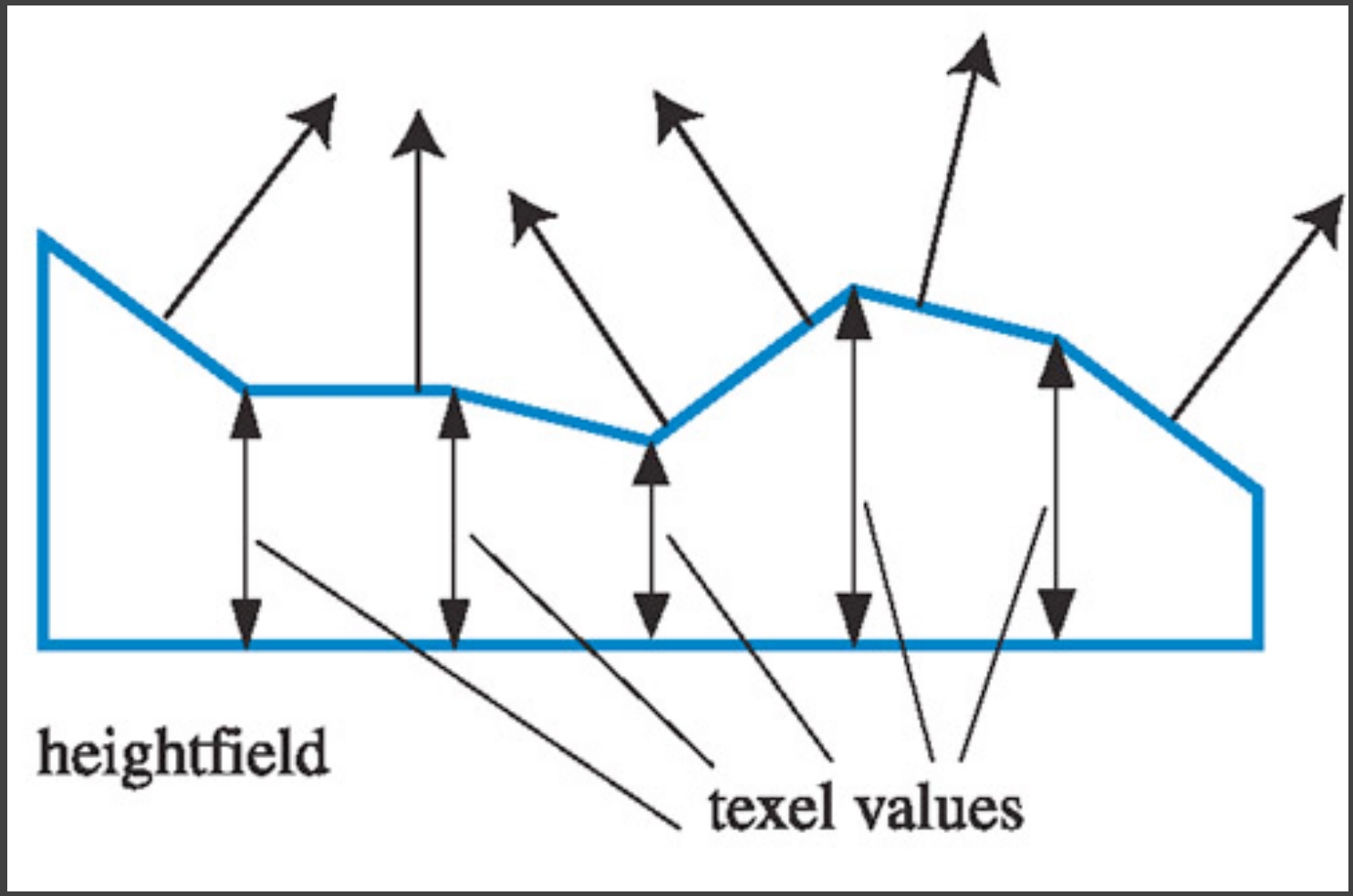
Antialiasing and Compositing

CS 4620 Lecture 23

Heightfield: Blinn's original idea

- Single scalar, more computation to infer N'



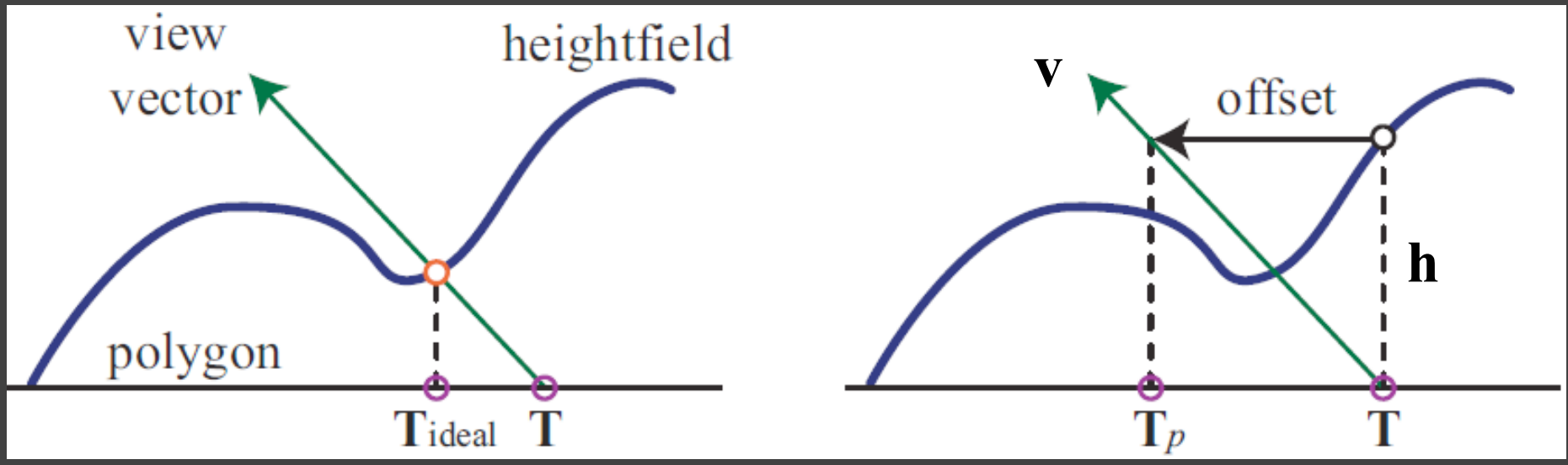
Parallax Mapping

- Problem with normal mapping
  - No self-occlusion
  - Supposed to be a height field but never see this occlusion across different viewing angles
- Parallax mapping
  - Positions of objects move relative to one other as viewpoint changes

Parallax Mapping

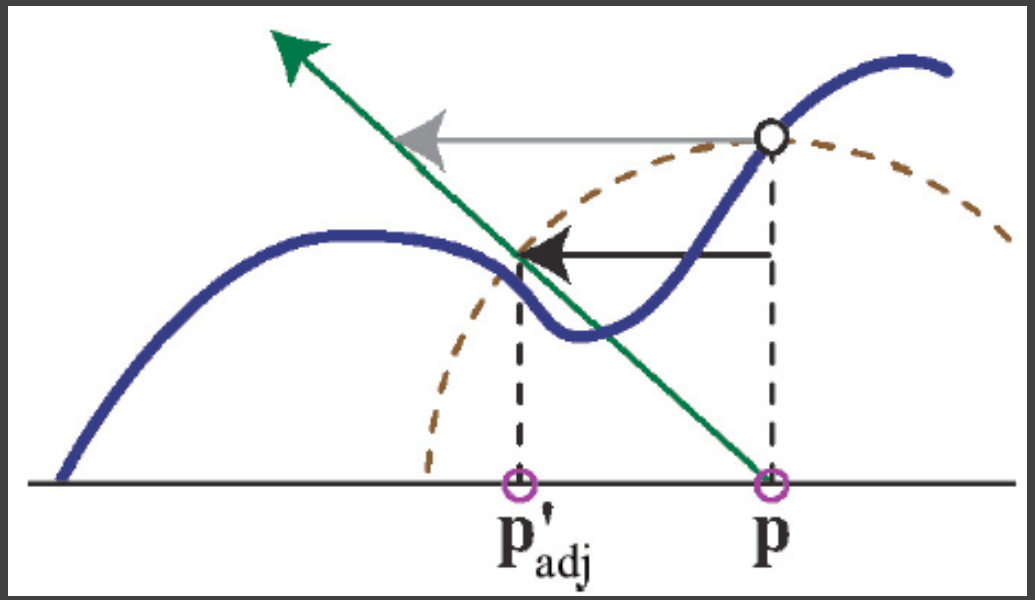
- Want T_{ideal}
- Use T_p to approximate it

$$p_{adj} = p + h\frac{v_{xy}}{v_z}$$



Parallax Offset Limiting

- Problem: at steep viewing, can offset too much
- Limit offset $p_{adj} = p + hv_{xy}$



Parallax Offset Limiting

- Widely used in games
  - the standard in bump mapping

Normal Mapping

Parallax Mapping Offset Limiting

1,100 polygon object w/ parallax occlusion mapping

1.5 million polygon

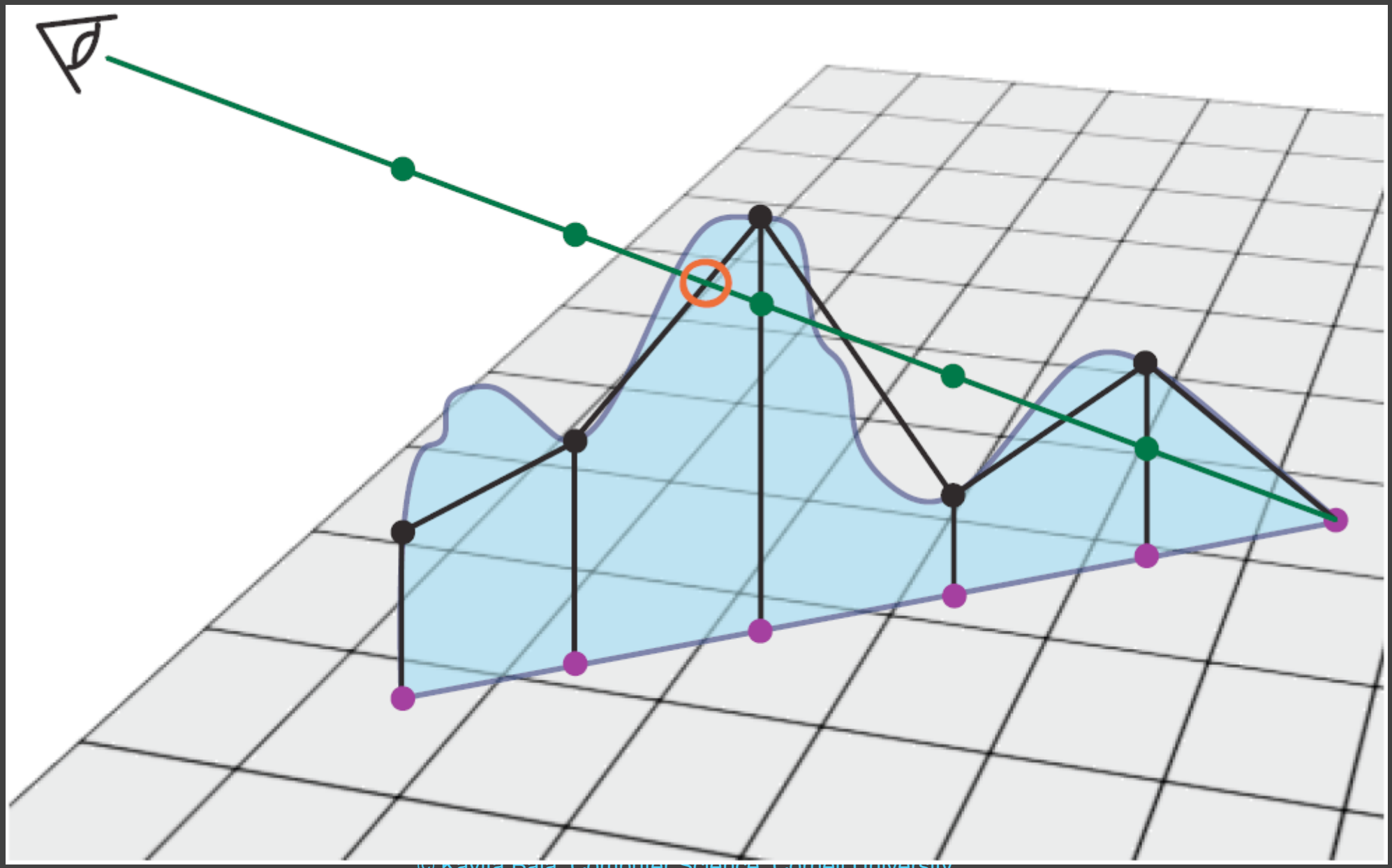
Parallax Mapping

- Parallax Mapping

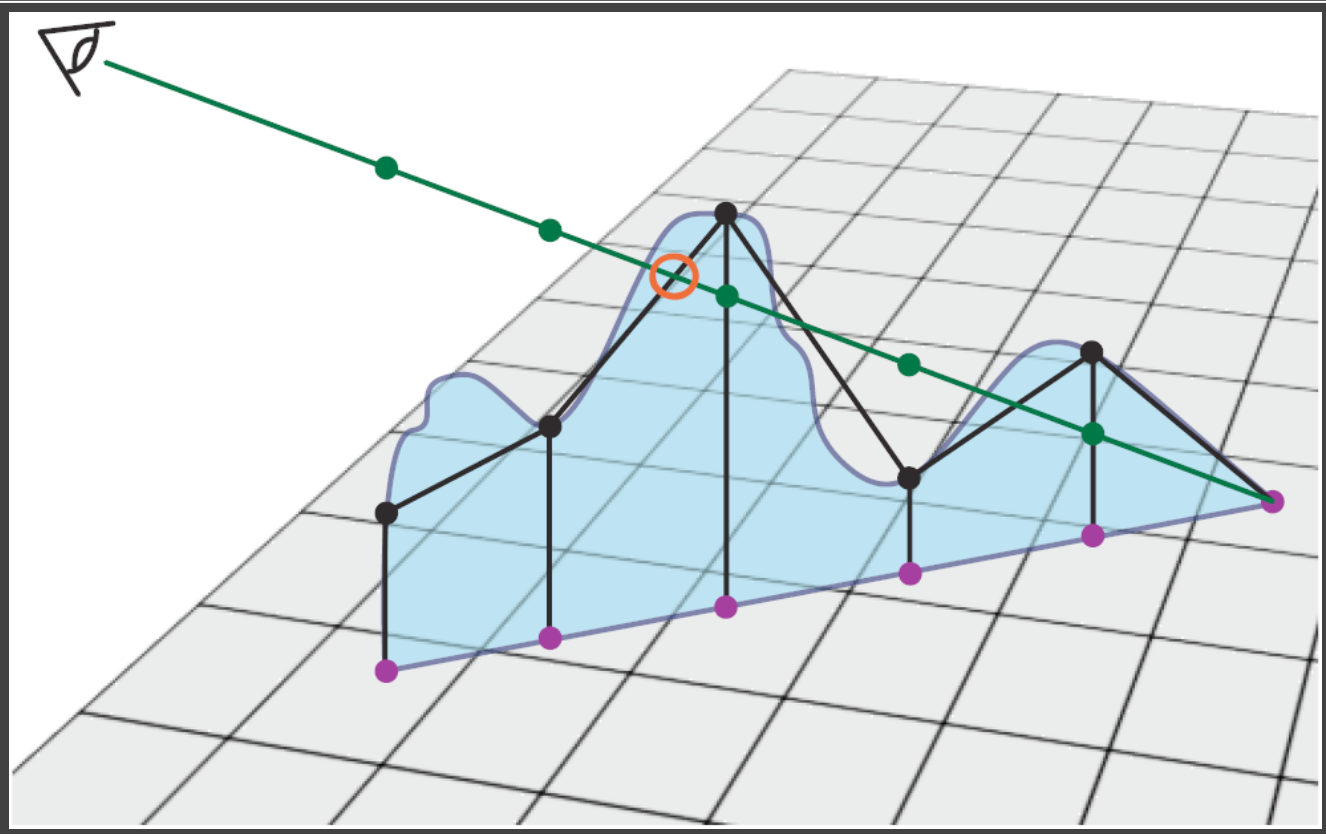
Relief Mapping

- Aka Parallax occlusion mapping, relief mapping, steep parallax mapping
- Tries to find where the view ray intersects the height field

Relief Mapping



```
Sample along ray (green points)
Lookup violet points (texture values)
    /* Infer the black line shape */
Compare green points with black points
Find intersect between two conditions
    prev: green above black
    next: green below black
```



Parallax Mapping

Relief Mapping

Relief Mapping

- Box

- Terrain

Crysis, Crytek

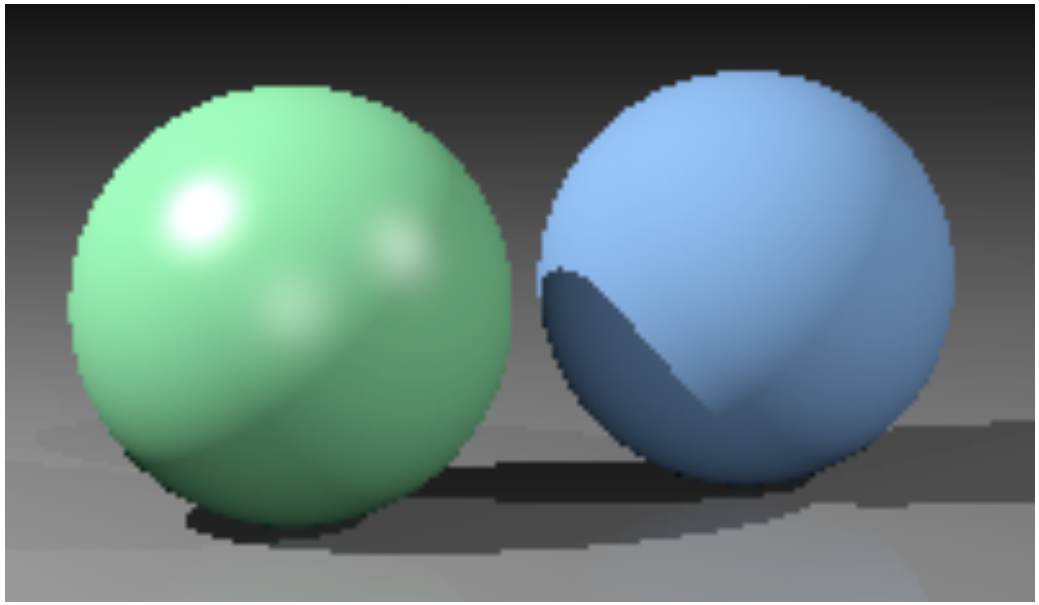
Stepping back to images

- Texture maps are images
- Image rasterization quality
- Ray tracing quality

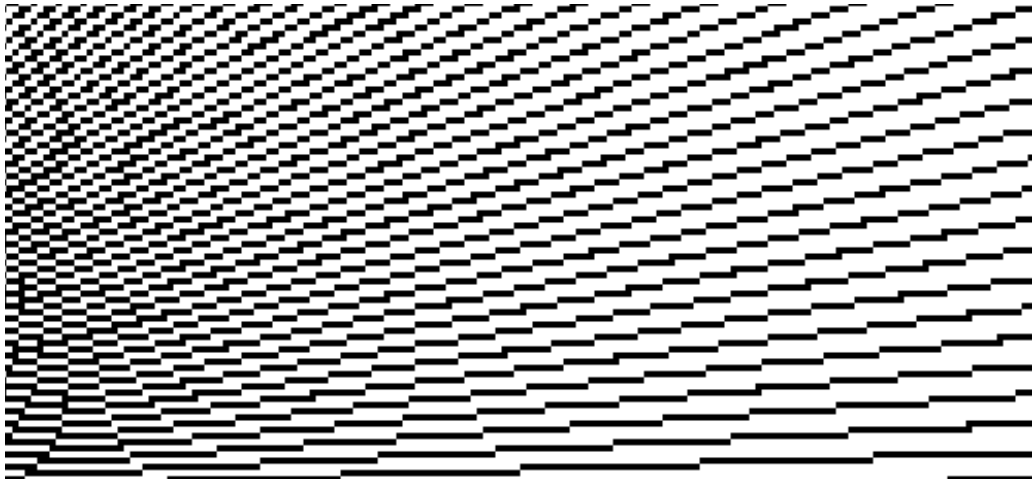
- Looking at single samples… need to go beyond
- Antialiasing and compositing

Aliasing

point sampling a
continuous image



continuous image defined
by ray tracing procedure



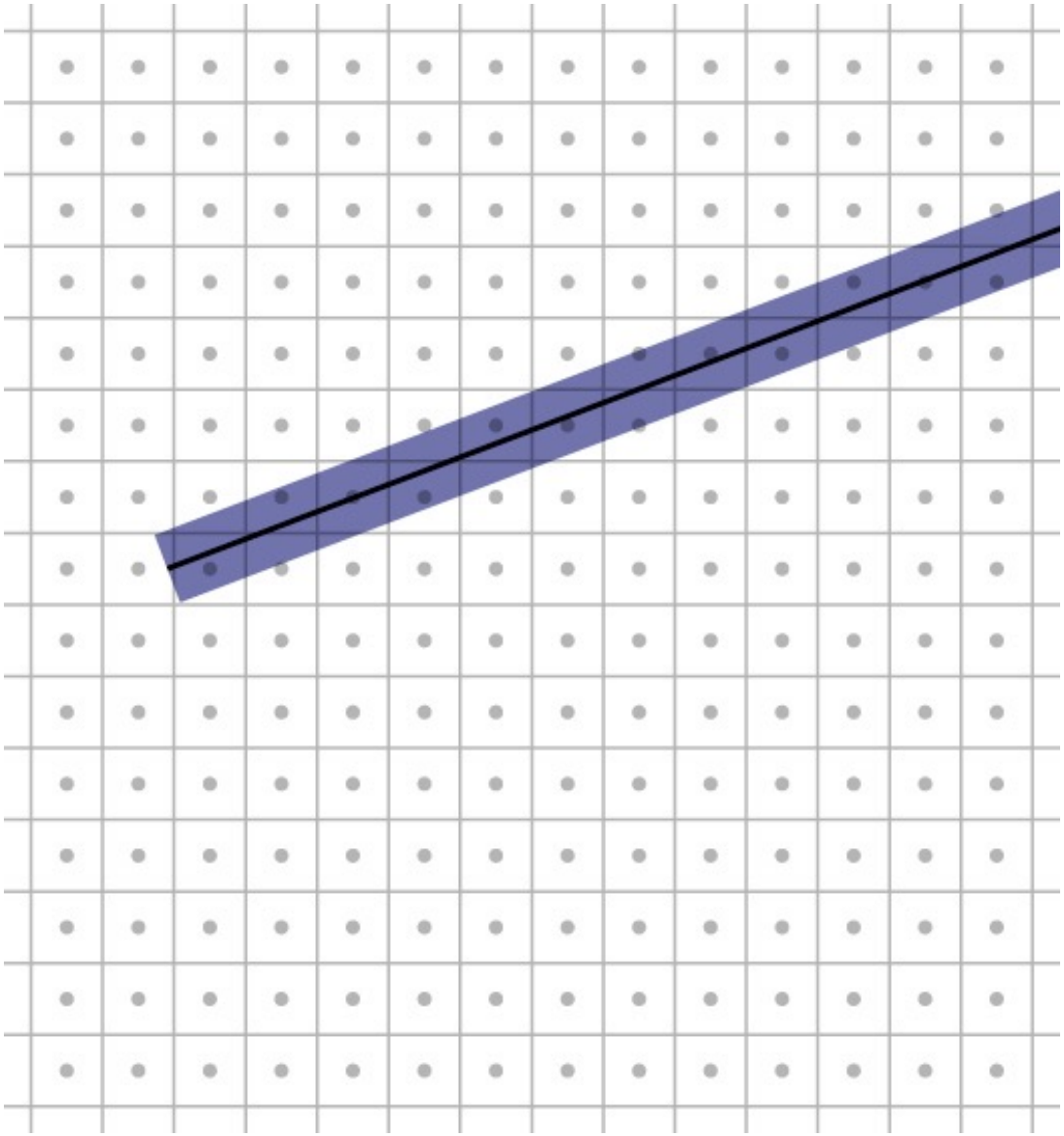
continuous image defined
by a bunch of black rectangles

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

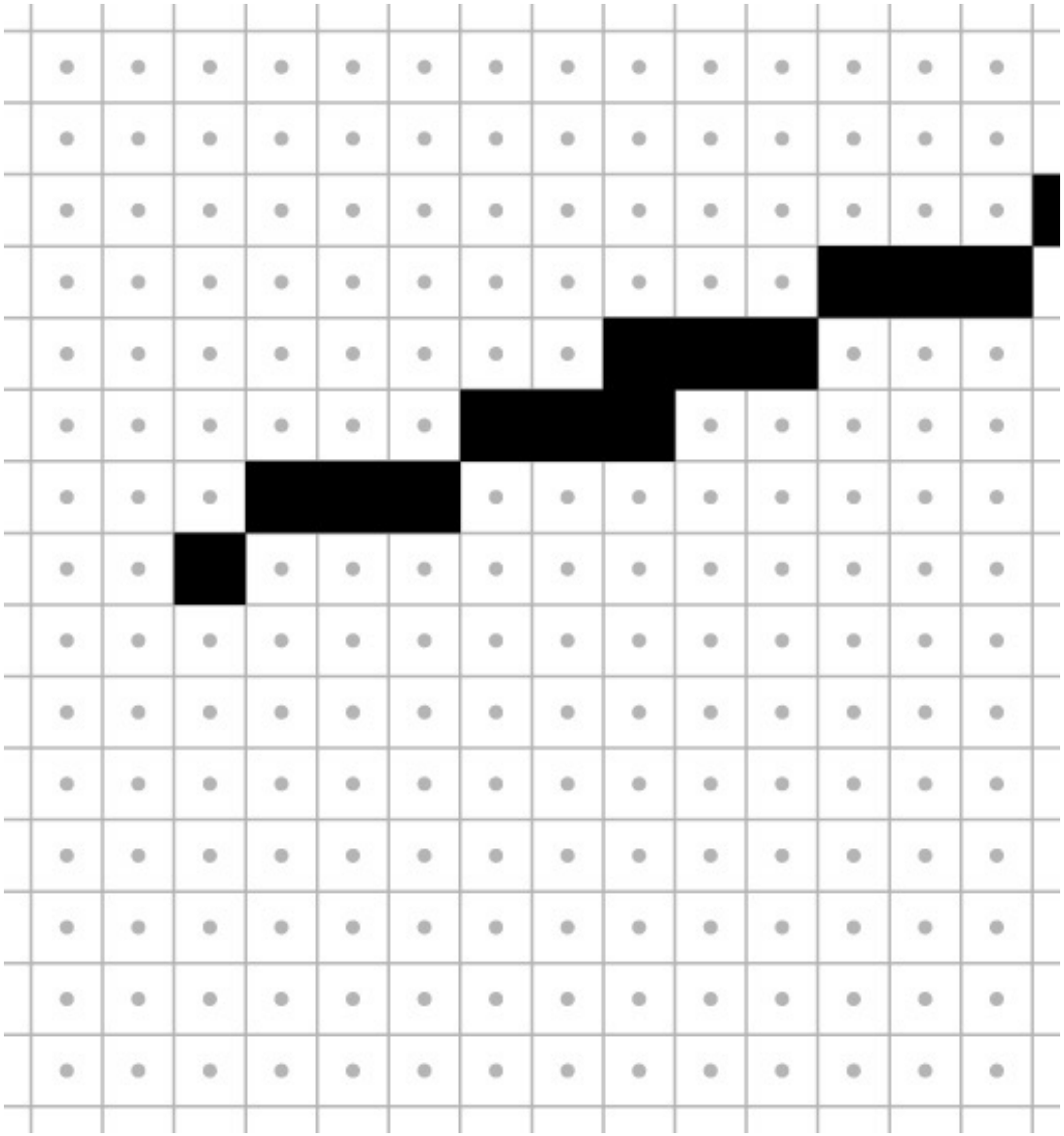
Rasterizing lines

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

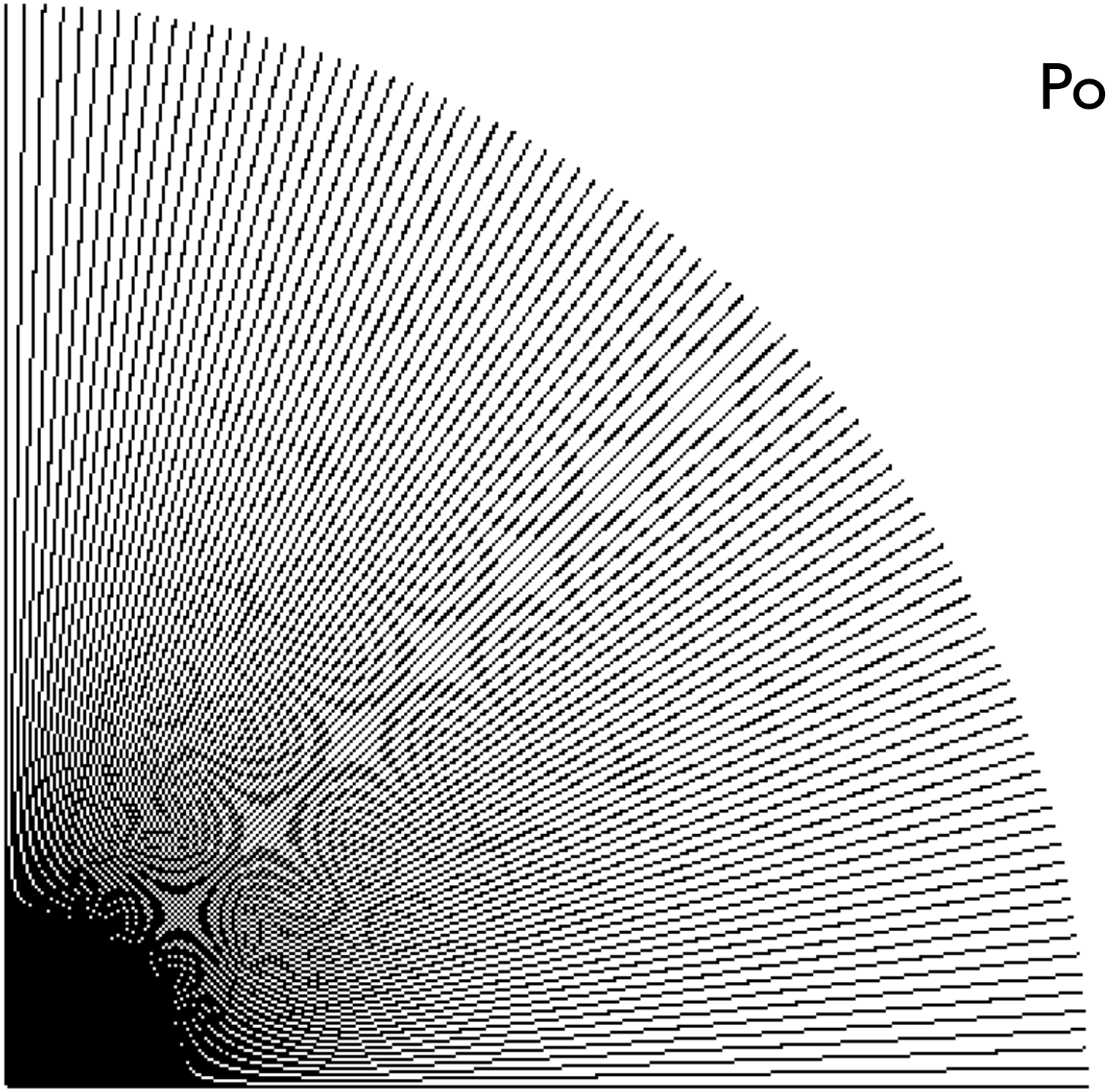


Point sampling

- 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

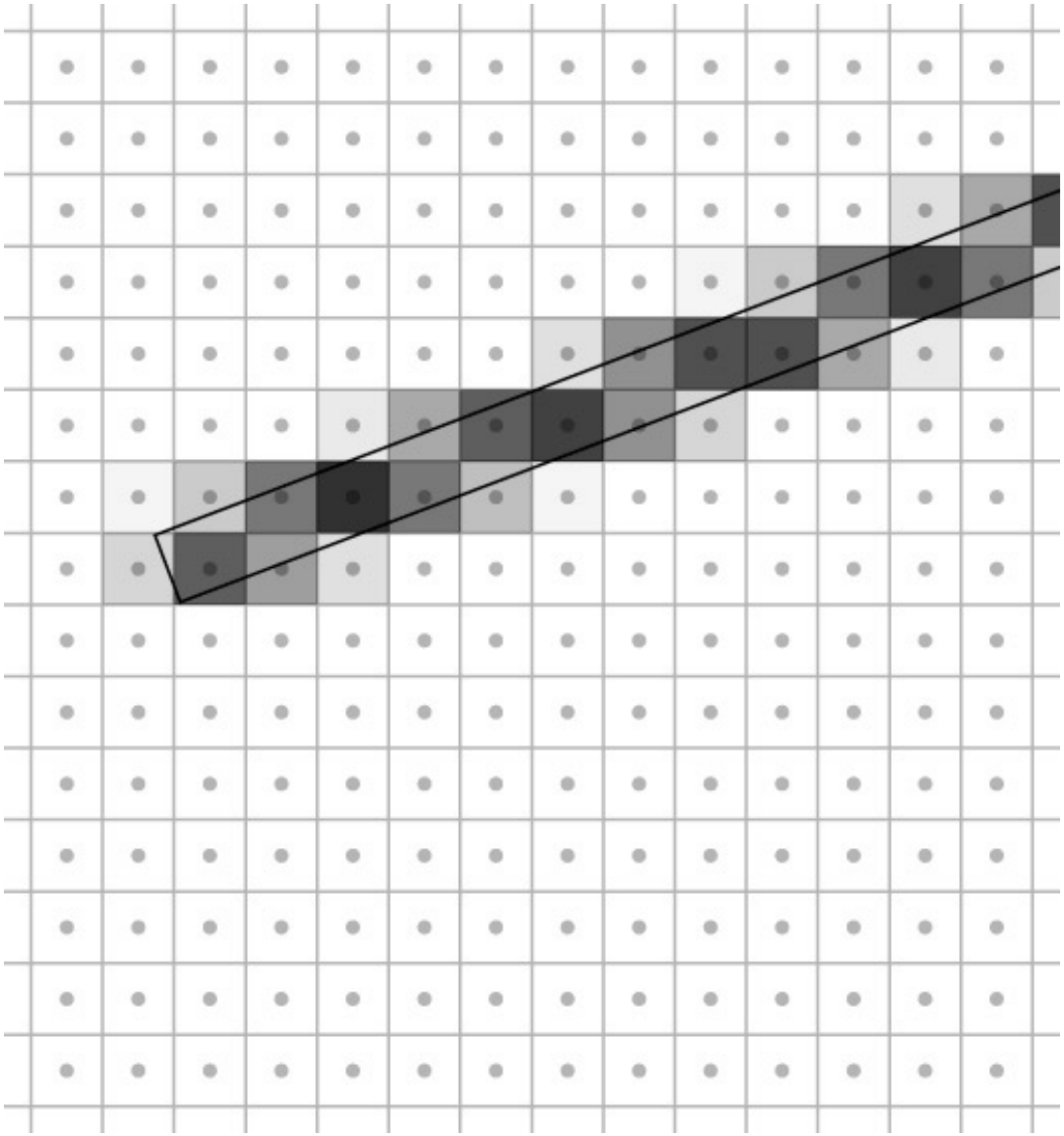


Antialiasing

- Point sampling makes an all-or-nothing choice in each pixel
  - therefore steps are inevitable when the choice changes
- 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

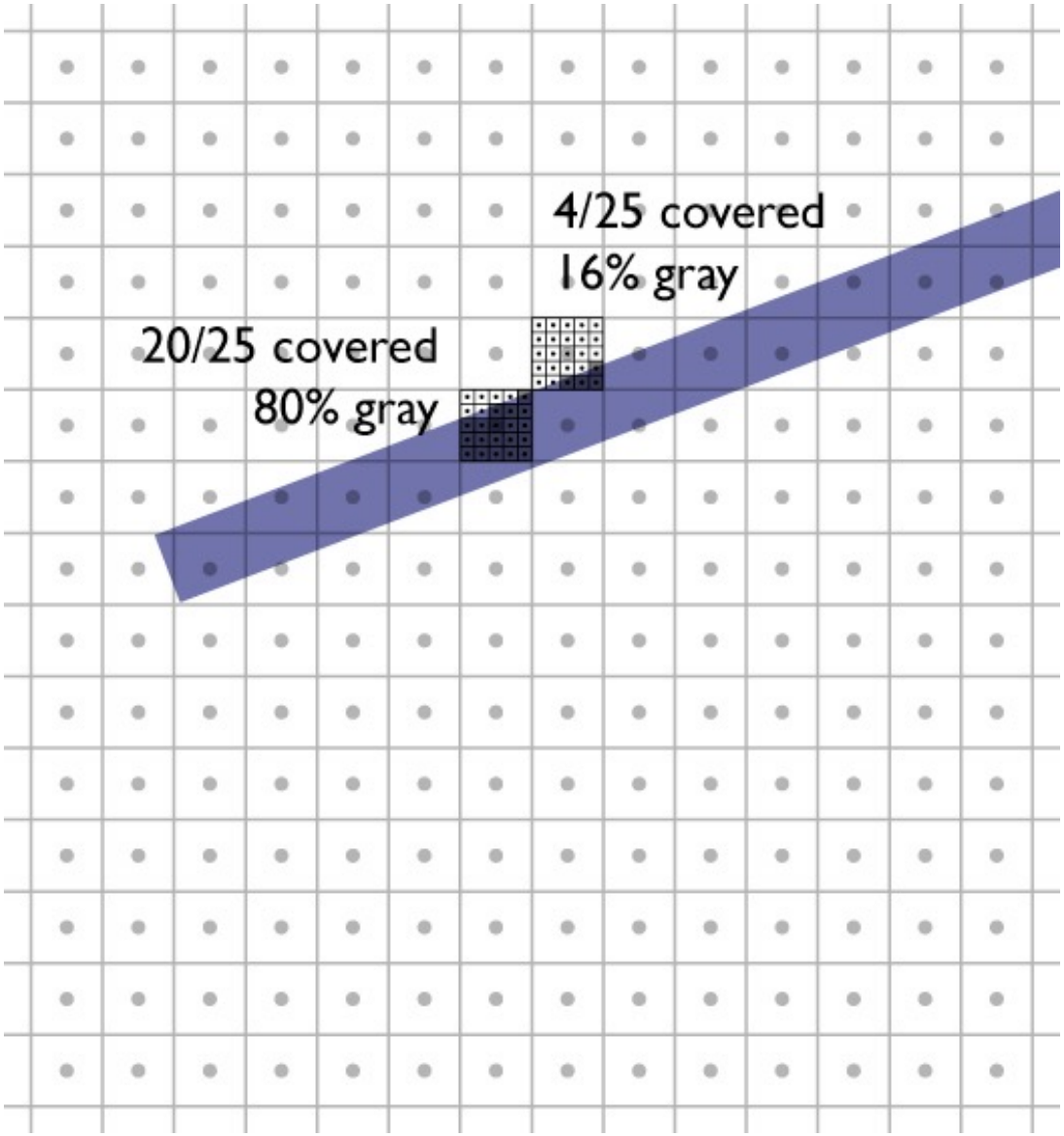


Box filtering

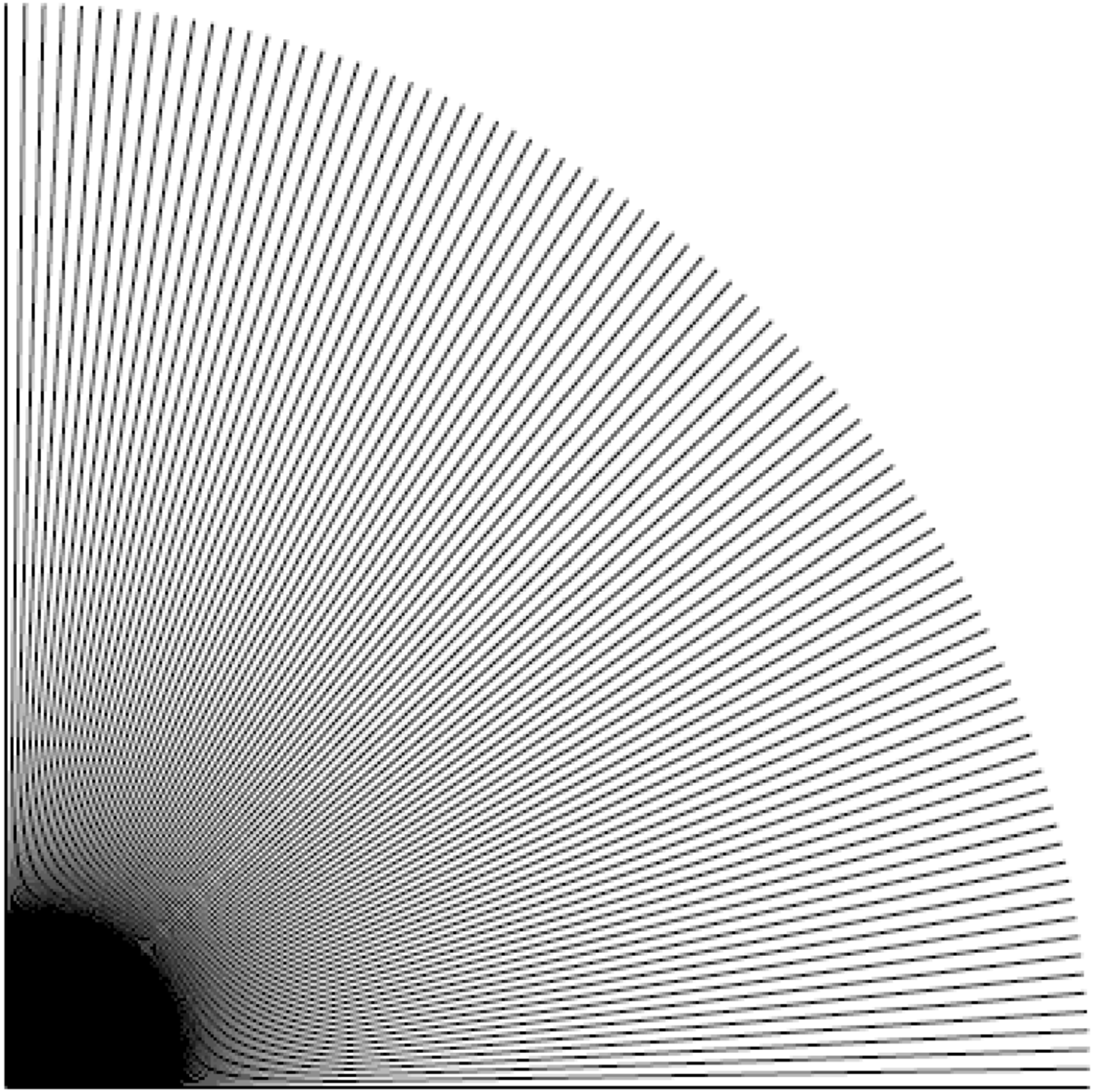
- Pixel intensity is proportional to area of overlap with square pixel area
- Also called "unweighted area averaging"

Box filtering by supersampling

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



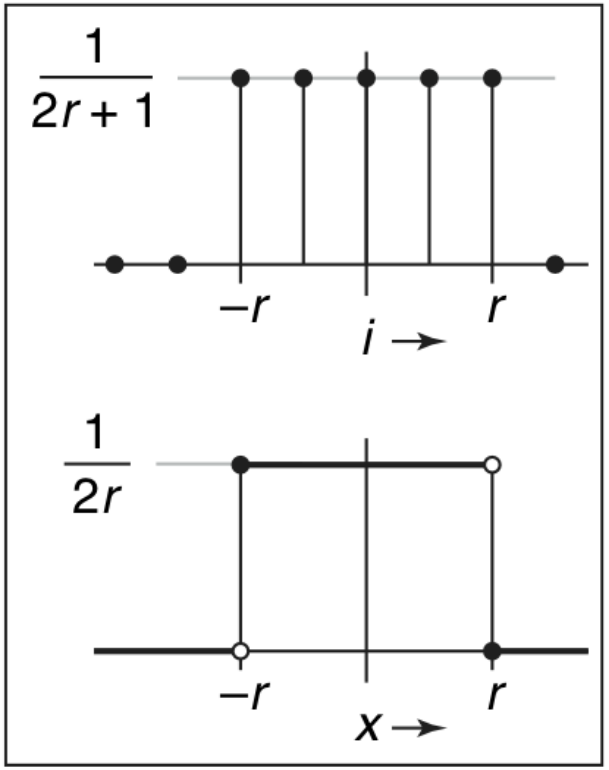
Box filtering in action



Box filter

$$a_{\text{box},r}[i] = \begin{cases} 1/(2r+1) & |i| \le r, \\ 0 & \text{otherwise.} \end{cases}$$

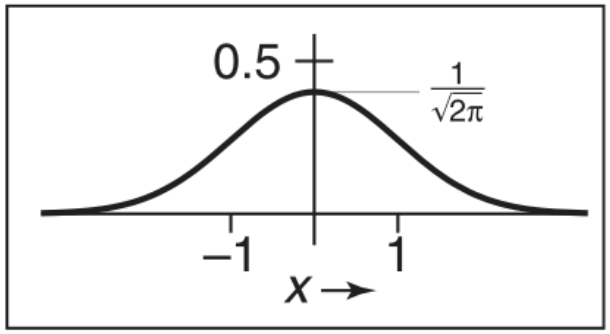
$$f_{\text{box},r}(x) = \begin{cases} 1/(2r) & -r \le x < r, \\ 0 & \text{otherwise.} \end{cases}$$



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)

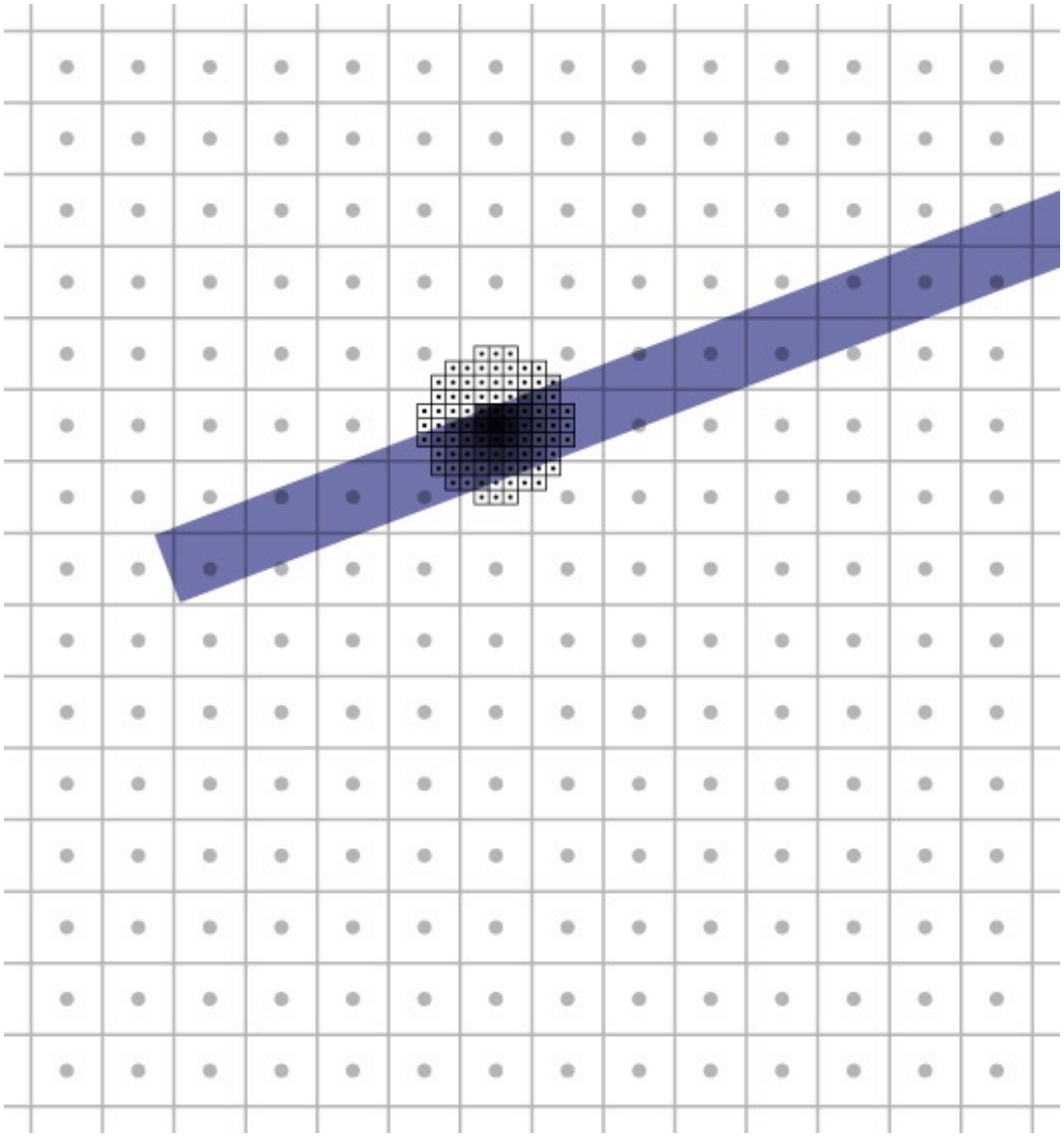
Gaussian filter



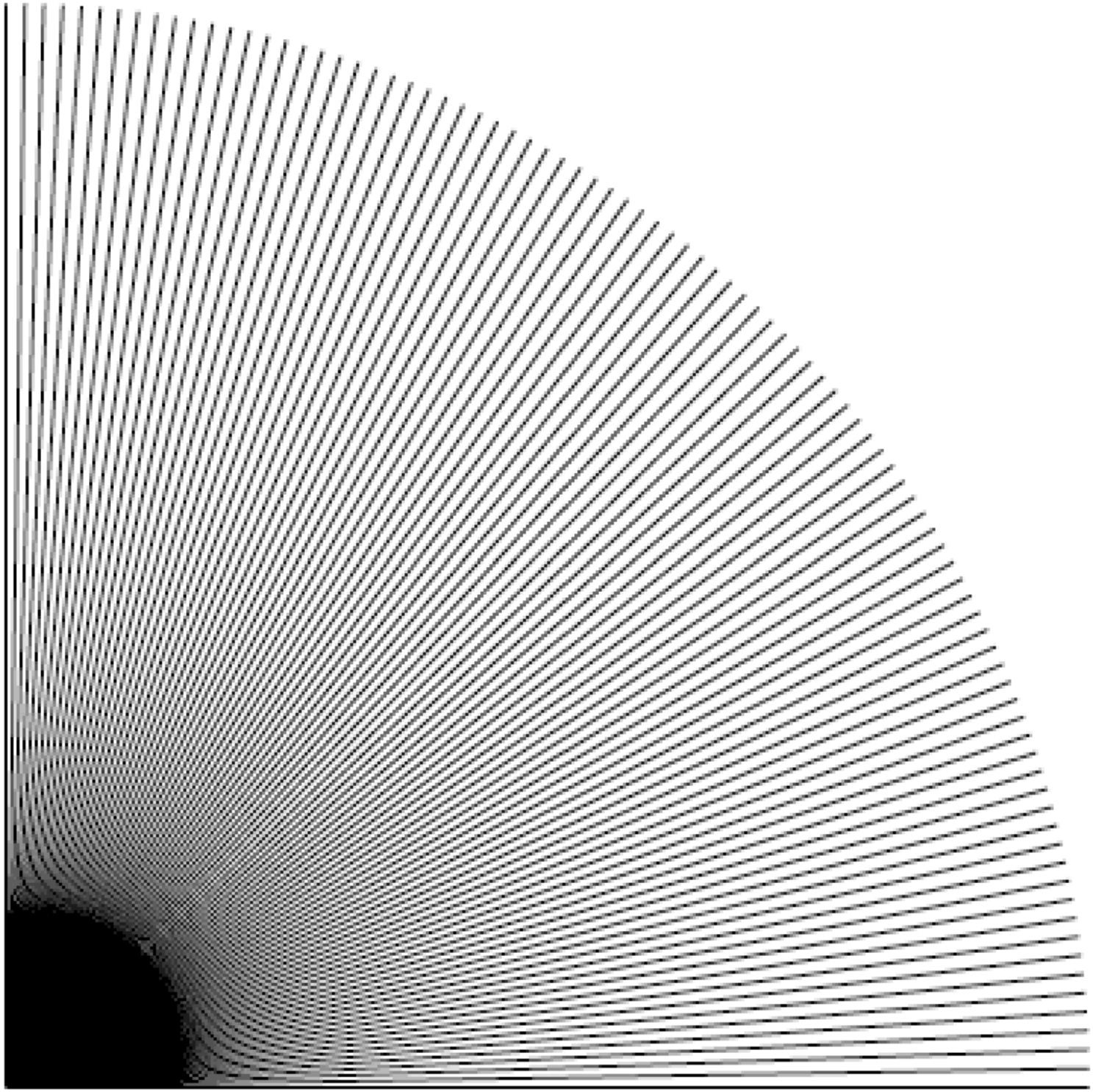
$$f_g(x) = \frac{1}{\sqrt{2\pi}} e^{-x^2/2}.$$

Weighted filtering by supersampling

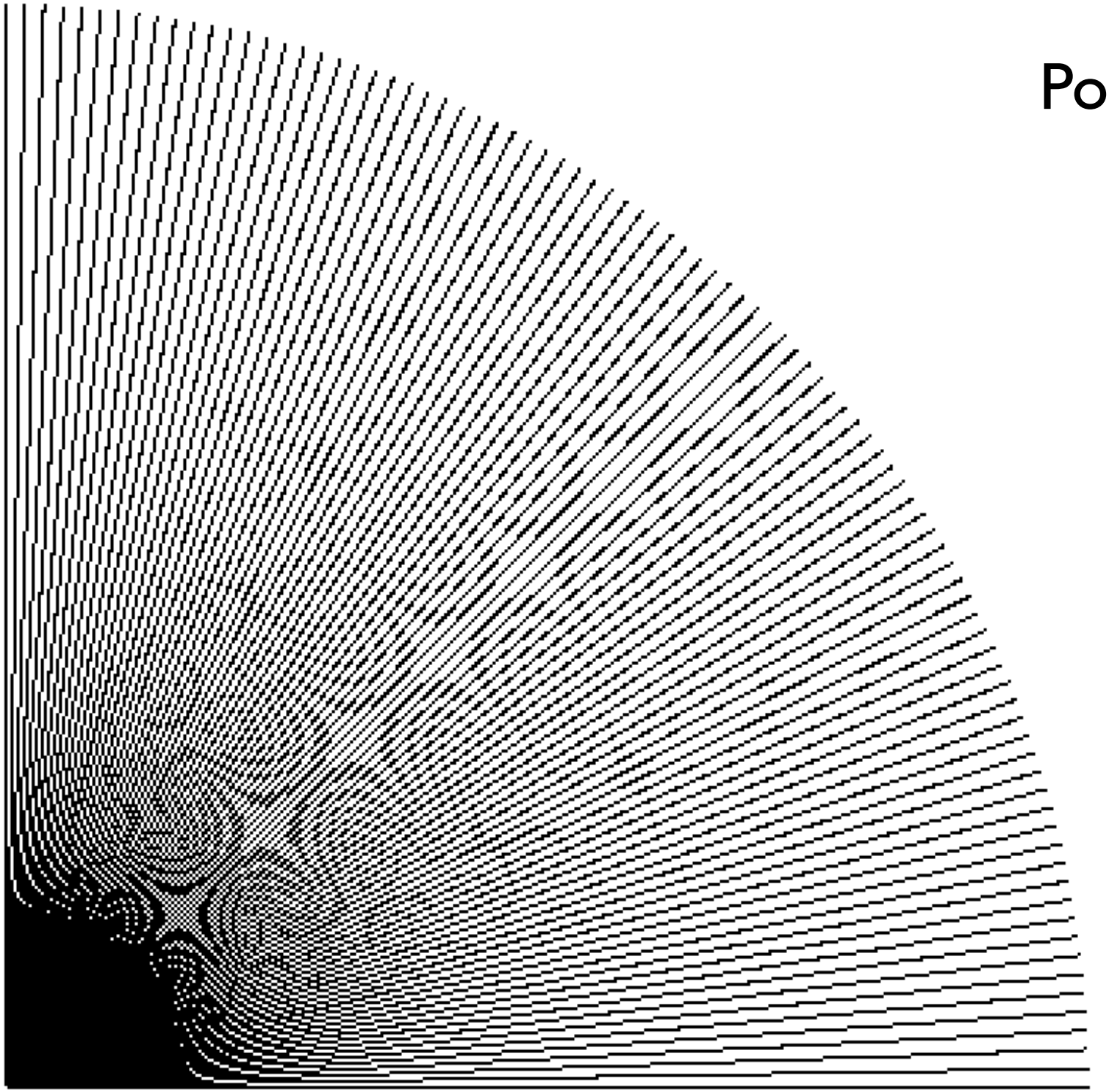
- Compute filtering integral by summing filter values for covered subpixels
- Simple, accurate
- But really slow



Gaussian filtering in action

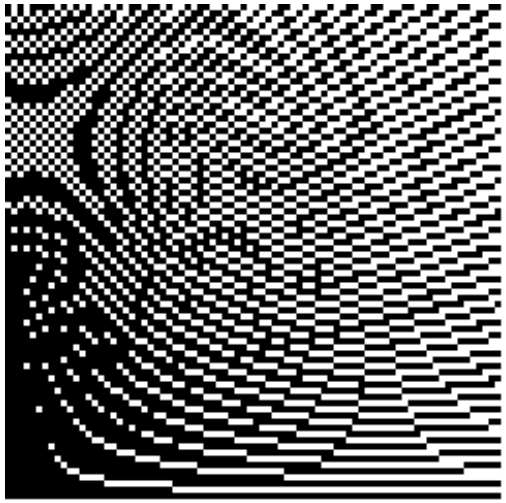


Box filtering in action

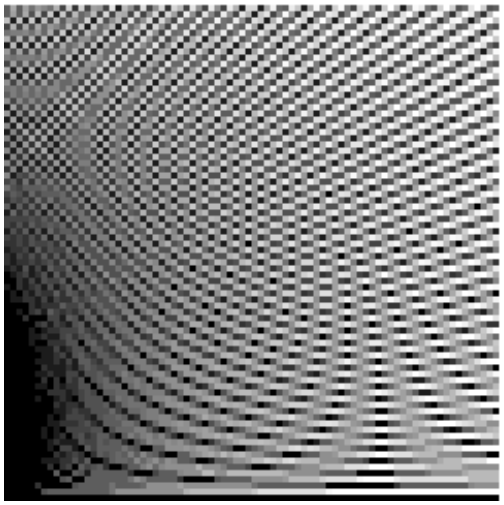


Point sampling in action

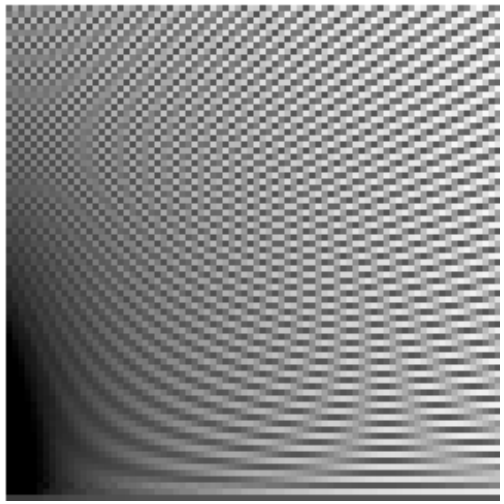
Filter comparison



Point sampling

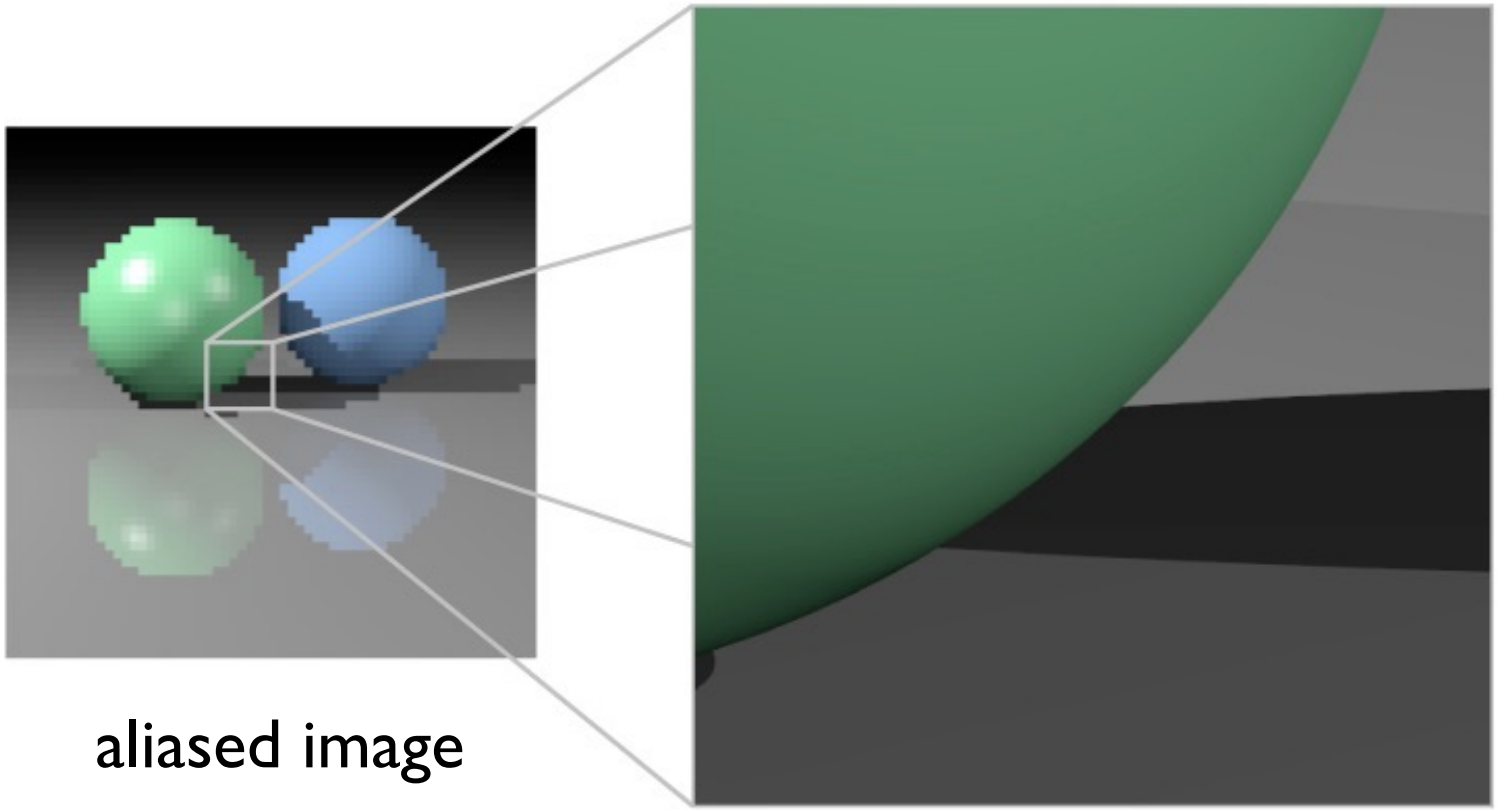


Box filtering



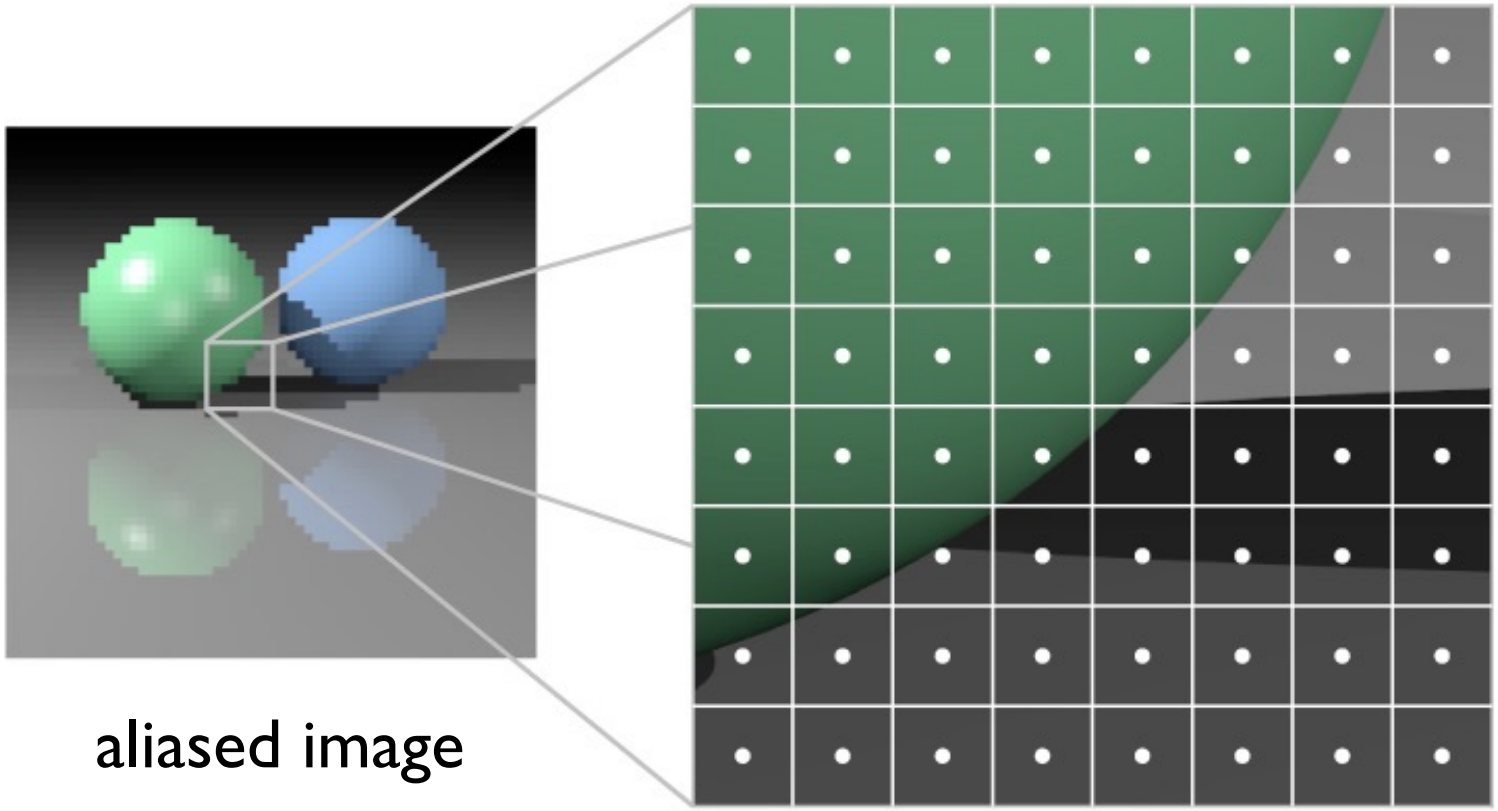
Gaussian filtering

Antialiasing in ray tracing



aliased image

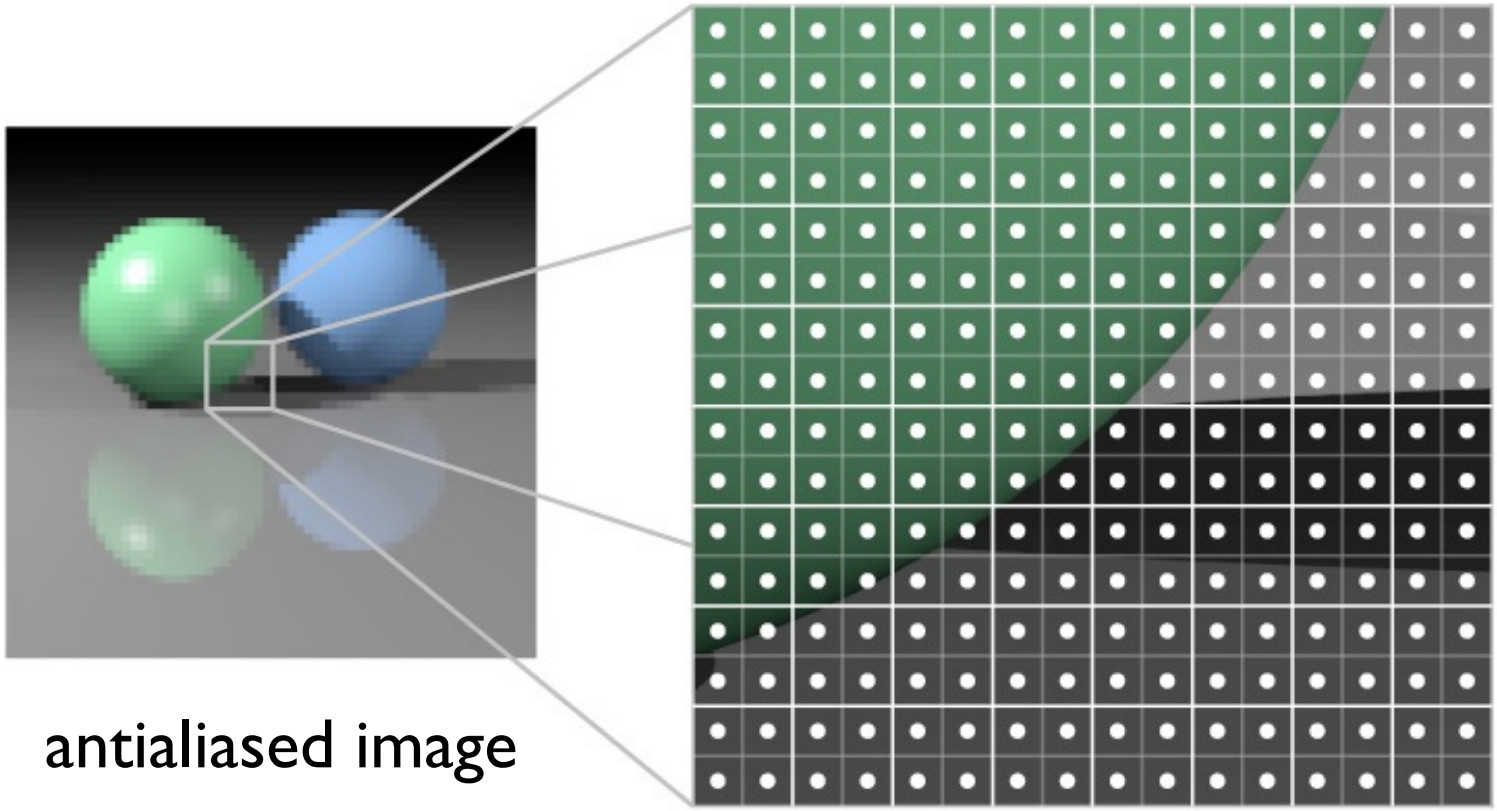
Antialiasing in ray tracing



aliased image

one sample per pixel

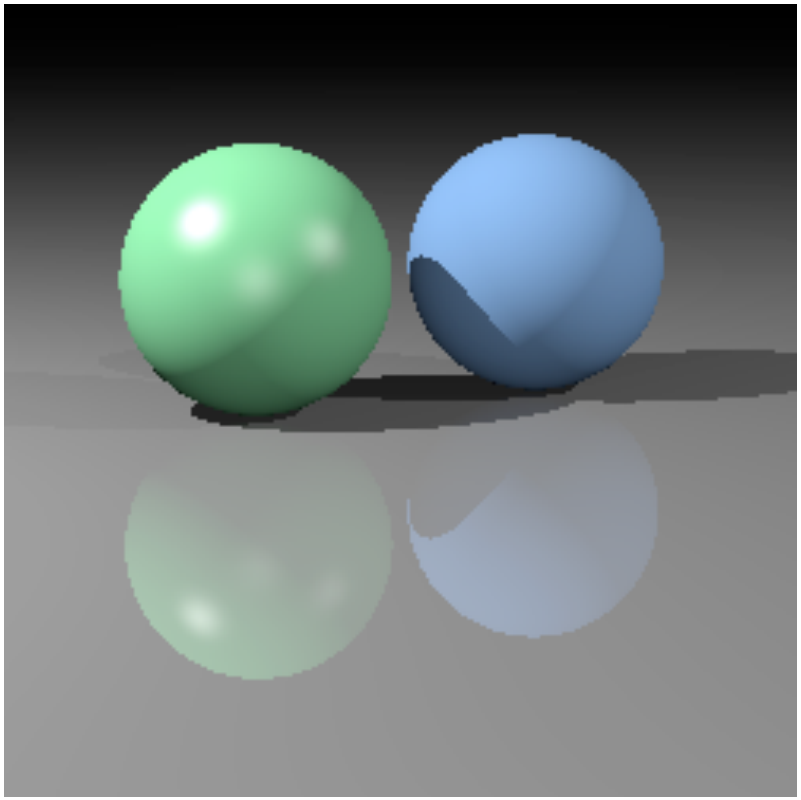
Antialiasing in ray tracing



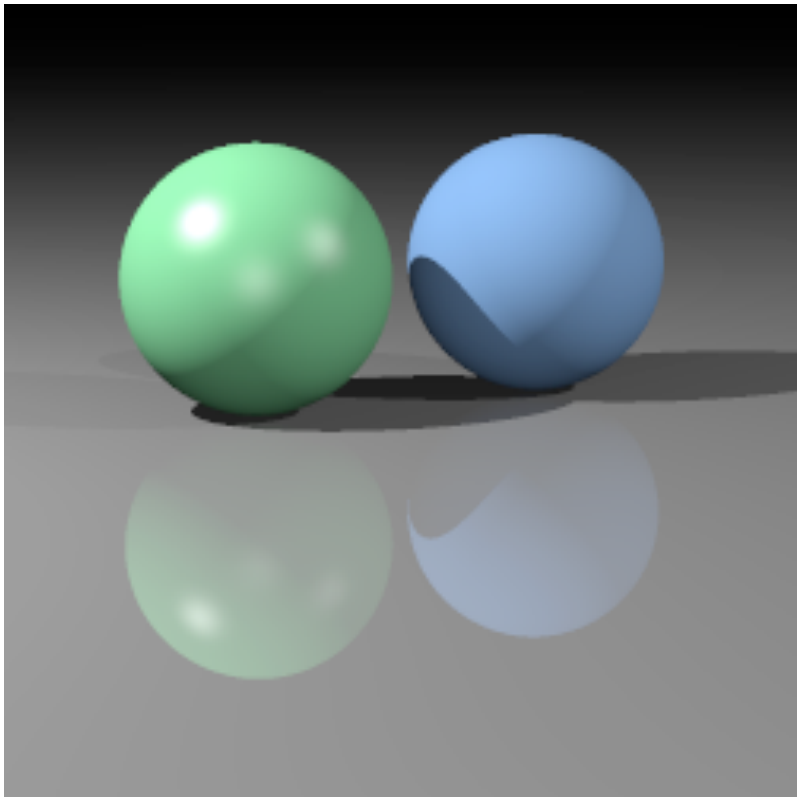
antialiased image

four samples per pixel

Antialiasing in ray tracing



one sample/pixel

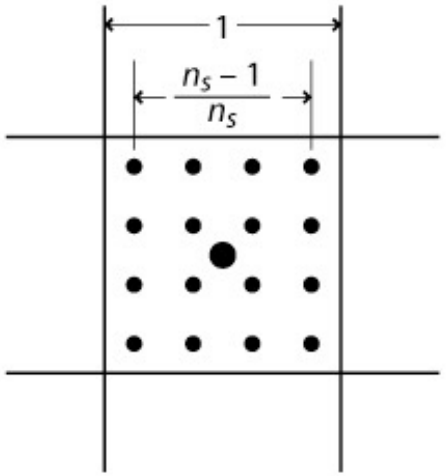


9 samples/pixel

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));
   }
```

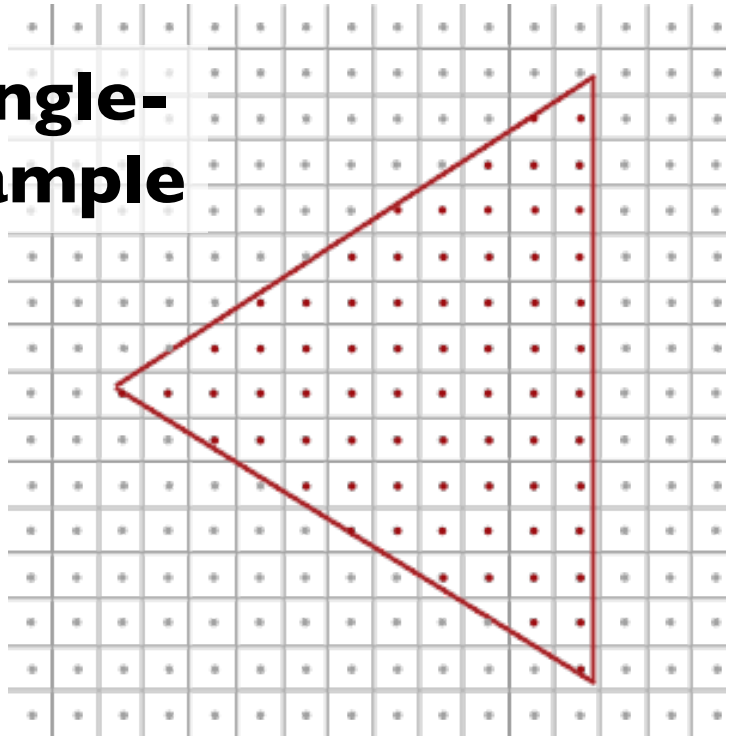

Supersampling vs. multisampling

- 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

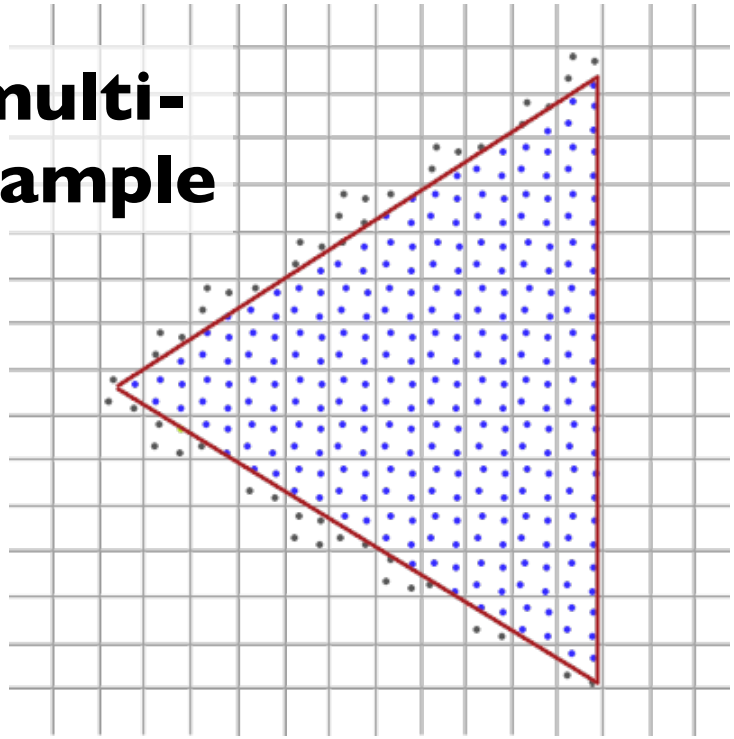
Multisample rasterization

- 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

single-sample



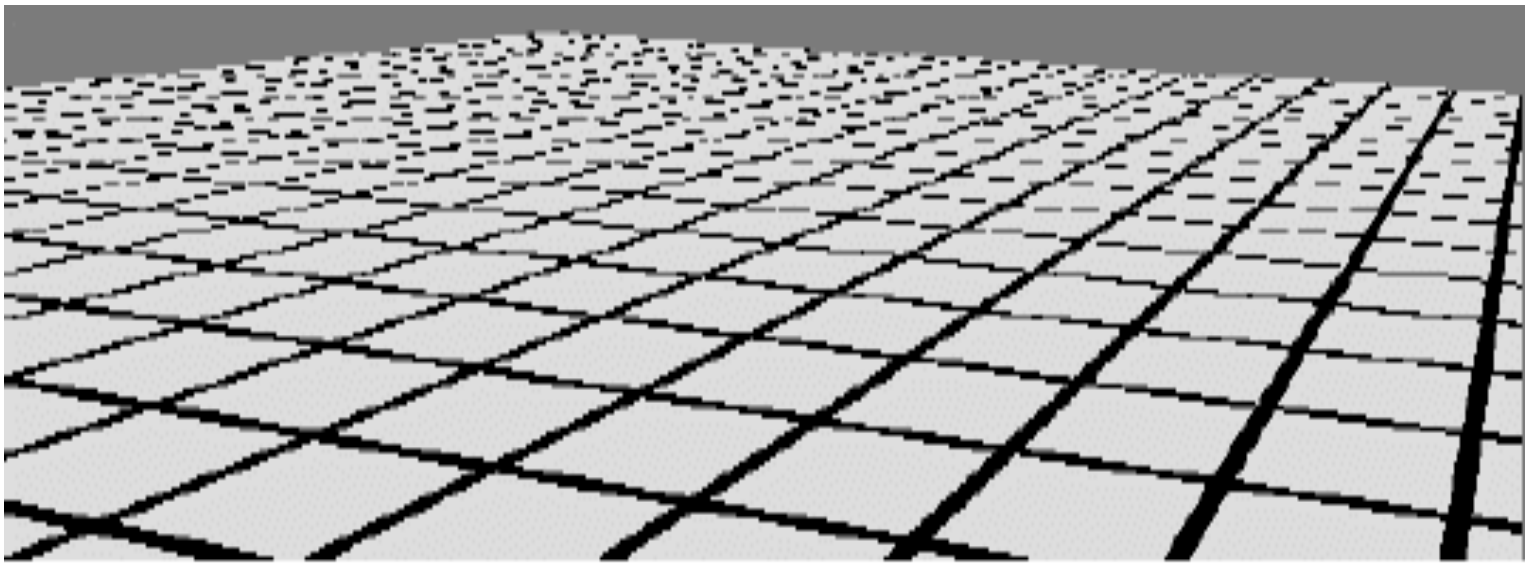
multi-sample



[http://www.learnopengl.com]

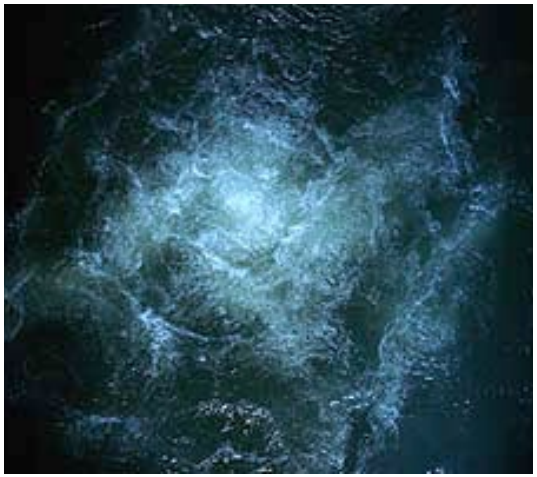
Antialiasing in textures

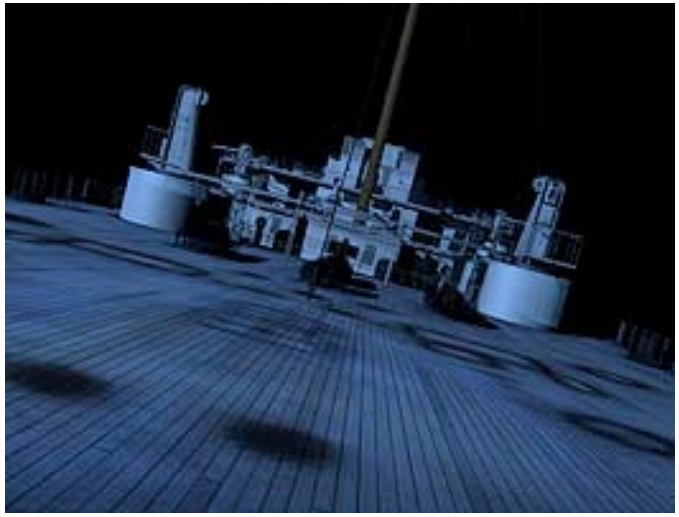
- Even with multisampling, we still only evaluate textures once per fragment
- Need to filter the texture somehow!
  - perspective produces very high image frequencies
  - (MIP Mapping)



Compositing

Compositing

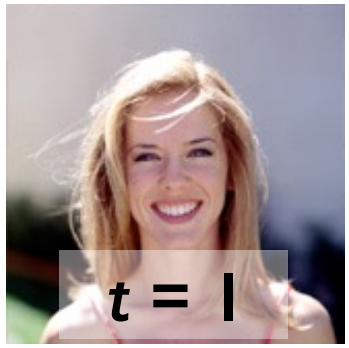




[*Titanic* ; DigitalDomain; vfxhq.com]

Combining images

- Often useful to 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

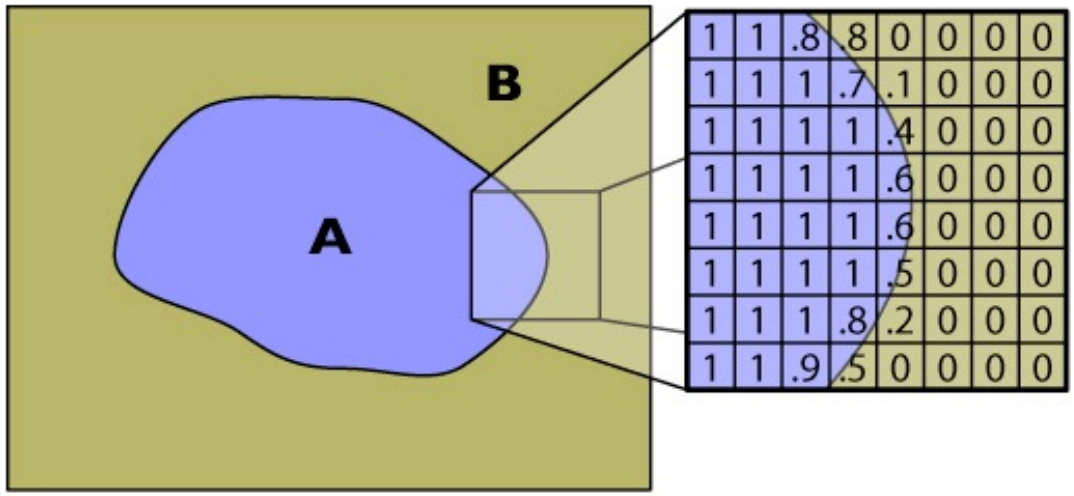
- First idea: store one bit per pixel
  - answers question "is this pixel part of the foreground?"

[Chuang et al. / Corel]

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

Partial pixel coverage

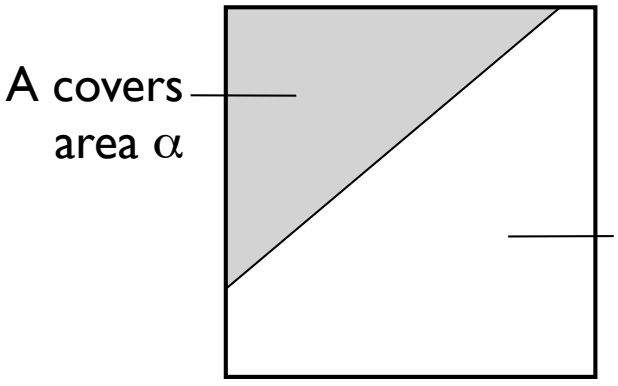
- The problem: pixels near boundary are not strictly foreground or background



  - 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 α



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

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

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

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

  - this is 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]

