MAC420/5744: Introdução à Computação Gráfica

Marcel P. Jackowski
mjack@ime.usp.br

Aula #5: Tonalização (Parte I)

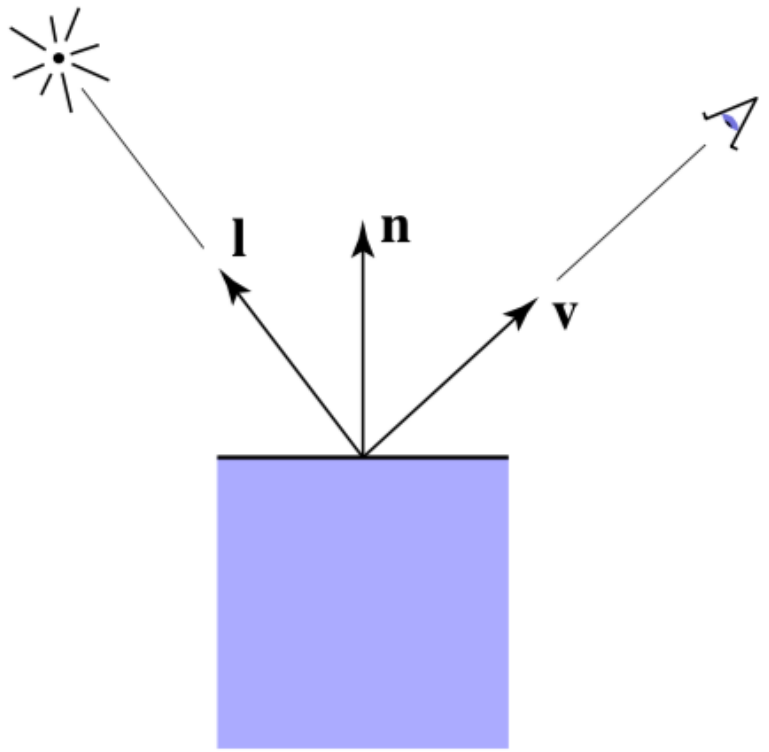
Some slides by Steve Marschner, Cornell CS4620/5620

Tonalização

Compute light reflected toward camera

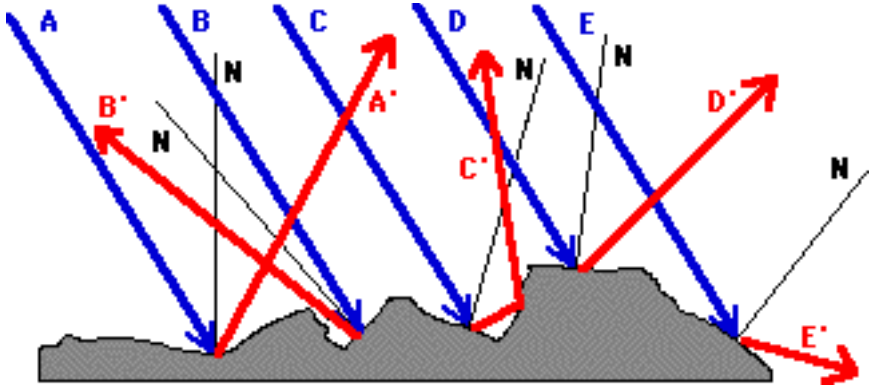
Inputs:

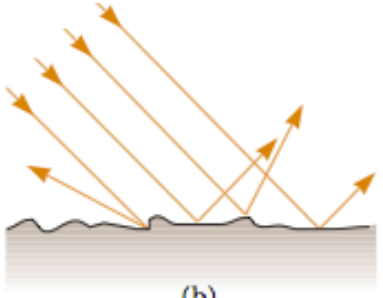
- eye direction
- light direction
  (for each of many lights)
- surface normal
- surface parameters
  (color, shininess, …)



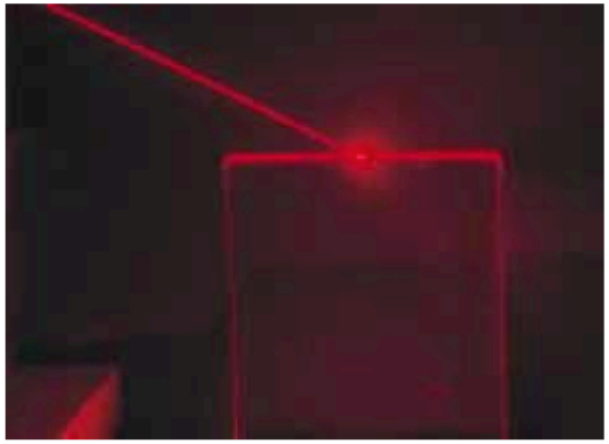
Reflexão difusa

- A maioria dos objetos, cujas superfícies são naturalmente rugosas, dão origem à reflexão difusa.





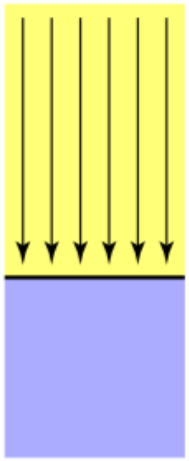
(b)



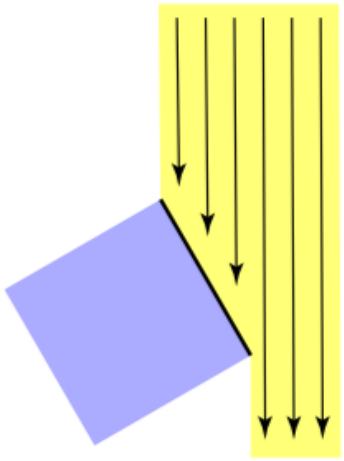
(d)

Reflexão difusa

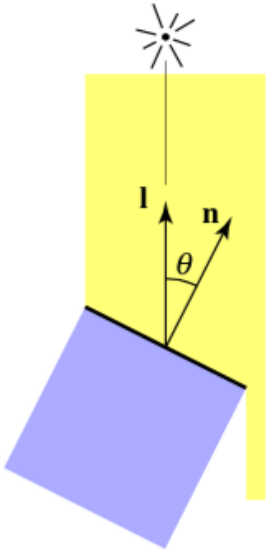
- Light is scattered uniformly in all directions
  - the surface color is the same for all viewing directions
- Lambert's cosine law



Top face of cube receives a certain amount of light

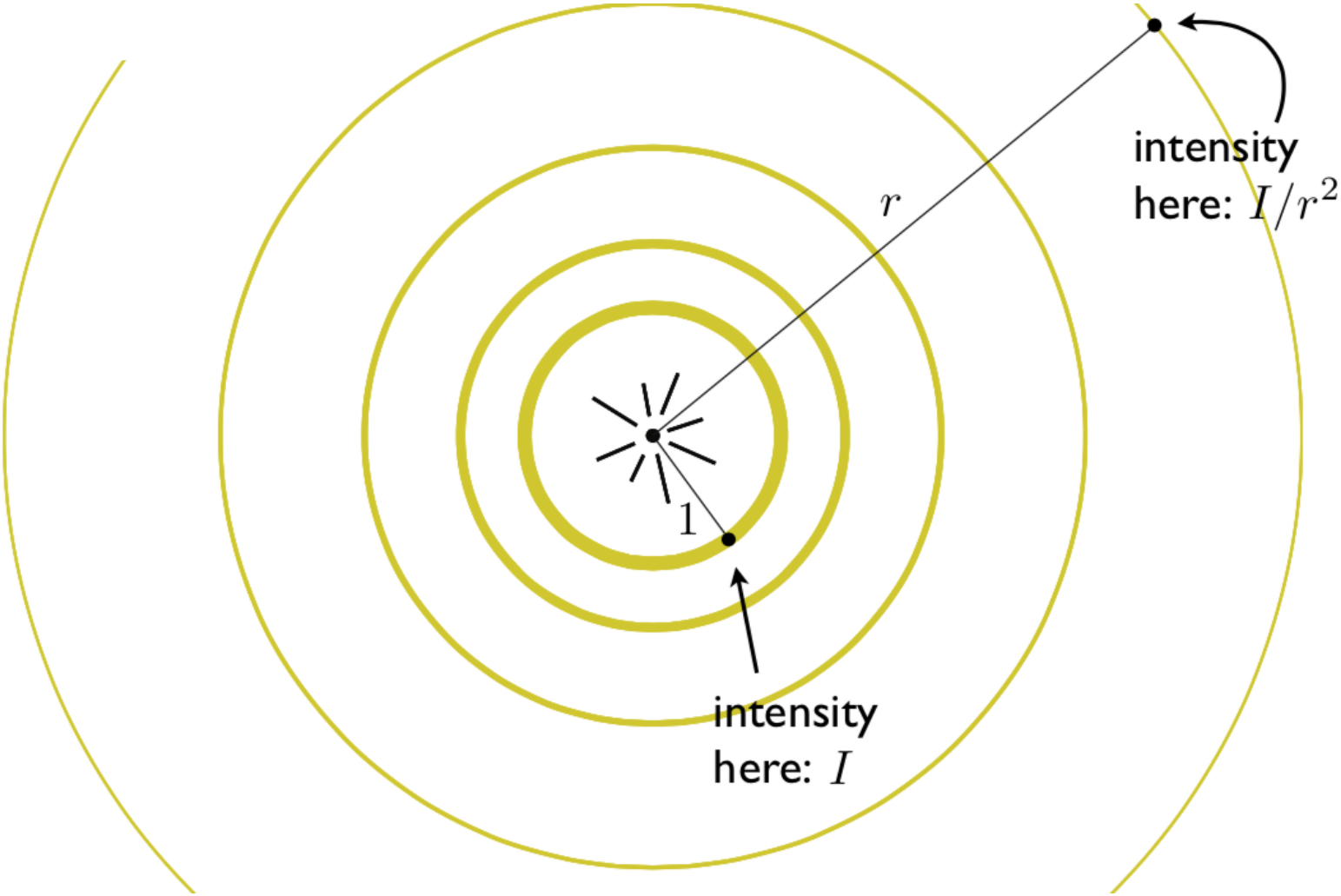


Top face of 60° rotated cube intercepts half the light



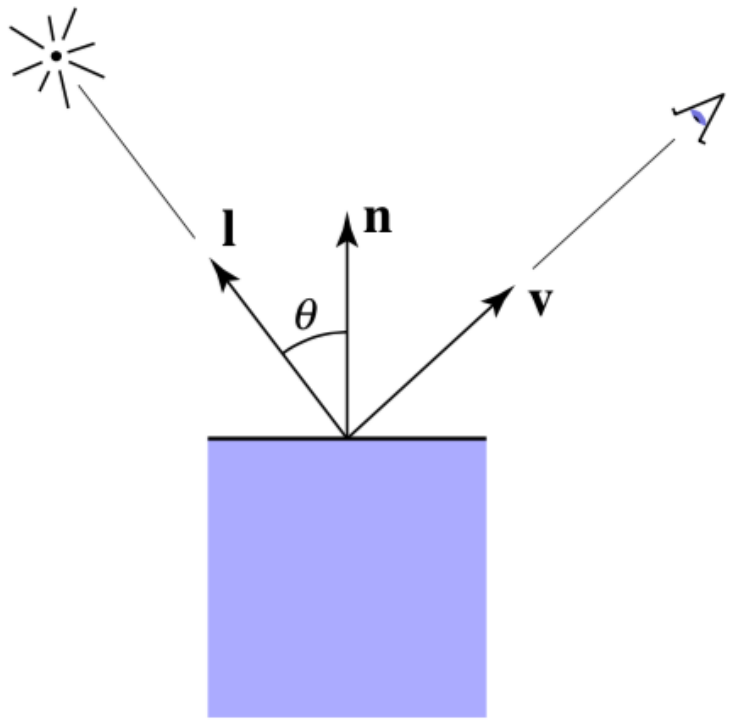
In general, light per unit area is proportional to $\cos\theta = \mathbf{l} \cdot \mathbf{n}$

Atenuação por distância



Reflexão Lambertiana

- Shading independent of view direction



illumination from source

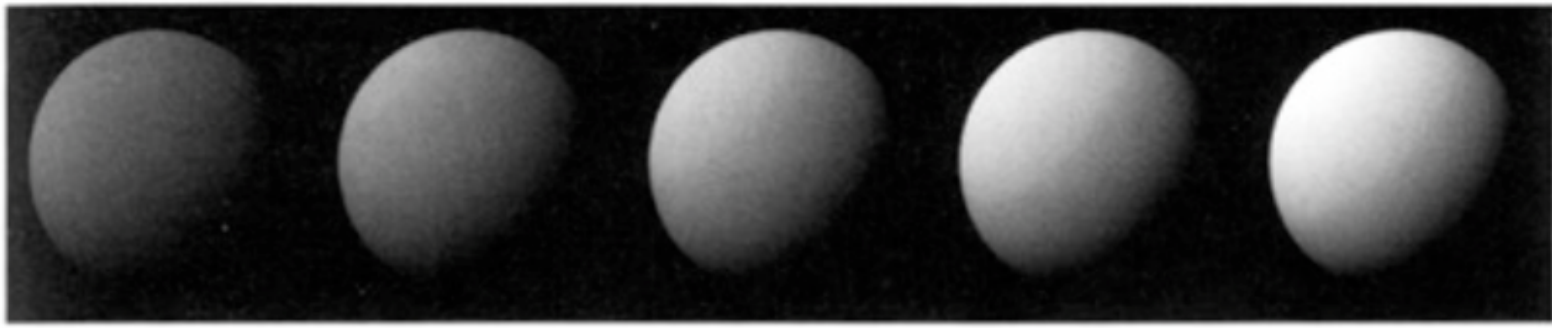
$$L_d = k_d \, (I/r^2) \max(0, \mathbf{n} \cdot \mathbf{l})$$

diffuse coefficient

diffusely reflected light

Tonalização difusa

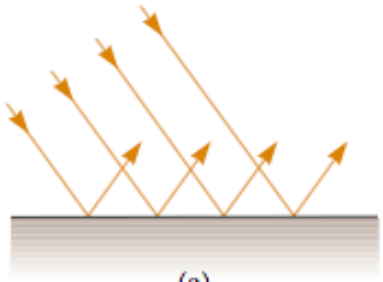
- Produces matte appearance



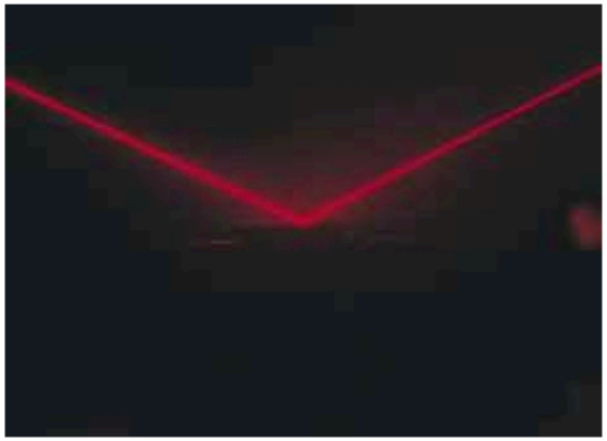
k_d ⟶

Reflexão especular

- Em materiais microscopicamente regulares (i.e. espelhos, água), a reflexão é especular.



(a)

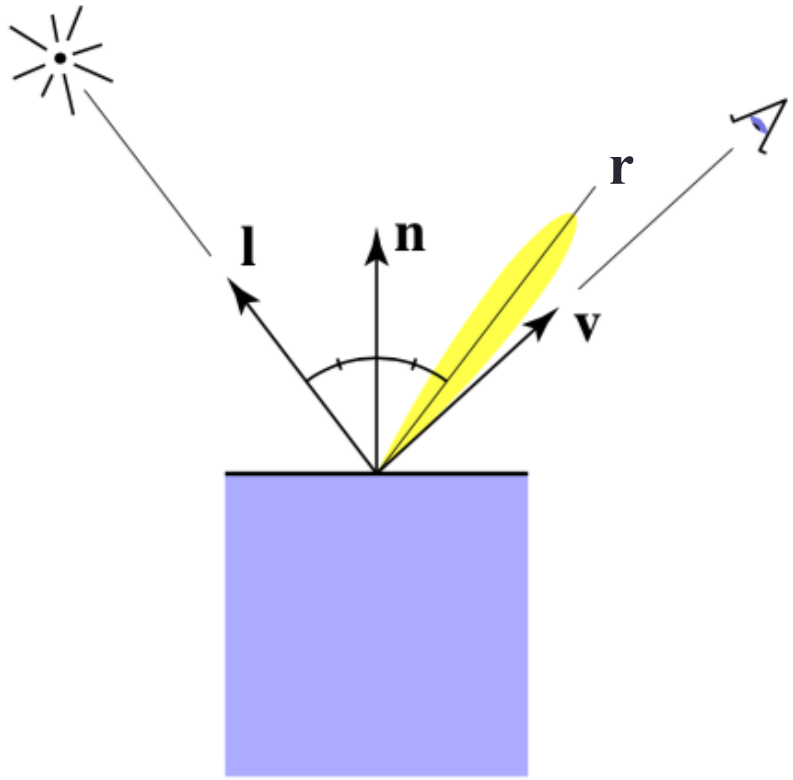


Courtesy of Henry Leap and Jim Lehman

(c)

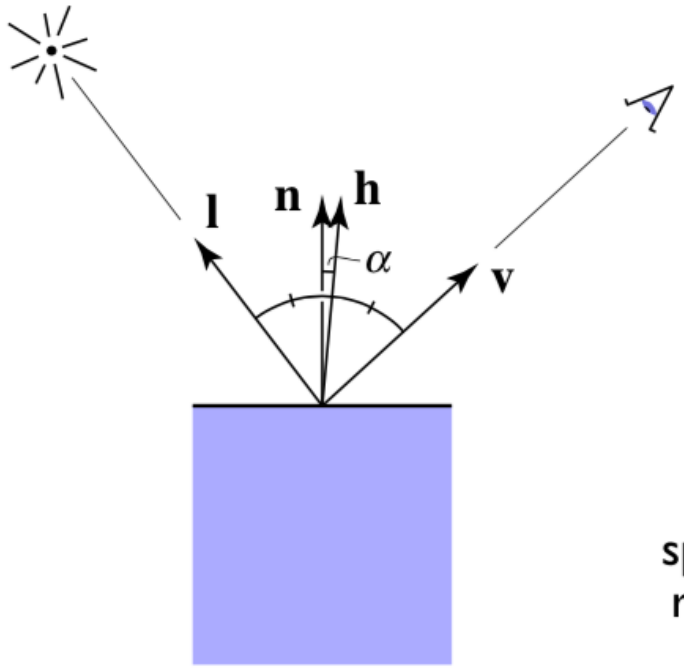
Reflexão especular

- Intensity depends on view direction
  - bright near mirror configuration



Modelo Blinn-Phong

- Close to mirror ⇔ half vector near normal
  - Measure "near" by dot product of unit vectors



$$\mathbf{h} = \text{bisector}(\mathbf{v}, \mathbf{l}) = \frac{\mathbf{v} + \mathbf{l}}{\|\mathbf{v} + \mathbf{l}\|}$$

$$L_s = k_s \, (I/r^2) \max(0, \cos\alpha)^p = k_s \, (I/r^2) \max(0, \mathbf{n} \cdot \mathbf{h})^p$$

specularly reflected light

specular coefficient

Modelo Phong/Blinn-Phong

- Increasing *n* narrows the lobe

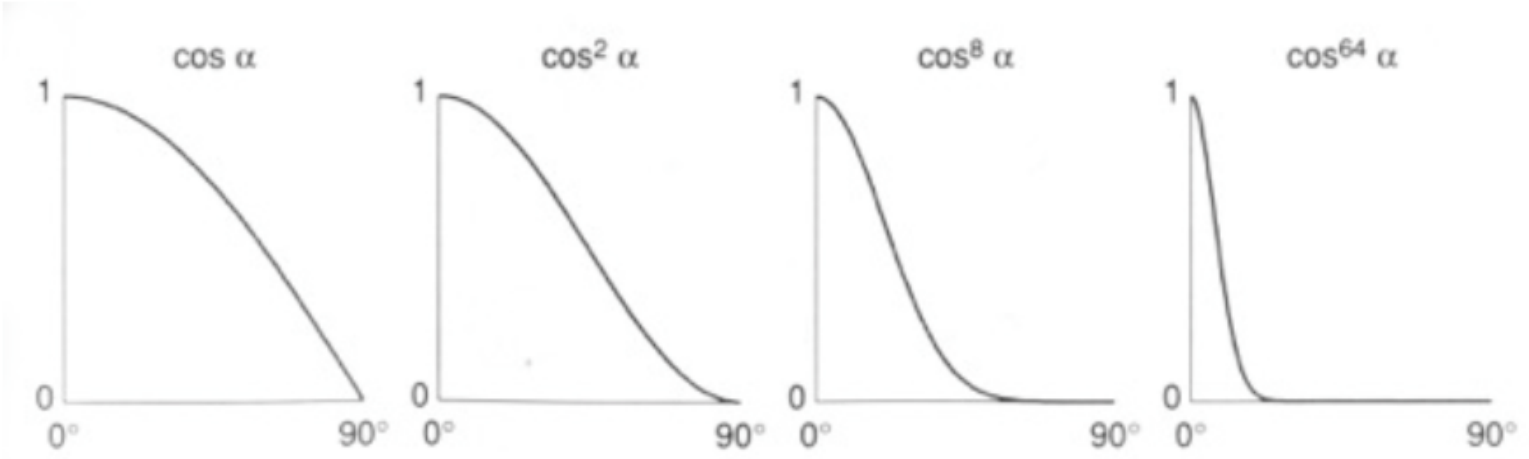
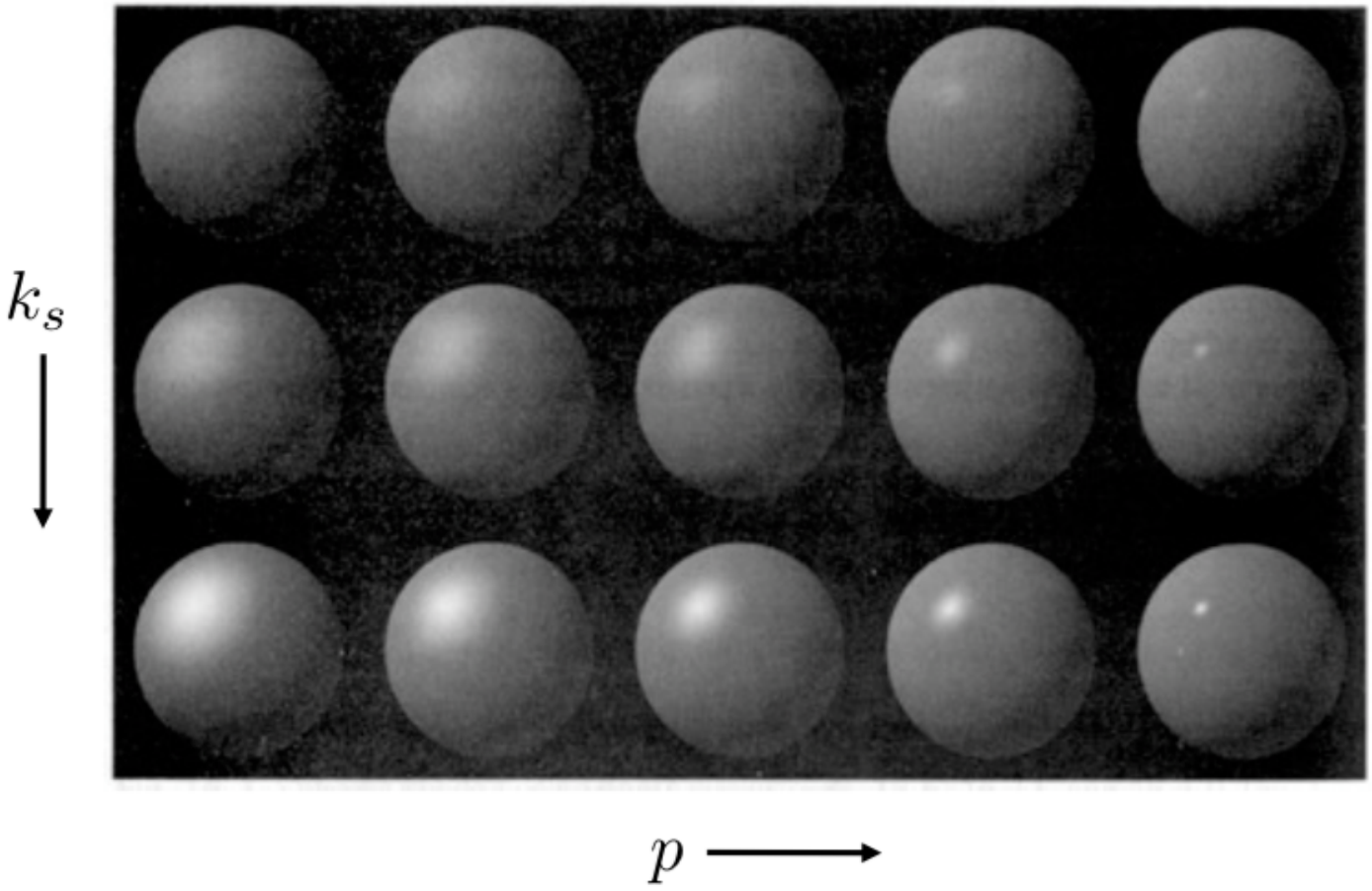


Fig. 16.9 Different values of $\cos^n \alpha$ used in the Phong illumination model.

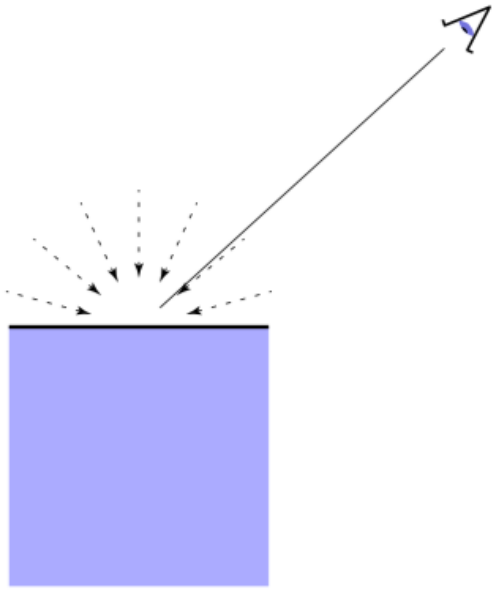
Valores entre 100 e 200 correspondem aos metais.
Valores entre 5 e 10 deixam a superfície com uma aparência plástica.

Tonalização especular



Iluminação ambiente

- Shading that does not depend on anything
  - add constant color to account for disregarded illumination and fill in black shadows



$$L_a = k_a I_a$$

ambient coefficient

reflected ambient light

Somando as contribuições

- Usually include ambient, diffuse, Phong in one model

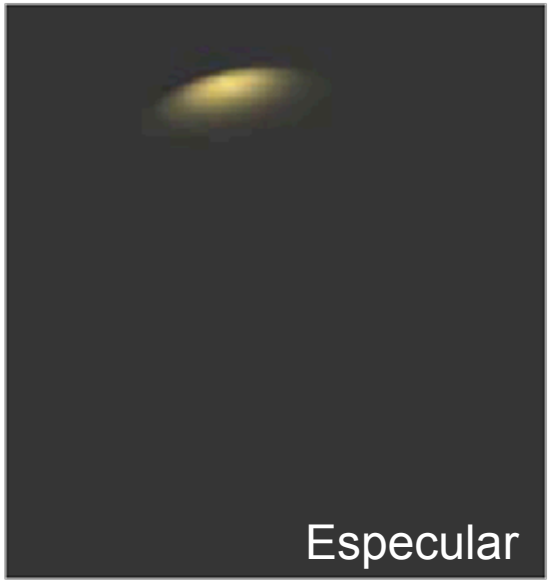
$$L = L_a + L_d + L_s$$
$$= k_a\, I_a + k_d\, (I/r^2) \max(0, \mathbf{n} \cdot \mathbf{l}) + k_s\, (I/r^2) \max(0, \mathbf{n} \cdot \mathbf{h})^p$$

- The final result is the sum over many lights

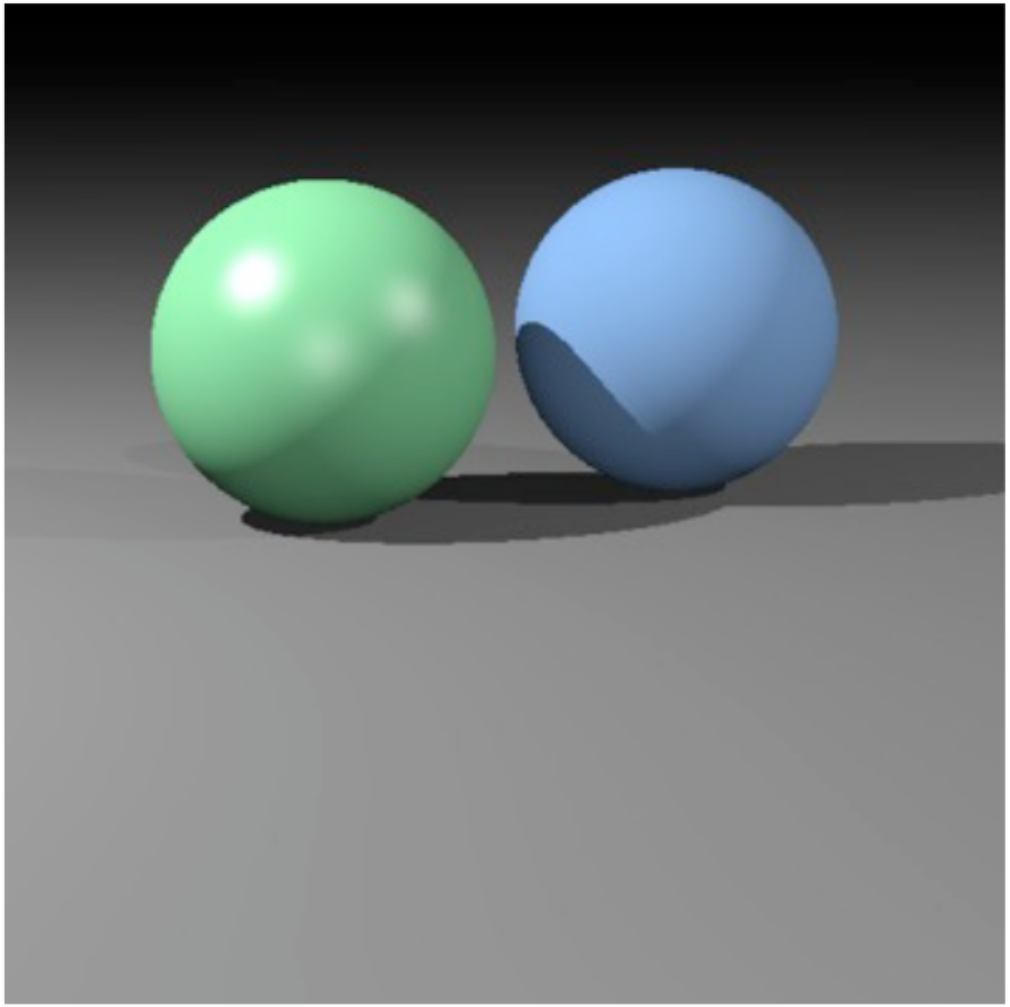
$$L = L_a + \sum_{i=1}^{N} [(L_d)_i + (L_s)_i]$$

$$L = k_a\, I_a + \sum_{i=1}^{N} \left[k_d\, (I_i/r_i^2) \max(0, \mathbf{n} \cdot \mathbf{l}_i) + \right.$$
$$\left. k_s\, (I_i/r_i^2) \max(0, \mathbf{n} \cdot \mathbf{h}_i)^p\right]$$

Componentes do modelo *Blinn-Phong*



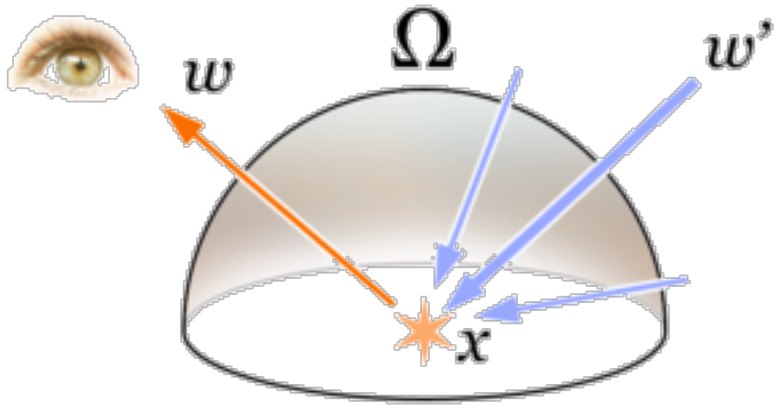
Tonalização difusa + especular



Equação de renderização

- Esse espalhamento infinito e absorção de luz pode ser descrito através da *equação de renderização*.

$$L_o(\mathbf{x}, \omega, \lambda, t) = L_e(\mathbf{x}, \omega, \lambda, t) + \int_\Omega f_r(\mathbf{x}, \omega', \omega, \lambda, t) L_i(\mathbf{x}, \omega', \lambda, t)(-\omega' \cdot \mathbf{n}) d\omega'$$



Kajiya, James T., The rendering equation", *SIGGRAPH 1986*

Mapeamentos de textura

- Objects have properties that vary across the surface

Mapeamentos de textura

- So we make the shading parameters vary across the surface

Mapeamentos de textura

- Adds visual complexity; makes appealing images

Mapeamentos de textura

- 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

Definição

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

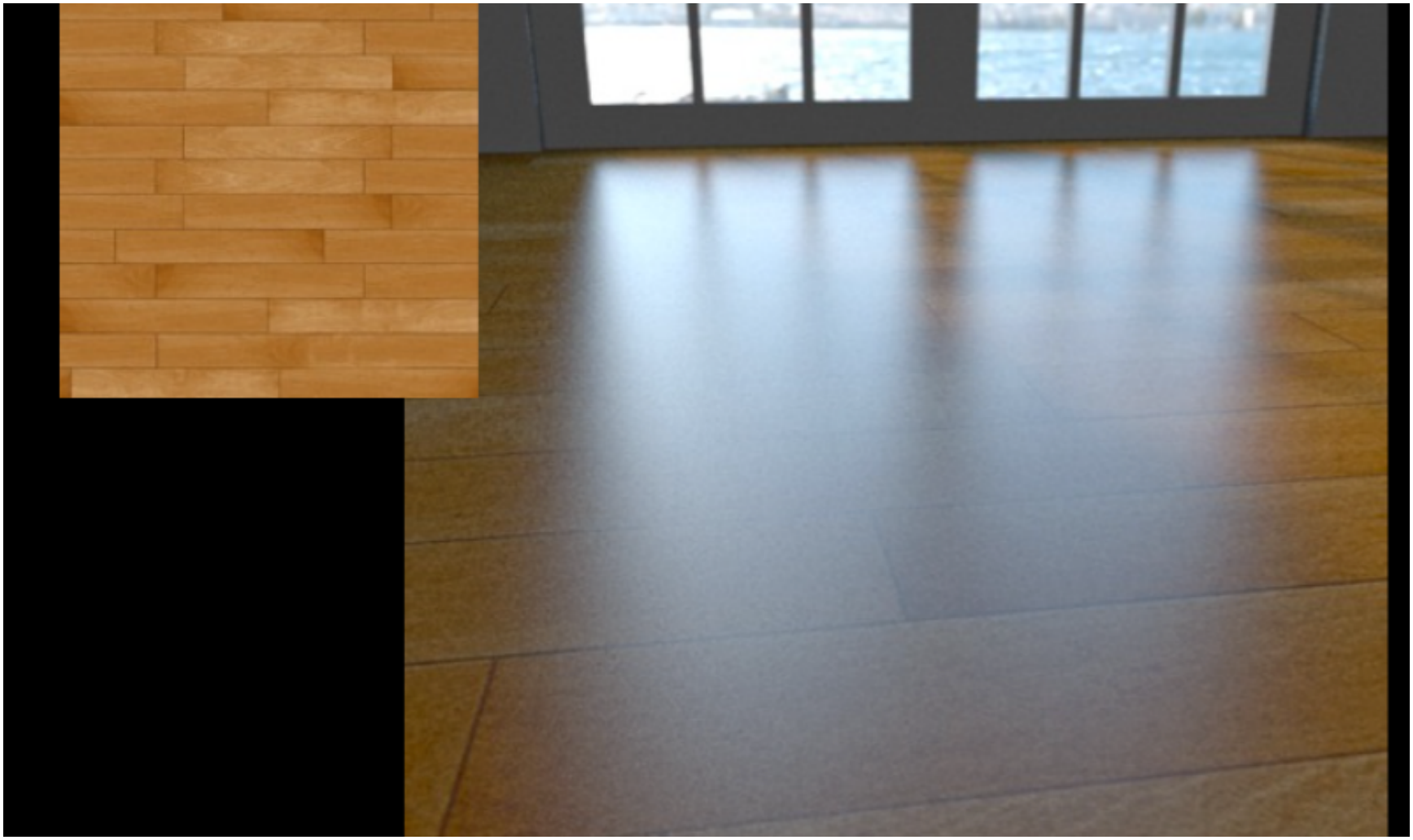
Exemplos

- 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

Exemplos



Exemplos

Mapeamento em superfícies

- 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

Parameterização de superfícies

- A surface in 3D is a two-dimensional thing
- Sometimes we need 2D coordinates for points on the surface
- Defining these coordinates is *parameterizing* the surface
- Examples:
  - cartesian coordinates on a rectangle (or other planar shape)
  - cylindrical coordinates (θ, y) on a cylinder
  - latitude and longitude on the Earth's surface
  - spherical coordinates (θ, ϕ) on a sphere

Exemplo: esfera unitária

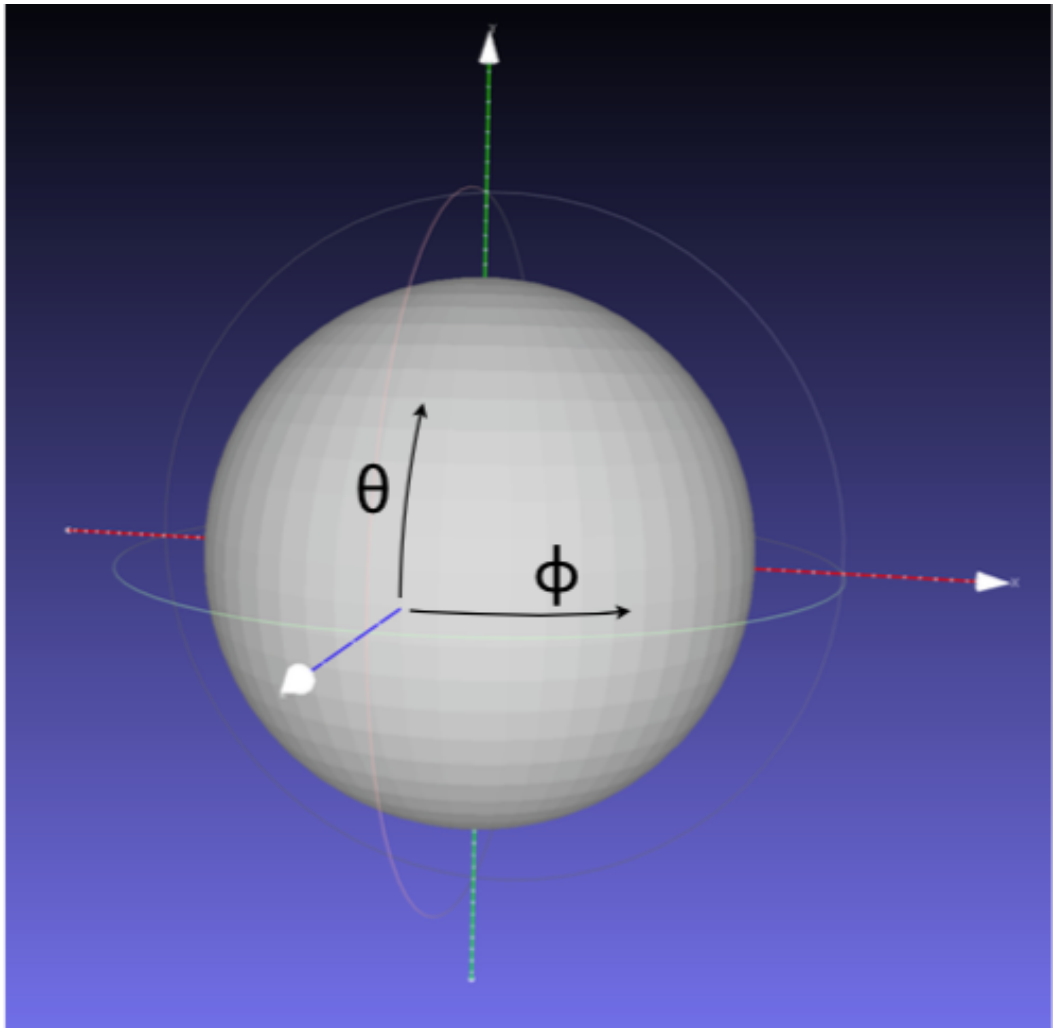
- position:

$$x = \cos\theta \sin\phi$$

$$y = \sin\theta$$

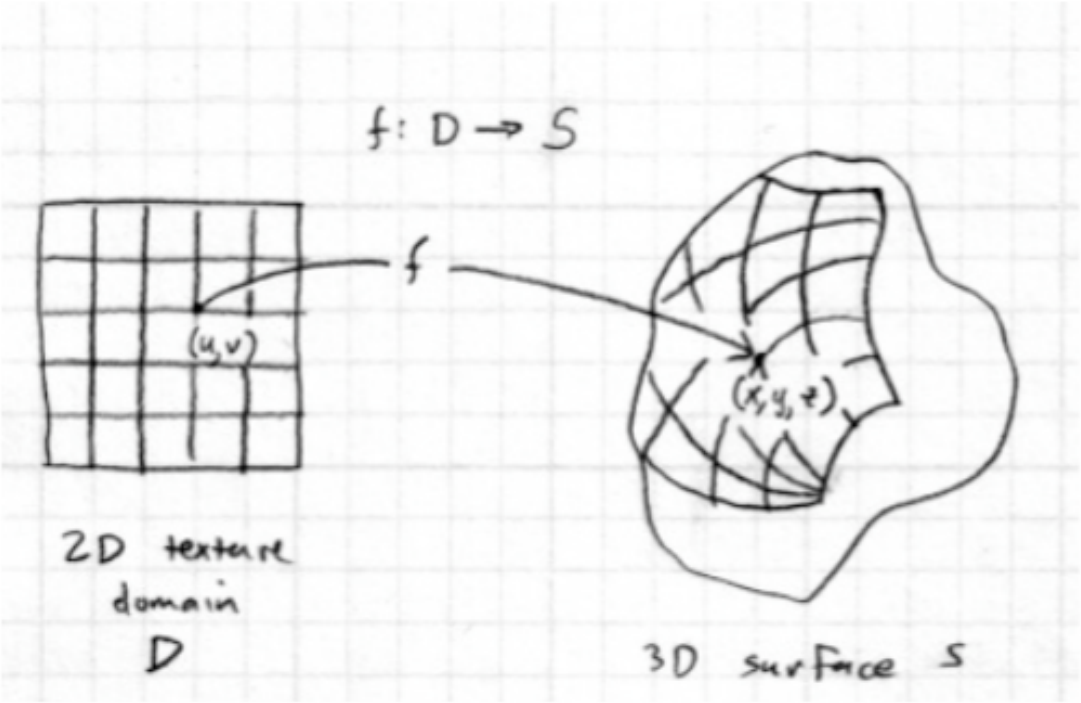
$$z = \cos\theta \cos\phi$$

- normal is position (easy!)



Mapeamento em superfícies

- "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

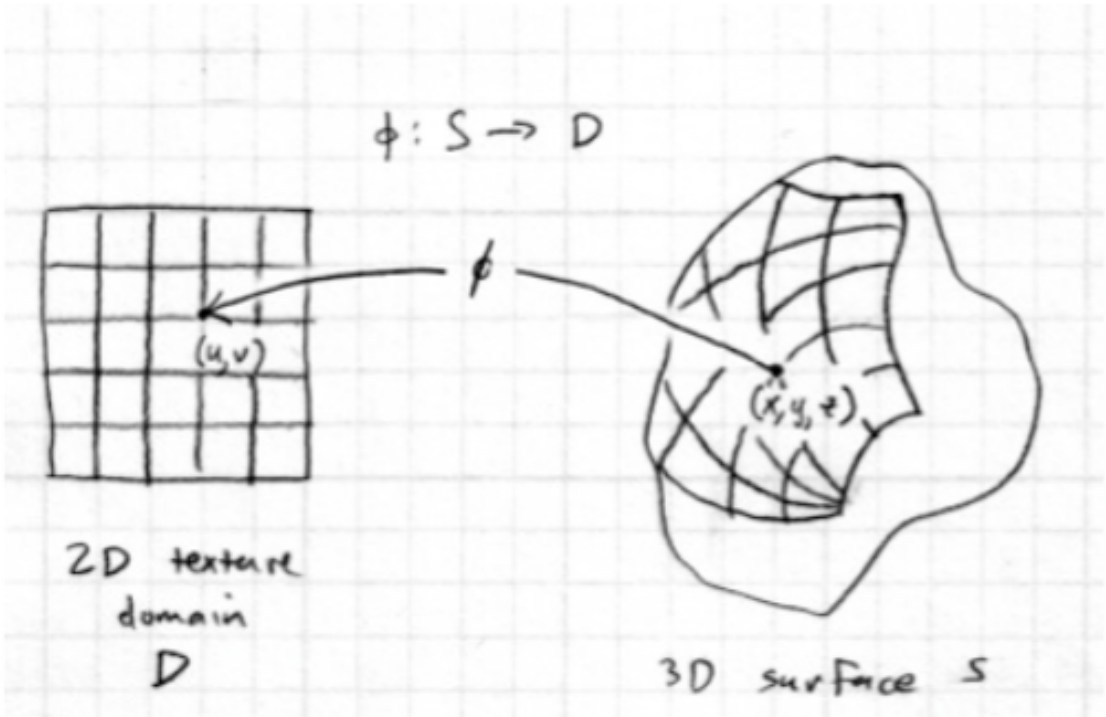


Mapeamento em superfícies

- 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.

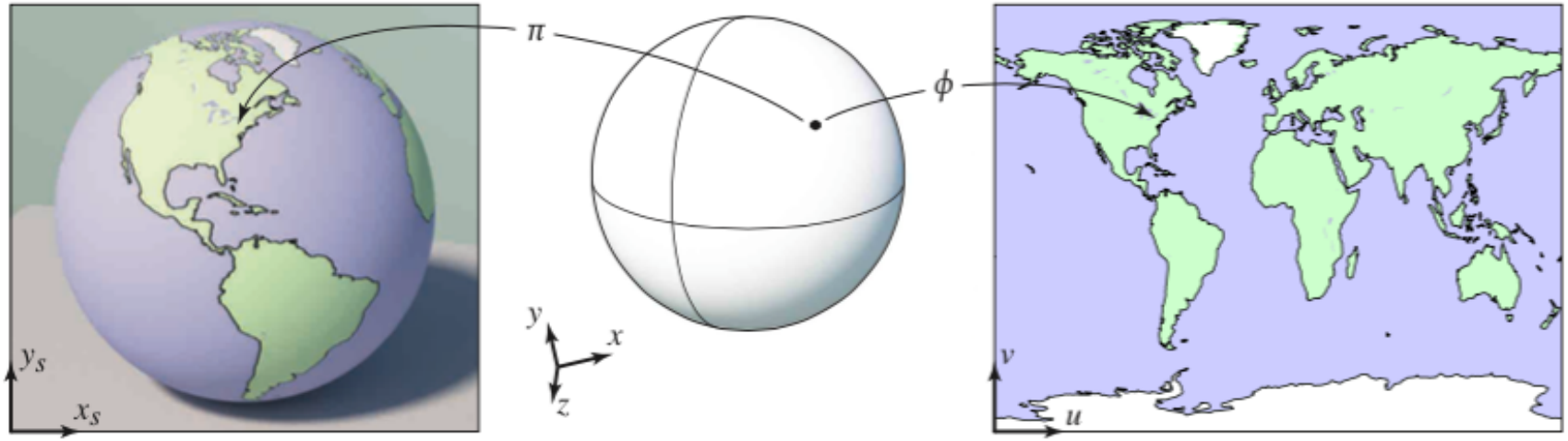
  $\phi : S \rightarrow \mathbb{R}^2$

  - when shading **p** get texture at ϕ(**p**)



Três espaços a considerar

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



Funções de mapeamento

- Define texture image as a function

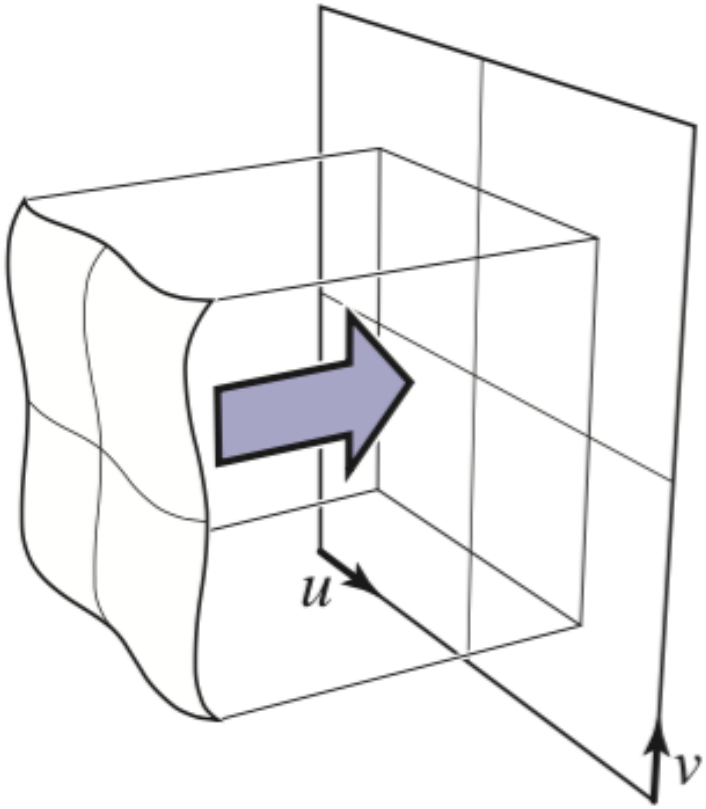
$$T : D \rightarrow C$$

  - where *C* is the set of colors for the diffuse component
- Diffuse color (for example) at point **p** is then

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

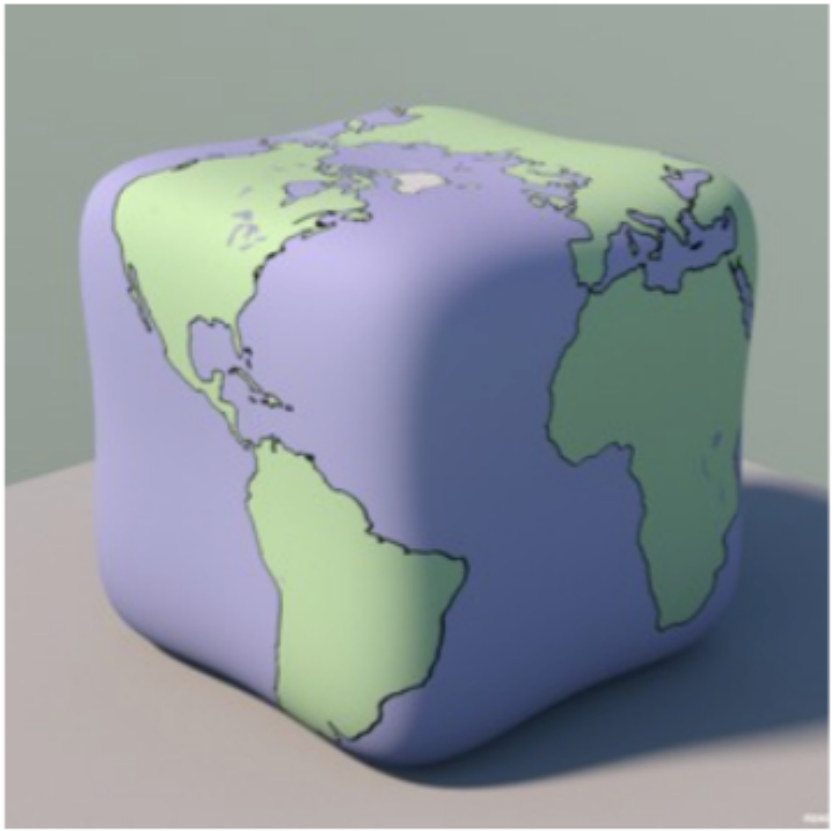
Exemplo de mapeamento

- Planar projection



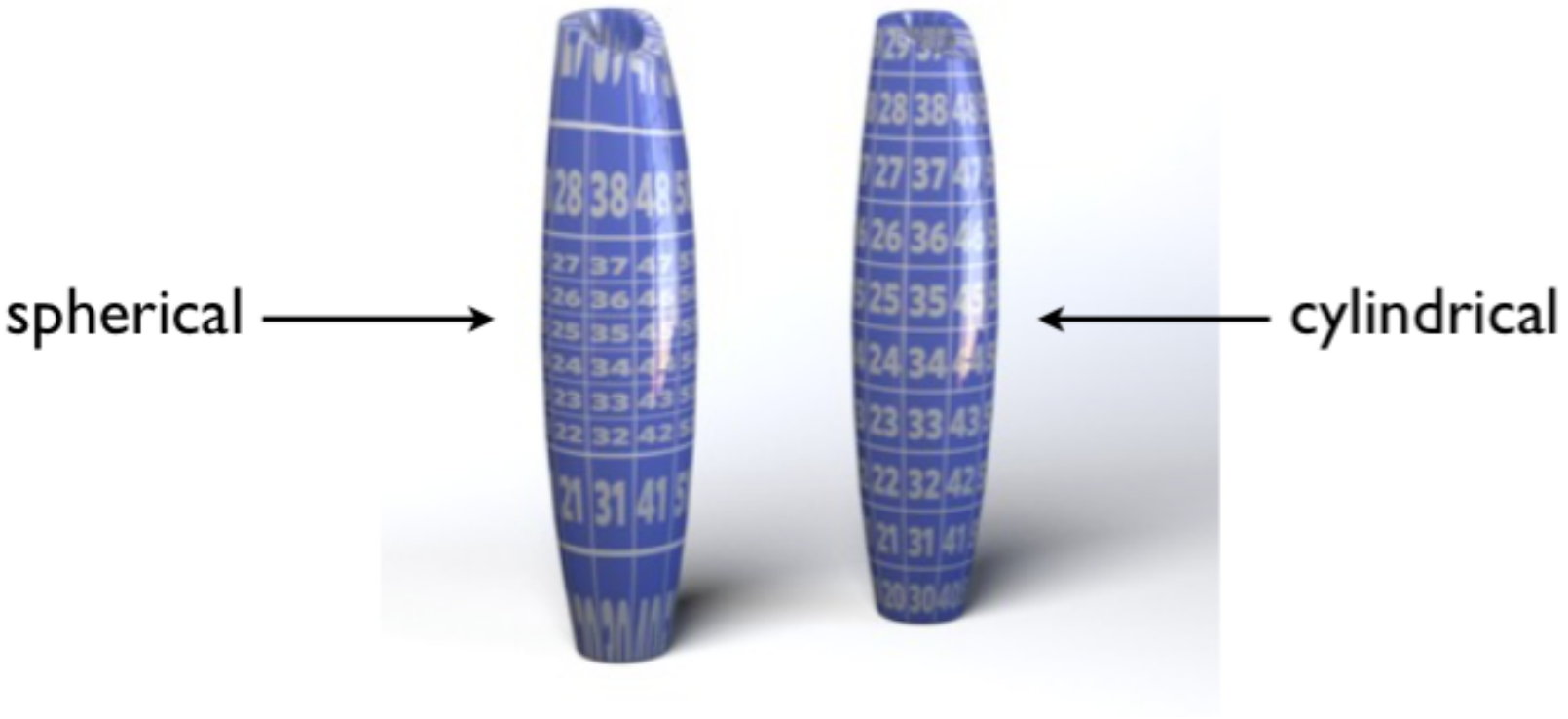
Exemplo de mapeamento

- Spherical projection



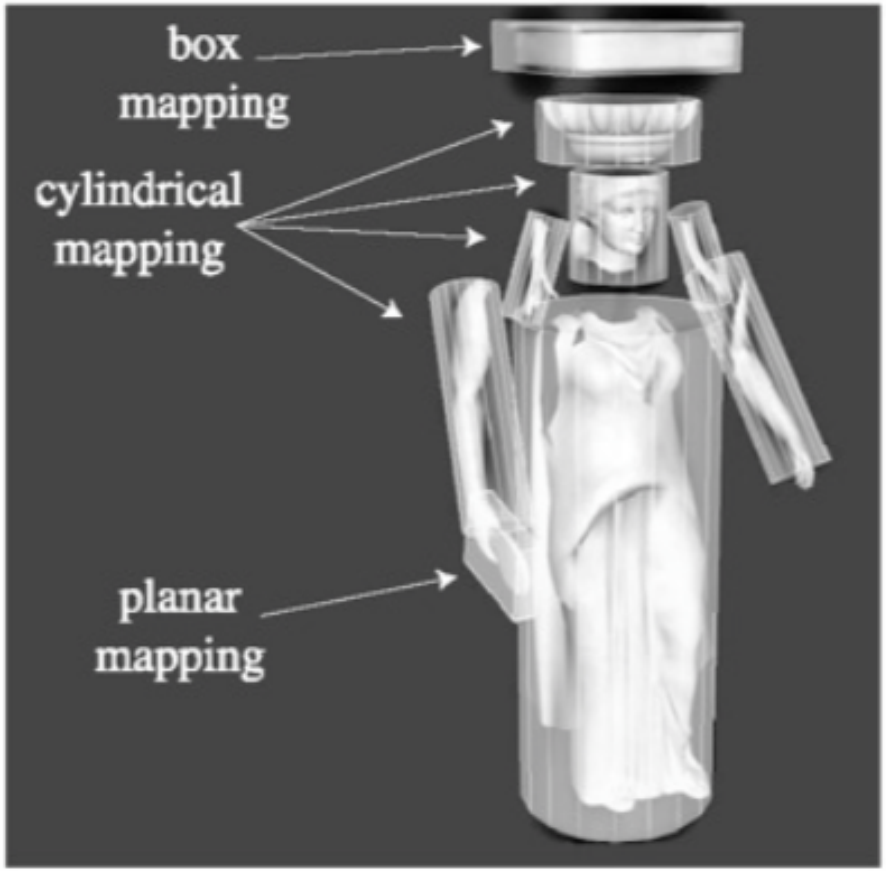
Exemplo de mapeamento

- Cylindrical projection



Exemplo de mapeamento

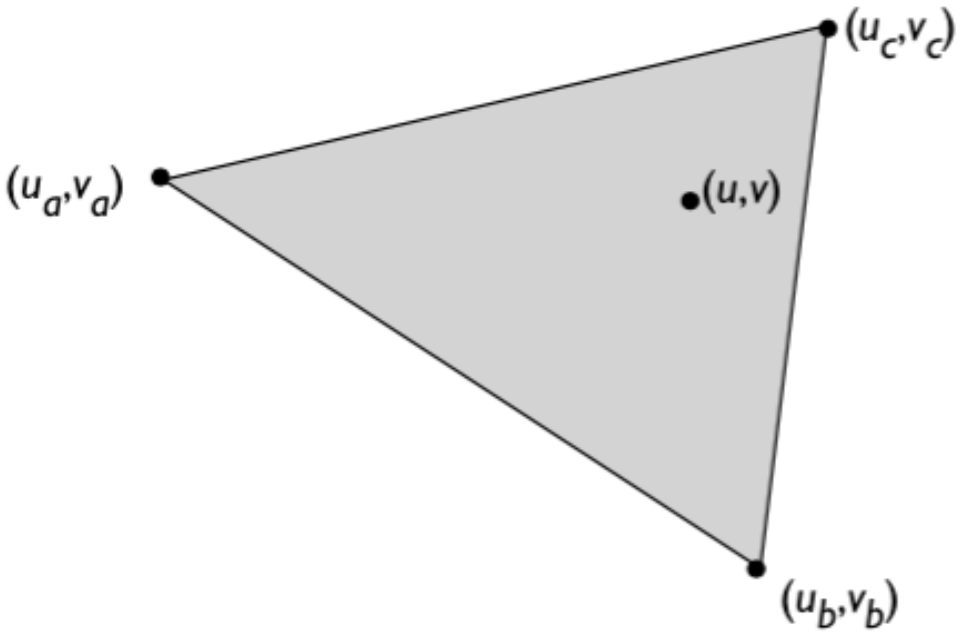
- Complex surfaces: project parts to parametric surfaces



[Tito Pagan]

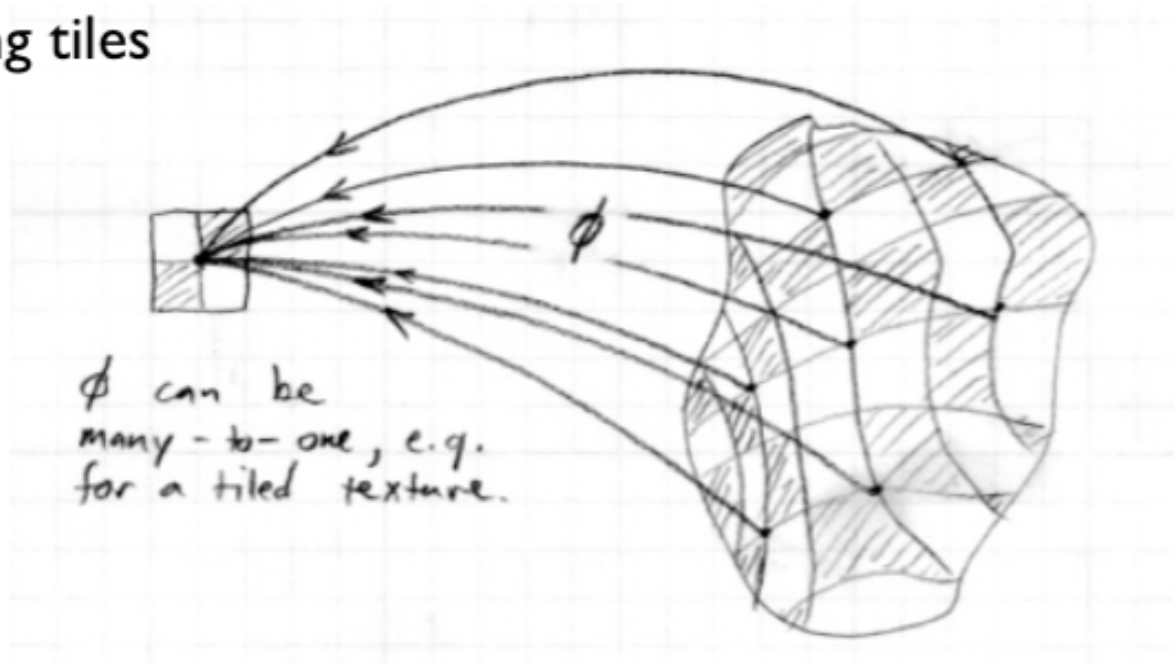
Coordenadas de textura

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



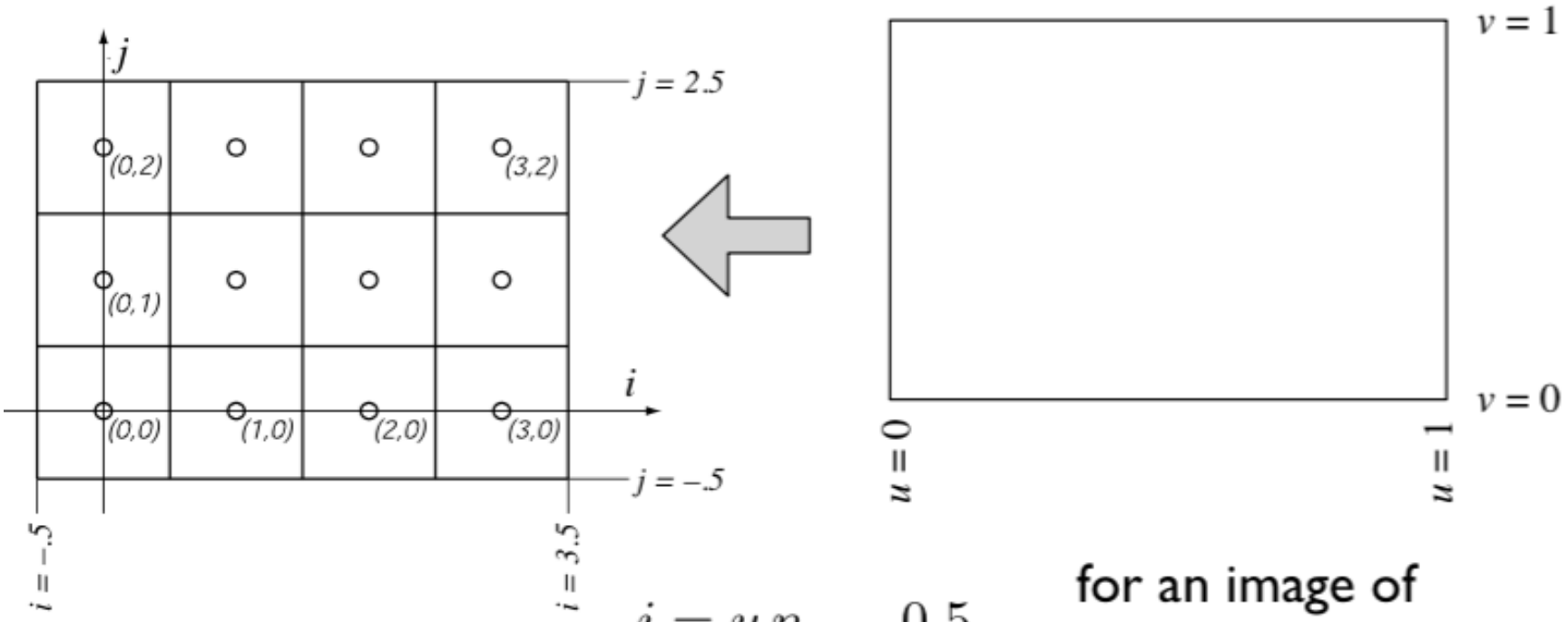
Funções de mapeamento

- 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 em texturas (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

Lookup e 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

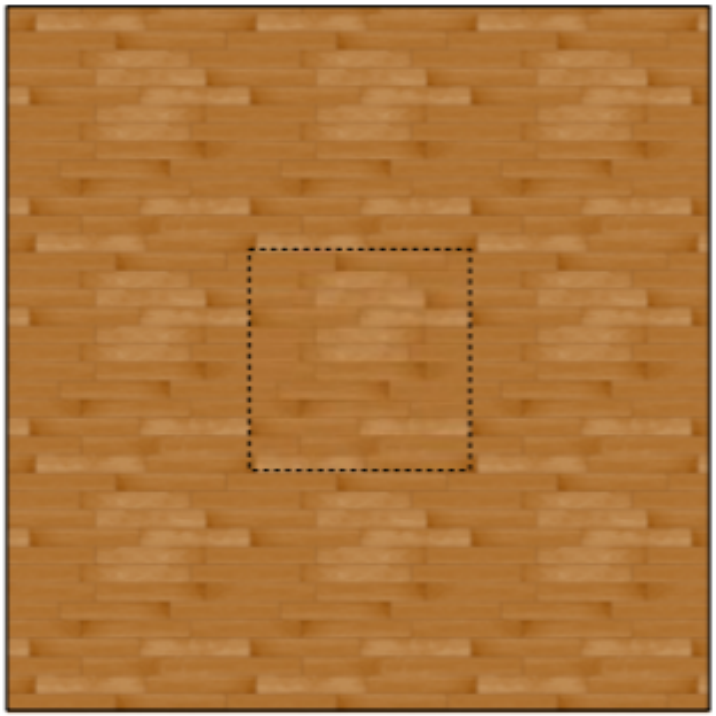
    $$i_{\text{pixel}} = \max(0, \min(n_x - 1, i_{\text{lookup}}))$$

  - 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

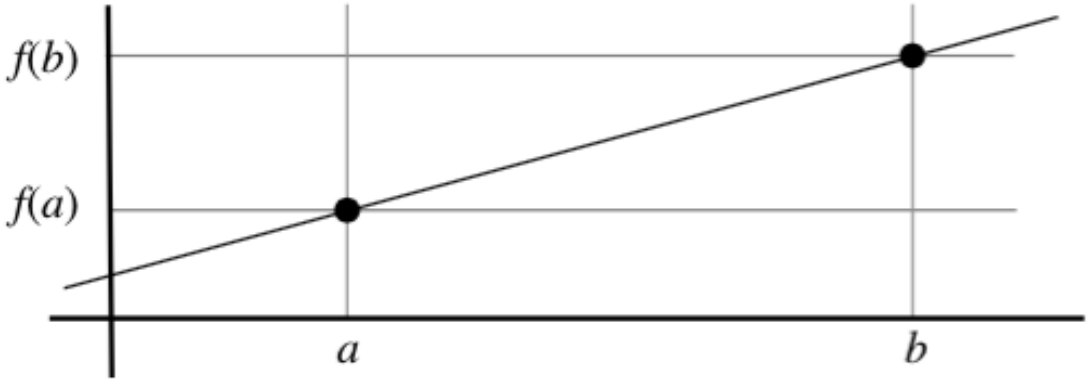
clamp



wrap

Interpolação linear

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



  - 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)

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

Interpolação linear

- 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*)

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

Interpolação linear em 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**)

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

Interpolação em 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)

- How to compute interpolated values
  1. during triangle intersection compute barycentric coords
  2. use barycentric coords to average attributes given at vertices

Coordenadas de textura em malhas

- 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

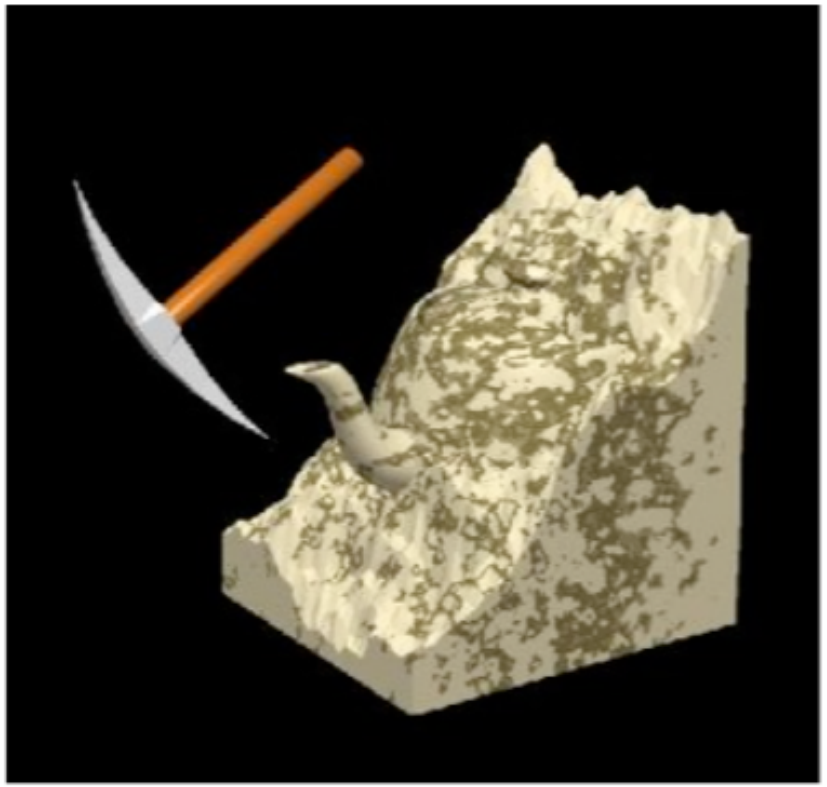
Definição (revisada)

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.

Texturas em 3D

- 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]

Síntese de imagens

```
for each object in the scene {
    for each pixel in the image {
        if (object affects pixel) {
            do something
        }
    }
}
```

object order
or
rasterization

```
for each pixel in the image {
    for each object in the scene {
        if (object affects pixel) {
            do something
        }
    }
}
```

image order
or
ray tracing

O modelo de tonalização que vimos aplica-se à ambas abordagens

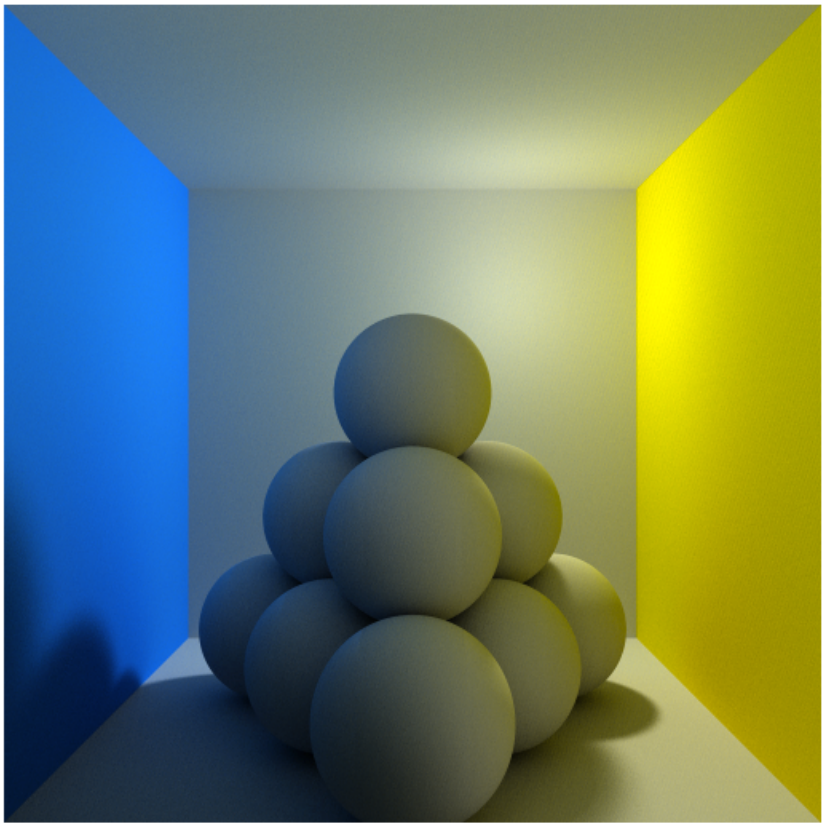
Ray tracers

- http://www.pbrt.org
- http://www.mitsuba-renderer.org/index.html

http://www.pbrt.org/gallery.php

Ray tracer em WebGL

- http://madebyevan.com/webgl-path-tracing/



Exercício

1. Derive as equações paramétricas de um tóro com raio *R* e raio interno *r*
2. Agora derive a representação paramétrica para seus vetores normais.