Textures and normals in ray tracing

CS 4620 Lecture 6

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

[Pi

Texture mapping

- Surface properties are not the same everywhere
  - diffuse color (k_d) varies due to changing pigmentation
  - brightness (k_s) and sharpenss (p) of specular highlight varies due to changing roughness 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 k_S, n are constant
- Glazed pot with finger prints
  - diffuse and specular colors k_D, k_S are constant
  - specular exponent n varies 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

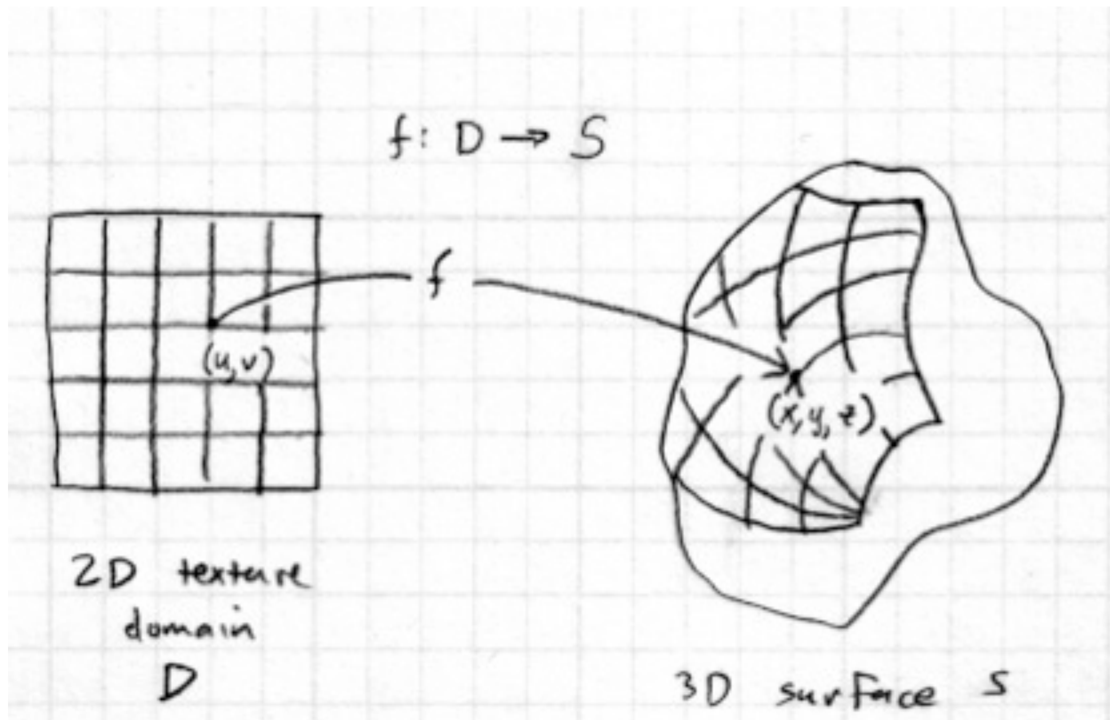

RENDERED USING MITSUBA

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

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

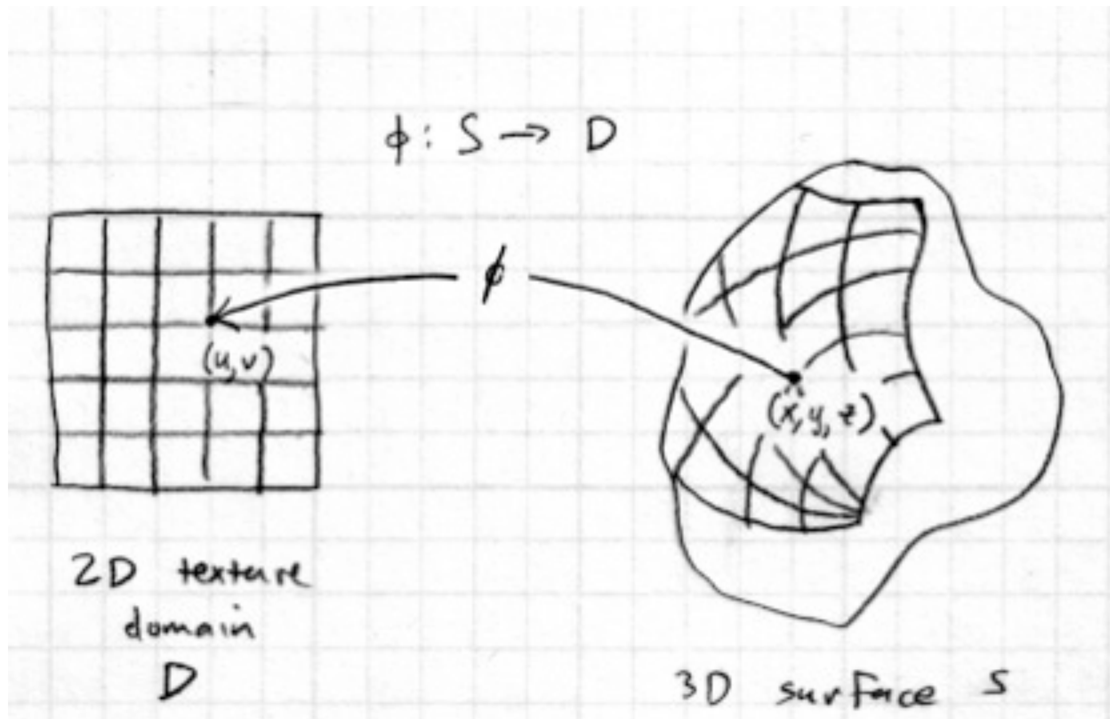


Texture coordinate functions

- 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.

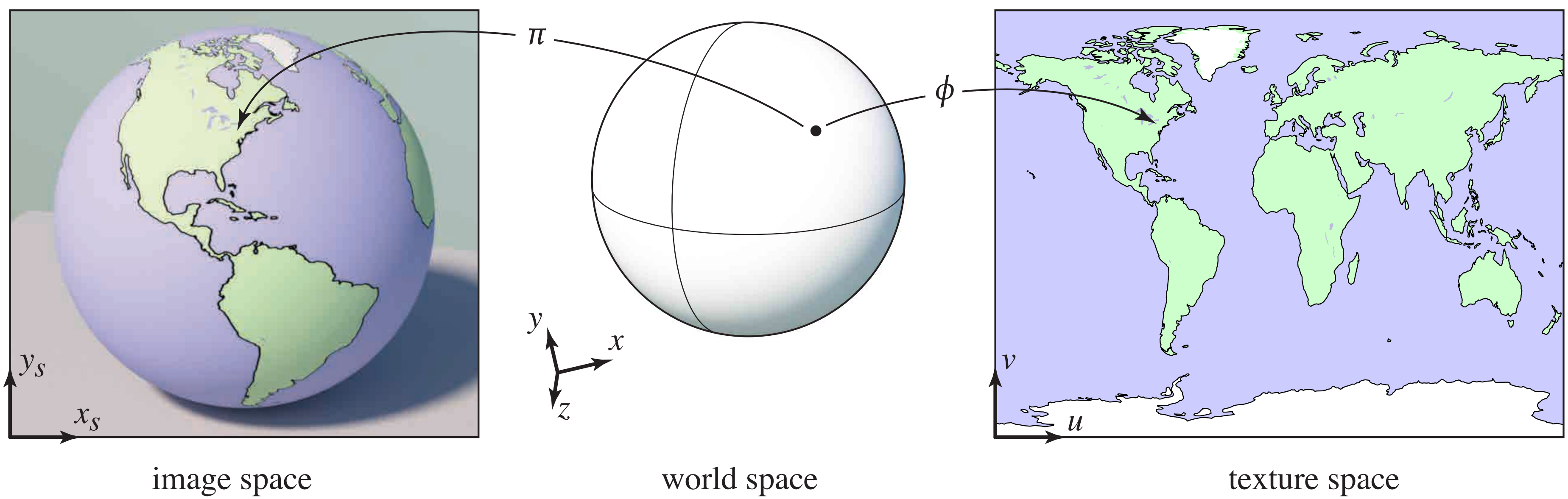
  $$\phi : S \to \mathbb{R}^2$$

  - when shading **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.



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 **p** is then

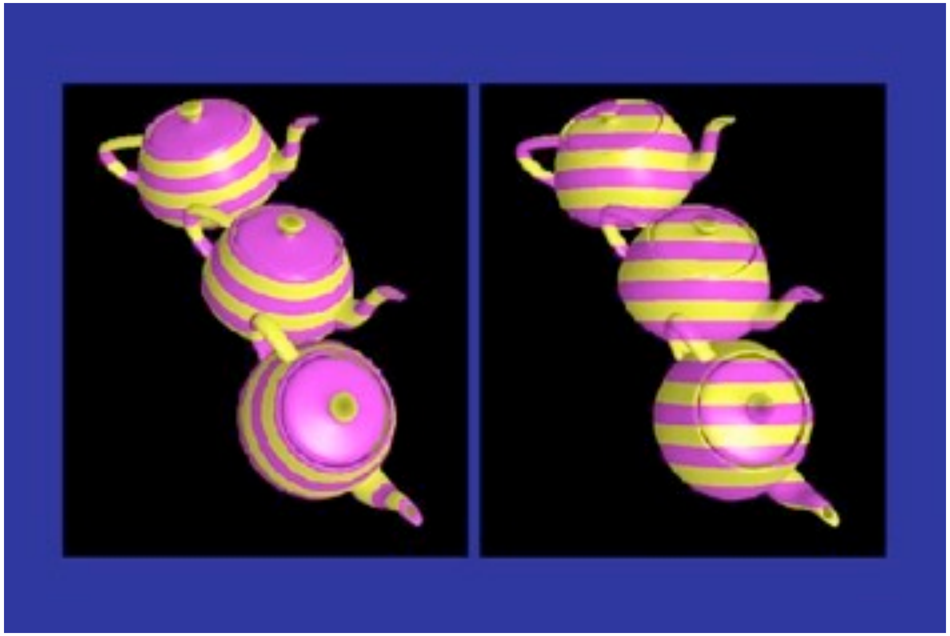
$$k_D(\mathbf{p}) = T(\phi(\mathbf{p}))$$

Coordinate functions: parametric

- For parametric surfaces you already have coordinates
- Need to be able to invert the parameterization
- E.g. for a rectangle…
- E.g. for a sphere…

Examples of coordinate functions

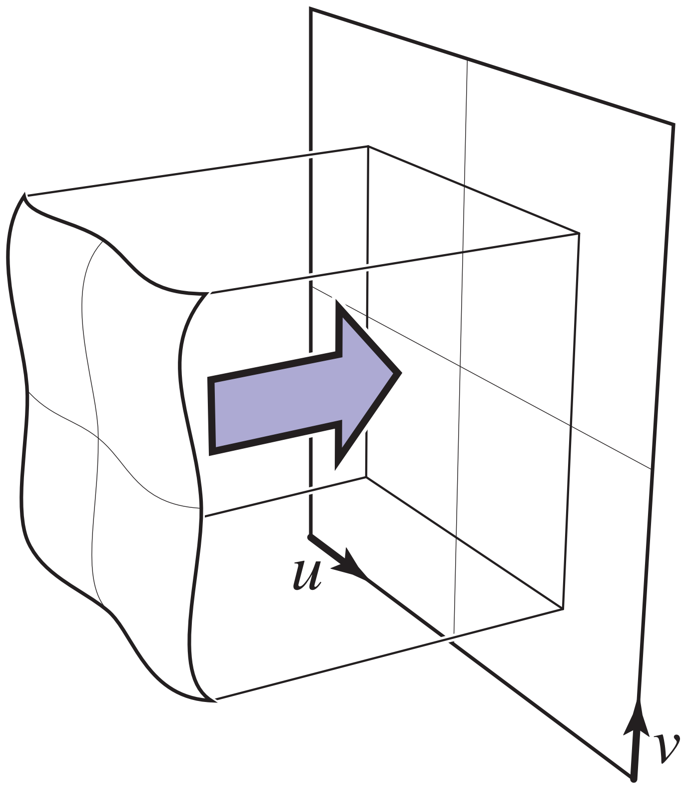
- For non-parametric surfaces it is trickier
  - directly use world coordinates
    - need to project one out



[Wolfe / SG97 Slide set]

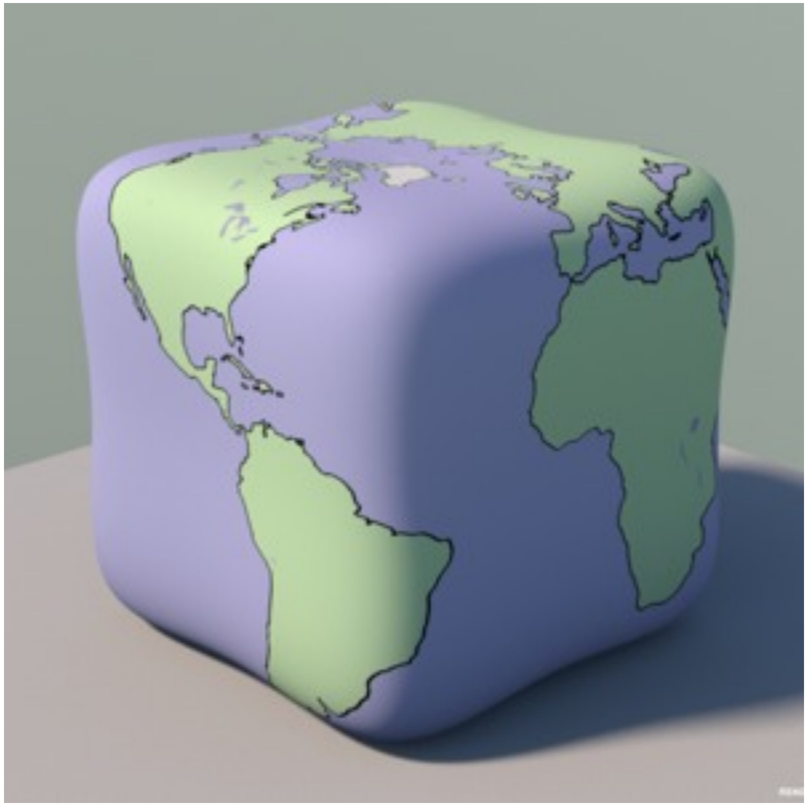
Examples of coordinate functions

- Planar projection



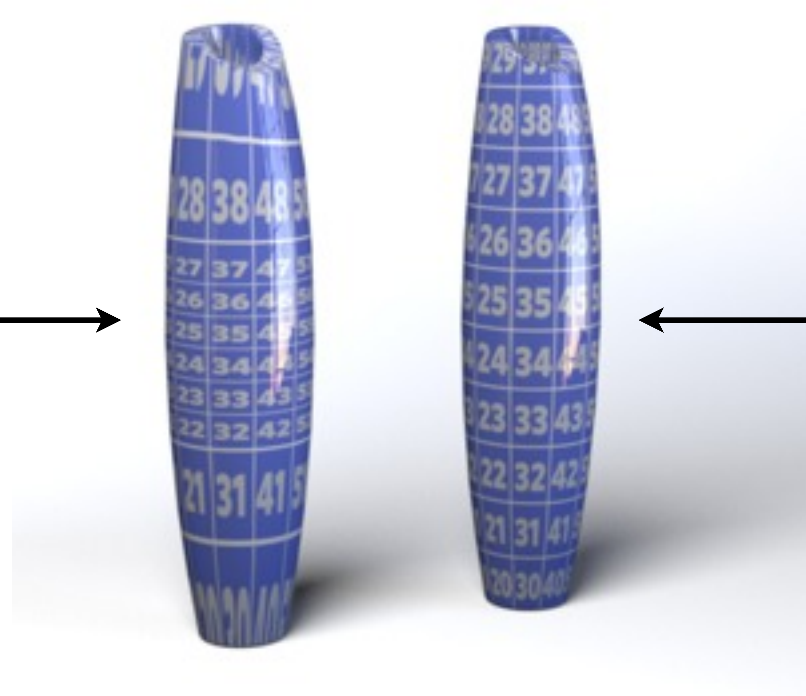
Examples of coordinate functions

- Spherical projection



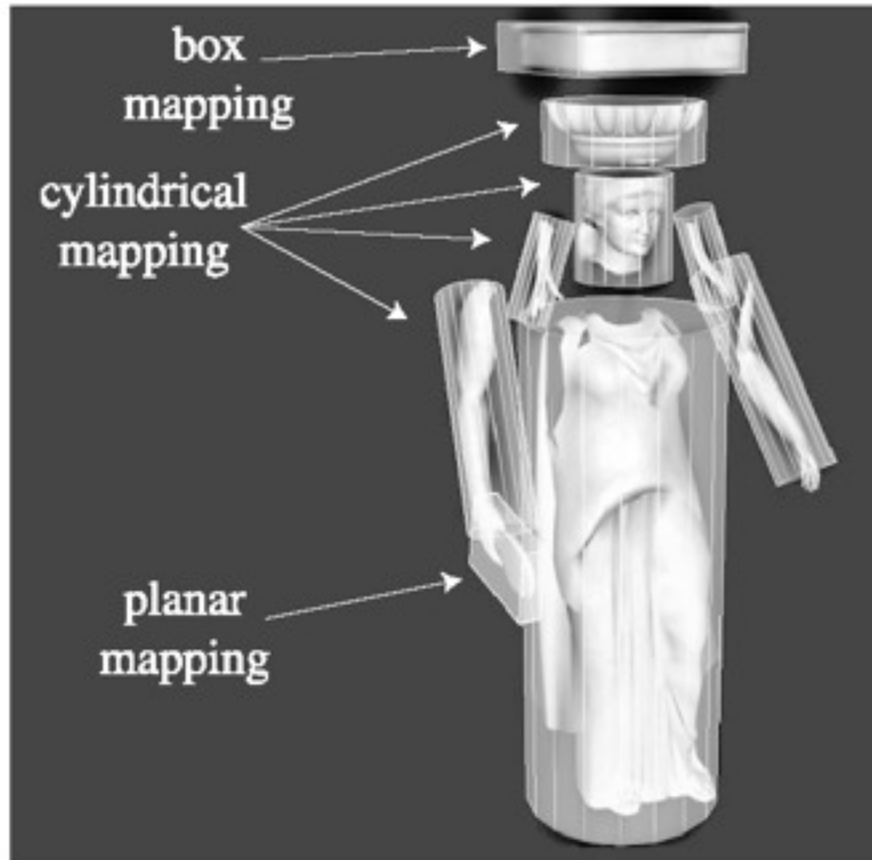
Examples of coordinate functions

- Cylindrical projection

spherical →  ← cylindrical

Examples of coordinate functions

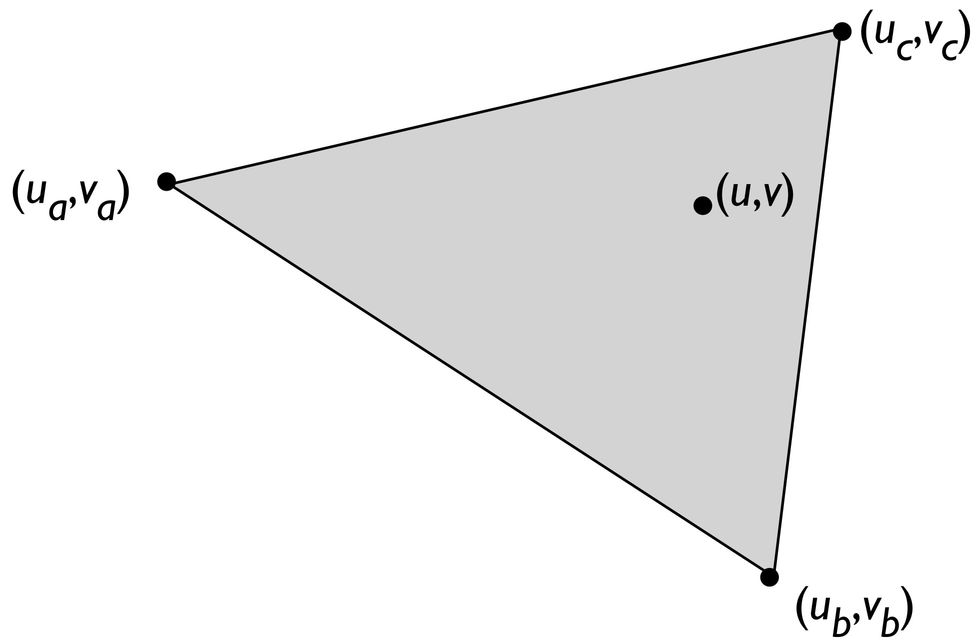
- Complex surfaces: project parts to parametric surfaces



[Tito Pagan]

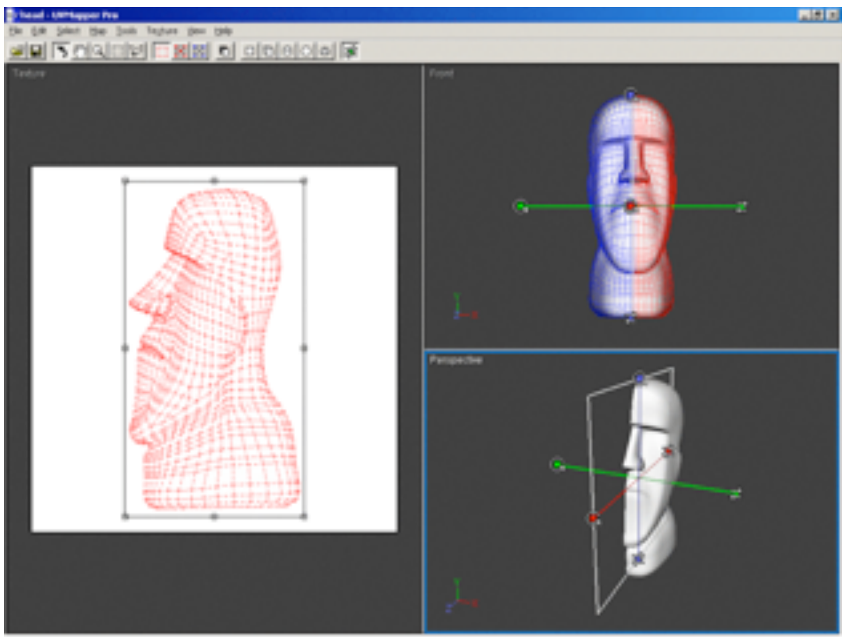
Examples of coordinate functions

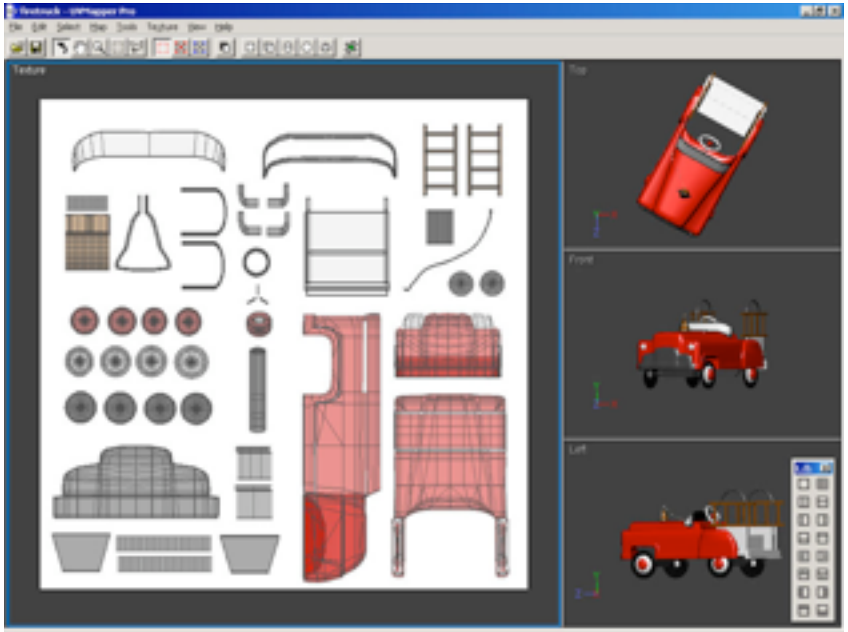
- Triangles
  - specify (u,v) for each vertex
  - define (u,v) for interior by linear (barycentric) interpolation

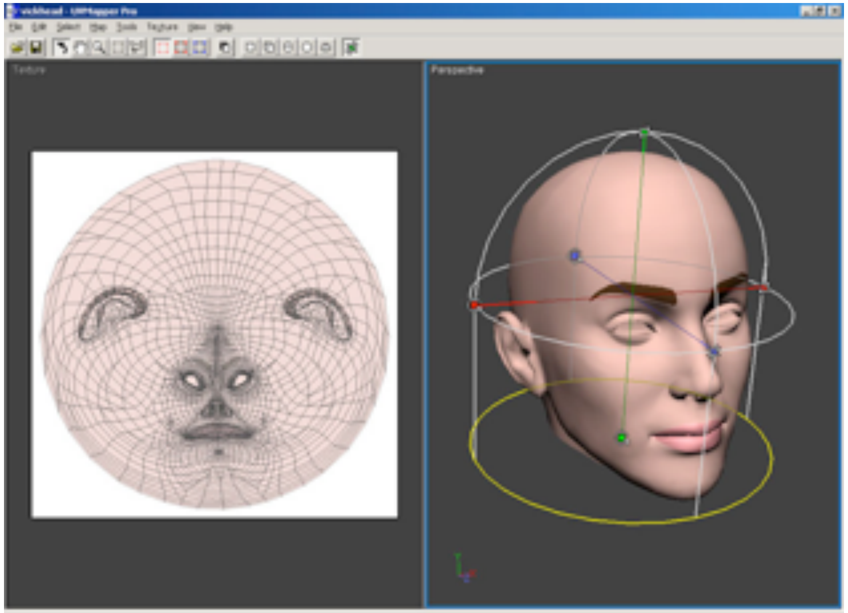


Example: UVMapper

http://www.uvmapper.com



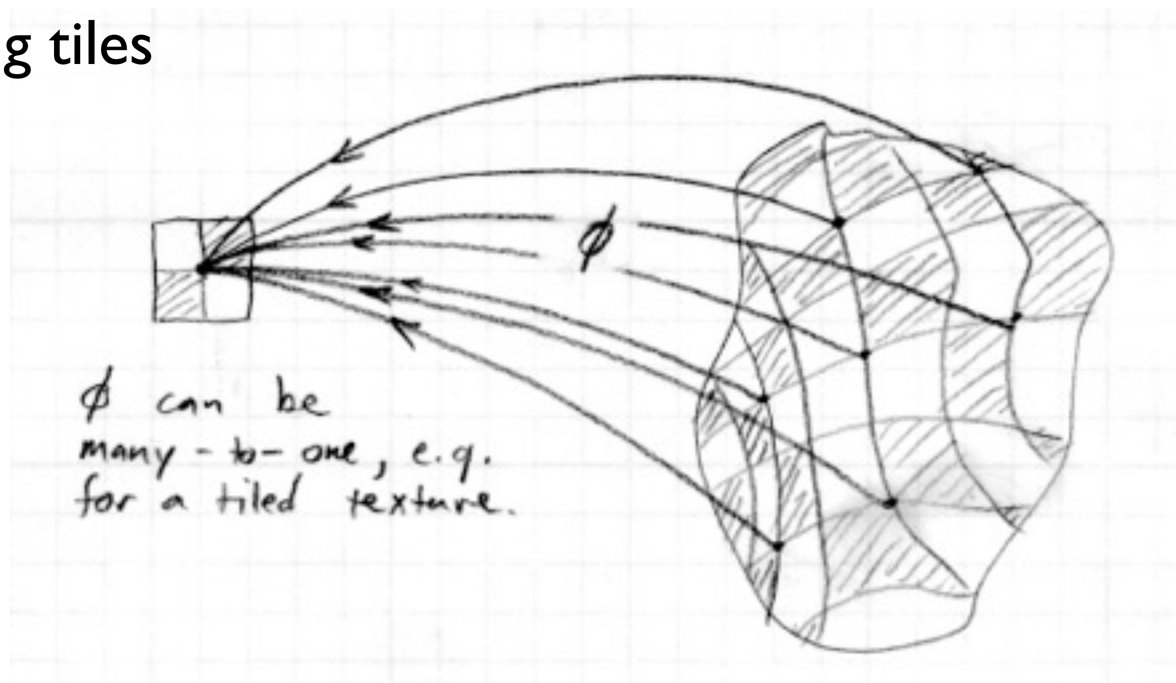




(with previous instructors James/Marschner, and some slides courtesy Leonard McMillan)

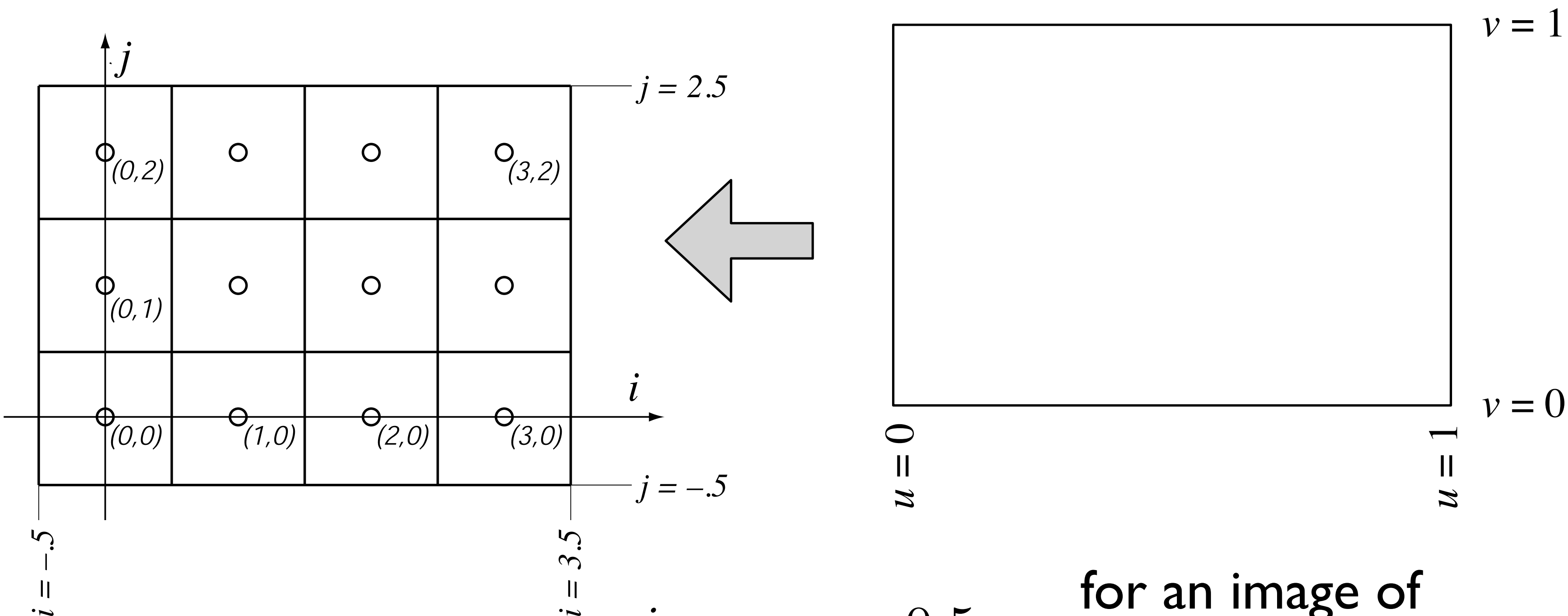
Texture coordinate functions

- 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)

- Related to texture coordinates in the same way as normalized image coordinate to pixel coordinates



$$i = u\,n_x - 0.5$$
$$j = v\,n_y - 0.5$$

for an image of n_x by n_y pixels

Texture lookups and wrapping

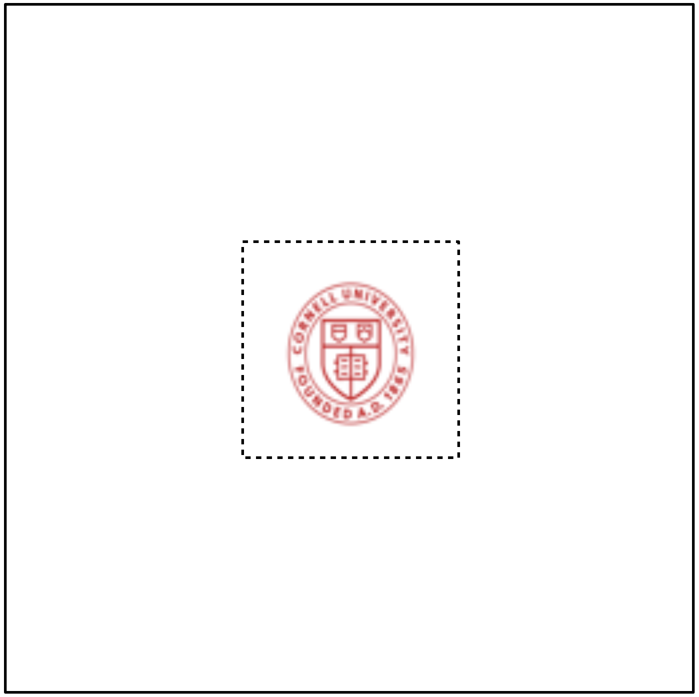
- In shading calculation, when you need a texture value you perform a *texture lookup*
- Convert (*u*, *v*) texture coordinates to (*i*, *j*) texel coordinates, and read a value from the image
  - simplest: round to nearest (nearest neighbor lookup)
  - various ways to be smarter and get smoother results
- What if *i* and *j* are out of range?
  - option 1, clamp: take the nearest pixel that is in the image

$$i_{\text{pixel}} = \max(0, \min(n_x - 1, i_{\text{lookup}}))$$

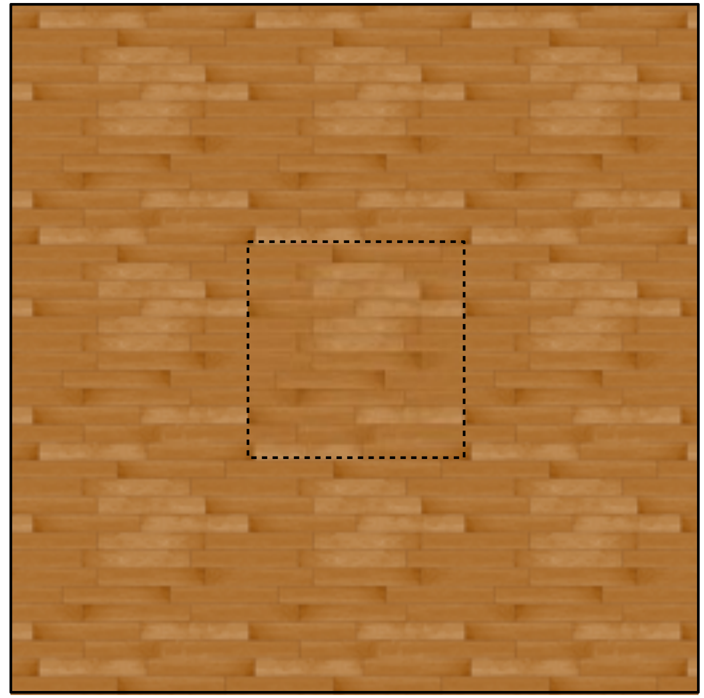
  - option 2, wrap: treat the texture as periodic, so that falling off the right side causes the look up to come in the left

$$i_{\text{pixel}} = \text{remainder}(i_{\text{lookup}}, n_x)$$

Wrapping modes



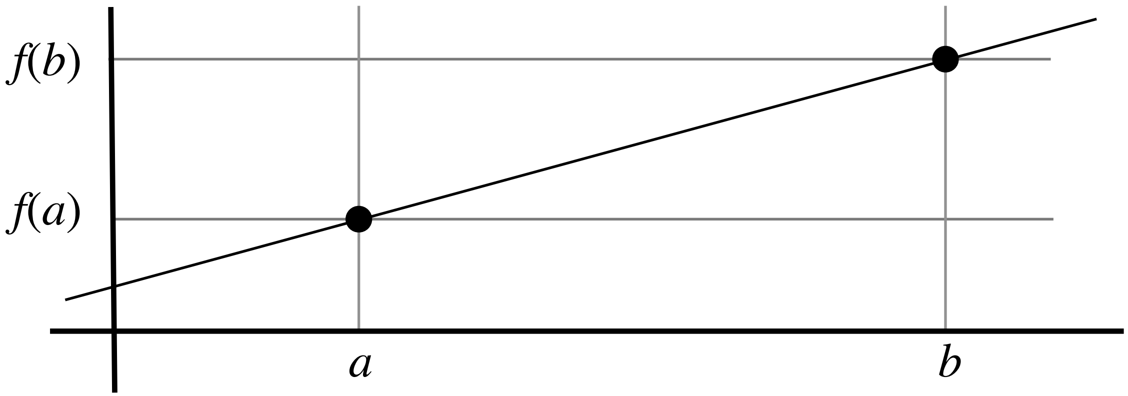
clamp



wrap

Linear interpolation, 1D domain

- Given values of a function f(x) for two values of x, you can define in-between values by drawing a line



See Shirley Sec. 2.6

- there is a unique line through the two points
- can write down using slopes, intercepts
- …or as a value added to f(a)
- …or as a convex combination of f(a) and f(b)

$$
\begin{aligned}
f(x) &= f(a) + \frac{x-a}{b-a}(f(b) - f(a)) \\
&= (1-\beta)f(a) + \beta f(b) \\
&= \alpha f(a) + \beta f(b)
\end{aligned}
$$

Linear interpolation in 1D

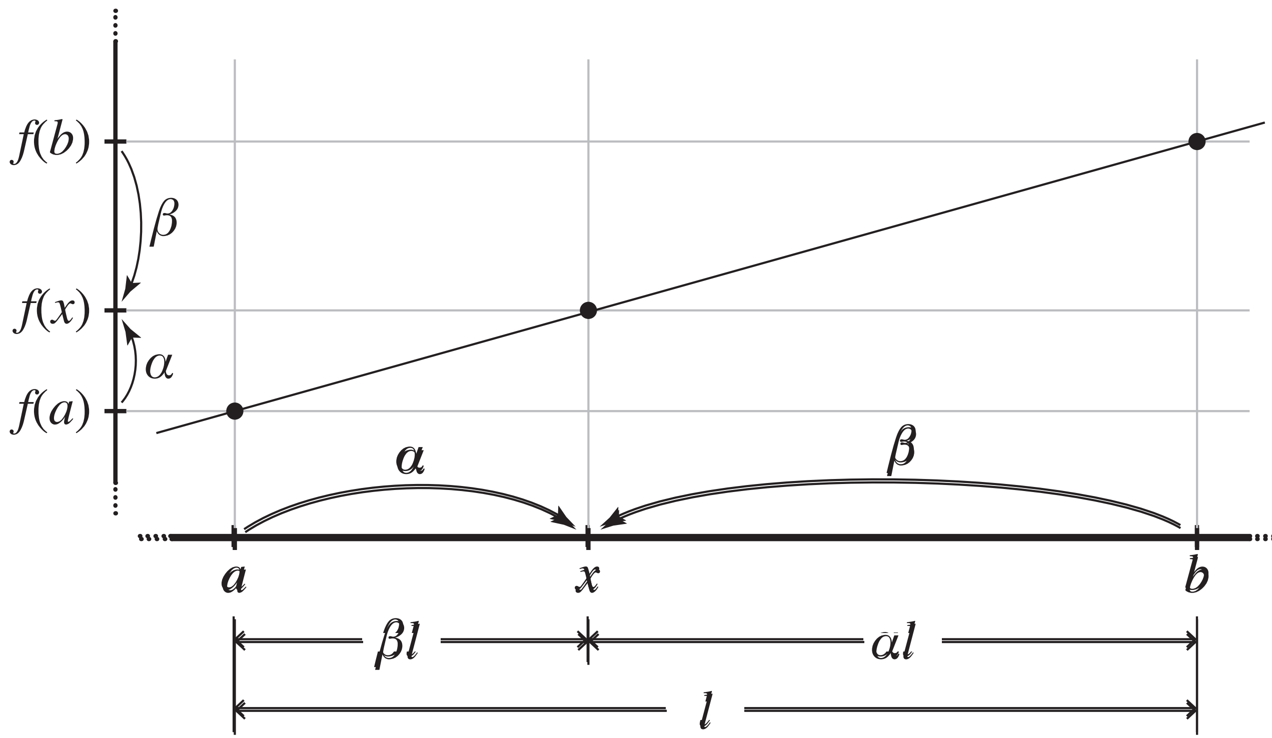
- Alternate story
  1. write *x* as convex combination of a and b

     $$x = \alpha a + \beta b \quad \text{where } \alpha + \beta = 1$$

  2. use the same weights to compute *f*(*x*) as a convex combination of *f*(*a*) and *f*(*b*)

     $$f(x) = \alpha f(a) + \beta f(b)$$

Linear interpolation in 1D



Linear interpolation in 2D

- Use the alternate story:
  1. Write **x**, the point where you want a value, as a convex linear combination of the vertices

     $$\mathbf{x} = \alpha\mathbf{a} + \beta\mathbf{b} + \gamma\mathbf{c} \quad \text{where } \alpha + \beta + \gamma = 1$$

  2. Use the same weights to compute the interpolated value *f*(**x**) from the values at the vertices, *f*(**a**), *f*(**b**), and *f*(**c**)

     $$f(\mathbf{x}) = \alpha f(\mathbf{a}) + \beta f(\mathbf{b}) + \gamma f(\mathbf{c})$$

See Shirley Sec. 2.7

Interpolation in ray tracing

- When values are stored at vertices, use linear (barycentric) interpolation to define values across the whole surface that:
  1. …match the values at the vertices
  2. …are continuous across edges
  3. …are piecewise linear (linear over each triangle)

     as a function of 3D position, not screen position—more later
- How to compute interpolated values
  1. during triangle intersection compute barycentric coords
  2. use barycentric coords to average attributes given at vertices

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 texure 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 optimization
    - try to choose vertex (u,v)s to result in a smooth, low distortion map

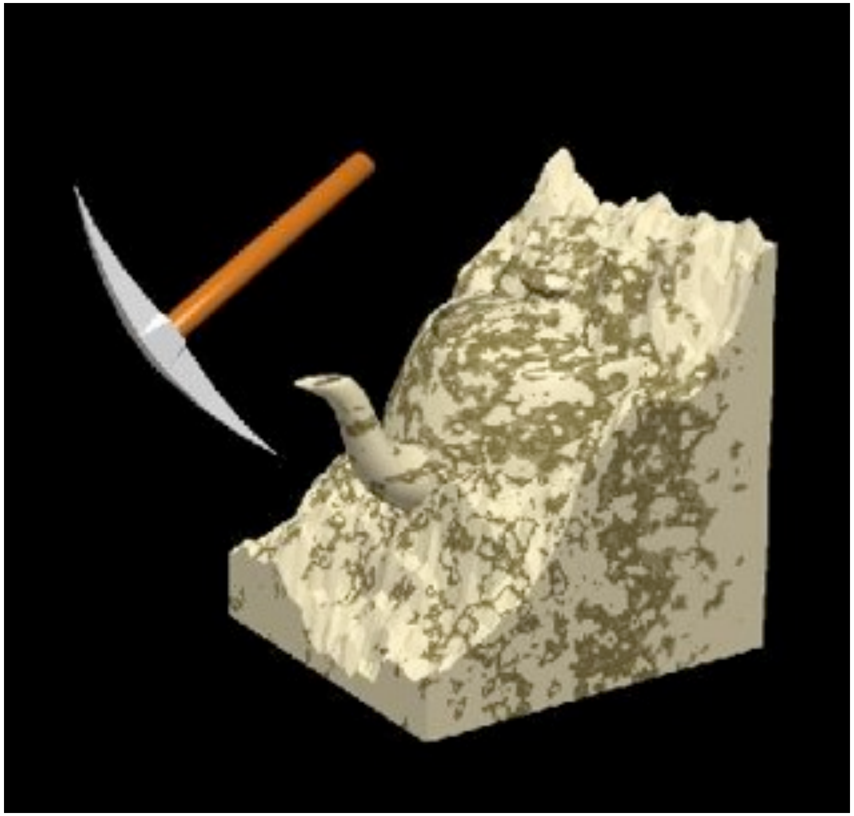
A refined definition

Texture mapping: a set of techniques for defining functions on surfaces, for a variety of uses.

- Let's look at some examples of more general uses of texture maps.

3D textures

- Texture is a function of (u, v, w)
  - can just evaluate texture at 3D surface point
  - good for solid materials
  - often defined procedurally



[Wolfe / SG97 Slide set]

