

EX-05 - 2D TRANSFORMATIONS IN C++ USING OPENGL

21/08/2021

Venkataraman Nagarajan, CSE - C
18500192

AIM

To implement 2d-Transformations in C++.

SPECIFICATION

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

- 1) Translation
- 2) Rotation
 - a) about origin
 - b) with respect to a fixed point (x_r, y_r)
- 3) Scaling with respect to
 - a) origin - Uniform Vs Differential Scaling
 - b) fixed point (x_f, y_f) - Uniform Vs Differential Scaling
- 4) Reflection with respect to
 - a) x-axis
 - b) y-axis
 - c) origin
 - d) the line $x = y$
- 5) Shearing
 - a) x-direction shear
 - b) y-direction shear

Note: Use Homogeneous coordinate representations and matrix multiplication to perform transformations. Divide the output window into four quadrants. (Use *LINES* primitive to draw x and y axis.

PROGRAM - 01

Traslation

```
1 // To apply the following 2D transformations on objects and to render the ↵
   final output along with the original object.
2
3 // 1) Translation
4
5 #include<bits/stdc++.h>
6 #include<GL/glut.h>
7
8 using namespace std;
9 using ld = long double;
10 using ll = long long;
11
12 #define X      first
13 #define Y      second
14
15 const int WINDOW_WIDTH = 900;
16 const int WINDOW_HEIGHT = 900;
17
18 const int X_MIN = -300;
19 const int X_MAX = 300;
20 const int Y_MIN = -300;
21 const int Y_MAX = 300;
22
23 void myInit();
24 void myDisplay();
25
26 void printAxes();
27 ll multiply(vector<ll> a, vector<ll> b);
28 vector<ll> multiply(vector<vector<ll>> a, vector<ll> b);
29
30 vector<ll> getHomogeneousPointCoords(pair<ll,ll> point, ll h=1);
31 vector<ll> translate(vector<ll> &point_matrix, ll tx=0, ll ty=0);
32 pair<ll,ll> getPoint(vector<ll> point_matrix);
33 void translateShape();
34
35 const ld PADDING = 0;
36 const ld STEP = 10;
37 const ld SCALE = 1;
38
39 int main(int argc, char* argv[]) {
40     glutInit(&argc,argv);
41     glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
42     glutInitWindowSize(WINDOW_WIDTH,WINDOW_HEIGHT);
```

```

43     glutCreateWindow("2D - Translation");
44     glutDisplayFunc(myDisplay);
45     myInit();
46     glutMainLoop();
47     return 1;
48 }
49
50 void myInit() {
51     glClearColor(1.0,1.0,1.0,0.0);
52     glColor3f(0.0f,0.0f,0.0f);
53     glPointSize(2.0);
54     glMatrixMode(GL_PROJECTION);
55     glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
56     glEnable( GL_BLEND );
57     glLoadIdentity();
58     gluOrtho2D(X_MIN,X_MAX,Y_MIN,Y_MAX);
59 }
60
61 void myDisplay() {
62     glClear(GL_COLOR_BUFFER_BIT);
63
64     printAxes();
65     translateShape();
66
67     glFlush();
68 }
69
70 void printAxes() {
71     glBegin(GL_LINES);
72
73     glColor3f(1.0f,0.0f,0.0f);
74     glVertex2d(X_MIN,0);
75     glVertex2d(X_MAX,0);
76
77     glColor3f(1.0f,0.0f,0.0f);
78     glVertex2d(0,Y_MIN);
79     glVertex2d(0,Y_MAX);
80
81     for(11 i=X_MIN;i<X_MAX;i+=STEP) {
82         glVertex2d(i,-0.3*STEP);
83         glVertex2d(i,0.3*STEP);
84     }
85
86     for(11 i=Y_MIN;i<Y_MAX;i+=STEP) {
87         glVertex2d(-0.3*STEP,i);
88         glVertex2d(0.3*STEP,i);
89     }

```

```

90
91     glEnd();
92 }
93
94 void translateShape() {
95     //Plot original shape;
96
97     vector<pair<ll,ll>> shape = {{20,20}, {20,50}, {50,50}, {50,20}};
98
99     glBegin(GL_POLYGON);
100    glColor4f(0.7f,0.0f,1.0f,1.0f);
101
102    for(auto point : shape) {
103        glVertex2d(point.X,point.Y);
104    }
105
106    glEnd();
107
108    //Translate shape;
109
110    glBegin(GL_POLYGON);
111    glColor4f(0.7f,0.0f,1.0f,0.4f);
112
113    for(auto point : shape) {
114        vector<ll> point_matrix = getHomogeneousPointCoords(point);
115        pair<ll,ll> translated_point = getPoint(translate(point_matrix↵
116            ,50,50));
117
118        glVertex2d(translated_point.X,translated_point.Y);
119    }
120
121    glEnd();
122 }
123
124 pair<ll,ll> getPoint(vector<ll> point_matrix) {
125     ll h = point_matrix[2];
126     ll x = point_matrix[0];
127     ll y = point_matrix[1];
128
129     return {x/h,y/h};
130 }
131
132 vector<ll> getHomogeneousPointCoords(pair<ll,ll> point, ll h) {
133     vector<ll> point_matrix;
134     point_matrix.push_back(h*point.first);
135     point_matrix.push_back(h*point.second);

```

```

136     point_matrix.push_back(h);
137     return point_matrix;
138 }
139
140 vector<ll> translate(vector<ll> &point_matrix, ll tx, ll ty) {
141     vector<vector<ll>> translate_matrix = {
142                                     {1,0,tx},
143                                     {0,1,ty},
144                                     {0,0,1}
145                                     };
146     return multiply(translate_matrix, point_matrix);
147 }
148
149 vector<ll> multiply(vector<vector<ll>> a, vector<ll> b) {
150     vector<ll> result;
151     for(int i=0;i<a.size();i++) {
152         ll temp = multiply(a[i],b);
153         result.push_back(temp);
154     }
155     return result;
156 }
157
158 ll multiply(vector<ll> a, vector<ll> b) {
159     ll result=0;
160     for(int i=0;i<a.size();i++) {
161         result+=(a[i]*b[i]);
162     }
163     return result;
164 }

```

SAMPLE I/O

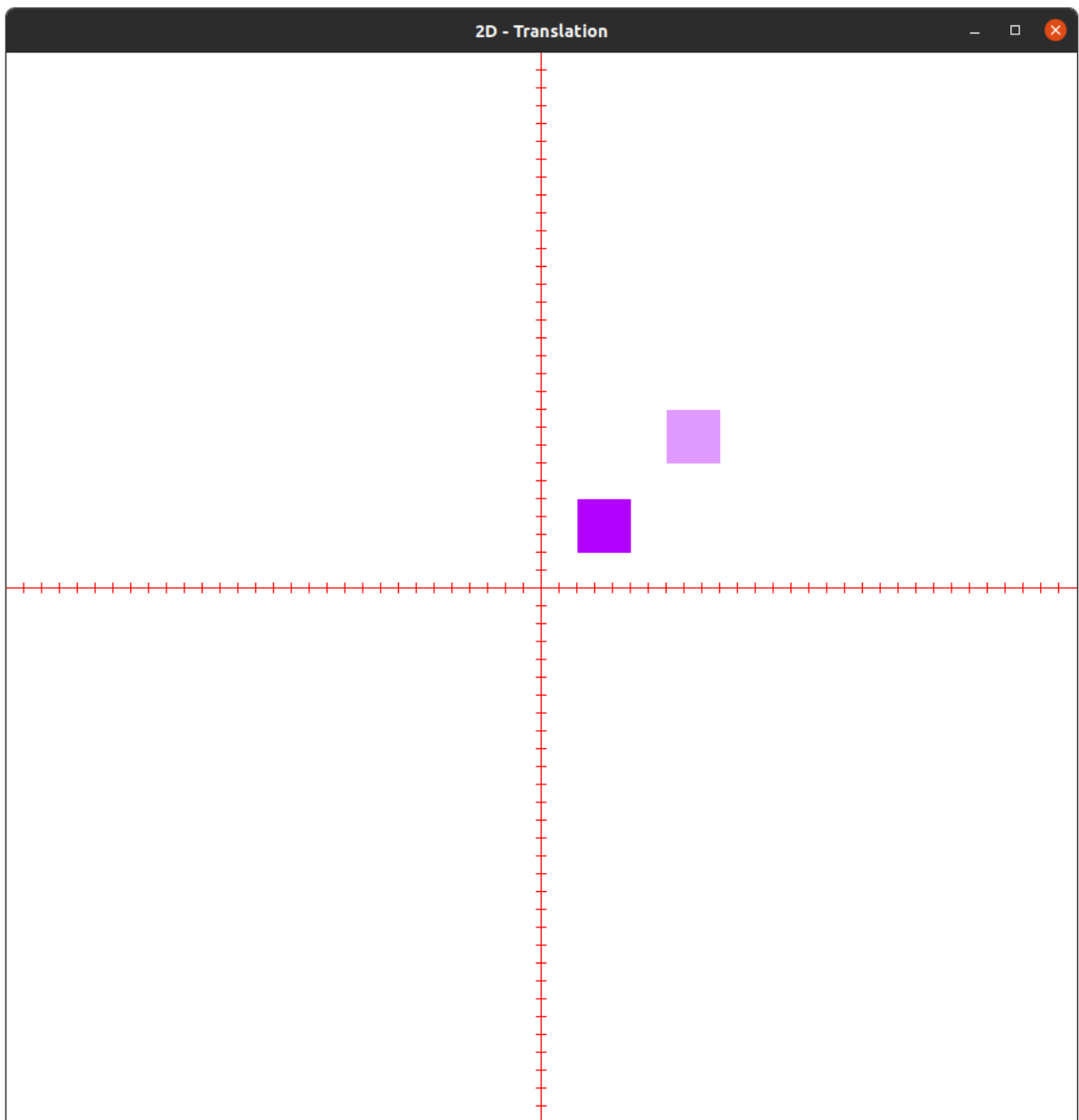


Figure 1: The solid square is translated to the translucent one

PROGRAM - 02

Rotation

```
1 // To apply the following 2D transformations on objects and to render the ↵
   final output along with the original object.
2
3 // 2) Rotation
4 //     a) about origin
5 //     b) with respect to a fixed point (xr,yr)
6
7 #include<bits/stdc++.h>
8 #include<GL/glut.h>
9
10 using namespace std;
11 using ld = long double;
12 using ll = long long;
13
14 #define X      first
15 #define Y      second
16
17 const int WINDOW_WIDTH = 900;
18 const int WINDOW_HEIGHT = 900;
19
20 const int X_MIN = -300;
21 const int X_MAX = 300;
22 const int Y_MIN = -300;
23 const int Y_MAX = 300;
24
25 const ld PADDING = 0;
26 const ld STEP = 10;
27 const ld SCALE = 1;
28 const ld PI = 3.14159265358979323846264338327950288419716939937510582;
29
30 void myInit();
31 void myDisplay();
32
33 void printAxes();
34 ld getRadian(ld degree);
35 ld multiply(vector<ld> a, vector<ll> b);
36 vector<ld> multiply(vector<vector<ld>> a, vector<ll> b);
37
38 vector<ll> getHomogeneousPointCoords(pair<ll,ll> point, ll h=1);
39 vector<ld> rotate(vector<ll> &point_matrix, ld angle=0, pair<ll,ll> pivot=↵
   make_pair(0,0));
40 pair<ld,ld> getPoint(vector<ld> point_matrix);
41 void rotateShape();
```

```

42
43
44 int main(int argc, char* argv[]) {
45     glutInit(&argc, argv);
46     glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
47     glutInitWindowSize(WINDOW_WIDTH, WINDOW_HEIGHT);
48     glutCreateWindow("2D - Rotation");
49     glutDisplayFunc(myDisplay);
50     myInit();
51     glutMainLoop();
52     return 1;
53 }
54
55 void myInit() {
56     glClearColor(1.0, 1.0, 1.0, 0.0);
57     glColor3f(0.0f, 0.0f, 0.0f);
58     glPointSize(5.0);
59     glMatrixMode(GL_PROJECTION);
60     glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
61     glEnable( GL_BLEND );
62     glLoadIdentity();
63     gluOrtho2D(X_MIN, X_MAX, Y_MIN, Y_MAX);
64 }
65
66 void myDisplay() {
67     glClear(GL_COLOR_BUFFER_BIT);
68
69     printAxes();
70     rotateShape();
71
72     glFlush();
73 }
74
75 void printAxes() {
76     glBegin(GL_LINES);
77
78     glColor3f(1.0f, 0.0f, 0.0f);
79     glVertex2d(X_MIN, 0);
80     glVertex2d(X_MAX, 0);
81
82     glColor3f(1.0f, 0.0f, 0.0f);
83     glVertex2d(0, Y_MIN);
84     glVertex2d(0, Y_MAX);
85
86     for (ll i=X_MIN; i<X_MAX; i+=STEP) {
87         glVertex2d(i, -0.3*STEP);
88         glVertex2d(i, 0.3*STEP);

```



```

89     }
90
91     for(int i=Y_MIN;i<Y_MAX;i+=STEP) {
92         glVertex2d(-0.3*STEP,i);
93         glVertex2d(0.3*STEP,i);
94     }
95
96     glEnd();
97 }
98
99 void rotateShape() {
100     //Plot original shape;
101
102     vector<pair<int,int>> shape = {{20,20}, {60,20}, {40,60}};
103
104     glBegin(GL_POLYGON);
105     glColor4f(0.7f,0.0f,1.0f,1.0f);
106
107     for(auto point : shape) {
108         glVertex2d(point.X,point.Y);
109     }
110
111     glEnd();
112
113     //Rotate shape with respect to origin;
114
115     glBegin(GL_POLYGON);
116     glColor4f(0.7f,0.0f,1.0f,0.4f);
117
118     for(auto point : shape) {
119         vector<int> point_matrix = getHomogeneousPointCoords(point);
120         pair<int,int> rotated_point = getPoint(rotate(point_matrix,90));
121
122         glVertex2d(rotated_point.X,rotated_point.Y);
123     }
124
125     glEnd();
126
127     //Rotate shape with respect to a fixed point;
128
129     pair<int,int> pivot = {40,70};
130
131     glBegin(GL_POINTS);
132     glColor4f(0.8f,0.0f,0.5f,0.9f);
133     glVertex2d(pivot.X, pivot.Y);
134     glEnd();
135

```

```

136     glBegin(GL_POLYGON);
137     glColor4f(0.7f,0.0f,1.0f,0.4f);
138
139     for(auto point : shape) {
140         vector<ll> point_matrix = getHomogeneousPointCoords(point);
141         pair<ld,ld> rotated_point = getPoint(rotate(point_matrix,180,pivot↵
            ));
142
143         glVertex2d(rotated_point.X,rotated_point.Y);
144     }
145
146     glEnd();
147 }
148
149 pair<ld,ld> getPoint(vector<ld> point_matrix) {
150     ll h = point_matrix[2];
151     ld x = point_matrix[0];
152     ld y = point_matrix[1];
153
154     return {x/h,y/h};
155 }
156
157 ld getRadian(ld degree) {
158     return degree*PI/180;
159 }
160
161 vector<ll> getHomogeneousPointCoords(pair<ll,ll> point, ll h) {
162     vector<ll> point_matrix;
163     point_matrix.push_back(h*point.first);
164     point_matrix.push_back(h*point.second);
165     point_matrix.push_back(h);
166     return point_matrix;
167 }
168
169 vector<ld> rotate(vector<ll> &point_matrix, ld angle, pair<ll,ll> pivot) {
170     angle = getRadian(angle);
171     ll xr = pivot.X;
172     ll yr = pivot.Y;
173
174     vector<vector<ld>> rotate_matrix = {
175         {cos(angle), -sin(angle), xr*(1-cos(angle)↵
            + yr*sin(angle))},
176         {sin(angle), cos(angle), yr*(1-cos(angle)↵
            - xr*sin(angle))},
177         {0, 0, 1}
178     };
179     return multiply(rotate_matrix, point_matrix);

```

```
180 }
181
182 vector<ld> multiply(vector<vector<ld>> a, vector<ll> b) {
183     vector<ld> result;
184     for(int i=0;i<a.size();i++) {
185         ll temp = multiply(a[i],b);
186         result.push_back(temp);
187     }
188     return result;
189 }
190
191 ld multiply(vector<ld> a, vector<ll> b) {
192     ld result=0;
193     for(int i=0;i<a.size();i++) {
194         result+=(a[i]*b[i]);
195     }
196     return result;
197 }
```

SAMPLE I/O

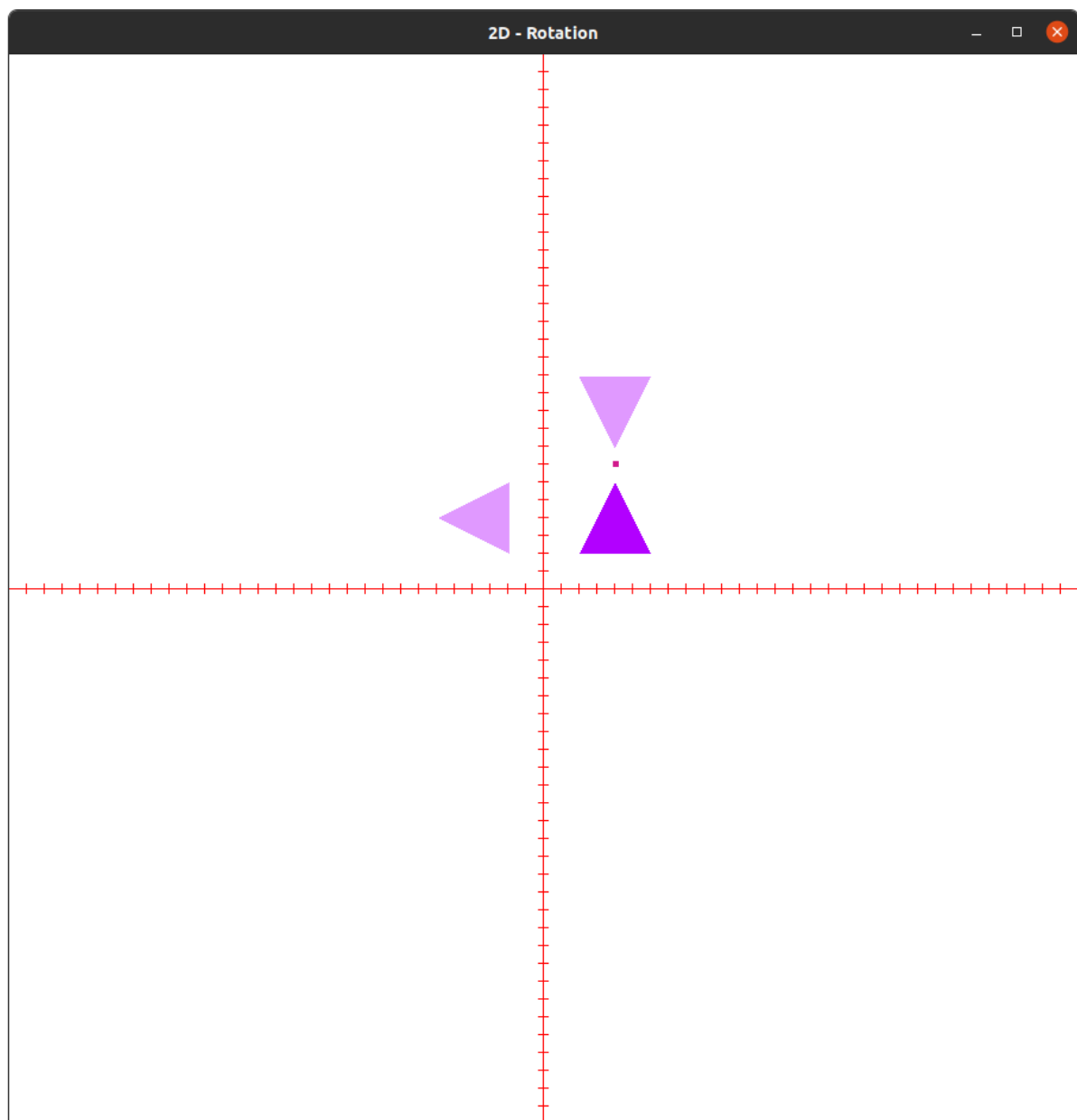


Figure 2: (a) The solid triangle is rotated 90° w.r.t origin to obtain triangle at 2^{nd} quadrant.

(b) The solid triangle is rotated 180° with respect to the point shown to obtain the triangle above it.

PROGRAM - 03

Scaling

```
1 // To apply the following 2D transformations on objects and to render the ↵
   final output along with the original object.
2
3 // 3) Scaling with respect to
4 //     a) origin - Uniform Vs Differential Scaling
5 //     b) fixed point (xf,yf)
6
7
8 #include<bits/stdc++.h>
9 #include<GL/glut.h>
10
11 using namespace std;
12 using ld = long double;
13 using ll = long long;
14
15 #define X      first
16 #define Y      second
17
18 const int WINDOW_WIDTH = 900;
19 const int WINDOW_HEIGHT = 900;
20
21 const int X_MIN = -300;
22 const int X_MAX = 300;
23 const int Y_MIN = -300;
24 const int Y_MAX = 300;
25
26 const ld PADDING = 0;
27 const ld STEP = 10;
28 const ld SCALE = 1;
29
30 void myInit();
31 void myDisplay();
32
33 void printAxes();
34 ld multiply(vector<ld> a, vector<ll> b);
35 vector<ld> multiply(vector<vector<ld>> a, vector<ll> b);
36
37 vector<ll> getHomogeneousPointCoords(pair<ll,ll> point, ll h=1);
38 vector<ld> scale(vector<ll> &point_matrix, ld sx=1, ld sy=2, pair<ll,ll> ↵
   pivot=make_pair(0,0));
39 pair<ld,ld> getPoint(vector<ld> point_matrix);
40 void scaleShape();
41
```

```

42
43 int main(int argc, char* argv[]) {
44     glutInit(&argc, argv);
45     glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
46     glutInitWindowSize(WINDOW_WIDTH, WINDOW_HEIGHT);
47     glutCreateWindow("2D - Scaling");
48     glutDisplayFunc(myDisplay);
49     myInit();
50     glutMainLoop();
51     return 1;
52 }
53
54 void myInit() {
55     glClearColor(1.0, 1.0, 1.0, 0.0);
56     glColor3f(0.0f, 0.0f, 0.0f);
57     glPointSize(5.0);
58     glMatrixMode(GL_PROJECTION);
59     glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
60     glEnable( GL_BLEND );
61     glLoadIdentity();
62     gluOrtho2D(X_MIN, X_MAX, Y_MIN, Y_MAX);
63 }
64
65 void myDisplay() {
66     glClear(GL_COLOR_BUFFER_BIT);
67
68     printAxes();
69     scaleShape();
70
71     glFlush();
72 }
73
74 void printAxes() {
75     glBegin(GL_LINES);
76
77     glColor3f(1.0f, 0.0f, 0.0f);
78     glVertex2d(X_MIN, 0);
79     glVertex2d(X_MAX, 0);
80
81     glColor3f(1.0f, 0.0f, 0.0f);
82     glVertex2d(0, Y_MIN);
83     glVertex2d(0, Y_MAX);
84
85     for (ll i=X_MIN; i<X_MAX; i+=STEP) {
86         glVertex2d(i, -0.3*STEP);
87         glVertex2d(i, 0.3*STEP);
88     }

```

```

89
90     for(ll i=Y_MIN;i<Y_MAX;i+=STEP) {
91         glVertex2d(-0.3*STEP,i);
92         glVertex2d(0.3*STEP,i);
93     }
94
95     glEnd();
96 }
97
98 void scaleShape() {
99
100     vector<pair<ll,ll>> shape;
101
102     //Plot original shape;
103
104     shape = {{20,40}, {60,40}, {40,80}};
105
106     glBegin(GL_POLYGON);
107     glColor4f(0.7f,0.0f,1.0f,1.0f);
108
109     for(auto point : shape) {
110         glVertex2d(point.X,point.Y);
111     }
112
113     glEnd();
114
115     //Scale shape w.r.t. origin - Uniform Scaling;
116
117     glBegin(GL_POLYGON);
118     glColor4f(0.7f,0.0f,1.0f,0.4f);
119
120     for(auto point : shape) {
121         vector<ll> point_matrix = getHomogeneousPointCoords(point);
122         pair<ld,ld> scaled_point = getPoint(scale(point_matrix, 2, 2));
123
124         glVertex2d(scaled_point.X,scaled_point.Y);
125     }
126
127     glEnd();
128
129
130
131
132     //Plot original shape;
133
134     shape = {{-20,40}, {-60,40}, {-40,80}};
135

```

```

136     glBegin(GL_POLYGON);
137     glColor4f(0.0f,1.0f,0.7f,1.0f);
138
139     for(auto point : shape) {
140         glVertex2d(point.X,point.Y);
141     }
142
143     glEnd();
144
145     //Scale shape w.r.t. origin - Differential Scaling;
146
147     glBegin(GL_POLYGON);
148     glColor4f(0.0f,1.0f,0.7f,0.4f);
149
150     for(auto point : shape) {
151         vector<ll> point_matrix = getHomogeneousPointCoords(point);
152         pair<ld,ld> scaled_point = getPoint(scale(point_matrix, 0.5, 2));
153
154         glVertex2d(scaled_point.X,scaled_point.Y);
155     }
156
157     glEnd();
158
159
160
161
162     //Plot original shape;
163
164     shape = {{20,-40}, {60,-40}, {40,-80}};
165
166     glBegin(GL_POLYGON);
167     glColor4f(0.7f,1.0f,0.0f,0.7f);
168
169     for(auto point : shape) {
170         glVertex2d(point.X,point.Y);
171     }
172
173     glEnd();
174
175     //Scale shape w.r.t. fixed point - Uniform Scaling;
176
177     pair<ll,ll> pivot = {40, -60};
178
179     glBegin(GL_POINTS);
180     glColor4f(0.0f,0.0f,1.0f,0.9f);
181     glVertex2d(pivot.X, pivot.Y);
182     glEnd();

```



```

183
184     glBegin(GL_POLYGON);
185     glColor4f(0.7f,1.0f,0.0f,0.6f);
186
187     for(auto point : shape) {
188         vector<ll> point_matrix = getHomogeneousPointCoords(point);
189         pair<ld,ld> scaled_point = getPoint(scale(point_matrix, 1.5, 1.5, ←
            pivot));
190
191         glVertex2d(scaled_point.X,scaled_point.Y);
192     }
193
194     glEnd();
195
196
197
198
199     //Plot original shape;
200
201     shape = {{-20,-40}, {-60,-40}, {-40,-80}};
202
203     glBegin(GL_POLYGON);
204     glColor4f(1.0f,0.7f,0.0f,0.7f);
205
206     for(auto point : shape) {
207         glVertex2d(point.X,point.Y);
208     }
209
210     glEnd();
211
212     //Scale shape w.r.t. fixed point - Differential Scaling;
213
214     pivot = {-40, -60};
215
216     glBegin(GL_POINTS);
217     glColor4f(1.0f,0.0f,0.0f,0.9f);
218     glVertex2d(pivot.X, pivot.Y);
219     glEnd();
220
221     glBegin(GL_POLYGON);
222     glColor4f(1.0f,0.7f,0.0f,0.5f);
223
224     for(auto point : shape) {
225         vector<ll> point_matrix = getHomogeneousPointCoords(point);
226         pair<ld,ld> scaled_point = getPoint(scale(point_matrix, 0.5, 1.5, ←
            pivot));
227

```

```

228         glVertex2d(scaled_point.X,scaled_point.Y);
229     }
230
231     glEnd();
232
233 }
234
235 pair<ld,ld> getPoint(vector<ld> point_matrix) {
236     ll h = point_matrix[2];
237     ld x = point_matrix[0];
238     ld y = point_matrix[1];
239
240     return {x/h,y/h};
241 }
242
243 vector<ll> getHomogeneousPointCoords(pair<ll,ll> point, ll h) {
244     vector<ll> point_matrix;
245     point_matrix.push_back(h*point.first);
246     point_matrix.push_back(h*point.second);
247     point_matrix.push_back(h);
248     return point_matrix;
249 }
250
251 vector<ld> scale(vector<ll> &point_matrix, ld sx, ld sy, pair<ll,ll> pivot←
    ) {
252
253     ll xf = pivot.X;
254     ll yf = pivot.Y;
255
256     vector<vector<ld>> scale_matrix = {
257         {sx, 0, xf*(1-sx)},
258         {0, sy, yf*(1-sy)},
259         {0, 0, 1}
260     };
261     return multiply(scale_matrix,point_matrix);
262 }
263
264 vector<ld> multiply(vector<vector<ld>> a, vector<ll> b) {
265     vector<ld> result;
266     for(int i=0;i<a.size();i++) {
267         ll temp = multiply(a[i],b);
268         result.push_back(temp);
269     }
270     return result;
271 }
272
273 ld multiply(vector<ld> a, vector<ll> b) {

```

```
274     ld result=0;
275     for(int i=0;i<a.size();i++) {
276         result+=(a[i]*b[i]);
277     }
278     return result;
279 }
```

SAMPLE I/O

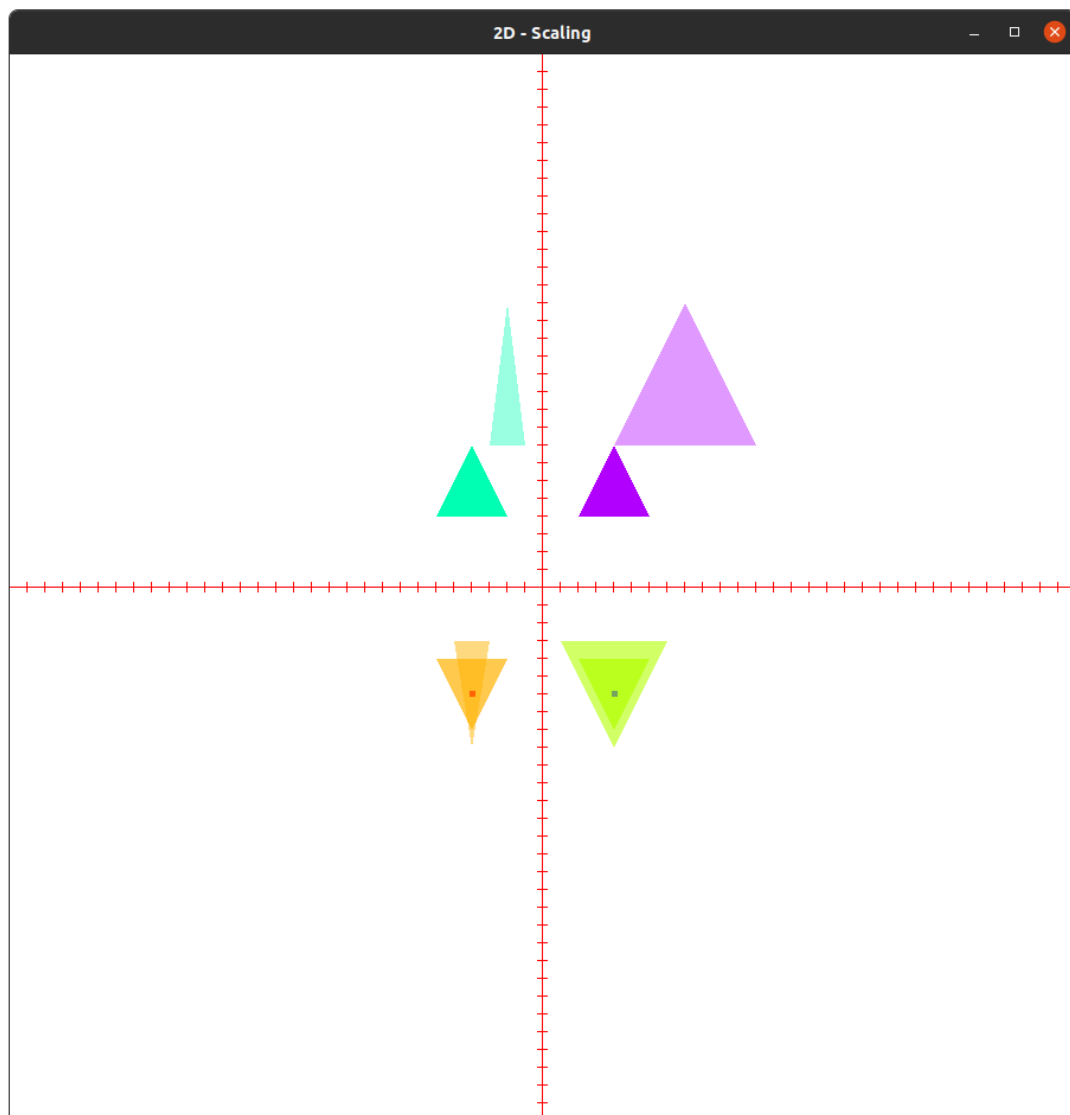


Figure 3: (a) The solid triangle in 1^{st} quadrant is scaled uniformly w.r.t origin to its translucent version.

(b) The solid triangle in 2^{nd} quadrant is scaled differentially w.r.t origin to its translucent version.

(c) The solid triangle in 3^{rd} quadrant is scaled uniformly w.r.t a fixed point shown to its translucent version.

(d) The solid triangle in 4^{th} quadrant is scaled differentially w.r.t fixed point shown to its translucent version.

PROGRAM - 04

Reflection

```
1 // To apply the following 2D transformations on objects and to render the ↵
   final output along with the original object.
2
3 // 4) Reflection with respect to
4 //     a) x-axis
5 //     b) y-axis
6 //     c) origin
7 //     d) the line x=y
8
9 #include<bits/stdc++.h>
10 #include<GL/glut.h>
11
12 using namespace std;
13 using ld = long double;
14 using ll = long long;
15
16 #define X      first
17 #define Y      second
18
19 const int WINDOW_WIDTH = 900;
20 const int WINDOW_HEIGHT = 900;
21
22 const int X_MIN = -300;
23 const int X_MAX = 300;
24 const int Y_MIN = -300;
25 const int Y_MAX = 300;
26
27 const ld PADDING = 0;
28 const ld STEP = 10;
29 const ld SCALE = 1;
30
31 void myInit();
32 void myDisplay();
33
34 void printAxes();
35 ld multiply(vector<ld> a, vector<ll> b);
36 vector<ld> multiply(vector<vector<ld>> a, vector<ll> b);
37
38 enum ReflectionType {X_AXIS, Y_AXIS, ORIGIN, X_EQUALS_Y_LINE};
39
40 vector<ll> getHomogeneousPointCoords(pair<ll,ll> point, ll h=1);
41 vector<ld> reflect(vector<ll> &point_matrix, ReflectionType type=ORIGIN);
42 pair<ld,ld> getPoint(vector<ld> point_matrix);
```

```

43 void reflectShape();
44
45
46 int main(int argc, char* argv[]) {
47     glutInit(&argc, argv);
48     glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
49     glutInitWindowSize(WINDOW_WIDTH, WINDOW_HEIGHT);
50     glutCreateWindow("2D - Reflection");
51     glutDisplayFunc(myDisplay);
52     myInit();
53     glutMainLoop();
54     return 1;
55 }
56
57 void myInit() {
58     glClearColor(1.0, 1.0, 1.0, 0.0);
59     glColor3f(0.0f, 0.0f, 0.0f);
60     glPointSize(2.0);
61     glMatrixMode(GL_PROJECTION);
62     glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
63     glEnable( GL_BLEND );
64     glLoadIdentity();
65     gluOrtho2D(X_MIN, X_MAX, Y_MIN, Y_MAX);
66 }
67
68 void myDisplay() {
69     glClear(GL_COLOR_BUFFER_BIT);
70
71     printAxes();
72     reflectShape();
73
74     glFlush();
75 }
76
77 void printAxes() {
78     glBegin(GL_LINES);
79
80     glColor3f(1.0f, 0.0f, 0.0f);
81     glVertex2d(X_MIN, 0);
82     glVertex2d(X_MAX, 0);
83
84     glColor3f(1.0f, 0.0f, 0.0f);
85     glVertex2d(0, Y_MIN);
86     glVertex2d(0, Y_MAX);
87
88     for (ll i=X_MIN; i<X_MAX; i+=STEP) {
89         glVertex2d(i, -0.3*STEP);

```

```

90         glVertex2d(i,0.3*STEP);
91     }
92
93     for(ll i=Y_MIN;i<Y_MAX;i+=STEP) {
94         glVertex2d(-0.3*STEP,i);
95         glVertex2d(0.3*STEP,i);
96     }
97
98     glEnd();
99 }
100
101 void reflectShape() {
102
103     vector<pair<ll,ll>> shape;
104
105     //Plot original shape;
106
107     shape = {{20,100}, {40,80}, {60,100}, {40,120}};
108
109     glBegin(GL_POLYGON);
110     glColor4f(0.0f,1.0f,0.7f,1.0f);
111
112     for(auto point : shape) {
113         glVertex2d(point.X,point.Y);
114     }
115
116     glEnd();
117
118     //Reflect shape along x-axis;
119
120     glBegin(GL_POLYGON);
121     glColor4f(0.0f,0.1f,0.7f,0.4f);
122
123     for(auto point : shape) {
124         vector<ll> point_matrix = getHomogeneousPointCoords(point);
125         pair<ld,ld> reflected_point = getPoint(reflect(point_matrix, ←
            X_AXIS));
126
127         glVertex2d(reflected_point.X,reflected_point.Y);
128     }
129
130     glEnd();
131
132     //Reflect shape along y-axis;
133
134     glBegin(GL_POLYGON);
135     glColor4f(0.0f,0.1f,0.7f,0.4f);

```

```

136
137     for(auto point : shape) {
138         vector<ll> point_matrix = getHomogeneousPointCoords(point);
139         pair<ld,ld> reflected_point = getPoint(reflect(point_matrix, ↵
            Y_AXIS));
140
141         glVertex2d(reflected_point.X,reflected_point.Y);
142     }
143
144     glEnd();
145
146     //Reflect shape along origin;
147
148     glBegin(GL_POLYGON);
149     glColor4f(0.0f,0.1f,0.7f,0.4f);
150
151     for(auto point : shape) {
152         vector<ll> point_matrix = getHomogeneousPointCoords(point);
153         pair<ld,ld> reflected_point = getPoint(reflect(point_matrix, ↵
            ORIGIN));
154
155         glVertex2d(reflected_point.X,reflected_point.Y);
156     }
157
158     glEnd();
159
160     //Reflect shape along x=y line;
161
162     glBegin(GL_LINES);
163     glColor4f(0.0f,1.0f,0.0f,0.4f);
164
165     glVertex2d(min(X_MIN,Y_MIN),min(X_MIN,Y_MIN));
166     glVertex2d(max(X_MAX,Y_MAX),max(X_MAX,Y_MAX));
167
168     glEnd();
169
170     glBegin(GL_POLYGON);
171     glColor4f(0.0f,0.1f,0.7f,0.4f);
172
173     for(auto point : shape) {
174         vector<ll> point_matrix = getHomogeneousPointCoords(point);
175         pair<ld,ld> reflected_point = getPoint(reflect(point_matrix, ↵
            X_EQUALS_Y_LINE));
176
177         glVertex2d(reflected_point.X,reflected_point.Y);
178     }
179

```



```

180     glEnd();
181 }
182
183 pair<ld,ld> getPoint(vector<ld> point_matrix) {
184     ll h = point_matrix[2];
185     ld x = point_matrix[0];
186     ld y = point_matrix[1];
187
188     return {x/h,y/h};
189 }
190
191 vector<ll> getHomogeneousPointCoords(pair<ll,ll> point, ll h) {
192     vector<ll> point_matrix;
193     point_matrix.push_back(h*point.first);
194     point_matrix.push_back(h*point.second);
195     point_matrix.push_back(h);
196     return point_matrix;
197 }
198
199 vector<ld> reflect(vector<ll> &point_matrix, ReflectionType type) {
200
201     vector<vector<ld>> reflection_matrix;
202
203     switch (type)
204     {
205         case X_AXIS:
206             reflection_matrix = {
207                 {1,0,0},
208                 {0,-1,0},
209                 {0,0,1}
210             };
211             break;
212
213         case Y_AXIS:
214             reflection_matrix = {
215                 {-1,0,0},
216                 {0,1,0},
217                 {0,0,1}
218             };
219             break;
220
221         case ORIGIN:
222             reflection_matrix = {
223                 {-1,0,0},
224                 {0,-1,0},
225                 {0,0,1}
226             };

```

```

227         break;
228
229     case X_EQUALS_Y_LINE:
230         reflection_matrix = {
231             {0,1,0},
232             {1,0,0},
233             {0,0,1}
234         };
235         break;
236
237     default:
238         reflection_matrix = {
239             {1,0,0},
240             {0,1,0},
241             {0,0,1}
242         };
243         break;
244     }
245
246     return multiply(reflection_matrix, point_matrix);
247 }
248
249 vector<ld> multiply(vector<vector<ld>> a, vector<ll> b) {
250     vector<ld> result;
251     for(int i=0;i<a.size();i++) {
252         ll temp = multiply(a[i],b);
253         result.push_back(temp);
254     }
255     return result;
256 }
257
258 ld multiply(vector<ld> a, vector<ll> b) {
259     ld result=0;
260     for(int i=0;i<a.size();i++) {
261         result+=(a[i]*b[i]);
262     }
263     return result;
264 }

```

SAMPLE I/O

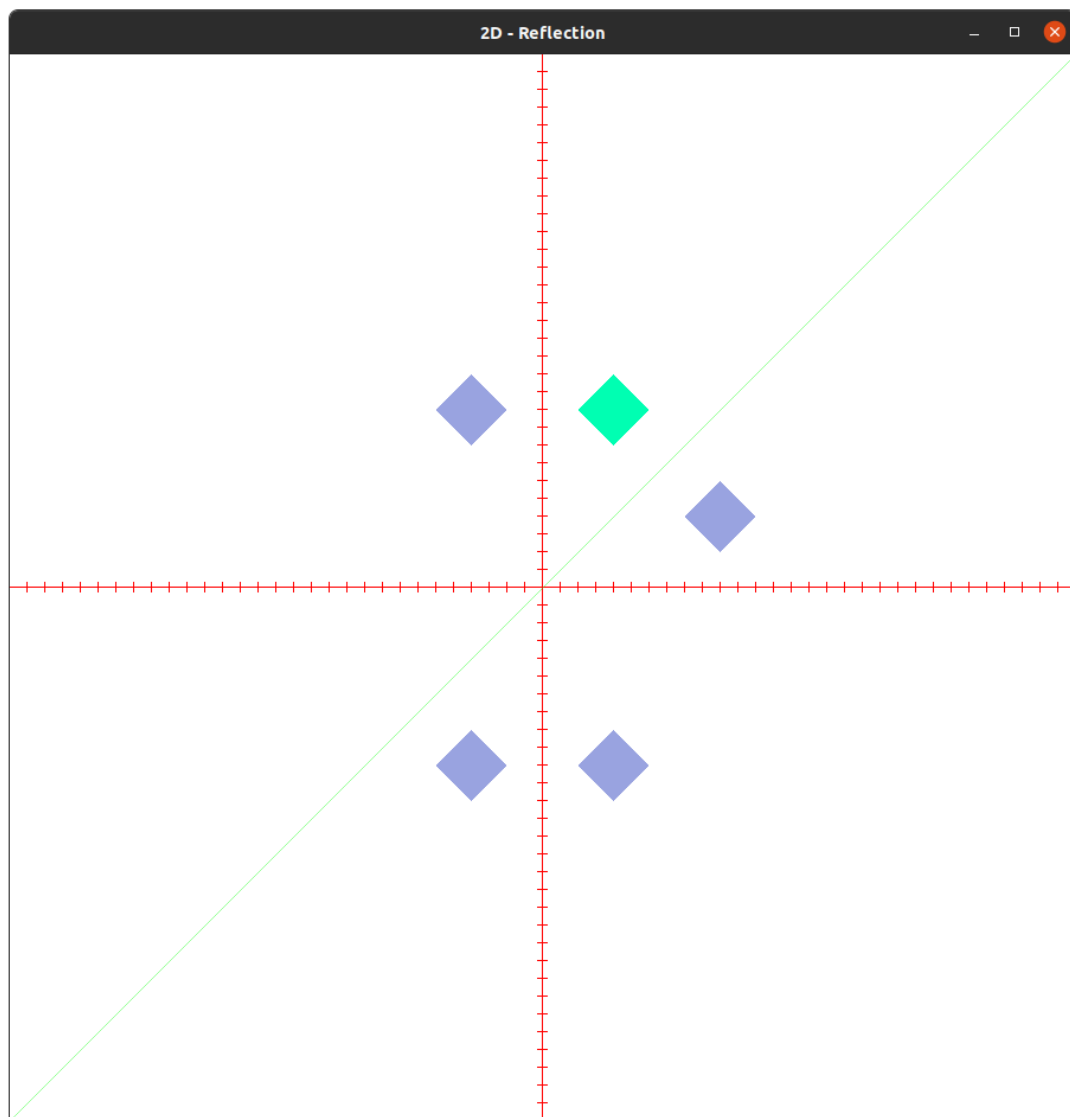


Figure 4: (a) The green diamond is reflected w.r.t x -axis to the blue one in 4th quadrant.
(b) The green diamond is reflected w.r.t y -axis to the blue one in 2nd quadrant.
(c) The green diamond is reflected w.r.t origin to the blue one in 3rd quadrant.
(d) The green diamond is reflected w.r.t green line $x = y$ to the blue one in 1st quadrant.

PROGRAM - 05

Shearing

```
1 // To apply the following 2D transformations on objects and to render the ↵
   final output along with the original object.
2
3 // 5) Shearing
4 //     a) x-direction shear
5 //     b) y-direction shear
6
7 #include<bits/stdc++.h>
8 #include<GL/glut.h>
9
10 using namespace std;
11 using ld = long double;
12 using ll = long long;
13
14 #define X      first
15 #define Y      second
16
17 const int WINDOW_WIDTH = 900;
18 const int WINDOW_HEIGHT = 900;
19
20 const int X_MIN = -300;
21 const int X_MAX = 300;
22 const int Y_MIN = -300;
23 const int Y_MAX = 300;
24
25 const ld PADDING = 0;
26 const ld STEP = 10;
27 const ld SCALE = 1;
28
29 void myInit();
30 void myDisplay();
31
32 void printAxes();
33 ld multiply(vector<ld> a, vector<ll> b);
34 vector<ld> multiply(vector<vector<ld>> a, vector<ll> b);
35
36 enum ShearType {X_SHEAR, Y_SHEAR};
37
38 vector<ll> getHomogeneousPointCoords(pair<ll,ll> point, ll h=1);
39 vector<ld> shear(vector<ll> &point_matrix, ShearType type=X_SHEAR, ld sh↵
   =1, ld refLine=0);
40 pair<ld,ld> getPoint(vector<ld> point_matrix);
41 void shearShape();
```

```

42
43
44 int main(int argc, char* argv[]) {
45     glutInit(&argc, argv);
46     glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
47     glutInitWindowSize(WINDOW_WIDTH, WINDOW_HEIGHT);
48     glutCreateWindow("2D - Shearing");
49     glutDisplayFunc(myDisplay);
50     myInit();
51     glutMainLoop();
52     return 1;
53 }
54
55 void myInit() {
56     glClearColor(1.0, 1.0, 1.0, 0.0);
57     glColor3f(0.0f, 0.0f, 0.0f);
58     glPointSize(2.0);
59     glMatrixMode(GL_PROJECTION);
60     glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
61     glEnable( GL_BLEND );
62     glLoadIdentity();
63     gluOrtho2D(X_MIN, X_MAX, Y_MIN, Y_MAX);
64 }
65
66 void myDisplay() {
67     glClear(GL_COLOR_BUFFER_BIT);
68
69     printAxes();
70     shearShape();
71
72     glFlush();
73 }
74
75 void printAxes() {
76     glBegin(GL_LINES);
77
78     glColor3f(1.0f, 0.0f, 0.0f);
79     glVertex2d(X_MIN, 0);
80     glVertex2d(X_MAX, 0);
81
82     glColor3f(1.0f, 0.0f, 0.0f);
83     glVertex2d(0, Y_MIN);
84     glVertex2d(0, Y_MAX);
85
86     for (ll i=X_MIN; i<X_MAX; i+=STEP) {
87         glVertex2d(i, -0.3*STEP);
88         glVertex2d(i, 0.3*STEP);

```

```

89     }
90
91     for(ll i=Y_MIN;i<Y_MAX;i+=STEP) {
92         glVertex2d(-0.3*STEP,i);
93         glVertex2d(0.3*STEP,i);
94     }
95
96     glEnd();
97 }
98
99 void shearShape() {
100
101     vector<pair<ll,ll>> shape;
102
103     //QUAD - 1
104
105     //Plot original shape;
106
107     shape = {{0,0}, {40,0}, {40,40}, {0,40}};
108
109     glBegin(GL_POLYGON);
110     glColor4f(0.7f,0.5f,1.0f,0.6f);
111
112     for(auto point : shape) {
113         glVertex2d(point.X,point.Y);
114     }
115
116     glEnd();
117
118     //Shear the shape w.r.t x-axis;
119
120     glBegin(GL_POLYGON);
121     glColor4f(0.0f,0.1f,0.7f,0.4f);
122
123     for(auto point : shape) {
124         vector<ll> point_matrix = getHomogeneousPointCoords(point);
125         pair<ld,ld> sheared_point = getPoint(shear(point_matrix, X_SHEAR, ↵
126             3));
127
128         glVertex2d(sheared_point.X,sheared_point.Y);
129     }
130
131     glEnd();
132
133     //Shear the shape w.r.t x-axis with y_ref;
134
135     glBegin(GL_POLYGON);

```

```

135     glColor4f(0.0f,0.1f,0.7f,0.4f);
136
137     for(auto point : shape) {
138         vector<ll> point_matrix = getHomogeneousPointCoords(point);
139         pair<ld,ld> sheared_point = getPoint(shear(point_matrix, X_SHEAR, ↵
            3, -30));
140
141         glVertex2d(sheared_point.X,sheared_point.Y);
142     }
143
144     glEnd();
145
146     //Shear the shape w.r.t y-axis;
147
148     glBegin(GL_POLYGON);
149     glColor4f(0.502f,0.0f,0.502f,0.6f);
150
151     for(auto point : shape) {
152         vector<ll> point_matrix = getHomogeneousPointCoords(point);
153         pair<ld,ld> sheared_point = getPoint(shear(point_matrix, Y_SHEAR, ↵
            3));
154
155         glVertex2d(sheared_point.X,sheared_point.Y);
156     }
157
158     glEnd();
159
160     //Shear the shape w.r.t y-axis with x_ref;
161
162     glBegin(GL_POLYGON);
163     glColor4f(0.502f,0.0f,0.502f,0.6f);
164
165     for(auto point : shape) {
166         vector<ll> point_matrix = getHomogeneousPointCoords(point);
167         pair<ld,ld> sheared_point = getPoint(shear(point_matrix, Y_SHEAR, ↵
            3, -30));
168
169         glVertex2d(sheared_point.X,sheared_point.Y);
170     }
171
172     glEnd();
173
174     //QUAD - 2
175
176     //Plot original shape;
177
178     shape = {{0,0}, {-40,0}, {-40,40}, {0,40}};

```

```

179
180     glBegin(GL_POLYGON);
181     glColor4f(0.486f,0.988f,0.000f,0.4f);
182
183     for(auto point : shape) {
184         glVertex2d(point.X,point.Y);
185     }
186
187     glEnd();
188
189     //Shear the shape w.r.t x-axis;
190
191     glBegin(GL_POLYGON);
192     glColor4f(0.596f,0.984f,0.596f,0.7f);
193
194     for(auto point : shape) {
195         vector<ll> point_matrix = getHomogeneousPointCoords(point);
196         pair<ld,ld> sheared_point = getPoint(shear(point_matrix, X_SHEAR, ↵
            -3));
197
198         glVertex2d(sheared_point.X,sheared_point.Y);
199     }
200
201     glEnd();
202
203     //Shear the shape w.r.t x-axis with y_ref;
204
205     glBegin(GL_POLYGON);
206     glColor4f(0.596f,0.984f,0.596f,0.7f);
207
208     for(auto point : shape) {
209         vector<ll> point_matrix = getHomogeneousPointCoords(point);
210         pair<ld,ld> sheared_point = getPoint(shear(point_matrix, X_SHEAR, ↵
            -3, -30));
211
212         glVertex2d(sheared_point.X,sheared_point.Y);
213     }
214
215     glEnd();
216
217     //Shear the shape w.r.t y-axis;
218
219     glBegin(GL_POLYGON);
220     glColor4f(0.678f,1.0f,0.184f,0.7f);
221
222     for(auto point : shape) {
223         vector<ll> point_matrix = getHomogeneousPointCoords(point);

```



```

224         pair<ld,ld> sheared_point = getPoint(shear(point_matrix, Y_SHEAR, ↵
           -3));
225
226         glVertex2d(sheared_point.X,sheared_point.Y);
227     }
228
229     glEnd();
230
231     //Shear the shape w.r.t y-axis with x_ref;
232
233     glBegin(GL_POLYGON);
234     glColor4f(0.678f,1.0f,0.184f,0.7f);
235
236     for(auto point : shape) {
237         vector<ll> point_matrix = getHomogeneousPointCoords(point);
238         pair<ld,ld> sheared_point = getPoint(shear(point_matrix, Y_SHEAR, ↵
           -3, 30));
239
240         glVertex2d(sheared_point.X,sheared_point.Y);
241     }
242
243     glEnd();
244
245     //QUAD - 3
246
247     //Plot original shape;
248
249     shape = {{0,0}, {-40,0}, {-40,-40}, {0,-40}};
250
251     glBegin(GL_POLYGON);
252     glColor4f(0.824f,0.412f,0.118f,0.6f);
253
254     for(auto point : shape) {
255         glVertex2d(point.X,point.Y);
256     }
257
258     glEnd();
259
260     //Shear the shape w.r.t x-axis;
261
262     glBegin(GL_POLYGON);
263     glColor4f(0.722f,0.525f,0.043f,0.7f);
264
265     for(auto point : shape) {
266         vector<ll> point_matrix = getHomogeneousPointCoords(point);
267         pair<ld,ld> sheared_point = getPoint(shear(point_matrix, X_SHEAR, ↵
           3));

```

```

268
269         glVertex2d(sheared_point.X,sheared_point.Y);
270     }
271
272     glEnd();
273
274     //Shear the shape w.r.t x-axis with y_ref;
275
276     glBegin(GL_POLYGON);
277     glColor4f(0.722f,0.525f,0.043f,0.7f);
278
279     for(auto point : shape) {
280         vector<ll> point_matrix = getHomogeneousPointCoords(point);
281         pair<ld,ld> sheared_point = getPoint(shear(point_matrix, X_SHEAR, ↵
282             3, 30));
283
284         glVertex2d(sheared_point.X,sheared_point.Y);
285     }
286
287     glEnd();
288
289     //Shear the shape w.r.t y-axis;
290
291     glBegin(GL_POLYGON);
292     glColor4f(0.627f,0.322f,0.176f,0.6f);
293
294     for(auto point : shape) {
295         vector<ll> point_matrix = getHomogeneousPointCoords(point);
296         pair<ld,ld> sheared_point = getPoint(shear(point_matrix, Y_SHEAR, ↵
297             3));
298
299         glVertex2d(sheared_point.X,sheared_point.Y);
300     }
301
302     glEnd();
303
304     //Shear the shape w.r.t y-axis with x_ref;
305
306     glBegin(GL_POLYGON);
307     glColor4f(0.627f,0.322f,0.176f,0.6f);
308
309     for(auto point : shape) {
310         vector<ll> point_matrix = getHomogeneousPointCoords(point);
311         pair<ld,ld> sheared_point = getPoint(shear(point_matrix, Y_SHEAR, ↵
312             3, 30));
313
314         glVertex2d(sheared_point.X,sheared_point.Y);
315     }
316
317     glEnd();

```

```

312     }
313
314     glEnd();
315
316     //QUAD - 4
317
318     //Plot original shape;
319
320     shape = {{0,0}, {40,0}, {40,-40}, {0,-40}};
321
322     glBegin(GL_POLYGON);
323     glColor4f(0.498f,1.0f,0.831f,0.6f);
324
325     for(auto point : shape) {
326         glVertex2d(point.X,point.Y);
327     }
328
329     glEnd();
330
331     //Shear the shape w.r.t x-axis;
332
333     glBegin(GL_POLYGON);
334     glColor4f(0.0f,0.545f,0.545f,0.6f);
335
336     for(auto point : shape) {
337         vector<ll> point_matrix = getHomogeneousPointCoords(point);
338         pair<ld,ld> sheared_point = getPoint(shear(point_matrix, X_SHEAR, ↵
            -3));
339
340         glVertex2d(sheared_point.X,sheared_point.Y);
341     }
342
343     glEnd();
344
345     //Shear the shape w.r.t x-axis with y_ref;
346
347     glBegin(GL_POLYGON);
348     glColor4f(0.0f,0.545f,0.545f,0.6f);
349
350     for(auto point : shape) {
351         vector<ll> point_matrix = getHomogeneousPointCoords(point);
352         pair<ld,ld> sheared_point = getPoint(shear(point_matrix, X_SHEAR, ↵
            -3, 30));
353
354         glVertex2d(sheared_point.X,sheared_point.Y);
355     }
356

```

```

357     glEnd();
358
359     //Shear the shape w.r.t y-axis;
360
361     glBegin(GL_POLYGON);
362     glColor4f(0.282f,0.820f,0.800f,0.6f);
363
364     for(auto point : shape) {
365         vector<ll> point_matrix = getHomogeneousPointCoords(point);
366         pair<ld,ld> sheared_point = getPoint(shear(point_matrix, Y_SHEAR, ↵
            -3));
367
368         glVertex2d(sheared_point.X,sheared_point.Y);
369     }
370
371     glEnd();
372
373     //Shear the shape w.r.t y-axis with x_ref;
374
375     glBegin(GL_POLYGON);
376     glColor4f(0.282f,0.820f,0.800f,0.6f);
377
378     for(auto point : shape) {
379         vector<ll> point_matrix = getHomogeneousPointCoords(point);
380         pair<ld,ld> sheared_point = getPoint(shear(point_matrix, Y_SHEAR, ↵
            -3, -30));
381
382         glVertex2d(sheared_point.X,sheared_point.Y);
383     }
384
385     glEnd();
386
387 }
388
389 pair<ld,ld> getPoint(vector<ld> point_matrix) {
390     ll h = point_matrix[2];
391     ld x = point_matrix[0];
392     ld y = point_matrix[1];
393
394     return {x/h,y/h};
395 }
396
397 vector<ll> getHomogeneousPointCoords(pair<ll,ll> point, ll h) {
398     vector<ll> point_matrix;
399     point_matrix.push_back(h*point.first);
400     point_matrix.push_back(h*point.second);
401     point_matrix.push_back(h);

```

```

402     return point_matrix;
403 }
404
405 vector<ld> shear(vector<ll> &point_matrix, ShearType type, ld sh, ld ↵
    refLine) {
406
407     vector<vector<ld>> shear_matrix;
408
409     switch (type)
410     {
411         case X_SHEAR:
412             shear_matrix = {
413                                     {1,sh,-sh*refLine},
414                                     {0,1,0},
415                                     {0,0,1}
416             };
417             break;
418
419         case Y_SHEAR:
420             shear_matrix = {
421                                     {1,0,0},
422                                     {sh,1,-sh*refLine},
423                                     {0,0,1}
424             };
425             break;
426
427         default:
428             shear_matrix = {
429                                     {1,0,0},
430                                     {0,1,0},
431                                     {0,0,1}
432             };
433             break;
434     }
435
436     return multiply(shear_matrix, point_matrix);
437 }
438
439 vector<ld> multiply(vector<vector<ld>> a, vector<ll> b) {
440     vector<ld> result;
441     for(int i=0;i<a.size();i++) {
442         ll temp = multiply(a[i],b);
443         result.push_back(temp);
444     }
445     return result;
446 }
447

```

```
448 ld multiply(vector<ld> a, vector<ll> b) {  
449     ld result=0;  
450     for(int i=0;i<a.size();i++) {  
451         result+=(a[i]*b[i]);  
452     }  
453     return result;  
454 }
```

SAMPLE I/O

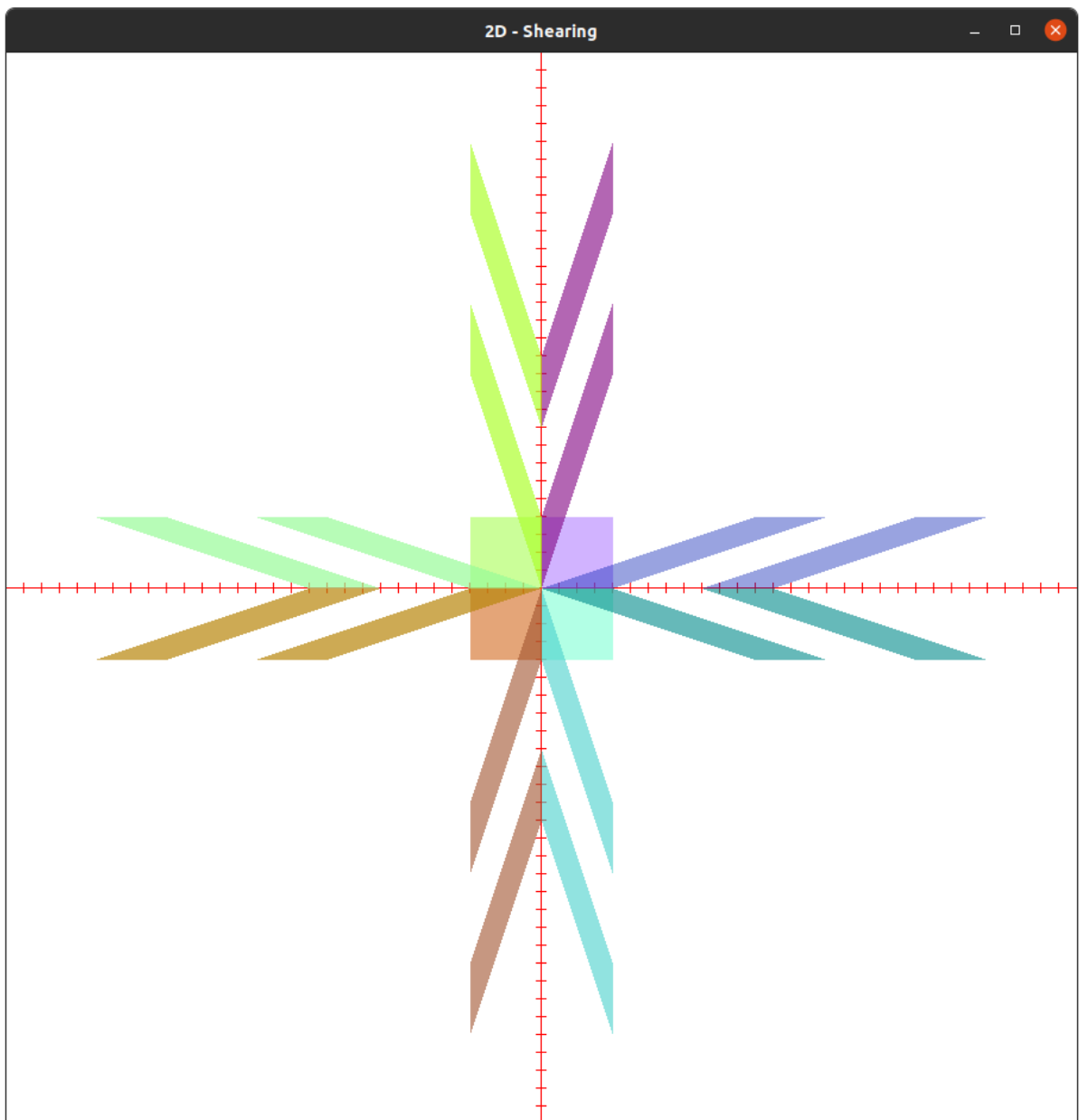


Figure 5: In each quadrant the square is (a) x-sheared. (b) x-sheared w.r.t a reference line. (c) y-sheared. (d) y-sheared w.r.t a reference line.

RESULT

The code to implement 2d transformations are written and output is verified.
