Name – Soumyadeb Misra

Roll – 002010501088

BCSE IIIrd year

SUB – Computer Graphics Lab Report

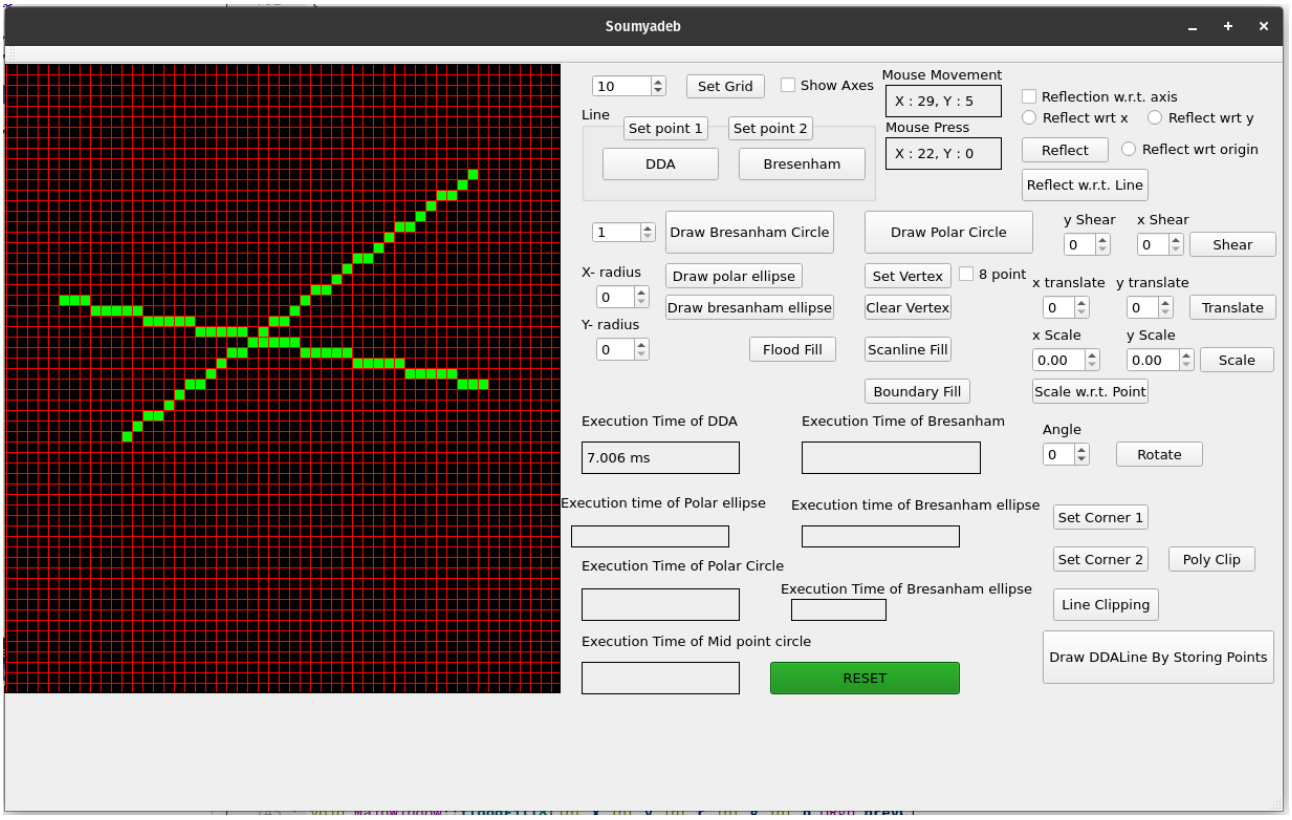
Assignment - Implement a line drawing algorithm to draw lines between two end points in the raster grid using, a) DDA b) Bresenham's line drawing algorithm. Show execution times for each algorithm in ms.

Code Snippet for DDA line :

```
void MainWindow::DDAline(int r, int g, int b) {
    double x1 = p1.x()/gridsize;
    double y1 = p1.y()/gridsize;
    double x2 = p2.x()/gridsize;
    double y2 = p2.y()/gridsize;
    double xinc, yinc, step;
    double slope = fabs(y2-y1)/fabs(x2-x1);
    if(slope  <= 1.00) {
        xinc = 1;
        yinc = slope;
        step = fabs(x2 - x1);
    } else {
        xinc = 1/slope;
        yinc = 1;
        step = fabs(y2 - y1);
    }
    if(x1 > x2) xinc *= -1;
    if(y1 > y2) yinc *= -1;
    double x = x1*gridsize + gridsize/2;
    double y = y1*gridsize + gridsize/2;
    for(int i=0;i<=step;i++) {
        point(x,y,r,g,b);
        x += xinc * gridsize;
        y += yinc * gridsize;
    }
}
```

Note: 1. The parameter r, g, b refer to ‘rgb’ colour code.
2. p1 and p2 are two ends of the line to be drawn.

Output:



Code Snippet for Bresenham line :

```
void MainWindow::on_bresenhamLine_clicked()
{
    int x1 = p1.x()/gridsize;
    int y1 = p1.y()/gridsize;
    int x2 = p2.x()/gridsize;
    int y2 = p2.y()/gridsize;
    int dx = fabs(x2 - x1);
    int dy = fabs(y2 - y1);

    int xinc = (x1 > x2 ? -1 : 1);
    int yinc = (y1 > y2 ? -1 : 1);
    bool flag = 1;
    int x = x1*gridsize + gridsize/2;
    int y = y1*gridsize + gridsize/2;
    if(dy > dx) { // if slope > 1, then swap
        swap(dx,dy);
        swap(x,y);
        swap(xinc,yinc);
        flag = 0;
    }
    int decision = 2*dy - dx;
    int step = dx;
    for(int i=0;i<=step;i++) {
        if(flag) point(x,y);
        else point(y,x);
        if(decision < 0) {
            x += xinc*gridsize;
```

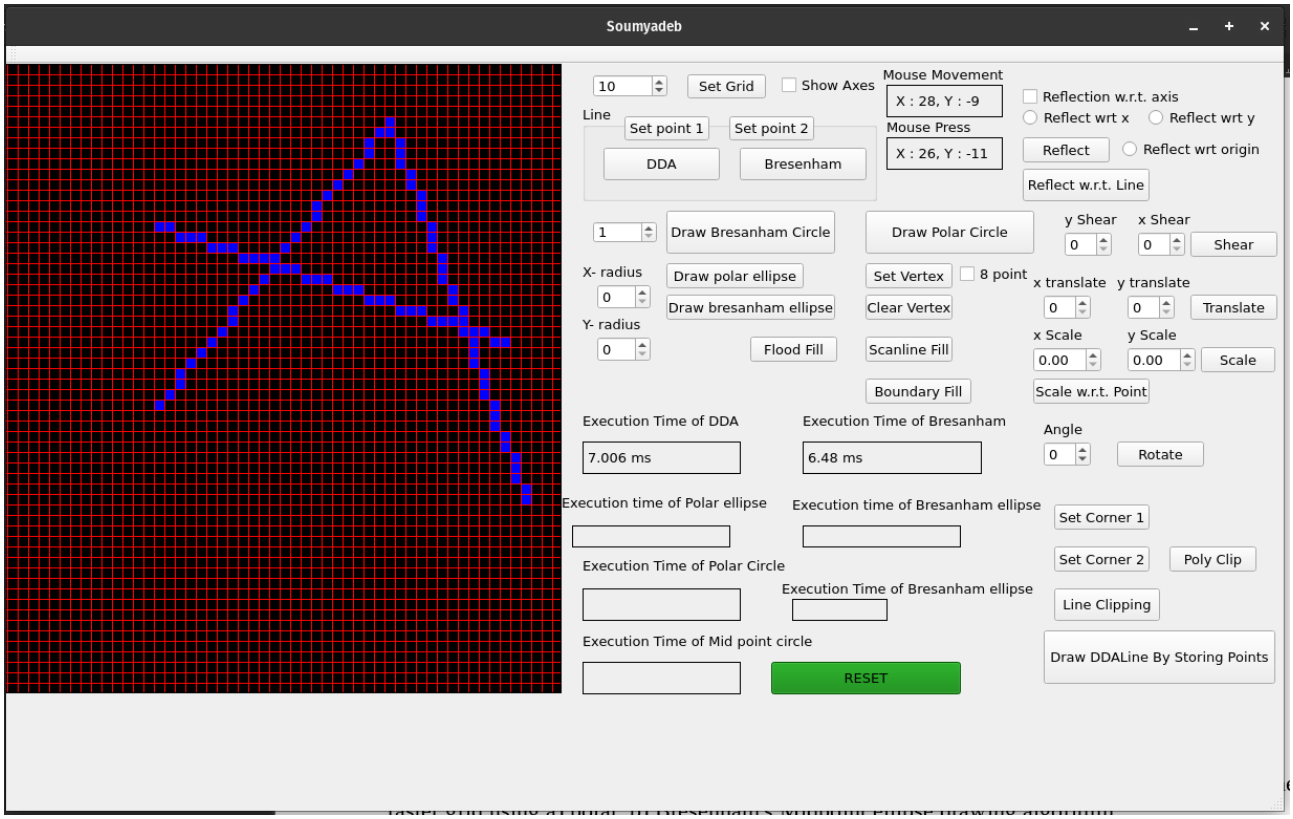
```
            decision += 2*dy;
        } else {
            x += xinc*gridsize;
            y += yinc*gridsize;
            decision += 2*dy - 2*dx;
        }
    }
}
```

Note :

1. p1 and p2 are two end points of the line.
2. The swap function :

```
void swap(int &a, int &b) {
    int temp = b;
    b = a;
    a = temp;
}
```

Output:



Assignment - Implement an ellipse drawing algorithm to draw a circle with a given radius in the raster grid using a) polar, b) Bresenham's Midpoint ellipse drawing algorithm.

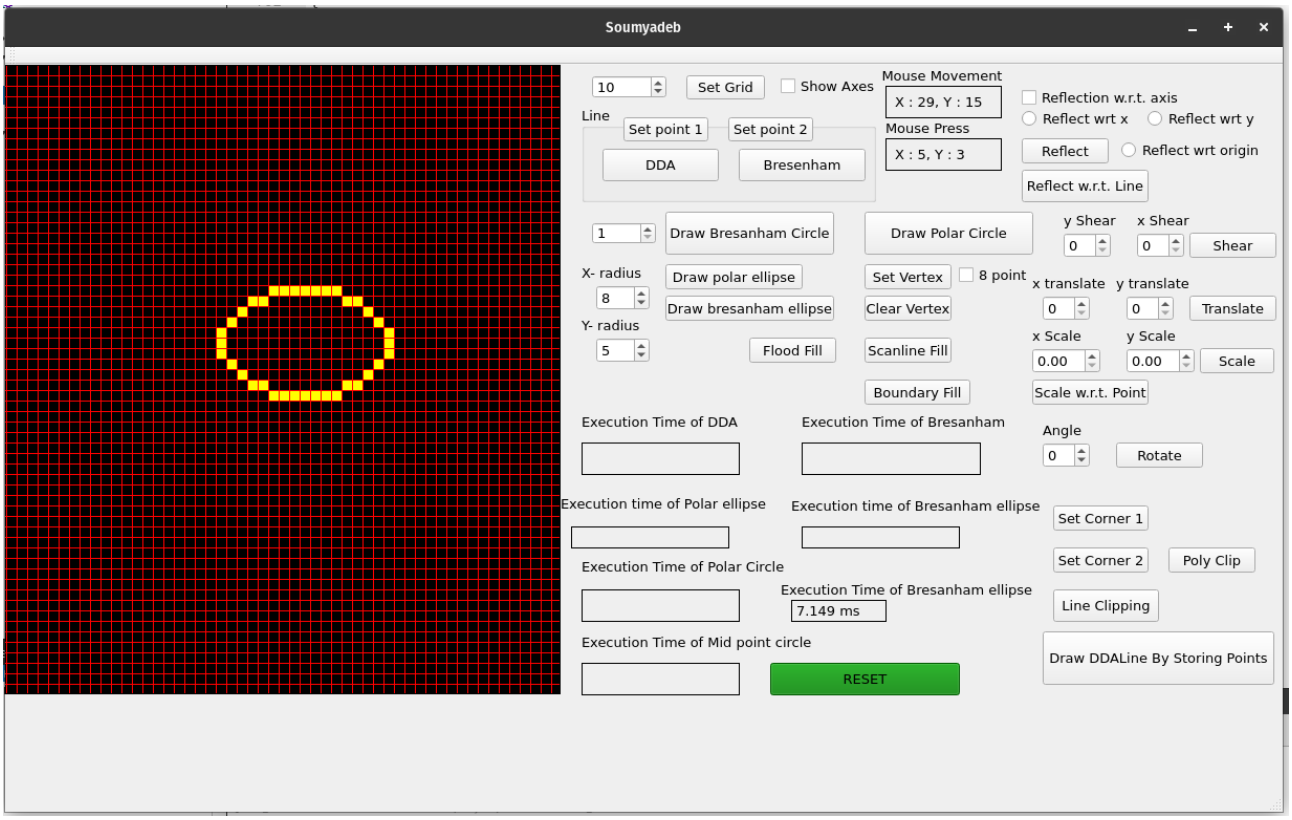
Code Snippet of Polar Ellipse Drawing Algorithm :

```
void MainWindow::on_pushButton_2_clicked()
{
    clock_t start,end;
    start = clock();
    p1.setX(ui->frame->x);
    p1.setY(ui->frame->y);

    int xr = ui->xradius->value();
    int yr = ui->yradius->value();
    int xc = p1.x();
    int yc = p1.y();

    float th1 = 0.0;
    float th2 = 90.0;
    while(th1<th2)
    {
        int x = xr*cos(th1)*gridsize;
        int y = yr*sin(th1)*gridsize;

        point(xc+x,yc+y,200,200,200);
        point(xc-x,yc+y,200,200,200);
        point(xc-x,yc-y,200,200,200);
        point(xc+x,yc-y,200,200,200);

        th1++;
//        delay(1);
    }
    end = clock();
       double time_taken =
(double(end-start)/(double)CLOCKS_PER_SEC)*(double)1000;
       ui->exec_polar_ellipse->setText(QString::number(time_taken)+" ms");
}
```

Note :

1. Time taken for execution is measured by subtracting the time at the beginning of execution from the time at the end of execution. Current time is found out using the ‘clock()’ function.
2. delay(int time); is a function to provide a delay to the execution of the function to properly understand its implementations.

Output:



Code Snippet of Bresanham's Midpoint Ellipse Drawing Algorithm :

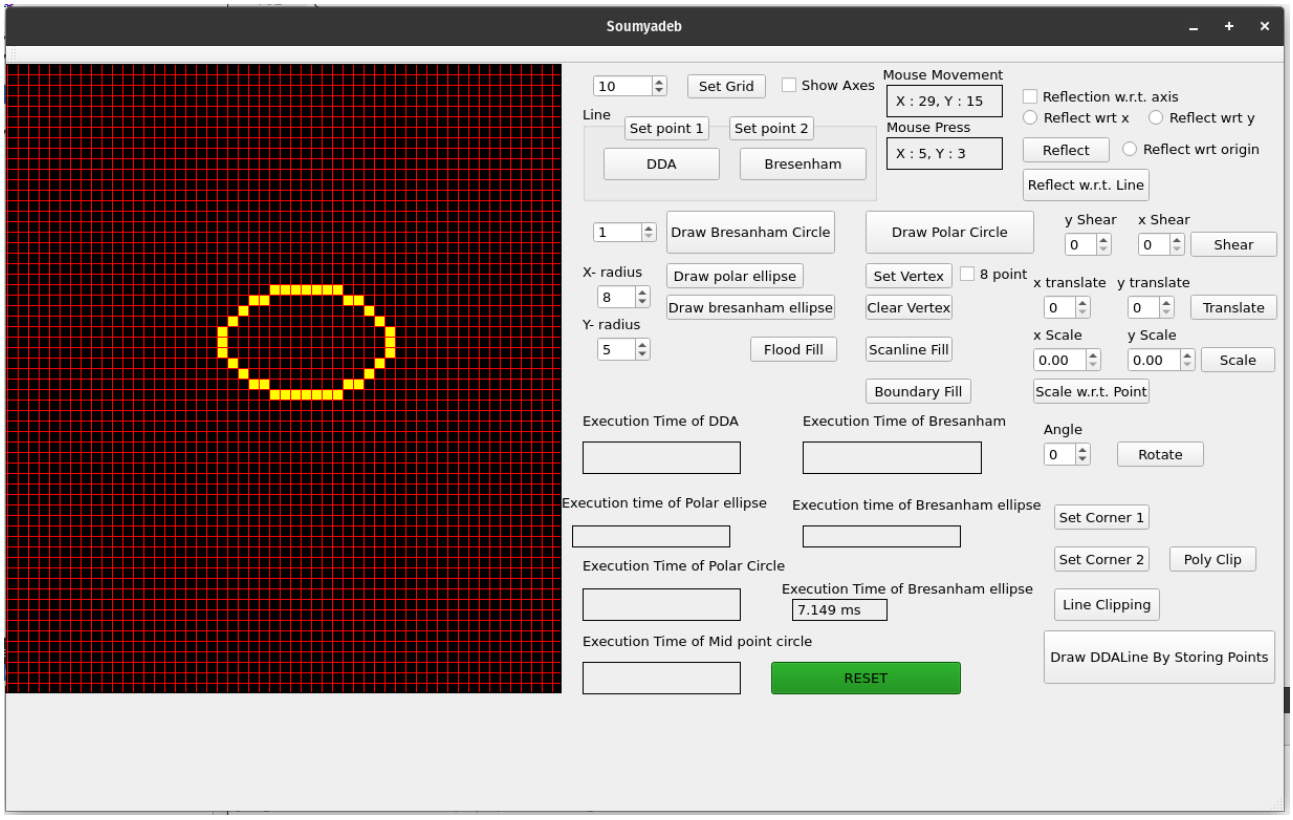
```
void MainWindow::on_ellipse_clicked()
{
      clock_t start,end;
    start = clock();

    p1.setX(ui->frame->x);
    p1.setY(ui->frame->y);
    int rx = ui->r1spinbox->value();
    int ry = ui->r2spinbox->value();
    int x_centre=p1.x();
    int y_centre=p1.y();
    int k = ui->gridspinbox->value();//GridSize

    x_centre=(x_centre/k)*k+k/2;
    y_centre=(y_centre/k)*k+k/2;

    int x=0;
    int y=ry;

    int rx2=rx*rx;
    int ry2=ry*ry;
    int tworx2=2*rx2;
    int twory2=2*ry2;
    int px=0.0;
    int py=tworx2*y;

//For first region
```

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

    while(px<py)
    {
        point(x_centre+x*k,y_centre+y*k);
        point(x_centre-x*k,y_centre+y*k);
        point(x_centre-x*k,y_centre-y*k);
        point(x_centre+x*k,y_centre-y*k);

        x++;
        px+=twory2;
        if(p1>=0) {
            y--;
            py-=tworx2;
            p1=p1+ry2+px-py;

        }  else{
            p1=p1+ry2+px;
        }
    }

    //For second region

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


    while(y>=0)
    {
        point(x_centre+x*k,y_centre+y*k);
        point(x_centre-x*k,y_centre+y*k);
        point(x_centre-x*k,y_centre-y*k);
        point(x_centre+x*k,y_centre-y*k);

        y--;
        py-=tworx2;
        if(p1<=0){
            x++;
            px+=twory2;
            p1=p1+rx2-py+px;

        }  else  {
            p1=p1+rx2-py;
        }
    }
end = clock();
          double time_taken =
(double(end-start)/(double)CLOCKS_PER_SEC)*(double)1000;
          ui->exec_bresanham_ellipse->setText(QString::number(time_taken)+"
ms");

}
```

Note :

1. Time taken for execution is measured by subtracting the time at the beginning of execution from the time at the end of execution. Current time is found out using the ‘clock()’ function.

Output:



Assignment - Implement the seed-fill algorithms: a) Boundary fill, b) Flood fill.

Code Snippet of Boundary Fill Algorithm :

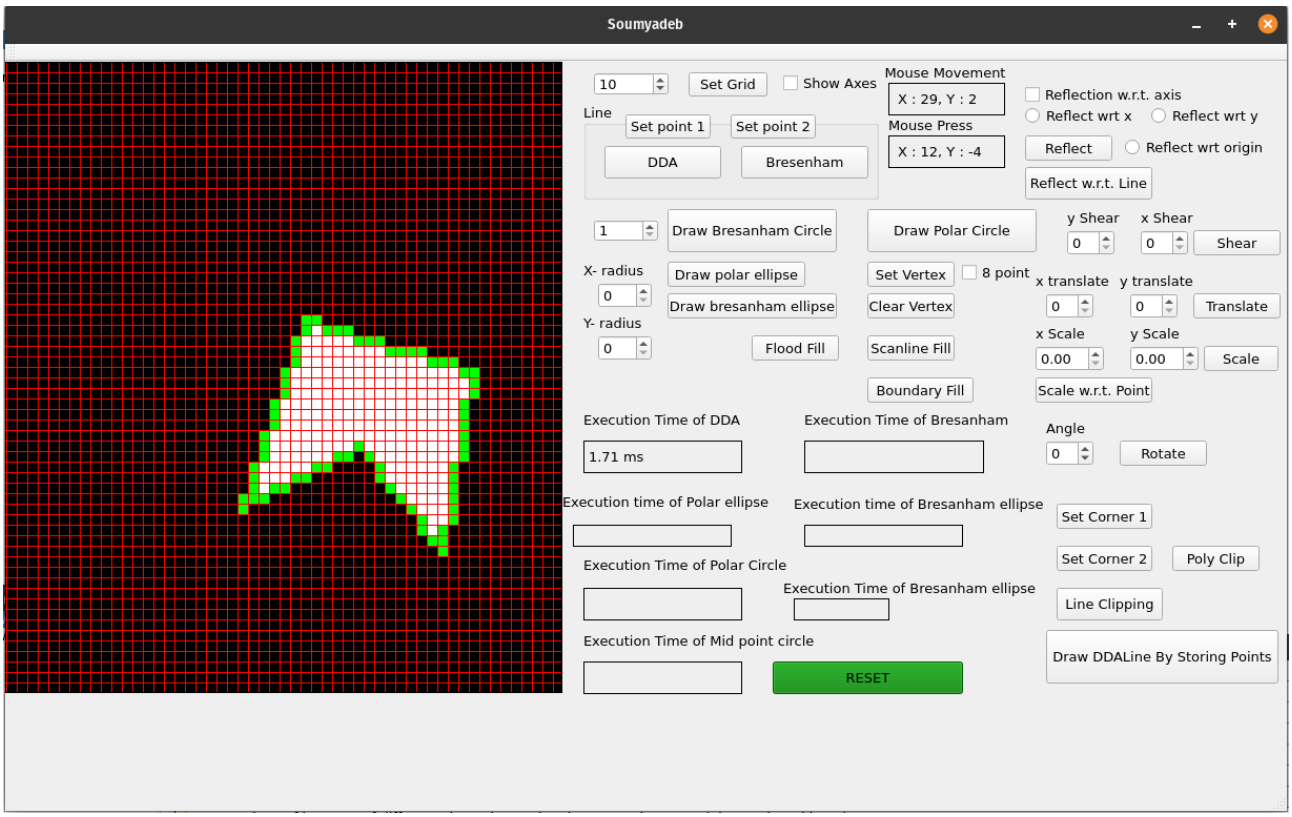
```
void MainWindow::boundFill(int x,int y,QRgb color,int r,int g,int b)
{
    if(x < 0 || y < 0 || x >= img.width() || y >= img.height()) return;
    if(img.pixel(x,y) == color || img.pixel(x,y) == qRgb(r,g,b)) return;
    point(x,y,r,g,b);
    delay(1);
    boundFill(x - gridsize, y, color,r,g,b);
    boundFill(x + gridsize, y, color,r,g,b);
    boundFill(x, y - gridsize, color,r,g,b);
    boundFill(x, y + gridsize, color,r,g,b);
}
void MainWindow::boundFill8(int x,int y,QRgb color,int r,int g,int b)
{
    if(x < 0 || y < 0 || x >= img.width() || y >= img.height()) return;
    if(img.pixel(x,y) == color || img.pixel(x,y) == qRgb(r,g,b)) return;
    point(x,y,r,g,b);
    delay(1);
    boundFill8(x - gridsize, y, color,r,g,b);
    boundFill8(x + gridsize, y, color,r,g,b);
    boundFill8(x, y - gridsize, color,r,g,b);
    boundFill8(x, y + gridsize, color,r,g,b);
    boundFill8(x - gridsize, y + gridsize, color,r,g,b);
    boundFill8(x + gridsize, y - gridsize, color,r,g,b);
    boundFill8(x - gridsize, y - gridsize, color,r,g,b);
    boundFill8(x + gridsize, y + gridsize, color,r,g,b);
}
```

```
void MainWindow::on_boundaryFill_clicked()
{
    p1.setX(ui->frame->x);
        p1.setY(ui->frame->y);
        int x = p1.x()/gridsize;
        int y = p1.y()/gridsize;
        x = x*gridsize + gridsize/2;
        y = y*gridsize + gridsize/2;
        point(x,y,0,0,0);
        if(ui->eight_point->isChecked()){
            boundFill8(x,y,qRgb(0,255,0),255,255,255);
        }
        else boundFill(x,y,qRgb(0,255,0),255,255,255);
}
```

Note :

1. boundFill8 is used for 8 point fill.
2. delay(int time); is a function to provide a delay to the execution of the function to properly understand its implementations.

Output:

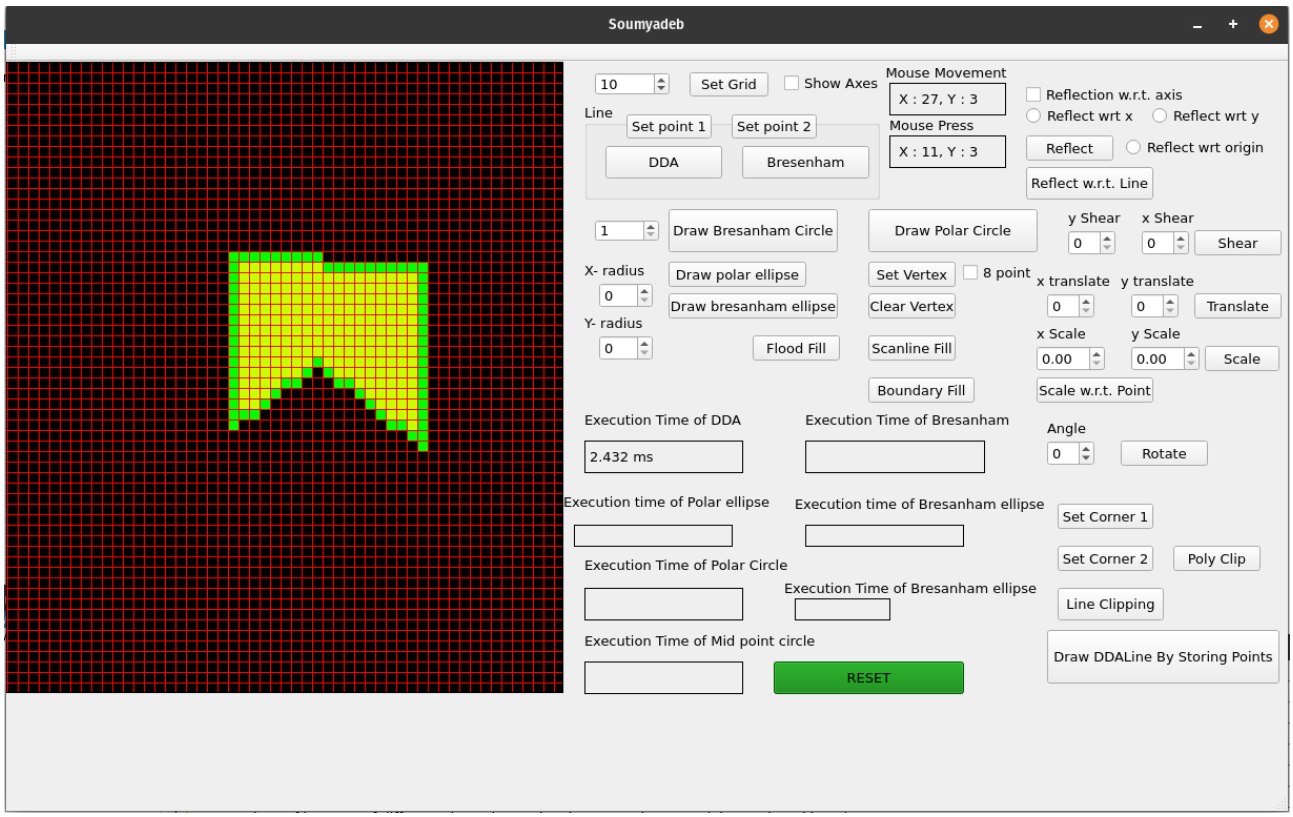


Code Snippet Of Flood Fill Algorithm :

```
void MainWindow:: floodFill(int x,int y,int r,int g,int b,QRgb prevC)
{
    if(x < 0 || y < 0 || x >= img.width() || y >= img.height()) return;
    if(img.pixel(x,y) == qRgb(r,g,b)) return;
    if(img.pixel(x,y) != prevC) return;
    point(x,y,r,g,b);
    delay(1);
    floodFill(x - gridsize, y, r,g,b,prevC);
    floodFill(x + gridsize, y, r,g,b,prevC);
```

```
    floodFill(x, y - gridsize, r,g,b,prevC);
    floodFill(x, y + gridsize, r,g,b,prevC);
}

void MainWindow::floodFill8(int x,int y,int r,int g,int b,QRgb prevC)
{
    if(x < 0 || y < 0 || x >= img.width() || y >= img.height()) return;
    if(img.pixel(x,y) == qRgb(r,g,b)) return;
    if(img.pixel(x,y) != prevC) return;
    point(x,y,r,g,b);
    delay(1);
    floodFill8(x - gridsize, y, r,g,b,prevC);
    floodFill8(x + gridsize, y, r,g,b,prevC);
    floodFill8(x, y - gridsize, r,g,b,prevC);
    floodFill8(x, y + gridsize, r,g,b,prevC);
    floodFill8(x - gridsize, y - gridsize, r,g,b,prevC);
    floodFill8(x + gridsize, y + gridsize, r,g,b,prevC);
    floodFill8(x + gridsize, y - gridsize, r,g,b,prevC);
    floodFill8(x - gridsize, y + gridsize, r,g,b,prevC);
}

void MainWindow::on_flood_fill_clicked()
{
    p1.setX(ui->frame->x);
        p1.setY(ui->frame->y);
        int x = p1.x()/gridsize;
        int y = p1.y()/gridsize;
        x = x*gridsize + gridsize/2;
        y = y*gridsize + gridsize/2;
        QRgb c = img.pixel(x,y);
        if(ui->eight_point->isChecked()){
            floodFill8(x,y,200,255,0,c);
        }
        else floodFill(x,y,200,255,0,c);
}
```

Output:


Soumyadeb
10
Set Grid
Show Axes
Mouse Movement
X : 27, Y : 3
Reflection w.r.t. axis
Reflect wrt x
Reflect wrt y
Line
Set point 1
Set point 2
DDA
Bresenham
Mouse Press
X : 11, Y : 3
Reflect
Reflect wrt origin
Reflect w.r.t. Line
1
Draw Bresanham Circle
Draw Polar Circle
y Shear
x Shear
0
0
Shear
X- radius
0
Draw polar ellipse
Draw bresanham ellipse
Set Vertex
8 point
Clear Vertex
x translate
y translate
0
0
Translate
Y- radius
0
Flood Fill
Scanline Fill
x Scale
y Scale
0.00
0.00
Scale
Boundary Fill
Scale w.r.t. Point
Execution Time of DDA
2.432 ms
Execution Time of Bresanham
Angle
0
Rotate
Execution time of Polar ellipse
Execution time of Bresanham ellipse
Set Corner 1
Set Corner 2
Poly Clip
Execution Time of Polar Circle
Execution Time of Bresanham ellipse
Line Clipping
Execution Time of Mid point circle
RESET
Draw DDALine By Storing Points