DEERWALK INSTITUTE OF TECHNOLOGY

Tribhuvan University

Faculties of Computer Science

Bachelors of Science in Computer Science and Information Technology (BSc. CSIT)

Course: Computer Graphics (CSC209)

Year/Semester: II/III

A Lab report on:

Implementation of 2D Transformation

Submitted by:
Name: Arun Mainali
Roll: 1307

Submitted to:
Binod Sitaula
Department of Computer Science

❖ LAB 5

OBJECTIVE:

Write a program in any high-level language to draw a polygon of your own choice and perform the following transformation:

a. Translation
b. Rotation
c. Scaling
d. Reflection
e. Shear

THEORY:

2D Transformation:

Transformation means changing some graphics into something else by applying rules. We can have various types of transformations such as translation, scaling up or down, rotation, shearing, etc. When a transformation takes place on a 2D plane, it is called 2D transformation.

Transformations play an important role in computer graphics to reposition the graphics on the screen and change their size or orientation.

Homogenous Coordinates

To perform a sequence of transformation such as translation followed by rotation and scaling, we need to follow a sequential process −

- Translate the coordinates,
- Rotate the translated coordinates, and then
- Scale the rotated coordinates to complete the composite transformation.

Translation
A translation moves an object to a different position on the screen. You can translate a point in 2D by adding translation coordinate (t_x, t_y) to the original coordinate X,YX,Y to get the new coordinate X′,Y′X′,Y′.

From the above figure, you can write that −

$$X' = X + t_x$$

$$Y' = Y + t_y$$

The pair (t_x, t_y) is called the translation vector or shift vector. The above equations can also be represented using the column vectors.

$$P=[X][Y]P=[X][Y]\ p' = [X'][Y'][X'][Y']T = [tx][ty][tx][ty]$$

We can write it as −

$$P' = P + T$$

Rotation:

In rotation, we rotate the object at particular angle θ thetatheta from its origin. From the following figure, we can see that the point PX,YX,Y is located at angle ϕ from the horizontal X coordinate with distance r from the origin.

Let us suppose you want to rotate it at the angle θ. After rotating it to a new location, you will get a new point P' X',Y'X',Y'.

Using standard trigonometric the original coordinate of point PX,YX,Y can be represented as −

$$X=r\cos\phi.....(1)X=r\cos\phi.....(1)$$

$$Y=r\sin\phi.....(2)Y=r\sin\phi.....(2)$$

Same way we can represent the point P' X',Y'X',Y' as −

$$x'=r\cos(\phi+\theta)=r\cos\phi\cos\theta-r\sin\phi\sin\theta.....(3)x'=r\cos(\phi+\theta)=r\cos\phi\cos\theta-r\sin\phi\sin\theta.....(3)$$

$$y'=r\sin(\phi+\theta)=r\cos\phi\sin\theta+r\sin\phi\cos\theta.....(4)y'=r\sin(\phi+\theta)=r\cos\phi\sin\theta+r\sin\phi\cos\theta.....(4)$$

Substituting equation 11 & 22 in 33 & 44 respectively, we will get

$$x'=x\cos\theta-y\sin\theta x'=x\cos\theta-y\sin\theta$$

$$y'=x\sin\theta+y\cos\theta y'=x\sin\theta+y\cos\theta$$

Representing the above equation in matrix form,

$$[X'Y']=[XY][\cos\theta-\sin\theta\sin\theta\cos\theta]OR[X'Y']=[XY][\cos\theta\sin\theta-\sin\theta\cos\theta]OR$$

$$P' = P \cdot R$$

Where R is the rotation matrix

$$R=[\cos\theta-\sin\theta\sin\theta\cos\theta]R=[\cos\theta\sin\theta-\sin\theta\cos\theta]$$

The rotation angle can be positive and negative.

For positive rotation angle, we can use the above rotation matrix. However, for negative angle rotation, the matrix will change as shown below −

$$R=[\cos(-\theta)-\sin(-\theta)\sin(-\theta)\cos(-\theta)]R=[\cos(-\theta)\sin(-\theta)-\sin(-\theta)\cos(-\theta)]$$

$$=[\cos\theta\sin\theta-\sin\theta\cos\theta](\because\cos(-\theta)=\cos\theta and \sin(-\theta)=-\sin\theta)=[\cos\theta-\sin\theta\sin\theta\cos\theta]$$

$$(\because\cos(-\theta)=\cos\theta and \sin(-\theta)=-\sin\theta)$$

Scaling:

To change the size of an object, scaling transformation is used. In the scaling process, you either expand or compress the dimensions of the object. Scaling can be achieved by multiplying the original coordinates of the object with the scaling factor to get the desired result.

Let us assume that the original coordinates are X,YX,Y, the scaling factors are (S_X, S_Y), and the produced coordinates are X',Y'X',Y'. This can be mathematically represented as shown below −

$$X' = X \cdot S_X \text{ and } Y' = Y \cdot S_Y$$

The scaling factor S_X, S_Y scales the object in X and Y direction respectively. The above equations can also be represented in matrix form as below −

$$(X'Y')=(XY)[Sx00Sy](X'Y')=(XY)[Sx00Sy]$$

OR

$$P' = P \cdot S$$

Where S is the scaling matrix. The scaling process is shown in the following figure.

If we provide values less than 1 to the scaling factor S, then we can reduce the size of the object. If we provide values greater than 1, then we can increase the size of the object.

Reflection

Reflection is the mirror image of original object. In other words, we can say that it is a rotation operation with 180°. In reflection transformation, the size of the object does not change.

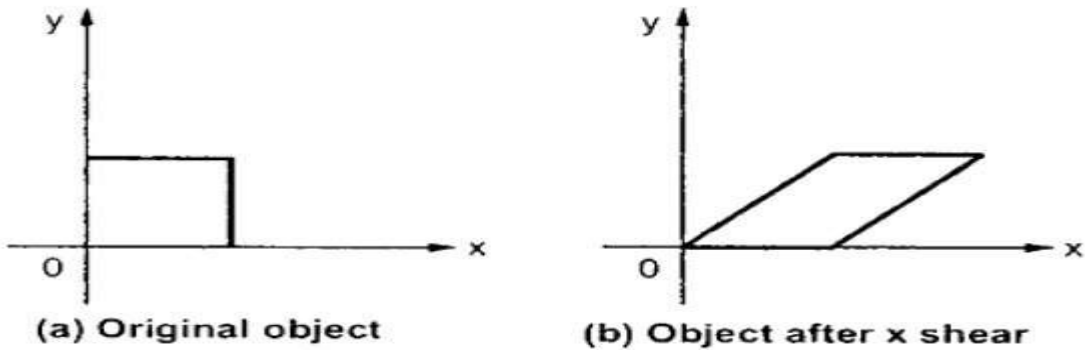
The following figures show reflections with respect to X and Y axes, and about the origin respectively.

Shear:

A transformation that slants the shape of an object is called the shear transformation. There are two shear transformations X-Shear and Y-Shear. One shifts X coordinates values and other shifts Y coordinate values. However; in both the cases only one coordinate changes its coordinates and other preserves its values. Shearing is also termed as Skewing.

X-Shear:

The X-Shear preserves the Y coordinate and changes are made to X coordinates, which causes the vertical lines to tilt right or left as shown in below figure.



(a) Original object (b) Object after x shear

The transformation matrix for X-Shear can be represented as −

Xsh=[1shx0010001]Xsh=[1shx0010001]

$$Y' = Y + Sh_y \cdot X$$

$$X' = X$$

Y-Shear:

The Y-Shear preserves the X coordinates and changes the Y coordinates which causes the horizontal lines to transform into lines which slopes up or down as shown in the following figure.

The Y-Shear can be represented in matrix from as −

Ysh[1shy0010001]Ysh[100shy10001]

$$X' = X + Sh_x \cdot Y$$

$$Y' = Y$$

Composite Transformation:

If a transformation of the plane T1 is followed by a second plane transformation T2, then the result itself may be represented by a single transformation T which is the composition of T1 and T2 taken in that order. This is written as T = T1·T2.

Composite transformation can be achieved by concatenation of transformation matrices to obtain a combined transformation matrix.

A combined matrix −

[T][X] = [X] [T1] [T2] [T3] [T4] …. [Tn]

Where [Ti] is any combination of

- Translation
- Scaling
- Shearing
- Rotation
- Reflection

The change in the order of transformation would lead to different results, as in general matrix multiplication is not cumulative, that is [A] . [B] ≠ [B] . [A] and the order of multiplication. The basic purpose of composing transformations is to gain efficiency by applying a single composed transformation to a point, rather than applying a series of transformation, one after another.

For example, to rotate an object about an arbitrary point (X_p, Y_p), we have to carry out three steps −

- Translate point (X_p, Y_p) to the origin.
- Rotate it about the origin.

- Finally, translate the center of rotation back where it belonged.

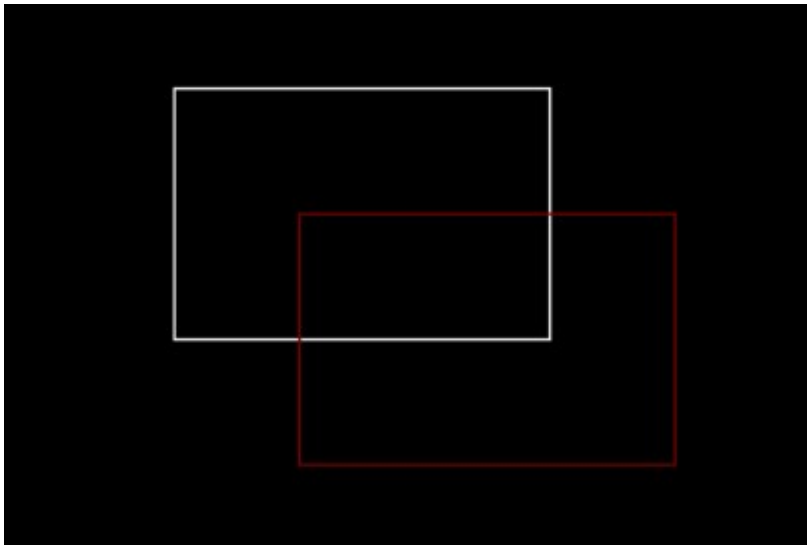
PROGRAM CODE

Translation

```
#include <graphics.h>
#include <conio.h>
#include <iostream>
using namespace std;
void drawRectangle(int x1, int y1, int x2, int y2) {
   rectangle(x1, y1, x2, y2);
}
void translateRectangle(int &x1, int &y1, int &x2, int &y2, int tx, int ty) {
   x1 += tx; y1 += ty;
   x2 += tx; y2 += ty;
}
int main() {
   int gd = DETECT, gm;
   initgraph(&gd, &gm, "C:\\Turboc3\\BGI");
   int x1 = 150, y1 = 100, x2 = 300, y2 = 200;
   int tx = 50, ty = 50; // Translation factors
   // Draw original rectangle
   setcolor(WHITE);
   drawRectangle(x1, y1, x2, y2);
   // Translate the rectangle
   translateRectangle(x1, y1, x2, y2, tx, ty);
   // Draw translated rectangle
   setcolor(RED);
   drawRectangle(x1, y1, x2, y2);
   getch();
```

```
    closegraph();
    return 0;
}
```

Output:



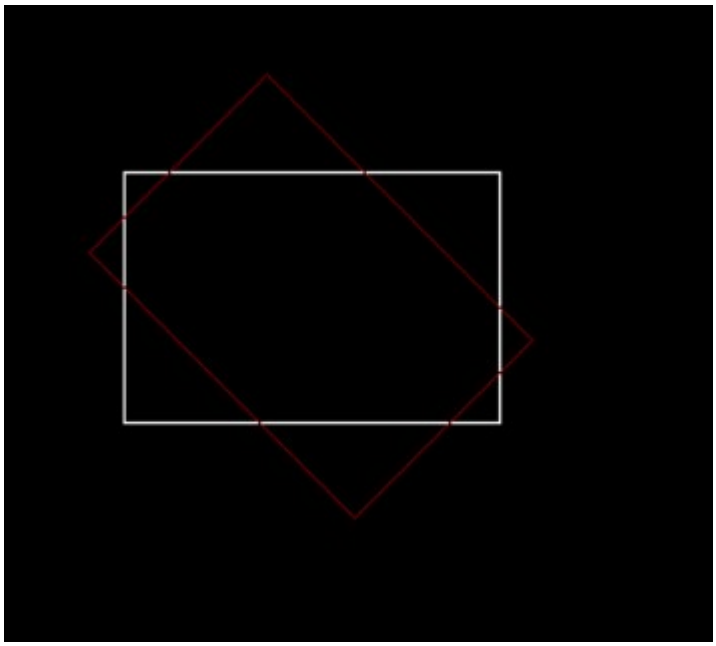
Rotation

```
#include <graphics.h>
#include <conio.h>
#include <iostream>
#include <math.h>

using namespace std;

void rotatePoint(int &x, int &y, int cx, int cy, float angle) {
    float rad = angle * (M_PI / 180.0);
    int newX = cx + (x - cx) * cos(rad) - (y - cy) * sin(rad);
    int newY = cy + (x - cx) * sin(rad) + (y - cy) * cos(rad);
    x = newX;
    y = newY;
}
```

```
void rotateRectangle(int x1, int y1, int x2, int y2, float angle) {
    int cx = (x1 + x2) / 2, cy = (y1 + y2) / 2;
    int x3 = x2, y3 = y1;
    int x4 = x1, y4 = y2;

    rotatePoint(x1, y1, cx, cy, angle);
    rotatePoint(x2, y2, cx, cy, angle);
    rotatePoint(x3, y3, cx, cy, angle);
    rotatePoint(x4, y4, cx, cy, angle);

    line(x1, y1, x3, y3);
    line(x3, y3, x2, y2);
    line(x2, y2, x4, y4);
    line(x4, y4, x1, y1);
}

int main() {
    int gd = DETECT, gm;
    initgraph(&gd, &gm, "");

    int x1 = 150, y1 = 100, x2 = 300, y2 = 200;
    float angle = 45;

    setcolor(WHITE);
    rectangle(x1, y1, x2, y2);

    setcolor(RED);
    rotateRectangle(x1, y1, x2, y2, angle);
```

```
    getch();
    closegraph();
    return 0;
}
```

Output:

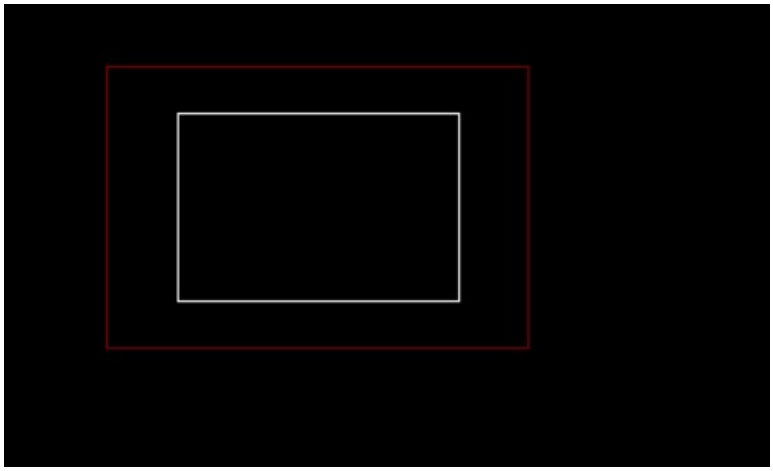


Scaling:

```
#include <graphics.h>
#include <conio.h>
#include <iostream>

using namespace std;

void scaleRectangle(int &x1, int &y1, int &x2, int &y2, float sx, float sy) {
    int cx = (x1 + x2) / 2, cy = (y1 + y2) / 2;
    x1 = cx + (x1 - cx) * sx;
```

```
    y1 = cy + (y1 - cy) * sy;
    x2 = cx + (x2 - cx) * sx;
    y2 = cy + (y2 - cy) * sy;
}

int main() {
    int gd = DETECT, gm;
    initgraph(&gd, &gm, "");

    int x1 = 150, y1 = 100, x2 = 300, y2 = 200;
    float sx = 1.5, sy = 1.5;

    setcolor(WHITE);
    rectangle(x1, y1, x2, y2);

    scaleRectangle(x1, y1, x2, y2, sx, sy);
    setcolor(RED);
    rectangle(x1, y1, x2, y2);

    getch();
    closegraph();
    return 0;
}
```

Output:

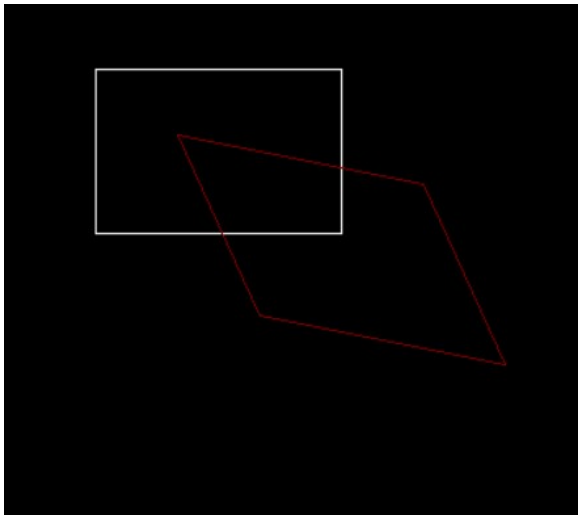


Shearing

```
#include <graphics.h>
#include <conio.h>
#include <iostream>

using namespace std;

void shearRectangle(int &x1, int &y1, int &x2, int &y2, float shx, float shy) {
    int x3 = x2, y3 = y1;
    int x4 = x1, y4 = y2;

    x1 += y1 * shx;
    x2 += y2 * shx;
    x3 += y3 * shx;
    x4 += y4 * shx;

    y1 += x1 * shy;
    y2 += x2 * shy;
    y3 += x3 * shy;
    y4 += x4 * shy;
```

```
    line(x1, y1, x3, y3);
    line(x3, y3, x2, y2);
    line(x2, y2, x4, y4);
    line(x4, y4, x1, y1);
}

int main() {
    int gd = DETECT, gm;
    initgraph(&gd, &gm, "");

    int x1 = 150, y1 = 100, x2 = 300, y2 = 200;
    float shx = 0.5, shy = 0.2;

    setcolor(WHITE);
    rectangle(x1, y1, x2, y2);

    setcolor(RED);
    shearRectangle(x1, y1, x2, y2, shx, shy);

    getch();
    closegraph();
    return 0;
}
```

Output:

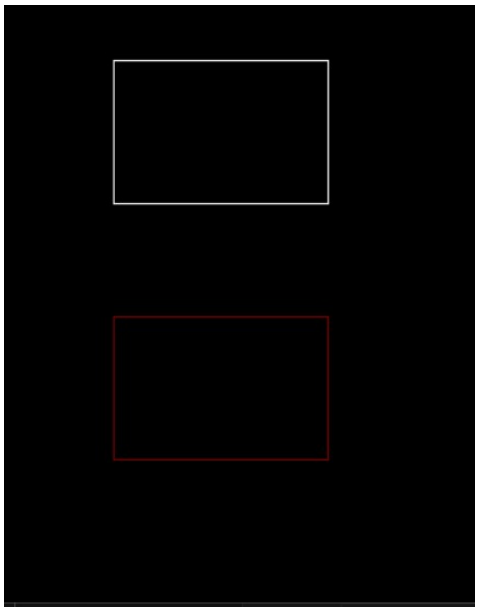


Reflection:

```
#include <graphics.h>
#include <conio.h>
#include <iostream>

using namespace std;

void reflectRectangle(int &x1, int &y1, int &x2, int &y2, bool reflectX, bool reflectY) {
    if (reflectX) {
        y1 = getmaxy() - y1;
        y2 = getmaxy() - y2;
    }
    if (reflectY) {
        x1 = getmaxx() - x1;
        x2 = getmaxx() - x2;
    }
}

int main() {
```

```
    int gd = DETECT, gm;
    initgraph(&gd, &gm, "");

    int x1 = 150, y1 = 100, x2 = 300, y2 = 200;

    setcolor(WHITE);
    rectangle(x1, y1, x2, y2);

    reflectRectangle(x1, y1, x2, y2, true, false);
    setcolor(RED);
    rectangle(x1, y1, x2, y2);

    getch();
    closegraph();
    return 0;
}
```

Output



CONCLUSION:

Hence, using C programming as the high-level language we were able to perform various types of 2d transformation like translation, rotation, scaling, shearing on the different polygons.