COMPUTER GRAPHICS LABORATORY REPORT

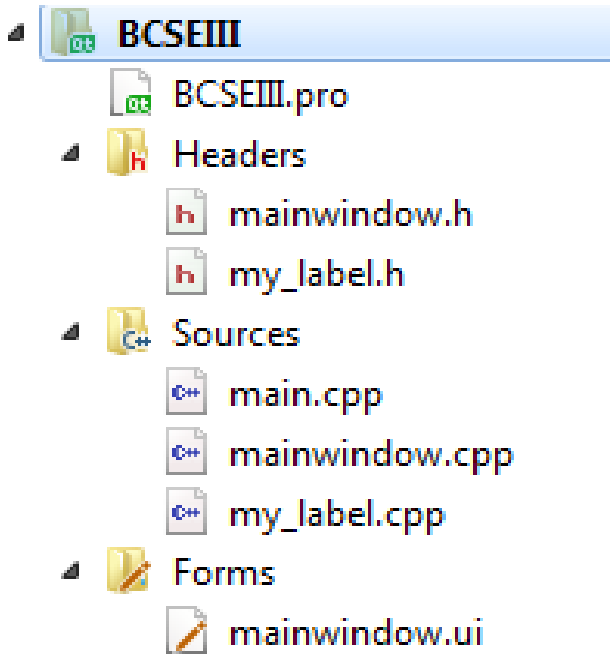
AMARTYA CHOUDURY
BODHISATTWA HALDER
INDIRA BISWAS

CONTENTS:

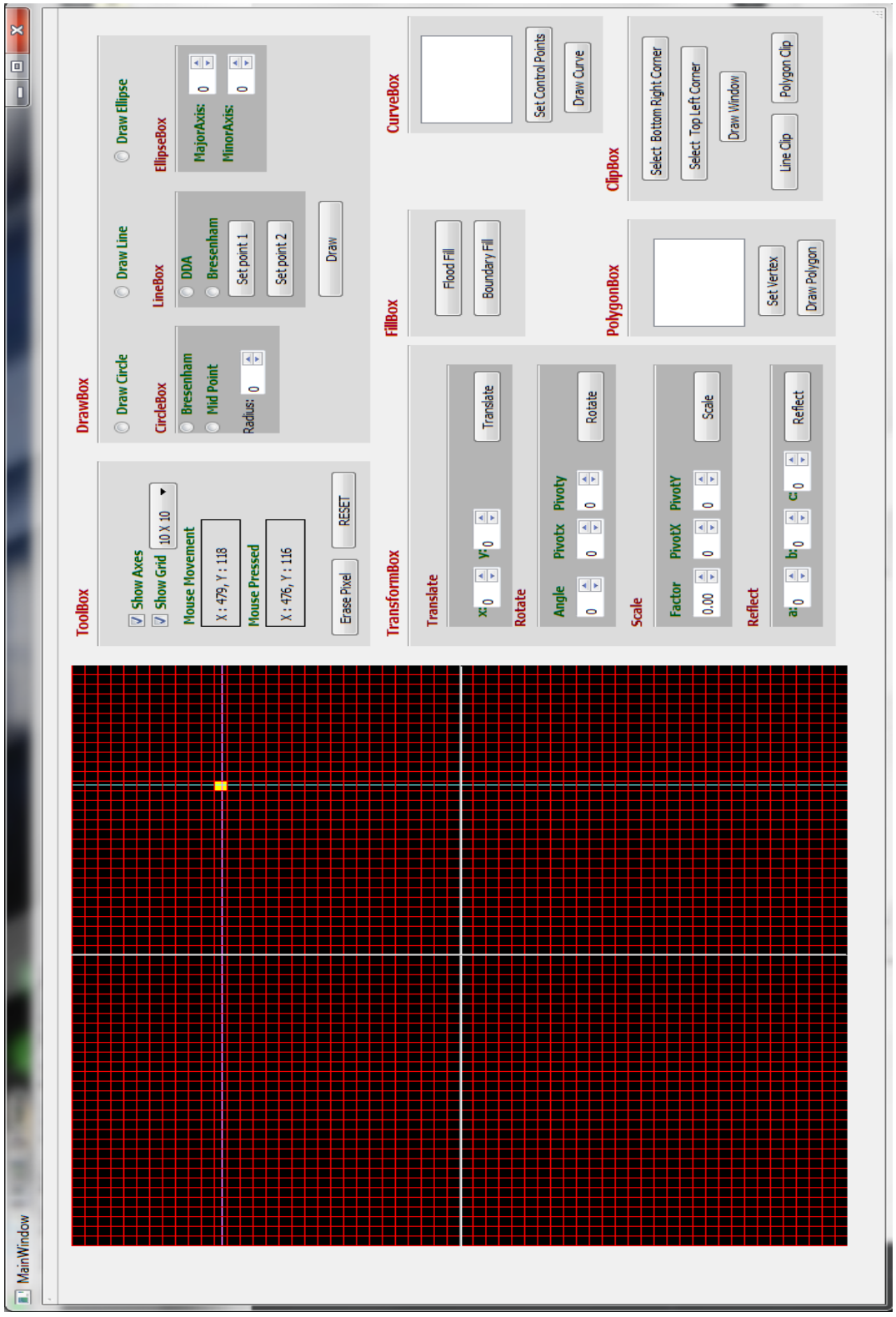
PROJECT DESCRIPTION:

The goal is to create a tool that can draw shapes on a rasterised screen , fill the shapes, perform basic transformations on the shapes and clip the shapes with respect to a clipping window. We use Qt 5.10.1 and Qt Creator 4.5.0 to create this tool .

Project files :



UI design:


MainWindow
ToolBox
Show Axes
Show Grid
10 X 10
Mouse Movement
X : 479, Y : 118
Mouse Pressed
X : 476, Y : 116
Erase Pixel
RESET
DrawBox
Draw Circle
Draw Line
Draw Ellipse
CircleBox
Bresenham
Mid Point
Radius: 0
LineBox
DDA
Bresenham
Set point 1
Set point 2
EllipseBox
MajorAxis: 0
MinorAxis: 0
Draw
TransformBox
Translate
x: 0
y: 0
Translate
Rotate
Angle
Pivotx
Pivoty
Rotate
Scale
Factor
PivotX
PivotY
0.00
Scale
Reflect
a: 0
b: 0
c: 0
Reflect
FillBox
Flood Fill
Boundary Fill
PolygonBox
Set Vertex
Draw Polygon
CurveBox
Set Control Points
Draw Curve
ClipBox
Select Bottom Right Corner
Select Top Left Corner
Draw Window
Line Clip
Polygon Clip

SETTING UP THE TOOL :

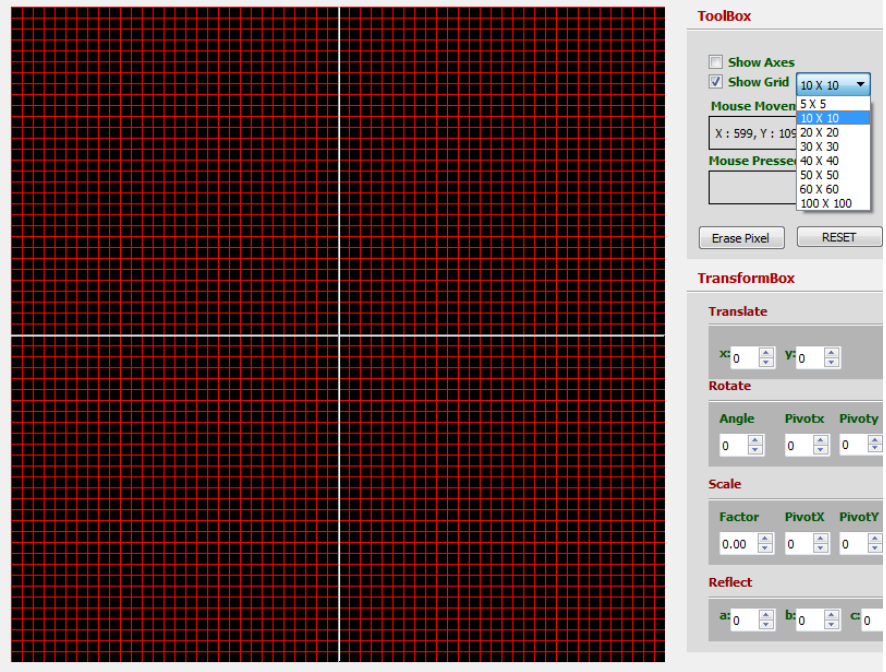
UI setup:

Code:

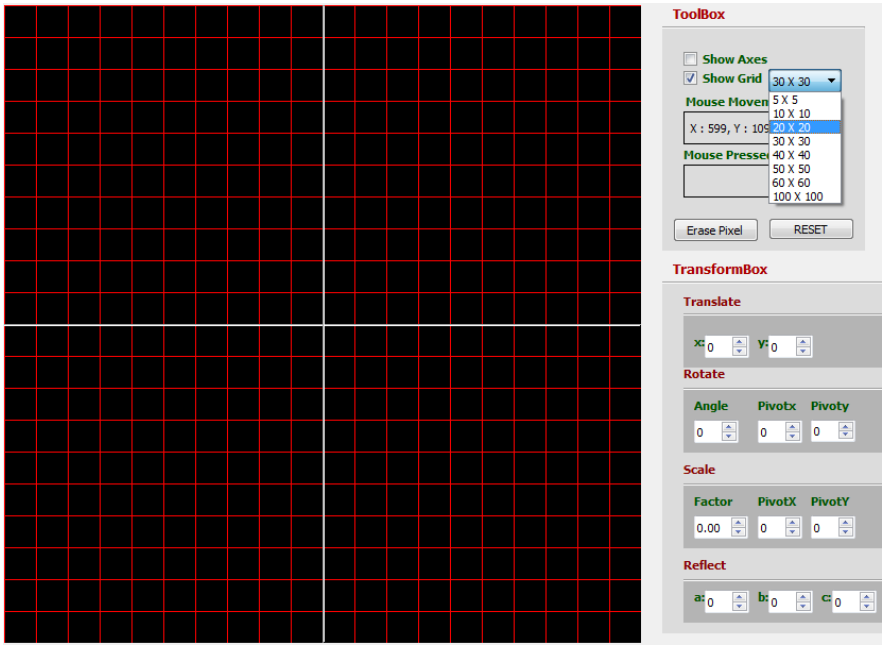
```
/*image on which shapes wouldbe drawn*/
QImage img;
/*image on which grids will be drawn*/
QImage grid;
/*size of grid , a variable */
int grid_size;

MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{   ui->setupUi(this);
    ui->x_axis->hide();
    ui->y_axis->hide();

    img=QImage(ui->label->width(),ui->label-
>height(),QImage::Format_RGB888);
    grid=QImage(ui->label->width(),ui->label-
>height(),QImage::Format_ARGB32);
    connect(ui->label,SIGNAL(Mouse_Pos()),this,SLOT(Mouse_Pressed()));

    /*enabling mouse functionality*/

connect(ui-
>label,SIGNAL(sendMousePosition(QPoint&)),this,SLOT(showMousePosition(Q
Point&)));
    ChangeGridSize(5);
    ui->label->setVisible(false);
    grid_size=5;

    /*setting upper and lower limits of  spinboxes */
    ui->xTrans->setMaximum(img.width()/grid_size);
    ui->yTrans->setMaximum(img.height()/grid_size);

    ui->pivotx->setMaximum(img.width()/grid_size);
    ui->pivoty->setMaximum(img.height()/grid_size);

    ui->xTrans->setMinimum(-img.width()/grid_size);
    ui->yTrans->setMinimum(-img.height()/grid_size);

    ui->pivotx->setMinimum(-img.width()/grid_size);
    ui->pivoty->setMinimum(-img.height()/grid_size);

    ui->scalePivotX->setMaximum(img.width()/grid_size);
```

```
    ui->scalePivotY->setMaximum(img.height()/grid_size);

    ui->scalePivotX->setMinimum(-img.width()/grid_size);
    ui->scalePivotY->setMinimum(-img.height()/grid_size);

    ui->a->setMaximum(img.width()/grid_size);
    ui->b->setMaximum(img.height()/grid_size);
    ui->c->setMaximum(img.height()/grid_size);

    ui->a->setMinimum(-img.width()/grid_size);
    ui->b->setMinimum(-img.height()/grid_size);
    ui->c->setMinimum(-img.height()/grid_size);

    ui->degrees->setMaximum(360);
    ui->degrees->setMinimum(-360);

    /*setting bg color of groupBoxes*/
    QPalette outerpal = palette();
    outerpal.setColor(QPalette::Background, qRgb(220,220,220));
    ui->toolBox->setAutoFillBackground(true);
    ui->toolBox->setPalette(outerpal);
    ui->drawBox->setAutoFillBackground(true);
    ui->drawBox->setPalette(outerpal);
    ui->fillBox->setAutoFillBackground(true);
    ui->fillBox->setPalette(outerpal);
    ui->transformBox->setAutoFillBackground(true);
    ui->transformBox->setPalette(outerpal);
    ui->clipBox->setAutoFillBackground(true);
    ui->clipBox->setPalette(outerpal);
    ui->curveBox->setAutoFillBackground(true);
    ui->curveBox->setPalette(outerpal);
    ui->polygonBox->setAutoFillBackground(true);
    ui->polygonBox->setPalette(outerpal);

    QPalette innerpal = palette();
    innerpal.setColor(QPalette::Background, qRgb(180,180,180));

    ui->circleBox->setAutoFillBackground(true);
    ui->circleBox->setPalette(innerpal);
    ui->ellipseBox->setAutoFillBackground(true);
    ui->ellipseBox->setPalette(innerpal);
    ui->lineBox->setAutoFillBackground(true);
    ui->lineBox->setPalette(innerpal);
    ui->scaleBox->setAutoFillBackground(true);
    ui->scaleBox->setPalette(innerpal);
    ui->translateBox->setAutoFillBackground(true);
    ui->translateBox->setPalette(innerpal);
    ui->rotatationBox->setAutoFillBackground(true);
    ui->rotatationBox->setPalette(innerpal);
    ui->reflectBox->setAutoFillBackground(true);
    ui->reflectBox->setPalette(innerpal);

}
```

Implementing Grid Structure :



10 x 10 grid

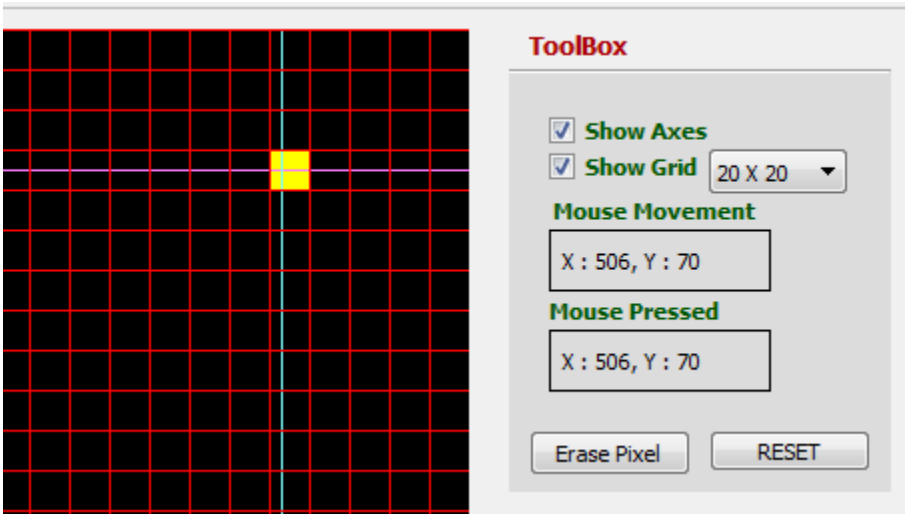


30 x 30 grid

Code:

```
void MainWindow::on_comboBox_activated(const QString &arg1)
{
    if(arg1=="5 X 5"){ChangeGridSize(5);grid_size=5;}
    if(arg1=="10 X 10") {ChangeGridSize(10);grid_size=10;}
    if(arg1=="20 X 20") {ChangeGridSize(20);grid_size=20;}
    if(arg1=="30 X 30") {ChangeGridSize(30);grid_size=30;}
    if(arg1=="40 X 40"){ ChangeGridSize(40);grid_size=40;}
    if(arg1=="50 X 50"){ ChangeGridSize(50);grid_size=50;}
    if(arg1=="60 X 60") {ChangeGridSize(60);grid_size=60;}
    if(arg1=="100 X 100") {ChangeGridSize(100);grid_size=100;}
}

void MainWindow::ChangeGridSize(int arg1)
{
    for(int i=0;i<grid.width();i++){
        for(int j=0;j<=grid.height();j++){
            grid.setPixel(i,j,qRgba(0,0,0,0));
        }
    }
    for(int i=0;i<grid.width();i+=arg1){
        for(int j=0;j<=grid.height();j++){
            grid.setPixel(i,j,qRgb(255,0,0));
        }
    }
    for(int i=0;i<grid.width();i+=arg1){
        for(int j=0;j<=grid.height();j++){
            grid.setPixel(j,i,qRgb(255,0,0));
        }
    }
    ui->label->setPixmap(QPixmap::fromImage(grid));
    on_pushButton_clicked();

}
```

Selecting points from the image:




```
void MainWindow::point(int x,int y)
{

    for(int i=x-x%grid_size;i<x-x%grid_size+grid_size;i++){
    for(int j=y-y%grid_size;j<y-y%grid_size+grid_size;j++){
        img.setPixel(i,j,qRgb(255,255,0));
    }
    }
    ui->frame->setPixmap(QPixmap::fromImage(img));
}

void MainWindow::erase(int x, int y)
{
    for(int i=x-x%grid_size;i<x-x%grid_size+grid_size;i++){
    for(int j=y-y%grid_size;j<y-y%grid_size+grid_size;j++){
        img.setPixel(i,j,qRgb(0,0,0));
    }
    }
    ui->frame->setPixmap(QPixmap::fromImage(img));
}

void MainWindow::showMousePosition(QPoint &pos)
{
    ui->mouse_movement->setText(" X : "+QString::number(pos.x())+", Y :
"+QString::number(pos.y()));
}
void MainWindow::Mouse_Pressed()
{
    ui->mouse_pressed->setText(" X : "+QString::number(ui->label->x)+",
Y : "+QString::number(ui->label->y));
    point(ui->label->x,ui->label->y);
    ui->x_axis->move(60,ui->label->y+10);
    ui->y_axis->move(ui->label->x+60,10);
}
```

DRAWING SHAPES:

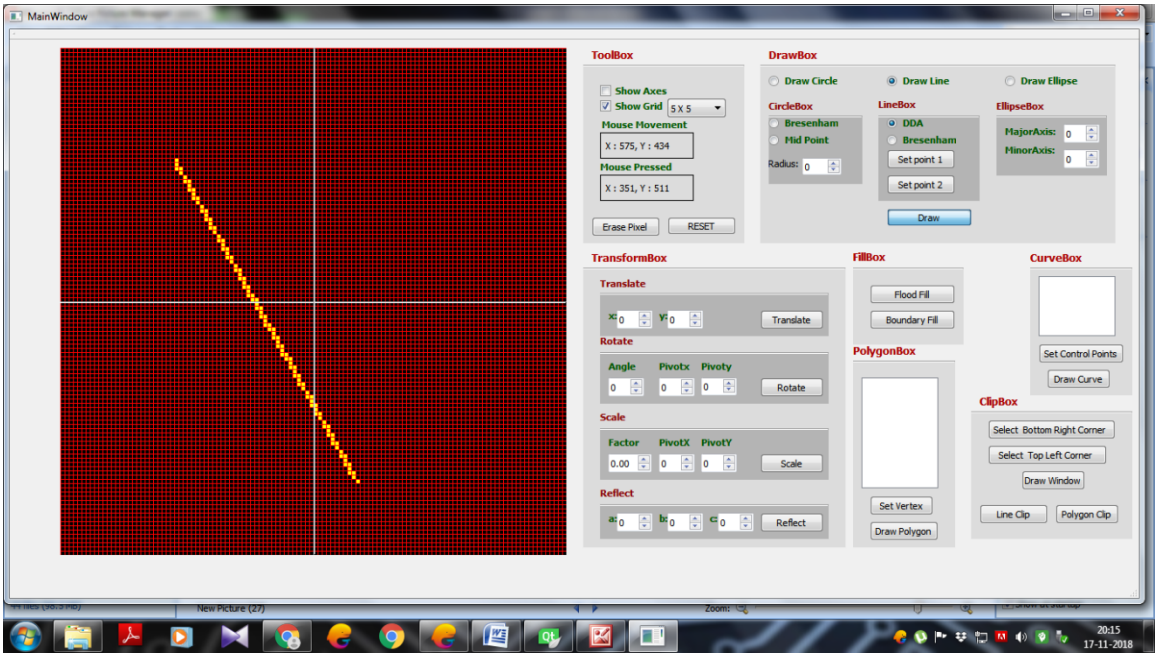
a.Line drawing algorithms:

i. DDA line drawing

Code:

```
void MainWindow::on_Draw_clicked()
{

    if(ui->draw_line->isChecked()){
        qDebug()<<p1.x();
        lineEp1.append(p1);
        lineEp2.append(p2);

        if(ui->dda->isChecked()){
            DDA(p1.x(),p1.y(),p2.x(),p2.y());
        }
        else if(ui->bresenham->isChecked()){
            bresenham(p1.x(),p1.y(),p2.x(),p2.y());
        }
        else{
        painter.setPen(Qt::red);
        painter.drawLine(p1,p2);
        }
    }
    ui->frame->setPixmap(QPixmap::fromImage(img));
}
void MainWindow::DDA(int x0, int y0, int x1, int y1){
    int dx=x1-x0;
    int dy=y1-y0;
    int step=abs(dx)>abs(dy)?abs(dx):abs(dy);
    double Xinc = dx / (float) step;
    double Yinc = dy / (float) step;
    double X = x0;
    double Y = y0;
    for(int i=0;i<=step;i++){
        point(X,Y);
        X+= Xinc;
        Y+= Yinc;
    }

}
```

Output:



ii. Bresenham line drawing

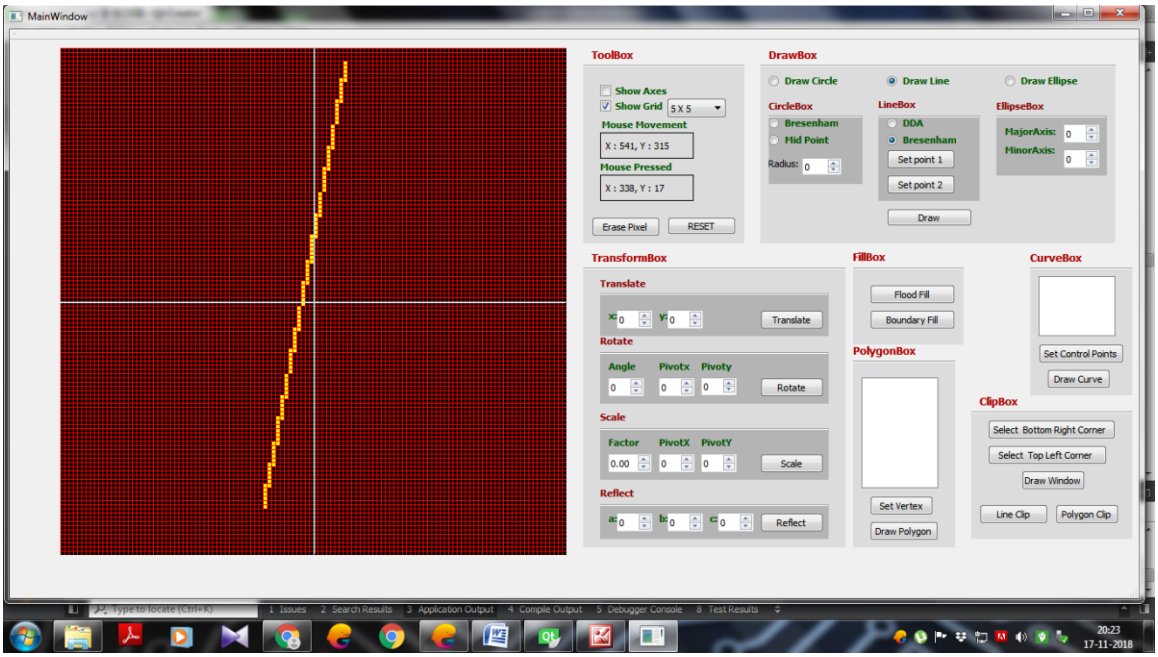
Code:

```
void MainWindow::bresenham(int x0, int y0, int x1, int y1){
    int dx=abs(x1-x0);
    int dy=abs(y1-y0);
    int p,x,y,xEnd,yEnd,m1,m2;
    if(dx>dy){
        p=2*dy-dx;
        m1=2*dy;
        m2=2*dy-2*dx;
        if(x0>x1){
            x=x1;
            y=y1;
            xEnd=x0;
            yEnd=y0;
            }
        else{
            x=x0;
            y=y0;
            xEnd=x1;
            yEnd=y1;
```

```
        }
        point(x,y);
        if(yEnd>y){
            while(x<xEnd){
               x++;
                if(p<0) p+=m1;
                else{
                    y++;
                    p+=m2;
                 }
                point(x,y);
            }
        }
        else{
            while(x<xEnd){
                x++;
                if(p<0) p+=m1;
                else{
                    y--;
                    p+=m2;
                 }
                point(x,y);
            }
        }
    }
    else{
        swap(dx,dy);
        swap(x0,y0);
        swap(x1,y1);
        p=2*dy-dx;
        m1=2*dy;
        m2=2*dy-2*dx;
        if(x0>x1){
            x=x1;
            y=y1;
            xEnd=x0;
            yEnd=y0;
            }
        else{
            x=x0;
            y=y0;
            xEnd=x1;
            yEnd=y1;
        }
        point(y,x);
        if(yEnd>y){
            while(x<xEnd){
               x++;
                if(p<0) p+=m1;
                else{
                    y++;
                    p+=m2;
                 }
                point(y,x);
            }
        }
        else{
```

```
            while(x<xEnd){
                x++;
                if(p<0)  p+=m1;
                else{
                    y--;
                    p+=m2;
                 }
                point(y,x);
            }
        }
    }
}
```

Output:



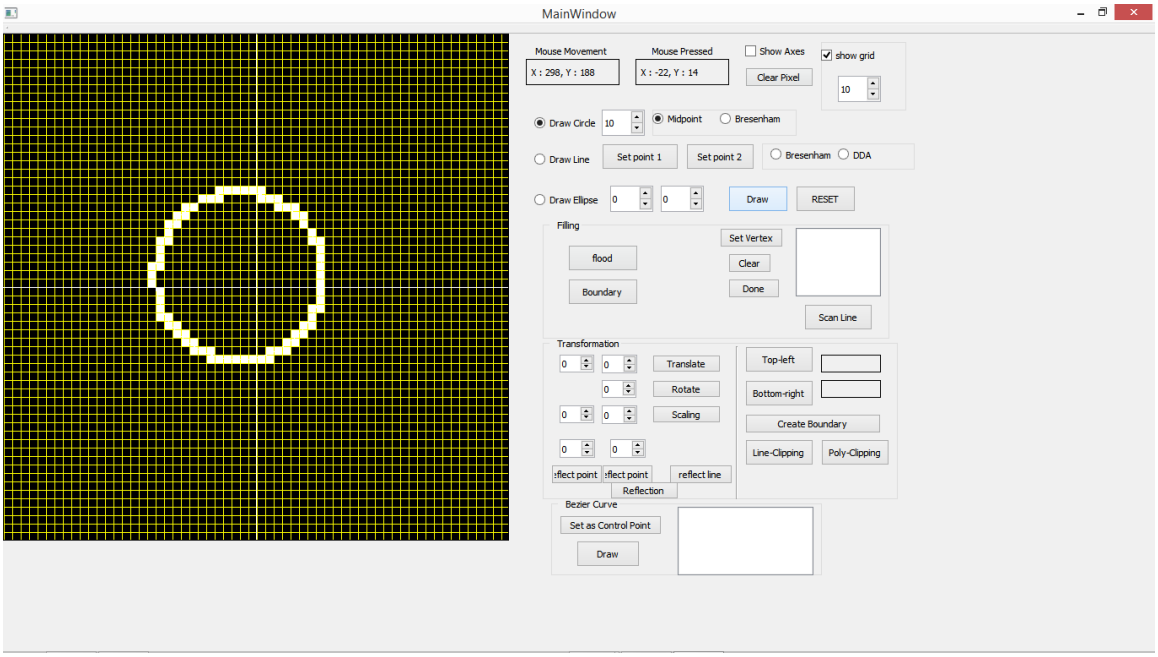
b. Circle drawing algorithms:

i. midpoint circle drawing

Code:

```
voidMainWindow::circle(intx0,inty0,intr,uintcolor)
{
r=r*(ui->gridsize->value());

//***************MID-POINTCIRCLE************

if(ui->circlemid->isChecked()){
intx=r;
inty=0;
interr=0;
while(x>=y){
point(x0+x,y0+y,color);
point(x0+y,y0+x,color);
point(x0-y,y0+x,color);
point(x0-x,y0+y,color);
point(x0-x,y0-y,color);
point(x0-y,y0-x,color);
point(x0+y,y0-x,color);
point(x0+x,y0-y,color);
if(err<=0){
y+=1;
err+=2*y+1;
}
if(err>0)
{
x-=1;
err-=2*x+1;
}
}
}
```

Output:



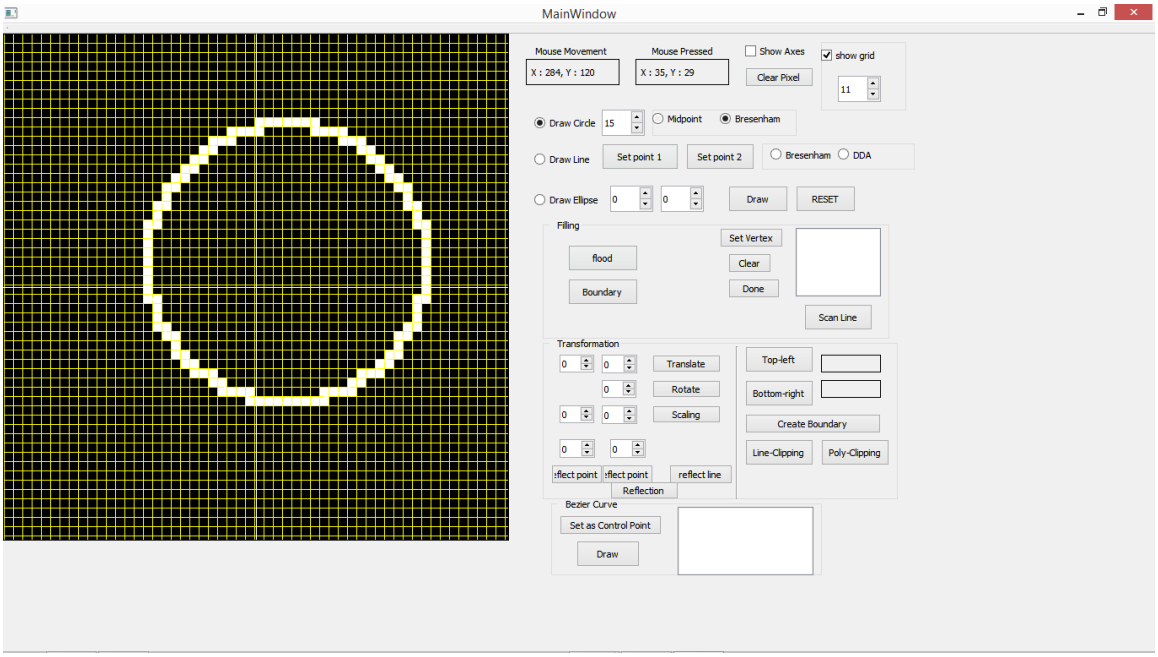
ii. bresenham circle

Code:

```
//*************BRESENHAMCIRCLE******************
elseif(ui->circlebres->isChecked()){
intx=0,y=r;
intd=3-2*r;
while(y>=x){
point(x0+x,y0+y,color);
point(x0-x,y0+y,color);
point(x0+x,y0-y,color);
point(x0-x,y0-y,color);
point(x0+y,y0+x,color);
point(x0-y,y0+x,color);
point(x0+y,y0-x,color);
point(x0-y,y0-x,color);
x++;
if(d>0){
y--;
d=d+4*(x-y)+10;
}
else
d=d+4*x+6;
point(x0+x,y0+y,color);
point(x0-x,y0+y,color);
```

```
point(x0+x,y0-y,color);
point(x0-x,y0-y,color);
point(x0+y,y0+x,color);
point(x0-y,y0+x,color);
point(x0+y,y0-x,color);
point(x0-y,y0-x,color);
}
}
clear(x0,y0);
}
```

Output:



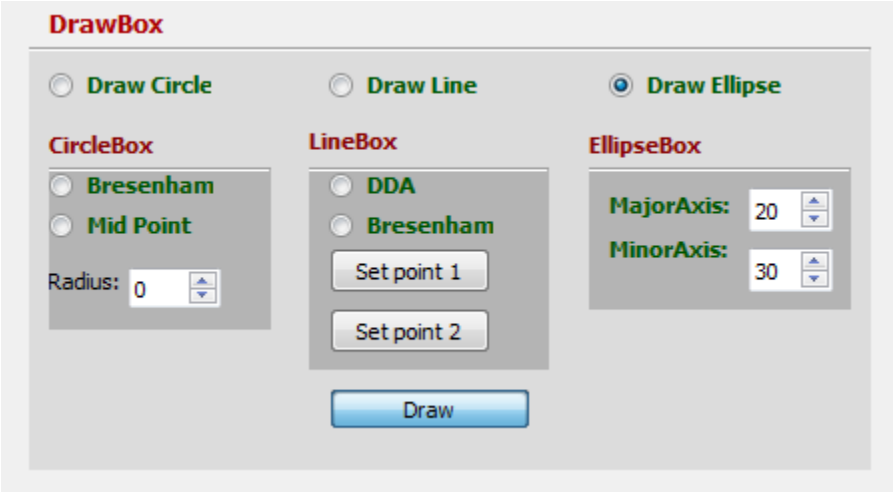
c.MidPoint ellipse drawing algorithm

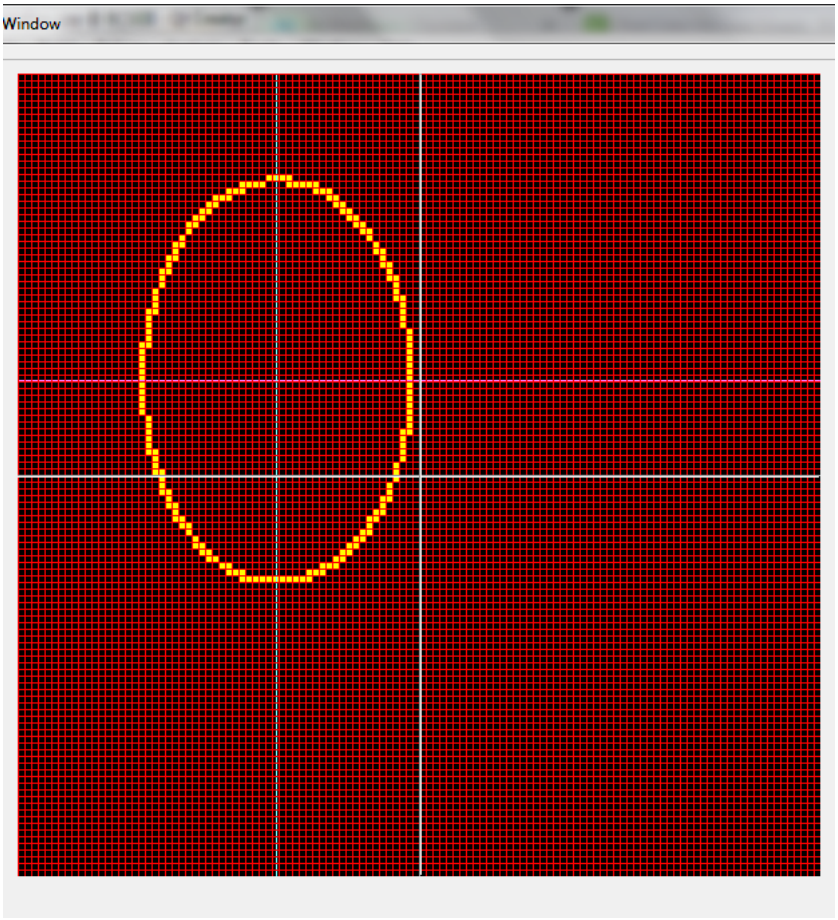
Code:

```
void MainWindow::on_Draw_clicked()
{

    if(ui->drawEllipse->isChecked()){
    /*Add ellipse centre ,major axis value ,minor axis value to list*/
        ellipseCentres.append(QPoint(ui->label->x,ui->label->y));
        ellipseMajor.append(ui->major->value());
        ellipseMinor.append(ui->minor->value());

        /*draw ellipse*/
  ellipse(ui->label->x,ui->label->y,(ui->major->value())*grid_size,(ui-
>minor->value())*grid_size);
    }

    ui->frame->setPixmap(QPixmap::fromImage(img));
}
void MainWindow::ellipse(int x0, int y0, int rx, int ry){
    int x,y;
    double p;
    p=ry*ry-rx*rx*ry+rx*rx/4;
    x=0;
    y=ry;
    erase(x0,y0);
    while(2.0*ry*ry*x <= 2.0*rx*rx*y){
        if(p < 0){
            x++;
            p = p+2*ry*ry*x+ry*ry;
        }
        else{
            x++;y--;
            p = p+2*ry*ry*x-2*rx*rx*y-ry*ry;
        }
        point(x0+x,y0+y);
        point(x0+x,y0-y);
        point(x0-x,y0+y);
        point(x0-x,y0-y);
       }

    p=ry*ry*(x+0.5)*(x+0.5)+rx*rx*(y-1)*(y-1)-rx*rx*ry*ry;
    while(y > 0){
        if(p <= 0){
            x++;
            y--;
            p = p+2*ry*ry*x-2*rx*rx*y+rx*rx;
        }
        else{
            y--;
            p = p-2*rx*rx*y+rx*rx;
        }
        point(x0+x,y0+y);
```

```
        point(x0+x,y0-y);
        point(x0-x,y0+y);
        point(x0-x,y0-y);
    }}
```

Output:



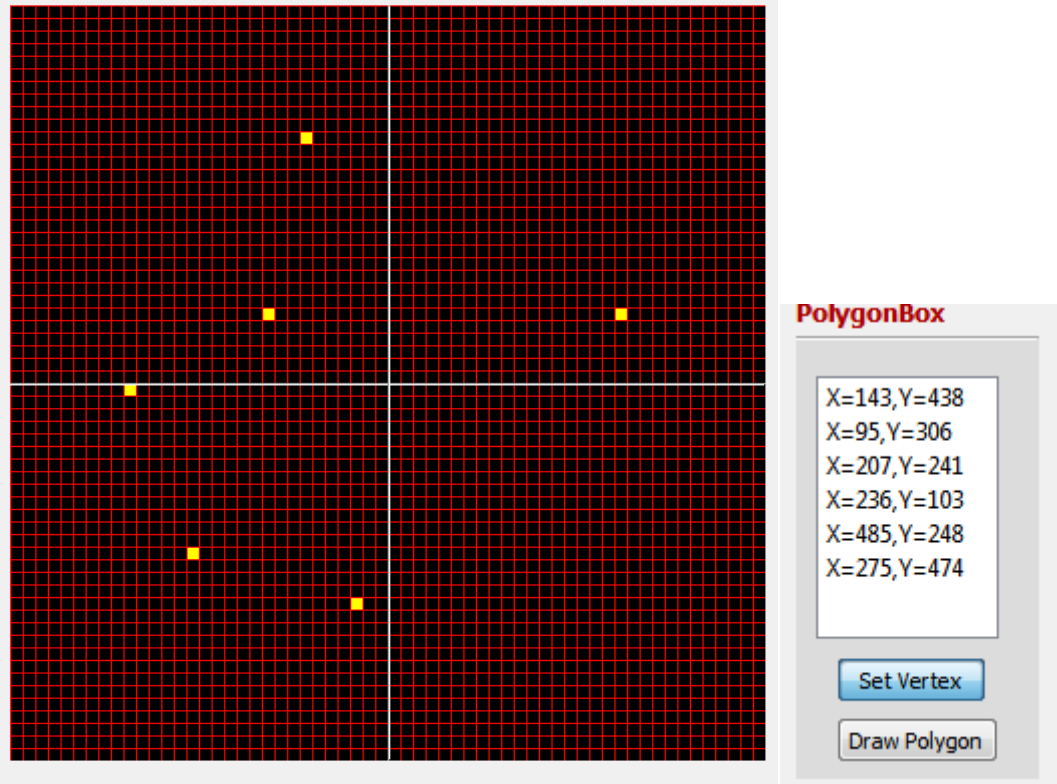


d. Polygon drawing

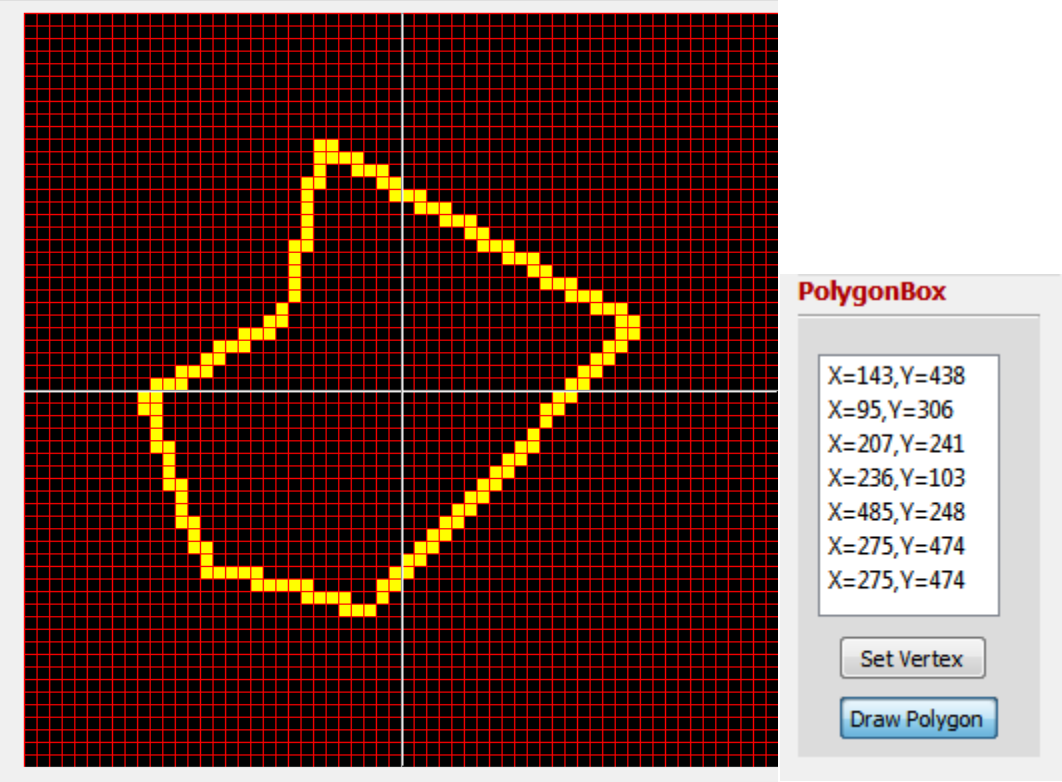
Code:

```
void MainWindow::on_setVertex_clicked()
{
    ui->polygonList->addItem(QString("X=%1,Y=%2").arg(ui->label-
>x).arg(ui->label->y));
    /*add vertex to vertex list*/
    polygonVertices.append(QPoint(ui->label->x,ui->label->y));
}


void MainWindow::on_drawPolygon_clicked()
{
    /*draw sides of the polygon in order they were clicked*/
    for(int i=1;i<=polygonVertices.size();i++){
        bresenham(polygonVertices[i-1].x(),polygonVertices[i-
1].y(),polygonVertices[i%polygonVertices.size()].x(),polygonVertices[i%
polygonVertices.size()].y());
}
}
```

Output:



Select Vertices



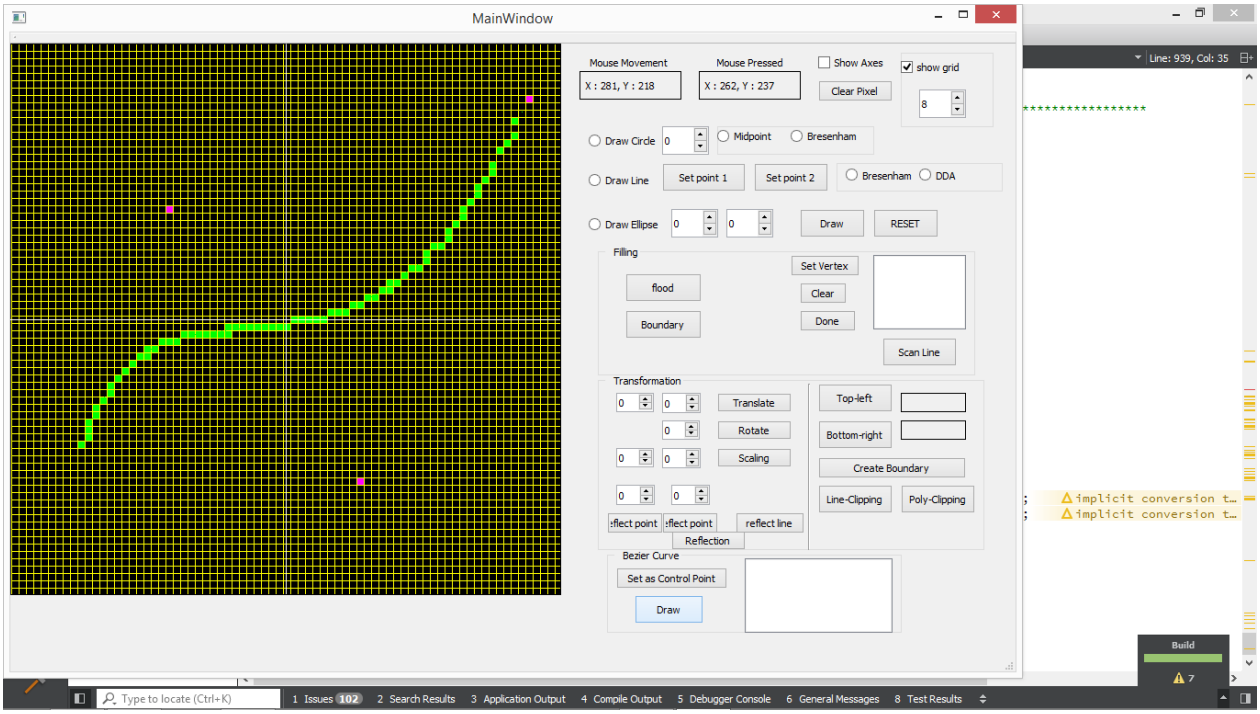
e. Bezier Curve drawing

Code:

```
intMainWindow::factorial(intn){
if(n>1)
returnn*factorial(n-1);
else
return1;
}

voidMainWindow::on_pushButton_clicked()
{
vec2dtmp(ui->frame->x,ui->frame->y);
vec2.append(tmp);
ui->pointlist->addItem(QString("X=%1,Y=%2").arg((ui->frame->x)-
300).arg(300-(ui->frame->y)));
}

voidMainWindow::on_bezier_draw_clicked(){
intn=vec2.length()-1,i;
intxt,yt;
for(doublet=0;t<=1;t=t+0.01){
xt=0;
yt=0;
```

```
i=0;
while(i<=n){
xt=xt+(factorial(n)/(factorial(n-i)*factorial(i)))*pow(1-t,n-
i)*pow(t,i)*vec2[i].x;
yt=yt+(factorial(n)/(factorial(n-i)*factorial(i)))*pow(1-t,n-
i)*pow(t,i)*vec2[i].y;
i++;
}
point(xt,yt,qRgb(0,255,0));
}
ui->pointlist->clear();
vec2.clear();
}
```

Output:



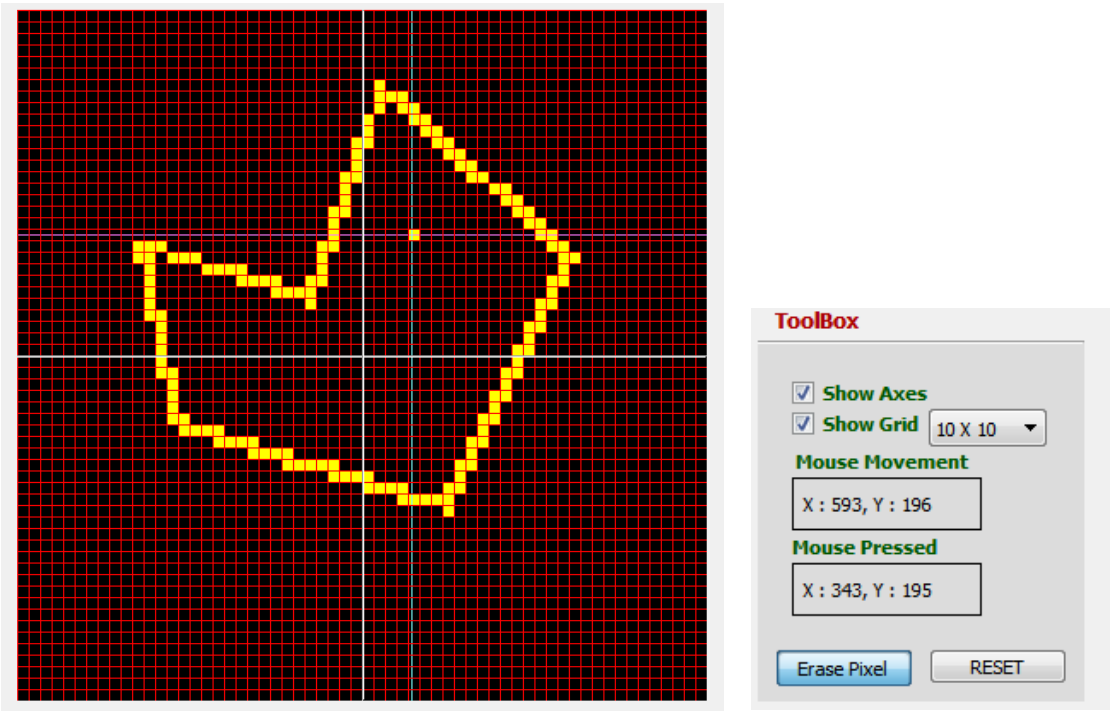
FILLING ALGORITHMS:

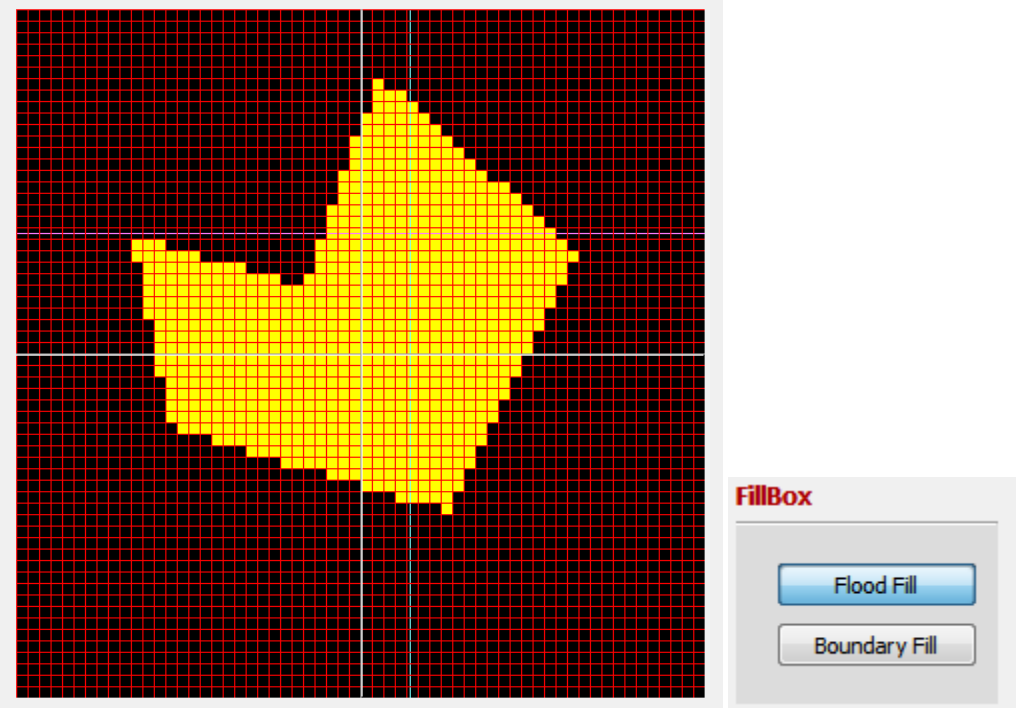
a.Flood fill

Code:

```
void MainWindow::floodFill(int x,int y,int oldcolor,int newcolor,int
gridsize){
    uint col=img.pixel(x,y);
    if(col== oldcolor)
    {
        point(x,y);
        /*recursively fill all the pixels which are background Color*/
        floodFill(x+gridsize,y,oldcolor,newcolor,gridsize);
        floodFill(x,y+gridsize,oldcolor,newcolor,gridsize);
        floodFill(x-gridsize,y,oldcolor,newcolor,gridsize);
        floodFill(x,y-gridsize,oldcolor,newcolor,gridsize);
    }
}
void MainWindow::on_Flood_clicked()
{
    floodFill(ui->label->x,ui->label-
>y,qRgb(0,0,0),qRgb(255,255,255),grid_size);
}
```

Output:



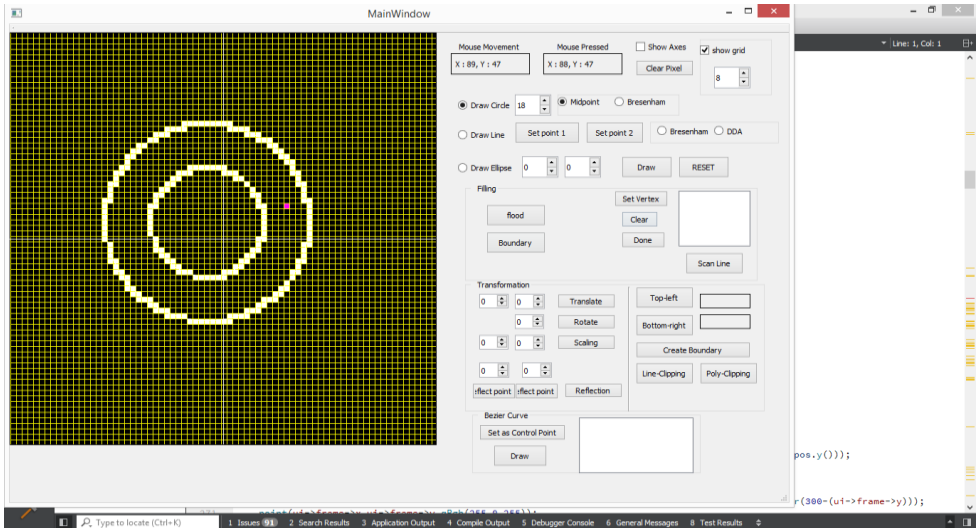


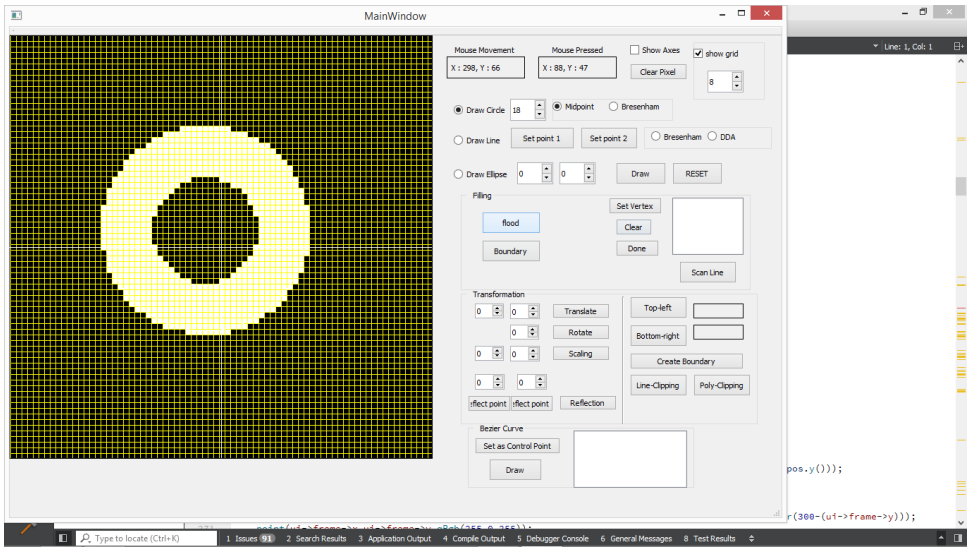
b.Boundary fill

Code:

```
voidMainWindow::boundaryfilled(intx,inty,uintold,uintnewc)
{
intr=ui->gridsize->value();
uintcolor=img.pixel(x,y);
uintboundary=qRgb(255,255,255);
if(color!=boundary){
point(x,y,newc);
boundaryfilled(x+r,y,old,newc);
boundaryfilled(x,y+r,old,newc);
boundaryfilled(x-r,y,old,newc);
boundaryfilled(x,y-r,old,newc);
}
}
voidMainWindow::on_boundaryfil_clicked()
{
boundaryfilled(ui->frame->x,ui->frame->y,qRgb(0,0,0),qRgb(0,255,255));
ui->frame->setPixmap(QPixmap::fromImage(img));
}
```

Output:


MainWindow
Mouse Movement
X : 89, Y : 47
Mouse Pressed
X : 88, Y : 47
Show Axes
show grid
Clear Pixel
Draw Circle
Midpoint
Bresenham
Draw Line
Set point 1
Set point 2
DDA
Draw Ellipse
Draw
RESET
Filling
flood
Boundary
Set Vertex
Clear
Done
Scan Line
Transformation
Translate
Rotate
Scaling
Reflection
Top-left
Bottom-right
Create Boundary
Line-Clipping
Poly-Clipping
Bezier Curve
Set as Control Point
Draw

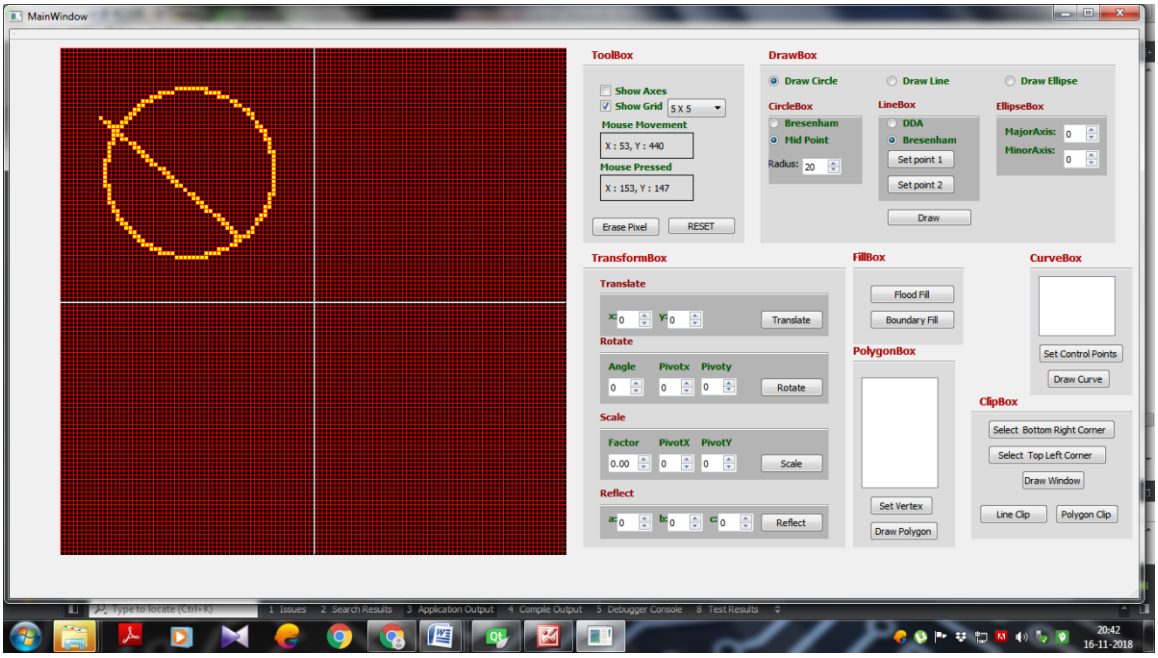

MainWindow
Mouse Movement
X : 298, Y : 66
Mouse Pressed
X : 88, Y : 47
Show Axes
show grid
Clear Pixel
Draw Circle
Midpoint
Bresenham
Draw Line
Set point 1
Set point 2
DDA
Draw Ellipse
Draw
RESET
Filling
flood
Boundary
Set Vertex
Clear
Done
Scan Line
Transformation
Translate
Rotate
Scaling
Reflection
Top-left
Bottom-right
Create Boundary
Line-Clipping
Poly-Clipping
Bezier Curve
Set as Control Point
Draw

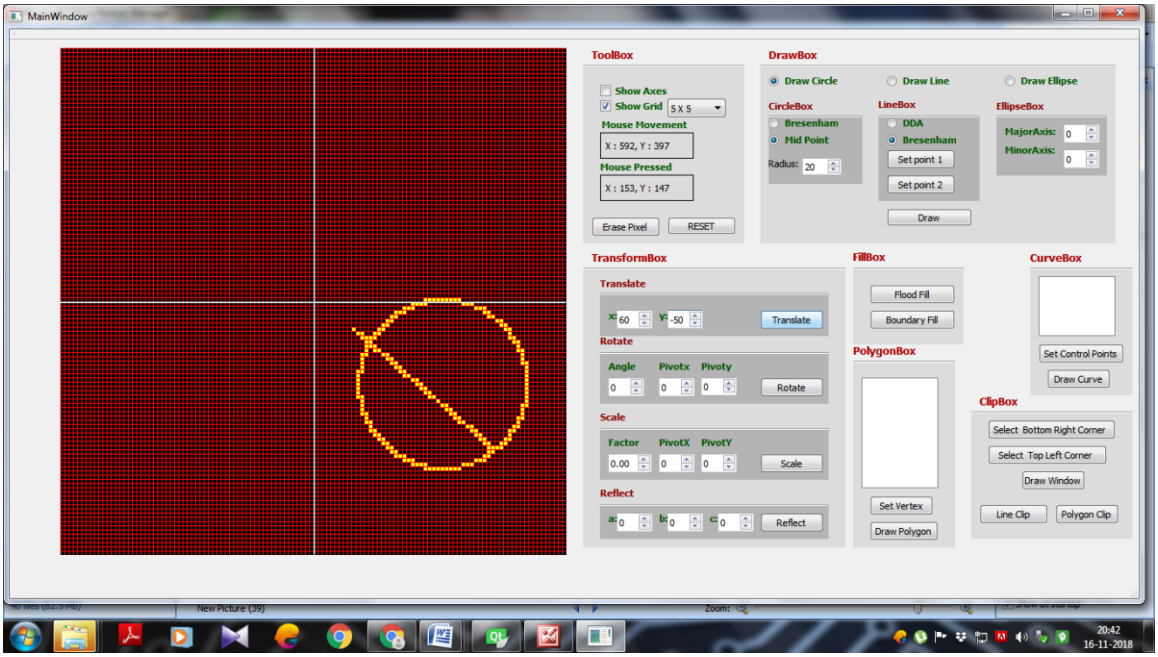
TRANSFORMATIONS:

a.Translation

Code:

```
void MainWindow::on_translate_clicked()
{
    /*erase screen*/
    on_pushButton_clicked();
    int xtrans=ui->xTrans->value();
    int ytrans=-ui->yTrans->value();

    /*shift line endpoints*/
    for(int i=0;i<lineEp1.length();i++){
        int xOld1=lineEp1[i].x();
        int yOld1=lineEp1[i].y();
        //calculate new endpoint values
        int xNew1=xOld1+xtrans*grid_size;
        int yNew1=yOld1+ytrans*grid_size;
        int xOld2=lineEp2[i].x();
        int yOld2=lineEp2[i].y();
        int xNew2=xOld2+xtrans*grid_size;
        int yNew2=yOld2+ytrans*grid_size;

        bresenham(xNew1,yNew1,xNew2,yNew2);

        //add back new endpoint values to list
        lineEp1[i].setX(xNew1);
        lineEp1[i].setY(yNew1);
        lineEp2[i].setX(xNew2);
        lineEp2[i].setY(yNew2);
    }
   /*shift circleCentres*/
    for(int i=0;i<sphereCentres.length();i++){
        int xOld1=sphereCentres[i].x();
        int yOld1=sphereCentres[i].y();
        //calculate new centre values
        int xNew1=xOld1+xtrans*grid_size;
        int yNew1=yOld1+ytrans*grid_size;
        circleBres(xNew1,yNew1,radiusList[i]);
        //add new centres back to list
        sphereCentres[i].setX(xNew1);
        sphereCentres[i].setY(yNew1);
    }
}
```



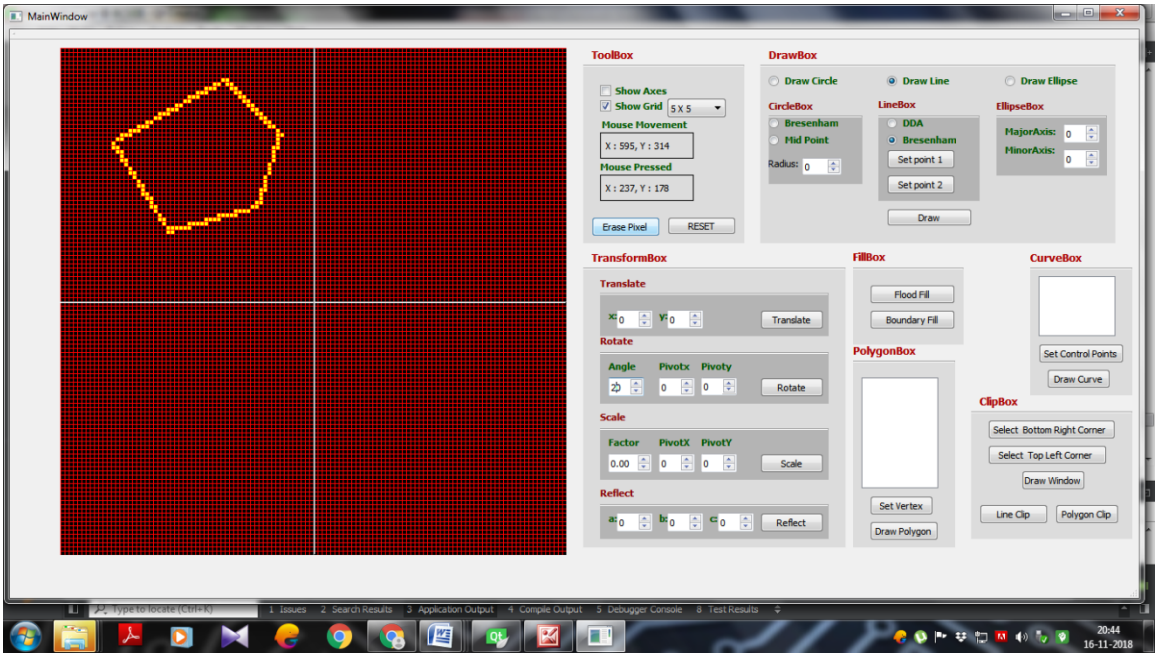


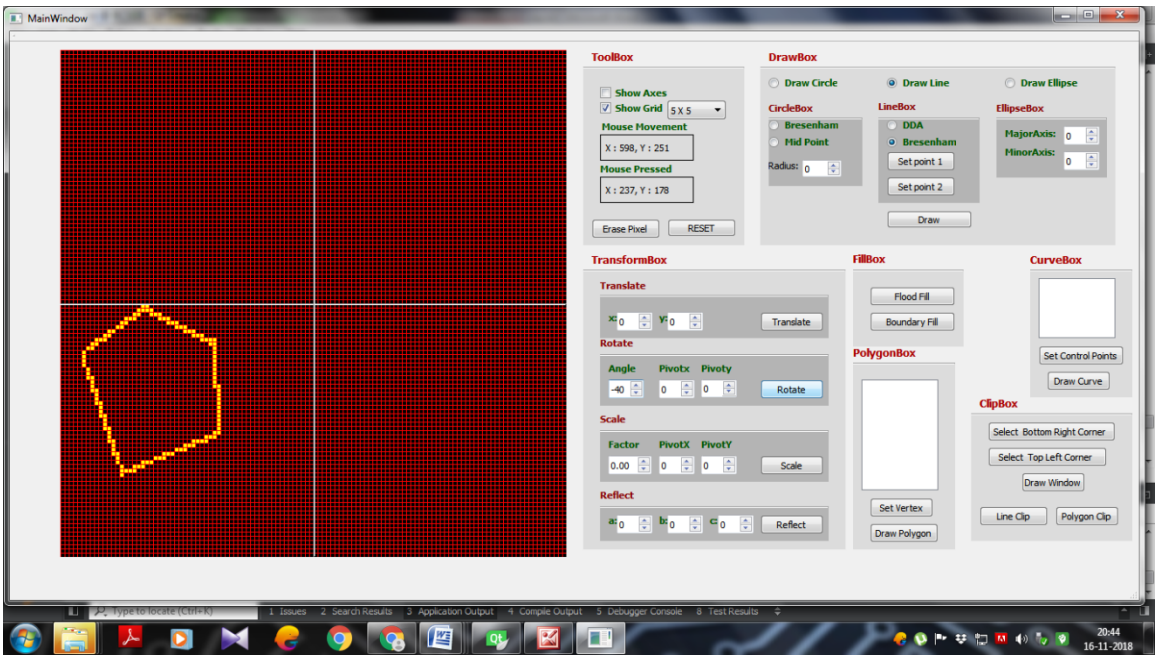
Translating 60 units and -50 units in x amd y direction respectively

b.Rotation

Code:

```
void MainWindow::on_rotate_clicked()
{
    on_pushButton_clicked();
    double rad=(ui->degrees->value()*2*M_PI)/180;

    int xr=ui->pivotx->value()*grid_size+300;
    int yr=-(ui->pivoty->value()*grid_size)+300;

    for(int i=0;i<lineEp1.length();i++){
        int xOld1=lineEp1[i].x();
        int yOld1=lineEp1[i].y();
        int xNew1=xr+(int)((double)(xOld1-xr)*qCos(rad)-(double)(yOld1-
yr)*qSin(rad));
        int yNew1=yr+(int)((double)(xOld1-xr)*qSin(rad)+(double)(yOld1-
yr)*qCos(rad));

        int xOld2=lineEp2[i].x();
        int yOld2=lineEp2[i].y();
        int xNew2=xr+(int)((double)(xOld2-xr)*qCos(rad)-(double)(yOld2-
yr)*qSin(rad));
        int yNew2=yr+(int)((double)(xOld2-xr)*qSin(rad)+(double)(yOld2-
yr)*qCos(rad));
        //erasebresenham(xOld1,yOld1,xOld2,yOld2);
        bresenham(xNew1,yNew1,xNew2,yNew2);

        lineEp1[i].setX(xNew1);
        lineEp1[i].setY(yNew1);
        lineEp2[i].setX(xNew2);
        lineEp2[i].setY(yNew2);
    }

    for(int i=0;i<sphereCentres.length();i++){
        int xOld1=sphereCentres[i].x();
        int yOld1=sphereCentres[i].y();
        int xNew1=xr+(int)((double)(xOld1-xr)*qCos(rad)-(double)(yOld1-
yr)*qSin(rad));
        int yNew1=yr+(int)((double)(xOld1-xr)*qSin(rad)+(double)(yOld1-
yr)*qCos(rad));
        //eraseCricleBres(xOld1,yOld1,radiusList[i]);
        circleBres(xNew1,yNew1,radiusList[i]);
        sphereCentres[i].setX(xNew1);
        sphereCentres[i].setY(yNew1);
    }
}
```





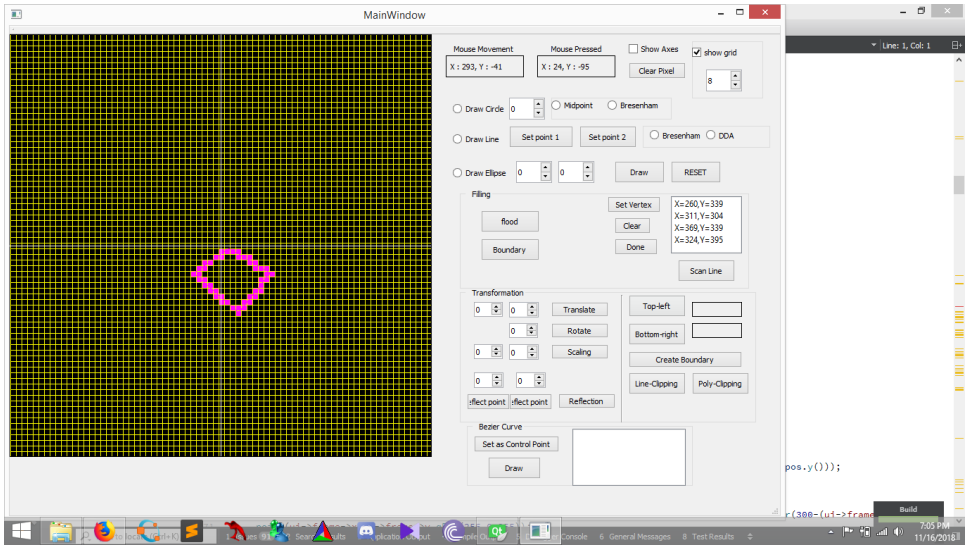
40 degree anticlockwise rotation about origin

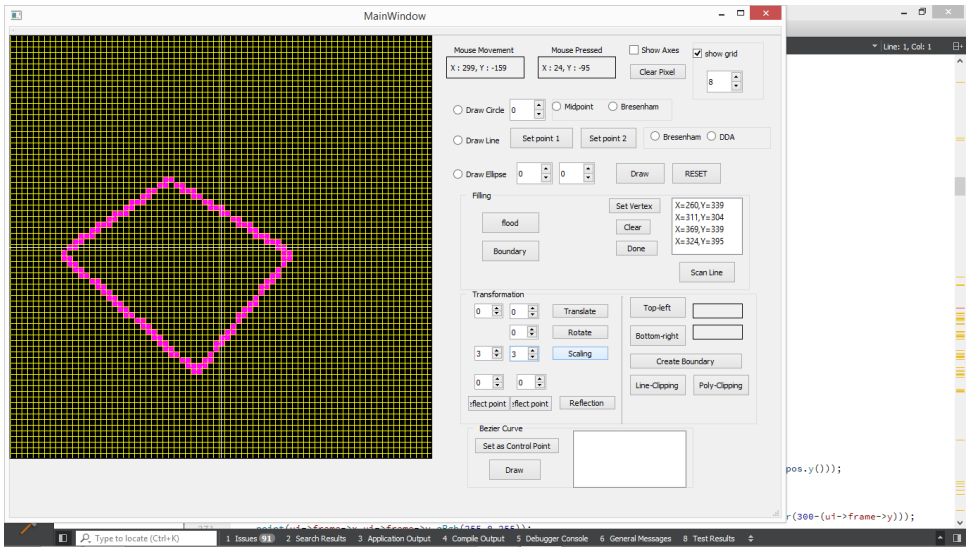
c.Scaling

Code:

```
voidMainWindow::on_scale_clicked(){
intk=ui->gridsize->value();
floatsx=ui->x_scale->value();
floatsy=ui->y_scale->value();
intpx=ui->pivot_x->value()*k+k/2+350;
intpy=350-ui->pivot_y->value()*k+k/2;
floattmat[3][3]={{sx,0,0},{0,sy,0},{0,0,1}};
intptmat[3][1];
inti,l;
for(i=0;i<3;i++){
ptmat[i][0]=0;
}
floatresmat[3][1];
l=vertex_list.size();
on_reset_clicked();
on_gridsize_valueChanged(k);
on_show_axes_clicked();
for(i=0;i<l;i++){
ptmat[0][0]=vertex_list[i].first-px;
ptmat[1][0]=py-vertex_list[i].second;
ptmat[2][0]=1;
mat_mult(tmat,ptmat,resmat);
vertex_list[i].first=(int)(resmat[0][0]+px);
vertex_list[i].second=(int)(py-resmat[1][0]);
if(i>0){
QPointpx,py;
px.setX(vertex_list[i-1].first);
px.setY(vertex_list[i-1].second);
py.setX(vertex_list[i].first);
py.setY(vertex_list[i].second);
DDA(px.x(),px.y(),py.x(),py.y(),qRgb(255,0,255));
}
}
DDA(vertex_list[0].first,vertex_list[0].second,vertex_list[l-
1].first,vertex_list[l-1].second,qRgb(255,0,255));
}
```

Output:


MainWindow
Mouse Movement
X : 293, Y : -41
Mouse Pressed
X : 24, Y : -95
Show Axes
show grid
Clear Pixel
Draw Circle
Midpoint
Bresenham
Draw Line
Set point 1
Set point 2
Bresenham
DDA
Draw Ellipse
Draw
RESET
Filling
flood
Boundary
Set Vertex
Clear
Done
X=260,Y=339
X=311,Y=304
X=369,Y=339
X=324,Y=395
Scan Line
Transformation
Translate
Rotate
Scaling
Reflection
Top-left
Bottom-right
Create Boundary
Line-Clipping
Poly-Clipping
Bezier Curve
Set as Control Point
Draw

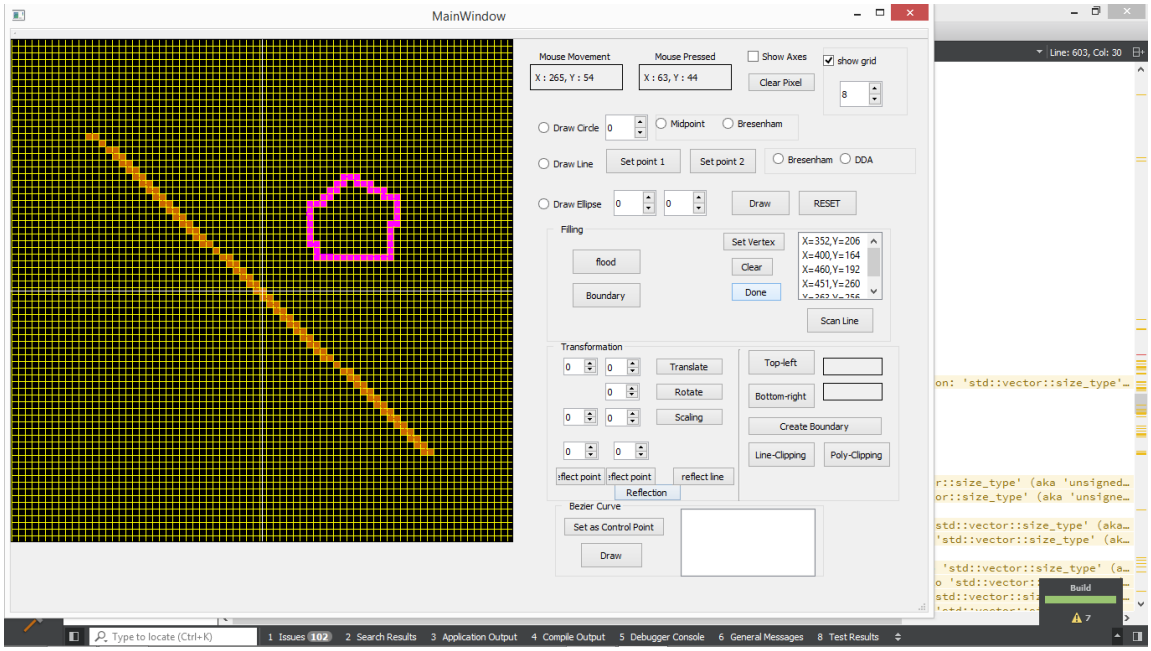

MainWindow
Mouse Movement
X : 299, Y : -159
Mouse Pressed
X : 24, Y : -95
Show Axes
show grid
Clear Pixel
Draw Circle
Midpoint
Bresenham
Draw Line
Set point 1
Set point 2
Bresenham
DDA
Draw Ellipse
Draw
RESET
Filling
flood
Boundary
Set Vertex
Clear
Done
X=260,Y=339
X=311,Y=304
X=369,Y=339
X=324,Y=395
Scan Line
Transformation
Translate
Rotate
Scaling
Reflection
Top-left
Bottom-right
Create Boundary
Line-Clipping
Poly-Clipping
Bezier Curve
Set as Control Point
Draw

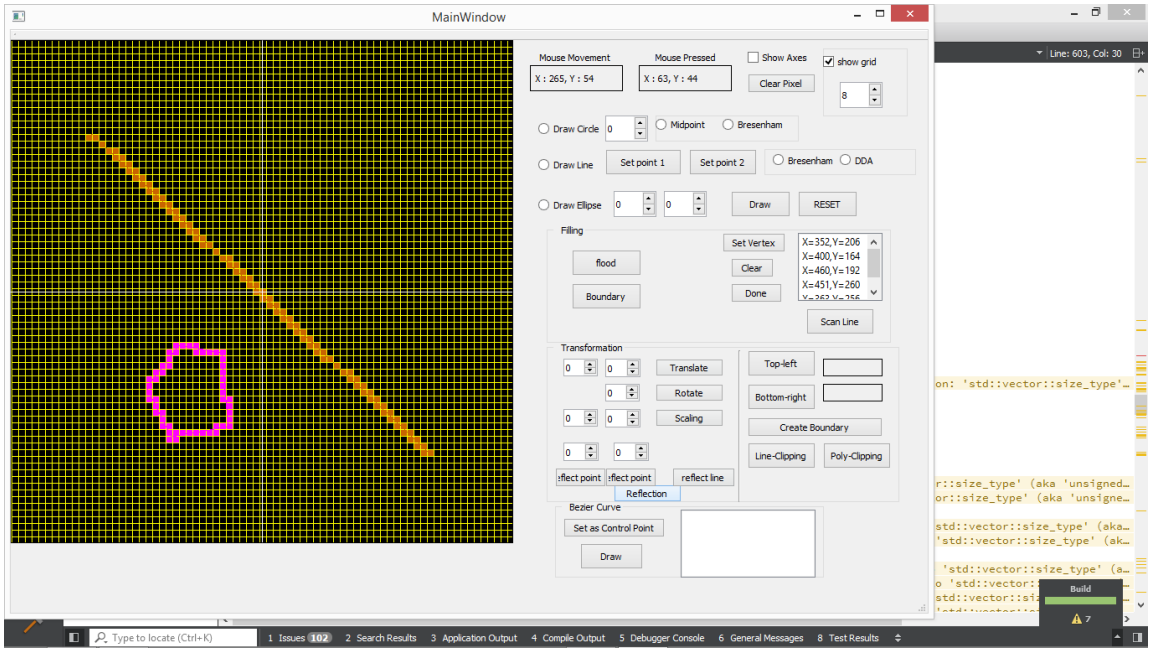
d.Reflection

Code:

```
voidMainWindow::on_rfp1_clicked()
{
ref1.setX(ui->frame->x);
ref1.setY(ui->frame->y);
}

voidMainWindow::on_rfp2_clicked()
{
ref2.setX(ui->frame->x);
ref2.setY(ui->frame->y);
//DDA(ref1.x(),ref1.y(),ref2.x(),ref2.y(),qRgb(200,100,0));
}

voidMainWindow::on_drawrefline_clicked()
{
DDA(ref1.x(),ref1.y(),ref2.x(),ref2.y(),qRgb(200,100,0));
}

voidMainWindow::on_reflect_clicked()
{
intk=ui->gridsize->value();
inta=(ref2.x()-ref1.y());
intb=(ref1.x()-ref2.x());
intc=-a*ref1.x()-b*ref1.y();
intnum,den=a*a+b*b,l=vertex_list.size(),x1,y1;
on_reset_clicked();
on_gridsize_valueChanged(k);
on_show_axes_clicked();
on_drawrefline_clicked();
QPointpx,py;
for(inti=0;i<l;i++){
x1=vertex_list[i].first;
y1=vertex_list[i].second;
num=-2*(a*x1+b*y1+c);
vertex_list[i].first=x1+a*num/den;
vertex_list[i].second=y1+b*num/den;
if(i>0){
px.setX(vertex_list[i-1].first);
px.setY(vertex_list[i-1].second);
py.setX(vertex_list[i].first);
py.setY(vertex_list[i].second);
DDA(px.x(),px.y(),py.x(),py.y(),qRgb(255,0,255));
}
}
DDA(vertex_list[0].first,vertex_list[0].second,vertex_list[l-
1].first,vertex_list[l-1].second,qRgb(255,0,255));
}
```

Output:


MainWindow
Mouse Movement
X : 265, Y : 54
Mouse Pressed
X : 63, Y : 44
Show Axes
show grid
Clear Pixel
Draw Circle
Midpoint
Bresenham
Draw Line
Set point 1
Set point 2
Bresenham
DDA
Draw Ellipse
Draw
RESET
Filling
flood
Boundary
Set Vertex
Clear
Done
X=352,Y=206
X=400,Y=164
X=460,Y=192
X=451,Y=260
Scan Line
Transformation
Translate
Rotate
Scaling
reflect line
Reflection
Top-left
Bottom-right
Create Boundary
Line-Clipping
Poly-Clipping
Bezier Curve
Set as Control Point
Draw
Line: 603, Col: 30
Build
Type to locate (Ctrl+K)
1 Issues 102
2 Search Results
3 Application Output
4 Compile Output
5 Debugger Console
6 General Messages
8 Test Results


MainWindow
Mouse Movement
X : 265, Y : 54
Mouse Pressed
X : 63, Y : 44
Show Axes
show grid
Clear Pixel
Draw Circle
Midpoint
Bresenham
Draw Line
Set point 1
Set point 2
Bresenham
DDA
Draw Ellipse
Draw
RESET
Filling
flood
Boundary
Set Vertex
Clear
Done
X=352,Y=206
X=400,Y=164
X=460,Y=192
X=451,Y=260
Scan Line
Transformation
Translate
Rotate
Scaling
reflect line
Reflection
Top-left
Bottom-right
Create Boundary
Line-Clipping
Poly-Clipping
Bezier Curve
Set as Control Point
Draw
Line: 603, Col: 30
Build
Type to locate (Ctrl+K)
1 Issues 102
2 Search Results
3 Application Output
4 Compile Output
5 Debugger Console
6 General Messages
8 Test Results

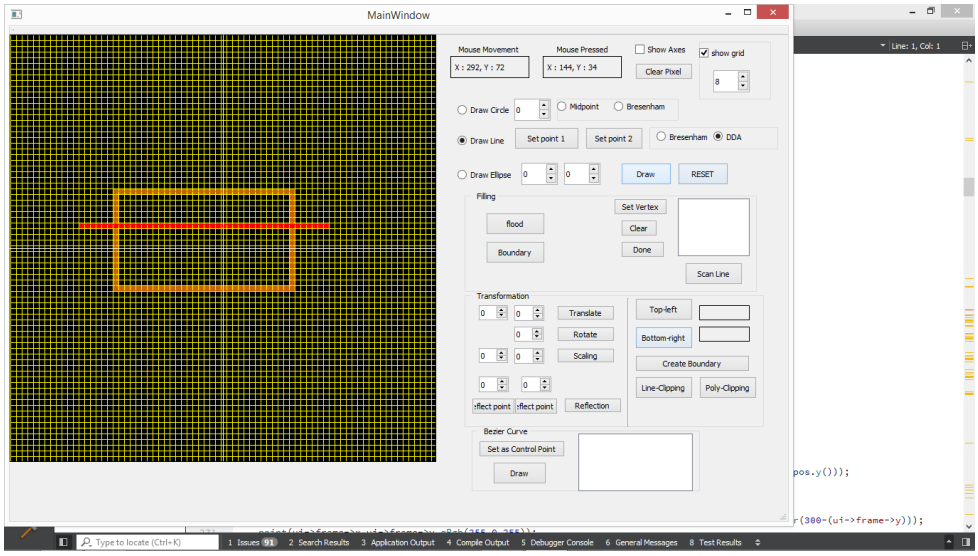
CLIPPING:

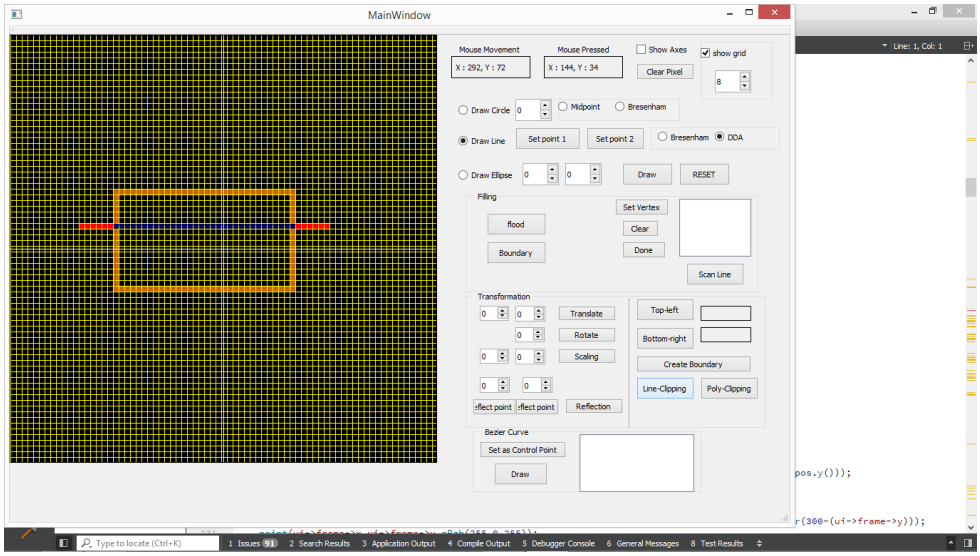
a.Cohen Sutherland Line Clipping

```
unsignedcharMainWindow::encode(QPointpt,QPointwinmin,QPointwinmax){
unsignedcharcode=0x00;
if(pt.x()<winmin.x())code=code|LEFT_EDGE;
if(pt.x()>winmax.x())code=code|RIGHT_EDGE;
if(pt.y()<winmin.y())code=code|BOTTOM_EDGE;
if(pt.y()>winmax.y())code=code|TOP_EDGE;
return(code);
}
voidMainWindow::swapPts(QPoint*point1,QPoint*point2){
QPointtmp;
tmp=*point1;
*point1=*point2;
*point2=tmp;
}
voidMainWindow::swapCodes(unsignedchar*c1,unsignedchar*c2){
unsignedchartmp;
tmp=*c1;
*c1=*c2;
*c2=tmp;
}
voidMainWindow::cohenSutherlandClip(QPointwinmin,QPointwinmax,QPointpp1
,QPointpp2){
unsignedcharcode1,code2;
intdone=FALSE,draw=FALSE,xx,yy;
floatm;
while(!done){
code1=encode(pp1,winmin,winmax);
code2=encode(pp2,winmin,winmax);
if(ACCEPT(code1,code2)){
done=TRUE;
draw=TRUE;
}
else{
if(REJECT(code1,code2))done=TRUE;
elseif(INSIDE(code1)){
swapPts(&pp1,&pp2);
swapCodes(&code1,&code2);
}
if(pp2.x()!=pp1.x())m=(pp2.y()-pp1.y())/(pp2.x()-pp1.x());
if(code1&LEFT_EDGE){
yy=(int)(pp1.y()+(winmin.x()-pp1.x())*m);
pp1.setY(yy);
pp1.setX(winmin.x());
}
elseif(code1&RIGHT_EDGE){
yy=(int)(pp1.y()+(winmax.x()-pp1.x())*m);
pp1.setY(yy);
pp1.setX(winmax.x());
}
```

```
elseif(code1&BOTTOM_EDGE){
if(pp2.x()!=pp1.x()){
xx=(winmin.y()-pp1.y())/m;
pp1.setX(pp1.x()+xx);
pp1.setY(winmin.y());
}
}
elseif(code1&TOP_EDGE){
if(pp2.x()!=pp1.x()){
xx=(winmax.y()-pp1.y())/m;
pp1.setX(pp1.x()+xx);
pp1.setY(winmax.y());
}
}
}
}
if(draw){DDA(ROUND(pp1.x()),ROUND(pp1.y()),ROUND(pp2.x()),ROUND(pp2.y()
),qRgb(0,0,100));
}
}
voidMainWindow::on_lineclip_clicked(){
//DDA(p1.x(),p1.y(),p2.x(),p2.y(),qRgb((0,0,0));
cohenSutherlandClip(bp1,bp2,p1,p2);
```

Output:





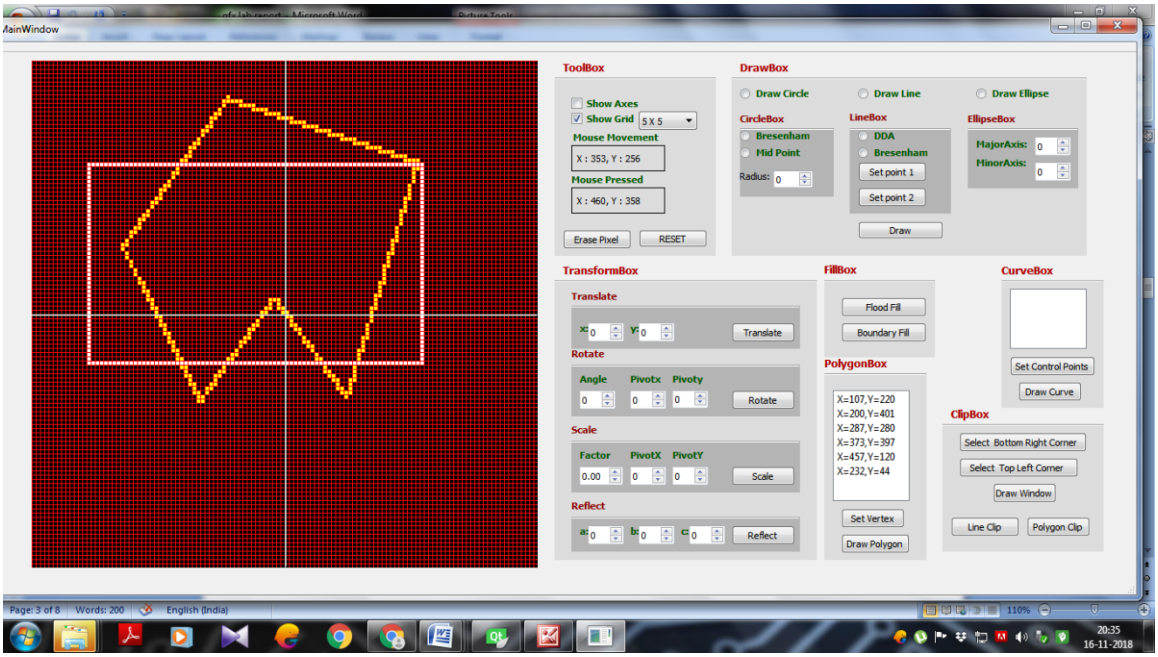
b. Sutherland Hodge Polygon Clipping

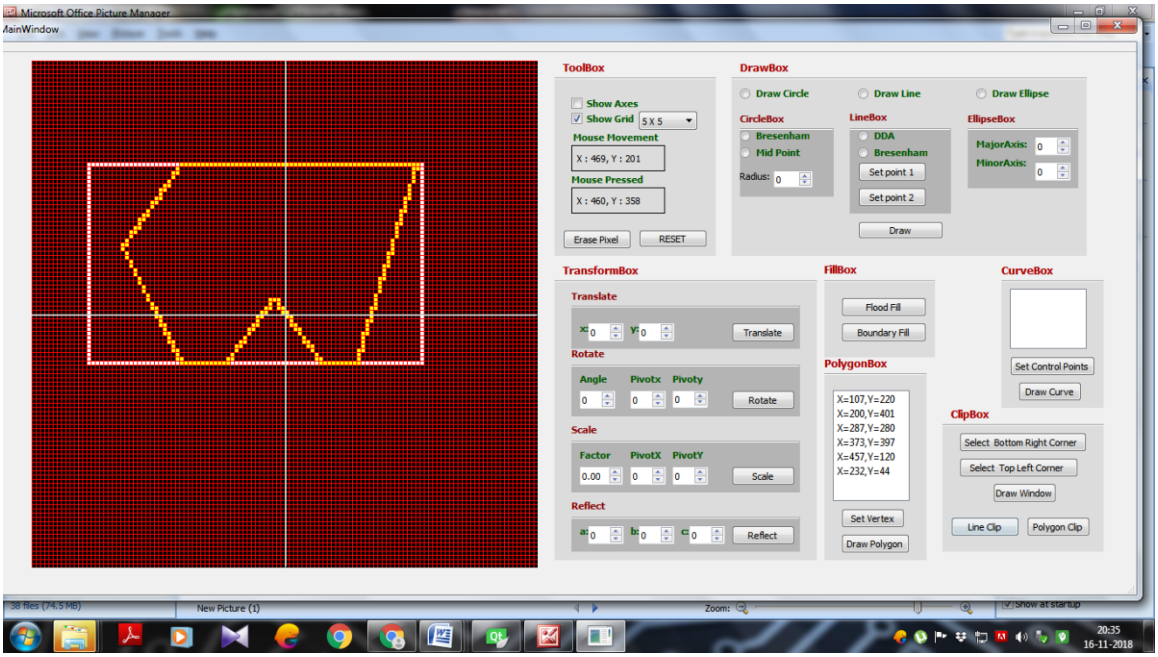
```
int MainWindow::x_intersect(int x1, int y1, int x2, int y2,
                int x3, int y3, int x4, int y4)
{
    int num = (x1*y2 - y1*x2) * (x3-x4) -
              (x1-x2) * (x3*y4 - y3*x4);
    int den = (x1-x2) * (y3-y4) - (y1-y2) * (x3-x4);
    return num/den;
}

// Returns y-value of point of intersectipn of
// two lines
int  MainWindow::y_intersect(int x1, int y1, int x2, int y2,
                int x3, int y3, int x4, int y4)
{
    int num = (x1*y2 - y1*x2) * (y3-y4) -
              (y1-y2) * (x3*y4 - y3*x4);
    int den = (x1-x2) * (y3-y4) - (y1-y2) * (x3-x4);
    return num/den;
}

// This functions clips all the edges w.r.t one clip
// edge of clipping area
void  MainWindow::clip(int x1, int y1, int x2, int y2)
{
     QList<QPoint> tempClip;
    // (ix,iy),(kx,ky) are the co-ordinate values of
    // the points
    for (int i = 0; i < polygonVertices.size(); i++)
    {
        // i and k form a line in polygon
        int k = (i+1) % polygonVertices.size();
```

```
            int ix = polygonVertices[i].x(), iy = polygonVertices[i].y();
            int kx = polygonVertices[k].x(), ky = polygonVertices[k].y();

            // Calculating position of first point
            // w.r.t. clipper line
            int i_pos = (x2-x1) * (iy-y1) - (y2-y1) * (ix-x1);

            // Calculating position of second point
            // w.r.t. clipper line
            int k_pos = (x2-x1) * (ky-y1) - (y2-y1) * (kx-x1);

            // Case 1 : When both points are inside
            if (i_pos < 0  && k_pos < 0)
            {
                //Only second point is added
                tempClip.append(QPoint(kx,ky));
            }

            // Case 2: When only first point is outside
            else if (i_pos >= 0  && k_pos < 0)
            {
                // Point of intersection with edge
                // and the second point is added
                tempClip.append(QPoint(x_intersect(x1,
                                                   y1, x2, y2, ix, iy, kx,
ky), y_intersect(x1,

y1, x2, y2, ix, iy, kx, ky)));


                tempClip.append(QPoint(kx,ky));
            }

            // Case 3: When only second point is outside
            else if (i_pos < 0  && k_pos >= 0)
            {
                //Only point of intersection with edge is added
                tempClip.append(QPoint(x_intersect(x1,
                                                   y1, x2, y2, ix, iy, kx,
ky), y_intersect(x1,

y1, x2, y2, ix, iy, kx, ky)));
            }

            // Case 4: When both points are outside
            else
            {
                //No points are added
            }
        }

        // Copying new points into original array
        // and changing the no. of vertices
        polygonVertices=tempClip;
    }

    // Implements Sutherland-Hodgman algorithm
```

```
void  MainWindow::suthHodgClip(int clipper_points[][2], int
clipper_size)
{
    //i and k are two consecutive indexes
    for (int i=0; i<clipper_size; i++)
    {
        int k = (i+1) % clipper_size;

        // We pass the current array of vertices, it's size
        // and the end points of the selected clipper line
        clip(clipper_points[i][0],
             clipper_points[i][1], clipper_points[k][0],
             clipper_points[k][1]);
    }

}
```

Output:




Microsoft Office Picture Manager
MainWindow
ToolBox
Show Axes
Show Grid
5 X 5
Mouse Movement
X : 469, Y : 201
Mouse Pressed
X : 460, Y : 358
Erase Pixel
RESET
DrawBox
Draw Circle
Draw Line
Draw Ellipse
CircleBox
Bresenham
Mid Point
Radius:
LineBox
DDA
Bresenham
Set point 1
Set point 2
Draw
EllipseBox
MajorAxis:
MinorAxis:
TransformBox
Translate
Translate
Rotate
Angle
Pivotx
Pivoty
Rotate
Scale
Factor
PivotX
PivotY
Scale
Reflect
Reflect
FillBox
Flood Fill
Boundary Fill
PolygonBox
X=107,Y=220
X=200,Y=401
X=287,Y=280
X=373,Y=397
X=457,Y=120
X=232,Y=44
Set Vertex
Draw Polygon
CurveBox
Set Control Points
Draw Curve
ClipBox
Select Bottom Right Corner
Select Top Left Corner
Draw Window
Line Clip
Polygon Clip
New Picture (1)
Zoom:
20:35
16-11-2018

ACKNOWLEDGEMENT

I have taken efforts in this project. However, it would not have been possible without the kind support and help of many individuals. I would like to extend my sincere thanks to all of them.

I am highly indebted to Mr. Nirmal Das for his guidance and constant supervision as well as for providing necessary information regarding the project & also for his support in completing the project.

My thanks and appreciations also go to my colleague in developing the project and people who have willingly helped me out with their abilities.

My thanks to Dr. Subhadip Basu for giving me the chance to do this project.

BIBLIOGRAPHY

- Computer Graphics ,C version –Hearn ,Baker
- https://math.stackexchange.com/questions/1013230/how-to-find-coordinates-of-reflected-point
- https://www.geeksforgeeks.org/line-clipping-set-1-cohen-sutherland-algorithm/
- https://www.geeksforgeeks.org/polygon-clipping-sutherland-hodgman-algorithm-please-change-bmp-images-jpeg-png/