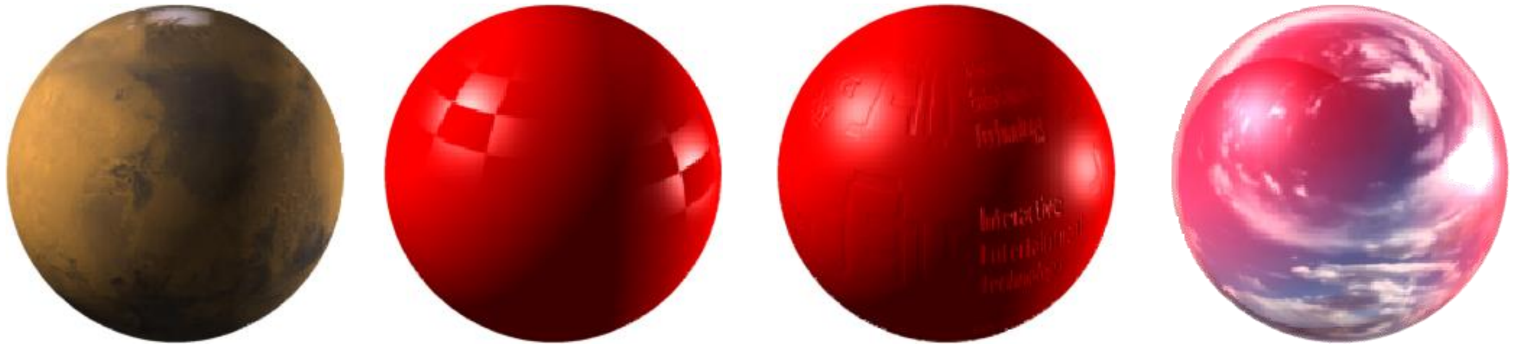
Surface Mapping One

CS7GV3 – Real-time Rendering

Textures

- Add complexity to scenes without additional geometry
  - Textures store this information, can be any dimension
- Many different types:
  - Diffuse – most common
  - Ambient, specular, gloss maps
  - Bump, normal, displacement maps
  - Reflection, refraction maps



Textures and Rendering

- Recall the Rendering Equation:

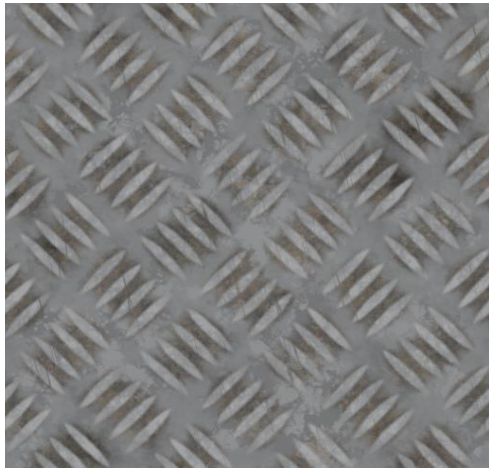
$$I(x,x') = g(x,x')[\varepsilon(x,x') + \int_S \rho(x,x',x'')I(x',x'')dx'']$$

  - I(x, x’) : intensity of light passing from x to x’
  - g(x, x’) = (geometry factor)
  - ε (x, x’) : intensity of light emitted by x and passing to x’
  - r (x, x’, x”) : bi-directional reflectance scaling factor for light passing from x” to x by reflecting off x’
- Based on last few lectures, lets simplify this as:

$$I = g\left(k_\varepsilon + \sum L^i_{df} k_{df} (\mathbf{n.l}) + L^i_s k_s (\mathbf{h.n})^\alpha + L^i_a k_a\right)$$

Diffuse Map

- Simplest texturing directly applies a colour to object
  - In basic illumination colour is mostly based on diffuse component



$$I = g\left(k_{\varepsilon} + \sum L^{i}_{df} k_{df} (\mathbf{n.l}) + L^{i}_{s} k_{s} (\mathbf{h.n})^{\alpha} + L^{i}_{a} k_{a}\right)$$

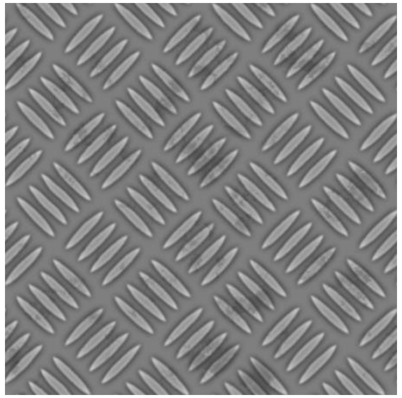
Emission Map

- A.k.a. Glow map
  - Models light sources on a surface
  - Does not depend on surface normal or light sources.

$$I = g\left(k_\varepsilon + \sum L_{df}^i k_{df}\left(\mathbf{n.l}\right) + L_s^i k_s\left(\mathbf{h.n}\right)^\alpha + L_a^i k_a\right)$$

Specular Map

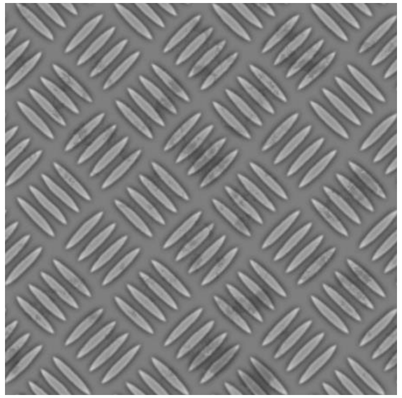
- Specular color : “Gloss map”
- Shininess
  - Control the **specular exponent** in Phong light model
  - Or roughness in other light models



$$I = g\left(k_\varepsilon + \sum L_{df}^i k_{df}(\mathbf{n.l}) + L_s^i k_s (\mathbf{h.n})^\alpha + L_a^i k_a\right)$$

Specular Map + Specular Exp. Map

Diffuse Color

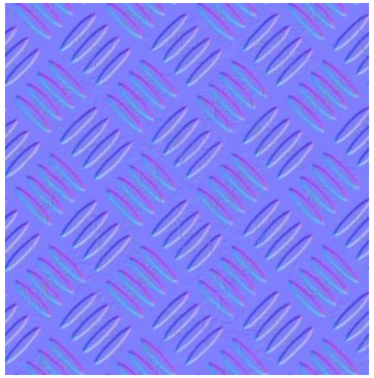


Specular Color

Specular Exponent

Normal Map

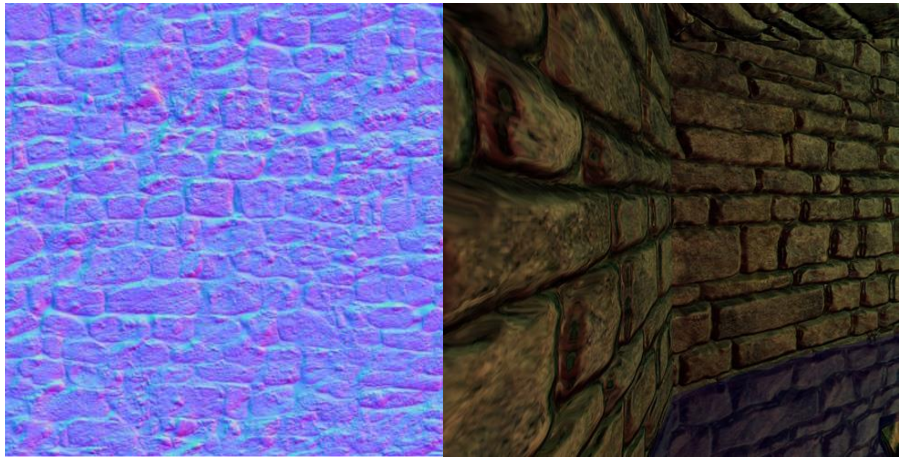
- Use texture as a input to perturb geometric representation

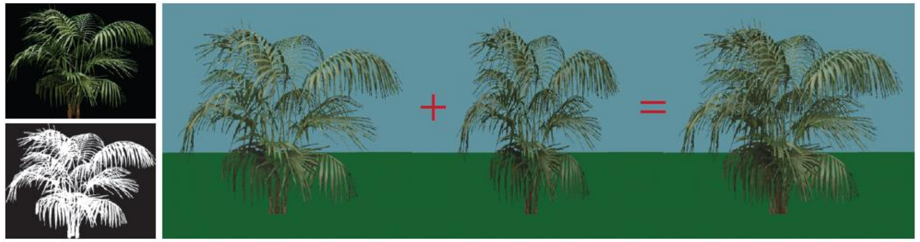


$$I = g\left(k_{\varepsilon} + \sum L_{df}^{i} k_{df} \left(\mathbf{n.l}\right) + L_{s}^{i} k_{s} \left(\mathbf{h.n}\right)^{\alpha} + L_{a}^{i} k_{a}\right)$$

Visibility Mappings

- Parallax mapping
  - Shift in texture lookup based on view-ray intersection with heightfield
  - Can account for occlusions
- Alpha Mapping:
  - Transparency mapping across primitive
- Shadow mapping

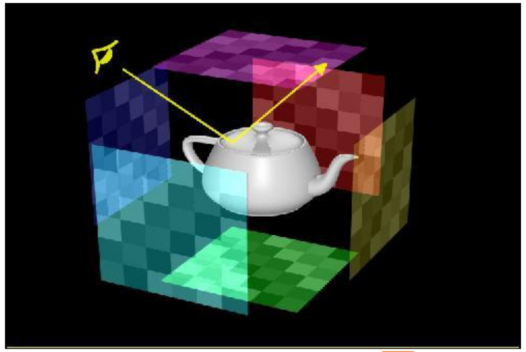




$$I = g\left(k_{\varepsilon} + \sum L^{i}_{df} k_{df} (\mathbf{n}.\mathbf{l}) + L^{i}_{s} k_{s} (\mathbf{h}.\mathbf{n})^{\alpha} + L^{i}_{a} k_{a}\right)$$

Light & Environment Mapping

- Approximation of incoming radiance
  - Light maps: pre-computed incident flux across 2D surfaces
  - Environment Maps: more complex 3D look up of incoming flux in scene
  - Can combine with other maps



$$I = g\left(k_{\varepsilon} + \sum L^{i}_{df} k_{df} (\mathbf{n.l}) + L^{i}_{s} k_{s} (\mathbf{h.n})^{\alpha} + L^{i}_{a} k_{a}\right)$$

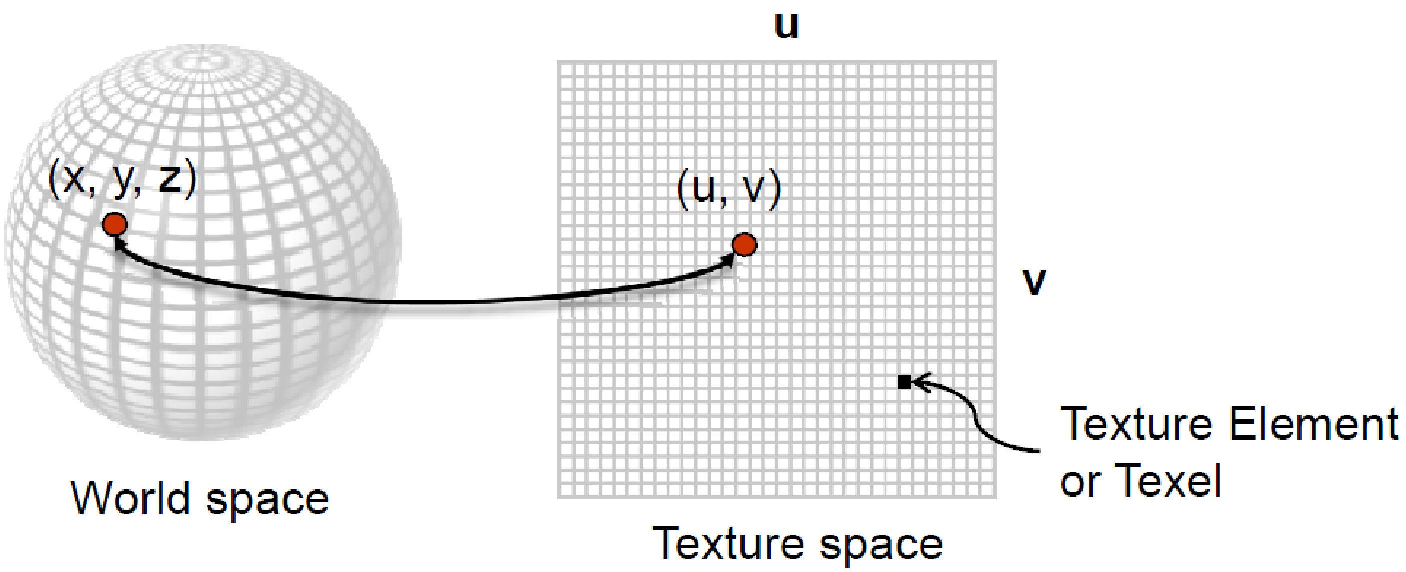
Texture d environment

Light Maps

Merged Scene

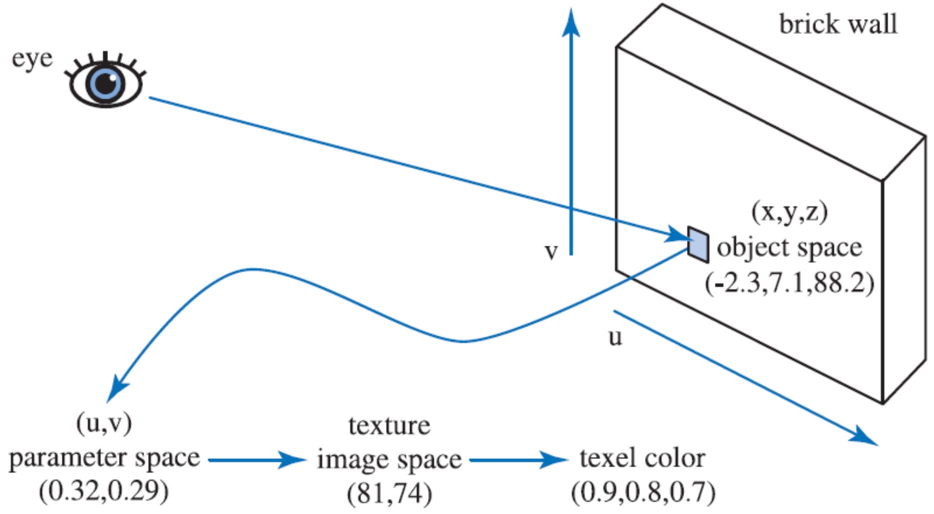
Texture Mapping

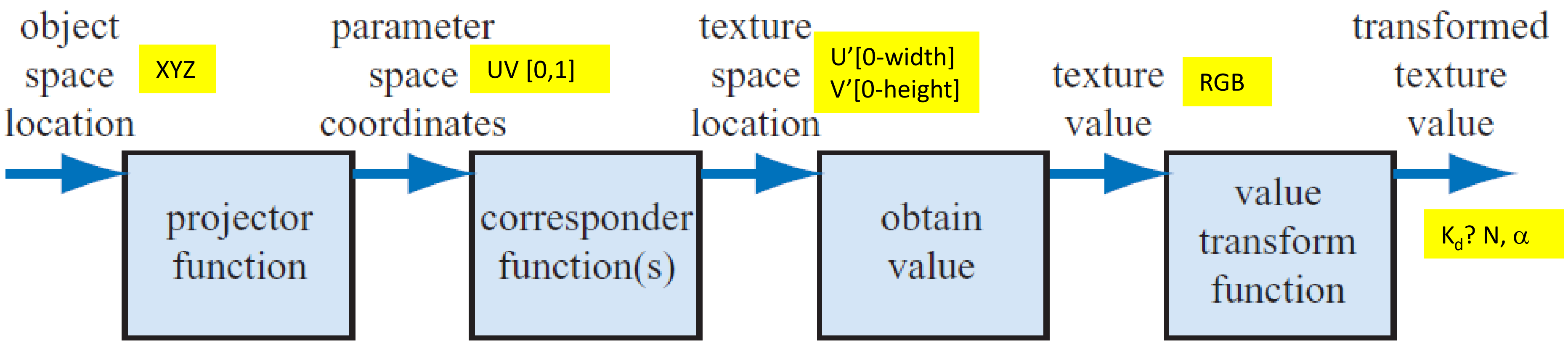
- For 2D textures, we need a 3D to 2D mapping
  - Sometimes called a "projector function"



  - Each vertex in the model will need a (u,v) co-ordinate
  - Normally defined by the artist and added to the vertex stream

Texture Mapping Components



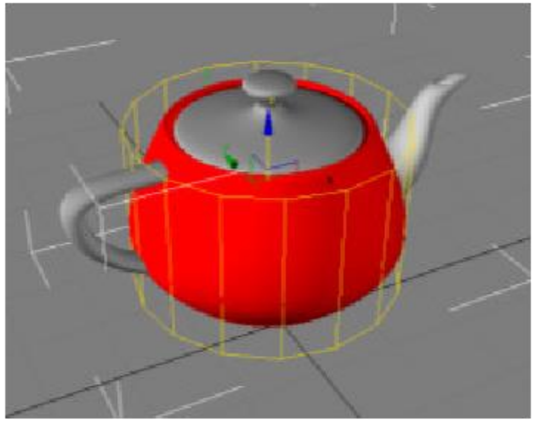


From: Akenine-Moller, Haines, Hoffman "**Real-time Rendering – 3rd Ed.**"

UV Mappings

Spherical Cylindrical Box Planar Wrap

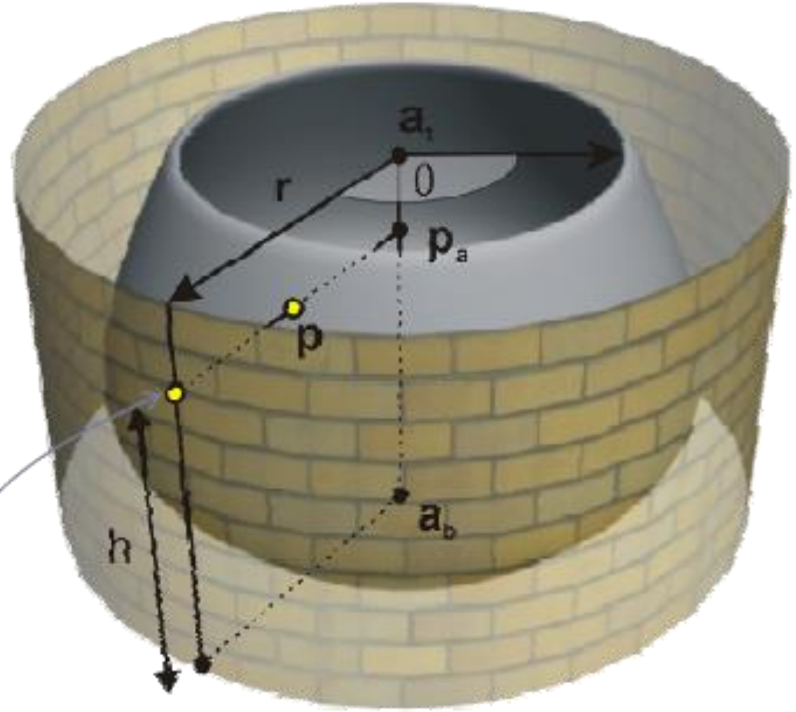
Uses multiple mappings



E.g. main body uses cylindrical

Example: Cylindrical

$$
\begin{aligned}
H &= ||\mathbf{a}_t - \mathbf{a}_b|| \\
\mathbf{a} &= \frac{\mathbf{a}_t - \mathbf{a}_b}{H} \\
h &= (\mathbf{p} - \mathbf{a}_b) \cdot \mathbf{a} \\
\mathbf{p}_a &= h\mathbf{a} \\
\mathbf{r} &= \frac{\mathbf{p} - \mathbf{p}_a}{||\mathbf{p} - \mathbf{p}_a||} \\
\theta &= \cos^{-1}(r_x)
\end{aligned}
$$



$$(u, v) = \left(\frac{\theta}{2\pi}, \frac{h}{H}\right)$$

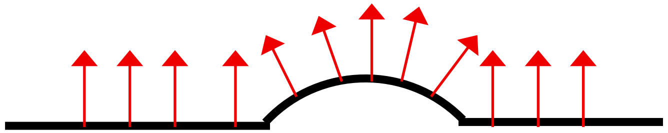
Tiling, Wrapping etc.

- Can specify rules for (u, v) behaviour outside [0, 1]
  - Tiling: number of repetitions of texture within space
  - Tiling Mode: normal, mirror, no-tiling (clamp)

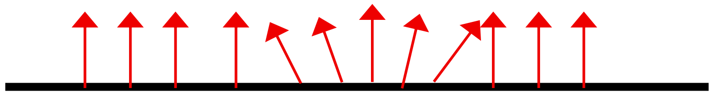
- The (u, v) co-ordinate can be manipulated by shader
  - Just another input – e.g. could rotate texture co-ordinate

Bump Mapping

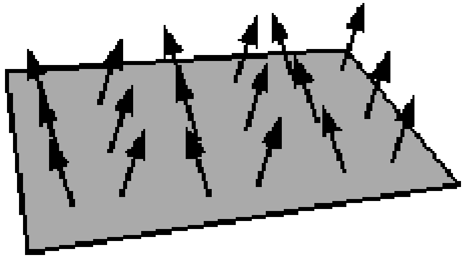
- Add surface geometric detail without additional vertices



A “real” bump distorts the directions of the normals (this effects calculations of light reflectance)



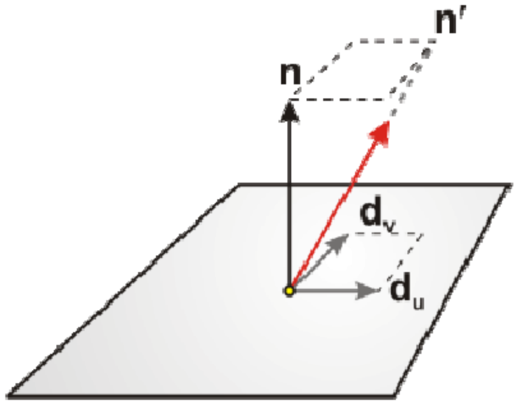
“Fake” bumps created by distorting the normals although the model geometry is still flat.



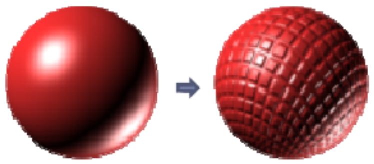
e.g. A bump-map texture applied to a flat polygon.

Bump Mapping

- Use a texture map to perturb the normal to the surface



$$\mathbf{n}' = \frac{\mathbf{n} + \mathbf{d}}{||\mathbf{n} + \mathbf{d}||}$$



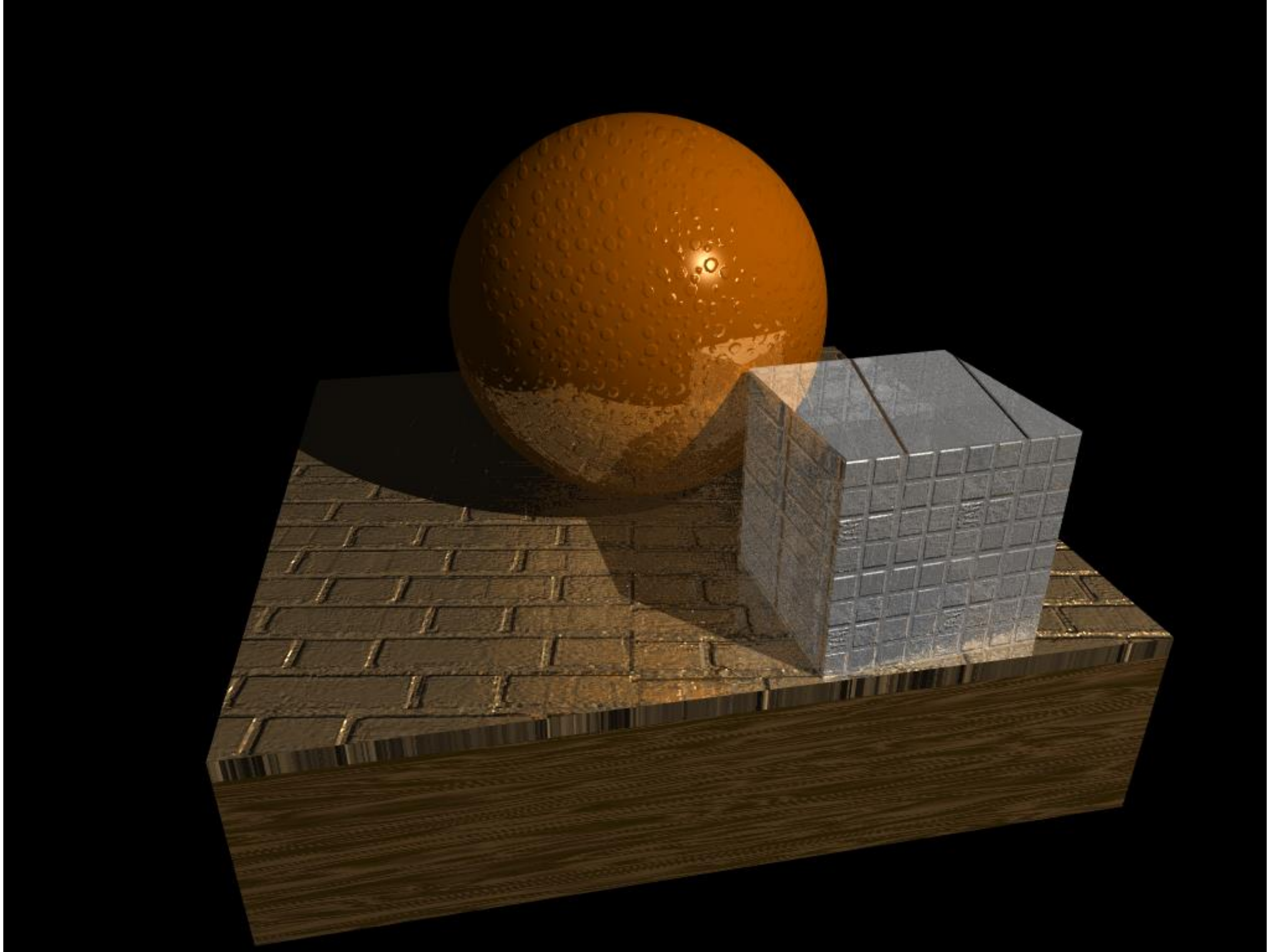
- Traditionally represent this as either:
  - d stored in 2 separate scalar textures (for d_u and d_v)
  - store a heightfield and compute or approximate d from the surface differentials

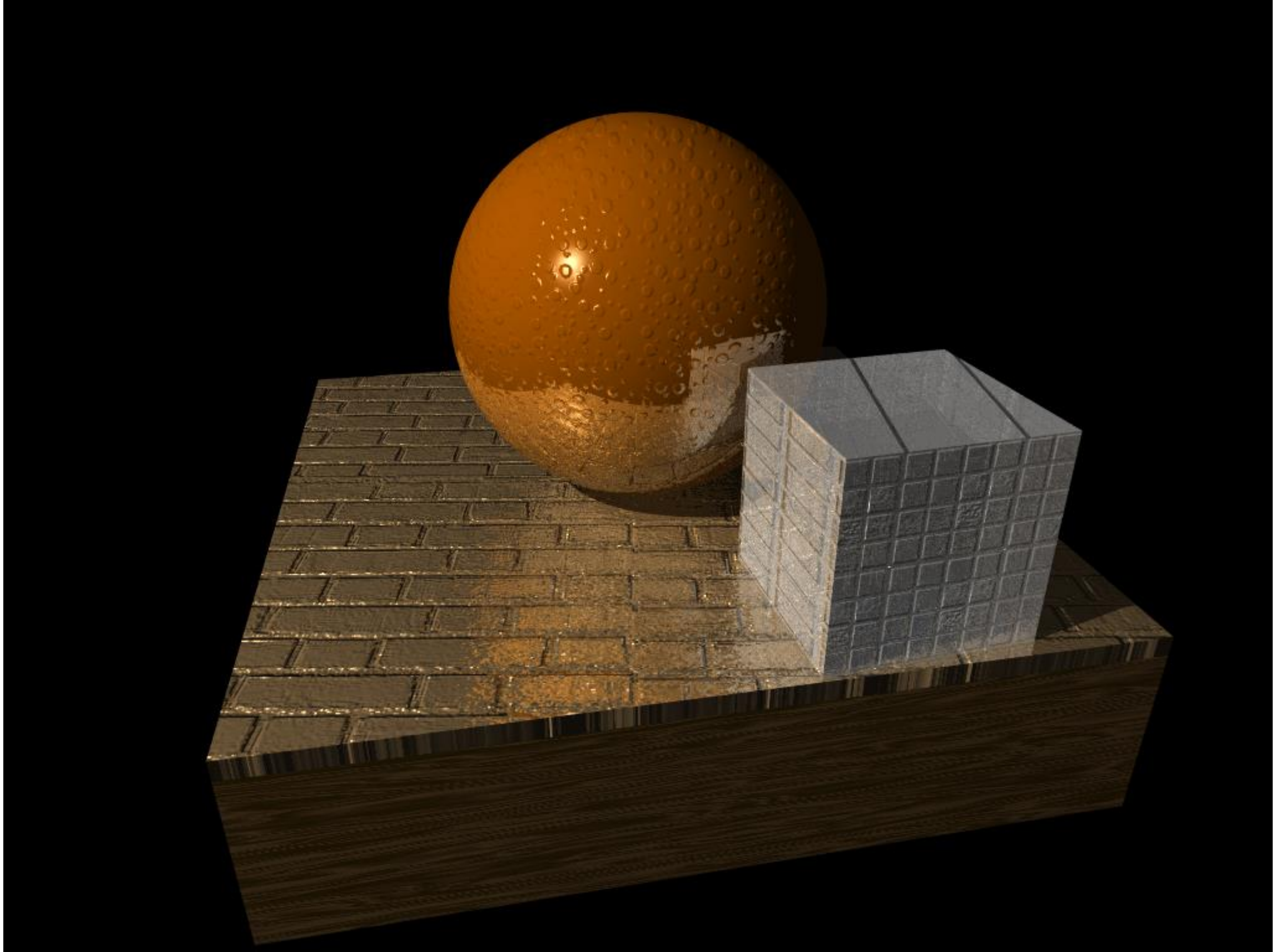
Bump/Normal Mapping

- Give the illusion of geometric detail
- Shape perception depends on lighting cues

Without normal mapping

With bump mapping





Normal Mapping

- A more commonly used method is normal mapping
  - Also known as dot3 bump mapping

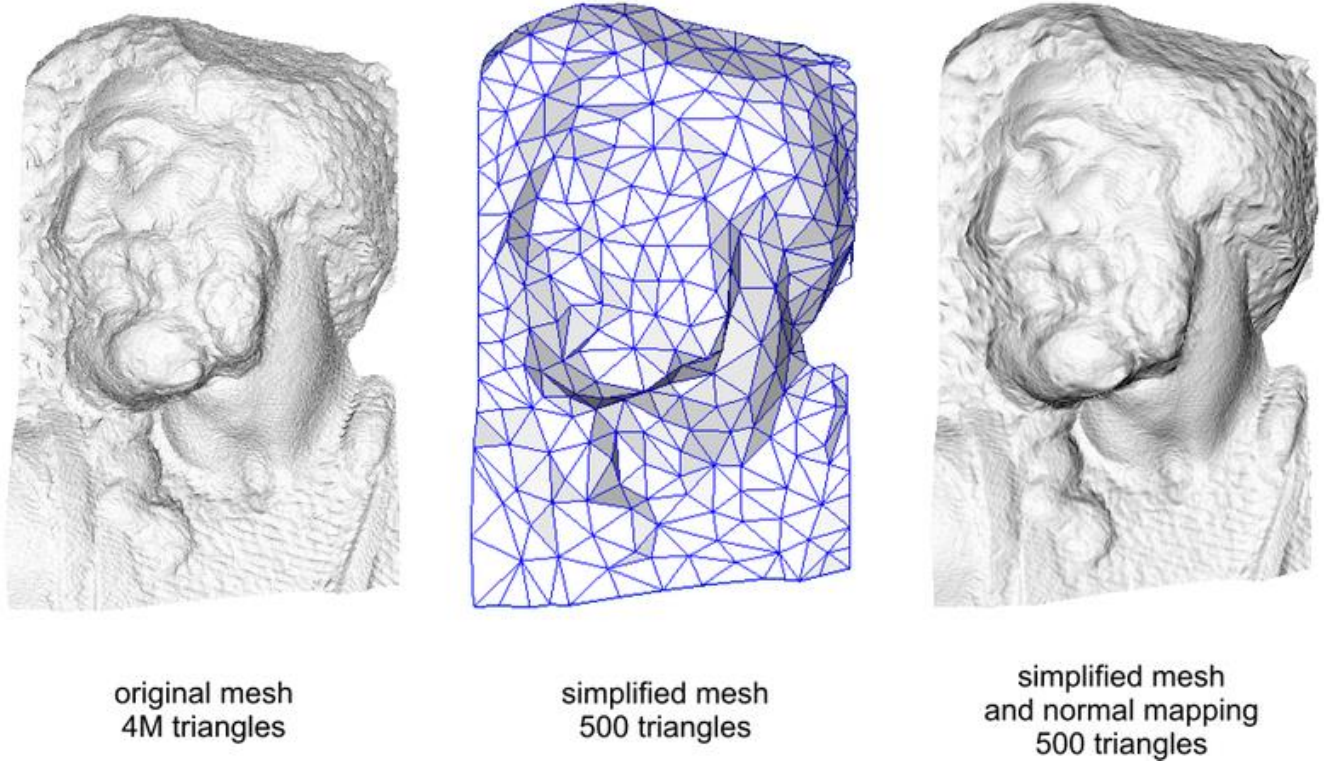
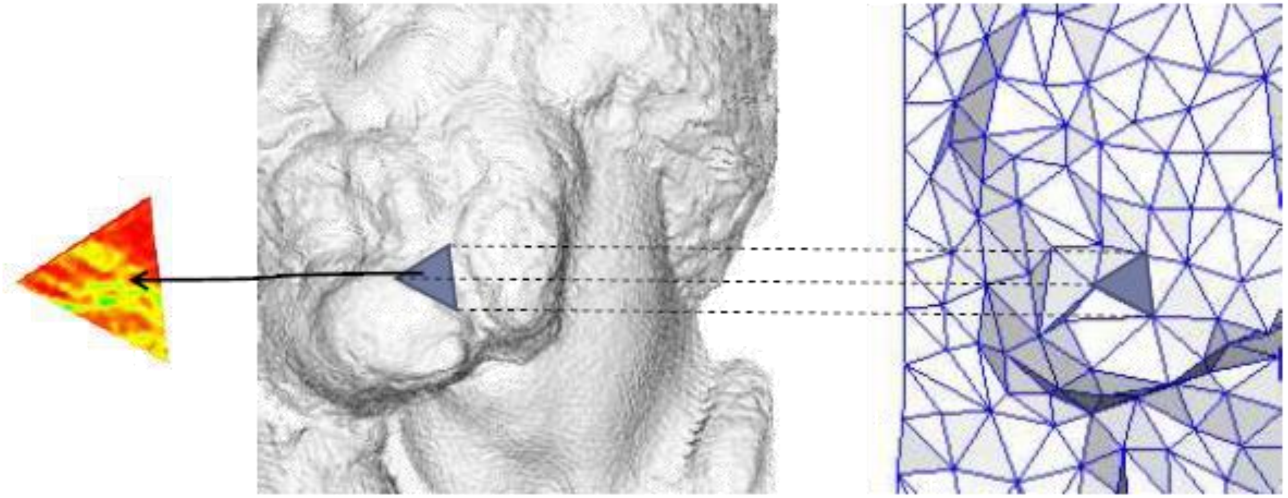


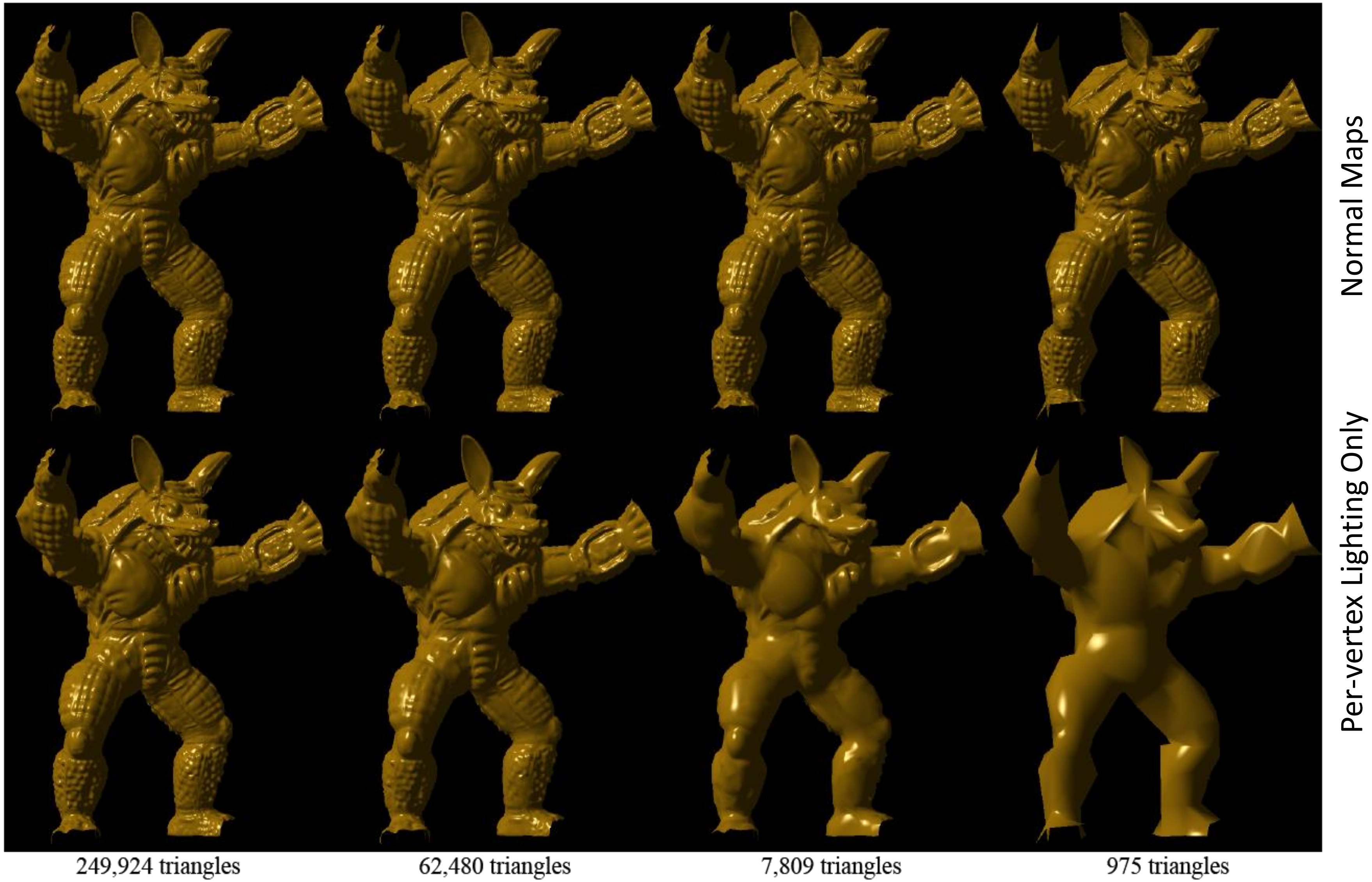
Image taken from Wikipedia available at http://en.wikipedia.org/wiki/Normal_mapping

Object Space Normal Mapping

- Basic Idea:
  - Store the actual surface normal in the texture (RGB = nx, ny, nz)
  - At each pixel, look up the normal map, and use this instead of the interpolated normal
  - Tool support required if generating normals from high-res surfaces



e.g. Combine with Mesh Simplification



Cohen, Olano, Manocha, “Appearance-preserving simplification”, SIGGRAPH, 1998. http://gamma.cs.unc.edu/APS/APS.pdf

Object Space Normal Mapping

- Object space has some problems:
  - Not very flexible
  - Strongly tied to specific object
  - Can’t tile map or use symmetry
  - Don’t work so well with MIP maps or sharp edges

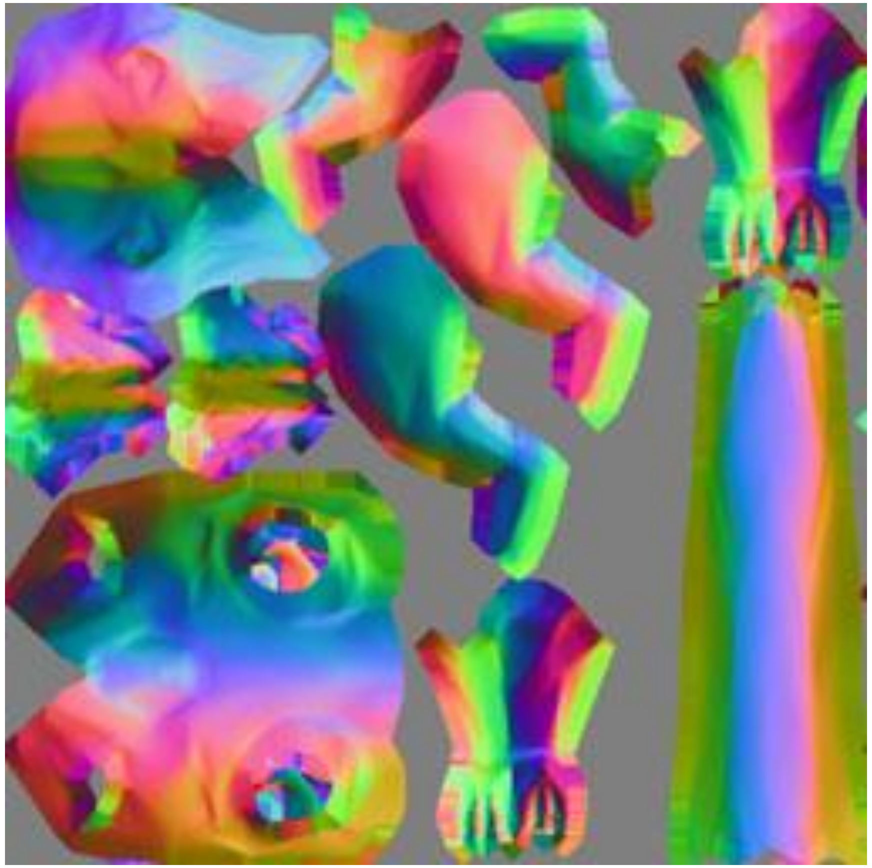


Image from http://www.3dkingdoms.com/tutorial.htm

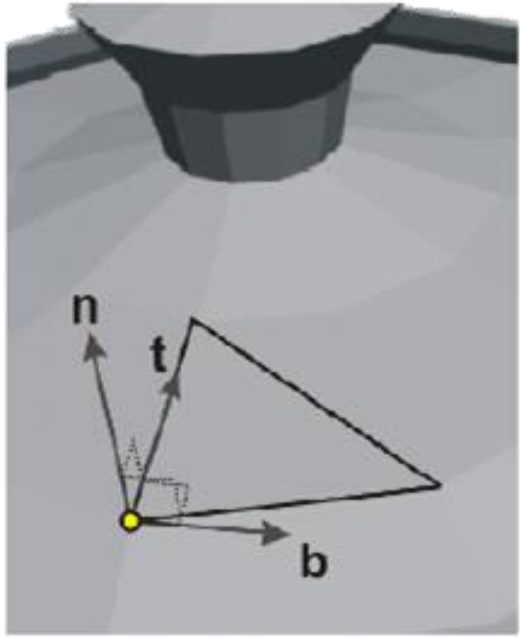
Tangent Space Normal Mapping

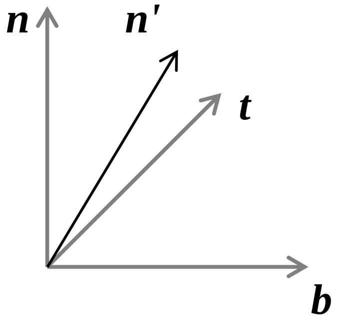
- Normal is stored relative to the **tangent space** of the object
  - Sort of like a “local normal”
  - Define a local co-ordinate frame
  - Form a co-ordinate system from normal **n** and tangent **t** and *binormal* **b**

$$\mathbf{b} = \mathbf{t} \times \mathbf{n}$$

$$\mathbf{M}_T = \begin{bmatrix} \mathbf{t} \\ \mathbf{b} \\ \mathbf{n} \end{bmatrix} = \begin{bmatrix} t_x & t_y & t_z \\ b_x & b_y & b_z \\ n_x & n_y & n_z \end{bmatrix}$$

  - Then define displaced normal **n’** in this space





Shader Setup

- Application must send normal and tangent vector to the shader
  - The normal is straightforward – available as built in attribute
  - The tangent is slightly tricky (for polygonal objects)
    - Pass this down as a custom attribute
    - Bi-tangent can be calculated in the shader
  - Must be consistent with the tangent vector to avoid interpolation problems

Lighting with the bump map

- Transform light and view vectors into tangent space (per vertex)
  - Vertex Shader:

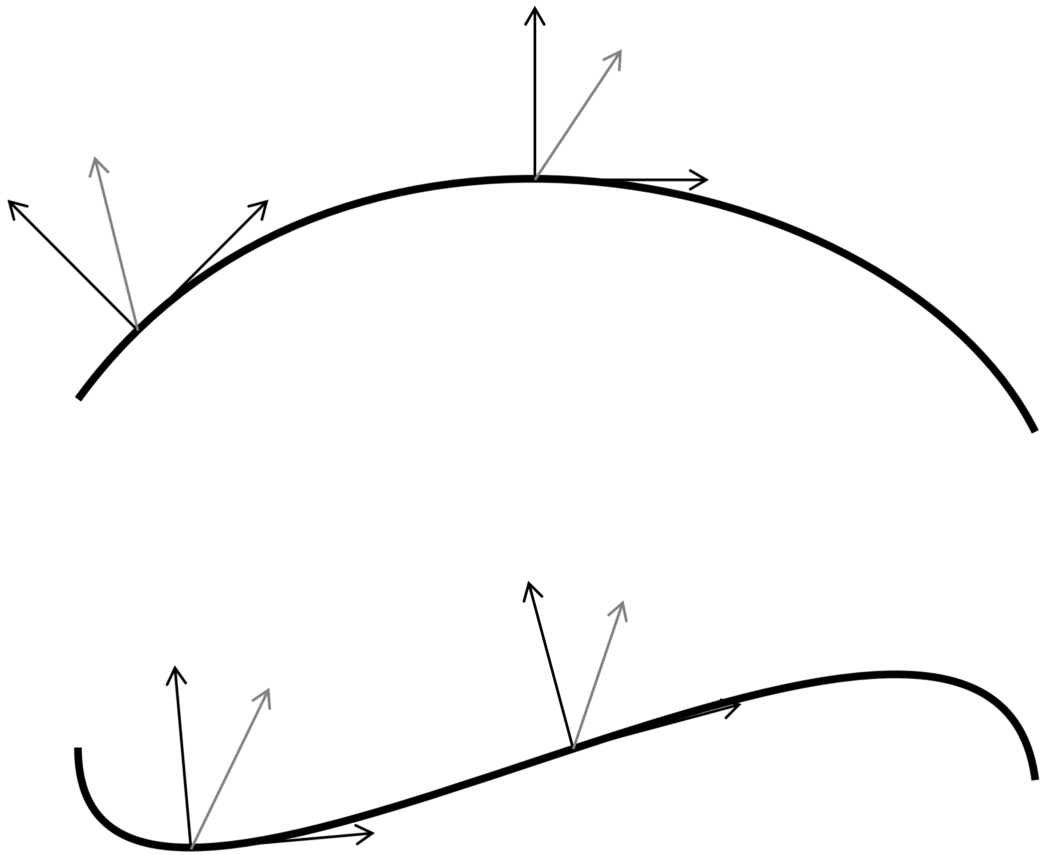
```
varying vec3 v;
varying vec3 l;

uniform vec4 L; //directional light in eye space

attribute vec3 rm_Tangent;
attribute vec3 rm_Bitangent;     //here we get the bitangent from application
                                 //but we could calculate this in the shader

void main(void)
{
   gl_Position = ftransform();
   gl_TexCoord[0] = gl_TextureMatrix[0]*gl_MultiTexCoord0;

   vec4 camera = gl_ModelViewMatrixInverse*vec4(0.0, 0.0, 0.0, 1.0);
   vec3 view = normalize(camera.xyz-gl_Vertex.xyz); //object space view and light vector
   vec3 light = normalize(gl_ModelViewMatrixTranspose*L).xyz;

   //TBNinv transforms vectors from object space to tangent space
   mat3 TBNinv(rm_Tangent, rm_Bitangent, gl_Normal);
   l = TBNinv*light;
   v = TBNinv*view;
}
```

Lighting with the bump map

- Fragment Shader

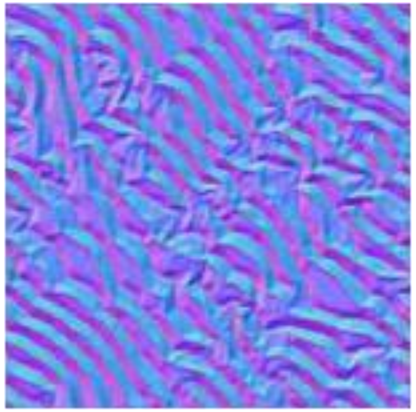
```
uniform vec4 ambientColor;
uniform vec4 diffuseColor;
uniform vec4 specularColor;
uniform sampler2D diffuseTex;
uniform sampler2D normalTex;
uniform float shininess;

varying vec3 v;
varying vec3 l;

void main(void)
{
   l = normalize(l);
   v = normalize(v);

   vec3 n = 2.0*texture2D( normalTex, gl_TexCoord[0].st ).xyz - 1.0; //tangent-space normal
   vec4 diffuseTerm = texture2D( diffuseTex, gl_TexCoord[0].st)* diffuseColor *(max( 0.0, dot(n, l)));
   vec3 r = reflect(-l, n); //tangent-space reflection vector
   vec4 specularTerm = specularColor*pow(max(0.0, dot(r, v)), shininess);

   gl_FragColor = ambientColor + diffuseTerm + specularTerm;

}
```

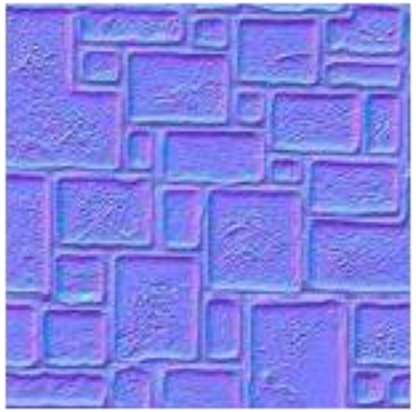
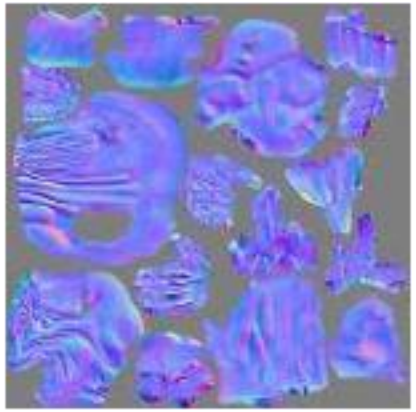
Tangent Space Normal Mapping



Tangent Space Normal Mapping

- Predominantly blue:



- Why?
  - No displacement means normal = n in tangent space
  - n = [0 0 1] which maps to RGB blue
  - Displaced normals are relatively close to this
- Storage:
  - Record 255*(n'+1)/2 in normal map (to map to [0,255] range)