COMPUTER GRAPHICS

PRACTICAL FILE

AMBARISH DEB

20078570008

| S.No | Practical |
| --- | --- |
| 1 | DDA Algorithm |
| 2 | Bresenham Line Algorithm |
| 3 | Bresenham Circle Algorithm |
| 4 | Midpoint Ellipse Algorithm |
| 5 | Cohen Sutherland Line Clipping Alg. |
| 6. | Scan Fill Algorithm |
| 7. | 2 Dimensional Transformations |
| 8. | 3 Dimensional Transformations |

DDA Algorithm

#include <cmath>

#include <cstdlib>

#include <graphics.h>

#include <iostream>

using namespace std;

void ddaLine(int x0, int y0, int x1, int y1, int val)

{

if (x0 == x1 && y0 == y1)

{

putpixel(x1, y1, val);

}

else

{

double x, y;

int dx = x1 - x0;

int dy = y1 - y0;

bool isRTL = !(x1 > x0);

float m = float(dy) / (float)(dx);

if (abs(m) <= 1)

{

if (!isRTL)

{

for (x = x0, y = y0; x <= x1; x++)

{

putpixel(x, y, val);

y += m;

}

}

else

{

for (x = x1, y = y1; x >= x0; x--)

{

putpixel(x, y, val);

y -= m;

}

}

}

else if (abs(m) > 1)

{

if (!isRTL)

{

for (x = x0, y = y0; y <= x1; y++)

{

putpixel(x, y, val);

x += 1 / m;

}

}

else

{

for (x = x1, y = y1; y >= x0; y--)

{

putpixel(x, y, val);

x -= 1 / m;

}

}

}

}

return;

}

int main(void)

{

int x0, y0, x1, y1;

cout << "Enter Left Endpoint (x0 y0): ";

cin >> x0 >> y0;

cout << "Enter Right Endpoint (x1 y1): ";

cin >> x1 >> y1;

cout << "Drawing Line..." << endl;

int gd = DETECT, gm;

initgraph(&gd, &gm, NULL);

ddaLine(x0, y0, x1, y1, WHITE);

delay(10e3);

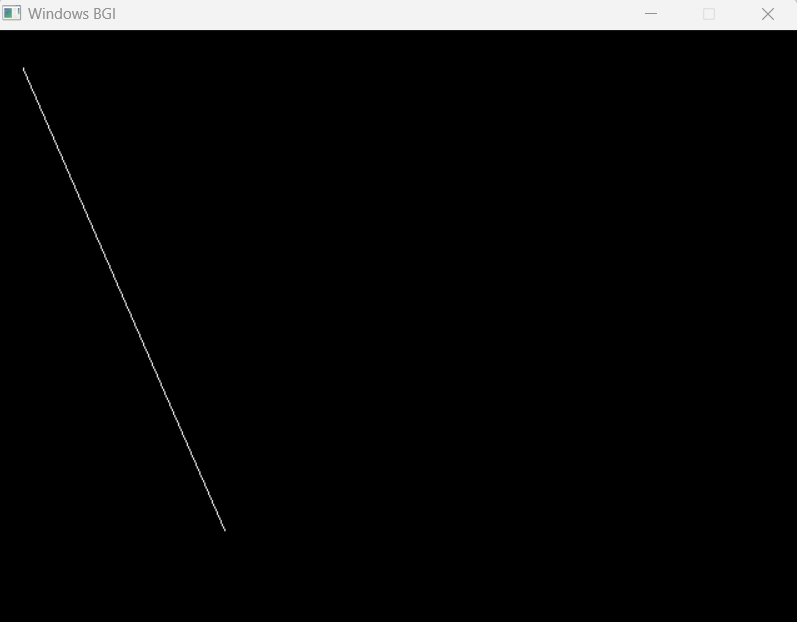
closegraph();

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

return 0;

}

Output-



Bresenham Line Drawing Algorithm

#include <cmath>

#include <cstdlib>

#include <graphics.h>

#include <iostream>

using namespace std;

void bresenhamLine(int x0, int y0, int x1, int y1, int val)

{

if (x0 == x1 && y0 == y1)

{

putpixel(x1, y1, val);

}

else

{

int dx = x1 - x0;

int dy = y1 - y0;

float m = float(dy) / (float)(dx);

if (m >= 1 || m <= 0)

{

cout << "ERROR: Slope must be between 0 and 1." << endl;

exit(1);

}

int d = 2 \* dy - dx;

int del\_E = 2 \* dy;

int del\_NE = 2 \* (dy - dx);

int x = x0;

int y = y0;

putpixel(x, y, val);

while (x < x1)

{

if (d <= 0)

{

d += del\_E;

x += 1;

}

else

{

d += del\_NE;

x += 1;

y += 1;

}

putpixel(x, y, val);

}

}

return;

}

int main(void)

{

int x0, y0, x1, y1;

cout << "Enter Left Endpoint (x0 y0): ";

cin >> x0 >> y0;

cout << "Enter Right Endpoint (x1 y1): ";

cin >> x1 >> y1;

cout << "Drawing Line..." << endl;

int gd = DETECT, gm;

initgraph(&gd, &gm, NULL);

bresenhamLine(x0, y0, x1, y1, WHITE);

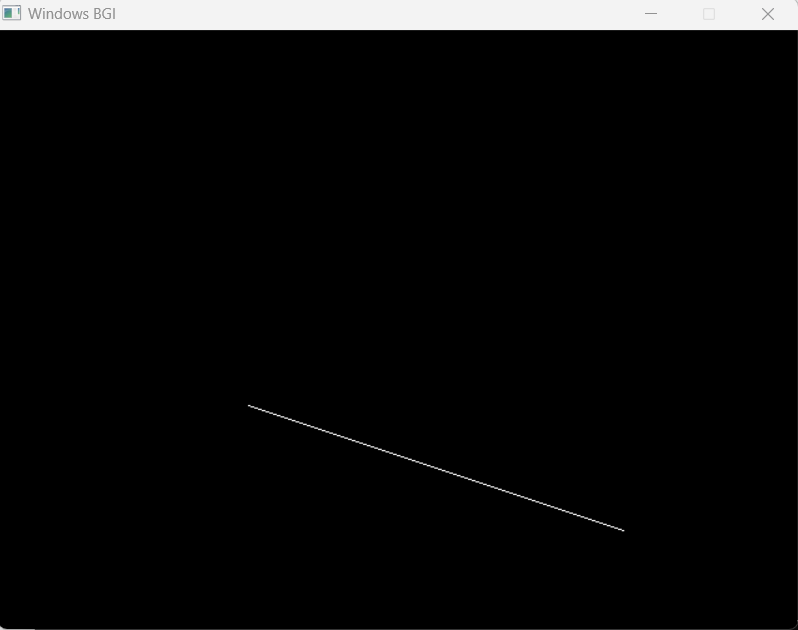
//delay(10000000000000);

closegraph();

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

return 0;

}



Bresenham Circle Drawing Algorithm

#include <stdio.h>

#include <dos.h>

#include <graphics.h>

#include <iostream.h>

void drawCircle(int xc, int yc, int x, int y)

{

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

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

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

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

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

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

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

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

}

void circleBres(int xc, int yc, int r)

{

int x = 0, y = r;

int d = 3 - 2 \* r;

drawCircle(xc, yc, x, y);

while (y >= x)

{

x++;

if (d > 0)

{

y--;

d = d + 4 \* (x - y) + 10;

}

else

d = d + 4 \* x + 6;

drawCircle(xc, yc, x, y);

delay(50);

}

}

int main()

{

int xc, yc, r;

cout<<"Enter radius: ";

cin>>r;

cout<<endl;

cout<<"Enter x and y coordinates separated by a space: ";

cin>>xc>>yc;

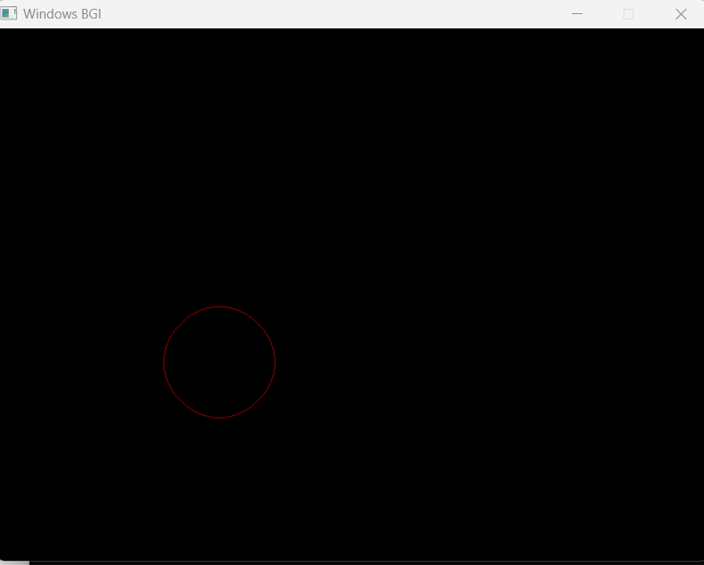
int gd = DETECT, gm;

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

circleBres(xc, yc, r);

return 0;

}



Midpoint Ellipse Algorithm

#include <graphics.h>

#include <stdlib.h>

#include <stdio.h>

#include <conio.h>

#include <alloc.h>

void ellipsemidpoint(int xcenter, int ycenter, int rx, int ry);

void plotpoint(int x,int xcenter, int ycenter,int y);

int maxx, maxy,xcenter,ycenter,rx,ry;

int main(void)

{

int gdriver=DETECT, gmode, errorcode;

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

errorcode = graphresult();

if (errorcode != grOk)

{

printf("Graphics error: %s\n", grapherrormsg(errorcode));

printf("Press any key to halt:");

getch();

exit(1);

}

maxx = getmaxx();

maxy = getmaxy();

xcenter= 250;

ycenter= 250;

rx= 50;

ry= 50;

ellipsemidpoint(xcenter,ycenter,rx,ry);

getch();

closegraph();

return(0);

}

void ellipsemidpoint( int xcenter, int ycenter,int rx, int ry)

{

int p,px,py,x,y,ry2,rx2,tworx2,twory2;

ry2= ry \* ry;

rx2= rx \* rx;

twory2= 2 \* ry2;

tworx2= 2 \* rx2;

x= 0;

y= ry;

plotpoint(x, xcenter, ycenter, y);

p= abs(ry2 - ry2 \* ry + ( 0.25 \* rx2));

px= 0;

py= tworx2 \* y;

while(px<py)

{

x= x+1;

px= px + twory2;

if(p>=0)

{

y= y-1;

py= py - tworx2;

}

if(p<0)

{

p= p + ry2 + px;

}

else

{

p= p + ry2 + px;

}

plotpoint( x, xcenter, ycenter, y);

}

p= abs(ry2 \* (x + 6.5) \* (x + 6.5) + rx2 \* (y - 1) \* (y - 1) - rx2 \* ry2);

while (y>0)

{

y= y-1;

py= py - tworx2;

if(p<0)

{

x= x +1;

px= px + twory2;

}

if(p>0)

{

p= p + rx2 - py;

}

else

{

p= p + rx2 -py +px;

plotpoint(x, xcenter, ycenter, y);

}

}

}

void plotpoint(int x,int xcenter, int ycenter,int y)

{

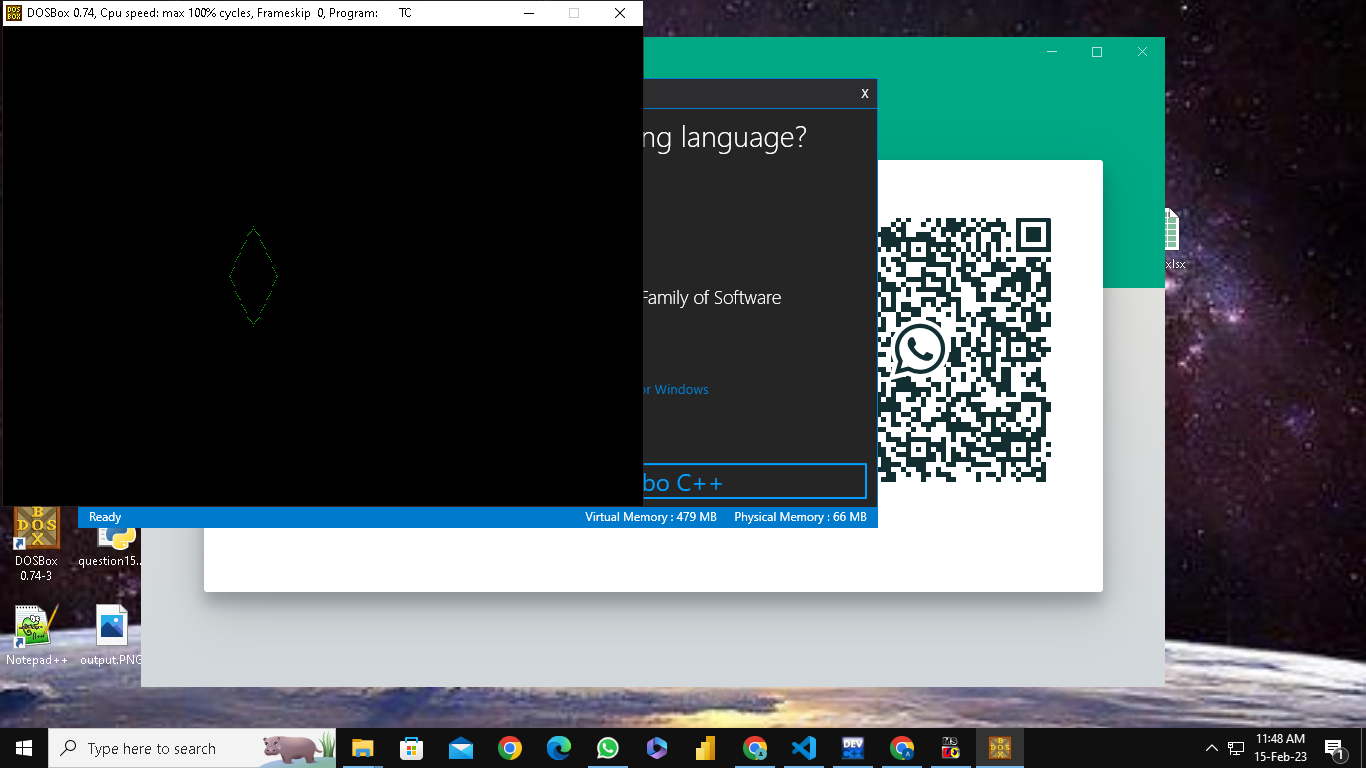
putpixel (xcenter + x, ycenter + y, 2);

putpixel (xcenter - x, ycenter + y, 2);

putpixel (xcenter + x, ycenter - y, 2);

putpixel (xcenter - x, ycenter - y, 2);

}



Cohen Sutherland Line Clipping Algorithm

#include <graphics.h>

// Constants used for calculating the region code of a point

const int INSIDE = 0; // 0000

const int LEFT = 1; // 0001

const int RIGHT = 2; // 0010

const int BOTTOM = 4; // 0100

const int TOP = 8; // 1000

// Function to calculate the region code of a point

int computeRegionCode(int x, int y, int xmin, int ymin, int xmax, int ymax) {

int code = INSIDE;

if (x < xmin) { // to the left of clip window

code |= LEFT;

} else if (x > xmax) { // to the right of clip window

code |= RIGHT;

}

if (y < ymin) { // below the clip window

code |= BOTTOM;

} else if (y > ymax) { // above the clip window

code |= TOP;

}

return code;

}

// Function to clip a line using Cohen-Sutherland algorithm

void clipLine(int x1, int y1, int x2, int y2, int xmin, int ymin, int xmax, int ymax) {

int code1 = computeRegionCode(x1, y1, xmin, ymin, xmax, ymax);

int code2 = computeRegionCode(x2, y2, xmin, ymin, xmax, ymax);

bool accept = false;

while (true) {

if (!(code1 | code2)) { // both endpoints inside clip window

accept = true;

break;

} else if (code1 & code2) { // both endpoints outside clip window, in same region

break;

} else { // partially inside clip window

int x, y;

int codeOut = code1 ? code1 : code2;

if (codeOut & TOP) { // point is above the clip window

x = x1 + (x2 - x1) \* (ymax - y1) / (y2 - y1);

y = ymax;

} else if (codeOut & BOTTOM) { // point is below the clip window

x = x1 + (x2 - x1) \* (ymin - y1) / (y2 - y1);

y = ymin;

} else if (codeOut & RIGHT) { // point is to the right of clip window

y = y1 + (y2 - y1) \* (xmax - x1) / (x2 - x1);

x = xmax;

} else { // point is to the left of clip window

y = y1 + (y2 - y1) \* (xmin - x1) / (x2 - x1);

x = xmin;

}

if (codeOut == code1) { // update first endpoint

x1 = x;

y1 = y;

code1 = computeRegionCode(x1, y1, xmin, ymin, xmax, ymax);

} else { // update second endpoint

x2 = x;

y2 = y;

code2 = computeRegionCode(x2, y2, xmin, ymin, xmax, ymax);

}

}

}

if (accept) {

line(x1, y1, x2, y2); // draw the clipped line

}

}

int main() {

int gd = DETECT, gm;

initgraph(&gd, &gm, ""); // initialize graphics window

// draw the clip window

int xmin = 100, ymin = 100, xmax = 400, ymax = 300;

rectangle(xmin, ymin, xmax, ymax);

// draw the original line

int x1 = 50, y1 = 200, x2 = 500, y2 = 50;

line(x1, y1, x2, y2);

// clip the line and draw the clipped line

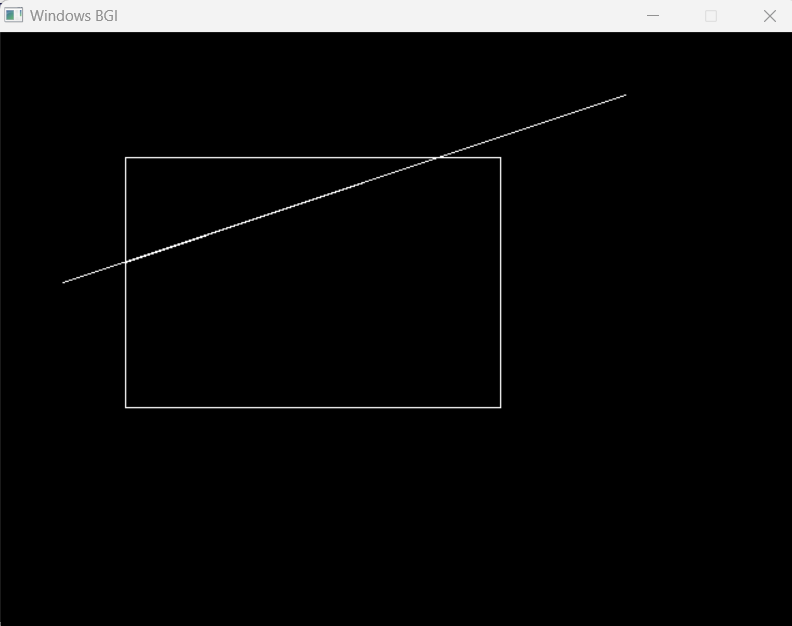
clipLine(x1, y1, x2, y2, xmin, ymin, xmax, ymax);

getch();

closegraph(); // close graphics window

return 0;

}



Scan Fill Algorithm

#include <conio.h>

#include <iostream>

#include <graphics.h>

#include <stdlib.h>

using namespace std;

//Declaration of class point

class point

{

public:

int x,y;

};

class poly

{

private:

point p[20];

int inter[20],x,y;

int v,xmin,ymin,xmax,ymax;

public:

int c;

void read();

void calcs();

void display();

void ints(float);

void sort(int);

};

void poly::read()

{

int i;

cout<<"\n\t SCAN\_FILL ALGORITHM";

cout<<"\n Enter the no of vertices of polygon:";

cin>>v;

if(v>2)

{

for(i=0;i<v; i++) //ACCEPT THE VERTICES

{

cout<<"\nEnter the co-ordinate no.- "<<i+1<<" : ";

cout<<"\n\tx"<<(i+1)<<"=";

cin>>p[i].x;

cout<<"\n\ty"<<(i+1)<<"=";

cin>>p[i].y;

}

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

p[i].y=p[0].y;

xmin=xmax=p[0].x;

ymin=ymax=p[0].y;

}

else

cout<<"\n Enter valid no. of vertices.";

}

//FUNCTION FOR FINDING

void poly::calcs()

{ //MAX,MIN

for(int i=0;i<v;i++)

{

if(xmin>p[i].x)

xmin=p[i].x;

if(xmax<p[i].x)

xmax=p[i].x;

if(ymin>p[i].y)

ymin=p[i].y;

if(ymax<p[i].y)

ymax=p[i].y;

}

}

//DISPLAY FUNCTION

void poly::display()

{

int ch1;

char ch='y';

float s,s2;

do

{

cout<<"\n\nMENU:";

cout<<"\n\n\t1 . Scan line Fill ";

cout<<"\n\n\t2 . Exit ";

cout<<"\n\nEnter your choice:";

cin>>ch1;

switch(ch1)

{

case 1:

s=ymin+0.01;

delay(100);

cleardevice();

while(s<=ymax)

{

ints(s);

sort(s);

s++;

}

break;

case 2:

exit(0);

}

cout<<"Do you want to continue?: ";

cin>>ch;

}while(ch=='y' || ch=='Y');

}

void poly::ints(float z) //DEFINE FUNCTION INTS

{

int x1,x2,y1,y2,temp;

c=0;

for(int i=0;i<v;i++)

{

x1=p[i].x;

y1=p[i].y;

x2=p[i+1].x;

y2=p[i+1].y;

if(y2<y1)

{

temp=x1;

x1=x2;

x2=temp;

temp=y1;

y1=y2;

y2=temp;

}

if(z<=y2&&z>=y1)

{

if((y1-y2)==0)

x=x1;

else // used to make changes in x. so that we can fill our polygon after cerain distance

{

x=((x2-x1)\*(z-y1))/(y2-y1);

x=x+x1;

}

if(x<=xmax && x>=xmin)

inter[c++]=x;

}

}

}

void poly::sort(int z) //SORT FUNCTION

{

int temp,j,i;

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

{

line(p[i].x,p[i].y,p[i+1].x,p[i+1].y); // used to make hollow outlines of a polygon

}

delay(100);

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

{

delay(100);

line(inter[i],z,inter[i+1],z); // Used to fill the polygon ....

}

}

int main() //START OF MAIN

{

int cl;

initwindow(500,600);

cleardevice();

poly x;

x.read();

x.calcs();

cleardevice();

cout<<"\n\tEnter the colour u want:(0-15)->"; //Selecting colour

cin>>cl;

setcolor(cl);

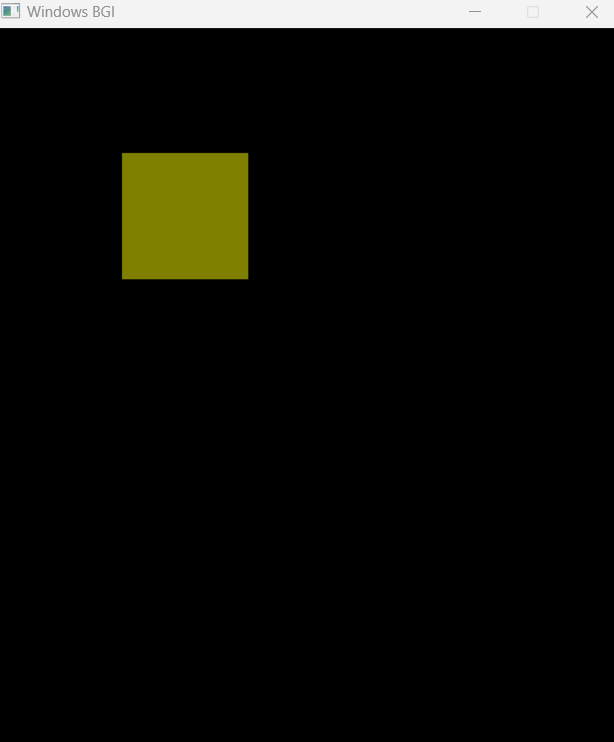
x.display();

closegraph(); //CLOSE OF GRAPH

getch();

return 0;

}



2 Dimensional Transformations

#include<iostream.h>

#include<stdio.h>

#include<conio.h>

#include<graphics.h>

#include<string.h>

#include<math.h>

int 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:

{

break;

}

default:

{

cout<<"Enter correct choice";

main();

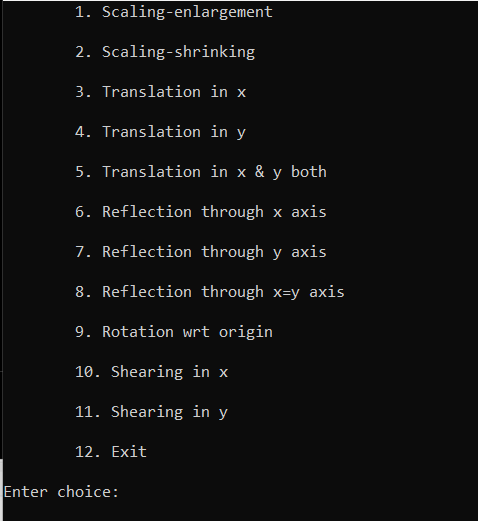
}

}

closegraph();

return 0;

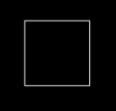
}



Before Scaling-



After Scaling-



3 Dimensional 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;

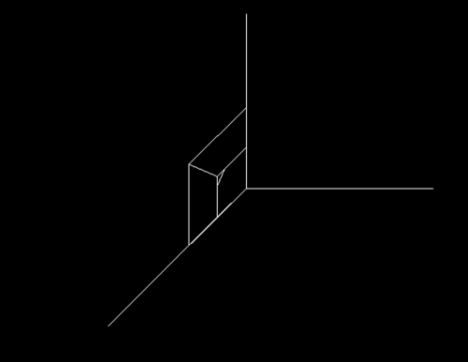
}

closegraph();

}

getch();

}



Hermite Curve

#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);

}

int 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;

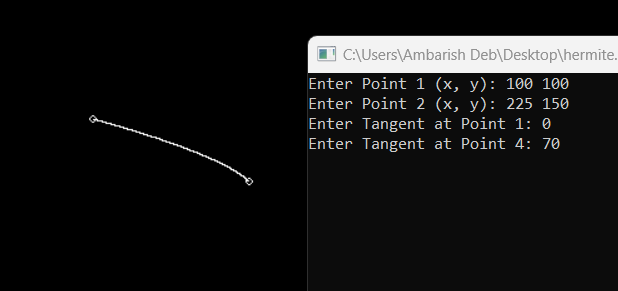
hermite(p1, p4, r1, r4);

getch();

closegraph();

return 0;

}



Bezier Curve

#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;

}

int 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 0;

}

