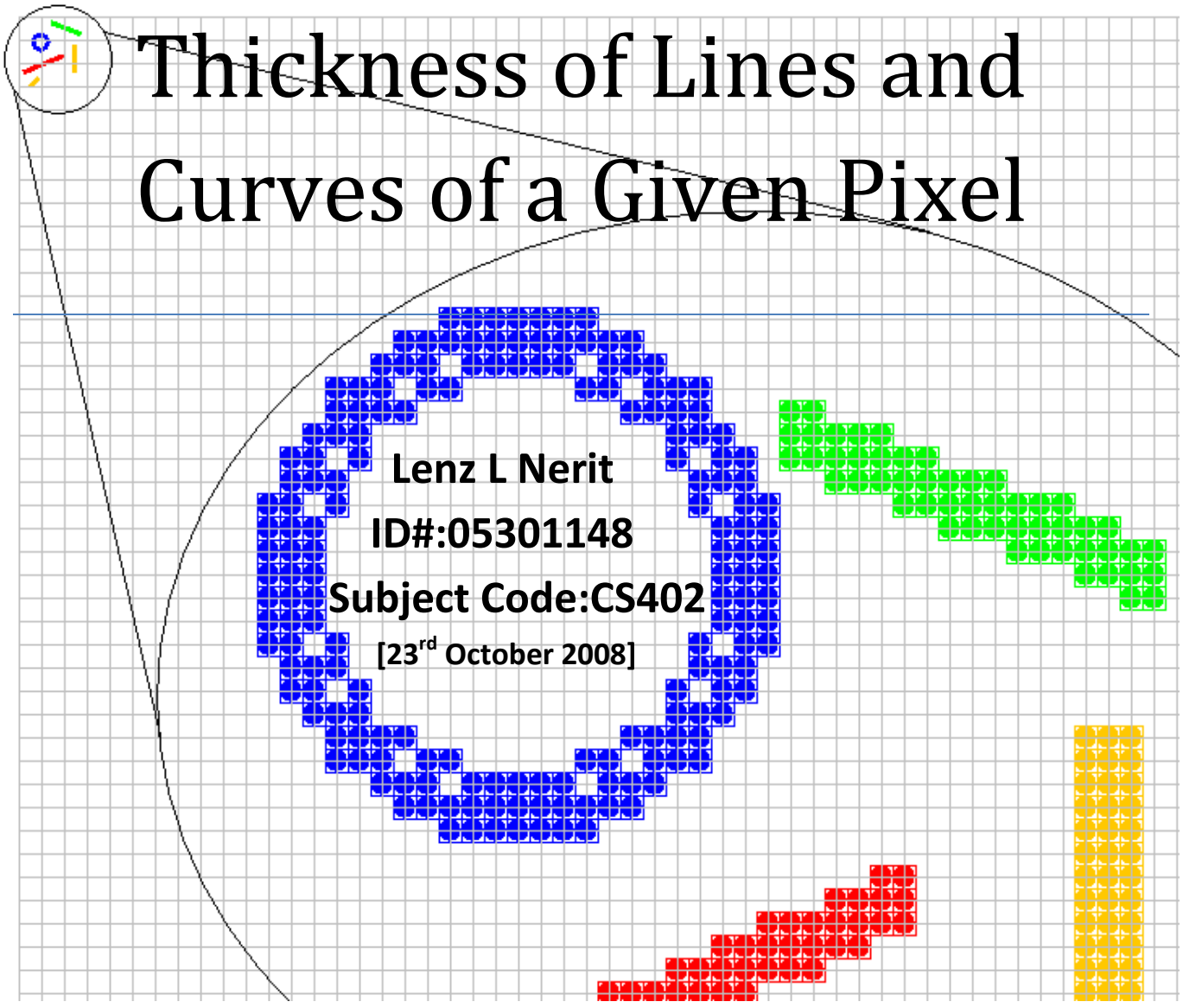
PAPUA NEW GUINEA UNIVERSITY OF TECHNOLOGY
DEPARTMENT OF MATHEMATICS & COMPUTER SCIENCE

FINAL YEAR PROJECT REPORT

Pixel Graphics

Thickness of Lines and Curves of a Given Pixel



Lenz L Nerit

ID#:05301148

Subject Code:CS402

[23rd October 2008]

[This document presents CS402,Final Year Project report.]

Table of Contents

1.0 Introduction

This project is about pixel graphics in computer image display. In particular, it is about how straight lines and curves are drawn onto the computer screen that has a discrete grid of pixels called *raster*. The objective of this project is to achieve how these single pixel lines and curves can be thickened on the raster grid of the computer screen.

1.1 Overview of Report

1.0 Introduction to Project
2.0 Background Review and Research
3.0 How Single Pixel Lines and Curves are Drawn on the Raster Grid
4.0 Determining Thickness of line or Curve of a given Pixel on the Raster Grid
5.0 A short account of Project Management
6.0 Application Development and the reflection of problems encountered.
7.0 The report finishes with a conclusion

2.0 Background Review and Research

2.1 Research

The research was carried out through two main sources, use of resources in the library (usually were not up to date) and the internet. The aim of the research was to discover the following:

- How straight lines and curves can be drawn with discrete pixels on the computer screen.
- How to set thickness of a line or a curve of a given pixel on the raster grid.

Setting thickness for lines and curve drawing is an essential part of programming in graphics. By varying the thickness of lines and outlines as we draw, we give a flat sheet of paper a meaningful third dimension.

2.2 What is Pixel and Pixel Resolution?

The word *pixel* is derived from the phrase "*picture element",* which is the basic unit of programmable color that can be displayed on a computer or in a computer image on the screen. This means that each piece of the screen can be set to a different color and intensity (or brightness). Every image on your computer is made up of colored grid of pixels.

The number of pixels that can be displayed on the screen is sometimes referred to as the *resolution* of the image though resolution has a more specific definition in some other context.
In this context, resolution refers to pixel count in a measured area of an image, which is often called *pixel per inch* (ppi).

Pixel counts can be expressed as a single number, as in a "three-megapixel" digital camera, which has a nominal three million pixels, or as a pair of numbers, as in a "640 by 480 display", which has 640 pixels along the x axis and 480 along y axis (as in a VGA display), and therefore has a total number of 640 × 480 = 307,200 pixels or 0.3 mega pixels.

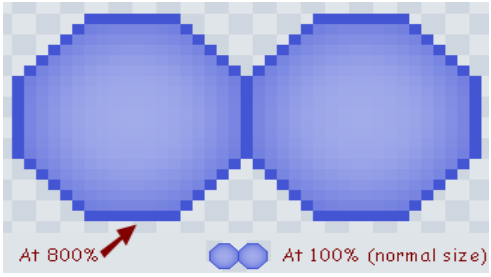


Figure 1 Rendering of the individual pixels

Figure 1 shows an image with a portion greatly enlarged, in which the individual pixels are rendered as little squares and can easily be seen. The more pixels are used to represent an image, the closer the result can resemble the original. In other words when the pixel resolution is high, there will be a sharp display on the screen but having low pixel will produce a less sharp image.

The physical unit of a pixel depends on how the resolution for the display screen is set. If the display screen is set to a maximum resolution, the physical size of the pixel will be equal to the physical size of the dot pitch of the display. However, if the display is set to resolution less than the maximum resolution, then a pixel will be larger than the physical size of the screens dots. In other words, a pixel will use more than one dot.

2.3 How are Pixels created on the Screen?

In most computer graphics systems, the basic display system is the Cathode Ray Tube (CRT).It is also called the picture tube of the monitor. The back end of the tube has a negatively charged cathode. It consists of conical shaped glass tubes attached to connector pins, an electron gun, a screen coated with phosphorus, a system to focus the electron beam and deflection plates or electromagnetic plates for controlling the horizontal and vertical positions of the beam.

The display appears when the electron gun shoots a controlled beam of electrons down the tube and strikes the specified locations on the screen. Due to the weightlessness of the electrons, the beam can be repositioned at extremely fast rates.

The output device for the graphics display is made up of tens of thousands of tiny dots. Each of the dots is a set of three of the phosphors that emits red, green and blue light. What happens is that when the CRT projects the three beams of the light, the rays pass through a mask. This mask protects the portions of the phosphors on the screen from the beam of electrons. Not only does it serve this purpose but it also helps to focus the beam so that each beam strikes only one color on the phosphor triads.
Once the electron beam strikes a point, the phosphoric surface becomes excited, thus causing that particular spot to enlighten as shown in Figure 2

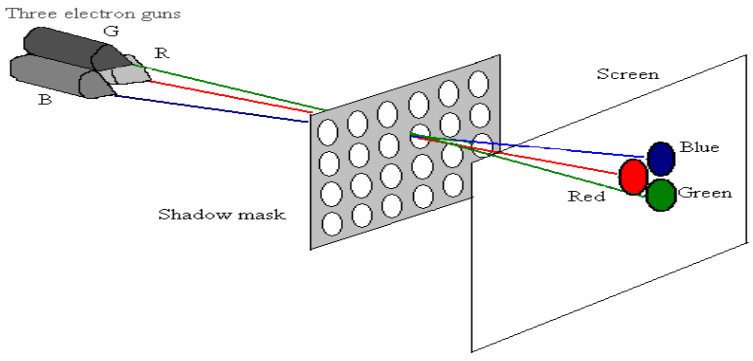


Figure 2 Display on the screen produced by the electron gun.

The production of the colors on the display screen is done by adjusting the intensity or the brightness of the electron beam. This then controls the amount of light given off by the phosphors on the screen. This is done by a display controller in the hardware by selecting a color value for each pixel and regulates the intensities of the electron beams.

The phosphor triads on the screen are grouped into dots that can be seen on the display screen. The small cluster of the dots on the screen form a larger dot called the *picture elements* or *pixels* [refer to figure 2].When the individual pixels are combined, they form what is called the *raster*, which are used to produce images and text.

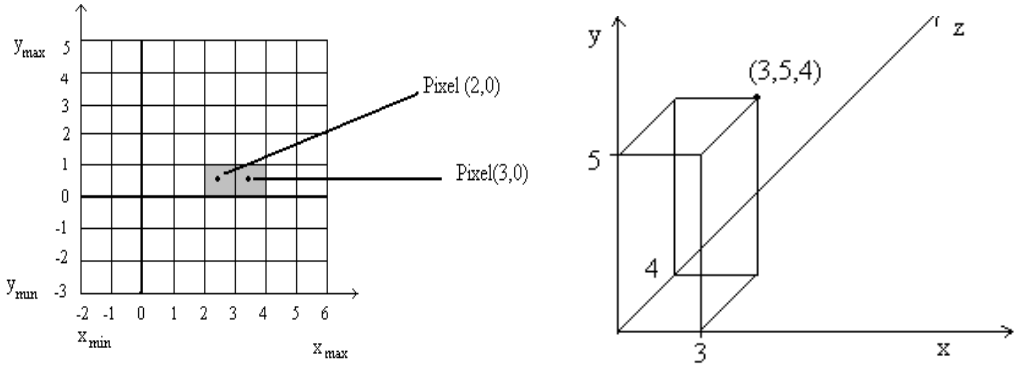
2.4 Addressing the Pixels in Raster

On the raster grid, each pixel has a coordinate point that can be identified and can be controlled or modified. An example of this could be changing a color for a particular pixel for a given address.

To fully understand how a pixel is addressed, think of a graph paper with numbered lines that are drawn on the bottom axis called the x-axis and up the vertical axis on the far left hand side on the graph paper called the y-axis. The intersection of the vertical and the horizontal lines on the graph paper forms what is called the *coordinate system.* The coordinate points are referred to as the address of a pixel called the *raster points.*

Once the address of a pixel is identified, the corresponding pixel can be illuminated by the graphics system. Figure 3 illustrates an illuminated pixel at the addresses (2,0) and (3,0).In this context, we assume that a pixel is a geometrically one unit above and to the right, though this is not a convention. However, on many computers, there are numbered grids of raster starting at the upper left corner because scan line process starts here at each refresh.

To make our life easy, we refer to the bottom left cell as the origin where the dimensions *x* and *y* start. This way of addressing pixels is applied in *2-Dimesional* space. Same principle is also applied in 3-Dimensional space except that a third axis called the z-axis is defined mathematically as illustrated in figure 3(b).



(a) (b)

Figure 3 addressing a pixel (a) on 2-D space and (b) 3-D space pixel addressing.

3.0 How Single Pixel Lines and Curves are Drawn on the Raster Grid

3.1. Use of Pixels in Line and Curve Drawing

To begin with, let's asked ourselves the question, *"Can we draw directly straight lines and curves of a given pixel on the raster grid?"* No, it is not possible to draw directly a straight line or a smooth curve from one point to another. This is because the CRT raster display can be a matrix of discrete pixels each of which can be brightened individually.

However, the lines or curves can be drawn only through the process of approximation by determining which pixels lie close to the desired line or curve .It is because of this property of pixel that makes lines or curves look jagged "stair step" and that the brightness of the lines or curves is distributed unevenly. On the horizontal and vertical axes, the lines will be brighter than the lines that are at tilted at an angle eg.45^{o} line. The overall process is referred to as *rasterisation.*

The application of the approximation of the pixels can be best established through some algorithms, which we shall look at next. Figure 4 illustrates the approximation process of determining which pixel to select in a raster grid.

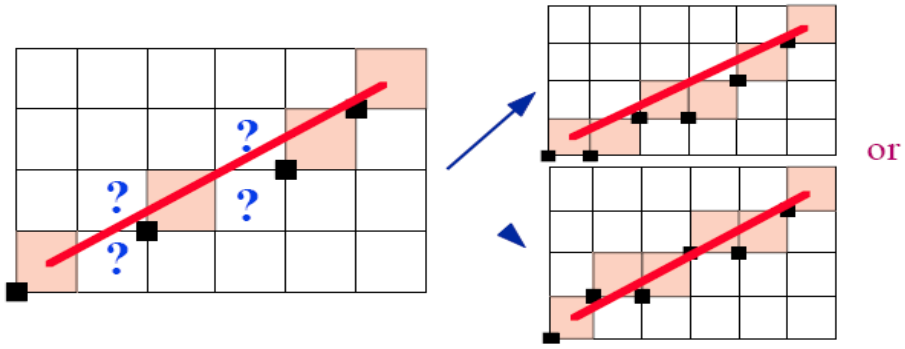


Figure 4 Use of approximation to determine which pixel to select for drawing lines.

3.2 Comparing Line Drawing Algorithms

Although, it is not possible to draw directly a straight line or a curve, it is made possible through a number of algorithms that are used in pixel drawing. In his section, I will briefly explain only three basic algorithms. These are namely:

- Direct Scan Conversion
- Digital Differential Analyzer (DDA).
- Bresenham Algorithm

In the three algorithms stated above, following symbols will be used:

- x – denote variable along the x-axis
- y – denote variable along the y-axis

- Δx – change in x variable
- Δy – change in y variable
- ε – denote error term

3.2.1 Direction Scan Conversion Algorithm

Let's consider the straight-line equation $y = mx + B$.
This algorithm performs a floating-point multiplication for every change in x. It requires a large number of floating point multiplication.
The process for the Direct Scan Conversion is stated as follows:

Let x1 be the left-hand end point pixel and x2 be the right-hand endpoint pixel

```
1. Start at the left-hand end point pixel x1
2. Move along the horizontal axis until
   we arrive at the right-hand end x2
3. For each iteration in x, compute the corresponding y value
4. Round this value to the nearest integer to estimate the pixel.
```

This algorithm is expensive since it increases the computation time for the CPU.

3.2.2 Digital Differential Analyzer (DDA)

This is another algorithm that is used for drawing a rasterised straight line. It is done by solving a differential linear equation. We shall refer to the linear equation $y= mx + B$. When we differentiate this, we get a constant as given below:

$$\frac{dy}{dx} = const \text{ or } \frac{\Delta y}{\Delta x} = \frac{y_2 - y_1}{x_2 - x_1}$$

The solution of the finite difference of the linear equation above becomes:

$$x_{i+1} = x_i + \Delta x$$
$$y_{i+1} = y_i + \frac{y_2 - y_1}{x_2 - x_1}\Delta y$$

where (x_1, y_1) and (x_2, y_2) are the end points of the required line. In fact, the given equation is a recursion relation for successive values for y in each iteration along the line. For a simple raster display, either Δx or Δy is chosen as one raster unit, depending on whichever is larger. In the

algorithm, the constant term *m* is calculated only once for that recursion process. The algorithm runs slowly in each iteration since it deals with floating point number arithmetic.

3.2.3 Bresenham's Line Drawing Algorithm

Another line drawing algorithm is the Bresenham's algorithm, which was originally developed for digital plotters. However, it has now become an essential tool for dealing with raster displays. The algorithm tries to pick the optimum raster points to form a straight line. It is achieved by incrementing x or y by one unit depending on the gradient of the line. The key idea in this algorithm is that, the increase in one of the variables by either zero or one unit is determined by the distance between the actual line location and the nearest grid locations (midpoints) called the error term denoted by ε . Figure 5 illustrates the basis for Bresenham's algorithm for drawing lines on a raster grid.

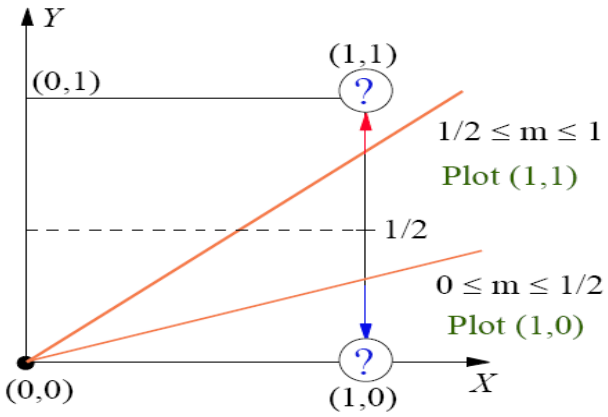


Figure 5 Basis of Brehensen's algorithm

In the diagram above, if the required line passes through (0,0) has a slope greater than ½ , then intercept x=1 will be closer to the line y=1.Thus the raster point that be best selected will be at point (1,1) rather than the point (1,0).

On the contrary, if the slope of the required line is less than ½, then the intercept x=0 will be closer to the point (1,0) which would be chosen as raster point. To illustrate this idea in full detail, let's draw a line between two end points and that will always draw pixels on the endpoints. Let's assume that the slope of the line is -1<m<1 and that x1<x2.

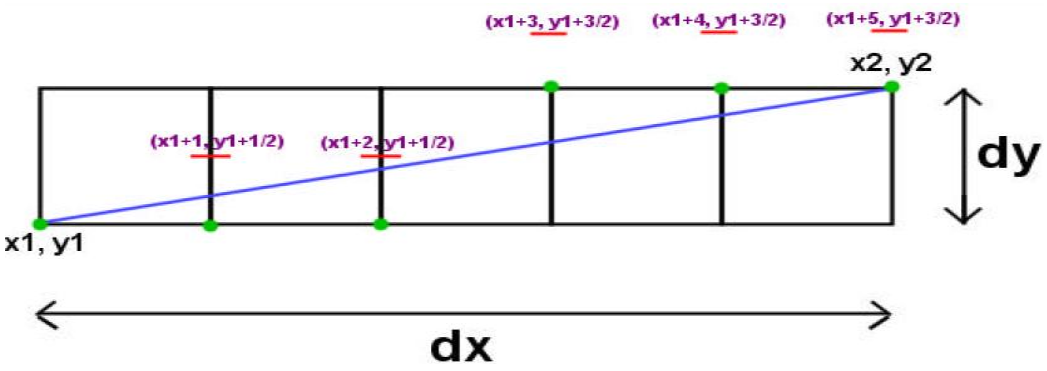


Figure 6.Selecting Pixels on the midpoints

The above picture is a good representation of how one might choose the pixels to light up. The blue line is the "true" line between the endpoints, but we approximate to the green pixels (dots) based on the relative positions of the blue line and the midpoints. Namely,

```
if (Blue Line<Midpoint)
      Plot_East_Pixel ()
else
      Plot_NothEast_Pixel ()
```

However, what would it be if we were given a point and a line? How would we know whether the line is below or above that line? To answer these questions, look at the straight-line equation.

$$y = \frac{dy}{dx}x + B$$

$$dx * y = dy * x + dx * B \qquad \text{rewritten}$$

$$0 = dy * x - dx * y + dx * B$$

Now, what this implies is that anywhere on the line, the left side of the equation equates to zero. You will also notice that at any point (x, y) above the line, the right side of the equation becomes a negative number and the opposite is true for any point (x, y) below the line. This lets us to define a function F:

$$F(x, y) = 2dy * x - 2dx * y + 2dx * B$$

such that for any point (x,y) that represents a midpoint between the next pixel to the right (East pixel) or the Northeast pixel,

$F(x, y) < 0$, when Blueline < Midpoint

$F(x, y) > 0$, when Blueline > Midpoint

Using our equation F(x,y), the choice to which pixel should be brighten whether (x1 +1,y1) or (x1+1,y1+1) can be determined from the first midpoint given (x1+1,y1+1/2).

This is illustrated as follows.

$$F(x_1+1, y_1+1/2) = 2dy*(x_1+1) - 2dx*(y_1+1/2) + 2dx*B....(1)$$

i.e

$$F(x_1+1, y_1+1/2) = 2dy*x_1 + 2dy - 2dx*y_1 - dx + 2dx*B$$

Now consider the point (x1, y1)

$$F(x_1, y_1) = 2dy*x_1 - 2dx*y_1 + 2dx*B = 0.....(2)$$

Therefore, equation (1) becomes

$$F(x_1+1, y_1+1/2) = F(x_1, y_1.) + 2dy - dx$$

$$F(x_1+1, y_1+1/2) = 2dy - dx$$

The reason why the factor 2 is introduced is that we want to eliminate the use of floating point numbers. Because each midpoint falls halfway between two pixels, and simply by multiplying both sides of that equation by 2 to eliminate division so that we can get an equation that uses only integers. This introduces the Bresenham's integer algorithm. The integer version of the pseudo code is stated below.

```
Void drawLine(int xP,int yP,int xQ ,int yQ)
{
    Start
    Declare and initialise the variables
            int x=xP,y=yP, ε = 0 ; Δx = xQ − xP , Δy = yQ − yP
            c = 2 * Δx ,
            M = 2 * Δy ;
            Loop
            {
                putPixel at point (x,y)
                If(x==xQ) break;
                Increment x value
                Increment the error term ε by the sum of M
                If ( ε > Δx )
                {
                    Increment the y value
                    Decrement the error term ε by the value c
                }
            } end of loop
    Stop.
}
```

3.2.4 Incremental Digital Display of Circular Arcs

Bresenham Algorithm for line drawing also applies for circular curve drawing.Rastering of the conic sections such as circles, ellipses, parabolas, hyperbolas and others of these sorts has been given a considerable attention. In fact, circle achieved the greatest attention of all. The easiest way to generate an algorithm is based on the following diagram.

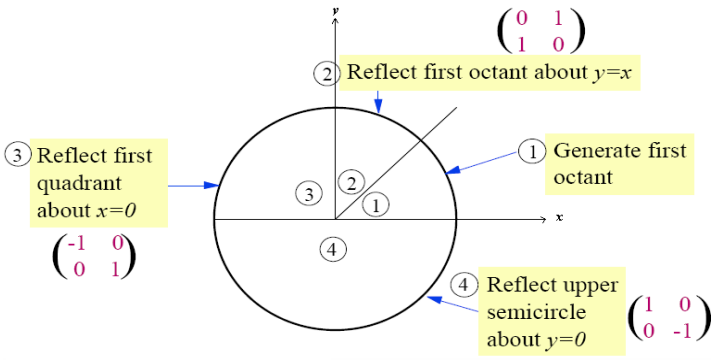


Figure 7. Generation of Complete circle from the first octant

In circle drawing, the key idea to compute the initial octant only from the equation $x^2 + y^2 = r^2$. This can then be reflected in other octants of the circle. If the initial octant is the first octant (0-45^o) that is generated, the second octant can be obtained by reflecting y=x .In doing so, it yields the first quadrant. The results in the first quadrant can be reflected through x=0 to obtain the second quadrant. The combination of upper quadrants can be reflected through y=0 to obtain the full circle.

The decision of choice to select which raster points to plot on the grid can be done by the mid-point algorithm. This is done by transforming the equation of the circle to $0 = x^2 + y^2 - r^2$ so that r will be calculated only once during the initialization. This zero in the transformed equation is then replaced by the error term ε .

The error term is initialized to offset ½, which is derived from the pixel at the start as illustrated below. The offset value ½, in fact, is the midpoint value of any given two raster point.

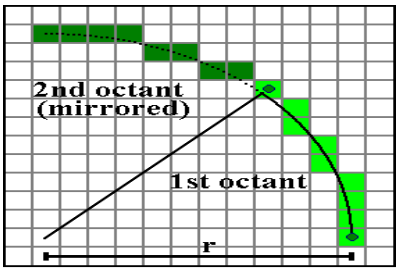


Figure 8. Raster curve drawn in the first quadrant

In each iteration, the value of r accumulates in the error term until it intersects with the perpendicular line. When it reaches this point, its value at this point is used for initializing. Below is the pseudo code to generate an arc in the first quadrant.

To make it clear, three new variables u,v and E were introduced. These are non-negative integer variables denoting the difference between two successive squares and the "error" respectively. The reason for this was to avoid computing the x^2 and y^2 so that the algorithm will work faster.

$u = (x-1)^2 - x^2$ ……..(1)

$v = y^2 - (y^2-1)^2 = 2y-1$……...(2)

$E = x + y^2 - r^2$ ……...(3)

Initially we have, x=0, y=r so that u=1 and v=2 * r-1 and E=0.In each step in the loop, we increase x by 1.Since, according to equation (1), this will increase the value of x^2 by *u*, we also have to increase E by v to satisfy the equation (3).Note also that increasing x by 1 implies that we have to increase u by 2.

Now we have to decide whether or not to decrease y by 1.If we do, the square y^2 decreases by v as indicated in equation (2) so that we also have to decrease E by v.This can be only achieved if and only if the test $v < 2*E$ is successful. On this basis we can write the pseudo code for an arc as given below.

```
Void arc(int x,int y,int r)
{
    Start
    Declare and initialize the variables
        int u=1,v=2*r-1,E=0;
        x=0;
        y=r;
        while(x<=y)
        {
            putPixel at point(x,y)
            Increment x
            Increment the E by u
            Increment u by a factor 2 each time through the loop
            if (v<2*E)
            {
                Decrement y value
                Decrement E by value of v
                Decrement v by a factor of 2
            }
        }
    End of while loop
}
Stop
```

4.0 Determining Thickness of line or Curve of a given Pixel on the Raster Grid

In mathematical terms, lines do not have thickness. However, they are at least one pixel in wide. The algorithms described earlier are used for drawing lines of one pixel wide. This difference in the notation of lines may not cause any problems if the pixels are very small in comparison with what we are drawing

Drawings can distil images of objectives into a relatively small number of simple lines. In doing so, they echo the way our brain simplify and interpret the raw data and arrives every second at our retina.

In order to produce a thickened line, a number of single thickness lines can be drawn in parallel. There are two ways of doing this:

1. Draw lines perpendicular to the ideal line stepping along it.
2. Draw lines parallel to the ideal line and step perpendicularly.

4.1 Thickness of a Line and a Curve

Line thickness is specified as a constant and this can be measured only in two orthogonal directions. That is between orthogonal and diagonal lines. Due to the fact that line thickness is defined as a fixed number of pixels (picture elements), lines of a varying pixel cannot be drawn. The effect of this results in the production of diamond-shaped ends for orthogonal lines and square-shaped ends for the horizontal lines.

To achieve a line of having given thickness, the requirement is to generate all points which are within a distance T_L to the left of the line AB and within a distance T_R to the right of the line AB as depicted in the diagram below.

(*Adopted from http://homepages.enterprise.net/murphy/thickline/*)

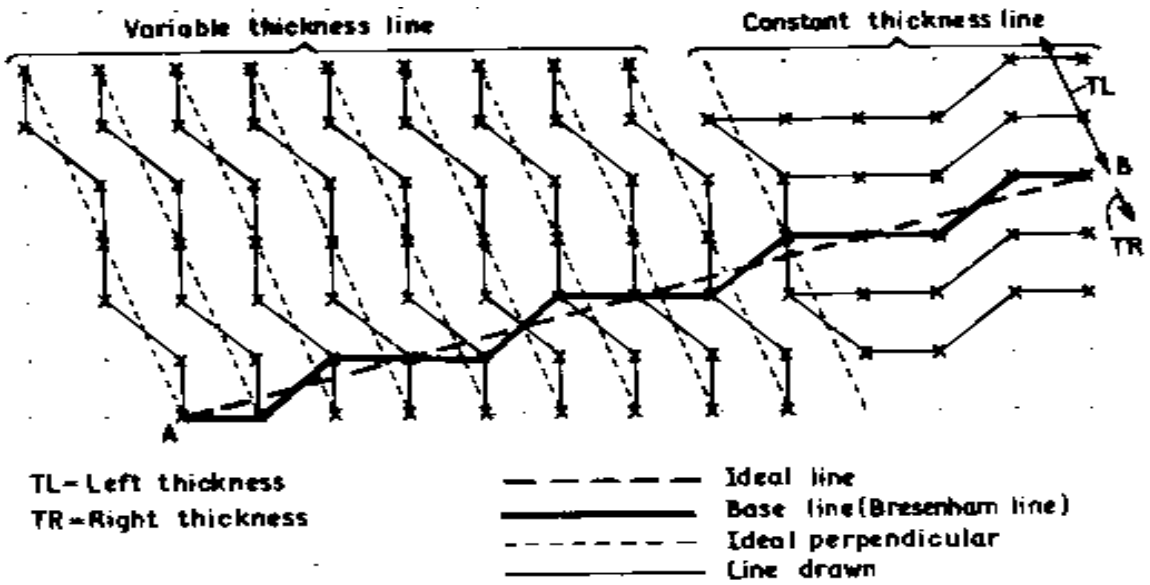


Figure 9 Diagram showing the requirements to generate thickness for line thickness

For normal constant-thickness lines T_L=T_R= constant. However, in the general case,
T_L =Fl (L) = function of line length L and
T_R =Fr (L) = another function of line length L

The concepts above can be further explained in terms of the diagram given below.

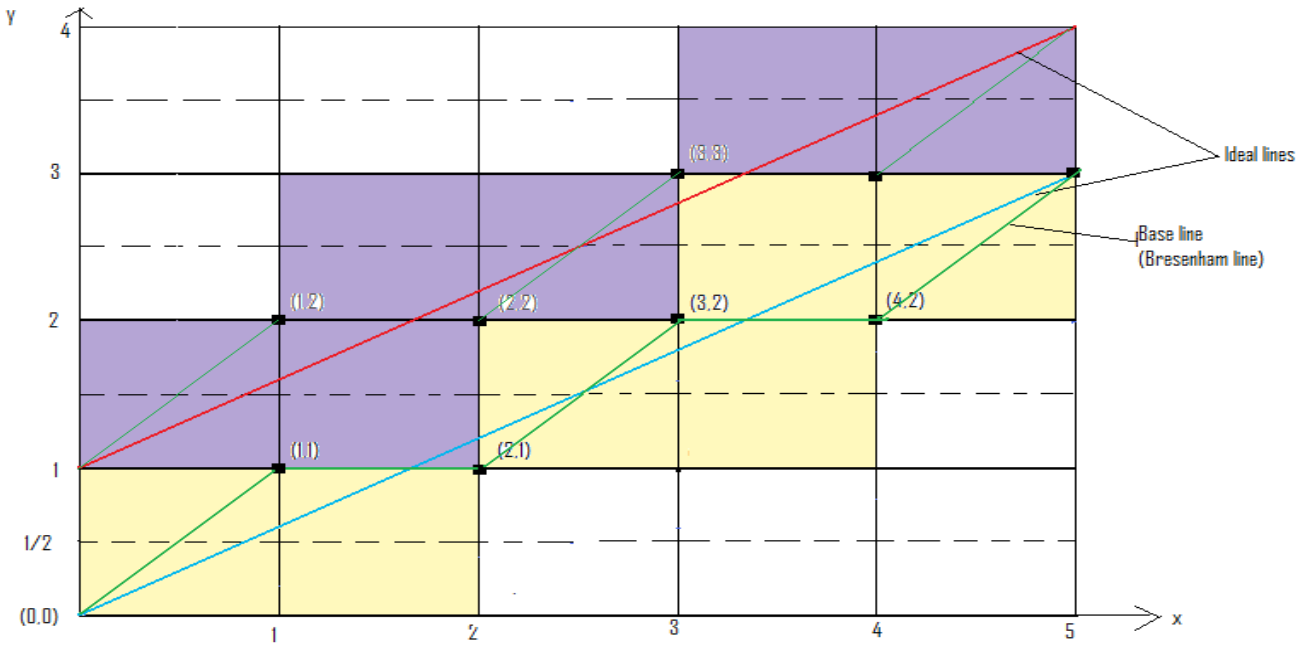


Figure 10 Illustration of how a line of given pixel thickness can be produced

In Figure 10, you can see that two single pixel lines are drawn parallel to each other as labeled ideal lines, one blue line (bottom) and the other is the red line (top).Along each line, the nearest raster points that lie close to the lines were selected (black dots) and the raster grids were shaded as indicated in the diagram above. Along the bottom line, the raster grids were shaded in orange and along the top line, the raster grids were shaded in purple to produce a line that has thickness of two pixels.

To determine the thickness along the curve, perpendicular Lines are drawn to the curve at all points along the curve.
In the thick line algorithm, if a traverse line is added in parallel to it, a number of effects would be achieved. For instance, a dashed or dotted lines could be produce, if a normal line in applied in parallel to the line direction. If a secondary line pattern is applied perpendicular to the line, then a variety of effects can be obtained. A traversed dot pattern will produce a set of parallel lines. The two edge pattern of the thickened line can be drawn by setting an appropriate transverse line pattern.

5.0 Project Management

This chapter briefly describes the planning made at the start of the project and the time management involved during development.

5.1 Research Plan

Originally, the research was planned out as stated in the table below. However, due to external problems the schedule was not strictly adhered to.

The practical part of the project can involve a lot of unexpected events that need to be accounted for in the best way possible whilst planning time. The following steps were taken at this stage:

1. Listing of all major development jobs to be done.
2. Break down of these jobs into smaller tasks.

Week	Task Scheduled	Task Completed (Y/N)	Comments
1-7	Detailed theory research into the field of pixel Graphics.	Y	Fully completed
	Discover what is pixel graphics and its use, how it is generated	Y	Fully completed
	The techniques used to draw images onto screen.	Y	Fully completed
	Also, discover the best approach into achieving the aim of producing thickness of a line or a curve	Y	Research into producing thickness of lines and curves.
8-11	Development (coding) of the application prototype. This was to simulate how pixel is used in graphics display thickness of line and curve drawing.	Y	Implemented the line drawing algorithm (Bresenham algorithm) for single line thickness. Setting thickness of lines & curves was achieved by drawing single pixel lines in parallel.Also used setStroke method in Java to produce thickness of lines of given pixels
12-14	The Documentation of the report	Y	Fully completed

Figure 11. The overall research plan schedule.

6.0 Application Development Process

Specifying thickness of a line or a curve in graphics context has, to an extent, been analyzed as it has been introduced in the project. This was done in the interest of taking the reader through a chronological account of the project. This section is therefore to bring the previous discussions together, evaluate the end-result and look towards how the aim was achieved in the area of setting thickness of line or curve drawing for a given number of pixels.

6.1 Problems Tackled

A number of problems have been faced, as I progressed through the development phase of the project. Here is a short account of the major problems encountered and the reasoning behind the solutions employed.

The main problem had been the "power-cuts" or the power interruptions. The continuous interruptions to the power caused setbacks to the progress of the project. In this situation, I had to work on the project using the power from my laptop battery.

There had also been interruptions to my project schedule due to the weeklong strike that was organized on the campus. To make up for the number of hours lost during this time, I had to spend extra hours on the project.

6.2 Finding and Testing

It was discovered from this research that graphics programming can be done in any programming language. Most of the languages were equally compatible with the graphics library. The languages could be both high-level languages such as Java, Pascal, COBOL, C/C++ and also low level languages such as assembly language (asm).

In my project I did an application prototype of graphics programming in Java. In Java objects used for rendering on the screen are based on the graphics context. The objects are already defined in the library. What I had to do was to import the appropriate classes from the packages that were defined in the library.The packages that I imported from the library were mainly the *abstract windowing toolkit* and the *swing*.

```
import javax.swing.*;
import java.awt.*;
```

With regard to the use of the line drawing algorithms, I implemented the *Bresenham's Integer line drawing algorithm* as it was efficient over the other line drawing algorithms. This was due to the fact that it uses integer values to determine the nearest raster points. In general integer operations are faster than floating point numbers.

To produce a line or curve of a given thickness, say 10 pixel lines, what I did was to draw single thickness lines in parallel and step along perpendicularly and step along selecting the nearest pixel points. In doing so, a line with desired thickness was produced.

To achieve this, I used *for loop* statement. Below is a cut down version of the original code. In the piece of code below, the variables *(Orx, Ory),(mousex and mousey)* were declared as integer types. In fact the variables represented the coordinate points of the mouse on the application interface.

```
 1 public void releasedLine()
 2         {
 3                 Graphics g = drawPanel.getGraphics();
 4                 int pixel = Integer.parseInt((String) combo.getSelectedItem());
 5                 if ((Math.abs(Orx - mousex) + Math.abs(Ory - mousey)) != 0)
 6                 {
 7                         initialLine = true;
 8                         if (Math.abs(Orx-mousex)!=0)
 9                         {
10                                 for(int i=1;i<=pixel;i++)
11                                 {
12                                         drawLine(g,Orx, Ory+pixel-i, mousex, mousey+pixel-i);
13                                 }
14                         }
15                         else
16                         {
17                                 for(int j=1;j<=pixel;j++)
18                                 {
19                                         drawLine(g,Orx+pixel-j, Ory, mousex+pixel-j, mousey);
20                                 }
21                         }
22                 }
23
24         }
25
```

Due to the fact that drawing in Java is done in graphics context, in line 3 of the code, I have declared a Graphics object which I could use to draw on the panel object called *drawPanel*.

In line 4, the String object from the combo box is converted to integer using the *pasrseInt* method from the *Integer* wrapper class. The converted value is then assigned the variable *pixel* of type integer which set the maximum value for the number of parallel lines to be drawn. In fact, on the application interface, the user would specify the thickness of lines and curves by the selecting a number from the combo box.

The statement at line 5, tests for existence of a line between any two points (Orx, Ory) and (mousex, mousey).If a line was drawn then the nested statements in the brackets are executed. The inner nested *if- statement* at line 8 tests to see if the there was any change along the x-axis. If there exists an absolute difference along the x-axis, that indicates that there exist a line so line 10 would be executed so that loop would draw parallel lines specified by the *pixel* value one unit below the i^{th}

line, selecting the nearest raster points. In other words, the parallel lines drawn would have only y-values incremented but the x-values at each raster point will remain constant.

Similarly, if the *else-statement* at line 15 holds true, then the *for loop* at line 17 would draw parallel lines one unit below or the j^{th} line, selecting the nearest raster points. For lines that are drawn in the vertical direction, the lines would be drawn parallel to the right of the initial line. The interval difference between the parallel lines would occur for x-values but the y-values would remain constant.
The *drawLine()* methods stated at line 12 and 19 is the Bresenham method for drawing lines which I implemented in the code. In Java, there is also a predefined method to draw line with the same method name as I have defined in my program. However, the later can only allow four parameters to be passed. The parameters passed can be of type *int* or floating point numbers (*float*).The former could take in five parameters of type *int*.The methods are stated below.

```
drawLine(int xP,int yP,int xQ,int yQ){
}
 public void drawLine(Graphics g,int xP,int yP,int xQ,int yQ){
 }
```

Though they have one method name with different arguments, their usage would also be different when drawing lines. The predefined has the following usage on the graphics context:

`g.drawLine(xP,yP,xQ,yQ);` where g specifies a graphics object.

Note that in the line drawing method that I have defined, I have introduced a Graphics object *g* as another parameter. In the usage, the object of type Graphics would be only passed as a parameter. In order to draw lines using this method, it is called from the other code segments.

```
drawLine(g,xP,yP,xQ,yQ);
```

The snapshot of the thickened lines drawn using the *Bresenham drawLine* method is shown in appendix C. It shows the illustration of how lines are drawn on the grids of discrete pixels.
In fact when line or curve is drawn on the raster grid, you would notice that some raster points may not be selected. This is due to the fact that only the points that line close to the ideal line were selected as we step along. This property can be seen clearly in the three-pixel circle drawn in Appendix C.

In the context of drawing producing thickening lines and curves, I also defined another method to set thickness to lines and curves. I used a *setStroke()* method. The Graphics2D context determines how to draw outline of a shape based on the current *Stoke.* This method sets the drawing Stroke to the behavior defined by pen. A user-defined pen must implement the *Stroke* interface. The AWT includes a *BasicStroke* class to define the end styles of a line segment, to specify the joining styles of line segments, and to create dashing patterns.

To control how lines are drawn, first create a *BasicStroke* object, then use the *setStroke* method to tell the Graphics2D object to use the *BasicStroke* object.

The code segment below illustrates how I applied this method.

```
1 public int setThickness(Graphics2D g)
2   {
3      int size = Integer.parseInt((String) combo.getSelectedItem());
4      g.setStroke(new BasicStroke(size));
5      return size;
6   }
```

At line 3, the item (value) selected from the combo box from the user interface is converted to and integer value using the wrapper class *Integer* and the *parseInt* method. The value is assigned to the integer variable *size*. The value of this variable is passed into the *BasicStroke* method (at line 4) to determine the thickness of the pen which produces the lines with desired number of pixels thick lines and curves. The snap shot of the thickness patterns of lines and curves produced using this (*setThickness()*) method is shown in appendix B.

The application was successfully tested and was demonstrated.

7.0 Conclusion

The drawing of lines and curves on a raster grid involves discrete pixels that are accessed individually. This means that lines or curves can be drawn by the best possible approximation of an ideal line given the inherent limitations of a raster display. This further implies that it is not possible to draw directly straight lines or smooth curves. As a result of this, the line or curve appears jaggy "staircase".

In most images we draw, you will notice that the lines are used to represent edges of the object. Furthermore, you would notice that they (images) are composed of relatively a number of small lines.

Setting thickness to lines or curves on the raster grid employs this idea. This implies that when we want to set thickness to line, a number of single line pixels must be drawn in parallel to the ideal or the true line and then step along by selecting the nearest raster points.

8.0 References

1. Rogers David F. Procedural Elements for Computer Graphics, New York: McGraw-Hill, 1985.
2. Sun Microsystems. An Introduction to Computer Graphics Concepts(From Pixels to Pictures).New York, Addision Wiley Publishing Company,Inc.,June 1991
3. Garry Marshall. Programming with Graphics. Great Britain, Granada Publishing, Inc., 1991.
4. David R. Clark. Computers for Image Making (Audio-visual media for education and research; vol.2), Great Britain, Pergamon Press, 1981.
5. A Wiley-Interscience Publication. Computer Image Generation. New York, John Wiley & Sons,Inc.,1983
6. G.Enderle, K.Kansy, G.Pfaff.Computer Graphics Programming (Symbolic Computer. Computer Graphics), Germany, Springer-Verlag, 1984.
7. Dennis Harris. Computer Graphics and Applications, New York, Chapman and Hall, 1984.
8. Vera B. Anand.Computer Graphics and Geometric Modeling for Engineers, Garamond, Von Hoffmann Press, September 1992.
9. Theo Pavlidis.Algorithm for Graphics and Image Processing.Rockville, USA, Computer Science Press, 1982.

10. http://en.wikipedia/wiki/Computer_graphics/
11. http://en.electronics.howstuffworks.com/
12. http://en.wikipedia.org/wiki/Raster_graphics
13. http://en.wikipedia.org/wiki/Cathode_ray_tube/
14. http://searchio-midmarket.techtarget.com/sDefinitions/
15. http://homepages.enterprise.net/murphy/thickline/

Appendix A: Definition of Terms

Algorithm – a logical sequence of steps for solving a problem, which can be translated into a computer program.

Color – Color is used here in the full scientific sense, embracing both luminance and chromaticity.

Coordinate point – The specific points of the x-axis and its corresponding value of the y-axis on the screen forms what is known as the *coordinate point.*

CRT displays – refer to the displays shown by Cathode Ray Tubes. It is done by projecting electron beams on the screen coated with chemical called Phosporus.When the beam strikes the screen, it excites the phosphors on the screen, thus emitting light and displays appear.

Display – refers to what is outputted onto the screen as image such as a pictures or drawings.

Image – refers to what is shown on the screen

Monochrome – means it can display only two values at each pixel on the screen, which can be blank and white or black and green or black and amber.

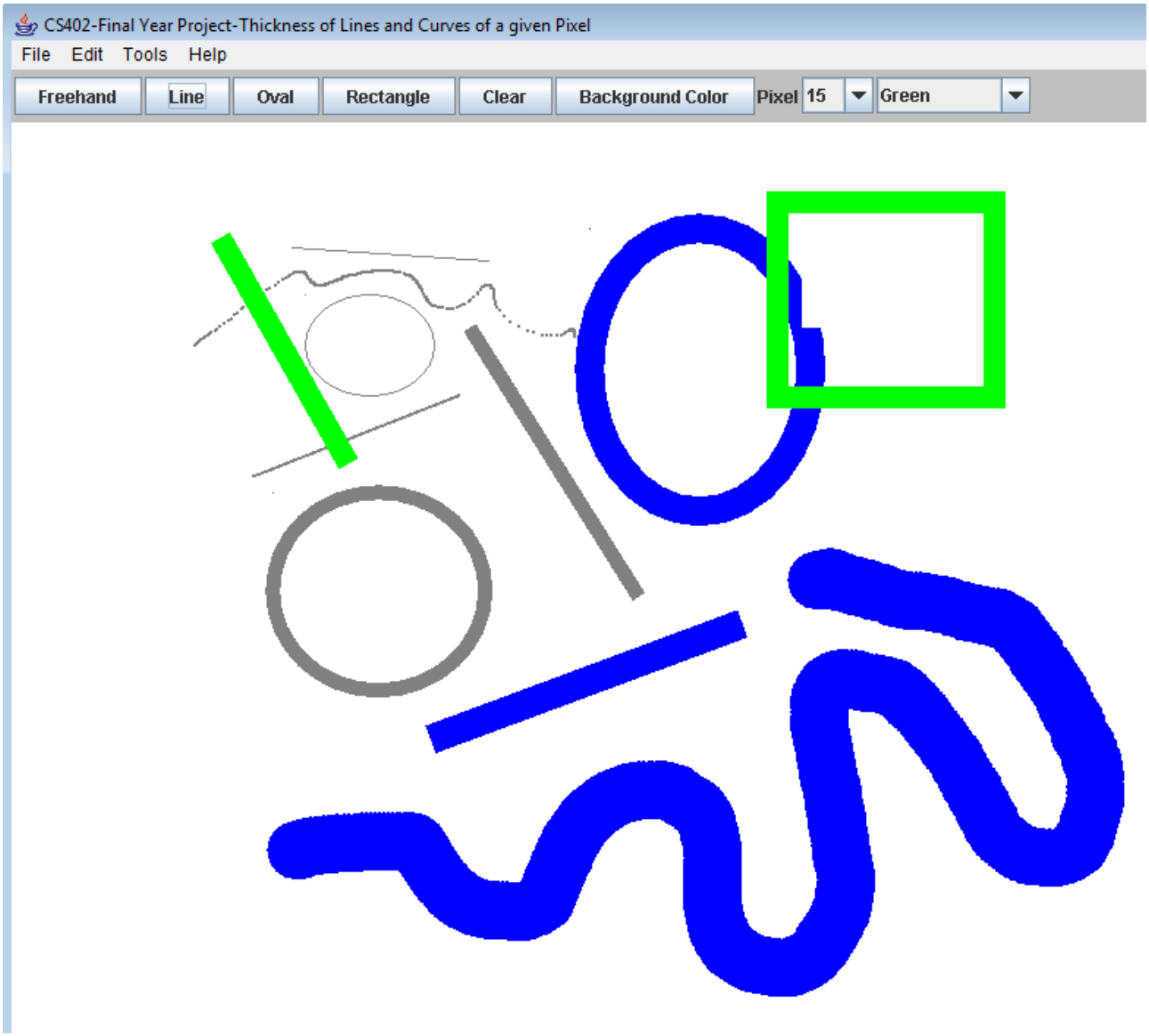
Pixel – is described as the smallest piece of the screen that can be controlled individually.

RBG – is a color model that is based on combining the primary colors Red, Blue and Green to produce other colors

Three-Dimensional (3-D) image – image displayed in x, y and third axis called a z-axis planes. An image with width, length and height.

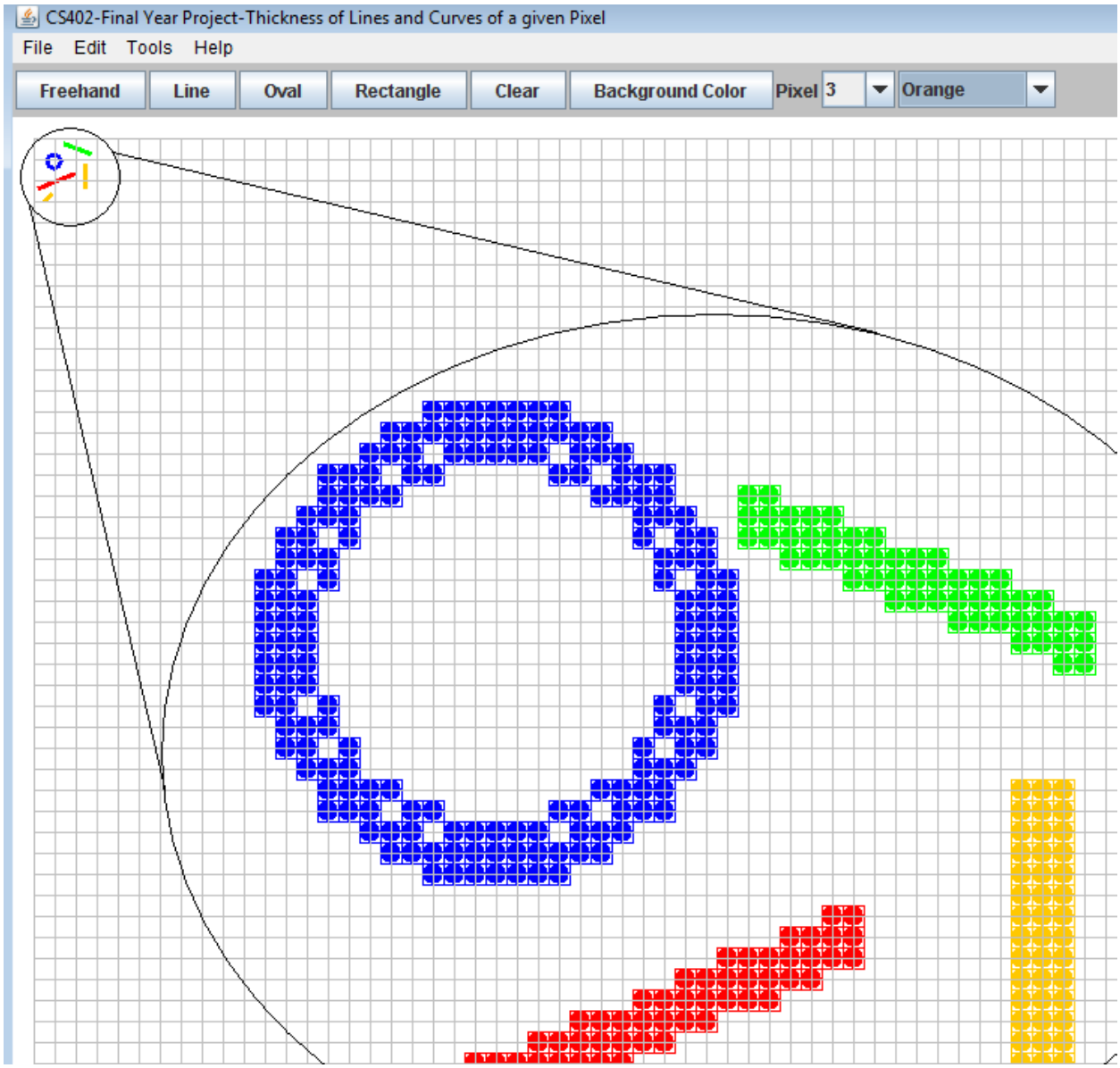
Two-Dimensional (2-D) image – image displayed in the x and y planes. An image with width and length only.

Appendix B: Different pixel thickness lines and curves drawn on the interface.



On the drawing window (panel), lines and curves were drawn with different thickness specified in the combo box. Here, you could see clearly a 15 pixel thick lines and curves appear thicker than those that were drawn with less number of pixels.

Appendix C: Illustration of the actual implementation of thickening lines and Curves using the Bresenham algorithm.



On the top right hand corner of the window shows what the actual lines would look like on a high resolution screen.Those lines are actually three pixels thick. In the foreground, you can notice how 3 pixels lines can be produced on a raster grid.