



LAB MANUAL ON COMPUTER GRAPHICS

For BE Computer III/I, BE-IT III/II, Bsc.CSIT II/I and BCA III/I

DECEMBER 9, 2021

PREPARED BY: MUKUNDA PAUDEL

Department of Computer and IT Engineering, EEC Sanepa-2, Lalitpur.

Objectives of Laboratory as course syllabus...

Implementation of various 2D and 3D graphics algorithm covered in the course using C / C++ and OpenGL.

Hardware and Software Requirements for implementing graphics concepts in C-Language

Software Requirement:

- ✓ Turbo C / C++ compiler that supports graphics.h package.
- ✓ special DOS-BOXed installer for Turbo C++ compiler
- ✓ Availability: Freely available and easy download
- ✓ OS: Windows

Minimum hardware requirements:

- ✓ **Processor:** 1GHz or faster with two or more cores on a compatible 64-bit processor or system on a chip (SoC)
- ✓ **Intel chipset** 810 mother board or higher version
- ✓ **Display:** 14'' color monitor or greater than that
- ✓ Mouse
- ✓ Keyboard
- ✓ **Storage:** 100 GB HDD or higher
- ✓ **RAM:** 2 GB or higher

Lab 1: Introduction to Computer Graphics Lab

Objectives:

- ✓ To provide overview of lab task, course objectives, tools used in lab of computer graphics and motivate student for new beginning.
- ✓ To learn the Basic of Computer graphics and inbuilt graphics function to draw different geometrical figures.

Background:

- ❖ In C graphics programming you have to use standard library functions (don't worry if you don't know functions) to get your task done.
- ❖ Just you pass arguments to the functions and it's done.
- ❖ Firstly you should know the function *initgraph()* which is used to initialize the graphics mode.
- ❖ To initialize graphics mode we use *initgraph()* function in our program.
- ❖ *Initgraph()* function is present in "graphics.h" header file, so your every graphics program should include "*graphics.h*" header file.
- ❖ Recall the concept of coordinate geometry, **line** , **circle**, **rectangular and arc**
- ❖ Recall the C program architecture and understand the graphics program architecture.

Simple Graphics program concept

Let's *Draw a circle* first then will draw other mathematical figures.

- ✓ To draw circle we have inbuilt function *circle()*
- ✓ *Circle()* takes 2 parameters as center and radius
- ✓ Center has two coordinate (x_c, y_c) and radius of defined unit

Similarly,

To draw line, *line*(x_1, y_1, x_2, y_2)

To draw rectangle, *rectangle* (coordinate of lower left end, coordinate of upper right end) e.g. *rectangle* ($x_{min}, y_{min}, x_{max}, y_{max}$)

To draw arc, *arc* (starting point, angle, end point)

Sample Program

```
#include<graphics.h>
#include<conio.h>
void main()
{
// this line is must for every graphics program

int gd = DETECT, gm;

// this line is must for every graphics program and path is defined as Turbo
installation.

// declaration of any variables must be done before calling initgraph()function.

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

circle(200,100,150);
getch();
closegraph();
}
```

Code Explanation:

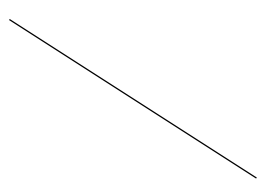
- ✓ this program initializes graphics mode and then closes it after a key is pressed
- ✓ We have declared two variables of **int** type **gd** and **gm** for graphics driver and graphics mode respectively, you can choose any other variable name as well.
- ✓ **DETECT** is a macro defined in "graphics.h" header file,
- ✓ then we have passed three arguments to initgraph() function first is the address of gd, second is the address of gm and third is the path where your BGI files are present (you have to adjust this accordingly where you Turbo C compiler is installed).
- ✓ Initgraph function automatically decides an appropriate graphics driver and mode such that maximum screen resolution is set
- ✓ Initgraph() function initializes the graphics mode and clears the screen
- ✓ getch() helps us to wait until a key is pressed,
- ✓ closegraph() function closes the graphics mode, and finally

- ✓ After you have understood `initgraph()` function then you can use functions to draw shapes such as circle, line, rectangle, etc.

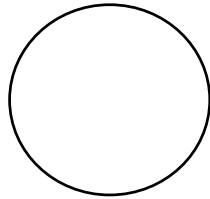
After these all concept and background..

Objective-2

Write a program to draw following geometrical shapes



Line



Circle

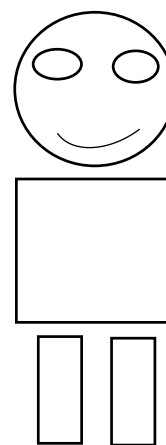
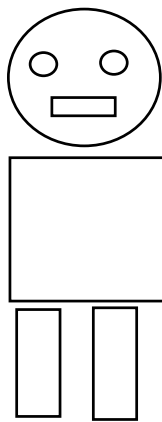


Rectangular



Arc

Using these all shapes lets create following Figure:



Program:

```
#include<stdio.h>
#include<conio.h>
#include<graphics.h>
Void main ()
{
int gd=DETECT,gm; (A)
```

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

```
circle (175,110,50);
```

```
circle (150,110,10);
```

```
circle (200,100,10);
```

```
rectangle (125,170,225,230); (B)
```

```
rectangle (125,240,150,290);(B)      getch(); (C)  
rectangle (200,240,225,290);        closegraph();  
arc (175,110,195,345,30); (C)      }
```

Output:

Lab 2: Digital Differential Analyzer (DDA) Algorithm

Objectives: Implementation of DDA Algorithm to draw a straight line.

DDA

- ❖ Digital differential Analyzer is a line drawing algorithm which calculates and plots coordinates on the basis of the previously calculated intermediate points until it reaches to the final point.
- ❖ DDA algorithm works on the concept of the slope-intercept equation.

Algorithm:

1. **Start**
2. Define an initial point (x0, y0) and a final point (x1, y1) to get the input from the user regarding the initial and final point.
3. After getting the input, calculate the value of dx and dy. (d represents the difference between two points)

$$\begin{aligned} dx &= x1 - x0, \\ dy &= y1 - y0 \end{aligned}$$

4. Find the slope of the line by using:

$$\text{Slope (m)} = dy/dx$$

- 4.1 **If** Slope (m) > 1 then do,

While x isn't equivalent to the final point (x1)

Do, $x = x + (1/m)$

$y = y + 1$

plot(x,y)

End while

else

While y is not equivalent to the final (y1) do,

$y = y + m$

$x = x + 1$

plot(x,y)**5. End****Program:**

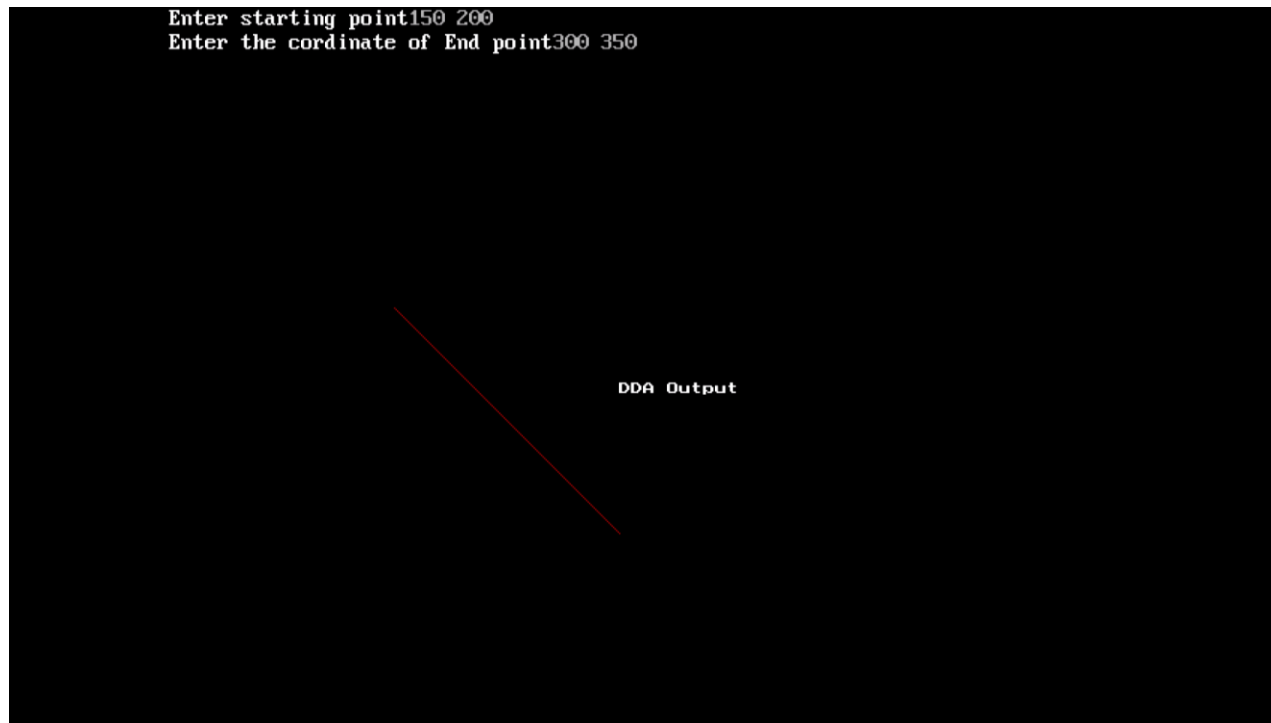
```

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

Void main ()
{
int x,y,x1,y1,i;
float m,dx,dy;
int gd=DETECT,gm;
initgraph(&gd,&gm,"C:\\TURBOC3\\BGI");
printf("Enter starting point");
scanf("%d%d",&x,&y);
printf("Enter the cordinate of End point");
scanf("%d%d", &x1,&y1);
dx= abs(x1-x);
dy= abs(y1-y);
m=dy/dx;
if (m>1.0)
{ (A) while(y<=y1)
{
putpixel(x,y,RED);
x=x+(1/m);
y=y+1.0;
}
}
else
{
While (x<=x1)
{
putpixel(x,y,RED);
y=y+m;
x=x+1.0;
}
}
Outtextxy(300,250,"DDA Output");
getch();
closegraph();
}

```


Output:



Lab 3: Bresenham's Line Drawing Algorithm

Objectives: Implementation of Bresenham's Line drawing algorithm to draw a straight line

Bresenham Line Drawing Algorithm

- ❖ Bresenham's line drawing algorithm is an efficient scan conversion algorithm to draw any line.
- ❖ BLA determines which order to form a close approximation to a straight line between two given points.
- ❖ Unlike DDA algorithm, it uses only integer incremental calculations during the process which saves much of the CPU time.

Algorithm: (for slope $(m) < 1$)

1. Start
2. Define two end points of line as (x_0, y_0) and (x_1, y_1) , decision parameter (p) , x , y , dx and dy . (if needed define other accordingly)
3. Calculate $dx = x_1 - x_0$ and $dy = y_1 - y_0$ and set $x = x_0$ and $y = y_0$
4. Check $(x < x_1)$

4.1 if $(p \geq 0)$

```
{
Plot (x,y);
x=x+1;
y=y+1;
p=p+2*dy-2*dx;
}
```

4.2 if $(p < 0)$

```
{
Plot (x,y);
x=x+1;
y=y;
p=p+2*dy;
}
```

5. Repeat step 3 until checked condition is valid.
6. End

Note: The algorithm above assumes *slopes are less than 1*. For other slopes we need to adjust the algorithm slightly

Program:

```

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

void BLA(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,RED);
            y=y+1;
            p=p+2*dy-2*dx;
        }
        else
        {
            Putpixel (x,y,RED);

```

```

        p=p+2*dy;
        }
        x=x+1;
        }
    }

    Void main()
    {
        int gdriver=DETECT, gmode, error,
        x0, y0, x1, y1;
        initgraph(&gdriver, &gmode,
        "C:\\TURBOC3\\BGI");

        printf("Enter the Co-ordinates of
        Starting point \n ");
        scanf("%d%d", &x0, &y0);

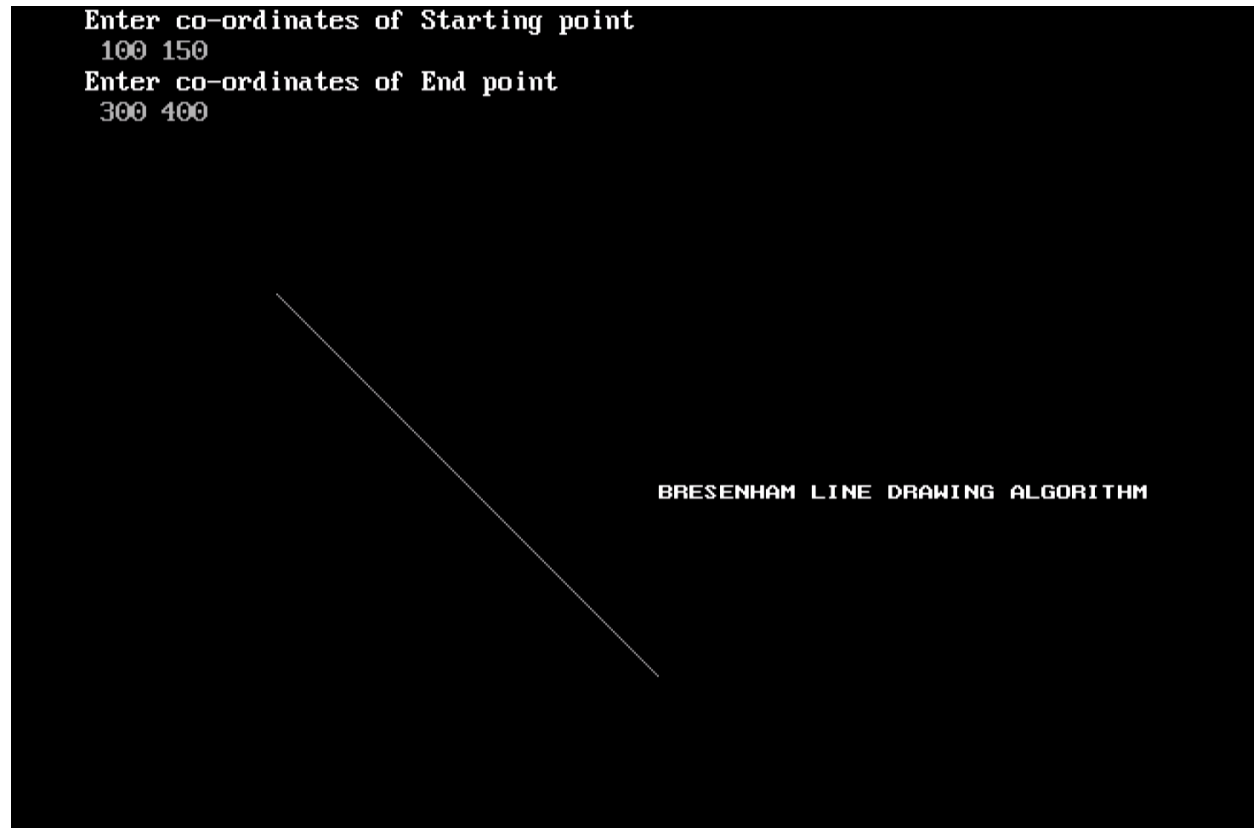
        printf("Enter the Co-ordinates of End
        point \n ");
        scanf("%d%d", &x1, &y1);
        BLA(x0, y0, x1, y1);
        Outtextxy(300,250, "BRESENHAM
        LINE DRAWING ALGORITHM");
        getch();
        closegraph();
    }

```

- Compile the above code: **Alt + F9**
- If any bug → Debug
- Run the project → **ctrl + F9**
- Find the following output

OUTPUT:

```
Enter co-ordinates of Starting point
100 150
Enter co-ordinates of End point
300 400
```



BRESENHAM LINE DRAWING ALGORITHM

Note:

For the slope (**m**) >1, you can implement similar concept. Just change the value of P_o , p_{k+1} , x_{k+1} and y_{k+1}

OR

You can make one program for both slope condition in following way:

| | |
|---------------------------------|-------------------------------------|
| if ($m < 1$) | else |
| { | { |
| // write the above program code | //write the code in similar way for |
| } | slope greater than 1. |
| | } |

LAB 4: Mid-Point Circle Drawing Algorithm

Objectives: Implementation of Circle Drawing Algorithm to draw a straight line.

- ❖ Circle drawing algorithm is based on 8 way symmetry property of circle.
- ❖ Every circle has 8 octants and the circle drawing algorithm generates all the points for one octant.
- ❖ The points for other 7 octants are generated by changing the sign towards X and Y coordinates.
- ❖ To take the advantage of 8 symmetry property, the circle must be formed assuming that the Centre point coordinates is (0, 0).
- ❖ If the Centre coordinates are other than (0, 0), then we add the X and Y coordinate values with each point of circle with the coordinate values generated by assuming (0, 0) as Centre point.

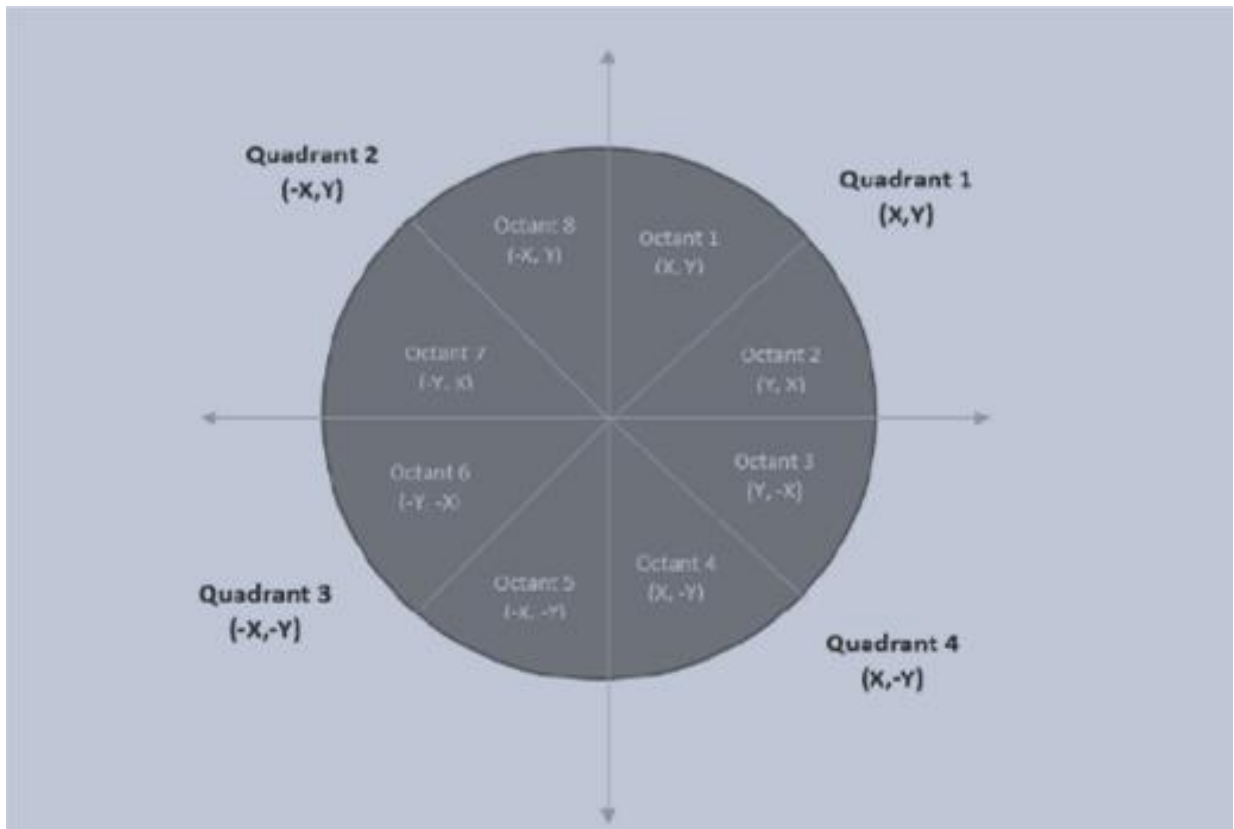


Figure: symmetric property of circle

Algorithm**Start**

1. Declare the Centre of Circle as (X_0, Y_0) and Radius of Circle r
2. Assign the starting point coordinates (X_0, Y_0) as

$$X_0 = 0$$

$$Y_0 = r$$

3. Calculate the value of initial decision parameter P_0 as-

$$P_0 = 1 - r$$

4. Suppose the current point is (X_k, Y_k) and the next point is (X_{k+1}, Y_{k+1}) . Find the next point of the first octant depending on the value of decision parameter P_k .

If $P_k < 0$, then

$$X_{k+1} = X_k + 1 \text{ and } Y_{k+1} = Y_k \text{ and}$$

$$P_{k+1} = P_k + 2X_k + 1$$

Else,

$$X_{k+1} = X_k + 1 \text{ and } Y_{k+1} = Y_k - 1$$

$$P_{k+1} = P_k + 2X_k - 2Y_k + 1$$

5. If the given centre point (X_0, Y_0) is not $(0, 0)$, then do the following and plot the point

$$X_{\text{plot}} = X_c + X_0$$

$$Y_{\text{plot}} = Y_c + Y_0$$

Here, (X_c, Y_c) denotes the current value of X and Y coordinates.

6. Repeat Step 4 and Step 5 until $X_{\text{plot}} \geq Y_{\text{plot}}$.

7. Generates all the points for one octant.

8. End

To find the points for other seven octants, see the above figure of eight way symmetry property of circle.

Program:

```
#include<stdio.h>

#include<graphics.h>

#include<dos.h>

Void MPCA (int xc, int yc, int r)

{

int p,x,y;

p=1-r;

x=0;

y=r;

While(x<=y)

{

putpixel(xc+x,yc+y,5);

putpixel(xc-y,yc-x,5);

putpixel(xc+y,yc-x,5);

putpixel(xc-y,yc+x,5);
```

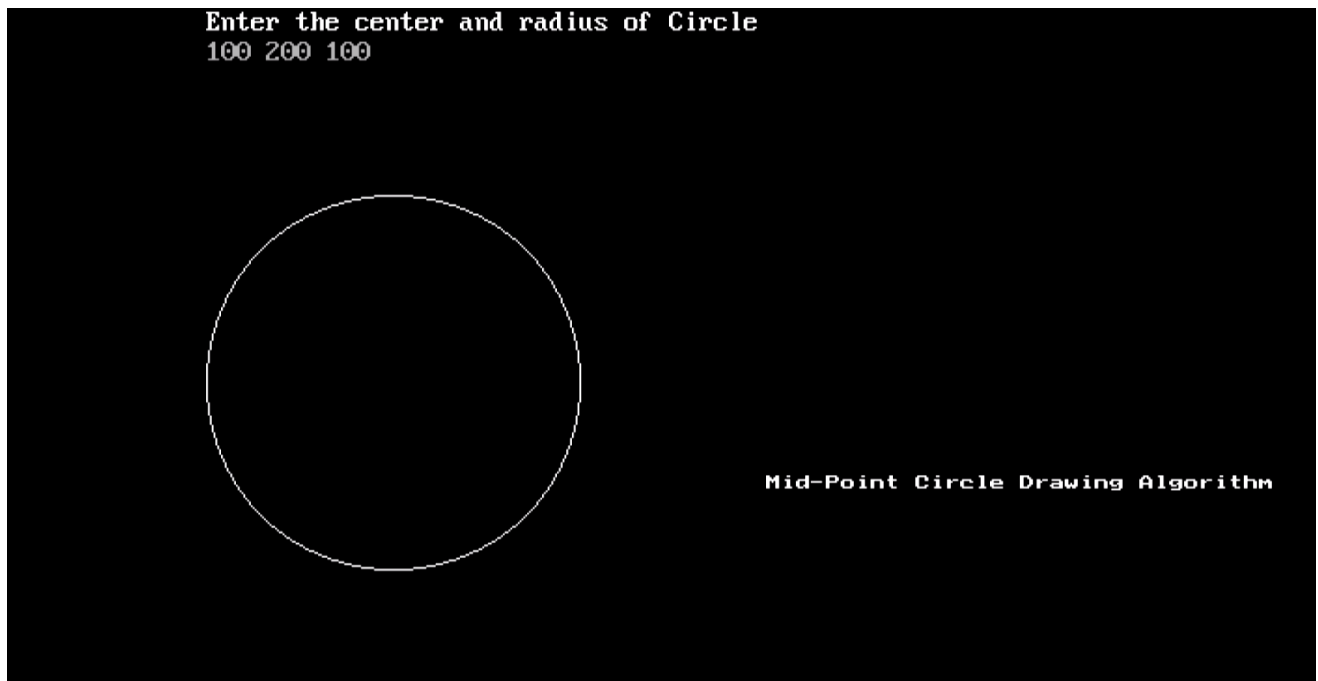
```
putpixel(xc+y,yc+x,5);
putpixel(xc-x,yc-y,5);
putpixel(xc+x,yc-y,5);
putpixel(xc-x,yc+y,5);
delay (200);
if (p>0)
{
    p=p+2*(x+1)-2*(y+1)+1;
    x++;
    y--; }
else
{
    p=p+2*(x+1)+1;
    x++;
}
}
}

Void main()
{
    int gd=DETECT,gm;
    int xc,yc,r;
```



```
initgraph (&gd,&gm,"C:\\TURBOC3\\BGI");  
printf ("Enter the center and radius of the Circle \n");  
scanf ("%d%d%d",&xc,&yc,&r);  
MPCA (xc,yc,r);  
Outtextxy (300, 250,"Mid-Point Circle Drawing Algorithm");  
getch();  
}
```

OUTPUT:



LAB 5: Ellipse Generation Algorithm

Objectives: To implement the Ellipse Generation Algorithm for drawing an ellipse of given center (x, y) and radius r_x and r_y .

In Ellipse, Symmetry between quadrants exists not symmetric between the two octants of a quadrant. Thus, we must calculate pixel positions along the elliptical arc through one quadrant and then we obtain positions in the remaining 3 quadrants by symmetry. The next pixel is chosen based on the decision parameter.

Algorithm:

Start

1. Input r_x , r_y , and ellipse center (x_c, y_c) , and obtain the first point on an ellipse centered on the origin as $(x_0, y_0) = (0, r_y)$
2. Calculate the initial parameter in region 1 as
 - 2.1. At each x_i position, starting at $i = 0$, if $p1_i < 0$, the next point along the ellipse centered on $(0, 0)$ is $(x_i + 1, y_i)$ and otherwise, the next point is $(x_i + 1, y_i - 1)$ and continue until (x_0, y_0) is the last position calculated in region 1.
3. Calculate the initial parameter in region 2 as
 - 3.1. At each y_i position, starting at $i = 0$, if $p2_i > 0$, the next point along the ellipse centered on $(0, 0)$ is $(x_i, y_i - 1)$ and otherwise, the next point is $(x_i + 1, y_i - 1)$ and Use the same incremental calculations as in region 1. Continue until $y = 0$.
4. For both regions determine symmetry points in the other three quadrants.
5. Move each calculated pixel position (x, y) onto the elliptical path centered on (x_c, y_c) and plot the coordinate values $x = x + x_c$, $y = y + y_c$.
6. End

Program:

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

void main()
{
int gd=DETECT,gm;
float p,x,y,xc,yc,a,b;
initgraph(&gd,&gm,"C:\\TURBOC3\\BGI");
cleardevice();
printf("Enter the Center of Ellipse xc and yc:\n");
scanf("%f%f",&xc,&yc);
printf("Enter major axis as a, and minor axis as b:\n");
scanf("%f%f",&a,&b);
```

```
x=0;
```

```
y=b;
```

//Region 1

```
p=(b*b)-(a*a*b)+(0.25*a*a);
```

```
do
{
putpixel(xc+x,yc+y,WHITE);
putpixel(xc+x,yc-y,WHITE);
putpixel(xc-x,yc+y,WHITE);
putpixel(xc-x,yc-y,WHITE);
```

```
if(p<0)
{
x=x+1;
p=p+2*b*b*x+b*b;
}
else
{
x=x+1;
```

```

y=y-1;
p=p+2*b*b*x-2*a*a*y+b*b;
}
}
while(2*b*b*x<2*a*a*y);

```

//Region 2

```

p=(b*b*(x+0.5)*(x+0.5))+((y-1)*(y-1)*a*a-a*a*b*b);

do
{
putpixel(xc+x,yc+y,WHITE);
putpixel(xc+x,yc-y,WHITE);
putpixel(xc-x,yc+y,WHITE);
putpixel(xc-x,yc-y,WHITE);


if(p>0)
{
y=y-1;
p=p-2*a*a*y+a*a;
}
else
{
x=x+1;
y=y-1;
p=p-2*a*a*y+2*b*b*x+a*a;
}
}
while(y!=0);
outtextxy(300,250, "Ellipse Generation Algorithm");
getch();
closegraph();
restorecrtmode();
}

```

OUTPUT:**Enter,****Xc and yc = 300 and 300****Xr and yr = 60 and 40**

```
Enter the Center of Ellipse xc and yc:
300 300
Enter major axis as a, and minor axis as b:
60 40

                                     Ellipse Generation Algorithm
```



LAB 6: Boundary Fill and Flood Algorithm

Boundary Fill

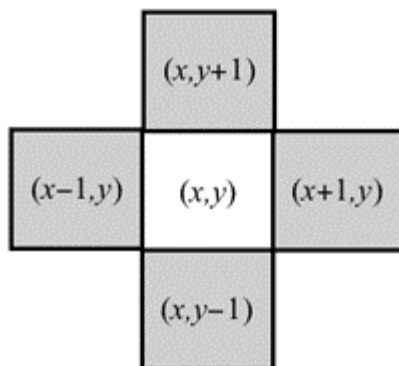
- ❖ Boundary Fill is seed fill algorithm in which edges of the polygon are drawn.
- ❖ Starting with some seed any point inside the polygon we examine the neighboring pixels to check whether the boundary pixel is reached.
- ❖ If boundary pixels are not reached, pixels are highlighted and process is continued until boundary pixels are reached.

Flood Fill

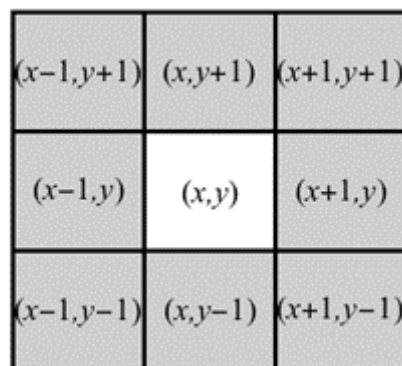
- ❖ Flood Fill is a seed fill algorithm similar to Boundary Fill algorithm but sometimes when it is required to fill in an area that is not defined within a single color boundary we use flood fill instead of boundary fill.
- ❖ In Flood Fill algorithm we start with some seed and examine the neighboring pixels, however pixels are checked for a specified interior color instead of boundary color and is replaced by a new color.

Both the algorithms can be implemented by using 4 connected or 8 connected region method.

4-Connected



8-Connected



Algorithms:**Boundary Fill Algorithm****Start**

1. Create a function named as boundaryfill with 4 parameters (x,y,fill_color,boundary_color).

```

Void boundaryfill(int x,int y,int fill_color,int boundary_color)
{
If (getpixel(x,y)!=boundary_color && getpixel(x,y)!=fill_color)
{
putpixel(x,y,f_color);
boundaryfill(x+1,y,fill_color, boundary _color);
boundaryfill(x,y+1,fill_color, boundary _color);
boundaryfill(x-1,y,fill_color, boundary _color);
boundaryfill(x,y-1,fill_color, boundary _color);
}
}
//getpixel(x,y) gives the color of specified pixel

```

2. Call it recursively until the boundary pixels are reached.

3. End.**Flood Fill Algorithm****Start**

1. Create a function called as **floodFill** (x,y,oldcolor,newcolor)

```

void floodFill(int x,int y,int oldcolor,int newcolor)
{
if(getpixel(x,y) == oldcolor)
{
putpixel(x,y,newcolor);
floodFill(x+1,y,oldcolor,newcolor);
floodFill(x,y+1,oldcolor,newcolor);
}
}

```

```
        floodFill(x-1,y,oldcolor,newcolor);
        floodFill(x,y-1,oldcolor,newcolor);
    }
}
```

2. Repeat until the polygon is completely filled.

3. End

Program for Boundary Fill

```
#include<stdio.h>

#include<graphics.h>

#include<dos.h>

void boundaryfill(int x,int y,int fill_color,int boundary_color)
{
    if(getpixel(x,y)!=boundary_color && getpixel(x,y)!=fill_color)
    {
        delay(20);

        putpixel(x,y,fill_color);

        boundaryfill(x+1,y,fill_color,boundary_color);

        boundaryfill(x,y+1,fill_color,boundary_color);

        boundaryfill(x-1,y,fill_color,boundary_color);

        boundaryfill(x,y-1,fill_color,boundary_color);

    }
}
```

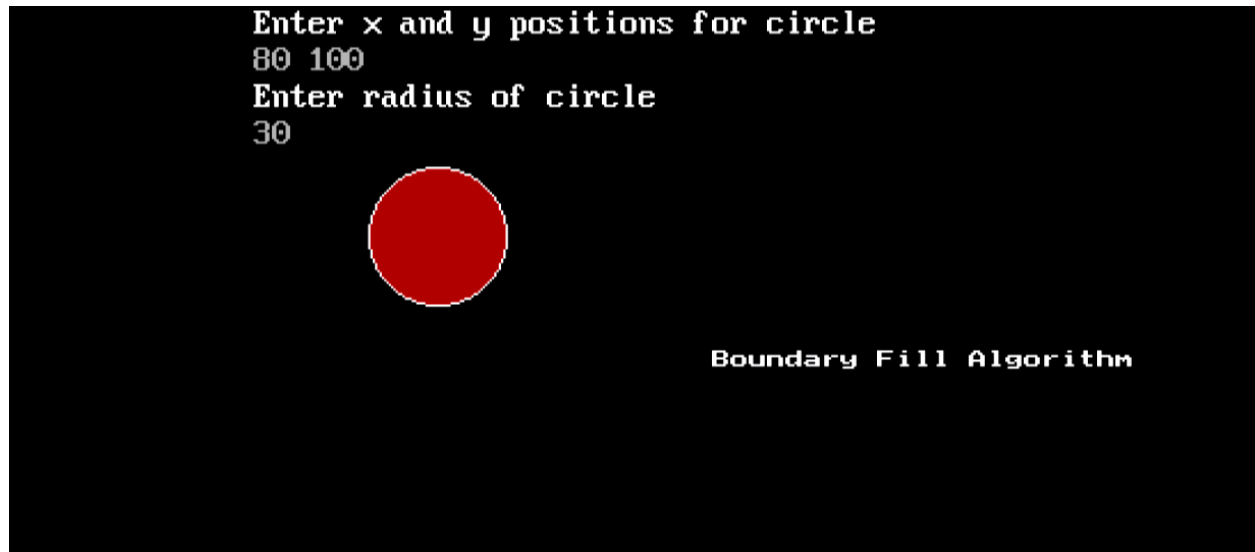


```
int main()
{
int gd=DETECT,gm;
int x,y,r;
initgraph(&gd,&gm,"C:\\TURBOC3\\BGI");
printf("Enter x and y positions for circle\n");
scanf("%d%d",&x,&y);
printf("Enter radius of circle\n");
scanf("%d",&r);
circle(x,y,r);
boundaryfill(x,y,4,15);
outtextxy(200,150, "Boundary Fill")
getch();
closegraph();
return 0;
}
```

Note:

Use radius of circle less than 40. Because this is recursive approach. Due to repeated recursion Dosbox runs out of memory & crashes. Use a Stack/Queue to store neighboring pixels to avoid recursion.

OR, you can use Dev C++.

OUTPUT:

Similarly,

- ✓ Implement Flood Fill algorithm yourself.

LAB 7: creation of different types of texts and fonts

Objectives: To write a program for displaying text in different sizes, different colors and different font styles by using different graphics functions available for text formatting in C.

Description:

The following graphics functions are available for text formatting in C.

1. **Outtext()** → for displaying text on output screen
2. **Outtextxy()** → for displaying text on output screen at location (x,y).
3. **Settextstyle()** → used for specifying font style, font display direction(vertical or horizontal), font size.
4. **Setcolor()** → used to change the current drawing color.e.g. setcolor(RED) or setcolor(4) changes the current drawing color to RED. Remember that default drawing color is WHITE.

Outtext

- ❖ outtext function displays text at current position.
- ❖ **Eg:** outtext("To display text at a particular position on the screen use outtextxy");

Outtextxy()

- ❖ outtextxy function display text or string at a specified point(x,y) on the screen.
- ❖ **Eg:** outtextxy(100, 100, "Outtextxy function");--displays the message "Outtextxy function" At screen location of (100, 100).

Settextstyle()

- ❖ Settextstyle function is used to change the way in which text appears, using it we can modify the size of text, change direction of text and change the font of text.
- ❖ **Eg:** settextstyle(font,direction,charsize);--
settextstyle(TRIPLEX_FONT,HORIZ_DIR,2);

Font –font style- it may be font name or integer value from 0- 9.

Different fonts are:
size(0-9)

DEFAULT_FONT,
TRIPLEX_FONT,
SMALL_FONT,
SANS_SERIF_FONT,
GOTHIC_FONT,
SCRIPT_FONT,
SIMPLEX_FONT,
TRIPLEX_SCR_FONT,
COMPLEX_FONT,
EUROPEAN_FONT,
BOLD_FONT

Directions are two

1. Horizontal direction (HORIZ_DIR or 0)
2. Vertical direction (VERT_DIR or 1)

Program:

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

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

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

```
Void main()
```

```
{
```

```
int gd=DETECT,gm;
```

```
int x=25,y=25,font=10;
```

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

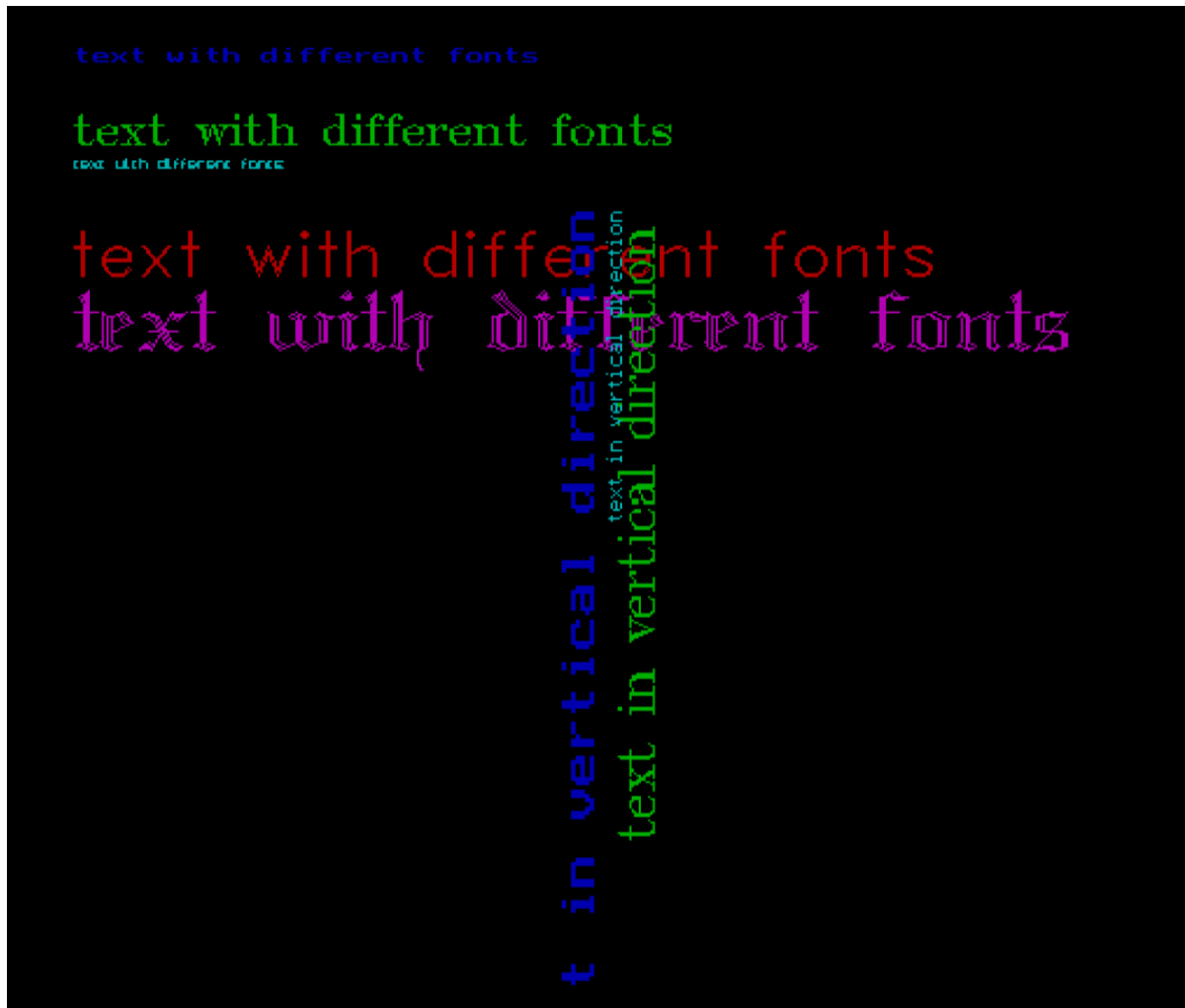
```
for (font=0;font<=4;font++)
```

```
{
```

```
settextstyle(font,HORIZ_DIR,font+1);// sets font type, font direction, size
```

```
setcolor(font+1); // sets color for text.  
  
outtextxy(x,y,"text with different fonts"); // prints message on screen at (x,y)  
  
y=y+25;  
  
}  
  
for (font=0;font<=2;font++)  
  
{  
  
settextstyle(font,VERT_DIR,font+2);  
  
setcolor(font+1);  
  
x=250;  
  
y=100;  
  
outtextxy(x,y,"text in vertical direction");  
  
y=y+25;  
  
}  
  
getch();  
  
closegraph();  
  
}
```

OUTPUT:



LAB 8: Scan Line Polygon Fill Algorithm

Objectives: To implement the Scan line polygon fill algorithm for coloring a given object.

The basic scan-line algorithm is as follows:

Step-1 Find the intersections of the scan line with all edges of the polygon

Step-2 Sort the intersections by increasing x coordinate

Step-3 Fill in all pixels between pairs of intersections that lie interior to the polygon

Process involved:

The scan-line polygon-filling algorithm involves

- ❖ the horizontal scanning of the polygon from its lowermost to its topmost vertex,
- ❖ identifying which edges intersect the scan-line, and
- ❖ Finally drawing the interior horizontal lines with the specified fill color process.

Algorithm Steps

1. The horizontal scanning of the polygon from its lowermost to its topmost vertex
2. Identify the edge intersections of scan line with polygon
3. Build the edge table
 - 3.1. Each entry in the table for a particular scan line contains the maximum y value for that edge, the x-intercept value (at the lower vertex) for the edge, and the inverse slope of the edge.
4. Determine whether any edges need to be splitted or not. If there is need to split, split the edges.
5. Add new edges and build modified edge table.
6. Build Active edge table for each scan line and fill the polygon based on intersection of scan line with polygon edges.

Program:

```
#include <stdio.h>

#include <conio.h>

#include <graphics.h>

void main()

{

int n,i,j,k,gd,gm,dy,dx;

int x,y,temp;

int a[20][2],xi[20];

float slope[20];

clrscr();

printf("\n\n\tEnter the no. of edges of polygon : ");

scanf("%d",&n);

printf("\n\n\tEnter the cordinales of polygon :\n\n\n ");

for(i=0;i<n;i++)

{

printf("\tX%d Y%d : ",i,i);

scanf("%d %d",&a[i][0],&a[i][1]);

}

a[n][0]=a[0][0];

a[n][1]=a[0][1];
```



```
detectgraph(&gd,&gm);

initgraph(&gd,&gm,"C:\\TurboC3\\BGI");

/*- draw polygon -*/

for(i=0;i<n;i++)

{

line(a[i][0],a[i][1],a[i+1][0],a[i+1][1]);

}

getch();

for(i=0;i<n;i++)

{

dy=a[i+1][1]-a[i][1];

dx=a[i+1][0]-a[i][0];

if(dy==0) slope[i]=1.0;

if(dx==0) slope[i]=0.0;

if((dy!=0)&&(dx!=0)) /*- calculate inverse slope -*/

{

slope[i]=(float) dx/dy;

}

}

for(y=0;y< 480;y++)

{
```

```
k=0;

for(i=0;i<n;i++)

    {

    if( ((a[i][1]<=y)&&(a[i+1][1]>y))||

    ((a[i][1]>y)&&(a[i+1][1]<=y)))

    {

    xi[k]=(int)(a[i][0]+slope[i]*(y-a[i][1]));

    k++;

    }

    }

for(j=0;j<k-1;j++) /*- Arrange x-intersections in order -*/

for(i=0;i<k-1;i++)

{

if(xi[i]>xi[i+1])

{

temp=xi[i];

xi[i]=xi[i+1];

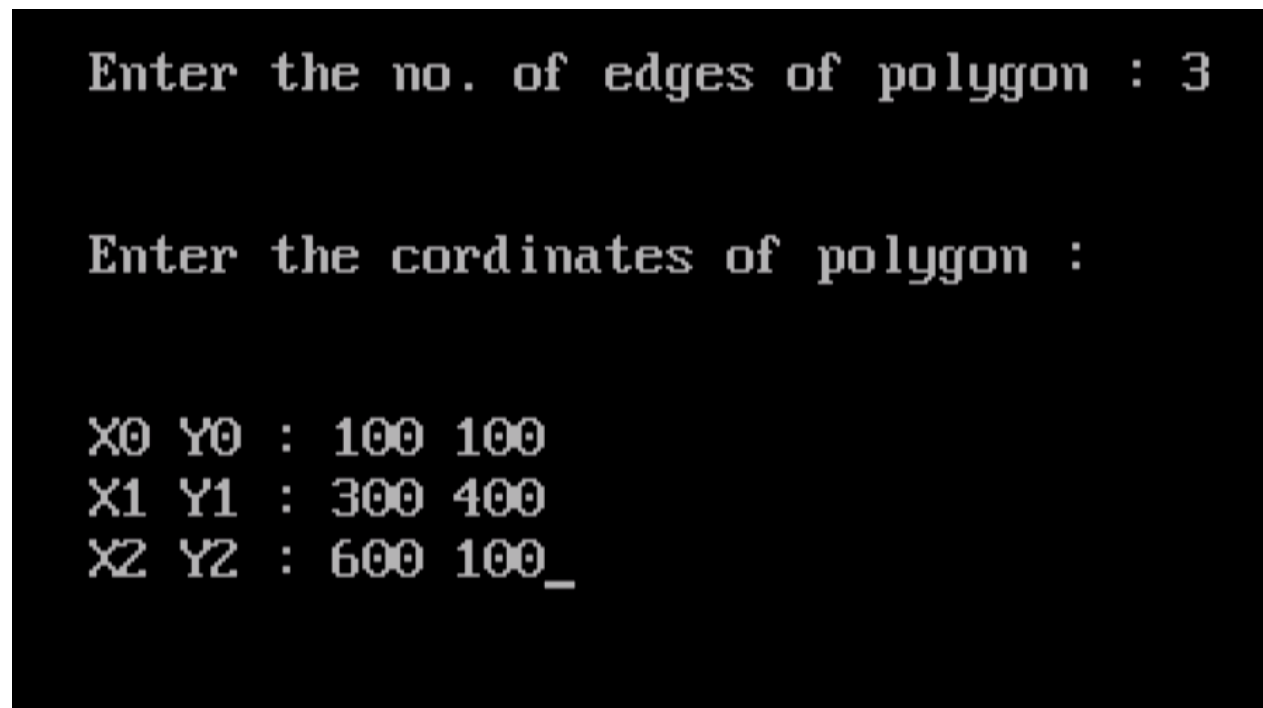
xi[i+1]=temp;

}

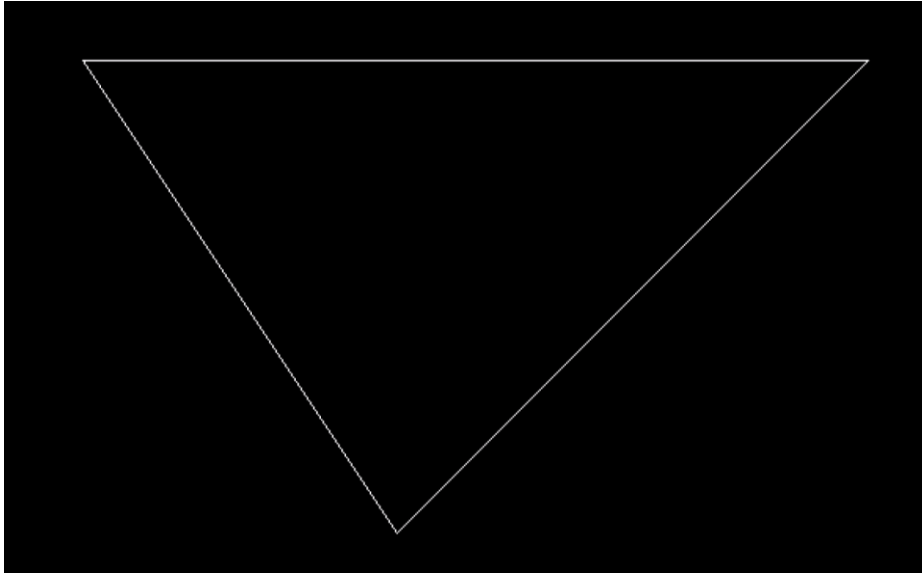
}

setcolor(3);
```

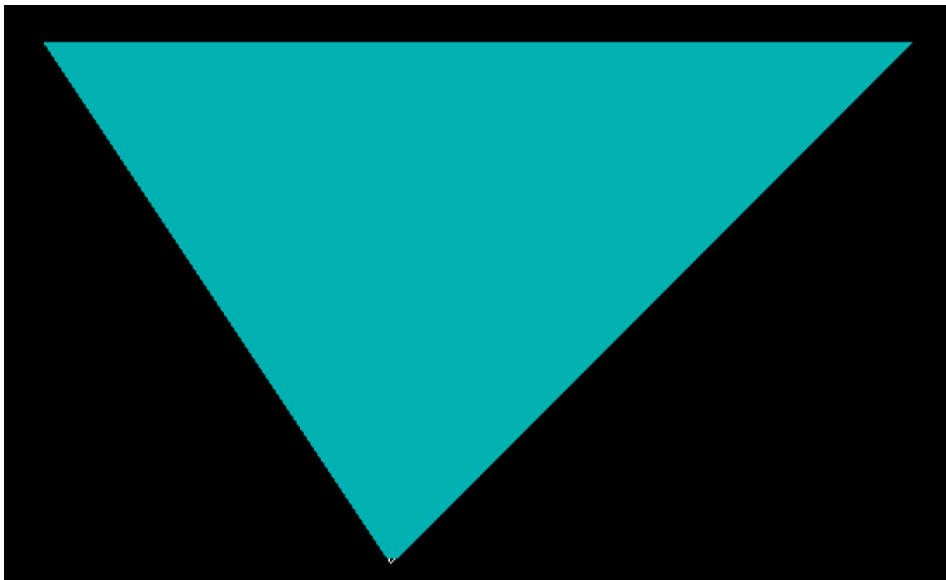
```
for(i=0;i<k;i+=2)
{
line(xi[i],y,xi[i+1]+1,y);
getch();
}
}
}
```

OUTPUT:

When you press Enter Key you get following polygon (i.e. Triangle)



Now, press Enter key continuously or click again and again and see following filled polygon. (try to visualize scan line polygon fill algorithm)



- Try next polygon of your choice. **Or** use following coordinates and visualize scan line polygon fill algorithm.

Number of edge = 6

(X_0, Y_0) to $(X_6, Y_6) \rightarrow (100, 200), (200, 200), (230, 300), (200, 400), (100, 400), (70, 300)$ respectively.

LAB 8: Two Dimensional (2D) Transformation

Objectives: To perform 2D Basic transformations (i.e. Translation, Scaling, and Rotation) on 2D object.

Transformations are helpful in changing the position, size, orientation, shape etc. of the object.

Translation

- ❖ 2D Translation is a process of moving an object from one position to another in a two dimensional plane.
- ❖ We translate two dimensional point by adding translation distance along x axis $\rightarrow t_x$ and translation distance along y axis $\rightarrow t_y$ to the original coordinate position (x, y) to move at new position (x', y') as:

$$\begin{aligned} x' &= x + t_x \\ \& \quad y' &= y + t_y \end{aligned}$$

Scaling

- ❖ It is a transformation that used to alter the size of an object.
- ❖ This operation is carried out by multiplying coordinate value (x, y) with scaling factor (sx, sy) respectively.

So, equation for scaling is given by:

$$\begin{aligned} x' &= x \cdot s_x \\ y' &= y \cdot s_y \end{aligned}$$

Rotation

- ❖ Consider a point $P(x, y)$ is original point and r is the constant distance from origin and θ is the original angular displacement from x-axis.
- ❖ Rotate $P(x, y)$ with an angle θ in anticlockwise direction and point obtained after rotation is $P'(x', y')$

Therefore,

$$\begin{aligned} x' &= x \cos \theta - y \sin \theta \\ y' &= x \sin \theta + y \cos \theta \end{aligned}$$

Algorithm for Basic Transformation**1. Start**

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

3.1. Translation

- 3.1.1 Get the translation value tx, ty
- 3.1.2 Move the 2d object with tx, ty ($x'=x+tx, y'=y+ty$)
- 3.1.3 Plot (x', y')

3.2. Scaling

- 3.2.1 Get the scaling value Sx, Sy
- 3.2.2. Resize the object with Sx, Sy ($x'=x*Sx, y'=y*Sy$)
- 3.2.3 Plot (x', y')

3.3 Rotation

- 3.3.1 Get the Rotation angle
- 3.3.2. Rotate the object by the angle ϕ

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

$$y' = x \sin \phi + y \cos \phi$$
- 3.3.3. Plot (x', y')

4. End**Program**

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <math.h>
void main()
{
    int x1,x2,x3,y1,y2,y3,nx1,nx2,nx3,ny1,ny2,ny3,c;
    int sx,sy,xt,yt,r;
    float t;
    int gd=DETECT,gm;
```

```
initgraph(&gd,&gm,"C:\\TURBOC3\\BGI");
printf("\n\t Enter the points of triangle");
setcolor(WHITE);
scanf("%d%d%d%d%d%d",&x1,&y1,&x2,&y2,&x3,&y3);
line(x1,y1,x2,y2);
line(x2,y2,x3,y3);
line(x3,y3,x1,y1);
getch();
printf("\n 1.Translation\n 2.Rotation\n 3.Scalling\n 4.exit");
printf("Enter your choice:");
scanf("%d",&c);
switch(c)
{
    case 1:
        printf("\n Enter the translation factor");
        scanf("%d%d",&xt,&yt);
        nx1=x1+xt;
        ny1=y1+yt;
        nx2=x2+xt;
        ny2=y2+yt;
        nx3=x3+xt;
        ny3=y3+yt;
        line(nx1,ny1,nx2,ny2);
        line(nx2,ny2,nx3,ny3);
        line(nx3,ny3,nx1,ny1);
```

```
getch();
```

case 2:

```
printf("\n Enter the angle of rotation");
scanf("%d",&r);
t=3.14*r/180;
nx1=abs(x1*cos(t)-y1*sin(t));
ny1=abs(x1*sin(t)+y1*cos(t));
nx2=abs(x2*cos(t)-y2*sin(t));
ny2=abs(x2*sin(t)+y2*cos(t));
nx3=abs(x3*cos(t)-y3*sin(t));
ny3=abs(x3*sin(t)+y3*cos(t));
line(nx1,ny1,nx2,ny2);
line(nx2,ny2,nx3,ny3);
line(nx3,ny3,nx1,ny1);
getch();
```

case 3:

```
printf("\n Enter the scalling factor");
scanf("%d%d",&sx,&sy);
nx1=x1*sx;
ny1=y2*sy;
nx2=x2*sx;
ny2=y2*sy;
nx3=x3*sx;
```

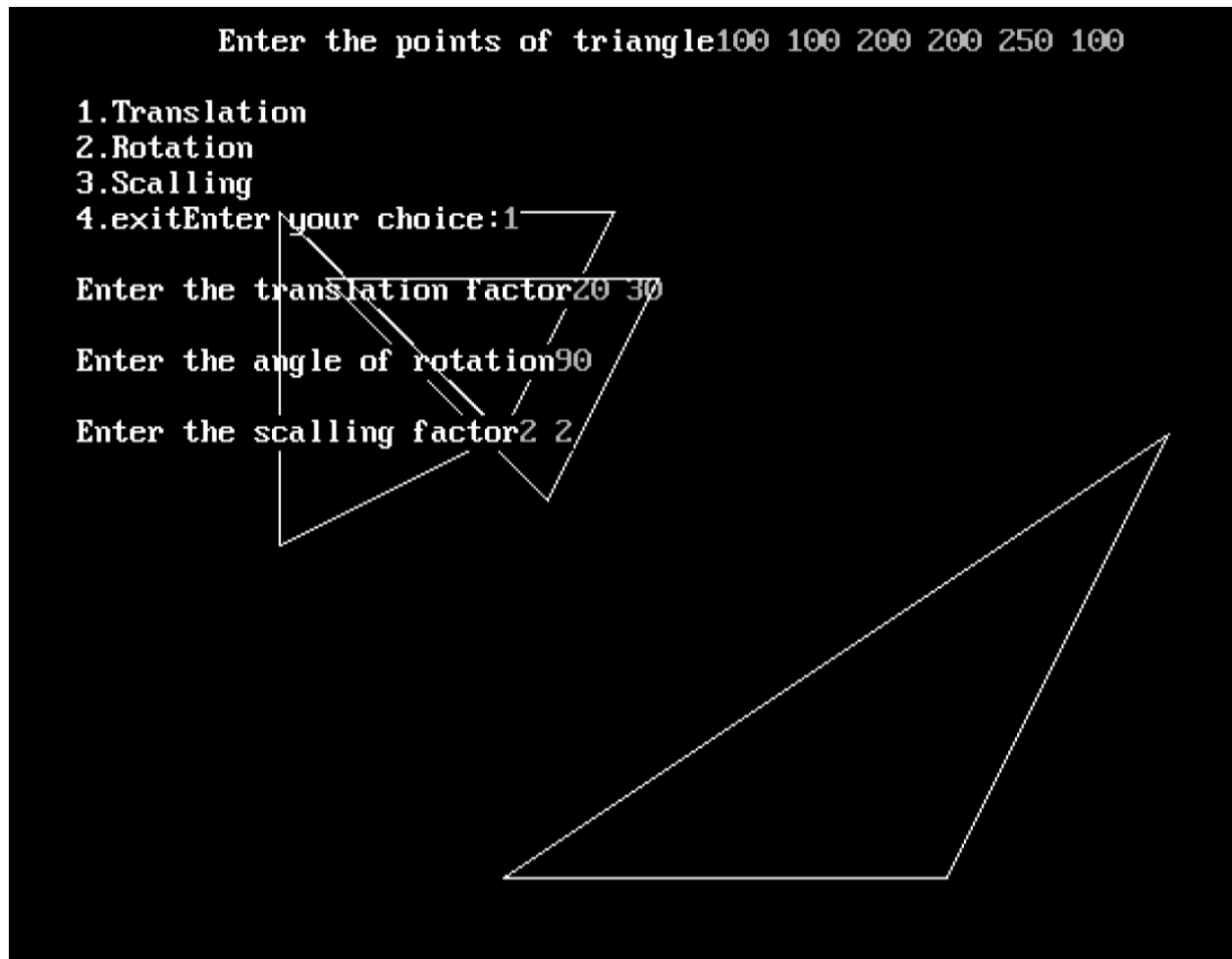


```
ny3=y3*sy;
line(nx1,ny1,nx2,ny2);
line(nx2,ny2,nx3,ny3);
line(nx3,ny3,nx1,ny1);
getch();

case 4:
    break;
default:
    printf("Enter the correct choice");
    }
    closegraph();
    }
```

NOTE: In this way you can add different case for reflection and shearing in above program.

- You can use setcolor() function to draw the transformed object in different color.
- During transformation, don't forget to consider the resolution of your PC.
- If you put large scaling factors or translation distance, translated object may lie outside the screen.

OUTPUT:

LAB 9: Bezier Curve Generation

Objectives: To generate a smooth curve by using Bezier curve technique for a given set of control points.

- ❖ **A Bezier curve** is a parametric curve frequently used in computer graphics and related fields.
- ❖ Generalizations of Bezier curves to higher dimensions are called Bezier surfaces, of which the Bezier triangle is a special case.
- ❖ In vector graphics, Bezier curves are used to model smooth curves that can be scaled indefinitely. "Paths," as they are commonly referred to in image manipulation programs are combinations of linked Bezier curves.
- ❖ Paths are not bound by the limits of rasterized images and are intuitive to modify. Bezier curves are also used in animation as a tool to control motion

Four points **P0**, **P1**, **P2** and **P3** in the plane or in higher-dimensional space define a cubic Bezier curve. The curve starts at **P0** going toward **P1** and arrives at **P3** coming from the direction of **P2**. Usually, it will not pass through **P1** or **P2**; these points are only there to provide directional information. The distance between **P0** and **P1** determines "how long" the curve moves into direction **P2** before turning towards **P3**.

Parametric equation for cubic Bezier curve is:

$$\mathbf{P}(t) = \mathbf{B}_0(1-t)^3 + \mathbf{B}_13t(1-t)^2 + \mathbf{B}_23t^2(1-t) + \mathbf{B}_3t^3$$

Program:

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

void bezier (int x[4], int y[4])
{
    int i;
    double t;

    for (t = 0.0; t < 1.0; t += 0.0005)
```

```
{
double xt = pow (1-t, 3) * x[0] + 3 * t * pow (1-t, 2) * x[1] + 3 * pow (t, 2) * (1-t)
* x[2] + pow (t, 3) * x[3];

double yt = pow (1-t, 3) * y[0] + 3 * t * pow (1-t, 2) * y[1] + 3 * pow (t, 2) * (1-t)
* y[2] + pow (t, 3) * y[3];

putpixel (xt, yt, WHITE);
}

for (i=0; i<4; i++)
putpixel (x[i], y[i], RED);
}

void main()
{
int x[4], y[4];
int i;

int gd = DETECT, gm;
initgraph (&gd, &gm, "C:\\TurboC3\\BGI");

printf ("Enter the x- and y-coordinates of the four control points.\n");

for (i=0; i<4; i++)

scanf ("%d%d", &x[i], &y[i]);

bezier (x, y);

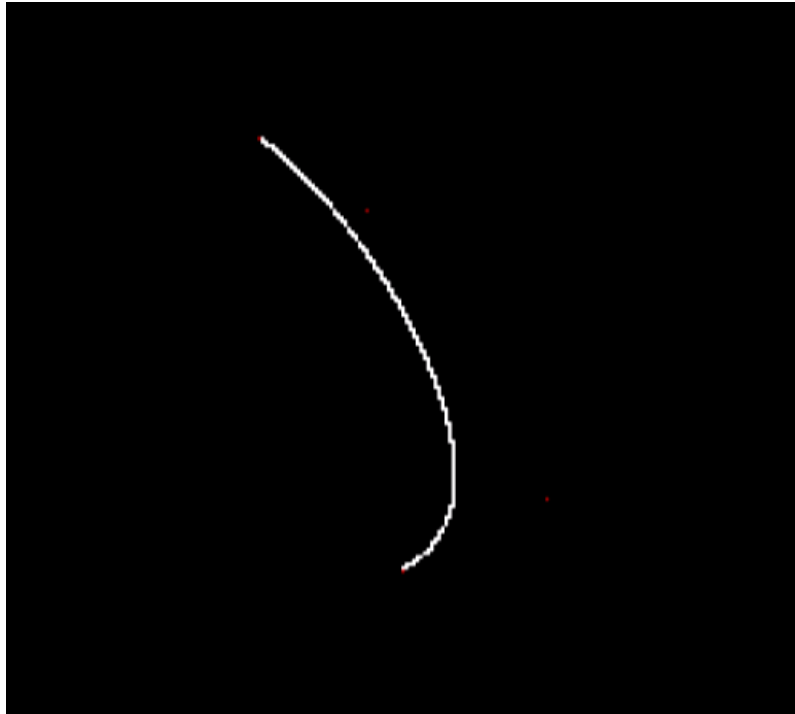
getch();

}
```

OUTPUT:

```
Enter the x- and y-coordinates of the four control points.  
100 100  
130 120  
180 200  
140 220
```

Press Enter Key, you will get following curve.



Here,

- Bezier curve lies inside the convex hull.
- Curve starts from p0 (red dot) and ends at p3 (red dot)
- Two red dots on the screen and p1 and p2. Which determines the shape of convex hull.

LAB 10: 3D Transformation

Basic 3D transformations are:

1. Translation
2. Scaling
3. Rotation

Translation

- ❖ Translation is the movement of the object from one position to another position along straight line path.
- ❖ t_x denotes translation distance along x-axis and t_y denotes translation distance along y axis.
- ❖ Translation Distance tells us, by how much units we should shift the object from one location to another along x, y-axis.

Let, $P(x,y,z)$ are old coordinates of a point. Then the new coordinates of that same point $P' (x',y',z')$ can be obtained as:

$$\begin{aligned}x' &= x + t_x \\ y' &= y + t_y \\ z' &= z + t_z\end{aligned}$$

Scaling:

- ❖ Scaling refers to changing the size of the object either by increasing or decreasing.
- ❖ Size of object will increase or decrease based on scaling factors along x and y-axis.
- ❖ Let $P (x, y,z)$ are old coordinates of object, then new coordinates of object after applying scaling transformation are obtained as:

$$\begin{aligned}x' &= x * s_x \\ y' &= y * s_y. \\ z' &= z * s_z.\end{aligned}$$

Where, s_x , s_y and s_z are scaling factors along x-axis, y-axis and z-axis.

Rotation

- ❖ A rotation repositions all points in an object along a circular path in the plane centered at the pivot point.
- ❖ We rotate an object by an angle θ .
- ❖ By default rotation is anticlockwise
- ❖ For a 3D rotation angle of rotation and axis of rotation are required.
- ❖ The axis can be either x or y or z.

Z-axis Rotation

$$\begin{aligned}x' &= x \cos \theta - y \sin \theta \\y' &= x \sin \theta + y \cos \theta \\z' &= z\end{aligned}$$

Y-axis Rotation

$$\begin{aligned}x' &= z \sin \theta + x \cos \theta \\y' &= y \\z' &= z \cos \theta - x \sin \theta\end{aligned}$$

X-axis Rotation

$$\begin{aligned}x' &= x \\y' &= y \cos \theta - z \sin \theta \\z' &= y \sin \theta + z \cos \theta\end{aligned}$$

Program for Translation:

```
#include<stdio.h>

#include<conio.h>

#include<math.h>

#include<process.h>

#include<graphics.h>

int x1,x2,y1,y2,mx,my,depth;
```

```
void draw();

void trans();

void main()

{

int gd=DETECT,gm,c;

initgraph(&gd,&gm,"C:\\TurboC3\\BGI");

printf("\n\t\t3D Translation\n\n");

printf("\nEnter 1st top value(x1,y1):");

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

printf("Enter right bottom value(x2,y2):");

scanf("%d%d",&x2,&y2);

depth=(x2-x1)/4;

mx=(x1+x2)/2;

my=(y1+y2)/2;

draw();

getch();

cleardevice();

trans();

getch();

}
```



```
void draw()

{

bar3d(x1,y1,x2,y2,depth,1);

}

void trans()

{

int a1,a2,b1,b2,dep,x,y;

printf("\n Enter the Translation Distances:");

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

a1=x1+x;

a2=x2+x;

b1=y1+y;

b2=y2+y;

dep=(a2-a1)/4;

bar3d(a1,b1,a2,b2,dep,1);

setcolor(5);

draw();

}
```

Output:

