Texturing

Course web page:

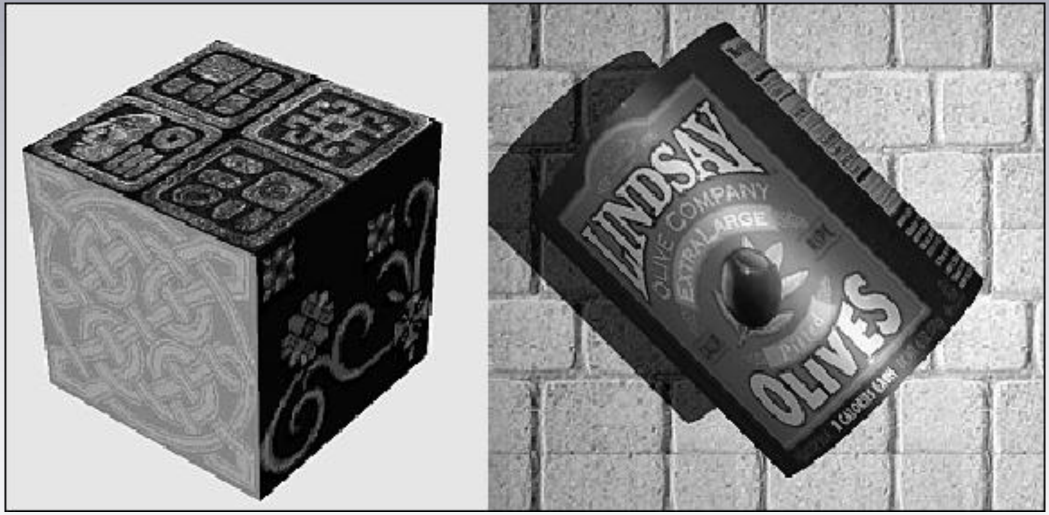
http://goo.gl/EB3aA

Outline

- Bump, displacement mapping
- Texturing pipeline

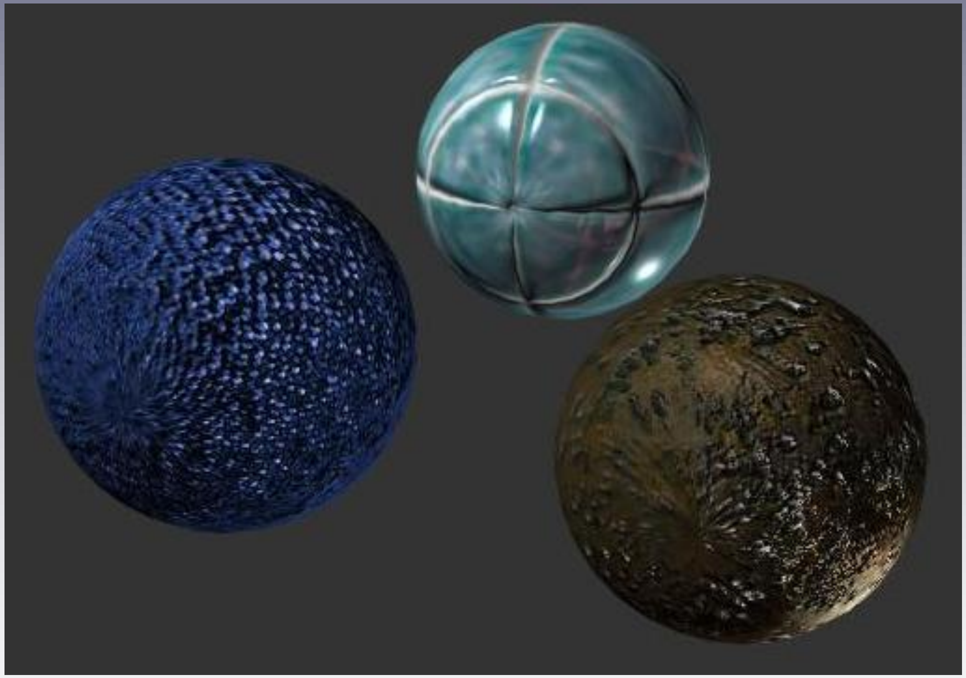
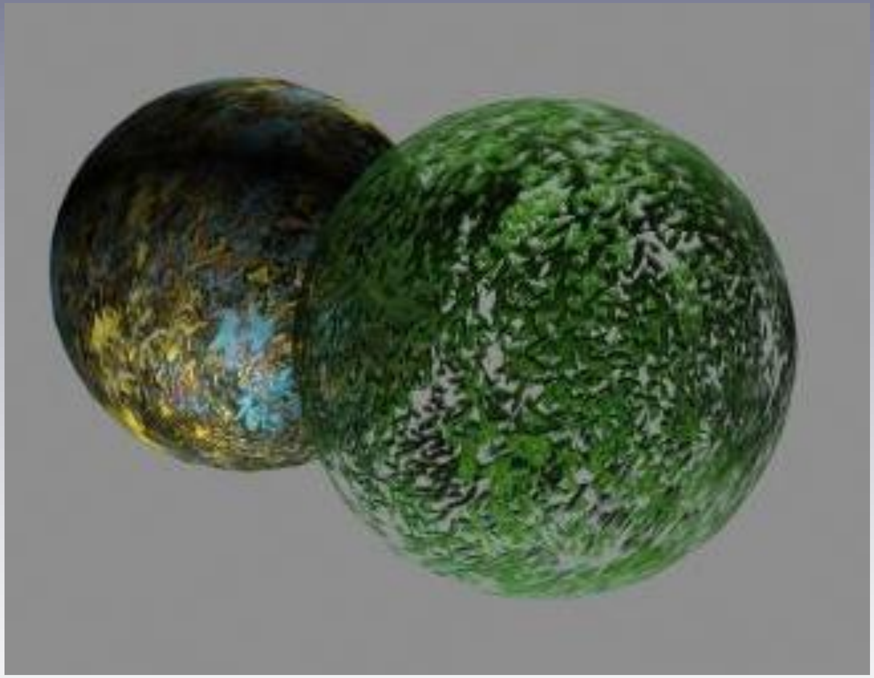
What is Texture Mapping?

- Spatially-varying modification of surface appearance at the pixel level
- Characteristics
  - Color
  - Shininess
  - Transparency
  - Bumpiness
  - Etc.



from Hill

Texture mapping: Examples

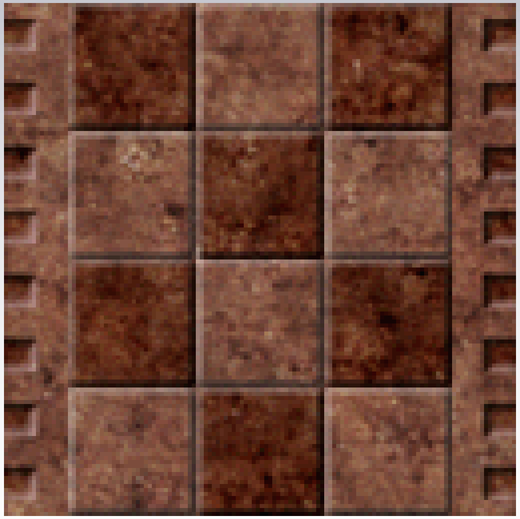
Examples of modulating color, bumpiness, shininess, transparency
with identical sphere geometry

Why use texture mapping?

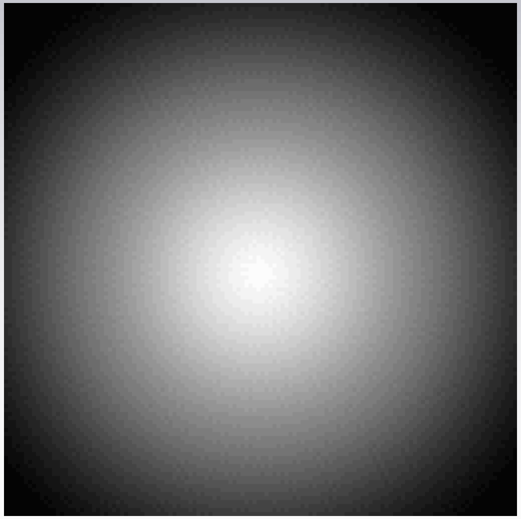
- More detail without the cost of more complicated geometry
  - Modeling
  - Display
- Layer multiple texture maps $\rightarrow$ Can modulate color **+ other** surface characteristics
- "Lookup table" for precomputed quantities
  - Lighting
  - Shadows
  - Etc.

Texture mapping application: Lightmaps

- Can't compute global illumination on the fly for interactive applications, but can precompute and turn into a texture maps



+



=

courtesy of K. Miller

Texture mapping application: Lightmaps

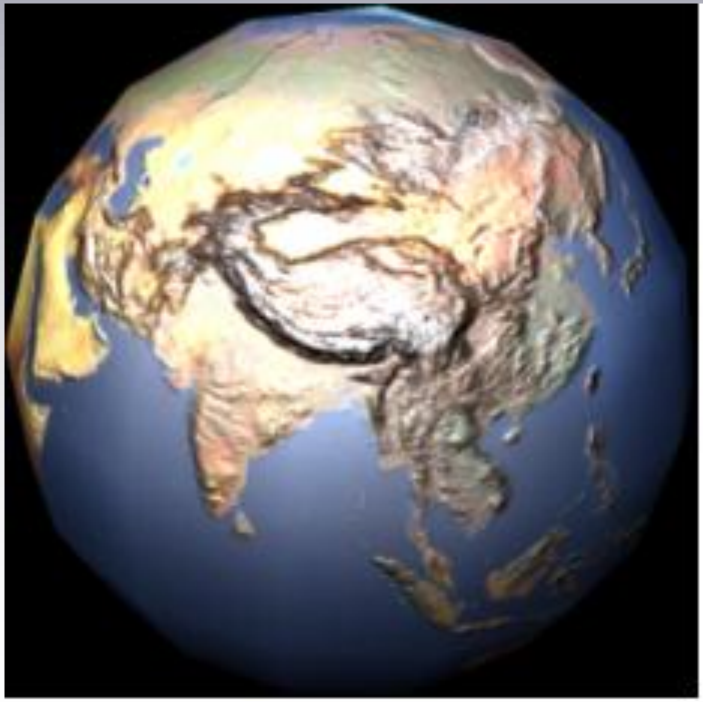
Tenebrae Quake screenshot

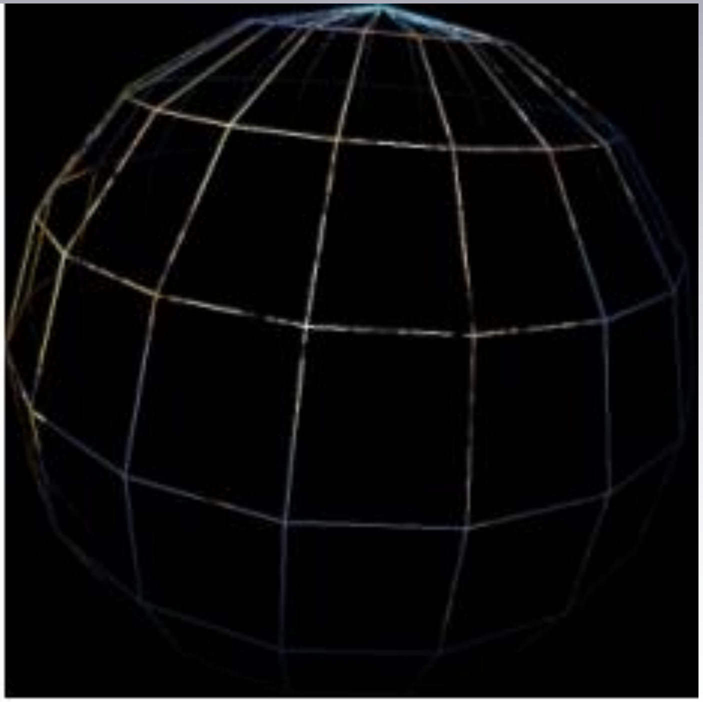
Bump Mapping

- So far we've been thinking of textures modulating color and transparency only
  - Billboards, decals, lightmaps, etc.
- But any other per-pixel properties are fair game...
- Pixel **normals** usually smoothly varying
  - Computed at vertices for Gouraud shading; color interpolated
  - Interpolated from vertices for Phong shading
- Textures allow setting per-pixel normal with a **bump map**

Bump mapping: Why?

- Can get a lot more surface detail without expense of more object vertices to light, transform

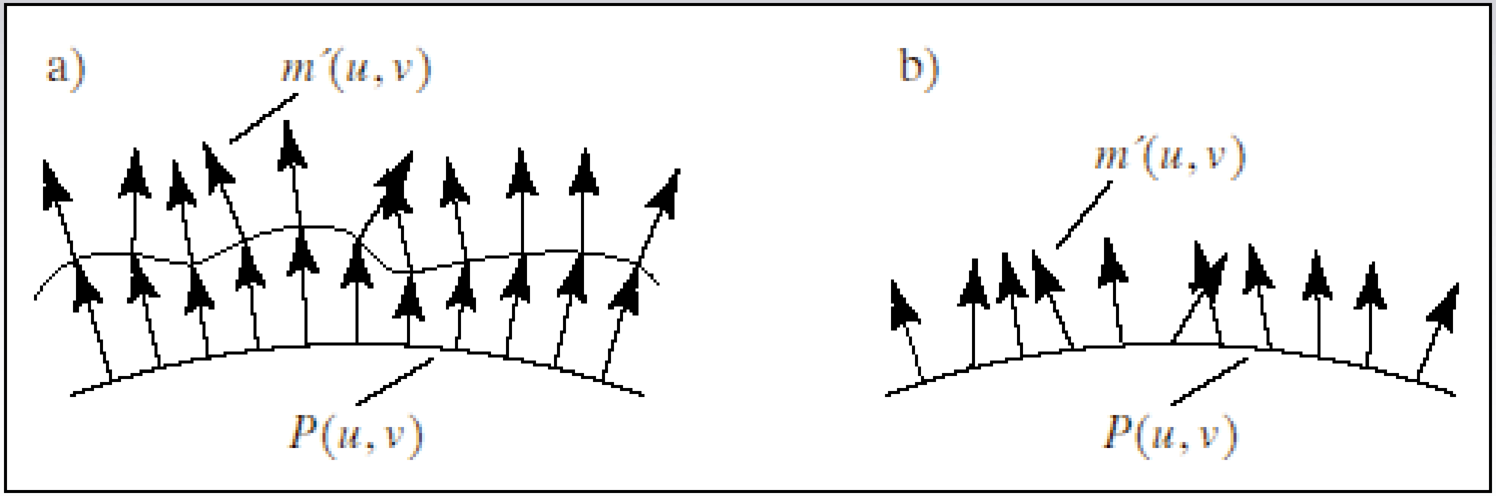




courtesy of Nvidia

Bump Mapping: How?

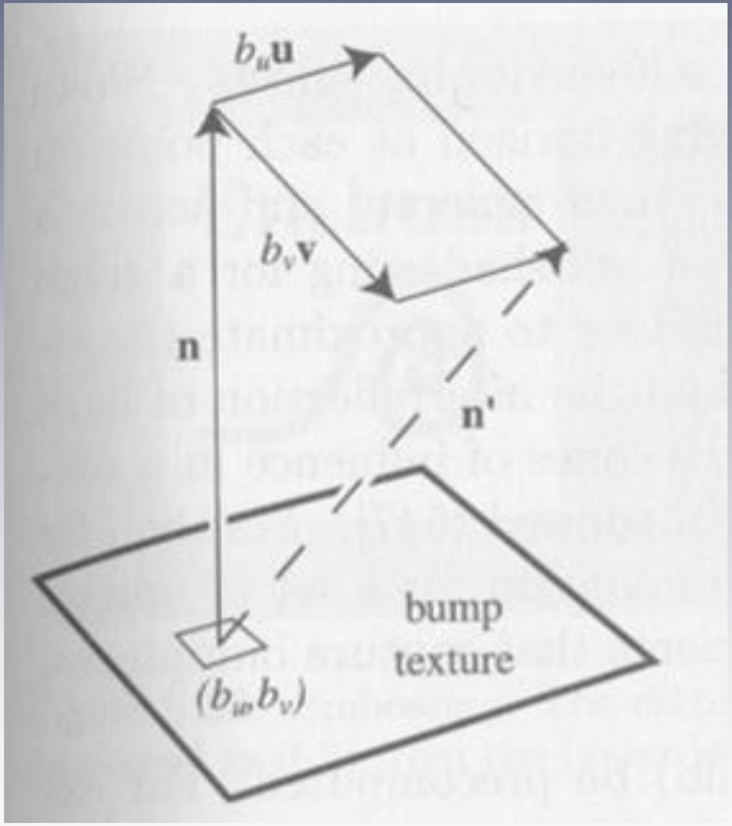
- Idea: Perturb pixel normals **n**(u, v) derived from object geometry to get additional detail for shading
- Compute lighting per pixel (like Phong)



from Hill

Bump mapping: Representations

- 3-D vector **m**(u, v) added directly to normal **n**
- Or: 2-D vector of coefficients (b_u, b_v) that scale **u**, **v** vectors tangent to surface



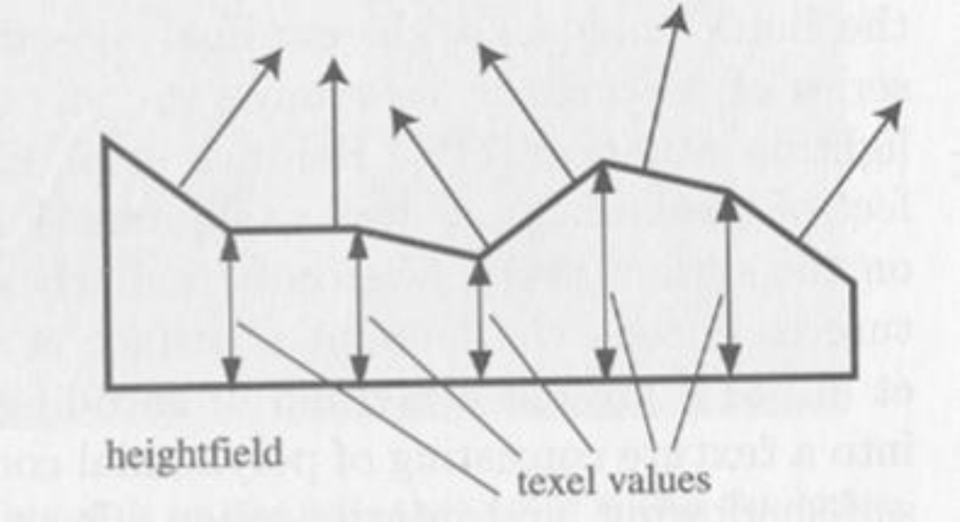
from Akenine-Moller & Haines

Bump representation: Height map f(u, v)

- Store just scalar "altitude" at each pixel
- Get b_u, b_v from partial derivatives:

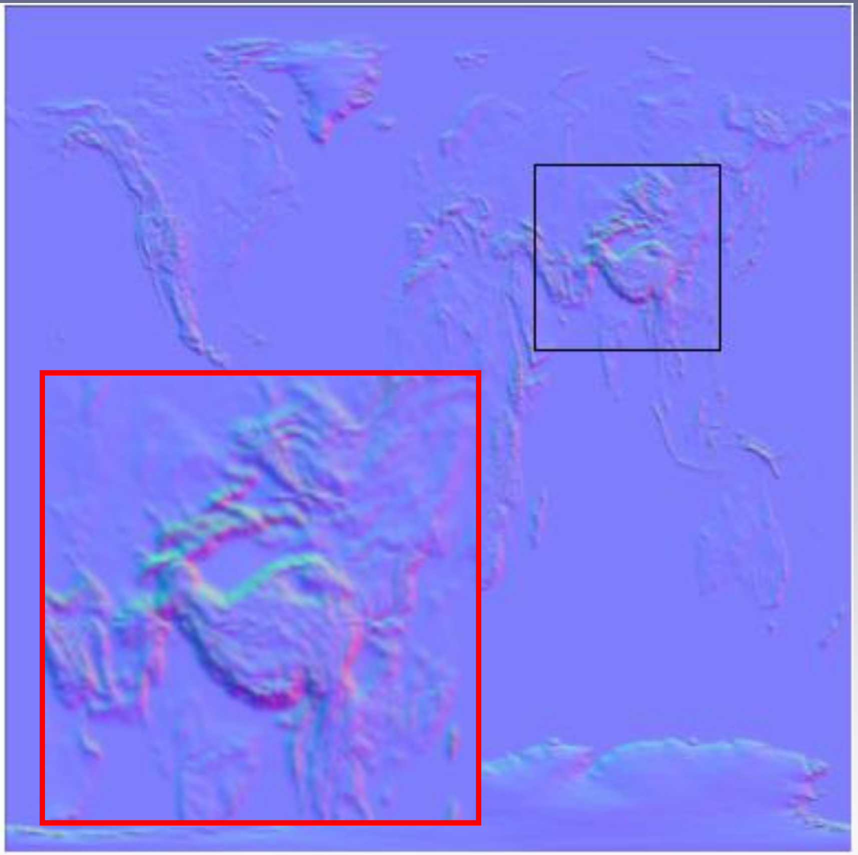
$$b_u = \frac{\partial f}{\partial u} \qquad b_v = \frac{\partial f}{\partial v}$$

  – Approximate with finite differencing



from Akenine-Moller & Haines

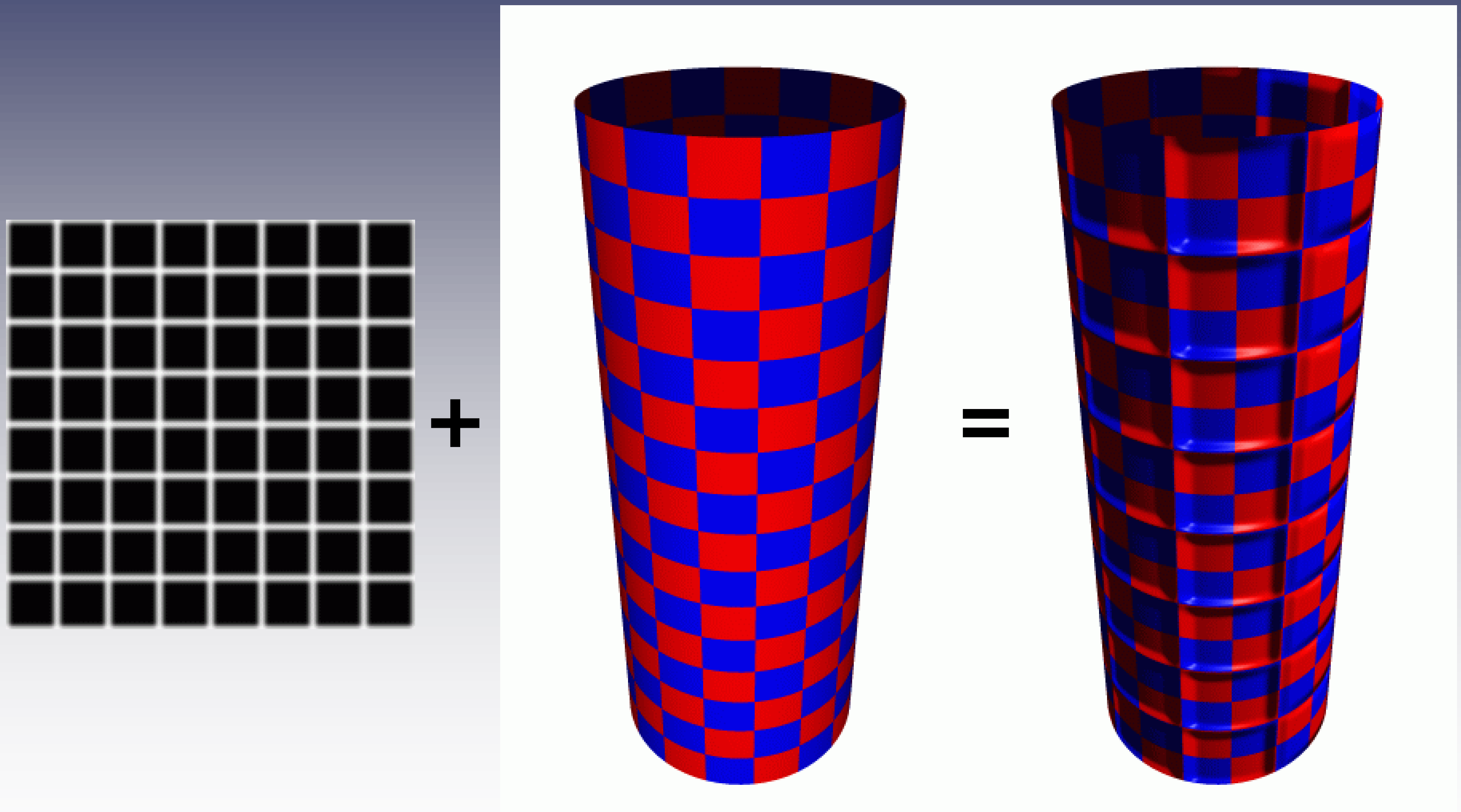
Example: Converting height maps to normal displacements



courtesy of Nvidia

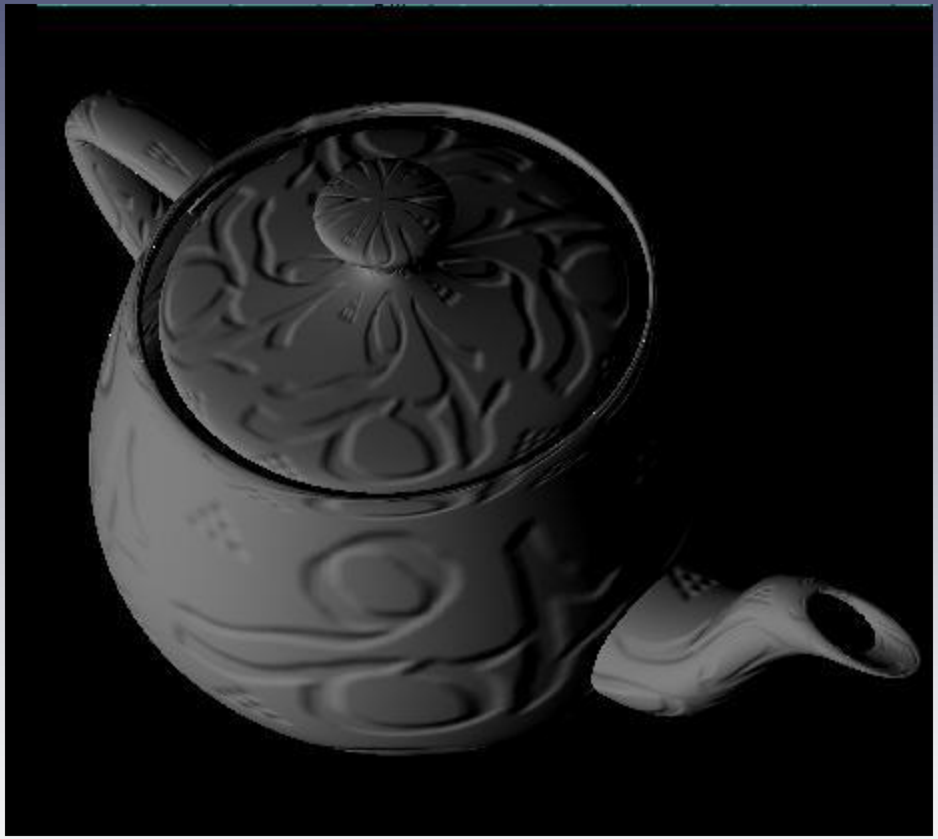
Z coordinate set to some constant scale factor; (X, Y) normalized to [0, 1] range.
Right image is mostly blue because “straight up” vector is (0.5, 0.5, 1)

Bump mapping: Example



from MIT CG lecture notes

Bump mapping: Example

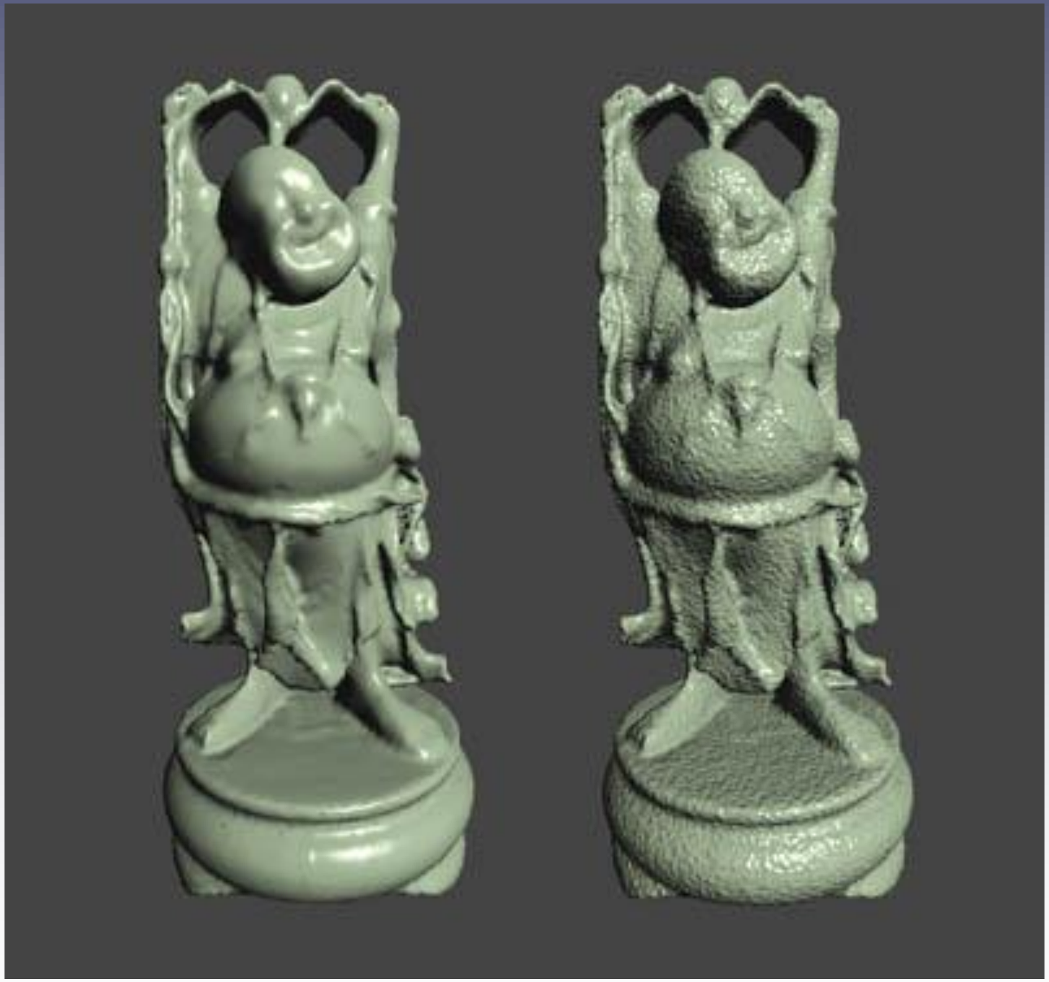


courtesy of A. Awadallah

Height map

Bump texture applied to teapot

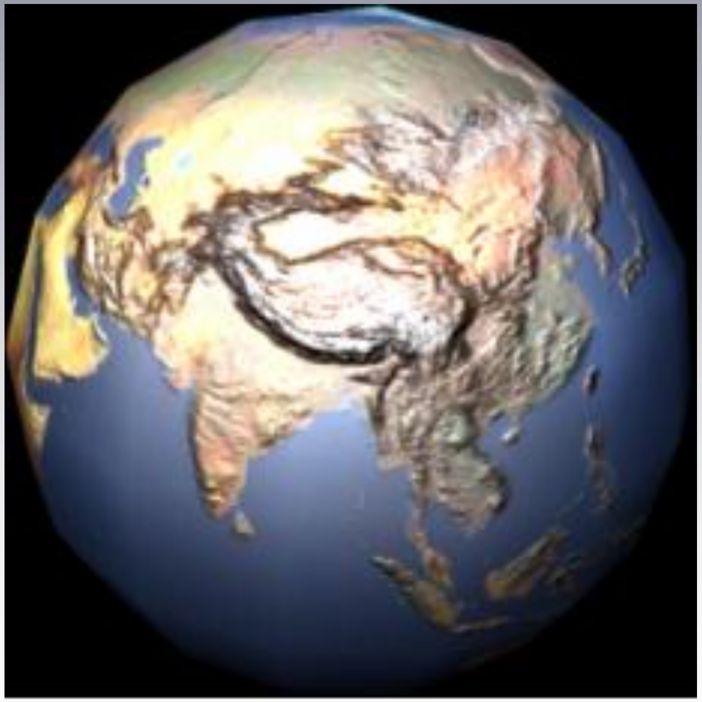
Procedural Bump Mapping



courtesy of Nvidia

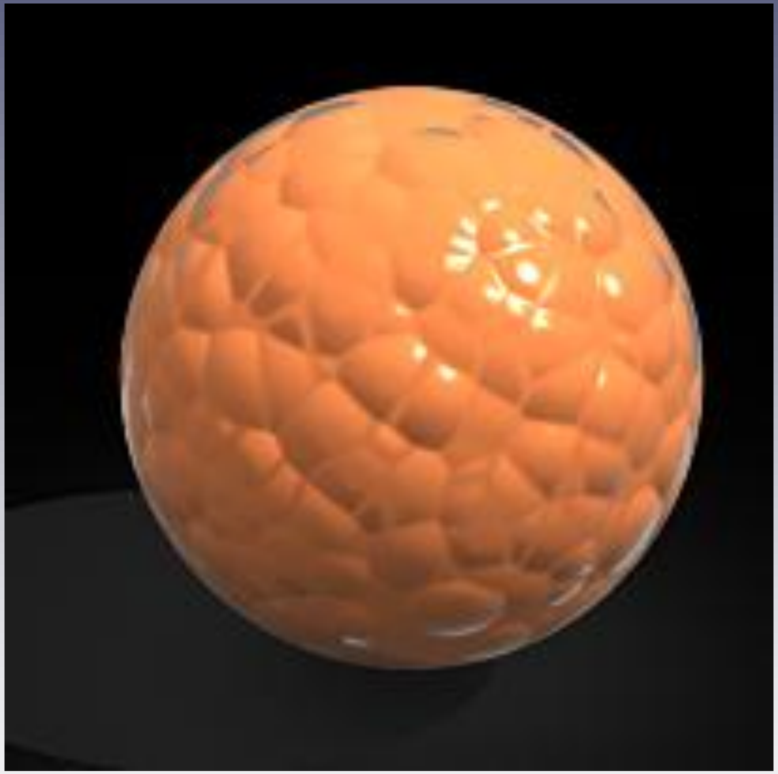
Bump mapping: Issues

- Bumps don’t cast shadows
- Geometry doesn’t change, so silhouette of object is unaffected
- Textures can be used modify underlying geometry with **displacement maps**
  - Generally in direction of surface normal

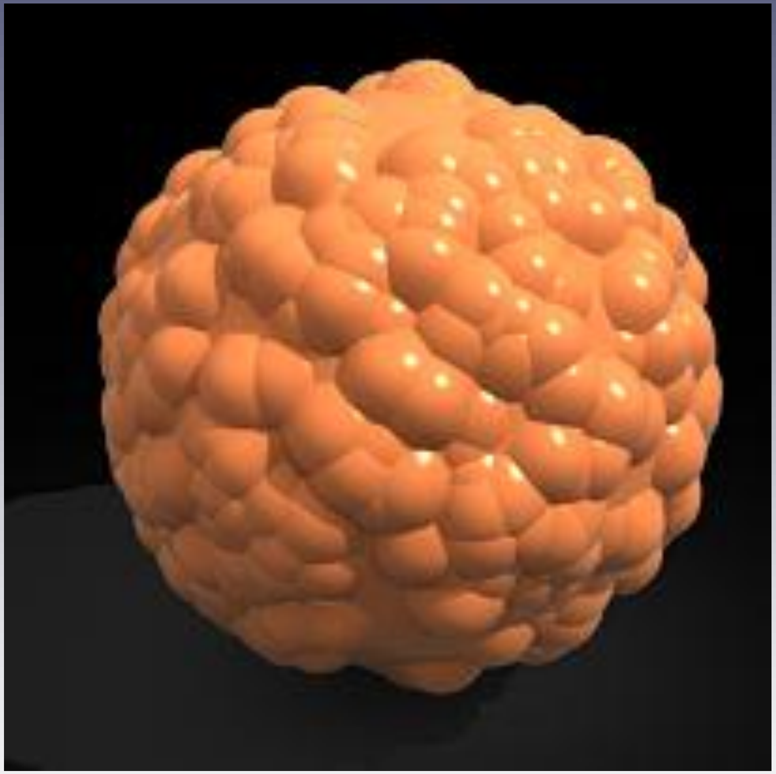


courtesy of Nvidia

Displacement Mapping



Bump mapping

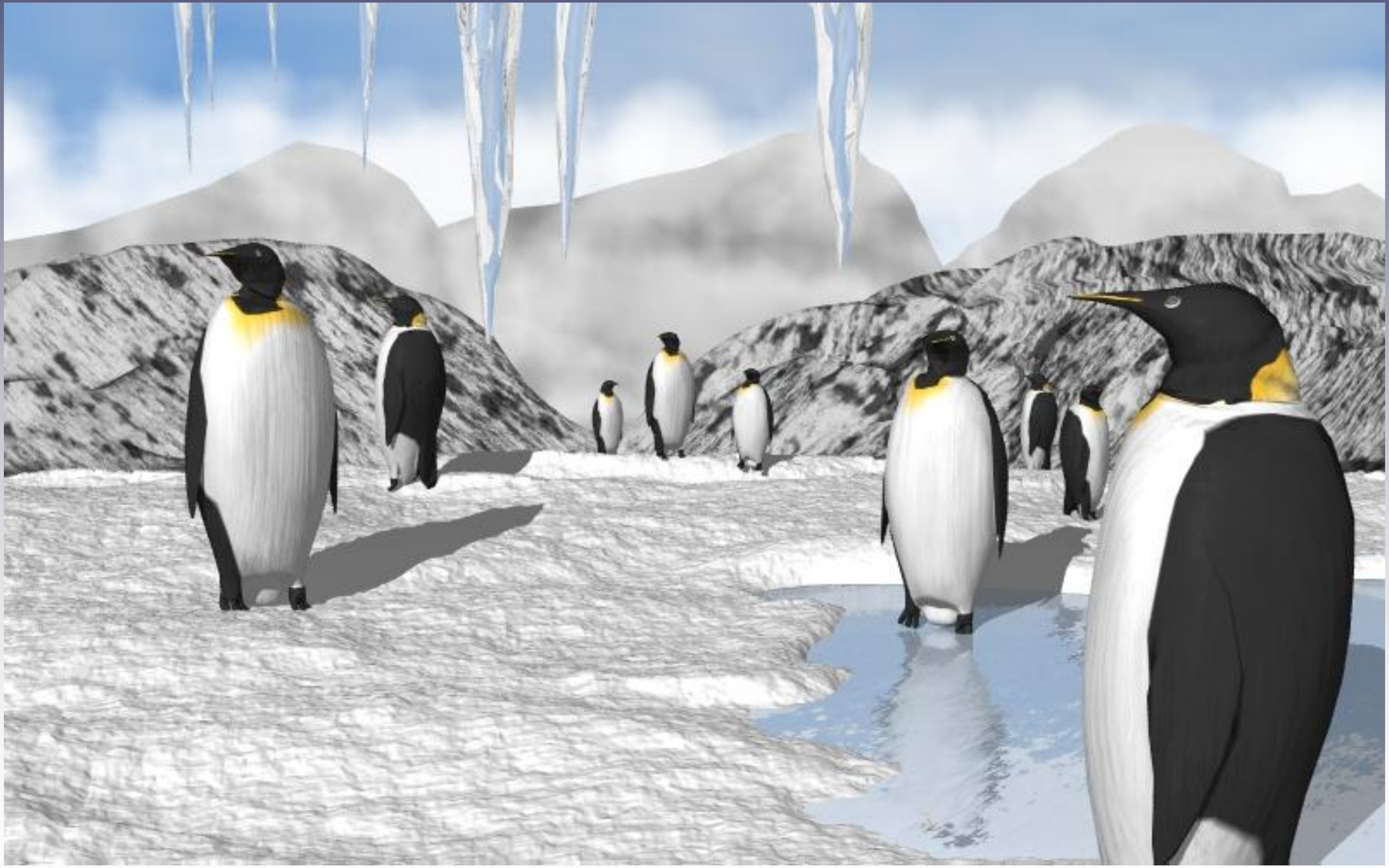


Displacement mapping

courtesy of spot3d.com

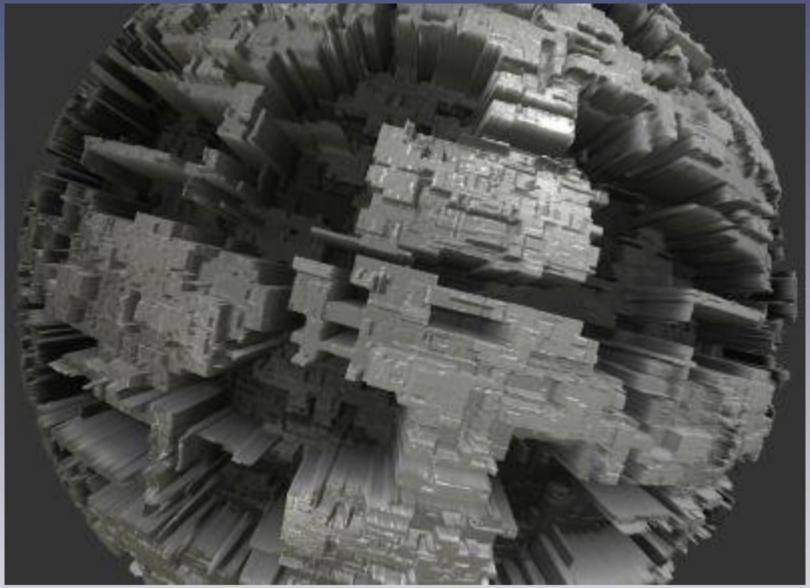
UNIVERSITY OF DELAWARE

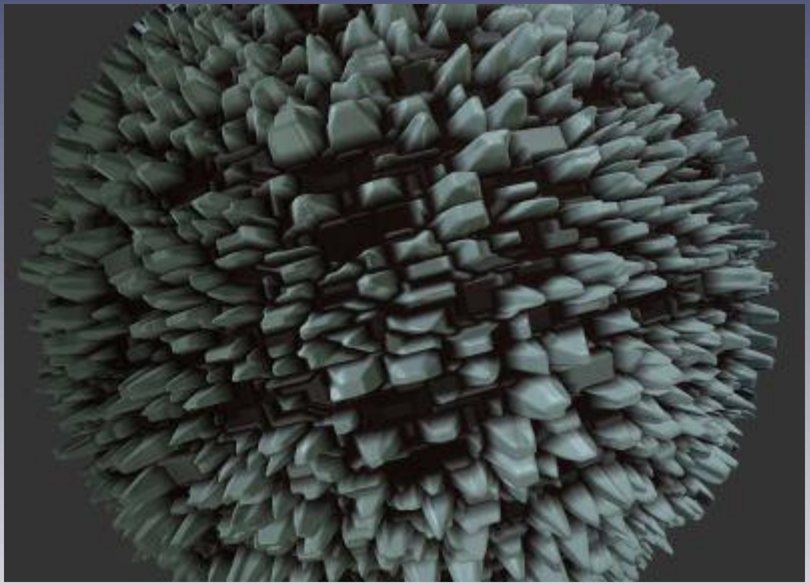
Displacement Mapping – Height Maps

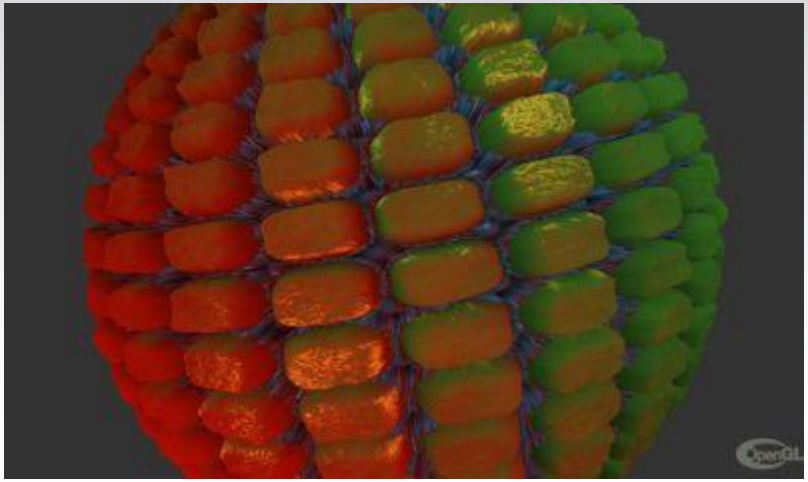


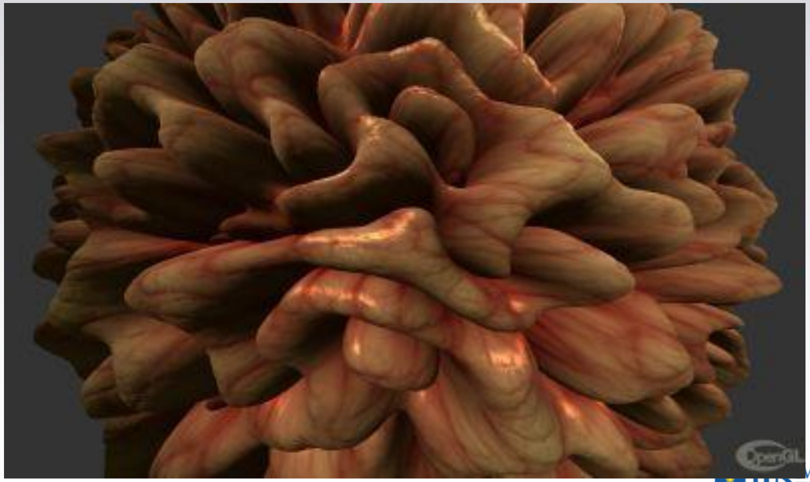
courtesy of artofillusion.org -- Julian MacDonald

Displacement Mapping the Sphere









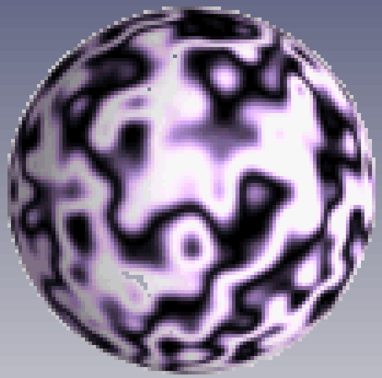
courtesy of geeks3d.com

Texture mapping: Steps

- **Creation**: Where does the texture image come from?
- **Geometry**: Transformation from 3-D shape locations to 2-D texture image coordinates
- **Rasterization**: What to draw at each pixel
  - E.g., bilinear interpolation vs. nearest-neighbor

Texturing: Creation

- Reproductions
  - Photographs
  - Handpainted
- Directly-computed functions
  - E.g., lightmaps, visibility maps
- Procedurally-built
  - Synthesize with randomness, pattern-generating rules, etc.
  - More about this in next lecture

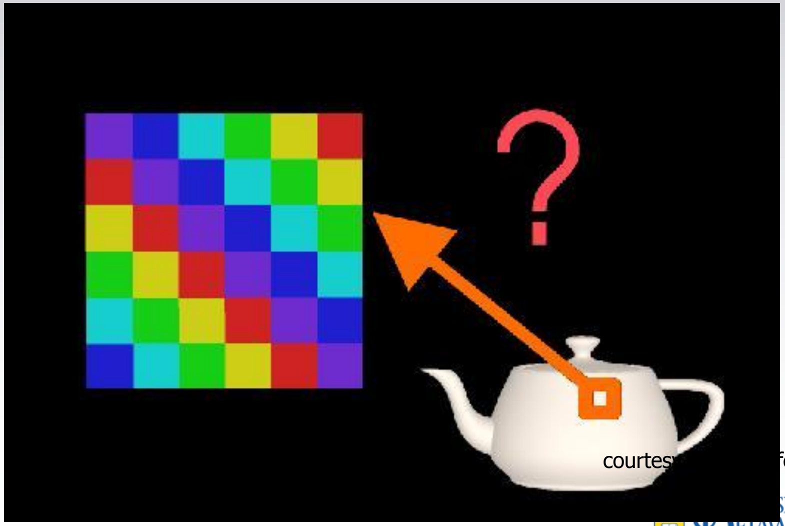


courtesy of H. Elias

Texturing Pipeline (Geometry + Rasterization)

1. Compute object space location (x, y, z) from screen space location (i, j)
2. Use **projector** function to obtain object surface coordinates (u, v) (3-D -> 2-D projection)
3. **Correspondender** function to find texel coordinates (s, t) (2-D -> 2-D transformation)
   - Scale, shift, wrap like viewport transform in geometry pipeline
4. Filter texel at (s, t)
5. Modify pixel (i, j)

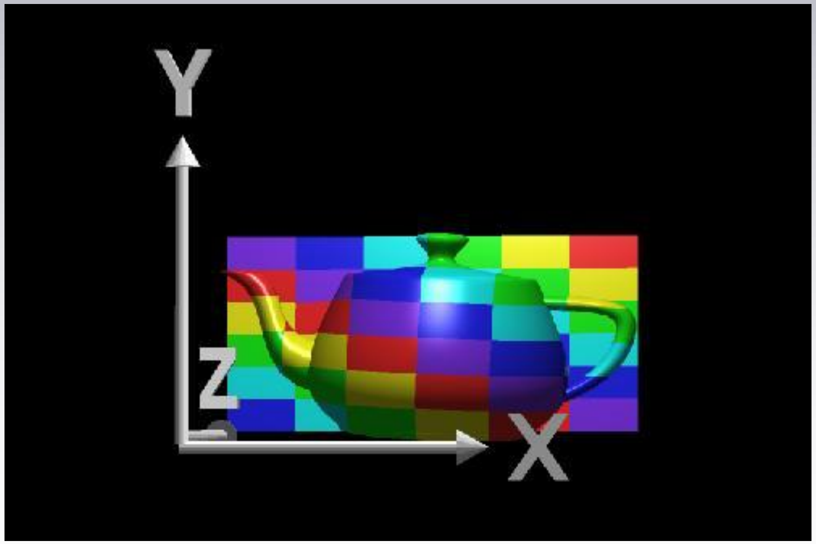
Rasterization



list adapted from Akenine-Moller & Haines

Projector Functions

- Want way to get **from 3-D point to 2-D surface** coordinates as an intermediate step
- Idea: Project complex object onto **simple object**'s surface with parallel or perspective projection (focal point inside object)
  - Plane
  - Cylinder
  - Sphere
  - Cube
  - Mesh: piecewise planar (how OpenGL does it)

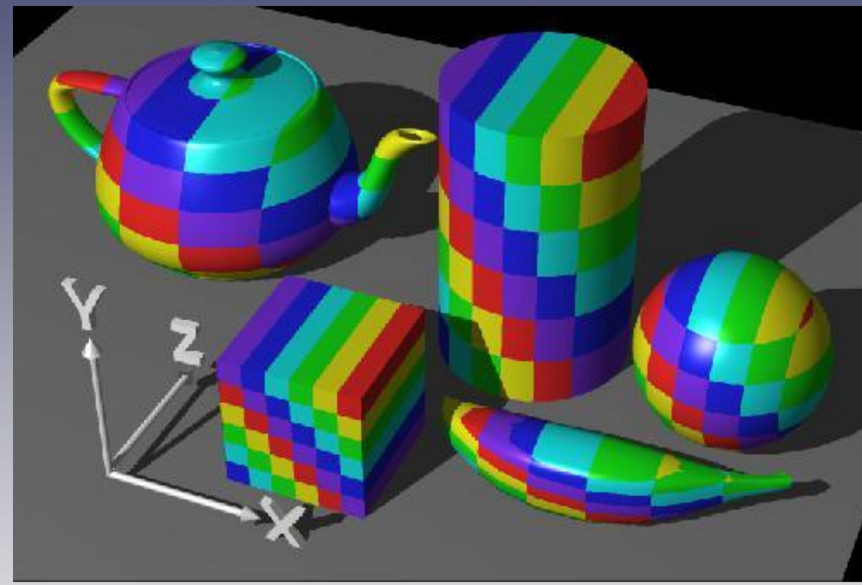


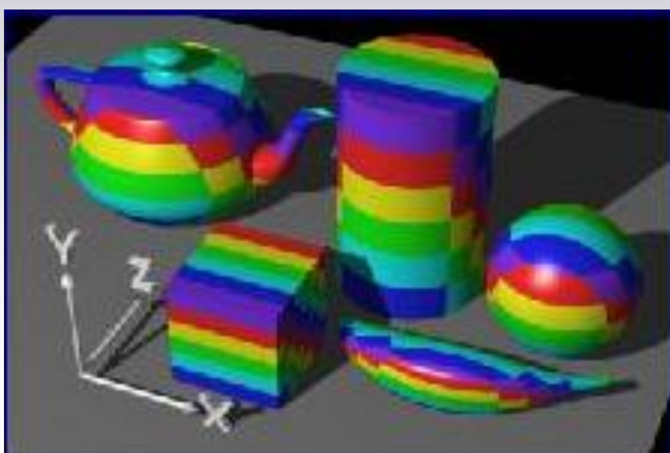
courtesy of R. Wolfe

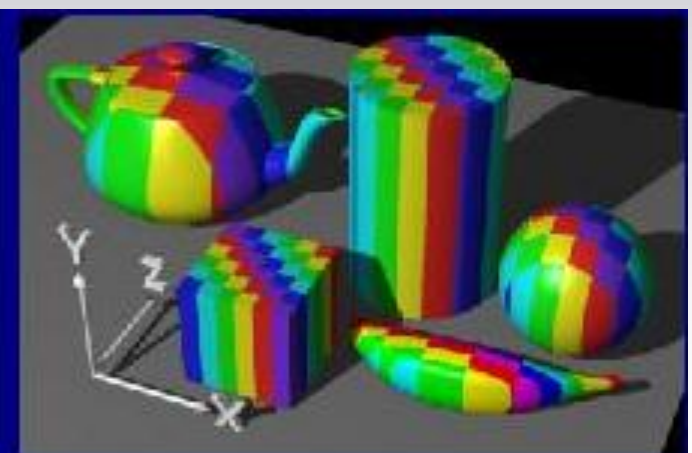
Planar projector

Planar projector

Orthographic projection onto XY plane: $u = x$, $v = y$







courtesy of R. Wolfe

...onto YZ plane

...onto XZ plane

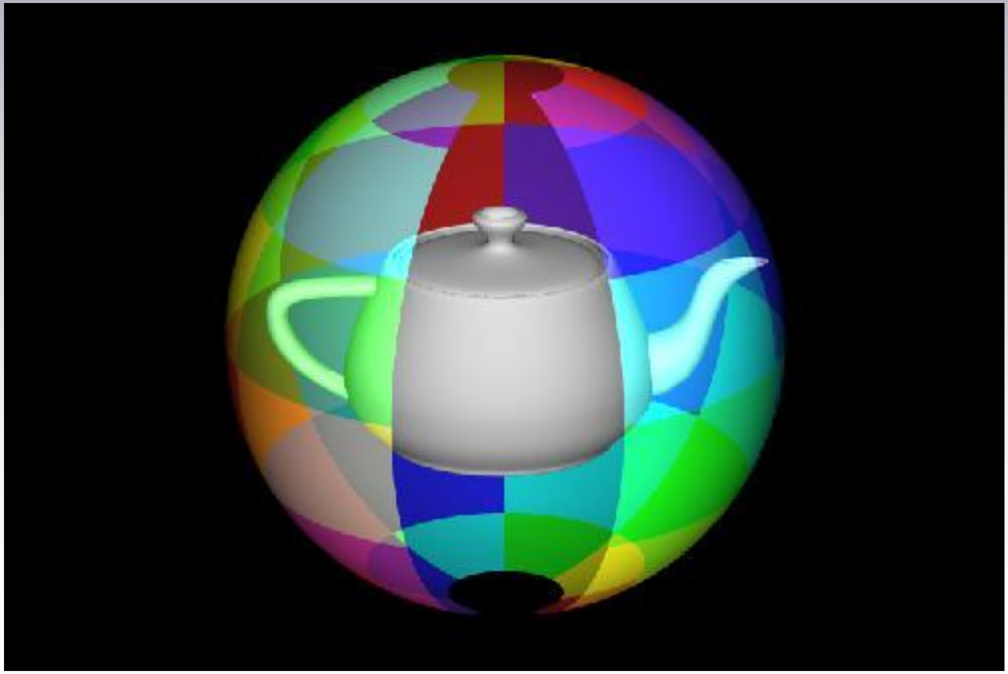
Cylindrical projector

- Convert rectangular coordinates (x, y, z) to cylindrical (r, h, θ), use only (h, θ) to index texture image

courtesy of
R. Wolfe

Spherical projector

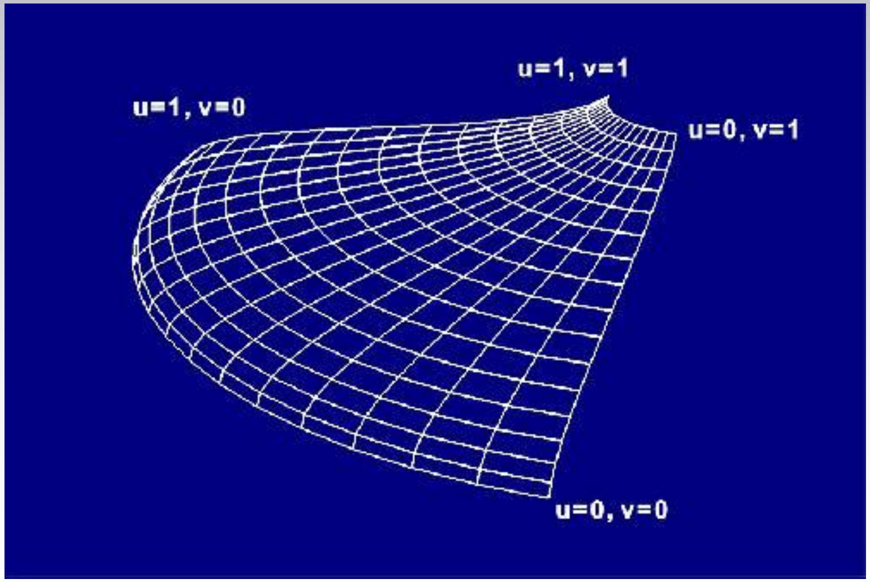
- Convert rectangular coordinates (x, y, z) to spherical (r, θ, Φ), use only (θ, Φ)



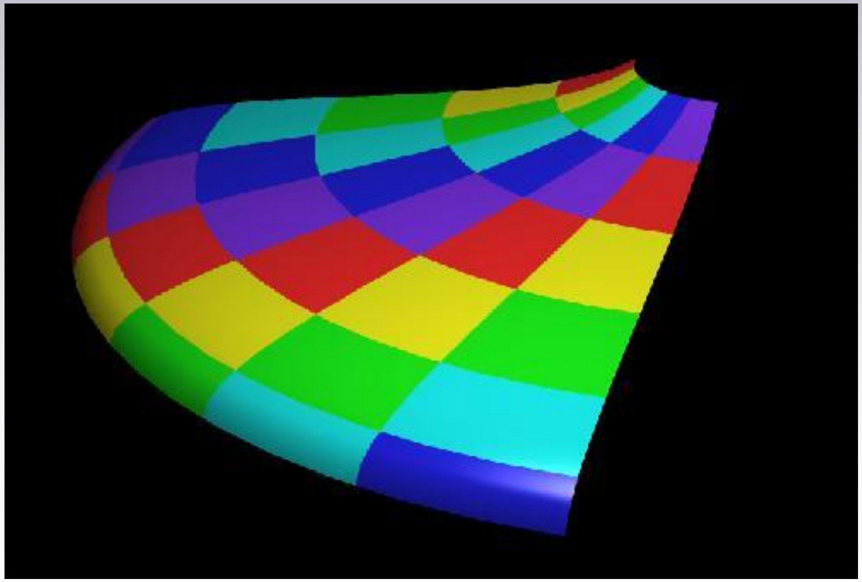
courtesy of R. Wolfe

Parametric Surfaces

- If we have a surface patch already parametrized by some **natural** (u, v) such that x = f(u, v), y = g(u, v), z = h(u, v), we can use parametric coordinates u, v without a projector

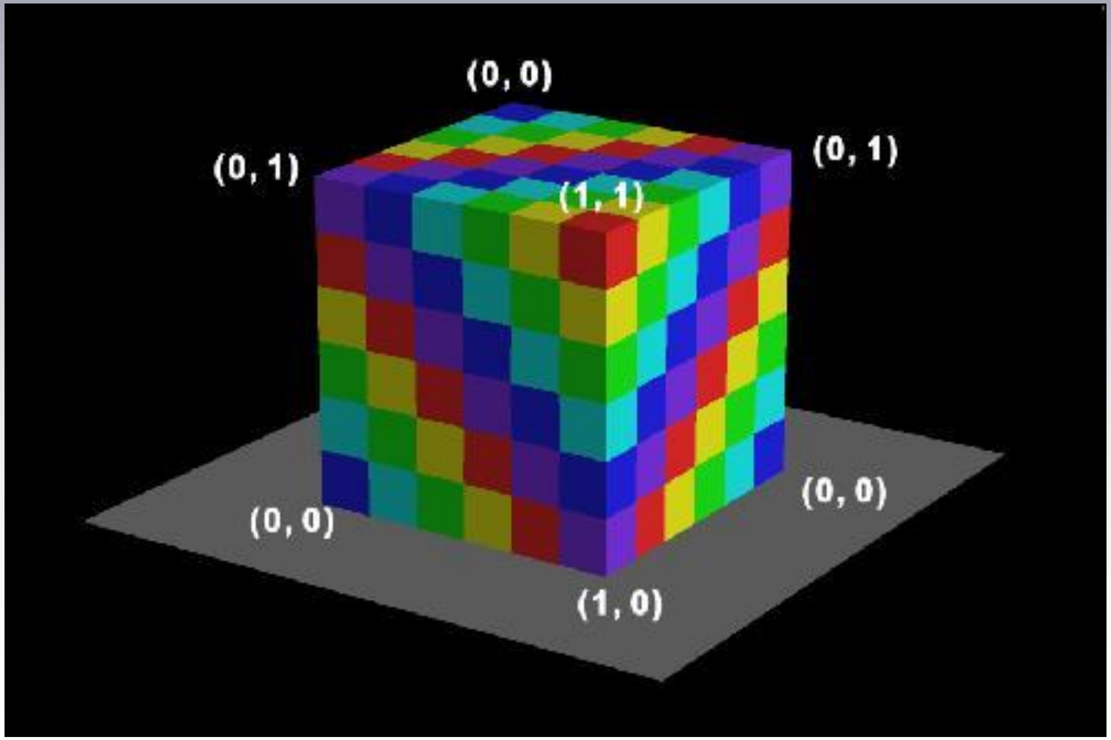


courtesy of R. Wolfe



Texture coordinates at vertices

- Polygons can be treated as parametric patches by assigning **texture coordinates** to vertices



courtesy of
R. Wolfe

OpenGL Texturing: Enabling and Drawing

- To draw textured shape, texturing must first be enabled: **`glEnable(GL_TEXTURE_2D)`**
- Load current texture image with **`glTexImage2D()`**
  - Width, height must be powers of 2 (plus 2 if border is used)
  - Only one texture current; faster to change textures by preloading all and switching with **`glBindTexture()`** rather than reloading each time (this is what Sprite.cpp does)
- Assign texture coordinates (s, t) to vertices with **`glTexCoord()`**
  - Similar to **`glColor()`** command—sets a property for subsequent vertices that holds until it is changed
  - See Robins' texture tutor