**DEERWALK INSTITUTE OF TECHNOLOGY**

**Tribhuvan University**

**Faculties of Computer Science**

**A logo of a sea creature

Description automatically generated**

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

**Course: Computer Graphics (CSC214)**

**Year/Semester: II/III**

**A Lab report on:**

**Implementation of 2D Transformation**

Submitted by:

Name: Parth Poudyal

Roll: 1317

Submitted to:

Binod Sitaula

Department of Computer Science

**Theory**

2D transformation:

Vectors in the Euclidean coordinate system can be transformed into something else using set of rules called Transformation. Transformation can be of multiple types ranging from Reflection, Rotation, Scaling, Translation, Shear, etc.

Homogeneous coordinate system:

When the coordinates in 2 dimensions are represented in 3 dimensions keeping the values of the z coordinate as 1 is known as homogeneous coordinate representation of 2 Dimensional vectors. This procedure is used to perform composite transformations for a set of vertices lying in the coordinate plane.

E.g. is the homogeneous coordinate representation for vertices (2,1) , (3,2) and (5,4)

Translation:

Translation is the shifting of position of objects in a screen.

Translation can be performed as follows:

X’ = X + tx

Y’ = Y + ty

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

Rotation

Rotation is the process of rotating an object about a pivot point (typically the origin) by a specified angle . For the new coordinates for X’ and Y’ on rotation about theta can be represented as:

Reflection

Reflection is the process of flipping a coordinate about an axis of rotation. 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.

Reflection about X-axis:

Reflection about Y-axis:

Scaling

Scaling is a process of increment or decrement of dimensions of an object by a certain ratio. Scaling can be achieved by multiplying the coordinates with a scaling factor to get the desired result.

For Scaling:

Here are scaling factors respectively for x and y directions. For un-uniform scaling, the values of

Shearing

Shearing transforms an object such that the shape changes, but the area remains unchanged. It can be applied horizontally or vertically.

For vertical articles under Shear:

In X direction (X – shear):

In Y direction (Y – Shear):

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 (Xp, Yp), we have to carry out three steps −

* Translate point (Xp, Yp) to the origin.
* Rotate it about the origin.
* Finally, translate the center of rotation back where it belonged.

**Programs**

**Translation :**

#include <iostream>

#include <graphics.h>

#include <conio.h>

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

    getch();

A black background with white and red rectangles

AI-generated content may be incorrect.    closegraph();

    return 0;

}

**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;

A white rectangle with red and black lines on a black background

AI-generated content may be incorrect.    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;}

**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;

}

A white rectangle on a black background

AI-generated content may be incorrect.

**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);

A white and red rectangles on a black background

AI-generated content may be incorrect.    shearRectangle(x1, y1, x2, y2, shx, shy);

    getch();

    closegraph();

    return 0;

}

**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)**

**{**

A screenshot of a black screen

AI-generated content may be incorrect.**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;**

**}**

**Conclusion**

From this project I learned about performing 2D transformations using graphics.h in C.