EX-06 - 2D Composite Transformations and Windowing in C++ using OpenGL

04/09/2021

Venkataraman Nagarajan, CSE - C

18500192

AIM

To implement 2d-Composite Transformations and windowing in C++.

SPECIFICATION

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

   The transformation can be any combination of the following.

   1) Translation
   2) Rotation
   3) Scaling
   4) Reflection
   5) Shearing

   Display the original and the transformed object. Calculate the final transformation matrix by multiplying the two individual transformation matrices and then apply it to the object.

   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)

b) 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.

PROGRAM - 01

Composite Transformations

```
1  // To compute the composite transformation matrix for any 2 ↩
       transformations given as input by the user and applying it on the ↩
       object.
2
3  // The transformation can be any combination of the following.
4  //     1) Translation
5  //     2) Rotation
6  //     3) Scaling
7  //     4) Reflection
8  //     5) Shearing
9
10 // Display the original and the transformed object.
11
12 // Calculate the final transformation matrix by multiplying the two ↩
       individual transformation
13 // matrices and then apply it to the object.
14
15 // Note: Use Homogeneous coordinate representations and matrix ↩
       multiplication to perform
16 // transformations. Divide the output window into four quadrants. (Use ↩
       LINES primitive to draw x
17 // and y axis)
18
19 #include<bits/stdc++.h>
20 #include<GL/glut.h>
21
22 using namespace std;
23 using ld = long double;
24 using ll = long long;
25
26 #define X       first
27 #define Y       second
28
29 const int WINDOW_WIDTH = 900;
30 const int WINDOW_HEIGHT = 900;
31
32 const int X_MIN = -300;
33 const int X_MAX = 300;
34 const int Y_MIN = -300;
35 const int Y_MAX = 300;
36
37 const ld PADDING = 0;
38 const ld STEP = 10;
```

```
39 const ld SCALE = 1;
40 const ld PI = 3.14159265358979323846264338327950288419716939937510582;
41
42 enum ReflectionType {X_AXIS, Y_AXIS, ORIGIN, X_EQUALS_Y_LINE, INVALID_};
43 enum ShearType {X_SHEAR, Y_SHEAR, INVALID};
44
45 void myInit();
46 void myDisplay();
47
48 void printAxes();
49 ld getRadian(ld degree);
50 ld multiply(vector<ld> a, vector<ll> b);
51 vector<ld> multiply(vector<vector<ld>> &a, vector<ll> b);
52 vector<vector<ld>> multiply(vector<vector<ld>> &a, vector<vector<ld>> &b);
53
54 pair<ld,ld> getPoint(vector<ld> point_matrix);
55 vector<ll> getHomogeneousPointCoords(pair<ll,ll> point, ll h=1);
56 vector<vector<ld>> getTransformMatrix();
57
58 vector<vector<ld>> translate(ld tx=0, ld ty=0);
59 vector<vector<ld>> rotate(ld angle=0, pair<ll,ll> pivot=make_pair(0,0));
60 vector<vector<ld>> scale(ld sx=1, ld sy=2, pair<ll,ll> pivot=make_pair↩
      (0,0));
61 vector<vector<ld>> reflect(ReflectionType type=ORIGIN);
62 vector<vector<ld>> shear(ShearType type=X_SHEAR, ld sh=1, ld refLine=0);
63
64 void transformShape();
65
66 int main(int argc,char* argv[]) {
67     glutInit(&argc,argv);
68     glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
69     glutInitWindowSize(WINDOW_WIDTH,WINDOW_HEIGHT);
70     glutCreateWindow("2D - Composite transforms");
71     glutDisplayFunc(myDisplay);
72     myInit();
73     glutMainLoop();
74     return 1;
75 }
76
77 void myInit() {
78     glClearColor(1.0,1.0,1.0,0.0);
79     glColor3f(0.0f,0.0f,0.0f);
80     glPointSize(5.0);
81     glMatrixMode(GL_PROJECTION);
82     glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
83     glEnable( GL_BLEND );
84     glLoadIdentity();
```

```
85      gluOrtho2D(X_MIN,X_MAX,Y_MIN,Y_MAX);
86  }
87
88  void myDisplay() {
89      glClear(GL_COLOR_BUFFER_BIT);
90
91      printAxes();
92      transformShape();
93
94      glFlush();
95  }
96
97  void printAxes() {
98      glBegin(GL_LINES);
99
100     glColor3f(1.0f,0.0f,0.0f);
101     glVertex2d(X_MIN,0);
102     glVertex2d(X_MAX,0);
103
104     glColor3f(1.0f,0.0f,0.0f);
105     glVertex2d(0,Y_MIN);
106     glVertex2d(0,Y_MAX);
107
108     for(ll i=X_MIN;i<X_MAX;i+=STEP) {
109         glVertex2d(i,-0.3*STEP);
110         glVertex2d(i,0.3*STEP);
111     }
112
113     for(ll i=Y_MIN;i<Y_MAX;i+=STEP) {
114         glVertex2d(-0.3*STEP,i);
115         glVertex2d(0.3*STEP,i);
116     }
117
118     glEnd();
119 }
120
121 vector<vector<ld>> getTransformMatrix() {
122     ll no_of_transformations;
123     string type;
124
125     vector<vector<ld>> transform_matrix = {{1,0,0},{0,1,0},{0,0,1}};
126
127     cout << "\n\t\t Composite Transformations \n\n";
128     cout << "Input Format: \n";
129     cout << "\t'T' tx ty (Translation by (tx,ty))\n";
130     cout << "\t'Ro' angle rx ry (Rotation by angle with ref to (rx,ry))\n"↩
        ;
```

```
131     cout << "\t'Sc' sx sy fx fy (Scale by sx & sy with ref to (fx,fy))\n";
132     cout << "\t'Re' type['X'/'Y'/'O'/'XY'] (Reflection about x-axis, y-axis, origin or x=y line)\n";
133     cout << "\t'Sh' type['X'/'Y'] sh refline ('type'_Shear by sh with ref to refline)\n";
134
135     cout << "\nSample Input: \n";
136     cout << "\t3\n";
137     cout << "\tT 4.5 10\n";
138     cout << "\tRo 45 0 0\n";
139     cout << "\tRe XY\n";
140
141     cout << "\nYour Input: \n";
142     cin >> no_of_transformations;
143
144     for(ll i=0;i<no_of_transformations;i++) {
145         cin>>type;
146         if(type=="T") {
147             ld tx,ty;
148             cin>>tx>>ty;
149
150             vector<vector<ld>> translation_matrix = translate(tx,ty);
151             transform_matrix = multiply(translation_matrix, transform_matrix);
152         } else if(type=="Ro") {
153             ld angle,rx,ry;
154             cin>>angle>>rx>>ry;
155
156             vector<vector<ld>> rotation_matrix = rotate(angle, make_pair(rx,ry));
157             transform_matrix = multiply(rotation_matrix, transform_matrix);
158         } else if(type=="Sc") {
159             ld sx,sy,fx,fy;
160             cin>>sx>>sy>>fx>>fy;
161
162             vector<vector<ld>> scaling_matrix = scale(sx,sy,make_pair(fx,fy));
163             transform_matrix = multiply(scaling_matrix, transform_matrix);
164         } else if(type=="Re") {
165             string type;
166             vector<vector<ld>> reflection_matrix;
167             cin>>type;
168
169             if(type=="X")       reflection_matrix = reflect(X_AXIS);
170             else if(type=="Y")  reflection_matrix = reflect(Y_AXIS);
171             else if(type=="O")  reflection_matrix = reflect(ORIGIN);
```

```
172             else if(type=="XY") reflection_matrix = reflect(↩
                    X_EQUALS_Y_LINE);
173             else                reflection_matrix = reflect(INVALID_);
174 
175             transform_matrix = multiply(reflection_matrix, ↩
                    transform_matrix);
176         } else if(type=="Sh") {
177             string type;
178             ld sh,refLine;
179             vector<vector<ld>> shear_matrix;
180             cin>>type>>sh>>refLine;
181 
182             if(type=="X")       shear_matrix = shear(X_SHEAR,sh,refLine);
183             else if(type=="Y")  shear_matrix = shear(Y_SHEAR,sh,refLine);
184             else                shear_matrix = shear(INVALID,sh,refLine);
185 
186             transform_matrix = multiply(shear_matrix, transform_matrix);
187         }
188     }
189 
190     cout << "\nOutput is displayed on the window\n\n";
191 
192     return transform_matrix;
193 }
194 
195 void transformShape() {
196 
197     vector<pair<ll,ll>> shape;
198 
199     //Plot original shape;
200 
201     shape = {{20,60}, {60,60}, {60,20}, {20,20}};
202 
203     glBegin(GL_POLYGON);
204     glColor4f(0.7f,0.0f,1.0f,0.6f);
205 
206     for(auto point : shape) {
207         glVertex2d(point.X,point.Y);
208     }
209 
210     glEnd();
211 
212     //Plot transformed shape;
213 
214     vector<vector<ld>> transform_matrix = getTransformMatrix();
215 
216     glBegin(GL_POLYGON);
```

```
217     glColor4f(0.82f,0.53f,1.0f,1.0f);
218
219     for(auto point : shape) {
220         pair<ld,ld> transformed_point = getPoint(multiply(transform_matrix↩
                , getHomogeneousPointCoords(point)));
221         glVertex2d(transformed_point.X,transformed_point.Y);
222     }
223
224     glEnd();
225
226 }
227
228 pair<ld,ld> getPoint(vector<ld> point_matrix) {
229     ll h = point_matrix[2];
230     ld x = point_matrix[0];
231     ld y = point_matrix[1];
232
233     return {x/h,y/h};
234 }
235
236 vector<ll> getHomogeneousPointCoords(pair<ll,ll> point, ll h) {
237     vector<ll> point_matrix;
238     point_matrix.push_back(h*point.first);
239     point_matrix.push_back(h*point.second);
240     point_matrix.push_back(h);
241     return point_matrix;
242 }
243
244 ld getRadian(ld degree) {
245     return degree*PI/180;
246 }
247
248 vector<vector<ld>> translate(ld tx, ld ty) {
249     vector<vector<ld>> translate_matrix = {
250                                             {1,0,tx},
251                                             {0,1,ty},
252                                             {0,0,1}
253                                         };
254     return translate_matrix;
255 }
256
257 vector<vector<ld>> scale(ld sx, ld sy, pair<ll,ll> pivot) {
258
259     ll xf = pivot.X;
260     ll yf = pivot.Y;
261
262     vector<vector<ld>> scale_matrix = {
```

```
                                {sx, 0, xf*(1-sx)},
                                {0, sy, yf*(1-sy)},
                                {0,  0, 1}
                            };
    return scale_matrix;
}

vector<vector<ld>> rotate(ld angle, pair<ll,ll> pivot) {
    angle = getRadian(angle);
    ll xr = pivot.X;
    ll yr = pivot.Y;

    vector<vector<ld>> rotate_matrix = {
                                {cos(angle), -sin(angle), xr*(1-cos(angle)↩
                                    + yr*sin(angle))},
                                {sin(angle),  cos(angle), yr*(1-cos(angle)↩
                                    - xr*sin(angle))},
                                {0         ,  0         ,1}
                            };
    return rotate_matrix;
}

vector<vector<ld>> reflect(ReflectionType type) {

    vector<vector<ld>> reflection_matrix;

    switch (type)
    {
        case X_AXIS:
            reflection_matrix = {
                                    {1,0,0},
                                    {0,-1,0},
                                    {0,0,1}
                                };
            break;

        case Y_AXIS:
            reflection_matrix = {
                                    {-1,0,0},
                                    {0,1,0},
                                    {0,0,1}
                                };
            break;

        case ORIGIN:
            reflection_matrix = {
                                    {-1,0,0},
```

```
                                    {0,-1,0},
                                    {0,0,1}
                                };
            break;

        case X_EQUALS_Y_LINE:
            reflection_matrix = {
                                    {0,1,0},
                                    {1,0,0},
                                    {0,0,1}
                                };
            break;

        default:
            reflection_matrix = {
                                    {1,0,0},
                                    {0,1,0},
                                    {0,0,1}
                                };
            break;
    }

    return reflection_matrix;
}

vector<vector<ld>> shear(ShearType type, ld sh, ld refLine) {

    vector<vector<ld>> shear_matrix;

    switch (type)
    {
        case X_SHEAR:
            shear_matrix = {
                                    {1,sh,-sh*refLine},
                                    {0,1,0},
                                    {0,0,1}
                                };
            break;

        case Y_SHEAR:
            shear_matrix = {
                                    {1,0,0},
                                    {sh,1,-sh*refLine},
                                    {0,0,1}
                                };
            break;

```

```
        default:
            shear_matrix = {
                                    {1,0,0},
                                    {0,1,0},
                                    {0,0,1}
                                };
            break;
    }

    return shear_matrix;
}

vector<vector<ld>> multiply(vector<vector<ld>> &a, vector<vector<ld>> &b) {
    vector<vector<ld>> result;
    for(int i=0; i<a.size(); i++) {
        vector<ld> row;
        for(int j=0; j<b[0].size(); j++) {
            ld sum = 0;
            for(int k=0; k<a[0].size(); k++) {
                sum += a[i][k]*b[k][j];
            }
            row.push_back(sum);
        }
        result.push_back(row);
    }
    return result;
}

vector<ld> multiply(vector<vector<ld>> &a, vector<ll> b) {
    vector<ld> result;
    for(int i=0;i<a.size();i++) {
        ll temp = multiply(a[i],b);
        result.push_back(temp);
    }
    return result;
}

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

SAMPLE I/0

```
LAB/Multimedia and graphics Lab(main*) » g++ "EX06 - 2D Composite Transforms and Windowing/01-CompositeTransform.cpp" -lGL -lGLU -lglut && ./a.out

                Composite Transformations

Input Format:
        'T' tx ty (Translation by (tx,ty))
        'Ro' angle rx ry (Rotation by angle with ref to (rx,ry))
        'Sc' sx sy fx fy (Scale by sx & sy with ref to (fx,fy))
        'Re' type['X'/'Y'/'O'/'XY'] (Reflection about x-axis, y-axis, origin or x=y line)
        'Sh' type['X'/'Y'] sh refline ('type'_Shear by sh with ref to refline)

Sample Input:
        3
        T 4.5 10
        Ro 45 0 0
        Re XY

Your Input:
2
Ro 135 0 0
T -20 -40

Output is displayed on the window
```

Figure 1: Input from the user

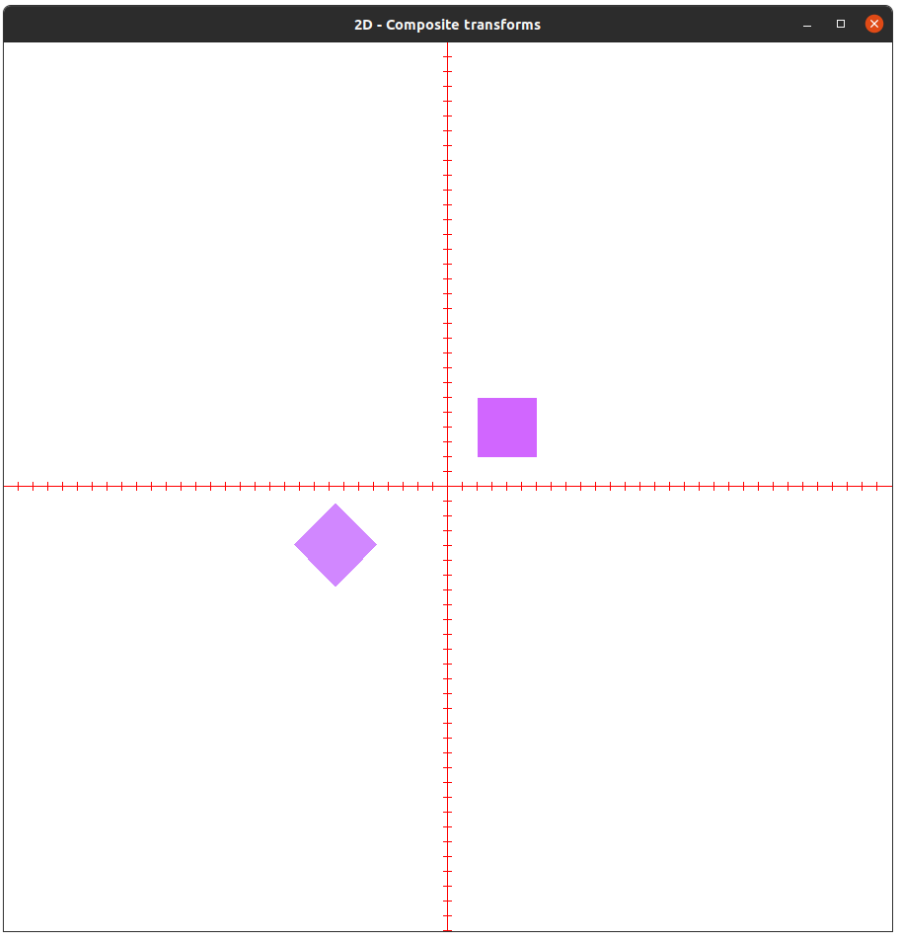

Figure 2: The solid square is transformed to the translucent one

PROGRAM - 02

Window to Viewport

```
// 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.

#include<bits/stdc++.h>
#include<GL/glut.h>

using namespace std;
using ld = long double;
using ll = long long;

#define X       first
#define Y       second

const int WINDOW_WIDTH = 1000;
const int WINDOW_HEIGHT = 1000;

const int X_MIN = -30;
const int X_MAX = 420;
const int Y_MIN = -30;
const int Y_MAX = 300;

const ld PADDING = 0;
const ld STEP = 10;
const ld SCALE = 1;

struct Display {
    ld X_MIN, X_MAX, Y_MIN, Y_MAX;

    Display(ld X_MIN, ld X_MAX, ld Y_MIN, ld Y_MAX):  X_MIN(X_MIN), X_MAX(↩
       X_MAX), Y_MIN(Y_MIN), Y_MAX(Y_MAX) {}

    void draw(ld Red = 0.0f, ld Green = 0.0f, ld Blue = 0.0f, ld Alpha = ↩
       1.0) {
        glBegin(GL_LINE_LOOP);
        glColor4f(Red, Green, Blue, Alpha);

        glVertex2d(X_MIN, Y_MIN);
        glVertex2d(X_MIN, Y_MAX);
        glVertex2d(X_MAX, Y_MAX);
        glVertex2d(X_MAX, Y_MIN);

        glEnd();
```

```
42      }
43  };
44
45  void myInit();
46  void myDisplay();
47
48  ld multiply(vector<ld> a, vector<ll> b);
49  vector<ld> multiply(vector<vector<ld>> &a, vector<ll> b);
50  vector<vector<ld>> multiply(vector<vector<ld>> &a, vector<vector<ld>> &b);
51
52  pair<ld,ld> getPoint(vector<ld> point_matrix);
53  vector<ll> getHomogeneousPointCoords(pair<ll,ll> point, ll h=1);
54  vector<vector<ld>> getTransformMatrix();
55
56  vector<vector<ld>> translate(ld tx=0, ld ty=0);
57  vector<vector<ld>> scale(ld sx=1, ld sy=2, pair<ll,ll> pivot=make_pair↩
        (0,0));
58
59  void transformShape();
60  void drawViewport(Display window, Display viewport, vector<vector<pair<ll,↩
        ll>>> &shapes);
61
62  int main(int argc,char* argv[]) {
63      glutInit(&argc,argv);
64      glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
65      glutInitWindowSize(WINDOW_WIDTH,WINDOW_HEIGHT);
66      glutCreateWindow("2D - Window to Viewport Transformation");
67      glutDisplayFunc(myDisplay);
68      myInit();
69      glutMainLoop();
70      return 1;
71  }
72
73  void myInit() {
74      glClearColor(1.0,1.0,1.0,0.0);
75      glColor3f(0.0f,0.0f,0.0f);
76      glPointSize(5.0);
77      glMatrixMode(GL_PROJECTION);
78      glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
79      glEnable( GL_BLEND );
80      glLoadIdentity();
81      gluOrtho2D(X_MIN,X_MAX,Y_MIN,Y_MAX);
82  }
83
84  void myDisplay() {
85      glClear(GL_COLOR_BUFFER_BIT);
86
```

```
87      transformShape();
88
89      glFlush();
90  }
91
92  vector<vector<ld>> getTransformMatrix(Display window, Display viewport) {
93
94      vector<vector<ld>> transform_matrix = {{1,0,0},{0,1,0},{0,0,1}};
95
96      vector<vector<ld>> translate_matrix = translate(viewport.X_MIN - ↩
            window.X_MIN, viewport.Y_MIN - window.Y_MIN);
97      transform_matrix = multiply(translate_matrix, transform_matrix);
98
99      vector<vector<ld>> scale_matrix = scale(
100                                            (viewport.X_MAX - viewport.X_MIN)↩
                                                /(window.X_MAX - window.X_MIN)↩
                                                ,
101                                           (viewport.Y_MAX - viewport.Y_MIN)↩
                                                /(window.Y_MAX - window.Y_MIN)↩
                                                ,
102                                           {viewport.X_MIN, viewport.Y_MIN}
103                                       );
104     transform_matrix = multiply(scale_matrix, transform_matrix);
105
106     return transform_matrix;
107 }
108
109 void drawViewport(Display window, Display viewport, vector<vector<pair<ll,↩
        ll>>> &shapes) {
110
111     vector<vector<ld>> transform_matrix;
112
113     //Plot viewport;
114
115     transform_matrix = getTransformMatrix(window, viewport);
116     viewport.draw(1.0, 0.0, 0.7);
117     for(auto shape: shapes) {
118         glBegin(GL_POLYGON);
119         glColor4f(1.0f,0.0f,0.7f,1.0f);
120
121         for(auto point : shape) {
122             pair<ld,ld> viewpoint = getPoint(multiply(transform_matrix, ↩
                    getHomogeneousPointCoords(point)));
123             glVertex2d(viewpoint.X, viewpoint.Y);
124         }
125
126         glEnd();
```

```
127     }
128 
129 }
130 
131 void transformShape() {
132 
133     Display window = Display(10, 150, 10, 250);
134     Display viewport1 = Display(180, 260, 10, 140);
135     Display viewport2 = Display(180, 370, 170, 250);
136     Display viewport3 = Display(290, 370, 10, 90);
137 
138     vector<vector<pair<ll,ll>>> shapes = {
139                 {{20,60}, {60,60}, {60,20}, {20,20}},
140                 {{30,30}, {120,30}, {75,120}},
141                 {{40,200}, {60,230}, {140,190}, {80,160}, {60,140}},
142                 {{140,20}, {120,220}, {140,240}}
143             };
144 
145 
146     //Plot window;
147 
148     window.draw(0.7, 0.0, 1.0);
149     for(auto shape: shapes) {
150         glBegin(GL_POLYGON);
151         glColor4f(0.7f,0.0f,1.0f,1.0f);
152 
153         for(auto point : shape) {
154             glVertex2d(point.X,point.Y);
155         }
156 
157         glEnd();
158     }
159 
160     drawViewport(window, viewport1, shapes);
161     drawViewport(window, viewport2, shapes);
162     drawViewport(window, viewport3, shapes);
163 
164 }
165 
166 pair<ld,ld> getPoint(vector<ld> point_matrix) {
167     ll h = point_matrix[2];
168     ld x = point_matrix[0];
169     ld y = point_matrix[1];
170 
171     return {x/h,y/h};
172 }
173 
```

```
174 vector<ll> getHomogeneousPointCoords(pair<ll,ll> point, ll h) {
175     vector<ll> point_matrix;
176     point_matrix.push_back(h*point.first);
177     point_matrix.push_back(h*point.second);
178     point_matrix.push_back(h);
179     return point_matrix;
180 }
181
182 vector<vector<ld>> translate(ld tx, ld ty) {
183     vector<vector<ld>> translate_matrix = {
184                                             {1,0,tx},
185                                             {0,1,ty},
186                                             {0,0,1}
187                                         };
188     return translate_matrix;
189 }
190
191 vector<vector<ld>> scale(ld sx, ld sy, pair<ll,ll> pivot) {
192
193     ll xf = pivot.X;
194     ll yf = pivot.Y;
195
196     vector<vector<ld>> scale_matrix = {
197                                 {sx, 0, xf*(1-sx)},
198                                 {0, sy, yf*(1-sy)},
199                                 {0,  0, 1}
200                             };
201     return scale_matrix;
202 }
203
204 vector<vector<ld>> multiply(vector<vector<ld>> &a, vector<vector<ld>> &b) ↩
    {
205     vector<vector<ld>> result;
206     for(int i=0; i<a.size(); i++) {
207         vector<ld> row;
208         for(int j=0; j<b[0].size(); j++) {
209             ld sum = 0;
210             for(int k=0; k<a[0].size(); k++) {
211                 sum += a[i][k]*b[k][j];
212             }
213             row.push_back(sum);
214         }
215         result.push_back(row);
216     }
217     return result;
218 }
219
```

```
vector<ld> multiply(vector<vector<ld>> &a, vector<ll> b) {
    vector<ld> result;
    for(int i=0;i<a.size();i++) {
        ll temp = multiply(a[i],b);
        result.push_back(temp);
    }
    return result;
}

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

SAMPLE I/0

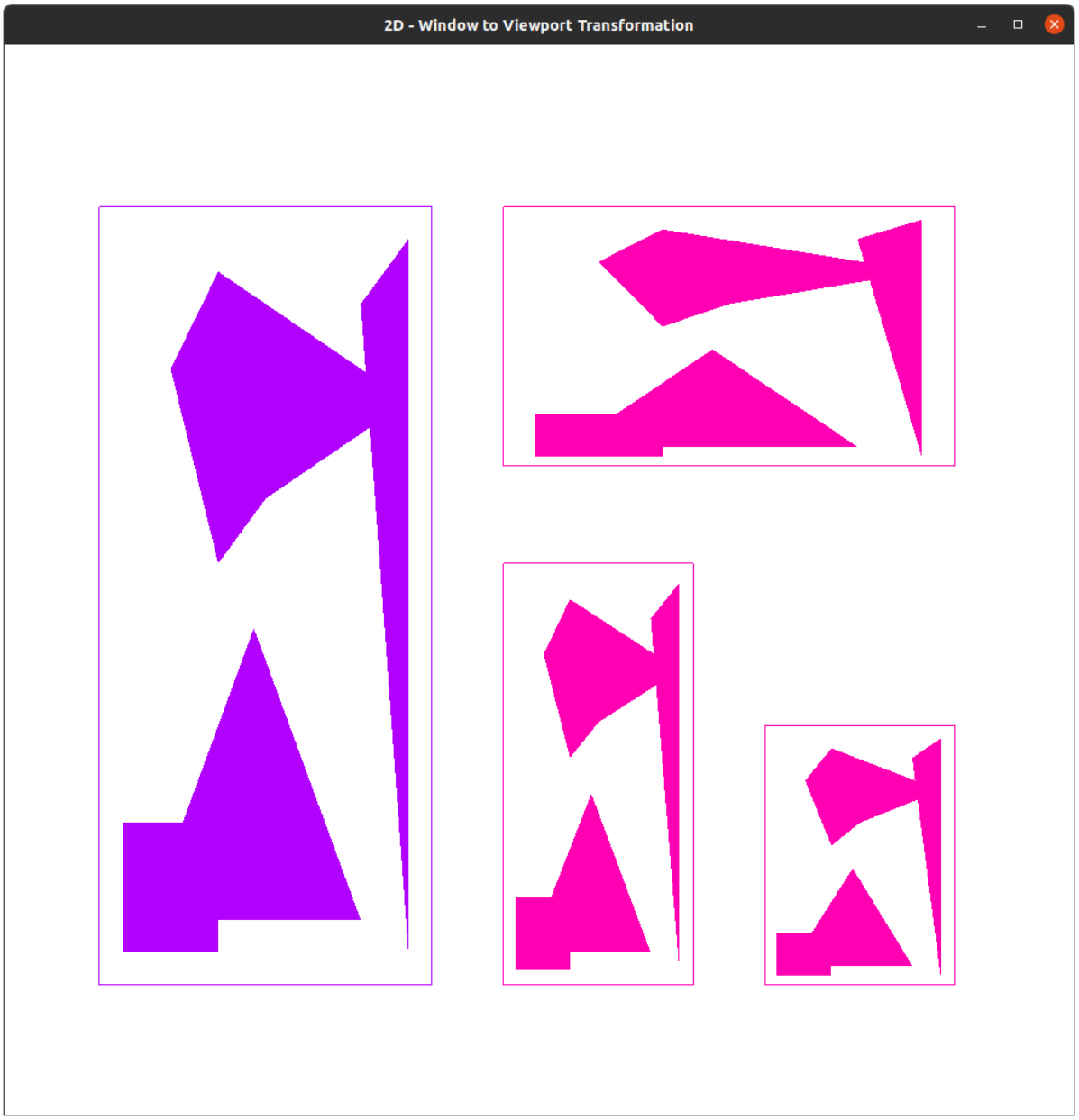

Figure 3: The violet window is transformed into different viewports

RESULT

The code to implement 2d composite transformations and window to viewport transformation are written and output is verified.
