COMPUTER GRAPHICS LAB REPORT

SOULIB GHOSH and MADHURAM JAJOO

JADAVPUR UNIVERSITY

NAME – SOULIB GHOSH	NAME – MADHURAM JAJOO
ROLL – 0016105010 47	ROLL – 0016105010 54
COURSE – BCSE III	COURSE – BCSE III
GROUP – A2	GROUP – A2

NOVEMBER 20, 2018

The following codes contain five files – my_label.h, my_label.cpp, main.cpp, mainwindow.h and mainwindow.cpp.

The following files are given. The output of the following programs are also shown.

my_label.h

```
#ifndef MY_LABEL_H
#define MY_LABEL_H

#include <QObject>
#include <QWidget>
#include <QLabel>
#include <QMouseEvent>
#include <QEvent>

class my_label : public QLabel
{
    Q_OBJECT

public:
    my_label(QWidget *parent = 0);
    int x,y;

protected:
    void mouseMoveEvent(QMouseEvent *ev);
    void mousePressEvent(QMouseEvent *ev);
signals:
    void sendMousePosition(QPoint&);
    void Mouse_Pos();
};

#endif // MY_LABEL_H
```

my_label.cpp

```
#include "my_label.h"

my_label::my_label(QWidget *parent):QLabel(parent)
{
    this->setMouseTracking(true);
}

void my_label::mouseMoveEvent(QMouseEvent *ev)
{
    QPoint mouse_pos=ev->pos();
    if(mouse_pos.x()<=this->size().width() && mouse_pos.y()<=this->size().height()){
        if(mouse_pos.x()>0 && mouse_pos.y()>0){
            emit sendMousePosition(mouse_pos);
        }
    }
}

void my_label::mousePressEvent(QMouseEvent *ev)
{
    if(ev->button()==Qt::LeftButton){
        this->x=ev->x();
        this->y=ev->y();
        emit Mouse_Pos();
    }
}
```

mainwindow.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QtGui>
#include <QtCore>
#include <QColorDialog>
#include<QVector>

namespace Ui {
class MainWindow;
}

class MainWindow : public QMainWindow
{
    Q_OBJECT
public slots:
    void Mouse_Pressed();
    void showMousePosition(QPoint& pos);
public:
    explicit MainWindow(QWidget *parent = 0);
    ~MainWindow();

private slots:
    void on_show_axes_clicked();
    void on_Draw_clicked();
    void on_set_point1_clicked();
    void on_set_point2_clicked();
    void on_pushButton_clicked();
    int changeX(int x);
    int changeY(int y);
    void on_show_grid_clicked();
    void drawLine(QPoint, QPoint, int);
    void drawCircle(QPoint, int, int);
    void drawEllipse(QPoint,int,int);
    void on_set_point_3_clicked();
    void on_Fill_clicked();
    void translate_object(int x,int y);
    void on_draw_ellipse_2_clicked();
    void on_Draw_2_clicked();
    void on_Draw_3_clicked();
    void on_Draw_4_clicked();
    void on_Draw_5_clicked();
    void on_Draw_6_clicked();
    void on_Draw_7_clicked();
    void on_Draw_8_clicked();
    void on_draw_ellipse_3_clicked();
    void on_Draw_9_clicked();
    void on_comboBox_3_activated(const QString &arg1);
    void on_Draw_10_clicked();
    void on_Draw_11_clicked();
    void on_Fill_2_clicked();
    void on_Draw_13_clicked();
    void on_Draw_12_clicked();

private:
    bool flag = false;
    QVector<QPoint> vertices;
    Ui::MainWindow *ui;
    QPoint p1,p2;
    void point(int,int);
    void point(int ,int , int , int, int);
    void drawDDA(QPoint, QPoint);
    void drawBresenham(QPoint, QPoint);
    void BresGentle(int,int,int,int);
    void BresSteep(int,int,int,int);
    void drawCirBres(QPoint, int);
    void drawCirMidPt(QPoint, int);
    void floodFill(QPoint);
    void floodFillRec(int, int, int, QRgb, int, int, int);
    void boundaryFill(QPoint);
    void boundaryFillRec(int, int, int, int, int, int, int, int, int);
    void scanLineFill(QPoint);
    int computeCode(double,double);
    int x_intersect(int,int,int,int,int,int,int,int);
    int y_intersect(int,int,int,int,int,int,int,int);
    void suthHodgClip();
    void clip(int,int,int,int);

};
#endif // MAINWINDOW_H
```

main.cpp

```
#include "mainwindow.h"
#include <QApplication>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    MainWindow w;
    w.show();

    return a.exec();
}
```

mainwindow.cpp

```
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include <QPixmap>
#include <QImage>
#include <iostream>
#include <QMouseEvent>
#include <QPainter>
#include <QPaintDevice>
#include <QPoint>
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&)));
    ui->comboBox->addItem("DDA Algorithm");
    ui->comboBox->addItem("Bresenham Algorithm");
}

MainWindow::~MainWindow()
{
    delete ui;
}

int MainWindow::changeX(int x){
    int k=ui->grid_size->value();
    x-=img.width()/2;
    x/=k;
    return x;
}

int MainWindow::changeY(int y){
    int k=ui->grid_size->value();
    y=img.width()/2-y;
    y/=k;
    return y;
}

void MainWindow::point(int x,int y)
{
    int k=ui->grid_size->value();
    if(k==1)
        img.setPixel(x,y,qRgb(255,255,0));
    else{
        int i,j;
        int stx=(x/k)*k;
        int sty=(y/k)*k;
        for(i=stx+1;i<stx+k;i++){
            for(j=sty+1;j<sty+k;j++)
                img.setPixel(i,j,qRgb(255,255,0));
        }
    }
    ui->frame->setPixmap(QPixmap::fromImage(img));
}

void MainWindow::point(int x,int y, int r, int g, int b)
{
    int k=ui->grid_size->value();
    if(k==1)
        img.setPixel(x,y,qRgb(r,g,b));
    else{
        int i,j;
        int stx=(x/k)*k;
        int sty=(y/k)*k;
        for(i=stx+1;i<stx+k;i++){
            for(j=sty+1;j<sty+k;j++)
                img.setPixel(i,j,qRgb(r,g,b));
        }
    }
    //ui->frame->setPixmap(QPixmap::fromImage(img));
}
```

```

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);
    //point(ui->frame->x, ui->frame->y,255,255,0);

    if(flag){
        p1.setX((ui->frame->x));
        p1.setY((ui->frame->y));

        if(vertices.size() > 0 && p1 == vertices[0]){

            flag = false;
            return;

        }

        vertices.push_back(p1);
    }
}

void MainWindow::on_show_axes_clicked()
{
    if(ui->show_axes->isChecked())
    {
        for(int i=0;i<img.height();i++)
            point(i,img.width()/2,0,255,255);
        for(int j=0;j<img.width();j++)
            point(img.height()/2,j,0,255,255);
    }
    else{
        for(int i=0;i<img.height();i++)
            point(i,img.width()/2,0,0,0);
        for(int j=0;j<img.width();j++)
            point(img.height()/2,j,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);
    }
```

```
}

void MainWindow::on_Draw_clicked()
{
    //int r0=ui->circle_radius->value();
    //QPainter painter(&img);
    //QPen pen;
    //pen.setWidth(1);
    //pen.setColor(Qt::red);
    if(ui->draw_circle->isChecked()){
        p1.setX(ui->frame->x);
        p1.setY(ui->frame->y);
        /*painter.setPen(pen);
        painter.drawEllipse(p1,r0,r0);*/
        drawCircle(p1,ui->circle_radius->value(),ui->comboBox_2->currentIndex());
    }
    if(ui->draw_line->isChecked()){
        //painter.setPen(Qt::red);
        drawLine(p1,p2,ui->comboBox->currentIndex());
    }
    if(ui->draw_ellipse->isChecked()){
        p1.setX(ui->frame->x);
        p1.setY(ui->frame->y);
        drawEllipse(p1,ui->ellipse_maj->value(),ui->ellipse_min->value());
    }


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

void MainWindow::on_pushButton_clicked()
{
    for(int j=0;j<img.height();j++)
    {
        for(int i=0;i<img.width();i++)
        {
            img.setPixel(i,j,qRgb(255,255,255));
        }
    }
    ui->frame->setPixmap(QPixmap::fromImage(img));
}

void MainWindow::on_show_grid_clicked()
{
    int i,j,k=ui->grid_size->value();
    if(ui->show_grid->isChecked()){
        for(i=0;i<=img.width();i+=k){
            for(j=0;j<=img.height();j++)
                img.setPixel(j,i,qRgb(255,0,0));
        }
        for(i=0;i<=img.height();i+=k){
            for(j=0;j<=img.width();j++)
                img.setPixel(i,j,qRgb(255,0,0));
        }
        ui->frame->setPixmap(QPixmap::fromImage(img));
    }
    else{
        for(i=0;i<=img.width();i++){
            for(j=0;j<=img.height();j++)
                img.setPixel(j,i,qRgb(0,0,0));
        }
        for(i=0;i<=img.height();i++){
            for(j=0;j<=img.width();j++)
                img.setPixel(i,j,qRgb(0,0,0));
        }
        ui->frame->setPixmap(QPixmap::fromImage(img));
    }

    on_show_axes_clicked();
}
```

Line Drawing:

DDA Line Drawing:

```
void MainWindow::drawDDA(QPoint p1, QPoint p2){
    int k=ui->grid_size->value();
    int x1=(p1.x()/k);
    int y1=(p1.y()/k);
    int x2=(p2.x()/k);
    int y2=(p2.y()/k);
    int dx=x2-x1;
    int dy=y2-y1;
    int st=(int)((fabs(dx)>fabs(dy))?fabs(dx):fabs(dy));
    float xi=((float)dx)/st;
    float yi=((float)dy)/st;

    float x=x1*k+k/2,y=y1*k+k/2;
    for(int i=1;i<=st;i++){
        x+=xi*k;
        y+=yi*k;
        point((int)(x+0.5),(int)(y+0.5));
    }
}
```

Bresenham Line Drawing:

```
void MainWindow::BresGentle(int x1,int y1, int x2, int y2){
    int k=ui->grid_size->value();
    int dx=(x2-x1)/k;
    int dy=(y2-y1)/k;
    int yi =k;
    if(dy<0){
        yi=-k;
        dy=-dy;
    }
    int dif=2*dy-dx;
    int i,j;
    j=(y1/k)*k+k/2;
    int xs=(x1/k)*k+k/2;
    int xd=(x2/k)*k+k/2;
    for(i=xs;i<=xd;i+=k){
        point(i,j);
        if (dif>0){
            j+=yi;
            dif-=2*dx;
        }
        dif+=2*dy;
    }
}

void MainWindow::BresSteep(int x1,int y1, int x2, int y2){
    int k=ui->grid_size->value();
    int dx=(x2-x1)/k;
    int dy=(y2-y1)/k;
    int xi =k;
    if(dx<0){
        xi=-k;
        dx=-dx;
    }
    int dif=2*dx-dy;
    int i,j;
    i=(x1/k)*k+k/2;
    int ys=(y1/k)*k+k/2;
    int yd=(y2/k)*k+k/2;
    for(j=ys;j<=yd;j+=k){
        point(i,j);
        if (dif>0){
            i+=xi;
            dif-=2*dy;
        }
        dif+=2*dx;
    }
}

void MainWindow::drawBresenham(QPoint p1, QPoint p2){

    if(fabs(p2.y()-p1.y())<fabs(p2.x()-p1.x())){
        if(p1.x()>p2.x())
            BresGentle(p2.x(),p2.y(),p1.x(),p1.y());
        else
            BresGentle(p1.x(),p1.y(),p2.x(),p2.y());
    }
    else{
        if(p1.y()>p2.y())
            BresSteep(p2.x(),p2.y(),p1.x(),p1.y());
        else
            BresSteep(p1.x(),p1.y(),p2.x(),p2.y());
    }
}
```

Circle Drawing:

Bresenham Circle Drawing:

```
void MainWindow::drawCirBres(QPoint p, int r){
    int xc=p.x(),yc=p.y();
    int i,k=ui->grid_size->value(),x=0,y=r*k,d=(3-2*r)*k;
    int dirx[]={+1,+1,-1,-1};
    int diry[]={+1,-1,+1,-1};
    xc=(xc/k)*k+k/2;
    yc=(yc/k)*k+k/2;
    while(y>=x){
        for(i=0;i<4;i++)
            point(xc+dirx[i]*x,yc+diry[i]*y);
        for(i=0;i<4;i++)
            point(xc+dirx[i]*y,yc+diry[i]*x);
        x++;
        if(d>0){
            y--;
            d+=4*(x-y)+10;
        }
        else
            d+=4*x+6;
    }
}
```

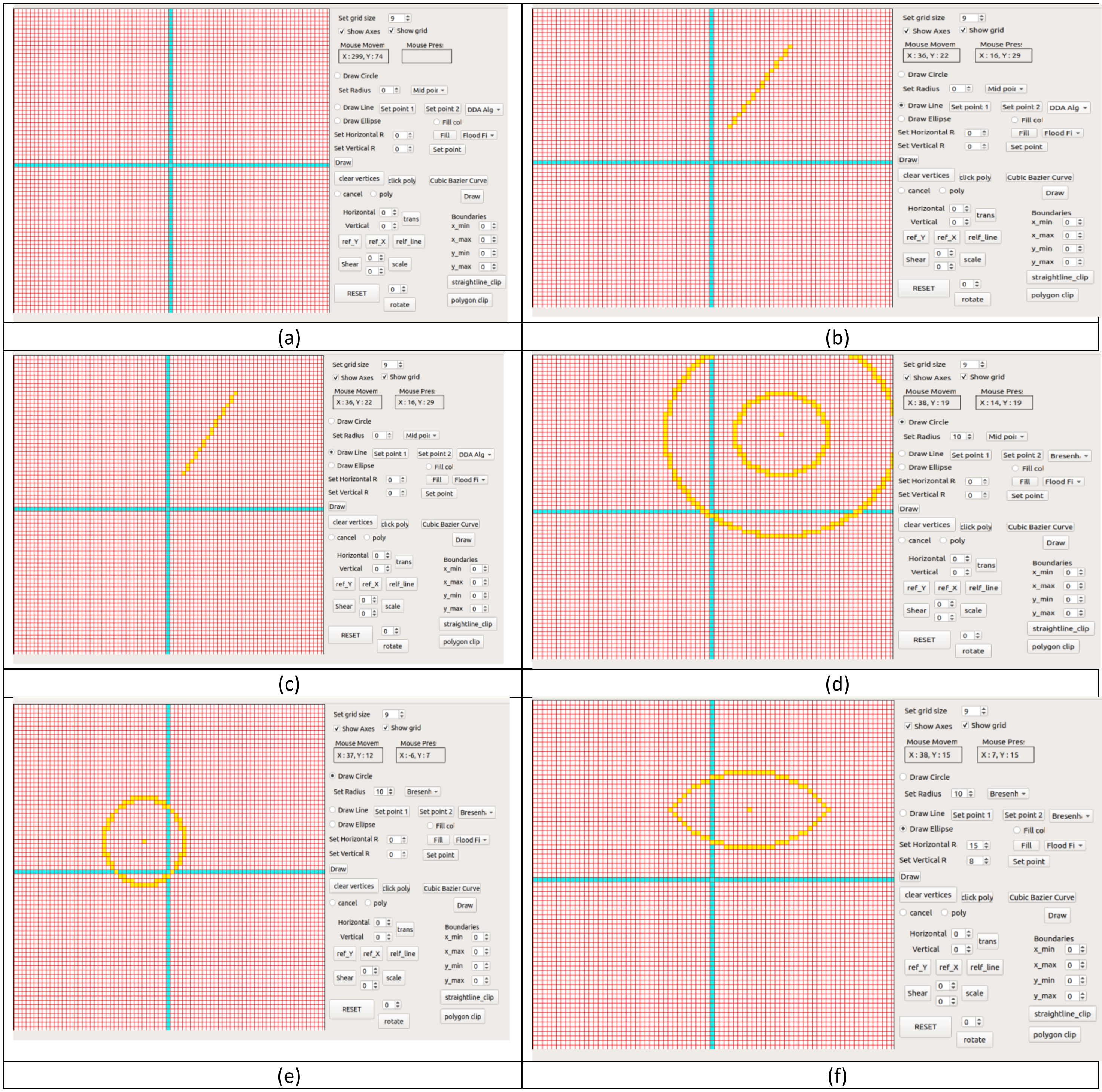
Mid-Point Circle Drawing:

```
void MainWindow::drawCirMidPt(QPoint p1, int r){
    int k=ui->grid_size->value();
    int xc=(p1.x()/k)*k+k/2,yc=(p1.y()/k)*k+k/2;
    int x=r*k,y=0;
    point(xc+x,yc+y);
    if(r>0){
        point(xc+x,yc-y);
        point(xc-x,yc+y);
        point(xc-x,yc-y);
    }
    int p=(1-r)*k;
    while(x>y){
        y++;
        if(p<=0)
            p+=2*y+1;
        else{
            x--;
            p+=2*y-2*x+1;
        }
        if(x<y)
            break;
        point(xc+x,yc+y);
        point(xc+x,yc-y);
        point(xc-x,yc+y);
        point(xc-x,yc-y);
        if(x!=y){
            point(xc+y,yc+x);
            point(xc+y,yc-x);
            point(xc-y,yc+x);
            point(xc-y,yc-x);
        }
    }

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

void MainWindow::drawLine(QPoint p1, QPoint p2, int i){
    if(i)
       drawBresenham(p1,p2);
    else
        drawDDA(p1,p2);
}

void MainWindow::drawCircle(QPoint p, int r, int i){
    if(i){
        drawCirBres(p,r);
    }
    else
        drawCirMidPt(p,r);
}
```

Ellipse Drawing:

```
void MainWindow::drawEllipse(QPoint p1,int a,int b){
    int k=ui->grid_size->value();
    int xc=(p1.x()/k)*k+k/2,yc=(p1.y()/k)*k+k/2;
    int x=0,y=b,xk,yk;
    int a2=a*a,b2=b*b,ta2=2*a2,tb2=2*b2;

    int px=0,py=ta2*y;

    double p=b2-a2*b+a2/4;
    while(px<py){
        xk=x*k;
        yk=y*k;
        point(xc+xk,yc+yk);
        point(xc+xk,yc-yk);
        point(xc-xk,yc+yk);
        point(xc-xk,yc-yk);
        x++;
        px+=tb2;
        if(p<0){
            p+=b2+px;
        }
        else{
            y--;
            py-=ta2;
            p+=b2+px-py;
        }
    }

    p=b2*((double)x+0.5)*((double)x+0.5)+a2*(y-1)*(y-1)-a2*b2;
    while(y>=0){
        xk=x*k;
        yk=y*k;
        point(xc+xk,yc+yk);
        point(xc+xk,yc-yk);
        point(xc-xk,yc+yk);
        point(xc-xk,yc-yk);
        y--;
        py-=ta2;
        if(p>0){
            p+=a2-py;
        }
        else{
            x++;
            px+=tb2;
            p-=a2-py+px;
        }
    }
}
```

The output of the above codes are shown in Figure 1.



(a)

(b)

(c)

(d)

(e)

(f)

Figure 1: (a) The grid (b) DDA Line (c) Bresenham Line (d) Mid-Point Circle (e) Bresenham Circle (f) Ellipse

Filling Methods:

```
void MainWindow::on_set_point_3_clicked()
{
    if(ui->fill_color->isChecked()){
        p1.setX(ui->frame->x);
        p1.setY(ui->frame->y);
    }
}

void MainWindow::on_Fill_clicked()
{
    int i=ui->comboBox_4->currentIndex()+1;
    switch(i){
        case 1:
            floodFill(p1);
        break;
        case 2:
            boundaryFill(p1);
        break;
        /*case 3:
            scanLineFill(p1);
        break;*/
    }
    ui->frame->setPixmap(QPixmap::fromImage(img));
}
```

Flood Fill:

```
void MainWindow::floodFill(QPoint p){


    int k=ui->grid_size->value();
    int xc=(p.x()/k)*k+k/2;
    int yc=(p.y()/k)*k+k/2;
    point(xc,yc,255,255,255);
    floodFillRec(xc,yc,k,img.pixel(xc,yc),135,241,112);
}

void MainWindow::floodFillRec(int x, int y, int k, QRgb q, int r, int g, int b){
    if(x<0 || x>img.width() || y<0 || y>img.height() || img.pixel(x,y)!=q)
        return;
    point(x,y,r,g,b);
    floodFillRec(x+k,y,k,q,r,g,b);
    floodFillRec(x,y+k,k,q,r,g,b);
    floodFillRec(x-k,y,k,q,r,g,b);
    floodFillRec(x,y-k,k,q,r,g,b);
}
```

Boundary Fill:

```
void MainWindow::boundaryFill(QPoint p){
    int k=ui->grid_size->value();
    int xc=(p.x()/k)*k+k/2;
    int yc=(p.y()/k)*k+k/2;
    point(xc,yc,255,255,255);
    boundaryFillRec(xc,yc,k,255,255,0,135,241,112);
}
```

```

void MainWindow::boundaryFillRec(int x, int y, int k, int rb, int gb, int bb, int rf, int gf, int bf){
    QColor cur(img.pixel(x,y));
    if(x<0 || x>img.width() || y<0 || y>img.height() || (cur.red()==rb && cur.green()==gb && cur.blue()==bb) || (cur.red()==rf && cur.green()==gf && cur.blue()==bf))
        return;
    point(x,y,rf,gf,bf);
    boundaryFillRec(x+k,y,k,rb,gb,bb,rf,gf,bf);
    boundaryFillRec(x,y+k,k,rb,gb,bb,rf,gf,bf);
    boundaryFillRec(x-k,y,k,rb,gb,bb,rf,gf,bf);
    boundaryFillRec(x,y-k,k,rb,gb,bb,rf,gf,bf);
}
```

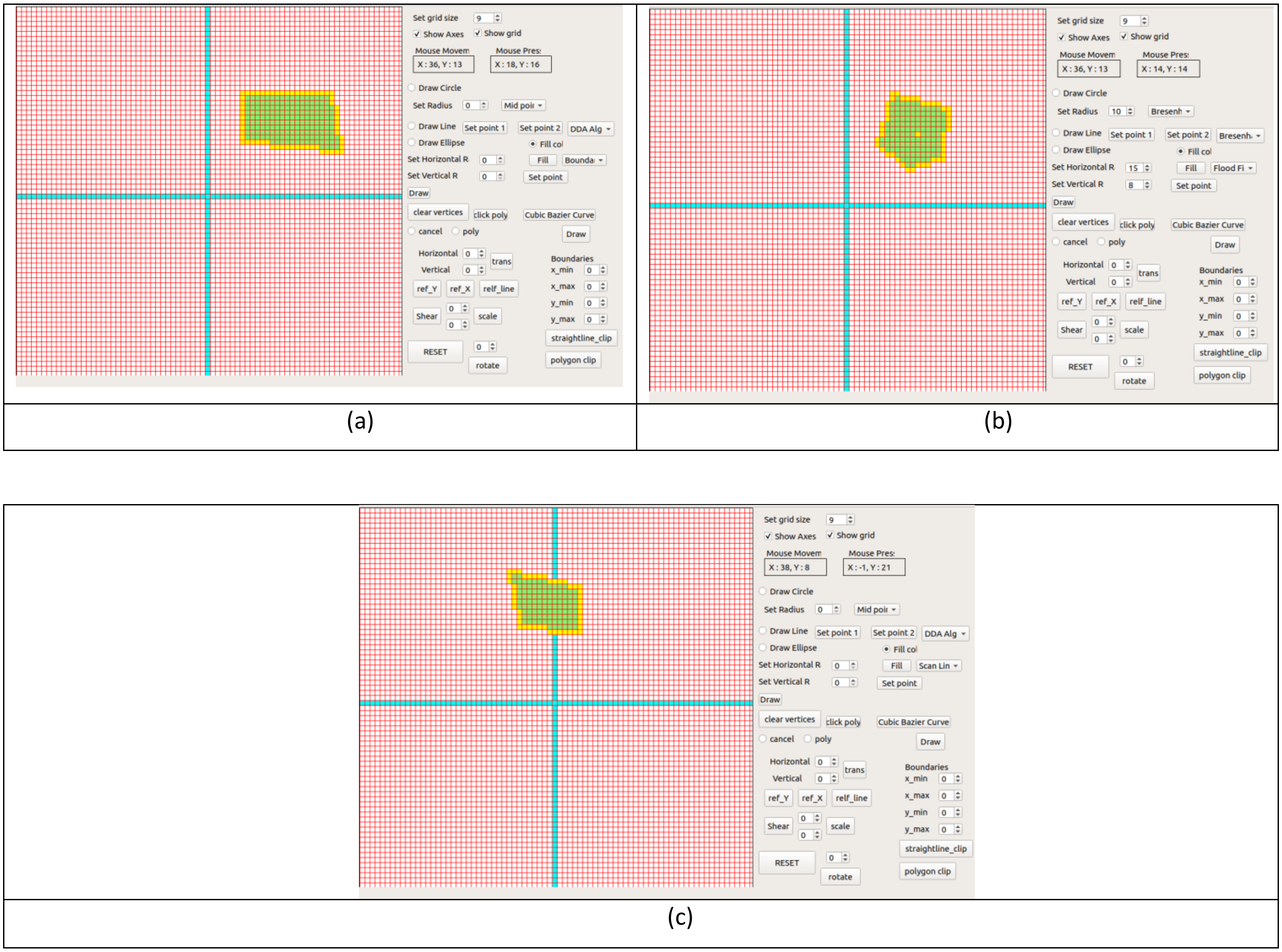
Scanline Fill:

```
#define maxHt 1800
#define maxWd 1000
#define maxVer 10000

typedef struct edgebucket
{
    int ymax;   //max y-coordinate of edge
    float xofymin;  //x-coordinate of lowest edge point updated only in aet
    float slopeinverse;
}EdgeBucket;

typedef struct edgetabletup
{
    // the array will give the scanline number
    // The edge table (ET) with edges entries sorted
    // in increasing y and x of the lower end
    int countEdgeBucket;    //no. of edgebuckets
    EdgeBucket buckets[maxVer];
}EdgeTableTuple;

EdgeTableTuple EdgeTable[maxHt], ActiveEdgeTuple;

// Scanline Function
void MainWindow::initEdgeTable()
{
    int i;
    for (i=0; i<maxHt; i++)
    {
        EdgeTable[i].countEdgeBucket = 0;
    }

    ActiveEdgeTuple.countEdgeBucket = 0;
}

void 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 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 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 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 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::scanLineFill()
{

        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)
                    {
                            QPoint px,py;
                            px.setX(x1);px.setY(i);
                            py.setX(x2);py.setY(i);
                            drawBresenham(px,py);
                    }
                }
                j++;
            }
            updatexbyslopeinv(&ActiveEdgeTuple);
        }
            vertex_list.clear();
    }
```

Output of filling methods are shown in Figure 2.



Set grid size 9

Mouse Movem X : 36, Y : 13

Mouse Pres: X : 18, Y : 16

Set Radius 0 Mid poir

DDA Alg

Set Horizontal R 0

Set Vertical R 0

Bounda

(a)

Set grid size 9

Mouse Movem X : 36, Y : 13

Mouse Pres: X : 14, Y : 14

Set Radius 10 Bresenh

Bresenh

Set Horizontal R 15

Set Vertical R 8

Flood Fi

(b)

Set grid size 9

Mouse Movem X : 38, Y : 8

Mouse Pres: X : -1, Y : 21

Set Radius 0 Mid poir

DDA Alg

Set Horizontal R 0

Set Vertical R 0

Scan Lin

(c)

Figure 2: (a) Flood Fill (b) Boundary Fill (c) Scanline Fill

Transformation:

Translation:

```
void MainWindow::translate_object(int xx,int yy){

    on_pushButton_clicked();
    on_show_grid_clicked();

    int l = vertices.size();
    int i;

    for(i=0;i<l;i++){
        QPoint p = vertices[i];
        p.setX(p.x()+xx);
        p.setY(p.y()+yy);

    }

    for(i=0;i<l-1;i++){
        drawBresenham(vertices[i], vertices[i+1]);
    }

    drawBresenham(vertices[0], vertices[l-1]);

}

void MainWindow::on_draw_ellipse_2_clicked()
{
    flag = true;
}

void MainWindow::on_Draw_2_clicked()
{
    int l = vertices.size();
    int i;

    for(i=0;i<l-1;i++){
        drawBresenham(vertices[i], vertices[i+1]);
    }

    drawBresenham(vertices[0], vertices[l-1]);
}

void MainWindow::on_Draw_3_clicked()
{
    int x = ui->hor_trans->value();
    int y = ui->Ver_trans->value();
    int k = ui->grid_size->value();
    int xx = x*k;
    int yy = y*k;
    int l = vertices.size();
    int i;


    on_pushButton_clicked();
    on_show_grid_clicked();
    for(i=0;i<l;i++){
        QPoint p = vertices[i];
        p.setX(p.x()+xx);
        p.setY(p.y()-yy);
        vertices[i] = p;

    }

    for(i=0;i<l-1;i++){
        drawBresenham(vertices[i], vertices[i+1]);
    }

    drawBresenham(vertices[0], vertices[l-1]);
}
```

Reflection about Y axis:

```
void MainWindow::on_Draw_4_clicked()
{

    on_pushButton_clicked();
    on_show_grid_clicked();
    int i;
    int wid = img.width();
    //int len = img.height();

    //QVector<QPoint> temp;
    int l = vertices.size();
    for(i=0;i<l;i++){
        QPoint p = vertices[i];
        p.setX(wid - p.x());
        p.setY(p.y());
        //temp.push_back(p);
        vertices[i] = p;

    }

    for(i=0;i<l-1;i++){
        drawBresenham(vertices[i], vertices[i+1]);
    }

    drawBresenham(vertices[0], vertices[l-1]);
}
```

Reflection about X axis:

```
void MainWindow::on_Draw_5_clicked()
{

    on_pushButton_clicked();
    on_show_grid_clicked();
    int i;
    //int wid = img.width();
    int len = img.height();

    //QVector<QPoint> temp;
    int l = vertices.size();
    for(i=0;i<l;i++){
        QPoint p = vertices[i];
        p.setX(p.x());
        p.setY( len - p.y());
      //  temp.push_back(p);
```

```
        vertices[i] = p;

    }

    for(i=0;i<l-1;i++){
        drawBresenham(vertices[i], vertices[i+1]);
    }

    drawBresenham(vertices[0], vertices[l-1]);
}
```

Scaling:

```
void MainWindow::on_Draw_6_clicked()
{
    on_pushButton_clicked();
    on_show_grid_clicked();

    int x = ui->scaleratio->value();
    int y = ui->scaleratio_2->value();
    int i;
    int wid = img.width();
    wid = wid/2;
    int len = img.height();
    len = len/2;

    //QVector<QPoint> temp;
    int l = vertices.size();
    for(i=0;i<l;i++){
        QPoint p = vertices[i];
        p.setX((x*(p.x() - wid)) + wid);
        p.setY((y*(p.y() - len)) + len);
        //temp.push_back(p);
        vertices[i] = p;

    }

    for(i=0;i<l-1;i++){
        drawBresenham(vertices[i], vertices[i+1]);
    }

    drawBresenham(vertices[0], vertices[l-1]);

}
```

Shear:

```
void MainWindow::on_Draw_7_clicked()
{

    on_pushButton_clicked();
    on_show_grid_clicked();

    int x = ui->scaleratio->value();
    int y = ui->scaleratio_2->value();
    int i;
    int wid = img.width();
    wid = wid/2;
    int len = img.height();
    len = len/2;

 //   QVector<QPoint> temp;
    int l = vertices.size();
    for(i=0;i<l;i++){
        QPoint p = vertices[i];

        int xx = p.x() - wid;
        int yy = len - p.y();
        p.setX(xx+yy*x + wid);
        p.setY(2*len - (yy+xx*y + len));
        vertices[i] = p;
       // temp.push_back(p);

    }

    for(i=0;i<l-1;i++){
        drawBresenham(vertices[i], vertices[i+1]);
    }

    drawBresenham(vertices[0], vertices[l-1]);
}
```

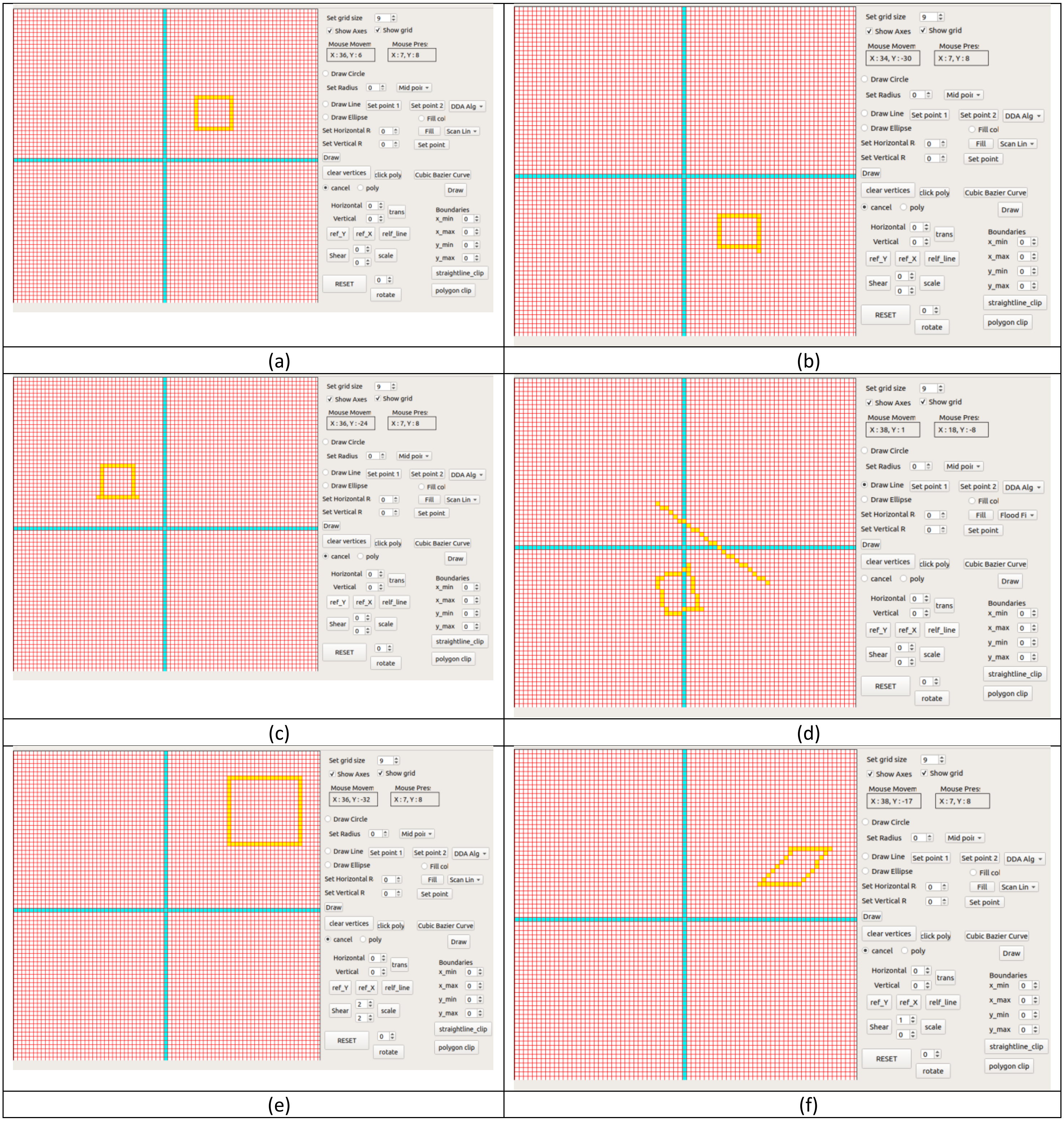
Reflection about a line:

```
void MainWindow::on_Draw_8_clicked()
{

    on_pushButton_clicked();
    on_show_grid_clicked();

    int wid = img.width();
    int len = img.height();
    wid = wid/2;
    len = len/2;

    int x1 = p1.x() - wid;
    int x2 = p2.x() - wid;
    int y1 = len - p1.y();
    int y2 = len - p2.y();

    int a,b,c;

    a = y2-y1;
    b = x1-x2;
    c = x1*(y1-y2) - y1*(x1-x2);

    int i;


    int l = vertices.size();
    for(i=0;i<l;i++){
        QPoint p = vertices[i];

        int x = p.x() - wid;
        int y = len - p.y();
        int xx = (x*(b*b - a*a) - 2*a*(b*y+c))/(a*a + b*b);
        int yy = (y*(a*a - b*b) - 2*b*(a*x+c))/(a*a + b*b);
        p.setX(xx + wid);
        p.setY(len - yy);
        vertices[i] = p;
    }

    for(i=0;i<l-1;i++){
        drawBresenham(vertices[i], vertices[i+1]);
    }

    drawBresenham(vertices[0], vertices[l-1]);
    drawBresenham(p1,p2);
}

void MainWindow::on_draw_ellipse_3_clicked()
{
    flag = false;
   // vertices.clear();
}

void MainWindow::on_Draw_9_clicked()
{
    flag = false;
    vertices.clear();
}
```

Rotation:

```
void MainWindow::on_Draw_10_clicked()
{

    on_pushButton_clicked();
    on_show_grid_clicked();

    int angle = ui->rotate_point->value();
    int x1 = p1.x();
    int y1 = p1.y();

    float rad = (3.14/180.0)*angle;
    float s = sin(rad);
    float c = cos(rad);
    int i;
    int l = vertices.size();
    for(i=0;i<l;i++){
        QPoint p = vertices[i];

        int x = p.x() - x1;
        int y = y1 - p.y();

        int xn = int(x*1.0*c - y*1.0*s);
        int yn = int(y*1.0*c + x*1.0*s);

        p.setX(xn + x1);
        p.setY(y1 - yn);
        vertices[i] = p;
    }
    for(i=0;i<l-1;i++){
        drawBresenham(vertices[i], vertices[i+1]);
    }

    drawBresenham(vertices[0], vertices[l-1]);

}
```

The output for various transformation is given in Figure 3.



(a) Set grid size 9; Mouse Movem X : 36, Y : 6; Mouse Pres X : 7, Y : 8

(b) Set grid size 9; Mouse Movem X : 34, Y : -30; Mouse Pres X : 7, Y : 8

(c) Set grid size 9; Mouse Movem X : 36, Y : -24; Mouse Pres X : 7, Y : 8

(d) Set grid size 9; Mouse Movem X : 38, Y : 1; Mouse Pres X : 18, Y : -8

(e) Set grid size 9; Mouse Movem X : 36, Y : -32; Mouse Pres X : 7, Y : 8; Shear/scale 2, 2

(f) Set grid size 9; Mouse Movem X : 38, Y : -17; Mouse Pres X : 7, Y : 8; Shear 1, 0

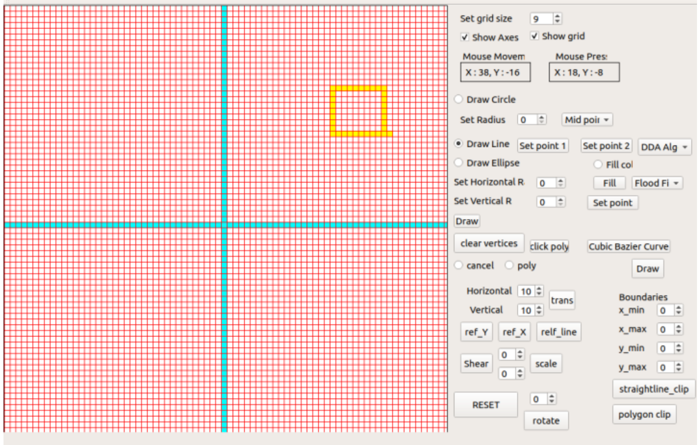
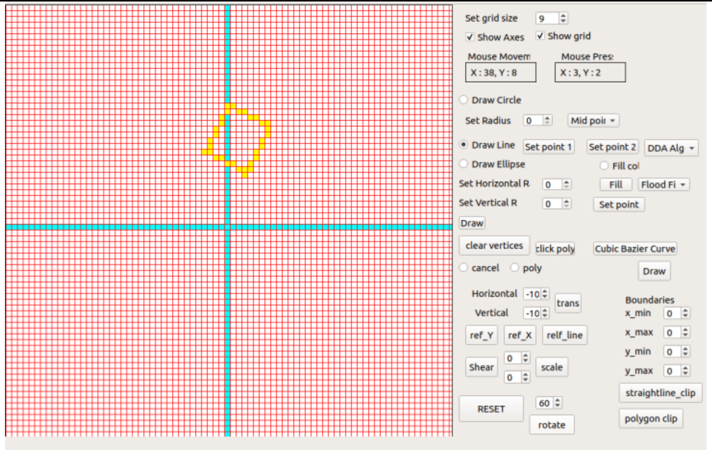
 Set grid size 9 ☑ Show Axes ☑ Show grid Mouse Movem X : 38, Y : -16 Mouse Pres: X : 18, Y : -8 ○ Draw Circle Set Radius 0 Mid poi ● Draw Line ○ Draw Ellipse ○ Fill col Set Horizontal R 0 Set Vertical R 0 ○ cancel ○ poly Horizontal 10 Vertical 10 Shear 0 0 rotate 0 Boundaries x_min 0 x_max 0 y_min 0 y_max 0 Set point 1 Set point 2 DDA Alg Fill Flood Fi Set point Draw clear vertices click poly Cubic Bazier Curve Draw trans ref_Y ref_X relf_line scale RESET straightline_clip polygon clip	 Set grid size 9 ☑ Show Axes ☑ Show grid Mouse Movem X : 38, Y : 8 Mouse Pres: X : 3, Y : 2 ○ Draw Circle Set Radius 0 Mid poi ● Draw Line ○ Draw Ellipse ○ Fill col Set Horizontal R 0 Set Vertical R 0 ○ cancel ○ poly Horizontal -10 Vertical -10 Shear 0 0 rotate 60 Boundaries x_min 0 x_max 0 y_min 0 y_max 0 Set point 1 Set point 2 DDA Alg Fill Flood Fi Set point Draw clear vertices click poly Cubic Bazier Curve Draw trans ref_Y ref_X relf_line scale RESET straightline_clip polygon clip
(g)	(h)

Figure 3: (a) The original drawing (b) Reflection about X axis (c) Reflection about Y axis (d) Reflection about a Line (e) Scaling (f) Shear (g) Translation (h) Rotation.

Clipping:

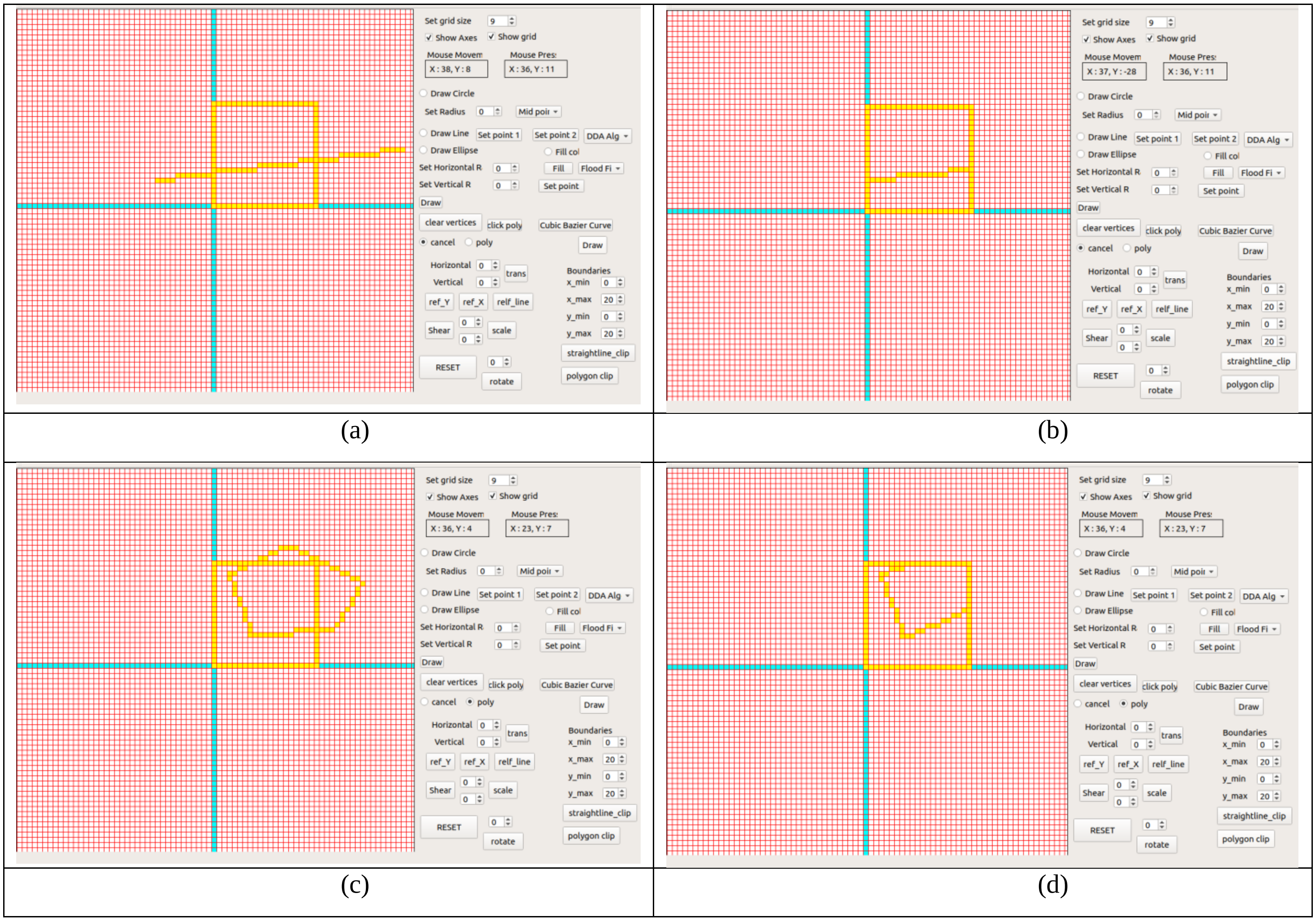
Cohen-Sutherland Line-Clipping:

```
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
int x_max,x_min,y_max,y_min;

void MainWindow::on_Draw_13_clicked()
{
    QPoint px,py;
    int k=ui->grid_size->value();
    x_max=ui->xmax->value()*k+350;
    x_min=ui->xmin->value()*k+350;
    y_max=350-ui->ymax->value()*k;
    y_min=350-ui->ymin->value()*k;
    px.setX(x_max);
    px.setY(y_max);
    py.setX(x_min);
    py.setY(y_max);
    drawBresenham(px,py);
    px.setX(x_min);
    px.setY(y_min);
    drawBresenham(px,py);
    py.setX(x_max);
    py.setY(y_min);
    drawBresenham(px,py);
    px.setX(x_max);
    px.setY(y_max);
    drawBresenham(px,py);
}

int MainWindow::computeCode(double x, double y)
{
    // initialized as being inside
    int code = INSIDE;
    if (x < x_min)       // to the left of rectangle
        code |= LEFT;
    else if (x > x_max)  // to the right of rectangle
        code |= RIGHT;
    if (y > y_min)       // below the rectangle
        code |= BOTTOM;
    else if (y < y_max)  // above the rectangle
        code |= TOP;

    return code;
}

int MainWindow::x_intersect(int x1, int y1, int x2, int y2,int x3, int y3, int x4, int y4)
{
    y1=ui->frame->height()-y1+1;
    y2=ui->frame->height()-y2+1;
    y3=ui->frame->height()-y3+1;
    y4=ui->frame->height()-y4+1;
    int num = (x1*y2 - y1*x2) * (x3-x4) -(x1-x2) * (x3*y4 - y3*x4);
    int den = (x1-x2) * (y3-y4) - (y1-y2) * (x3-x4);
    int retx=num/den;
    return retx;
}
int MainWindow::y_intersect(int x1, int y1, int x2, int y2,int x3, int y3, int x4, int y4)
{
    y1=ui->frame->height()-y1+1;
    y2=ui->frame->height()-y2+1;
    y3=ui->frame->height()-y3+1;
    y4=ui->frame->height()-y4+1;
    int num = (x1*y2 - y1*x2) * (y3-y4) -(y1-y2) * (x3*y4 - y3*x4);
    int den = (x1-x2) * (y3-y4) - (y1-y2) * (x3-x4);
    int rety= (ui->frame->height()-num/den+1);
    return rety;
}

void MainWindow::on_Draw_11_clicked()
{
    double x1=vertices[0].x();
    double y1=vertices[0].y();
    double x2=vertices[1].x();
    double y2=vertices[1].y();
    int code1=computeCode(x1,y1);
    int code2=computeCode(x2,y2);
    bool accept = false;
    while (true)
    {
        if ((code1 == 0) && (code2 == 0))
        {
            accept = true;
            break;
        }
        else if (code1 & code2)
        {
            break;
        }
        else
        {
            int code_out;
            int x, y;
            if (code1 != 0)
                code_out = code1;
            else
                code_out = code2;
            if (code_out & TOP)
                   {
                       x = x1 + (int)((double)(x2 - x1) *(double)(y_max - y1) /(double)(y2 - y1));
                       y = y_max;
                   }
                   else if (code_out & BOTTOM)
                   {
                       x = x1 + (int)((double)(x2 - x1) * (double)(y_min - y1) / (double)(y2 - y1));
                       y = y_min;
                   }
                   else if (code_out & RIGHT)
                   {
                       y = y1 + (int)((double)(y2 - y1) * (double)(x_max - x1) / (double)(x2 - x1));
                       x = x_max;
                   }
                   else if (code_out & LEFT)
                   {
                       y = y1 + (int)((double)(y2 - y1) * (double)(x_min - x1) / (double)(x2 - x1));
                       x = x_min;
                   }
                   if (code_out == code1)
                   {
                       x1 = x;
                       y1 = y;
                       code1 = computeCode(x1, y1);
                   }
                   else
                   {
                       x2 = x;
                       y2 = y;
                       code2 = computeCode(x2, y2);
                   }
               }
           }
           on_pushButton_clicked();
           on_show_grid_clicked();
           on_Draw_13_clicked();

           if (accept)
           {
               QPoint px,py;
               px.setX((int)x1);
               px.setY((int)y1);
               py.setX((int)x2);
               py.setY((int)y2);
               drawBresenham(px,py);
            }

        }
```

Sutherland-Hodgman's polygon-clipping

```
void MainWindow::clip(int x1, int y1, int x2, int y2)
{
    int poly_size=vertices.size()-1;
    int new_poly_size = 0;
    std::vector<std::pair<int,int> > new_points;
    for (int i = 0; i < poly_size; i++)
    {
        int k = (i+1) % poly_size;
        int ix = vertices[i].x(), iy = vertices[i].y();
        int kx = vertices[k].x(), ky = vertices[k].y();
        int i_pos,k_pos;
        if(x2==x1 && ix>x1) i_pos=1;
        else if(x2==x1 && ix<x1) i_pos=-1;
        else if(y2==y1 && iy<y1) i_pos=1;
        else i_pos=-1;
        if(x2==x1 && kx>x1) k_pos=1;
        else if(x2==x1 && kx<x1) k_pos=-1;
        else if(y2==y1 && ky<y1) k_pos=1;
        else k_pos=-1;
        if(y1>y2||x1>x2)
        {
            i_pos=(-1)*i_pos;
            k_pos=(-1)*k_pos;
        }
        if (i_pos >= 0  && k_pos >= 0)
        {
            new_points.push_back(std::make_pair(kx,ky));
            new_poly_size++;
        }
        else if (i_pos < 0  && k_pos >= 0)
        {
            new_points.push_back(std::make_pair(x_intersect(x1,y1, x2, y2, ix, iy, kx, ky),
                                                y_intersect(x1,y1, x2, y2, ix, iy, kx, ky)));
            new_poly_size++;
            new_points.push_back(std::make_pair(kx,ky));
            new_poly_size++;
        }
        else if (i_pos >= 0  && k_pos < 0)
        {

            new_points.push_back(std::make_pair(x_intersect(x1,y1, x2, y2, ix, iy, kx, ky),
                                                y_intersect(x1,y1, x2, y2, ix, iy, kx, ky)));
            new_poly_size++;
        }
        else
        {
        }
    }
    poly_size = new_poly_size;
    vertices.clear();
    for (int i = 0; i < new_points.size(); i++)
    {
        QPoint p;
        p.setX(new_points[i].first);
        p.setY(new_points[i].second);
        vertices.push_back(p);
    }
    if(poly_size>0){
        QPoint pp;
        pp.setX(new_points[0].first);
        pp.setY(new_points[0].second);
        vertices.push_back(pp);
    }
}
void MainWindow::suthHodgClip()
{

    clip(x_min,y_max,x_min,y_min); //Left
    if(vertices.size()>0)
        clip(x_min,y_min,x_max,y_min); //Bottom
    if(vertices.size()>1)
        clip(x_max,y_min,x_max,y_max); //Right
    if(vertices.size()>1)
        clip(x_max,y_max,x_min,y_max); //Top
    on_pushButton_clicked();
    on_show_grid_clicked();
    on_Draw_13_clicked();
    int l=vertices.size();
    QPoint px,py;
    int k;
    if(l>1){
       for(int i=0;i<l-1;i++){
           k=(i+1)%l;
           px.setX(vertices[i].x());
           px.setY(vertices[i].y());
           py.setX(vertices[k].x());
           py.setY(vertices[k].y());
           drawBresenham(px,py);
       }

    }

}

void MainWindow::on_Draw_12_clicked()
{
    suthHodgClip();
}
```

Output for the above two clipping algorithm is shown in Figure 4.



(a)

Set grid size 9 ☑ Show Axes ☑ Show grid

Mouse Movem X : 38, Y : 8 Mouse Pres: X : 36, Y : 11

● cancel ○ poly

Boundaries x_min 0 x_max 20 y_min 0 y_max 20

(b)

Set grid size 9 ☑ Show Axes ☑ Show grid

Mouse Movem X : 37, Y : -28 Mouse Pres: X : 36, Y : 11

● cancel ○ poly

Boundaries x_min 0 x_max 20 y_min 0 y_max 20

(c)

Set grid size 9 ☑ Show Axes ☑ Show grid

Mouse Movem X : 36, Y : 4 Mouse Pres: X : 23, Y : 7

○ cancel ● poly

Boundaries x_min 0 x_max 20 y_min 0 y_max 20

(d)

Set grid size 9 ☑ Show Axes ☑ Show grid

Mouse Movem X : 36, Y : 4 Mouse Pres: X : 23, Y : 7

○ cancel ● poly

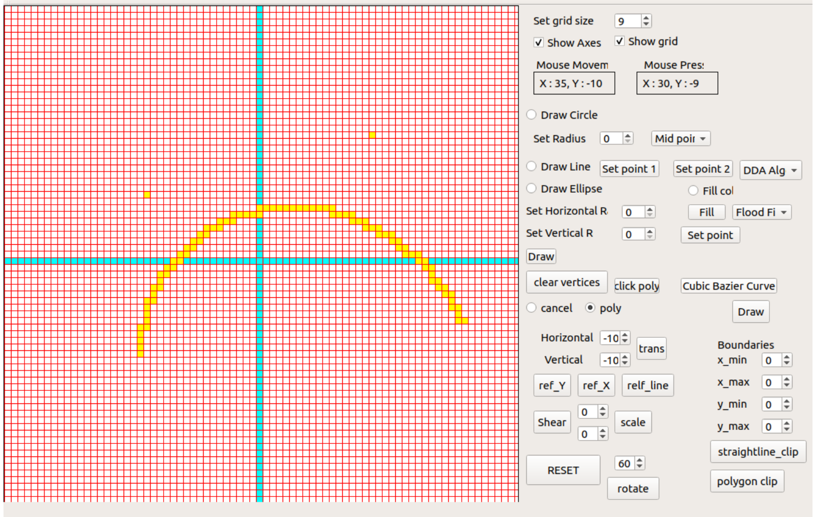
Boundaries x_min 0 x_max 20 y_min 0 y_max 20

Figure 4: (a) The straight line and the clipping window (b) The straight line after clipping (c) The polygon and the clipping window (d) The polygon after clipping

Bezier Curve:

```
void MainWindow::on_Fill_2_clicked()
{
    double xu = 0.0 , yu = 0.0 , u = 0.0 ;
    int i = 0 ;
    for(u = 0.0 ; u <= 1.0 ; u += 0.0001)
    {
        xu = pow(1-u,3)*vertices[0].x()+3*u*pow(1-u,2)*vertices[1].x()+3*pow(u,2)*(1-u)*vertices[2].x()+pow(u,3)*vertices[3].x();
        yu = pow(1-u,3)*vertices[0].y()+3*u*pow(1-u,2)*vertices[1].y()+3*pow(u,2)*(1-u)*vertices[2].y()+pow(u,3)*vertices[3].y();
        point((int)xu , (int)yu);
    }
}
```

Output for Bazier curve is shown in Figure 5.



Set grid size 9

☑ Show Axes ☑ Show grid

Mouse Movem X : 35, Y : -10

Mouse Pres: X : 30, Y : -9

○ Draw Circle

Set Radius 0

○ Draw Line

○ Draw Ellipse

○ Fill col

Set Horizontal R 0

Set Vertical R 0

○ cancel ● poly

Horizontal -10

Vertical -10

Boundaries

x_min 0

x_max 0

y_min 0

y_max 0

Shear 0 0

60

(a)

Figure 5: (a) Bazier curve