Antialiasing

CS4620 Lecture 16

(with previous instructors James/Bala, and some slides borrowed from Kurt Akeley)

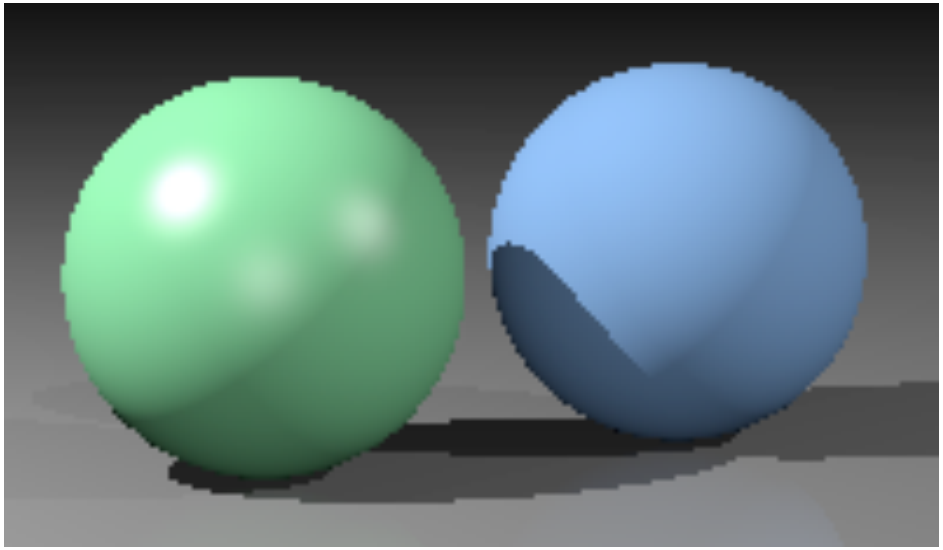
Pixel coverage

- 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

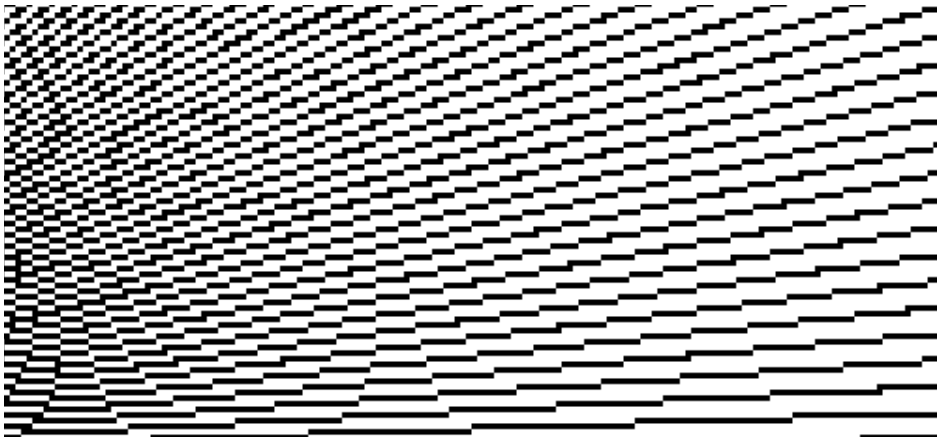
(with previous instructors James/Bala, and some slides borrowed from Kurt Akeley)

Aliasing

point sampling a continuous image:



continuous image defined by ray tracing procedure



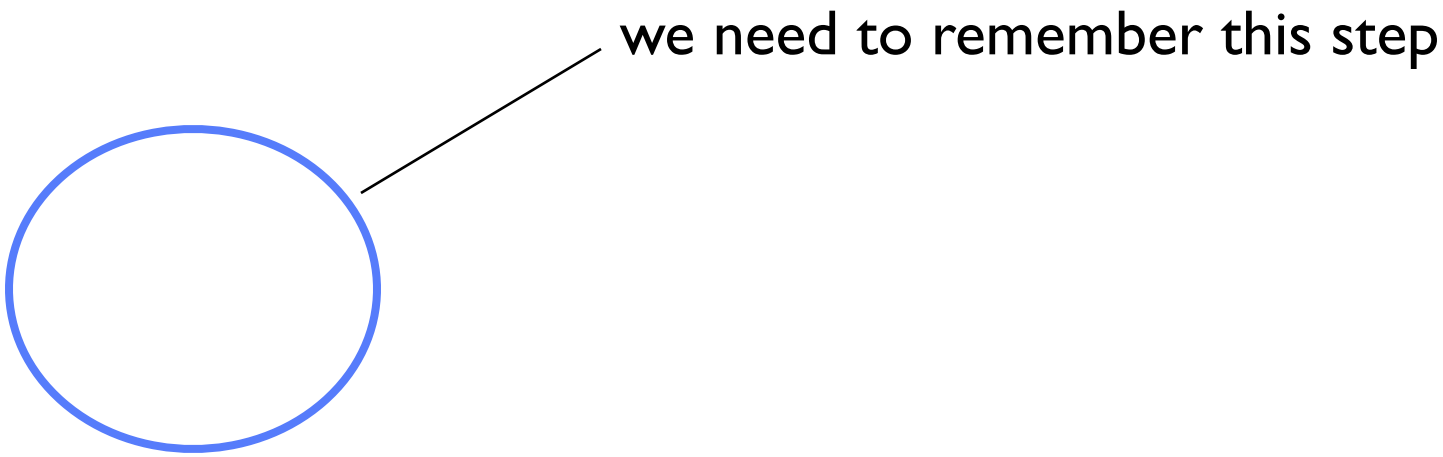
continuous image defined by a bunch of black rectangles

(with previous instructors James/Bala, and some slides borrowed from Kurt Akeley)

Signal processing view

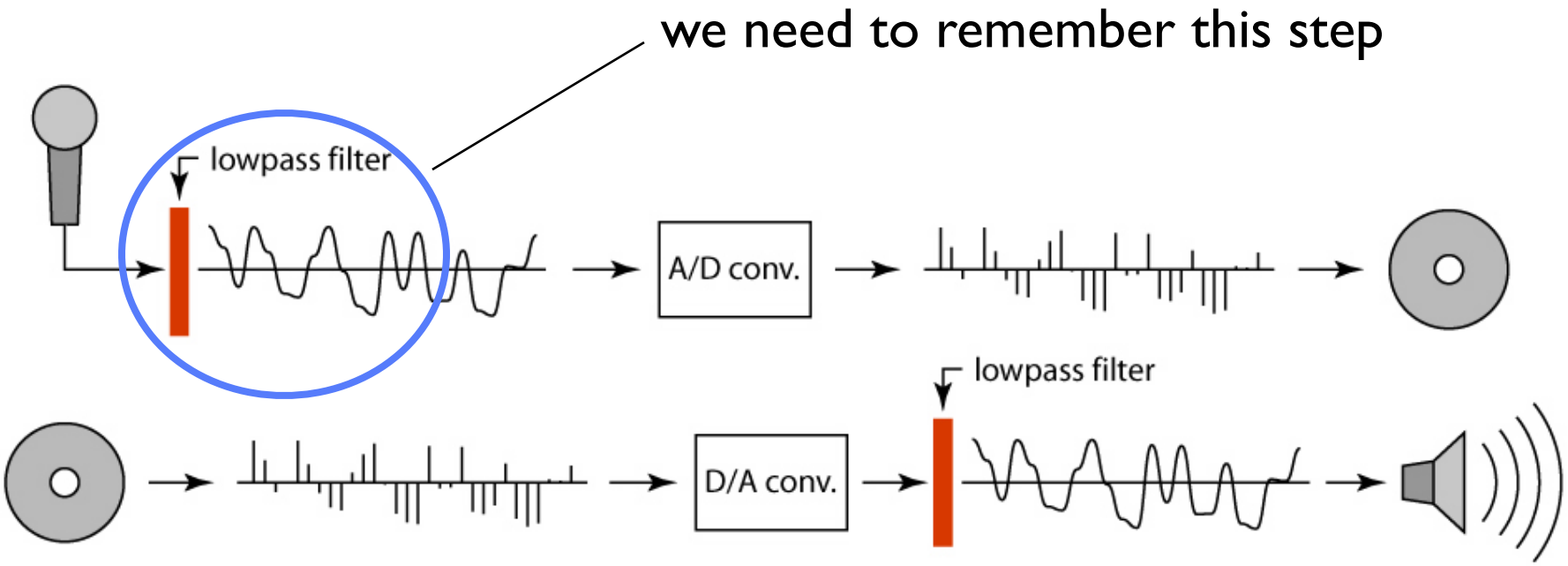
(with previous instructors James/Bala, and some slides borrowed from Kurt Akeley)

Signal processing view



(with previous instructors James/Bala, and some slides borrowed from Kurt Akeley)

Signal processing view



(with previous instructors James/Bala, and some slides borrowed from Kurt Akeley)

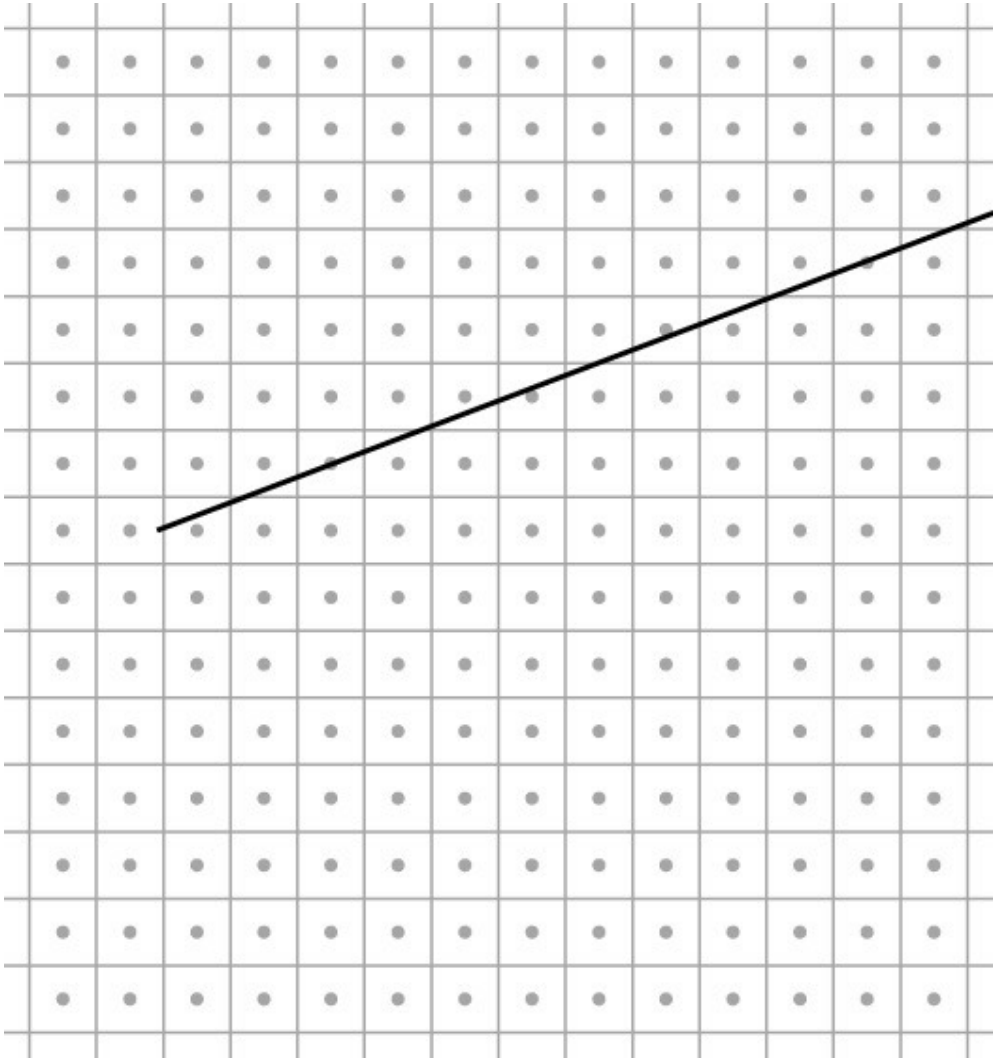
Antialiasing

- A name for techniques to prevent aliasing
- In image generation, we need to filter
  - Convolve continuous image with a sampling filter
  - Simple: average the image over an area (box filtering)
  - Better: weight by a smoother filter
- Methods depend on source of image
  - Rasterization (lines and polygons)
  - Point sampling (e.g. raytracing)
  - Texture mapping

(with previous instructors James/Bala, and some slides borrowed from Kurt Akeley)

Rasterizing lines

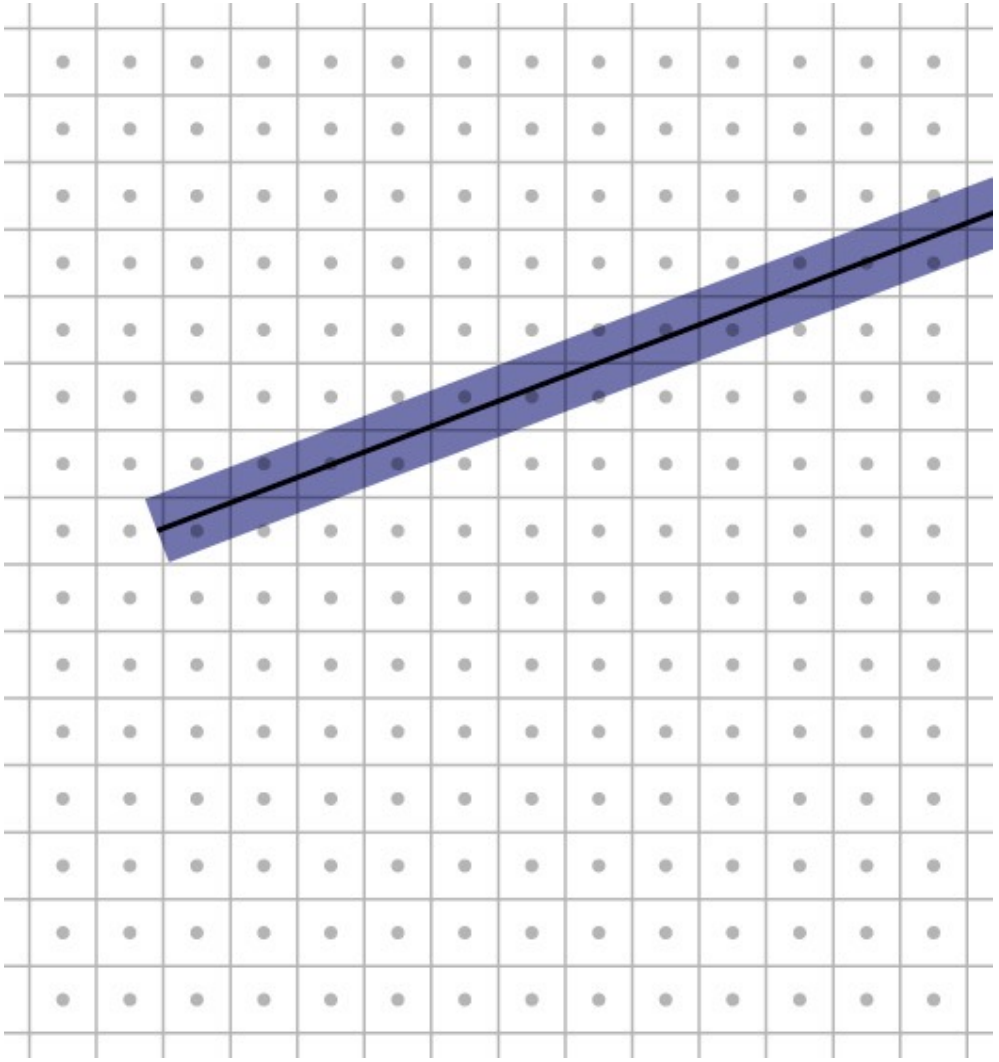
- Define line as a rectangle
- Specify by two endpoints
- Ideal image: black inside, white outside



(with previous instructors James/Bala, and some slides borrowed from Kurt Akeley)

Rasterizing lines

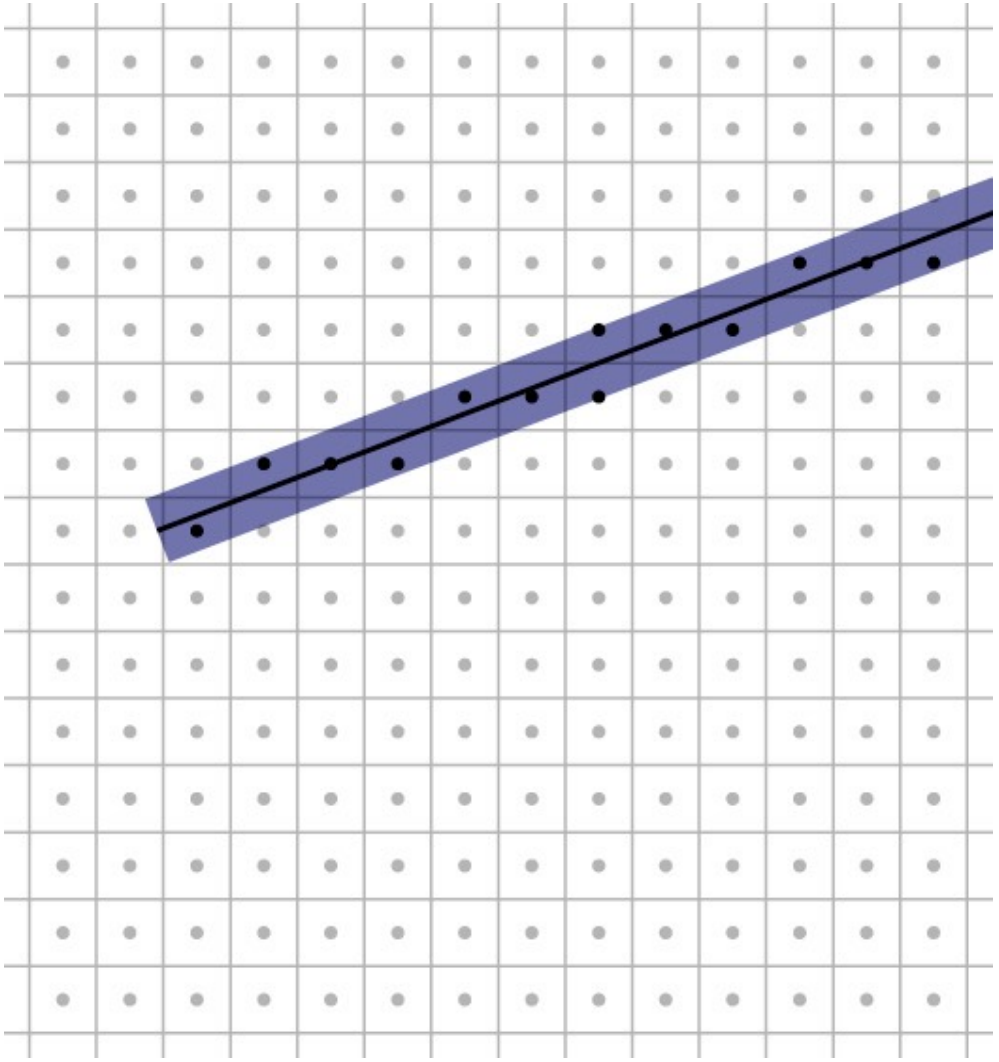
- Define line as a rectangle
- Specify by two endpoints
- Ideal image: black inside, white outside



(with previous instructors James/Bala, and some slides borrowed from Kurt Akeley)

Point sampling

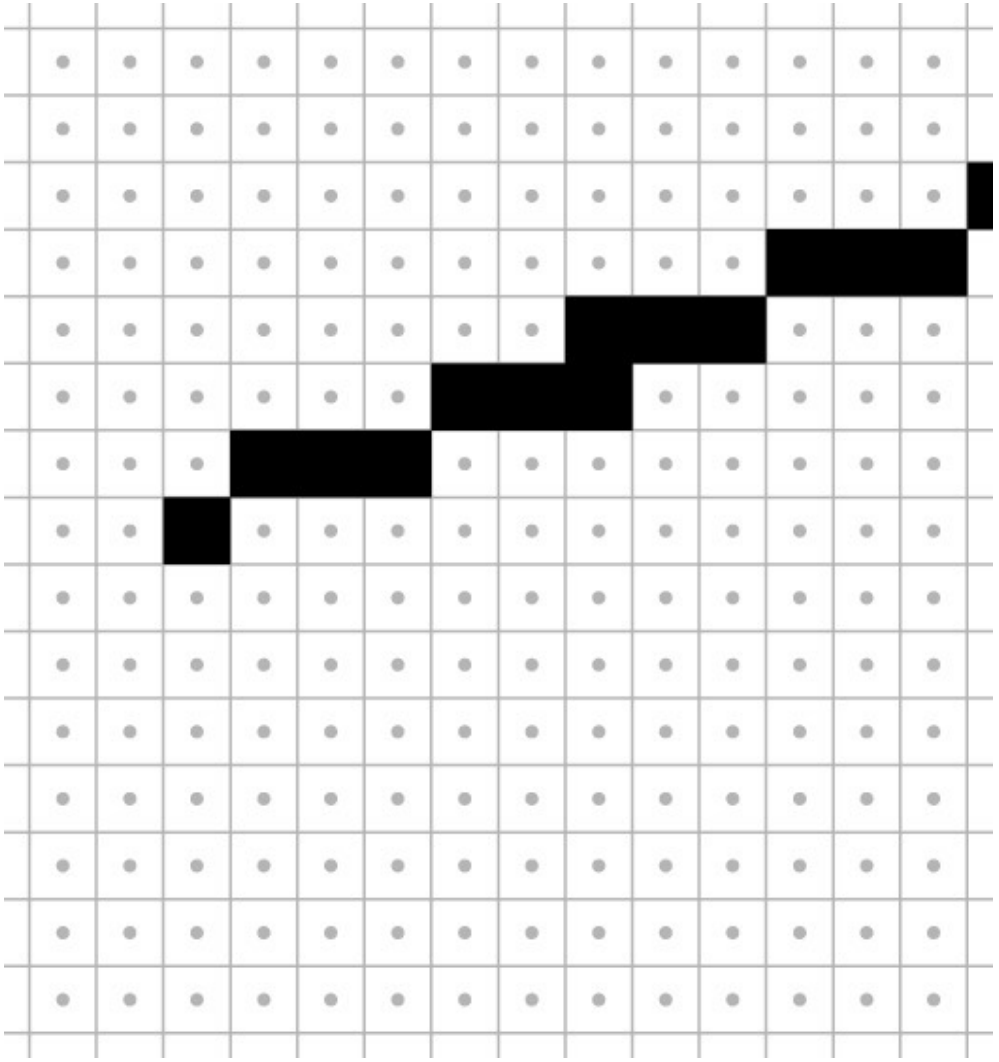
- Approximate rectangle by drawing all pixels whose centers fall within the line
- Problem: all-or-nothing leads to jaggies
  - this is sampling with no filter (aka. point sampling)

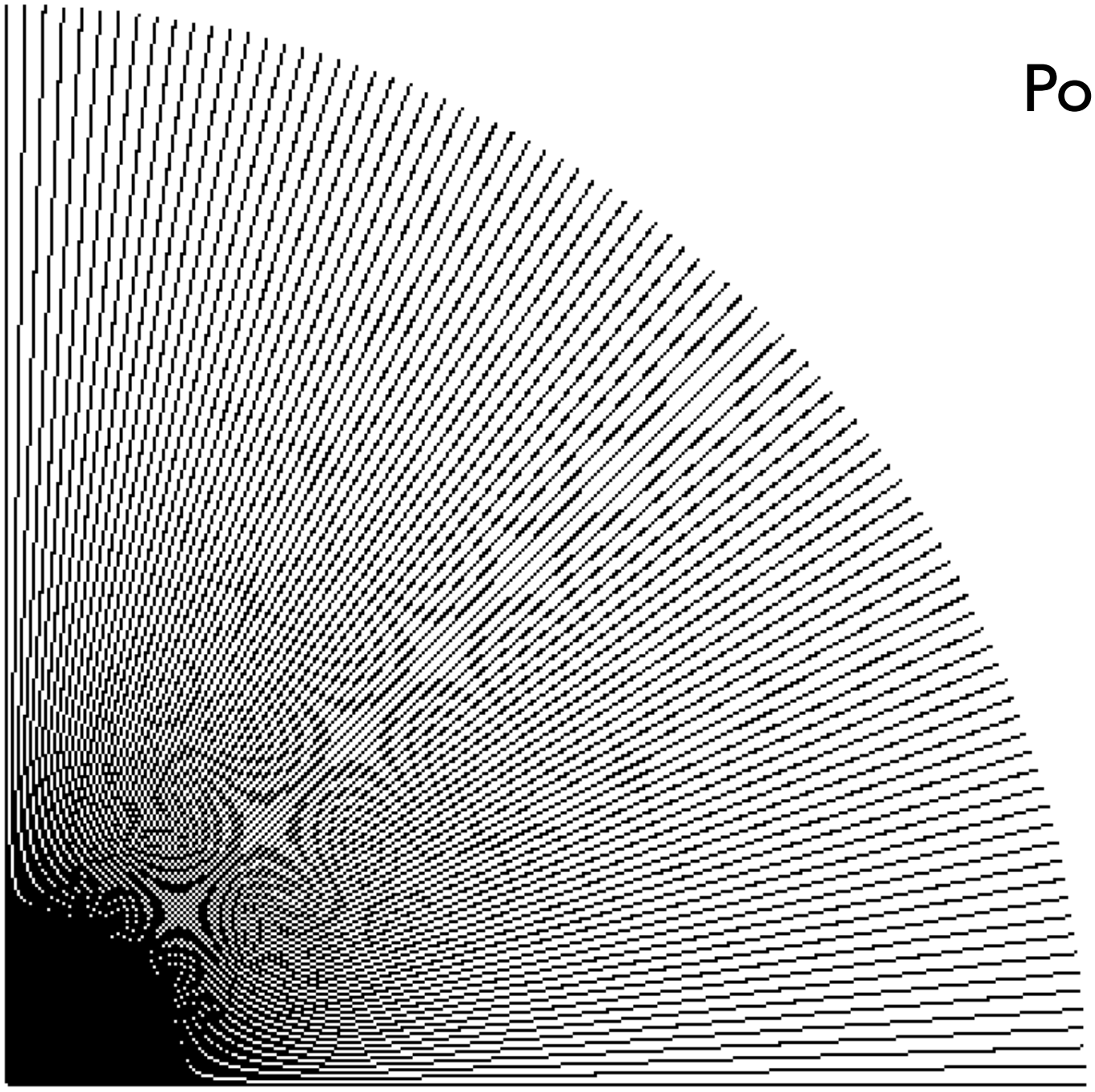


(with previous instructors James/Bala, and some slides borrowed from Kurt Akeley)

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)

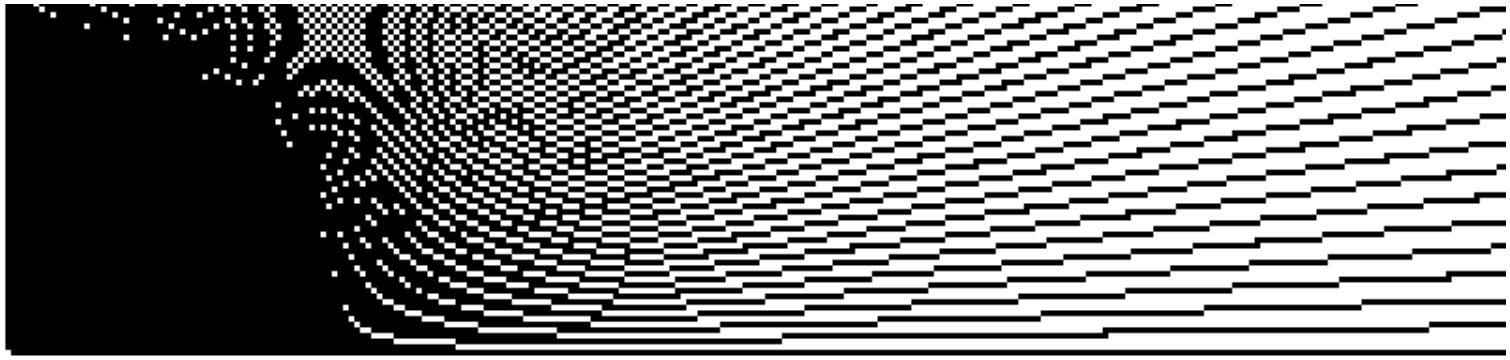




Point sampling in action

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!



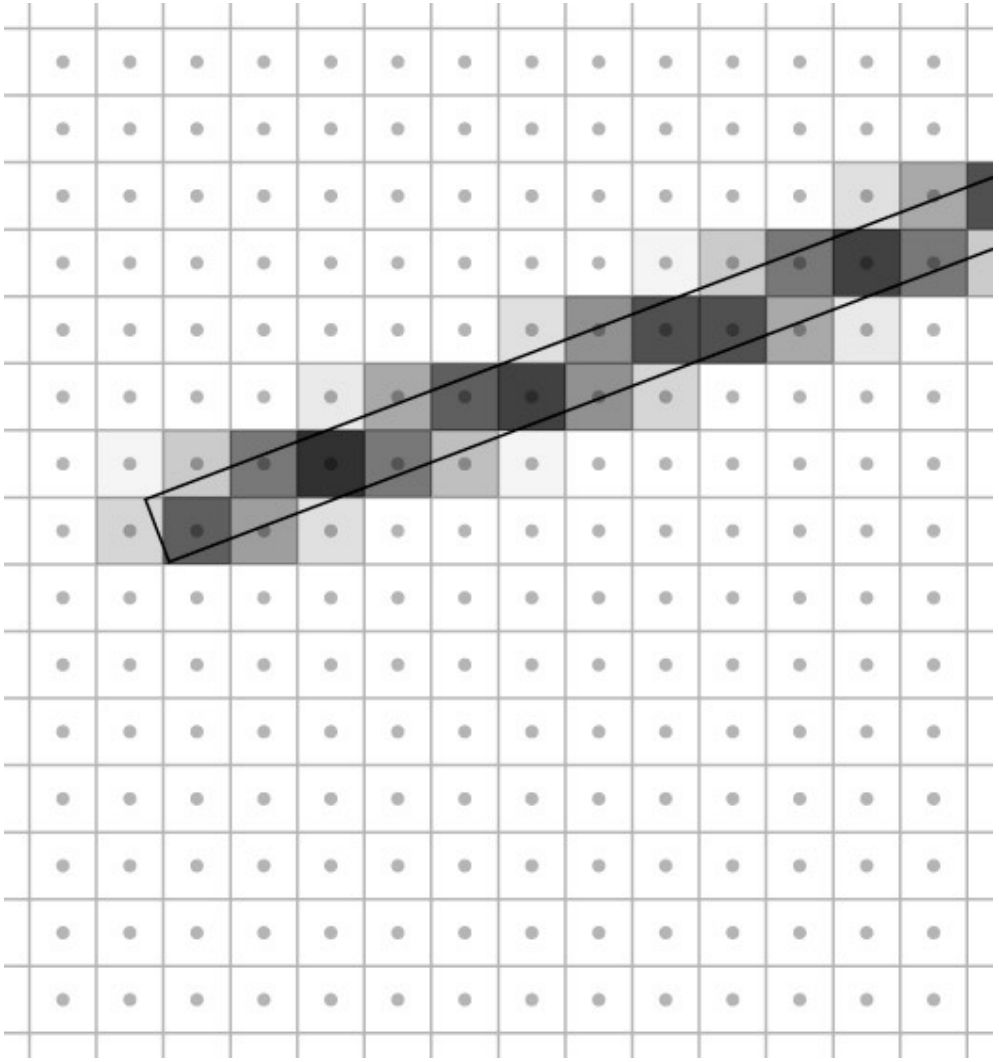
(with previous instructors James/Bala, and some slides borrowed from Kurt Akeley)

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.



(with previous instructors James/Bala, and some slides borrowed from Kurt Akeley)

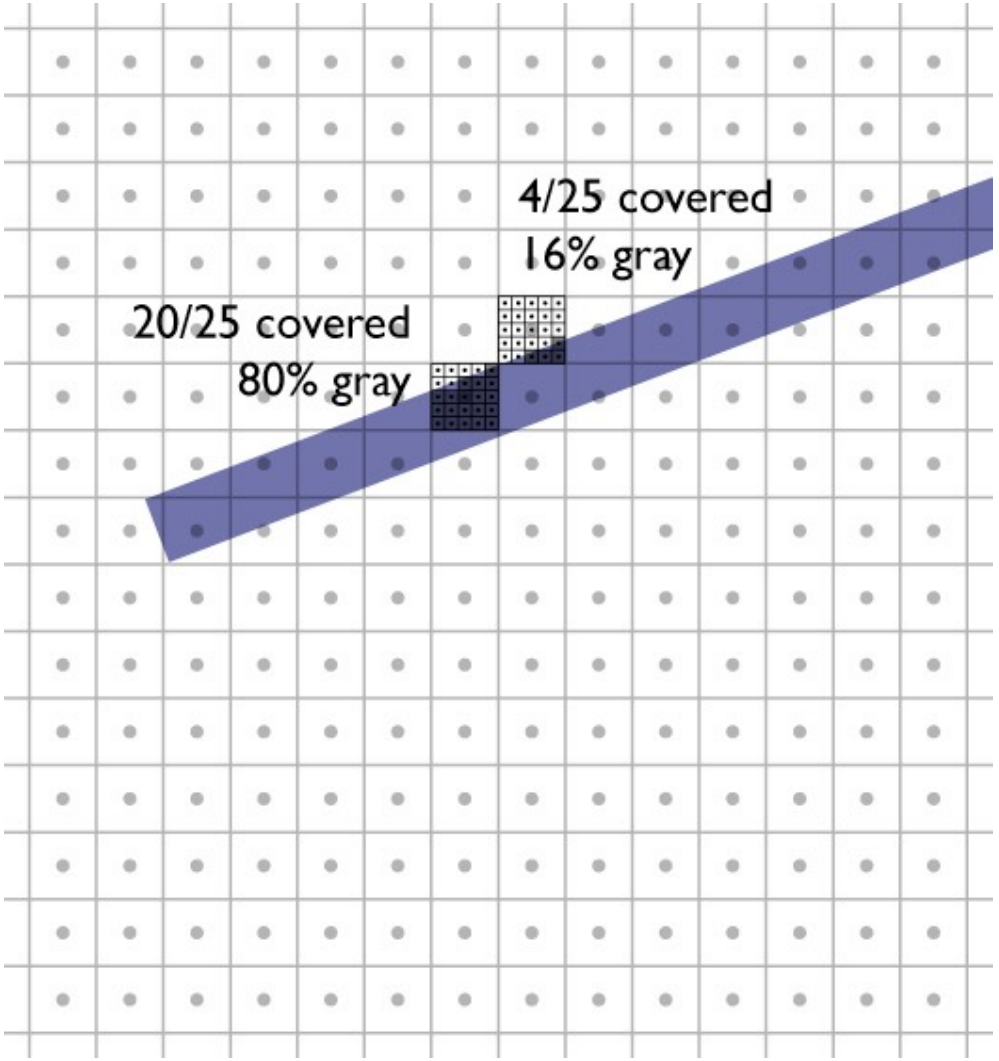
Box filtering

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

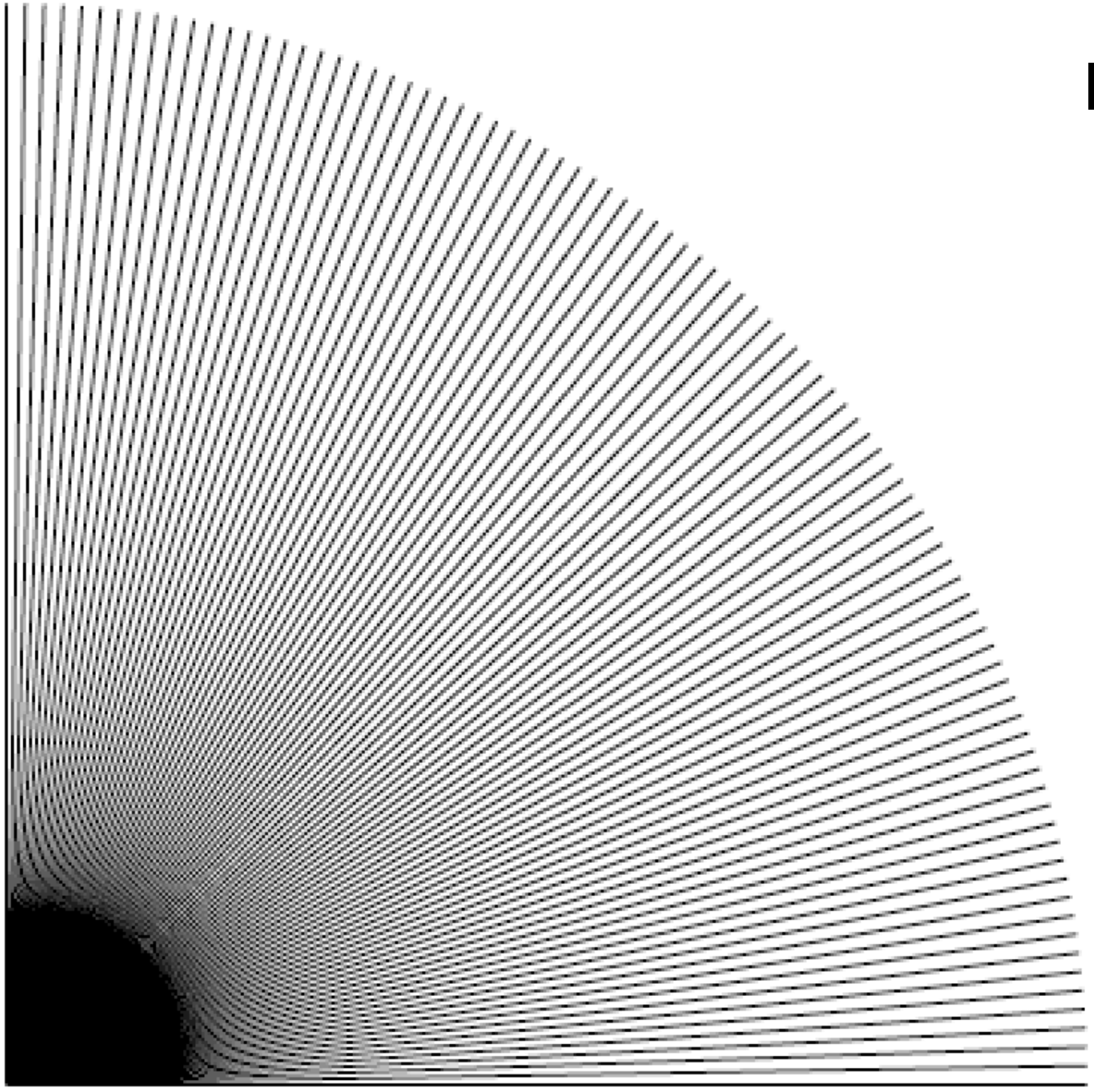
(with previous instructors James/Bala, and some slides borrowed from Kurt Akeley)

Box filtering by supersampling

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



(with previous instructors James/Bala, and some slides borrowed from Kurt Akeley)



Box filtering in action

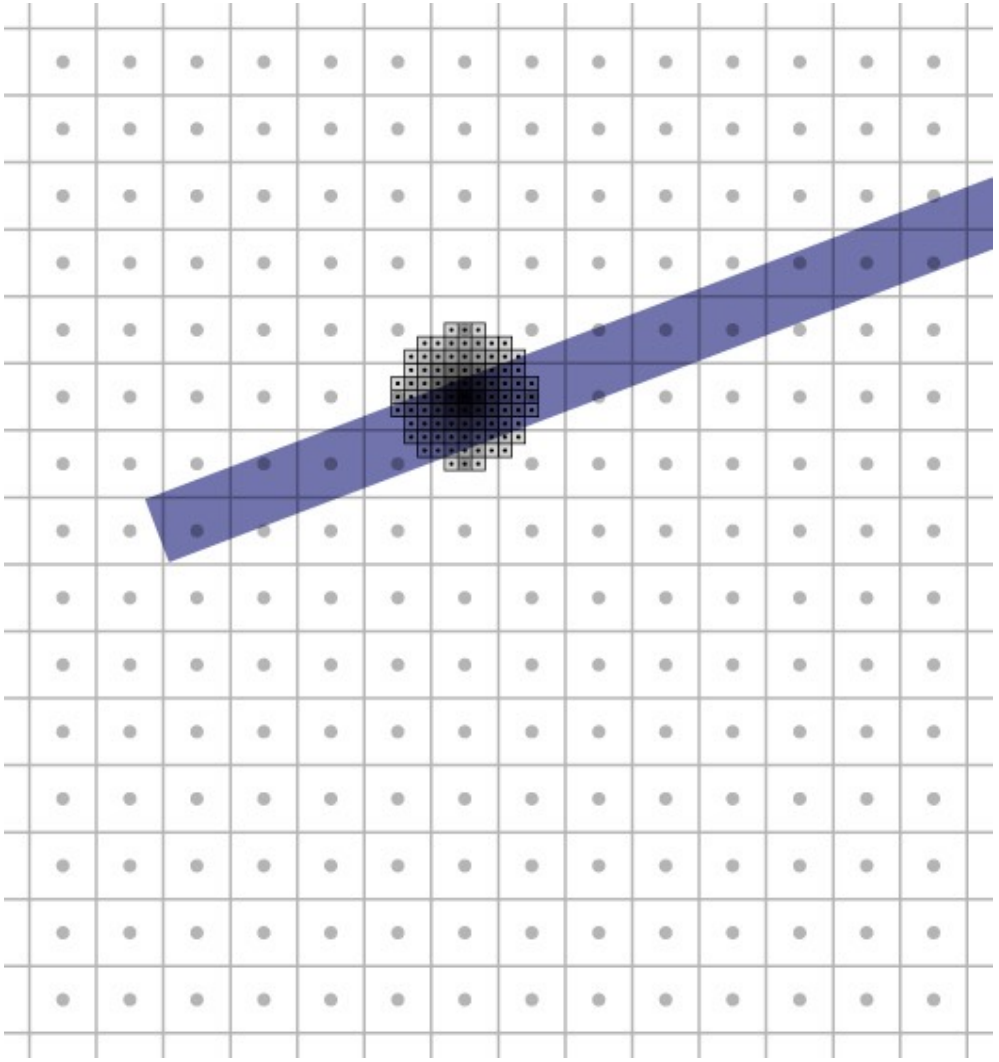
Weighted filtering

- 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)

(with previous instructors James/Bala, and some slides borrowed from Kurt Akeley)

Weighted filtering by supersampling

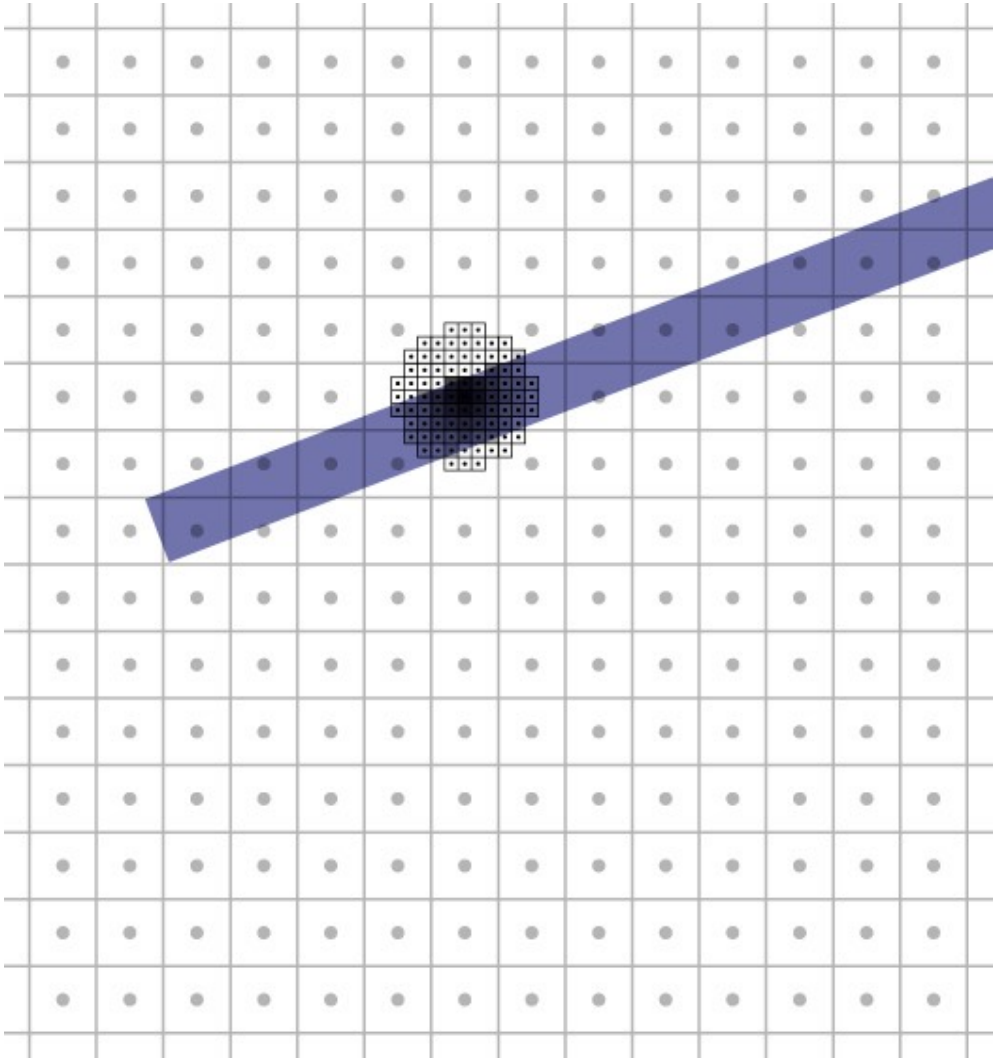
- Compute filtering integral by summing filter values for covered subpixels
- Simple, accurate
- But really slow



(with previous instructors James/Bala, and some slides borrowed from Kurt Akeley)

Weighted filtering by supersampling

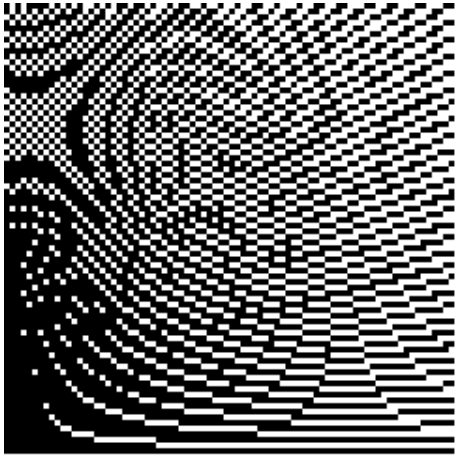
- Compute filtering integral by summing filter values for covered subpixels
- Simple, accurate
- But really slow



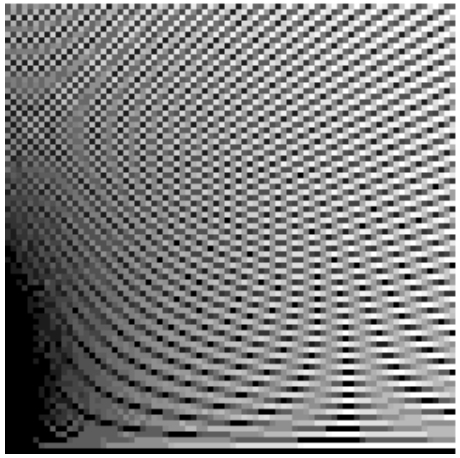
(with previous instructors James/Bala, and some slides borrowed from Kurt Akeley)

Gaussian filtering in action

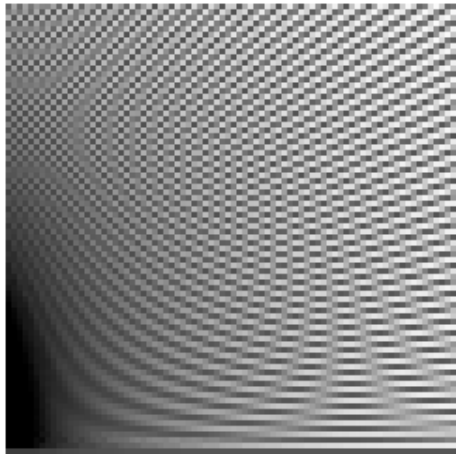
Filter comparison



Point sampling



Box filtering

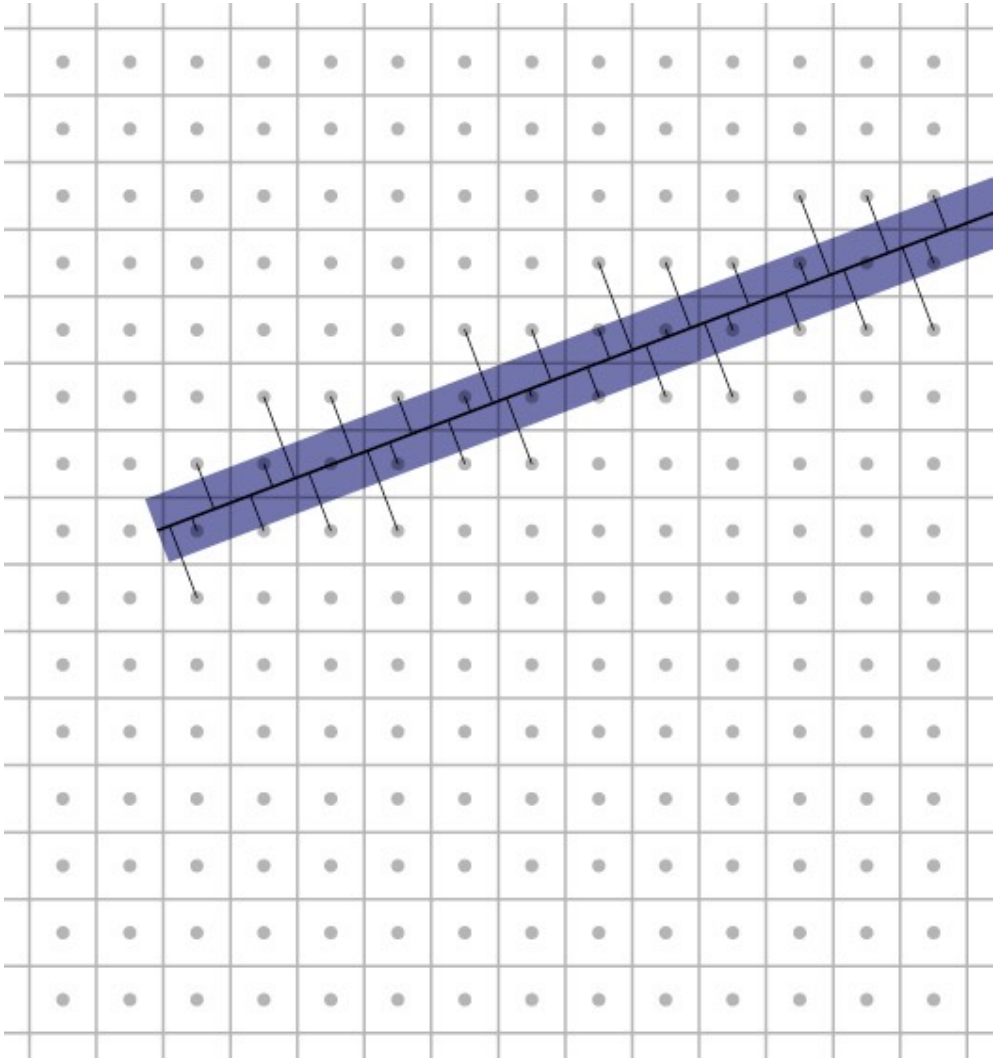


Gaussian filtering

(with previous instructors James/Bala, and some slides borrowed from Kurt Akeley)

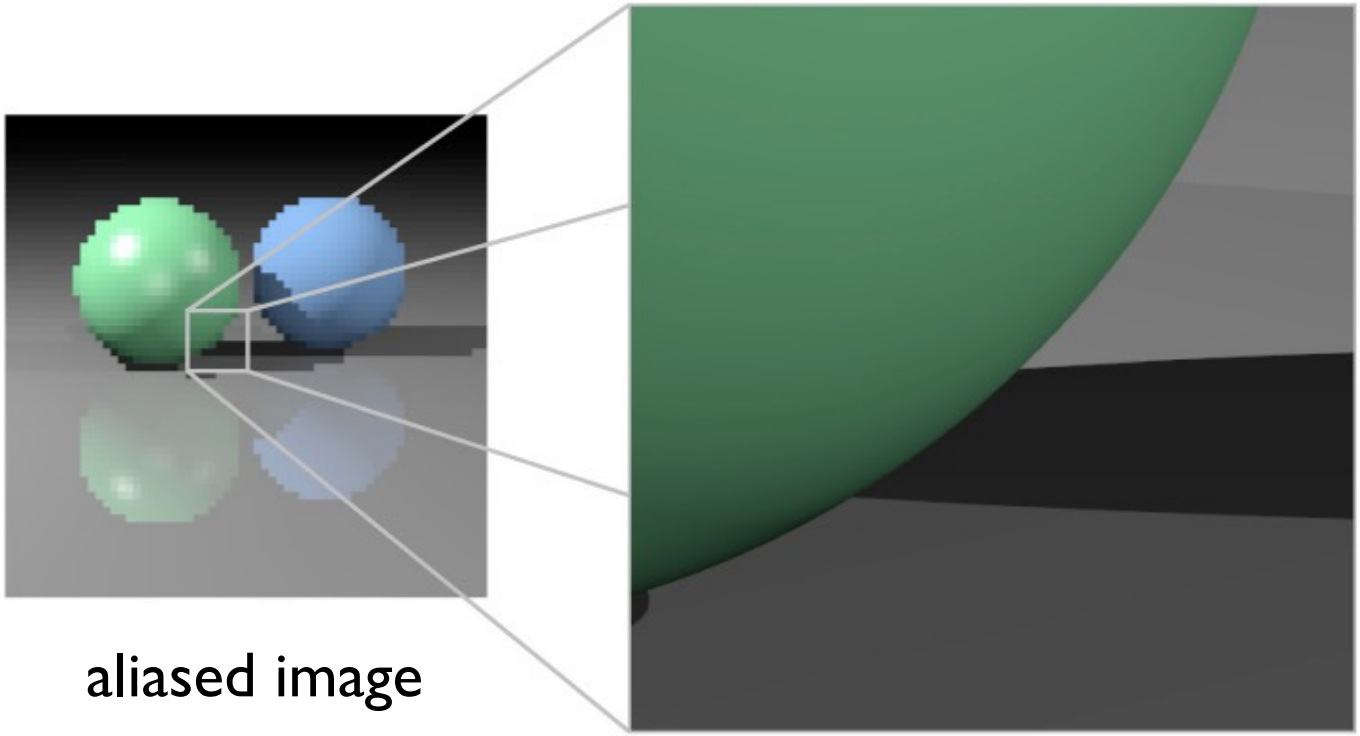
More efficient antialiased lines

- Filter integral is the same for pixels the same distance from the center line
- Just look up in precomputed table based on distance
  - Gupta-Sproull
- Some additional details at ends…



(with previous instructors James/Bala, and some slides borrowed from Kurt Akeley)

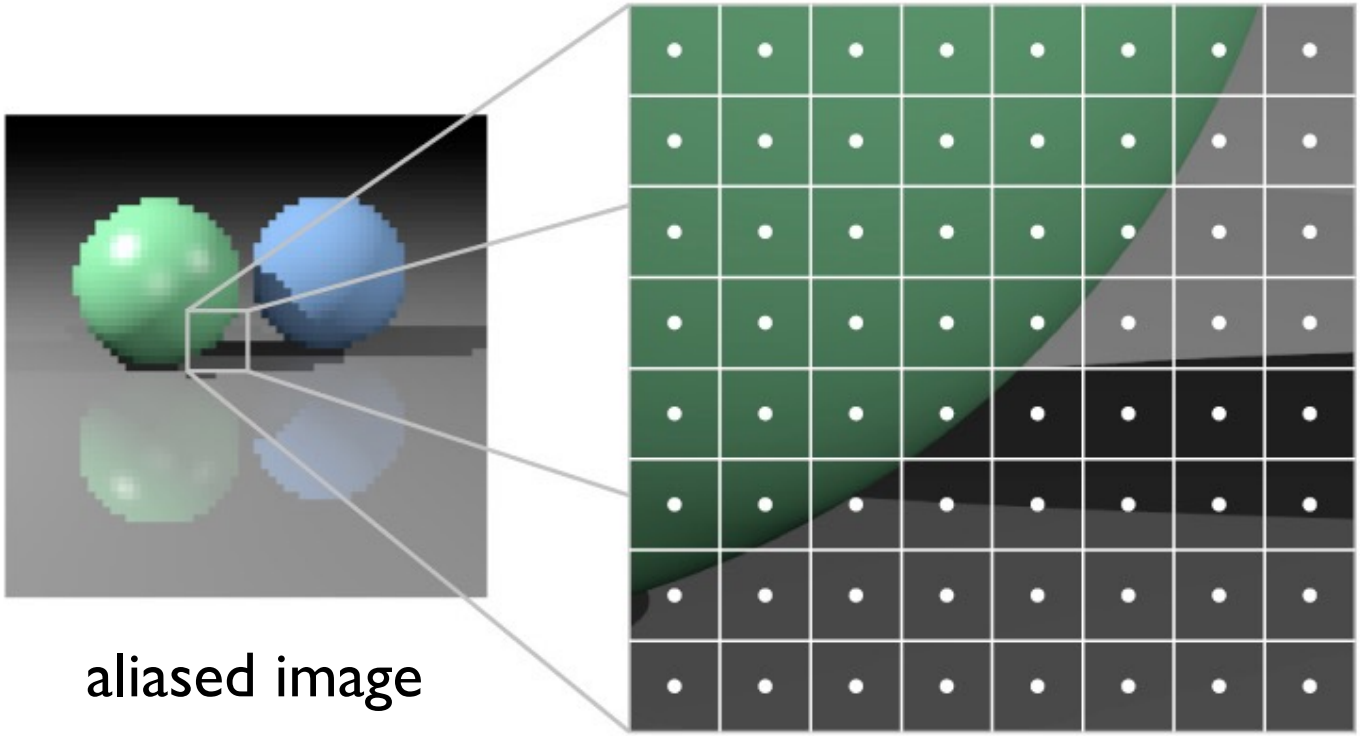
Antialiasing in ray tracing



aliased image

(with previous instructors James/Bala, and some slides borrowed from Kurt Akeley)

Antialiasing in ray tracing

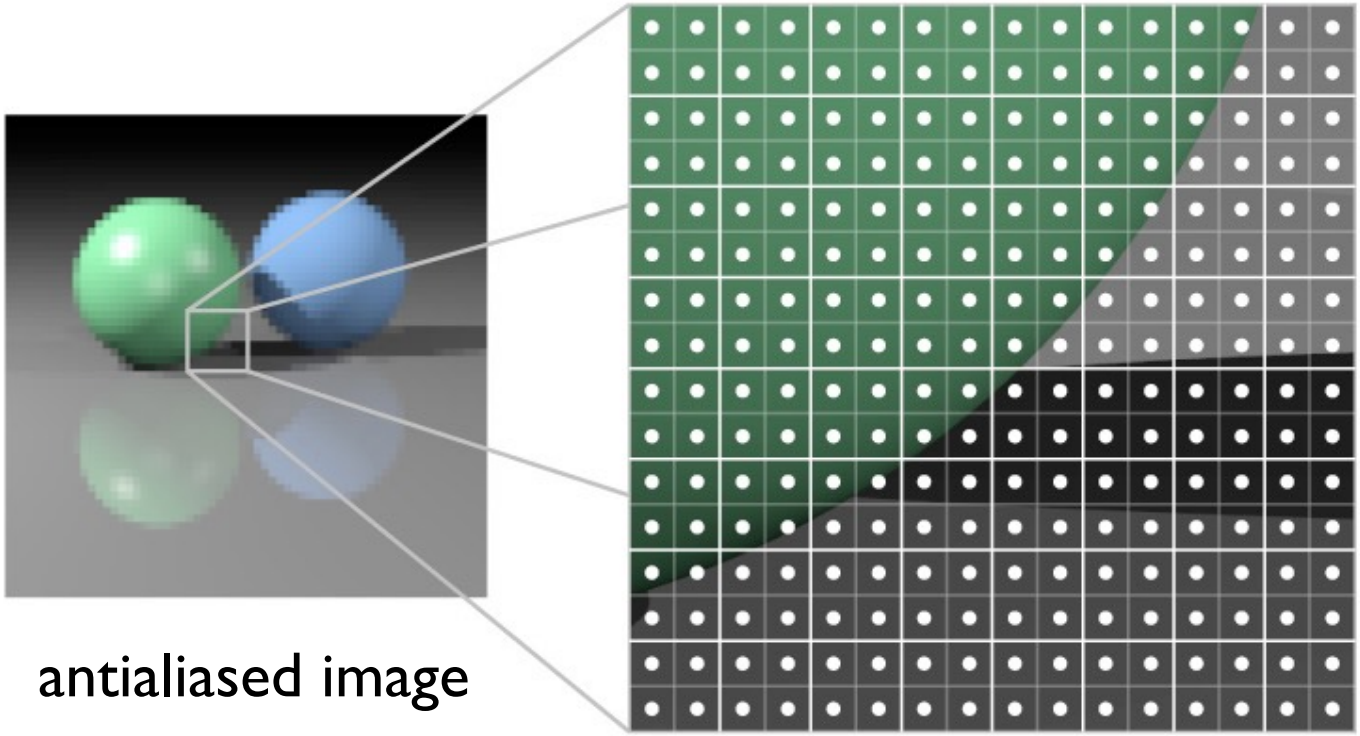


aliased image

one sample per pixel

(with previous instructors James/Bala, and some slides borrowed from Kurt Akeley)

Antialiasing in ray tracing

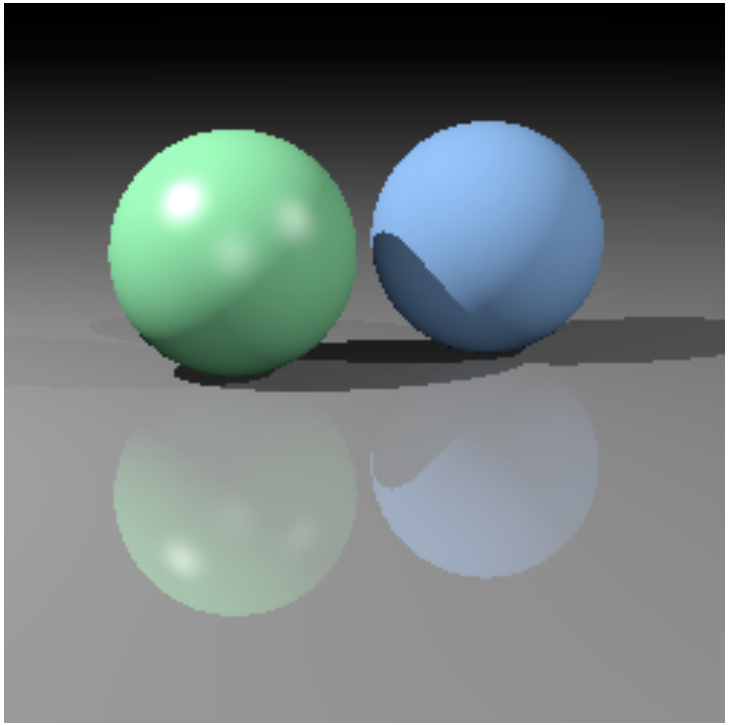


antialiased image

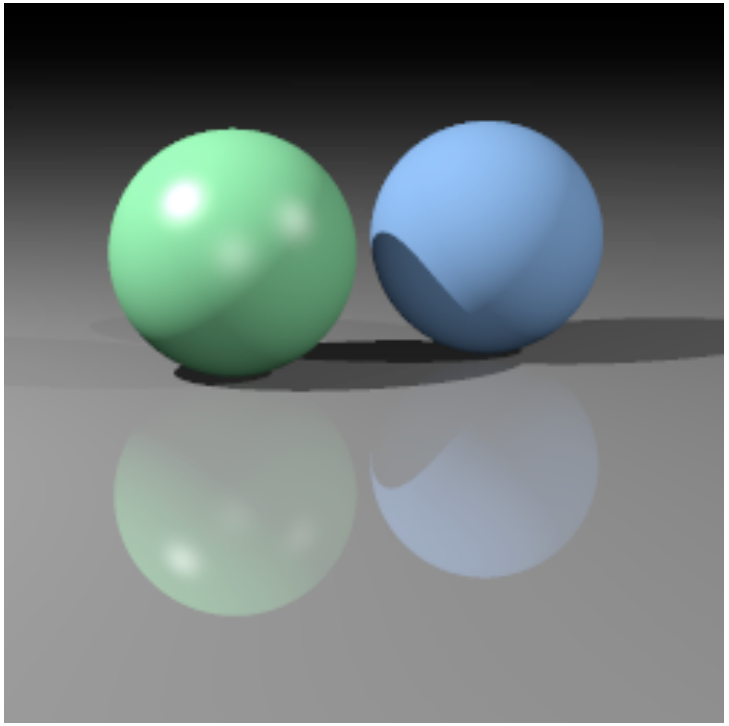
four samples per pixel

(with previous instructors James/Bala, and some slides borrowed from Kurt Akeley)

Antialiasing in ray tracing



one sample/pixel



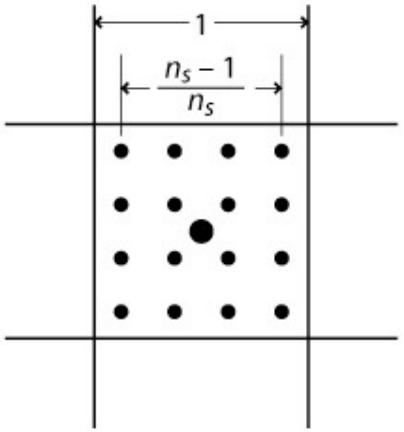
9 samples/pixel

(with previous instructors James/Bala, and some slides borrowed from Kurt Akeley)

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));
   }
```


(with previous instructors James/Bala, and some slides borrowed from Kurt Akeley)

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));
   }
```


(with previous instructors James/Bala, and some slides borrowed from Kurt Akeley)

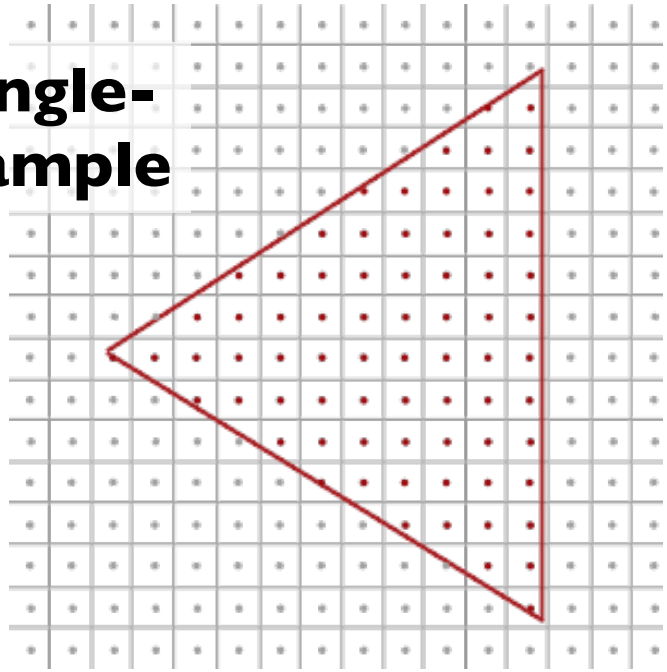
Supersampling vs. multisampling

- 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

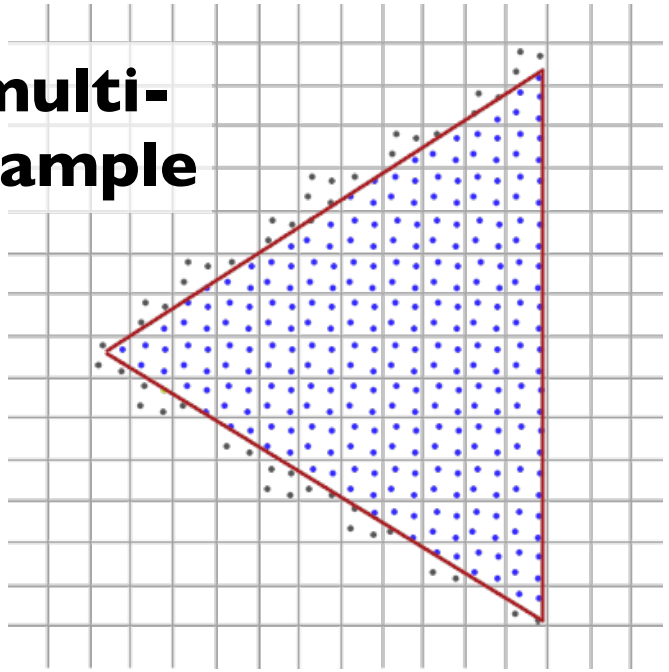
Multisample rasterization

- 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

single-sample



multi-sample



[http://www.learnopengl.com]

(with previous instructors James/Bala, and some slides borrowed from Kurt Akeley)

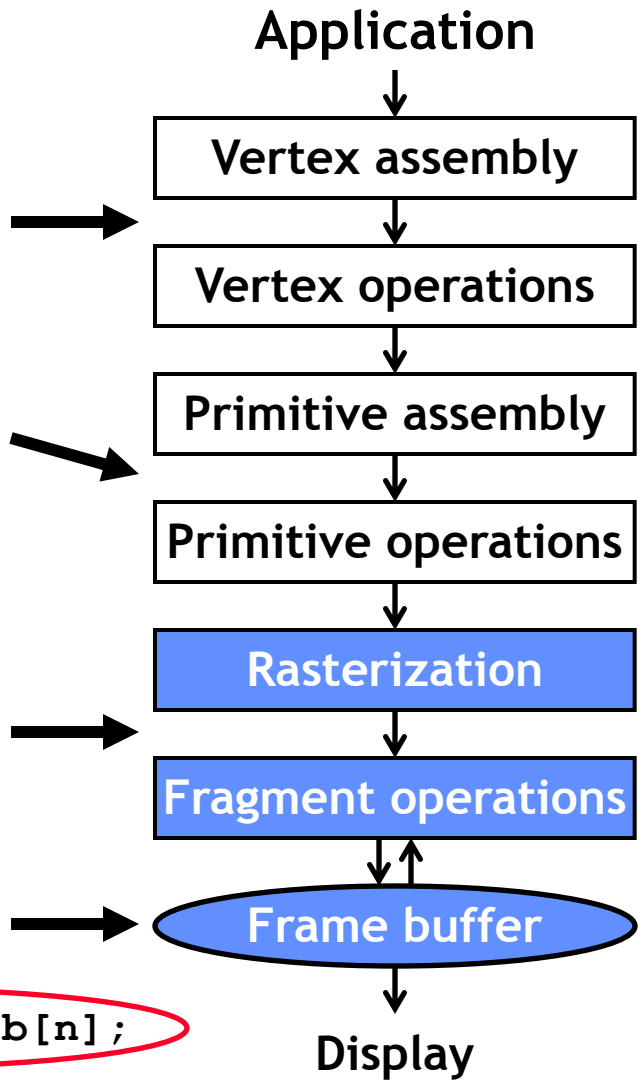
Multisample implementation (n samples)

```
struct {
  float x,y,z,w;
  float r,g,b,a;
} vertex;

struct {
  vertex v0,v1,v2
} triangle;

struct {
  short x,y;
  bool mask[n];
  float depth[n];
  float r,g,b,a;
} fragment;

struct {
  short R,G,B;
  int depth[n];
  byte r[n],g[n],b[n];
} pixel;
```



Point sampling at multisample locations sets mask and computes depths. Color is taken from center-most sample.

Fragment samples are depth-merged into the multisample buffer, then filtered to the color buffer.

Multisample rasterization operations

Fragment selection

- Identify pixels for which fragments are to be generated
- New: generate fragment if *any* sample is within the primitive
  - Requires tiled sampling, rather than point sampling
  - Generates more fragments

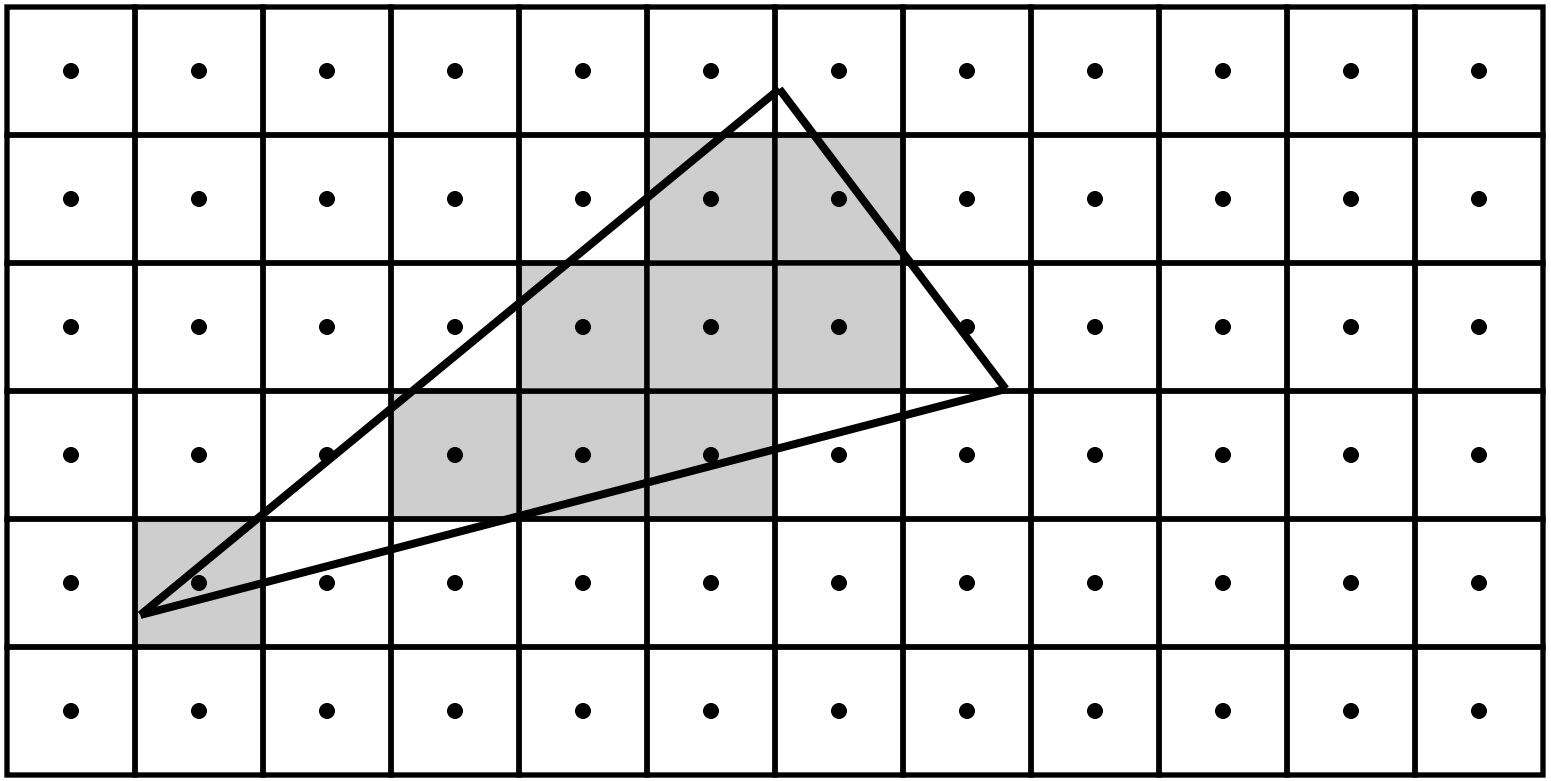
Attribute assignment

- Assign attribute values to each fragment
- Sample color at the center of the pixel (as before)
- New: compute the Boolean per-sample coverage mask
  - True if-and-only-if the sample is within the primitive
- New: compute depth values for each sample location

Point-sampled fragment selection

Generate fragment if pixel center is inside triangle

Implements point-sampled aliased rasterization



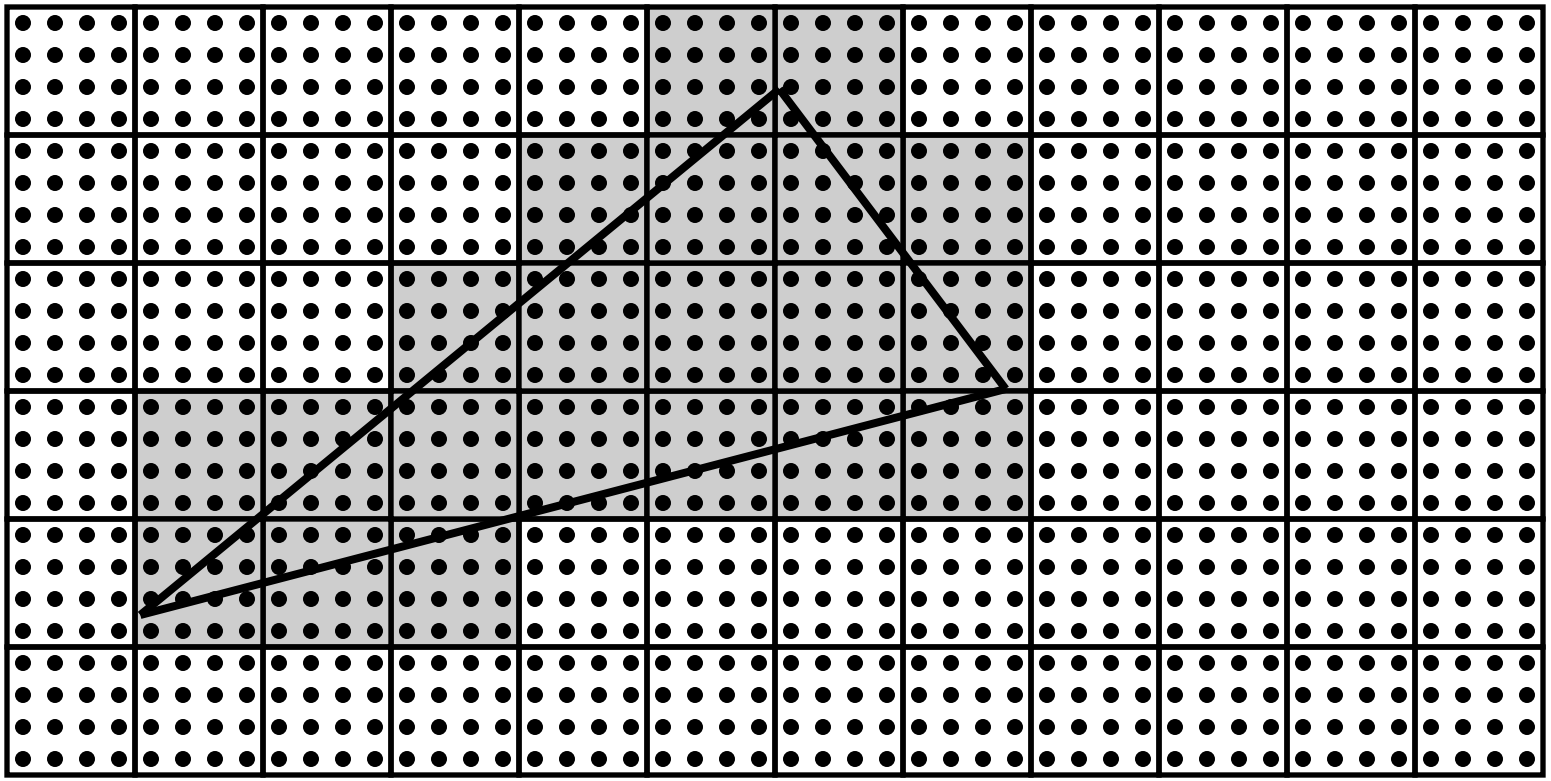
9 fragments generated

Tiled fragment selection

Generate fragment if unit square intersects triangle

Implements multisample rasterizations

- 4x4 sample pattern with unit-square filter extent



21 fragments generated