**ASSIGNMENT 1**

**Aim** - Write C++/JavaScript/Java program to draw line using DDA and Bresenham‘s algorithm.

**Objective** - To learn about DDA algorithm for making a line and its drawbacks like Aliasing effect and overcoming it by Bresenham’s algorithm and learning about its advantages and disadvantages over DDA.

**Theory** - In computer graphics, a **digital differential analyzer** (**DDA**) is hardware or software used for interpolation of variables over an interval between start and end point. DDAs are used for rasterization of lines, triangles and polygons.

**Bresenham's line algorithm** is a line drawing algorithm that determines the points of an *n*-dimensional raster that should be selected in order to form a close approximation to a straight line between two points.

Drawback of DDA algorithm overcome by Bresenham’s algorithm-

Bresenham’s algorithm is faster than DDA algorithm in line drawing because it performs only addition and subtraction in its calculation and uses only integer arithmetic, so it runs significantly faster. DDA algorithm is not as accurate and efficient as Bresenham algorithm.

**Algorithm-**

**DDA Algorithm**-

1. Read the line end points(x1, y1) and (x2, y2) such that they are not equal

[ if equal then plot that point and exit ].

2. ∆x = | x2-x1 | and ∆y = | y2- y1 |

3. if (∆ x >= y ) then //find the approx. length of line

length = ∆ x else

length = ∆y

4. ∆x =(x2-x1) / length //select the raster unit which may be

∆y =(y2-y1) / length //by a unit

5. x= x1+ 0.5 \* sign(∆x) //Round the values (Use Sign function

y= y1+ 0.5 \* sign(∆y) //to enable line drawing in any quad.)

6. i=1 //Plot the pixels

while (i<= length)

plot ( integer(x), integer(y))

x= x+ ∆x

y= y+ ∆y

i = i+1

7. Stop.

**Bresenham’s Algorithm-**

1. Read the line end point (x1,y1) and (x2,y2) such that they are not equal.

2. ∆x= |x2-x1|and ∆y= |y2-y1|

3. [Initialize starting point]

x=x1

y=y1

4. s1= Sign(x2 – x1)

s2= Sign(y2 – y1)

[Sign function returns -1, 0, 1 depending on whether its argument is <0, =0, >0 respectively]

5. If ∆y > ∆x then Exchange ∆x and ∆y

Ex\_change = 1 else Ex\_change = 0

endif

[Interchange ∆x and ∆y depending on the slope of the line and set Ex\_change Flag accordingly]

6. e=2\*∆y-∆x

[Initialize value of decision variable or error to compensate for nonzero intercept.]

7. i=1 [initialize counter]

8. plot (x,y)

9. while (e>=0)

{ if (Ex\_change = 1 ) then

x= x + s1

else y = y + s2

endif

e=e-2\*∆x

}

10. If Ex\_change = 1 then

y = y + s2

else x = x + s1

endif

e=e+2\*∆y

11. i=i+1

12. if (i<=∆x) then go to step 8

13. Stop

**Code-**

**DDA Line-**

#include<graphics.h>

#include<iostream>

using namespace std;

int sign(int n)

{

if(n>0)

{

return n;

}

else

{

return -n;

}

}

void dda(int x1,int y1,int x2,int y2)

{

int x,y,length;

int v1,v2,i;

v1=x2-x1;

v2=y2-y1;

if(v1>=v2)

{

length=v1;

}

else

{

length=v2;

}

v1=(x2-x1)/length;

v2=(y2-y1)/length;

x=x1+0.5\*sign(v1);

y=y1+0.5\*sign(v2);

i=1;

while(i<=length)

{

putpixel(x,y,RED);

x=x+v1;

y=y+v2;

i++;

}

}

int main()

{

int gd=DETECT,gm;

initgraph(&gd,&gm,NULL);

dda(150,50,250,150);

dda(150,50,50,150);

dda(150,50,400,50);

dda(250,150,500,150);

dda(50,150,50,350);

dda(250,150,250,350);

dda(400,50,500,150);

dda(500,150,500,350);

dda(50,350,500,350);

getch();

closegraph();

return 0;

}

**Bresenham’s Line-**

#include<iostream>

#include<graphics.h>

using namespace std;

void bresenham(int x1, int y1, int x2, int y2)

{ int slope\_error\_new;

int m\_new = 2 \* (y2 - y1);

if((x2-x1)<0)

slope\_error\_new = m\_new + (x2 - x1);

else

slope\_error\_new = m\_new - (x2 - x1);

for (int x = x1, y = y1; x <= x2; x++)

{

//cout << "(" << x << "," << y << ")\n";

putpixel(x,y,BLUE);delay(100);

// Add slope to increment angle formed

slope\_error\_new += m\_new;

if (slope\_error\_new >= 0)

{

y++;

slope\_error\_new -= 2 \* (x2 - x1);

}

}

}

int main()

{ int gd=DETECT,gm;

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

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

bresenham(100, 100, 250, 300);

bresenham(200,100, 150, 300);

//bresenham(300, 400, 500, 400);

//bresenham(300, 400, 500, 400);

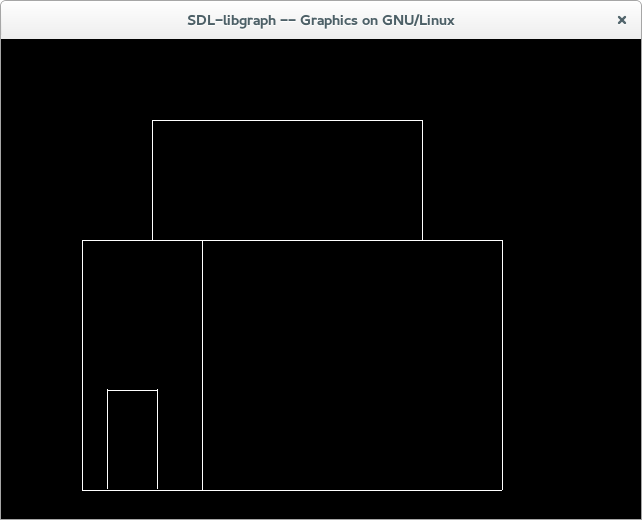
getch();

return 0;

}

**Output –**

**DDA Line –**

****

**Bresenham’s Line –**



**Conclusion** : figure using DDA and a line using Bresenham’s algorithm was made and the drawbacks of DDA were overcome by Bresenham’s algorithm.

**ASSIGNMENT 2**

**Aim** - Write C++/JavaScript/Java program to draw circle using Bresenham‘s algorithm.

**Objective**- To draw a circle using Bresenham’s algorithm and to understand the decision parameter, how the algorithm works and how is it different from the DDA circle algorithm.

**Theory-**

1. This algorithm considers the eight-way symmetry of the circle to generate it.
2. It plots 1/8 part of the circle, i.e. from 90 to 45 degrees.
3. As circle is drawn from 90 to 45 degrees, the x moves in positive direction and y moves in the negative direction.
4. To achieve best approximation to the true circle we have to select those pixels in the raster that fall the least distance from the true circle.
5. Each new point closest to the true circle can be found by applying either of two position:

-Increment in x direction by one unit or

-Increment in x direction and negative y direction both by one unit.

**Algorithm-**

1. Read the radius (r) of the circle

2. d =3 – 2r

[Initialize the decision variable]

3. x = 0, y = r

[Initialize starting point]

4. Do

{ Plot (x1+x, y1+y)

Plot (x1-x, y1+y)

Plot (x1+x, y1-y)

Plot (x1-x, y1-y)

Plot (x1-y, y1+x)

Plot (x1+y, y1-x)

Plot (x1-y, y1-x)

Plot (x1+y, y1+x)

If (d<0) then

{

d = d + 4x +6

}

Else

{d =d + 4 (x – y) + 10

y = y – 1

}

x = x + 1

}while (x < y);

1. Stop

**Code –**

#include <iostream>

#include <graphics.h>

using namespace std;

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

{

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

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

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

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

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

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

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

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

}

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, r2;

cout<<"Enter the center coordinates : ";

cin>>xc>>yc;

cout<<"Enter redius : ";

cin>>r2;

int gd = DETECT, gm;

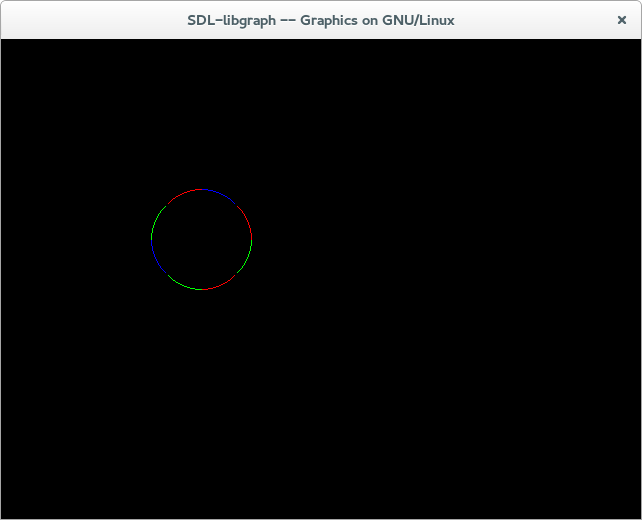
initgraph(&gd, &gm, NULL);

circleBres(xc, yc, r2);

return 0;

}

**Output –**

****

**Conclusion –** We achieved making a Bresenham; circle using the algorithm and also learnt about why and how it is faster from the DDA circle drawing algorithm.

**ASSIGNMENT 3**

**Aim -**Write C++/JavaScript/Java program to draw 2-D object and perform following basic transformations,

a) Scaling

b) Translation

c) Rotation

**Objective –** To learn how to perform basic transformations on a 2D object and learning about the 2D matrices of all the three transformations i.e. scaling, rotation and translation and using it to code for the transformation.

**Theory –**

**2D Transformation**

It is a process of changing the position of the object, or changing the

size of the object or changing the orientation of the object or may be

any combination of these.

**Translation -** It is a process of changing the position of an object.

In translation the object’s position is moved by a certain distance and the translation distance pair is called the shift vector ot the translation vector denoted by (tx,ty).It’s matrix is as represented below-

[x’ y’ 1] = [x y 1] \* 1 0 0

0 1 0

tx ty 1

**Scaling –** Scaling is the process to expand or contract a particular object. If the scaling factor is less than 0 then the object contracts and if it is greater than 0 then the object is magnified. The amount by which the object is scaled is represented by the pair (sx,sy) and its 2D matrix is as represented below-

[x’ y’] = [x y] \* Sx 0

0 Sy

**Rotation -** Movement of a point about origin in clockwise or anticlockwise direction is known as rotation.

Object is positioned along a circular path in the xy plane, & a rotation angle θ is used to position the rotation point about which the object is rotated.

x = rcosθ

y= rsinθ

Its 2D matrix is as represented below-

[x’ y’] = [x y] \* cosθ sinθ

-sinθ cosθ

**Algorithm –**

**Translation –**

1. Read the number of sides of polygon

2. Enter the co-ordinates of polygon

3. Enter the translation on X & Y direction

4. Move the line from initial position ax[0] , ay[0]

5. for I=0 to n ;

lineto(ax[I],ay[I]);

6. Draw line from ax[0],ay[0]

7. for I=0 to n

cx[I]=ax[I]+tx

cy[I]=ay[I]+ty

8. Now move the lines to initial position back for close polygon

ax[0],ay[0]

9. for I=0 to n

Draw line to (cx[I],cy[I])

Draw line to (cx[0],cy[0])

10.Stop.

**Scaling –**

1. Read the number of sides of polygon

2. Enter the co-ordinates of polygon

3. Enter the scaling sx & sy in x & y direction respectively

4. Move current position to ax[0] and ay[0]

5. Draw all sides of polygon

6. Multiply all co-ordinates by scaling factor sx & sy

For I=0 to n

cx[I] = ax[I] \* sx

cy[I] = ax[I] \* sy

7. Move current position to cx[0],cy[0]

8. Draw all sides of polygon i.e. scaled polygon

9. Stop

**Rotation –**

1. Start

2. Define values for x, y co-ordinates

3. Draw the original 2D object (line)

4. For rotation of object get the angle of rotation

5. Convert the angle in radian value

6. Calculate new vertices (displacement of object) by using 2

dimensional transformation matrix for the rotation .

7. Draw both the objects ( original line and rotated line)

8. Stop.

**Code –**

**Translation –**

#include<iostream>

#include<graphics.h>

using namespace std;

int sign (int x)

{

if(x>0)

{

return 1;

}

else if(x=0)

{

return 0;

}

else

{

return -1;

}

}

int bresenham(int x1,int y1,int x2,int y2)

{

int e,x,y,s1,s2,ex\_change,dx,dy,temp;

dx=abs(x2-x1);

dy=abs(y2-y1);

x=x1,y=y1;

s1=sign(dx);

s2=sign(dy);

if(dy>dx)

{

temp=dy;

dy=dx;

dx=temp;

ex\_change=1;

}

else

{

ex\_change=0;

}

e=2\*dy-dx;

int i=1;

while(i<=dx)

{

putpixel(x,y,10);

while(e>=0)

{

if(ex\_change==1)

x=x+s1;

else

y=y+s2;

e=e-2\*dx;

}

if(ex\_change==1)

y=y+s2;

else

x=x+s1;

e=e+2\*dy;

i++;

}

}

int main()

{

int gm,gd=DETECT;

initgraph(&gm,&gd,NULL);

bresenham(20,30,100,100);

getch();

closegraph();

int tx=1,ty=1;

cout<<"\n enter the value of translation factor in x and y direction respectively : ";

cin>>tx>>ty;

int original\_coordinates[2][3]={20,30,1,100,100,1};

int translation\_matrix[3][3]={1,0,0,0,1,0,tx,ty,1};

int new\_coordinates[2][3]={0};

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

{

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

{

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

{

new\_coordinates[i][j]+=original\_coordinates[i][k]\*translation\_matrix[k][j];

}

}

}

initgraph(&gm,&gd,NULL);

bresenham(new\_coordinates[0][0],new\_coordinates[0][1],new\_coordinates[1][0],new\_coordinates[1][1]);

getch();

closegraph();

return 0;

}

**Scaling –**

#include<stdio.h>

#include<graphics.h>

#include<math.h>

#include<iostream>

using namespace std;

//function for triangle scaling

void scaling(int x1,int y1,int x2,int y2,int x3,int y3)

{

int sx,sy,xn1,yn1,xn2,xn3,yn3,yn2;

int gd = DETECT, gm;

cout<<"ehter the scalling vector\n";

cin>>sx>>sy;

//scale the co-ordinates of triangle vertices by multiplying the scaling vector

xn1=x1\*sx;

yn1=y1\*sy;

xn2=x2\*sx;

yn2=y2\*sy;

xn3=x3\*sx;

yn3=y3\*sy;

//initializing the graph

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

//print the intitial triangle

line(x1,y1,x2,y2);

line(x1,y1,x3,y3);

line(x2,y2,x3,y3);

delay(5000);

//print the scaled triangle

line(xn1,yn1,xn2,yn2);

line(xn1,yn1,xn3,yn3);

line(xn2,yn2,xn3,yn3);

delay(5000);

cleardevice();

}

int main()

{

int ch,x1,y1,x2,y2,x3,y3;

cout<<"Enter the vertex coordinates of the triangle\n";

cin>>x1>>y1>>x2>>y2>>x3>>y3;

scaling(x1,y1,x2,y2,x3,y3);

getch();

closegraph();

return 0;

}

**Rotation –**

#include<iostream>

#include<graphics.h>

#include<cmath>

using namespace std;

int sign (int x)

{

if(x>0)

{

return 1;

}

else if(x=0)

{

return 0;

}

else

{

return -1;

}

}

int bresenham(float x1,float y1,float x2,float y2)

{

int e,x,y,s1,s2,ex\_change,dx,dy,temp;

dx=abs(x2-x1);

dy=abs(y2-y1);

x=x1,y=y1;

s1=sign(dx);

s2=sign(dy);

if(dy>dx)

{

temp=dy;

dy=dx;

dx=temp;

ex\_change=1;

}

else

{

ex\_change=0;

}

e=2\*dy-dx;

int i=1;

while(i<=dx)

{

putpixel(x,y,10);

while(e>=0)

{

if(ex\_change==1)

x=x+s1;

else

y=y+s2;

e=e-2\*dx;

}

if(ex\_change==1)

y=y+s2;

else

x=x+s1;

e=e+2\*dy;

i++;

}

}

int main()

{

int gm,gd=DETECT;

initgraph(&gm,&gd,NULL);

bresenham(20,30,100,100);

getch();

closegraph();

float angle=1;

cout<<"\n enter the value of angle of rotation : ";

cin>>angle;

float radian\_angle=(angle\*M\_PI)/180;

float acos=1,asin=1;

acos=cos(radian\_angle);

asin=sin(radian\_angle);

float original\_coordinates[2][3]={20,30,1,100,100,1};

float rotation\_matrix[3][3]={acos,-asin,0,asin,acos,0,0,0,1};

float new\_coordinates[2][3]={0};

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

{

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

{

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

{

new\_coordinates[i][j]+=original\_coordinates[i][k]\*rotation\_matrix[k][j];

}

}

}

initgraph(&gm,&gd,NULL);

bresenham(20,30,100,100);

bresenham(new\_coordinates[0][0],new\_coordinates[0][1],new\_coordinates[1][0],new\_coordinates[1][1]);

getch();

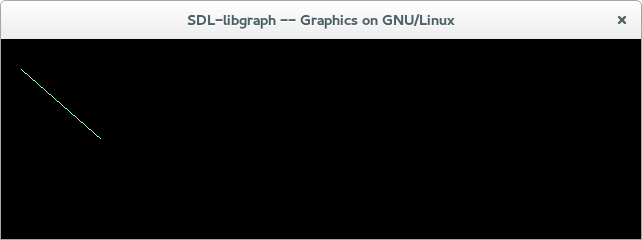
closegraph();

return 0;

}

**Output –**

**Translation –**



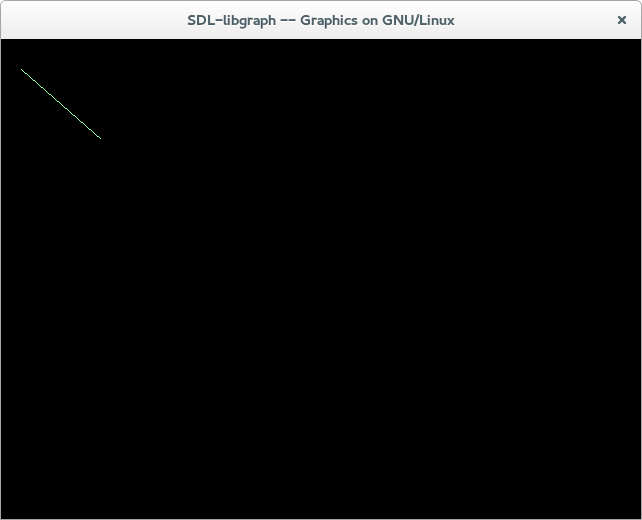


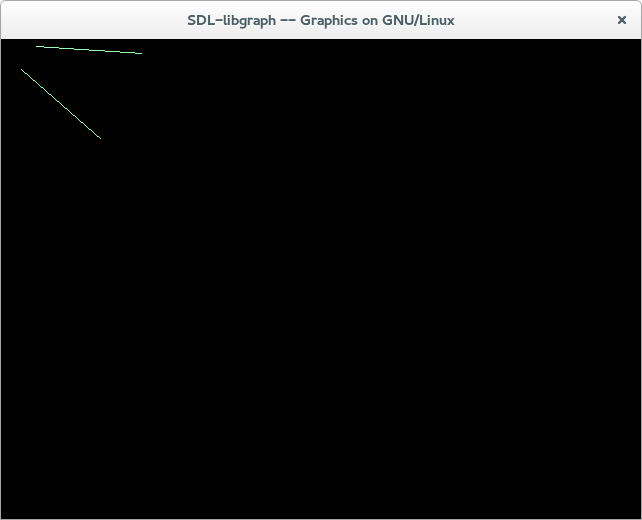
**Scaling** –





**Rotation –**





**Conclusion –** Welearnt about the 2D transformation and how an object can be translated, scaled or rotated.

**ASSIGNMENT 4**

**Aim -** Write C++/JavaScript/Java program to fill polygon using scan line algorithm.

**Objective –** To learn about the scan line algorithm and how it works by finding the intersection points and colouring the object by running a scan line from left to right and top to bottom.

**Theory -** Scanline filling is basically filling up of polygons using horizontal lines or scanlines. The purpose of the Scanline Polygon Filling algorithm is to fill (colour) the interior pixels of a polygon given only the vertices of the figure. This algorithm works by intersecting scanline with polygon edges and fills the polygon between pairs of intersections.

Components of scanline polygon fill –

1. **Edge Table:**It consists of several edge lists -> holds all the edges that compose the figure. When creating edges, the vertices of the edge need to be ordered from left to right and thee edges are maintained in increasing yMin order. Filling is complete once all the edges are removed from the Edge Table.
2. **Active List:** IT maintains the current edges being used to fill in the polygon. Edges are pushed into the AL from the Edge Table when an edge’s yMin is equal to the current scan line being processed.  
   The Active List will be re-sorted after every pass.

**Algorithm –**

1. Read n, the number of vertices of polygon.

2. Read x and y coordinates of all vertices in array x[n] & y[n].

3. Find ymin & ymax

4. Store the initial x value (x1) y values y1 & y2 for two endpoints and x increment dx from scan line to scan line for each edge in the array edges [n] [4]. While doing this check that y1 > y2, if not interchange y1 & y2 & corresponding x1 & x2 so that for each edge, y1 represents its maximum coordinate and y2 represents its minimum y coordinate.

5. Sort the rows of array, edges [n] [4] in descending order of y1, descending order of y2 & ascending order of x2.

6. Set y = ymax

7. Find the active edges & update active edge list:

if ( y > y2 and y <= y1) { edge is active} else {edge is not active}

8. Compute the x intersection for all active edges for current y value [initially x-intersect is x1 & x intersects for successive y values can be given as xi+1 xi + dx where dx = - 1/m and m=y2-y1/x2-x1 i.e. slope of a line segment.

9. If x intersect is vertex i.e. x - intersect = x1 and y=y1 then apply vertex test to check whether to consider one intersect or two intersects. Store all x intersects in the x- intersect [ ] array.

10. Sort x - intersect [ ] array in the ascending order,

11. Extract pairs of intersects from the sorted x-intersect [ ] array.

12. Pass pairs of x values to line drawing routine to draw corresponding line segments.

13. Set y = y-1

14. Repeat steps 7 through 13 until y >= ymin

15. Stop.

**Code** –

#include <iostream>

#include <graphics.h>

#include <stdlib.h>

using namespace std;

int ch1;

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++) {

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.";

}

void poly::calcs()

{

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;

}

}

void poly::display()

{

char ch;

ch1 = 1;

float s,s2;

do

{

switch(ch1)

{

case 1:

s=ymin+0.01;

delay(50);

while(s<=ymax)

{

ints(s);

sort(s);

s++;

}

cleardevice();

break;

case 2:

exit(0);

}

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

cin>>ch;

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

}

void poly::ints(float z)

{

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 {

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

x=x+x1;

}

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

inter[c++]=x;

}

}

}

void poly::sort(int z)

{

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);

}

delay(100);

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

{

delay(100);

line(inter[i],z,inter[i+1],z);

}

}

int main()

{

int cl;

poly x;

x.read();

x.calcs();

//cleardevice();

cout<<"\n\nMENU:";

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

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

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

cin>>ch1;

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

cin>>cl;

int gd=DETECT,gm;

initgraph(&gd,&gm,NULL);

setcolor(cl);

x.display();

closegraph();

getch();

return 0;

}

**Output –**

**Conclusion –** We learnt how to apply scan line algorithm for polygon filling and learnt how it was better than the seed fill algorithms.

**ASSIGNMENT 5**

**Aim-** Write C++/JavaScript/Java program to draw a polygon and fill it with desired colour using Seed fill algorithm.

**Objective –** To learn about the seed fill algorithms and comparing them with other polygon filling algorithms and knowing the disadvantages and advantages of the same.

**Theory –**

**Seed fill algorithm –**

It assumes that at least one pixel is interior to the polygon. It tries to find all other pixels interior to the polygon & subsequently colour them. One way to fill a polygon is to start from a given “seed”, point known to be inside the polygon and highlight outward from this point. It is of two types Boundary fill and Flood fill.

**Boundary fill algorithm –**

1. In this algorithm adjacent pixels are checked for boundary colour. It starts with the given seed. If pixel colour is not equal to boundary colour, the pixel is coloured.
2. In this way we make use 4 or 8 connected points i.e. symmetric property. But eight connected approach is more accurate.
3. The approach where neighbouring pixels are highlighted until we encounter the boundary pixels.

**Flood fill algorithm –**

This approach is called flood fill because colour flows from the seed pixel until reaching the polygon boundary, like water flooding on the surface of the container.

Starting with some seed point, inside the poly6gon, and examine the

neighbouring pixels to check whether the boundary pixel is reached

by checking the background colour.

If boundary pixels are not reached, pixels are highlighted and the

process is continued until boundary pixels are reached.

**Algorithm –**

**Boundary fill algorithm -**

1. Start

2. int current x, y, boundary colour

3. find current colour, current colour=getcolour(x,y)

4. if(current colour != fill colour)&&(current colour!=boundary

colour)

5. then plot (x, y, fillcolour)

6. Boundary fill (x+1, y, fill colour, boundary colour)

7. Boundary fill (x-1, y, fill colour, boundary colour)

8. Boundary fill (x, y+1, fill colour, boundary colour)

9. Boundary fill (x, y-1, fill colour, boundary colour)

10. End

**Flood fill algorithm –**

Flood\_Fill (x, y, f\_color, b\_color)

{

If (getpixel(x,y) = b\_color)

{

putpixel(x, y, f\_color)

flood\_fill(x+1,y,f\_color,b\_color)

flood \_fill(x,y+1,f\_color,b\_color)

flood \_fill(x-1,y,f\_color,b\_color)

flood \_fill(x,y-1,f\_color,b\_color)

}

}

**Code –**

**Boundary fill –**

#include<iostream>

#include<graphics.h>

using namespace std;

void boundaryfill(int x, int y, int f\_color, int b\_color)

{

if(getpixel(x,y)!=b\_color && getpixel(x,y)!=f\_color)

{

putpixel(x,y,f\_color);

boundaryfill(x+1,y,f\_color,b\_color);

boundaryfill(x,y+1,f\_color,b\_color);

boundaryfill(x-1,y,f\_color,b\_color);

boundaryfill(x,y-1,f\_color,b\_color);

}

}

int main()

{

int x,y,r;

x=100,y=200,r=50;

int gd=DETECT,gm;

initgraph(&gd,&gm,NULL);

circle(x,y,r);

boundaryfill(x,y,6,15);

getch();

closegraph();

return 0;

}

**Flood fill –**

#include<iostream>

#include<graphics.h>

using namespace std;

void floodFill(int x,int y,int oldcolor,int newcolor)

{

if(getpixel(x,y) == oldcolor)

{

putpixel(x,y,newcolor);

floodFill(x+1,y,oldcolor,newcolor);

floodFill(x,y+1,oldcolor,newcolor);

floodFill(x-1,y,oldcolor,newcolor);

floodFill(x,y-1,oldcolor,newcolor);

}

}

//getpixel(x,y) gives the color of specified pixel

int main()

{

int gm,gd=DETECT,radius;

int x,y;

cout<<"Enter x and y positions for circle\n";

cin>>x>>y;

cout<<"Enter radius of circle\n";

cin>>radius;

initgraph(&gd,&gm,NULL);

circle(x,y,radius);

floodFill(x,y,0,15);

delay(10000);

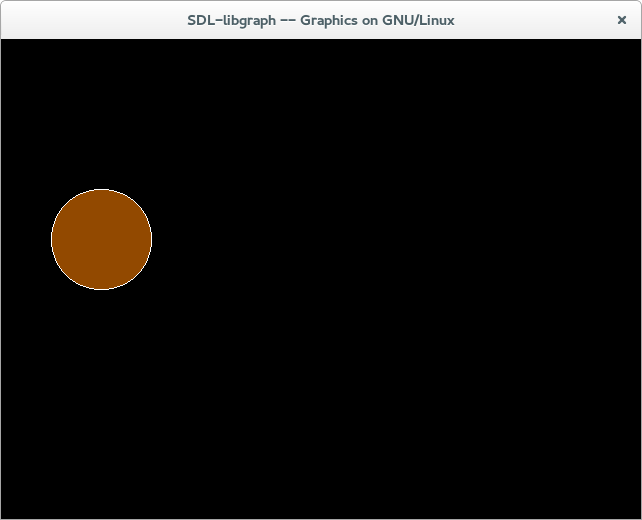
closegraph();

return 0;

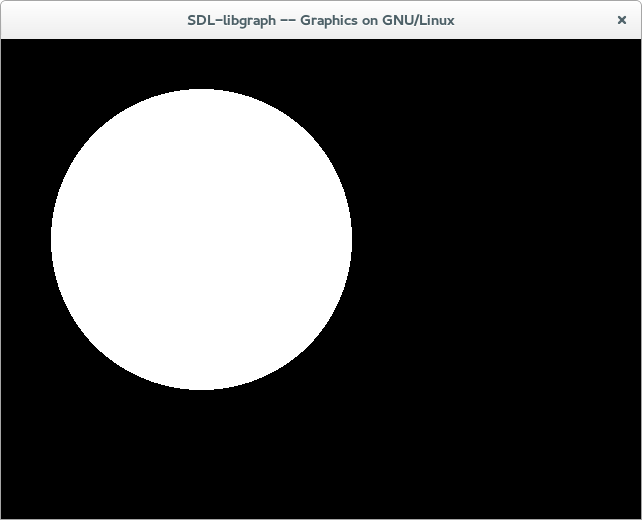
}

**Output –**

**Boundary fill –**

****

**Flood fill –**

****

**Conclusion –** Both the seed fill algorithms were applied and were compared in various terms like which one was faster and better and the polygons were filled.

**ASSIGNMENT 6**

**Aim -** Write C++/JavaScript/Java program to generate Hilbert curve using concept of fractals.

**Objective** –To write a program for Hilbert curve.

**Theory –**

Hilbert functions can help build indexes for spatial databases; when searching for a

record in close geographic location they can help determine the priority for exploring. Outside

of database work, they are sometimes used in image process. When converting an image to

gray scale through the dithering of black and white pixels, carry-over values from thresholding

can be carried forward and, by the use of a Hilbert Curve, the excess can be distributed in a

pattern less obvious to the eye.

**Algorithm –**

void hilbert(int dir, int rot, int order)

{

if (order == 0) return;

dir = dir + rot;

hilbert(dir, -rot, order - 1);

step(dir);

dir = dir - rot;

hilbert(dir, rot, order - 1);

step(dir);

hilbert(dir, rot, order - 1);

dir = dir - rot;

step(dir);

hilbert(dir, -rot, order - 1);

}

**Code –**

#include <iostream>

#include <stdlib.h>

#include <graphics.h>

#include <math.h>

using namespace std;

void move(int j,int h,int &x,int &y)

{

if(j==1)

y-=h;

else if(j==2)

x+=h;

else if(j==3)

y+=h;

else if(j==4)

x-=h;

lineto(x,y);

}

void hilbert(int r,int d,int l,int u,int i,int h,int &x,int &y)

{

if(i>0)

{

i--;

hilbert(d,r,u,l,i,h,x,y);

move(r,h,x,y);

hilbert(r,d,l,u,i,h,x,y);

move(d,h,x,y);

hilbert(r,d,l,u,i,h,x,y);

move(l,h,x,y);

hilbert(u,l,d,r,i,h,x,y);

}

}

int main()

{

int n,x1,y1;

int x0=50,y0=150,x,y,h=10,r=2,d=3,l=4,u=1;

cout<<"\nGive the value of n: ";

cin>>n;

x=x0;y=y0;

int gm,gd=DETECT;

initgraph(&gd,&gm,NULL);

moveto(x,y);

hilbert(r,d,l,u,n,h,x,y);

delay(10000);

closegraph();

return 0;

}

**Output –**

**Conclusion –**We have generated a Hilbert curve using concept of fractrals using C++.

**ASSIGNMNENT 7**

**Aim –** Write C++/JavaScript/Java program to generate Bouncing ball animation using Direct3D / Maya / Blender.

**Objective –**

**Theory –**

**Code –**

**Output –**

**Conclusion –**

**ASSIGNMENT 8,9**

**Aim –** Write C++/JavaScript/Java program to simulate any one of the scene- (Mini Project)

1. Airplane Landing
2. ii. Vehicle locomotion

Write C++/JavaScript/Java program to simulate any one of the scene- (Mini Project)

1. Bird Flying ii.
2. Any Game with locomotion

**Objective –**

**Theory –**

**Code –**

**Output –**

**Conclusion –**