LAB MANUAL

COMPUTER GRAPHICS

Department of Computer Science and Engineering

UNIVERSITY COLLEGE OF ENGINEERING AND TECHNOLOGY

(Under Vinoba Bhave University)

Hazaribag, Jharkhand, India

Prepared By:
NILABH KUMAR
CSE DEPT

Table of Contents		
S.No	**Name of Experiment**	**Page No**

Hardware and Software Requirements

For implementing graphics concepts in C-Language:

1. **Software Requirement:**

   Turbo C / C++ compiler that supports graphics.h package.

   special DOSBOXed installer for Turbo C++ compiler
   Download from following links:
   http://www.megaleecher.net/Download_Turbo_For_Windows

2. **Minimum hardware requirements:**

   Intel Pentium III800 MHz Processor or higher version
   Intel chipset 810 mother board or higher version
   14'' color monitor or greater than that
   Mouse
   Keyboard
   2GB HDD or greater
   256 MB RAM or greater

For doing key frame animation:

Software Requirements: Adobe Flash Professional version 6 .

Download Adobe Flash CS6 which contains Flash Professional Also and install

Hardware Requirements:

Intel® Pentium® 4, Intel Centrino®, Intel Xeon®, or Intel Core™ Duo (or compatible) processor
Microsoft® Windows® 7 (64 bit) or Windows 8 (64 bit)
4GB of RAM
2.5GB of available hard-disk space for installation; additional free space required during installation (cannot install on removable flash storage devices)
1024x768 display (1280x800 recommended)
QuickTime 10.x software recommended

Basic Structure of a C-graphics program:

```
#include<stdio.h>
#include<graphics.h>//must be included for every graphics program
#include<conio.h>
#include<dos.h> //for including delay function.
void main()
{
int gd=DETECT,gm; //gd=detects best available graphics driver, gm =graphics mode.
initgraph(&gd,&gm,"C:\\TurboC3\\BGI");// for initializing graph mode
// above 2 steps are must for every graphics program.
//declaration of any variables must be done before calling  initgraph() function.
// next write code for producing requiring design or drawing object
line(100,100,200,200);//draws a line segment.
getch();
}
```

Experiment – 1

Digital Differential Analyzer Algorithm

1 : Digital differential analyzer (DDA) is used for linear interpolation of variables over an interval between given start, end points and for rasterization of lines, triangles and polygons. Using DDA Algorithm, Write a C-Program to draw a line segment between two given points?

Aim: To implement DDA Algorithm for drawing a line segment between two given end points A (x1, y1) and B(x2, y2).

Description: DDA algorithm is an incremental scan conversion method. Here we perform calculations at each step using the results from the preceding step. The characteristic of the DDA algorithm is to take unit steps along one coordinate and compute the corresponding values along the other coordinate. The unit steps are always along the coordinate of greatest change, e.g. if dx = 10 and dy = 5, then we would take unit steps along x and compute the steps along y.

In DDA we need to consider two cases;

One is slope of the line less than or equal to one (|m| ≤1) and slope of the line greater than one (m| > 1).

1)When **|m| ≤ 1** means y2-y1 = x2-x1 or y2-y1 <x2-x1.In both these cases we assume x to be the major axis. Therefore we sample x axis at unit intervals and find the y values corresponding to each x value. We have the slope equation as

$$\Delta y = m \Delta x$$

$$y2-y1 = m (x2-x1)$$

In general terms we can say that $y_{i+1} - y_i = m(x_{i+1} - x_i)$. But here $\Delta x = 1$; therefore the equation reduces to $y_{i+1} = y_i + m = y_i + dy/dx$.

2) When **| m| > 1** means y2-y1> x2-x1 and therefore we assume y to be the major axis. Here we sample y axis at unit intervals and find the x values corresponding to each y value. We have the slope equation as

$$\Delta y = m \Delta x$$

$$y2-y1 = m (x2-x1)$$

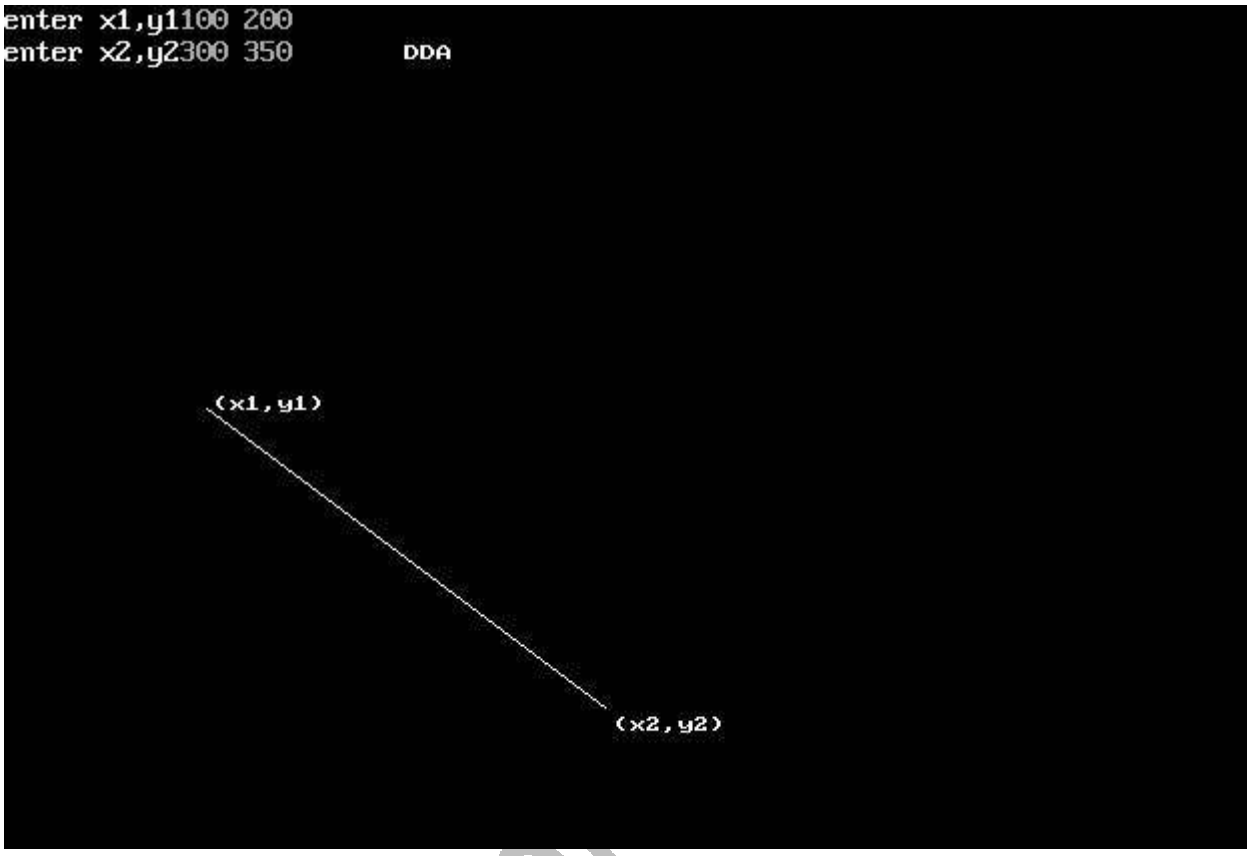
Algorithm:

1. Start.
2. Declare variables x,y,x1,y1,x2,y2,k,dx,dy,s,xi,yi and also declare
   gdriver=DETECT, mode.
3. Initialize the graphic mode with the path location in TurboC3 folder.
4. Input the two line end-points and store the left end-points in (x1,y1).
5. Load (x1, y1) into the frame buffer; that is, plot the first point. put x=x1,y=y1.
6. Calculate dx=x2-x1 and dy=y2-y1.
7. If abs (dx) > abs (dy), do s=abs(dx).
8. Otherwise s= abs(dy).
9. Then xi=dx/s and yi=dy/s.
10. Start from k=0 and continuing till k<s,the points will be
    i. x=x+xi.
    ii. Y=y+yi.
11. Plot pixels using putpixel at points (x,y) in specified colour.
12. Close Graph and stop.

Program:

```
#include<stdio.h>
#include<graphics.h>
#include<math.h>
float round(float a);
void main()
{
        int gd=DETECT,gm;
        // gd=graphics driver (detects best graphics driver and assigns it as default, gm=graphics
mode.
        int x1,y1,x2,y2,steps,k;
        float xincr,yincr,x,y,dx,dy;
        printf("enter x1,y1");
        scanf("%d%d",&x1,&y1);
        printf("enter x2,y2");
        scanf("%d%d",&x2,&y2);
        initgraph(&gd,&gm,"c:\\turboc3\\BGI");//initializes the graph
        dx=x2-x1;
        dy=y2-y1;
        if(abs(dx)>abs(dy))
                steps=abs(dx);
        else
                steps=abs(dy);
        xincr=dx/steps;
        yincr=dy/steps;
        x=x1;
        y=y1;
        for(k=1;k<=steps;k++)
        {
                delay(100);//for seeing the line drawing process slowly.
                x+=xincr;
                y+=yincr;
                putpixel(round(x),round(y),WHITE);
        }
        outtextxy(200,20,"DDA"); // for printing text at desired screen location.
        outtextxy(x1+5,y1-5,"(x1,y1)");
        outtextxy(x2+5,y2+5,"(x2,y2)");
        getch();
        closegraph();  // closes the graph and comes back to previous graphic mode.
}
float round(float a)
{
        int b=a+0.5;
        return b;
}
```

Output:



Experiment 2

Bresenham's Line Drawing Algorithm

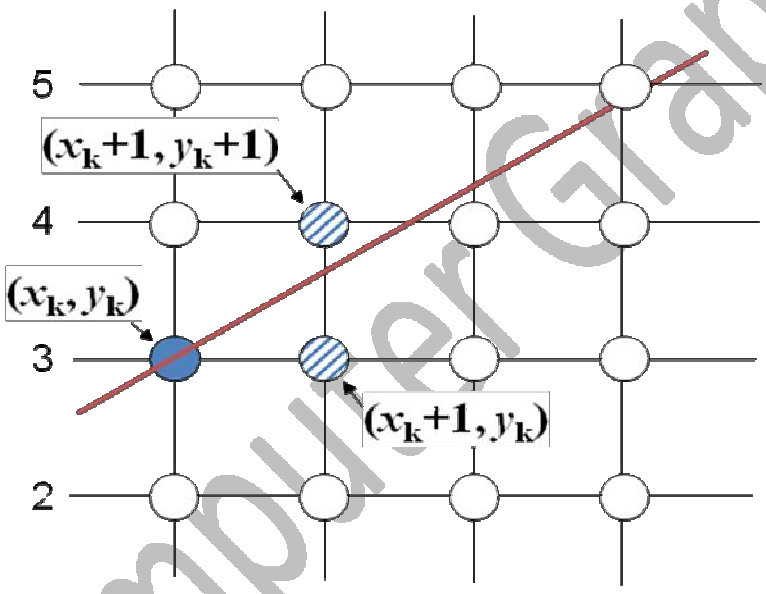
2: Bresenham's line algorithm is an algorithm which determines which order to form a close approximation to a straight line between two given points. Write a C program for determining pixel activation list between two given points in order to draw line segment using bresenham's Line drawing algorithm?

Aim: To implement Bresenham's line drawing algorithm for drawing a line segment between two given endpoints A (x1, y2) and B(x2, y2).

Description:

Basic Concept:

Move across the *x* axis in unit intervals and at each step choose between two different *y* coordinates



For example, from position (2, 3) we have to choose between (3, 3) and (3, 4). We would like the point that is closer to the original line

So we have to take decision to choose next point. So next pixels are selected based on the value of decision parameter p. The equations are given in below algorithm.

Algorithm:

BRESENHAM'S LINE DRAWING ALGORITHM

1. Input the two line end-points, storing the left end-point in (x_0, y_0)
2. Plot the point (x_0, y_0)
3. Calculate the constants Δx, Δy, $2\Delta y$, and ($2\Delta y - 2\Delta x$) and get the first value for the decision parameter as:

$$p_0 = 2\Delta y - \Delta x$$

4. At each x_k along the line, starting at $k = 0$, perform the following test. If $p_k < 0$, the next point to plot is (x_k+1, y_k) and:

$$p_{k+1} = p_k + 2\Delta y$$

Otherwise, the next point to plot is (x_k+1, y_k+1) and:

$$p_{k+1} = p_k + 2\Delta y - 2\Delta x$$

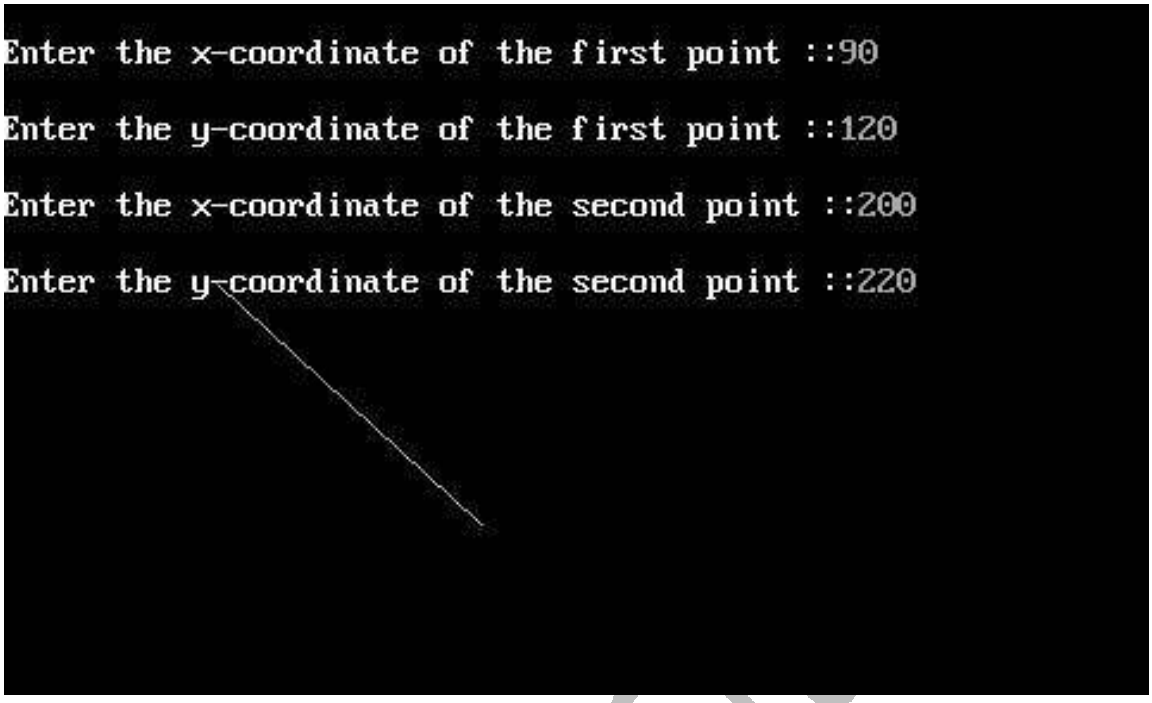
5. Repeat step 4 ($\Delta x - 1$) times

NOTE: The algorithm and derivation above assumes slopes are less than 1. For other slopes we need to adjust the algorithm slightly

Program:

```
#include<stdio.h>
#include<conio.h>
#include<graphics.h>
void main()
{
        int x,y,x1,y1,x2,y2,p,dx,dy; int gd=DETECT,gm;
        initgraph(&gd,&gm,"C:\\TurboC3\\BGI");
        printf("\nEnter the x-coordinate of the first point ::");
        scanf("%d",&x1);
        printf("\nEnter the y-coordinate of the first point ::");
        scanf("%d",&y1);
        printf("\nEnter the x-coordinate of the second point ::");
        scanf("%d",&x2);
        printf("\nEnter the y-coordinate of the second point ::");
        scanf("%d",&y2);
        x=x1;
        y=y1;
        dx=x2-x1;
        dy=y2-y1;
        putpixel(x,y,2);
        p=(2*dy-dx);
        while(x<=x2)
        {
          if(p<0)
          {
                    x=x+1;
                    p=p+2*dy;
           }
          else
          {
                   x=x+1;
                   y=y+1;
                   p=p+(2*dy)-(2*dx);
            }
           putpixel(x,y,7);
        }
        getch();
        closegraph();
}
```

Output:



Experiment-3

Midpoint Circle Generation Algorithm

3: Using Midpoint circle generation algorithm which is a variant of Bresenham's line algorithm, write a C-Program to generate pixel activation list for drawing a circle with a given centre of circle P(x, y) and a radius r?

Aim: To implement midpoint circle generation algorithm or bresenham's circle algorithm for drawing a circle of given center (x, y) and radius r.

Description:

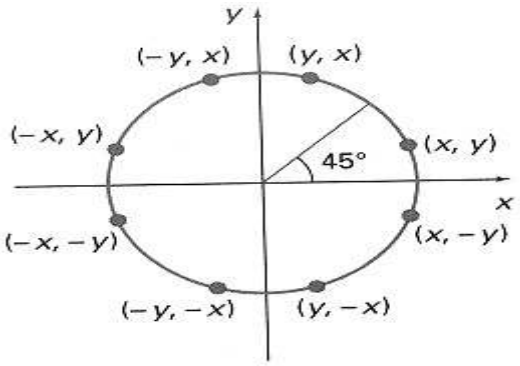
Circles have the property of being highly symmetrical, which is handy when it comes to drawing them on a display screen.

We know that there are 360 degrees in a circle. First we see that a circle is symmetrical about the x axis, so only the first 180 degrees need to be calculated.

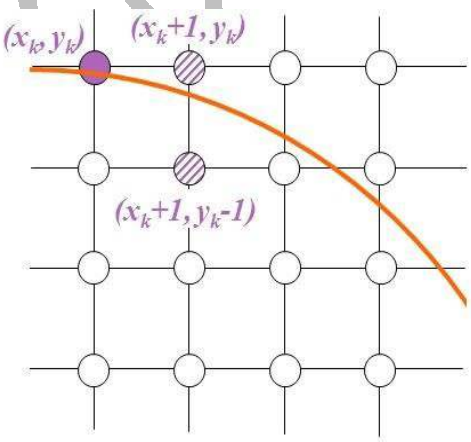
Next we see that it's also symmetrical about the y axis, so now we only need to calculate the first 90 degrees.

Finally we see that the circle is also symmetrical about the 45 degree diagonal axis, so we only need to calculate the first 45 degrees.

We only need to calculate the values on the border of the circle in the first octant. The other values may be determined by symmetry.



Bresenham's circle algorithm calculates the locations of the pixels in the first 45 degrees. It assumes that the circle is centered on the origin. So for every pixel (x, y) it calculates, we draw a pixel in each of the eight octants of the circle. This is done till when the value of the y coordinate equals the x coordinate. The pixel positions for determining symmetry are given in the below algorithm.



Assume that we have just plotted point (x_k, y_k)

The next point is a choice between (x_k+1, y_k) and (x_k+1, y_k-1)

We would like to choose the point that is nearest to the actual circle

So we use decision parameter here to decide.

Algorithm:

1. Input radius *r* and circle centre (x_c, y_c), then set the coordinates for the first point on the circumference of a circle centred on the origin as:

$$(x_0, y_0) = (0, r)$$

2. Calculate the initial value of the decision parameter as:

$$p_0 = \frac{5}{4} - r$$

3. Starting with $k = 0$ at each position x_k, perform the following test. If $p_k < 0$, the next point along the circle centred on $(0, 0)$ is (x_k+1, y_k) and:

$$p_{k+1} = p_k + 2x_{k+1} + 1$$

Otherwise the next point along the circle is (x_k+1, y_k-1) and:

$$p_{k+1} = p_k + 2x_{k+1} + 1 - 2y_{k+1}$$

4. Determine symmetry points in the other seven octants
5. Move each calculated pixel position (x, y) onto the circular path centred at (x_c, y_c) to plot the coordinate values:
6. Repeat steps 3 to 5 until $x >= y$

$$x = x + x_c \qquad y = y + y_c$$

Symmetric pixel positions:

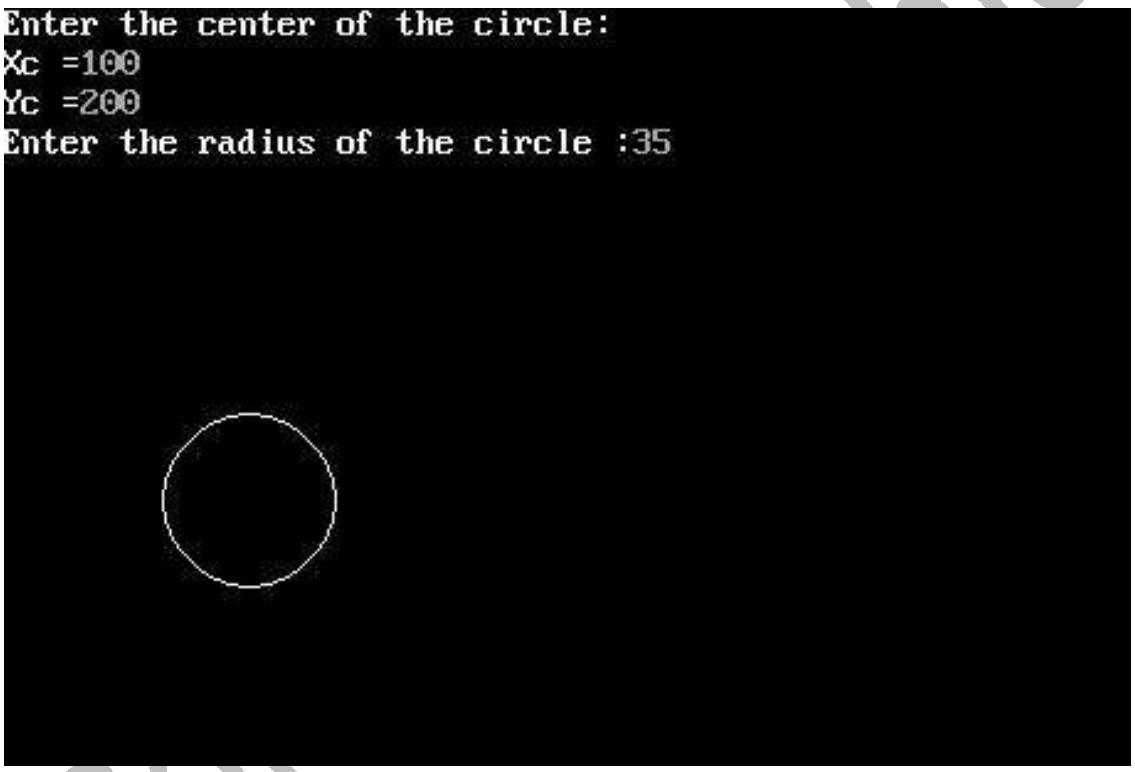
```
putpixel(xc+x,yc-y,GREEN); //For pixel (x,y)
putpixel(xc+y,yc-x, GREEN); //For pixel (y,x)
putpixel(xc+y,yc+x, GREEN); //For pixel (y,-x)
putpixel(xc+x,yc+y, GREEN); //For pixel (x,-y)
putpixel(xc-x,yc+y, GREEN); //For pixel (-x,-y)
putpixel(xc-y,yc+x, GREEN); //For pixel (-y,-x)
putpixel(xc-y,yc-x, GREEN); //For pixel (-y,x)
putpixel(xc-x,yc-y, GREEN); //For pixel (-x,y)
```

Program:

```
#include<dos.h>
#include<stdio.h>
#include<conio.h>
#include<graphics.h>
void draw_circle(int,int,int);
void symmetry(int,int,int,int);
void main()
{
          int xc,yc,R;
          int gd=DETECT,gm;
          initgraph(&gd,&gm,"C:\\TurboC3\\BGI");
          printf("Enter the center of the circle:\n");
          printf("Xc =");
          scanf("%d",&xc);
          printf("Yc =");
          scanf("%d",&yc);
          printf("Enter the radius of the circle :");
          scanf("%d",&R);
          draw_circle(xc,yc,R);
          getch();
          closegraph();
}

void draw_circle(int xc,int yc,int rad)
{
        int x = 0; int y
        = rad; int p =
        1-rad;
        symmetry(x,y,xc,yc);
        for(x=0;y>x;x++)
          {
                    if(p<0)
                   p += 2*x + 3;
                   else
                   {
                     p += 2*(x-y) + 5;
                     y--;
                   }
                  symmetry(x,y,xc,yc);
                 delay(50);
           }
}

void symmetry(int x,int y,int xc,int yc)
{
          putpixel(xc+x,yc-y,GREEN);  //For pixel (x,y)
          delay(50);
          putpixel(xc+y,yc-x, GREEN);  //For pixel (y,x)
          delay(50);
          putpixel(xc+y,yc+x, GREEN);  //For pixel (y,-x)
```

```
        delay(50);
        putpixel(xc+x,yc+y, GREEN);  //For pixel (x,-y)
        delay(50);
        putpixel(xc-x,yc+y, GREEN);  //For pixel (-x,-y)
        delay(50);
        putpixel(xc-y,yc+x, GREEN);  //For pixel (-y,-x)
        delay(50);
        putpixel(xc-y,yc-x, GREEN);  //For pixel (-y,x)
        delay(50);
        putpixel(xc-x,yc-y, GREEN);  //For pixel (-x,y)
        delay(50);
}
```

Output:



Experiment 4

Ellipse Generation Algorithm

4: Using Midpoint ellipse generation algorithm which is a variant of Bresenham's line algorithm, write a C-Program to generate pixel activation list for drawing a ellipse?

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

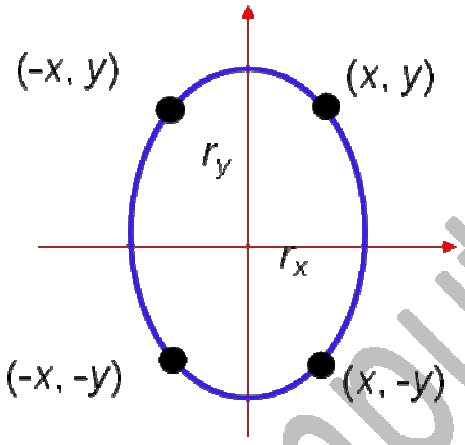
Description:

Basic Concept: 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. The required conditions are given in following algorithm.

Algorithm:

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

   $$p1_0 = r_y^2 - r_x^2 r_y + \frac{1}{4} r_x^2$$

3. 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

   $$p1_{i+1} = p1_i + 2r_y^2 x_{i+1} + r_y^2$$

   Otherwise, the next point is $(x_i + 1, y_i - 1)$ and

   $$p1_{i+1} = p1_i + 2r_y^2 x_{i+1} - 2r_x^2 y_{i+1} + r_y^2$$

   and continue until $2r_y^2 x \geq 2r_x^2 y$

4. (x_0, y_0) is the last position calculated in region 1. Calculate the initial parameter in region 2 as

   $$p2_0 = r_y^2 (x_0 - \frac{1}{2})^2 + r_x^2 (y_0 - 1)^2 - r_x^2 r_y^2$$

5. 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

   $$p2_{i+1} = p2_i - 2r_x^2 y_{i+1} + r_x^2$$

   Otherwise, the next point is $(x_i + 1, y_i - 1)$ and $p2_{i+1} = p2_i + 2r_y^2 x_{i+1} - 2r_x^2 y_{i+1} + r_x^2$

   Use the same incremental calculations as in region 1. Continue until $y = 0$.

6. For both regions determine symmetry points in the other three quadrants.

7. Move each calculated pixel position (x, y) onto the elliptical path centered on (x_c, y_c) and plot the coordinate values

   $x = x + x_c$, $y = y + y_c$

Program:

```
#include<stdio.h>
#include<conio.h>
#include<graphics.h>
#include<math.h>
void disp();
float x,y; int
xc,yc; void
main()
{
        int gd=DETECT,gm;
        int rx,ry;
        float p1,p2;
        clrscr();
        initgraph(&gd,&gm,"C:\\TurboC3\\BGI");
        printf("Enter the center point :");
        scanf("%d%d",&xc,&yc);
        printf("Enter the value for Rx and Ry :");
        scanf("%d%d",&rx,&ry);
        x=0;
        y=ry;
        disp();
        p1=(ry*ry)-(rx*rx*ry)+(rx*rx)/4;
        while((2.0*ry*ry*x)<=(2.0*rx*rx*y))
        {
                x++;
                if(p1<=0)
                        p1=p1+(2.0*ry*ry*x)+(ry*ry);
                else
                {
                        y--;
                        p1=p1+(2.0*ry*ry*x)-(2.0*rx*rx*y)+(ry*ry);
                }
                disp();
                x=-x;
                disp();
                x=-x;
        } x=rx;
        y=0;
        disp();
        p2=(rx*rx)+2.0*(ry*ry*rx)+(ry*ry)/4;
        while((2.0*ry*ry*x)>(2.0*rx*rx*y))
        {
                y++;
                if(p2>0)
                        p2=p2+(rx*rx)-(2.0*rx*rx*y);
                else
                {
                        x--;
```

```
                        p2=p2+(2.0*ry*ry*x)-(2.0*rx*rx*y)+(rx*rx);
                }
                disp();
                y=-y;
                disp();
                y=-y;
        }
        getch();
        closegraph();
}
void disp()
{
      delay(50);
      putpixel(xc+x,yc+y,10);
      putpixel(xc-x,yc+y,10);
      putpixel(xc+x,yc-y,10);
      putpixel(xc-x,yc-y,10);
}
```

Output:

Experiment- 5

Creating various types of texts and fonts

5: Using different graphics functions available for text formatting in C-Language, Write a C program for displaying text in different sizes, different colors, different font styles?

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

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()**--------setcolor function is 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:

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)

Character size(0-9)

Program:

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

Experiment- 6

Creating two dimensional objects

6: Using certain graphic functions available in C-language for drawing lines, rectangles & circles, Write a C-Program which generates pixel activation list for drawing the following simple two dimensional objects (Circle, Ellipse…..).

i) House ii) Car iii) Fish iv) man?

Aim: To write a C-program for creating simple two dimensional shape of house, car, fish, man using lines, circles etc.

Description:

The following graphics functions are available for creating two dimensional shapes in C.

line circle
ellipse
rectangle
drawpoly

line: line function is used to draw a line from a point(x1,y1) to point(x2,y2) i.e. (x1,y1) and (x2,y2) are end points of the line.
Declaration: - line(x1, y1, x2, y2); ---line (100,200,300,400);

Circle: Circle function is used to draw a circle with center (x, y) and third parameter specifies the radius of the circle.
Declaration: circle(x, y, r)—circle (100, 200, 25) ;(25 is radius of circle, (100,100) is center of circle).

Ellipse: Ellipse is used to draw an ellipse (x, y) are coordinates of center of the ellipse, startangle is the starting angle, endangle is the ending angle, and fifth and sixth parameters specifies the X and Y radius of the ellipse. To draw a complete ellipse strangles and end angle should be 0 and 360 respectively.
Usage: ellipse(x, y, startangle, endangle, xradius, yradius);--ellipse(100,200,0,360,25,45);((100,200) is center of ellipse, 0 is start angle, 360 is end angle, 25 is x-axis radius, 45 is radius circle).

Rectangle: rectangle function is used to draw a rectangle. Coordinates of left top and right bottom corner are required to draw the rectangle. left specifies the X-coordinate of top left corner, top specifies the Y-coordinate of top left corner, right specifies the X-coordinate of right bottom corner, bottom specifies the Y-coordinate of right bottom corner.

Syntax: rectangle(left,top,right,bottom);--rectangle(100,200,300,400);

Drawpoly: Drawpoly function is used to draw polygons i.e. triangle, rectangle, pentagon, hexagon etc.

Syntax: drawpoly(num,points);--num indicates number of vertices of polygon. Num = num+1.

Example: we will draw a triangle using drawpoly, consider for example the array :-
int points[] = { 320, 150, 420, 300, 250, 300, 320, 150};

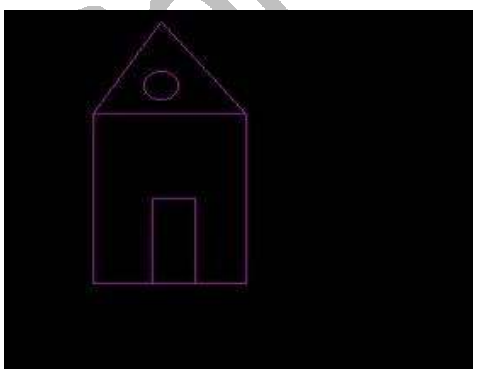
points array contains coordinates of triangle which are (320, 150), (420, 300) and (250, 300). Note that last point(320, 150) in array is same as first.

Number of vertices are denoted by num. for any polygon, number of vertices are (num+1). For triangle, number of vertices are 4.

i) Program for creating House shape:

```
#include<stdio.h>
#include<conio.h>
#include<graphics.h>
void main()
{
        int gd=DETECT,gm;
        initgraph(&gd,&gm,"C:\\TURBOC3\\BGI");
        setcolor(5);
        rectangle(60,80,150,200);
        rectangle(95,140,120,200);
        line(60,80,100,15);
        line(100,15,150,80);
        circle(100,60,10);
        getch();
        closegraph();
}
```

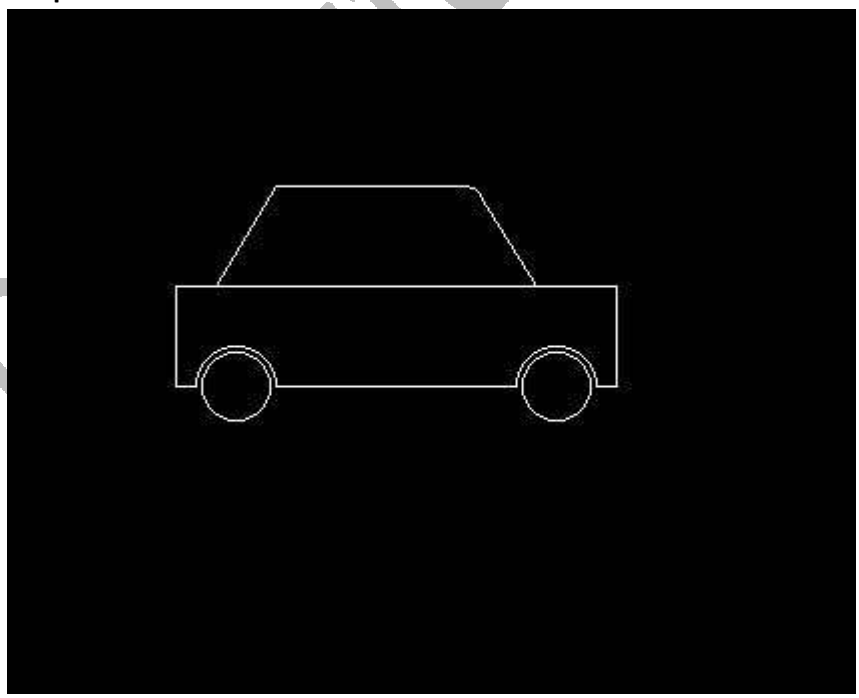
Output:



ii) Program for creating simple car shape:

```
#include<stdio.h>
#include<conio.h>
#include<graphics.h>
#include<dos.h>
void main()
{
   int gd = DETECT, gm;
   initgraph(&gd, &gm, "C:\\TurboC3\\BGI");
   cleardevice();
   line( 150, 100,  242, 100);
   ellipse(242, 105, 0, 90, 10, 5);
   line(150, 100,  120, 150);
   line(252, 105,  280, 150);
   line(100, 150,  320, 150);
   line(100, 150,  100, 200);
   line(320, 150,  320, 200);
   line(100, 200,  110, 200);
   line( 320, 200,   310, 200);
   arc(130, 200, 0, 180, 20);
   arc( 290, 200, 0, 180, 20);
   line( 270, 200,  150, 200);
   circle(130, 200, 17);
   circle(290, 200, 17);
   getch();
}
```

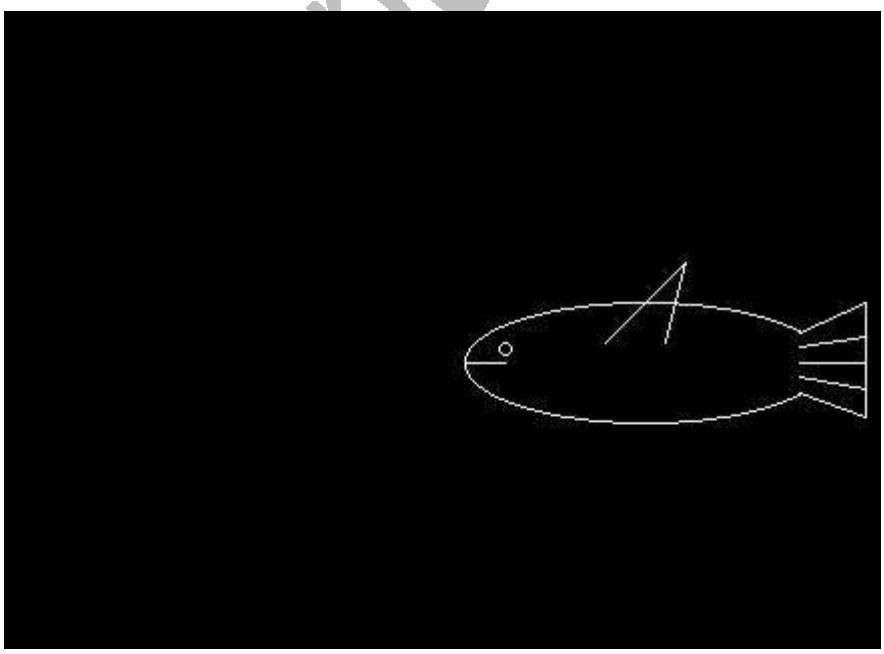
Output:



iii)Program for creating fish:

```
#include<stdlib.h>
#include<conio.h>
#include<dos.h>
#include<graphics.h>
#include<ctype.h>
void main()
{
        int gd=DETECT,gm;
        initgraph(&gd,&gm,"C:\\TurboC3\\BGI");
        cleardevice();
        ellipse(520,200,30,330,90,30);
        circle(450,193,3);
        line(430,200,450,200);
        line(597,185,630,170);
        line(597,215,630,227);
        line(630,170,630,227);
        line(597,200,630,200);
        line(597,192,630,187);
        line(597,207,630,213);
        line(500,190,540,150);
        line(530,190,540,150);
    getch();
}
```

Output:



iv)Program for creating man object:

```
#include<stdio.h>
#include<graphics.h>
#include<conio.h>
void main()
{
                    int gd=DETECT,gm;
                    initgraph(&gd,&gm,"C:\\TurboC3\\BGI");
                    setcolor(9);
                    circle(150,150,35);
                    line(150,185,150,300);
                    line(150,200,120,230);
                    line(150,200,180,230);
                    line(150,300,120,330);
                    line(150,300,180,330);
                    outtextxy(230,350,"HI, This is Computer Graphics");
                    getch();
}
```

Output:

Sample program illustrating the use drawpoly() function:

Aim: To draw a triangle using drawpoly() function.

Description:

Using drawpoly (), we can draw any polygon of any number of vertices.

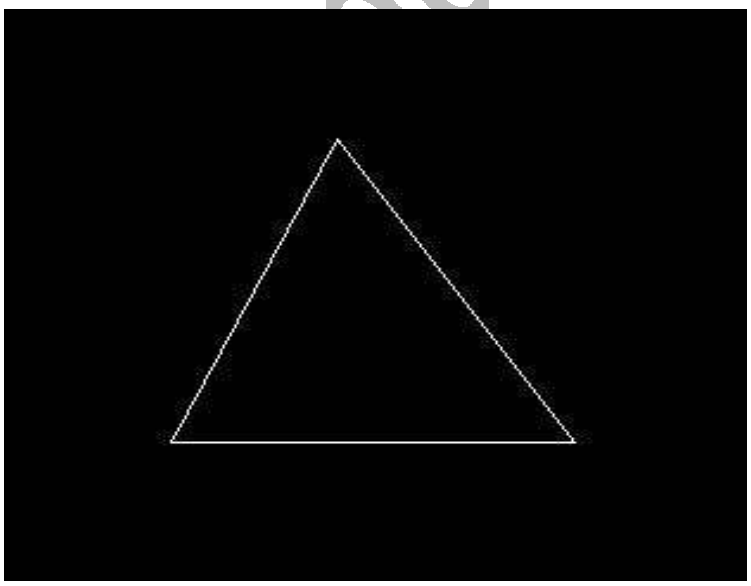
Syntax: drawpoly (num, points);--num indicates number of vertices of polygon. Num = num+1.
Example: we will draw a triangle using drawpoly, consider for example the array:-
int points[] = { 320, 150, 420, 300, 250, 300, 320, 150};

1) Points array contains coordinates of triangle which are (320, 150), (420, 300) and (250, 300). Note that last point (320, 150) in array is same as first.
2) Number of vertices is denoted by num. for any polygon, numbers of vertices are (num+1). For triangle, number of vertices are 4.

Program:

```
#include <graphics.h>
#include <conio.h>
main()
{
        int gd=DETECT,gm,points[]={320,150,420,300,250,300,320,150};
         initgraph(&gd, &gm, "C:\\TurboC3\\BGI");
         drawpoly(4, points);
         getch();
        closegraph();
        return 0;
}
```

Output:



Experiment 7

Two Dimensional Transformations

7: Write a C-program for performing the basic 2D transformations such as translation, Scaling, Rotation, shearing and reflection for a given 2D object?

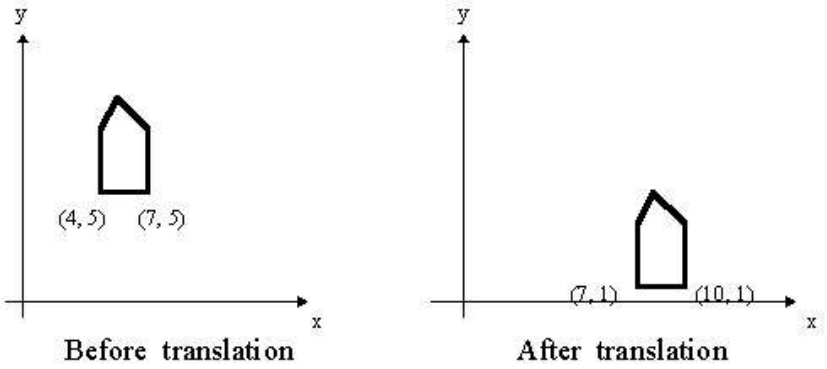
Aim: To apply the basic 2D transformations such as translation, Scaling, Rotation, shearing and reflection for a given 2D object.

Description: We have to perform 2D transformations on 2D objects. Here we perform transformations on a line segment.

The 2D transformations are:

1. Translation
2. Scaling
3. Rotation
4. Reflection
5. Shear

1. Translation: Translation is defined as moving the object from one position to another position along straight line path.



We can move the objects based on translation distances along x and y axis. tx denotes translation distance along x-axis and ty denotes translation distance along y axis.

Translation Distance: It is nothing but by how much units we should shift the object from one location to another along x, y-axis.

Consider (x,y) are old coordinates of a point. Then the new coordinates of that same point (x',y') can be obtained as follows:

X'=x+tx

Y'=y+ty

We denote translation transformation as P. we express above equations in matrix form as:

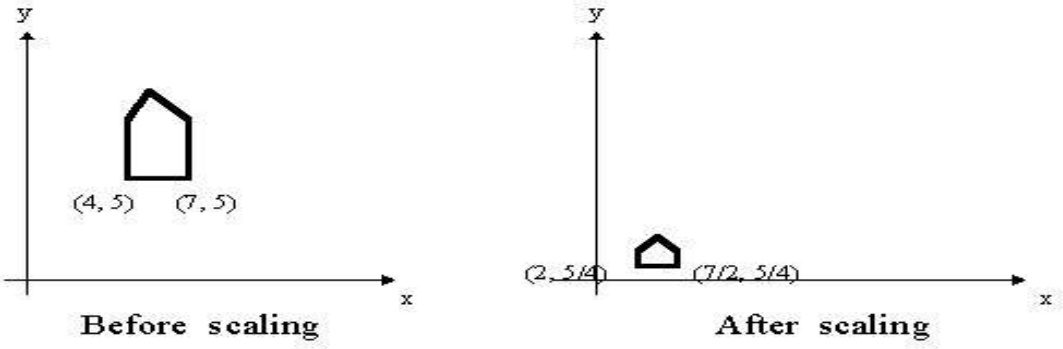
$$P' = P + T$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

x,y---old coordinates x',y'—new coordinates after translation

tx,ty—translation distances, T is

2. Scaling: scaling refers to changing the size of the object either by increasing or decreasing. We will increase or decrease the size of the object based on scaling factors along x and y-axis.



If (x, y) are old coordinates of object, then new coordinates of object after applying scaling transformation are obtained as:

x'=x*sx

y'=y*sy.

sx and sy are scaling factors along x-axis and y-axis. we express the above equations in matrix form as:

$$\begin{matrix} x & s_x & 0 & x \\ y & 0 & s_y & y \end{matrix}$$

Scaling Matrix

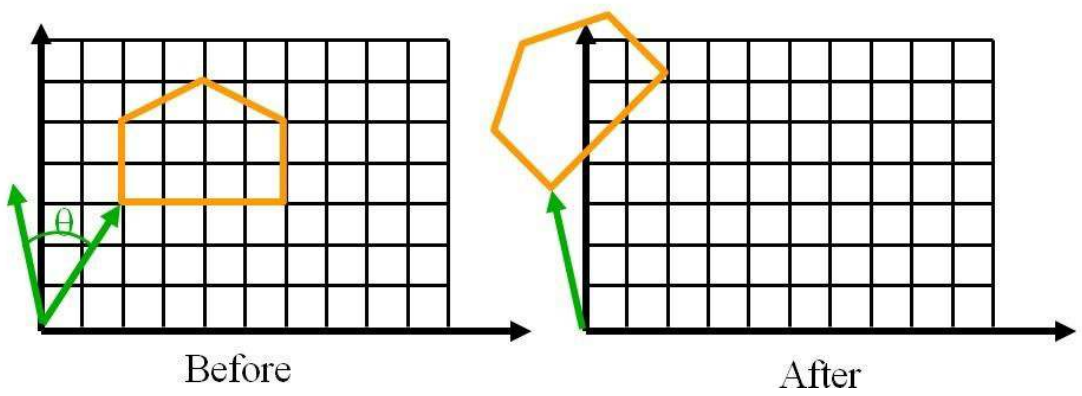
3. **Rotation:** A rotation repositions all points in an object along a circular path in the plane centered at the pivot point. We rotate an object by an angle theta.

New coordinates after rotation depend on *both x* and *y*

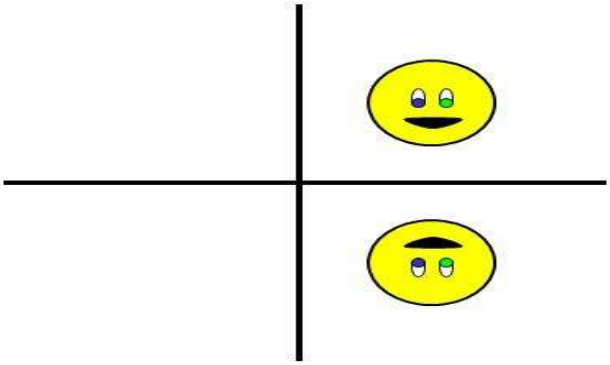
- $x' = x\cos\theta - y\sin\theta$
- $y' = x\sin\theta + y\cos\theta$
- **or in matrix form:**

P' = R • P, $R \begin{matrix} \cos & \sin \\ \sin & \cos \end{matrix}$

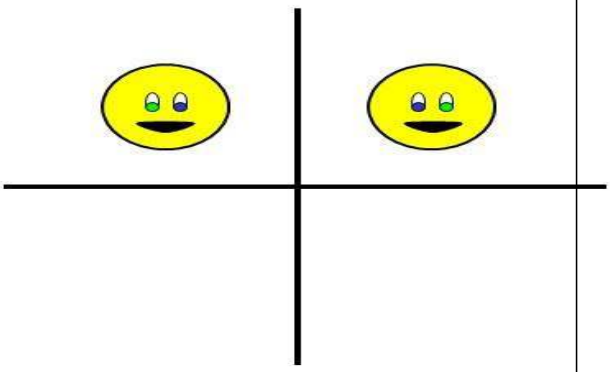
R-rotation matrix.



4. **Reflection:** Reflection is nothing but producing mirror image of an object. Reflection can be done just by rotating the object about given axis of reflection with an angle of 180 degrees.



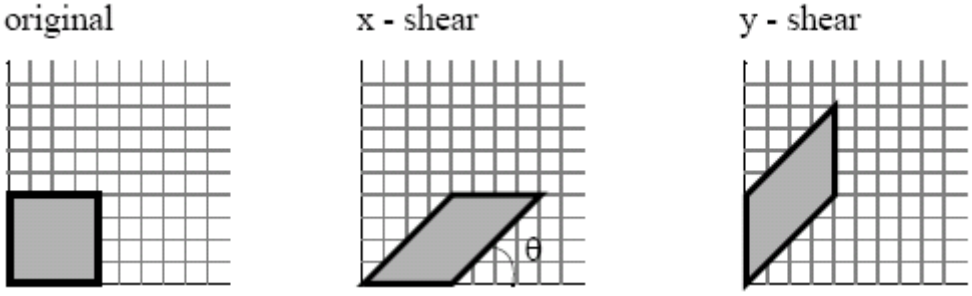
X-axis reflection



Y-axis reflection

5. Shear:

1. Shear is the translation along an axis by an amount that increases linearly with another axis (**Y**). It produces shape distortions as if objects were composed of layers that are caused to slide over each other.
2. Shear transformations are very useful in creating italic letters and slanted letters from regular letters.
3. Shear transformation changes the shape of the object to a slant position.



4. Shear transformation is of 2 types:

a. X-shear: changing x-coordinate value and keeping y constant

x'=x+shx*y

y'=y

b. Y-shear: changing y coordinates value and keeping x constant

x'=x

y'=y+shy*x

shx and shy are shear factors along x and y-axis.

1. **Program for translation:**

```
#include<stdio.h>
#include<conio.h>
#include<graphics.h>
#include<math.h>
void main()
{
        int gd=DETECT,gm;
        int x1,y1,x2,y2,tx,ty,x3,y3,x4,y4;
        initgraph(&gd,&gm,"C:\\TurboC3\\BGI");
        printf("Enter the starting point of line segment:");
        scanf("%d %d",&x1,&y1);
        printf("Enter the ending point of line segment:");
        scanf("%d %d",&x2,&y2);
        printf("Enter translation distances tx,ty:\n");
        scanf("%d%d",&tx,&ty);
        setcolor(5);
        line(x1,y1,x2,y2);
        outtextxy(x2+2,y2+2,"Original line");
        x3=x1+tx;
        y3=y1+ty;
        x4=x2+tx;
        y4=y2+ty;
        setcolor(7);
        line(x3,y3,x4,y4);
        outtextxy(x4+2,y4+2,"Line after translation");
        getch();
}
```

Output:

2. Program for scaling:

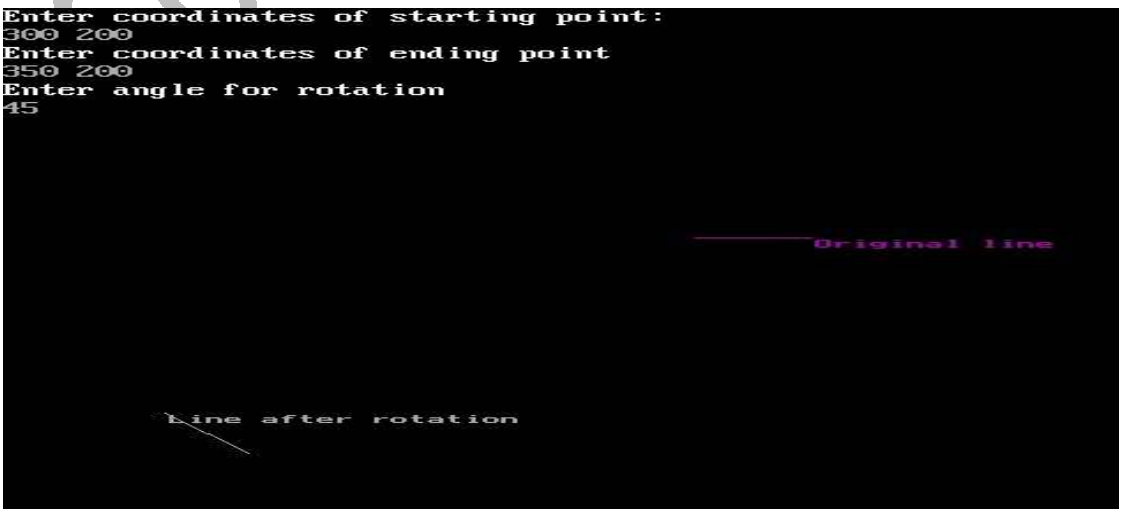
```
#include<stdio.h>
#include<conio.h>
#include<graphics.h>
#include<math.h>
void main()
{
        int gd=DETECT,gm;
        float x1,y1,x2,y2,sx,sy,x3,y3,x4,y4;
        initgraph(&gd,&gm,"C:\\TurboC3\\BGI");

        printf("Enter the starting point coordinates:");
        scanf("%f %f",&x1,&y1);
        printf("Enter the ending point coordinates:");
        scanf("%f %f",&x2,&y2);
        printf("Enter scaling factors sx,sy:\n");
        scanf("%f%f",&sx,&sy);
        setcolor(5);
        line(x1,y1,x2,y2);
        outtextxy(x2+2,y2+2,"Original line");
        x3=x1*sx;
        y3=y1*sy;
        x4=x2*sx;
        y4=y2*sy;
        setcolor(7);
        line(x3,y3,x4,y4);
        outtextxy(x3+2,y3+2,"Line after scaling");
        getch();
}
```

Output:

3. Program for Rotation:

```
#include<stdio.h>
#include<conio.h>
#include<graphics.h>
#include<math.h>
void main()
{
        int gd=DETECT,gm;
        float x1,y1,x2,y2,x3,y3,x4,y4,a,t;
        initgraph(&gd,&gm,"C:\\TurboC3\\BGI");
        printf("Enter coordinates of starting point:\n");
        scanf("%f%f",&x1,&y1);
        printf("Enter coordinates of ending point\n");
        scanf("%f%f",&x2,&y2);
        printf("Enter angle for rotation\n");
        scanf("%f",&a);
        setcolor(5); line(x1,y1,x2,y2);
        outtextxy(x2+2,y2+2,"Original line");
        t=a*(3.14/180);
        x3=(x1*cos(t))-(y1*sin(t));
        y3=(x1*sin(t))+(y1*cos(t));
        x4=(x2*cos(t))-(y2*sin(t));
        y4=(x2*sin(t))+(y2*cos(t));
        setcolor(7);
        line(x3,y3,x4,y4);
        outtextxy(x3+2,y3+2,"Line after rotation");
        getch();

}
```

Output:

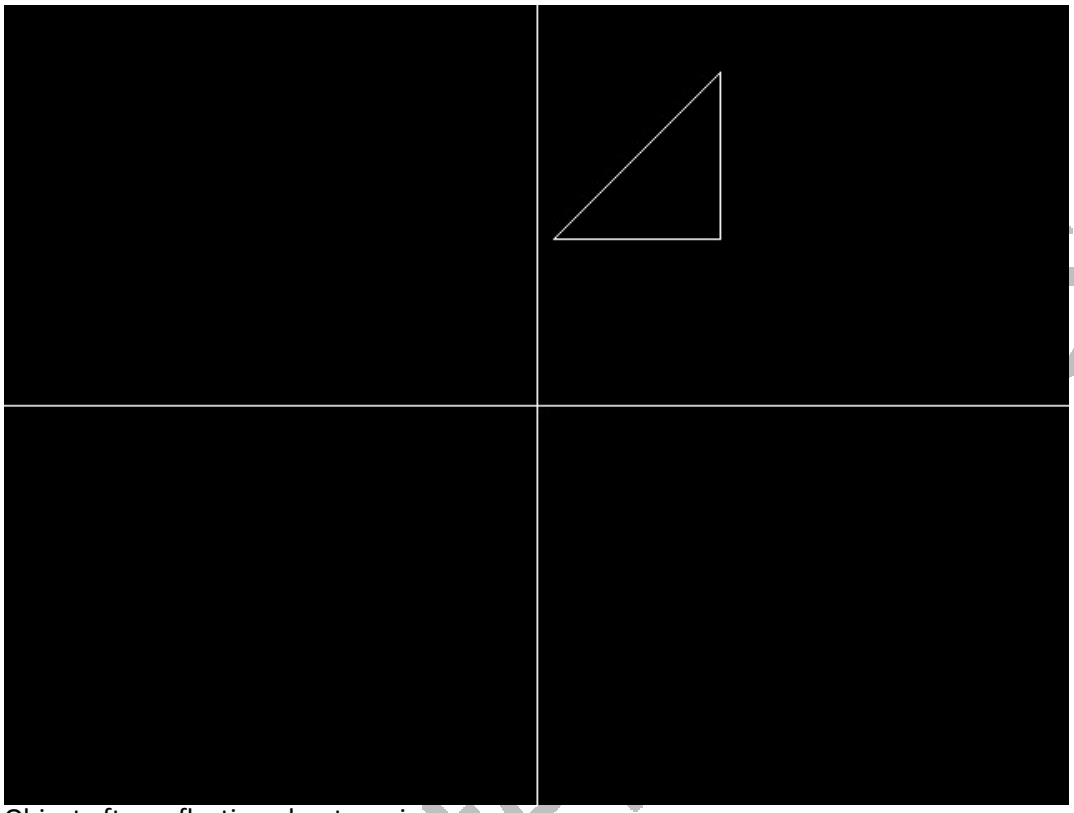


4. Program for reflection along x-axis:

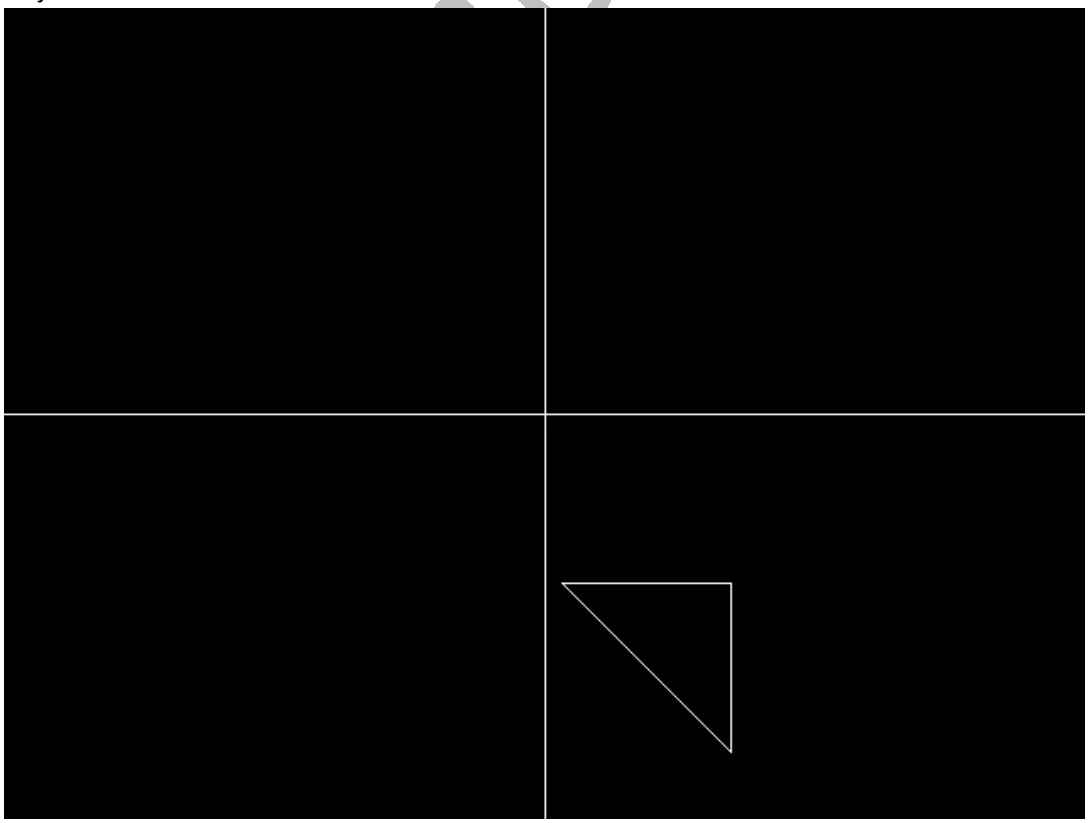
```
# include <stdio.h>
# include <conio.h>
# include <graphics.h>
# include <math.h>

char IncFlag;
int PolygonPoints[3][2] ={{10,100},{110,100},{110,200}};
void PolyLine()
{
        int iCnt;
        cleardevice();
        line(0,240,640,240);
        line(320,0,320,480);
        for (iCnt=0; iCnt<3; iCnt++)
        {
                line(PolygonPoints[iCnt][0],PolygonPoints[iCnt][1],
                  PolygonPoints[(iCnt+1)%3][0],PolygonPoints[(iCnt+1)%3][1]);
        }
}
void Reflect()
{
        float Angle;
        int iCnt; int
        Tx,Ty;
        printf("endl");;
        for (iCnt=0; iCnt<3; iCnt++)
                PolygonPoints[iCnt][1] = (480 - PolygonPoints[iCnt][1]);
}
void main()
{
        int gDriver = DETECT, gMode;
        int iCnt;
        initgraph(&gDriver, &gMode, "C:\\TurboC3\\BGI");
        for (iCnt=0; iCnt<3; iCnt++)
        {
                PolygonPoints[iCnt][0] += 320;
                PolygonPoints[iCnt][1] = 240 - PolygonPoints[iCnt][1];
        }
        PolyLine();
        getch();
        Reflect();
        PolyLine();
        getch();
```

}

Output:
Object before reflection about x-axis



Object after reflection about x-axis:

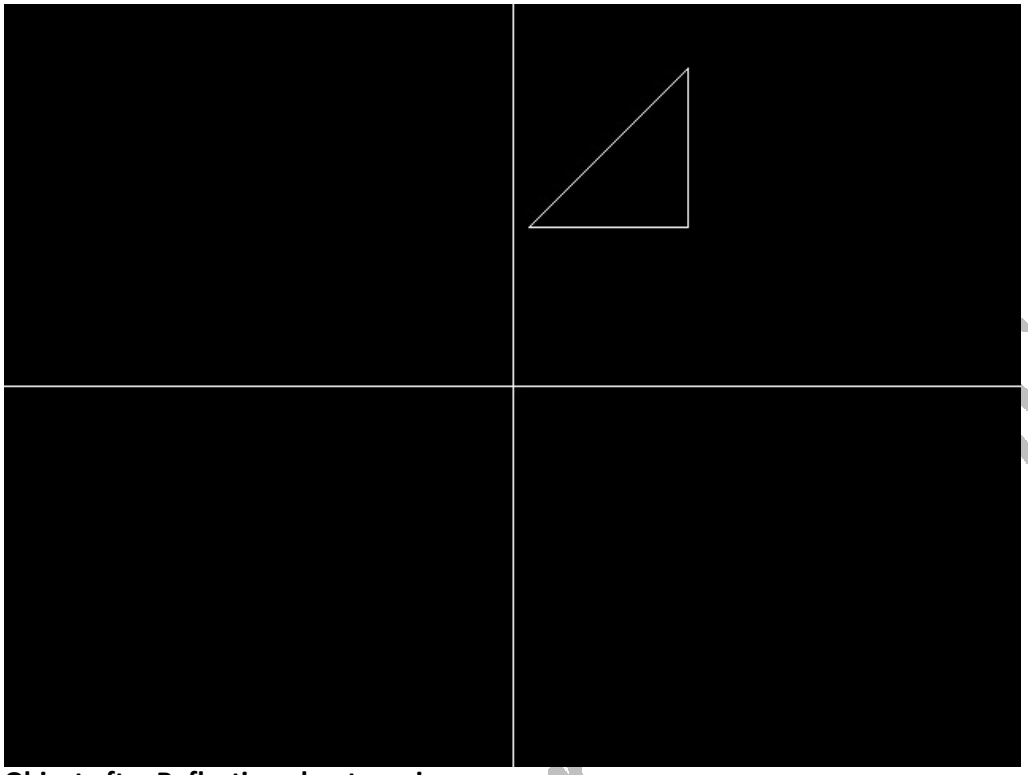


5. Program for Reflection about y-axis:

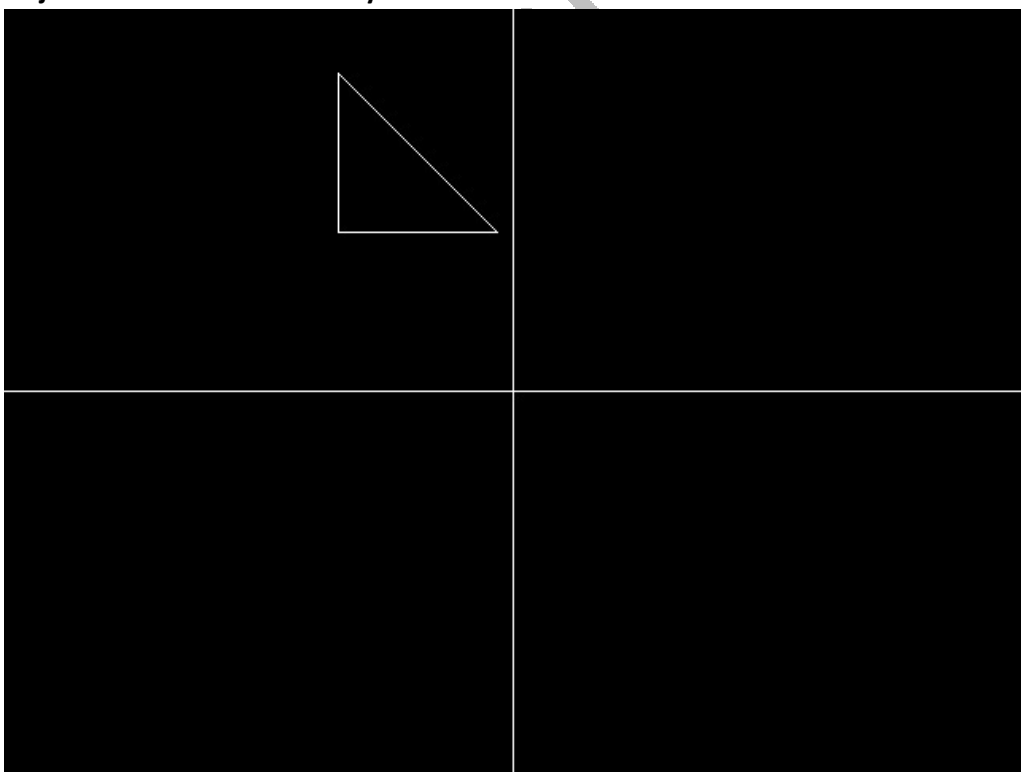
```
# include <stdio.h>
# include <conio.h>
# include <graphics.h>
# include <math.h>

char IncFlag;
int PolygonPoints[3][2] =
        {{10,100},{110,100},{110,200}};

void PolyLine()
{
        int iCnt;
        cleardevice();
        line(0,240,640,240);
        line(320,0,320,480);
        for (iCnt=0; iCnt<3; iCnt++)
        {
                line(PolygonPoints[iCnt][0],PolygonPoints[iCnt][1],
                  PolygonPoints[(iCnt+1)%3][0],PolygonPoints[(iCnt+1)%3][1]);
        }
}
void Reflect()
{
        float Angle;
        int iCnt;
        int Tx,Ty;

        for (iCnt=0; iCnt<3; iCnt++)
                PolygonPoints[iCnt][0] = (640 - PolygonPoints[iCnt][0]);
}

void main()
{
        int gd = DETECT, gm;
        int iCnt;
        initgraph(&gd, &gm, "C:\\TurboC3\\BGI");
        for (iCnt=0; iCnt<3; iCnt++)
        {
                PolygonPoints[iCnt][0] += 320;
                PolygonPoints[iCnt][1] = 240 - PolygonPoints[iCnt][1];
        }
        PolyLine();
        getch();
        Reflect();
        PolyLine();
        getch();
}
```

Output:

Object before reflection:



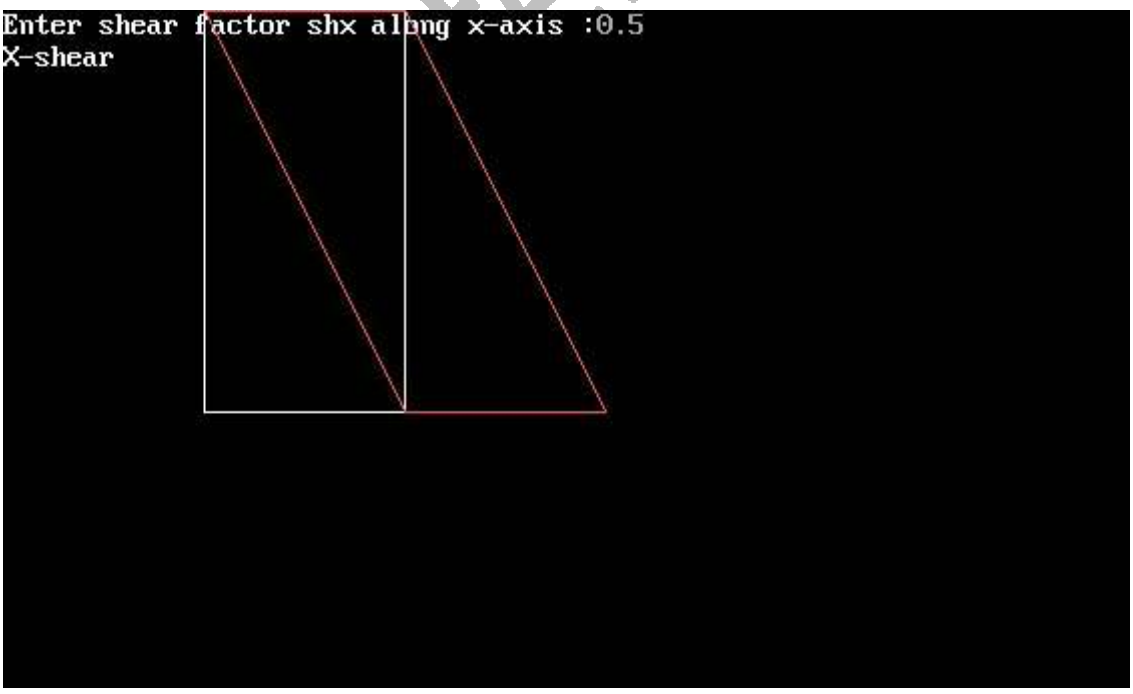
Object after Reflection about y-axis:



6. Program for X-shear:

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

void main()
{
        int gd=DETECT,gm; float shx,shy;
       initgraph(&gd,&gm,"C:\\TurboC3\\BGI");
       printf("Enter shear factor shx along x-axis :");
       scanf("%f",&shx);
        line(100,0,200,0);
        line(200,0,200,200);
        line(200,200,100,200);
        line(100,200,100,0);
        printf("X-shear");
        setcolor(12);
        line((100+(0*shx)),0,(200+(0*shx)),0);
        line((200+(0*shx)),0,(200+(200*shx)),200);
        line((200+(200*shx)),200,(100+(200*shx)),200);
        line((100+(200*shx)),200,(100+(0*shx)),0);
        getch();
}
```

Output:

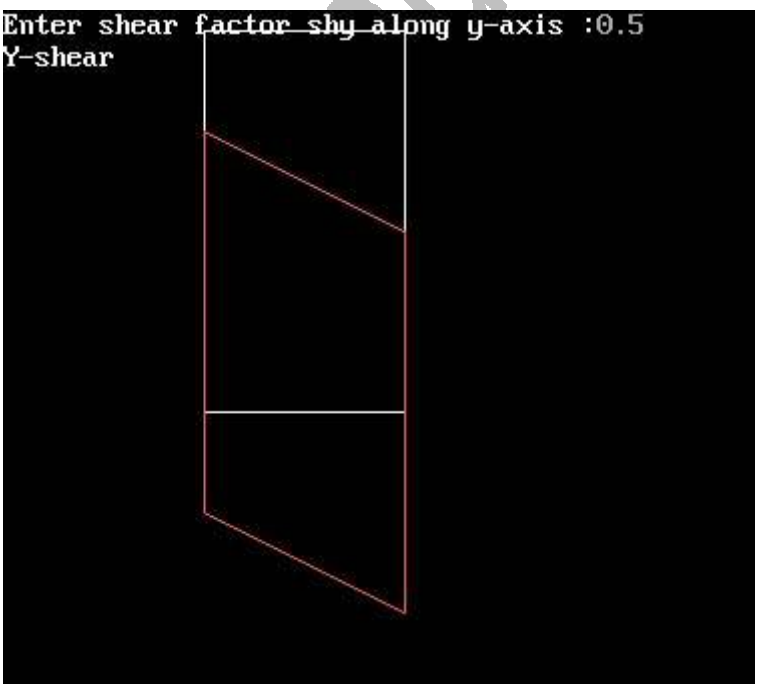


In above output, red lined rectangle denotes object after x-shear transformation.

7. Program for Y-shear:

```
#include<stdio.h>
#include<conio.h>
#include<dos.h>
#include<graphics.h>
void main()
{
        int gd=DETECT,gm; float shx,shy;
        initgraph(&gd,&gm,"C:\\TurboC3\\BGI");
        printf("Enter shear factor shy along y-axis :");
        scanf("%f",&shy);
        line(100,10,200,10);
        line(200,10,200,200);
        line(200,200,100,200);
        line(100,200,100,10);
        printf("Y-shear");
        setcolor(12);
        line(100,10+(shy*100),200,10+(shy*200));
        line(200,10+(shy*200),200,200+(shy*200));
        line(200,200+(shy*200),100,200+(shy*100));
        line(100,200+(shy*100),100,10+(shy*100));
        getch();
        closegraph();

}
```

Output:



red rectangle denotes object after y-shear.

Experiment-8

Coloring the Pictures

Note: For filling a given picture or object with color's, we can do it in two ways in C programming. The two ways are given below:

i. Using filling algorithms such as Floodfill algorithm, Boundary fill algorithm and scanline polygon fill algorithm, we can color the objects.

ii. Using inbuilt graphics functions such as floodfill(),setfillstyle() we can fill the object with color's directly without using any filling algorithm.

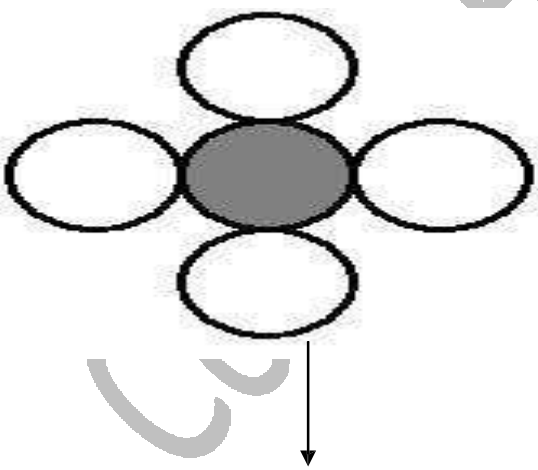
8-i: By using the concept of flood fill algorithm, Write a C- program for filling a given rectangle object with color?

Aim: To implement flood fill algorithm for filling a rectangle with given color.

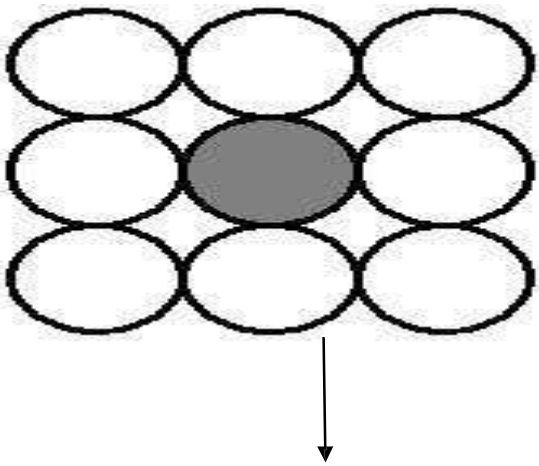
Description:

Sometimes we want to fill in (recolor) an area that is not defined within a single color boundary. We paint such areas by replacing a specified interior color instead of searching for a boundary color value. This approach is called a flood-fill algorithm.

1. We start from a specified interior pixel (x, y) and reassign all pixel values that are currently set to a given interior color with the desired fill color.
2. If the area has more than one interior color, we can first reassign pixel values so that all interior pixels have the same color.
3. Using either 4-connected or 8-connected approach, we then step through pixel positions until all interior pixels have been repainted.



4-connected approach



8-connected approach

Algorithm:

4-connected approach:

1. We can implement flood fill algorithm by using recursion.
2. First all the pixels should be reassigned to common color. here common color is black.
3. Start with a point inside given object, check the following condition:

   if(getpixel(x,y)==old_col)---old_col is common color
4. If above condition is satisfied, then following 4 steps are followed in filling the object.

   //Recursive 4-way floodfill.

```
putpixel(x,y,fill_col);
flood(x+1,y,fill_col,old_col);
flood(x-1,y,fill_col,old_col);
flood(x,y+1,fill_col,old_col);
flood(x,y-1,fill_col,old_col);
```

8-connected approach:

1. We can implement flood fill algorithm by using recursion.
2. First all the pixels should be reassigned to common color. here common color is black.
3. Start with a point inside given object, check the following condition:

   if(getpixel(x,y)==old_col)---old_col is common color
4. If above condition is satisfied, then following 8 steps are followed in filling the object.

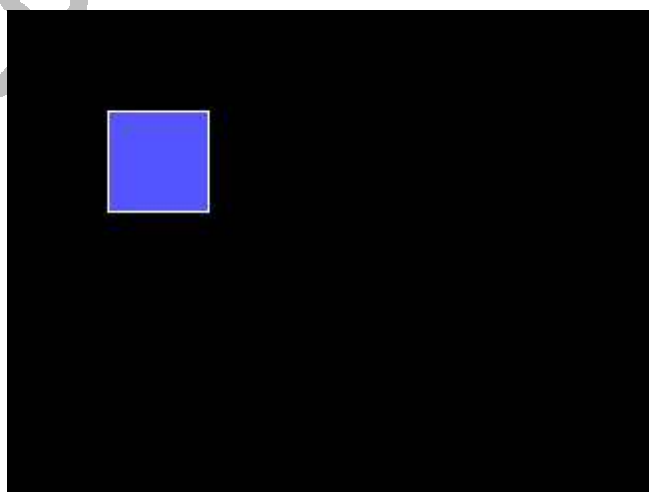
   //Recursive 4-way floodfill.

```
putpixel(x,y,fill_col);
flood(x+1,y,fill_col,old_col);
flood(x-1,y,fill_col,old_col);
flood(x,y+1,fill_col,old_col);
flood(x,y-1,fill_col,old_col);
flood(x + 1, y - 1, fill_col, old_col);
flood(x + 1, y + 1, fill_col, old_col);
flood(x - 1, y - 1, fill_col, old_col);
flood(x - 1, y + 1, fill_col, old_col);
```

Program for 4-connected flood fill:

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

void flood(int,int,int,int);

void main()
{
        int gd,gm=DETECT;
        clrscr();
        detectgraph(&gd,&gm);
        initgraph(&gd,&gm,"C:\\TurboC3\\BGI");
        rectangle(50,50,100,100);
        flood(55,55,9,0);
        getch();
}

void flood(int x,int y, int fill_col, int old_col)
{
        if(getpixel(x,y)==old_col)
        {
                delay(10);
                putpixel(x,y,fill_col);
                flood(x+1,y,fill_col,old_col);
                flood(x-1,y,fill_col,old_col);
                flood(x,y+1,fill_col,old_col);
                flood(x,y-1,fill_col,old_col);
        }
}
```

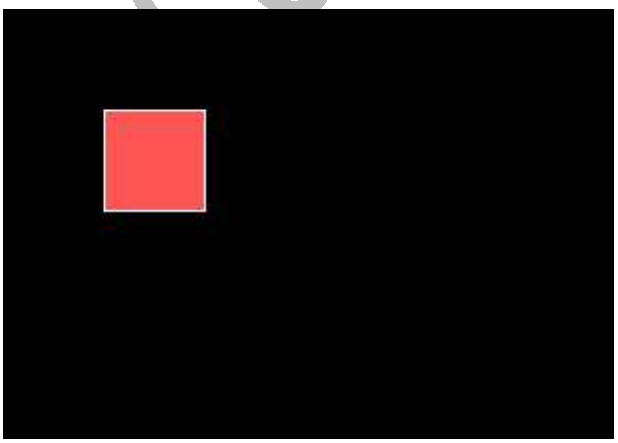
Output:



Program for 8-connected flood fill:

```
#include<stdio.h>
#include<conio.h>
#include<graphics.h>
#include<dos.h>
void flood(int,int,int,int);
void main()
{
        int gd,gm=DETECT;
        clrscr();
        detectgraph(&gd,&gm);
        initgraph(&gd,&gm,"C:\\TurboC3\\BGI");
        rectangle(50,50,100,100);
        flood(55,55,12,0);
        getch();
}
void flood(int x,int y, int fill_col, int old_col)
{
        if(getpixel(x,y)==old_col)
        {
                delay(10); putpixel(x,y,fill_col);
                flood(x+1,y,fill_col,old_col);
                flood(x-1,y,fill_col,old_col);
                flood(x,y+1,fill_col,old_col);
                flood(x,y-1,fill_col,old_col);
                flood(x + 1, y - 1, fill_col, old_col);
                flood(x + 1, y + 1, fill_col, old_col);
                flood(x - 1, y - 1, fill_col, old_col);
                flood(x - 1, y + 1, fill_col, old_col);

        }
}
```

Output:



8-ii: By using the concept of Boundary fill algorithm, Write a C- program for filling a given rectangle object with color?

Aim: To implement Boundary fill algorithm for filling a rectangle with given color.

Description:

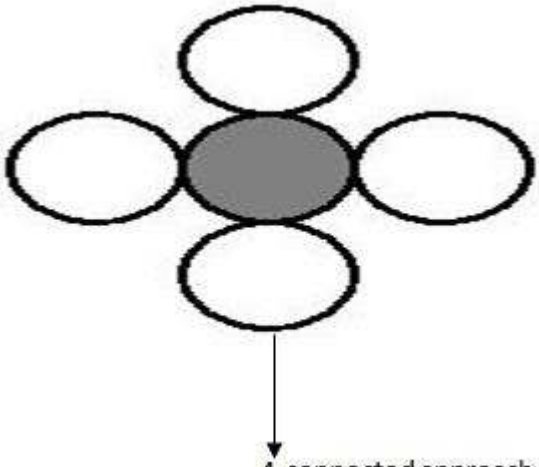
Start at a point inside a region and paint the interior outward toward the boundary. If the boundary is specified in a single color, the fill algorithm processed outward pixel by pixel until the boundary color is encountered.

A boundary-fill procedure accepts as input the coordinate of the interior point (x, y), a fill color, and a boundary color.

Algorithm:

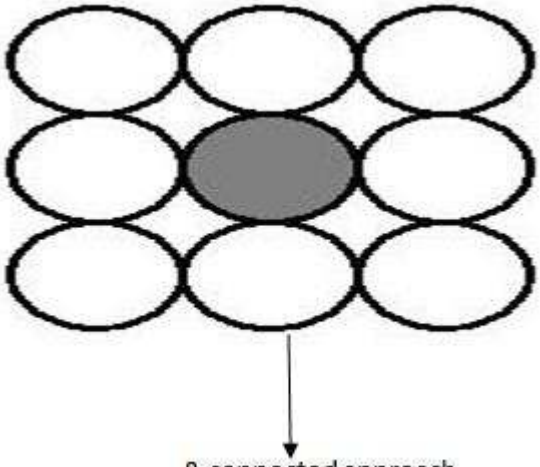
The following steps illustrate the idea of the recursive boundary-fill algorithm:

1. Start from an interior point.
2. If the current pixel is not already filled and if it is not an edge point, then set the pixel with the fill color, and store its neighboring pixels (**4 or 8-connected**). Store only neighboring pixel that is not already filled and is not an edge point.
3. Select the next pixel from the stack, and continue with step 2.



4-connected approach

In 4 connected approach, we can fill an object in only 4 directions. We have 4 possibilities for proceeding to next pixel from current pixel.



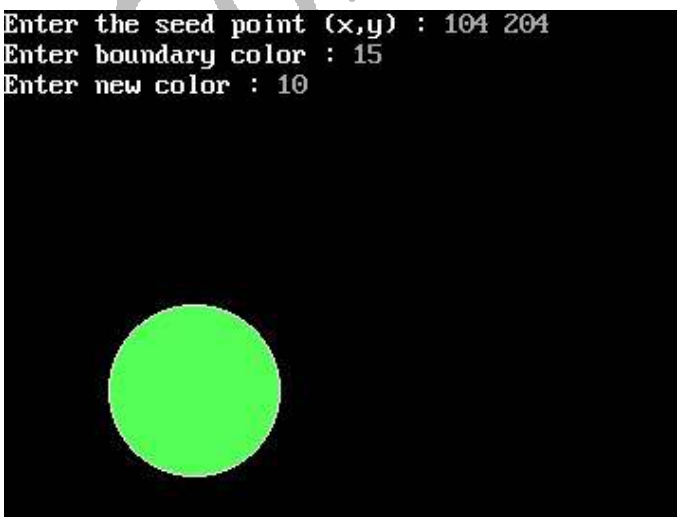
8-connected approach

In 8 connected approach, we can fill an object in 8 directions. We have 8 possibilities for proceeding to next pixel from current pixel.

Program for 4-connected Boundary fill Algorithm:

```
#include<stdio.h>
#include<conio.h>
#include<graphics.h>
#include<dos.h>
void boundary_fill(int x, int y, int fcolor, int bcolor)
{
   if ((getpixel(x, y) != bcolor) && (getpixel(x, y) != fcolor))
    {                delay(10);
                     putpixel(x, y, fcolor);
                     boundary_fill(x + 1, y, fcolor, bcolor);
                     boundary_fill(x - 1, y, fcolor, bcolor);
                     boundary_fill(x, y + 1, fcolor, bcolor);
                     boundary_fill(x, y - 1, fcolor, bcolor);
   }
}
void main()
{
        int x, y, fcolor, bcolor;
        int gd=DETECT,gm;
        initgraph(&gd, &gm, "C:\\TurboC3\\BGI");
        printf("Enter the seed point (x,y) : ");
        scanf("%d%d", &x, &y);
        printf("Enter boundary color : ");
        scanf("%d", &bcolor);
        printf("Enter new color : ");
        scanf("%d", &fcolor);
        circle(100,200,45);
        boundary_fill(x,y,fcolor,bcolor);
        getch();
}
```

Output:



Program for 8 connected Boundary fill Algorithm:

```
#include<stdio.h>
#include<conio.h>
#include<graphics.h>
#include<dos.h>
void boundary_fill(int x, int y, int fcolor, int bcolor)
{

   if ((getpixel(x, y) != bcolor) && (getpixel(x, y) != fcolor))
    {                    delay(10);
                         putpixel(x, y, fcolor);
                         boundary_fill(x + 1, y, fcolor, bcolor);
                         boundary_fill(x , y+1, fcolor, bcolor);
                         boundary_fill(x+1, y + 1, fcolor, bcolor);
                         boundary_fill(x-1, y - 1, fcolor, bcolor);
                         boundary_fill(x-1, y, fcolor, bcolor);
                         boundary_fill(x , y-1, fcolor, bcolor);
                         boundary_fill(x-1, y + 1, fcolor, bcolor);
                         boundary_fill(x+1, y - 1, fcolor, bcolor);

     }
}
void main()
{
        int x, y, fcolor, bcolor;
        int gd=DETECT,gm;
        initgraph(&gd, &gm, "C:\\TurboC3\\BGI");
        printf("Enter the seed point (x,y) : ");
        scanf("%d%d", &x, &y);
        printf("Enter boundary color : ");
        scanf("%d", &bcolor);
        printf("Enter new color : ");
        scanf("%d", &fcolor);
        rectangle(50,50,100,100);
        boundary_fill(x,y,fcolor,bcolor);
        getch();
}
```

Output:

8-iii: By using the concept of Scan line polygon fill algorithm, write a C- program for filling a given object with color?

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

Description:

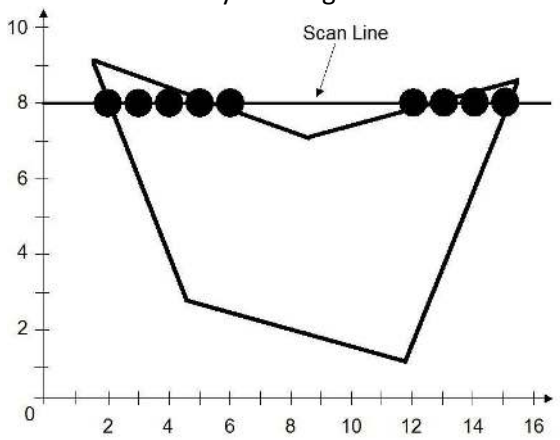
The basic scan-line algorithm is as follows:

Find the intersections of the scan line with all edges of the polygon
Sort the intersections by increasing x coordinate
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:

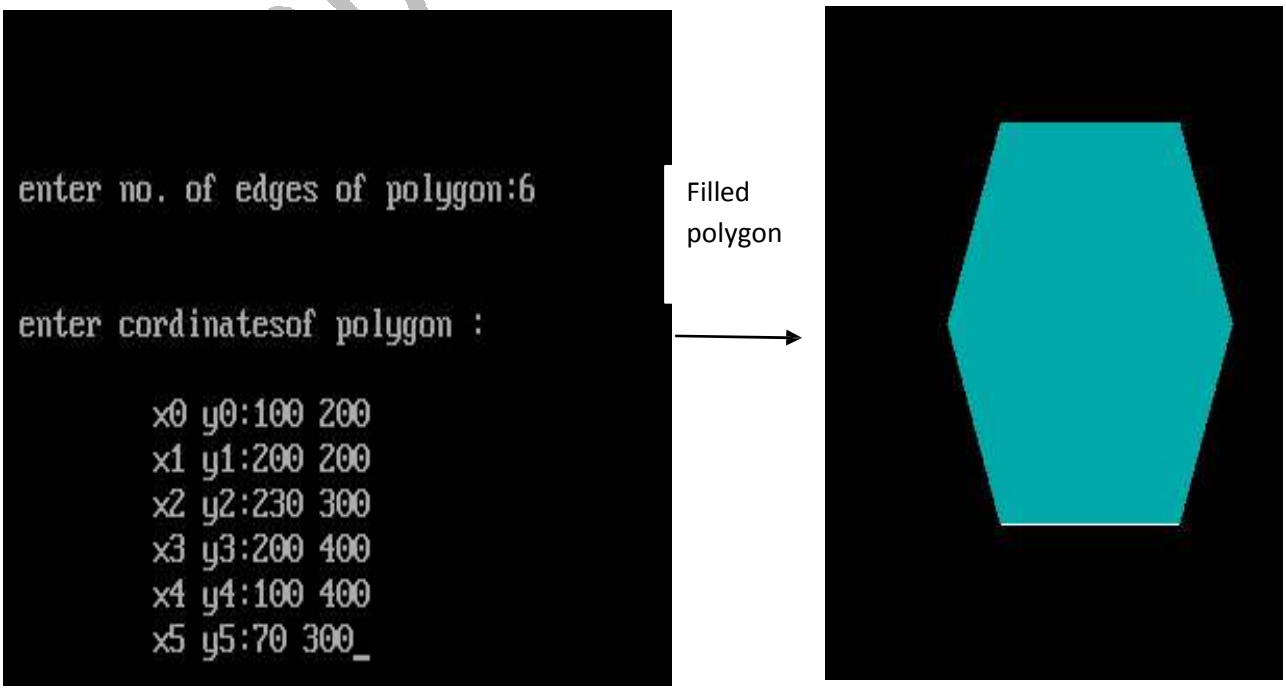
1. the horizontal scanning of the polygon from its lowermost to its topmost vertex
2. identify the edge intersections of scan line with polygon
3. Build the edge table
   a. Each entry in the table for a particular scan line contains the maximum y value for that edge, the x-intercept value (at the lower 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 scanline with polygon edges.

Program for scan line polygon fill algorithm:

```
#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 cordinates 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:



8-iv: Different graphic Functions are available in C-Language for filling a given object with colors. Using the graphic functions, Write a C program for

i. Filling a circle with any color?
ii. Filling a polygon with any color?

Aim: To fill a given object with colors by using inbuilt graphics functions.

Description:

For filling a given object with colors, we have the following graphics functions.

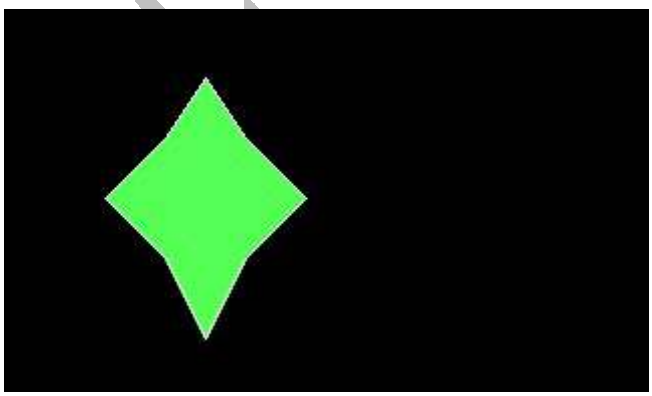
1. Fillpoly
2. Floodfill
3. Setfillstyle
4. Setcolor
5. fillellipse

Fillpoly: Fillpoly function is used to draw a polygon of any number of vertices and fills that polygon with colors. It requires same arguments as drawpoly function. fillpoly fills the polygon by using current fill pattern and color which can be changed using setfillstyle function.

Program illustrating the use of Fillpoly function:

```
#include <graphics.h>
#include <conio.h>
 main()
{
  int gd=DETECT,gm,points[]={100,200,120,230,150,260,120,290,100,330,80,290,50,260,80,230,100,200};
   initgraph(&gd, &gm, "C:\\TurboC3\\BGI");
   setfillstyle(SOLID_FILL,10); //giving the type of fill
   fillpoly(9, points); //drawing a polygon of 9 vertices with coordinates defined in points array and filling
                        with color    defined in setfillstyle function.
   getch();
  closegraph();
  return 0;
}
```

Output:



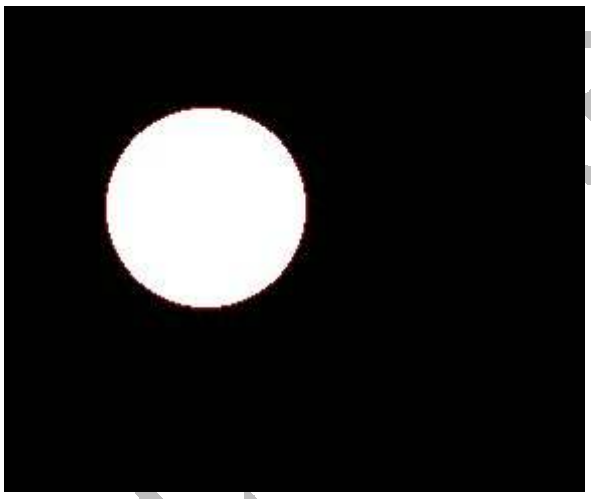
Floodfill: floodfill function is used to fill an enclosed area of an object. Current fill pattern and fill color is used to fill the area.(x, y) is any point on the screen if (x,y) lies inside the area then inside will be filled otherwise outside will be filled,border specifies the color of boundary of area. To change fill pattern and fill color use setfillstyle.

Syntax: floodfill(x,y,boundarycolor);

Program illustrating the use of Floodfill function:

```
#include<graphics.h>
#include<conio.h>
main()
{
        int gd = DETECT, gm;
         initgraph(&gd, &gm, "C:\\TurboC3\\BGI");
         setcolor(RED);
         circle(100,100,50);
         floodfill(100,100,RED);
         getch();
         closegraph();
         return 0;
}
```

Output:



Setfillstyle: setfillstyle function sets the current fill pattern and fill color.

Syntax: setfillstyle(pattern,color);

1) Here, pattern refers to type of filling. The pattern value may be integer value or its corresponding pattern name in CAPS. The patterns available are listed below:

EMPTY_FILL----0,
SOLID_FILL----1,
LINE_FILL-----2,
LTSLASH_FILL--3,
SLASH_FILL----4,
BKSLASH_FILL--5,
LTBKSLASH_FILL-6,
HATCH_FILL-----7,
XHATCH_FILL----8,
INTERLEAVE_FILL-9,
WIDE_DOT_FILL--10,
CLOSE_DOT_FILL-11,
USER_FILL------12.[integer value from 0 to 12 refers to its corresponding pattern name]

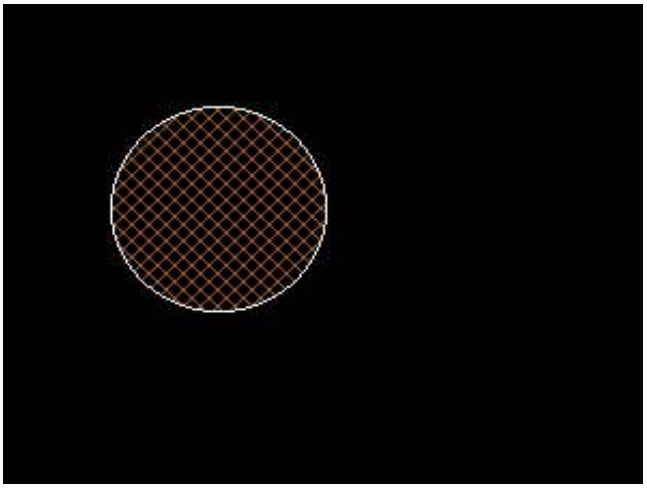
2) Color refers to fill color .The below table shows color names and their equivalent integer values.

Constant	Value
BLACK	0
BLUE	1
GREEN	2
CYAN	3
RED	4
MAGENTA	5
BROWN	6
LIGHTGRAY	7
DARKGRAY	8
LIGHTBLUE	9
LIGHTGREEN	10
LIGHTCYAN	11
LIGHTRED	12
LIGHTMAGENTA	13
YELLOW	14
WHITE	15
BLINK	128

Program illustrating the use of setfillstyle:

```
#include<graphics.h>
#include<conio.h>
main()
{
      int gd = DETECT, gm;
      initgraph(&gd, &gm, "C:\\TurboC3\\BGI");
      setfillstyle(XHATCH_FILL, RED);
      circle(100, 100, 50);
      floodfill(100, 100, WHITE);
      getch();
     closegraph();
}
```

Output



Setcolor: In Turbo Graphics each color is assigned a number. Total 16 colors are available. Strictly speaking number of available colors depends on current graphics mode and driver.

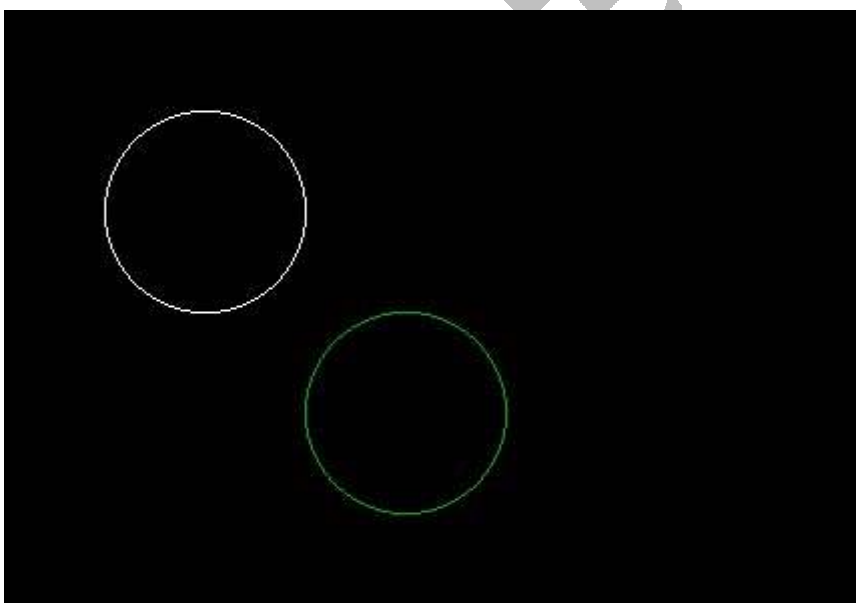
For Example :- BLACK is assigned 0, RED is assigned 4 etc. setcolor function is used to change the current drawing color.e.g. setcolor(RED) or setcolor(4) changes the current drawing color to RED. default drawing color is WHITE.

Syntax: setcolor(color name or its equivalent integer value) –setcolor(GREEN);

Program illustrating the use of setcolor function:

```
#include<graphics.h>
#include<conio.h>
main()
{
        int gd = DETECT, gm;
        initgraph(&gd,&gm,"C:\\TurboC3\\BGI");
        circle(100,100,50);       /* drawn in white color */
        setcolor(GREEN);
        circle(200,200,50);       /* drawn in green color  */
        getch();
        closegraph();
        return 0;
}
```

Output:



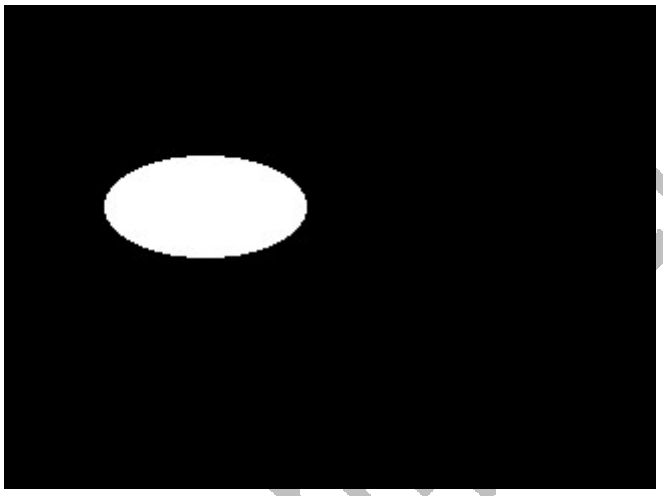
Fill ellipse: This function draws and fills the ellipse with default color White. If we want to fill another color means, we have to use setfillstyle function.

Syntax: fillellipse(x, y, xradius, yradius);

Program illustrating the use of fillellipse function:

```
#include<graphics.h>
#include<conio.h>
 main()
{
        int gd = DETECT, gm;
        initgraph(&gd, &gm, "C:\\TurboC3\\BGI");
        fillellipse(100, 100, 50, 25);
        getch();
        closegraph();
        return 0;
}
```

Output:



Experiment 9

Three Dimensional Transformations

9: Write a C-program for performing the basic transformations such as translation, Scaling, Rotation for a given 3D object?

Aim: To apply the basic transformations such as translation, Scaling, Rotation for a given 3D object.

Description: We have to perform transformations on 3D objects. Here we perform transformations on a line segment.

The transformations are:

Translation
Scaling
Rotation

Translation: Translation is defined as moving the object from one position to another position along straight line path.

We can move the objects based on translation distances along x and y axis. tx denotes translation distance along x-axis and ty denotes translation distance along y axis.

Translation Distance: It is nothing but by how much units we should shift the object from one location to another along x, y-axis.

Consider (x,y) are old coordinates of a point. Then the new coordinates of that same point (x',y') can be obtained as follows:

X'=x+tx
Y'=y+ty
Z'=z+tz

We denote translation transformation as P.

2. Scaling: scaling refers to changing the size of the object either by increasing or decreasing. We will increase or decrease the size of the object based on scaling factors along x and y-axis.

If (x, y) are old coordinates of object, then new coordinates of object after applying scaling transformation are obtained as:

x'=x*sx
y'=y*sy.
Z'=z*sz.

sx ,sy and sz are scaling factors along x-axis, y-axis and z-axis. we express the above equations in matrix form as:

3. **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 theta.

The Transformation matrices for above 3D transformations are given below:

X-Rotation in 3D

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi & 0 \\ 0 & \sin\phi & \cos\phi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Z-Rotation in 3D

$$\begin{bmatrix} \cos\phi & -\sin\phi & 0 & 0 \\ \sin\phi & \cos\phi & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Scale in 3D

$$\begin{bmatrix} Sx & 0 & 0 & 0 \\ 0 & Sy & 0 & 0 \\ 0 & 0 & Sz & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

(4x4)*(4x1) = (4x1)

Y-Rotation in 3D

$$\begin{bmatrix} \cos\phi & 0 & \sin\phi & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\phi & 0 & \cos\phi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Translation in 3D

$$\begin{bmatrix} 1 & 0 & 0 & Tx \\ 0 & 1 & 0 & Ty \\ 0 & 0 & 1 & Tz \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

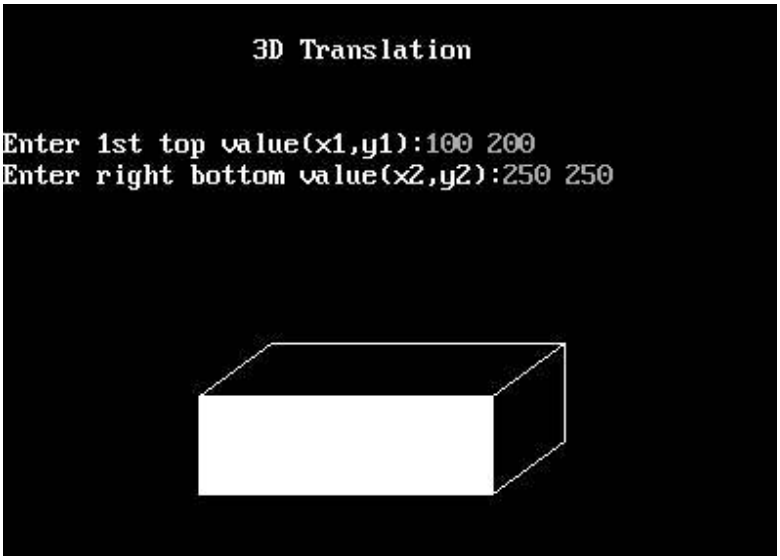
Matrix Multiplication

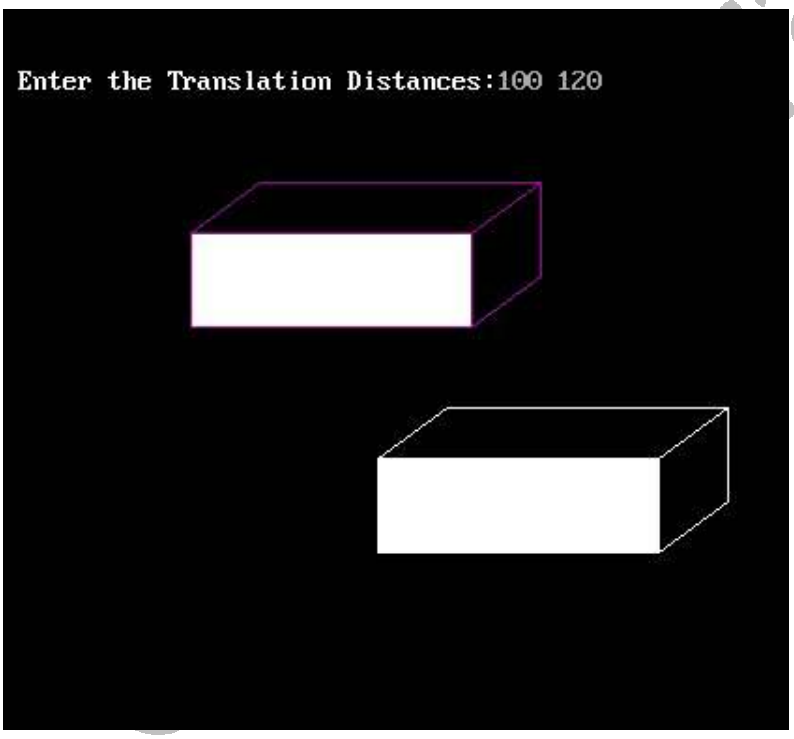
$$\begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ m & n & o & p \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x' \\ y' \\ z' \\ q \end{bmatrix}$$

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 for Translation:

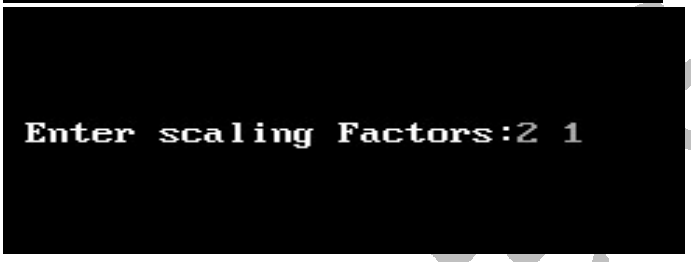


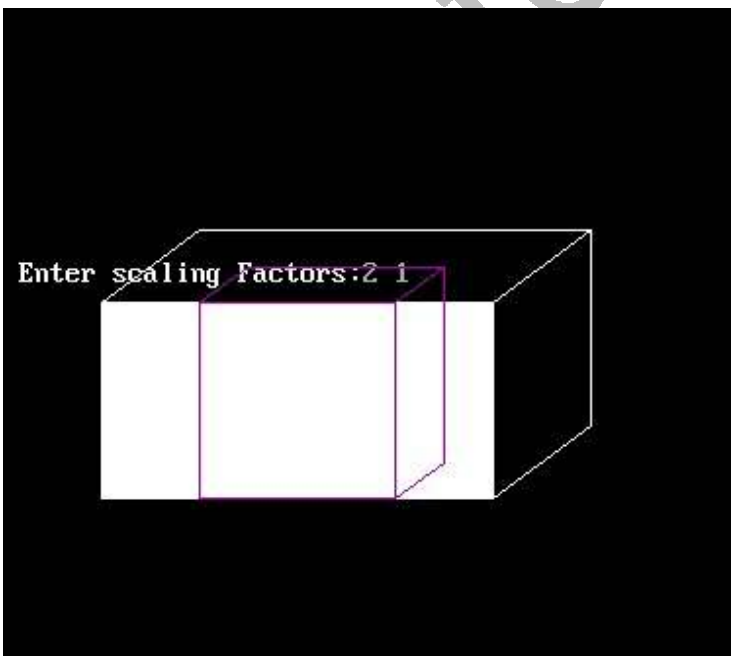


Program for scaling:

```
#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  scale();
void main()
{

            int gd=DETECT,gm,c;
            initgraph(&gd,&gm,"C:\\TurboC3\\BGI");
            printf("\n\t\t3D Scaling\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();
            scale();
            getch();
 }
void draw()
{
   bar3d(x1,y1,x2,y2,depth,1);
}
void scale()
{
            int x,y,a1,a2,b1,b2,dep;
            printf("\n\n Enter scaling Factors:");
            scanf("%d%d",&x,&y);
            a1=mx+(x1-mx)*x;
            a2=mx+(x2-mx)*x;
            b1=my+(y1-my)*y;
            b2=my+(y2-my)*y;
            dep=(a2-a1)/4;
            bar3d(a1,b1,a2,b2,dep,1);
            setcolor(5);
            draw();
}
```

Output For scaling:



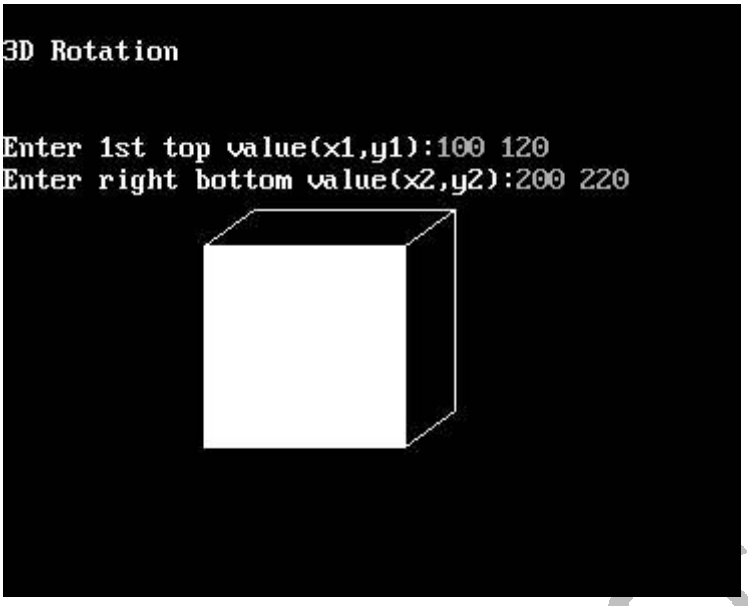


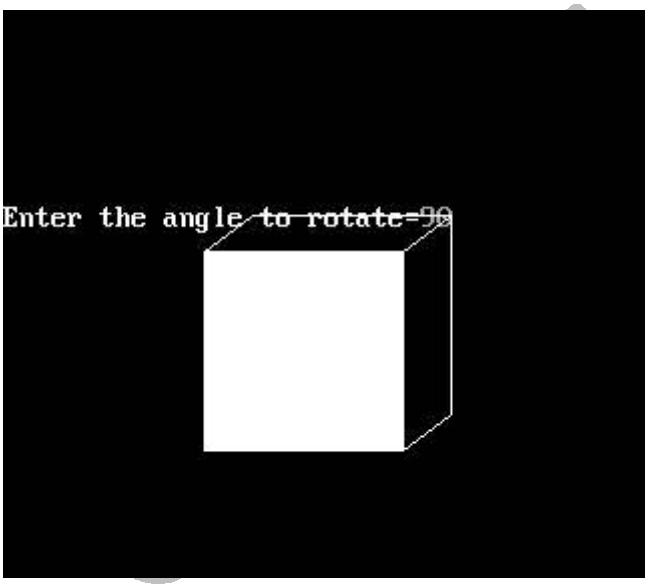
Program for Rotation:

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
#include<graphics.h>
int x1,x2,y1,y2,mx,my,depth;
void draw();
void rotate();
void main()
{
        int gd=DETECT,gm,c;
        initgraph(&gd,&gm,"C:\\TurboC3\\BGI");
        printf("\n3D Transformation Rotating\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();
        rotate();
        getch();
  }
void draw()
{
    bar3d(x1,y1,x2,y2,depth,1);
}
void rotate()
{
        float t;
        int a1,b1,a2,b2,dep;
        printf("Enter the angle to rotate=");
        scanf("%f",&t);
        t=t*(3.14/180);
        a1=mx+(x1-mx)*cos(t)-(y1-my)*sin(t);
        a2=mx+(x2-mx)*cos(t)-(y2-my)*sin(t);
        b1=my+(x1-mx)*sin(t)-(y1-my)*cos(t);
        b2=my+(x2-mx)*sin(t)-(y2-my)*cos(t);
        if(a2>a1)
          dep=(a2-a1)/4;
        else
         dep=(a1-a2)/4;
```

```
        bar3d(a1,b1,a2,b2,dep,1);  setcolor(5);
  }
```

Output:





Experiment 10

Curve Generation

10: Write a C-program for generating a curve for a given set of control points?

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

Description:

A Bézier curve is a parametric curve frequently used in computer graphics and related fields. Generalizations of Bézier curves to higher dimensions are called Bézier surfaces, of which the Bézier triangle is a special case.

In vector graphics, Bézier 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 Bézier curves. Paths are not bound by the limits of rasterized images and are intuitive to modify. Bézier curves are also used in animation as a tool to control motion

Four points $\mathbf{P}_0$, $\mathbf{P}_1$, $\mathbf{P}_2$ and $\mathbf{P}_3$ in the plane or in higher-dimensional space define a cubic Bézier curve. The curve starts at $\mathbf{P}_0$ going toward $\mathbf{P}_1$ and arrives at $\mathbf{P}_3$ coming from the direction of $\mathbf{P}_2$. Usually, it will not pass through $\mathbf{P}_1$ or $\mathbf{P}_2$; these points are only there to provide directional information. The distance between $\mathbf{P}_0$ and $\mathbf{P}_1$ determines "how long" the curve moves into direction $\mathbf{P}_2$ before turning towards $\mathbf{P}_3$.

The explicit form of the curve is:

$$\mathbf{B}(t) = (1-t)^3\mathbf{P}_0 + 3(1-t)^2 t\mathbf{P}_1 + 3(1-t)t^2\mathbf{P}_2 + t^3\mathbf{P}_3 \ , \ t \in [0,1].$$

Program for Bezier curve:

```
#include <stdio.h>
#include <stdlib.h>
#include <graphics.h>
#include <math.h>
void bezier (int x[4], int y[4])
{
   int gd = DETECT, gm;
   int i;
   double t;
   initgraph (&gd, &gm, "C:\\TurboC3\\BGI");
   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], YELLOW);
   getch();
   closegraph();
   return;
}

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

Output:

```
Enter the x- and y-coordinates of the four control points.
100 110
130 120
180 200
240 220_
```

Simple animations using transformations

11. Write C-programs for designing simple animations using transformations?

Aim: To develop programs for making simple animations using transformations like rotation, scaling and translation**.** The simple animations are given below:

i. Circle moving from left to right and vice versa
ii. Man object moving
iii. Wind mill rotation
iv. Man walking
v. Simple animation of football goal

Description:

For moving any object, we incrementally calculate the object coordinates and redraw the picture to give a feel of animation by using for loop.

Suppose if we want to move a circle from left to right means, we have to shift the position of circle along x-direction continuously in regular intervals.

The below programs illustrate the movement of objects by using for loop and also using transformations like rotation, translation etc.

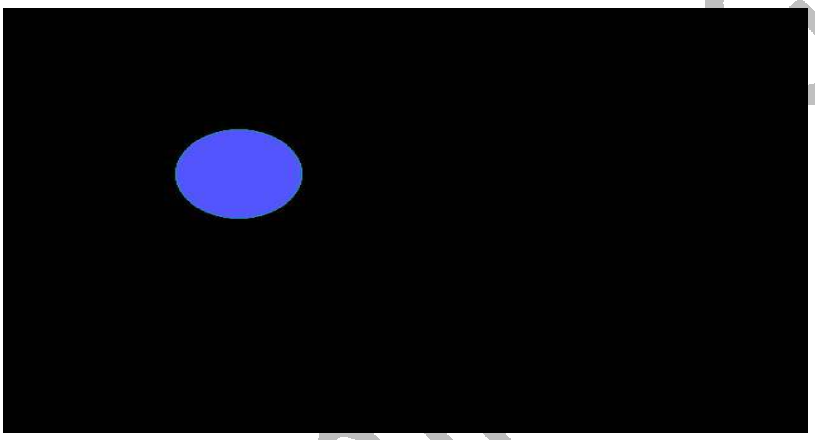
For windmill rotation, we use 2D rotation concept and formulas.

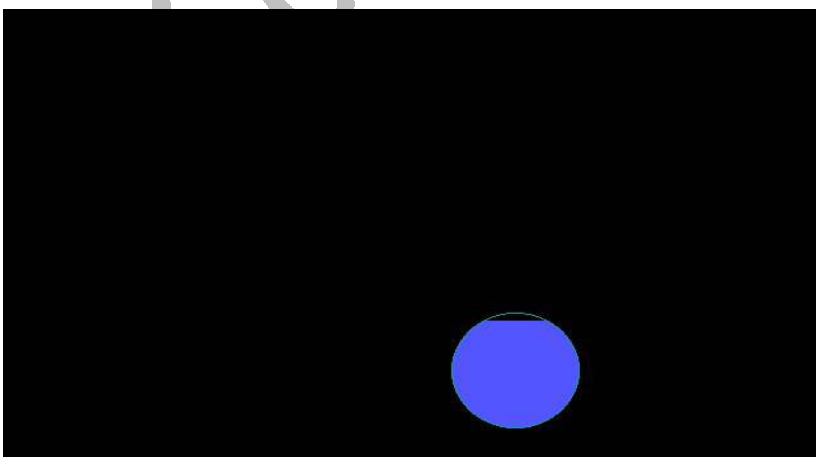
i. Program for moving circle in different directions

```
#include<stdio.h>
#include<graphics.h>
#include<conio.h>
#include<dos.h>
void main()
{
    int gd=DETECT,gm,i;
    initgraph(&gd,&gm,"C:\\TurboC3\\BGI");
    //for moving circle from left to right,the following loop works
    for(i=50;i<=getmaxx();i++)
    {
      setcolor(3);
      setfillstyle(SOLID_FILL,9);
      circle(50+i,50,50);
      floodfill(52+i,52,3);
      delay(20);
      cleardevice();
    }
    //for moving circle from right to left, the following loop works
    for(i=getmaxy();i>=0;i--)
```

```
    {
      setcolor(3);
      setfillstyle(SOLID_FILL,9);
      circle(i,50,50);
      floodfill(i+2,52,3);
      delay(20);
      cleardevice();
    }
    //for moving circle from top to bottom,the following loop works
    for(i=50;i<=getmaxy();i++)
    {
      setcolor(3);
      setfillstyle(SOLID_FILL,9);
      circle(50,i,50);
      floodfill(52,i+2,3);
      delay(20);
      cleardevice();
    }
    //for moving circle from bottom to top,the following loop works
    for(i=getmaxy();i>=0;i--)
    {
      setcolor(3);
      setfillstyle(SOLID_FILL,9);
      circle(50,i,50);
      floodfill(52,i+2,3);
      delay(20);
      cleardevice();
    }
    //for moving circle in diagonal direction,the following loop works
    for(i=50;i<=getmaxx();i++)
    {
      setcolor(3);
      setfillstyle(SOLID_FILL,9);
      circle(i,i,50);
      floodfill(i+2,i+2,3);
      delay(20);
      cleardevice();
    }
    //for moving circle in reverse diagonal direction,the following loop works
    for(i=getmaxx();i>=0;i--)
    {
      setcolor(3);
      setfillstyle(SOLID_FILL,9);
      circle(i,i,50);
      floodfill(i+2,i+2,3);
      delay(20);
      cleardevice();
    }

    getch();
}
```

Output for moving circle:



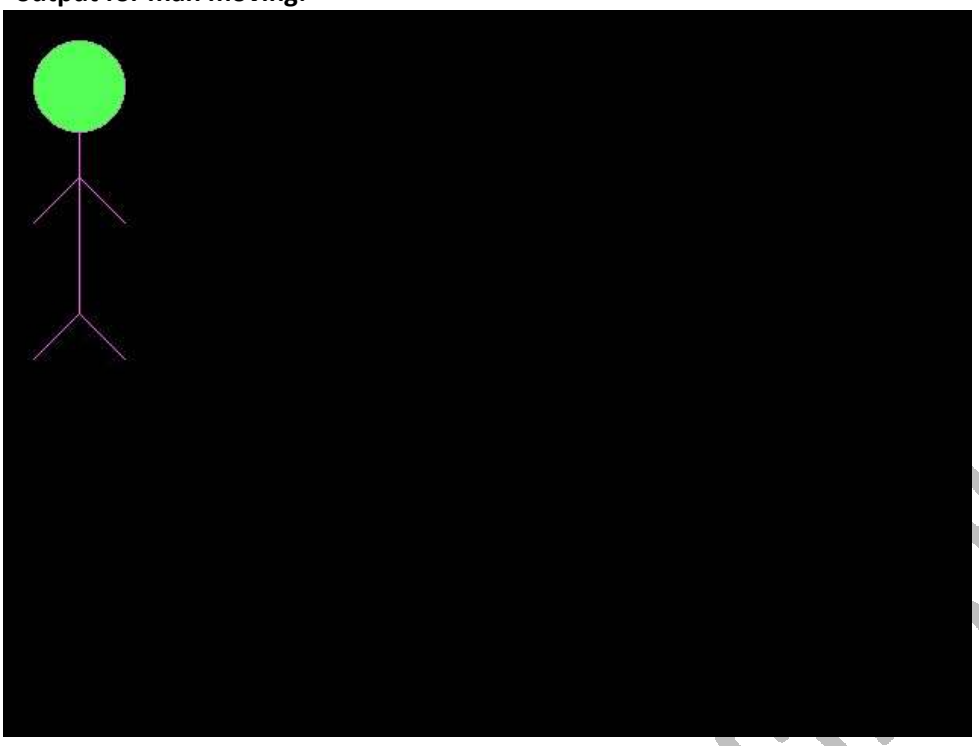


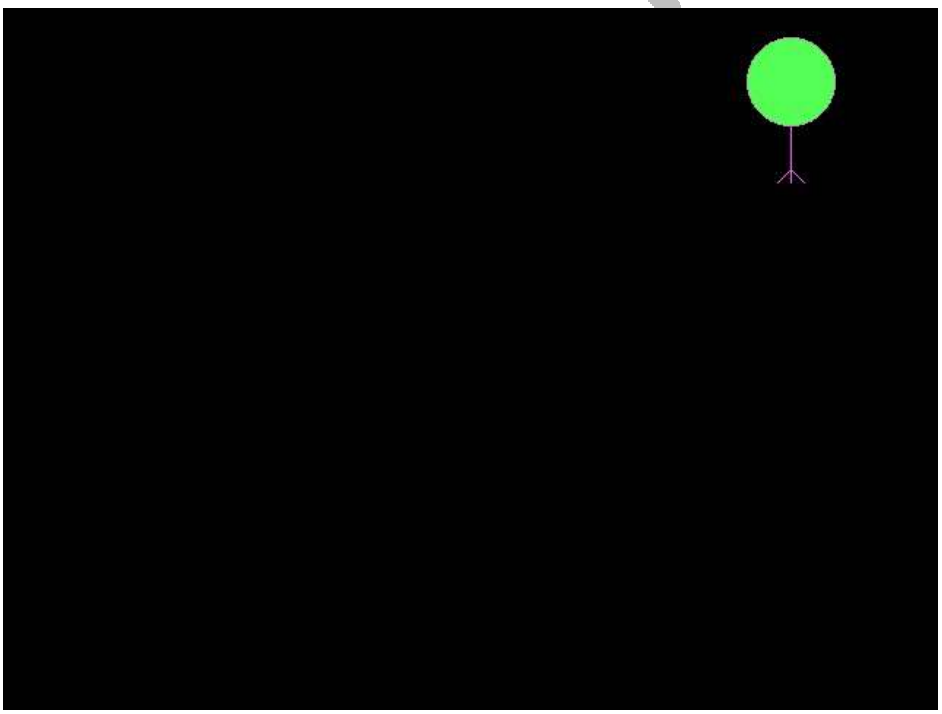
ii. program for man object moving:

```
#include<stdio.h>
#include<graphics.h>
#include<conio.h>
#include<dos.h>
void main()
{
        int gd=DETECT,gm,i;
        initgraph(&gd,&gm,"C:\\TurboC3\\BGI");
        //creation of man object by using circle,line.
        setcolor(7);
        setfillstyle(SOLID_FILL,10);
        circle(50,50,30);  // drawing head
        floodfill(52,52,7);
        setcolor(13);
        line(50,80,50,200); //drawing body
        line(50,110,20,140); //left hand
        line(50,110,80,140); //right hand
        line(50,200,20,230); //left leg
        line(50,200,80,230); //right leg
        // for loop for moving man
        for(i=50;i<=getmaxx();i++)
        {
                        setcolor(7);
                        setfillstyle(SOLID_FILL,10);
                        circle(i,50,30);  // drawing head
                        floodfill(i+2,52,7);
                        setcolor(13);
                        line(i,80,i,200); //drawing body
                        line(i,110,i-30,140); //left hand
                        line(i,110,i+30,140); //right hand
                        line(i,200,i-30,230); //left leg
                        line(i,200,i+30,230); //right leg
                        cleardevice();
                        delay(10);
        }
        //doing simple animation using translation
         for(i=50;i<=getmaxx()/2;i++)
        {
                        setcolor(7);
                        setfillstyle(SOLID_FILL,10);
                        circle(i,50,30);  // drawing head
                        floodfill(i+2,52,7);
                        setcolor(13);
                        line(i,80,i,200); //drawing body
                        line(i,110,i-30,140); //left hand
                        line(i,110,i+30,140); //right hand
                        line(i,200,i-30,230); //left leg
                        line(i,200,i+30,230); //right leg
                        cleardevice(); delay(10);
        }
```

```
        getch();
}
```

output for man moving:



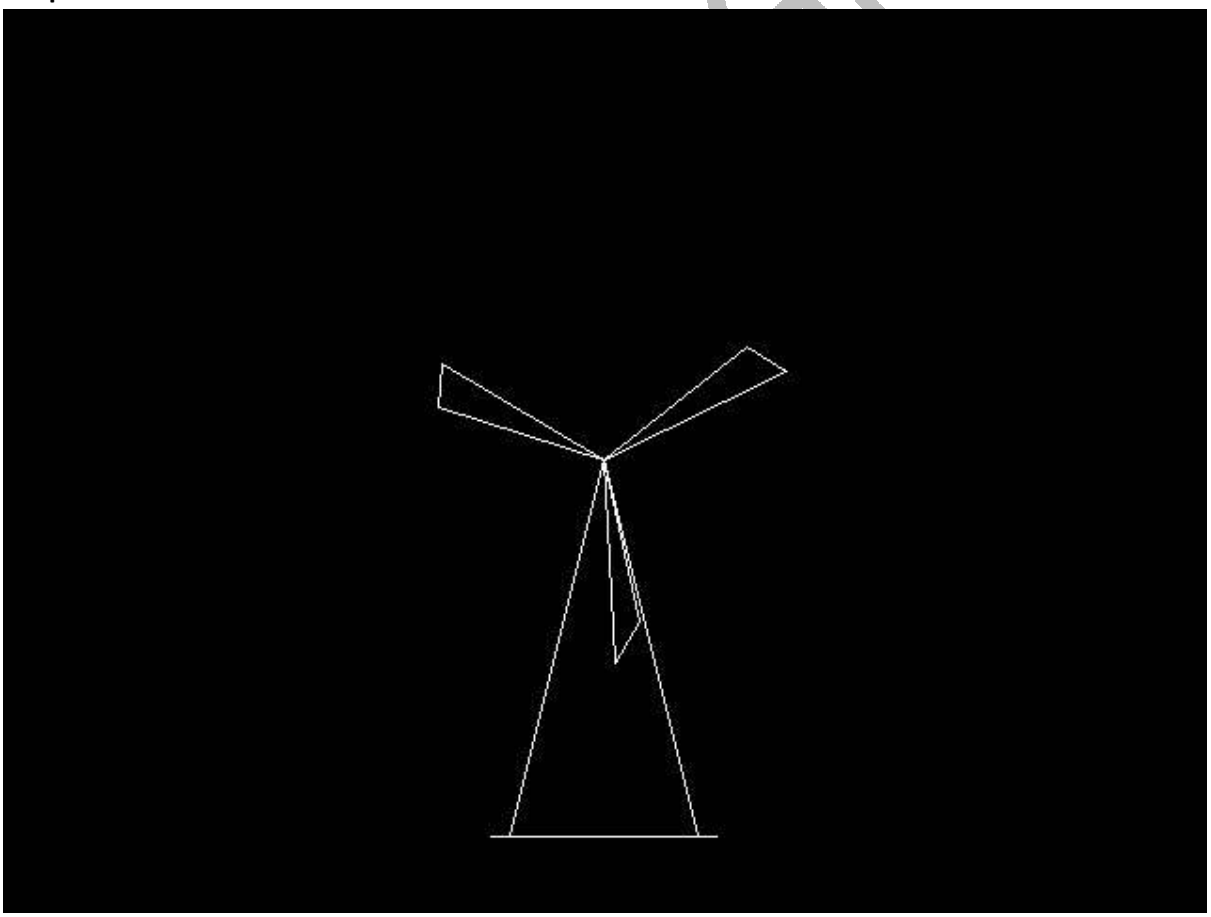


iii.program for windmill rotation:

```
#include<stdio.h>
#include<graphics.h>
#include<math.h>
void wind(float x[7],float y[7]);
void main()
{
        int gd=DETECT,gm;
        float x[7],y[7],maxx,maxy,xw1,yw1,xw2,yw2;
        float theta=30;
        initgraph(&gd,&gm,"C:\\TurboC3\\BGI");
        maxx=getmaxx();
        maxy=getmaxy();
        x[0]=maxx/2;
        y[0]=maxy/2;
        x[1]=y[4]=x[2]=-90;
        y[6]=y[5]=y[1]=60;
        y[2]=35;
        y[3]=-100;
        x[4]=20;
        x[3]=0;
        x[5]=90;
        x[6]=65;
        theta=theta*22/7/180;
        while(kbhit()==0)
        {
                wind(x,y);
                xw1=cos(theta)*x[1]+sin(theta)*y[1];
                yw1=-sin(theta)*x[1]+cos(theta)*y[1];
                xw2=cos(theta)*x[2]+sin(theta)*y[2];
                yw2=-sin(theta)*x[2]+cos(theta)*y[2];
                x[1]=xw1;
                y[1]=yw1;
                x[2]=xw2;
                y[2]=yw2;
                xw1=cos(theta)*x[3]+sin(theta)*y[3];
                yw1=-sin(theta)*x[3]+cos(theta)*y[3];
                xw2=cos(theta)*x[4]+sin(theta)*y[4];
                yw2=-sin(theta)*x[4]+cos(theta)*y[4];
                x[3]=xw1;
                y[3]=yw1;
                x[4]=xw2;
                y[4]=yw2;
                xw1=cos(theta)*x[5]+sin(theta)*y[5];
                yw1=-sin(theta)*x[5]+cos(theta)*y[5];
                xw2=cos(theta)*x[6]+sin(theta)*y[6];
                yw2=-sin(theta)*x[6]+cos(theta)*y[6];
                x[5]=xw1;
                y[5]=yw1;
                x[6]=xw2;
                y[6]=yw2;
```

```
                delay(50);
                cleardevice();
            }
            closegraph();
}

void wind(float x[7],float y[7])
{
        cleardevice();
        line(x[0],y[0],x[0]-50,y[0]+200);
        line(x[0],y[0],x[0]+50,y[0]+200);
        line(x[0]-60,y[0]+200,x[0]+60,y[0]+200);
        line(x[0],y[0],x[0]+x[1],y[0]-y[1]);
        line(x[0],y[0],x[0]+x[2],y[0]-y[2]);
        line(x[0]+x[1],y[0]-y[1],x[0]+x[2],y[0]-y[2]);
        line(x[0],y[0],x[0]+x[3],y[0]-y[3]);
        line(x[0],y[0],x[0]+x[4],y[0]-y[4]);
        line(x[0]+x[3],y[0]-y[3],x[0]+x[4],y[0]-y[4]);
        line(x[0],y[0],x[0]+x[5],y[0]-y[5]);
        line(x[0],y[0],x[0]+x[6],y[0]-y[6]);
        line(x[0]+x[5],y[0]-y[5],x[0]+x[6],y[0]-y[6]);
}
```

Output:

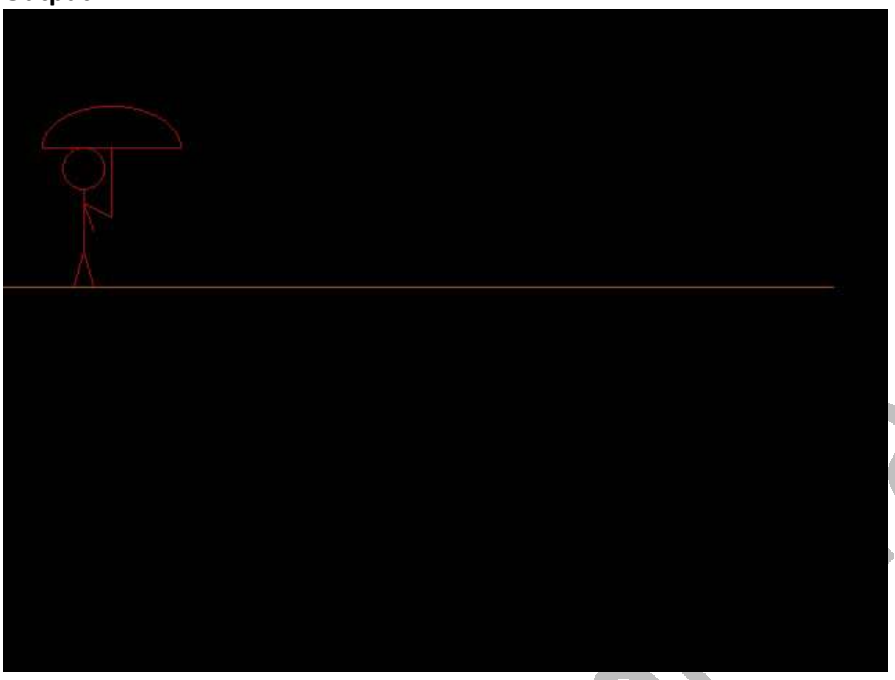


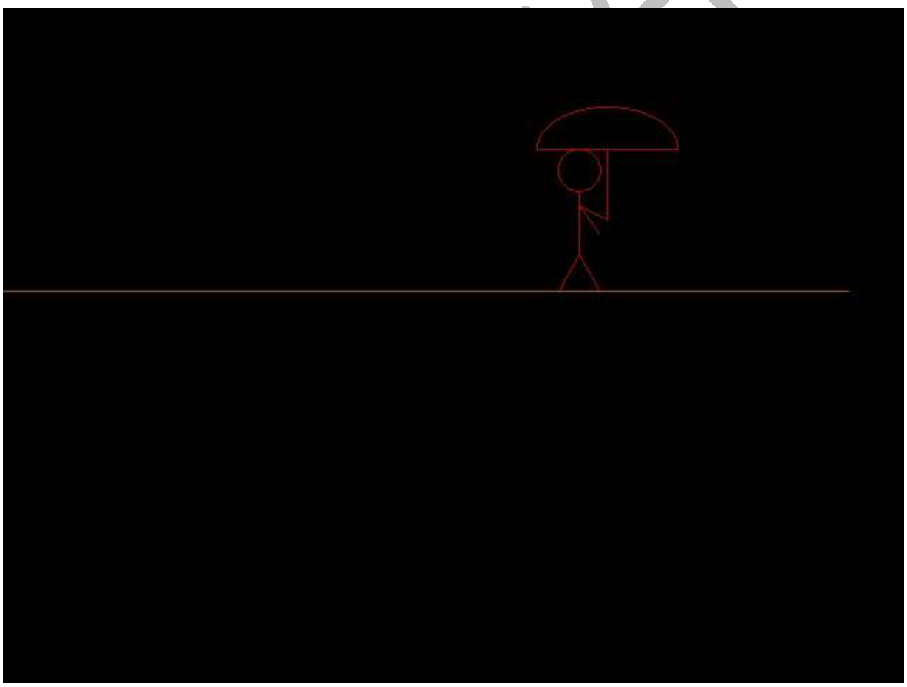
iv. program for man walking:

```
#include<stdio.h>
#include<dos.h>
#include<conio.h>
#include<graphics.h>
#include<stdlib.h>
void main()
{
        int gd = DETECT, gm = DETECT, c = -200, i = 0, x = 40, l = 15, h = 15, ht = 0;
        initgraph(&gd, &gm, "C:\\TurboC3\\BGI");
        cleardevice();
        setcolor(BROWN);
        line(0, 201, 600, 201);
      cont:
        while (!kbhit()) {
            setcolor(4);
            ellipse(x, 100, 0, 180, 50, 30);
            line(x - 50, 100, x + 50, 100);
            line(x, 100, x, 150);
            circle(x - 20, 115, 15);
            line(x - 20, 130, x - 20, 175);
            line(x - 20, 175, x - 20 - l, 200);
            line(x - 20, 175, x - 20 + l, 200);
            line(x - 20, 140, x, 150);
            line(x - 20, 140, x - 20 - h, 160);
            setcolor(0);
            delay(50);
            ellipse(x, 100, 0, 180, 50, 30);
            line(x - 50, 100, x + 50, 100);
            line(x, 100, x, 150);
            circle(x - 20, 115, 15);
            line(x - 20, 130, x - 20, 175);
            line(x - 20, 175, x - 20 - l, 200);
            line(x - 20, 175, x - 20 + l, 200);
            line(x - 20, 140, x, 150);
            line(x - 20, 140, x - 20 - h, 160);
            line(x + 50, 100, x + 50, 200);
            x++;
            l--;
            if (l == -15)
               l = 15;
            if (ht == 1)
               h++;
            else
               h--;

            if (h == 15)
              ht = 0;
            else if (h == -15)
               ht = 1;
```

```
        }
        if (getch() == ' ') {
            while (!kbhit());
            getch();
            goto cont;
        }
}
```

Output:





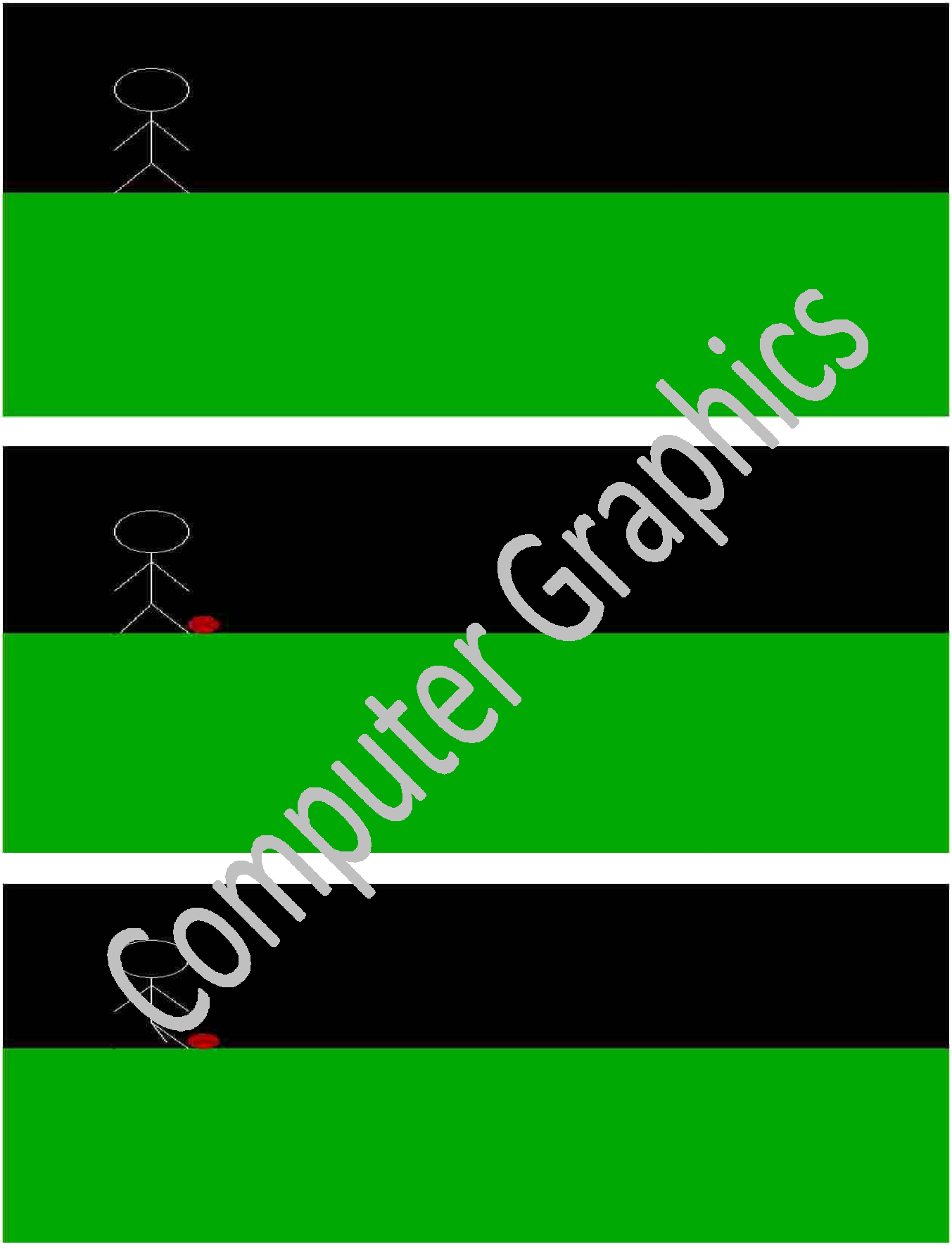
vi. program for simple animation of football goal:

```
#include<stdio.h>
#include<conio.h>
#include<graphics.h>
#include<dos.h>
void main()
{
        int  x=0,gd=DETECT,gm,points[]={0,220,1600,220,1600,900,0,900,0,220};
        float y=0;
        initgraph(&gd,&gm,"C:\\TURBOC3\\BGI");
        setcolor(MAGENTA);
        settextstyle(6,HORIZ_DIR,6);
        outtextxy(200,250,"Hi");
        delay(1000); cleardevice();
        settextstyle(7,VERT_DIR,1);
        outtextxy(200,50,"GET READY FOR ANIMATION");
        delay(1000);
        cleardevice();
        setcolor(GREEN);
        setfillstyle(SOLID_FILL,GREEN);
        fillpoly(5,points);
        setcolor(WHITE);
        circle(100,100,25);
        delay(1000);
        line(100,125,100,185);
        delay(1000);
        line(100,135,125,170);
        delay(1000);
        line(100,135,75,170);
        delay(1000);
        line(100,185,125,220);
        delay(1000);
        line(100,185,75,220);
        delay(1000);
        setcolor(RED);
        setfillstyle(SOLID_FILL,RED);
        fillellipse(135+x,210-y,10,10);
        for(x=0;x<50;x++)
        {
                setcolor(WHITE);
                line(100,185,75+x,220-y);
                delay(100);
                setcolor(BLACK);
                line(100,185,75+x,220-y);
                y=y+0.25;
        }
        setcolor(WHITE);
        line(100,185,125,220);
        line(100,185,75,220);
```

```
for(x=0,y=0;y<100;x++)
        {
                setcolor(RED);
                setfillstyle(SOLID_FILL,RED);
                fillellipse(135+x,210-y,10,10);
                delay(10);
                setcolor(GREEN);
                setfillstyle(SOLID_FILL,GREEN);
                fillpoly(5,points);
                setcolor(BLACK);
                setfillstyle(SOLID_FILL,BLACK);
                fillellipse(135+x,210-y,10,10);
                y=y+0.5;
        }
        for(;x<490;x++)
        {
                setcolor(RED);
                setfillstyle(SOLID_FILL,RED);
                fillellipse(135+x,y,10,10);
                delay(10);
                setcolor(BLACK);
                setfillstyle(SOLID_FILL,BLACK);
                fillellipse(135+x,y,10,10);
                y=y+0.25;
        }
        setcolor(RED);
        setfillstyle(SOLID_FILL,RED);
        fillellipse(135+x,y,10,10);
        delay(2000);
        cleardevice();
        setbkcolor(CYAN);
        settextstyle(3,HORIZ_DIR,10);
        outtextxy(200,80,"GOAL");
        getch();
        closegraph();
}
```

Output:

GET READY FOR ANIMATION


Computer Graphics

12. Key Frame Animation

A key frame in animation and filmmaking is a drawing that defines the starting and ending points of any smooth transition. The drawings are called "frames" because their position in time is measured in frames on a strip of film. A sequence of keyframes defines *which* movement the viewer will see, whereas the position of the keyframes on the film, video or animation defines the timing of the movement. Because only two or three keyframes over the span of a second do not create the illusion of movement, the remaining frames are filled with inbetweens.

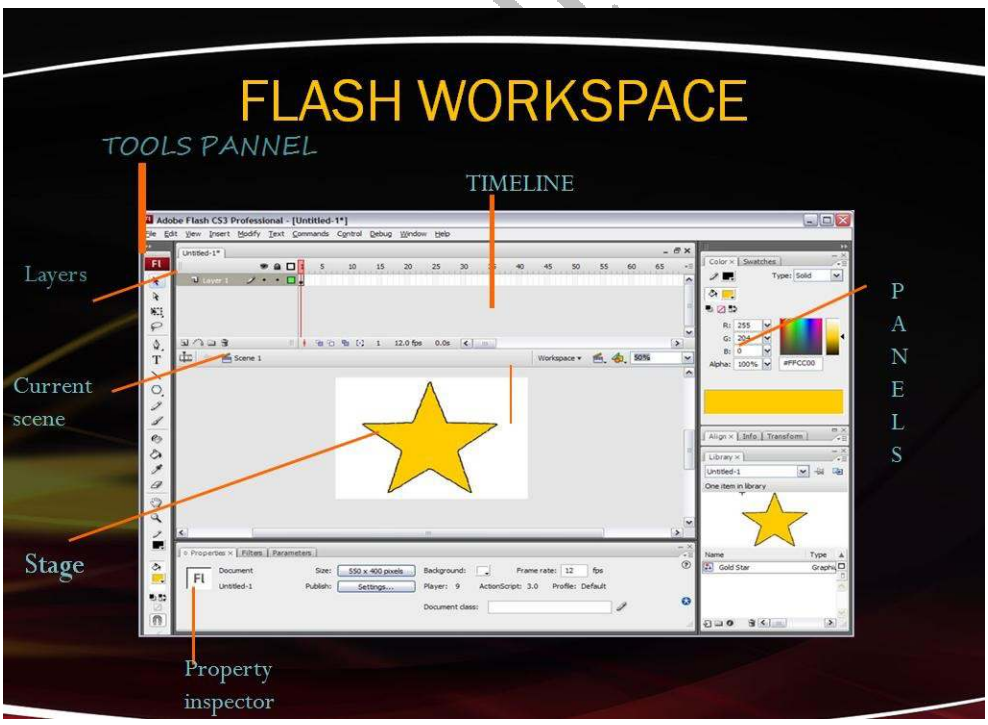
Adobe Flash (formerly called "Macromedia Flash") is a multimedia and software platform used for authoring of vector graphics, animation, games and Rich Internet Applications (RIAs) which can be viewed, played and executed in Adobe Flash Player. Flash is frequently used to add streamed video or audio players, advertisement and interactive multimedia content to web pages, although usage of Flash on websites is declining.

Animation created in flash is organized using a timeline

TIMELINE: a graphical representation of sequence of frames

FLASH- a scripting language….it is a action script..it is used to add interactivity to animation& to build web app's which uses information created in flash..

Flash Workspace:



Stage:

Compose movie content on the Stage.

*Set Stage size by selecting Modify > Document:

*To specify the Stage size in pixels, enter values in the Width and Height boxes. The minimum size is 1 x 1 pixels; the maximum is 2880 x 2880 pixels.

*To minimize document size, click the Contents button to the right of Match (but first create all of the objects on the Stage).

*To set the Stage size to the maximum available print area, click Printer

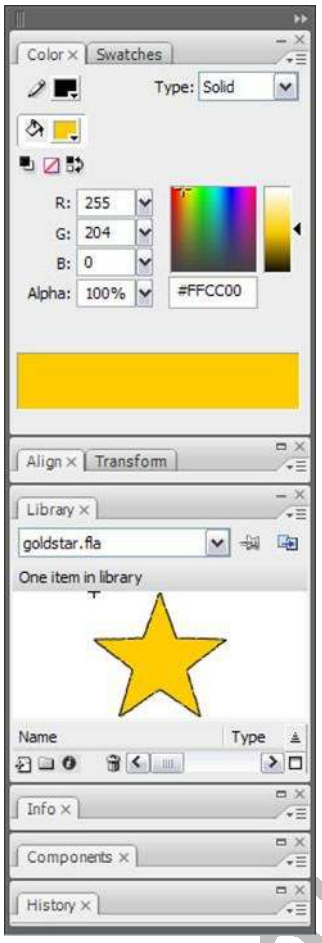
Tools Panel:

Panels:

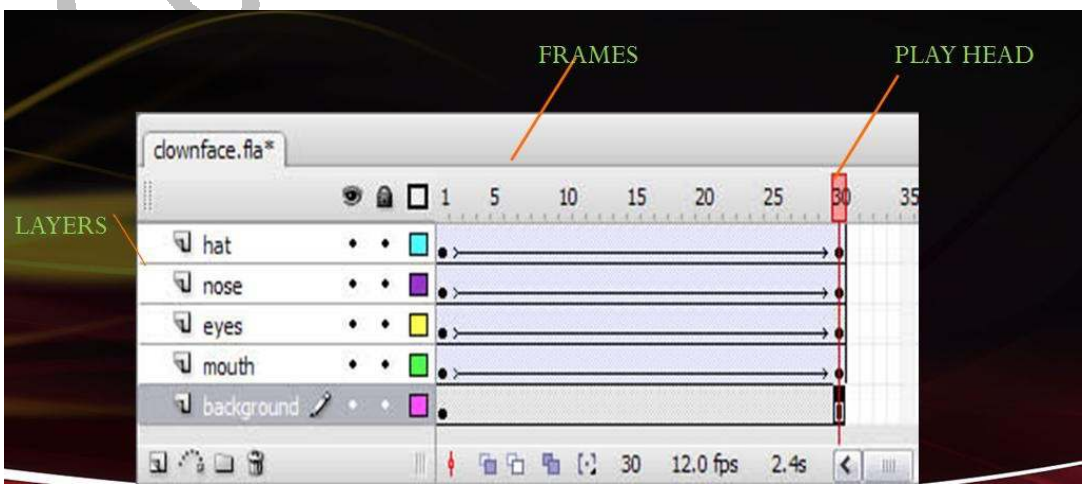
Panels provide additional tools for creating and editing movies.

*Click the panel menu to view additional options for the current panel.

*You can hide or show panels by using the options on the Window menu



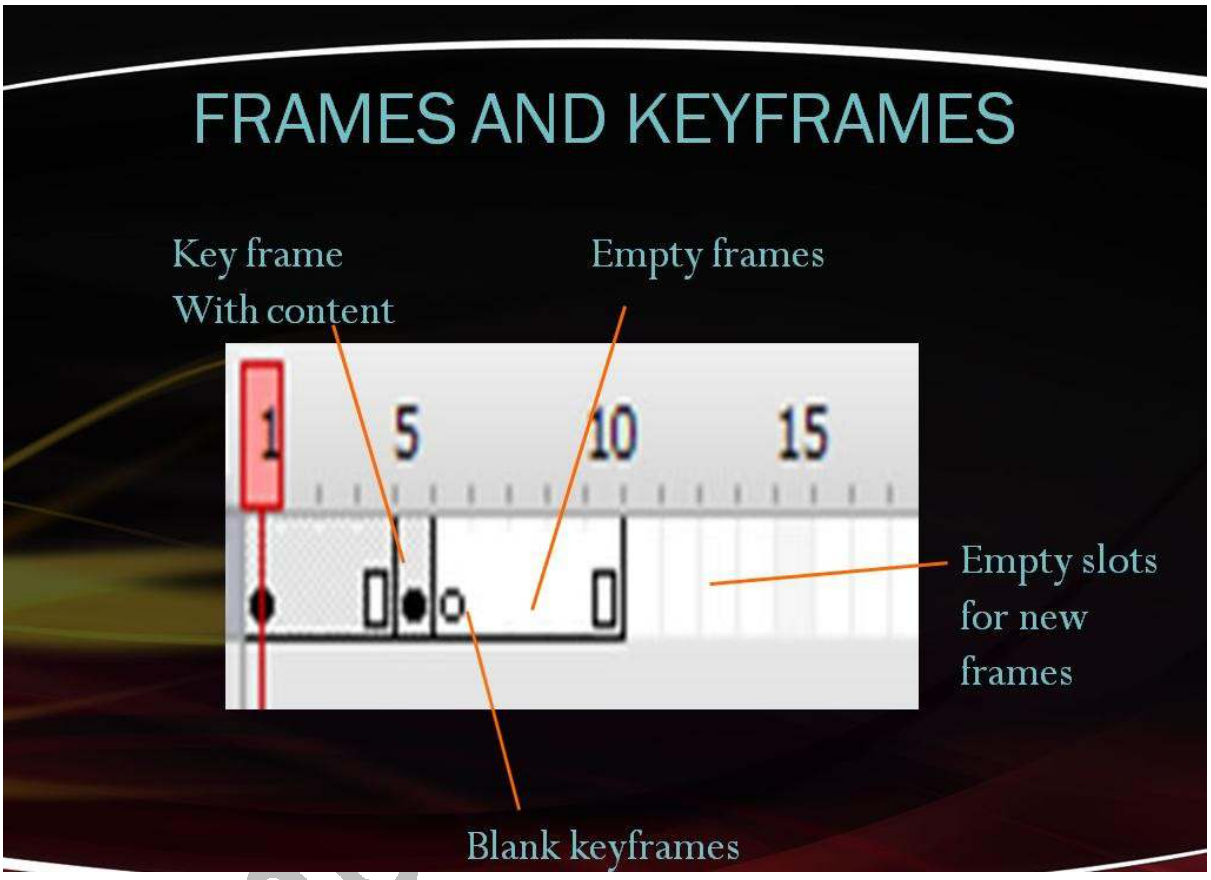
You can organize and control the content of a movie over timeline.



FRAMES AND KEYFRAMES:

Frames: Like films, Flash movies divide lengths of time into frames, which are organized on the Timeline.

Keyframes: Frames that define a change in what is displayed in a movie or include frame actions to modify a movie. When you open a new blank movie document, it contains one layer with one blank keyframe**.**



Layers:

*****Layers are like multiple film strips stacked on top of each other, each with a different element that appears on the Stage.

*Graphics

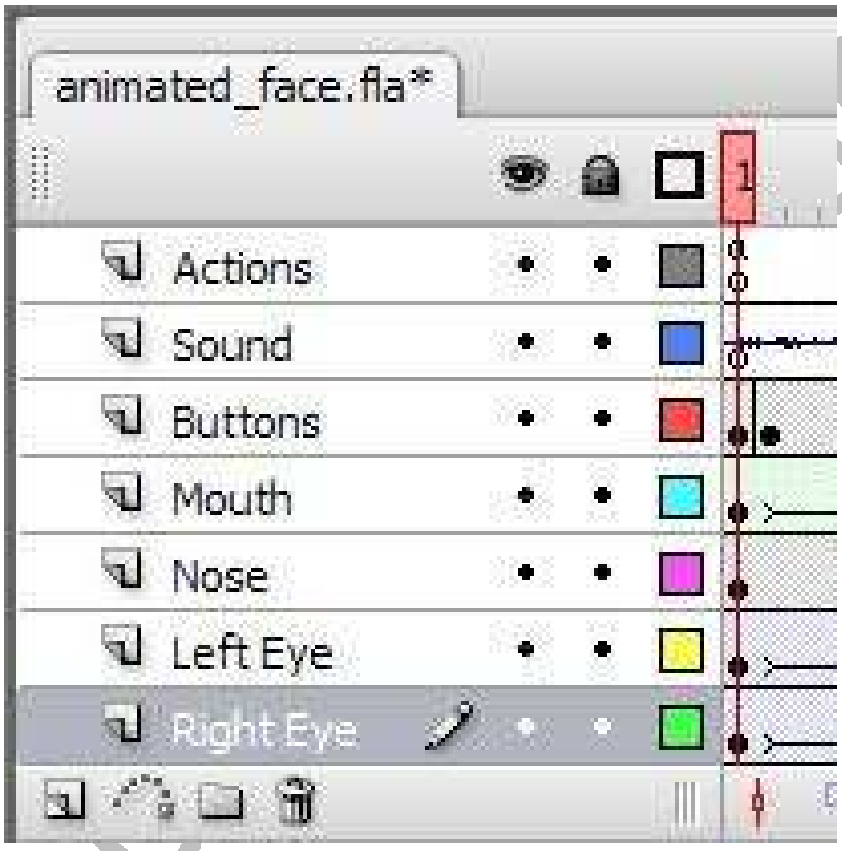
*Animations

*Text

*Sounds

*Buttons

*Frame actions



SYMBOLS AND LIBRARIES:

*Symbols are elements you reuse within a movie to reduce file size.

*Types of symbols include graphics, buttons, movie clips, sound files, and text.

*A library is where you store and organize symbols.

*When you drag a symbol from a library to the Stage, you create an instance of the symbol.

ADVANTAGES OF USING SYMBOLS:

**Easy editing*: If you change the symbol in the library, all instances of the symbol are updated automatically.

**Smaller file sizes*: Symbols are downloaded only once, regardless of the number of instances you've included in the movie. This reduces the size of your published movies and decreases download times

Animation with tweening:

***Tweening:** A series of frames that change incrementally to create smooth movement or change over time.

*You can set the beginning and ending frames and have Flash automatically create the frames in between.

*Flash has two types of tweening:

shape tweening and

Motion tweening.

Shape and motion tweening:

In Flash, a shape is a vector-based object. You create a shape by using the drawing tools or by importing a vector drawing from another program.

Use **shape tweening** to animate one shape into another. You cannot shape-tween grouped objects, bitmaps, text that has not been broken apart, or symbols.

Use **motion tweening** to animate symbols, groups, and text blocks.

ACTION SCRIPT:

ActionScript statements instruct a movie to do something while it is playing. ActionScript can be attached to a frame or to an object:

Frames: ActionScript attached to a frame is triggered when the movie plays that frame.

Objects: ActionScript attached to an object is triggered when the viewer interacts with the object, such as moving the pointer over a hotspot or clicking a button.

Procedure for creating a shape tween

The following steps show how to create a shape tween from frame 1 to frame 30 of the Timeline. However, you can create tweens in any part of the Timeline that you choose.

1. In frame 1, draw a square with the Rectangle tool.
2. Select frame 30 of the same layer and add a blank keyframe by choosing Insert > Timeline > Blank Keyframe or pressing F7.
3. On the Stage, draw a circle with the Oval tool in frame 30.
   You should now have a keyframe in frame 1 with a square and a keyframe in frame 30 with a circle.
4. In the Timeline, select one of the frames in between the two keyframes in the layer containing the two shapes.
5. Choose Insert > Shape Tween.
   Flash interpolates the shapes in all the frames between the two keyframes.
6. To preview the tween, scrub the playhead across the frames in the Timeline, or press the Enter key.
7. To tween motion in addition to shape, move the shape in frame 30 to a location on the Stage that is different from the location of the shape in frame 1.
   Preview the animation by pressing the Enter key.
8. To tween the color of the shape, make the shape in frame 1 a different color from the shape in frame 30.
9. To add easing to the tween, select one of the frames between the two keyframes and enter a value in the Ease field in the Property inspector.
   Enter a negative value to ease the beginning of the tween. Enter a positive value to ease the end of the tween

Procedure for creating motion tween:

1. Create a new Flash document and name it something like Motion Tween Bounce.
   Flash creates a document with one layer and a keyframe at Frame 1 by default.
2. In the Timeline, select Frame 1.
3. In the Toolbar, select the oval tool; set the Line Color to None.
4. Near the top of the Stage, draw a circle.
   This circle will be the ball. Make it fairly large.
5. With Frame 1 still selected, from the Insert menu, choose Create Motion Tween (Choose Insert > Create Motion Tween.
   Flash creates a symbol from the objects on the Stage. Flash gives the symbol a default name based on the number of tweening objects already created in the movie. In this case, Flash turns the ball into a symbol named Tween 1
   The Create Motion Tween command turns an editable object on the Stage in the selected frame into a symbol and names the symbol Tween 1, Tween 2, and so on.
6. In the Timeline, select Frame 5.
7. Choose Insert > Frame.
   Flash adds frames containing a dotted line. The dotted line indicates that these frames are set to contain a motion tween but something is wrong and Flash cannot complete the tween. In this case, the keyframe that describes where the ball should be at the end of this animation sequence is missing.
   Adding frames to the motion tween results in a temporarily broken tween (this dashed line in the Timeline indicates this).
8. In Frame 5, move the circle to the bottom of the Stage to create the downward bounce of the ball.
   Flash creates a keyframe in Frame 5 with the symbol located at the bottom of the Stage. Flash then updates the Timeline to give you information about the tween. In the in-between frames that contain the motion tween, Flash replaces the dotted line with an arrow, indicating that tweening takes place in these frames. These in-between frames are still empty, but they no longer display the content of the previous keyframe—they display the incrementally changed content Flash creates.
   Once you have created a motion tween over a range of frames, repositioning the content of a frame causes Flash to create a new keyframe in the current frame and complete the tween.
9. In the Timeline, select Frame 10.
10. Choose Insert > Frame.
    Flash extends the motion-tween tinting to Frame 10. A dotted line indicating an incomplete tween appears in frames 6 through 10.
11. In Frame 10, move the circle to the top of the Stage to create the upward bounce of the ball.
    Flash creates a new keyframe to contain the changed content and puts the tweening arrow over the in-between frames.
    Adding frames to the end of a motion tween extends the tween. Repositioning the ball in the last frame of the tween completes the tween. Flash creates a new keyframe for the repositioned ball.
12. From the Control menu, choose Play to preview the animation.
    You've created another version of the simple bouncing ball. As in the frame-by-frame animation you created in Chapter 8, you created new content for just three frames, yet this tweened animation is much smoother than the three-keyframe animation you created with the frame-by-frame technique. That's because you've actually created a ten-frame animation; you're just letting Flash do the work of repositioning the ball on the in-between frames.

Procedure for using bone tool in adobe flash:

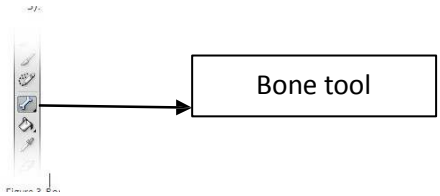
1. Create a new Flash document and make sure to select ActionScript 3.0. The Bone tool will only work with AS 3.0 documents

2. Draw an object on the Stage. For this example, I kept it simple and used the Rectangle tool to create a basic shape.

3. Once you are done creating your shape, convert it to a Movie Clip or a Graphic symbol.

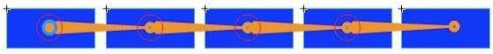
4. Since you'll need more than one object to create a chain of linked objects, duplicate the symbol by holding down the Alt (Windows) or Option (Mac OS) key and dragging the symbol to a new location. Flash will duplicate the instance every time you click and drag it. Repeat this procedure a few more times to create multiple instances of the same symbol

5. Link all of these objects together to create your armature. Select the Bone tool (X) from the Tools panel

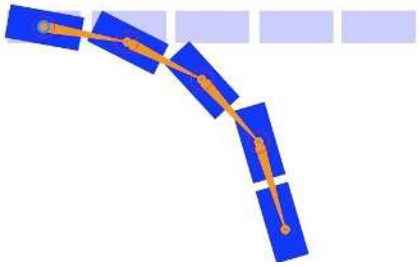


6. Decide what will be your parent or root symbol instance in the armature. This is the symbol instance to which you will apply the first bone segment. Then drag to the next symbol instance to link them together. When you release the mouse, a solid bone segment will appear between the symbol instances

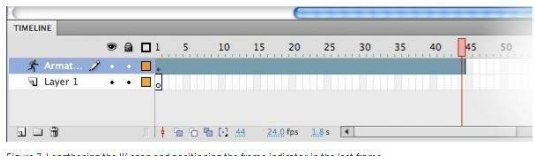
7. Repeat this procedure to link the second symbol instance to the third instance. Continue dragging from one symbol to the next until you have linked all symbol instances with bones



8. The next step is the fun part. Select the Selection tool from the Tools panel (V) and drag the last bone in your chain. The entire armature can now be manipulated in real time as you drag the last bone around the Stage



9. It's easy to animate your armature by increasing the amount of frames of the IK span by clicking and dragging its edge to the desired frame number. With the frame indicator on a new frame number, drag your armature to a new position. Flash will insert a keyframe in the current frame number and interpolate the motion within the IK span.



Define Computer graphics.
Computer graphics may be defined as a pictorial representation or graphical representation of objects in a computer.

Define Random scan/Raster scan displays?
Random scan is a method in which the display is made by the electronic beam which is directed only to the points or part of the screen where the picture is to be drawn. Picture is drawn as series of lines.
The Raster scan system is a scanning technique in which the electrons sweep from top to bottom and from left to right. The intensity is turned on or off to light and unlight the pixel.

List out the merits and demerits of Penetration techniques?
The merits and demerits of the Penetration techniques are as follows
- It is an inexpensive technique
- It has only four colors
- The quality of the picture is not good when it is compared to other techniques
- It can display color scans in monitors
- Poor limitation etc.

Define pixel.
Pixel is shortened forms of picture element. Each screen point is referred to as pixel or pel.

What is frame buffer?
Picture definition is stored in a memory area called frame buffer or refresh buffer.

DDA algorithm?
Digital differential analyzer is a scan conversion line algorithm based on calculating either dx or dy at unit intervals in one coordinate and determine corresponding integer values nearest to the line path for the other coordinate.

Merits and de-merits of DDA:

Merit:
1. It is a faster method for calculating pixel positions than direct use.
2. It eliminates multiplication by making use of raster characteristics.

De-merits:
1. It is time consuming
2. It is difficult to perform rounding and arithmetic operations.

What is Need for Bresenhams algorithm?
Since this algorithm uses only incremental integer calculations we need to decide which of the two possible pixel positions is closer to the line path at each sample step.

What is Decision parameter and its need?

Decision parameter is an integer parameter whose value is proportional to the difference between the separations of two pixel positions from the actual line path. It is used to decide which of the two possible pixel positions is closer to the line path at each sample step.

Properties of circle

1. Circle type
2. Width
3. Color
4. dot-dash patterns
5. Pen or brush options

Attribute primitive

The parameter that affects the way a primitive is to be displayed is referred to as attribute parameters. Eg. Color, size that determine fundamental characteristics of a primitive.

Attributes of a line

1. Line type
2. Line width
3. Pen and brush options
4. Line color

What is PHIGS?

PHIGS is Programmers hierarchical interactive graphics standard. It is the second software standard to be developed and approved by standard organization. It is an extension of graphical kernel system (GKS). It increases capabilities for object modeling, color specifications, surface rendering and picture manipulations.

Give some examples for computer graphics standards.

i. CORE – The Core graphics standard
j. GKS -- The Graphics Kernel system
k. PHIGS – The Programmers Hierarchical Interactive Graphics System.
l. GSX – The Graphics system extension
m. NAPLPS – The North American presentation level protocol syntax.

What are the different types of line type attributes?

solid lines
dotted lines
dashed lines

What is pixel mask?

It is a string containing the digits 1 and 0, to indicate which positions to plot along the line path.

How will you specify the line width for lines with |m| < 1?

For lines with slope magnitude less than 1, we can modify a line drawing routine to display thick lines by plotting a vertical span of pixels at each x position along the line. The no. of pixels in each span is set equal to the integer magnitude of parameter lw.

How will you specify the line width for lines with |m| >1?
For lines with slope magnitude greater than 1, we can modify a thick line with horizontal spans alternatively picking up pixels to the right and left of the line path.

What is a Look-up table?
It is a scheme for storing color values in a color look-up table where frame buffer values are now used as indices into the color table. This scheme would allow the user to select any 256 colors for simultaneous display from a palette of nearly 17 million colors.

What are the area-fill attributes?
1. Fill styles
2. Pattern fill
3. Soft fill

How areas are displayed with 3 basic fill styles?
1. Hollow with a color border- using only the boundary outline
2. Filled with solid color- displayed in a single color including the borders
3. Filled with specified pattern or design

Give types of algorithms for filling object with colors?
Boundary fill Algorithm
Flood Fill Algorithm
Scanline polygon fill Algorithm

What is Transformation?
Transformation is the process of introducing changes in the shape size and orientation of the object using scaling rotation reflection shearing & translation etc.

What is translation?
Translation is the process of changing the position of an object in a straight-line path from one coordinate location to another.

What is rotation?
A 2-D rotation is done by repositioning the coordinates along a circular path, in the x-y plane by making an angle with the axes.

What is scaling?
The scaling transformations changes the shape of an object and can be carried out by multiplying each vertex (x,y) by scaling factor Sx,Sy where Sx is the scaling factor of x and Sy is the scaling factor of y.

What is shearing?
The shearing transformation actually slants the object along the X direction or the Y direction as required.ie; this transformation slants the shape of an object along a required plane.

What is reflection?
The reflection is actually the transformation that produces a mirror image of an object. For this use some angles and lines of reflection.

What is the need of homogeneous coordinates?

To perform more than one transformation at a time, use homogeneous coordinates or matrixes. They reduce unwanted calculations intermediate steps saves time and memory and produce a sequence of transformations.

Distinguish between uniform scaling and differential scaling.

When the scaling factors sx and sy are assigned to the same value, a uniform scaling is produced that maintains relative object proportions. Unequal values for sx and sy result in a differential scaling that is often used in design application.

What is fixed point scaling?

The location of a scaled object can be controlled by a position called the fixed point that is to remain unchanged after the scaling transformation.

Distinguish between window port & view port.

A portion of a picture that is to be displayed by a window is known as window port. The display area of the part selected or the form in which the selected part is viewed is known as view port.

Define clipping.

Clipping is the method of cutting a graphics display to neatly fit a predefined graphics region or the view port.

What is covering (exterior clipping)?

This is just opposite to clipping. This removes the lines coming inside the windows and displays the remaining. Covering is mainly used to make labels on the complex pictures.

List out the various Text clipping?

_ All-or-none string clipping - if all of the string is inside a clip window, keep it otherwise discards.
_ All-or-none character clipping – discard only those characters that are not completely inside the window. Any character that either overlaps or is outside a window boundary is clipped.
_ Individual characters – if an individual character overlaps a clip window boundary, clip off the parts of the character that are outside the window.

What are the various representation schemes used in three dimensional objects?

Boundary representation (B-res) – describe the 3 dimensional object as a set of surfaces that separate the object interior from the environment.
Space- portioning representation – describe interior properties, by partitioning the spatial region containing an object into a set of small, no overlapping, contiguous solids.

What is Polygon mesh?

Polygon mesh is a method to represent the polygon, when the object surfaces are tiled, it is more convenient to specify the surface facets with a mesh function. The various meshes are Triangle strip – (n-2) connected triangles Quadrilateral mesh – generates (n-1)(m-1) Quadrilateral

What are the advantages of rendering polygons by scan line method?

i. The max and min values of the scan were easily found.
ii. The intersection of scan lines with edges is easily calculated by a simple incremental method.
iii. The depth of the polygon at each pixel is easily calculated by an incremental method.

What is a Blobby object?

Some objects do not maintain a fixed shape, but change their surface characteristics in certain motions or when in proximity to other objects. That is known as blobby objects. Example – molecular structures, water droplets.

Define Octrees.

Hierarchical tree structures called octrees, are used to represent solid objects in some graphics systems. Medical imaging and other applications that require displays of object cross sections commonly use octree representation.

Define B-Spline curve.

A B-Spline curve is a set of piecewise (usually cubic) polynomial segments that pass close to a set of control points. However the curve does not pass through these control points, it only passes close to them.

What is a Blobby object?

Some objects do not maintain a fixed shape, but change their surface Characteristics in certain motions or when in proximity to other objects. That is known as blobby objects. Example – molecular structures, water droplets.

What are the steps involved in 3D transformation pipeline?

- Modeling Transformation
- Viewing Transformation
- Projection Transformation
- Workstation Transformation

What do you mean by view plane?

A view plane is nothing but the film plane in camera which is positioned and oriented for a particular shot of the scene.

Define projection.

The process of converting the description of objects from world coordinates to viewing coordinates is known as projection

What you mean by parallel projection?

Parallel projection is one in which z coordinates is discarded and parallel lines from each vertex on the object are extended until they intersect the view plane.

What do you mean by Perspective projection?

Perspective projection is one in which the lines of projection are not parallel. Instead, they all converge at a single point called the center of projection.

What is Projection reference point?

In Perspective projection, the lines of projection are not parallel. Instead, they all converge at a single point called Projection reference point.

What is the use of Projection reference point?

In Perspective projection, the object positions are transformed to the view plane along these converged projection line and the projected view of an object is determined by calculating the intersection of the converged projection lines with the view plane.

What are the different types of parallel projections?

The parallel projections are basically categorized into two types, depending on the relation between the direction of projection and the normal to the view plane. They are orthographic parallel projection and oblique projection.

What is orthographic parallel projection?

When the direction of the projection is normal (perpendicular) to the view plane then the projection is known as orthographic parallel projection

what is orthographic oblique projection?

When the direction of the projection is not normal (not perpendicular) to the view plane then the projection is known as oblique projection.

What is an axonometric orthographic projection?

The orthographic projection can display more than one face of an object. Such an orthographic projection is called axonometric orthographic projection.

What is cavalier projection?

The cavalier projection is one type of oblique projection, in which the direction of projection makes a 45-degree angle with the view plane.

What is vanishing point?

The perspective projections of any set of parallel lines that are not parallel to the projection plane converge to appoint known as vanishing point.

What do you mean by principle vanishing point?

The vanishing point of any set of lines that are parallel to one of the three principle axes of an object is referred to as a principle vanishing point or axis vanishing point.

What is view reference point?

The view reference point is the center of the viewing coordinate system. It is often chosen to be close to or on the surface of the some object in the scene.

What is the use of control points?

Spline curve can be specified by giving a set of coordinate positions called control points, which indicates the general shape of the curve, can specify spline curve.

What is Bezier Basis Function?

Bezier Basis functions are a set of polynomials, which can be used instead of the primitive polynomial basis, and have some useful properties for interactive curve design.

What is surface patch?

A single surface element can be defined as the surface traced out as two parameters (u, v) take all possible values between 0 and 1 in a two-parameter representation. Such a single surface element is known as a surface patch.

Define B-Spline curve.

A B-Spline curve is a set of piecewise (usually cubic) polynomial segments that pass close to a set of control points. However the curve does not pass through these control points, it only passes close to them.

What is a spline?

To produce a smooth curve through a designed set of points, a flexible strip called spline is used. Such a spline curve can be mathematically described with a piecewise cubic polynomial function whose first and second derivatives are continuous across various curve section.

What are the different ways of specifying spline curve?

• Using a set of boundary conditions that are imposed on the spline.
• Using the state matrix that characteristics the spline
• Using a set of blending functions that calculate the positions along the curve path by specifying combination of geometric constraints on the curve

What are the important properties of Bezier Curve?

• It needs only four control points
• It always passes through the first and last control points
• The curve lies entirely within the convex half formed by four control points.

Differentiate between interpolation spline and approximation spline.

When the spline curve passes through all the control points then it is called interpolate. When the curve is not passing through all the control points then that curve is called approximation spline.

Define computer graphics animation.

Computer graphics animation is the use of computer graphics equipment where the graphics output presentation dynamically changes in real time. This is often also called real time animation.Visula changes of scene over screen with respect to time.

Define frame.

One of the shape photographs that a film or video is made of is known as frame.

What is key frame?

One of the shape photographs that a film or video is made of the shape of an object is known initially and for a small no of other frames called keyframe

Give the steps for designing an animation sequence?

The steps are:

1. Story board layout
2. Object parameters
3. Key frame generation
4. In-between generation

Give computer animation languages?

C, LISP, PASCAL, FORTRAN, FLASH, MAYA etc

MUST DO PRACTICE EXERCISE PROGRAMS

1. Design a ROBO using set of line segments generated by DDA Algorithm?
2. Design our national flag using set of lines generated by DDA or Bresenhams line drawing algorithm?
3. Design a solar planet system using a set of circles generated by midpoint circle generation algorithm?
4. Design a sky consisting of clouds using set of ellipses or circles generated by mid point ellipse generation algorithm?
5. Draw 3 clouds by using graphics functions and fill the 3 clouds with saphron, white and green colors by using flood fill Algorithm?
6. Draw our national flag by using graphics functions and fill appropriate colors by using boundary fill algorithm?
7. Draw a Mickey Mouse shape by using DDA, circle generation algorithm?
8. Design any cartoon animation which is sequence of at least 10-15 frames using adobe flash professional?
9. Using shape tweening and motion tweening, generate any animation sequence?
10. Using bone tool in adobe flash professional, generate an animation sequence of cartoon character walking?

Graphics.h Wikipedia

FUNCTIONS OF GRAPHICS.H

C graphics using graphics.h functions can be used to draw different shapes, display text in different fonts, change colors and many more. Using functions of graphics.h in turbo c compiler you can make graphics programs, animations, projects and games. You can draw circles, lines, rectangles, bars and many other geometrical figures. You can change their colors using the available functions and fill them. Following is a list of functions of graphics.h header file. Every function is discussed with the arguments it needs, its description, possible errors while using that function and a sample c graphics program with its output.

arc	circle	closegraph
bar	cleardevice	detectgraph
bar3d	clearviewport	drawpoly
ellipse	floodfill	getbkcolor
fillellipse	getarccoords	getcolor
fillpoly	getaspectratio	getdefaultpalette
getdrivername	getgraphmode	getmaxcolor
getfillpattern	getimage	getmaxmode
getfillsettings	getlinesettings	getmaxx
getmaxy	getpalette	gettextsettings
getmodename	getpalettesize	getviewsettings
getmoderange	getpixel	getx
gety	graphfreemem	imagesize
graphdefaults	graphgetmem	initgraph
grapherrormsg	graphresult	installuserdriver
installuserfont	lineto	outtext
line	moverel	outtextxy
linerel	moveto	pieslice
putimage	registerbgidriver	registerfarbgifont
putpixel	registerfarbgidriver	restorecrtmode
rectangle	registerbgifont	sector
setactivepage	setbkcolor	setfillstyle
setallpalette	setcolor	setgraphbufsize
setaspectratio	setfillpattern	setgraphmode
setlinestyle	settextjustify	setviewport
setpalette	settextstyle	setvisualpage
setrgbpalette	setusercharsize	setwritemode
textheight		
textwidth		

1) arc

arc function in c

Declaration :- void arc(int x, int y, int stangle, int endangle, int radius);
arc function is used to draw an arc with center (x,y) and stangle specifies starting angle, endangle specifies the end angle and last parameter specifies the radius of the arc. arc function can also be used to draw a circle but for that starting angle and end angle should be 0 and 360 respectively.

C programming source code for arc

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

main()
{
  int gd = DETECT, gm;

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

  arc(100, 100, 0, 135, 50);

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

2) bar

bar function in c

Declaration :- void bar(int left, int top, int right, int bottom);

Bar function is used to draw a 2-dimensional, rectangular filled in bar . Coordinates of left top and right bottom corner are required to draw the bar. Left specifies the X-coordinate of top left corner, top specifies the Y-coordinate of top left corner, right specifies the X-coordinate of right bottom corner, bottom specifies the Y-coordinate of right bottom corner. Current fill pattern and fill color is used to fill the bar. To change fill pattern and fill color use setfillstyle.

C programming code

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

main()
{
  int gd = DETECT, gm;

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

  bar(100, 100, 200, 200);

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

- 3) bar3d

bar3d function in c

Declaration :- void bar3d(int left, int top, int right, int bottom, int depth, int topflag);

bar3d function is used to draw a 2-dimensional, rectangular filled in bar . Coordinates of left top and right bottom corner of bar are required to draw the bar. left specifies the X-coordinate of top left corner, top specifies the Y-coordinate of top left corner, right specifies the X-coordinate of right bottom corner, bottom specifies the Y-coordinate of right bottom corner, depth specifies the depth of bar in pixels, topflag determines whether a 3 dimensional top is put on the bar or not (if it is non-zero then it is put otherwise not). Current fill pattern and fill color is used to fill the bar. To change fill pattern and fill color use setfillstyle.

C program of bar3d

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

main()
{
  int gd = DETECT, gm;

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

  bar3d(100, 100, 200, 200, 20, 1);

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

•4) circle

circle function in c

Declaration :- void circle(int x, int y, int radius);

circle function is used to draw a circle with center (x,y) and third parameter specifies the radius of the circle. The code given below draws a circle.

C program for circle

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

main()
{
  int gd = DETECT, gm;

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

  circle(100, 100, 50);

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

- 5) cleardevice

cleardevice function in c

Declaration :- void cleardevice();
cleardevice function clears the screen in graphics mode and sets the current position to (0,0). Clearing the screen consists of filling the screen with current background color.

C program for cleardevice

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

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

  outtext("Press any key to clear the screen.");
  getch();
  cleardevice();
  outtext("Press any key to exit...");

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

- 6) closegraph

closegraph function in c

closegraph function closes the graphics mode, deallocates all memory allocated by graphics system and restores the screen to the mode it was in before you called initgraph.

Declaration :- void closegraph();

C code of closegraph

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

main()
{
  int gd = DETECT, gm;

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

  outtext("Press any key to close the graphics mode...");

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

- 7) drawpoly

drawpoly function in c

Drawpoly function is used to draw polygons i.e. triangle, rectangle, pentagon, hexagon etc.

Declaration :- void drawpoly(int num, int *polypoints);

num indicates (n+1) number of points where n is the number of vertices in a polygon, polypoints points to a sequence of (n*2) integers . Each pair of integers gives x and y coordinates of a point on the polygon. We specify (n+1) points as first point coordinates should be equal to $(n+1)^{th}$ to draw a complete figure.

To understand more clearly we will draw a triangle using drawpoly, consider for example the array :-
int points[] = { 320, 150, 420, 300, 250, 300, 320, 150};

points array contains coordinates of triangle which are (320, 150), (420, 300) and (250, 300). Note that last point(320, 150) in array is same as first. See the program below and then its output, it will further clear your understanding.

C program for drawpoly

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

main()
{
  int gd=DETECT,gm,points[]={320,150,420,300,250,300,320,150};

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

  drawpoly(4, points);

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

- 8) ellipse

ellipse function in c

Declarations of ellipse function :-
void ellipse(int x, int y, int stangle, int endangle, int xradius, int yradius);

Ellipse is used to draw an ellipse (x,y) are coordinates of center of the ellipse, stangle is the starting angle, end angle is the ending angle, and fifth and sixth parameters specifies the X and Y radius of the ellipse. To draw a complete ellipse strangles and end angle should be 0 and 360 respectively.

C programming code for ellipse

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

main()
{
  int gd = DETECT, gm;

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

  ellipse(100, 100, 0, 360, 50, 25);

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

- 9) fillellipse

fillellipse function in c

Declaration of fillellipse function :-
void fillellipse(int x, int y, int xradius, int yradius);
x and y are coordinates of center of the ellipse, xradius and yradius are x and y radius of ellipse respectively.

C program for fillellipse

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

main()
{
  int gd = DETECT, gm;

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

  fillellipse(100, 100, 50, 25);

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

- 10) fillpoly

fillpoly function in c

Fillpoly function draws and fills a polygon. It require same arguments as drawpoly.

Declaration :- void drawpoly(int num, int *polypoints);
For details of arguments see drawpoly.
fillpoly fills using current fill pattern and color which can be changed using setfillstyle.

C programming code

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

main()
{
  int gd=DETECT,gm,points[]={320,150,440,340,230,340,320,150};

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

  fillpoly(4, points);

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

- 11) floodfill

floodfill function

Declaration :- void floodfill(int x, int y, int border);

floodfill function is used to fill an enclosed area. Current fill pattern and fill color is used to fill the area.(x, y) is any point on the screen if (x,y) lies inside the area then inside will be filled otherwise outside will be filled,border specifies the color of boundary of area. To change fill pattern and fill color use setfillstyle. Code given below draws a circle and then fills it.

C programming code

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

main()
{
  int gd = DETECT, gm;

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

  setcolor(RED);
  circle(100,100,50);
  floodfill(100,100,RED);

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

•12) getarccords

getarcoords function in c

Declaration :- void getarccoords(struct arccoordstype *var);

getarccoords function is used to get coordinates of arc which is drawn most recently. arccoordstype is a predefined structure which is defined as follows:

```
struct arccoordstype
{
  int x, y;             /*  center point of arc    */
  int xstart, ystart;   /*  start position         */
  int xend, yend;       /*  end position           */
};
```

address of a structure variable of type arccoordstype is passed to function getarccoords.

C program of getarccoords

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

main()
{
  int gd = DETECT, gm;
  struct arccoordstype a;
  char arr[100];

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

  arc(250,200,0,90,100);
  getarccoords(&a);

  sprintf(arr,"(%d, %d)",a.xstart,a.ystart);
  outtextxy(360,195,arr);

  sprintf(arr,"(%d, %d)",a.xend,a.yend);
  outtextxy(245,85,arr);

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

- 13) getbkcolor

getbkcolor function in c

getbkcolor function returns the current background color

Declaration : int getbkcolor();

e.g. color = getbkcolor(); // color is an int variable
if current background color is GREEN then color will be 2.

C program for getbkcolor

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

main()
{
  int gd = DETECT, gm, bkcolor;
  char a[100];

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

  bkcolor = getbkcolor();

  sprintf(a,"Current background color = %d", bkcolor);
  outtextxy( 10, 10, a);

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

• 14) getcolor

getcolor function

getcolor function returns the current drawing color.

Declaration : int getcolor();

e.g. a = getcolor(); // a is an integer variable
if current drawing color is WHITE then a will be 15.

C programming code for getcolor

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

main()
{
  int gd = DETECT, gm, drawing_color;
  char a[100];

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

  drawing_color = getcolor();

  sprintf(a,"Current drawing color = %d", drawing_color);
  outtextxy( 10, 10, a );

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

- 15) getdrivername

getdrivername function

getdrivername function returns a pointer to the current graphics driver.

C program for getdrivername

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

main()
{
  int gd = DETECT, gm;

  char *drivername;

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

  drivername = getdrivername();
  outtextxy(200, 200, drivername);

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

•16) getimage

getimage function in c

getimage function saves a bit image of specified region into memory, region can be any rectangle.

Declaration:- void getimage(int left, int top, int right, int bottom, void *bitmap);

getimage copies an image from screen to memory. Left, top, right, and bottom define the area of the screen from which the rectangle is to be copied, bitmap points to the area in memory where the bit image is stored.

17) getmaxcolor

getmaxcolor function

getmaxcolor function returns maximum color value for current graphics mode and driver. Total number of colors available for current graphics mode and driver are (getmaxcolor() + 1) as color numbering starts from zero.

Declaration :- int getmaxcolor();

C program of getmaxcolor

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

main()
{
  int gd = DETECT, gm, max_colors;
  char a[100];

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

  max_colors = getmaxcolor();

  sprintf(a,"Maximum number of colors for current graphics mode and driver =
%d",max_colors+1);
  outtextxy(0, 40, a);

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

•18) getmaxx

getmaxx function in c

getmaxx function returns the maximum X coordinate for current graphics mode and driver.

Declaration :- int getmaxx();

C program for getmaxx

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

main()
{
  int gd = DETECT, gm, max_x;
  char array[100];

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

  max_x = getmaxx();

  sprintf(array, "Maximum X coordinate for current graphics mode and driver =
%d.",max_x);
  outtext(array);

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

- 19)getmaxy

getmaxy function in c

getmaxy function returns the maximum Y coordinate for current graphics mode and driver.

Declaration :- int getmaxy();

C program for getmaxy

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

main()
{
  int gd = DETECT, gm, max_y;
  char array[100];

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

  max_y = getmaxy();

  sprintf(array, "Maximum Y coordinate for current graphics mode and driver is =
%d.",max_y);
  outtext(array);

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

•20) getpixel

getpixel function in c

getpixel function returns the color of pixel present at location(x, y).

Declaration :- int getpixel(int x, int y);

C program for getpixel

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

main()
{
 int gd = DETECT, gm, color;
 char array[50];

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

 color = getpixel(0, 0);

 sprintf(array,"color of pixel at (0,0) = %d",color);
 outtext(array);

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

•21) getx

getx function in c

getx function returns the X coordinate of current position.

Declaration :- int getx();

C program of getx

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

main()
{
  int gd = DETECT, gm;
  char array[100];

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

  sprintf(array, "Current position of x = %d",getx());

  outtext(array);

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

•22) gety

gety function in c

gety function returns the y coordinate of current position.

Declaration :- int gety();

C programming source code for gety

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

main()
{
  int gd = DETECT, gm, y;
  char array[100];

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

  y = gety();

  sprintf(array, "Current position of y = %d", y);

  outtext(array);

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

•23) graphdefaults

graphdefaults function in c

graphdefaults function resets all graphics settings to their defaults.

Declaration :- void graphdefaults();

It resets the following graphics settings :-

Sets the viewport to the entire screen.
Moves the current position to (0,0).
Sets the default palette colors, background color, and drawing color.
Sets the default fill style and pattern.
Sets the default text font and justification.

C programming source code for graphdefaults

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

main()
{
  int gd = DETECT, gm;

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

  setcolor(RED);
  setbkcolor(YELLOW);

  circle(250, 250, 50);

  getch();
  graphdefaults();

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

- 24)grapherrormsg

grapherrormsg function in c

grapherrormsg function returns an error message string.

Declaration :- char *grapherrormsg(int errorcode);

C programming code for grapherrormsg

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

main()
{
  int gd, gm, errorcode;

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

  errorcode = graphresult();

  if(errorcode != grOk)
  {
    printf("Graphics error: %s\n", grapherrormsg(errorcode));
    printf("Press any key to exit.");
    getch();
    exit(1);
  }

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

- 25) imagesize

imagesize function in c

imagesize function returns the number of bytes required to store a bitimage. This function is used when we are using getimage.

Declaration:- unsigned int imagesize(int left, int top, int right, int bottom);

C programming code for imagesize

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

main()
{
  int gd = DETECT, gm, bytes;
  char array[100];

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

  circle(200, 200, 50);
  line(150, 200, 250, 200);
  line(200, 150, 200, 250);

  bytes = imagesize(150, 150, 250, 250);
  sprintf(array, "Number of bytes required to store required area = %d", bytes);
  outtextxy(10, 280, array);

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

•26) line

line function in c

line function is used to draw a line from a point(x1,y1) to point(x2,y2) i.e. (x1,y1) and (x2,y2) are end points of the line.The code given below draws a line.

Declaration :- void line(int x1, int y1, int x2, int y2);

C programming code for line

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

main()
{
  int gd = DETECT, gm;

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

  line(100, 100, 200, 200);

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

•27) lineto

lineto function in c

lineto function draws a line from current position(CP) to the point(x,y), you can get current position using getx and gety function.

C programming code for lineto

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

main()
{
  int gd = DETECT, gm;

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

  moveto(100, 100);
  lineto(200, 200);

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

- 28) linerel

linerel function in c

Linerel function draws a line from the current position(CP) to a point that is a relative distance (x, y) from the CP, then advances the CP by (x, y). You can use getx and gety to find the current position.

C programming code for linerel

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

main()
{
  int gd = DETECT, gm;

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

  moveto(250, 250);
  linerel(100, -100);

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

- 29) moveto

moveto function in c

moveto function changes the current position (CP) to (x, y)

Declaration :- void moveto(int x, int y);

C programming code for moveto

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

main()
{
  int gd = DETECT, gm;
  char msg[100];

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

  sprintf(msg, "X = %d, Y = %d",getx(),gety());

  outtext(msg);

  moveto(50, 50);

  sprintf(msg, "X = %d, Y = %d", getx(), gety());

  outtext(msg);

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

•30) moverel

moverel function in c

moverel function moves the current position to a relative distance.

Declaration :- void moverel(int x, int y);

C programming code for moverel

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

main()
{
  int gd = DETECT, gm, x, y;
  char message[100];

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

  moveto(100, 100);
  moverel(100, -100);

  x = getx();
  y = gety();

  sprintf(message, "Current x position = %d and y position = %d", x, y);
  outtextxy(10, 10, message);

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

- 31) outtext

outtext function

outtext function displays text at current position.

Declaration :- void outtext(char *string);

C programming code for outtext

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

main()
{
  int gd = DETECT, gm;

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

  outtext("To display text at a particular position on the screen use outtextxy");

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

•32) outtextxy

outtextxy function in c

outtextxy function display text or string at a specified point(x,y) on the screen.

Declaration :- void outtextxy(int x, int y, char *string);
x, y are coordinates of the point and third argument contains the address of string to be displayed.

C programming code for outtextxy

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

main()
{
  int gd = DETECT, gm;

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

  outtextxy(100, 100, "Outtextxy function");

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

- 33) pieslice

pieslice function in c

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

main()
{
  int gd = DETECT, gm;

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

  pieslice(200,200,0,135,100);

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

- 34) putimage

putimage function in c

putimage function outputs a bit image onto the screen.

Declaration:- void putimage(int left, int top, void *ptr, int op);

putimage puts the bit image previously saved with getimage back onto the screen, with the upper left corner of the image placed at (left, top). ptr points to the area in memory where the source image is stored. The op argument specifies a operator that controls how the color for each destination pixel on screen is computed, based on pixel already on screen and the corresponding source pixel in memory.

•35) putpixel

putpixel function in c

putpixel function plots a pixel at location (x, y) of specified color.

Declaration :- void putpixel(int x, int y, int color);

For example if we want to draw a GREEN color pixel at (35, 45) then we will write putpixel(35, 35, GREEN); in our c program, putpixel function can be used to draw circles, lines and ellipses using various algorithms.

C programming code for putpixel

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

main()
{
  int gd = DETECT, gm;

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

  putpixel(25, 25, RED);

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

- 36) rectangle

rectangle function in c

Declaration :- void rectangle(int left, int top, int right, int bottom);

rectangle function is used to draw a rectangle. Coordinates of left top and right bottom corner are required to draw the rectangle. left specifies the X-coordinate of top left corner, top specifies the Y-coordinate of top left corner, right specifies the X-coordinate of right bottom corner, bottom specifies the Y-coordinate of right bottom corner. The code given below draws a rectangle.

c programming code for rectangle

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

main()
{
  int gd = DETECT, gm;

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

  rectangle(100,100,200,200);

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

-

37) sector

sector function in c

Sector function draws and fills an elliptical pie slice.

Declaration :- void sector(int x, int y, int stangle, int endangle, int xradius, int yradius);

C programming code for sector

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

main()
{
  int gd = DETECT, gm;

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

  sector(100, 100, 0, 135, 25, 35);

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

- 38) setbkcolor

setbkcolor function in c

Declaration :- void setbkcolor(int color);

setbkcolor function changes current background color e.g. setbkcolor(YELLLOW) changes the current background color to YELLOW.
Remember that default drawing color is WHITE and background color is BLACK.

C programming code for setbkcolor

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

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

  outtext("Press any key to change the background color to GREEN.");
  getch();

  setbkcolor(GREEN);

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

•39) setcolor

setcolor function in c

Declaration :- void setcolor(int color);

In Turbo Graphics each color is assigned a number. Total 16 colors are available. Strictly speaking number of available colors depends on current graphics mode and driver.For Example :- BLACK is assigned 0, RED is assigned 4 etc. setcolor function is 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.

C programming code for setcolor

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

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

  circle(100,100,50);        /* drawn in white color */
  setcolor(RED);
  circle(200,200,50);        /* drawn in red color  */

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

- 40) setfillstyle

setfillstyle function in c

setfillstyle function sets the current fill pattern and fill color.

Declaration :- void setfillstyle(int pattern, int color);

Different fill styles:

```
enum fill_styles
{
  EMPTY_FILL,
  SOLID_FILL,
  LINE_FILL,
  LTSLASH_FILL,
  SLASH_FILL,
  BKSLASH_FILL,
  LTBKSLASH_FILL,
  HATCH_FILL,
  XHATCH_FILL,
  INTERLEAVE_FILL,
  WIDE_DOT_FILL,
  CLOSE_DOT_FILL,
  USER_FILL
};
```

C programming source code for setfillstyle

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

main()
{
  int gd = DETECT, gm;

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

  setfillstyle(XHATCH_FILL, RED);
  circle(100, 100, 50);
  floodfill(100, 100, WHITE);

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

•41) setlinestyle

setlinestyle in c

Declaration:
void setlinestyle(int linestyle, unsigned upattern, int thickness);

Available line styles:

```
enum line_styles
{
  SOLID_LINE,
  DOTTED_LINE,
  CENTER_LINE,
  DASHED_LINE,
  USERBIT_LINE
};
```

C programming code

```
#include <graphics.h>

main()
{
  int gd = DETECT, gm, c , x = 100, y = 50;

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

  for ( c = 0 ; c < 5 ; c++ )
  {
     setlinestyle(c, 0, 2);

     line(x, y, x+200, y);
     y = y + 25;
  }

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

•42) settextstyle

settextstyle function in c

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.

Declaration :- void settextstyle(int font, int direction, int charsize);
font argument specifies the font of text, Direction can be HORIZ_DIR (Left to right) or VERT_DIR (Bottom to top).

Different fonts

```
enum font_names
{
  DEFAULT_FONT,
  TRIPLEX_FONT,
  SMALL_FONT,
  SANS_SERIF_FONT,
  GOTHIC_FONT,
  SCRIPT_FONT,
  SIMPLEX_FONT,
  TRIPLEX_SCR_FONT,
  COMPLEX_FONT,
  EUROPEAN_FONT,
  BOLD_FONT
};
```

C programming source code for settextstyle

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

main()
{
  int gd = DETECT, gm, x = 25, y = 25, font = 0;
  initgraph(&gd,&gm,"C:\\ TurboC3\\BGI");

  for ( font = 0 ; font <= 10 ; font++)
  {
    settextstyle(font, HORIZ_DIR, 1);
    outtextxy(x, y, "Text with different fonts");
    y = y + 25;
  }

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

•43) setviewport

setviewport function in c

setviewport function sets the current viewport for graphics output.

Declaration :- void setviewport(int left, int top, int right, int bottom, int clip);

setviewport function is used to restrict drawing to a particular portion on the screen. For example setviewport(100 , 100, 200, 200, 1);
will restrict our drawing activity inside the rectangle(100,100, 200, 200).
left, top, right, bottom are the coordinates of main diagonal of rectangle in which we wish to restrict our drawing. Also note that the point (left, top) becomes the new origin.

C programming source code for setviewport

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

main()
{
  int gd = DETECT, gm, midx, midy;

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

  midx = getmaxx()/2;
  midy = getmaxy()/2;

  setviewport(midx - 50, midy - 50, midx + 50, midy + 50, 1);
  circle(50, 50, 55);

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

•44) textheight

textheight function in c

textheight function returns the height of a string in pixels.

Declaration :- int textheight(char *string);

C programming source code for textheight

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

main()
{
   int gd = DETECT, gm, height;
   char array[100];

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

   height = textheight("C programming");

   sprintf(array,"Textheight = %d",height);
   outtext(array);

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

•45) textwidth

textwidth function in c

textwidth function returns the width of a string in pixels.

Declaration :- int textwidth(char *string);

C programming source code for textwidth

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

main()
{
  int gd = DETECT, gm, width;
  char array[100];

  initgraph(&gd,  &gm,  "C:\\TC\\BG  I");

  width  =  textwidth("C  programming");

  sprintf(array,"Textwidth = %d",width);
  outtext(array);

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