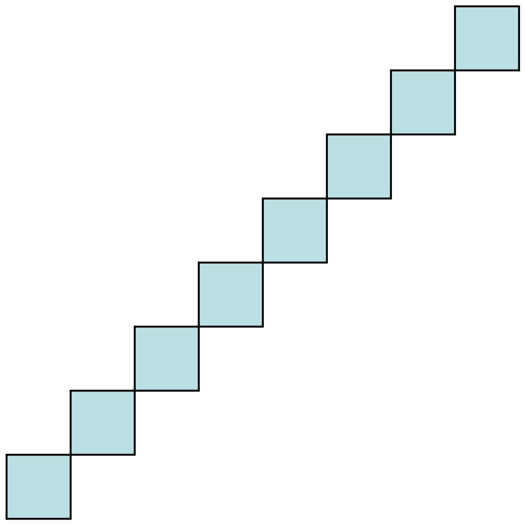
Assignment

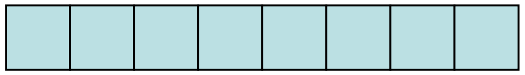
A few details regarding the anti-aliasing and polygon filling have been added to the on-line version of the assignment. A few typos have been fixed.

Algorithms for the meat of the assignment will be discussed in this lecture

Line drawing--simple line (Bresenham) brightness issues (Addendum to previous lecture’s slides)

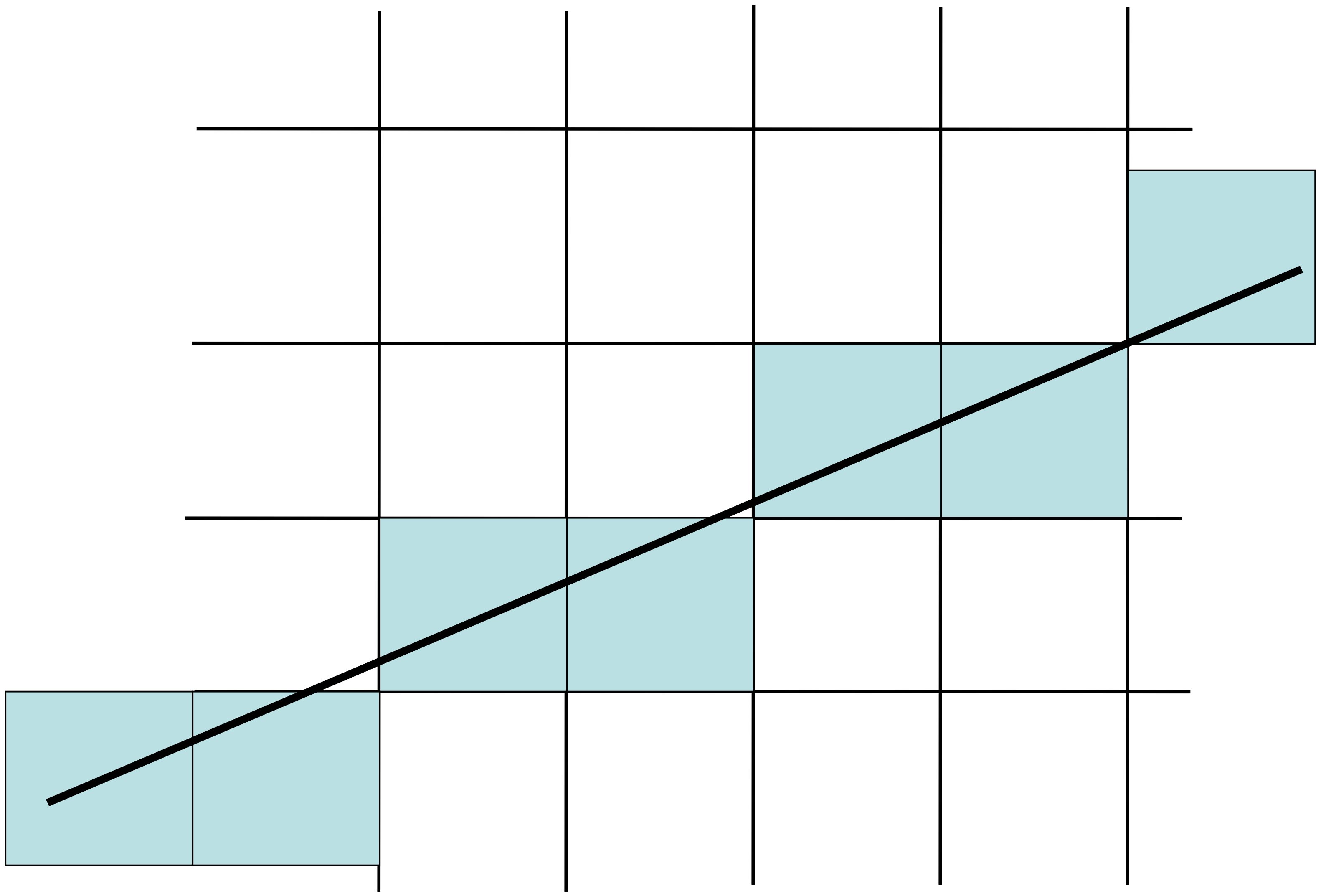


8 pixels per 8*sqrt(2) length



8 pixels for 8 length
(Brighter)

Line drawing--aliasing (review from last lecture)

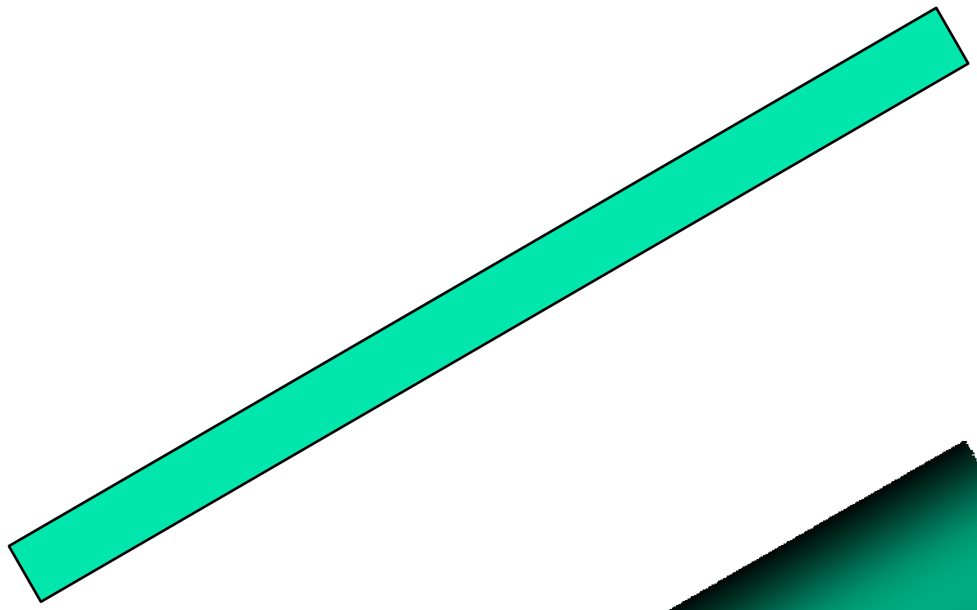


Aliasing (review)

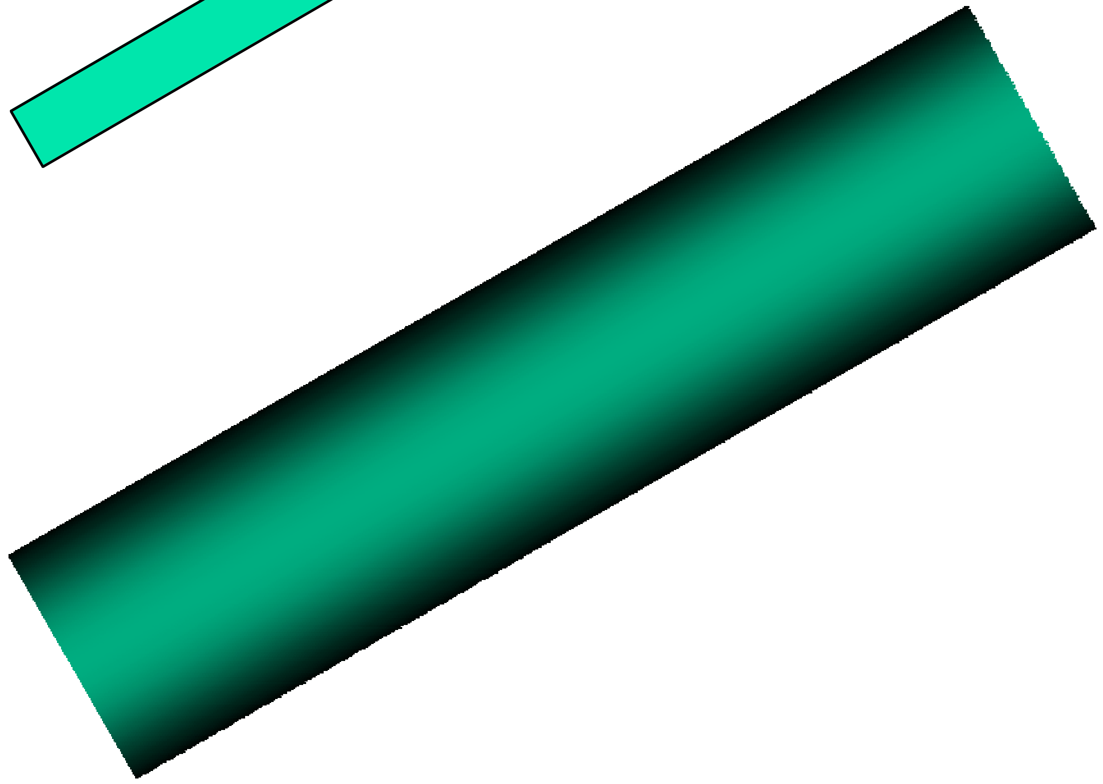
- Sampling problem--we are using discrete binary squares to represent perfect mathematical entities
- General approach to reducing aliasing is to exploit ability to draw levels of gray between black and white.
- Example--give the line some width and make the brightness proportional to the area that the pixel shares with line.
- We will take a sampling approach.

Aliasing via sampling

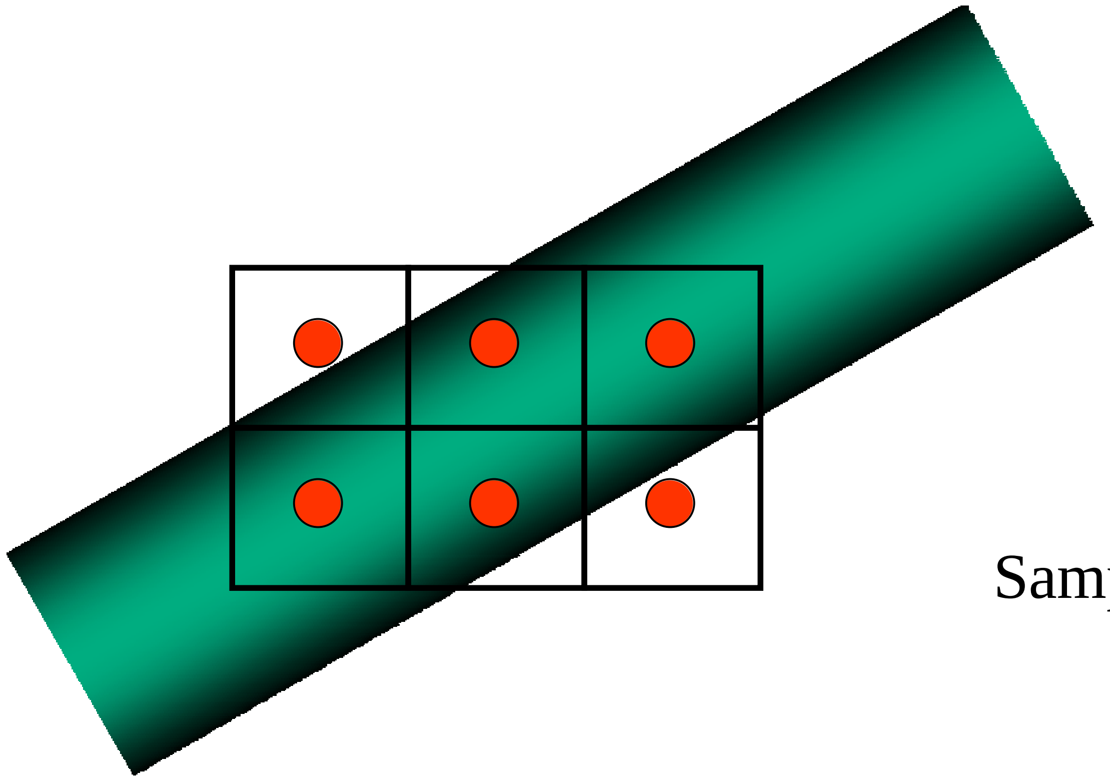
- Smooth (convolve) the object to be drawn with a filter for each pixel
- This blurs the object, widens the area it occupies
- Now we “sample” the blurred image--i.e., report the value of the blurred function at the (x,y) of interest, and then fill the square with that brightness.



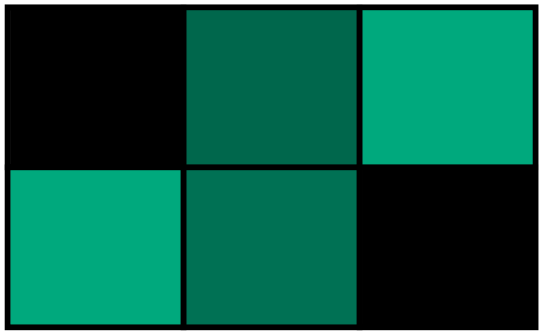
Line with width



Blurred



Sample



Paint with
sample value

Aliasing via sampling

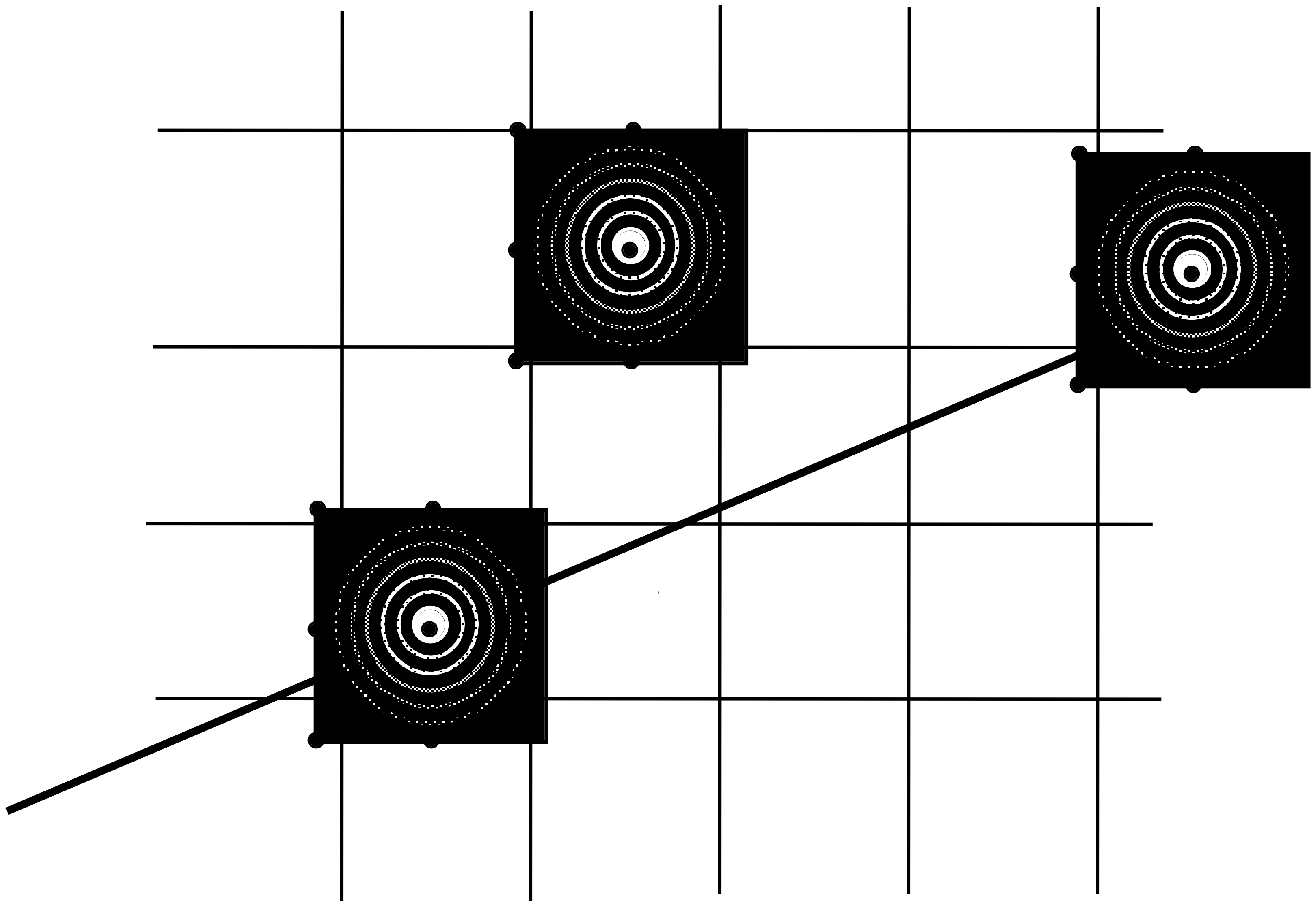
- Ideal filter is usually Gaussian
- Easier and much faster to approximate Gaussian with a cone
- See Figure 3.38, page 124.
- (Trying this out is part of first assignment)

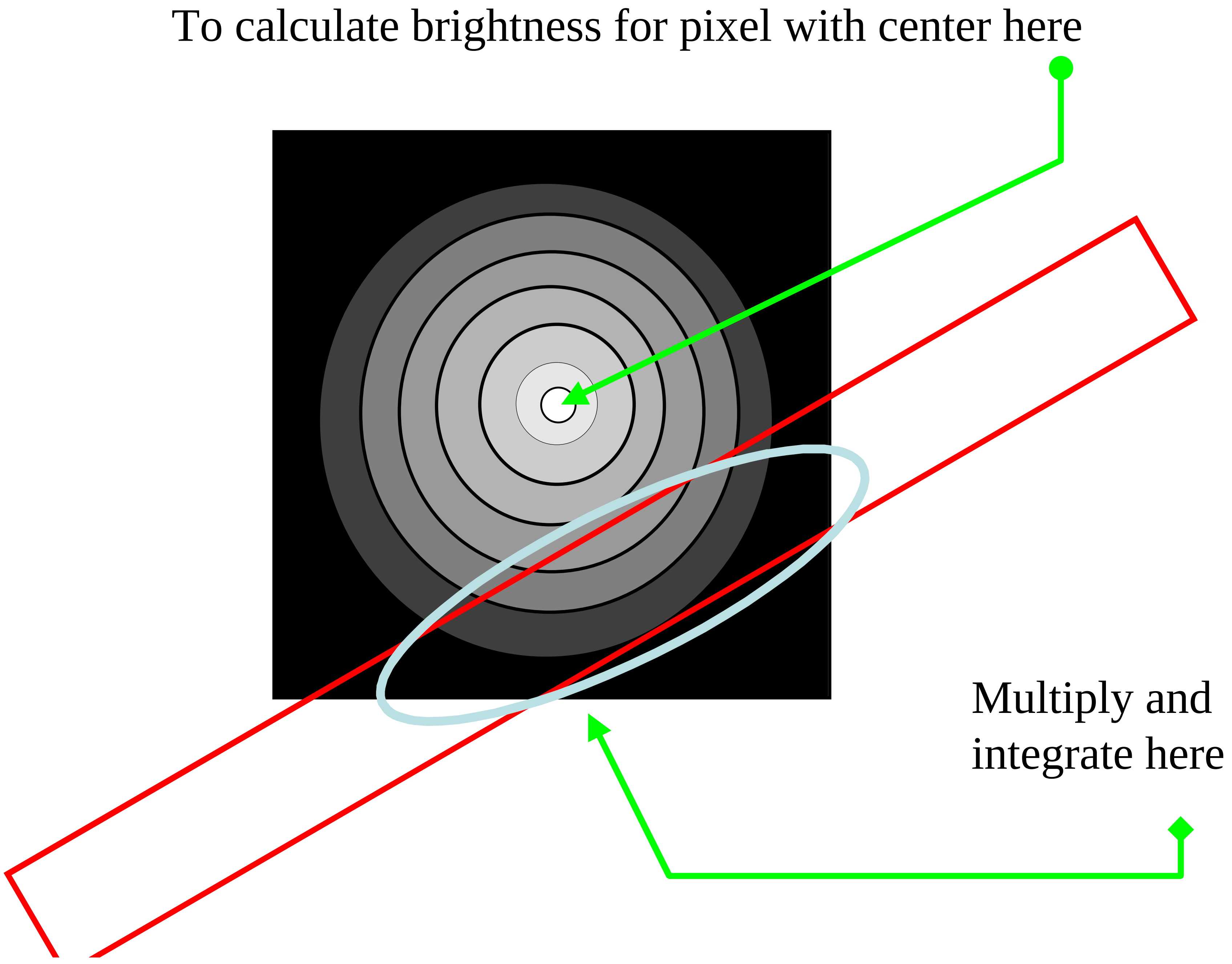
Anti-aliasing via sampling

Technically we “convolve” the function representing the primitive g(x,y) with the filter, h (ζ, η)

$$g \otimes h = \iint g(x-\xi, y-\eta)h(\xi,\eta)d\xi d\eta$$

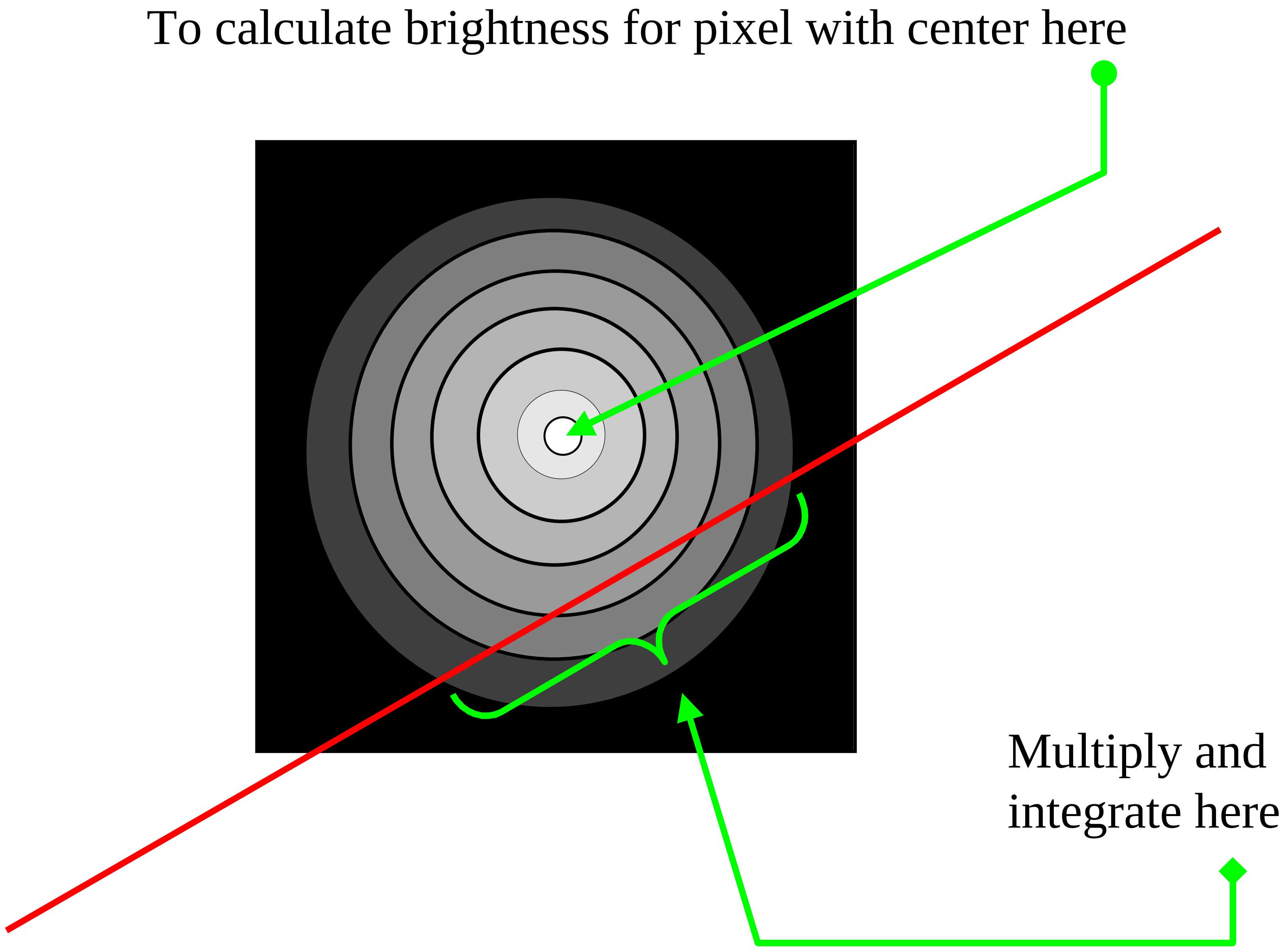
Line drawing--anti-aliasing--a filter at each point (3 shown)



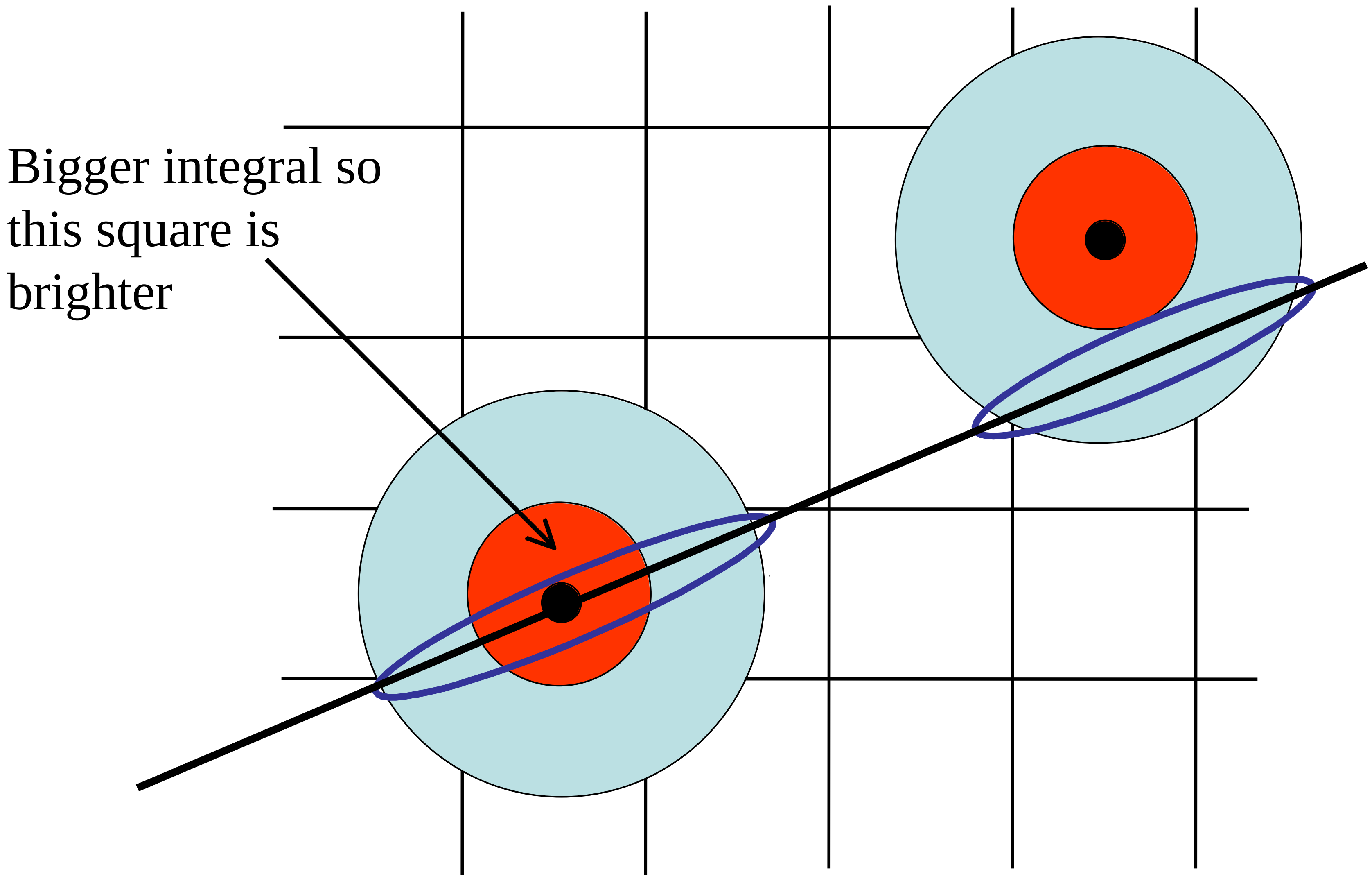

To calculate brightness for pixel with center here
Multiply and integrate here

Line with no width

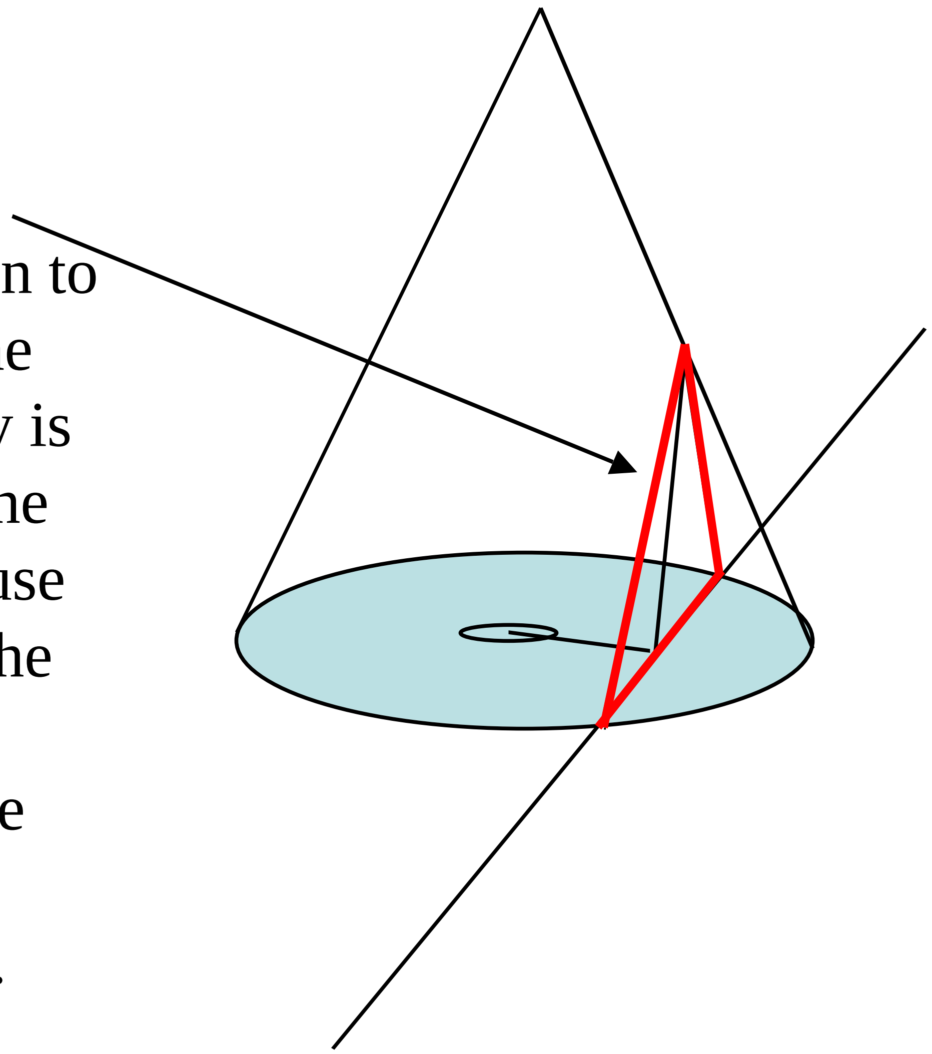
- If line has no width, then it is a line of “delta” functions.
- Algorithmically simpler: Just integrate intersection of blurring function and line in 1D.
- Normalization--simply ensure that if the line goes through the filter center, that the pixel gets the full color of the line.


To calculate brightness for pixel with center here
Multiply and integrate here

Line with cone example

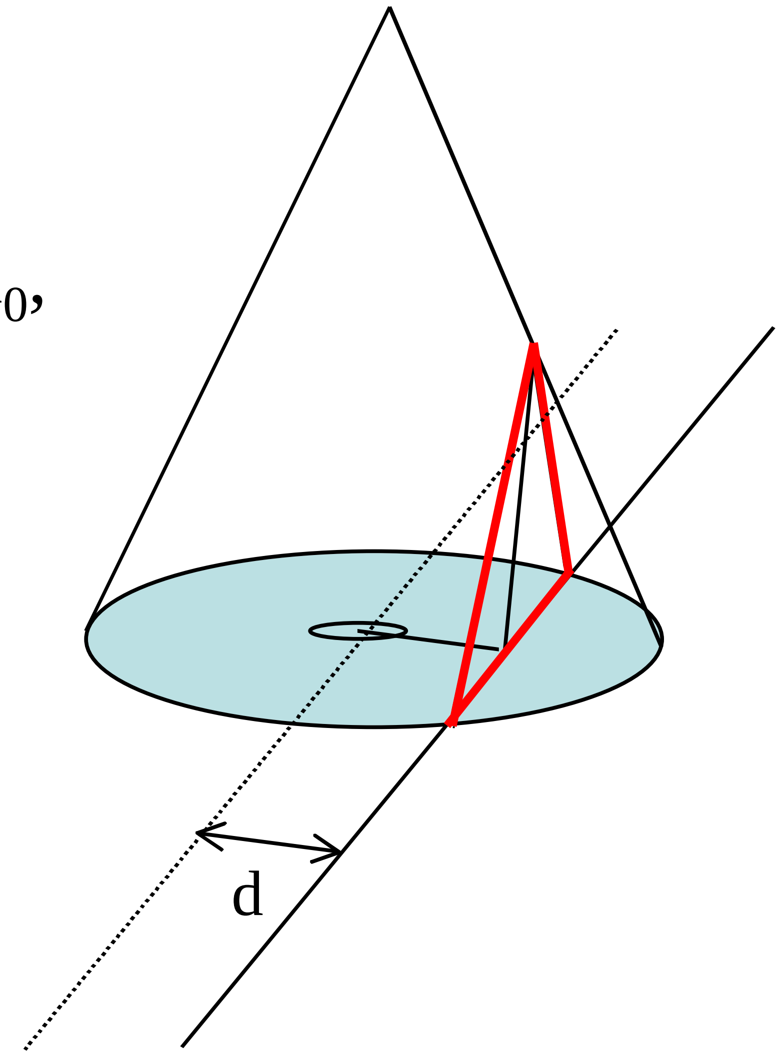


Linear approximation to boundary. The real boundary is curved. For the assignment, use the line. IE, the weighting function is the area triangle shown in red.



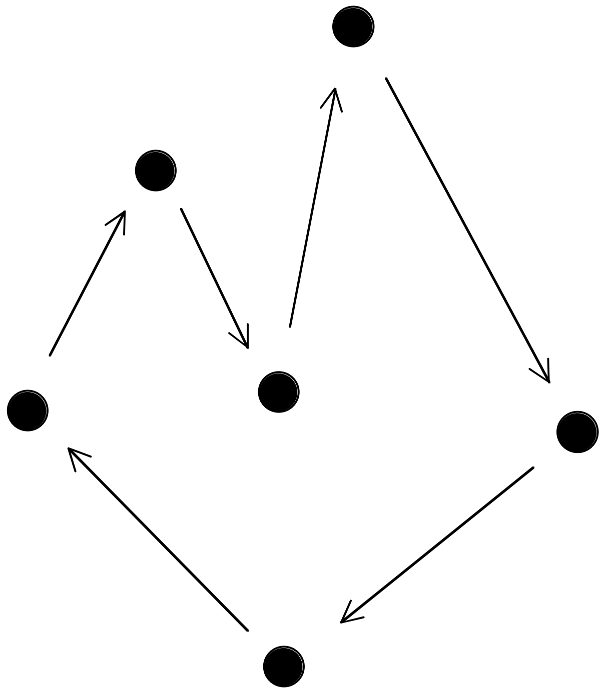
Hints for the assignment

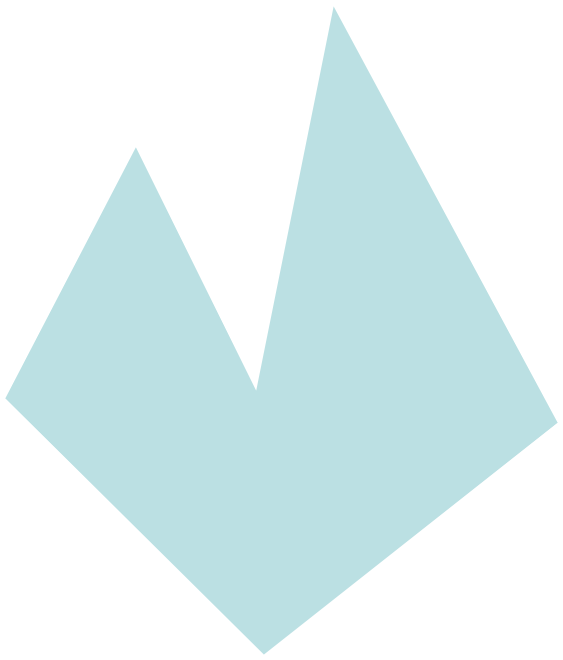
- Derive an expression which takes (m,b) for the line y=mx+b, and the grid point (x_0, y_0), and computes the weight for that pixel based on the distance, d, to the line. You will first need to figure out an expression for d.
- If (x_0, y_0) is on y=mx+b, the weight should be 1.
- The radius and the height of the cone is 1 "block pixel".



Scan converting polygons

(Text: Section 3.5 (see 3.4 also))


Have

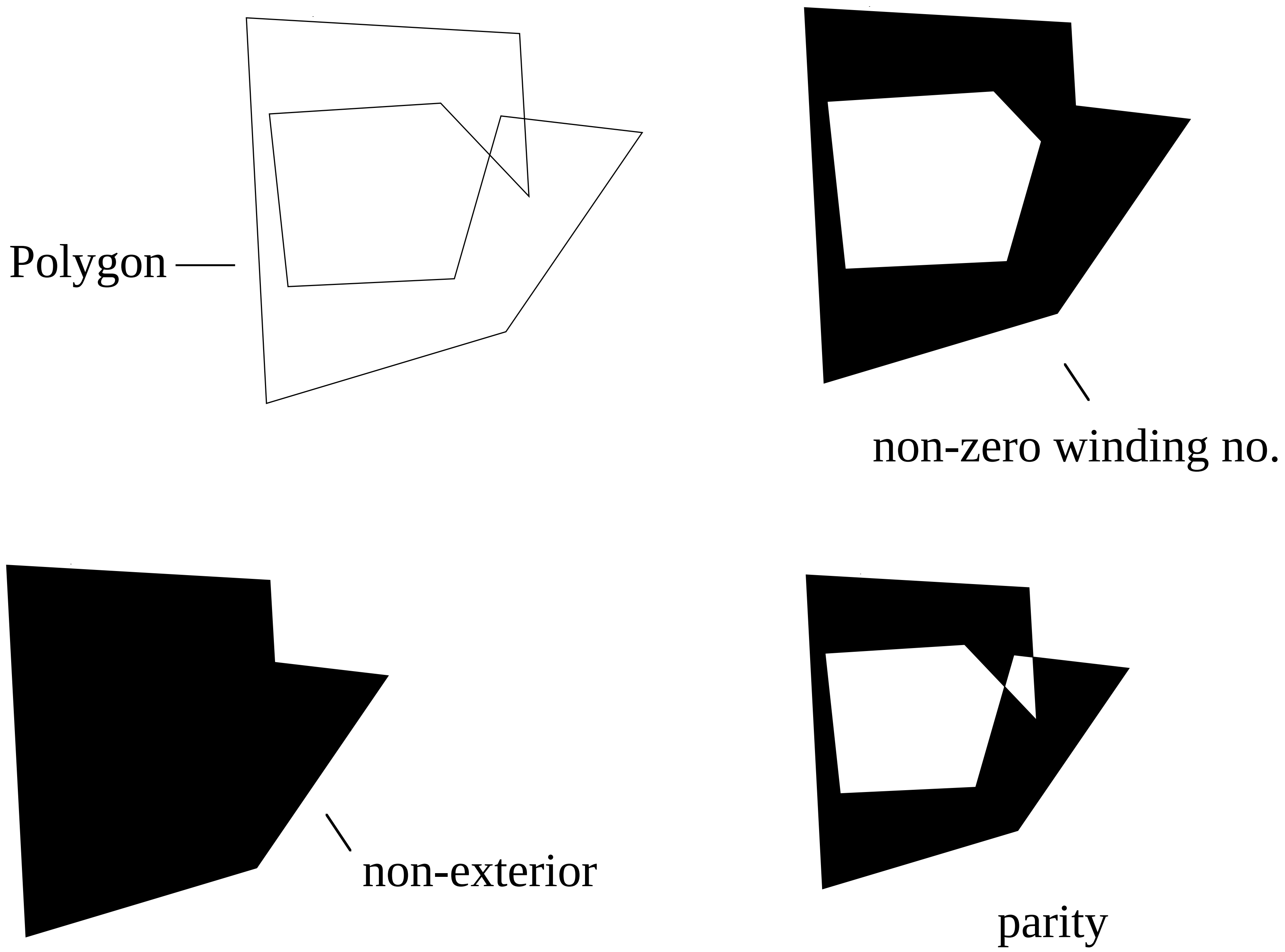

Need

Filling polygons

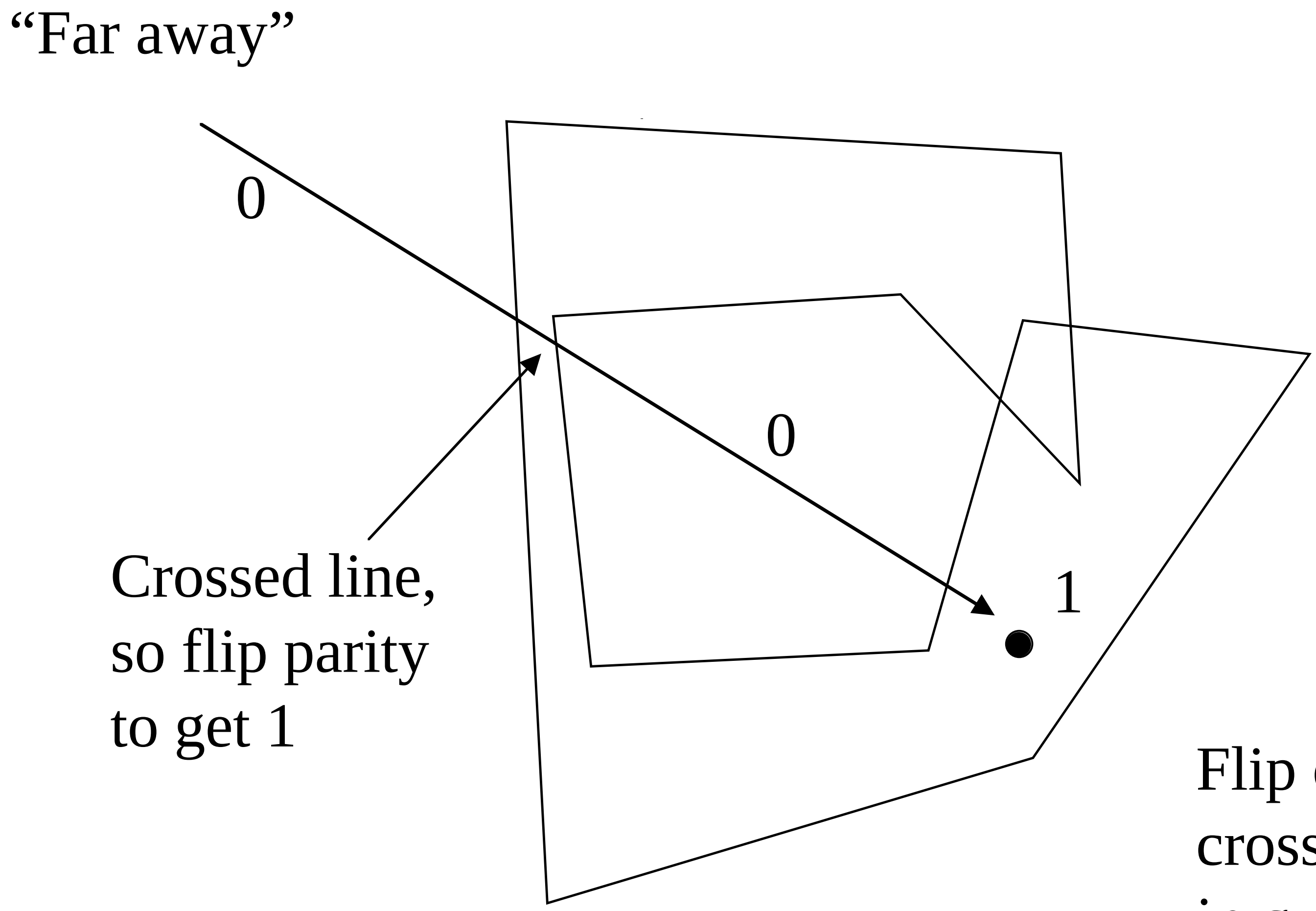
- Polygons are defined by a list of edges - each is a pair of vertices (order counts)
- Assume that each vertex is an integer vertex, and polygon lies within frame buffer
- Need to define what is inside and what is outside

What is inside?

- Easy for simple polygons - no self intersections
- For general polygons, three rules are used:
  - non-exterior rule
  - non-zero winding number rule
  - parity rule (most common--this is the one we will generally used)


Polygon
non-zero winding no.
non-exterior
parity

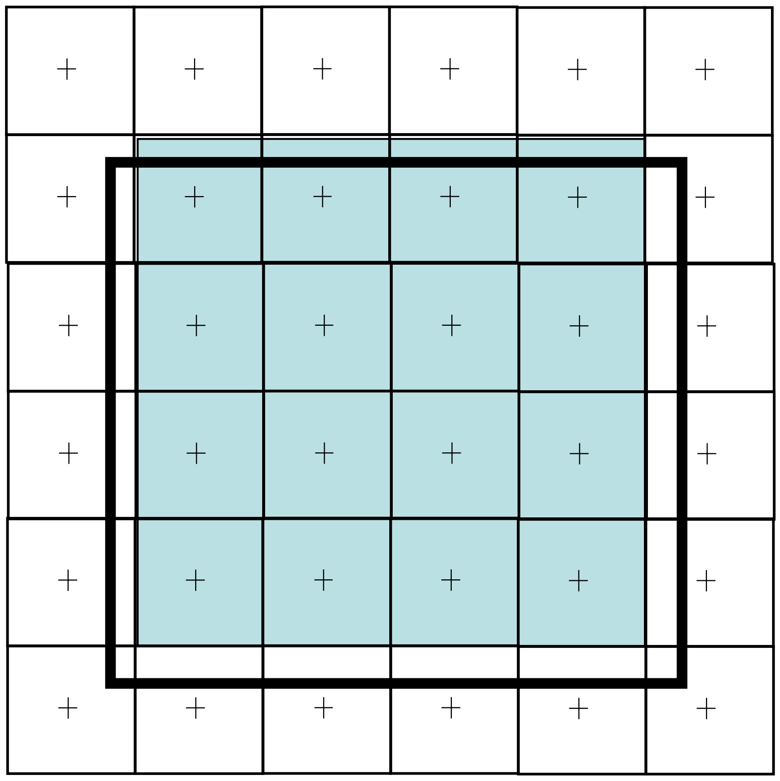
Parity rule--details



Flip once for each line crossing. Value at point in question is 1, so point is inside.

Which pixel is inside?

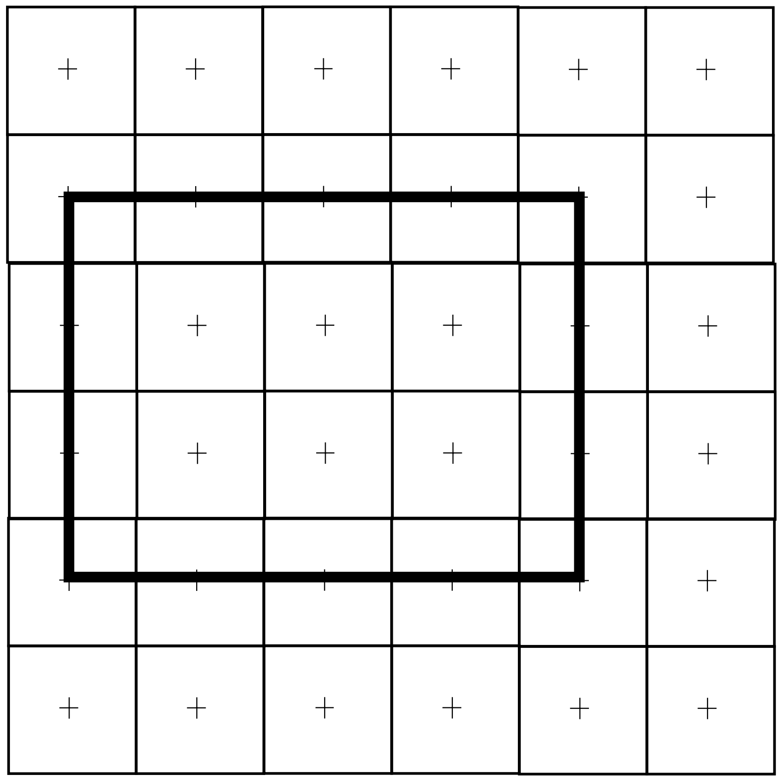
- Each pixel is a *sample*, at coordinates (x, y).
  - imagine a piece of paper, where coordinates are continuous
  - pixels are samples on a grid of a drawing on this piece of paper.
- If ideal point (corresponding to grid center) is inside, pixel is inside.

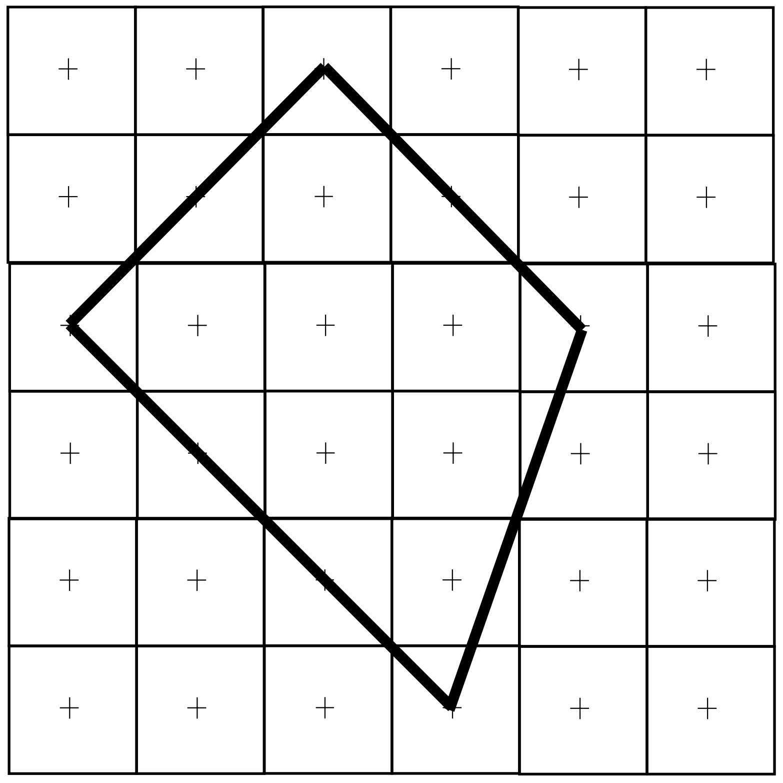


Ambiguous cases

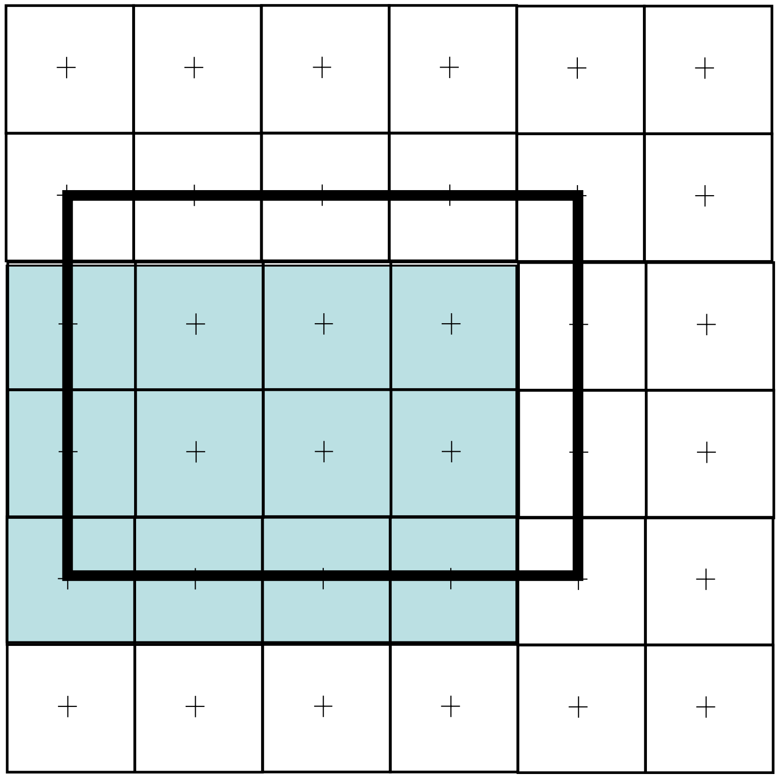
- What if a pixel is exactly on the edge?
- Polygons are usually adjacent to other polygons, so we want a rule which will give the pixel to *one* of the adjacent polygons or the *other* (as much as possible).
- "Draw left and bottom edges"
  - if $(x+\partial, y)$ is in, pixel is in
  - horizontal edge? if $(x+\partial, y+\varepsilon)$ is in, pixel is in
  - what if it's on a vertex?--essentially "left and bottom", but see book for rule which works during scan conversion ("count only y_{min} vertices for parity calculation")

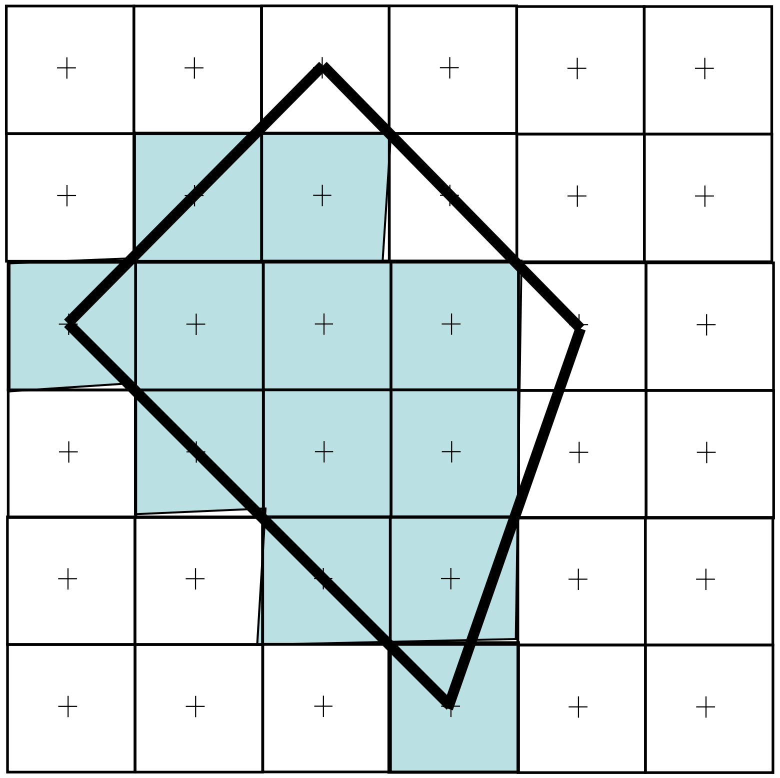
Ambiguous inside cases (?)



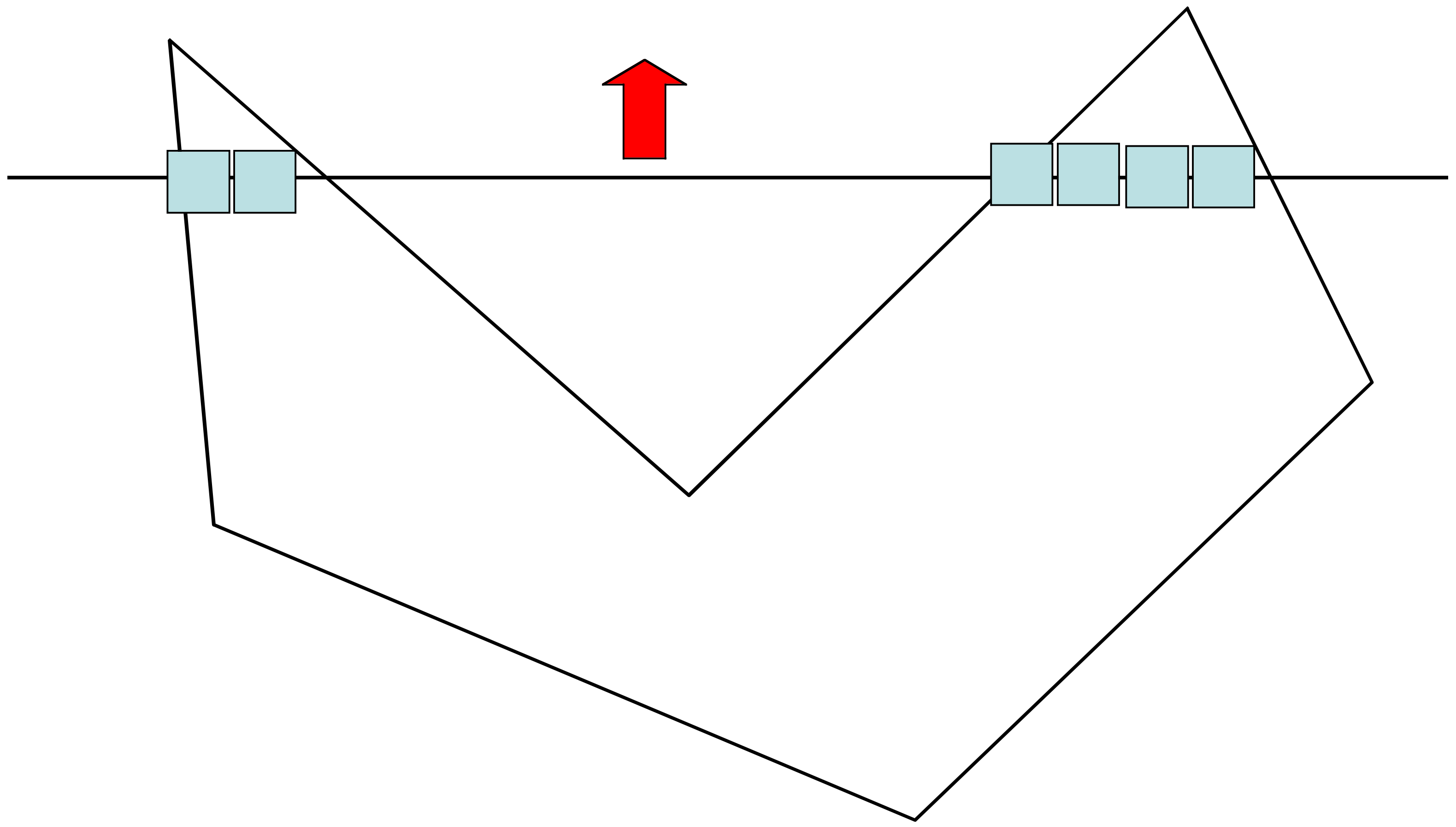


Ambiguous inside cases (answer)





Sweep fill

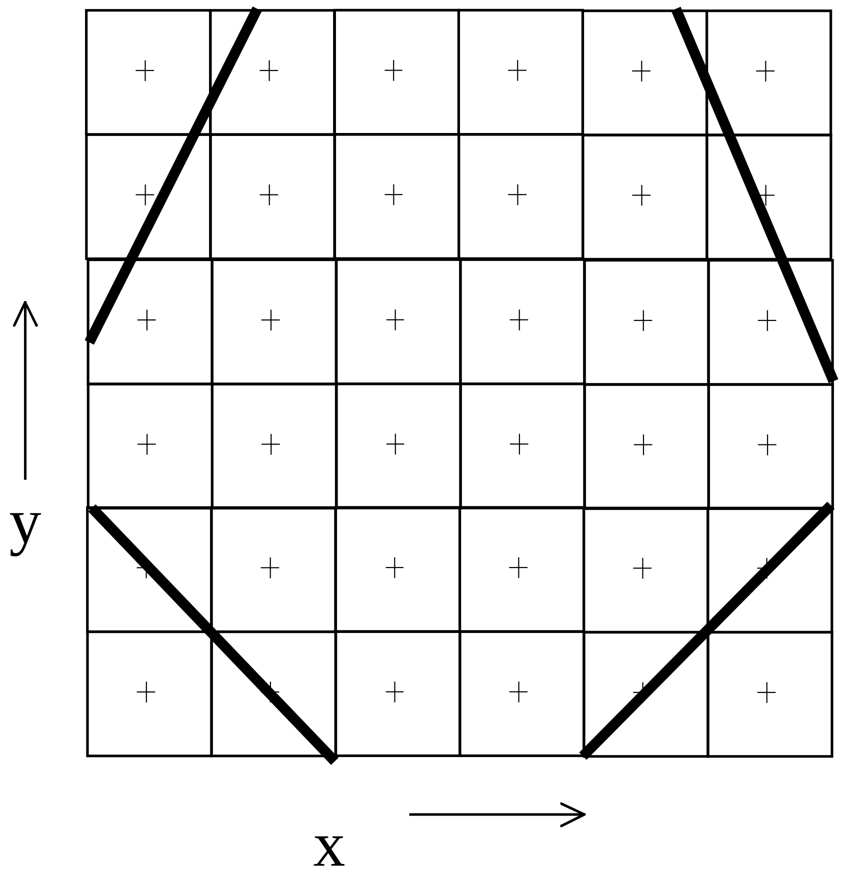


Sweep fill

- Reduces to filling many spans
- Inside/outside parity is relatively straightforward
- Need to compute the spans, then fill
- Need to update the spans for each scan
- Need to implement “inside” rule for ambiguous cases.

Spans

- Process - fill the bottom horizontal span of pixels; move up and keep filling
- have xmin, xmax.
- define:
  - floor(x):= if x integer, x else truncate(x)
  - ceiling(x):= truncate(x)+1
- fill from ceiling(xmin) up to but not including floor(xmax)
- consistent with convention



Algorithm

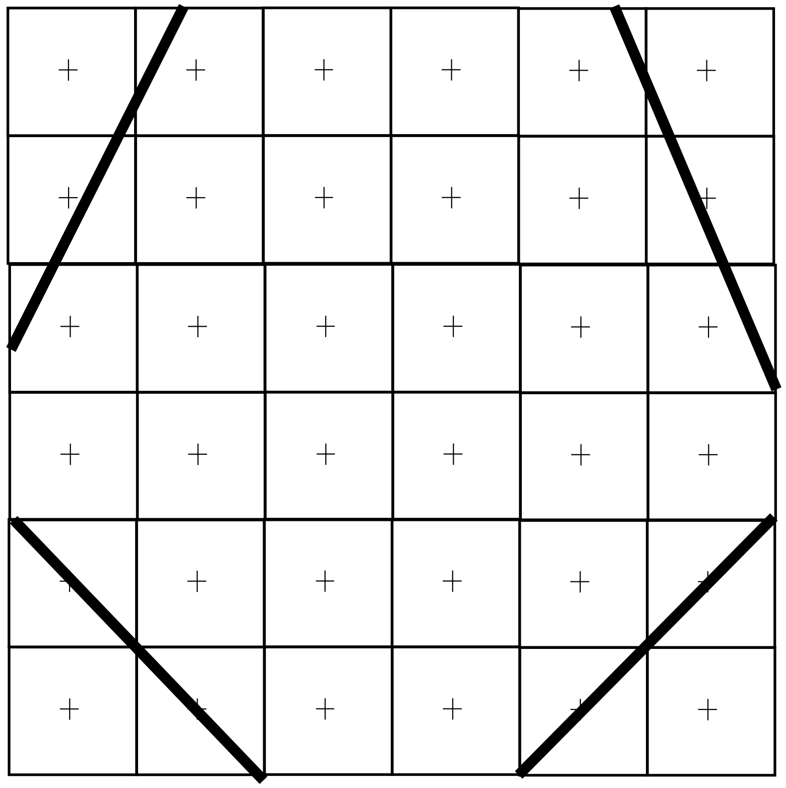
- For each row in the polygon:
  - Throw away irrelevant edges
  - Obtain newly relevant edges
  - Fill spans
  - Update spans
- Issues:
  - what aspects of edges need to be stored?
  - when is an edge relevant/irrelevant?

The next span - 1

- for an edge, have y=mx+c
- hence, if y_n=m x_n +c, then y_{n+1}=y_n+1=m (x_n+1/m)+c
- hence, *if there is no change in the edges*, have:

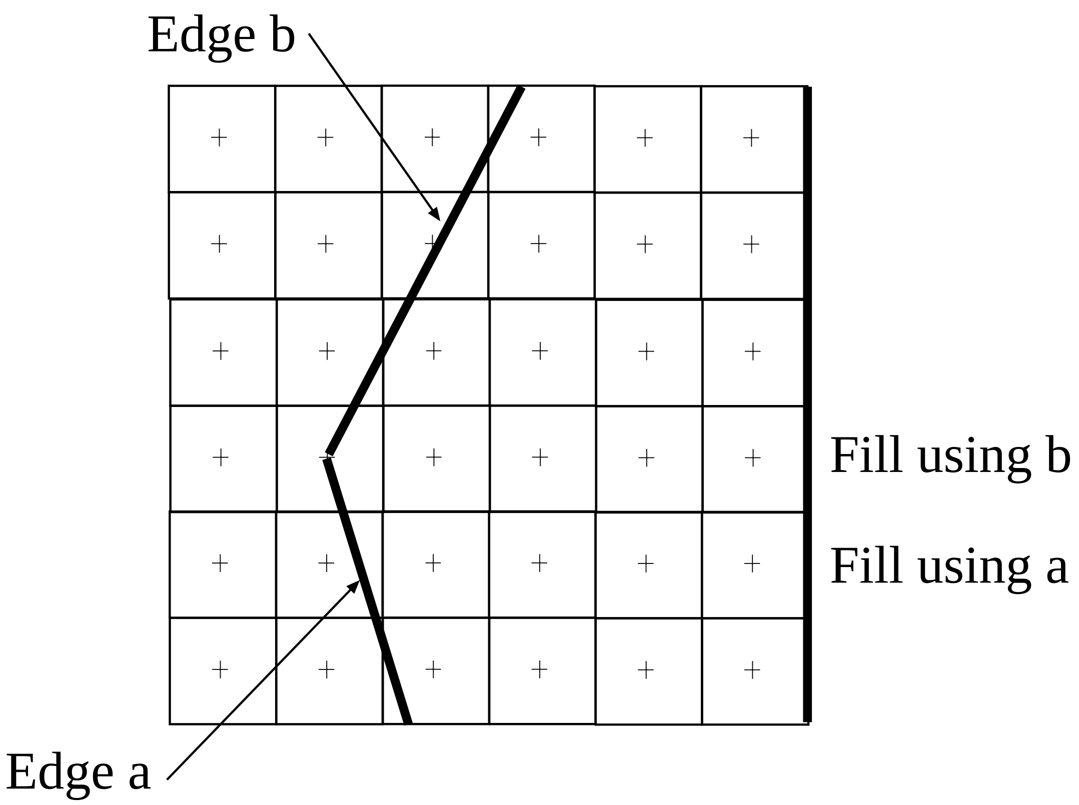
  xmax->xmax+(1/m)(xmax)

  xmin->xmin+(1/m)(xmin)



The next span - 2

- Horizontal edges are irrelevant
- Edge is irrelevant - when y>=ymax of edge (note appeal to convention)
- Similarly, edge is relevant when y>=ymin of edge


Edge b
Fill using b
Fill using a
Edge a