**COMPUTER GRAPHICS**

**PRACTICAL FILE**

Q1.Write a program to implement Bresenham’s line drawing algorithm and DDA

solution :

**Bresenham’s**

#include<iostream.h>

#include<conio.h>

#include<graphics.h>

#include<math.h>

void bresenham\_line\_algo(int x1,int y1,int x2,int y2)

{

float dy,dx,p;

int x,y,xend,yend,dx1,dy1;

int xmid,ymid;

xmid=getmaxx()/2;

ymid=getmaxy()/2;

dx=abs(x2-x1);

dy=abs(y2-y1);

dx1=x2-x1;

dy1=y2-y1;

float m;

m=dy1/dx1;

const int c1=2\*dy,c2=2\*(dy-dx);

const int d1=2\*dx,d2=2\*(dx-dy);

if(m<=1&&m>0){

p=2\*dy-dx;

if(x1>x2){

x=x2;

y=y2;

xend=x1;

}

else{

x=x1;

y=y1;

xend=x2;

}

putpixel(x,y,YELLOW);

while(x<xend){

x++;

if(p<0){

p=p+c1;

}

else{

y++;

p=p+c2;

}

putpixel(xmid+x,ymid-y,YELLOW);

}

}

else{

p=2\*dx-dy;

if(y1>y2){

x=x2;

y=y2;

yend=y1;

}

else{

x=x1;

y=y1;

yend=y2;

}

putpixel(x,y,YELLOW);

while(y<yend){

y++;

if(p<0){

p=p+d1;

}

else{

x--;

p=p+d2;

}

putpixel(xmid+x,ymid-y,YELLOW);

}

}

}

void main(){

clrscr();

int gdriver=DETECT,gmode,errorcode;

int x1,y1,x2,y2,xmid,ymid;

initgraph(&gdriver,&gmode,"C:\\TURBOC3\\BGI");

cout<<"Enterthexco-ordinateoffirstpoint:";

cin>>x1;

cout<<"\nEntertheyco-ordinateoffirstpoint:";

cin>>y1;

cout<<"\nEnterthexco-ordinateofsecondpoint:";

cin>>x2;

cout<<"\nEntertheyco-ordinateofsecondpoint:";

cin>>y2;

xmid=getmaxx()/2;

ymid=getmaxy()/2;

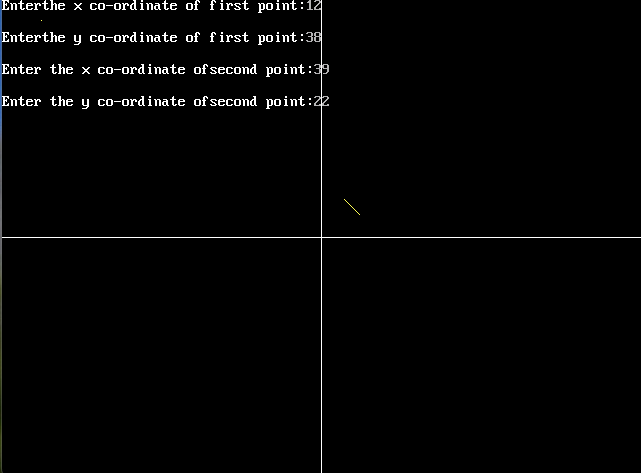
line(xmid,0,xmid,getmaxy());

line(0,ymid,getmaxx(),ymid);

bresenham\_line\_algo(x1,y1,x2,y2);

getch();

}



DDA

#include<iostream.h>

#include<conio.h>

#include<graphics.h>

#include<math.h>

void dda\_line\_algo(intx1,inty1,intx2,inty2){

int dx,dy,st;

dx=x2-x1;

dy=y2-y1;

float y,x,xinc,yinc;

int xmid,ymid;

xmid=getmaxx()/2;

ymid=getmaxy()/2;

if(abs(dx)>abs(dy)){

st=abs(dx);

}

else{

st=abs(dy);

}

xinc=dx/st;

yinc=dy/st;

x=x1;

y=y1;

for(inti=0;i<st;i++){

x+=xinc;

y+=yinc;

putpixel(ceil(x)+xmid,ymid-ceil(y),YELLOW);

}

}

void main(){

clrscr();

int gdriver=DETECT,gmode,errorcode;

int x1,y1,x2,y2,xmid,ymid;

initgraph(&gdriver,&gmode,"C:\\TURBOC3\\BGI");

cout<<"Enterthexco-ordinateoffirstpoint:";

cin>>x1;

cout<<"\nEntertheyco-ordinateoffirstpoint:";

cin>>y1;

cout<<"\nEnterthexco-ordinateofsecondpoint:";

cin>>x2;

cout<<"\nEntertheyco-ordinateofsecondpoint:";

cin>>y2;

xmid=getmaxx()/2;

ymid=getmaxy()/2;

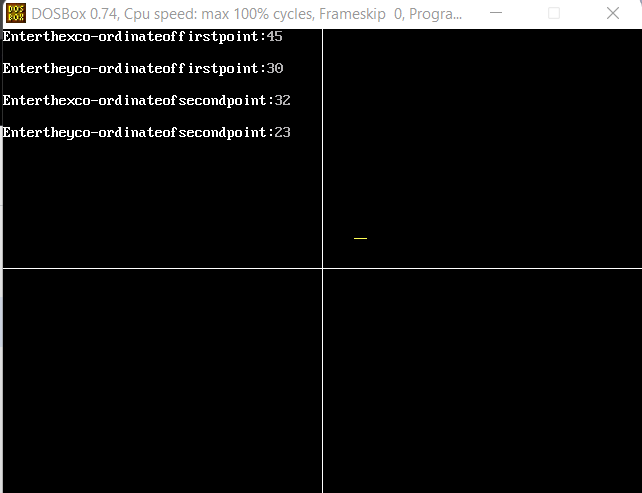
line(xmid,0,xmid,getmaxy());

line(0,ymid,getmaxx(),ymid);

dda\_line\_algo(x1,y1,x2,y2);

getch();

}



Q2: Write a program to implement mid point circle algorithm

Solution:

#include<iostream.h>

#include<conio.h.>

#include<graphics.h>

void midpointcircle(int,int,int);

void main()

{

int x,y,r;

clrscr();

cout<<"ENTER THE VALUES FOR X,Y,R : ";

cin>>x>>y>>r;

midpointcircle(x,y,r);

getch();

}

void midpointcircle(int a,intb,int r)

{

int gd=DETECT,gm;

initgraph(&gd,&gm,"c:\\turboc3\\bgi");

setfillstyle(1,GREEN);

setcolor(GREEN);

outtextxy(100,10,"PROGRAM TO START CONVERT A CIRCLE USING MID POINT ALGORITHM");

line(10,480,10,0);

outtextxy(480,10,"Y");

line(0,470,640,470);

outtextxy(640,70,"X");

int x=0,y=r,d=1-r;

while(x<=y)

{

putpixel(10+(a+x),470-(b+y),2);

putpixel(10+(a+x),470-(b-y),2);

putpixel(10+(a-x),470-(b+y),2);

putpixel(10+(a+y),470-(b+x),2);

putpixel(10+(a-x),470-(b-y),2);

putpixel(10+(a+y),470-(b-x),2);

putpixel(10+(a-y),470-(b+x),2);

putpixel(10+(a-y),470-(b-x),2);

if(d<0)

d=d+(2\*x)+3;

else

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

y--;

}

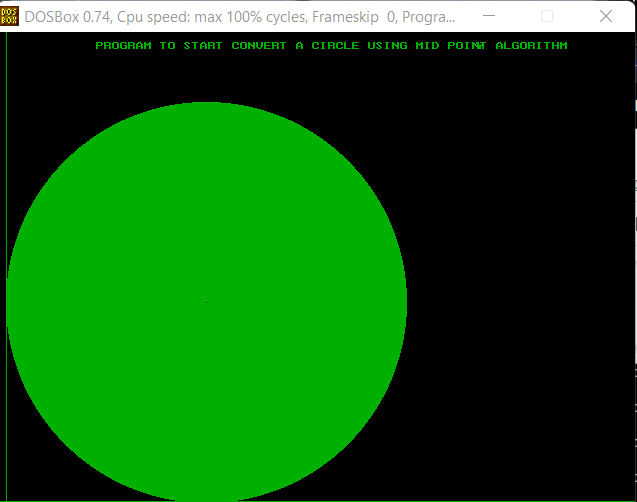
x++;

}

floodfill(10+a,470-b,GREEN);

putpixel(10+a,470-b,0);

}



Q3: Write a program to clip a line using Cohen and Sutherland line clipping algorithm.

Solution:

#include<iostream.h>

#include<conio.h>

#include<graphics.h>

#include<math.h>

void Window()

{

line (200,200,350,200);

line(350,200,350,350);

line(200,200,200,350);

line(200,350,350,350);

}

void Code(char c[4],float x,float y)

{ c[0]=(x<200)?'1':'0';

c[1]=(x>350)?'1':'0';

c[2]=(y<200)?'1':'0';

c[3]=(y>350)?'1':'0';

}

void Clipping (char c[],char d[],float &x,float&y,float m)

{

int flag=1,i=0;

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

{

if(c[i]!='0' && d[i]!='0')

{

flag=0;

break;

}

if(flag)

{

if(c[0]!='0')

{

y=m\*(200-x)+y;

x=200;

}

else if(c[1]!='0')

{

y=m\*(350-x)+y;

x=350;

}

else if(c[2]!='0')

{

x=((200-y)/m)+x;

y=200;

}

else if(c[3]!='0')

{

x=((350-y)/m)+x;

y=350;

}

}

if (flag==0)

cout<<"Line lying outside";

}

}

void main()

{

int gdriver = DETECT, gmode, errorcode;

float x1,y1,x2,y2;

float m;

char c[4],d[4];

clrscr();

initgraph(&gdriver, &gmode, "//Turboc3//bgi");

cout<<"Enter coordinates";

cin>>x1>>y1>>x2>>y2;

cout<<"Before clipping";

Window();

line(x1,y1,x2,y2);

getch();

cleardevice();

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

Code(c,x1,y1);

Code(d,x2,y2) ;

Clipping(c,d,x1,y1,m);

Clipping(d,c,x2,y2,m);

cout<<"After Clipping";

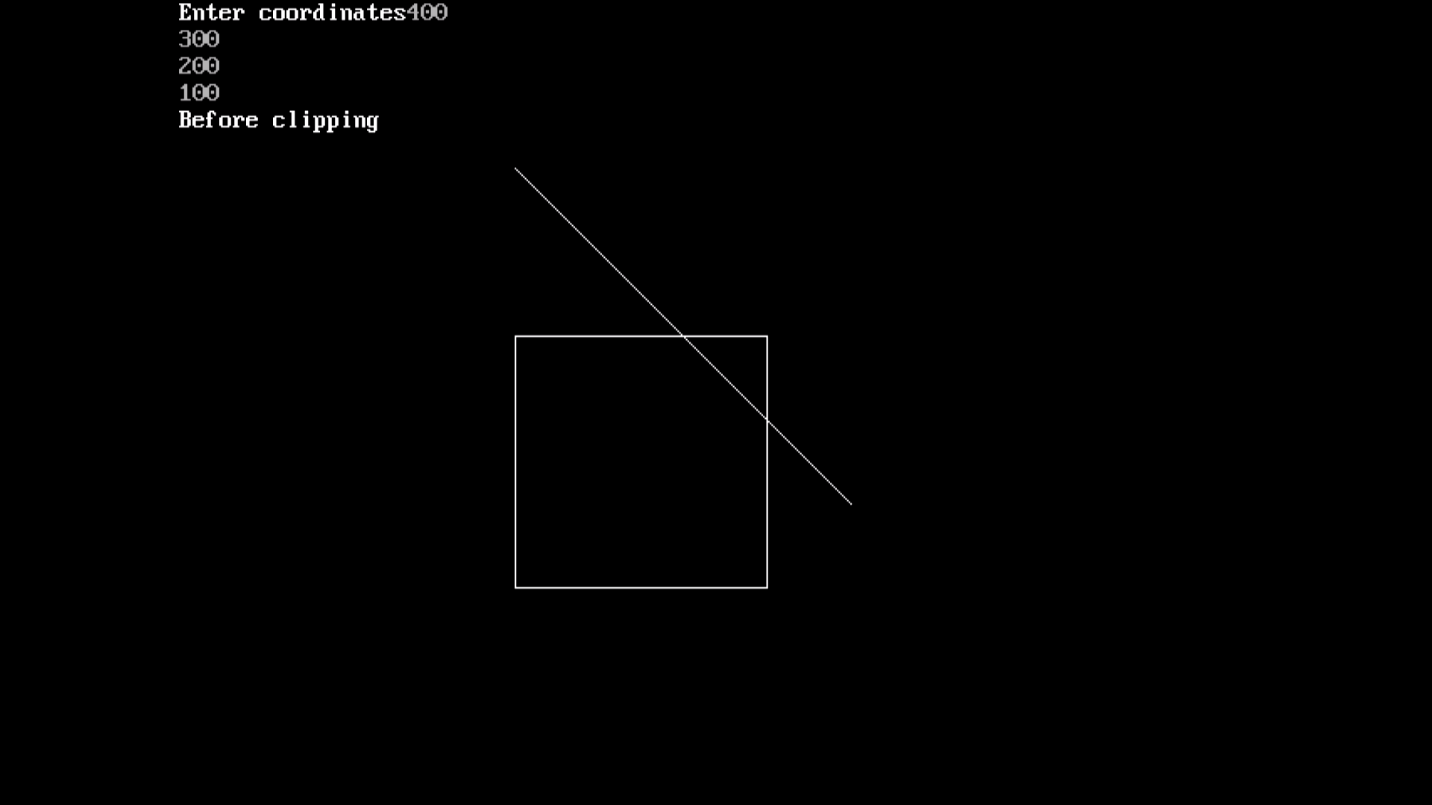
Window();

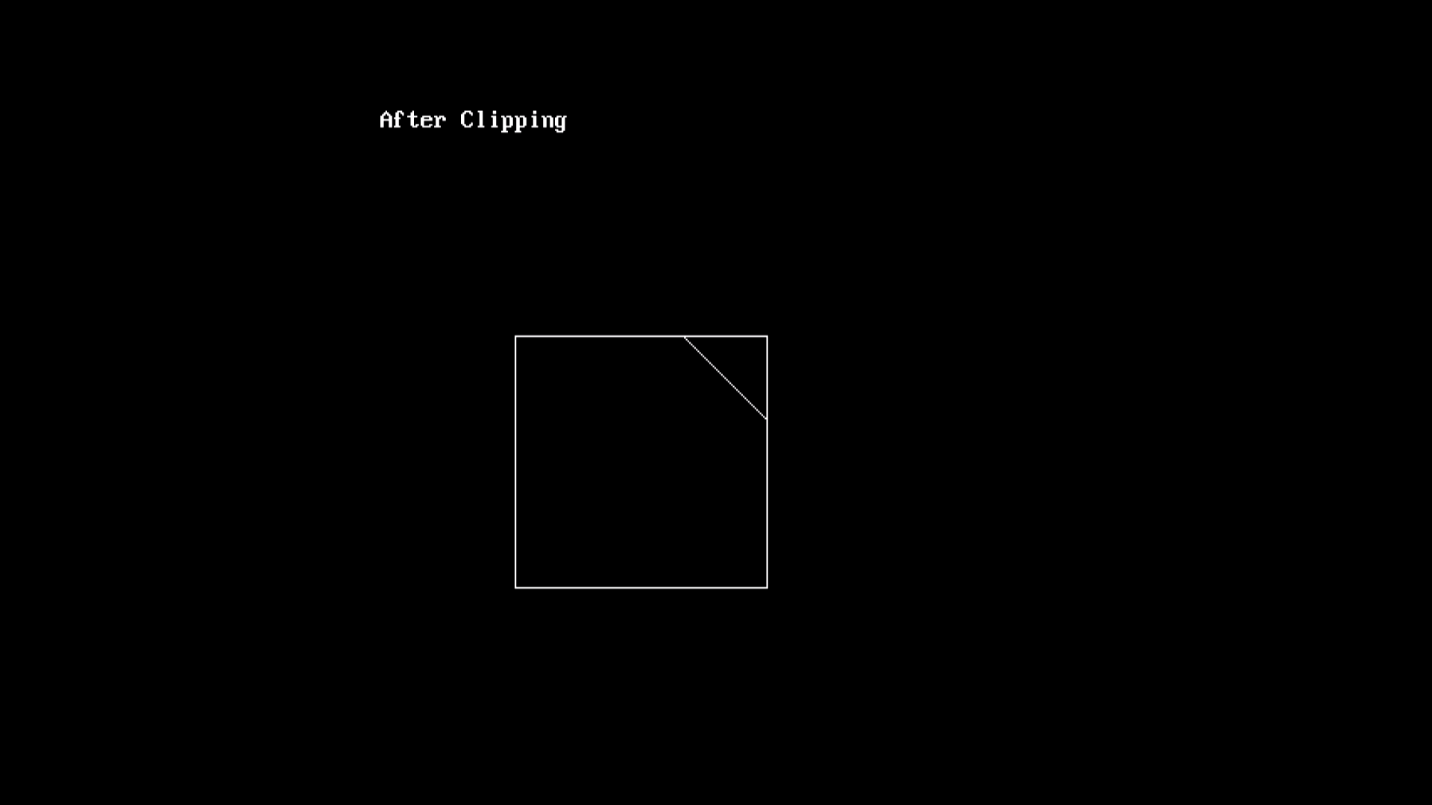
line(x1,y1,x2,y2);

getch();

closegraph();

}





Q4: Write a program to clip a polygon using the Sutherland Hodgeman algorithm.

Solution:

#include<iostream.h>

#include<conio.h>

#include<stdio.h>

#include<graphics.h>

#include<dos.h>

union REGS i, o;

struct pt

{

int x, y;

};

float xl, xr, yt, yb, m, slope[20];

int bc = 0, xc, yc, n = 0, k;

int dx, dy, x, y, temp, a[20][2], xi[20];

struct point

{

float x, y;

};

enum bound

{

left,

right,

bottom,

top

};

int inside( struct point p, enum bound b )

{

int c = 1;

switch( b )

{

case left:

if( p.x< xl )

c = 0;

break;

case right:

if( p.x>xr )

c = 0;

break;

case bottom:

if( p.y>yb )

c = 0;

break;

case top:

if( p.y<yt )

c = 0;

break;

}

return c;

}

struct point intersect( struct point p1, struct point p2, enum bound b )

{

struct point t;

float m = 0;

if( p2.x != p1.x )

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

switch( b )

{

case left:

t.x = xl;

t.y = p2.y + ( xl - p2.x ) \* m;

break;

case right:

t.x = xr;

t.y = p2.y + ( xr - p2.x ) \* m;

break;

case bottom:

t.y = yb;

if( p1.x == p2.x )

t.x = p2.x;

else

t.x = p2.x + ( yb - p2.y ) / m;

break;

case top:

t.y = yt;

if( p1.x == p2.x )

t.x = p2.x;

else

t.x = p2.x + ( yt - p2.y ) / m;

break;

}

return t;

}

int initmouse( )

{

i.x.ax = 0;

int86( 0X33, &i, &o );

return( o.x.ax );

}

void showmouseptr( )

{

i.x.ax = 1;

int86( 0X33, &i, &o );

}

void hidemouseptr( )

{

i.x.ax = 2;

int86( 0X33, &i, &o );

}

void getmousepos( int \*button, int \*x, int \*y )

{

i.x.ax = 3;

int86( 0X33, &i, &o );

\*button = o.x.bx;

\*x = o.x.cx;

\*y = o.x.dx;

}

void main( )

{

enum bound b;

int cou, i, flag;

struct point p[30], pout[30], z;

int gdriver = DETECT, gmode;

initgraph( &gdriver, &gmode, "..\\BGI" );

cleardevice( );

showmouseptr( );

while( bc != 2 )

{

getmousepos( &bc, &xc, &yc );

if( bc == 1 )

{

p[n].x = xc;

p[n].y = yc;

n++;

hidemouseptr( );

if( n> 1 )

line( p[n-2].x, p[n-2].y, xc, yc );

showmouseptr( );

delay( 100 );

}

}

p[n] = p[0];

hidemouseptr( );

line( p[n-1].x, p[n-1].y, p[n].x, p[n].y );

showmouseptr( );

getmousepos( &bc, &xc, &yc );

flag = 1;

bc = 0;

while( bc != 2 )

{

if( (bc == 1 ) && ( flag == 1 ) )

{

xl = xc;

yt = yc;

flag = 2;

}

else

{

xr = xc;

yb = yc;

}

getmousepos( &bc, &xc, &yc );

}

rectangle( xl, yt, xr, yb );

getch( );

for(b=left; b <= top; b++)

{

cou = -1;

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

if( ( inside( p[i], b ) == 0 ) && ( inside( p[i + 1], b ) == 1 ) )

{

z = intersect( p[i], p[i + 1], b );

pout[++cou] = z;

pout[++cou] = p[i + 1];

}

else

if( ( inside( p[i], b ) == 1 ) && ( inside( p[i + 1], b ) == 1 ) )

pout[++cou] = p[i + 1];

else

if( ( inside( p[i], b ) == 1 ) && ( inside( p[i + 1], b ) == 0 ) )

{

z = intersect( p[i], p[i + 1], b );

pout[++cou] = z;

}

pout[++cou] = pout[0];

n = cou;

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

p[i] = pout[i];

}

getch( );

cleardevice( );

rectangle( xl, yt, xr, yb );

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

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

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

{

a[i][0] = p[i].x;

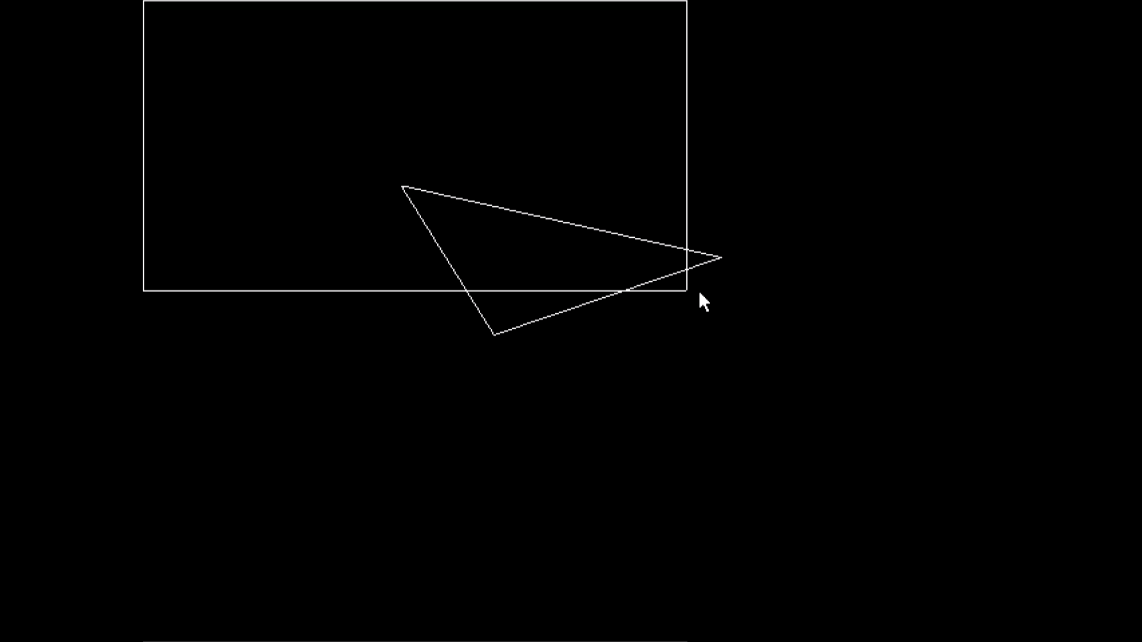
a[i][1] = p[i].y;

}

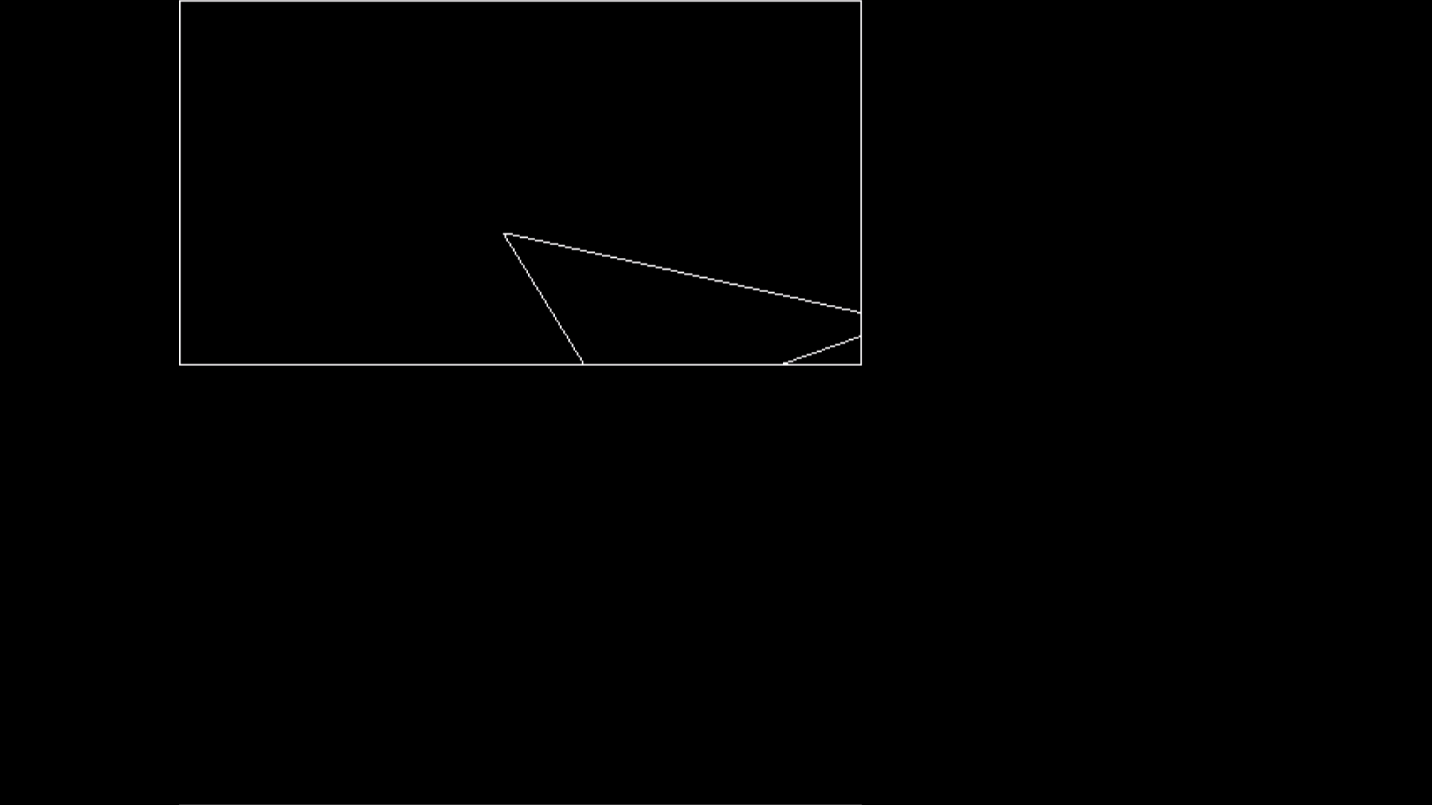
getch( );

closegraph( );

}



Before clipping



After clipping

Q5. Write a program to fill a polygon using Scan line fill algorithm.

Program

#include<iostream.h>

#include<conio.h>

#include<graphics.h>

#include<dos.h>

struct edge{

int x1,y1,x2,y2,flag;

};

void main(){

int gd=DETECT,gm,n,i,j,k;

edge ed[10], temped;

float dx,dy,m[10],x\_int[10],inter\_x[10];

int x[10],y[10],ymax=0,ymin=480,yy,temp;

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

cout<<"Enter the no. of vertices of the polygon : ";

cin>>n;

cout<<"\nEnter the vertices :- \n";

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

cout<<"P"<<i+1<<" : ";

cin>>x[i]>>y[i];

if(y[i]>ymax)

ymax=y[i];

if(y[i]<ymin)

ymin=y[i];

ed[i].x1=x[i];

ed[i].y1=y[i];

}

for(i=0;i<n-1;i++){

ed[i].x2=ed[i+1].x1;

ed[i].y2=ed[i+1].y1;

ed[i].flag=0;

}

ed[i].x2=ed[0].x1;

ed[i].y2=ed[0].y1;

ed[i].flag=0;

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

if(ed[i].y1 < ed[i].y2){

temp=ed[i].x1;

ed[i].x1=ed[i].x2;

ed[i].x2=temp;

temp=ed[i].y1;

ed[i].y1=ed[i].y2;

ed[i].y2=temp;

}

}

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

line(ed[i].x1, ed[i].y1,ed[i].x2,ed[i].y2);

}

for(i=0;i<n-1;i++){

for(j=0;j<n-1;j++){

if(ed[j].y1<ed[j+1].y1){

temped=ed[j];

ed[j]=ed[j+1];

ed[j+1]=temped;

}

if(ed[j].y1==ed[j+1].y1){

if(ed[j].y2<ed[j+1].y2){

temped=ed[j];

ed[j]=ed[j+1];

ed[j+1]=temped;

}

if(ed[j].y2==ed[j+1].y2){

if(ed[j].x1<ed[j+1].x1){

temped=ed[j];

ed[j]=ed[j+1];

ed[j+1]=temped;

}

}

}

}

}

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

dx=ed[i].x2-ed[i].x1;

dy=ed[i].y2-ed[i].y1;

if(dy==0){

m[i]=0;

}

else {

m[i]=dx/dy;

}

inter\_x[i]=ed[i].x1;

}

yy=ymax;

while(yy>ymin){

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

if(yy>ed[i].y2 &&yy<=ed[i].y1){

ed[i].flag=1;

}

else{

ed[i].flag=0;

}

}

j=0;

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

if(ed[i].flag==1){

if(yy==ed[i].y1){

x\_int[j]==ed[i].x1;

j++;

if(ed[i-1].y1==yy&& ed[i-1].y1<yy) {

x\_int[j]=ed[i].x1;

j++;

}

if(ed[i+1].y1==yy&& ed[i+1].y1<yy){

x\_int[j]=ed[i].x1;

j++;

}

}

else {

x\_int[j]=inter\_x[i]+(-m[i]);

inter\_x[i]=x\_int[j];

j++;

}

}

}

for(i=0;i<j;i++){

for(k=0;k<j-1;k++){

if(x\_int[k]>x\_int[k+1]){

temp=(int)x\_int[k];

x\_int[k]=x\_int[k+1];

x\_int[k+1]=temp;

}

}

}

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

line((int)x\_int[i],yy,(int)x\_int[i+1],yy);

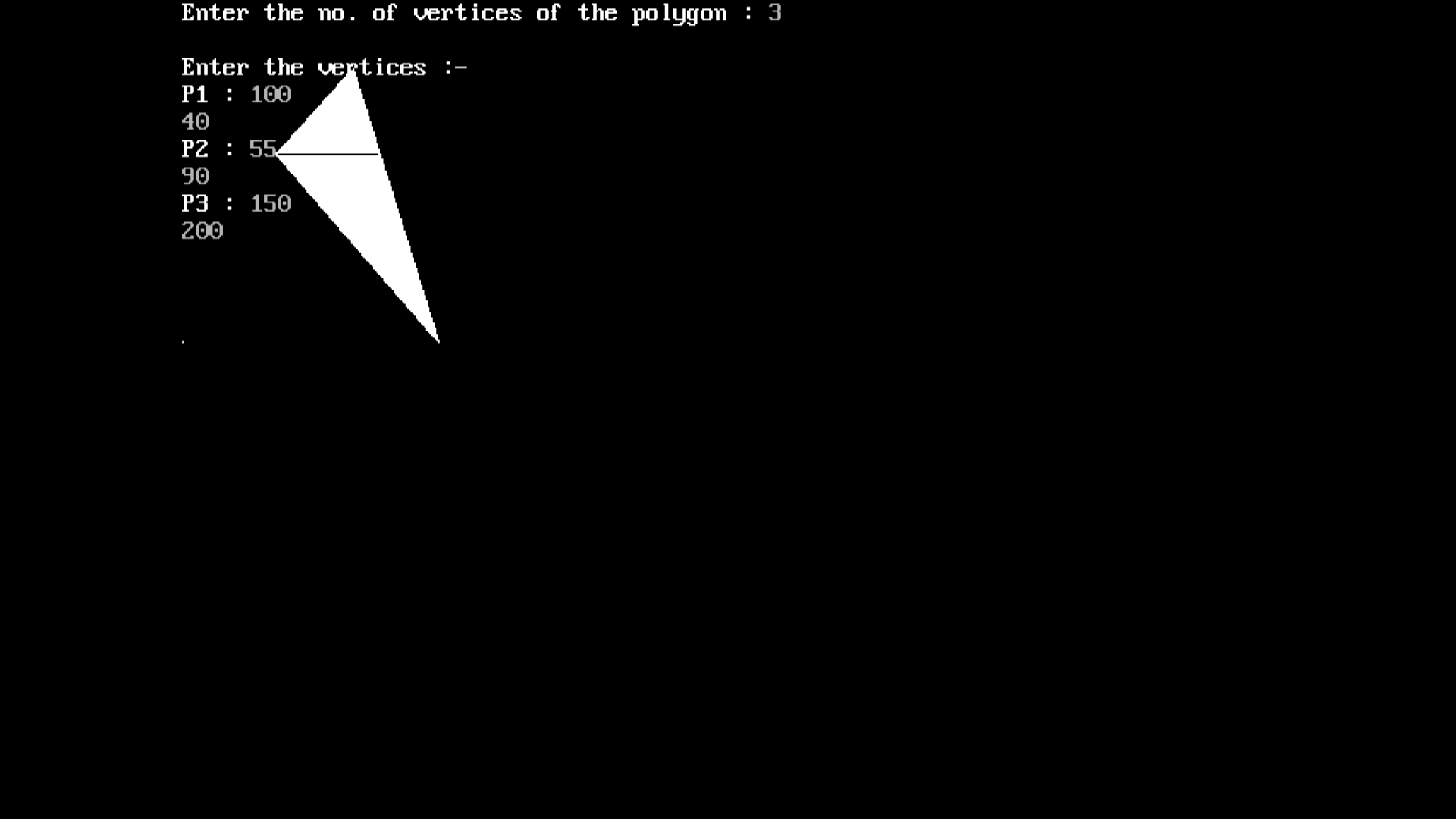
}

yy--;

}

getch();

}



Q6.Write a program to apply various 2D transformations on a 2D object.

Program

#include<graphics.h>

#include<stdlib.h>

#include<stdio.h>

#include<iostream.h>

#include<conio.h>

#include<math.h>

int mat[3][3];

void dda\_line(int x1 , int y1 , int x2 , int y2 , int col){

int dx ,dy , st;

dx = x2 - x1;

dy = y2 - y1;

float y , x , xinc , yinc;

int xmid ,ymid;

xmid = getmaxx()/2;

ymid = getmaxy()/2;

if(abs(dx) > abs(dy)){

st = abs(dx);

}

else{

st = abs(dy);

}

xinc = dx / st;

yinc = dy / st;

x = x1;

y = y1;

for(int i=0 ; i<st ; i++){

x += xinc;

y += yinc;

putpixel(ceil(x) + xmid ,ymid - ceil(y),col);

}

}

void rotate(){

int xmid ,ymid;

xmid = getmaxx()/2;

ymid = getmaxy()/2;

line(xmid , 0 , xmid , getmaxy());

line(0 , ymid , getmaxx() , ymid);

int c[3][2] ,l , m, i , j , k;

int a[3][2]={{200,200},{200,100},{100,200}};

int t[2][2]={{0,1},{-1,0}};

for( i = 0 ; i< 3 ; i++){

for(j=0 ; j<2 ; j++){

c[i][j]=0;

}

}

dda\_line(a[0][0],a[0][1],a[1][0],a[1][1],YELLOW);

dda\_line(a[1][0],a[1][1],a[2][0],a[2][1],YELLOW);

dda\_line(a[2][0],a[2][1],a[0][0],a[0][1],YELLOW);

for ( i=0;i<3;i++){

for ( j=0;j<2;j++){

for ( k=0;k<2;k++){

c[i][j]=c[i][j]+(a[i][k]\*t[k][j]);

}

}

}

dda\_line(c[0][0],c[0][1],c[1][0],c[1][1],GREEN);

dda\_line(c[1][0],c[1][1],c[2][0],c[2][1],GREEN);

dda\_line(c[2][0],c[2][1],c[0][0],c[0][1],GREEN);

}

void reflection(){

int xmid ,ymid;

xmid = getmaxx()/2;

ymid = getmaxy()/2;

line(xmid , 0 , xmid , getmaxy());

line(0 , ymid , getmaxx() , ymid);

int c[3][2] ,l , m, i , j , k;

int a[3][2]={{200,200},{200,100},{100,200}};

int t[2][2]={{0,-1},{-1,0}};

for( i = 0 ; i< 3 ; i++){

for(j=0 ; j<2 ; j++){

c[i][j]=0;

}

}

dda\_line(a[0][0],a[0][1],a[1][0],a[1][1],YELLOW);

dda\_line(a[1][0],a[1][1],a[2][0],a[2][1],YELLOW);

dda\_line(a[2][0],a[2][1],a[0][0],a[0][1],YELLOW);

for ( i=0;i<3;i++){

for ( j=0;j<2;j++){

for ( k=0;k<2;k++){

c[i][j]=c[i][j]+(a[i][k]\*t[k][j]);

}

}

}

dda\_line(c[0][0],c[0][1],c[1][0],c[1][1],GREEN);

dda\_line(c[1][0],c[1][1],c[2][0],c[2][1],GREEN);

dda\_line(c[2][0],c[2][1],c[0][0],c[0][1],GREEN);

}

void scaling(){

int xmid ,ymid;

xmid = getmaxx()/2;

ymid = getmaxy()/2;

line(xmid , 0 , xmid , getmaxy());

line(0 , ymid , getmaxx() , ymid);

int c[3][2] ,l , m, i , j , k;

int a[3][2]={{20,20},{20,10},{10,20}};

int t[2][2]={{5,0},{0,5}};

for( i = 0 ; i< 3 ; i++){

for(j=0 ; j<2 ; j++){

c[i][j]=0;

}

}

dda\_line(a[0][0],a[0][1],a[1][0],a[1][1],YELLOW);

dda\_line(a[1][0],a[1][1],a[2][0],a[2][1],YELLOW);

dda\_line(a[2][0],a[2][1],a[0][0],a[0][1],YELLOW);

for ( i=0;i<3;i++){

for ( j=0;j<2;j++){

for ( k=0;k<2;k++){

c[i][j]=c[i][j]+(a[i][k]\*t[k][j]);

}

}

}

dda\_line(c[0][0],c[0][1],c[1][0],c[1][1],GREEN);

dda\_line(c[1][0],c[1][1],c[2][0],c[2][1],GREEN);

dda\_line(c[2][0],c[2][1],c[0][0],c[0][1],GREEN);

}

void multi(int a[3][3] , int b[3][3] ){

int i , j ,k;

int c[3][3];

for( i = 0 ; i< 3 ; i++){

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

c[i][j]=0;

}

}

for ( i=0;i<3;i++){

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

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

c[i][j]=c[i][j]+(a[i][k]\*b[k][j]);

}

}

}

for( i = 0 ; i< 3 ; i++){

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

mat[i][j]=c[i][j];

}

}

}

void reflection\_arbitrary(){

int xmid ,ymid;

xmid = getmaxx()/2;

ymid = getmaxy()/2;

line(xmid , 0 , xmid , getmaxy());

line(0 , ymid , getmaxx() , ymid);

int a[3][3]={{200,200,1},{200,100,1},{100,200,1}};

int t[3][3]={{1,0,0},{0,1,0},{0,0,1}};

int r[3][3]={{-1,0,0},{0,-1,0},{0,0,1}};

int ref[3][3]={{1,0,0},{0,-1,0},{0,0,1}};

int rinv[3][3]={{-1,0,0},{0,-1,0},{0,0,1}};

int tinv[3][3]={{1,0,0},{0,1,0},{0,1,1}};

dda\_line(a[0][0],a[0][1],a[1][0],a[1][1],YELLOW);

dda\_line(a[1][0],a[1][1],a[2][0],a[2][1],YELLOW);

dda\_line(a[2][0],a[2][1],a[0][0],a[0][1],YELLOW);

multi(t,r);

multi(mat,ref);

multi(mat,rinv);

multi(mat,tinv);

multi(a,mat);

dda\_line(mat[0][0],mat[0][1],mat[1][0],mat[1][1],GREEN);

dda\_line(mat[1][0],mat[1][1],mat[2][0],mat[2][1],GREEN);

dda\_line(mat[2][0],mat[2][1],mat[0][0],mat[0][1],GREEN);

}

void rotation\_arbitrary(){

int xmid ,ymid;

xmid = getmaxx()/2;

ymid = getmaxy()/2;

line(xmid , 0 , xmid , getmaxy());

line(0 , ymid , getmaxx() , ymid);

int c[3][3] , i , j , k;

int l[1][3]={{200,200,1}};

int a[3][3]={{200,200,1},{200,100,1},{100,200,1}};

int t[3][3]={{1,0,0},{0,1,0},{-133,-133,1}};

int r[3][3]={{-1,0,0},{0,-1,0},{0,0,1}};

int tinv[3][3]={{1,0,0},{0,1,0},{133,133,1}};

dda\_line(a[0][0],a[0][1],a[1][0],a[1][1],YELLOW);

dda\_line(a[1][0],a[1][1],a[2][0],a[2][1],YELLOW);

dda\_line(a[2][0],a[2][1],a[0][0],a[0][1],YELLOW);

multi(t,r);

multi(mat,tinv);

for( i = 0 ; i< 3 ; i++){

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

c[i][j]=0;

}

}

for ( i=0;i<3;i++){

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

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

c[i][j]=c[i][j]+(a[i][k]\*mat[k][j]);

}

}

}

dda\_line(c[0][0],c[0][1],c[1][0],c[1][1],GREEN);

dda\_line(c[1][0],c[1][1],c[2][0],c[2][1],GREEN);

dda\_line(c[2][0],c[2][1],c[0][0],c[0][1],GREEN);

}

void main(){

clrscr();

int gdriver = DETECT ,gmode , errorcode;

initgraph(&gdriver, &gmode, "C:\\TURBOC3\\BGI");

int n , m;

cout<<" 1.Rotation \n 2.Reflection \n 3.Scaling \n

4.Reflection about an arbitrary axis \n";

cout<<" 5.Rotation about an arbitrary point\n";

cout<<"Enter your choice : ";

cin>>n;

switch(n){

case 1 : rotate();

break;

case 2 : reflection();

break;

case 3 : scaling();

break;

case 4 :reflection\_arbitrary();

break;

case 5 :rotation\_arbitrary();

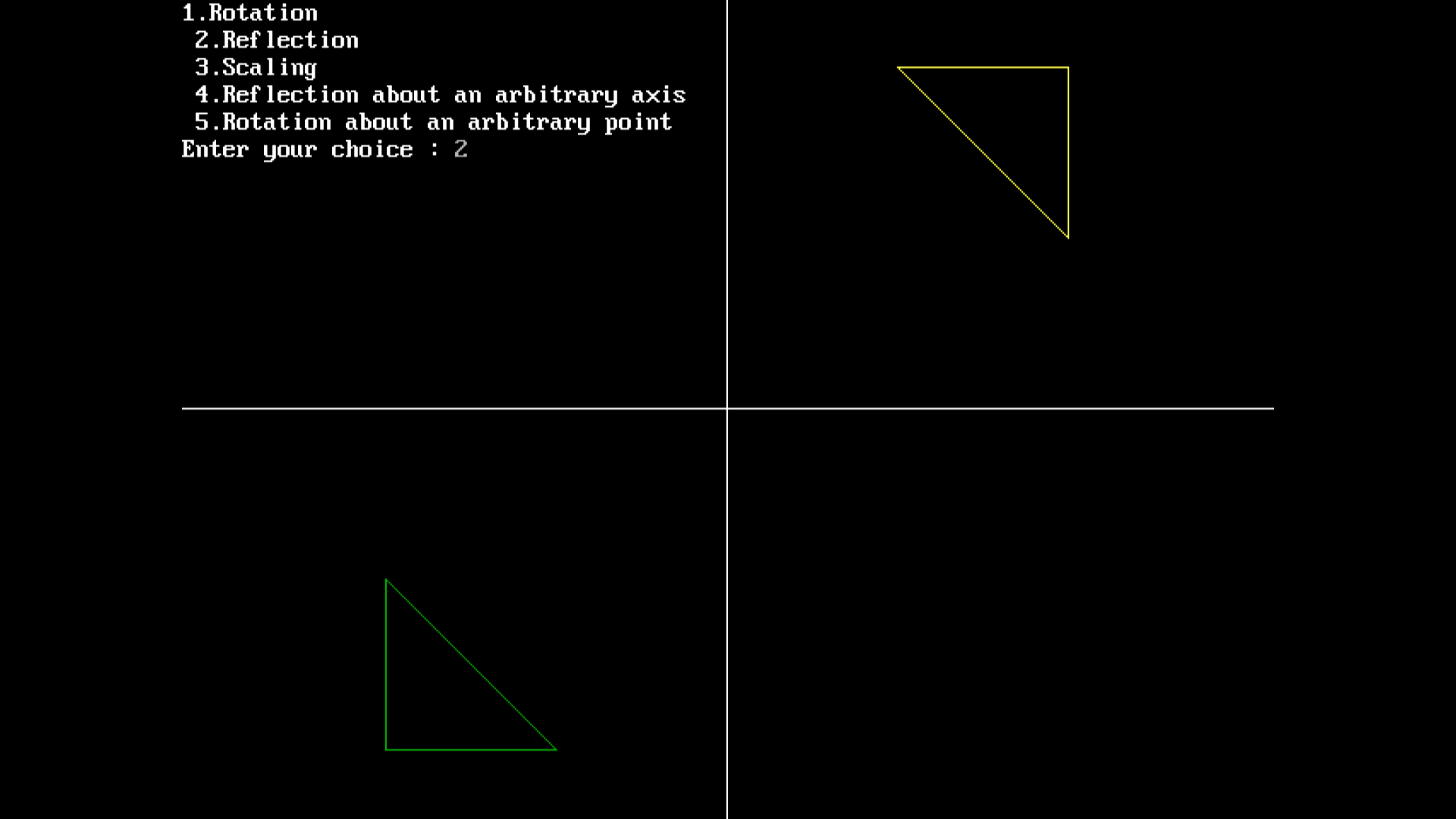
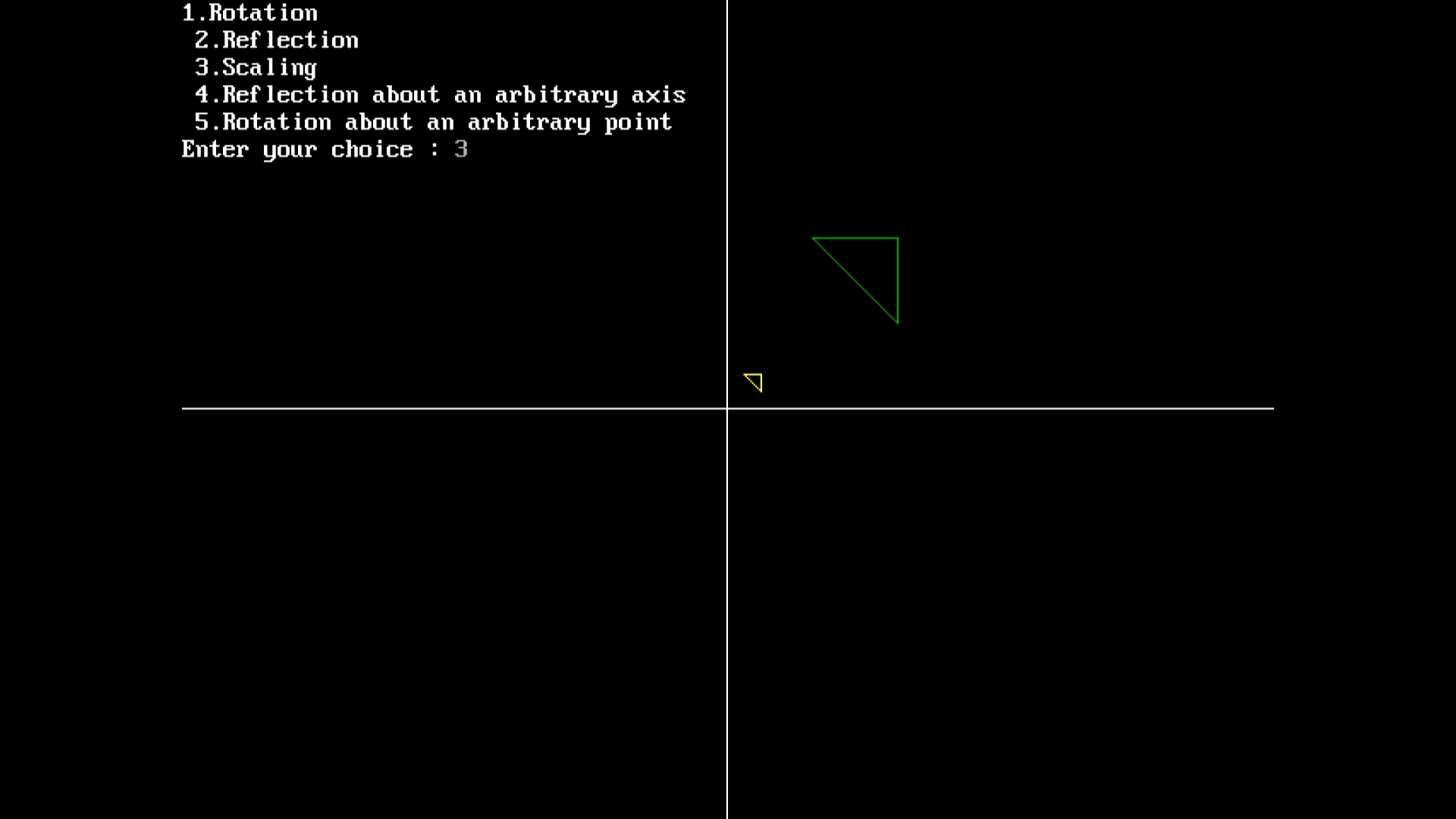
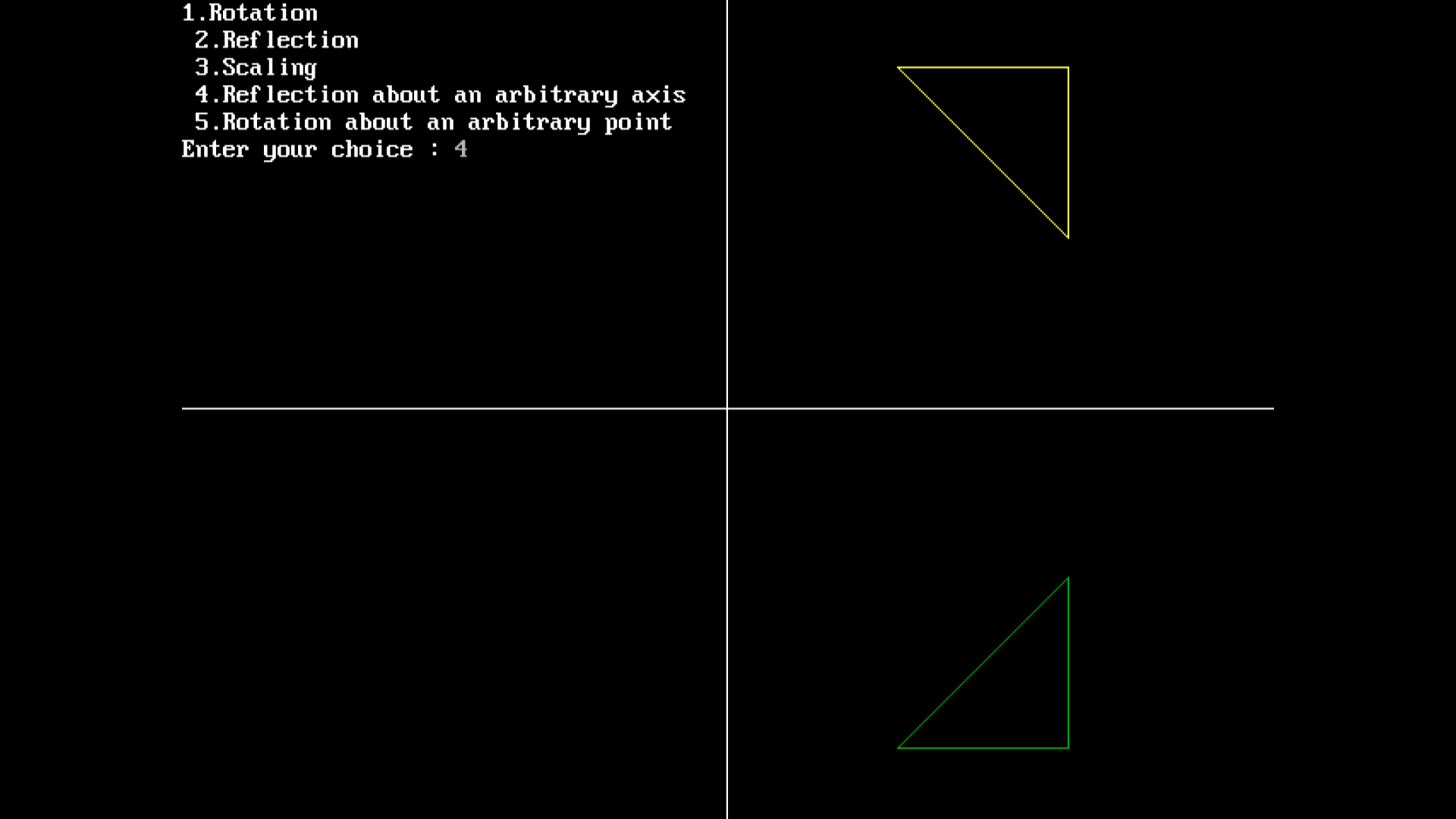
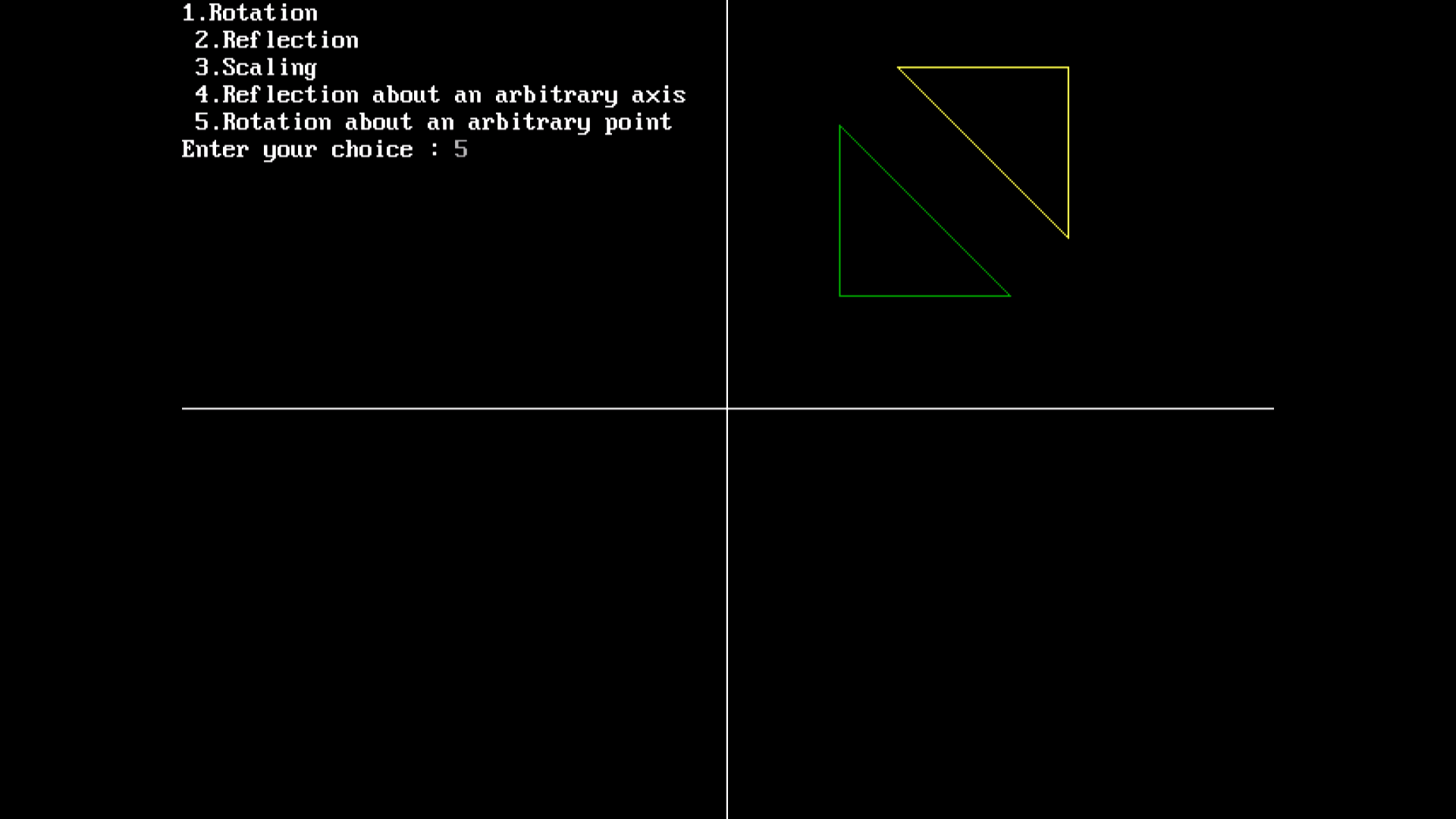
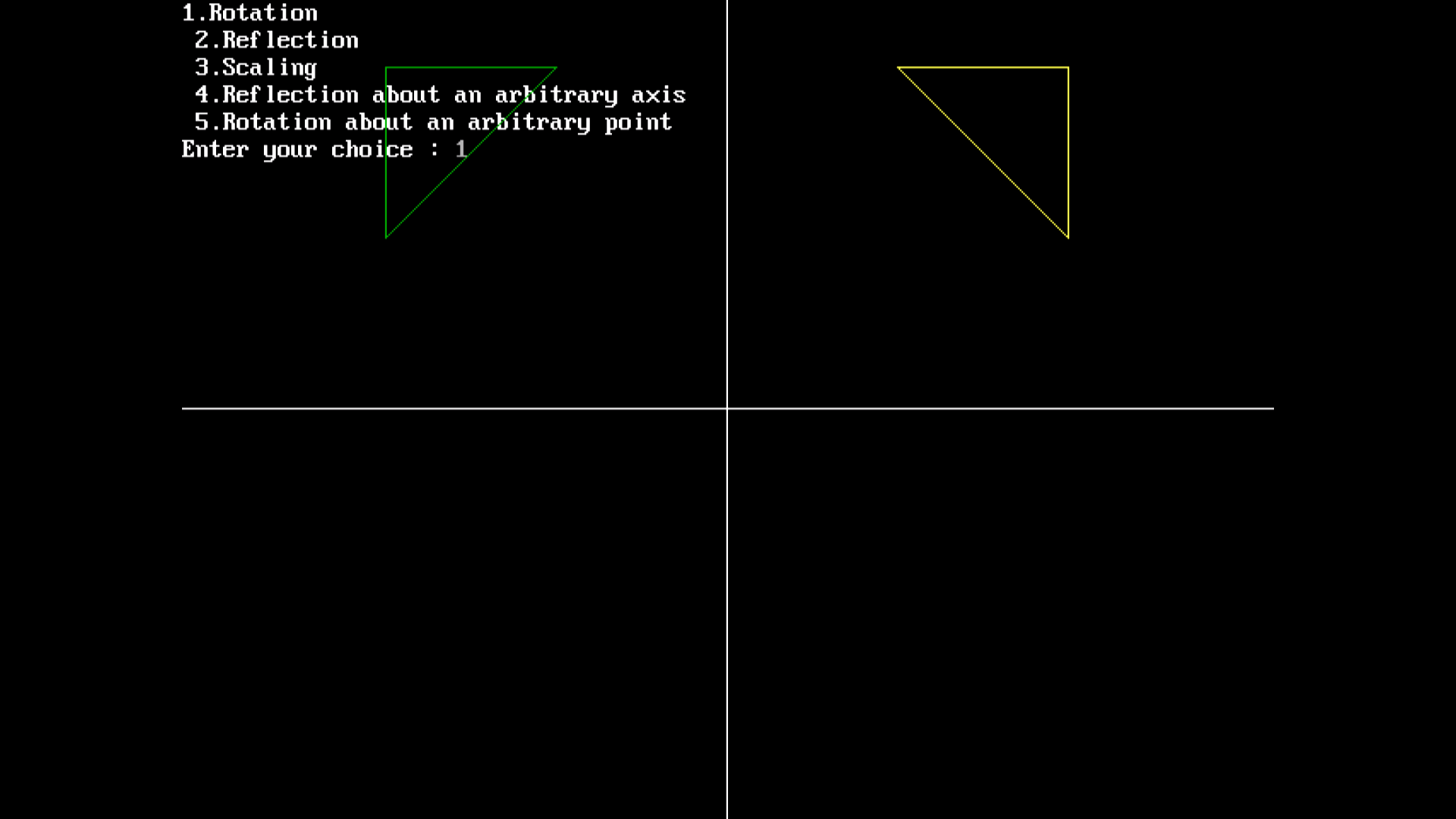
break;

default :cout<<"Invalid Choice\n";

}

getch();

}



Q7. Write a program to apply various 3D transformations on a 3D object and then apply parallel and perspective projection on it.

Program

#include<iostream.h>

#include<dos.h>

#include<stdio.h>

#include<math.h>

#include<conio.h>

#include<graphics.h>

#include<process.h>

double x1,x2,y1,y2;

void draw\_cube(double edge[20][3]){

int i;

cleardevice();

for(i=0;i<19;i++){

x1=edge[i][0]+edge[i][2]\*(cos(2.3562));

y1=edge[i][1]-edge[i][2]\*(sin(2.3562));

x2=edge[i+1][0]+edge[i+1][2]\*(cos(2.3562));

y2=edge[i+1][1]-edge[i+1][2]\*(sin(2.3562));

line(x1+320,240-y1,x2+320,240-y2);

}

line(320,240,320,25);

line(320,240,550,240);

line(320,240,150,410);

}

void translate(double edge[20][3]){

int a,b,c;

int i;

cout<<"Enter the Translation Factors : ";

cin>>a>>b>>c;

cleardevice();

for(i=0;i<20;i++){

edge[i][0]+=a;

edge[i][0]+=b;

edge[i][0]+=c;

}

draw\_cube(edge);

}

void rotate(double edge[20][3]){

int n;

int i;

double temp,theta,temp1;

cleardevice();

cout<<" 1.X-Axis \n 2.Y-Axis \n 3.Z-Axis \n";

cout<<"Enter your choice : ";

cin>>n;

switch(n){

case 1: cout<<" Enter The Angle ";

cin>>theta;

theta=(theta\*3.14)/180;

for(i=0;i<20;i++){

edge[i][0]=edge[i][0];

temp=edge[i][1];

temp1=edge[i][2];

edge[i][1]=temp\*cos(theta)-temp1\*sin(theta);

edge[i][2]=temp\*sin(theta)+temp1\*cos(theta);

}

draw\_cube(edge);

break;

case 2: cout<<" Enter The Angle ";

cin>>theta;

theta=(theta\*3.14)/180;

for(i=0;i<20;i++){

edge[i][1]=edge[i][1];

temp=edge[i][0];

temp1=edge[i][2];

edge[i][0]=temp\*cos(theta)+temp1\*sin(theta);

edge[i][2]=-temp\*sin(theta)+temp1\*cos(theta);

}

draw\_cube(edge);

break;

case 3: cout<<" Enter The Angle ";

cin>>theta;

theta=(theta\*3.14)/180;

for(i=0;i<20;i++){

edge[i][2]=edge[i][2];

temp=edge[i][0];

temp1=edge[i][1];

edge[i][0]=temp\*cos(theta)-temp1\*sin(theta);

edge[i][1]=temp\*sin(theta)+temp1\*cos(theta);

}

draw\_cube(edge);

break;

}

}

void reflect(double edge[20][3]){

int n;

int i;

cleardevice();

cout<<" 1.X-Axis \n 2.Y-Axis \n 3.Z-Axis \n";

cout<<" Enter Your Choice : ";

cin>>n;

switch(n){

case 1: for(i=0;i<20;i++){

edge[i][0]=edge[i][0];

edge[i][1]=-edge[i][1];

edge[i][2]=-edge[i][2];

}

draw\_cube(edge);

break;

case 2: for(i=0;i<20;i++){

edge[i][1]=edge[i][1];

edge[i][0]=-edge[i][0];

edge[i][2]=-edge[i][2];

}

draw\_cube(edge);

break;

case 3: for(i=0;i<20;i++){

edge[i][2]=edge[i][2];

edge[i][0]=-edge[i][0];

edge[i][1]=-edge[i][1];

}

draw\_cube(edge);

break;

}

}

void perspect(double edge[20][3]){

int n;

int i;

double p,q,r;

cleardevice();

cout<<" 1.X-Axis \n 2.Y-Axis \n 3.Z-Axis\n";

cout<<" Enter Your Choice : ";

cin>>n;

switch(n){

case 1: cout<<" Enter P : ";

cin>>p;

for(i=0;i<20;i++){

edge[i][0]=edge[i][0]/(p\*edge[i][0]+1);

edge[i][1]=edge[i][1]/(p\*edge[i][0]+1);

edge[i][2]=edge[i][2]/(p\*edge[i][0]+1);

}

draw\_cube(edge);

break;

case 2: cout<<" Enter Q : ";

cin>>q;

for(i=0;i<20;i++){

edge[i][1]=edge[i][1]/(edge[i][1]\*q+1);

edge[i][0]=edge[i][0]/(edge[i][1]\*q+1);

edge[i][2]=edge[i][2]/(edge[i][1]\*q+1);

}

draw\_cube(edge);

break;

case 3: cout<<" Enter R : ";

cin>>r;

for(i=0;i<20;i++){

edge[i][2]=edge[i][2]/(edge[i][2]\*r+1);

edge[i][0]=edge[i][0]/(edge[i][2]\*r+1);

edge[i][1]=edge[i][1]/(edge[i][2]\*r+1);

}

draw\_cube(edge);

break;

}

}

void main(){

clrscr();

int gdriver = DETECT ,gmode , errorcode;

initgraph(&gdriver, &gmode, "C:\\TURBOC3\\BGI");

int n;

double

edge[20][3]={100,0,0,100,100,0,0,100,0,0,100,100,0,0,100,0,0,0,100,

0,0,

100,0,100,100,75,100,75,100,100,100,100,75,100,100,0,100,100,75,

100,75,100,75,100,100,0,100,100,0,100,0,0,0,0,0,0,100,100,0,100};

cout<<" 1.Draw Cube \n 2.Rotation \n 3.Reflection \n";

cout<<" 4.Translation \n 5.Perspective Projection \n";

cout<<" Enter Your Choice : ";

cin>>n;

switch(n){

case 1: draw\_cube(edge);

break;

case 2: rotate(edge);

break;

case 3: reflect(edge);

break;

case 4: translate(edge);

break;

case 5: perspect(edge);

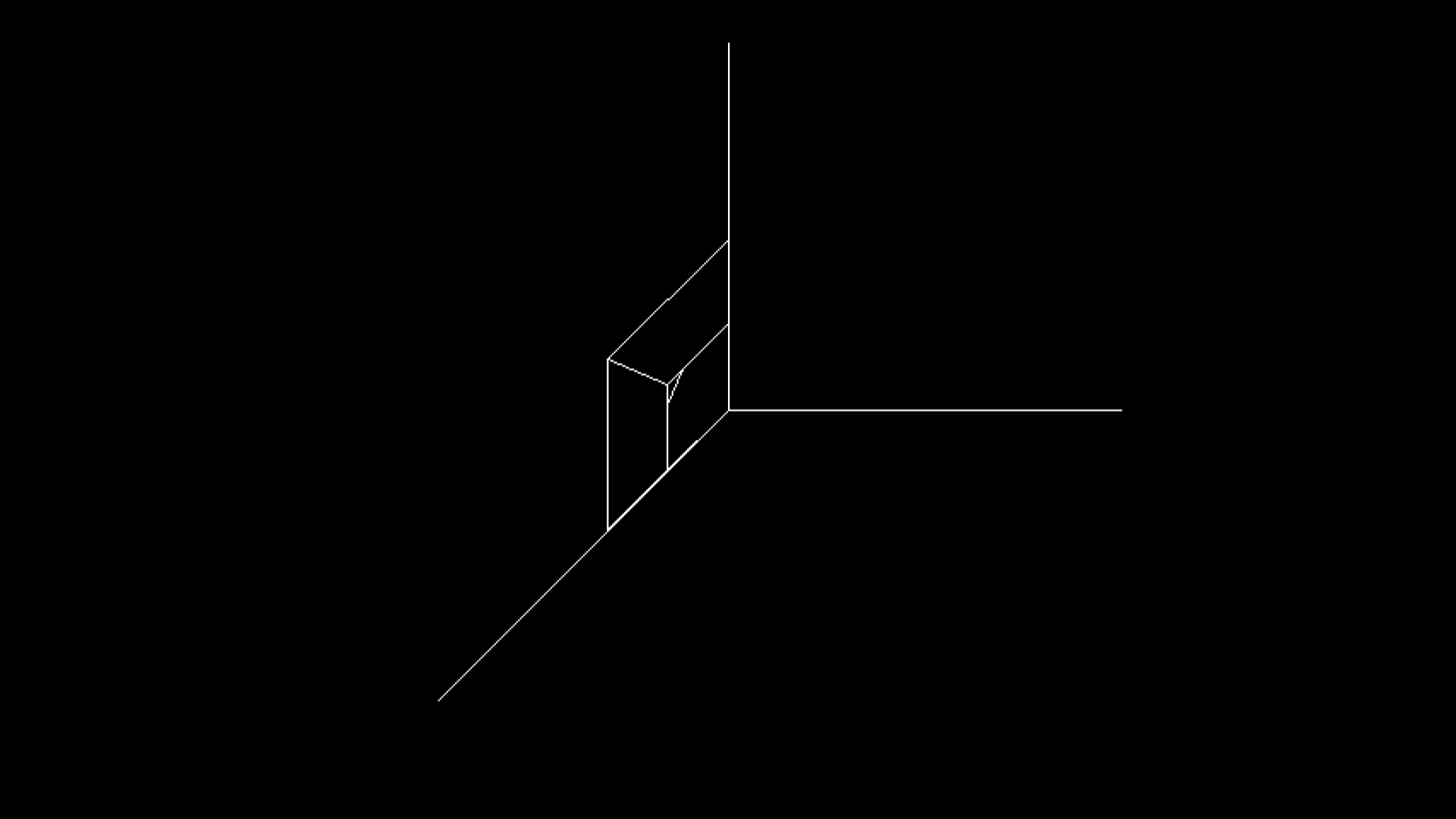
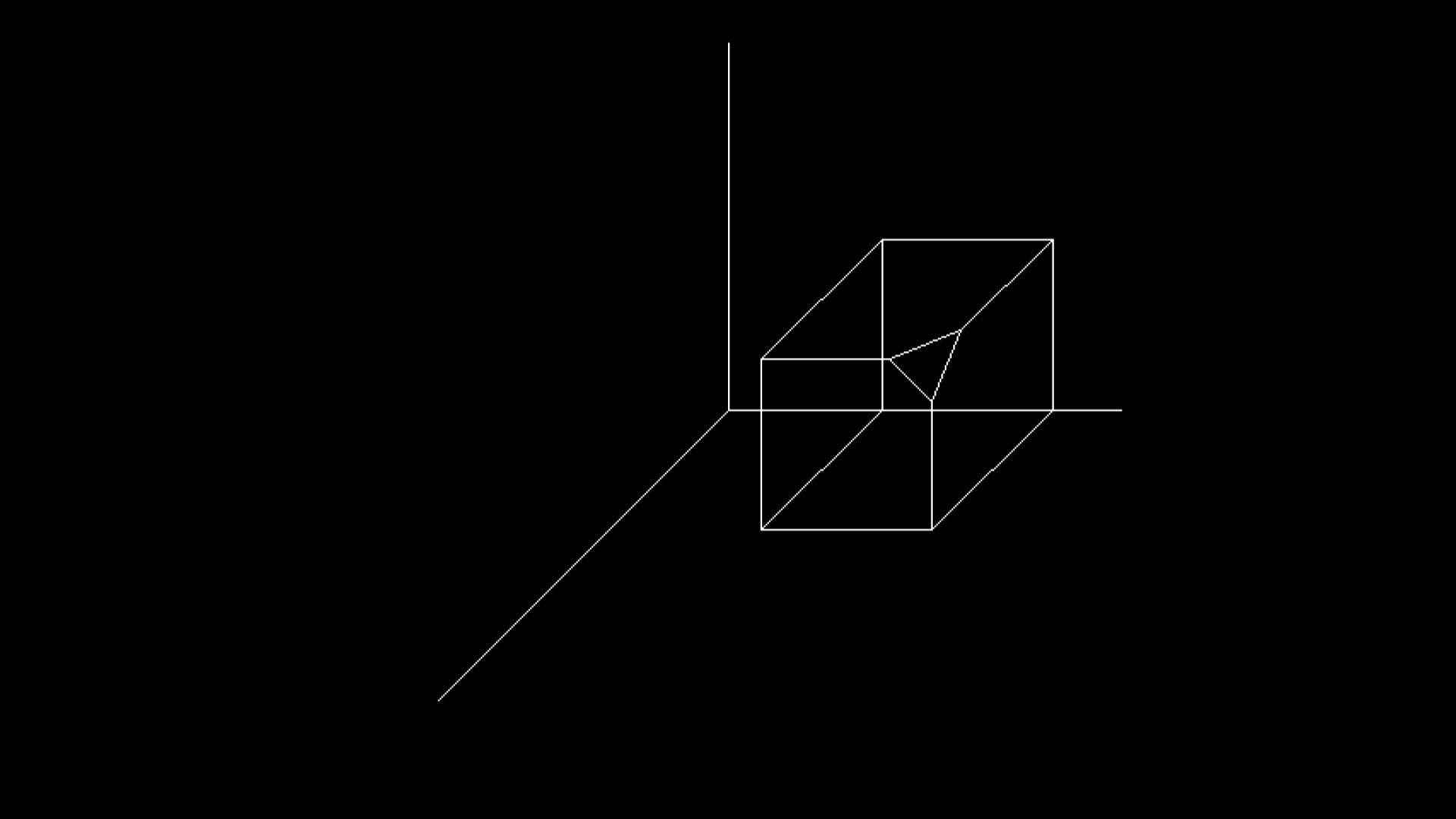
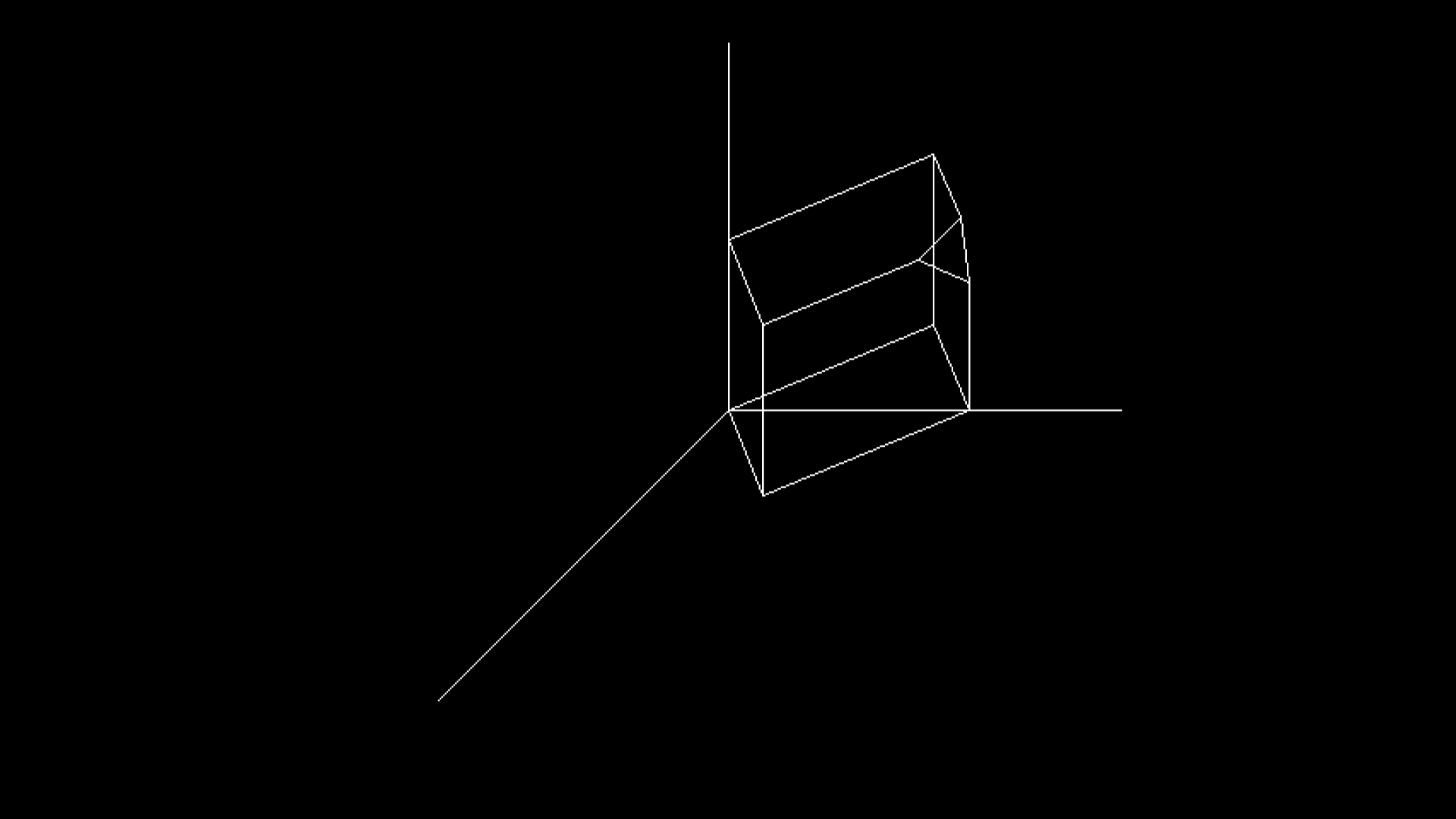
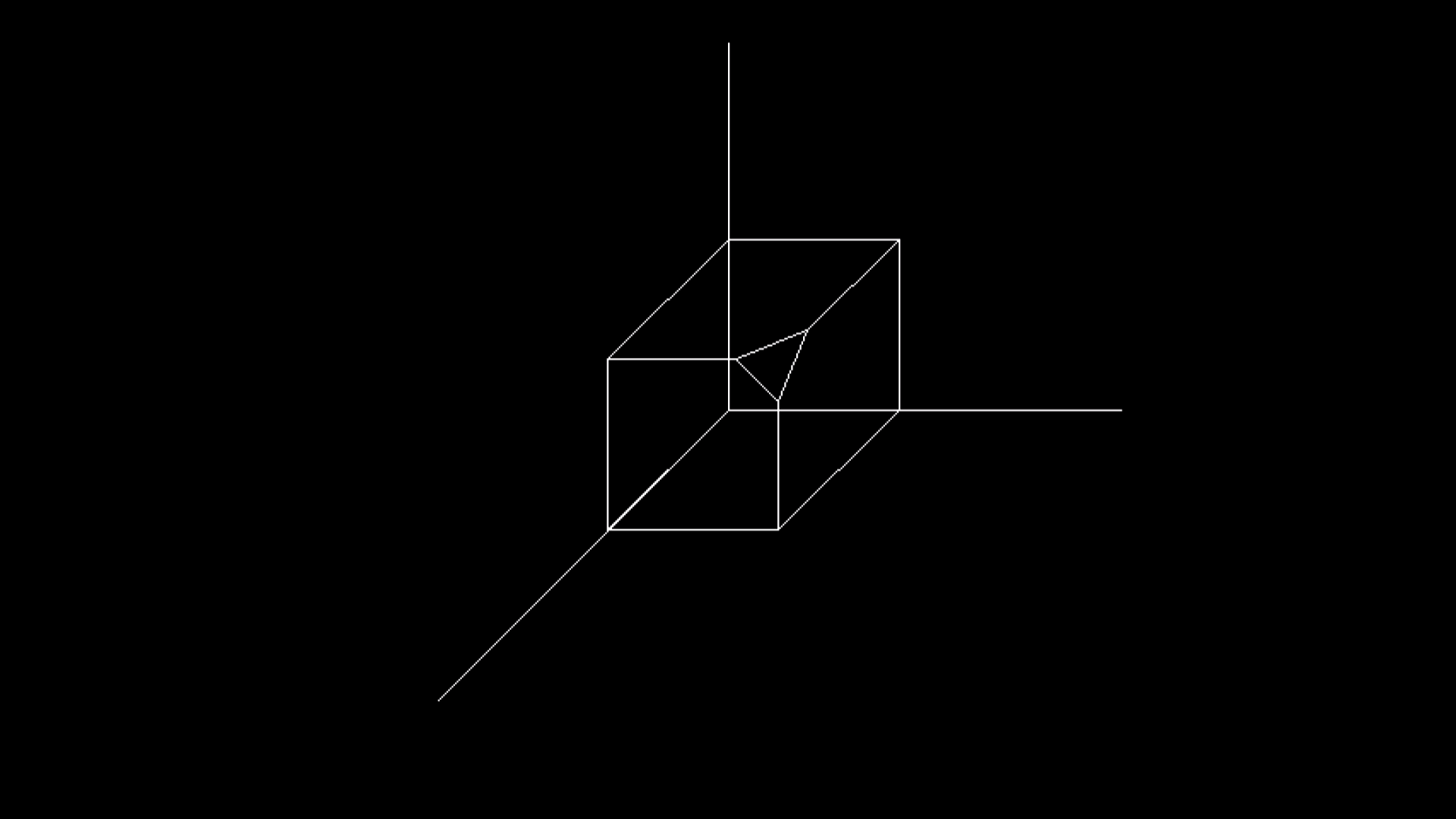
break;

default: cout<<" Invalid Choice\n ";

}

getch();

}



Q8. Write a program to draw Hermite and Bezier curve implement both.

Program

#include<iostream.h>

#include<conio.h>

#include<graphics.h>

#include<math.h>

voidbezier\_curve(int x[4],int y[4]){

for(double t=0.0;t<1.0;t=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,YELLOW);

}

for(inti=0;i<3;i++){

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

}

}

Void hermite\_curve(int x1,int y1,int x2,int y2,double t1,double

t4){

Float x,y,t;

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

x=(2\*t\*t\*t-3\*t\*t+1)\*x1+(-2\*t\*t\*t+3\*t\*t)\*x2+(t\*t\*t-

2\*t\*t+t)\*t1+(t\*t\*t-t\*t)\*t4;

y=(2\*t\*t\*t-3\*t\*t+1)\*y1+(-2\*t\*t\*t+3\*t\*t)\*y2+(t\*t\*t-

2\*t\*t+1)\*t1+(t\*t\*t-t\*t)\*t4;

putpixel(x,y,YELLOW);

}

putpixel(x1,y1,GREEN);

putpixel(x2,y2,GREEN);

line(x1,y1,x2,y2);

}

Void main()

{

clrscr();

Int gdriver=DETECT,gmode,errorcode;

intx1,y1,x2,y2,n;

Double t1,t4;

initgraph(&gdriver,&gmode,"C:\\TURBOC3\\BGI");

intx[4],y[4];

inti;

cout<<"1.BezierCurve \n 2.HermiteCurve\n";

cout<<"Enter your choice:";

cin>>n;

if(n==1){

cout<<"Enter x and y coordinates\n";

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

cout<<"x"<<i+1<<":";

cin>>x[i];

cout<<"y"<<i+1<<":";

cin>>y[i];

cout<<endl;

}

bezier\_curve(x,y);

}

elseif(n==2){

cout<<"Enter the x coordinate of 1st hermite point:";

cin>>x1;

cout<<"Enter the y coordinate of 1st hermite point:";

cin>>y1;

cout<<"Enter the x coordinate of 4th hermite point:";

cin>>x2;

cout<<"Enter the y coordinate of 4th hermite point:";

cin>>y2;

cout<<"Enter tangent at p1:";

cin>>t1;

cout<<"Enter tangent at p4:";

cin>>t4;

hermite\_curve(x1,y1,x2,y2,t1,t4);

}

else{

cout<<"\nInvalid Choice";

}

getch();

}

