Procedure:

1. Choose a 2D object.
2. Input the initial coordinates of the object's vertices.
3. Input the rotation angle.
4. Apply the rotation algorithm to obtain the transformed coordinates.
5. Visualize and compare the original and rotated objects.

3. Scaling:

Algorithm Steps:

1. Input the object's vertices.
2. Input the scaling factors (sx, sy).
3. For each vertex (x, y), perform the following:
   a. New_x = x * sx
   b. New_y = y * sy
4. Update the object's vertices with the new coordinates.

Procedure:

1. Choose a 2D object.
2. Input the initial coordinates of the object's vertices.
3. Input the scaling factors.
4. Apply the scaling algorithm to obtain the transformed coordinates.
5. Visualize and compare the original and scaled objects.

Coding:

```
#include<graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
void main()
{
    int graphdriver = DETECT, graphmode, errorc
    int i;
    int x2, y2, x1, y1, x, y;
    printf(" Enter the 2 line end points")
    printf("x1, y1, x2, y2");
    scanf("%d %d %d %d", &x1, &y1, &x2, &y2)
    initgraph (&graphdriver, &graphmode,&
        "C:\\tc\\BGI");
    line (x1, y1, x2, y2);
    printf (" Enter translation co-ordinates ");
    printf ("x, y");
    scanf ("%d %d", &x, &y);
```

```
x1 = x1+x;
y1 = y1+y;
x2 = x2+x;
y2 = y2+y;
printf("line after translation");
line(x1,y1,x2,y2);
getch();
closegraph();
}
```

* 2D Rotation :

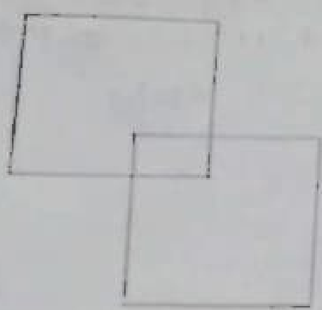
```
#include<graphics.h>
#include<stdlib.h>
#include <stdio.h>
#include <math.h>
void main(){
initgraph(driver = DETECT,
graphmode, errorcode;
int i;
int x2, y2, x1, y1, x, y, xn, yn;
double r11, r12, r21, r22, th;
clrscr();
printf(" Enter the 2 line end
        points");
printf("x1, y1, x2, y2");
scanf("%d %d %d %d",&x1,
    &x2, &y1, &y2);
nitgraph(&graphdriver,
graphmode "C:\\TC\\BGI");
ne(x1, y1, x2, y2);
intf("\n\n\nEnter the
    angle");
nf("%lf", &th);
I = cos((th+3.1428)/180);
= sin((th + 3.1428)/180);
```

```
r21 = (-sin((th+3.1428)/180));
r22 = cos((th + 3.1428)/180);
xn = ((x2 * r11) - (y2 * r12));
yn = ((x2 * r12) + (y2 * r11));
line(x1,y1,xn,yn);
getch();
closegraph();
}
```

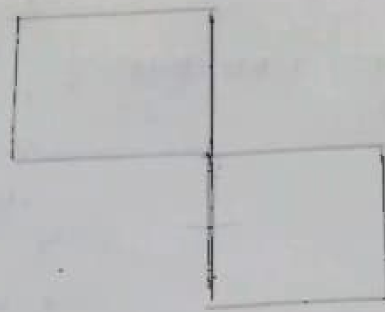
+ 2D Scaling

```
#include<graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
void main() {
int graphdriver = DETECT,
graphmode, errorcode;
int i;
int x2, y2, x1, y1, x, y;
printf ("Enter 2 line end points"
printf ("x1, y1, x2, y2");
scanf(" %d%d%d %d", &x1, y1, &x2
initgraph (&graphdriver,
&graphmode, "C:\\TC\\BGI"));
line (x1, y1, x2, y2);
printf ("Enter scaling coordinates
printf("x,y");
scanf("%d %d ",&x,&y);
x1 = (x1 * x);
y1 = (y1 * y);
x2 = (x2 * x);
y2 = (y2 + y1);
printf("Line After scaling :");
line (x1, y1, x2, y2);
getch();
closegraph();
}
```

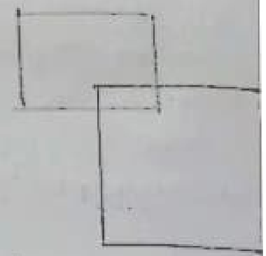
OUTPUT



Translation



Rotation



Scaling

Post Practical Questions:

1. **Transformation Order:** Explain the importance of the order in which translation, rotation, and scal transformations are applied. Provide examples to illustrate how changing the order affects the final res

Ans: The order of the composite transformation is Scale, then rotate, then translate. The result of th immediately preceding example is the same as the result of the first example.

2. **Impact on Geometric Shapes:** Consider a geometric shape (e.g., a rectangle or triangle) and di how each transformation (translation, rotation, scaling) affects the shape. Provide before-and illustrations for clarity.

Ans: The geometric transformation is a bijection of a set that has a geometric structure by itself or an other set. It is a shape is transpered its apperreance is change.

3. **Transformation Matrices:** Describe the role of transformation matrices in 2D graphics transfo
Illustrate how matrices are used to combine multiple transformations efficiently.

Ans: 2D graphics you may have encountered situ
where you need to transform or manipul
& mases shapes or text using different
or scale.

Conclusion: In this practical, we learn about 2D
Transformation operation, translation
Rotation and scaling.

Marks out of 10	
Signature with Date of Completion	

Coding:

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

# define Round (val) ((int)(val+.5))
int maxx, maxy, minx, miny;
void main() {
    int gd = DETECT, gm;
    void clipping (int xa, int ya, int xb, int yb);
    int xa, xb, ya, yb;
    printf("Enter the window co-ordination");
    scanf("%d %d %d %d", &minx, &miny,
            &maxx, &maxy);
    printf("Enter the two end points for the
            line");
    scanf ("%d %d %d %d", &xa, &ya, &xb, &yb)
    initgraph (&gd, &gm, " ");
    rectangle (minx, miny, maxx, maxy);
    line (xa, ya, xb, yb);
    getch();
    closegraph();
}
```

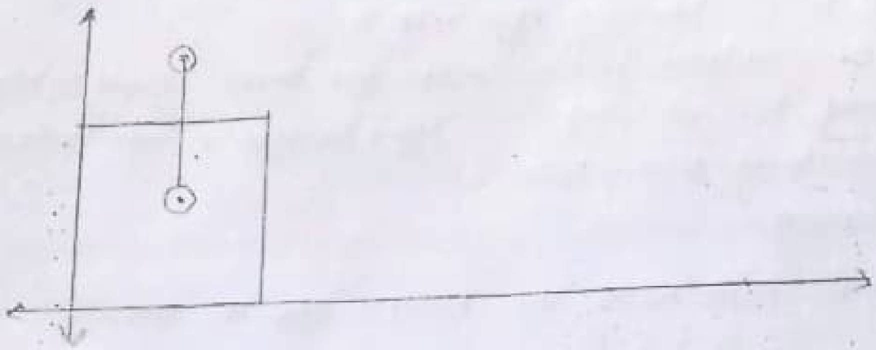


```
void clipping (int xa, int ya, int xb, int yb)
{
    int Dx = xb - xa, Dy = yb - ya, steps, k;
```

```
int visible1= 0, visible2= 0;
floatx xin, yin, x=xa, y=ya;
if (abs(Dx) > abs (Dy))
    steps= abs(DX);
else
    steps = abs (DY);
xin= Dx / (float) steps;
yin= Dy /(float) steps;
putpixel (Round(x), Round(y),2);
}
for (k=0; k< steps; k++)
{
  x += xin;
  y+ = yin;
  if( y > min y && y <max x )
  {
      visible1 =1;
      putpixel (Round(x) ,(Round(y),2);
  }else{
      visible 2=1;
  if (visible==0)
     outtextxy (20,200, "completely visible");
  if (visible ==1 && completely visible2==1)
     outtextxy ( 20, 20, "Partialy visible");
  if (visible= =1 && visible 2= =0)
     outtexty (20,20," Completely visible");
```

OUTPUT

The area of code of 1st point is 0000
The area of code of end point is 1000.



Post Practical Questions:

1. **Clipping Window:** Explain the concept of a clipping window and its role in the Cohen-Sutherland Line Clipping Algorithm. Discuss how the algorithm determines whether a line is inside, outside, or partially inside the clipping window.

Ans: A clipping window, in computer graphics, defines the region of a screen where drawing or rendering is allowed. The cohen-Sutherland clipping algorithm is a method used to determine whether a line segment lies completely within, completely outside or partially within the clipping window.

2. Discuss the classification of lines in Cohen-Sutherland as entirely inside, entirely outside, or partially inside the clipping window. Illustrate each scenario with examples.

Ans: In the cohen-Sutherland algorithm lines are classified as either entierely inside, entierely outside or partially inside the clipping window based on their endpoint codes. By classifying lines in this manner, the cohen-sutherland algorithm efficiently determines which portion of line area to be clipped.

SILVER OAK UNIVERSITY
EDUCATION TO INNOVATION

on Techniques: Explore optimization techniques for the Cohen-Sutherland Line Clipping Discuss strategies to improve its efficiency, especially in scenarios involving numerous lines.

are the optimization techniques for the cohen-
rland line clipping Algorithm.
ing box optimization - Special Data Structure
egment Reordering - Hardware Acceleration
Sutherland reduction
elization

n this practical, we learn about the
en- Sutherland line clipping Algorithm.

of 10	
vith Date of Completion	

SILVER OAK UNIVERSITY
EDUCATION TO INNOVATION

PRACTICAL - 9

AIM: Perform the Liang-Barsky Line Clipping Algorithm.

Description:

In this practical, students will explore and implement the Liang-Barsky Line Clipping Algorithm. This algorithm is widely used for clipping lines against a rectangular window in computer graphics. Understanding line clipping is crucial for displaying only the relevant portions of a line within a specified window.

Objective:

- Understand the principles of the Liang-Bafsky Line Clipping Algorithm.
- Learn the steps involved in clipping lines against a rectangular window.
- Implement the algorithm to efficiently clip lines.
- Visualize the results and observe the impact of the clipping process.

Liang-Barsky Line Clipping Algorithm:

Algorithm Steps:

1. Define the rectangular clipping window.
2. Calculate the parameters p1, p2, p3, and p4 for the line segment using the window boundaries.
3. Check if the line is completely outside the window (reject) based on the calculated parameters.
4. If the line is not rejected, calculate the values of t1 and t2 using the window boundaries.
5. If t2 < t1, reject the line since it is entirely outside the window.
6. Clip the line using the calculated values of t1 and t2.
7. Update the line coordinates based on the clipping.

rocedure:

1. Define a rectangular clipping window (e.g., (xmin, ymin), (xmax, ymax)).
2. Choose a line segment with endpoints (x1, y1) and (x2, y2).
3. Apply the Liang-Barsky Line Clipping Algorithm to clip the line against the window.
4. Visualize and compare the original line with the clipped line.
5. Experiment with different lines and window configurations to observe the algorithm's behavior.

ding:

```
# include <stdio.h>
# include <conio.h>
# include <graphics.h>
# include <math.h>
void main()
{
    int i, gm, gd;
    int x1, x2, y1, y2, xmin, xmax, ymin, xx1, xx2, yy1, yy2;
    float t1, t2, p[4], q[4], temp;
    detectgraph(&gd, &gm)
```

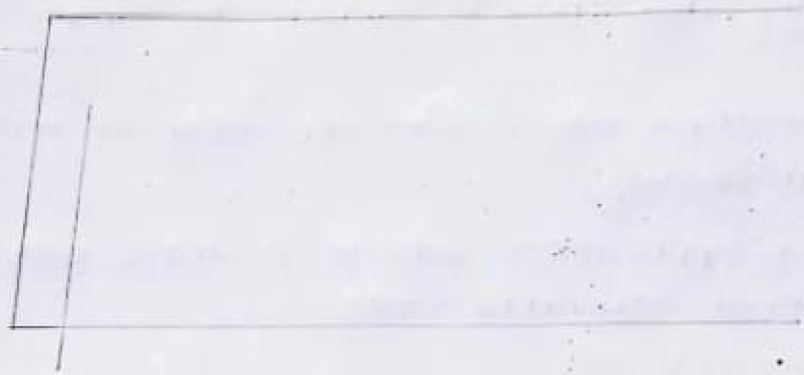
```
initgraph(&gd, &gm, "c:\\TC\\BGI");
xmin= 150;
xmax= 450;
ymin= 100;
ymax= 400;
rectangle(xmin, ymin, xmax,
          ymax);
printf("Enter starting point:");
scanf("%d %d", &x1, &y1);
printf("Enter ending point : ");
scanf("%d %d", &x2, &y2);
printf(" Line after clipping: ");
line(x1, y1, x2, y2);
getch();
clrscr();
printf("Line after clipping:")
rectangle(xmin, ymin, xmax,
          ymax);
P[0] = -(x2-x1);
P[1] = (x2-x1);
P[2] = -(y2-y1);
P[3] = (y2-y1);
q[0]= (x1-xmin);
q[1]= (xmax - x1);
q[2] = (y1-ymin);
q[3]= (ymax-y1);
for (i=0; i<4; i++

  if (p[i]==0)
  {
```

```
printf("\n Line is parallel
       to one of the
       clipping boundary");
 if (q[i] >=0)
 {
    if (y1 <ymin)
    {
      y1 = ymin;
    }
    if (y2 > ymax)
    {
       y2 = ymax;
    }
    line(x1, y1, x2, y2);
 }
 if (i>1)
 {
    if (x1< xmin)
    {
      x1= xmin;
    }
    if (x2 > xmax)
    {
      x2 = xmax;
    }
    line (x1, y1, x2, y2);
  }
}
getch();
return();
}
}
```

OUTPUT

enter the starting point = 100
100
Enter the ending point = 200
line after clipping



Post Practical Questions:

1. **Parameterization of Lines:** Explain the concept of parameterization of lines in the Liang-Barsky Line Clipping Algorithm. Discuss how this parameterization simplifies the clipping process.

Ans: In the Liang - Barsky line clipping Algorithm, parameterization of lines involves representing a line segment in terms of parameters rather than explicity defining its endpoints improves efficiency compares to other methods.

2. **Clipping against Multiple Edges:** Discuss the approach taken by Liang-Barsky for clipping against multiple edges simultaneously. Provide examples to illustrate the algorithm's behavior in different scenarios.

Ans: Here an overview of approach :
- initialization
- intersection tests
- iteration
- final clipping

→ If the line segment lies entirely inside the clipping window, the algorithm will not need to perform any clipping.

3. **Comparative Analysis:** Compare the Liang-Barsky Line Clipping Algorithm with other line clipping algorithms, such as Cohen-Sutherland. Discuss advantages, disadvantages, and scenarios where one algorithm might be preferred over the other.

Ans: Parameterization :- parameterization of lines, making it computationally efficient.

- Simplicity - The algorithm is relatively simple to implement.
- Handleys multiple edges : Can clip against multiple edges stimutensonaly.
- Suitable four applications where multiple edges need to be clipped stimuttaneusly.

Conclusion:

In this practical, we learn about the Liang - Barsky Line clipping Algorithm.

Marks out of 10	
Signature with Date of Completion	

SILVER OAK UNIVERSITY
EDUCATION TO INNOVATION

Implement the animation algorithm for the moving car and bouncing ball.
Adjust parameters such as speed, direction, and bouncing behavior to observe different effects
Run the animation and visualize the movement of the car and the bouncing ball.
Experiment with modifying the algorithm to achieve different animation behaviors.

Coding:

```
# include <stdio.h>
# include <stdlib.h>
# include <conio.h>
# include <graphics.h>
# include <math.h>
void main()
{
    int gd = DETECT, gm;
    int i, x, y, flag = 0;
    initgraph (&gd, &gm, "C:\\TC \\BGI");
     x = getmax() /2;
     y = 30;
     while (!kbhit())
     {
          if (y > getmaxy() - 30 || y <= 30)
               flag = !flag;
               setcolour (RED);
               setfillstyle (SOLID-FILL, RED);
                circle (x, y, 30);
                floodfill (x, y, RED);
                delay (25);
                cleardevice();
```

```
    if (flag)
    {
        y = y+2;
    }
    else {
        y = y-2;
    }
  }
  getch();
  closegraph();
}
```

OUTPUT

SILVER OAK UNIVERSITY
EDUCATION TO INNOVATION

Post Practical Questions:

1. Discuss the animation techniques used in your program for creating the motion of a car and bouncing ball. Explain how the frame-by-frame rendering and timing are managed.

Ans: Frame - by frame animation, twinning, timing and frame state management, interpolation, physics based Animation.
- Frame-by frame rendering & timing involves cartesy coordinations of keyframes.

2. **Realistic Movement:** How would you enhance the realism of the car's movement? Discuss additional features or considerations that could be added to improve the natural appearance of motion in the animation.

Ans: To enhance the realism of a car's movement in animation, could be implemented
- Realistic, physics animation
- Dynamic lightning & scheduling
- Particle effects
- Sound effects
- Camera movement

3. **User Interaction:** Implement user interaction in your animation, allowing users to control the speed or direction of the moving car and bouncing ball. Discuss the significance of user interaction in enhancing the overall animation experience.

Ans: Overall user interaction plays a crucial role in enhancing the overall animation experience by by increasing engagement, enabling customization, promoting learning, interactivity & providing feedback & responsiveness.

Conclusion: In this practical, we learn about animation algorithm for moving car & Bouncing ball.

Marks out of 10	
Signature with Date of Completion	