CS4620/5620: Lecture 25

Texture Mapping

Announcements

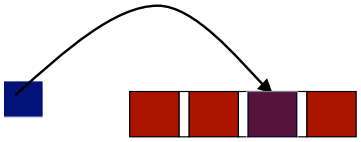
- PA2B out
- Plan for next few lectures
  - Texturing, Anti-aliasing, sampling theory, Perlin noise
  - Monday: Splines

When viewed from a distance

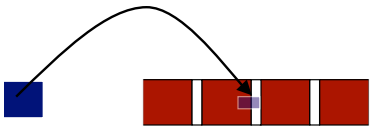
- Aliasing!
- Also, minification

How does area map over distance?

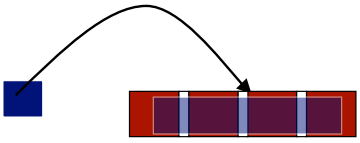
- At optimal viewing distance:
  - One-to-one mapping between pixel area and texel area



- When closer
  - Each pixel is a small part of the texel

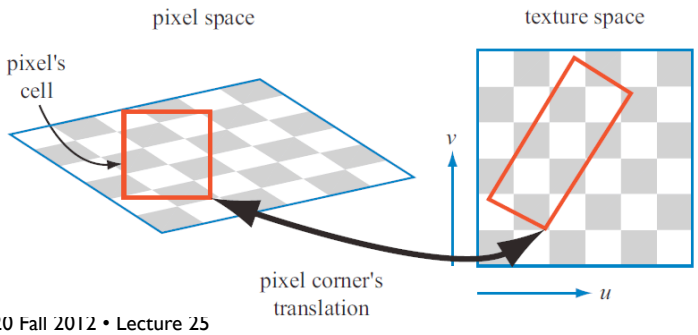


- When farther
  - Each pixel could include many texels



Theoretical Solution

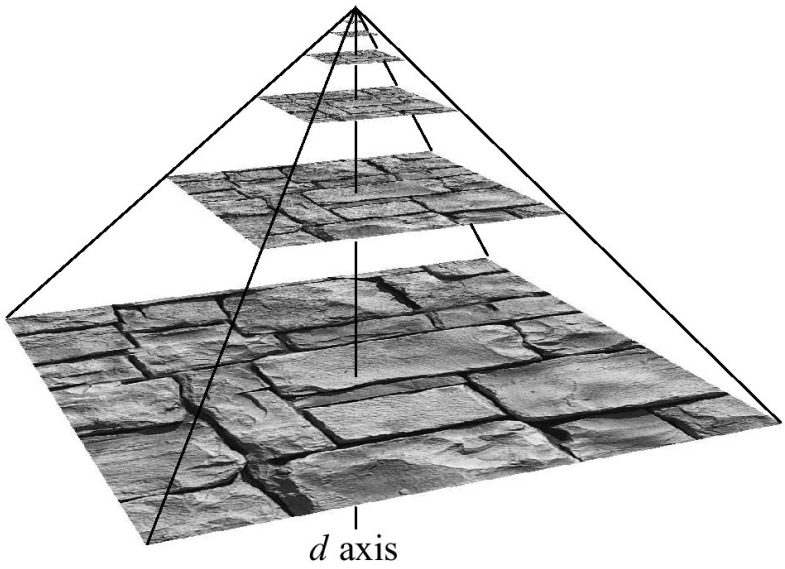
- Find the area of pixel in texture space
- "Filter" the area to compute "average" texture color
  - Filtering eliminates high frequency artifacts
  - How to filter?
    - Analytically compute area
    - But too expensive



MIP Maps

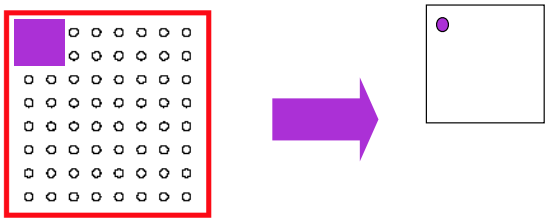
- MIP Maps
  - Multum in Parvo: Much in little, many in small places
  - Proposed by Lance Williams
- Stores pre-filtered versions of texture
- Supports very fast lookup

Mipmap image pyramid



[Akenine-Möller & Haines 2002]

Filtering by Averaging



- Each pixel in a level corresponds to 4 pixels in lower level
  - Average
  - Gaussian filtering (more on this next lecture)

Using the MIP Map

- Find the MIP Map level where the pixel has a 1-to-1 mapping

- How?
  - Find largest side of pixel footprint in texture space
    - Pick level where that side corresponds to a texel
  - Compute derivatives to find pixel footprint
    - Intuition for derivatives

Using the MIP Map

- Find the MIP Map level where the pixel has a 1-to-1 mapping

- How?
  - Find largest side of pixel footprint in texture space
    - Pick level where that side corresponds to a texel
  - Compute derivatives to find pixel footprint
    - x derivative: $\frac{\partial u}{\partial x}$ $\frac{\partial v}{\partial x}$
    - y derivative: $\frac{\partial u}{\partial y}$ $\frac{\partial v}{\partial y}$

Given derivatives: what is level?

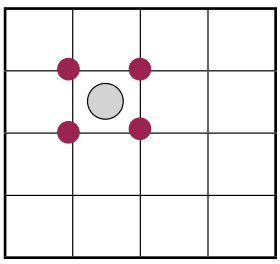
$$level = log[max(\frac{du}{dx}, \frac{dv}{dx}, \frac{du}{dy}, \frac{dv}{dy})]$$

$$level = log\sqrt{(\frac{du}{dx})^2 + (\frac{dv}{dx})^2 + (\frac{du}{dy})^2 + (\frac{dv}{dy})^2}$$

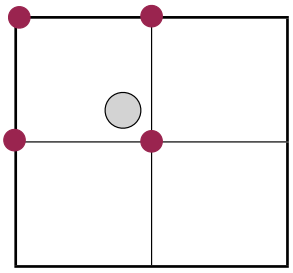
- Gradients
  - Available in pixel shader (except where there is dynamic branching)

Using the MIP Map

- In level, find texel and
  - Return the texture value: point sampling (but still better)!
  - Bilinear interpolation
  - Trilinear interpolation



Level i



Level i+1

Interpolation

- Bilinear interpolation for (u, v)
  - (u_0, v_0), (u_1, v_1), (u_2, v_2), (u_3, v_3)
  - T_0, T_1, T_2, T_3

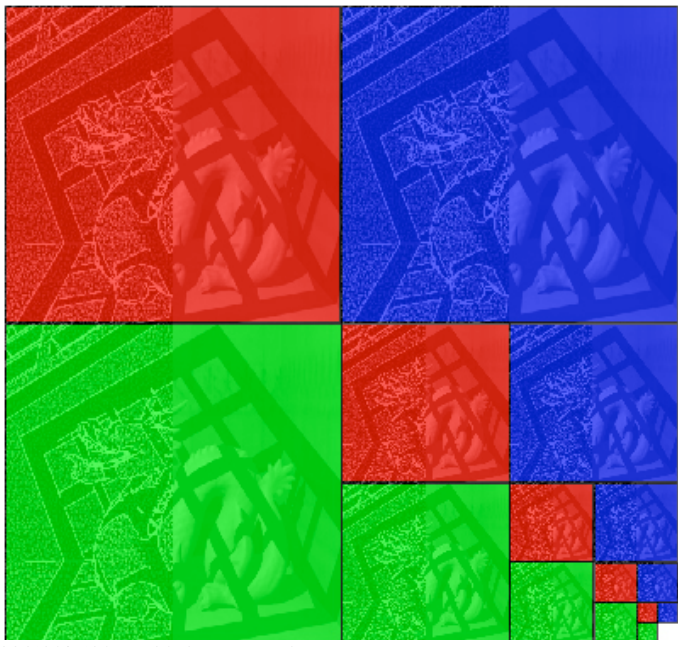
$$B(u,v) = (u_1-u)[(v-v_0)\,T_3 + (v_1-v)\,T_0] + (u-u_0)[(v-v_0)\,T_2 + (v_1-v)\,T_1]$$

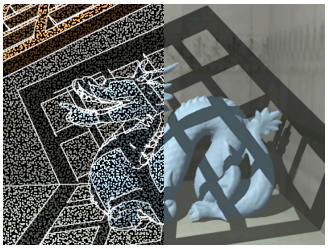
- Trilinear interpolation

  $(d1-d)\,B(u, v, d0) + (d-d0)\,B(u, v, d1)$

Memory Usage

- What happens to size of texture?

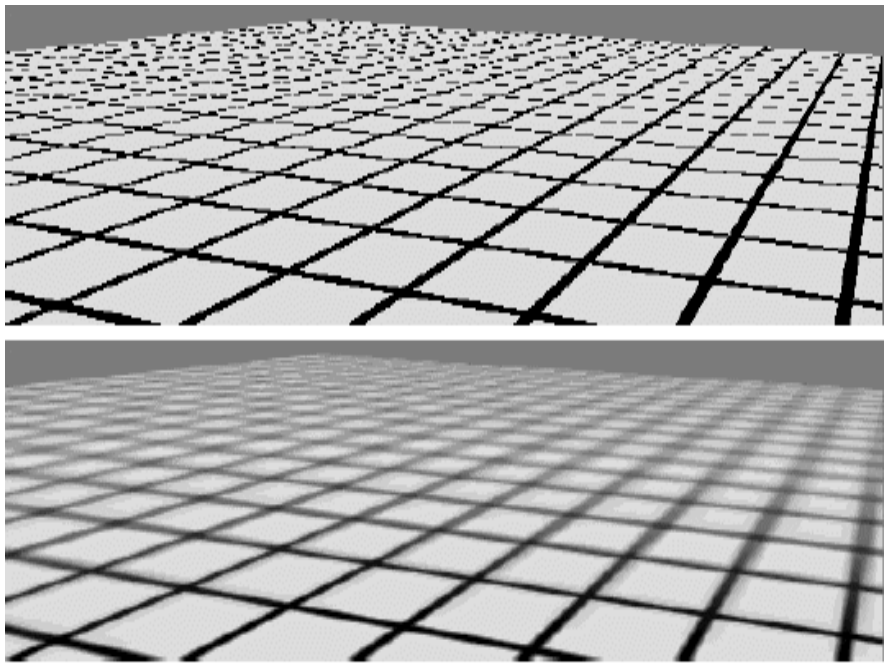




MIPMAP

- Multi-resolution image pyramid
  - Pre-sampled computation of MIPMAP
  - 1/3 more memory
- Bilinear or Trilinear interpolation

Texture minification



point sampled minification

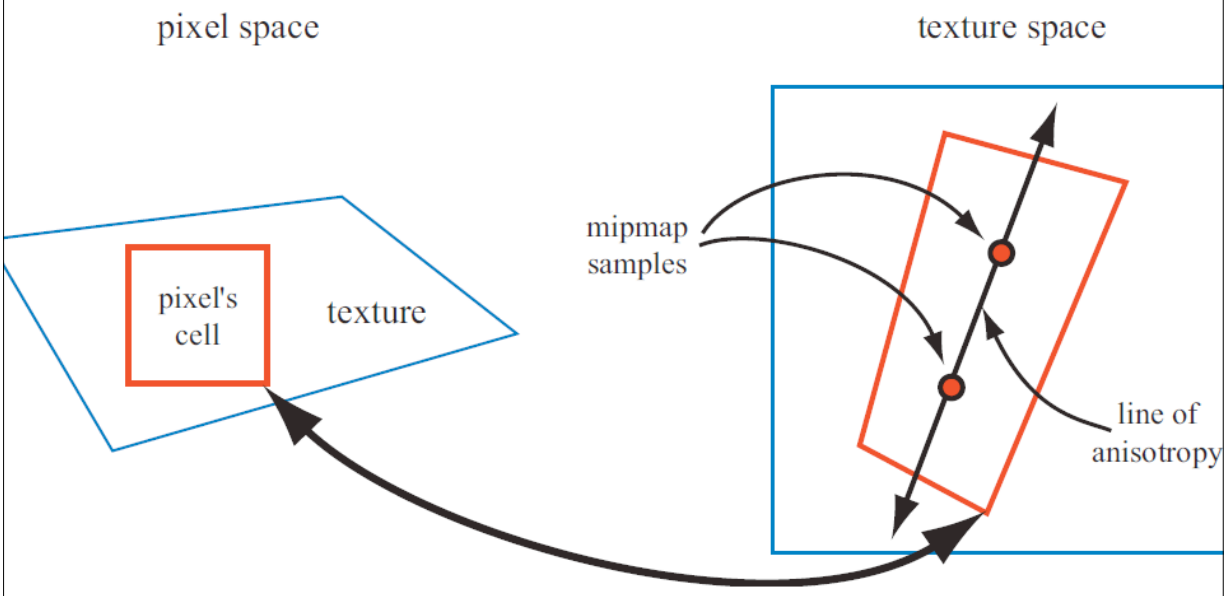
mipmap minification

[Akenine-Möller & Haines 2002]

Some basic assumptions

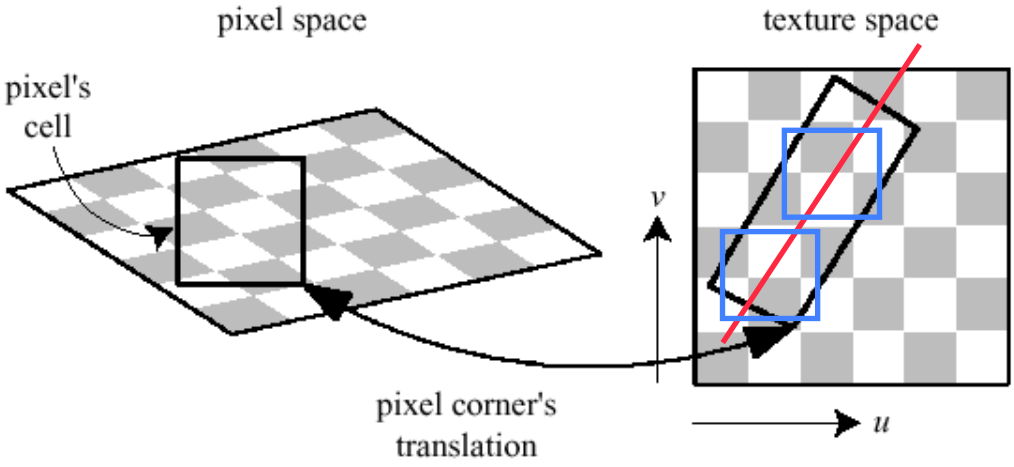
- Can't really precompute every possible required area
- Assume that pixel only maps to squares in texture space
  - In fact, assume it maps to squares at particular locations

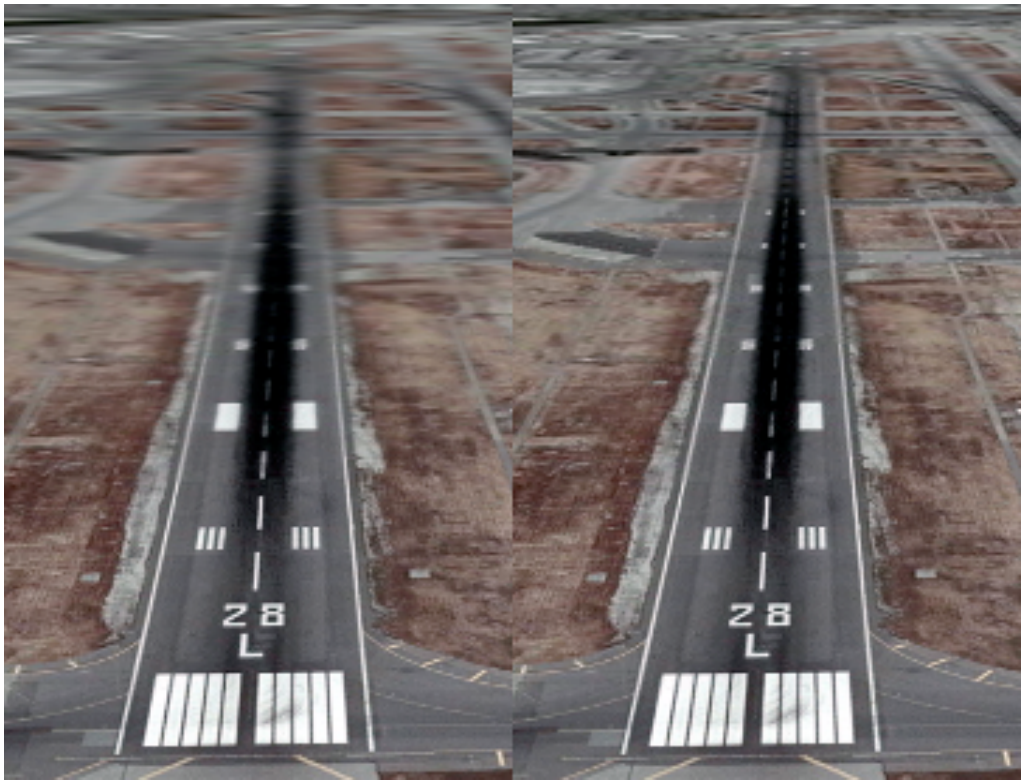
Anisotropic Filtering



Anisotropic Filtering

- GPU supports multiple reads: 16x



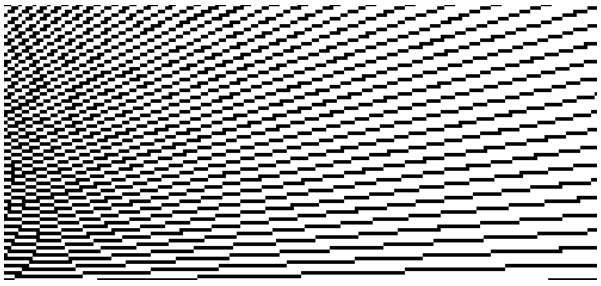


Sampling and Antialiasing

Aliasing

point sampling a continuous image:

continuous image defined by a bunch of black rectangles

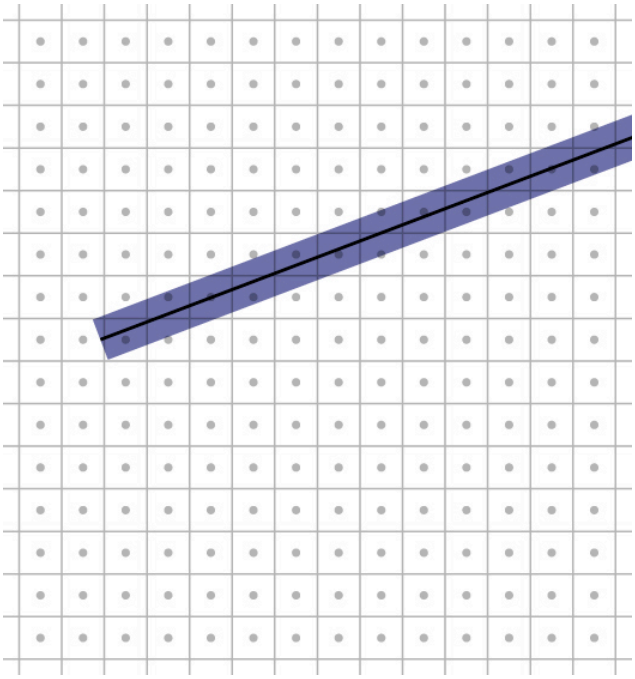


Antialiasing

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

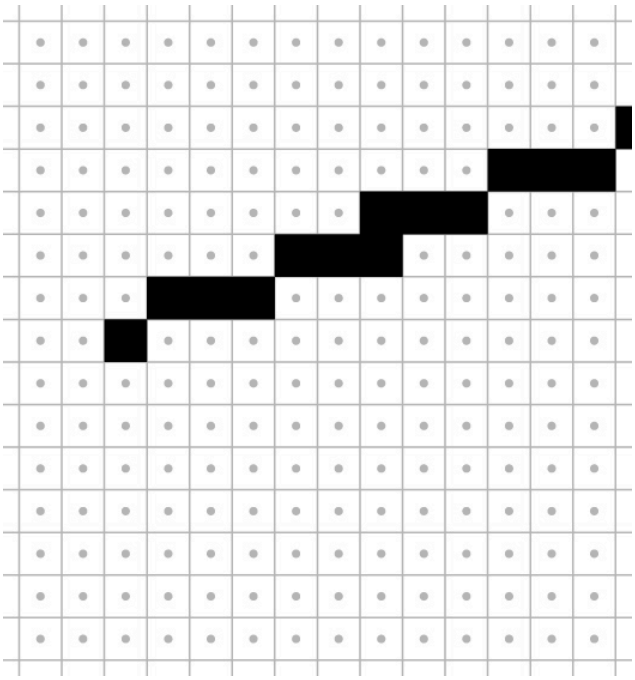
Rasterizing lines

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

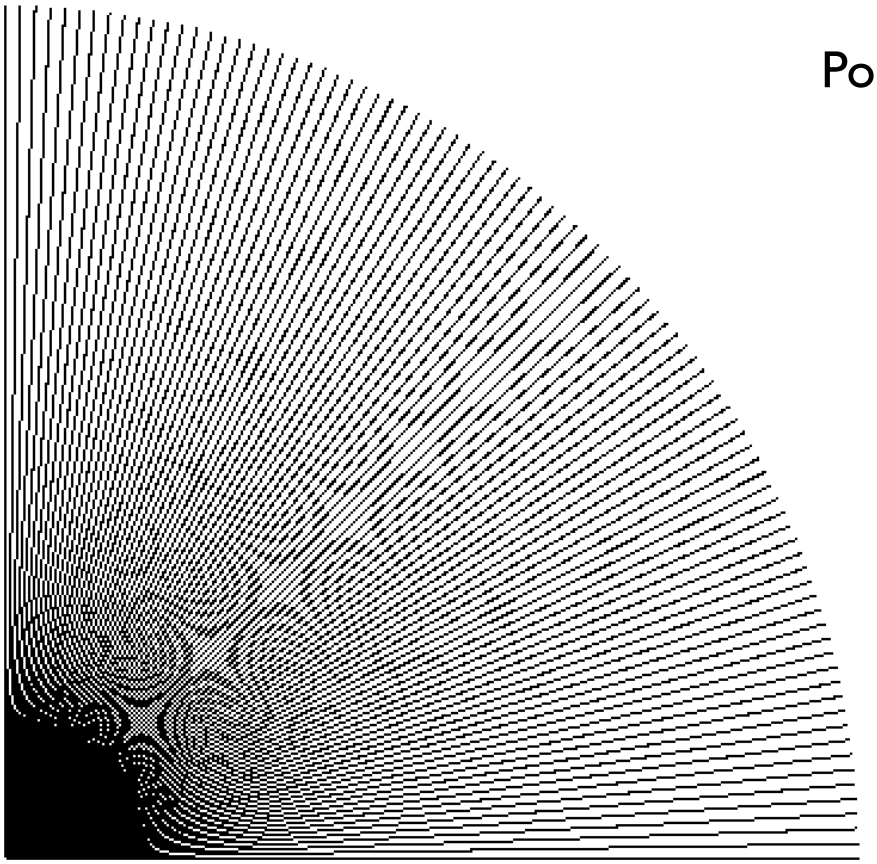


Point sampling

- 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)

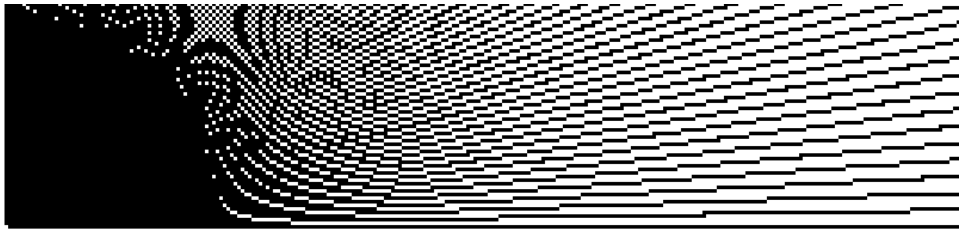


Point sampling in action



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!

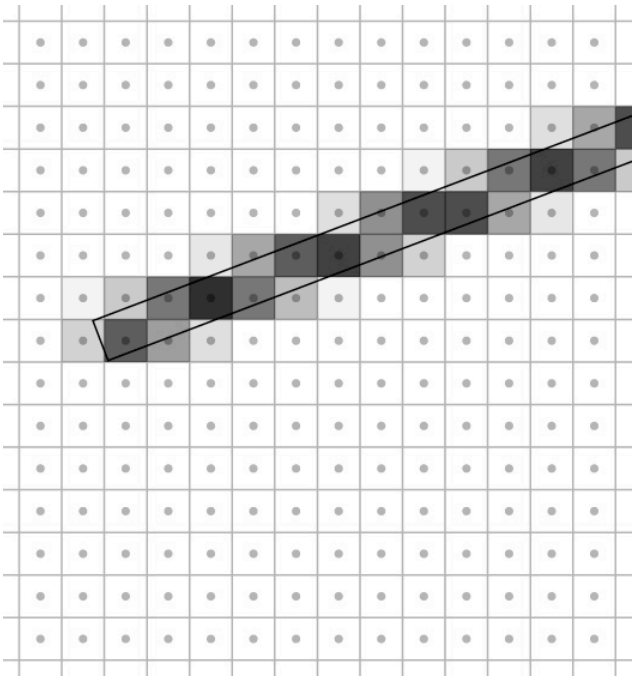


Antialiasing

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

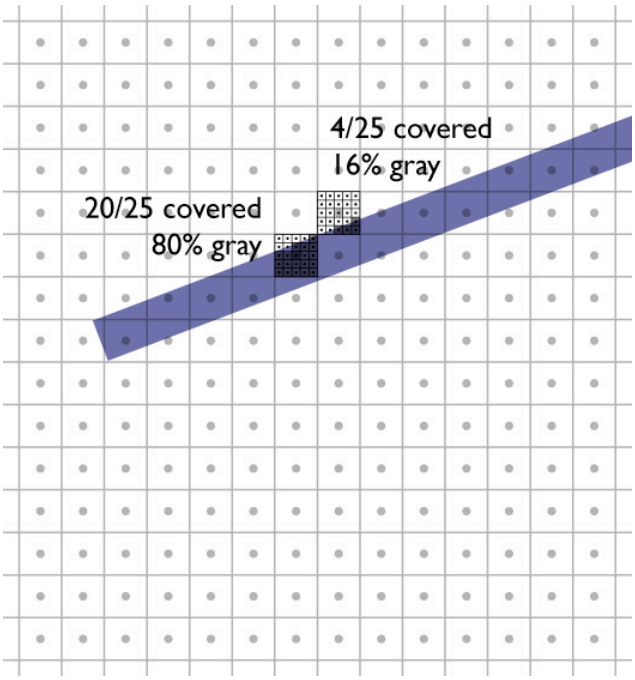


Box filtering

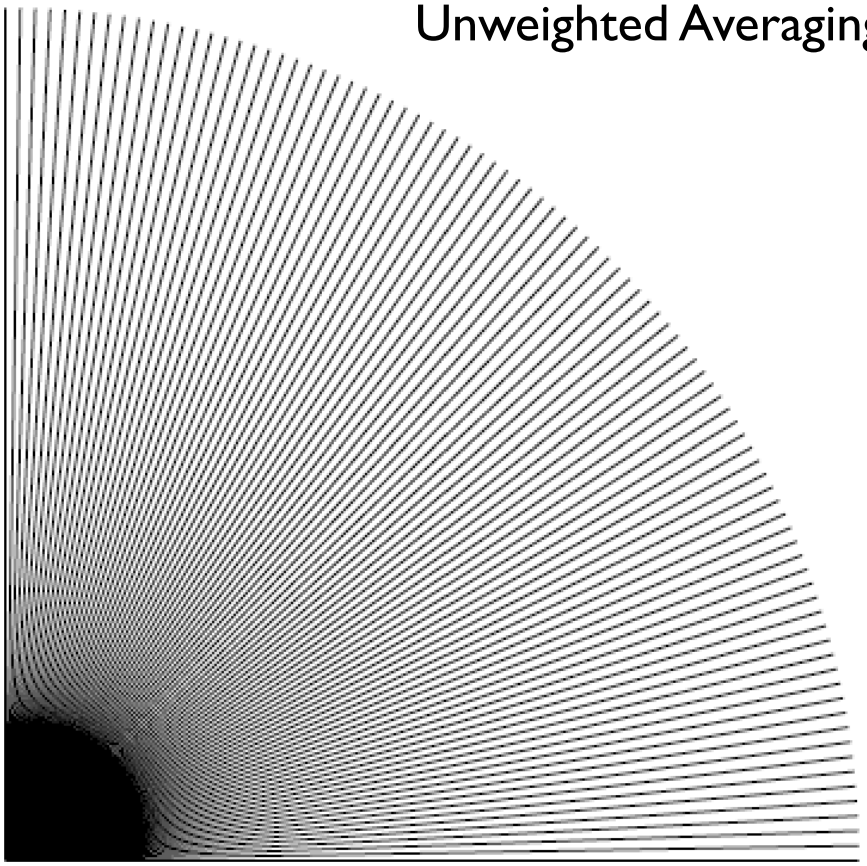
- Pixel intensity is proportional to area of overlap with square pixel area
- Also called "unweighted area averaging"

Box filtering by supersampling

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



Unweighted Averaging/Box filtering in action

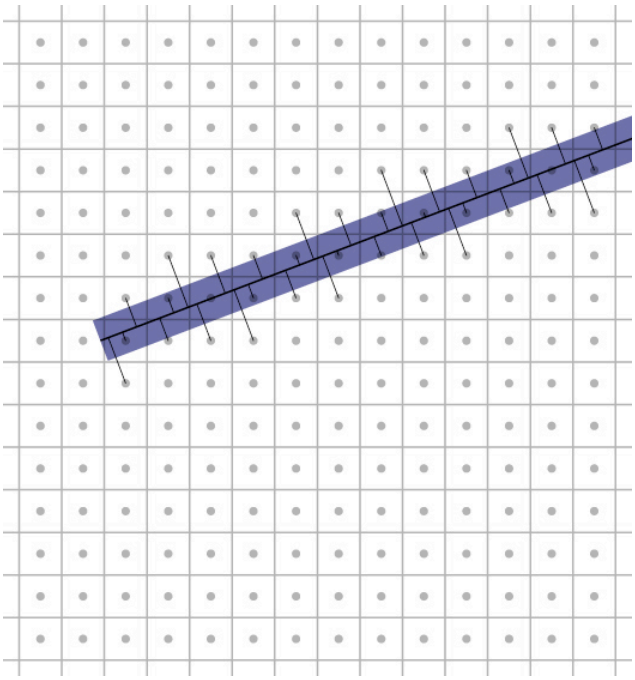


Antialiasing and resampling

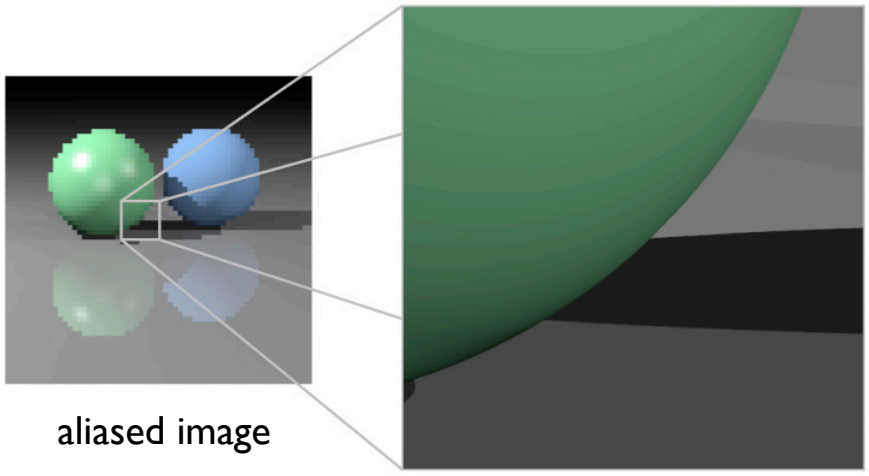
- Antialiasing by regular supersampling is *the same as* rendering a larger image and then resampling it to a smaller size
- Convolution of filter with high-res image produces an estimate of the area of the primitive in the pixel.
- So we can re-think this
  - one way: we're computing area of pixel covered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

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
- Does not handle ends…

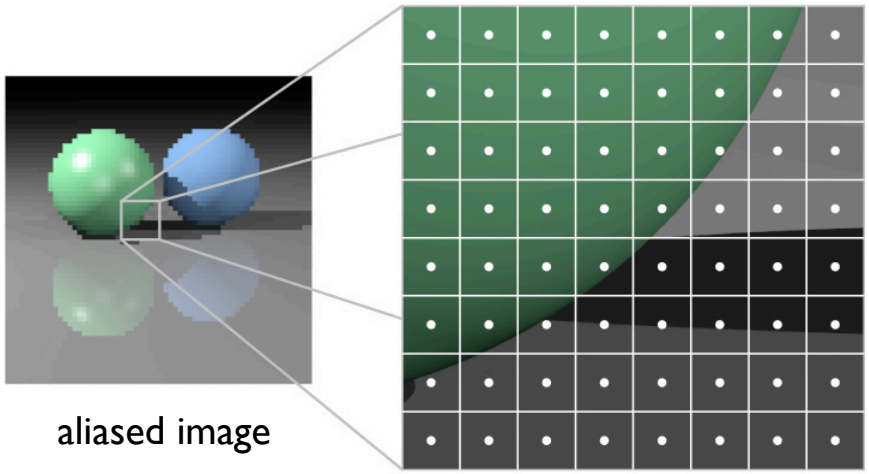


Antialiasing in ray tracing



aliased image

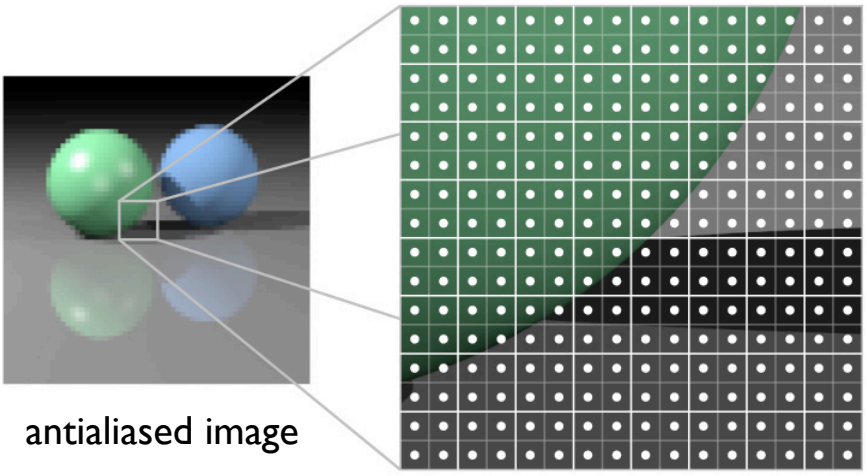
Antialiasing in ray tracing



aliased image

one sample per pixel

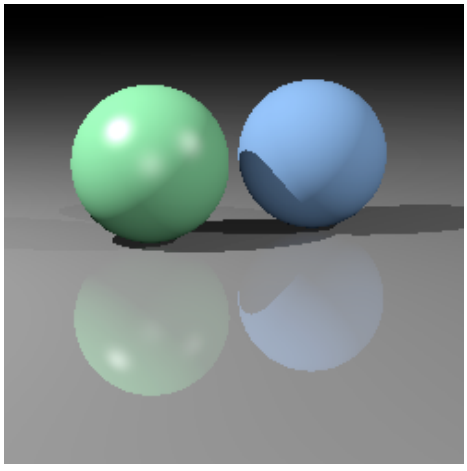
Antialiasing in ray tracing



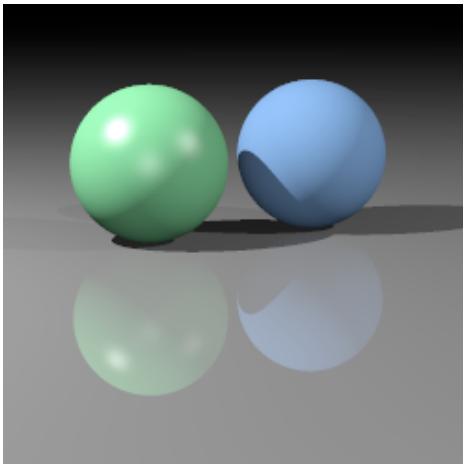
antialiased image

four samples per pixel

Antialiasing in ray tracing



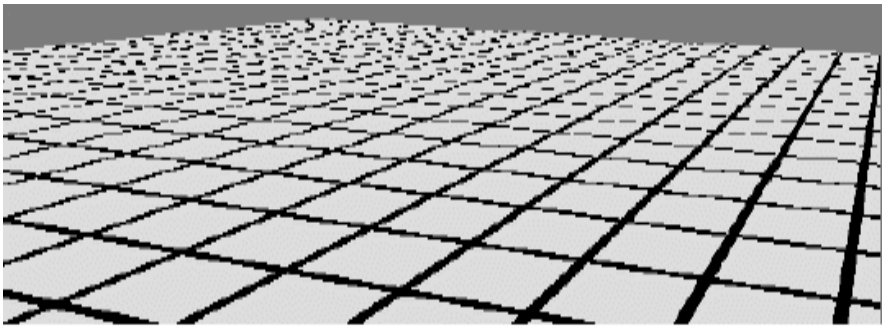
one sample/pixel



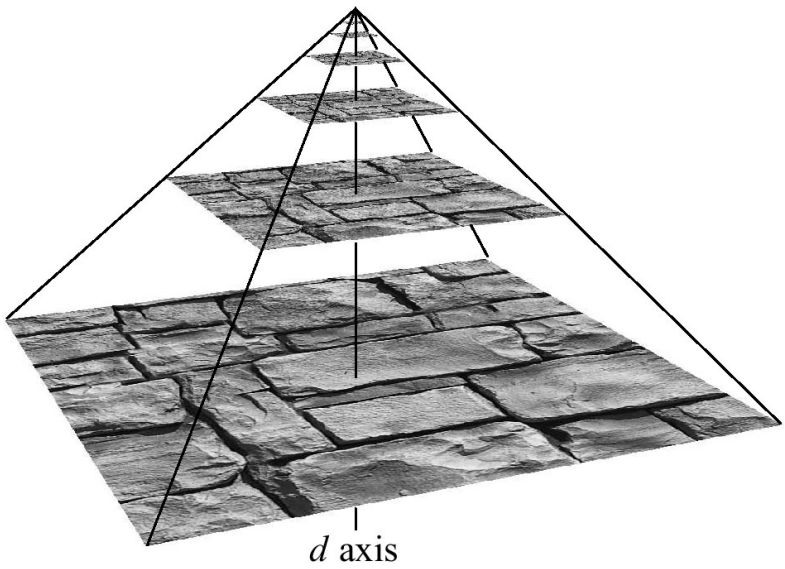
9 samples/pixel

Antialiasing in textures

- Would like to render textures with one (or few) s/p
- Need to filter first!
  - perspective produces very high image frequencies

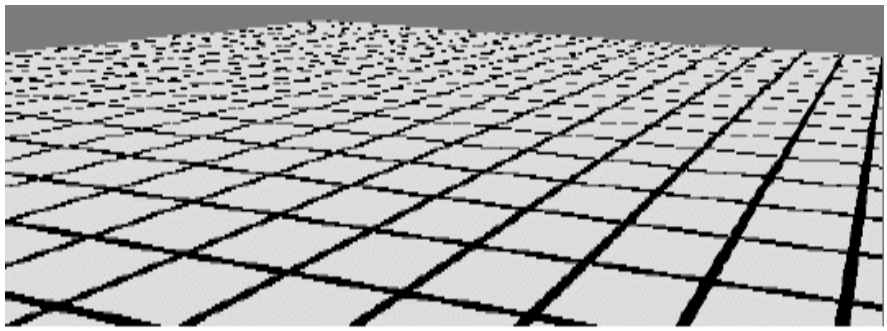


Mipmap image pyramid

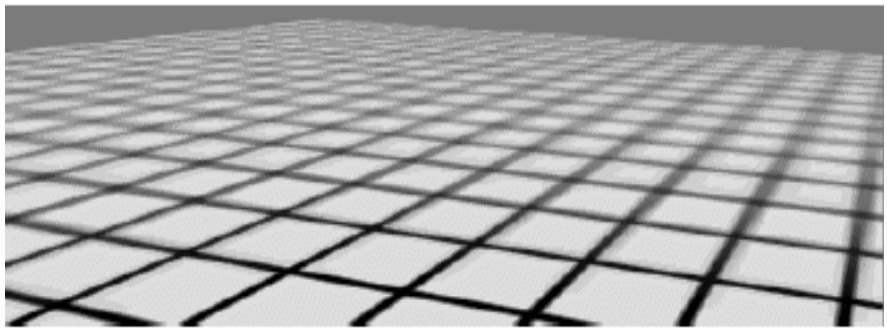


[Akenine-Möller & Haines 2002]

Texture minification



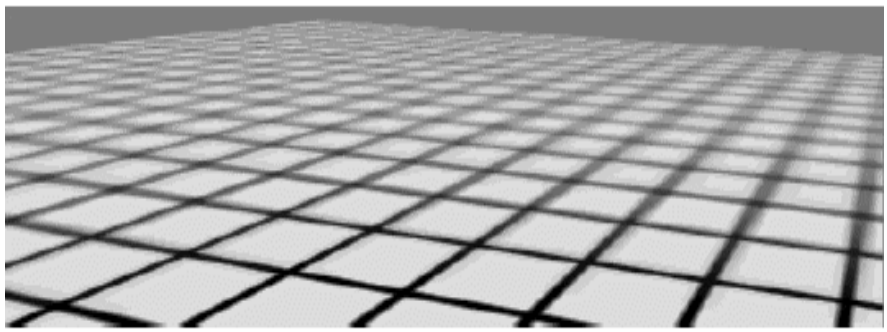
point sampled minification



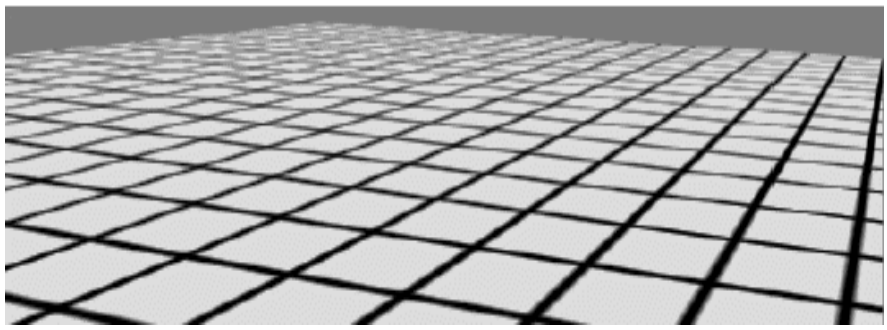
mipmap minification

[Akenine-Möller & Haines 2002]

Texture minification



mipmap minification



higher quality minification

[Akenine-Möller & Haines 2002]