

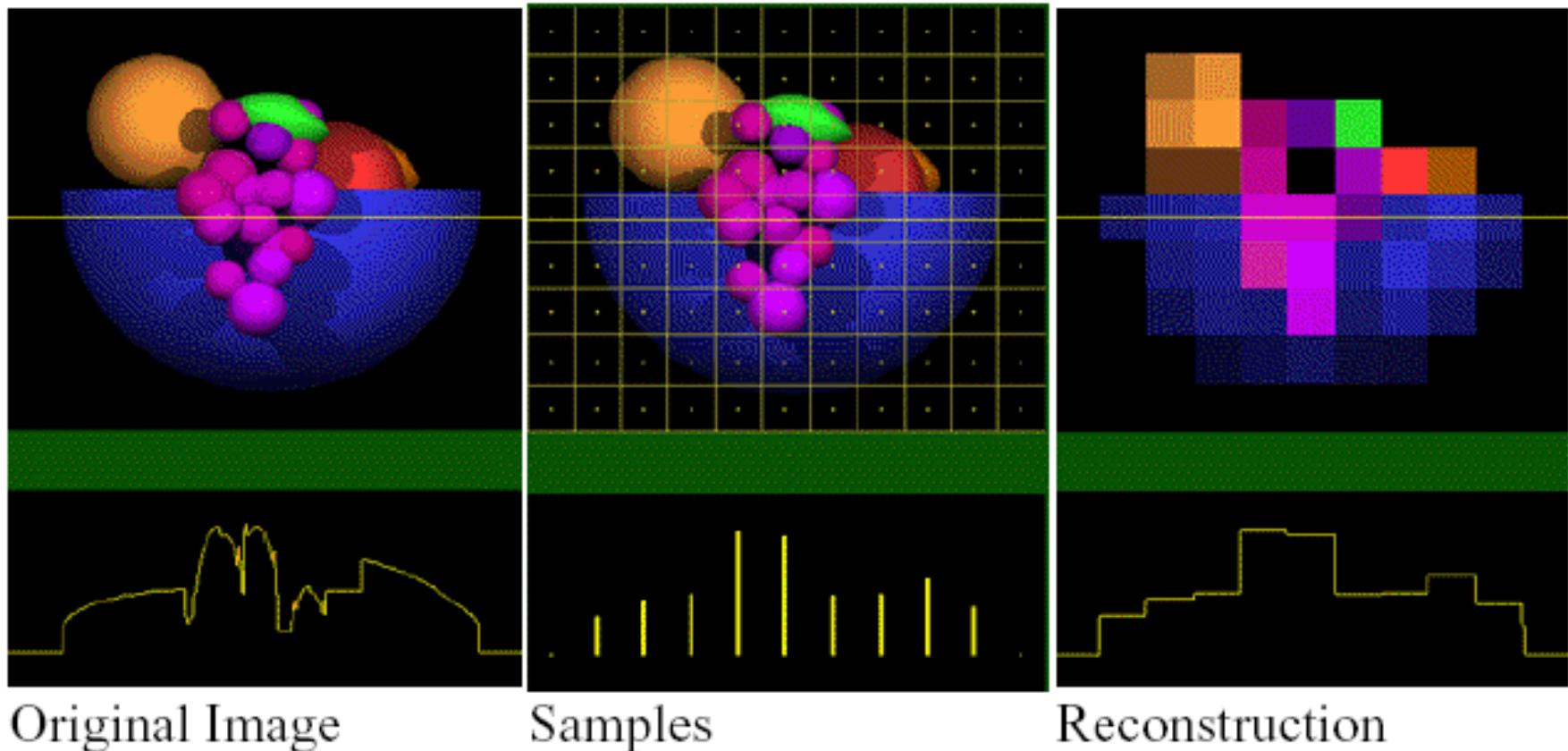
# Sampling, Aliasing, & Mipmaps

50.017 Graphics and Visualization  
Sai-Kit Yeung, SUTD ISTD

Notes courtesy by Wojciech Matusik 1

# Examples of Aliasing

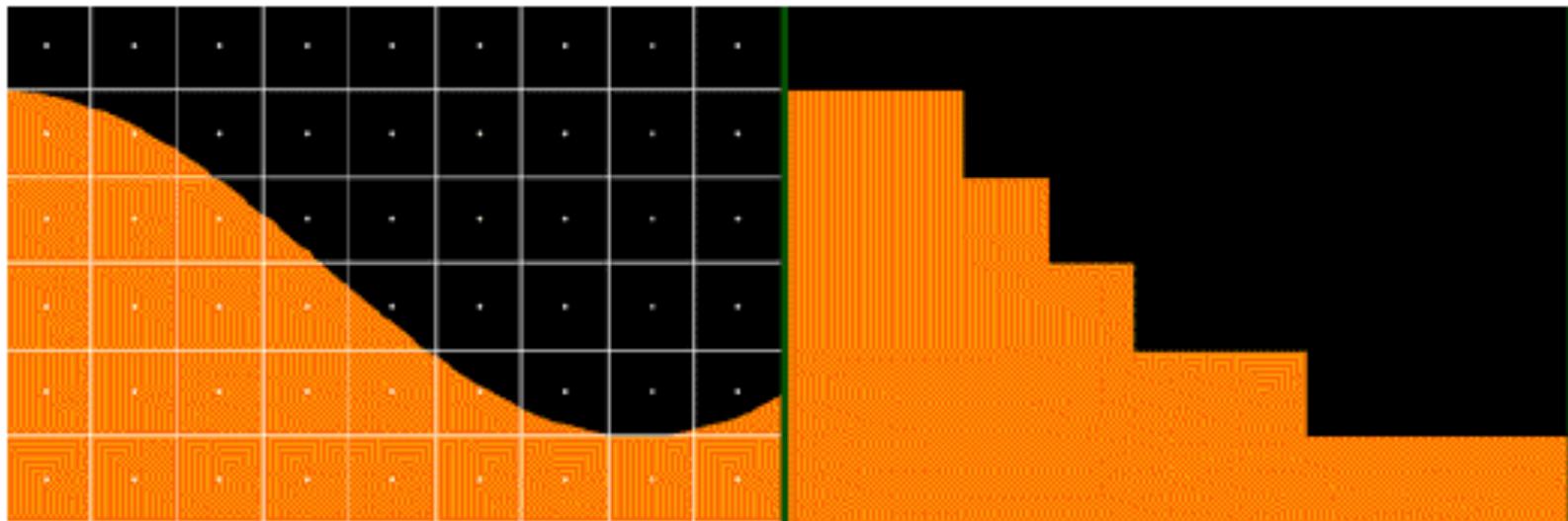
---



# Examples of Aliasing

---

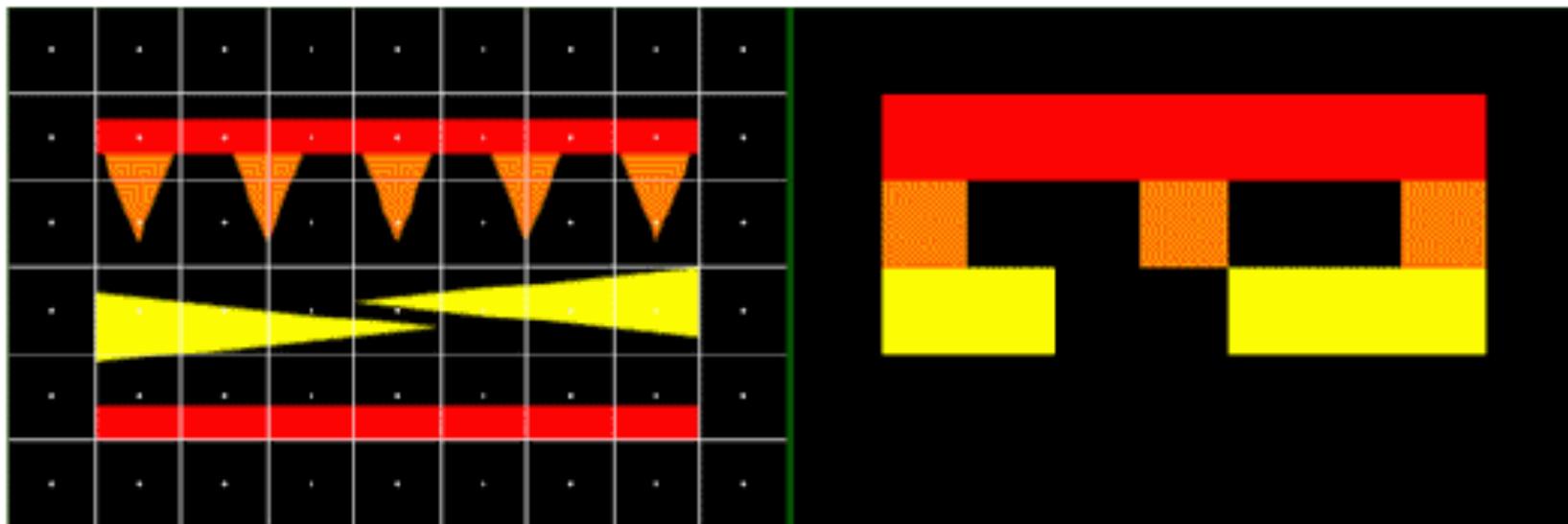
Jagged boundaries



# Examples of Aliasing

---

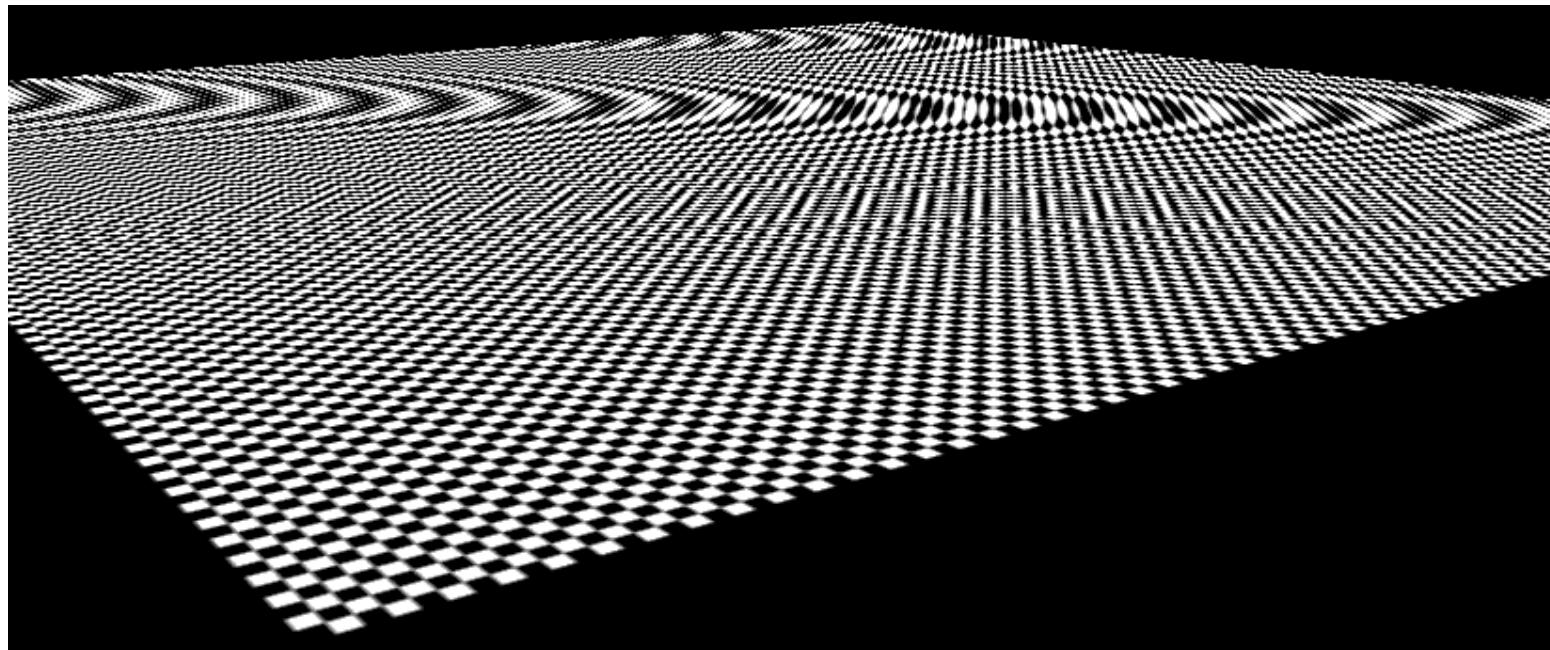
Improperly rendered detail



# Examples of Aliasing

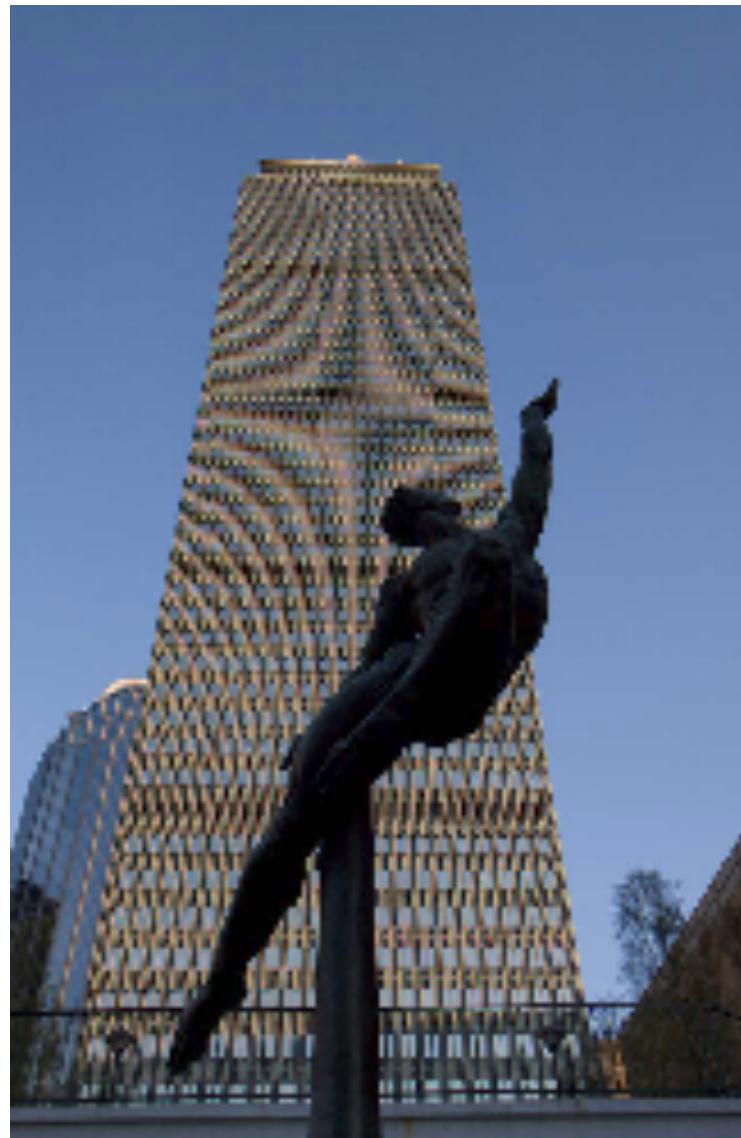
---

## Texture Errors



# In photos too

---



See also <http://vimeo.com/26299355>

# Philosophical perspective

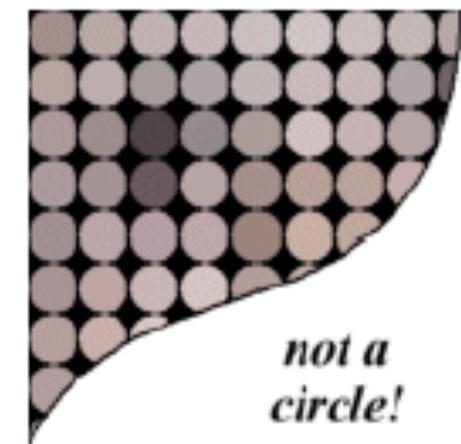
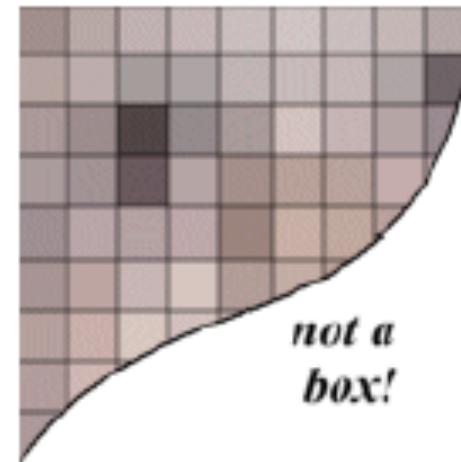
---

- The physical world is continuous, inside the computer things need to be discrete
- Audio – 44k per second
- Lots of computer graphics is about translating continuous problems into discrete solutions
  - e.g. ODEs for physically-based animation, global illumination, meshes to represent smooth surfaces, rasterization, antialiasing
- Careful mathematical understanding helps do the right thing

# What is a Pixel?

---

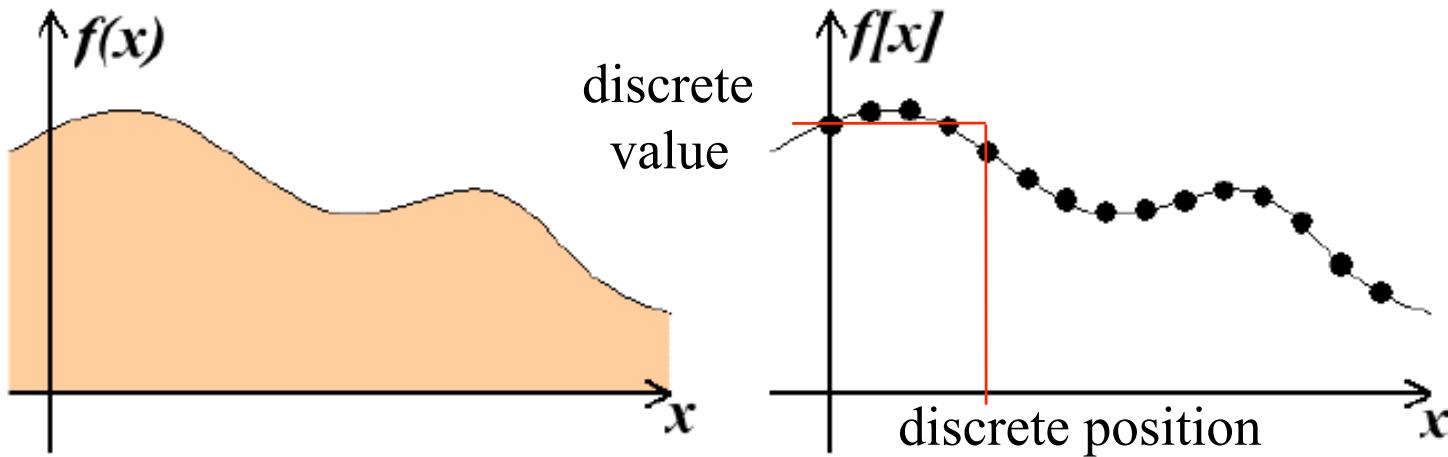
- A pixel is not:
  - a box
  - a disk
  - a teeny tiny little light
- A pixel “looks different” on different display devices
- A pixel is a sample
  - it has no dimension
  - it occupies no area
  - it cannot be seen
  - it has a coordinate
  - it has a value



# More on Samples

---

- In signal processing, the process of mapping a continuous function to a discrete one is called **sampling** (domain of the function)
- The process of mapping a continuous variable to a discrete one is called **quantization** (range of the function)
  - Gamma correction (non linear mapping)
- To represent or render an image using a computer, we must both sample and quantize
  - Today we focus on the effects of sampling and how to fight them



# Sampling & reconstruction

---

The visual array of light is a continuous function

1/ we sample it

- with a digital camera, or with our ray tracer
- This gives us a finite set of numbers,  
not really something we can see
- We are now inside the discrete computer world

2/ we need to get this back to the physical world:  
we reconstruct a continuous function

- for example, the point spread of a pixel on a CRT or LCD
- Both steps can create problems
  - pre-aliasing caused by sampling
  - post-aliasing caused by reconstruction
  - We focus on the former

# Sampling & reconstruction

---

The visual array of light is a continuous function

1/ we sample it

- with a digital camera, or with our ray tracer
- This gives us a finite set of numbers,  
not really something we can see
- We are now inside the discrete computer world

2/ we need to get this back to the physical world:  
we reconstruct a continuous function

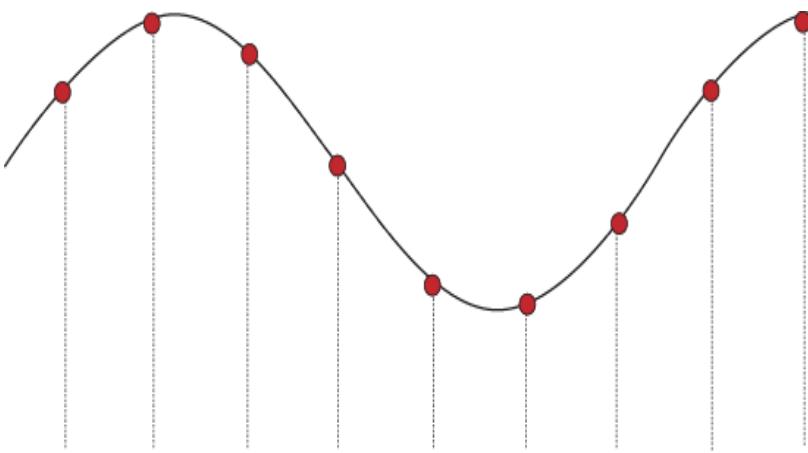
- for example, the point spread of a pixel on a CRT or LCD
- Both steps can create problems
  - pre-aliasing caused by sampling
  - post-aliasing caused by reconstruction
  - We focus on the former

Questions?

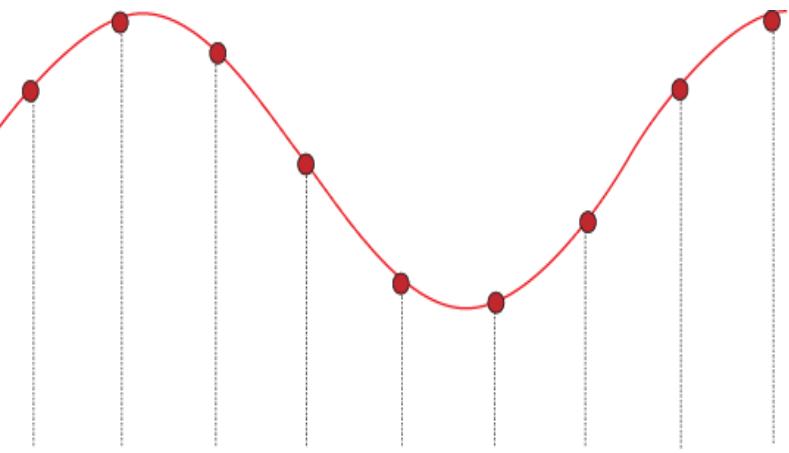
# Sampling Density

---

- If we're lucky, sampling density is enough



Input

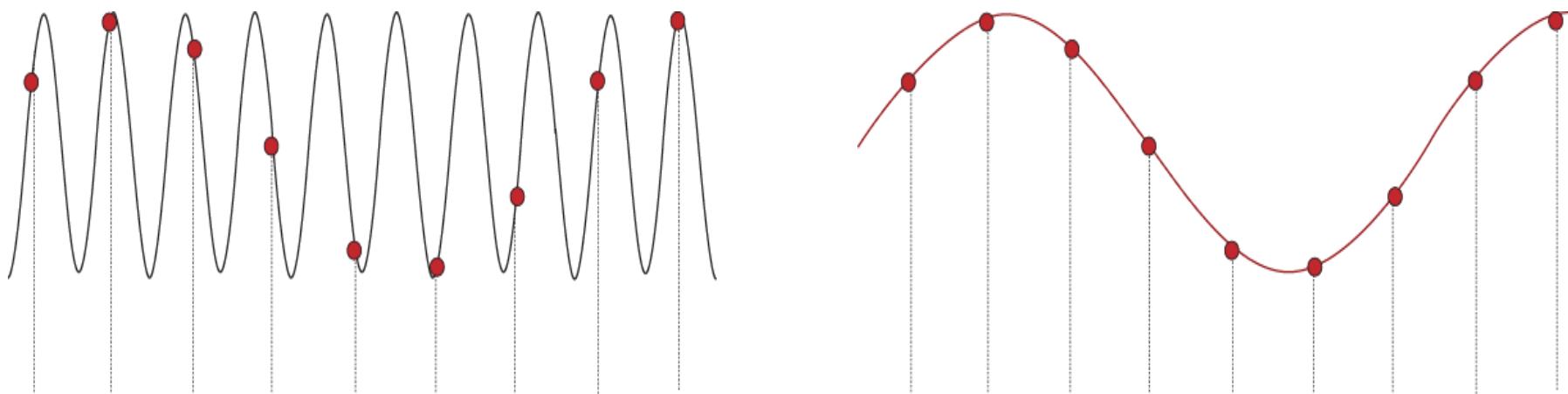


Reconstructed

# Sampling Density

---

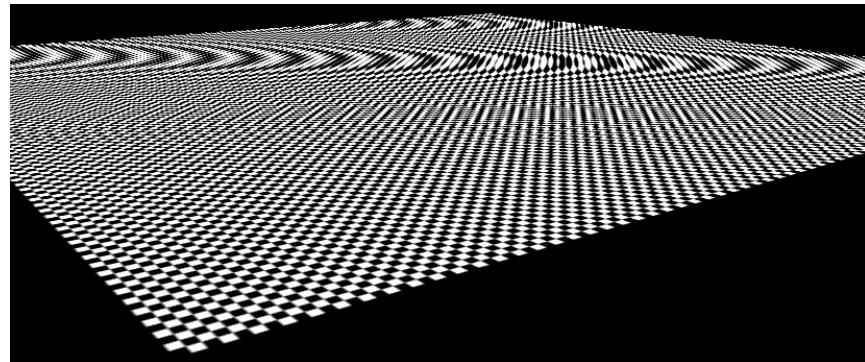
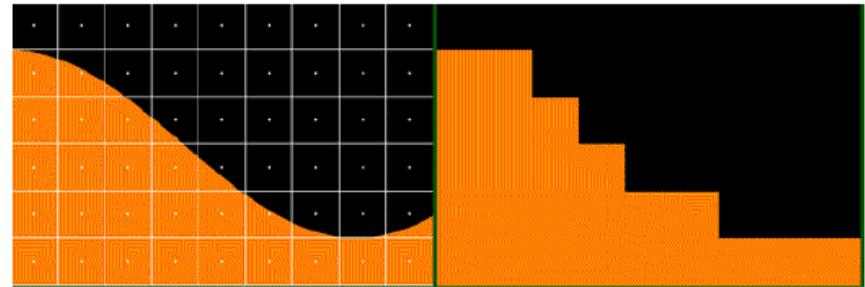
- If we insufficiently sample the signal, it may be mistaken for something simpler during reconstruction (that's aliasing!)
- This is why it's called aliasing: the new low-frequency sine wave is an alias/ghost of the high-frequency one



# Discussion

---

- Types of aliasing
  - Edges
    - mostly directional aliasing  
(vertical and horizontal edges rather than actual slope)
  - Repetitive textures
    - Paradigm of aliasing
    - Harder to solve right
    - Motivates fun mathematics



# Solution?

---

- How do we avoid that high-frequency patterns mess up our image?

# Solution?

---

- How do we avoid that high-frequency patterns mess up our image?
- We blur!
  - In the case of audio, people first include an analog low-pass filter before sampling
  - For ray tracing/rasterization: compute at higher resolution, blur, resample at lower resolution
  - For textures, we can also blur the texture image before doing the lookup
- To understand what really happens, we need serious math

# Solution?

# Questions?

---

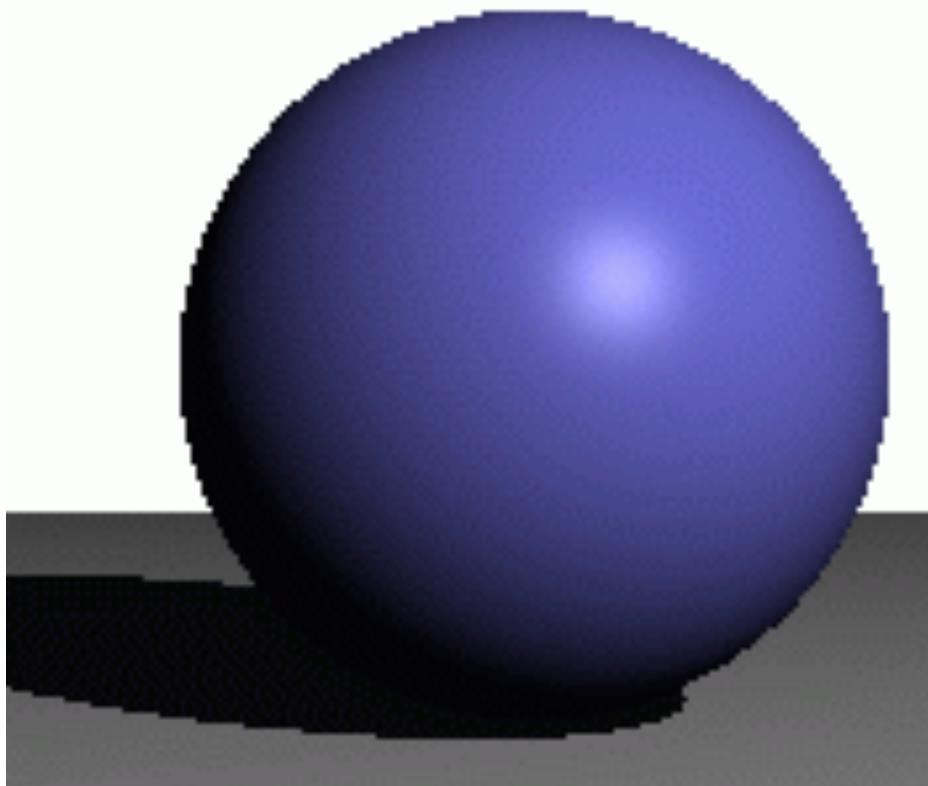
- How do we avoid that high-frequency patterns mess up our image?
- We blur!
  - In the case of audio, people first include an analog low-pass filter before sampling
  - For ray tracing/rasterization: compute at higher resolution, blur, resample at lower resolution
  - For textures, we can also blur the texture image before doing the lookup
- To understand what really happens, we need serious math

# In practice: Supersampling

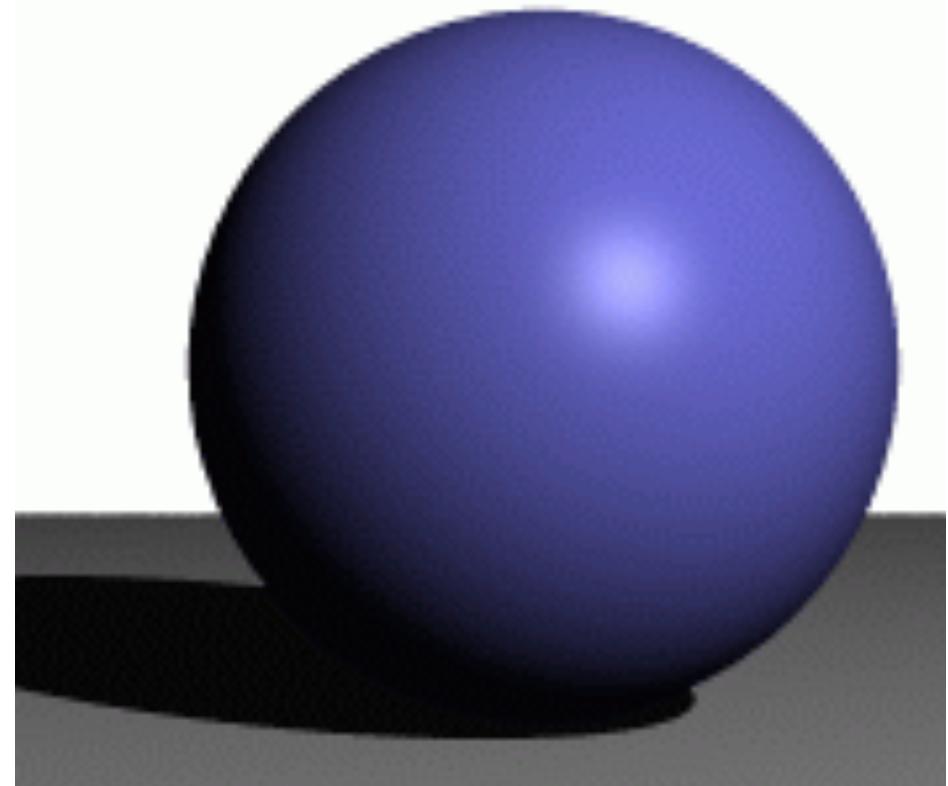
---

- Your intuitive solution is to compute multiple color values per pixel and average them

jaggies



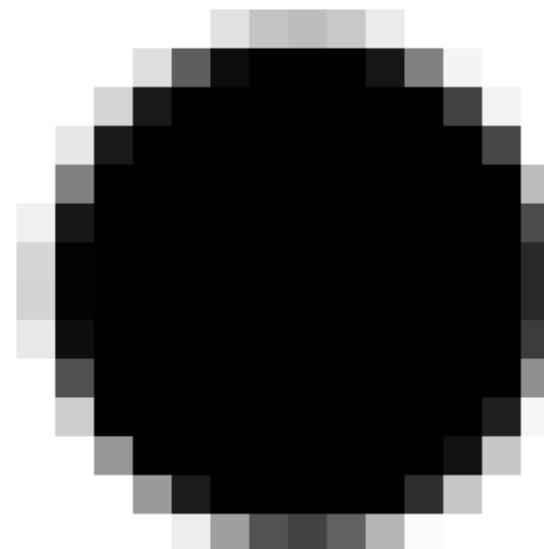
w/ antialiasing



# Uniform supersampling

---

- Compute image at resolution  $k * \text{width}$ ,  $k * \text{height}$
- Downsample using low-pass filter  
(e.g. Gaussian, sinc, bicubic)



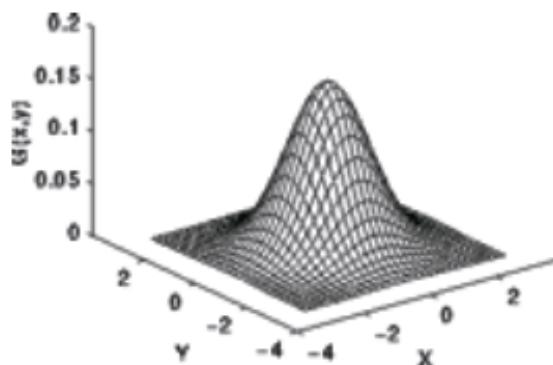
# Low pass / convolution

---

- Each output (low-res) pixel is a weighted average of input subsamples
- Weight depends on relative spatial position
- For example:
  - Gaussian as a function of distance
  - 1 inside a square, zero outside (box)

$$\frac{1}{273}$$

|   |    |    |    |   |
|---|----|----|----|---|
| 1 | 4  | 7  | 4  | 1 |
| 4 | 16 | 26 | 16 | 4 |
| 7 | 26 | 41 | 26 | 7 |
| 4 | 16 | 26 | 16 | 4 |
| 1 | 4  | 7  | 4  | 1 |

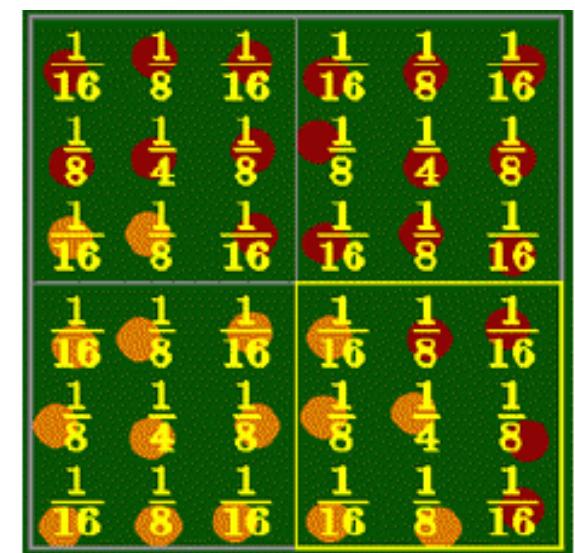
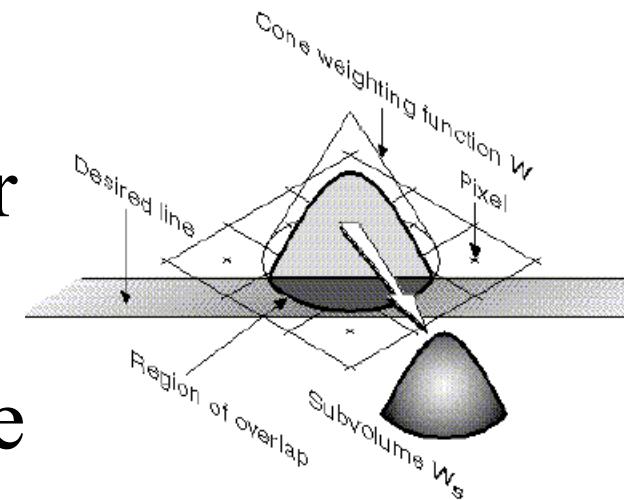


Gaussian

# In practice: Supersampling

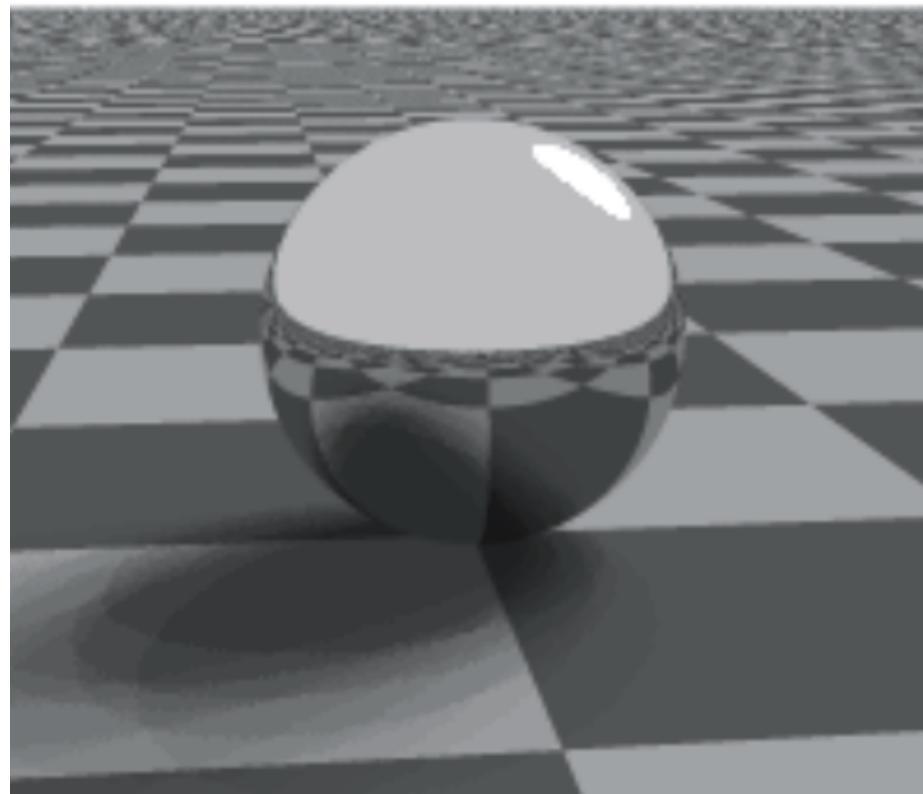
---

- Your intuitive solution is to compute multiple color values per pixel and average them
- A better interpretation of the same idea is that
  - You first create a higher resolution image
  - You blur it (low pass, prefilter)
  - You resample it at a lower resolution

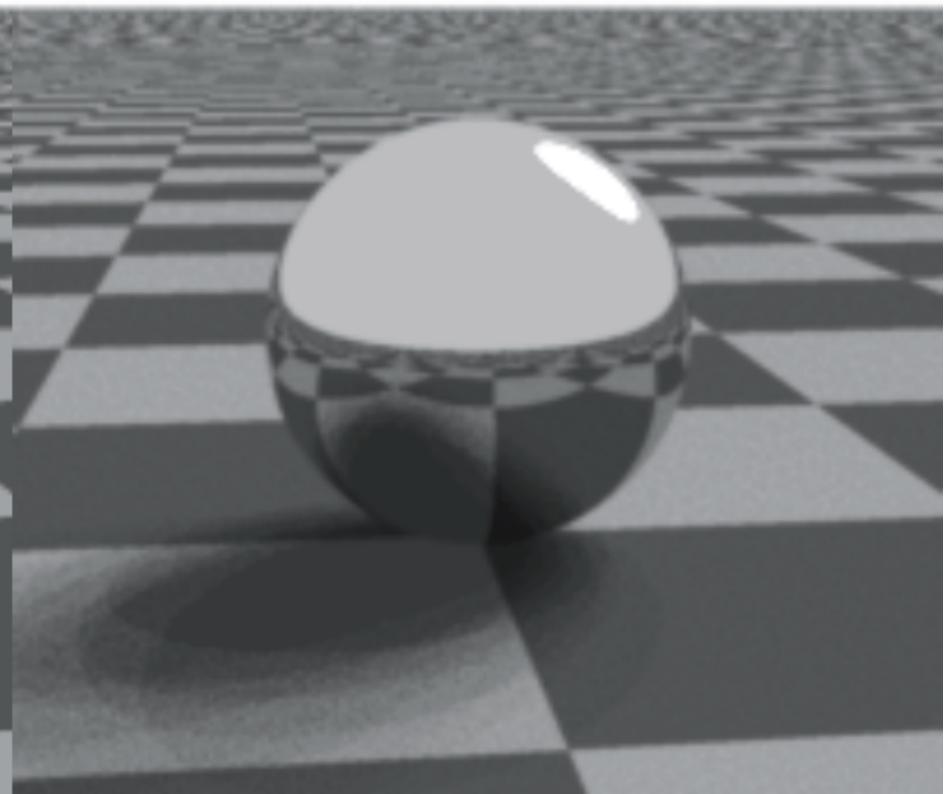


---

Box



Gaussian

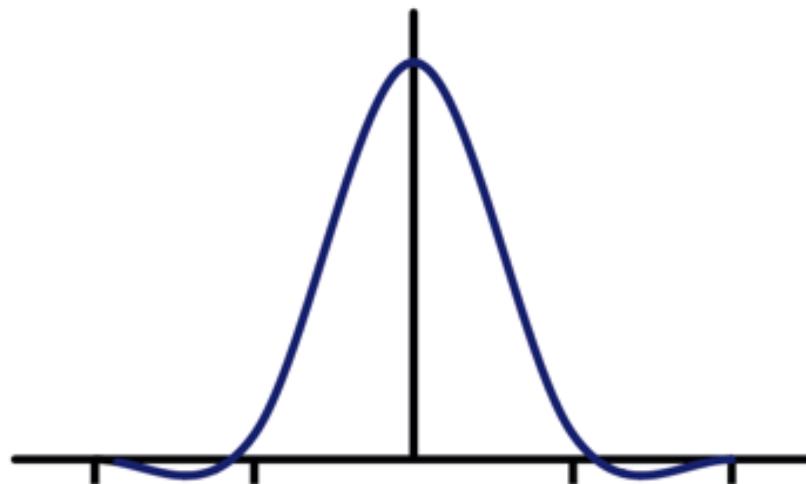


<http://www.cs.unc.edu/~narain/courses/comp870/a1-sampling/>

# Recommended filter

---

- Bicubic
  - [http://www.mentallandscape.com/  
Papers\\_siggraph88.pdf](http://www.mentallandscape.com/Papers_siggraph88.pdf)
- Good tradeoff between sharpness and aliasing



[http://de.wikipedia.org/wiki/Datei:Mitchell\\_Filter.svg](http://de.wikipedia.org/wiki/Datei:Mitchell_Filter.svg)

# Piecewise-cubic

---

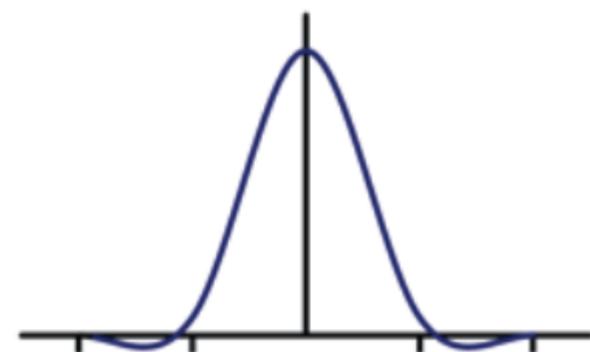
- General formula

$$k(x) = \begin{cases} P|x|^3 + Q|x|^2 + R|x| + S & \text{if } |x| < 1 \\ T|x|^3 + U|x|^2 + V|x| + W & \text{if } 1 \leq |x| < 2 \\ 0 & \text{otherwise} \end{cases}$$

where P, Q, R, S, T, U, V, W are parameters

- But we want the derivatives to be zero at the boundary and constant signals to be well reconstructed. Reduces to 2 parameters

$$k(x) = \frac{1}{6} \begin{cases} (12 - 9B - 6C)|x|^3 + & \text{if } |x| < 1 \\ (-18 + 12B + 6C)|x|^2 + (6 - 2B) & \\ (-B - 6C)|x|^3 + (6B + 30C)|x|^2 + & \text{if } 1 \leq |x| < 2 \\ (-12B - 48C)|x| + (8B + 24C) & \\ 0 & \text{otherwise} \end{cases}$$



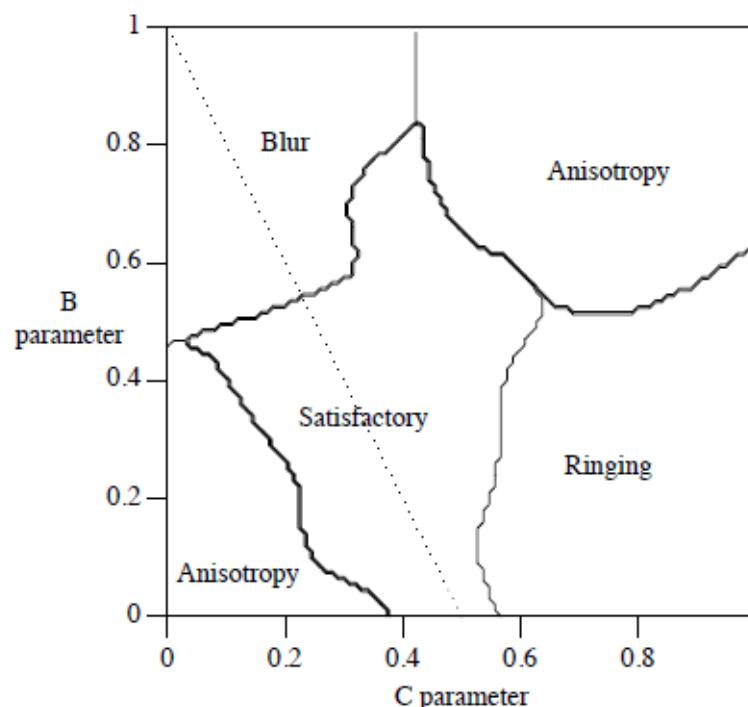
# Choosing the parameters

---

- Empirical tests determined usable parameters
  - Mitchell, Don and Arun Netravali, "Reconstruction Filters in Computer Graphics", SIGGRAPH 88.

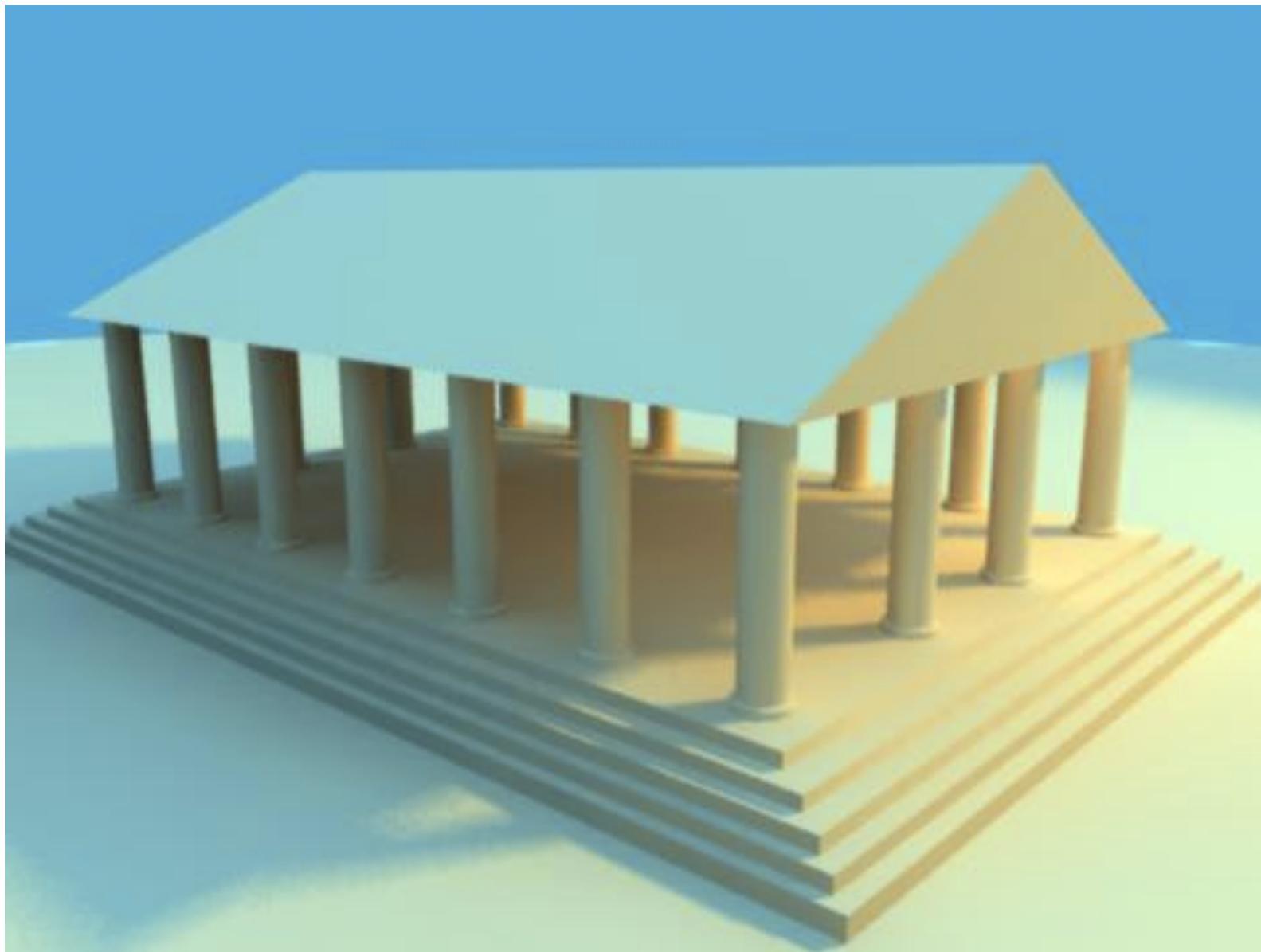
[http://www.mentallandscape.com/Papers\\_siggraph88.pdf](http://www.mentallandscape.com/Papers_siggraph88.pdf)

<http://dl.acm.org/citation.cfm?id=378514>



# Box

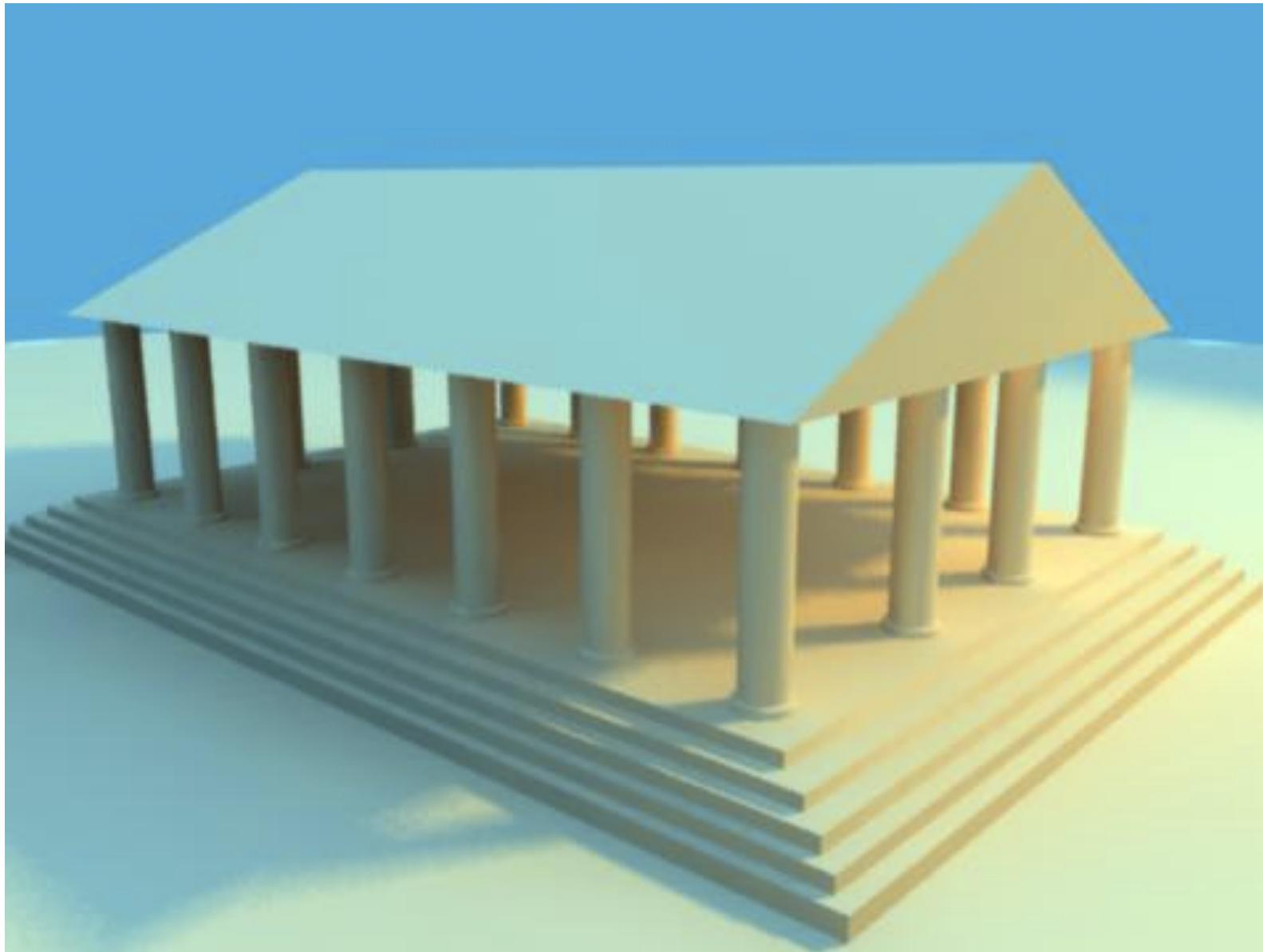
---



[http://www.willsmith.org/maya\\_gi\\_tutorial\\_1/Wills\\_Maya GI\\_Tweaking\\_Guide\\_1.html](http://www.willsmith.org/maya_gi_tutorial_1/Wills_Maya GI_Tweaking_Guide_1.html)

# Gauss

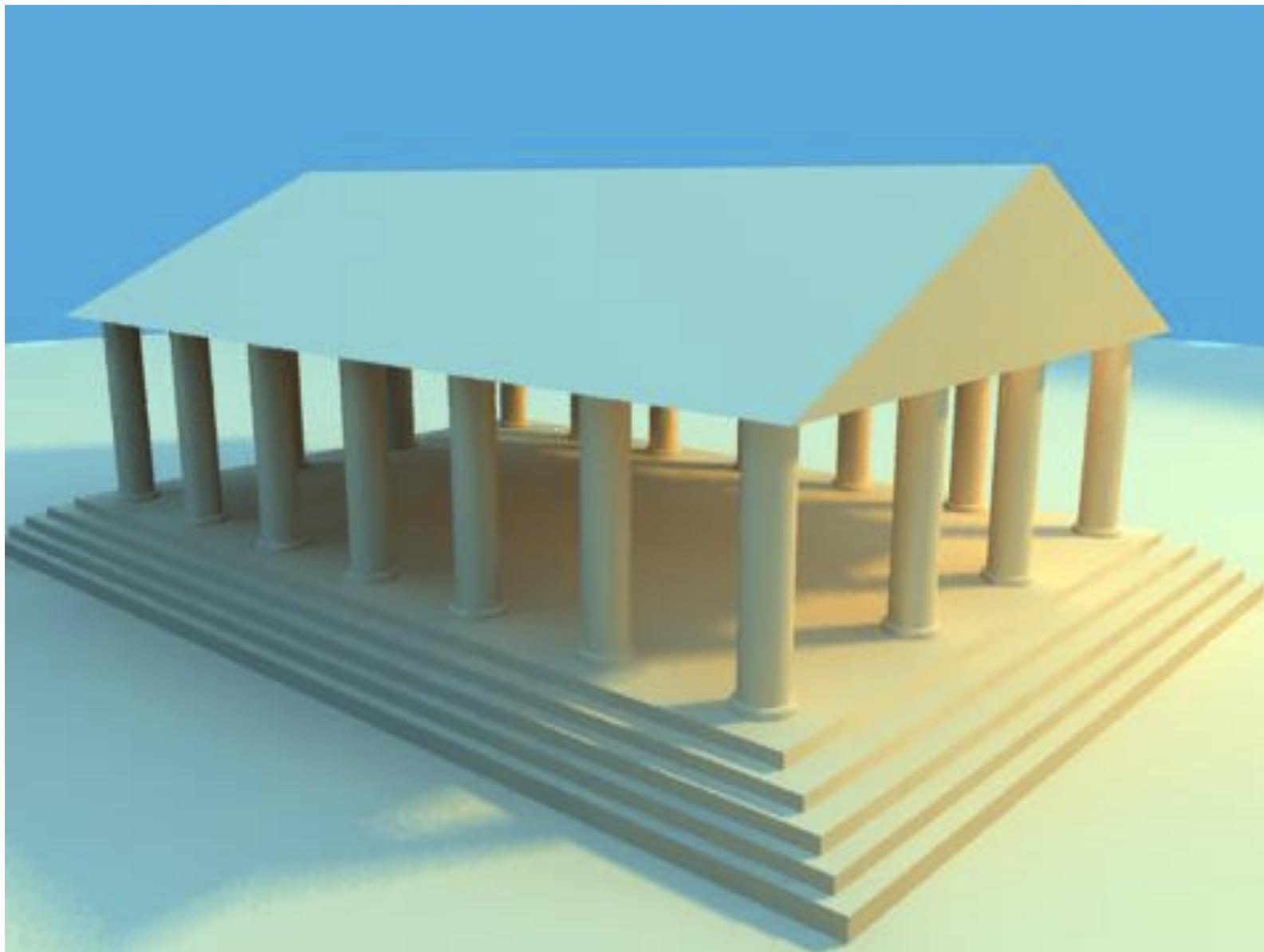
---



[http://www.willsmith.org/maya\\_gi\\_tutorial\\_1/Wills\\_Maya GI\\_Tweaking\\_Guide\\_1.html](http://www.willsmith.org/maya_gi_tutorial_1/Wills_Maya GI_Tweaking_Guide_1.html)

# Mitchell bicubic

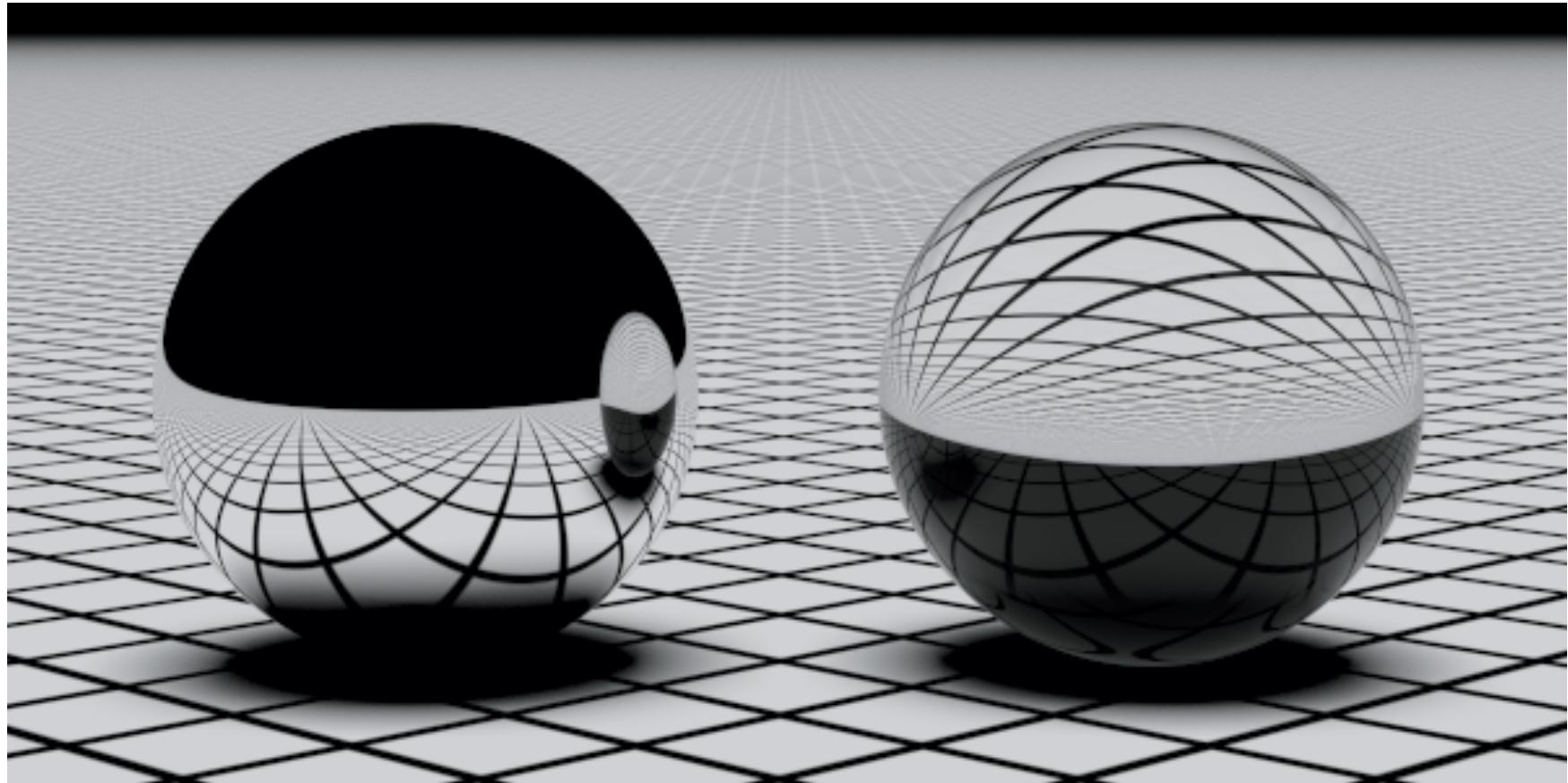
---



[http://www.willsmith.org/maya\\_gi\\_tutorial\\_1/Wills\\_Maya GI\\_Tweaking\\_Guide\\_1.html](http://www.willsmith.org/maya_gi_tutorial_1/Wills_Maya GI_Tweaking_Guide_1.html)

# Box

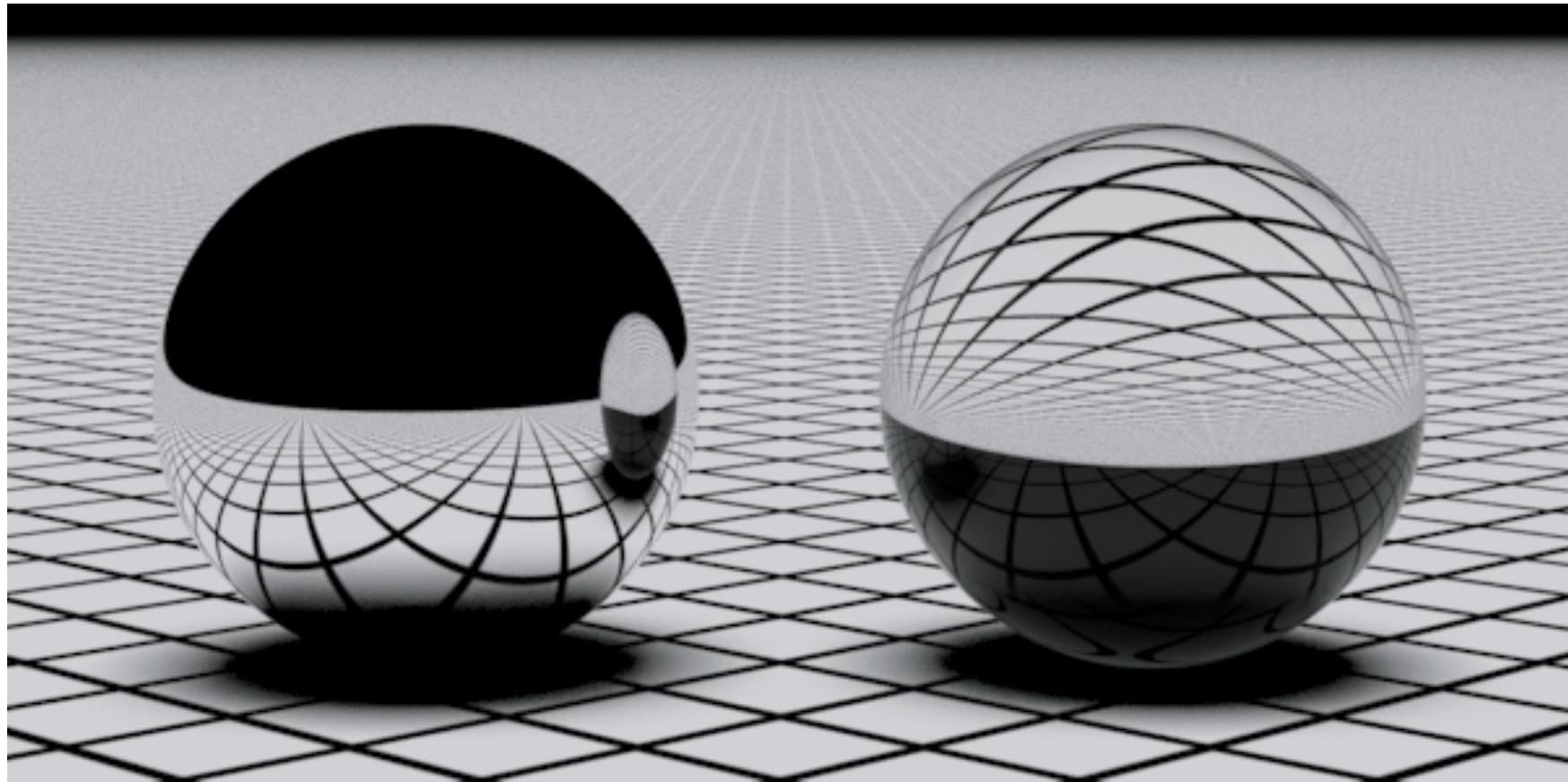
---



<http://rise.sourceforge.net/cgi-bin/makepage.cgi?Filtering>

# Mitchell-Netravali cubic (1/3. 1/3)

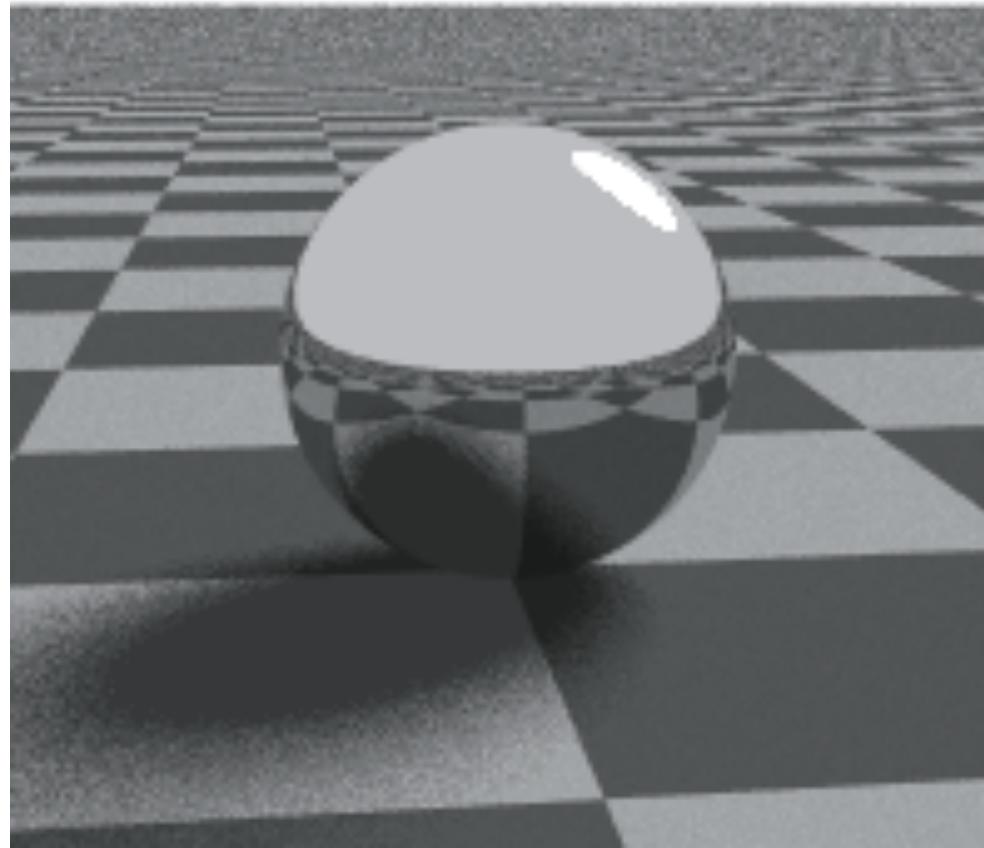
---



<http://rise.sourceforge.net/cgi-bin/makepage.cgi?Filtering>

# Box

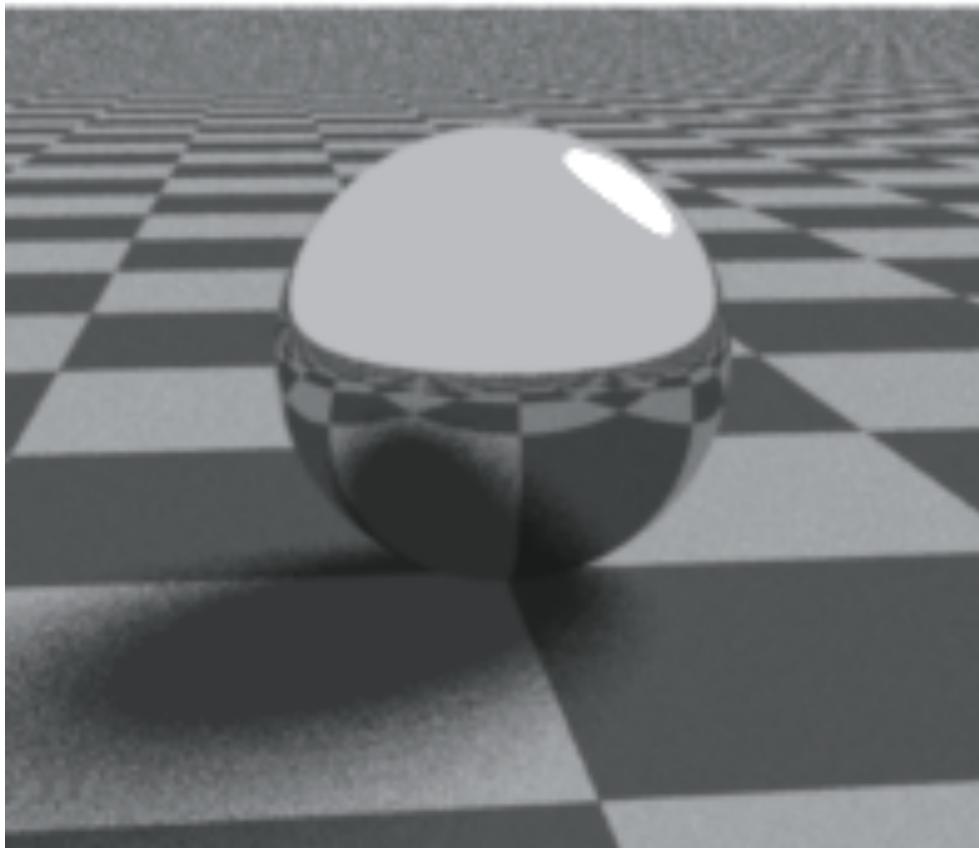
---



<http://www.cs.unc.edu/~narain/courses/comp870/a1-sampling/>

# Gaussian

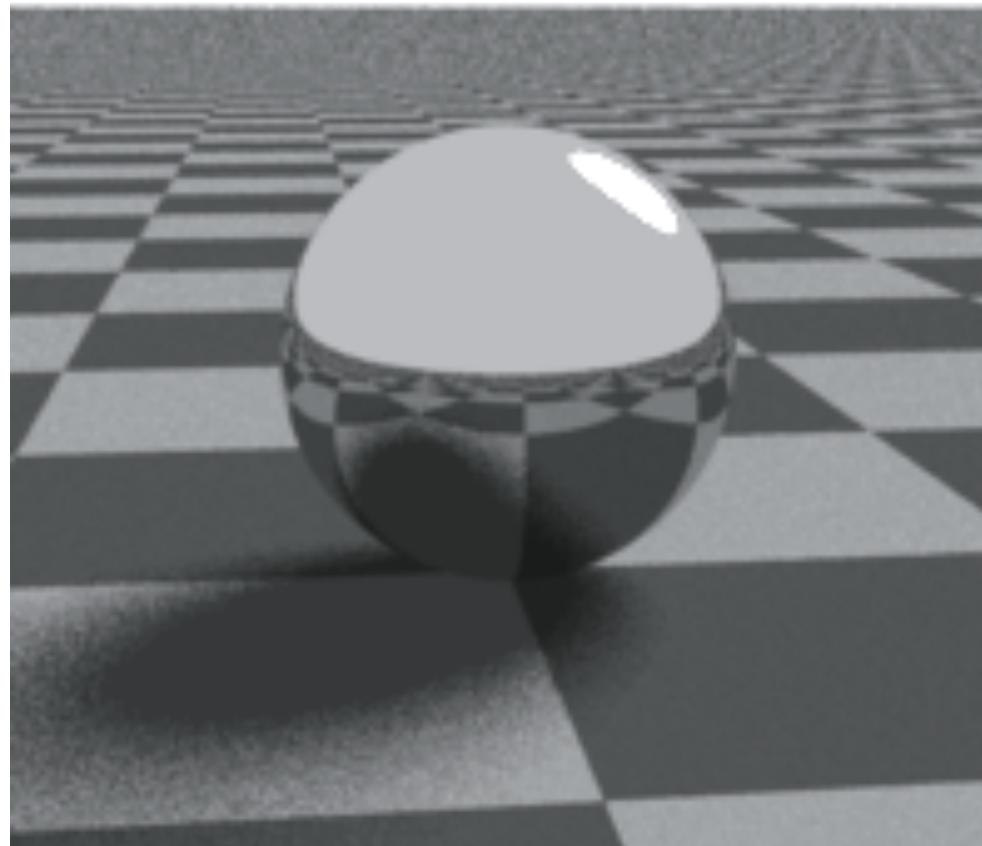
---



<http://www.cs.unc.edu/~narain/courses/comp870/a1-sampling/>

# Mitchell-Netravali cubic

---

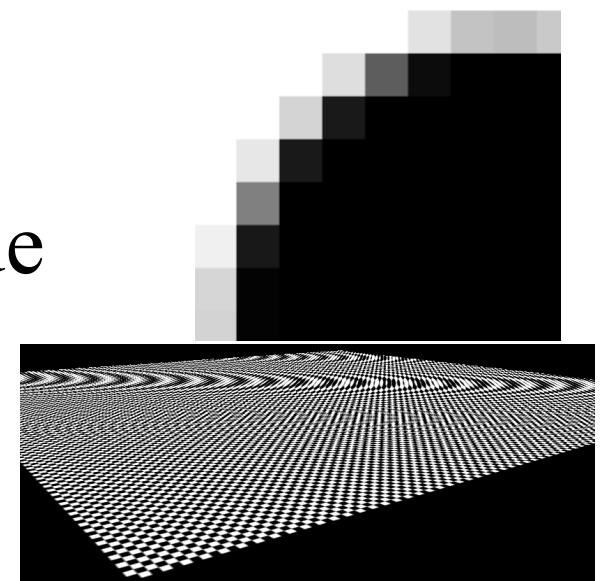


<http://www.cs.unc.edu/~narain/courses/comp870/a1-sampling/>

# Uniform supersampling

---

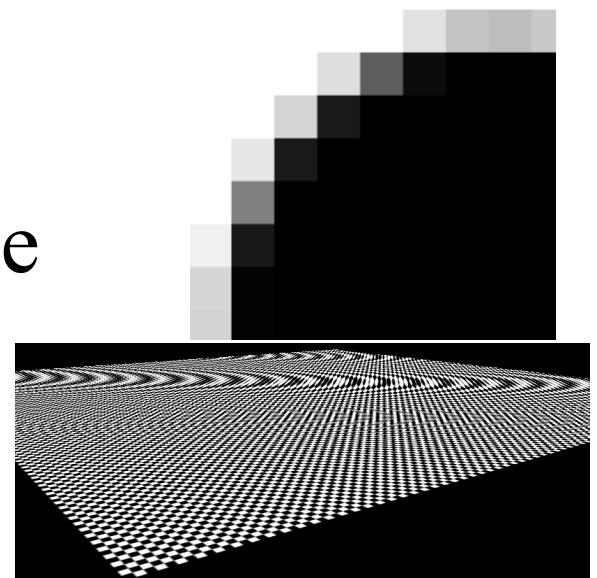
- Advantage:
  - The first (super)sampling captures more high frequencies that are not aliased
  - Downsampling can use a good filter
- Issues
  - Frequencies above the (super)sampling limit are still aliased
- Works well for edges, since spectrum replication is less an issue
- Not as well for repetitive textures
  - But solution soon



# Uniform supersampling Questions?

---

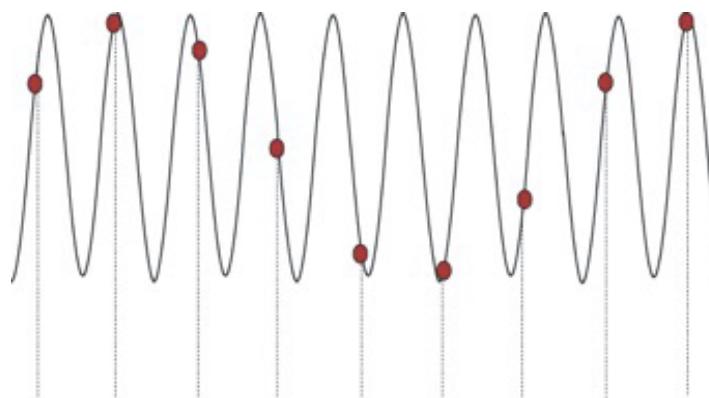
- Advantage:
  - The first (super)sampling captures more high frequencies that are not aliased
  - Downsampling can use a good filter
- Issues
  - Frequencies above the (super)sampling limit are still aliased
- Works well for edges, since spectrum replication is less an issue
- Not as well for repetitive textures
  - But solution soon



# Uniform supersampling

---

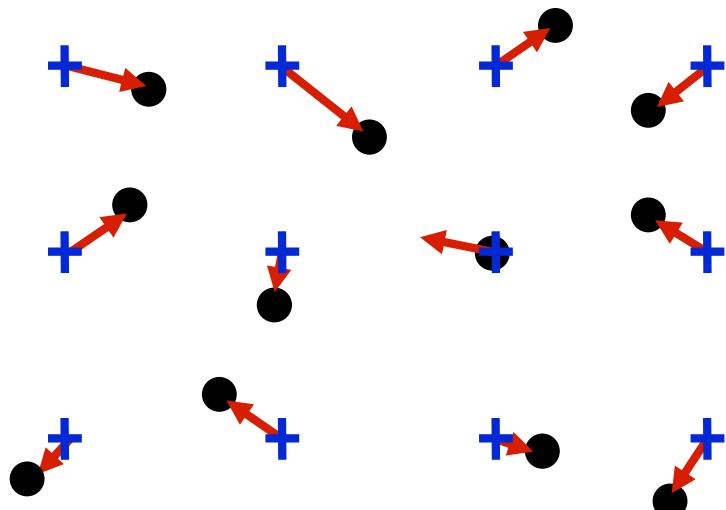
- Problem: supersampling only pushes the problem further: The signal is still not bandlimited
- Aliasing happens
- Especially if the signal and the sampling are regular



# Jittering

---

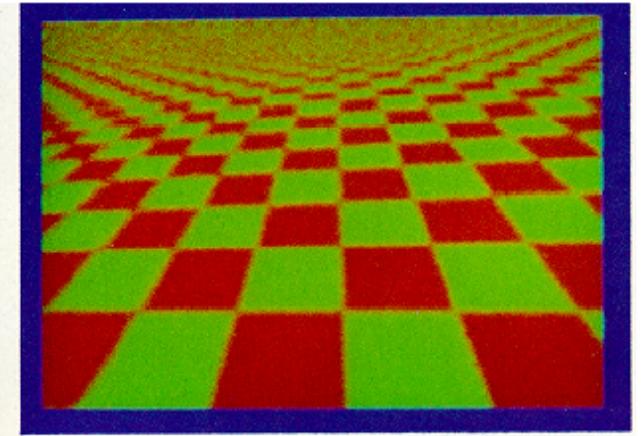
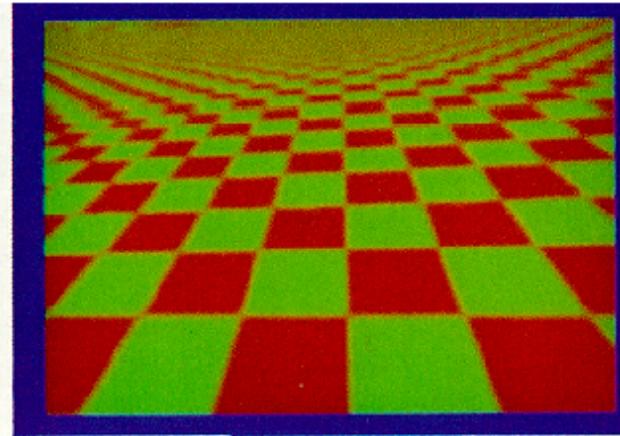
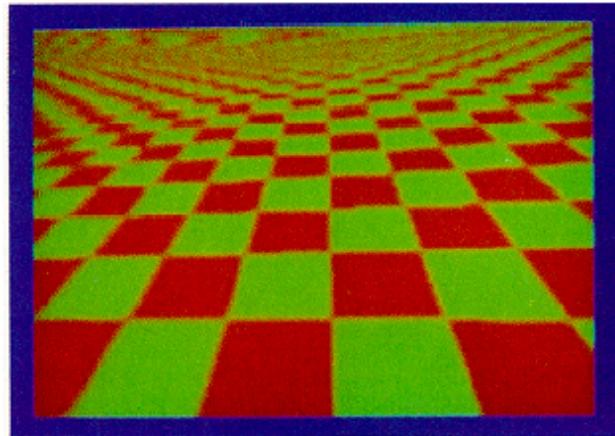
- Uniform sample + random perturbation
- Sampling is now non-uniform
- Signal processing gets more complex
- In practice, adds noise to image
- But noise is better than aliasing Moiré patterns



# Jittered supersampling

---

1 sample / pixel



0 jittering  
by 1

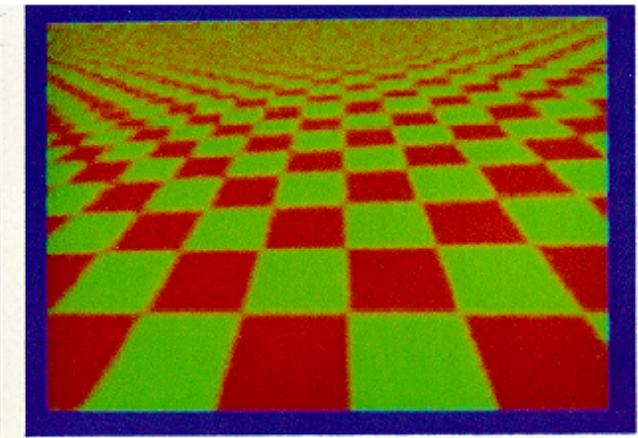
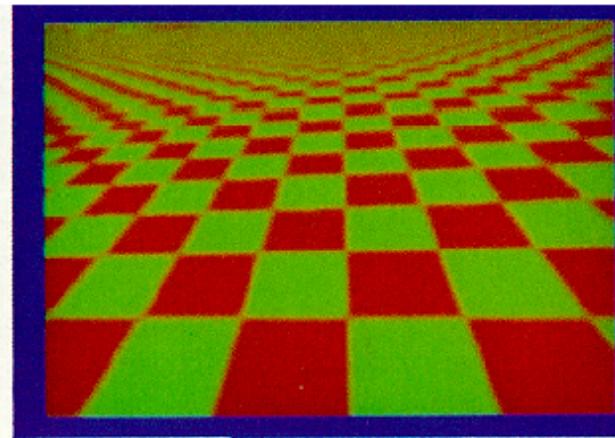
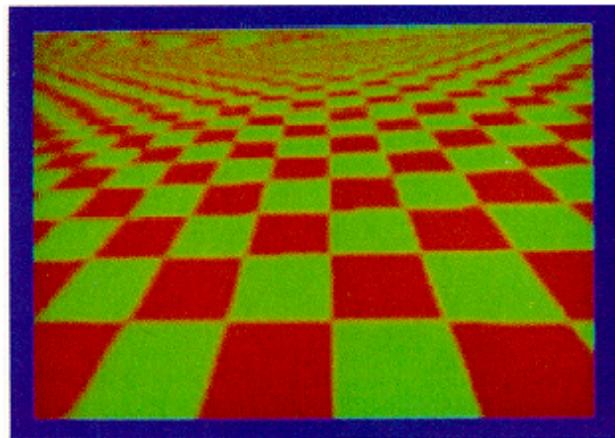
jittering by 0.5

jittering 38

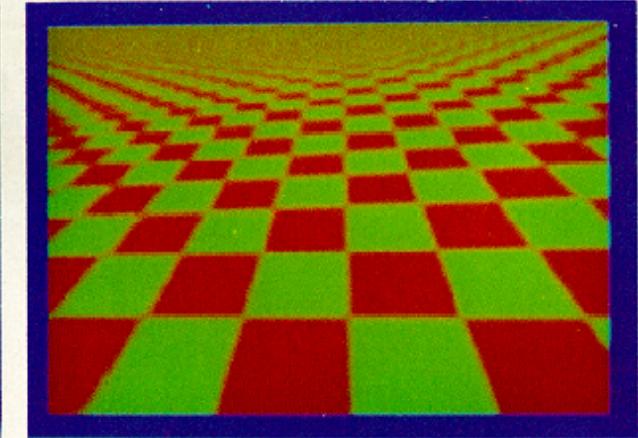
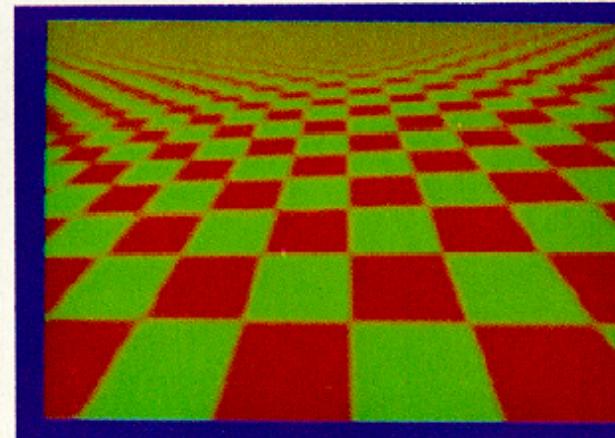
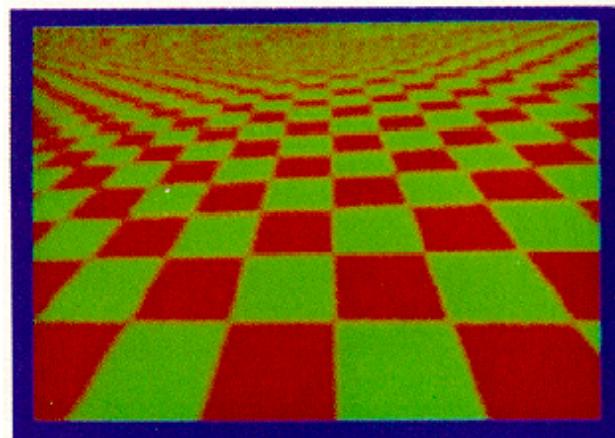
# Jittered supersampling

---

1 sample / pixel



2 sample / pixel



0 jittering  
by 1

jittering by 0.5

jittering  
39

# Jittering

---

- Displaced by a vector a fraction of the size of the subpixel distance
- Low-frequency Moire (aliasing) pattern replaced by noise
- Extremely effective
- Patented by Pixar!
- When jittering amount is 1, equivalent to stratified sampling (cf. later)

# Recap: image antialiasing

---

- Render multiple samples per pixel
- Jitter the sample locations
- Use appropriate filter to reconstruct final image
  - Bicubic for example

# Recap: image antialiasing

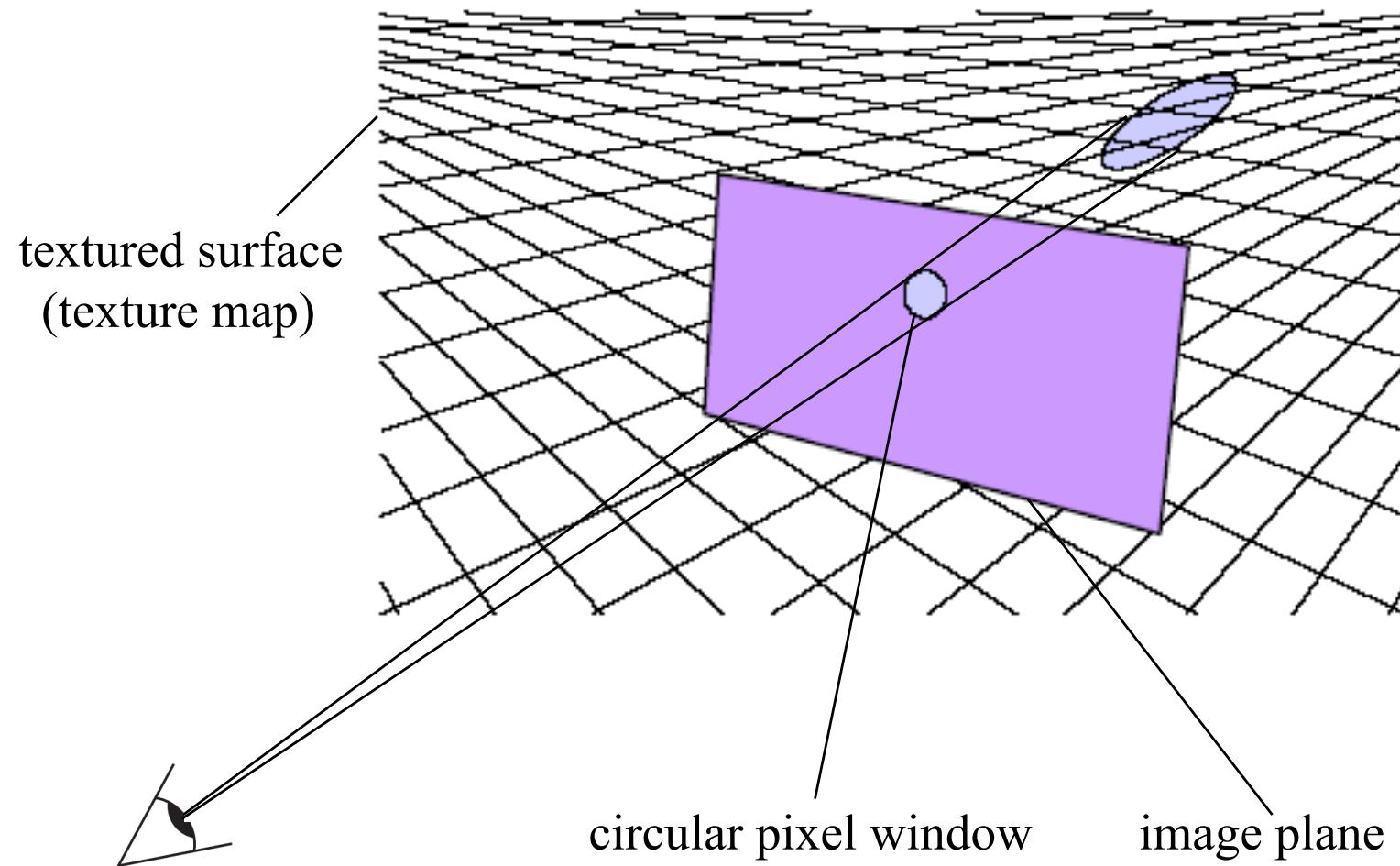
---

- Render multiple samples per pixel
- Jitter the sample locations
- Use appropriate filter to reconstruct final image
  - Bicubic for example

Questions?

# Sampling Texture Maps

---

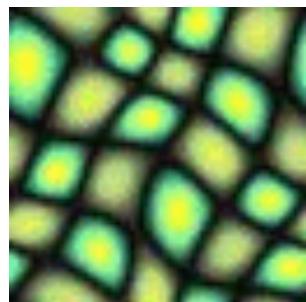


- How to map the texture area seen through the pixel window to a single pixel value?

# Sampling Texture Maps

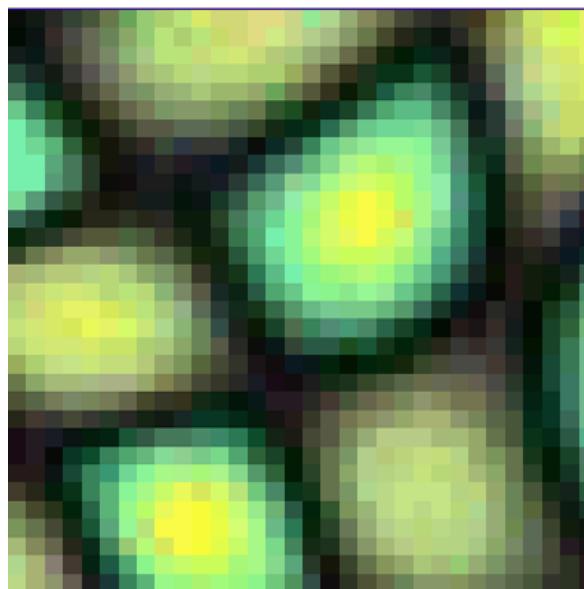
---

- When texture mapping it is rare that the screen-space sampling density matches the sampling density of the texture.

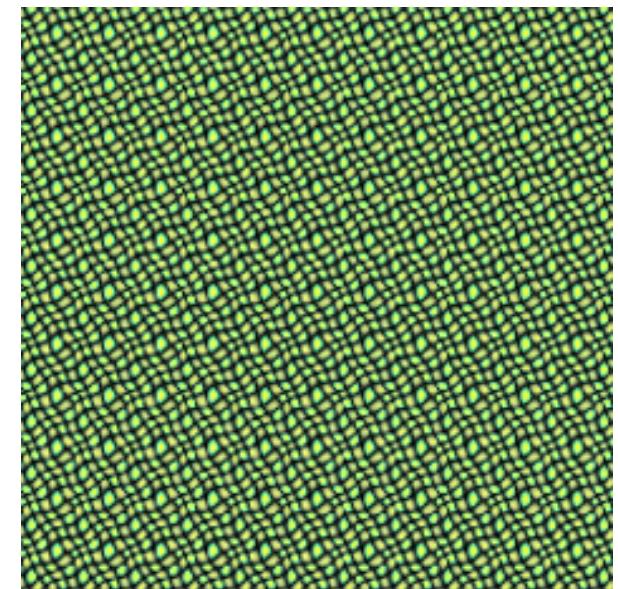


64x64 pixels

Original Texture



Magnification for Display

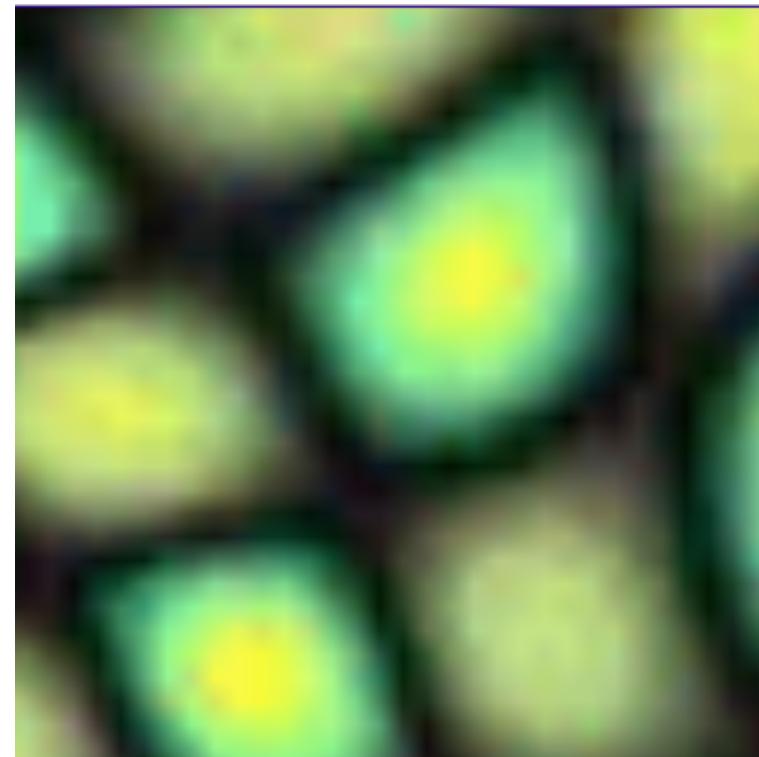
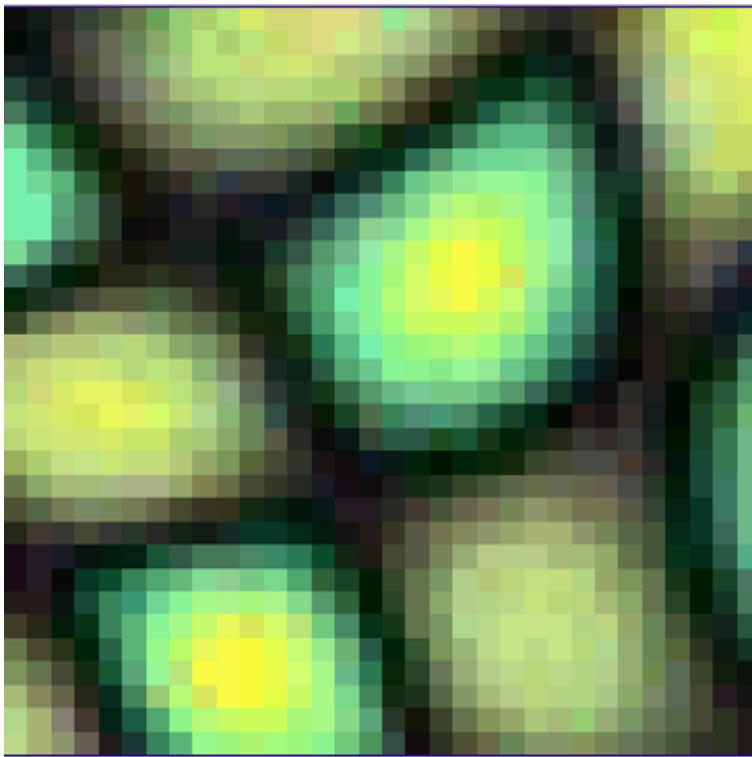


Minification for Display

# Linear Interpolation

---

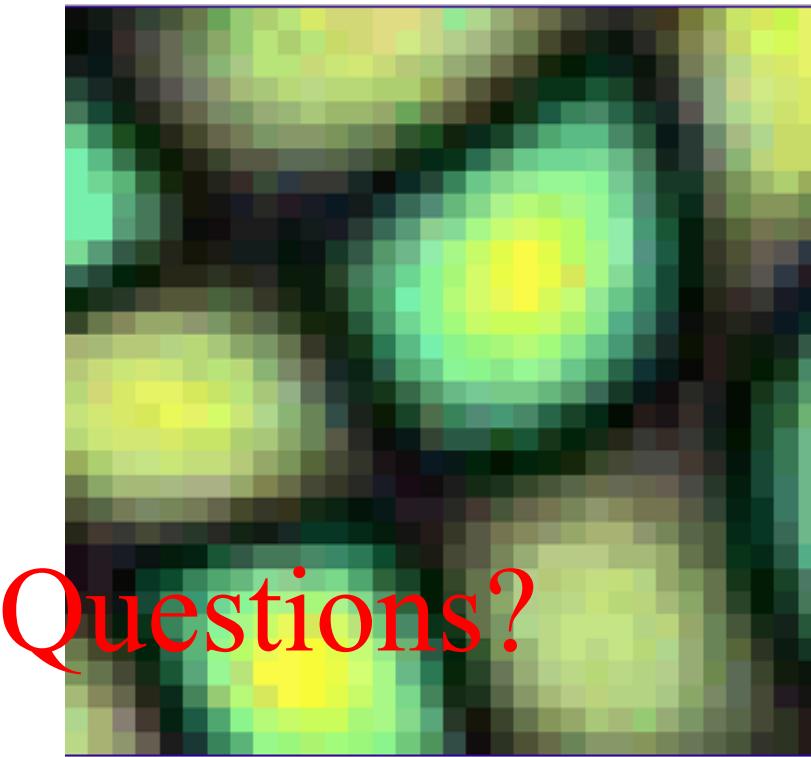
- Tell OpenGL to use a tent filter instead of a box filter.
- Magnification looks better, but blurry
  - (texture is under-sampled for this resolution)
  - Oh well.



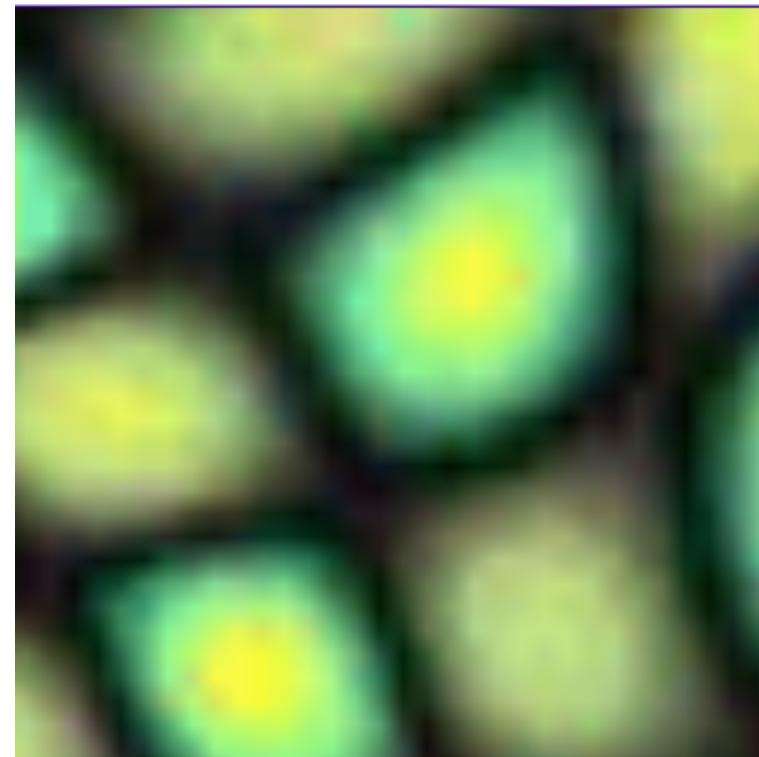
# Linear Interpolation

---

- Tell OpenGL to use a tent filter instead of a box filter.
- Magnification looks better, but blurry
  - (texture is under-sampled for this resolution)
  - Oh well.

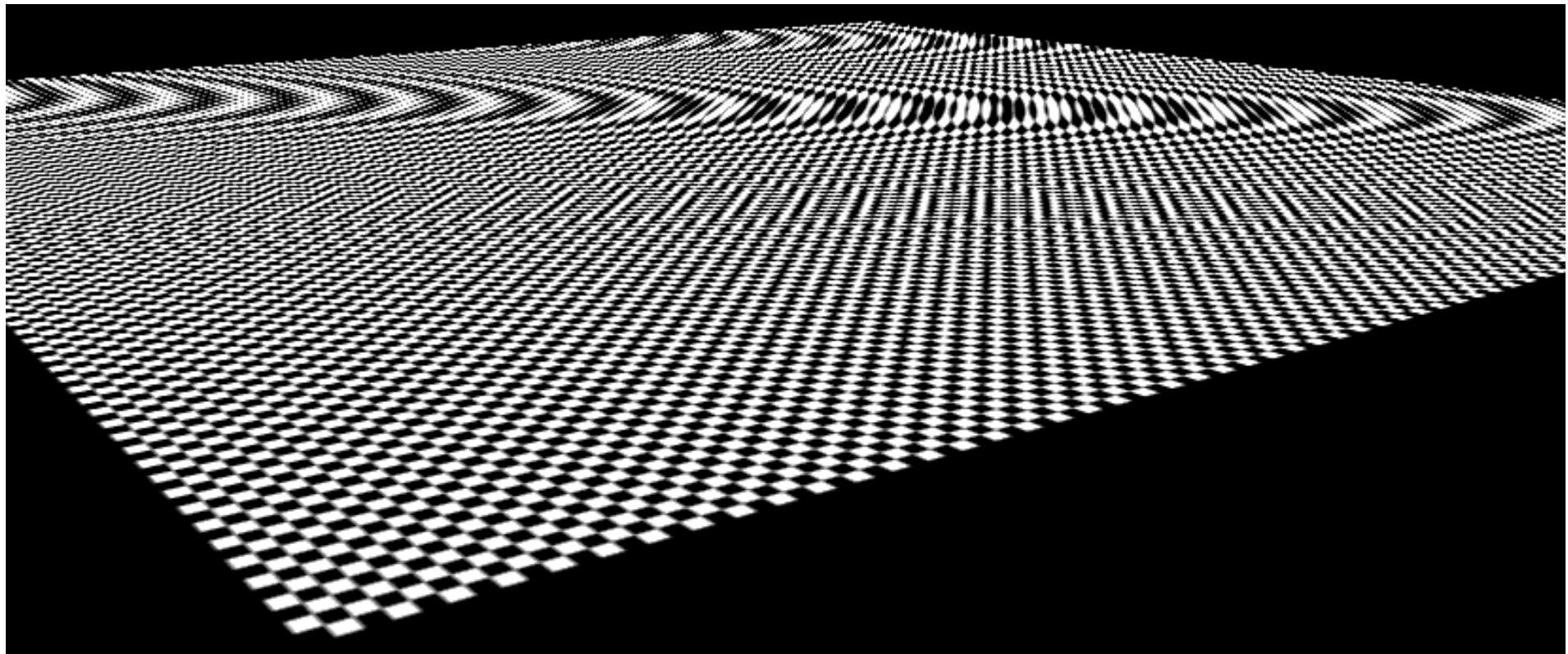


Questions?



# Minification: Examples of Aliasing

---



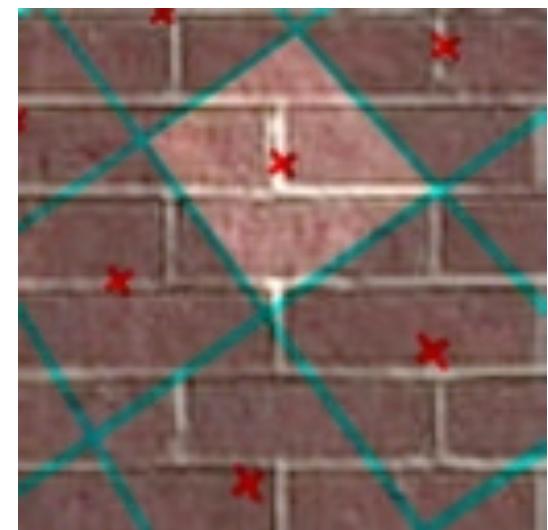
# Spatial Filtering

---

- Remove the high frequencies which cause artifacts in texture minification.
- Compute a spatial integration over the extent of the pixel
- This is equivalent to convolving the texture with a filter kernel centered at the sample (i.e., pixel center)!
- Expensive to do during rasterization, but an approximation it can be precomputed



projected texture in image plane

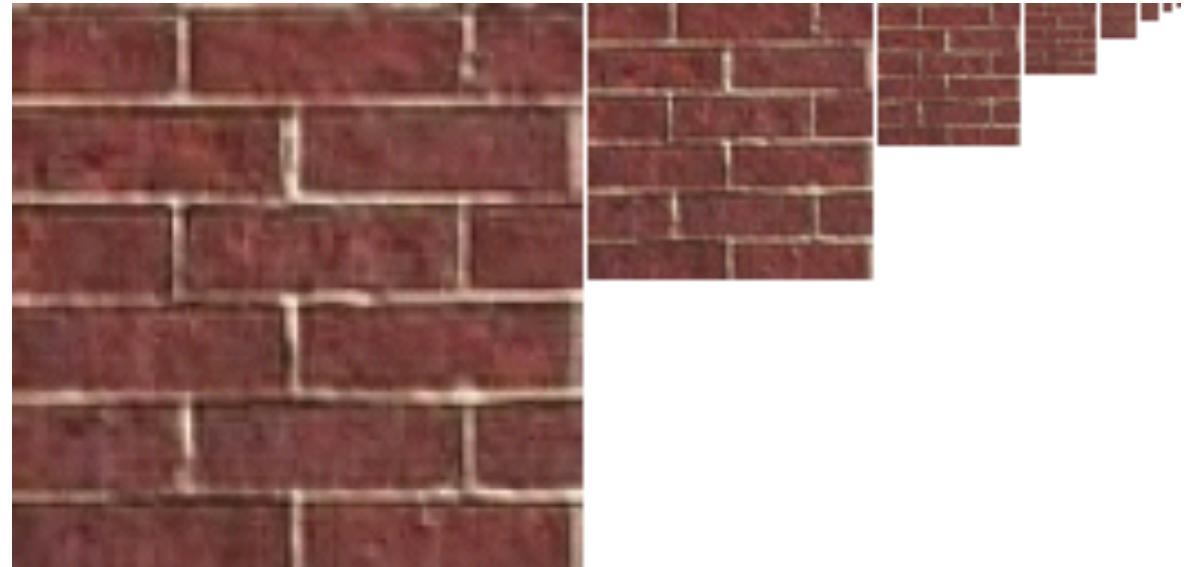


pixels projected in texture plane

# MIP Mapping

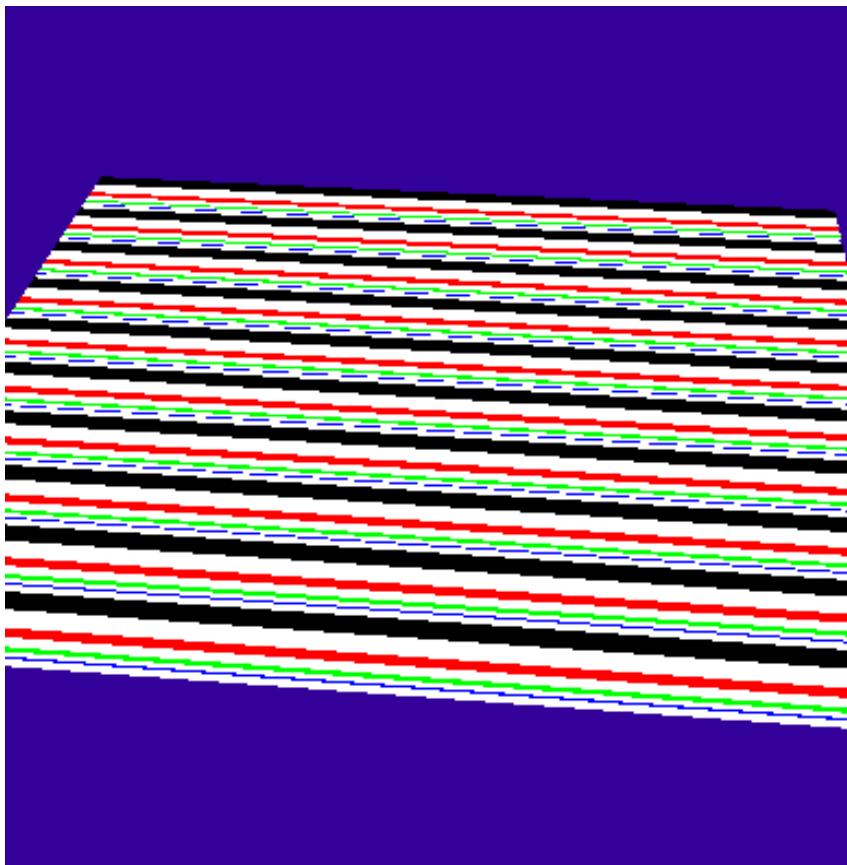
---

- Construct a pyramid of images that are pre-filtered and re-sampled at  $1/2$ ,  $1/4$ ,  $1/8$ , etc., of the original image's sampling
- During rasterization we compute the index of the decimated image that is sampled at a rate closest to the density of our desired sampling rate
- MIP stands for multum in parvo which means many in a small place

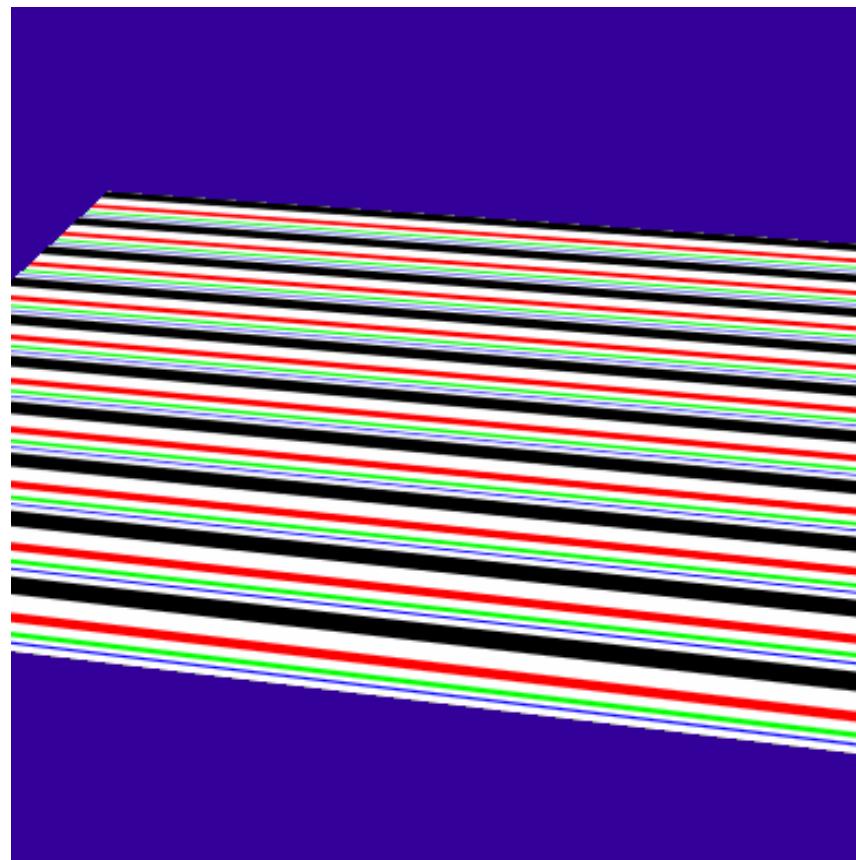


# MIP Mapping Example

---



Nearest Neighbor

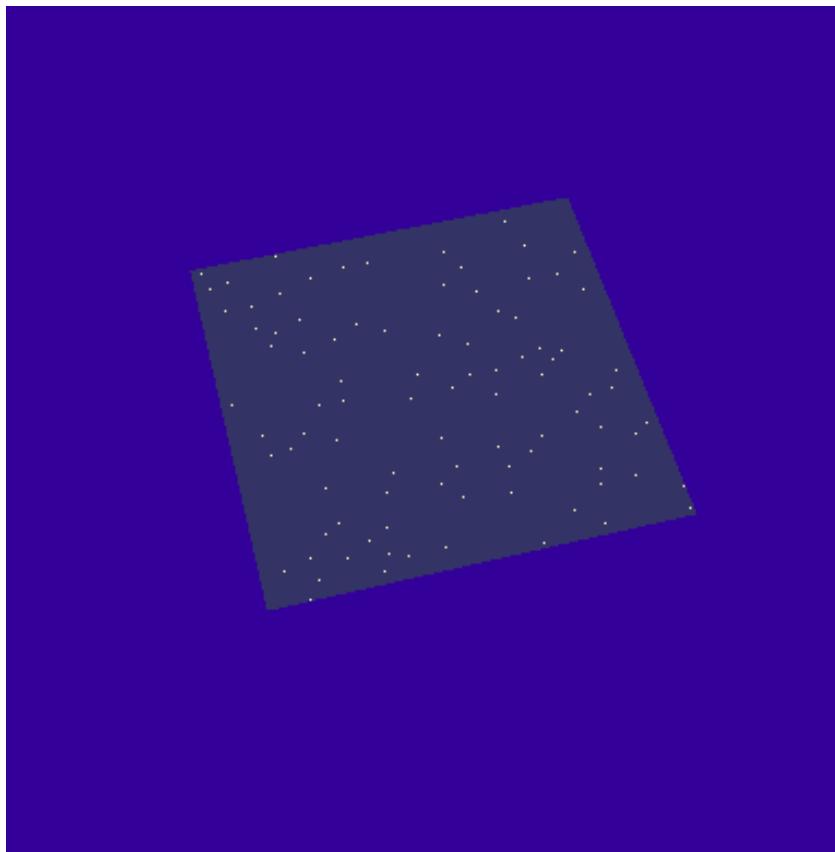


MIP Mapped (Bi-Linear)

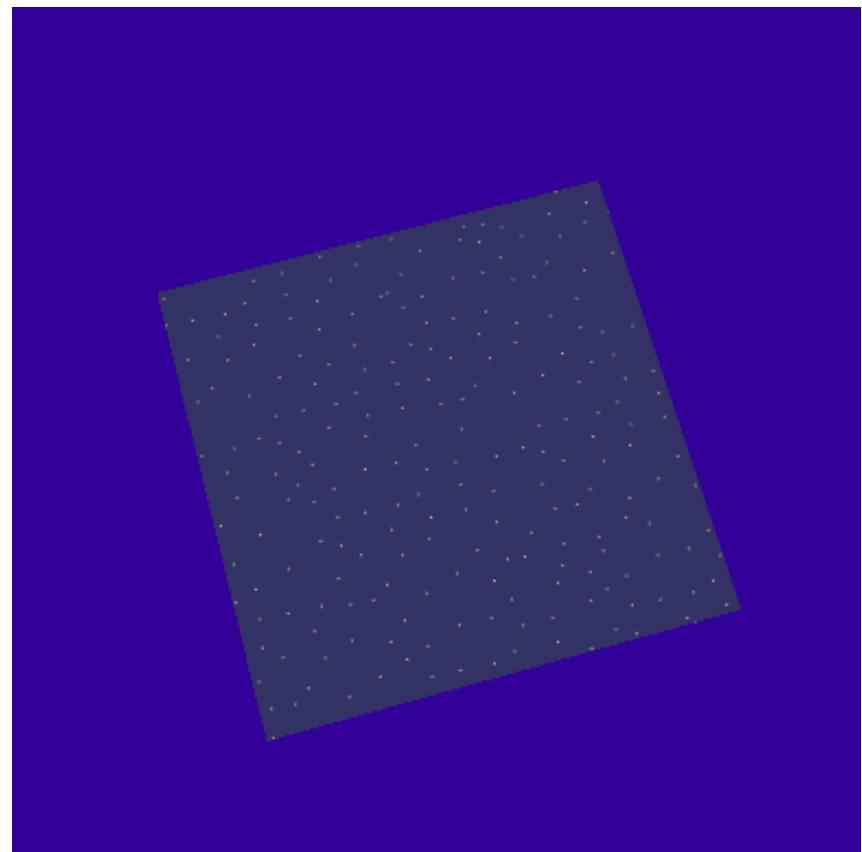
# MIP Mapping Example

---

- Small details may "pop" in and out of view



Nearest Neighbor

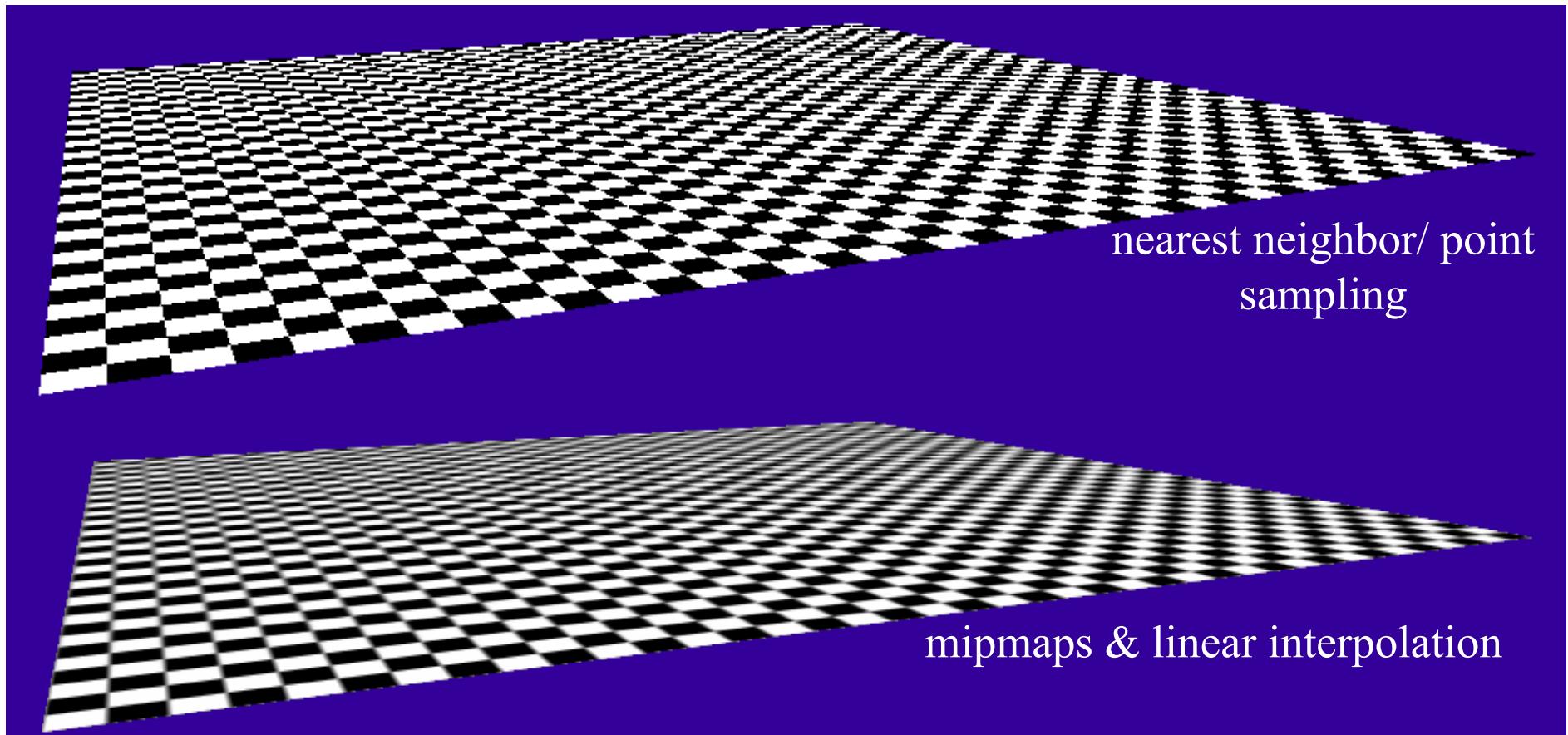


MIP Mapped (Bi-Linear)

# Examples of Aliasing

---

## Texture Errors

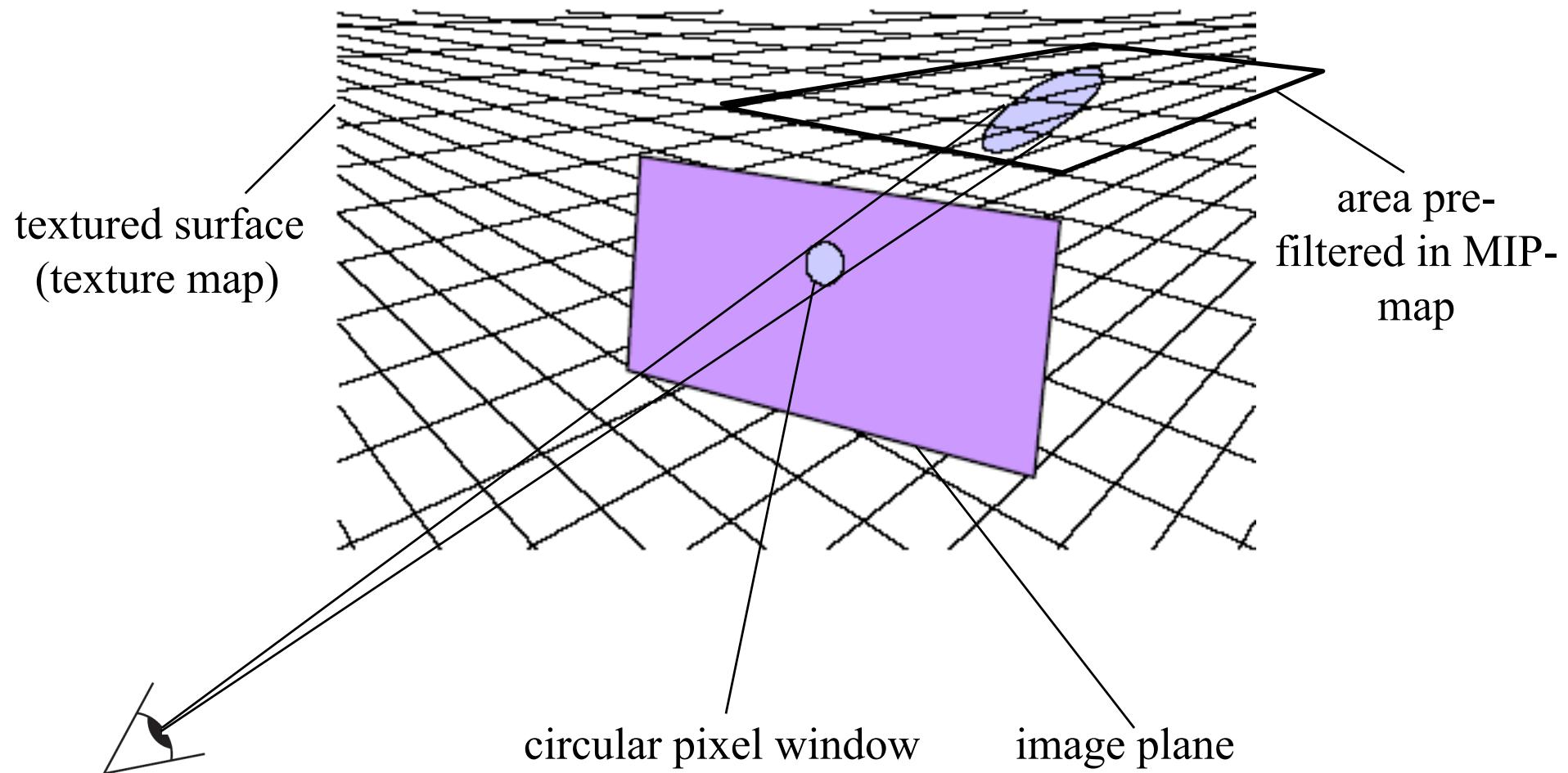


nearest neighbor/ point  
sampling

mipmaps & linear interpolation

# Finding the mip level

---



- Square MIP-map area is a bad approximation

# Finding the MIP level

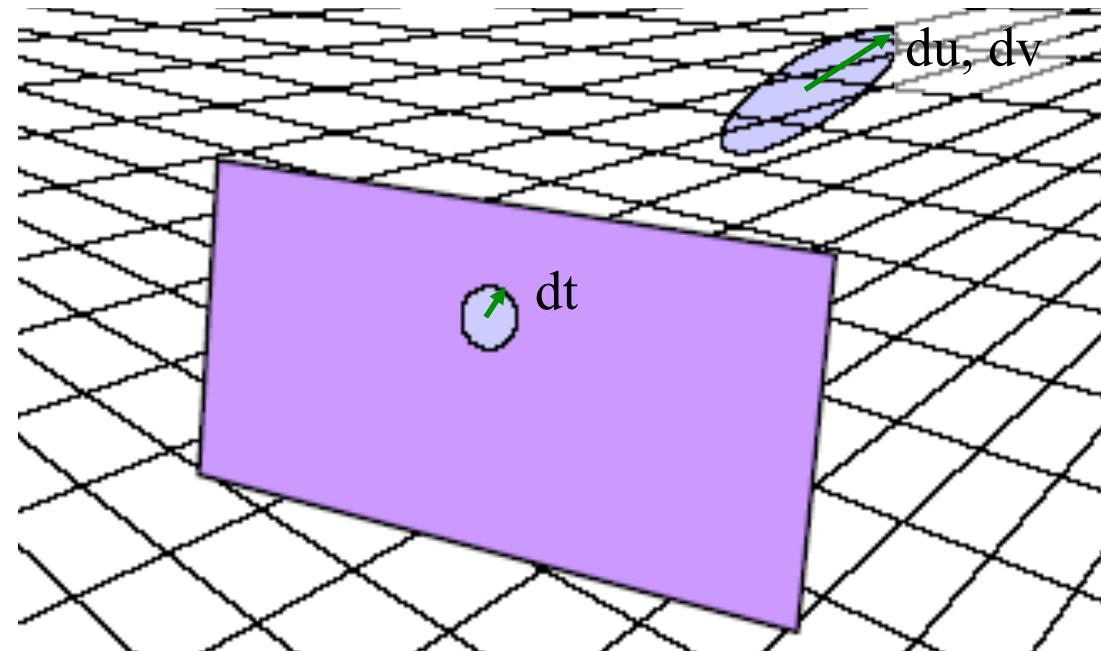
---

How does a screen-space change  $dt$  relates to a texture-space change  $du, dv$ .

=> derivatives, (  $du/dt, dv/dt$  ).

e.g. computed by hardware during rasterization

often: finite difference (pixels are handled by quads)



# MIP Indices

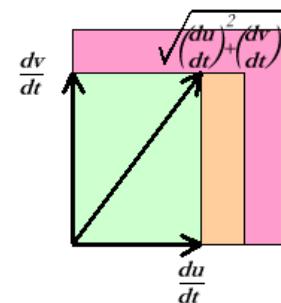
Actually, you have a choice of ways to translate this **derivative value** into a MIP level.

Because we have two derivatives, for u and for v (anisotropy)

This also brings up one of the shortcomings of MIP mapping. MIP mapping assumes that both the u and v components of the texture index are undergoing a uniform scaling, while in fact the terms  $du/dt$  and  $dv/dt$  are relatively independent. Thus, we must make some sort of compromise. Two of the most common approaches are given below:

$$level = \log_2 \left( \sqrt{\left( \frac{du}{dt} \right)^2 + \left( \frac{dv}{dt} \right)^2} \right)$$

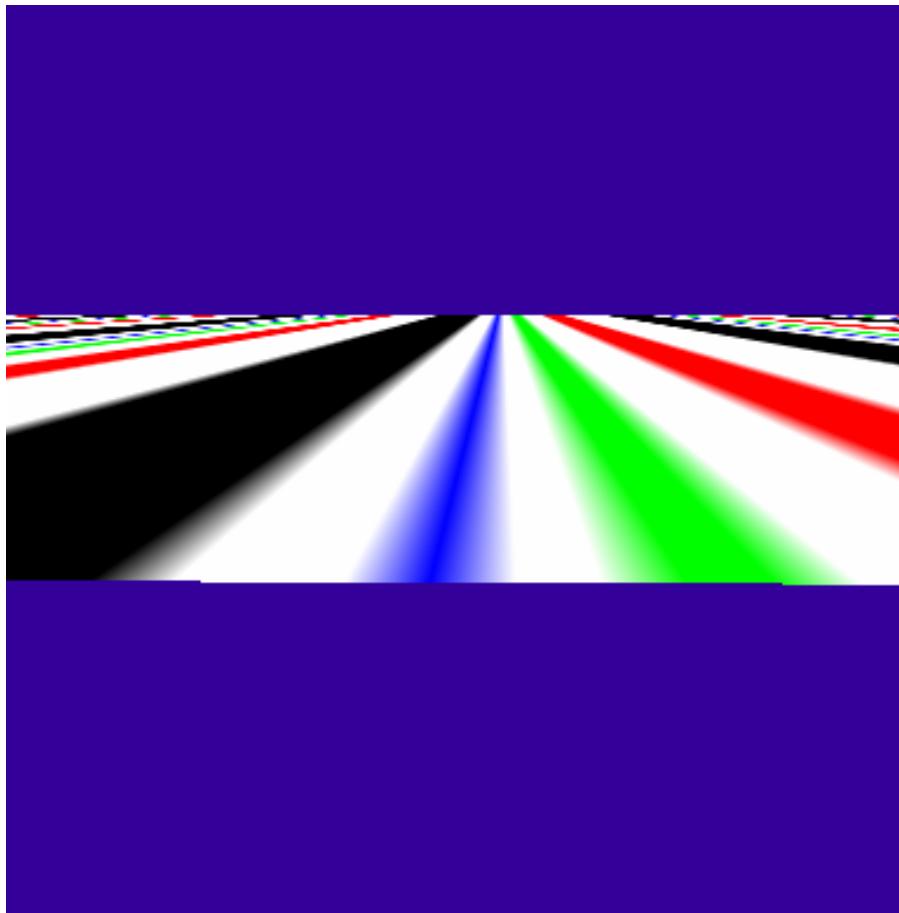
$$level = \log_2 \left( \max \left( \left| \frac{du}{dt} \right|, \left| \frac{dv}{dt} \right| \right) \right)$$



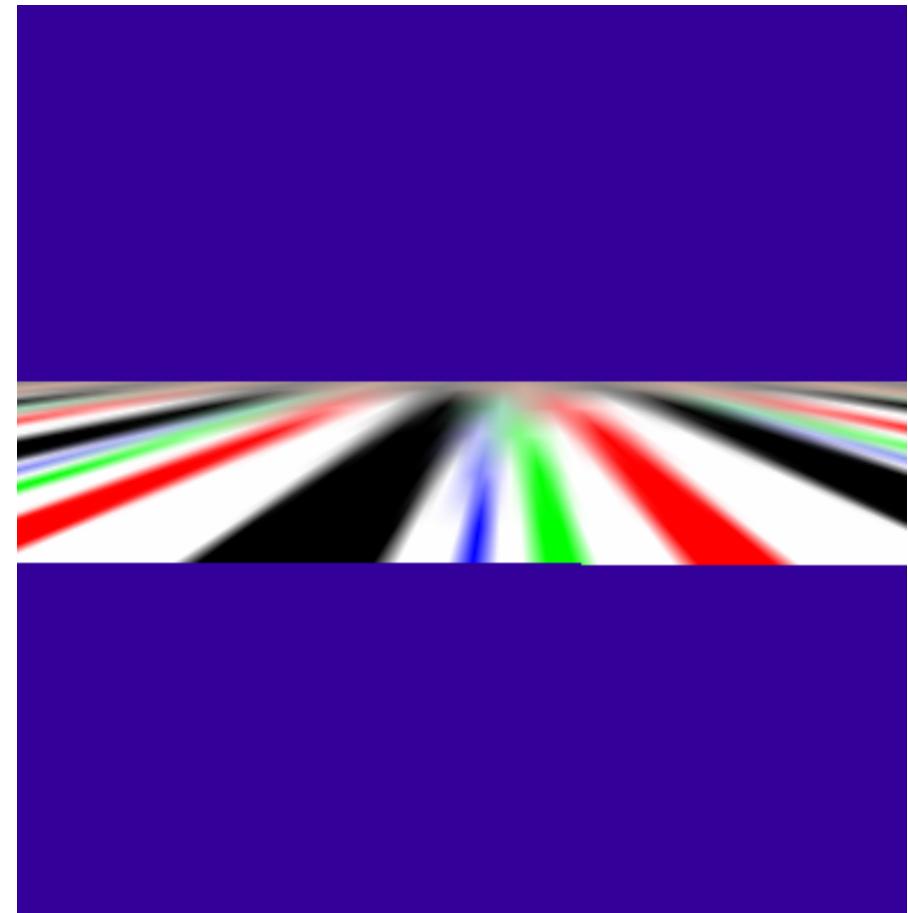
# Anisotropy & MIP-Mapping

---

- What happens when the surface is tilted?



Nearest Neighbor



MIP Mapped (Bi-Linear)

# Elliptical weighted average

---

- Isotropic filter wrt screen space
- Becomes anisotropic in texture space
- e.g. use anisotropic Gaussian
- Called Elliptical Weighted Average (EWA)

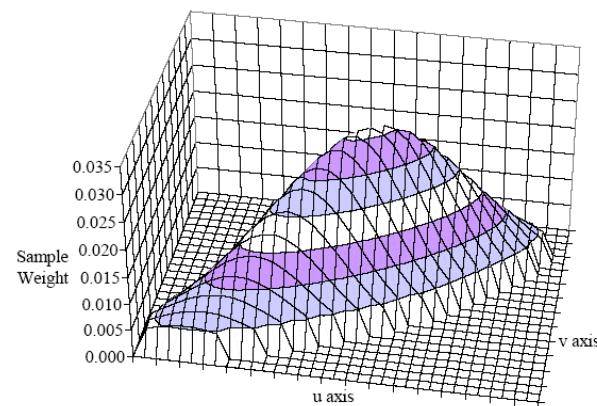
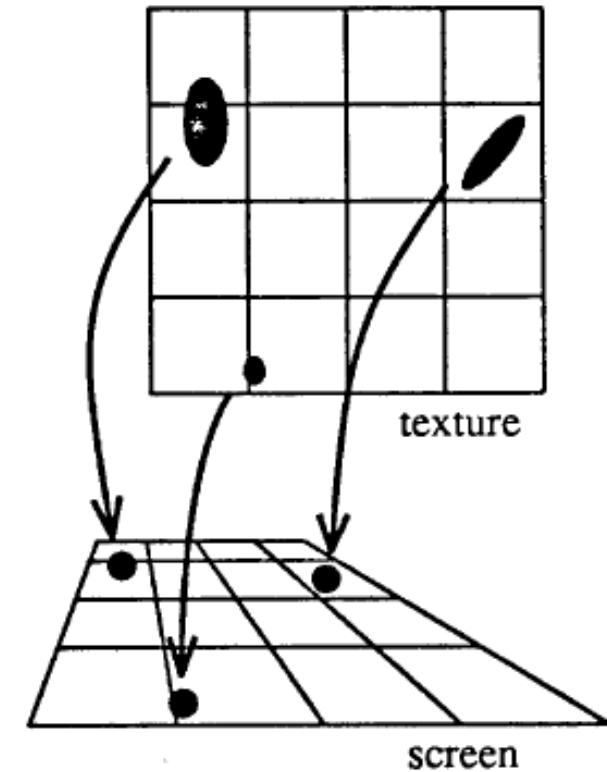


Figure 3: A perspective projection of a Gaussian filter into texture space.

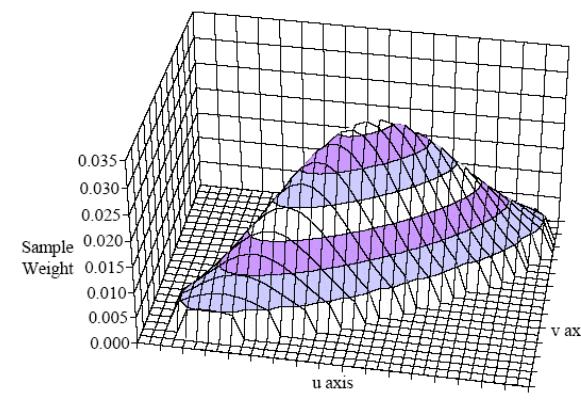
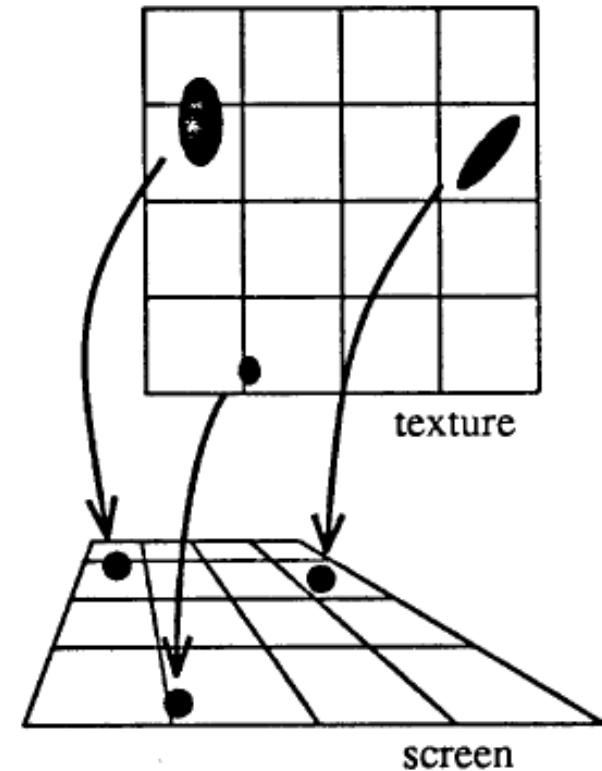


Figure 4: An affine projection of a Gaussian filter into texture space.

# Problem definition

---

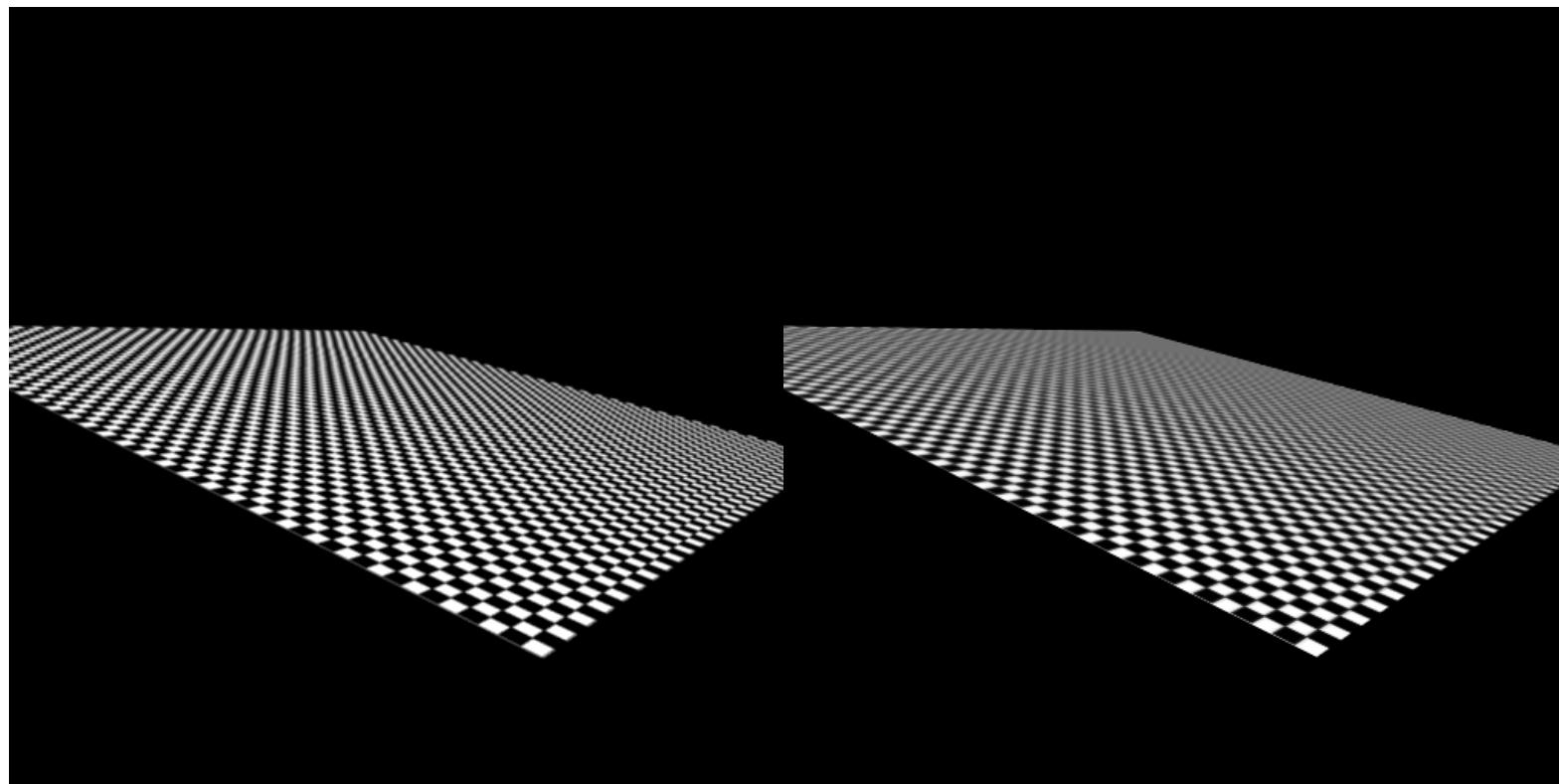
- A texture filtering algorithm should return the convolution of the projected pixel footprint with all the texels covered by this footprint.
- Poor approximation of this convolution sum results in poor texture filtering (aliasing or blurring).
- EWA directly computes this convolution sum by iterating over the texels inside the ellipse box.



# Image Quality Comparison

---

- Trilinear mipmapping



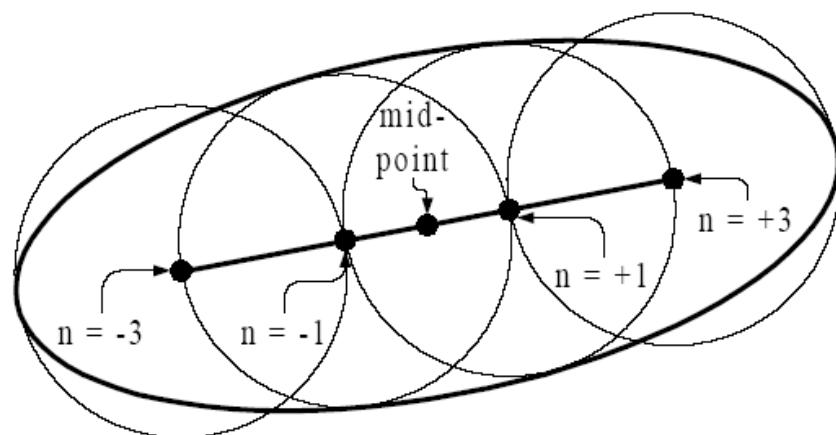
EWA

trilinear mipmapping

# Approximation of anisotropic

---

- Feline: Fast Elliptical Lines for Anisotropic Texture Mapping Joel McCormack, Ronald Perry, Keith I. Farkas, and Norman P. Jouppi SIGGRAPH 1999
- Andreas Schilling, Gunter Knittel & Wolfgang Strasser. Texram: A Smart Memory for Texturing. IEEE Computer Graphics and Applications, 16(3): 32-41, May 1996.
- Approximate Anisotropic Gaussian by a set of isotropic “probes”



# FELINE results

---

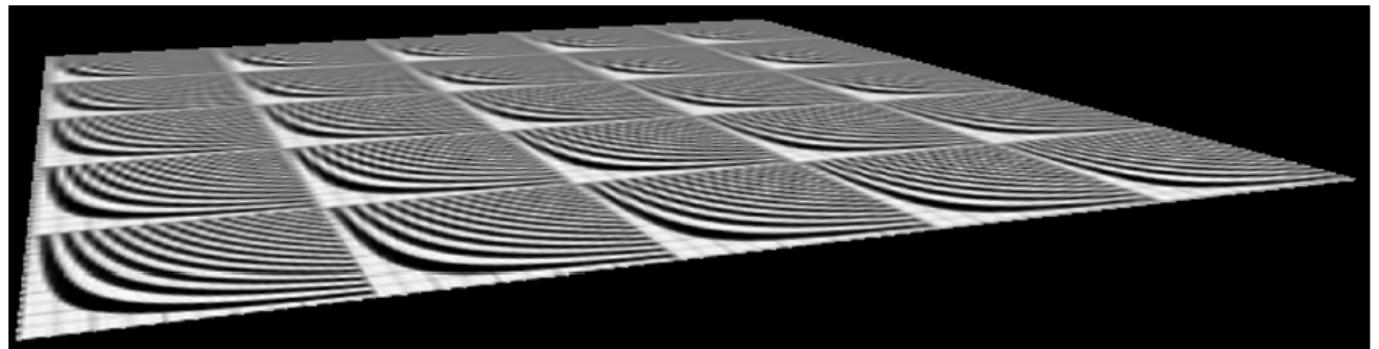


Figure 10: Trilinear paints curved lines with blurring.

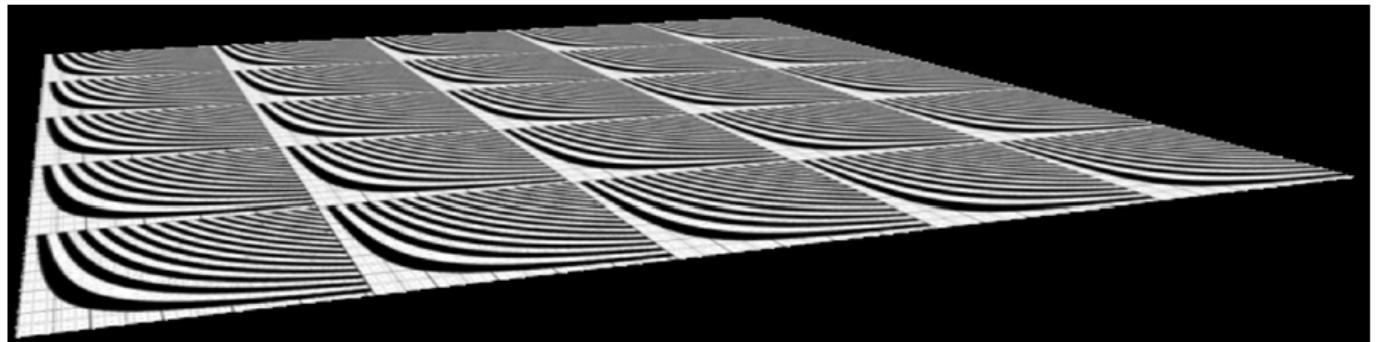


Figure 13: “High-quality” Simple Feline paints curved lines with few artifacts.

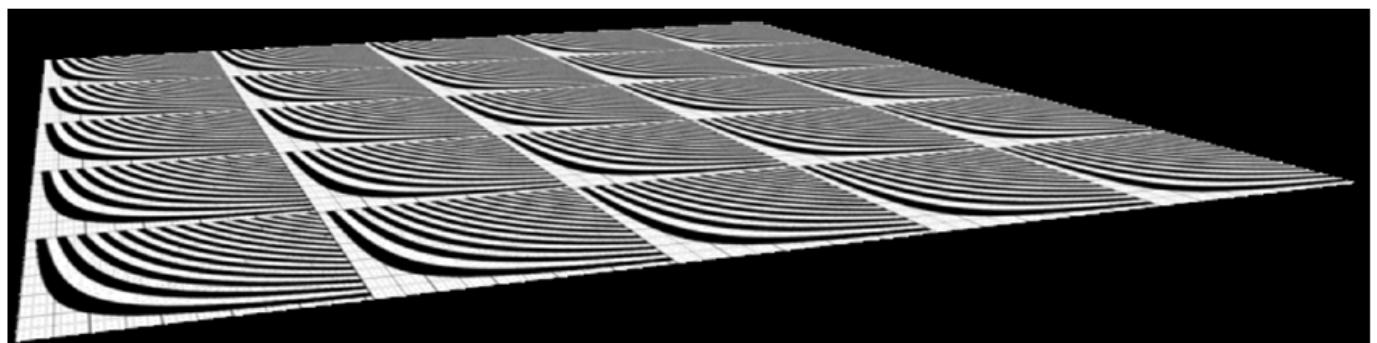


Figure 14: Mip-mapped EWA paints curved lines with few artifacts.

# Questions?

---