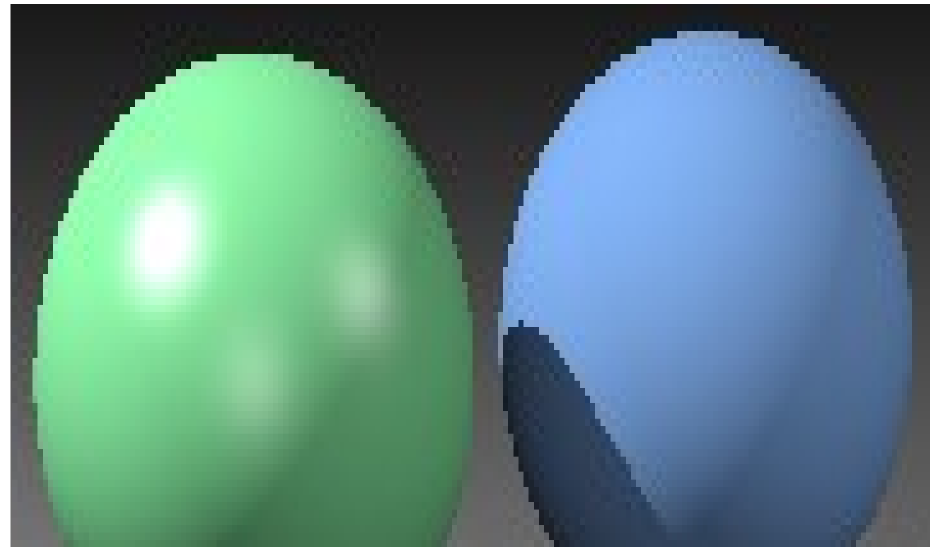
Antialiasing & Compositing

CS465 Lecture 17

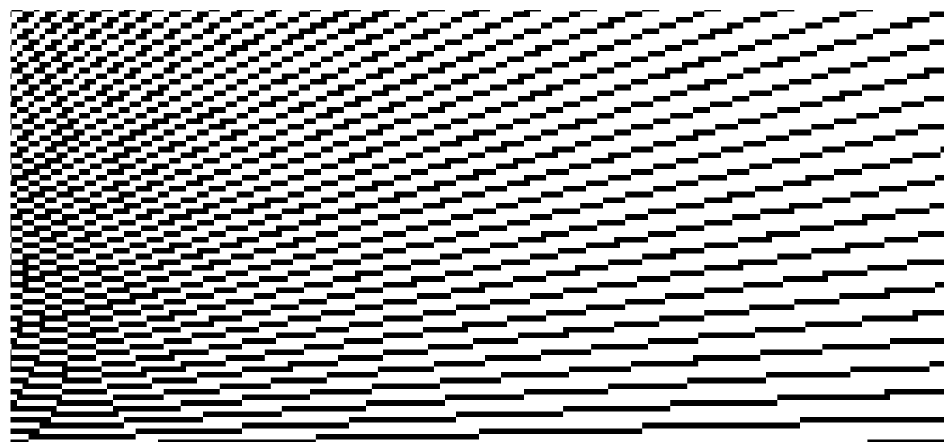
Aliasing

point sampling a continuous image:

continuous image defined by ray tracing procedure



continuous image defined by a bunch of black rectangles

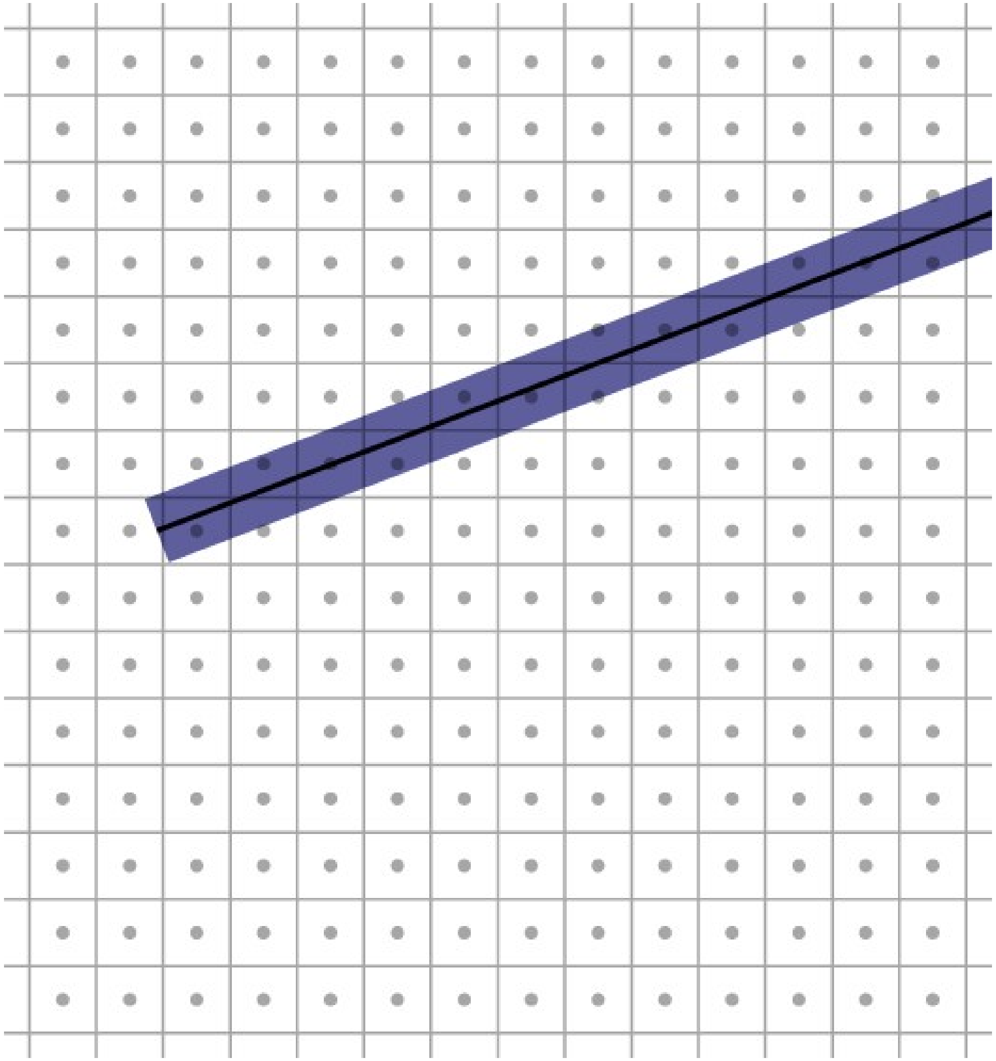


Antialiasing

- A name for techniques to prevent aliasing
- In image generation, we need to lowpass filter
  - 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

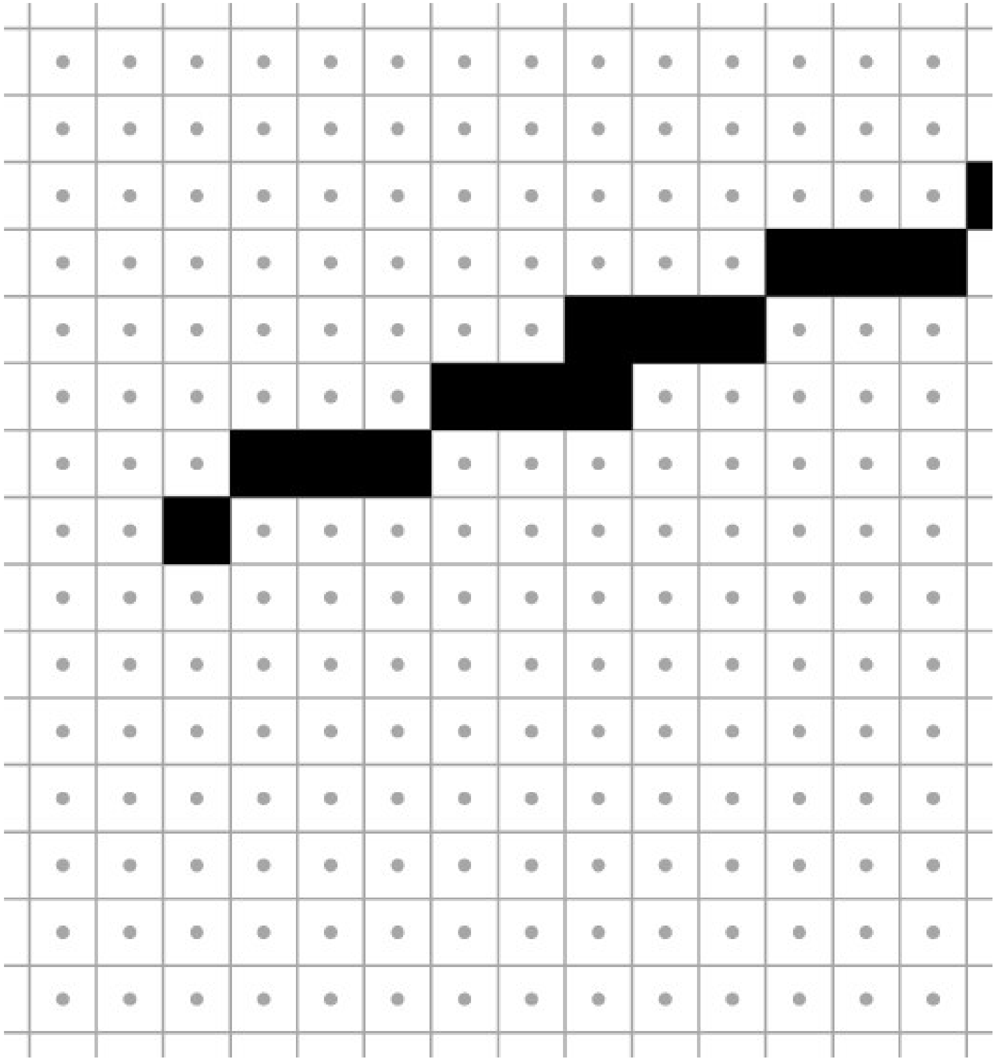
Rasterizing lines

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

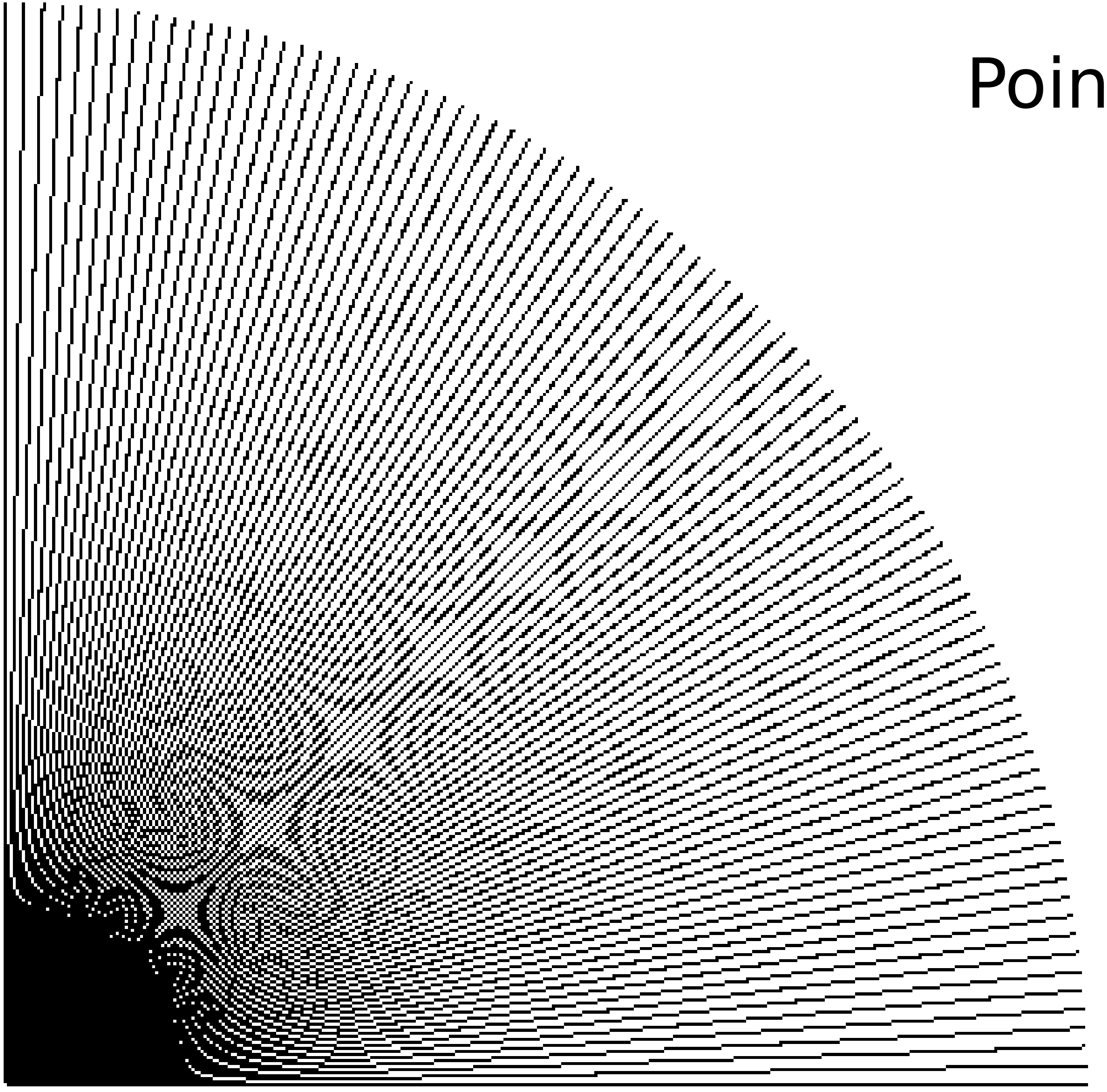


Point sampling

- Approximate rectangle by drawing all pixels whose centers fall within the line
- Problem: all-or-nothing leads to jaggies

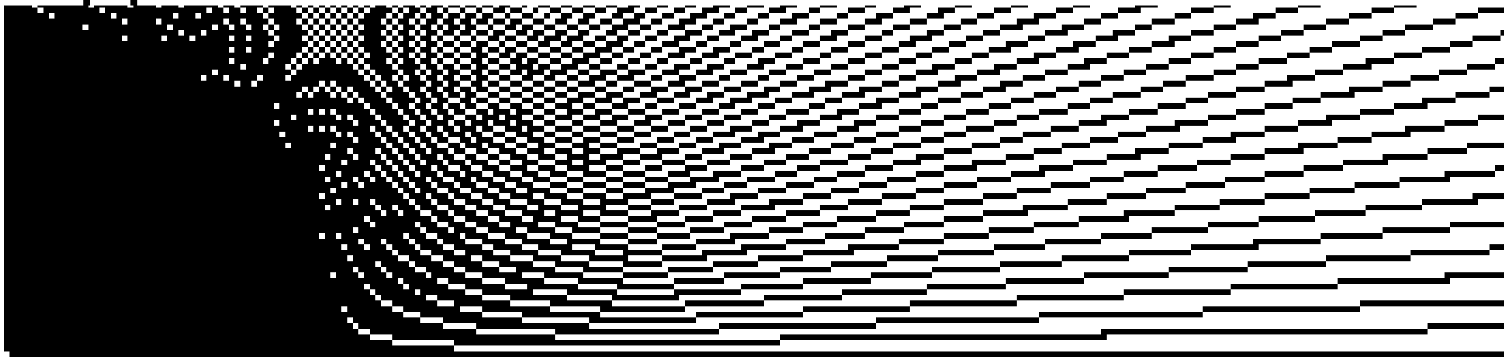


Point sampling in action



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!

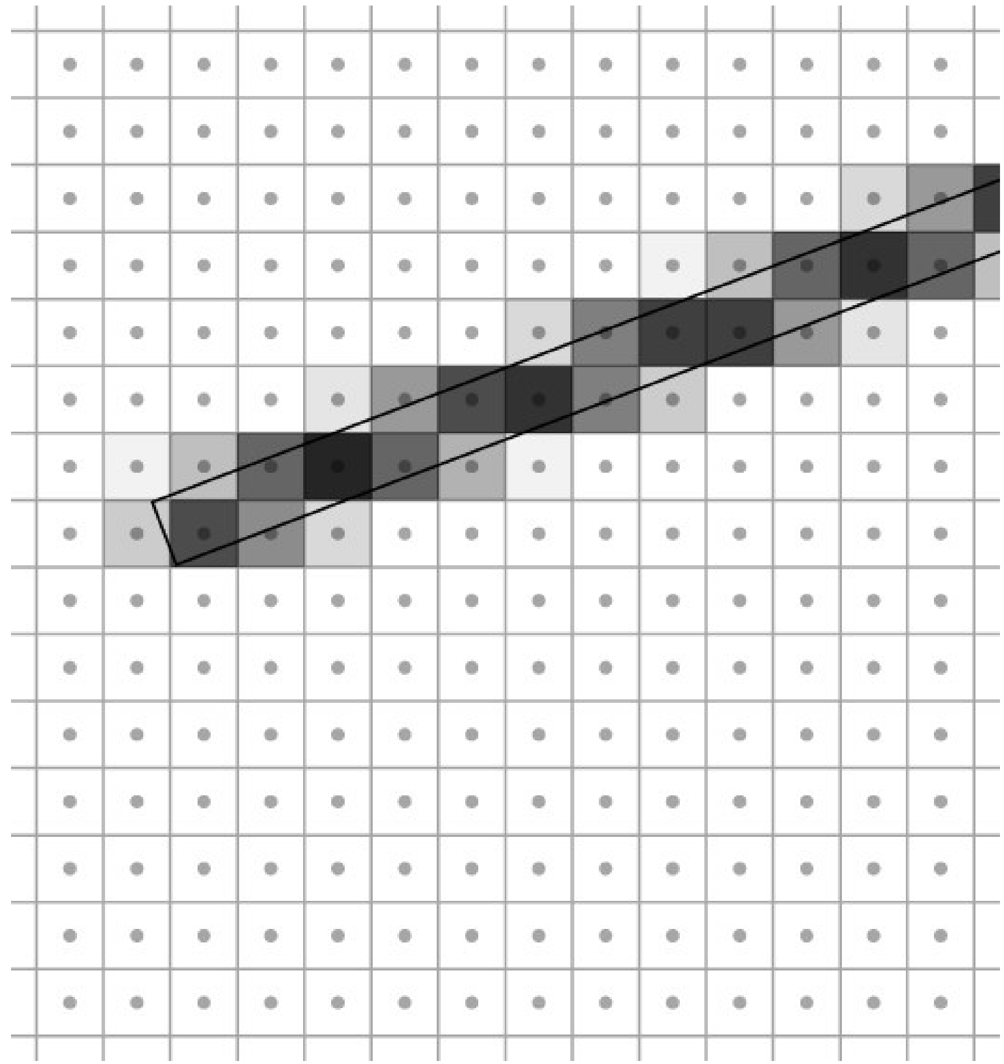


Antialiasing

- Point sampling makes an all-or-nothing choice in each pixel
  - therefore steps are inevitable when the choice changes
  - discontinuities are BAD in computer graphics
- 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.

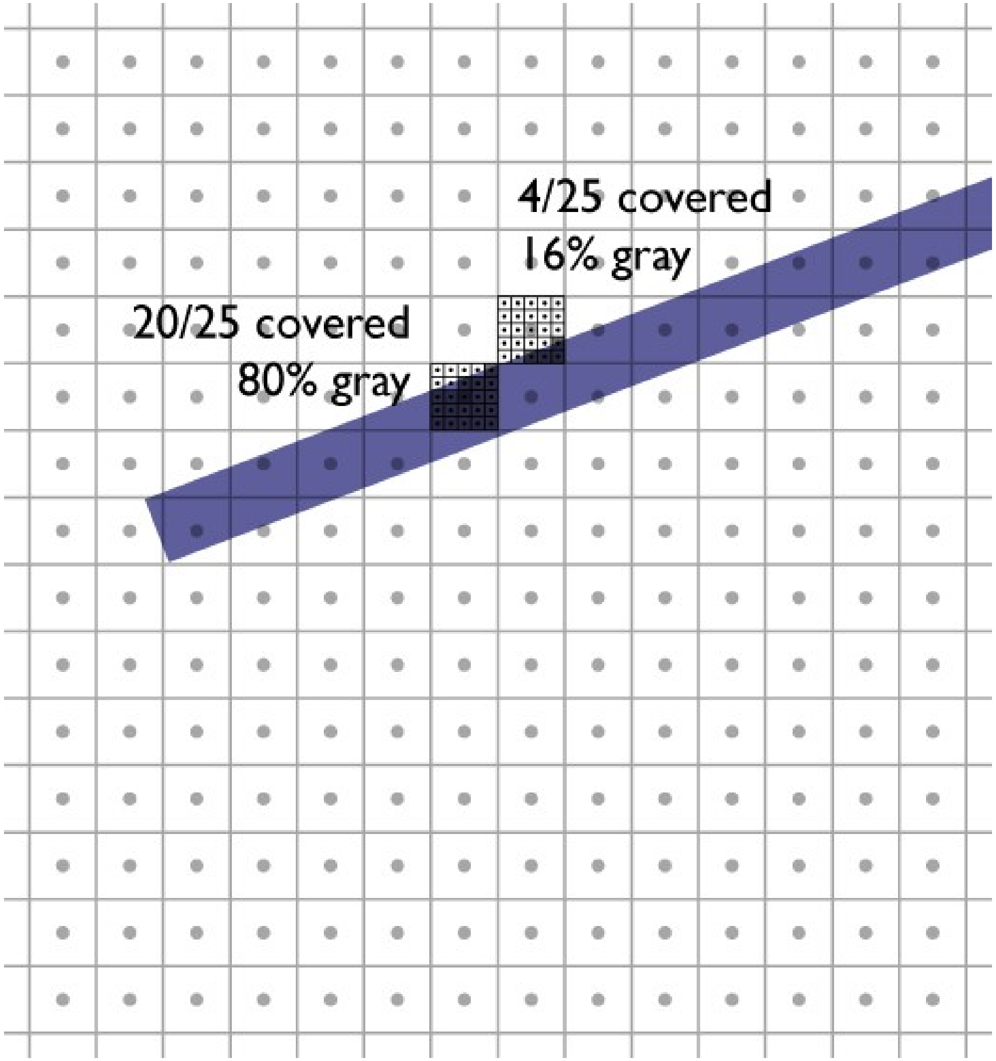


Box filtering

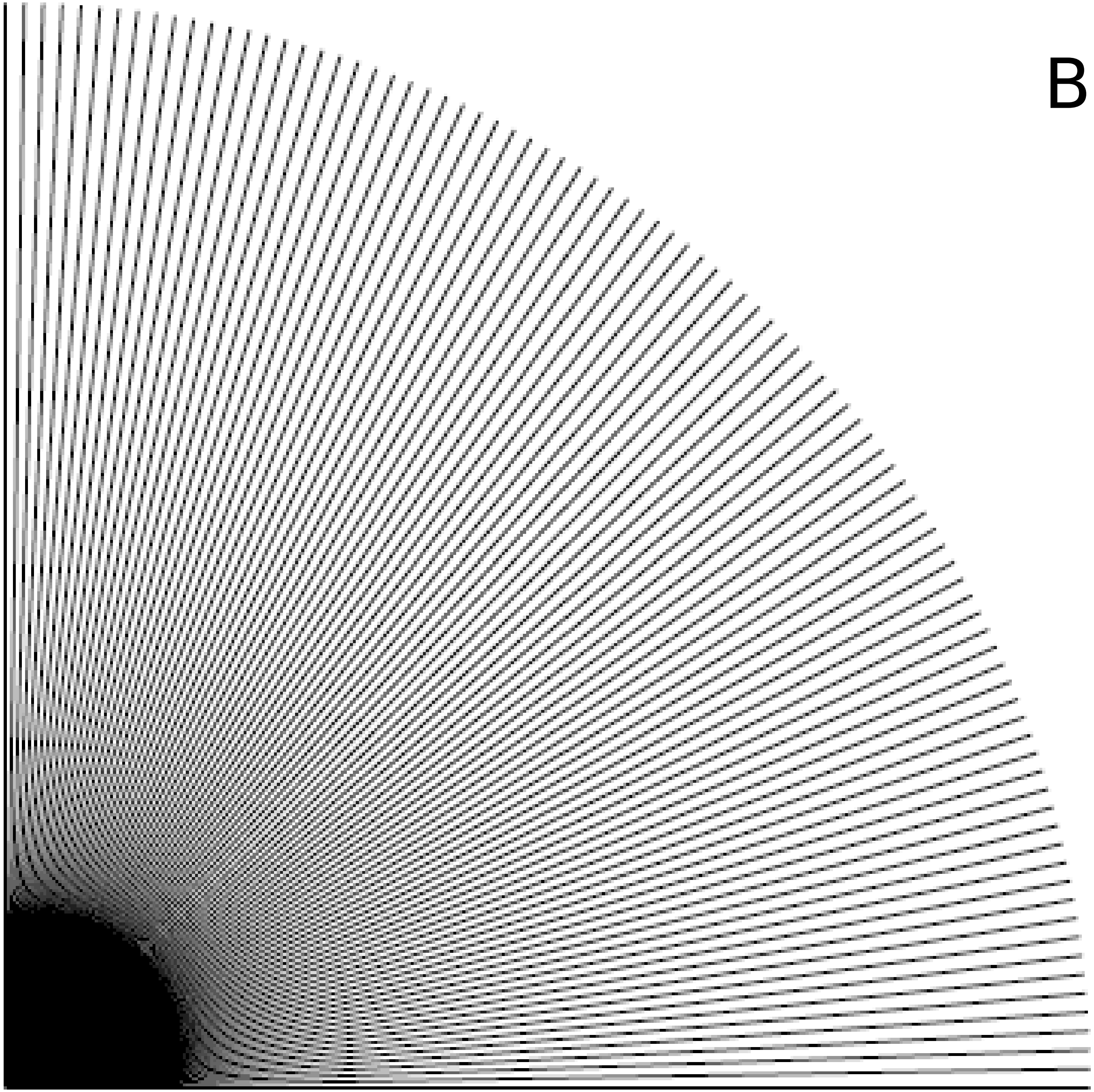
- Pixel intensity is proportional to area of overlap with square pixel area
- Also called “unweighted area averaging”

Box filtering by supersampling

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



Box filtering in action

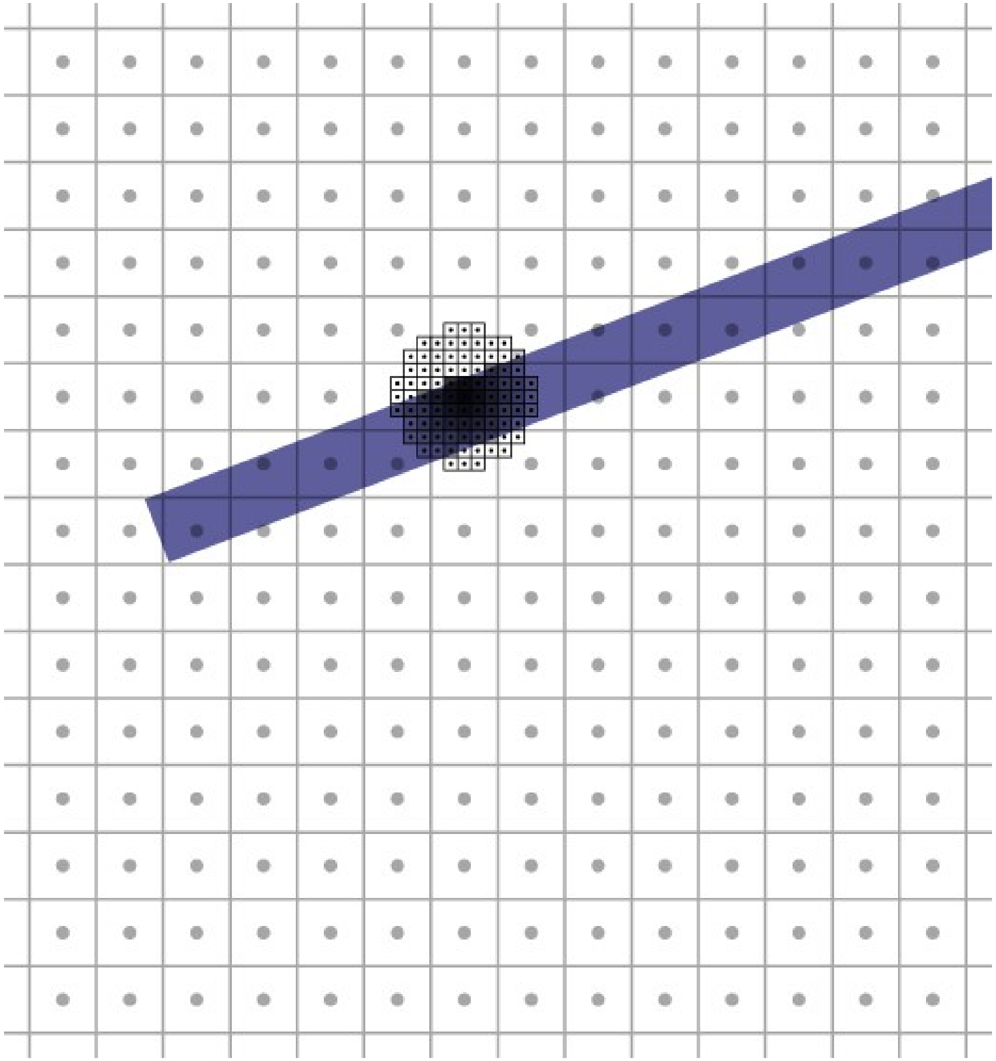


Weighted filtering

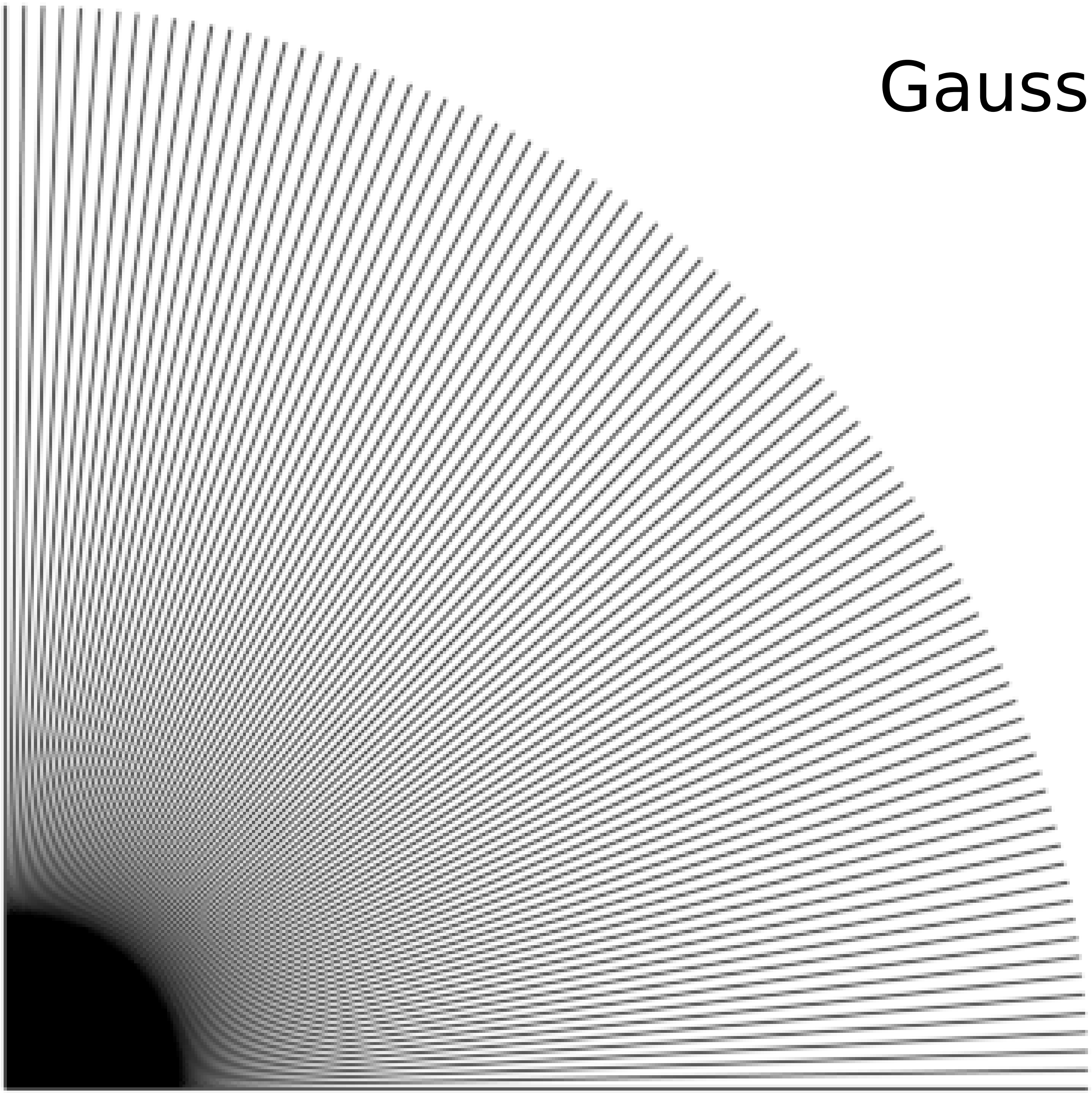
- Box filtering problem: treats area near edge same as area near center
  - results in pixel turning on “too abruptly”
- Alternative: weight area by a smoother filter
  - unweighted averaging corresponds to using a box function
  - sharp edges mean high frequencies
    - so want a filter with good extinction for higher freqs.
  - a gaussian is a popular choice of smooth filter
  - important property: normalization (unit integral)

Weighted filtering by supersampling

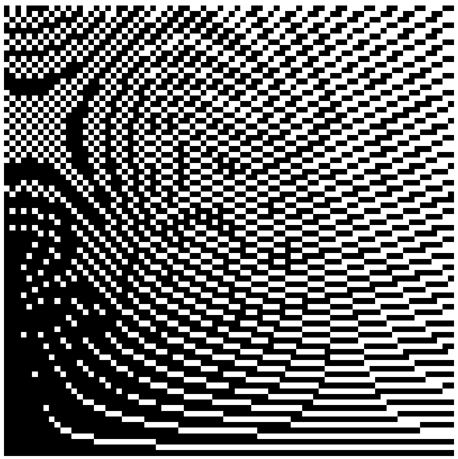
- Compute filtering integral by summing filter values for covered subpixels
- Simple, accurate
- But really slow



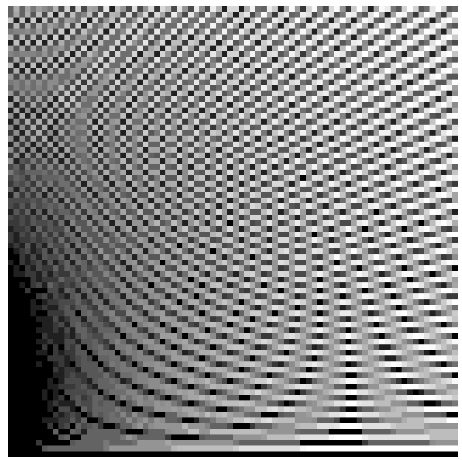
Gaussian filtering in action



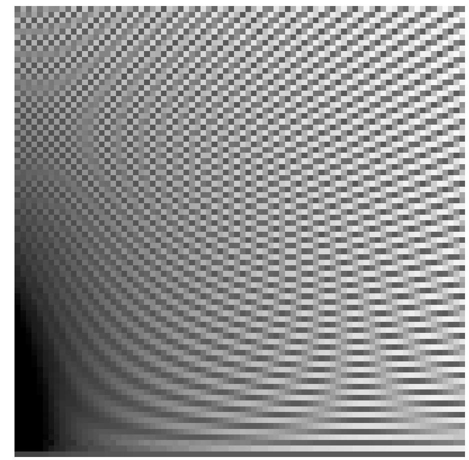
Filter comparison



Point sampling



Box filtering

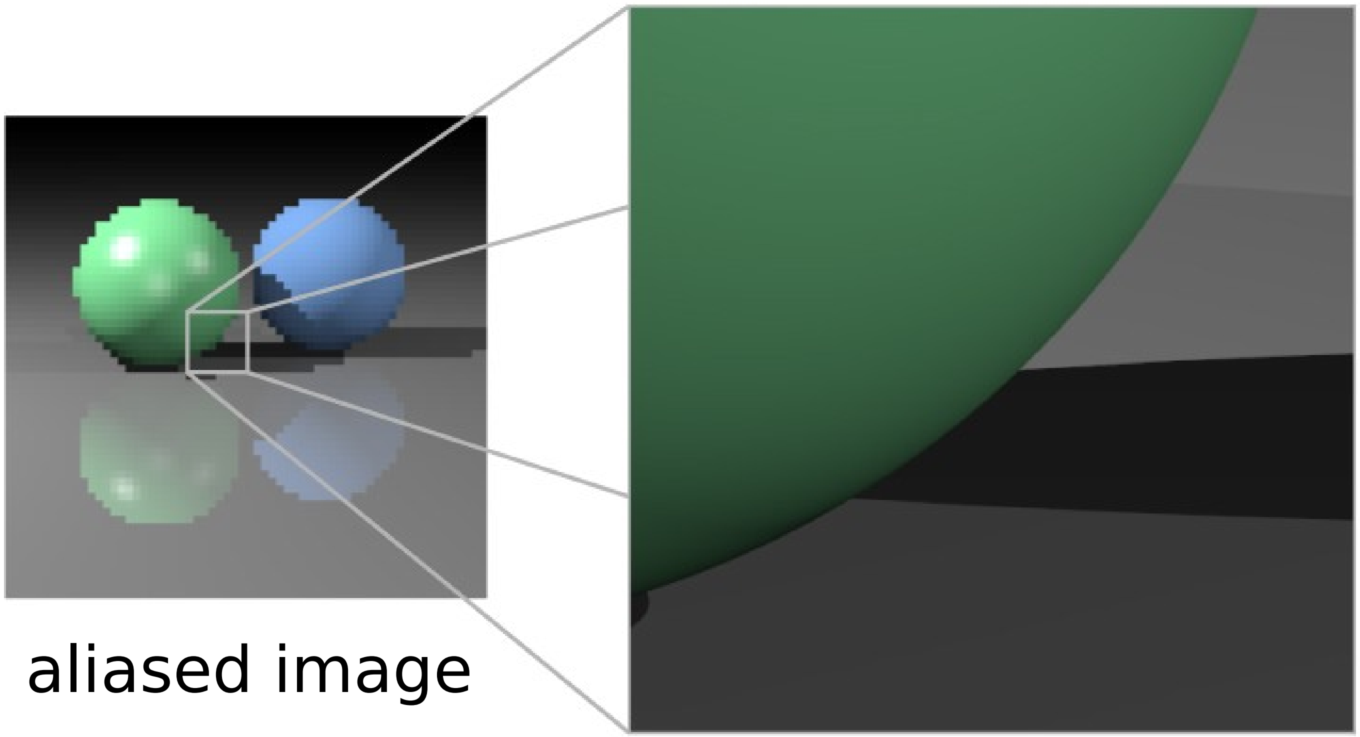


Gaussian filtering

Antialiasing and resampling

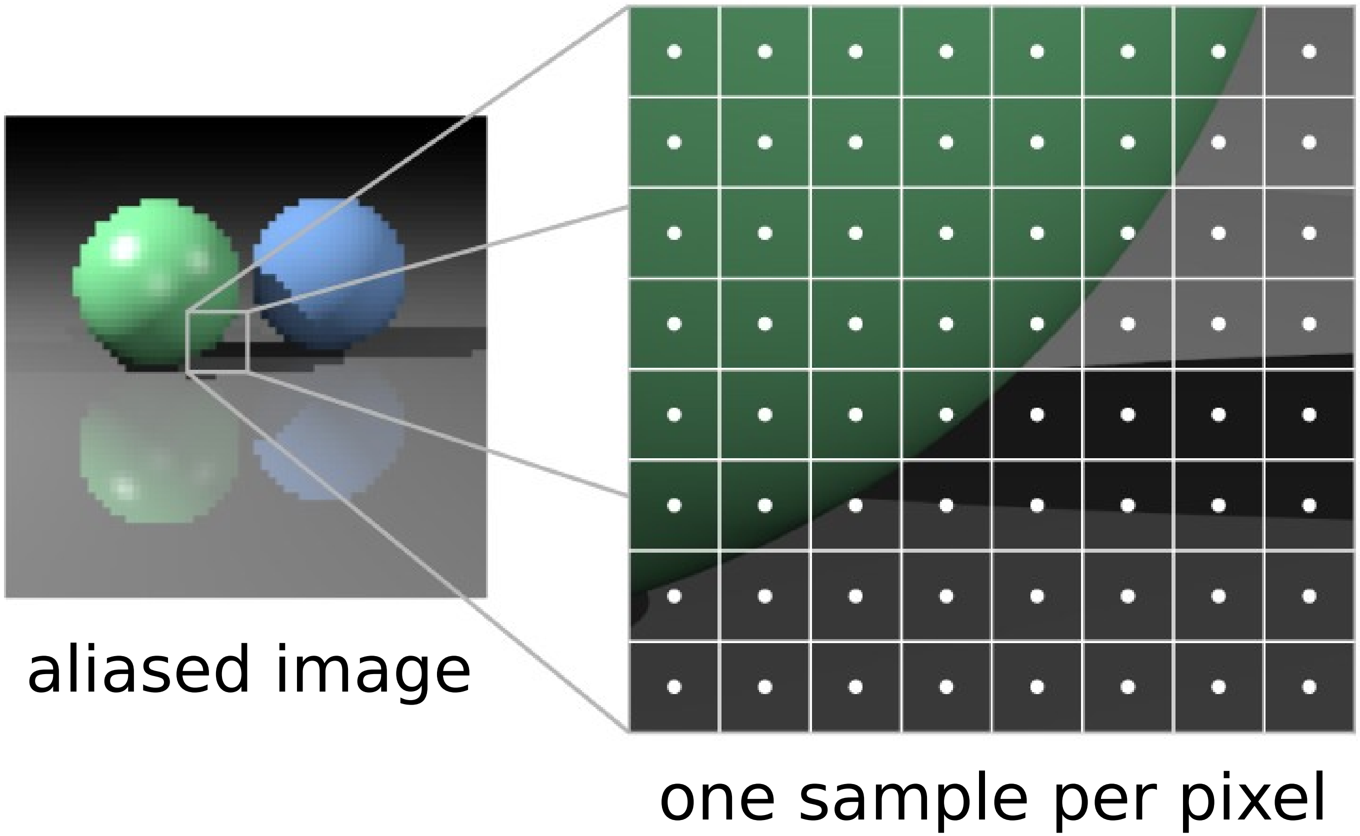
- Antialiasing by regular supersampling is *the same as* rendering a larger image and then resampling it to a smaller size
- Convolution of filter with high-res image produces an estimate of the area of the primitive in the pixel.
- So we can re-think this
  - one way: we're computing area of pixel covered by primitive
  - another way: we're computing average color of pixel
    - this way generalizes easily to arbitrary filters, arbitrary images

Antialiasing in ray tracing

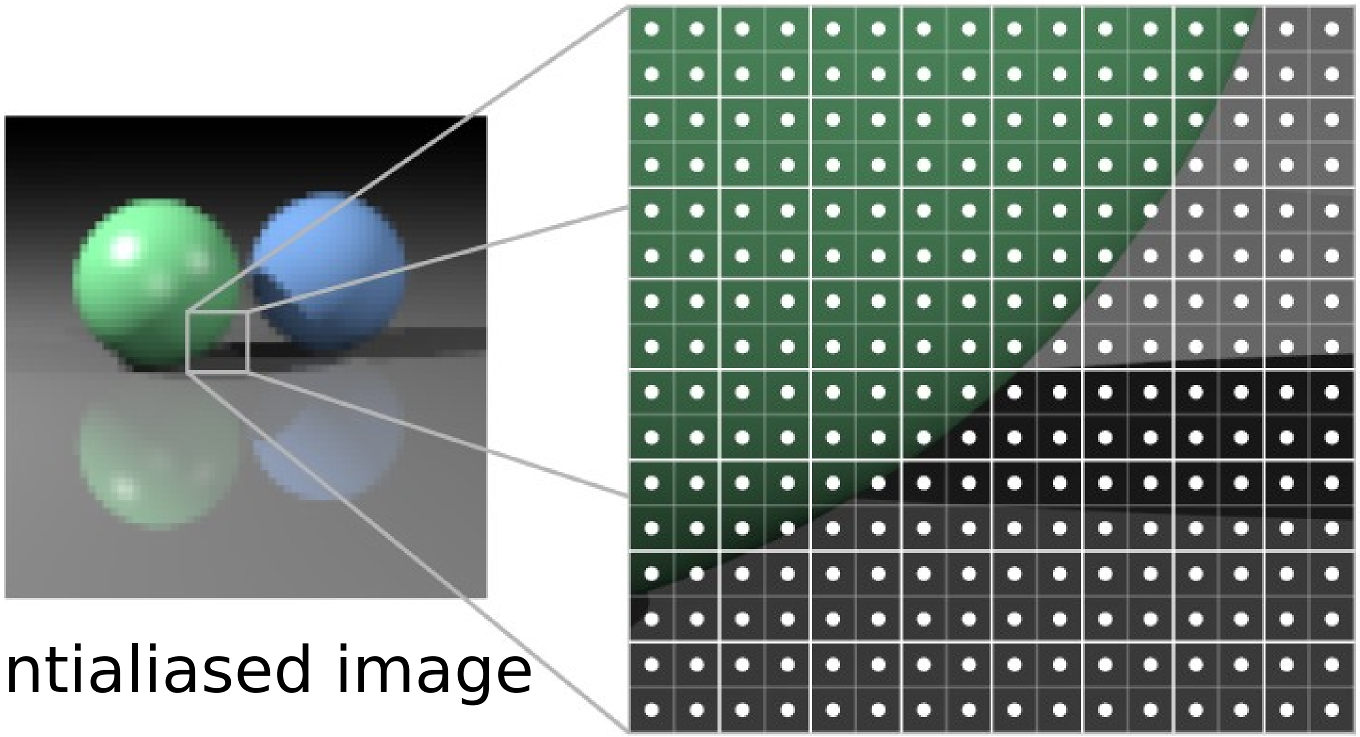


aliased image

Antialiasing in ray tracing



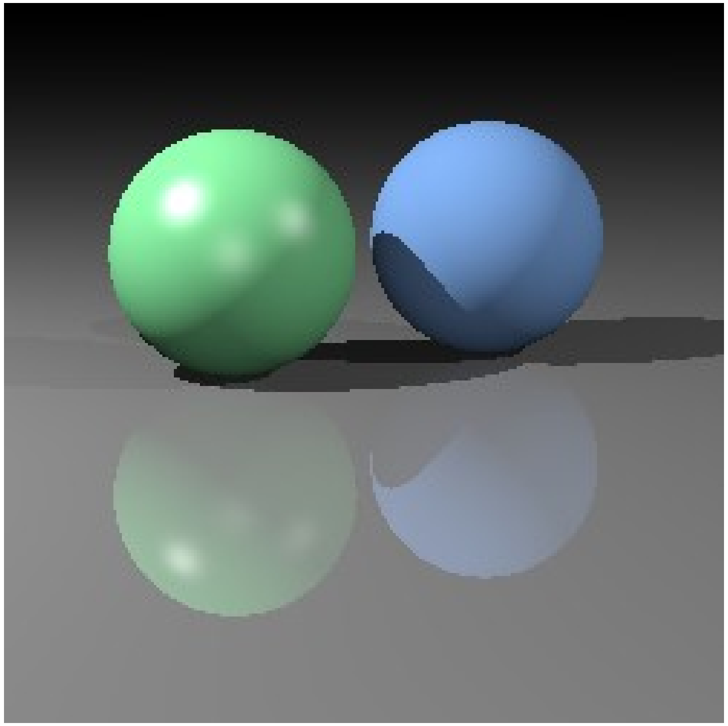
Antialiasing in ray tracing



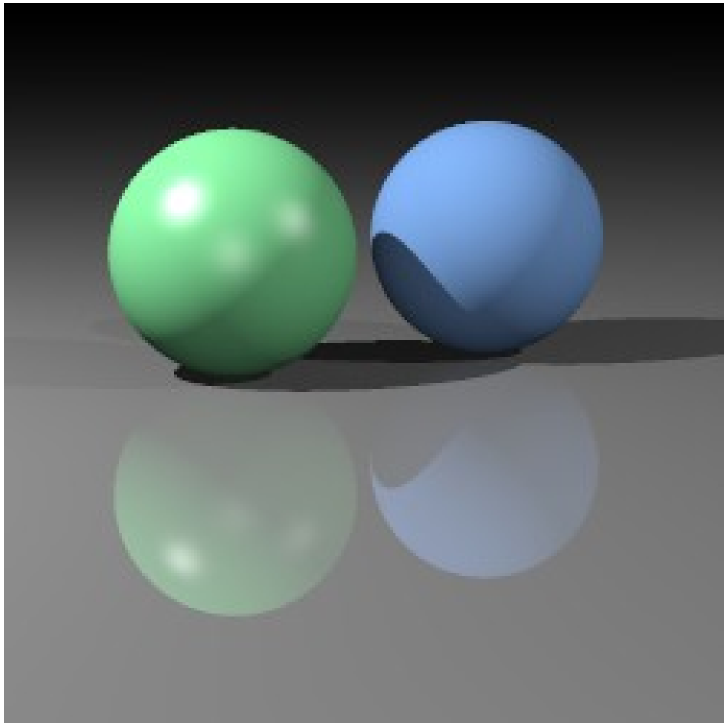
antialiased image

four samples per pixel

Antialiasing in ray tracing



one sample/pixel

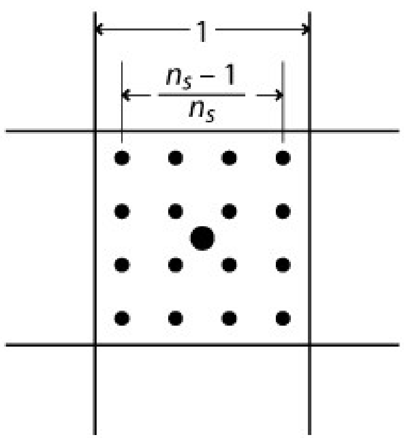


9 samples/pixel

Details of supersampling

- For image coordinates with integer pixel centers:

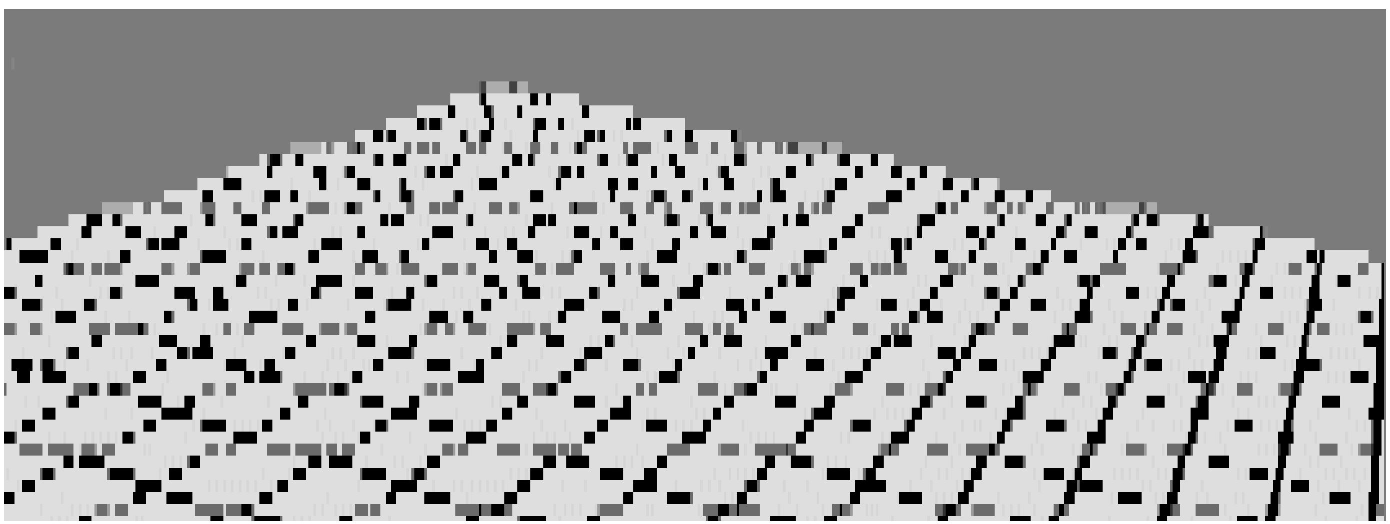
```
// one sample per pixel
for iy = 0 to (ny-1) by 1
   for ix = 0 to (nx-1) by 1 {
      ray = camera.getRay(ix, iy);
      image.set(ix, iy, trace(ray));
   }
```




```
// 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));
   }
```

Antialiasing in textures

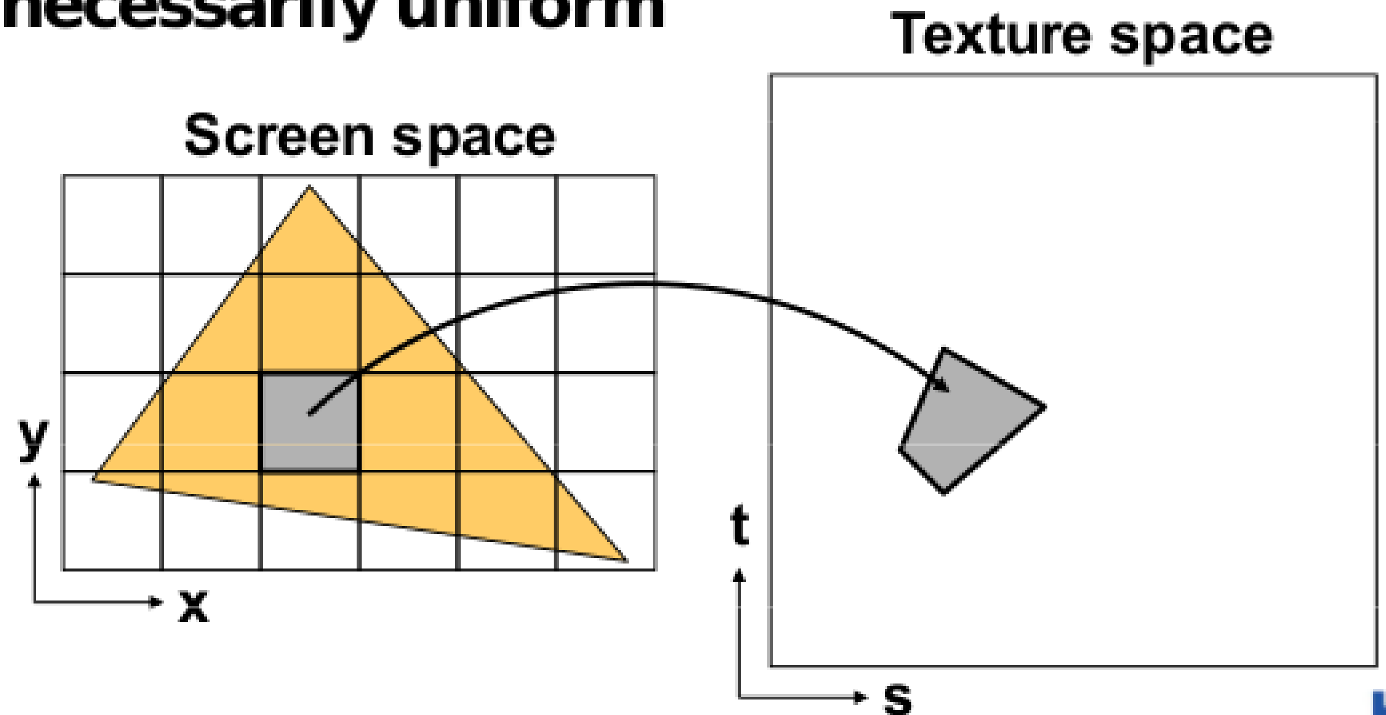
- Would like to render textures with one (or few) sampling without aliasing
- Need to filter first!
  - perspective produces very high image frequencies



[Akenine-Möller & Haines 2002]

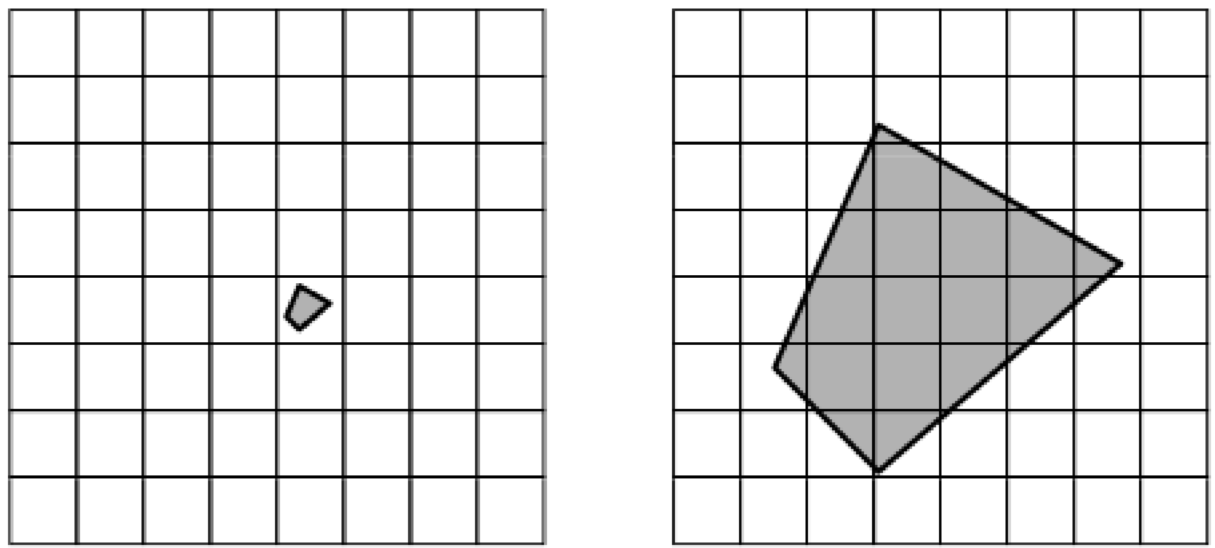
Sampling texture maps

- The uniform sampling pattern in screen space cooresponds to some sampling pattern in texture space that is not necessarily uniform



Sampling density mismatch

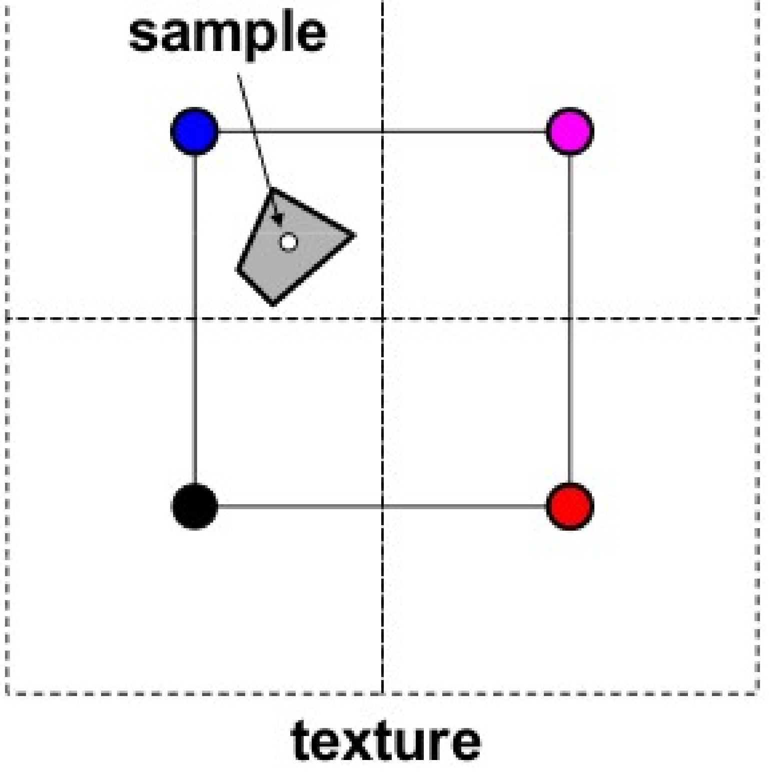
- **Sampling density in texture space rarely matches the sample density of the texture itself**



Oversampling
(Magnification)

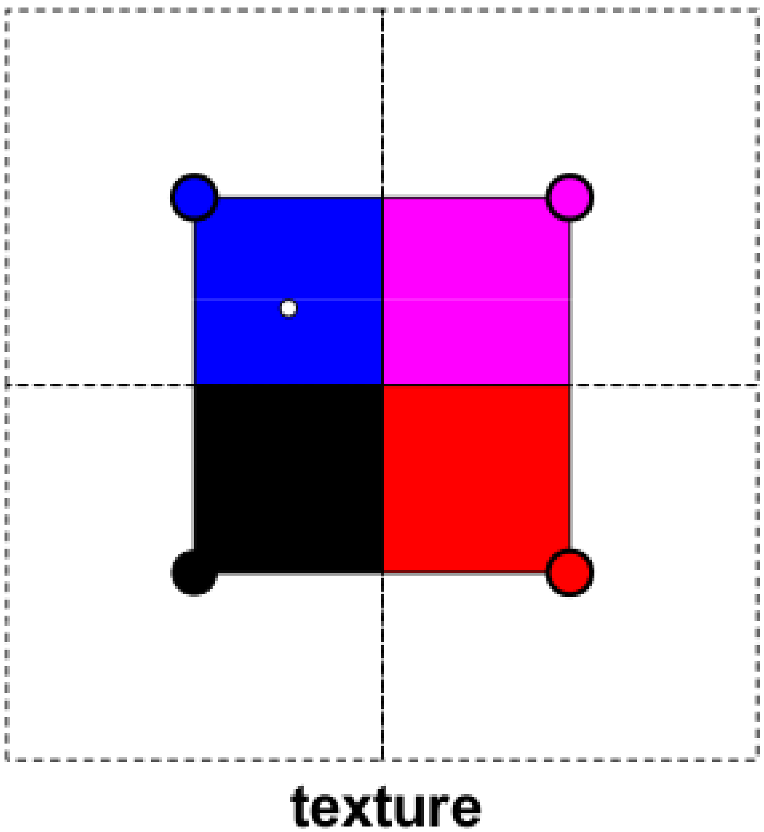
Undersampling
(Minification)

Handling oversampling (magnification)



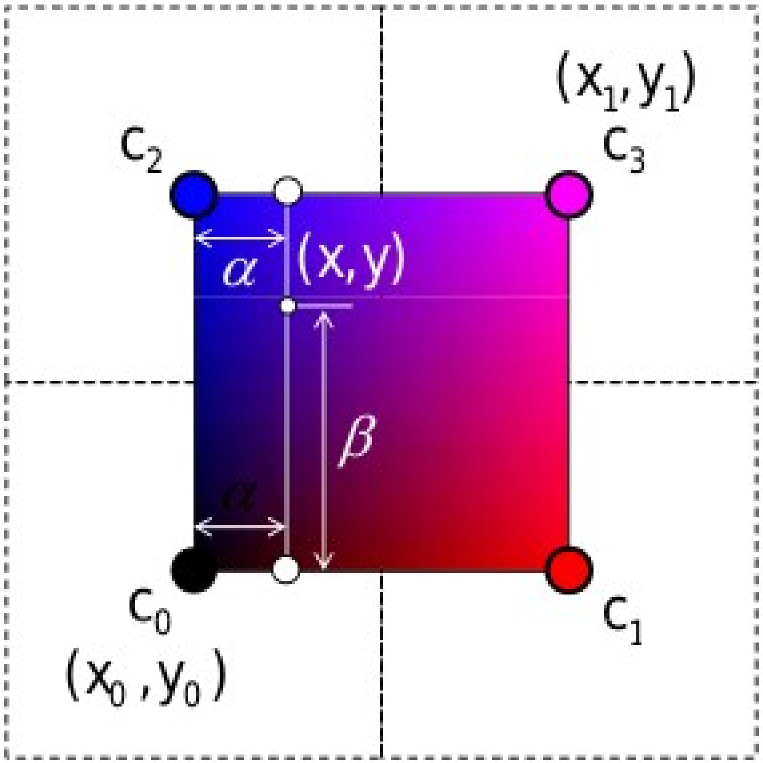
- How do we compute the color to assign to this sample?

Handling oversampling (magnification)



- How do we compute the color to assign to this sample?
- Nearest neighbor – take the color of the closest texel

Handling oversampling (magnification)



texture

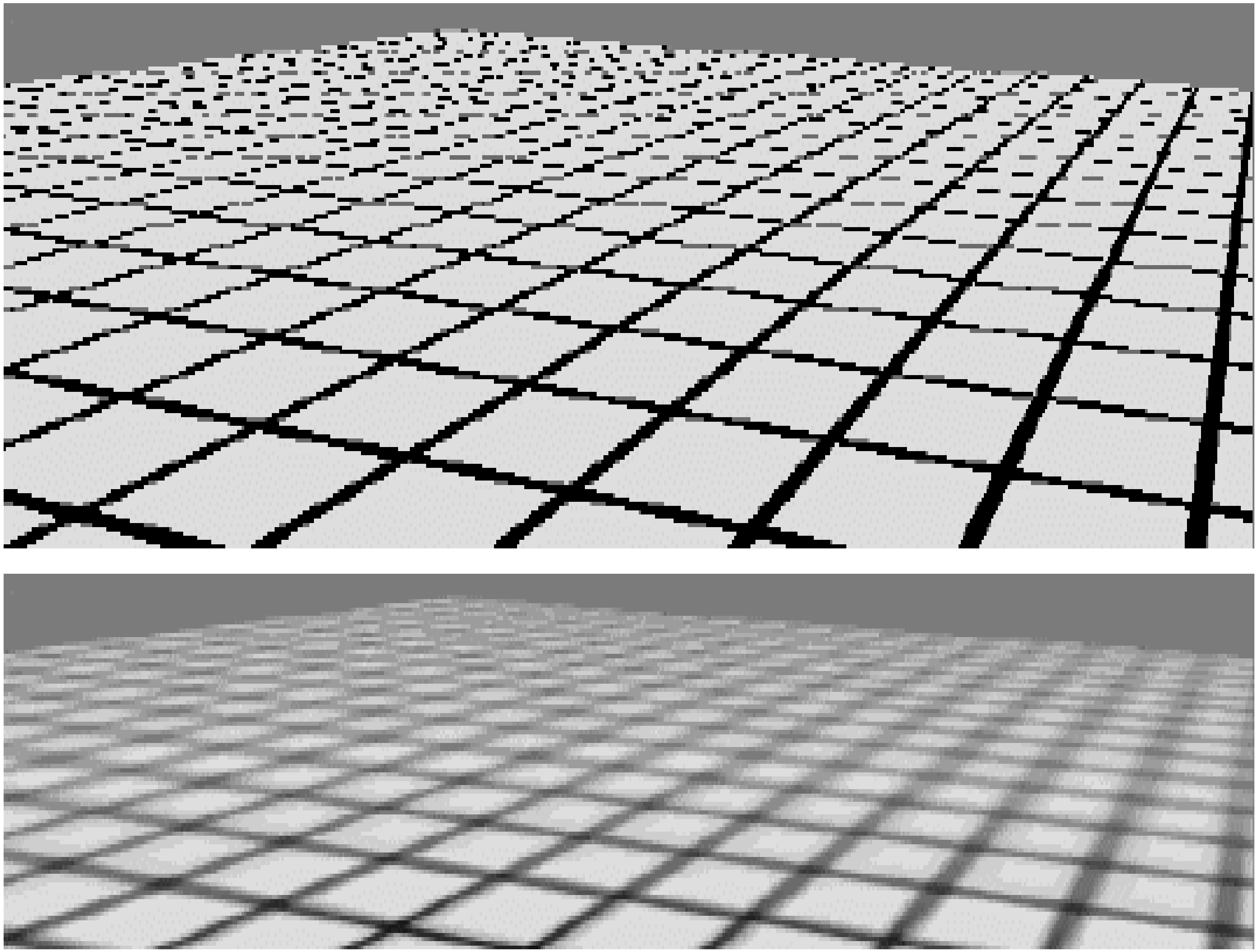
- **How do we compute the color to assign to this sample?**
- **Nearest neighbor – take the color of the closest texel**
- **Bilinear interpolation**

$$\alpha = \frac{x - x_0}{x_1 - x_0} \qquad \beta = \frac{y - y_0}{y_1 - y_0}$$

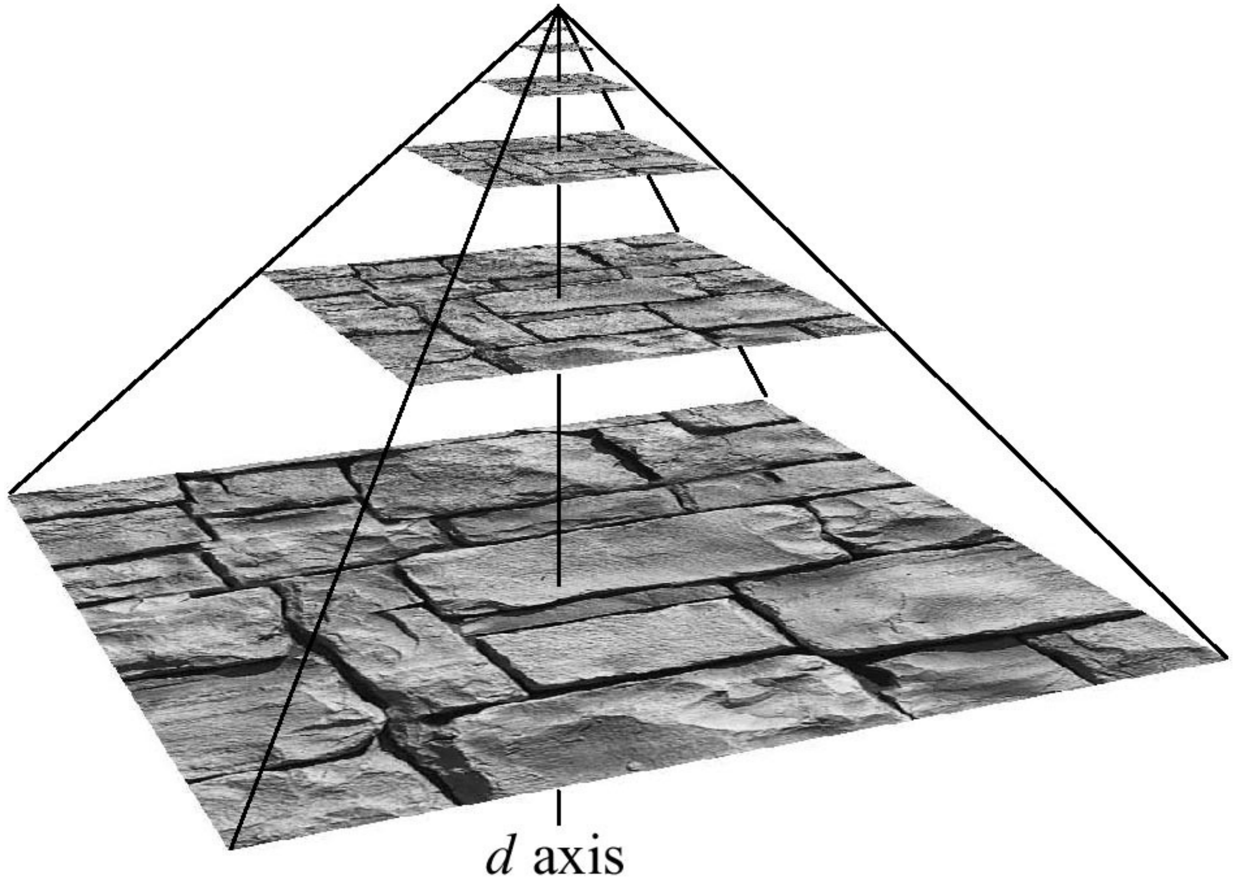
$$c = ((1-\alpha)c_0 + \alpha c_1)(1-\beta) + ((1-\alpha)c_2 + \alpha c_3)\beta$$

KAIST

Texture minification



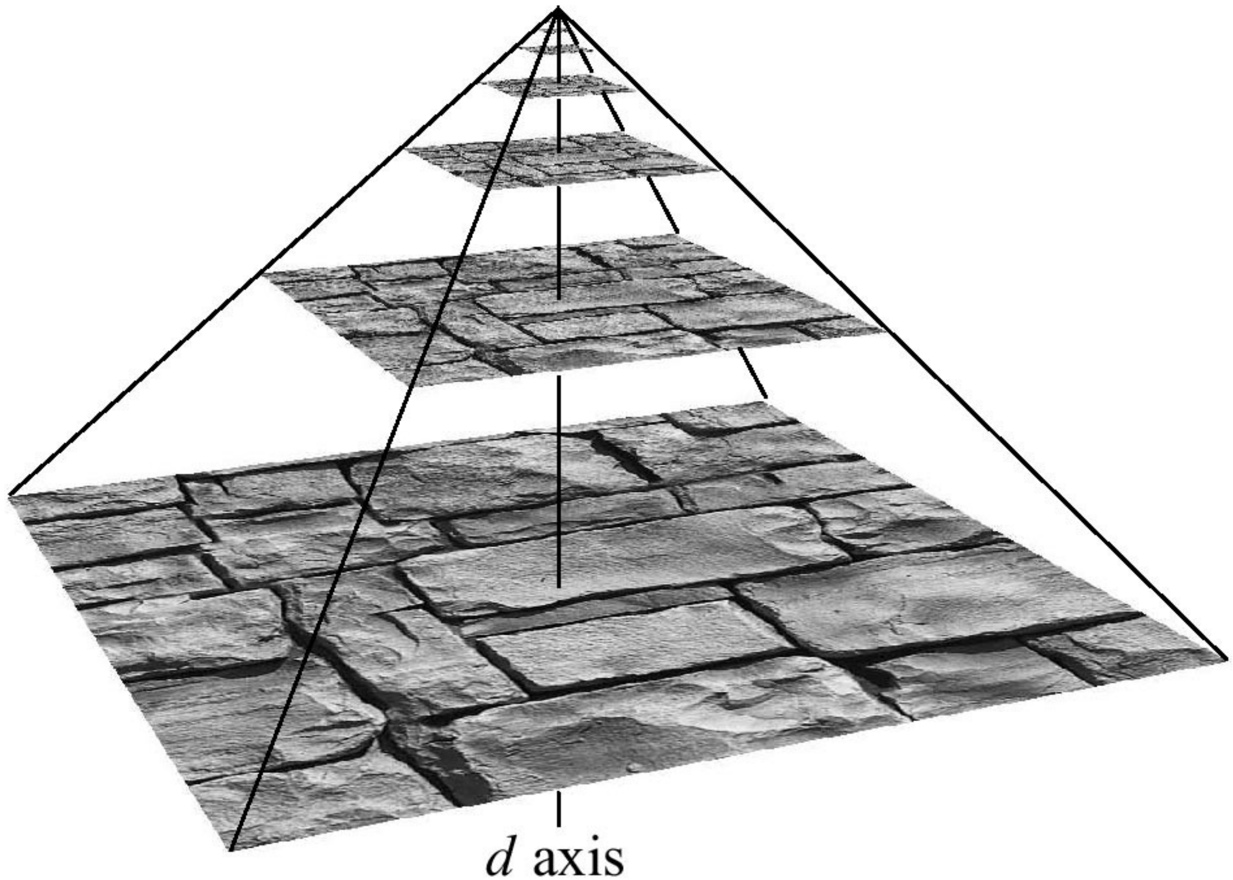
Mipmap image pyramid



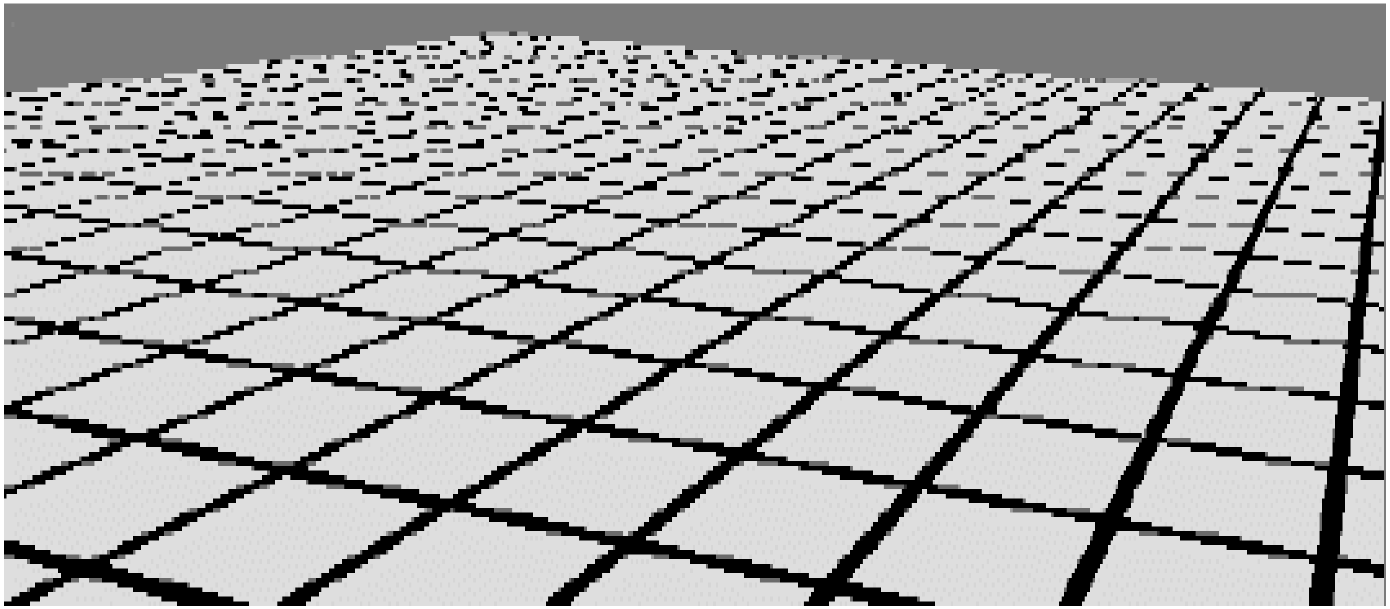
[Akenine-Möller & Haines 2002]

Finding MIP level

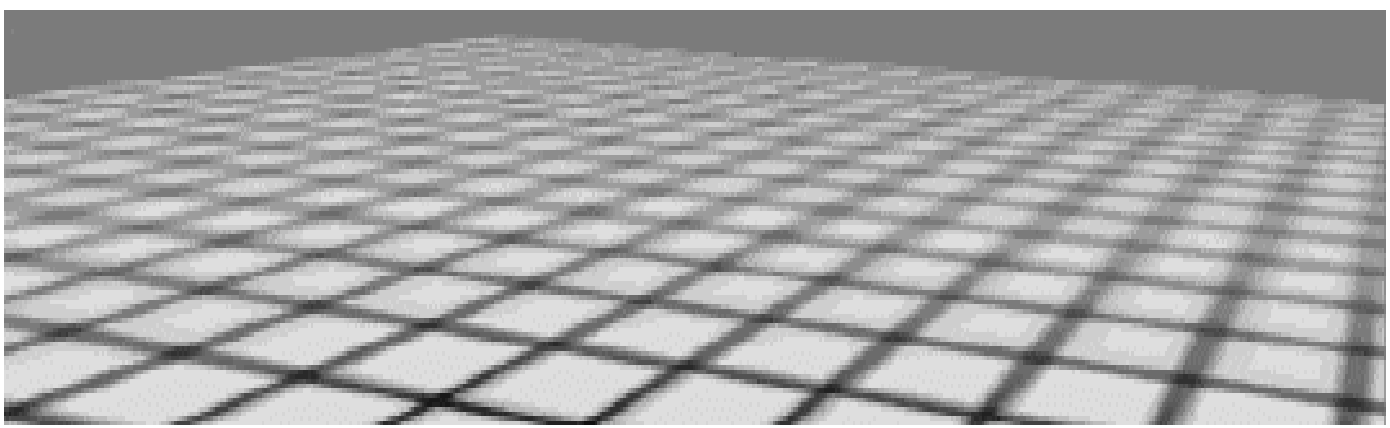
- Use the projection of a pixel in screen into texture space to figure out which level to use



Texture minification



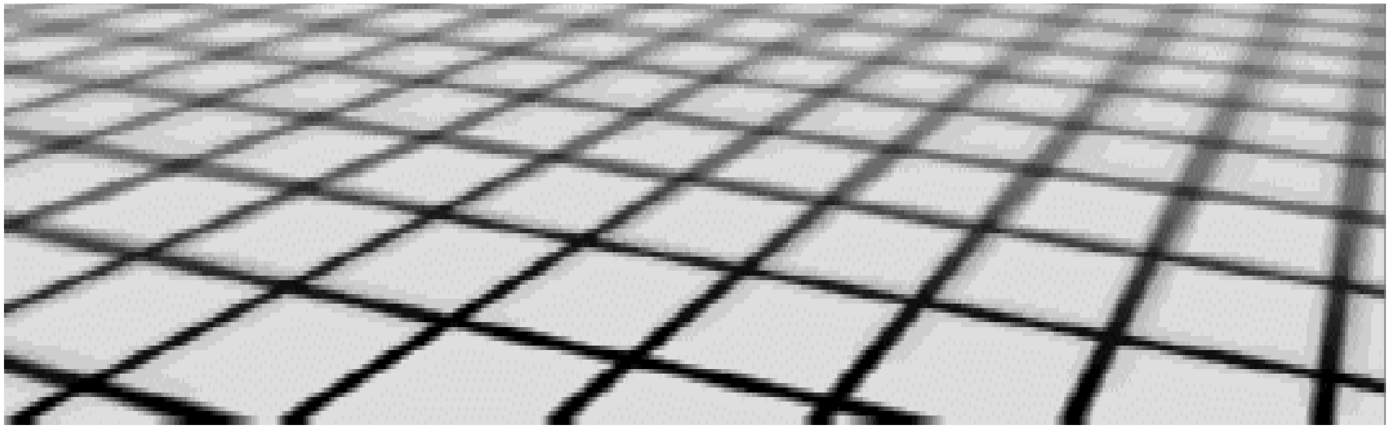
point
sampled
minificatio
n



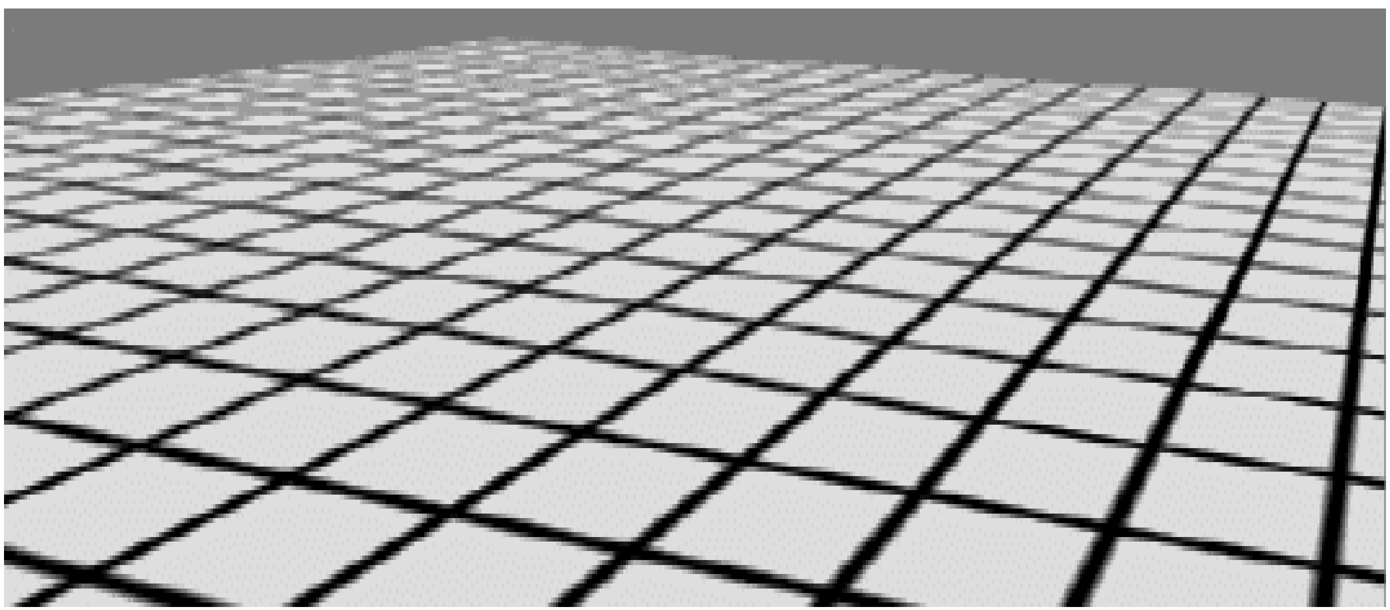
mipmap
minificatio
n

[Akenine-Möller & Haines 2002]

Texture minification



mipmap minificatio
n

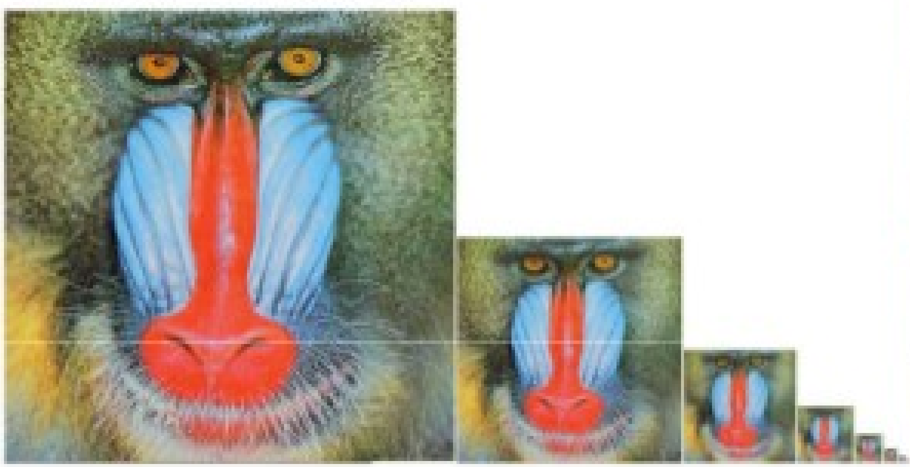


higher quality minificatio
n

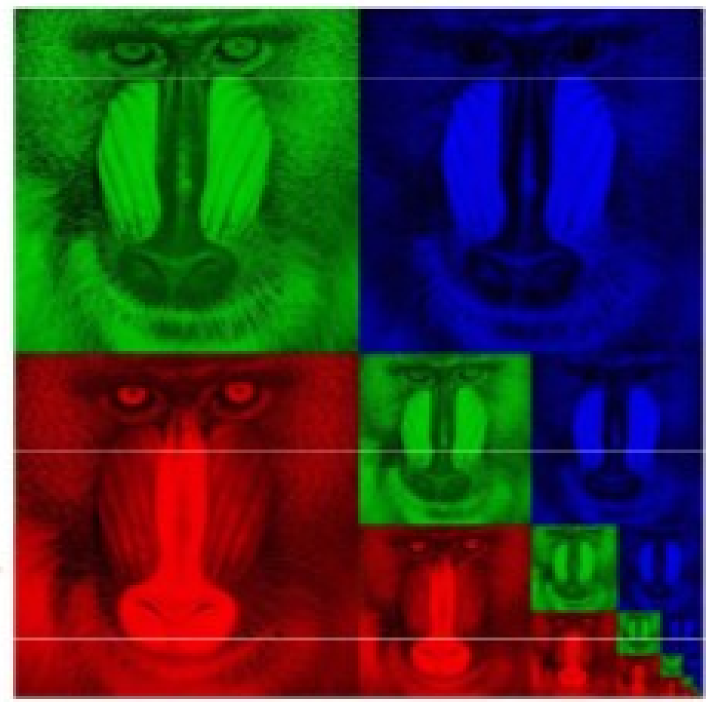
[Akenine-Möller & Haines 2002]

Storing MIP Maps

- 1/3 overhead of maintaining the MIP maps



10-level mip map



Memory format of a mip map

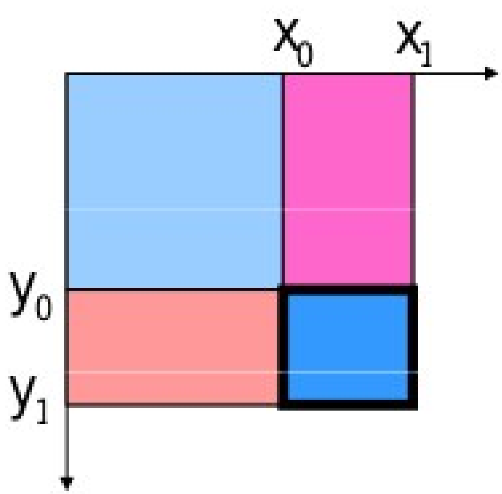
Summed-Area Tables

- Another way of performing the prefiltering integration on the fly
- Each entry in the summed area table is the sum of all entries above and to the left:

1	6	8	3
0	0	3	7
4	7	8	8
5	0	9	9

→

1	7	15	18
1	7	18	28
5	18	37	55
10	23	51	78



What is the sum of the highlighted region?

$$T(x_1,y_1) - T(x_1,y_0) - T(x_0,y_1) + T(x_0,y_0)$$

Divide out area $(y_1 - y_0)(x_1 - x_0)$

KAIST

Summed-Area Tables

- How much storage does a summed-area table require?
- Does it require more or less work per pixel than a MIP map?