

Texture mapping basics

CS 4620 Lecture 8

Texture mapping

- **Objects have properties that vary across the surface**



Texture Mapping

- So we make the shading parameters vary across the surface



[Foley et al. / Perlin]

Texture mapping

- Adds visual complexity; makes appealing images



P.

Texture mapping

- **Surface properties are not the same everywhere**
 - diffuse color (k_d) varies due to changing pigmentation
 - roughness (α) and refractive index (n) for specular highlight vary due to changing finishes and surface contamination
- **Want functions that assign properties to points on the surface**
 - the surface is a 2D domain
 - given a surface parameterization, just need function on plane
 - images are a handy way to represent such functions
 - can represent using any image representation
 - raster texture images are very popular

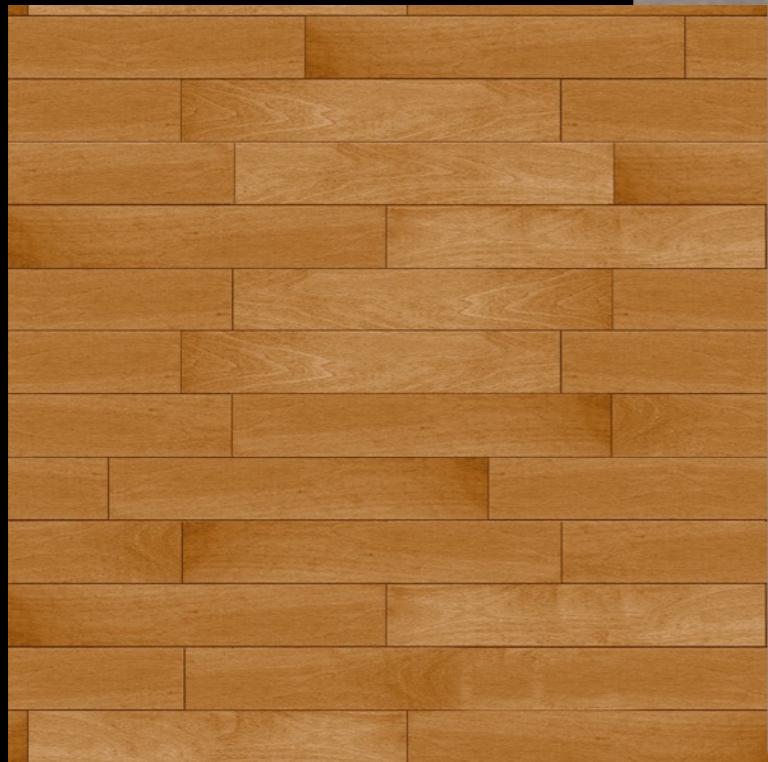
A first definition

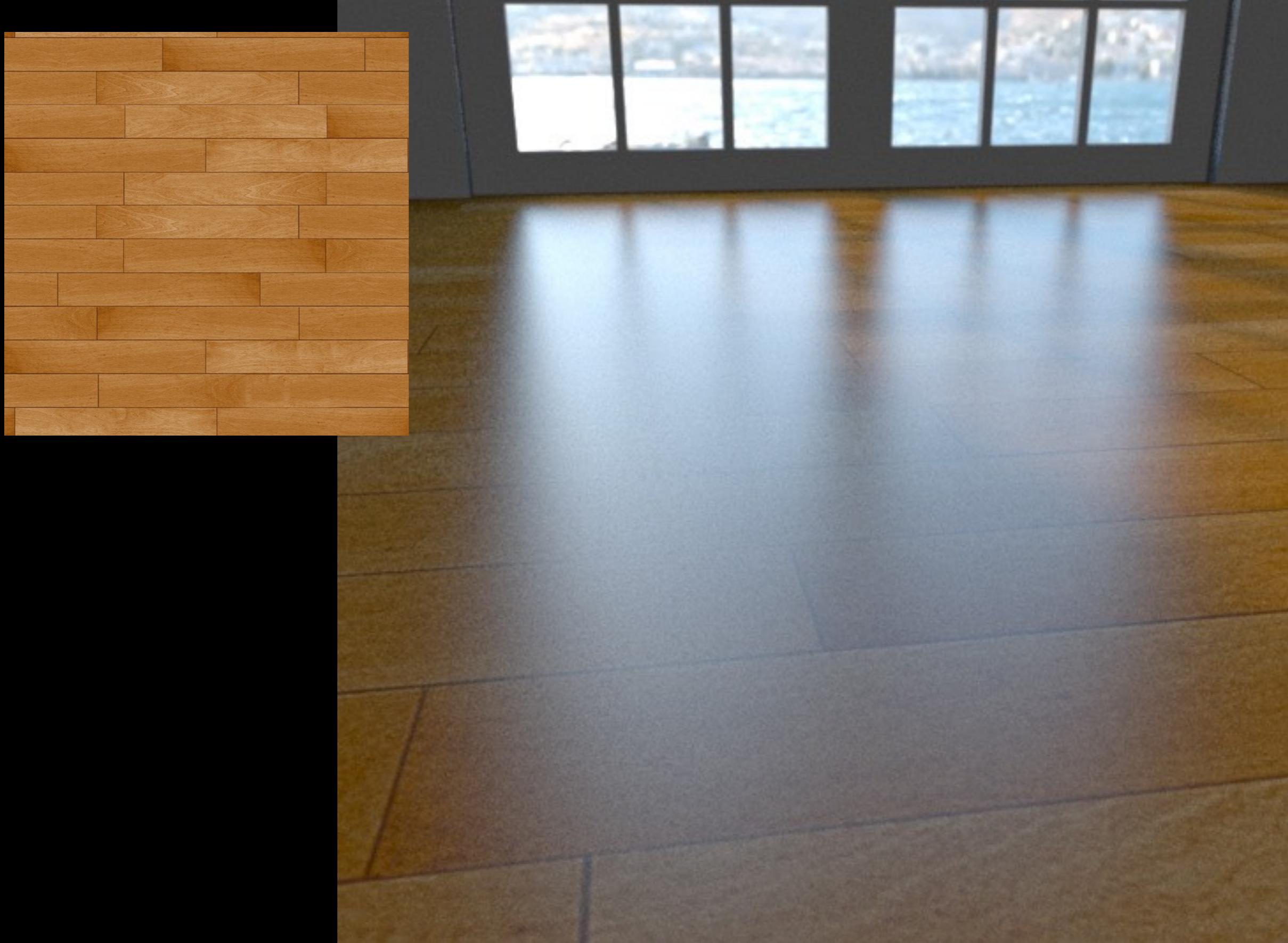
Texture mapping: a technique of defining surface properties (especially shading parameters) in such a way that they vary as a function of position on the surface.

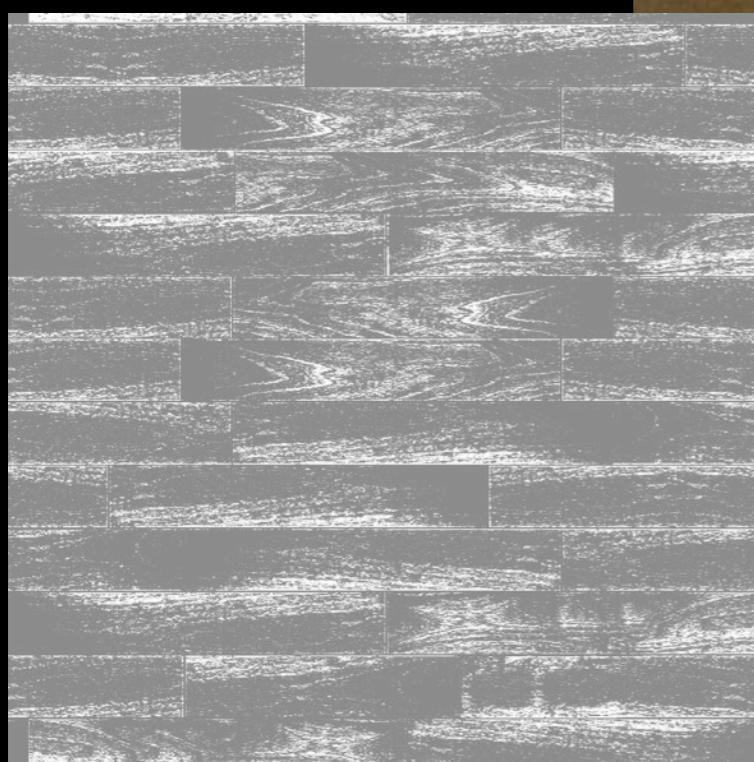
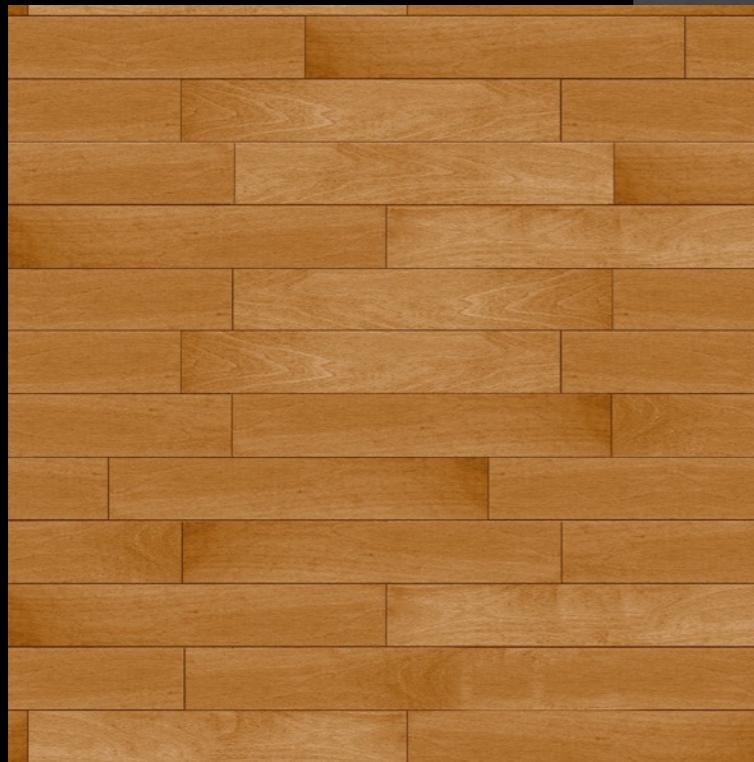
- **This is very simple!**
 - but it produces complex-looking effects

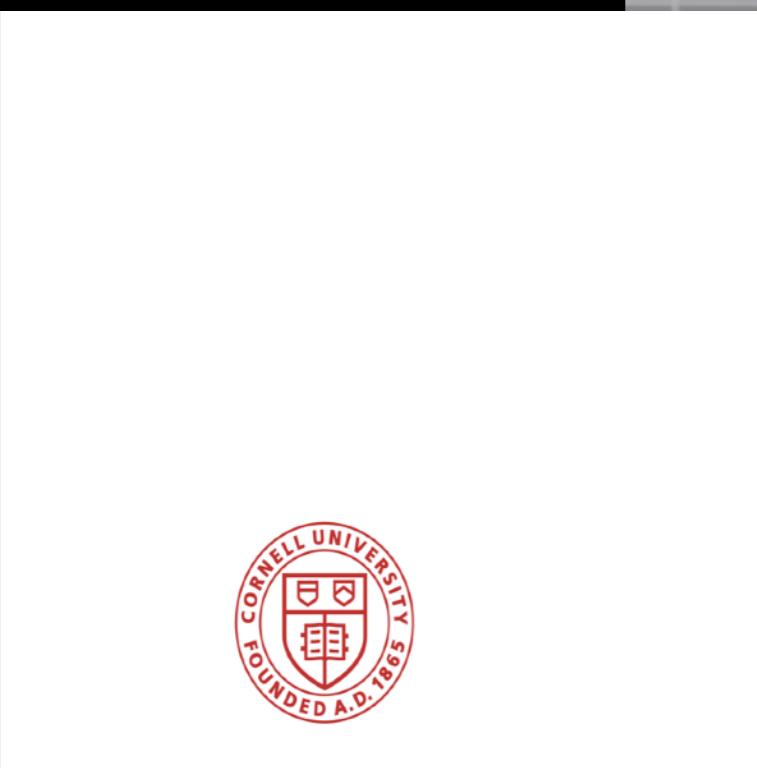
Examples

- **Wood gym floor with smooth finish**
 - diffuse color k_D varies with position
 - specular properties α, n are constant
- **Glazed pot with finger prints**
 - diffuse color k_D are constant
 - roughness α and refractive index n vary with position
- **Adding dirt to painted surfaces**
- **Simulating stone, fabric, ...**
 - to approximate effects of small-scale geometry
 - they look flat but are a lot better than nothing

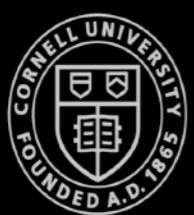








RENDERED USING MITSUBA



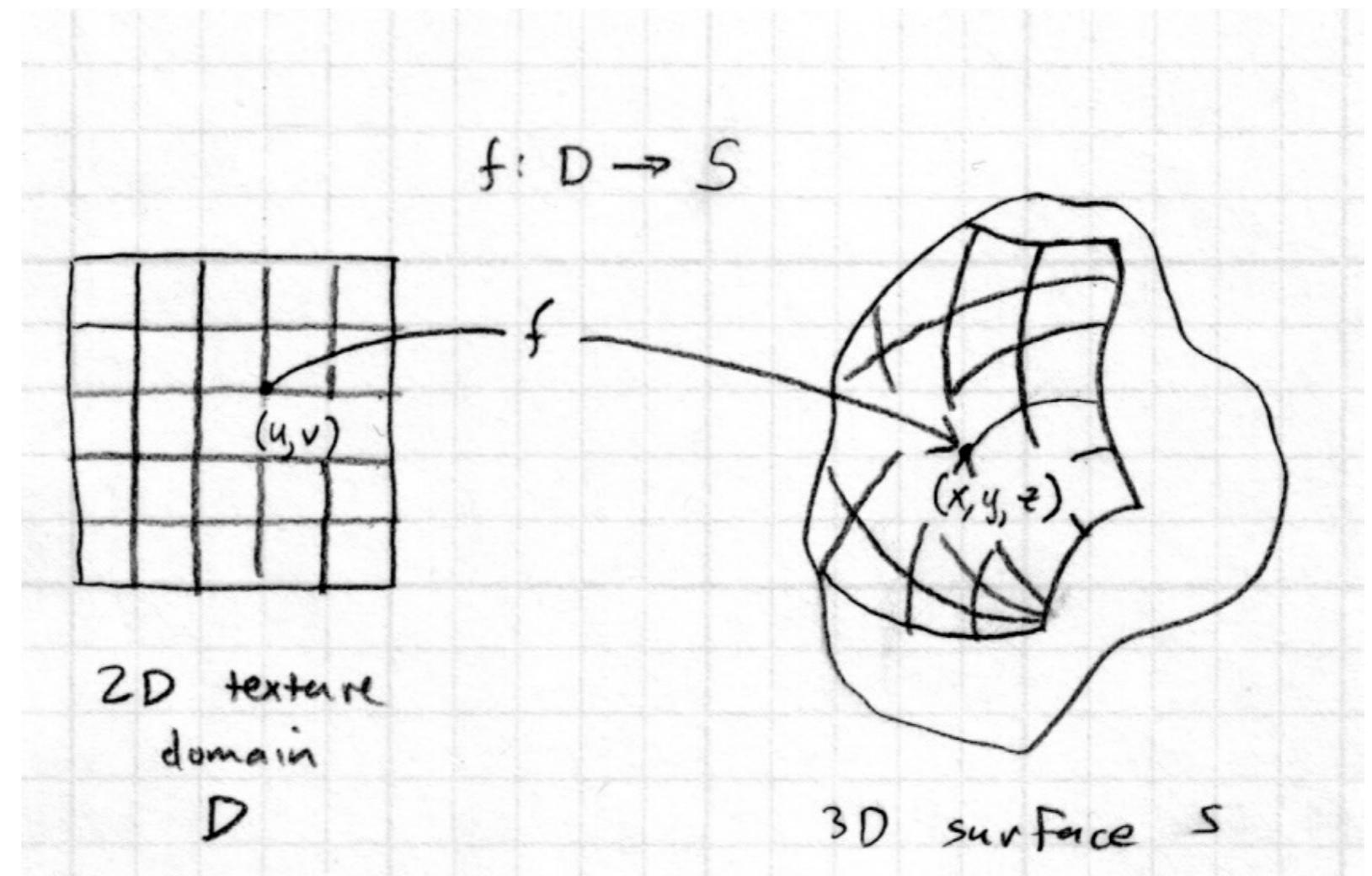
Mapping textures to surfaces

- **Usually the texture is an image (function of u, v)**
 - the big question of texture mapping: where on the surface does the image go?
 - obvious only for a flat rectangle the same shape as the image
 - otherwise more interesting

Aside: parametric
vs. implicit
vs. piecewise
surface models

Mapping textures to surfaces

- “Putting the image on the surface”
 - this means we need a function f that tells where each point on the image goes
 - this looks a lot like a parametric surface function
 - for parametric surfaces (e.g. sphere, cylinder) you get f for free



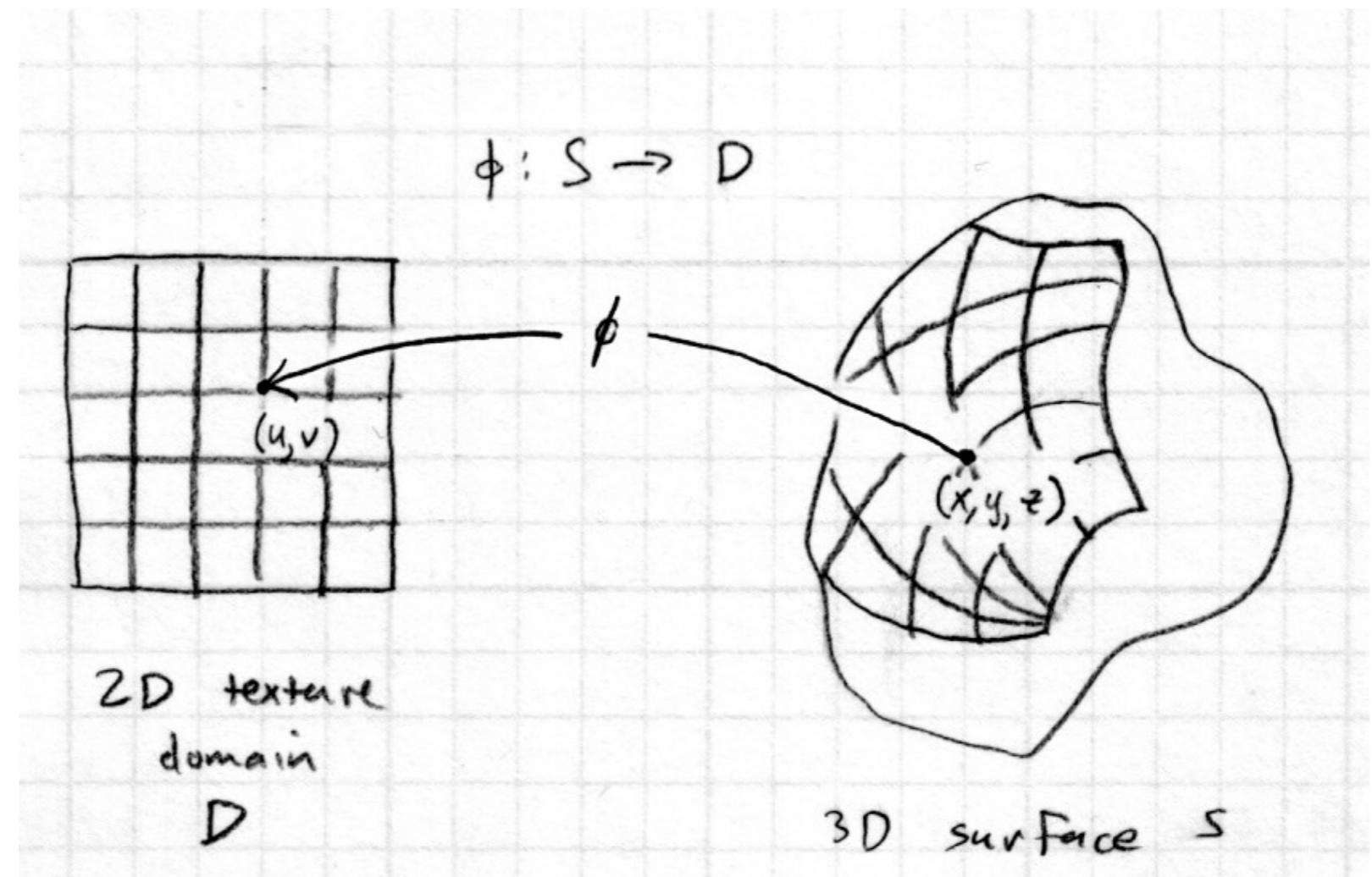
Texture coordinate functions

- **Non-parametrically defined surfaces: more to do**
 - can't assign texture coordinates as we generate the surface
 - need to have the *inverse* of the function f

- **Texture coordinate fn.**

$$\phi : S \rightarrow \mathbb{R}^2$$

- when shading \mathbf{p} get texture at $\phi(\mathbf{p})$



Three spaces

- Surface lives in 3D world space
- Every point also has a place where it goes in the image and in the texture.

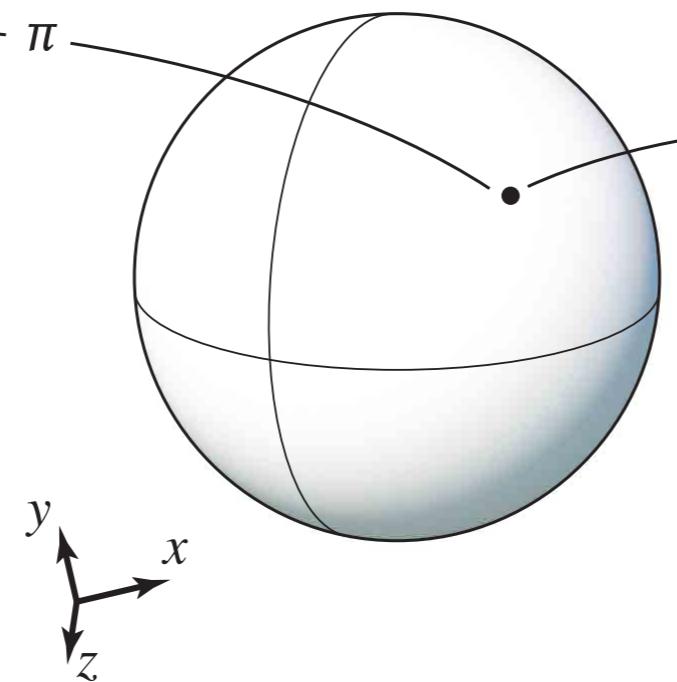
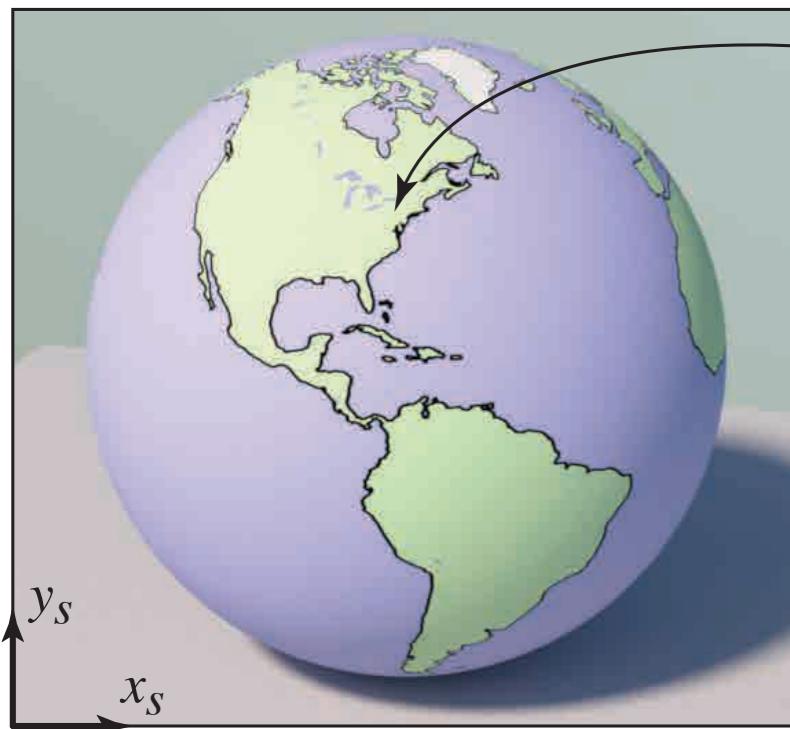
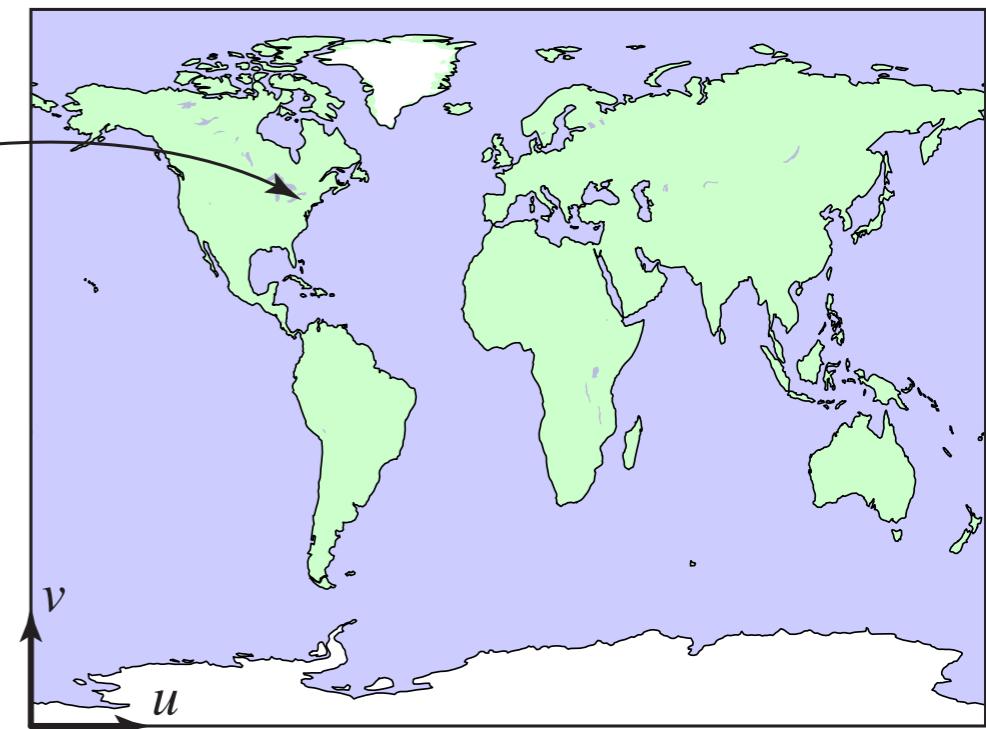


image space



texture space

Texture coordinate functions

- **Define texture image as a function**

$$T : D \rightarrow C$$

- where C is the set of colors for the diffuse component

- **Diffuse color (for example) at point \mathbf{p} is then**

$$k_D(\mathbf{p}) = T(\phi(\mathbf{p}))$$

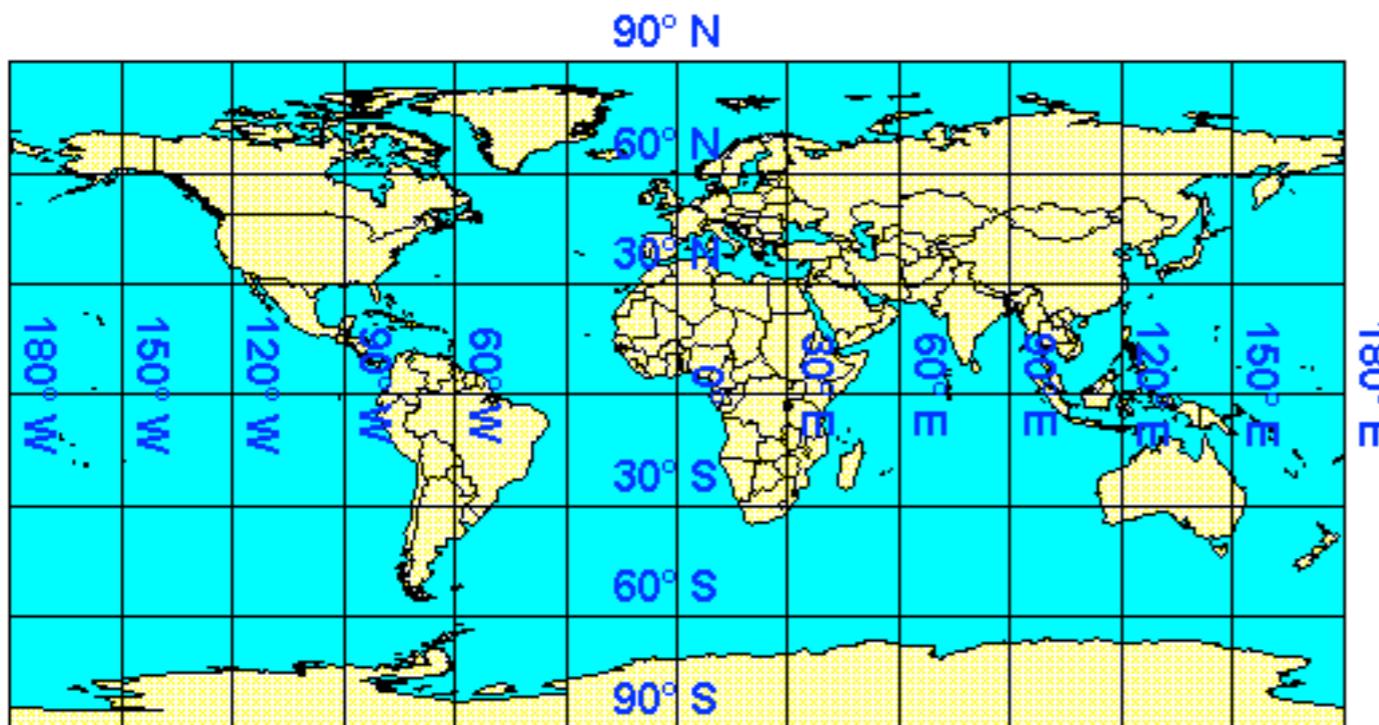
Examples of coordinate functions

- **A rectangle**
 - image can be mapped directly, unchanged

Examples of coordinate functions

- **For a sphere: latitude-longitude coordinates**
 - ϕ maps point to its latitude and longitude

[map: Peter H. Dana]

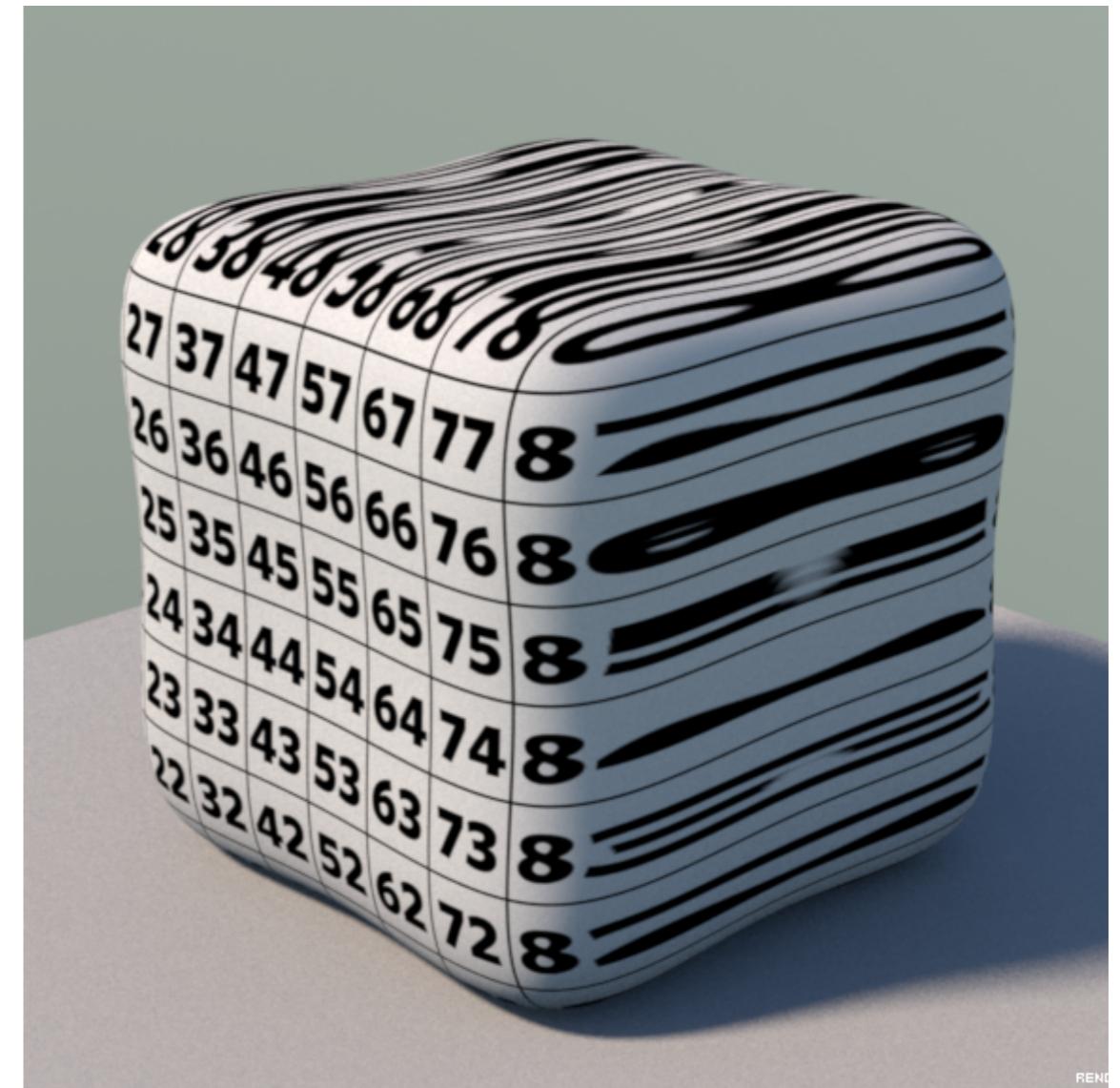
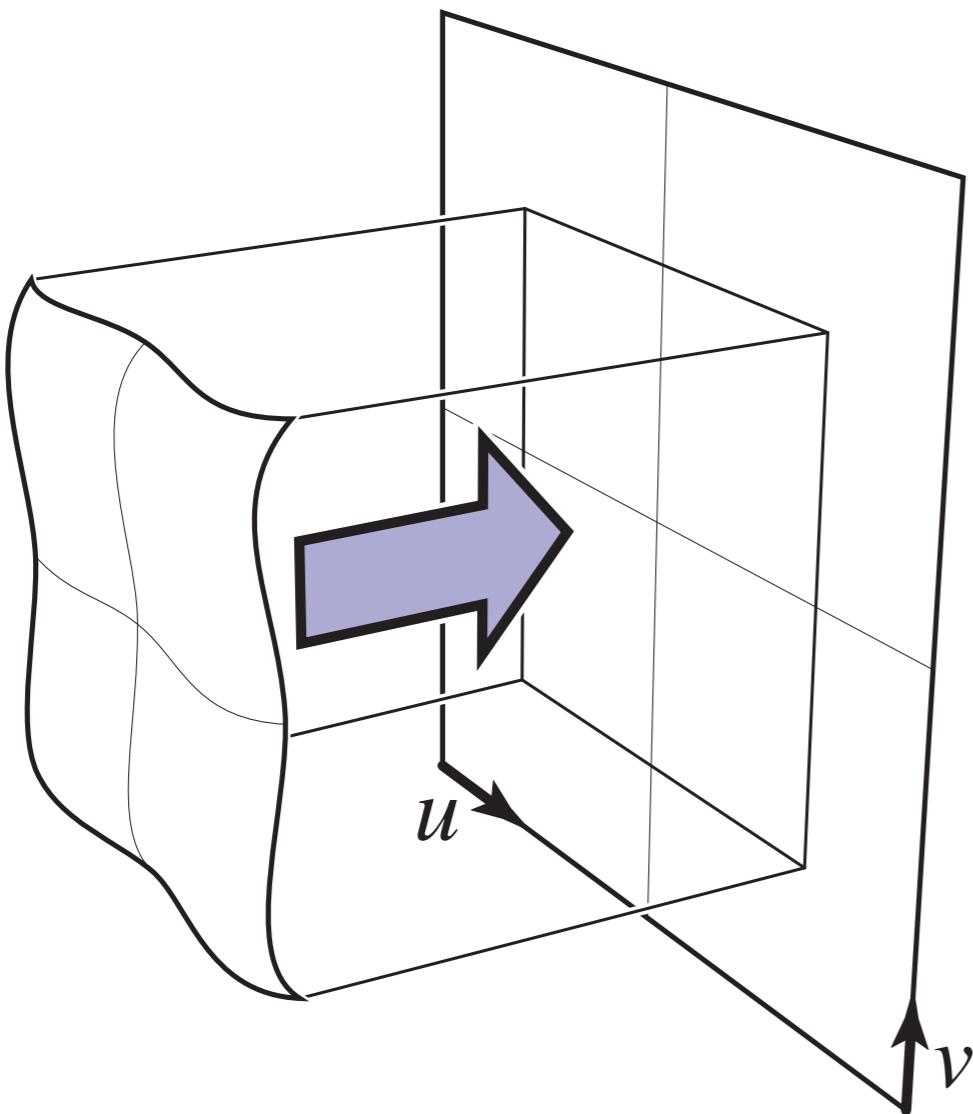


Coordinate functions: non-parametric

- **Many surfaces don't come with coordinates**
 - implicit surfaces
 - random triangle meshes
- **Some surface come with coordinates that are incomplete**
 - triangle meshes: natural coordinate for each triangle
 - spline surfaces: natural coordinates for each patch
- **Generally need to be able to compute texture coordinates from the surface position**

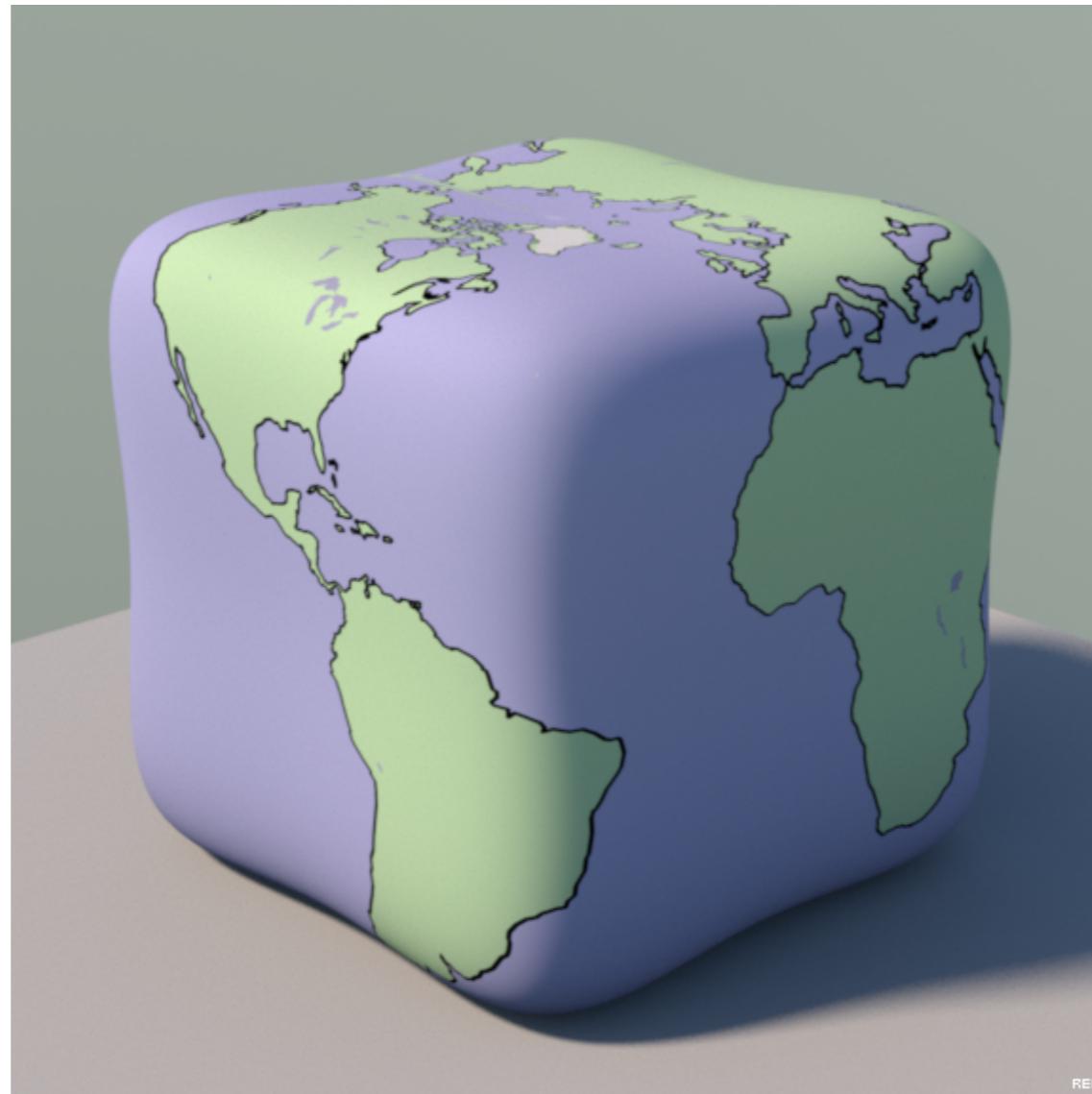
Examples of coordinate functions

- **Planar projection**



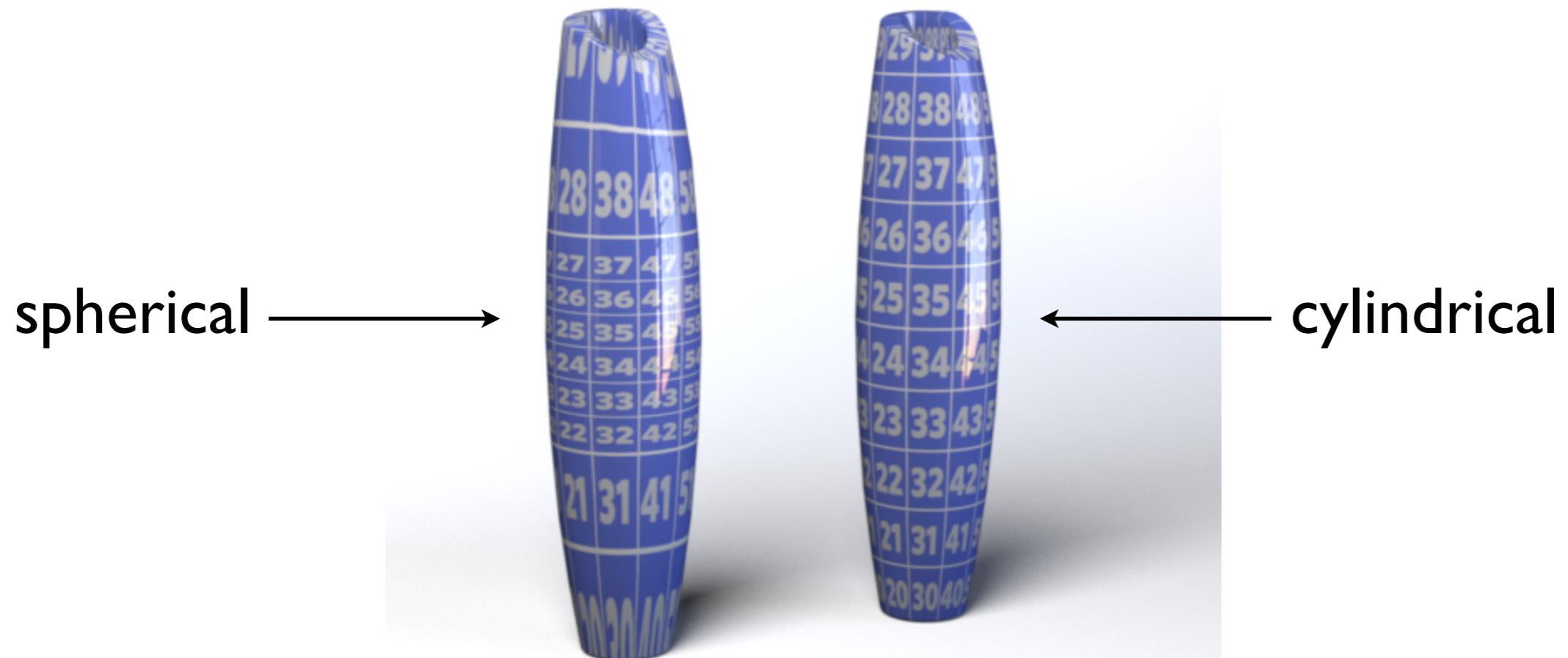
Examples of coordinate functions

- **Spherical projection**



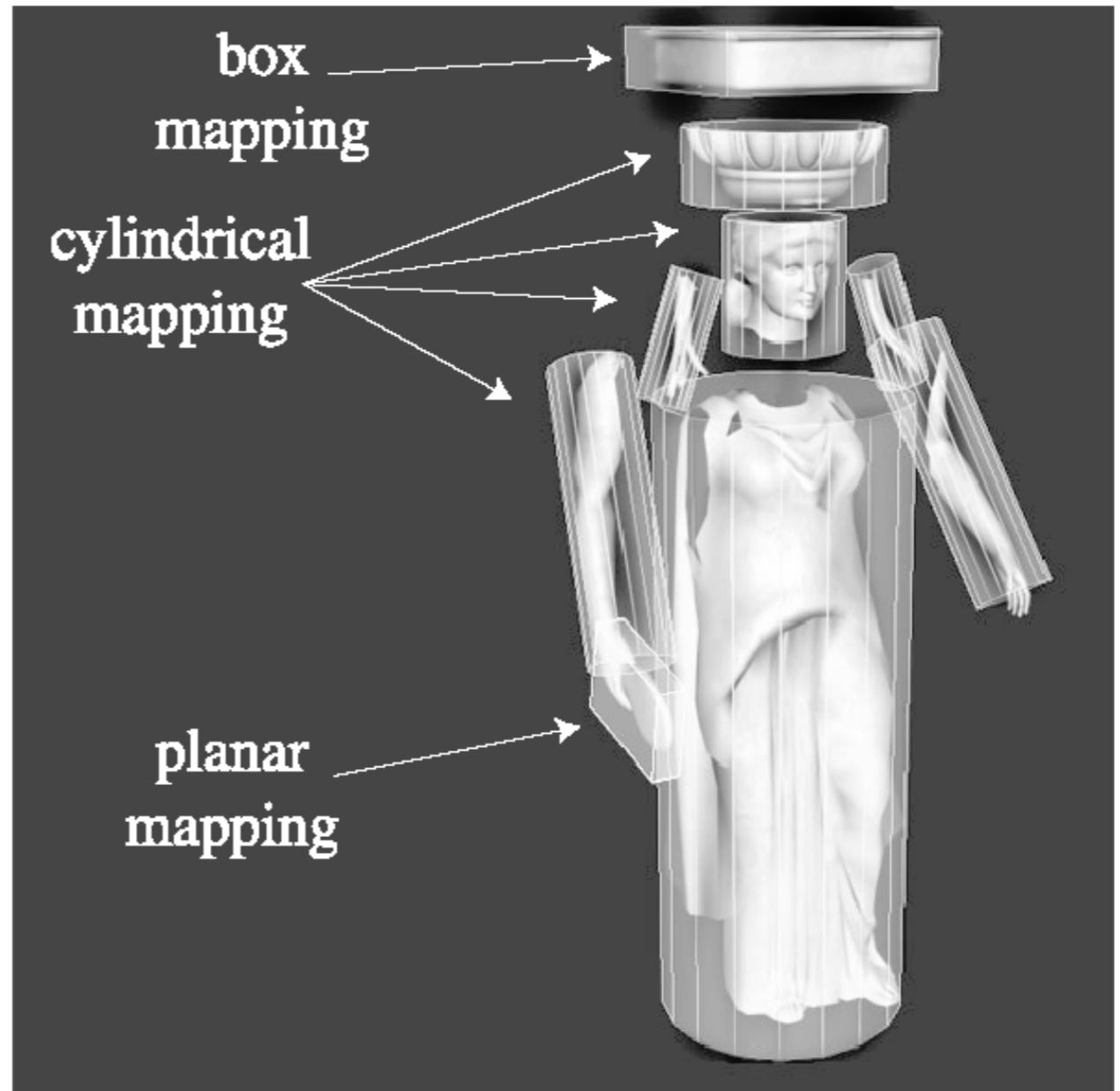
Examples of coordinate functions

- **Cylindrical projection**



Examples of coordinate functions

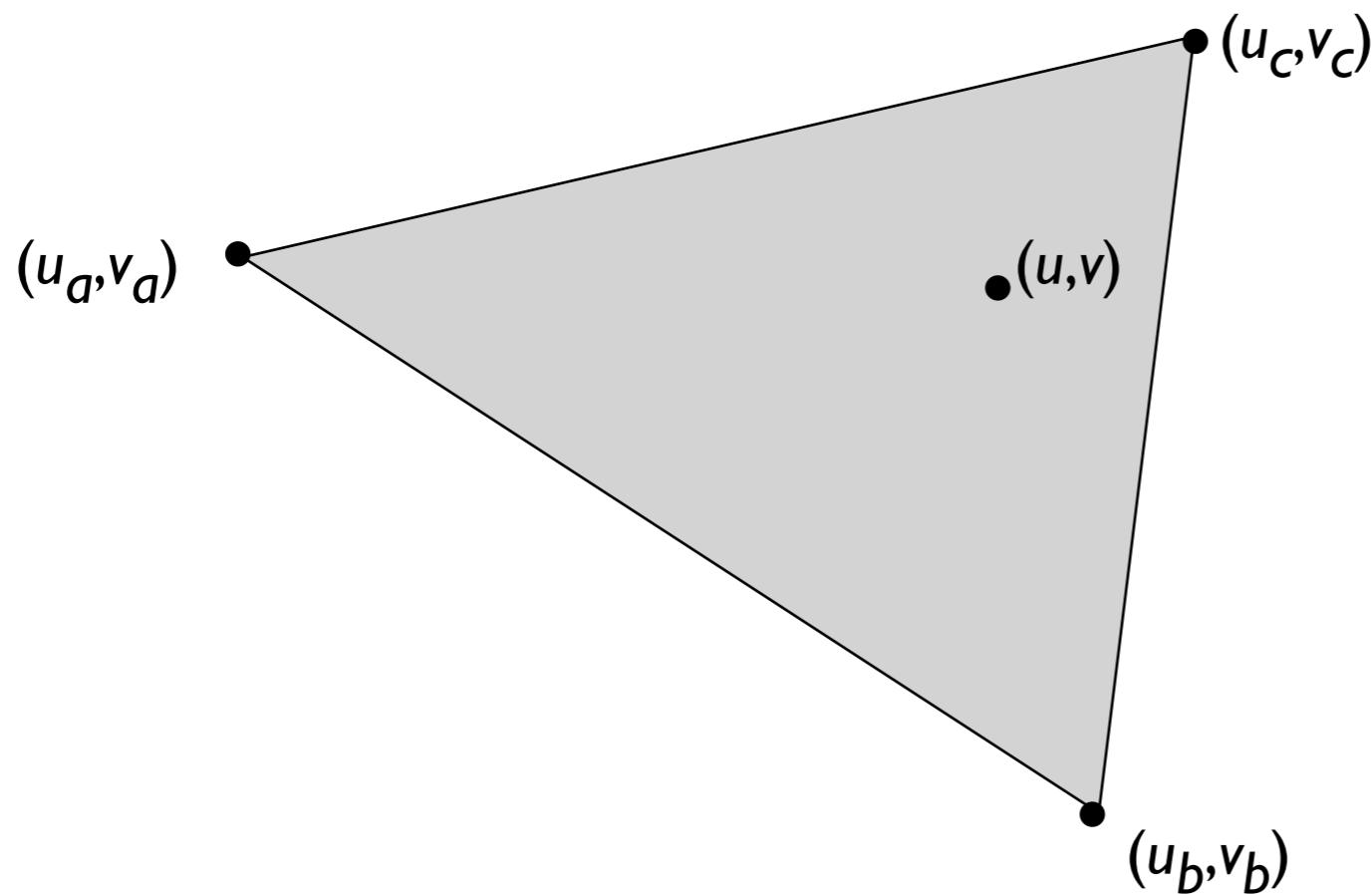
- **Complex surfaces: project parts to parametric surfaces**



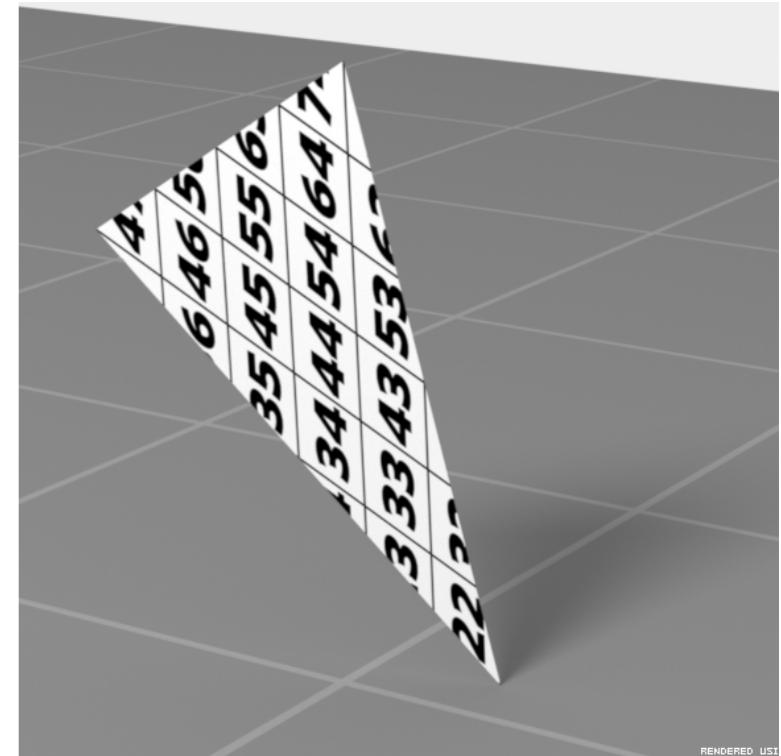
[Tito Pagan]

Examples of coordinate functions

- **Triangles**
 - specify (u,v) for each vertex
 - define (u,v) for interior by linear (barycentric) interpolation



09	19	29	39	49	59	69	79	89	99
08	18	28	38	48	58	68	78	88	98
07	17	27	37	47	57	67	77	87	97
06	16	26	36	46	56	66	76	86	96
05	15	25	35	45	55	65	75	85	95
04	14	24	34	44	54	64	74	84	94
03	13	23	33	43	53	63	73	83	93
02	12	22	32	42	52	62	72	82	92
01	11	21	31	41	51	61	71	81	91
00	10	20	30	40	50	60	70	80	90

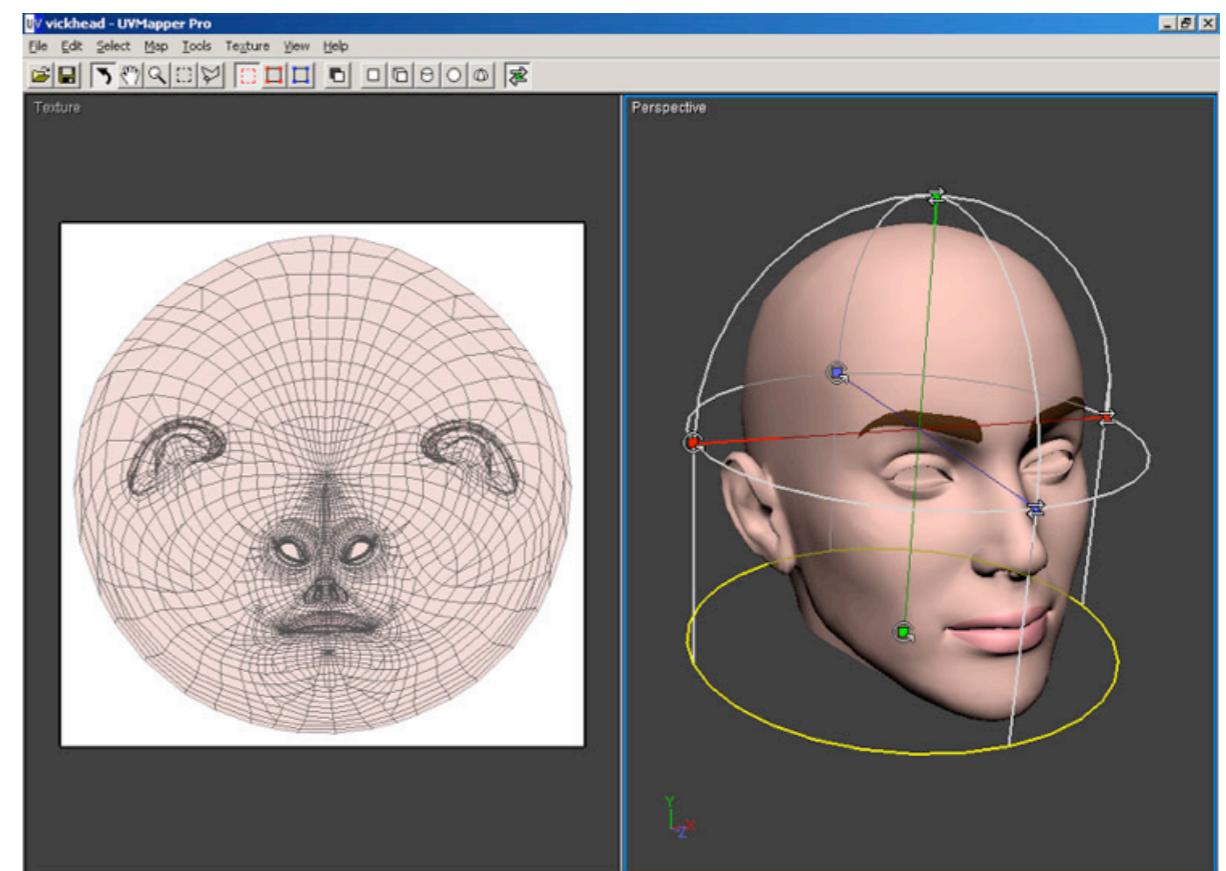
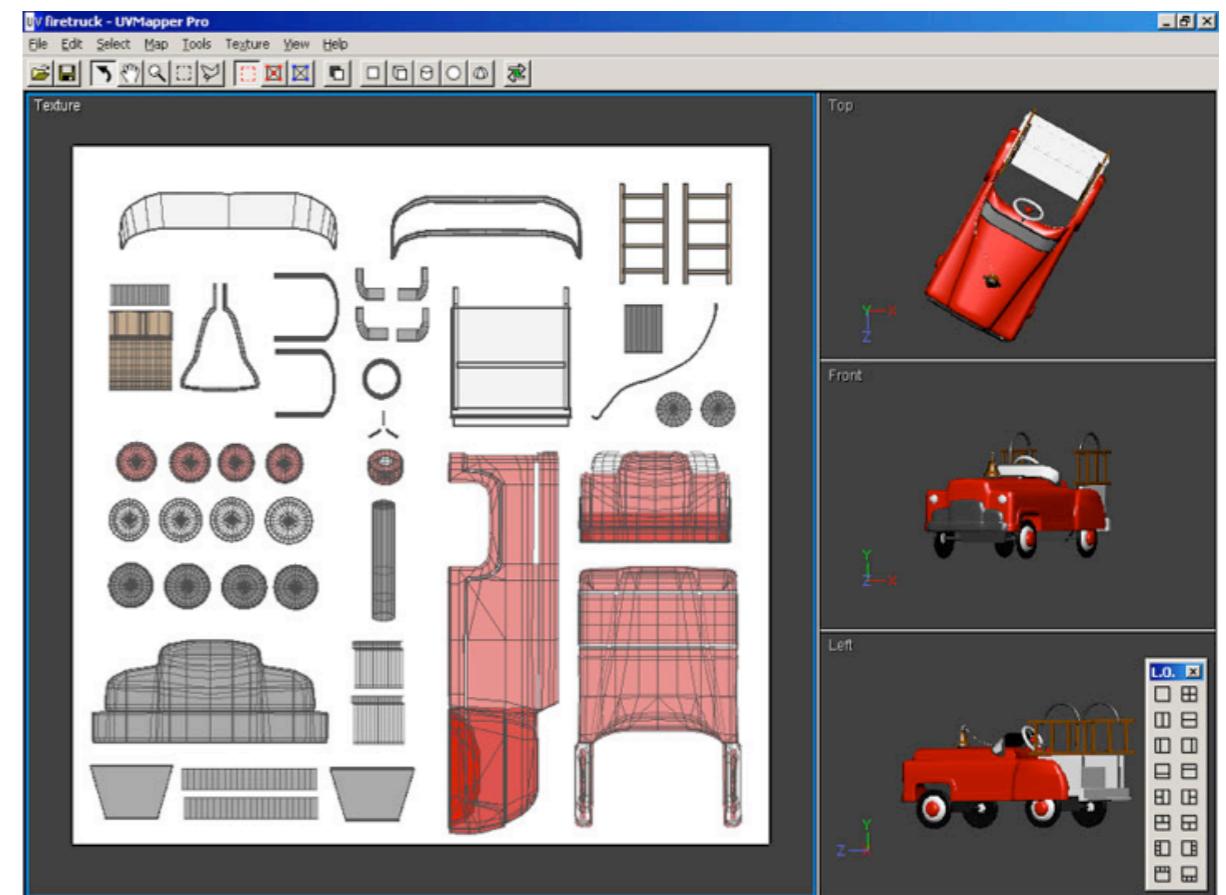
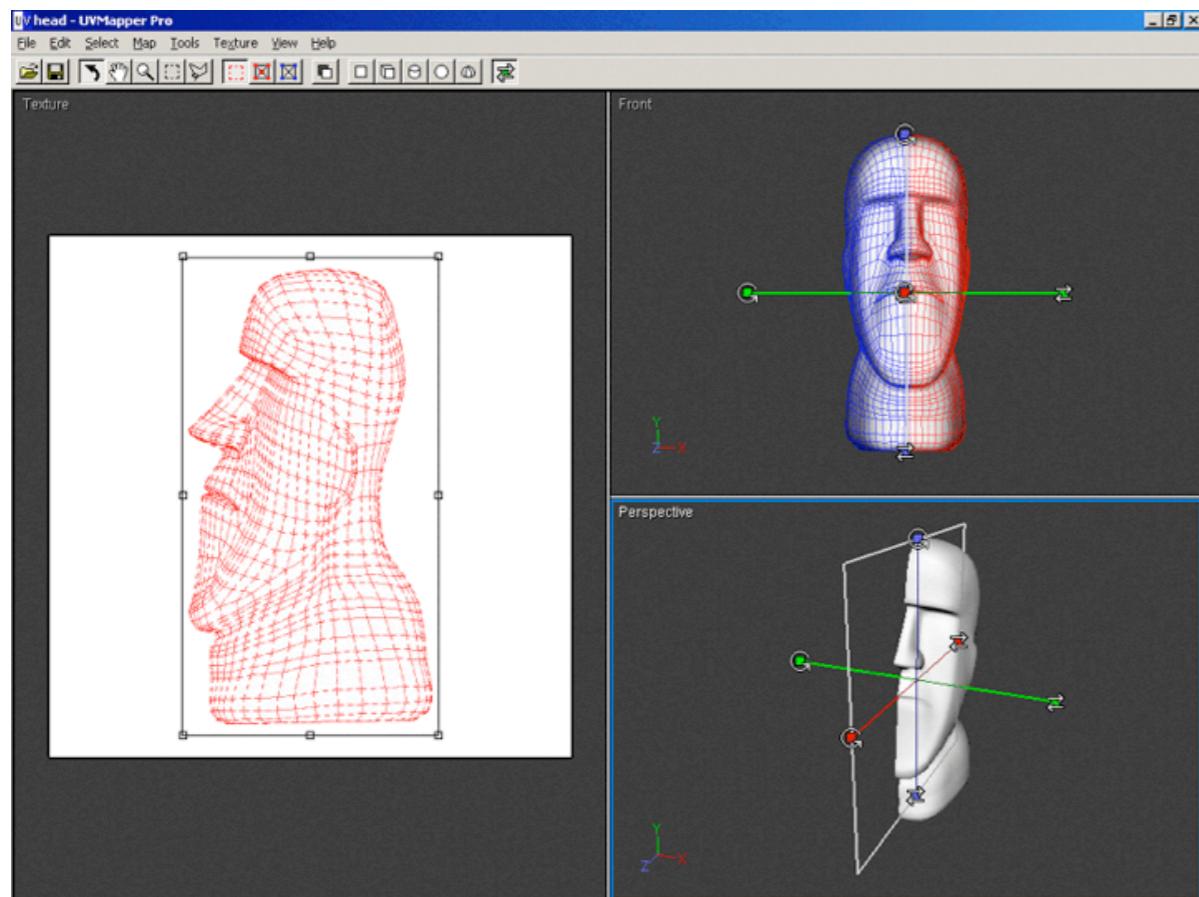


Texture coordinates on meshes

- **Texture coordinates become per-vertex data like vertex positions**
 - can think of them as a second position: each vertex has a position in 3D space and in 2D texture space
- **How to come up with vertex (u,v) s?**
 - use any or all of the methods just discussed
 - in practice this is how you implement those for curved surfaces approximated with triangles
 - use some kind of numerical optimization
 - try to choose vertex (u,v) s to result in a smooth, low distortion map

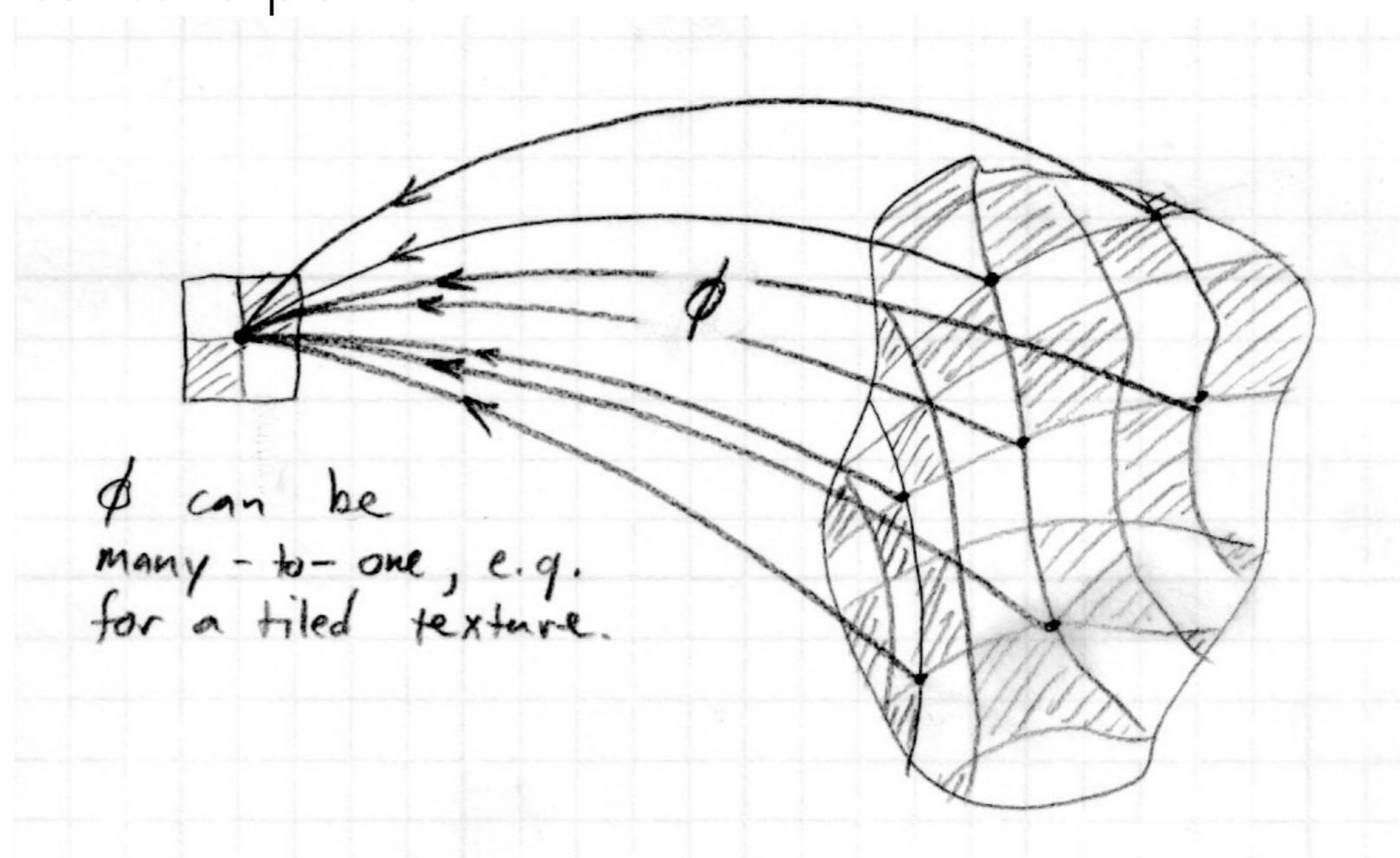
Example: UVMapper

<http://www.uvmapper.com>



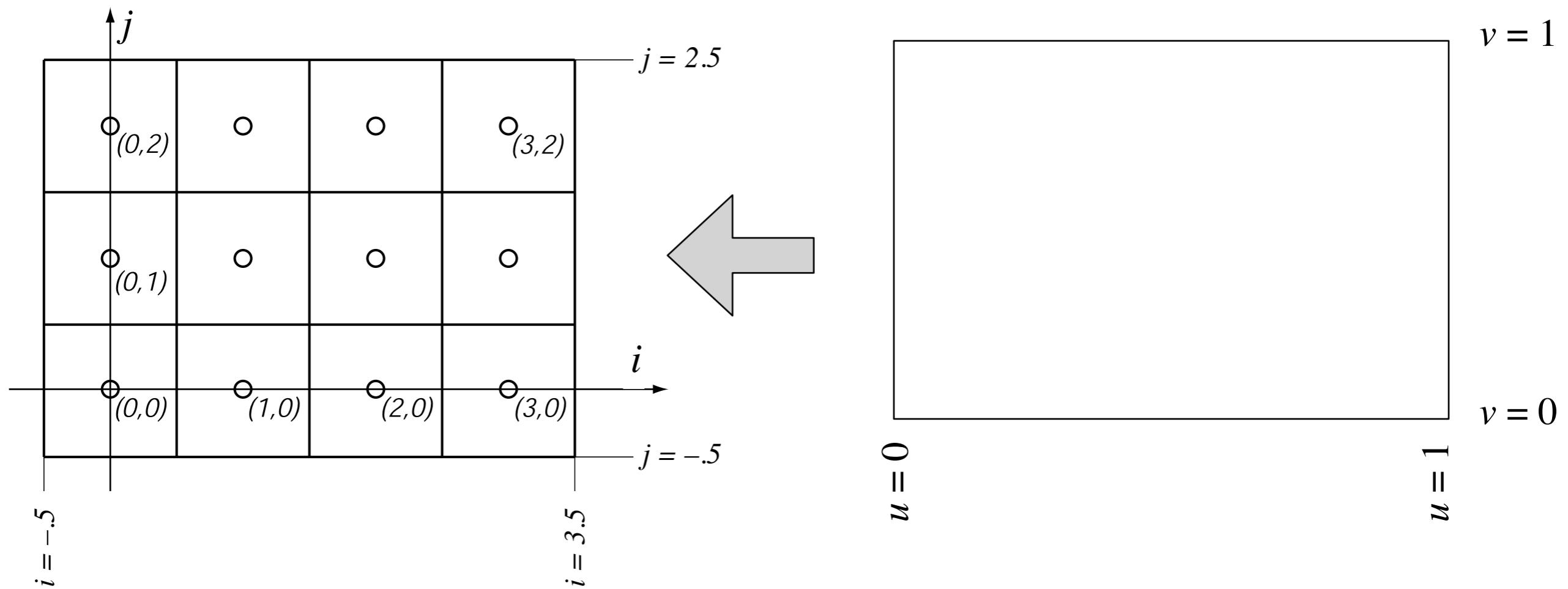
Texture coordinate functions

- **Mapping from S to D can be many-to-one**
 - that is, every surface point gets only one color assigned
 - but it is OK (and in fact useful) for multiple surface points to be mapped to the same texture point
 - e.g. repeating tiles



Pixels in texture images (texels)

- **Texels relate to texture coordinates in the same way as pixels relate to normalized image coordinates**



$$i = \text{round}(u n_u - 0.5) = \text{floor}(u n_u)$$

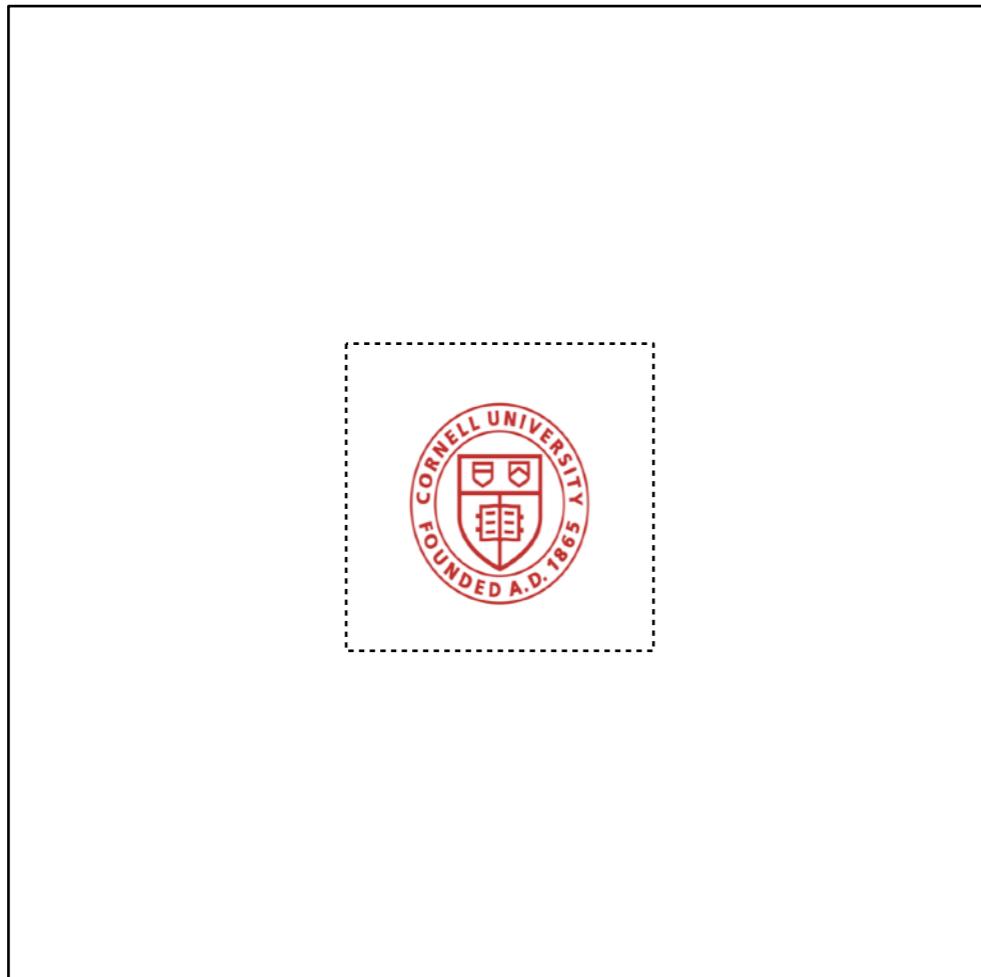
$$j = \text{round}(v n_v - 0.5) = \text{floor}(v n_v)$$

for a texture image
of n_u by n_v pixels

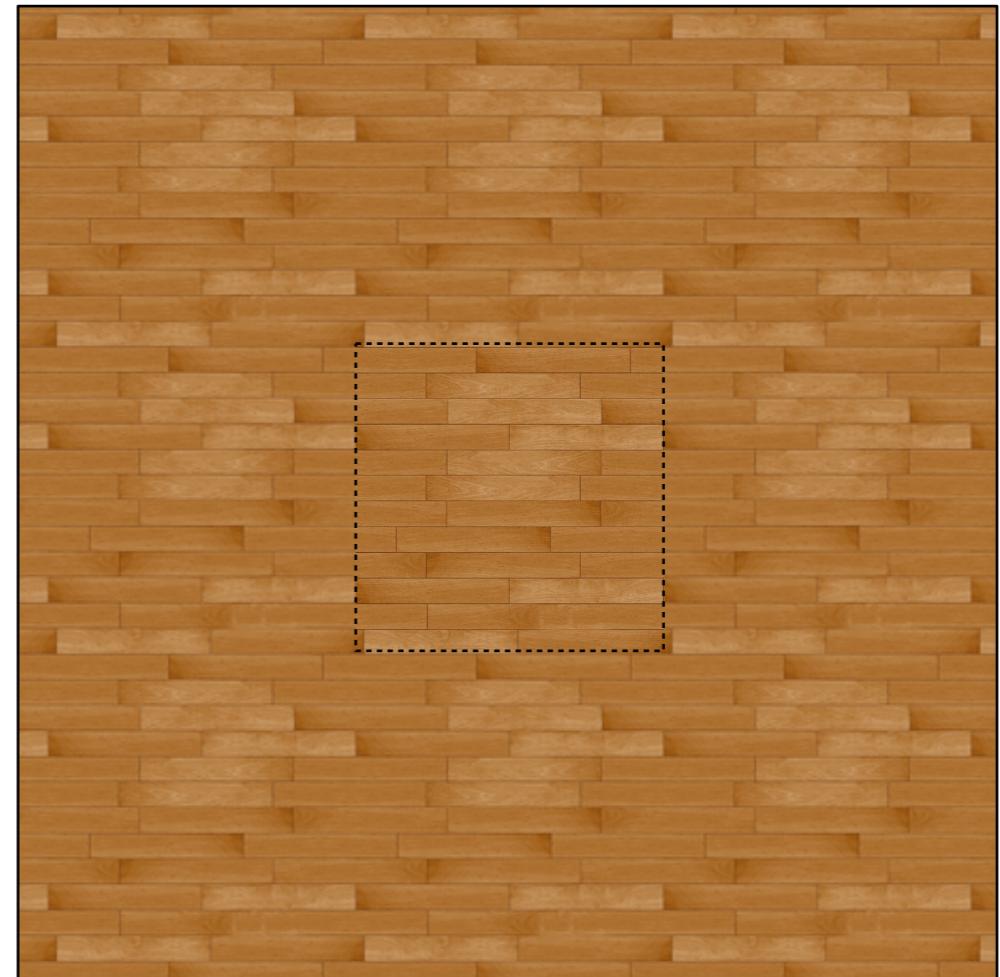
Texture lookups and wrapping

- In shading calculation, when you need a texture value you perform a *texture lookup*
- Convert (u, v) texture coordinates to (i, j) texel coordinates, and read a value from the image
- What if i and j are out of range?
 - option 1, clamp: take the nearest pixel that is in the image
$$i_{\text{pixel}} = \max(0, \min(n_x - 1, i_{\text{lookup}}))$$
 - option 2, wrap: treat the texture as periodic, so that falling off the right side causes the look up to come in the left
$$i_{\text{pixel}} = \text{remainder}(i_{\text{lookup}}, n_x)$$

Wrapping modes



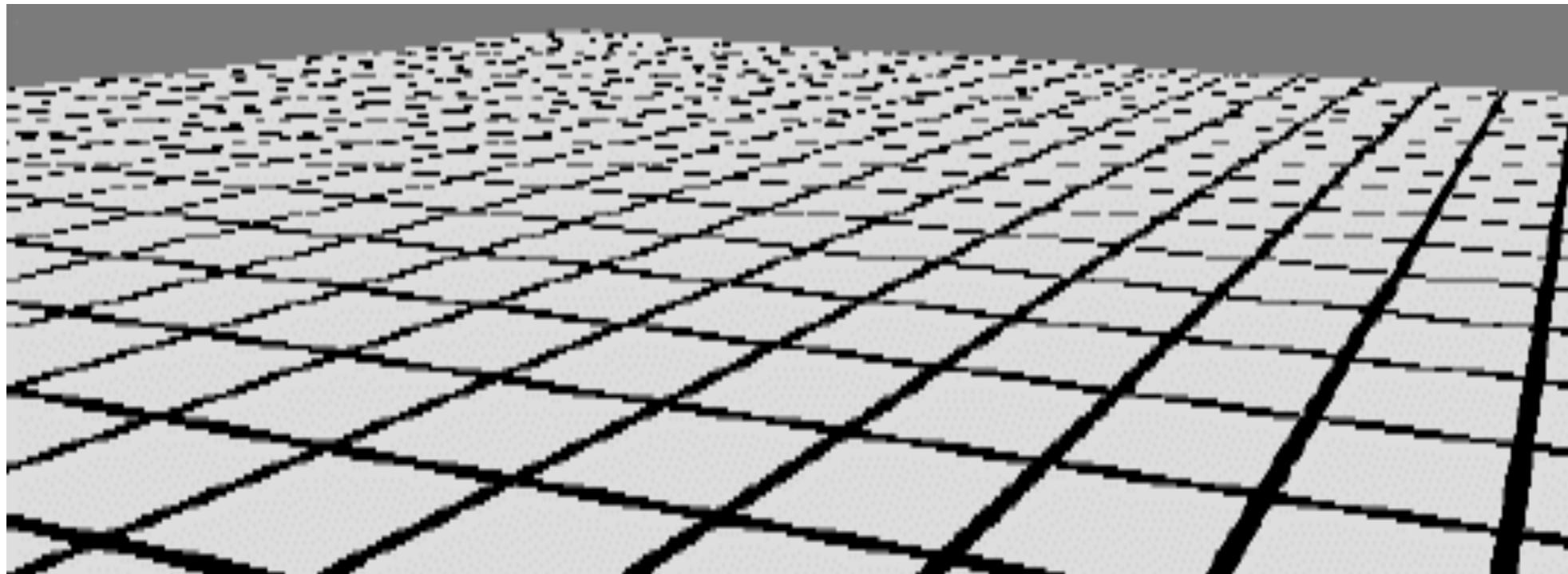
clamp



wrap

Antialiasing in textures

- Even with multisampling, we still only evaluate textures once per fragment
- Need to filter the texture somehow!
 - perspective produces very fine subpixel detail



[Akenine-Möller & Haines 2002]

Texture mapping from 0 to infinity

- When you go close...

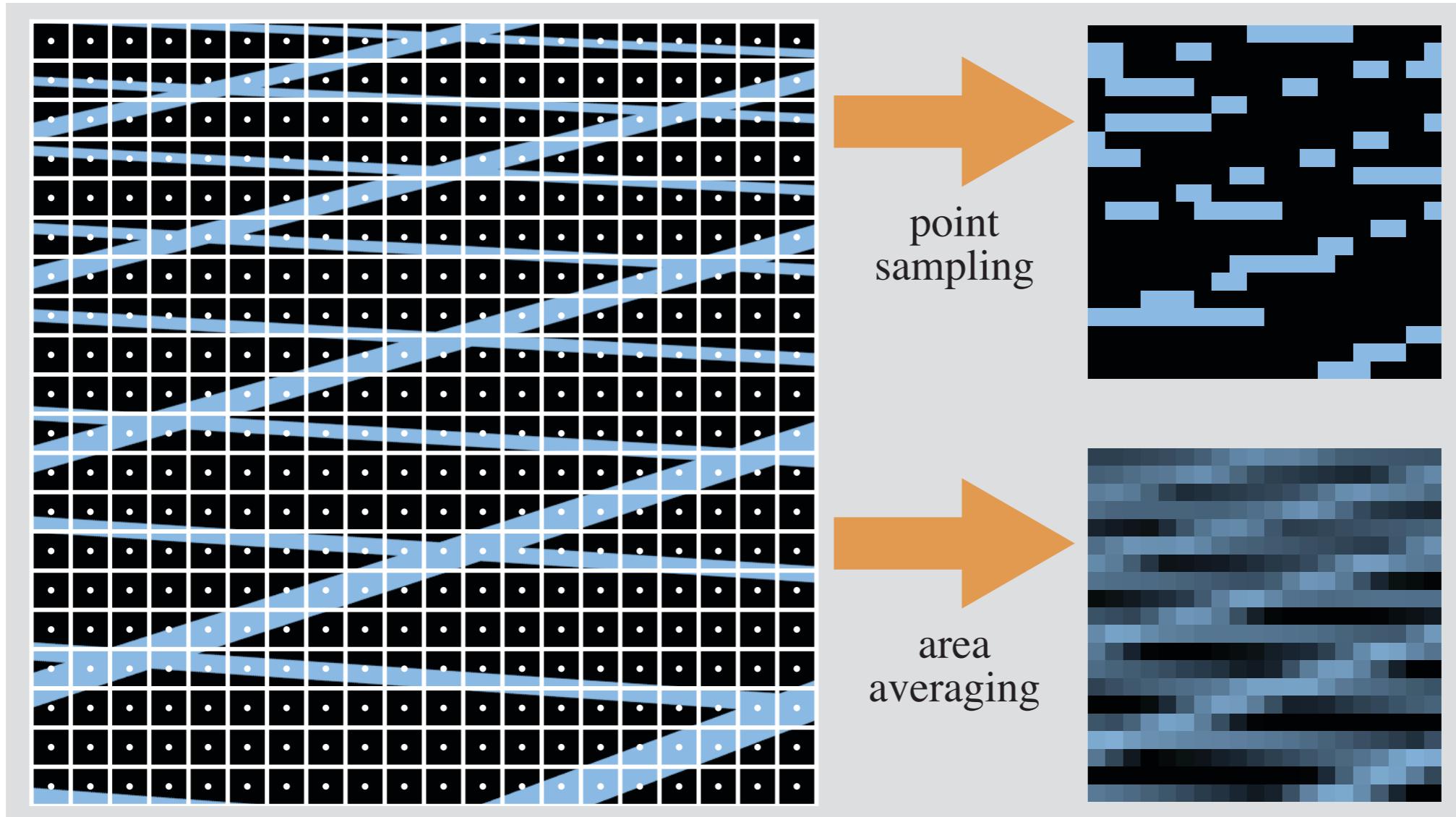


Texture mapping from 0 to infinity

- When you go far...



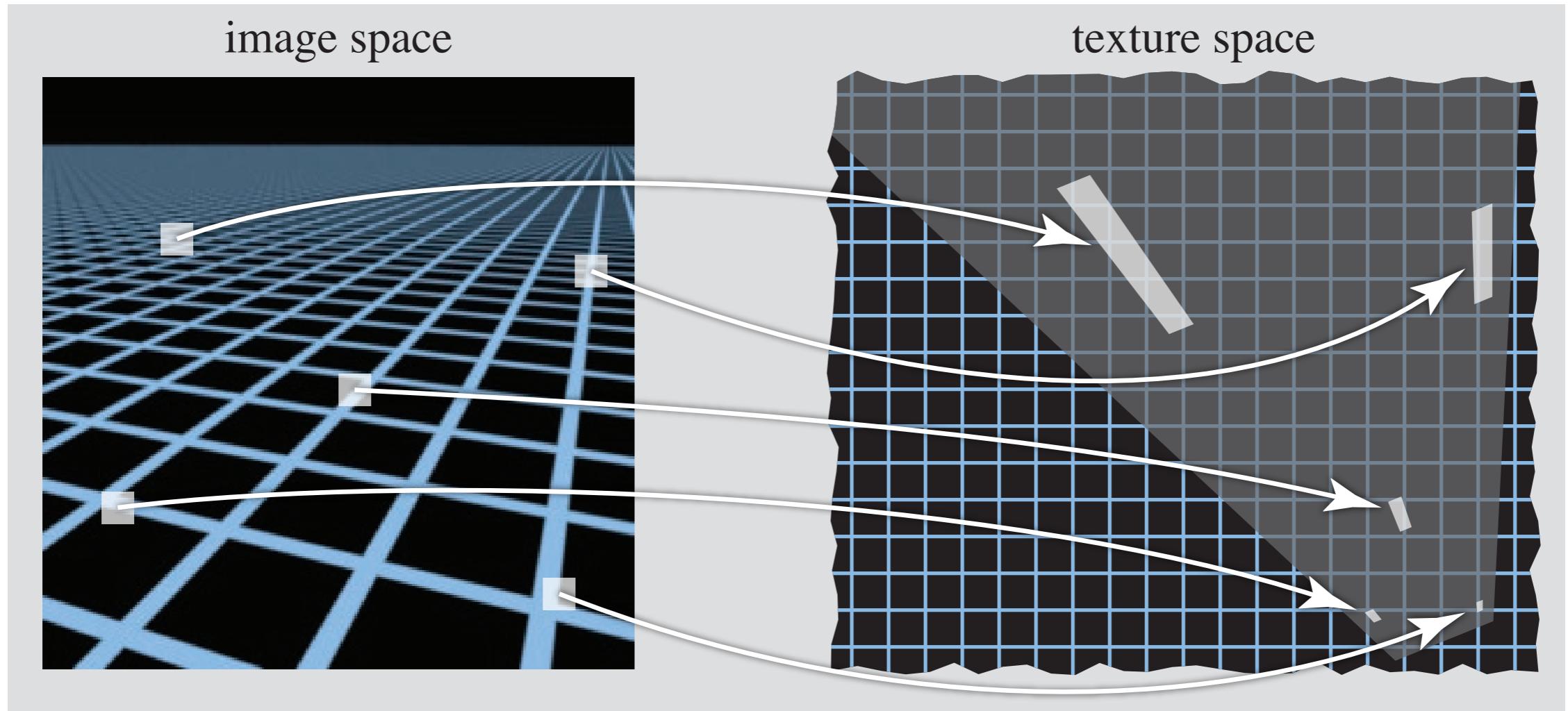
Solution: pixel filtering



Pixel filtering in texture space

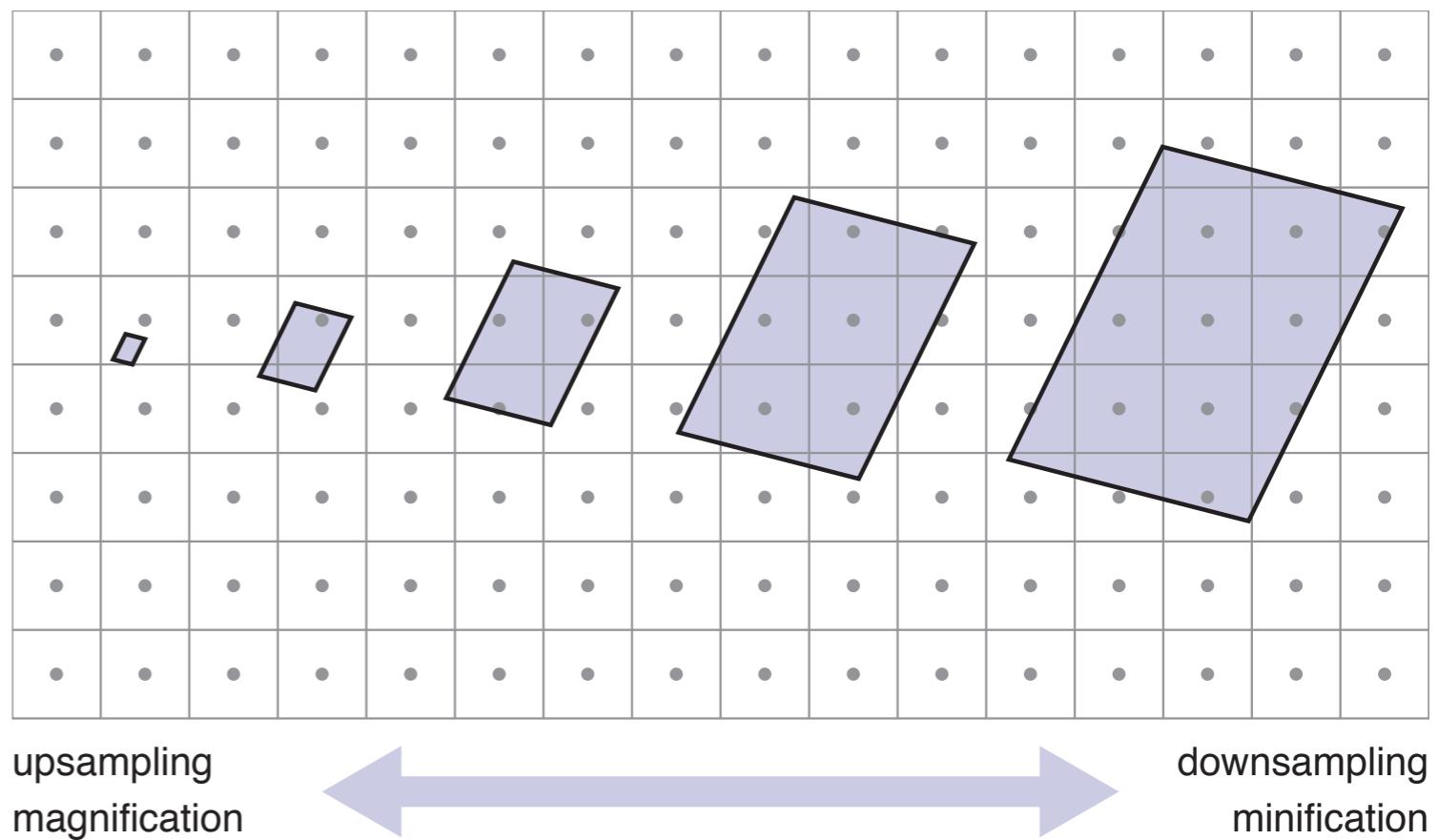
- **Sampling is happening in image space**
 - sample is a weighted average over a pixel-sized area
 - uniform, predictable, friendly!
- **Signal is defined in texture space**
 - mapping between image and texture is nonuniform
 - each sample is a weighted average over a different sized and shaped area
 - irregular, unpredictable, unfriendly!
- **This is a change of variable**
 - integrate over texture coordinates rather than image coordinates

Pixel footprints

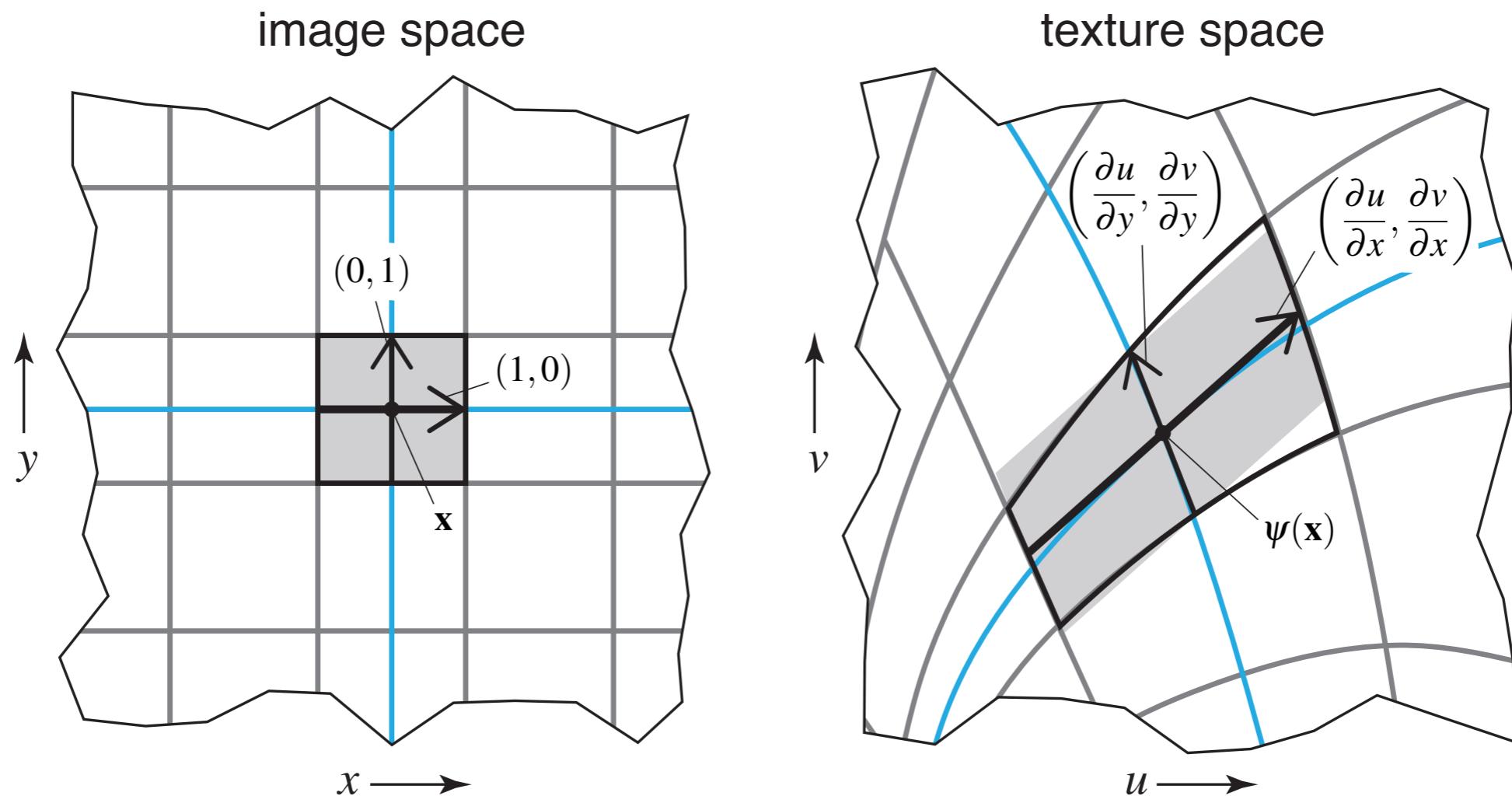


How does area map over distance?

- **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”

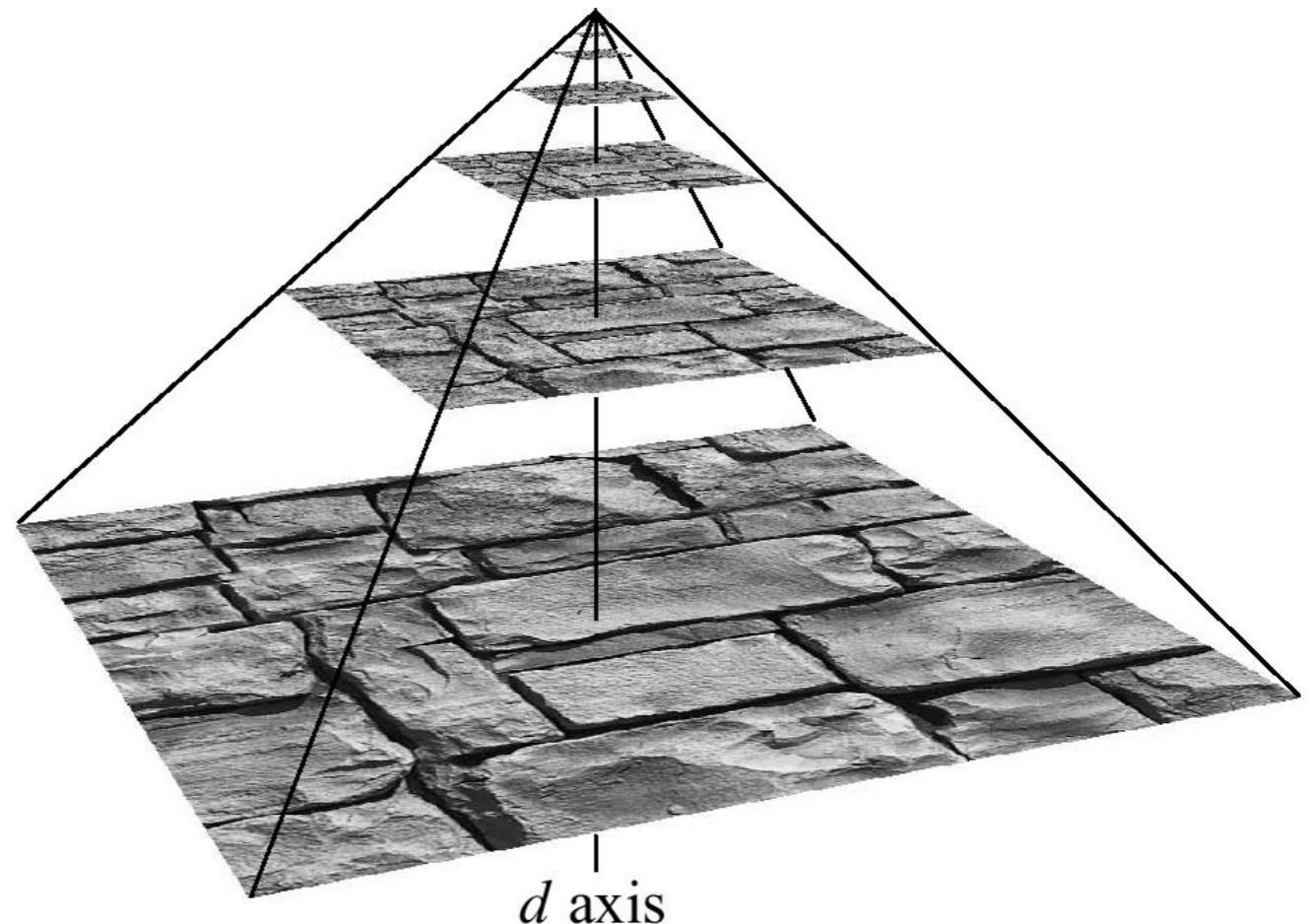


Sizing up the situation with the Jacobian



Mipmap image pyramid

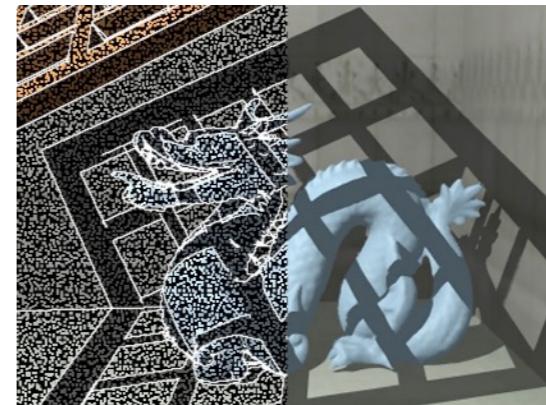
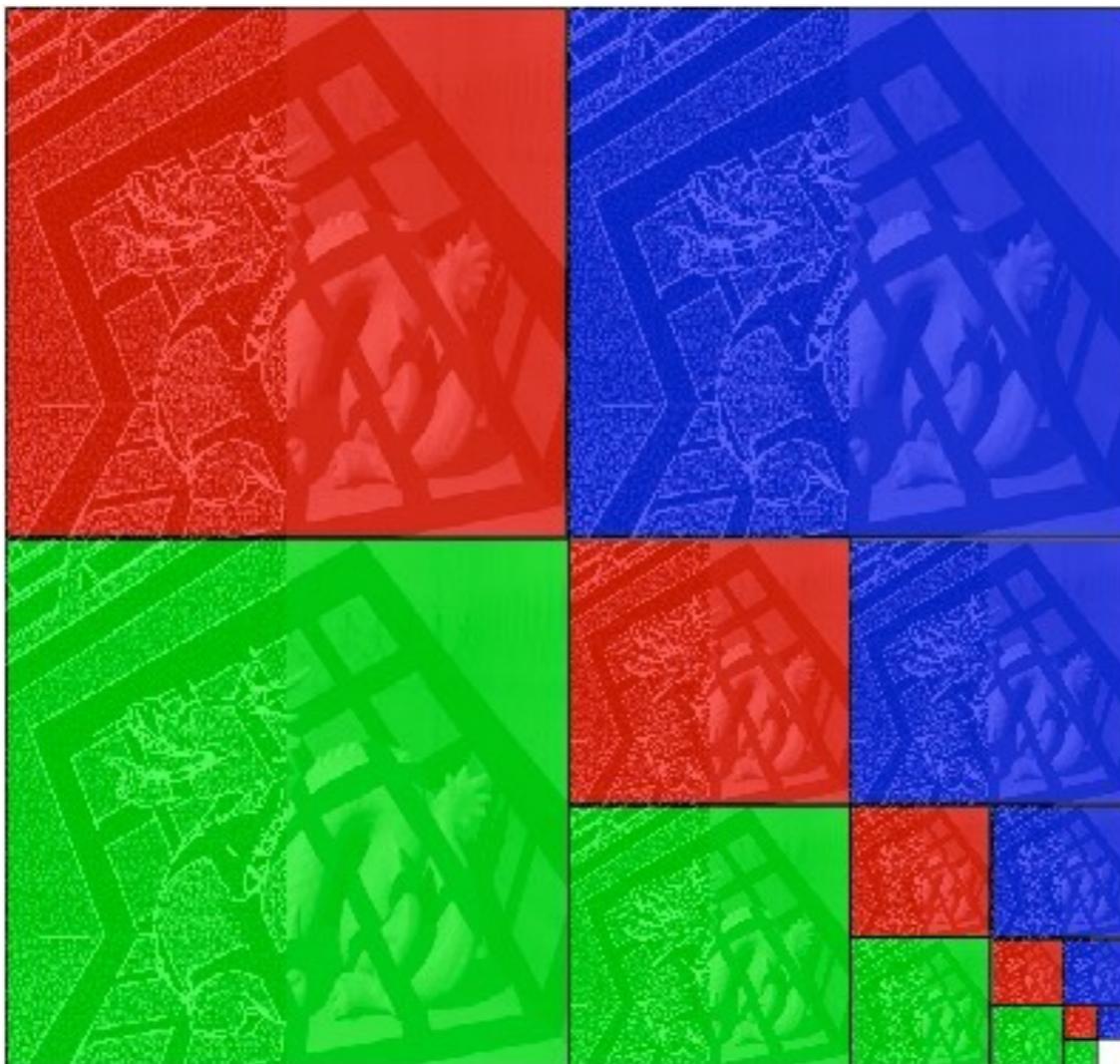
- **MIP Maps**
 - Multum in Parvo: Much in little, many in small places
 - Proposed by Lance Williams
- **Stores pre-filtered versions of texture**
- **Supports very fast lookup**
 - but only of circular filters at certain scales



[Akamine-Möller & Haines 2002]

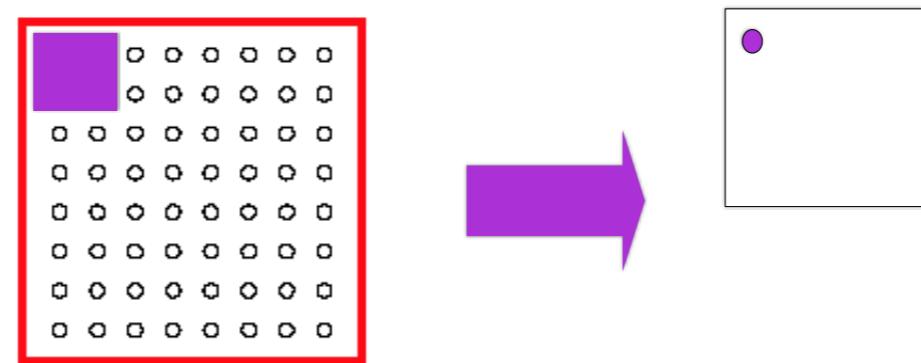
Memory Usage

- **What happens to size of texture?**



[Kavita Bala]

Building the MIP map



- **Each pixel in a level corresponds to 4 pixels in lower level**
 - Can use average (box filter) downsampling
 - Better: gaussian or other smooth filter

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
 - Use Jacobian to find pixel footprint

- x derivative:

$$\frac{\partial u}{\partial x} \quad \frac{\partial v}{\partial x}$$

- y derivative:

$$\frac{\partial u}{\partial y} \quad \frac{\partial v}{\partial y}$$

Given derivatives: what is level?

- **A compromise: reduce jacobian matrix to a single number**
 - two popular ways:

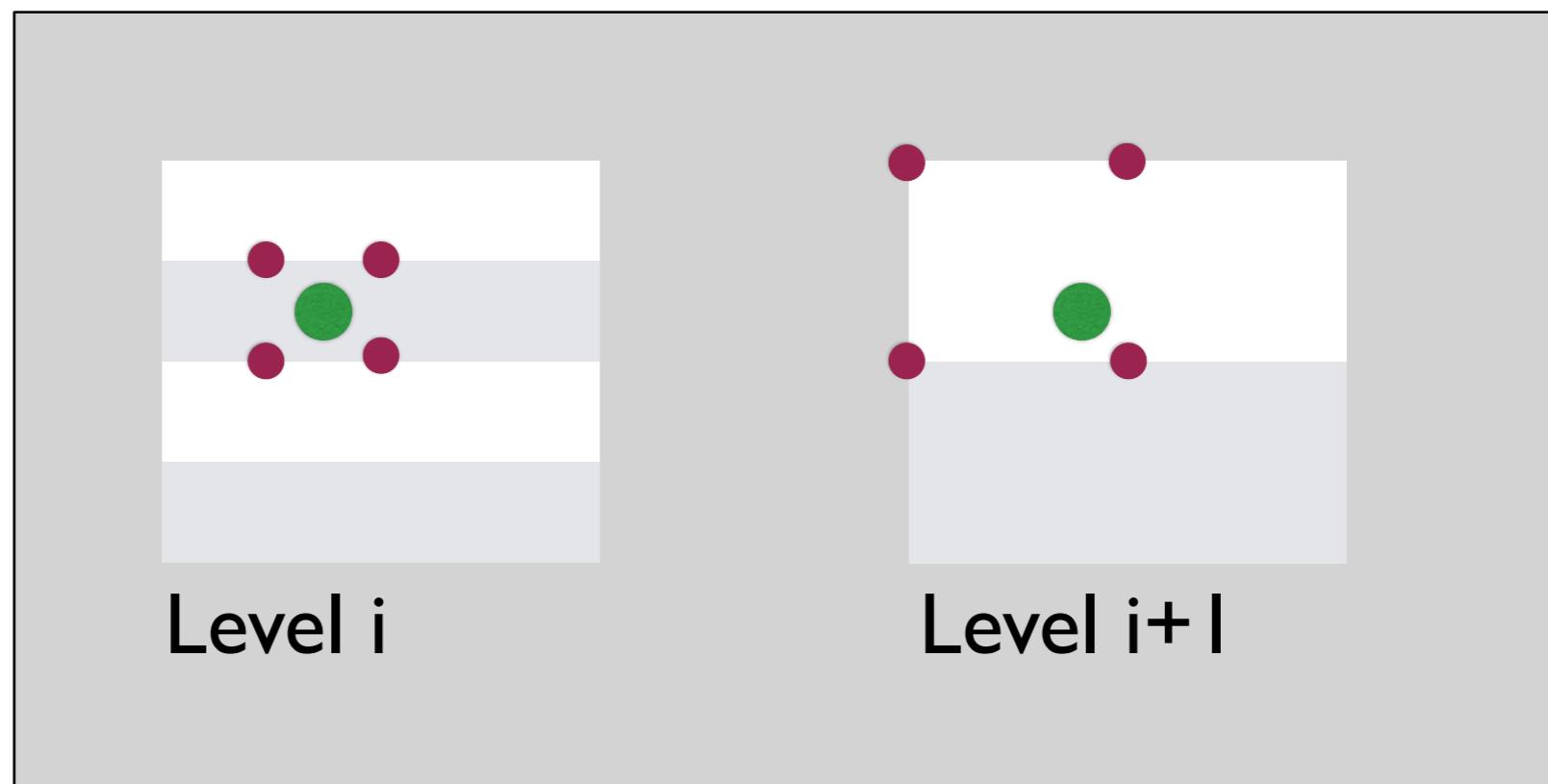
$$level = \log[\max\left(\frac{du}{dx}, \frac{dv}{dx}, \frac{du}{dy}, \frac{dv}{dy}\right)]$$

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

- MIP map pixels are 2^k in size at level k
 - so level is log-base-2 of the footprint size

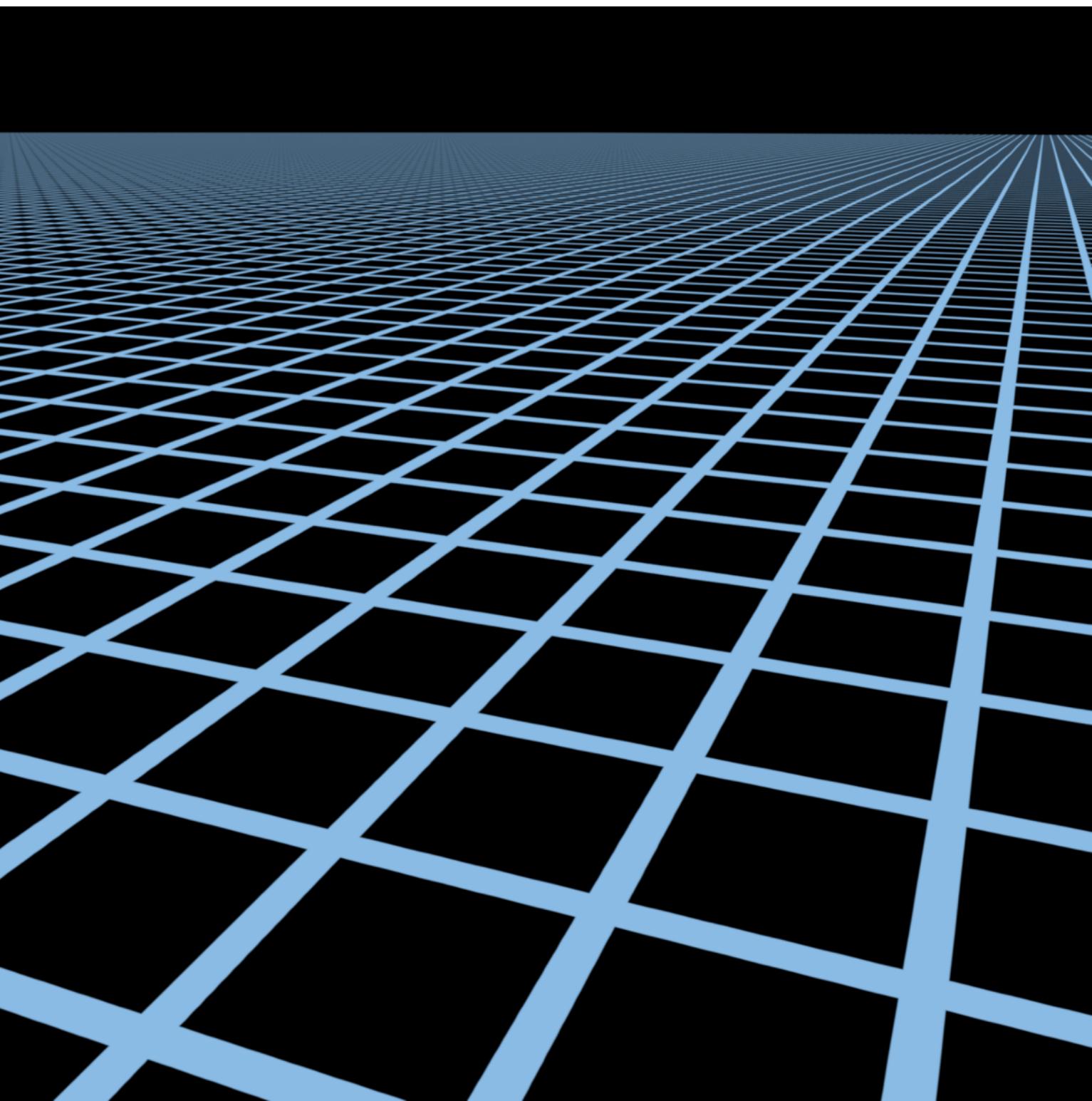
Using the MIP Map

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

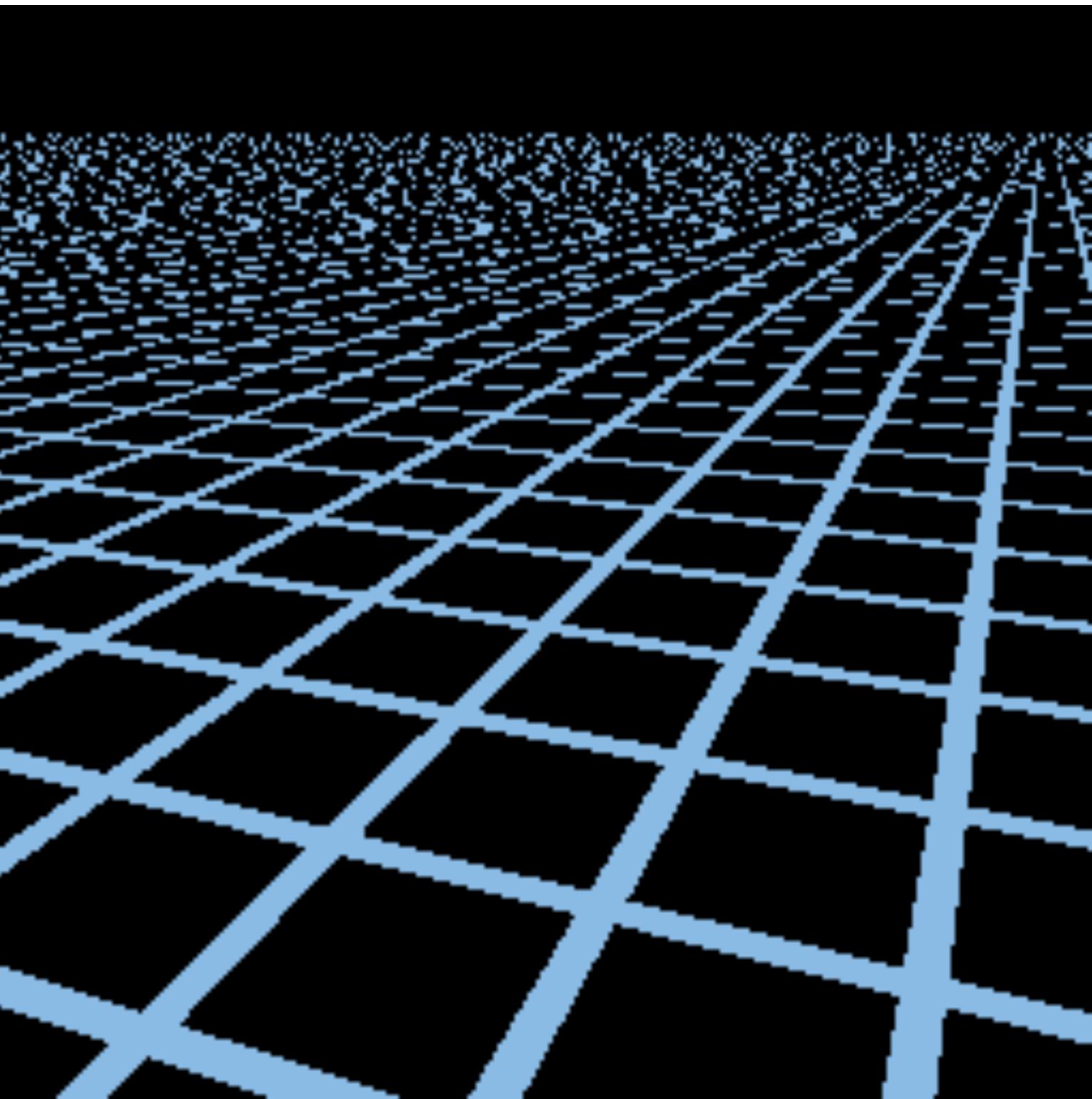


[Kavita Bala]

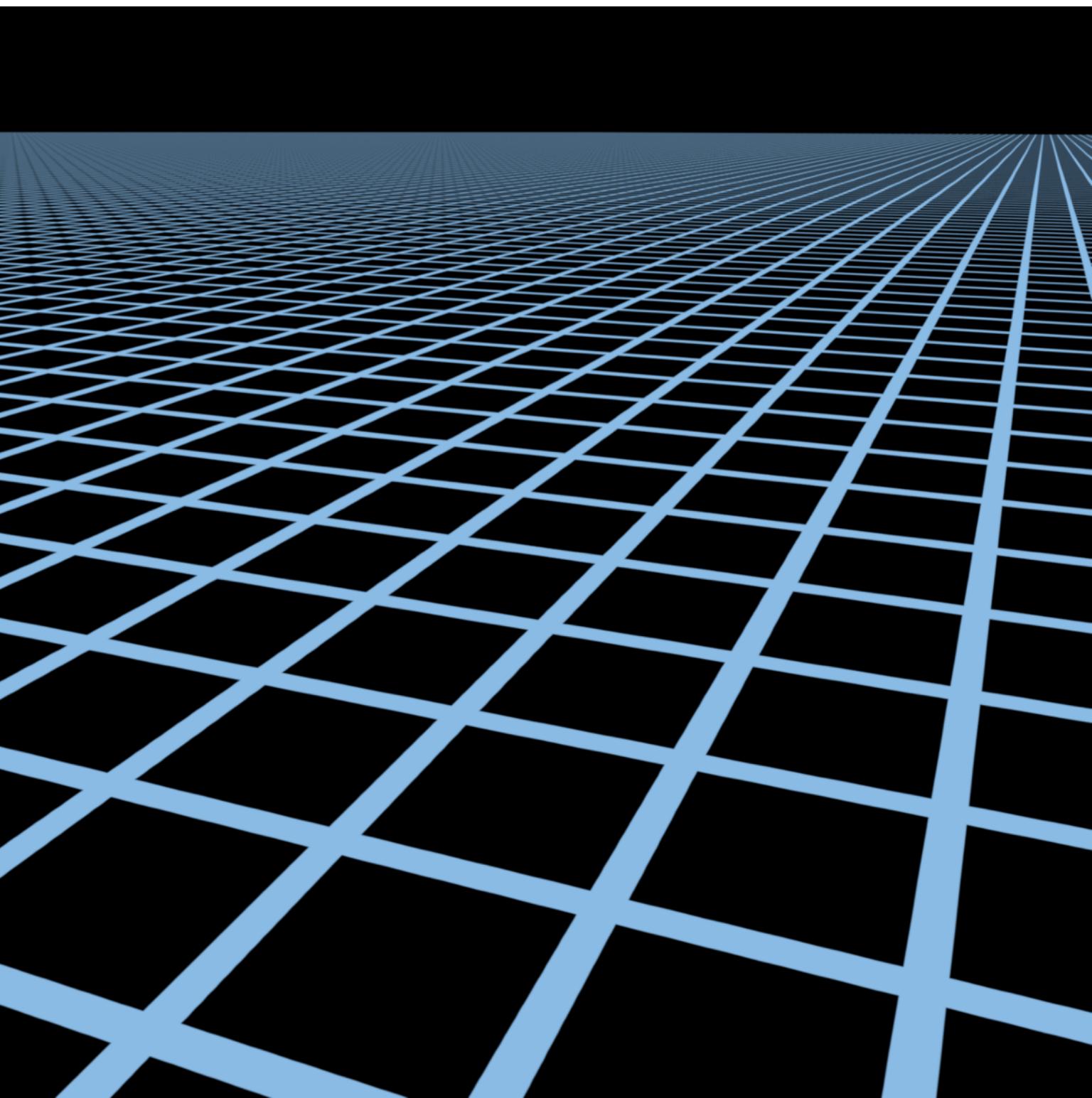
Point sampling



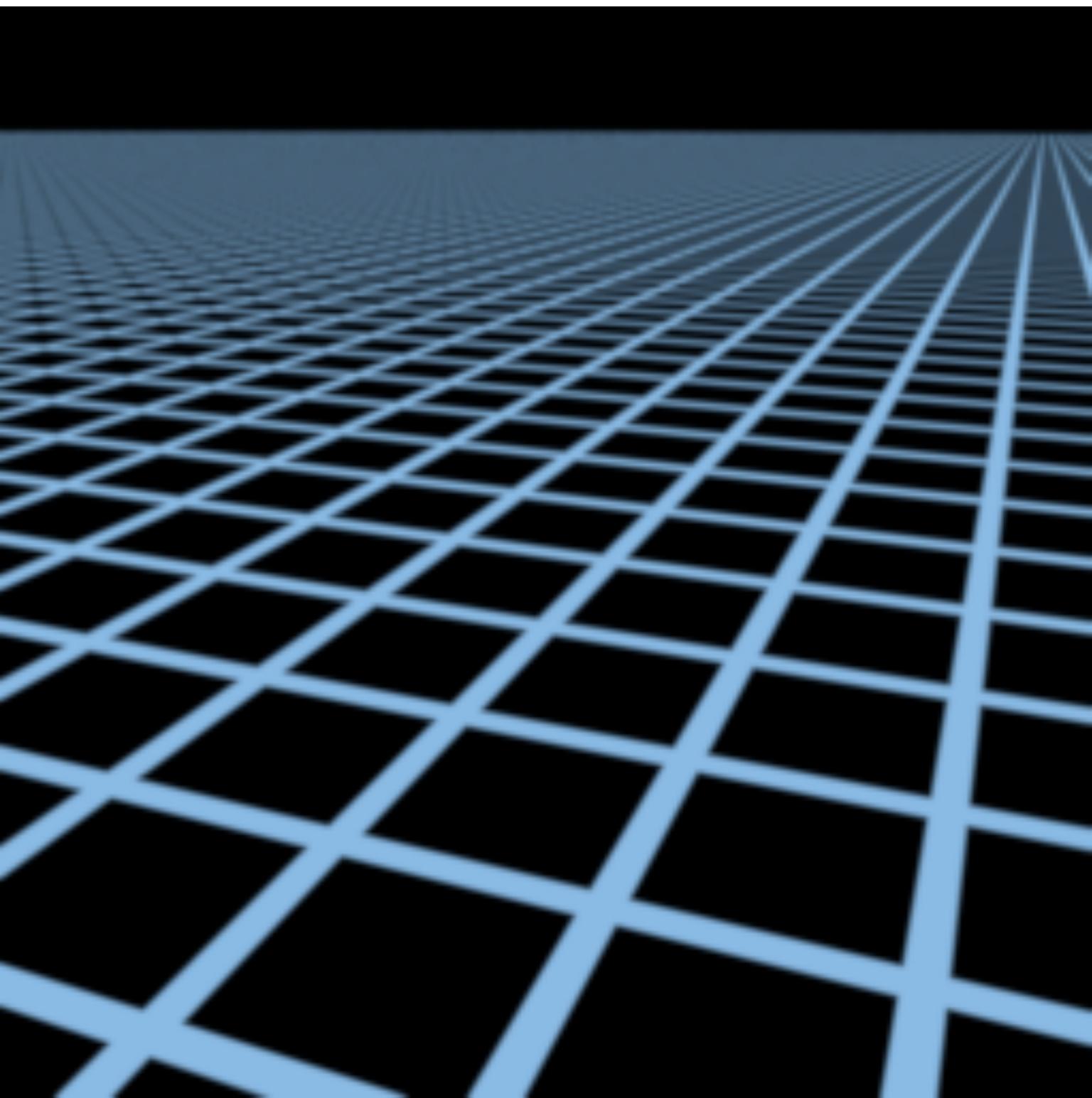
Point sampling



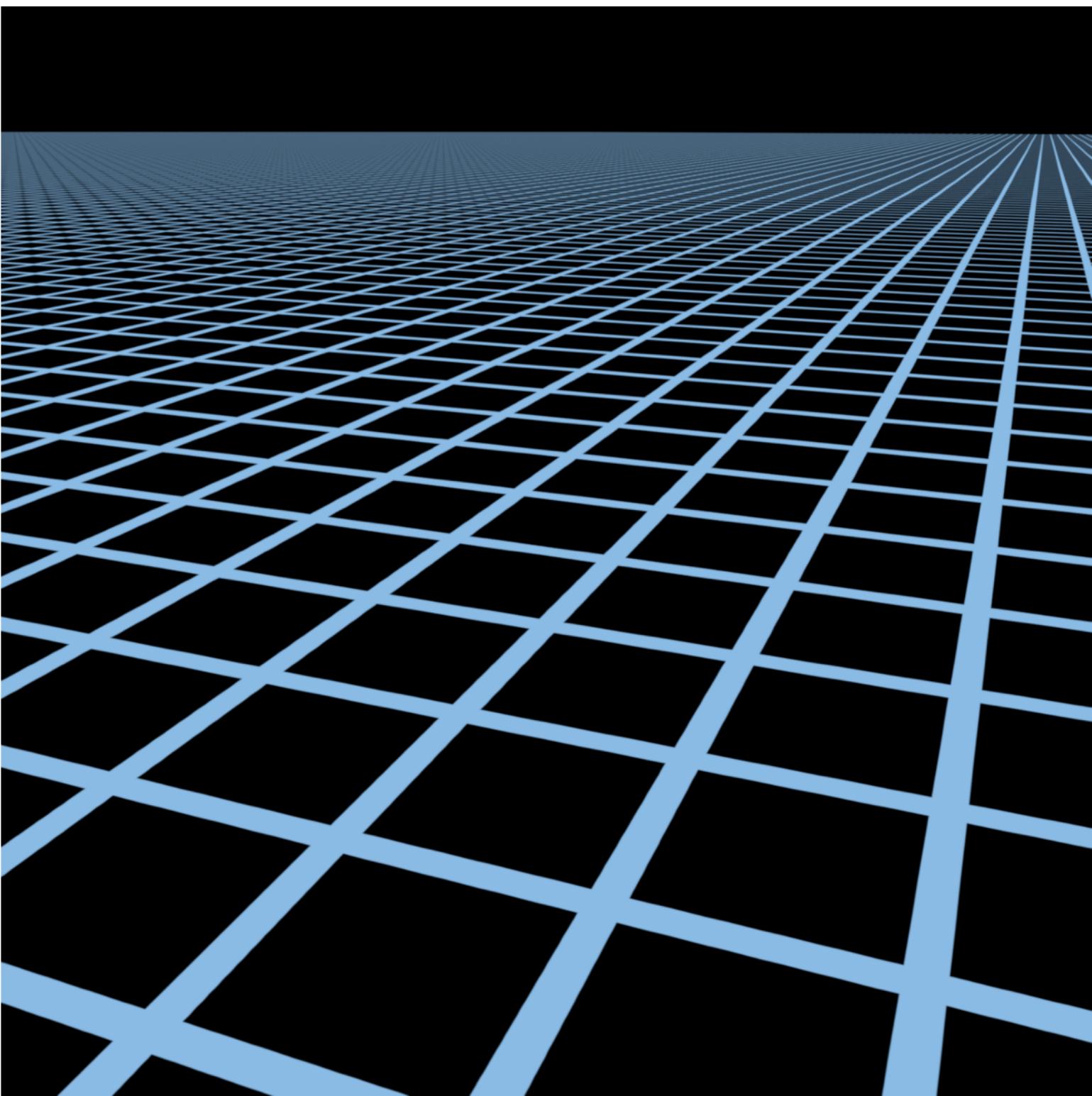
Reference: gaussian
sampling by



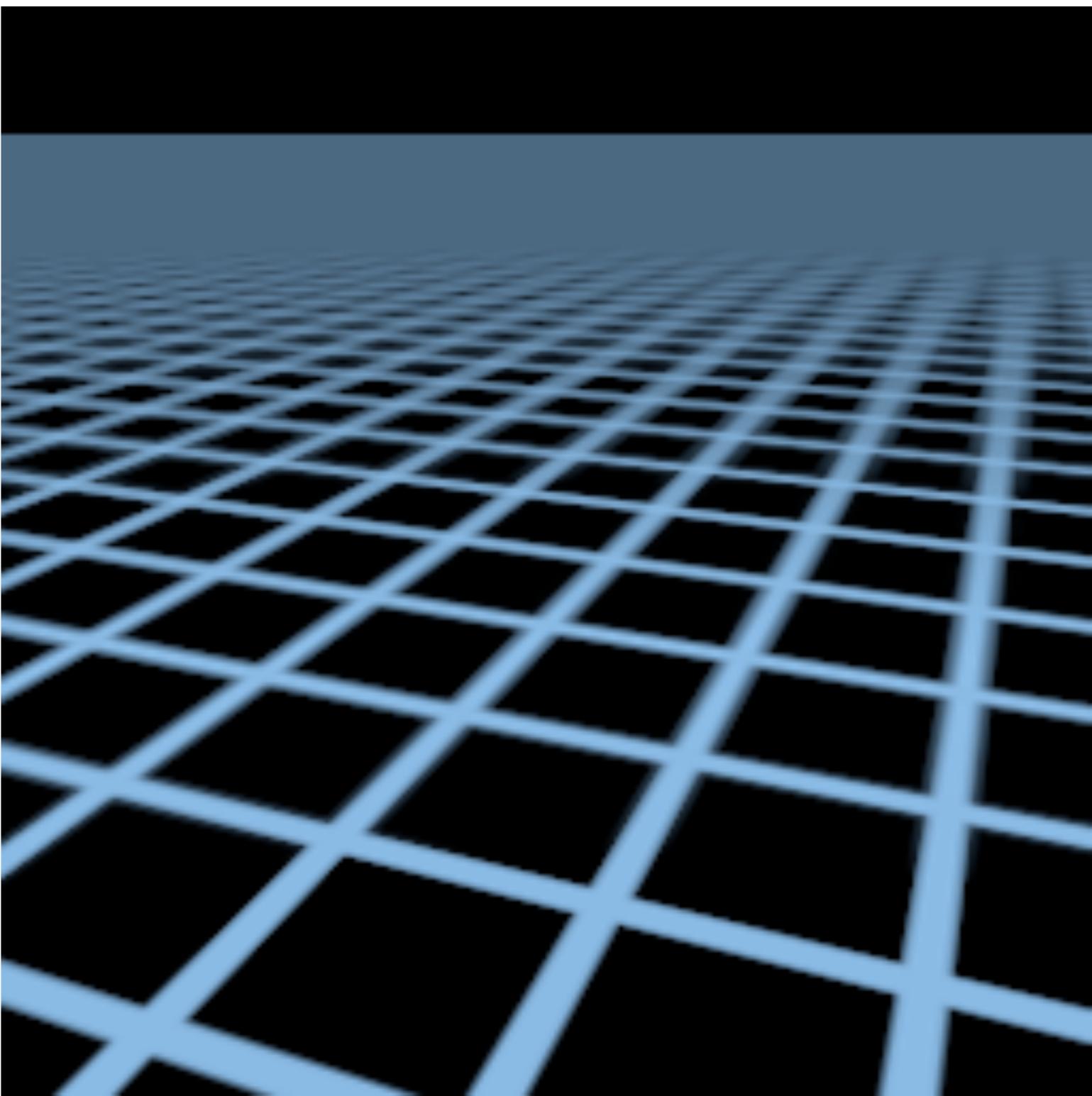
Reference: gaussian
sampling by



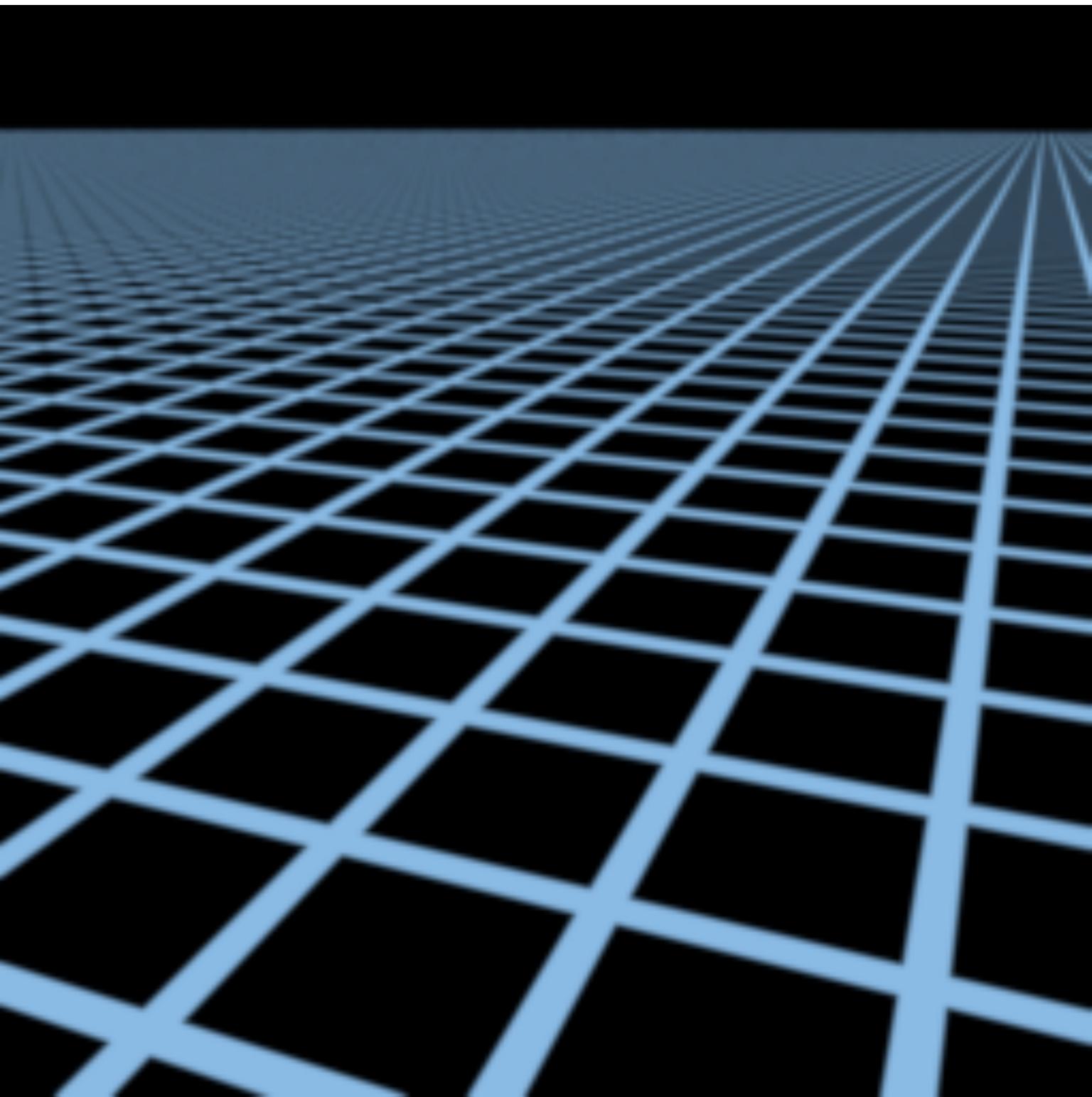
Texture minification with a mipmap



Texture minification with a mipmap



Texture minification: supersampling



Texture minification: supersampling

