Computer Graphics Lab

Name: Tonmoy Biswas

Roll No: 002110501133

Class: BCSE 3rd Year 1st Sem

Section: A3

Assignment Topics

1. Create a raster grid showing 4 quadrants to plot pixels of variable sizes.
2. Implement a line drawing algorithm to draw lines between two end points in the raster grid using, a) Parametric Line Drawing b) DDA c) Bresenham's line drawing algorithm. Show execution times for each algorithm in ms. Check for all possible line endpoints in 4 quadrants.
3. Implement a circle drawing algorithm to draw a circle with a given radius in the raster grid using, a) polar b) Bresenham's Midpoint circle drawing algorithm. Optional: a) Cartesian b) Midpoint. Check for execution time in ms.
4. 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. Check for execution time in ms.
5. Implement the seed-fill algorithms: a) Boundary fill, b) Flood fill.
6. Draw a closed polygon. Implement scanline fill algorithm to fill the polygon.
7. Draw a closed polygon and implement different transformation functions (with respect to origin) on it. a) translation, b) rotation, c) scaling, d) shear, e) reflection with respect to x/y axes. Extend the algorithm to apply the transformations successively on the same object using homogeneous coordinates and matrix multiplication.
8. Composite Transformation. a) Rotation and scaling with respect to an arbitrary point, b) reflection with respect to an arbitrary line.
9. Implement Line Clipping with respect to a rectangular clip window, using a) Cohen-Sutherland Algorithm, b) Liang-Barsky Algorithm (optional)
10. Implement Polygon Clipping with respect to a rectangular clip window, using a) Sutherland-Hodgeman Algorithm, b) Weiler-Atherton Algorithm (optional).

Assignment 1:

Statement: Create a raster grid showing 4 quadrants to plot pixels of variable sizes.

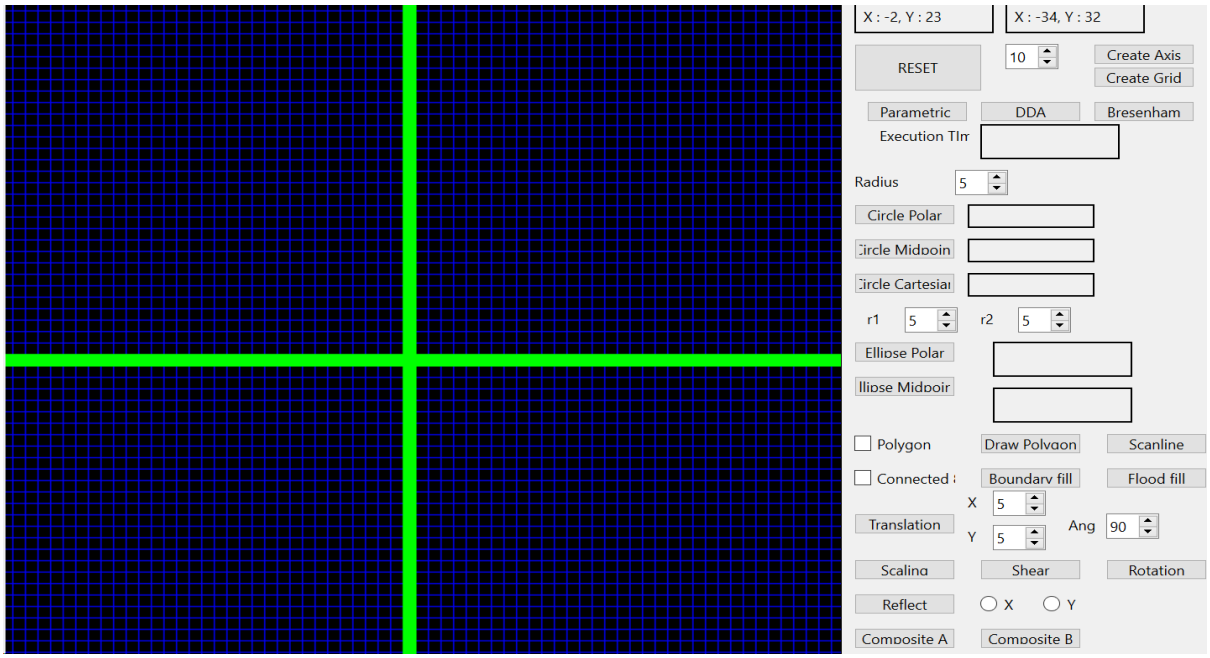
Code: on_draw_grid_with_axes_clicked()

```
{
   int val=ui->spinBox->value();
   int low=(350/val)*val;
   int high=(350/val)*val+val;
   if(ui->show_axes->isChecked()){
      for(int i=0;i<img.width();i++)
      {
         for(int j=low;j<=high;j++){
            img.setPixel(i,j,qRgb(0,255,0));
            img.setPixel(j,i,qRgb(0,255,0));
            //if(arr[i/val][j/val])
         }
      }
   }
   else{
      for(int i=0;i<img.width();i++)
      {
         for(int j=low;j<=high;j++){
```

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

}

Output:



Assignment 2:

Statement: Implement a line drawing algorithm to draw lines between two end points in the raster grid using, a) Parametric Line Drawing b) DDA c) Bresenham's line drawing algorithm. Show execution times for each algorithm in ms. Check for all possible line endpoints in 4 quadrants.

Parametric Code: on_parametric_line_clicked()

```
{
   int x1=point1.first,y1=point1.second;
   int x2=point2.first,y2=point2.second;
//    float m=float(y1-y2)/float(x1-x2);
   vector<pair<int,int>> points;
   clock_t start,end;
//    time(&start);
   start=clock();
//    bool flag=true;
   if(x1==x2){
```

```
        int low=min(y1,y2);
        int high=max(y1,y2);
        for(int y=low;y<=high;y++){
//          setColor(x1,y,1);
           points.push_back({x1,y});
           //to track the set point
//          arr[x1][y]=1;
        }}
    else if(abs(y1-y2)<=abs(x1-x2)){
        int low=min(x1,x2);
        int high=max(x1,x2);
        float m=float(y1-y2)/float(x1-x2);
        for( int ix=low; ix<=high; ix++ )
        {
           //float fy = y0 + float(yn-y0)/float(xn-x0)*ix;                    // round to
nearest whole integer
           float fy=m*(ix-x1)+y1;
           int iy = (int)(fy+0.5);
//          setColor(ix,iy,1);
           points.push_back({ix,iy});
           //to track the set point
//          arr[ix][iy]=1;
        }}
    else{
        int low=min(y1,y2);
        int high=max(y1,y2);
        float m=float(y1-y2)/float(x1-x2);
        for(int iy=low;iy<=high;iy++){
```

```
            float fx=(iy-y1)/m+x1;
            int ix=(int)(fx+0.5);
//              setColor(ix,iy,1);
            points.push_back({ix,iy});
            //to track the set point
//              arr[ix][iy]=1;
        }
    }
    for(auto it:points){
        setColor(it.first,it.second,1);
//      s.insert({{it.first,it.second},1});
    }
    end=clock();
    double time_taken=double(end-start);
    ui->execution_time->setText(QString::number(time_taken)+" ms");
    int val=ui->spinBox->value();
    int originX=350/val;
    int originY=350/val;
    for(auto it:points){
//      setColor(it.first,it.second,1);
        s.insert({{it.first-originX,it.second-originY},1});
    }
}
```

DDA Code: on_DDA_line_clicked()

```
{
    int x0=point1.first,y0=point1.second;
    int xn=point2.first,yn=point2.second;
vector<pair<int,int>> points;
```

```
    bool flag=true;
    clock_t start,end;
//    time(&start);
    start=clock();
    if(x0==xn){
        int low=min(y0,yn);
        int high=max(y0,yn);
        for(int y=low;y<=high;y++){
//            setColor(xn,y,2);
            points.push_back({xn,y});
            //to track the set point
//            arr[xn][y]=2;
        }
//        return ;
        flag=false;
    }
    else if(y0==yn){
        int low=min(x0,xn);
        int high=max(x0,xn);
        for(int x=low;x<=high;x++){
//            setColor(x,yn,2);
            points.push_back({x,yn});
            //to track the set point
//            arr[x][yn]=2;
        }
//        return ;
        flag=false;
    }
```

```
if(flag){
   float dx = float(xn - x0); // total span in x
   float dy = float(yn - y0); // total span in y
   //    float y = float(y0);
   //    float x = float(x0);
   float Dx, Dy, x, y; // incremental steps in x & y
   // determine if slope m GT or LT 1
   if( abs(dx) >= abs(dy) )
   {
      Dx = 1;
      Dy = dy/dx;// floating division, but only done once per line
      if(x0<xn){
         x=x0;
         y=y0;
      }
      else{
         x=xn;
         y=yn;
      } }
   else
   {
      Dx = dx/dy;
      Dy = 1;
      if(y0<yn){
         x=x0;
         y=y0;
      }
      else{
```

```
            x=xn;
            y=yn;
          }}
      int ix, iy; // pixel coords
      for( int k=0; k<=abs(((abs(dx)>abs(dy))? dx:dy)); k++ )
      {
         ix = int(x + 0.5); // round to nearest pixel coordinate
         iy = int(y + 0.5);
         points.push_back({ix,iy});
         x += Dx;
         y += Dy;
      } }
   for(auto it:points){
      setColor(it.first,it.second,2);}
   end=clock();
   double time_taken=double(end-start);
   ui->execution_time->setText(QString::number(time_taken)+" ms");
   int val=ui->spinBox->value();
   int originX=350/val;
   int originY=350/val;
}
```

Bresenham Code: on_Bresenham_line_clicked()

```
{
   int x0=point1.first,y0=point1.second;
   int xn=point2.first,yn=point2.second;
   vector<pair<int,int>> points;
   clock_t start,end;
   start=clock();
```

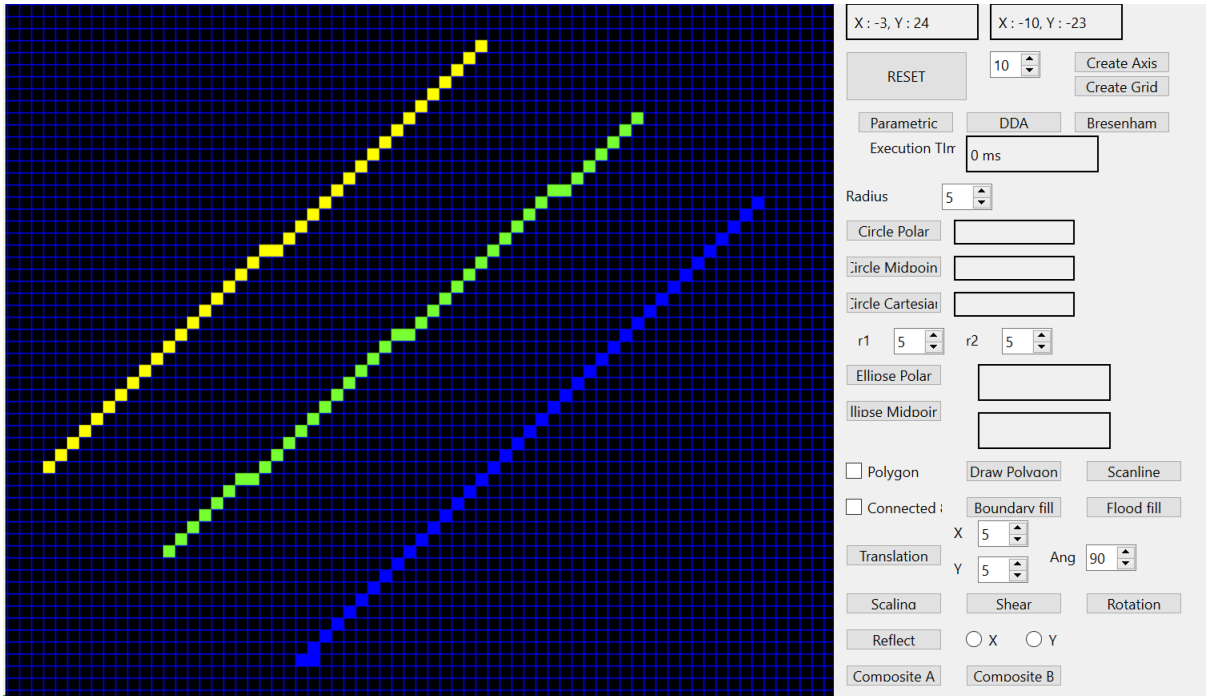
```
    if(point1.first>point2.first){
        swap(x0,xn);
        swap(y0,yn);
    }
    if(abs(xn-x0)>=abs(yn-y0)){
        int dx=xn-x0;
        int dy=yn-y0;
        int dx2=2*dx;
        int dy2=2*dy;
        int dp=dy2-dx2;
        int dn=dy2+dx2;
        int p0=dy2-dx;
        int yk=y0;
        bool isNegSlop;
        if(yn>=y0){
            isNegSlop=false;
        }
        else{
            isNegSlop=true;
        }
        for(int xk=x0;xk<=xn;xk++){
//          setColor(xk,yk,3);
            points.push_back({xk,yk});
            if(isNegSlop){
                if(p0<0){
                    yk--;
                    p0=p0+dn;
                }
```

```
            else{
               //              yk++;
               p0=p0+dy2;
            } }
         else{
            if(p0<0){
               p0=p0+dy2;
            }
            else{
               yk++;
               p0=p0+dp;
            }} }  }
   else{
      swap(x0,y0);
      swap(xn,yn);
      if(x0>xn){
         swap(x0,xn);
         swap(y0,yn);}
      int dx=xn-x0;
      int dy=yn-y0;
      int dx2=2*dx;
      int dy2=2*dy;
      int dp=dy2-dx2;
      int dn=dy2+dx2;
      int p0=dy2-dx;
      int yk=y0;
      bool isNegSlop;
      if(yn>=y0){
```

```
        isNegSlop=false;
    }
    else{
        isNegSlop=true;
    }
    for(int xk=x0;xk<=xn;xk++){
//          setColor(yk,xk,3);
        points.push_back({yk,xk});
        if(isNegSlop){
            if(p0<0){
                yk--;
                p0=p0+dn;
            }
            else{
                //              yk++;
                p0=p0+dy2;
            }}
        else{
            if(p0<0){
                p0=p0+dy2;
            }
            else{
                yk++;
                p0=p0+dp;
            }}}}
   //   setPoints(points,2);
   for(auto it:points){
       setColor(it.first,it.second,3);
```

```
        sleep(s_time);
//        s.insert({{it.first,it.second},3});
    }
    end=clock();
    double time_taken=double(end-start);
    ui->execution_time->setText(QString::number(time_taken)+" ms");
 int val=ui->spinBox->value();
    int originX=350/val;
    int originY=350/val;
    for(auto it:points){
        //        setColor(it.first,it.second,1);
        s.insert({{it.first-originX,it.second-originY},3});
    }}
```

Output:



Left: Parametric Middle: DDA Right: Bresenham

Assignment 3:

Statement: Implement a circle drawing algorithm to draw a circle with a given radius in the raster grid using, a) polar b) Bresenham's Midpoint circle drawing algorithm. Optional: a) Cartesian b) Midpoint. Check for execution time in ms.

Polar Code: on_circle_polar_clicked()

```
{int centerX=point2.first;
   int centerY=point2.second;
   int r=ui->Radius->value();
   double inc=((double)1/(r));
   double low=3.14/4;
   double high=3.14/2;
   clock_t start,end;
   start=clock();
   int r2=r*r;
   vector<pair<int,int>> points;
   points.push_back({centerX+0,centerY+r});
   points.push_back({centerX+0,centerY-r});
   points.push_back({centerX+r,centerY+0});
   points.push_back({centerX-r,centerY+0});
   for(double it=low;it<=high;it+=inc){
//       int y=sqrt(r2-i*i)+0.5;
      int x=r*cos(it)+0.5;
      int y=r*sin(it)+0.5;
      int i=x;
      if(i>y){
         break;}
      points.push_back({centerX+i,centerY+y});
      points.push_back({centerX+i,centerY-y});
      points.push_back({centerX-i,centerY+y});
```

```
        points.push_back({centerX-i,centerY-y});
        points.push_back({centerX+y,centerY+i});
        points.push_back({centerX+y,centerY-i});
        points.push_back({centerX-y,centerY+i});
        points.push_back({centerX-y,centerY-i});}
    //color the pixels
    for(auto it:points){
        setColor(it.first,it.second,4);
        sleep(s_time);
    }
    end=clock();
    double time_taken=double(end-start);
    ui->exe_time_cir_polar->setText(QString::number(time_taken)+" ms");
    int val=ui->spinBox->value();
    int originX=350/val;
    int originY=350/val;
    for(auto it:points){
        //      setColor(it.first,it.second,1);
        s.insert({{it.first-originX,it.second-originY},4});
    }}
```

Bresenham Code: on_circle_Bresenham_clicked()

```
{
    int centerX=point2.first;
    int centerY=point2.second;
    int r=ui->Radius->value();
    int r2=r*r;
    vector<pair<int,int>> points;
    clock_t start,end;
```

```
start=clock();
int p=1-r;
int x=0,y=r;
points.push_back({centerX+0,centerY+r});
points.push_back({centerX+0,centerY-r});
points.push_back({centerX+r,centerY+0});
points.push_back({centerX-r,centerY+0});
while(x<=y){
   if(p<0){
      p=p+2*(x+1)+1;
      x=x+1;
      y=y;}
   else{
      p=p+2*(x+1)+1-2*(y-1);
      x=x+1;
      y=y-1;}
   int i=x;
   if(x>y){
      break;    }
   points.push_back({centerX+i,centerY+y});
  points.push_back({centerX+i,centerY-y});
   points.push_back({centerX-i,centerY+y});
   points.push_back({centerX-i,centerY-y});
   points.push_back({centerX+y,centerY+i});
   points.push_back({centerX+y,centerY-i});
   points.push_back({centerX-y,centerY+i});
   points.push_back({centerX-y,centerY-i});
}
```

```
    //color the pixels
    for(auto it:points){
       setColor(it.first,it.second,5);
       sleep(s_time);
    }
    end=clock();
    double time_taken=double(end-start);
    ui->exe_time_cir_midpoint->setText(QString::number(time_taken)+" ms");
    int val=ui->spinBox->value();
    int originX=350/val;
    int originY=350/val;
    for(auto it:points){
      s.insert({{it.first-originX,it.second-originY},5});}}
```

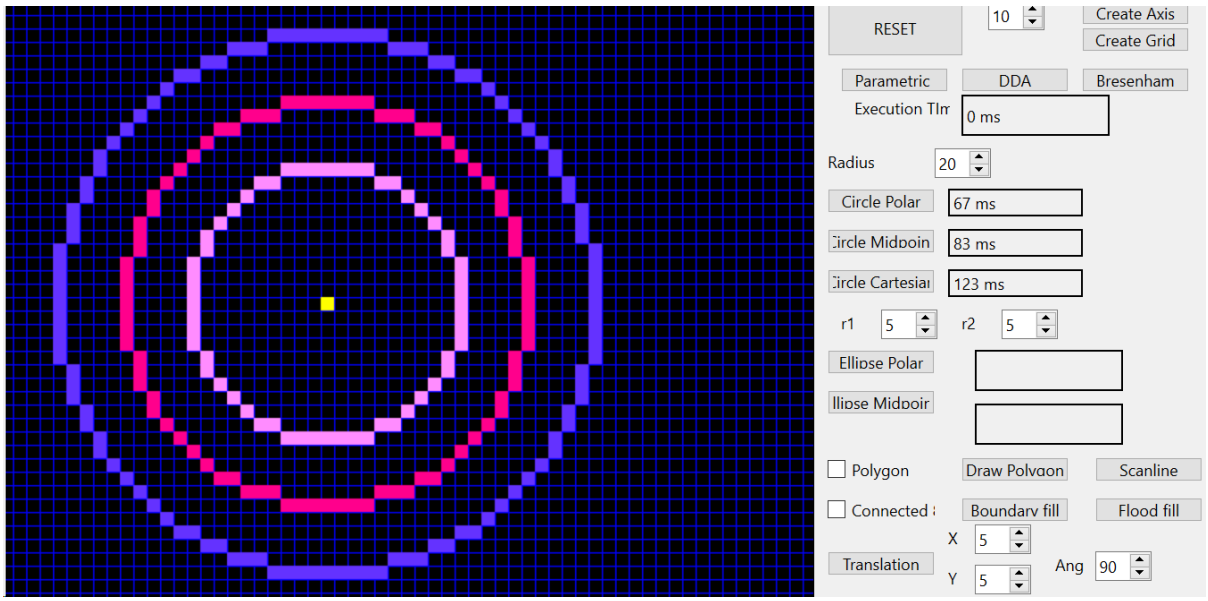
Cartesian Code: on_circle_cartesian_clicked()

```
{
    int centerX=point2.first;
    int centerY=point2.second;
    int r=ui->Radius->value();
//    double inc=((double)1/(r));
    double low=0;
    double high=r;
    clock_t start,end;
    start=clock();
int r2=r*r;
    vector<pair<int,int>> points;
    points.push_back({centerX+0,centerY+r});
    points.push_back({centerX+0,centerY-r});
    points.push_back({centerX+r,centerY+0});
```

```
    points.push_back({centerX-r,centerY+0});
    for(int it=low;it<=high;it++){
            int y=sqrt(r2-it*it)+0.5;
//      int x=r*cos(it)+0.5;
//      int y=r*sin(it)+0.5;
      int i=it;
      if(i>y){
         break;}
      points.push_back({centerX+i,centerY+y});
      points.push_back({centerX+i,centerY-y});
      points.push_back({centerX-i,centerY+y});
      points.push_back({centerX-i,centerY-y});
      points.push_back({centerX+y,centerY+i});
      points.push_back({centerX+y,centerY-i});
      points.push_back({centerX-y,centerY+i});
      points.push_back({centerX-y,centerY-i});    }
    //color the pixels
    for(auto it:points){
      setColor(it.first,it.second,8);
      sleep(s_time);
    }
    end=clock();
    double time_taken=double(end-start);
    ui->exe_time_cir_cartesian->setText(QString::number(time_taken)+" ms");
    int val=ui->spinBox->value();
    int originX=350/val;
    int originY=350/val;
    for(auto it:points){
```

```
    //      setColor(it.first,it.second,1);
    s.insert({{it.first-originX,it.second-originY},8});
  }}
```

Output:



Inner: Polar Middle: Bresenham Outer: Cartesian

Assignment 4:

Statement: 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. Check for execution time in ms.

Polar Code: on_ellipse_polar_clicked()

```
{
   int centerX=point2.first;
   int centerY=point2.second;
   int b=ui->r1->value();
   int a=ui->r2->value();
   vector<pair<int,int>> points;
   clock_t start,end;
   start=clock();
   points.push_back({centerX+0,centerY+b});
```

```
    points.push_back({centerX+0,centerY-b});
    points.push_back({centerX+a,centerY+0});
    points.push_back({centerX-a,centerY+0});
//    int x=0,y=b;
//    int b2=b*b;
//    int a2=a*a;
    double inc=((double)1)/max(a,b);
    double low=0;
    double high=3.14/2;
    for(double it=low;it<=high;it+=inc){
        int x=a*cos(it)+0.5;
        int y=b*sin(it)+0.5;
        points.push_back({centerX+x,centerY+y});
        points.push_back({centerX+x,centerY-y});
        points.push_back({centerX-x,centerY+y});
        points.push_back({centerX-x,centerY-y});}
    //color the pixels
    for(auto it:points){
        setColor(it.first,it.second,7);
        sleep(s_time);
    }
    end=clock();
    double time_taken=double(end-start);
    ui->exe_time_ell_polar->setText(QString::number(time_taken)+" ms");
    int val=ui->spinBox->value();
    int originX=350/val;
    int originY=350/val;
    for(auto it:points){
```

```
    //      setColor(it.first,it.second,1);
      s.insert({{it.first-originX,it.second-originY},7});
   }}
```

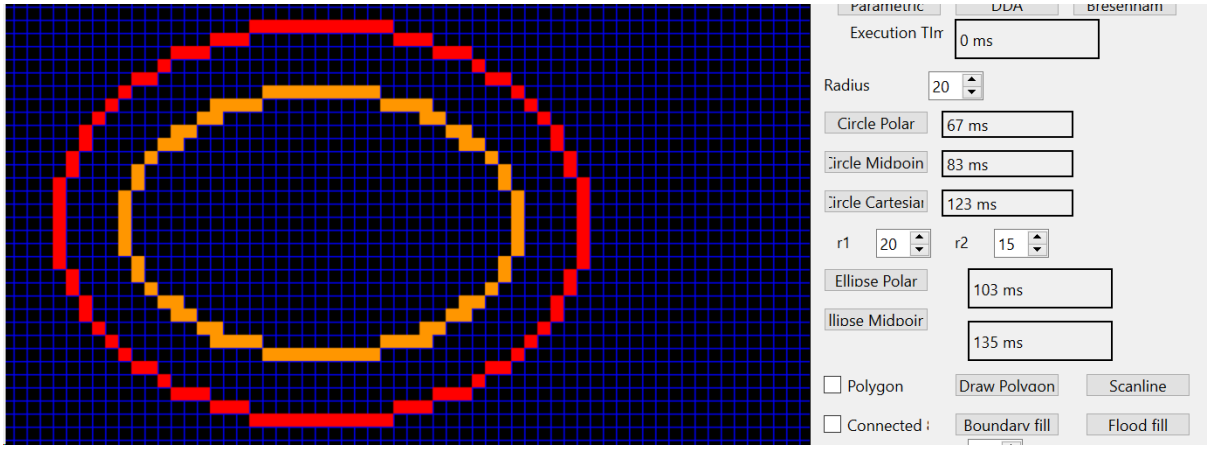
Bresenham Code: on_Bresenham_ellipse_clicked()

```
{
   int centerX=point2.first;
   int centerY=point2.second;
   int b=ui->r1->value();
   int a=ui->r2->value();
   vector<pair<int,int>> points;
   clock_t start,end;
   start=clock();
   points.push_back({centerX+0,centerY+b});
   points.push_back({centerX+0,centerY-b});
   points.push_back({centerX+a,centerY+0});
   points.push_back({centerX-a,centerY+0});
   int x=0,y=b;
   int b2=b*b;
   int a2=a*a;
   double d1=b2-a2*b+0.25*a2;
   while(a2*(y-0.5)>b2*(x+1)){
      if(d1<0){
         x++;
         d1+=2*b2*x+b2;
      }
      else{
         x++;
         y--;
```

```
        d1+=2*b2*x+b2-2*a2*y;
    }
    points.push_back({centerX+x,centerY+y});
    points.push_back({centerX+x,centerY-y});
    points.push_back({centerX-x,centerY+y});
    points.push_back({centerX-x,centerY-y});
}
double d2=b2*pow(x+0.5,2)+a2*pow(y-1,2)-a2*b2;
while(y>0){
    if(d2<0){
        y--;
        x++;
        d2+=a2*(1-2*y)+2*x*b2; }
    else{
        y--;
        d2+=a2*(1-2*y); }
    points.push_back({centerX+x,centerY+y});
    points.push_back({centerX+x,centerY-y});
    points.push_back({centerX-x,centerY+y});
    points.push_back({centerX-x,centerY-y});}
//color the pixels
for(auto it:points){
    setColor(it.first,it.second,6);
    sleep(s_time);
}
end=clock();
double time_taken=double(end-start);
ui->exe_time_ell_midpoint->setText(QString::number(time_taken)+" ms");
```

```
int val=ui->spinBox->value();
int originX=350/val;
int originY=350/val;
for(auto it:points){
    //      setColor(it.first,it.second,1);
    s.insert({{it.first-originX,it.second-originY},6});
}}
```

Output:



Inner: Polar Outer: Bresenham

Assignment 5:

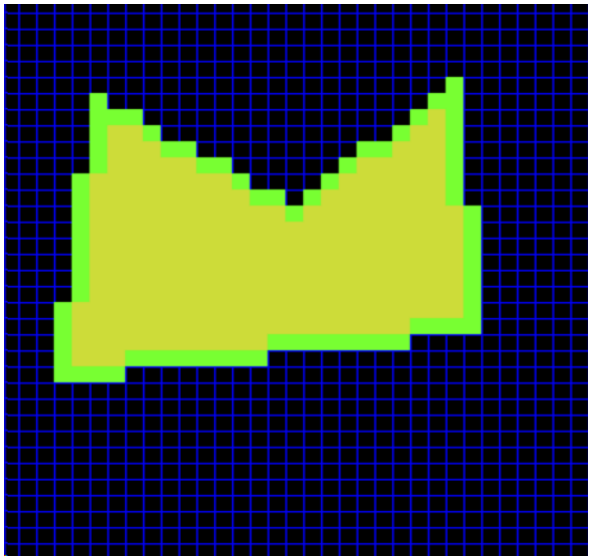
Statement: Implement the seed-fill algorithms: a) Boundary fill, b) Flood fill.

Boundary Fill Code: boundary_fill(int x,int y,QRgb fill_color,QRgb boundary_color){

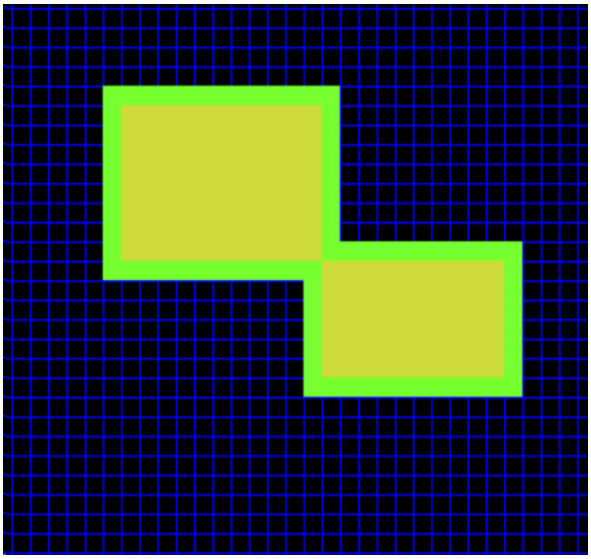
```
int val=ui->spinBox->value();
int originX=350/val;
int originY=350/val;
int mid_point_x=x*val+(val)/2;
int mid_point_y=y*val+(val)/2;
if((mid_point_x<0) || (mid_point_x>700) || (mid_point_y<0) || (mid_point_y>700)){
    return ;
```

```
    }
if(img.pixel(mid_point_y,mid_point_x)!=boundary_color&&img.pixel(mid_poin
t_y,mid_point_x)!=fill_color){
        int x_low=x*val;
        int y_low=y*val;
        int x_high=x_low+val;
        int y_high=y_low+val;
//        myDelay();
        for(int i=x_low;i<x_high;i++){
            for(int j=y_low;j<y_high;j++){
                img.setPixel(j,i,fill_color);
            }
        }
        ui->frame->setPixmap(QPixmap::fromImage(img));
        boundary_fill(x,y+1,fill_color,boundary_color);
        boundary_fill(x,y-1,fill_color,boundary_color);
        boundary_fill(x-1,y,fill_color,boundary_color);
        boundary_fill(x+1,y,fill_color,boundary_color);
        if(ui->connected_8->isChecked()){
            boundary_fill(x+1,y+1,fill_color,boundary_color);
            boundary_fill(x+1,y-1,fill_color,boundary_color);
            boundary_fill(x-1,y+1,fill_color,boundary_color);
            boundary_fill(x-1,y-1,fill_color,boundary_color);
        }}}
```

Output:



4 connected boundary fill



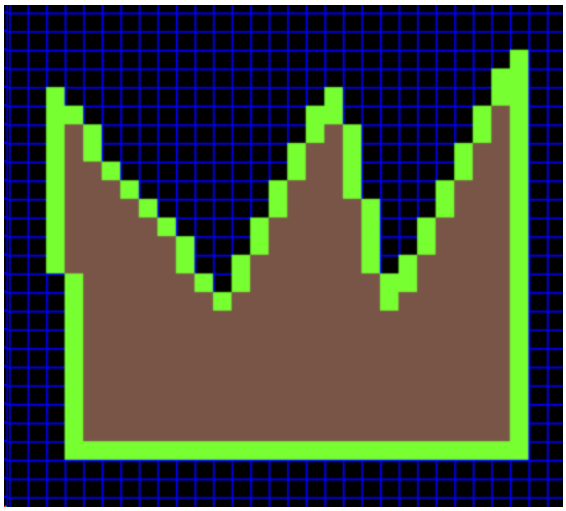
8 connected boundary fill

Flood fill Code:

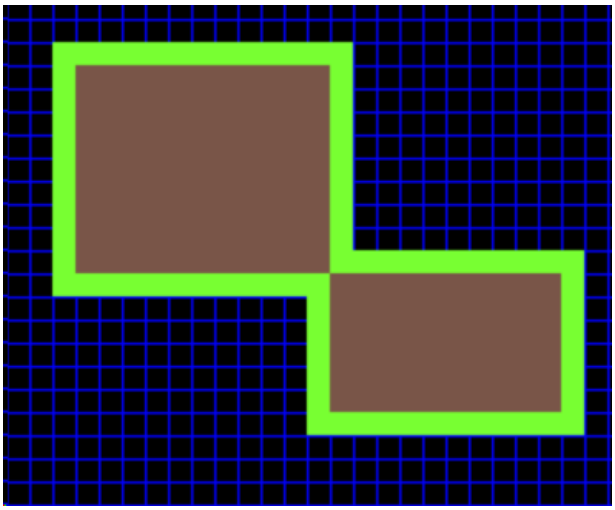
```
 flood_fill(int x,int y,QRgb old_color){
    int val=ui->spinBox->value();
    int mid_point_x=x*val+(val)/2;
    int mid_point_y=y*val+(val)/2;
    if((mid_point_x<0) || (mid_point_x>700) || (mid_point_y<0) ||
(mid_point_y>700)){
        return ;
    }
    QRgb fill_color=qRgb(121,85,72);
if(img.pixel(mid_point_y,mid_point_x)==old_color/*||img.pixel(mid_point_y,m
id_point_x)==qRgb(255,255,0)*/){
        int x_low=x*val;
        int y_low=y*val;
        int x_high=x_low+val;
        int y_high=y_low+val;
        for(int i=x_low;i<x_high;i++){
            for(int j=y_low;j<y_high;j++){
                img.setPixel(j,i,fill_color);}}
```

```
ui->frame->setPixmap(QPixmap::fromImage(img));
flood_fill(x,y+1,old_color);
flood_fill(x,y-1,old_color);
flood_fill(x-1,y,old_color);
flood_fill(x+1,y,old_color);
if(ui->connected_8->isChecked()){
   flood_fill(x+1,y+1,old_color);
   flood_fill(x+1,y-1,old_color);
   flood_fill(x-1,y+1,old_color);
   flood_fill(x-1,y-1,old_color);
}}}
```

Output:



4 Connected flood fill



8 connected flood fill