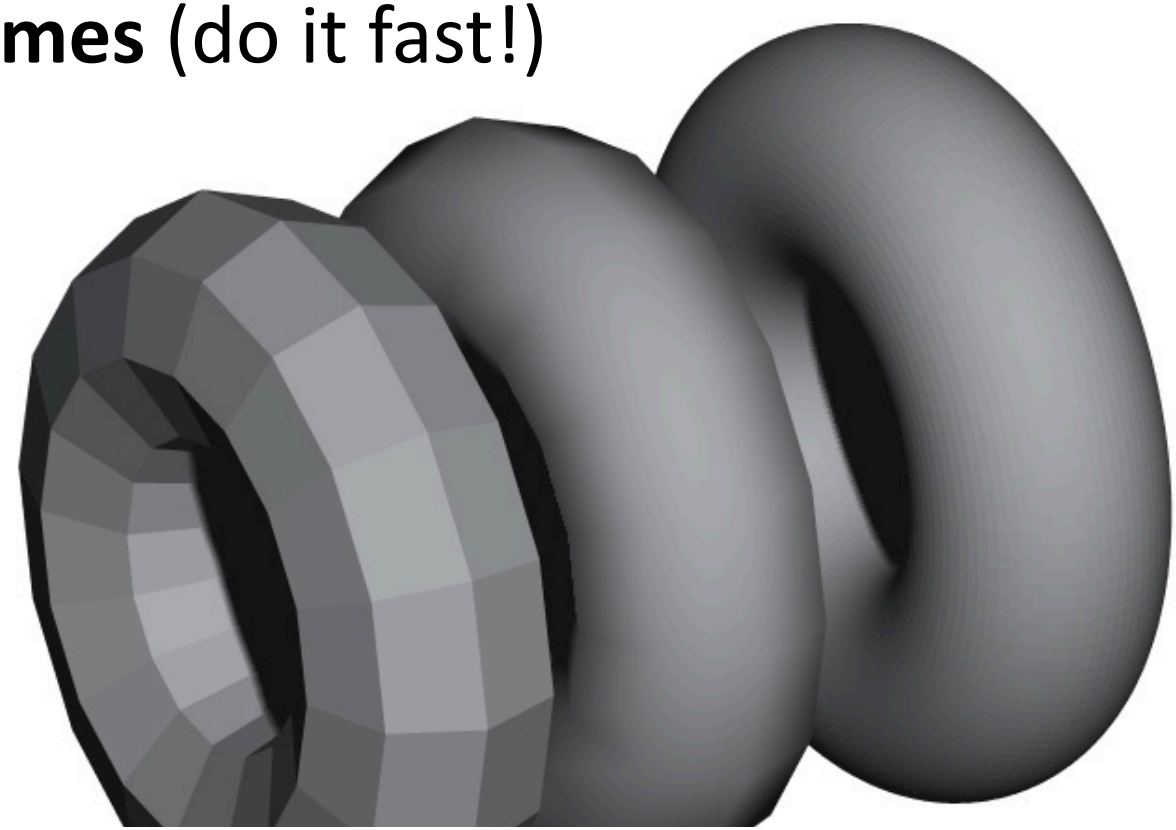
#4: Faux Geometry

Rendering tricks that give the appearance of geometry.

WHY fake it?

- **Algorithmic simplicity** (get it done!)
- **Representational range** (do it all!)
- **Rendering times** (do it fast!)
- **File sizes**



HOW fake it?

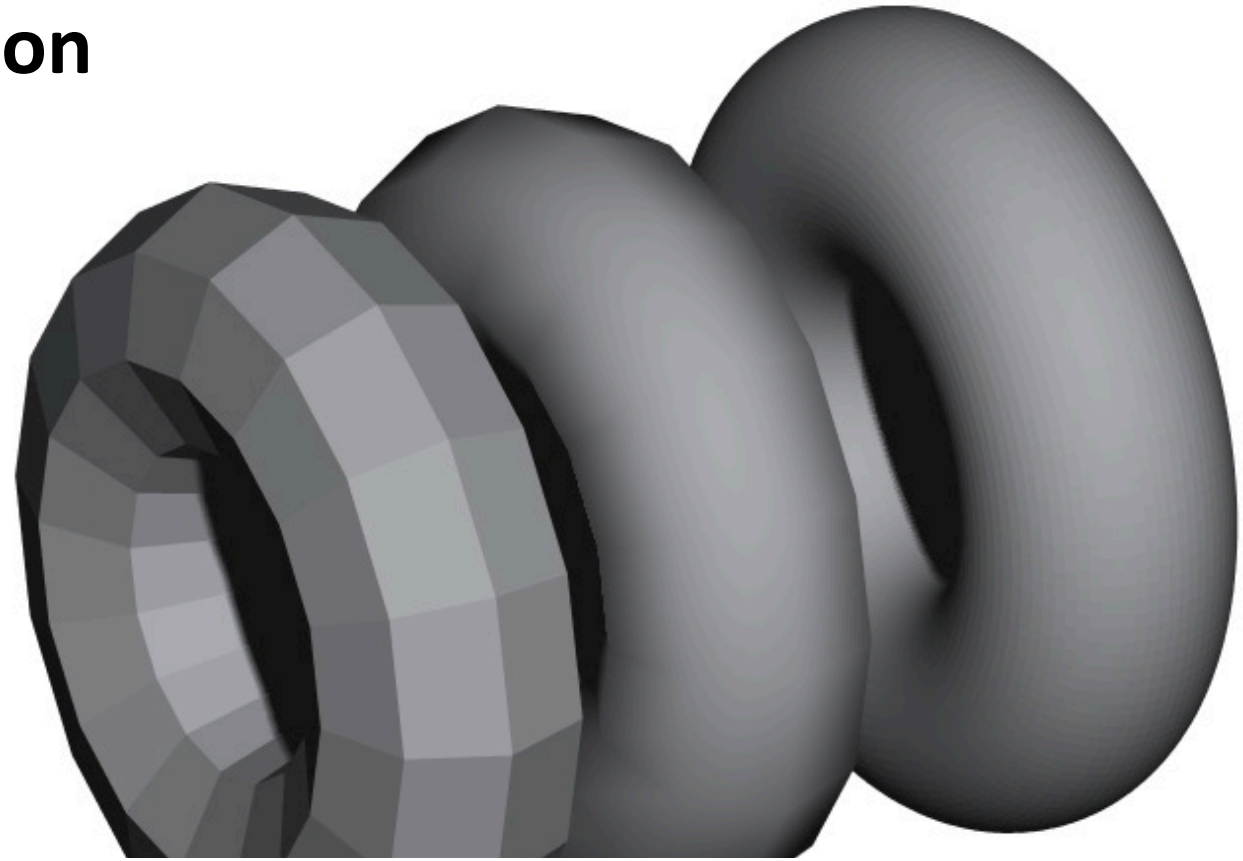
Tweak the rendering equation @ ea pixel:

$$I_r = I_i \bullet k_s \bullet \cos\theta$$

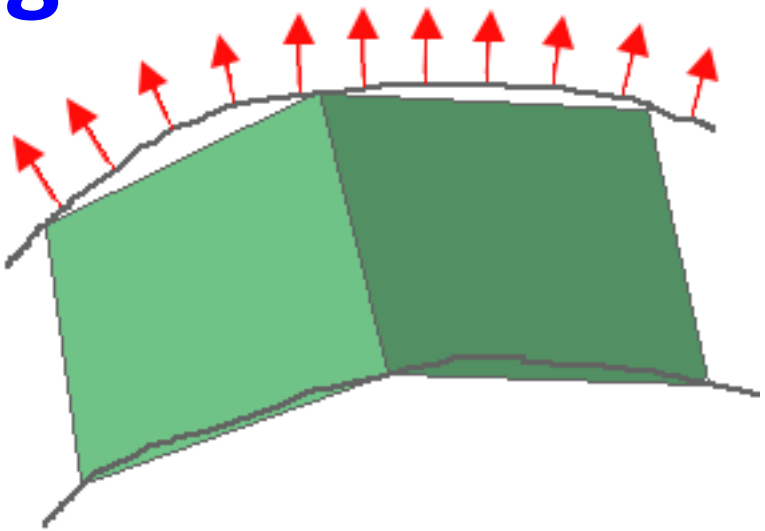
- Smoothing ($\cos\theta$)
- Color (k_s)
- Transparency (k_s)
- Irregularities (θ)

Smooth shading

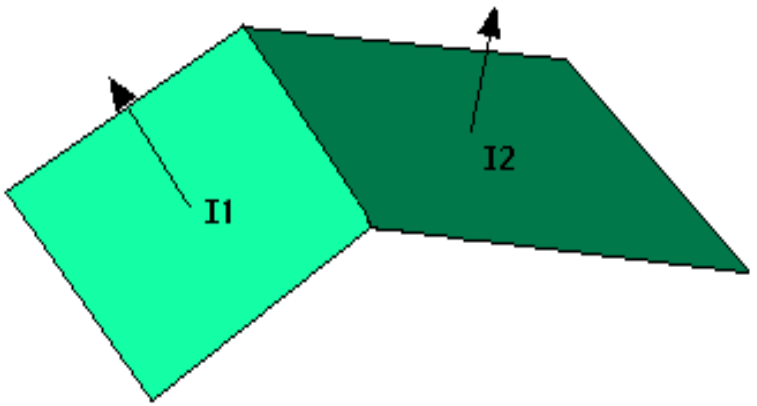
- **Making 128 polygons *do the work of* 11,680**
- **Leverages z-buffer hidden-surface computation**



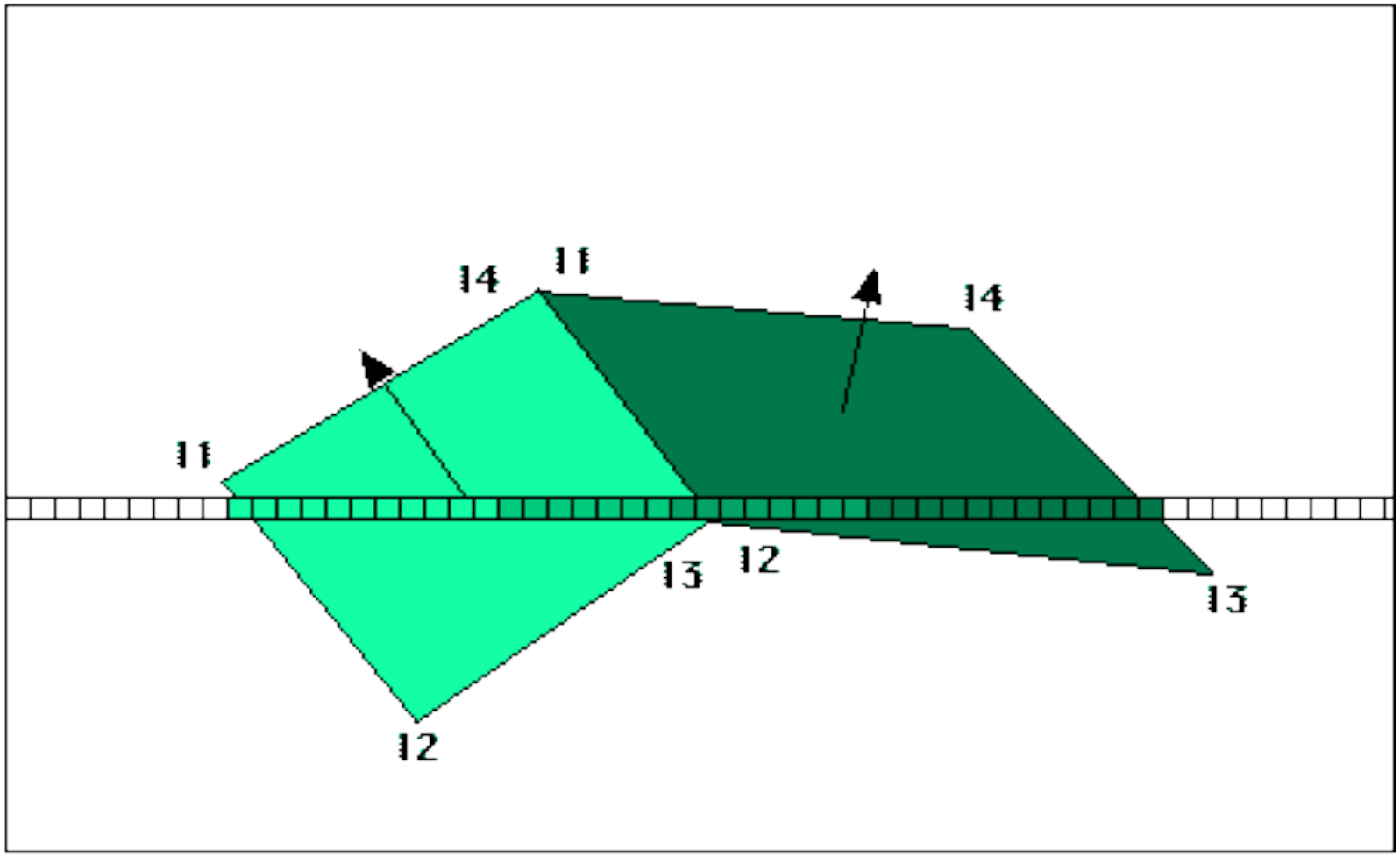
Smooth shading



- **Tessellation insight**
  - Polygons are approximate
  - Normals can vary!
  - How to calculate?
- **Gouraud said…** ("forget the normals")
  - Calculate the *intensity* ($I_i \bullet \cos\theta$) at each corner
  - Average adjacent corners
  - Interpolate along edges

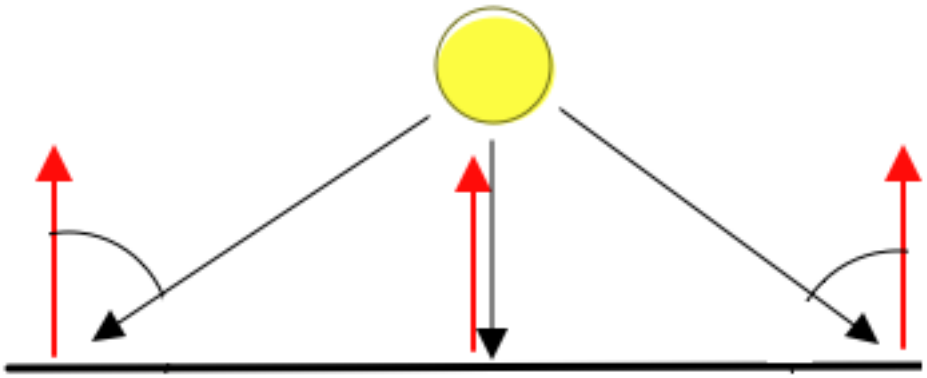


Smooth shading



Smooth shading

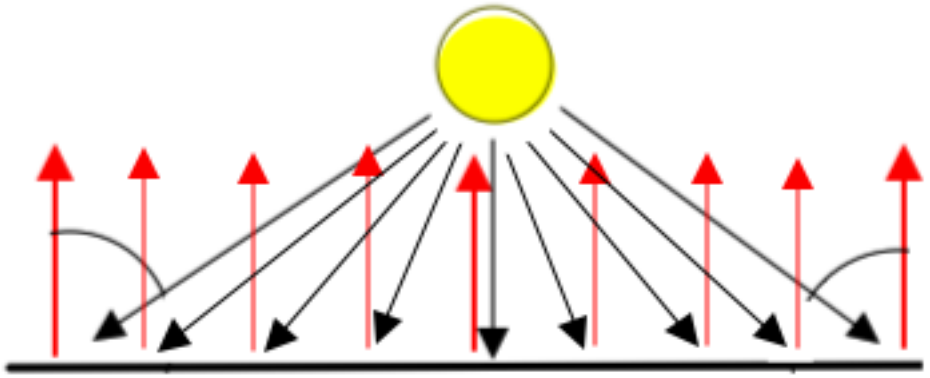
- **The problem with Gouraud is …**
- With in-scene lights, there should be hot spots



Smooth shading

So Phong says …

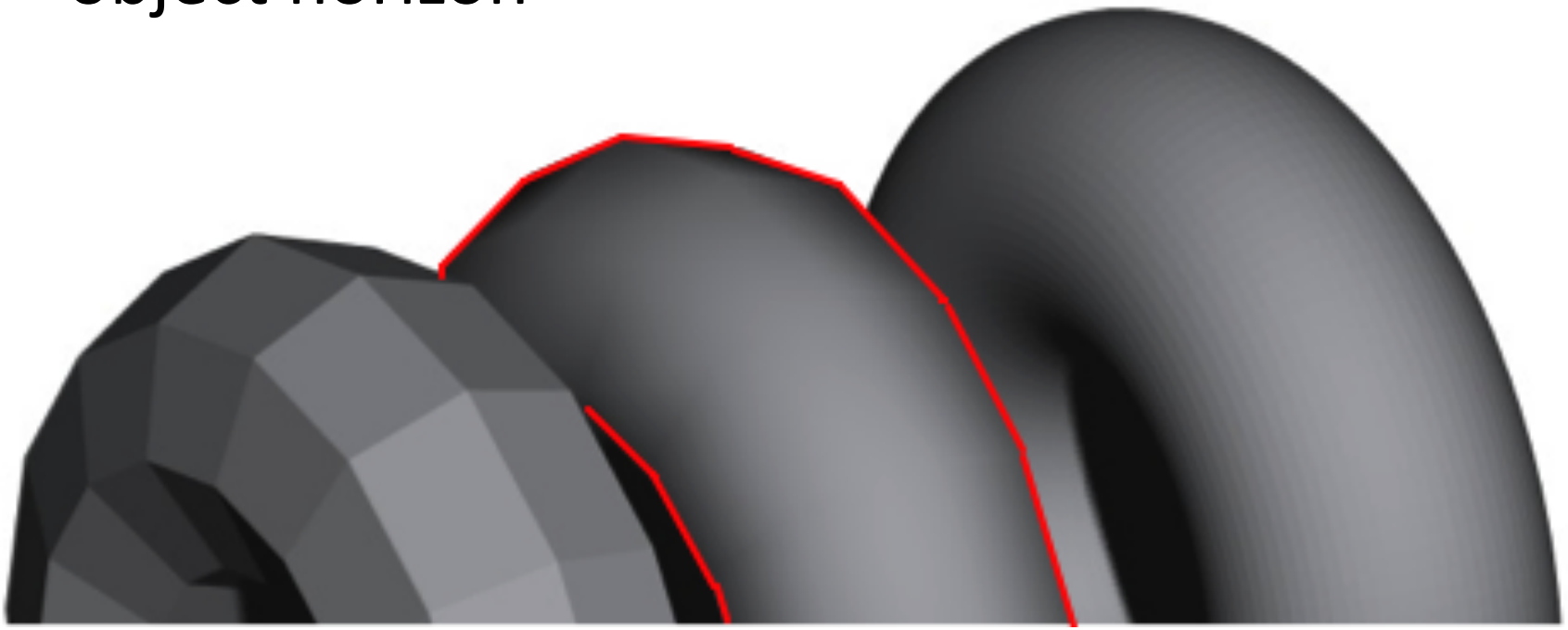
- Compute an **average normal** at each vertex,
- Interpolate the normals and compute intensity at each pixel (3x the work, better result)



Smooth shading

Nobody's perfect …

- Phong smoothing still shows polygons at the object horizon



Surface Coloration: Textures

- **If we're coloring pixels one by one anyway, maybe other 'tweaks' would be useful?**

$$I_r = I_i \bullet k_s \bullet \cos\theta$$

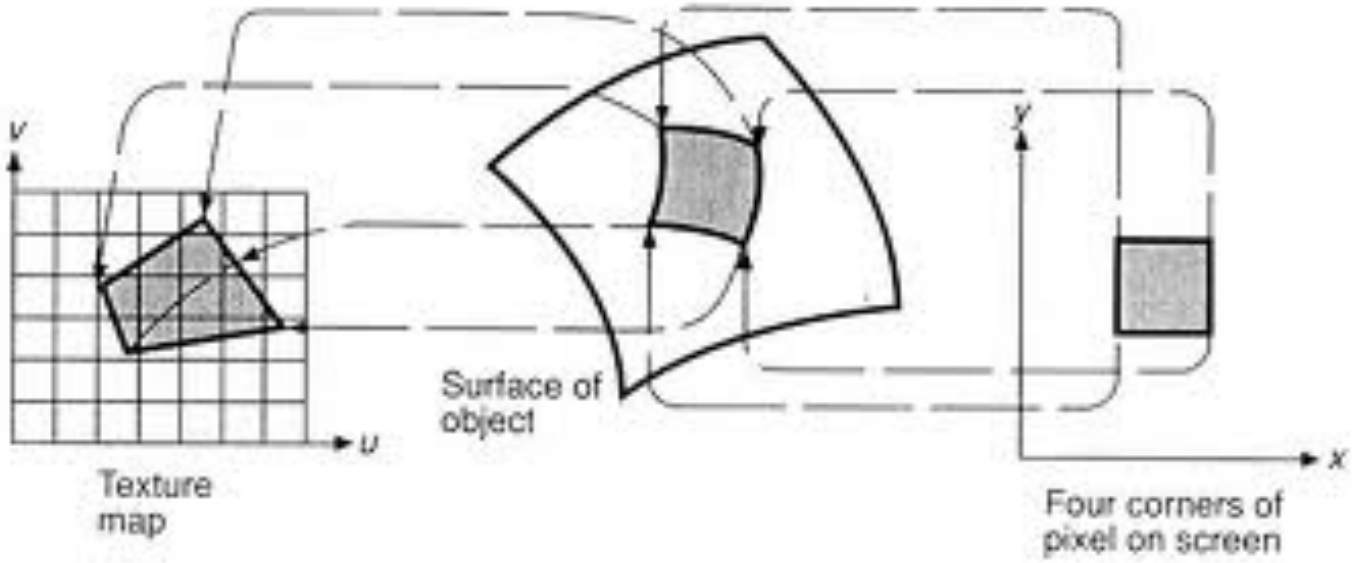
Let's change the color (k_s)!

- **How to *represent* the color variation?**
  - **Compute a color? (like we did with smoothing)**
  - **Look up the color in a "database"**

Surface Coloration: Textures

Compute a color ("procedural" or "solid")

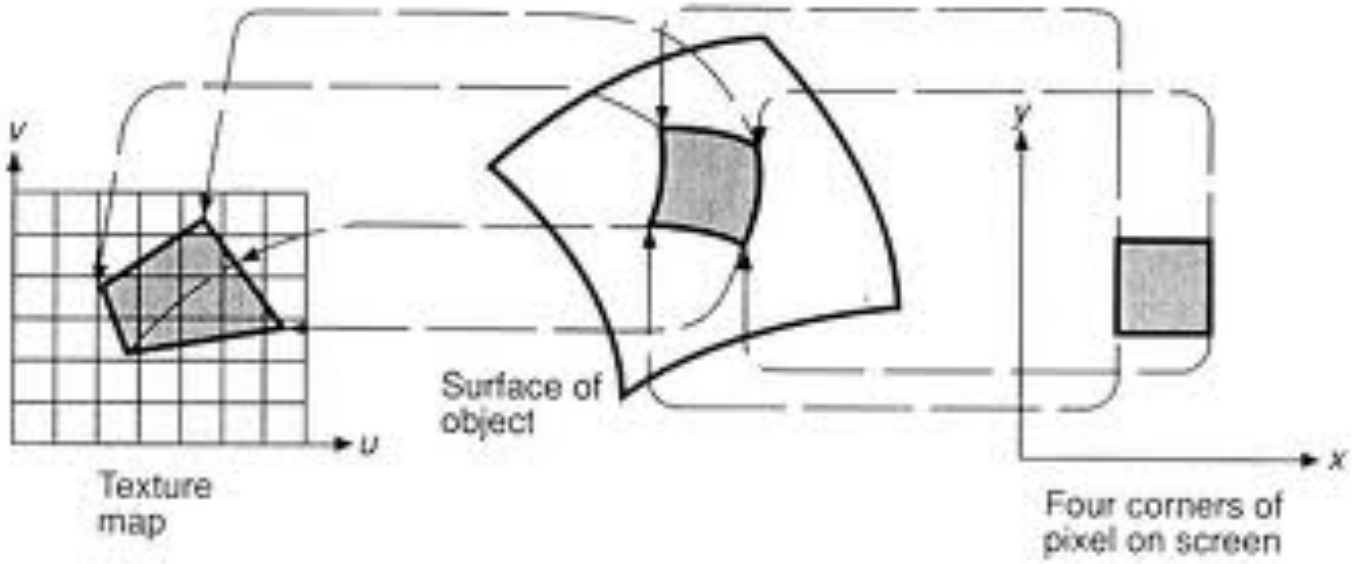
- **Trace the pixel backwards; build an equation?**
- **"UV-coordinates" 2D position ON the surface**
- **Can take surface orientation into account**
- **Challenge: what's the function for "granite" ?**



Surface Coloration: Textures

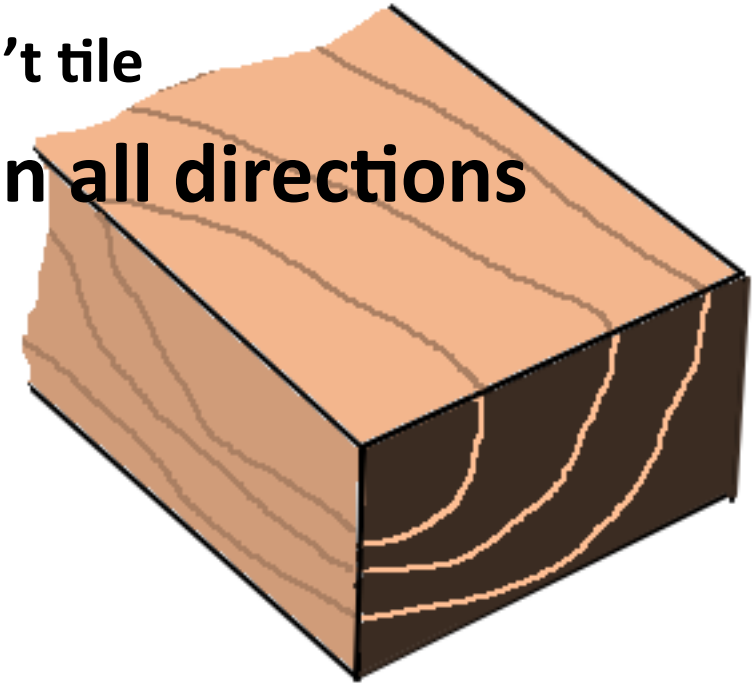
Look up a color

- **Existing color databases?** ***raster images*****!**
- **Us UV-coordinates to compute row & column**
- **Challenge: anisotropic materials**



Surface Coloration: Textures

- How does color vary?
  - In patterns (tile, wallpaper, concrete formwork, etc)
    - Manufactured patterns usually “tile”
  - Without pattern (stone, wood, wear, dirt, graffiti)
    - Natural patterns usually don’t tile
- Anisotropic: Not uniform in all directions
  - Solid wood v. Wood-veneer
  - Wood grain carries over to rings



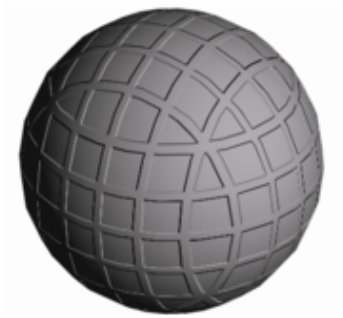
Surface Coloration: Textures

Compare & Contrast

Procedural (computed)	Image-based (look-up)
Computationally intensive	RAM intensive
No repeats	Repeats
Variable resolution	Fixed resolution
Solid	Veneer
Hard to make	Easy to make

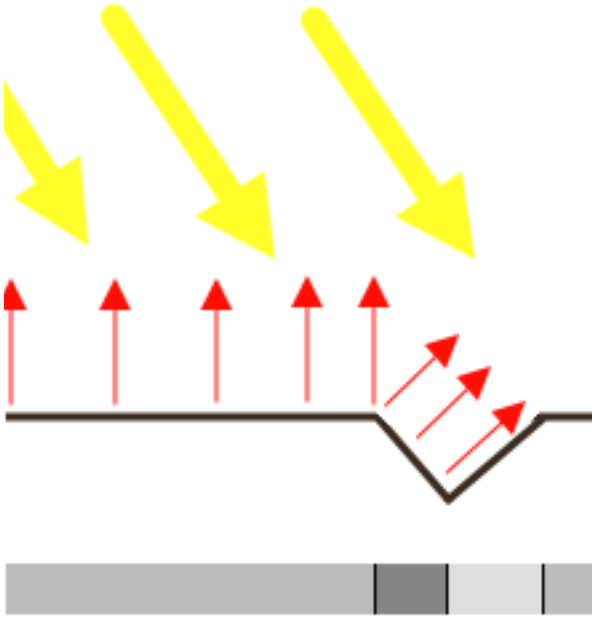
Surface Transparency

- Adjustment to z-buffer behavior
- Check pixel transparency as well as z-location
- Combine surface colors for final appearance
- "Zero polygon geometry"
  - Wire mesh
  - Deck rails & handrails on stairs
  - Distant openings in walls, etc.
- Uses image-map alpha channel or B/W image



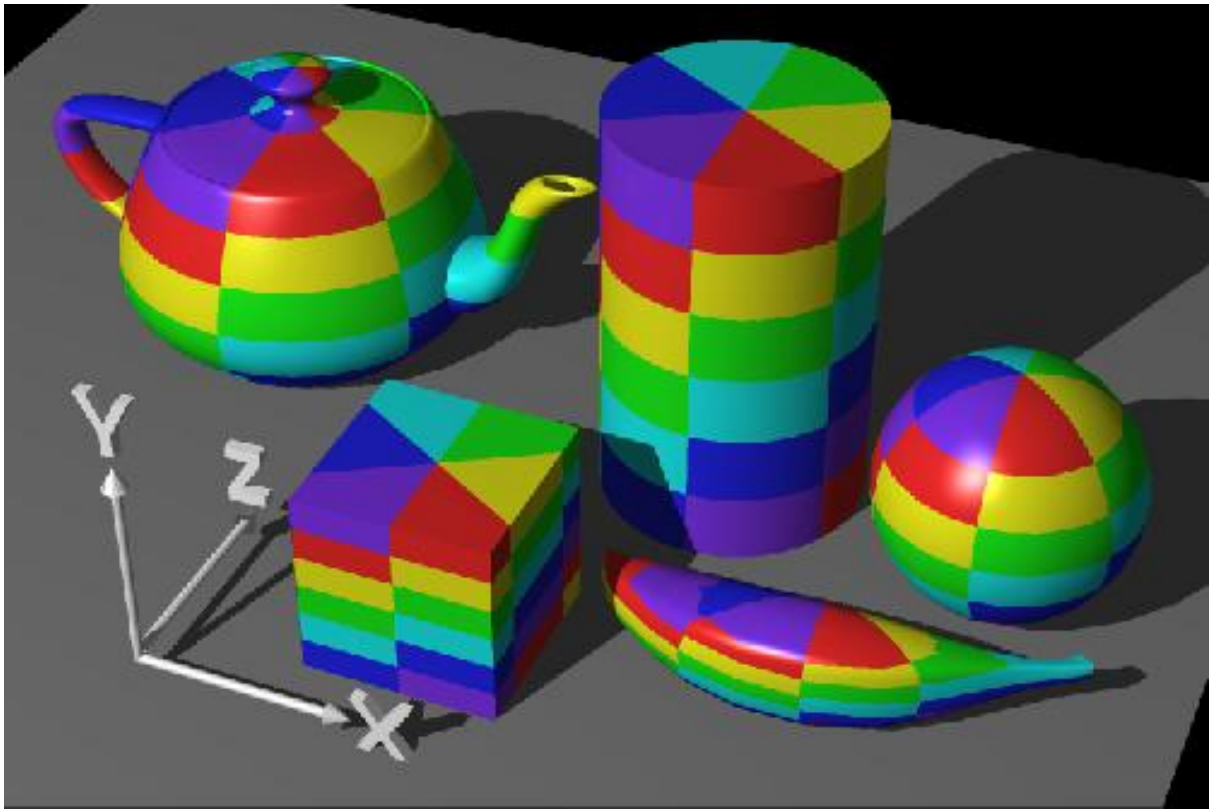
Surface Irregularities: Bumps

- Simulates small scale bumps
  - Tile grout lines
  - Brick mortar lines
  - Steel tread plate
  - Concrete form lines & form ties
- No horizon disturbance
- Image-based: gray-scale image controls



Texture-Surface Alignment

- Mapping/projection type (planar, square, cylindrical, spherical)
- Scale
- Origin
- Orientation
- N-repeats
- Mirroring



Textures as Many Maps

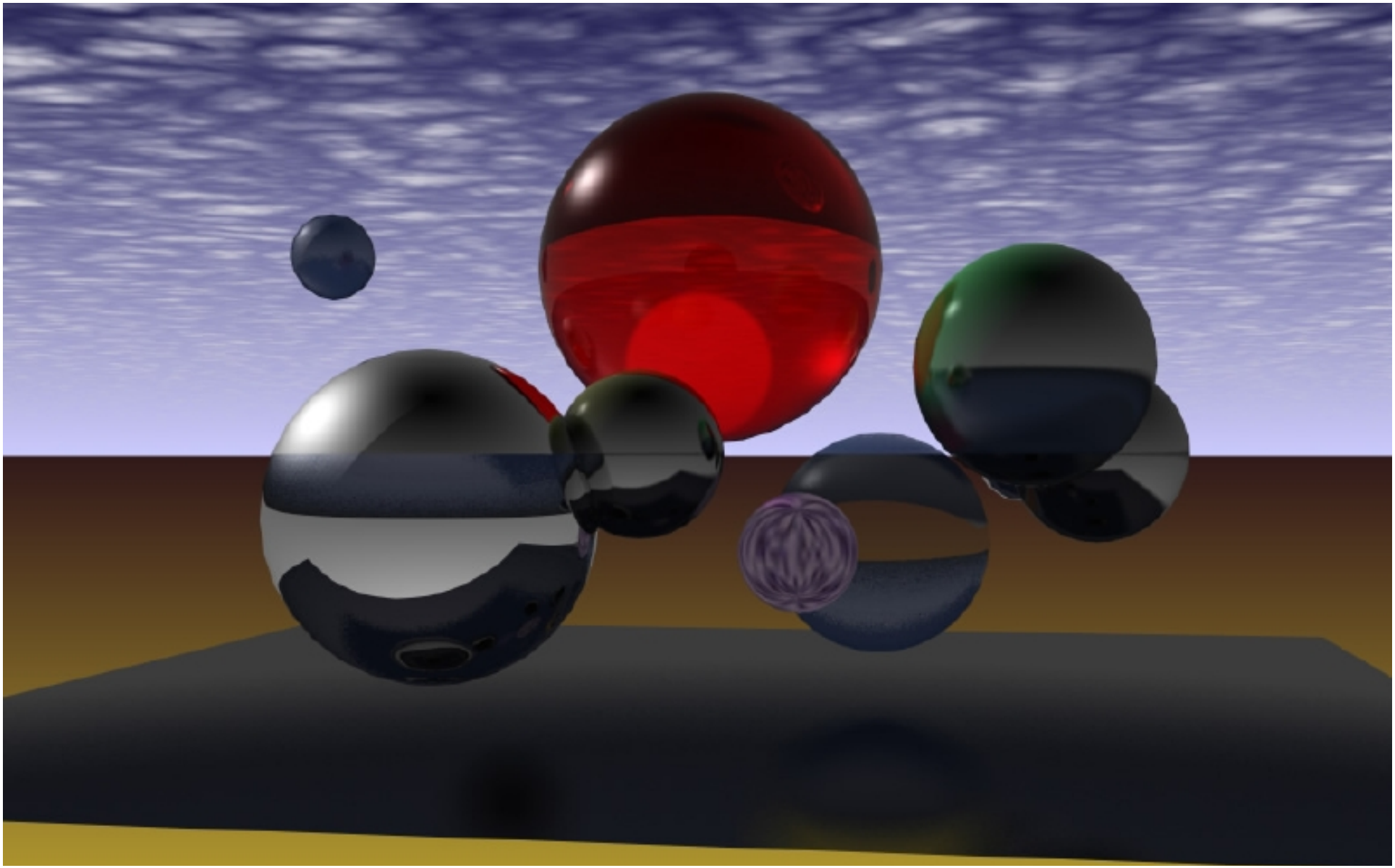
- Same resolution (same # pixels)
- There may be multiple map opportunities
  - Environment (what's reflected)
  - Diffuse color vs. specular color
  - Transparency
  - Bump
  - Displacement
- Know what they do!

Example: Transparency Handrails

Example: One-poly Mike

Example: Controlling the Smooth

Example: Ground; Sky; Reflection



Example: Ground; Sky; Trees

Example: Floor; Carpet; Stone

Example: Floor; Concrete

#4: Faux Geometry

Rendering tricks that give the appearance of geometry.