## Lecture 2: 14 September 1999

**Notes from last time:**
Please sign the circulating sign-up sheet
**only if you haven't already done so**
Final projects:
Examples were meant to be inspiring !
Don't confuse animations with video
Substantive design, implementation experience

**Administrative stuff:**
Linear Algebra review session
Tomorrow (W), 7-9pm in 34-101 (Damian)
Textbook (Hearn & Baker)
Book is not available at Coop (my mistake, sorry)
Available at Quantum Books now ($57.60, no tax)
Barnes & Noble (bn.com) quotes $63.75 next-day
Copies on reserve at LCS reading room
Assignment 1 (web signup, etc.) due this Friday
Make sure to make your page readable via NFS:

```
% chmod a+r homepage.html
```

Asst. 2 (2D Segment processing) due next Friday

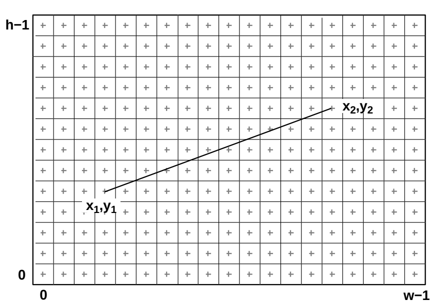


Today: 2D Segment Processing
Rasterization, Clipping
H&B §3.1-3.6; §6.5-6.7



## Framebuffer Model

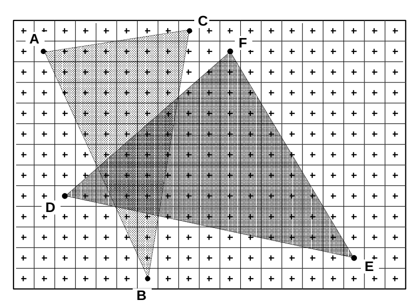
Raster Display: 2D array of picture elements (pixels)
Pixels individually set/cleared (greyscale, color)
Window coordinates: pixels centered at integers






## 2D Scan Conversion

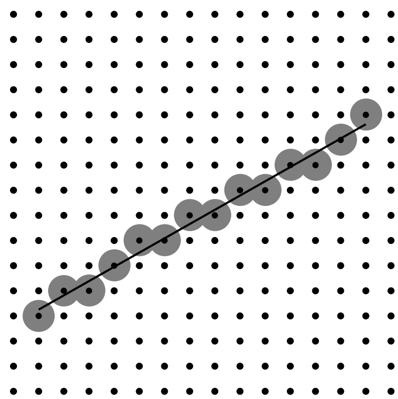
Geometric "primitive:" specified or rendered object
2D: point, line, polygon, circle...
(3D: point, line, polyhedron, sphere...)
Challenge: primitives are continuous; screen is discrete
Solution: compute & display discrete approximation




Scan Conversion: algorithms for efficient generation
of the samples comprising this approximation

## Scan Converting 2D Line Segments

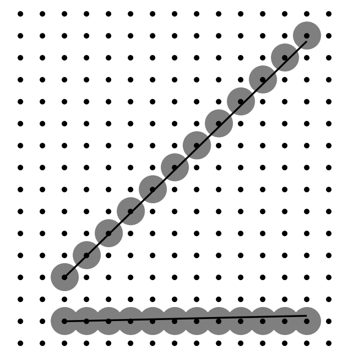
Given:
Segment endpoints (integers $x_1, y_1, x_2, y_2$)
Framebuffer access via *setPixel*$(x, y)$
Identify:
Set of $x, y$ for which to call *setPixel*$(x, y)$
I.e., the set of pixels to "light up" for segment





## Algorithm Design Choices

For $m \equiv dy/dx$, assume: $0 < m < 1$ (Why?)
Exactly one pixel per column (Why?)
fewer $\Rightarrow$ disconnected
more $\Rightarrow$ "too thick"
"connectedness" with just 1 pixel per column
Note: brightness can vary with slope

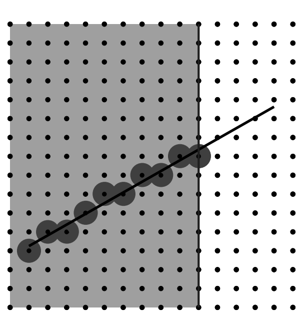


How could we compensate for this?
Answer: antialiasing



## Naive algorithm

First attempt: simply compute $y$ as a function of $x$
(Conceptually: move vertical scan line from $x_1$ to $x_2$)

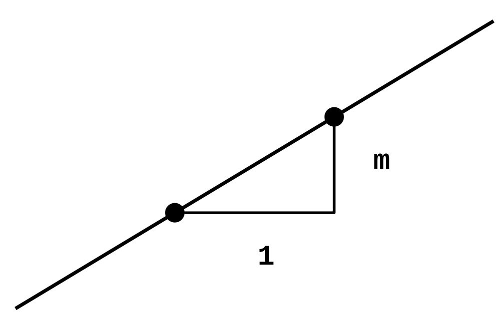


- Express $y$ as function of $x$: How?
- Round $y$ (Why?)
- Call *setPixel* $(x, \text{rnd}(y(x)))$



## Efficiency

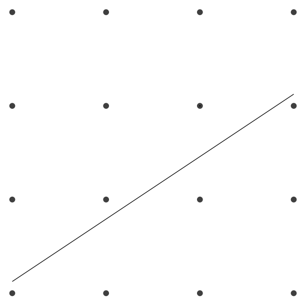
Computing $y$ values is expensive (how?)
Observe: $y$ `+=` $m$ at each $x$ step ($m = dy/dx$)
First example of **spatial coherence**




Incremental algorithm:
Start at $(x_1, y_1)$
Thereafter, increment $y$ value by slope $m$
Note: $x$ integer, but $y$ floating point

## Bresenhams "DDA"

DDA = Digital Differential Analyzer
Select pixel **vertically** closest to segment

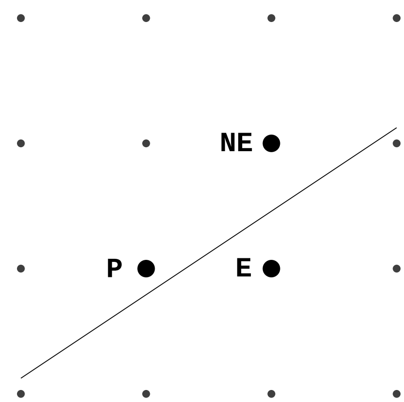


Justification: intuitive, efficient
  Also: pixel center always within 0.5 vertically
  Is selection criterion well-defined ?
What other rules could we have used?



## Bresenham Segment Algorithm

Another scan line algorithm
Same output as naive & incremental algorithms
Observation: after pixel $P$ at $(x_P, y_P)$
  next pixel must be either $E$ or $NE$

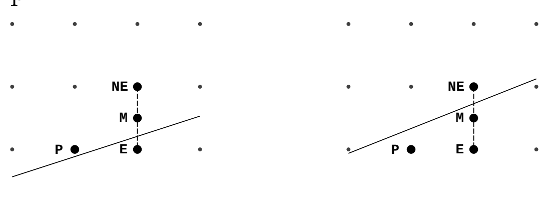



Why?

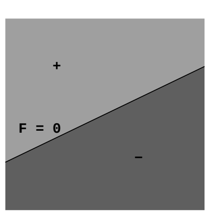


## Bresenham Step

Which pixel to choose: $E$ or $NE$?




  Choose $E$ if segment passes below or through $M$
  Choose $NE$ if segment passes above $M$
Use **implicit equation** for underlying line **L**:
$F(x, y) = 0$, where $F(x, y) \equiv y - mx - b$ (Why?)
$F$ positive above **L**, zero on **L**, negative below **L**

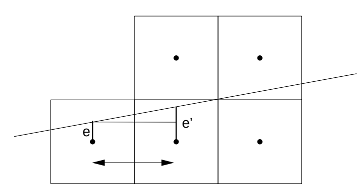



What is the meaning of the *value* of $F(x, y)$ ?
Define an error term $e$ as $e \equiv -F(x, y)$
  choose $NE$ if $e$ $\boxed{\phantom{xxxxxx}}$ ; otherwise choose $E$.



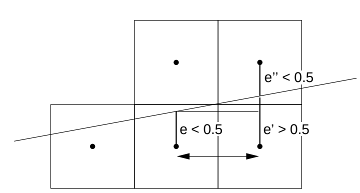
## Using the Error Term

Compute $e'$ under (unconditional) $x$ increment:




$$
\begin{aligned}
e &= -F(x_P, y_P) \\
&= -y_P + m x_P + b \\
e' &= -F(x_P + 1, y_P) \\
&= -y_P + m(x_P + 1) + b \\
&= -y_P + m x_P + b + m \\
&= \boxed{\phantom{xxxx}}
\end{aligned}
$$

Under what condition should we choose $E$?
Under what condition should we choose $NE$?




In this case, how should $e'$ be computed?
So: initialize $x$, $y$, $e$; loop over $x_1..x_2$, plot, update

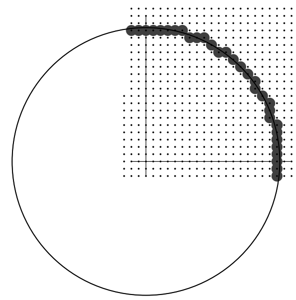
## Bresenham Implementation

We've sketched case in which $x_1 < x_2$, $m <= 1$
This is Assignment 2, part A (demo, Damian).
Required:
  Implement using integer arithmetic only
Optional:
  Handle all eight octants
  Ditto, but do so using one `for` loop
  Minimize the total number of Java statements
  Generalize to circles, conics, etc.

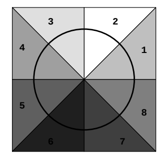


## Circle Scan Conversion

Circle of radius $R$ centered at origin



Need generate pixels for 2nd octant only

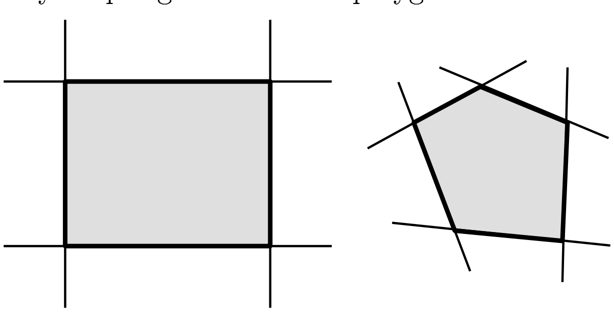



Why use the second octant?
Slope progresses from 0 to -1
  Analog of Bresenham Segment Algorithm



## 2D Clipping

So far, assumed line segment lies within viewport
But, some vertices might be specified outside
Want geometry confined to *clip region*
  Today: clip region is convex polygon



  Most frequent case: clip rectangle

$$x_{min} \le x \le x_{max}$$

$$y_{min} \le y \le y_{max}$$

  Clipping is the process of pruning geometric
    primitives to the clip region

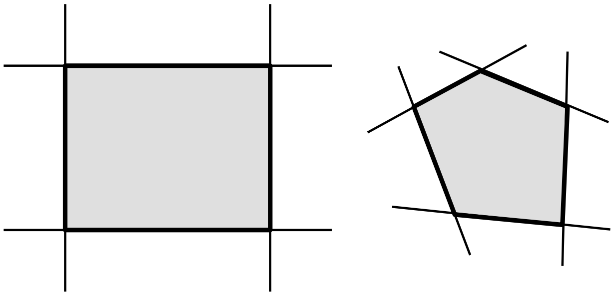


## Clipping Approaches

1) Scan convert elsewhere; copy bitmap to viewport
2) Scissor: clip on the fly during scan conversion
3) Clip analytically: revise input geometry
Today: analytical clipping of
  Points
  Segments
Later:
  Wireframe polygons
  Filled polygons

## Point Clipping

Idea: retain point iff it is inside clip region



Clip region described by 4 inequalities:

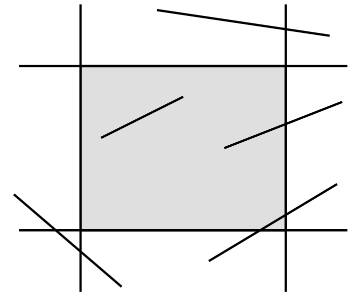
$$x \geq x_{min}$$
$$x \leq x_{max}$$
$$y \geq y_{min}$$
$$y \leq y_{max}$$

Generalization to arbitrary convex regions?



## Segment Clipping: Cases

Output must be a single segment, or vacuous (why?)



Both endpoints in clip region? *Trivial accept.*
One endpoint in clip region, one endpoint outside?
  Must exist intersection point $p$ with edge of
    clip region; replace outside endpoint with $p$
Otherwise, both endpoints outside (two cases):
  No intersection with clip region
    Absorb segment (report "external")
  Intersections with two edges of clip region
    Output segment between intersection points
    Handle degeneracies (e.g., corner crossing)



## Brute-Force Clipping

Classify endpoints to identify relevant case
Compute intersection with each clip region edge
  (Identify 0, 1, or 2 clipped points)
How to compute intersection points?
Express segment parametrically as function of $t$:

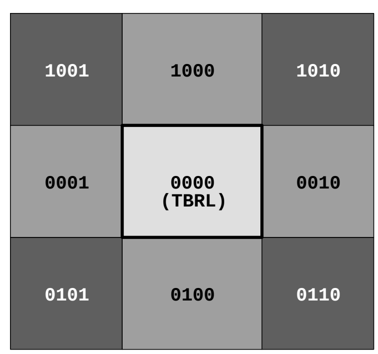
$$x = x_0 + t(x_1 - x_0)$$
$$y = y_0 + t(y_1 - y_0)$$

  (What values does $t$ take on?)
Plug in to expressions for clip boundaries:
Example: $x = x_{min}$ yields $t = \boxed{\phantom{xxxx}}$
  Check $t$ to make sure it is inside $[0, 1]$
  (Careful: what if segment parallel to boundary?)
Brute force algorithm is expensive



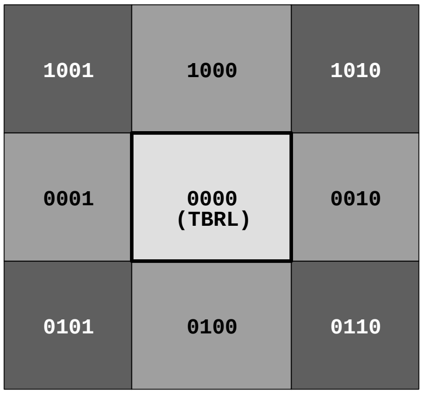
## Cohen-Sutherland Algorithm

Reduces number of intersection computations
Works for any convex polygonal clip region
Observation: can often **trivially reject** segment (how?)




Strategy: classify each endpoint with respect
  to T,B,R,L half-planes (1 bit per plane):
    $[y \geq y_{max}]$ (outside Top)
    $[y \leq y_{min}]$ (outside Bottom)
    $[x \geq x_{max}]$ (outside Right)
    $[x \leq x_{min}]$ (outside left)
  Easy: each bit is sign bit of difference

## Cohen-Sutherland Algorithm

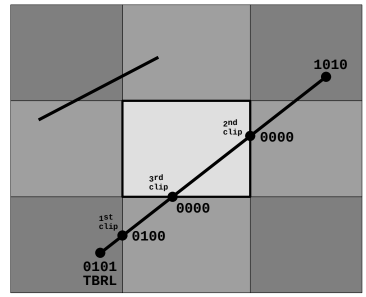



Concatenate TBRL bits into 4-bit “outcodes”
- Trivial accept iff both outcodes are zero
- Trivial reject iff some bit 1 for both endpoints
- Otherwise, what do we know about segment?



## Cohen-Sutherland Algorithm

If neither trivial reject nor accept, then:
- There is at least one 1 bit
- Each 1 has corresponding 0 at other endpoint
  - Thus each 1 implies crossed boundary!
- Thus, if half-plane $H$’s bit = 1 at endpoint $A$
  - $A$ outside $H$, other endpoint inside $H$
  - Clip endpoint $A$ to halfplane $H$, replace $A$




Idea: Clip segment in order Left, Right, Bottom, Top
- After each clip, recompute appropriate outcode
- Continue clipping until trivial accept/reject
  - (Must get one or the other, eventually)



## Cohen-Sutherland Implementation

We’ve sketched `OutCode` and `ClipSegment`
This is Assignment 2, part B (demo, Damian).
Required:
- Implement using floating point
- Report cliptype as internal, clipped, external

Optional:
- Minimize the total number of Java statements
- Generalize to convex/concave polygons
- Generalize to convex/concave clip regions !