

Department of Computer Science and Engineering

Shivanirudh S G, 185001146, Semester VII

26 August 2021

UCS1712 - Graphics and Multimedia Lab

Exercise 5: 2D Transformations in C++ using OpenGL

Objective:

To apply the 2D transformations on objects and to render the final output along with the original object.

Code:

```
1  #ifndef LOPENGL_H
2  #define LOPENGL_H
3
4  #include <GL/freeglut.h>
5  #include <GL/gl.h>
6  #include <GL/glu.h>
7  #include <math.h>
8  #include <stdio.h>
9  #include <iostream>
10 #include <vector>
11 #include <ctime>
12 using namespace std;
13
14 #endif
```

```

1  #ifndef LUTIL_H
2  #define LUTIL_H
3
4  #include "Headers.h"
5
6  //Screen Constants
7  const int SCREEN_WIDTH = 640;
8  const int SCREEN_HEIGHT = 480;
9  const int SCREEN_FPS = 60;
10 const int POINT_SIZE=3;
11 vector<int> X_points;
12 vector<int> Y_points;
13
14 vector<pair<double, double>> transforms;
15
16 int edge_count;
17 int transformation;
18 double factor_x, factor_y;
19 double angle_radians;
20
21 bool initGL();
22
23 void update();
24
25 void render();
26
27 void y_axis();
28
29 void x_axis();
30
31 void setEdgeCount(int option);
32
33 vector<vector<double>> matrix_multiplication(double points[][3],
34         double matrix[][3], int r1, int c1, int r2, int c2);
35
36 void drawPolygon();
37
38 void drawTranslatedPolygon(double x, double y);
39
40 void drawRotatedPolygon(double angle_radians, double x, double y);
41
42 void drawScaledPolygon(double x, double y, double xf, double yf);
43
44 void drawReflectedPolygon(double angle_radians, double intercept);
45
46 void drawShearedPolygon(double shx, double shy);
47
48 #endif
49
50 #include "Signatures.h"
51
52 bool initGL(){
53
54     //Initialize Projection Matrix
55     glMatrixMode( GL_PROJECTION );
56     glLoadIdentity();
57     gluOrtho2D(-640.0,640.0,-480.0,480.0);
58
59 }

```

```

10 //Initialize clear color
11 glClearColor( 0.f, 0.f, 0.f, 1.f );
12 glColor3f(0.0f, 0.0f, 0.0f);
13
14 glPointSize(POINT_SIZE);
15 glEnable(GL_POINT_SMOOTH);
16
17 //Check for error
18 GLenum error = glGetError();
19 if( error != GL_NO_ERROR )
20 {
21     printf( "Error initializing OpenGL! %s\n", gluErrorString(
22         error ) );
23     return false;
24 }
25 return true;
26 }
27
28 void update(){
29
30 }
31
32 void render(){
33     glClear(GL_COLOR_BUFFER_BIT);
34     glColor3f(1.0, 1.0, 1.0);
35     drawPolygon();
36     glFlush();
37
38     while(true){
39         glClear(GL_COLOR_BUFFER_BIT);
40         glColor3f(1.0, 1.0, 1.0);
41
42         cout<<"Choose transformation: "<<endl;
43         cout<<"1 for Translation"<<endl<<"2 for Rotation"<<endl;
44         cout<<"3 for Scaling"<<endl<<"4 for Reflection"<<endl;
45         cout<<"5 for shearing"<<endl<<"0 to Exit"<<endl;
46         cout<<"Enter your choice: ";cin>>transformation;
47
48         cout<<"transformation: "<<transformation<<endl;
49         if(!transformation){
50             return;
51         }
52
53         if(transformation == 1){
54             cout<<"Enter the translation factor for X and Y: ";
55             cin>>factor_x >> factor_y;
56             drawPolygon();
57             drawTranslatedPolygon(factor_x,factor_y);
58         }
59         else if(transformation == 2){
60             double angle;
61             cout<<"Enter the rotation angle: ";
62             cin>>angle;
63             double x,y;
64             cout<<"Enter point about which to be rotated: ";
65             cin>>x>>y;

```

```

66         angle_radians = angle * 3.1416 / 180;
67         drawPolygon();
68         drawRotatedPolygon(angle_radians, x, y);
69     }
70     else if(transformation == 3){
71         cout<<"Enter the scaling factor for X and Y: ";
72         cin>>factor_x >> factor_y;
73         double x,y;
74         cout<<"Enter point about which to be scaled: ";
75         cin>>x>>y;
76         drawPolygon();
77         drawScaledPolygon(factor_x,factor_y, x, y);
78     }
79     else if(transformation == 4){
80         double angle;
81         cout<<"Enter the angle with X-axis and Y-intercept of
the mirror: ";
82         cin>>angle >> factor_y;
83         angle_radians = angle * 3.1416/180;
84         drawPolygon();
85         drawReflectedPolygon(angle_radians,factor_y);
86     }
87     else if(transformation == 5){
88         cout<<"Enter the shearing factor for X and Y: ";
89         cin>>factor_x >> factor_y;
90         drawPolygon();
91         drawShearedPolygon(factor_x, factor_y);
92     }
93     else if(transformation){
94         cout<<"Invalid option"<<endl;
95     }
96     else;
97 }
98
99 glFlush();
100 }
101
102 void y_axis(){
103     glColor3f(1.0,1.0,1.0);
104     glBegin(GL_LINES);
105         glVertex2d(0, -480.0);
106         glVertex2d(0, 480.0);
107     glEnd();
108     // glFlush();
109 }
110
111 void x_axis(){
112     glColor3f(1.0,1.0,1.0);
113     glBegin(GL_LINES);
114         glVertex2d(-640.0, 0);
115         glVertex2d(640.0, 0);
116     glEnd();
117     // glFlush();
118 }
119
120 void setEdgeCount(int option){
121     if(option == 0){

```

```

122         cout<<"Invalid"<<endl;
123     }
124     else if(option == 1 || option == 2){
125         edge_count = 2;
126     }
127     else{
128         edge_count = option;
129     }
130 }
131
132 vector<vector<double>> matrix_multiplication(double points[][3],
133     double matrix[][3], int r1, int c1, int r2, int c2){
134     vector<vector<double>> ans(3);
135
136     for(int i=0;i<r1;i++){
137         for(int j=0;j<c2;j++){
138             ans[i].push_back(0);
139         }
140     }
141     for(int i=0;i<r1;i++){
142         for(int j=0;j<c2;j++){
143             for(int k=0;k<r2;k++){
144                 ans[i][j] += (points[i][k] * matrix[k][j]);
145             }
146         }
147     }
148     return ans;
149 }
150
151 double round(double d){
152     return floor(d + 0.5);
153 }
154
155 void drawPolygon(){
156     y_axis();
157     x_axis();
158     glColor3f(1.0,1.0,1.0);
159     if(edge_count==2)
160         glBegin(GL_LINES);
161     else
162         glBegin(GL_POLYGON);
163
164     for (int i = 0; i < edge_count; i++){
165         glVertex2d(X_points[i], Y_points[i]);
166     }
167     glEnd();
168     glFlush();
169 }
170
171 void drawTranslatedPolygon(double x, double y){
172     double translate_matrix[2][1] = {{x}, {y}};
173     for(int i=0;i<edge_count;i++){
174         double new_x = X_points[i] + translate_matrix[0][0];
175         double new_y = Y_points[i] + translate_matrix[1][0];
176         transforms.push_back(pair<double, double>(new_x, new_y));
177     }

```

```

178     glFlush();
179     glColor3f(0.0, 1.0, 0.0);
180     if (edge_count == 2)
181         glBegin(GL_LINES);
182     else
183         glBegin(GL_POLYGON);
184
185     for(pair<double, double> p: transforms){
186         glVertex2d(p.first, p.second);
187     }
188     glEnd();
189     glFlush();
190     transforms.clear();
191 }
192
193 void drawRotatedPolygon(double angle_radians, double x, double y){
194     double rotate_matrix[2][3] = {{cos(angle_radians), sin(
195         angle_radians)}, {-sin(angle_radians), cos(angle_radians)}};
196     double adjust_matrix[1][2] = {-x, -y};
197     double reset_matrix[1][2] = {x, y};
198     for(int i=0; i<edge_count; i++){
199         double points[1][3] = {{X_points[i]+adjust_matrix[0][0],
200             Y_points[i]+adjust_matrix[0][1]}};
201         vector<vector<double>> ans = matrix_multiplication(points,
202             rotate_matrix, 1, 2, 2, 2);
203         transforms.push_back(pair<double, double>(ans[0][0]+
204             reset_matrix[0][0], ans[0][1]+reset_matrix[0][1]));
205     }
206
207     glFlush();
208     glColor3f(0.0, 1.0, 0.0);
209     if (edge_count == 2)
210         glBegin(GL_LINES);
211     else
212         glBegin(GL_POLYGON);
213
214     for(pair<double, double> p: transforms){
215         glVertex2d(round(p.first), round(p.second));
216     }
217     glEnd();
218     glFlush();
219     transforms.clear();
220 }
221
222 void drawScaledPolygon(double x, double y, double xf, double yf){
223     double scale_matrix[2][3] = {{x, 0}, {0, y}};
224     for(int i=0; i<edge_count; i++){
225         double points[1][3] = {{X_points[i], Y_points[i]}};
226         vector<vector<double>> ans = matrix_multiplication(points,
227             scale_matrix, 1, 2, 2, 2);
228         double x_impact = xf*(1-x);
229         double y_impact = yf*(1-y);
230         transforms.push_back(pair<double, double>(ans[0][0]+
231             x_impact, ans[0][1]+y_impact));
232     }
233
234     glFlush();

```

```

229     glColor3f(0.0, 1.0, 0.0);
230     if (edge_count == 2)
231         glBegin(GL_LINES);
232     else
233         glBegin(GL_POLYGON);
234
235     for(pair<double, double> p: transforms){
236         glVertex2d(round(p.first), round(p.second));
237     }
238     glEnd();
239     glFlush();
240     transforms.clear();
241 }
242
243 void drawReflectedPolygon(double angle_radians, double intercept){
244     double reflect_matrix[3][3] = {{cos(2*angle_radians), sin(2*
245                                     angle_radians), intercept*sin(2*angle_radians)},
246                                     {sin(2*angle_radians), -cos(2*
247                                     angle_radians), -intercept*cos(2*angle_radians)+1},
248                                     {0, 0, 1}};
249     for(int i=0;i<edge_count;i++){
250         double points[1][3] = {{X_points[i], Y_points[i], 0}};
251         vector<vector<double>> ans = matrix_multiplication(points,
252 reflect_matrix, 1, 3, 3, 3);
253         transforms.push_back(pair<double, double>(ans[0][0], ans
254 [0][1]));
255     }
256
257     glFlush();
258     glColor3f(0.0, 1.0, 0.0);
259     if (edge_count == 2)
260         glBegin(GL_LINES);
261     else
262         glBegin(GL_POLYGON);
263
264     for(pair<double, double> p: transforms){
265         glVertex2d(round(p.first), round(p.second));
266     }
267     glEnd();
268     glFlush();
269     transforms.clear();
270 }
271
272 void drawShearedPolygon(double shx, double shy){
273     double reflect_matrix[3][3] = {{1, shy, 0},
274                                     {shx, 1, 0},
275                                     {0, 0, 1}};
276     for(int i=0;i<edge_count;i++){
277         double points[1][3] = {{X_points[i], Y_points[i], 0}};
278         vector<vector<double>> ans = matrix_multiplication(points,
279 reflect_matrix, 1, 3, 3, 3);
280         transforms.push_back(pair<double, double>(ans[0][0], ans
281 [0][1]));
282     }
283
284     glFlush();
285     glColor3f(0.0, 1.0, 0.0);

```

```

280     if (edge_count == 2)
281         glBegin(GL_LINES);
282     else
283         glBegin(GL_POLYGON);
284
285     for(pair<double, double> p: transforms){
286         glVertex2d(round(p.first), round(p.second));
287     }
288     glEnd();
289     glFlush();
290     transforms.clear();
291 }

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      int option=0;
16      cout<<"Choose number of edges: (1 for line, 3 and upwards for
17      polygon): ";
18      cin>>option;
19
20      setEdgeCount(option);
21      cout<<"Enter vertices: "<<endl;
22      for(int i=0;i<edge_count;i++){
23          cout<<"Vertex "<<i+1<<" (x,y): ";
24          int x,y;
25          cin>>x>>y;
26          X_points.push_back(x);
27          Y_points.push_back(y);
28      }
29
30      drawPolygon();
31      cout<<"Number of edges: "<<edge_count<<endl;
32
33      if( !initGL() )
34      {
35          printf( "Unable to initialize graphics library!\n" );
36          return 1;
37      }
38
39
40      glutDisplayFunc( render );
41
42      glutTimerFunc( 1000 / SCREEN_FPS, runMainLoop, 0 );
43
44      glutMainLoop();

```



```
45
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

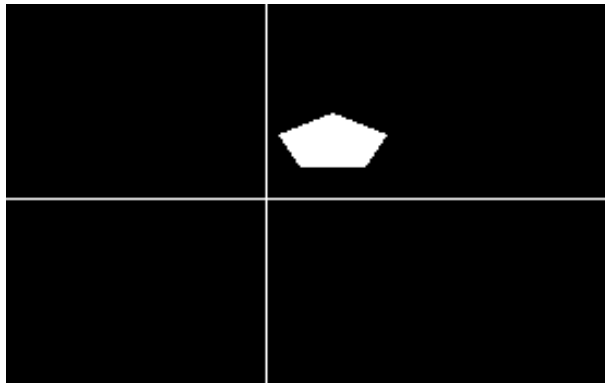
Vertex 2 (x,y): 10 60

Vertex 3 (x,y): 60 80

Vertex 4 (x,y): 110 60

Vertex 5 (x,y): 90 30

Number of edges: 5



Translation:

Choose transformation:

1 for Translation

2 for Rotation

3 for Scaling

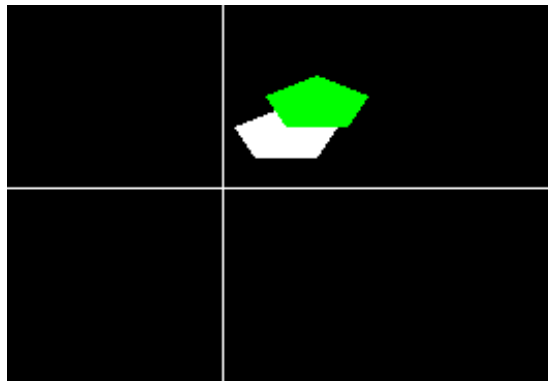
4 for Reflection

5 for shearing

0 to Exit

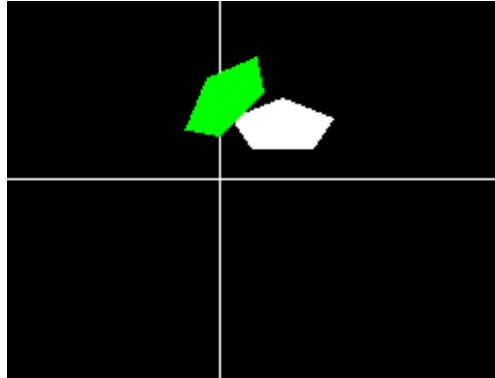
Enter your choice: 1

Enter the translation factor for X and Y: 30 30



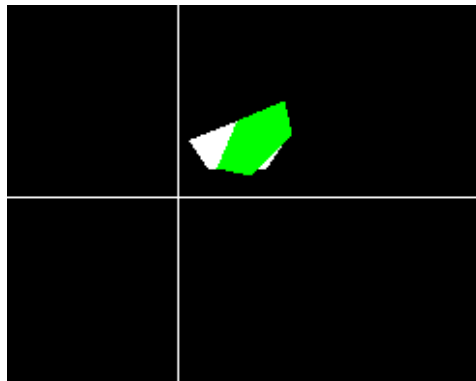
Rotation:
About origin:

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



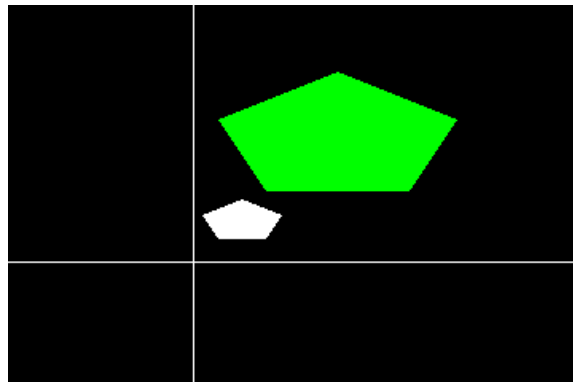
About fixed point:

Enter the rotation angle: 45
Enter point about which to be rotated: 60 80

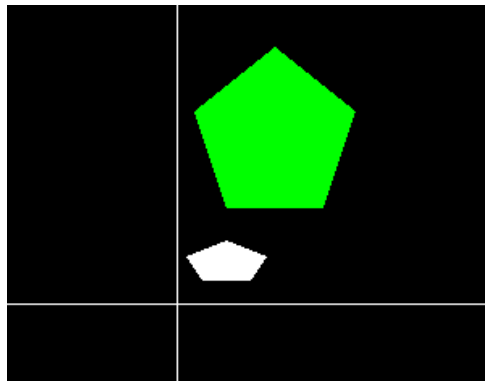


Scaling:
About origin - Uniform:

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



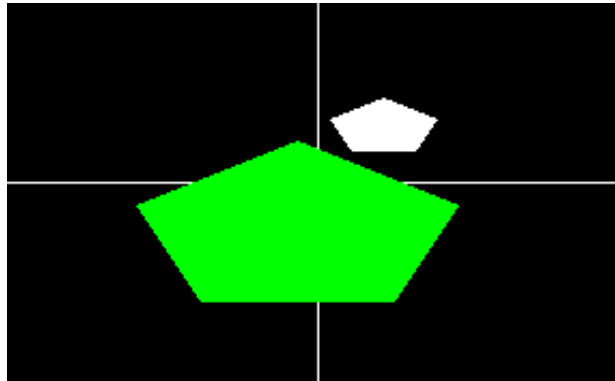
About origin - Differential:
Enter the scaling factor for X and Y: 2 4
Enter point about which to be scaled: 0 0



About fixed point:

Enter the scaling factor for X and Y: 3 3

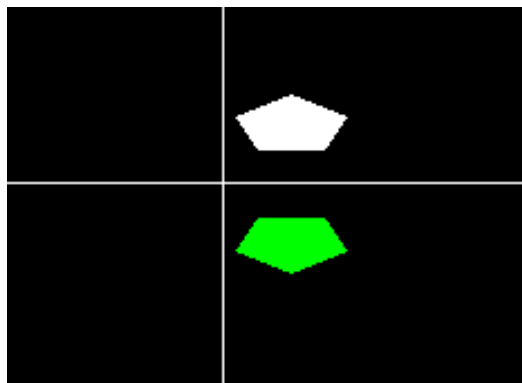
Enter point about which to be scaled: 100 100



Reflection:

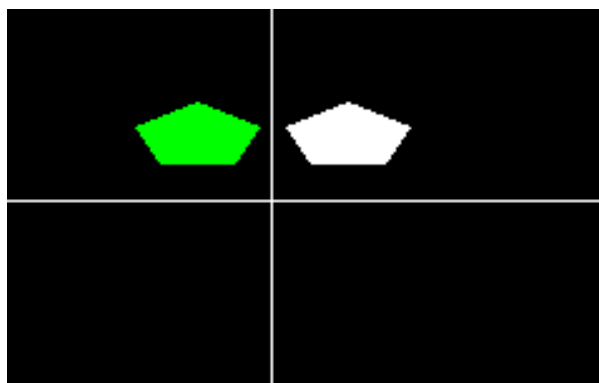
With respect to X-axis:

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



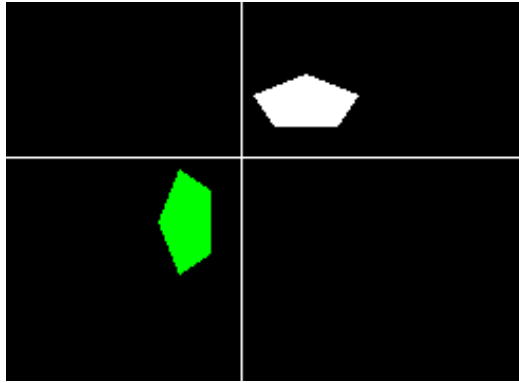
With respect to Y-axis:

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



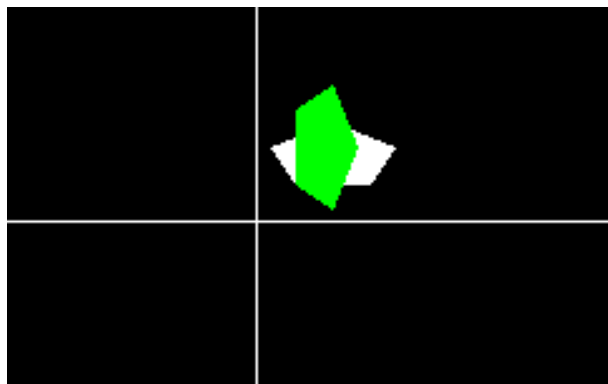
With respect to origin:

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



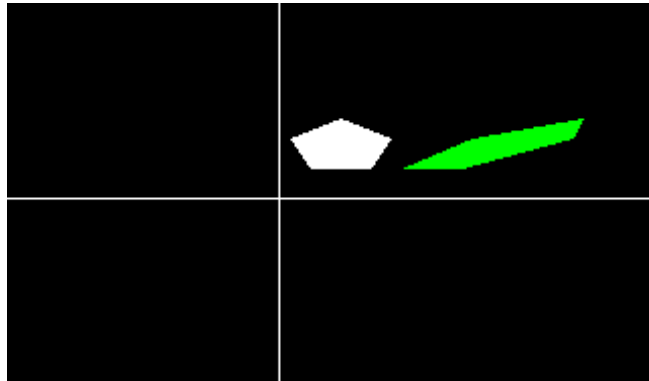
With respect to line $x=y$:

Enter the angle with X-axis and Y-intercept of the mirror: 45 0



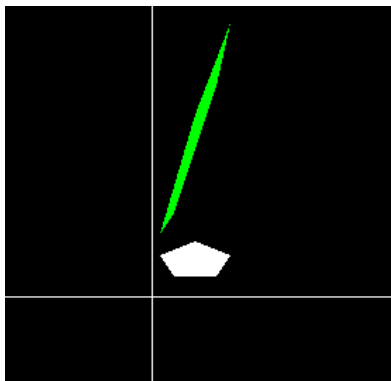
Shearing:
X-direction:

Enter the shearing factor for X and Y: 3 0



Y-direction:

Enter the shearing factor for X and Y: 0 3



Both X and Y directions:

Enter the shearing factor for X and Y: 3 3

