

**Index**

| **S.No.** | **Topic** |
| --- | --- |
| 1. | Bresenham’s Line |
| 2. | Mid-Point Circle |
| 3. | Cohen and Sutherland line clipping |
| 4. | Sutherland Hodgeman Polygon Clipping |
| 5. | Scan Line fill |
| 6. | 2D Transformations |
| 7. | 3D Transformations |
| 8. | Hermite Curve |
| 9. | Bezier Curve |

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

**Source Code:**

*#include*<stdio.h>

*#include*<conio.h>

*#include*<graphics.h>

*#include*<math.h>

*#include*<iostream.h>

int x0,x1,y0,y1;

void input()

{

cout<<"Enter x0: ";

cin>>x0;

cout<<"Enter y0: ";

cin>>y0;

cout<<"Enter x1: ";

cin>>x1;

cout<<"Enter y1: ";

cin>>y1;

}

void midpoint()

{

int dy=y1-y0;

int dx=x1-x0;

int d=dy-dx/2;

int de=dy;

int dne=dy-dx;

int x= x0;

int y=y0;

*while*(x<=x1)

{

putpixel(x,y,WHITE);

*if*(d>0)

{

x=x+1;

y=y+1;

d=d+dne;

}

*else*

{

x=x+1;

d=d+de;

}

}

}

void main()

{

int gd=DETECT, gm;

clrscr();

input();

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

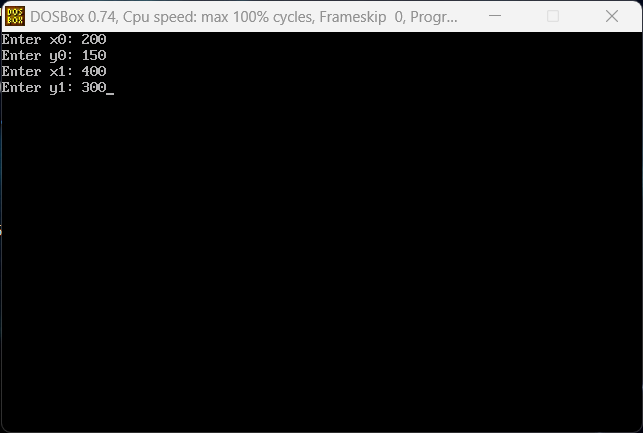
midpoint();

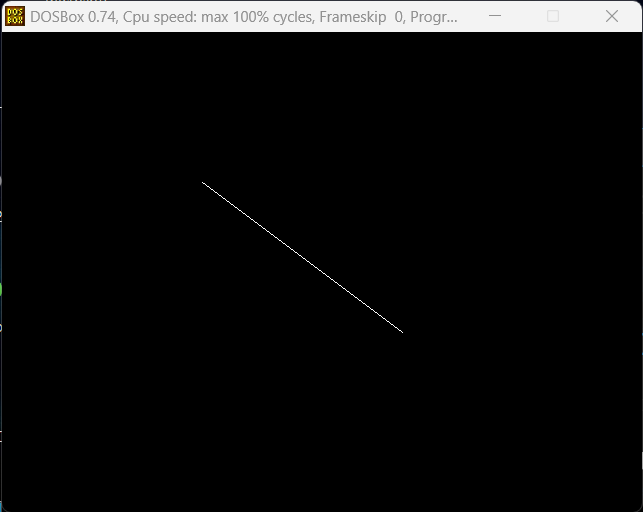
getch();

closegraph();

}

**Output:-**





Q2. Write a program to implement mid-point circle drawing algorithm.

**Source Code:**

*#include* <math.h>

*#include* <stdlib.h>

*#include* <graphics.h>

*#include* <iostream.h>

*#include* <dos.h>

void drawCirclePoints(int x, int y, int val, int c\_x, int c\_y)

{

putpixel(c\_x + x, c\_y + y, val);

putpixel(c\_x + y, c\_y + x, val);

putpixel(c\_x + y, c\_y + -x, val);

putpixel(c\_x + x, c\_y + -y, val);

putpixel(c\_x + -x, c\_y + -y, val);

putpixel(c\_x + -y, c\_y + -x, val);

putpixel(c\_x + -y, c\_y + x, val);

putpixel(c\_x + -x, c\_y + y, val);

*return*;

}

void midpointCircle(int r, int val, int c\_x = 0, int c\_y = 0)

{

int x = 0;

int y = r;

int d = 1 - r;

drawCirclePoints(x, y, val, c\_x, c\_y);

*while* (y > x)

{

*if* (d < 0)

{

d += 2 \* x + 3;

x += 1;

}

*else*

{

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

x += 1;

y -= 1;

}

drawCirclePoints(x, y, val, c\_x, c\_y);

}

*return*;

}

int main(void)

{

int x, y, r;

cout << "Enter Centre (x y): ";

cin >> x >> y;

cout << "Enter Radius (r): ";

cin >> r;

cout << "Drawing Circle..." << endl;

int gd = DETECT, gm;

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

midpointCircle(r, WHITE, x, y);

delay(5000);

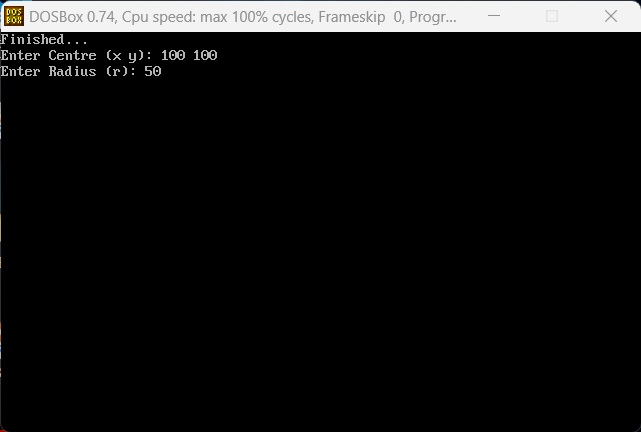
closegraph();

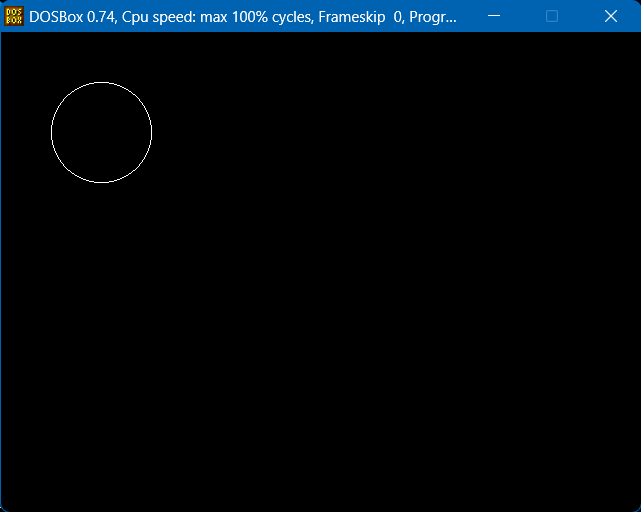
cout << "Finished..." << endl;

*return* 0;

}

**Output:-**





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

**Source Code:**

*#include*<conio.h>

*#include*<stdio.h>

*#include*<graphics.h>

*#include*<math.h>

*#include*<iostream.h>

void main()

{

int a[4],b[4];

float m,xnew,ynew;

float xl=100,yl=100,xh=300,yh=300,xa=10,ya=200,xb=250,yb=150;

int gd = DETECT,gm;

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

setcolor(5);

line(xa,ya,xb,yb);

setcolor(12);

rectangle(xl,yl,xh,yh);

m = (yb-ya)/(xb-xa);

*if*(xa < xl)

a[3] = 1;

*else* a[3] = 0;

*if*(xa>xh)

a[2] = 1;

*else* a[2] = 0;

*if*(ya < yl)

a[1] = 1;

*else* a[1] = 0;

*if* (ya > yh)

a[0] = 1;

*else* a[0] = 0;

*if*(xb < xl)

b[3] = 1;

*else* b[3] = 0;

*if*(xb>xh)

b[2] = 1;

*else* b[2] = 0;

*if*(yb < yl)

b[1] = 1;

*else* b[1] = 0;

*if* (yb > yh)

b[0] = 1;

*else* b[0] = 0;

cout<<"press a key to continue";

getch();

*if*(a[0] == 0 && a[1] == 0 && a[2] == 0 && a[3] == 0 && b[0] == 0 && b[1] == 0 && b[2] == 0 && b[3] == 0 )

{

cout<<"no clipping";

line(xa,ya,xb,yb);

}

*else* *if*(a[0]&&b[0] || a[1]&&b[1] || a[2]&&b[2] || a[3]&&b[3])

{

clrscr();

cout<<"line discarded";

rectangle(xl,yl,xh,yh);

}

*else*

{

*if*(a[3] == 1 && b[3]==0)

{

ynew = (m \* (xl-xa)) + ya;

setcolor(12);

rectangle(xl,yl,xh,yh);

setcolor(0);

line(xa,ya,xb,yb);

setcolor(15);

line(xl,ynew,xb,yb);

}

*else* *if*(a[2] == 1 && b[2] == 0)

{

ynew = (m \* (xh-xa)) + ya;

setcolor(12);

rectangle(xl,yl,xh,yh);

setcolor(0);

line(xa,ya,xb,yb);

setcolor(15);

line(xl,ynew,xb,yb);

}

*else* *if*(a[1] == 1 && b[1] == 0)

{

xnew = xa + (yl-ya)/m;

setcolor(0);

line(xa,ya,xb,yb);

setcolor(15);

line(xnew,yh,xb,yb);

}

*else* *if*(a[0] == 1 && b[0] == 0)

{

xnew = xa + (yh-ya)/m;

setcolor(0);

line(xa,ya,xb,yb);

setcolor(15);

line(xnew,yh,xb,yb);

}

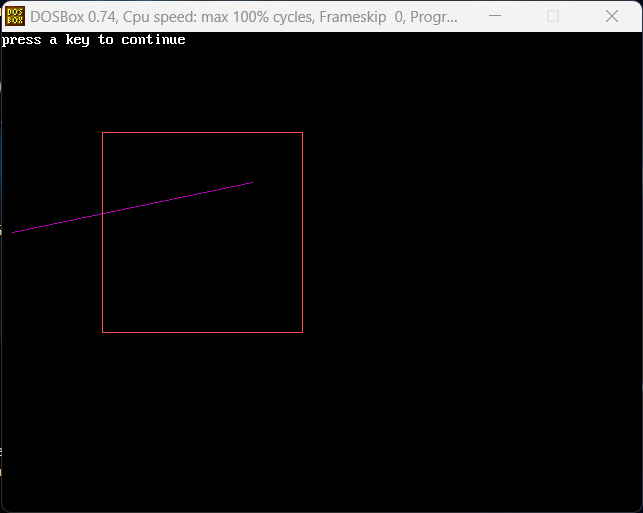
}

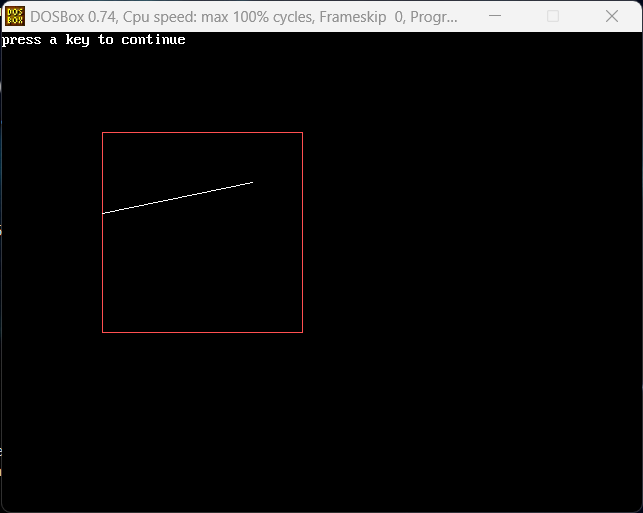
getch();

closegraph();

}

**Output:-**





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

**Source Code:**

*#include* <conio.h>

*#include* <graphics.h>

*#include* <iostream.h>

*#include* <math.h>

*#include* <stdio.h>

*#include* <stdlib.h>

typedef unsigned int outcode;

outcode compOutcode(double x, double y);

enum

{

TOP = 0x1,

BOTTOM = 0x2,

RIGHT = 0x4,

LEFT = 0x8

};

double xmin, xmax, ymin, ymax;

outcode compOutcode(double x, double y)

{

outcode code = 0;

*if* (y > ymax)

code |= TOP;

*else* *if* (y < ymin)

code |= BOTTOM;

*if* (x > xmax)

code |= RIGHT;

*else* *if* (x < xmin)

code |= LEFT;

*return* code;

}

void clipPolygon(int x0, int y0, int x1, int y1)

{

int accept = 0, done = 0;

outcode outcode0, outcode1, outcodeOut;

outcode0 = compOutcode(x0, y0);

outcode1 = compOutcode(x1, y1);

*do*

{

*if* (!(outcode0 | outcode1))

{

accept = 1;

done = 1;

}

*else* *if* (outcode0 & outcode1)

done = 1;

*else*

{

double x, y;

outcodeOut = outcode0 ? outcode0 : outcode1;

*if* (outcodeOut & TOP)

{

x = x0 + (x1 - x0) \* (ymax - y0) / (y1 - y0);

y = ymax;

}

*else* *if* (outcodeOut & BOTTOM)

{

x = x0 + (x1 - x0) \* (ymin - y0) / (y1 - y0);

y = ymin;

}

*else* *if* (outcodeOut & RIGHT)

{

y = y0 + (y1 - y0) \* (xmax - x0) / (x1 - x0);

x = xmax;

}

*else*

{

y = y0 + (y1 - y0) \* (xmin - x0) / (x1 - x0);

x = xmin;

}

*if* (outcodeOut == outcode0)

{

x0 = x;

y0 = y;

outcode0 = compOutcode(x0, y0);

}

*else*

{

x1 = x;

y1 = y;

outcode1 = compOutcode(x1, y1);

}

}

} *while* (done == 0);

*if* (accept)

line(x0, y0, x1, y1);

}

void main()

{

int i, n;

int gd = DETECT, gm;

int poly[24];

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

cout << "Enter Bounds of Clipping Rectangle: ";

cout << "\n\txmin: ";

cin >> xmin;

cout << "\tymin: ";

cin >> ymin;

cout << "\txmax: ";

cin >> xmax;

cout << "\tymax: ";

cin >> ymax;

cout << "Enter Number of Edges in Polygon: ";

cin >> n;

cout << "Enter Coordinates of the Polygon: ";

*for* (i = 0; i < 2 \* n; i++)

cin >> poly[i];

poly[2 \* n] = poly[0];

poly[2 \* n + 1] = poly[1];

cleardevice();

rectangle(xmin, ymin, xmax, ymax);

drawpoly(n + 1, poly);

getch();

cleardevice();

rectangle(xmin, ymin, xmax, ymax);

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

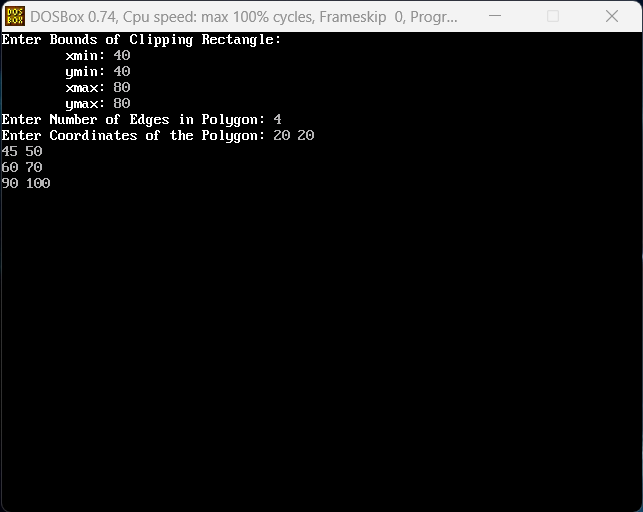
clipPolygon(poly[2 \* i], poly[(2 \* i) + 1], poly[(2 \* i) + 2], poly[(2 \* i) + 3]);

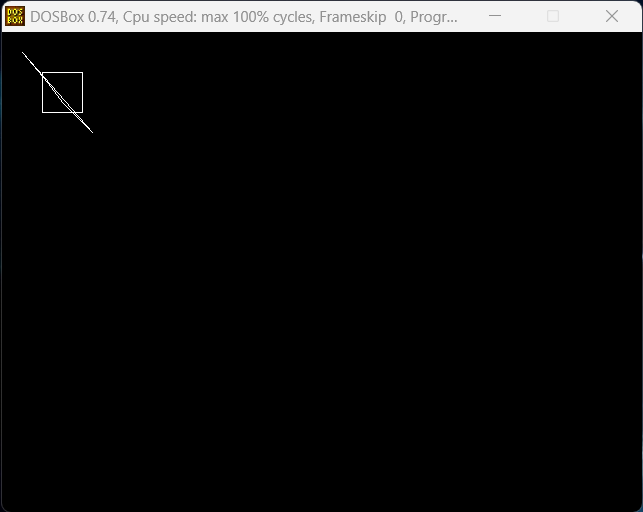
getch();

closegraph();

}

**Output:-**







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

**Source Code:**

*// Scan line Polygon filling algo*

*#include*<iostream.h>

*#include*<conio.h>

*#include*<graphics.h>

*#include*<dos.h>

void main()

{

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

int x,y,temp;

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

float slope[20];

clrscr();

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

cin>>n;

cout<<"Enter the coordinates of polygon:\n\n";

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

{

cout << "X" << i << " Y" << i<<" : ";

cin >> a[i][0]>>a[i][1];

}

a[i][0]=a[0][0];

a[i][1]=a[0][1];

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

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

{

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

}

getch();

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

{

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

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

*if*(dy==0) slope[i]=1;

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

*if*((dy!=0)&&(dx!=0))

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

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

{

k=0;

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

{

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

{

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

k++;

}

}

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

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

{

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

{

temp=xi[i];

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

xi[i+1]=temp;

}

}

setcolor(2);

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

{

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

delay(20);

}

} *// for loop 480*

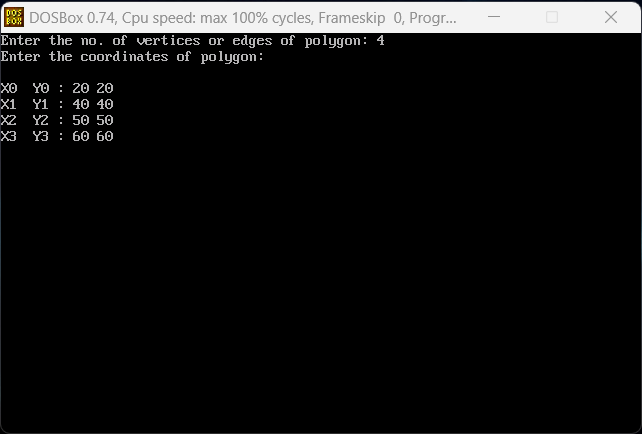
getch();

closegraph();

}

}

**Output:-**







Q6. Write a program to apply various 2D transformations on a 2D object(Use homogeneous Coordinates).

**Source Code:**

*#include*<iostream.h>

*#include*<stdio.h>

*#include*<conio.h>

*#include*<graphics.h>

*#include*<string.h>

*#include*<math.h>

void main()

{

int gd=DETECT, gm, ch;

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

cleardevice();

cout<<"\t1. Scaling-enlargement \n\n";

cout<<"\t2. Scaling-shrinking \n\n";

cout<<"\t3. Translation in x \n\n";

cout<<"\t4. Translation in y \n\n";

cout<<"\t5. Translation in x & y both \n\n";

cout<<"\t6. Reflection through x axis \n\n";

cout<<"\t7. Reflection through y axis \n\n";

cout<<"\t8. Reflection through x=y axis \n\n";

cout<<"\t9. Rotation wrt origin \n\n";

cout<<"\t10. Shearing in x \n\n";

cout<<"\t11. Shearing in y \n\n";

cout<<"\t12. Exit()\n\n";

cout<<"Enter choice: ";

cin>>ch;

*switch*(ch)

{

*case* 1:

{

int x1=30,y1=30,x2=70,y2=70,x=2,y=2;

cleardevice();

cout<<"\nRectangle before scaling-enlargement:\n";

rectangle(x1,y1,x2,y2);

getch();

cleardevice();

cout<<"\n\n\nrectangle after scaling-enlargement:\n";

rectangle(x1\*x,y1\*y,x2\*x,y2\*y);

getch();

main();

}

*case* 2:

{

int x1=30,y1=30,x2=70,y2=70,x=2,y=2;

cleardevice();

cout<<"\nRectangle before scaling-shrinking:\n";

rectangle(x1,y1,x2,y2);

getch();

cleardevice();

cout<<"\n\n\nrectangle after scaling-shrinking:\n";

rectangle(x1/x,y1/y,x2/x,y2/y);

getch();

main();

}

*case* 3:

{

int tx=50,ty=50,x1=100,x2=230,y1=100,y2=70;

cleardevice();

cout<<"Rectangle before translation in x:\n";

rectangle(x1,y1,x2,y2);

getch();

cleardevice();

cout<<"Rectangle after translation in x:\n";

rectangle(x1+tx,y1,x2+tx,y2);

getch();

main();

}

*case* 4:

{

int tx=50,ty=50,x1=100,x2=230,y1=100,y2=70;

cleardevice();

cout<<"Rectangle before translation in y:\n";

rectangle(x1,y1,x2,y2);

getch();

cleardevice();

cout<<"Rectangle after translation in y:\n";

rectangle(x1,y1+ty,x2,y2+ty);

getch();

main();

}

*case* 5:

{

int tx=50,ty=50,x1=100,x2=230,y1=100,y2=70;

cleardevice();

cout<<"Rectangle before translation in x & y both:\n";

rectangle(x1,y1,x2,y2);

getch();

cleardevice();

cout<<"Rectangle after translation in x & y both:\n";

rectangle(x1+tx,y1+ty,x2+tx,y2+ty);

getch();

main();

}

*case* 6:

{

int x1=50,y1=150,x2=75,y2=125,x3=100,y3=150,xt;

cleardevice();

cout<<"\n\n\nTriangle before reflection through x axis:\n";

line(x1,y1,x2,y2);

line(x1,y1,x3,y3);

line(x2,y2,x3,y3);

getch();

cleardevice();

cout<<"\n\n\nTriangle after reflection through x axis:\n";

line(-x1+200,y1,-x2+200,y2);

line(-x1+200,y1,-x3+200,y3);

line(-x2+200,y2,-x3+200,y3);

getch();

main();

}

*case* 7:

{

int x1=50,y1=150,x2=75,y2=125,x3=100,y3=150,xt;

cleardevice();

cout<<"\n\n\nTriangle before reflection through y axis:\n";

line(x1,y1,x2,y2);

line(x1,y1,x3,y3);

line(x2,y2,x3,y3);

getch();

cleardevice();

cout<<"\n\n\nTriangle after reflection through y axis:\n";

line(x1,-y1+200,x2,-y2+200);

line(x1,-y1+200,x3,-y3+200);

line(x2,-y2+200,x3,-y3+200);

getch();

main();

}

*case* 8:

{

int x1=50,y1=150,x2=75,y2=125,x3=100,y3=150,xt;

cleardevice();

cout<<"\n\n\nTriangle before reflection through x=y axis:\n";

line(x1,y1,x2,y2);

line(x1,y1,x3,y3);

line(x2,y2,x3,y3);

getch();

cleardevice();

cout<<"\n\n\nTriangle after reflection through x=y axis:\n";

line(-x1+200,-y1+200,-x2+200,-y2+200);

line(-x1+200,-y1+200,-x3+200,-y3+200);

line(-x2+200,-y2+200,-x3+200,-y3+200);

getch();

main();

}

*case* 9:

{

long x1=100,y1=100,x2=200,y2=200;

double d1,xt,yt;

cleardevice();

cout<<"\n\n\nEnter angle of rotation: ";

cin>>d1;

d1=(d1\*3.14)/180.0;

xt=x1+((x2-x1)\*cos(d1)-(y2-y1)\*sin(d1));

yt=y1+((x2-x1)\*sin(d1)+(y2-y1)\*cos(d1));

line(x1,y1,x2,y2);

getch();

cleardevice();

cout<<"\nLine after rotation";

line(x1,y1,(int)xt,(int)yt);

getch();

main();

}

*case* 10:

{

int x1=100,x2=100,y1=100,y2=30,x3=170,y3=30,x4=170,y4=40,shx=5;

cleardevice();

cout<<"\n\n\nRectangle before shearing in x:\n";

line(x1,y1,x2,y2);

line(x1,y1,x4,y4);

line(x2,y2,x3,y3);

line(x3,y3,x4,y4);

getch();

cleardevice();

cout<<"\n\n\nRectangle after shearing in x:\n";

line(x1+shx\*y1,y1,x2+shx\*y2,y2);

line(x1+shx\*y1,y1,x4+shx\*y4,y4);

line(x2+shx\*y2,y2,x3+shx\*y3,y3);

line(x3+shx\*y3,y3,x4+shx\*y4,y4);

getch();

main();

}

*case* 11:

{

int x1=60,x2=60,y1=100,y2=30,x3=80,y3=30,x4=80,y4=40,shx=5;

cleardevice();

cout<<"\n\n\nRectangle before shearing in y:\n";

line(x1,y1,x2,y2);

line(x1,y1,x4,y4);

line(x2,y2,x3,y3);

line(x3,y3,x4,y4);

getch();

cleardevice();

cout<<"\n\n\nRectangle after shearing in y:\n";

line(x1,y1+shx\*x1,x2,y2+shx\*x2);

line(x1,y1+shx\*x1,x4,y4+shx\*x4);

line(x2,y2+shx\*x2,x3,y3+shx\*x3);

line(x3,y3+shx\*x3,x4,y4+shx\*x4);

getch();

main();

}

*case* 12:

{

cout<<"THANK YOU!!";

*break*;

}

*default*:

{

cout<<"Enter correct choice!!";

main();

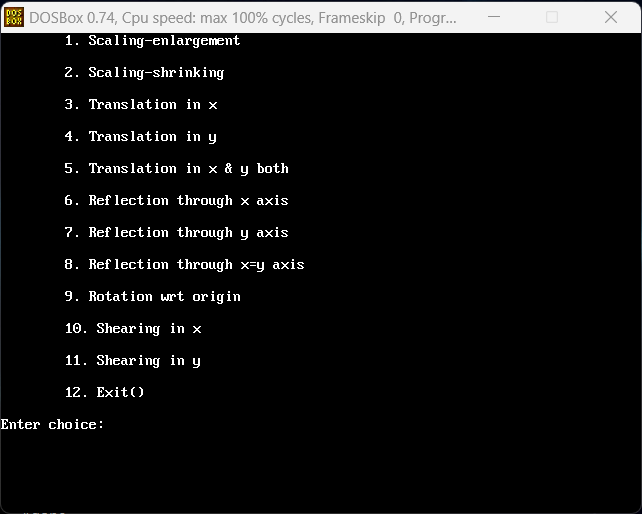
}

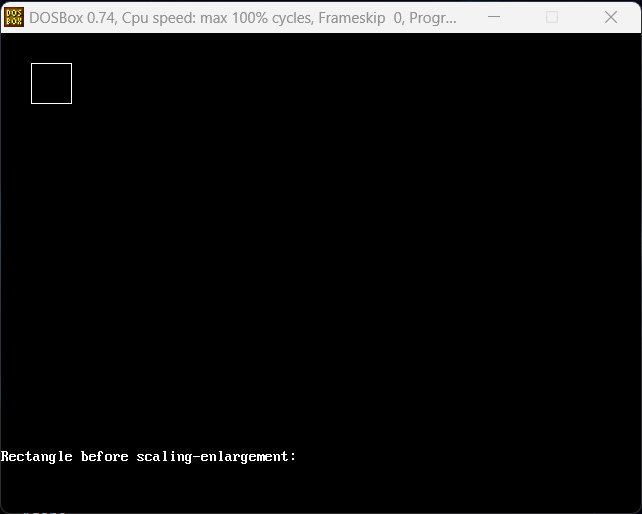
}

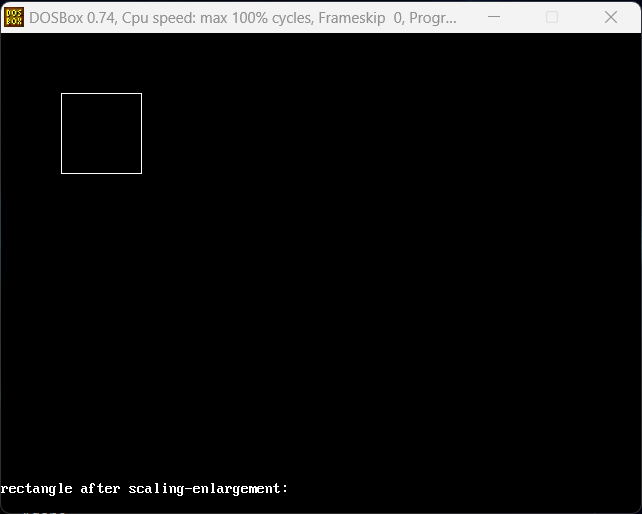
closegraph();

}

**Output:-**







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

**Source Code:**

*// Program for 3-D Transformation*

*#include*<iostream.h>

*#include*<dos.h>

*#include*<stdio.h>

*#include*<math.h>

*#include*<conio.h>

*#include*<graphics.h>

*#include*<process.h>

int gd=DETECT,gm;

double x1,x2,y1,y2;

void show\_message()

{

char \*mess[]={"-","=","["," ","3","D","-","T","r","a","n","s",

"f","o","r","m","a","t","i","o","n"," ","]","=","-"};

int xx=28,xxx=52,i,j;

\_setcursortype(\_NOCURSOR);

*for*(i=0,j=24;i<15,j>=12;i++,j--)

{

gotoxy(xx,1);

cout<<mess[i];

xx++;

gotoxy(xxx,1);

cout<<mess[j];

xxx--;

delay(50);

}

\_setcursortype(\_NORMALCURSOR);

}

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

{

initgraph(&gd,&gm,"..\bgi");

int i;

clearviewport();

*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);

getch();

closegraph();

}

void scale(double edge[20][3])

{

double a,b,c;

int i;

cout<<" Enter The Scaling Factors :=";

cin>>a>>b>>c;

initgraph(&gd,&gm,"..\bgi");

clearviewport();

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

{

edge[i][0]=edge[i][0]\*a;

edge[i][1]=edge[i][1]\*b;

edge[i][2]=edge[i][2]\*c;

}

draw\_cube(edge);

closegraph();

}

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

{

int a,b,c;

int i;

cout<<" Enter The Translation Factors :=";

cin>>a>>b>>c;

initgraph(&gd,&gm,"..\bgi");

clearviewport();

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

{

edge[i][0]+=a;

edge[i][0]+=b;

edge[i][0]+=c;

}

draw\_cube(edge);

closegraph();

}

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

{

int ch;

int i;

double temp,theta,temp1;

clrscr();

cout<<"-=[ Reflection About ]=-";

cout<<"1:==> X-Axis ";

cout<<"2:==> Y-Axis ";

cout<<"3:==> Z-Axis ";

cout<<" Enter Your Choice :=";

cin>>ch;

*switch*(ch)

{

*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 ch;

int i;

clrscr();

cout<<"-=[ Reflection About ]=-";

cout<<"1:==> X-Axis ";

cout<<"2:==> Y-Axis ";

cout<<"3:==> Z-Axis ";

cout<<" Enter Your Choice :=";

cin>>ch;

*switch*(ch)

{

*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 ch;

int i;

double p,q,r;

clrscr();

cout<<" -=[ Perspective Projection About ]=-";

cout<<"1:==> X-Axis ";

cout<<"2:==> Y-Axis ";

cout<<"3:==> Z-Axis ";

cout<<"Enter Your Choice :=";

cin>>ch;

*switch*(ch)

{

*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*;

}

closegraph();

}

void main()

{

int gd = DETECT , gm ;

clrscr();

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

int choice;

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

};

*while*(1)

{

clrscr();

show\_message();

cout<<"\n1:==> Draw Cube ";

cout<<"\n2:==> Scaling ";

cout<<"\n3:==> Rotation ";

cout<<"\n4:==> Reflection ";

cout<<"\n5:==> Translation ";

cout<<"\n6:==> Perspective Projection ";

cout<<"\n7:==> Exit ";

cout<<"\n\n Enter Your Choice :=";

cin>>choice;

*switch*(choice)

{

*case* 1:

draw\_cube(edge);

*break*;

*case* 2:

scale(edge);

*break*;

*case* 3:

rotate(edge);

*break*;

*case* 4:

reflect(edge);

*break*;

*case* 5:

translate(edge);

*break*;

*case* 6:

perspect(edge);

*break*;

*case* 7:

exit(0);

*default*:

cout<<"Press A Valid Key...!!! ";

getch();

*break*;

}

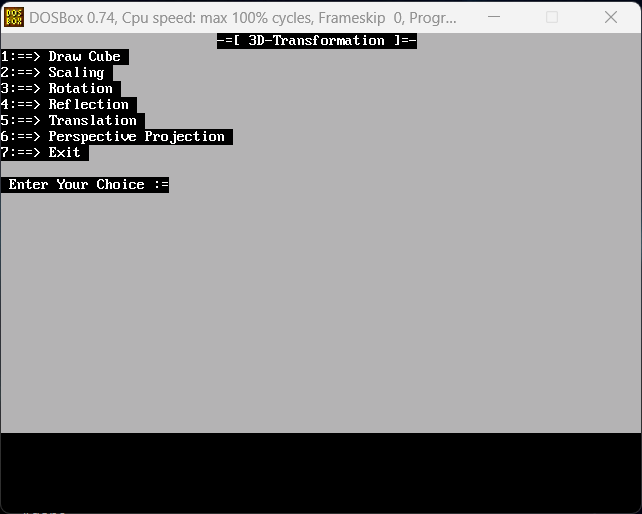
closegraph();

}

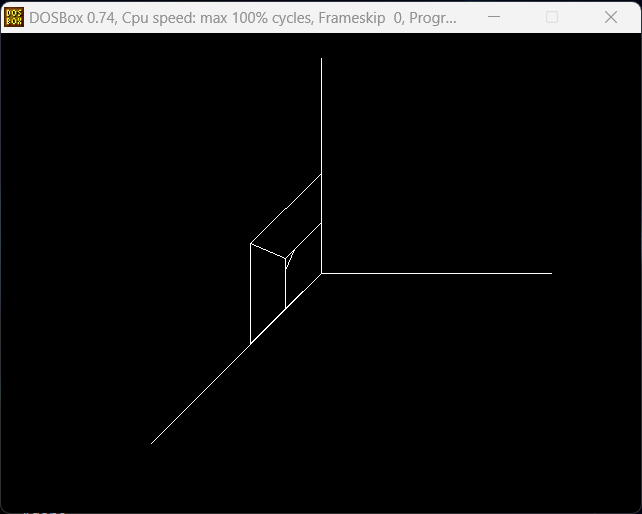
getch();

}

**Output:-**







Q8\_a. Write a program to draw Hermite curve.

**Source Code:**

*#include* <conio.h>

*#include* <graphics.h>

*#include* <iostream.h>

*#include* <math.h>

*#include* <stdio.h>

*#include* <stdlib.h>

struct point

{

int x, y;

};

void hermite(point p1, point p4, double r1, double r4)

{

float x, y, t;

*for* (t = 0.0; t <= 1.0; t += 0.00005)

{

x = (2 \* pow(t, 3) - 3 \* pow(t, 2) + 1) \* p1.x +

(-2 \* pow(t, 3) + 3 \* pow(t, 2)) \* p4.x +

(pow(t, 3) - 2 \* pow(t, 2) + t) \* r1 +

(pow(t, 3) - pow(t, 2)) \* r4;

y = (2 \* pow(t, 3) - 3 \* pow(t, 2) + 1) \* p1.y +

(-2 \* pow(t, 3) + 3 \* pow(t, 2)) \* p4.y +

(pow(t, 3) - 2 \* pow(t, 2) + 1) \* r1 +

(pow(t, 3) - pow(t, 2)) \* r4;

putpixel(x, y, WHITE);

}

circle(p1.x, p1.y, 3);

circle(p4.x, p4.y, 3);

}

void main()

{

point p1, p4;

double r1, r4;

int gd = DETECT, gm;

initgraph(&gd, &gm, "..\\BGI");

cout << "Enter Point 1 (x, y): ";

cin >> p1.x >> p1.y;

cout << "Enter Point 2 (x, y): ";

cin >> p4.x >> p4.y;

cout << "Enter Tangent at Point 1: ";

cin >> r1;

cout << "Enter Tangent at Point 4: ";

cin >> r4;

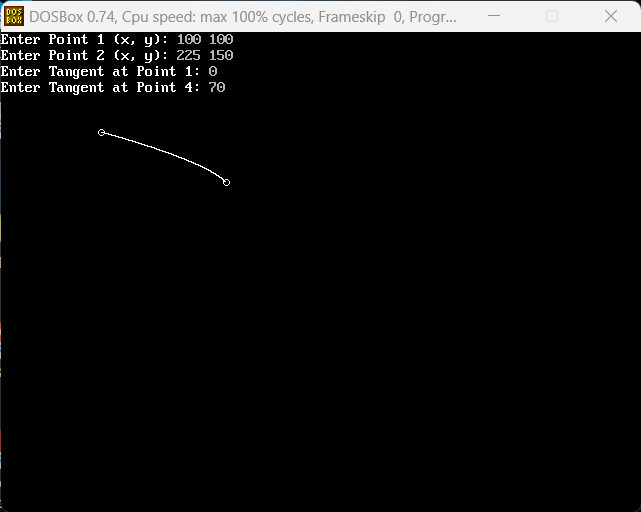
hermite(p1, p4, r1, r4);

getch();

closegraph();

}

**Output:-**



Q8\_b. Write a program to draw Bezier curve.

**Source Code:**

*#include* <conio.h>

*#include* <graphics.h>

*#include* <iostream.h>

*#include* <math.h>

*#include* <stdio.h>

*#include* <stdlib.h>

void bezier(int x[4], int y[4])

{

*for* (double t = 0.0; t < 1.0; t += 0.00005)

{

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

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

putpixel(xt, yt, WHITE);

}

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

{

circle(x[i], y[i], 3);

}

getch();

closegraph();

*return*;

}

void main()

{

int i;

int x[4], y[4];

int gd = DETECT, gm, errorcode;

initgraph(&gd, &gm, "..\\bgi");

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

{

cout << "Enter Point " << i + 1 << " (x, y): ";

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

}

bezier(x, y);

*return*;

}

**Output:-**

