**Department of Computer Science and Engineering**

| Course Code: CSE 423 | Credits: 1.5 |
|---|---|
| Course Name: Computer Graphics | Semester: Fall'20 |

**Lab 03**

**Mid-point Line Drawing Algorithm**

## I.  Topic Overview:

The students were introduced to DDA line drawing algorithm in the previous class. Given, the co-ordinates of any two points on the line: $(x1, y1)$ & $(x2, y2)$, the student's task is to find all the intermediate points required for drawing the line on the computer screen of pixels where every pixel has integer coordinates. The DDA algorithm works fine but is slow due to floating point arithmetic.

Therefore, we would be introducing mid-point line drawing algorithm in today's class to speed things up a little bit by using integer arithmetic. This algorithm tries to find the best approximation of the next point on the line using some criteria. The benefit of using approximation over exact points is to speed up the whole drawing process - making it particularly suitable for practical implementation. To keep things simple, we would make the following assumptions:

*   The line will be drawn from left to right
*   For the given co-ordinates $(x1, y1)$ & $(x2, y2)$: $x1<x2$ & $y1<y2$
*   Slope of the line is between 0 & 1 (Zone-0), i.e., we will be writing a program to draw a line from left bottom to top right.

However, a line can have any slope $(-1 \leq m \leq 1)$ and any direction. Therefore, we need to adjust our program so that it can handle every single case of a line. We can consider a given line can be in any one of the 8 zones in the following figure. We would be implementing mid-point line drawing algorithm only for Zone 0, but would adjust our program so that it can draw any line in any zone.

The students might think that they could implement separate methods to draw lines in 8 different zones. Though the methods are supposed to be quite similar to each other, it's a lot to have 8 separate methods to draw a line!
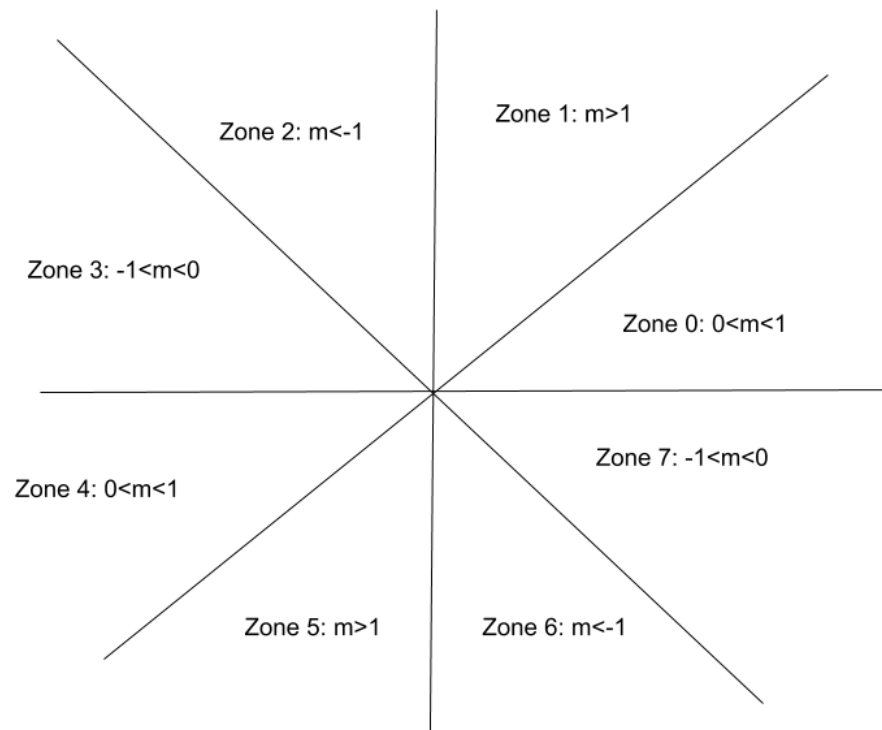


Couldn't we get rid of all these redundant methods & have only one method that would handle lines in all 8 zones? It would make things more compact, flexible & robust. In today's class we will introduce that idea to the students!

We will achieve our goal of using a single method to draw lines by manipulating a special property of lines: **"Eight way symmetry".** The key idea is that we will use Zone-0's line drawing method to draw *any line in any zone.* For this purpose,

- First we need to map any point in any zone to a point in zone 0 [**Convert point in any zone to point in zone 0**].

- Then we will simply use Zone-0's line drawing method to calculate the intermediate points representing the line [**Run Zone 0's line drawing method**].
- Finally, before drawing the pixels on the screen, we need to convert back the points in Zone 0 to its original zone [**Convert back point in Zone -0 to a point in its original zone**].

## II. Lesson Fit:

a. Experiment 3: Implementation of Mid-point line drawing algorithm using 8-way symmetry.

b. First of all, the students should have a clear understanding of DDA line drawing algorithm so that they can understand why we need to implement a new algorithm!

c. The students should have a clear idea about the slopes of lines in different zones.

d. They should also have a clear idea about the co-ordinates of points in different zones.

## III. Learning Outcome:

After this lecture, the students will be able to:

a. Learn how to use a metric to determine the next pixel on the line rather than using the line equation to get the exact measure of the next pixel.

b. Learn why it is acceptable to use approximation over exact calculation to enjoy faster computation at the cost of reduced precision.

c. Learn how to map a problem to an instance of another problem for which solution exists. And after getting the solution they will know how to convert it back to its original instance.

d. Learn that they need not write new methods every time for a new problem. They can re-use existing methods to solve a new problem. And this intuition and knowledge about reusing existing solutions to solve a new problem is important & meaningful in many contexts.

## IV. Anticipated Challenges and Possible Solutions

a. The students need to carefully convert point from any zone to a point in zone 0. They also need to convert back the points to their original zone before drawing the pixels. Most of the time, the students aren't careful during conversion & make mistakes!

**Solutions:**

    i. Have a sound idea about the coordinates & slopes in different zones.

b. The students should write their algorithm in a way so that the drawing of the line doesn't become dependent on the order of end points of a line. For example, it should not be the case that the implementation of the algorithm draws line properly for $(X_1, Y_1)$ and $(X_2, Y_2)$ points but fails when points are given in reverse order $(X_2, Y_2)$ and $(X_1, Y_1)$. This should not be the case.

**Solutions:**

    i. Swap!

## V. Acceptance and Evaluation

Students will start implementing the algorithm after we finish our lecture. If a student fails to complete the implementation he/she will have to complete by that night and show it to the class teacher on his/her consultation period. If a student also fails to do that then instead of 10 we will evaluate the student on 7.

1. Lab evaluation marks: **out of 10**
2. Late Lab evaluation marks: **out of 7**

## VI. Activity Detail

a. **Hour 1:**
   **Discussion:**

   During the first hour of the class the teachers will discuss the mid-point line drawing algorithm in detail. The basic idea is as follows: For any given/calculated previous pixel $P(X_p, Y_p)$, there are two candidates for the next pixel closest to the line, $E(X_p+1, Y_p)$ and $NE(X_p+1, Y_p+1)$ (**E** stands for East and **NE** stands for North-East).
   In Mid-Point algorithm we do following.

1. Find middle of two possible next points. Middle of $E(X_p+1, Y_p)$ and $NE(X_p+1, Y_p+1)$ is $M(X_{p+1}, Y_p+1/2)$.
2. If M is above the line, then choose E as next point.
3. If M is below the line, then choose NE as next point.
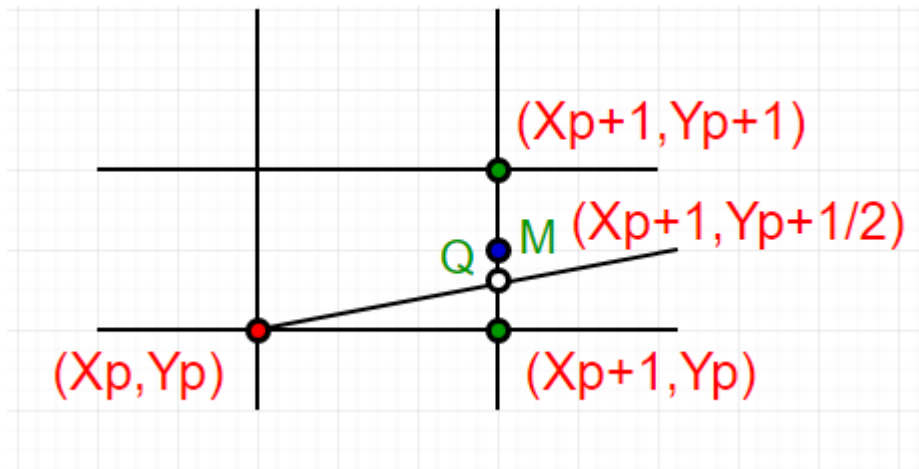


**How to find if a point is above a line or below a line?**

```
Let us consider a line y = mx + B.

We can re-write the equation as :

y = (dy/dx)x + B or

(dy)x + B(dx) - y(dx) = 0

Let F(x, y) = (dy)x - y(dx) + B(dx)    -----(1)
Let we are given two end points of a line (under
above assumptions)
-> For all points (x,y) on the line,
      the solution to F(x, y) is 0.
-> For all points (x,y) above the line,
      F(x, y) result in a negative number.
-> And for all points (x,y) below the line,
      F(x, y) result in a positive number.
```
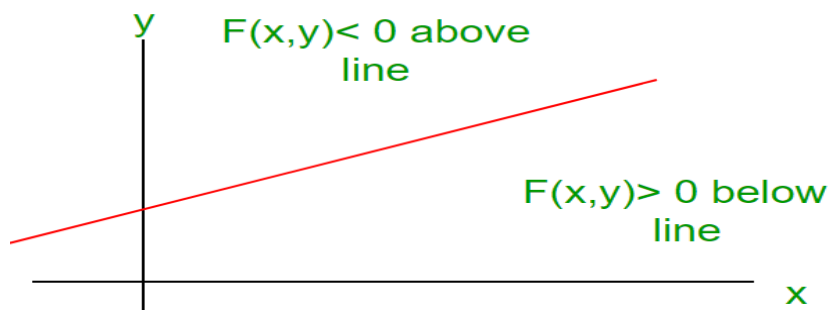


This relationship is used to determine the relative position of M.

The algorithm works as follows:

```
DrawLine(int x1, int y1, int x2, int y2)
{
    int dx, dy, d, incE, incNE, x, y;
    dx = x2 - x1;
    dy = y2 - y1;
    d   = 2*dy - dx;
    incE  = 2*dy;
    incNE = 2*(dy - dx);
    y = y1;
    for (x=x1; x<=x2; x++)

    {
        WritePixel(x, y);
        if (d>0) {
            d = d + incNE;
            y = y + 1;
        } else {
            d = d + incE;
        }
    }
}
```

The teachers will explain the details of the algorithm in the class. The students will try to implement the algorithm in the 1ˢᵗ hour of the lab.

b. **Hour: 2**

**Discussion:**

During this period the teachers will discuss how the students can utilize the above mentioned line drawing algorithm for Zone-0 to draw lines in other zones. First of all, given the co-ordinates of any two points on the line, the students need to convert them in points of Zone-0. For this purpose, the students need to determine in which zone the given co-ordinates are. For example,

```
int FindZone(int x1, int y1, int x2, int y2)
{

        if (abs(dx)>=abs(dy)){
                if(dx>0 && dy>0)
                        Zone=0;
                // write conditions for other zones
        }
        else{
                if(dx>0 && dy>0)
                        Zone=1;
                // write conditions for other zones
        }
        return Zone ;
}
```
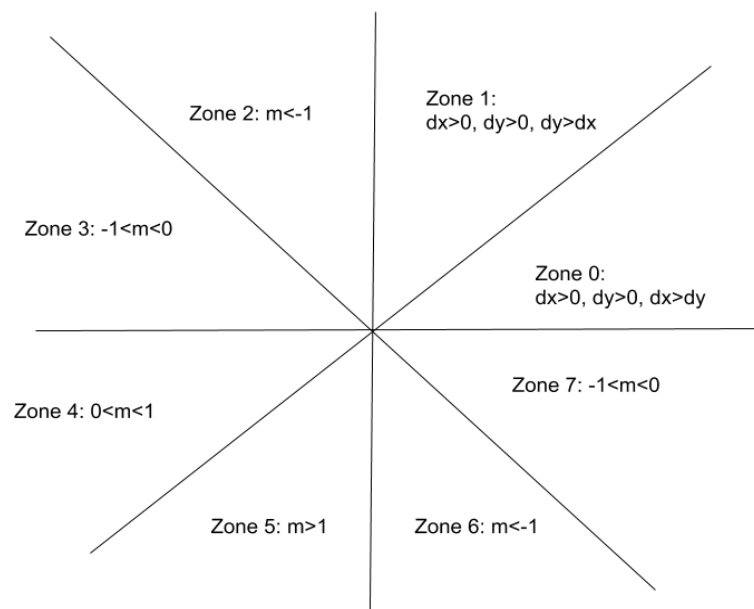
Now, to convert the co-ordinates of any zone to the coordinate of zone 0: For example in Zone 2: (x<0, y>0 & abs(dy)>abs(dx)) while in Zone 0: (x>0, y>0 & abs(dx)>abs(dy)). To convert a point of zone 2 to a point of zone 0, we need to swap its x & y. Since y>0, the x co-ordinate of the point in Zone 0 will be, x of zone 0=y of zone 2. But since x<0 in zone 2, the y co-ordinate of the point in Zone-0 will be, y of zone 0= -x of zone 2 so that it becomes positive. The students need to figure out the required conversion for points in other zones.
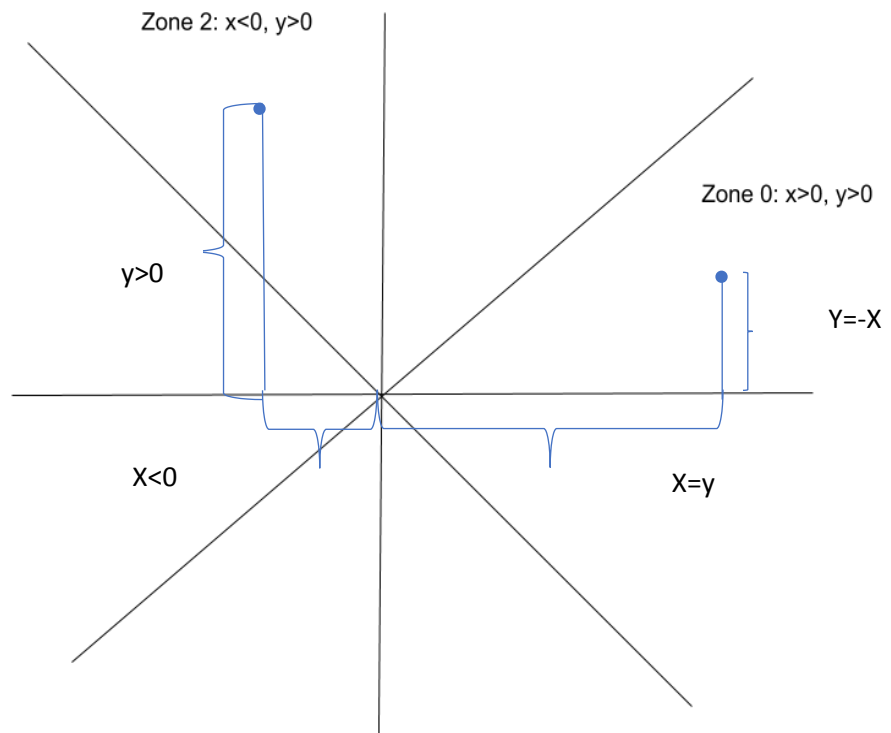


The students will convert the given co-ordinates of the line to co-ordinates of Zone-0. Then they will use Zone-0's midpoint line drawing algorithm to calculate the intermediate points. However, before drawing the pixels the students need to convert them back to their original zone. For example, given any point (x,y) in Zone -0, if we want to convert it back to its original zone, say zone 2, then we need to swap its x & y co-ordinates again. Since x>0 in zone 0 then the point's y

co-ordinate in zone 2 will be y of zone 2=x of zone 0. But since y>0 in zone 0 but x<0 in zone 2, zone 2's x co-ordinate will be x of zone 2= -y of zone 0. So, the basic steps are as follows:

1. Given two points (x1, y1) & (x2,y2)
2. $Zone_1$=findZone(x1,y1) & $Zone_2$=findZone(x2,y2). For simplicity we assume both endpoints of the line are in same zone, i.e., $Zone_1$= $Zone_2$
3. Convert (x1, y1) from $Zone_1$ to a point of Zone-0, say (x1', y1')
4. Convert (x2, y2) from $Zone_2$ to a point of Zone-0, say (x2', y2')
5. Run mid-point line drawing algorithm for zone 0 using (x1', y1') & (x2', y2') as input co-ordinates.
6. Calculate the intermediate points (x, y).
7. Now before drawing the pixels, convert (x,y) to its original zone.

c. **Hour: 3**

**Discussion**

The students will implement the 8-way symmetry for mid-point line drawing algorithm.

**Problem Task:**

i. Implement midpoint line drawing algorithm for zone 0

ii. Implement eight way symmetry for midpoint line drawing algorithm.

**VII.    Home tasks**

Now the students know how to use a single method to draw any line, they can use it to draw the last 2 digits of their student ID. For example, if your student ID is 17123456, then you should draw '56' on your output screen using the Mid-point Line Drawing Algorithm.