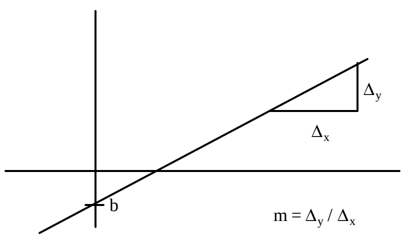
Computer Graphics

Z Sweedyk
Lecture 2
9/4/00

Line Segments

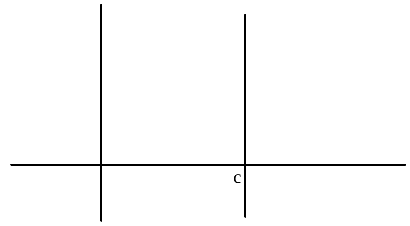
- **Scan converting line segments**
  - Naïve algorithm
  - Midpoint algorithm
  - Bresenham's algorithm
- Clipping line segments (intro)
  - Scissoring
  - Analytical clipping

Line: Set of points (x,y) satisfying y = mx + b



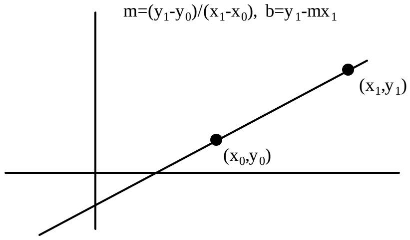
Line: Set of points (x,y) satisfying y = mx + b

Or x=c in the case of a vertical line



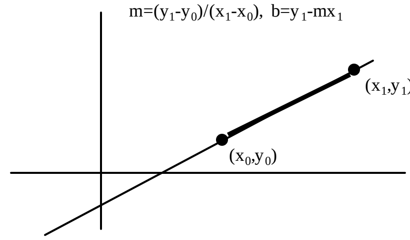
Line through (x_0,y_0) and (x_1,y_1)

$m=(y_1-y_0)/(x_1-x_0)$, $b=y_1-mx_1$



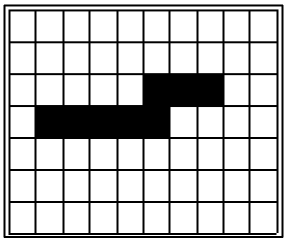
Line segment with endpoints (x_0,y_0) and (x_1,y_1)

$m=(y_1-y_0)/(x_1-x_0)$, $b=y_1-mx_1$

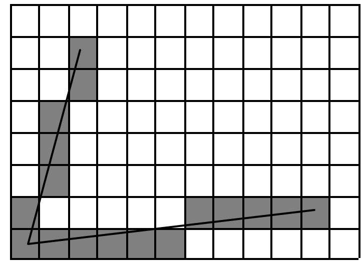


Scan Converting Line Segments

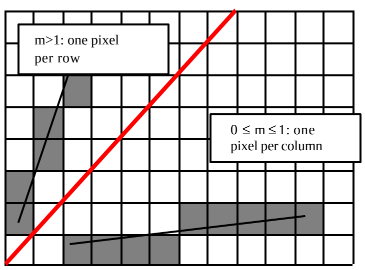
Which pixels should be on?



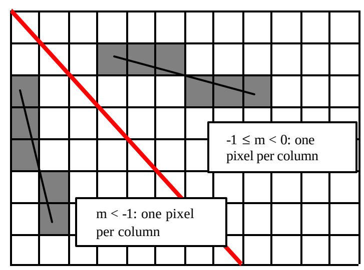
1-pixel wide lines



1-pixel wide lines: m≥0



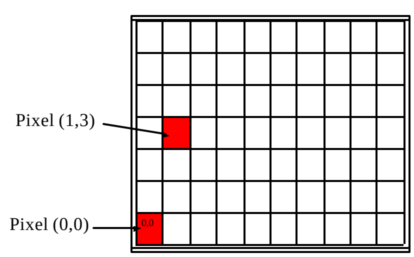
1-pixel wide lines: m<0



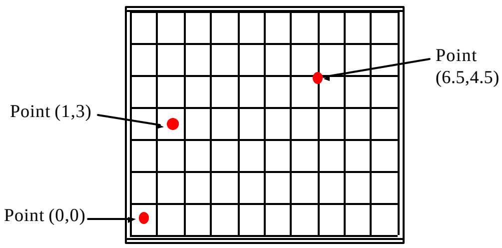
Scan Conversion

- Input: Endpoint Pixels
- Output: Pixels to turn on for a 1-pixel wide line segment

Points and Pixels

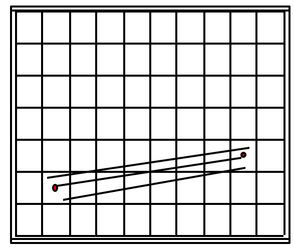


Points and Pixels



Equivalence Classes of Ideal Lines

If that's not good enough you need better resolution!!!!!



Line Segments

- **Scan converting line segments**
  - Naïve algorithm
  - Midpoint algorithm
  - Bresenham's algorithm
- Clipping line segments
  - Scissoring
  - Analytical clipping
- Antialiasing

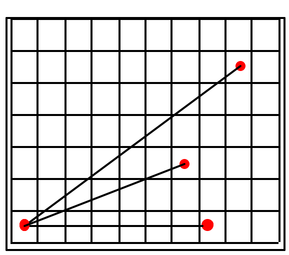
Claim

All we have to do is solve the scan conversion problem for the special case where

1. $0 \leq m \leq 1$
2. $x_0 = y_0 = 0$

We'll prove this later … first we'll devise an algorithm for the special case.

Ideal lines we'll scan convert

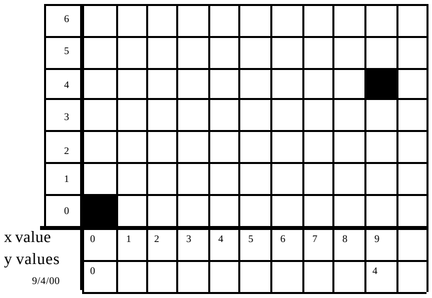


Line Segments

- **Scan converting line segments**
  - **Naïve algorithm (special case)**
  - Midpoint algorithm
  - Bresenham's algorithm
- Clipping line segments
  - Scissoring
  - Analytical clipping
- Antialiasing

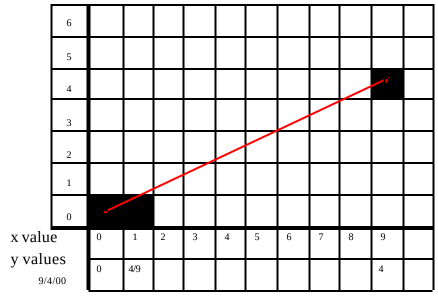
Endpoints: (0,0) and (9,4)

y=mx, m = 4/9



Endpoints: (0,0) and (9,4)

y=mx, m = 4/9



Special Case Line Algorithm 1

$x_0=y_0=0$, $0 \le m \le 1$

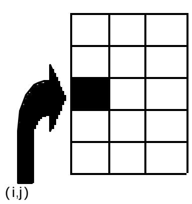
```
SpecialCaseLine1(int x1 , int y1 )
   int current x=0
   float current_y=0,
   float m = (float) y1 / (float) x1
   while (current_x ≤ x1 )
        DrawPixel(current_x,round(current_y))
        current_x += 1, current_y += m
```

Floating Point

Line Segments

- **Scan converting line segments**
  - Naïve algorithm
  - **Midpoint algorithm**
  - Bresenham's algorithm
- Clipping line segments
  - Scissoring
  - Analytical clipping
- Antialiasing

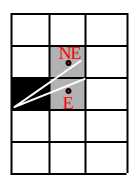
Let's try to do better!



•Suppose we've just drawn the (i,j) pixel. How do we choose the next pixel?

How do we choose next pixel?

y=mx, $0 \le m \le 1$

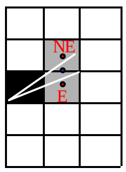


•The options are E and NE

•Choose pixel whose center is closest to the ideal line

How do we choose next pixel?

$y=mx, 0 \le m \le 1$



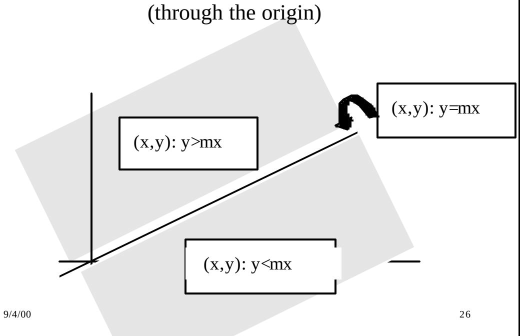
Define (u,v) to be the midpoint of the line segment between the centers of NE and E

If (u,v) lies below the ideal line: go NE

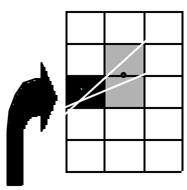
Else: go E

Line in the plane

(through the origin)



The Test



(i,j)

$u=i+1$

$v=j+1/2$

Ideal line: $y=mx$ ½

If $v < mu$ then go NE

Else go E

Special Case Line Algorithm 2

($x_0=y_0=0$ and $0 \le m \le 1$)

```
SpecialCaseLine2(x1,y1)

 m = y1/x1
 i=0, j=0
 while i<x1
      write-pixel(i,j)
      if j+1/2 ½< m·(i+1)
              i+=1, j+=1     // go NE
      else i+=1             // go E
```

Floating Point

Modified Test

Is $j + 1/2½ < m(i+1)$?

⇕

Is $j + 1/2½ < (y_1 / x_1) \cdot (i+1)$?

⇕

Is $x_1(2j + 1) < 2y_1(i+1)$?

Important: x_1 and y_1 are integers!

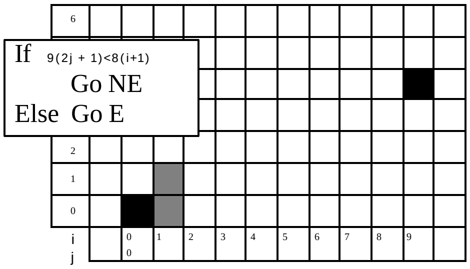
Special Case Line Algorithm 3

($x_0=y_0=0$ and $0 \le m \le 1$)

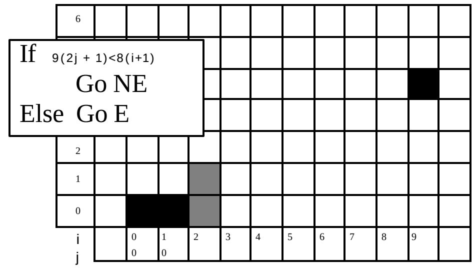
```
SpecialCaseLine3 (x1,y1)

 i=0, j=0
 while current_i<x1
      write-pixel(I,j)
      if x1(2j+1)< 2y1(i+1)
              i+=1, j+=1    // go NE
      else i+=1             // go E
```

Endpoints (0,0) & (9,4):



Endpoints (0,0) & (9,4):



Special Case Line Algorithm 3

(x_0=y_0=0 and $0 \le m \le 1$)

```
SpecialCaseLine3 (x1,y1)

  i=0, j=0
  while current_i<x1
     write-pixel(I,j)
     if x1(2j+1)< 2y1(i+1)
          i+=1, j+=1    // go NE
     else i+=1          // go E
```

Multiplication

Line Segments

- **Scan converting line segments**
  - Naïve algorithm
  - Midpoint algorithm
  - **Bresenham's algorithm**
- Clipping line segments
  - Scissoring
  - Analytical clipping
- Antialiasing

Sleight of hand

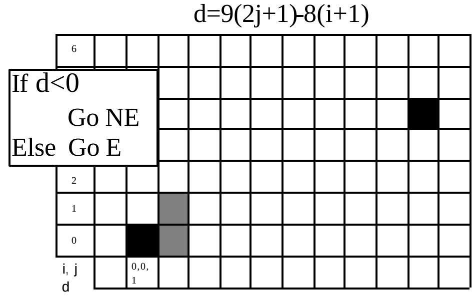
```
if x1(2j+1)< 2y1(i+1)
   go NE
else go E
```

$\Updownarrow$

```
d = x1(2j+1)-2y1(i+1)   if d<0
   go NE
else go E
```

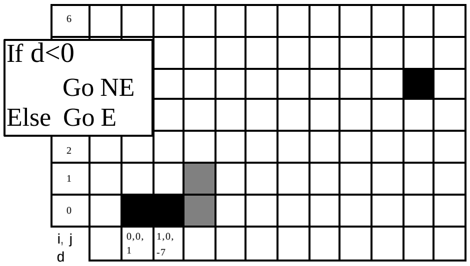

Endpoints (0,0) & (9,4):

d=9(2j+1)-8(i+1)



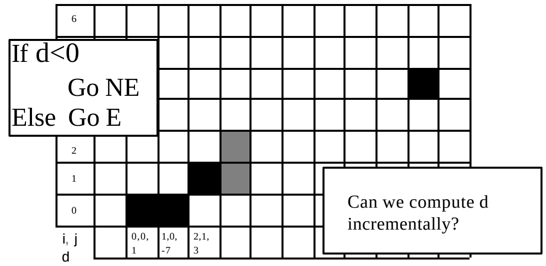
Endpoints (0,0) & (9,4):

d=9(2j+1)-8(i+1)



Endpoints (0,0) & (9,4):

d=9(2j+1)-8(i+1)



Special Case Bresenham's

(x_0=y_0=0 and 0≤m≤1)

```
SpecialCaseBresenham's(x1,y1)

  i=0, j=0
  d = x1-2y1
  while i<x1
     write-pixel(I,j)
     if d<0
          i+=1, j+=1, d+=2(x1-y1)
     else
          i+=1, d-=2y1
```

Do as addition

Yay!

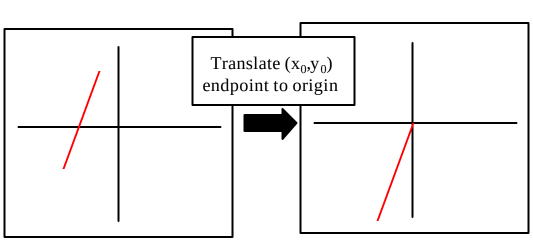
Claim

All we have to do is solve the problem for the special case where

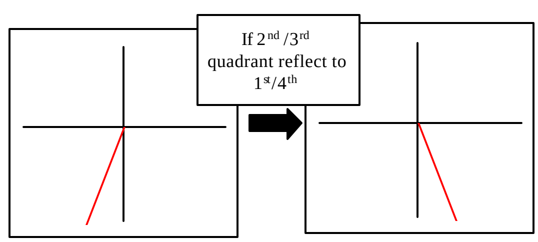
1. 0≤m<1
2. x_0=y_0=0

Now we'll prove this claim

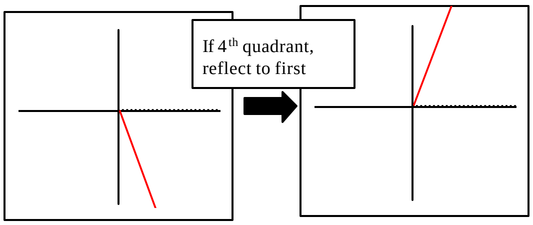
To solve general case:



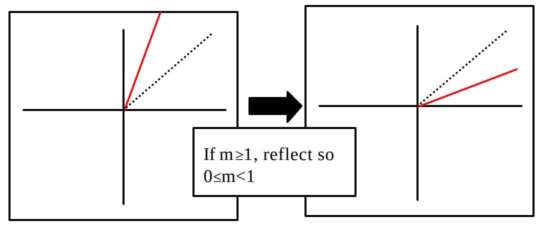
To solve general case:



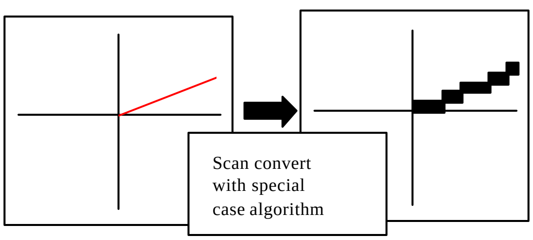
To solve general case:



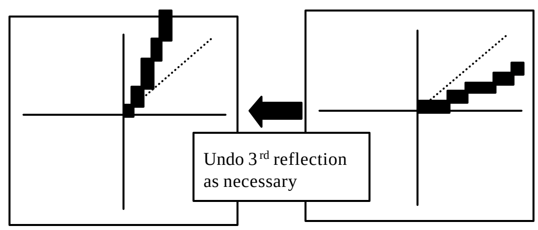
To solve general case:



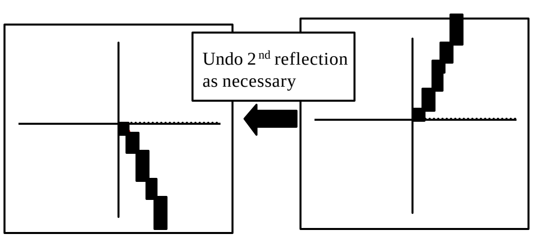
To solve general case:



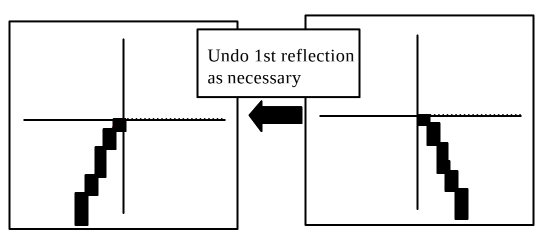
To solve general case:



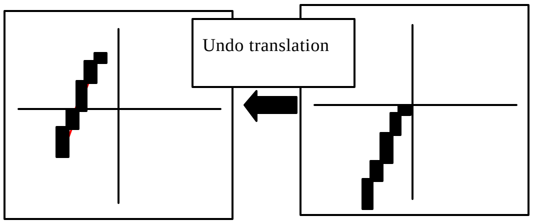
To solve general case:



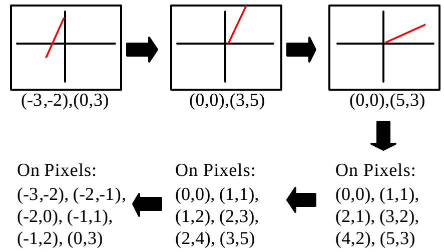
To solve general case:



To solve general case:



Example



Implementation

- Think it through
- Get special case working first
- Endpoint order MATTERS!

Endpoint Order

- Why it matters

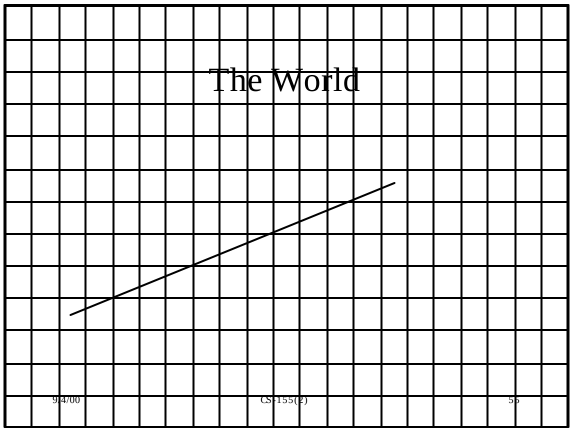
Implementation??

- Modified special case Bresenham's for different slopes/endpoint orders
- Reduction (in this case Bresenham's should return an array of y-values)

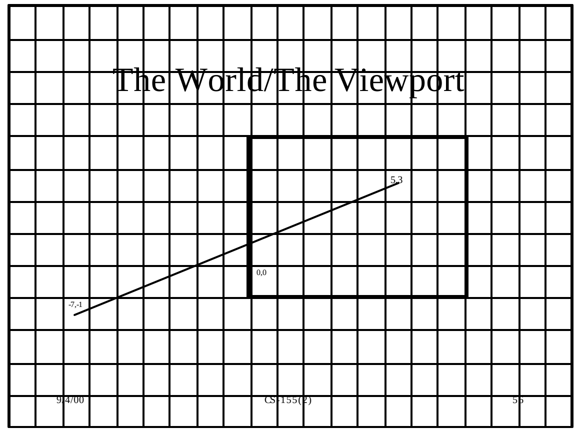
Line Segments

- Scan converting line segments
  - Naïve algorithm
  - Midpoint algorithm
  - Bresenham's algorithm
- **Clipping line segments (intro)**
  - Scissoring
  - Analytical clipping

The World



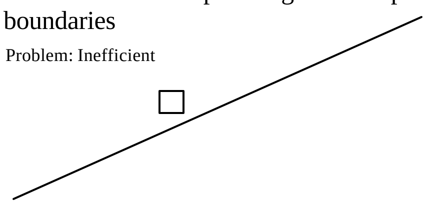
The World/The Viewport



Answer 1: Scissoring

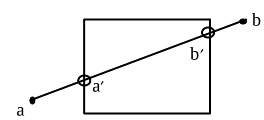
- Compute pixels in world grid
- Then check chosen pixels against viewport boundaries

Problem: Inefficient



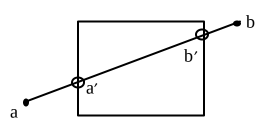
Answer 2: Analytical Clipping

- Compute intersection points
- Scan convert line segment between intersection points



But … a’ and b’ are not likely to have integer coordinates

1. We could go back to using floating point operations.
2. We could round the endpoint coordinates and scan convert.
3. We could do it the right way.

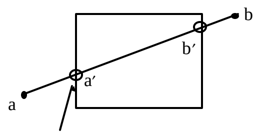


The right way

```
SpecialCaseBresenham's(x1,y1,xfirst,xlast)
  m=y1/x1
  i=xfirst, j=round(m*xfirst)
  d = x1(2*j+1)-2 y1(i+1)
  while i<xlast
     write-pixel(i,j)
     if d<0
          i+=1, j+=1, d+=2(x1-y1)
     else
          i+=1, d-=2y1
```

Answer 2: Analytical Clipping

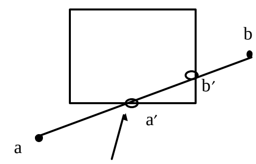
- Compute intersection points
- Scan convert line segment between intersection points



Intersection of lines: y=mx and x=c

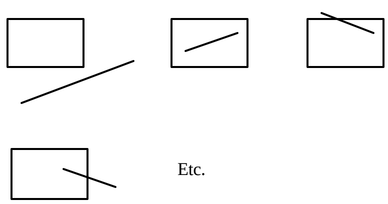
Answer 2: Analytical Clipping

- Compute intersection points
- Scan convert line segment between intersection points



Or Intersection of lines:
y=mx and y=d

Or



Next time

- Clipping line segments concluded.
- Polygons