CS451 Texturing 3

Bumpmap + Basic Lighting

Jyh-Ming Lien

Department of Computer SCience

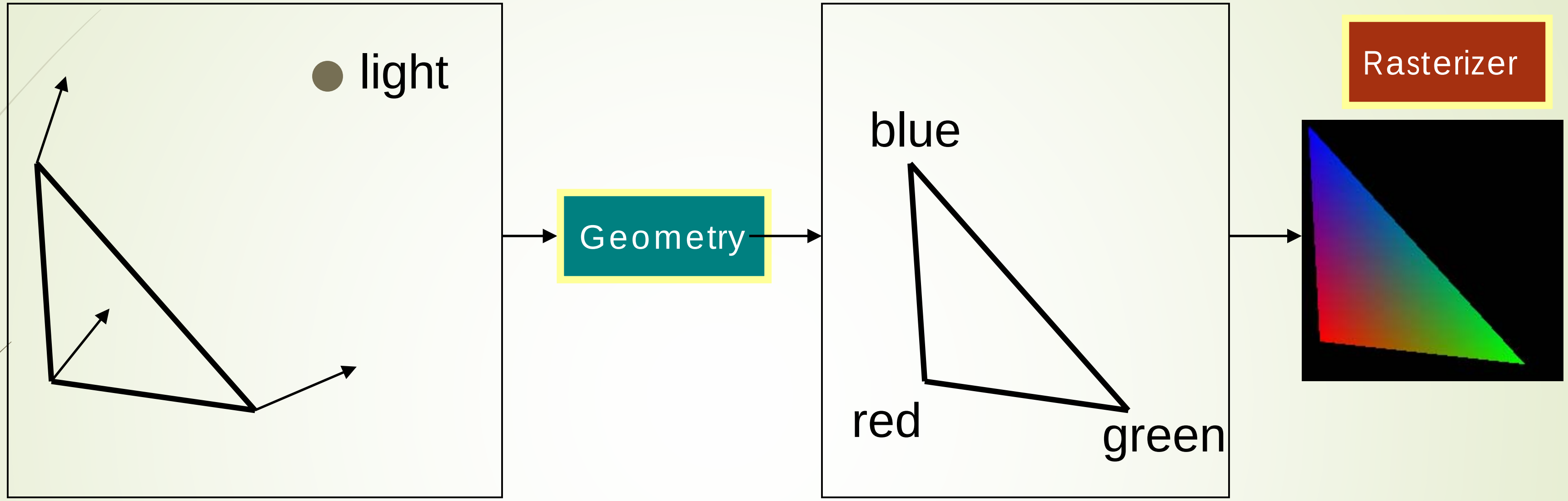
George Mason University

Based on Tomas Akenine-Möller's lecture note

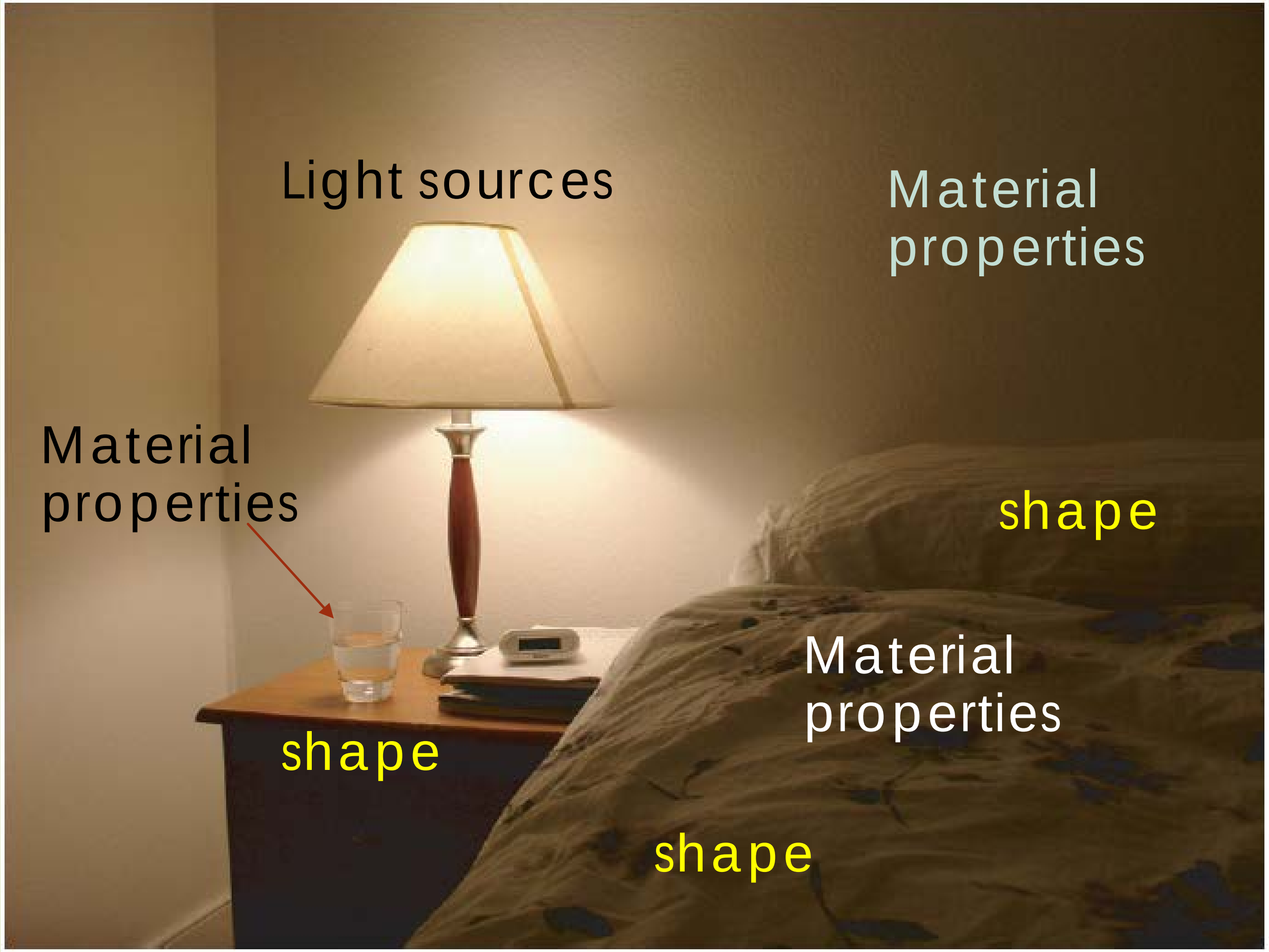
Note

- Skinning weights correction
  - this does not sum up to 1 $w_i = 1 - \frac{d_i}{\sum d_i}$
  - A simple hack $w_i = \frac{1/d_i}{\sum 1/d_i}$
  - Or, $x_i = d_{max} - d_i + c$ then $w_i = \frac{x_i}{\sum x_i}$
- Your PA2 is graded. Email your TA, he will share a Google doc that has all of your scores so far
- This lecture is from chapter 5 and chapter 6 in the textbook
  - This covers basic ideas in lighting; details in lighting and shading will come next (Chapter 7)
  - Your next assignment (PA4) will be Bumpmap

Compute lighting at vertices, then interpolate over triangle



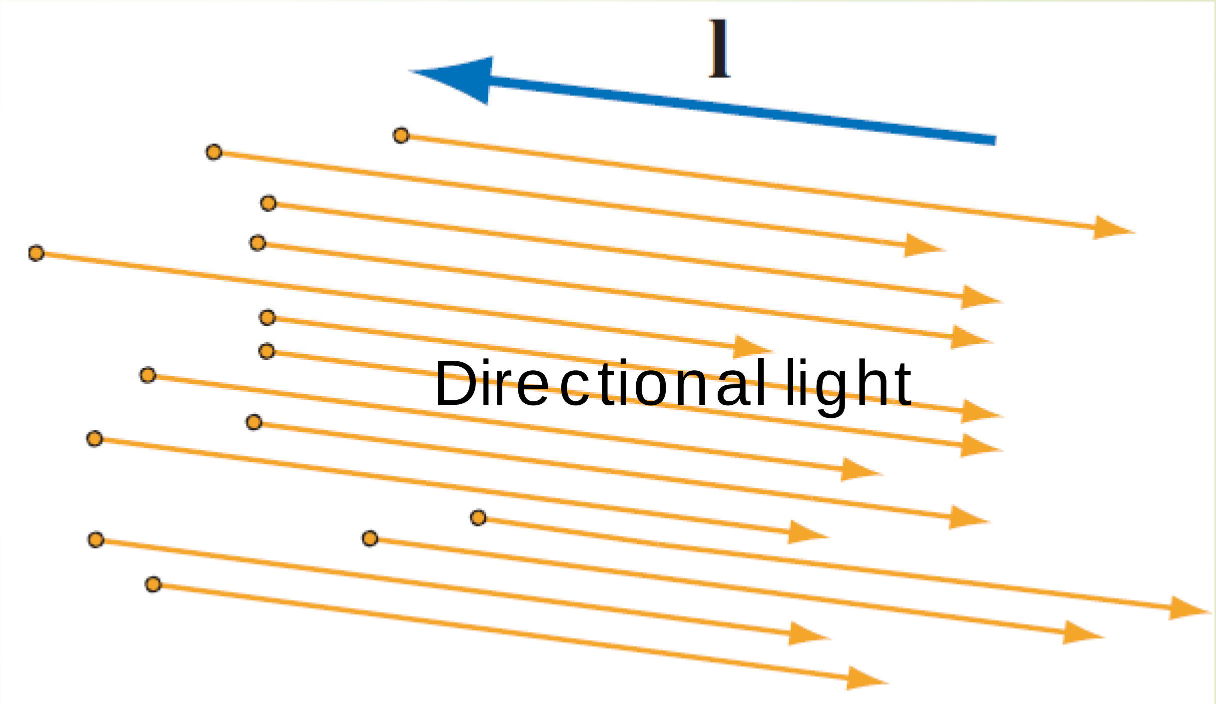
- How compute lighting?
- We could set colors per vertex manually
- For a **little** more realism, compute lighting from
  - Light sources
  - Material properties
  - Geometrical relationships


Light sources
Material properties
Material properties
shape
Material properties
shape
shape

Basic Ligthing

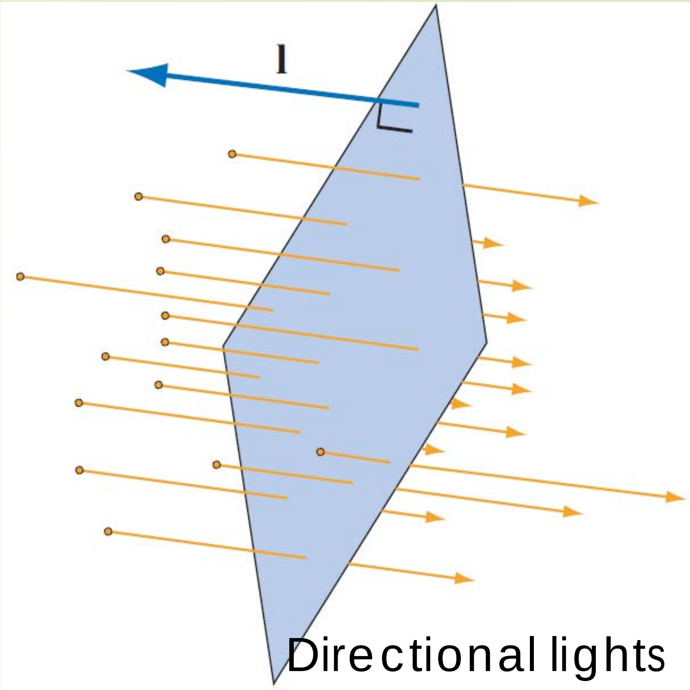
- Light souce
  - Directional light
  - Spot light
  - Ambient light
  - ...
- Light characteristic
  - Direction of light
  - Radiometry: Amount of illumination

Light vector
points in the opposite direction that the light is traveling

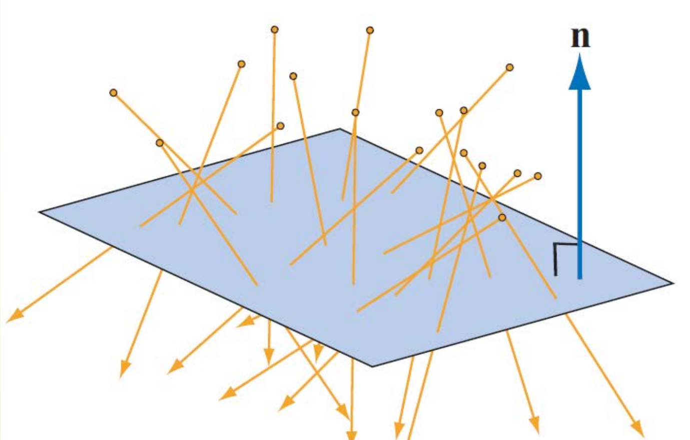


Basic Ligthing

- Irradiance
  - Sum of energies passing through a unit area perpendicualr to **l** in one second
- OpenGL hacks these with
  - Linear combinabtion of ambient, diffuse, spectular terms
  - $\mathbf{I} = \mathbf{i}_{amb} + \mathbf{i}_{diff} + \mathbf{i}_{spec}$



Directional lights



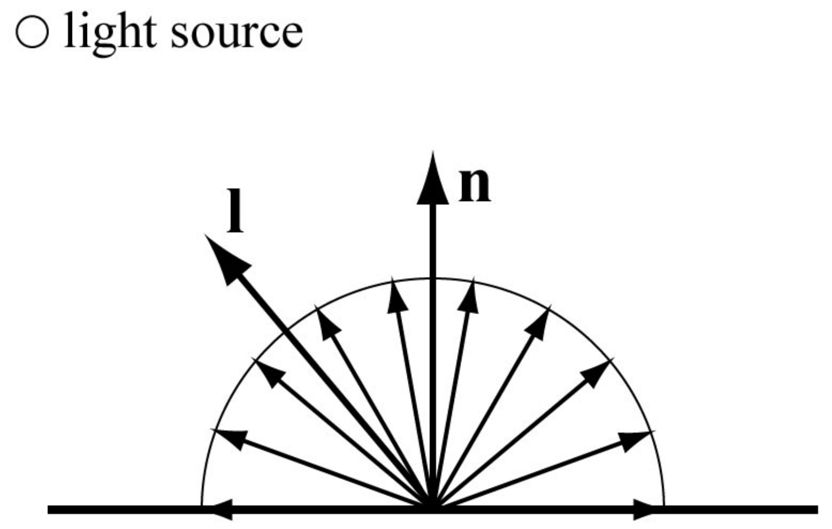
Irradiacnce measured in all incoming directions

Ligthing Difusse Term

- Diffuse is Lambert's law:

$$i_{diff} = \mathbf{n} \cdot \mathbf{l} = \cos\phi$$

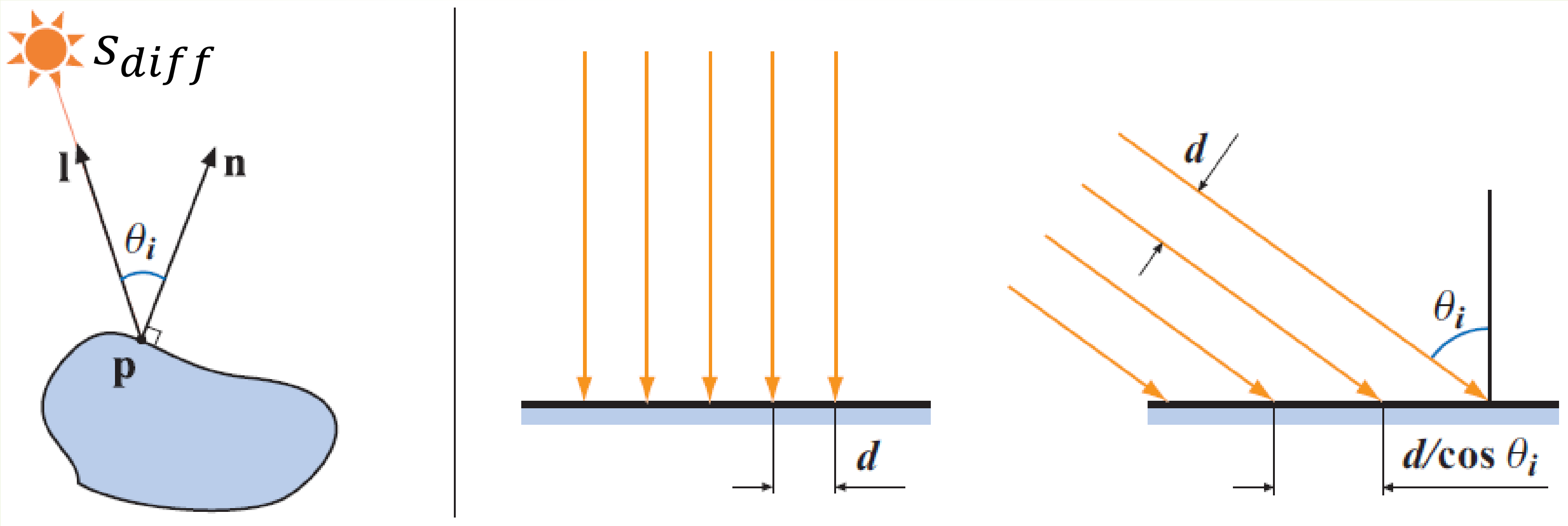
- Photons are scattered equally in all directions



Ligthing Difusse Term

- Photons are scattered equally in all directions

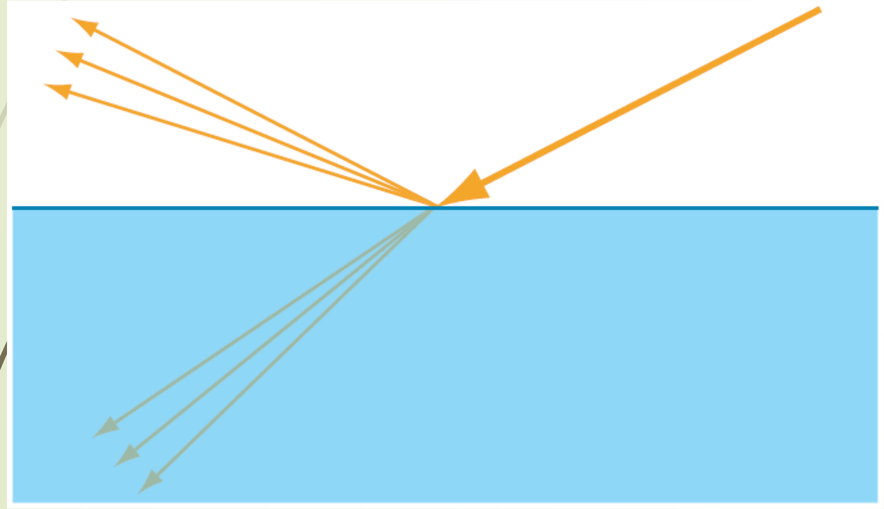
$$\mathbf{i}_{diff} = (\mathbf{n} \cdot \mathbf{l})\mathbf{m}_{diff} \otimes E_L$$



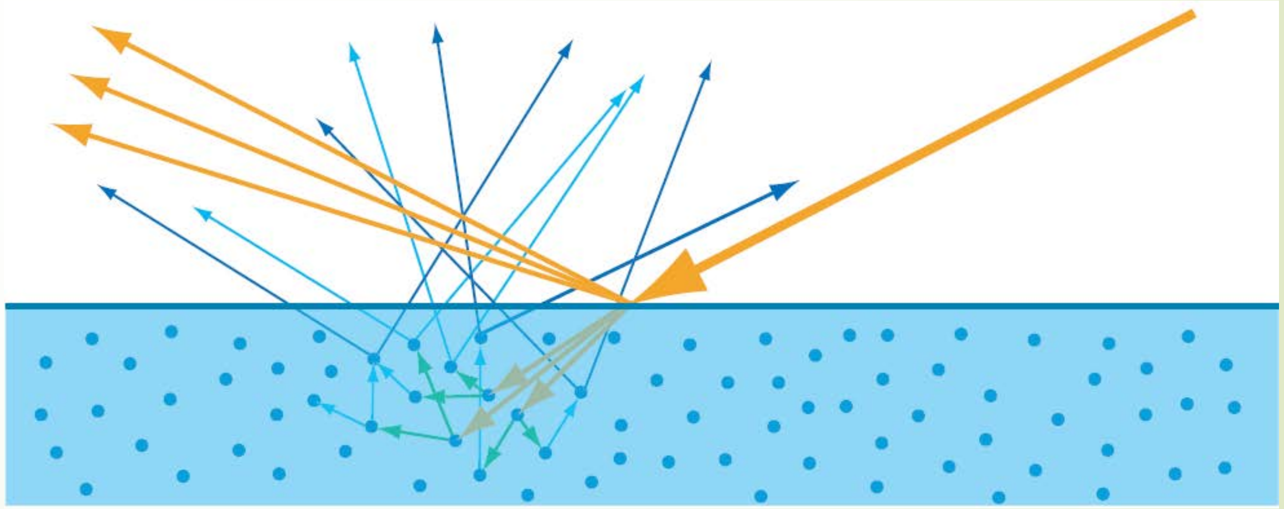
Ligthing Difusse Term

- Where does the term m_{diff} come from?

$$\mathbf{i}_{diff} = (\mathbf{n} \cdot \mathbf{l})\mathbf{m}_{diff} \otimes E_L$$

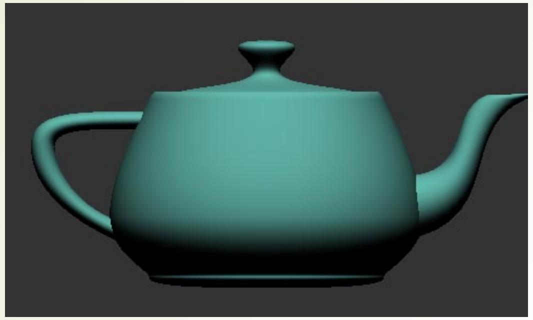


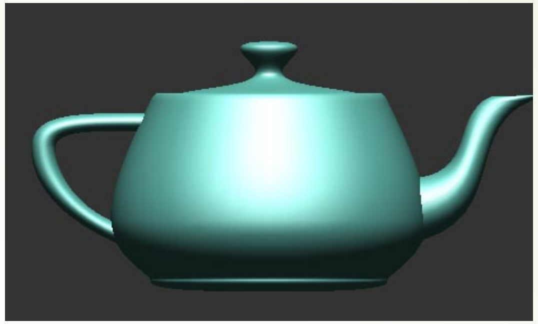
Material A



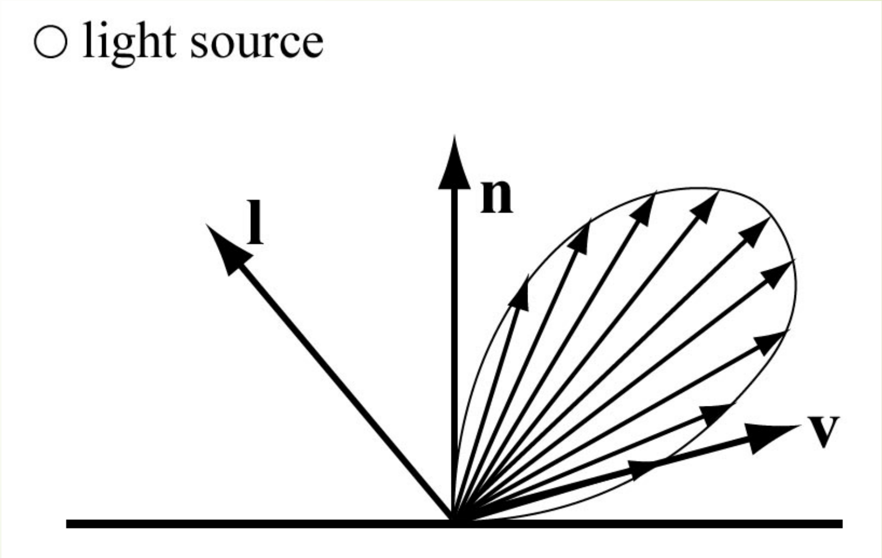
Material B

Lighting Specular Term: i_{spec}





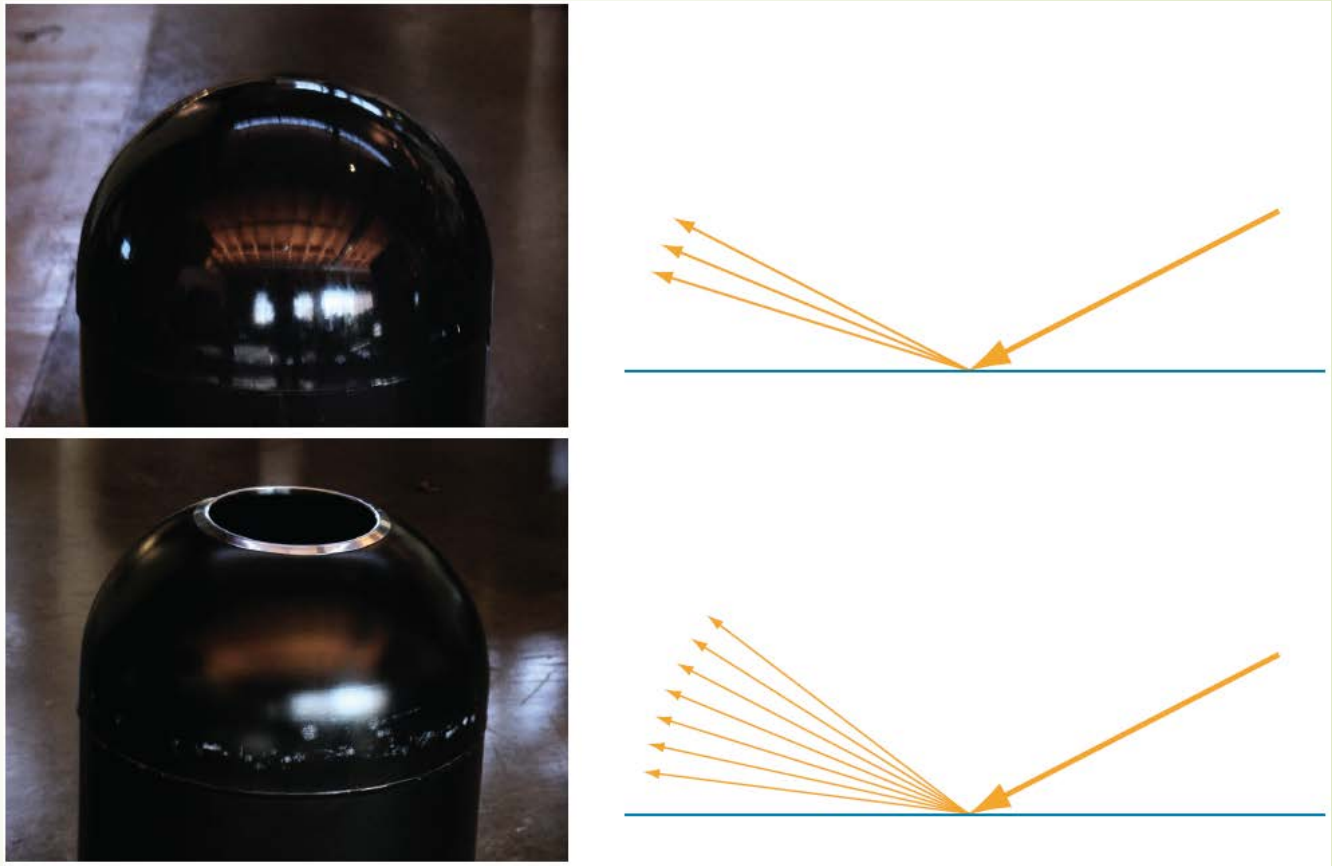
- Diffuse is dull (left)
- Specular: simulates a highlight



Lighting Specular

Material A: smoother
Tight bright highlight

Material A: rougher
Broad dim highlight



Ambient component: $\mathbf{i}_{amb}$

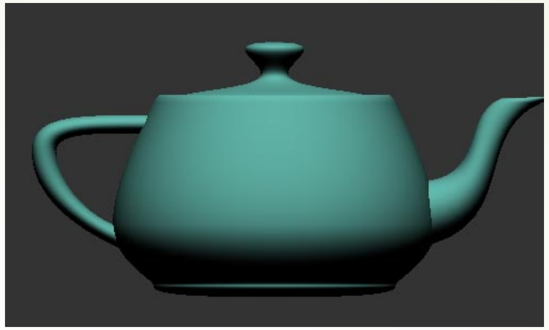
- Ad-hoc – tries to account for light coming from other surfaces
- Just add a constant color:

$$\mathbf{i}_{amb} = \mathbf{m}_{amb} \otimes \mathbf{s}_{amb}$$

Lighting

$i=i_{amb}+i_{diff}+i_{spec}$

+

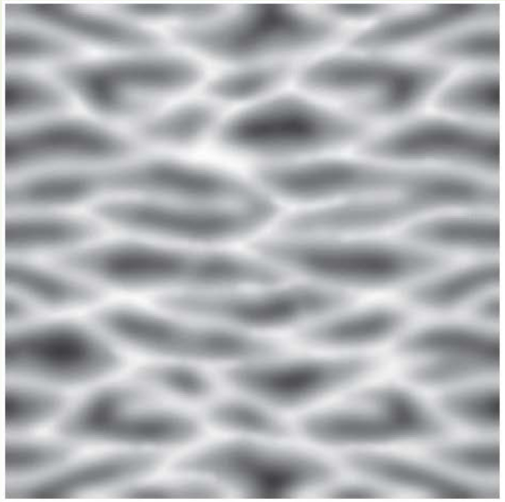


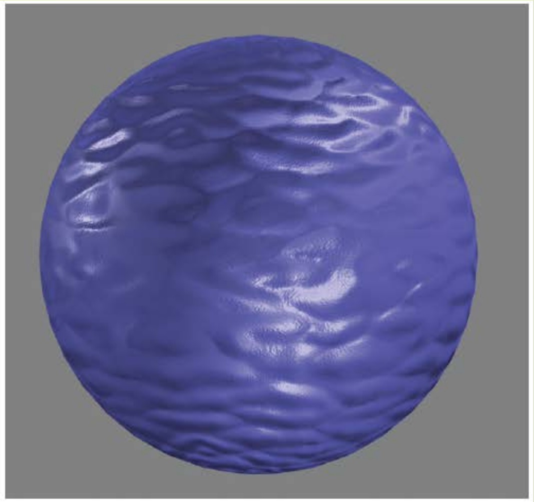
+

=

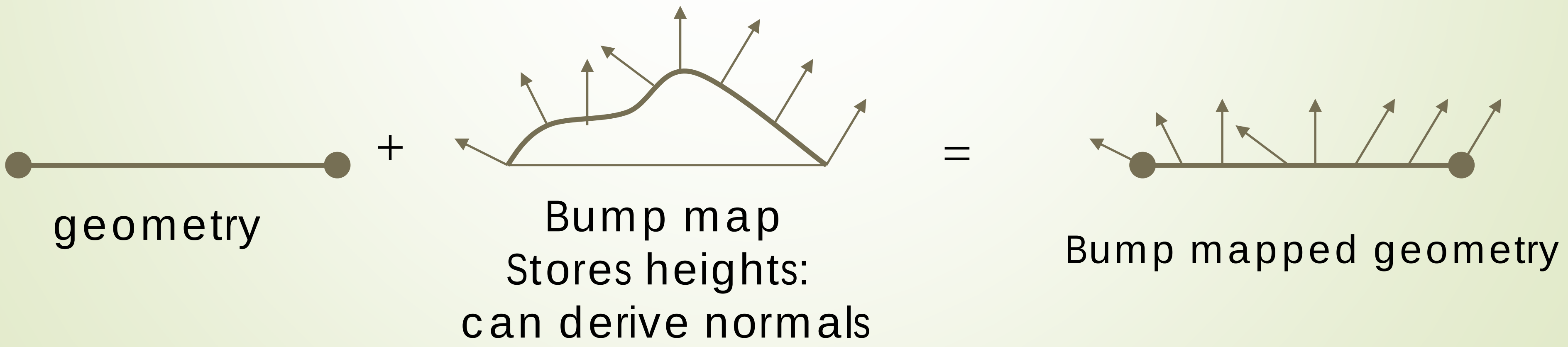
- This is just a hack!
- Has little to do with how reality works!

Bump mapping





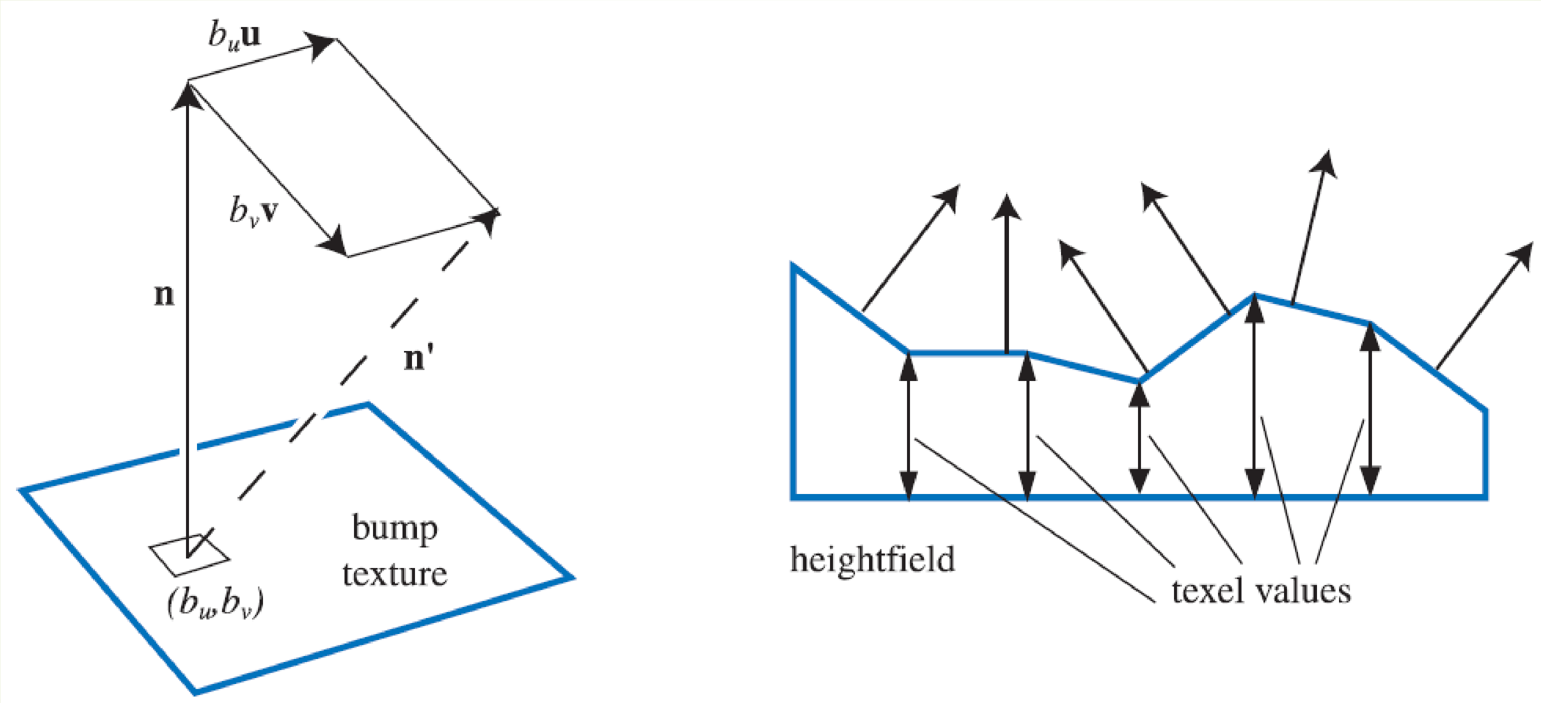
- by Blinn in 1978
- Inexpensive way of simulating wrinkles and bumps on geometry
  - Too expensive to model these geometrically
- Instead let a texture modify the normal at each pixel, and then use this normal to compute lighting



Bump mapping lighting

- Diffuse: $\mathbf{n}\cdot\mathbf{l}$ Specular: $(\mathbf{n}\cdot\mathbf{h})^m$
- Assume directional lights
- Diffuse: fetch per pixel normal from bumpmap
  - Then compute per-pixel dot product with lightvector (constant)
- Specular: h is (assumed) constant for dir. lights
  - compute per-pixel dot product with normal from bumpmap
  - Gives $(\mathbf{n}\cdot\mathbf{h})$, then $(\mathbf{n}\cdot\mathbf{h})^2$, $(\mathbf{n}\cdot\mathbf{h})^4$, $(\mathbf{n}\cdot\mathbf{h})^8$

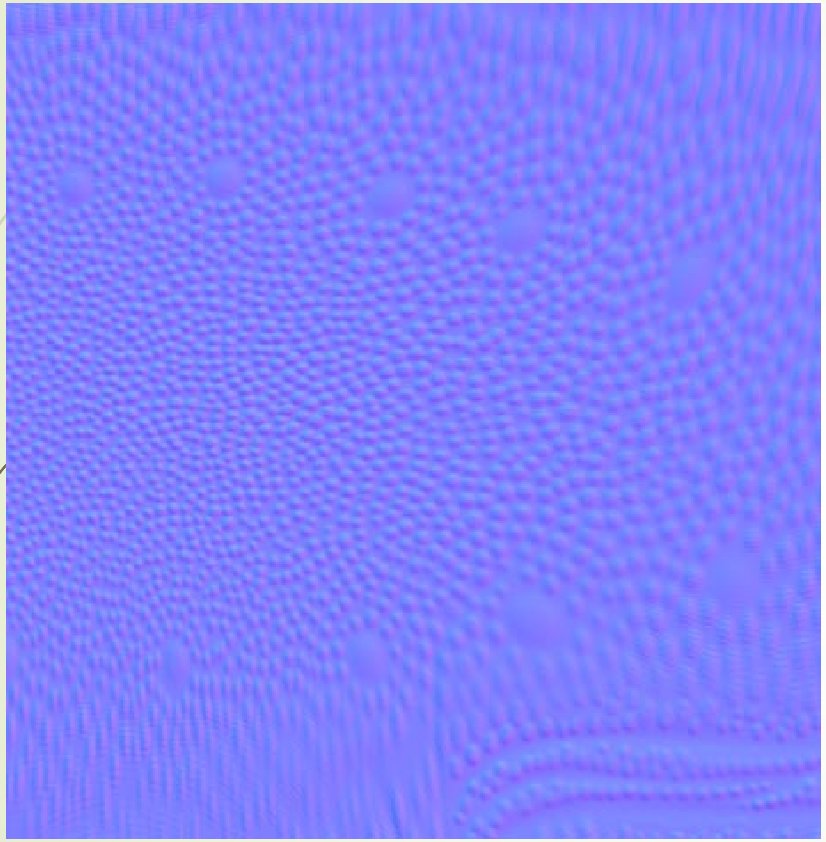
Normal directions in Bump Mapping



Approach #1
Offset vector map
Each pixel stores (b_w, b_v)

Approach #2
Height field, each pixel stores how high the center point is

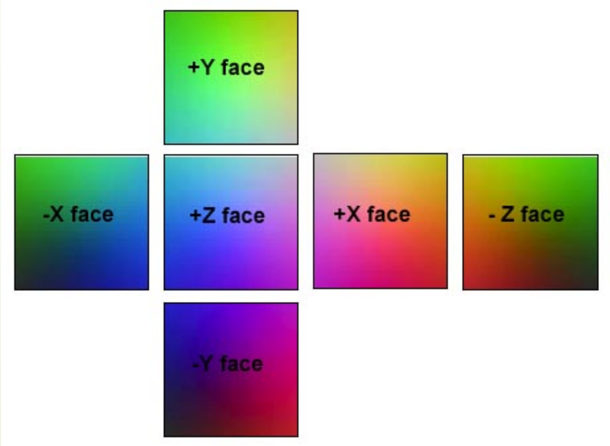
Normal mapping



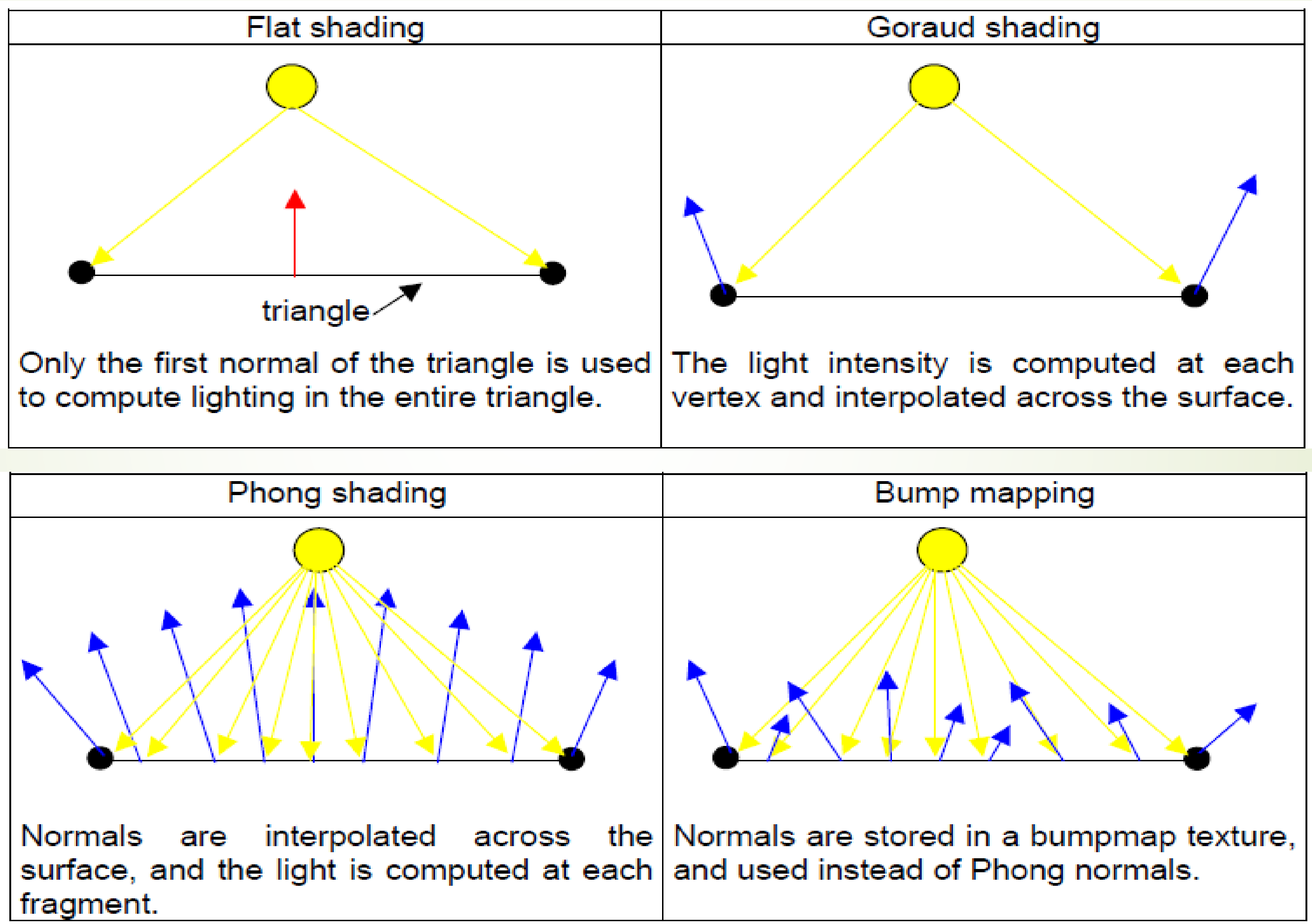
Approach #3
Normal map (faster)
Each pixel stores perturbed direction (x,y,z)

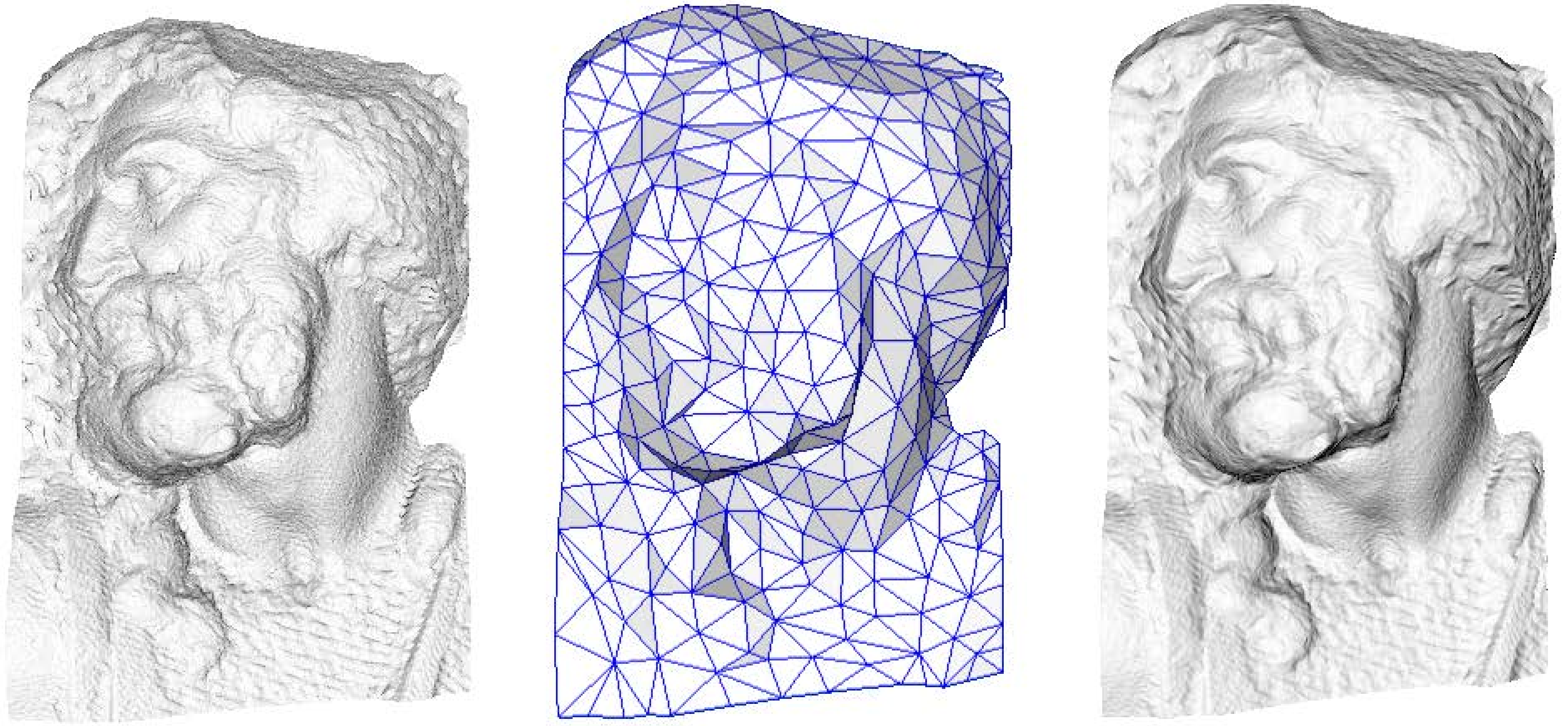
Normal mapping in World Space

- Store normals in texture
  - $n = (n_x, n_y, n_z)$ are in [-1,1]
  - $n = \left(\frac{n_x+1}{2}, \frac{n_y+1}{2}, \frac{n_z+1}{2}\right)$ in [0,1]
  - Mult by 255 (8 bit per color component)
- Usually combine with cubemap texture



Comparing to other Shading Methods





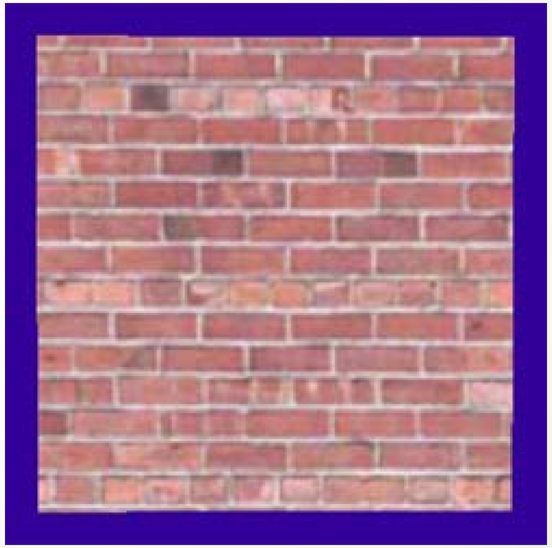
original mesh
4M triangles

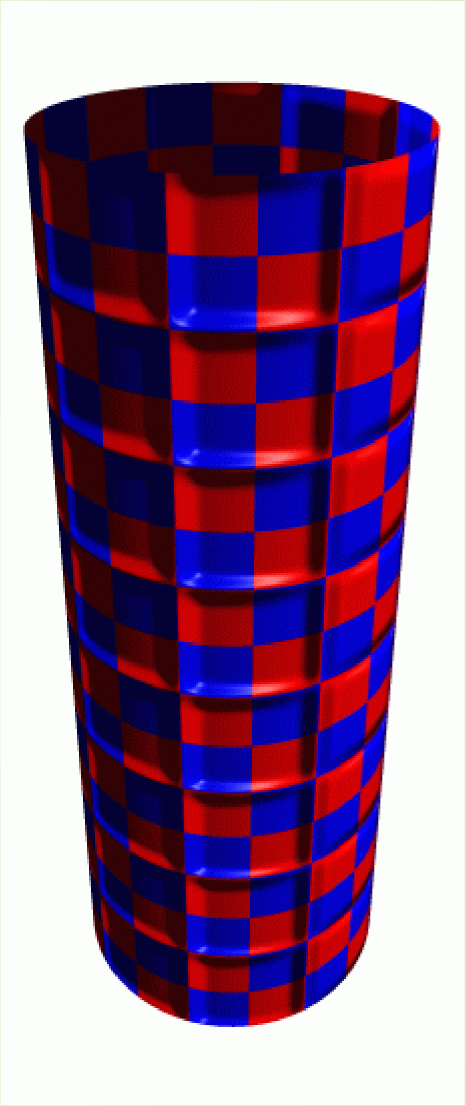
simplified mesh
500 triangles

simplified mesh
and normal mapping
500 triangles

What's Missing?

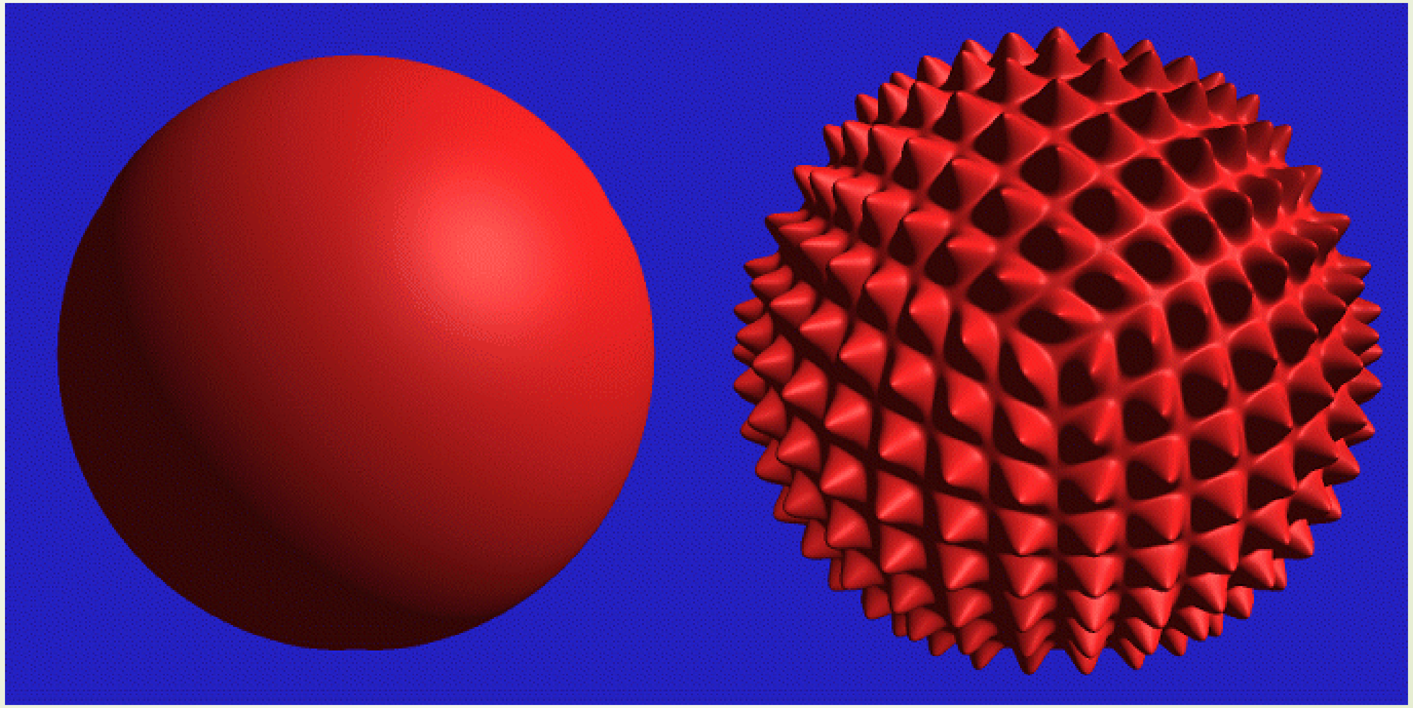
- There are no bumps on the silhouette of a bump-mapped object
- Bump maps don’t allow self-occlusion or self-shadowing





Displacement Mapping

- Use the texture map to actually move the surface point
- The geometry must be displaced before visibility is determined



Displacement Mapping

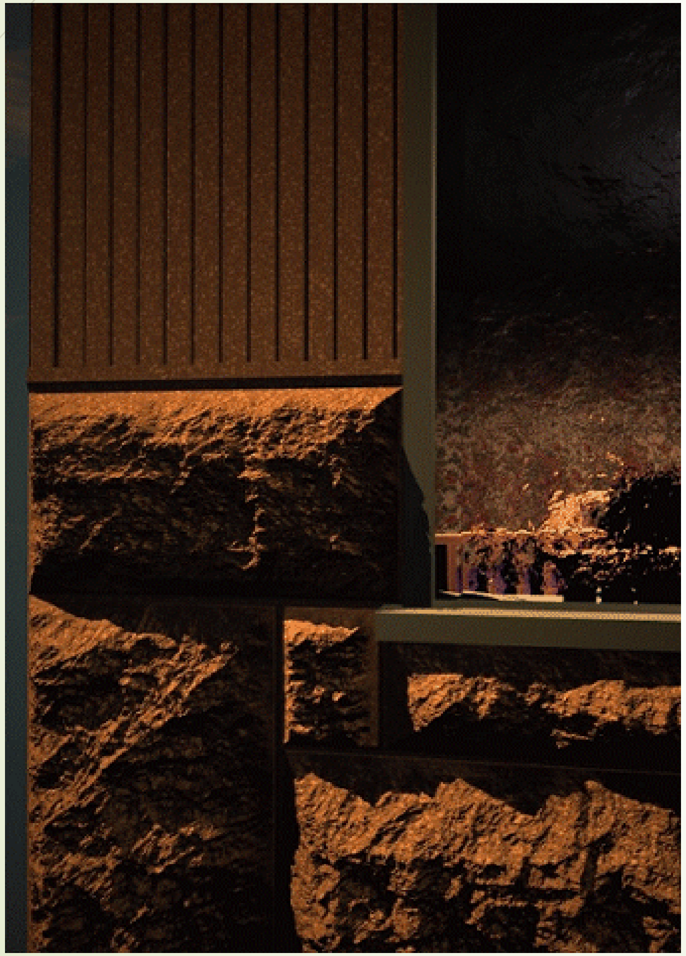


Image from:

Geometry Caching for
Ray-Tracing Displacement Maps

by Matt Pharr and Pat Hanrahan.

note the detailed shadows
cast by the stones

Displacement Mapping

Other

- 3D textures:
  - Feasible on modern hardware as well
  - Texture filtering is no longer trilinear
  - Rather quadlinear (linear interpolation 4 times)
  - Enables new possibilities
    - Can store light in a room, for example
- Multitexturing
  - More than one set of texture coords per vertex
  - The output from the first texture stage is input to the next
  - Opens up for many possibilities

Next Lecture

- Tangent space normal mapping
- Parallax mapping
- Relief mapping
- Some GLSL (OpenGL Shading Language)