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

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

// 1) Translation

#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 = 900;
const int WINDOW_HEIGHT = 900;

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

void myInit();
void myDisplay();

void printAxes();
ll multiply(vector<ll> a, vector<ll> b);
vector<ll> multiply(vector<vector<ll>> a, vector<ll> b);

vector<ll> getHomogeneousPointCoords(pair<ll,ll> point, ll h=1);
vector<ll> translate(vector<ll> &point_matrix, ll tx=0, ll ty=0);
pair<ll,ll> getPoint(vector<ll> point_matrix);
void translateShape();

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

int main(int argc,char* argv[]) {
    glutInit(&argc,argv);
    glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
    glutInitWindowSize(WINDOW_WIDTH,WINDOW_HEIGHT);
    glutCreateWindow("2D - Translation");
    glutDisplayFunc(myDisplay);
    myInit();
    glutMainLoop();
    return 1;
}

void myInit() {
    glClearColor(1.0,1.0,1.0,0.0);
    glColor3f(0.0f,0.0f,0.0f);
    glPointSize(2.0);
    glMatrixMode(GL_PROJECTION);
    glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
    glEnable( GL_BLEND );
    glLoadIdentity();
    gluOrtho2D(X_MIN,X_MAX,Y_MIN,Y_MAX);
}

void myDisplay() {
    glClear(GL_COLOR_BUFFER_BIT);

    printAxes();
    translateShape();

    glFlush();
}

void printAxes() {
    glBegin(GL_LINES);

    glColor3f(1.0f,0.0f,0.0f);
    glVertex2d(X_MIN,0);
    glVertex2d(X_MAX,0);

    glColor3f(1.0f,0.0f,0.0f);
    glVertex2d(0,Y_MIN);
    glVertex2d(0,Y_MAX);

    for(ll i=X_MIN;i<X_MAX;i+=STEP) {
        glVertex2d(i,-0.3*STEP);
        glVertex2d(i,0.3*STEP);
    }

    for(ll i=Y_MIN;i<Y_MAX;i+=STEP) {
        glVertex2d(-0.3*STEP,i);
        glVertex2d(0.3*STEP,i);
    }

    glEnd();
}

void translateShape() {
    //Plot original shape;

    vector<pair<ll,ll>> shape = {{20,20}, {20,50}, {50,50}, {50,20}};

    glBegin(GL_POLYGON);
    glColor4f(0.7f,0.0f,1.0f,1.0f);

    for(auto point : shape) {
        glVertex2d(point.X,point.Y);
    }

    glEnd();

    //Translate shape;

    glBegin(GL_POLYGON);
    glColor4f(0.7f,0.0f,1.0f,0.4f);

    for(auto point : shape) {
        vector<ll> point_matrix = getHomogeneousPointCoords(point);
        pair<ll,ll> translated_point = getPoint(translate(point_matrix,50,50));

        glVertex2d(translated_point.X,translated_point.Y);
    }

    glEnd();
}

pair<ll,ll> getPoint(vector<ll> point_matrix) {
    ll h = point_matrix[2];
    ll x = point_matrix[0];
    ll y = point_matrix[1];

    return {x/h,y/h};
}


vector<ll> getHomogeneousPointCoords(pair<ll,ll> point, ll h) {
    vector<ll> point_matrix;
    point_matrix.push_back(h*point.first);
    point_matrix.push_back(h*point.second);
    point_matrix.push_back(h);
    return point_matrix;
}

vector<ll> translate(vector<ll> &point_matrix, ll tx, ll ty) {
    vector<vector<ll>> translate_matrix = {
                                            {1,0,tx},
                                            {0,1,ty},
                                            {0,0,1}
                                        };
    return multiply(translate_matrix, point_matrix);
}

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

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

SAMPLE I/0

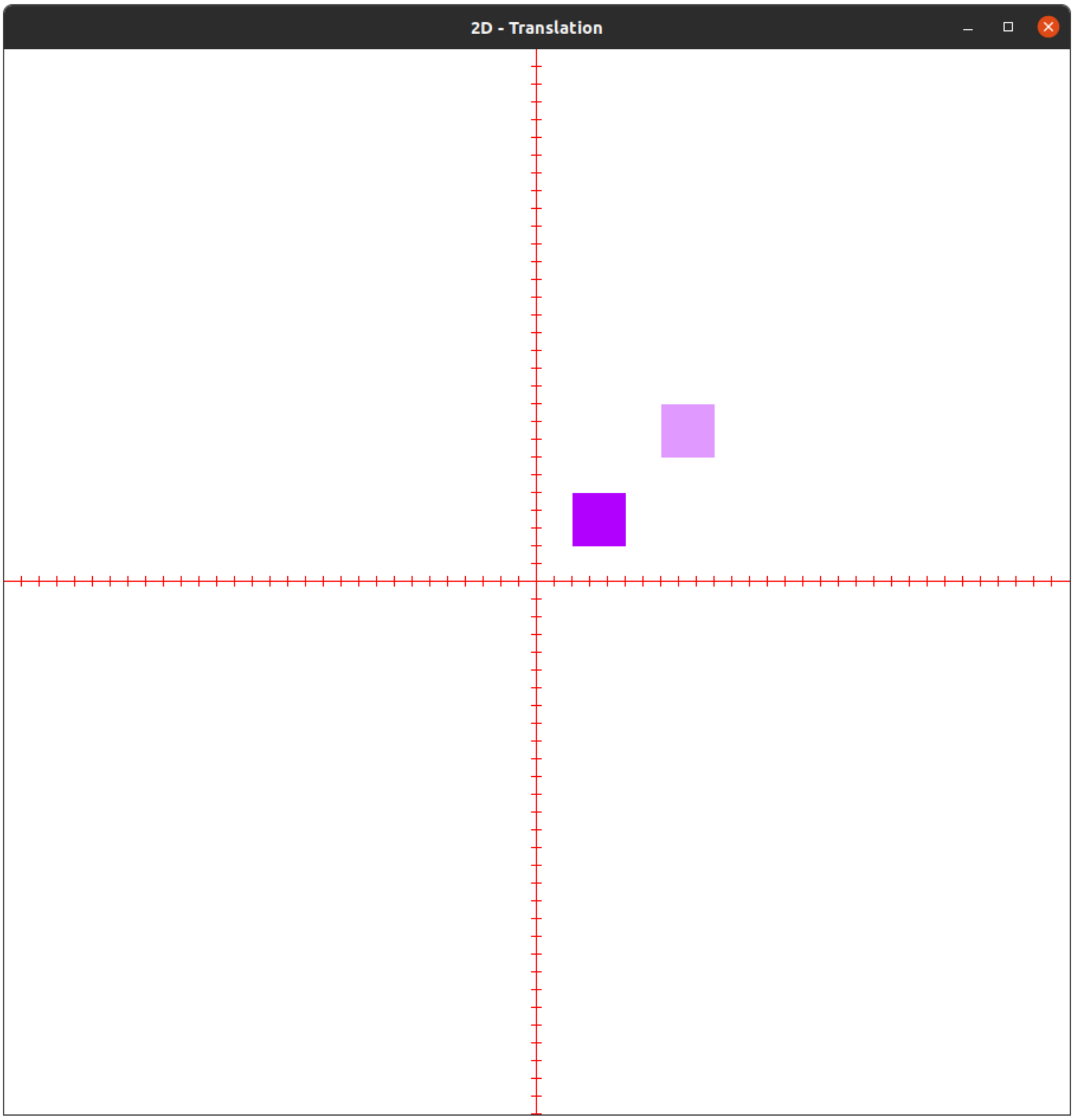

Figure 1: The solid square is translated to the translucent one

PROGRAM - 02

Rotation

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

// 2) Rotation
//      a) about origin
//      b) with respect to a fixed point (xr,yr)

#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 = 900;
const int WINDOW_HEIGHT = 900;

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

const ld PADDING = 0;
const ld STEP = 10;
const ld SCALE = 1;
const ld PI = 3.14159265358979323846264338327950288419716939937510582;

void myInit();
void myDisplay();

void printAxes();
ld getRadian(ld degree);
ld multiply(vector<ld> a, vector<ll> b);
vector<ld> multiply(vector<vector<ld>> a, vector<ll> b);

vector<ll> getHomogeneousPointCoords(pair<ll,ll> point, ll h=1);
vector<ld> rotate(vector<ll> &point_matrix, ld angle=0, pair<ll,ll> pivot=make_pair(0,0));
pair<ld,ld> getPoint(vector<ld> point_matrix);
void rotateShape();
```

```
int main(int argc,char* argv[]) {
    glutInit(&argc,argv);
    glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
    glutInitWindowSize(WINDOW_WIDTH,WINDOW_HEIGHT);
    glutCreateWindow("2D - Rotation");
    glutDisplayFunc(myDisplay);
    myInit();
    glutMainLoop();
    return 1;
}

void myInit() {
    glClearColor(1.0,1.0,1.0,0.0);
    glColor3f(0.0f,0.0f,0.0f);
    glPointSize(5.0);
    glMatrixMode(GL_PROJECTION);
    glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
    glEnable( GL_BLEND );
    glLoadIdentity();
    gluOrtho2D(X_MIN,X_MAX,Y_MIN,Y_MAX);
}

void myDisplay() {
    glClear(GL_COLOR_BUFFER_BIT);

    printAxes();
    rotateShape();

    glFlush();
}

void printAxes() {
    glBegin(GL_LINES);

    glColor3f(1.0f,0.0f,0.0f);
    glVertex2d(X_MIN,0);
    glVertex2d(X_MAX,0);

    glColor3f(1.0f,0.0f,0.0f);
    glVertex2d(0,Y_MIN);
    glVertex2d(0,Y_MAX);

    for(ll i=X_MIN;i<X_MAX;i+=STEP) {
        glVertex2d(i,-0.3*STEP);
        glVertex2d(i,0.3*STEP);
    }

    for(ll i=Y_MIN;i<Y_MAX;i+=STEP) {
        glVertex2d(-0.3*STEP,i);
        glVertex2d(0.3*STEP,i);
    }

    glEnd();
}

void rotateShape() {
    //Plot original shape;

    vector<pair<ll,ll>> shape = {{20,20}, {60,20}, {40,60}};

    glBegin(GL_POLYGON);
    glColor4f(0.7f,0.0f,1.0f,1.0f);

    for(auto point : shape) {
        glVertex2d(point.X,point.Y);
    }

    glEnd();

    //Rotate shape with respect to origin;

    glBegin(GL_POLYGON);
    glColor4f(0.7f,0.0f,1.0f,0.4f);

    for(auto point : shape) {
        vector<ll> point_matrix = getHomogeneousPointCoords(point);
        pair<ld,ld> rotated_point = getPoint(rotate(point_matrix,90));

        glVertex2d(rotated_point.X,rotated_point.Y);
    }

    glEnd();

    //Rotate shape with respect to a fixed point;

    pair<ll,ll> pivot = {40,70};

    glBegin(GL_POINTS);
    glColor4f(0.8f,0.0f,0.5f,0.9f);
    glVertex2d(pivot.X, pivot.Y);
    glEnd();

    glBegin(GL_POLYGON);
    glColor4f(0.7f,0.0f,1.0f,0.4f);

    for(auto point : shape) {
        vector<ll> point_matrix = getHomogeneousPointCoords(point);
        pair<ld,ld> rotated_point = getPoint(rotate(point_matrix,180,pivot));

        glVertex2d(rotated_point.X,rotated_point.Y);
    }

    glEnd();
}

pair<ld,ld> getPoint(vector<ld> point_matrix) {
    ll h = point_matrix[2];
    ld x = point_matrix[0];
    ld y = point_matrix[1];

    return {x/h,y/h};
}

ld getRadian(ld degree) {
    return degree*PI/180;
}

vector<ll> getHomogeneousPointCoords(pair<ll,ll> point, ll h) {
    vector<ll> point_matrix;
    point_matrix.push_back(h*point.first);
    point_matrix.push_back(h*point.second);
    point_matrix.push_back(h);
    return point_matrix;
}

vector<ld> rotate(vector<ll> &point_matrix, 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 multiply(rotate_matrix, point_matrix);
```

```
}

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

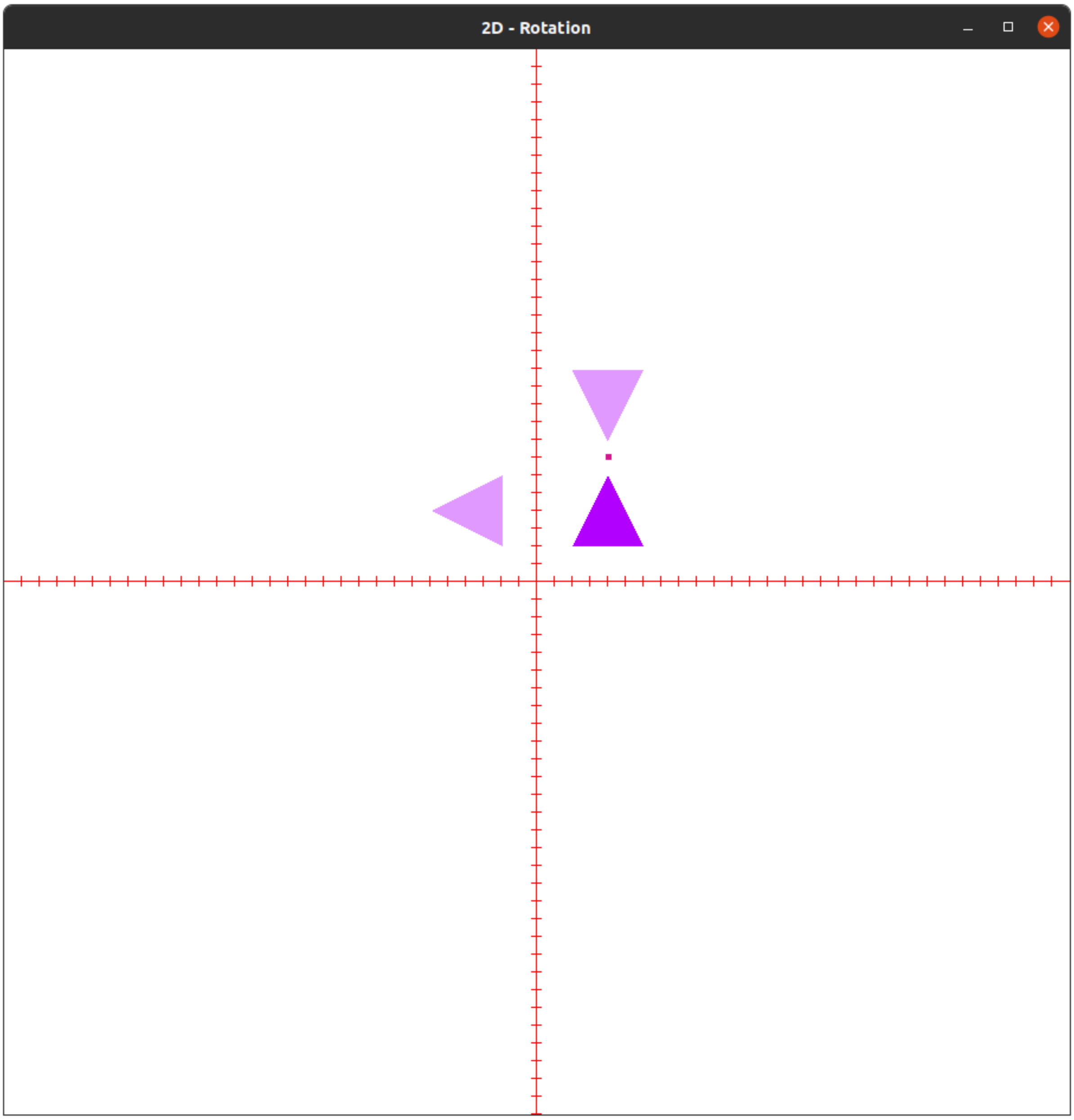

Figure 2: (a) The solid triangle is rotated $90°$ w.r.t origin to obtain triangle at 2^{nd} quadrant. (b) The solid triangle is rotated $180°$ with respect to the point shown to obtain the triangle above it.

PROGRAM - 03

Scaling

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

// 3) Scaling with respect to
//      a) origin - Uniform Vs Differential Scaling
//      b) fixed point (xf,yf)


#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 = 900;
const int WINDOW_HEIGHT = 900;

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

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

void myInit();
void myDisplay();

void printAxes();
ld multiply(vector<ld> a, vector<ll> b);
vector<ld> multiply(vector<vector<ld>> a, vector<ll> b);

vector<ll> getHomogeneousPointCoords(pair<ll,ll> point, ll h=1);
vector<ld> scale(vector<ll> &point_matrix, ld sx=1, ld sy=2, pair<ll,ll> pivot=make_pair(0,0));
pair<ld,ld> getPoint(vector<ld> point_matrix);
void scaleShape();

```

```

int main(int argc,char* argv[]) {
    glutInit(&argc,argv);
    glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
    glutInitWindowSize(WINDOW_WIDTH,WINDOW_HEIGHT);
    glutCreateWindow("2D - Scaling");
    glutDisplayFunc(myDisplay);
    myInit();
    glutMainLoop();
    return 1;
}

void myInit() {
    glClearColor(1.0,1.0,1.0,0.0);
    glColor3f(0.0f,0.0f,0.0f);
    glPointSize(5.0);
    glMatrixMode(GL_PROJECTION);
    glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
    glEnable( GL_BLEND );
    glLoadIdentity();
    gluOrtho2D(X_MIN,X_MAX,Y_MIN,Y_MAX);
}

void myDisplay() {
    glClear(GL_COLOR_BUFFER_BIT);

    printAxes();
    scaleShape();

    glFlush();
}

void printAxes() {
    glBegin(GL_LINES);

    glColor3f(1.0f,0.0f,0.0f);
    glVertex2d(X_MIN,0);
    glVertex2d(X_MAX,0);

    glColor3f(1.0f,0.0f,0.0f);
    glVertex2d(0,Y_MIN);
    glVertex2d(0,Y_MAX);

    for(ll i=X_MIN;i<X_MAX;i+=STEP) {
        glVertex2d(i,-0.3*STEP);
        glVertex2d(i,0.3*STEP);
    }

    for(ll i=Y_MIN;i<Y_MAX;i+=STEP) {
        glVertex2d(-0.3*STEP,i);
        glVertex2d(0.3*STEP,i);
    }

    glEnd();
}

void scaleShape() {

    vector<pair<ll,ll>> shape;

    //Plot original shape;

    shape = {{20,40}, {60,40}, {40,80}};

    glBegin(GL_POLYGON);
    glColor4f(0.7f,0.0f,1.0f,1.0f);

    for(auto point : shape) {
        glVertex2d(point.X,point.Y);
    }

    glEnd();

    //Scale shape w.r.t. origin - Uniform Scaling;

    glBegin(GL_POLYGON);
    glColor4f(0.7f,0.0f,1.0f,0.4f);

    for(auto point : shape) {
        vector<ll> point_matrix = getHomogeneousPointCoords(point);
        pair<ld,ld> scaled_point = getPoint(scale(point_matrix, 2, 2));

        glVertex2d(scaled_point.X,scaled_point.Y);
    }

    glEnd();




    //Plot original shape;

    shape = {{-20,40}, {-60,40}, {-40,80}};

    glBegin(GL_POLYGON);
    glColor4f(0.0f,1.0f,0.7f,1.0f);

    for(auto point : shape) {
        glVertex2d(point.X,point.Y);
    }

    glEnd();

    //Scale shape w.r.t. origin - Differential Scaling;

    glBegin(GL_POLYGON);
    glColor4f(0.0f,1.0f,0.7f,0.4f);

    for(auto point : shape) {
        vector<ll> point_matrix = getHomogeneousPointCoords(point);
        pair<ld,ld> scaled_point = getPoint(scale(point_matrix, 0.5, 2));

        glVertex2d(scaled_point.X,scaled_point.Y);
    }

    glEnd();




    //Plot original shape;

    shape = {{20,-40}, {60,-40}, {40,-80}};

    glBegin(GL_POLYGON);
    glColor4f(0.7f,1.0f,0.0f,0.7f);

    for(auto point : shape) {
        glVertex2d(point.X,point.Y);
    }

    glEnd();

    //Scale shape w.r.t. fixed point - Uniform Scaling;

    pair<ll,ll> pivot = {40, -60};

    glBegin(GL_POINTS);
    glColor4f(0.0f,0.0f,1.0f,0.9f);
    glVertex2d(pivot.X, pivot.Y);
    glEnd();

    glBegin(GL_POLYGON);
    glColor4f(0.7f,1.0f,0.0f,0.6f);

    for(auto point : shape) {
        vector<ll> point_matrix = getHomogeneousPointCoords(point);
        pair<ld,ld> scaled_point = getPoint(scale(point_matrix, 1.5, 1.5, pivot));

        glVertex2d(scaled_point.X,scaled_point.Y);
    }

    glEnd();




    //Plot original shape;

    shape = {{-20,-40}, {-60,-40}, {-40,-80}};

    glBegin(GL_POLYGON);
    glColor4f(1.0f,0.7f,0.0f,0.7f);

    for(auto point : shape) {
        glVertex2d(point.X,point.Y);
    }

    glEnd();

    //Scale shape w.r.t. fixed point - Differential Scaling;

    pivot = {-40, -60};

    glBegin(GL_POINTS);
    glColor4f(1.0f,0.0f,0.0f,0.9f);
    glVertex2d(pivot.X, pivot.Y);
    glEnd();

    glBegin(GL_POLYGON);
    glColor4f(1.0f,0.7f,0.0f,0.5f);

    for(auto point : shape) {
        vector<ll> point_matrix = getHomogeneousPointCoords(point);
        pair<ld,ld> scaled_point = getPoint(scale(point_matrix, 0.5, 1.5, pivot));

        glVertex2d(scaled_point.X,scaled_point.Y);
    }

    glEnd();

}

pair<ld,ld> getPoint(vector<ld> point_matrix) {
    ll h = point_matrix[2];
    ld x = point_matrix[0];
    ld y = point_matrix[1];

    return {x/h,y/h};
}

vector<ll> getHomogeneousPointCoords(pair<ll,ll> point, ll h) {
    vector<ll> point_matrix;
    point_matrix.push_back(h*point.first);
    point_matrix.push_back(h*point.second);
    point_matrix.push_back(h);
    return point_matrix;
}

vector<ld> scale(vector<ll> &point_matrix, ld sx, ld sy, pair<ll,ll> pivot) {

    ll xf = pivot.X;
    ll yf = pivot.Y;

    vector<vector<ld>> scale_matrix = {
                                {sx, 0, xf*(1-sx)},
                                {0, sy, yf*(1-sy)},
                                {0,  0, 1}
                            };
    return multiply(scale_matrix,point_matrix);
}

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

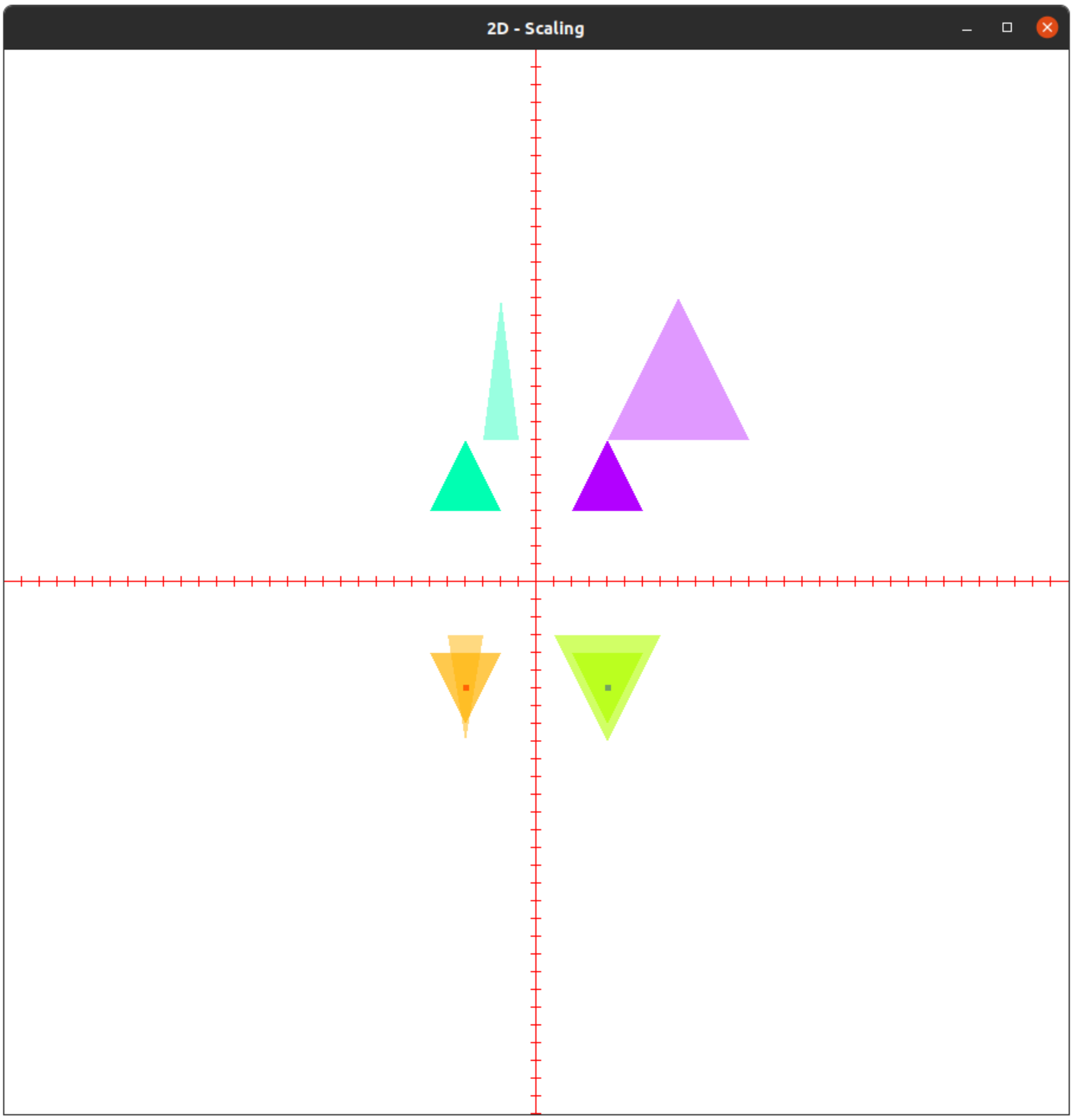

Figure 3: (a) The solid triangle in 1^{st} quadrant is scaled uniformly w.r.t origin to its translucent version.
(b) The solid triangle in 2^{nd} quadrant is scaled differentially w.r.t origin to its translucent version.
(c) The solid triangle in 3^{rd} quadrant is scaled uniformly w.r.t a fixed point 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

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

// 4) Reflection with respect to
//      a) x-axis
//      b) y-axis
//      c) origin
//      d) the line x=y

#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 = 900;
const int WINDOW_HEIGHT = 900;

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

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

void myInit();
void myDisplay();

void printAxes();
ld multiply(vector<ld> a, vector<ll> b);
vector<ld> multiply(vector<vector<ld>> a, vector<ll> b);

enum ReflectionType {X_AXIS, Y_AXIS, ORIGIN, X_EQUALS_Y_LINE};

vector<ll> getHomogeneousPointCoords(pair<ll,ll> point, ll h=1);
vector<ld> reflect(vector<ll> &point_matrix, ReflectionType type=ORIGIN);
pair<ld,ld> getPoint(vector<ld> point_matrix);
```

```
void reflectShape();


int main(int argc,char* argv[]) {
    glutInit(&argc,argv);
    glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
    glutInitWindowSize(WINDOW_WIDTH,WINDOW_HEIGHT);
    glutCreateWindow("2D - Reflection");
    glutDisplayFunc(myDisplay);
    myInit();
    glutMainLoop();
    return 1;
}

void myInit() {
    glClearColor(1.0,1.0,1.0,0.0);
    glColor3f(0.0f,0.0f,0.0f);
    glPointSize(2.0);
    glMatrixMode(GL_PROJECTION);
    glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
    glEnable( GL_BLEND );
    glLoadIdentity();
    gluOrtho2D(X_MIN,X_MAX,Y_MIN,Y_MAX);
}

void myDisplay() {
    glClear(GL_COLOR_BUFFER_BIT);

    printAxes();
    reflectShape();

    glFlush();
}

void printAxes() {
    glBegin(GL_LINES);

    glColor3f(1.0f,0.0f,0.0f);
    glVertex2d(X_MIN,0);
    glVertex2d(X_MAX,0);

    glColor3f(1.0f,0.0f,0.0f);
    glVertex2d(0,Y_MIN);
    glVertex2d(0,Y_MAX);

    for(ll i=X_MIN;i<X_MAX;i+=STEP) {
        glVertex2d(i,-0.3*STEP);
        glVertex2d(i,0.3*STEP);
    }

    for(ll i=Y_MIN;i<Y_MAX;i+=STEP) {
        glVertex2d(-0.3*STEP,i);
        glVertex2d(0.3*STEP,i);
    }

    glEnd();
}

void reflectShape() {

    vector<pair<ll,ll>> shape;

    //Plot original shape;

    shape = {{20,100}, {40,80}, {60,100}, {40,120}};

    glBegin(GL_POLYGON);
    glColor4f(0.0f,1.0f,0.7f,1.0f);

    for(auto point : shape) {
        glVertex2d(point.X,point.Y);
    }

    glEnd();

    //Reflect shape along x-axis;

    glBegin(GL_POLYGON);
    glColor4f(0.0f,0.1f,0.7f,0.4f);

    for(auto point : shape) {
        vector<ll> point_matrix = getHomogeneousPointCoords(point);
        pair<ld,ld> reflected_point = getPoint(reflect(point_matrix, ↩
            X_AXIS));

        glVertex2d(reflected_point.X,reflected_point.Y);
    }

    glEnd();

    //Reflect shape along y-axis;

    glBegin(GL_POLYGON);
    glColor4f(0.0f,0.1f,0.7f,0.4f);

    for(auto point : shape) {
        vector<ll> point_matrix = getHomogeneousPointCoords(point);
        pair<ld,ld> reflected_point = getPoint(reflect(point_matrix, ↩
           Y_AXIS));

        glVertex2d(reflected_point.X,reflected_point.Y);
    }

    glEnd();

    //Reflect shape along origin;

    glBegin(GL_POLYGON);
    glColor4f(0.0f,0.1f,0.7f,0.4f);

    for(auto point : shape) {
        vector<ll> point_matrix = getHomogeneousPointCoords(point);
        pair<ld,ld> reflected_point = getPoint(reflect(point_matrix, ↩
           ORIGIN));

        glVertex2d(reflected_point.X,reflected_point.Y);
    }

    glEnd();

    //Reflect shape along x=y line;

    glBegin(GL_LINES);
    glColor4f(0.0f,1.0f,0.0f,0.4f);

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

    glEnd();

    glBegin(GL_POLYGON);
    glColor4f(0.0f,0.1f,0.7f,0.4f);

    for(auto point : shape) {
        vector<ll> point_matrix = getHomogeneousPointCoords(point);
        pair<ld,ld> reflected_point = getPoint(reflect(point_matrix, ↩
           X_EQUALS_Y_LINE));

        glVertex2d(reflected_point.X,reflected_point.Y);
    }

    glEnd();
}

pair<ld,ld> getPoint(vector<ld> point_matrix) {
    ll h = point_matrix[2];
    ld x = point_matrix[0];
    ld y = point_matrix[1];

    return {x/h,y/h};
}

vector<ll> getHomogeneousPointCoords(pair<ll,ll> point, ll h) {
    vector<ll> point_matrix;
    point_matrix.push_back(h*point.first);
    point_matrix.push_back(h*point.second);
    point_matrix.push_back(h);
    return point_matrix;
}

vector<ld> reflect(vector<ll> &point_matrix, 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 multiply(reflection_matrix, point_matrix);
}

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

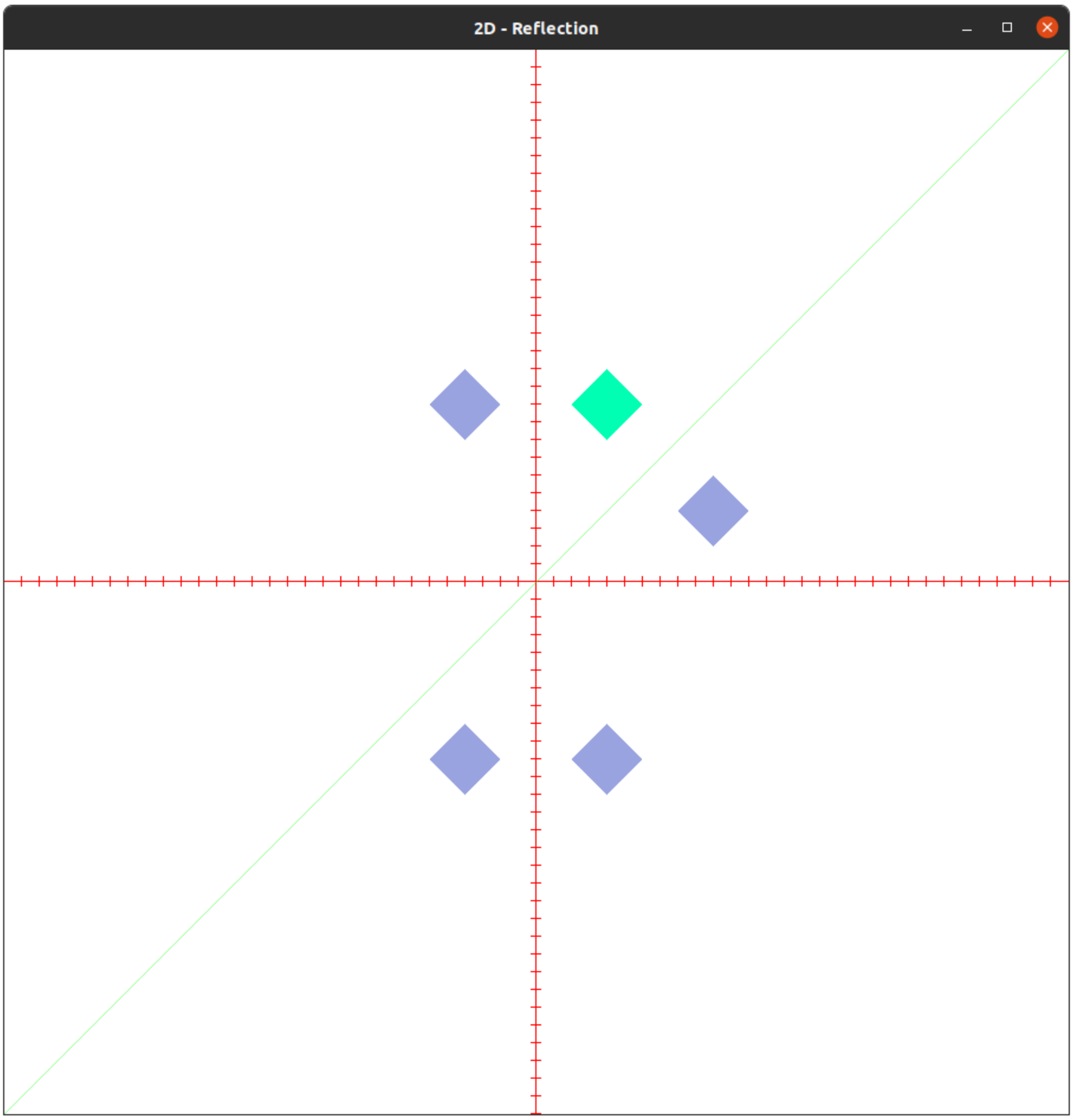

Figure 4: (a) The green diamond is reflected w.r.t x-axis to the blue one in 4^{th} quadrant.
(b) The green diamond is reflected w.r.t y-axis to the blue one in 2^{nd} quadrant.
(c) The green diamond is reflected w.r.t origin to the blue one in 3^{rd} quadrant.
(d) The green diamond is reflected w.r.t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

PROGRAM - 05

Shearing

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

// 5) Shearing
//     a) x-direction shear
//     b) y-direction shear

#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 = 900;
const int WINDOW_HEIGHT = 900;

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

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

void myInit();
void myDisplay();

void printAxes();
ld multiply(vector<ld> a, vector<ll> b);
vector<ld> multiply(vector<vector<ld>> a, vector<ll> b);

enum ShearType {X_SHEAR, Y_SHEAR};

vector<ll> getHomogeneousPointCoords(pair<ll,ll> point, ll h=1);
vector<ld> shear(vector<ll> &point_matrix, ShearType type=X_SHEAR, ld sh=1, ld refLine=0);
pair<ld,ld> getPoint(vector<ld> point_matrix);
void shearShape();
```

```


int main(int argc,char* argv[]) {
    glutInit(&argc,argv);
    glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
    glutInitWindowSize(WINDOW_WIDTH,WINDOW_HEIGHT);
    glutCreateWindow("2D - Shearing");
    glutDisplayFunc(myDisplay);
    myInit();
    glutMainLoop();
    return 1;
}

void myInit() {
    glClearColor(1.0,1.0,1.0,0.0);
    glColor3f(0.0f,0.0f,0.0f);
    glPointSize(2.0);
    glMatrixMode(GL_PROJECTION);
    glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
    glEnable( GL_BLEND );
    glLoadIdentity();
    gluOrtho2D(X_MIN,X_MAX,Y_MIN,Y_MAX);
}

void myDisplay() {
    glClear(GL_COLOR_BUFFER_BIT);

    printAxes();
    shearShape();

    glFlush();
}

void printAxes() {
    glBegin(GL_LINES);

    glColor3f(1.0f,0.0f,0.0f);
    glVertex2d(X_MIN,0);
    glVertex2d(X_MAX,0);

    glColor3f(1.0f,0.0f,0.0f);
    glVertex2d(0,Y_MIN);
    glVertex2d(0,Y_MAX);

    for(ll i=X_MIN;i<X_MAX;i+=STEP) {
        glVertex2d(i,-0.3*STEP);
        glVertex2d(i,0.3*STEP);
    }

    for(ll i=Y_MIN;i<Y_MAX;i+=STEP) {
        glVertex2d(-0.3*STEP,i);
        glVertex2d(0.3*STEP,i);
    }

    glEnd();
}

void shearShape() {

    vector<pair<ll,ll>> shape;

    //QUAD - 1

    //Plot original shape;

    shape = {{0,0}, {40,0}, {40,40}, {0,40}};

    glBegin(GL_POLYGON);
    glColor4f(0.7f,0.5f,1.0f,0.6f);

    for(auto point : shape) {
        glVertex2d(point.X,point.Y);
    }

    glEnd();

    //Shear the shape w.r.t x-axis;

    glBegin(GL_POLYGON);
    glColor4f(0.0f,0.1f,0.7f,0.4f);

    for(auto point : shape) {
        vector<ll> point_matrix = getHomogeneousPointCoords(point);
        pair<ld,ld> sheared_point = getPoint(shear(point_matrix, X_SHEAR, 3));

        glVertex2d(sheared_point.X,sheared_point.Y);
    }

    glEnd();

    //Shear the shape w.r.t x-axis with y_ref;

    glBegin(GL_POLYGON);
    glColor4f(0.0f,0.1f,0.7f,0.4f);

    for(auto point : shape) {
        vector<ll> point_matrix = getHomogeneousPointCoords(point);
        pair<ld,ld> sheared_point = getPoint(shear(point_matrix, X_SHEAR, 3, -30));

        glVertex2d(sheared_point.X,sheared_point.Y);
    }

    glEnd();

    //Shear the shape w.r.t y-axis;

    glBegin(GL_POLYGON);
    glColor4f(0.502f,0.0f,0.502f,0.6f);

    for(auto point : shape) {
        vector<ll> point_matrix = getHomogeneousPointCoords(point);
        pair<ld,ld> sheared_point = getPoint(shear(point_matrix, Y_SHEAR, 3));

        glVertex2d(sheared_point.X,sheared_point.Y);
    }

    glEnd();

    //Shear the shape w.r.t y-axis with x_ref;

    glBegin(GL_POLYGON);
    glColor4f(0.502f,0.0f,0.502f,0.6f);

    for(auto point : shape) {
        vector<ll> point_matrix = getHomogeneousPointCoords(point);
        pair<ld,ld> sheared_point = getPoint(shear(point_matrix, Y_SHEAR, 3, -30));

        glVertex2d(sheared_point.X,sheared_point.Y);
    }

    glEnd();

    //QUAD - 2

    //Plot original shape;

    shape = {{0,0}, {-40,0}, {-40,40}, {0,40}};

    glBegin(GL_POLYGON);
    glColor4f(0.486f,0.988f,0.000f,0.4f);

    for(auto point : shape) {
        glVertex2d(point.X,point.Y);
    }

    glEnd();

    //Shear the shape w.r.t x-axis;

    glBegin(GL_POLYGON);
    glColor4f(0.596f,0.984f,0.596f,0.7f);

    for(auto point : shape) {
        vector<ll> point_matrix = getHomogeneousPointCoords(point);
        pair<ld,ld> sheared_point = getPoint(shear(point_matrix, X_SHEAR, -3));

        glVertex2d(sheared_point.X,sheared_point.Y);
    }

    glEnd();

    //Shear the shape w.r.t x-axis with y_ref;

    glBegin(GL_POLYGON);
    glColor4f(0.596f,0.984f,0.596f,0.7f);

    for(auto point : shape) {
        vector<ll> point_matrix = getHomogeneousPointCoords(point);
        pair<ld,ld> sheared_point = getPoint(shear(point_matrix, X_SHEAR, -3, -30));

        glVertex2d(sheared_point.X,sheared_point.Y);
    }

    glEnd();

    //Shear the shape w.r.t y-axis;

    glBegin(GL_POLYGON);
    glColor4f(0.678f,1.0f,0.184f,0.7f);

    for(auto point : shape) {
        vector<ll> point_matrix = getHomogeneousPointCoords(point);
        pair<ld,ld> sheared_point = getPoint(shear(point_matrix, Y_SHEAR, -3));

        glVertex2d(sheared_point.X,sheared_point.Y);
    }

    glEnd();

    //Shear the shape w.r.t y-axis with x_ref;

    glBegin(GL_POLYGON);
    glColor4f(0.678f,1.0f,0.184f,0.7f);

    for(auto point : shape) {
        vector<ll> point_matrix = getHomogeneousPointCoords(point);
        pair<ld,ld> sheared_point = getPoint(shear(point_matrix, Y_SHEAR, -3, 30));

        glVertex2d(sheared_point.X,sheared_point.Y);
    }

    glEnd();

    //QUAD - 3

    //Plot original shape;

    shape = {{0,0}, {-40,0}, {-40,-40}, {0,-40}};

    glBegin(GL_POLYGON);
    glColor4f(0.824f,0.412f,0.118f,0.6f);

    for(auto point : shape) {
        glVertex2d(point.X,point.Y);
    }

    glEnd();

    //Shear the shape w.r.t x-axis;

    glBegin(GL_POLYGON);
    glColor4f(0.722f,0.525f,0.043f,0.7f);

    for(auto point : shape) {
        vector<ll> point_matrix = getHomogeneousPointCoords(point);
        pair<ld,ld> sheared_point = getPoint(shear(point_matrix, X_SHEAR, 3));

        glVertex2d(sheared_point.X,sheared_point.Y);
    }

    glEnd();

    //Shear the shape w.r.t x-axis with y_ref;

    glBegin(GL_POLYGON);
    glColor4f(0.722f,0.525f,0.043f,0.7f);

    for(auto point : shape) {
        vector<ll> point_matrix = getHomogeneousPointCoords(point);
        pair<ld,ld> sheared_point = getPoint(shear(point_matrix, X_SHEAR, ↩
           3, 30));

        glVertex2d(sheared_point.X,sheared_point.Y);
    }

    glEnd();

    //Shear the shape w.r.t y-axis;

    glBegin(GL_POLYGON);
    glColor4f(0.627f,0.322f,0.176f,0.6f);

    for(auto point : shape) {
        vector<ll> point_matrix = getHomogeneousPointCoords(point);
        pair<ld,ld> sheared_point = getPoint(shear(point_matrix, Y_SHEAR, ↩
           3));

        glVertex2d(sheared_point.X,sheared_point.Y);
    }

    glEnd();

    //Shear the shape w.r.t y-axis with x_ref;

    glBegin(GL_POLYGON);
    glColor4f(0.627f,0.322f,0.176f,0.6f);

    for(auto point : shape) {
        vector<ll> point_matrix = getHomogeneousPointCoords(point);
        pair<ld,ld> sheared_point = getPoint(shear(point_matrix, Y_SHEAR, ↩
           3, 30));

        glVertex2d(sheared_point.X,sheared_point.Y);
    }

    glEnd();

    //QUAD - 4

    //Plot original shape;

    shape = {{0,0}, {40,0}, {40,-40}, {0,-40}};

    glBegin(GL_POLYGON);
    glColor4f(0.498f,1.0f,0.831f,0.6f);

    for(auto point : shape) {
        glVertex2d(point.X,point.Y);
    }

    glEnd();

    //Shear the shape w.r.t x-axis;

    glBegin(GL_POLYGON);
    glColor4f(0.0f,0.545f,0.545f,0.6f);

    for(auto point : shape) {
        vector<ll> point_matrix = getHomogeneousPointCoords(point);
        pair<ld,ld> sheared_point = getPoint(shear(point_matrix, X_SHEAR, -3));

        glVertex2d(sheared_point.X,sheared_point.Y);
    }

    glEnd();

    //Shear the shape w.r.t x-axis with y_ref;

    glBegin(GL_POLYGON);
    glColor4f(0.0f,0.545f,0.545f,0.6f);

    for(auto point : shape) {
        vector<ll> point_matrix = getHomogeneousPointCoords(point);
        pair<ld,ld> sheared_point = getPoint(shear(point_matrix, X_SHEAR, -3, 30));

        glVertex2d(sheared_point.X,sheared_point.Y);
    }

    glEnd();

    //Shear the shape w.r.t y-axis;

    glBegin(GL_POLYGON);
    glColor4f(0.282f,0.820f,0.800f,0.6f);

    for(auto point : shape) {
        vector<ll> point_matrix = getHomogeneousPointCoords(point);
        pair<ld,ld> sheared_point = getPoint(shear(point_matrix, Y_SHEAR, -3));

        glVertex2d(sheared_point.X,sheared_point.Y);
    }

    glEnd();

    //Shear the shape w.r.t y-axis with x_ref;

    glBegin(GL_POLYGON);
    glColor4f(0.282f,0.820f,0.800f,0.6f);

    for(auto point : shape) {
        vector<ll> point_matrix = getHomogeneousPointCoords(point);
        pair<ld,ld> sheared_point = getPoint(shear(point_matrix, Y_SHEAR, -3, -30));

        glVertex2d(sheared_point.X,sheared_point.Y);
    }

    glEnd();

}

pair<ld,ld> getPoint(vector<ld> point_matrix) {
    ll h = point_matrix[2];
    ld x = point_matrix[0];
    ld y = point_matrix[1];

    return {x/h,y/h};
}

vector<ll> getHomogeneousPointCoords(pair<ll,ll> point, ll h) {
    vector<ll> point_matrix;
    point_matrix.push_back(h*point.first);
    point_matrix.push_back(h*point.second);
    point_matrix.push_back(h);
    return point_matrix;
}

vector<ld> shear(vector<ll> &point_matrix, 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 multiply(shear_matrix, point_matrix);
}

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

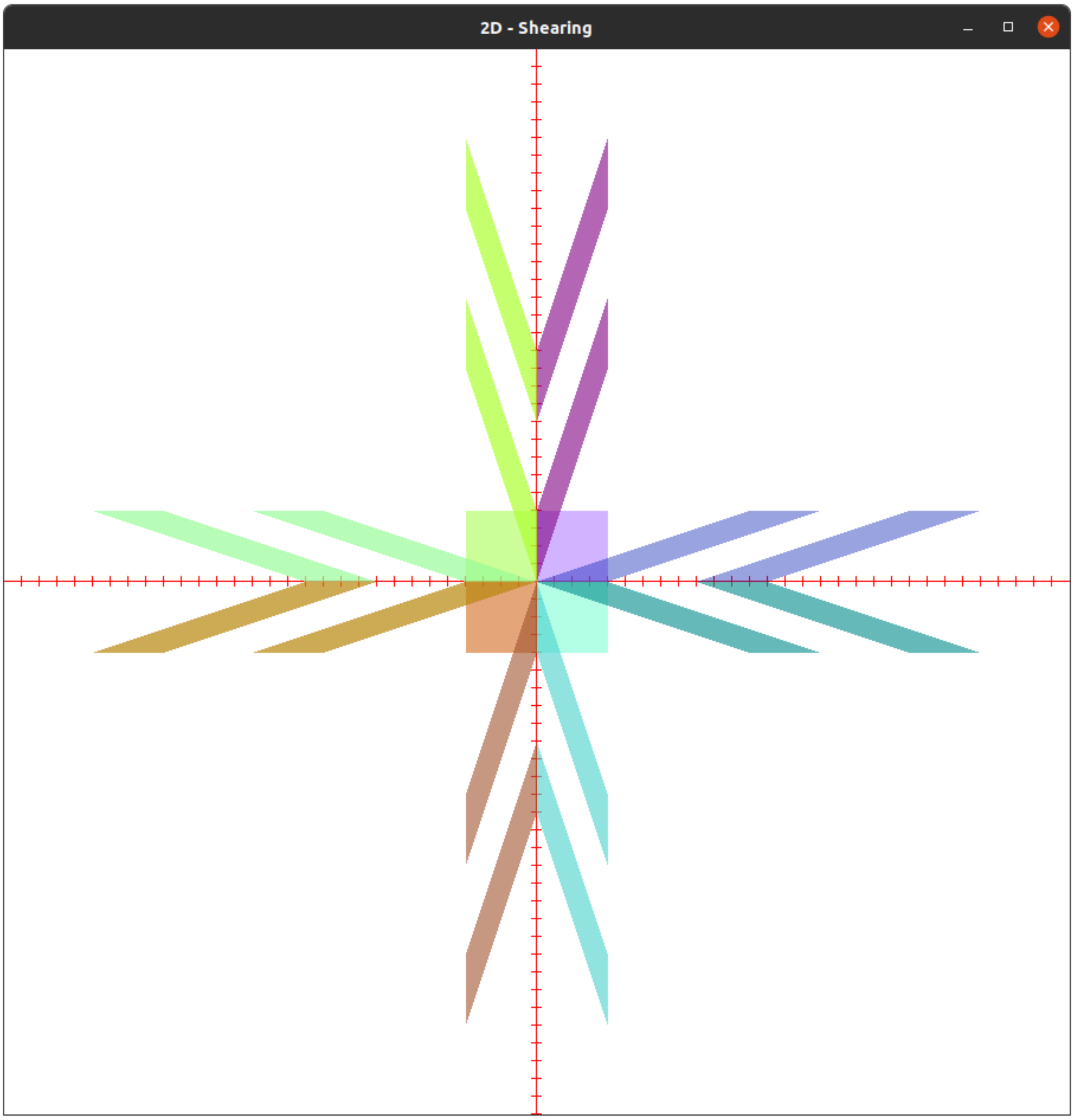

Figure 5: In each quadrant the sqaure is (a) x-sheared. (b) x-sheared w.r.t a reference line. (c) y-sheared. (d) y-sheared w.r.t a reference line.

RESULT

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