CS100433
Part III Rendering - Rasterization

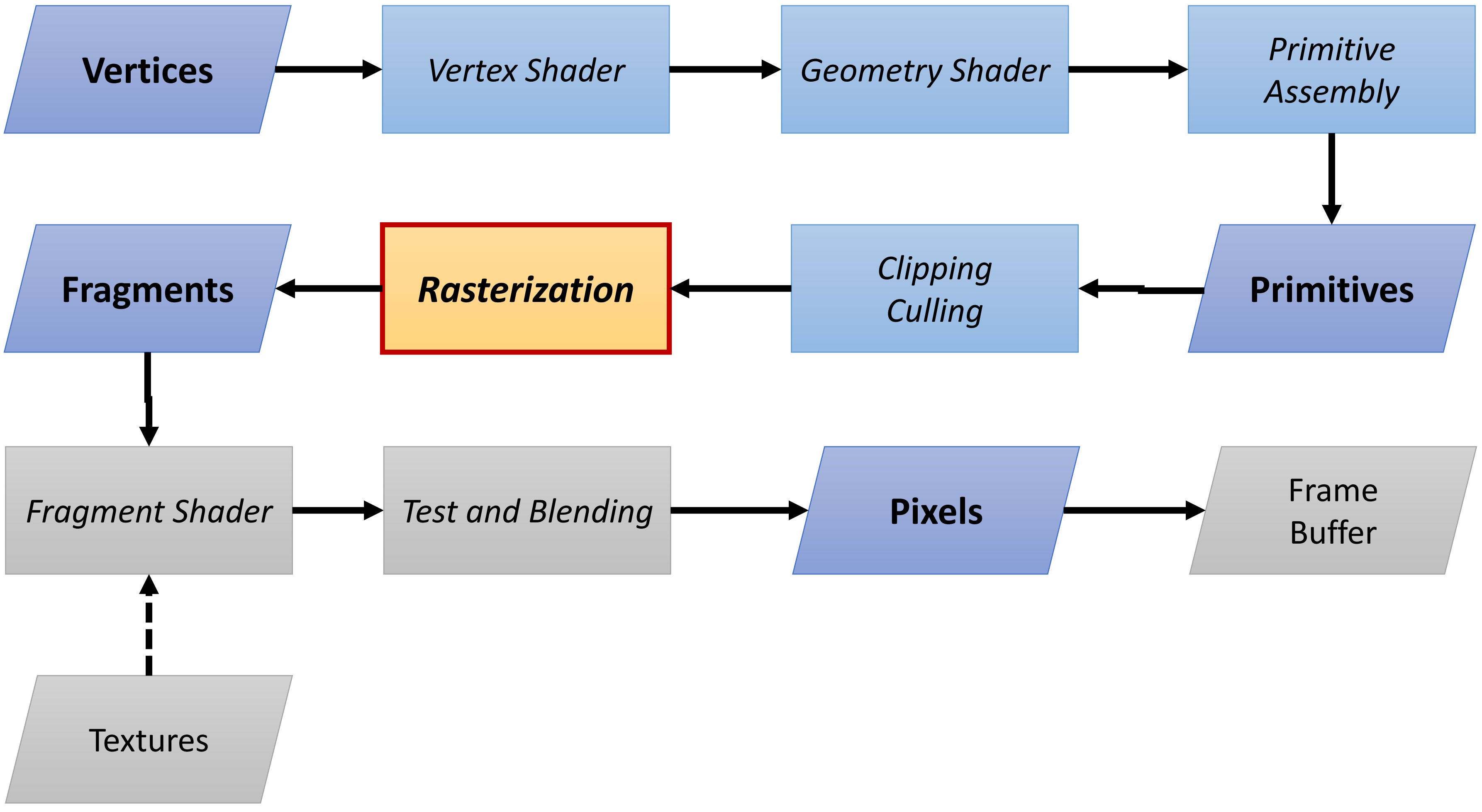
Junqiao Zhao 赵君峤

Department of Computer Science and Technology

College of Electronics and Information Engineering

Tongji University

The graphics pipeline



Vector Graphics

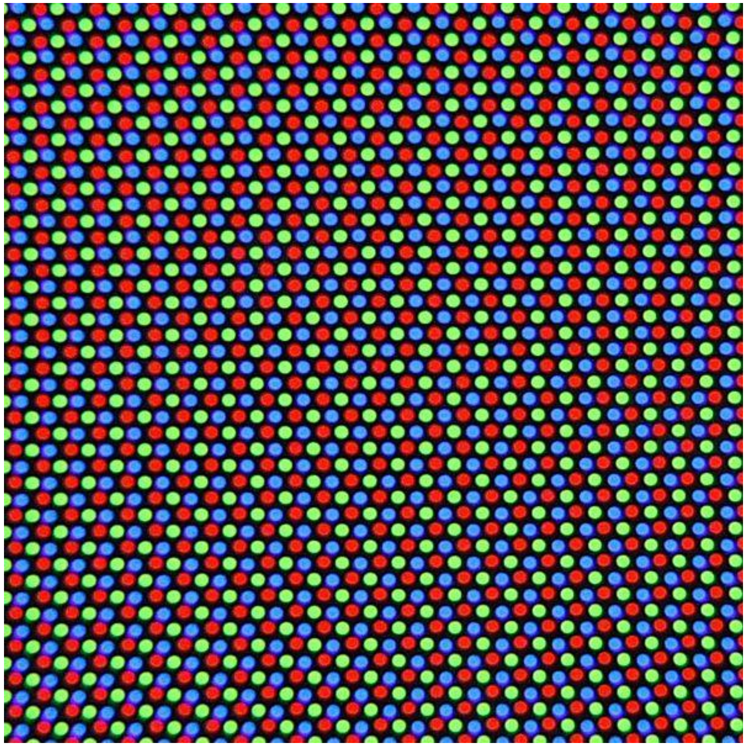
- Algebraic equations describe shapes.
- Can render type and large areas of color with relatively small file sizes
- Can be reduced/enlarged with no loss of quality
- Cannot show continuous tone images (photographs, color blends)
- Examples:
  - Plotters, Oscilloscopes,
  - Illustrator, Flash, PDF

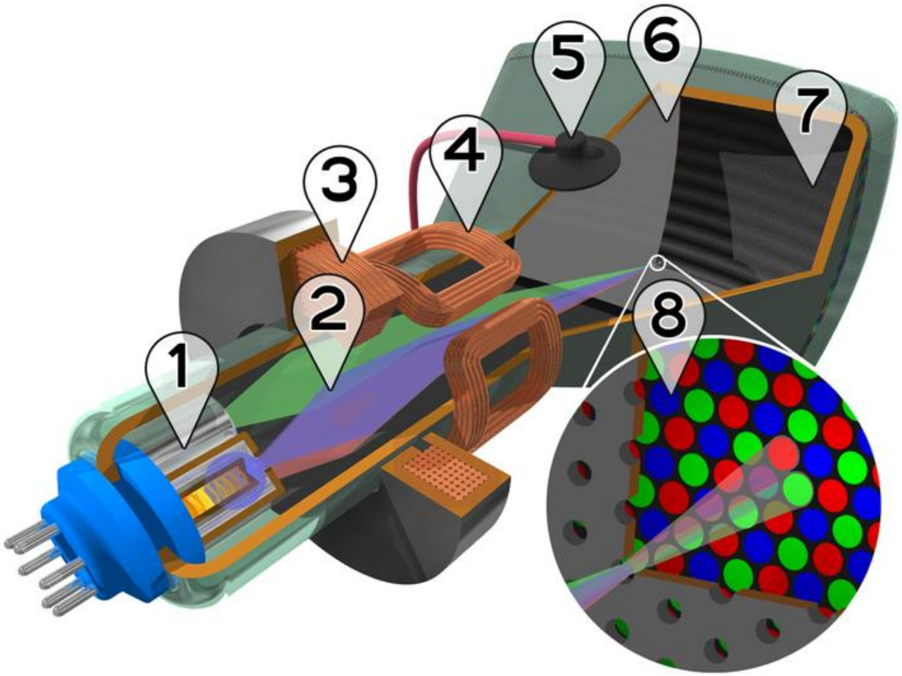
Raster Graphics

- Pixel-based / Bit-mapped graphics
- Grid of pixels
- Size of grid = resolution of image
- Poor scalability : zoom in - loss of quality (jagged look)
- Best for large photographic images
- Modification at pixel level - texture mapping, blending, alpha channels, antialiasing, etc.
- Examples :
  - CRT, LCD, Dot-matrix printers
  - Adobe Photoshop, BMP

The Display Hardware

- Cathode Ray Tube (CRT)

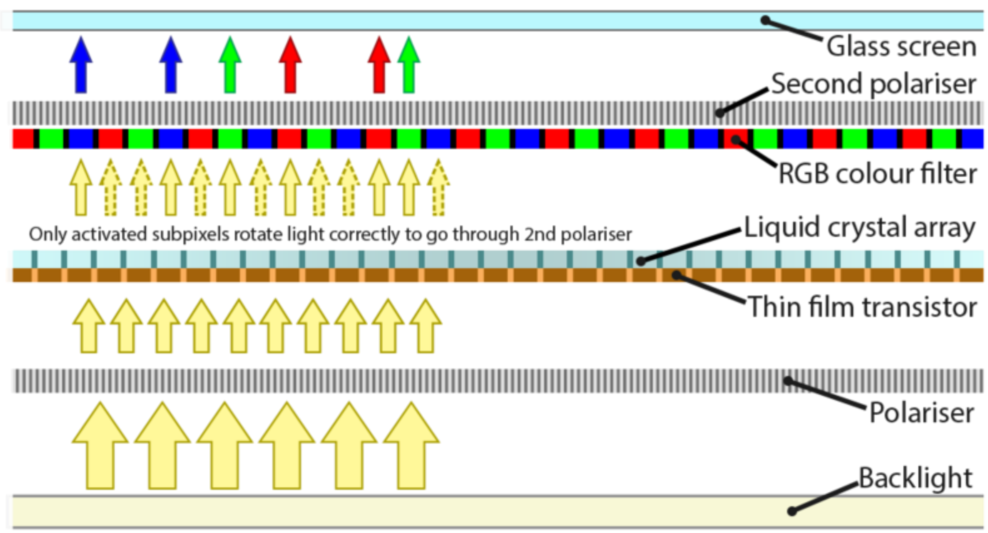


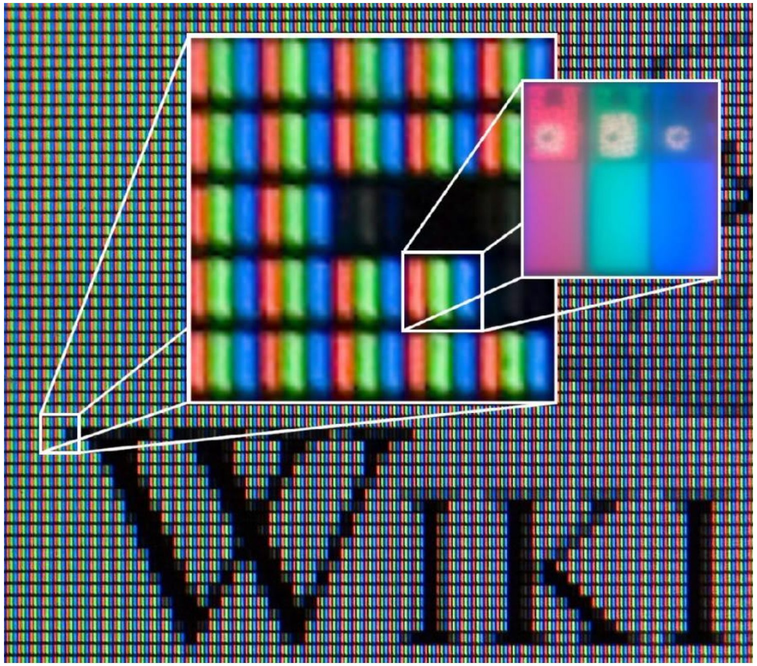


(Wikipedia)

The Display Hardware

- Liquid-crystal display (LCD)





(Website)

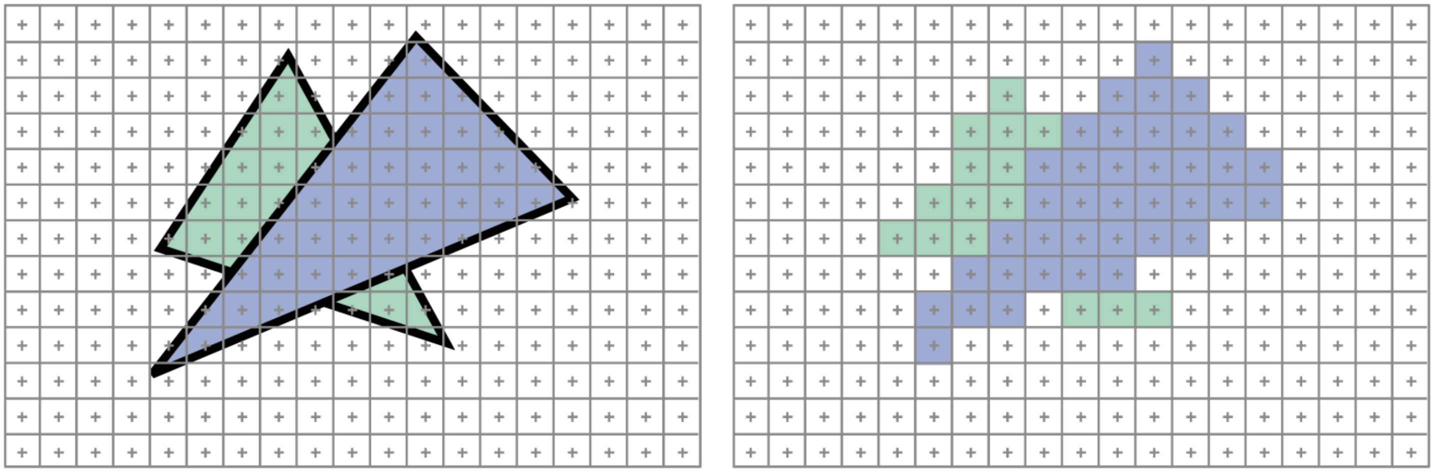
Frame Buffer Model

- Raster Display: 2D array of picture elements (pixels)
- Fragments have a location (pixel location) and other attributes such color and texture coordinates that are determined by interpolating values at vertices
- Pixel colors determined later using color, texture, and other vertex properties

(Durand and Cutler MIT)

Rasterization

- Geometric primitives (point, line, polygon, circle, polyhedron, sphere...)
- Primitives are continuous; screen is discrete
- Scan Conversion: algorithms for *efficient* generation of the samples comprising this approximation



(Durand and Cutler MIT)

Rasterization

Rasterization

= Scan Conversion

= Converting a continuous object such as a line or a circle into discrete pixels.

Rasterization

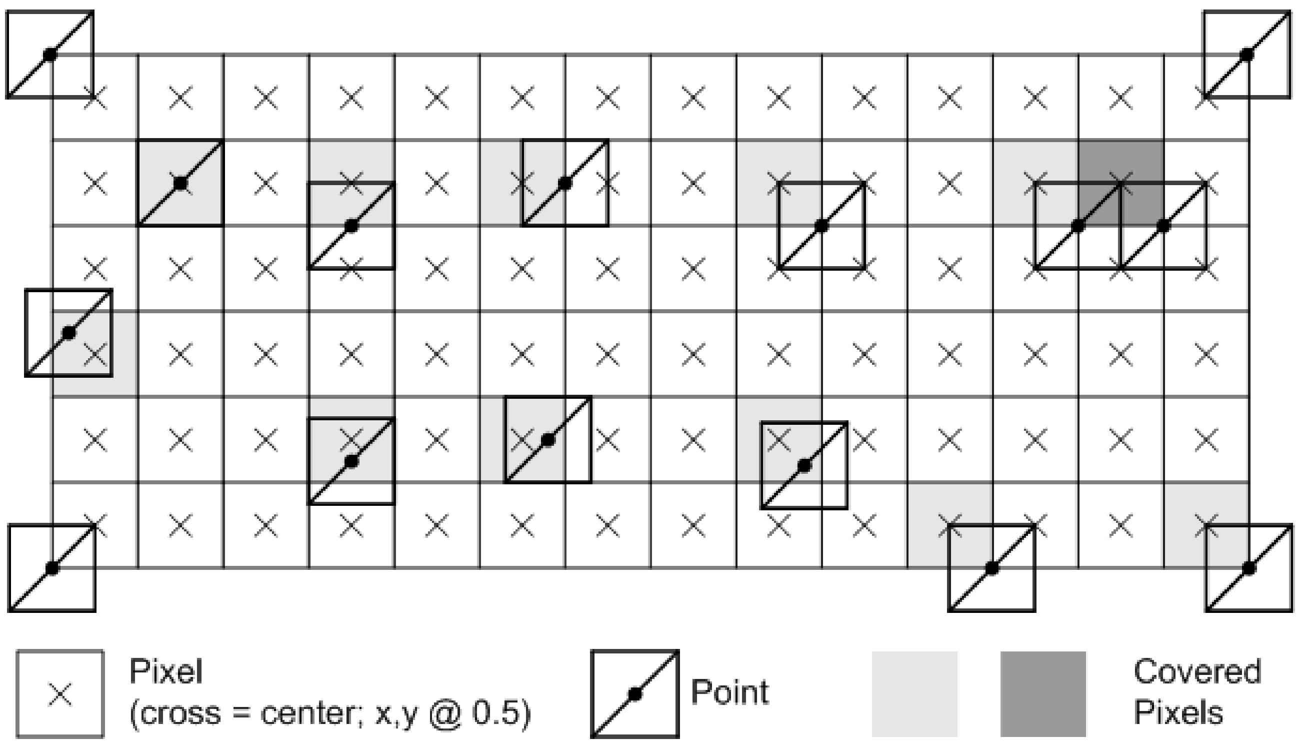
- First job: enumerate the pixels covered by a primitive
  - simple, aliased definition: pixels whose centers fall inside
- Second job: interpolate values across the primitive
  - e.g. colors computed at vertices
  - e.g. normals at vertices

Primitives to be Rasterized

- **Points**
- **Line segments**
  - and chains of connected line segments
- **Triangles**
- And that's all!
  - Curves? Approximate them with chains of line segments
  - Polygons? Break them up into triangles
  - Curved regions? Approximate them with triangles
- Trend has been toward minimal primitives
  - simple, uniform, repetitive: good for parallelism

Rasterize Points

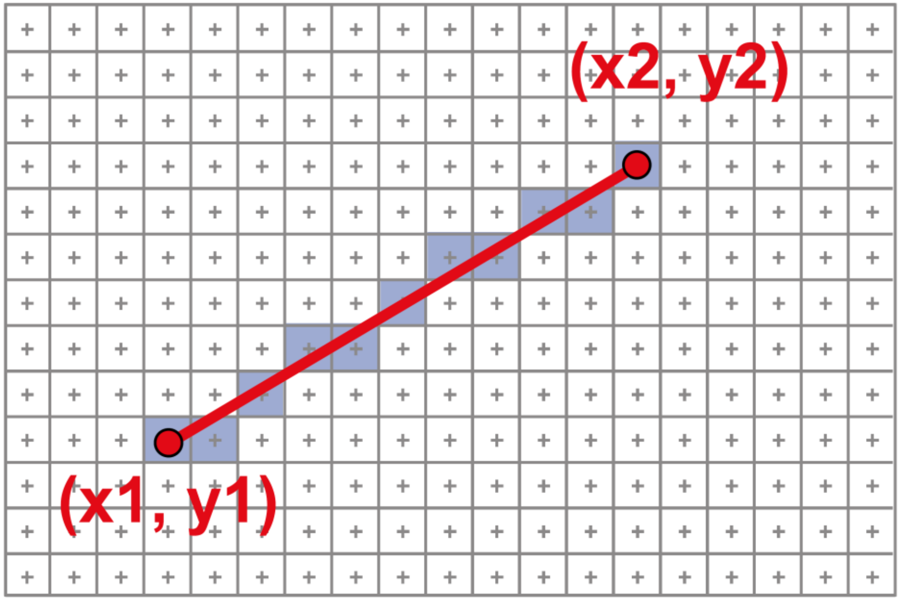
- How?
- Rules



(Microsoft Dx11)

Rasterize Lines

- Given:
  - Segment endpoints (integers x1, y1; x2, y2)
- Identify:
  - Set of pixels (x, y) to display for segment

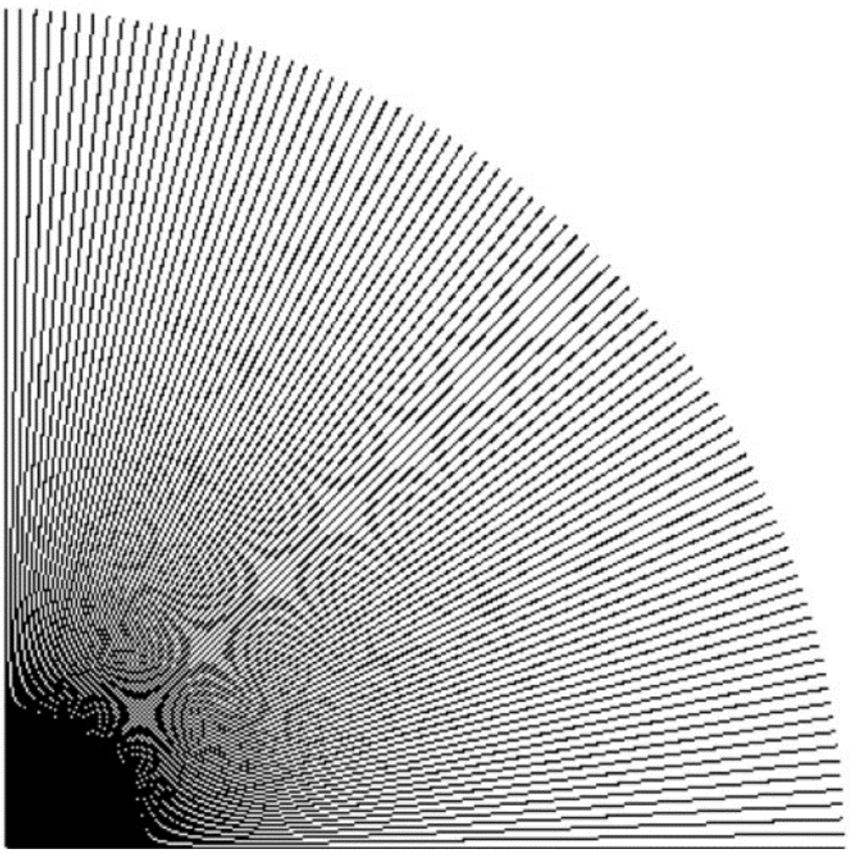


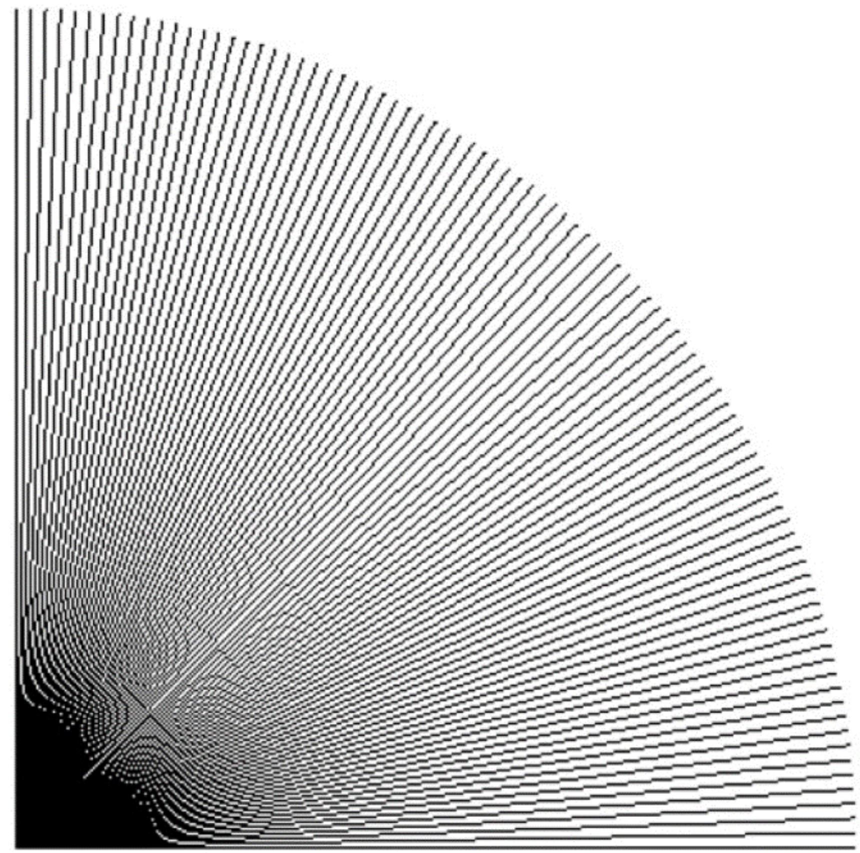
(Durand and Cutler MIT)

Requirements

- Transform continuous primitive into discrete samples
- Uniform thickness & brightness
- Continuous appearance
- No gaps
- Accuracy
- Speed

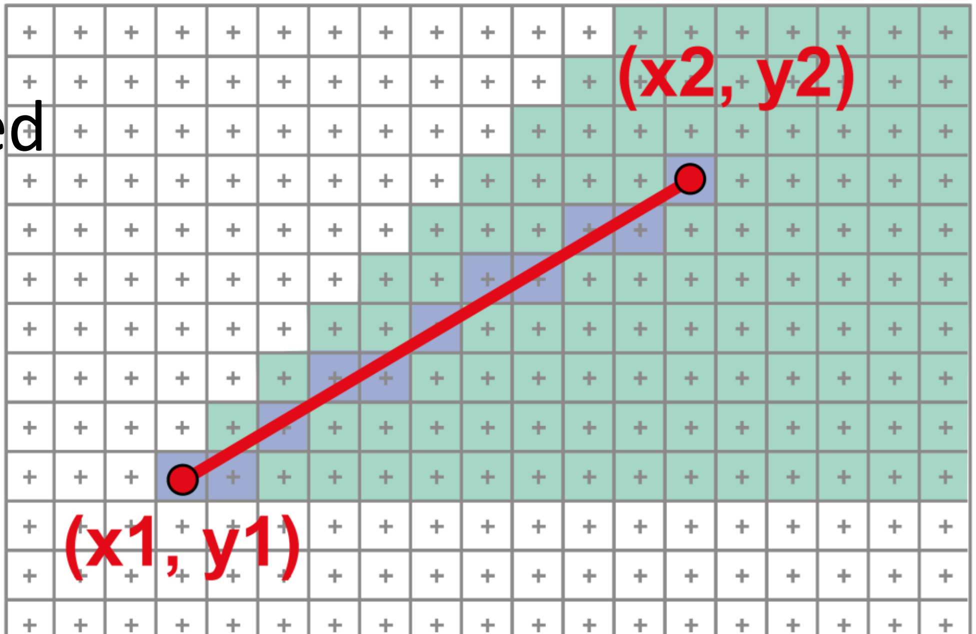
Comparison





Think about?

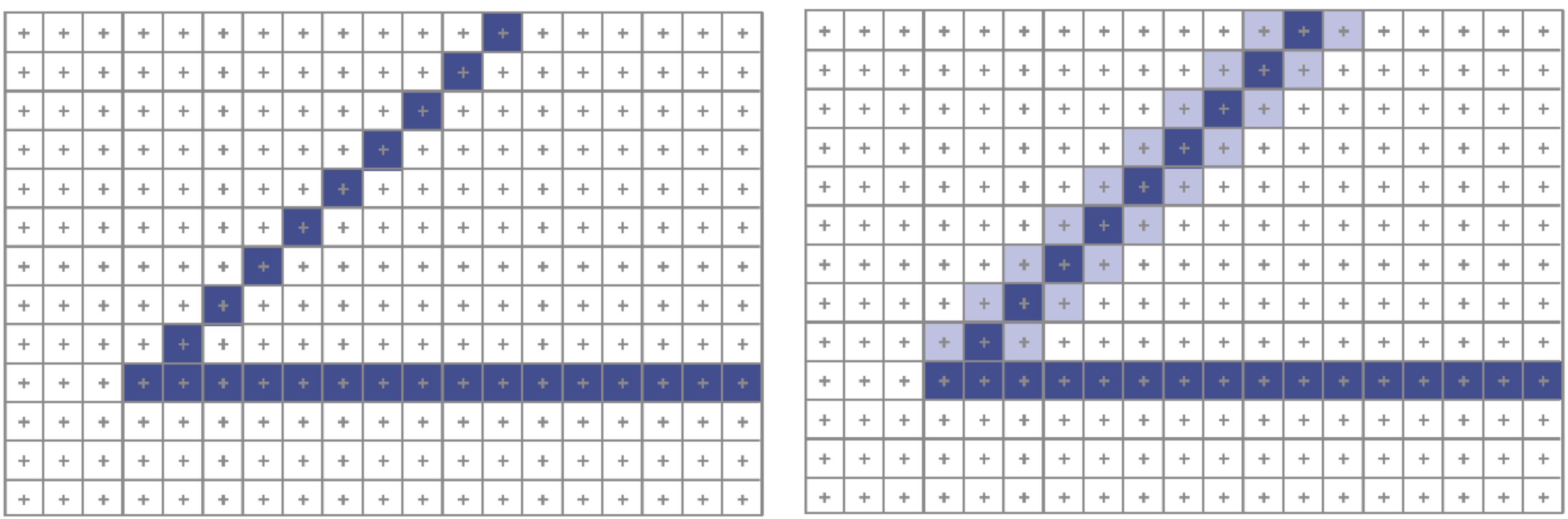
- Assume:
  - $m = \frac{dy}{dx}, 0 < m < 1$
  - $x2 > x1$
- One pixel per column
  - Fewer -> disconnected
  - More -> thick



(Durand and Cutler MIT)

Think about?

- Note: brightness can vary with slope
  - What is the maximum variation?
- How can we compensate for this?
  - Antialiasing



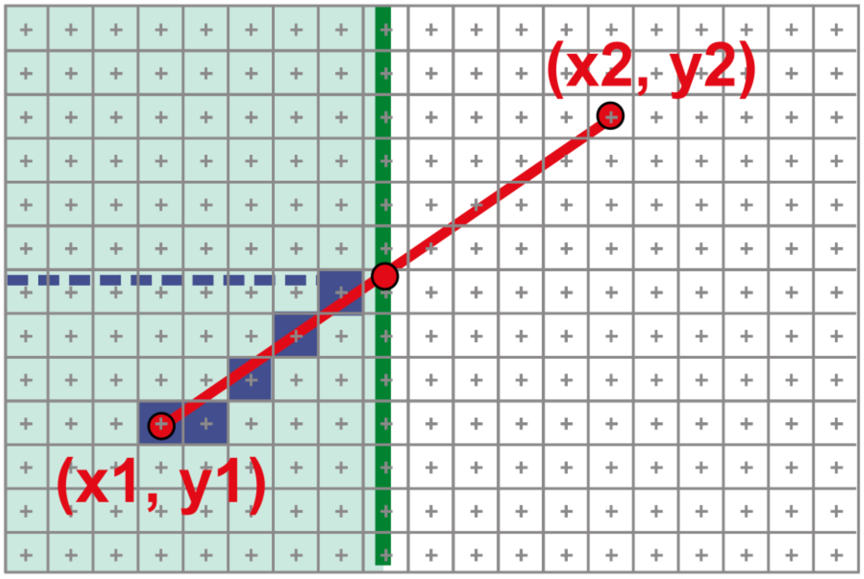
(Durand and Cutler MIT)

The naive method

- Simply compute y as a function of x
  - Conceptually: move vertical scan line from x1 to x2

$$\Delta x = 1$$

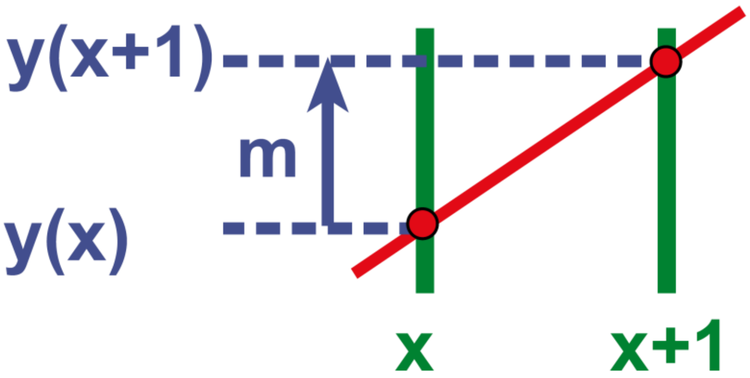
  - What is the expression of y as function of x?
  - Set **pixel (x, round (y(x)))**
- $y = y1 + m(x - x1)$
- $m = \frac{dy}{dx}$



(Durand and Cutler MIT)

Improvement

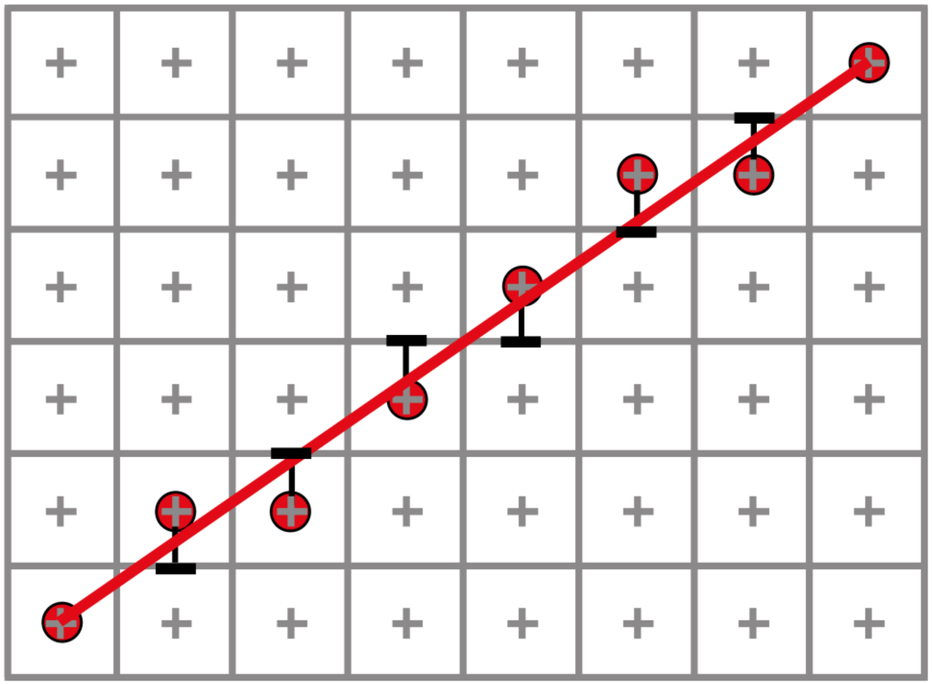
- Computing y value is expensive
  - $y_i = y1 + m(x_i - x1)$
  - $m = \frac{dy}{dx}$
- Observe:
  - $y_i = y1 + m(x_i - x1)$
  - $y_{i+1} = y1 + m(x_i + \Delta x - x1)$
    $= y_i + m\Delta x$
- In each step, y += m, x += 1 !



(Durand and Cutler MIT)

DDA (Digital Difference Analyzer)

- Select pixel vertically closest to line segment
  - intuitive, efficient,
  - pixel center always within 0.5 vertically



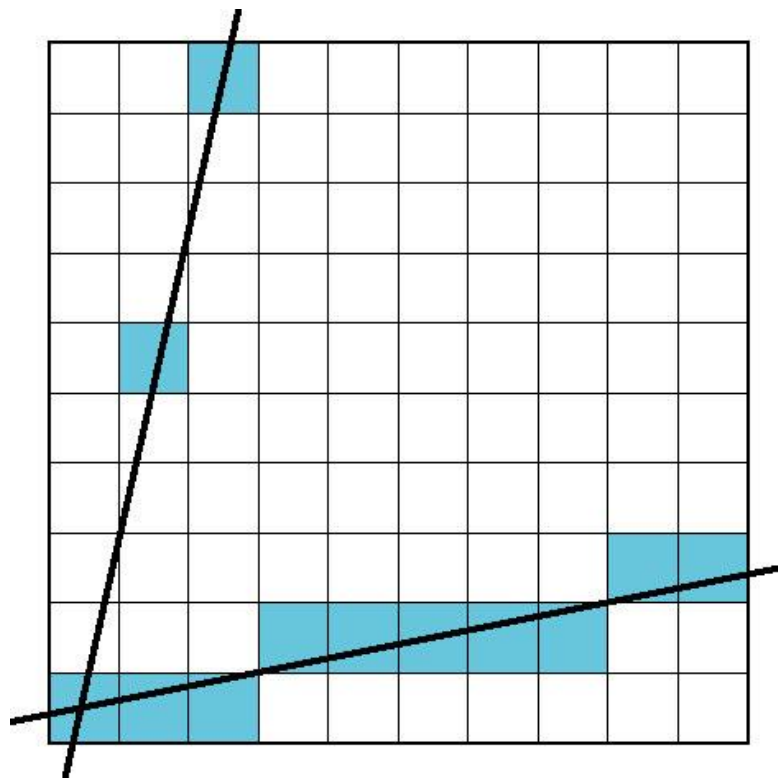
(Durand and Cutler MIT)

DDA pseudo code

```
For(x=x1; x<=x2,x++) {
  y+=m;
  SetPixel(x, round(y));
}
```

How to solve general cases?

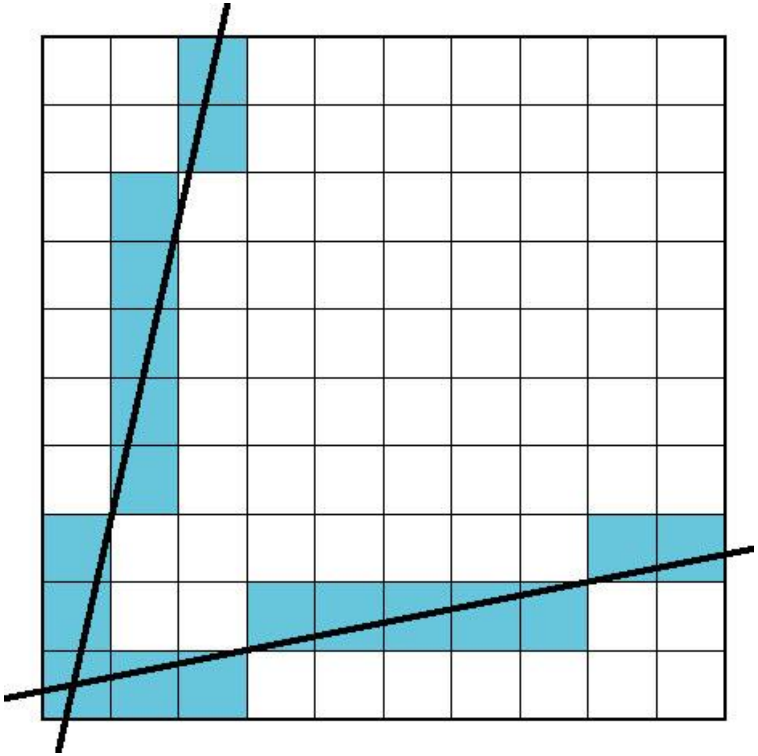
- Previously, we assumed x2 > x1 , $0 < m < 1$
- E.g. When x2 < x1? when $m > 1$?



(Eg Angel)

Use Symmetry

- For m > 1, swap role of x and y
  - For each y, plot closest x
  - $x_{i+1} = x_i + \frac{1}{m}$



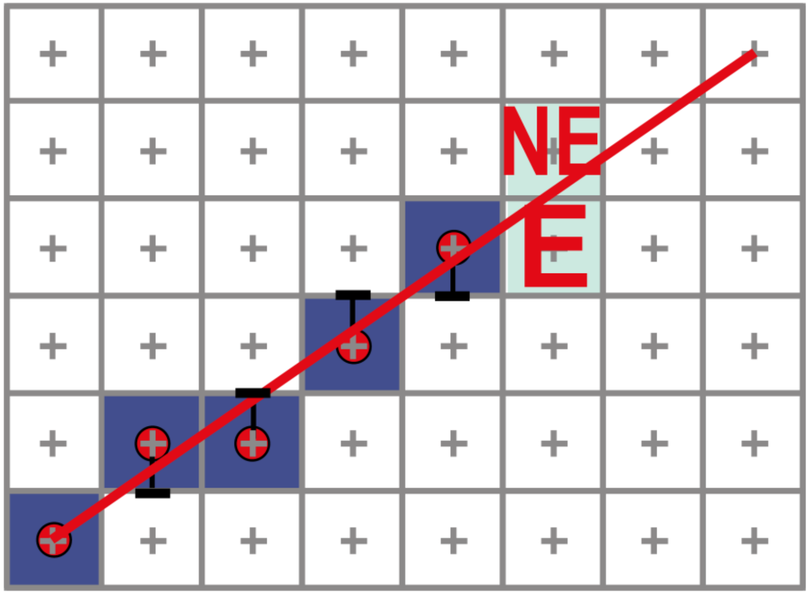
(Eg Angel)

DDA pseudo code

```
dx = x2-x1;dy = y2-y1;
If |dx| > |dy| then step = dx else step =
dy;
For(i=0; i<step,i++) {
  x+=dx/steps;
  y+=dy/steps;
  SetPixel(round(x), round(y));
}
```

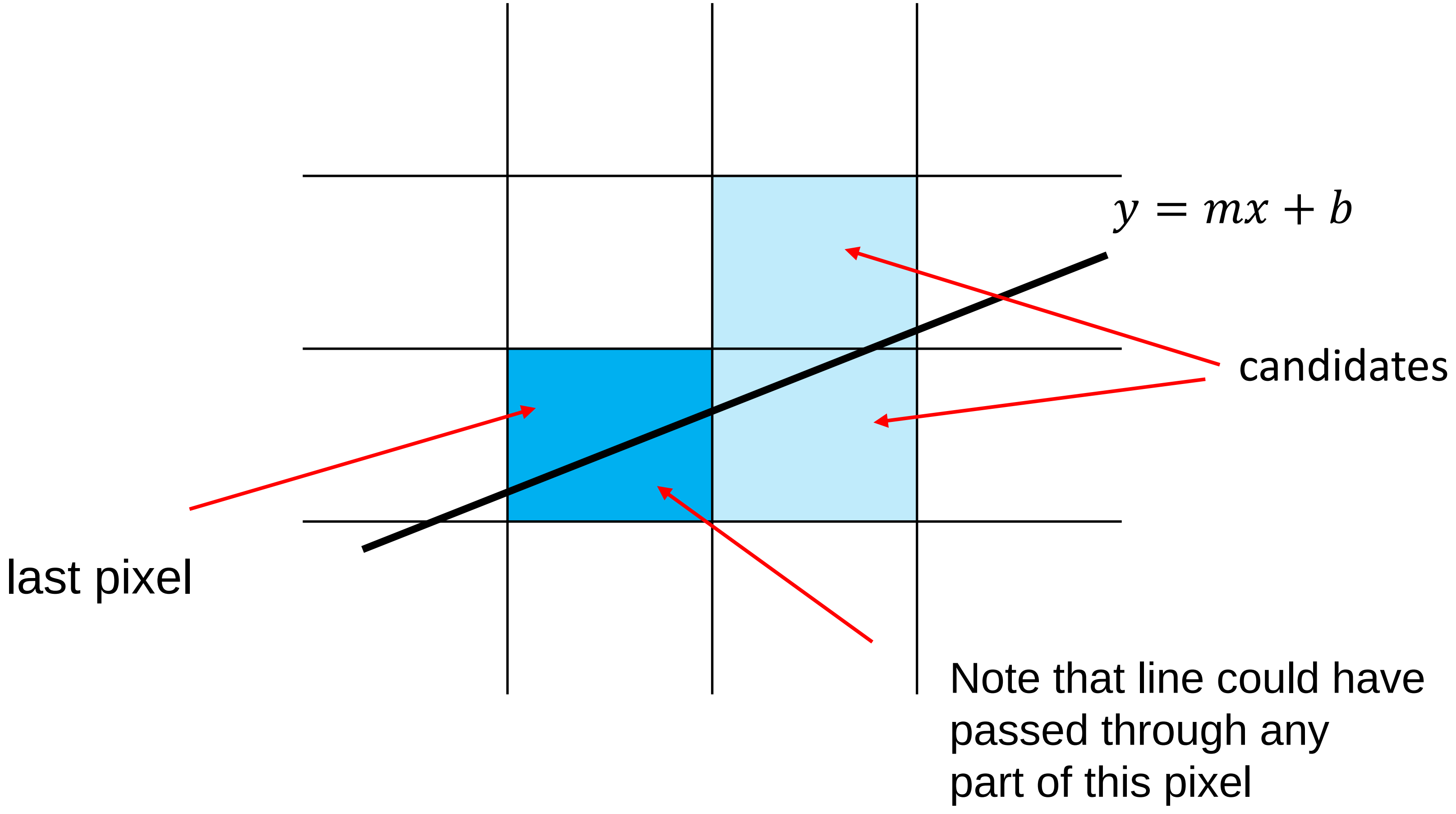
Bresenham’s algorithm

- The short comings of DDA
  - Floating point arithmetic
  - Round operation
- Observe again:
  - If we're at pixel P(x_p, y_p), the next pixel must be
  - either E ($x_p + 1, y_p$) or
  - NE ($x_p + 1, y_p + 1$)
  - Why?

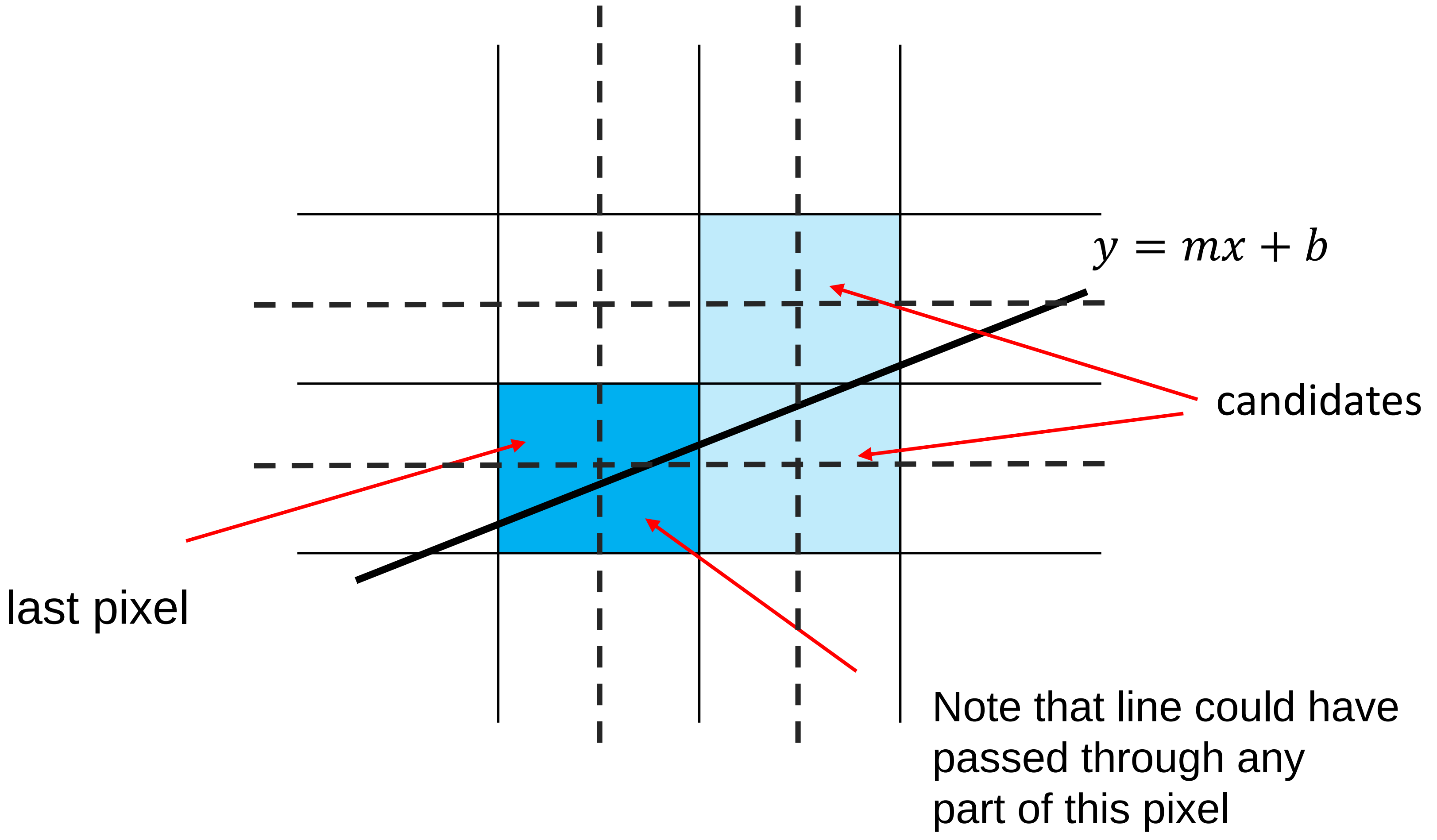


(Durand and Cutler MIT)

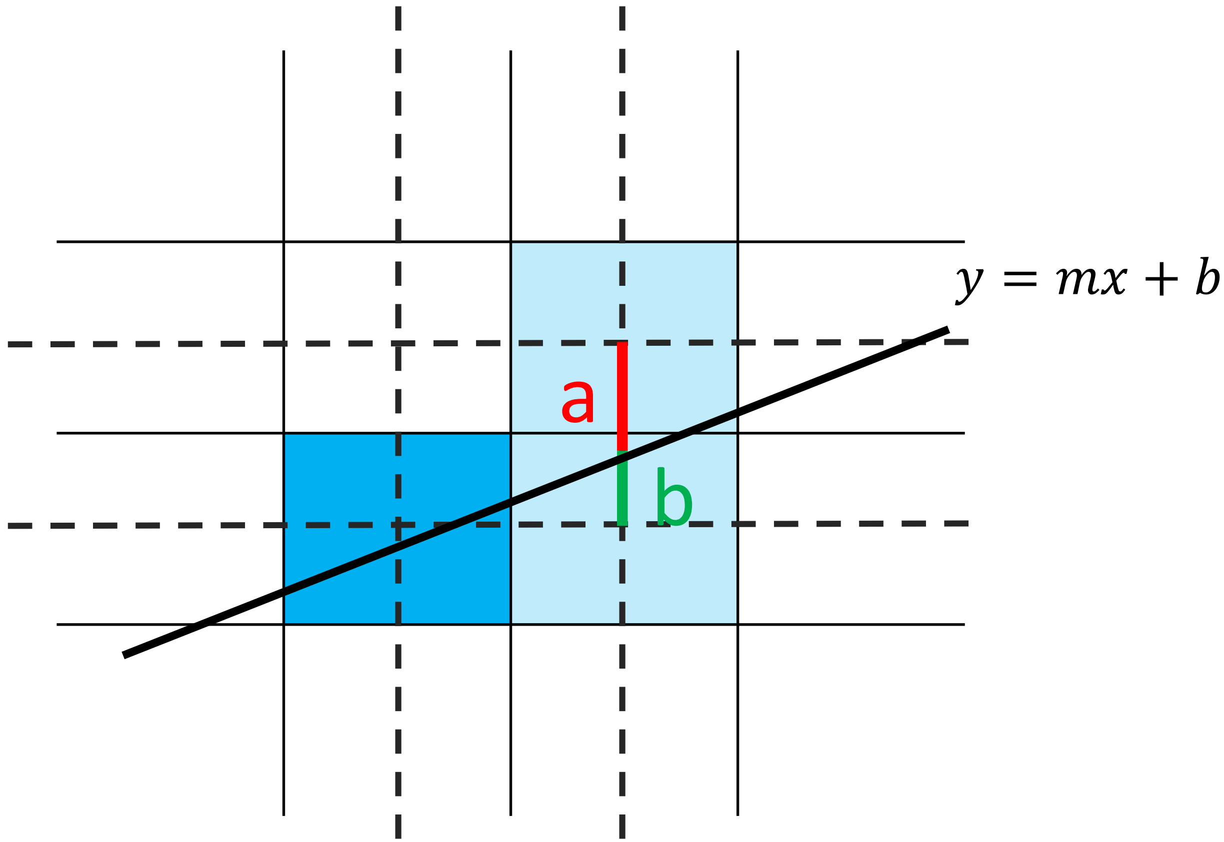
Candidate Pixels



Candidate Pixels

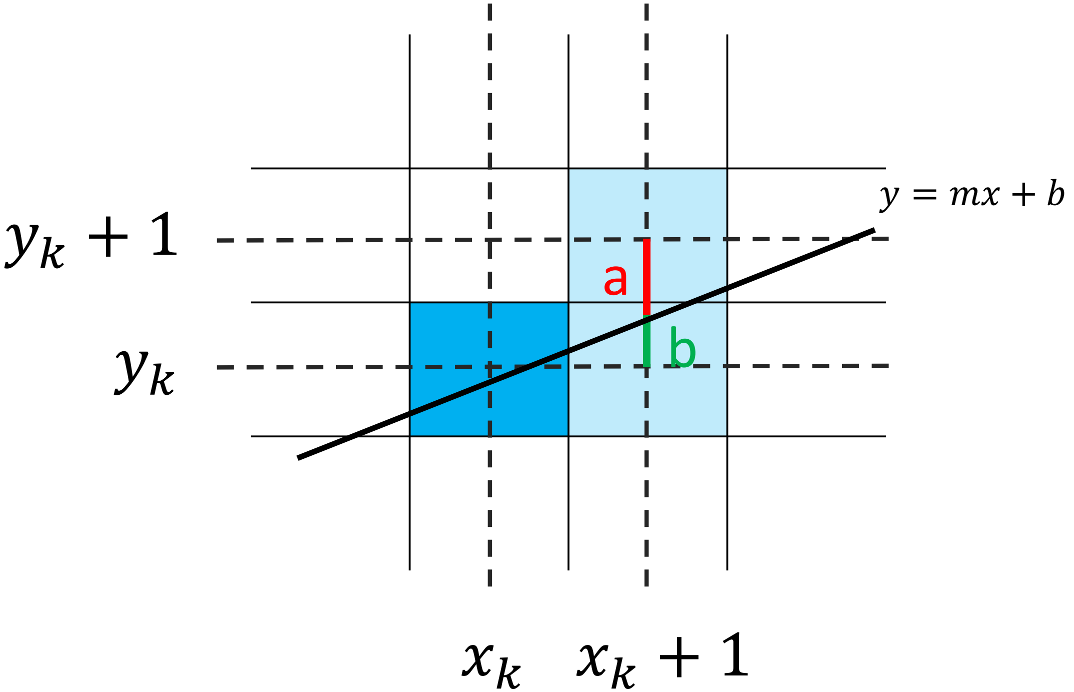


The decision function



The decision parameter

- If $a < b$
  - Then upper pixel
- If $a > b$
  - Then lower pixel
- $a = (y_k + 1) - y = y_k + 1 - m(x_k + 1) - b$
- $b = y - y_k = m(x_k + 1) + b - y_k$
- $let\ d = \Delta x(b - a) = 2\Delta y \cdot x_k - 2\Delta x \cdot y_k + c$



d is an integer
d > 0 use upper pixel
d < 0 use lower pixel

Bresenham's algorithm

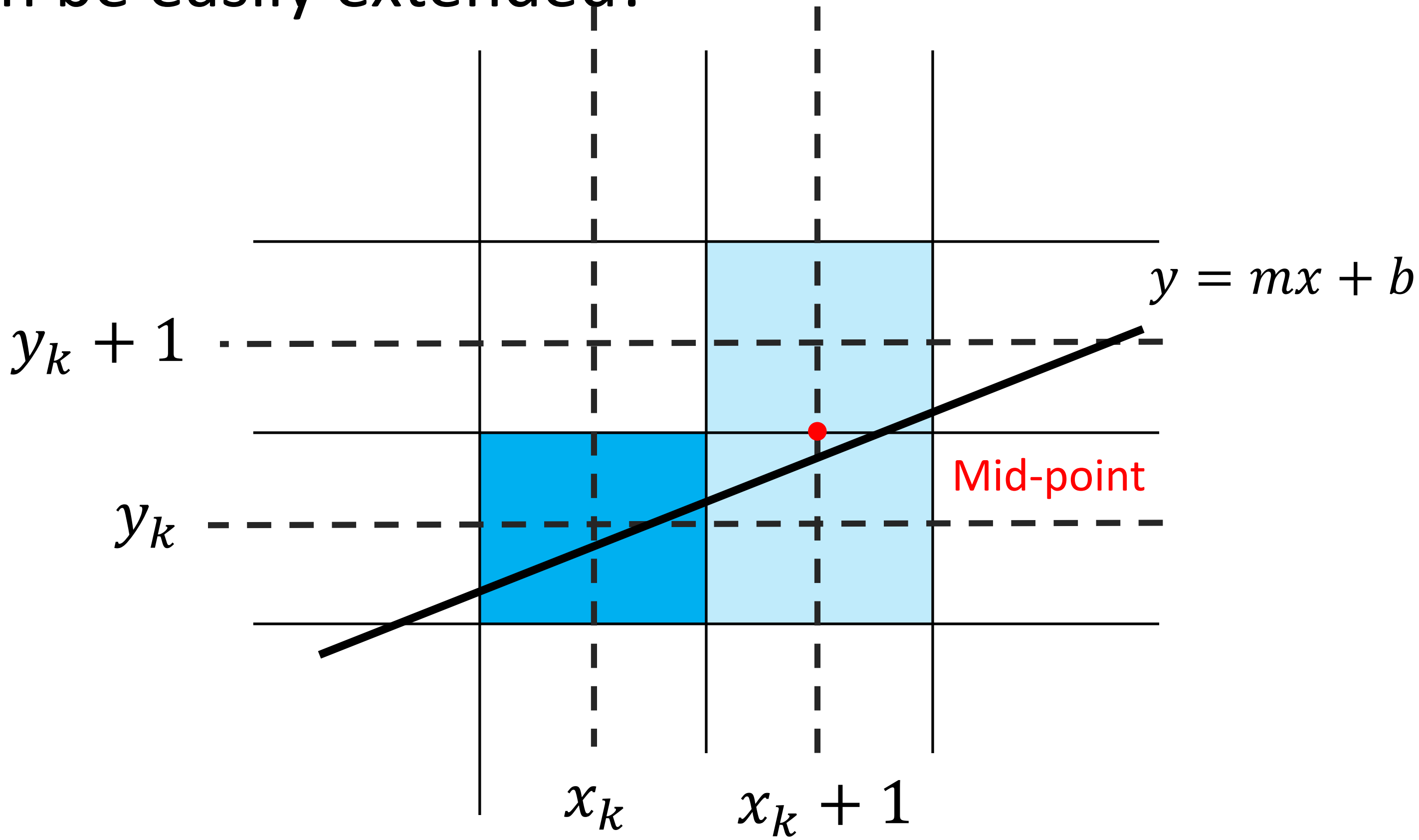
- Further observation:
- $d_k = 2\Delta y \cdot x_k - 2\Delta x \cdot y_k + c$
- $d_{k+1} = 2\Delta y \cdot x_{k+1} - 2\Delta x \cdot y_{k+1} + c$
- $d_{k+1} - d_k = 2\Delta y \cdot (x_{k+1} - x_k) - 2\Delta x \cdot (y_{k+1} - y_k)$
- Here:
  - $x_{k+1} - x_k = 1, if\ 0 < m < 1$
  - $y_{k+1} - y_k = 0\ or\ 1, depending\ on\ the\ sign\ of\ d_k$
  - $d_0 = 2\Delta y - \Delta x$ (substitute y_0 to d_k)

Bresenham's algorithm

- Given x_1, y_1, x_2, y_2
- Calculate $\Delta x, \Delta y$
- $d_0 = 2\Delta y - \Delta x$
- SetPixel(x_0, y_0)
- At each x_k, k starts from k=0, do $\Delta x - 1$ steps
    - If d_k >0, SetPixel(x_k+1, y_k +1) and
    $$d_{k+1} = d_k + 2\Delta y - 2\Delta x$$
    - If d_k <0, SetPixel(x_k+1, y_k) and
    $$d_{k+1} = d_k + 2\Delta \mathrm{y}$$

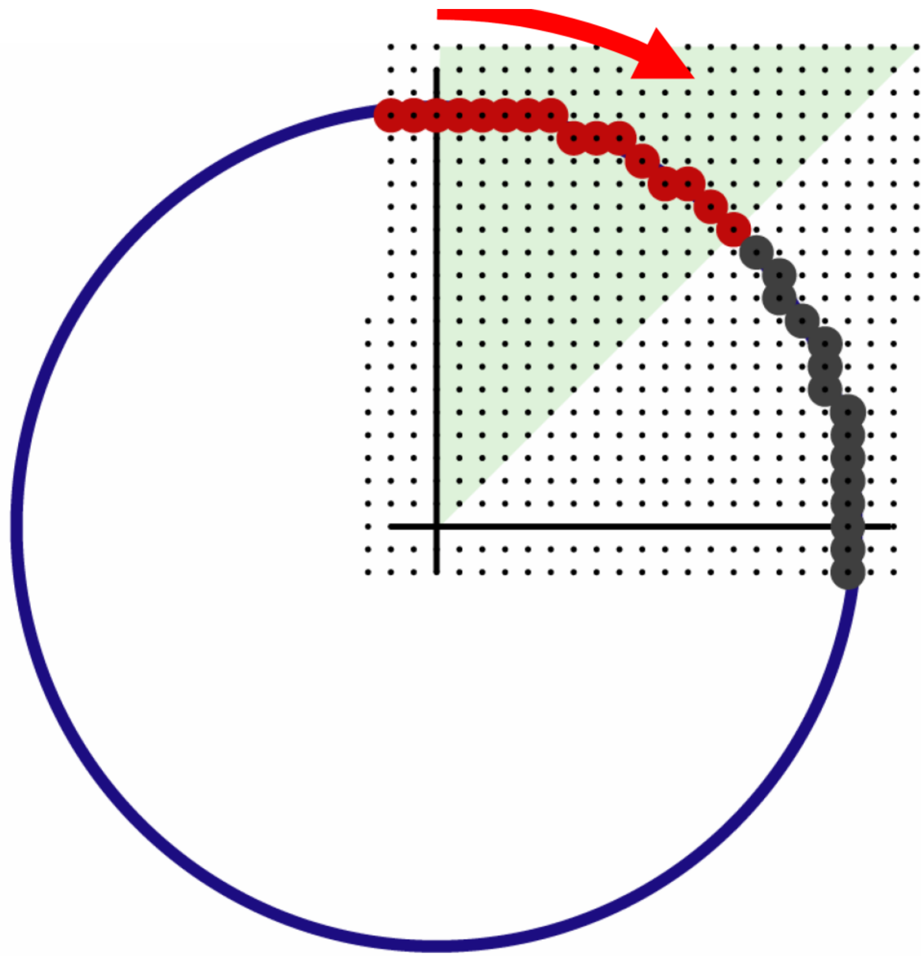
Mid-point algorithm

- Similar to Bresenham's
- But can be easily extended!

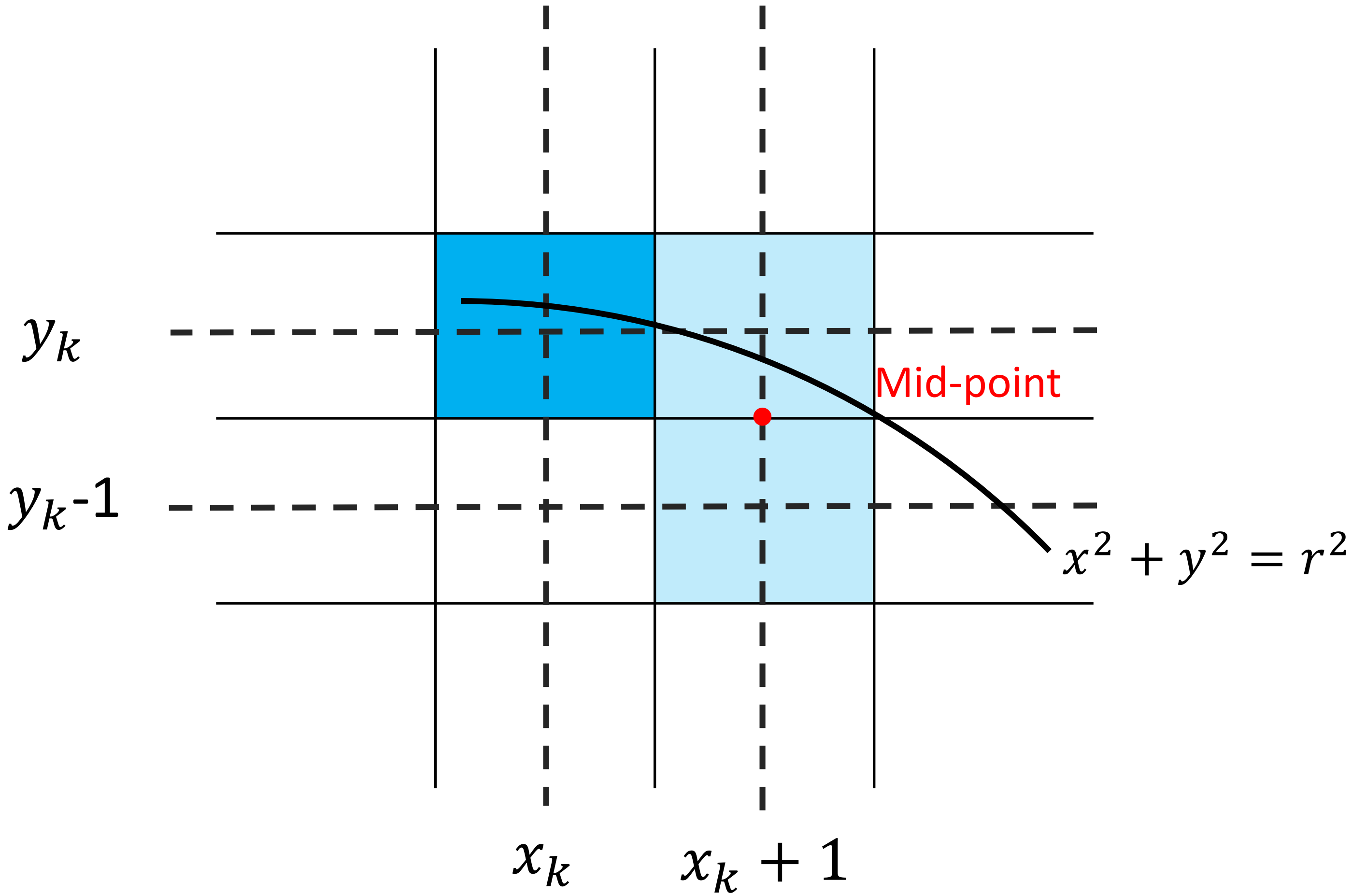


Curve

- Take Circle as an example
- Generate pixels for 1/8 octant only
- Slope progresses from 0 →1
- Analog of Bresenham's Algorithm

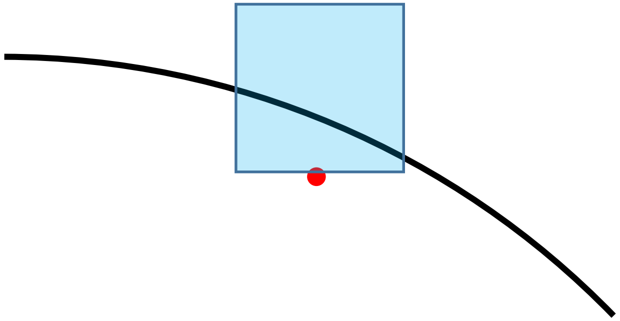


(Durand and Cutler MIT)

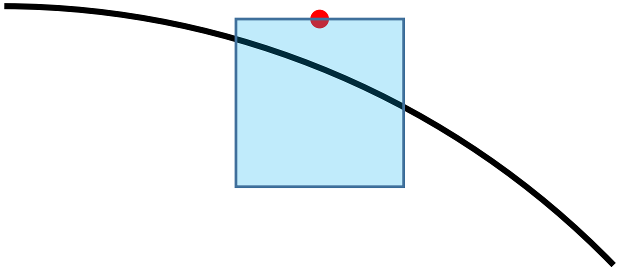

y_k
y_k-1
Mid-point
$x^2 + y^2 = r^2$
x_k
$x_k + 1$

Midpoint Circle Algorithm

- Decision Function:
  - $d_k = D(x_k + 1, y_k - 0.5) = (x_k+1)^2 + (y_k - 0.5)^2 - r^2$
  - If $d_k < 0$, the mid point is inside the circle, choose upper



  - If $d_k > 0$, the mid point is outside the circle,choose lower
  - If $d_k = 0$, the mid point is on the circle,choose lower



Midpoint Circle Algorithm

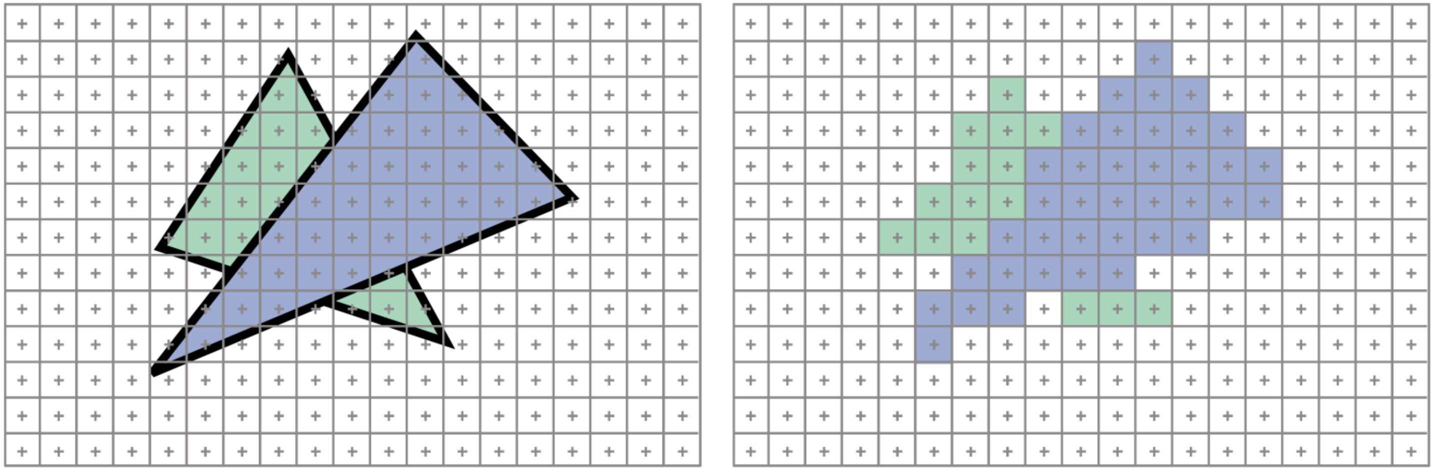
- Observation
  - $d_{k+1} = D(x_{k+1} + 1, y_{k+1} - 0.5) = (x_{k+1}{+}1)^2 + (y_{k+1} + 0.5)^2 - r^2$
  - $d_{k+1} = d_k + 2(x_k + 1) + \left(y_{k+1}^2 - y_k^2\right) - (y_{k+1} - y_k) + 1$
  - $y_{k+1}$is either y_k or y_{k-1}, depending on the sign of d_k

  - On each iteration:
  - $x \mathrel{+}= 1$
  - $d_k \mathrel{+}= 2x_{k+1} + 1$ if $d_k < 0$
  - $d_k \mathrel{+}= 2x_{k+1} + 1 - 2y_{k+1}$ if $d_k > 0$

- Questions?

Rasterize Polygons

- Scan Conversion = Fill Polygons

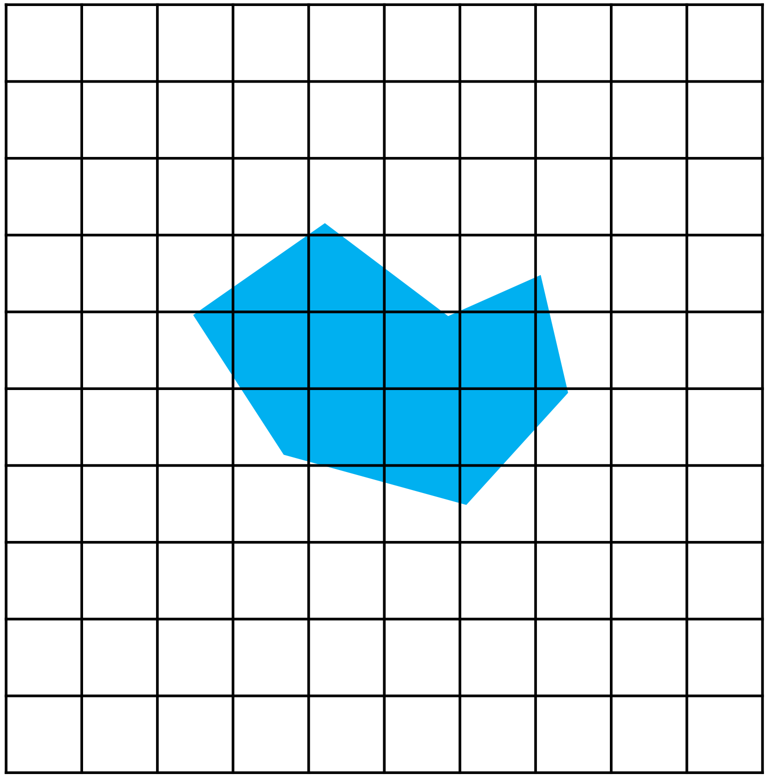


(Durand and Cutler MIT)

Brute force solution

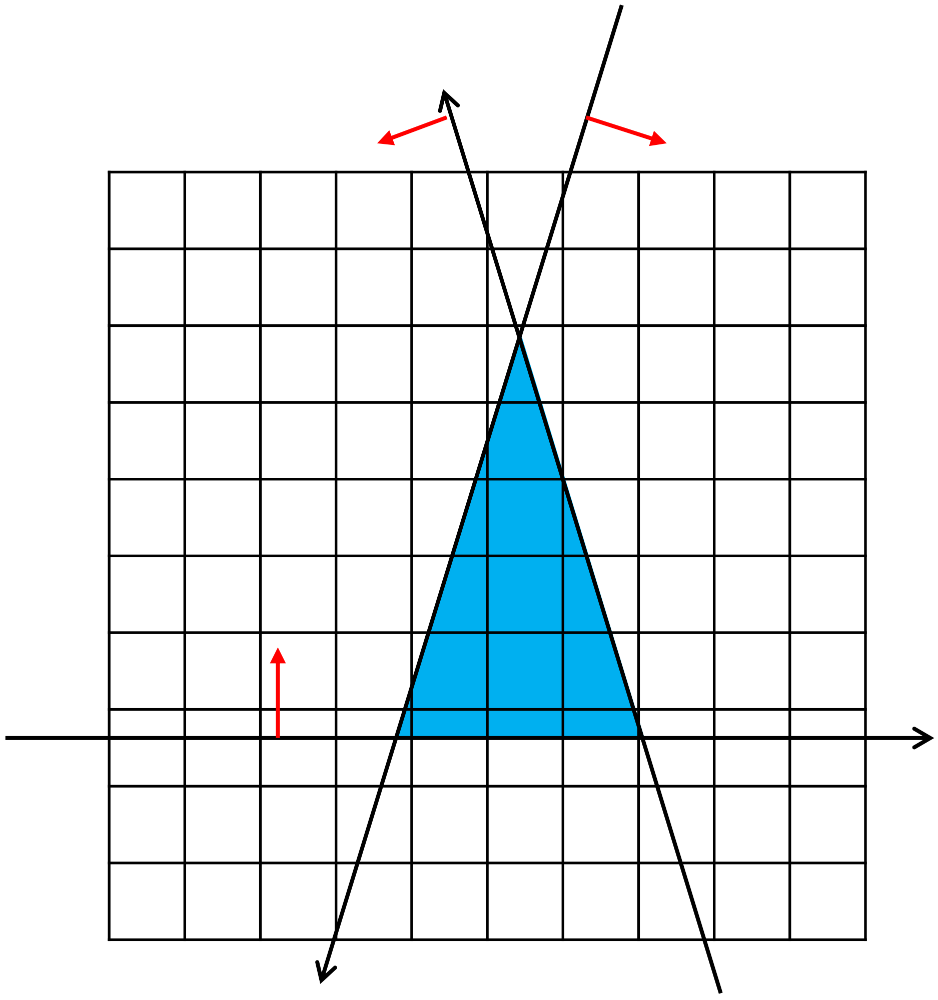
1. Enumerate all the pixels
2. Test whether a pixel is inside of the polygon
3. If inside then draw the pixel

But how to know if a pixel is inside or not?



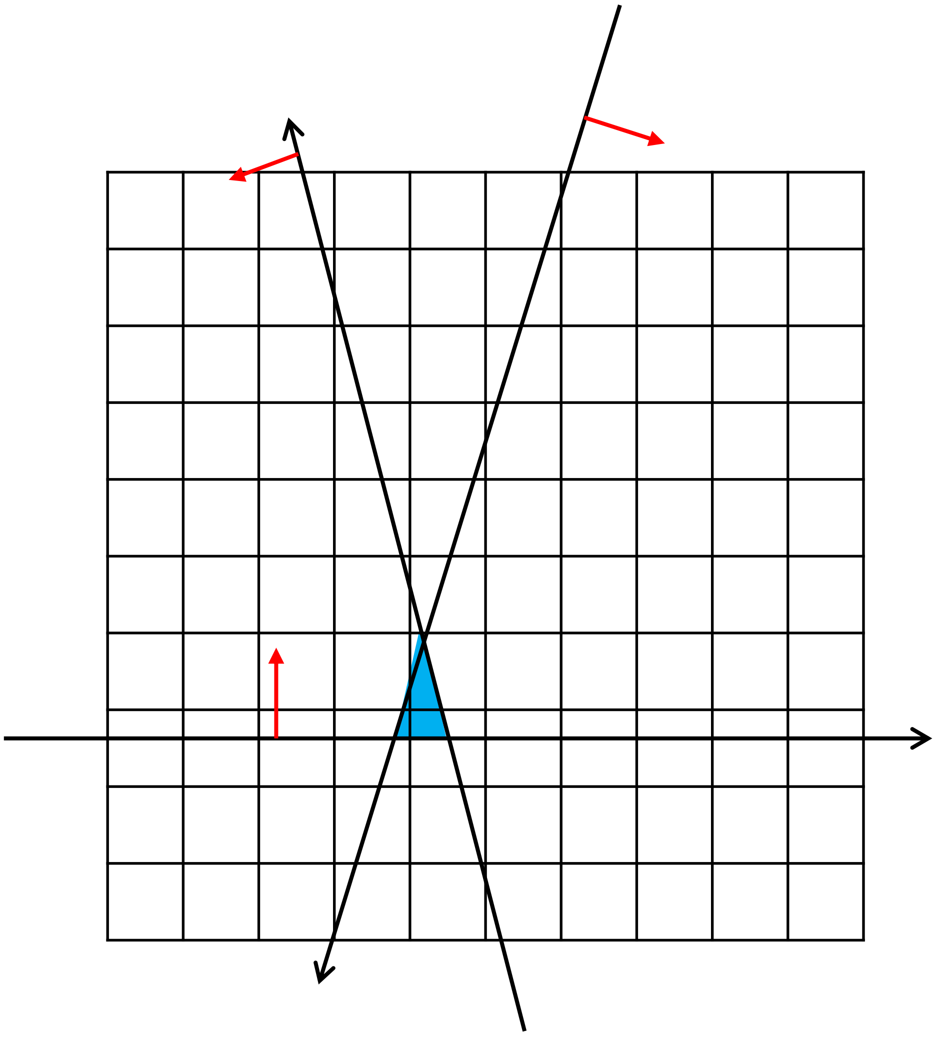
Triangles

- Simplest polygon
- how to know if a pixel is inside or not?
- "clip" against the triangle



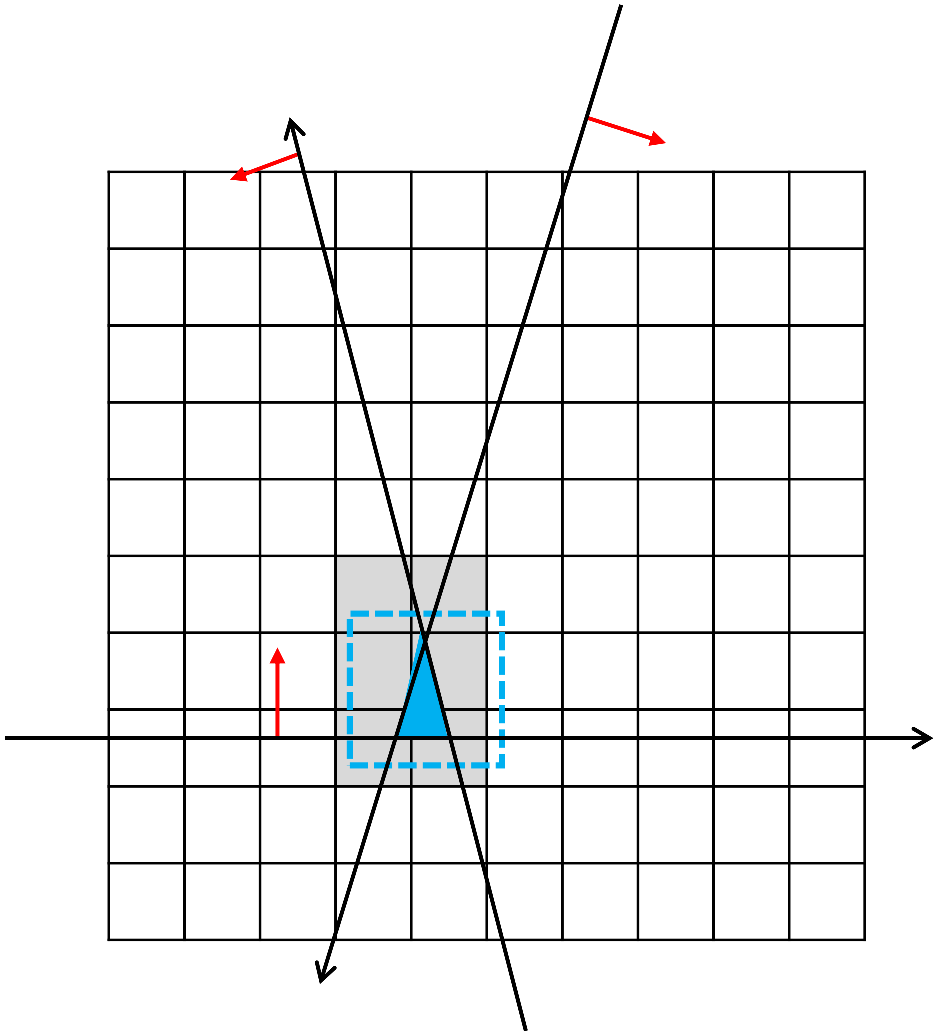
Triangles

- Problem?
- A lot of use less computation when the triangle is small
- Always the case



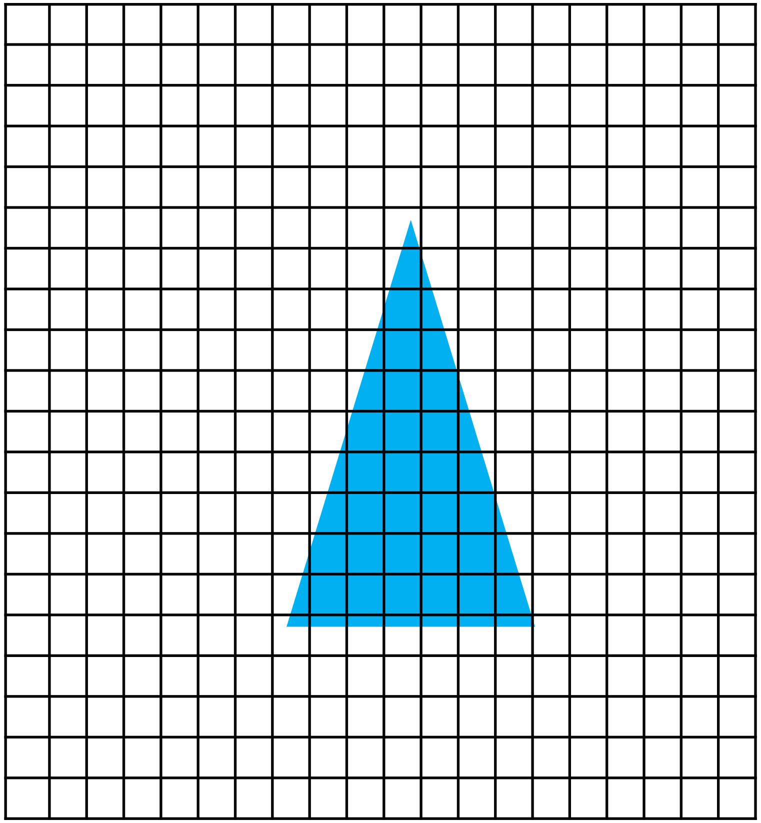
Triangles

- Improvement
- Bounding box of the triangle
- Only compute the enclosed pixels



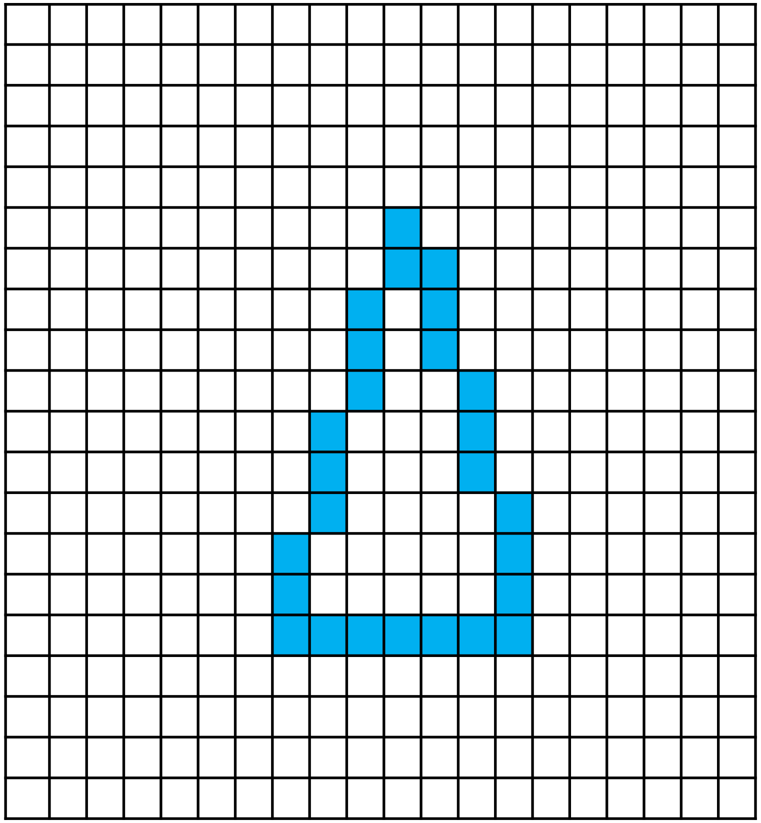
Triangles

- However, we have to compute the line equations which are expensive
- Take the advantage of line rasterization



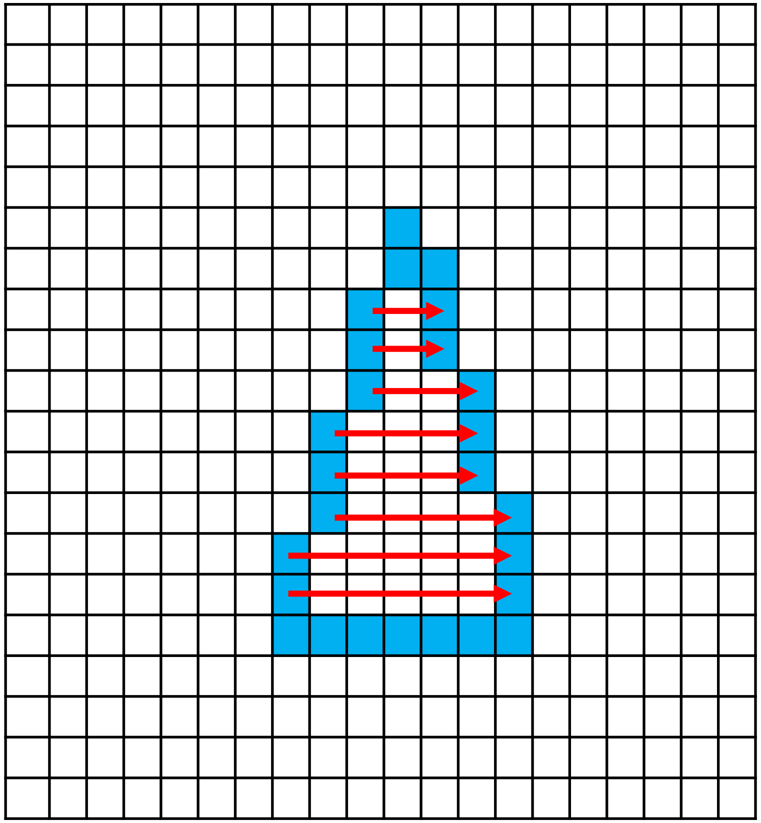
Triangles

- However, we have to compute the line equations which are expensive
- Take the advantage of line rasterization



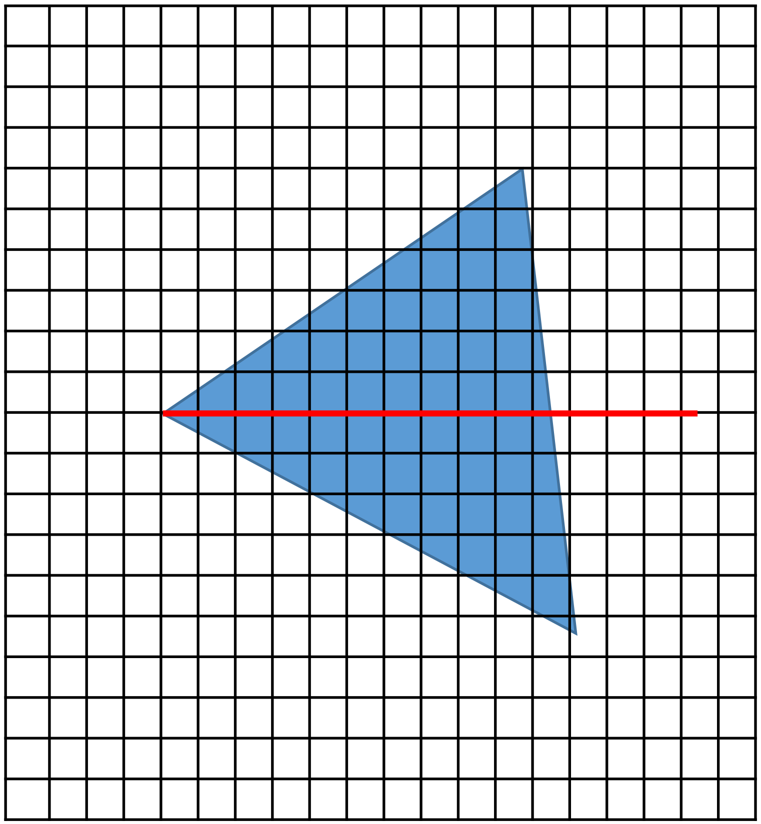
Triangles

- Compute the boundary pixels
- Fill the spans
- How?
  - Draw edges vertically
  - Fill in horizontal spans for each scanline
- Take advantage of spatial coherence
- Take advantage of edge linearity



Triangles

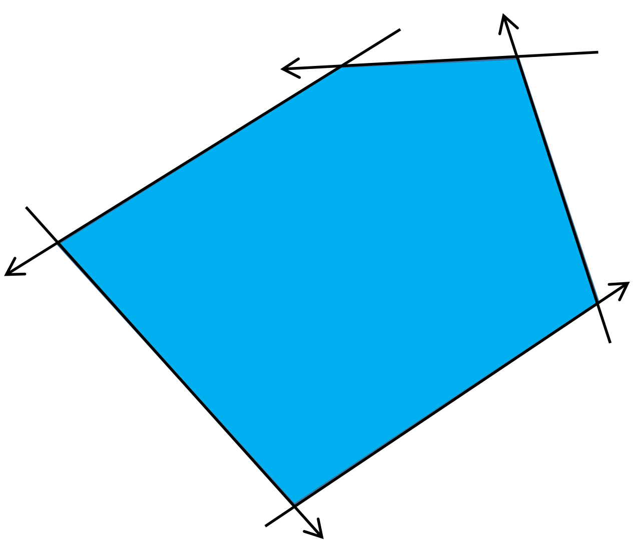
- How about this?
- The line changes at one vertex
- One way is to split
- Why?

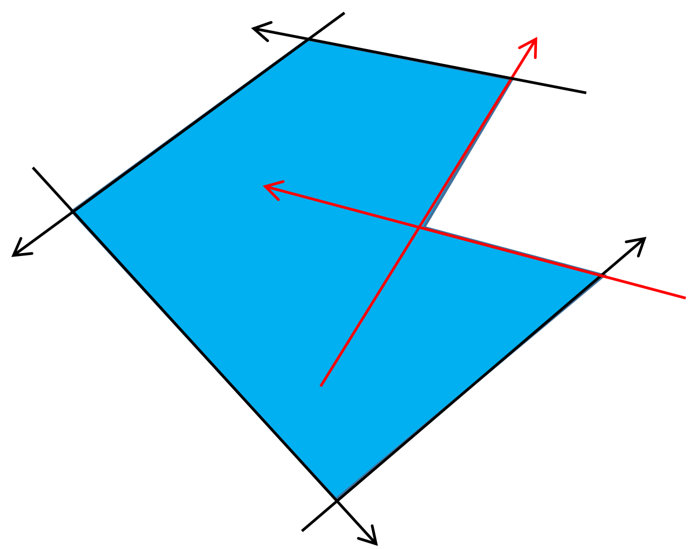


- Questions?

Inside or outside test

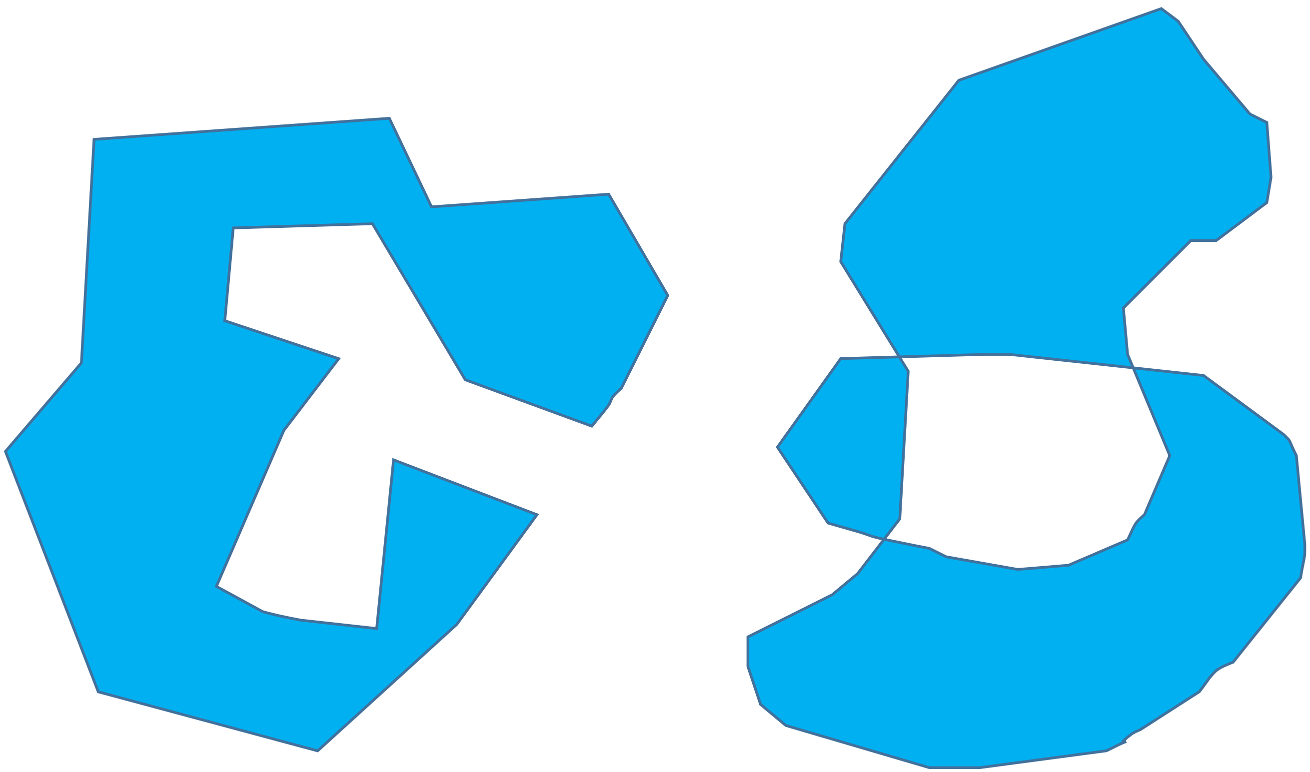
- How to know if a pixel/point is inside of a polygon or not?
  - Triangle method works only for convex ones





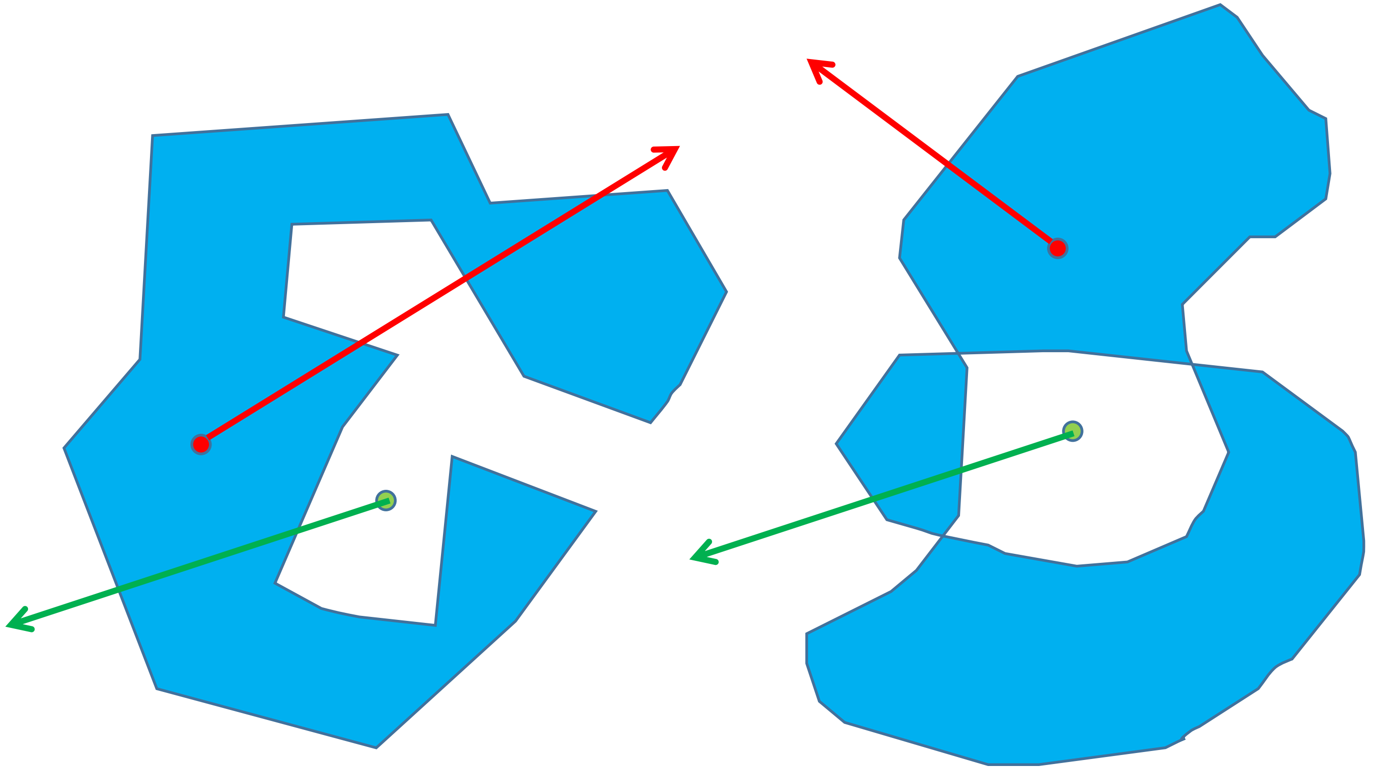
Various polygons

- Can be messy!



Inside polygon rule

- Odd-Even rule



Flood Fill

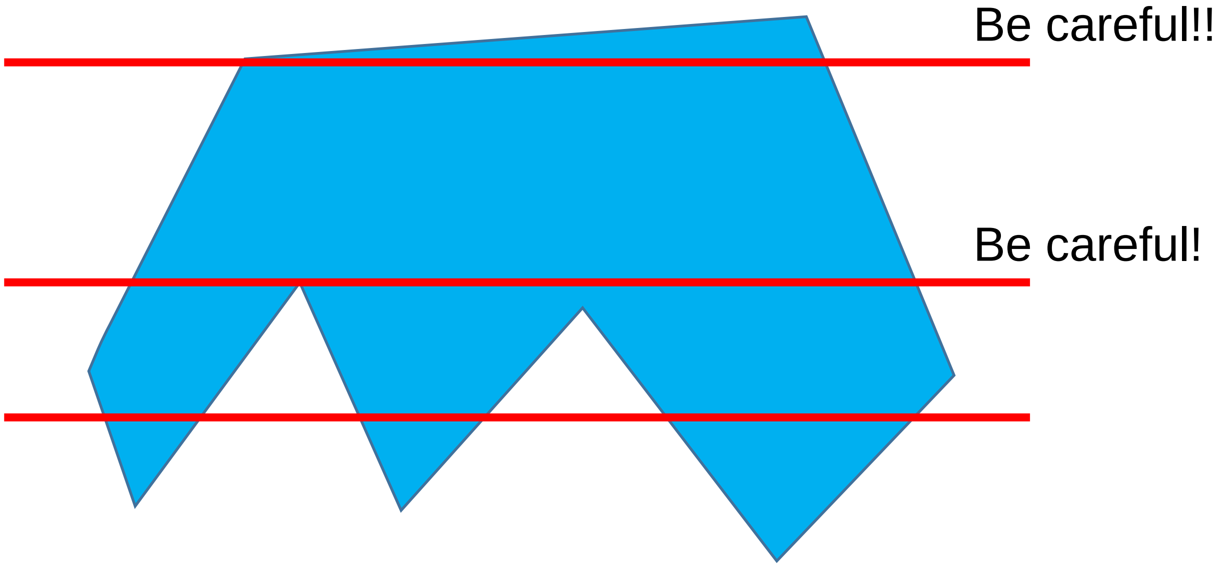
- Fill can be done recursively if we know a seed point located inside (WHITE)
- Scan convert edges into buffer in edge/inside color (BLACK)

```
flood_fill(int x, int y) {
    if(read_pixel(x,y)= = WHITE) {
       write_pixel(x,y,BLACK);
       flood_fill(x-1, y);
       flood_fill(x+1, y);
       flood_fill(x, y+1);
       flood_fill(x, y-1);
}   }
```

(Eg Angel)

Inside polygon rule

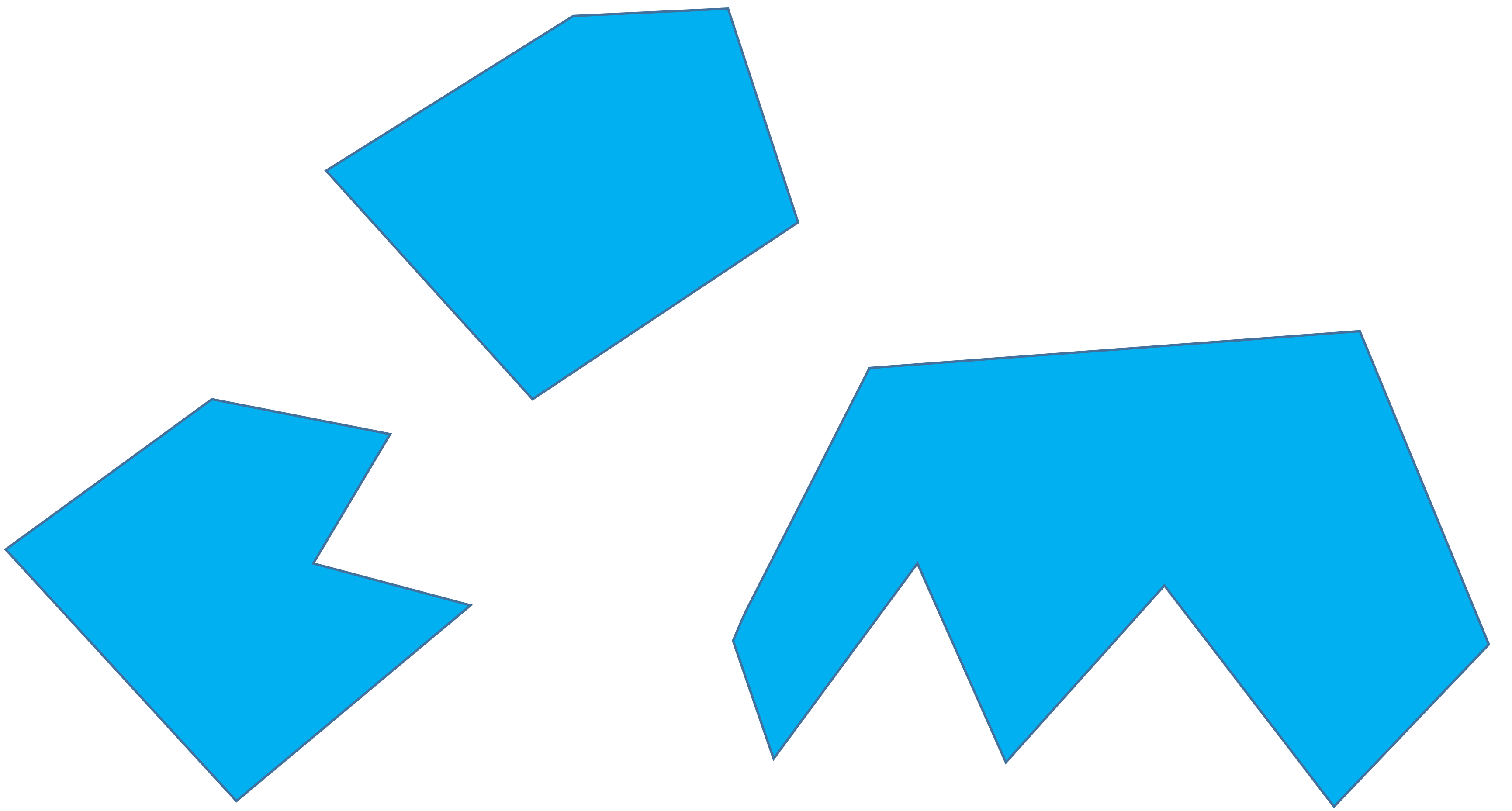
- Odd-parity rule
  - Set parity even
  - Invert parity at each intersection
  - Draw pixels when parity if odd



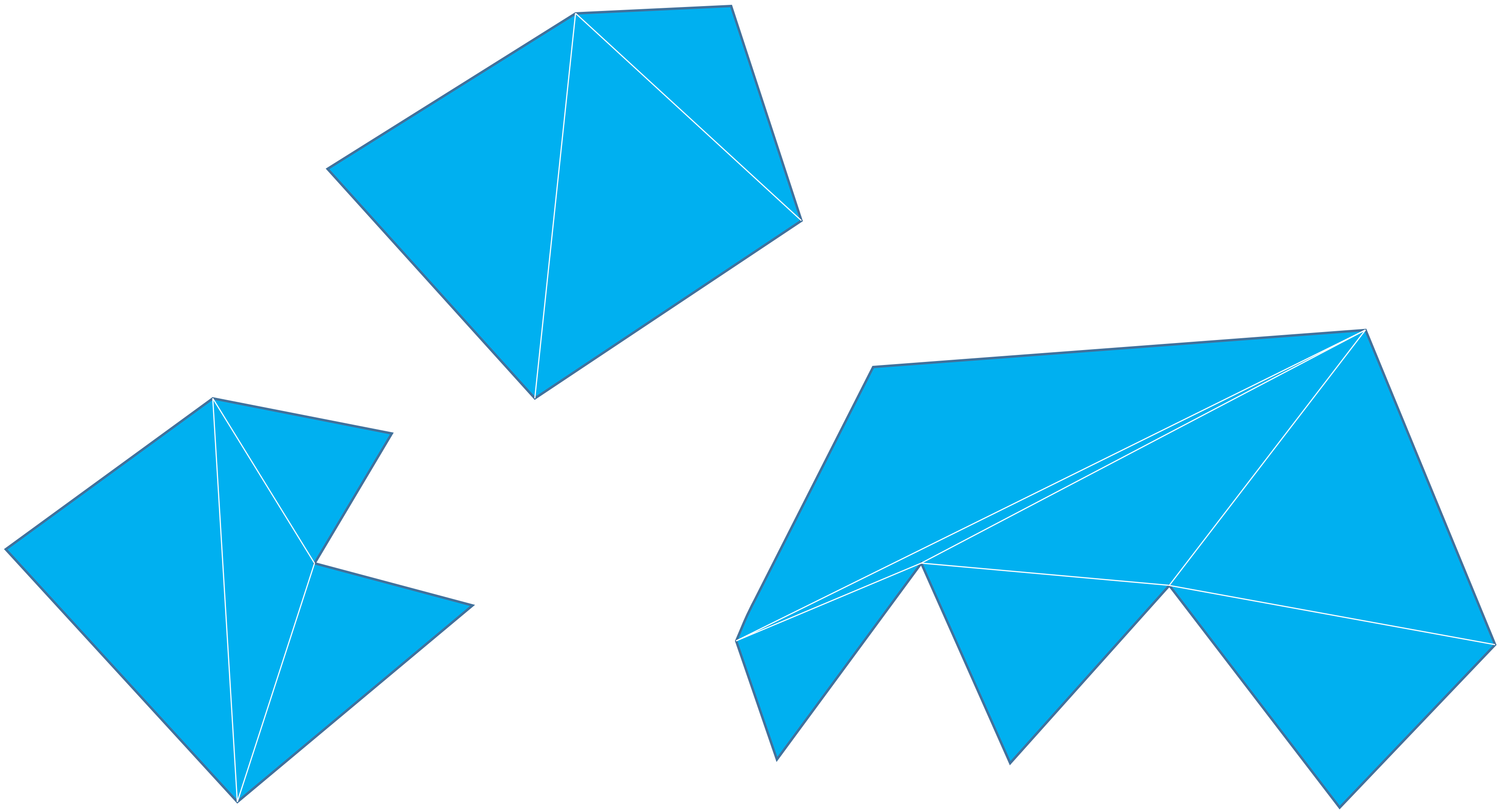
Triangulation

- Rasterization algorithms can take advantage of triangle properties
- Graphics hardware is optimized for triangles
- Because triangle drawing is so fast, many systems will subdivide polygons into triangles prior to scan conversion
- How?

Triangulation

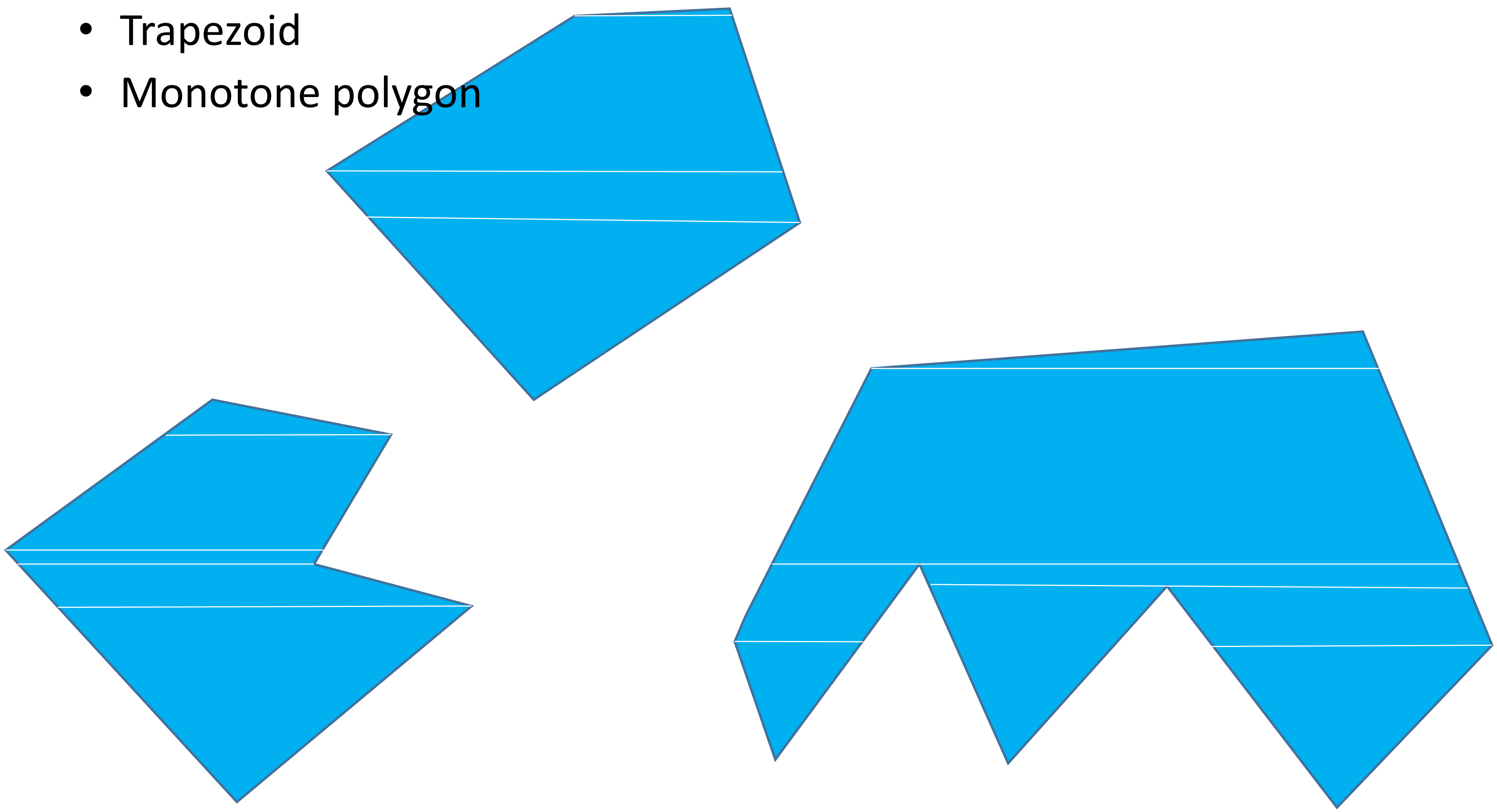


Ear-clipping



Trapezoid decomposition

- Seidel's Algorithm
  - Trapezoid
  - Monotone polygon

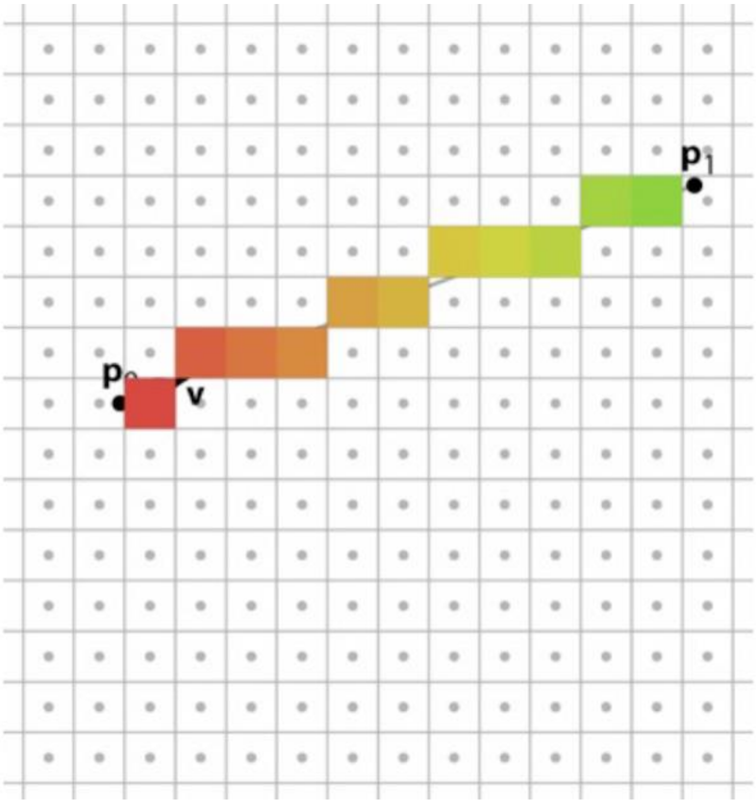


Interpolation of attributes

- Rasterizer does not only select pixels
- Attributes are interpolated during rasterization
    - Depth
    - Color
    - Texture coords
    - Many others

Linear interpolation

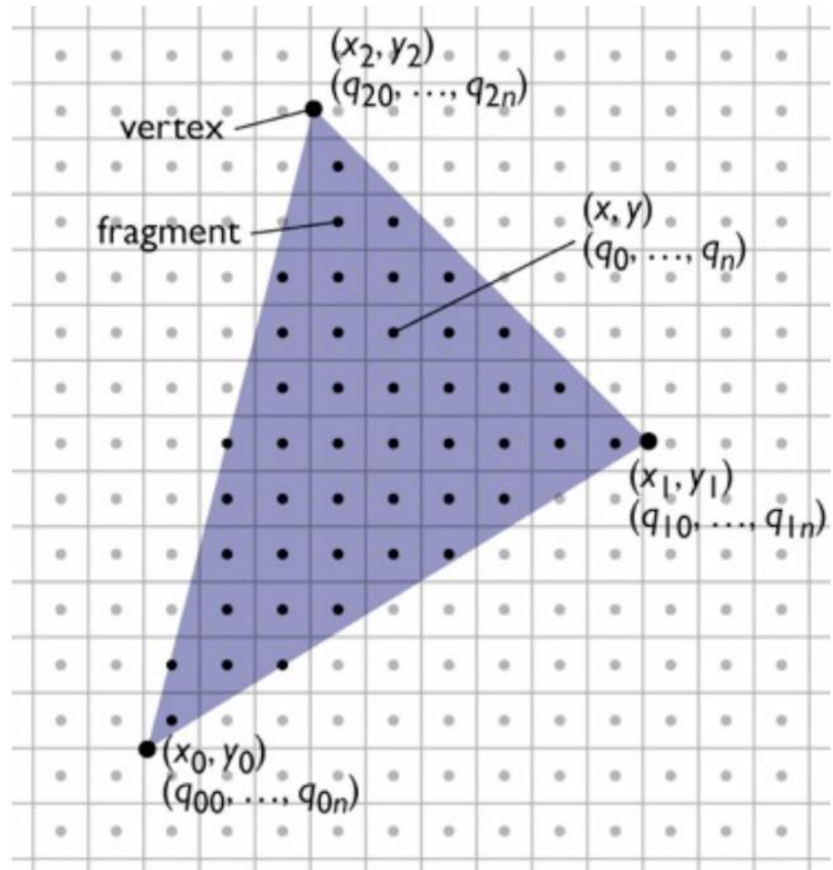
- Define 2D function by projection on line
  - this is linear in 2D
  - therefore can use DDA to interpolate



(2013 Steve Marschner)

Interpolation for triangles

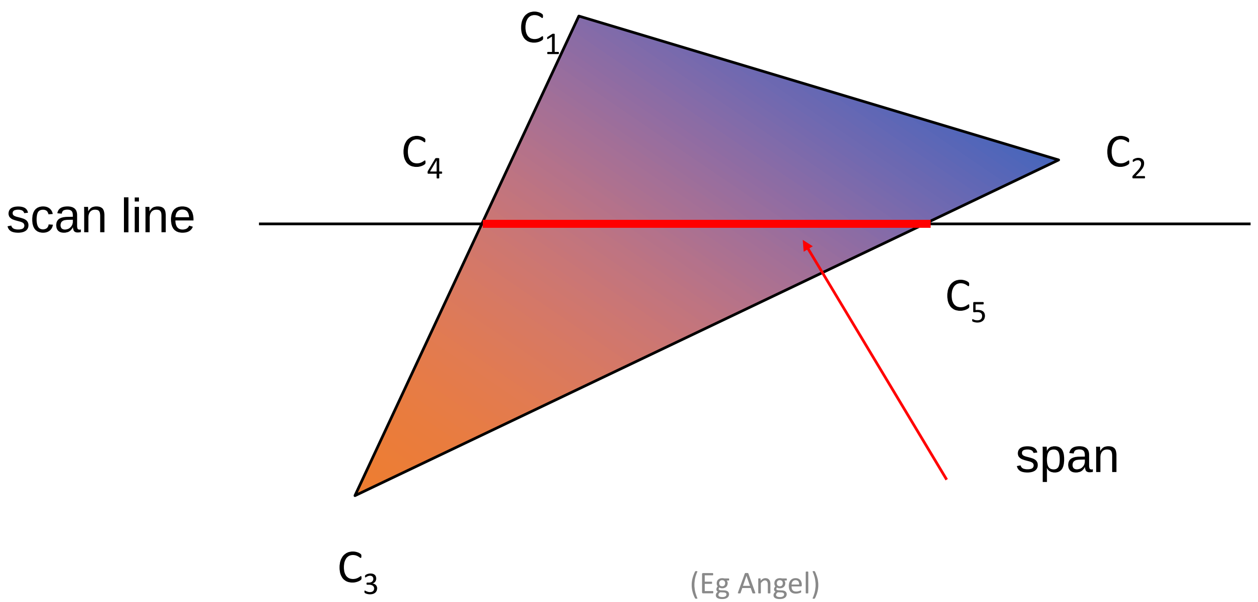
- Barycentric coordinates
- Or just interpolation while filling spans



(2013 Steve Marschner)

Using Interpolation

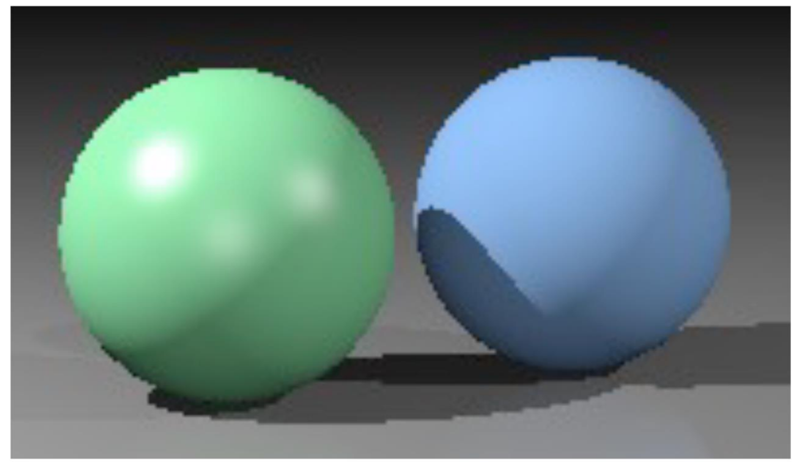
C_1 C_2 C_3 specified by **`glColor`** or by vertex shading
C_4 determined by interpolating between C_1 and C_3
C_5 determined by interpolating between C_2 and C_3
interpolate between C_4 and C_5 along span

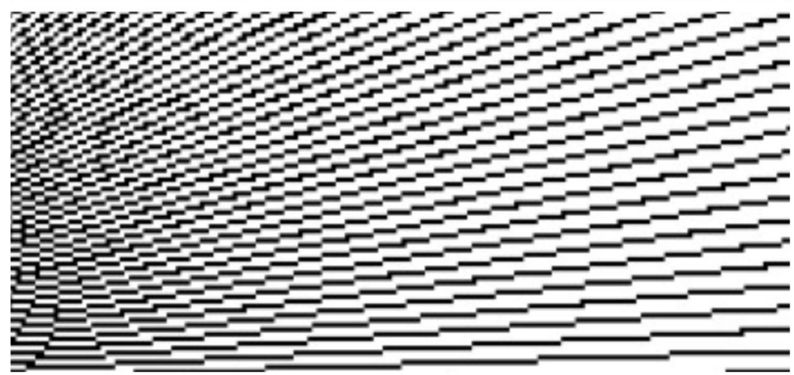


(Eg Angel)

Anti-aliasing

- Why?
- What is aliasing?
  - Discretization artefacts of continuous shapes

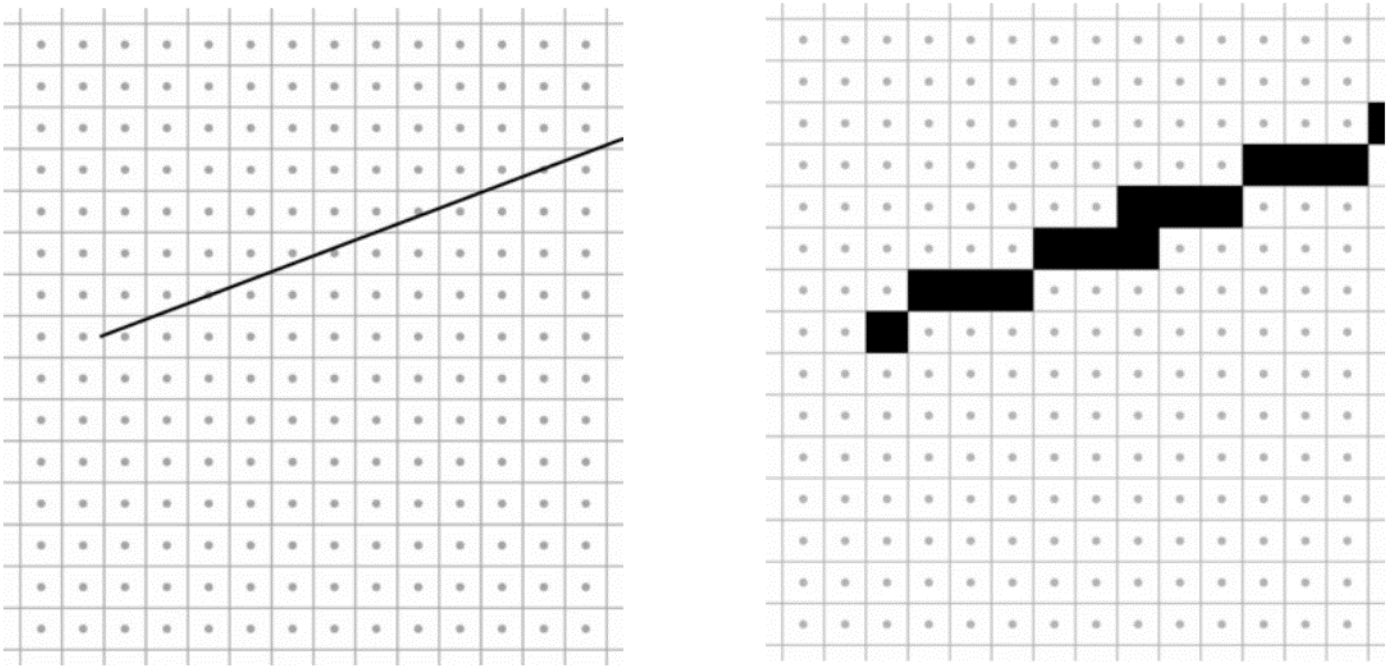




(2013 Steve Marschner)

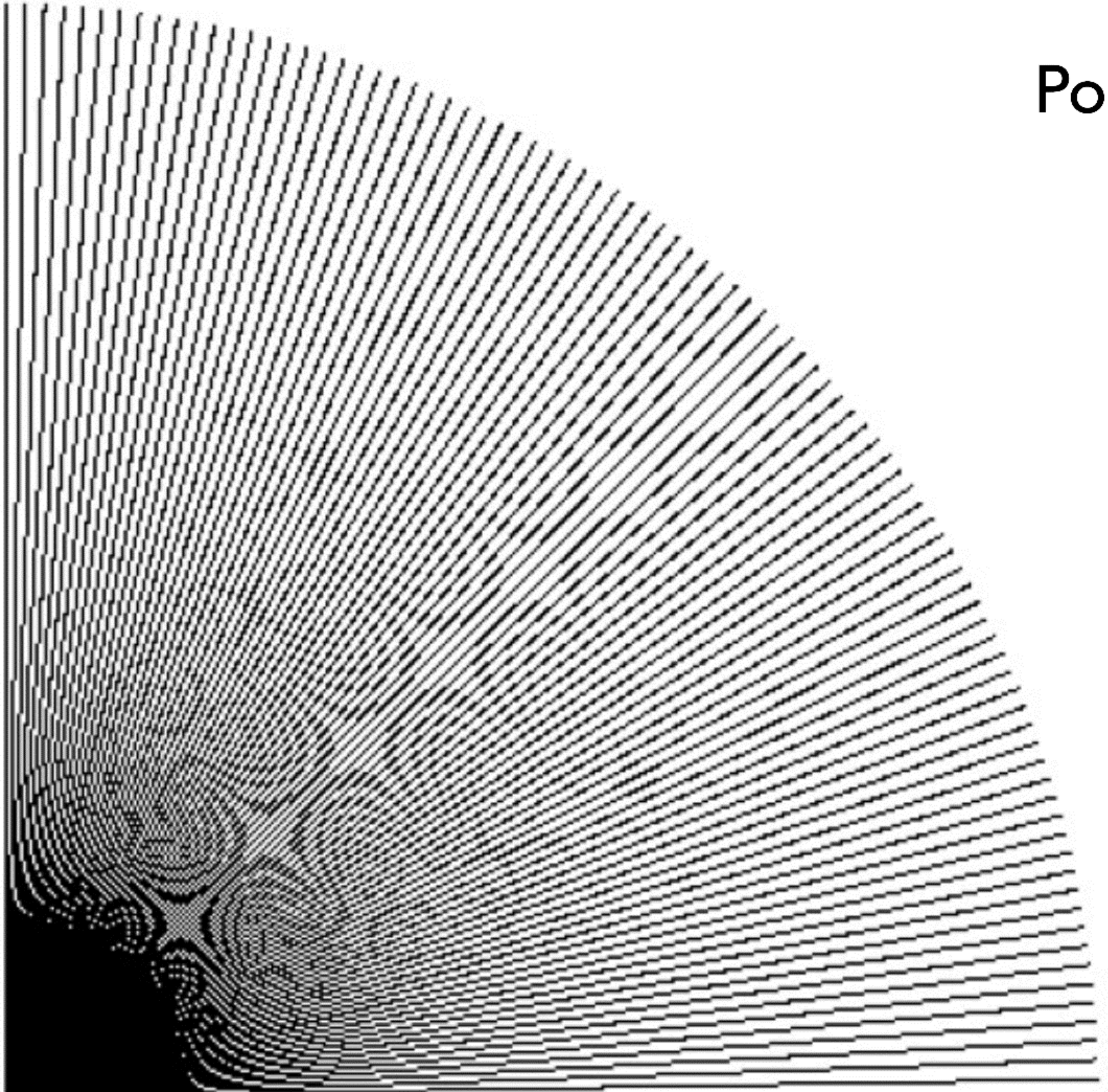
Aliasing

- Rasterizing lines
  - We sample the line by just selecting or ignoring pixels



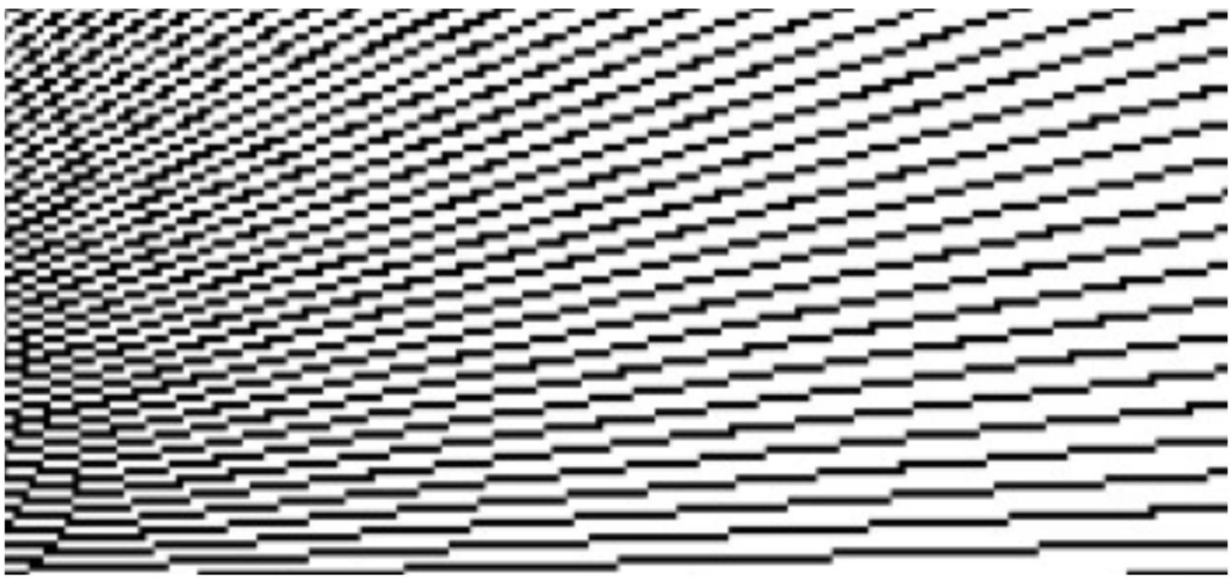
(2013 Steve Marschner)

Point sampling in action



Aliasing

- The lines have stair steps and variations in width
- Sharp edges of line contain high frequencies

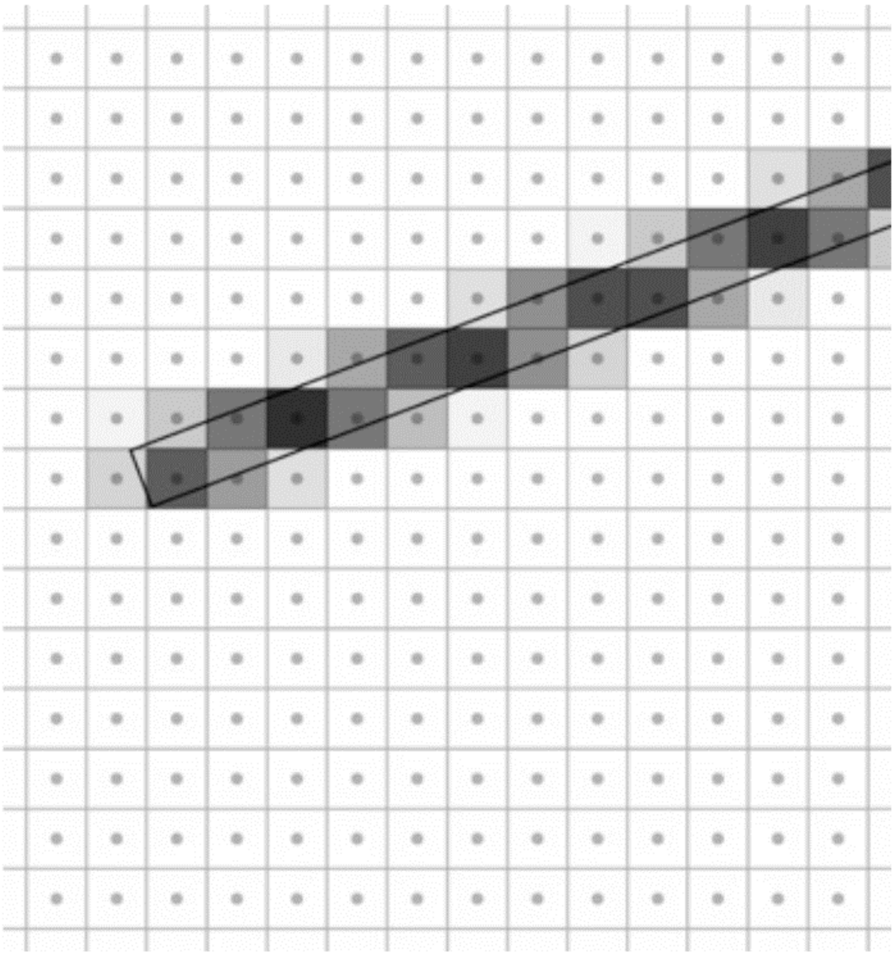


Antialiasing

- On bitmap devices this can not be avoid
  - We need high resolution
  - 600+ dpi in printers
- On continuous-tone devices they can be alleviate

Antialiasing

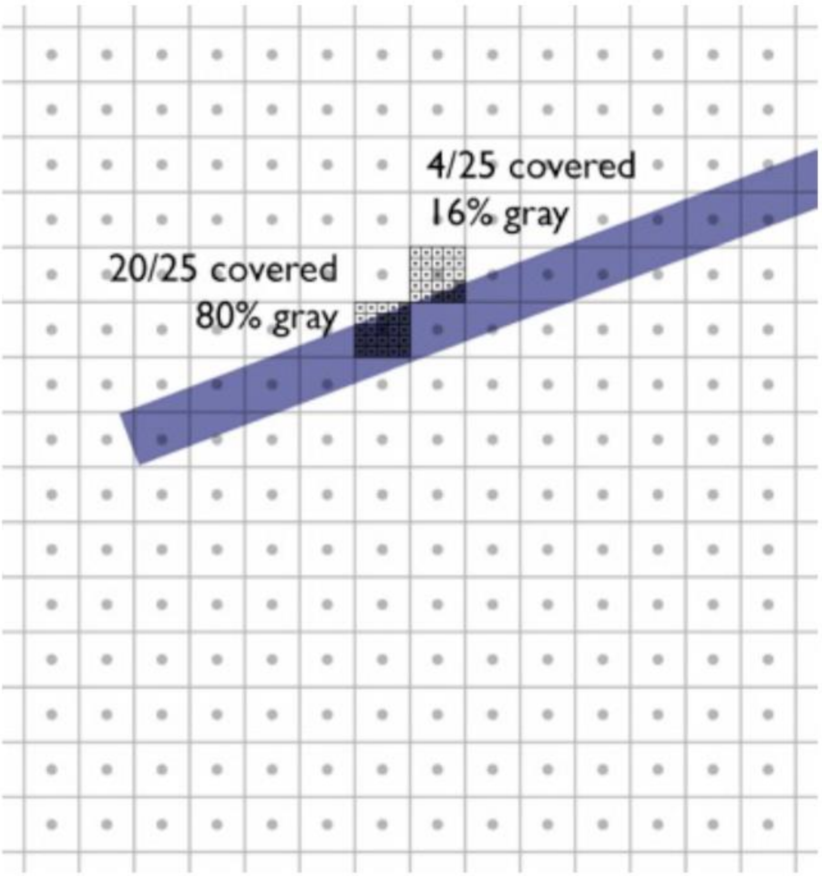
- Basic idea: replace “is the image black at the pixel center?” with “how much is pixel covered by black?”
- Replace yes/no question with quantitative question.



(2013 Steve Marschner)

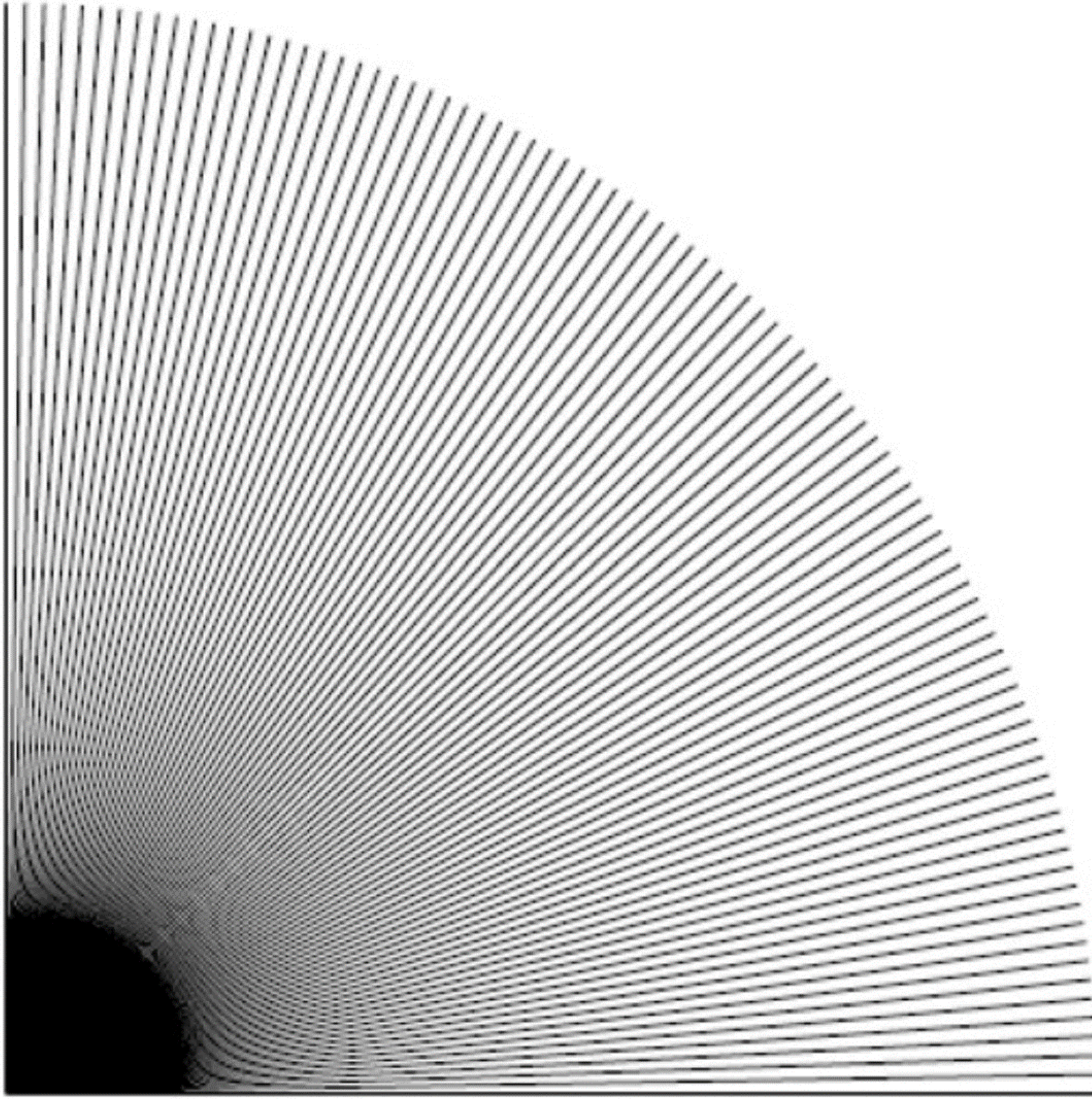
Box filtering

- Compute coverage fraction by counting subpixels
- Simple, accurate
- Slow
- Unweighted filtering



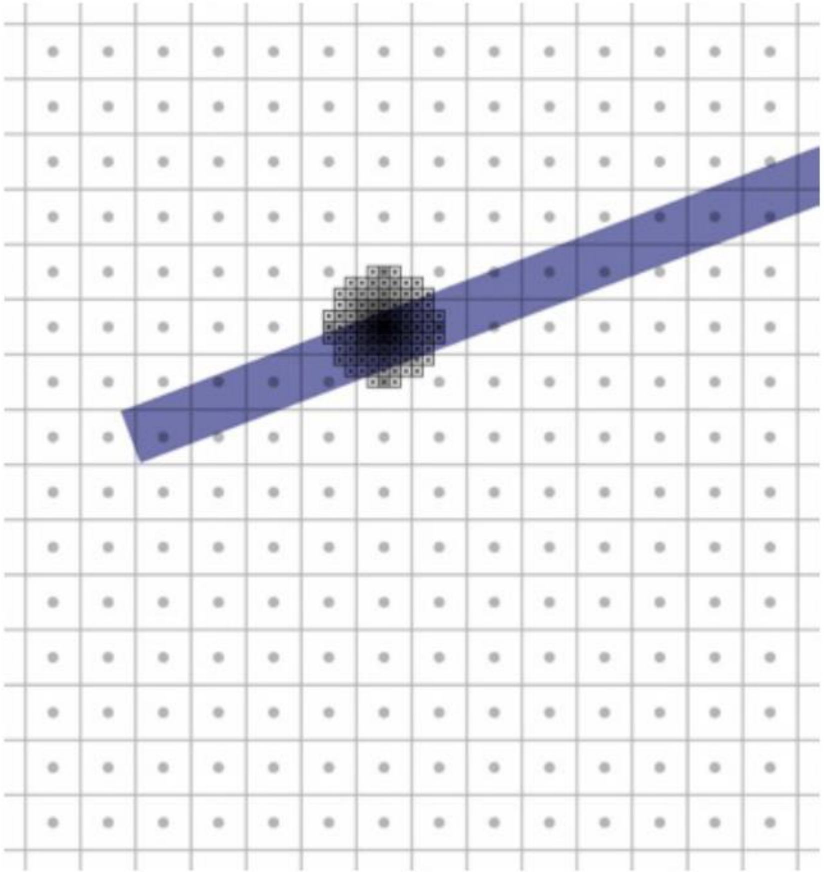
(2013 Steve Marschner)

Box filtering in action



Weighted filtering

- Compute filtering integral by summing filter values for covered subpixels
- Simple, accurate
- But really slow

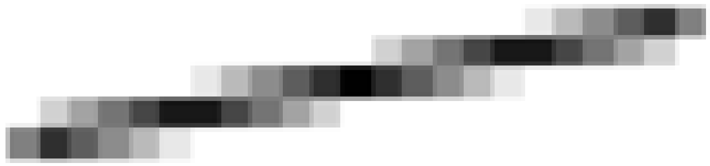


(2013 Steve Marschner)

Gaussian filtering in action

Xiaolin Wu's line algorithm

- Bresenham's algorithm draws lines extremely quickly, but it does not perform anti-aliasing
- The algorithm consists of drawing pairs of pixels straddling the line, each coloured according to its distance from the line.



References

- Ed Angel, CS/EECE 433 Computer Graphics, University of New Mexico
- Cutler and Durand, MIT EECS 6.837
- Steve Marschner, CS4620/5620 Computer Graphics, Cornell
- Tom Thorne, COMPUTER GRAPHICS, The University of Edinburgh
- Elif Tosun, Computer Graphics, The University of New York

- Questions?