

08 Texture Antialiasing

Steve Marschner
CS5625 Spring 2016

Overview

Basic sampling problem

- Texture mapping defines a signal in image space
- That signal needs to be filtered: convolved with a filter
- Approximating this drives all the basic algorithms

Antialiasing nonlinear shading

- Basic sampling suffices only if pixel and texture are linearly related
- Normal mapping is the most important nonlinearity

Texture mapping from 0 to infinity

When you go close...



Texture mapping from 0 to infinity

When you go far...

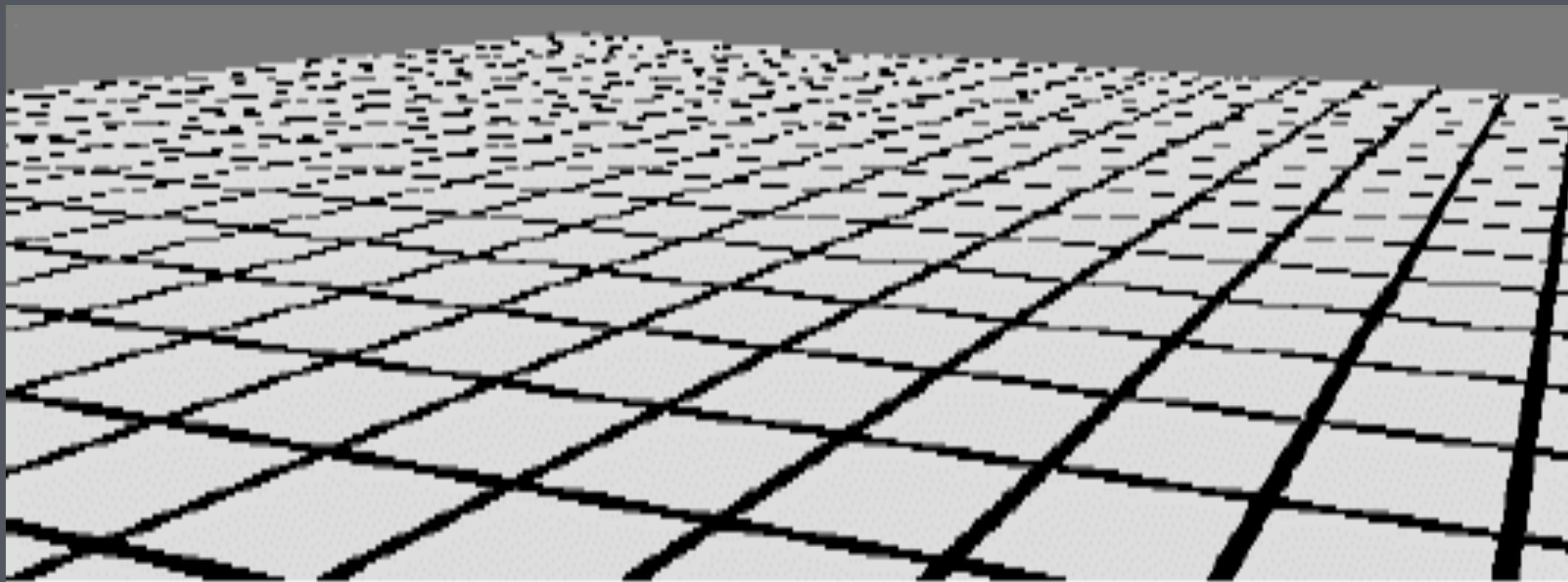


Solution: pixel filtering

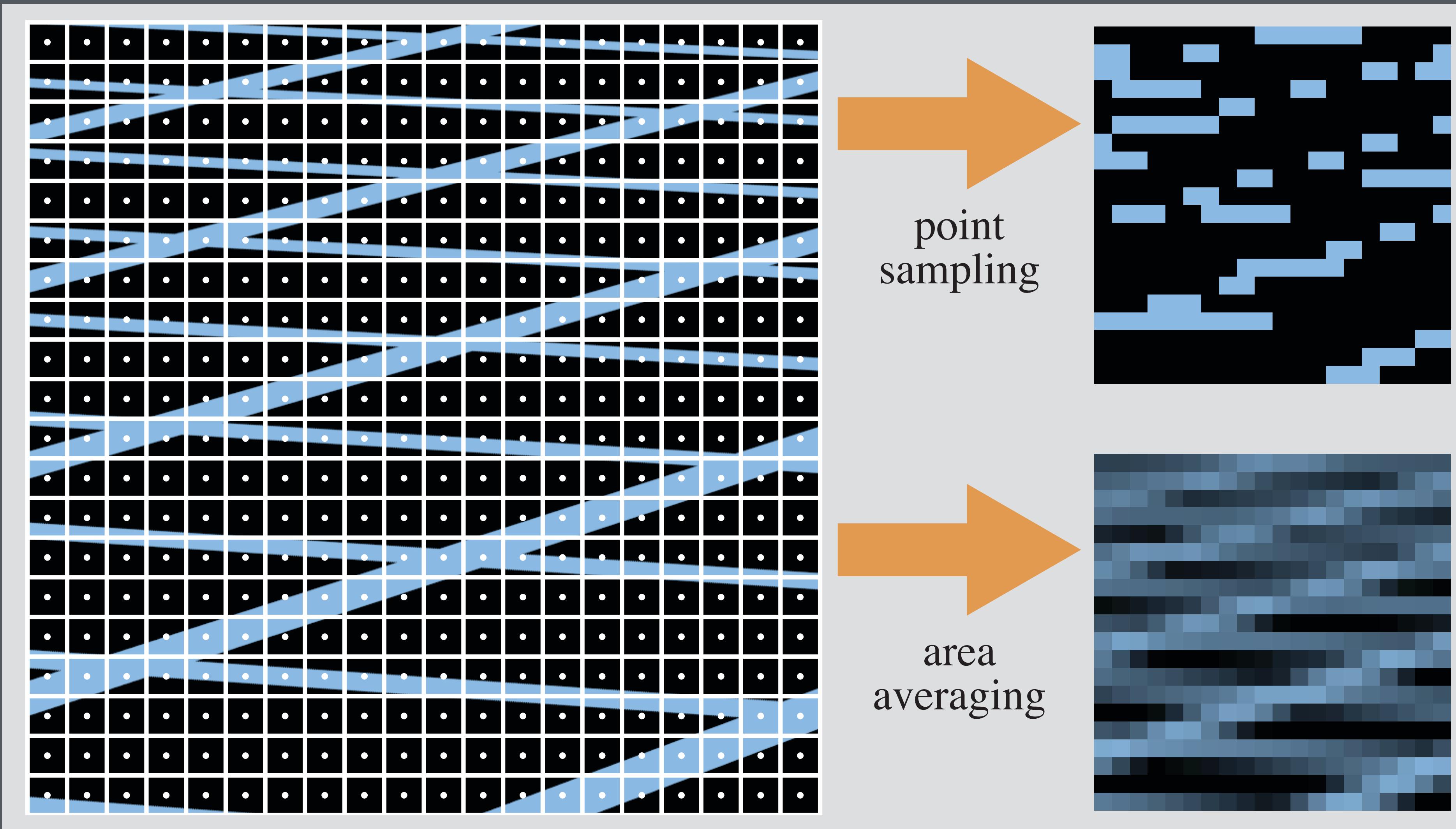
Problem: Perspective produces very high image frequencies

Solution

- Would like to render textures with one (few) samples/pixel
- Need to filter first!



Solution: pixel filtering



Pixel filtering in texture space

Sampling is happening in image space

- therefore the sampling filter is defined in image space
- sample is a weighted average over a pixel-sized area
- uniform, predictable, friendly problem!

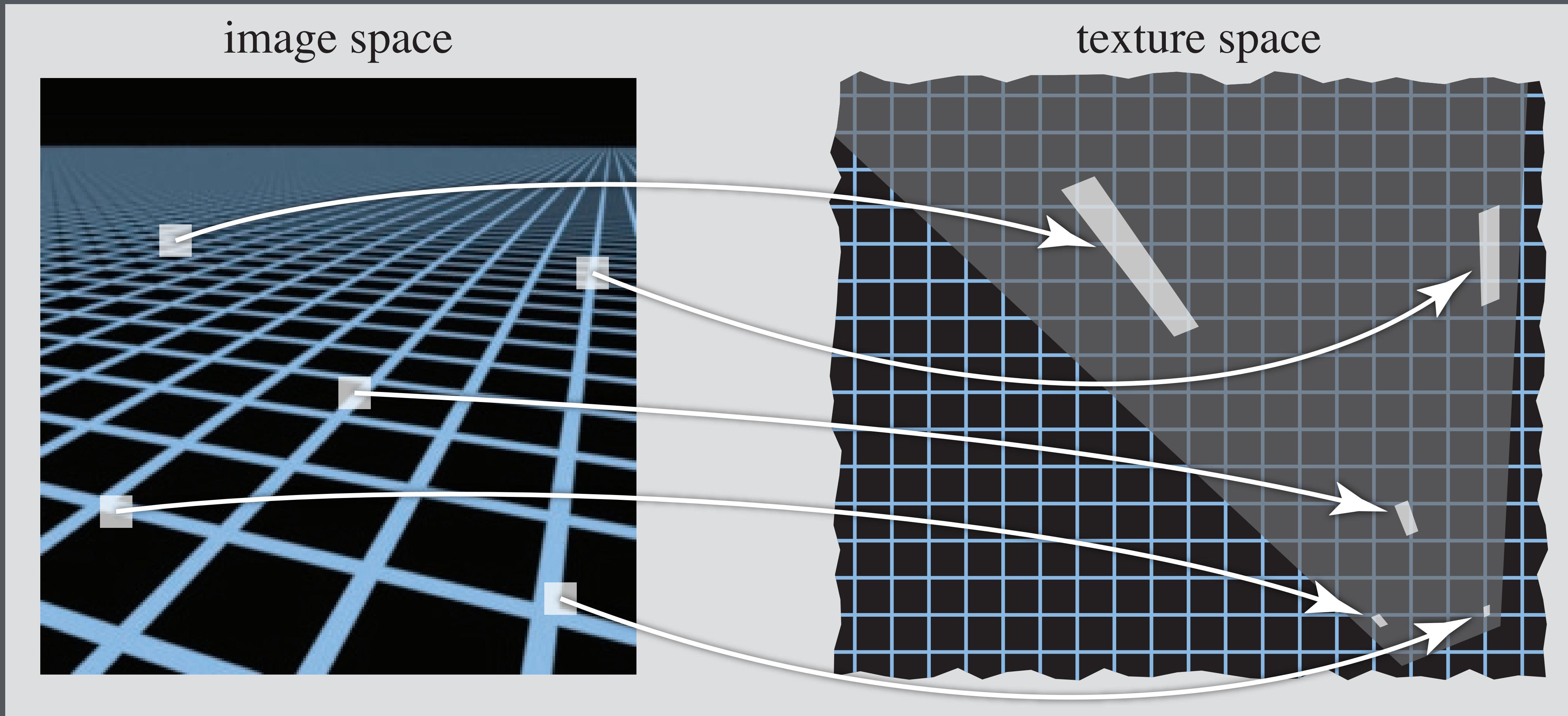
Signal is defined in texture space

- mapping between image and texture is nonuniform
- each sample is a weighted average over a different sized and shaped area
- irregular, unpredictable, unfriendly!

This is a change of variable

- integrate over texture coordinates rather than image coordinates

Pixel footprints



How does area map over distance?

At optimal viewing distance:

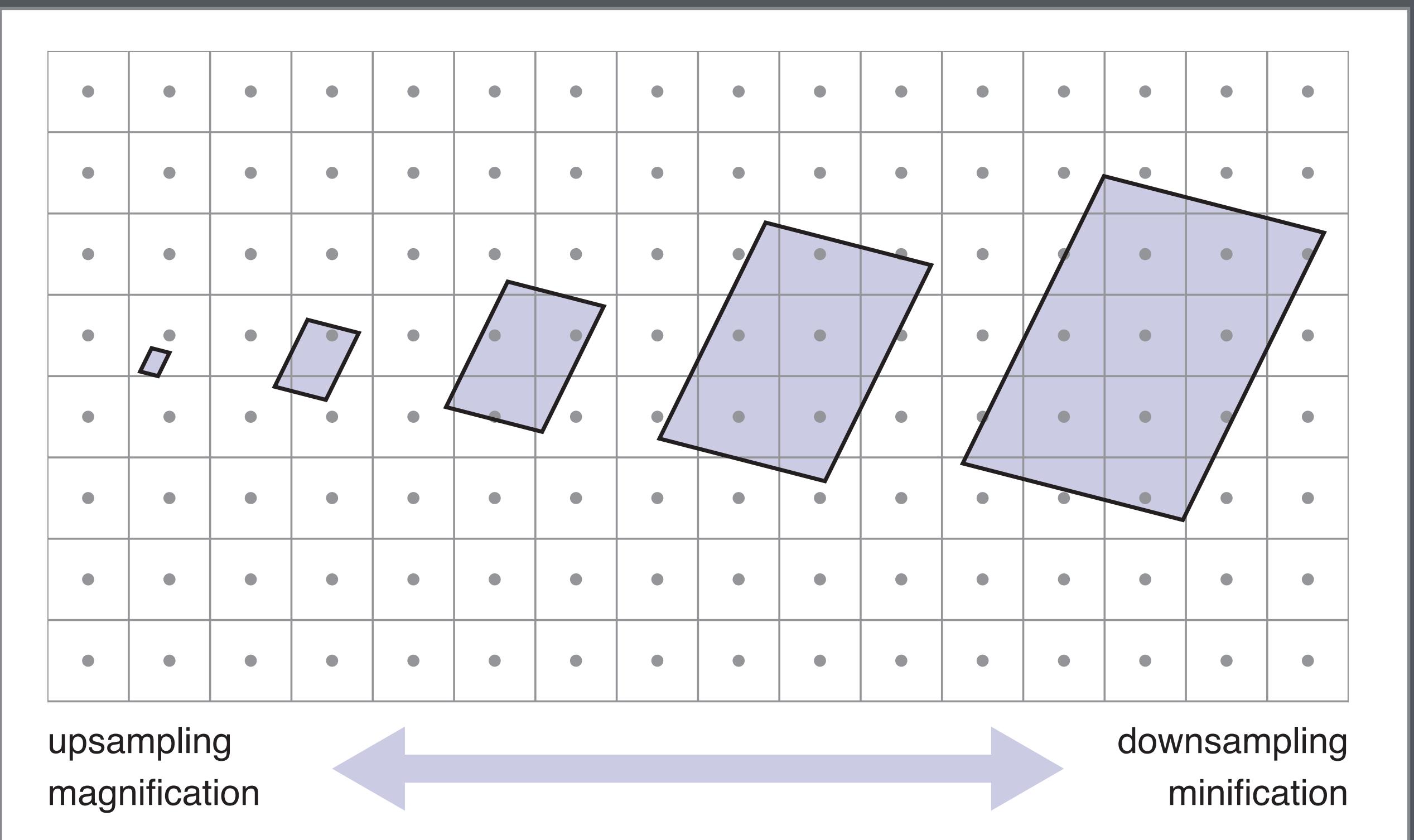
- One-to-one mapping between pixel area and texel area

When closer

- Each pixel is a small part of the texel
- magnification

When farther

- Each pixel could include many texels
- “minification”



How to get a handle on pixel footprint

We have a nonlinear mapping to deal with

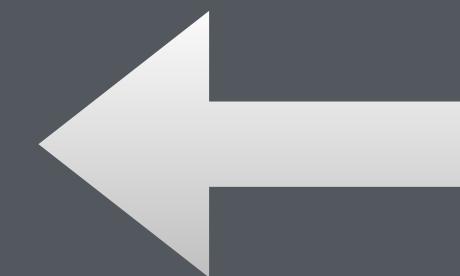
- image position as a function of texture coordinates: $\mathbb{R}^2 \rightarrow \mathbb{R}^2 : \mathbf{u} \mapsto \mathbf{x}(\mathbf{u})$
- but that is too hard

Instead use a local linear approximation

- hinges on the derivative of $\mathbf{u} = (u, v)$ wrt. $\mathbf{x} = (x, y)$

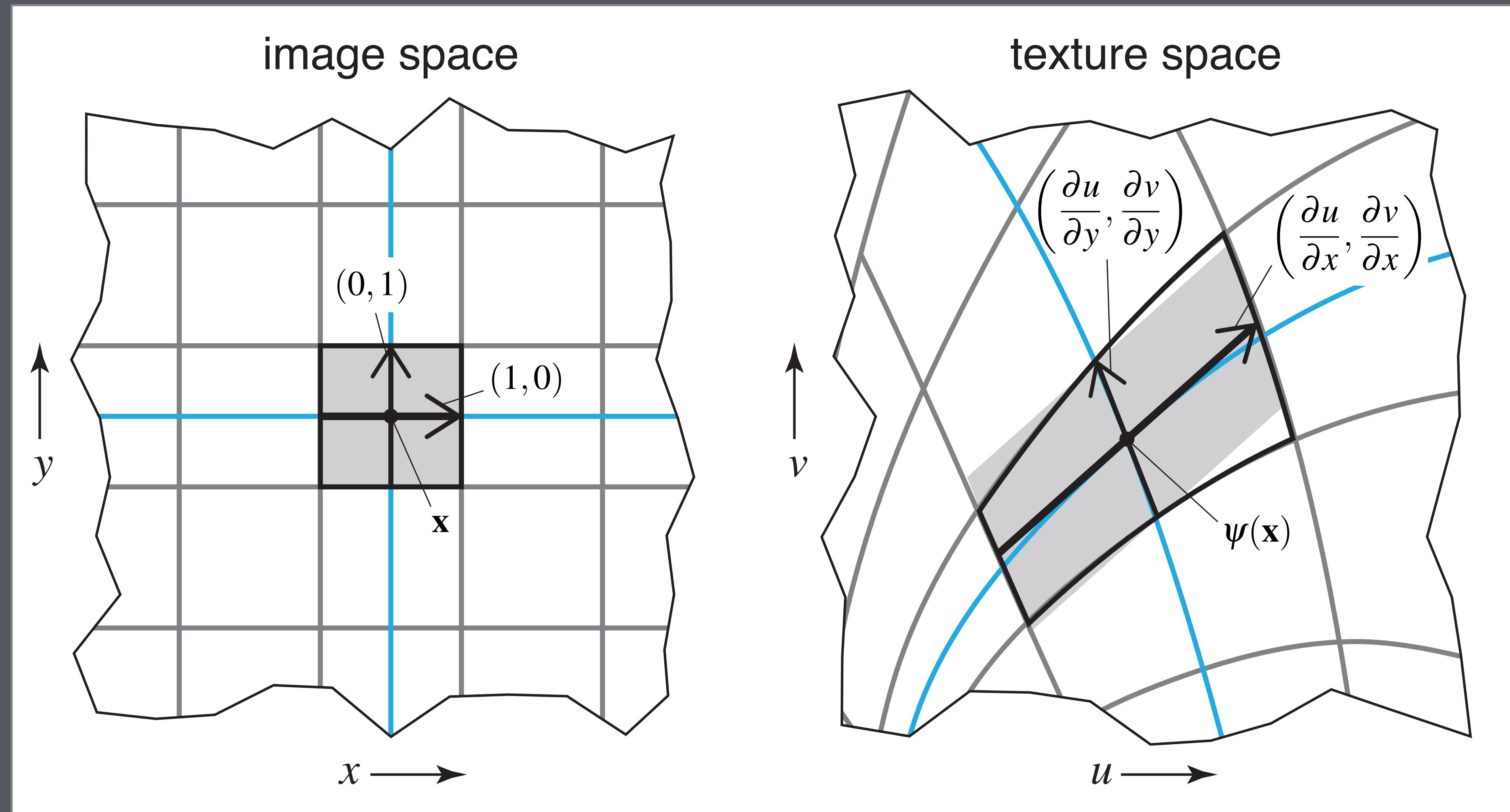
$$\mathbf{u}(\mathbf{x} + \Delta\mathbf{x}) \approx \mathbf{u}(\mathbf{x}) + \frac{\partial \mathbf{u}}{\partial \mathbf{x}} \Delta\mathbf{x}$$

$$\frac{\partial \mathbf{u}}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial u}{\partial x} & \frac{\partial u}{\partial y} \\ \frac{\partial v}{\partial x} & \frac{\partial v}{\partial y} \end{bmatrix}$$



Matrix derivative,
or Jacobian

Sizing up the situation with the Jacobian



How to tell minification from magnification

Difference is the size of the derivative

- but what is “size”?
- area: determinant of Jacobian: $\left| \frac{\partial \mathbf{u}}{\partial \mathbf{x}} \right|$
- max-stretch: 2-norm of Jacobian (requires a singular-value computation)
- Frobenius norm of matrix (RMS of 4 entries, easy to compute)
- max dimension of bounding box of quadrilateral footprint: max-abs of 4 entries (conservative)

Take your pick; magnification is when size is more than about 1

Solutions for Minification

For magnification, use a good image interpolation method

- bilinear (usual) or bicubic filter (fancier, smoother) are good picks
- nearest neighbor (box filter) will give you Minecraft-style blockies

For minification, use a good sampling filter to average

- box (simple, though not usually easier)
- gaussian (good choice)

Challenge is to approximate the integral efficiently!

- mipmaps
- multi-sample anisotropic filtering (based on mipmap)

Mipmap image pyramid

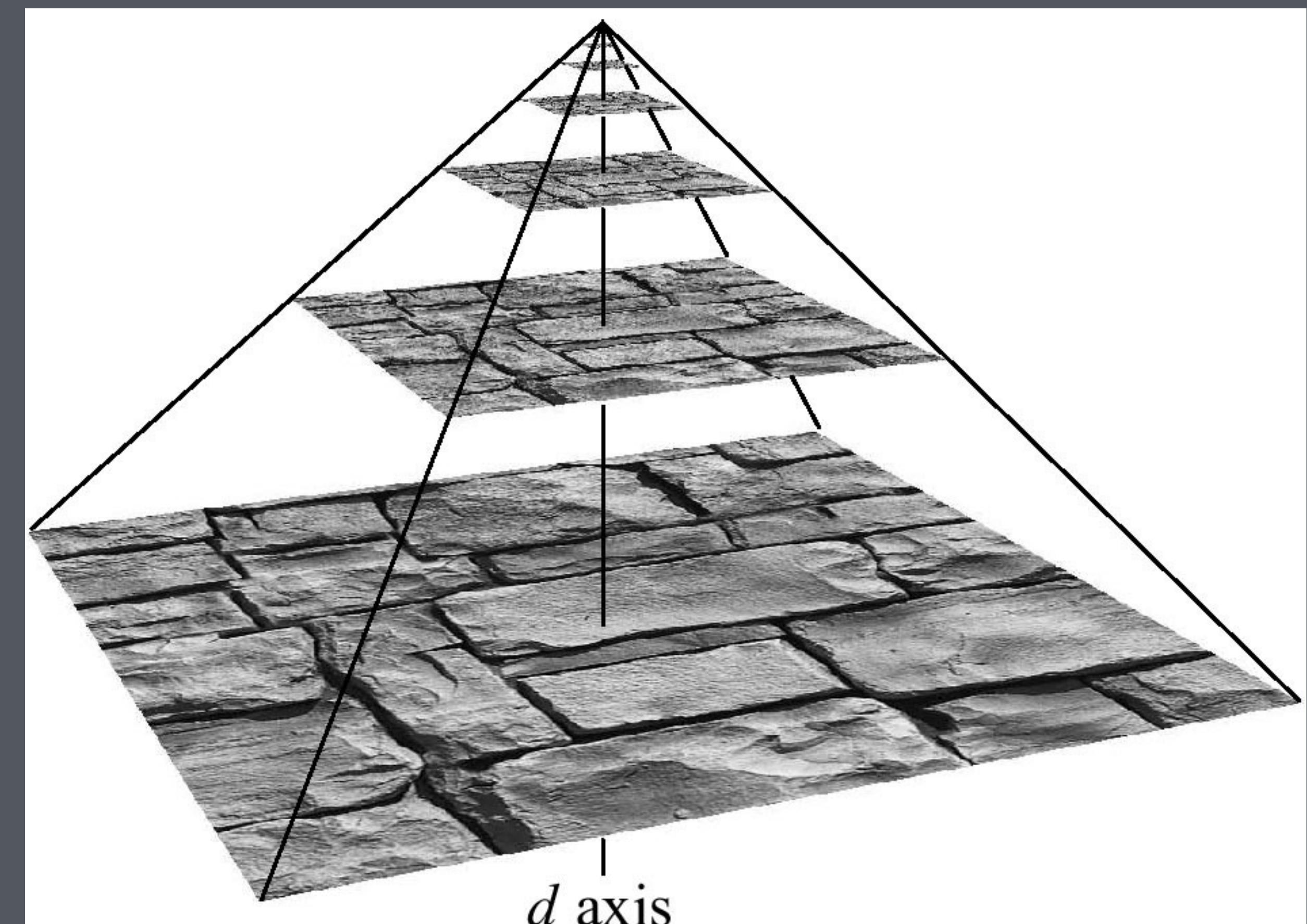
MIP Maps

- Multum in Parvo: Much in little, many in small places
- Proposed by Lance Williams

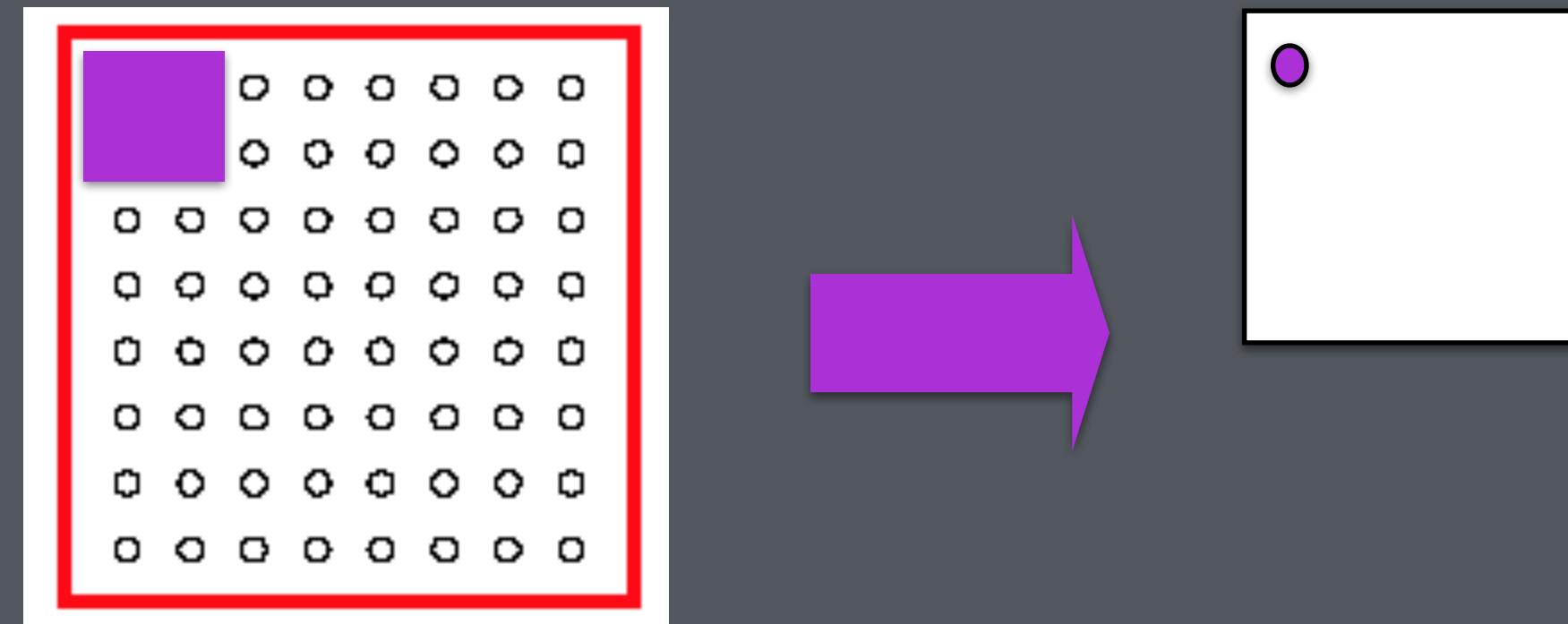
**Stores pre-filtered versions
of texture**

Supports very fast lookup

- but only of circular filters
at certain scales



Filtering by Averaging



Each pixel in a level corresponds to 4 pixels in lower level

- Average
- Gaussian filtering

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:
 - y derivative:

$$\begin{array}{cc} \frac{\partial u}{\partial x} & \frac{\partial v}{\partial x} \\ \hline \frac{\partial u}{\partial y} & \frac{\partial v}{\partial y} \end{array}$$

Given derivatives: what is level?

Derivatives

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

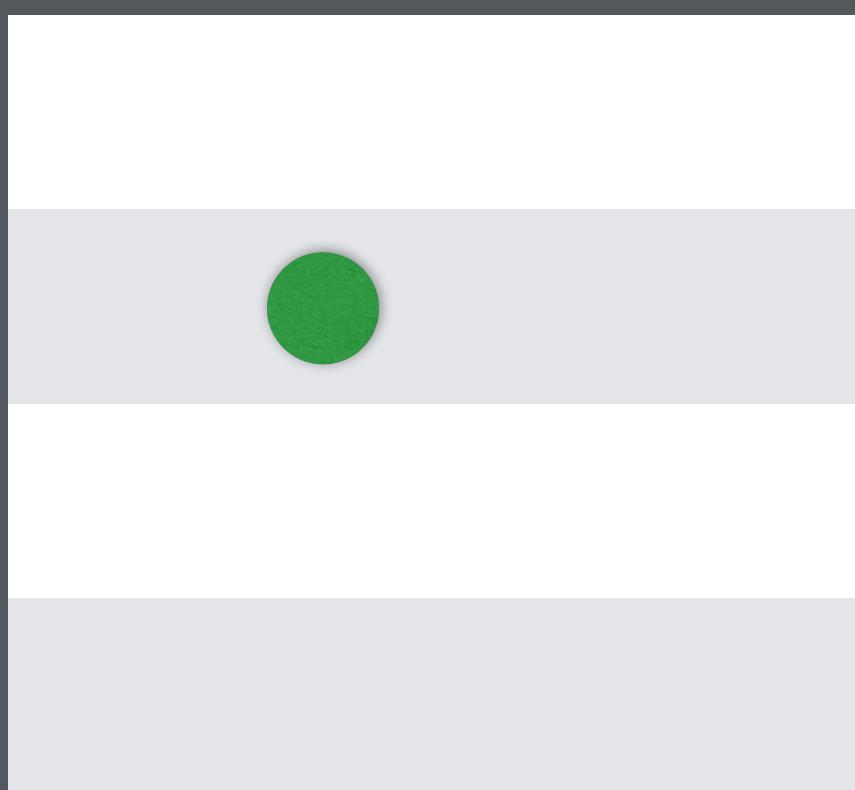
$$level = \log[\max\left(\frac{du}{dx}, \frac{dv}{dx}, \frac{du}{dy}, \frac{dv}{dy}\right)]$$

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

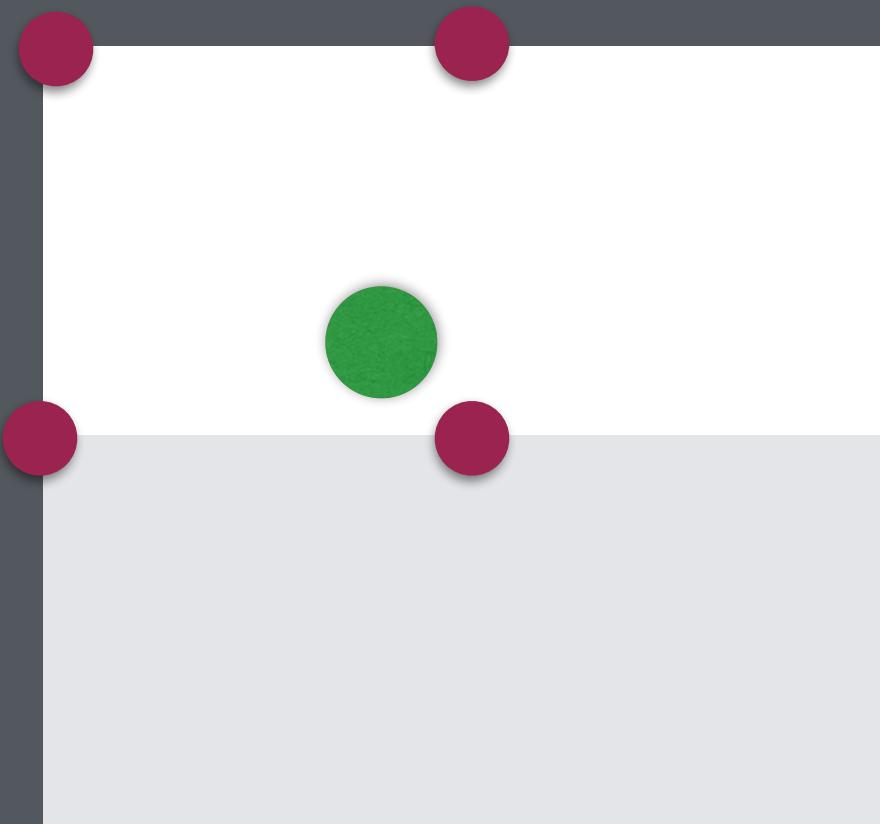
Using the MIP Map

In level, find texel and

- Return the texture value: point sampling (but still better)!
- Bilinear interpolation
- Trilinear interpolation



Level i

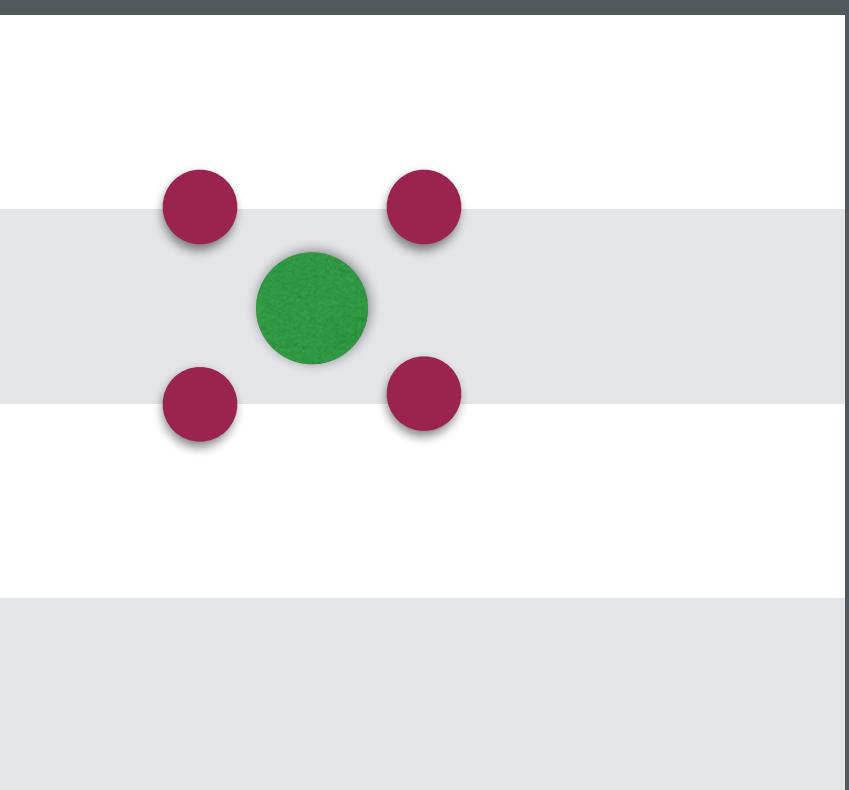


Level i+1

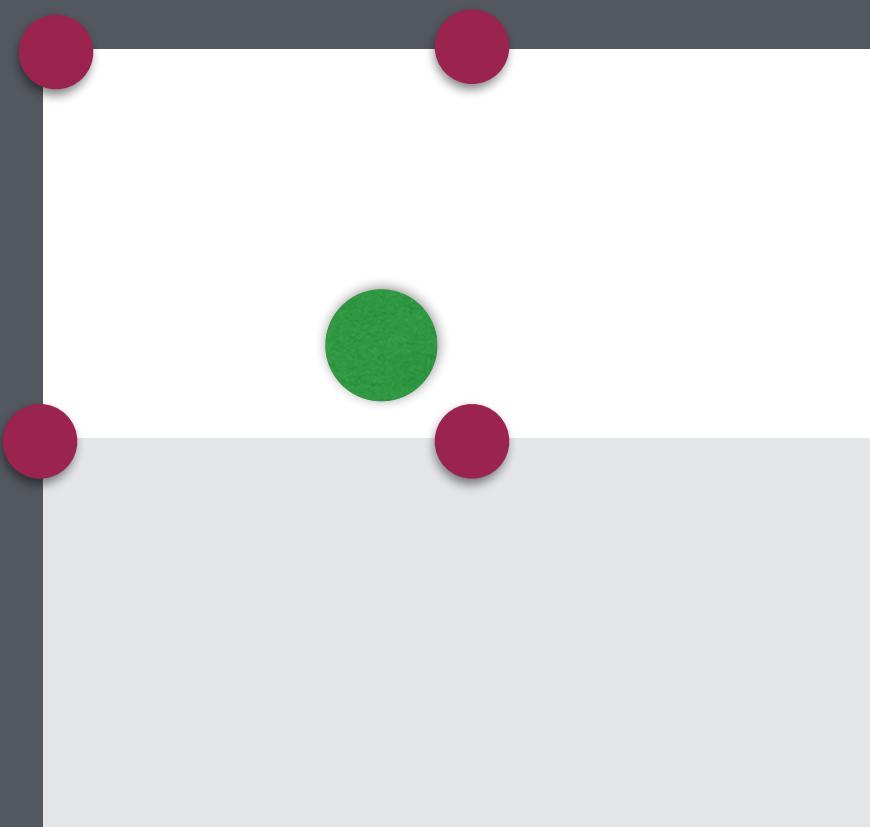
Using the MIP Map

In level, find texel and

- Return the texture value: point sampling (but still better)!
- Bilinear interpolation
- Trilinear interpolation



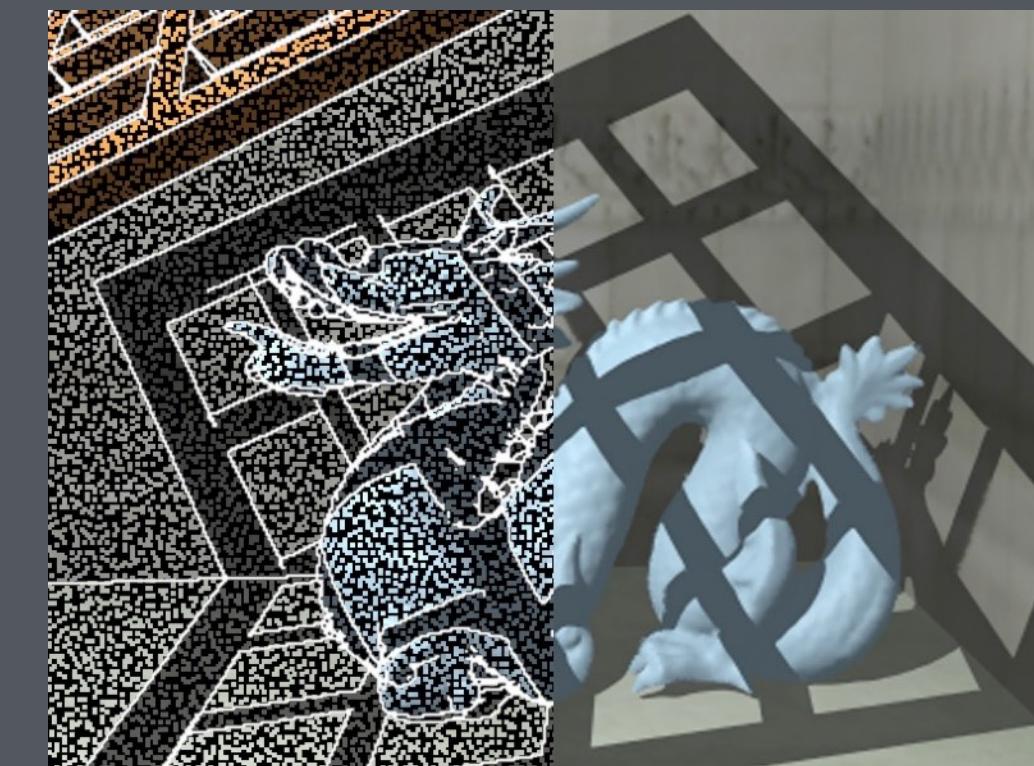
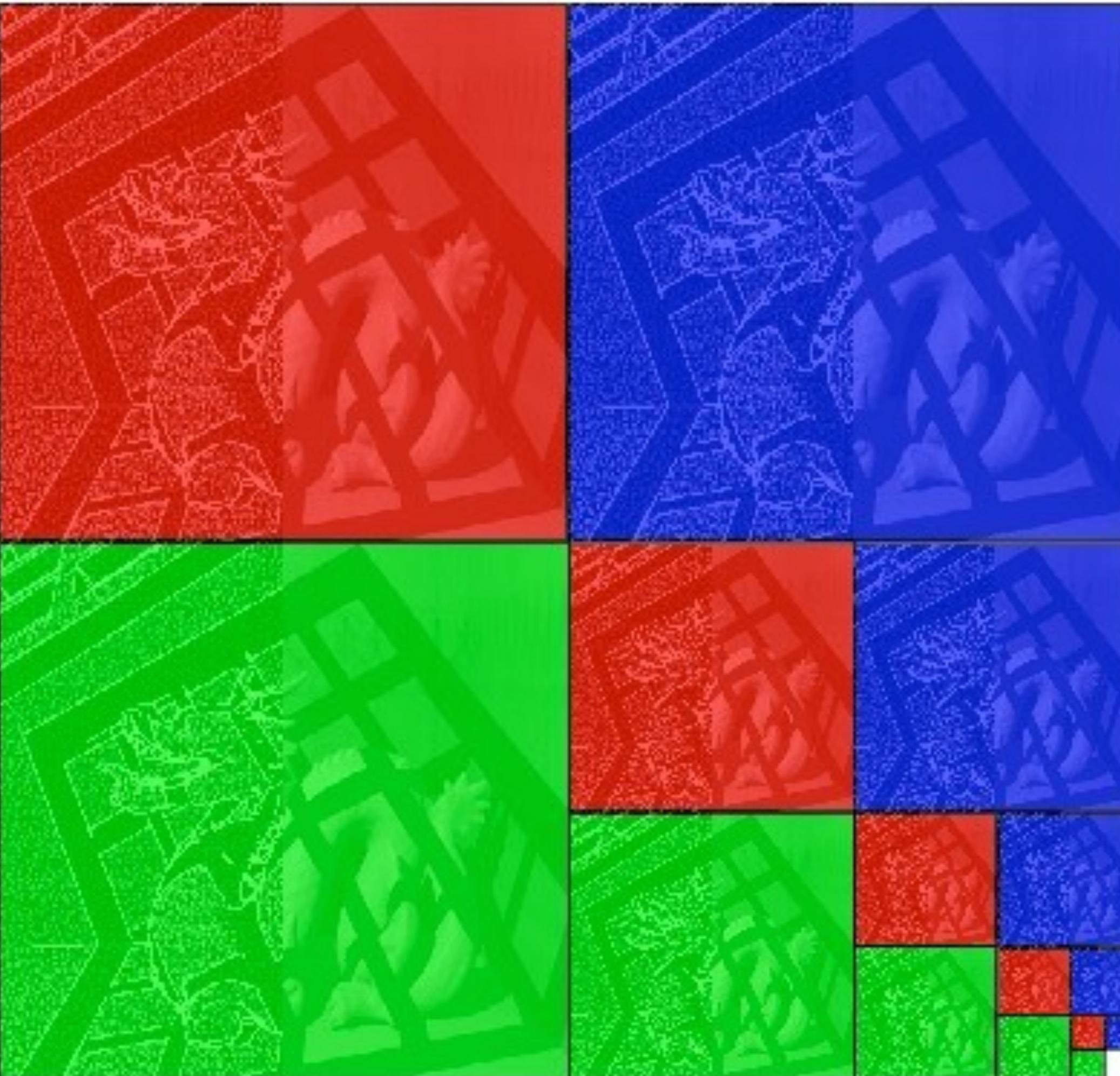
Level i



Level i+1

Memory Usage

What happens to size of texture?



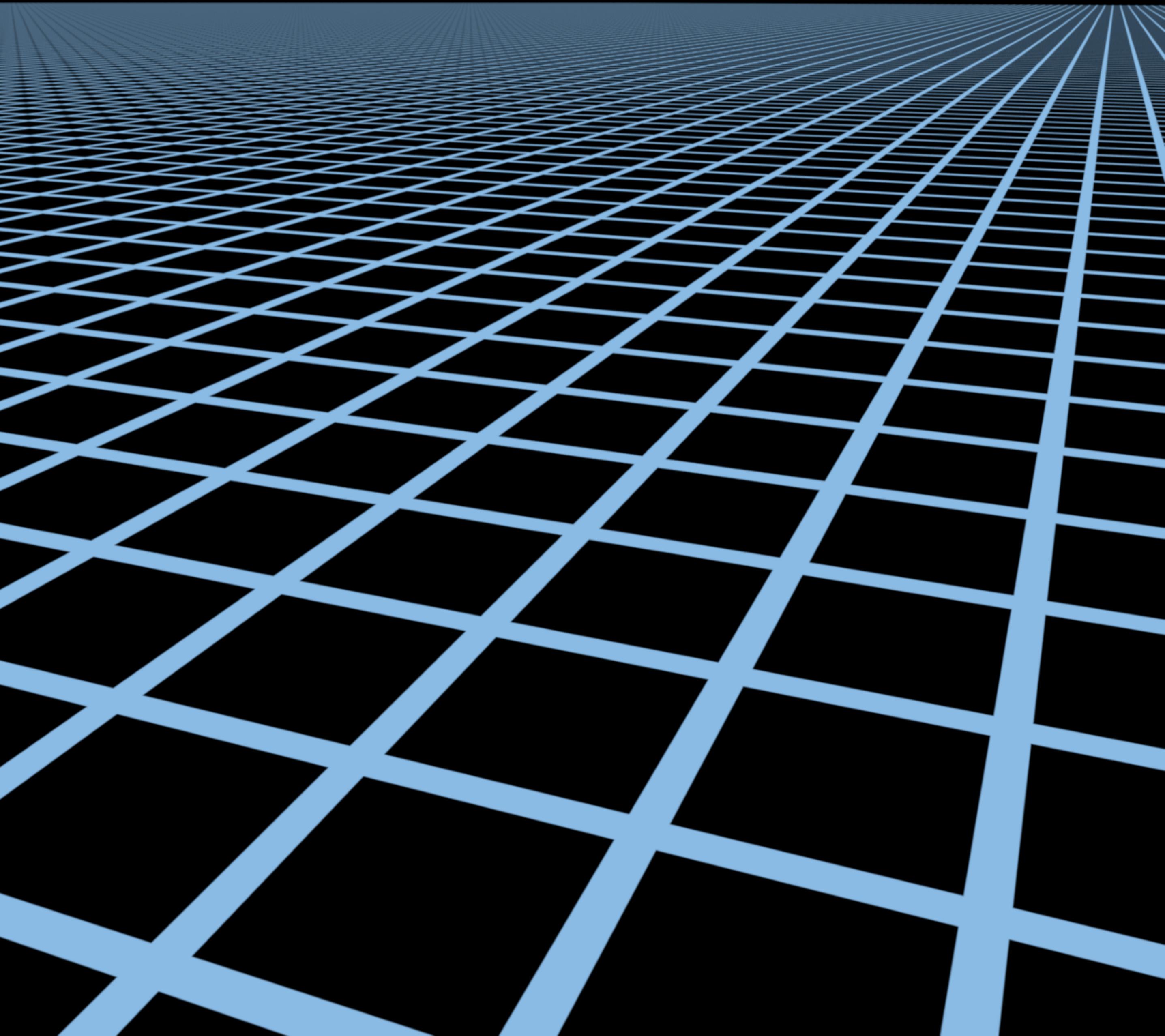
MIPMAP

Multi-resolution image pyramid

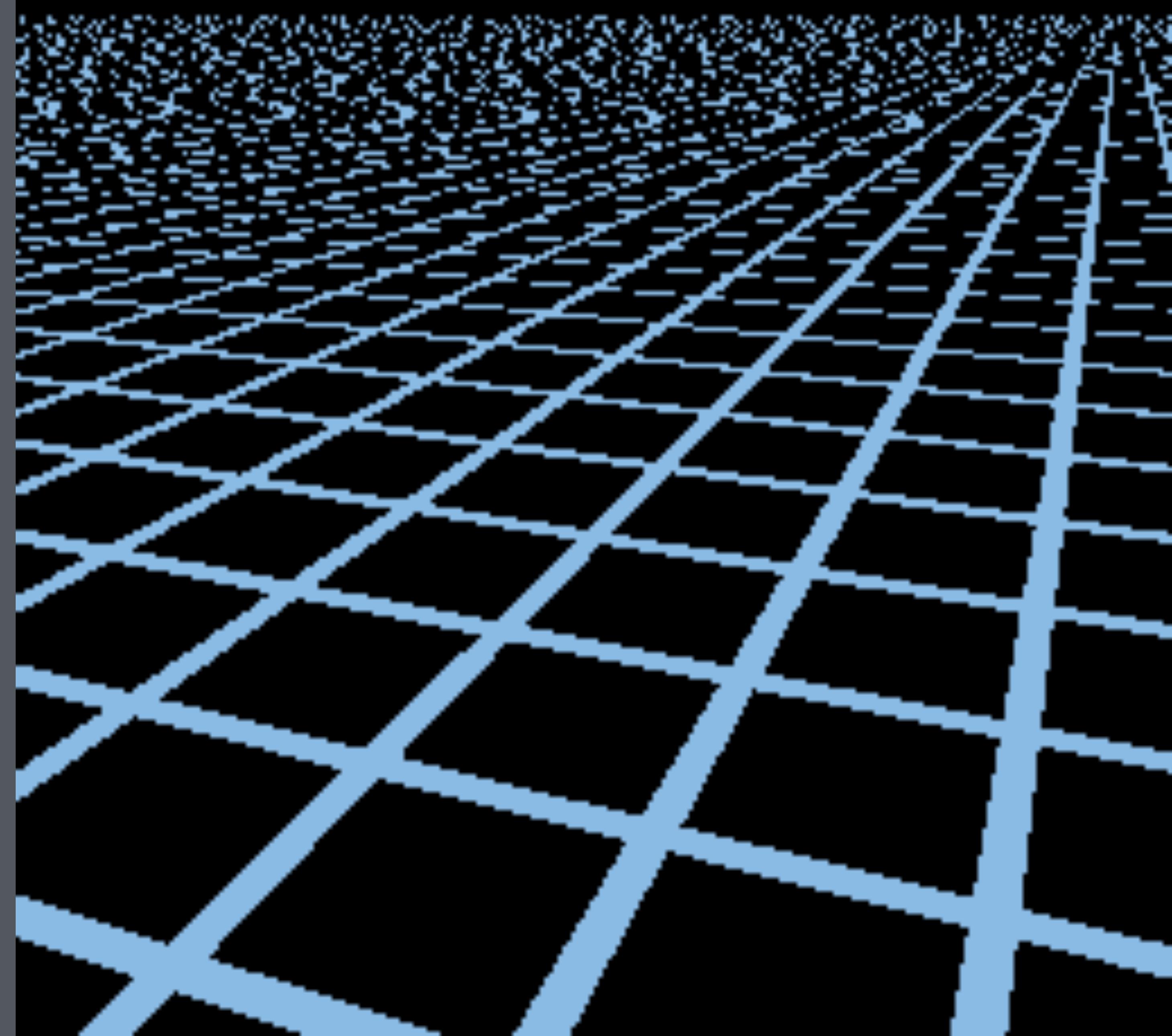
- Pre-sampled computation of MIPMAP
- 1/3 more memory

Bilinear or Trilinear interpolation

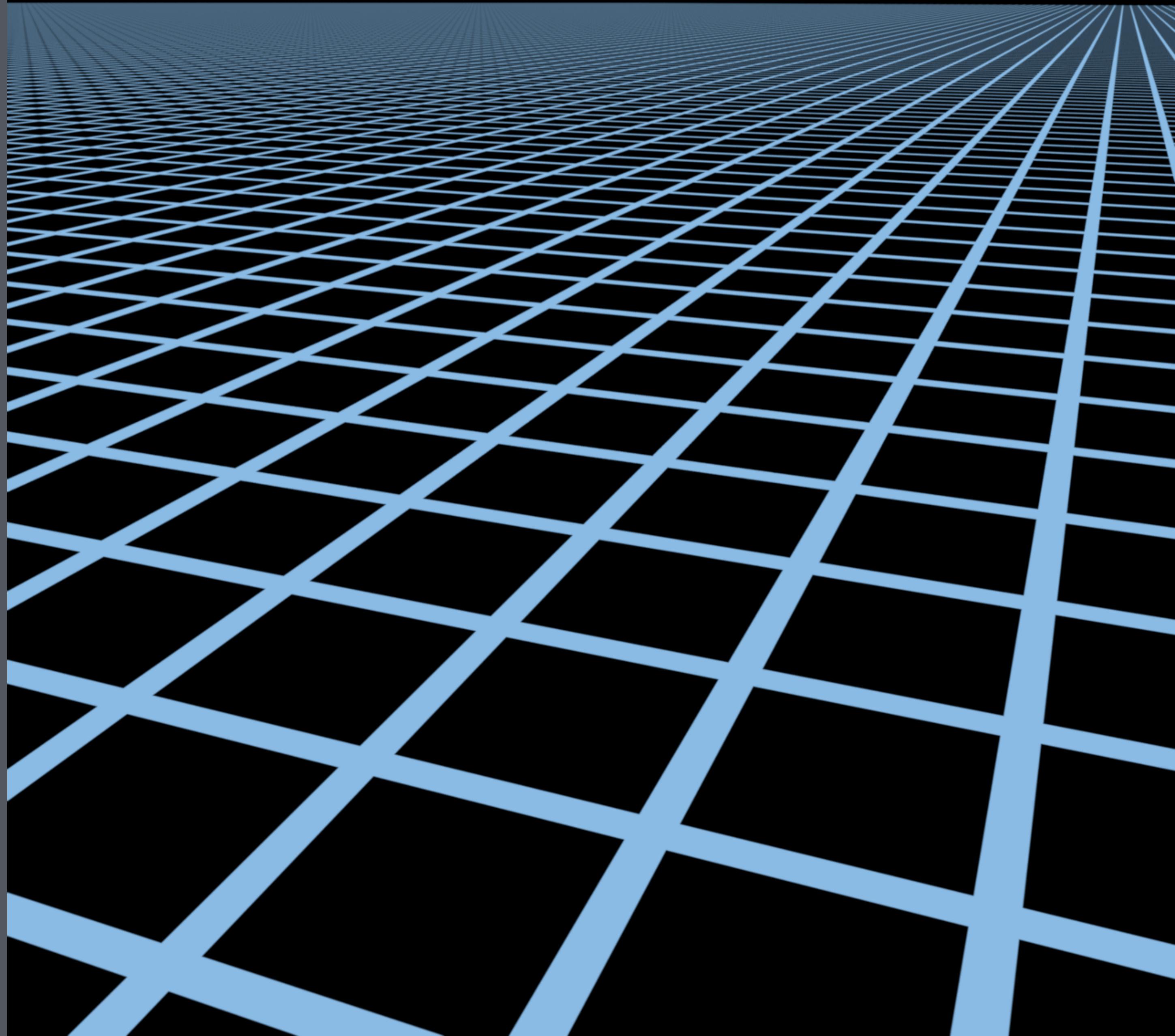
Point sampling



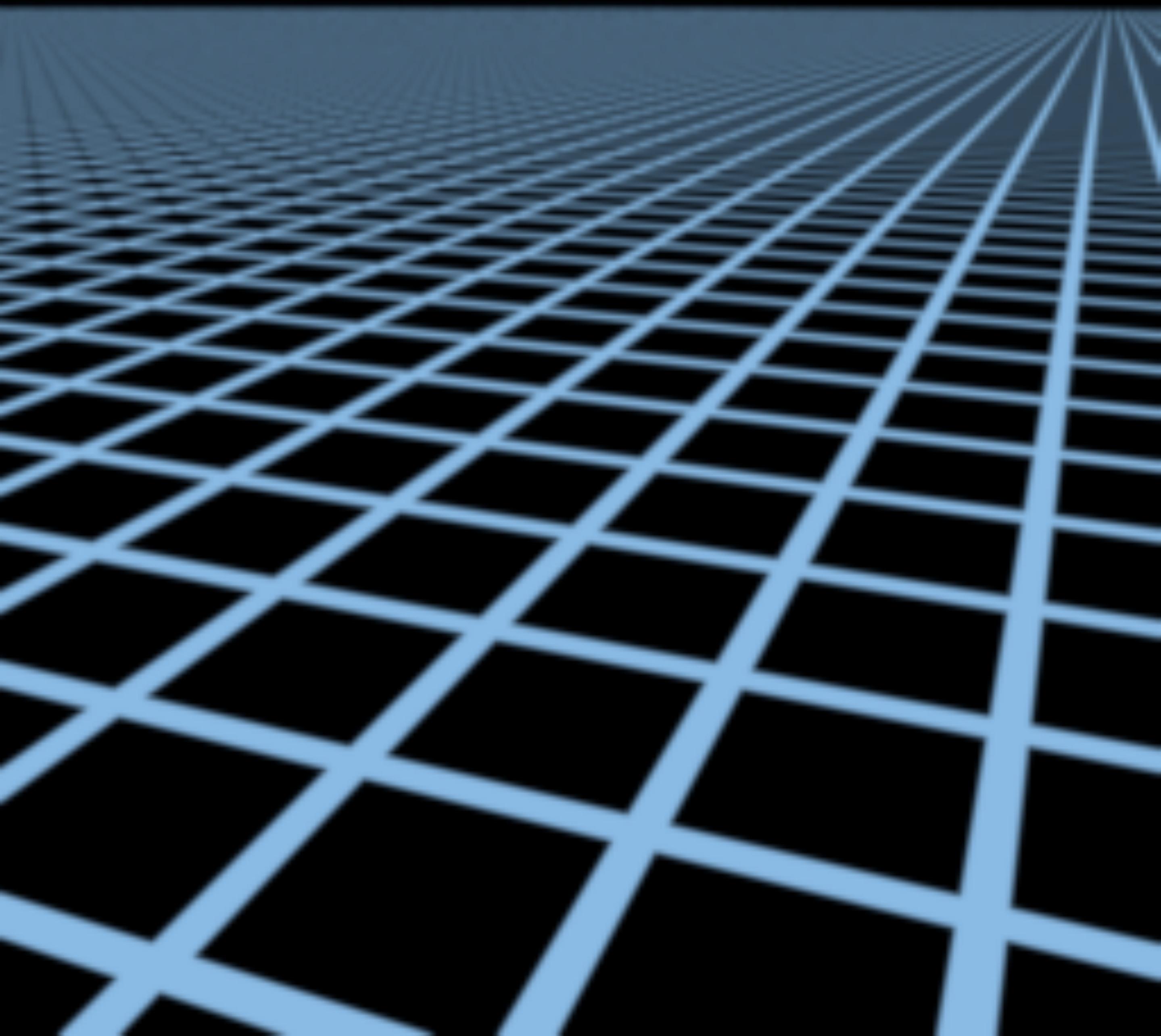
Point sampling



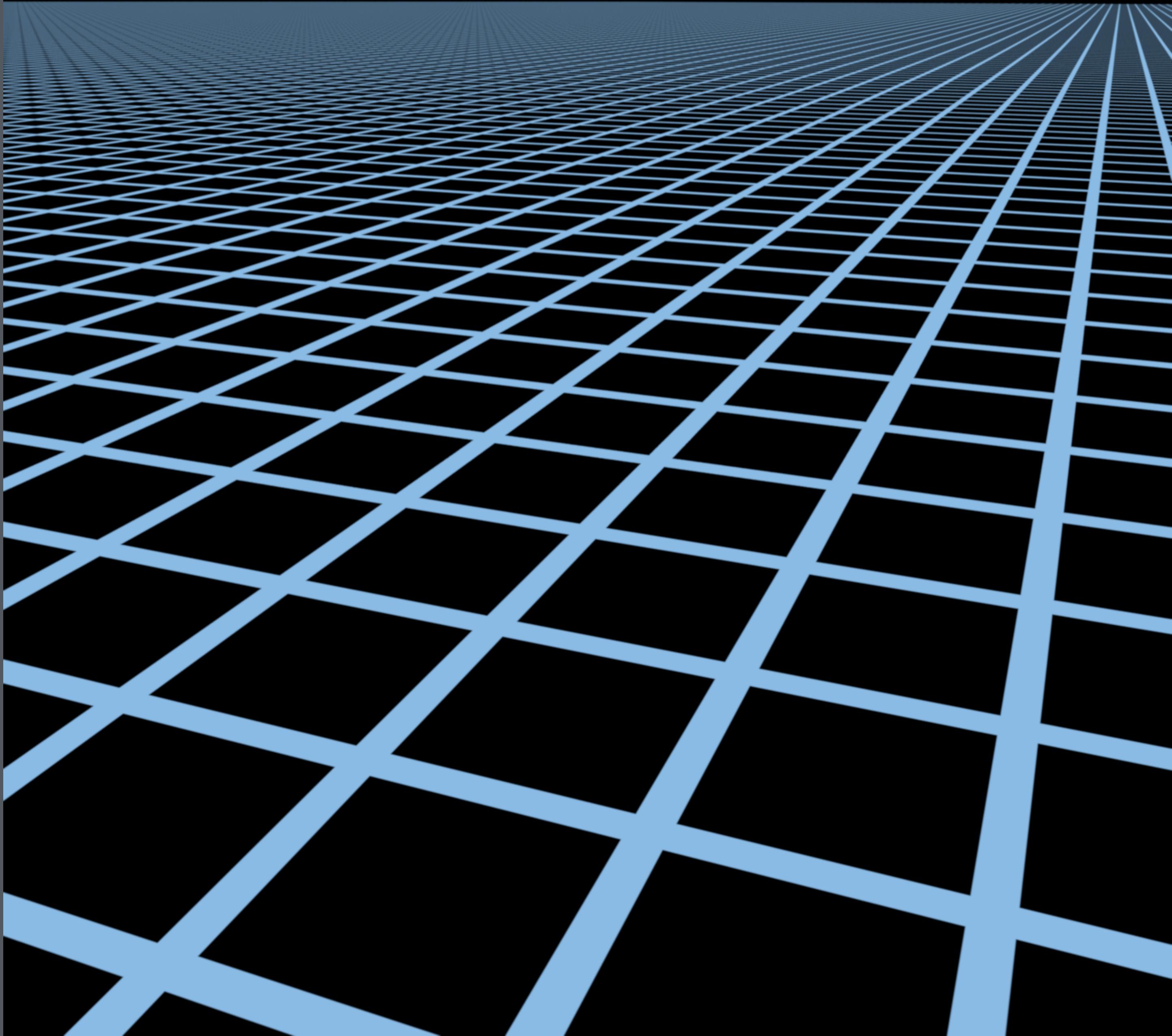
Reference: gaussian
sampling by
512x supersampling



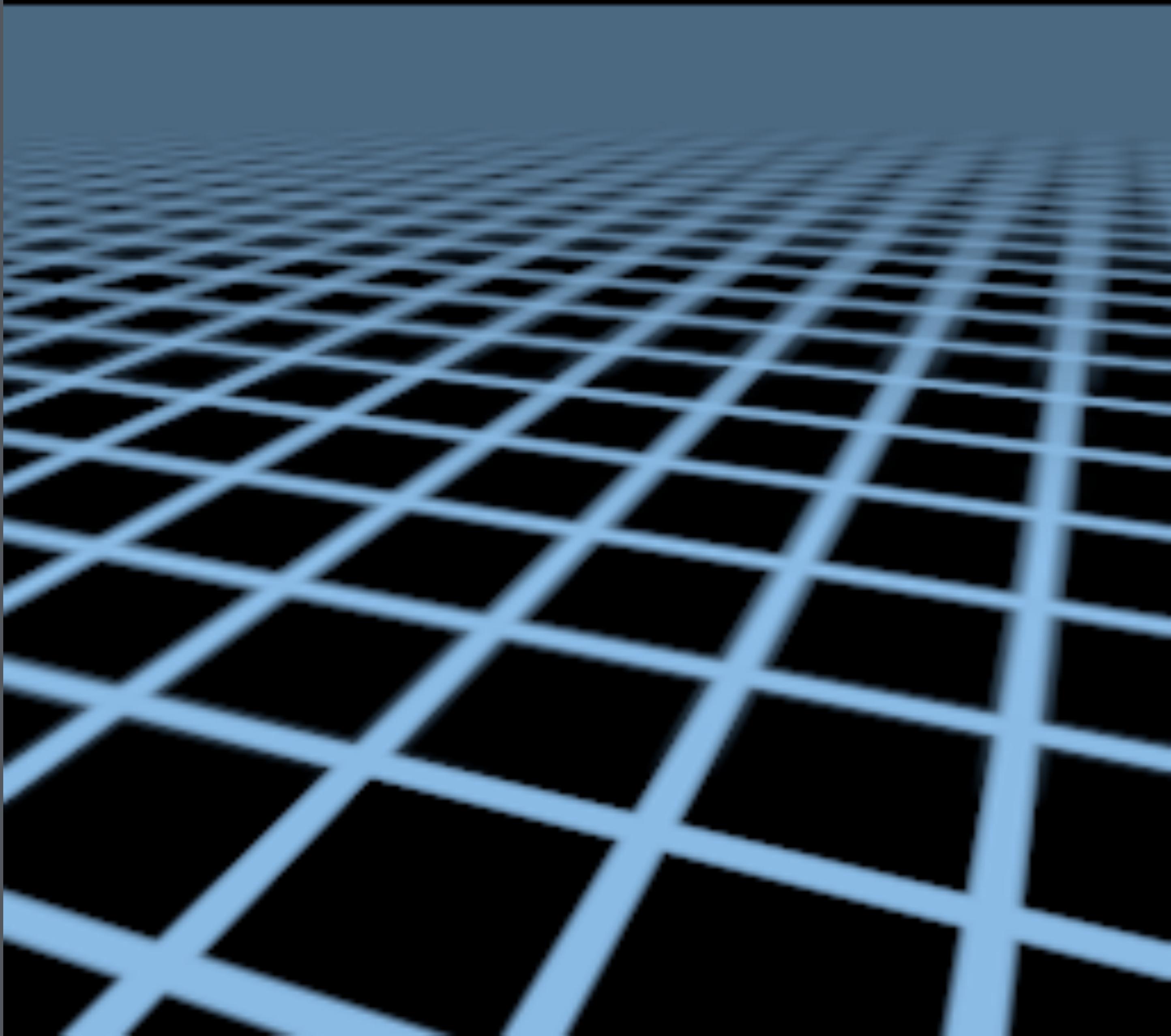
Reference: gaussian
sampling by
512x supersampling



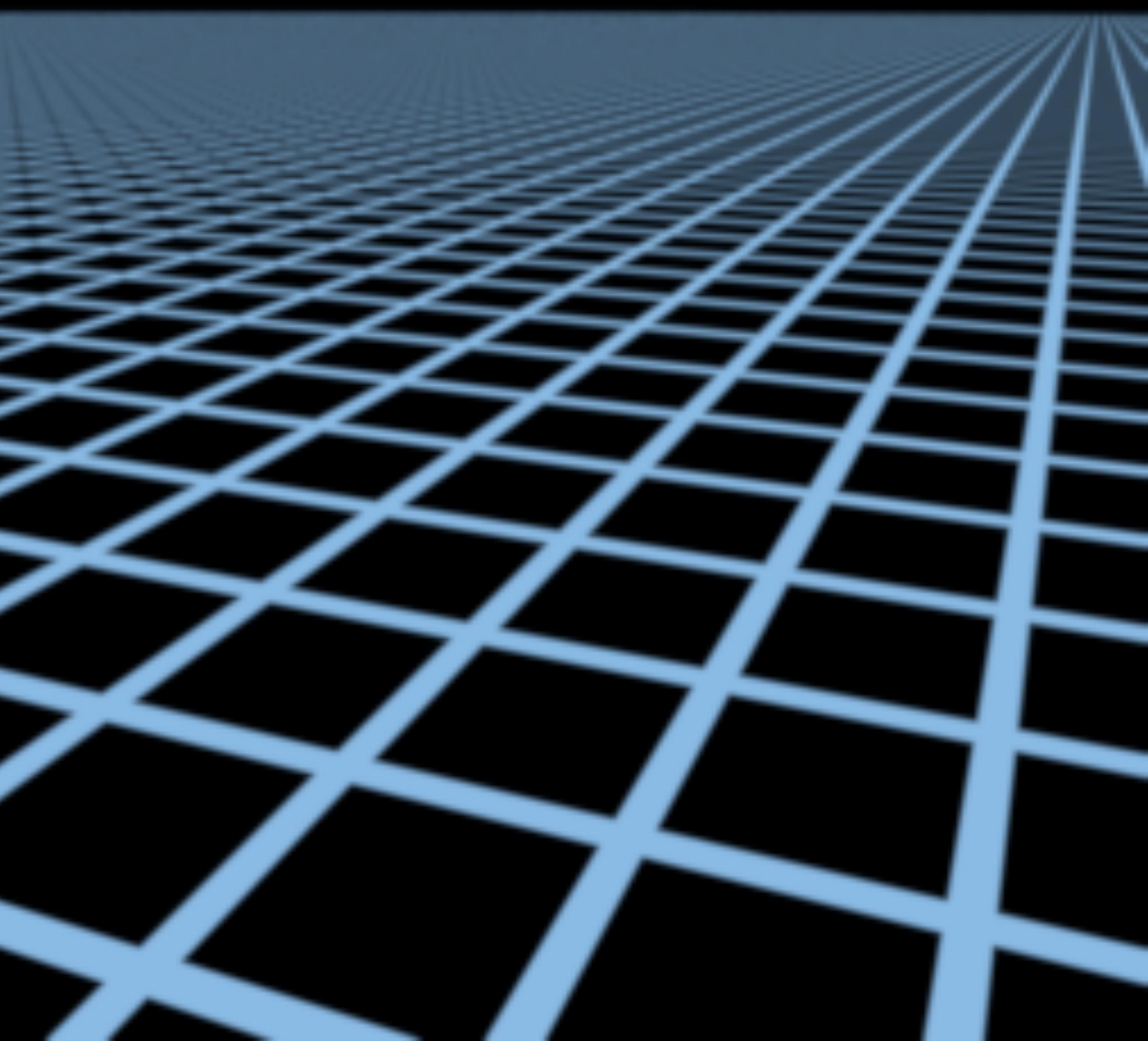
Texture minification with a mipmap



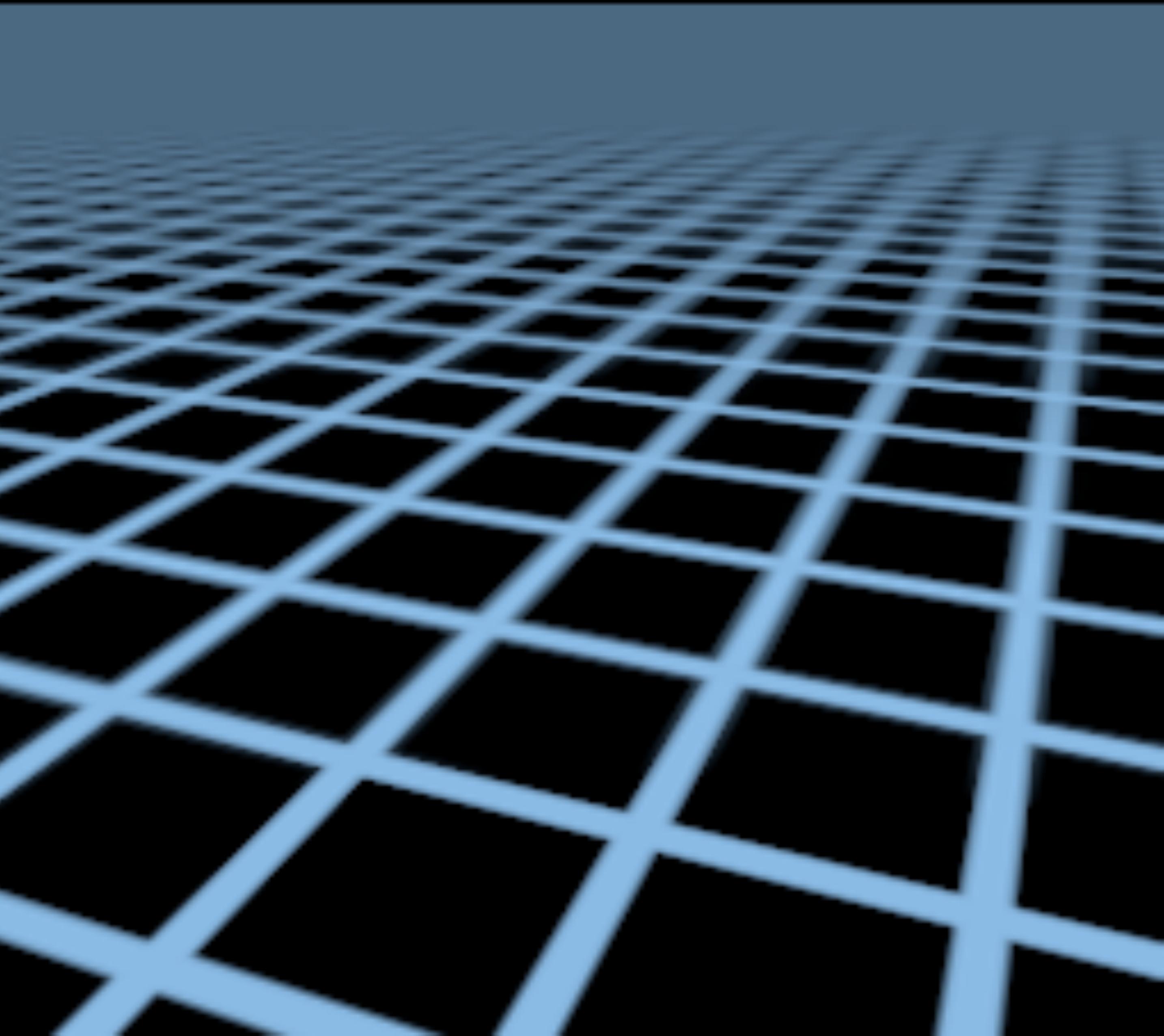
Texture minification with a mipmap



Texture minification:
supersampling vs.
mipmap



Texture minification:
supersampling vs.
mipmap



How to do better?

RIP maps and summed-area tables

- can look up axis-aligned rectangular areas
- diagonals still a big problem!

Elliptical Weighted Average (EWA) filter

- perform multiple lookups
- accumulate using filtering weights
- MIP map pyramid still helps!

EWA filtering (attributed to Greene & Heckbert, but they didn't work out the MIP map part)

Treat pixel as circular

- e.g. Gaussian filter

Use linear apx. for distortion

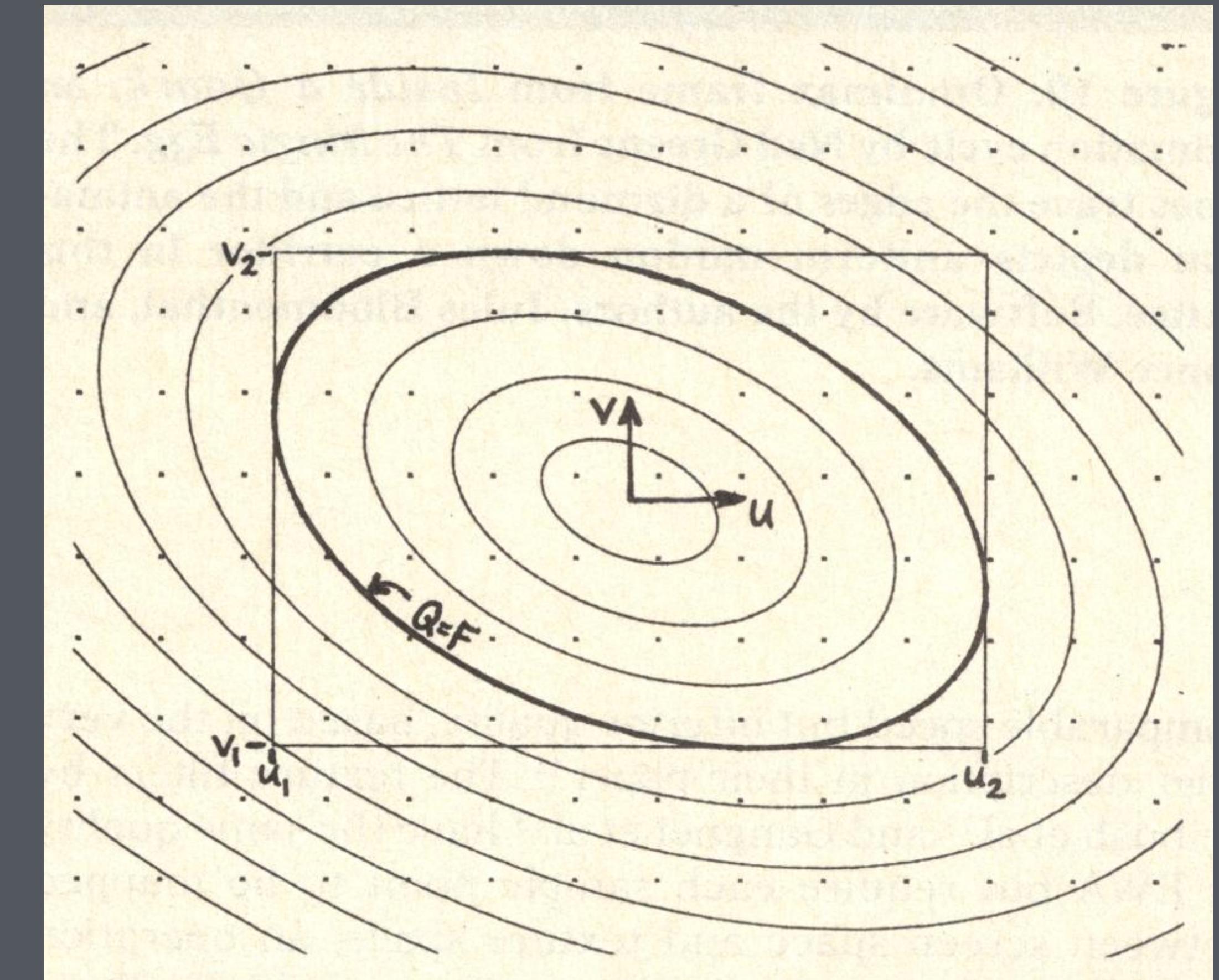
- circular pixel maps to elliptical footprint
- ellipse dimensions calc'd from quadratic

Loop over texels inside ellipse

- actually over bounding rect
- weight by filter value and accumulate

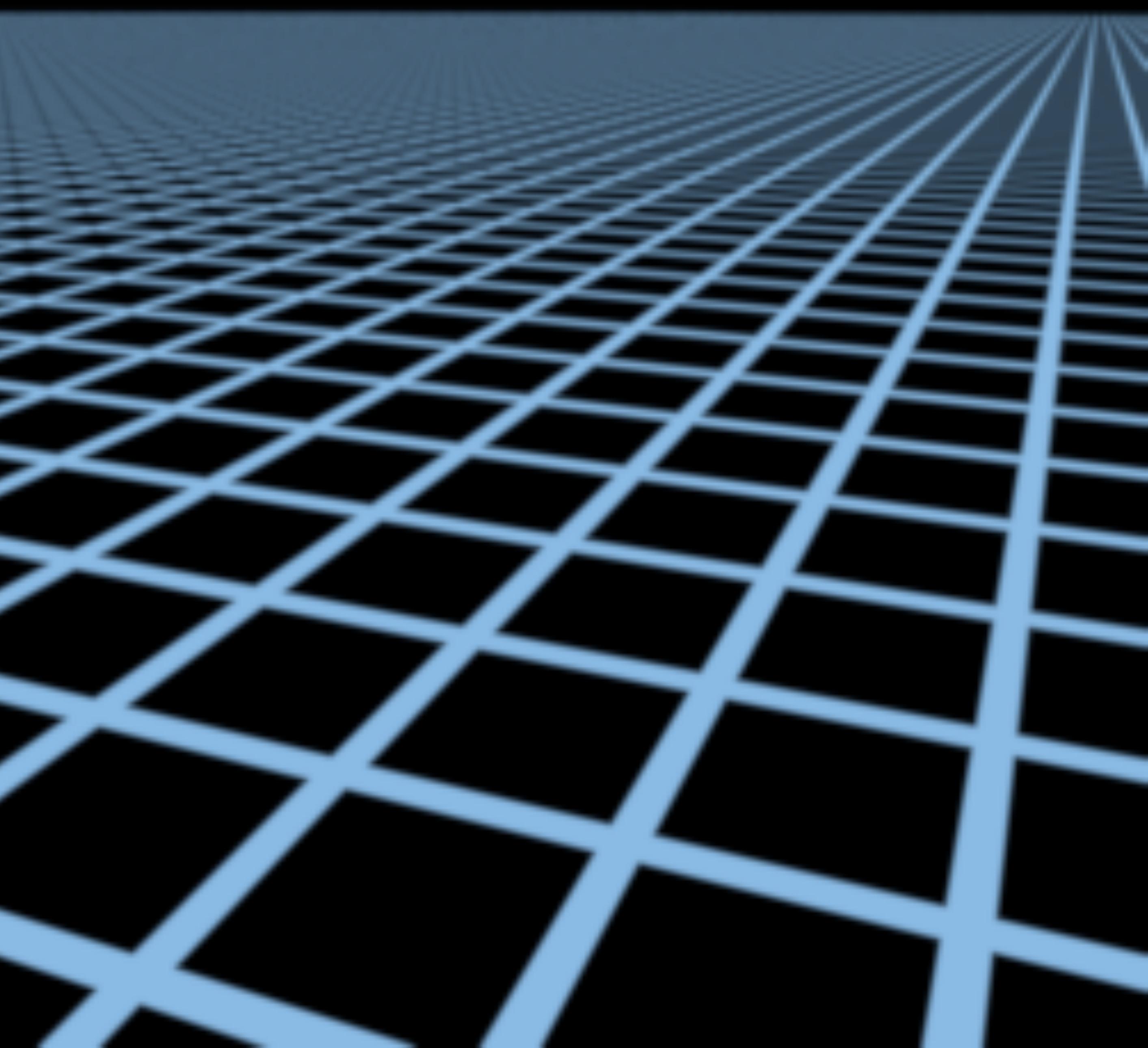
Select appropriate MIP map level

- so that minor radius is 1–2 texels

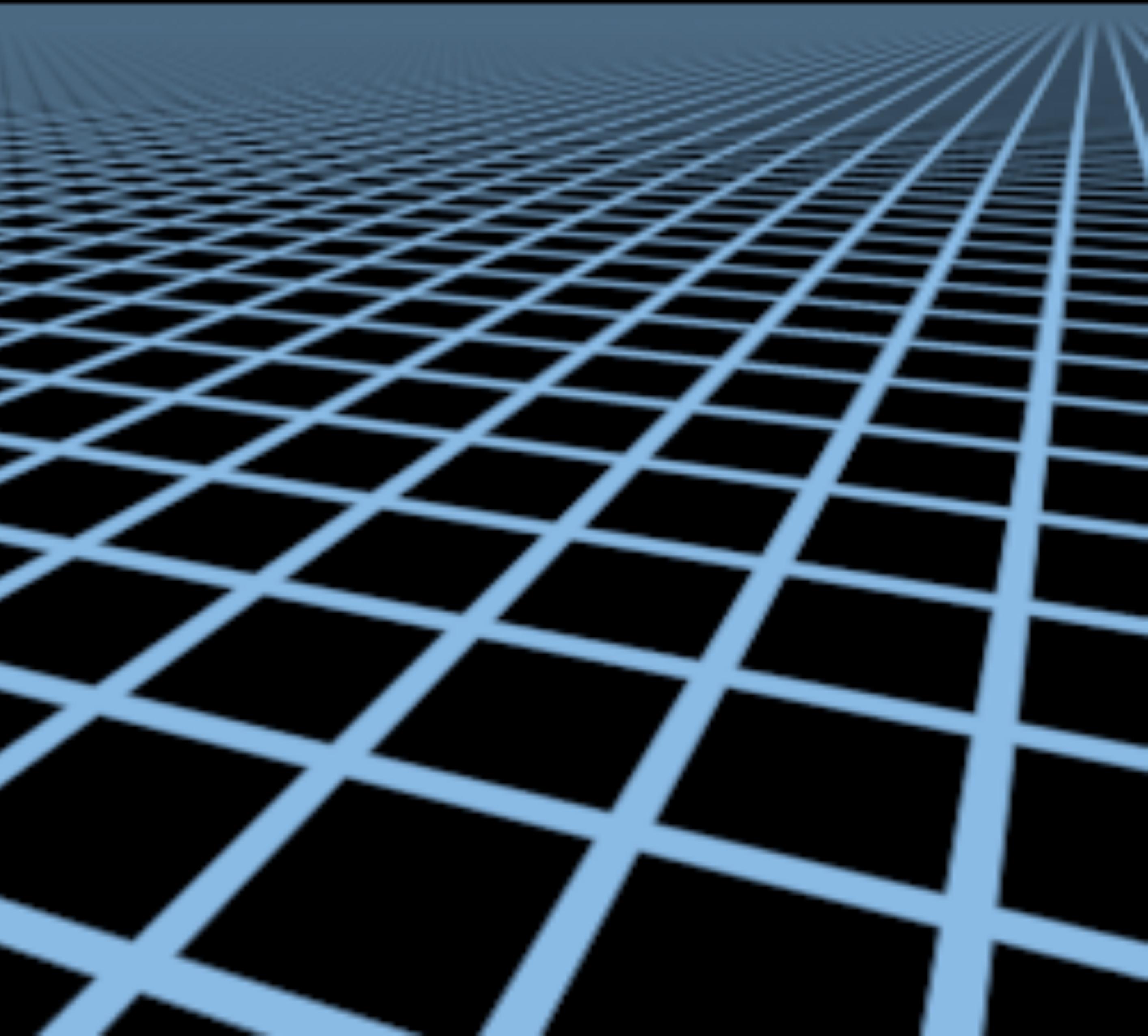


Greene & Heckbert '86

Texture minification:
supersampled vs.
EWA



Texture minification:
supersampled vs.
EWA



Simpler anisotropic MIP mapping

EWA requires a lot of lookups for diagonally oriented footprints

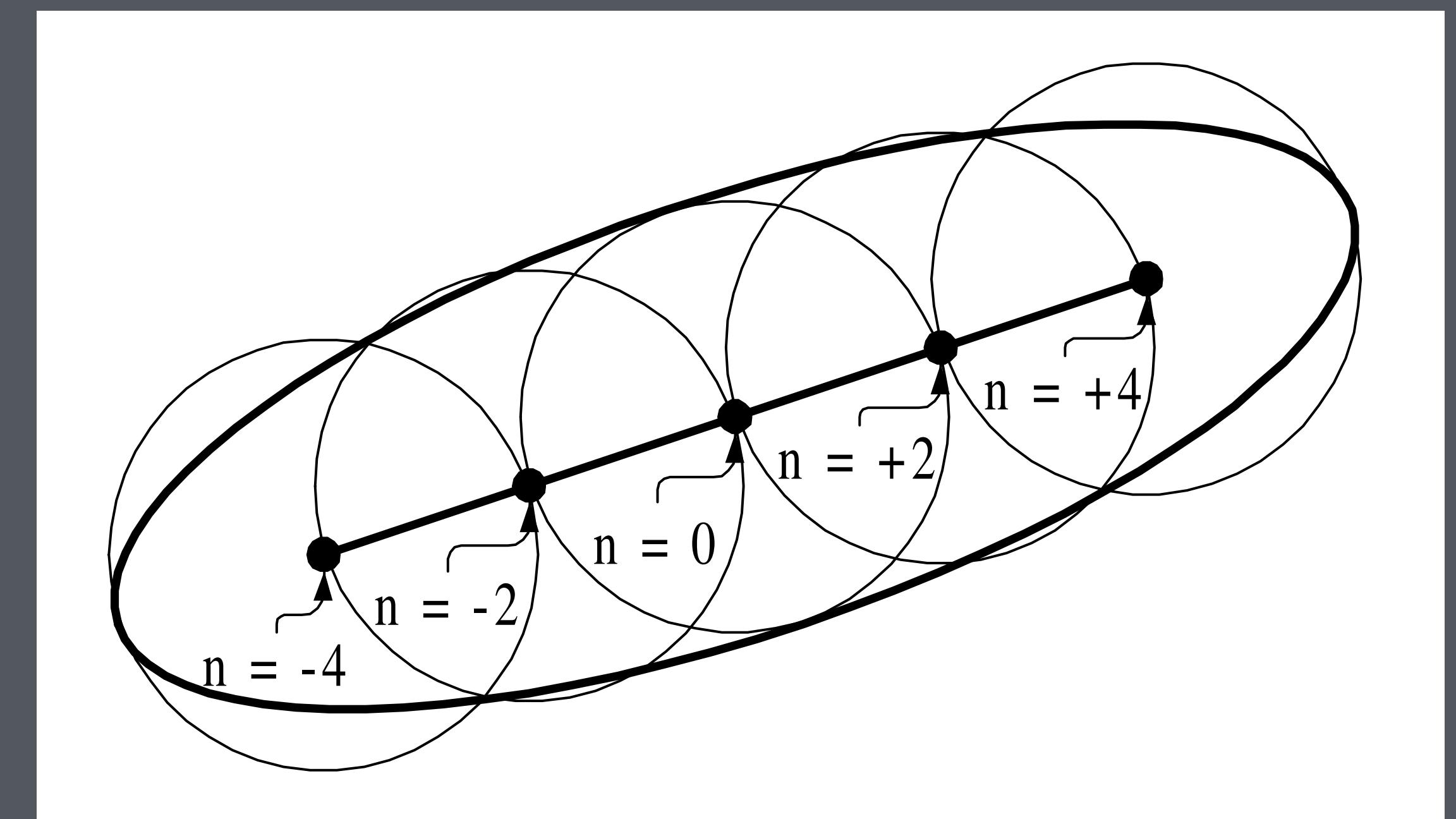
Instead, approximate your footprint as a single line of blobs

- each blob is produced by taking a single bilinear sample using the standard MIP map

Number of samples proportional to major:minor axis ratio

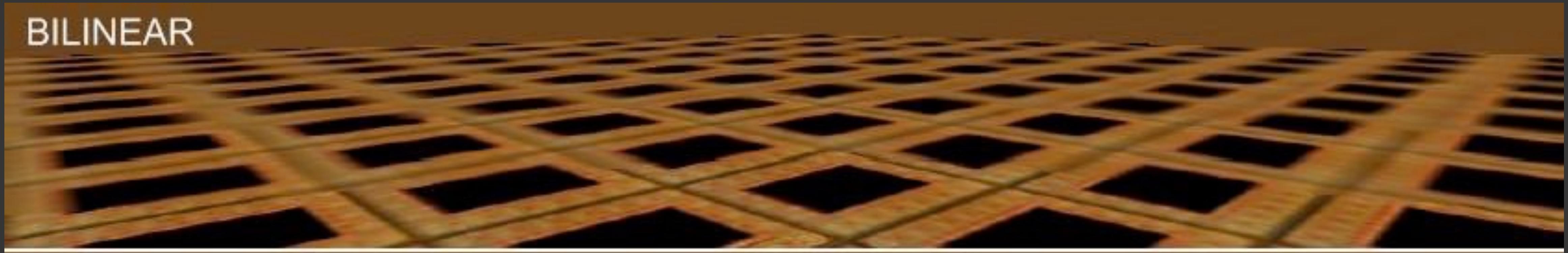
- with some limit to bound slowness in extreme cases

This is the kind of method used when GPU says it uses “16x anisotropic texture sampling”

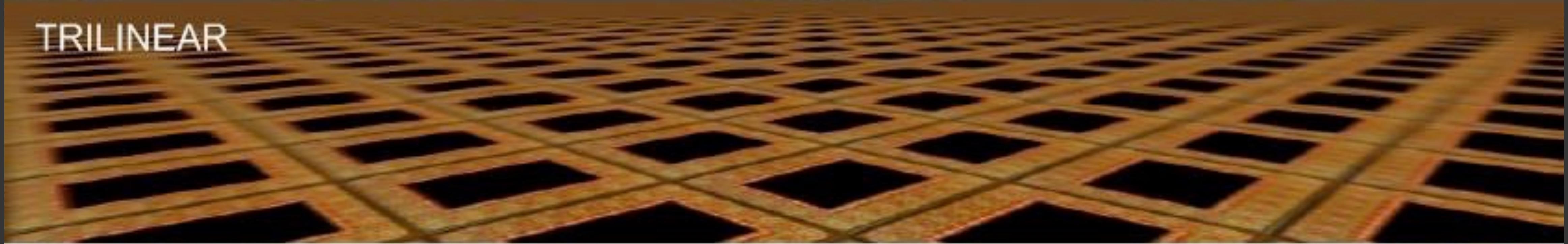


FELINE: McCormack et al. 1999

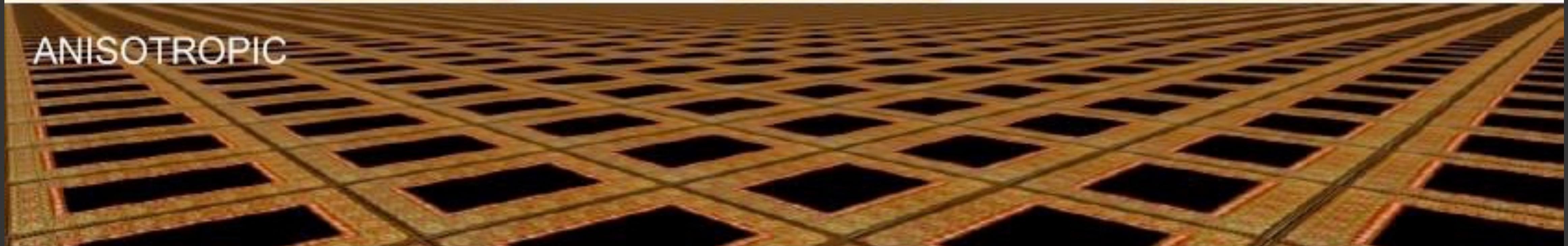
BILINEAR



TRILINEAR



ANISOTROPIC

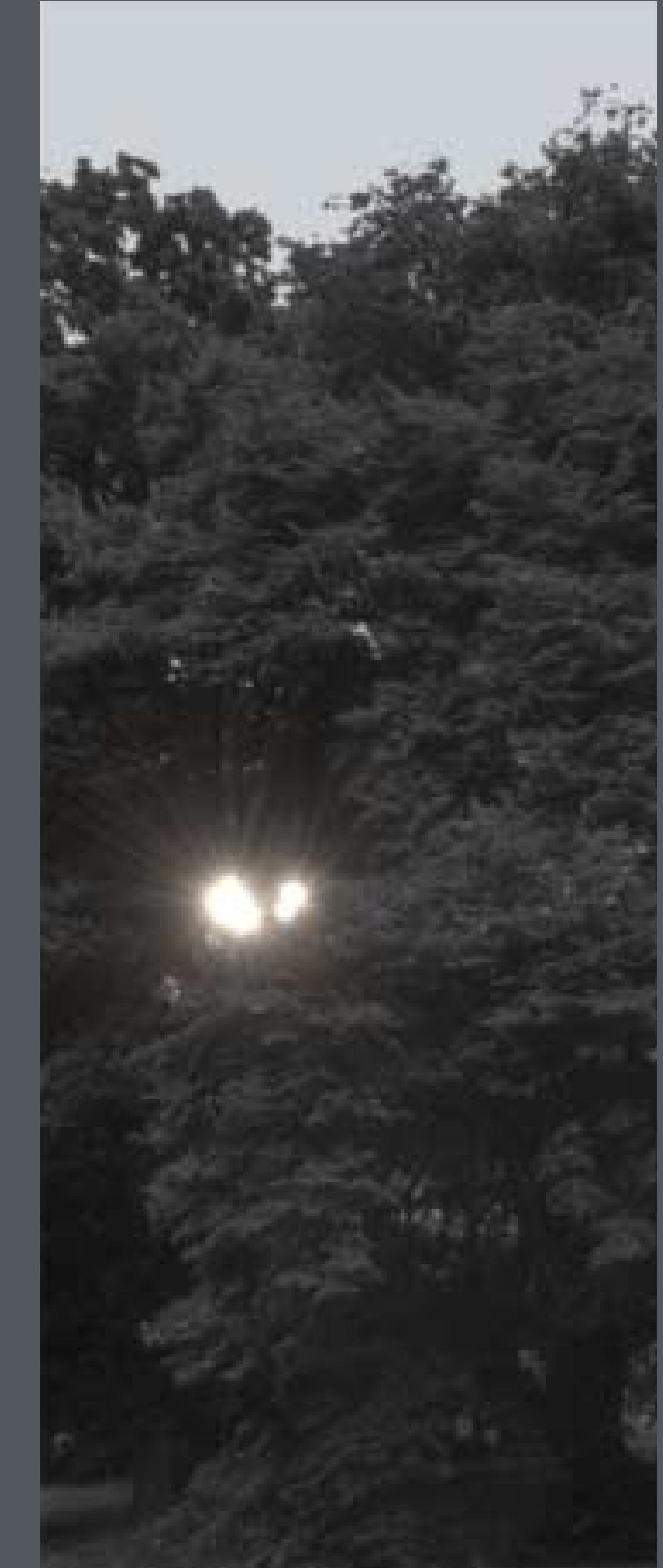
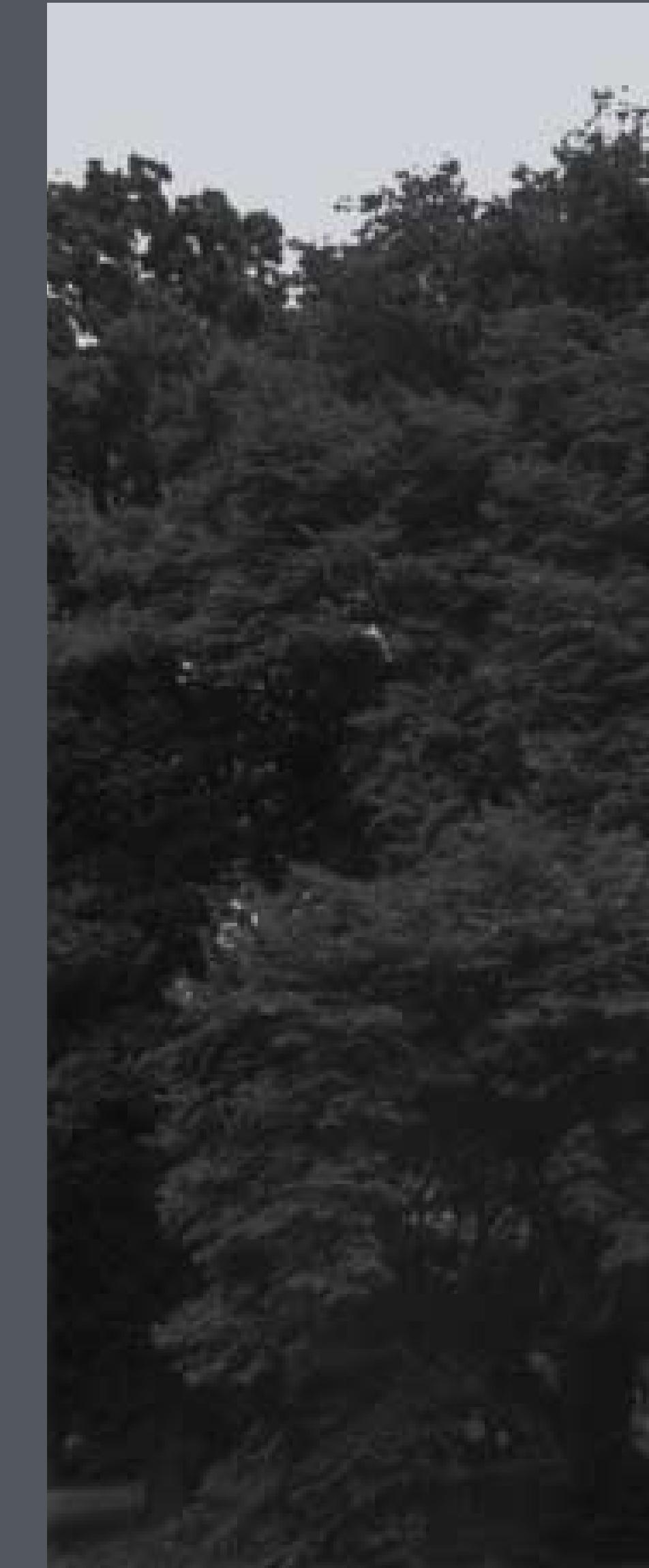
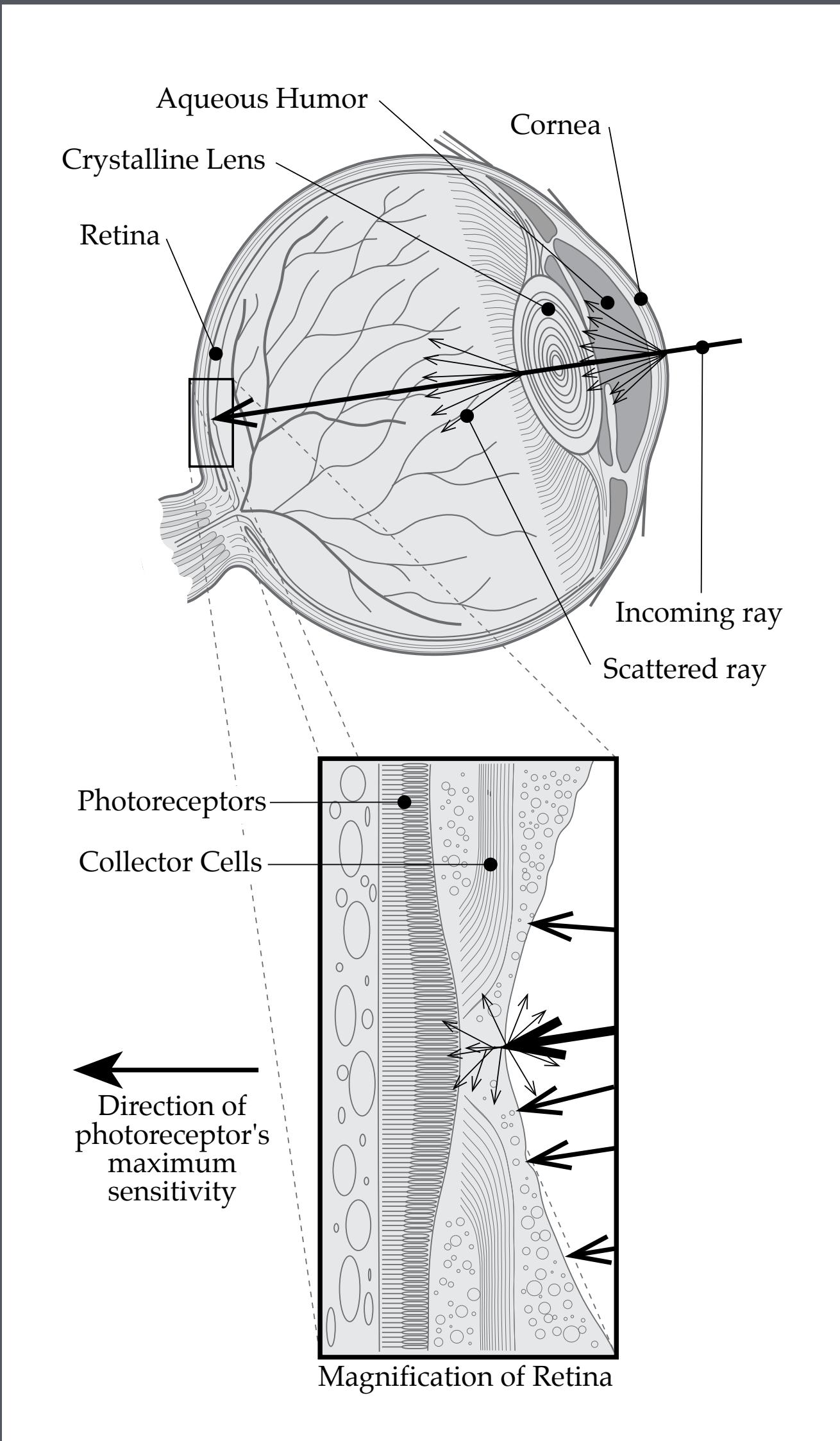


Bloom

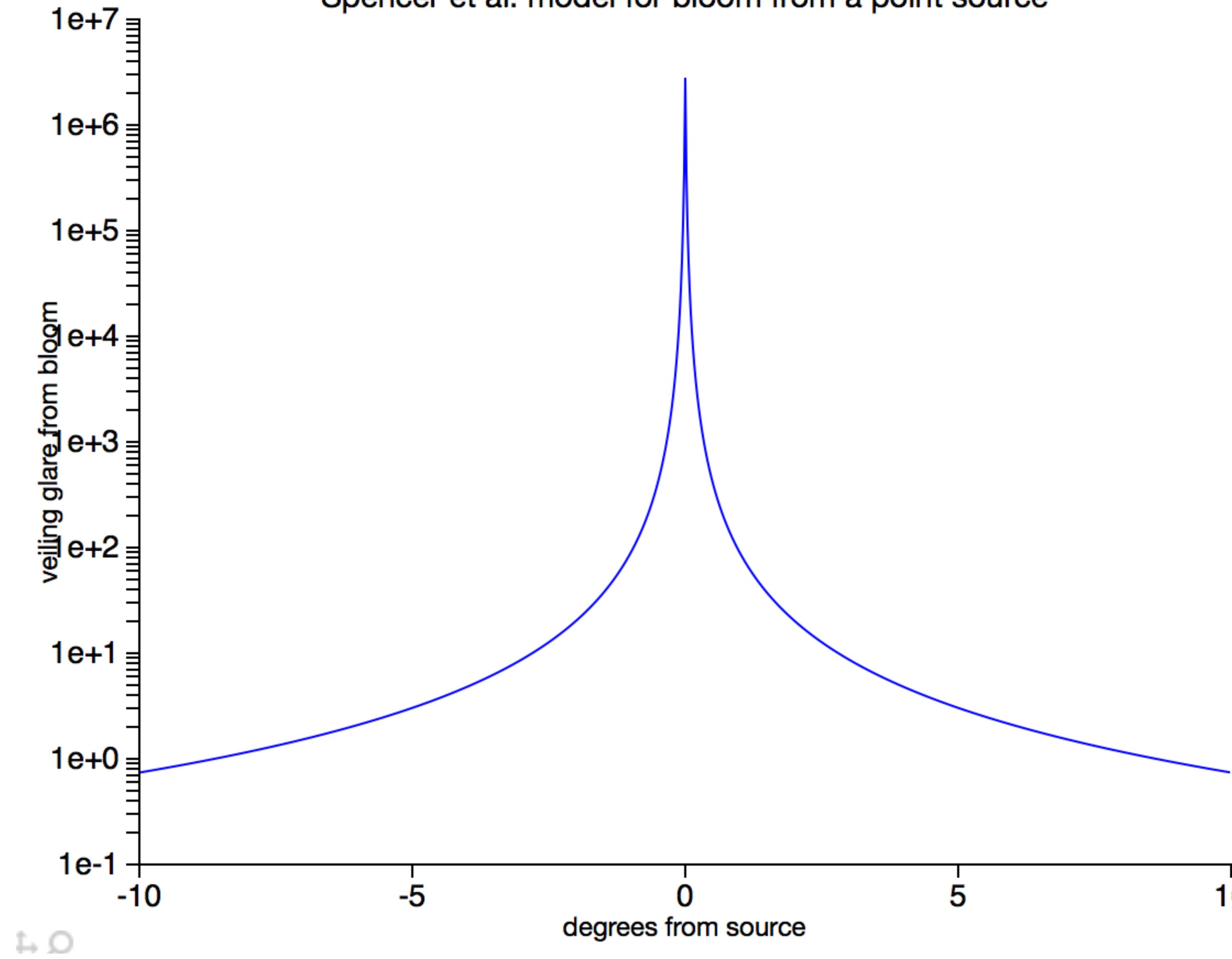


Journey—Thatgamecompany (2012)

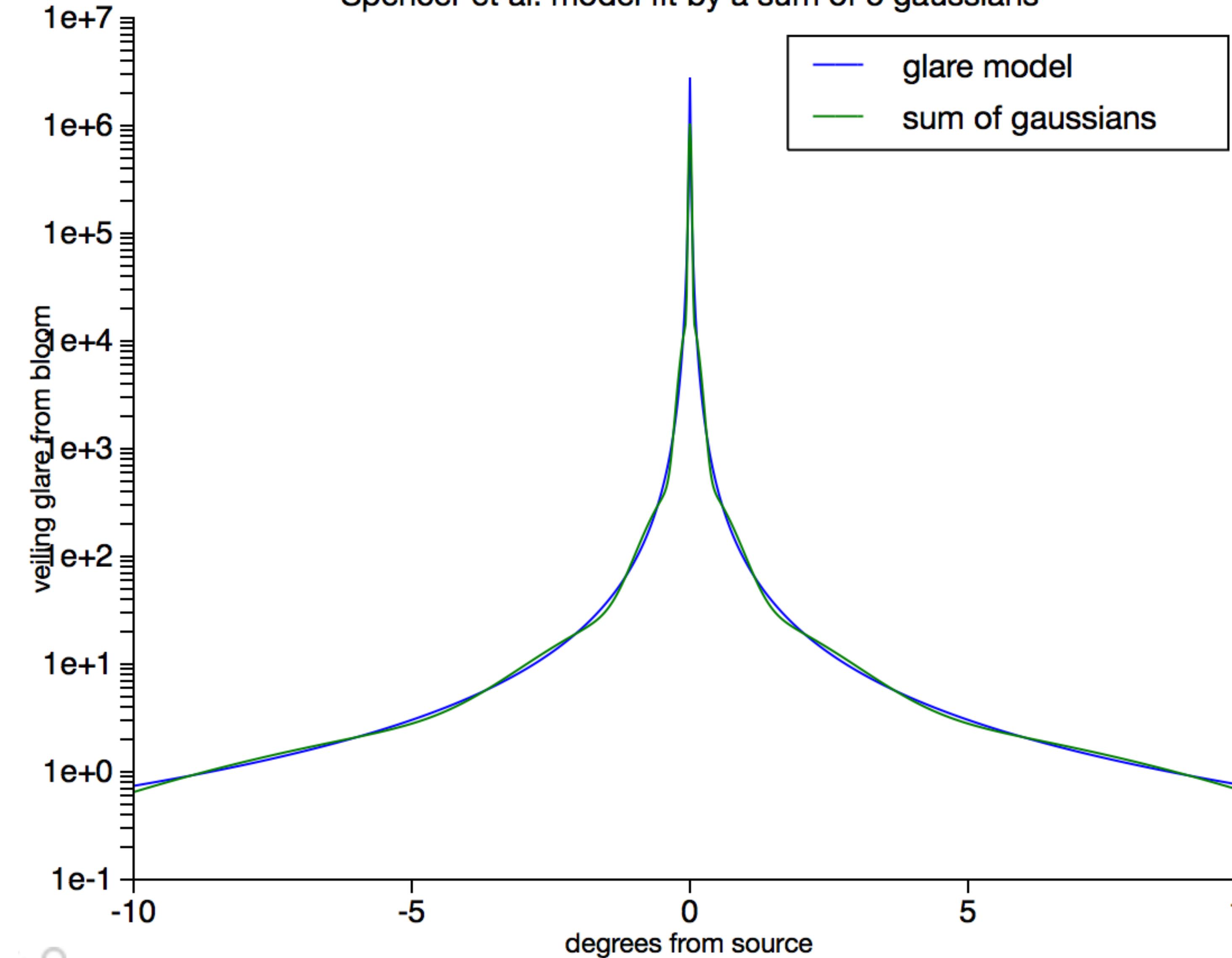
Modeling flare in the eye



Spencer et al. model for bloom from a point source



Spencer et al. model fit by a sum of 5 gaussians



$$0.8843\delta(x) + 0.1g(6.2, x) + 0.012g(24.9, x) + 0.0027g(81.0, x) + 0.001g(263, x)$$

where $g(\sigma, x)$ is a gaussian with mean 0 and standard deviation σ

Constructing bloom from blurs



Constructing bloom from blurs



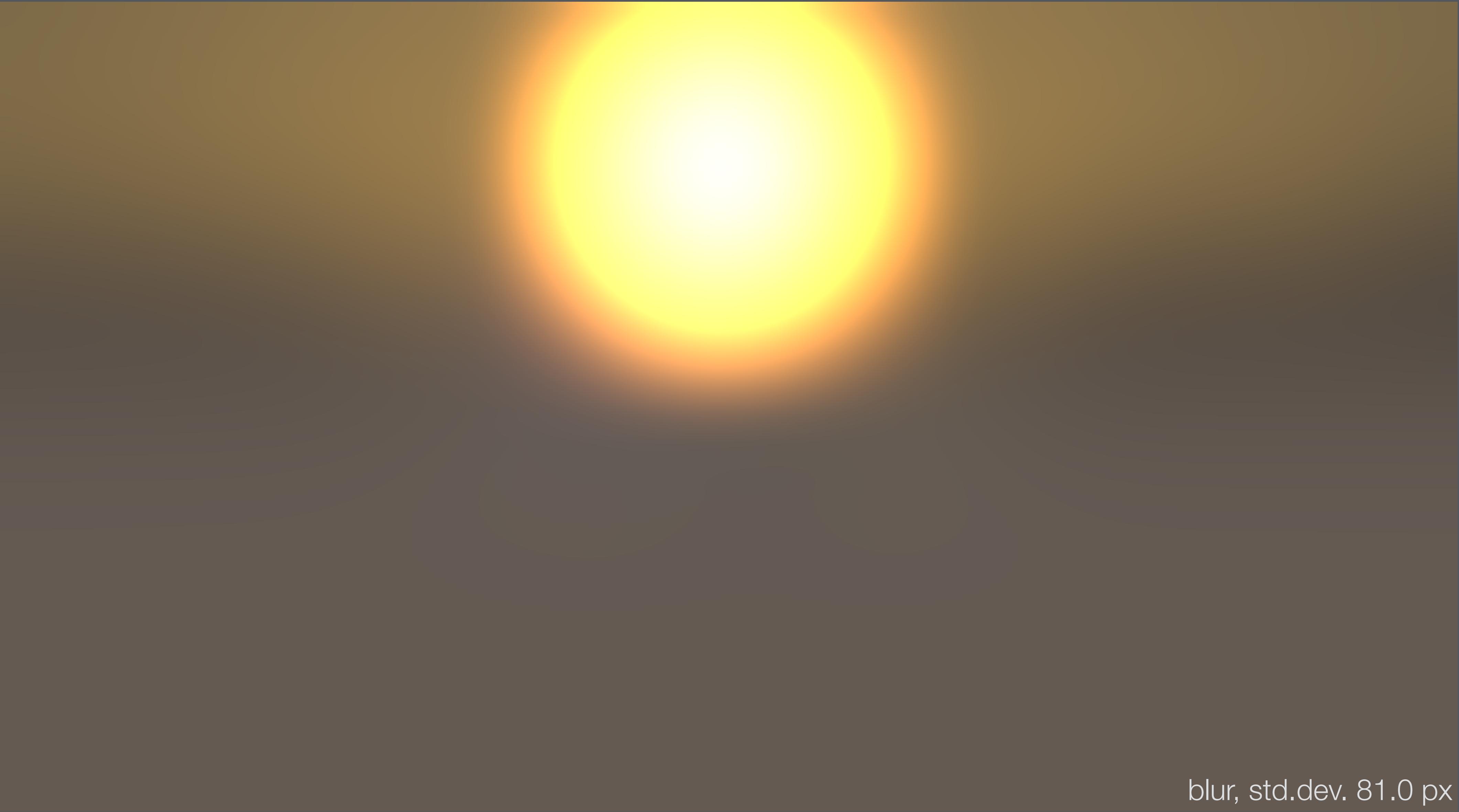
blur, std.dev. 6.2 px

Constructing bloom from blurs



blur, std.dev. 24.9 px

Constructing bloom from blurs

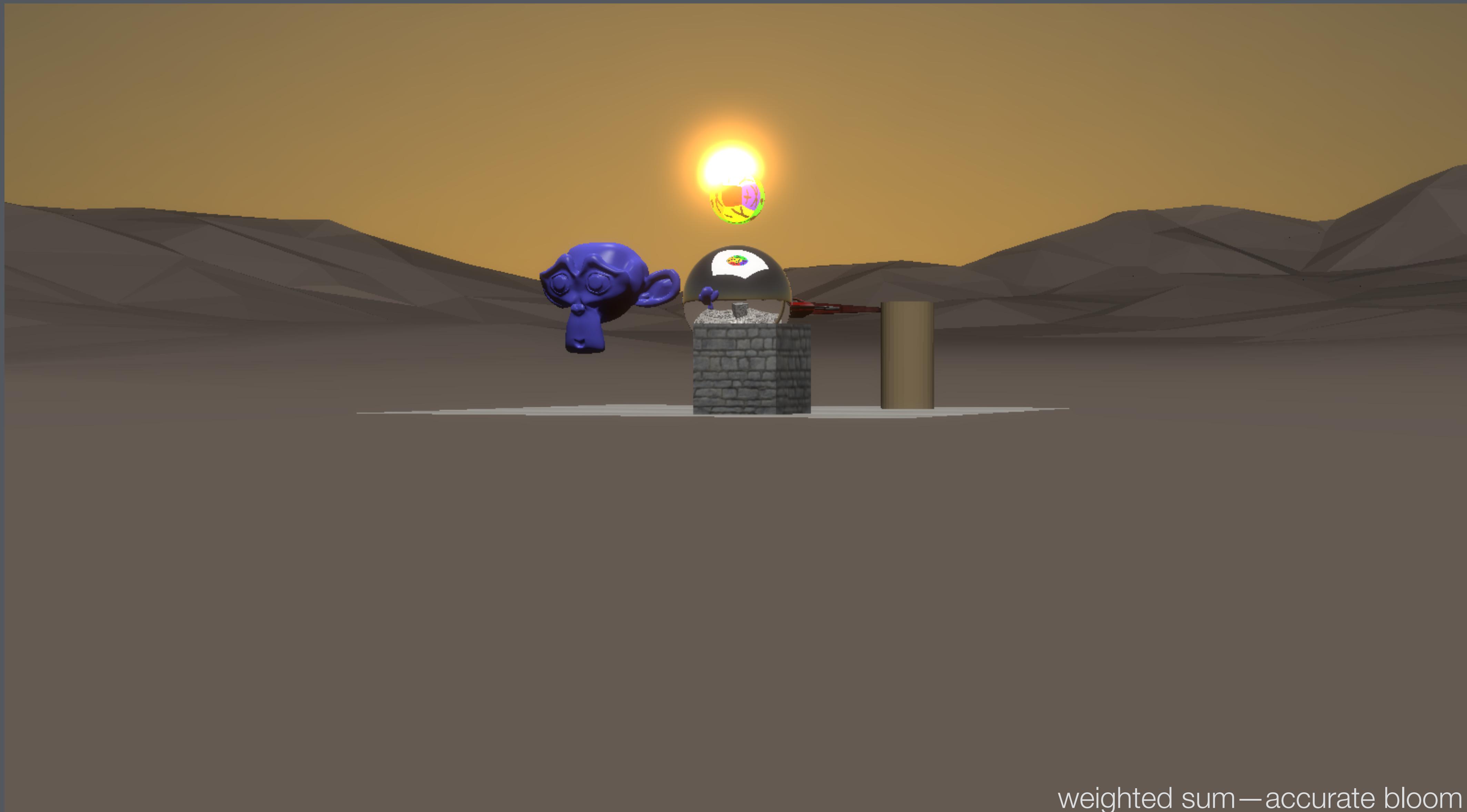


Constructing bloom from blurs



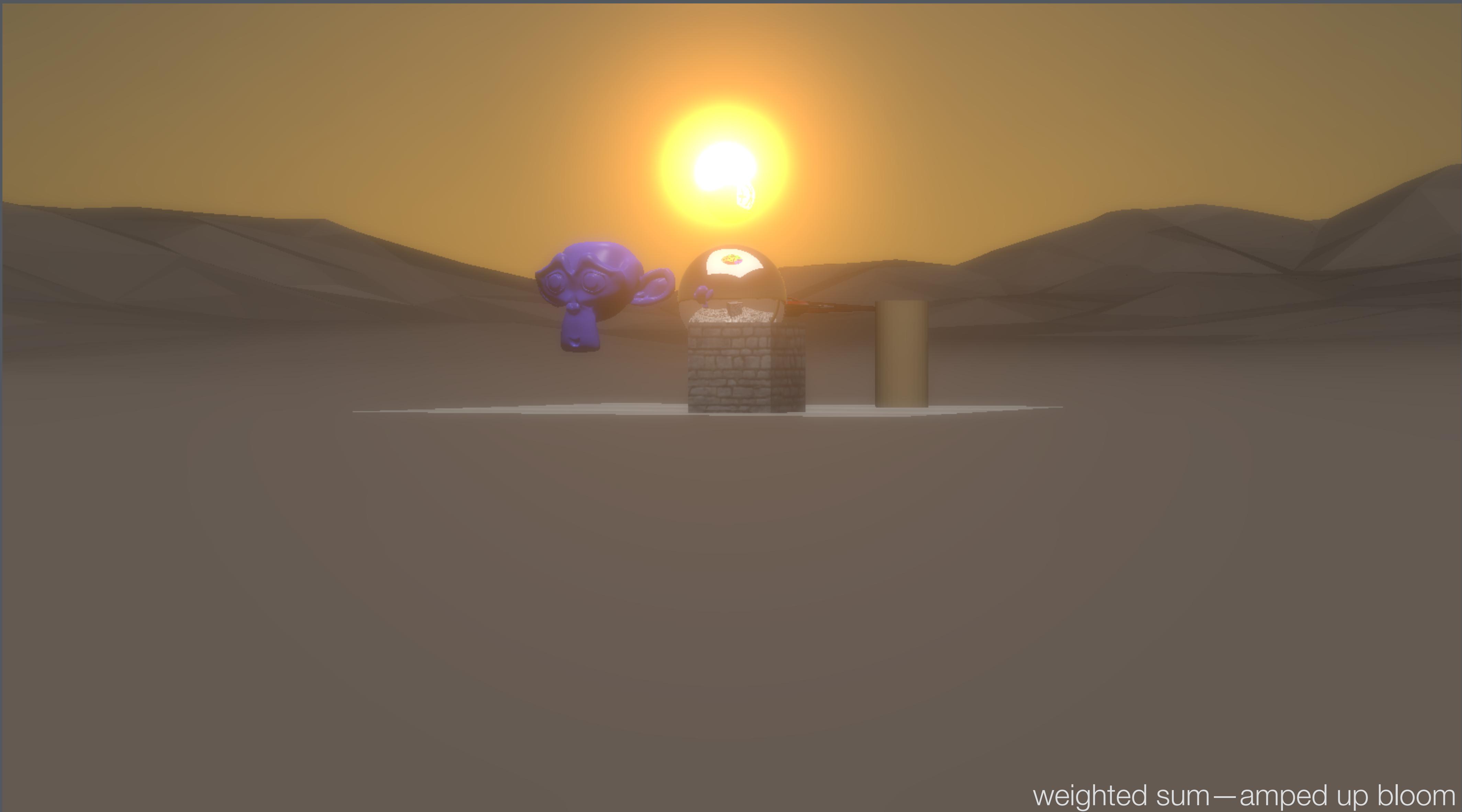
blur, std.dev. 263 px

Constructing bloom from blurs



weighted sum—accurate bloom

Constructing bloom from blurs



weighted sum—amped up bloom

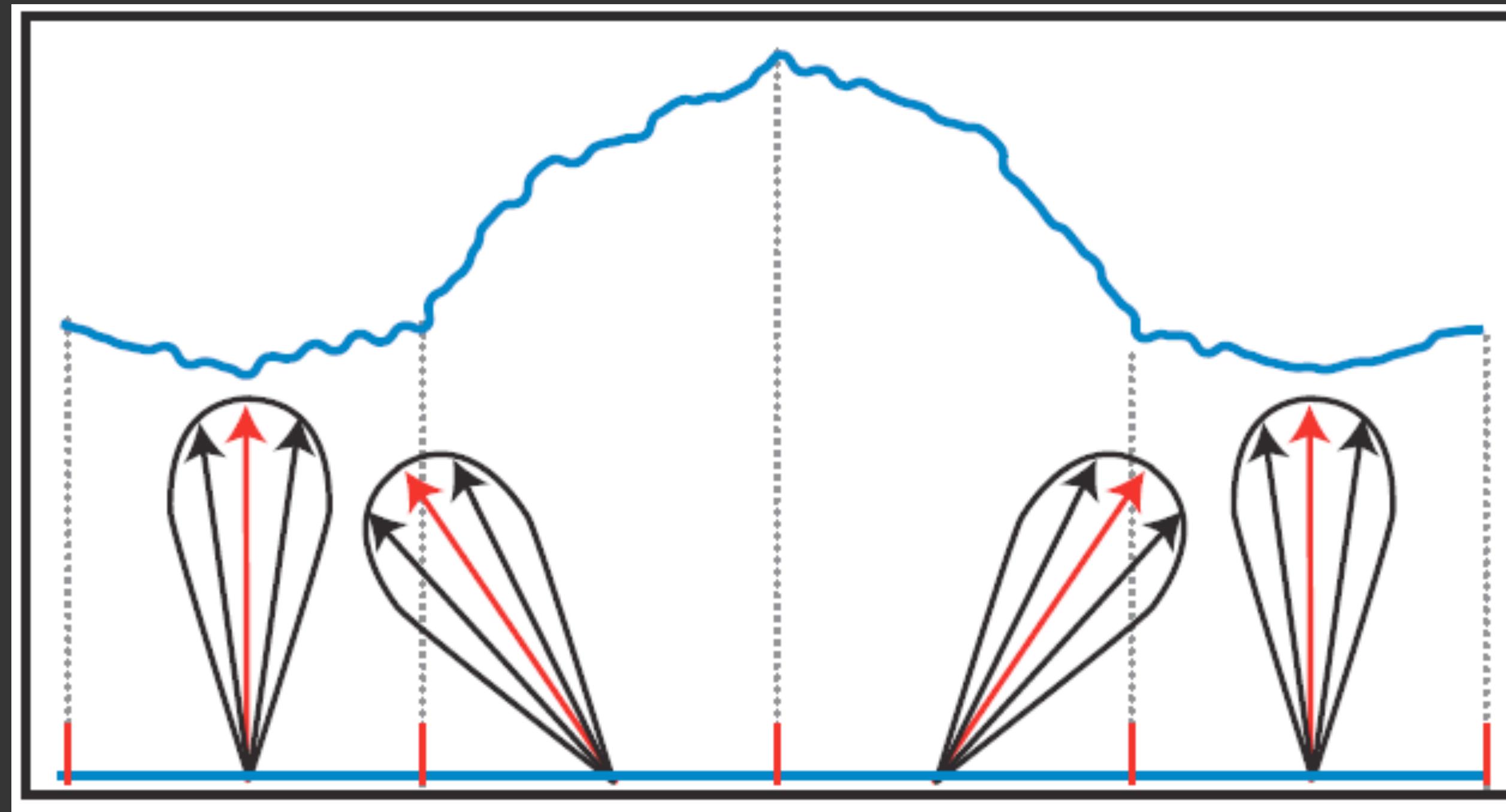
Modeling flare in a camera lens



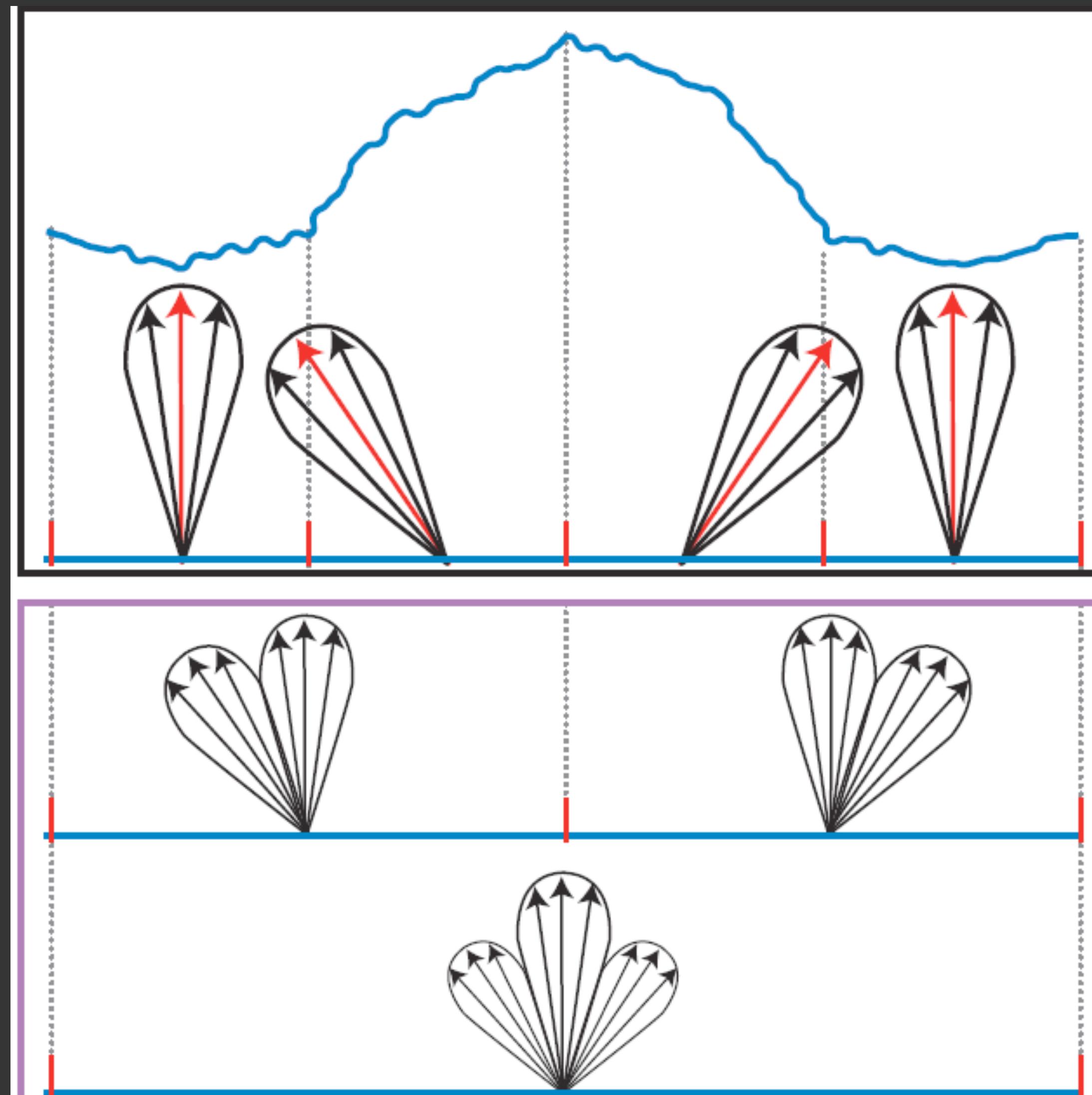
Hullin et al. SIGGRAPH 2011

MIPMapping Normal Maps

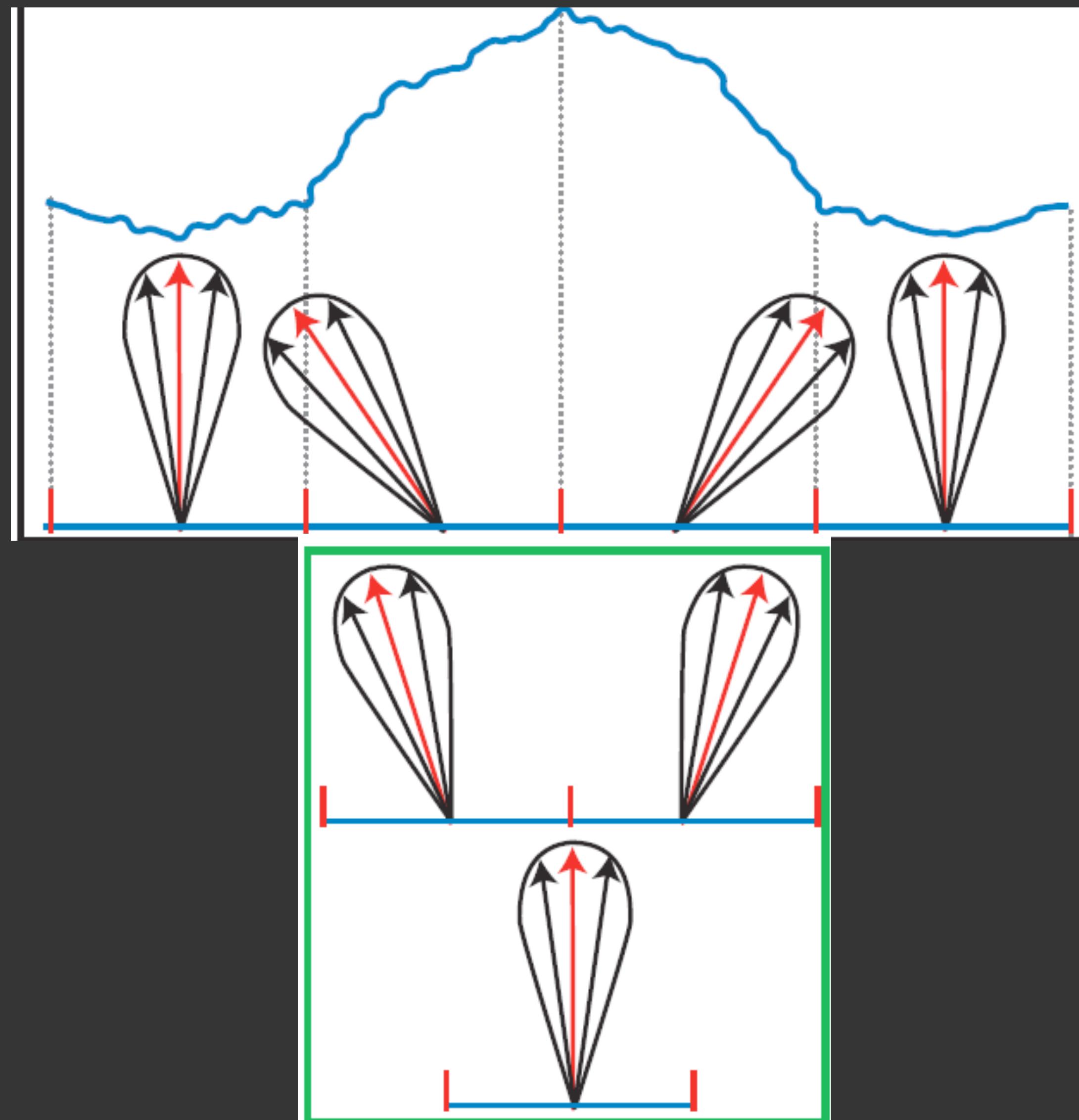
- Normal Distribution Function (NDF)
 - Normal in texel + phong lobe represent a bunch of small detail



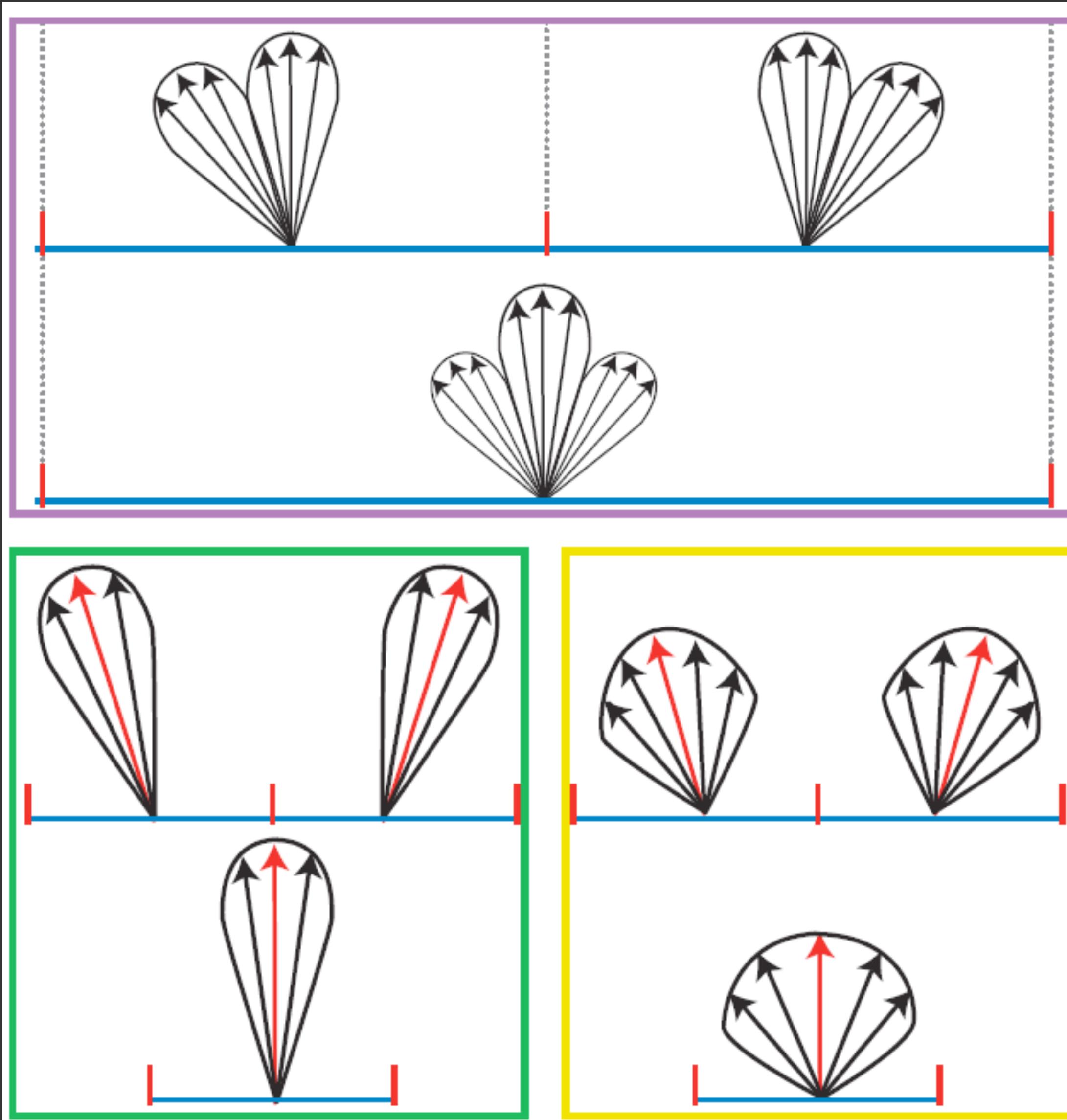
Normal Map Minification: Ideal



Normal Map Minification: Averaging



Normal Map Minification: Refit



LEAN Mapping

Linear Efficient Anisotropic Normal Mapping

A practical and efficient normal map antialiasing approach

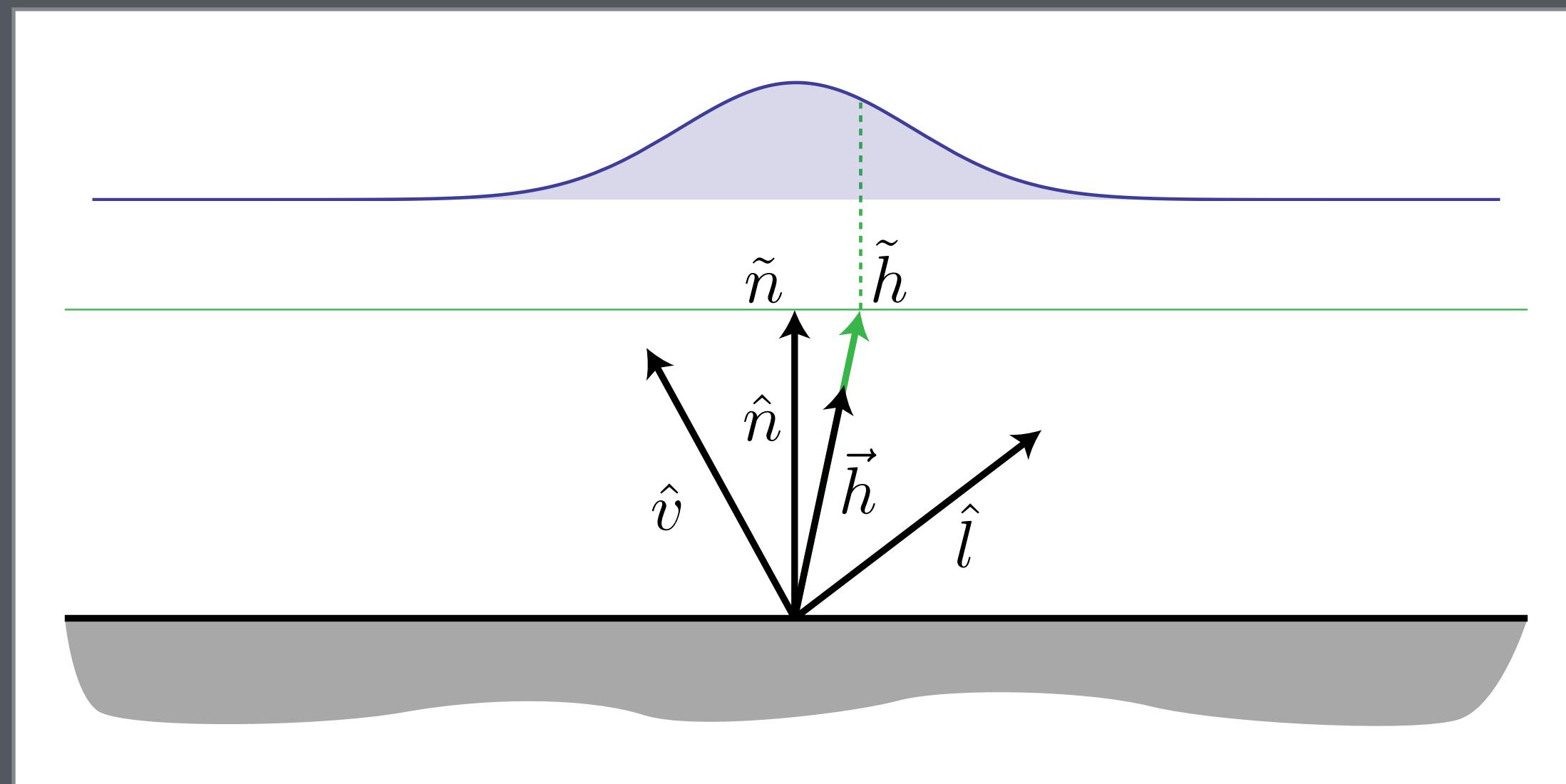
Key ideas:

- Approximate normal mapping as defining a shifted normal distribution function (NDF) (rather than changing the shading frame)

$$e^{-\frac{1}{2} \tilde{h}_b^T \Sigma^{-1} \tilde{h}_b} \rightarrow e^{-\frac{1}{2} (\tilde{h}_n - \tilde{b}_n)^T \Sigma^{-1} (\tilde{h}_n - \tilde{b}_n)}$$

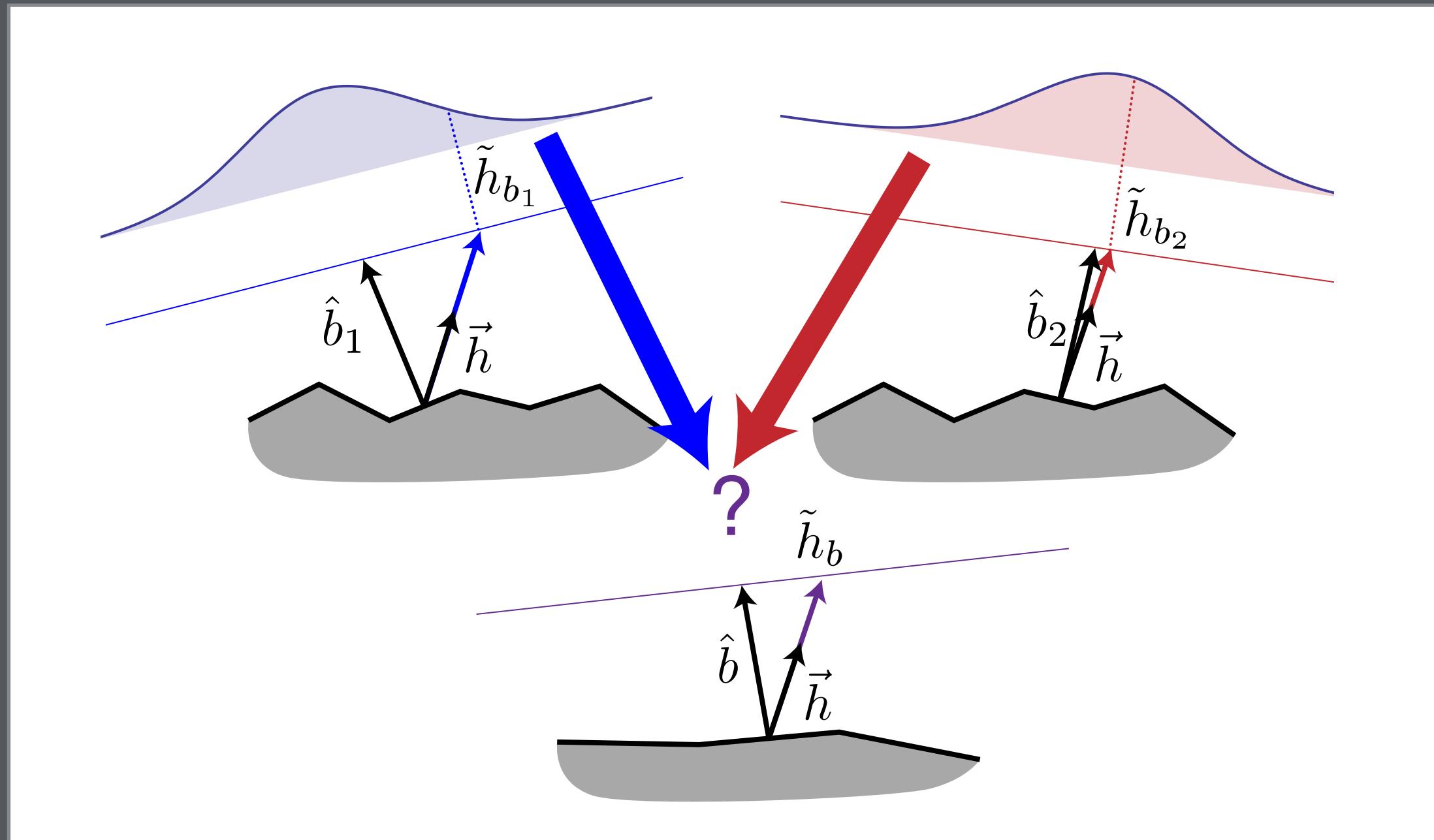
- Use Gaussians for the NDFs
- Approximate the sum of multiple Gaussians by adding the first and second moments

LEAN Mapping

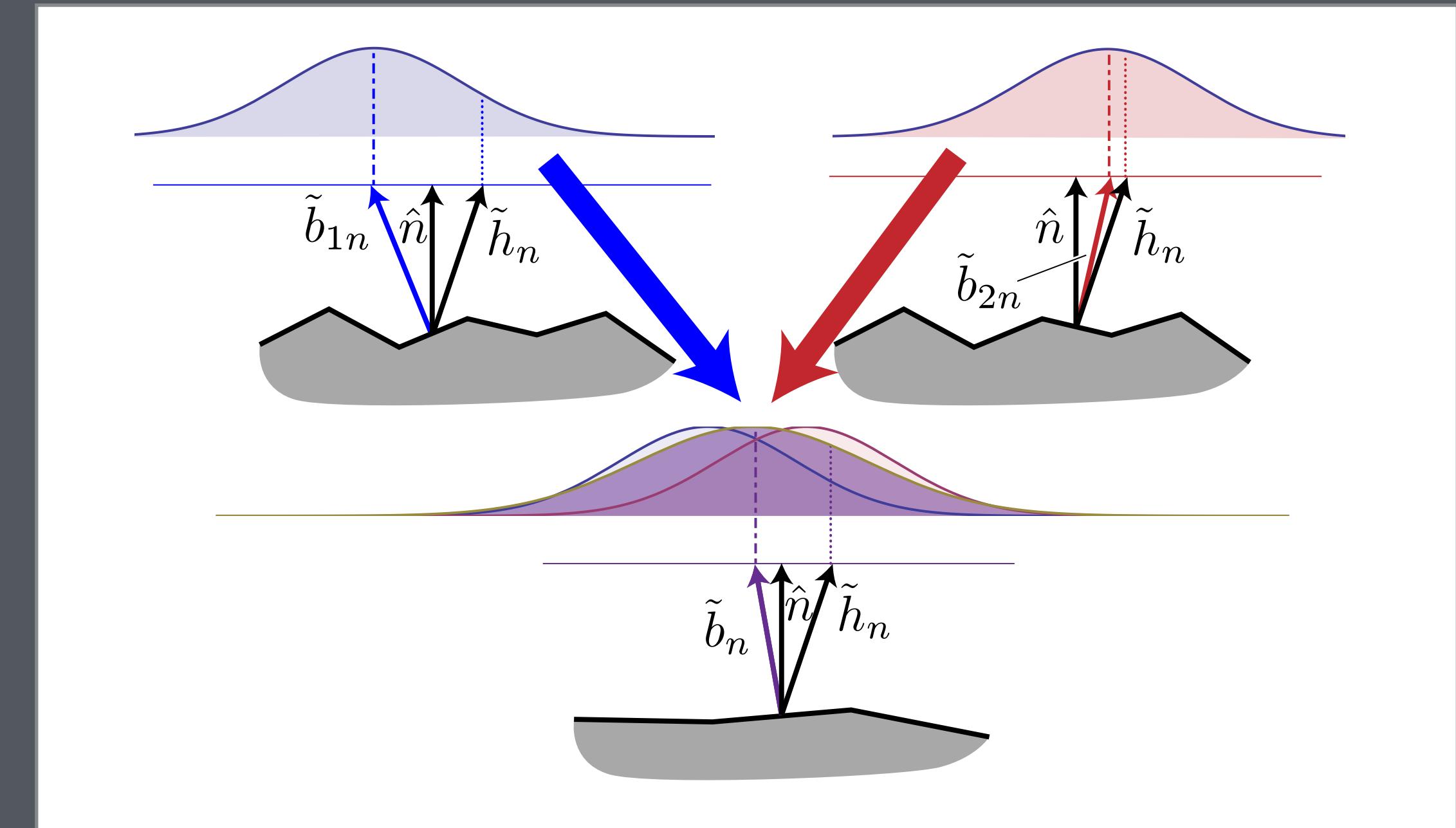


an NDF in tangent-vector space

LEAN Mapping



combining two centered NDFs
in different tangent spaces



combining two off-center NDFs
in a common tangent space

LEAN mapping bottom line [Olano & Baker 2010]

Given normals from a normal map:

$$N = (\vec{b}_n.x, \vec{b}_n.y, \vec{b}_n.z)$$

Store the following in the base level texture:

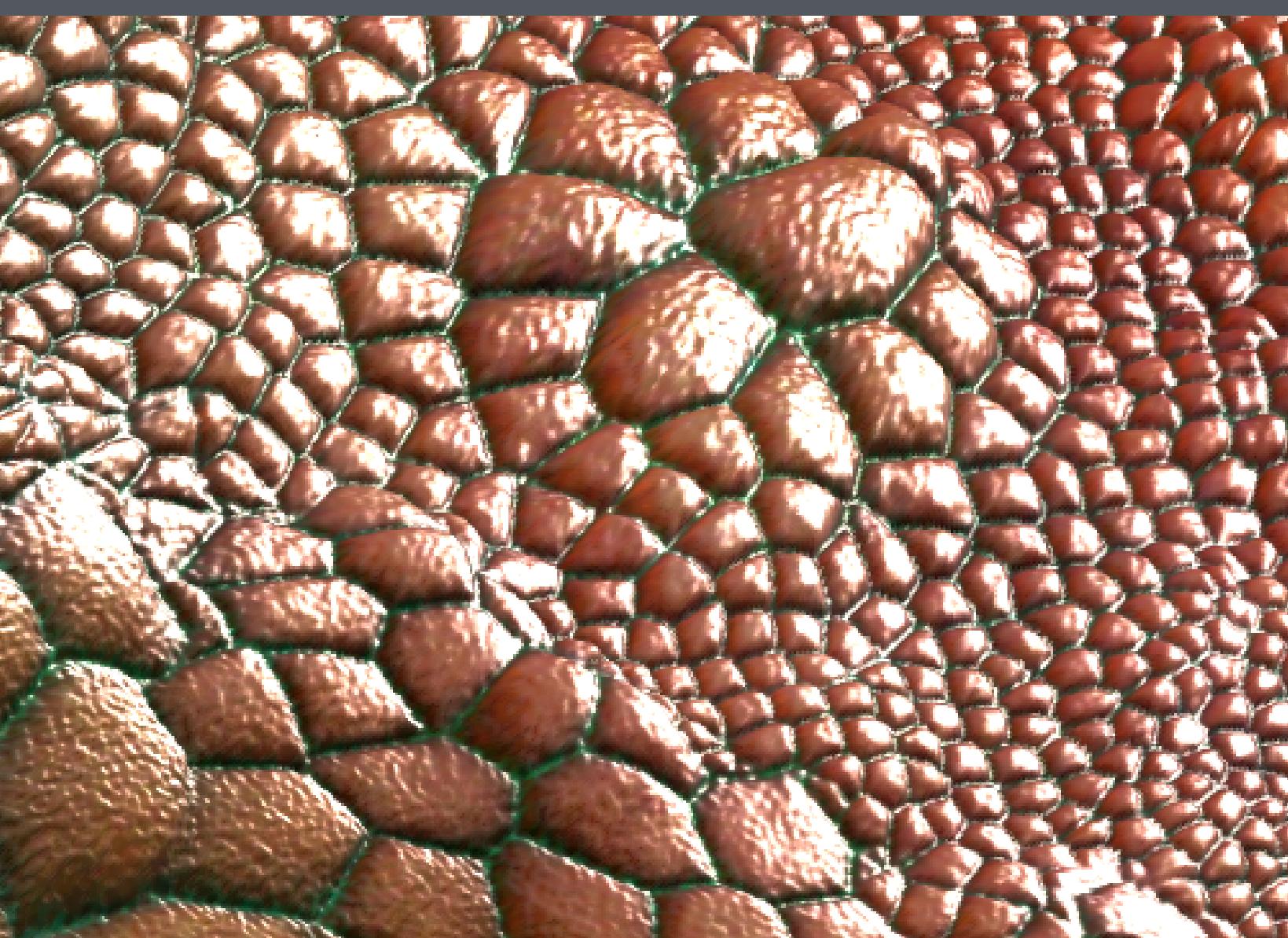
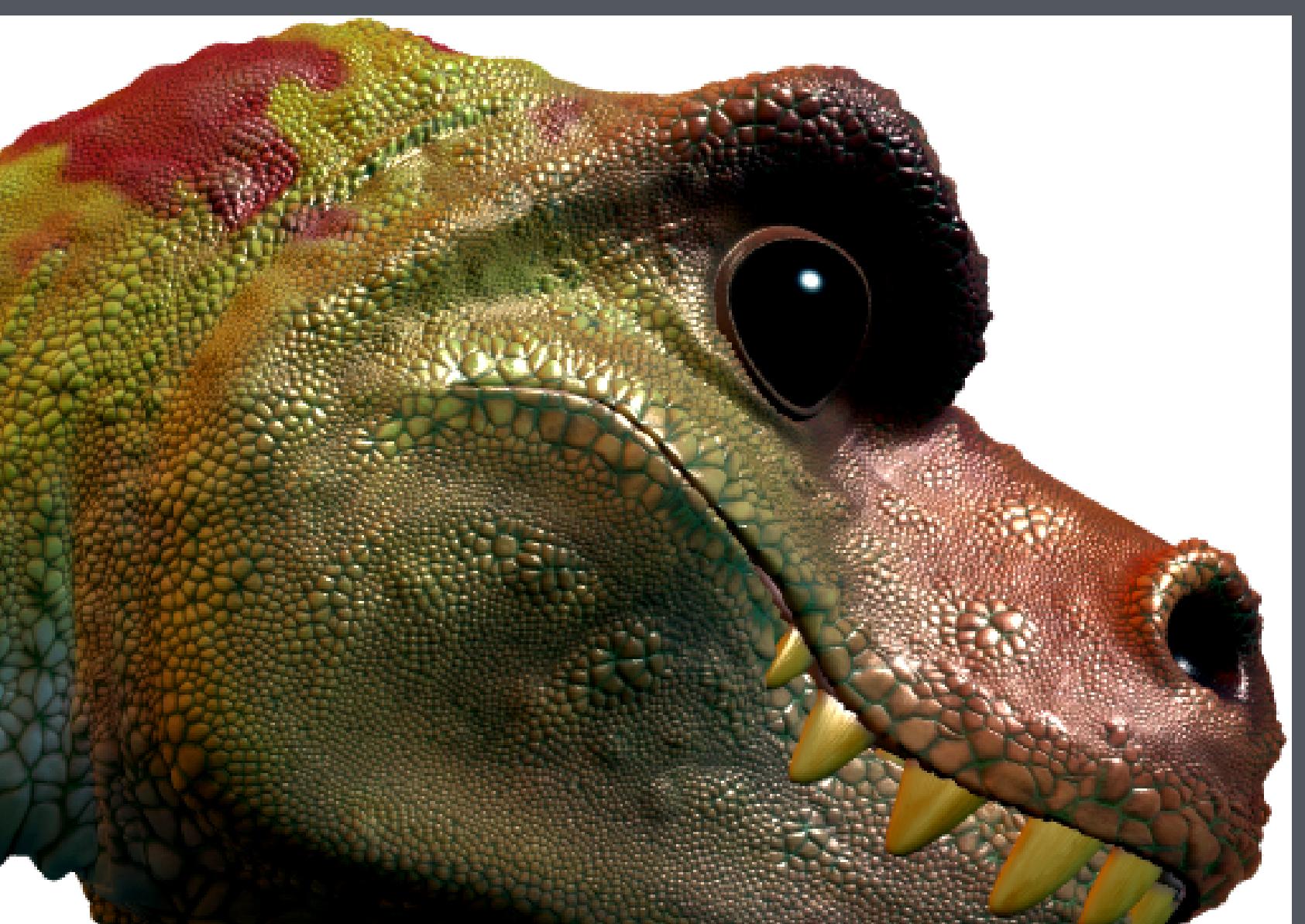
$$\begin{aligned} B &= (\tilde{b}_n.x, \tilde{b}_n.y) \\ M &= (\tilde{b}_n.x^2, \tilde{b}_n.y^2, \tilde{b}_n.x \tilde{b}_n.y) \end{aligned} \quad (\tilde{b}_n.x, \tilde{b}_n.y) = (\vec{b}_n.x / \vec{b}_n.z, \vec{b}_n.y / \vec{b}_n.z)$$

Allow the textures B and M to be filtered by the MIP map machinery,
then at shading time use an NDF defined by the mean B and the covariance:

$$\Sigma = \begin{bmatrix} M.x - B.x * B.x & M.z - B.x * B.y \\ M.z - B.x * B.y & M.y - B.y * B.y \end{bmatrix}$$



LEAN mapping [Olano & Baker I3D 2010]



LEADR mapping [Dupuy et al. SIGGRAPH 2013]