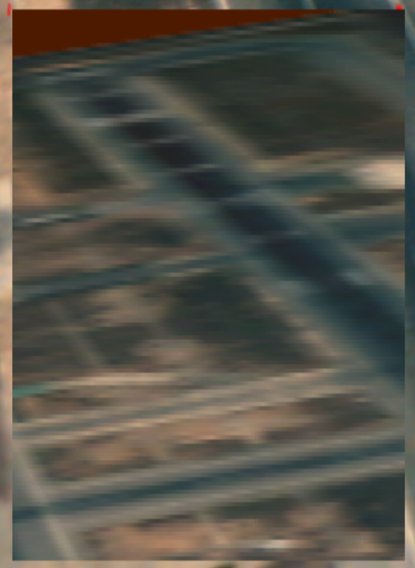
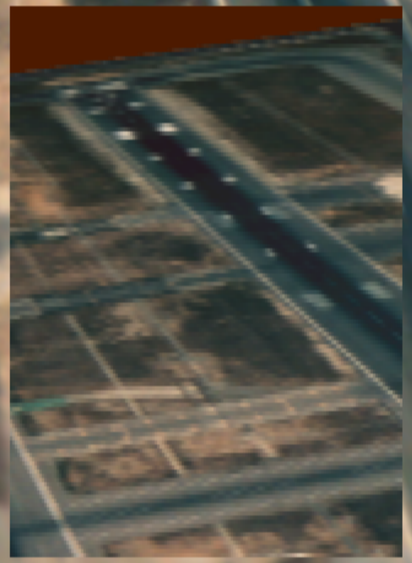
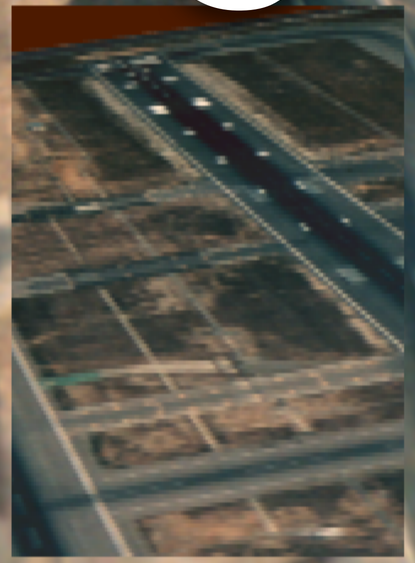
COL781: Computer Graphics

9. Texture Filtering

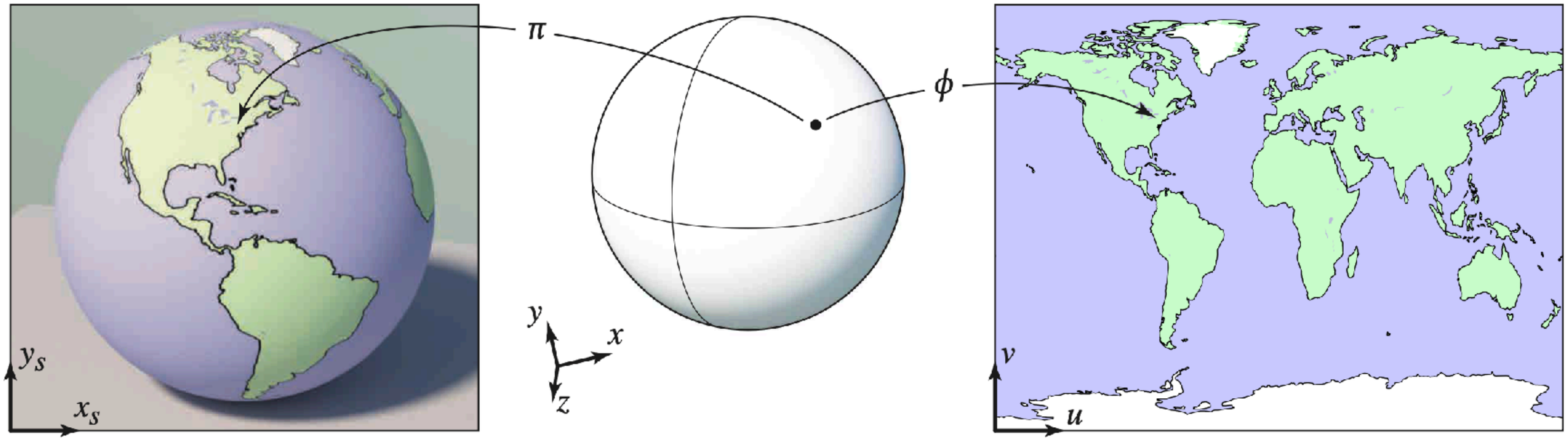


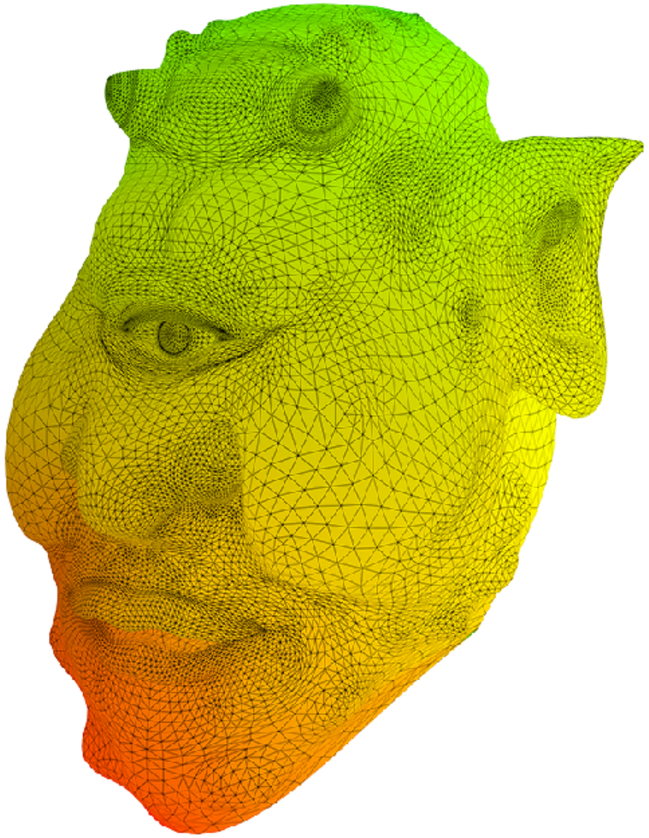
Olano et al. 2001

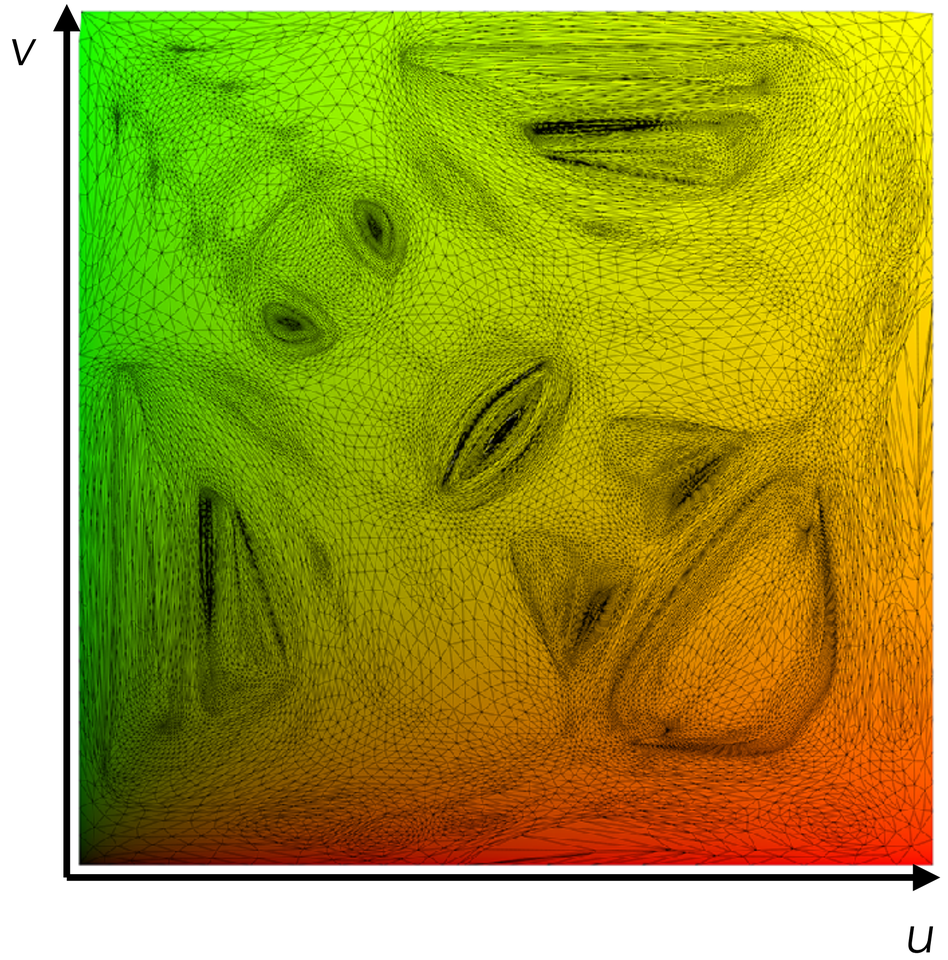
Any questions about Assignment 1?

Texture mapping

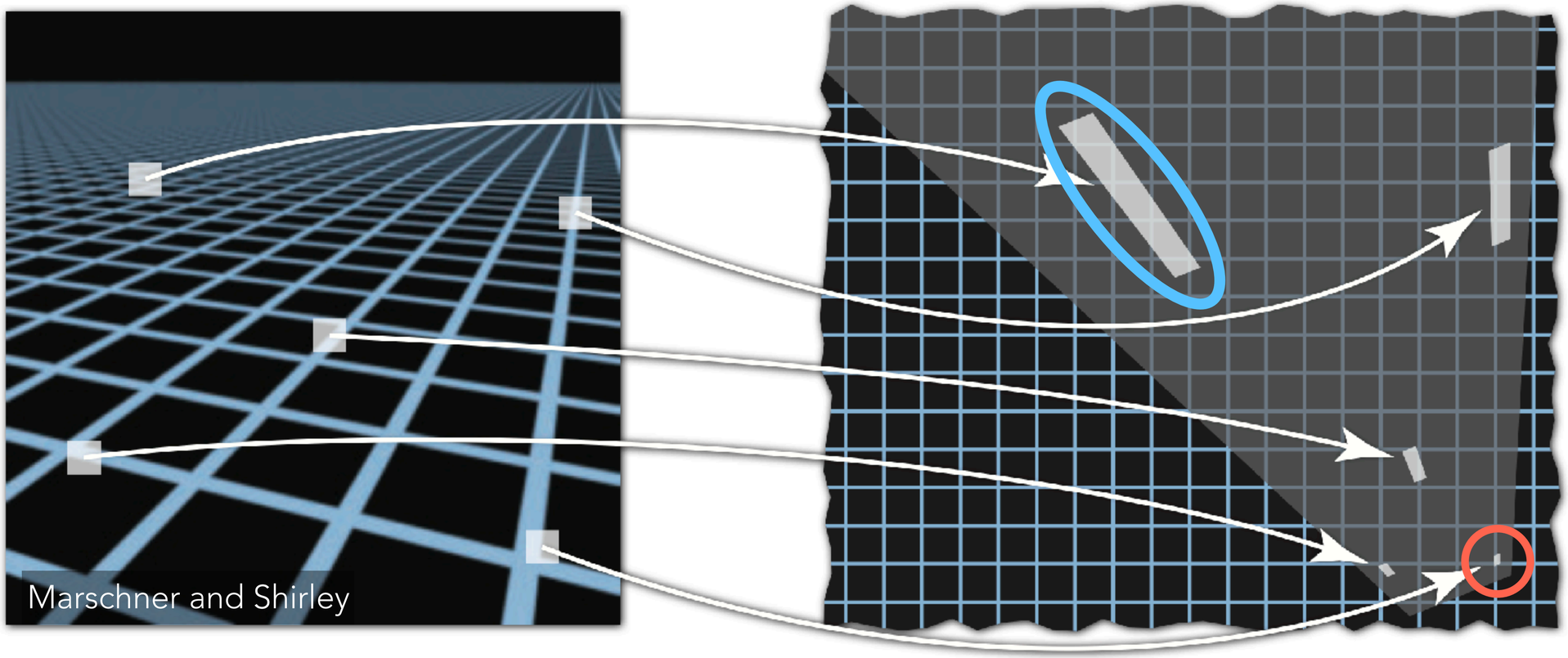


Screen space World space Texture space



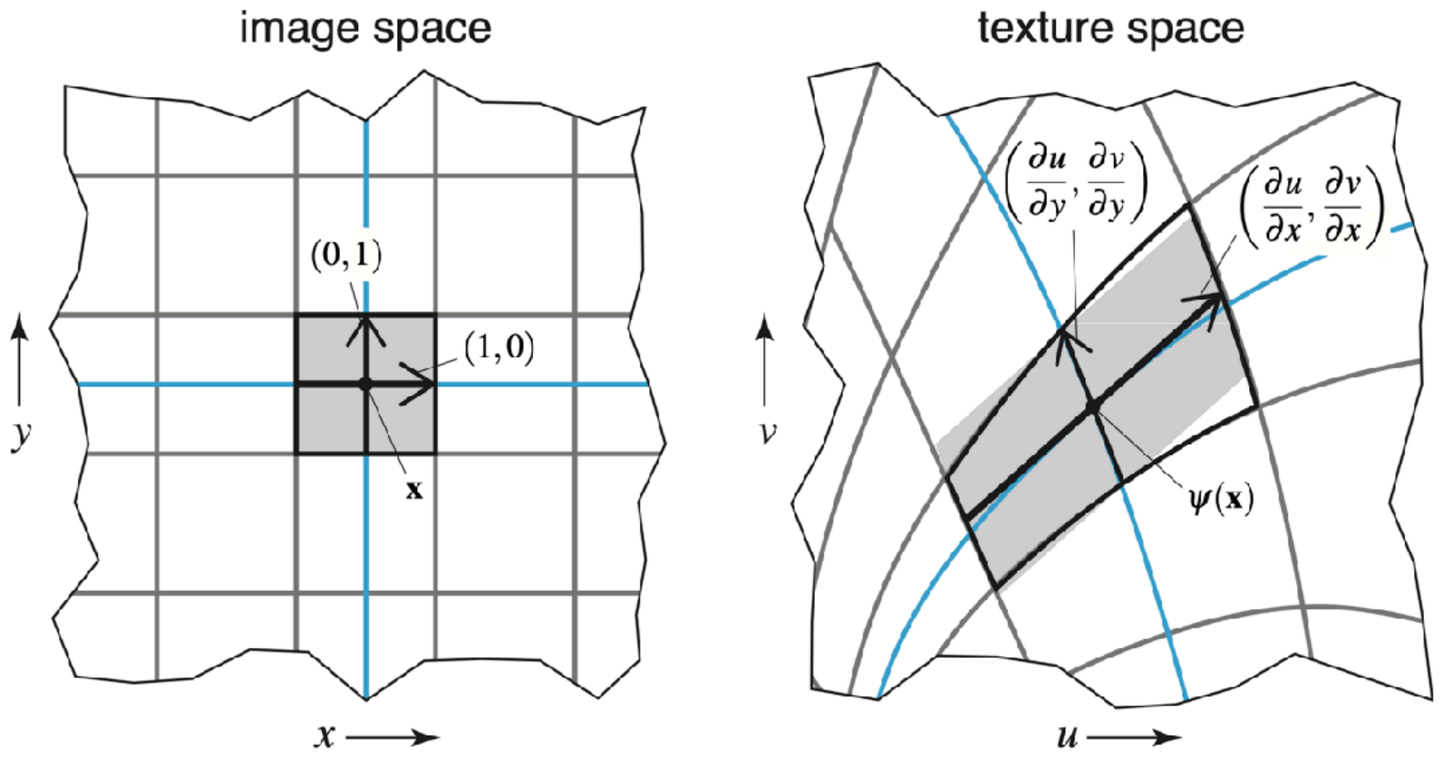


The mesh has to come with texture coordinates assigned to each mesh vertex. Otherwise, we don’t know how to map a texture image onto the mesh.



Texture mapping creates a very irregular sampling pattern!

- Some regions are **magnified**: multiple screen samples per texel
- Some regions are "**minified**": multiple texels per sample

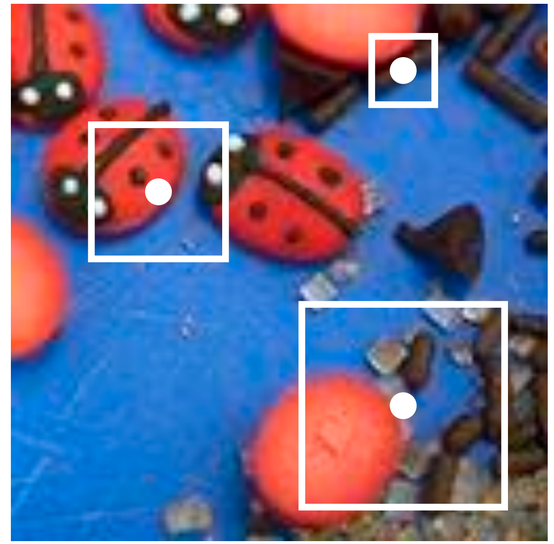


Marschner and Shirley

Evaluated for each sample while rasterizing the triangle
(analytically… or just take differences with adjacent pixels)

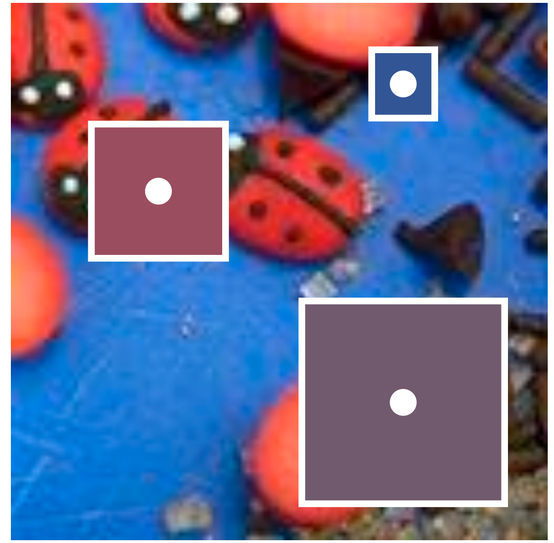
To start, let's assume the footprint is square with side D

⇒ Need to compute (weighted?) average of D^2 texels!



Solution:

- Precompute filtered (blurred) version of texture
- For each sample, look up just 1 texel in filtered image



But D will be different for different pixels…

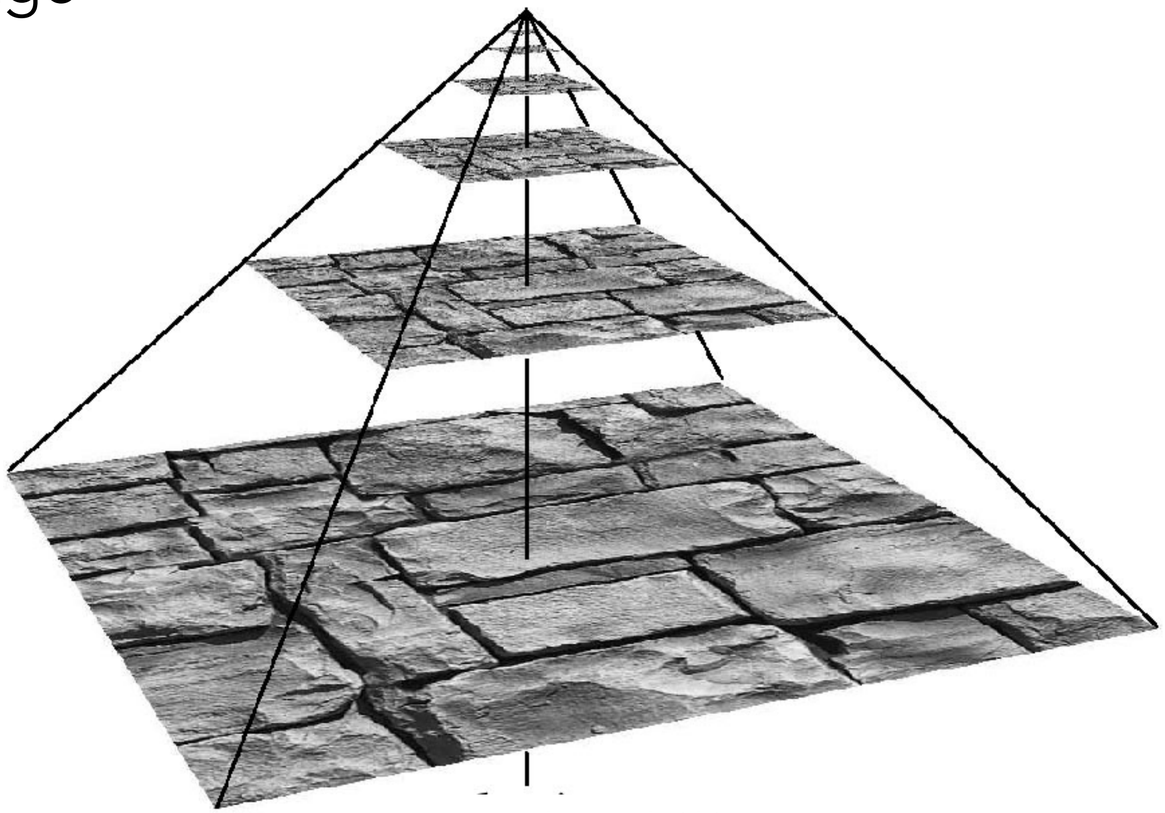
Mipmaps

Store pre-filtered versions of texture image for many different filter sizes

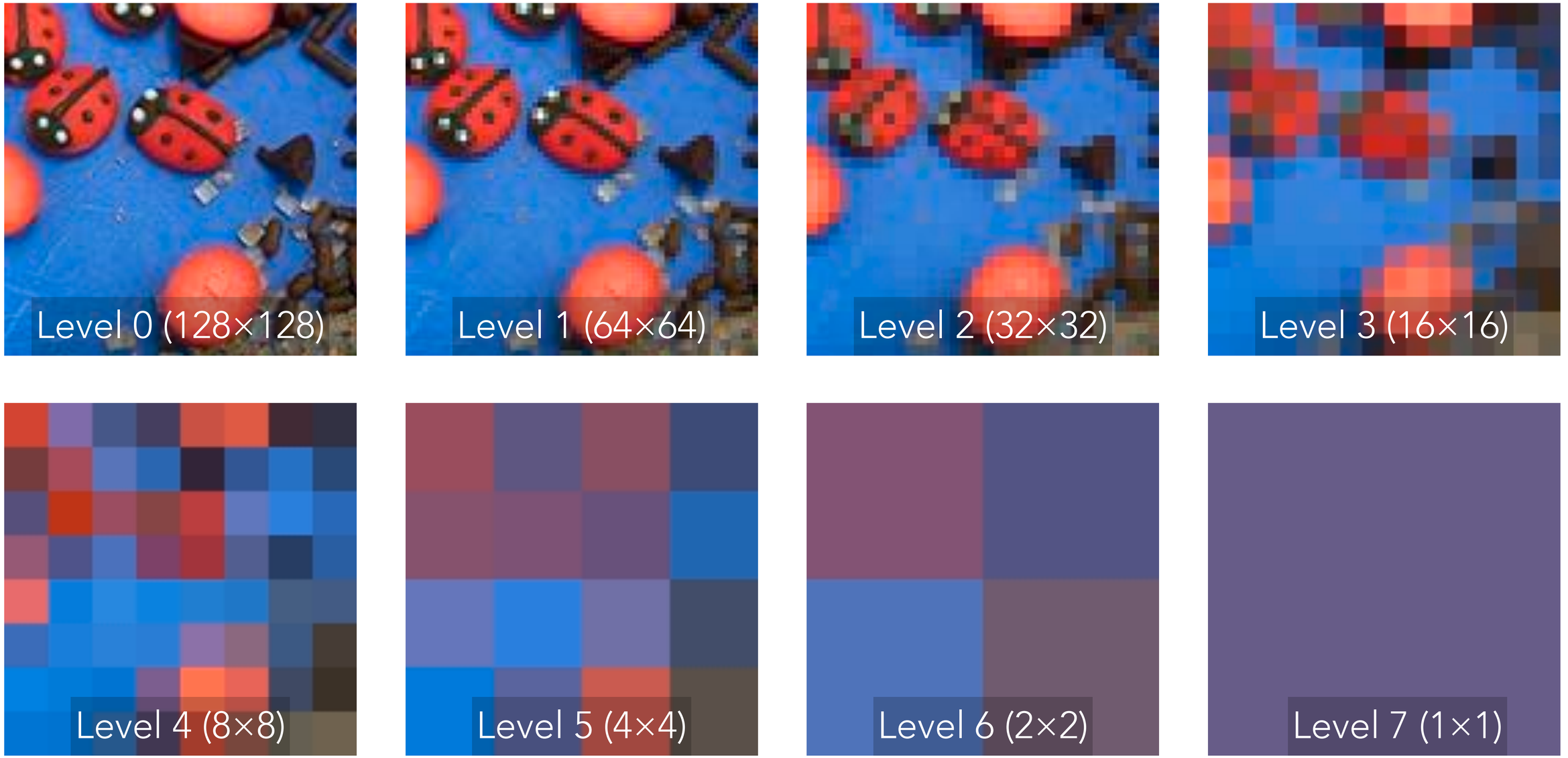
(Basically the same as image pyramids in image processing / computer vision)

Compute recursively by averaging and downsampling

Proposed by Lance Williams in 1983.
MIP = *multum in parvo* ("much in little")



mipmap level (*k*)


Level 0 (128×128)
Level 1 (64×64)
Level 2 (32×32)
Level 3 (16×16)
Level 4 (8×8)
Level 5 (4×4)
Level 6 (2×2)
Level 7 (1×1)

Ren Ng

Everything at level 0 (no filtering)

Everything at level 2 (downsampled by 4x)

Everything at level 4 (downsampled by 16x)

Using the mipmap

1 texel at level $k \approx$ square of width 2^k texels in original texture

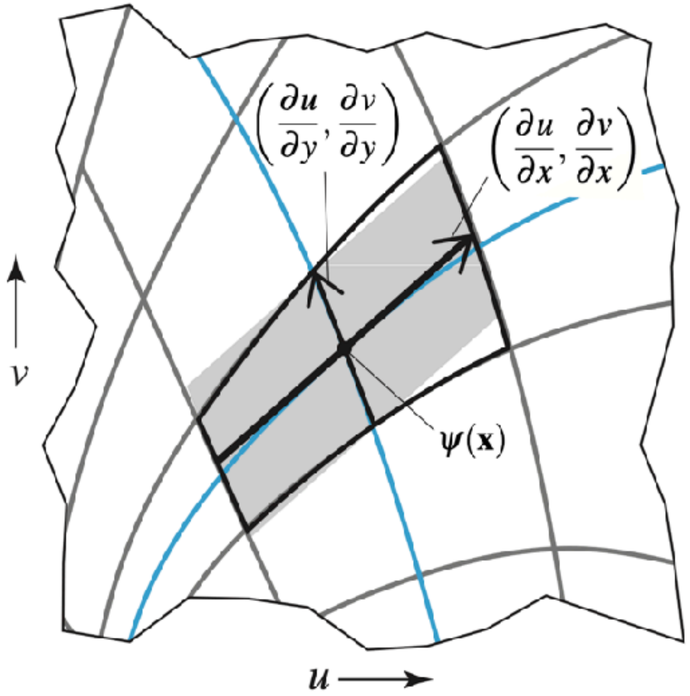
So if pixel footprint is square of width D, look up mipmap at level $k = \log_2 D$

How to compute "width" in general?

$$D = \max(|\mathrm{d}u/\mathrm{d}x|, |\mathrm{d}v/\mathrm{d}x|, |\mathrm{d}u/\mathrm{d}y|, |\mathrm{d}v/\mathrm{d}y|)$$

$$D = \max(\sqrt{(\mathrm{d}u/\mathrm{d}x)^2 + (\mathrm{d}v/\mathrm{d}x)^2}, \sqrt{(\mathrm{d}u/\mathrm{d}y)^2 + (\mathrm{d}v/\mathrm{d}y)^2})$$

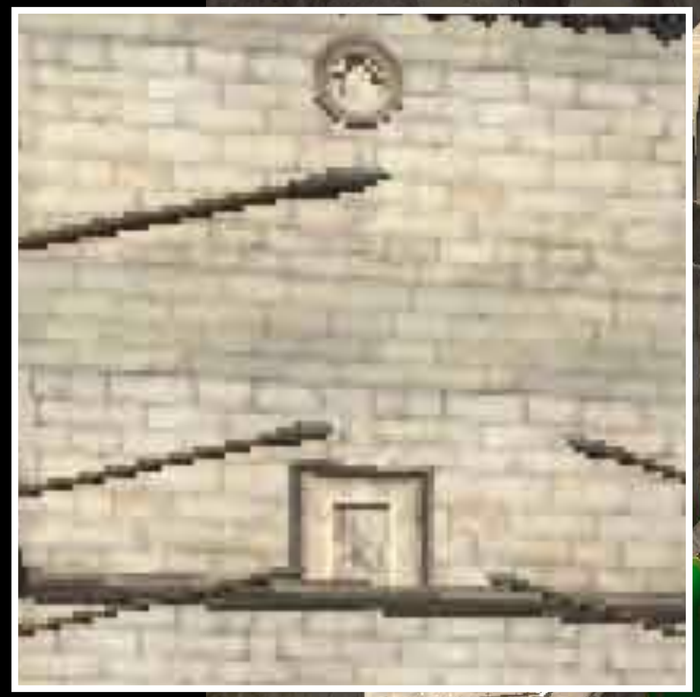
(Why max and not min or average?)



Visualization of mipmap level

Mipmap level $k = \log_2 D$ rounded to nearest integer

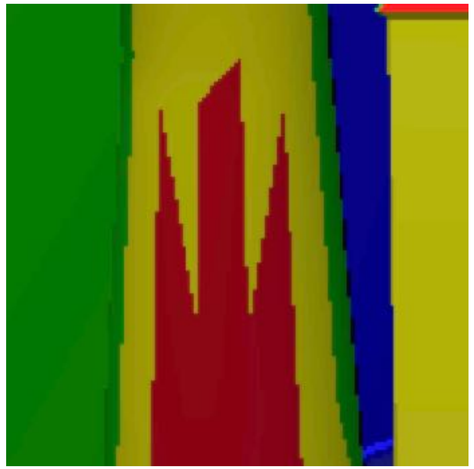
Mipmapped textures



Visualization of mipmap level

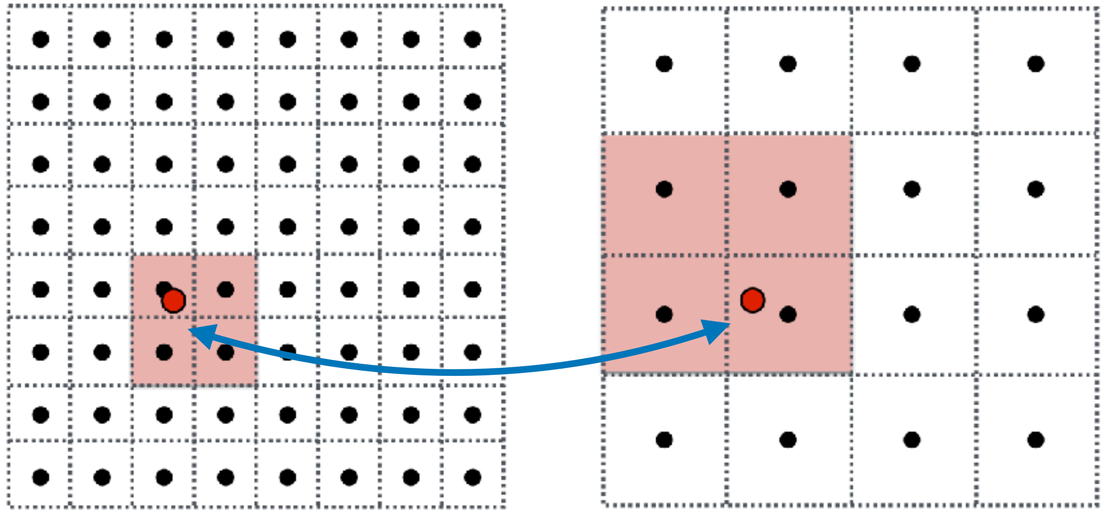
Continuous mipmap level $k = \log_2 D$

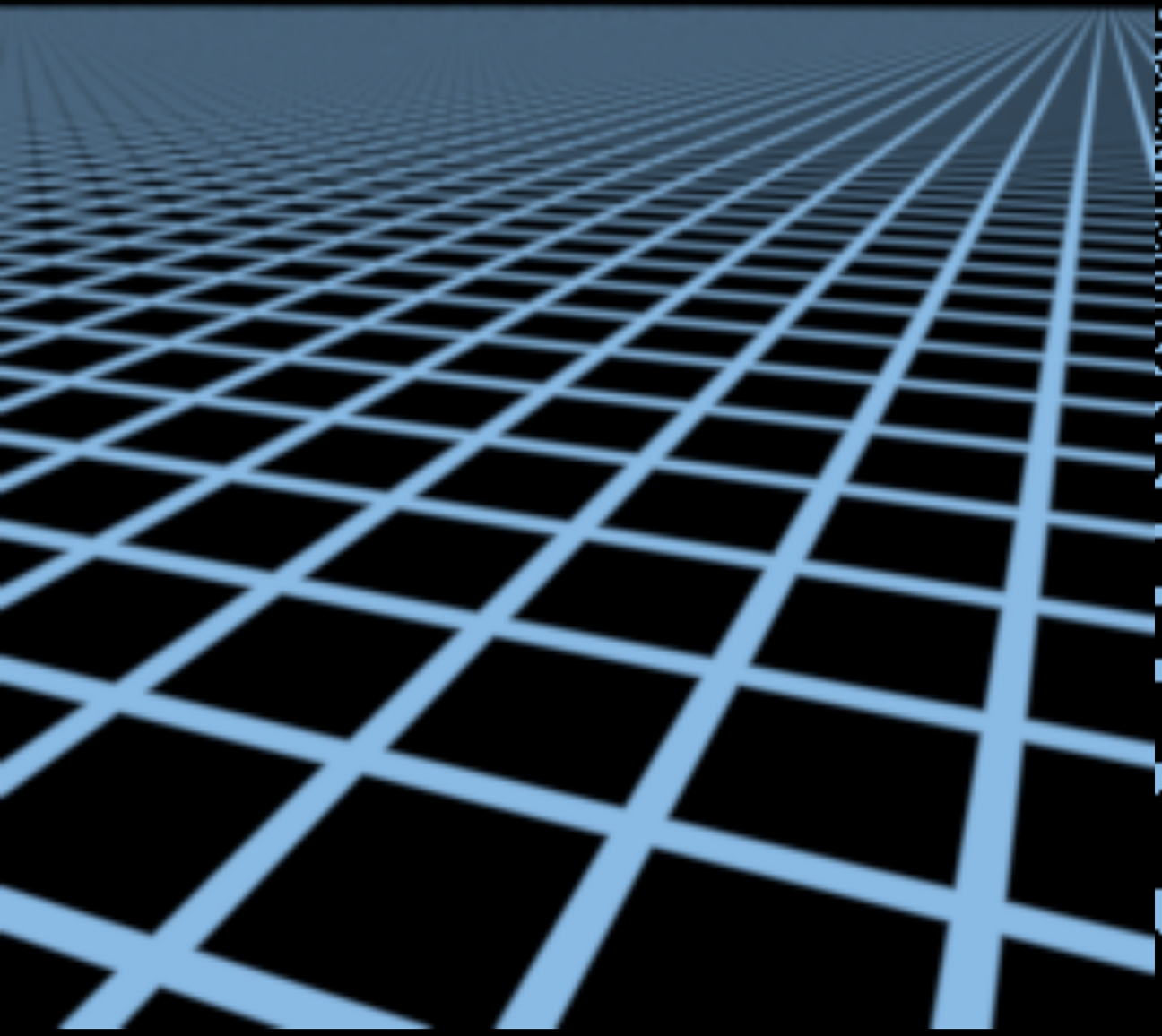
Basic mipmapping produces discontinuous “jumps” in texture detail



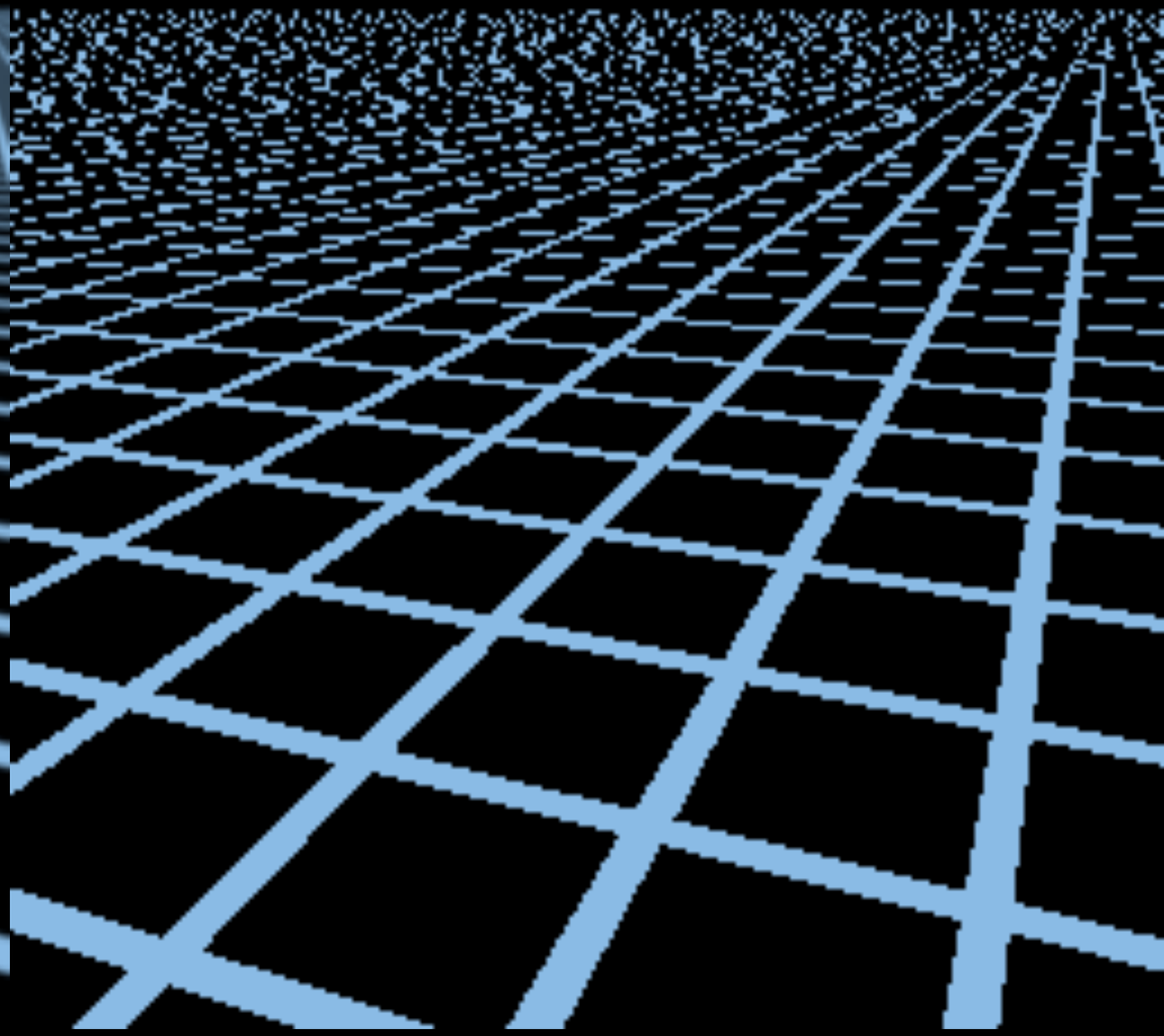
Trilinear filtering: interpolate between results of two adjacent mipmap levels

- Bilinear interpolation at level $\lfloor k \rfloor$
- Bilinear interpolation at level $\lfloor k \rfloor + 1$
- Linear interpolation between them

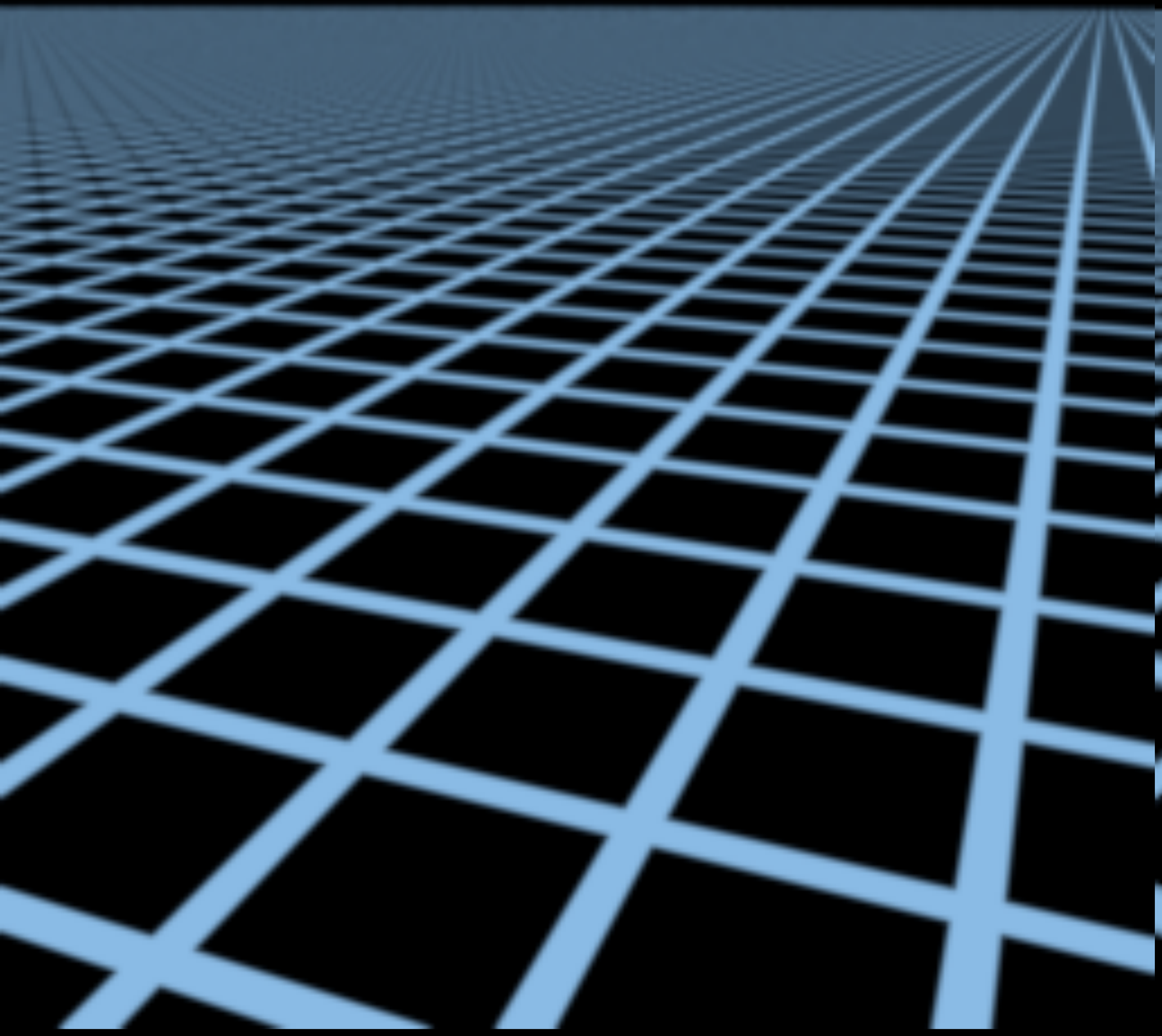




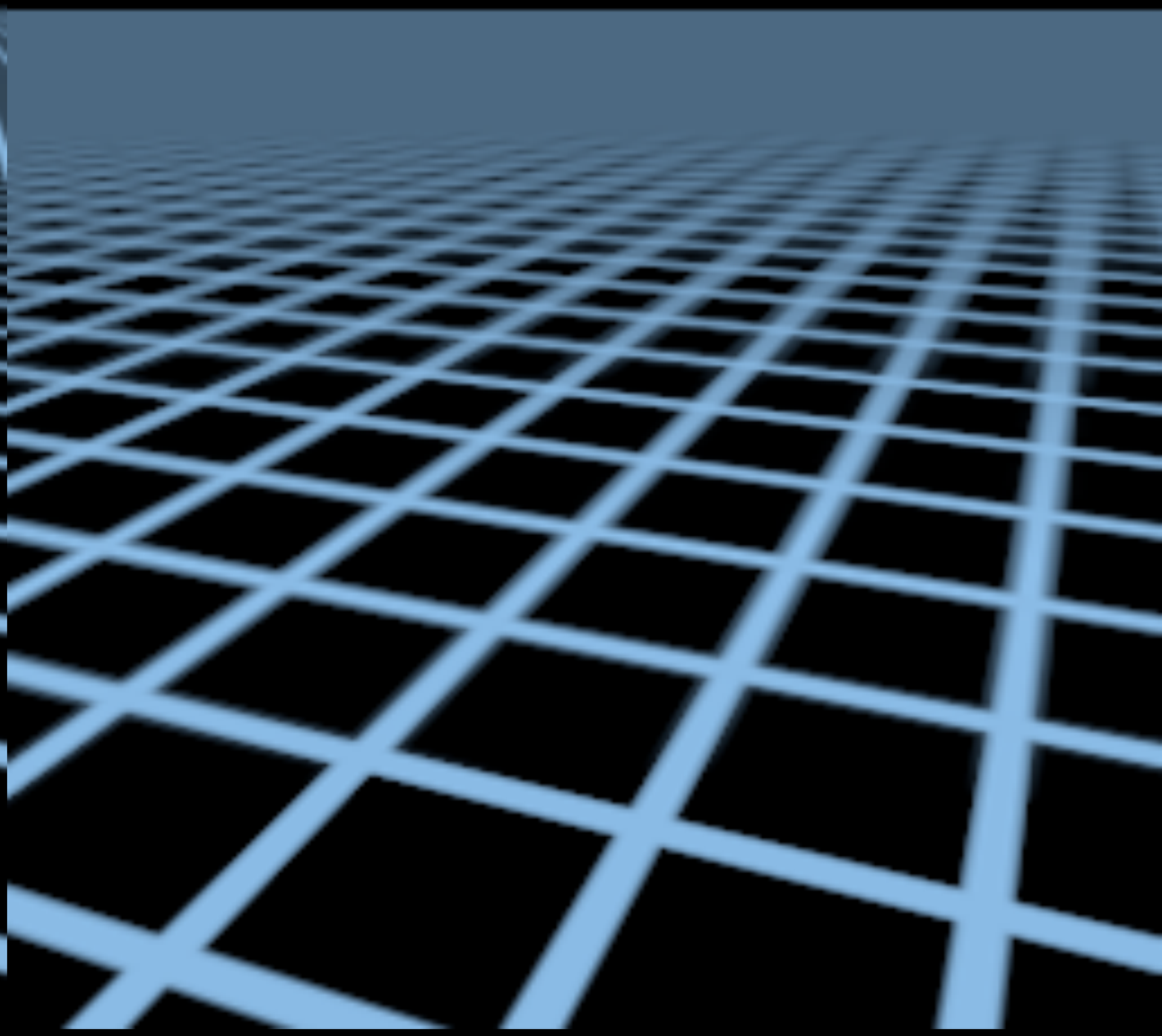
Supersampled reference (256×256, 512 spp)



Point sampling (256×256)

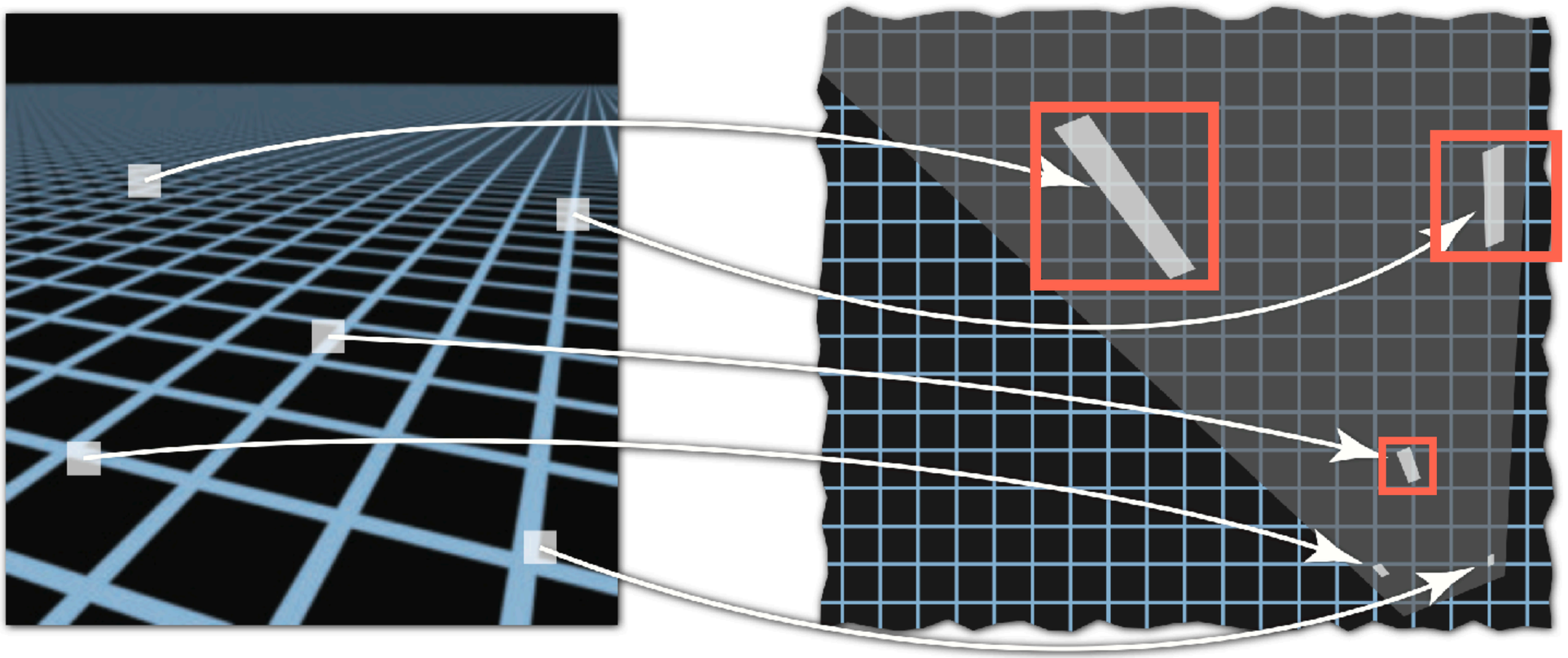


Supersampled reference (256×256, 512 spp)

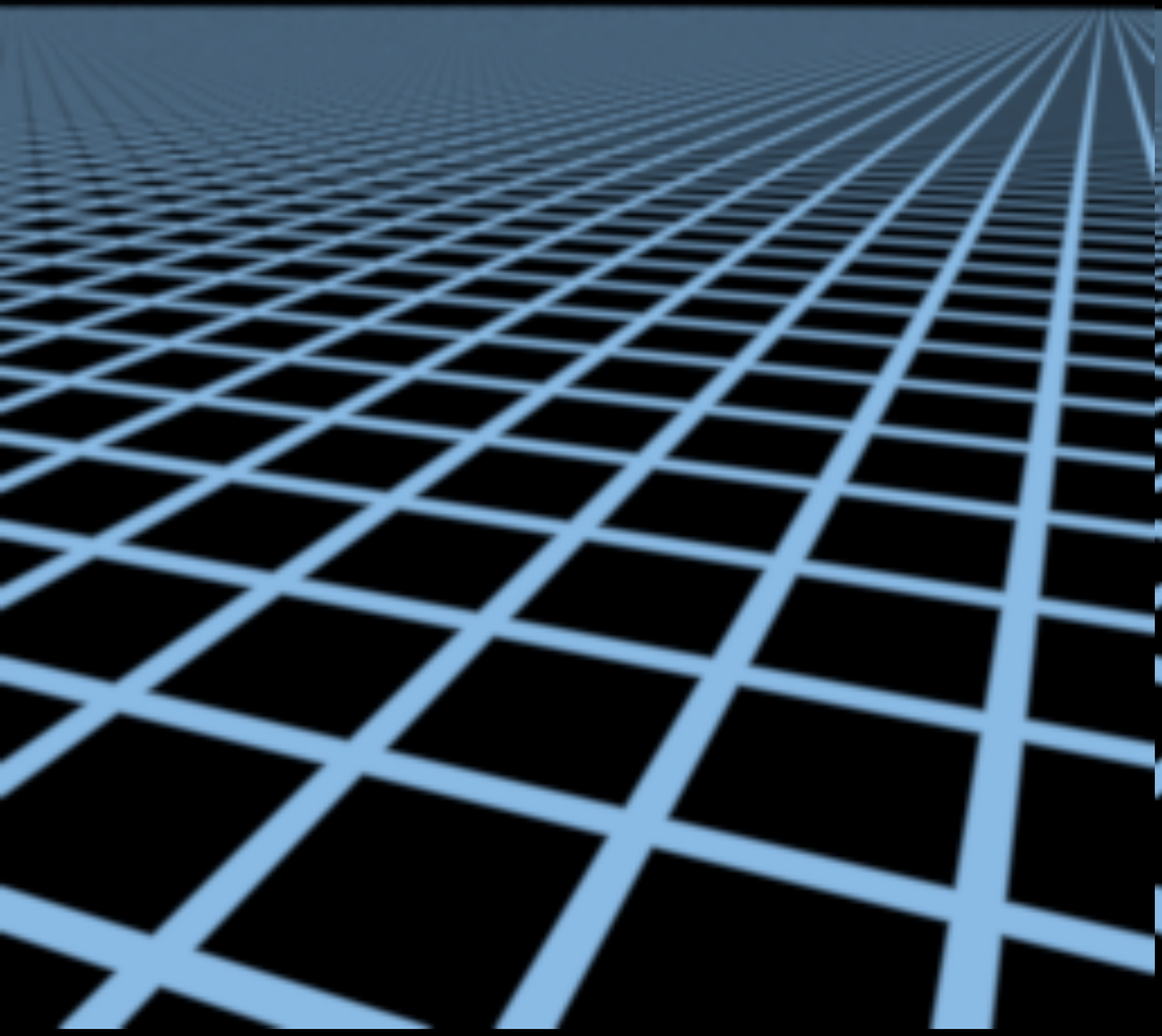


Mipmap with trilinear filtering

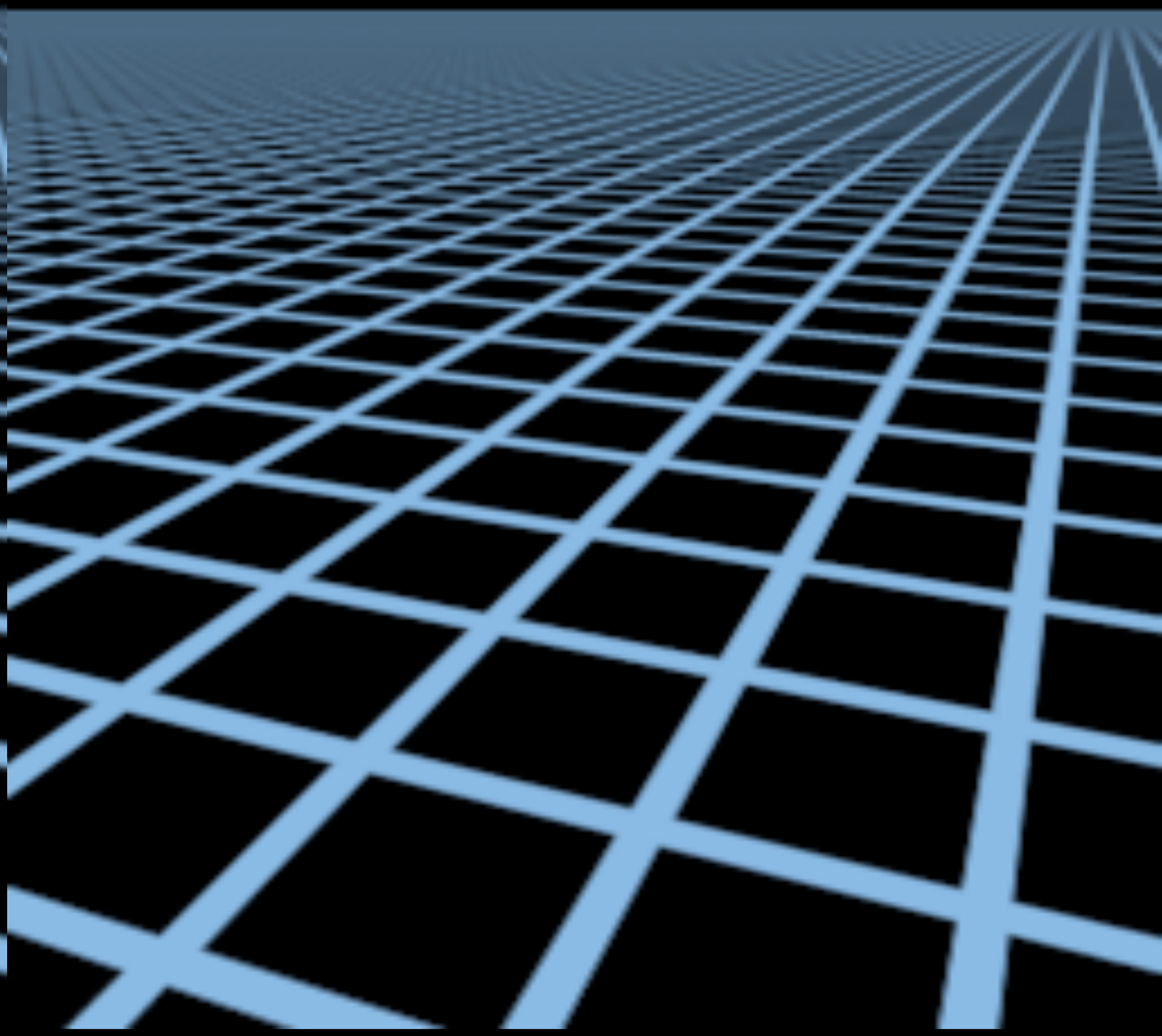
At grazing angles, pixel footprint is very stretched out!



Mipmaps only allow isotropic filtering (same in all directions)



Supersampled reference (256×256, 512 spp)



Elliptical weighted average (EWA)

Anisotropic filtering

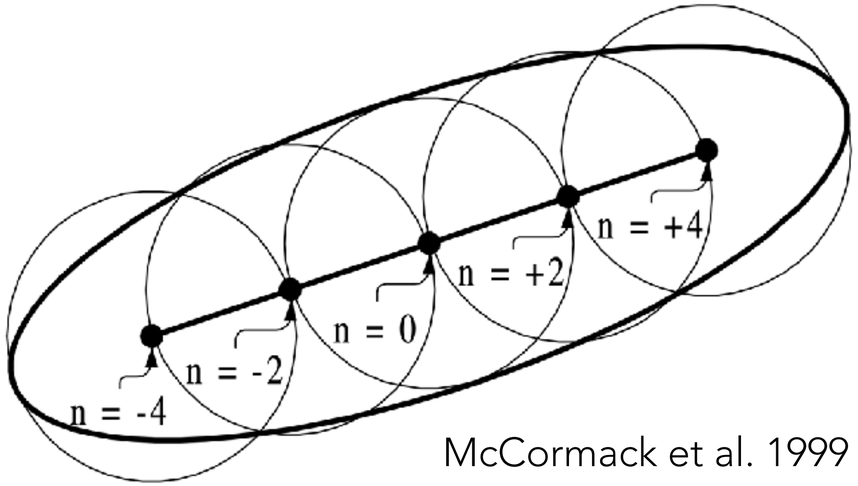
Not on the exam :)

Treat pixel as circular (e.g. Gaussian kernel)
→ maps to ellipse in texture space
→ approximate as line of blobs

Choose mipmap level using minor axis

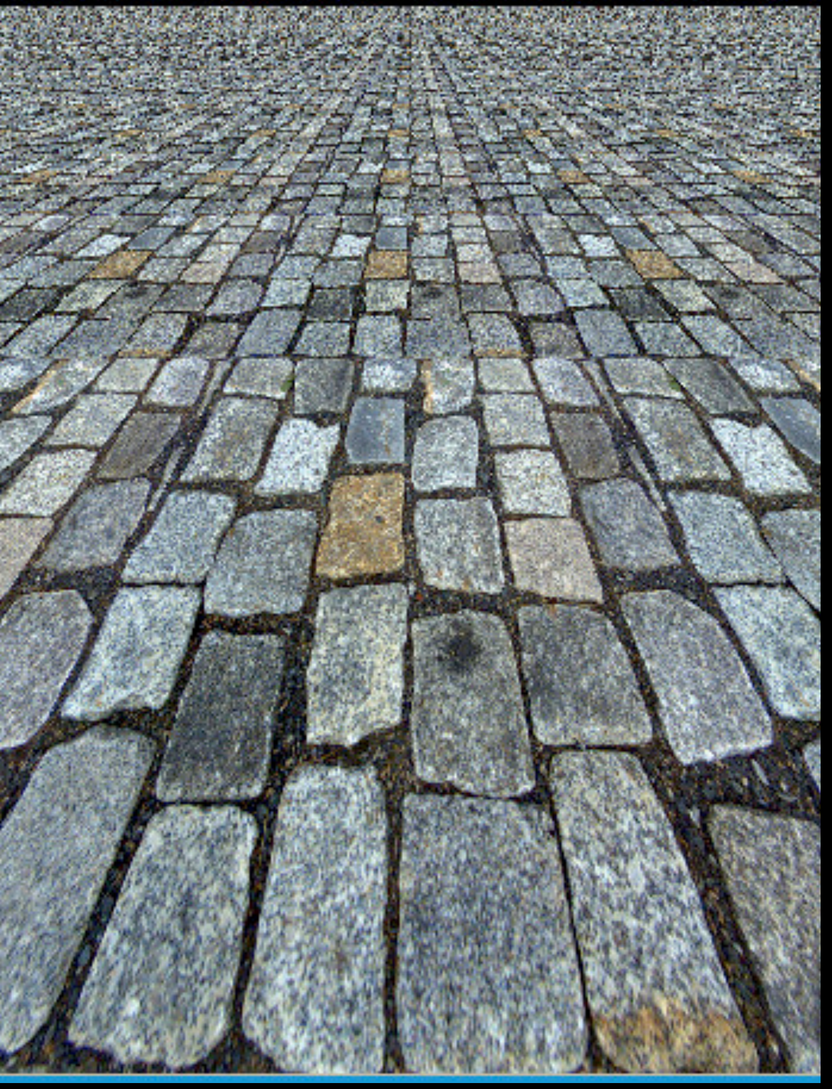
Take multiple samples along major axis

This is what GPUs do when they say “16x anisotropic filtering”



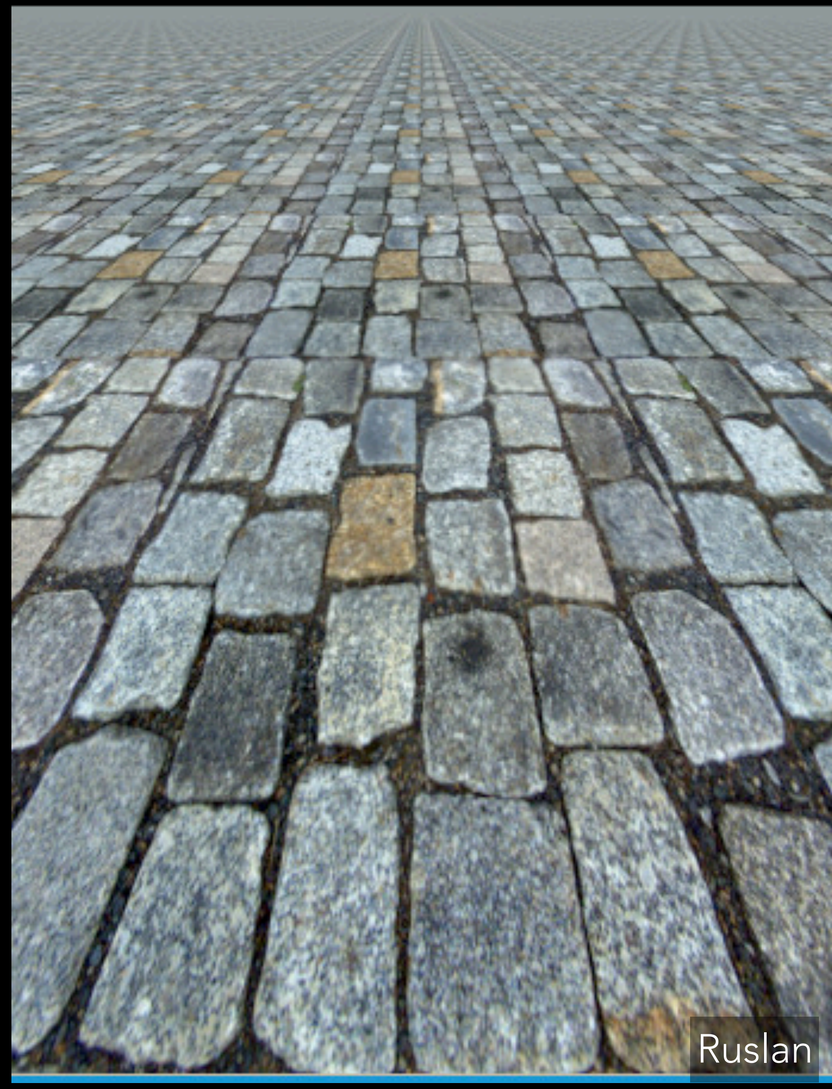
McCormack et al. 1999

[Original idea by Greene and Heckbert 1986, faster approximation using mipmaps by McCormack et al. 1999]



No filtering

Mipmapping



Anisotropic filtering

Homework suggestion: Make your own aliasing

Load an image (using SDL2_image, or just matplotlib.image in Python) and create a distorted version using a nonlinear transformation $u(x, y)$, $v(x, y)$