**GRAPHICS LAB REPORT**

**CLASS-UG-III SECTION:A1**

**NAME:** ARKA DEBNATH

**ROLL NO:** 001610501007

**CONTENTS**

|  |  |  |
| --- | --- | --- |
| **Sl. No.** | **Topic** | **Page no.** |
| 1. | Common Code | 3-6 |
| 2. | DDA Line Drawing Algorithm | 7-8 |
| 3. | Bressenham Line Drawing Algorithm | 9-10 |
| 4. | Midpoint circle drawing Algorithm |  |
| 5. | Bressenham Circle Drawing Algorithm |  |
| 6. | Ellipse Drawing Algorithm |  |
| 7. | Flood fill Algorithm |  |
| 8. | Boundary fill Algorithm |  |
| 9. | Scanline fill Algorithm |  |
| 10. | Transformation: Translate |  |
| 11. | Transformation: Scale |  |
| 12. | Transformation: Rotate |  |
| 13. | Transformation: Shear |  |
| 14. | Transformation: Reflection |  |
| 15. | Cohen Sutherland line clipping algorithm |  |
| 16. | Sutherland Hodgman polygon clipping algorithm |  |
| 17. | Bezier curve |  |

**Common Code**

This is the code for the common part which include grid drawing, point drawing, etc.

#include "mainwindow.h"

#include "ui\_mainwindow.h"

#include <QPixmap>

#include <QImage>

#include <iostream>

#include <string>

#include <QMouseEvent>

#include <QPainter>

#include <QPaintDevice>

#include <QPoint>

#include <unistd.h>

#include <vector>

**using** **namespace** std;

QImage img=QImage(700,700,QImage::Format\_RGB888);

MainWindow::MainWindow(QWidget \*parent) :

QMainWindow(parent),

ui(**new** Ui::MainWindow)

{

ui->setupUi(**this**);

ui->x\_axis->hide();

ui->y\_axis->hide();

connect(ui->frame,SIGNAL(Mouse\_Pos()),**this**,SLOT(Mouse\_Pressed()));

connect(ui->frame,SIGNAL(sendMousePosition(QPoint&)),**this**,SLOT(showMousePosition(QPoint&)));

}

MainWindow::~MainWindow()

{

**delete** ui;

}

void MainWindow::point(int x,int y,int r,int g,int b)

{

int k = ui->gridsize->value();*//GridSize*

**if**(k>1)

{

int startX=x/k\*k;

int startY=y/k\*k;

*//Drawing the pixels*

**for**(int i=startX+1;i<(startX+k);i++)

**for**(int j=startY+1;j<(startY+k);j++)

img.setPixel(i,j,qRgb(r,g,b));

}

**else**

img.setPixel(x,y,qRgb(r,g,b));

ui->frame->setPixmap(QPixmap::fromImage(img));

}

int MainWindow::changeX(int x)

{

int k = ui->gridsize->value();

x=x-img.width()/2;

x/=k;

**return** x;

}

int MainWindow::changeY(int y)

{

int k = ui->gridsize->value();

y=img.height()/2-y;

y/=k;

**return** y;

}

void MainWindow::showMousePosition(QPoint &pos)

{

ui->mouse\_movement->setText(" X : "+QString::number(changeX(pos.x()))+", Y : "+QString::number(changeY(pos.y())));

}

void MainWindow::Mouse\_Pressed()

{

ui->mouse\_pressed->setText(" X : "+QString::number(changeX(ui->frame->x))+", Y : "+QString::number(changeY(ui->frame->y)));

*//point(ui->frame->x,ui->frame->y);*

ui->x\_axis->move(0,ui->frame->y);

ui->y\_axis->move(ui->frame->x,0);

}

void MainWindow::on\_show\_axes\_clicked()

{

**if**(ui->show\_axes->isChecked())

{

*//Draw Y-axis*

**for**(int i=0;i<=img.height();i++)

point(i,img.width()/2,0,247,255);

*//Draw X-axis*

**for**(int i=0;i<=img.width();i++)

point(img.height()/2,i,0,247,255);

}

**else**{

*//Draw Y-axis*

**for**(int i=0;i<=img.height();i++)

point(i,img.width()/2,0,0,0);

*//Draw X-axis*

**for**(int i=0;i<=img.width();i++)

point(img.height()/2,i,0,0,0);

}

}

void MainWindow::on\_set\_point1\_clicked()

{

**if**(ui->draw\_line->isChecked()){

p1.setX(ui->frame->x);

p1.setY(ui->frame->y);

}

}

void MainWindow::on\_set\_point2\_clicked()

{

**if**(ui->draw\_line->isChecked()){

p2.setX(ui->frame->x);

p2.setY(ui->frame->y);

}

}

Resulting grid

|  |
| --- |
|  |

**DDA Line Drawing Algorithm**

void MainWindow::on\_dda\_clicked()

{

*//This function draws a line between the two selected points using DDA algorithm*

int k = ui->gridsize->value();

*//Now p1 and p2 contains the points*

int x1=(p1.x()/k);

int y1=(p1.y()/k);

int x2=(p2.x()/k);

int y2=(p2.y()/k);

*// here we implement the algorithm*

int dx=x2-x1;

int dy=y2-y1;

int steps=0; *//number of steps to put pixel*

**if**(abs(dx)>abs(dy))

steps=abs(dx);

**else**

steps=abs(dy);

double Xinc=(double)dx/(double)steps;

double Yinc=(double)dy/(double)steps;

double x=x1\*k+k/2;

double y=y1\*k+k/2;

**for**(int v=0; v <steps; v++)

{

x = x + Xinc\*k;

y = y + Yinc\*k;

point((int)(x+0.5),(int)(y+0.5));

}

}

Output:

|  |
| --- |
|  |

**Bressenham Line Drawing Algorithm**

void MainWindow::on\_bress\_clicked()

{

int k = ui->gridsize->value();

*//Store the two points*

int x1=p1.x();

int y1=p1.y();

int x2=p2.x();

int y2=p2.y();

int dx=x2-x1;

int dy=y2-y1;

x1=(x1/k)\*k+k/2;

y1=(y1/k)\*k+k/2;

x2=(x2/k)\*k+k/2;

y2=(y2/k)\*k+k/2;

int xinc=(dx>0)?k:-k;

int yinc=(dy>0)?k:-k;

dx=abs(dx);

dy=abs(dy);

*//Case for gentle slope*

**if**(dx>dy)

{

int p=2\*(dy)-dx;

int y=y1;

**for**(int x=x1; x!=x2; x+=xinc)

{

point(x,y);

**if**(p>=0)

{

y+=yinc;

p-=2\*dx;

}

p+=2\*dy;

}

}

*//Case for steep slope*

**else**

{

int p=2\*(dx)-dy;

int x=x1;

**for**(int y=y1; y!=y2; y+=yinc)

{

point(x,y);

**if**(p>=0)

{

x+=xinc;

p-=2\*(dy);

}

p+=2\*(dx);

}

}

}

Output:

|  |
| --- |
|  |

**Midpoint Circle Drawing Algorithm**

void MainWindow::on\_midpt\_clicked()

{

*//Get the radius*

int r0=ui->circle\_radius->value();

*//Set the centre*

**if**(ui->draw\_circle->isChecked()){

p1.setX(ui->frame->x);

p1.setY(ui->frame->y);

drawCircle(p1,r0);

}

}

void MainWindow::drawCircle(QPoint p1, int r0)

{

*//Function to draw the circle*

int x\_centre=p1.x();

int y\_centre=p1.y();

int k = ui->gridsize->value();*//GridSize*

x\_centre=(x\_centre/k)\*k+k/2;

y\_centre=(y\_centre/k)\*k+k/2;

int x=r0\*k;

int y=0;

point(x+x\_centre,y+y\_centre);

**if**(r0>0)

{

point(x+x\_centre,-y+y\_centre);

point(y+x\_centre,x+y\_centre);

point(-y+x\_centre,x+y\_centre);

}

int P=(1-r0)\*k;

**while**(x>y)

{

y++;

**if**(P<=0)

P=P+2\*y+1;

**else**

{

x--;

P=P+2\*y-2\*x+1;

}

**if**(x<y)

**break**;

point(x+x\_centre,y+y\_centre);

point(-x+x\_centre,y+y\_centre);

point(x+x\_centre,-y+y\_centre);

point(-x+x\_centre,-y+y\_centre);

**if**(x!=y)

{

point(y+x\_centre,x+y\_centre);

point(-y+x\_centre,x+y\_centre);

point(y+x\_centre,-x+y\_centre);

point(-y+x\_centre,-x+y\_centre);

}

}

}

Output:

|  |
| --- |
|  |

**Bressenham Circle Drawing Algorithm**

void MainWindow::on\_bress\_circle\_clicked()

{

//Get the radius

int r0=ui->circle\_radius->value();

//Set the centre

if(ui->draw\_circle->isChecked()){

p1.setX(ui->frame->x);

p1.setY(ui->frame->y);

drawCircleBress(p1,r0);

}

}

void MainWindow::drawCircleBress(QPoint p1, int r0)

{

//Function to draw the circle

int x\_centre=p1.x();

int y\_centre=p1.y();

int k = ui->gridsize->value();//GridSize

x\_centre=(x\_centre/k)\*k+k/2;

y\_centre=(y\_centre/k)\*k+k/2;

int y=r0\*k;

int x=0;

int d=(3-2\*r0)\*k;

while(y>=x)

{

point(x\_centre+x,y\_centre+y);

point(x\_centre+x,y\_centre-y);

point(x\_centre-x,y\_centre+y);

point(x\_centre-x,y\_centre-y);

point(x\_centre+y,y\_centre+x);

point(x\_centre+y,y\_centre-x);

point(x\_centre-y,y\_centre+x);

point(x\_centre-y,y\_centre-x);

x++;

if(d>0)

{

y--;

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

}

else

{

d=d+4\*x+6;

}

}

}

Output:

|  |
| --- |
|  |

**Ellipse Drawing Algorithm**

void MainWindow::on\_ellipse\_clicked()

{

*//Get the radius*

int rx=ui->ellipse\_rx->value();

int ry=ui->ellipse\_ry->value();

*//Set the centre*

**if**(ui->draw\_ellipse->isChecked()){

p1.setX(ui->frame->x);

p1.setY(ui->frame->y);

drawEllipse(p1,rx,ry);

}

}

void MainWindow::drawEllipse(QPoint p, int rx, int ry)

{

*//Function to draw the ellipse*

*//Get the centre*

int x\_centre=p.x();

int y\_centre=p.y();

int k = ui->gridsize->value();*//GridSize*

x\_centre=(x\_centre/k)\*k+k/2;

y\_centre=(y\_centre/k)\*k+k/2;

int x=0;

int y=ry;

int rx2=rx\*rx;

int ry2=ry\*ry;

int tworx2=2\*rx2;

int twory2=2\*ry2;

int px=0.0;

int py=tworx2\*y;

*//For first region*

int p1=ry2-rx2\*ry+(0.25)\*rx2; *//Initial value of decision paramemter*

**while**(px<py)

{

point(x\_centre+x\*k,y\_centre+y\*k);

point(x\_centre-x\*k,y\_centre+y\*k);

point(x\_centre-x\*k,y\_centre-y\*k);

point(x\_centre+x\*k,y\_centre-y\*k);

x++;

px+=twory2;

**if**(p1>=0)

{

y--;

py-=tworx2;

p1=p1+ry2+px-py;

}

**else**

{

p1=p1+ry2+px;

}

}

*//For second region*

p1=ry2\*((double)x+0.5)\*((double)x+0.5)+rx2\*(y-1)\*(y-1)-rx2\*ry2; *//Initial value of decision paramemter*

**while**(y>=0)

{

point(x\_centre+x\*k,y\_centre+y\*k);

point(x\_centre-x\*k,y\_centre+y\*k);

point(x\_centre-x\*k,y\_centre-y\*k);

point(x\_centre+x\*k,y\_centre-y\*k);

y--;

py-=tworx2;

**if**(p1<=0)

{

x++;

px+=twory2;

p1=p1+rx2-py+px;

}

**else**

{

p1=p1+rx2-py;

}

}

}

Output:

|  |
| --- |
|  |

**Flood Fill Algorithm**

void MainWindow::on\_flood\_fill\_clicked()

{

*//Function to implement flood fill*

int x1=p1.x();

int y1=p1.y();

int k = ui->gridsize->value();

x1=(x1/k)\*k+k/2;

y1=(y1/k)\*k+k/2;

flood\_fill\_util(x1,y1,k,img.pixel(x1,y1),255,255,255);

}

void MainWindow::flood\_fill\_util(int x1, int y1,int k, QRgb q1, int r,int g,int b)

{

**if**(x1<=0 || x1>img.width()|| y1<=0 || y1>img.height())

**return**;

**if**(img.pixel(x1,y1)!=q1)

**return**;

**if**(img.pixel(x1,y1)==qRgb(r,g,b))

**return**;

point(x1,y1,255,255,255);

flood\_fill\_util(x1+k,y1,k,q1,r,g,b);

flood\_fill\_util(x1-k,y1,k,q1,r,g,b);

flood\_fill\_util(x1,y1+k,k,q1,r,g,b);

flood\_fill\_util(x1,y1-k,k,q1,r,g,b);

}

Output:

|  |
| --- |
|  |

**Boundary Fill Algorithm**

void MainWindow::on\_boundary\_fill\_clicked()

{

*//Function to implement boundary fill*

int x1=p1.x();

int y1=p1.y();

int k = ui->gridsize->value();

x1=(x1/k)\*k+k/2;

y1=(y1/k)\*k+k/2;

boundary\_fill\_util(x1,y1,k,qRgb(255,255,0),255,255,255);

}

void MainWindow::boundary\_fill\_util(int x1, int y1, int k, QRgb edgecolor, int r, int g, int b)

{

QRgb current;

current=img.pixel(x1,y1);

**if**(x1<=0 || x1>img.width()|| y1<=0 || y1>img.height())

**return**;

**if**(current!=edgecolor && current!=qRgb(r,g,b))

{

point(x1,y1,255,255,255);

boundary\_fill\_util(x1+k,y1,k,edgecolor,r,g,b);

boundary\_fill\_util(x1-k,y1,k,edgecolor,r,g,b);

boundary\_fill\_util(x1,y1+k,k,edgecolor,r,g,b);

boundary\_fill\_util(x1,y1-k,k,edgecolor,r,g,b);

}

}

Output:

|  |
| --- |
|  |

**Scan Line Filling Algorithm**

*// ============================== SCAN LINE FILL =================================*

void MainWindow::on\_scanline\_clicked()

{

*// for initialising*

EdgeList.clear();

initEdgeTable();

}

void MainWindow::on\_set\_vertex\_clicked()

{

int k=ui->gridsize->value();

int x=((ui->frame->x)/k)\*k+k/2;

int y=((ui->frame->y)/k)\*k+k/2;

EdgeList.push\_back(make\_pair(x,y));

int i=EdgeList.size();

**if**(EdgeList.size()>1)

{

storeEdgeInTable(EdgeList[i-2].first, EdgeList[i-2].second, EdgeList[i-1].first, EdgeList[i-1].second);*//storage of edges in edge table.*

p1.setX(EdgeList[EdgeList.size()-1].first);

p2.setX(EdgeList[EdgeList.size()-2].first);

p1.setY(EdgeList[EdgeList.size()-1].second);

p2.setY(EdgeList[EdgeList.size()-2].second);

on\_bress\_clicked();

}

}

void MainWindow::initEdgeTable()

{

int i;

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

{

EdgeTable[i].countEdgeBucket = 0;

}

ActiveEdgeTuple.countEdgeBucket = 0;

}

void MainWindow::insertionSort(EdgeTableTuple \*ett)

{

int i,j;

EdgeBucket temp;

**for** (i = 1; i < ett->countEdgeBucket; i++)

{

temp.ymax = ett->buckets[i].ymax;

temp.xofymin = ett->buckets[i].xofymin;

temp.slopeinverse = ett->buckets[i].slopeinverse;

j = i - 1;

**while** ((temp.xofymin < ett->buckets[j].xofymin) && (j >= 0))

{

ett->buckets[j + 1].ymax = ett->buckets[j].ymax;

ett->buckets[j + 1].xofymin = ett->buckets[j].xofymin;

ett->buckets[j + 1].slopeinverse = ett->buckets[j].slopeinverse;

j = j - 1;

}

ett->buckets[j + 1].ymax = temp.ymax;

ett->buckets[j + 1].xofymin = temp.xofymin;

ett->buckets[j + 1].slopeinverse = temp.slopeinverse;

}

}

void MainWindow::storeEdgeInTuple (EdgeTableTuple \*receiver,int ym,int xm,float slopInv)

{

(receiver->buckets[(receiver)->countEdgeBucket]).ymax = ym;

(receiver->buckets[(receiver)->countEdgeBucket]).xofymin = (float)xm;

(receiver->buckets[(receiver)->countEdgeBucket]).slopeinverse = slopInv;

insertionSort(receiver);

(receiver->countEdgeBucket)++;

}

void MainWindow::storeEdgeInTable (int x1,int y1, int x2, int y2)

{

float m,minv;

int ymaxTS,xwithyminTS, scanline;

**if** (x2==x1)

{

minv=0.000000;

}

**else**

{

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

**if** (y2==y1)

**return**;

minv = (float)1.0/m;

}

**if** (y1>y2)

{

scanline=y2;

ymaxTS=y1;

xwithyminTS=x2;

}

**else**

{

scanline=y1;

ymaxTS=y2;

xwithyminTS=x1;

}

storeEdgeInTuple(&EdgeTable[scanline],ymaxTS,xwithyminTS,minv);

}

void MainWindow::removeEdgeByYmax(EdgeTableTuple \*Tup,int yy)

{

int i,j;

**for** (i=0; i< Tup->countEdgeBucket; i++)

{

**if** (Tup->buckets[i].ymax == yy)

{

**for** ( j = i ; j < Tup->countEdgeBucket -1 ; j++ )

{

Tup->buckets[j].ymax =Tup->buckets[j+1].ymax;

Tup->buckets[j].xofymin =Tup->buckets[j+1].xofymin;

Tup->buckets[j].slopeinverse = Tup->buckets[j+1].slopeinverse;

}

Tup->countEdgeBucket--;

i--;

}

}

}

void MainWindow::updatexbyslopeinv(EdgeTableTuple \*Tup)

{

int i;

**for** (i=0; i<Tup->countEdgeBucket; i++)

{

(Tup->buckets[i]).xofymin =(Tup->buckets[i]).xofymin + (Tup->buckets[i]).slopeinverse;

}

}

void MainWindow::on\_fill\_scan\_clicked()

{

int i, j, x1, ymax1, x2, ymax2, FillFlag = 0, coordCount;

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

{

**for** (j=0; j<EdgeTable[i].countEdgeBucket; j++)

{

storeEdgeInTuple(&ActiveEdgeTuple,EdgeTable[i].buckets[j].

ymax,EdgeTable[i].buckets[j].xofymin,

EdgeTable[i].buckets[j].slopeinverse);

}

removeEdgeByYmax(&ActiveEdgeTuple, i);

insertionSort(&ActiveEdgeTuple);

j = 0;

FillFlag = 0;

coordCount = 0;

x1 = 0;

x2 = 0;

ymax1 = 0;

ymax2 = 0;

**while** (j<ActiveEdgeTuple.countEdgeBucket)

{

**if** (coordCount%2==0)

{

x1 = (int)(ActiveEdgeTuple.buckets[j].xofymin);

ymax1 = ActiveEdgeTuple.buckets[j].ymax;

**if** (x1==x2)

{

**if** (((x1==ymax1)&&(x2!=ymax2))||((x1!=ymax1)&&(x2==ymax2)))

{

x2 = x1;

ymax2 = ymax1;

}

**else**

{

coordCount++;

}

}

**else**

{

coordCount++;

}

}

**else**

{

x2 = (int)ActiveEdgeTuple.buckets[j].xofymin;

ymax2 = ActiveEdgeTuple.buckets[j].ymax;

FillFlag = 0;

**if** (x1==x2)

{

**if** (((x1==ymax1)&&(x2!=ymax2))||((x1!=ymax1)&&(x2==ymax2)))

{

x1 = x2;

ymax1 = ymax2;

}

**else**

{

coordCount++;

FillFlag = 1;

}

}

**else**

{

coordCount++;

FillFlag = 1;

}

**if**(FillFlag)

{

p1.setX(x1);p1.setY(i);

p2.setX(x2);p2.setY(i);

on\_bress\_clicked();

}

}

j++;

}

updatexbyslopeinv(&ActiveEdgeTuple);

}

EdgeList.clear();

}

*//============================ SCAN LINE END =============================================================*

Output:

|  |
| --- |
|  |