13 Texture filtering

Steve Marschner

CS5625 Spring 2022

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

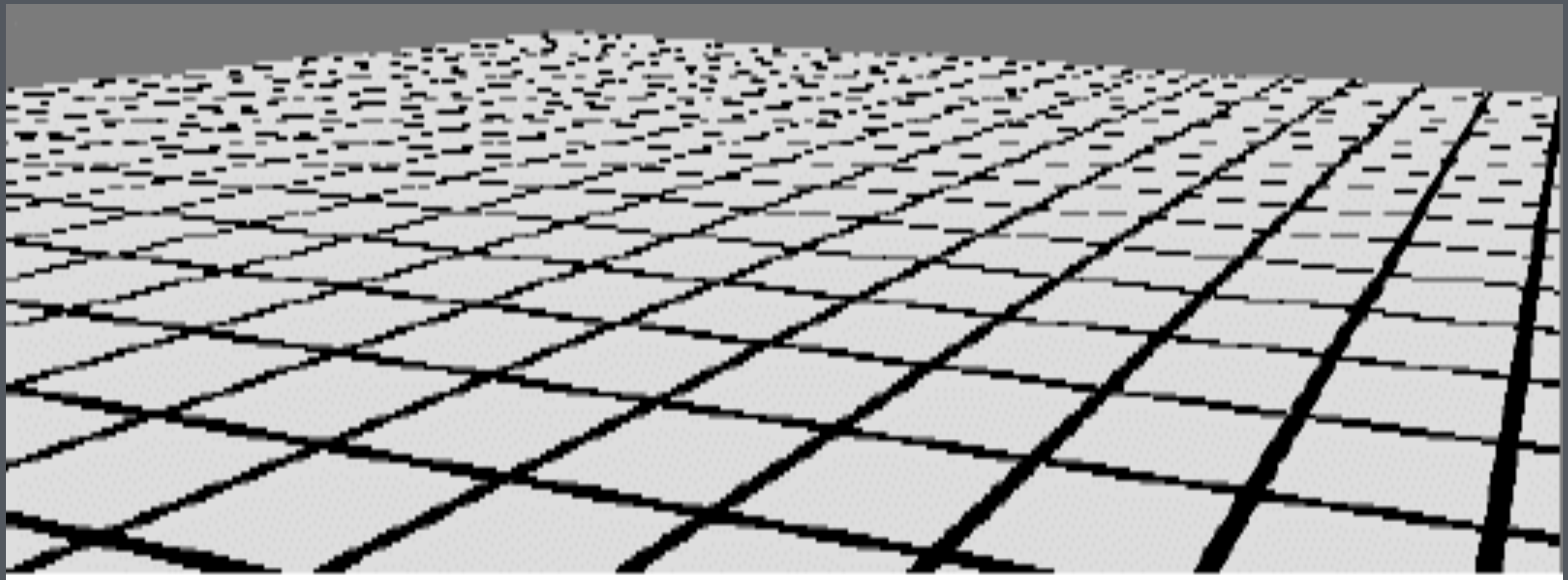
When you go far…

Solution: pixel filtering

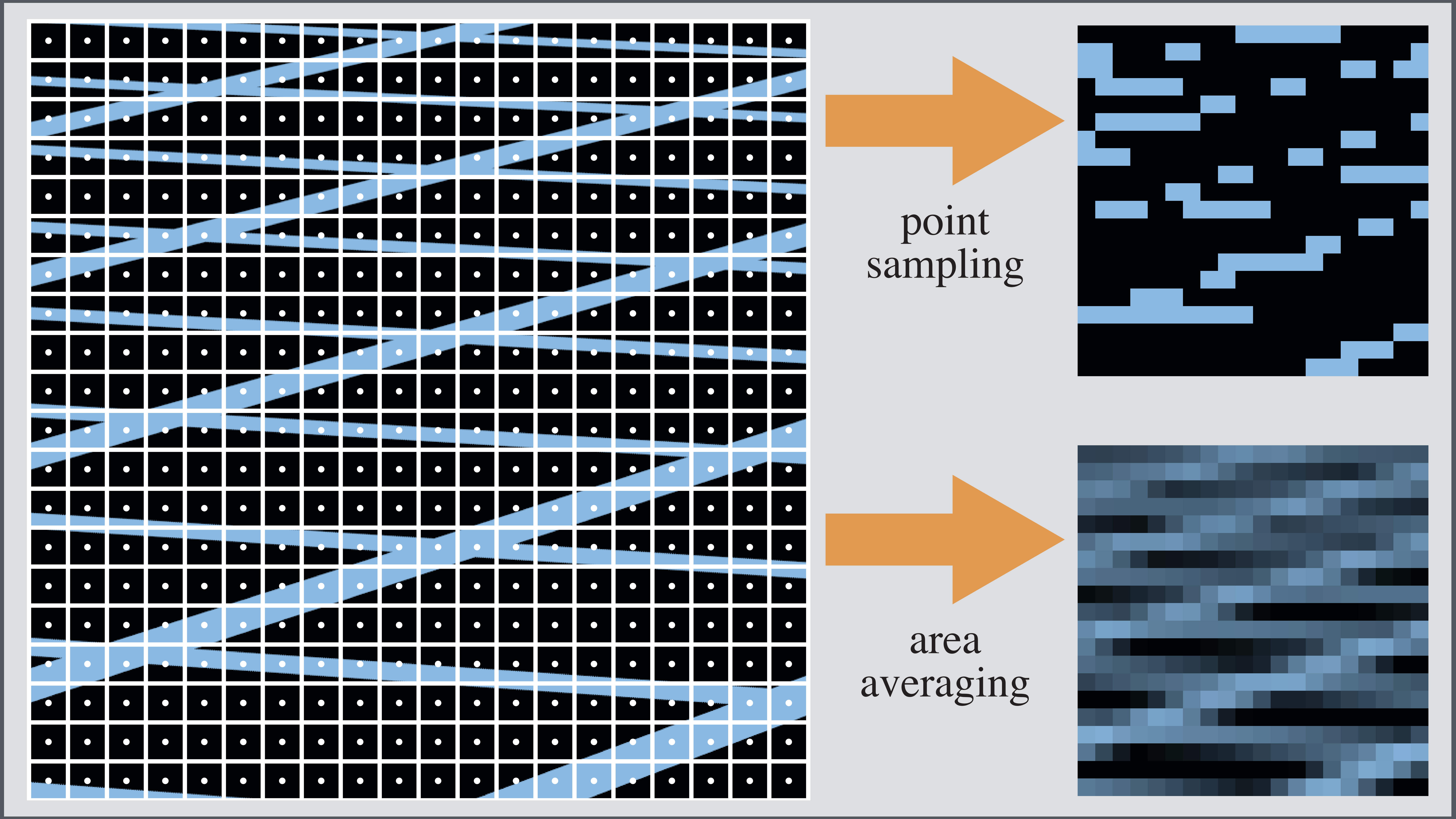
Problem: Perspective produces very high image frequencies

Solution

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



Solution: pixel filtering



point sampling

area averaging

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!

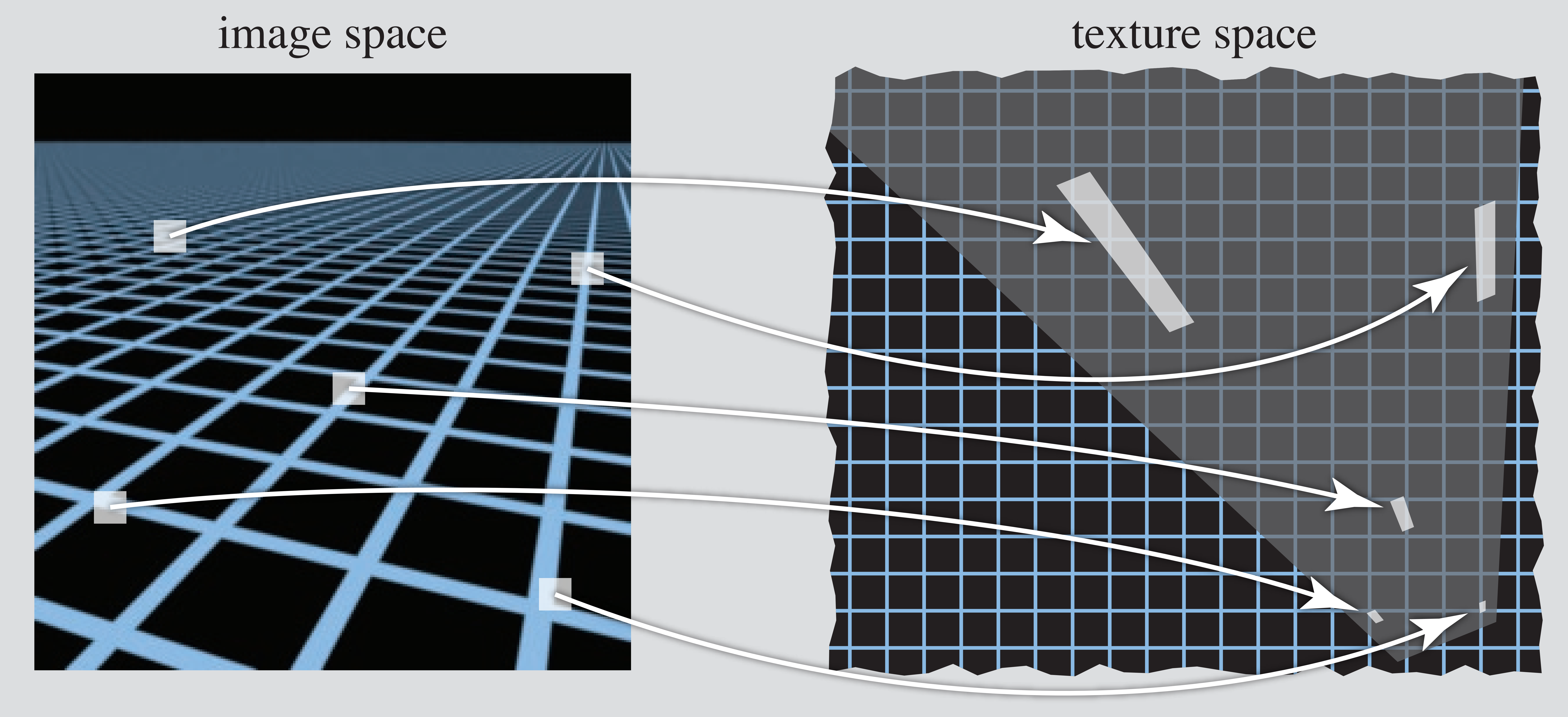
Signal is defined in texture space

- 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:

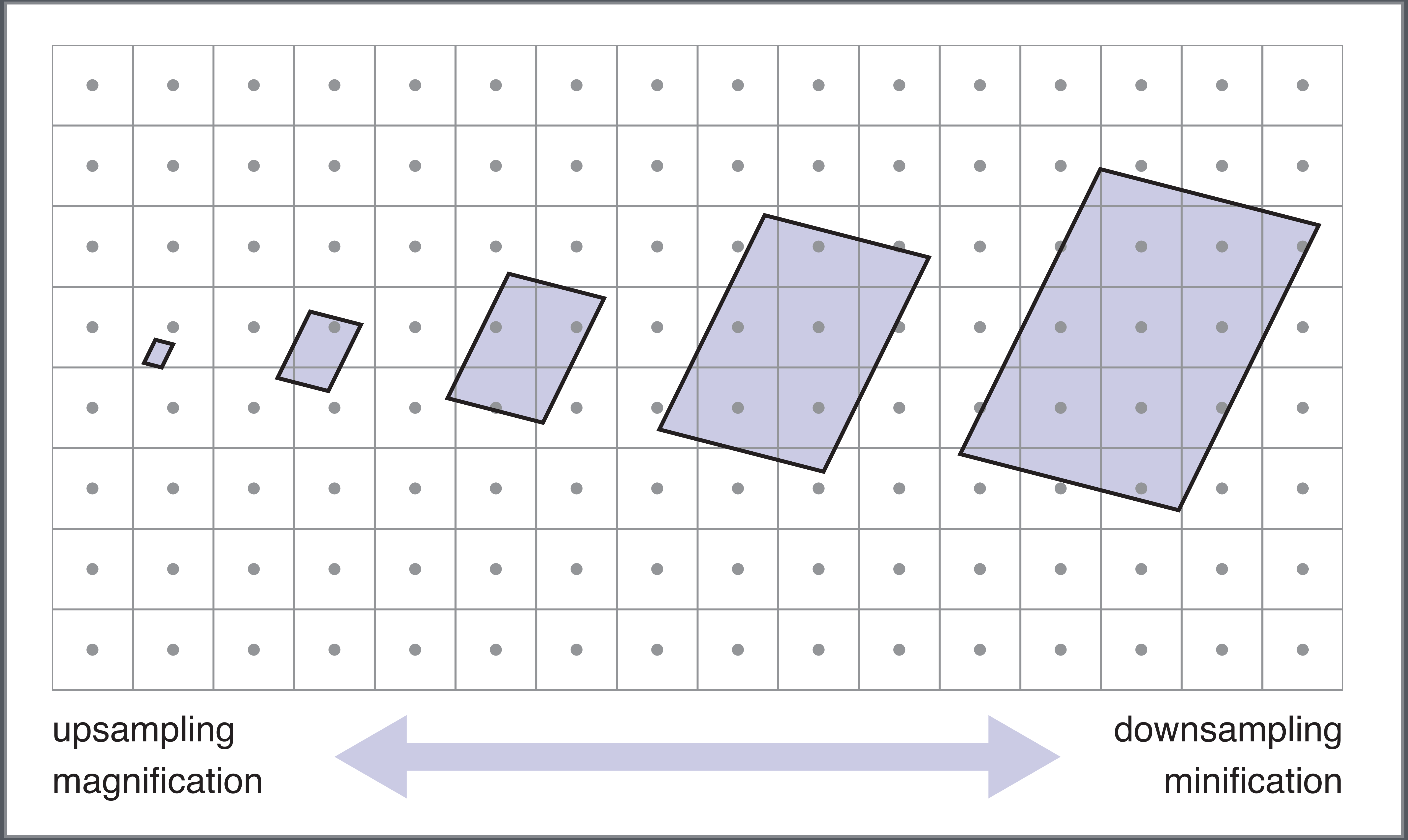
- One-to-one mapping between pixel area and texel area

When closer

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

When farther

- Each pixel could include many texels
- “minification”
- averaging is needed



How to get a handle on pixel footprint

We have a nonlinear mapping to deal with

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

Instead use a local linear approximation

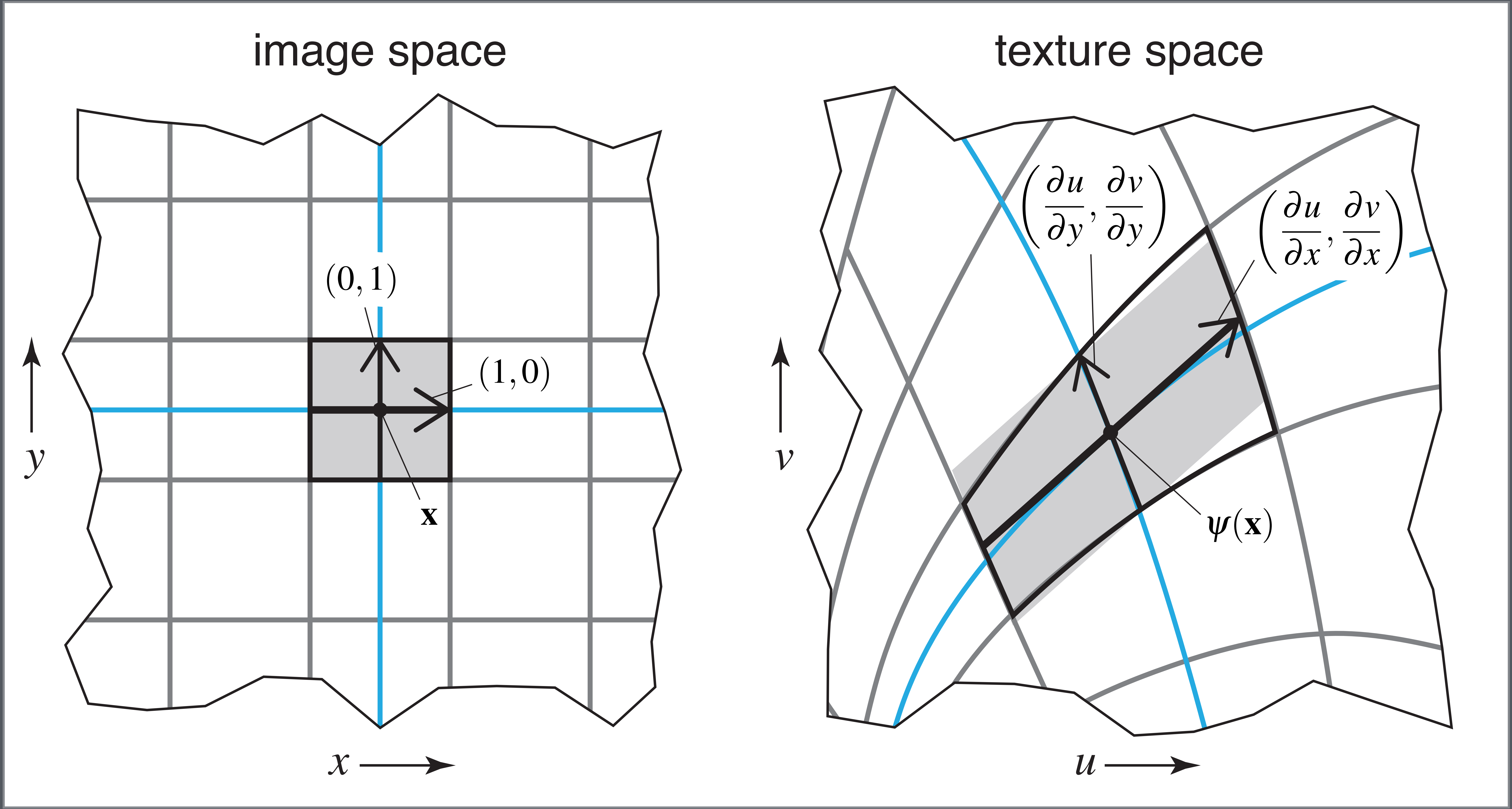
- hinges on the derivative of **u** = (u,v) wrt. **x** = (x,y)

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

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

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

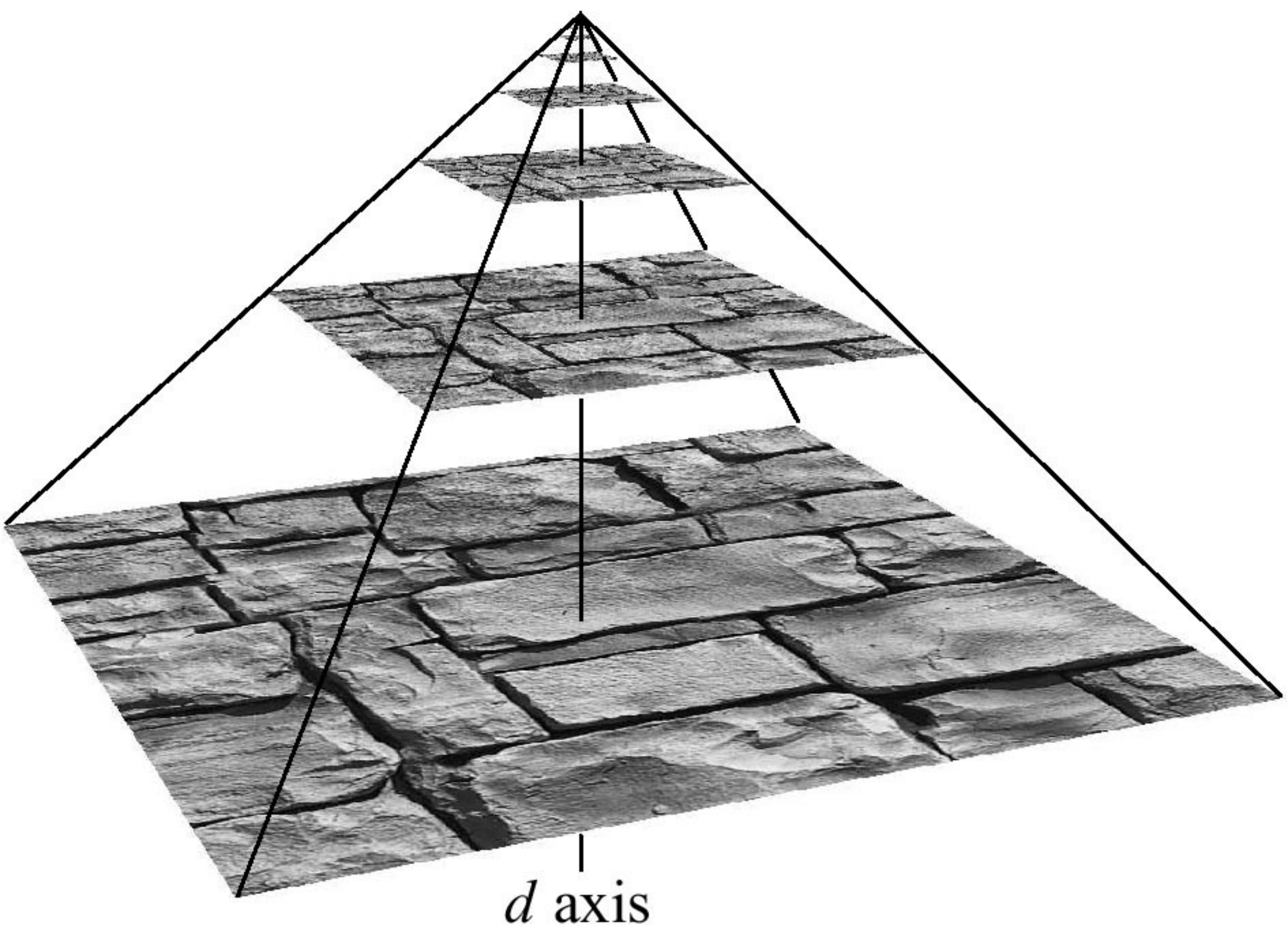
MIP Maps

- 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



Given derivatives: what is level?

Need to reduce the matrix to a single number

- aka. choosing a matrix norm; several choices available with different tradeoffs
- elementwise max partial derivative:

$$l = \log\left[\max\left(\left|\frac{\partial u}{\partial x}\right|, \left|\frac{\partial v}{\partial x}\right|, \left|\frac{\partial u}{\partial y}\right|, \left|\frac{\partial v}{\partial y}\right|\right)\right]$$

- root-mean-square of partial derivatives:

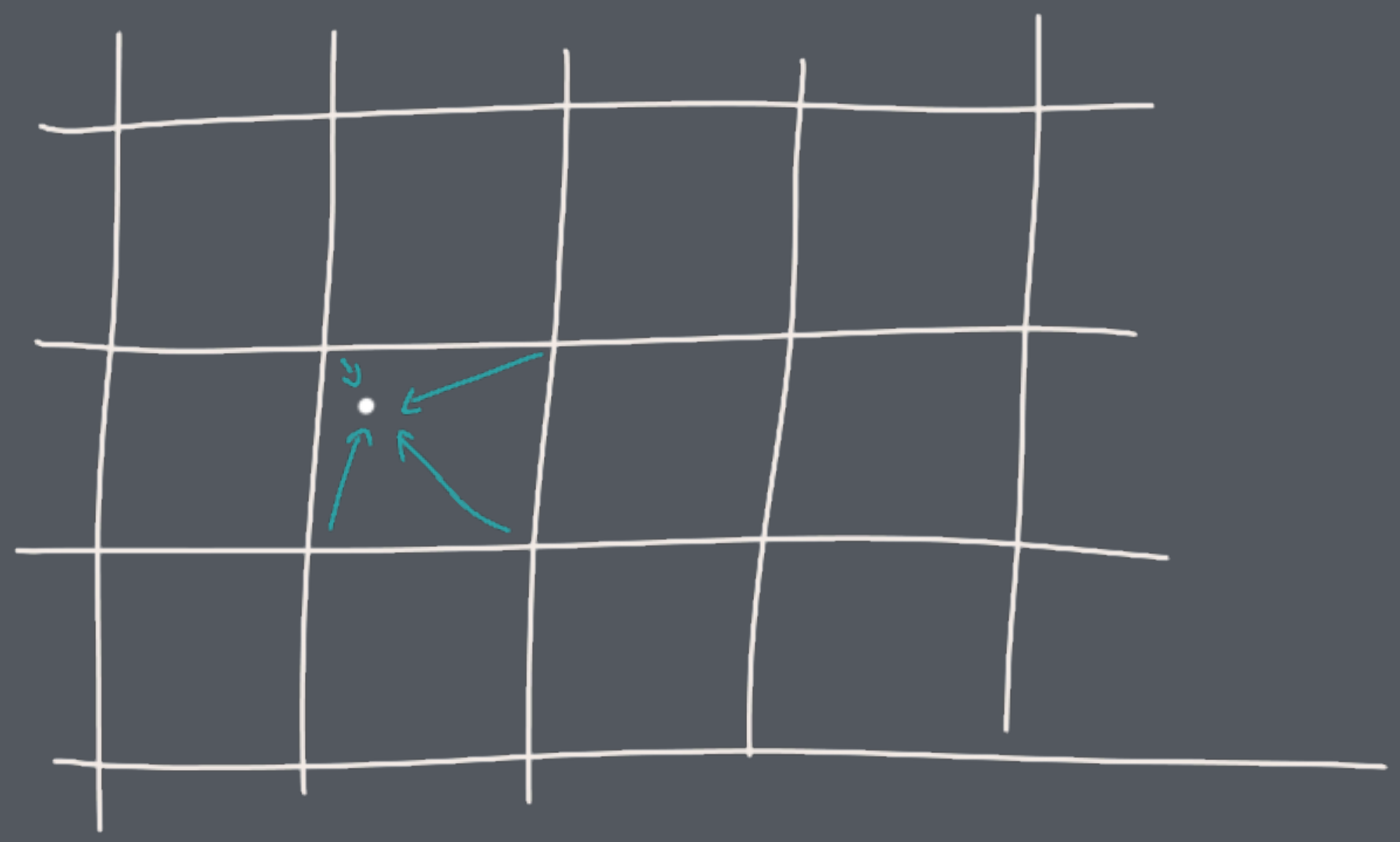
$$l = \log\sqrt{\left(\frac{\partial u}{\partial x}\right)^2 + \left(\frac{\partial v}{\partial x}\right)^2 + \left(\frac{\partial u}{\partial y}\right)^2 + \left(\frac{\partial v}{\partial y}\right)^2}$$

- either way, you get a non-integer level at which to look up

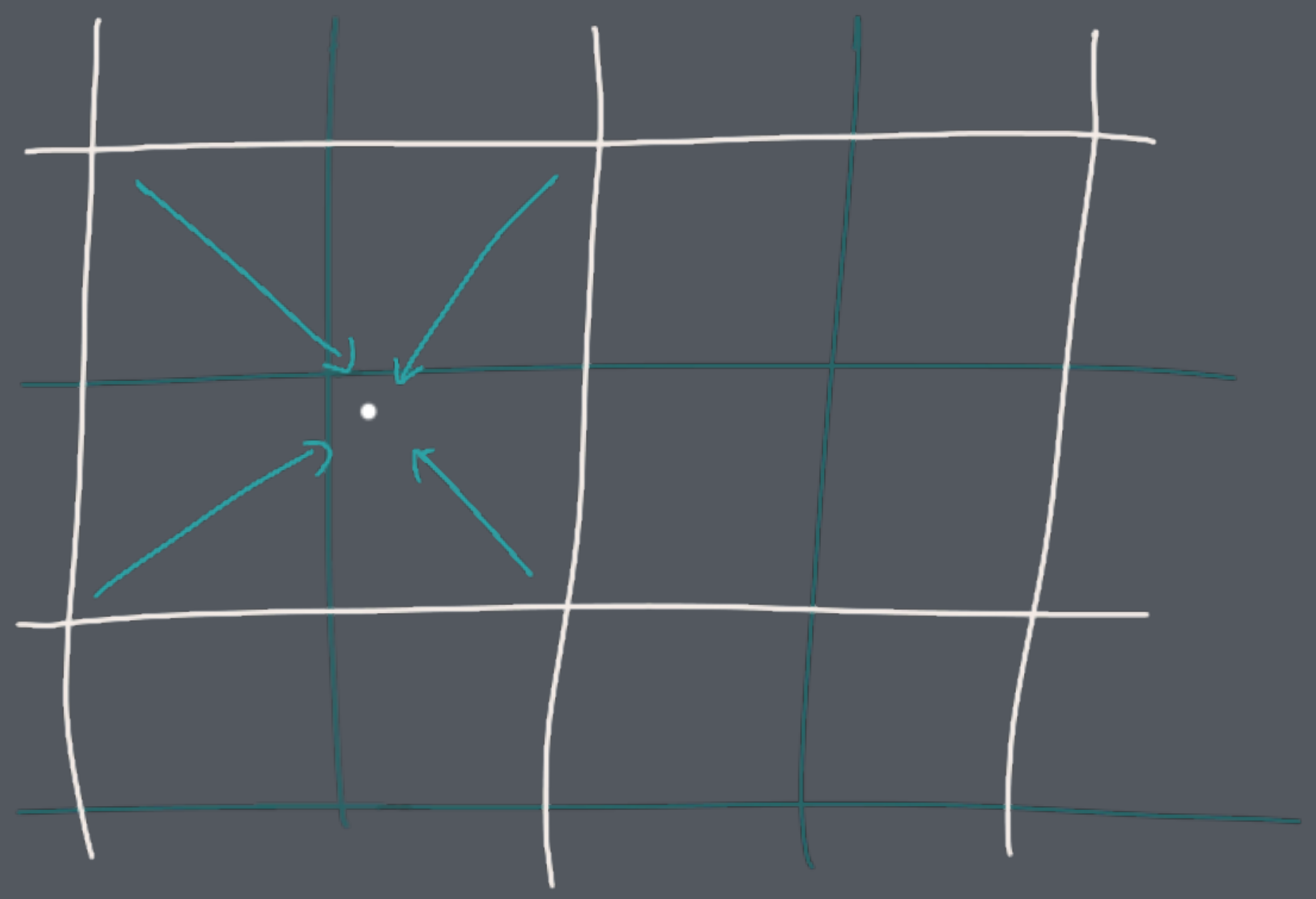
Using the MIP Map

In level, find texel and

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



Level *k*

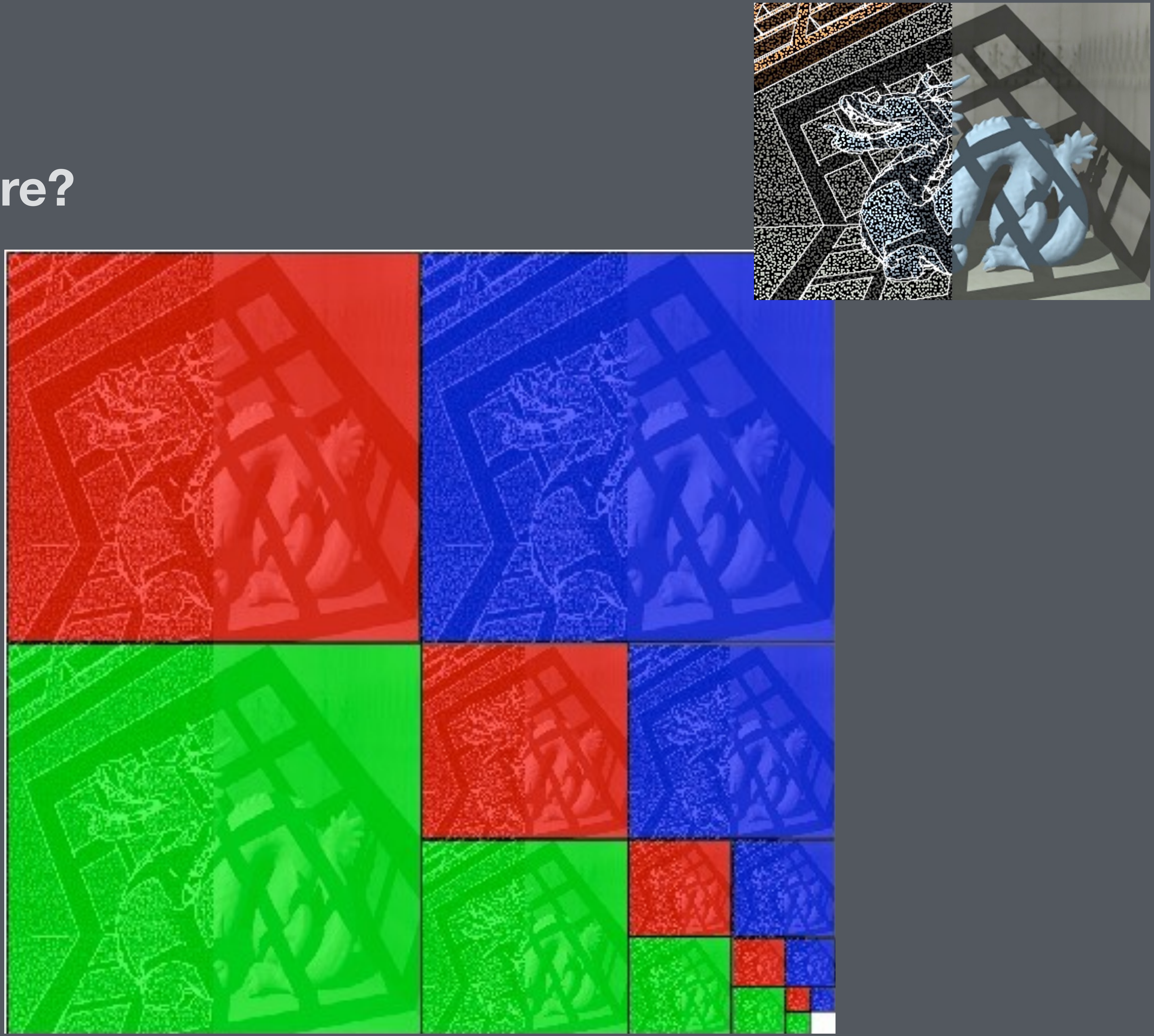


Level (*k* + 1)

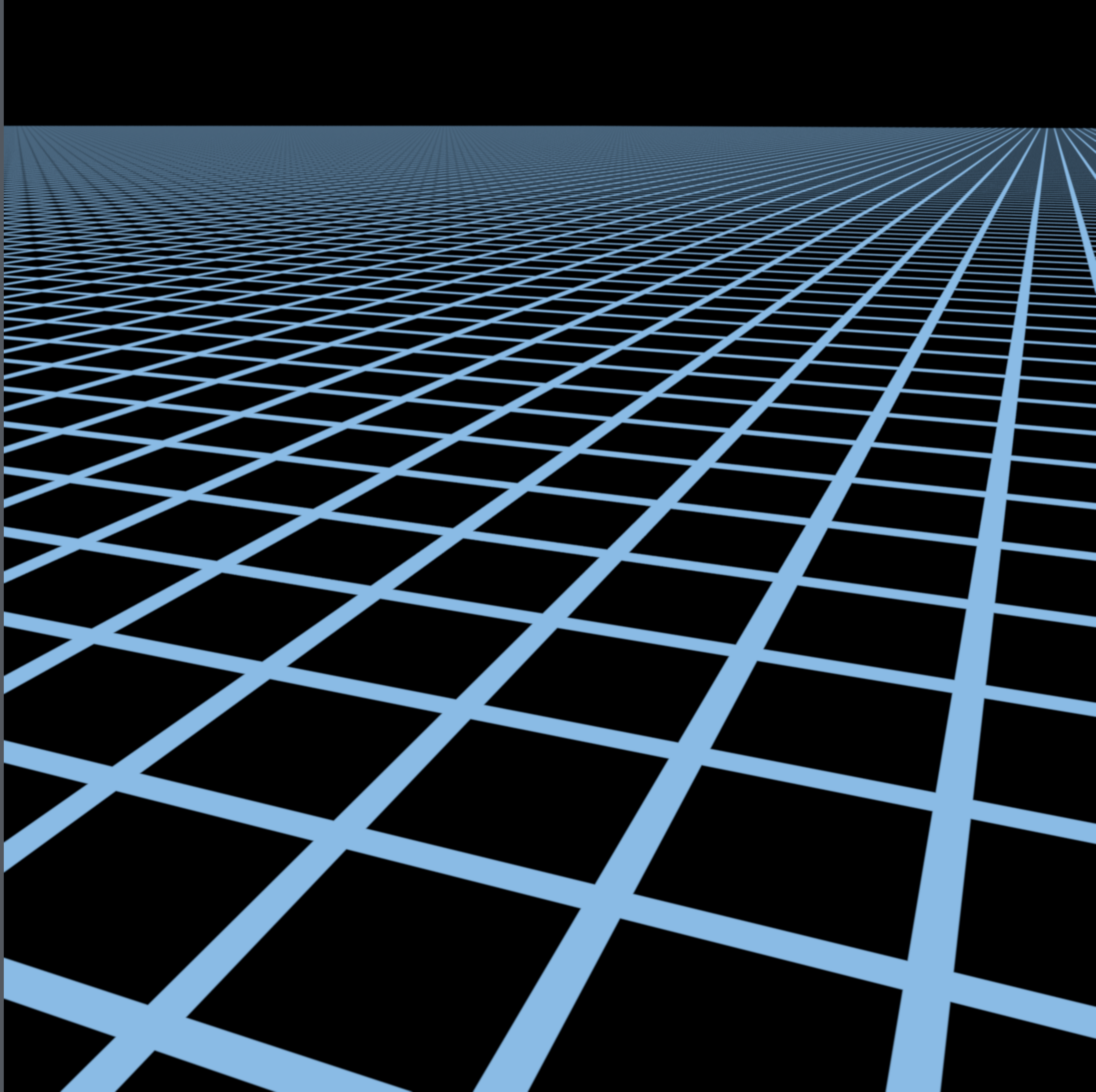
Memory Usage

What happens to size of texture?

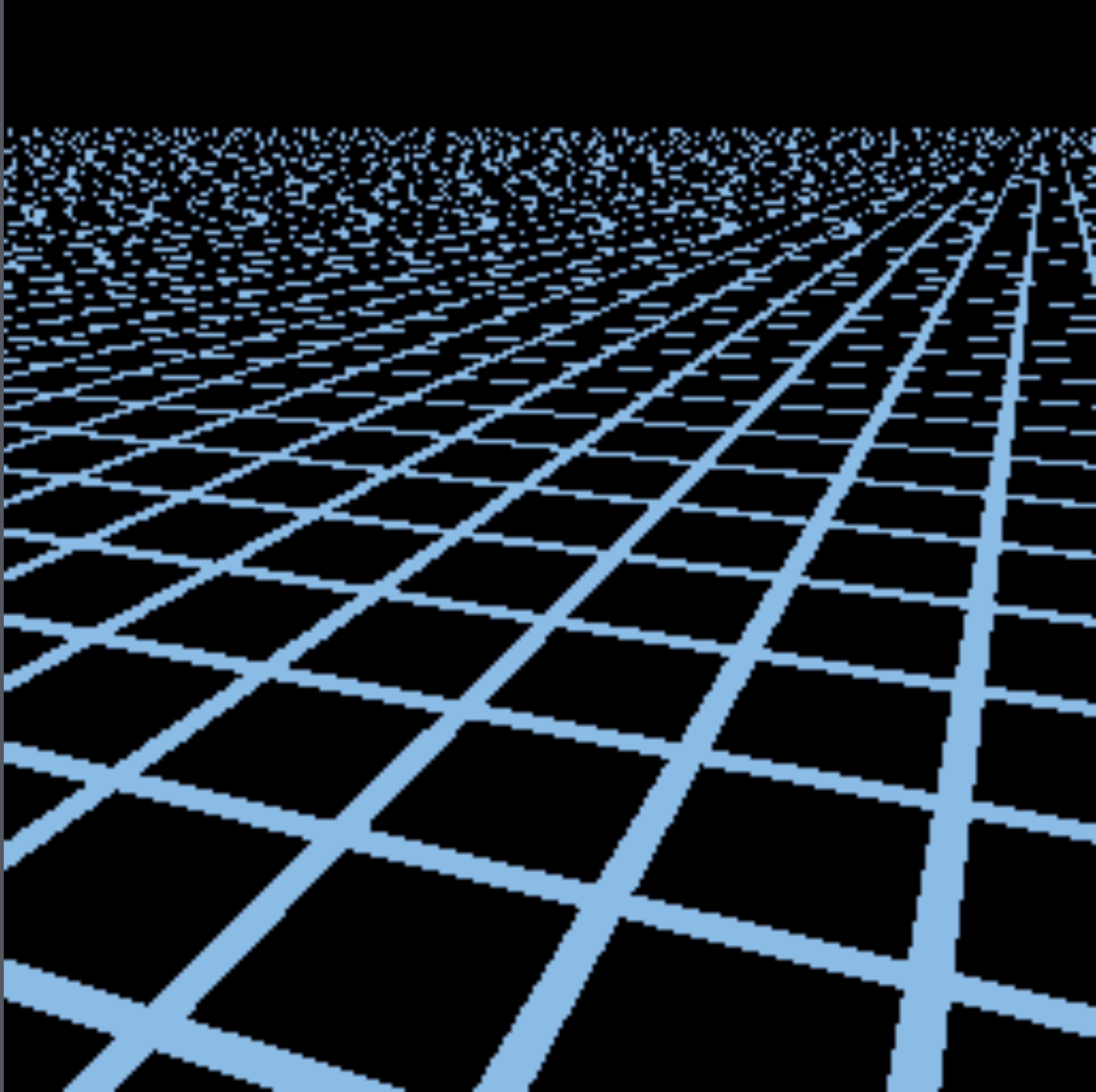
- level 1 takes 1/4 the memory of level 0
- level 2 takes 1/16, etc.
- in total, adds 1/3 to the storage requirements



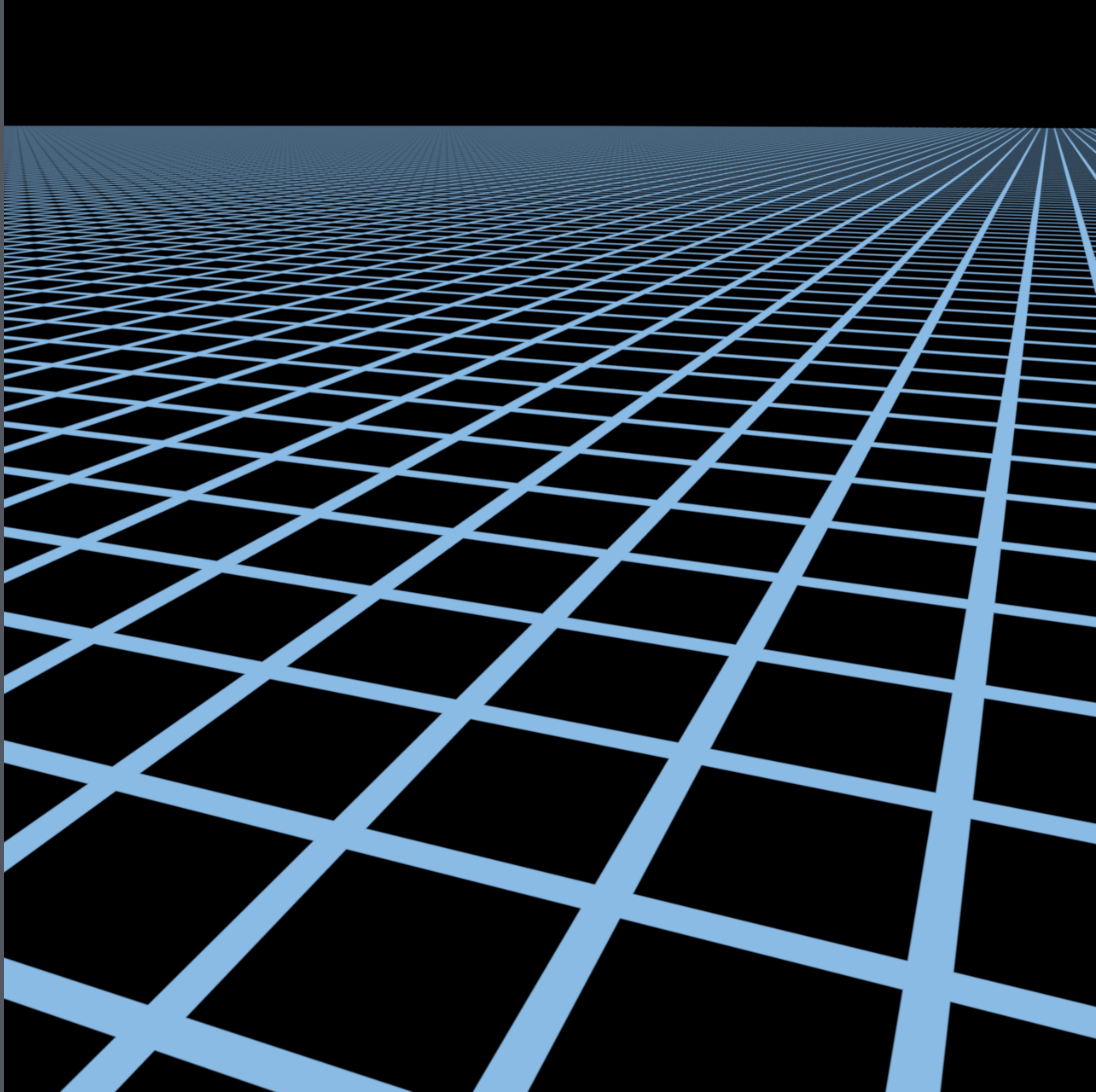
Point sampling



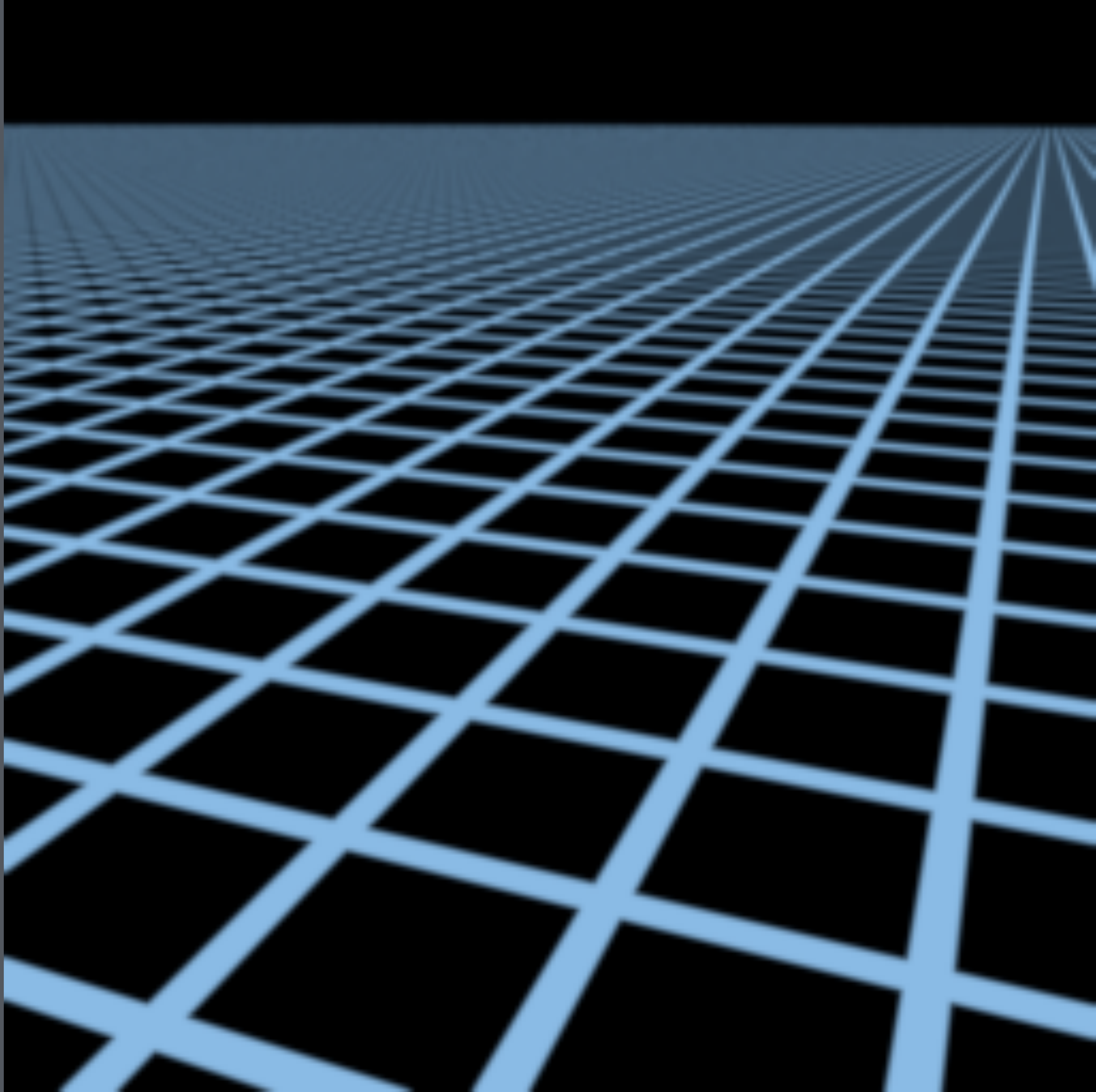
Point sampling



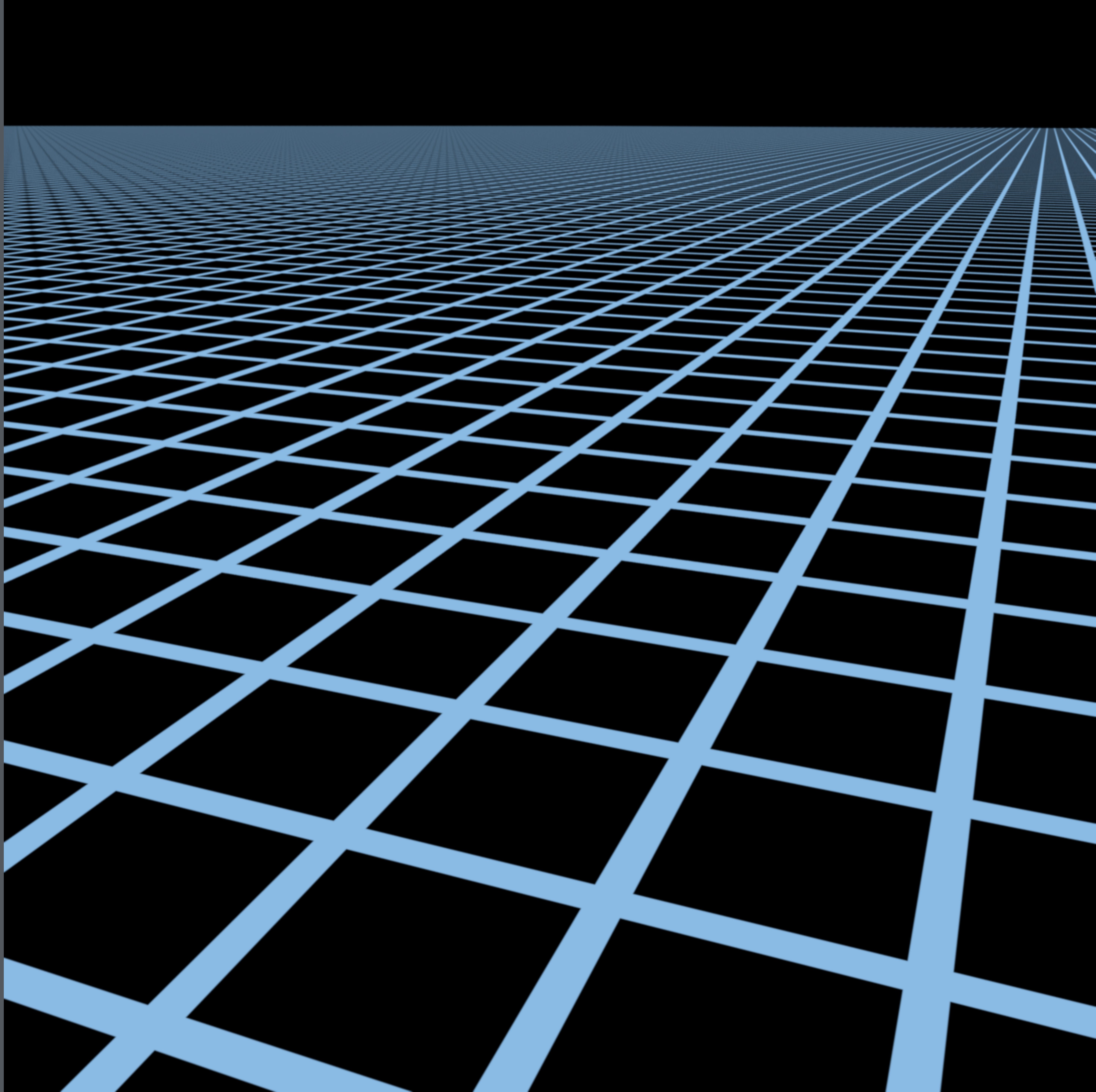
Reference: gaussian sampling by 512x supersampling



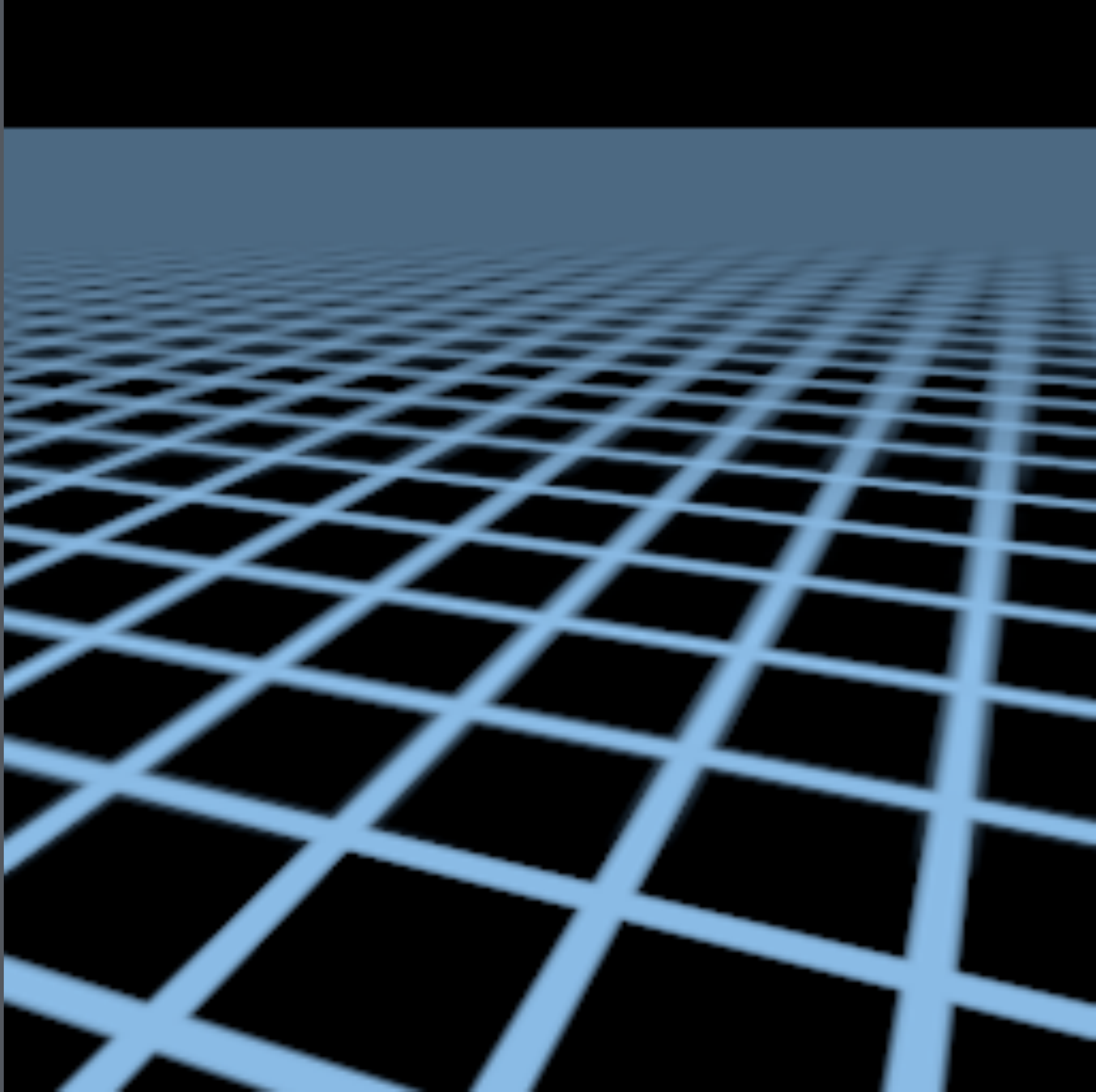
Reference: gaussian sampling by 512x supersampling



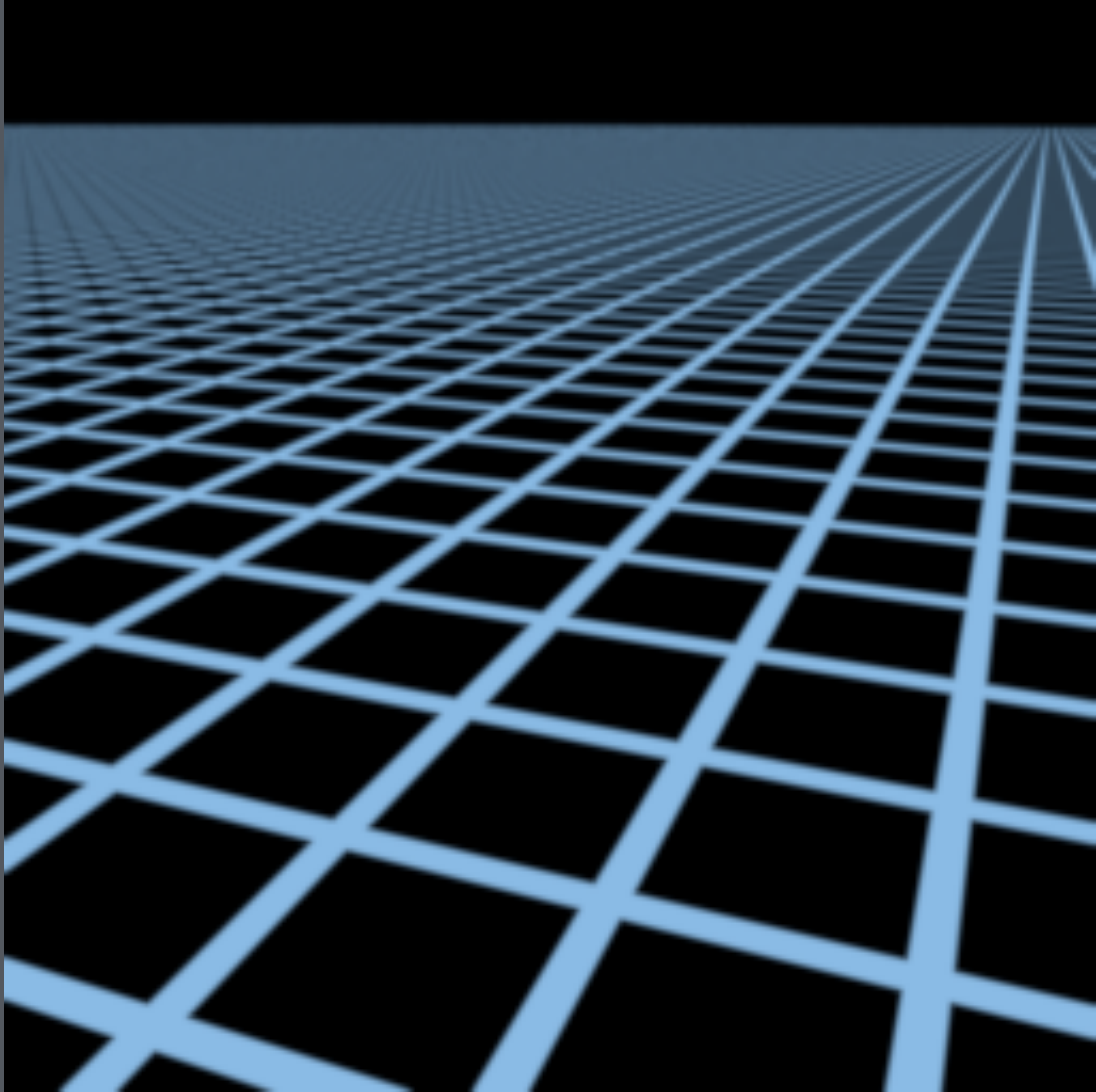
Texture minification with a mipmap



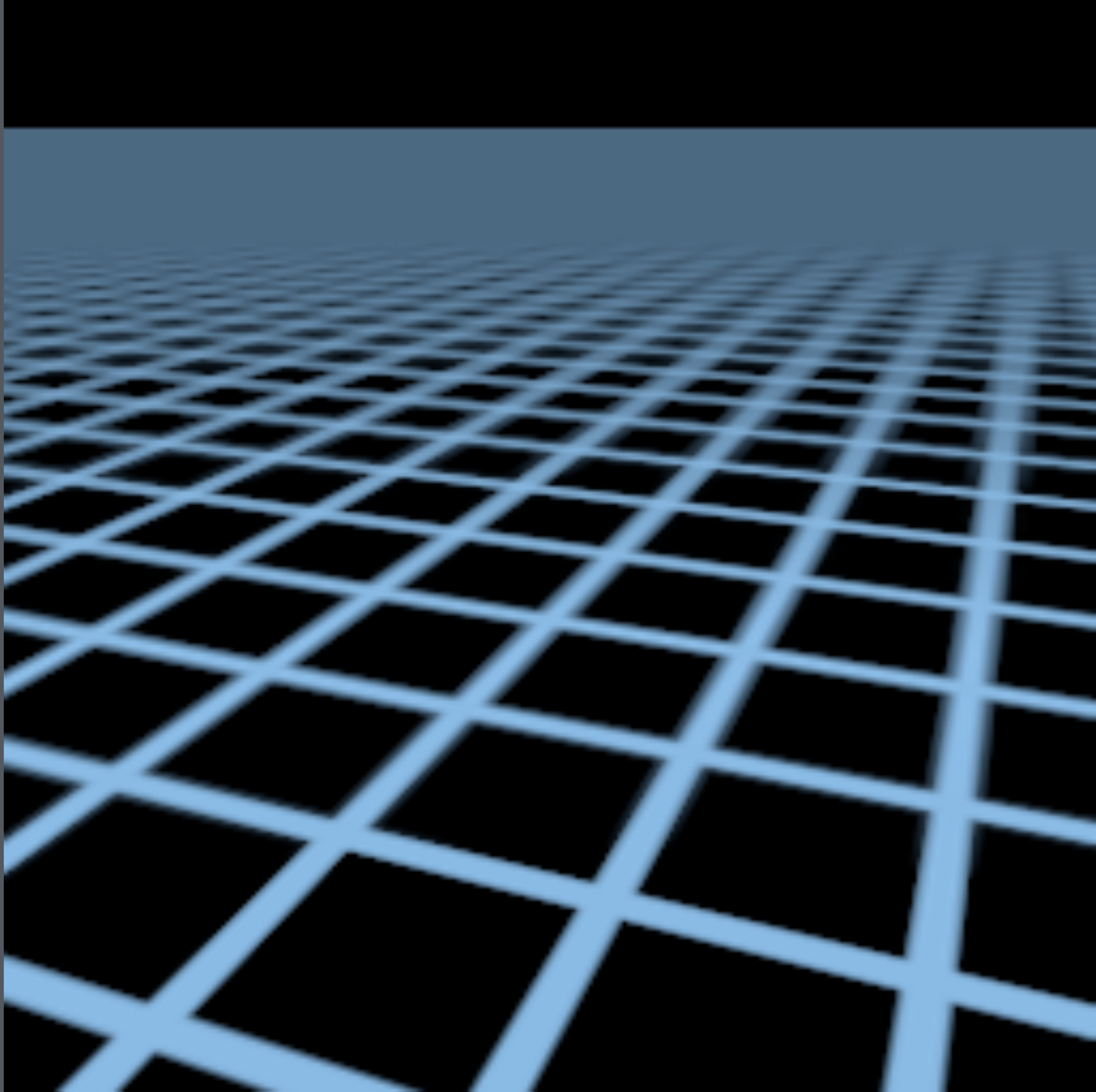
Texture minification with a mipmap



Texture minification: supersampling vs. mipmap



Texture minification: supersampling vs. mipmap



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

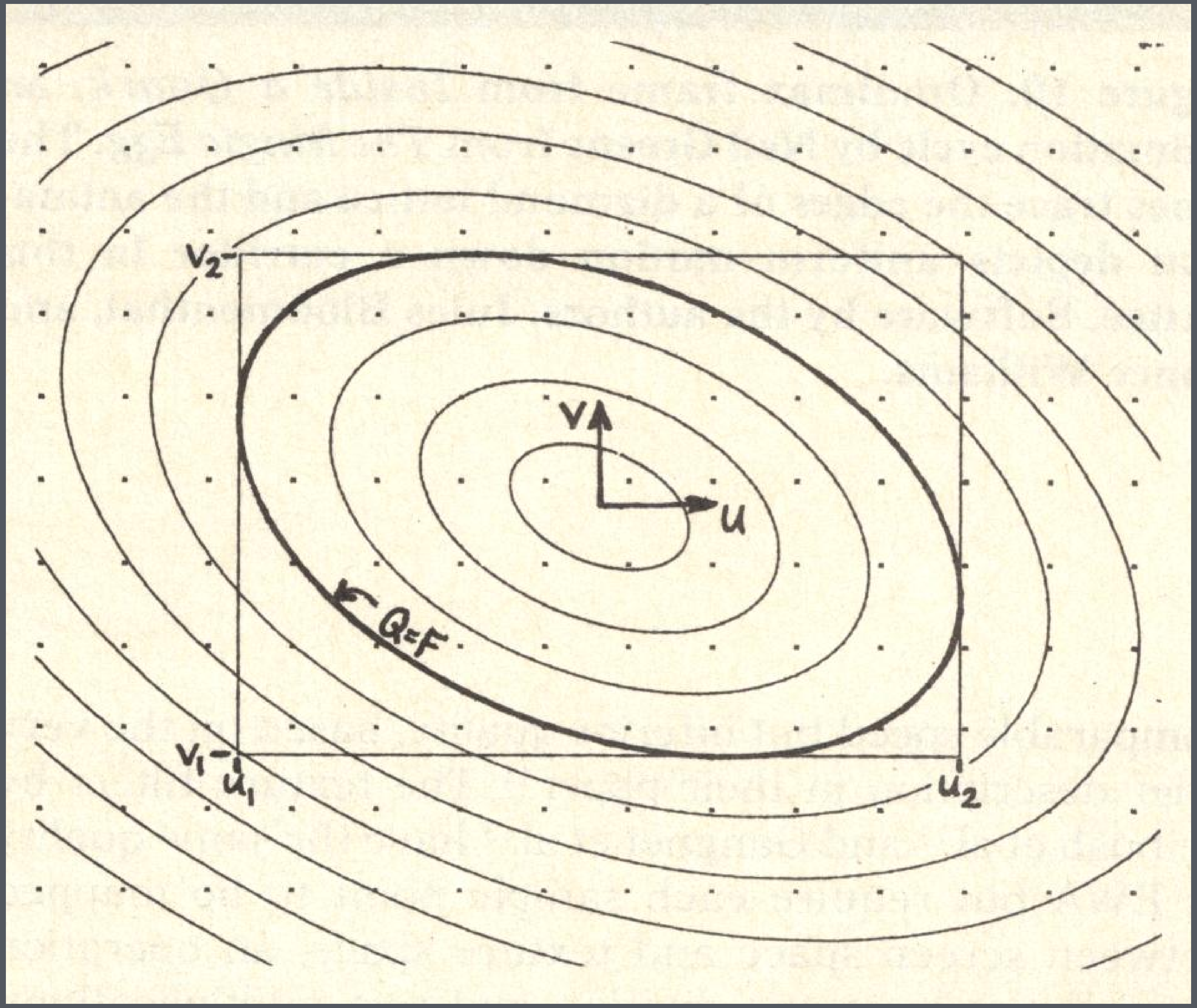
- circular pixel maps to elliptical footprint
- ellipse dimensions calc'd from quadratic

Loop over texels inside ellipse

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

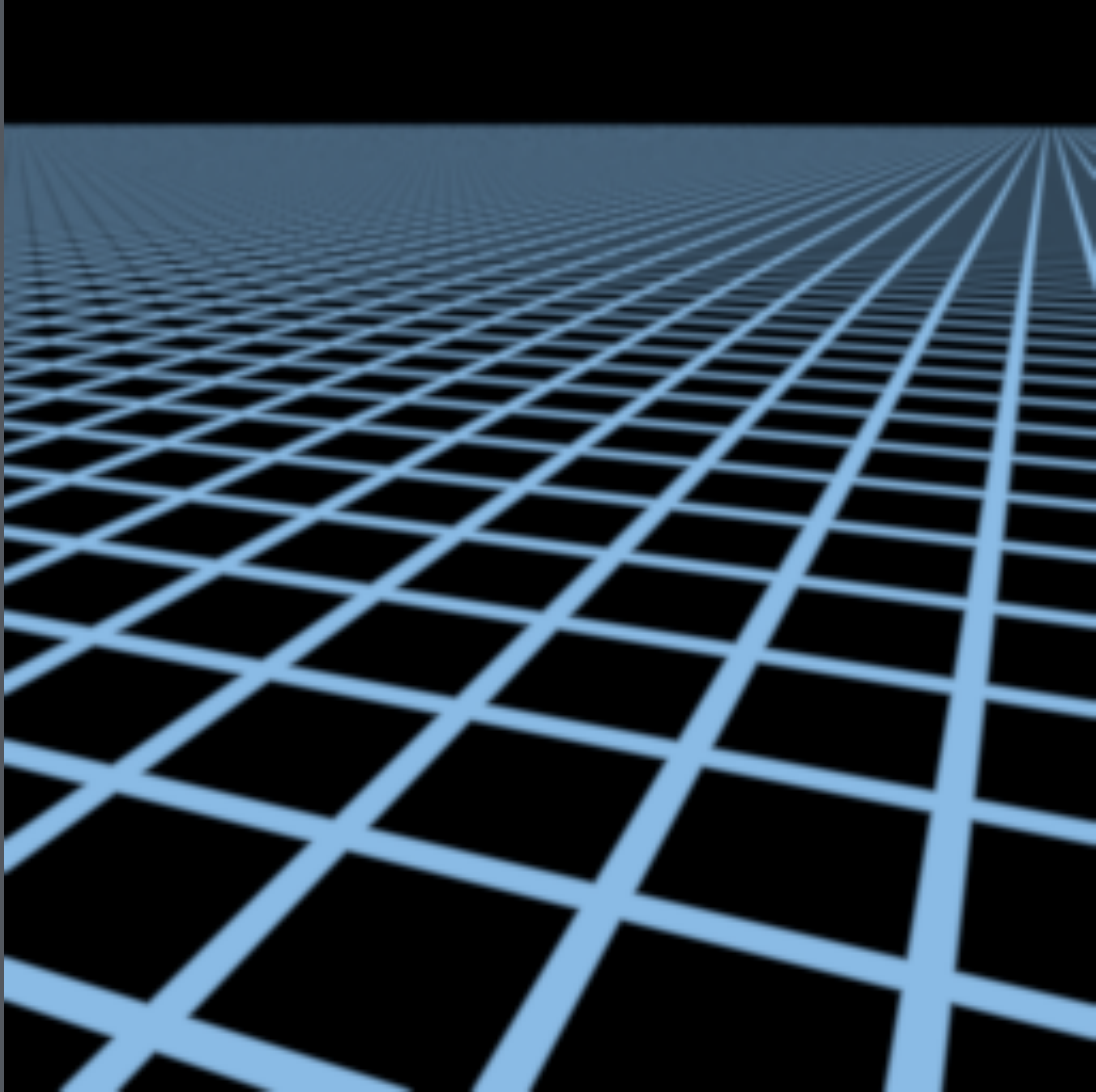
Select appropriate MIP map level

- so that minor radius is 1–2 texels

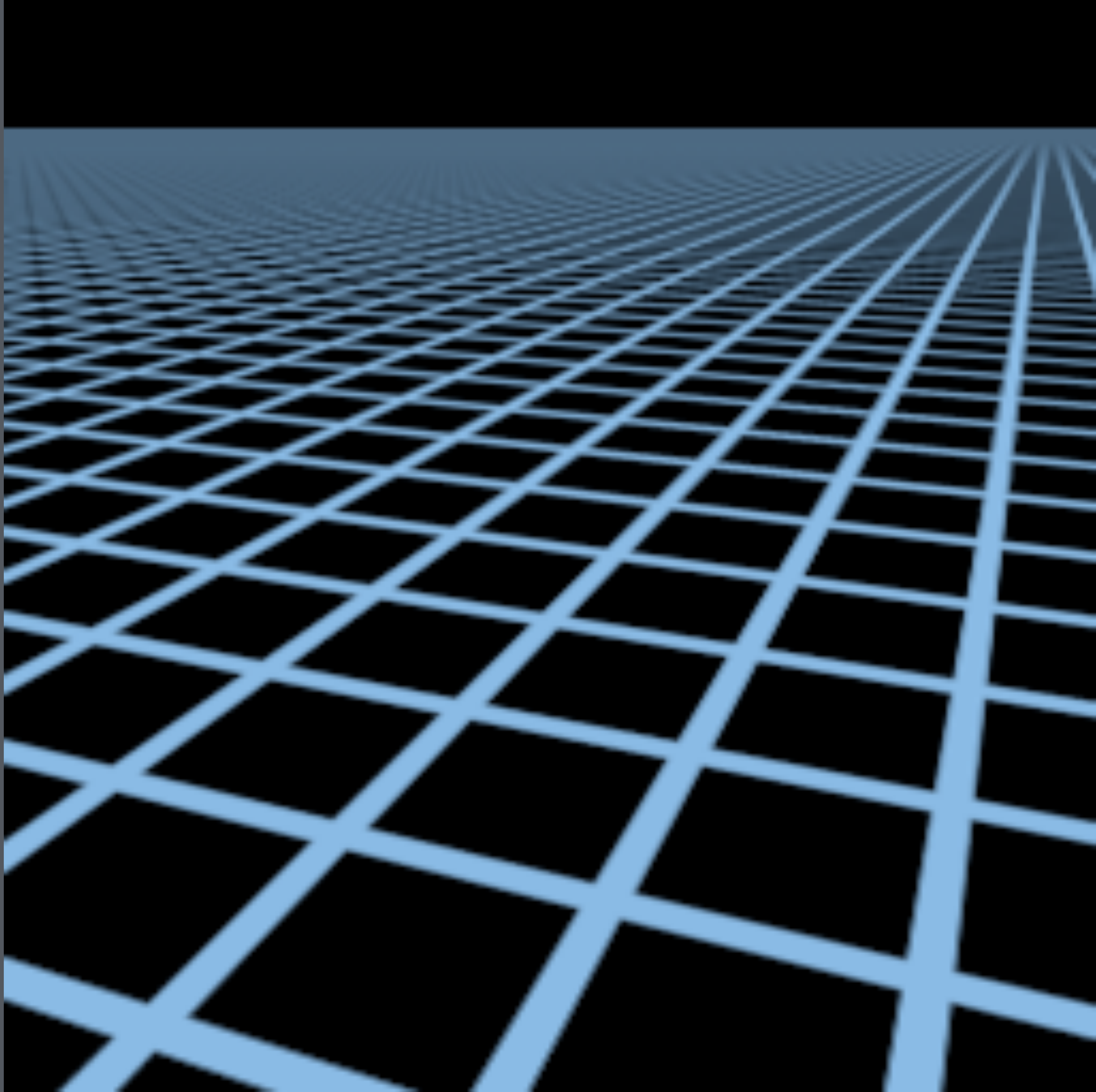


Greene & Heckbert '86

Texture minification: supersampled vs. EWA



Texture minification: supersampled vs. EWA



Simpler anisotropic MIP mapping

EWA requires a lot of lookups for diagonally oriented footprints

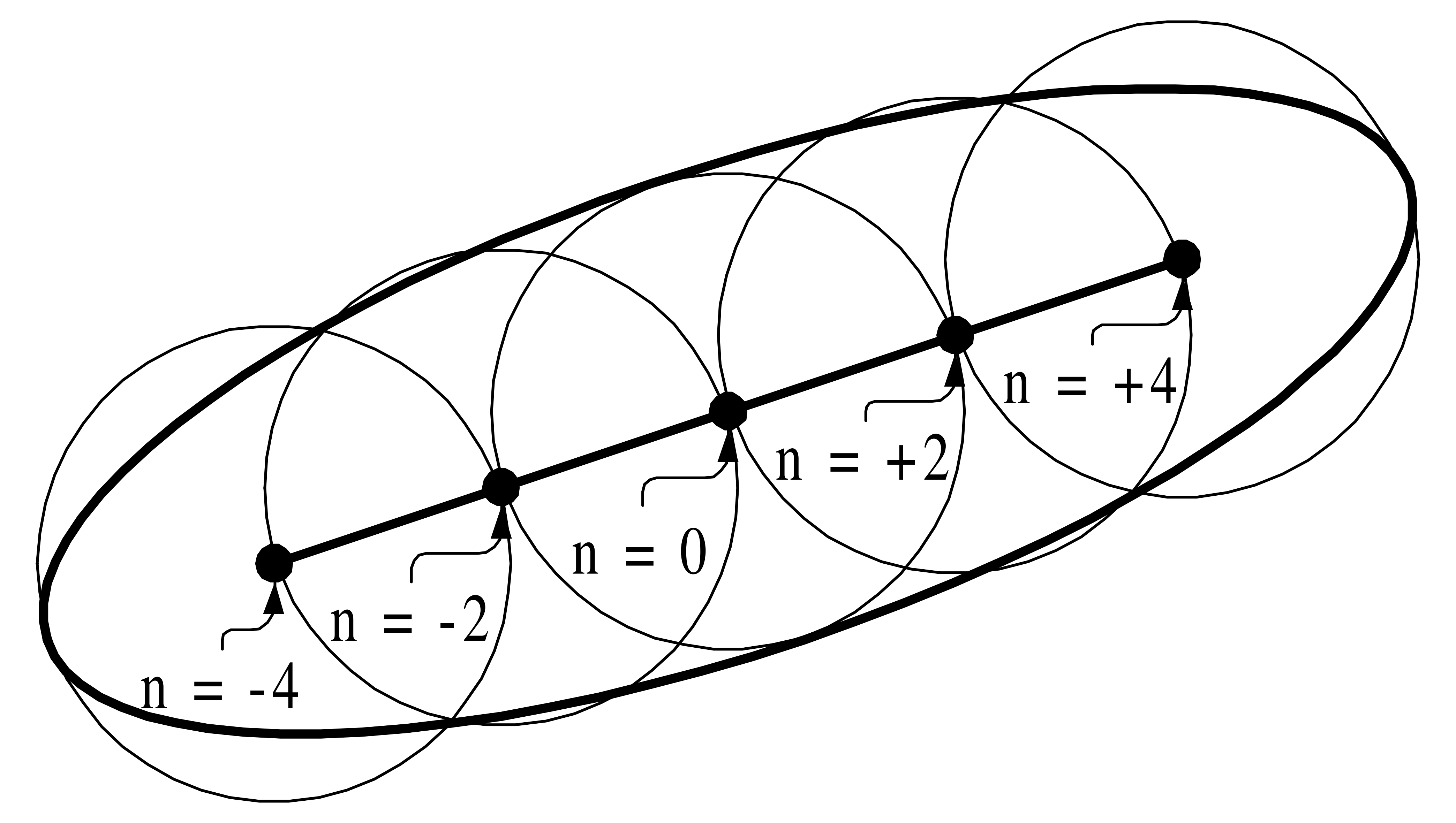
Instead, approximate your footprint as a single line of blobs

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

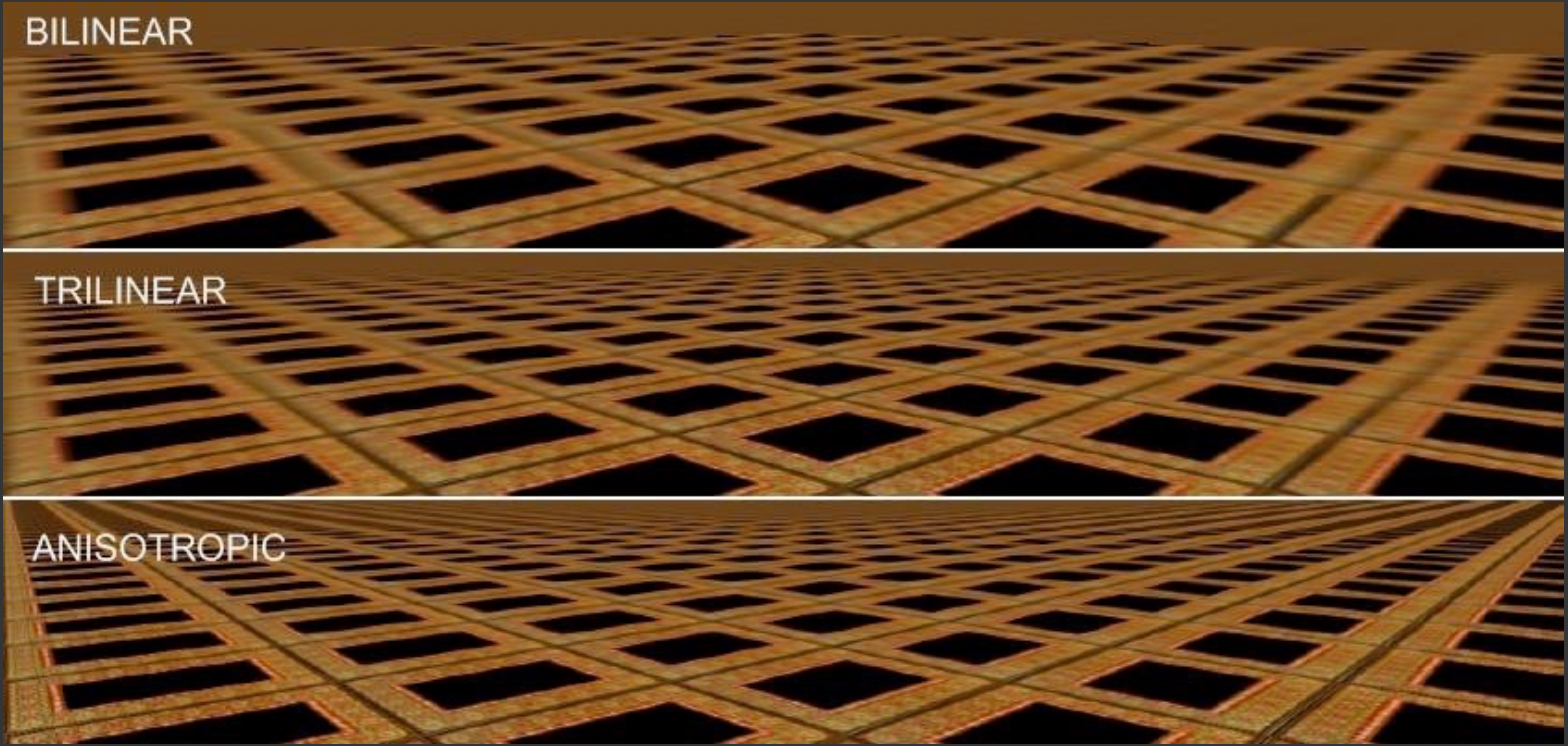
Number of samples proportional to major:minor axis ratio

- 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



slide courtesy of Kavita Bala, Cornell University

Filtering normal maps

Normal (or bump) maps can produce aliasing too

- shiny surface => color very sensitive to normal
- normal swings around faster as camera moves away => high contrast, high detail image

Filtering the normal map does the wrong thing

- shiny, bumpy surface at a distance becomes a shiny smooth surface
- microfacet theory tells us the non-resolved bumps produce a rough surface appearance

Normal map filtering is about producing appropriate BRDF at large scales

- bumps filtered away, replaced by roughness
- surfaces can become anisotropic depending on normal map content

LEAN Mapping

Linear **E**fficient **A**nisotropic **N**ormal **Mapping**

A practical and efficient normal map antialiasing approch

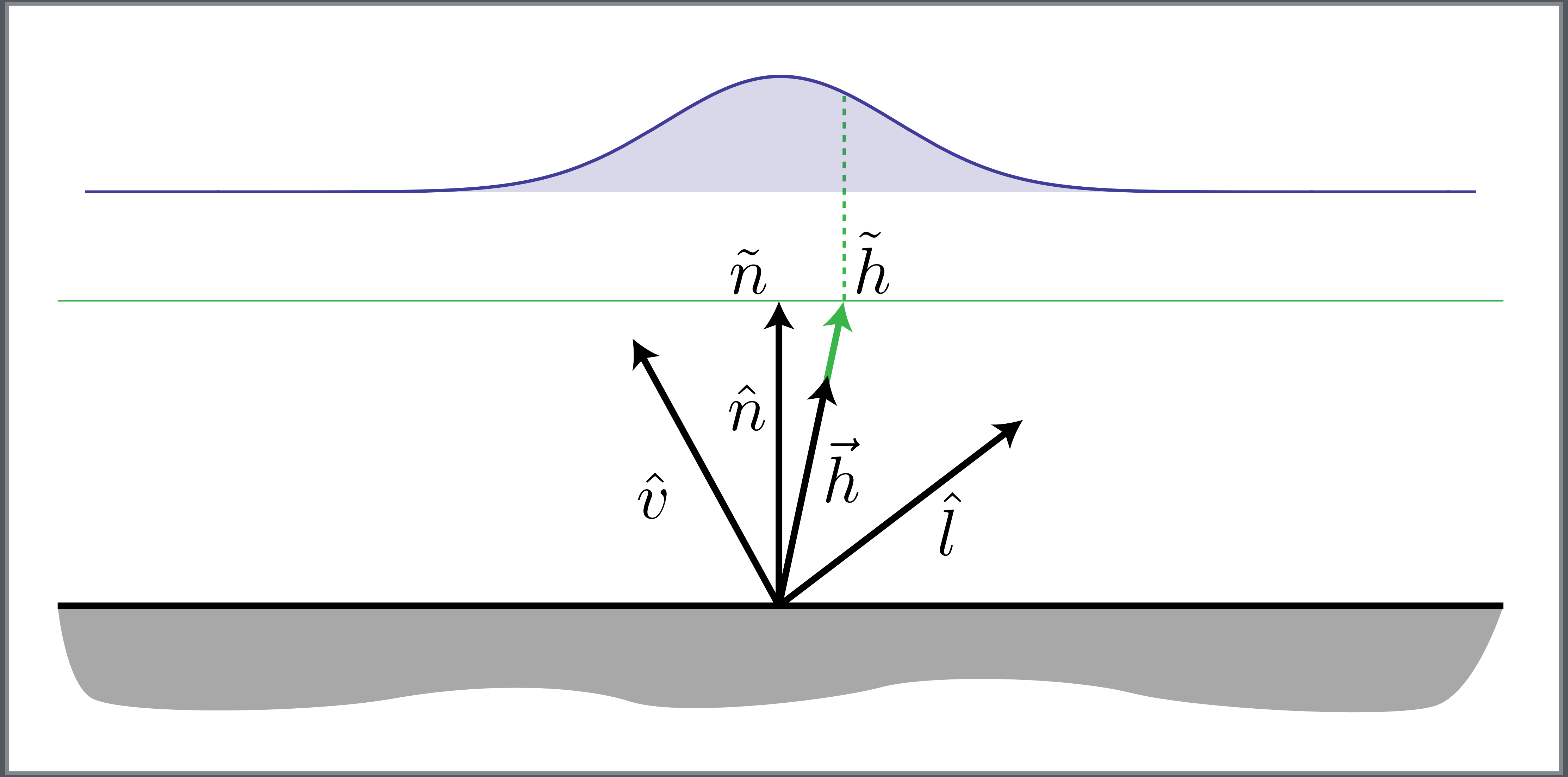
Key ideas:

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

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

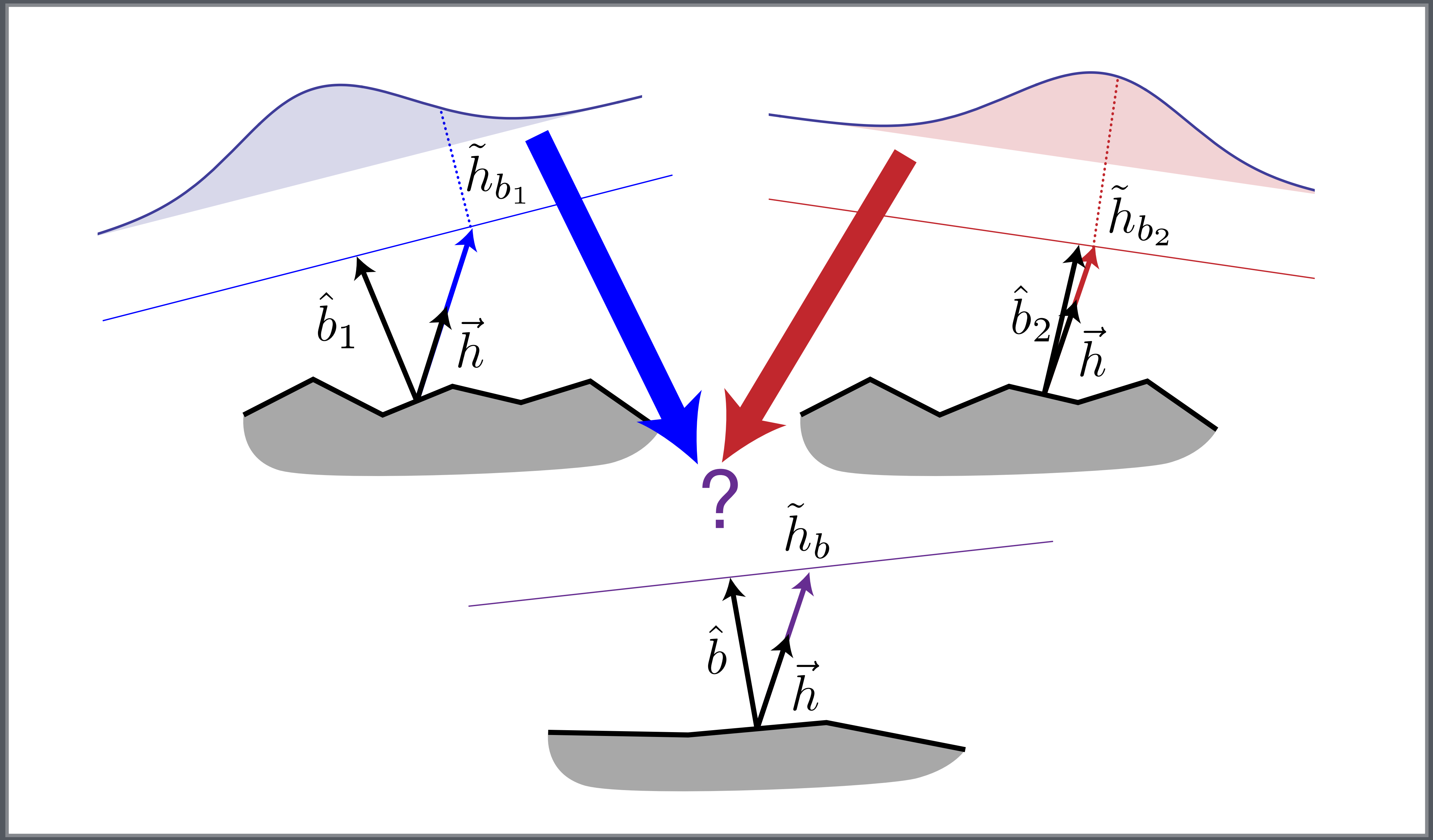
- Use Gaussians for the NDFs
- Approximate the sum of multiple Gaussians by adding the first and second moments

LEAN Mapping

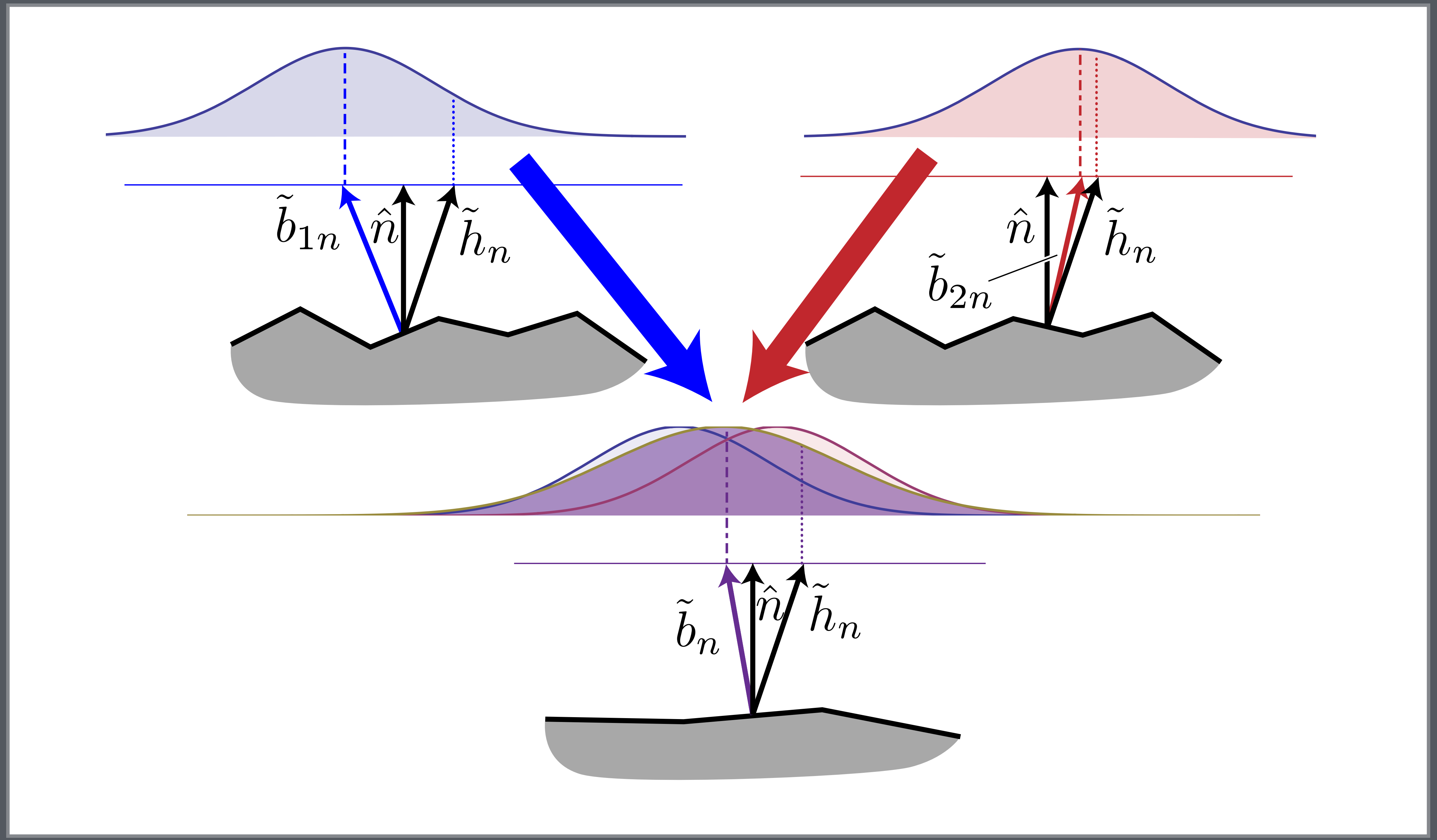


an NDF in tangent-vector space

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:

$$B = (\tilde{b}_n.x,\ \tilde{b}_n.y)$$

$$M = (\tilde{b}_n.x^2,\ \tilde{b}_n.y^2,\ \tilde{b}_n.x\,\tilde{b}_n.y)$$

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

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]

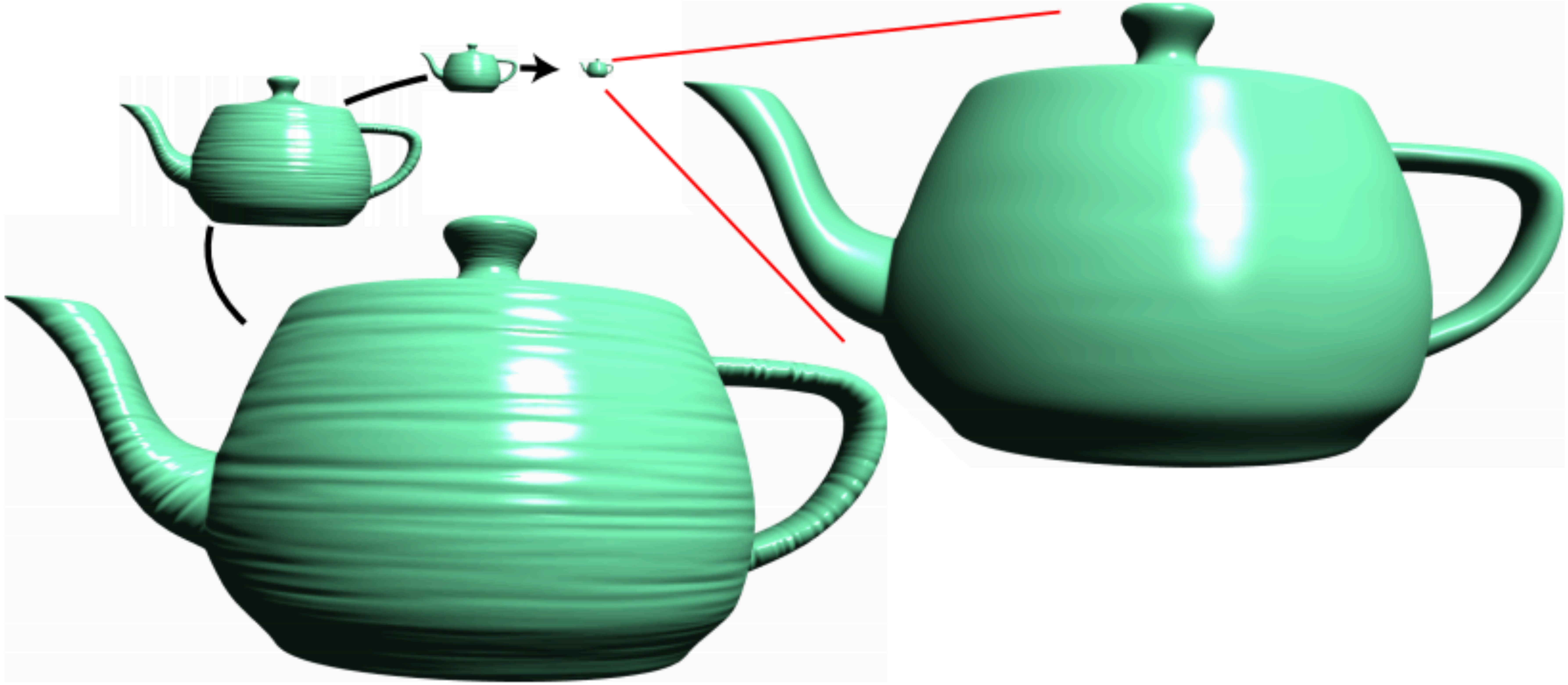


Figure 13: *Anisotropic bump pattern as a model moves away.*

LEAN mapping [Olano & Baker I3D 2010]

LEADR mapping [Dupuy et al. SIGGRAPH 2013]