DDA & BRESENHEM

Algorithm (DDA)

Step1: Start Algorithm

Step2: Declare x1,y1,x2,y2,dx,dy,x,y as integer variables.

Step3: Enter value of x1,y1,x2,y2

Step4: Calculate dx = x2-x1

Step5: Calculate dy = y2-y1

Step6: If abs(dx) > abs(dy) Then

    step = abs (dx)

    else

    step = abs (dy)

Step7: xinc=dx/step

    yinc=dy/step

    x = x1

    y = y1

Step8: Set pixel (x, y)

Step9: x = x + xinc y = y + yinc

Set pixels (Round (x), Round (y))

Step10: Repeat step 9 until x = x2

Step11: End Algorithm

Program (DDA)

```
#include<graphics.h>
#include<math.h>
#include<conio.h>
void main()
{
int x0,y0,x1,y1,i=0; float delx,dely,len,x,y;
int gr=DETECT,gm;
initgraph(&gr,&gm,"C:\\TURBOC3\\BGI\\BIN");
printf("\nEnter the values ofx1,y1,x2,y2 = ");
scanf("%d %d",&x0,&y0,&x1,&y1);
dely=abs(y1-y0); delx=abs(x1-x0);
if(delx<dely)
{
len = dely;
}else
{
len=delx; }
delx=(x1-x0)/len;
dely=(y1-y0)/len;
x=x0+0.5; y=y0+0.5;
do{
putpixel(x,y,3);
x=x+delx; y=y+dely; i++;
delay(30); }
while(i<=len);
getch();
closegraph();
}
```

Algorithm (BRESENHEM)

Step 1: Enter the 2 end points for a line and store the left end point in (X0,Y0).

Step 2: Plot the first point be loading (X0,Y0) in the frame buffer.

Step 3: determine the initial value of the decision parameter by calculating the constants dx, dy, 2dy and 2dy-2dx as P0 = 2dy –dx

Step 4: for each Xk, conduct the following test, starting from k= 0

If Pk <0, then the next point to be plotted is at (Xk+1, Yk) and Pk+1 = Pk + 2dy

else, the next point is (Xk+1, Yk+1) and Pk+1 = Pk + 2dy –2dx (step 3)

Step 5: iterate through step (4) dx times.

Program (BRESENHEM)

```
#include<stdio.h>

#include<graphics.h>

void drawline(int x0, int y0, int x1, int y1)

{

  int dx, dy, p, x, y;

  dx=x1-x0;

  dy=y1-y0;

  x=x0;

  y=y0;

  p=2*dy-dx;

  while(x<x1)

  {

    if(p>=0)

    {

      putpixel(x,y,7);

      y=y+1;

      p=p+2*dy-2*dx;   }
```

```
        else
        {
            putpixel(x,y,7);
            p=p+2*dy;}
            x=x+1;
        }
}
int main()
{
    int gdriver=DETECT, gmode, error, x0, y0, x1, y1;
    initgraph(&gdriver, &gmode, "c:\\turboc3\\bgi");
    printf("\nEnter the values ofx1,y1,x2,y2 = ");
    scanf("%d %d",&x0,&y0,&x1,&y1);
     drawline(x0, y0, x1, y1);
    getch();
    closegraph();
    return 0;

}
```

Midpoint Algorithm

Algorithm

Step1: Put x =0, y =r

We have p=1-r

Step2: Repeat steps while x ≤ y

Plot (x, y)

If (p<0)

Then set p = p + 2x + 3

Else

p = p + 2(x-y)+5

y =y - 1 (end if)

x =x+1 (end loop)

Step3: End

Program:

```
#include<stdio.h>
#include<conio.h>
#include<graphics.h>
void drawcircle(int x0, int y0, int radius)
{
int x = radius; int y = 0; int err = 0;
while (x >= y)
{
putpixel(x0 + x, y0 + y, 7);
putpixel(x0 + y, y0 + x, 7);
putpixel(x0 - y, y0 + x, 7);
putpixel(x0 - x, y0 + y, 7);
putpixel(x0 - x, y0 - y, 7);
putpixel(x0 - y, y0 - x, 7);
putpixel(x0 + y, y0 - x, 7);
putpixel(x0 + x, y0 - y, 7);
if (err <= 0)
{
```

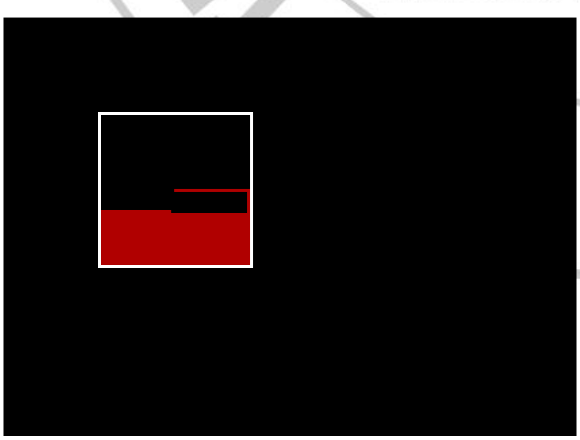
```
y += 1;
err += 2*y + 1;
}
if (err > 0) {
x -= 1;
err -= 2*x + 1;
delay(20);
}
}
}
void main()
{
int gd=DETECT, gm,x, y, r;
initgraph(&gd, &gm, "C:\\TURBOC3\\BGI");
printf("Enter radius of circle: ");
scanf("%d", &r);
printf("Enter co-ordinates of center(x and y): ");
scanf("%d%d", &x, &y);
drawcircle(x, y, r);
getch();
closegraph();
}
```

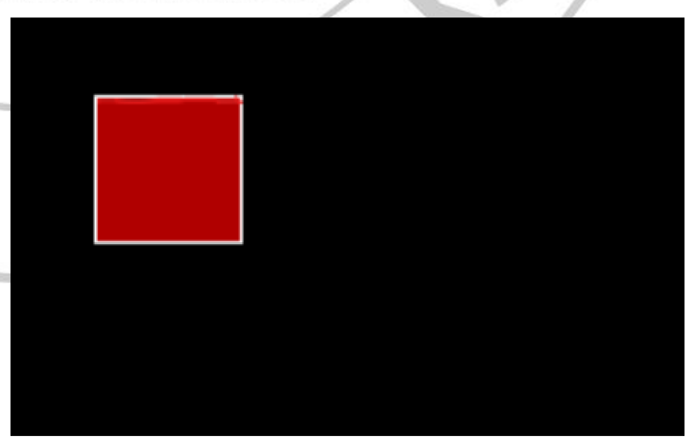
Programs with Output:

BOUNDARY FILL PROGRAM:

```
#include<stdio.h>
#include<graphics.h>
#include<math.h>
void boundaryFill8(int x, int y, int fill_color, int boundary_color){
    if(getpixel(x,y)!=boundary_color && getpixel(x,y)!=fill_color)
    {
     putpixel(x,y,fill_color);
     delay(10);
     boundaryFill8(x+1,y,fill_color,boundary_color);
     boundaryFill8(x,y+1,fill_color,boundary_color);
     boundaryFill8(x-1,y,fill_color,boundary_color);
     boundaryFill8(x,y-1,fill_color,boundary_color);
     boundaryFill8(x-1,y-1,fill_color,boundary_color);
     boundaryFill8(x-1,y+1,fill_color,boundary_color);
     boundaryFill8(x+1,y-1,fill_color,boundary_color);
     boundaryFill8(x+1,y+1,fill_color,boundary_color);
   }
}
void main(){
int gd=DETECT,gm;
initgraph(&gd,&gm,"C:\\TC\\BGI");
rectangle(50,50,100,100);
boundaryFill8(75,75,4,15);
getch();
closegraph();}
```

BOUNDARY FILL OUTPUT:





FLOOD FILL PROGRAM:

```
#include<graphics.h>
#include<stdio.h>
void flood(int x, int y, int new_col, int old_col)
{
if(getpixel(x,y)==old_col)
{
putpixel(x,y,new_col);
delay(50);
flood(x+1,y,new_col,old_col);
flood(x-1,y,new_col,old_col);
flood(x,y+1,new_col,old_col);
flood(x,y-1,new_col,old_col);
}
}
void main()
{
 int gd = DETECT, gm;
 int top, left, bottom, right, x, y, newcolor, oldcolor;
 //Initialize graph
 initgraph(&gd, &gm, "C:\\TC\\BGI");
 //Rectangle Co-ordinate
 top = left = 50; bottom = right = 100;
 //Rectangle for print rectangle
 rectangle(left, top, right, bottom);
 //Filling Start Co-ordinatex
 = 51; y = 51;
 //New color to fill
 newcolor = 12;
 //New clor which you want to fill
 oldcolor = 0;
 //Call for fill rectangle
 flood(x, y, newcolor, oldcolor);
 getch();
 closegraph();
}
```

2D TRANSFORMATION (ROTATION, TRANSLATION, SCALING)

Algorithm

1. Start
2. Initialize the graphics mode.
3. Construct a 2D object (use Drawpoly()) e.g. (x,y)

A) Translation

a. Get the translation value tx, ty

b. Move the 2d object with tx, ty (x'=x+tx,y'=y+ty)

c. Plot (x',y')

B) Scaling

a. Get the scaling value Sx,Sy

b. Resize the object with Sx,Sy (x'=x*Sx,y'=y*Sy)

c. Plot (x',y')

C) Rotation

a. Get the Rotation angle

b. Rotate the object by the angle ϕ

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

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

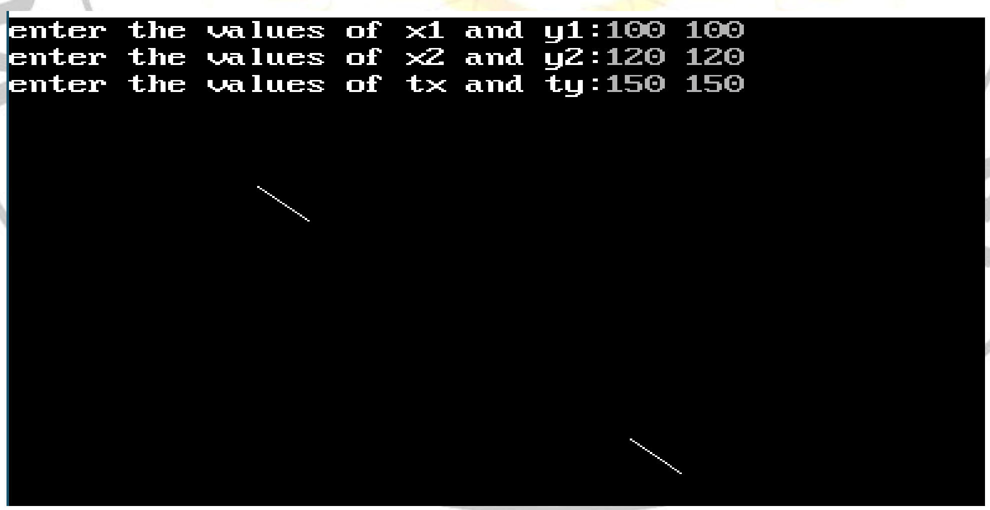
c. Plot (x',y')

Programs:

TRANSLATION:

```
#include<stdio.h>
#include <graphics.h>
#include <conio.h>
void main()
{
int gd = DETECT, gm;
int xmax, ymax,x1,y1,x2,y2,tx,ty;
initgraph(&gd, &gm, “c:\turboc3\bin”);
printf(“Enter the values X1 and y1:”);
scanf(“%d %d”,&x1,&y1);
printf(“Enter the values of X2 and y2:”);
scanf(“%d %d”,&x2,&y2);
printf(“Enter the values of tx and ty:”);
scanf(“%d %d”,&tx,&ty);
line(x1,y1,x2,y2);
line(x1+tx,y1+ty,x2+tx,y2+ty);
getch();
closegraph();
}
```

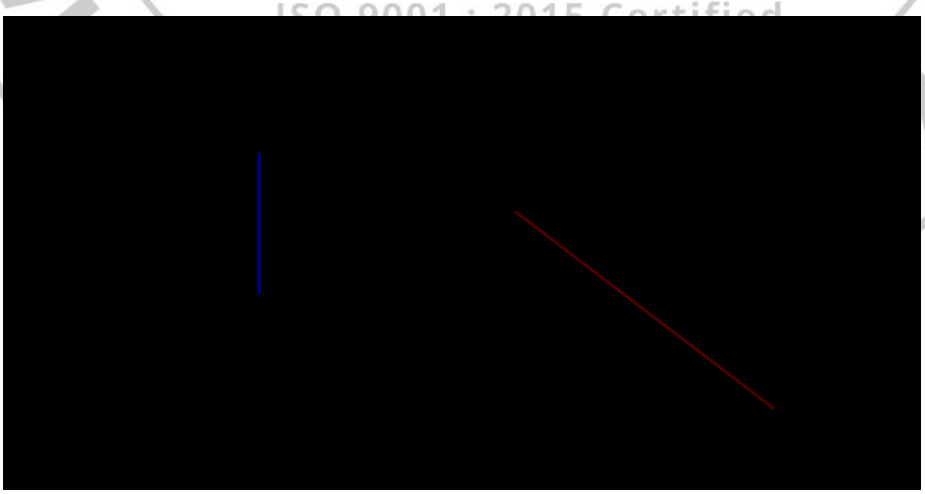
Output:



ROTATION:

```
#include<stdio.h>
#include<conio.h>
#include<graphics.h>
#include<math.h>
void main()
{
int gd=DETECT,gm;
int x1,y1,x2,y2;
double s,c, angle;
printf("Enter coordinates of line: ");
scanf("%d%d%d%d",&x1,&y1,&x2,&y2);
printf("Enter rotation angle: ");
scanf("%lf", &angle);
initgraph(&gd, &gm, "C:\TurboC3\\Bin");
setcolor(RED);
line(x1,y1,x2,y2);
c = cos((angle *3.14)/180);
s = sin((angle *3.14)/180);
x1 = floor(x1 * c - y1 * s);
y1 = floor(x1 * s + y1 * c);
x2 = floor(x2 * c - y2 * s);
y2 = floor(x2 * s + y2 * c);
setcolor(BLUE);
line(x1, y1 ,x2, y2);
getch();
closegraph();
}
```

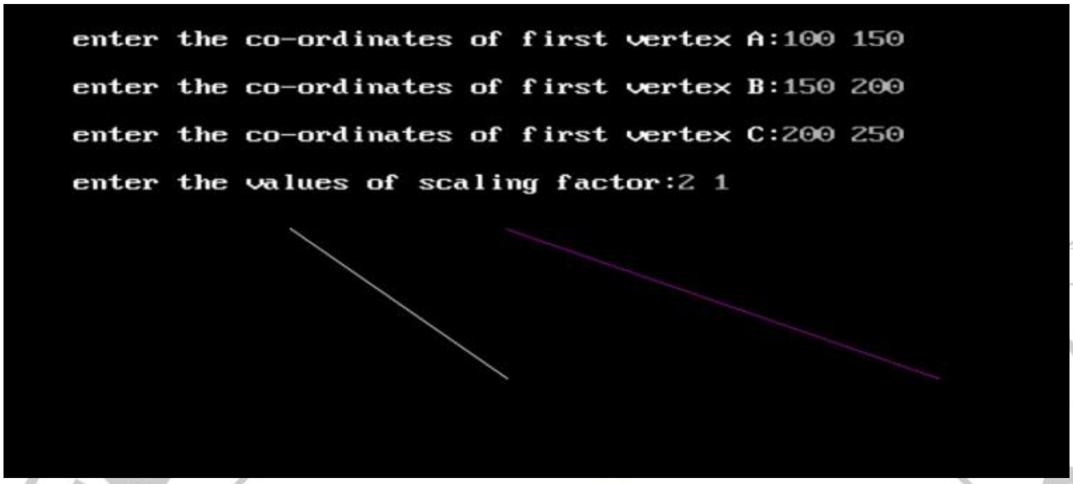
Output:



SCALING:

```
#include<stdio.h>
#include<conio.h>
#include<graphics.h>
#include<math.h>
void main()
{
int x1,y1,x2,y2,x3,y3,sx,sy;
int gd=DETECT,gm;
initgraph(&gd,&gm,"C:\\TURBOC3\\BGI");
printf("\nenter the co-ordinates of first vertex A:");
scanf("%d %d",&x1,&y1);
printf("\nenter the co-ordinates of first vertex B:");
scanf("%d %d",&x2,&y2);
printf("\nenter the co-ordinates of first vertex C:");
scanf("%d %d",&x3,&y3);

line(x1,y1,x2,y2);
line(x2,y2,x3,y3);
line(x3,y3,x1,y1);

printf("\nenter the values of scaling factor:");
scanf("%d %d",&sx,&sy);

x1 = x1 * sx;
y1 = y1 * sy;
x2 = x2 * sx;
y2 = y2 * sy;
x3 = x3 * sx;
y3 = y3 * sy;

setcolor(5);

line(x1,y1,x2,y2);
line(x2,y2,x3,y3);
line(x3,y3,x1,y1);

getch();
closegraph();
}
```

Output:


enter the co-ordinates of first vertex A:100 150
enter the co-ordinates of first vertex B:150 200
enter the co-ordinates of first vertex C:200 250
enter the values of scaling factor:2 1

2D TRANSFORMATION (REFLECTION, SHEARING)

Reflection Algorithm

Step 1. Initialize graphics library and get graphics mode DETECT.
Step 2. Clear the graphics window.
Step 3. Set the initial coordinates of the triangle using 'x1', 'x2', 'x3', 'y1', 'y2', and 'y3'.
Step 4. Draw a vertical and a horizontal line to divide the window into four quadrants.
Step 5. Draw an object in the second quadrant and display it.
Step 6. Reflect the object about the Y-axis to obtain the mirror image in the first quadrant.
Step 7. Reflect the object about the X-axis to obtain the mirror image in the fourth quadrant.
Step 8. Close the graphics window.

Shearing Algorithm

Step 1. Declare variables for graphics driver and mode, coordinates of the triangle (x, y), (x1, y1), (x2, y2), and shearing factor (shear_f).
Step 2. Take user input for the three triangle coordinates and shearing factor using the scanf() function.
Step 3. Draw the initial triangle using the line() function.
Step 4. Apply shearing transformation on the triangle coordinates by adding y multiplied by the shearing factor to x using the equation: x = x + y * shear_f. Apply the same equation to x1 and x2.
Step 5. Draw the transformed triangle using the line() function.
Step 6. Stop

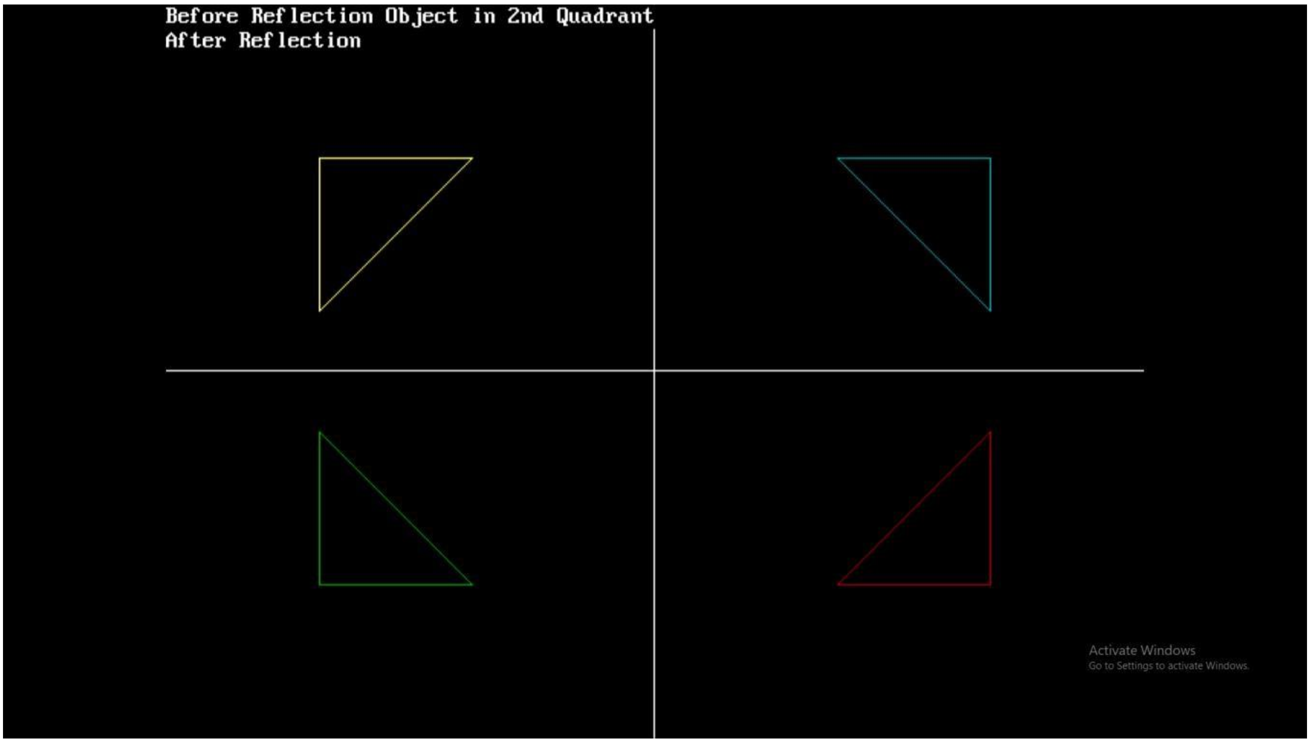
Programs with output:

Reflection:

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

void main()
{

        int gm, gd = DETECT, ax, x1 = 100;
        int x2 = 100, x3 = 200, y1 = 100;
        int y2 = 200, y3 = 100;


        initgraph(&gd, &gm, ""C:\\TURBOC3\\BGI"");
        cleardevice();


        line(getmaxx() / 2, 0, getmaxx() / 2,
                getmaxy());
        line(0, getmaxy() / 2, getmaxx(),
                getmaxy() / 2);


        printf("Before Reflection Object"
                " in 2nd Quadrant");


        setcolor(14);
        line(x1, y1, x2, y2);
        line(x2, y2, x3, y3);
        line(x3, y3, x1, y1);
        getch();


        printf("\nAfter Reflection");



        setcolor(4);
        line(getmaxx() - x1, getmaxy() - y1,
                getmaxx() - x2, getmaxy() - y2);

        line(getmaxx() - x2, getmaxy() - y2,
                getmaxx() - x3, getmaxy() - y3);

        line(getmaxx() - x3, getmaxy() - y3,
                getmaxx() - x1, getmaxy() - y1);

                        setcolor(3);
```

```
        line(getmaxx() - x1, y1,
                getmaxx() - x2, y2);
        line(getmaxx() - x2, y2,
                getmaxx() - x3, y3);
        line(getmaxx() - x3, y3,
                getmaxx() - x1, y1);



        setcolor(2);
        line(x1, getmaxy() - y1, x2,
                getmaxy() - y2);
        line(x2, getmaxy() - y2, x3,
                getmaxy() - y3);
        line(x3, getmaxy() - y3, x1,
                getmaxy() - y1);
        getch();

        // Close the graphics
        closegraph();
}
```

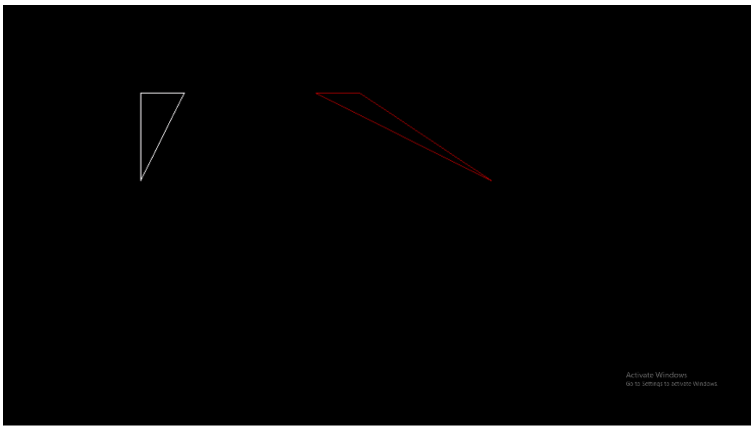
Output:



Shearing:

```
#include<stdio.h>
#include<graphics.h>
#include<conio.h>
void main()
{
int gd=DETECT,gm;
int x,y,x1,y1,x2,y2,shear_f;
initgraph(&gd,&gm,"C:\\TURBOC3\\BGI");
printf("\n please enter first coordinate = ");
scanf("%d %d",&x,&y);
printf("\n please enter second coordinate = ");
scanf("%d %d",&x1,&y1);
printf("\n please enter third coordinate = ");
scanf("%d %d",&x2,&y2);
printf("\n please enter shearing factor x = ");
scanf("%d",&shear_f);
cleardevice();
line(x,y,x1,y1);
line(x1,y1,x2,y2);
line(x2,y2,x,y);

setcolor(RED);
x=x+ y*shear_f;
x1=x1+ y1*shear_f;
x2=x2+ y2*shear_f;

line(x,y,x1,y1);
line(x1,y1,x2,y2);
line(x2,y2,x,y);
getch();
closegraph();
}
```

Output:



COHEN SUTHERLAND LINE

Algorithm

1. Read two end points of the line P1(x1,y1) and P2(x2,y2)

2. Read two corners (top-left and bottom right) of the window (wx1,wy1) and (wx2,wy2)

3. Assign the region codes for end points p1 and p2 using following steps

Initialize code with bits 0000

Set bit 1= if(x<wx1)
Set bit 2= if(x>wx2)
Set bit 3= if(x<wy1)
Set bit 4= if(x>wy1)

4. check for visibility of line p1 and p2

a. If the region codes for both endpoints p1 and p2 are zero then the line is completely visible . Hence draw the line and goto step 9

b. If the region codes for both endpoints p1 and p2 are non-zero and the logical ANDing of them is also non zero then the line is completely invisible, so reject the line and goto step 9

c. If the region codes for both endpoints p1 and p2 do not satisfy condition 4a and 4b the line is partially visible

5. Determine the intersecting edge of the clipping window by inspecting the region codes of 2 end points

a. If the region codes for both endpoints p1 and p2 are non-zero, find intersection points pe1, pe2 with boundary edges of clipping window with respect to p1 & p2

b. If the region codes for any one endpoint are non zero, find intersection points pe1, pe2 with boundary edges of clipping window with respect to it.

6. Divide the line segments considering intersection points

7. Reject the line segment if any one end point as it appears outside the clipping window

8. Draw the remaining line segments

9. Stop

Program

```
#include<graphics.h>

#include<dos.h>

#include<conio.h>

#include<stdio.h>

#include<math.h>

{

int rcode_begin[4]={0,0,0,0},rcode_end[4]={0,0,0,0},region_code[4];

int W_xmax,W_ymax,W_xmin,W_ymin,flag=0;

float slope; int x,y,x1,y1,i, xc,yc;

int gr=DETECT,gm;

initgraph(&gr,&gm,"C:\\TURBOC3\\BGI");

printf("\n*** Cohen Sutherlsnd Line Clipping algorithm ****\n");

printf("\n First enter XMax, YMax =");

scanf("%d%d",&W_xmax,&W_ymax);

printf("\n Now, enter XMin, YMin =");

scanf("%d %d",&W_xmin,&W_ymin);

printf("\n Please enter intial point x and y = ");

scanf("%d%d",&x,&y);

printf("\n Now, enter final point x1 and y1 = ");

scanf("%d %d",&x1,&y1);

cleardevice();

rectangle(W_xmin,W_ymin,W_xmax,W_ymax);

line(x,y,x1,y1);

line(0,0,600,0);

line(0,0,0,600);

if(y>W_ymax) {

rcode_begin[0]=1; // Top

flag=1 ;
```

```
} if(y<W_ymin)
{
rcode_begin[1]=1; // Bottom flag=1;
} if(x>W_xmax)
{
rcode_begin[2]=1; // Right
flag=1; } if(x<W_xmin) {
rcode_begin[3]=1; //Left
flag=1;
}
if(y1>W_ymax){ rcode_end[0]=1; // Top flag=1;
} if(y1<W_ymin)
{
rcode_end[1]=1; // Bottom flag=1;
}
if(x1>W_xmax){ rcode_end[2]=1; // Right flag=1;

}
if(x1<W_xmin){
rcode_end[3]=1; //Left
flag=1; }
if(flag==0
){
printf("No need of clipping as it is already in window"); } flag=1;
for(i=0;i<4;i++)
{
region_code[i]= rcode_begin[i] && rcode_end[i] ; if(region_code[i]==1) flag=0; } if(flag==0) {
printf("\n Line is completely outside the window");
} else{ slope=(float)(y1-
```

```
y)/(x1-x);
if(rcode_begin[2]==0 && rcode_begin[3]==1) //left
{
y=y+(float) (W_xmin-x)*slope ; x=W_xmin;
}
if(rcode_begin[2]==1 && rcode_begin[3]==0) // right {
y=y+(float) (W_xmax-x)*slope ; x=W_xmax;
}
if(rcode_begin[0]==1 && rcode_begin[1]==0) // top {
x=x+(float) (W_ymax-y)/slope ; y=W_ymax;
}
if(rcode_begin[0]==0 && rcode_begin[1]==1) // bottom {
x=x+(float) (W_ymin-y)/slope ; y=W_ymin;
}
// end points
if(rcode_end[2]==0 && rcode_end[3]==1) //left {
y1=y1+(float) (W_xmin-x1)*slope ; x1=W_xmin;
}
if(rcode_end[2]==1 && rcode_end[3]==0) // right {
y1=y1+(float) (W_xmax-x1)*slope ;

 x1=W_xmax;
}
if(rcode_end[0]==1 && rcode_end[1]==0) // top {
x1=x1+(float) (W_ymax-y1)/slope ;
y1=W_ymax;
}
if(rcode_end[0]==0 && rcode_end[1]==1) // bottom {
x1=x1+(float) (W_ymin-y1)/slope ;
```

```
y1=W_ymin;
}}
delay(1000);
clearviewport()
; rectangle(W_xmin,W_ymin,W_xmax,W_ymax); line(0,0,600,0);
line(0,0,0,600);
setcolor(RED);
line(x,y,x1,y1);
getch();
closegraph();
}
```

Sutherland Hodgemen Polygon

Step 1: Read co-ordinates of all vertices of the polygon.

Step 2: Read co-ordinates of the clipping window.

Step 3: Consider the left edge of window.

Step 4: Compare vertices of each of polygon, individually with the clipping plane.

Step 5: Save the resulting intersections and vertices in the new list of vertices according to four possible relationships between the edge and the clipping boundary.

Step 6: Repeat the steps 4 and 5 for remaining edges of clipping window. Each time resultant list of vertices is successively passed to process next edge of clipping window.

Step 7: Stop.

Program

```
#include<stdio.h>
#include<graphics.h>
#include<conio.h>
#include<stdlib.h>
int main()
{
int gd,gm,n,*x,i,k=0;
int w[]={220,140,420,140,420,340,220,340,220,140};
detectgraph(&gd,&gm);
initgraph(&gd,&gm,"c:\\turboc3\\bgi");
printf("Window:-");
setcolor(RED);
drawpoly(5,w);
printf("Enter the no. of vertices of polygon: ");
scanf("%d",&n);
x = malloc(n*2+1);
printf("Enter the coordinates of points:\n");
k=0;
for(i=0;i<n*2;i+=2)
{
printf("(x%d,y%d): ",k,k);
```

```
scanf("%d,%d",&x[i],&x[i+1]);
k++;
}
x[n*2]=x[0];
x[n*2+1]=x[1];
setcolor(WHITE);
drawpoly(n+1,x);
printf("\nPress a button to clip a polygon..");
getch();
setcolor(RED);
drawpoly(5,w);
setfillstyle(SOLID_FILL,BLACK);
floodfill(2,2,RED);
gotoxy(1,1);
printf("\nThis is the clipped polygon..");
getch();
cleardevice();
closegraph();
return 0;
}
```

BEZIER

Algorithm

1. Initialize graphics library

2. Prompt the user to input the x and y coordinates of four control points for the Bezier curve. Store these coordinates in two arrays - x[] and y[].

3. Plot the four control points on the graphics window using the putpixel() function.

4.

put_x = pow(1-t,3)x[0] + 3tpow(1-t,2)x[1] + 3tt*(1-t)x[2] + pow(t,3)x[3]

put_y = pow(1-t,3)y[0] + 3tpow(1-t,2)y[1] + 3tt*(1-t)*y[2] + pow(t,3)*y[3]

5. For each point generated in the previous step, plot it on the graphics window using the putpixel() function.

6. Close the graphics window

7. Stop

Fractal

Algorithm

1. Initialize graphics library

2. Define a function drawfern() that takes five parameters - x, y, l, arg, and n - and recursively draws a fractal fern based on these parameters.

3. Set initial values for x, y, l, and a, and call drawfern() with these values and an initial value of n.

4. Wait for user input using the getch() function, which pauses the program until a key is pressed.

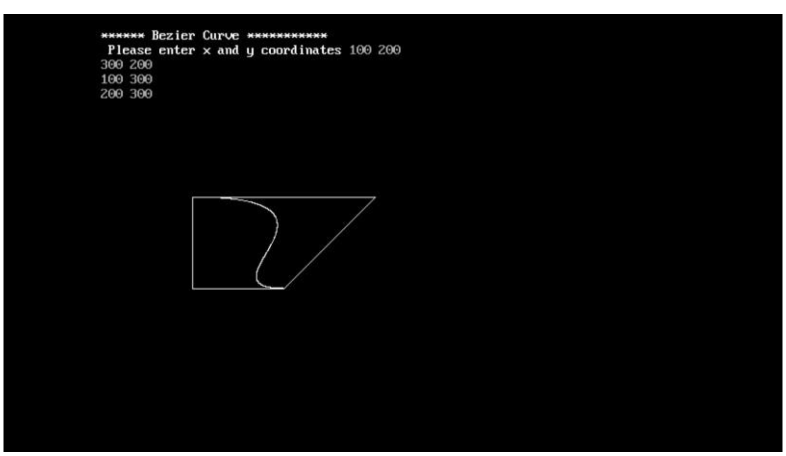
5. Stop

Program with Output:

Bezier curve for n control points:

```
#include<graphics.h>
#include<math.h>
#include<conio.h>
#include<stdio.h>
void main()
{
int  x[4],y[4],i;
double put_x,put_y,t;
int gr=DETECT,gm;
initgraph(&gr,&gm,"C:\\TURBOC3\\BGI");
printf("\n****** Bezier Curve ***********");
printf("\n Please enter x and y coordinates ");
for(i=0;i<4;i++)
{
scanf("%d%d",&x[i],&y[i]);
putpixel(x[i],y[i],3);                // Control Points
}

for(t=0.0;t<=1.0;t=t+0.001)             // t always lies between 0 and 1
{
put_x = pow(1-t,3)*x[0] + 3*t*pow(1-t,2)*x[1] + 3*t*t*(1-t)*x[2] + pow(t,3)*x[3]; // Formula to draw curve
put_y =  pow(1-t,3)*y[0] + 3*t*pow(1-t,2)*y[1] + 3*t*t*(1-t)*y[2] + pow(t,3)*y[3];
putpixel(put_x,put_y, WHITE);            // putting pixel
}
getch();
closegraph();
}
```

Output:



Program with Output:

Fractals:

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

int a;
void drawfern(int x,int y,int l,int arg,int n)
{
int x1,y1,i;
int l1,xpt,ypt;

if(n>0&&!kbhit())
 {
 x1=(int)(x-l*sin(arg*3.14/180));
 y1=(int)(y-l*cos(arg*3.14/180));
 line(x,y,x1,y1);
 l1=(int)(l/5);
 for(i=1;i<6;i++)
  {
  xpt=(int)(x-i*l1*sin(arg*3.14/180));
  ypt=(int)(y-i*l1*cos(arg*3.14/180));
  drawfern(xpt,ypt,(int)(l/(i+1)),arg+a,n-1);
  drawfern(xpt,ypt,(int)(l/(i+1)),arg-a,n-1);
  }
 }
}
void main()
{
int gd=DETECT,gm,x,y,l;
initgraph(&gd,&gm,"C:\\TURBOC3\\BGI\\");
x=getmaxx()/2;
y=getmaxy()/2;
l=150;
a=45;
setcolor(YELLOW);
drawfern(x,y,l,0,5);
getch();
}
```