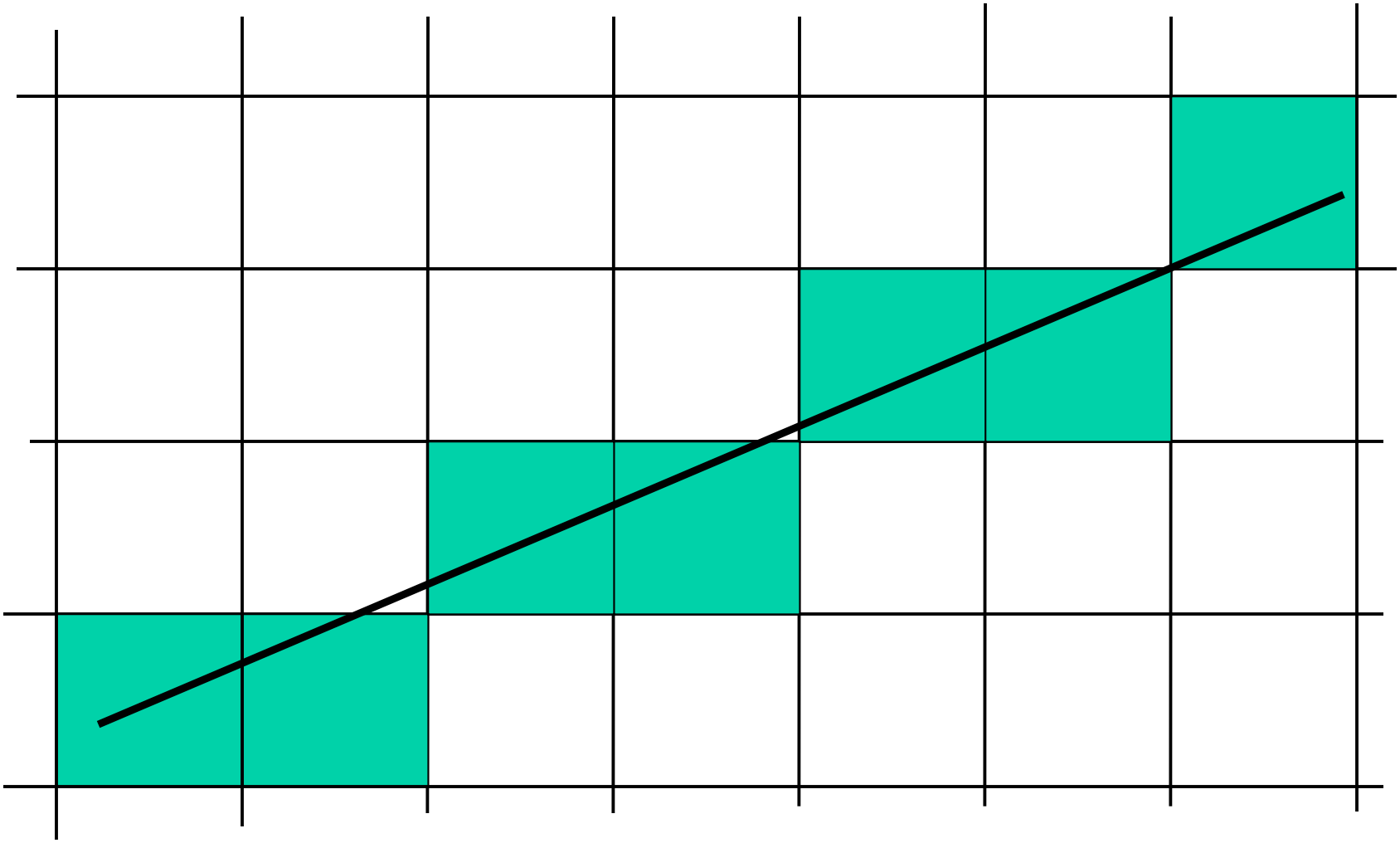
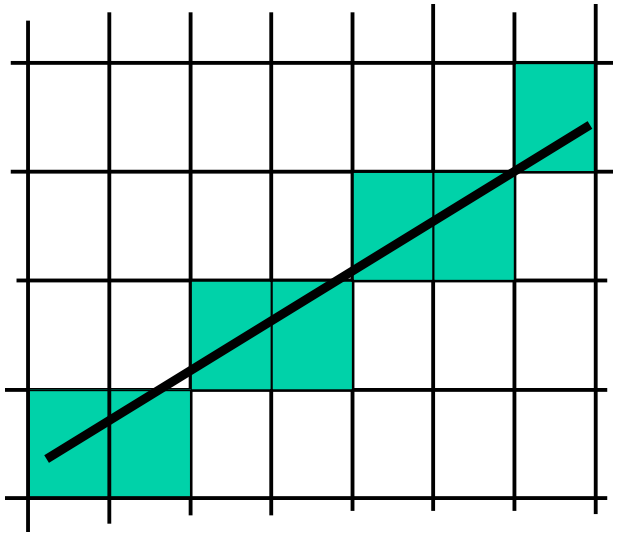
Line drawing--aliasing

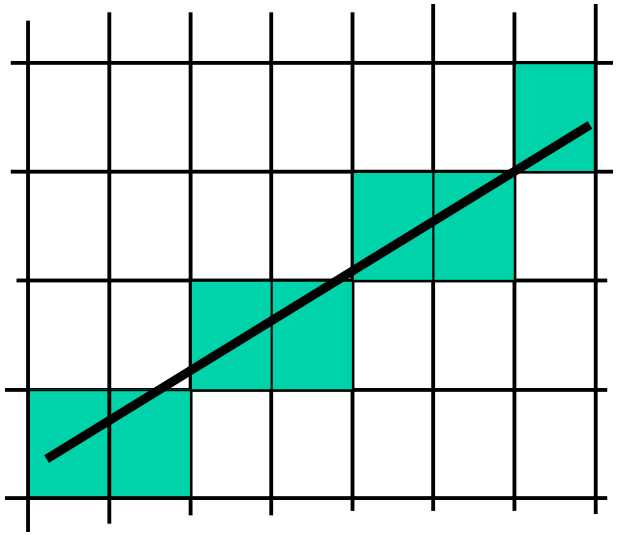


Aliasing

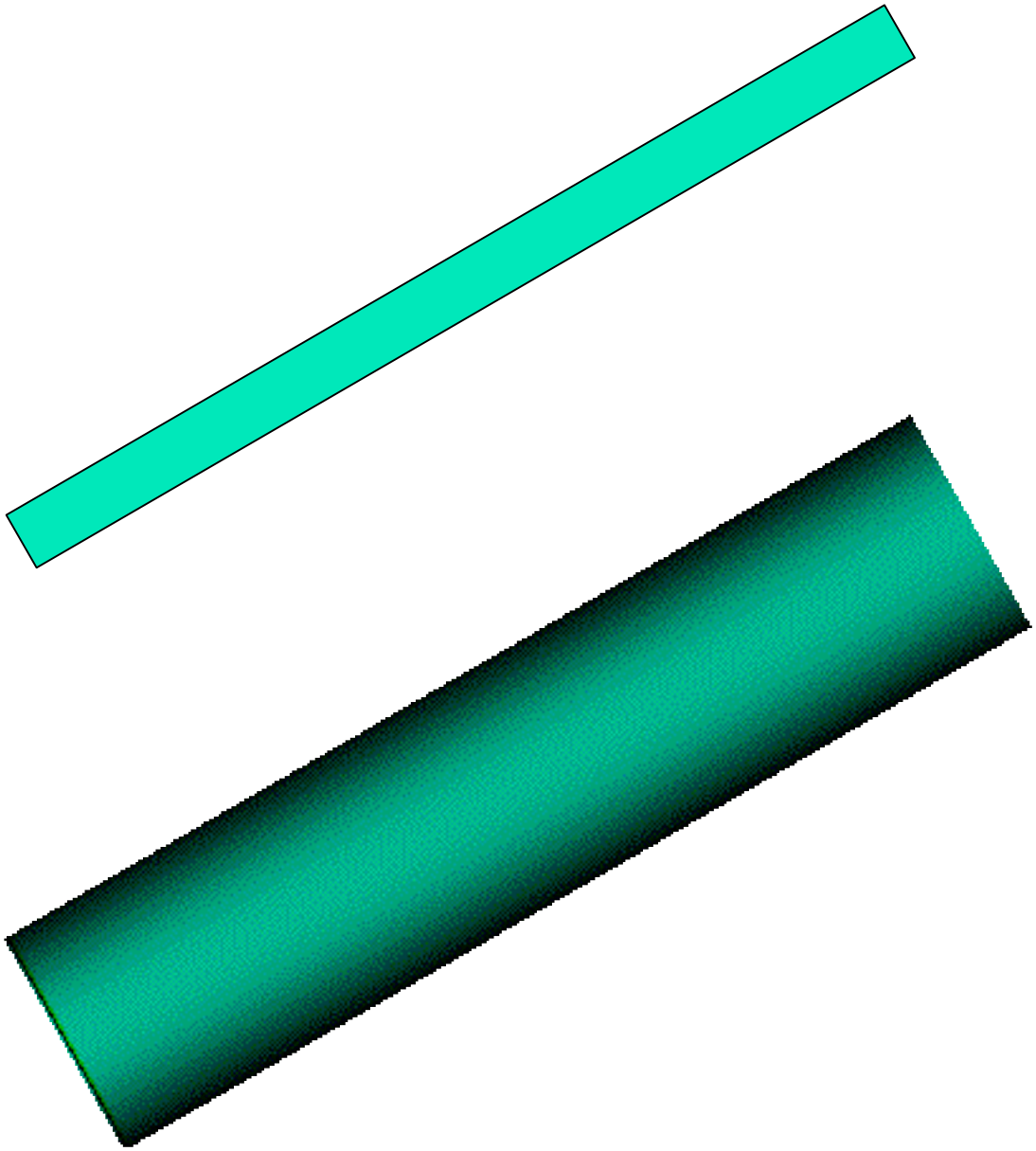


- Sampling problem--we are using discrete binary squares to represent perfect mathematical entities
- General approach to reducing aliasing is to exploit ability to draw levels of gray between black and white.
- Example--give the line some width; brightness is proportional to area that pixel shares with line
- Sampling theory gives a principled way to do this.

Aliasing via sampling

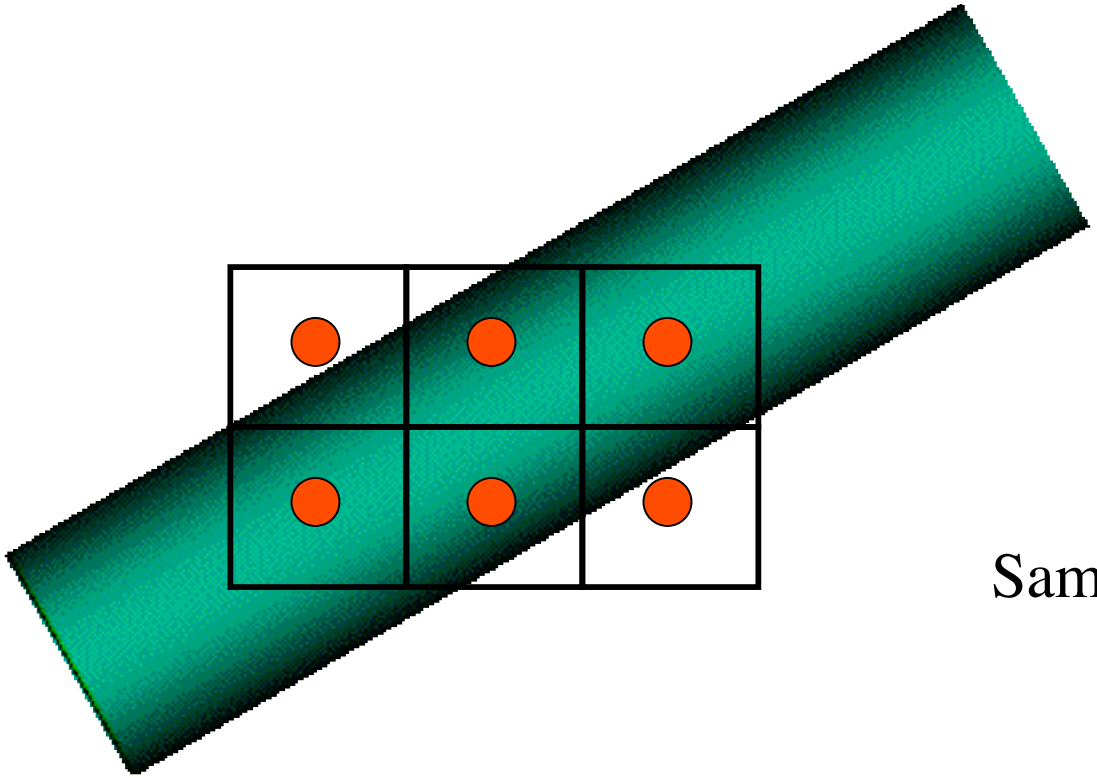


- Smooth (convolve) the object to be drawn with a filter for each pixel
- This blurs the object, widens the area it occupies
- Now we “sample” the blurred image--i.e., report the value of the blurred function at the (x,y) of interest, and then fill the square with that brightness.

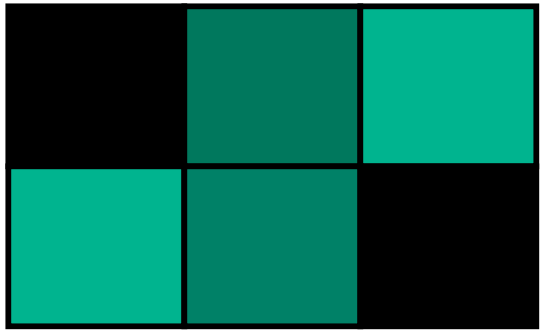


Line with width

Blurred



Sample



Paint with
sample value

Aliasing via sampling

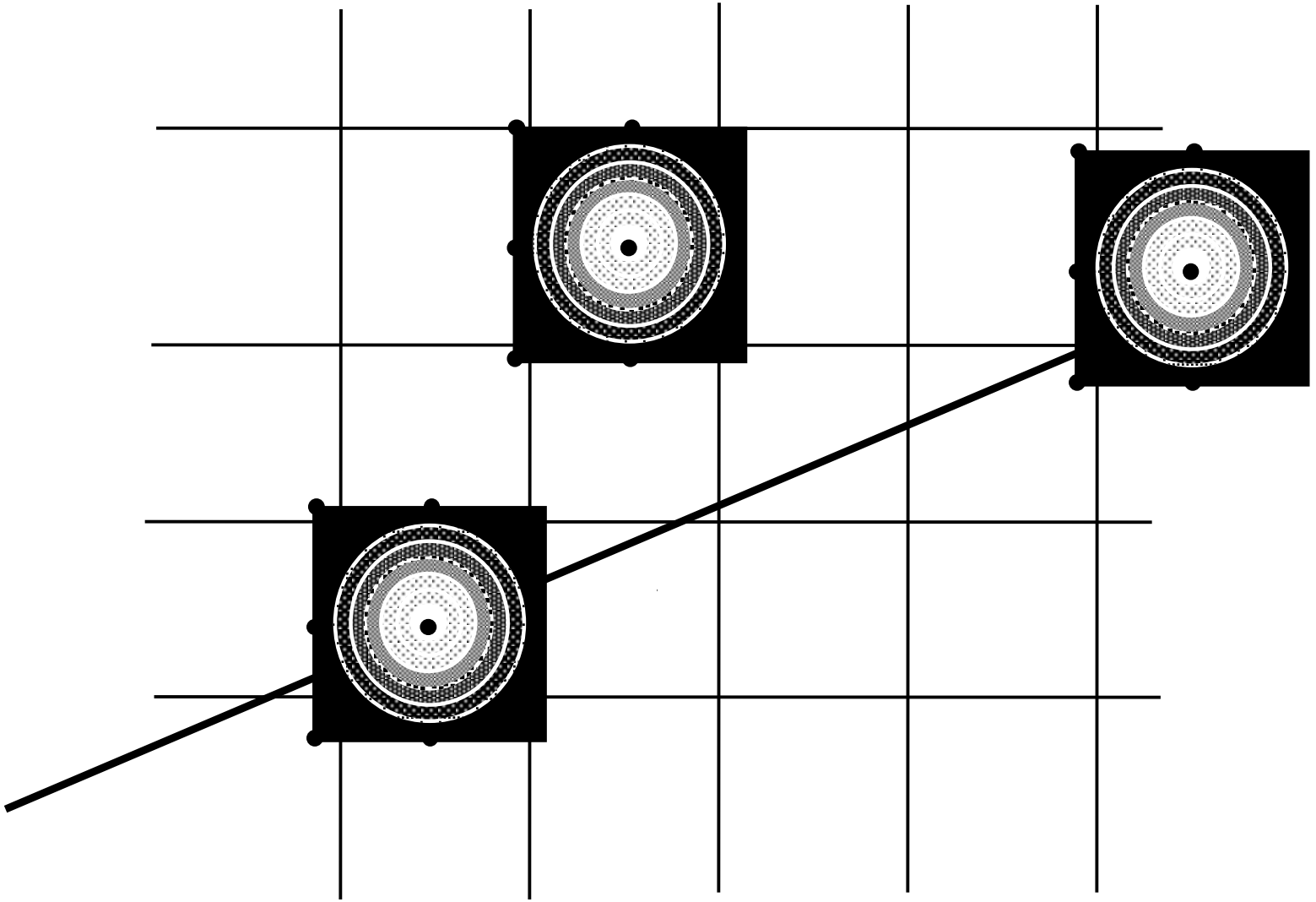
- Ideal filter is usually Gaussian
- Easier and much faster to approximate Gaussian with a cone

Anti-aliasing via sampling

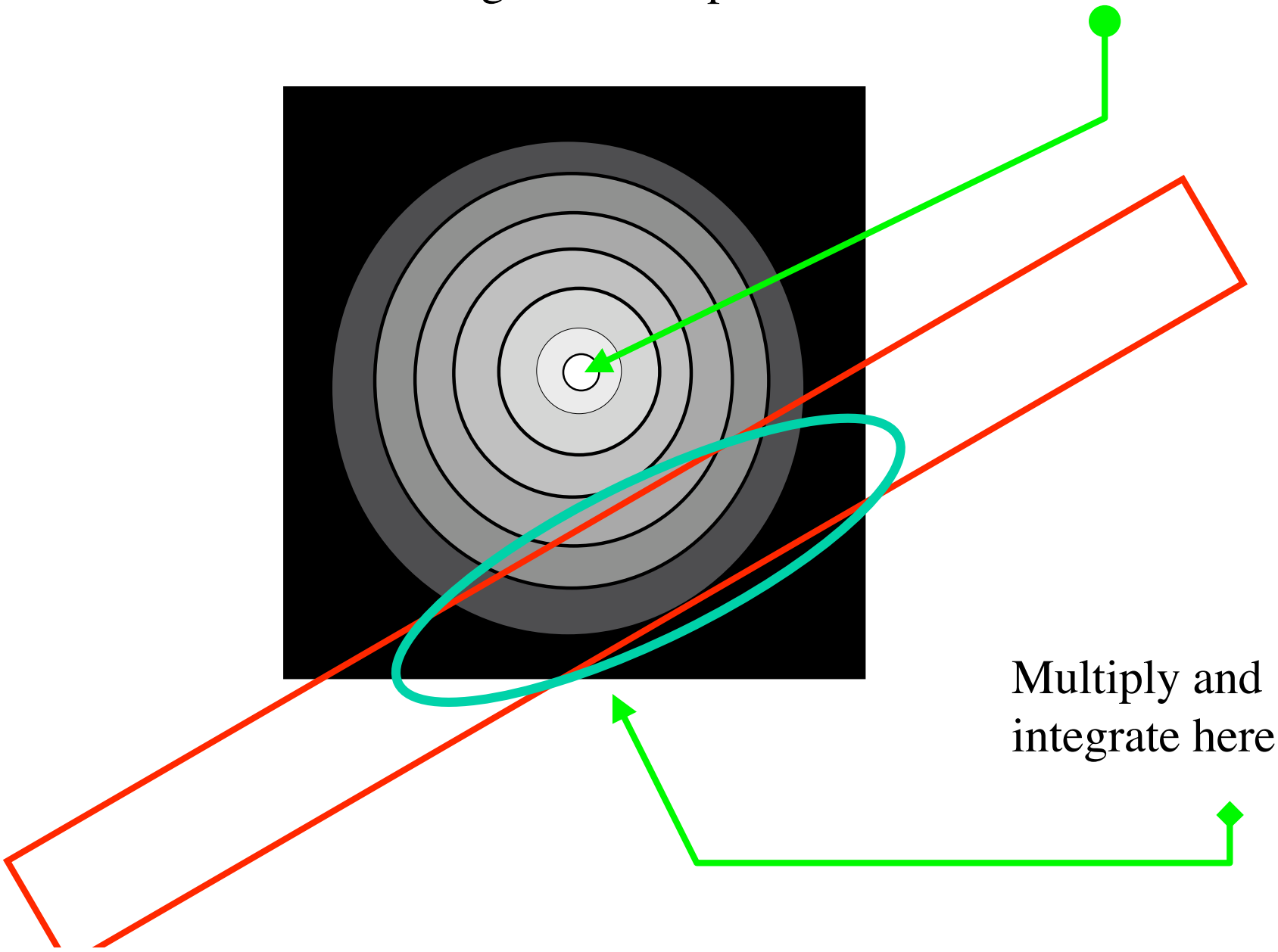
Technically we “convolve” the function representing the primitive g(x,y) with the filter, h (α, β)

$$g \quad h = \iint g(x-\alpha, y-\beta)\, h(\alpha,\beta)\, d\alpha\, d\beta$$

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



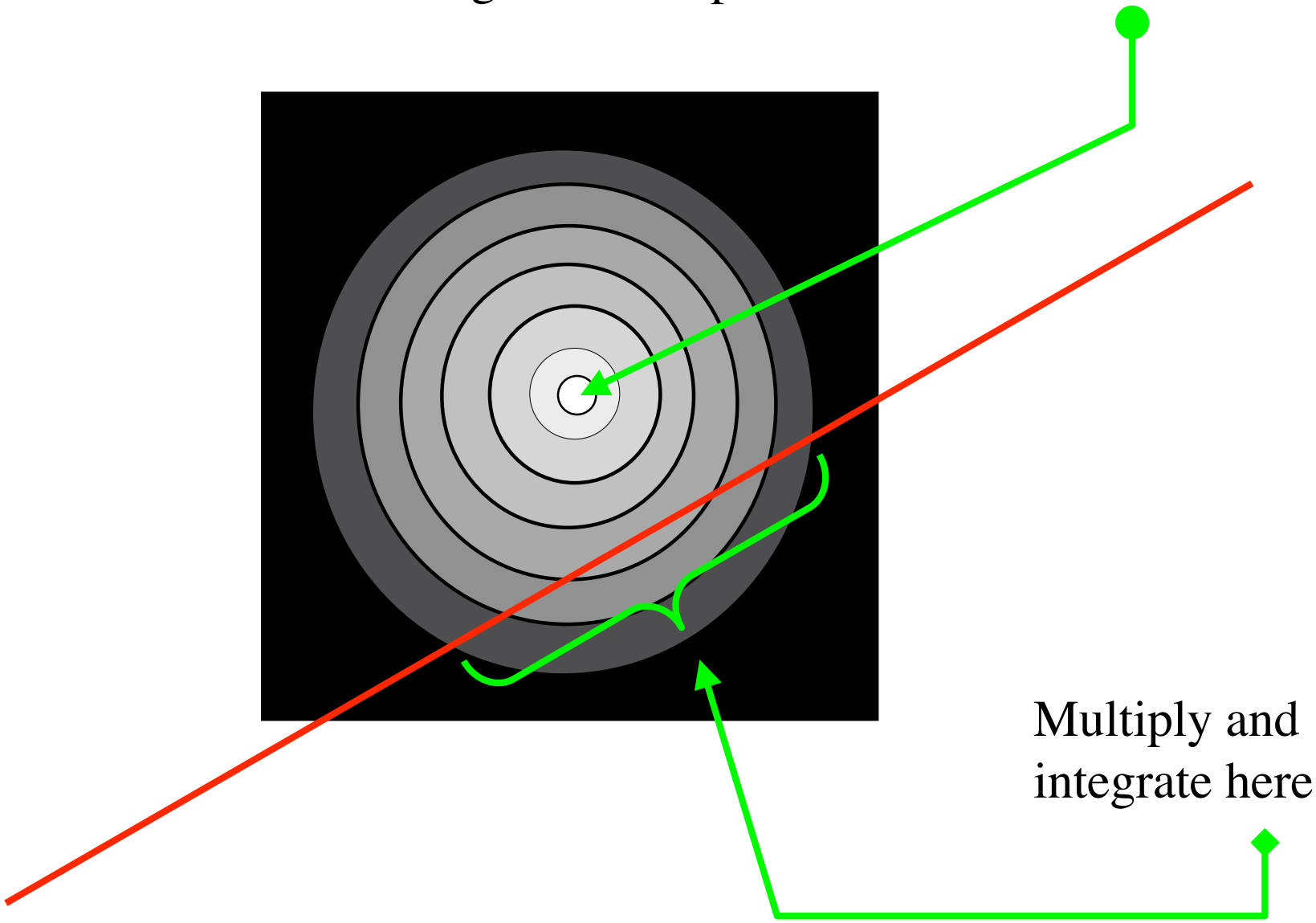
To calculate brightness for pixel with center here



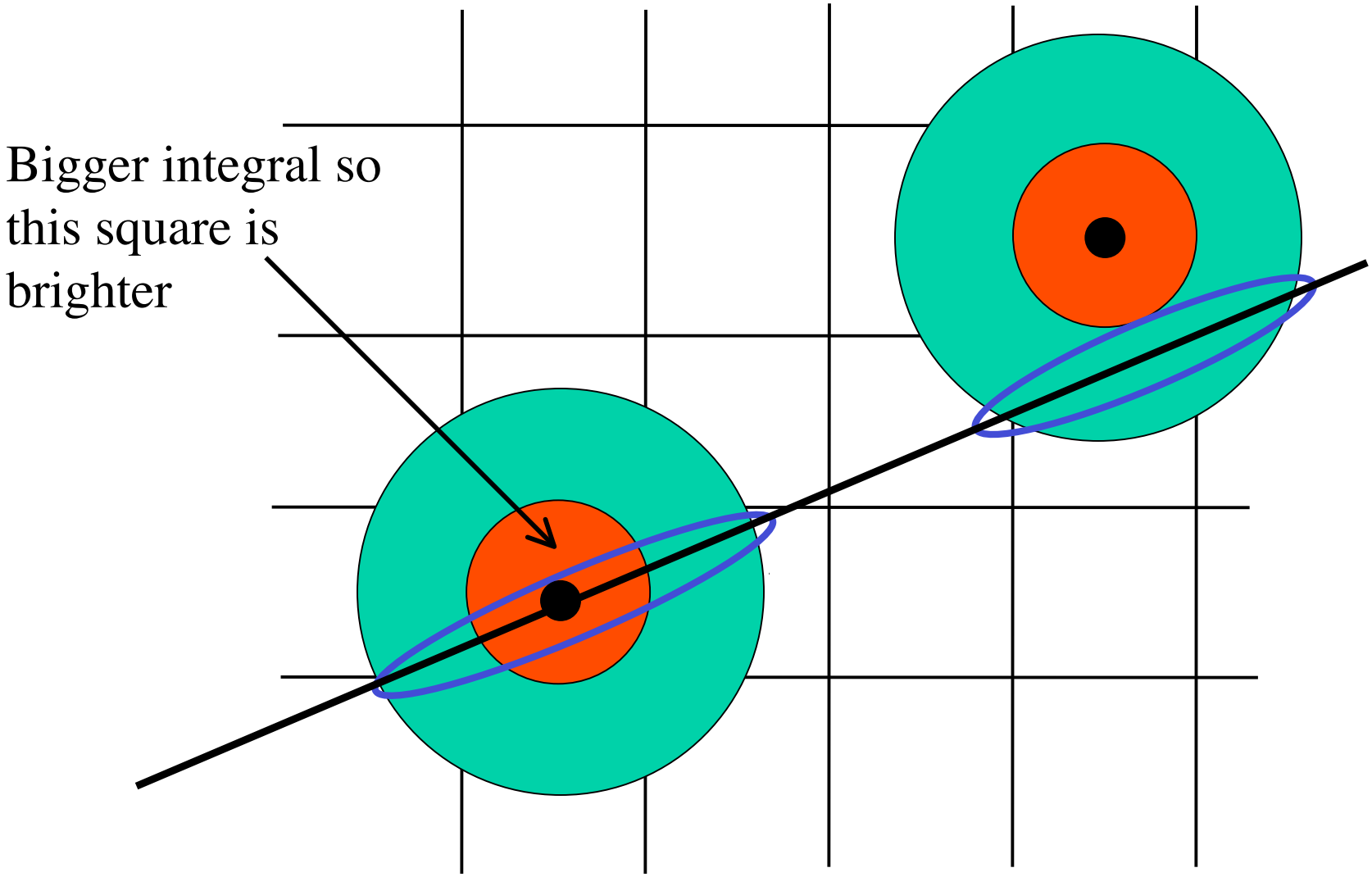
Line with no width

- 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--scale things so that if the line goes through the filter center, that the pixel gets the full color of the line.

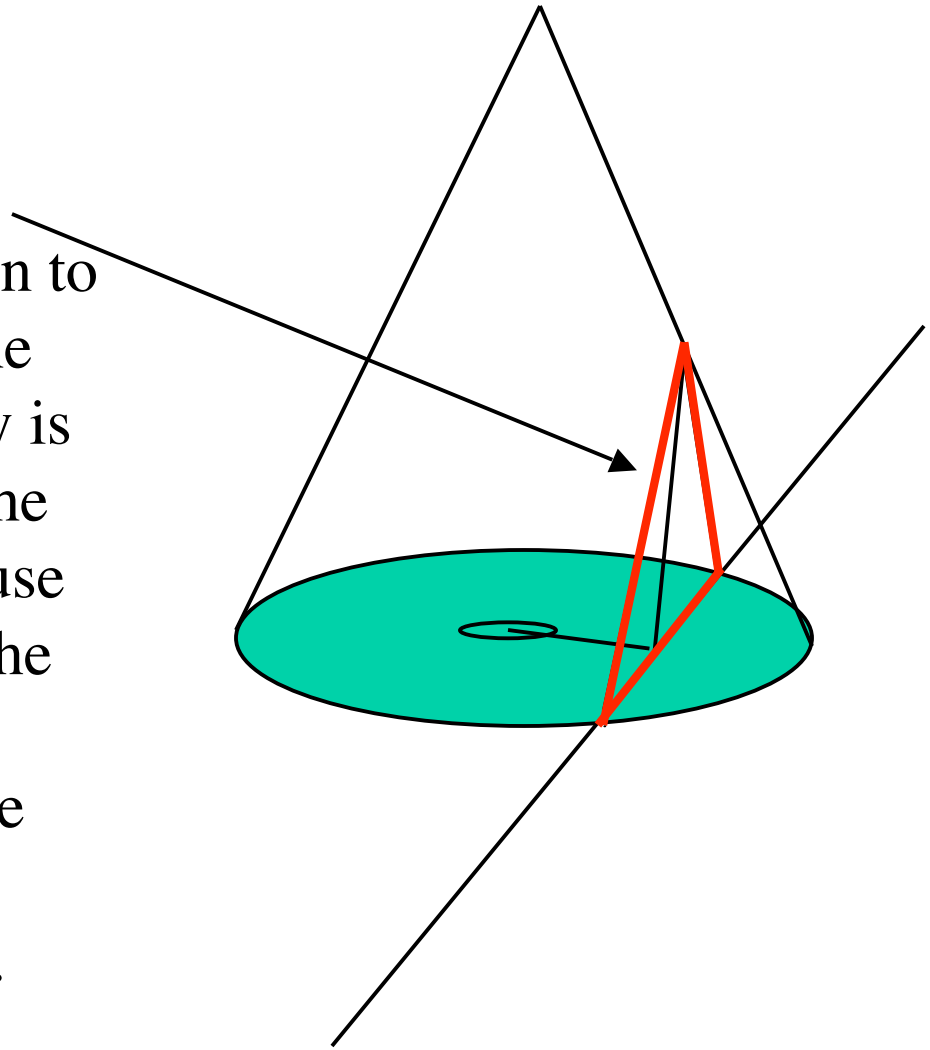
To calculate brightness for pixel with center here



Line with cone example

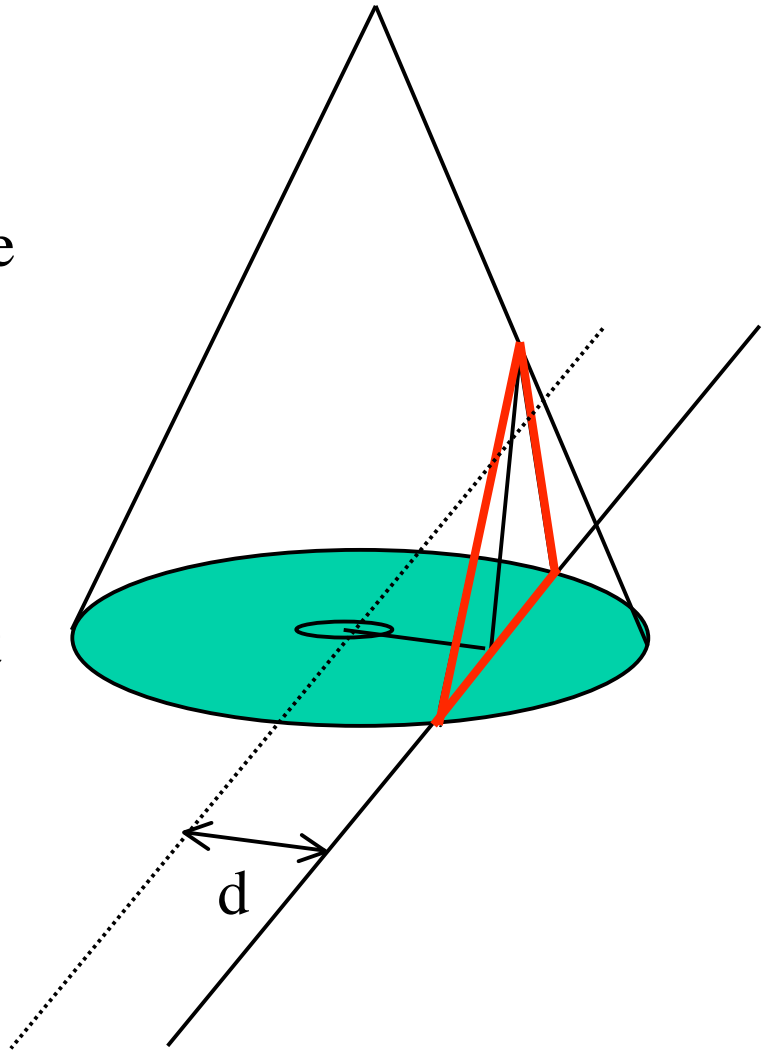


Linear approximation to boundary. The real boundary is curved. For the assignment, use the line. IE, the weighting function is the area triangle shown in red.



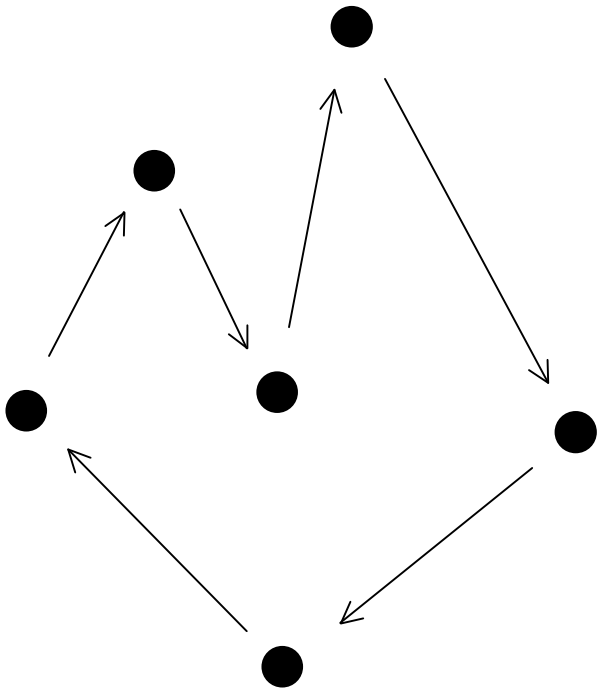
Hints for the assignment

- 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”.

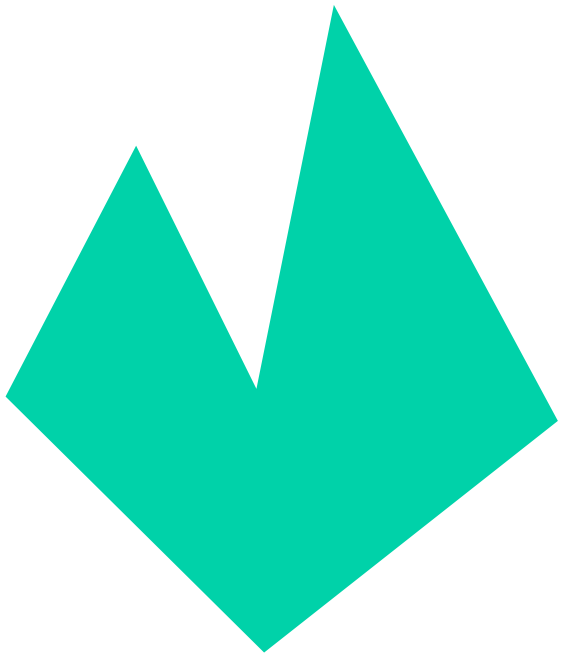


Scan converting polygons

(Handout: Section 3.5 (see 3.4 also))



Have



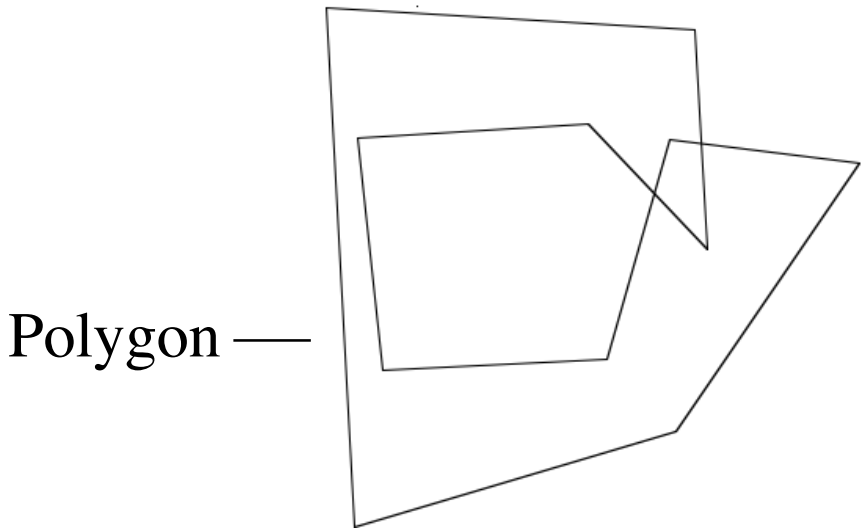
Need

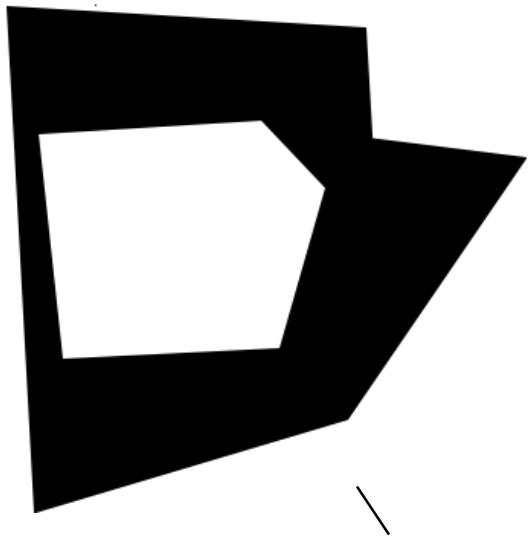
Filling polygons

- 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

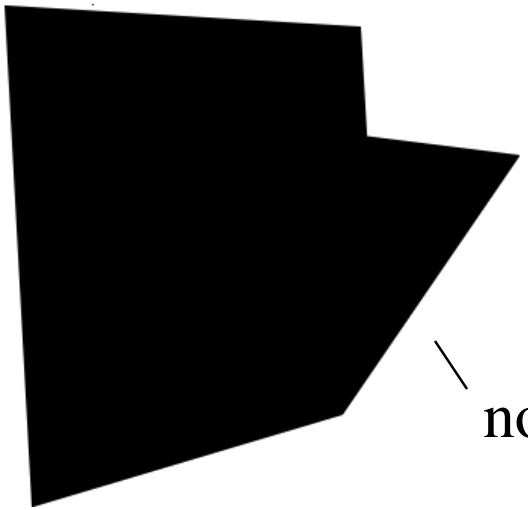
What is inside?

- 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)

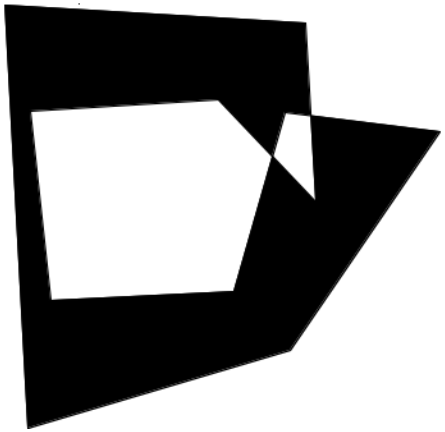




non-zero winding no.



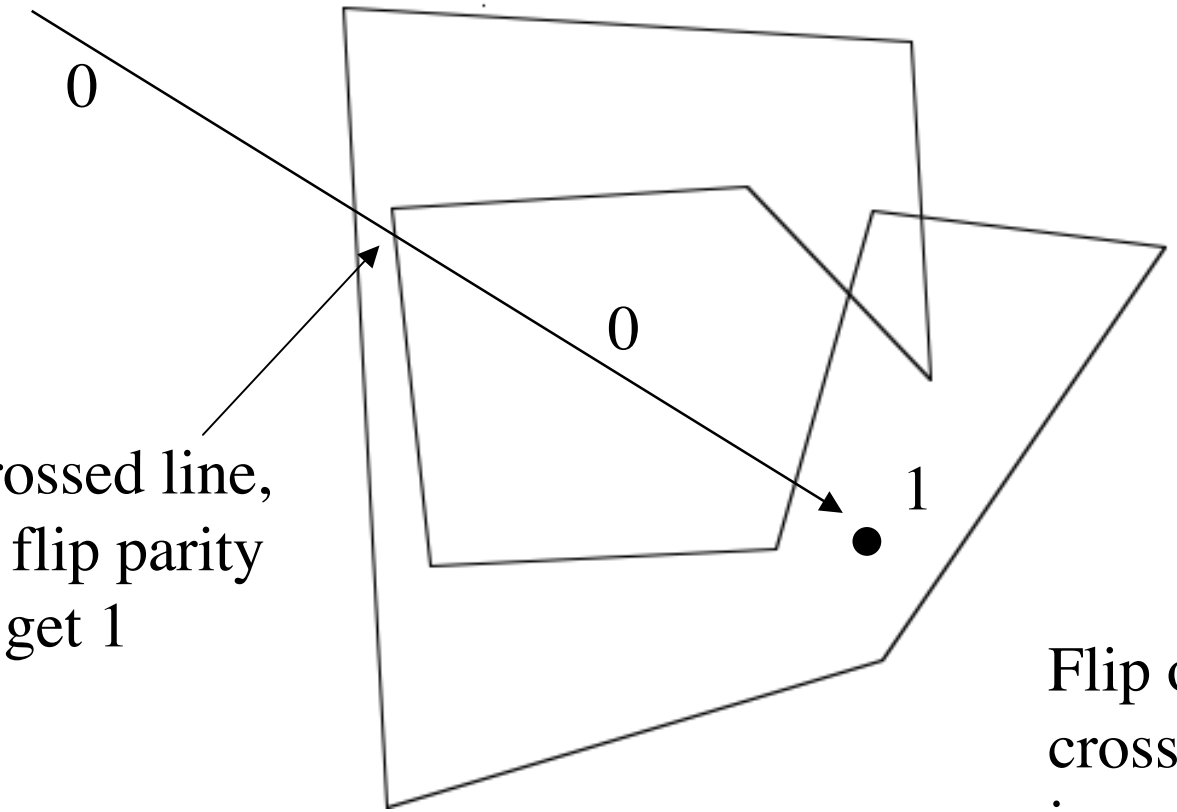
non-exterior



parity

Parity rule--details

"Far away"



Crossed line, so flip parity to get 1

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