CSC 433/533
Computer Graphics

Alon Efrat
Credit: Joshua Levine

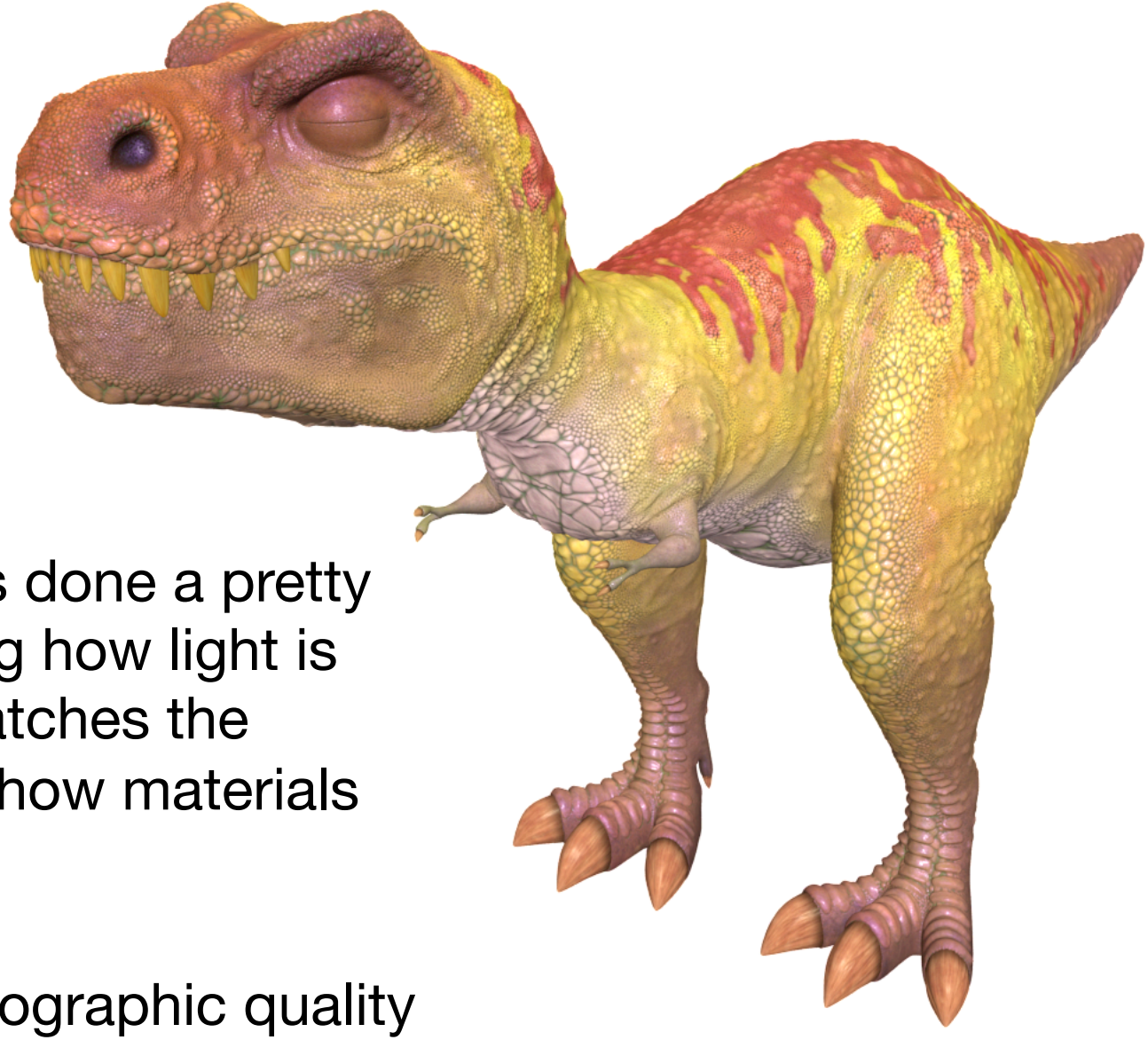
Lecture 22
Textures

Today's Agenda

- Reminders:
  - A06 Questions?
- Goals for today: Discuss textures!

Textures

Challenge: Real World Surfaces Have Complex Materials



- While ray tracing has done a pretty good job of capturing how light is modeled, it only scratches the surface at modeling how materials look

- Goal: Replicate photographic quality by varying shading parameters?

http://ptex.us/ptexpaper.html

Texture Mapping

Texture Mapping

- Models attributes of surfaces that **vary as position changes**, but do not affect the shape of the surface.

- Examples: wood grain, wrinkles in skin, woven structures in cloth, defects in metal surfaces, patterns (in general), …

Texture Maps

- Idea: model this variation using an image, called a **texture map** (or, sometimes “texture image” or just “texture”)
- The texture map stores the surface details
  - Typically, shading parameters like k_d and k_s
- Can be used in lots of interesting ways to achieve complex effects

Meshlab Examples

Texture Lookups

- Since the texture is an image, we need a way to index into it given a surface position
  - Or, where on the surface does the image go?
- Given a position, we lookup the **texture coordinates**, given as (u,v) values that refer to positions in the image
  - Easy to define for some shapes, can be very hard for others

Computing Texture Coordinates

- Idea: We will model this problem using a **texture coordinate function**,

$$\phi: S \rightarrow T, \text{ for all } (x,y,z) \in S \text{ and } (u,v) \in T$$

- When shading a point (x,y,z), we compute ϕ(x,y,z) to get the appropriate pixel (u,v) in the texture.

- u and v normally values in [0,1] (and then are scaled to the size of the texture)

Computing Texture Coordinates - Example

```
function texture_lookup(tex, u, v) {
  let i = Math.round(u * tex.width() - 0.5);
  let j = Math.round(v * tex.height() - 0.5);
  return tex.get_pixel(i,j);
}

function shade_surface_point(surf, pt, tex) {
  let normal = surf.get_normal(pt);
  [u,v] = surf.get_texcoord(pt);
  let diffuse_color = texture_lookup(tex,u,v);
  //compute shading using diffuse_color and normal
  //return shading result
}
```

Three Spaces

- Just like we have mappings from world space to image, we use ϕ as another mapping

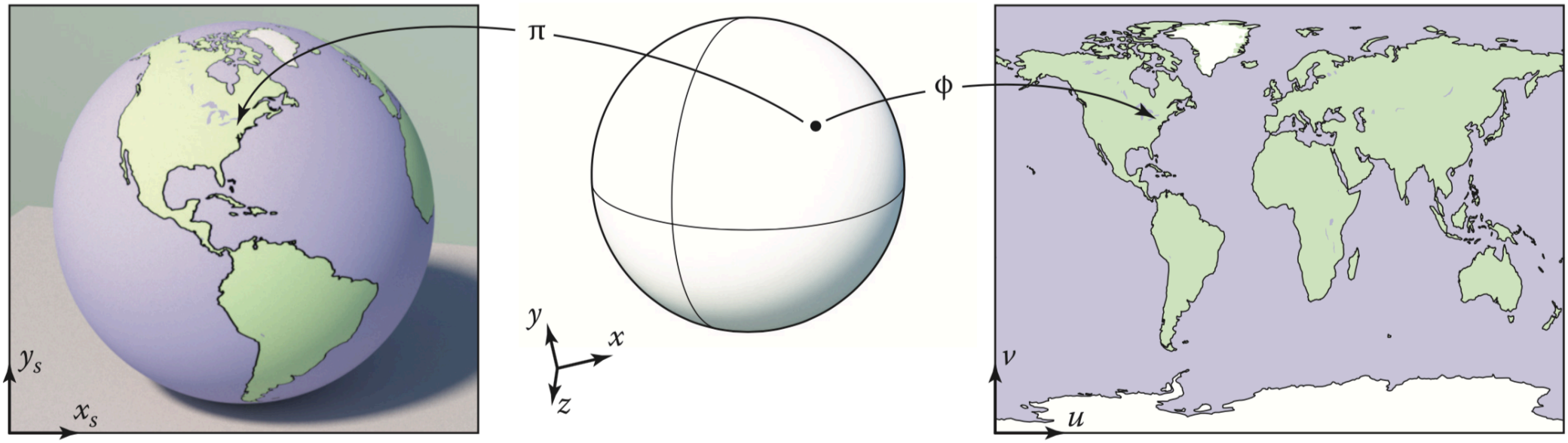


Image space | Surface S in world space | Texture space, T

Examples of Texture Coordinate Functions

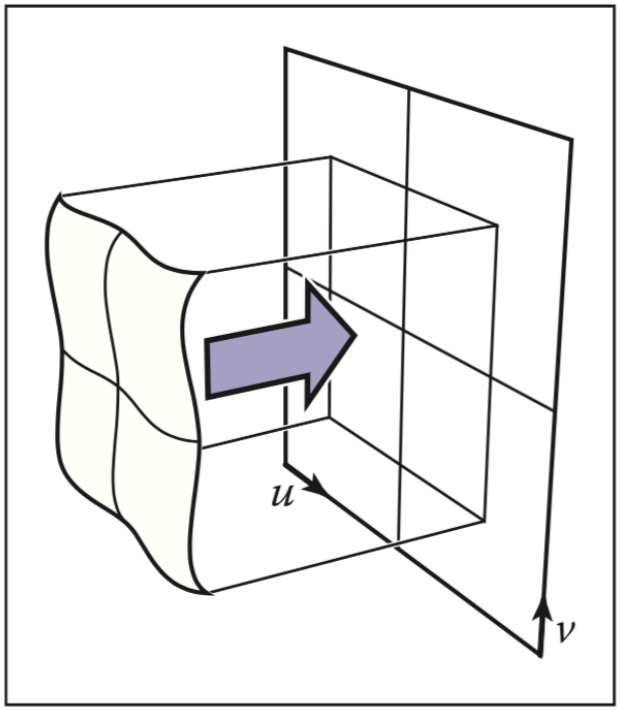
Problem #1: Defining Texture Coordinate Functions

- Defining ϕ can be very difficult for complex shapes
- Similar to the problem of taking a surface and flattening it
  - e.g. Cartographers problem
- Inevitably will have distortion of areas, angles, or distances

Planar Projection

- Flatten to a plane (e.g. dropping the z coordinate) or doing coordinate transform in the space of the the plane

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



Spherical Projection

- Convert (x,y,z) to spherical coordinates, discard radius

  $r = \text{sqrt}(x^2+y^2+z^2)$

  $u = (\pi + \text{atan2}(y,x)) / (2\pi)$

  $v = (\pi + \text{acos}(z/r)) / (\pi)$

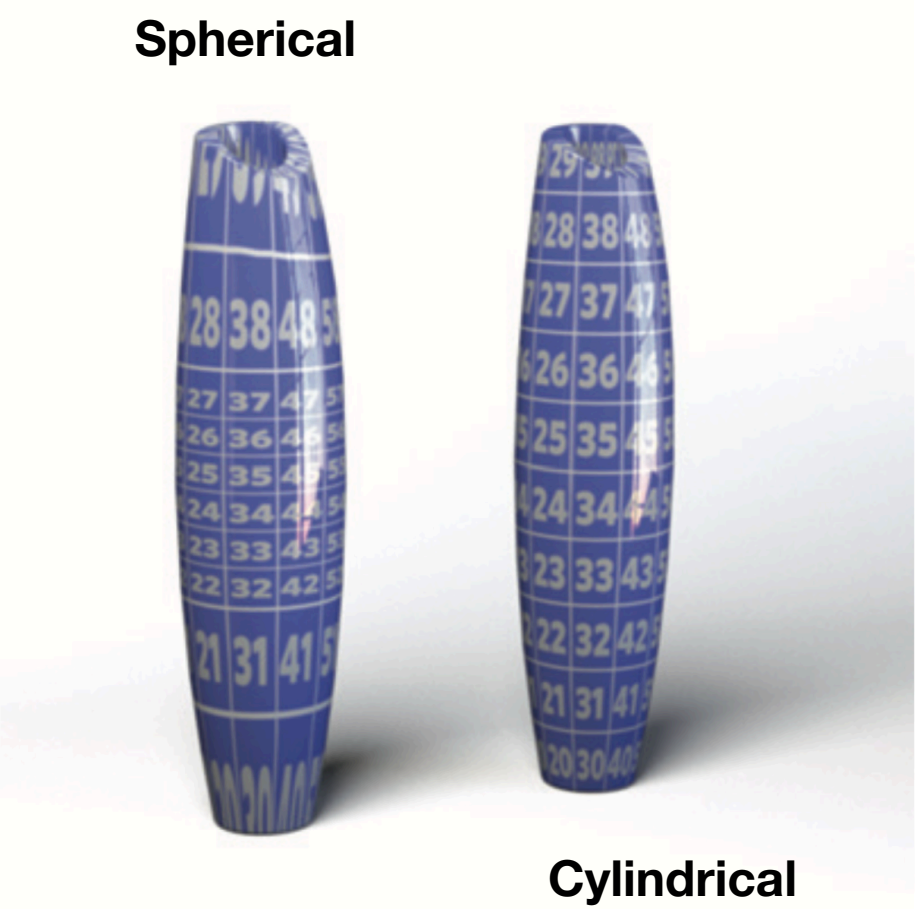
- Similar to casting a ray outward from center

Cylindrical Projection

- Convert (x,y,z) to cylindrical coordinates, discard radius

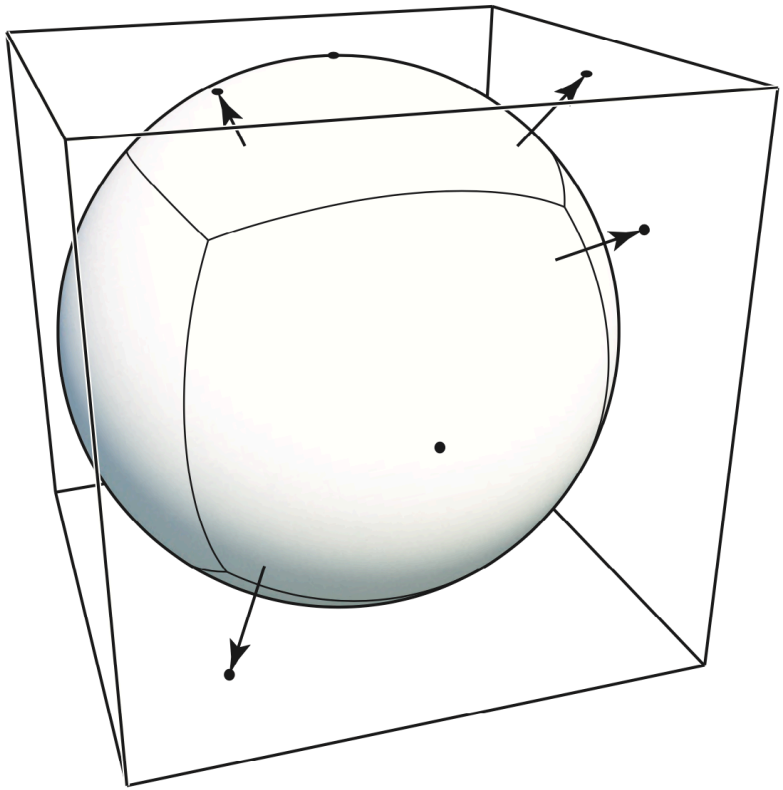
  u = (π + atan2(y,x)) / (2π)

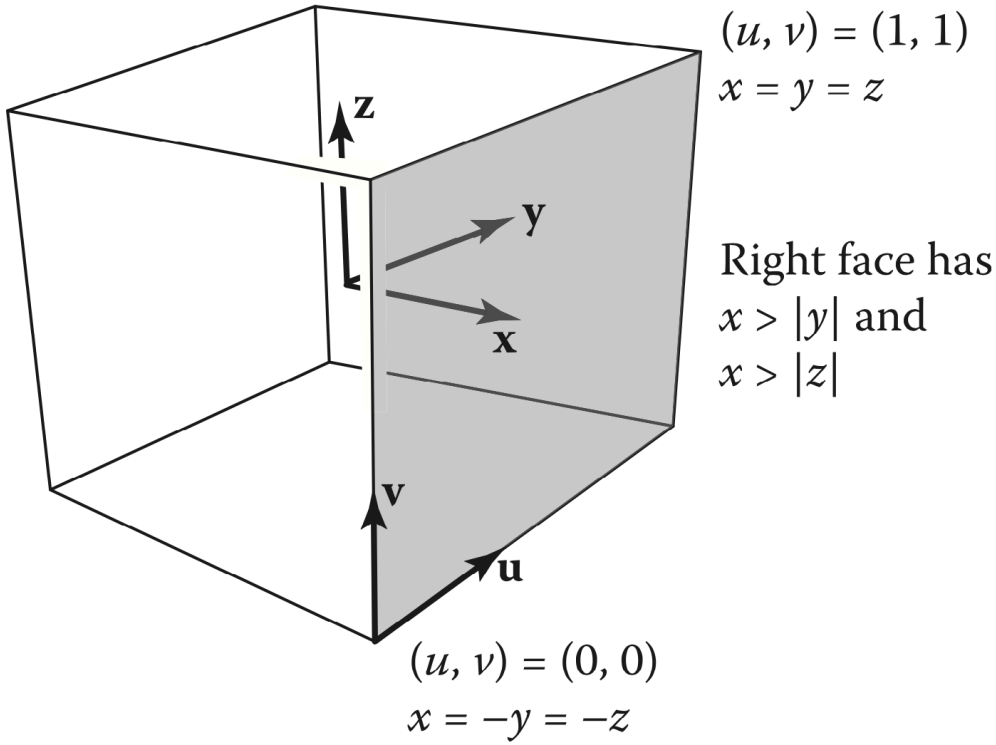
  v = 0.5 + z/2



Cubemaps

- Project onto faces of a cube, 6 planar projections
- Seams can be tricky to handle.

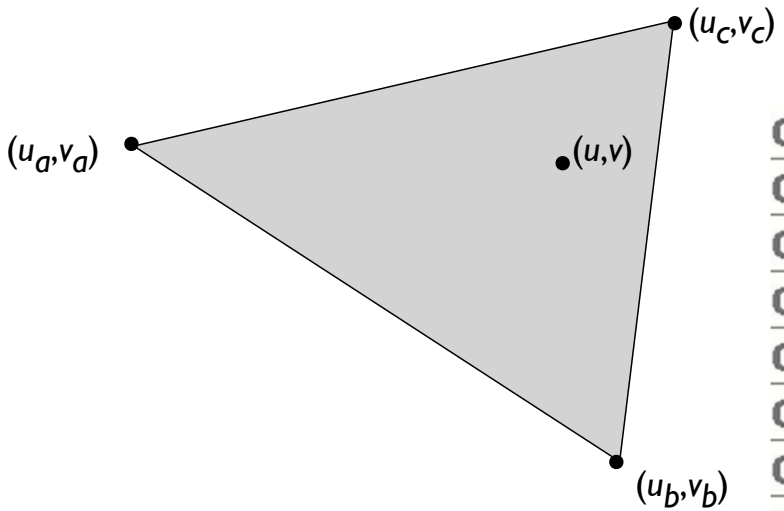


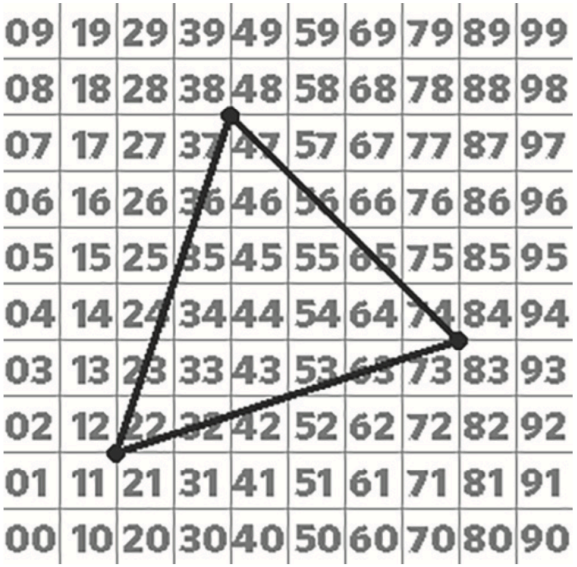


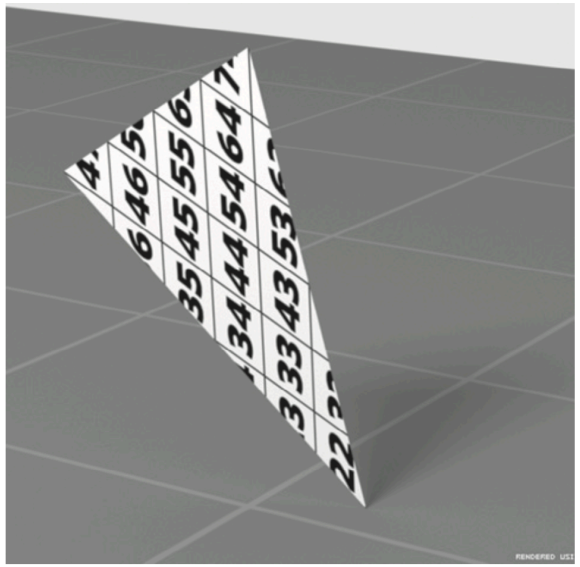
Interpolated Texture Coordinates

- Explicitly store (u,v) coordinates on the vertices of a triangle mesh, interpolate in the center using barycentric coordinates

- Texture coordinates just another per-vertex data. How to compute them? Can be difficult!







Properties of Texture Coordinate Functions

Goals for Texture Functions

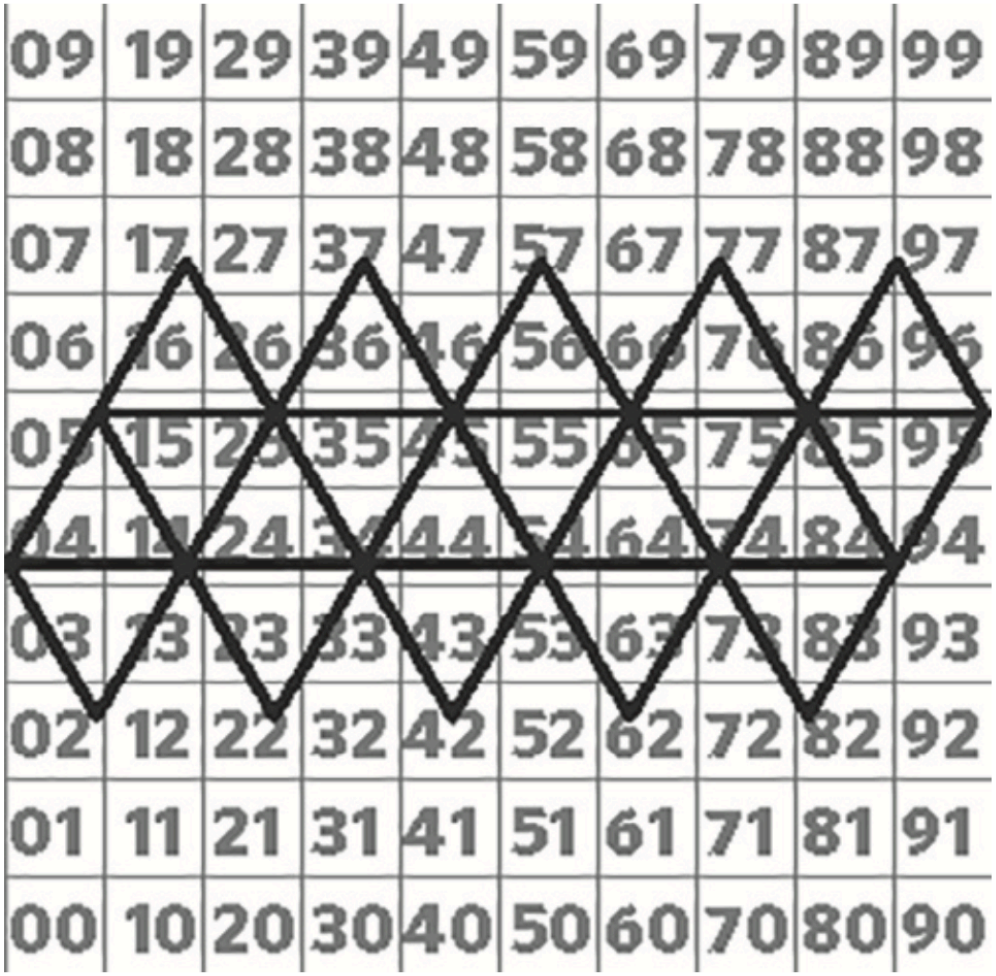
1. One-to-one vs one-to-many:
   - Each point on the surface should map to a different point on the texture, unless you want repetition

2. Size distortion:
   - Scale of texture kept constant across the surface

3. Shape distortion:
   - Shapes/Angles in the texture should state similarly shaped

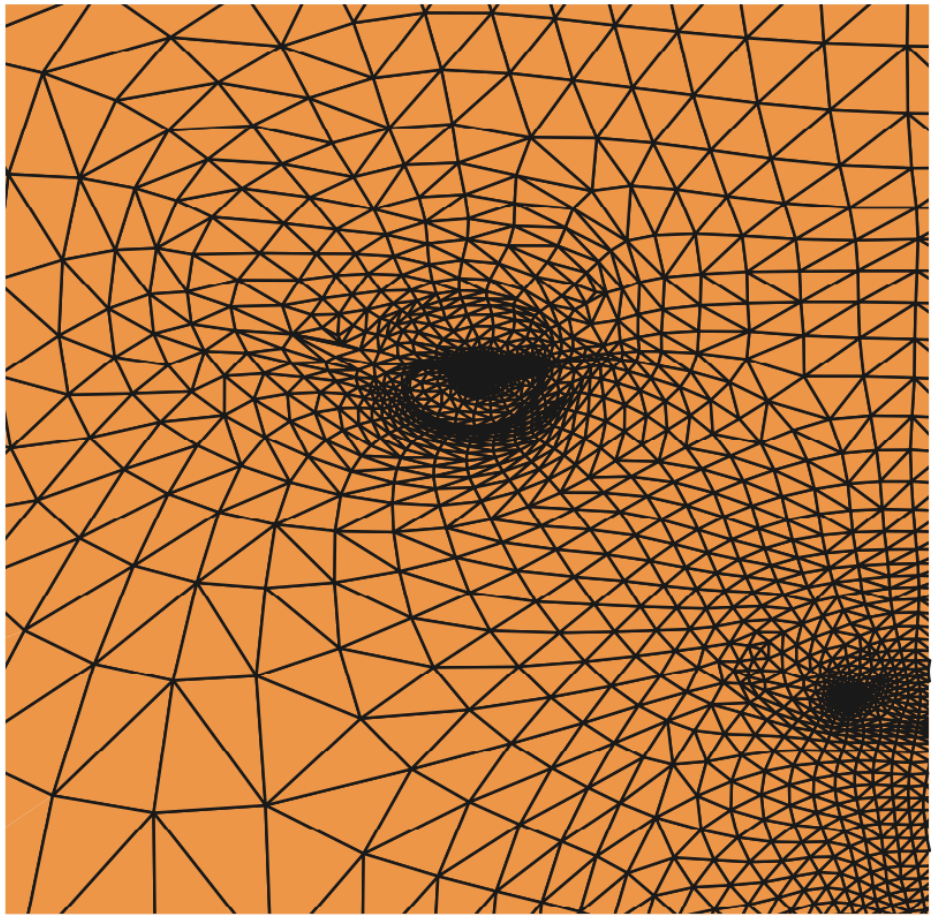
4. Continuity:
   - Are there visible seams? ϕ should have as few discontinuities as possible

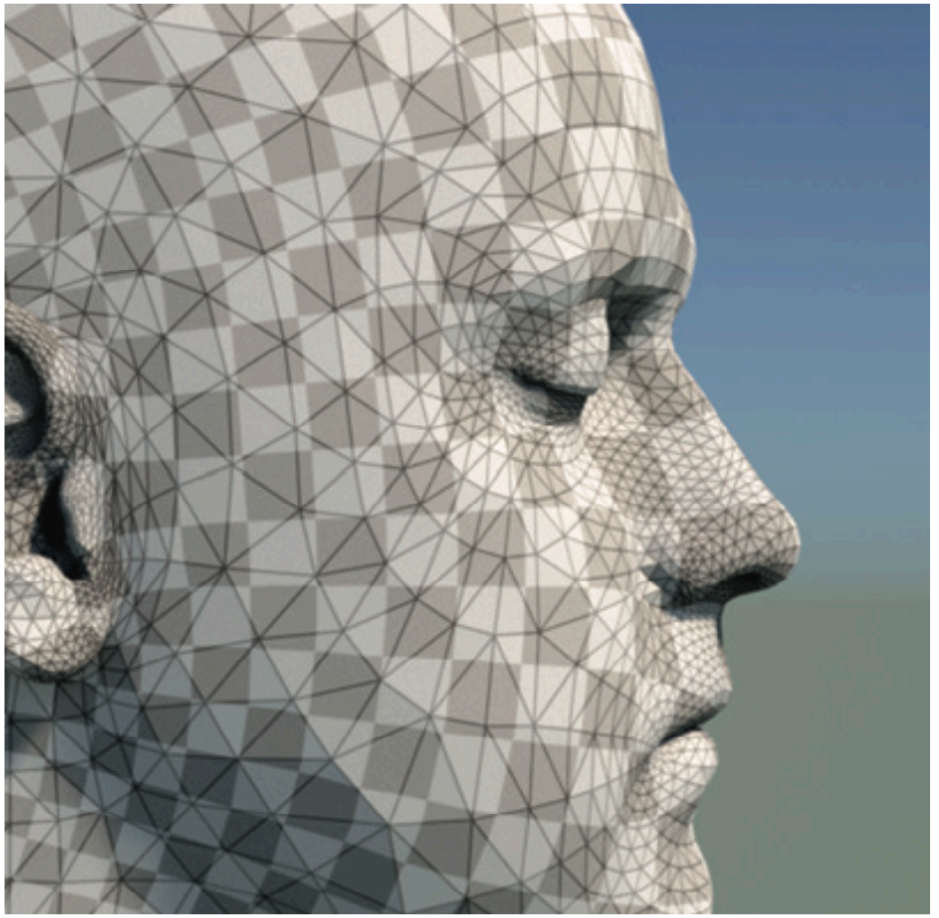
Distortions vs. Discontinuities



No distortion to area,
Many discontinuities

Shape vs. Area Distortions

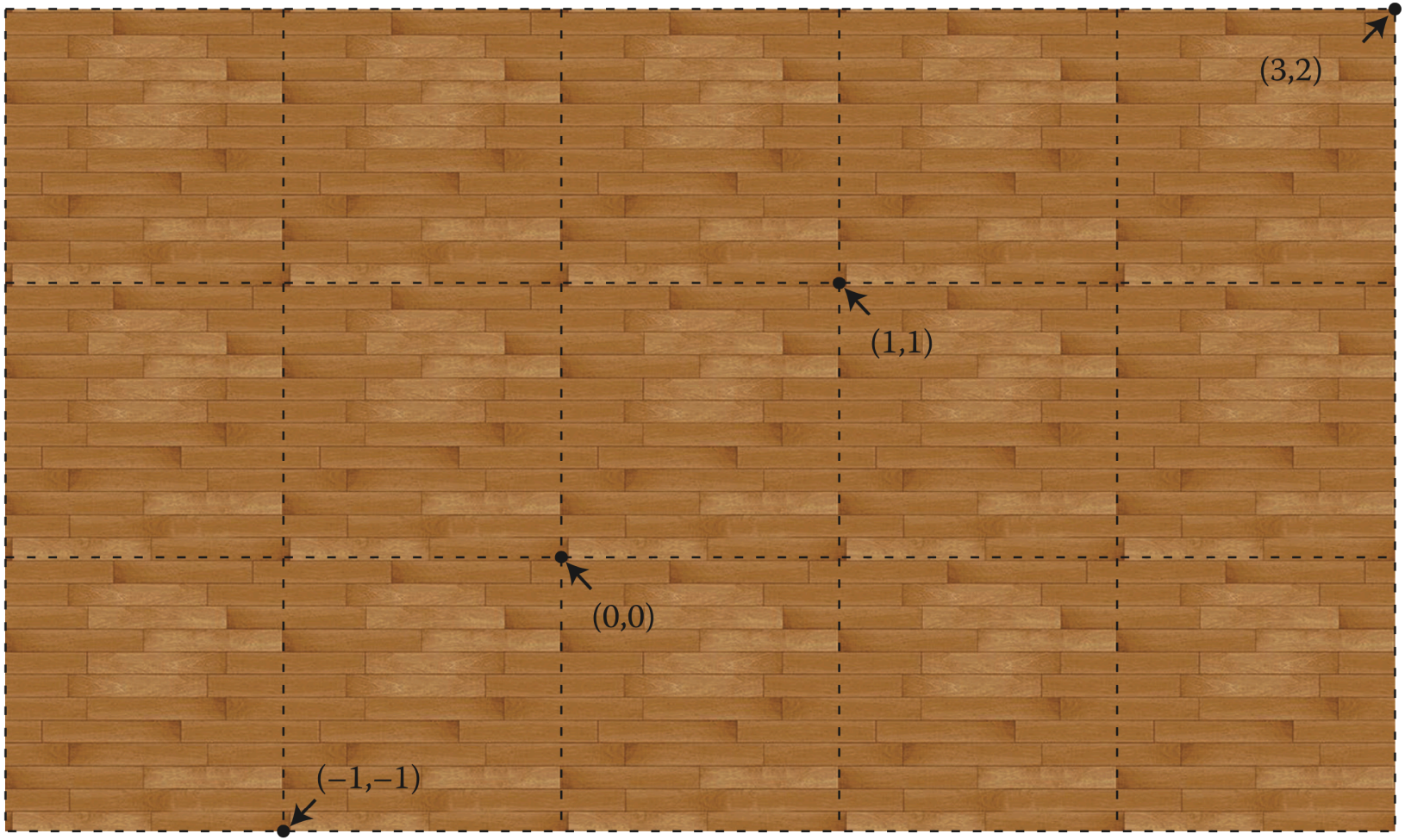




Low shape distortion,
Moderate area distortion

Tiling and Wrapping

- Can be achieved by modifying the mapping to cycle around in various ways (similar to boundary conditions for image processing)
- Could also just clamp values



Applications of Textures

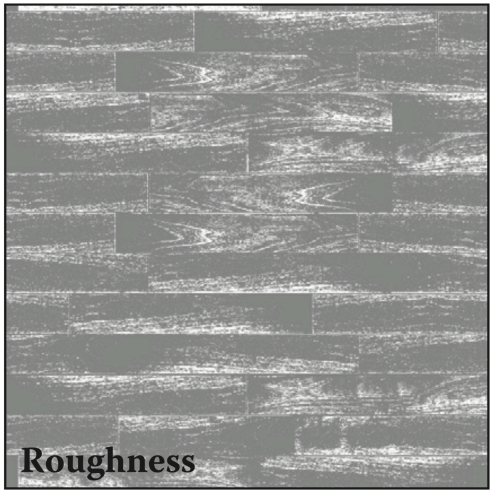
Controlling Shading Parameters

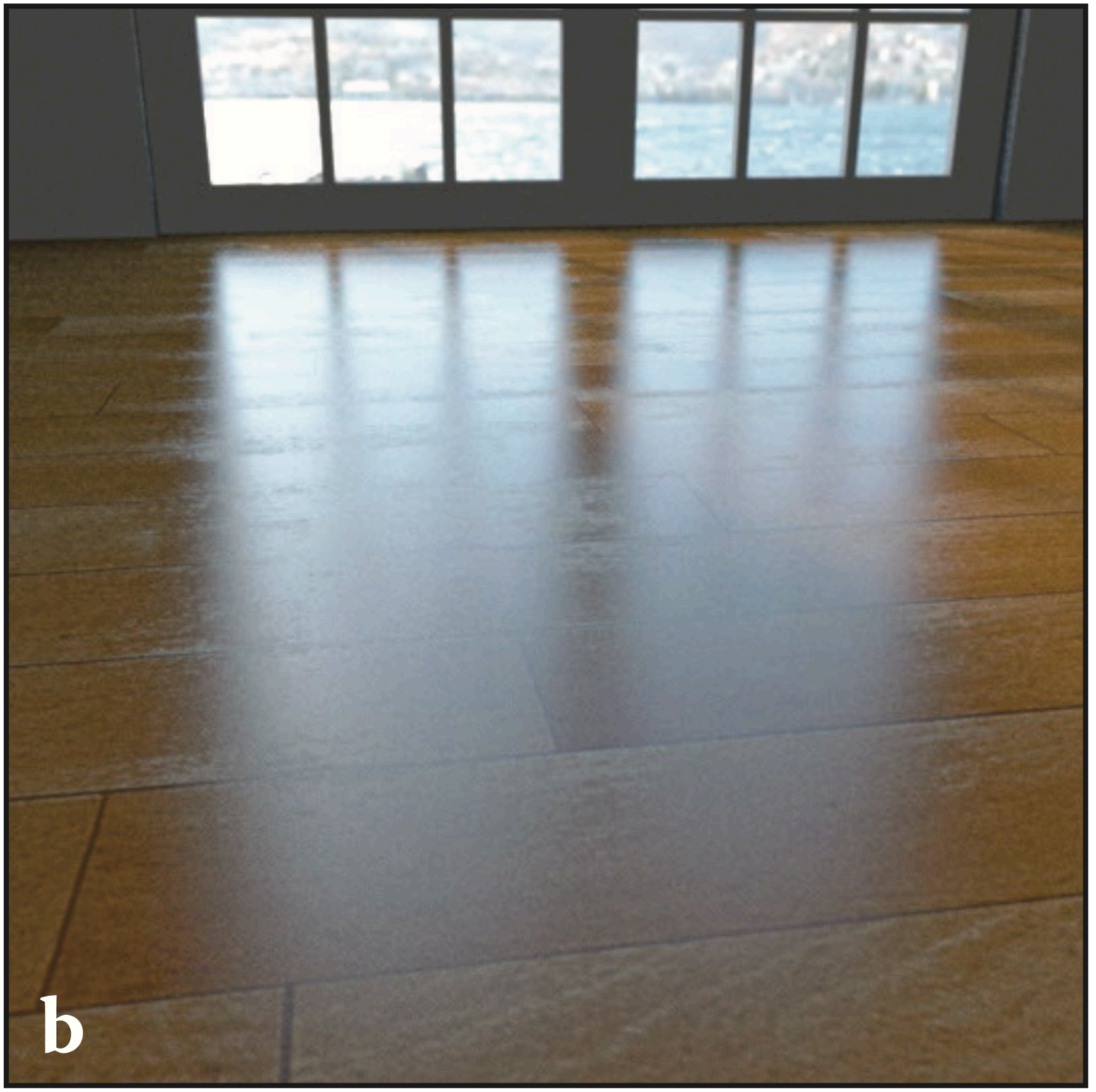
- Can look up k_d and k_s, or both


Color


a


Color


Roughness

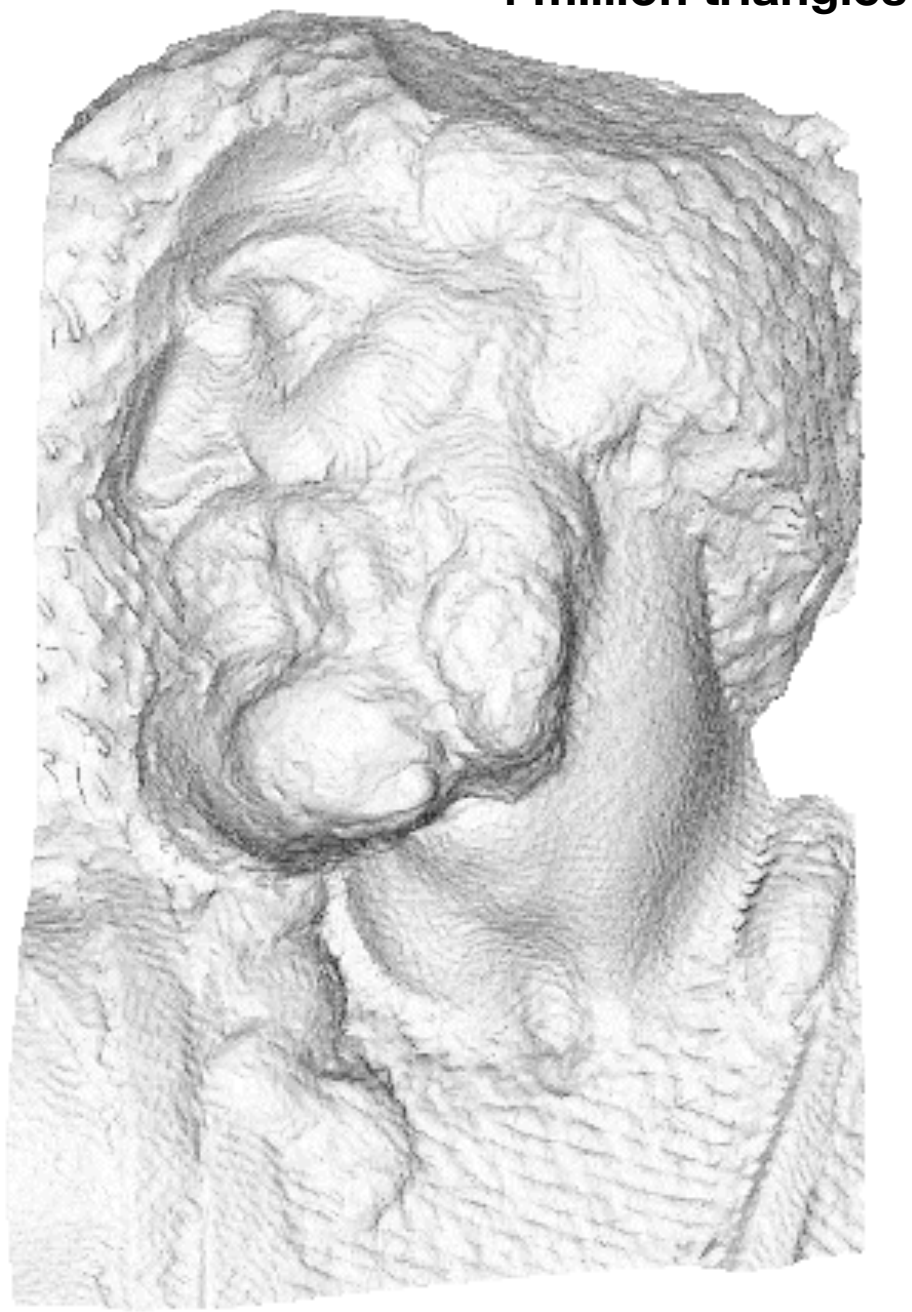

b

Normal Maps

- Can also use textures to look up normal information for the surface
- Typically, store the normal vector (n_x,n_y,n_z) as (r,g,b) values for the pixel
- Problem: orientation of the surface could change — normals are usually defined relative to a local coordinate space

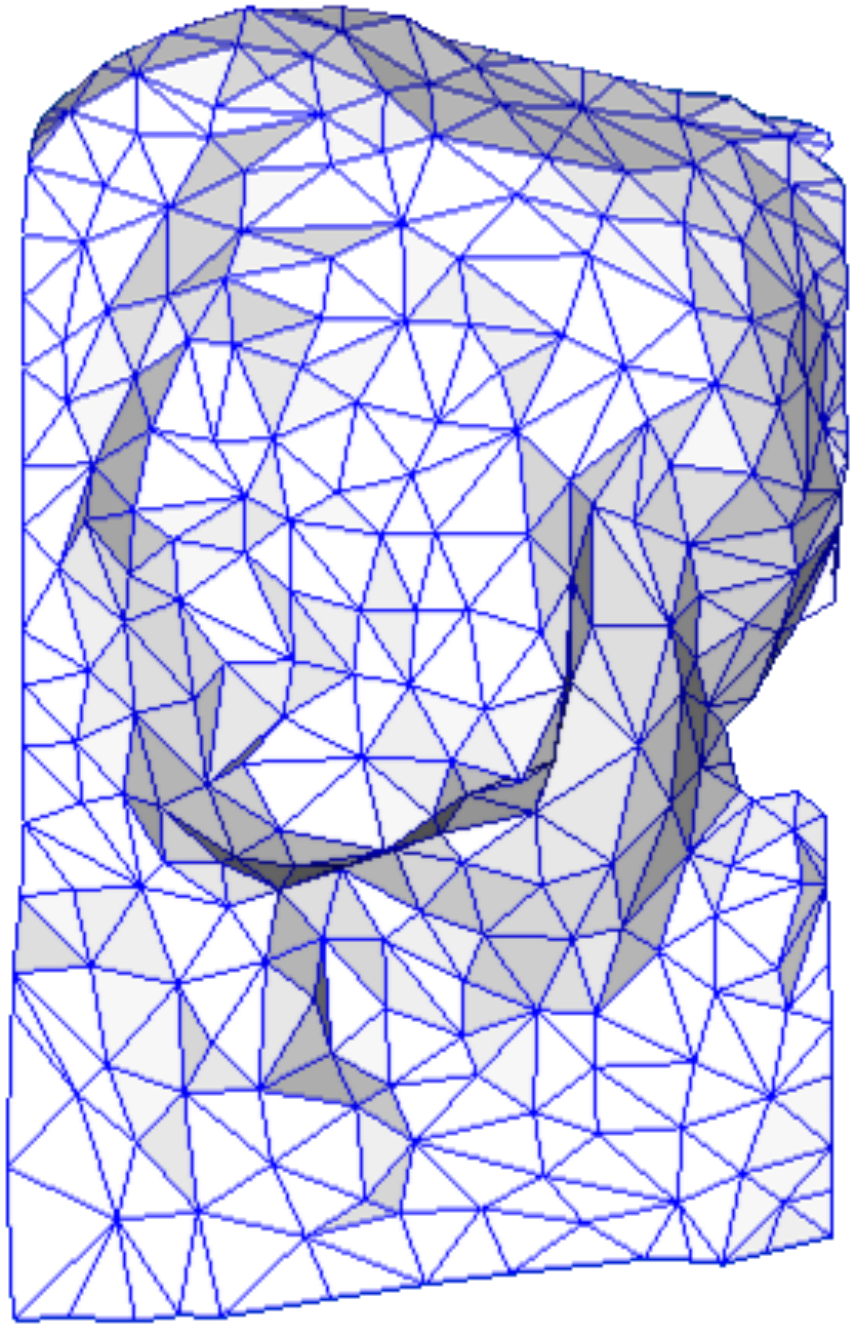
Normal Map Example

- Transfer details from high resolution mesh to normal map image

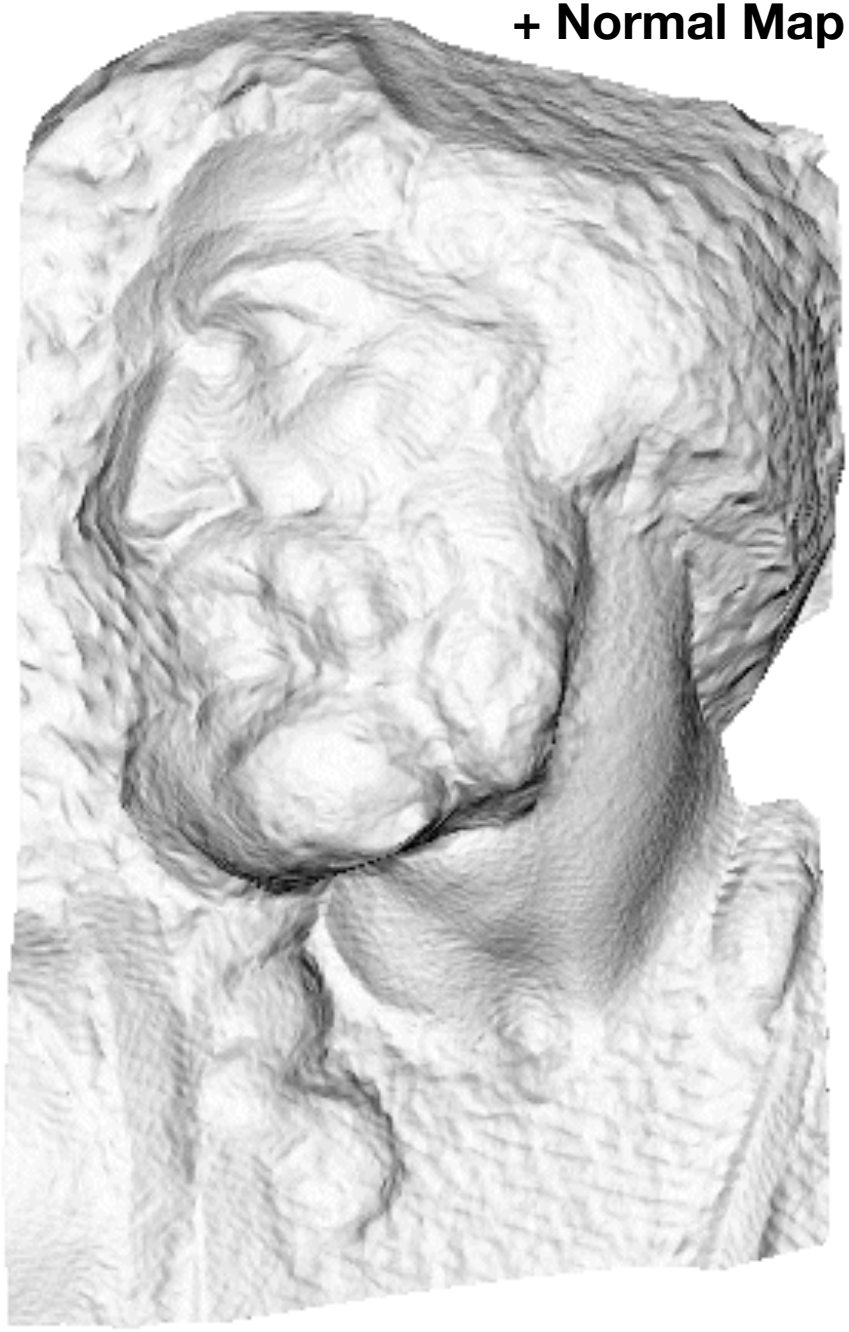


https://en.wikipedia.org/wiki/Normal_mapping

500 triangles



500 triangles
+ Normal Map

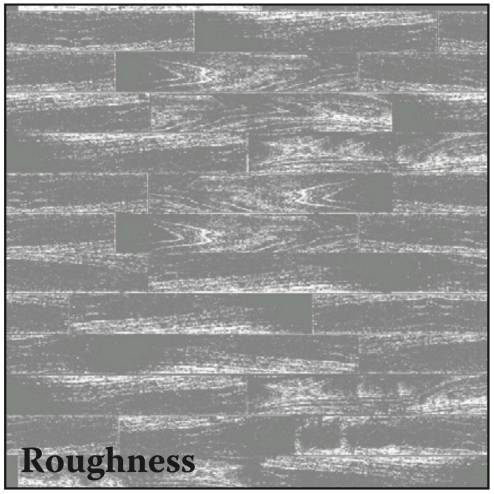


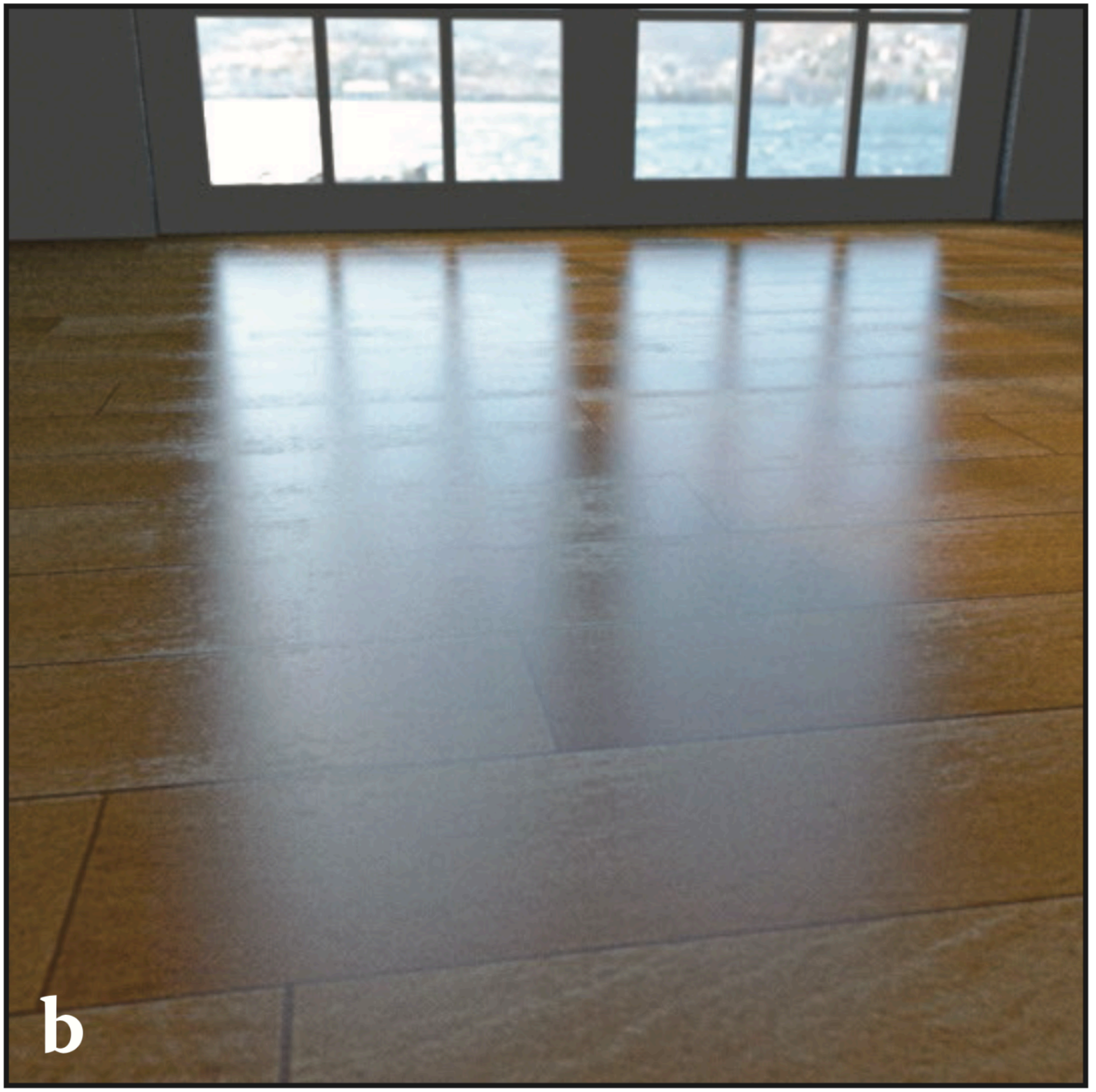
https://en.wikipedia.org/wiki/Normal_mapping

Bump Maps

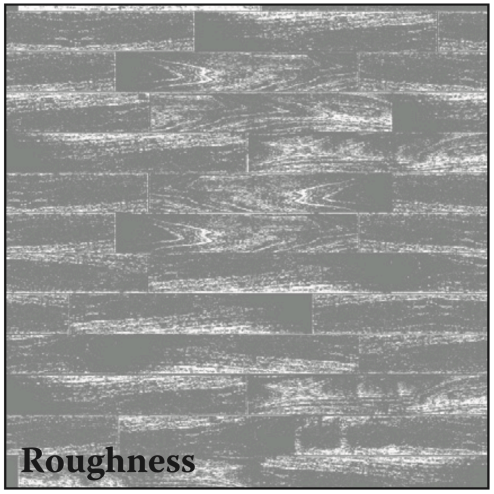
- Normals specified indirectly using a height field
  - The bump map encodes a local offset of the detailed surface above the original smooth surfaces
- Normal map is derived by taking the derivative of the bump map

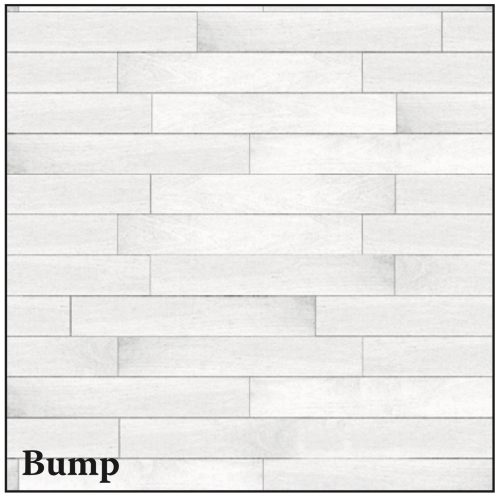

Color


Roughness


b


Color


Roughness


Bump


C

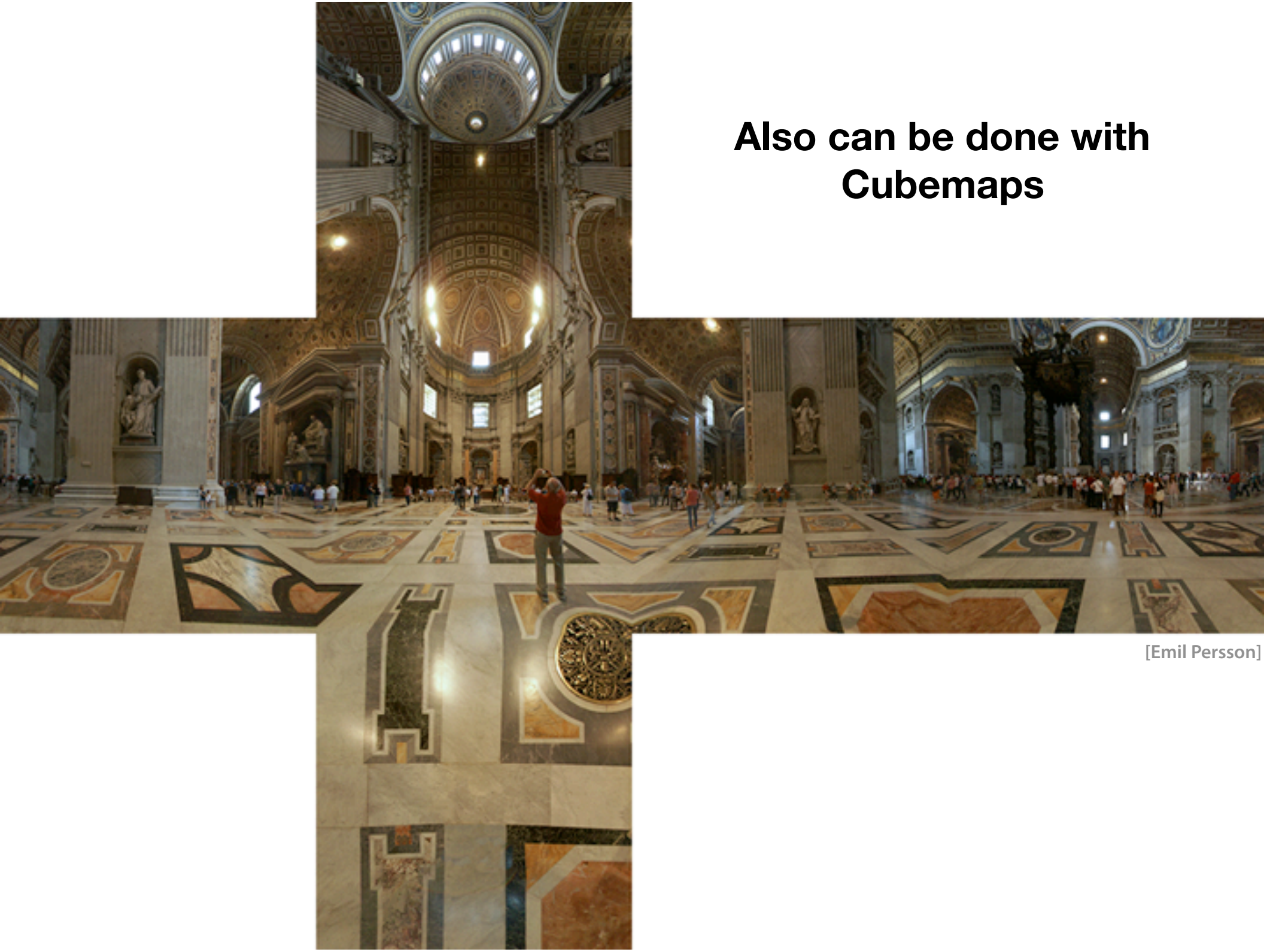
Environment Maps

- Use a texture to lookup values for rays that don't hit any objects

```
function trace_ray(ray, scene) {
  if (surface = scene.intersect(ray)) {
    return surface.shade(ray);
  } else {
    u,v = spheremap_coords(ray.direction);
    return texture_lookup(scene.env_map, u, v);
}
```

[Paul Debevec]

Also can be done with Cubemaps



[Emil Persson]

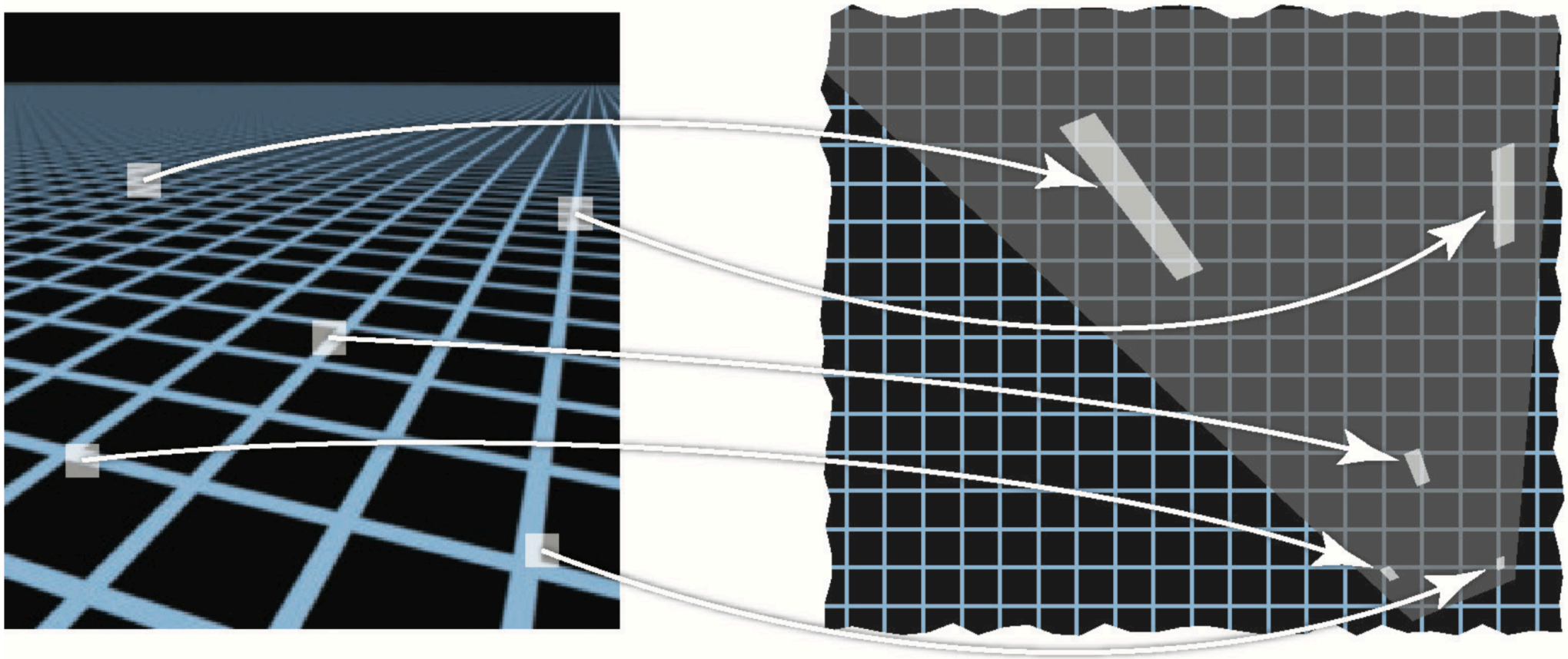
Antialiasing and Mipmaps

Problem: Sampling Textures Can Lead to Aliasing

- Just as we've seen with image processing and raytracing applications, if details are not captured with sufficient samples we can see noticeable artifacts
- Solution: use a better sampling/reconstruction

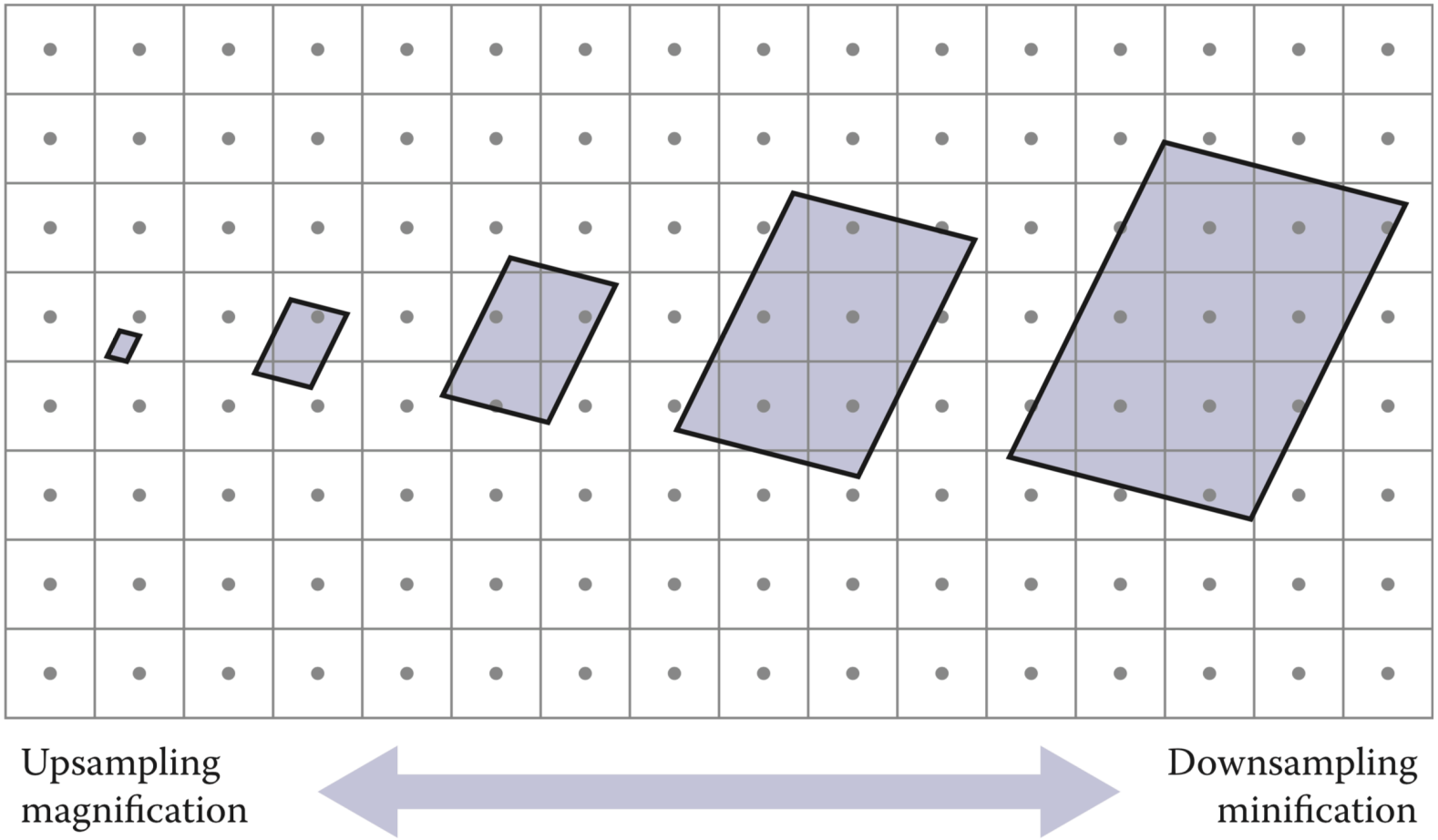
Pixel Footprints

- Can vary in size, shape, and orientation relative to the texture



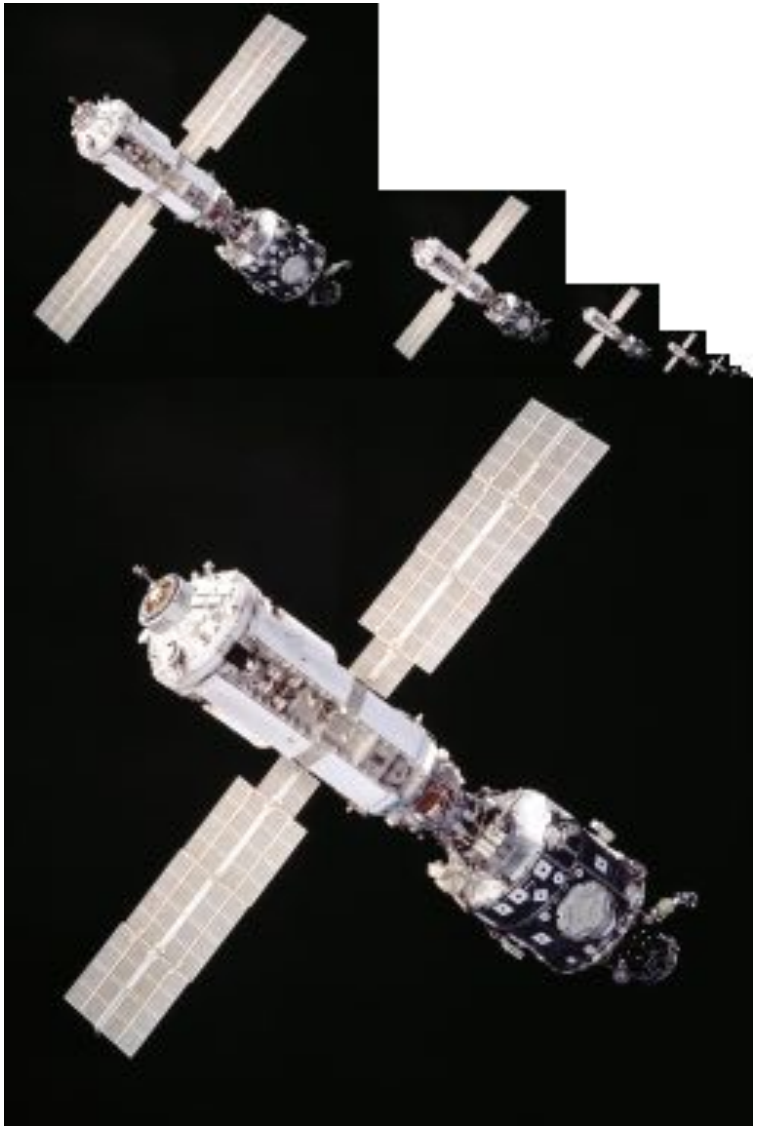
Sampling and Reconstruction

- If footprint is small, need better reconstruction (e.g. bilinear instead of nearest neighbor)
- If the footprint is large, need to average many samples



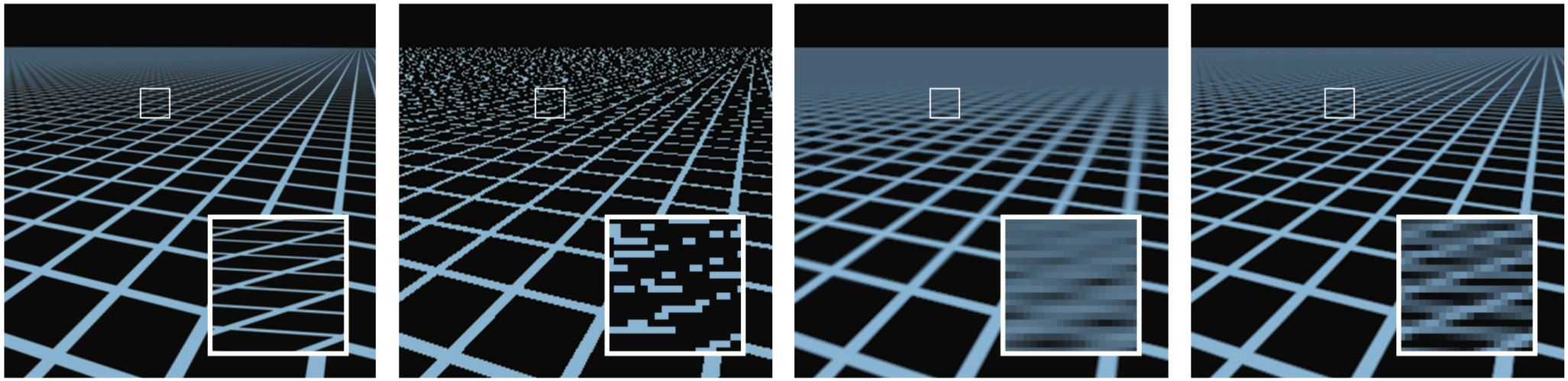
Mipmaps

- To quickly compute averages, store the texture at multiple resolutions
- For each lookup, estimate the size of the footprint and index into the mipmap accordingly



https://en.wikipedia.org/wiki/Mipmap

Correcting Aliasing



High-resolution image | Point sampling | Trilinear mipmap | Anisotropic filtering

Lec23 Required Reading

- FOCG, Ch. 16.1-16.2, 16.4
- Be sure to see recommended reading too!

Reminder:
Assignment 06

Assigned: Wednesday, Nov. 13
Written Due: Monday, Nov. 25, 4:59:59 pm
Program Due: Wednesday, Nov. 27, 4:59:59 pm