## **ANNA UNIVERSITY**

# **MADRAS INSTITUE OF TECHNOLOGY**

## **CHROMPET, CHENNAI – 600 044.**

**BONAFIDE CERTIFICATE**

*Name :*  Shreayaas S Iyer

*Subject :* Graphics and Multimedia Lab

*Department :* Information Technology

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 2 | 0 | 1 | 8 | 5 | 0 | 6 | 1 | 1 | 5 |

REGISTER NO.

***Certified that the bonafide record of practical work done by***

***Mr./~~Miss~~* Shreayaas S Iyer**

***in the* Graphics and Multimedia *Laboratory subject code* IT 7711**

***during the Period* September *2021 –*  *2021***

**Date : COURSE-IN-CHARGE**

***Submitted for Practical Examination held on***

***Examiners***

**1.**

**2.**

Index

**S. No**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Remark**  **Page**  **Mark**  **Name of the experiment**  **Date** | | | | | |
| **1a** |  | **Study of 2D graphical primitives** |  |  |  |
| **1b** |  | **Study of in-built functions in 2D primitives** |  |  |  |
| **1c** |  | **Basic Graphics Programs** |  |  |  |
| **2** |  | **Digital Differential Algorithm** |  |  |  |
| **3** |  | **Bresenham Line Drawing Algorithm** |  |  |  |
| **4** |  | **Midpoint Circle Algorithm** |  |  |  |
| **5** |  | **Midpoint Ellipse Algorithm** |  |  |  |
| **6** |  | **Window to viewport Transformation** |  |  |  |
| **7** |  | **Liang Barsky Clipping Algorithm** |  |  |  |
| **8** |  | **Cohen Sutherland Clipping Algorithm** |  |  |  |
| **9** |  | **2D Transformations on an object** |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

# Experiment 1a) Study of Basic 2D graphical primitives

**Date :**

**Aim**: To study about various 2D- graphical primitives involved in Computer graphics.

**DEFINITION –**

Graphical primitive: In graphics, primitives are basic elements, such as lines, curves, and polygons, which you can combine to create more complex graphical images. In programming, primitives are the basic operations supported by the programming language.

In reality, Computer graphics refer different things in different contexts:

– Pictures, scenes that are generated by a computer.

– Tools used to make such pictures, software and hardware, input/output devices.

– The whole field of study that involves these tools and the pictures they produce.

Types of 2D graphics

1. Raster - Raster graphics are the most common and are used for digital photos, Web graphics, icons, and other types of images. They are composed of a simple grid of pixels, which can each be a different color.

2. Vector - made up of paths, which may be lines, shapes, letters, or other scalable objects. They are often used for creating logos, signs, and other types of drawings. Unlike raster graphics, vector graphics can be scaled to a larger size without losing quality.

Primitives :

A basic non-divisible graphical element for input or output within a computer-graphics system.

Input primitive

Typical input primitives include polyline, poly marker, and fill area.

Clipping of an output primitive cannot be guaranteed to produce another output primitive. Typical input primitives are locator, choice, and valuator. Input primitives often have a style of echoing associated with them.

Output primitives

Output primitives are the geometric structures that has attributes such as straight line segments (pixel array) and patterns i.e.; polygon colour areas, used to describe the shapes and colours of the objects. Points and straight line segments are the simplest geometric components of pictures. Additional output primitive includes: circles and other conic sections, quadric surfaces, spline curves and surfaces, polygon colour areas and character strings. This includes picture generation algorithm by examining device-level algorithms for displaying two-dimensional output primitives, with emphasis on scan-conversion methods for raster graphics system.

Different 2D graphical primitives

Point -

Point plotting is accomplished by converting a single coordinate position furnished by an application program into appropriate operations for the output device. With a CRT monitor, for example, the electron beam is turned on to illuminate the screen phosphor at the selected location.

Line -

Line drawing is accomplished by calculating intermediate positions along the line path between two specified end points positions. An output device is then directed to fill in these positions between the end points.

Rectangular Shapes -

The Rectangle2D, RoundRectangle2D, Arc2D, and Ellipse2D primitives are all derived from the RectangularShape class. This class defines methods for Shape objects that can be described by a rectangular bounding box. The geometry of a Rectangular Shaped object can be extrapolated from a rectangle that completely encloses the outline of the Shape.

Areas -

With the Area class, you can perform boolean operations, such as union, intersection, and subtraction, on any two Shape objects. This technique, often referred to as *constructive area geometry*, enables you to quickly create complex Shape objects without having to describe each line segment or curve.

Line Drawing Algorithms

• Digital Differential Analyzer (DDA) Algorithm

• Bresenham’s Line Algorithm

• Parallel Line Algorithm

**Result:**

Basic 2D primitives in graphics have been successfully studied.

# Experiment 1b) Study of in-built functions in 2D graphical primitives

**Date :**

**Aim**: To study about various 2D- graphical primitives inbuilt-functions involved in Computer graphics.

**Functions with Syntax:**

❖ arc - it used to draw an arc with center (x, y) and stangle specifies starting angle, endangle specifies the end angle and last parameter specifies the radius of the arc. arc function can also be used to draw a circle but for that starting angle and end angle should be 0 and 360 respectively.

Declaration: void arc(int x, int y, int stangle, int endangle, int radius);

❖ bar - Bar function is used to draw a 2-dimensional, rectangular filled in bar . Coordinates of left top and right bottom corner are required to draw the bar. Left specifies the X-coordinate of top left corner, top specifies the Y-coordinate of top left corner, right specifies the X-coordinate of right bottom corner, bottom specifies the Y-coordinate of right bottom corner.

Declaration: void bar(int left, int top, int right, int bottom);

❖ bar3d - bar3d function is used to draw a 2-dimensional, rectangular filled in bar . Coordinates of left top and right bottom corner of bar are required to draw the bar. left specifies the X-coordinate of top left corner, top specifies the Y-coordinate of top left corner, right specifies the X-coordinate of right bottom corner, bottom specifies the Y-coordinate of right bottom corner, depth specifies the depth of bar in pixels, topflag determines whether a 3 dimensional top is put on the bar or not ( if it's non-zero then it's put otherwise not ).

Declaration: void bar3d(int left, int top, int right, int bottom, int depth, int topflag);

❖closegraph - closegraph function closes the graphics mode, deallocates all memory allocated by graphics system and restores the screen to the mode it was in before you called initgraph.

Declaration: void closegraph();

❖ drawpoly - Drawpoly function is used to draw polygons i.e. triangle, rectangle, pentagon, hexagon etc.

Declaration: void drawpoly( int num, int \*polypoints );

❖ Ellipse - Ellipse is used to draw an ellipse (x,y) are coordinates of center of the ellipse, stangle is the starting angle, end angle is the ending angle, and fifth and sixth parameters specifies the X and Y radius of the ellipse.

Declarations: void ellipse(int x, int y, int stangle, int endangle, int xradius, int yradius);

❖ fillellipse - x and y are coordinates of center of the ellipse, xradius and yradius are x and y radius of ellipse respectively.

Declaration: void fillellipse(int x, int y, int xradius, int yradius);

❖ fillpoly - Fillpoly function draws and fills a polygon. It require same arguments as drawpoly.

Declaration: void drawpoly( int num, int \*polypoints );

❖ floodfill - floodfill function is used to fill an enclosed area.

Declaration: void floodfill(int x, int y, int border);

❖ getcolor - getcolor function returns the current drawing color.

Declaration: int getcolor();

❖ getdrivername - Function getdrivername returns a pointer to the current graphics driver.

Declaration: char \*name = getdrivername();

❖ getimage - getimage function saves a bit image of specified region into memory, region can be any rectangle.

Declaration: void getimage(int left, int top, int right, int bottom, void \*bitmap);

❖ getmaxx - getmaxx function returns the maximum X coordinate for current graphics mode and driver.

Declaration: int getmaxx();

❖ getmaxy - getmaxy function returns the maximum Y coordinate for current graphics mode and driver.

Declaration: int getmaxy();

❖ getpixel - getpixel function returns color of pixel present at location(x, y).

Declaration: int getpixel(int x, int y);

❖ setbkcolor - setbkcolor function changes current background color

Declaration: void setbkcolor(int color);

❖ setcolor – setcolor function is used to change the current drawing color.

Declaration: void setcolor(int color);

**Result:** Various graphics functions has been studied and noted.

# Experiment 1c) Basic Graphic Programs

**Date :**

**Aim**: To implement basic graphic programs using 2D primitives in C

**1) Printing basic 2D shapes**

**Code :**

#include<graphics.h>

#include<conio.h>

main(){

int gd=DETECT,gm;

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

setbkcolor(BLACK);

line(50,40,190,40);

rectangle(70,115,215,165);

arc(120,200,180,0,30);

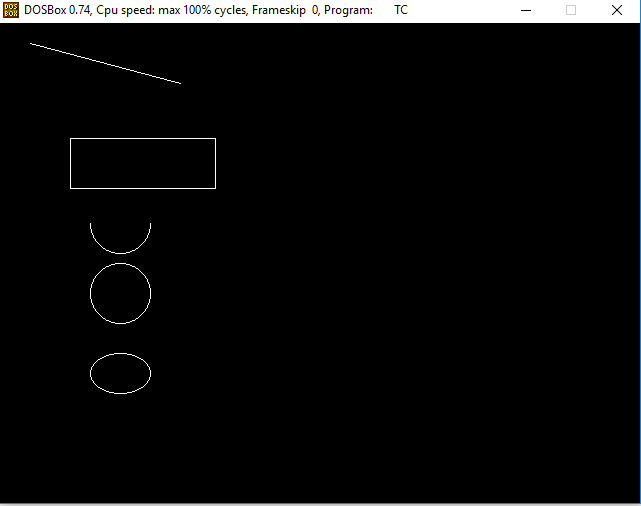
circle(120,270,30);

ellipse(120,350,0,360,30,20);

getch();

}

**Output :**



**2) Printing circular pattern**

**Code :**

#include<graphics.h>

#include<conio.h>

main()

{

int gd = DETECT, gm, x, y, color, angle = 0;

struct arccoordstype a, b;

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

delay(2000);

while(angle<=360)

{

setcolor(BLACK);

arc(getmaxx()/2,getmaxy()/2,angle,angle+2,100);

setcolor(BLUE);

getarccoords(&a);

circle(a.xstart,a.ystart,25);

setcolor(BLACK);

arc(getmaxx()/2,getmaxy()/2,angle,angle+2,150);

getarccoords(&a);

setcolor(YELLOW);

circle(a.xstart,a.ystart,25);

angle = angle+5;

delay(50);

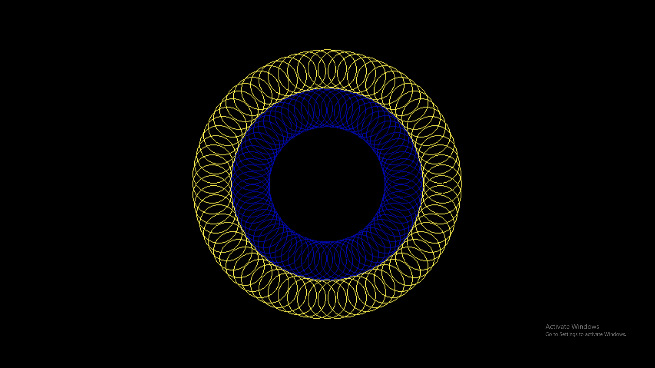
}

getch();

closegraph();

}

**Output :**



**Result –** Basic Graphic programs are implemented in C.

# Experiment 2) Digital Differential Algorithm

**Date :**

**Aim**: To implement DD algorithm in C to draw a line between two endpoints

**Algorithm :**

* Get two endpoint pixel positions as Input.
* Horizontal and vertical differences between the endpoint positions are assigned to parameters dx and dy. (Calculate dx=xb-xa and dy=yb-ya).
* The difference with the greater magnitude determines the value of parameter steps.
* Starting with pixel position (xa, ya), determine the offset needed at each step to generate the next pixel position along the line path.
* Loop the following process for a number of times. Use a unit of increment or decrement in the x and y direction b. if xa is less than xb the values of increment in the x and y directions are 1 and m c. if xa is greater than xb then the decrements -1 and – m are used.

**Code :**

#include<graphics.h>

#include<conio.h>

#include<stdio.h>

main() {

int gd = DETECT ,gm, i;

float x, y,dx,dy,steps;

int x0, x1, y0, y1;

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

setbkcolor(WHITE);

x0 = 100 , y0 = 200, x1 = 500, y1 = 300;

dx = (float)(x1 - x0);

dy = (float)(y1 - y0);

if(dx>=dy){

steps = dx; }

else{

steps = dy; }

dx = dx/steps;

dy = dy/steps;

x = x0; y = y0; i = 1;

while(i<= steps){

putpixel(x, y, BLUE);

x += dx; y += dy;

i=i+1;

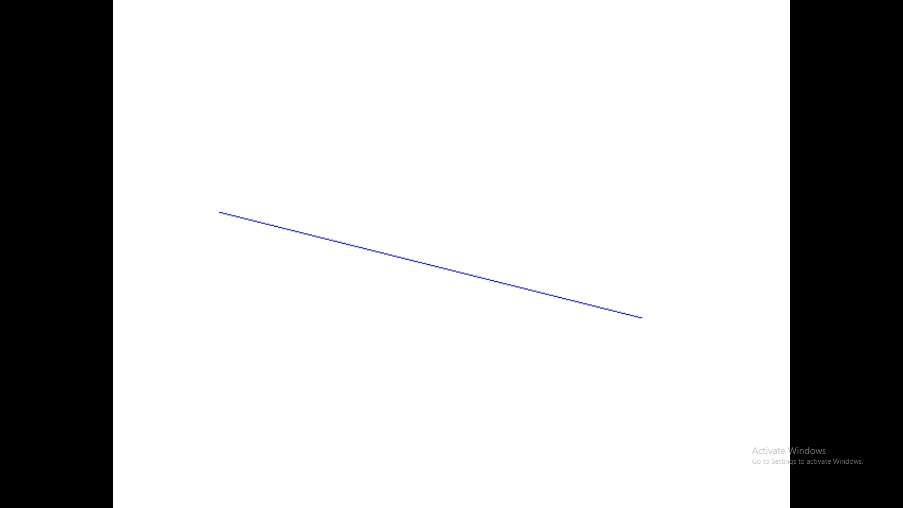
}

getch();

closegraph();

}

**Output :**



**Result :** Digital Differential Algorithm is implemented in C.

# Experiment 3) Bresenham Algorithm

**Date :**

**Aim**: To implement Bresenham’s algorithm in C to draw a line between two endpoints

**Algorithm :**

* Input the two line endpoints and store the left end point in (x0, y0).
* Load (x0, y0) into frame buffer, ie. Plot the first point.
* Calculate the constants Δx, Δy, 2Δy and obtain the starting value for the decision parameter as
* P0 = 2Δy-Δx
* At each xk along the line, starting at k=0 perform the following test If Pk < 0, the next point to plot is (xk+1, yk) and Pk+1 = Pk + 2Δy otherwise, the next point to plot is (xk+1, yk+1) and Pk+1 = Pk + 2Δy - 2Δx
* Perform step4 Δx times.

**Code :**

#include<iostream.h>

#include<graphics.h>

void drawline(int x0, int y0, int x1, int y1{

int dx, dy, p, x, y;

dx =x1-x0;

dy =y1-y0;

x=x0;

y=y0;

p=2\*dy-dx

while(x<x1){

if(p>=0){

putpixel(x,y,7);

y=y+1;

p=p+2\*dy-2\*dx;

}

else{

putpixel(x,y,7);

p=p+2\*dy;

}

x=x+1;

}

}

int main(){

int gdriver=DETECT, gmode, error, x0=100, y0=100, x1=200, y1=200;

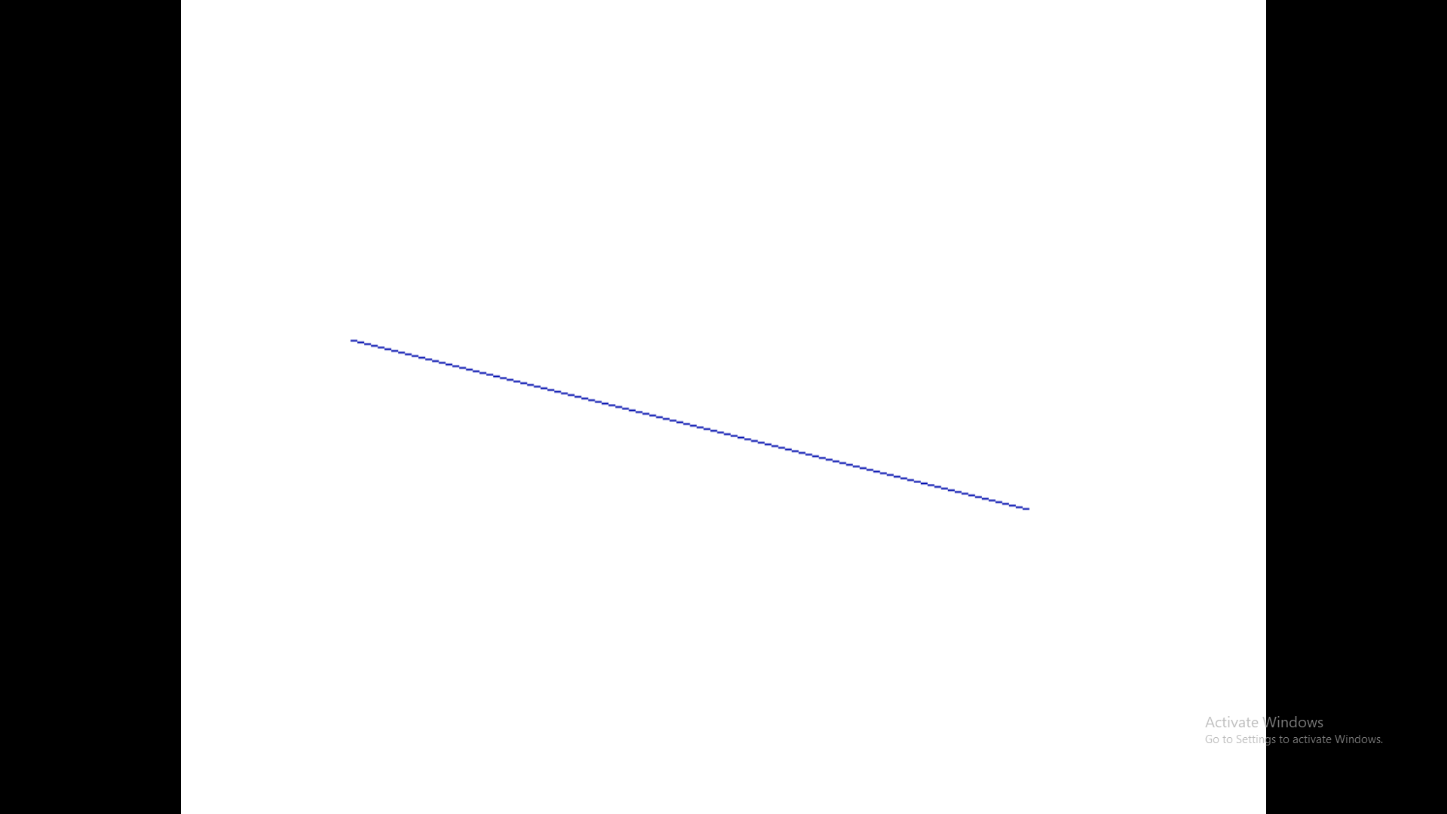
initgraph(&gdriver, &gmode, "c:\\turboc3\\bgi");

drawline(x0, y0, x1, y1);

return 0;

}

**Output :**



**Result :** Bresenham’s algorithm is implemented in C.

# Experiment 4) Midpoint Circle Algorithm

**Date :**

**Aim**: To implement Midpoint Circle Algorithm in C

**Algorithm :**

* Initially assume x =0, y =r we have p =1-r
* Repeat the steps while x ≤ y
  + Plot (x, y)
  + If (p<0) Then have p = p + 2x + 3
  + Else p = p + 2(x-y) +5 y =y - 1
  + (end if)
  + x =x+1
* (end loop)

**Code :**

#include<graphics.h>

#include<stdio.h>

void pixel(int xc,int yc,int x,int y);

int main(){

int gd,gm,xc,yc,r,x,y,p;

detectgraph(&gd,&gm);

initgraph(&gd,&gm,"C://TurboC3//BGI");

printf("Enter center of circle :");

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

printf("Enter radius of circle :");

scanf("%d",&r);

x=0;

y=r;

p=1-r;

pixel(xc,yc,x,y);

while(x<y)

{ if(p<0) {

x++;

p=p+2\*x+1;

}

else{

x++;

y--;

p=p+2\*(x-y)+1;

}

pixel(xc,yc,x,y);

}

getch();

closegraph();

return 0;

}

void pixel(int xc,int yc,int x,int y) {

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

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

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

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

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

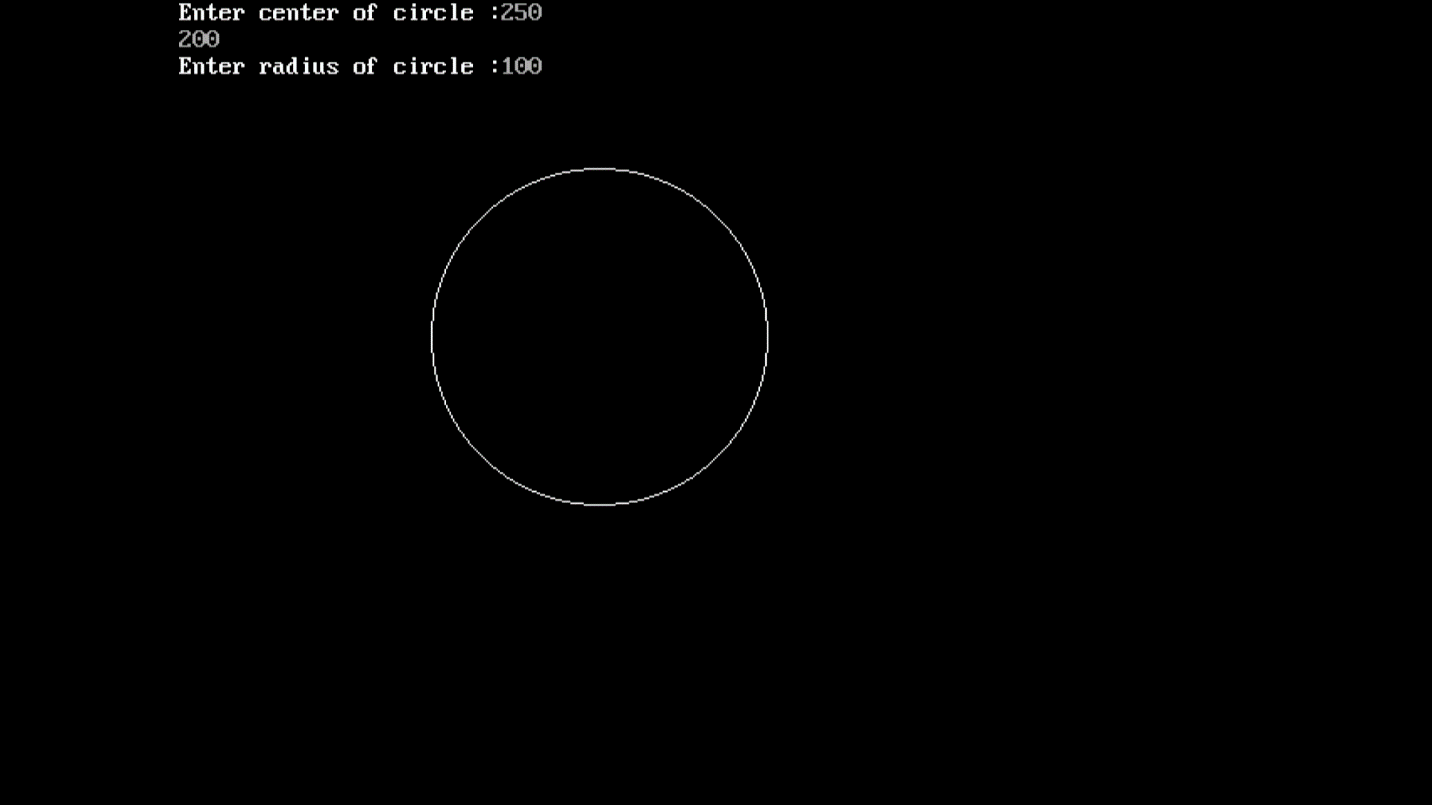
putpixel(xc+y,yc-x,WHITE);

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

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

}

**Output :**

****

**Result :** Mid point circle algorithm is implemented in C.

# Experiment 5) Midpoint Ellipse Algorithm

**Date :**

**Aim**: To implement Midpoint Ellipse Algorithm in C

**Algorithm :**

* Get input radius along x axis and y axis and find centre of ellipse.
* We assume the centre of ellipse to be at origin and the first point as:
* (x, y0)= (0, ry).
* Find the initial decision parameter for region 1.
* For every xk position in region 1 :
  + If p1k<0 then the next point along the is (xk+1 , yk)
  + Else, the next point is (xk+1, yk-1 ).
* Obtain the initial value in region 2 using the last point (x0, y0) of region 1.
* At each yk in region 2 starting at k =0 check the condition.
  + If p2k>0 the next point is (xk, yk-1)
  + Else, the next point is (xk+1, yk -1)
* Now obtain the symmetric points in the three quadrants and plot the coordinate value as: x=x+xc, y=y+yc
* Repeat the steps for region 1 until 2ry2x & gt=2rx2y

**Code :**

#include <graphics.h>

#include <stdlib.h>

#include <math.h>

#include <stdio.h>

#include <conio.h>

main (){

float x,y,a, b,r,p,h,k,p1,p2;

h=319;

k=239;

a=50;

b=40;

int gdriver = DETECT,gmode, errorcode;

int midx, midy, i;

initgraph (&gdriver, &gmode, " ");

x=0;

y=b;

p1 = ((b \* b)-(a \* a \* b) + (a \* a)/4);

putpixel (x+h, y+k, RED);

putpixel (-x+h, -y+k, RED);

putpixel (x+h, -y+k, RED);

putpixel (-x+h, y+k, RED);

if (p1 < 0)

p1 += ((2 \*b \* b) \*(x+1))-((2 \* a \* a)\*(y-1)) + (b \* b);

else

{

p1+= ((2 \*b \* b) \*(x+1))-((2 \* a \* a)\*(y-1))-(b \* b);

y--;

}

x++;

p2 =((b \* b)\* (x + 0.5))+((a \* a)\*(y-1) \* (y-1))-(a \* a \*b \* b);

while (y>=0){

if (p2>0)

p2=p2-((2 \* a \* a)\* (y-1))+(a \*a);

else{

p2=p2-((2 \* a \* a)\* (y-1))+((2 \* b \* b)\*(x+1))+(a \* a);

x++;

}

y--;

putpixel (x+h, y+k, RED);

putpixel (-x+h, -y+k, RED);

putpixel (x+h, -y+k, RED);

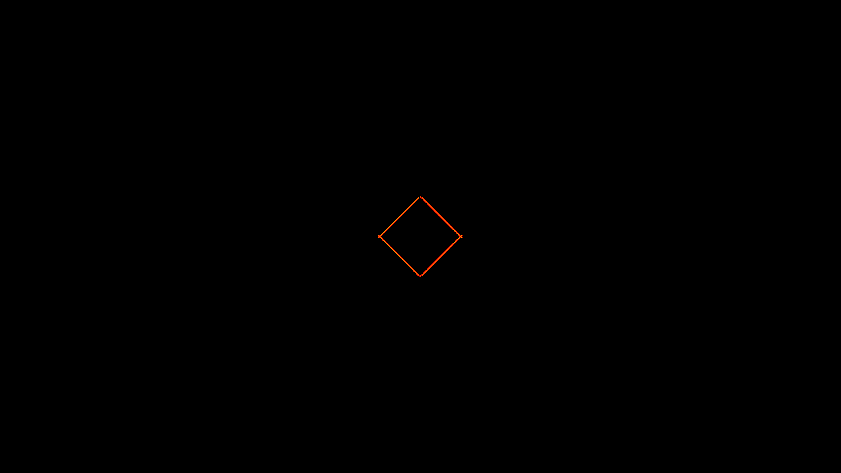
putpixel (-x+h, y+k, RED);

}

getch();

}

**Output :**



**Result :** Mid Point Ellipse Algorithm is implemented in C.

# Experiment 6) Window to viewport transformation

**Date :**

**Aim**: To implement window to viewport transformation in C.

**Code :**

#include <graphics.h>

#include<conio.h>

#include <stdio.h>

main(){

int wxmax,wymax,wxmin,wymin;

int vxmax,vymax,vxmin,vymin;

float sx,sy;

int x,x1,x2,y,y1,y2;

int gr=DETECT ,gm;

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

printf("Enter coordinates for triangle x,y");

scanf("%d %d",&x,&y);

printf("\n x1 and y1");

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

printf("\n x2 and y2");

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

printf("Enter window coordinates for triangle wxmax,wymax");

scanf("%d %d",&wxmax,&wymax);

printf("Enter window coordinates for triangle wxmin,wymin");

scanf("%d %d",&wxmin,&wymin);

cleardevice();

delay(50);

rectangle(wxmin,wymin,wxmax,wymax);

outtextxy(wxmin,wymin-10,"Window");

line(x,y,x1,y1);

line(x1,y1, x2,y2);

line(x2,y2,x,y);

vxmin=300; vymin=30; vxmax=550; vymax=350;

rectangle(vxmin,vymin,vxmax,vymax);

outtextxy(vxmin,vymin-10,"Viewport");

sx=(float)(vxmax-vxmin)/(wxmax-wxmin);

sy=(float)(vymax-vymin)/(wymax-wymin);

x=vxmin+(float)((x-wxmin)\*sx);

y=vymin+(float)((y-wymin)\*sy);

x1=vxmin+(float)((x1-wxmin)\*sx);

x2=vxmin+(float)((x2-wxmin)\*sx);

y1=vymin+(float)((y1-wymin)\*sx);

y2=vymin+(float)((y2-wymin)\*sx);

line(x,y,x1,y1);

line(x1,y1, x2,y2);

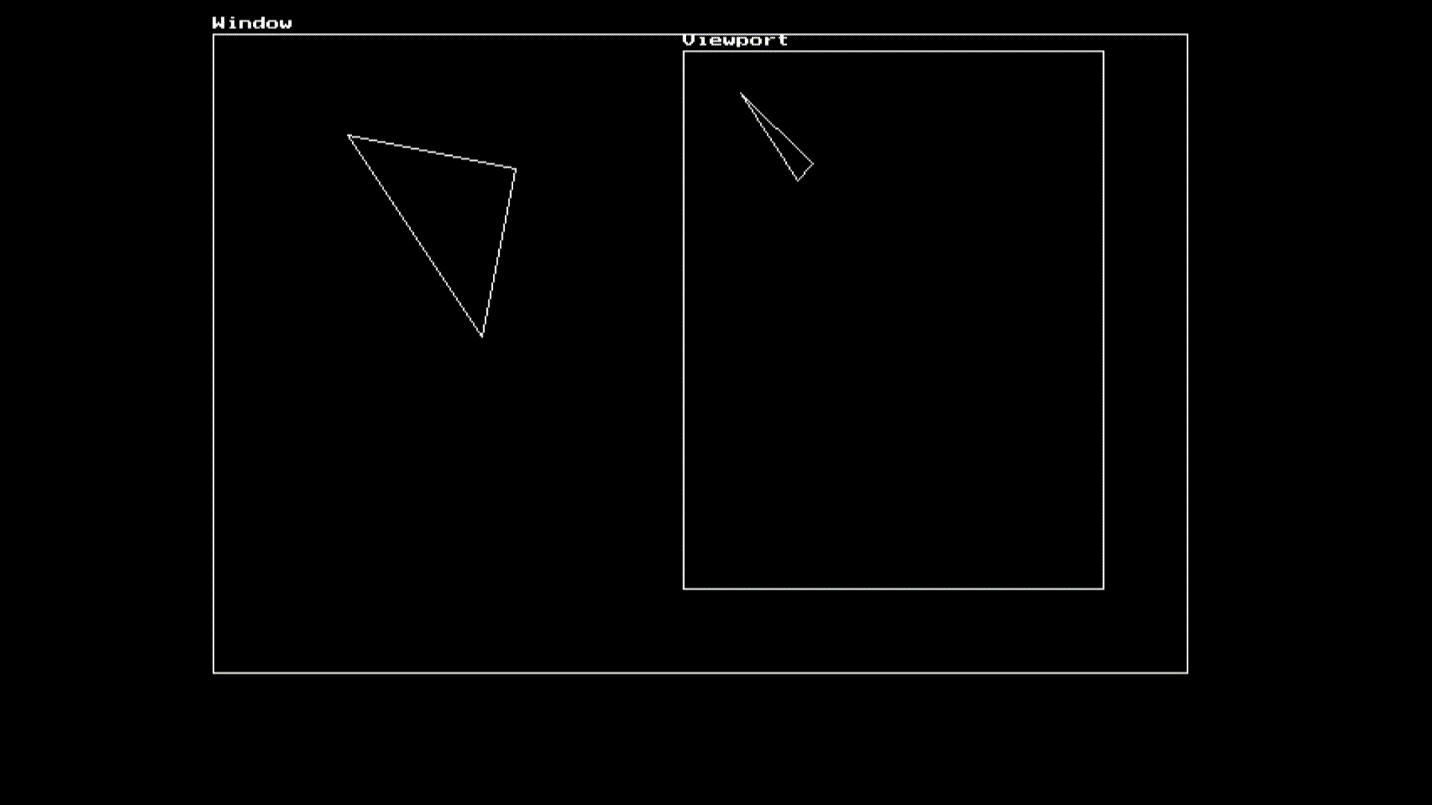
line(x2,y2,x,y);

getch();

closegraph();

}

**Output :**



**Result :** Window to viewport transformation is carried out in C.

# Experiment 7) Liang Barsky Clipping Algorithm

**Date :**

**Aim**: To implement Liang Barsky clipping algorithm in C.

**Algorithm :**

* Assume tmin=0 and tmax=1
* Calculate the values of tL,tR,tT and tB. If t<tmin or t<tmax ignore it and go to the next edge Otherwise classify the t value as entering or exiting value
  + If t is entering value set tmin= t
  + if t is exiting value set tmax=t
* If tmin< tmax then draw a line from (x1 + dx\*tmin, y1 + dy\*tmin) to (x1 + dx\*tmax, y1 + dy\*tmax)
* If the line crosses over the window, you will see (x1 + dx\*tmin, y1 + dy\*tmin) and (x1 + dx\*tmax , y1 + dy\*tmax) are intersection between line and edge.

**Code :**

#include<stdio.h>

#include<graphics.h>

#include<math.h>

#include<dos.h>

main(){

int i,gd=DETECT,gm;

int x1,y1,x2,y2,xmin,xmax,ymin,ymax,xx1,xx2,yy1,yy2,dx,dy;

float t1,t2,p[4],q[4],temp;

x1=120; y1=120;

x2=300; y2=300;

xmin=100; ymin=100;

xmax=250; ymax=250;

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

rectangle(xmin,ymin,xmax,ymax);

dx=x2-x1; dy=y2-y1;

p[0]=-dx; p[1]=dx;

p[2]=-dy; p[3]=dy;

q[0]=x1-xmin;

q[1]=xmax-x1;

q[2]=y1-ymin;

q[3]=ymax-y1;

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

if(p[i]==0){

printf("line is parallel to one of the clipping boundary");

if(q[i]>=0){

if(i<2) {

if(y1<ymin) {

y1=ymin;

}

if(y2>ymax)

y2=ymax;

}

line(x1,y1,x2,y2);

}

if(i>1) {

if(x1<xmin) {

x1=xmin;

}

if(x2>xmax) {

x2=xmax;

}

line(x1,y1,x2,y2);

}

}

t1=0;

t2=1;

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

temp=q[i]/p[i];

if(p[i]<0) {

if(t1<=temp)

t1=temp;

}

else{

if(t2>temp)

t2=temp;

}

if(t1<t2) {

xx1 = x1 + t1 \* p[1];

xx2 = x1 + t2 \* p[1];

yy1 = y1 + t1 \* p[3];

yy2 = y1 + t2 \* p[3];

line(xx1,yy1,xx2,yy2);

}

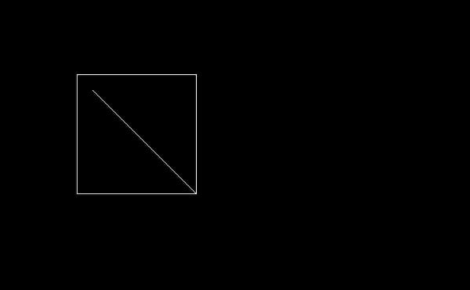
}

delay(3000);

closegraph();

}

**Output :**



**Result :** Liang Barsky Clipping algorithm is implemented in C.

# Experiment 8) Cohen Sutherland Clipping Algorithm

**Date :**

**Aim**: To implement Cohen Sutherland clipping algorithm in C++.

**Algorithm :**

* Assign a region code for two endpoints of given line
* If both endpoints have a region code 0000 then given line is completely inside and we will keep this line
* If step 2 fails, perform the logical AND operation for both region codes.
  + If the result is not 0000, then given line is completely outside.
  + Else line is partially inside.
    - Choose an endpoint of the line that is outside the given rectangle.
    - Find the intersection point of the rectangular boundary (based on region code).
    - Replace endpoint with the intersection point and update the region code.
    - Repeat step 2 until we find a clipped line either trivially accepted or rejected.
* Repeat step 1 for new inputs.

**Code :**

#include <stdio.h>

class CohenSutherLandAlgo {

private:

double x1,y1,x2,y2;

double x\_max,y\_max,x\_min,y\_min;

const int INSIDE = 0;

const int LEFT = 1;

const int RIGHT = 2;

const int BOTTOM = 4;

const int TOP = 8;

public:

CohenSutherLandAlgo() {

x1 = 0.0;

x2 = 0.0;

y1 = 0.0;

y2 = 0.0;

}

void getCoordinates();

void getClippingRectangle();

int generateCode(double x, double y);

void cohenSutherland();

};

void CohenSutherLandAlgo::getCoordinates(){

printf("\nEnter Co-ordinates of P1(X1,Y1) of Line Segment : ");

printf("\Enter X1 Co-ordinate : ");

scanf( "%lf",&x1);

printf( "\Enter Y1 Co-ordinate : ");

scanf( "%lf",&y1);

printf("\nEnter Co-ordinates of P2(X2,Y2) of Line Segment : ");

printf( "\Enter X2 Co-ordinate : ");

scanf( "%lf",& x2);

printf( "\Enter Y2 Co-ordinate : ");

scanf( "%lf",& y2);

}

void CohenSutherLandAlgo::getClippingRectangle(){

printf( "\nEnter the Co-ordinates of Interested Rectangle.");

printf( "\nEnter the X\_MAX : ");

scanf( "%lf",&x\_max);

printf( "\nEnter the Y\_MAX : ");

scanf( "%lf",& y\_max);

printf( "\nEnter the X\_MIN : ");

scanf( "%lf",& x\_min);

printf( "\nEnter the Y\_MIN : ");

scanf( "%lf",& y\_min);

}

int CohenSutherLandAlgo::generateCode(double x, double y) {

int code = INSIDE;

if (x < x\_min)

code |= LEFT;

else if (x > x\_max)

code |= RIGHT;

if (y < y\_min)

code |= BOTTOM;

else if (y > y\_max

code |= TOP;

return code;

}

void CohenSutherLandAlgo::cohenSutherland() {

int code1 = generateCode(x1, y1);

int code2 = generateCode(x2, y2);

bool accept = false;

while (true) {

if ((code1 == 0) && (code2 == 0)) {

accept = true;

break;

}

else if (code1 & code2) {

break;

}

else {

int code\_out;

double x, y;

if (code1 != 0)

code\_out = code1;

else

code\_out = code2;

if (code\_out & TOP) {

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

y = y\_max;

}

else if (code\_out & BOTTOM) {

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

y = y\_min;

}

else if (code\_out & RIGHT) {

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

x = x\_max;

}

else if (code\_out & LEFT) {

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

x = x\_min;

}

if (code\_out == code1) {

x1 = x;

y1 = y;

code1 = generateCode(x1, y1);

}

else{

x2 = x;

y2 = y;

code2 = generateCode(x2, y2);

}

}

}

if (accept) {

printf("Line accepted from ( %lf %lf ) to (%lf %lf )",x1 ,y1,x2,y2 );

}

else

printf("Line can't be drawn");

}

int main() {

CohenSutherLandAlgo c;

c.getCoordinates();

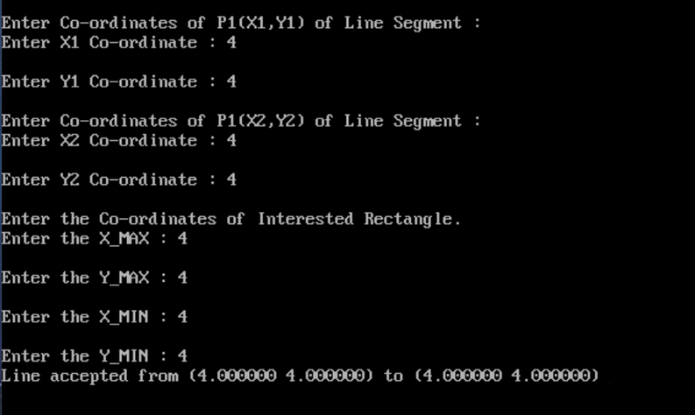
c.getClippingRectangle();

c.cohenSutherland();

return 0;

}

**Output :**



**Result :** Cohen Sutherland Algorithm for line clipping is implemented in C++

# Experiment 9) 2D transformation of an object

**Date :**

**Aim**: To implement 2D transformation of an object in C

**Algorithm :**

* Take 2\*2 matrix
* For each point of (1) make a 2\*1 matrix P where P[0][0] equals to the x coordinate of the point and P[1][0] equals to the y coordinate of the point.
* Multiply the matrix with the point P to get new co-ordinate.
* Draw the final result using the new points.

**Code :**

#include<stdio.h>

#include<graphics.h>

void findNewCoordinate(int s[][2], int p[][1])

{

int temp[2][1] = { 0 };

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

for (int j = 0; j < 1; j++)

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

p[i][j] += (s[i][k] \* p[k][j]);

p[0][0] = temp[0][0];

p[1][0] = temp[1][0];

}

void scale(int x[], int y[], int sx, int sy)

{

line(x[0], y[0], x[1], y[1]);

line(x[1], y[1], x[2], y[2]);

line(x[2], y[2], x[0], y[0]);

int s[2][2] = { sx, 0, 0, sy };

int p[2][1];

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

{

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

p[1][0] = y[i];

findNewCoordinate(s, p);

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

y[i] = p[1][0];

}

line(x[0], y[0], x[1], y[1]);

line(x[1], y[1], x[2], y[2]);

line(x[2], y[2], x[0], y[0]);

}

int main()

{

int x[] = { 100, 200, 400 };

int y[] = { 200, 100, 100 };

int sx = 2, sy = 2;

int gd, gm;

detectgraph(&gd, &gm);

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

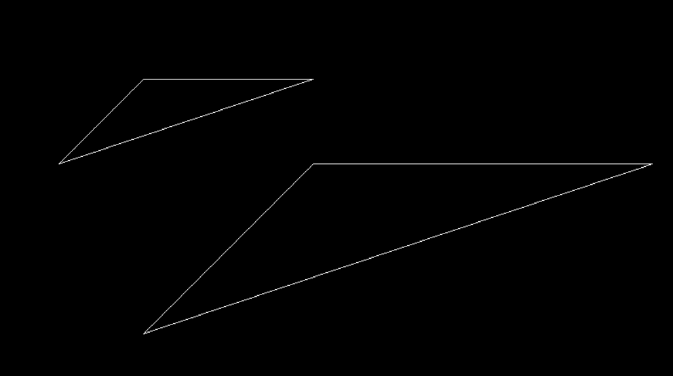
scale(x, y, sx,sy);

getch();

return 0;

}

**Output :**



**Result :** 2D transformation of an object is carried out in C