



**GUJARAT TECHNOLOGICAL UNIVERSITY**



**Government Engineering College, Bhavnagar**

**Subject: Computer Graphics**

**B.E. C.E. Semester-5<sup>th</sup>**  
**(Computer Branch)**

**Submitted By:**

**Name: Vankani Arjun BakulBhai**

**Enrollment: 180210107060**

**Prof. Kirit Rathod**  
(Faculty Guide)

**Prof. KARSHAN KANDORIYA**  
(Head of the Department)  
Academic Year (2020-21)

# Index

<b>SR No.</b>	<b>Algorithm (Lab work)</b>	<b>Page No</b>
<b>1</b>	<b>Graphics commands</b>	<b>03</b>
<b>2</b>	<b>DDA Line drawing algorithm</b>	<b>06</b>
<b>3</b>	<b>Bresenham's Line drawing algorithm</b>	<b>09</b>
<b>4</b>	<b>Bresenham's Circle drawing algorithm</b>	<b>14</b>
<b>5</b>	<b>Different types of lines</b>	<b>17</b>
<b>6</b>	<b>2D Transformation operation – (Translation, Rotation and Scaling)</b>	<b>19</b>
<b>7</b>	<b>Line Clipping Algorithm</b>	<b>32</b>
<b>8</b>	<b>Polygon clipping algorithm</b>	<b>42</b>
<b>9</b>	<b>MATLAB commands</b>	<b>49</b>
<b>10</b>	<b>Generate the complement image using MATLAB</b>	<b>51</b>
<b>11</b>	<b>Extra Program on animation and geometry</b>	<b>52</b>

# Computer Graphics

## List of Experiments

### Q-1. To study the various graphics commands in C language.

- In computer graphics relate to 2D,3D transformation and create 2d or 3d model also. Let first we want to understand some basic command relate to it and how graphics library import to code.

#### 1. initgraph()

- `void far initgraph(int far *graphdriver, int far *graph mode, char far *pathtodriver);`

To start the graphics system, you must first call `initgraph`. `initgraph` initializes the graphics system by loading a graphics driver from disk then putting the system into graphics mode. `initgraph` also resets all graphics settings (color, palette, current position, viewport, etc.) to them

defaults, then resets graph result to 0.

Main two content is important to initialize graphics

- 1) Graphdriver:- Integer that specifies the graphics driver to be used.
- 2) Graphmode:- integer that specifies the initial graphics mode (unless \*graphdriver = DETECT).

#### 2. closegraph()

- `void far closegraph(void);`

`Closegraph` deallocates all memory allocated by the graphics system. It then restores the screen to the mode it was in before you called `initgraph`.

#### 3. detectgraph()

- `void far detectgraph(int far *graphdriver, int far *graphmode);`

`Detectgraph` detects your system's graphics adapter and chooses the mode that provides the highest resolution for that adapter. If no graphics hardware is detected, `detectgraph` sets \*graphdriver to `grNotDetected`, and `graphresult` returns `grNotDetected`.

The main reason to call `detectgraph` directly is to override the graphics mode that `detectgraph` recommends to `initgraph`.

#### 4. delay()

- `void delay(unsigned milliseconds);`

Suspends execution for interval (milliseconds). With a call to delay, the current program is suspended from execution for the time specified by the argument milliseconds. It is not necessary to make a calibration call to delay before using it.

### 5. **getpixel(), putpixel()**

- unsigned far getpixel(int x, int y);
- void far putpixel(int x, int y, int color);

Getpixel gets the color of a specified pixel

Putpixel plots a pixel at a specified point

### 6. **arc(), circle()**

- void far arc(int x, int y, int stangle, int endangle, int radius);
- void far circle(int x, int y, int radius);

Arc draws a circular arc in the current drawing color.

Circle draws a circle in the current drawing color.

(x, y) Center point of arc, circle, or pie slice

stangle Start angle in degrees

endangle End angle in degrees

radius Radius of arc, circle, and pie slice

### 7. **getcolor(), setcolor()**

- int far getcolor(void);
- void far setcolor(int color);

Getcolor returns the current drawing color.

Setcolor sets the current drawing color to color, which can range from 0 to getmaxcolor.

To select a drawing color with setcolor, you can pass either the color number or the equivalent color name.

### 8. **getmaxx() and getmaxy()**

- int far getmaxx(void);
- int far getmaxy(void);

Getmaxx returns the maximum x value (screen-relative) for the current graphics driver and mode.

Getmaxy returns the maximum y value (screen-relative) for the current graphics driver and mode. For example, on a CGA in 320 x 200 mode, getmaxx returns 319 and getmaxy returns 199.

### 9. line()

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

Line draws a line from (x1, y1) to (x2, y2) using the current color, line style, and thickness. It does not update the current position (CP).

### 10. getx(), gety()

- int far getx(void);
- int far gety(void);

Getx returns the x-coordinate of the current graphics position.

Gety returns the y-coordinate of the current graphics position. The values are viewport-relative.

### 11. settextjustify()

Sets text justification for graphics mode

- void far settextjustify(int horiz, int vert);

Text output after a call to settextjustify is justified around the current position (CP) horizontally and vertically, as specified.

The default justification settings are LEFT\_TEXT (for horizontal) and TOP\_TEXT (for vertical)

### 12. outtext(), outtextxy()

- void far outtext(char far \*textstring);
- void far outtextxy(int x, int y, char far \*textstring);

Outtext and outtextxy display a text string, using the current justification settings and the current font, direction, and size.

Outtext outputs textstring at the current position (CP) outtextxy displays textstring in the viewport at the position (x, y).

## Q-2. Develop the DDA Line drawing algorithm using C language

### DDA Algorithm

- Digital Differential Analyzer (DDA) algorithm is the simple line generation algorithm which is explained step by step here.
- **Step 1** – Get the input of two end points  $(X_0, Y_0)$  and  $(X_1, Y_1)$ .
- **Step 2** – Calculate the difference between two end points.
- $dx = X_1 - X_0$
- $dy = Y_1 - Y_0$
- **Step 3** – Based on the calculated difference in step-2, you need to identify the number of steps to put pixel. If  $dx > dy$ , then you need more steps in x coordinate; otherwise in y coordinate.

```
if (absolute(dx) > absolute(dy))
    Steps = absolute(dx);
else
    Steps = absolute(dy);
```

- **Step 4** – Calculate the increment in x coordinate and y coordinate.
- X increment =  $dx / (\text{float}) \text{ steps}$ ;
- Y increment =  $dy / (\text{float}) \text{ steps}$ ;
- **Step 5** – Put the pixel by successfully incrementing x and y coordinates accordingly and complete the drawing of the line.

```
For (int v=0; v < Steps; v++)
{
    x = x + X increment;
    y = y + Y increment;
    putpixel(Round(x), Round(y));
}
```

### Code:

```
#include<stdio.h>>

#include<graphics.h>

#include<conio.h>

#include<math.h>

#include<dos.h>

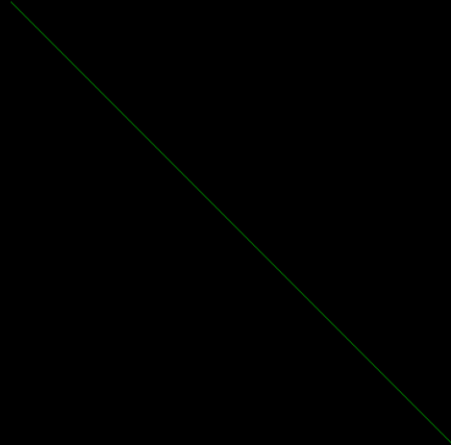
void main()

{
```

```
int gd = DETECT, gm;  
int i, x1, y1, x2, y2, dx, dy, steps, xi, yi;  
initgraph(&gd, &gm, "C:/TURBOC3/BGI");  
printf("Enter values of x1 and y1 \n");  
scanf("%d %d", &x1, &y1);  
printf("Enter values of x2 and y2 \n");  
scanf("%d %d", &x2, &y2);  
dx = x2 - x1;  
dy = y2 - y1;  
if (abs(dx) > abs(dy))  
    { steps = abs(dx); }  
else  
    { steps = abs(dy); }  
xi = dx / steps;  
yi = dy / steps;  
for (i = 0; i < steps; i++)  
{  
    putpixel(x1, y1, 2);  
    x1 = x1 + xi;  
    y1 = y1 + yi;  
}  
getch();  
closegraph();  
return 0;  
}
```

Output:

```
Enter values of x1 and y1  
100  
100  
Enter values of x2 and y2  
500  
500
```



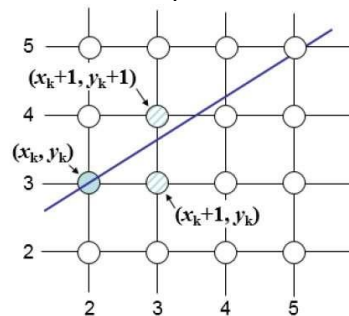


### Q-3. Develop the Bresenham's Line drawing algorithm using C language

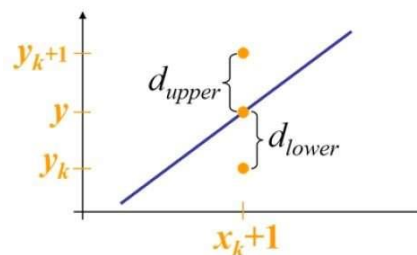
#### Bresenham's Line Generation

- The Bresenham algorithm is another incremental scan conversion algorithm. The big advantage of this algorithm is that, it uses only integer calculations. Moving across the x axis in unit intervals and at each step choose between two different y coordinates.

For example, as shown in the following illustration, from position 2,3 you need to choose between 3,3 and 3,4. You would like the point that is closer to the original line.



At sample position  $X_{k+1}$ ,  $Y_{k+1}$ , the vertical separations from the mathematical line are labelled as  $d_{upper}$  and  $d_{lower}$ .



From the above illustration, the y coordinate on the mathematical line at  $x_{k+1}$  is –

$$Y = mX_{k+1} + b$$

So,  $d_{upper}$  and  $d_{lower}$  are given as follows –

$$d_{lower} = y - y_k$$

$$= m(X_{k+1}) + b - Y_k = m(X_{k+1}) + b - Y_k$$

and

$$d_{upper} = (y_{k+1}) - y$$

$$= Y_{k+1} - m(X_{k+1}) - b = Y_{k+1} - m(X_{k+1}) - b$$

You can use these to make a simple decision about which pixel is closer to the mathematical line. This simple decision is based on the difference between the two pixel positions.

$$d_{lower}-d_{upper}=2m(x_k+1)-2y_k+2b-1 \quad d_{lower}-d_{upper}=2m(x_k+1)-2y_k+2b-1$$

Let us substitute  $m$  with  $dy/dx$  where  $dx$  and  $dy$  are the differences between the end-points.

$$\begin{aligned} dx(d_{lower}-d_{upper}) &= dx(2dydx(x_k+1)-2y_k+2b-1) \quad dx(d_{lower}-d_{upper})=dx(2dydx(x_k+1)-2y_k+2b-1) \\ &= 2dy.x_k-2dx.y_k+2dy+2dx(2b-1)=2dy.x_k-2dx.y_k+2dy+2dx(2b-1) \\ &= 2dy.x_k-2dx.y_k+C=2dy.x_k-2dx.y_k+C \end{aligned}$$

So, a decision parameter  $P_k$  for the  $k$ th step along a line is given by –

$$\begin{aligned} p_k &= dx(d_{lower}-d_{upper}) \quad p_k=dx(d_{lower}-d_{upper}) \\ &= 2dy.x_k-2dx.y_k+C=2dy.x_k-2dx.y_k+C \end{aligned}$$

The sign of the decision parameter  $P_k$  is the same as that of  $d_{lower}-d_{upper}$ .

If  $p_k$  is negative, then choose the lower pixel, otherwise choose the upper pixel.

Remember, the coordinate changes occur along the x axis in unit steps, so you can do everything with integer calculations. At step  $k+1$ , the decision parameter is given as –

$$p_{k+1}=2dy.x_{k+1}-2dx.y_{k+1}+C \quad p_{k+1}=2dy.x_{k+1}-2dx.y_{k+1}+C$$

Subtracting  $p_k$  from this we get –

$$p_{k+1}-p_k=2dy(x_{k+1}-x_k)-2dx(y_{k+1}-y_k) \quad p_{k+1}-p_k=2dy(x_{k+1}-x_k)-2dx(y_{k+1}-y_k)$$

But,  $x_{k+1}$  is the same as  $(x_k)+1$ . So –

$$p_{k+1}=p_k+2dy-2dx(y_{k+1}-y_k) \quad p_{k+1}=p_k+2dy-2dx(y_{k+1}-y_k)$$

Where,  $y_{k+1}-y_k$  is either 0 or 1 depending on the sign of  $P_k$ .

The first decision parameter  $p_0$  is evaluated at  $(x_0, y_0)$  is given as –

$$p_0=2dy-dx \quad p_0=2dy-dx$$

Now, keeping in mind all the above points and calculations, here is the Bresenham algorithm for slope  $m < 1$  –

**Step 1** – Input the two end-points of line, storing the left end-point in  $(x_0, y_0)$ .

**Step 2** – Plot the point  $(x_0, y_0)$ .

**Step 3** – Calculate the constants  $dx$ ,  $dy$ ,  $2dy$ , and  $2dy-2dx$  and get the first value for the decision parameter as –

$$p_0=2dy-dx \quad p_0=2dy-dx$$

**Step 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

$$p_{k+1}=p_k+2dy \quad p_{k+1}=p_k+2dy$$

Otherwise,

$$(x_k, y_{k+1})$$

$$p_{k+1} = p_k + 2dy - 2dx \quad p_{k+1} = p_k + 2dy - 2dx$$

**Step 5** – Repeat step 4  $dx-1$  times.

For  $m > 1$ , find out whether you need to increment  $x$  while incrementing  $y$  each time.

After solving, the equation for decision parameter  $P_k$  will be very similar, just the  $x$  and  $y$  in the equation gets interchanged.

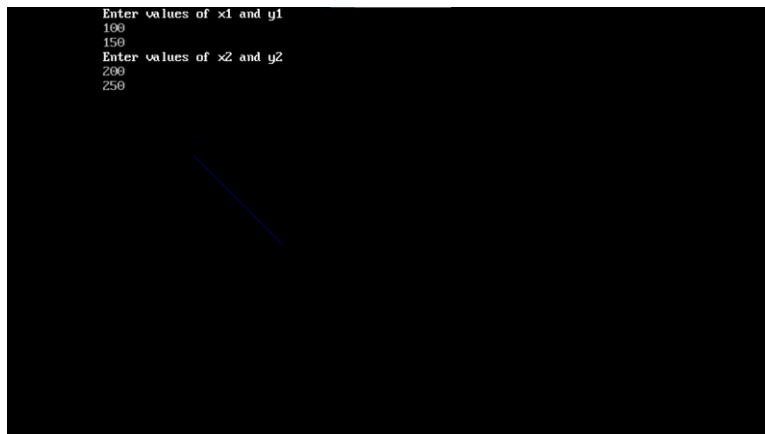
## CODE:

```
#include<stdio.h>
#include<graphics.h>
#include<conio.h>
#include<dos.h>
void main()
{
    int gd=DETECT,gm,i,m,dx,dy,xn,yn,x1,x2,y1,y2;
    initgraph(&gd,&gm,"C:/TURBOC3/BGI");
    printf("Enter values of x1 and y1 \n");
    scanf("%d %d",&x1,&y1);
    printf("Enter values of x2 and y2 \n");
    scanf("%d %d",&x2,&y2);
    dx = x2-x1;
    dy = y2-y1;
    xn = x1;
    yn = y1;
    for(i = 1;i<=dx;i++)
    {
        putpixel(xn,yn,1);
        delay(50);
```

```
while(m>=0)
{
    yn = yn+1;
    m = m - 2*dx;

}
xn = xn+1;
m = m+2*dy;
}
getch();
closegraph();
}
```

Output:



## Differentiate between DDA Algorithm and Bresenham's Line

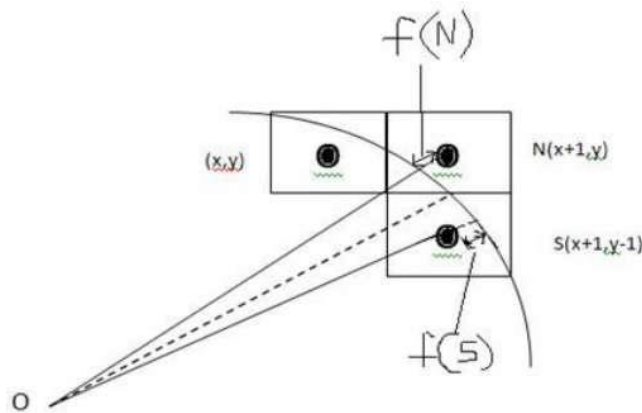
DDA Algorithm	Bresenham's Line Algorithm
1. DDA Algorithm use floating point, i.e., Real Arithmetic.	1. Bresenham's Line Algorithm use fixed point, i.e., Integer Arithmetic
2. DDA Algorithms uses multiplication & division its operation	2. Bresenham's Line Algorithm uses only subtraction and addition its operation
3. DDA Algorithm is slowly than Bresenham's Line Algorithm in line drawing because it uses real arithmetic (Floating Point operation)	3. Bresenham's Algorithm is faster than DDA Algorithm in line because it involves only addition & subtraction in its calculation and uses only integer arithmetic.
4. DDA Algorithm is not accurate and efficient as Bresenham's Line Algorithm.	4. Bresenham's Line Algorithm is more accurate and efficient at DDA Algorithm.
5. DDA Algorithm can draw circle and curves but are not accurate as Bresenham's Line Algorithm	5. Bresenham's Line Algorithm can draw circle and curves with more accurate than DDA Algorithm.

## Q-4. Develop the Bresenham's Circle drawing algorithm using C language

### ○ Bresenham's Algorithm

We cannot display a continuous arc on the raster display. Instead, we have to choose the nearest pixel position to complete the arc.

From the following illustration, you can see that we have put the pixel at  $X, Y$  location and now need to decide where to put the next pixel – at  $N(X+1, Y)$  or at  $S(X+1, Y-1)$ .



This can be decided by the decision parameter  $d$ .

- If  $d \leq 0$ , then  $N(X+1, Y)$  is to be chosen as next pixel.
- If  $d > 0$ , then  $S(X+1, Y-1)$  is to be chosen as the next pixel.

### Algorithm

**Step 1** – Get the coordinates of the center of the circle and radius, and store them in  $x$ ,  $y$ , and  $R$  respectively. Set  $P=0$  and  $Q=R$ .

**Step 2** – Set decision parameter  $D = 3 - 2R$ .

**Step 3** – Repeat through step-8 while  $P \leq Q$ .

**Step 4** – Call Draw Circle  $X, Y, P, Q, X, Y, P, Q$ .

**Step 5** – Increment the value of  $P$ .

**Step 6** – If  $D < 0$  then  $D = D + 4P + 6$ .

**Step 7** – Else Set  $R = R - 1$ ,  $D = D + 4P - Q - Q + 10$ .

**Step 8** – Call Draw Circle  $X, Y, P, Q, X, Y, P, Q$ .

Draw Circle Method ( $X, Y, P, Q$ ).

Call Putpixel ( $X + P, Y + Q$ ).

Call Putpixel ( $X - P, Y + Q$ ).

Call Putpixel (X + P, Y - Q).  
 Call Putpixel (X - P, Y - Q).  
 Call Putpixel (X + Q, Y + P).  
 Call Putpixel (X - Q, Y + P).  
 Call Putpixel (X + Q, Y - P).  
 Call Putpixel (X - Q, Y - P).

### CODE:

```
#include<stdio.h>
#include<dos.h>
#include<conio.h>
#include<graphics.h>
void drawCir(int xr,int yr,int x,int y)
{
    putpixel(xr+x, yr+y, 1);
    putpixel(xr-x, yr+y, 2);
    putpixel(xr+x, yr-y, 3);
    putpixel(xr-x, yr-y, 4);
    putpixel(xr+y, yr+x, 5);
    putpixel(xr-y, yr+x, 6);
    putpixel(xr+y, yr-x, 7);
    putpixel(xr-y, yr-x, 8);
}
void circleAlgo(int xr,int yr, int r)
{
    int x = 0, y = r;
    int d = 3 - 2 * r;
    drawCir(xr,yr,x,y);
    while(y >= x)
    {
        x++;

        if(d > 0)
        {
            y--;
            d = d + 4 * (x-y) + 10;
        }
        else
        {
            d = d + 4 * x + 6;
        }
        drawCir(xr,yr,x,y);
        delay(100);
    }
}
```

```
}  
  
int main()  
{  
    int xr = 50, yr = 50, r2 = 30;  
    int gd = DETECT, gm;  
    initgraph(&gd, &gm, "C:/TURBOC3/BGI");  
    circleAlgo(xr, yr, r2);  
    getch();  
    return 0;  
}
```

Output:





**Q-5. Develop the C program for to display different types of lines****➤ Code:**

```
#include <graphics.h>
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <conio.h>
void main()
{
    int gd=DETECT,gm;
    int s;
    int x1,x2,y1,y2;
    char *lname[]={"SOLID LINE","DOTTED LINE","CENTER LINE",
    "DASHED LINE","USERBIT LINE"};
    initgraph(&gd,&gm,"C://TURBOC3//bgi");
    clrscr();
    cleardevice();
        printf("\nEnter starting point (x1 ,y1):");
        scanf("%d %d",&x1,&y1);
        printf("\nEnter End point (x2,y2):");
        scanf("%d %d",&x2,&y2);
    printf("Line styles:");
    for (s=0;s<5;s++)
    {setcolor(s+1);
    setlinestyle(s,1,3);
    line(x1,y1+s*50,x2,y2+s*50);

    outtextxy(x2+20,y2+20+s*50,lname[s]);
    }
    getch();
    closegraph();
}
```

}

Output:

```
Enter starting point (x1 ,y1):100
130

Enter End point (x2,y2):200
250

Line styles:

SOLID LINE
DOTTED LINE
CENTER LINE
DASHED LINE
USERBIT LINE
```

## Q-6. Perform the following 2D Transformation operation - Translation, Rotation and Scaling

➤ Types of Transformations:

Translation

Scaling

Rotating

Reflection

Shearing

### 1) Translation

$$x_1 = x + T_x$$

$$y_1 = y + T_y$$

**Code:**

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

// function to translate rectangle
void translateRectangle ( int P[][2], int T[])
{
    /* init graph and rectangle() are used for
    representing rectangle through graphical functions */
    int gd = DETECT, gm, errorcode;
    initgraph (&gd, &gm, "c:\\TURBOC3\\bgi");
    setcolor (2);
    // rectangle (Xmin, Ymin, Xmax, Ymax)
    // original rectangle
    rectangle (P[0][0], P[0][1], P[1][0], P[1][1]);

    // calculating translated coordinates
    P[0][0] = P[0][0] + T[0];
    P[0][1] = P[0][1] + T[1];
    P[1][0] = P[1][0] + T[0];
    P[1][1] = P[1][1] + T[1];

    // translated rectangle (Xmin, Ymin, Xmax, Ymax)
```

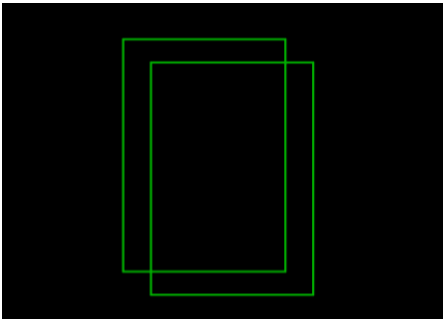
```

// setcolor(3);
rectangle (P[0][0], P[0][1], P[1][0], P[1][1]);
// closegraph();
}

// driver program
int main()
{
    // Xmin, Ymin, Xmax, Ymax as rectangle
    // coordinates of top left and bottom right points
    int P[2][2] = {50, 80, 120, 180};
    int T[] = {12, 10}; // translation factor
    translateRectangle (P, T);
    getch();
    return 0;
}

```

Output:



## (2) Scaling:

It is used to alter or change the size of objects. The change is done using scaling factors. There are two scaling factors, i.e.  $S_x$  in x direction  $S_y$  in y-direction. If the original position is  $x$  and  $y$ . Scaling factors are  $S_x$  and  $S_y$  then the value of coordinates after scaling will be  $x_1$  and  $y_1$ .

Enlargement: If  $T_1 = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}$ , If  $(x_1 \ y_1)$  is original position and  $T_1$  is translation vector then  $(x_2 \ y_2)$  are coordinated after scaling

$$[x_2 \ y_2] = [x_1 \ y_1] \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix} = [2x_1 \ 2y_1]$$

### Code:

```

#include<iostream.h>
#include<graphics.h>

```

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

```
void mul(int mat[3][3],int vertex[10][3],int n);
void scale(int vertex[10][3],int n);
void init(int vertex[10][3],int n);
```

```
int main()
{
    int i,x,y;
    int vertex[10][3],n;

    clrscr();

    cout<<"\nEnter the no. of vertex : ";
    cin>>n;
    for(i=0;i<n;i++)
    {
        cout<<"Enter the points (x,y): ";
        cin>>x>>y;
        vertex[i][0]=x;
        vertex[i][1]=y;
        vertex[i][2]=1;
    }

    scale(vertex,n);

    getch();
    return 0;
}
```

```
void init(int vertex[10][3],int n)
{
    int gd=DETECT,gm,i;
    initgraph(&gd,&gm,"C:\\turboc3\\bgi");

    setcolor(10);
    line(0,240,640,240);    //drawing X axis
    line(320,0,320,480);    //drawing Y axis

    setcolor(3);
```

```

line(450,20,490,20);
setcolor(15);
line(450,50,490,50);
setcolor(6);
outtextxy(500,20,"Original");
outtextxy(500,50,"Transformed");

setcolor(3);

for(i=0;i<n-1;i++)
{
    line(320+vertex[i][0],240-vertex[i][1],320+vertex[i+1][0],240-vertex[i+1][1]);
}
line(320+vertex[n-1][0],240-vertex[n-1][1],320+vertex[0][0],240-vertex[0][1]);
}

void mul(int mat[3][3],int vertex[10][3],int n)
{
    int i,j,k;  // loop variables
    int res[10][3];

    for(i=0;i<n;i++)
    {
        for(j=0;j<3;j++)
        {
            res[i][j]=0;
            for(k=0;k<3;k++)
            {
                res[i][j] = res[i][j] + vertex[i][k]*mat[k][j];
            }
        }
    }
    setcolor(15);
    for(i=0;i<n-1;i++)
    {
        line(320+res[i][0],240-res[i][1],320+res[i+1][0],240-res[i+1][1]);
    }
    line(320+res[n-1][0],240-res[n-1][1],320+res[0][0],240-res[0][1]);
}

void scale(int vertex[10][3],int n)
{
    //scaling transformation matrix
    int scale_array[3][3];
    int x,y;
    cout<<"\nEnter scale factor in X direction: ";

```

```
cin>>x;
cout<<"Enter scale factor in Y direction: ";
cin>>y;
```

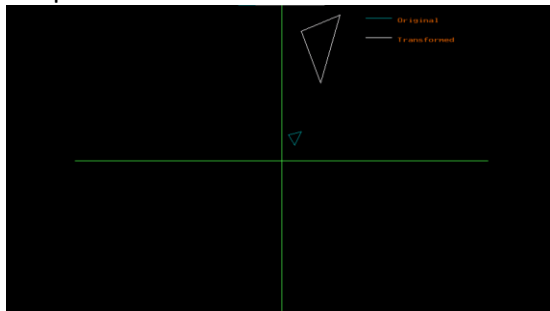
```
    scale_array[0][0]=x;
    scale_array[1][0]=0;
    scale_array[2][0]=0;
    scale_array[0][1]=0;
    scale_array[1][1]=y;
    scale_array[2][1]=0;
    scale_array[0][2]=0;
    scale_array[1][2]=0;
    scale_array[2][2]=1;

    init(vertex,n);

    mul(scale_array,vertex,n);
```

```
}
```

Output:



### (3) Rotation:

It is a process of changing the angle of the object. Rotation can be clockwise or anticlockwise. For rotation, we have to specify the angle of rotation and rotation point. Rotation point is also called a pivot point. It is print about which object is rotated.

Code:

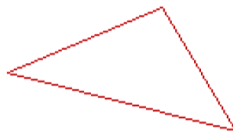
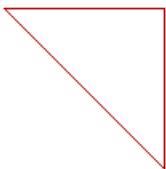
```
#include<stdio.h>
#include<graphics.h>
#include<math.h>
#include<conio.h>
main()
{
    int gd=0,gm,x1,y1,x2,y2,x3,y3;
    double s,c, angle;
    initgraph(&gd, &gm, "C:\\TURBOC3\\BGI");
```

```

setcolor(RED);
printf("Enter coordinates of triangle: ");
scanf("%d%d%d%d%d%d",&x1,&y1,&x2,&y2, &x3, &y3);
setbkcolor(WHITE);
cleardevice();
line(x1,y1,x2,y2);
line(x2,y2, x3,y3);
line(x3, y3, x1, y1);
getch();
setbkcolor(BLACK);
printf("Enter rotation angle: ");
scanf("%lf", &angle);
setbkcolor(WHITE);
c = cos(angle *M_PI/180);
s = sin(angle *M_PI/180);
x1 = floor(x1 * c + y1 * s);
y1 = floor(-x1 * s + y1 * c);
x2 = floor(x2 * c + y2 * s);
y2 = floor(-x2 * s + y2 * c);
x3 = floor(x3 * c + y3 * s);
y3 = floor(-x3 * s + y3 * c);
cleardevice();
line(x1, y1 ,x2, y2);
line(x2,y2, x3,y3);
line(x3, y3, x1, y1);
getch();
closegraph();
return 0;
}

```

Output: 25 degree



## (4) Reflection:

1. Reflection about the x-axis
2. Reflection about the y-axis
3. Reflection about an axis perpendicular to xy plane and passing through the origin
4. Reflection about line  $y=x$



**Code:**

```

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

#define PI 3.141592654

typedef float matrix[3][3];
matrix initialmatrix;
enum axis {xaxis, yaxis, yequals_minusx, yequals_x};
void unit_matrix(matrix m)
{
    int r,c;
    for (r=0; r<3; r++)
        for (c=0; c<3; c++)
            m[r][c] = (r==c);
}
void matrixmultiply(matrix a, matrix b)
{
    int r,c;
    matrix tmp;
    for (r=0; r<3; r++)
        for (c=0; c<3; c++)
            tmp[r][c] = a[r][0]*b[0][c] + a[r][1]*b[1][c] + a[r][2]*b[2][c];
    for (r=0; r<3; r++)
        for (c=0; c<3; c++)
            b[r][c] = tmp[r][c];
}
void mirror2d(axis a)
{
    matrix m;
    unit_matrix(m);
    switch (a)
    {
        case xaxis:
            m[1][1] = - 1;
            break;
        case yaxis:
            m[0][0] = - 1;
            break;
        case yequals_minusx:
            m[0][0] = 0; m[0][1] = -1; m[1][0] = -1; m[1][1] = 0;
            break;
        case yequals_x:

```

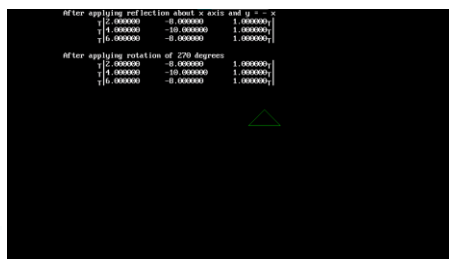
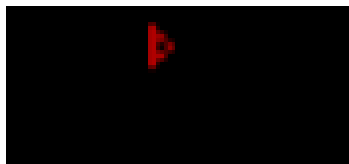
```

        m[0][0] = 0; m[0][1] = 1; m[1][0] = 1; m[1][1] = 0;
        break;
    }
    matrixmultiply(m, initialmatrix);
}
void translate2d(float tx, float ty)
{
    matrix m;
    unit_matrix(m);
    m[0][2] = tx; m[1][2] = ty;
    matrixmultiply(m, initialmatrix);
}
void scale2d(float sx, float sy)
{
    matrix m;
    unit_matrix(m);
    m[0][0] = sx;
    m[1][1] = sy;
    matrixmultiply(m, initialmatrix);
}
void rotate2d(float radianangle)
{
    matrix m;
    unit_matrix(m);
    m[0][0] = cos(radianangle);
    m[1][0] = sin(radianangle);
    m[0][1] = - sin(radianangle);
    m[1][1] = cos(radianangle);
    matrixmultiply(m, initialmatrix);
}
void main(void)
{
    int gd=DETECT, gm;
    initgraph(&gd, &gm, "C:/TURBOC3/BGI");
    float rotateangle = 270 * PI/180;
    //Defining initialmatrix, a triangle
    initialmatrix[0][0] = 8; initialmatrix[0][1] = 10; initialmatrix[0][2] = 8;
    initialmatrix[1][0] = 2; initialmatrix[1][1] = 4; initialmatrix[1][2] = 6;
    initialmatrix[2][0] = 1; initialmatrix[2][1] = 1; initialmatrix[2][2] = 1;
    setcolor(RED);
    //Drawing initialmatrix (Triangle)
    line(initialmatrix[0][0],initialmatrix[1][0],initialmatrix[0][1],initialmatrix[1][1]);
    line(initialmatrix[0][1],initialmatrix[1][1],initialmatrix[0][2],initialmatrix[1][2]);
    line(initialmatrix[0][2],initialmatrix[1][2],initialmatrix[0][0],initialmatrix[1][0]);
    //Wait for a keystroke
    getch();
}

```

```
//2D Operations
//Refer Solved example 4 - 11
//Mirroring about x axis and then about y = -x
mirror2d(xaxis);
mirror2d(yequals_minusx);
//Print the final Matrix
printf("After applying reflection about x axis and y = - x\n");
for (int i=0;i<3;i++)
    printf("\t3%f\t%f\t%f^3\n", initialmatrix[0][i], initialmatrix[1][i], initialmatrix[2][i]);
//Starting again with initial matrix
initialmatrix[0][0] = 8; initialmatrix[0][1] = 10; initialmatrix[0][2] = 8;
initialmatrix[1][0] = 2; initialmatrix[1][1] = 4; initialmatrix[1][2] = 6;
initialmatrix[2][0] = 1; initialmatrix[2][1] = 1; initialmatrix[2][2] = 1;
//Rotating the triangle by 270 degrees
rotate2d(rotateangle);
printf("\nAfter applying rotation of 270 degrees\n");
for (i=0;i<3;i++)
    printf("\t3%f\t%f\t%f^3\n", initialmatrix[0][i], initialmatrix[1][i], initialmatrix[2][i]);
//Translating and Magnifying for proper visualization
//of transformed triangle and displaying the transformed initialmatrix
scale2d(15, 15);
translate2d(getmaxx()/2, getmaxy()/2 + 100);
setcolor(GREEN);
line(initialmatrix[0][0],initialmatrix[1][0],initialmatrix[0][1],initialmatrix[1][1]);
line(initialmatrix[0][1],initialmatrix[1][1],initialmatrix[0][2],initialmatrix[1][2]);
line(initialmatrix[0][2],initialmatrix[1][2],initialmatrix[0][0],initialmatrix[1][0]);
getch();
}
```

Output:



### 5) Shearing:

It is transformation which changes the shape of object. The sliding of layers of object occur. The shear can be in one direction or in two directions.

Code:

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

```

void main()
{
int gd=DETECT,gm,option,xref,yref;
int i,maxx,maxy,x1,y1,x2,y2,x3,y3,x4,y4,gap=50;
float shx=0.0,shy=0.0;
char str[5];
clrscr();
initgraph(&gd,&gm,"C:\\TURBOC3\\BGI");

printf("enter the endpoints of the top of the rectangle (x1,y1) & (x2,y2):");
scanf("%d%d%d%d",&x1,&y1,&x2,&y2);
printf("enter the endpoints of bottom of the rectangle (x3,y3) & (x4,y4)");
scanf("%d%d%d%d",&x3,&y3,&x4,&y4);
printf(" Enter the axis to shear\n");
printf(" 1 - X axis Shear\n");
printf(" 2 - Y axis shear\n");
scanf("%d",&option );
if(option==1)
{
printf("enter the value for x-axis shear( can be fraction too):");
scanf("%f",&shx);
}
else
{
printf("enter the value for y-axis shear( can be fraction too):");
scanf("%f",&shy);
}

clearviewport();
maxx= getmaxx();
maxy=getmaxy();
line(3,maxy-1,maxx-5,maxy-1);
line(5,5,5,maxy-3);

for( i= 0;i<maxx-5;i=i+gap)          // code to display co-ordinates
{
outtextxy(i+3,maxy-7,"|");
itoa(i,str,10);
outtextxy(i,maxy-10,str);
}
for( i= maxy;i>0;i=i-gap)
{
outtextxy(3,i,"-");
itoa(maxy-i,str,10);
outtextxy(9,i,str);
}

setcolor(50);          // drawing rectangle using endpoints
line(x1,maxy-y1,x2,maxy-y2);
line(x3,maxy-y3,x4,maxy-y4);
line(x1,maxy-y1,x3,maxy-y3);
line(x2,maxy-y2,x4,maxy-y4);
outtextxy(10,10,"hit any key to see the shearing effect" );

```

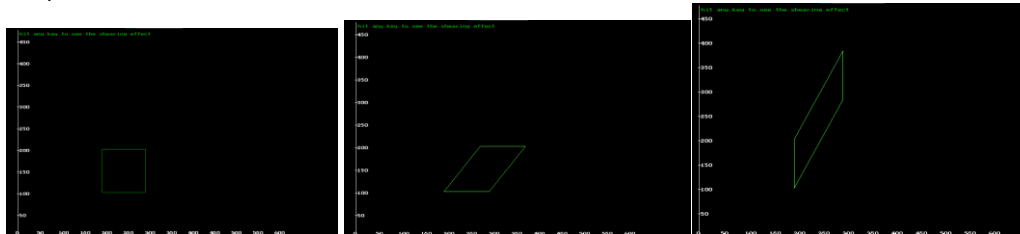
```

getch();
setcolor(0); // to hide the rectangle drawn
line(x1,maxy-y1,x2,maxy-y2);
line(x3,maxy-y3,x4,maxy-y4);
line(x1,maxy-y1,x3,maxy-y3);
line(x2,maxy-y2,x4,maxy-y4);

setcolor(58); // to redraw the rectangle
if(option==1)
{
    // shearing about x axis so only points x1 and x2 need to be recomputed
    line(x1+shx*y1,maxy-y1,x2+shx*y2,maxy-y2);
    line(x3,maxy-y3,x4,maxy-y4);
    line(x1+shx*y1,maxy-y1,x3,maxy-y3);
    line(x2+shx*y2,maxy-y2,x4,maxy-y4);
}
else
{
    // shearing about y axis so only points y2 and y4 need to be recomputed
    line(x1,maxy-y1,x2,maxy-(y2+shy*x2));
    line(x3,maxy-y3,x4,maxy-(y4+shy*x4));
    line(x1,maxy-y1,x3,maxy-y3);
    line(x2,maxy-(y2+shy*x2),x4,maxy-(y4+shy*x4));
}
getch();
closegraph();
}

```

Output:



## 2 D transformation:

CODE:

```

#include <graphics.h>
#include <stdio.h>
#include <conio.h>
#include <math.h>
#include <stdlib.h>
void main()
{
    int gm,gd=DETECT,midx,midy,c;
    float x1,x2,x3,y1,y2,y3,x11,x22,x33,y11,y22,y33,sfx,sfy,tpx,tpy,ang,t,a,b;

```

```

clrscr();
initgraph(&gd,&gm,"C:/TURBOC3/BGI");
midx=getmaxx()/2;
midy=getmaxy()/2;
line(midx,0,midx,getmaxy());
line(0,midy,getmaxx(),midy);
printf("2D Translation, Rotation and Scaling\n");
printf("Enter the points (x1,y1), (x2,y2) and (x3,y3) of triangle\n");
scanf("%f%f%f%f%f%f",&x1,&y1,&x2,&y2,&x3,&y3);
setcolor(WHITE);
line(midx+x1,midy-y1,midx+x2,midy-y2);
line(midx+x2,midy-y2,midx+x3,midy-y3);
line(midx+x3,midy-y3,midx+x1,midy-y1);
printf("\n 1.Transaction\n 2.Rotation\n 3.Scalling\n 4.exit\n");
a: printf("Enter your choice:");
scanf("%d",&c);
switch(c)
{
case 1:
printf("Enter the translation factor\n");
scanf("%f%f",&tpx,&tpy);
x11=x1+tpx;
y11=y1+tpy;
x22=x2+tpx;
y22=y2+tpy;
x33=x3+tpx;
y33=y3+tpy;
setcolor(RED);
line(midx+x11,midy-y11,midx+x22,midy-y22);
line(midx+x22,midy-y22,midx+x33,midy-y33);
line(midx+x33,midy-y33,midx+x11,midy-y11);
break;

case 2:
printf("Enter the angle of rotation\n");
scanf("%f",&ang);
printf("Enter rotetion point\n");
scanf("%f%f",&a,&b);
t=3.14*ang/180;
x11=abs(x1*cos(t)-y1*sin(t)+a*(1-cos(t))+b*sin(t));
y11=abs(x1*sin(t)+y1*cos(t)+b*(1-cos(t))-a*sin(t));
x22=abs(x2*cos(t)-y2*sin(t)+a*(1-cos(t))+b*sin(t));
y22=abs(x2*sin(t)+y2*cos(t)+b*(1-cos(t))-a*sin(t));
x33=abs(x3*cos(t)-y3*sin(t)+a*(1-cos(t))+b*sin(t));
y33=abs(x3*sin(t)+y3*cos(t)+b*(1-cos(t))-a*sin(t));
setcolor(BLUE);
line(midx+x11,midy-y11,midx+x22,midy-y22);
line(midx+x22,midy-y22,midx+x33,midy-y33);
line(midx+x33,midy-y33,midx+x11,midy-y11);
break;

case 3:
printf("Enter the scalling factor\n");
scanf("%f%f",&sfx,&sfy);

```

```

printf("Enter scaling point\n");
scanf("%f%f",&a,&b);
x11=x1*sfx+a*(1-sfx);
y11=y2*sfx+b*(1-sfy);
x22=x2*sfx+a*(1-sfx);
y22=y2*sfx+b*(1-sfy);
x33=x3*sfx+a*(1-sfx);
y33=y3*sfx+b*(1-sfy);
setcolor(YELLOW);
line(midx+x11,midy-y11,midx+x22,midy-y22);
line(midx+x22,midy-y22,midx+x33,midy-y33);
line(midx+x33,midy-y33,midx+x11,midy-y11);
break;

```

```

case 4:
exit(0);
break;

```

```

default:
printf("Enter the correct choice\n");
}
goto a;
getch();
}

```

(2)Output:

```

Enter the points (x1,y1), (x2,y2) and (x3,y3) of triangle
20
10
45
20
80

1.Transaction
2.Rotation
3.Scalling
4.exit
Enter your choice:1
Enter the translation factor
4
6
Enter your choice:2
Enter the angle of rotation
25
Enter rotation point
0
0
Enter your choice:3
Enter the scalling factor
12
10
Enter scaling point
0
0
Enter your choice:

```

## Q-7. Perform the Line Clipping Algorithm

### Cohen-Sutherland Line Clipping Algorithm

**Step1:** Calculate positions of both endpoints of the line

**Step2:** Perform OR operation on both of these end-points

**Step3:** If the OR operation gives 0000

Then

line is considered to be visible

else

Perform AND operation on both endpoints

If And  $\neq$  0000

then the line is invisible

else

And=0000

Line is considered the clipped case.

**Step4:** If a line is clipped case, find an intersection with boundaries of the window

$$m = (y_2 - y_1) / (x_2 - x_1)$$

**(a)** If bit 1 is "1" line intersects with left boundary of rectangle window

$$y_3 = y_1 + m(x - x_1)$$

where  $x = x_{wmin}$

where  $x_{wmin}$  is the minimum value of X co-ordinate of window

**(b)** If bit 2 is "1" line intersect with right boundary

$$y_3 = y_1 + m(x - x_1)$$

where  $x = x_{wmax}$

where  $x$  more is maximum value of X co-ordinate of the window

**(c)** If bit 3 is "1" line intersects with bottom boundary

$$x_3 = x_1 + (y - y_1) / m$$

where  $y = y_{wmin}$

$y_{wmin}$  is the minimum value of Y co-ordinate of the window

**(d)** If bit 4 is "1" line intersects with the top boundary

$$x_3 = x_1 + (y - y_1) / m$$

where  $y = y_{wmax}$

$y_{wmax}$  is the maximum value of Y co-ordinate of the window

## Code:

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

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

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



```
#include<graphics.h>

#include<dos.h>

typedef struct coordinate

{

    int x,y;

    char code[4];

}PT;

void drawwindow();

void drawline(PT p1,PT p2);

PT setcode(PT p);

int visibility(PT p1,PT p2);

PT resetendpt(PT p1,PT p2);

void main()

{

    int gd=DETECT,v,gm;

    PT p1,p2,p3,p4,ptemp;

    printf("\nEnter x1 and y1\n");

    scanf("%d %d",&p1.x,&p1.y);

    printf("\nEnter x2 and y2\n");

    scanf("%d %d",&p2.x,&p2.y);

    initgraph(&gd,&gm,"c:\\turbo3\\bgi");

    drawwindow();

    delay(500);

    drawline(p1,p2);

    delay(500);
```

```
cleardevice();

delay(500);

p1=setcode(p1);

p2=setcode(p2);

v=visibility(p1,p2);

delay(500);

switch(v)
{
case 0: drawwindow();

        delay(500);

        drawline(p1,p2);

        break;

case 1: drawwindow();

        delay(500);

        break;

case 2: p3=resetendpt(p1,p2);

        p4=resetendpt(p2,p1);

        drawwindow();

        delay(500);

        drawline(p3,p4);

        break;

}

delay(5000);

closegraph();
}
```

```
void drawwindow()
{
    line(150,100,450,100);

    line(450,100,450,350);

    line(450,350,150,350);

    line(150,350,150,100);
}

void drawline(PT p1,PT p2)
{
    line(p1.x,p1.y,p2.x,p2.y);
}

PT setcode(PT p) //for setting the 4 bit code
{
    PT ptemp;

    if(p.y<100)
        ptemp.code[0]='1';    //Top
    else
        ptemp.code[0]='0';

    if(p.y>350)
        ptemp.code[1]='1';    //Bottom
    else
        ptemp.code[1]='0';

    if(p.x>450)
        ptemp.code[2]='1';    //Right
    else
```

```
        ptemp.code[2]='0';

    if(p.x<150)

        ptemp.code[3]='1';    //Left

    else

        ptemp.code[3]='0';

    ptemp.x=p.x;

    ptemp.y=p.y;

    return(ptemp);

}

int visibility(PT p1,PT p2)

{

    int i,flag=0;

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

    {

        if((p1.code[i]!='0') || (p2.code[i]!='0'))

            flag=1;

    }

    if(flag==0)

        return(0);

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

    {

        if((p1.code[i]==p2.code[i]) && (p1.code[i]=='1'))

            flag='0';

    }

}
```

```
        if(flag==0)
            return(1);

        return(2);
    }

PT resetendpt(PT p1,PT p2)
{
    PT temp;

    int x,y,i;

    float m,k;

    if(p1.code[3]=='1')
        x=150;

    if(p1.code[2]=='1')
        x=450;

    if((p1.code[3]=='1') || (p1.code[2]=='1'))
    {
        m=(float)(p2.y-p1.y)/(p2.x-p1.x);

        k=(p1.y+(m*(x-p1.x)));

        temp.y=k;

        temp.x=x;

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

            temp.code[i]=p1.code[i];

        if(temp.y<=350 && temp.y>=100)

            return (temp);

    }

    if(p1.code[0]=='1')
```

```
        y=100;

    if(p1.code[1]=='1')

        y=350;

    if((p1.code[0]=='1') || (p1.code[1]=='1'))

    {

        m=(float)(p2.y-p1.y)/(p2.x-p1.x);

        k=(float)p1.x+(float)(y-p1.y)/m;

        temp.x=k;

        temp.y=y;

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

            temp.code[i]=p1.code[i];

        return(temp);

    }

    else

        return(p1);

}
```

Output:



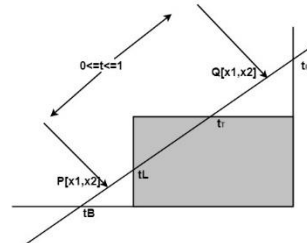
## Liang-Barsky Line Clipping Algorithm:

Liang and Barsky have established an algorithm that uses floating-point arithmetic but finds the appropriate endpoints with at most four computations. This algorithm uses the parametric equations for a line and solves four inequalities to find the range of the parameter for which the line is in the viewport.

Let  $P(x_1, y_1)$ ,  $Q(x_2, y_2)$  is the line which we want to study. The parametric equation of the line segment from gives x-values and y-values for every point in terms of a parameter that ranges from 0 to 1. The equations are

$$x = x_1 + (x_2 - x_1) * t = x_1 + dx * t \text{ and } y = y_1 + (y_2 - y_1) * t = y_1 + dy * t$$

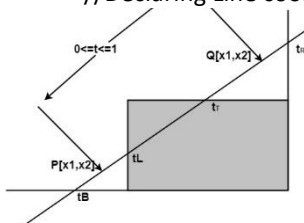
We can see that when  $t = 0$ , the point computed is  $P(x_1, y_1)$ ; and when  $t = 1$ , the point computed is  $Q(x_2, y_2)$ .



### Code:

```
#include <graphics.h>
#include <conio.h>
//Macro to eliminate fractional part of a floating number

#define round(a) ((int)(a + 0.5))
#define TRUE 1
#define FALSE 0
//Structure for window coordinates
struct window_coordinates
{
    float xmin, xmax, ymin, ymax;
};
//Structure for any line coordinates
struct line_coordinates
{
    float x1, y1, x2, y2;
};
int clip_line(window_coordinates wc, line_coordinates &lc);
int check_line(float d, float q, float &u1, float &u2);
void main(void)
{
    int driver=DETECT, mode;
    initgraph(&driver, &mode, "c:\\TURBOC3\\bgi");
    //Declaring Window Coordinates
    window_coordinates win;
    win.xmin = 100; win.xmax = 540;
    win.ymin = 100; win.ymax = 380;
    //Declaring Line coordinates which is to be clipped against the window
```



```

line_coordinates lc;
lc.x1 = 120; lc.y1 = 12;
lc.x2 = 300; lc.y2 = 390;
//Drawing the window boundaries
setcolor(YELLOW);
rectangle(win.xmin, win.ymin, win.xmax, win.ymax);
//Drawing the Initial line
setcolor(LIGHTGREEN);
line(lc.x1,lc.y1,lc.x2,lc.y2);
getch();
setcolor(LIGHTRED);
//Drawing the final clipped line after clipping with circle at its ends.
if(clip_line(win, lc))
{
line(round(lc.x1),round(lc.y1),round(lc.x2),round(lc.y2));
circle(round(lc.x1),round(lc.y1),4);
circle(round(lc.x2),round(lc.y2),4);
}
getch();
}
//Subroutine for Main Clipping Algorithm
int clip_line(window_coordinates win, line_coordinates &lc)
{
float u1 = 0;//umax
float u2 = 1;//umin
float dx = lc.x2-lc.x1;
float dy = lc.y2-lc.y1;
float d1 = -dx;
float d2 = dx;
float d3 = -dy;
float d4 = dy;
float q1 = lc.x1 - win.xmin;//Left Edge
float q2 = win.xmax - lc.x1;//Right Edge
float q3 = lc.y1 - win.ymin;//Bottom Edge
float q4 = win.ymax - lc.y1;//Top Edge
if(check_line(d1,q1,u1,u2) && check_line(d2,q2,u1,u2) &&
    check_line(d3,q3,u1,u2) && check_line(d4,q4,u1,u2))
{
if(u2 < 1)
{
lc.x2 = lc.x1 + (u2*dx);
lc.y2 = lc.y1 + (u2*dy);
}
if(u1 > 0)
{
lc.x1+= u1*dx;
lc.y1+= u1*dy;
}
return TRUE;
}
return FALSE;
}
//Subroutine for Trivial rejection test and finding the maximum of the lower

```

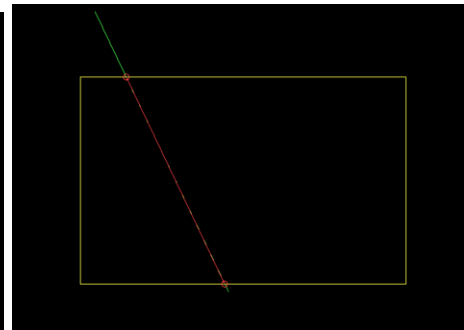
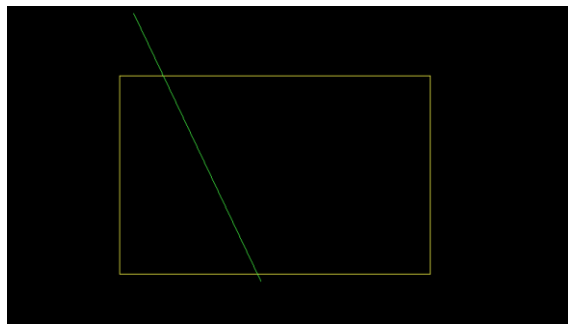


```

//sets of parameter value and minimum of maximum sets of parameter values
int check_line(float d, float q, float &u1, float &u2)
{
    int visible = TRUE;
    float u = q/d;
    if(d < 0)//look for upper limit
    {
        if(u > u2)          //check for trivially invisible
            visible = FALSE;
        else if (u > u1)//find the minimum of the maximum
            u1 = u;
    }
    else if(d > 0)          //look of lower limit
    {
        if(u < u1)          //check for trivially invisible
            visible = FALSE;
        else if (u < u2)//find the maximum of the minimum
            u2 = u;
    }
    else
    {
        if(q < 0)//line is outside
            visible = FALSE;
    }
    return visible;
}

```

Output:



**Q-8. Perform the Polygon clipping algorithm****Code:**

```
#include<iostream.h>
#include<conio.h>
#include<graphics.h>
#define round(a) ((int)(a+0.5))
int k;
float xmin,ymin,xmax,ymax,arr[20],m;
void clipl(float x1,float y1,float x2,float y2)
{
    if(x2-x1)
        m=(y2-y1)/(x2-x1);
    else
        m=100000;
    if(x1 >= xmin && x2 >= xmin)
    {
        arr[k]=x2;
        arr[k+1]=y2;
        k+=2;
    }
    if(x1 < xmin && x2 >= xmin)
    {
        arr[k]=xmin;
        arr[k+1]=y1+m*(xmin-x1);
        arr[k+2]=x2;
        arr[k+3]=y2;
        k+=4;
    }
    if(x1 >= xmin && x2 < xmin)
    {
        arr[k]=xmin;
```

```
        arr[k+1]=y1+m*(xmin-x1);
        k+=2;
    }
}

void clipt(float x1,float y1,float x2,float y2)
{
    if(y2-y1)
        m=(x2-x1)/(y2-y1);
    else
        m=100000;
    if(y1 <= ymax && y2 <= ymax)
    {
        arr[k]=x2;
        arr[k+1]=y2;
        k+=2;
    }
    if(y1 > ymax && y2 <= ymax)
    {
        arr[k]=x1+m*(ymax-y1);
        arr[k+1]=ymax;
        arr[k+2]=x2;
        arr[k+3]=y2;
        k+=4;
    }
    if(y1 <= ymax && y2 > ymax)
    {
        arr[k]=x1+m*(ymax-y1);
        arr[k+1]=ymax;
        k+=2;
    }
}
```

```
}
```

```
void clipr(float x1,float y1,float x2,float y2)
```

```
{
```

```
    if(x2-x1)
```

```
        m=(y2-y1)/(x2-x1);
```

```
    else
```

```
        m=100000;
```

```
    if(x1 <= xmax && x2 <= xmax)
```

```
    {
```

```
        arr[k]=x2;
```

```
        arr[k+1]=y2;
```

```
        k+=2;
```

```
    }
```

```
    if(x1 > xmax && x2 <= xmax)
```

```
    {
```

```
        arr[k]=xmax;
```

```
        arr[k+1]=y1+m*(xmax-x1);
```

```
        arr[k+2]=x2;
```

```
        arr[k+3]=y2;
```

```
        k+=4;
```

```
    }
```

```
    if(x1 <= xmax && x2 > xmax)
```

```
    {
```

```
        arr[k]=xmax;
```

```
        arr[k+1]=y1+m*(xmax-x1);
```

```
        k+=2;
```

```
    }
```

```
}
```

```
void clipb(float x1,float y1,float x2,float y2)
```

```

{
    if(y2-y1)
        m=(x2-x1)/(y2-y1);
    else
        m=100000;
    if(y1 >= ymin && y2 >= ymin)
    {
        arr[k]=x2;
        arr[k+1]=y2;
        k+=2;
    }
    if(y1 < ymin && y2 >= ymin)
    {
        arr[k]=x1+m*(ymin-y1);
        arr[k+1]=ymin;
        arr[k+2]=x2;
        arr[k+3]=y2;
        k+=4;
    }
    if(y1 >= ymin && y2 < ymin)
    {
        arr[k]=x1+m*(ymin-y1);
        arr[k+1]=ymin;
        k+=2;
    }
}

```

```

void main()
{
    int gdriver=DETECT,gmode,n,poly[20];
    float xi,yi,xf,yf,polyy[20];

```

```

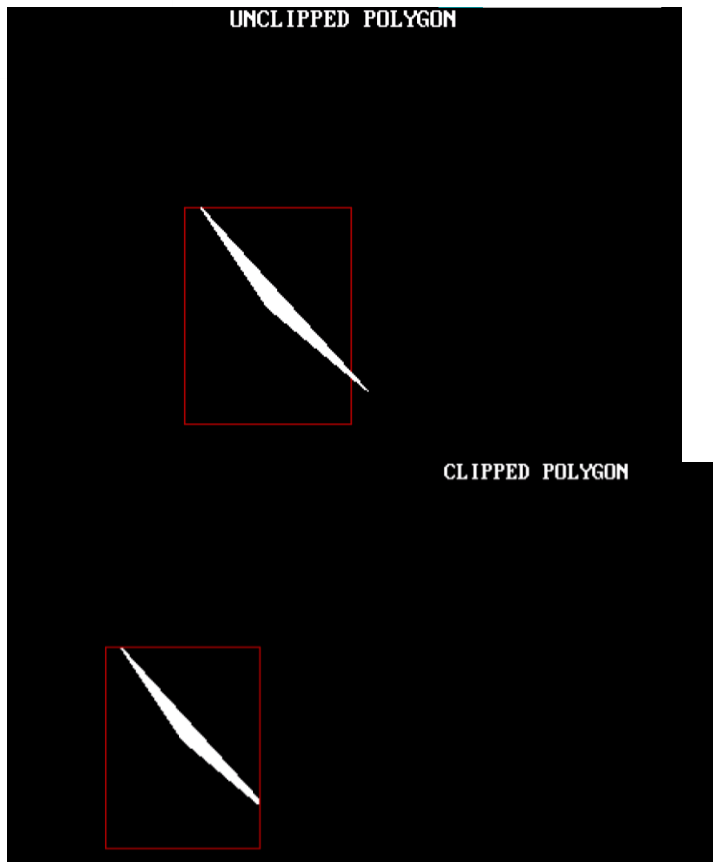
clrscr();
cout<<"Coordinates of rectangular clip window :\\nxmin,ymin
:";
cin>>xmin>>ymin;
cout<<"xmax,ymax      :";
cin>>xmax>>ymin;
cout<<"\\n\\nPolygon to be clipped :\\nNumber of sides      :";
cin>>n;
cout<<"Enter the coordinates :";
for(int i=0;i < 2*n;i++)
    cin>>polyy[i];
polyy[i]=polyy[0];
polyy[i+1]=polyy[1];
for(i=0;i < 2*n+2;i++)
    poly[i]=round(polyy[i]);
initgraph(&gdriver,&gmode,"C:/TURBOC3/BGI");
setcolor(RED);
rectangle(xmin,ymin,xmax,ymin);
cout<<"\\t\\tUNCLIPPED POLYGON";
setcolor(WHITE);
fillpoly(n,poly);
    getch();
cleardevice();
k=0;
for(i=0;i < 2*n;i+=2)
    clipl(polyy[i],polyy[i+1],polyy[i+2],polyy[i+3]);
n=k/2;
for(i=0;i < k;i++)
    polyy[i]=arr[i];
polyy[i]=polyy[0];
polyy[i+1]=polyy[1];

```

```

k=0;
for(i=0;i < 2*n;i+=2)
    clipt(polyyy[i],polyyy[i+1],polyyy[i+2],polyyy[i+3]);
n=k/2;
for(i=0;i < k;i++)
    polyyy[i]=arr[i];
polyyy[i]=polyyy[0];
polyyy[i+1]=polyyy[1];
k=0;
for(i=0;i < 2*n;i+=2)
    clipr(polyyy[i],polyyy[i+1],polyyy[i+2],polyyy[i+3]);
n=k/2;
for(i=0;i < k;i++)
    polyyy[i]=arr[i];
polyyy[i]=polyyy[0];
polyyy[i+1]=polyyy[1];
k=0;
for(i=0;i < 2*n;i+=2)
    clipb(polyyy[i],polyyy[i+1],polyyy[i+2],polyyy[i+3]);
for(i=0;i < k;i++)
    poly[i]=round(arr[i]);
if(k)
    fillpoly(k/2,poly);
setcolor(RED);
rectangle(xmin,ymax,xmax,ymin);
cout<<"\tCLIPPED POLYGON";
getch();
closegraph();
}

```

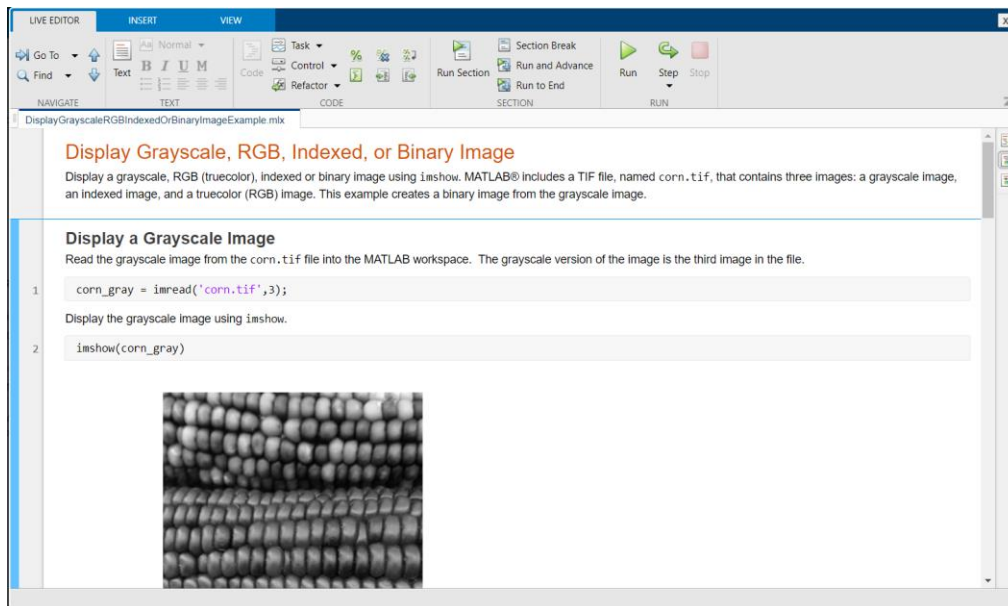




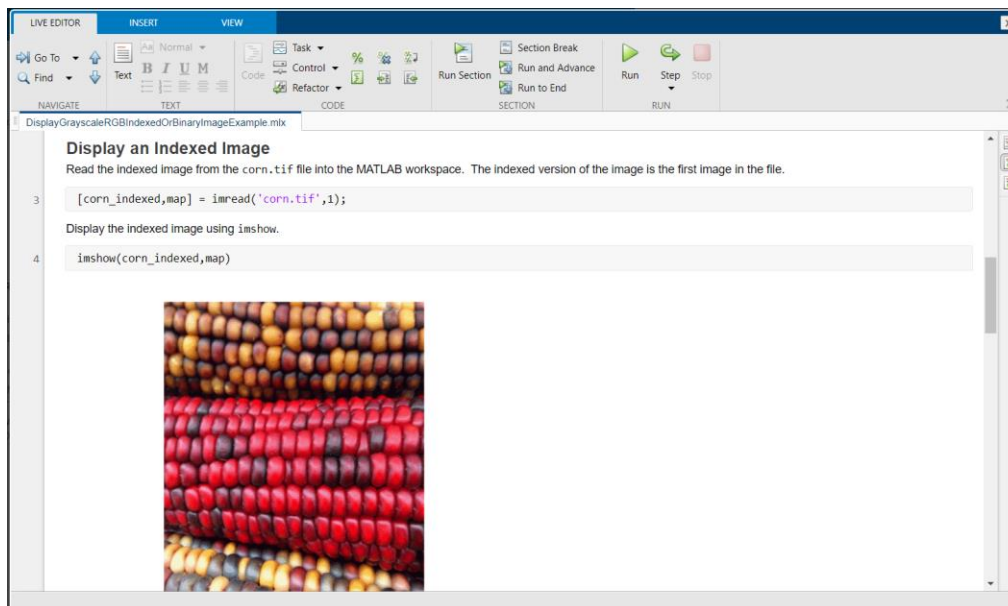
### Q-9. Perform the following tasks using MATLAB commands.

- Read the grayscale and color image.
- Display images on the computer monitor
- Write images in your destination folder.

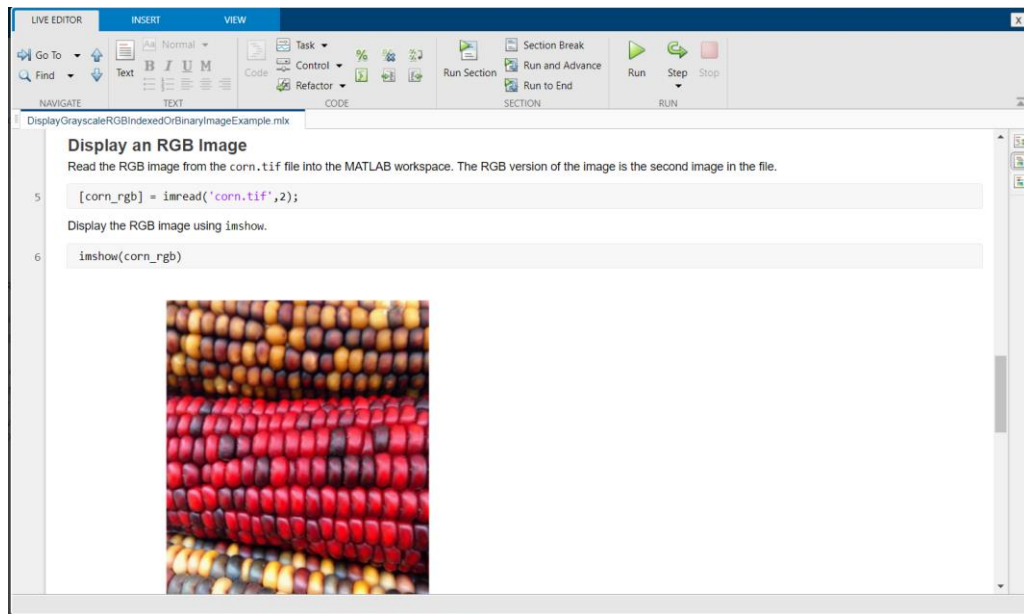
#### ➤ Gray Scale Image



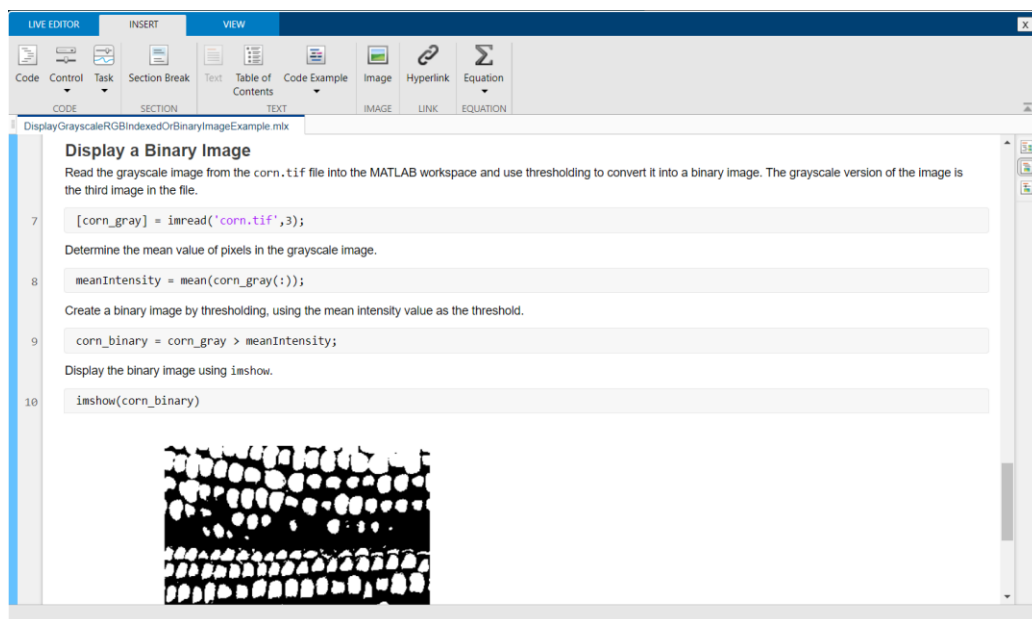
#### ➤ Indexed Image



## ➤ RGB image



## ➤ Binary Image



### Q-10) Generate the complement image using MATLAB.

#### ➤ Complement

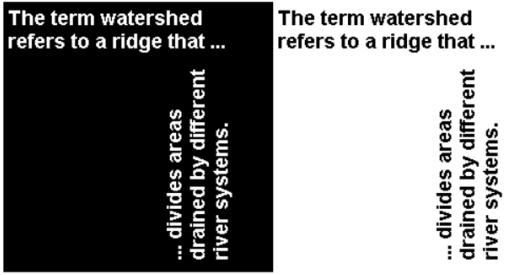
ReverseBlackAndWhiteInBinaryImageExample.mlx

#### Reverse Black and White in a Binary Image

```

1  bw = imread('text.png');
2  bw2 = imcomplement(bw);
3  imshowpair(bw,bw2,'montage')

```




ReverseBlackAndWhiteInBinaryImageExample.mlx    CreateComplementOfIntensityImageExample.mlx

#### Create the Complement of an Intensity Image

```

1  I = imread('cameraman.tif');
2  J = imcomplement(I);
3  imshowpair(I,J,'montage')

```



## Extra Program:

### 1) Car Moving

#### Code:

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

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

    /* initialize graphic mode */
    initgraph(&gd, &gm, "C:\\\\TURBOC3\\\\BGI");
    /* maximum pixel in horizontal axis */
    maxx = getmaxx();
    /* mid pixel in vertical axis */
    midy = getmaxy()/2;

    for (i=0; i < maxx-150; i=i+5) {
        /* clears screen */
        cleardevice();

        /* draw a white road */
        setcolor(WHITE);
        line(0, midy + 37, maxx, midy + 37);

        /* Draw Car */
        setcolor(YELLOW);
        setfillstyle(SOLID_FILL, RED);

        line(i, midy + 23, i, midy);
        line(i, midy, 40 + i, midy - 20);
        line(40 + i, midy - 20, 80 + i, midy - 20);
        line(80 + i, midy - 20, 100 + i, midy);
        line(100 + i, midy, 120 + i, midy);
        line(120 + i, midy, 120 + i, midy + 23);
        line(0 + i, midy + 23, 18 + i, midy + 23);
```

```

arc(30 + i, midy + 23, 0, 180, 12);
line(42 + i, midy + 23, 78 + i, midy + 23);
arc(90 + i, midy + 23, 0, 180, 12);
line(102 + i, midy + 23, 120 + i, midy + 23);
line(28 + i, midy, 43 + i, midy - 15);
line(43 + i, midy - 15, 57 + i, midy - 15);
line(57 + i, midy - 15, 57 + i, midy);
line(57 + i, midy, 28 + i, midy);
line(62 + i, midy - 15, 77 + i, midy - 15);
line(77 + i, midy - 15, 92 + i, midy);
line(92 + i, midy, 62 + i, midy);
line(62 + i, midy, 62 + i, midy - 15);
floodfill(5 + i, midy + 22, YELLOW);
setcolor(BLUE);
setfillstyle(SOLID_FILL, DARKGRAY);
/* Draw Wheels */
circle(30 + i, midy + 25, 9);
circle(90 + i, midy + 25, 9);
floodfill(30 + i, midy + 25, BLUE);
floodfill(90 + i, midy + 25, BLUE);
/* Add delay of 0.1 milli seconds */
delay(100);
}

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

```

Output:



## 2)Bouncing Ball

### Code:

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

int main() {
    int gd = DETECT, gm;
    int i, x, y, flag=0;
    initgraph(&gd, &gm, "C:\\TURBOC3\\BGI");

    /* get mid positions in x and y-axis */
    x = getmaxx()/2;
    y = 30;

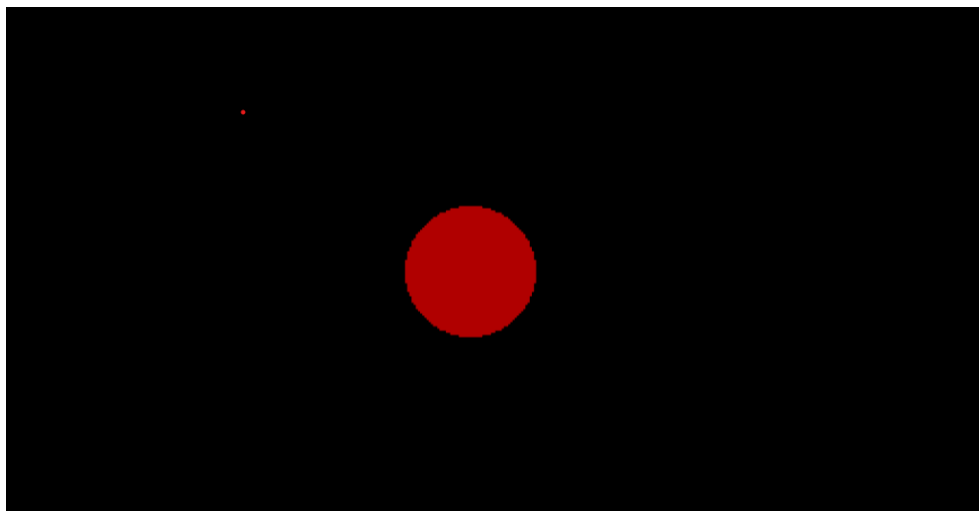
    while (!kbhit()) {
        if(y >= getmaxy()-30 || y <= 30)
            flag = !flag;
        /* draws the gray board */
        setcolor(RED);
        setfillstyle(SOLID_FILL, RED);
        circle(x, y, 30);
        floodfill(x, y, RED);
```

```
/* delay for 50 milli seconds */  
delay(50);
```

```
/* clears screen */  
cleardevice();  
if(flag){  
    y = y + 5;  
} else {  
    y = y - 5;  
}  
}
```

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

Output:



### 3) Home

#### Code:

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

int main(){
    int gd = DETECT,gm;
    initgraph(&gd, &gm, "C:\\\\TURBOC3\\\\BGI");
    /* Draw Hut */
    setcolor(WHITE);
    rectangle(150,180,250,300);
    rectangle(250,180,420,300);
    rectangle(180,250,220,300);

    line(200,100,150,180);
    line(200,100,250,180);
    line(200,100,370,100);
    line(370,100,420,180);

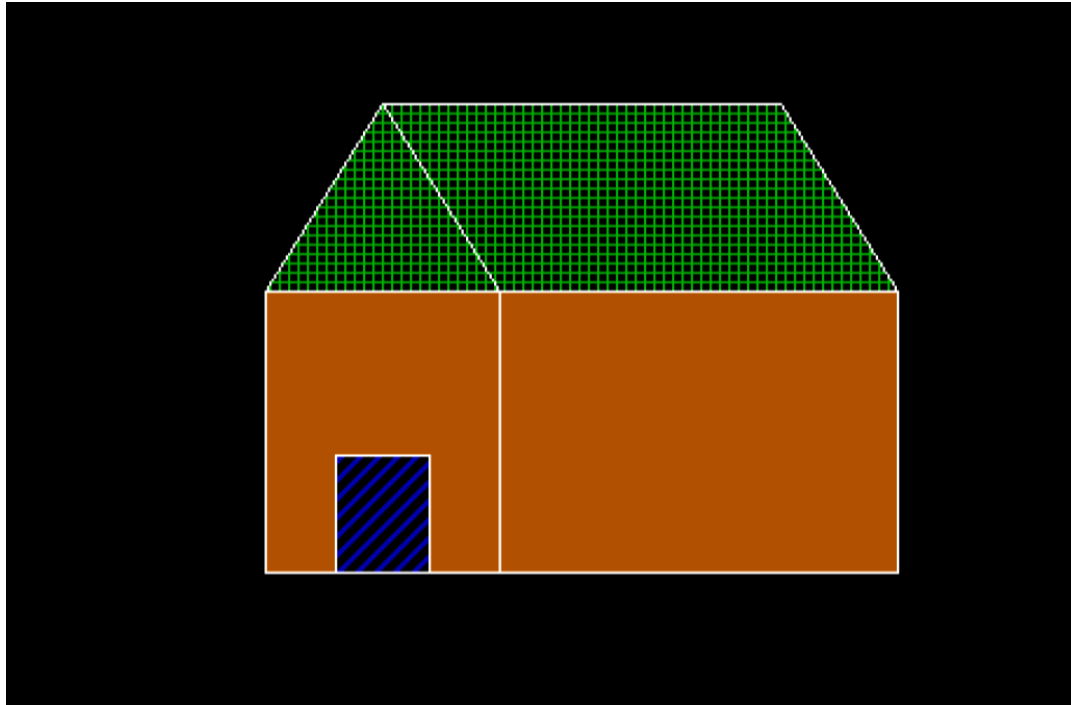
    /* Fill colours */
    setfillstyle(SOLID_FILL, BROWN);
    floodfill(152, 182, WHITE);
    floodfill(252, 182, WHITE);
    setfillstyle(SLASH_FILL, BLUE);
    floodfill(182, 252, WHITE);
    setfillstyle(HATCH_FILL, GREEN);
    floodfill(200, 105, WHITE);
    floodfill(210, 105, WHITE);

    getch();
}
```



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

Output:



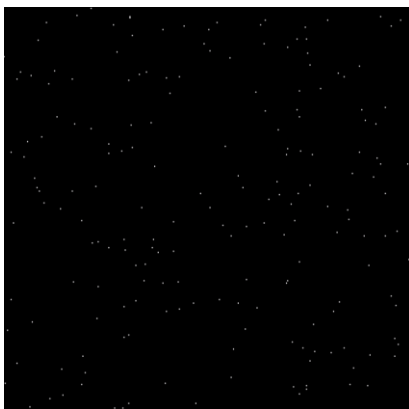
#### 4) Star on sky animation

##### Code:

```
#include <conio.h>  
#include <graphics.h>  
#include <dos.h>  
#include <stdlib.h>  
  
int main() {  
    int gd = DETECT, gm;  
    int i, x, y;  
    initgraph(&gd, &gm, "C:\\TURBOC3\\BGI");
```

```
while (!kbhit()) {  
    /* color 500 random pixels on screen */  
    for(i=0; i<=500; i++) {  
        x=rand()%getmaxx();  
        y=rand()%getmaxy();  
        putpixel(x,y,15);  
    }  
    delay(500);  
  
    /* clears screen */  
    cleardevice();  
}  
  
getch();  
closegraph();  
return 0;  
}
```

Output:



## 5) Pie Chart

### Code:

```
#include<graphics.h>

#include<conio.h>


int main() {

    int gd = DETECT, gm, x, y;

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


    settextstyle(BOLD_FONT,HORIZ_DIR,2);
    outtextxy(220,10,"PIE CHART");

    /* Setting cordinate of center of circle */
    x = getmaxx()/2;
    y = getmaxy()/2;


    settextstyle(SANS_SERIF_FONT,HORIZ_DIR,1);
    setfillstyle(SOLID_FILL, RED);
    pieslice(x, y, 0, 60, 120);
    outtextxy(x + 140, y - 70, "FOOD");

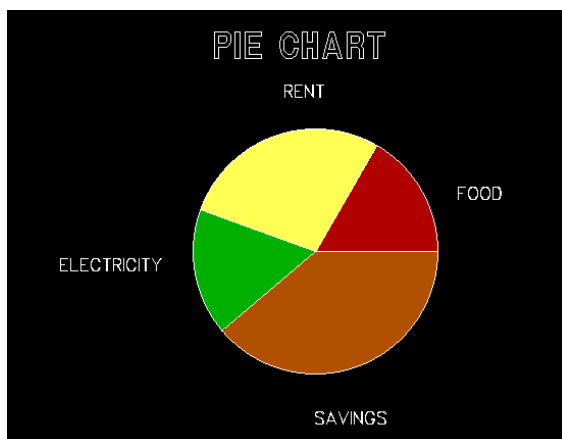

    setfillstyle(SOLID_FILL, YELLOW);
    pieslice(x, y, 60, 160, 120);
    outtextxy(x - 30, y - 170, "RENT");
```

```
setfillstyle(SOLID_FILL, GREEN);  
pieslice(x, y, 160, 220, 120);  
outtextxy(x - 250, y, "ELECTRICITY");
```

```
setfillstyle(SOLID_FILL, BROWN);  
pieslice(x, y, 220, 360, 120);  
outtextxy(x, y + 150, "SAVINGS");
```

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

```
}
```



## 6) Real Time watch

### Code:

```
#include <conio.h>

#include <graphics.h>

#include <time.h>

#include <dos.h>

#include <string.h>


int main() {

    int gd = DETECT, gm;

    int midx, midy;

    long current_time;

    char timeStr[256];


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


    /* mid pixel in horizontal and vertical axis */

    midx = getmaxx() / 2;

    midy = getmaxy() / 2;


    while (!kbhit()) {

        cleardevice();

        setcolor(WHITE);

        setfillstyle(SOLID_FILL, WHITE);

        rectangle(midx - 250, midy - 40, midx + 250, midy + 40);

        floodfill(midx, midy, WHITE);

        /* Get Current epoch time in seconds */

        current_time = time(NULL);

        /* store the date and time in string */

        strcpy(timeStr, ctime(&current_time));
```

```
setcolor(RED);  
settextjustify(CENTER_TEXT, CENTER_TEXT);  
settextstyle(SANS_SERIF_FONT, HORIZ_DIR, 4);  
  
moveto(midx, midy);  
/* print current time */  
outtext(timeStr);  
/* Add delay of 1000 milliseconds(1 second) */  
delay(1000);  
}  
  
getch();  
closegraph();  
return 0;  
}
```

Output:



Mon Nov 02 16:30:36 2020

## 7) Mickey Mouse moving

Output:

