AISSMS
COLLEGE OF ENGINEERING
Approved by AICTE, New Delhi, Recognized by
Govt. of Maharashtra, Affiliated to Savitribai Phule Pune University
and recognized 2(f) and 12(B) by UGC (Id.No. PU / PN/ Engg. / 093 (1992)
Accredited by NAAC with 'A+' Grade

Name :- Pratik Pingale *Class :- SE Comp. 1*

Roll no. :- 19CO056

Experiment B-7

Aim :

To write the C++ program to draw 2D objects and perform following basic transformations a)Scaling b)Traslation c)Rotation.

Algorithm:

Part a: Scaling

Step 1: Read n as number of vertices of the polygon.

Step 2: read x and y coordinates of all vertices in array x[n] and y[n].

Step 3: Drew polygon with these vertices.

Step 4: Make a 3x3 scaling matrix S as:

Sx 0 0

0 Sy 0

0 0 1

Step 5: For each point of the polygon:

(i) Make a 3x1 matrix P, where P[0][0] equals

to x coordinate of the point and P[1][0]

equal to y coordinate of the point and P[2][0]

equals to 1.

(ii) Multiply scaling matrix S with point

matrix P to get the new coordinate.

Step 6: Draw the polygon using new coordinates.

Step 7: End.

Part b: Translation

Step 1: Read n as number of vertices of polygon.

Step 2: read x and y coordinates of all vertices in array x[n] and y[n].

Step 3: Drew polygon with these vertices.

Step 4: Create 3x3 translation matrix T as:

1 0 Tx

0 1 Ty

0 1 1

Step 5: For each point of polygon:

(i) Make a 3x1 matrix P, where P[0][0] equals

to x coordinate of the point and P[1][0]

equal to y coordinate of the point and P[2][0]

equals to 1.

(ii)Calculate new co-ordinates multiply Translation matrix T with matrix P to get new co-ordinates.

Step 6: Draw polygon with new co-ordinates.

Step 7: End.

Part c: Rotation

Step 1: Read n as number of vertices of polygon.

Step 2: read x and y coordinates of all vertices in array x[n] and y[n].

Step 3: Drew polygon with these vertices.

Step 4: Create two rotation matrices R1 for Anti-clock wise rotation
And R2 for clockwise rotation.

$$R1 = \begin{vmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{vmatrix} \qquad R2 = \begin{vmatrix} \cos(\theta) & \sin(\theta) & 0 \\ -\sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{vmatrix}$$

Step 5: For each point of the polygon:

(i) Make a 3x1 matrix P, where P[0][0] equals
to x coordinate of the point and P[1][0]
equals to y coordinate of the point and P[2][0]
equals to 1.

(ii) To rotate polygon anti-clock wise multiply rotation matrix
R1 with matrix P and to rotate polygon clock-wise
multiply rotation matrix R2 with P to get new co-ordinates.

Step 6: Draw polygon with new vertices.

Step 7: End.

Program:

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

using namespace std;
class trans {
    int no;
    int mat1[10][10], transMatrix[3][3], scalMatrix[3][3];
    float rotMatrix[3][3];
    int tx, ty, sx, sy;
    float theta;
    public:
        void accept() {
            cout << "\nEnter Number of points:";
            cin >> no;
            for (int i = 0; i < no; i++) {
                cout << "\nEnter for Point " << i + 1 << ":";
                cout << "\nEnter X coOr:";
                cin >> mat1[i][0];
                cout << "\nEnter Y Coor:";
                cin >> mat1[i][1];
                mat1[i][2] = 1;
            }
        }
    void showMatrix() {
        cout << "\nMAtrix:\n";
        for (int i = 0; i < no; i++) {
            for (int j = 0; j < 3; j++) {
                cout << "\t" << mat1[i][j];
            }
            cout << "\n";
        }
    }
    void draw() {
        int i;
        delay(100);
        setbkcolor(15);
        for (i = 0; i < no - 1; i++) {
            line1(mat1[i][0] + 100, mat1[i][1] + 100, mat1[i + 1][0] + 100,
mat1[i + 1][1] + 100);
        }
        line1(mat1[i][0] + 100, mat1[i][1] + 100, mat1[0][0] + 100, mat1[0][1] +
100);
    }
    void line1(int x1, int y1, int x2, int y2) {
        float dx = x2 - x1;
        float dy = y2 - y1;
        int length;
```

```
        if (abs(dx) > abs(dy))
            length = abs(dx);
        else
            length = abs(dy);
        float xinc = dx / length;
        float yinc = dy / length;
        float x = x1;
        float y = y1;
        int i = 0;
        while (i ≤ length) {
            putpixel(x, y, BLACK);
            x = x + xinc;
            y = y + yinc;
            i++;
        }
    }
    void createTrans() {
        cout << "\nEnter Tx: and Ty:";
        cin >> tx >> ty;
        transMatrix[2][0] = tx;
        transMatrix[2][1] = ty;
        transMatrix[0][0] = transMatrix[1][1] = transMatrix[2][2] = 1;
        transMatrix[1][0] = transMatrix[0][1] = transMatrix[1][2] = 0;
        cout << "\nTrans Matrix:\n";
        for (int i = 0; i < 3; i++) {
            for (int j = 0; j < 3; j++)
                cout << "\t" << transMatrix[i][j];
            cout << "\n";
        }
    }
    trans operator * (trans b) {
        trans c;
        c.no = no;
        for (int i = 0; i < no; i++) {
            for (int j = 0; j < 3; j++) {
                c.mat1[i][j] = 0;
            }
        }
        for (int i = 0; i < no; i++) {
            for (int j = 0; j < 3; j++) {
                for (int k = 0; k < 3; k++) {
                    c.mat1[i][j] = c.mat1[i][j] + mat1[i][k] * b.transMatrix[k]
[j];
                }
            }
        }
        return c;
    }
    void createScal() {
        cout << "\nEnter Sx and SY:";
        cin >> sx >> sy;
        scalMatrix[0][0] = sx;
        scalMatrix[1][1] = sy;
        scalMatrix[2][2] = 1;
```

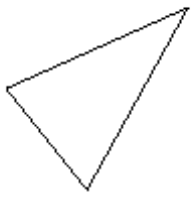
```
        scalMatrix[0][1] = scalMatrix[0][2] = scalMatrix[1][0] = scalMatrix[1][2]
= scalMatrix[2][0] =
            scalMatrix[2][1] = 0;
        cout << "\nScaling Matrix:\n";
        for (int i = 0; i < 3; i++) {
            for (int j = 0; j < 3; j++)
                cout << "\t" << scalMatrix[i][j];
            cout << "\n";
        }
    }
    trans operator + (trans b) {
        trans c;
        c.no = no;
        for (int i = 0; i < no; i++) {
            for (int j = 0; j < 3; j++) {
                c.mat1[i][j] = 0;
                for (int k = 0; k < 3; k++) {
                    c.mat1[i][j] = c.mat1[i][j] + mat1[i][k] * b.scalMatrix[k]
[j];
                }
            }
        }
        return c;
    }
    void createRota() {
        cout << "\nEnter The angle by which to be rotated:";
        cin >> theta;
        int choice;
        cout << "\n1.Anticlockwise\n2.Clockwise\nEnter Choice:";
        cin >> choice;
        rotMatrix[0][2] = rotMatrix[1][2] = rotMatrix[2][0] = rotMatrix[2][1] =
0;
        rotMatrix[2][2] = 1;
        rotMatrix[0][0] = rotMatrix[1][1] = cos(theta * M_PI / 180);
        if (choice == 1) {
            rotMatrix[0][1] = sin(theta * M_PI / 180);
            rotMatrix[1][0] = -sin(theta * M_PI / 180);
        } else {
            rotMatrix[0][1] = -sin(theta * M_PI / 180);
            rotMatrix[1][0] = sin(theta * M_PI / 180);
        }
        cout << "\nRota Matrix:\n";
        for (int i = 0; i < 3; i++) {
            for (int j = 0; j < 3; j++) {
                cout << "\t" << rotMatrix[i][j];
            }
            cout << "\n";
        }
    }
    trans operator - (trans b) {
        trans c;
        c.no = no;
        for (int i = 0; i < no; i++) {
            for (int j = 0; j < 3; j++) {
```

```
                c.mat1[i][j] = 0;
            }
        }
        for (int i = 0; i < no; i++) {
            for (int j = 0; j < 3; j++) {
                for (int k = 0; k < 3; k++) {
                    c.mat1[i][j] = c.mat1[i][j] + mat1[i][k] * b.rotMatrix[k][j];
                }
            }
        }
        return c;
    }
};
int main() {
    int gd = DETECT, gm;
    int choice = 0;
    trans a, b, c;
    a.accept();
    a.showMatrix();
    initgraph( & gd, & gm, NULL);
    a.draw();
    while (choice != 4) {
        cout << "\nMenu\n1.Translation\n2.Scaling\n3.Rotation\n4.Exit\nEnter
Choice : ";
        cin >> choice;
        switch (choice) {
        case 1:
            b.createTrans();
            c = a * b;
            c.showMatrix();
            c.draw();
            break;
        case 2:
            b.createScal();
            c = a + b;
            c.showMatrix();
            c.draw();
            break;
        case 3:
            b.createRota();
            c = a - b;
            c.showMatrix();
            c.draw();
            break;
        case 4:
            break;
        }
    }
    closegraph();
    return 0;
}
```

Output:

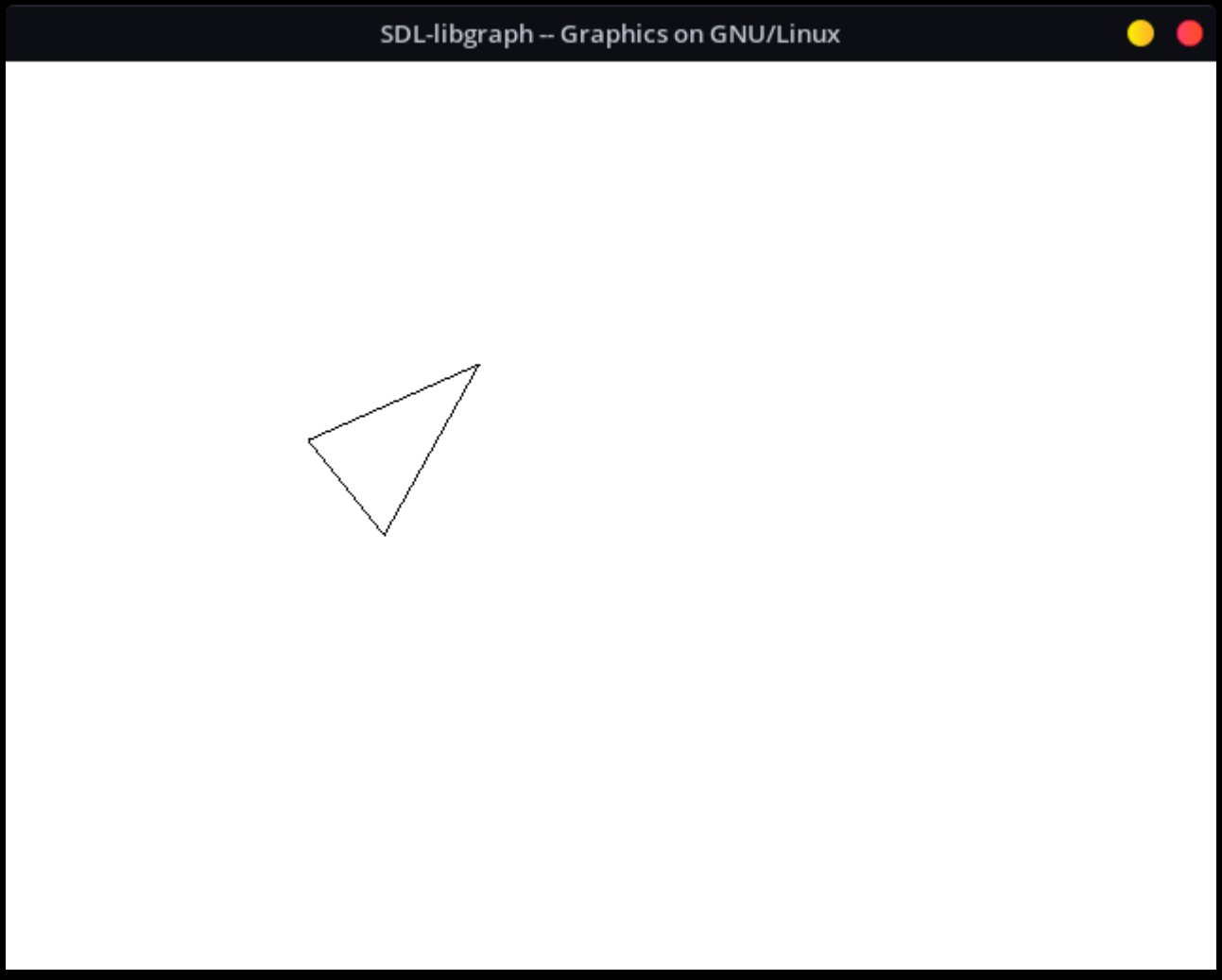
```
g++ "/home/proxima/Documents/Extras/CG/Practicals/2-d transformation.cpp" -o

~/Doc/E/C/Practicals
 g++ "/home/proxima/Documents/Extras/CG/Practicals/2-d transformation.cpp" -o "/home/proxima/Documents/Extras/CG/Practicals/2-d transformation" -lgraph -lGL -lGLU -lglut && "/home/proxima/Documents/Extras/CG/Practicals/2-d transformation"

Enter Number of points:3

Enter for Point 1:
Enter X coOr:10

Enter Y Coor:50

Enter for Point 2:
Enter X coOr:50

Enter Y Coor:100

Enter for Point 3:
Enter X coOr:100

Enter Y Coor:10

MAtrix:
        10      50      1
        50      100     1
        100     10      1

Menu
1.Translation
2.Scaling
```

SDL-libgraph -- Graphics on GNU/Linux



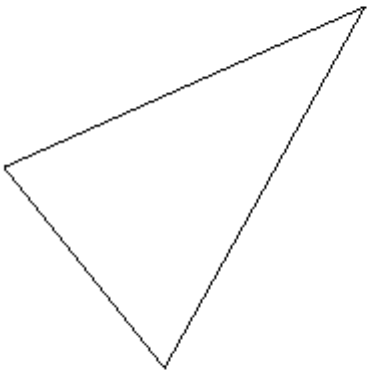
```
g++ "/home/proxima/Documents/Extras/CG/Practicals/2-d transformation.cpp" -o
Menu
1.Translation
2.Scaling
3.Rotation
4.Exit
Enter Choice : [xcb] Unknown sequence number while processing queue
[xcb] Most likely this is a multi-threaded client and XInitThreads has not been called
[xcb] Aborting, sorry about that.
2-d transformation: ../../src/xcb_io.c:260: poll_for_event: Assertion `!xcb_xlib_threads_sequence_lo
st' failed.
1

Enter Tx: and Ty:50 50

Trans Matrix:
        1       0       0
        0       1       0
        50      50      1

MAtrix:
        60      100     1
        100     150     1
        150     60      1
```

SDL-libgraph -- Graphics on GNU/Linux



```
g++ "/home/proxima/Documents/Extras/CG/Practicals/2-d transformation.cpp" -o
Menu
1.Translation
2.Scaling
3.Rotation
4.Exit
Enter Choice : 2

Enter Sx and SY:2 2

Scaling Matrix:
        2       0       0
        0       2       0
        0       0       1

MAtrix:
        20      100     1
        100     200     1
        200     20      1

Menu
1.Translation
2.Scaling
3.Rotation
4.Exit
Enter Choice : 3
```

SDL-libgraph -- Graphics on GNU/Linux



```
g++ "/home/proxima/Documents/Extras/CG/Practicals/2-d transformation.cpp" -o

Menu
1.Translation
2.Scaling
3.Rotation
4.Exit
Enter Choice : 3

Enter The angle by which to be rotated:25

1.Anticlockwise
2.Clockwise
Enter Choice:2

Rota Matrix:
        0.906308        -0.422618       0
        0.422618        0.906308        0
        0       0       1

MAtrix:
        30      41      1
        87      69      1
        94      -32     1
```