

Translation, Scaling and Shearing

1 Objective

To implement translation, scaling and shearing on given objects represented by their vertices.

2 Theory

2.1 Translation

Translation is the process of moving an object from one position to another. The object is translated by adding the translation vector to the coordinates of each vertex of the object. The translation vector is given by

$$\begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

2.2 Scaling

Scaling is the process of changing the size of an object. The object is scaled by multiplying the coordinates of each vertex of the object by the scaling factor. The scaling factor is given by

$$\begin{bmatrix} s_x \\ s_y \end{bmatrix}$$

2.3 Shearing

Shearing is the process of changing the shape of an object. The object is sheared by adding a multiple of one coordinate to the other coordinate of each vertex of the object. The shearing factor is given by

$$\begin{bmatrix} sh_x \\ sh_y \end{bmatrix}$$

3 Algorithm

3.1 Translation

1. Read the coordinates of the object.
2. Read the translation vector.
3. Add the translation vector to the coordinates of each vertex of the object.
4. Plot the translated object.

3.2 Scaling

1. Read the coordinates of the object.
2. Read the scaling factor.
3. Multiply the coordinates of each vertex of the object by the scaling factor.
4. Plot the scaled object.

3.3 Shearing

1. Read the coordinates of the object.
2. Read the shearing factor.
3. Add a multiple of one coordinate to the other coordinate of each vertex of the object.
4. Plot the sheared object.

4 Source Code

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

```
// translate to the axes defined by the computer
```

```
void translateAxes(int *x, int *y)
```

```
{  
    *x = (*x + WIDTH/2);  
    *y = (- *y + HEIGHT/2);  
}
```

```
void drawTriangle(int coordinates[3][2])
```

```
{  
    int copy[3][2];  
    for(unsigned i = 0; i<3; i++)  
    {  
        copy[i][0] = coordinates[i][0];  
        copy[i][1] = coordinates[i][1];  
    }  
    for(unsigned i = 0; i<3; i++)  
        translateAxes(&copy[i][0], &copy[i][1]);  
    // plot the translated points  
    for(unsigned i = 0; i<3; i++)  
        line(copy[i][0], copy[i][1], copy[(i+1)%3][0], copy[(i+1)%3][1]);  
}
```

```
void drawObject(unsigned num_points, int coordinates[][2])
```

```
{  
    int copy[num_points][2];  
    for(unsigned i = 0; i<num_points; i++)  
    {  
        copy[i][0] = coordinates[i][0];  
        copy[i][1] = coordinates[i][1];  
    }  
    for(unsigned i = 0; i<num_points; i++)  
        translateAxes(&copy[i][0], &copy[i][1]);  
    // plot the translated points  
    for(unsigned i = 0; i<num_points; i++)  
        line(copy[i][0], copy[i][1], copy[(i+1)%num_points][0], copy[(i+1)%num_points][1]);  
}
```

```
void scale(unsigned num_points, int coordinates[][2], float sx, float sy, int axis[2])
```

```
{  
    for (unsigned i = 0; i < num_points; i++)  
        
$$\frac{1}{2}$$

```

```

    {
        coordinates[i][0] = axis[0] + (coordinates[i][0] - axis[0]) * sx;
        coordinates[i][1] = axis[1] + (coordinates[i][1] - axis[1]) * sy;
    }
}

void shearX(unsigned num_points, int coordinates[][2], float shx, int ref)
{
    for (unsigned i = 0; i < num_points; i++)
    {
        coordinates[i][0] = coordinates[i][0] + (coordinates[i][1] - ref) * shx;
    }
}

void shearY(unsigned num_points, int coordinates[][2], float shy, int ref)
{
    for (unsigned i = 0; i < num_points; i++)
    {
        coordinates[i][1] = coordinates[i][1] + (coordinates[i][0] - ref) * shy;
    }
}

void translate(unsigned num_points, int coordinates[][2], int tx, int ty)
{
    for (unsigned i = 0; i < num_points; i++)
    {
        coordinates[i][0] += tx;
        coordinates[i][1] += ty;
    }
}

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

    // coordinates of a triangle in the first quadrant
    int triangle[3][2] =
    {
        {-50, 0},
        {-100, -100},
        {-50, -100}
    };

    int axis[2] = {0, 0};

    // translation stuff
    cleardevice();
    outtextxy(WIDTH/2, 10, "Translation");
    drawTriangle(triangle);
    translate(3, triangle, 100, 0);
    drawTriangle(triangle);

```

```

delay(2000);

// scaling stuff
cleardevice();
outtextxy(WIDTH/2, 10, "Scaling");
drawTriangle(triangle);
scale(3, triangle, 0.6, 0.6, axis);
drawTriangle(triangle);
delay(2000);

// shearing stuff
cleardevice();
outtextxy(WIDTH/2, 10, "Shearing");

int square[][2] =
{
    {0, 0},
    {100, 0},
    {100, 50},
    {0, 50}
};
drawObject(4, square);
delay(1000);

shearX(4, square, 2, 0);
drawObject(4, square);
delay(1000);
cleardevice();

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

```

5 Output

```

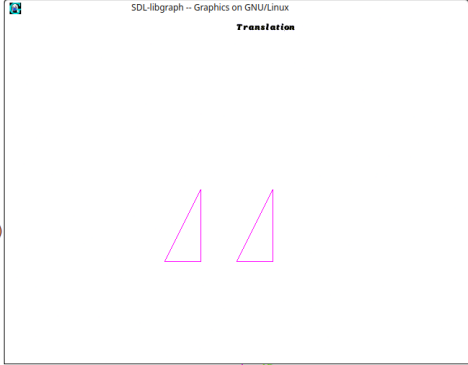
9 //coordinates of a triangle in the first quadrant
8 int triangle[3][2] =
7 {
6     {-50, 0},
5     {-100, -100},
4     {-50, -100}
3 };
2 int axis[2] = {0, 0};
1
71 //translation stuff
1 cleardevice();
2 outtextxy(WIDTH/2, 10, "Translation");
3 drawTriangle(triangle);
4 translate(3, triangle, 100, 0);
5 drawTriangle(triangle);
6 delay(2000);
7
8 // smooth translation
9 /*
10 for(size_t i = 0; i<200; i++)
11 {
12     cleardevice();
13     outtextxy(WIDTH/2, 10, "Translation")
14     translate(3, triangle, 1, 0);
15     drawTriangle(triangle);
16     delay(10);
17 }
18 */
19
20 //scaling stuff
21 cleardevice();
22 outtextxy(WIDTH/2, 10, "Scaling");
23 drawTriangle(triangle);
24 scale(3, triangle, 0.6, 0.6, axis);
25 drawTriangle(triangle);
26 delay(2000);
27
28
main.c

```

```

4 [jentishp@monika cg]$ make -B 2>/dev/null
3 gcc -Wall -Wextra -std=c99 -ggdb -Iinclude -Iinclude/algorithms -Lli
2 b src/main.c -o bin/main -lm -lgraph -lX11 -lSDL
1 ./bin/main
5
1
2
3
4
5
6

```



```

25
26
27
28
29
30
31
32
33

```

```

71:22 t//~/D/t/s/l/cg//68400:/bin/bash [-] < 5:1

```

Figure 1: Translation

```

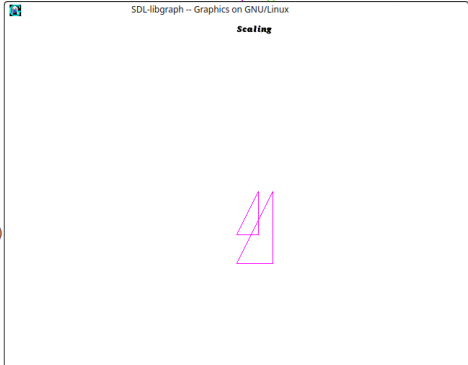
9 //coordinates of a triangle in the first quadrant
8 int triangle[3][2] =
7 {
6     {-50, 0},
5     {-100, -100},
4     {-50, -100}
3 };
2 int axis[2] = {0, 0};
1
71 //translation stuff
1 cleardevice();
2 outtextxy(WIDTH/2, 10, "Translation");
3 drawTriangle(triangle);
4 translate(3, triangle, 100, 0);
5 drawTriangle(triangle);
6 delay(2000);
7
8 // smooth translation
9 /*
10 for(size_t i = 0; i<200; i++)
11 {
12     cleardevice();
13     outtextxy(WIDTH/2, 10, "Translation")
14     translate(3, triangle, 1, 0);
15     drawTriangle(triangle);
16     delay(10);
17 }
18 */
19
20 //scaling stuff
21 cleardevice();
22 outtextxy(WIDTH/2, 10, "Scaling");
23 drawTriangle(triangle);
24 scale(3, triangle, 0.6, 0.6, axis);
25 drawTriangle(triangle);
26 delay(2000);
27
28
main.c

```

```

4 [jentishp@monika cg]$ make -B 2>/dev/null
3 gcc -Wall -Wextra -std=c99 -ggdb -Iinclude -Iinclude/algorithms -Lli
2 b src/main.c -o bin/main -lm -lgraph -lX11 -lSDL
1 ./bin/main
5
1
2
3
4
5
6

```



```

25
26
27
28
29
30
31
32
33

```

```

71:22 t//~/D/t/s/l/cg//68400:/bin/bash [-] < 5:1

```

Figure 2: Scaling

```

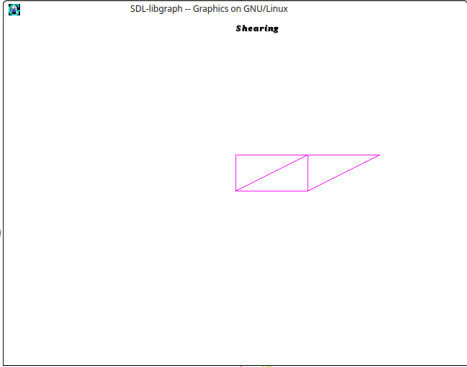
9 //coordinates of a triangle in the first quadrant
8 int triangle[3][2] =
7 {
6     {-50, 0},
5     {-100, -100},
4     {-50, -100}
3 };
2 int axis[2] = {0, 0};
1
71 //translation stuff
1 cleardevice();
2 outtextxy(WIDTH/2, 10, "Translation");
3 drawTriangle(triangle);
4 translate(3, triangle, 100, 0);
5 drawTriangle(triangle);
6 delay(2000);
7
8 // smooth translation
9 /*
10 for(size_t i = 0; i<200; i++)
11 {
12     cleardevice();
13     outtextxy(WIDTH/2, 10, "Translation");
14     translate(3, triangle, 1, 0);
15     drawTriangle(triangle);
16     delay(10);
17 }
18 */
19
20
21 //scaling stuff
22 cleardevice();
23 outtextxy(WIDTH/2, 10, "Scaling");
24 drawTriangle(triangle);
25 scale(3, triangle, 0.6, 0.6, axis);
26 drawTriangle(triangle);
27 delay(2000);
28
main.c

```

```

4 [jenishp@monika cg]$ make -B 2>/dev/null
3 gcc -Wall -Wextra -std=c99 -ggdb -Iinclude -Iinclude/algorithms -Lli
2 b src/main.c -o bin/main -lm -lgraph -lX11 -lSDL
1 ./bin/main
5
1
2
3
4
5
6
25
26
27
28
29
30
31
32
33

```



```

71:22 t//~/D/t/s/L/cg//68400:/bin/bash [-] <| 13% 5:1

```

Figure 3: Shearing

6 Conclusion

Given objects such as triangle and square represented by their vertices, translation, scaling and shearing were implemented on them using the graphics.h library.