
int transformation;
double factor_x, factor_y;
double angle_radians;

bool initGL();

void update();

void render();

void y_axis();

void x_axis();

void setEdgeCount(int option);

vector<vector<double>> matrix_multiplication(double points[][3],
    int pr);

void drawPolygon();

void applyTransforms();

void drawTransfomedPolygon();

void setTranslateMatrix(double x, double y, int transform_number);

void setRotateMatrix(double angle_radians, double x, double y, int
    transform_number);

void setScaleMatrix(double x, double y, double xf, double yf, int
    transform_number);

void setReflectMatrix(double angle_radians, double intercept, int
    transform_number);

void setShearMatrix(double shx, double shy, int transform_number);

```

```
#endif
```

```
#include "Signatures.h"

bool initGL(){

    //Initialize Projection Matrix
    glMatrixMode( GL_PROJECTION );
    glLoadIdentity();
    gluOrtho2D(-640.0,640.0,-480.0,480.0);

    //Initialize clear color
    glClearColor( 0.f, 0.f, 0.f, 1.f );
    glColor3f(0.0f, 0.0f, 0.0f);

    glPointSize(POINT_SIZE);
    glEnable(GL_POINT_SMOOTH);

    //Check for error
    GLenum error = glGetError();
    if( error != GL_NO_ERROR )
    {
        printf( "Error initializing OpenGL! %s\n", gluErrorString( error ) );
        return false;
    }

    return true;
}

void update(){

}

void render(){
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(1.0, 1.0, 1.0);
    drawPolygon();
    glFlush();

    while(true){
        glClear(GL_COLOR_BUFFER_BIT);
        glColor3f(1.0, 1.0, 1.0);

        cout<<"Choose first transformation: "<<endl;
        cout<<"1 for Translation"<<endl<<"2 for Rotation"<<endl;
        cout<<"3 for Scaling"<<endl<<"4 for Reflection"<<endl;
        cout<<"5 for shearing"<<endl<<"0 to Exit"<<endl;
        cout<<"Enter your choice: ";cin>>transformation;

        if(!transformation){
            return;
        }

        if(transformation == 1){
            cout<<"Enter the translation factor for X and Y: ";
            cin>>factor_x >> factor_y;
            drawPolygon();
            setTranslateMatrix(factor_x,factor_y, 1);
        }
        else if(transformation == 2){
            double angle;
            cout<<"Enter the rotation angle: ";
            cin>>angle;
            double x,y;
            cout<<"Enter point about which to be rotated: ";
            cin>>x>>y;
            angle_radians = angle * 3.1416 / 180;
            drawPolygon();
            setRotateMatrix(angle_radians, x, y, 1);
        }
        else if(transformation == 3){
            cout<<"Enter the scaling factor for X and Y: ";
            cin>>factor_x >> factor_y;
            double x,y;
            cout<<"Enter point about which to be scaled: ";
            cin>>x>>y;
            drawPolygon();
            setScaleMatrix(factor_x,factor_y, x, y, 1);
        }
        else if(transformation == 4){
            double angle;
            cout<<"Enter the angle with X-axis and Y-intercept of the mirror: ";
            cin>>angle >> factor_y;
            angle_radians = angle * 3.1416/180;
            drawPolygon();
            setReflectMatrix(angle_radians,factor_y, 1);
        }
        else if(transformation == 5){
            cout<<"Enter the shearing factor for X and Y: ";
            cin>>factor_x >> factor_y;
            drawPolygon();
            setShearMatrix(factor_x, factor_y, 1);
        }
        else if(transformation){
            cout<<"Invalid option"<<endl;
        }
        else;

        cout<<"Choose second transformation: "<<endl;
        cout<<"1 for Translation"<<endl<<"2 for Rotation"<<endl;
        cout<<"3 for Scaling"<<endl<<"4 for Reflection"<<endl;
        cout<<"5 for shearing"<<endl<<"0 to Exit"<<endl;
        cout<<"Enter your choice: ";cin>>transformation;

        if(!transformation){
            return;
        }

        if(transformation == 1){
            cout<<"Enter the translation factor for X and Y: ";
            cin>>factor_x >> factor_y;
            drawPolygon();
            setTranslateMatrix(factor_x,factor_y, 2);
        }
        else if(transformation == 2){
            double angle;
            cout<<"Enter the rotation angle: ";
            cin>>angle;
            double x,y;
            cout<<"Enter point about which to be rotated: ";
            cin>>x>>y;
            angle_radians = angle * 3.1416 / 180;
            drawPolygon();
            setRotateMatrix(angle_radians, x, y, 2);
        }
        else if(transformation == 3){
            cout<<"Enter the scaling factor for X and Y: ";
            cin>>factor_x >> factor_y;
            double x,y;
            cout<<"Enter point about which to be scaled: ";
            cin>>x>>y;
            drawPolygon();
            setScaleMatrix(factor_x,factor_y, x, y, 2);
        }
        else if(transformation == 4){
            double angle;
            cout<<"Enter the angle with X-axis and Y-intercept of the mirror: ";
            cin>>angle >> factor_y;
            angle_radians = angle * 3.1416/180;
            drawPolygon();
            setReflectMatrix(angle_radians,factor_y, 2);
        }
        else if(transformation == 5){
            cout<<"Enter the shearing factor for X and Y: ";
            cin>>factor_x >> factor_y;
            drawPolygon();
            setShearMatrix(factor_x, factor_y, 2);
        }
        else if(transformation){
            cout<<"Invalid option"<<endl;
        }
        else;

        applyTransforms();
        drawTransfomedPolygon();
    }

    glFlush();
}

void y_axis(){
    glColor3f(1.0,1.0,1.0);
    glBegin(GL_LINES);
        glVertex2d(0, -480.0);
        glVertex2d(0, 480.0);
    glEnd();
    // glFlush();
}

```

```
void x_axis(){
    glColor3f(1.0,1.0,1.0);
    glBegin(GL_LINES);
        glVertex2d(-640.0, 0);
        glVertex2d(640.0, 0);
    glEnd();
    // glFlush();
}

void setEdgeCount(int option){
    if(option == 0){
        cout<<"Invalid"<<endl;
    }
    else if(option == 1 || option == 2){
        edge_count = 2;
    }
    else{
        edge_count = option;
    }
}

vector<vector<double>> matrix_multiplication(double points[][3], int pr){

    vector<vector<double>> final_transform(3);
    vector<vector<double>> ans(3);

    for(int i=0;i<pr;i++){
        for(int j=0;j<3;j++){
            ans[i].push_back(0);
        }
    }
    for(int i=0;i<3;i++){
        for(int j=0;j<3;j++){
            final_transform[i].push_back(0);
        }
    }

    for(int i=0;i<3;i++){
        for(int j=0;j<3;j++){
            for(int k=0;k<3;k++){
                final_transform[i][j] += (transform1[i][k] * transform2[k][j]);
            }
        }
    }
    for(int i=0;i<pr;i++){
        for(int j=0;j<3;j++){
            for(int k=0;k<3;k++){
                ans[i][j] += (points[i][k] * final_transform[k][j]);
            }
        }
    }
    return ans;
}

```

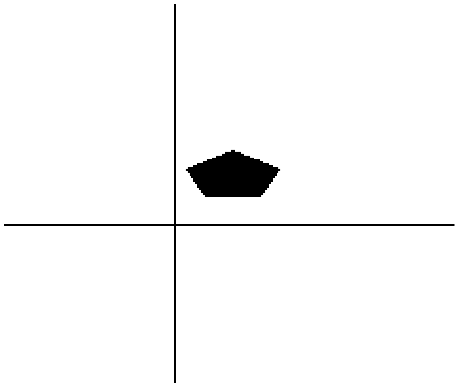
```
double round(double d){
    return floor(d + 0.5);
}

void drawPolygon(){
    y_axis();
    x_axis();
    glColor3f(1.0,1.0,1.0);
    if(edge_count==2)
        glBegin(GL_LINES);
    else
        glBegin(GL_POLYGON);

    for (int i = 0; i < edge_count; i++){
        glVertex2d(X_points[i], Y_points[i]);
    }
    glEnd();
    glFlush();
}

void applyTransforms(){
    for(int i=0;i<edge_count;i++){
        double points[1][3] = {X_points[i], Y_points[i], 1};
        vector<vector<double>> ans = matrix_multiplication(points, 1);
        transforms.push_back(pair<double, double>(ans[0][0],ans[0][1]));
    }
}

void drawTransfomedPolygon(){

    glFlush();
    glColor3f(0.0, 1.0, 0.0);
    if (edge_count == 2)
        glBegin(GL_LINES);
    else
        glBegin(GL_POLYGON);

    for(pair<double, double> p: transforms){
        glVertex2d(round(p.first), round(p.second));
    }
    glEnd();
    glFlush();
    transforms.clear();

}
void setTranslateMatrix(double x, double y, int transform_number){
    if(transform_number == 1){
        for(int i=0;i<3;i++){
            for(int j=0;j<3;j++){
                if(i == j){
                    transform1[i][j] = 1;
                }
                else{
                    transform1[i][j] = 0;
                }
            }
        }
        transform1[0][2] = x;
        transform1[1][2] = y;
    }
    else{
        for(int i=0;i<3;i++){
            for(int j=0;j<3;j++){
                if(i == j){
                    transform2[i][j] = 1;
                }
                else{
                    transform2[i][j] = 0;
                }
            }
        }
        transform2[0][2] = x;
        transform2[1][2] = y;
    }
}

void setRotateMatrix(double angle_radians, double x, double y, int transform_number){

    double adjust_matrix[3][3] = {{1, 0, -x},
                                  {0, 1, -y},
                                  {0, 0, 1}};
    double reset_matrix[3][3] = {{1, 0, x},
                                {0, 1, y},
                                {0, 0, 1}};
    double dummy_matrix[3][3] = {{cos(angle_radians), sin(angle_radians), 0},
                                {-sin(angle_radians), cos(angle_radians), 0},
                                {0, 0, 1}};

    double final_transform[3][3];
    if(angle_radians<0){
        dummy_matrix[0][1] *= -1;
        dummy_matrix[1][0] *= -1;
    }

    for(int i=0;i<3;i++){
        for(int j=0;j<3;j++){
            final_transform[i][j] = 0;
        }
    }

    for(int i=0;i<3;i++){
        for(int j=0;j<3;j++){
            for(int k=0;k<3;k++){
                final_transform[i][j] += (reset_matrix[i][k] * dummy_matrix[k][j]);
            }
        }
    }


    if(transform_number == 1){
        for(int i=0;i<3;i++){
            for(int j=0;j<3;j++){
                transform1[i][j] = 0;
            }
        }
        for(int i=0;i<3;i++){
            for(int j=0;j<3;j++){
                for(int k=0;k<3;k++){
                    transform1[i][j] += (final_transform[i][k] * adjust_matrix[k][j]);
                }
            }
        }
    }
    else{
        for(int i=0;i<3;i++){
            for(int j=0;j<3;j++){
                transform2[i][j] = 0;
            }
        }
        for(int i=0;i<3;i++){
            for(int j=0;j<3;j++){
                for(int k=0;k<3;k++){
                    transform2[i][j] += (final_transform[i][k] * adjust_matrix[k][j]);
                }
            }
        }
    }
}

void setScaleMatrix(double x, double y, double xf, double yf, int transform_number){
    if(transform_number == 1){
        for(int i=0;i<3;i++){
            for(int j=0;j<3;j++){
                if(i == j){
                    transform1[i][j] = 1;
                }
                else{
                    transform1[i][j] = 0;
                }
            }
        }
        transform1[0][0] = x;
        transform1[1][1] = y;
    }
    else{
        for(int i=0;i<3;i++){
            for(int j=0;j<3;j++){
                if(i == j){
                    transform2[i][j] = 1;
                }
                else{
                    transform2[i][j] = 0;
                }
            }
        }
        transform2[0][0] = x;
        transform2[1][1] = y;
    }
}

void setReflectMatrix(double angle_radians, double intercept, int transform_number){
    if(transform_number == 1){
        transform1[0][0] = cos(2*angle_radians);
        transform1[0][1] = sin(2*angle_radians);
        transform1[0][2] = intercept*sin(2*angle_radians);

        transform1[1][0] = sin(2*angle_radians);
        transform1[1][1] = -cos(2*angle_radians);
        transform1[1][2] = intercept*cos(2*angle_radians)+1;

        transform1[2][0] = 0;
        transform1[2][1] = 0;
        transform1[2][2] = 1;
    }
    else{
        transform2[0][0] = cos(2*angle_radians);
        transform2[0][1] = sin(2*angle_radians);
        transform2[0][2] = intercept*sin(2*angle_radians);

        transform2[1][0] = sin(2*angle_radians);
        transform2[1][1] = -cos(2*angle_radians);
        transform2[1][2] = intercept*cos(2*angle_radians)+1;

        transform2[2][0] = 0;
        transform2[2][1] = 0;
        transform2[2][2] = 1;
    }
}

void setShearMatrix(double shx, double shy, int transform_number){
    if(transform_number == 1){
        for(int i=0;i<3;i++){
            for(int j=0;j<3;j++){
                if(i == j){
                    transform1[i][j] = 1;
                }
                else{
                    transform1[i][j] = 0;
                }
            }
        }
        transform1[0][1] = shy;
        transform1[1][0] = shx;
    }
    else{
        for(int i=0;i<3;i++){
            for(int j=0;j<3;j++){
                if(i == j){
                    transform2[i][j] = 1;
                }
                else{
                    transform2[i][j] = 0;
                }
            }
        }
        transform2[0][1] = shy;
        transform2[1][0] = shx;
    }
}
```

```
#include "Helpers.h"

void runMainLoop(int val);

int main( int argc, char* args[] ){

    glutInit( &argc, args );

    glutInitContextVersion( 2, 1 );

    glutInitDisplayMode( GLUT_SINGLE|GLUT_RGB );
    glutInitWindowSize( SCREEN_WIDTH, SCREEN_HEIGHT );
    glutCreateWindow( "OpenGL" );

    int option=0;
    cout<<"Choose number of edges: (1 for line, 3 and upwards for polygon): ";
    cin>>option;

    setEdgeCount(option);
    cout<<"Enter vertices: "<<endl;
    for(int i=0;i<edge_count;i++){
        cout<<"Vertex "<<i+1<<" (x,y): ";
        int x,y;
        cin>>x>>y;
        X_points.push_back(x);
        Y_points.push_back(y);

    }

    drawPolygon();
    cout<<"Number of edges: "<<edge_count<<endl;

    if( !initGL() )
    {
        printf( "Unable to initialize graphics library!\n" );
        return 1;
    }


    glutDisplayFunc( render );

    glutTimerFunc( 1000 / SCREEN_FPS, runMainLoop, 0 );

    glutMainLoop();

    return 0;
}

void runMainLoop( int val ){
    update();
    render();

    glutTimerFunc( 1000 / SCREEN_FPS, runMainLoop, val );
}
```

Output:

Original Image:

```
Choose number of edges: (1 for line, 3 and upwards for polygon): 5
Enter vertices:
Vertex 1 (x,y): 30 30
Vertex 2 (x,y): 10 60
Vertex 3 (x,y): 60 80
Vertex 4 (x,y): 110 60
Vertex 5 (x,y): 90 30
Number of edges: 5
```

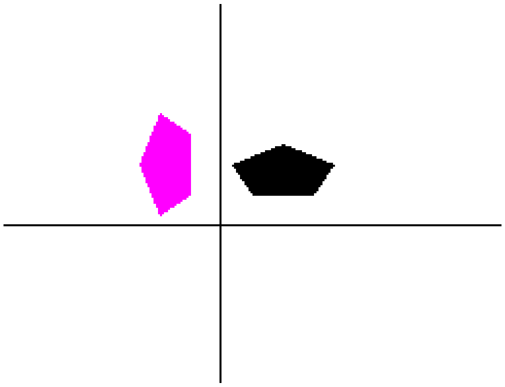


First transform: Translation

```
Choose first transformation:
1 for Translation
2 for Rotation
3 for Scaling
4 for Reflection
5 for shearing
0 to Exit
Enter your choice: 1

Enter the translation factor for X and Y: 40 40
```

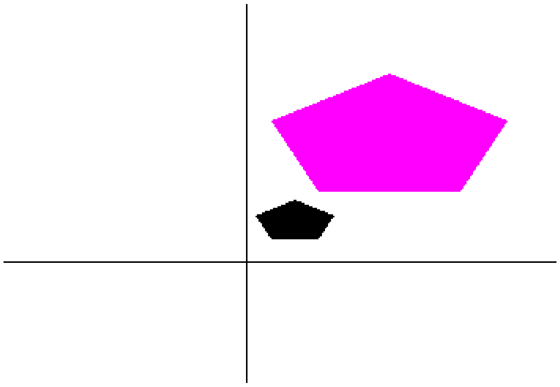
Second Transform: Rotation

```
Enter the rotation angle: 90
Enter point about which to be rotated: 0 0
```



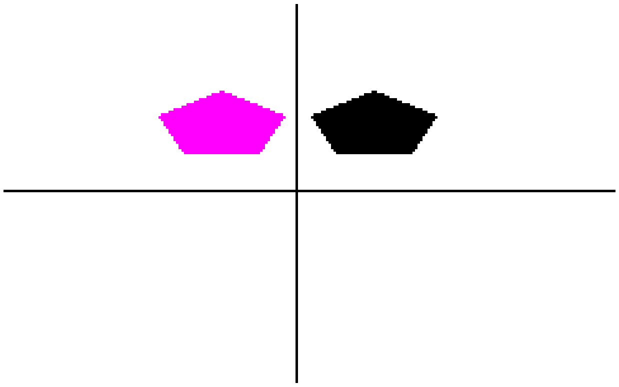
Second Transform: Scaling

```
Enter the scaling factor for X and Y: 3 3
Enter point about which to be scaled: 0 0
```



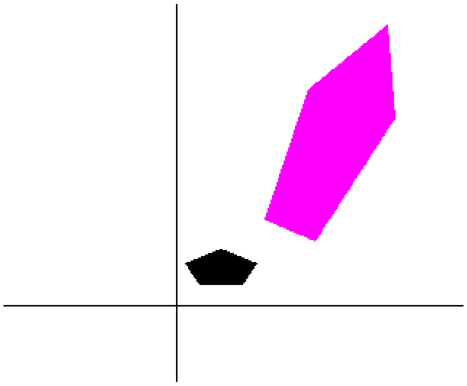
Second Transform: Reflection

```
Enter the angle with X-axis and Y-intercept of the mirror: 90 0
```



Second Transform: Shearing

```
Enter the shearing factor for X and Y: 3 3
```

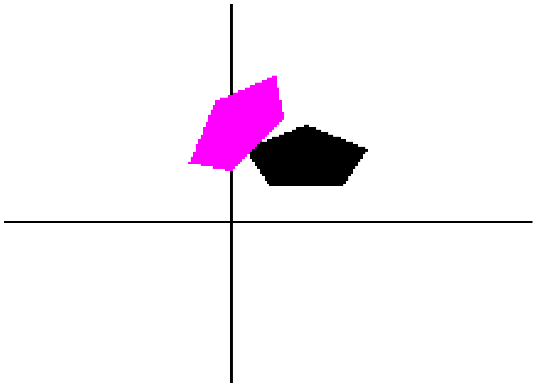


First transform: Rotation

```
Choose first transformation:
1 for Translation
2 for Rotation
3 for Scaling
4 for Reflection
5 for shearing
0 to Exit
Enter your choice: 2
Enter the rotation angle: 45
Enter point about which to be rotated: 0 0
```

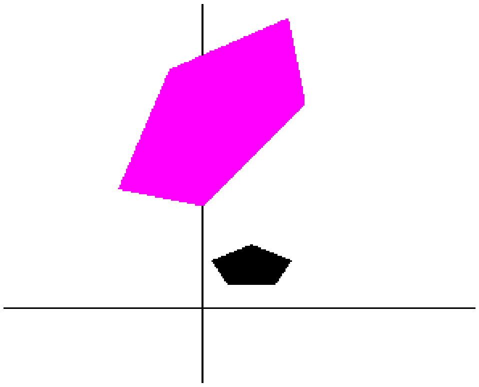
Second Transform: Translation

```
Enter the translation factor for X and Y: 50 50
```



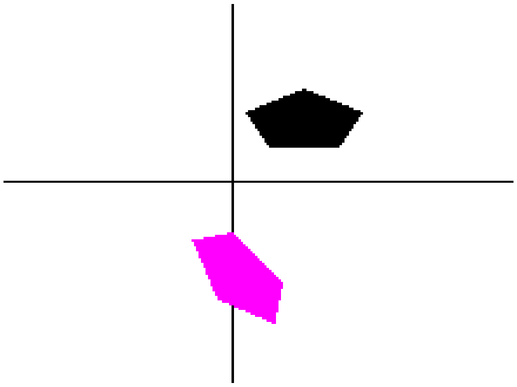
Second Transform: Scaling

Enter the scaling factor for X and Y: 3 3
Enter point about which to be scaled: 0 0



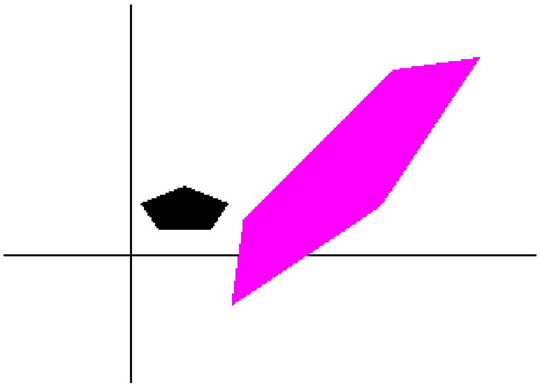
Second Transform: Reflection

Enter the angle with X-axis and Y-intercept of the mirror: 0 0



Second Transform: Shearing

```
Enter the shearing factor for X and Y: 3 3
```

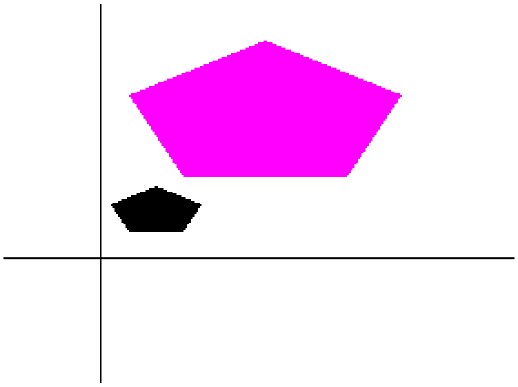


First transform: Scaling

```
Choose first transformation:
1 for Translation
2 for Rotation
3 for Scaling
4 for Reflection
5 for shearing
0 to Exit
Enter your choice: 3
Enter the scaling factor for X and Y: 3 3
Enter point about which to be scaled: 0 0
```

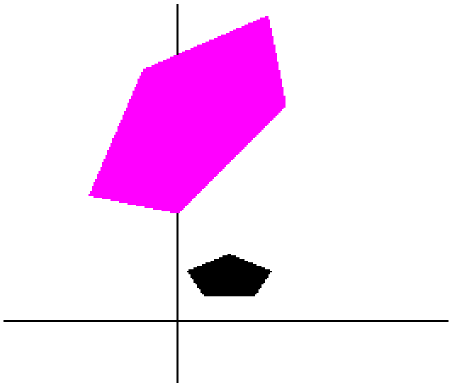
Second Transform: Translation

```
Enter the translation factor for X and Y: 50 50
```



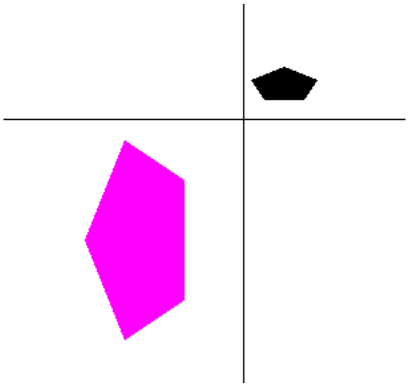
Second Transform: Rotation

Enter the rotation angle: 45
Enter point about which to be rotated: 0 0



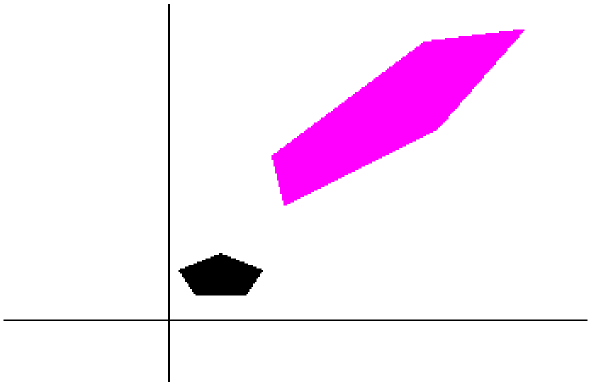
Second Transform: Reflection

Enter the angle with X-axis and Y-intercept of the mirror: 135 0



Second Transform: Shearing

```
Enter the shearing factor for X and Y: 0.5 0.5
```

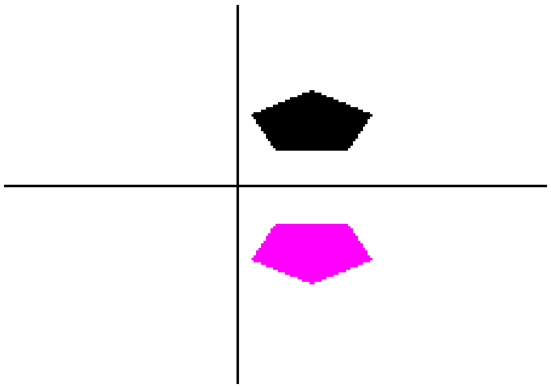


First transform: Reflecttion

```
Choose first transformation:
1 for Translation
2 for Rotation
3 for Scaling
4 for Reflection
5 for shearing
0 to Exit
Enter your choice: 4
Enter the angle with X-axis and Y-intercept of the mirror: 0 0
```

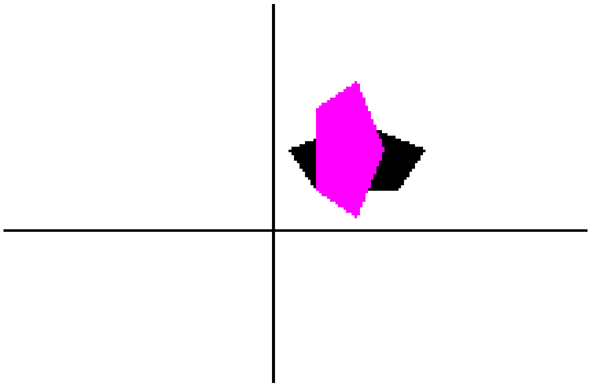
Second Transform: Translation

```
Enter the translation factor for X and Y: 50 50
```



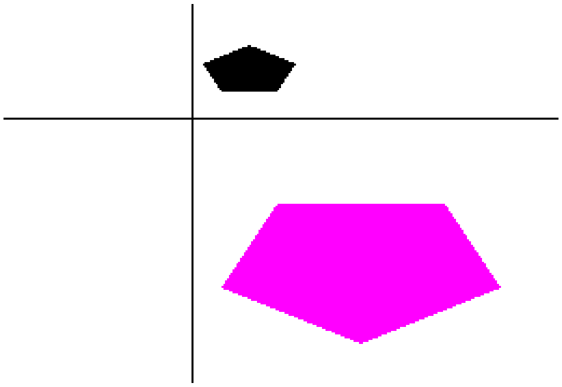
Second Transform: Rotation

```
Enter the rotation angle: 90
Enter point about which to be rotated: 0 0
```



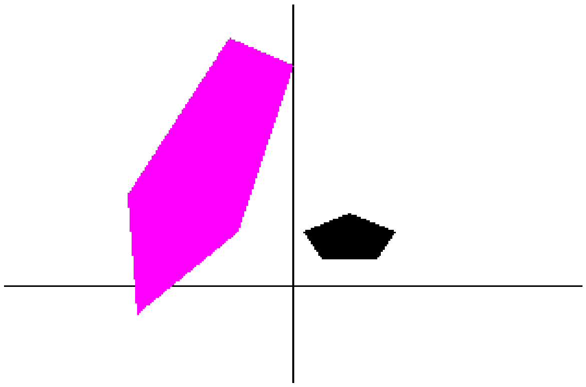
Second Transform: Scaling

```
Enter the scaling factor for X and Y: 3 3
Enter point about which to be scaled: 0 0
```



Second Transform: Shearing

```
Enter the shearing factor for X and Y: 3 3
```

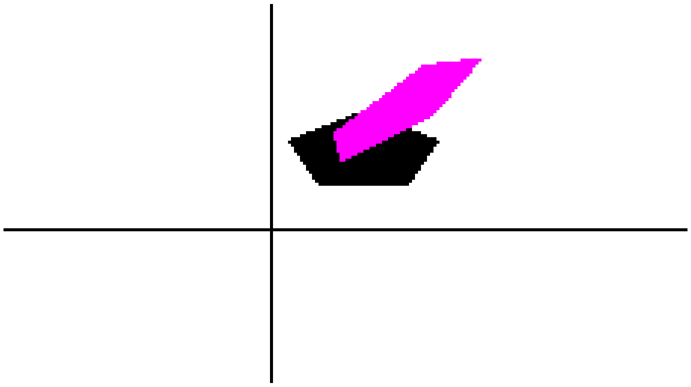


First transform: Shearing

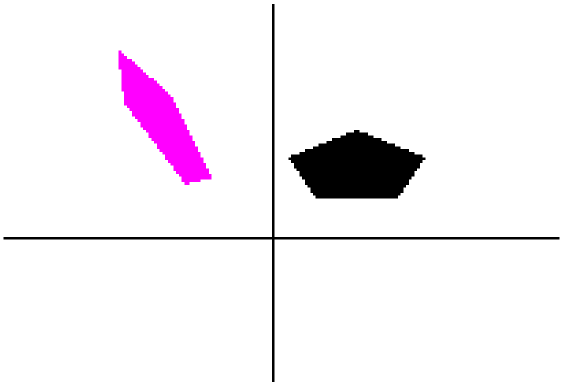
```
Choose first transformation:
1 for Translation
2 for Rotation
3 for Scaling
4 for Reflection
5 for shearing
0 to Exit
Enter your choice: 5
Enter the shearing factor for X and Y: 0.5 0.5
```

Second Transform: Translation

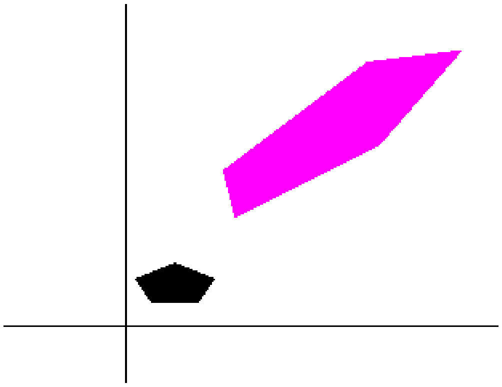
```
Enter the translation factor for X and Y: 40 40
```



Second Transform: Rotation Enter the rotation angle: 90
Enter point about which to be rotated: 0 0

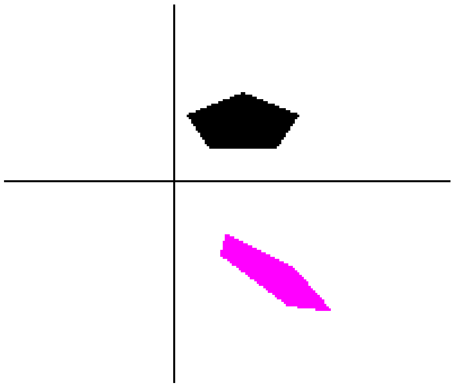


Second Transform: Scaling
Enter the scaling factor for X and Y: 3 3
Enter point about which to be scaled: 0 0



Second Transform: Reflection

```
Enter the angle with X-axis and Y-intercept of the mirror: 0 0
```



Aim:

Create a window with any 2D object and a different sized viewport. Apply window to viewport transformation on the object. Display both window and viewport.

Code:

```
#ifndef LOPENGL_H
#define LOPENGL_H

#include <GL/freeglut.h>
#include <GL/gl.h>
#include <GL/glu.h>
#include <math.h>
#include <stdio.h>
#include<iostream>
#include<vector>
#include<ctime>
using namespace std;

#endif
```

```
#ifndef LUTIL_H
#define LUTIL_H

#include "Headers.h"

//Screen Constants
const int SCREEN_WIDTH = 640;
const int SCREEN_HEIGHT = 480;
const int SCREEN_FPS = 60;
const int POINT_SIZE=2;

double Sx, Sy;

//pairs of the form (min, max)
pair<double, double> window_x_dims, window_y_dims;
pair<double, double> viewport_x_dims, viewport_y_dims;

int edge_count;

vector<pair<double, double>> original_points, transformed_points;

bool initGL();

void update();

void render();

void lineloop(double x1, double y1, double x2, double y2);

void setEdgeCount(int option);

void computeScaleFactor();

void computeTransformedPoints();

void drawWindow();

void drawWindowFigure();

void drawViewport();

void drawViewportFigure();

#endif
```

```
#include "Signatures.h"

bool initGL(){
    //Initialize Projection Matrix
    glMatrixMode( GL_PROJECTION );
    glLoadIdentity();
    gluOrtho2D(0.0,640.0,0.0,480.0);

    //Initialize Modelview Matrix
    glMatrixMode( GL_MODELVIEW );
    glLoadIdentity();

    // glTranslatef( SCREEN_WIDTH / 3.f, SCREEN_HEIGHT / 3.f, 0.f );

    //Initialize clear color
    glClearColor( 0.f, 0.f, 0.f, 1.f );

    glPointSize(POINT_SIZE);
    glEnable(GL_POINT_SMOOTH);

    //Check for error
    GLenum error = glGetError();
    if( error != GL_NO_ERROR )
    {
        printf( "Error initializing OpenGL! %s\n", gluErrorString( error ) );
        return false;
    }

    return true;
}

void update(){

}

void render(){
    drawWindow();
    drawWindowFigure();
    drawViewport();

    computeScaleFactor();
    computeTransformedPoints();

    drawViewportFigure();

    glFlush();
}

void setEdgeCount(int option){
    if(option == 0){
        cout<<"Invalid"<<endl;
    }
    else if(option == 1 || option == 2){
        edge_count = 2;
    }
    else{
        edge_count = option;
    }
}

void lineloop(double x1, double y1, double x2, double y2) {

    glBegin(GL_LINE_LOOP);

    glVertex2d(x1,y1);
    glVertex2d(x2,y1);
    glVertex2d(x2,y2);
    glVertex2d(x1,y2);

    glEnd();
}

void drawWindow(){
    glColor3f(1.0,1.0,1.0);
    lineloop(window_x_dims.first, window_y_dims.first,
    window_x_dims.second, window_y_dims.second);
}

void drawWindowFigure(){
    glColor3f(1.0,1.0,1.0);
    if(edge_count==2)
        glBegin(GL_LINES);
    else
        glBegin(GL_POLYGON);

    for (int i = 0; i < edge_count; i++){
        glVertex2d(original_points[i].first, original_points[i].
    second);
    }
    glEnd();
    glFlush();
}

void drawViewport(){
    glColor3f(0.0,1.0,0.0);
    lineloop(viewport_x_dims.first, viewport_y_dims.first,
    viewport_x_dims.second, viewport_y_dims.second);
}

void drawViewportFigure(){
    glColor3f(0.0,1.0,0.0);
    if(edge_count==2)
        glBegin(GL_LINES);
    else
        glBegin(GL_POLYGON);

    for (int i = 0; i < edge_count; i++){
        glVertex2d(transformed_points[i].first, transformed_points[
    i].second);
    }
    glEnd();
    glFlush();
}

void computeScaleFactor(){

    double xNr = viewport_x_dims.second - viewport_x_dims.first;
    double xDr = window_x_dims.second - window_x_dims.first;

    Sx = xNr/xDr;

    double yNr = viewport_y_dims.second - viewport_y_dims.first;
    double yDr = window_y_dims.second - window_y_dims.first;

    Sy = yNr/yDr;
}

void computeTransformedPoints(){
    for(int i=0;i<edge_count;i++){
        pair<double, double> p = original_points[i];
        double xw = p.first;
        double yw = p.second;

        double xv = viewport_x_dims.first + (xw - window_x_dims.
    first) * Sx;
        double yv = viewport_y_dims.first + (yw - window_y_dims.
    first) * Sy;

        transformed_points.push_back(pair<double, double>(xv, yv));
    }
}
```

```
#include "Helpers.h"

void runMainLoop(int val);

int main( int argc, char* args[] ){

    glutInit( &argc, args );

    glutInitContextVersion( 2, 1 );

    glutInitDisplayMode( GLUT_SINGLE|GLUT_RGB );
    glutInitWindowSize( SCREEN_WIDTH, SCREEN_HEIGHT );
    glutCreateWindow( "OpenGL" );

    cout<<"Enter window dimensions: "<<endl;
    cout<<"Enter minimum X value: "; cin>>window_x_dims.first;
    cout<<"Enter maximum X value: "; cin>>window_x_dims.second;
    cout<<"Enter minimum Y value: "; cin>>window_y_dims.first;
    cout<<"Enter maximum Y value: "; cin>>window_y_dims.second;


    int option=0;
    cout<<"Choose number of edges: (1 for line, 3 and upwards for polygon): ";
    cin>>option;

    setEdgeCount(option);
    cout<<"Enter vertices: "<<endl;
    for(int i=0;i<edge_count;i++){
        cout<<"Vertex "<<i+1<<" (x,y): ";
        double x,y;
        cin>>x>>y;
        original_points.push_back(pair<double, double>(x, y));
    }

    drawWindowFigure();


    cout<<"Enter viewport dimensions: "<<endl;
    cout<<"Enter minimum X value: "; cin>>viewport_x_dims.first;
    cout<<"Enter maximum X value: "; cin>>viewport_x_dims.second;
    cout<<"Enter minimum Y value: "; cin>>viewport_y_dims.first;
    cout<<"Enter maximum Y value: "; cin>>viewport_y_dims.second;


    if( !initGL() )
    {
        printf( "Unable to initialize graphics library!\n" );
        return 1;
    }

    glutDisplayFunc( render );

    glutTimerFunc( 1000 / SCREEN_FPS, runMainLoop, 0 );

    glutMainLoop();

    return 0;
}

void runMainLoop( int val ){
    update();
    render();

    glutTimerFunc( 1000 / SCREEN_FPS, runMainLoop, val );
}
```

Output:

```
Enter window dimensions:
Enter minimum X value: 50
Enter maximum X value: 250
Enter minimum Y value: 50
Enter maximum Y value: 250

    Choose number of edges: (1 for line, 3 and upwards for polygon): 5
Enter vertices:
Vertex 1 (x,y): 80 80
Vertex 2 (x,y): 60 110
Vertex 3 (x,y): 110 130
Vertex 4 (x,y): 160 110
Vertex 5 (x,y): 140 80

    Enter viewport dimensions:
Enter minimum X value: 50
Enter maximum X value: 100
Enter minimum Y value: 300
Enter maximum Y value: 350
```