Announcements

Written Assignment 2 is out – see the web page

Texture and other Mappings

Shadows
Texture Mapping
Bump Mapping
Displacement Mapping
Environment Mapping

Watt Chapter 8

COMPUTER GRAPHICS

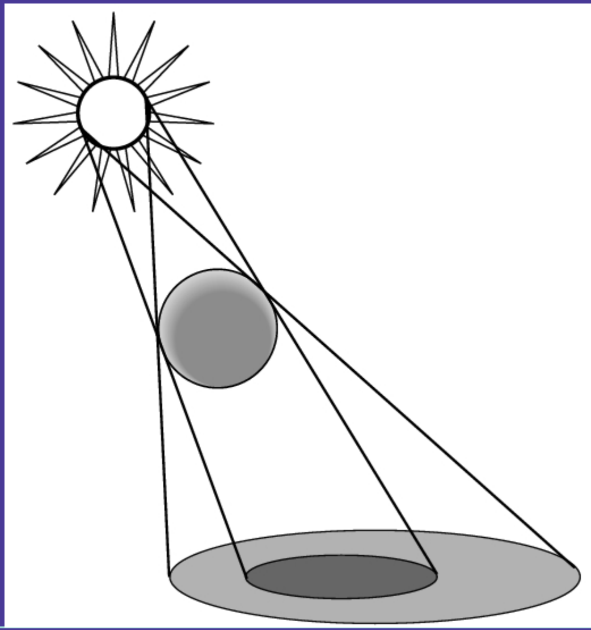
15-462

10/08/01

Shadows

Shadows occur where objects are hidden from a light source

- Omit any intensity contribution from hidden light sources
- Umbra and penumbra (function of size of light source)
- Soft shadows and hard shadows (point light source at a distance)
- Important perceptual clue for connecting objects to ground (feet in particular)
- But object-to-object shadows also important
- And in general have shadows from many light sources



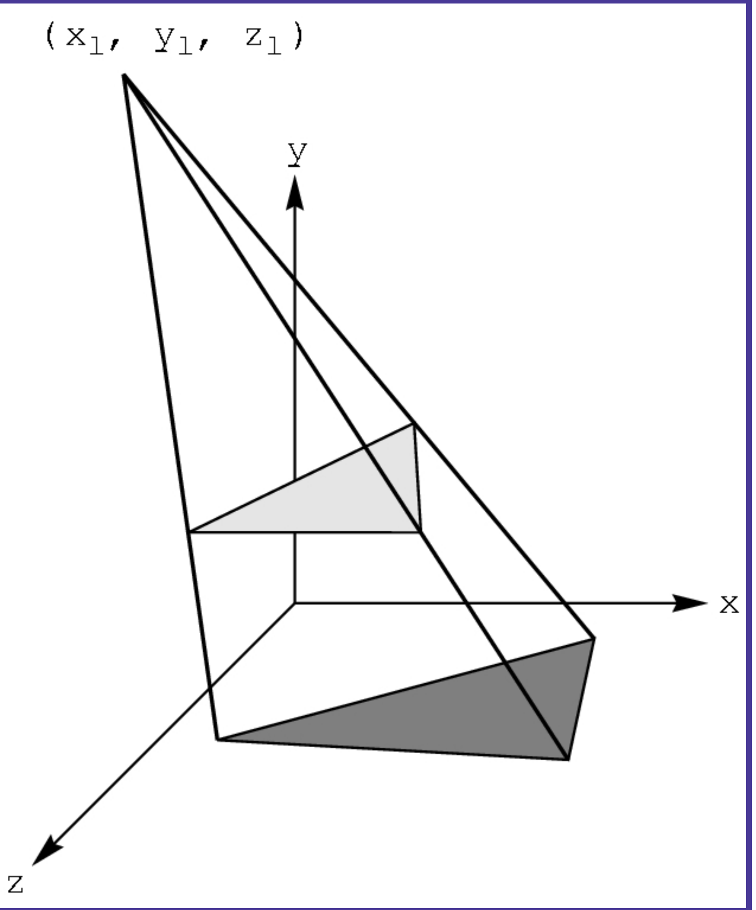
Shadows

Simplest case:

Point light source (no penumbra) and shadows only on the ground plane

Good for flight simulators

Shadow is projection of polygon onto the surface with the center of projection at the light source



Shadows

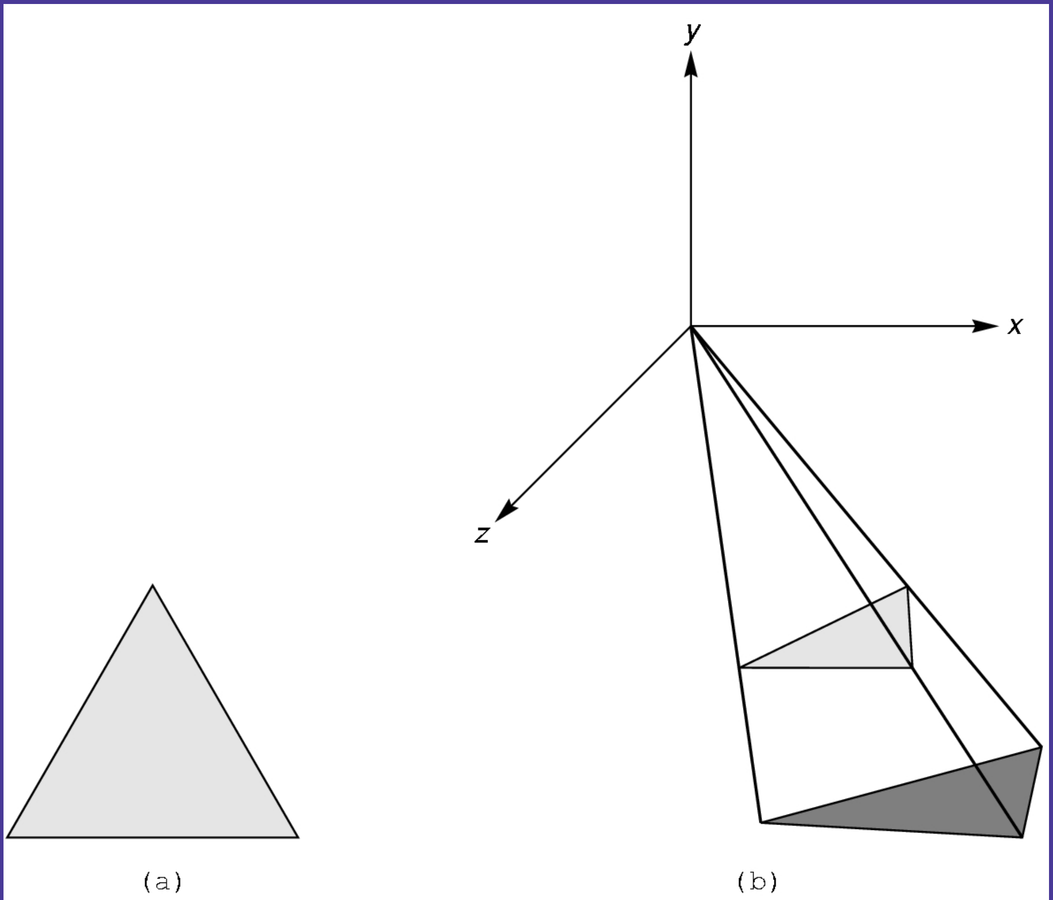
Put light source at origin with $T(-x_1, -y_1, -z_1)$

Simple perspective projection through the origin:

$$M = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1/y_1 & 0 & 0 \end{bmatrix}$$

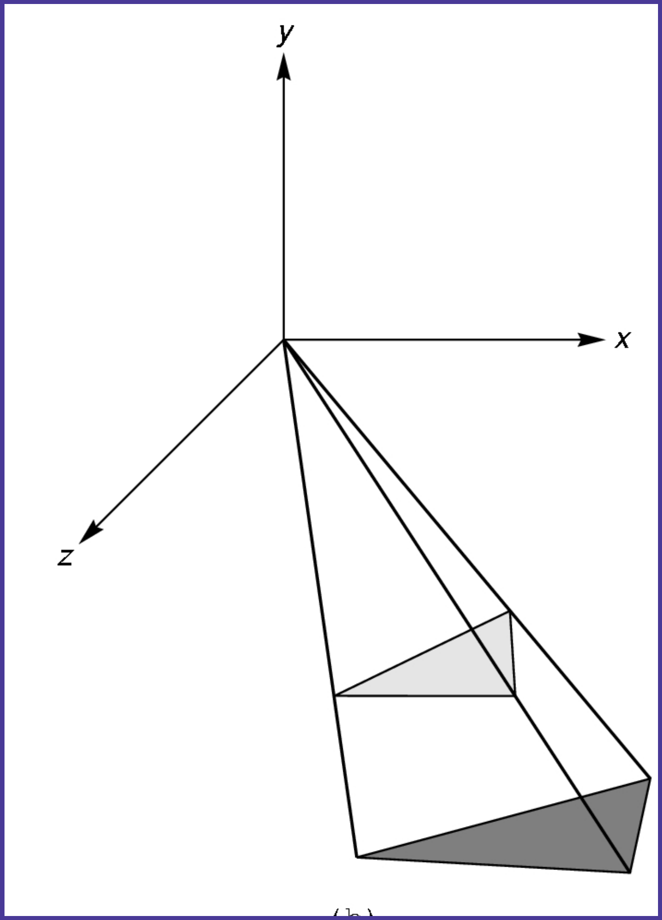
Translate back with

$$T(x_1, y_1, z_1)$$



Or in OpenGL

```
GlFloat m[16]; /* shadow projection matrix */
For (i=0;i<15;i++) m[i]=0.0;
m[0] = m[5] =m[10] = 1.0; m[7] = -1.0/y1;
glBegin(GL_POLYGON);
/* draw the polygon normally */
glEnd();
glMatrixMode(GL_MODELVIEW);
glPushMatrix(); /* save state*/
glTranslatef(x1,y1,z1); /* translate back */
glMultMatrixf(m); /* project*/
glTranslate(-x1,-y1,-z1); /* move light to origin */
glColor3fv(shadow_color);
glBegin(GL_POLYGON);
/* draw the polygon again */
glEnd();
glPopMatrix(); /* restore state *
```



Shadows

If a perspective transform does a point light source, what does a directional light source?

Shadows

But shadow maps only do shadows on the ground plane and the objects have to be far enough apart to not cast shadows on each other (although your eye isn't particularly good at picking up these problems).

Z-buffer Algorithm:

Render from light source to compute depth map (z distance to closest object for each pixel in the map).

While rendering, if a point (x,y,z) is visible, map (x,y,z) in the coordinates of the viewpoint to (x',y',z'), the coordinates of the point from the light. If z' is greater than the value in the z-buffer for that point, then a surface is nearer to the light source than the point under consideration and the point is in shadow. If so, render with a shadow intensity, if not render as normal.

Handles multiple light sources (with multiple z-buffers), moving objects and lights (at the cost of several renderings). Clearly a winning strategy with hardware.

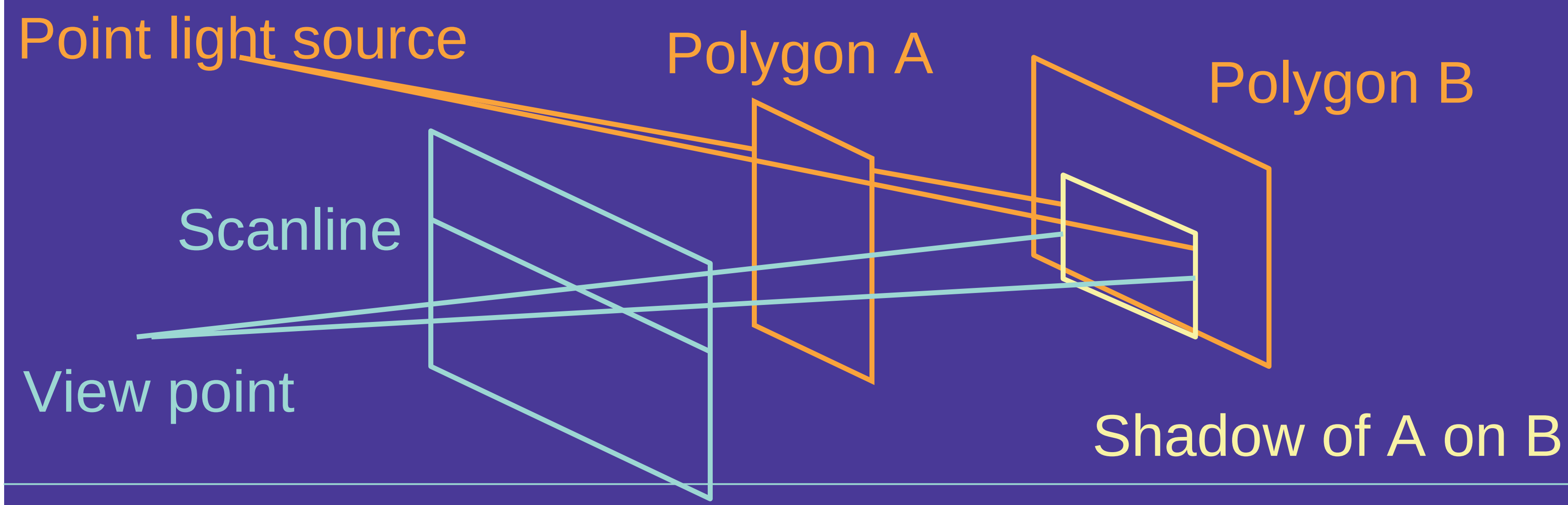
Other Shadow Algorithms

Projecting polygons—scan line (Appel 1968)

Build a data structure that links all polygons where one might shadow the other (depends on polygon location and lighting source location—animation would require recalculating). Multiple data structures for multiple lights.

When scan converting have three possibilities

- Shadow polygon does not cover generated scan line segment
- Shadow polygon completely covers segment (adjust intensity for shadow)
- Shadow polygon partially covers segment, subdivide and repeat

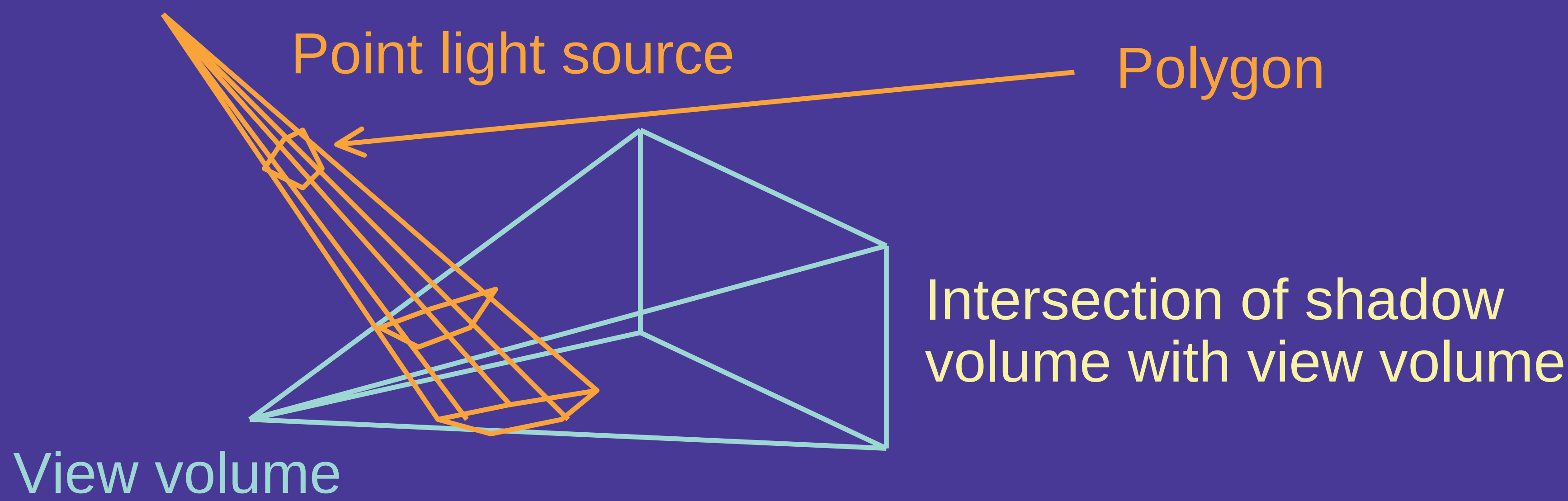


Other Shadow Algorithms

Shadow Volumes (Crow 1977)

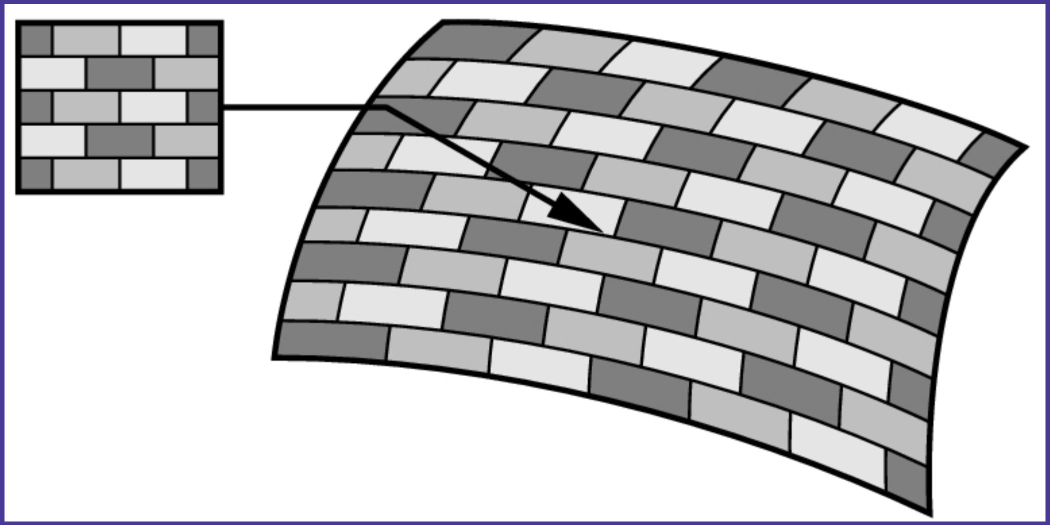
Compute the volume of space swept out by an object, intersected with the viewing volume.

Include these invisible objects in rendering pipeline.
Render visible objects as in shadow if they are in front of a back facing shadow polygon and in back of a front facing polygon. Extend the z buffer to contain this extra information.

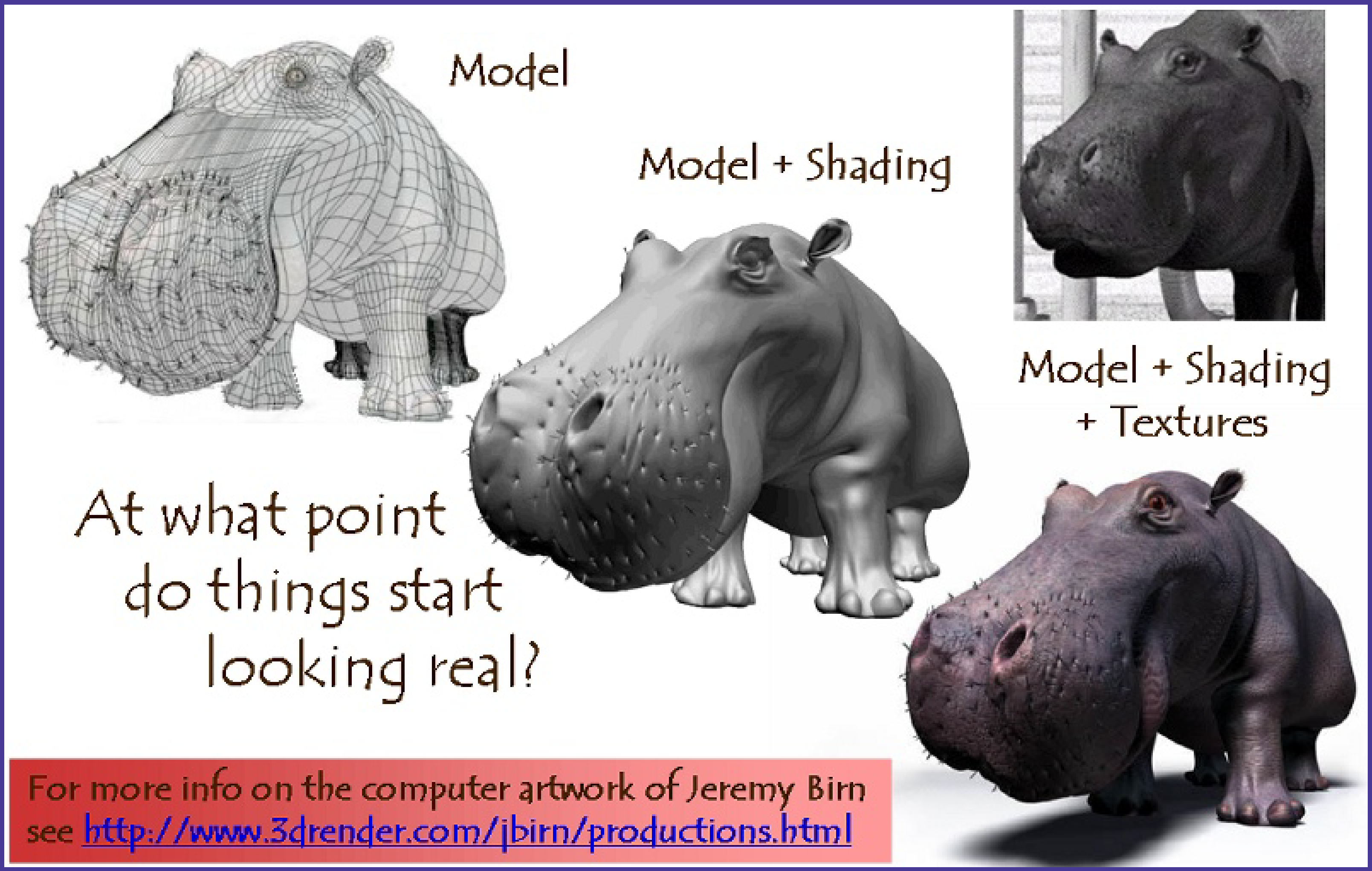


Last time we talked about shading. But uniformly colored or shaded surfaces are unrealistic.

- Real objects have small surface features
- One option: use a huge number of polygons with appropriate surface coloring and reflectance characteristics
- Another option: use a mapping algorithm to modify the shading algorithm
  - – Texture mapping
  - – Bump mapping
  - – Displacement mapping
  - – Environmental mapping



The Quest for Visual Realism



2D Texture Mapping

Texture images to make our surfaces more life-like

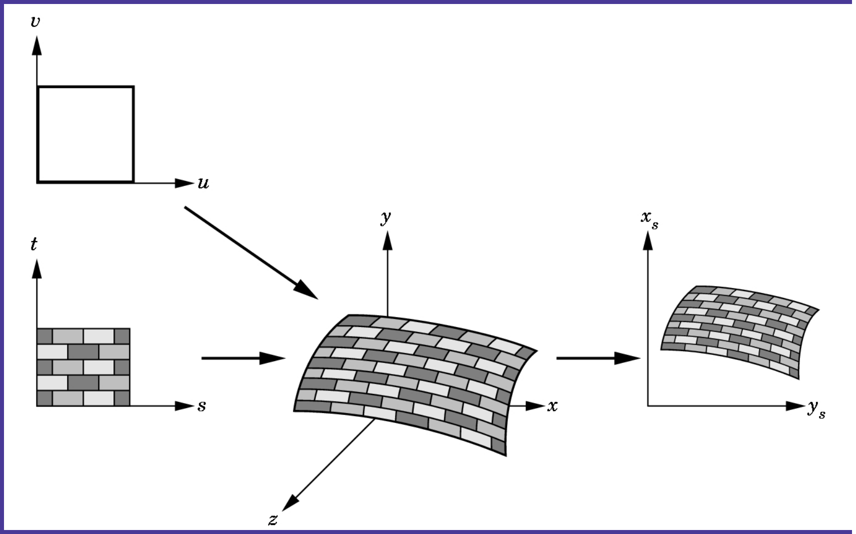
- Scan textures from the world (clouds, wood grain) or paint them yourself
- Store the texture in a 2D image
- Map the image onto the surface by a function which maps *(u,v)* coordinates of our parametric surface onto *(x,y)* image coordinates
- When shading a surface point, we look up the appropriate pixel from the 2D image, and use that to affect the final color

Voila! Your favorite picture painted onto a donut.

This technique is called

parametric texture mapping

But how to map from texture coordinates to object coordinates? Easy for a parametric surface, less obvious for other models.



Texture Mapping: General

Texture Space | Object Space | Screen Space

$$T(u)$$

$$T(u,v) \qquad (x_w, y_w, z_w) \longrightarrow (x_s, y_s)$$

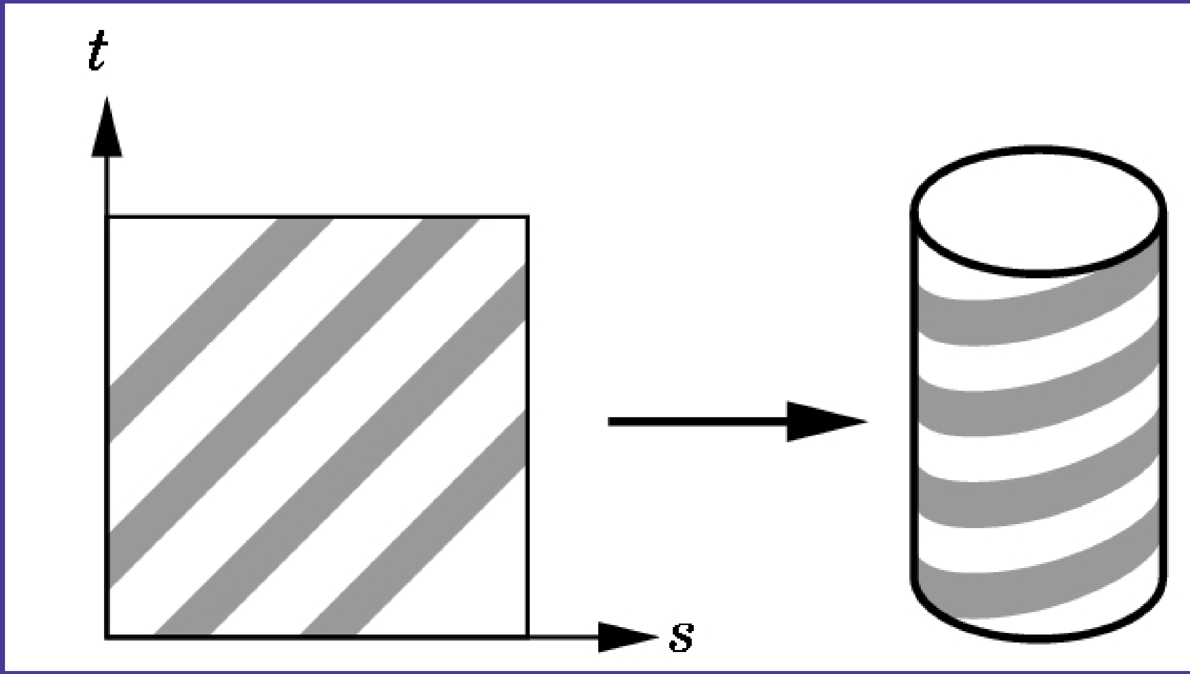
$$T(u,v,w)$$

Specifying the Mapping Function

Some objects have natural parameterizations:

– Sphere: use spherical coordinates $(\phi,\theta)=(2\pi u,\pi v)$

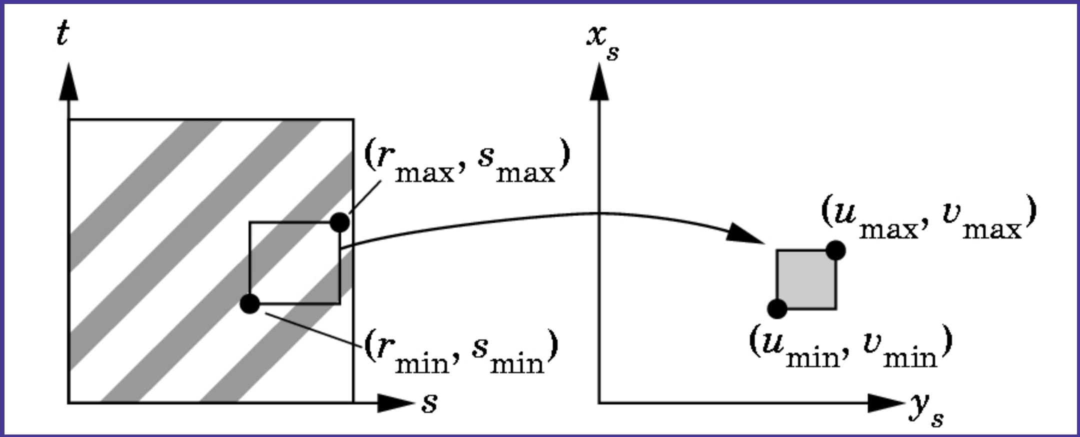
– Cylinder: use cylindrical coordinates $(u,\theta)=(u,2\pi v)$



Specifying the Mapping Function

Some objects have natural parameterizations:

– Parametric surface (such as B-spline or Bezier): use patch parameters *(u,v)*



$$u = u_{min} + \frac{s - s_{min}}{s_{max} - s_{min}}(u_{max} - u_{min})$$

$$v = v_{min} + \frac{t - t_{min}}{t_{max} - t_{min}}(v_{max} - v_{min})$$

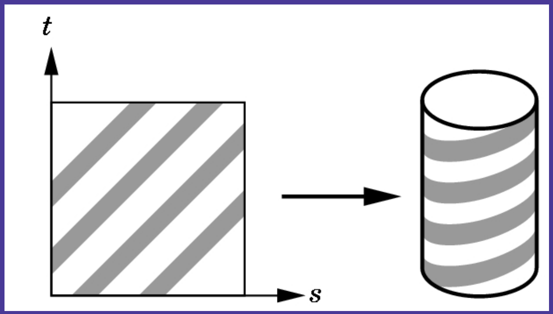
Doesn't take into account the curvature of the surface: stretching. Just like with roller coaster.

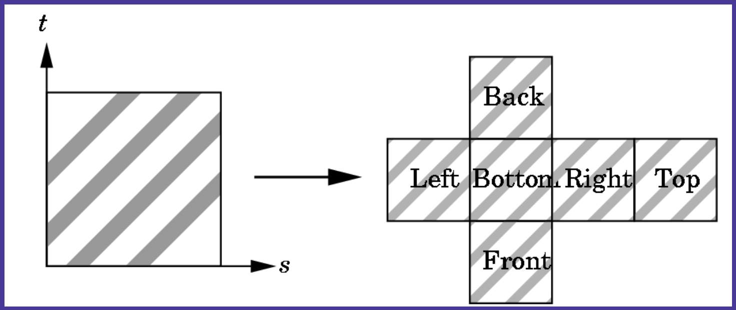
Specifying the Mapping Function

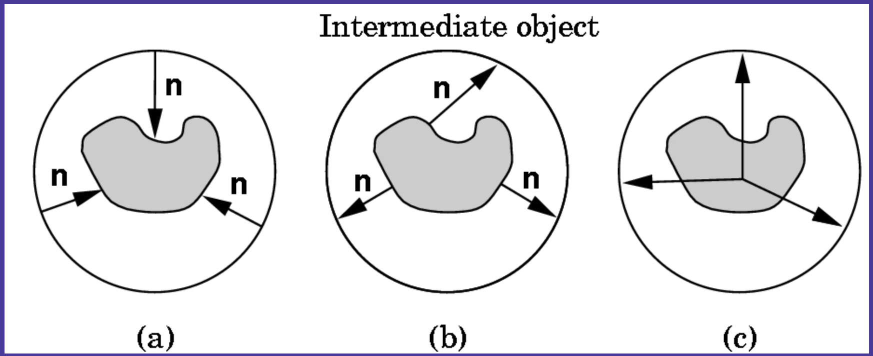
What about arbitrary polygonal objects?

Two step mapping:

- To a canonical shape first
- Then project normals from object

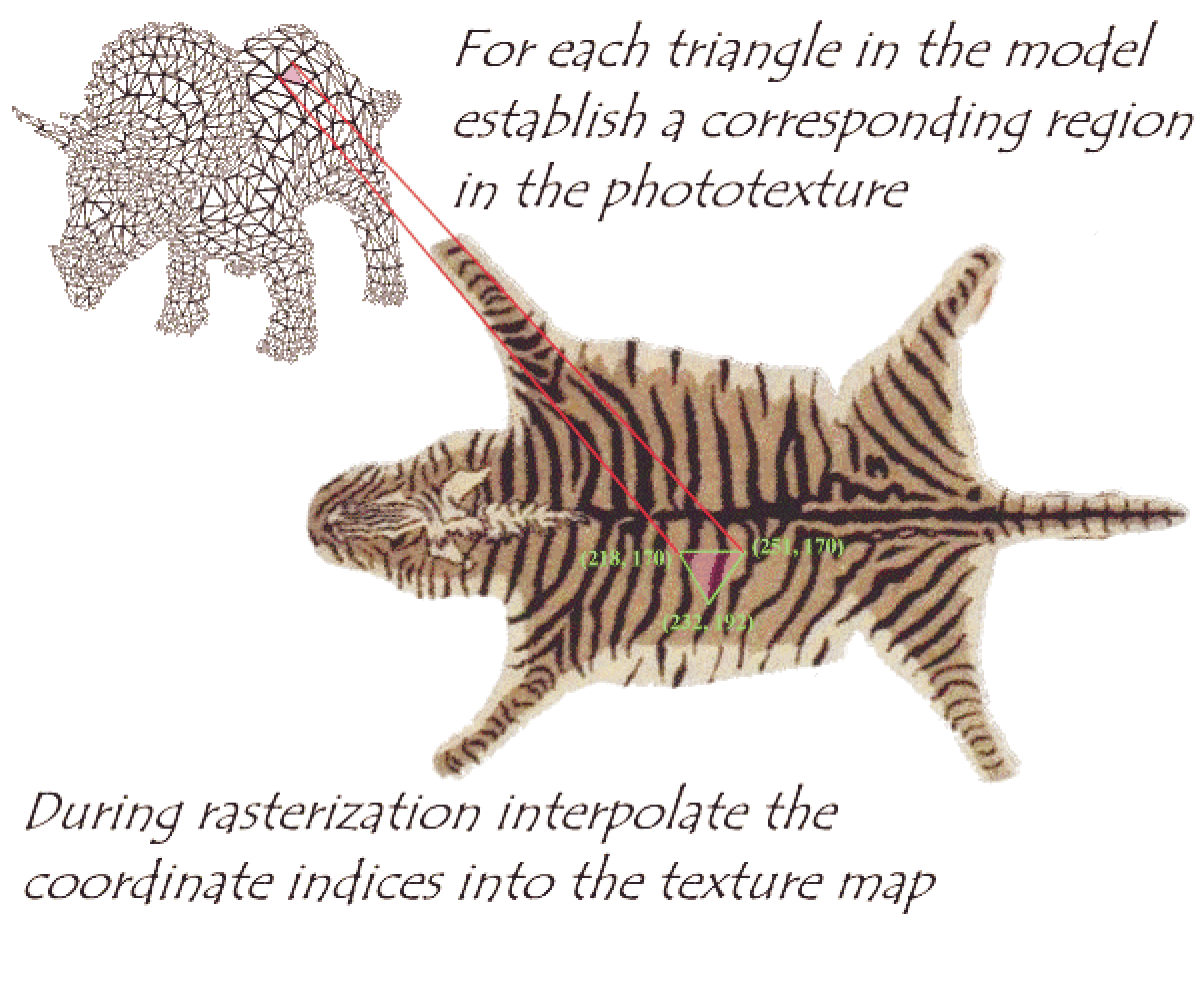






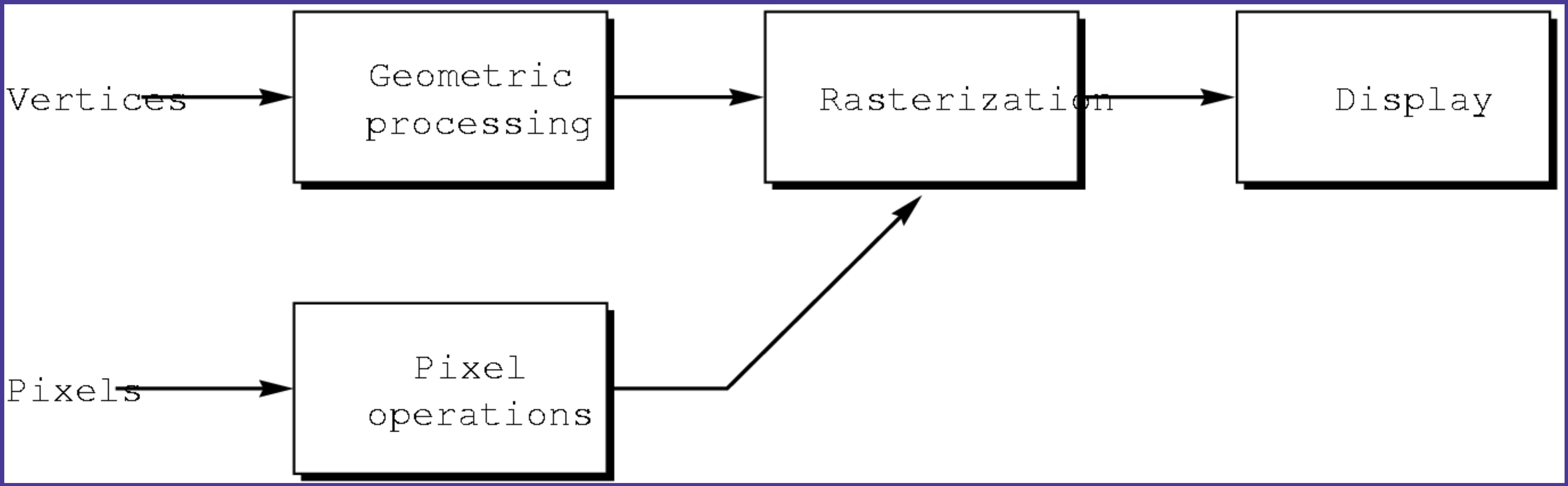
a) From texture value to object
b) Use normals to find texture
c) Line from center to point to texture value

Or design the mapping by hand



Texture Mapping in OpenGL

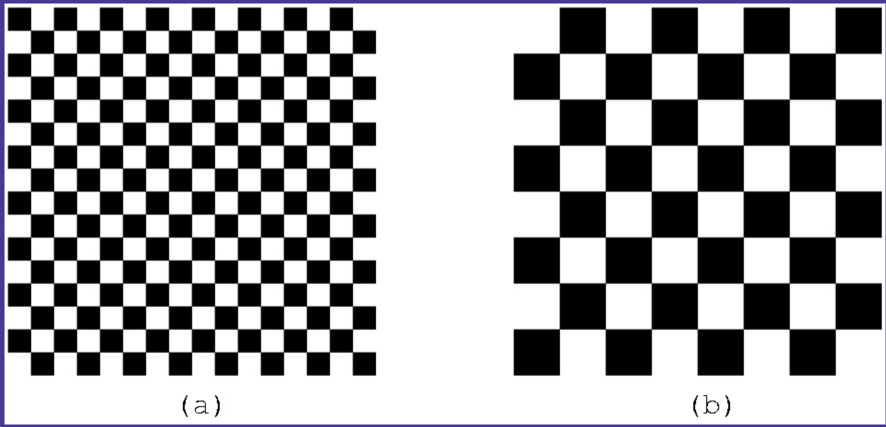
A parallel pipeline for pixel operations: Texture mapping is part of the shading process



Texture Mapping in OpenGL

```
Glubyte my_texels[512][512];
glTexImage2D(GL_TEXTURE_2D, 0, 3, 512, 512, 0,
GL_RGB,GL_UNSIGNED_BYTE, my_texels);
/* level, components, w, h, border, format, type, tarray */
glEnable(GL_TEXTURE_2D);
/* assign texture coordinates */
glBegin(GL_QUAD);
        glTexCoord2f(0.0, 0.0);
        glVertex2f(x1,y1,z1);
        glTexCoord2f(1.0, 0.0);
        glVertex2f(x2,y2,z2);
        glTexCoord2f(1.0,1.0);
        glVertex2f(x3,y3,z3);
        glTexCoord2f(0.0,1.0);
        glVertex2f(x4,y4,z4);
glEnd();
```

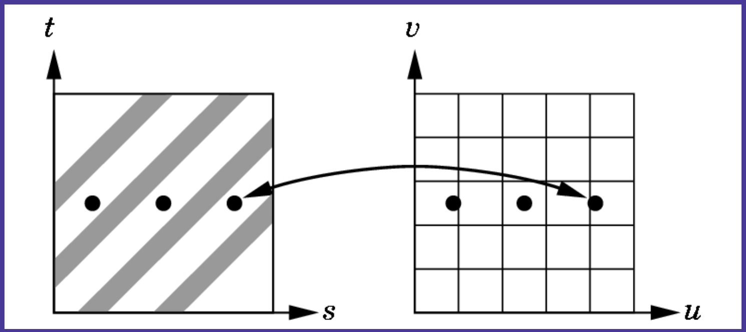
Can map to all or part of array



Grungy details we've ignored

- Specify s or t out of range? Use GL_TEXTURE_WRAP in glTexParameter because many textures are carefully designed to repeat

- Aliasing? Mapping doesn't send you to the center of a texel. Can average nearest 2x2 texels using GL_LINEAR

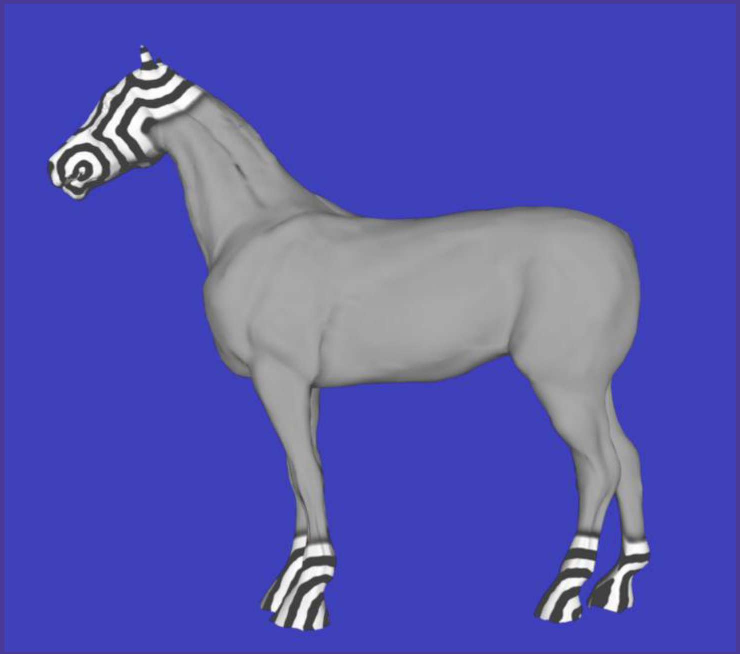


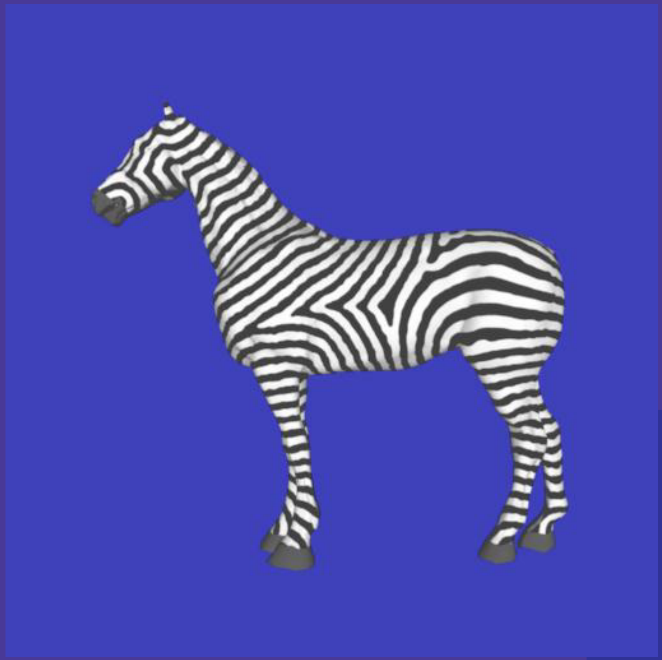
- Mipmapping: use textures of varying resolutions. 64x64 becomes 32x32,16x16,8x8,4x4,2x2 and 1x1 arrays with gluBuild2Dmipmaps

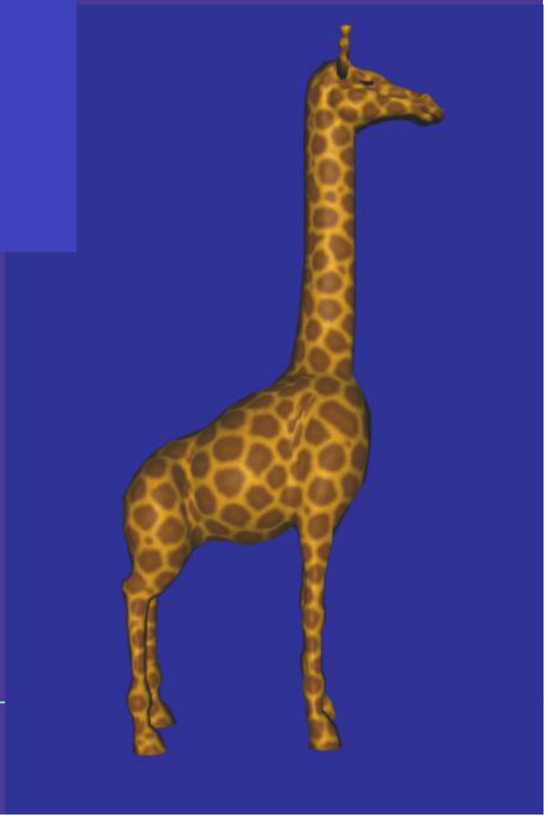
Texture Generation

Photographs
Drawings
Procedural methods (2D or 3D)
- Associate each x,y,z value directly with an s,t,r value in the texture block (sculpting in marble and granite)

Procedural Methods







Reaction-Diffusion
Greg Turk, Siggraph '91

Solid Textures

- Have a 3-D array of texture values (e.g., a block of marble)
  – Use a function [xyz] -> [RGB] to map colors to points in space
- Such a 3D map is called a *solid texture map*
- In practice the map is often defined procedurally
  – No need to store an entire 3D array of colors
  – Just define a function to generate a color for each 3D point
- The most interesting solid textures are random ones
  – a great marble algorithm has now become cliché
- Evaluate the texture coordinates in object coordinates - otherwise moving the object changes its texture!

From: *An Image Synthesizer*
by Ken Perlin, SIGGRAPH '85

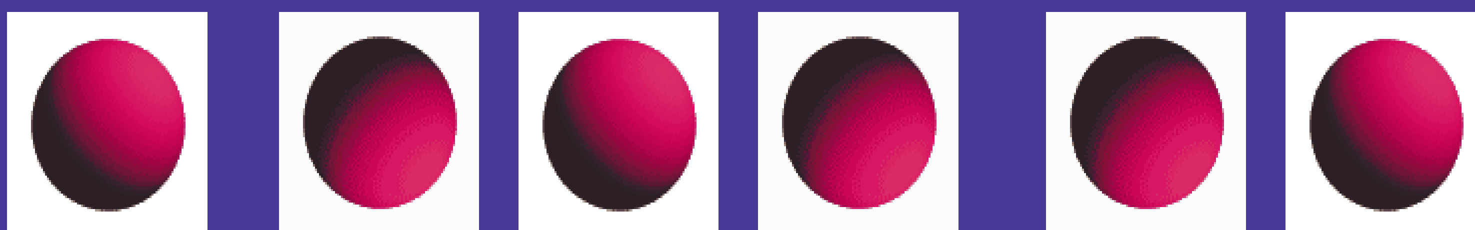
Uses for Texture Mapping (Heckbert 1986)

Use texture to affect a variety of parameters

- surface color (Catmull 1974) - color (radiance) of each point on surface
- surface reflectance - reflectance coefficients k_d, k_s, or n_{shiny}
- normal vector - bump mapping (Blinn 1978)
- geometry - displacement mapping
- transparency - transparency mapping (clouds) (Gardener 1985)
- light source radiance - environment mapping (Blinn 1978)

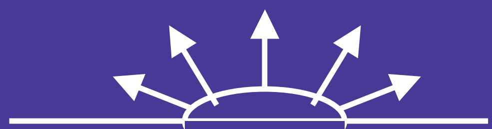
Bump Mapping: A Dirty Trick

- Which spots bulge out, and which are indented?
- Answer: None! This is a flat image.
- The human visual system is hard-coded to expect light from above
- In CG, we can perturb the normal vector without having to make any actual change to the shape.

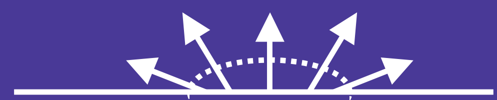


Bump Mapping

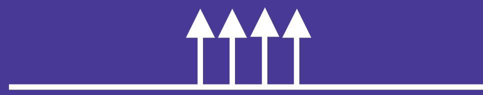
- Basic texture mapping paints on to a smooth surface
- How do you make a surface look *rough*?
  - – Option 1: model the surface with many small polygons
  - – Option 2: perturb the normal vectors before the shading calculation



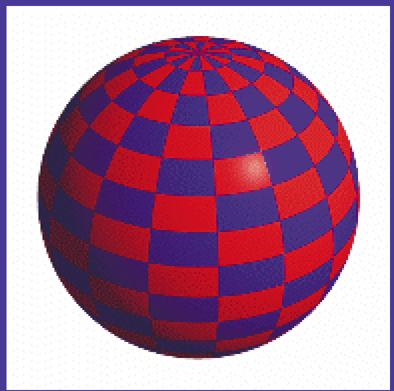
Real Bump



Fake Bump



Flat Plane

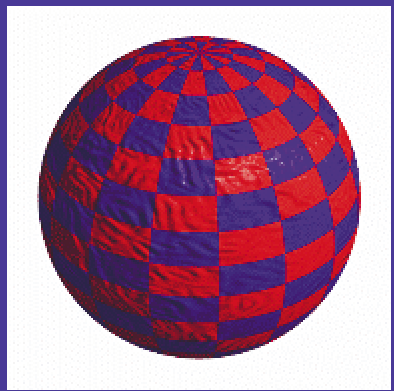


Sphere w/Diffuse Texture Map

+

Bump Map

=



Sphere w/Diffuse Texture + Bump Map

Bump Mapping

- Basic texture mapping paints on to a smooth surface
- How do you make a surface look *rough*?
  - – Option 1: model the surface with many small polygons
  - – Option 2: perturb the normal vectors before the shading calculation
    - » the surface doesn't actually change, but shading makes it look that way
    - » bump map fakes small displacements above or below the true surface
    - » can use texture-mapping for this
      - – texture image gives amount to perturb surface normal

What kind of anomaly will this produce?

Greg Turk

Bump Mapping

Let $\mathbf{p}(u,v)$ be a point on a parametric surface.

Unit normal is
$$\mathbf{n} = \frac{\mathbf{p_u} \times \mathbf{p_v}}{|\mathbf{p_u} \times \mathbf{p_v}|}$$

Where
$$\mathbf{p}_u = \begin{bmatrix} \frac{\partial x}{\partial u} \\ \frac{\partial y}{\partial u} \\ \frac{\partial z}{\partial u} \end{bmatrix} \qquad \mathbf{p}_v = \begin{bmatrix} \frac{\partial x}{\partial v} \\ \frac{\partial y}{\partial v} \\ \frac{\partial z}{\partial v} \end{bmatrix}$$

Partial derivatives lying in the tangent plane to surface at point P

Bump Mapping

Displace the surface in the normal direction by $d(u,v)$

Then $\mathbf{p}' = \mathbf{p} + d(u,v)\mathbf{n}$

We don't actually change the surface (p), just the normal. Need to calculate:

$$\mathbf{n}' = \mathbf{p}_u' \times \mathbf{p}_v'$$

Bump Mapping

Compute the partial derivatives by differentiating p'

$$\mathbf{p}'_u = \mathbf{p}_u + \frac{\partial d}{\partial u}\mathbf{n} + d(u,v)\mathbf{n}_u$$

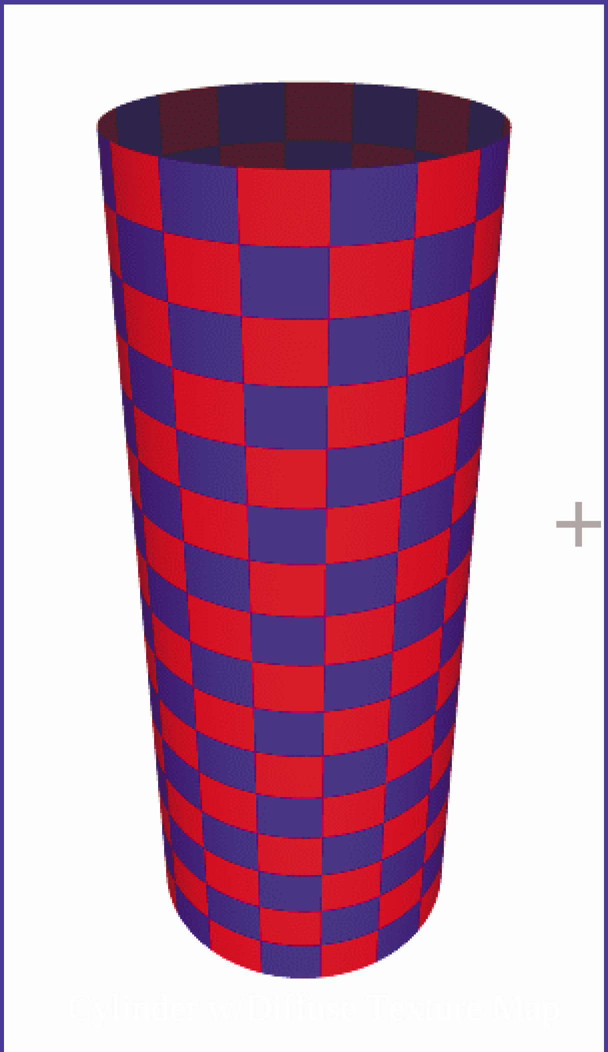
$$\mathbf{p}'_v = \mathbf{p}_v + \frac{\partial d}{\partial v}\mathbf{n} + d(u,v)\mathbf{n}_v$$

If d is small

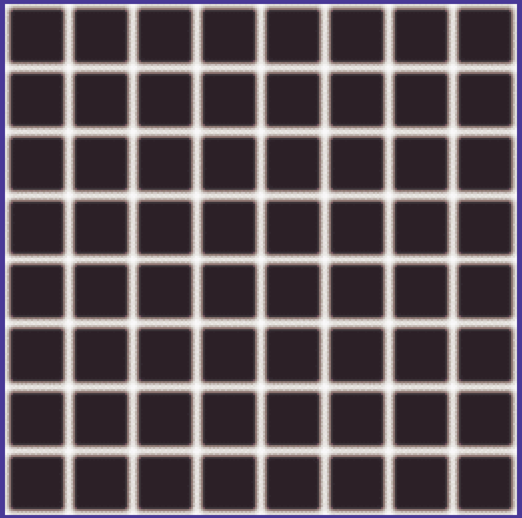
$$\mathbf{n}' = \mathbf{n} + \frac{\partial d}{\partial v}\mathbf{n}\times\mathbf{p}_v + \frac{\partial d}{\partial u}\mathbf{n}\times\mathbf{p}_u$$

Pre-compute arrays of $\frac{\partial d}{\partial v}, \frac{\partial d}{\partial u}$

Another Bump Mapping Example



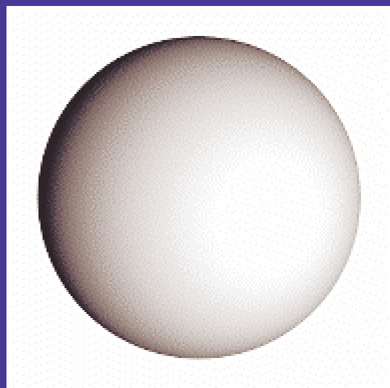
+



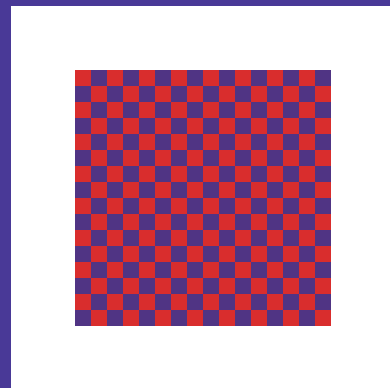
Bump Map

=

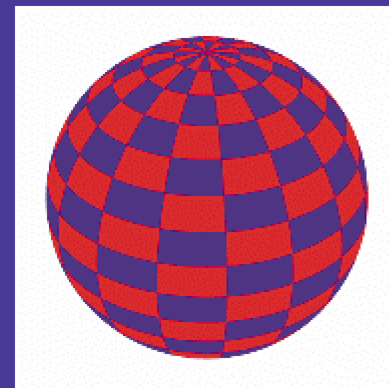
Radiance vs. Reflectance Mapping



+

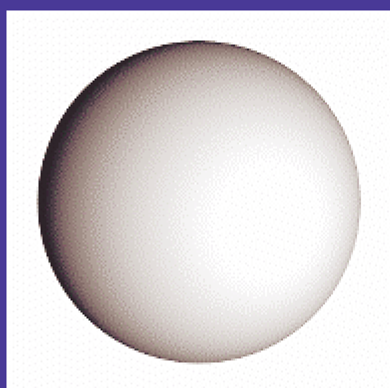


=

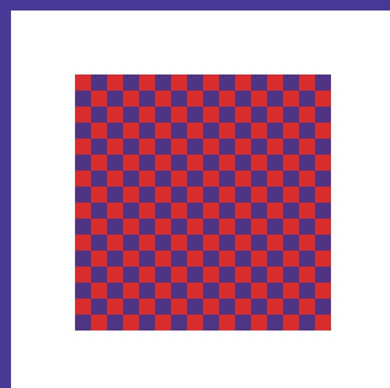


Sphere w/ Uniform Diffuse coefficient | Radiance Map | Sphere w/ Radiance Map

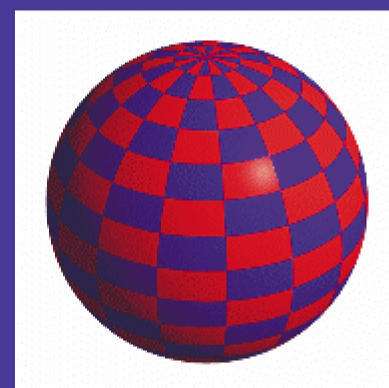
Texture specifies (isotropic) radiance for each point on surface



+



=

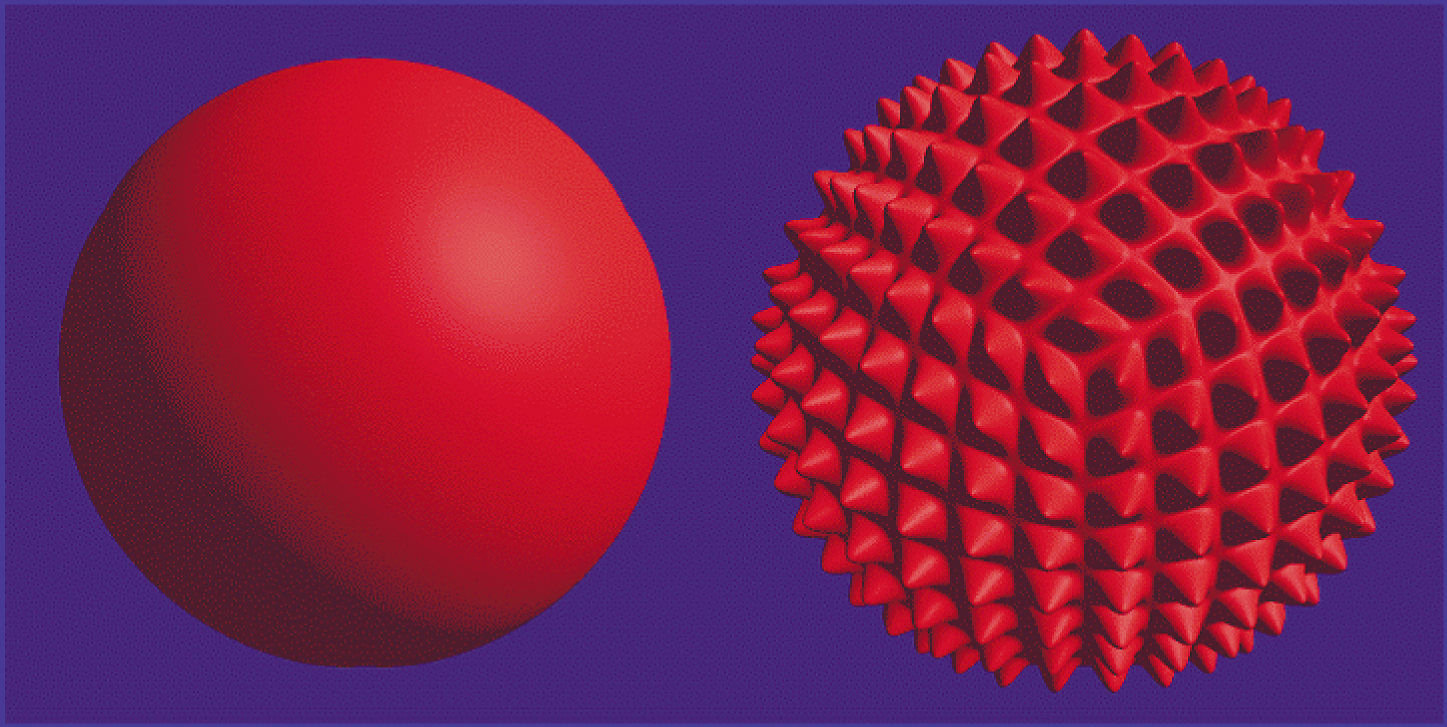


Sphere w/ Uniform Diffuse coefficient | Reflectance (k_d) Map | Sphere w/ Reflectance Map

Texture specifies diffuse color (k_d coefficients) for each point on surface
- three coefficients, one each for R, G, and B radiance channels

Displacement Mapping

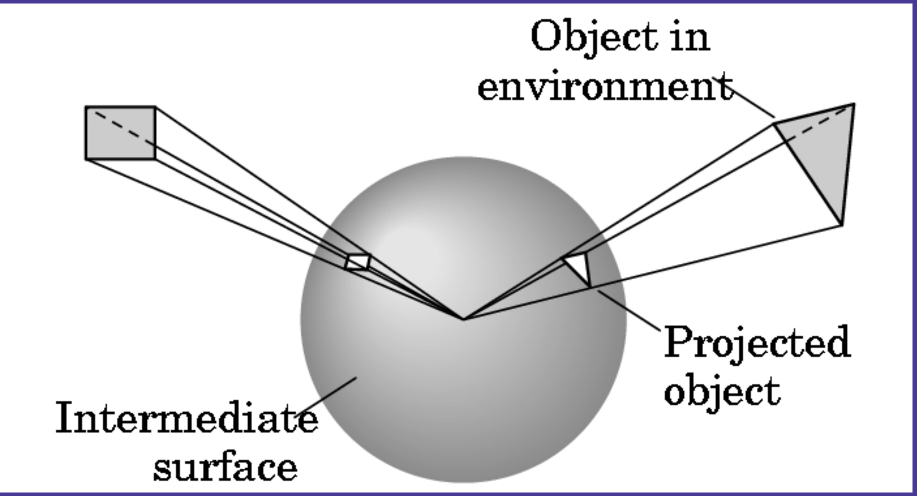
- Use texture map to displace each point on the surface
  - Texture value gives amount to move in direction normal to surface

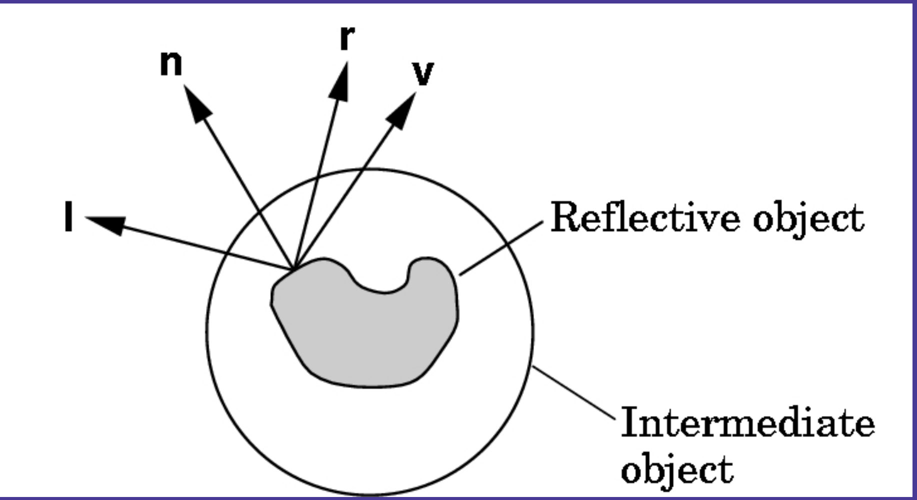


- How is this different from bump mapping?

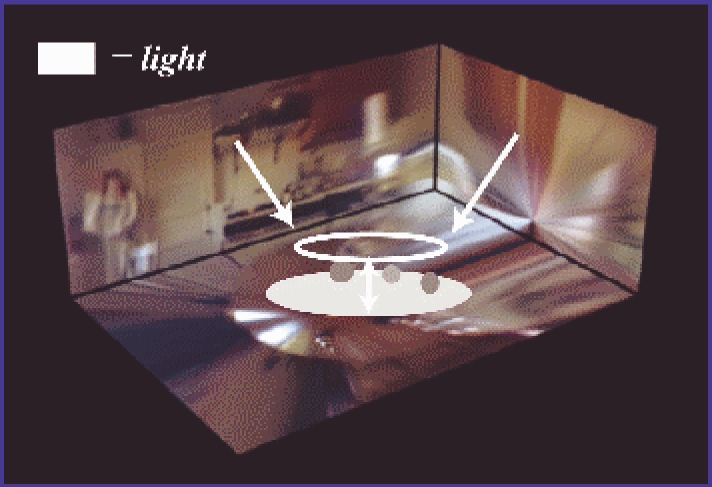
Environment Mapping

Specular reflections that mirror the environment





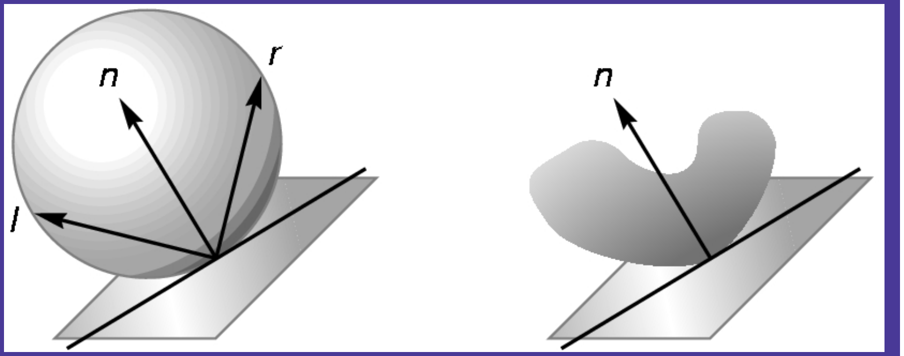
Cube is a natural intermediate object for a room



Environment Mapping

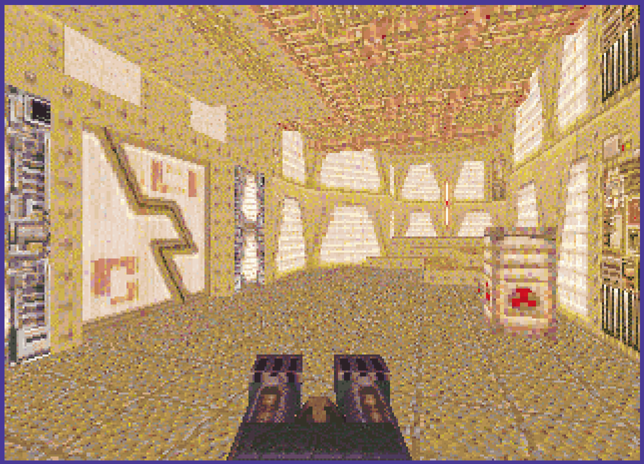
- Generate the environment map either by scanning or by rendering the scene from the point of view of the object (win here because people’s ability to do the reverse mapping in their heads is bad—they won’t notice flaws)
- OpenGL can automatically generate the coordinates for a spherical mapping. Given a vertex and a normal, find point on sphere that has same tangent:

```
glTexGenfv(GL_S, GL_SPHERE_MAP,0);
glTexGenfv(GL_T,GL_SPHERE_MAP,0);
glEnable (GL_TEXTURE_GEN_S);
glEnable (GL_TEXTURE_GEN_T);
```



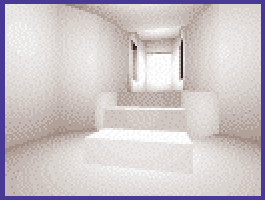
More Tricks: Light Mapping

- *Quake* uses *light maps* in addition to (radiance) texture maps. Texture maps are used to add detail to surfaces, and light maps are used to store pre-computed illumination. The two are multiplied together at run-time, and cached for efficiency.



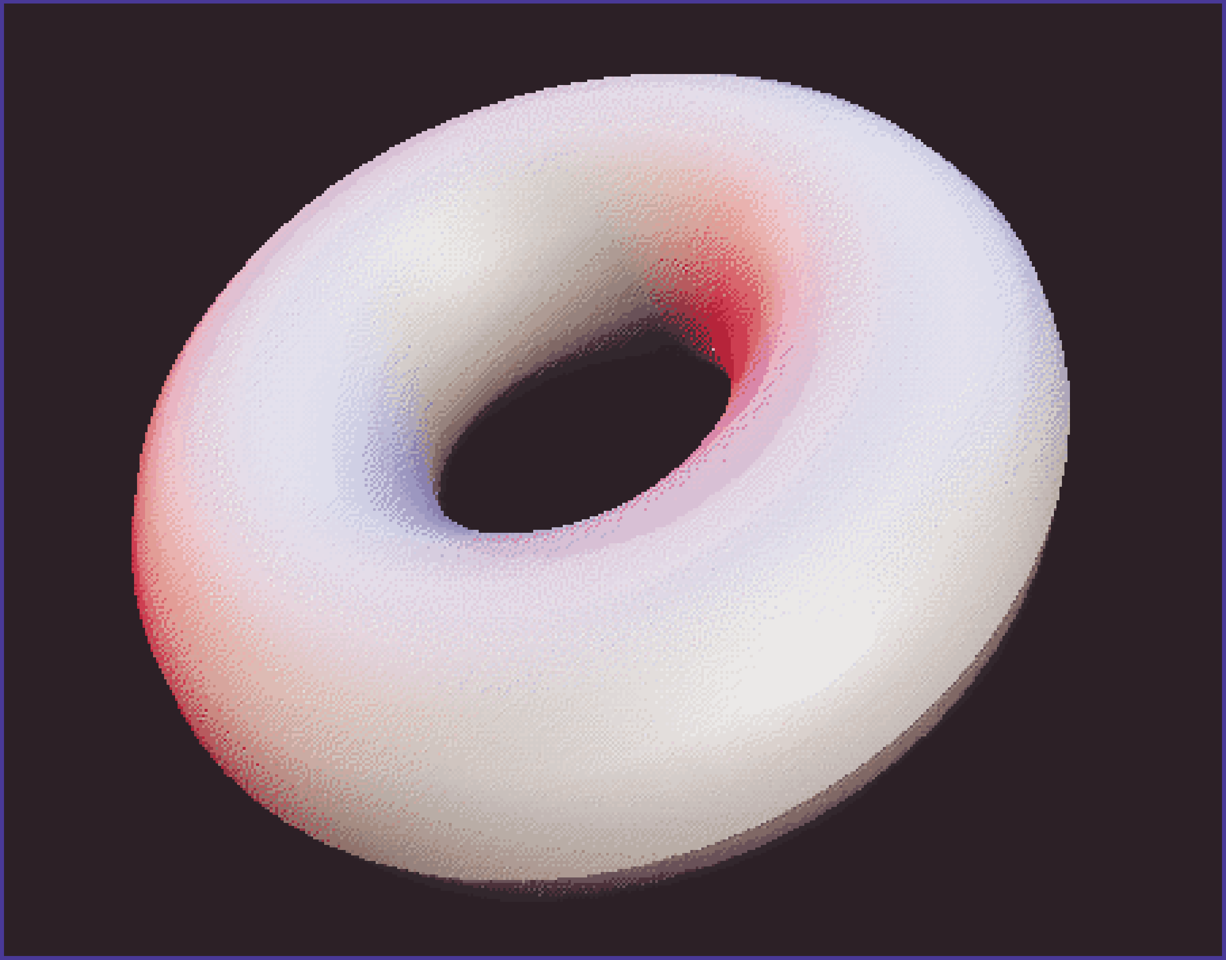
Radiance Texture Map Only

Radiance Texture + Light Map



Light Map

A Donut



A Much More Interesting Donut

Videos?

Announcements

Written Assignment 2 is out – see the web page