

# COMPUTER GRAPHICS LAB REPORT

SOULIB GHOSH and MADHURAM JAJOO



# JADAVPUR UNIVERSITY

NAME – SOULIB GHOSH

ROLL – 0016105010 47

COURSE – BCSE III

GROUP – A2

NAME – MADHURAM JAJOO ROLL – 0016105010 54 COURSE – BCSE III 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

```
1) #ifndef MY LABEL H
2) #define MY_LABEL_H
3)
4) #include <QObject>
5) #include <QWidget>
6) #include <QLabel>
7) #include <QMouseEvent>
8) #include <QEvent>
9)
10) class my_label : public QLabel
11) {
12)
        Q_OBJECT
13)
14) public:
        my label(QWidget *parent = 0);
15)
       int x,y;
16)
17)
18) protected:
       void mouseMoveEvent(OMouseEvent *ev);
       void mousePressEvent(QMouseEvent *ev);
20)
21) signals:
22) void sendMousePosition(QPoint&);
23)
        void Mouse_Pos();
24) };
25)
26) #endif // MY_LABEL_H
```

#### my\_label.cpp

```
1. #include "my_label.h"
2.
3. my_label::my_label(QWidget *parent):QLabel(parent)
4. {
5.
       this->setMouseTracking(true);
6. }
8. void my_label::mouseMoveEvent(QMouseEvent *ev)
9. {
10.
       QPoint mouse_pos=ev->pos();
11.
       if(mouse_pos.x()<=this->size().width() && mouse_pos.y()<=this->size().height()){
12.
           if(mouse_pos.x()>0 && mouse_pos.y()>0){
13.
               emit sendMousePosition(mouse_pos);
14.
15.
       }
16.}
17.
18. void my_label::mousePressEvent(QMouseEvent *ev)
19. {
20.
       if(ev->button()==Qt::LeftButton){
21.
           this->x=ev->x();
22.
           this->y=ev->y();
```

```
23. emit Mouse_Pos();
24. }
25.}
```

#### mainwindow.h

```
1. #ifndef MAINWINDOW H
2. #define MAINWINDOW H
3.
4. #include <QMainWindow>
5. #include <QtGui>
6. #include <QtCore>
7. #include <QColorDialog>
8. #include<QVector>
9.
10. namespace Ui {
11. class MainWindow;
12. }
13.
14. class MainWindow : public QMainWindow
15. {
       Q OBJECT
16.
17. public slots:
       void Mouse_Pressed();
18.
19.
       void showMousePosition(QPoint& pos);
20. public:
21.
       explicit MainWindow(QWidget *parent = 0);
22.
       ~MainWindow();
23.
24. private slots:
25.
       void on_show_axes_clicked();
26.
       void on_Draw_clicked();
27.
       void on_set_point1_clicked();
28.
       void on_set_point2_clicked();
29.
       void on_pushButton_clicked();
30.
       int changeX(int x);
31.
       int changeY(int y);
32.
       void on_show_grid_clicked();
33.
       void drawLine(QPoint, QPoint, int);
34.
       void drawCircle(QPoint, int, int);
35.
       void drawEllipse(QPoint,int,int);
36.
       void on_set_point_3_clicked();
37.
       void on_Fill_clicked();
38.
       void translate_object(int x,int y);
39.
       void on_draw_ellipse_2_clicked();
40.
       void on_Draw_2_clicked();
41.
       void on_Draw_3_clicked();
       void on Draw 4 clicked();
42.
       void on_Draw_5_clicked();
43.
       void on_Draw_6_clicked();
44.
       void on_Draw_7_clicked();
45.
46.
       void on_Draw_8_clicked();
       void on_draw_ellipse_3_clicked();
47.
48.
       void on_Draw_9_clicked();
49.
       void on_comboBox_3_activated(const QString &arg1);
50.
       void on_Draw_10_clicked();
       void on_Draw_11_clicked();
51.
52.
       void on Fill 2 clicked();
53.
       void on Draw 13 clicked();
54. void on_Draw_12_clicked();
```

```
55.
56. private:
57.
       bool flag = false;
58.
       QVector<QPoint> vertices;
59.
       Ui::MainWindow *ui;
60.
       QPoint p1,p2;
       void point(int,int);
61.
       void point(int ,int , int , int, int);
62.
       void drawDDA(QPoint, QPoint);
63.
64.
       void drawBresenham(QPoint, QPoint);
       void BresGentle(int,int,int,int);
65.
       void BresSteep(int,int,int,int);
66.
       void drawCirBres(QPoint, int);
67.
68.
       void drawCirMidPt(QPoint, int);
69.
       void floodFill(QPoint);
       void floodFillRec(int, int, int, QRgb, int, int, int);
70.
71.
       void boundaryFill(QPoint);
72.
       73.
       void scanLineFill(QPoint);
74.
       int computeCode(double,double);
75.
       int x intersect(int,int,int,int,int,int,int);
76.
       int y_intersect(int,int,int,int,int,int,int);
77.
       void suthHodgClip();
78.
       void clip(int,int,int,int);
79.
80.};
81. #endif // MAINWINDOW H
```

#### main.cpp

```
1. #include "mainwindow.h"
2. #include <QApplication>
3.
4. int main(int argc, char *argv[])
5. {
6.     QApplication a(argc, argv);
7.     MainWindow w;
8.     w.show();
9.
10.     return a.exec();
11. }
```

#### mainwindow.cpp

```
    #include "mainwindow.h"

2. #include "ui mainwindow.h"
3. #include <QPixmap>
4. #include <QImage>
5. #include <iostream>
6. #include <QMouseEvent>
7. #include <QPainter>
8. #include <QPaintDevice>
9. #include <QPoint>
10. QImage img=QImage(700,700,QImage::Format_RGB888);
11. MainWindow::MainWindow(QWidget *parent) :
12.
        QMainWindow(parent),
13.
       ui(new Ui::MainWindow)
14. {
15.
       ui->setupUi(this);
```

```
ui->x_axis->hide();
17.
        ui->y_axis->hide();
        connect(ui->frame,SIGNAL(Mouse_Pos()),this,SLOT(Mouse_Pressed()));
18.
19.
        connect(ui-
    >frame,SIGNAL(sendMousePosition(QPoint&)),this,SLOT(showMousePosition(QPoint&)));
        ui->comboBox->addItem("DDA Algorithm");
20.
        ui->comboBox->addItem("Bresenham Algorithm");
21.
22. }
23.
24. MainWindow::~MainWindow()
25. {
26.
        delete ui;
27. }
28.
29. int MainWindow::changeX(int x){
        int k=ui->grid size->value();
31.
        x-=img.width()/2;
32.
        x/=k;
33.
        return x;
34. }
35.
36. int MainWindow::changeY(int y){
        int k=ui->grid_size->value();
37.
        y=img.width()/2-y;
38.
39.
        y/=k;
40.
        return y;
41. }
42.
43. void MainWindow::point(int x,int y)
44. {
45.
        int k=ui->grid_size->value();
46.
        if(k==1)
47.
            img.setPixel(x,y,qRgb(255,255,0));
48.
        else{
            int i,j;
49.
50.
            int stx=(x/k)*k;
51.
            int sty=(y/k)*k;
52.
            for(i=stx+1;i<stx+k;i++){</pre>
53.
                for(j=sty+1;j<sty+k;j++)</pre>
54.
                    img.setPixel(i,j,qRgb(255,255,0));
55.
            }
56.
57.
        ui->frame->setPixmap(QPixmap::fromImage(img));
58.}
59.
60. void MainWindow::point(int x,int y, int r, int g, int b)
61. {
62.
        int k=ui->grid_size->value();
63.
        if(k==1)
64.
            img.setPixel(x,y,qRgb(r,g,b));
65.
        else{
66.
            int i,j;
67.
            int stx=(x/k)*k;
68.
            int sty=(y/k)*k;
69.
            for(i=stx+1;i<stx+k;i++){</pre>
70.
                for(j=sty+1;j<sty+k;j++)</pre>
71.
                    img.setPixel(i,j,qRgb(r,g,b));
72.
73.
74.
       //ui->frame->setPixmap(QPixmap::fromImage(img));
75.}
```

```
76.
77. void MainWindow::showMousePosition(QPoint &pos)
78. {
79.
        ui->mouse_movement-
    >setText(" X : "+QString::number(changeX(pos.x()))+", Y : "+QString::number(changeY(pos
    .y())));
80.}
81. void MainWindow::Mouse Pressed()
82. {
83.
        ui->mouse pressed->setText(" X : "+QString::number(changeX(ui->frame-
    >x))+", Y: "+QString::number(changeY(ui->frame->y)));
84.
        point(ui->frame->x,ui->frame->y);
85.
        ui->x_axis->move(0,ui->frame->y);
86.
        ui->y_axis->move(ui->frame->x,0);
87.
        //point(ui->frame->x, ui->frame->y,255,255,0);
88.
89.
        if(flag){
            p1.setX((ui->frame->x));
90.
91.
            p1.setY((ui->frame->y));
92.
93.
            if(vertices.size() > 0 && p1 == vertices[0]){
94.
95.
                 flag = false;
96.
                 return;
97.
98.
99.
100.
                   vertices.push_back(p1);
101.
                }
102.
103.
104.
            void MainWindow::on show axes clicked()
105.
            {
                if(ui->show axes->isChecked())
106.
107.
108.
                    for(int i=0;i<img.height();i++)</pre>
109.
                        point(i,img.width()/2,0,255,255);
110.
                    for(int j=0;j<img.width();j++)</pre>
111.
                        point(img.height()/2,j,0,255,255);
112.
                }
113.
                else{
114.
                    for(int i=0;i<img.height();i++)</pre>
115.
                        point(i,img.width()/2,0,0,0);
116.
                    for(int j=0;j<img.width();j++)</pre>
117.
                        point(img.height()/2,j,0,0,0);
118.
119.
            }
120.
           void MainWindow::on set point1 clicked()
121.
            {
122.
                if(ui->draw line->isChecked()){
123.
                    p1.setX(ui->frame->x);
124.
                    p1.setY(ui->frame->y);
125.
                }
126.
127.
128.
           void MainWindow::on_set_point2_clicked()
129.
130.
                if(ui->draw line->isChecked()){
131.
                    p2.setX(ui->frame->x);
132.
                    p2.setY(ui->frame->y);
133.
                }
```

```
134.
135.
136.
            void MainWindow::on_Draw_clicked()
137.
138.
                //int r0=ui->circle_radius->value();
139.
                //QPainter painter(&img);
140.
                //QPen pen;
141.
                //pen.setWidth(1);
142.
                //pen.setColor(Qt::red);
143.
                if(ui->draw circle->isChecked()){
144.
                    p1.setX(ui->frame->x);
145.
                    p1.setY(ui->frame->y);
146.
                    /*painter.setPen(pen);
147.
                    painter.drawEllipse(p1,r0,r0);*/
148.
                    drawCircle(p1,ui->circle_radius->value(),ui->comboBox_2-
    >currentIndex());
149.
150.
                if(ui->draw_line->isChecked()){
151.
                    //painter.setPen(Qt::red);
152.
                    drawLine(p1,p2,ui->comboBox->currentIndex());
153.
154.
                if(ui->draw ellipse->isChecked()){
155.
                    p1.setX(ui->frame->x);
156.
                    p1.setY(ui->frame->y);
157.
                    drawEllipse(p1,ui->ellipse_maj->value(),ui->ellipse_min->value());
158.
159.
160.
161.
                //ui->frame->setPixmap(QPixmap::fromImage(img));
162.
163.
164.
            void MainWindow::on pushButton clicked()
165.
            {
166.
                for(int j=0;j<img.height();j++)</pre>
167.
                    for(int i=0;i<img.width();i++)</pre>
168.
169.
170.
                         img.setPixel(i,j,qRgb(255,255,255));
171.
                    }
172.
173.
                ui->frame->setPixmap(QPixmap::fromImage(img));
174.
175.
            void MainWindow::on show grid clicked()
176.
177.
178.
                int i,j,k=ui->grid size->value();
179.
                if(ui->show grid->isChecked()){
180.
                    for(i=0;i<=img.width();i+=k){</pre>
                         for(j=0;j<=img.height();j++)</pre>
181.
182.
                             img.setPixel(j,i,qRgb(255,0,0));
183.
184.
                    for(i=0;i<=img.height();i+=k){</pre>
185.
                         for(j=0;j<=img.width();j++)</pre>
186.
                             img.setPixel(i,j,qRgb(255,0,0));
187.
                    ui->frame->setPixmap(QPixmap::fromImage(img));
188.
189.
190.
                else{
191.
                    for(i=0;i<=img.width();i++){</pre>
192.
                         for(j=0;j<=img.height();j++)</pre>
193.
                             img.setPixel(j,i,qRgb(0,0,0));
```

```
194.
                     }
195.
                     for(i=0;i<=img.height();i++){</pre>
196.
                         for(j=0;j<=img.width();j++)</pre>
197.
                              img.setPixel(i,j,qRgb(0,0,0));
198.
199.
                     ui->frame->setPixmap(QPixmap::fromImage(img));
200.
201.
202.
                on_show_axes_clicked();
203.
            }
```

#### **Line Drawing:**

#### **DDA Line Drawing:**

```
1. void MainWindow::drawDDA(QPoint p1, QPoint p2){
2. int k=ui->grid_size->value();
3.
        int x1=(p1.x()/k);
        int y1=(p1.y()/k);
4.
5.
        int x2=(p2.x()/k);
        int y2=(p2.y()/k);
6.
        int dx=x2-x1;
7.
8.
        int dy=y2-y1;
9.
        int st=(int)((fabs(dx)>fabs(dy))?fabs(dx):fabs(dy));
10.
        float xi=((float)dx)/st;
        float yi=((float)dy)/st;
11.
12.
13.
        float x=x1*k+k/2, y=y1*k+k/2;
14.
        for(int i=1;i<=st;i++){</pre>
15.
            x+=xi*k;
            y+=yi*k;
16.
17.
            point((int)(x+0.5),(int)(y+0.5));
18.
       }
19. }
```

#### **Bresenham Line Drawing:**

```
1. void MainWindow::BresGentle(int x1,int y1, int x2, int y2){
2.
        int k=ui->grid size->value();
3.
        int dx=(x2-x1)/k;
4.
        int dy=(y2-y1)/k;
5.
        int yi =k;
6.
        if(dy<0){
7.
            yi=-k;
8.
            dy=-dy;
9.
10.
        int dif=2*dy-dx;
11.
        int i,j;
        j=(y1/k)*k+k/2;
12.
13.
        int xs=(x1/k)*k+k/2;
14.
        int xd=(x2/k)*k+k/2;
15.
        for(i=xs;i<=xd;i+=k){</pre>
16.
            point(i,j);
17.
            if (dif>0){
18.
                 j+=yi;
19.
                 dif-=2*dx;
20.
            dif+=2*dy;
21.
22.
```

```
23.}
24.
25. void MainWindow::BresSteep(int x1,int y1, int x2, int y2){
26.
        int k=ui->grid_size->value();
27.
        int dx=(x2-x1)/k;
28.
        int dy=(y2-y1)/k;
29.
        int xi =k;
30.
        if(dx<0){
31.
            xi=-k;
32.
            dx=-dx;
33.
34.
        int dif=2*dx-dy;
        int i,j;
35.
        i=(x1/k)*k+k/2;
36.
37.
        int ys=(y1/k)*k+k/2;
38.
        int yd=(y2/k)*k+k/2;
39.
        for(j=ys;j<=yd;j+=k){</pre>
40.
            point(i,j);
41.
            if (dif>0){
42.
                i+=xi;
43.
                dif-=2*dy;
44.
45.
            dif+=2*dx;
46.
        }
47.}
48.
49. void MainWindow::drawBresenham(QPoint p1, QPoint p2){
50.
51.
        if(fabs(p2.y()-p1.y())<fabs(p2.x()-p1.x())){</pre>
52.
            if(p1.x()>p2.x())
53.
                BresGentle(p2.x(),p2.y(),p1.x(),p1.y());
54.
            else
                BresGentle(p1.x(),p1.y(),p2.x(),p2.y());
55.
56.
        }
57.
        else{
58.
            if(p1.y()>p2.y())
59.
                BresSteep(p2.x(),p2.y(),p1.x(),p1.y());
60.
61.
                BresSteep(p1.x(),p1.y(),p2.x(),p2.y());
62.
63.}
```

#### **Circle Drawing:**

#### **Bresenham Circle Drawing:**

```
1. void MainWindow::drawCirBres(QPoint p, int r){
2.
        int xc=p.x(),yc=p.y();
3.
        int i,k=ui->grid_size->value(),x=0,y=r*k,d=(3-2*r)*k;
4.
        int dirx[]={+1,+1,-1,-1};
5.
        int diry[]={+1,-1,+1,-1};
        xc=(xc/k)*k+k/2;
6.
7.
        yc=(yc/k)*k+k/2;
8.
        while(y>=x){
9.
            for(i=0;i<4;i++)</pre>
10.
                 point(xc+dirx[i]*x,yc+diry[i]*y);
11.
            for(i=0;i<4;i++)</pre>
12.
                point(xc+dirx[i]*y,yc+diry[i]*x);
13.
            x++;
14.
            if(d>0){
```

#### **Mid-Point Circle Drawing:**

```
1. void MainWindow::drawCirMidPt(QPoint p1, int r){
        int k=ui->grid_size->value();
2.
3.
        int xc=(p1.x()/k)*k+k/2,yc=(p1.y()/k)*k+k/2;
4.
        int x=r*k,y=0;
5.
        point(xc+x,yc+y);
6.
        if(r>0){
            point(xc+x,yc-y);
7.
8.
            point(xc-x,yc+y);
9.
            point(xc-x,yc-y);
10.
11.
        int p=(1-r)*k;
        while(x>y){
12.
13.
            y++;
            if(p<=0)
14.
15.
                p+=2*y+1;
16.
            else{
17.
18.
                p+=2*y-2*x+1;
19.
20.
            if(x<y)</pre>
21.
                break;
22.
            point(xc+x,yc+y);
23.
            point(xc+x,yc-y);
24.
            point(xc-x,yc+y);
25.
            point(xc-x,yc-y);
26.
            if(x!=y){
                point(xc+y,yc+x);
27.
28.
                point(xc+y,yc-x);
29.
                point(xc-y,yc+x);
30.
                point(xc-y,yc-x);
31.
            }
32.
33.
34.
        ui->frame->setPixmap(QPixmap::fromImage(img));
35.}
36.
37. void MainWindow::drawLine(QPoint p1, QPoint p2, int i){
38. if(i)
           drawBresenham(p1,p2);
39.
40.
       else
41.
            drawDDA(p1,p2);
42.}
43.
44. void MainWindow::drawCircle(QPoint p, int r, int i){
45.
        if(i){
46.
            drawCirBres(p,r);
47.
48.
        else
            drawCirMidPt(p,r);
49.
50.}
```

#### **Ellipse Drawing:**

```
1. void MainWindow::drawEllipse(QPoint p1,int a,int b){
       int k=ui->grid_size->value();
2.
3.
        int xc=(p1.x()/k)*k+k/2,yc=(p1.y()/k)*k+k/2;
4.
       int x=0,y=b,xk,yk;
5.
        int a2=a*a,b2=b*b,ta2=2*a2,tb2=2*b2;
6.
7.
        int px=0,py=ta2*y;
8.
9.
        double p=b2-a2*b+a2/4;
10.
       while(px<py){</pre>
11.
            xk=x*k;
            yk=y*k;
12.
13.
            point(xc+xk,yc+yk);
14.
            point(xc+xk,yc-yk);
15.
            point(xc-xk,yc+yk);
16.
            point(xc-xk,yc-yk);
17.
            X++;
18.
            px+=tb2;
19.
            if(p<0){
20.
                p+=b2+px;
21.
            else{
22.
23.
                y--;
24.
                py-=ta2;
25.
                p+=b2+px-py;
26.
27.
        }
28.
29.
        p=b2*((double)x+0.5)*((double)x+0.5)+a2*(y-1)*(y-1)-a2*b2;
30.
        while(y>=0){
31.
            xk=x*k;
            yk=y*k;
32.
33.
            point(xc+xk,yc+yk);
34.
            point(xc+xk,yc-yk);
35.
            point(xc-xk,yc+yk);
36.
            point(xc-xk,yc-yk);
37.
            y--;
38.
            py-=ta2;
39.
            if(p>0){
40.
                p+=a2-py;
41.
            }
42.
            else{
43.
                x++;
44.
                px+=tb2;
45.
                p-=a2-py+px;
46.
47.
        }
48. }
```

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



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()

2. {
3.
        if(ui->fill_color->isChecked()){
4.
            p1.setX(ui->frame->x);
5.
            p1.setY(ui->frame->y);
6.
7. }
8.
9. void MainWindow::on_Fill_clicked()
10. {
        int i=ui->comboBox_4->currentIndex()+1;
11.
       switch(i){
12.
13.
            case 1:
14.
                floodFill(p1);
            break;
15.
16.
            case 2:
17.
                boundaryFill(p1);
            break;
18.
19.
            /*case 3:
20.
               scanLineFill(p1);
21.
            break;*/
22.
23.
        ui->frame->setPixmap(QPixmap::fromImage(img));
24. }
```

#### **Flood Fill:**

```
1. void MainWindow::floodFill(QPoint p){
2.
3.
       int k=ui->grid_size->value();
4.
5.
        int xc=(p.x()/k)*k+k/2;
6.
       int yc=(p.y()/k)*k+k/2;
        point(xc,yc,255,255,255);
7.
8.
       floodFillRec(xc,yc,k,img.pixel(xc,yc),135,241,112);
9. }
10.
11. void MainWindow::floodFillRec(int x, int y, int k, QRgb q, int r, int g, int b){
12.
       if(x<0 || x>img.width() || y<0 || y>img.height() || img.pixel(x,y)!=q)
13.
            return;
14.
       point(x,y,r,g,b);
15.
        floodFillRec(x+k,y,k,q,r,g,b);
16.
       floodFillRec(x,y+k,k,q,r,g,b);
        floodFillRec(x-k,y,k,q,r,g,b);
17.
18.
       floodFillRec(x,y-k,k,q,r,g,b);
19.}
```

#### **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);
    }
```

```
8.
9. void MainWindow::boundaryFillRec(int x, int y, int k, int rb, int gb, int bb, int rf, i
    nt gf, int bf){
10.
        QColor cur(img.pixel(x,y));
11.
        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))
12.
13.
        point(x,y,rf,gf,bf);
14.
        boundaryFillRec(x+k,y,k,rb,gb,bb,rf,gf,bf);
15.
        boundaryFillRec(x,y+k,k,rb,gb,bb,rf,gf,bf);
16.
        boundaryFillRec(x-k,y,k,rb,gb,bb,rf,gf,bf);
17.
        boundaryFillRec(x,y-k,k,rb,gb,bb,rf,gf,bf);
18.}
```

#### **Scanline Fill:**

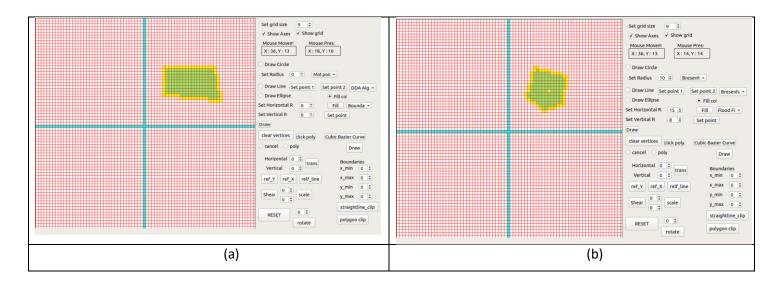
```
1. #define maxHt 1800
2. #define maxWd 1000
3. #define maxVer 10000
4.
5. typedef struct edgebucket
6. {
7.
                    //max y-coordinate of edge
        int ymax;
8.
       float xofymin; //x-coordinate of lowest edge point updated only in aet
9.
        float slopeinverse;
10. }EdgeBucket;
11.
12. typedef struct edgetabletup
13. {
14.
       // the array will give the scanline number
        // The edge table (ET) with edges entries sorted
15.
16.
       // in increasing y and x of the lower end
17.
        int countEdgeBucket;
                               //no. of edgebuckets
18.
        EdgeBucket buckets[maxVer];
19. }EdgeTableTuple;
20.
21. EdgeTableTuple EdgeTable[maxHt], ActiveEdgeTuple;
22.
23. // Scanline Function
24. void MainWindow::initEdgeTable()
25. {
26.
       int i;
27.
        for (i=0; i<maxHt; i++)</pre>
28.
29.
            EdgeTable[i].countEdgeBucket = 0;
30.
31.
32.
       ActiveEdgeTuple.countEdgeBucket = 0;
33.}
34.
35. void insertionSort(EdgeTableTuple *ett)
36. {
37.
        int i,j;
38.
        EdgeBucket temp;
39.
       for (i = 1; i < ett->countEdgeBucket; i++)
40.
41.
42.
            temp.ymax = ett->buckets[i].ymax;
43.
            temp.xofymin = ett->buckets[i].xofymin;
44.
            temp.slopeinverse = ett->buckets[i].slopeinverse;
```

```
45.
            j = i - 1;
46.
47.
        while ((temp.xofymin < ett->buckets[j].xofymin) && (j >= 0))
48.
49.
            ett->buckets[j + 1].ymax = ett->buckets[j].ymax;
50.
            ett->buckets[j + 1].xofymin = ett->buckets[j].xofymin;
51.
            ett->buckets[j + 1].slopeinverse = ett->buckets[j].slopeinverse;
52.
            j = j - 1;
53.
54.
        ett->buckets[j + 1].ymax = temp.ymax;
55.
        ett->buckets[j + 1].xofymin = temp.xofymin;
56.
        ett->buckets[j + 1].slopeinverse = temp.slopeinverse;
57.
58.}
59.
60.
61. void storeEdgeInTuple (EdgeTableTuple *receiver,int ym,int xm,float slopInv)
62. {
        (receiver->buckets[(receiver)->countEdgeBucket]).ymax = ym;
63.
        (receiver->buckets[(receiver)->countEdgeBucket]).xofymin = (float)xm;
64.
65.
        (receiver->buckets[(receiver)->countEdgeBucket]).slopeinverse = slopInv;
66.
67.
        insertionSort(receiver);
68.
69.
        (receiver->countEdgeBucket)++;
70.
71.}
72.
73. void storeEdgeInTable (int x1,int y1, int x2, int y2)
74. {
75.
        float m,minv;
76.
        int ymaxTS,xwithyminTS, scanline;
77.
78.
       if (x2==x1)
79.
80.
            minv=0.000000;
81.
82.
        else
83.
84.
        m = ((float)(y2-y1))/((float)(x2-x1));
85.
86.
        if (y2==y1)
87.
            return;
88.
89.
        minv = (float)1.0/m;
90.
91.
92.
       if (y1>y2)
93.
        {
94.
            scanline=y2;
95.
            ymaxTS=y1;
96.
            xwithyminTS=x2;
97.
98.
        else
99.
        {
100.
                    scanline=y1;
101.
                   ymaxTS=y2;
102.
                   xwithyminTS=x1;
103.
104.
               storeEdgeInTuple(&EdgeTable[scanline],ymaxTS,xwithyminTS,minv);
105.
           }
```

```
106.
107.
           void removeEdgeByYmax(EdgeTableTuple *Tup,int yy)
108.
109.
               int i,j;
110.
               for (i=0; i< Tup->countEdgeBucket; i++)
111.
112.
113.
                    if (Tup->buckets[i].ymax == yy)
114.
115.
                        for ( j = i ; j < Tup->countEdgeBucket -1 ; j++ )
116.
117.
                                 Tup->buckets[j].ymax =Tup->buckets[j+1].ymax;
118.
                                 Tup->buckets[j].xofymin =Tup->buckets[j+1].xofymin;
119.
                                 Tup->buckets[j].slopeinverse = Tup-
    >buckets[j+1].slopeinverse;
120.
121.
                            Tup->countEdgeBucket--;
122.
                            i--;
123.
124.
125.
           }
126.
           void updatexbyslopeinv(EdgeTableTuple *Tup)
127.
128.
129.
               int i;
130.
131.
               for (i=0; i<Tup->countEdgeBucket; i++)
132.
133.
                    (Tup->buckets[i]).xofymin =(Tup->buckets[i]).xofymin + (Tup-
    >buckets[i]).slopeinverse;
134.
135.
136.
137.
138.
           void MainWindow::scanLineFill()
139.
140.
141.
                    int i, j, x1, ymax1, x2, ymax2, FillFlag = 0, coordCount;
142.
143.
                    for (i=0; i<maxHt; i++)</pre>
144.
145.
                        for (j=0; j<EdgeTable[i].countEdgeBucket; j++)</pre>
146.
                            storeEdgeInTuple(&ActiveEdgeTuple,EdgeTable[i].buckets[j].
147.
148.
                                     ymax,EdgeTable[i].buckets[j].xofymin,
149.
                                     EdgeTable[i].buckets[j].slopeinverse);
150.
                        }
151.
152.
                        removeEdgeByYmax(&ActiveEdgeTuple, i);
153.
154.
                        insertionSort(&ActiveEdgeTuple);
155.
                        j = 0;
156.
                        FillFlag = 0;
157.
                        coordCount = 0;
158.
                        x1 = 0;
159.
                        x2 = 0;
160.
                        ymax1 = 0;
161.
                        ymax2 = 0;
162.
                        while (j<ActiveEdgeTuple.countEdgeBucket)</pre>
163.
164.
                            if (coordCount%2==0)
```

```
165.
                             {
166.
                                 x1 = (int)(ActiveEdgeTuple.buckets[j].xofymin);
167.
                                 ymax1 = ActiveEdgeTuple.buckets[j].ymax;
168.
                                 if (x1==x2)
169.
170.
                                     if (((x1==ymax1)&&(x2!=ymax2))||((x1!=ymax1)&&(x2==ymax2)
    )))
171.
                                     {
172.
                                         x2 = x1;
173.
                                         ymax2 = ymax1;
174.
175.
                                     else
176.
177.
178.
                                         coordCount++;
179.
                                     }
180.
                                 }
181.
                                 else
182.
183.
                                         coordCount++;
184.
                                 }
185.
186.
                             else
187.
                                 x2 = (int)ActiveEdgeTuple.buckets[j].xofymin;
188.
                                 ymax2 = ActiveEdgeTuple.buckets[j].ymax;
189.
190.
191.
                                 FillFlag = 0;
192.
                                 if (x1==x2)
193.
                                 {
194.
                                     if (((x1==ymax1)&&(x2!=ymax2))||((x1!=ymax1)&&(x2==ymax2)
    )))
195.
196.
                                         x1 = x2;
                                         ymax1 = ymax2;
197.
198.
                                     }
199.
                                     else
200.
201.
                                          coordCount++;
202.
                                         FillFlag = 1;
203.
204.
                                 }
205.
                                 else
206.
207.
                                     coordCount++;
208.
                                     FillFlag = 1;
209.
                             if(FillFlag)
210.
211.
                             {
212.
                                     QPoint px,py;
213.
                                     px.setX(x1);px.setY(i);
214.
                                     py.setX(x2);py.setY(i);
215.
                                     drawBresenham(px,py);
216.
                             }
217.
                        }
218.
                        j++;
219.
220.
                    updatexbyslopeinv(&ActiveEdgeTuple);
221.
222.
                    vertex_list.clear();
223.
```

## Output of filling methods are shown in Figure 2.



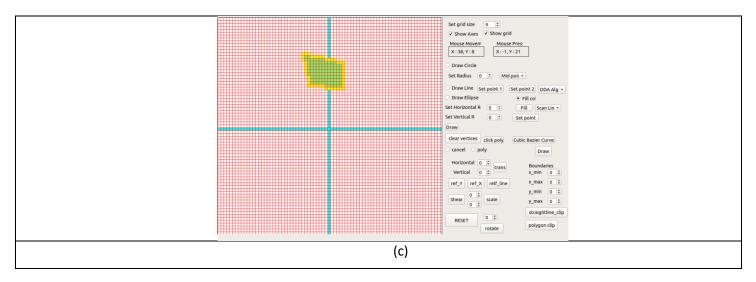


Figure 2: (a) Flood Fill (b) Boundary Fill (c) Scanline Fill

#### **Transformation:**

#### **Translation:**

```
1. void MainWindow::translate_object(int xx,int yy){
2.
3.
        on pushButton clicked();
4.
        on_show_grid_clicked();
5.
       int l = vertices.size();
6.
7.
        int i;
8.
9.
        for(i=0;i<1;i++){</pre>
10.
            QPoint p = vertices[i];
            p.setX(p.x()+xx);
11.
12.
            p.setY(p.y()+yy);
13.
14.
15.
16.
        for(i=0;i<1-1;i++){</pre>
17.
            drawBresenham(vertices[i], vertices[i+1]);
18.
19.
20.
        drawBresenham(vertices[0], vertices[1-1]);
21.
22.}
23.
24. void MainWindow::on_draw_ellipse_2_clicked()
25. {
26.
       flag = true;
27. }
28.
29. void MainWindow::on_Draw_2_clicked()
30. {
31.
        int l = vertices.size();
32.
       int i;
33.
34.
        for(i=0;i<1-1;i++){</pre>
35.
            drawBresenham(vertices[i], vertices[i+1]);
36.
37.
38.
        drawBresenham(vertices[0], vertices[1-1]);
39. }
40.
41. void MainWindow::on Draw 3 clicked()
42. {
43.
        int x = ui->hor_trans->value();
44.
       int y = ui->Ver_trans->value();
45.
        int k = ui->grid_size->value();
46.
       int xx = x*k;
47.
        int yy = y*k;
       int 1 = vertices.size();
48.
49.
        int i;
50.
51.
52.
       on_pushButton_clicked();
53.
        on_show_grid_clicked();
54.
       for(i=0;i<1;i++){</pre>
55.
            QPoint p = vertices[i];
56.
            p.setX(p.x()+xx);
```

```
p.setY(p.y()-yy);
57.
58.
            vertices[i] = p;
59.
60.
61.
62.
        for(i=0;i<1-1;i++){</pre>
            drawBresenham(vertices[i], vertices[i+1]);
63.
64.
65.
66.
        drawBresenham(vertices[0], vertices[1-1]);
67.}
```

#### Reflection about Y axis:

```
1. void MainWindow::on_Draw_4_clicked()
2. {
3.
4.
        on_pushButton_clicked();
5.
        on_show_grid_clicked();
        int i;
6.
7.
        int wid = img.width();
8.
        //int len = img.height();
9.
10.
       //QVector<QPoint> temp;
11.
        int l = vertices.size();
12.
        for(i=0;i<1;i++){</pre>
13.
            QPoint p = vertices[i];
            p.setX(wid - p.x());
14.
15.
            p.setY(p.y());
16.
            //temp.push_back(p);
17.
            vertices[i] = p;
18.
19.
        }
20.
21.
        for(i=0;i<1-1;i++){</pre>
22.
            drawBresenham(vertices[i], vertices[i+1]);
23.
24.
25.
        drawBresenham(vertices[0], vertices[1-1]);
26.}
```

#### **Reflection about X axis:**

```
    void MainWindow::on_Draw_5_clicked()

2. {
3.
4.
        on_pushButton_clicked();
5.
        on_show_grid_clicked();
6.
        int i;
        //int wid = img.width();
7.
8.
        int len = img.height();
9.
10.
       //QVector<QPoint> temp;
11.
        int 1 = vertices.size();
        for(i=0;i<1;i++){</pre>
12.
13.
            QPoint p = vertices[i];
14.
            p.setX(p.x());
            p.setY( len - p.y());
15.
          // temp.push_back(p);
16.
```

#### **Scaling:**

```
    void MainWindow::on_Draw_6_clicked()

2. {
3.
        on_pushButton_clicked();
4.
        on_show_grid_clicked();
5.
6.
        int x = ui->scaleratio->value();
7.
        int y = ui->scaleratio_2->value();
8.
        int i;
9.
        int wid = img.width();
        wid = wid/2;
10.
        int len = img.height();
11.
12.
        len = len/2;
13.
14.
        //QVector<QPoint> temp;
15.
        int 1 = vertices.size();
16.
        for(i=0;i<1;i++){</pre>
17.
            QPoint p = vertices[i];
18.
            p.setX((x*(p.x() - wid)) + wid);
19.
            p.setY((y*(p.y() - len)) + len);
20.
            //temp.push_back(p);
21.
            vertices[i] = p;
22.
23.
24.
25.
        for(i=0;i<1-1;i++){</pre>
            drawBresenham(vertices[i], vertices[i+1]);
26.
27.
28.
29.
        drawBresenham(vertices[0], vertices[1-1]);
30.
31. }
```

#### **Shear:**

```
    void MainWindow::on Draw 7 clicked()

2. {
3.
4.
        on_pushButton_clicked();
5.
        on_show_grid_clicked();
6.
7.
        int x = ui->scaleratio->value();
        int y = ui->scaleratio_2->value();
8.
        int i;
9.
10.
        int wid = img.width();
11.
        wid = wid/2;
12.
        int len = img.height();
```

```
13.
        len = len/2;
14.
15. //
          QVector<QPoint> temp;
16. int l = vertices.size();
17.
        for(i=0;i<1;i++){</pre>
            QPoint p = vertices[i];
18.
19.
20.
            int xx = p.x() - wid;
21.
            int yy = len - p.y();
22.
            p.setX(xx+yy*x + wid);
23.
            p.setY(2*len - (yy+xx*y + len));
24.
            vertices[i] = p;
25.
           // temp.push_back(p);
26.
27.
28.
29.
        for(i=0;i<1-1;i++){</pre>
30.
            drawBresenham(vertices[i], vertices[i+1]);
31.
32.
33.
        drawBresenham(vertices[0], vertices[1-1]);
34.}
```

#### Reflection about a line:

```
1. void MainWindow::on_Draw_8_clicked()
2. {
3.
4.
        on_pushButton_clicked();
5.
        on_show_grid_clicked();
6.
7.
        int wid = img.width();
       int len = img.height();
8.
9.
        wid = wid/2;
10.
       len = len/2;
11.
12.
       int x1 = p1.x() - wid;
13.
        int x2 = p2.x() - wid;
       int y1 = len - p1.y();
14.
        int y2 = len - p2.y();
15.
16.
17.
        int a,b,c;
18.
19.
        a = y2-y1;
20.
       b = x1-x2;
21.
        c = x1*(y1-y2) - y1*(x1-x2);
22.
23.
        int i;
24.
25.
26.
       int l = vertices.size();
27.
        for(i=0;i<1;i++){</pre>
28.
            QPoint p = vertices[i];
29.
30.
            int x = p.x() - wid;
            int y = len - p.y();
31.
32.
            int xx = (x*(b*b - a*a) - 2*a*(b*y+c))/(a*a + b*b);
33.
            int yy = (y*(a*a - b*b) - 2*b*(a*x+c))/(a*a + b*b);
            p.setX(xx + wid);
34.
35.
            p.setY(len - yy);
```

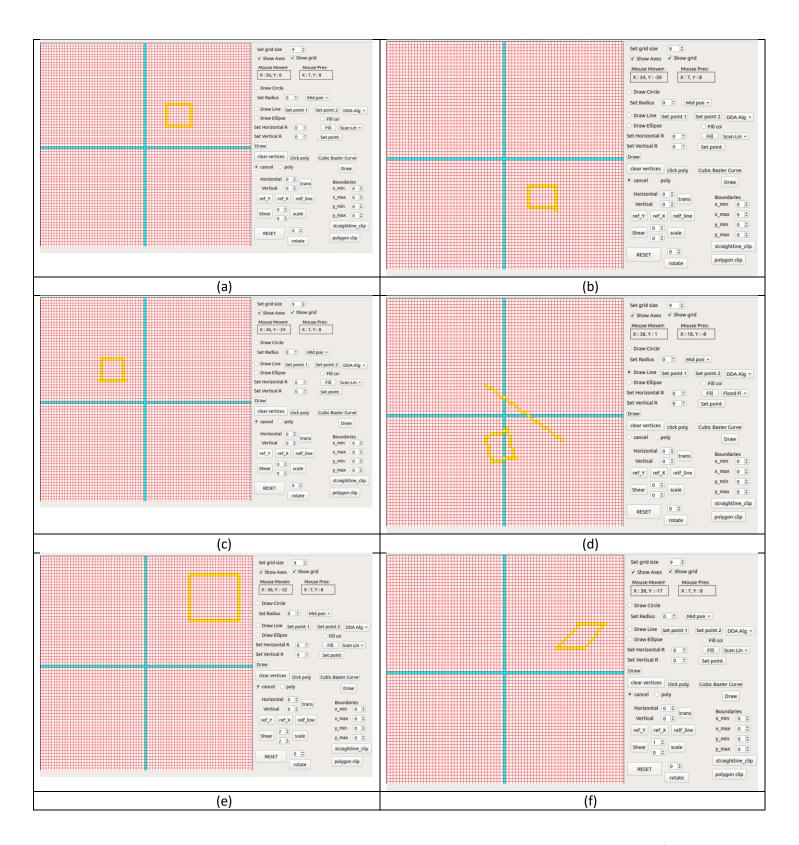
```
36.
            vertices[i] = p;
37.
        }
38.
39.
        for(i=0;i<1-1;i++){</pre>
40.
            drawBresenham(vertices[i], vertices[i+1]);
41.
        }
42.
43.
        drawBresenham(vertices[0], vertices[1-1]);
44.
        drawBresenham(p1,p2);
45.}
46.
47. void MainWindow::on_draw_ellipse_3_clicked()
48. {
49.
        flag = false;
50.
       // vertices.clear();
51.}
52.
53. void MainWindow::on_Draw_9_clicked()
54. {
55.
        flag = false;
        vertices.clear();
56.
57.}
```

#### **Rotation:**

```
    void MainWindow::on_Draw_10_clicked()

2. {
3.
4.
        on_pushButton_clicked();
5.
        on_show_grid_clicked();
6.
7.
        int angle = ui->rotate_point->value();
8.
        int x1 = p1.x();
9.
        int y1 = p1.y();
10.
11.
        float rad = (3.14/180.0)*angle;
12.
        float s = sin(rad);
13.
      float c = cos(rad);
        int i;
14.
15.
        int l = vertices.size();
16.
        for(i=0;i<1;i++){</pre>
17.
            QPoint p = vertices[i];
18.
19.
            int x = p.x() - x1;
20.
            int y = y1 - p.y();
21.
            int xn = int(x*1.0*c - y*1.0*s);
22.
23.
            int yn = int(y*1.0*c + x*1.0*s);
24.
25.
            p.setX(xn + x1);
            p.setY(y1 - yn);
26.
27.
            vertices[i] = p;
28.
29.
        for(i=0;i<1-1;i++){</pre>
30.
            drawBresenham(vertices[i], vertices[i+1]);
31.
32.
33.
        drawBresenham(vertices[0], vertices[1-1]);
34.
35.}
```

The output for various transformation is given in Figure 3.



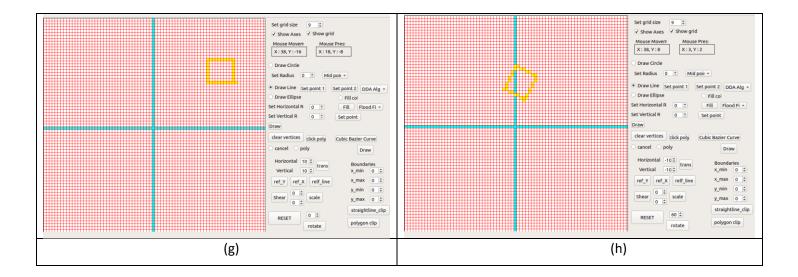


Figure 3: (a) The original drawing (b) Reflection about X axis (c) Reflection about Y axis (d) Reflection about a Line (e) Scaling (f) Shear (g) Translation (h) Rotation.

#### **Clipping:**

### **Cohen-Sutherland Line-Clipping:**

```
    const int INSIDE = 0; // 0000

2. const int LEFT = 1; // 0001
3. const int RIGHT = 2; // 0010
4. const int BOTTOM = 4; // 0100
5. const int TOP = 8; // 1000
6. int x_max,x_min,y_max,y_min;
8. void MainWindow::on_Draw_13_clicked()
9. {
10.
       QPoint px,py;
11.
       int k=ui->grid size->value();
12. x max=ui->xmax->value()*k+350;
13.
       x min=ui->xmin->value()*k+350;
14. y_max=350-ui->ymax->value()*k;
15.
       y_min=350-ui->ymin->value()*k;
16. px.setX(x_max);
17.
       px.setY(y_max);
18. py.setX(x_min);
19.
       py.setY(y_max);
20. drawBresenham(px,py);
21.
       px.setX(x min);
22. px.setY(y min);
23.
       drawBresenham(px,py);
24. py.setX(x max);
25.
       py.setY(y min);
    drawBresenham(px,py);
26.
27.
       px.setX(x_max);
28. px.setY(y_max);
```

```
29.
       drawBresenham(px,py);
30.}
31.
32. int MainWindow::computeCode(double x, double y)
33. {
34. // initialized as being inside
35.
       int code = INSIDE;
36.
       if (x < x_min)
                           // to the left of rectangle
37.
           code |= LEFT;
38.
       else if (x > x max) // to the right of rectangle
39.
           code |= RIGHT;
40.
       if (y > y_min)
                            // below the rectangle
41.
           code |= BOTTOM;
42.
       else if (y < y_max) // above the rectangle</pre>
43.
           code |= TOP;
44.
45.
       return code;
46.}
47.
48. int MainWindow::x_intersect(int x1, int y1, int x2, int y2,int x3, int y3, int x4, int
  y4)
49. {
50.
       y1=ui->frame->height()-y1+1;
51.
       y2=ui->frame->height()-y2+1;
52.
       y3=ui->frame->height()-y3+1;
       y4=ui->frame->height()-y4+1;
53.
       int num = (x1*y2 - y1*x2) * (x3-x4) - (x1-x2) * (x3*y4 - y3*x4);
54.
       int den = (x1-x2) * (y3-y4) - (y1-y2) * (x3-x4);
55.
56.
       int retx=num/den;
       return retx;
57.
58.}
59. int MainWindow::y intersect(int x1, int y1, int x2, int y2,int x3, int y3, int x4, int
   v4)
60. {
       y1=ui->frame->height()-y1+1;
61.
       y2=ui->frame->height()-y2+1;
62.
       y3=ui->frame->height()-y3+1;
       y4=ui->frame->height()-y4+1;
64.
       int num = (x1*y2 - y1*x2) * (y3-y4) - (y1-y2) * (x3*y4 - y3*x4);
       int den = (x1-x2) * (y3-y4) - (y1-y2) * (x3-x4);
67.
        int rety= (ui->frame->height()-num/den+1);
68.
       return rety;
69.}
70.
71. void MainWindow::on Draw 11 clicked()
72. {
73.
       double x1=vertices[0].x();
       double y1=vertices[0].y();
74.
75.
        double x2=vertices[1].x();
       double y2=vertices[1].y();
76.
77.
       int code1=computeCode(x1,y1);
78.
       int code2=computeCode(x2,y2);
79.
        bool accept = false;
80.
       while (true)
81.
82.
           if ((code1 == 0) && (code2 == 0))
83.
84.
               accept = true;
85.
                break;
86.
           else if (code1 & code2)
87.
```

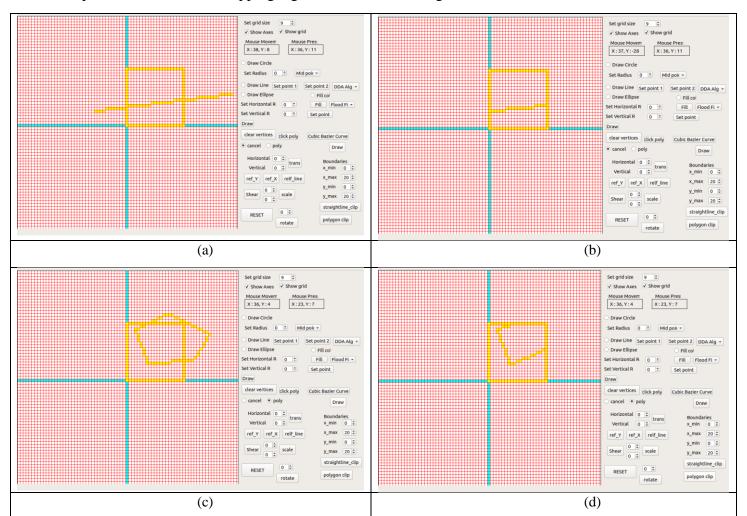
```
88.
89.
                 break;
90.
91.
            else
92.
93.
                 int code_out;
94.
                int x, y;
95.
                 if (code1 != 0)
96.
                     code_out = code1;
97.
98.
                     code out = code2;
99.
                 if (code_out & TOP)
100.
101.
                            x = x1 + (int)((double)(x2 - x1) *(double)(y_max - y1) /(double)
    (y2 - y1));
102.
                            y = y_max;
103.
104.
                        else if (code_out & BOTTOM)
105.
                            x = x1 + (int)((double)(x2 - x1) * (double)(y_min - y1) / (double)
106.
   e)(y2 - y1));
107.
                            y = y_{min};
108.
                        else if (code_out & RIGHT)
109.
110.
                            y = y1 + (int)((double)(y2 - y1) * (double)(x_max - x1) / (double)
    e)(x2 - x1));
112.
                            x = x_max;
113.
                        }
                        else if (code out & LEFT)
114.
115.
                        {
116.
                            y = y1 + (int)((double)(y2 - y1) * (double)(x_min - x1) / (double)
   e)(x2 - x1));
117.
                            x = x_{min};
118.
                        }
119.
                        if (code_out == code1)
120.
121.
                            x1 = x;
122.
                            y1 = y;
123.
                             code1 = computeCode(x1, y1);
124.
                        }
125.
                        else
126.
127.
                            x2 = x;
128.
                            y2 = y;
                             code2 = computeCode(x2, y2);
129.
130.
131.
                    }
132.
133.
                on pushButton clicked();
                on_show_grid_clicked();
134.
135.
                on_Draw_13_clicked();
136.
137.
                if (accept)
138.
                    QPoint px,py;
139.
140.
                    px.setX((int)x1);
141.
                    px.setY((int)y1);
142.
                    py.setX((int)x2);
143.
                    py.setY((int)y2);
144.
                    drawBresenham(px,py);
```

```
145. }
146.
147. }
```

#### Sutherland-Hodgman's polygon-clipping

```
1. void MainWindow::clip(int x1, int y1, int x2, int y2)
2. {
3.
        int poly_size=vertices.size()-1;
4.
        int new_poly_size = 0;
5.
        std::vector<std::pair<int,int> > new_points;
6.
        for (int i = 0; i < poly_size; i++)</pre>
7.
8.
            int k = (i+1) % poly_size;
9.
            int ix = vertices[i].x(), iy = vertices[i].y();
10.
            int kx = vertices[k].x(), ky = vertices[k].y();
11.
            int i_pos,k_pos;
12.
            if(x2==x1 && ix>x1) i_pos=1;
13.
            else if(x2==x1 && ix<x1) i_pos=-1;</pre>
            else if(y2==y1 && iy<y1) i_pos=1;</pre>
14.
15.
            else i_pos=-1;
16.
            if(x2==x1 && kx>x1) k pos=1;
17.
            else if(x2==x1 && kx<x1) k pos=-1;
            else if(y2==y1 && ky<y1) k_pos=1;</pre>
18.
19.
            else k pos=-1;
20.
            if(y1>y2||x1>x2)
21.
            {
22.
                i pos=(-1)*i_pos;
23.
                k_pos=(-1)*k_pos;
24.
25.
            if (i pos >= 0 && k pos >= 0)
26.
27.
                new_points.push_back(std::make_pair(kx,ky));
28.
                new_poly_size++;
29.
30.
            else if (i pos < 0 && k pos >= 0)
31.
            {
32.
                new points.push back(std::make pair(x intersect(x1,y1, x2, y2, ix, iy, kx,
    ky),
33.
                                                       y_intersect(x1,y1, x2, y2, ix, iy, kx,
    ky)));
34.
                new poly size++;
35.
                new_points.push_back(std::make_pair(kx,ky));
36.
                new_poly_size++;
37.
38.
            else if (i pos >= 0 && k pos < 0)
39.
            {
40.
                new points.push back(std::make pair(x intersect(x1,y1, x2, y2, ix, iy, kx,
41.
    ky),
42.
                                                       y_intersect(x1,y1, x2, y2, ix, iy, kx,
   ky)));
43.
                new_poly_size++;
44.
            }
45.
            else
46.
            {
47.
48.
```

```
49.
        poly_size = new_poly_size;
50.
        vertices.clear();
51.
        for (int i = 0; i < new_points.size(); i++)</pre>
52.
53.
            QPoint p;
54.
            p.setX(new_points[i].first);
55.
            p.setY(new_points[i].second);
            vertices.push back(p);
56.
57.
58.
        if(poly_size>0){
59.
            QPoint pp;
            pp.setX(new_points[0].first);
60.
            pp.setY(new_points[0].second);
61.
62.
            vertices.push_back(pp);
63.
64.}
65. void MainWindow::suthHodgClip()
66. {
67.
68.
        clip(x_min,y_max,x_min,y_min); //Left
69.
        if(vertices.size()>0)
70.
            clip(x_min,y_min,x_max,y_min); //Bottom
71.
        if(vertices.size()>1)
72.
            clip(x_max,y_min,x_max,y_max); //Right
73.
        if(vertices.size()>1)
74.
            clip(x_max,y_max,x_min,y_max); //Top
75.
        on pushButton clicked();
76.
        on_show_grid_clicked();
77.
        on_Draw_13_clicked();
78.
        int l=vertices.size();
79.
        QPoint px,py;
80.
        int k;
81.
        if(1>1){
82.
           for(int i=0;i<l-1;i++){</pre>
83.
               k=(i+1)%1;
84.
               px.setX(vertices[i].x());
               px.setY(vertices[i].y());
85.
               py.setX(vertices[k].x());
86.
87.
               py.setY(vertices[k].y());
88.
               drawBresenham(px,py);
89.
           }
90.
91.
        }
92.
93.}
95. void MainWindow::on Draw 12 clicked()
96. {
97.
        suthHodgClip();
98.}
```



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

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()

    2. {
                                                                                  double xu = 0.0 , yu = 0.0 , u = 0.0 ;
    3.
    4.
                                                                               int i = 0;
                                                                                    for(u = 0.0 ; u <= 1.0 ; u += 0.0001)
    5.
    6.
                                                                                                                         xu = pow(1-u,3)*vertices[0].x()+3*u*pow(1-u,2)*vertices[1].x()+3*pow(u,2)*(1-u,2)*vertices[1].x()+3*pow(u,2)*(1-u,2)*vertices[1].x()+3*pow(u,2)*(1-u,2)*vertices[1].x()+3*pow(u,2)*(1-u,2)*vertices[1].x()+3*pow(u,2)*(1-u,2)*vertices[1].x()+3*pow(u,2)*(1-u,2)*vertices[1].x()+3*pow(u,2)*(1-u,2)*vertices[1].x()+3*pow(u,2)*(1-u,2)*vertices[1].x()+3*pow(u,2)*(1-u,2)*vertices[1].x()+3*pow(u,2)*(1-u,2)*vertices[1].x()+3*pow(u,2)*(1-u,2)*vertices[1].x()+3*pow(u,2)*(1-u,2)*vertices[1].x()+3*pow(u,2)*(1-u,2)*vertices[1].x()+3*pow(u,2)*(1-u,2)*vertices[1].x()+3*pow(u,2)*(1-u,2)*vertices[1].x()+3*pow(u,2)*(1-u,2)*vertices[1].x()+3*pow(u,2)*(1-u,2)*vertices[1].x()+3*pow(u,2)*(1-u,2)*vertices[1].x()+3*pow(u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u,2)*(1-u
                                        u)*vertices[2].x()+pow(u,3)*vertices[3].x();
   8.
                                                                                                                        yu = pow(1-u,3)*vertices[0].y()+3*u*pow(1-u,2)*vertices[1].y()+3*pow(u,2)*(1-u,2)*vertices[1].y()+3*pow(u,2)*(1-u,2)*vertices[1].y()+3*pow(u,2)*(1-u,2)*vertices[1].y()+3*pow(u,2)*(1-u,2)*vertices[1].y()+3*pow(u,2)*(1-u,2)*vertices[1].y()+3*pow(u,2)*(1-u,2)*vertices[1].y()+3*pow(u,2)*(1-u,2)*vertices[1].y()+3*pow(u,2)*(1-u,2)*vertices[1].y()+3*pow(u,2)*(1-u,2)*vertices[1].y()+3*pow(u,2)*(1-u,2)*vertices[1].y()+3*pow(u,2)*(1-u,2)*vertices[1].y()+3*pow(u,2)*(1-u,2)*vertices[1].y()+3*pow(u,2)*(1-u,2)*vertices[1].y()+3*pow(u,2)*(1-u,2)*vertices[1].y()+3*pow(u,2)*(1-u,2)*vertices[1].y()+3*pow(u,2)*(1-u,2)*vertices[1].y()+3*pow(u,2)*(1-u,2)*vertices[1].y()+3*pow(u,2)*(1-u,2)*vertices[1].y()+3*pow(u,2)*(1-u,2)*vertices[1].y()+3*pow(u,2)*(1-u,2)*vertices[1].y()+3*pow(u,2)*(1-u,2)*vertices[1].y()+3*pow(u,2)*(1-u,2)*vertices[1].y()+3*pow(u,2)*vertices[1].y()+3*pow(u,2)*vertices[1].y()+3*pow(u,2)*vertices[1].y()+3*pow(u,2)*vertices[1].y()+3*pow(u,2)*vertices[1].y()+3*pow(u,2)*vertices[1].y()+3*pow(u,2)*vertices[1].y()+3*pow(u,2)*vertices[1].y()+3*pow(u,2)*vertices[1].y()+3*pow(u,2)*vertices[1].y()+3*pow(u,2)*vertices[1].y()+3*pow(u,2)*vertices[1].y()+3*pow(u,2)*vertices[1].y()+3*pow(u,2)*vertices[1].y()+3*pow(u,2)*vertices[1].y()+3*pow(u,2)*vertices[1].y()+3*pow(u,2)*vertices[1].y()+3*pow(u,2)*vertices[1].y()+3*pow(u,2)*vertices[1].y()+3*pow(u,2)*vertices[1].y()+3*pow(u,2)*vertices[1].y()+3*pow(u,2)*vertices[1].y()+3*pow(u,2)*vertices[1].y()+3*pow(u,2)*vertices[1].y()+3*pow(u,2)*vertices[1].y()+3*pow(u,2)*vertices[1].y()+3*pow(u,2)*vertices[1].y()+3*pow(u,2)*vertices[1].y()+3*pow(u,2)*vertices[1].y()+3*pow(u,2)*vertices[1].y()+3*pow(u,2)*vertices[1].y()+3*pow(u,2)*vertices[1].y()+3*pow(u,2)*vertices[1].y()+3*pow(u,2)*vertices[1].y()+3*pow(u,2)*vertices[1].y()+3*pow(u,2)*vertices[1].y()+3*pow(u,2)*vertices[1].y()+3*pow(u,2)*vertices[1].y()+3*pow(u,2)*vertices[1].y()+3*pow(u,2)*vertices[1].y()+3*pow(u,2)*vertices[1].y()+3*pow(u,2)*vertices[1].y()+3*pow(u,2)*vertices[1].y()+3*pow(u,2)*vertices[1].y()+3*pow(u,2)*vertices[1].
                                     u)*vertices[2].y()+pow(u,3)*vertices[3].y();
    9.
                                                                                                                           point((int)xu , (int)yu);
   10.
11. }
```

Output for Bazier curve is shown in Figure 5.

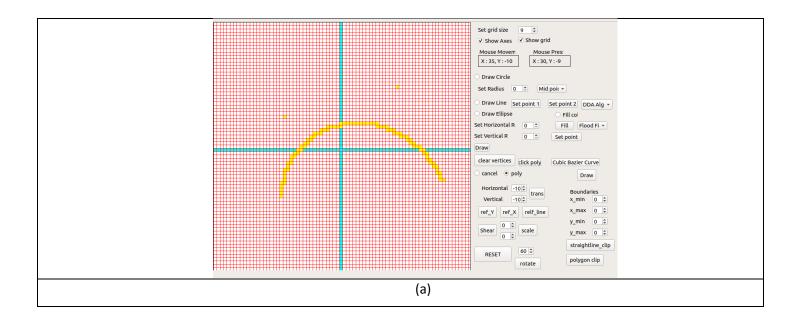


Figure 5: (a) Bazier curve