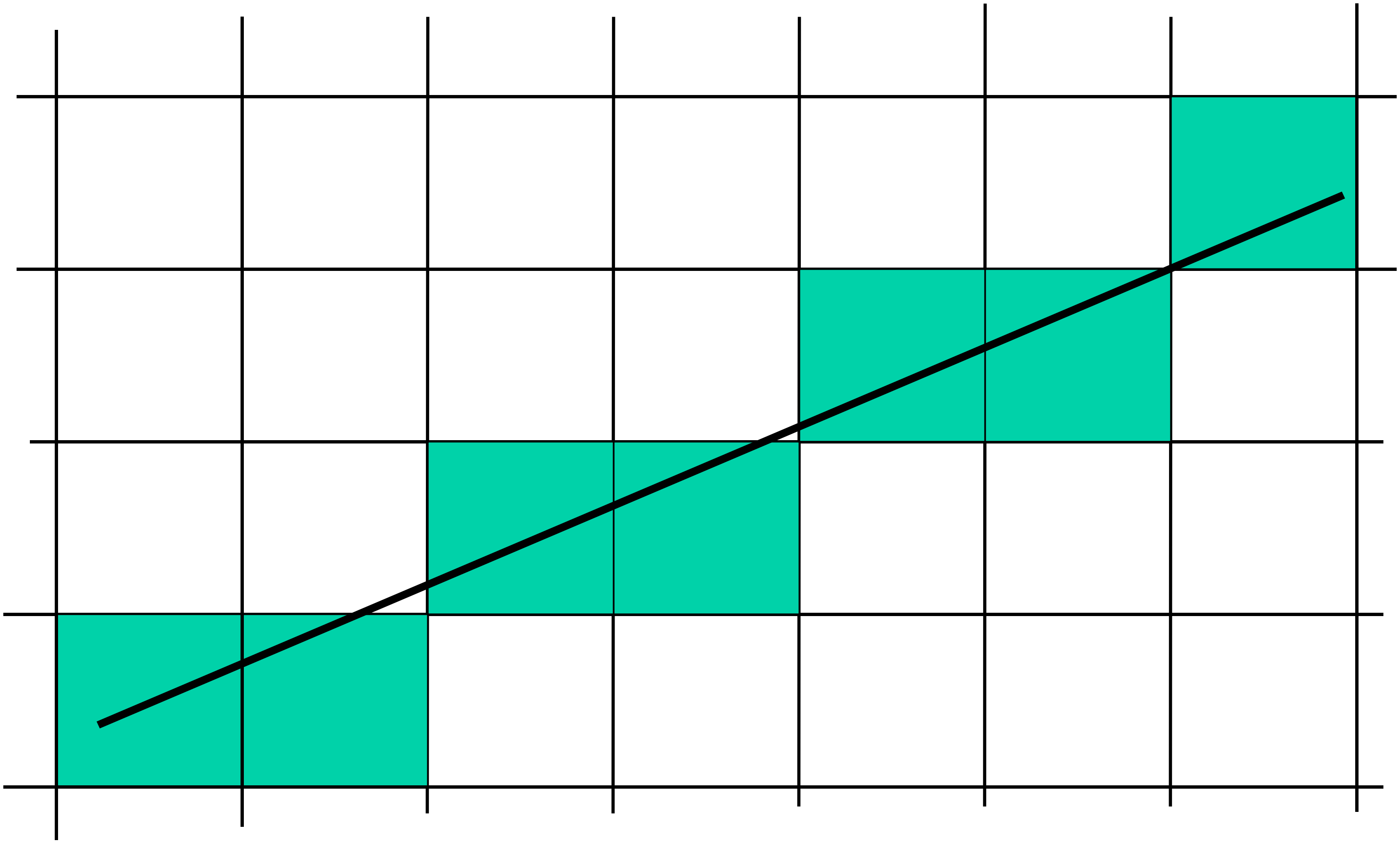
Line drawing--aliasing



Aliasing

- We are using discrete binary squares to represent perfect mathematical entities
- To get a value for that that square we used a “sample” at a particular discrete location.
- The sample is somewhat arbitrary due to the choice of discretization, and reflects our discretization (leading to the jagged edges)
- Points and lines as discussed so far have no width so to have them visible we concocted a way to sample them based on which discrete cell was closer
- Solutions?

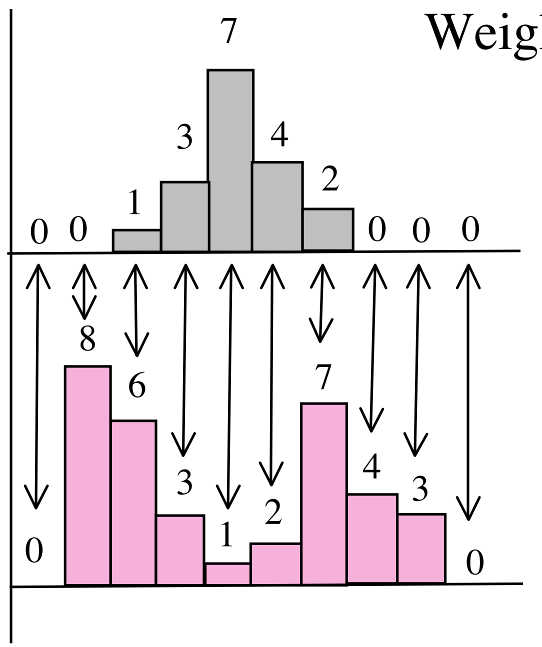
Aliasing (cont)

- 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
- A more principled approach is to “filter” before sampling.

Linear Filters (background)

- General process: Form new image whose pixels are a **weighted sum** of original pixel values, using the same set of weights at each point.

Weights (kernel of filter)

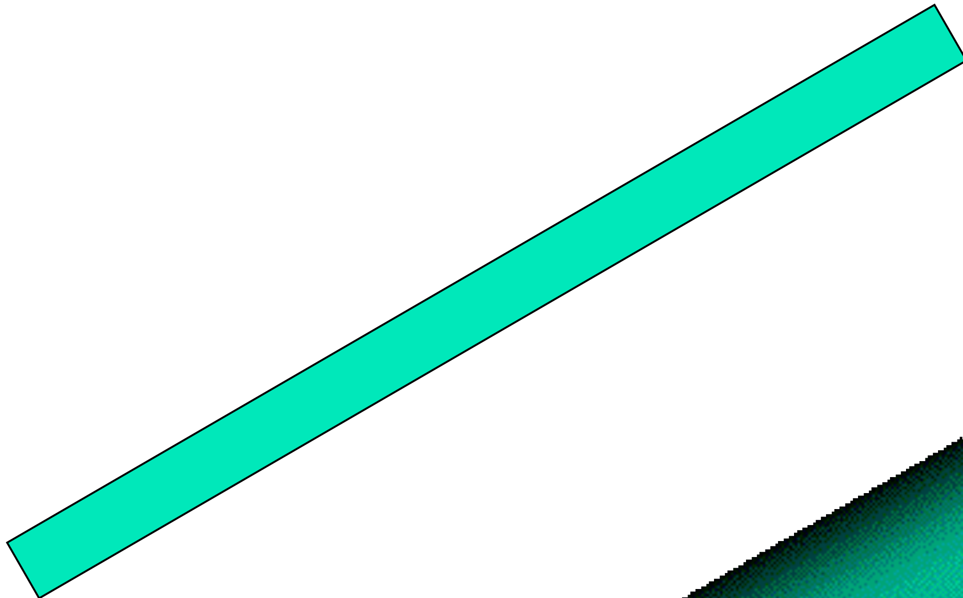


Multiply lined up pairs of numbers and then sum up to get weighted average at the filter location. Then shift the filter and do the same to get the next value.

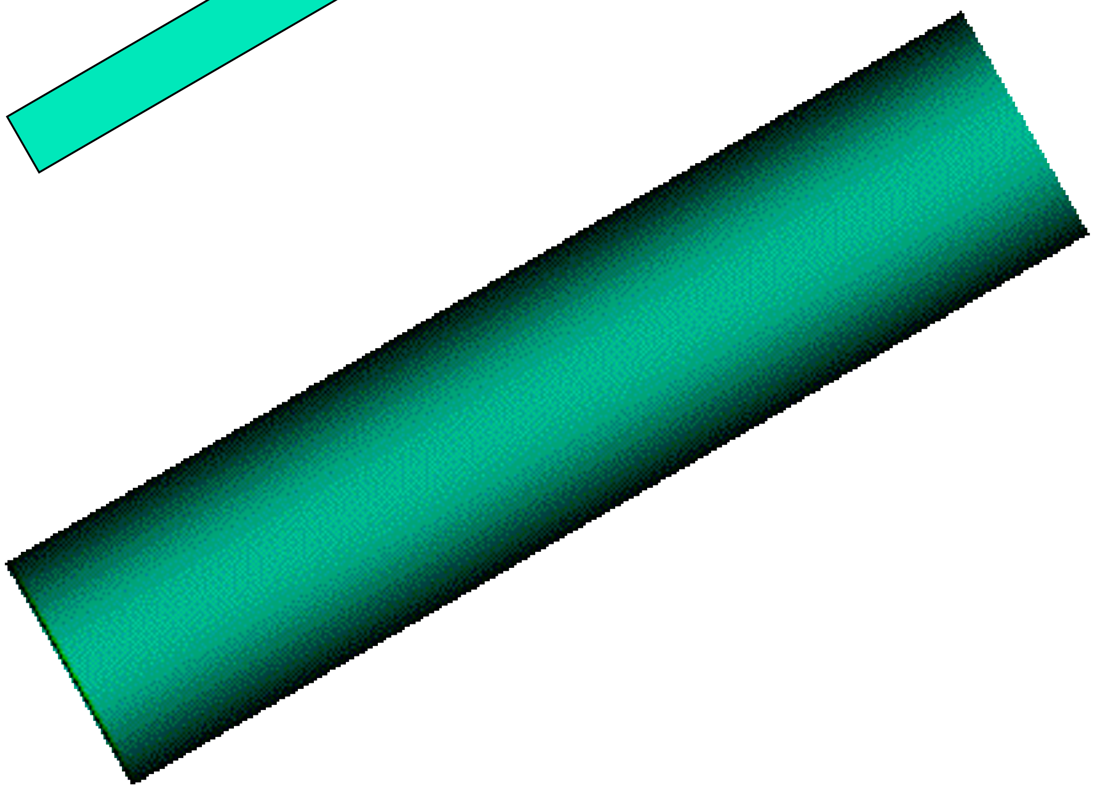
Signal

Aliasing via filtering and then sampling

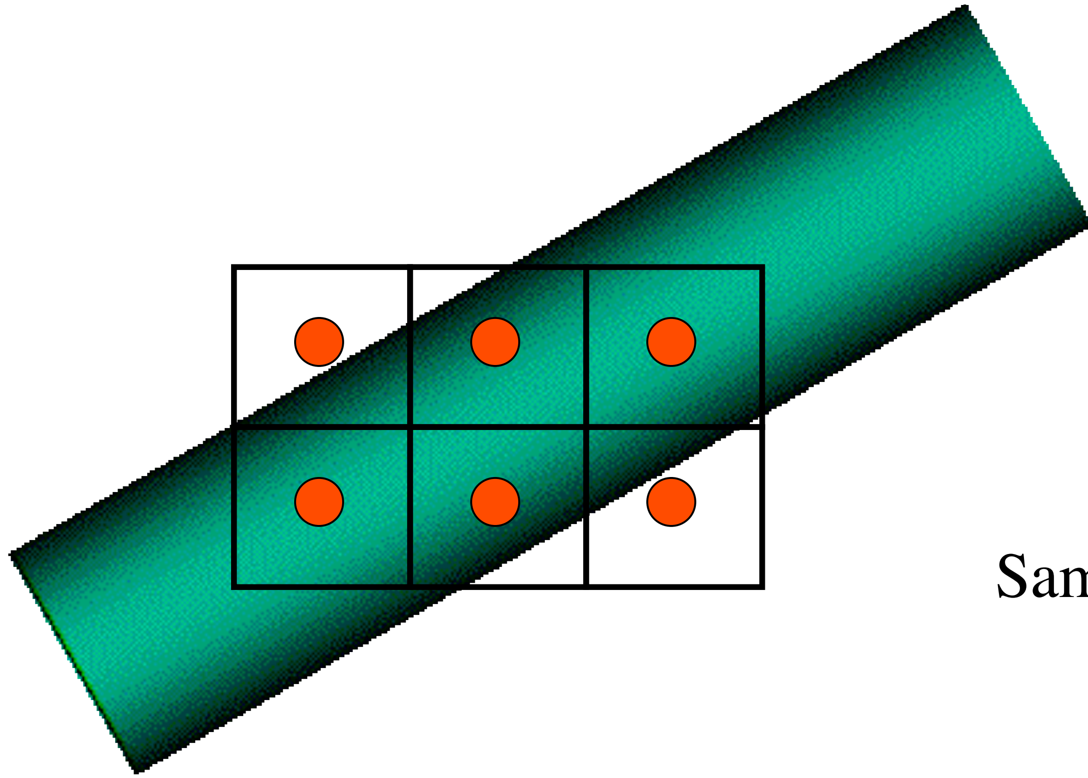
- A filter can be thought of as a weighted average. The weights are given by the filter function. (Examples to come).
- **Conceptually**, we smooth (convolve) the object to be drawn by applying the filter to the mathematical representation.
- This blurs the object, widens the area it occupies
- Now we “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.
- (**Technically** we only need to blur at the sampling locations)



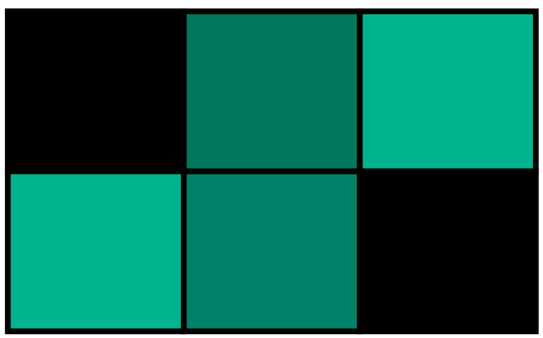
Line with width



Blurred



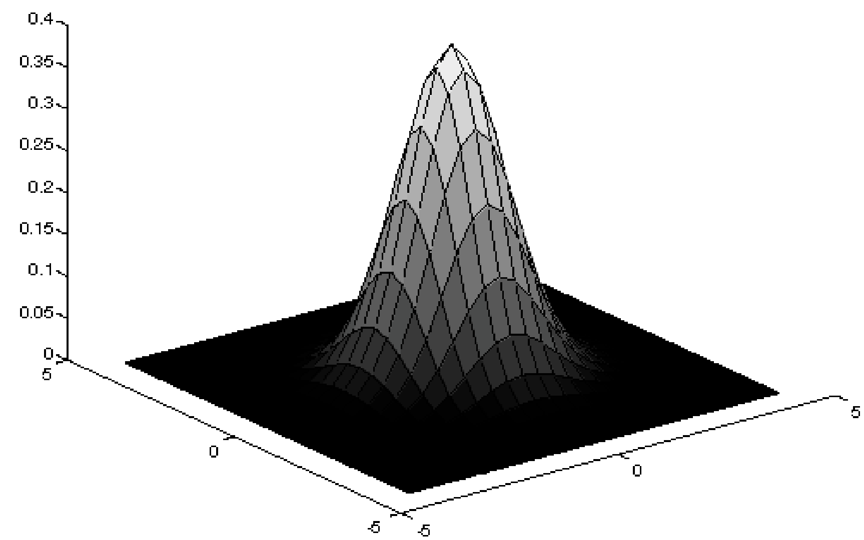
Sample



Paint with
sample value

Aliasing via filtering and then sampling

- Ideal filter is usually Gaussian
- Easier and much faster to approximate Gaussian with a cone
- See Figure 3.38, page 12



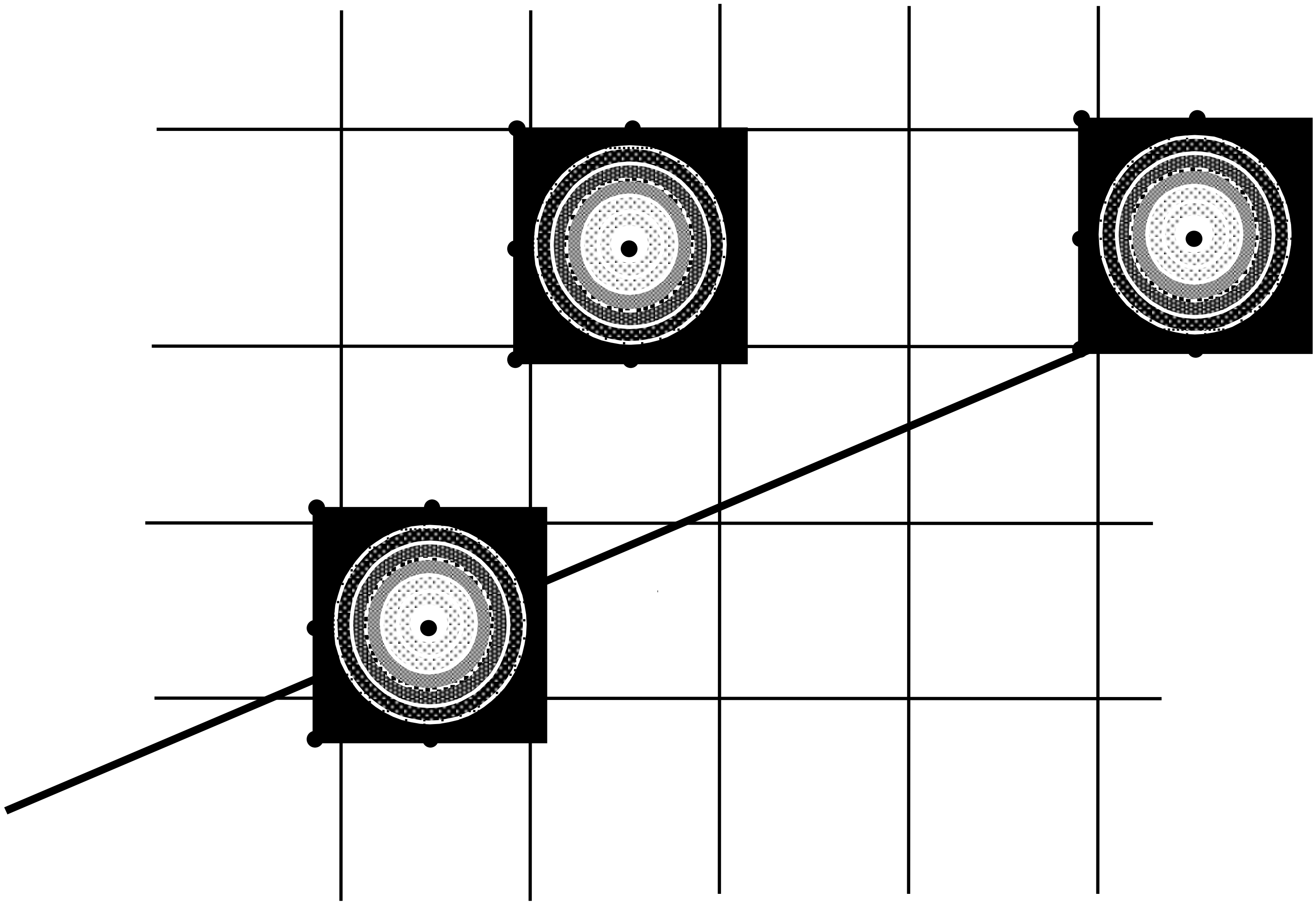
Anti-aliasing via filtering and then sampling

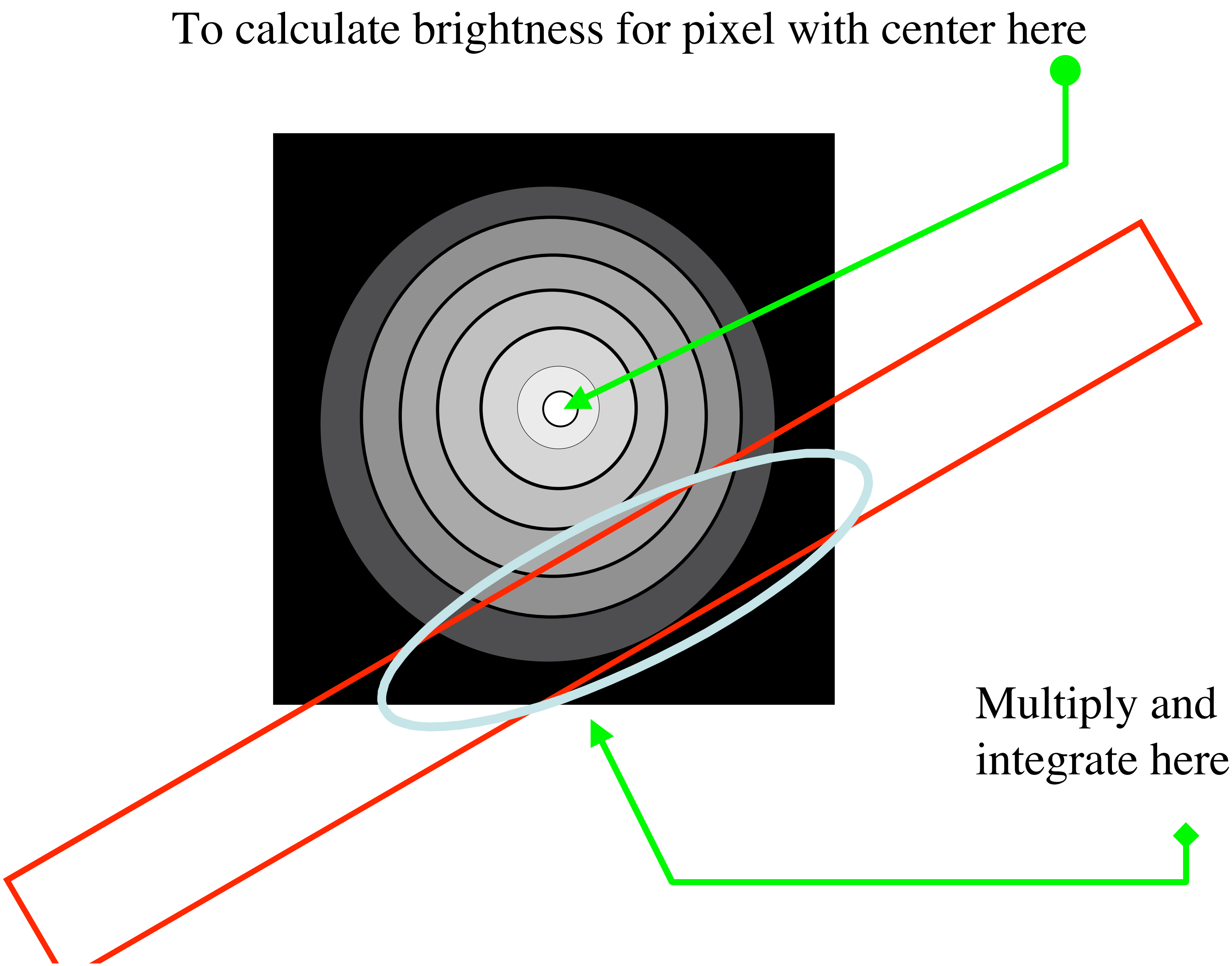
Technically we "convolve" the function representing the primitive g(x,y) with the filter, h (α, β)

$$g * h = \int\int g(x-\alpha, y-\beta)\, h(\alpha,\beta)\, d\alpha\, d\beta$$

Exact expression is optional

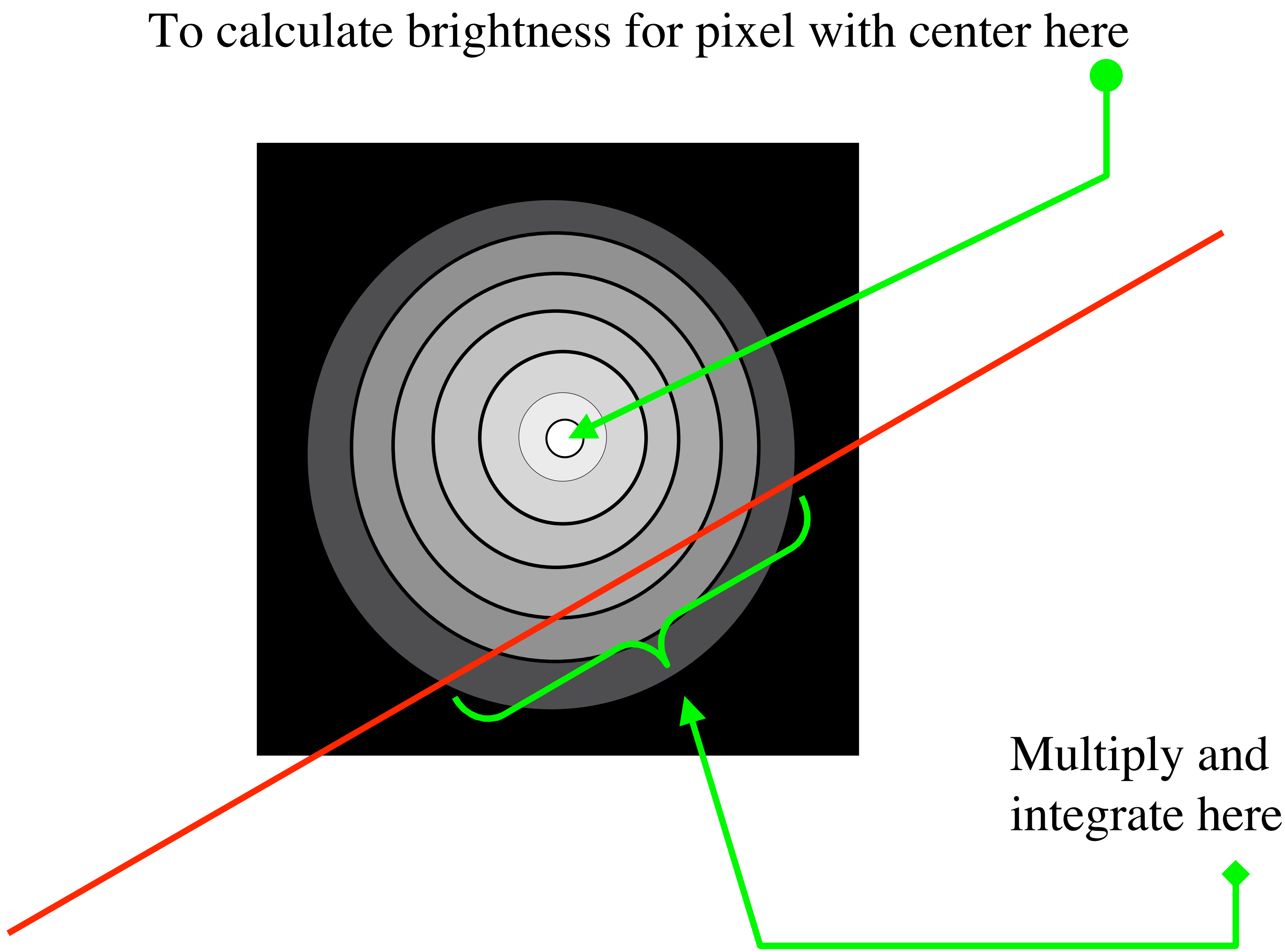
Line drawing--anti-aliasing--a filter at each point (3 shown)



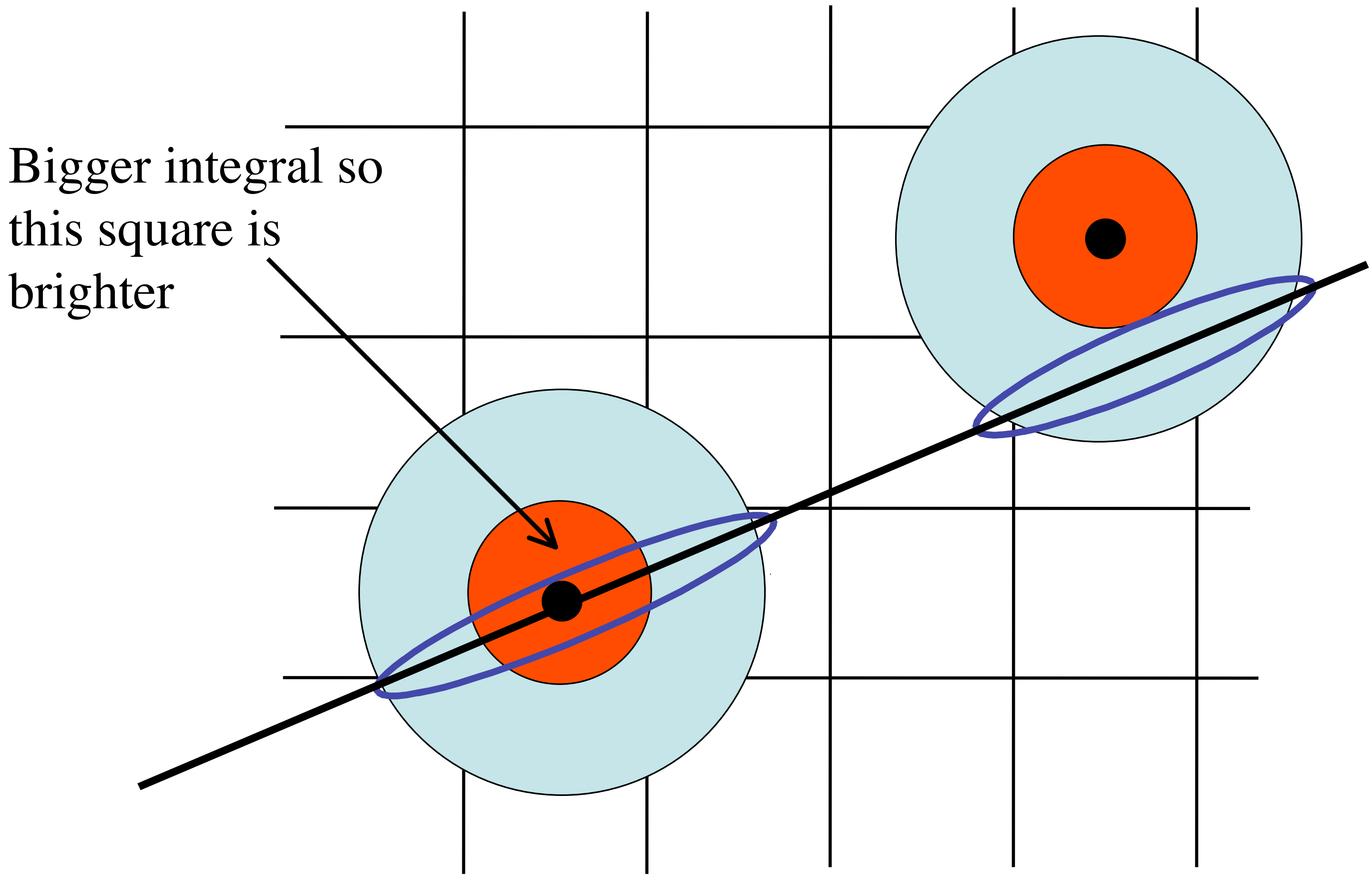

To calculate brightness for pixel with center here
Multiply and integrate here

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 (along the line).
- Normalization--ensure that if the line goes through the filter center, that the pixel gets the full color of the line.

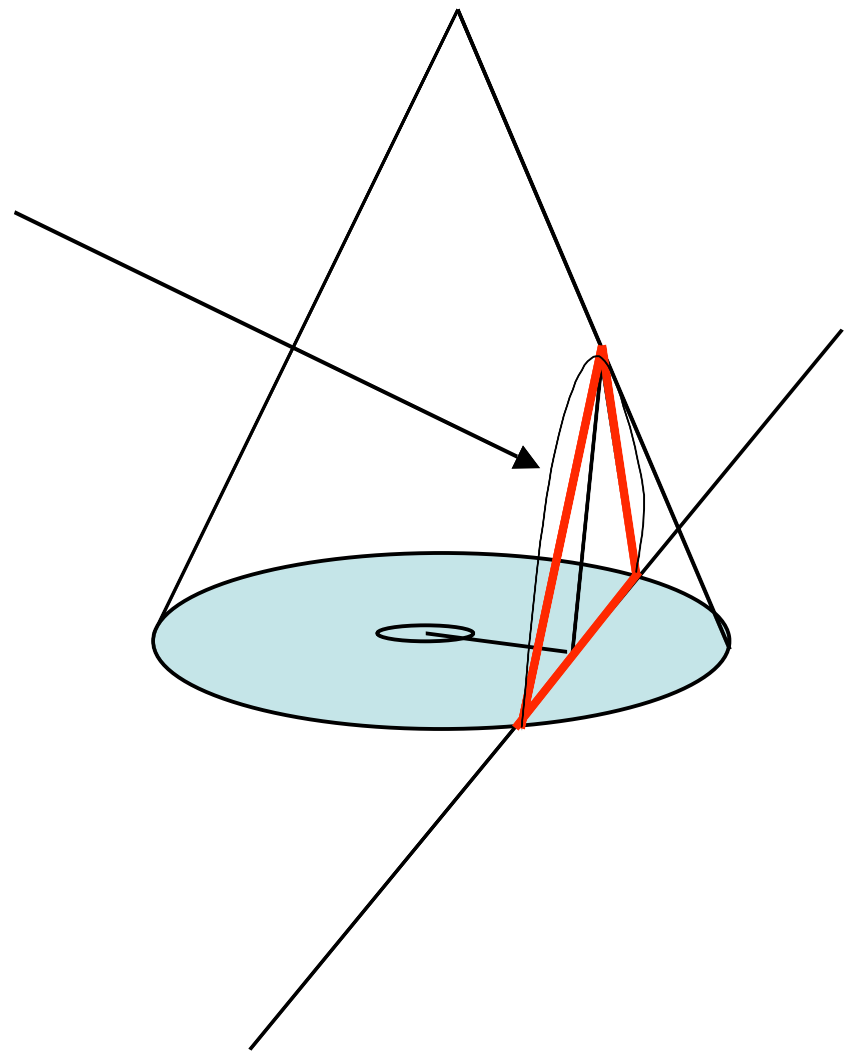

To calculate brightness for pixel with center here
Multiply and integrate here

Line with cone example



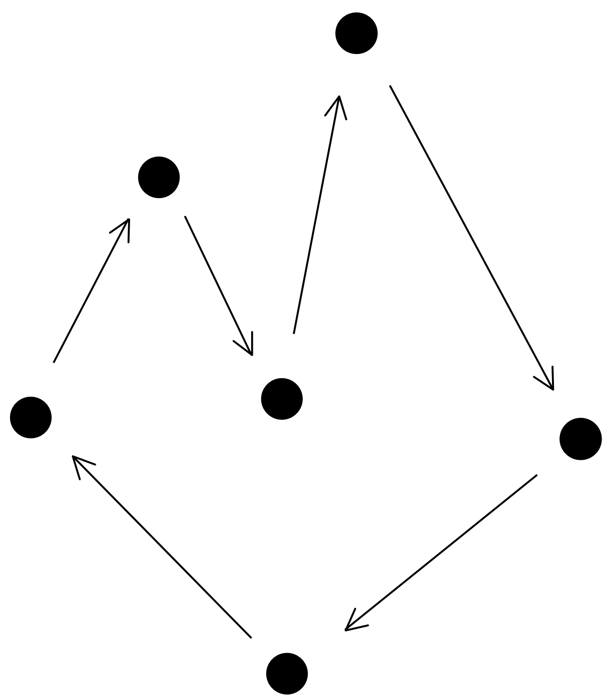
Approximating a Gaussian filter with a cone

Parabolic boundary which can be approximated with the lines shown in red. In either case, an analytical solution can be computed so that filtering can be done by a formula (rather than numerical integration).



Scan converting polygons

(Text: Section 3.5 (see 3.4 also))



Have

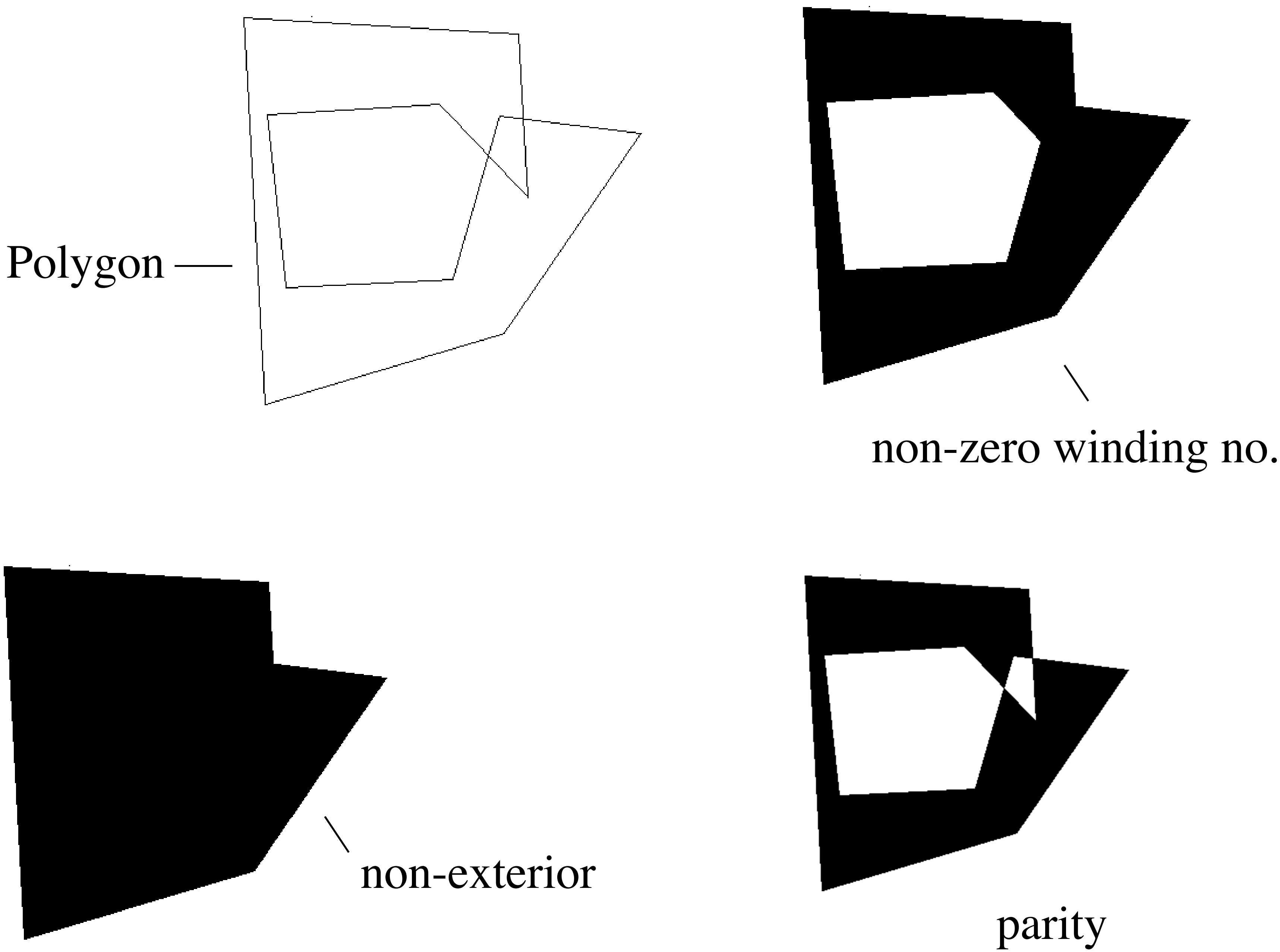
Need

Filling polygons

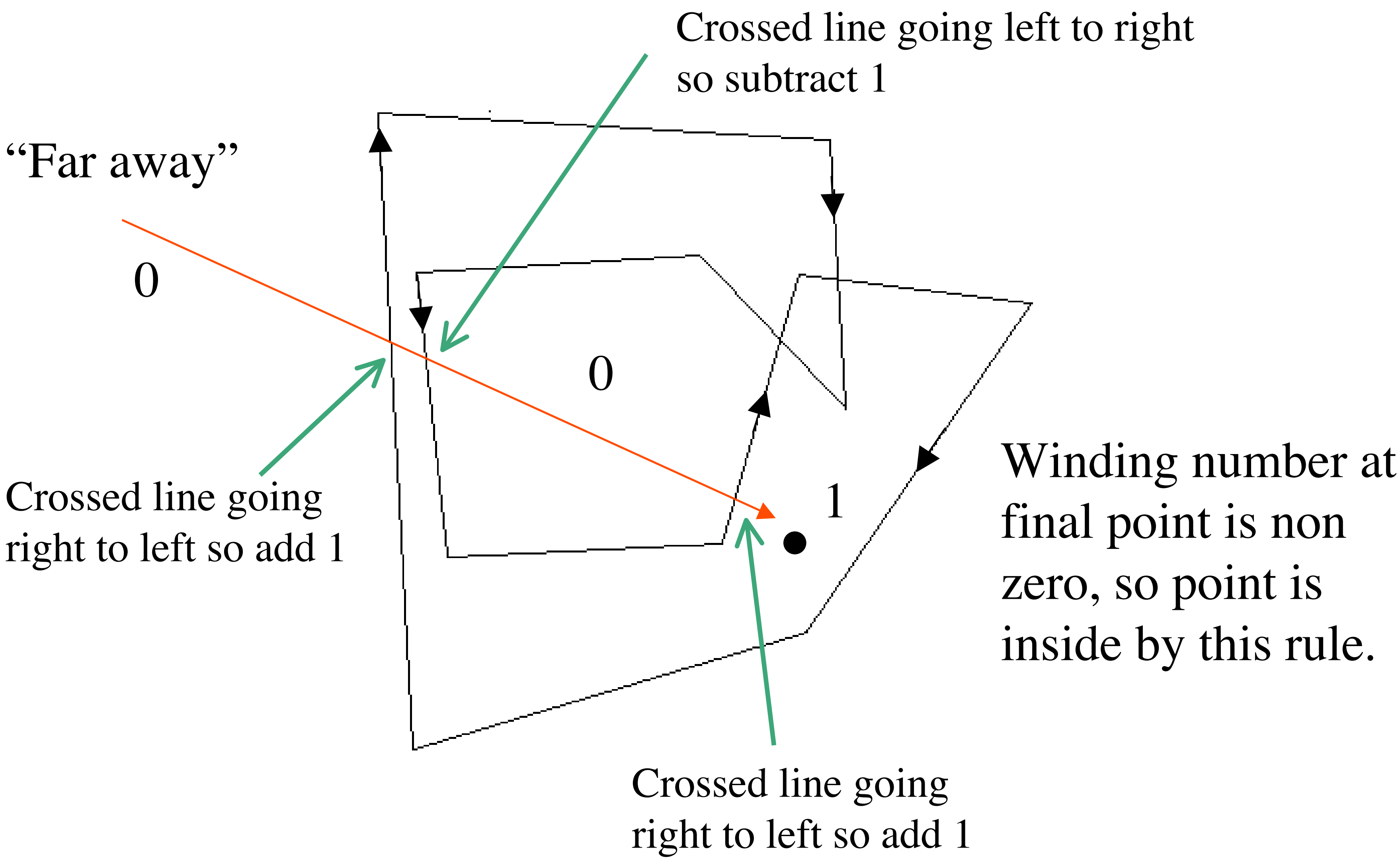
- Polygons are defined by a list of edges - each is a pair of vertices (order counts)
- Assume that each vertex is an integer vertex, and polygon lies within frame buffer
- Need to define what is inside and what is outside

What is inside?

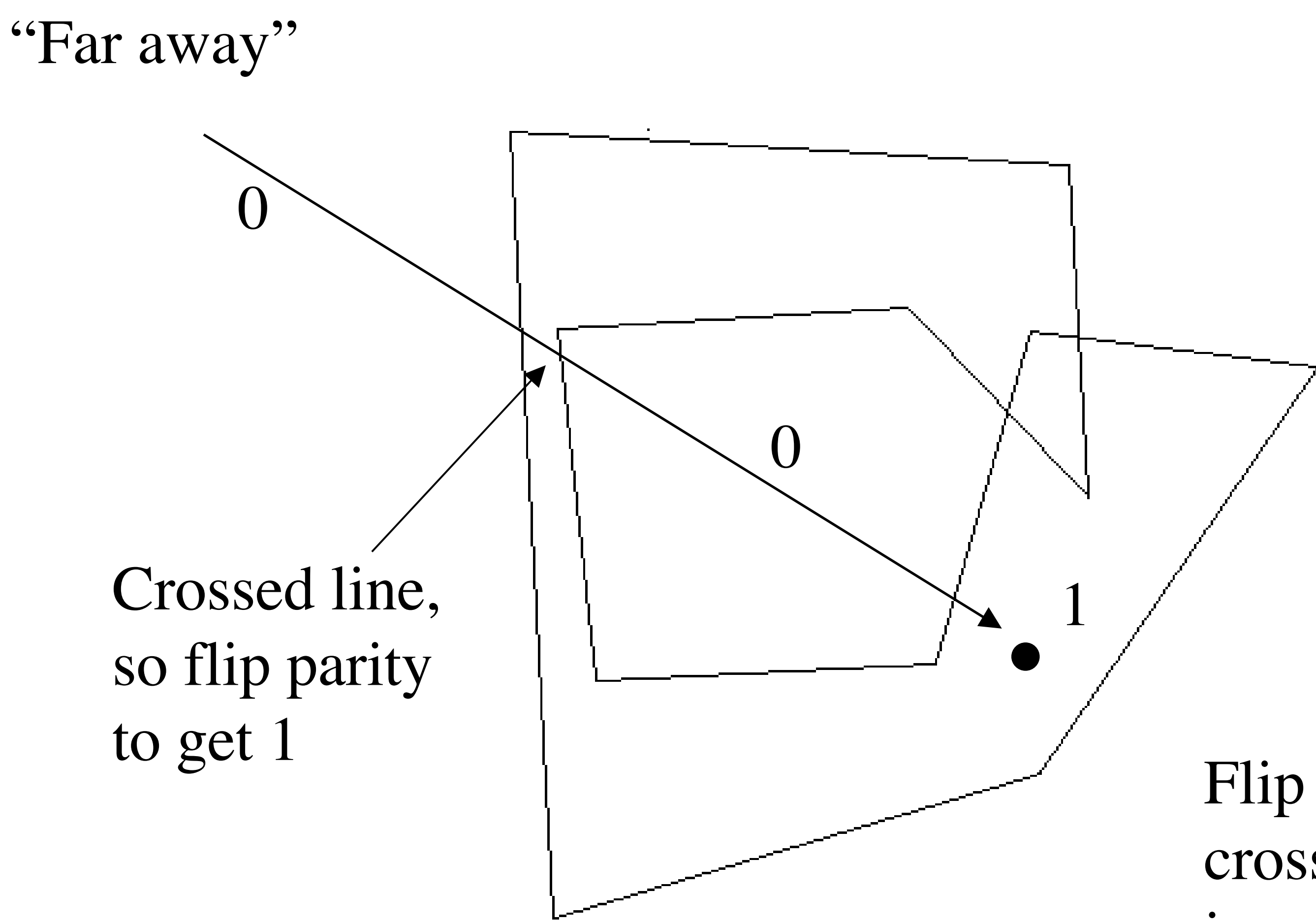
- Easy for simple polygons - no self intersections
- For general polygons, three rules are used:
  – non-exterior rule
    - (Can you get arbitrarily far away from the polygon without crossing a line)
  – non-zero winding number rule
  – parity rule (most common--this is the one we will generally used)


Polygon
non-zero winding no.
non-exterior
parity

Non-zero winding number--details



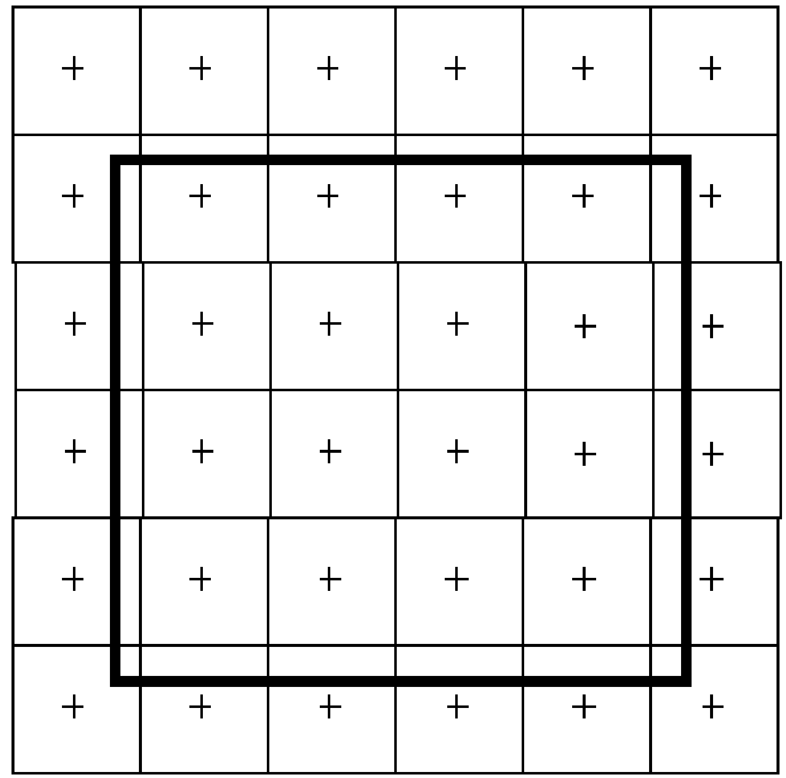
Parity rule--details



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.



Which pixel is inside?

In the context of the sweep fill algorithm to come soon: Suppose we are sweeping from left to right

1) Going from outside to inside, then take true intersection, and **round up** to get first interior point.
2) Going from inside to outside, then take true intersection, and **round down** to get last interior point.

Note that if we are considering an adjacent polygon, 1) and 2) are reversed, so it should be clear that for most cases, the pixels owned by each polygon is well defined (and we don't erase any when drawing the other polygon). Exceptions?

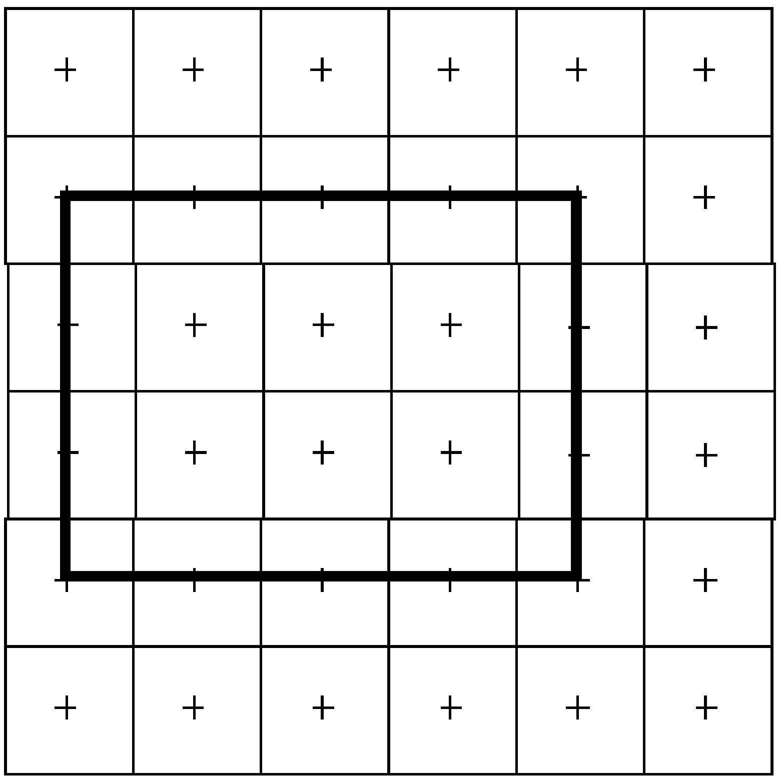
Ambiguous cases

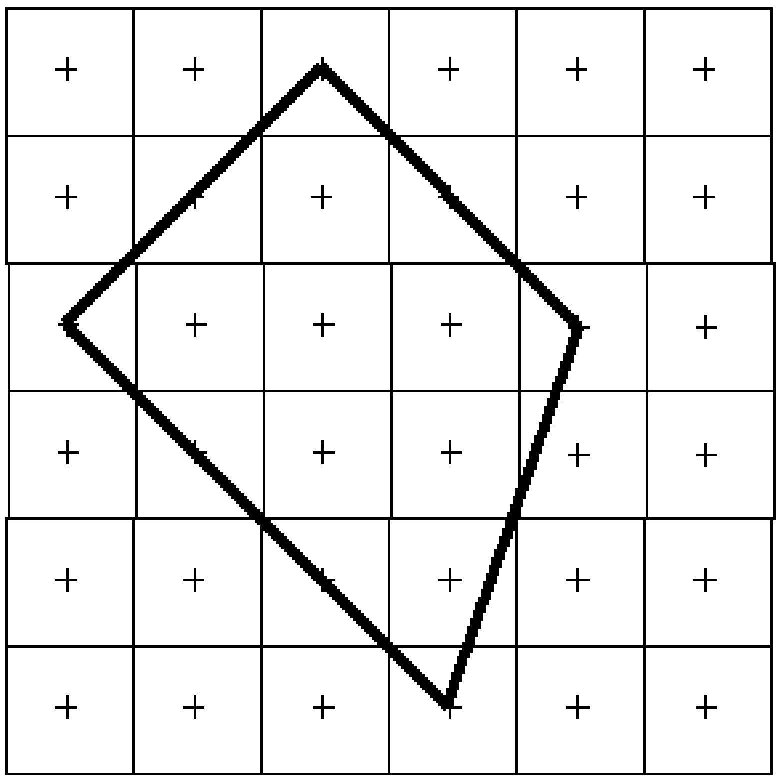
- What if a pixel is exactly on the edge?
- Polygons are usually adjacent to other polygons, so we want a rule which will give the pixel to *one* of the adjacent polygons or the *other* (as much as possible).
- Basic rule: Draw left and bottom edges
  - horizontal edge? if $(x+\partial, y+\square)$ is in, pixel is in
  - otherwise if $(x+\partial, y)$ is in, pixel is in
  - in practice one implements a sweep fill procedure that is consistent with this rule (we don't test the rule explicitly)

Ambiguous cases

- What if it is a vertex between the two cases?
  - In this case we essentially draw left vertices and bottom vertices, but details get absorbed into scan conversion (sweep fill). Note that the algorithm in the book solves this problem by making a special case for parity calculation (y_{min} vertices are counted for parity calculation, but y_{max} are not)
  - As mentioned on the bottom of page 86 of the text, there is no perfect solution to the problem. There will be edges that could be closed, but are left open in case another polygon comes by that would compete for pixels, and, on occasion, there is a “hole” (preferred compared to rewriting pixels).

Ambiguous inside cases (?)





Ambiguous inside cases (answer)