**PROGRAMS:**

* WAP in C to draw a solid line using Bresenham Algorithm.

#include<stdio.h>

#include<conio.h>

#include<graphics.h>

#include<math.h>

void main()

{

int gd=DETECT,gm,i;

float x1,y1,x2,y2,dx,dy,xinc,yinc,x,y,e;

initgraph(&gd,&gm,"");

cleardevice();

printf("Enter the co-ordinates of point1: ");

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

printf("Enter the co-ordinates of point2: ");

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

dx=abs(x2-x1);

dy=abs(y2-y1);

if(x1<x2)

xinc=1;

else

xinc=-1;

if(y1<y2)

yinc=1;

else

yinc=-1;

x=x1;

y=y1;

putpixel((int)x,(int)y,WHITE);

if(dx>=dy)

{

e=2\*dy-dx;

while(x!=x2)

{

if(e<0)

e=e+2\*dy;

else

{

e=e+2\*(dy-dx);

y=y+yinc;

}

x=x+xinc;

putpixel((int)x,(int)y,WHITE);

}

}

else

{

e=2\*dx-dy;

while(y!=y2)

{

if(e<0)

e=e+2\*dx;

else

{

e=e+2\*(dx-dy);

x=x+xinc;

}

y=y+yinc;

putpixel((int)x,(int)y,WHITE);

}

}

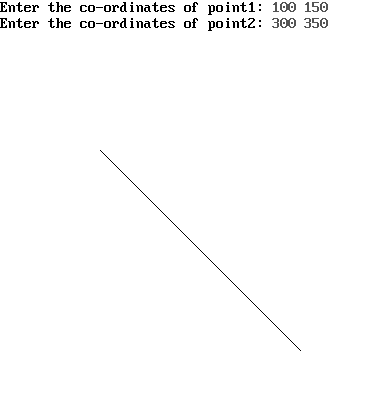
getch();

closegraph();

restorecrtmode();

}

**OUTPUT:**



* WAP in C to draw a dotted line using Bresenham Algorithm.

#include<stdio.h>

#include<conio.h>

#include<graphics.h>

#include<math.h>

void main()

{

int gd=DETECT,gm,i,count=0;

float x1,y1,x2,y2,dx,dy,xinc,yinc,x,y,e;

initgraph(&gd,&gm,"");

cleardevice();

printf("Enter the co-ordinates of point1: ");

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

printf("Enter the co-ordinates of point2: ");

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

dx=abs(x2-x1);

dy=abs(y2-y1);

if(x1<x2)

xinc=1;

else

xinc=-1;

if(y1<y2)

yinc=1;

else

yinc=-1;

x=x1;

y=y1;

putpixel((int)x,(int)y,WHITE);

if(dx>=dy)

{

e=2\*dy-dx;

while(x!=x2)

{

if(e<0)

e=e+2\*dy;

else

{

e=e+2\*(dy-dx);

y=y+yinc;

}

x=x+xinc;

count++;

if(count%3==0)

putpixel((int)x,(int)y,WHITE);

}

}

else

{

e=2\*dx-dy;

while(y!=y2)

{

if(e<0)

e=e+2\*dx;

else

{

e=e+2\*(dx-dy);

x=x+xinc;

}

y=y+yinc;

count++;

if(count%3==0)

putpixel((int)x,(int)y,WHITE);

}

}

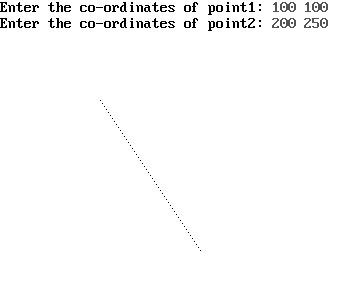
getch();

closegraph();

restorecrtmode();

}

**OUTPUT:**



* WAP in C to draw a dashed line using Bresenham Algorithm.

#include<stdio.h>

#include<conio.h>

#include<graphics.h>

#include<math.h>

void main()

{

int gd=DETECT,gm,i,count=0;

float x1,y1,x2,y2,dx,dy,xinc,yinc,x,y,e;

initgraph(&gd,&gm,"");

cleardevice();

printf("Enter the co-ordinates of point1: ");

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

printf("Enter the co-ordinates of point2: ");

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

dx=abs(x2-x1);

dy=abs(y2-y1);

if(x1<x2)

xinc=1;

else

xinc=-1;

if(y1<y2)

yinc=1;

else

yinc=-1;

x=x1;

y=y1;

putpixel((int)x,(int)y,WHITE);

if(dx>=dy)

{

e=2\*dy-dx;

while(x!=x2)

{

if(e<0)

e=e+2\*dy;

else

{

e=e+2\*(dy-dx);

y=y+yinc;

}

x=x+xinc;

count++;

if(count%3!=0)

putpixel((int)x,(int)y,WHITE);

}

}

else

{

e=2\*dx-dy;

while(y!=y2)

{

if(e<0)

e=e+2\*dx;

else

{

e=e+2\*(dx-dy);

x=x+xinc;

}

y=y+yinc;

count++;

if(count%3!=0)

putpixel((int)x,(int)y,WHITE);

}

}

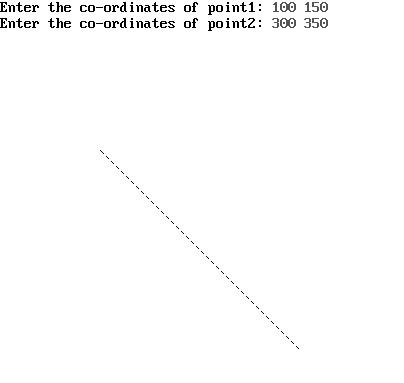
getch();

closegraph();

restorecrtmode();

}

**OUTPUT:**



* WAP in C to draw a dotted line using Bresenham Algorithm.

#include<stdio.h>

#include<conio.h>

#include<graphics.h>

#include<math.h>

void main()

{

int gd=DETECT,gm,i;

float x1,y1,x2,y2,dx,dy,xinc,yinc,x,y,e,thickness,wx,wy;

initgraph(&gd,&gm,"");

cleardevice();

printf("Enter the co-ordinates of point1: ");

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

printf("Enter the co-ordinates of point2: ");

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

printf("Enter thickness: ");

scanf("%f",&thickness);

dx=abs(x2-x1);

dy=abs(y2-y1);

if(x1<x2)

xinc=1;

else

xinc=-1;

if(y1<y2)

yinc=1;

else

yinc=-1;

x=x1;

y=y1;

wx=(thickness-1)\*sqrt(pow((x2-x1),2)+pow((y2-y1),2))/(2\*abs(y2-y1));

wy=(thickness-1)\*sqrt(pow((x2-x1),2)+pow((y2-y1),2))/(2\*abs(x2-x1));

putpixel((int)x,(int)y,WHITE);

if(dx<=dy)

{

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

{

putpixel((int)x+i,(int)y,WHITE);

putpixel((int)x-i,(int)y,WHITE);

}

}

else

{

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

{

putpixel((int)x,(int)y+i,WHITE);

putpixel((int)x,(int)y-i,WHITE);

}

}

if(dx>=dy)

{

e=2\*dy-dx;

while(x!=x2)

{

if(e<0)

e=e+2\*dy;

else

{

e=e+2\*(dy-dx);

y=y+yinc;

}

x=x+xinc;

putpixel((int)x,(int)y,WHITE);

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

{

putpixel((int)x,(int)y+i,WHITE);

putpixel((int)x,(int)y-i,WHITE);

}

}

}

else

{

e=2\*dx-dy;

while(y!=y2)

{

if(e<0)

e=e+2\*dx;

else

{

e=e+2\*(dx-dy);

x=x+xinc;

}

y=y+yinc;

putpixel((int)x,(int)y,WHITE);

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

{

putpixel((int)x+i,(int)y,WHITE);

putpixel((int)x-i,(int)y,WHITE);

}

}

}

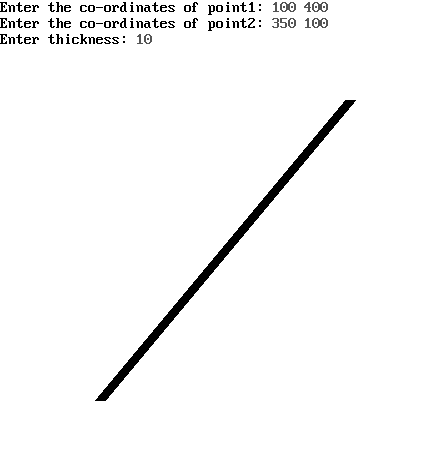
getch();

closegraph();

restorecrtmode();

}

**OUTPUT:**



* WAP in C to draw an ellipse using Midpoint Ellipse Algorithm.

#include<stdio.h>

#include<conio.h>

#include<graphics.h>

#include<math.h>

void drawEllipse(float xc, float yc, float a, float b);

void main()

{

int gd=DETECT,gm;

float a,b,xc,yc;

initgraph(&gd,&gm,"");

cleardevice();

printf("Enter the co-ordinates of centre: ");

scanf("%f %f",&xc,&yc);

printf("Enter the values of a and b: ");

scanf("%f %f",&a,&b);

drawEllipse(xc,yc,a,b);

getch();

closegraph();

restorecrtmode();

}

void drawEllipse(float xc, float yc, float a, float b)

{

int i

float x,y,p;

x=0;

y=b;

p=b\*b-a\*a\*b+(a\*a/4);

do

{

delay(100);

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

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

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

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

if(p<0)

{

x=x+1;

p=p+2\*b\*b\*x+b\*b;

}

else

{

x=x+1;

y=y-1;

p=p+2\*b\*b\*x-2\*a\*a\*y+b\*b;

}

}while(b\*b\*x<=a\*a\*y);

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

do

{

delay(100);

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

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

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

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

if(p>0)

{

y=y-1;

p=p-2\*a\*a\*y+a\*a;

}

else

{

x=x+1;

y=y-1;

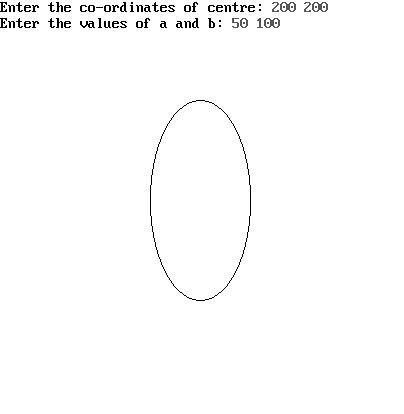
p=p-2\*a\*a\*y+2\*b\*b\*x+a\*a;

}

}while(y!=0);

}

**OUTPUT:**



* WAP in C to draw a circle using Midpoint Circle Algorithm.

#include<stdio.h>

#include<conio.h>

#include<graphics.h>

#include<math.h>

void drawCircle(float xc, float yc, float r);

void main()

{

int gd=DETECT,gm,i;

float r,xc,yc;

initgraph(&gd,&gm,"");

cleardevice();

printf("Enter the co-ordinates of centre: ");

scanf("%f %f",&xc,&yc);

printf("Enter the radius ");

scanf("%f",&r);

drawCircle(xc,yc,r);

getch();

closegraph();

restorecrtmode();

}

void drawCircle(float xc, float yc, float r)

{

float x,y,d;

d=(5/4)-r;

x=0;

y=r;

do

{

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

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

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

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

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

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

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

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

if(d<0)

d=d+2\*x+3;

else

{

d=d+2\*(x-y)+5;

y=y-1;

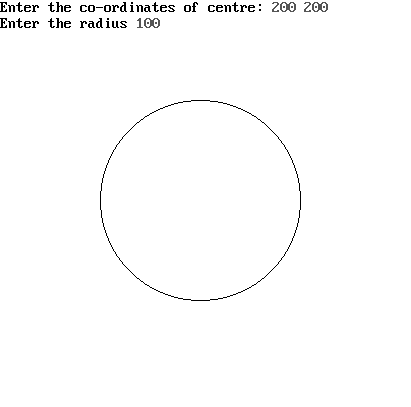
}

x=x+1;

}while(x<=y);

}

**OUTPUT:**



* WAP in C to implement Boundary Fill Algorithm..

#include<stdio.h>

#include<conio.h>

#include<graphics.h>

void bFill\_four(int x, int y, int bcolor, int fcolor);

void bFill\_eight(int x, int y, int bcolor, int fcolor);

void drawPolygon();

void main()

{

int choice,gd=DETECT,gm;

initgraph(&gd,&gm,"");

cleardevice();

drawPolygon();

getch();

closegraph();

restorecrtmode();

}

void drawPolygon()

{

int choice,bcolor,fcolor,n,x[10],y[10],seedx,seedy,i;

printf("Enter the number of co-ordinates: ");

scanf("%d",&n);

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

{

printf("Enter the co-ordinates of point %d: ",i+1);

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

}

x[n]=x[0];

y[n]=y[0];

printf("Enter Boundary Color: ");

scanf("%d",&bcolor);

setcolor(bcolor);

printf("Enter Fill Color: ");

scanf("%d",&fcolor);

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

{

line(x[i],y[i],x[i+1],y[i+1]);

}

printf("1. 4-Connected\n");

printf("2. 8-Connected\n");

printf("Enter your choice: ");

scanf("%d",&choice);

switch(choice)

{

case 1 : printf("Enter seed point: ");

scanf("%d %d",&seedx,&seedy);

bFill\_four(seedx,seedy,bcolor,fcolor);

break;

case 2 : printf("Enter seed point: ");

scanf("%d %d",&seedx,&seedy);

bFill\_eight(seedx,seedy,bcolor,fcolor);

break;

default : printf("Invalid choice");

}

}

void bFill\_four(int x, int y, int bcolor, int fcolor)

{

int current;

current=getpixel(x,y);

if(current!=bcolor && current!=fcolor)

{

delay(1);

putpixel(x,y,fcolor);

bFill\_four(x+1,y,bcolor,fcolor);

bFill\_four(x,y+1,bcolor,fcolor);

bFill\_four(x,y-1,bcolor,fcolor);

bFill\_four(x-1,y,bcolor,fcolor);

}

}

void bFill\_eight(int x, int y, int bcolor, int fcolor)

{

int current;

current=getpixel(x,y);

if(current!=bcolor && current!=fcolor)

{

delay(1);

putpixel(x,y,fcolor);

bFill\_eight(x+1,y,bcolor,fcolor);

bFill\_eight(x-1,y,bcolor,fcolor);

bFill\_eight(x,y+1,bcolor,fcolor);

bFill\_eight(x,y-1,bcolor,fcolor);

bFill\_eight(x+1,y-1,bcolor,fcolor);

bFill\_eight(x+1,y+1,bcolor,fcolor);

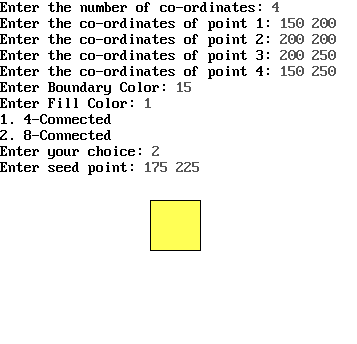
bFill\_eight(x-1,y-1,bcolor,fcolor);

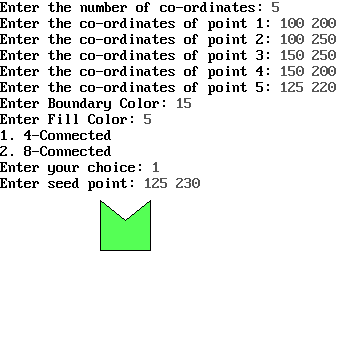
bFill\_eight(x-1,y+1,bcolor,fcolor);

}

}

**OUTPUT:**





* WAP in C to implement Flood Fill Algorithm.

#include<stdio.h>

#include<conio.h>

#include<graphics.h>

void fFill\_four(int x, int y, int ocolor, int fcolor);

void fFill\_eight(int x, int y, int ocolor, int fcolor);

void drawPolygon();

void main()

{

int choice,gd=DETECT,gm;

initgraph(&gd,&gm,"");

cleardevice();

drawPolygon();

getch();

closegraph();

restorecrtmode();

}

void drawPolygon()

{

int choice,ocolor=0,fcolor,n,x[10],y[10],seedx,seedy,i;

printf("Enter the number of co-ordinates: ");

scanf("%d",&n);

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

{

printf("Enter the co-ordinates of point %d: ",i+1);

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

}

x[n]=x[0];

y[n]=y[0];

printf("Enter Fill Color: ");

scanf("%d",&fcolor);

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

{

line(x[i],y[i],x[i+1],y[i+1]);

}

printf("1. 4-Connected\n");

printf("2. 8-Connected\n");

printf("Enter your choice: ");

scanf("%d",&choice);

switch(choice)

{

case 1 : printf("Enter seed point: ");

scanf("%d %d",&seedx,&seedy);

fFill\_four(seedx,seedy,ocolor,fcolor);

break;

case 2 : printf("Enter seed point: ");

scanf("%d %d",&seedx,&seedy);

fFill\_eight(seedx,seedy,ocolor,fcolor);

break;

default : printf("Invalid choice");

}

}

void fFill\_four(int x, int y, int ocolor, int fcolor)

{

int current;

current=getpixel(x,y);

if(current==ocolor)

{

delay(1);

putpixel(x,y,fcolor);

fFill\_four(x+1,y,ocolor,fcolor);

fFill\_four(x,y+1,ocolor,fcolor);

fFill\_four(x,y-1,ocolor,fcolor);

fFill\_four(x-1,y,ocolor,fcolor);

}

}

void fFill\_eight(int x, int y, int ocolor, int fcolor)

{

int current;

current=getpixel(x,y);

if(current==ocolor)

{

delay(1);

putpixel(x,y,fcolor);

fFill\_eight(x+1,y,ocolor,fcolor);

fFill\_eight(x-1,y,ocolor,fcolor);

fFill\_eight(x,y+1,ocolor,fcolor);

fFill\_eight(x,y-1,ocolor,fcolor);

fFill\_eight(x+1,y-1,ocolor,fcolor);

fFill\_eight(x+1,y+1,ocolor,fcolor);

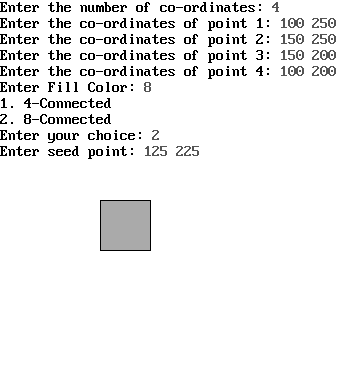
fFill\_eight(x-1,y-1,ocolor,fcolor);

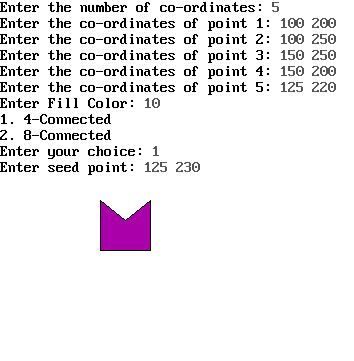
fFill\_eight(x-1,y+1,ocolor,fcolor);

}

}

**OUTPUT:**





* WAP in C to 2D Rotation about an arbitrary point.

#include<stdio.h>

#include<conio.h>

#include<graphics.h>

#include<math.h>

void matrix(float x[4][3],float y[3][3],float z[4][3]);

float a[4][3],t1[3][3],r[3][3],t2[3][3],f[4][3],f1[4][3],f2[4][3];

void main()

{

int i,j,gd=DETECT,gm;

float theta;

initgraph(&gd,&gm,"");

cleardevice();

printf("Enter the co-ordinates of the polygon:\n");

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

{

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

a[i][2]=1;

}

cleardevice();

line(0,240,640,240);

line(320,0,320,480);

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

{

line(320+a[i][0],240-a[i][1], 320+a[(i+1)%4][0],240-a[(i+1)%4][1]);

}

getch();

cleardevice();

for(i=0;i<2;i++)

{

for(j=0;j<3;j++)

{

if(i==j)

t1[i][j]=1;

else

t1[i][j]=0;

}

}

t1[2][0]=-a[0][0];

t1[2][1]=-a[0][1];

t1[2][2]=1;

matrix(a,t1,f);

printf("Enter the degree of rotation in degrees: ");

scanf("%f",&theta);

theta=theta\*3.14/180;

r[0][0]=r[1][1]=cos(theta);

r[0][1]=sin(theta);

r[1][0]=-sin(theta);

r[0][2]=r[1][2]=r[2][0]=r[2][1]=0;

r[2][2]=1;

matrix(f,r,f1);

for(i=0;i<2;i++)

{

for(j=0;j<3;j++)

{

if(i==j)

t2[i][j]=1;

else

t2[i][j]=0;

}

}

t2[2][0]=+a[0][0];

t2[2][1]=+a[0][1];

t2[2][2]=1;

matrix(f1,t2,f2);

}

void matrix(float x[4][3],float y[3][3], float z[4][3])

{

int i,j,k;

line(0,240,640,240);

line(320,0,320,480);

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

{

for(j=0;j<3;j++)

{

z[i][j]=0;

for(k=0;k<3;k++)

{

z[i][j]=(int)z[i][j]+(int)(x[i][k]\*y[k][j]);

}

}

}

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

{

line(320+z[i][0],240-z[i][1], 320+z[(i+1)%4][0],240-z[(i+1)%4][1]);

}

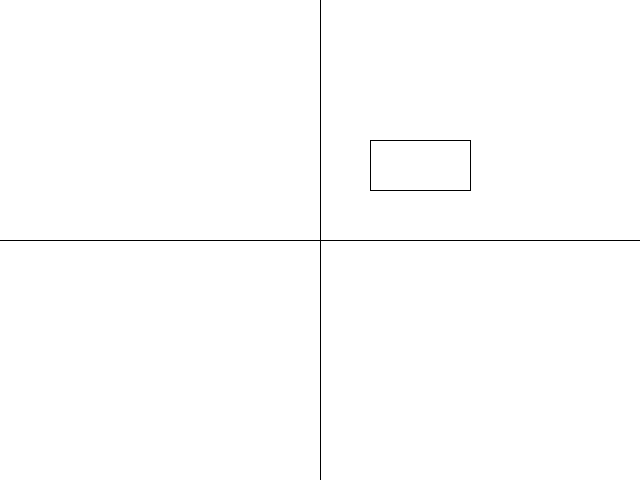
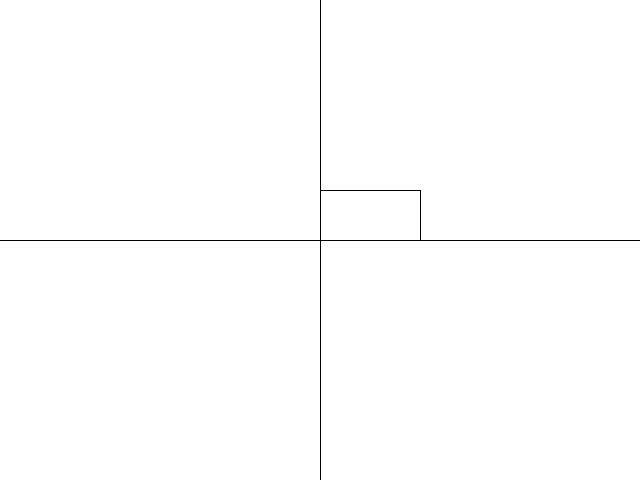
getch();

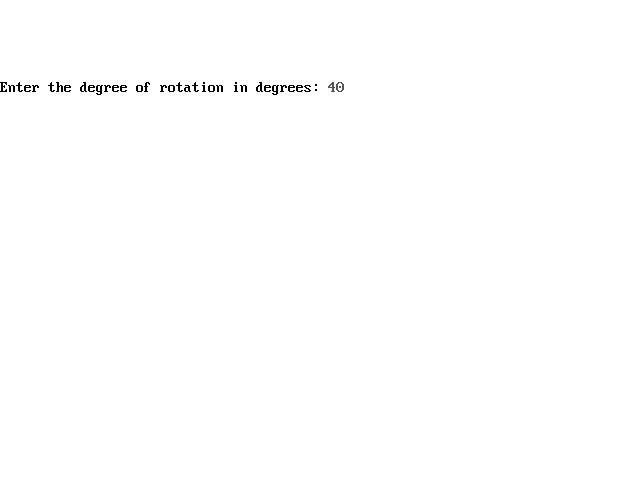
cleardevice();

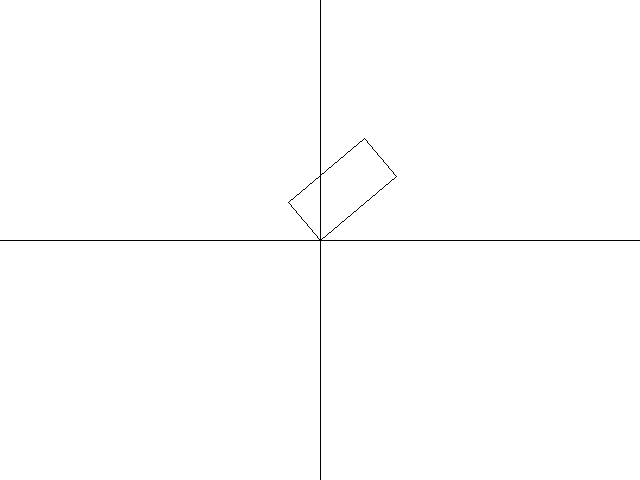
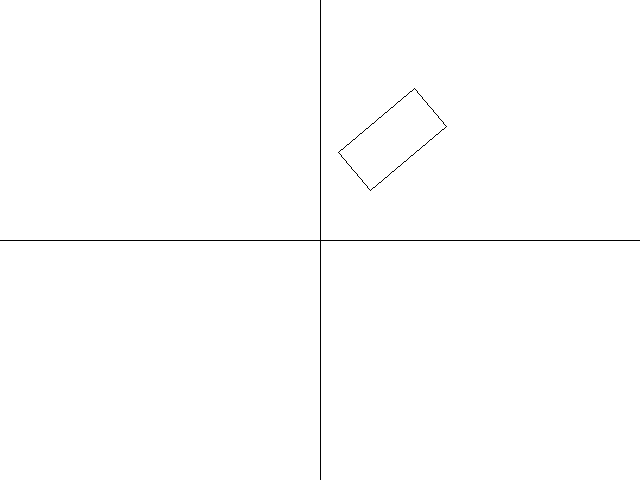
}

**OUTPUT:**





* WAP in C to implement 2D Transformations.

#include<stdio.h>

#include<conio.h>

#include<graphics.h>

#include<math.h>

void translation()

{

int tx,ty,i;

int t[5][2]= {10,10, 50,10, 50,50, 10,50, 10,10};

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

{

line(320+t[i][0],240-t[i][1], 320+t[i+1][0],240-t[i+1][1]);

}

printf("Enter translation in x and y: ");

scanf("%d %d",&tx,&ty);

for(i=0;i<5;i++)

{

t[i][0]=t[i][0]+tx;

t[i][1]=t[i][1]+ty;

}

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

{

line(320+t[i][0],240-t[i][1], 320+t[i+1][0],240-t[i+1][1]);

}

}

void rotation()

{

float theta;

int rot[5][2]={0,0, 0,50, 70,50, 70,0, 0,0};

int rot\_dash[5][2];

int i;

setcolor(BLUE);

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

{

line(320+rot[i][0],240-rot[i][1], 320+rot[i+1][0],240-rot[i+1][1]);

}

printf("Enter rotation in degrees: ");

scanf("%f",&theta);

theta=theta\*3.14/180;

for(i=0;i<5;i++)

{

rot\_dash[i][0]=(int)(rot[i][0]\*cos(theta))-(int)(rot[i][1]\*sin(theta));

rot\_dash[i][1]=(int)(rot[i][0]\*sin(theta))+(int)(rot[i][1]\*cos(theta));

}

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

{

line(320+rot\_dash[i][0],240-rot\_dash[i][1], 320+rot\_dash[i+1][0],240-rot\_dash[i+1][1]);

}

}

void scaling()

{

float scaling\_factor;

int t[5][2]={25,40, 25,-40, -25,-40, -25,40, 25,40};

int i;

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

{

line(320+t[i][0],240-t[i][1], 320+t[i+1][0],240-t[i+1][1]);

}

printf("Enter scaling factor: ");

scanf("%f",&scaling\_factor);

for(i=0;i<5;i++)

{

t[i][0]=(t[i][0]\*scaling\_factor);

t[i][1]=(t[i][1]\*scaling\_factor);

}

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

{

line(320+t[i][0],240-t[i][1], 320+t[i+1][0],240-t[i+1][1]);

}

}

void reflection()

{

int choice;

int t[5][2]= {10,10, 50,10, 50,50, 10,50, 10,10};

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

{

line(320+t[i][0],240-t[i][1], 320+t[i+1][0],240-t[i+1][1]);

}

printf("\n\*\*\*\*\*\*MAIN MENU\*\*\*\*\*\*\*\n");

printf("1. About X\n");

printf("2. About Y\n");

printf("3. About O\n");

printf("Enter your choice: ");

scanf("%d",&choice);

switch(choice)

{

case 1 : for(i=0;i<5;i++)

{

t[i][1]=-t[i][1];

}

break;

case 2 : for(i=0;i<5;i++)

{

t[i][0]=-t[i][0];

}

break;

case 3 : for(i=0;i<5;i++)

{

t[i][0]=-t[i][0];

t[i][1]=-t[i][1];

}

break;

}

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

{

line(320+t[i][0],240-t[i][1], 320+t[i+1][0],240-t[i+1][1]);

}

}

void shearing()

{

int shx,shy,i;

int t[5][2]= {10,10, 50,10, 50,50, 10,50, 10,10};

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

{

line(320+t[i][0],240-t[i][1], 320+t[i+1][0],240-t[i+1][1]);

}

printf("Enter shear in x and y: ");

scanf("%d %d",&tshx,&shy);

for(i=0;i<5;i++)

{

t[i][0]=t[i][0]+t[i][0]\*shy;

t[i][1]=t[i][1]+t[i][1]\*shx;

}

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

{

line(320+t[i][0],240-t[i][1], 320+t[i+1][0],240-t[i+1][1]);

}

}

void main()

{

int choice,gd=DETECT,gm;

initgraph(&gd,&gm,"");

cleardevice();

printf("\n\*\*\*\*\*\*MAIN MENU\*\*\*\*\*\*\*\n");

printf("1. Translation\n");

printf("2. Rotation\n");

printf("3. Scaling\n");

printf("4. Reflection\n");

printf("5. Shearing\n");

printf("Enter your choice: ");

scanf("%d",&choice);

clrscr();

cleardevice();

line(0,240,640,240);

line(320,0,320,480);

switch(choice)

{

case 1 : translation();

break;

case 2 : rotation();

break;

case 3 : scaling();

break;

case 4 : reflection();

break;

case 5 : shearing();

break;

default : printf("Invalid choice");

}

getch();

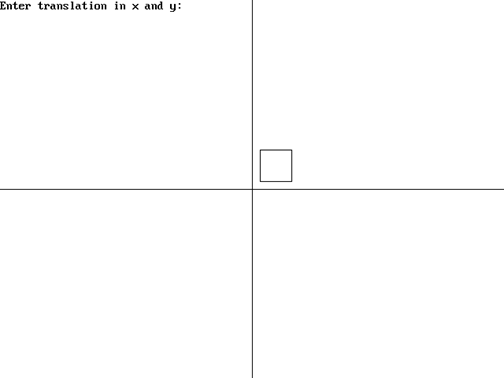
closegraph();

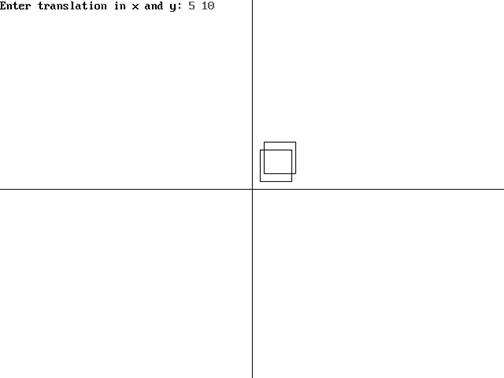
restorecrtmode();

}

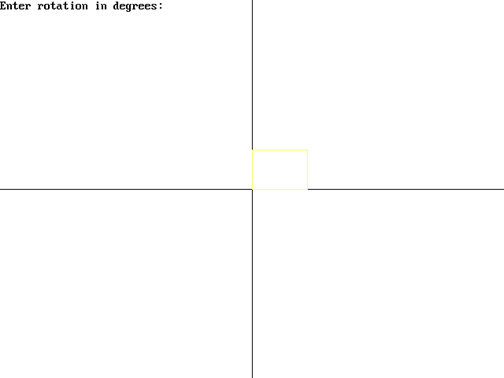
**OUTPUT:**

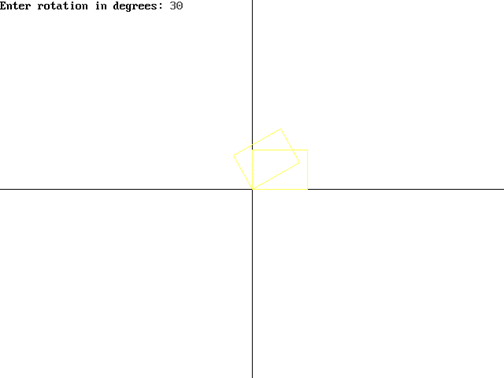




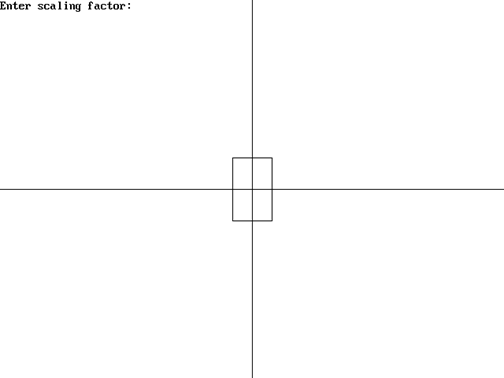


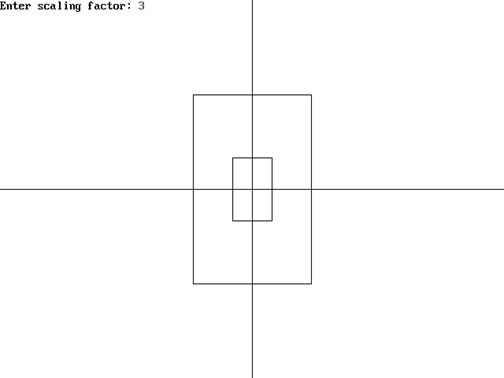




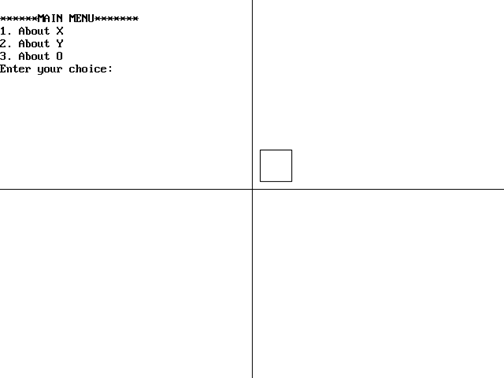


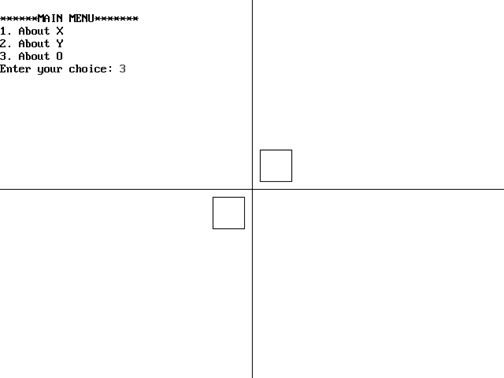


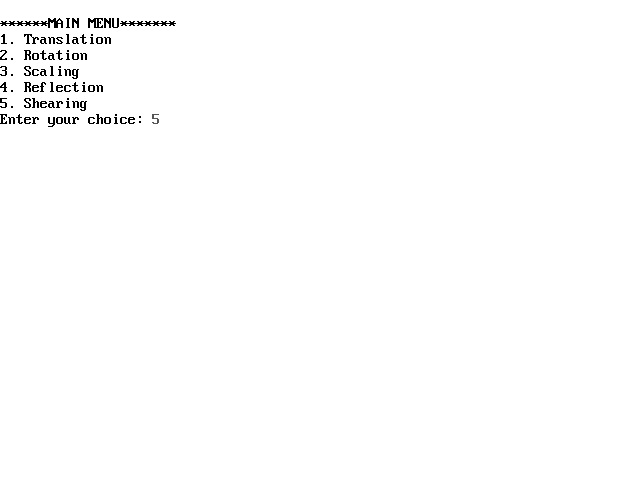


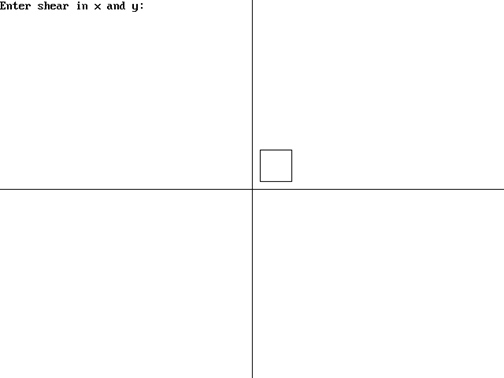


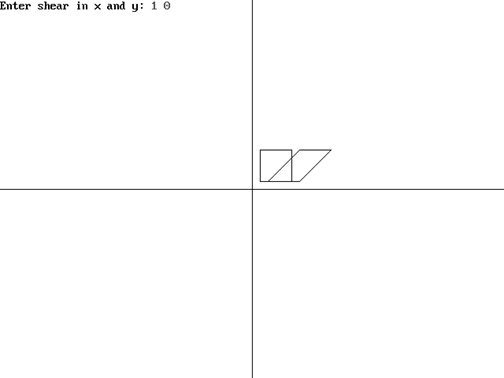












* WAP in C to implement Bezier Curve.

#include<conio.h>

#include<stdio.h>

#include<graphics.h>

void main()

{

int gd=DETECT,gm;

int x[4],y[4],px,py,i;

double t;

initgraph(&gd,&gm,"");

printf("Enter four control points of bezier curve: \n");

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

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

for(t=0.0;t<=1.0;t+=0.001)

{

px=(1-t)\*(1-t)\*(1-t)\*x[0]+3\*t\*(1-t)\*(1-t)\*x[1]+3\*t\*t\*(1-t)\*x[2]+t\*t\*t\*x[3];

py=(1-t)\*(1-t)\*(1-t)\*y[0]+3\*t\*(1-t)\*(1-t)\*y[1]+3\*t\*t\*(1-t)\*y[2]+t\*t\*t\*y[3];

putpixel(px,py,WHITE);

delay(2);

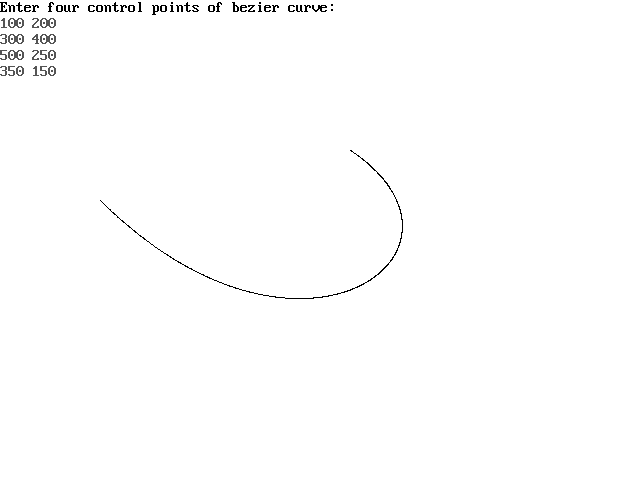
}

getch();

closegraph();

}

**OUTPUT:**



* WAP in C to implement Cohen Sutherland Algorithm.

#include<stdio.h>

#include<conio.h>

#include<graphics.h>

void main()

{

int gd=DETECT, gm;

float i,xmax,ymax,xmin,ymin,x1,y1,x2,y2,m;

float start[4],end[4],code[4];

initgraph(&gd,&gm," ");

cleardevice();

printf("Enter the bottom-left coordinate of viewport: ");

scanf("%f %f",&xmin,&ymin);

printf("Enter the top-right coordinate of viewport: ");

scanf("%f %f",&xmax,&ymax);

printf("Enter the coordinates for starting point of line: ");

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

printf("Enter the coordinates for ending point of line: ");

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

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

{

start[i]=0;

end[i]=0;

}

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

if(x1 <xmin) start[0]=1;

if(x1 >xmax) start[1]=1;

if(y1 >ymax) start[2]=1;

if(y1 <ymin) start[3]=1;

if(x2 <xmin) end[0]=1;

if(x2 >xmax) end[1]=1;

if(y2 >ymax) end[2]=1;

if(y2 <ymin) end[3]=1;

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

code[i]=start[i]&&end[i];

if((code[0]==0)&&(code[1]==0)&&(code[2]==0)&&(code[3]==0))

{

if((start[0]==0)&&(start[1]==0)&&(start[2]==0)&&(start[3]==0)&&(end[0]==0)&&(end[1]==0)&&(end[2]==0)&&(end[3]==0))

{

cleardevice();

printf("\nLine is totally visible and not a clipping candidate");

rectangle(xmin,ymin,xmax,ymax);

line(x1,y1,x2,y2);

getch();

cleardevice();

rectangle(xmin,ymin,xmax,ymax);

line(x1,y1,x2,y2);

}

else

{

cleardevice();

printf("\nLine will be partially visible");

rectangle(xmin,ymin,xmax,ymax);

line(x1,y1,x2,y2);

getch();

if((start[2]==0)&&(start[3]==1))

{

x1=x1+(ymin-y1)/m;

y1=ymin;

}

if((end[2]==0)&&(end[3]==1))

{

x2=x2+(ymin-y2)/m;

y2=ymin;

}

if((start[2]==1)&&(start[3]==0))

{

x1=x1+(ymax-y1)/m;

y1=ymax;

}

if((end[2]==1)&&(end[3]==0))

{

x2=x2+(ymax-y2)/m;

y2=ymax;

}

if((start[1]==0)&&(start[0]==1))

{

y1=y1+m\*(xmin-x1);

x1=xmin;

}

if((end[1]==0)&&(end[0]==1))

{

y2=y2+m\*(xmin-x2);

x2=xmin;

}

if((start[1]==1)&&(start[0]==0))

{

y1=y1+m\*(xmax-x1);

x1=xmax;

}

if((end[1]==1)&&(end[0]==0))

{

y2=y2+m\*(xmax-x2);

x2=xmax;

}

cleardevice();

printf("\nAfter Clippling");

rectangle(xmin,ymin,xmax,ymax);

line(x1,y1,x2,y2);

}

}

else

{

cleardevice();

printf("\nLine will be invisible");

rectangle(xmin,ymin,xmax,ymax);

line(x1,y1,x2,y2);

getch();

cleardevice();

rectangle(xmin,ymin,xmax,ymax);

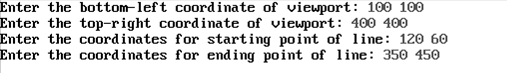
}

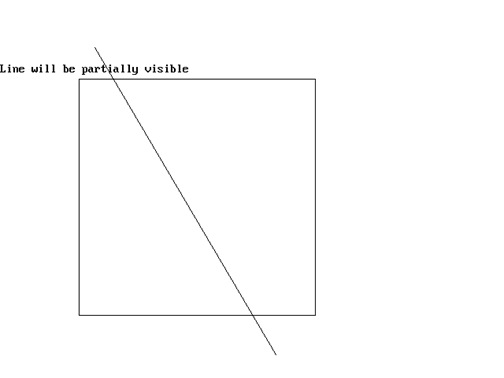
getch();

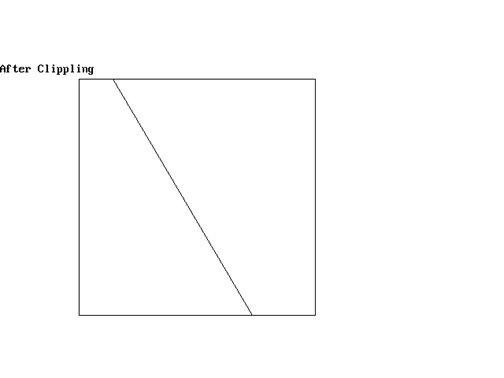
closegraph();

}

**OUTPUT:**







**Experiment No 11**

**Aim:** To draw basic diagrams using openGL.

**Theory**: OpenGL (Open Graphics Library) is a cross-platform, hardware-accelerated, language-independent, industrial standard API for producing 3D (including 2D) graphics. Modern computers have dedicated GPU (Graphics Processing Unit) with its own memory to speed up graphics rendering. OpenGL is the software interface to graphics hardware. In other words, OpenGL graphic rendering commands issued by your applications could be directed to the graphic hardware and accelerated.

We use 3 sets of libraries in our OpenGL programs:

1. Core OpenGL (GL): consists of hundreds of commands, which begin with a prefix "gl" (e.g., glColor, glVertex, glTranslate, glRotate). The Core OpenGL models an object via a set of geometric primitives such as point, line and polygon.
2. OpenGL Utility Library (GLU): built on-top of the core OpenGL to provide important utilities (such as setting camera view and projection) and more building models (such as qradric surfaces and polygon tessellation). GLU commands start with a prefix "glu" (e.g., gluLookAt, gluPerspective).

OpenGL Utilities Toolkit (GLUT): OpenGL is designed to be independent of the windowing system or operating system. GLUT is needed to interact with the Operating System (such as creating a window, handling key and mouse inputs); it also provides more building models (such as sphere and torus). GLUT commands start with a prefix of "glut" (e.g., glutCreatewindow, glutMouseFunc). GLUT is platform independent, which is built on top of platform-specific OpenGL extension such as GLX for X Window System, WGL for Microsoft Window, and AGL, CGL or Cocoa for Mac OS.  
Quoting from the [opengl.org](http://www.opengl.org/resources/libraries/glut/):

a primitive "object" can be anything from a 3D point to a line to a triangle to an n-sided polygon. By the way, many of these things I learned from numerous OpenGL books. And I have to say that helped me a lot as far as the understanding of new 3D concepts goes.

Anyway, each primitive has at least one vertex. What is a vertex exactly? Is it one of the points in a polygon or is it a stand-alone point in space? Well it can be both, depending on how you think about it. With points, vertex is just that - the point. A line has only 2 vertices - its starting point and its ending point.

With polygons, there should be more than 2 vertices since polygons are surfaces defined by more or equal to 3 vertices residing on the same plane. A triangle is, for instance, a polygon with 3 vertices. This all should be obvious to you at this point, if you're serious about 3D graphics. Note that a 3D cube cannot be considered a primitive. Generally, primitives restrict themselves to triangles. A four-sided polygon can generate a quad but that quad will still be made out of 2 polygons. Points and lines can also be considered primitives.

Now that we briefly went over primitives, the next question is how does OpenGL define primitives?. From now on I won't mention the word primitive anymore when I talk about a single primitive. I will name these objects by their respective names they are given (e.g. **points**, **polygons**, **lines**...). The main function (and probably the most used OpenGL function) is function named glVertex. This function defines a point (or a vertex) in your 3D world and it can vary from receiving 2 up to 4 coordinates. Let's take a look at these variations:

**glVertex2f(100.0f, 150.0f);** defines a point at x = 100, y = 150, z = 0; this function takes only 2 parameters, z is always 0. glVertex2f can be used in special cases and won't be used a lot unless you're working with pseudo-2D sprites or triangles and points that always have to be constrained by the depth coordinate.

**glVertex3f(100.0f, 150.0f, -25.0f);** defines a point at x = 100, y = 150, z = -25.0f; this function takes 3 parameters, defining a fully 3D point in your world. This function will be used a lot to define any kind of shapes you will possibly want.

**glVertex4f(100.0f, 150.0f, -25.0f, 1.0f);** this is the same as glVertex3f, the only difference is in the last coordinate that specifies a scaling factor. The scaling factor is set to 1.0f by default. This won't make a lot of use and I'm not going to explain this function in details. It can be used to make your 3D points look thicker than one pixel. I don't want to sound pathetic but why would you want to use that functionality? Anyway, glVertex alone won't draw anything on the screen, it merely defines a vertex, usually of a more complex object. To really start displaying something on the screen you will have to use two additional functions. These functions are

**glBegin(int mode);** and **glEnd( void )**;

glBegin and glEnd delimit the vertices of a primitive or a group of like primitives. What this means is that everytime you want to draw a primitive on the screen you will first have to call glBegin, specifying what kind of primitive it is that you want to draw in the mode parameter of glBegin, and then list all vertices one by one (by sequentially calling glVertex) and finally call glEnd to let OpenGL know that you're done drawing a primitive. The parameter mode of the function glBegin can be one of the following:

1. GL\_POINTS
2. GL\_LINES
3. GL\_LINE\_STRIP
4. GL\_LINE\_LOOP
5. GL\_TRIANGLES
6. GL\_TRIANGLE\_STRIP
7. GL\_TRIANGLE\_FAN
8. GL\_QUADS
9. GL\_QUAD\_STRIP
10. GL\_POLYGON

**Conclusion:** Thus we have studied drawing basic diagrams using OpenGL.

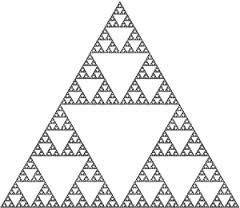
**Experiment No 12**

**Aim:** To draw Sierpenski Gasket using openGL.

**Theory:**

**Sierpenski Gasket:**

The **Sierpinski triangle** (also with the original orthography *Sierpiński*), also called the **Sierpinski gasket** or the **Sierpinski Sieve**, is a [fractal](http://en.wikipedia.org/wiki/Fractal) and [attractive fixed set](http://en.wikipedia.org/wiki/Attractive_fixed_set) with the overall shape of an equilateral triangle, subdivided recursively into smaller equilateral triangles. Originally constructed as a curve, this is one of the basic examples of [self-similar](http://en.wikipedia.org/wiki/Self-similarity) sets, i.e. it is a mathematically generated pattern that can be reproducible at any magnification or reduction. It is named after the [Polish](http://en.wikipedia.org/wiki/Poland)[mathematician](http://en.wikipedia.org/wiki/Mathematician)[WacławSierpiński](http://en.wikipedia.org/wiki/Wac%C5%82aw_Sierpi%C5%84ski), but appeared as a decorative pattern many centuries prior to the work of Sierpińsk



The construction proceedsas follows:

• 1. Pick an initial point atrandom inside the triangle

• 2. Select one of the threevertices at random

• 3. Find the point halfwaybetween the point and thevertex

• 4. Mark/Draw that half-waypoint

• 5. Replace the initial point withthis new point

• 6. Go to step

point2 vertices[3] = {{0.0,0.0}, {250.0,500}, {500.0, 0.0}};

static point2 p={75.0, 50.0};

int j,k;

for(k=0; k<5000;k++)

{j=rand()%3;

p[0]=(p[0]+triangle[j][0])/2;

p[1]=(p[1]+triangle[j][1])/2;

glBegin(GLPOINTS);

glVertex2fv(p);

**Conclusion:** Thus we have implemented Sierpenski gasket using OpenGL