Handout: Rasterizing Lines

A. Naive approach: use line equation, ______________________

```
// first octant, 0 < m < 1
m = ___________________
b = ___________________
for( x = __; x ≤ ____; ++x )      ←
  y = ________________
  plot( x, ________________ )     ← choose the pixel ____________________
```

Inefficient due to ___________________________ .

B. Improvement:

```
yi    = _____________________
yi+1  = _________________ + b
      = ___ (__________) + b
      = _____________________
      = _____________________
yi+1  = _____________________     ← next value of y obtained by ___________
```

C. Pseudo-code:

```
// first octant, 0 < m < 1, line from (x1,y1) to (x2,y2)
compute __
compute __
plot( __, __ )
compute __
y = __
while( _________ )
  increment x1 by ___
  increment y by ___
  plot( x1, _______________ )
```

No multiplies ⇒ this is called an

"___________________________________"

or, a "DDA":

"___________________________________"

// DDA, <u>all octants</u> (lab assignment implements!):

compute ___

compute ___

plot(___, ___)

// Shallow slope case (can be either ____ or ____):

if ____ < slope < ____

  compute m

  // Do we increment or decrement x ?

  dx < 0 ⇒ ____________ , otherwise ____________

  // Make slope + or –

  m = ___________

  y = ___________

  while(_____________)

    increment x_1 by _______

    increment y by _______

    plot(x_1, floor(y + 0.5))

else // Steep slope case: _________________

  // same algorithm as above, except

  // switch _____ and _____, _____ and _____

  plot(floor(x + 0.5), y_1)

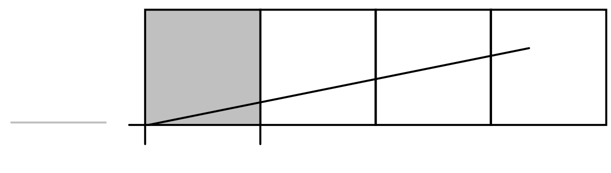
Handout: Rasterizing Lines

Bresenham's Algorithm (1st octant illustrates the general idea but extends to all octants)

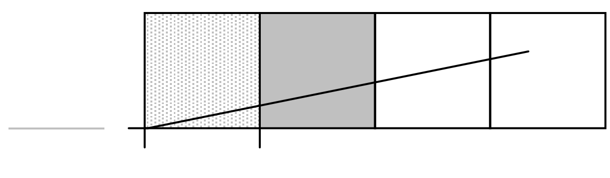
1st octant ⇒ ______________ . Rasterizing a line from __________ to__________ .

Starting at __________ , increment ________ each time.

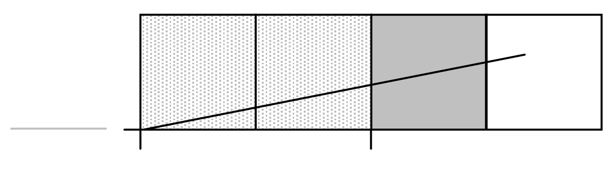
Question: As we move to the right, ________________________________ ?



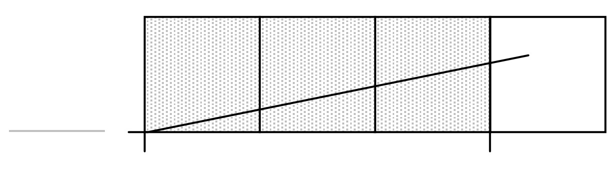
Keep track of ___________________ the distance from ______________________ to ________________________ of the line at __________________________ .



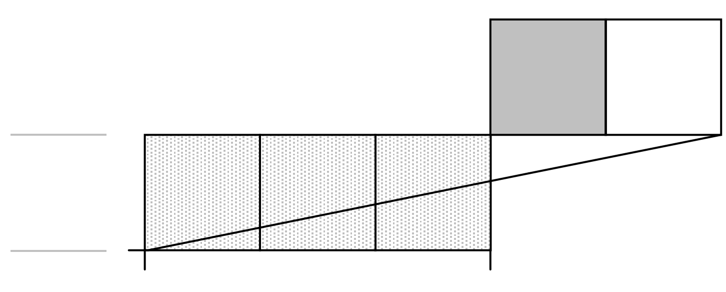
Each time we ______________ , we _______________ by ______________________ :



When the error value _____________ , the line is closer to ______________________ ...



... so we _________________ , and reset the error term, making it relative to the new scanline by _____________________ :



Handout: Rasterizing Lines

Naïve Bresenham(x_1, y_1, x_2, y_2) // First octant only!

```
dx = __________
dy = __________
e  = __________
de = __________                    <-- ___________
y  = __________
for( x from _____________ )
  plot( __,__ )
  e = _________
  if( _________ )                  <-- ___________
    _______
    e = _____________              <-- ___________
```

Improved Bresenham(x_1, y_1, x_2, y_2) // First octant only!

```
dx = |x2 - x1|
dy = |y2 - y1|
e = 0
__________                         <-- ___________
y = y1
for( x from x1 to x2 )
  plot(x,y)
  e = e + de
  if( ______________ )             <-- ___________
    ++y
    e = ____________               <-- ___________
```

Integer Bresenham(x_1, y_1, x_2, y_2) // First octant only!

```
dx = |x2 - x1|
dy = |y2 - y1|
e = 0
de = dy
y = y1
for( x from x1 to x2 )
  plot(x,y)
  e = e + de
  if( _________ )                       <-- ___________
    ++y
    e = e - dx
```

Handout: Rasterizing Lines

Swapping ____________ (reflection about `y = x`) when ____________

converts ____________ to the steep slope case above.

```
Bresenham(x1, y1, x2, y2) // First and second octants only
steep = ____________ > ____________(boolean flag)

if( steep )

   ________________

   ________________

dx = ______________

dy = ______________

e  = ______________

de = ______________

y  = ______________

for( x from x1 to x2 )

   if( steep ) ________________ else ________________

   e = ______________

   if( ______________ )

      ++y

      e = ____________
```

Remaining octants: ________________ and/or ________________, as appropriate.

Bresenham(x_1, y_1, x_2, y_2) // All octants (lab implements):

```
steep = |y2 - y1| > |x2 - x1|
if( steep )
  swap(x1, y1)
  swap(x2, y2)
if( ____________ )
   ____________
   ____________
dx = ________________
dy = |y2 - y1|
e  = 0
de = dy
y  = y1
if( ____________ ) ____________ else ________________
for( x from x1 to x2 )
  if( steep ) plot(y,x) else plot(x,y)
  e = e + de
  if( 2e ≥ dx )
    y += ____________
```

```
e = e - dx
```