Computer Graphics Lecture 08: Texture Mapping

DR. ELVIS S. LIU
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY
SPRING 2018

Texture Mapping – Motivations

- Sophisticated illumination models
  - Realistic physics-based looking surfaces
  - Not easy to model
  - Mathematically and computationally challenging
- Phong illumination/shading
  - Easy to model
  - Relatively quick to compute
  - Dull surfaces

Texture Mapping – Motivations (cont.)

- Surfaces “in the wild” are very complex
- Cannot model all the fine variations
- We need to find ways to add surface detail
- How?

How to add more detail to a model?

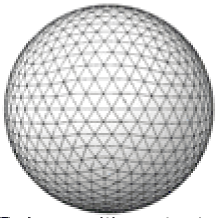
- Add more detailed geometry; more, smaller triangles:
  - Pros: Responds realistically to lighting, other surface interaction
  - Cons: Difficult to generate, takes longer to render, takes more memory space
- Map a texture to a model:
  - Pros: Can be stored once and reused, easily compressed to reduce size, rendered very quickly, very intuitive to use, especially useful on far-away objects like terrain, sky, “billboards” (texture mapped quad) - all used extensively in videogames, etc.
  - Cons: Very crude approximation of real life. Surfaces still look smooth since geometry is not changed. Need to consider perspective for real effectiveness

Texture Mapping – a cheat

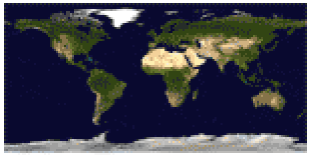
- Map surface details from a predefined, easy to model table (texture) to a simple polygon

Texture Mapping – Overview

- Texture mapping:
  - Implemented in hardware on every GPU
  - Simplest surface detail hack, dating back to the ‘60s GE flight simulator and its terrain generator
- Technique:
  - “Paste” the texture, a photograph or pixmap (e.g., a brick pattern, a wood grain pattern, a sky with clouds) on a surface to add detail without adding more polygons
  - Map texture onto surface to assign surface color (vs. using object color) or to alter object’s surface color
  - Think of texture map as stretchable contact paper



Sphere with no texture



Texture image

Sphere with texture

What to put in a texture map?

- Diffuse, ambient, specular, or any kind of color
- Specular exponents, transparency or reflectivity coefficients
- Surface normal data (e.g. normal mapping or bump mapping)
- Projected reflections or shadows

Mapping Process

- A function is a mapping
  - Takes any value in the domain as an input and outputs (“maps it to”) one unique value in the co-domain.
- Mappings in “Intersect”: linear transformations with matrices
  - Map screen space points (input) to camera space rays (output)
  - Map camera space rays into world space rays
  - Map world space rays into un-transformed object space for intersecting
  - Map intersection point normals to world space for lighting
- Mapping a texture:
  - Take points on the surface of an object (domain)
  - Return a corresponding entry in the texture (co-domain)

What is an image?

- How can I find an appropriate value for an arbitrary (not necessarily integer) index?
  - How would I rotate an image 45 degrees?
  - How would I translate it 0.5 pixels?

What is a texture?

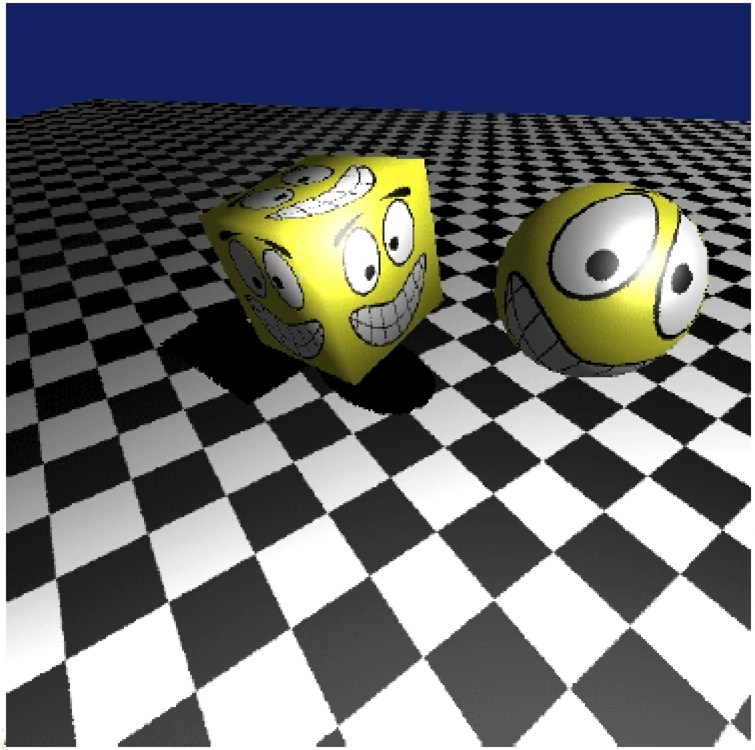
- Given the (texture/image index) (u,v), want:
- – F(u,v) ==> a continuous reconstruction
  - = { R(u,v), G(u,v), B(u,v) }
  - = { I(u,v) }
  - = { index(u,v) }
  - = { alpha(u,v) }
  - = { normals(u,v) }
  - = { surface_height(u,v) }
  - = ...

What is a texture?

- Color
- Specular ‘color’ (environment map)
- Normal vector perturbation (bump map)
- Displacement mapping
- Transparency

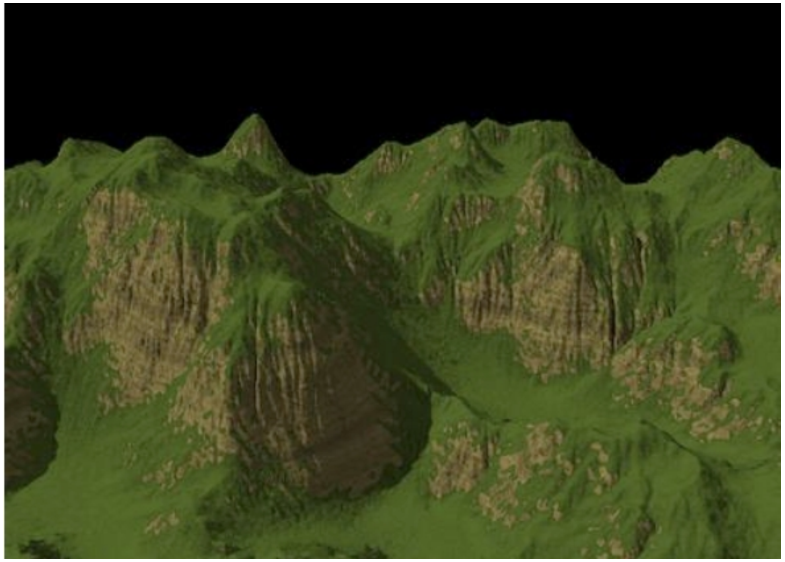
RGB Textures

- Places an image on the object
- “typical” texture mapping



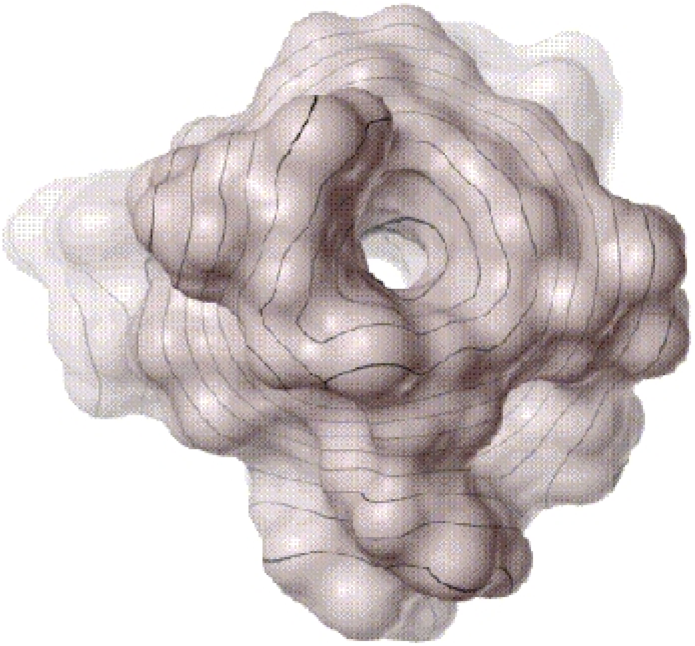
Dependent Textures

- Perform table look-ups after the texture samples have been computed



Intensity Modulation Textures

- Multiply the objects color by that of the texture

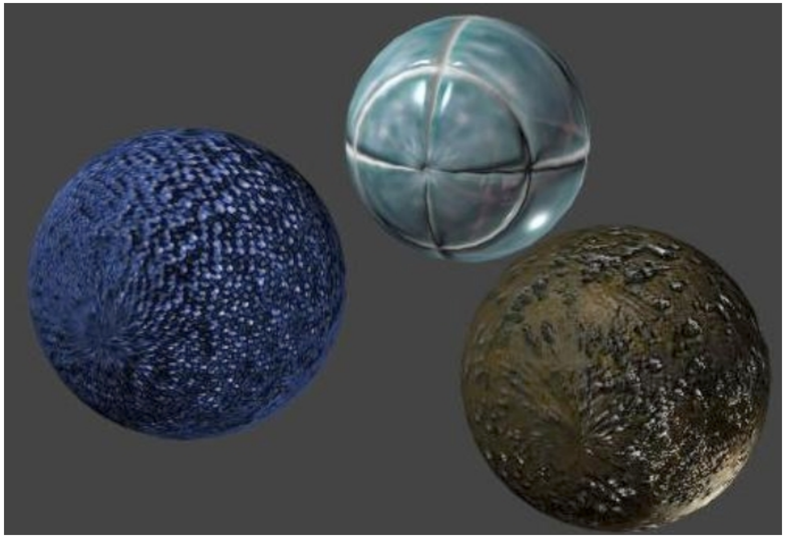


Opacity Textures

- A binary mask, really redefines the geometry

Bump Mapping

- Modifies the surface normals

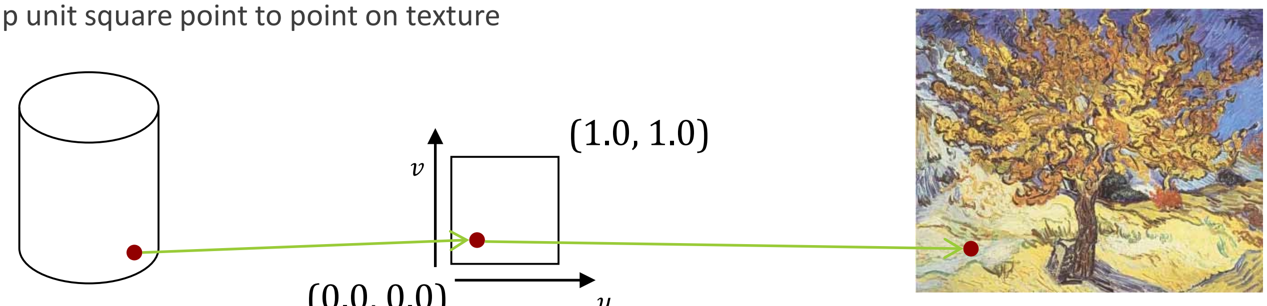


Displacement Mapping

- Modifies the surface position in the direction of the surface normal

Texture Mapping Technique

- Texture mapping is process of mapping a geometric point in space to a value (color, normal, other…) in a texture map of arbitrary width and height
  - The goal is to map any arbitrary object geometry to a texture map
  - Done in two steps:
    - Map a point on object to a point on unit square (a proxy for the actual texture map)
    - Map unit square point to point on texture

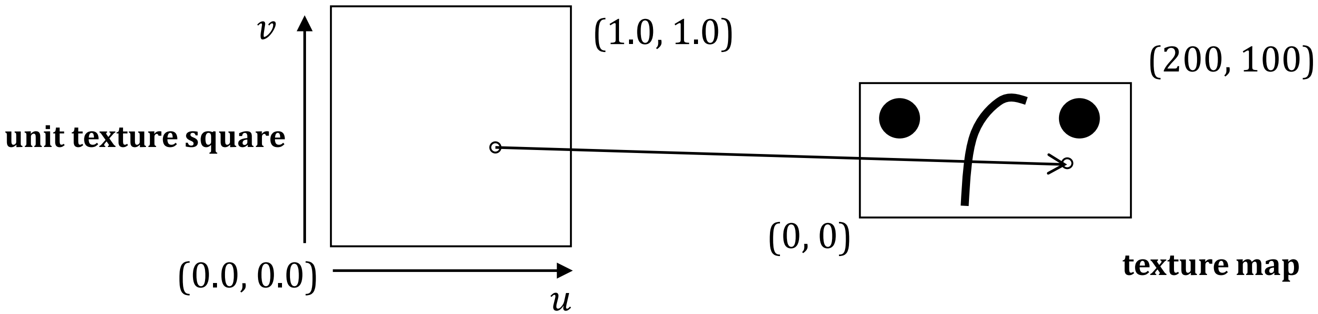


Van Gogh

  - Second mapping much easier, we'll cover it first – both maps based on proportionality
  - This 2D uv coordinate system is unrelated to the camera's 3D uvw coordinate system!
  - Here, the uv unit square is oriented with (0,0) in the bottom corner. It could have (0,0) in upper left; the choice is arbitrary. In Ray, use the latter.

Texture Mapping Technique (cont.)

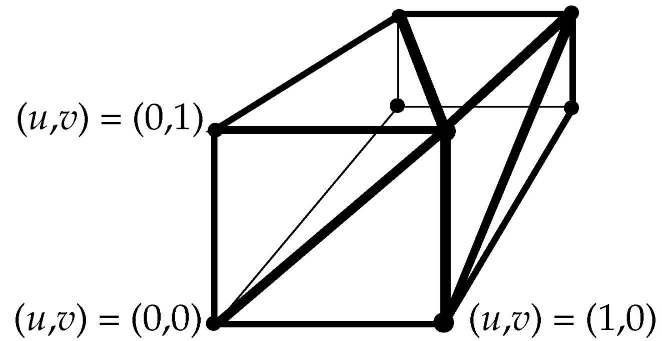
- Mapping a point (u, v) in unit square to a texture of arbitrary width w and height h:
  - Corresponding point on texture map is proportional on each axis

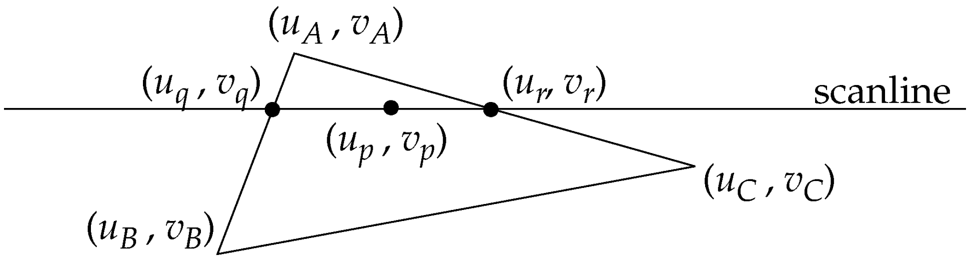


  - Above: (0.0, 0.0) → (0, 0); (1.0, 1.0) → (200, 100); (0.7, 0.45) → (140, 45)
  - Once you have coordinates for texture, just look up color of texture at these coordinates
  - Coordinates not always a discrete (int) point on texture as they are mapped from points in continuous uv space. May need to average neighboring texture pixels (i.e., filter)

Texture Mapping Individual Polygons

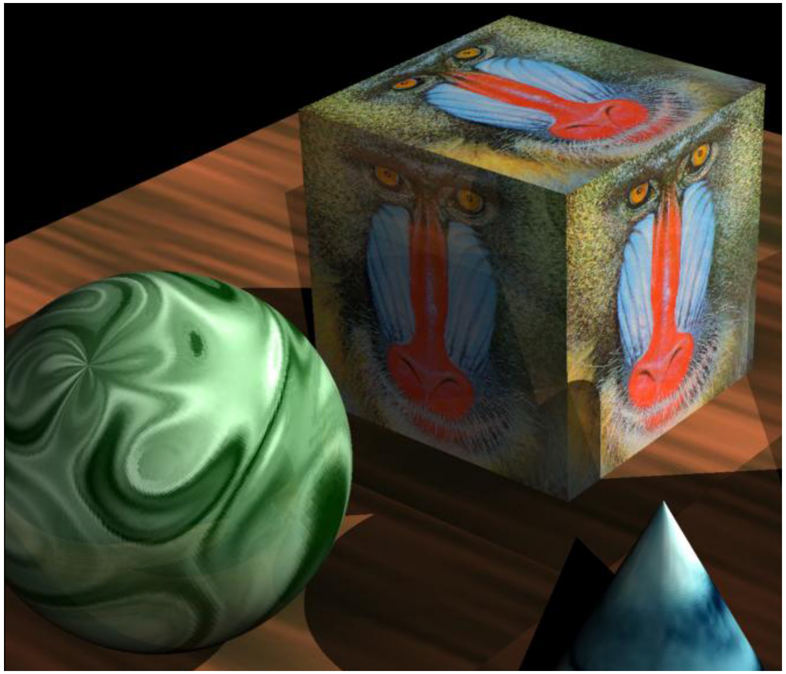
- (u, v) texture coordinates are pre-calculated and specified per vertex
- Vertices may have different texture coordinates for different faces
- Texture coordinates are linearly interpolated across polygon, as usual





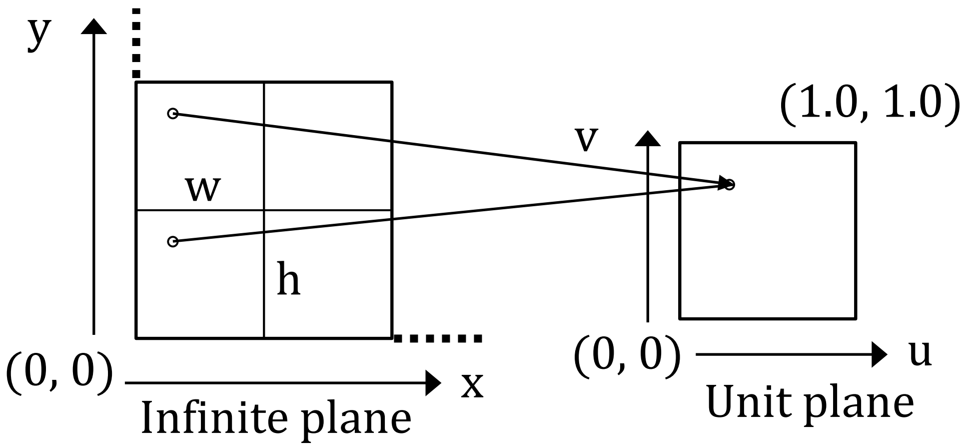
Mapping from point on object to (u, v) square

- Texture mapping in “Ray”: mapping solids
- Using ray tracing, get an intersection point (x, y, z) in object space
- Need to map this point to a point on the (u, v) unit square, so we can map that to a texture value
- Three easy cases: planes, cylinders, and spheres
- Easiest to compute the mapping from (x, y, z) coordinates in object space to (u, v)
- Can cause unwanted texture scaling (use filters!)
- Texture filtering is an option in most graphics libraries
- OpenGL allows you to choose filtering method
  - GL_NEAREST: Picks the nearest pixel in the texture
  - GL_LINEAR: Weighted average of the 4 nearest pixels



Texture Mapping Large Quads

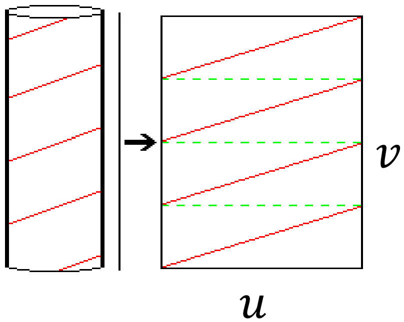
- How to map a point on a very large quad to a point on the unit square?
- Tiling: texture is repeated over and over across infinite plane
- Given coordinates (x, y) of a point on an arbitrarily large quad to tile with quads of size (w,h), the (u, v) coordinates on the unit square are:

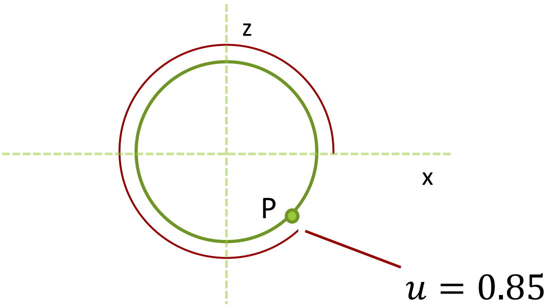


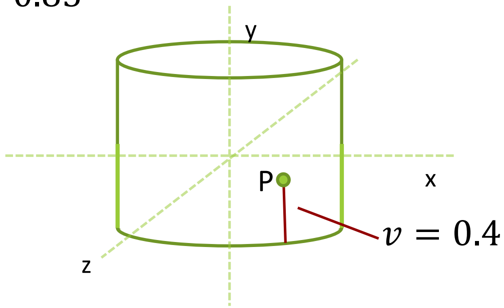
$$(u, v) = \left(\frac{(x \,\%\, w)}{w}, \frac{(y \,\%\, h)}{h}\right)$$

Texture Mapping Cylinders and Cones

- Given a point P on the surface:
  - If it's on one of the caps, map as though the cap is a plane
  - If it's on the curved surface:
    - Use position of point around perimeter to determine u
    - Use height of point to determine v
  - Mapping v is trivial: [-0.5, 0.5] for unit cylinder gets mapped to [0.0, 1.0] just by adding 0.5

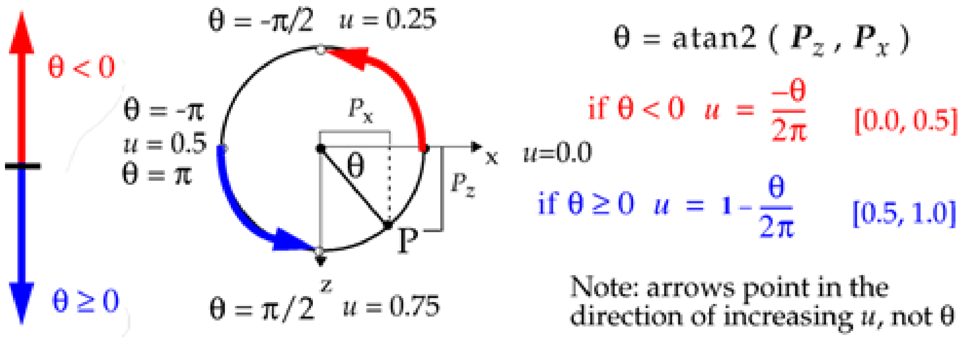






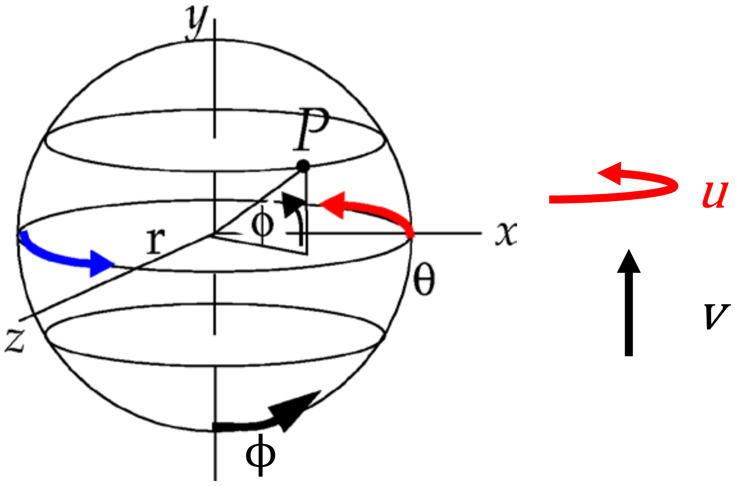
Computing u coordinate for cones and cylinders

- Must map all points on perimeter to [0, 1], going CCW in normal polar coordinate system(see arrows)
- Note where positive first quadrant is, based on *z* pointing down in top view of XYZ space
- Easiest way is to say $u = \frac{\theta}{2\pi}$, but computing θ can be tricky
- $\text{atan}(\frac{z}{x})$ yields $\theta \in (\frac{-\pi}{2}, \frac{\pi}{2})$, mapping two perimeter positions to the same θ value
  - Example: $\text{atan}(\frac{1}{1}) = \text{atan}(\frac{-1}{-1}) = \frac{\pi}{4}$
- $\text{atan2}(z, x)$ yields $\theta \in (-\pi, \pi)$
  - But isn't continuous -- see diagram
  - The 2 in atan2 just means 2nd form



Texture Mapping for Spheres

- Find (u, v) coordinates for P
- We compute u the same we do for cylinders and cones: distance around perimeter of circle
- At poles, v=0 or v=1, there is a singularity. Set u to some predefined value. (0.5 is good)
- v is a function of the latitude ϕ of P



$$\phi = \sin^{-1}\frac{P_y}{r}$$

r = radius

$$v = \frac{\phi}{\pi} + \frac{1}{2}$$

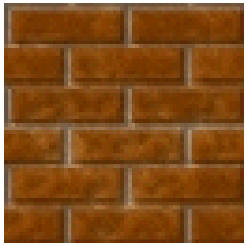
$$-\frac{\pi}{2} \leq \phi \leq \frac{\pi}{2}$$

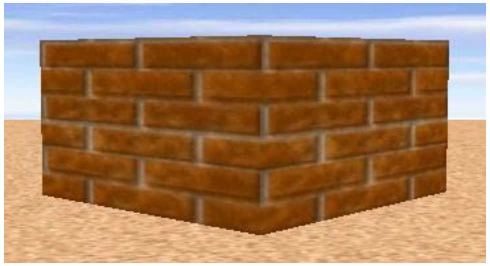
Texture Mapping Style

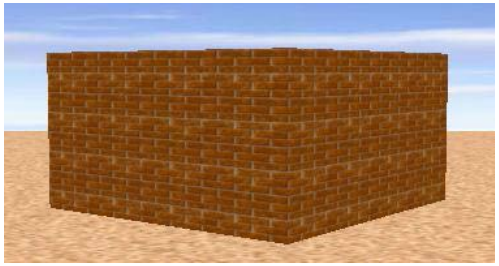
LECTURE 08: TEXTURE MAPPING

Tiling

- We want to create a brick wall with a brick pattern texture
- A brick pattern is very repetitive, we can use a small texture and “tile” it across the wall
- Tiling allows you to scale repetitive textures to make texture elements just the right size.


Texture


Without Tiling


With Tiling

Stretching

- With non-repetitive textures, we have less flexibility
- Have to fill an arbitrarily large object with a texture of finite size
- Can’t tile (will be noticeable), have to stretch instead
- Example, creating a sky backdrop:

Texture

Applied with stretching

Complex Geometry

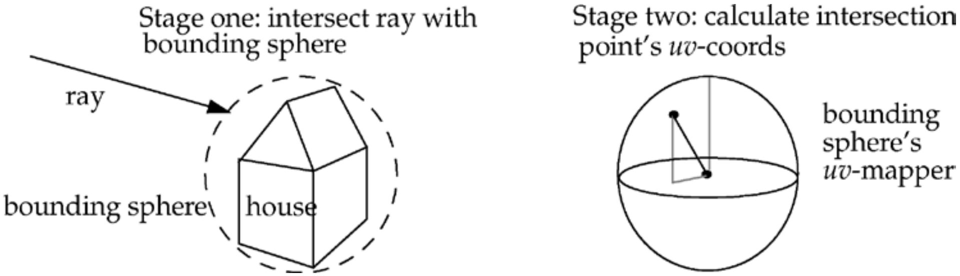
LECTURE 08: TEXTURE MAPPING

Texture Mapping Complex Geometry

- Sometimes, reducing objects to primitives for texture mapping doesn't achieve the right result.
  - Consider a simple house shape as an example
  - If we texture map it using polygons, we get discontinuities at some edges.
- Easy solution: Pretend object is a sphere and texture map using the sphere map

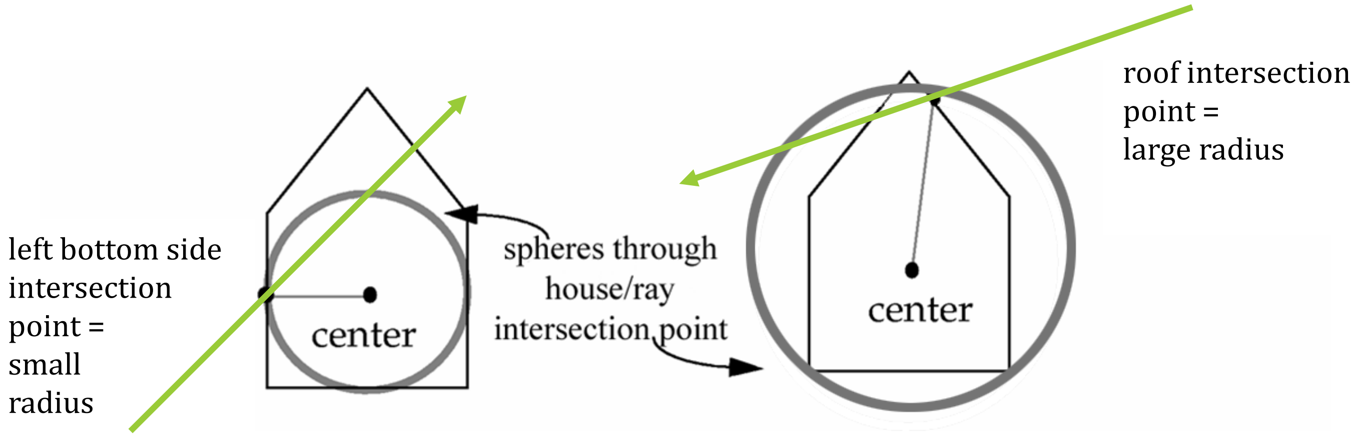
Texture Mapping Complex Geometry (cont.)

- Intuitive approach: Place a bounding sphere around the complex object
  - Find ray's object space intersection with bounding sphere
  - Convert to intersection point *uv*-coordinates
- Don't actually need to construct a bounding sphere
  - Once have intersection point with object, just treat it as though it were on a sphere passing through point. Same results, but different radii
  - This works because the *(u, v)* coordinates on a sphere don't depend on the sphere's radius



Texture Mapping Complex Geometry (cont.)

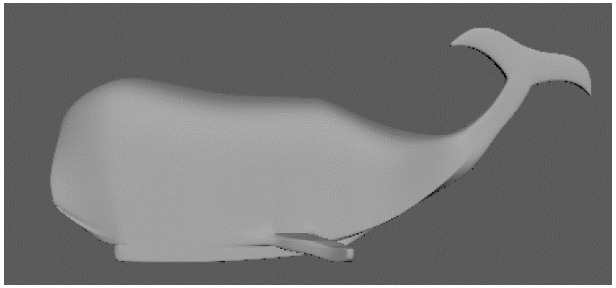
- When we treat the object intersection point as a point on a sphere passing through the point, our “sphere” will vary in radius

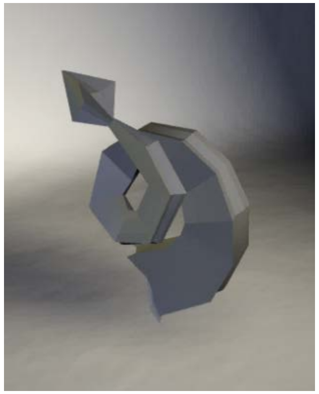


- But radius doesn’t affect (u, v) coordinates on a sphere
  - Only the angles matter (φ and θ in spherical coordinates)

Results

- Results of spherical (u, v) mapping on house:
  - Hey, that looks pretty good. Will it always work?
- For example, what if we want to put a texture on these objects?





Complex Geometry in Real Applications

- When texture mapping in videogames or films, objects will almost always be more complicated than primitives or that house shape
  - Common objects include humans, monsters, and other organic shapes
- You also want precise control over how the texture map looks on the object
  - Imagine texture mapping a human face with the eyes lined up wrong with the model geometry – viewers would definitely notice!
- Therefore, most cases of texture mapping in the “real world” of these industries are done using 3D modelling programs like Maya, Zbrush, Blender, etc.
  - Our examples are from Maya, but the technique would be similar in the other programs

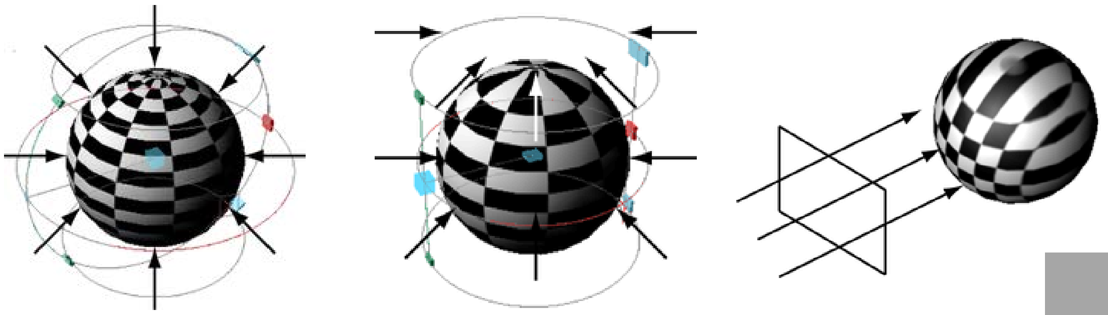
Real Application - Examples

- Here's a very compressed overview of the process:

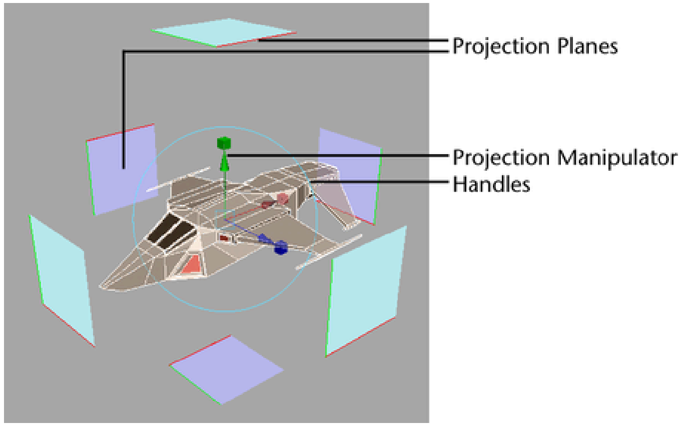
- Ultimately, the goal is to make every face on the object correspond to a section of the (0,0) to (1,1) (u,v) space

Real Application – Examples (cont.)

- The main difficulty still lies in generating that mapping
- In addition to the spherical mapping we covered previously (left), in Maya, you can also do cylindrical (middle) or planar mapping (right) when texture mapping objects

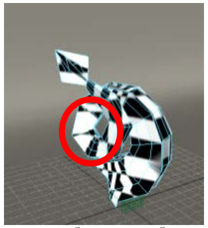


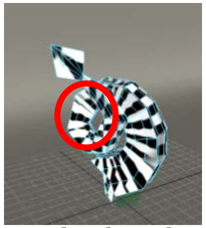
- Maya also offers an “Automatic” mapping
  - Uses multiple projection planes
- Each mapping has drawbacks

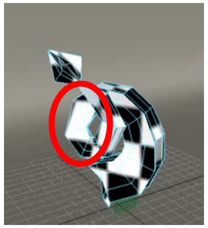


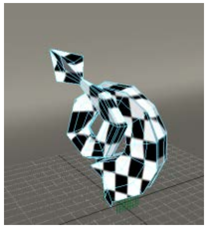
Real Application – Examples (cont.)

- Testing with a checkerboard pattern is useful when looking for problems with (u, v) mappings.
  - The goal is to minimize uneven distortion of the pattern.
- Spherical, cylindrical, and automatic have a lot of distortion on this twisty object.
  - Red circles show uneven checkers on all these mappings – bad!
- Planar is okay when viewed from one axis, but the (u, v) map overlaps itself and two axes are ignored.
  - This leads to distortion when viewing from the other axes.

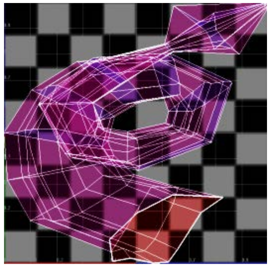

Spherical


Cylindrical

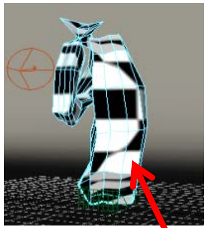

Automatic


Planar

=



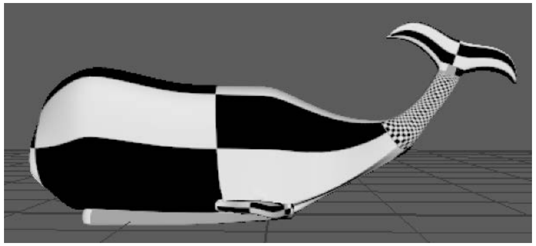
=

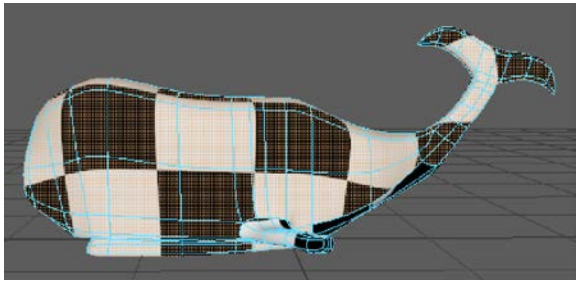

Uh-oh!

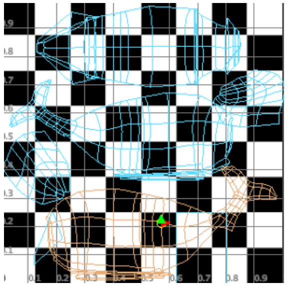
What the planar uv map looks like "unwrapped."
Pink = overlapping squares

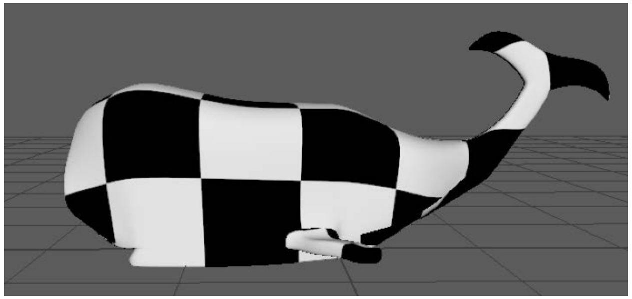
Real Application – Examples (cont.)

- Maya will give you UV coordinates automatically
  - Most times these UVs aren't quite what we want or look distorted.
- Usually, we need to go in and modify the UVs to get something that we are happy with.
  - We do this in Maya by selecting faces to be part of a UV "shell". We can cut and sew shells as needed.
- Once we get the UV map right, we'll see that the checkerboard is much less distorted
  - It's hard to get the UVs totally perfect. Oftentimes, we can hide some of the problems by putting seams on the bottom or other parts that won't be as visible.



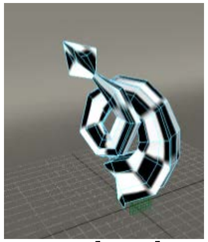






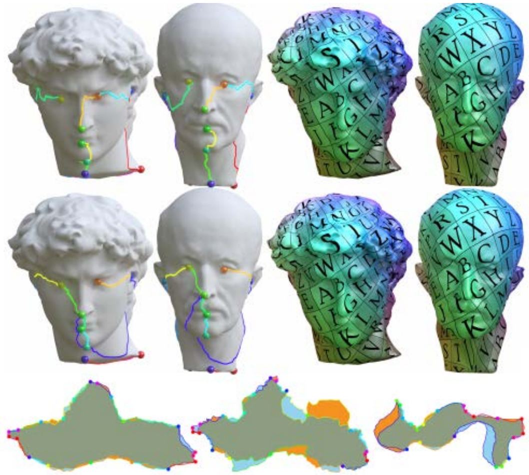
Real Application – Examples (cont.)

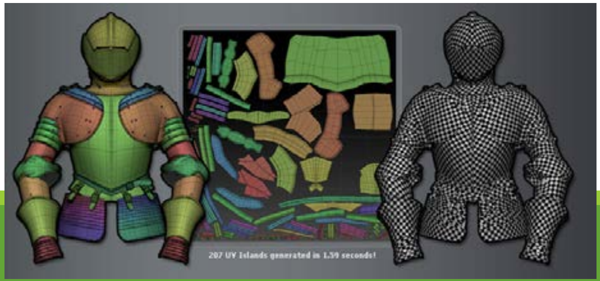
- There is no good solution
- To get the look they want, the modelers will often have to go in and manually cut and sew edges in the (u, v) maps
- However, computers are getting better– there are several complex techniques for making texture maps that look seamless
- Other programs try to generate maps that put the discontinuities in places where the real objects would have seams.



Handmade

=





Bump Mapping

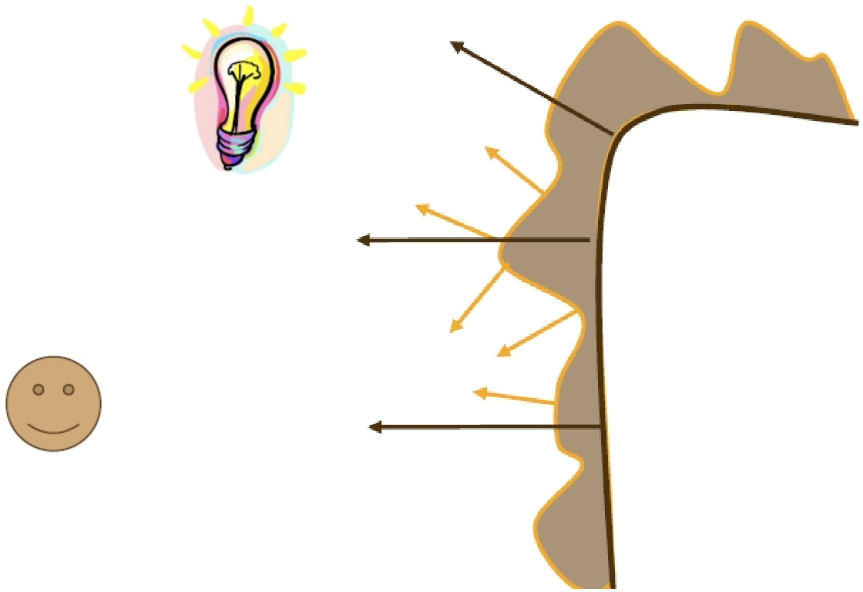
LECTURE 08: TEXTURE MAPPING

What is Bump Mapping?

- “Real” texture - Many textures are the result of small perturbations in the surface geometry
- Modeling these changes would result in an explosion in the number of geometric primitives.
- Bump mapping attempts to alter the lighting across a polygon to provide the illusion of texture.

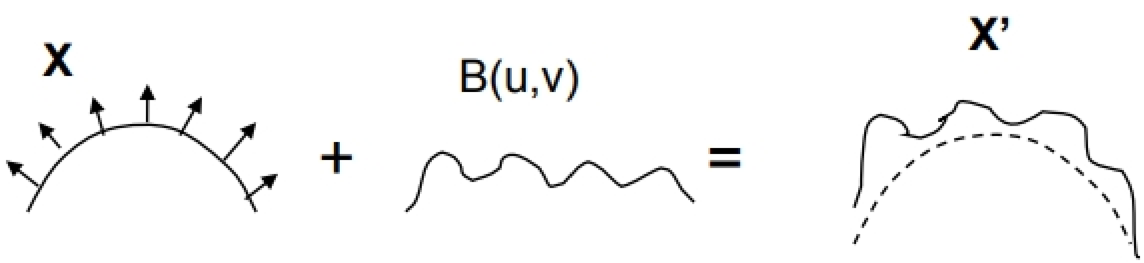
What is Bump Mapping (cont.)

- Consider the lighting for a modeled surface
- We can model this as deviations from some base surface.
- The question is then how these deviations change the lighting



Contracting a Bump Map

- Assumption: small deviations in the normal direction to the surface
  - $\overrightarrow{x'} = \vec{x} + B\vec{N}$
- Where B is defined as a 2D function parameterized over the surface
  - B = f(u,v)

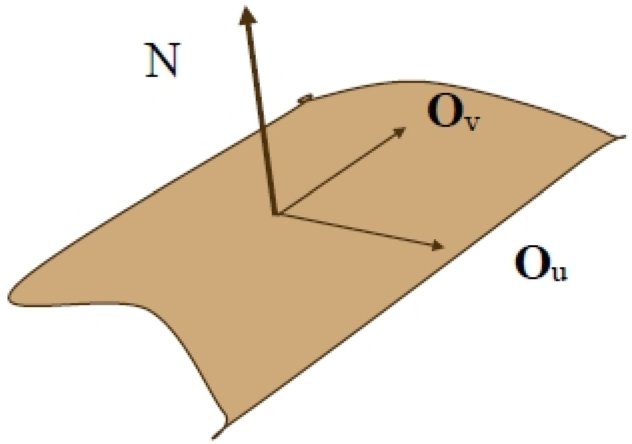


Contracting a Bump Map (cont.)

- Step 1: Putting everything into the same coordinate frame as B(u,v)
  - x(u,v), y(u,v), z(u,v) – this is given for parametric surfaces, but easy to derive for other analytical surfaces
  - Or $O(u,v) = [x(u,v), y(u,v), z(u,v)]^T$

The Original Normal

- Define the tangent plane to the surface at a point (u,v) by using the two vectors O_u and O_v
- Analytic derivatives or, you can compute them using central difference:
  - $O_u = (O(u+1,v) - O(u-1,v)) /2$
  - $O_v = (O(u,v+1) - O(u,v-1)) /2$
- The normal is then given by
  - $N = O_u \times O_v$

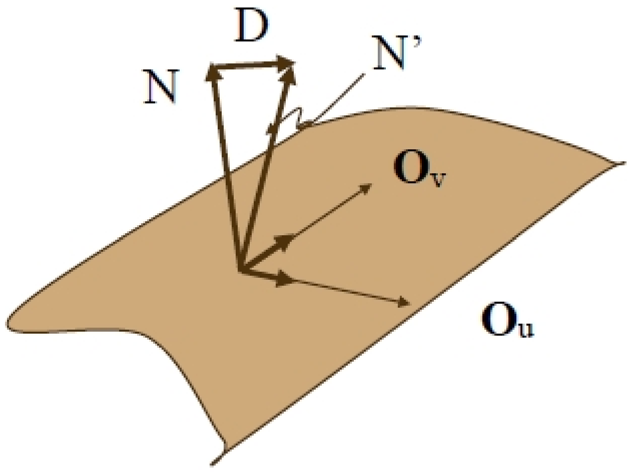


New Surface Positions

- The new surface positions are then given by:
  - $O'(u,v) = O(u,v) + B(u,v)\ N$
  - Where, $N = N\ /\ |N|$
- Differentiating leads to:
  - $O'_u = O_u + B_u\ N + B\ (N)_u \approx O_u + B_u\ N$
  - $O'_v = O_v + B_v\ N + B\ (N)_v \approx O_v + B_v\ N$
  - If B is small

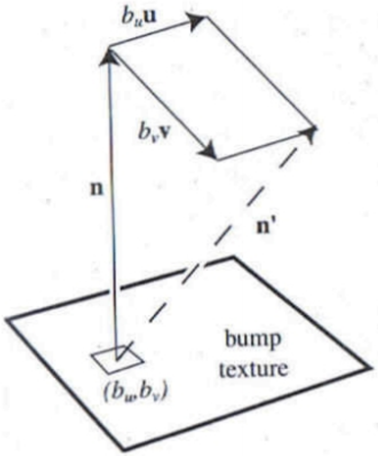
The New Normal

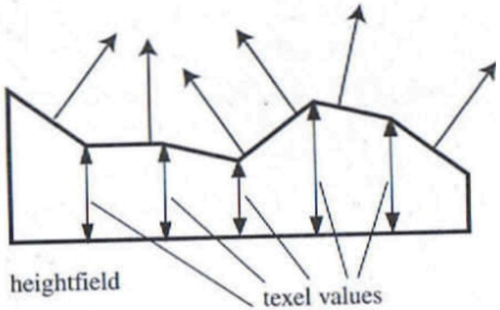
- This leads to a new normal:
  - $N'(u,v) = O_u \times O_v + B_u(N \times O_v) - B_v(N \times O_u) + B_u\, B_v(N \times N)$
  - $= N + B_u(N \times O_v) - B_v(N \times O_u)$
  - $= N + D$



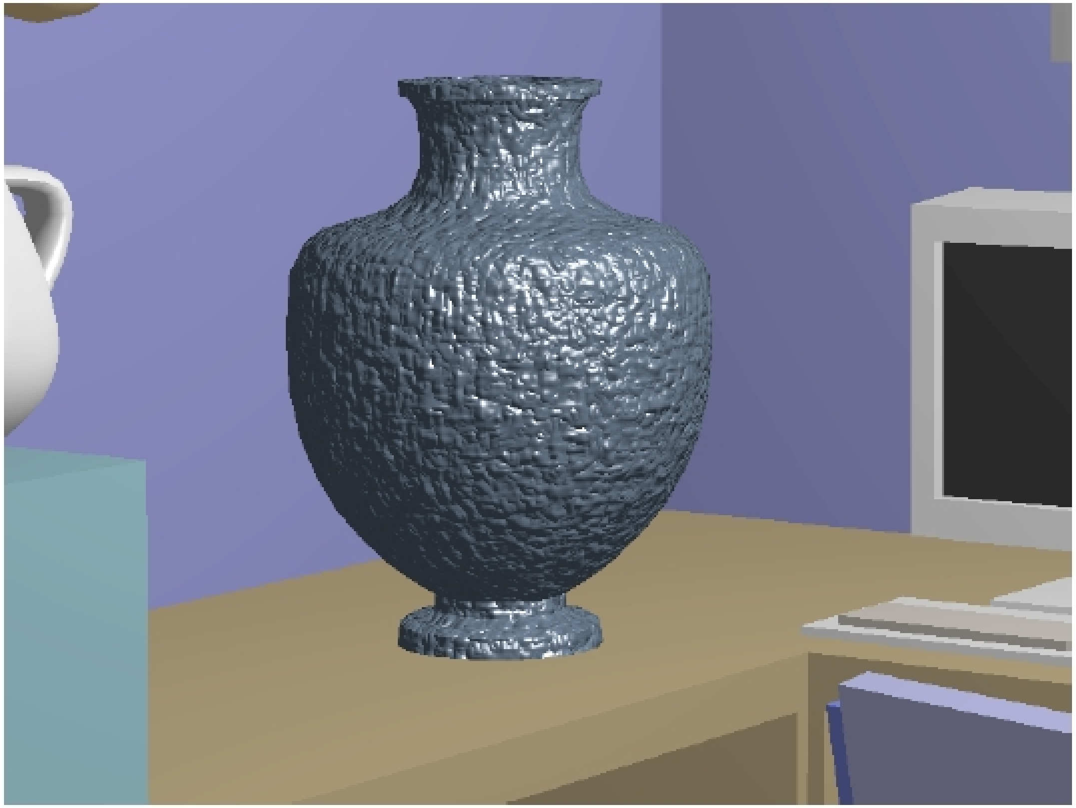
Bump Map Representation

- For efficiency, we can store B_u and B_v in a 2-component texture map
  - This is commonly called a offset vector map
  - Note: B_u and B_v are oriented in tangent-space
- B_u and B_v are used to modify the normal N
- Another way is to represent the bump as a high field
  - The high field can be used to derive B_u and B_v (using central difference)

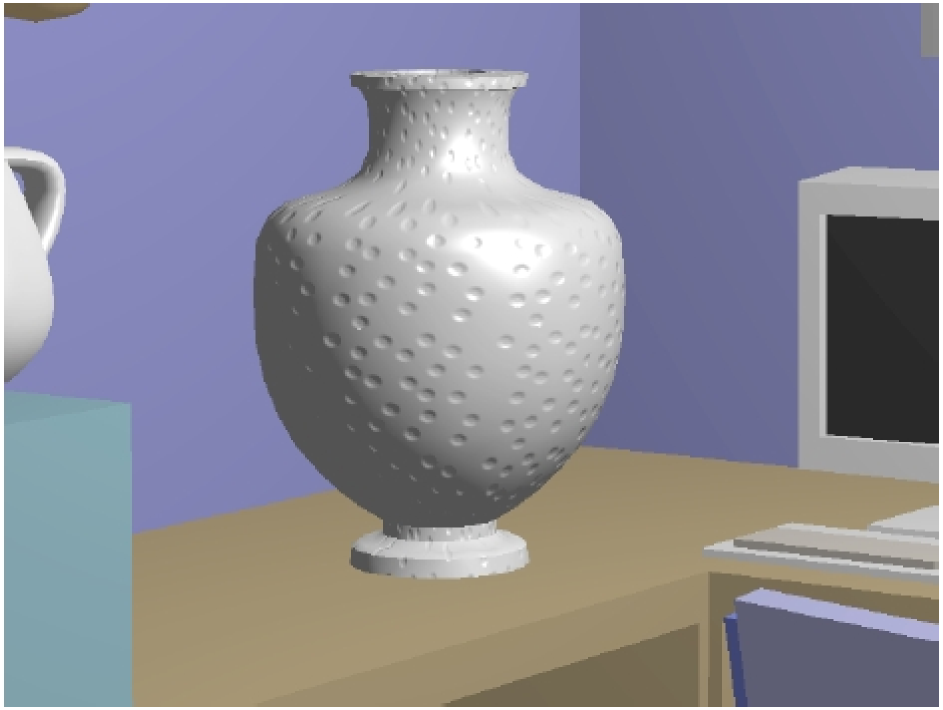




Procedurally Bump Mapped Object

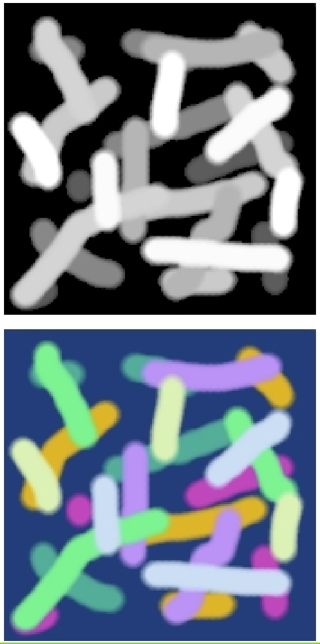


Bump mapped based on a cylindrical texture space



Bump Map and Texture Map

- Bump mapping is often combined with texture mapping
- The picture below is a bump map has been used to (apparently) perturb the surface and a coincident texture map to colour the 'bump objects'

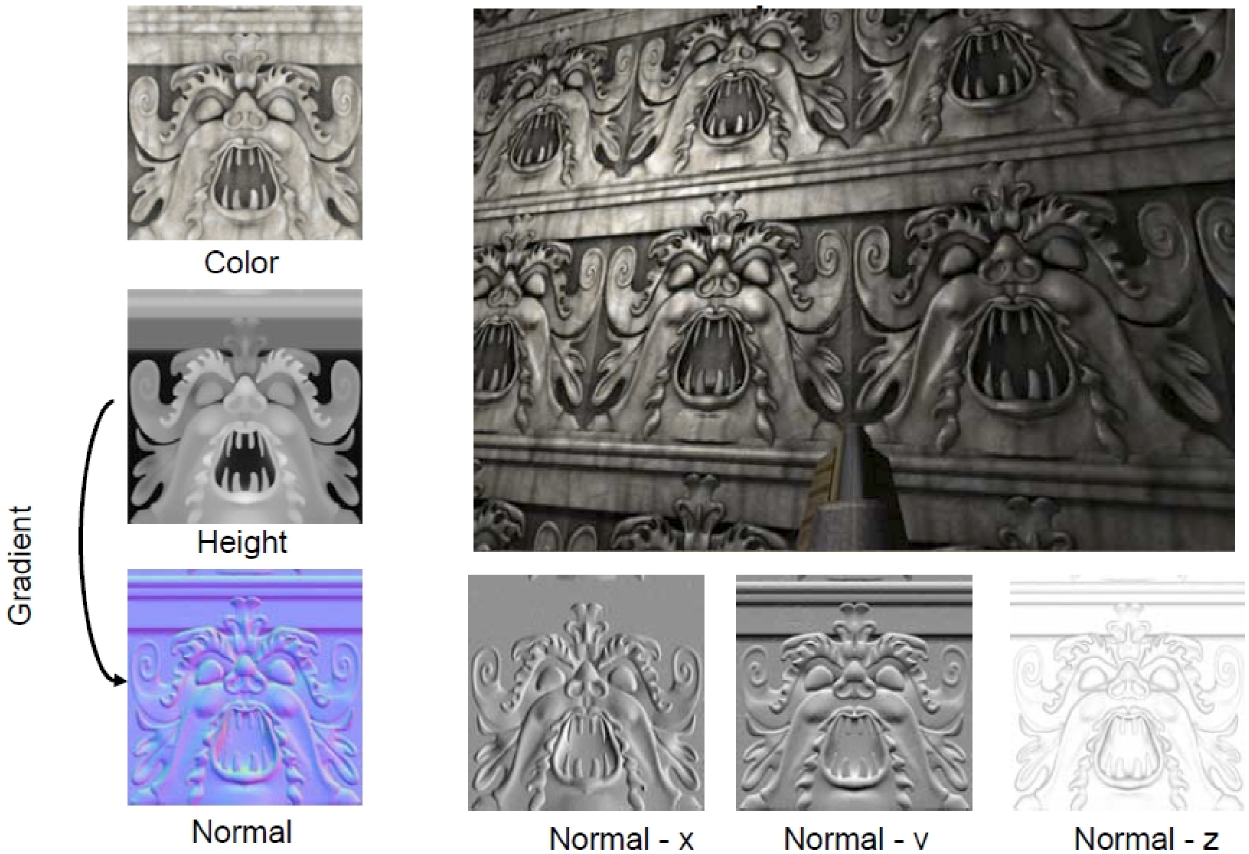


Bump Mapping and Texture Mapping on Text

Normal Map

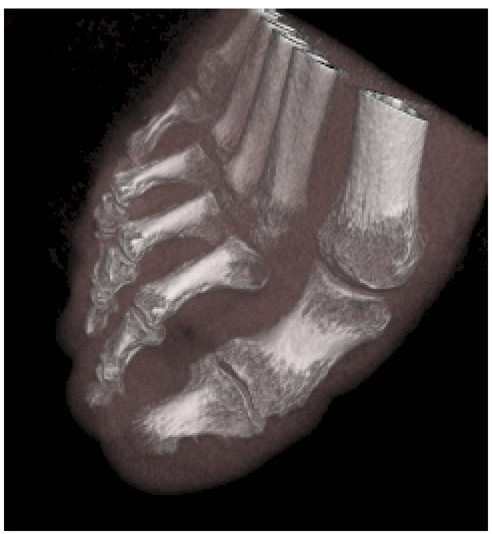
- Pre-computation of modified normal vector N’
- Stored in texture (RGB)=(N_x, N_y, N_z)
- Illumination computation per pixel
  - For example in fragment program
  - Per-vertex light vector (toward light source) is interpolated

Normal Map Example

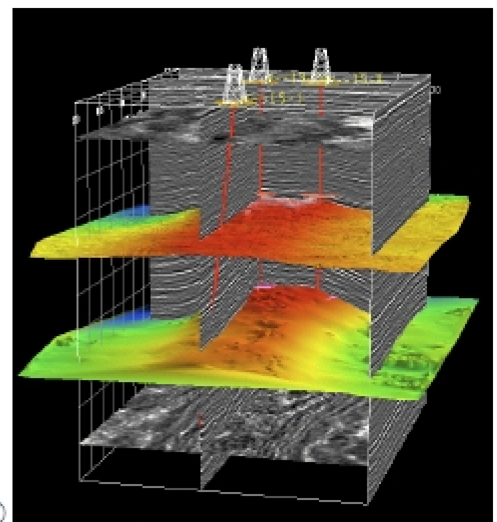


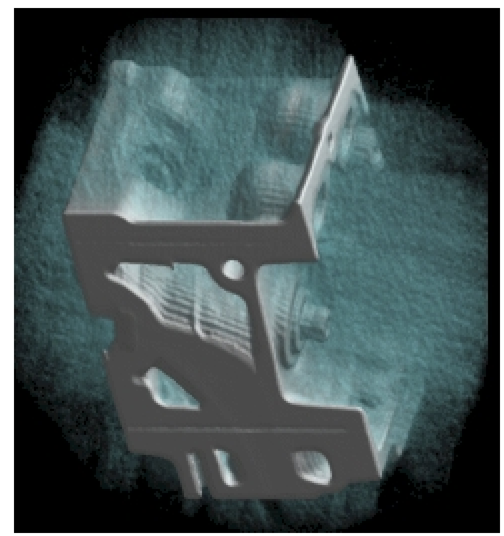
3D Textures

- Representation on 3D domain
- Often used for volume representation and rendering
  - Texture = uniform grid



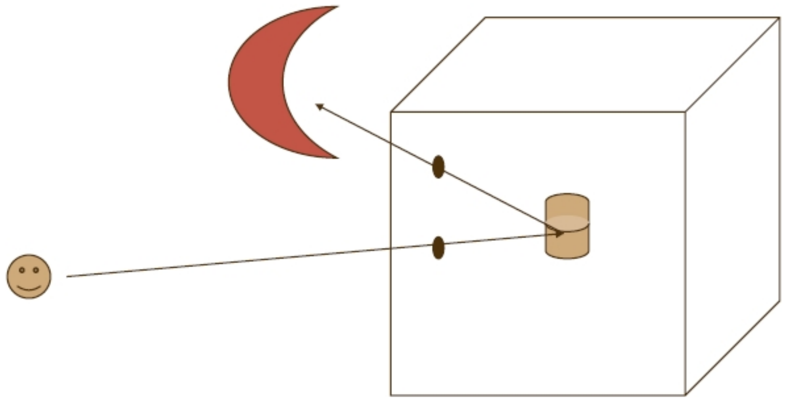
©





Environment Mapping

- Used to show the reflected colors in shiny objects

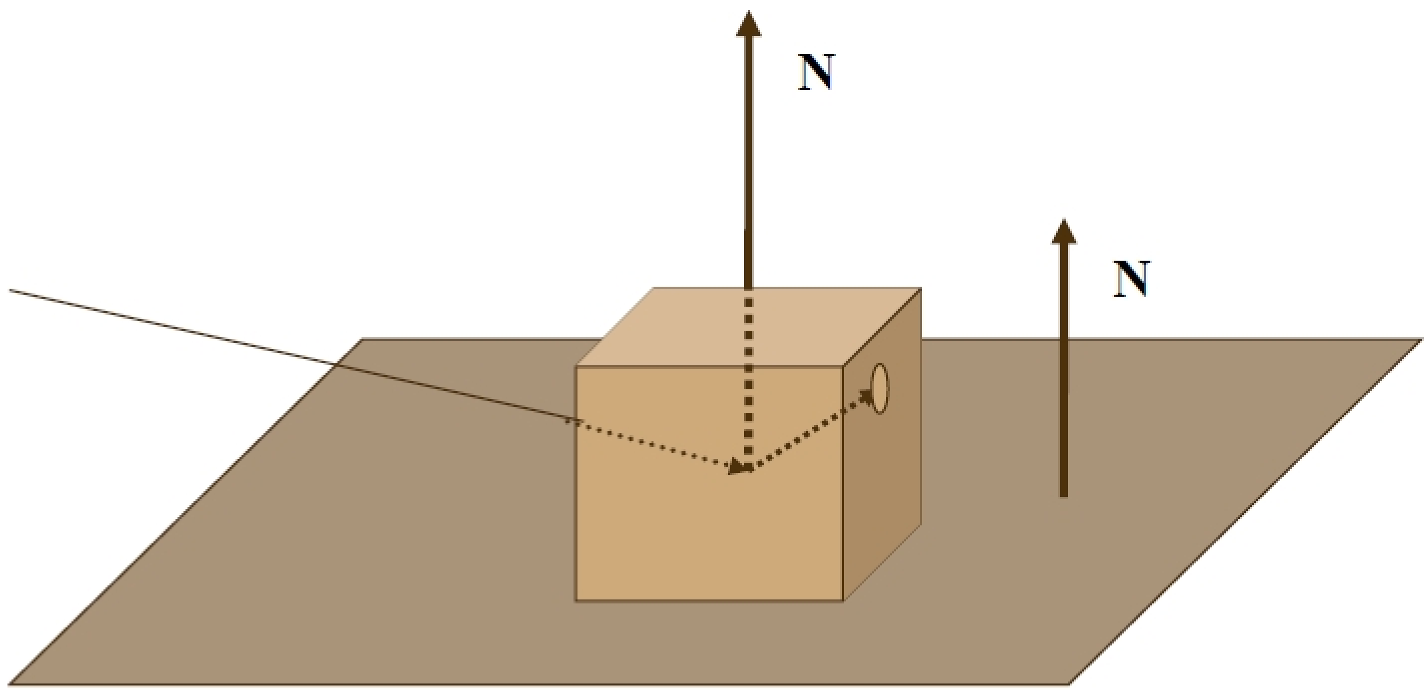


Environment Mapping (cont.)

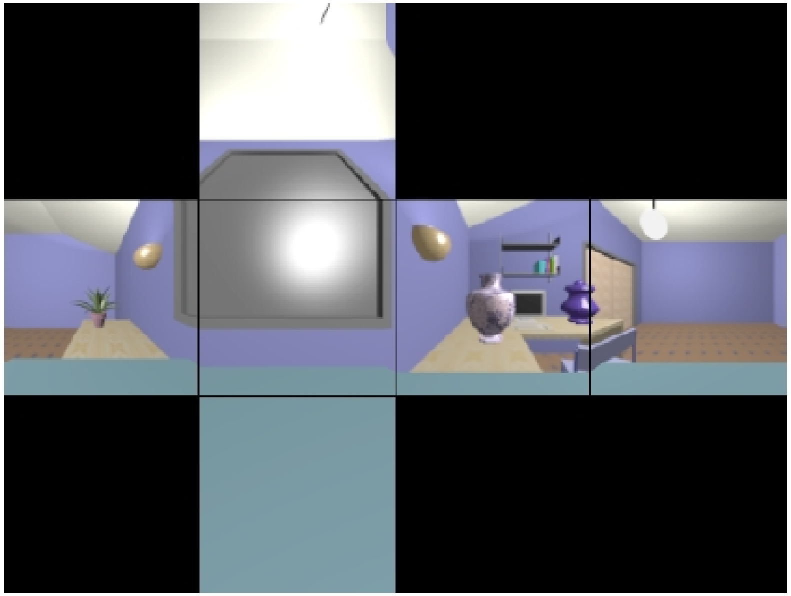
- Create six views from the shiny object's centroid
- When scan-converting the object, index into the appropriate view and pixel
- Use reflection vector to index
- Largest component of reflection vector will determine the face

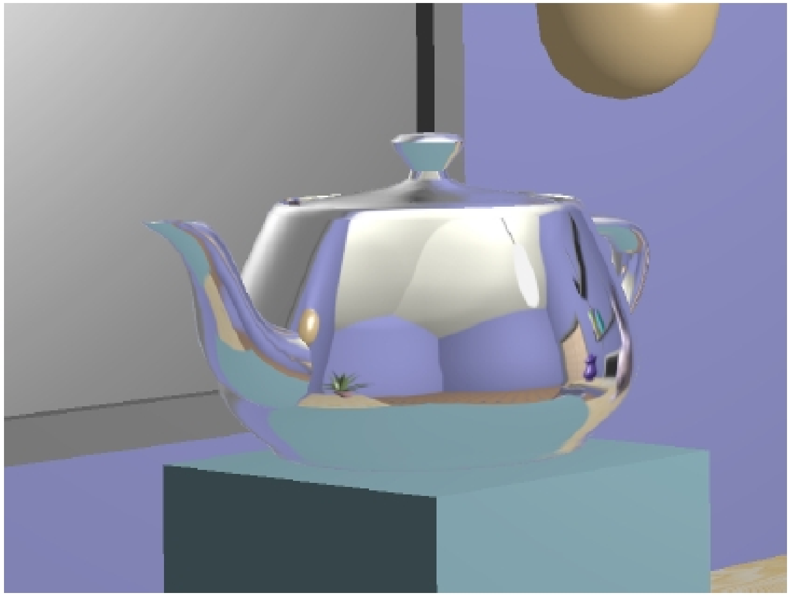
Environment Mapping Problems

- Reflection is about object's centroid
  - Okay for small objects and distant reflections

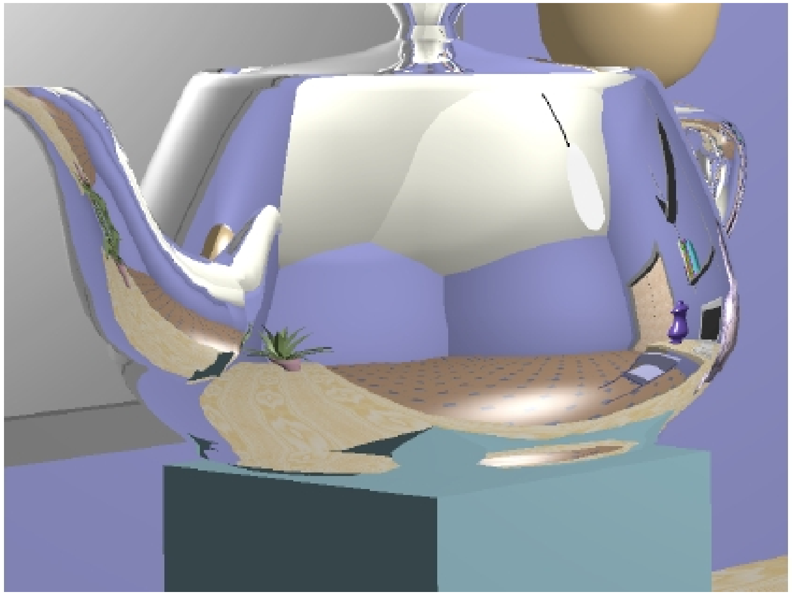


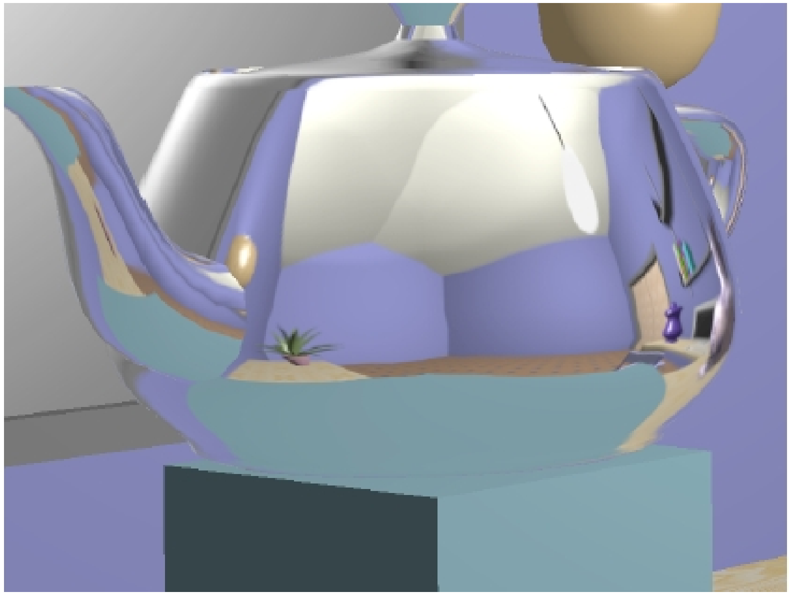
Environment Mapping Problems





Which one is ray-traced?



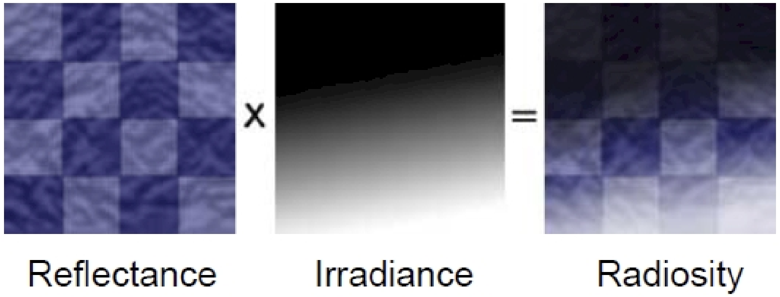


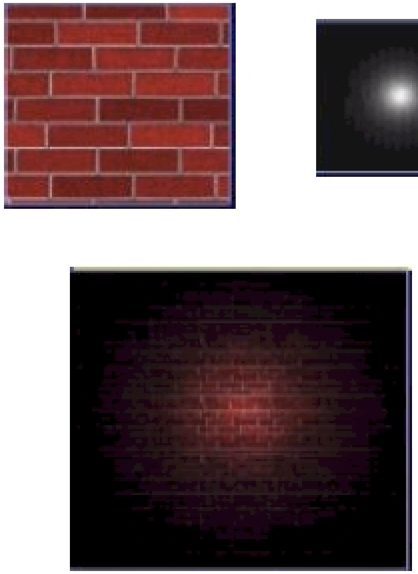
Light Maps

- Precompute the light in the scene
- Typically works only for view-independent light (diffuse light)
- Combine (texture-map) these light maps onto the polygon

Light Maps (cont.)

- Combination:
  - Structural texture
  - Light texture
- Light maps for diffuse reflection
  - Only Luminance channel
  - Low resolution is sufficient
  - Packing in “large” 2D texture





Combination with Textured Scene

Example: Moving Spotlight