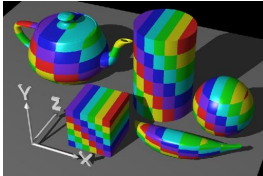


CS770/870 Spring 2017

Texture Mapping



Related material

Cunningham: Ch. 9

Angel 5e: 8.6-8.8, 8.10

Angel 6e: 7.4-7.10

Hill and Kelley: Ch. 8.5

Heckbert, Survey of Texture Mapping, *IEEE Computer Graphics and Applications*, Nov. 1986.

Heckbert and Moreton, Interpolation for Polygon Texture Mapping and Shading, *State of the Art in Computer Graphics: Visualization and Modeling*, Springer-Verlag, 1991

Rosalee Wolfe, Teaching Texture Mapping Visually, Google: Siggraph Texture Tutorial

Preview

- 2D textures
 - mapped onto 3D surfaces
 - polygons
 - polygon meshes
 - curved surfaces
 - indirect mapping
 - direct mapping
 - incorporating into scan conversion
 - linear interpolation
 - rational linear interpolation
- 3D textures
- Bump mapping
- Displacement mapping

2D Texture mapping

- Most real surfaces have patterns/textures
- Inefficient to model a textured surface as many hundreds or thousands of very small triangles
- Instead, *map* a texture onto surface if visible
- Texture usually defined as *pixmap*, whose elements are called *texels*
- Need mapping of texture space to surface
- For each image pixel covered by an object, find texture pixel(s), *texels*, that map to it
- But that means we need a mapping of the texture space to the object's surface

Texture Mapping Overview

- Conceptually, we *map a texture to a surface*
 - want to do mapping in the context of filling pixels
 - Surface + texture must map to pixels
 - So, $\text{map}: T \Rightarrow S$ is a transformation specification
- Classes of mappings
 - indirect mapping via an intermediary surface
 - map texture space to a simpler object and then map the surfaces of that object to the real object
 - direct mappings
 - parametric surface representation
 - 2D coordinate frame transformation

Indirect Mapping

Using an Intermediary Surface

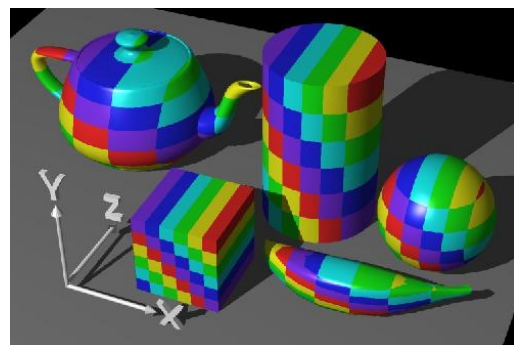
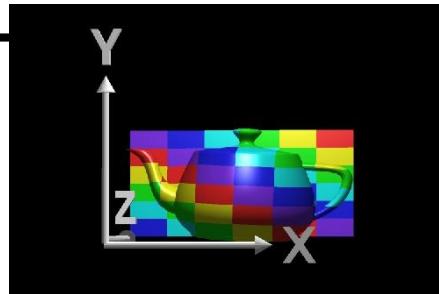
- Can sometimes get a useful, inexpensive result by mapping the texture to a simple surface and then mapping that to the object
 - Plane normal to a principal axis
 - bounding cylinder
 - bounding sphere
 - bounding box
- The surface is called the map shape
- There various ways to map from the object to the shape

CS770/870 Spring 2017 Bergeron

5

Mapping via a Principal Plane

- Map texture (r,s) to object (x,y) , usually with scale
 - usually define texture space as going from $(0,0)$ to $(1,1)$
 - scale that to just cover the extent of object in xy or yz or xz
 - for each visible point on object surface, easy to select the (r,s) color, where $r=x$ and $s=y$ (if using xy plane)



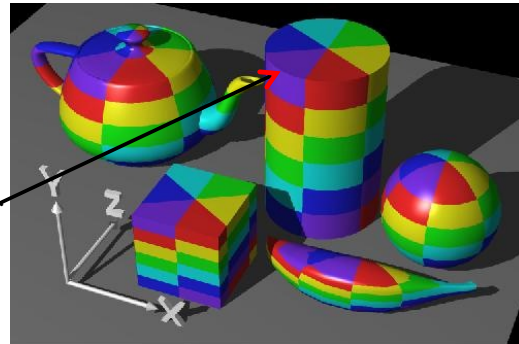
Images from R. Wolfe, Siggraph Tutorial, 1997

CS770/870 Spring 2017 Bergeron

6

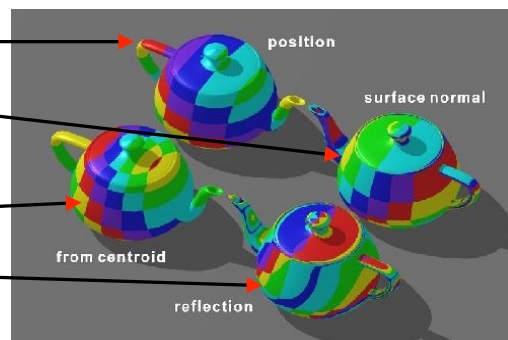
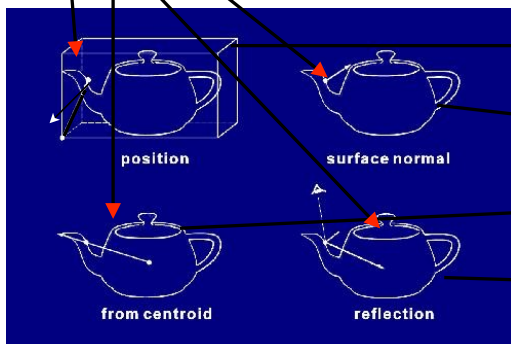
Mapping via a Cylinder

- Easy to wrap texture around a cylinder and map that to object
 - Cylinder is rolled up rectangle
 - map image s to cylinder y
 - map image r to cylinder ϕ
 - can project object's visible x, y position onto cylinder
- Cylinder top and bottom
 - can have “own” texture
 - can “extend” top/bottom edge of side texture along edge to center



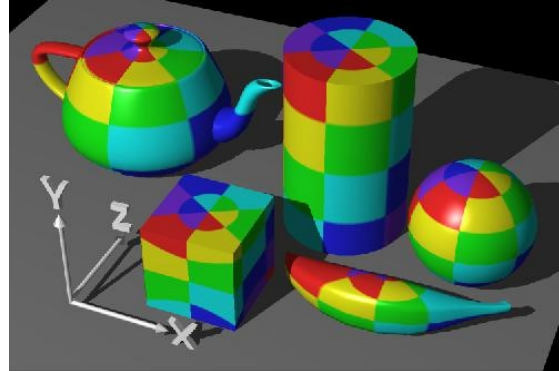
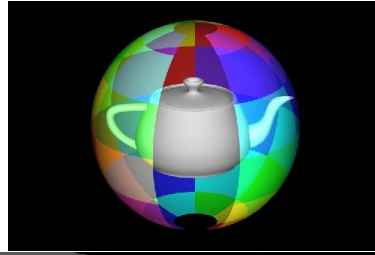
Mapping from Object to Map Shape

- Many ways to map point on object to the map shape
 - position: visible point's position mapped to shape (projection)
 - Using coord system of map shape (cylindrical, spherical, or Cartesian)
 - surface normal from visible point to shape
 - centroid to visible point to shape
 - principal reflection direction of visible point wrt to viewer



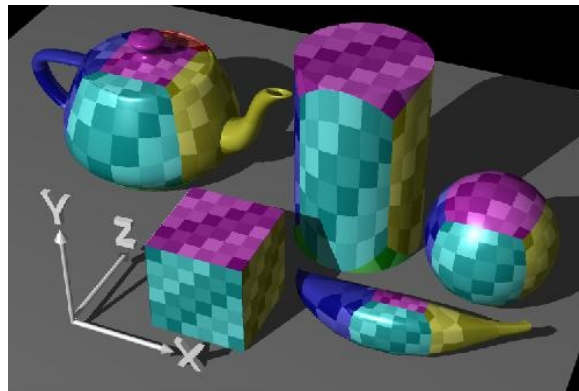
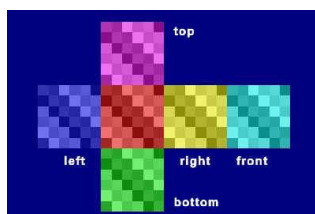
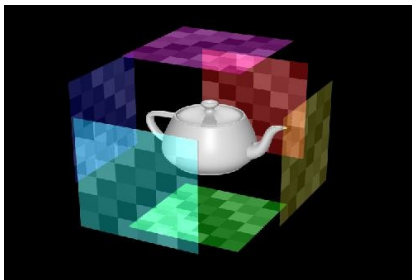
Mapping via a Sphere

- Except for poles, mapping to a sphere, then object is pretty easy
 - could wrap cylinder around sphere, map to sphere, then to object
 - but, it's not hard to map directly to sphere, then to object



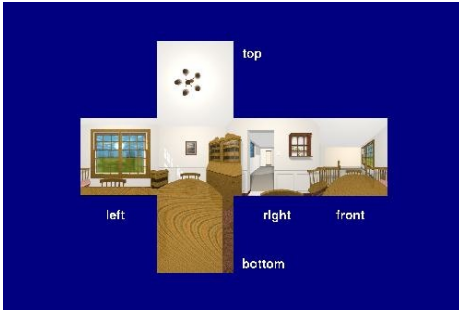
Mapping via a Bounding Box

- Using a box as the intermediary requires a more complex texture, but mapping is still easy



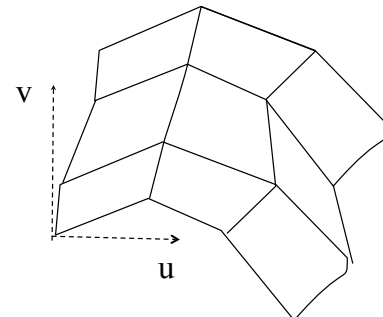
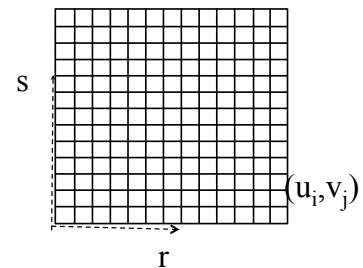
Environment Mapping

- When the textures on the box represent a “photo” of the environment in each direction, it’s called an environment mapping or reflection mapping.



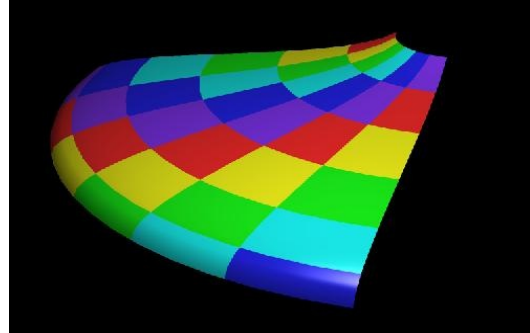
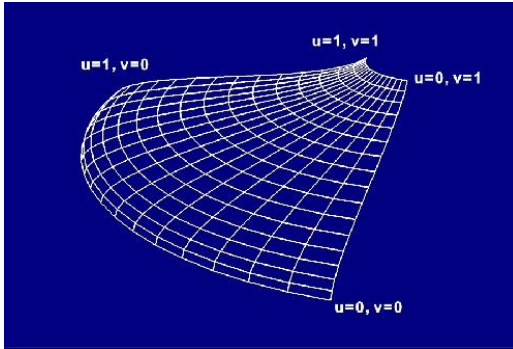
Parametric Surface Representation

- Easy to define 2d parametric texel coordinates, (r, s)
- Not so easy to define a parametric representation for every polygon; same continuity problem
- If a parametric curved surface generated a polygonal mesh, can save the parametric variables at each mesh point, (u_i, v_j)



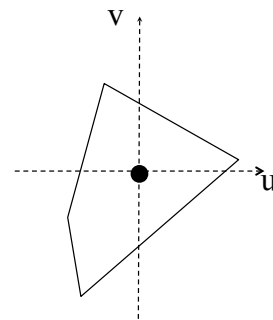
Parametric Mapping Example

- Given mesh generated from a surface, can just store parametric values for the surface points with each mesh point, then look up texel value for visible pixels.



2D Coordinate Frame Mapping

- Coordinate frame transform needs
 - a texture coordinate system
 - easy to derive from the texel index coordinates (2D array)
 - a coordinate system defined on each polygon face
 - tedious for scene designer
 - heuristic algorithm might work:
 - origin is geometric center
 - face v -axis is projection of object coordinate y onto plane, if it exists, and face u -axis is orthogonal to face y , such that $u v n$ (outward normal) is RHS
 - But, what about neighboring faces? How will texture mapping change at boundaries?



Might be ok if have only a few large faces with no continuity needed across them

Coordinate System Issues

- Object is defined in its own coordinate system, OC
- OC are mapped to world coordinates, WC
- WC are mapped to eye coordinates EC
- EC are mapped to normalized projection coords, NPC
- NPC are mapped to screen coordinates, SC

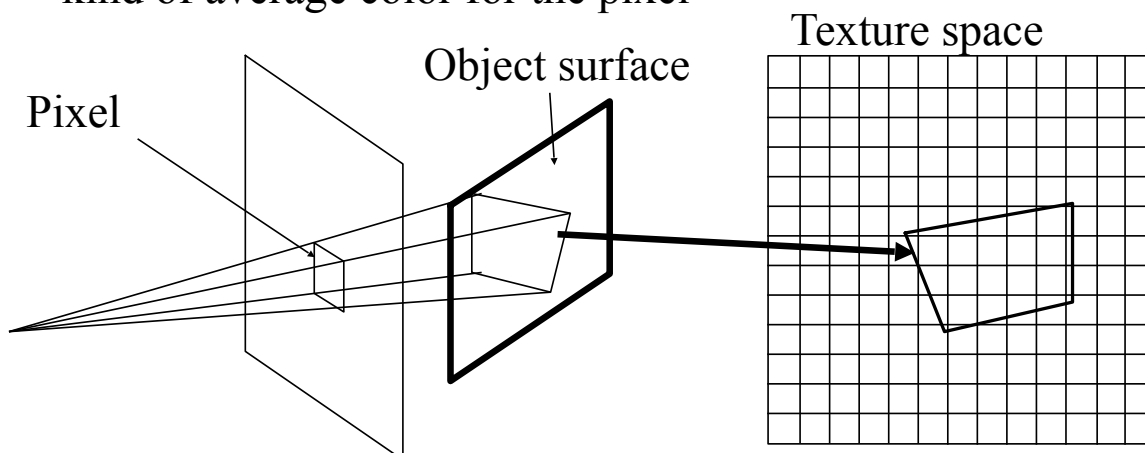
Where is texture mapping done?

- For performance reasons it should be done in screen coordinates – that's the only way to avoid excessive calculations.

Pixel-based Mapping

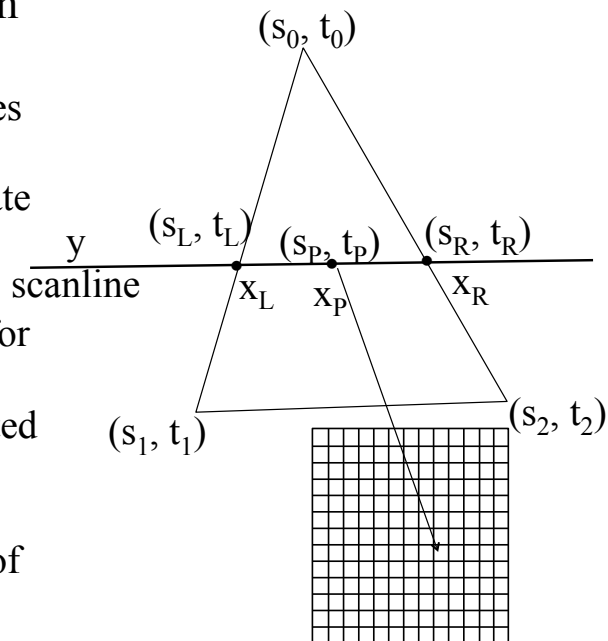
Map pixel to object surface, then to texture space

If image of pixel covers multiple *texels*, take some kind of average color for the pixel



Texture Mapping During Scan Conversion

- Render texture during scan conversion
 - Compute texture coordinates at vertices
 - For each scanline, interpolate coordinates along polygon edges
 - Interpolate along scanline for each pixel
 - At each pixel use interpolated texture coords to look up color in texture map
- Need more work to find area of pixel in texture

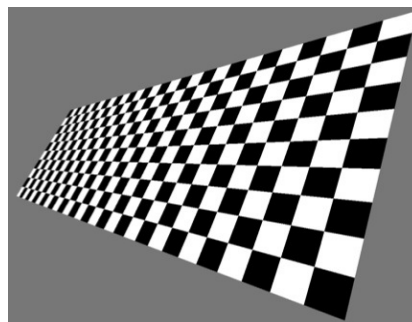
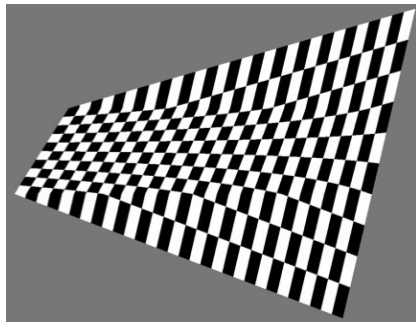


CS770/870 Spring 2017 Bergeron

17

Perspective Transformation Problem

- Linear interpolation fails with perspective projection
 - polygon faces are *foreshortened*; z values along projected plane are not linear in screen space



- Elegant solution: Heckbert and Moreton (UNH CS grad)

Images from Heckbert and Moreton, 1991

CS770/870 Spring 2017 Bergeron

18

Rational Linear Interpolation

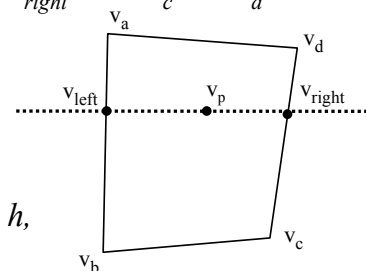
- Heckbert and Moreton's approach:
 - Given a set of vertex *parameters* in object space, $p_i, i=1 \dots n$
 - map vertex $(ox, oy, oz, 1)$ to 3D screen space to get (wx, wy, wz, w)
 - clip against frustum, linearly interpolate all 4 vertex coords as needed
 - divide all parameters and coordinates by w to get:

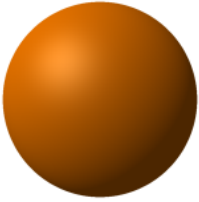
$$s_i = p_i/w, i=1 \dots n \text{ and } (x, y, z, 1)$$
 - add a new parameter, $s_{n+1} = 1/w$
 - apply linear interpolation to all of $(x, y, z, s_1, s_2, \dots, s_n, s_{n+1})$ throughout scan conversion algorithm.
 - at each pixel, compute $r_i = s_i / s_{n+1}$ for $i = 1, \dots, n$
 - Key idea:
 - Do linear interpolation of *normalized* parameters and the homogeneous normalization factor, $1/w$
 - Correct parameter is the interpolated *normalized* parameter $(s_1, s_2, \dots, s_n)$ **divided by** interpolated *normalization* factor, s_{n+1} .

Scan Conversion Parameters

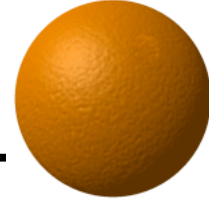
- Identify parameters such as texture coords, color etc.
- Interpolate these parameters divided by the homogeneous coord
 - Texture coordinates (u,v) , divided by w , which is $(u/w, v/w)$
 - Vertex colors (r,g,b) , divided by w $(r/w, g/w, b/w)$
 - Vertex normals divided by w : $(n_x/w, n_y/w, n_z/w)$
 - **plus** the homogeneous normalization factor $(h=1/w)$
- For each (new) parameter value, v (including h)
 - Interpolate v along edges: get v_{left} from v_a and v_b and v_{right} from v_c and v_d
 - Interpolate parameters between edges across scanline: get v_p from v_{left} and v_{right} .
 - For each pixel, p , and each parameter, v_p

$$v'_p = v_p / h_p = v_p / (1/w_p) = v_p * w_p$$
 - So, by interpolating u/w and h and dividing result by h , the homogeneous factor is canceled at the end.





Bump Mapping



- Texture maps can also be used to perturb the surface normal at a pixel before lighting model applied
 - Texture map interpreted as an intensity field (1 variate)
 - At $\text{texel}[i,j]$, compute approximation to the gradient at that point
$$\begin{aligned} dy &= \text{texel}[i+1, j] - \text{texel}[i, j] \\ dx &= \text{texel}[i, j+1] - \text{texel}[i, j] \end{aligned}$$
 - If the surface normal is $N = (N_x, N_y, N_z)$
$$\text{let } N' = (N_x + dx, N_y + dy, N_z)$$
 - Perturbs normal at each pixel differently according to the texture values
- Use color texture map to directly store the perturbation: $(r, g, b) \Rightarrow (dx, dy, dz)$
- Normal maps use the rgb to store the normal itself

Displacement Mapping

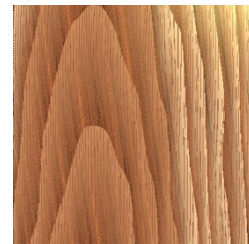
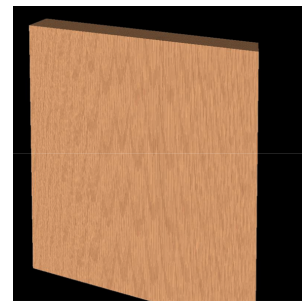
- Bump mapping gives the appearance of a bumpy surface without creating one; this is easily seen in the smooth silhouette
- Displacement mapping changes the *geometry* of the surface by moving vertex positions by a displacement determined from a texture map.
 - some techniques can insert new vertices (tessellate an existing triangle) to get higher resolution surface structure
 - provides correct silhouettes and visibility
 - harder to get correct shadows from the displacements
- Related: parallax mapping, relief mapping

3D Textures

- Defined by
 - a volume data set; a 3D array of *voxels*, or
 - a procedural specification that computes 3D values as needed
- Texture values are accessed by 3 indices (or 3 parametric variables)
- A 3D volume (not just its faces) is mapped to the 3D texture space
 - the faces are rendered as usual, but the texture mapping uses the 3D object coordinates to map to 3D texture coordinates

3D Texture Mapping Characteristics

- Great for modeling wood, marble, stone, etc.
 - great continuity and consistency around corners
 - 3D texture mapping is equivalent to solid wood cabinets and solid countertops (granite, Corian, Ice Stone, etc.)
 - 2D texture mapping is equivalent to laminate cabinets and formica countertops
 - there are pretty good models for generating realistic solid representations of these surfaces
- Much more computation and/or memory needs compared with 2D texture mapping
 - volume data approach is especially expensive because of memory needs -- unless the texture is highly repeatable



Images from Senior thesis by
Claurissa Tuttle, Univ of Utah

Review

- 2D textures mapped onto 3D surfaces
 - Indirect (some kind of projection from a simpler object)
 - Direct via parametric surface representation
- Bump mapping
- Displacement mapping
- 3D texture mapping