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();
```

```
56              setTranslateMatrix(factor_x,factor_y, 1);
57          }
58          else if(transformation == 2){
59              double angle;
60              cout<<"Enter the rotation angle: ";
61              cin>>angle;
62              double x,y;
63              cout<<"Enter point about which to be rotated: ";
64              cin>>x>>y;
65              angle_radians = angle * 3.1416 / 180;
66              drawPolygon();
67              setRotateMatrix(angle_radians, x, y, 1);
68          }
69          else if(transformation == 3){
70              cout<<"Enter the scaling factor for X and Y: ";
71              cin>>factor_x >> factor_y;
72              double x,y;
73              cout<<"Enter point about which to be scaled: ";
74              cin>>x>>y;
75              drawPolygon();
76              setScaleMatrix(factor_x,factor_y, x, y, 1);
77          }
78          else if(transformation == 4){
79              double angle;
80              cout<<"Enter the angle with X-axis and Y-intercept of
    the mirror: ";
81              cin>>angle >> factor_y;
82              angle_radians = angle * 3.1416/180;
83              drawPolygon();
84              setReflectMatrix(angle_radians,factor_y, 1);
85          }
86          else if(transformation == 5){
87              cout<<"Enter the shearing factor for X and Y: ";
88              cin>>factor_x >> factor_y;
89              drawPolygon();
90              setShearMatrix(factor_x, factor_y, 1);
91          }
92          else if(transformation){
93              cout<<"Invalid option"<<endl;
94          }
95          else;
96
97          cout<<"Choose second transformation: "<<endl;
98          cout<<"1 for Translation"<<endl<<"2 for Rotation"<<endl;
99          cout<<"3 for Scaling"<<endl<<"4 for Reflection"<<endl;
100         cout<<"5 for shearing"<<endl<<"0 to Exit"<<endl;
101         cout<<"Enter your choice: ";cin>>transformation;
102
103         if(!transformation){
104             return;
105         }
106
107         if(transformation == 1){
108             cout<<"Enter the translation factor for X and Y: ";
109             cin>>factor_x >> factor_y;
110             drawPolygon();
111             setTranslateMatrix(factor_x,factor_y, 2);
```

```
112         }
113         else if(transformation == 2){
114             double angle;
115             cout<<"Enter the rotation angle: ";
116             cin>>angle;
117             double x,y;
118             cout<<"Enter point about which to be rotated: ";
119             cin>>x>>y;
120             angle_radians = angle * 3.1416 / 180;
121             drawPolygon();
122             setRotateMatrix(angle_radians, x, y, 2);
123         }
124         else if(transformation == 3){
125             cout<<"Enter the scaling factor for X and Y: ";
126             cin>>factor_x >> factor_y;
127             double x,y;
128             cout<<"Enter point about which to be scaled: ";
129             cin>>x>>y;
130             drawPolygon();
131             setScaleMatrix(factor_x,factor_y, x, y, 2);
132         }
133         else if(transformation == 4){
134             double angle;
135             cout<<"Enter the angle with X-axis and Y-intercept of
    the mirror: ";
136             cin>>angle >> factor_y;
137             angle_radians = angle * 3.1416/180;
138             drawPolygon();
139             setReflectMatrix(angle_radians,factor_y, 2);
140         }
141         else if(transformation == 5){
142             cout<<"Enter the shearing factor for X and Y: ";
143             cin>>factor_x >> factor_y;
144             drawPolygon();
145             setShearMatrix(factor_x, factor_y, 2);
146         }
147         else if(transformation){
148             cout<<"Invalid option"<<endl;
149         }
150         else;
151 
152         applyTransforms();
153         drawTransfomedPolygon();
154     }
155 
156     glFlush();
157 }
158 
159 void y_axis(){
160     glColor3f(1.0,1.0,1.0);
161     glBegin(GL_LINES);
162         glVertex2d(0, -480.0);
163         glVertex2d(0, 480.0);
164     glEnd();
165     // glFlush();
166 }
167 
```

```
168 void x_axis(){
169     glColor3f(1.0,1.0,1.0);
170     glBegin(GL_LINES);
171         glVertex2d(-640.0, 0);
172         glVertex2d(640.0, 0);
173     glEnd();
174     // glFlush();
175 }
176 
177 void setEdgeCount(int option){
178     if(option == 0){
179         cout<<"Invalid"<<endl;
180     }
181     else if(option == 1 || option == 2){
182         edge_count = 2;
183     }
184     else{
185         edge_count = option;
186     }
187 }
188 
189 vector<vector<double>> matrix_multiplication(double points[][3],
    int pr){
190 
191     vector<vector<double>> final_transform(3);
192     vector<vector<double>> ans(3);
193 
194     for(int i=0;i<pr;i++){
195         for(int j=0;j<3;j++){
196             ans[i].push_back(0);
197         }
198     }
199     for(int i=0;i<3;i++){
200         for(int j=0;j<3;j++){
201             final_transform[i].push_back(0);
202         }
203     }
204 
205     for(int i=0;i<3;i++){
206         for(int j=0;j<3;j++){
207             for(int k=0;k<3;k++){
208                 final_transform[i][j] += (transform1[i][k] *
    transform2[k][j]);
209             }
210         }
211     }
212     for(int i=0;i<pr;i++){
213         for(int j=0;j<3;j++){
214             for(int k=0;k<3;k++){
215                 ans[i][j] += (points[i][k] * final_transform[k][j])
    ;
216             }
217         }
218     }
219     return ans;
220 }
221 
```

```
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;
                }
```

```
277             }
278         }
279         transform1[0][2] = x;
280         transform1[1][2] = y;
281     }
282     else{
283         for(int i=0;i<3;i++){
284             for(int j=0;j<3;j++){
285                 if(i == j){
286                     transform2[i][j] = 1;
287                 }
288                 else{
289                     transform2[i][j] = 0;
290                 }
291             }
292         }
293         transform2[0][2] = x;
294         transform2[1][2] = y;
295     }
296 }
297
298 void setRotateMatrix(double angle_radians, double x, double y, int
    transform_number){
299
300     double adjust_matrix[3][3] = {{1, 0, -x},
301                                   {0, 1, -y},
302                                   {0, 0, 1}};
303     double reset_matrix[3][3] = {{1, 0, x},
304                                  {0, 1, y},
305                                  {0, 0, 1}};
306     double dummy_matrix[3][3] = {{cos(angle_radians), sin(
    angle_radians), 0},
307                                  {-sin(angle_radians), cos(
    angle_radians), 0},
308                                  {0, 0, 1}};
309
310     double final_transform[3][3];
311     if(angle_radians<0){
312         dummy_matrix[0][1] *= -1;
313         dummy_matrix[1][0] *= -1;
314     }
315
316     for(int i=0;i<3;i++){
317         for(int j=0;j<3;j++){
318             final_transform[i][j] = 0;
319         }
320     }
321
322     for(int i=0;i<3;i++){
323         for(int j=0;j<3;j++){
324             for(int k=0;k<3;k++){
325                 final_transform[i][j] += (reset_matrix[i][k] *
    dummy_matrix[k][j]);
326             }
327         }
328     }
329
```

```
330
331     if(transform_number == 1){
332         for(int i=0;i<3;i++){
333             for(int j=0;j<3;j++){
334                 transform1[i][j] = 0;
335             }
336         }
337         for(int i=0;i<3;i++){
338             for(int j=0;j<3;j++){
339                 for(int k=0;k<3;k++){
340                     transform1[i][j] += (final_transform[i][k] *
    adjust_matrix[k][j]);
341                 }
342             }
343         }
344     }
345     else{
346         for(int i=0;i<3;i++){
347             for(int j=0;j<3;j++){
348                 transform2[i][j] = 0;
349             }
350         }
351         for(int i=0;i<3;i++){
352             for(int j=0;j<3;j++){
353                 for(int k=0;k<3;k++){
354                     transform2[i][j] += (final_transform[i][k] *
    adjust_matrix[k][j]);
355                 }
356             }
357         }
358     }
359 }
360
361 void setScaleMatrix(double x, double y, double xf, double yf, int
    transform_number){
362     if(transform_number == 1){
363         for(int i=0;i<3;i++){
364             for(int j=0;j<3;j++){
365                 if(i == j){
366                     transform1[i][j] = 1;
367                 }
368                 else{
369                     transform1[i][j] = 0;
370                 }
371             }
372         }
373         transform1[0][0] = x;
374         transform1[1][1] = y;
375     }
376     else{
377         for(int i=0;i<3;i++){
378             for(int j=0;j<3;j++){
379                 if(i == j){
380                     transform2[i][j] = 1;
381                 }
382                 else{
383                     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){
```

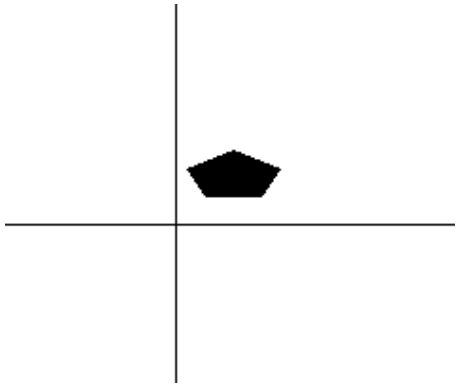
```
                    transform2[i][j] = 1;
                }
                else{
                    transform2[i][j] = 0;
                }
            }
        }
        transform2[0][1] = shy;
        transform2[1][0] = shx;
    }
}
```

```
#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();

```

```
46      return 0;
47  }
48
49  void runMainLoop( int val ){
50      update();
51      render();
52
53      glutTimerFunc( 1000 / SCREEN_FPS, runMainLoop, val );
54  }
```

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



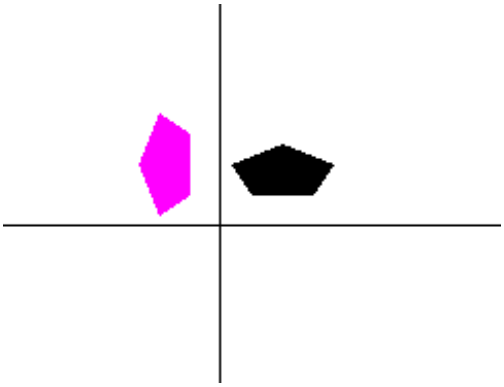
First transform: Translation

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

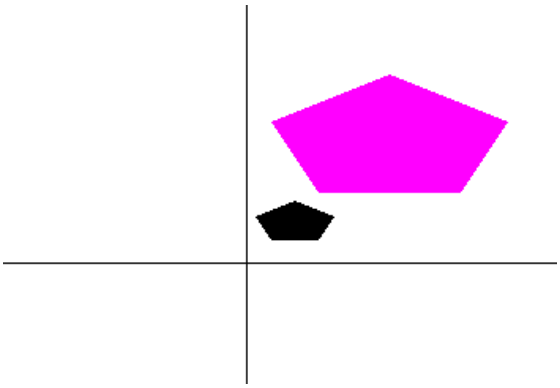
Second Transform: Rotation

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



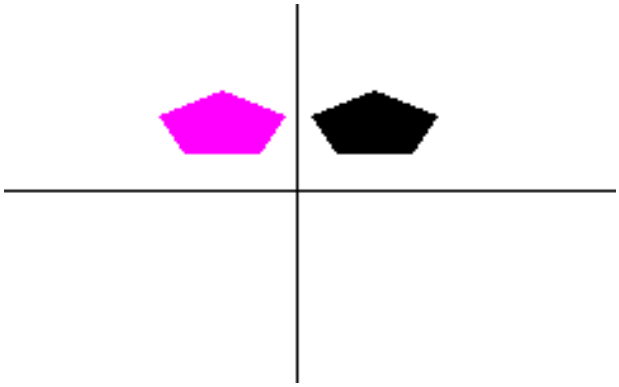
Second Transform: Scaling

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



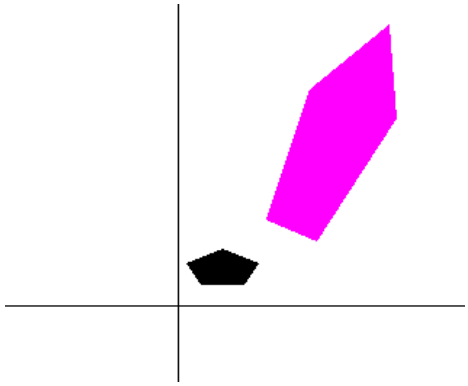
Second Transform: Reflection

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

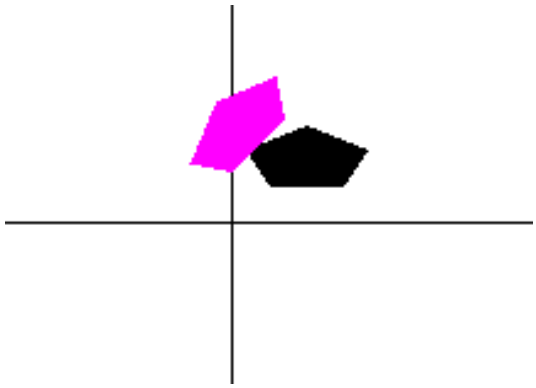


First transform: Rotation

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

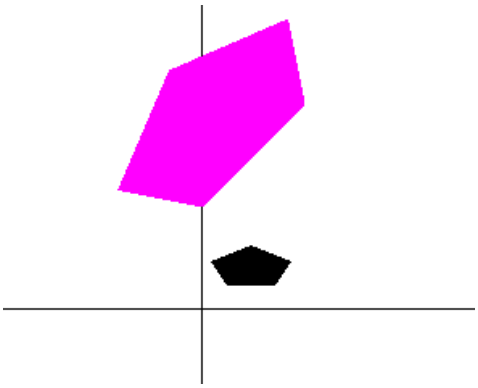
Second Transform: Translation

Enter the translation factor for X and Y: 50 50



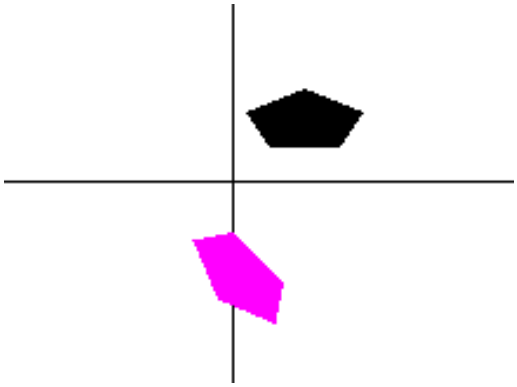
Second Transform: Scaling

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



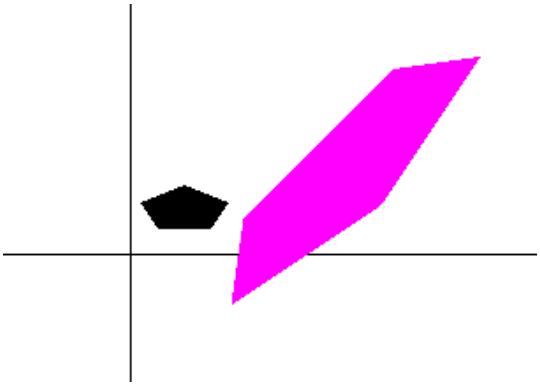
Second Transform: Reflection

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

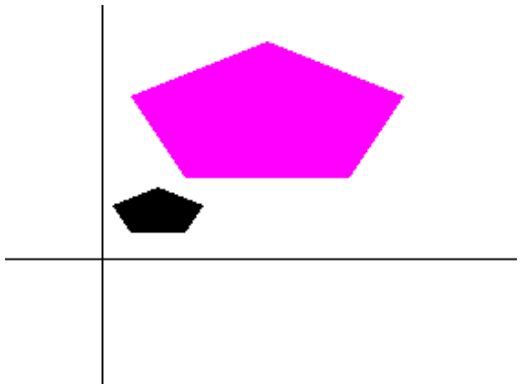


First transform: Scaling

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

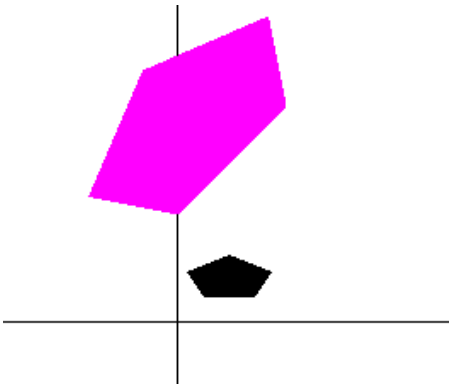
Enter the translation factor for X and Y: 50 50



Second Transform: Rotation

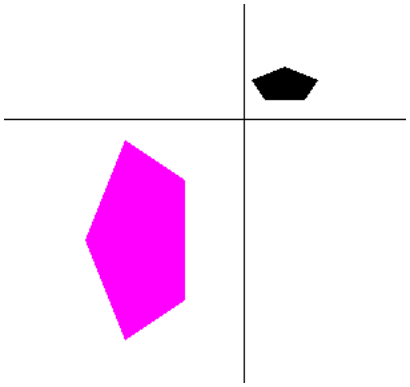
Enter the rotation angle: 45

Enter point about which to be rotated: 0 0



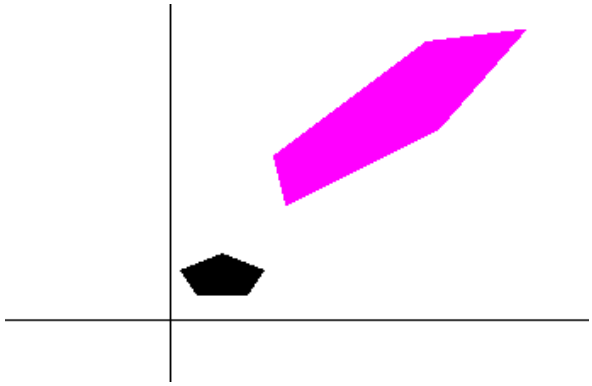
Second Transform: Reflection

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

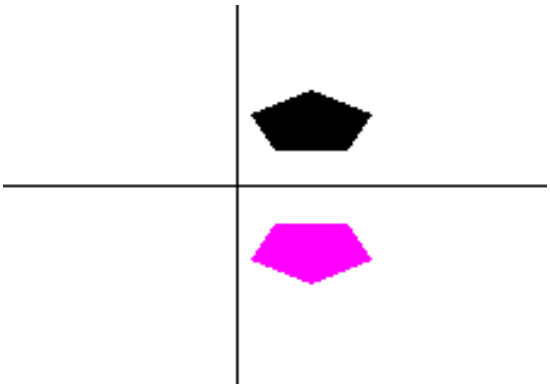


First transform: Reflecttion

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

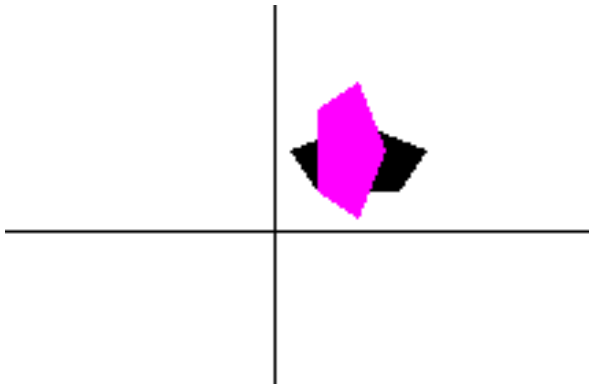
Second Transform: Translation

Enter the translation factor for X and Y: 50 50



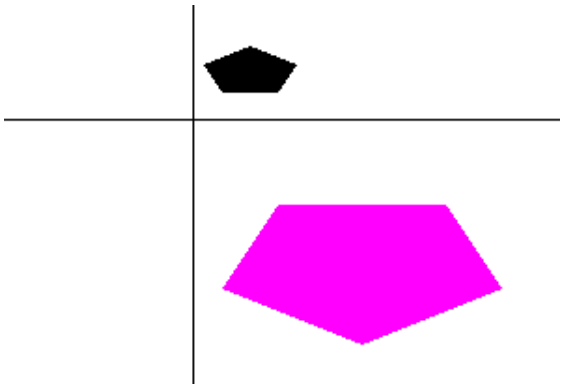
Second Transform: Rotation

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



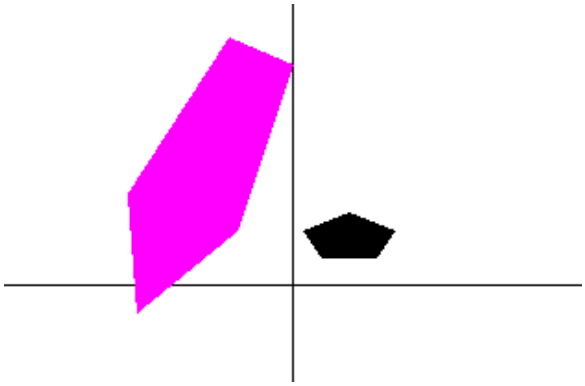
Second Transform: Scaling

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

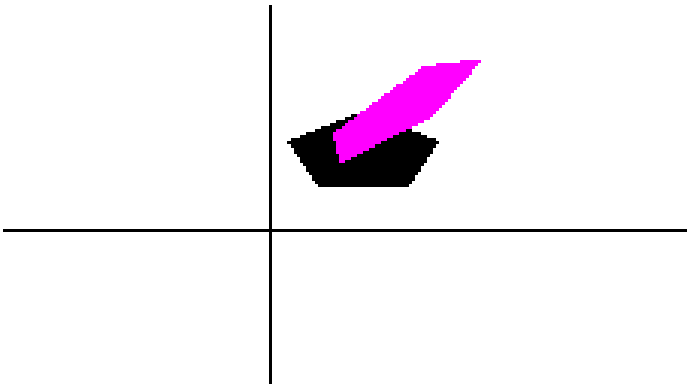


First transform: Shearing

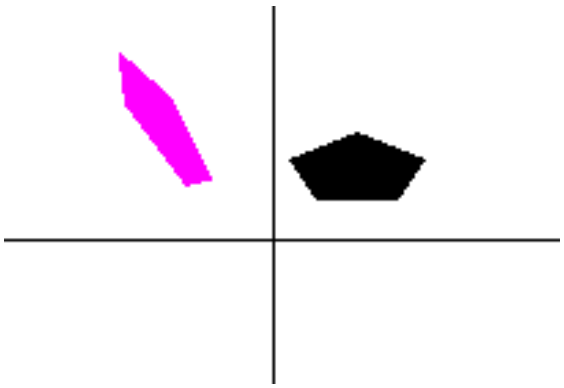
Choose first transformation:
1 for Translation
2 for Rotation
3 for Scaling
4 for Reflection
5 for shearing
0 to Exit
Enter your choice: 5
Enter the shearing factor for X and Y: 0.5 0.5

Second Transform: Translation

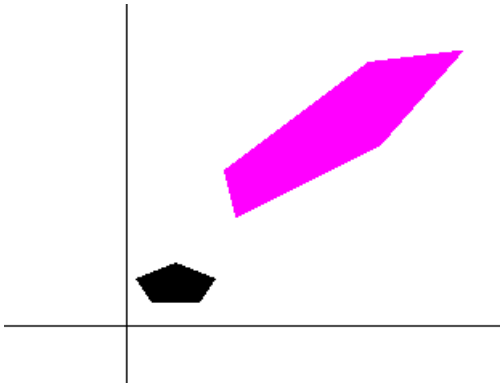
Enter the translation factor for X and Y: 40 40



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

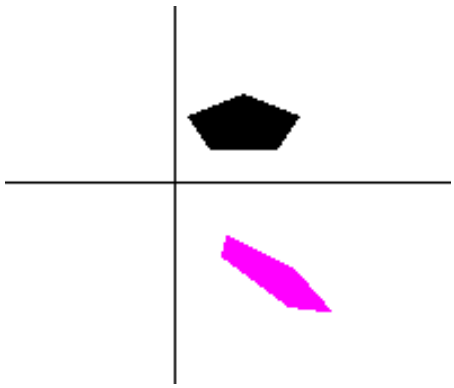


Second Transform: Scaling
Enter the scaling factor for X and Y: 3 3
Enter point about which to be scaled: 0 0



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();

```

```
28 void lineloop(double x1, double y1, double x2, double y2);
29
30 void setEdgeCount(int option);
31
32 void computeScaleFactor();
33
34 void computeTransformedPoints();
35
36 void drawWindow();
37
38 void drawWindowFigure();
39
40 void drawViewport();
41
42 void drawViewportFigure();
43
44 #endif
```

```
1  #include "Signatures.h"
2
3  bool initGL(){
4      //Initialize Projection Matrix
5      glMatrixMode( GL_PROJECTION );
6      glLoadIdentity();
7      gluOrtho2D(0.0,640.0,0.0,480.0);
8
9      //Initialize Modelview Matrix
10     glMatrixMode( GL_MODELVIEW );
11     glLoadIdentity();
12
13     // glTranslatef( SCREEN_WIDTH / 3.f, SCREEN_HEIGHT / 3.f, 0.f )
       ;
14
15     //Initialize clear color
16     glClearColor( 0.f, 0.f, 0.f, 1.f );
17
18     glPointSize(POINT_SIZE);
19     glEnable(GL_POINT_SMOOTH);
20
21     //Check for error
22     GLenum error = glGetError();
23     if( error != GL_NO_ERROR )
24     {
25         printf( "Error initializing OpenGL! %s\n", gluErrorString(
       error ) );
26         return false;
27     }
28
29     return true;
30 }
31
32 void update(){
33
34 }
35
36 void render(){
37     drawWindow();
38     drawWindowFigure();
```

```
39      drawViewport();
40
41      computeScaleFactor();
42      computeTransformedPoints();
43
44      drawViewportFigure();
45
46      glFlush();
47  }
48
49  void setEdgeCount(int option){
50      if(option == 0){
51          cout<<"Invalid"<<endl;
52      }
53      else if(option == 1 || option == 2){
54          edge_count = 2;
55      }
56      else{
57          edge_count = option;
58      }
59  }
60
61  void lineloop(double x1, double y1, double x2, double y2) {
62
63      glBegin(GL_LINE_LOOP);
64
65      glVertex2d(x1,y1);
66      glVertex2d(x2,y1);
67      glVertex2d(x2,y2);
68      glVertex2d(x1,y2);
69
70      glEnd();
71  }
72
73  void drawWindow(){
74      glColor3f(1.0,1.0,1.0);
75      lineloop(window_x_dims.first, window_y_dims.first,
        window_x_dims.second, window_y_dims.second);
76  }
77
78  void drawWindowFigure(){
79      glColor3f(1.0,1.0,1.0);
80      if(edge_count==2)
81          glBegin(GL_LINES);
82      else
83          glBegin(GL_POLYGON);
84
85      for (int i = 0; i < edge_count; i++){
86          glVertex2d(original_points[i].first, original_points[i].
        second);
87      }
88      glEnd();
89      glFlush();
90  }
91
92  void drawViewport(){
93      glColor3f(0.0,1.0,0.0);
```

```
94      lineloop(viewport_x_dims.first, viewport_y_dims.first,
        viewport_x_dims.second, viewport_y_dims.second);
95  }
96
97  void drawViewportFigure(){
98      glColor3f(0.0,1.0,0.0);
99      if(edge_count==2)
100         glBegin(GL_LINES);
101     else
102         glBegin(GL_POLYGON);
103
104     for (int i = 0; i < edge_count; i++){
105         glVertex2d(transformed_points[i].first, transformed_points[
        i].second);
106     }
107     glEnd();
108     glFlush();
109 }
110
111 void computeScaleFactor(){
112
113     double xNr = viewport_x_dims.second - viewport_x_dims.first;
114     double xDr = window_x_dims.second - window_x_dims.first;
115
116     Sx = xNr/xDr;
117
118     double yNr = viewport_y_dims.second - viewport_y_dims.first;
119     double yDr = window_y_dims.second - window_y_dims.first;
120
121     Sy = yNr/yDr;
122 }
123
124 void computeTransformedPoints(){
125     for(int i=0;i<edge_count;i++){
126         pair<double, double> p = original_points[i];
127         double xw = p.first;
128         double yw = p.second;
129
130         double xv = viewport_x_dims.first + (xw - window_x_dims.
        first) * Sx;
131         double yv = viewport_y_dims.first + (yw - window_y_dims.
        first) * Sy;
132
133         transformed_points.push_back(pair<double, double>(xv, yv));
134     }
135 }
```

```
1   #include "Helpers.h"
2
3   void runMainLoop(int val);
4
5   int main( int argc, char* args[] ){
6
7       glutInit( &argc, args );
8
9       glutInitContextVersion( 2, 1 );
10
11      glutInitDisplayMode( GLUT_SINGLE|GLUT_RGB );
```

```
12      glutInitWindowSize( SCREEN_WIDTH, SCREEN_HEIGHT );
13      glutCreateWindow( "OpenGL" );
14
15      cout<<"Enter window dimensions: "<<endl;
16      cout<<"Enter minimum X value: "; cin>>window_x_dims.first;
17      cout<<"Enter maximum X value: "; cin>>window_x_dims.second;
18      cout<<"Enter minimum Y value: "; cin>>window_y_dims.first;
19      cout<<"Enter maximum Y value: "; cin>>window_y_dims.second;
20
21
22      int option=0;
23      cout<<"Choose number of edges: (1 for line, 3 and upwards for
        polygon): ";
24      cin>>option;
25
26      setEdgeCount(option);
27      cout<<"Enter vertices: "<<endl;
28      for(int i=0;i<edge_count;i++){
29          cout<<"Vertex "<<i+1<<" (x,y): ";
30          double x,y;
31          cin>>x>>y;
32          original_points.push_back(pair<double, double>(x, y));
33      }
34
35      drawWindowFigure();
36
37
38      cout<<"Enter viewport dimensions: "<<endl;
39      cout<<"Enter minimum X value: "; cin>>viewport_x_dims.first;
40      cout<<"Enter maximum X value: "; cin>>viewport_x_dims.second;
41      cout<<"Enter minimum Y value: "; cin>>viewport_y_dims.first;
42      cout<<"Enter maximum Y value: "; cin>>viewport_y_dims.second;
43
44
45      if( !initGL() )
46      {
47          printf( "Unable to initialize graphics library!\n" );
48          return 1;
49      }
50
51      glutDisplayFunc( render );
52
53      glutTimerFunc( 1000 / SCREEN_FPS, runMainLoop, 0 );
54
55      glutMainLoop();
56
57      return 0;
58  }
59
60  void runMainLoop( int val ){
61      update();
62      render();
63
64      glutTimerFunc( 1000 / SCREEN_FPS, runMainLoop, val );
65  }
```

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