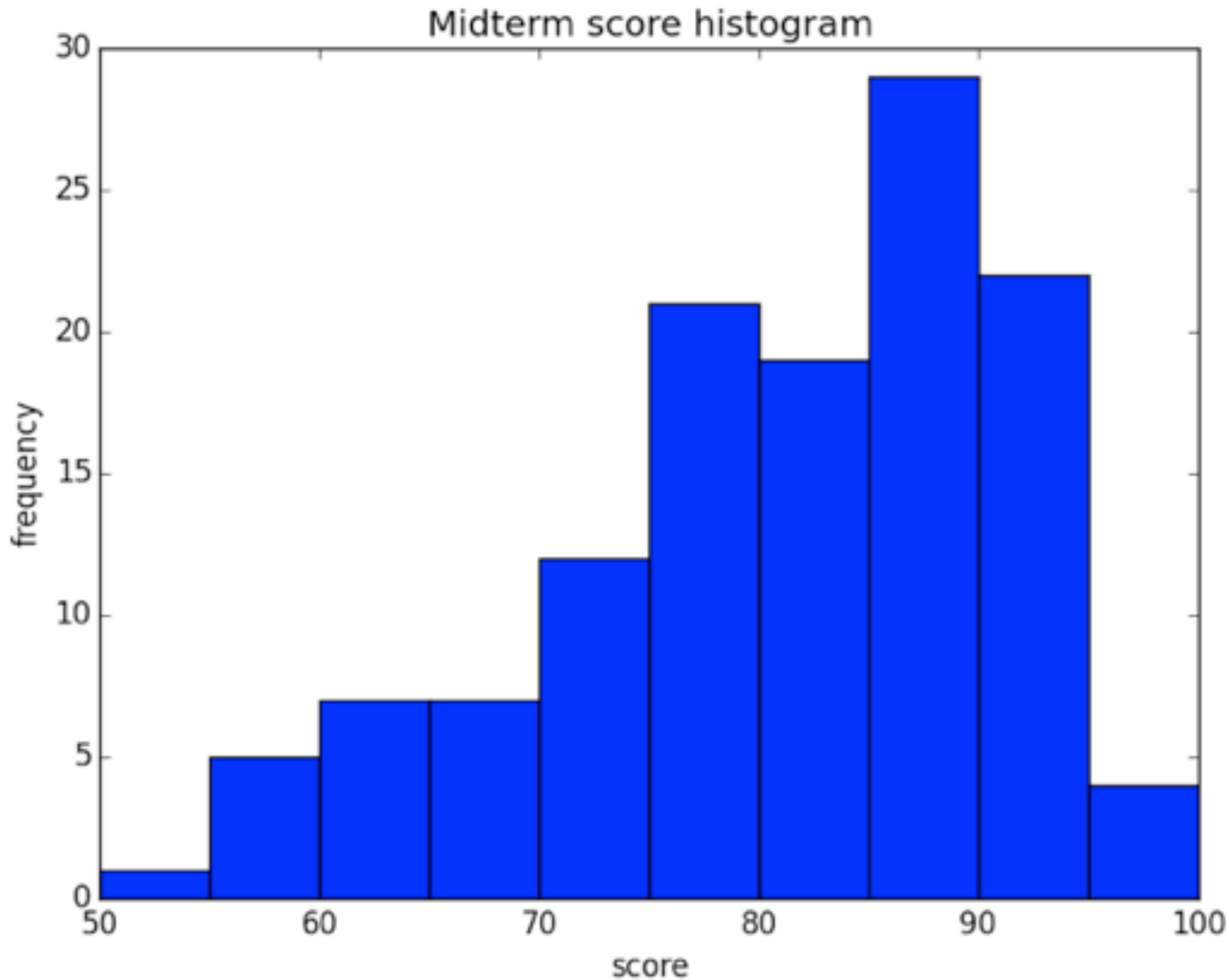
# Antialiasing & Compositing

CS4620 Lecture 17





(with previous instructors James/Bala, and some slides courtesy Leonard McMillan)



Midterm score histogram
frequency
score
0
5
10
15
20
25
30
50
60
70
80
90
100



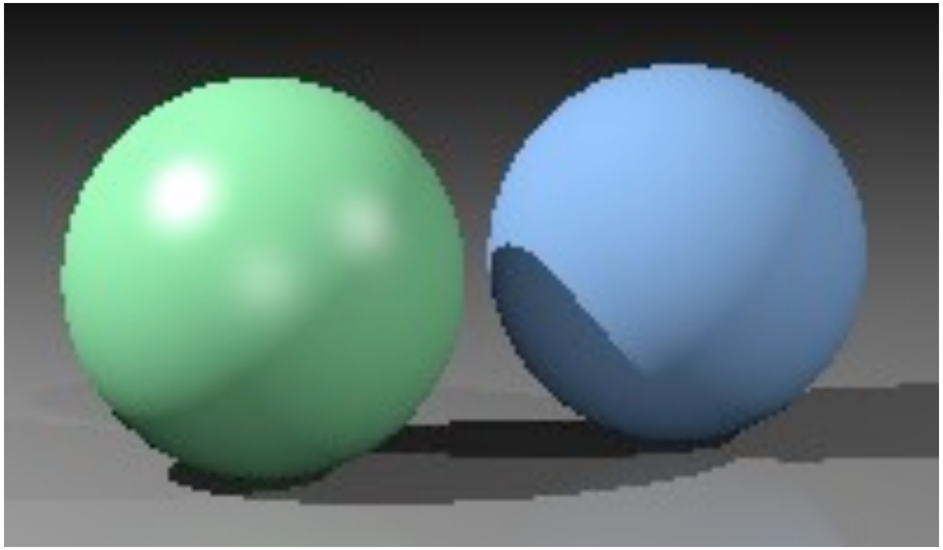
# Pixel coverage

- Antialiasing and compositing both deal with questions of pixels that contain unresolved detail
- Antialiasing: how to carefully throw away the detail
- Compositing: how to account for the detail when combining images

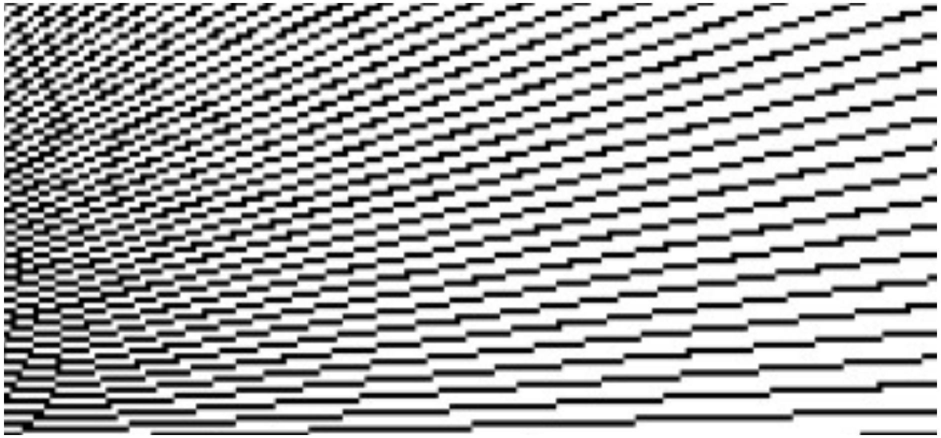
# Aliasing

point sampling a
continuous image:

continuous image defined
by ray tracing procedure



continuous image defined
by a bunch of black rectangles

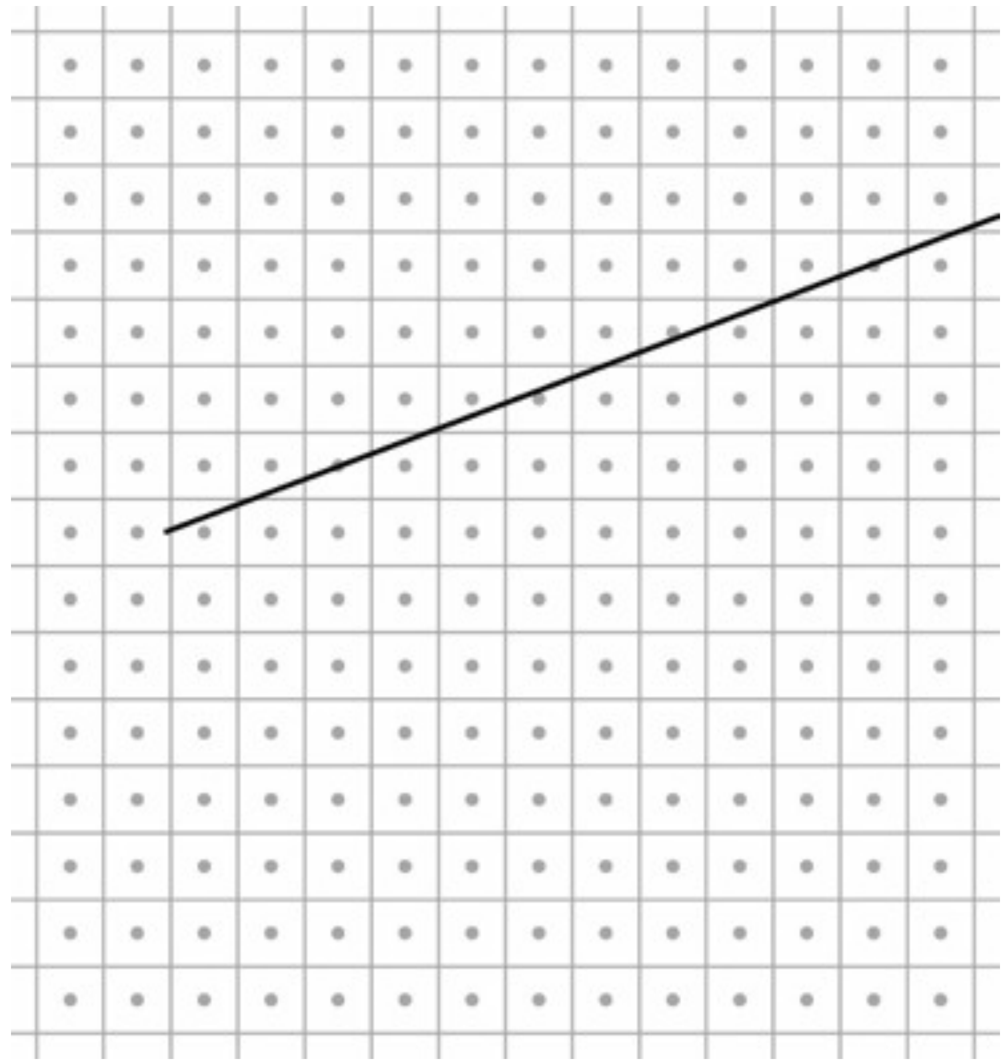


# Antialiasing

- A name for techniques to prevent aliasing
- In image generation, we need to *filter*
  - Boils down to averaging the image over an area
  - Weight by a filter
- Methods depend on source of image
  - Rasterization (lines and polygons)
  - Point sampling (e.g. raytracing)
  - Texture mapping

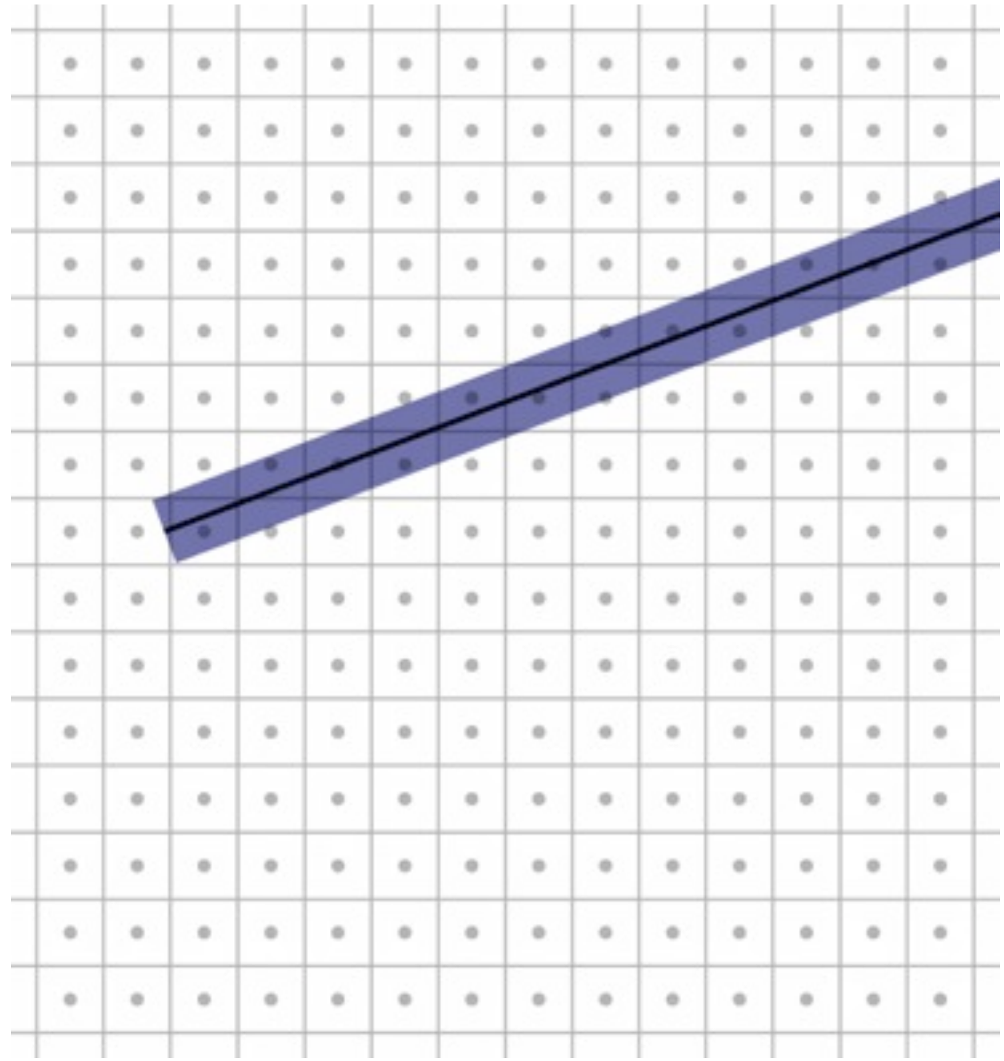
# Rasterizing lines

- Define line as a rectangle
- Specify by two endpoints
- Ideal image: black inside, white outside



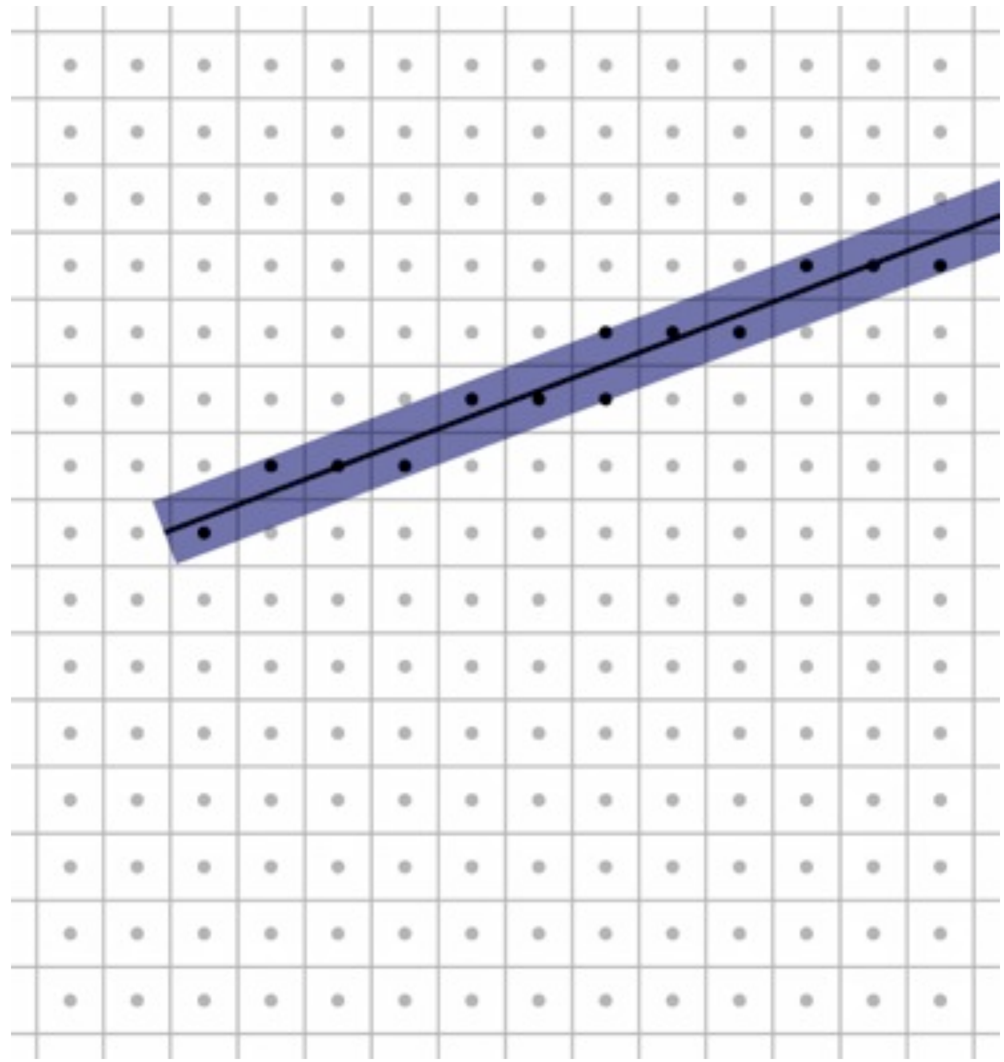
# Rasterizing lines

- Define line as a rectangle
- Specify by two endpoints
- Ideal image: black inside, white outside



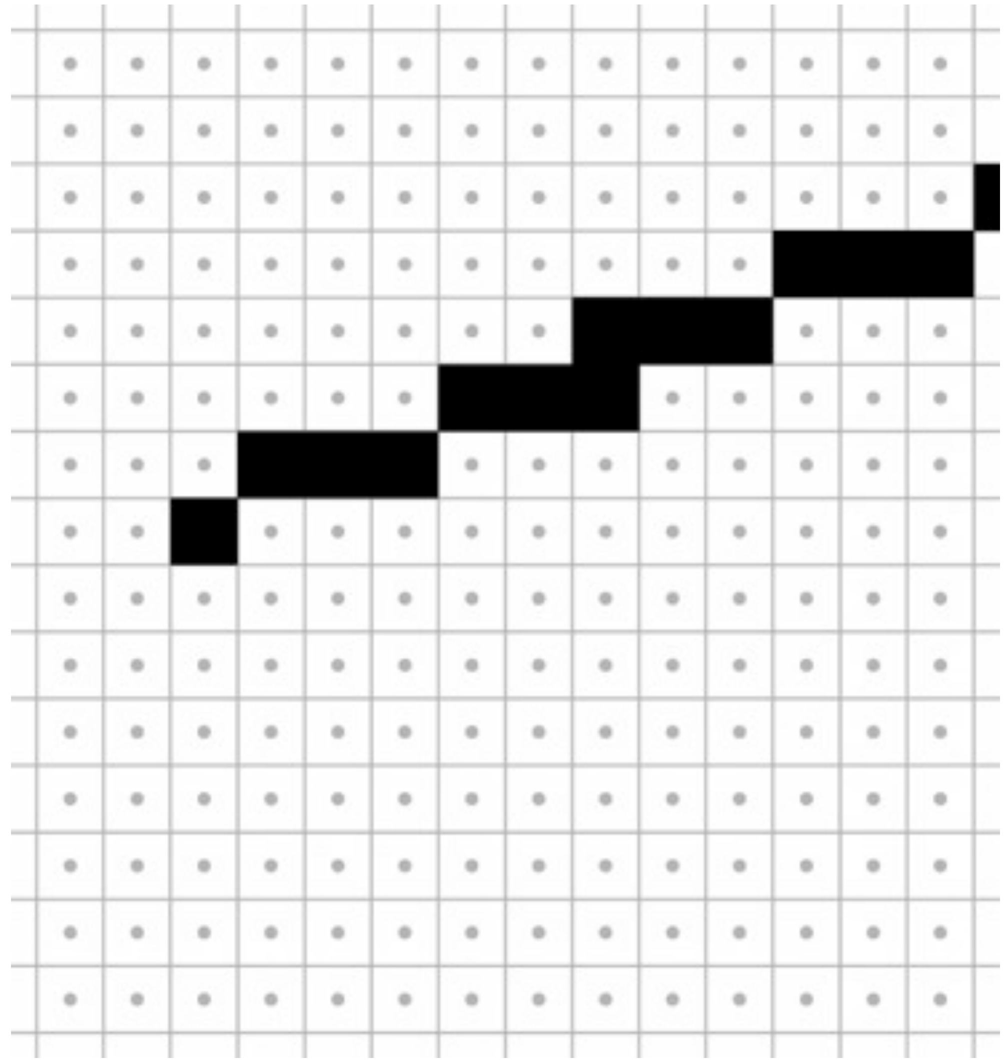
# Point sampling

- Approximate rectangle by drawing all pixels whose centers fall within the line
- Problem: all-or-nothing leads to jaggies
  - this is sampling with no filter (aka. point sampling)

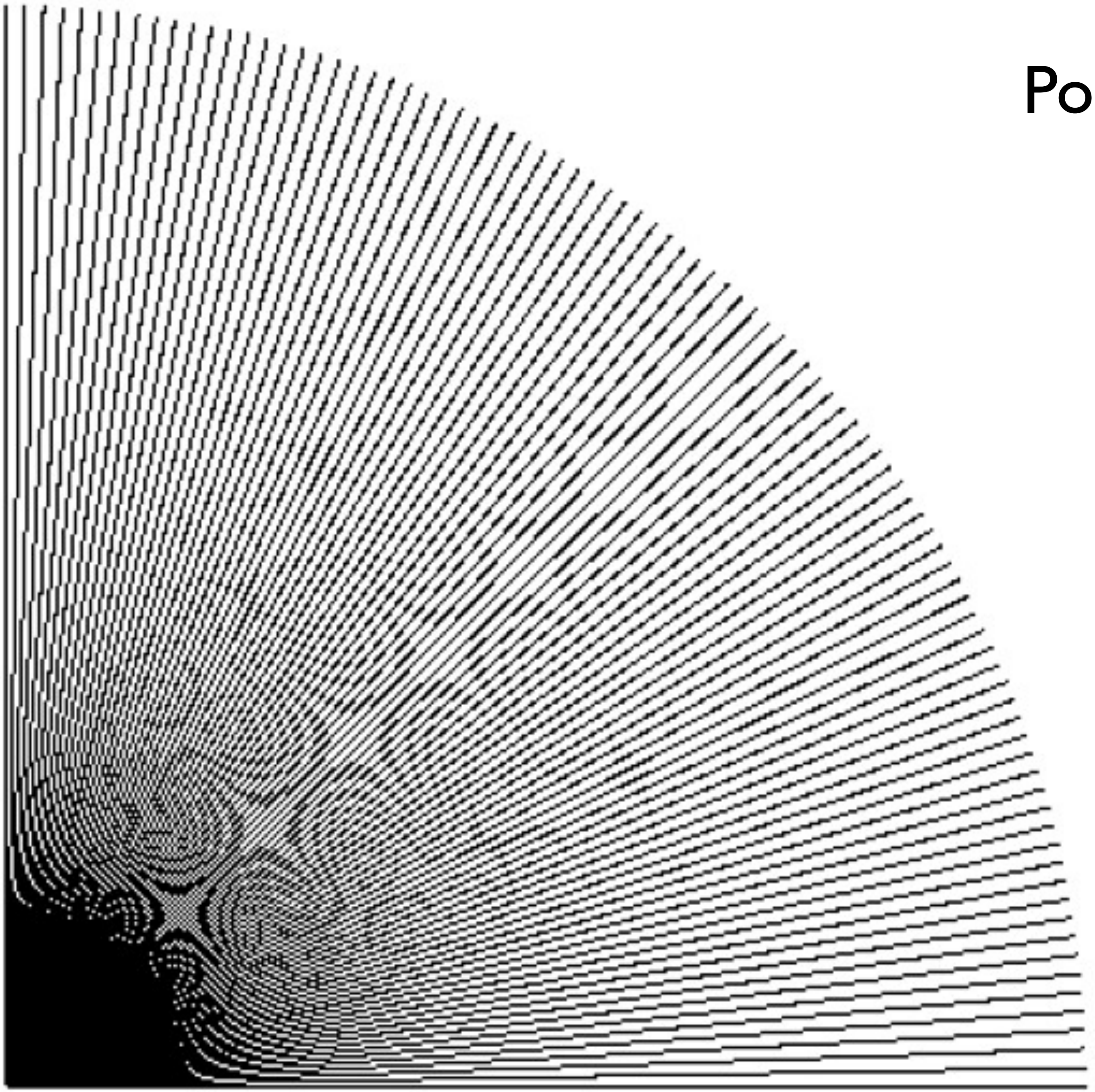


# Point sampling

- Approximate rectangle by drawing all pixels whose centers fall within the line
- Problem: all-or-nothing leads to jaggies
  - this is sampling with no filter (aka. point sampling)

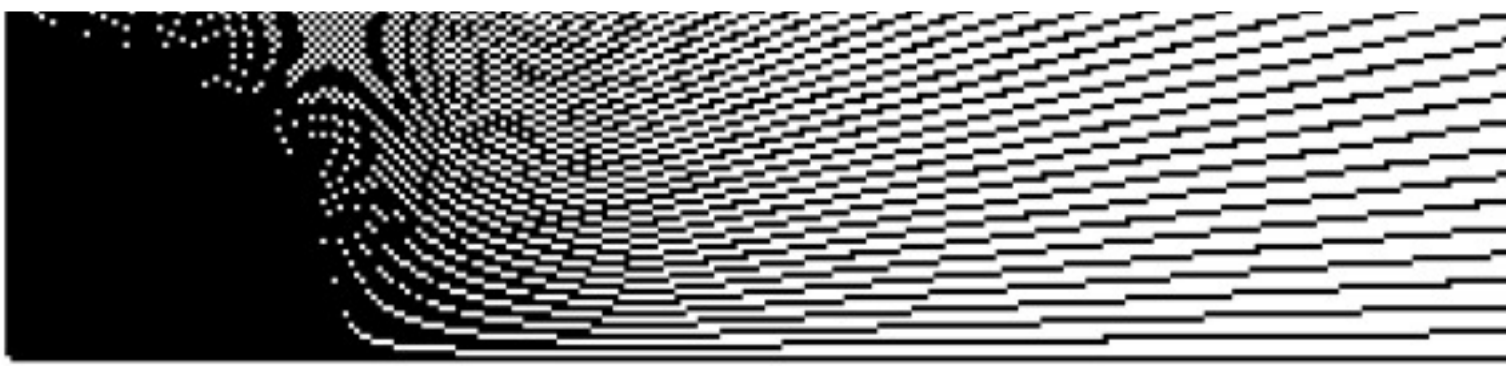


# Point sampling in action



# Aliasing

- Point sampling is fast and simple
- But the lines have stair steps and variations in width
- This is an aliasing phenomenon
  - Sharp edges of line contain high frequencies
- Introduces features to image that are not supposed to be there!

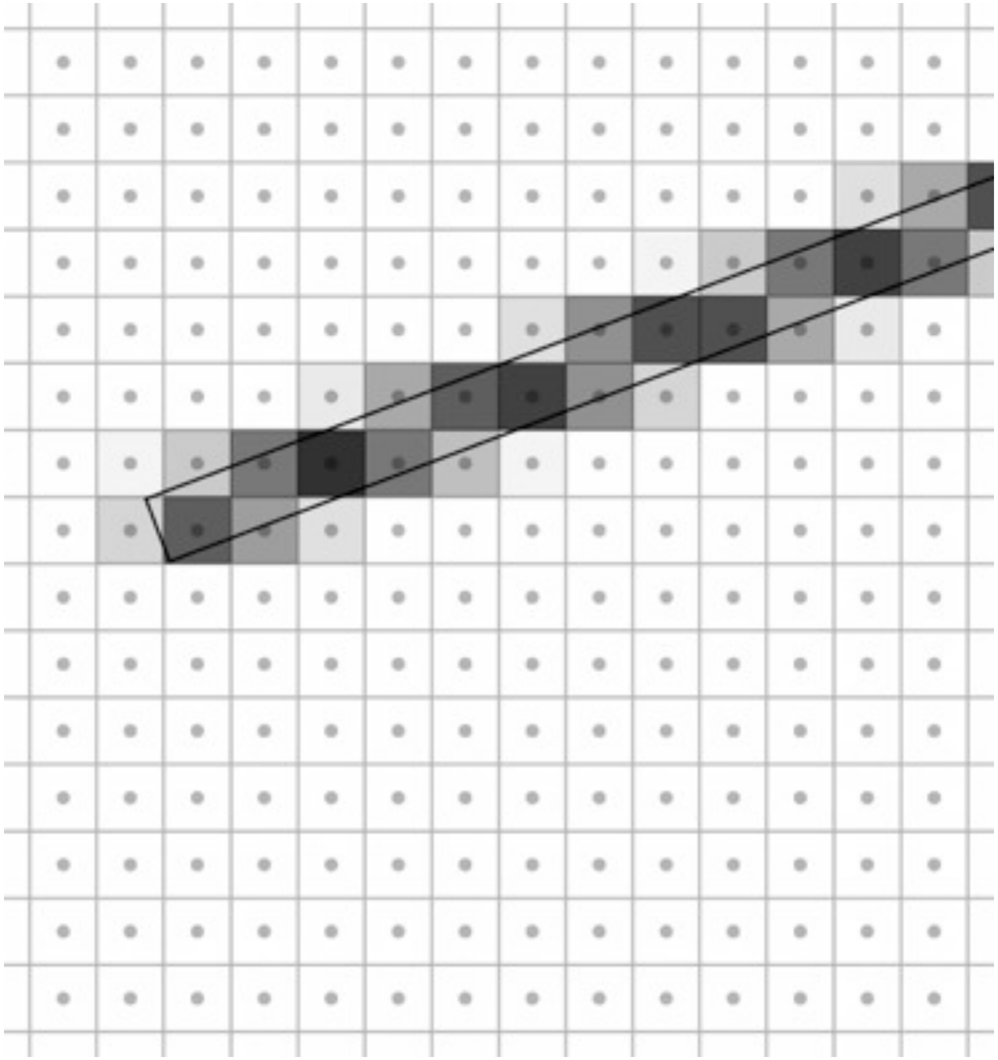


# Antialiasing

- Point sampling makes an all-or-nothing choice in each pixel
  - therefore steps are inevitable when the choice changes
  - yet another example where discontinuities are bad
- On bitmap devices this is necessary
  - hence high resolutions required
  - 600+ dpi in laser printers to make aliasing invisible
- On continuous-tone devices we can do better

# 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.

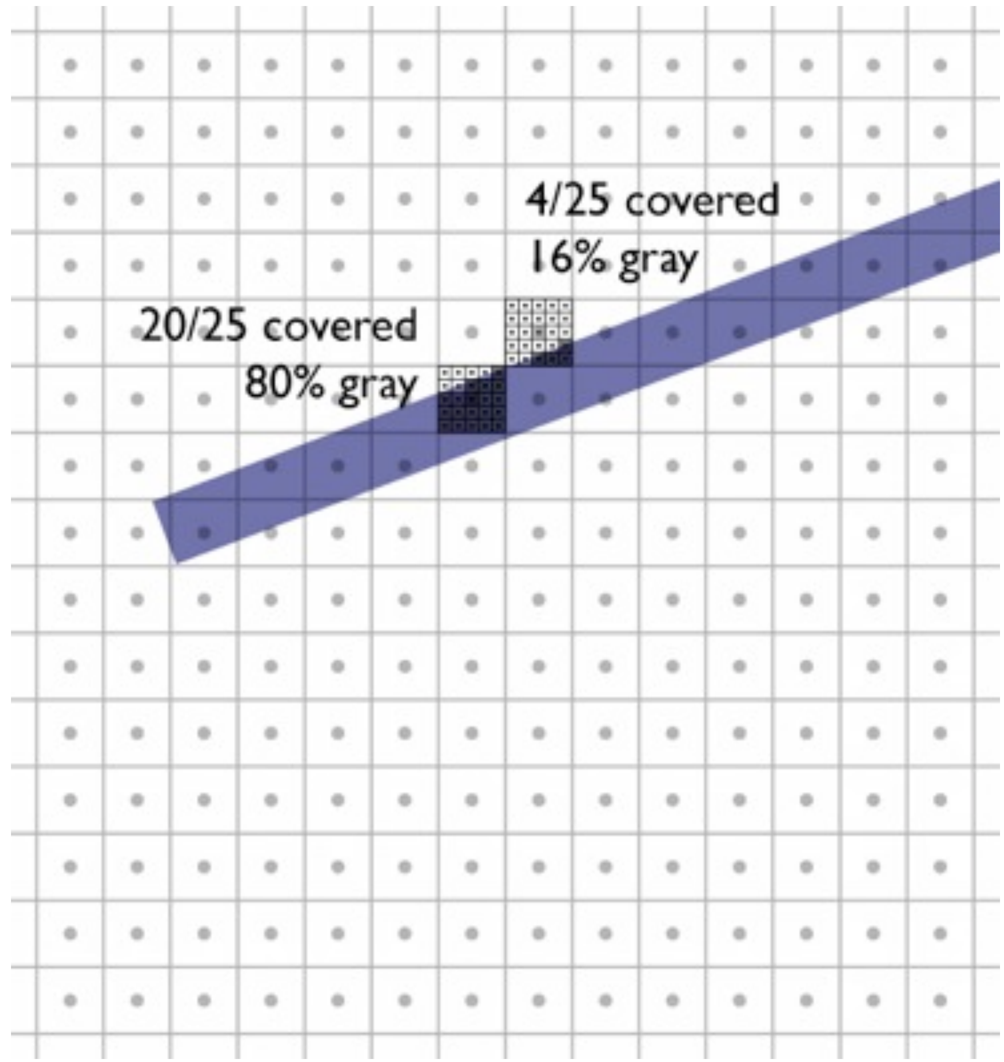


# Box filtering

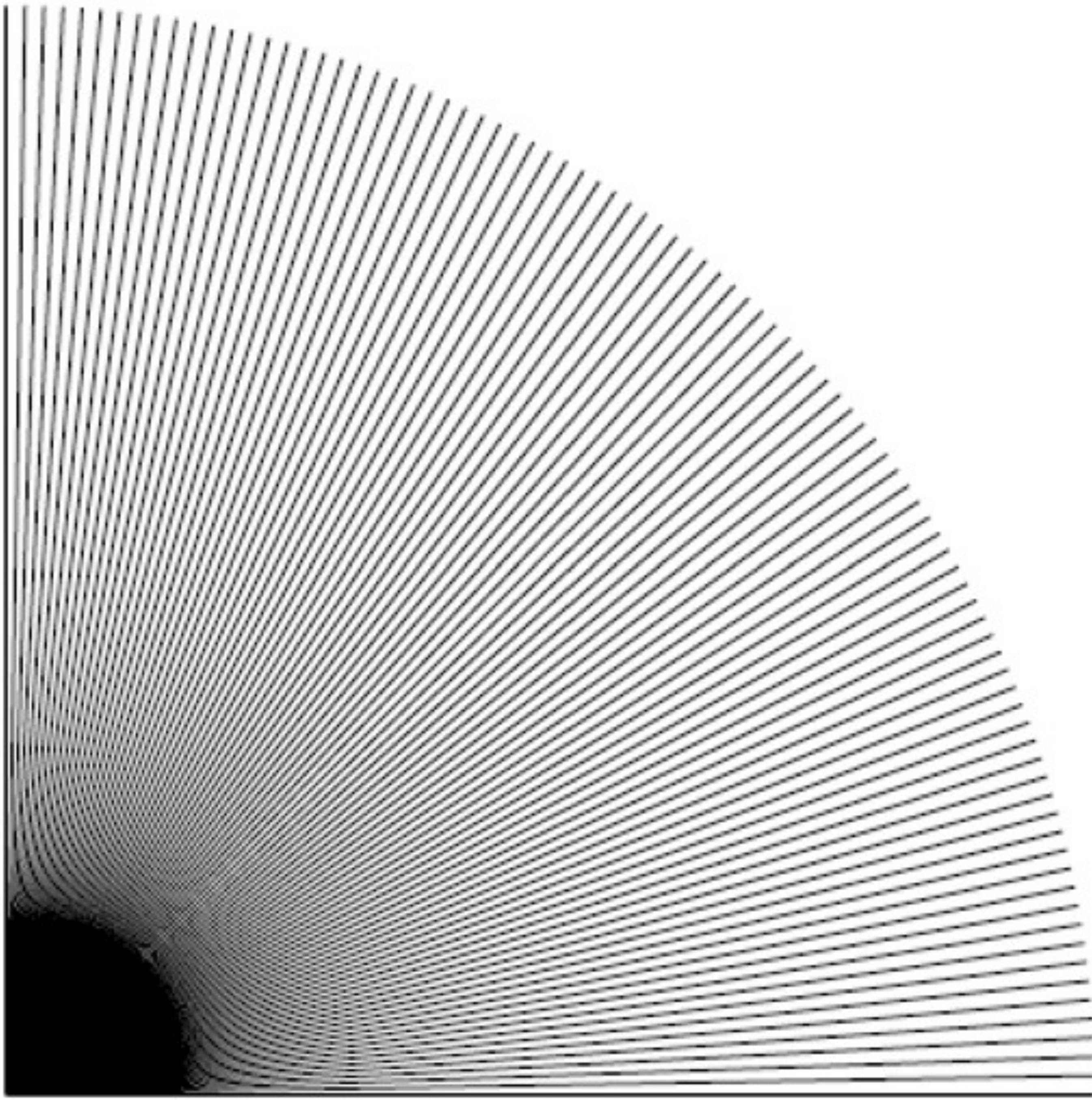
- Pixel intensity is proportional to area of overlap with square pixel area
- Also called "unweighted area averaging"

# Box filtering by supersampling

- Compute coverage fraction by counting subpixels
- Simple, accurate
- But slow



# Box filtering in action

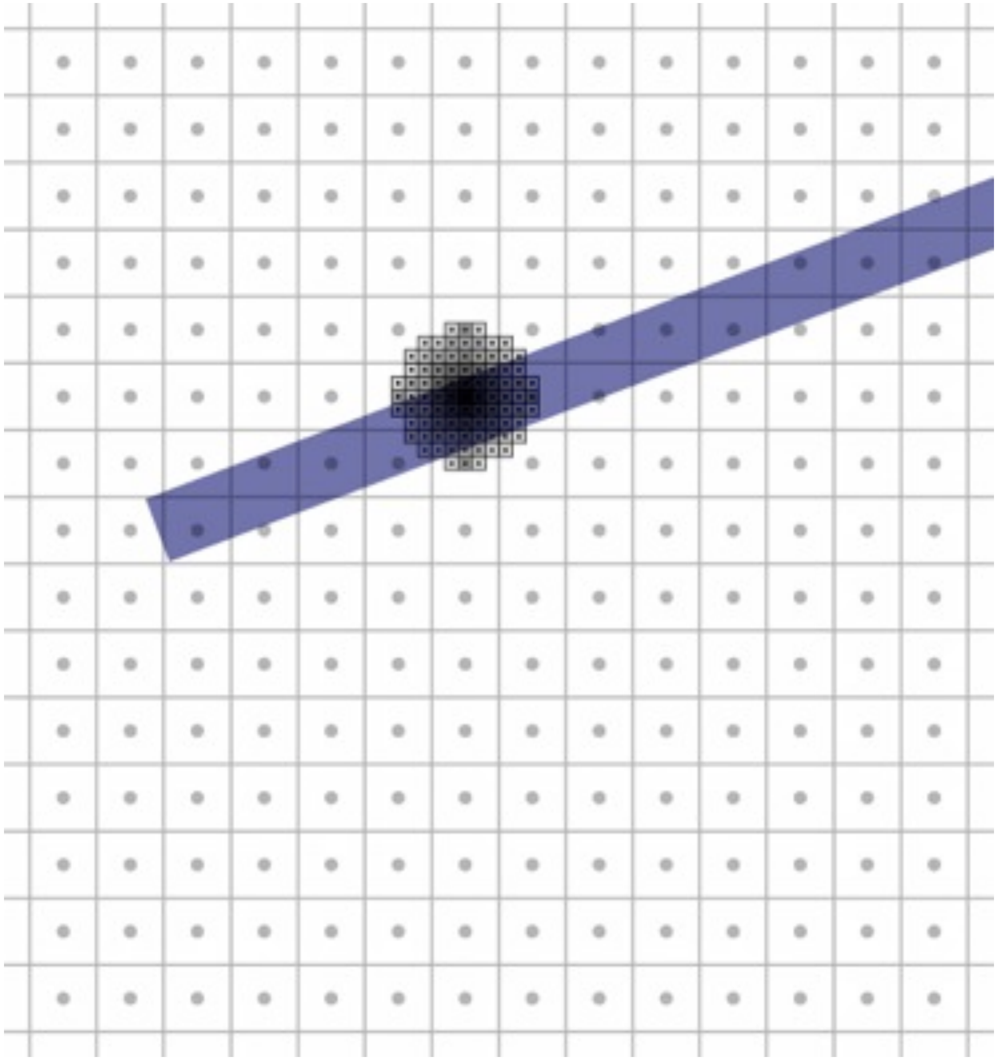


# Weighted filtering

- Box filtering problem: treats area near edge same as area near center
  - results in pixel turning on "too abruptly"
- Alternative: weight area by a smooth function
  - unweighted averaging corresponds to using a box function
  - a gaussian is a popular choice of smooth filter
  - important property: normalization (unit integral)

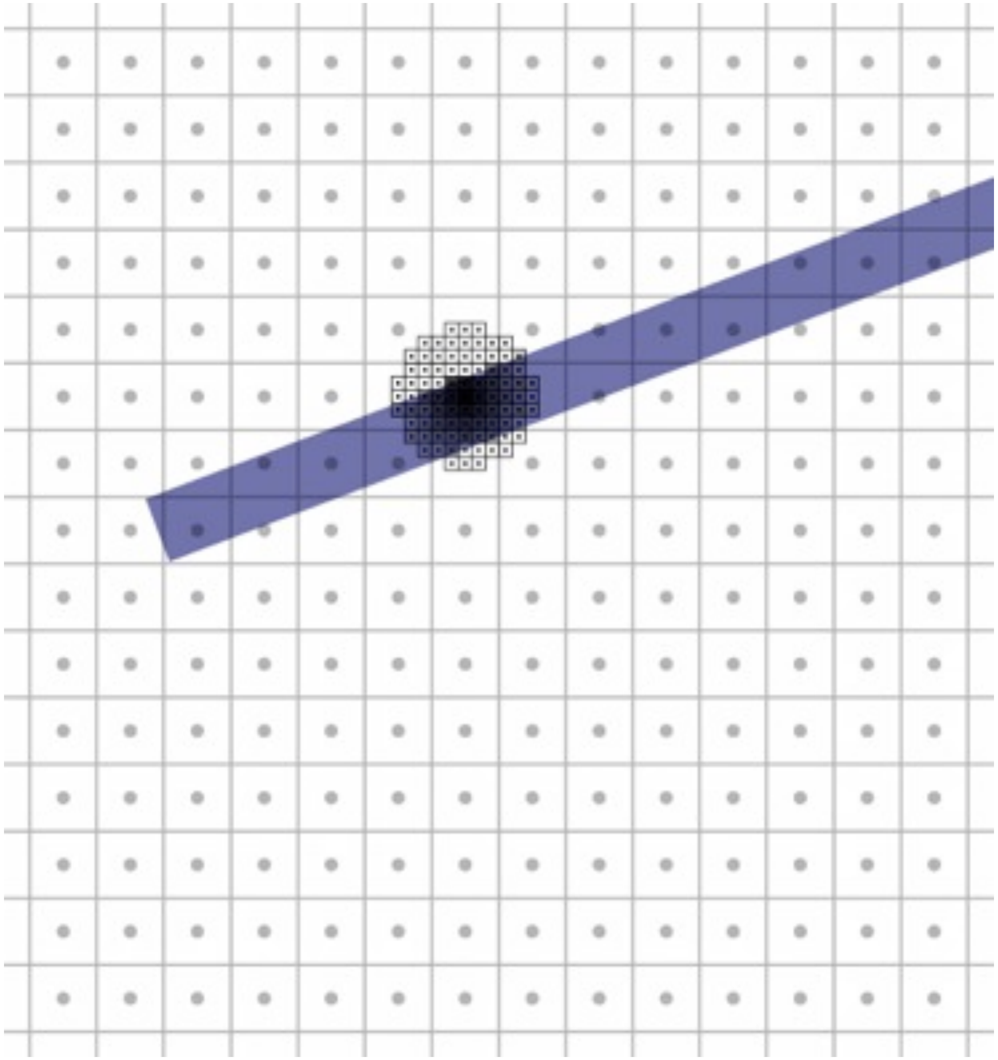
# Weighted filtering by supersampling

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



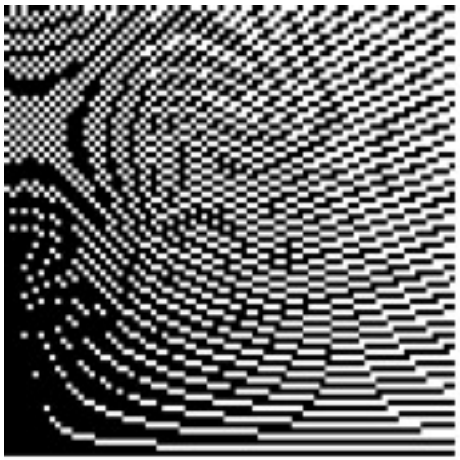
# Weighted filtering by supersampling

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

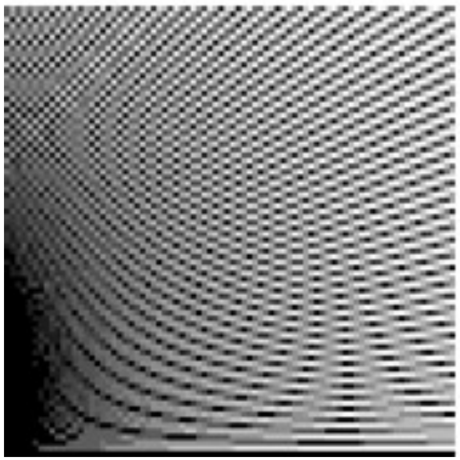


# Gaussian filtering in action

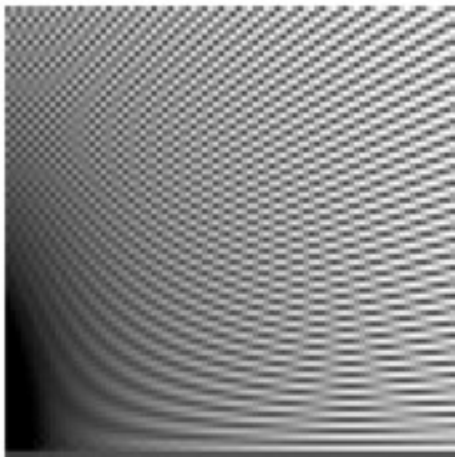
# Filter comparison



Point sampling

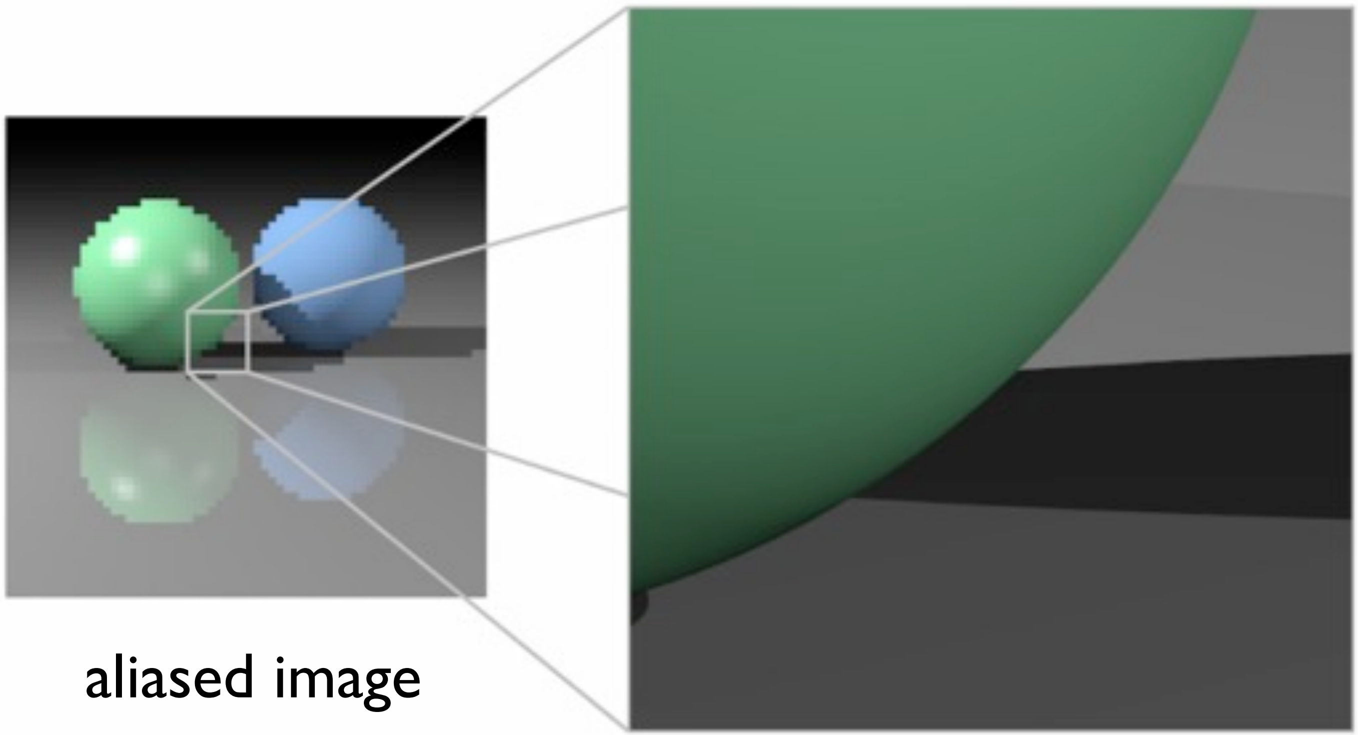


Box filtering



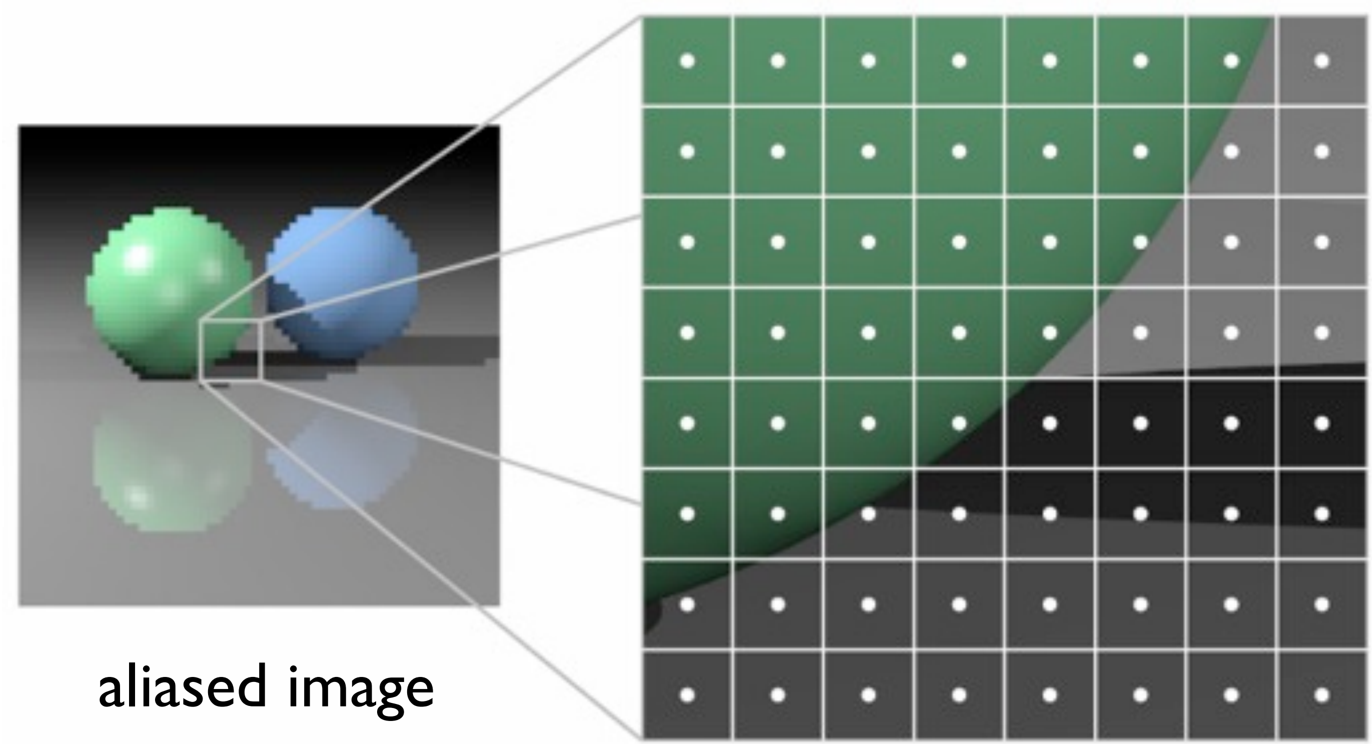
Gaussian filtering

# Antialiasing in ray tracing



aliased image

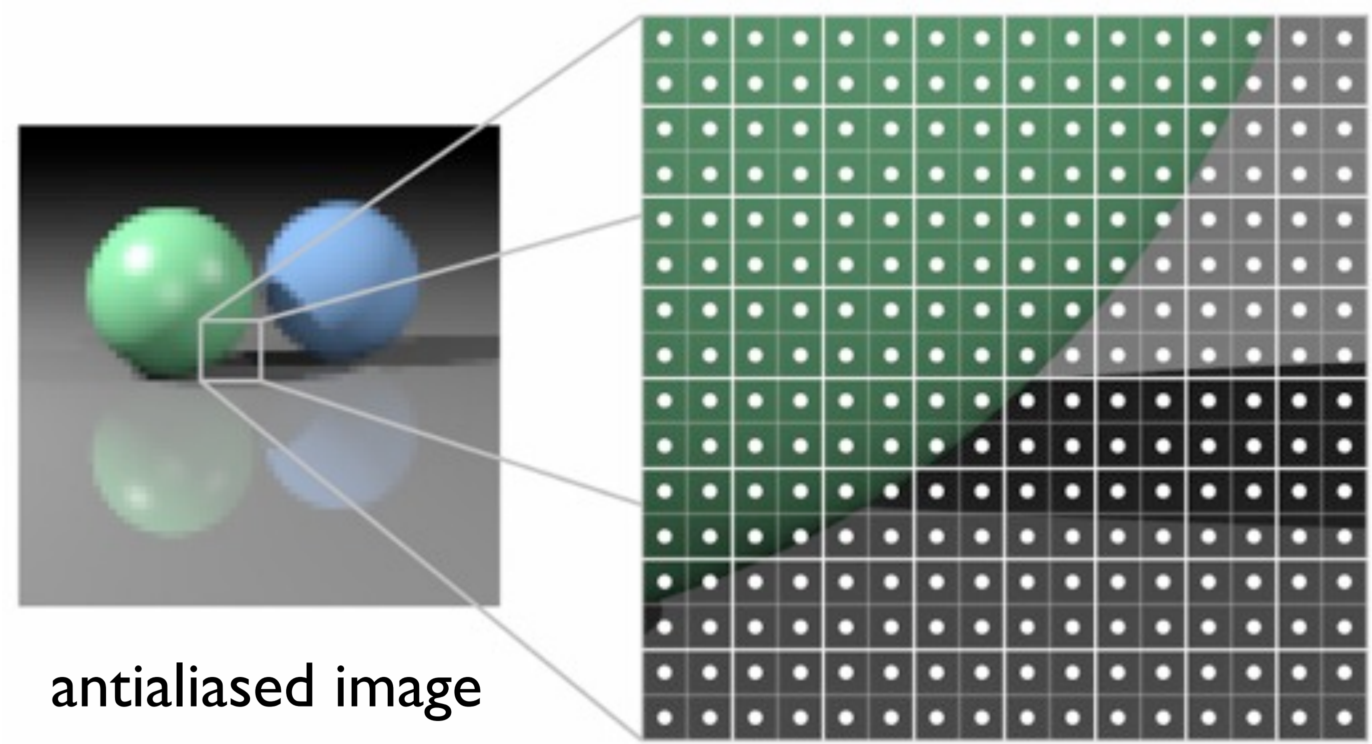
# Antialiasing in ray tracing



aliased image

one sample per pixel

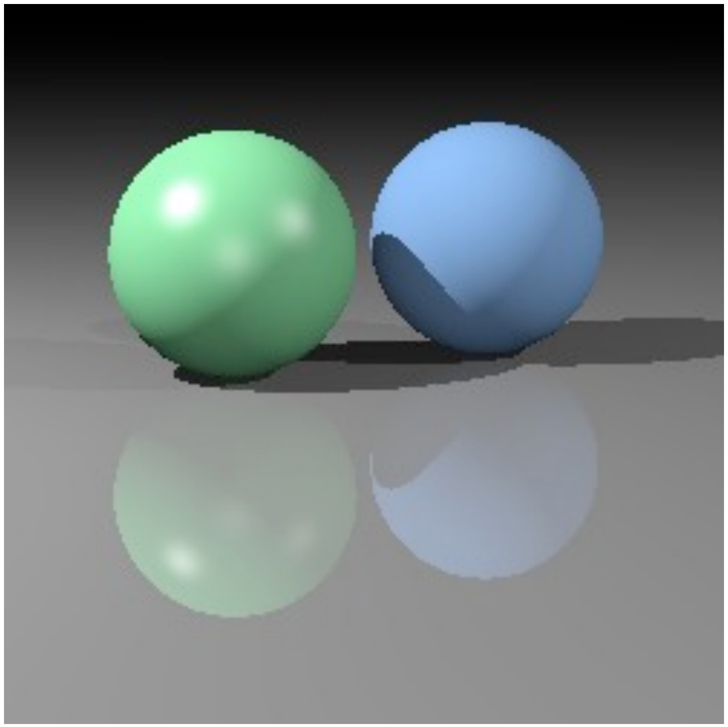
# Antialiasing in ray tracing



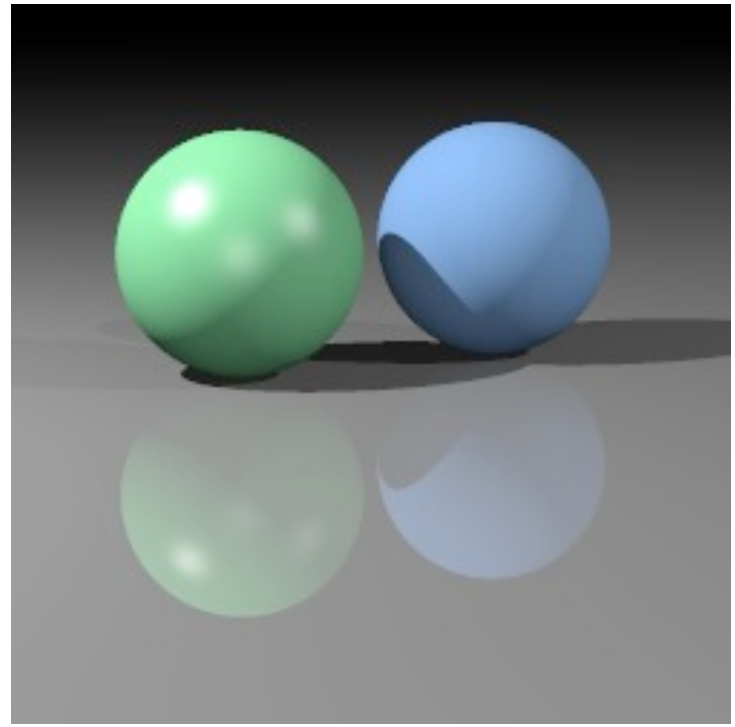
antialiased image

four samples per pixel

# Antialiasing in ray tracing



one sample/pixel

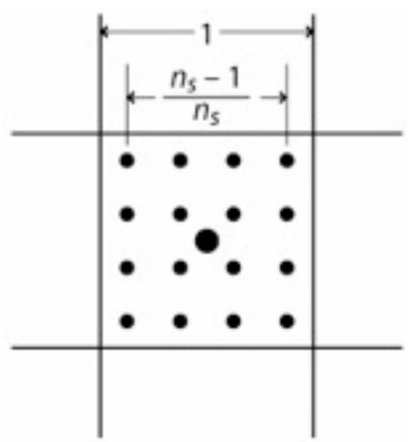


9 samples/pixel

# Details of supersampling

- For image coordinates with integer pixel centers:

```
// one sample per pixel
for iy = 0 to (ny–1) by 1
   for ix = 0 to (nx–1) by 1 {
      ray = camera.getRay(ix, iy);
      image.set(ix, iy, trace(ray));
   }
```




```
// ns^2 samples per pixel
for iy = 0 to (ny–1) by 1
   for ix = 0 to (nx–1) by 1 {
      Color sum = 0;
      for dx = –(ns–1)/2 to (ns–1)/2 by 1
         for dy = –(ns–1)/2 to (ns–1)/2 by
1 {
            x = ix + dx / ns;
            y = iy + dy / ns;
            ray = camera.getRay(x, y);
            sum += trace(ray);
         }
      image.set(ix, iy, sum / (ns*ns));
   }
```

# Details of supersampling

- For image coordinates in unit square

```
// one sample per pixel
for iy = 0 to (ny−1) by 1
   for ix = 0 to (nx−1) by 1 {
      double x = (ix + 0.5) / nx;
      double y = (iy + 0.5) / ny;
      ray = camera.getRay(x, y);
      image.set(ix, iy, trace(ray));
   }
```

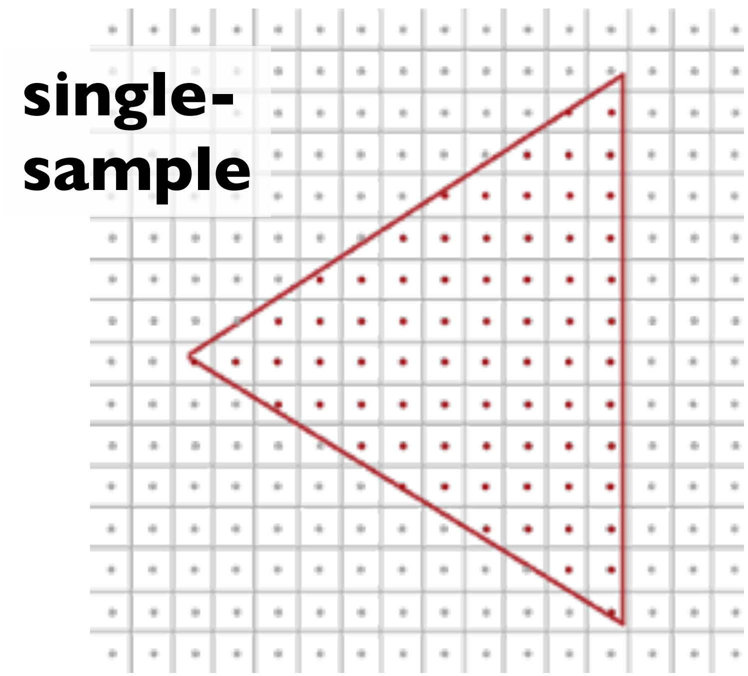
```
// ns^2 samples per pixel
for iy = 0 to (ny−1) by 1
   for ix = 0 to (nx−1) by 1 {
      Color sum = 0;
      for dx = 0 to (ns−1) by 1
         for dy = 0 to (ns−1) by 1 {
            x = (ix + (dx + 0.5) / ns) / nx;
            y = (iy + (dy + 0.5) / ns) / ny;
            ray = camera.getRay(x, y);
            sum += trace(ray);
         }
      image.set(ix, iy, sum / (ns*ns));
   }
```

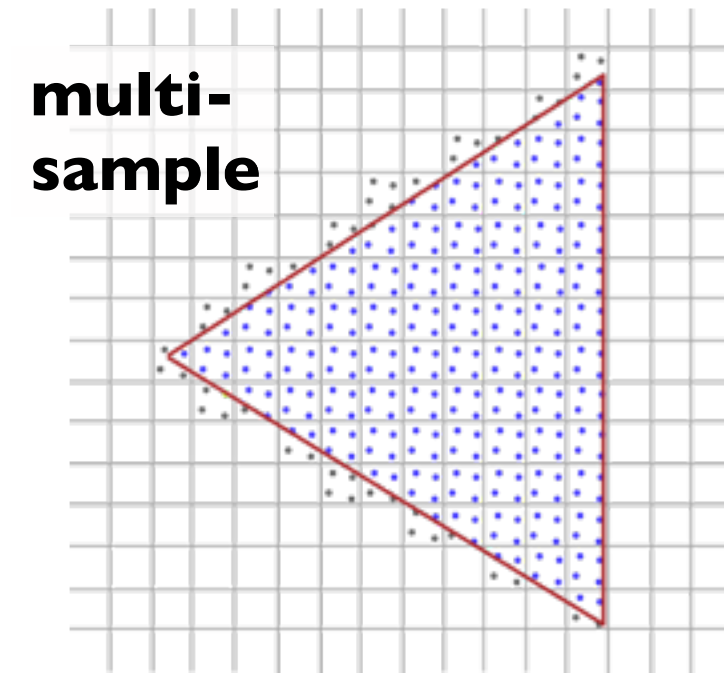
# Supersampling vs. multisampling

- Supersampling is terribly expensive
- GPUs use an approximation called *multisampling*
  - Compute one shading value per pixel
  - Store it at many subpixel samples, each with its own depth

# Multisample rasterization

- Each fragment carries several (color,depth) samples
  - shading is computed per-fragment
  - depth test is resolved per-sample
  - final color is average of sample colors

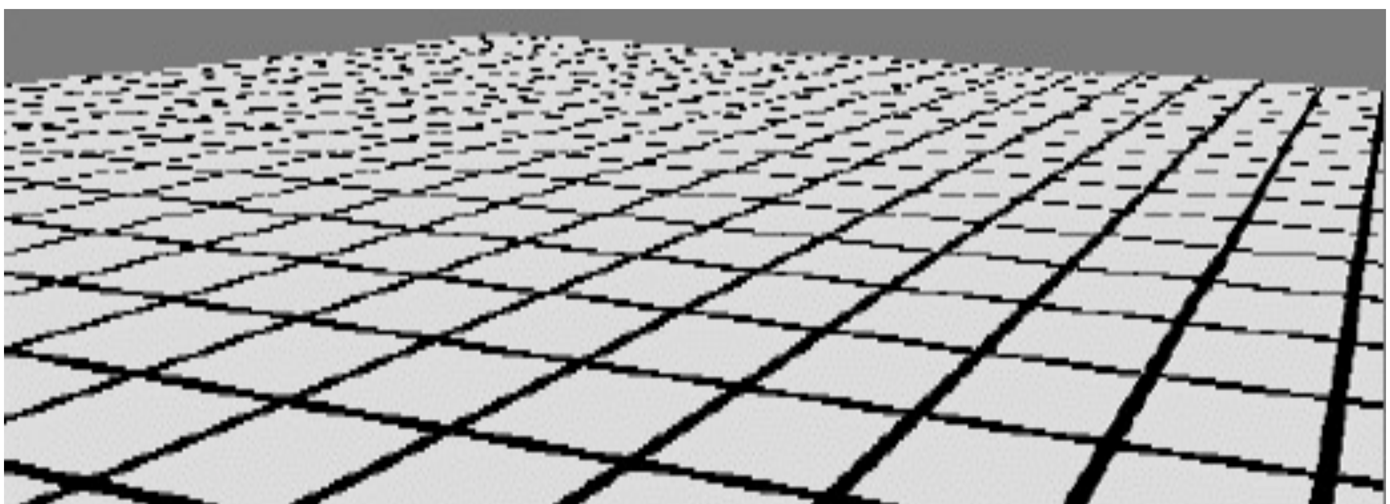



[http://www.learnopengl.com]

# Antialiasing in textures

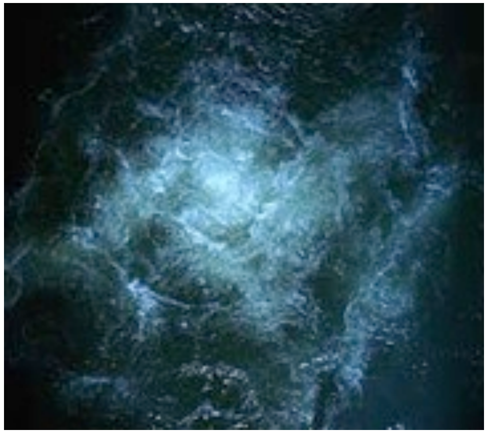
- Even with multisampling, we still only evaluate textures once per fragment
- Need to filter the texture somehow!
  - perspective produces very high image frequencies
  - (will return to this topic later, time permitting)

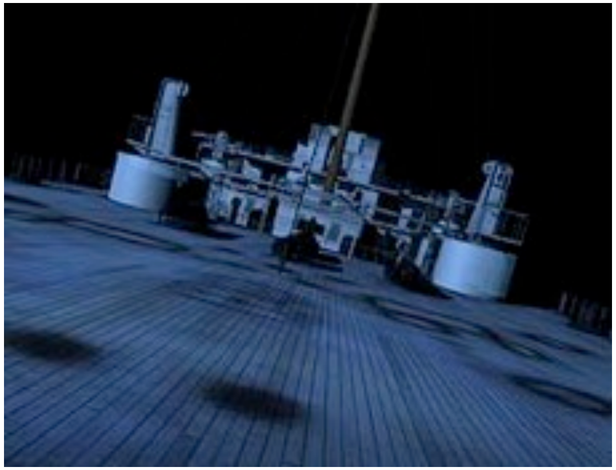


# Compositing

# Compositing





[*Titanic* ; DigitalDomain; vfxhq.com]

# Combining images

- Often useful combine elements of several images
- Trivial example: video crossfade
  - smooth transition from one scene to another

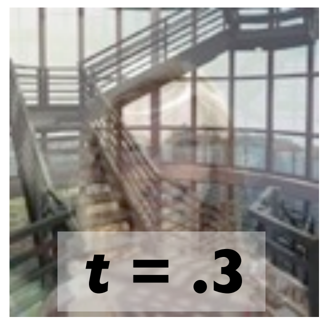

$$r_C = t r_A + (1-t) r_B$$
$$g_C = t g_A + (1-t) g_B$$
$$b_C = t b_A + (1-t) b_B$$

  - note: weights sum to 1.0
    - no unexpected brightening or darkening
    - no out-of-range results
  - this is *linear interpolation*

# Combining images

- Often useful combine elements of several images
- Trivial example: video crossfade
  - smooth transition from one scene to another






$$r_C = t r_A + (1 - t) r_B$$
$$g_C = t g_A + (1 - t) g_B$$
$$b_C = t b_A + (1 - t) b_B$$

  - note: weights sum to 1.0
    - no unexpected brightening or darkening
    - no out-of-range results
  - this is *linear interpolation*

# Combining images

- Often useful combine elements of several images
- Trivial example: video crossfade
  - smooth transition from one scene to another

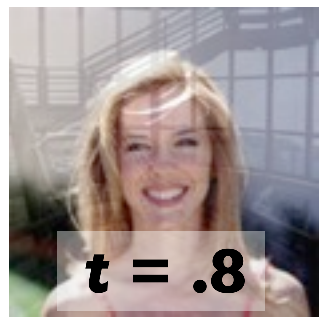

$$r_C = t r_A + (1 - t) r_B$$
$$g_C = t g_A + (1 - t) g_B$$
$$b_C = t b_A + (1 - t) b_B$$

  - note: weights sum to 1.0
    - no unexpected brightening or darkening
    - no out-of-range results
  - this is *linear interpolation*

# Combining images

- Often useful combine elements of several images
- Trivial example: video crossfade
  - smooth transition from one scene to another



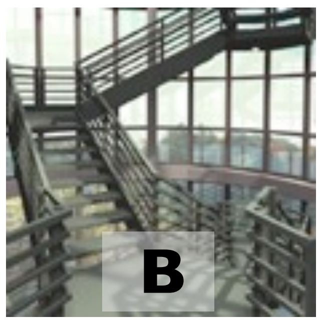


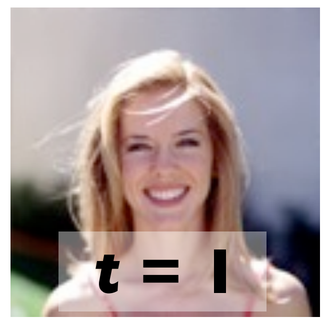

$$r_C = t r_A + (1-t) r_B$$
$$g_C = t g_A + (1-t) g_B$$
$$b_C = t b_A + (1-t) b_B$$

  - note: weights sum to 1.0
    - no unexpected brightening or darkening
    - no out-of-range results
  - this is *linear interpolation*

# Combining images

- Often useful combine elements of several images
- Trivial example: video crossfade
  - smooth transition from one scene to another


$$r_C = t r_A + (1 - t) r_B$$
$$g_C = t g_A + (1 - t) g_B$$
$$b_C = t b_A + (1 - t) b_B$$

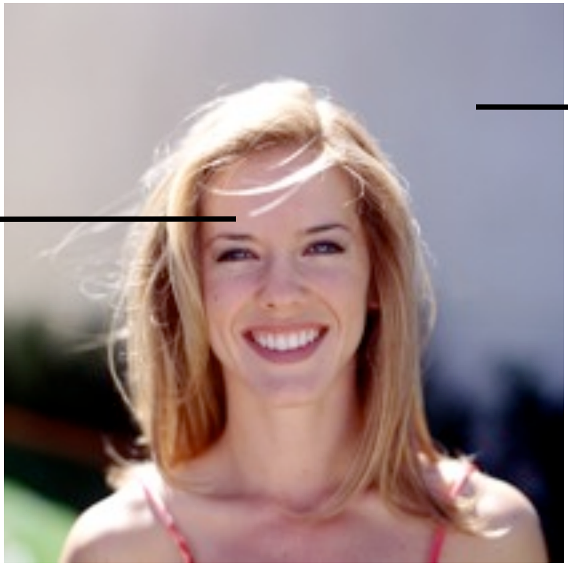
  - note: weights sum to 1.0
    - no unexpected brightening or darkening
    - no out-of-range results
  - this is *linear interpolation*

# Foreground and background

- In many cases just adding is not enough
- Example: compositing in film production
  - shoot foreground and background separately
  - also include CG elements
  - this kind of thing has been done in analog for decades
  - how should we do it digitally?

# Foreground and background

- How we compute new image varies with position




[Chuang et al. / Corel]

- Therefore, need to store some kind of tag to say what parts of the image are of interest

# Binary image mask

- First idea: store one bit per pixel
  - answers question “is this pixel part of the foreground?”

[Chuang et al. / Corel]

  - causes jaggies similar to point-sampled rasterization
  - same problem, same solution: intermediate values

# Binary image mask

- First idea: store one bit per pixel
  - answers question "is this pixel part of the foreground?"

[Chuang et al. / Corel]

  - causes jaggies similar to point-sampled rasterization
  - same problem, same solution: intermediate values

# Binary image mask

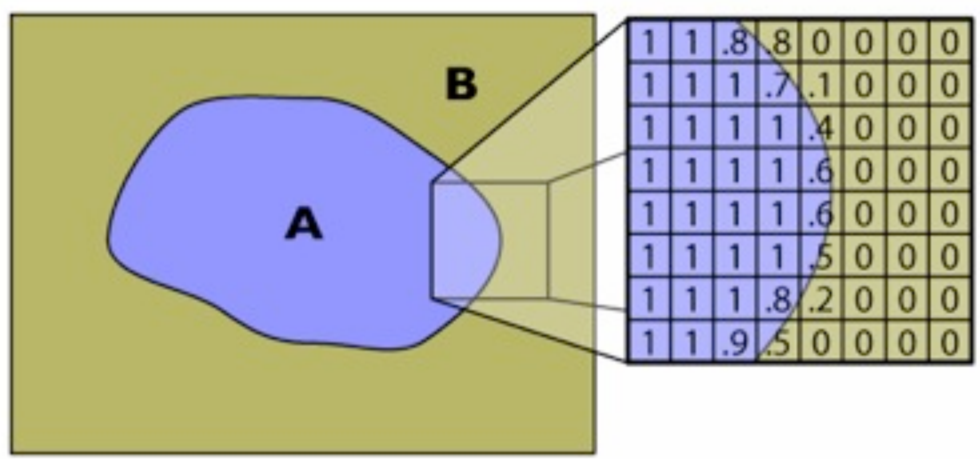
- First idea: store one bit per pixel
  - answers question “is this pixel part of the foreground?”

[Chuang et al. / Corel]

  - causes jaggies similar to point-sampled rasterization
  - same problem, same solution: intermediate values

# Partial pixel coverage

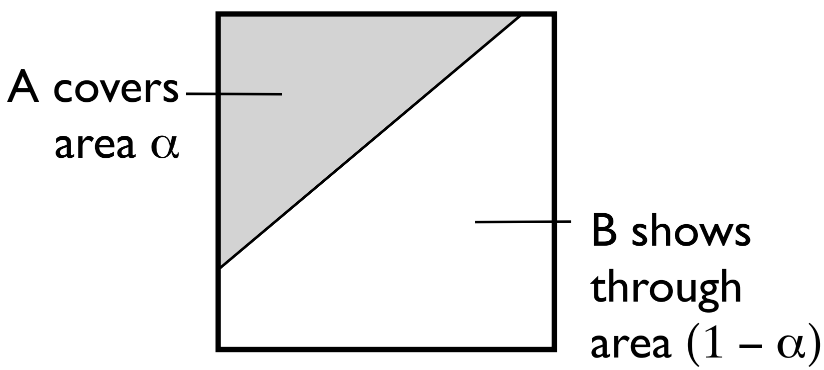
- The problem: pixels near boundary are not strictly foreground or background




  - how to represent this simply?
  - interpolate boundary pixels between the fg. and bg. colors

# Alpha compositing

- Formalized in 1984 by Porter & Duff
- Store fraction of pixel covered, called $\alpha$




$$E = A \textbf{ over } B$$

$$r_E = \alpha_A r_A + (1 - \alpha_A) r_B$$

$$g_E = \alpha_A g_A + (1 - \alpha_A) g_B$$

$$b_E = \alpha_A b_A + (1 - \alpha_A) b_B$$

  – this exactly like a spatially varying crossfade

- Convenient implementation
  – 8 more bits makes 32
  – 2 multiplies + 1 add per pixel for compositing

# Alpha compositing—example

[Chuang et al. / Corel]

# Alpha compositing—example

[Chuang et al. / Corel]

# Alpha compositing—example

[Chuang et al. / Corel]

# Compositing composites

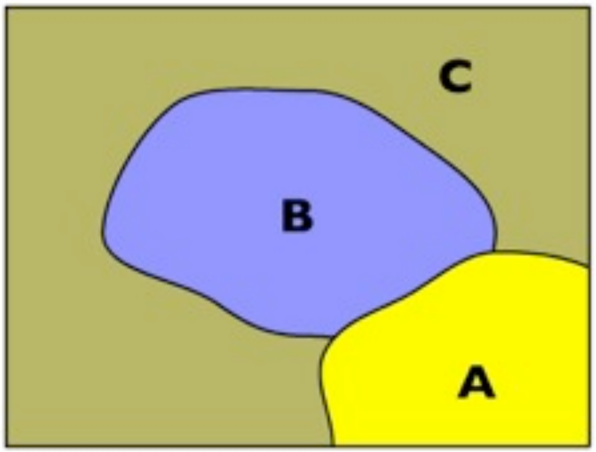
- so far have only considered single fg. over single bg.
- in real applications we have *n* layers
  - *Titanic* example
  - compositing foregrounds to create new foregrounds
    - what to do with $\alpha$?
- desirable property: associativity

$$A \textbf{ over } (B \textbf{ over } C) = (A \textbf{ over } B) \textbf{ over } C$$

  - to make this work we need to be careful about how $\alpha$ is computed

# Compositing composites

- Some pixels are partly covered in more than one layer

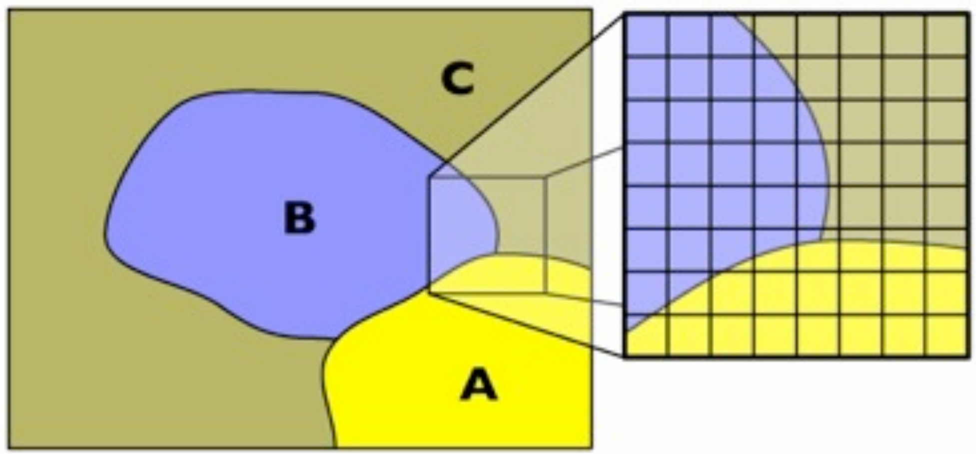



  - in D = A **over** (B **over** C) what will be the result?

$$
\begin{aligned}
c_D &= \alpha_A c_A + (1-\alpha_A)[\alpha_B c_B + (1-\alpha_B) c_C] \\
&= \alpha_A c_A + (1-\alpha_A)\alpha_B c_B + (1-\alpha_A)(1-\alpha_B) c_C
\end{aligned}
$$

# Compositing composites

- Some pixels are partly covered in more than one layer

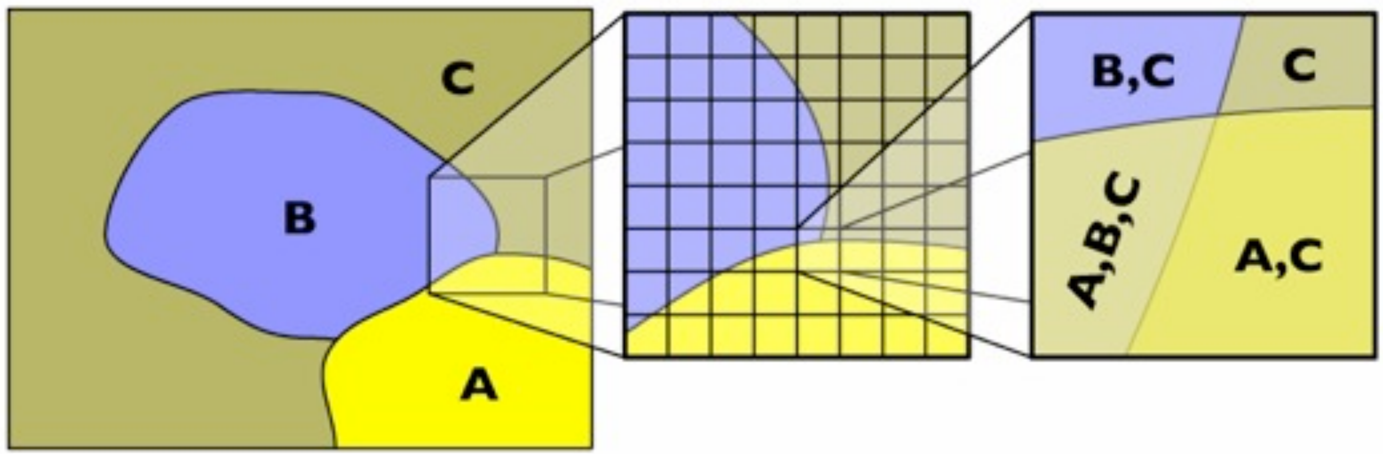



  – in D = A **over** (B **over** C) what will be the result?

$$c_D = \alpha_A c_A + (1 - \alpha_A)[\alpha_B c_B + (1 - \alpha_B) c_C]$$
$$= \alpha_A c_A + (1 - \alpha_A)\alpha_B c_B + (1 - \alpha_A)(1 - \alpha_B) c_C$$

# Compositing composites

- Some pixels are partly covered in more than one layer

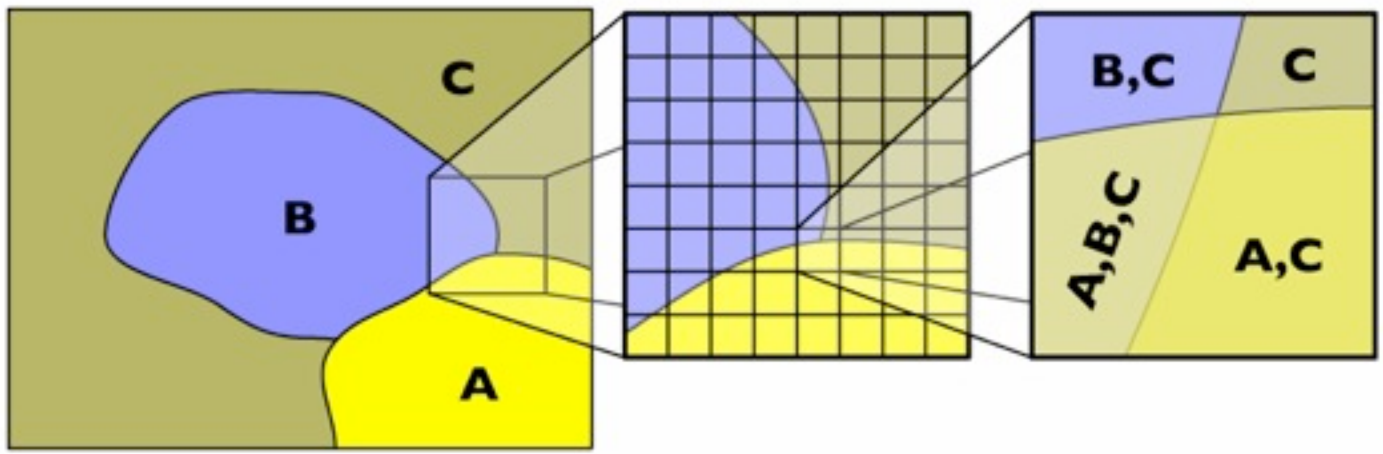



  - in D = A **over** (B **over** C) what will be the result?

$$c_D = \alpha_A c_A + (1 - \alpha_A)[\alpha_B c_B + (1 - \alpha_B) c_C]$$
$$= \alpha_A c_A + (1 - \alpha_A)\alpha_B c_B + (1 - \alpha_A)(1 - \alpha_B) c_C$$

# Compositing composites

- Some pixels are partly covered in more than one layer




  – in D = A **over** (B **over** C) what will be the result?

$$c_D = \alpha_A c_A + (1 - \alpha_A)[\alpha_B c_B + (1 - \alpha_B) c_C]$$
$$= \alpha_A c_A + (1 - \alpha_A)\alpha_B c_B + (1 - \alpha_A)(1 - \alpha_B) c_C$$

Fraction covered by neither A nor B

# Associativity?

- What does this imply about (A **over** B)?
  - Coverage has to be

$$\begin{aligned}\alpha_{(A\ \mathbf{over}\ B)} &= 1 - (1-\alpha_A)(1-\alpha_B) \\ &= \alpha_A + (1-\alpha_A)\alpha_B\end{aligned}$$

  - …but the color values then don't come out nicely in D = (A **over** B) **over** C:

$$\begin{aligned}c_D &= \alpha_A c_A + (1-\alpha_A)\alpha_B c_B + (1-\alpha_A)(1-\alpha_B)c_C \\ &= \alpha_{(A\ \mathbf{over}\ B)}(\cdots) + (1-\alpha_{(A\ \mathbf{over}\ B)})c_C\end{aligned}$$

# An optimization

- Compositing equation again

$$c_E = \alpha_A c_A + (1 - \alpha_A) c_B$$

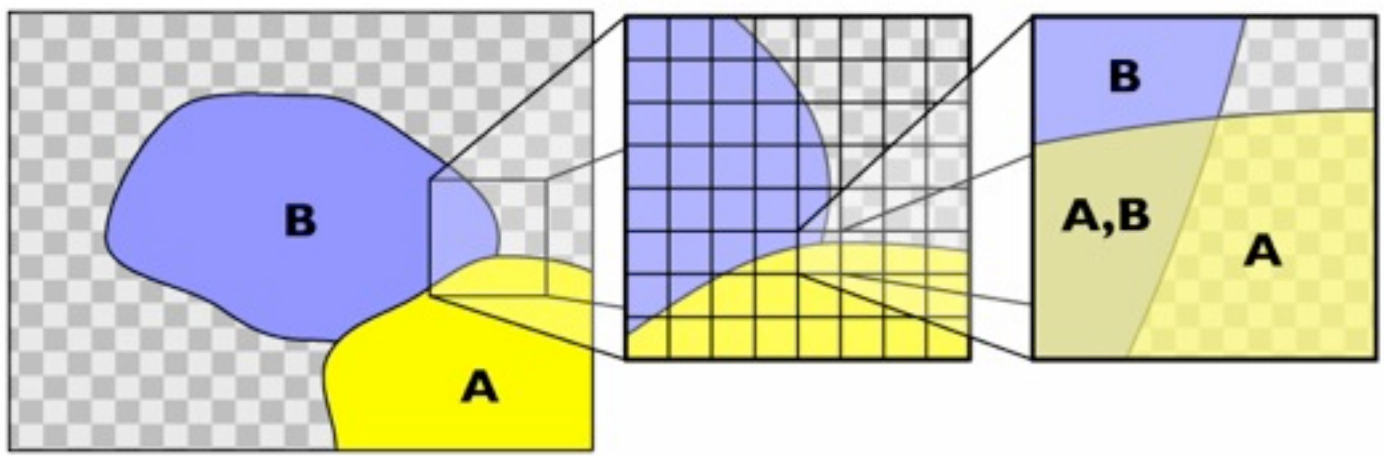
- Note $c_A$ appears only in the product $\alpha_A c_A$
  - so why not do the multiplication ahead of time?
- Leads to *premultiplied alpha*:
  - store pixel value ($r'$, $g'$, $b'$, $\alpha$) where $c' = \alpha c$
  - E = A **over** B becomes

$$c'_E = c'_A + (1 - \alpha_A) c'_B$$

  - this turns out to be more than an optimization…
  - hint: so far the background has been opaque!

# Compositing composites

- What about just E = A **over** B (with B transparent)?

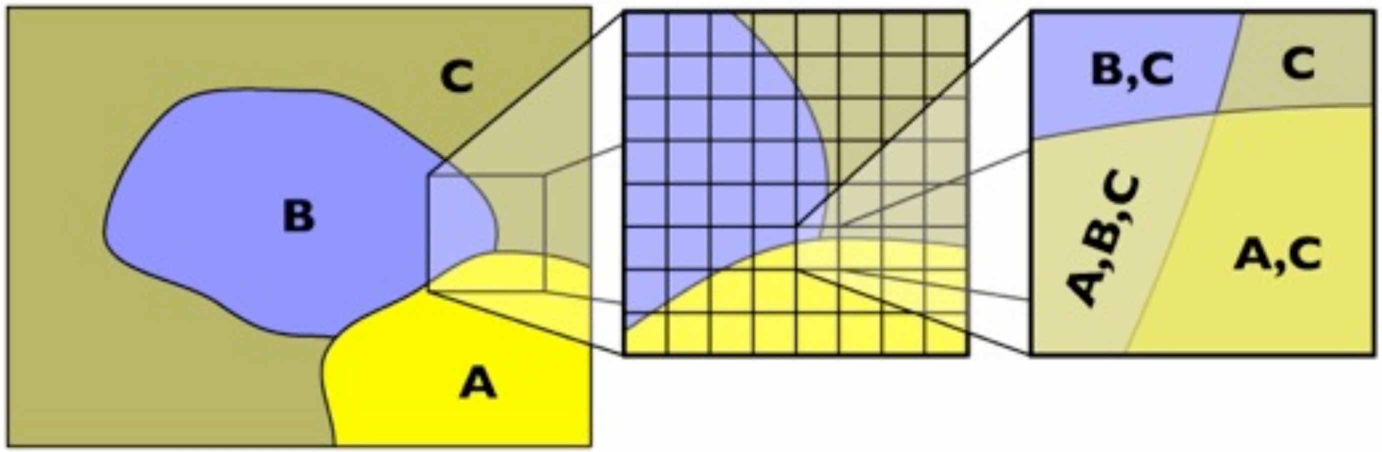



  – in premultiplied alpha, the result

$$\alpha_E = \alpha_A + (1 - \alpha_A)\alpha_B$$

  looks just like blending colors, and it leads to associativity.

# Associativity!

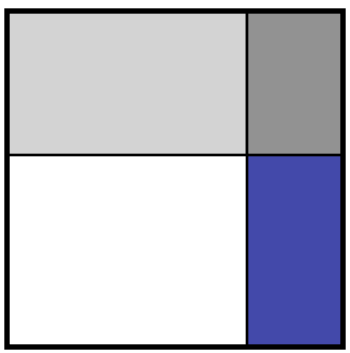



$$\begin{aligned} c_D &= c'_A + (1-\alpha_A)[c'_B + (1-\alpha_B)c'_C] \\ &= [c'_A + (1-\alpha_A)c'_B] + (1-\alpha_A)(1-\alpha_B)c'_C \\ &= c'_{(A\ \textbf{over}\ B)} + (1-\alpha_{(A\ \textbf{over}\ B)})c'_C \end{aligned}$$

– This is another good reason to premultiply

# Independent coverage assumption

- Why is it reasonable to blend $\alpha$ like a color?
- Simplifying assumption: covered areas are independent
  - that is, uncorrelated in the statistical sense

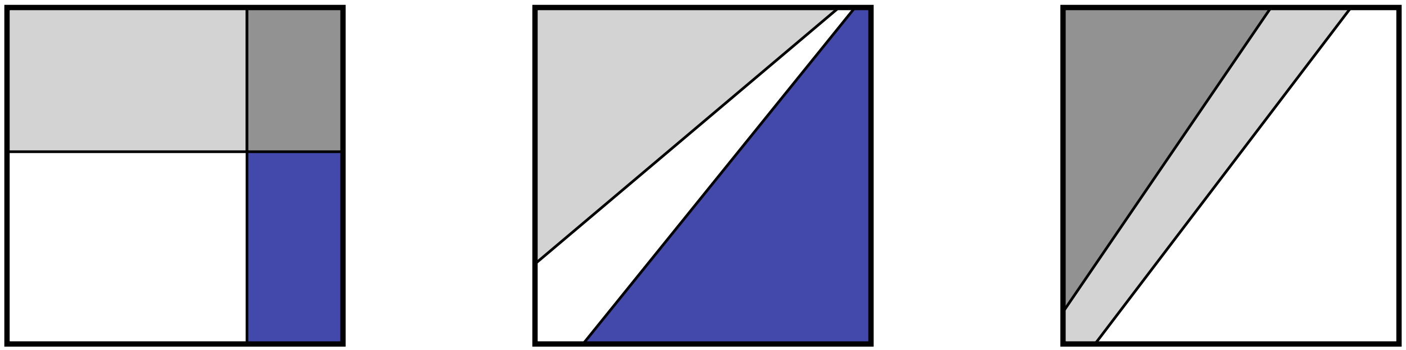


| *description* | *area* |
|---|---|
| $\bar{A} \cap \bar{B}$ | $(1-\alpha_A)(1-\alpha_B)$ |
| $A \cap \bar{B}$ | $\alpha_A(1-\alpha_B)$ |
| $\bar{A} \cap B$ | $(1-\alpha_A)\alpha_B$ |
| $A \cap B$ | $\alpha_A \alpha_B$ |

[Porter & Duff 84]

# Independent coverage assumption

- Holds in most but not all cases



this | not this | or this

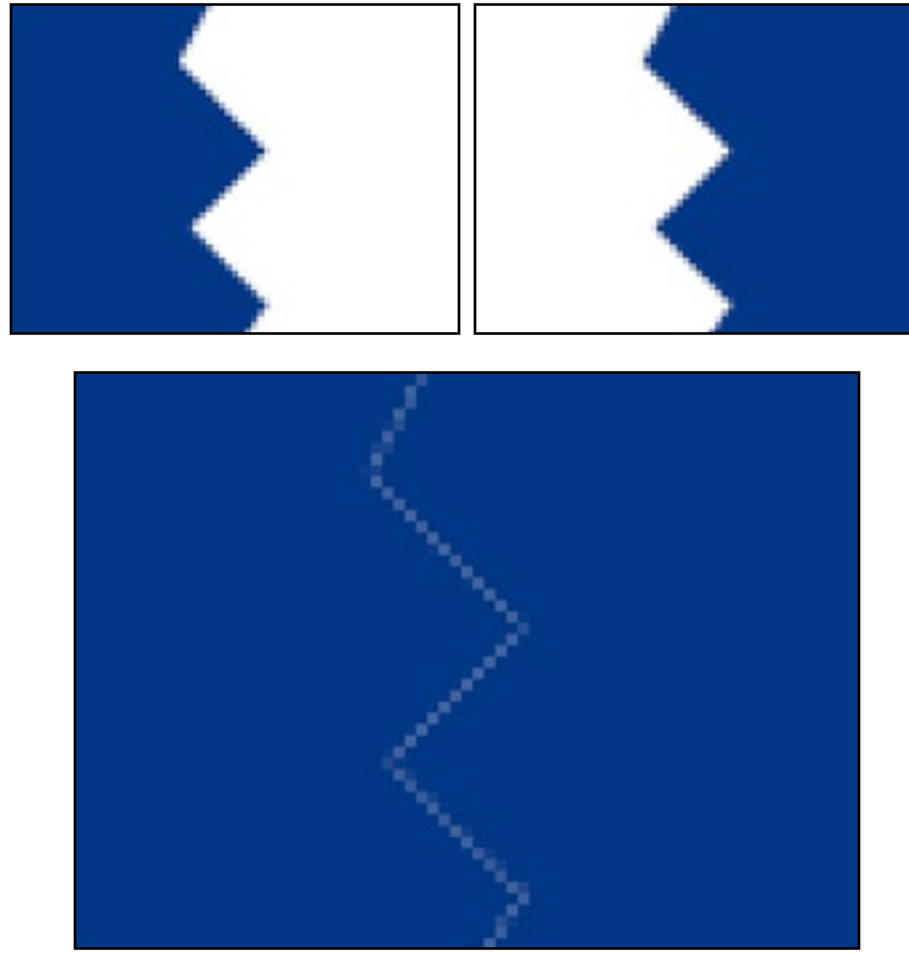
- This will cause artifacts
  - but we'll carry on anyway because it is simple and usually works…

# Alpha compositing—failures

[Cornell PCG]

[Chuang et al. / Corel]

positive correlation:
too much foreground

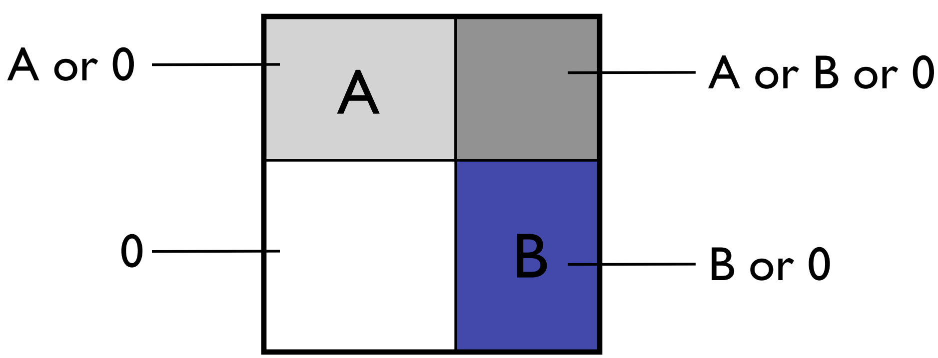


negative correlation:
too little foreground

# Other compositing operations

- Generalized form of compositing equation:

$$\alpha_E = A \textbf{ op } B$$
$$c'_E = F_A c'_A + F_B c'_B$$




1 x 2 x 3 x 2 = 12 reasonable choices

| operation | quadruple | diagram | $F_A$ | $F_B$ |
|---|---|---|---|---|
| *clear* | (0,0,0,0) | | 0 | 0 |
| *A* | (0,A,0,A) | | 1 | 0 |
| *B* | (0,0,B,B) | | 0 | 1 |
| *A* **over** *B* | (0,A,B,A) | | 1 | $1-\alpha_A$ |
| *B* **over** *A* | (0,A,B,B) | | $1-\alpha_B$ | 1 |
| *A* **in** *B* | (0,0,0,A) | | $\alpha_B$ | 0 |
| *B* **in** *A* | (0,0,0,B) | | 0 | $\alpha_A$ |
| *A* **out** *B* | (0,A,0,0) | | $1-\alpha_B$ | 0 |
| *B* **out** *A* | (0,0,B,0) | | 0 | $1-\alpha_A$ |
| *A* **atop** *B* | (0,0,B,A) | | $\alpha_B$ | $1-\alpha_A$ |
| *B* **atop** *A* | (0,A,0,B) | | $1-\alpha_B$ | $\alpha_A$ |
| *A* **xor** *B* | (0,A,B,0) | | $1-\alpha_B$ | $1-\alpha_A$ |

[Porter & Duff 84]