Texture Mapping

Roger Crawfis
Ohio State University

Outline

- Modeling surface details with images.
- Texture parameterization
- Texture evaluation
- Anti-aliasing and textures.

Texture Mapping

- Why use textures?

Texture Mapping

- Modeling complexity

Quote

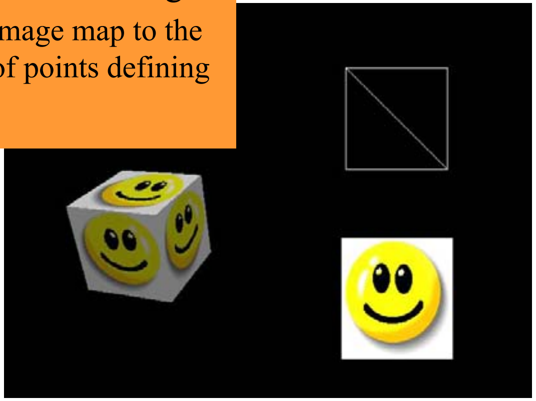
"I am interested in the effects on an object that speak of human intervention. This is another factor that you must take into consideration. How many times has the object been painted? Written on? Treated? Bumped into? Scraped? This is when things get exciting. I am curious about: the wearing away of paint on steps from continual use; scrapes made by a moving dolly along the baseboard of a wall; acrylic paint peeling away from a previous coat of an oil base paint; cigarette burns on tile or wood floors; chewing gum – the black spots on city sidewalks; lover's names and initials scratched onto park benches…"

- Owen Demers

[digital] Texturing & Painting, 2002

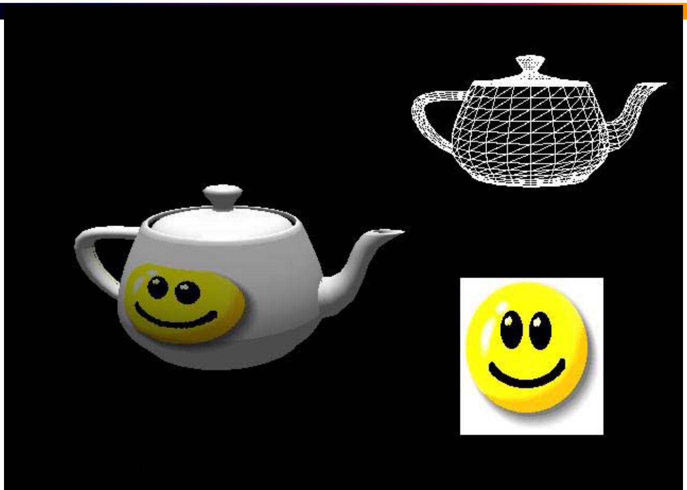
Texture Mapping

- Given an object and an image:
  - How does the image map to the vertices or set of points defining the object?

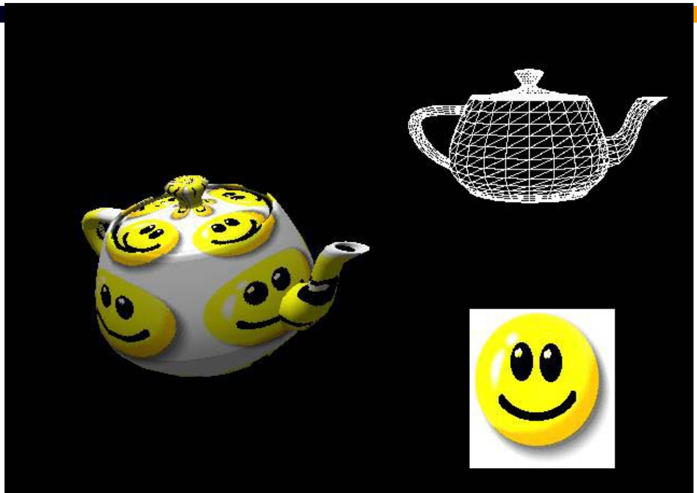


Jason Bryan

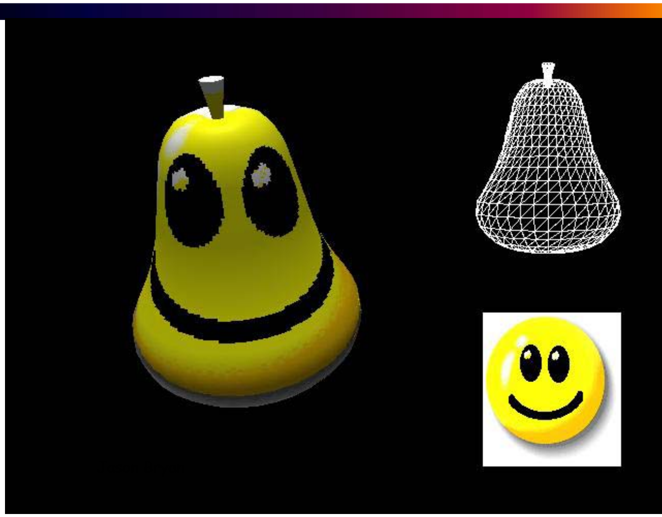
Texture Mapping



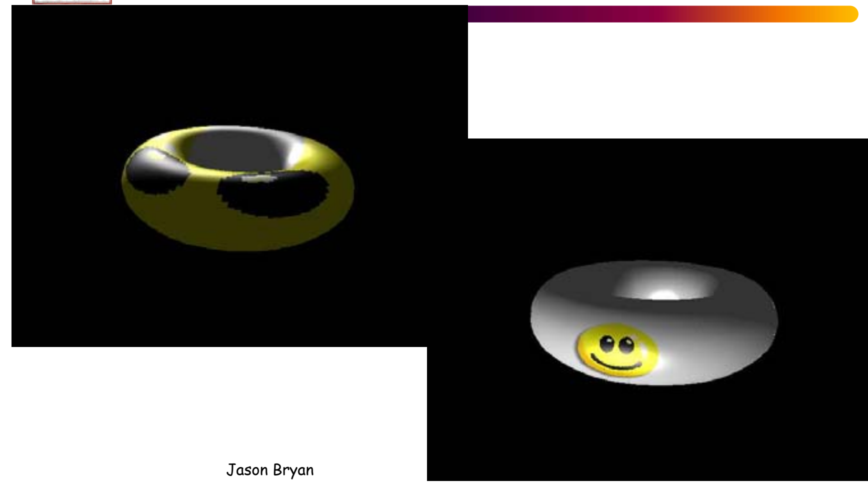
Texture Mapping



Texture Mapping



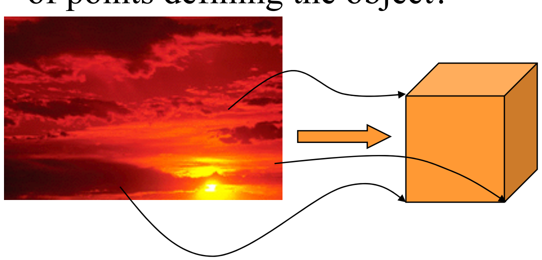
Texture Mapping



Jason Bryan

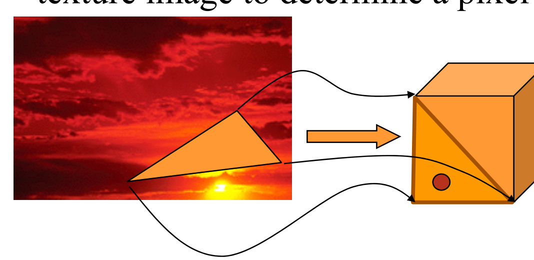
Texture Mapping

- Given an object and an image:
  - How does the image map to the vertices or set of points defining the object?



Texture Mapping

- Given an object with an image mapped to it:
  - How do we use the color information from the texture image to determine a pixel's color?



Texture Mapping

- Problem #1 Fitting a square peg in a round hole

Texture Mapping

- Problem #2 Mapping from a pixel to a texel

What is an image?

- How would I rotate an image 45 degrees?
- How would I translate it 0.5 pixels?

What is a Texture?

- Given the (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 the source of your Texture?

- Procedural Image
- RGB Image
- Intensity image
- Opacity table

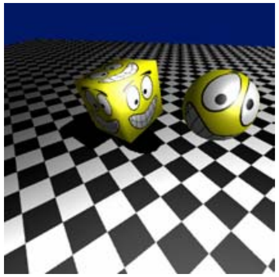
Procedural Texture

Periodic and everything else

Checkerboard

Scale: s= 10

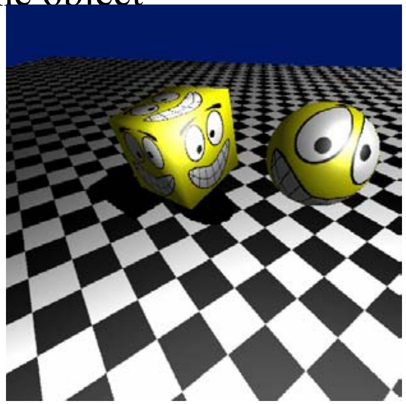
```
If (u * s) % 2=0 && (v * s)%2=0
        texture(u,v) = 0; // black
Else
        texture(u,v) = 1; // white
```



RGB Textures

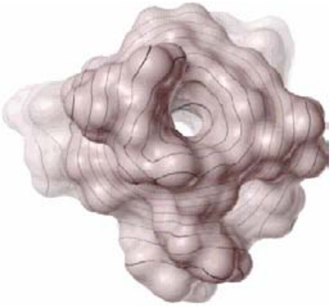
- Places an image on the object

Camuto 1998



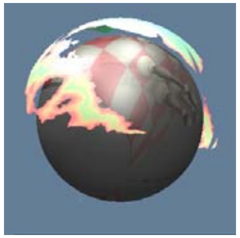
Intensity Modulation Textures

- Multiply the objects color by that of the texture.



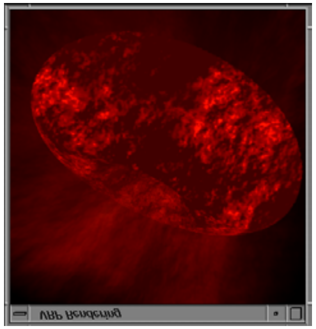
Opacity Textures

- A binary mask, really redefines the geometry.



Color Index Textures

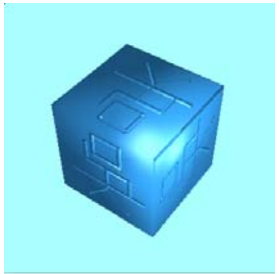
- New Microsoft Extension for 8-bit textures.
- Also some cool new extensions to SGI's OpenGL to perform table look-ups after the texture samples have been computed.

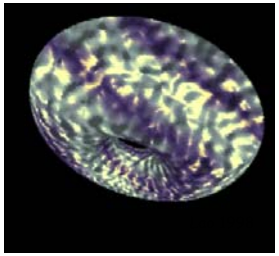


Lao 1998

Bump Mapping

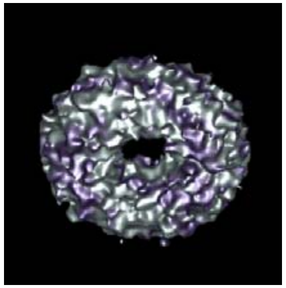
- This modifies the surface normals.
- More on this later.





Displacement Mapping

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



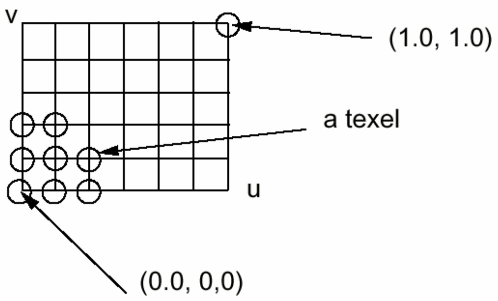
Lao 1998

Reflection Properties

- Kd, Ks
- BDRF's
  - Brushed Aluminum
  - Tweed
  - Non-isotropic or anisotropic surface micro facets.

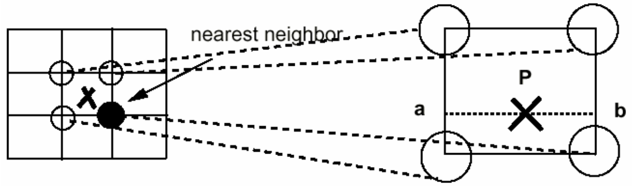
Texture and Texel

- Each pixel in a texture map is called a *Texel*
- Each Texel is associated with a 2D, (u,v), texture coordinate
- The range of u, v is [0.0,1.0]



(u,v) tuple

- For any (u,v) in the range of (0-1, 0-1), we can find the corresponding value in the texture using some interpolation



Two-Stage Mapping

1. Model the mapping: (x,y,z) -> (u,v)
2. Do the mapping

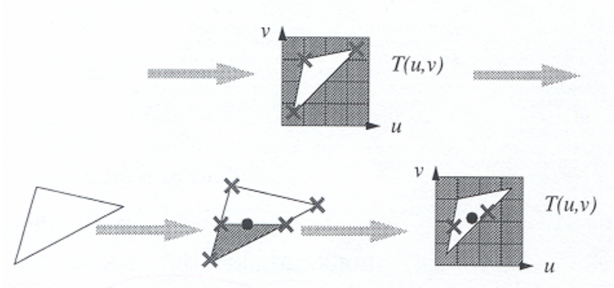


Image space scan

For each scanline, y

For each pixel, x

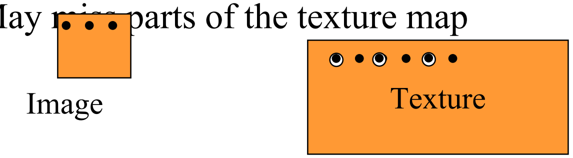
compute u(x,y) and v(x,y)

copy texture(u,v) to image(x,y)

- Samples the warped texture at the appropriate image pixels.
- **inverse** mapping

Image space scan

- Problems:
  - Finding the inverse mapping
    - Use one of the analytical mappings that are invertable.
    - Bi-linear or triangle inverse mapping
  - May miss parts of the texture map



Texture space scan

For each v

For each u

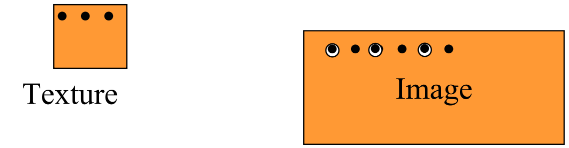
compute x(u,v) and y(u,v)

copy texture(u,v) to image(x,y)

- Places each texture sample to the mapped image pixel.
- **forward** mapping

Texture space scan

- Problems:
  - May not fill image
  - Forward mapping needed



Continuous functions *F(u,v)*

- We are given a discrete set of values:
  - **F**[i,j] for i=0,…,N, j=0,…,M
- Nearest neighbor:
  - **F**(u,v) = **F**[round(N*u), round(M*v)]
- Linear Interpolation:
  - i = floor(N*u), j = floor(M*v)
  - interpolate from **F**[i,j], **F**[i+1,j], **F**[i,j+1], **F**[i+1,j+1]

How do we get *F(u,v)*?

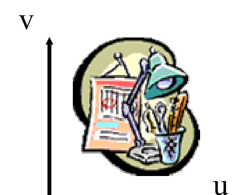
- Higher-order interpolation
  - **F**(u,v) = $\sum_i \sum_j$ **F**[i,j] *h(u,v)*
  - *h(u,v) is called the reconstruction kernel*
    - *Guassian*
    - *Sinc function*
    - *splines*
  - Like linear interpolation, need to find neighbors.
    - Usually four to sixteen

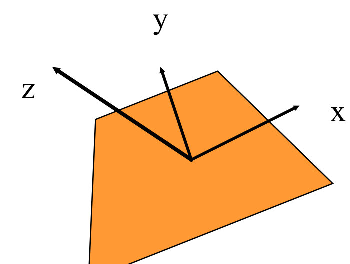
Texture Parameterization

- Definition:
  - The process of assigning texture coordinates or a texture mapping to an object.
- The mapping can be applied:
  - Per-pixel
  - Per-vertex

Texture Parameterization

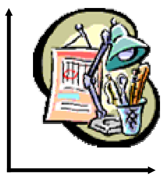
- Mapping to a 3D Plane
  - Simple Affine transformation
    - rotate
    - scale
    - translate

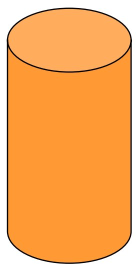




Texture Parameterization

- Mapping to a Cylinder
  - Rotate, translate and scale in the uv-plane
  - u -> theta
  - v -> z
  - x = r cos(theta), y = r sin(theta)



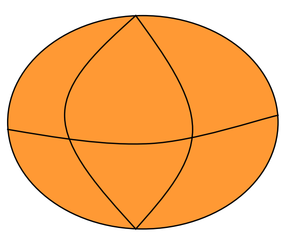


Texture Parameterization

- Mapping to Sphere
  - Impossible!!!!
  - Severe distortion at the poles
  - u -> theta
  - v -> phi
  - x = r sin(theta) cos(phi)
  - y = r sin(theta) sin(phi)
  - z = r cos(theta)

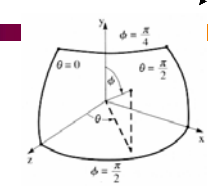
Texture Parameterization

- Mapping to a Sphere



Example (Rogers)

Part of a sphere



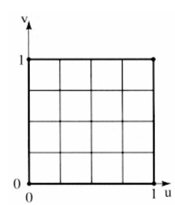
$x(\theta,\phi) = \sin\theta \sin\phi$

$y(\theta,\phi) = \cos\phi$

$z(\theta,\phi) = \cos\theta \sin\phi$

$0 \leq \theta \leq \pi/2$

$\pi/4 \leq \phi \leq \pi/2$

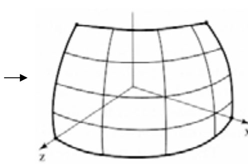


$(u,v) = (0,0) \Leftrightarrow (\theta,\phi) = (0, \pi/2)$

$(u,v) = (1,0) \Leftrightarrow (\theta,\phi) = (\pi/2, \pi/2)$

$(u,v) = (0,1) \Leftrightarrow (\theta,\phi) = (0, \pi/4)$

$(u,v) = (1,1) \Leftrightarrow (\theta,\phi) = (\pi/2, \pi/4)$



- Setup up surface, define correspondence, and voila!

Example Continued

- Can even solve for (θ,ϕ) and (u,v)
  - A= $\pi/2$, B=0, C=$-\pi/4$, D=$\pi/2$

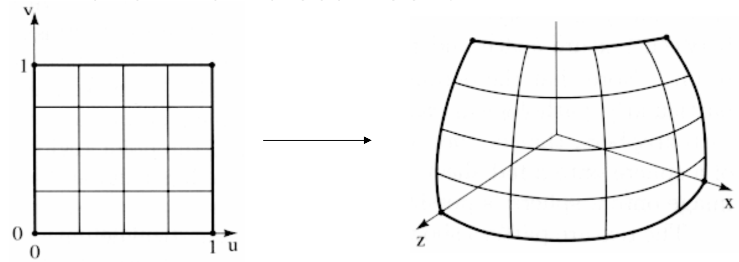
$$\theta(u,v) = \frac{\pi}{2}u \qquad \phi(u,v) = \frac{\pi}{2} - \frac{\pi}{4}v$$

$$u(\theta,\phi) = \frac{\theta}{\pi/2} \qquad v(\theta,\phi) = \frac{\pi/2 - \phi}{\pi/4}$$

So looks like we have the texture space ⇔ object space part done!

All Is Not Good

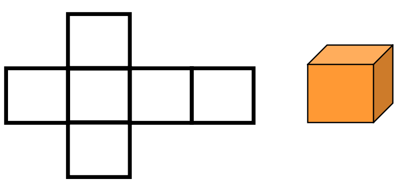
- Let's take a closer look:

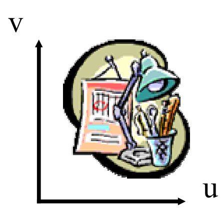


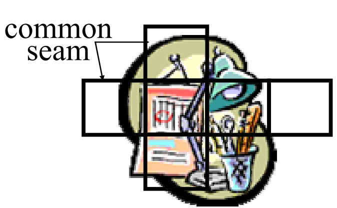
Started with squares and ended with curves ☹
It only gets worse for larger parts of the sphere

Texture Parameterization

- Mapping to a Cube

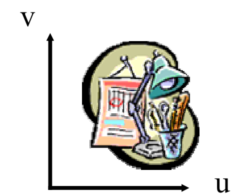


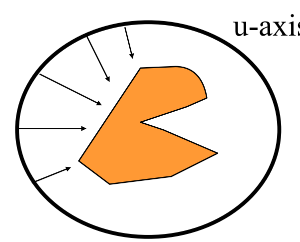




Two-pass Mappings

- Map texture to:
  - Plane
  - Cylinder
  - Sphere
  - Box
- Map object to same.





S and O Mapping

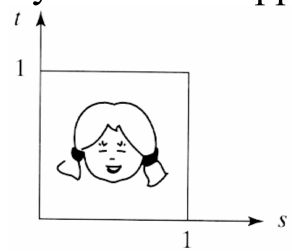
- Pre-distort the texture by mapping it onto a simple surface like a plane, cylinder, sphere, or box
- Map the result of that onto the surface
- Texture → Intermediate is S mapping
- Intermediate → Object is O mapping

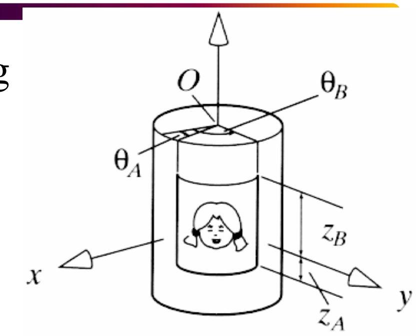
$$(u,v) \xrightarrow{S} (x_i,y_i) \xrightarrow{T} (x_o,y_o,z_o)$$

Texture space ⟶ Intermediate space ⟶ Object space

S Mapping Example

- Cylindrical Mapping

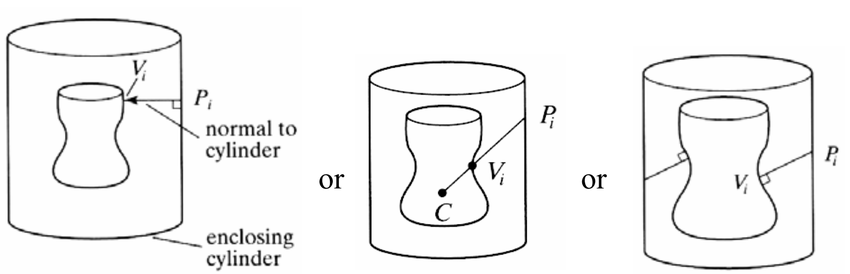




$$s = \frac{\theta - \theta_A}{\theta_B - \theta_A} \qquad t = \frac{z - z_A}{z_B - z_A}$$

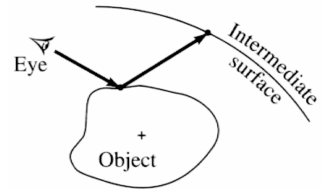
O Mapping

- A method to relate the surface to the cylinder

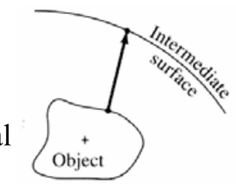


O Mappings Cont'd

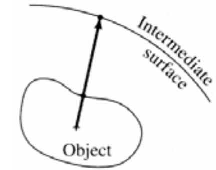
- Bier and Sloan defined 4 main ways



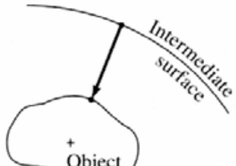
Reflected ray



Object normal



Object centroid



Intermediate surface normal

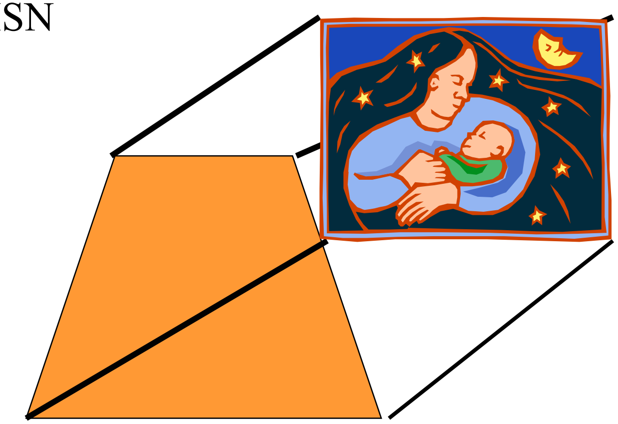
Texture Parameterization

- Plane/ISN (projector)
  - Works well for planar objects
- Cylinder/ISN (shrink-wrap)
  - Works well for solids of revolution
- Box/ISN
- Sphere/Centroid
- Box/Centroid

Works well for roughly spherical shapes

Texture Parameterization

- Plane/ISN



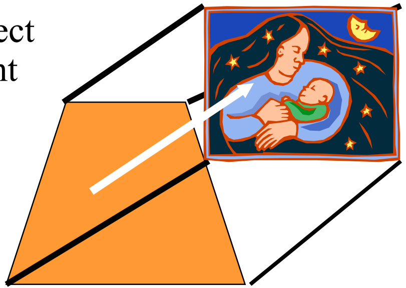
Texture Parameterization

- Plane/ISN
  - Resembles a slide projector
  - Distortions on surfaces perpendicular to the plane.

Watt

Texture Parameterization

- Plane/ISN
  - Draw vector from point (vertex or object space pixel point) in the direction of the texture plane.
  - The vector will intersect the plane at some point depending on the coordinate system



Texture Parameterization

- Cylinder/ISN
  - Distortions on horizontal planes
  - Draw vector from point to cylinder
  - Vector connects point to cylinder axis

Watt

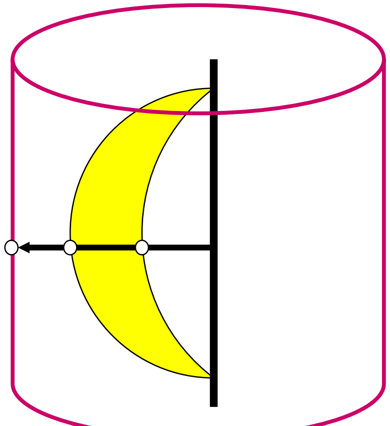
Texture Parameterization

- Sphere/ISN
  - Small distortion everywhere.
  - Draw vector from sphere center through point on the surface and intersect it with the sphere.

Watt

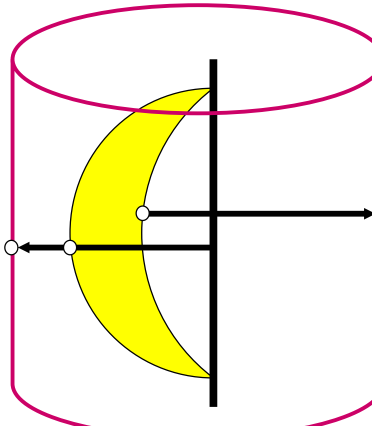
Texture Parameterization

- What is this ISN?
  - Intermediate surface normal.
  - Needed to handle concave objects properly.
  - Sudden flip in texture coordinates when the object crosses the axis.



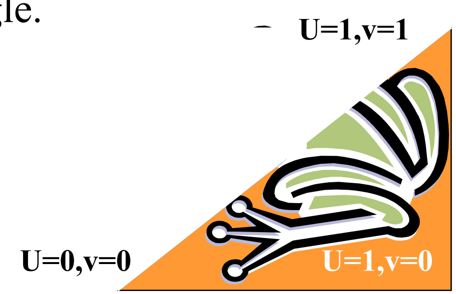
Texture Parameterization

- Flip direction of vector such that it points in the same half-space as the outward surface normal.



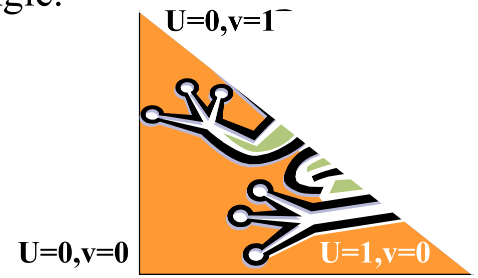
Triangle Mapping

- Given: a triangle with texture coordinates at each vertex.
- Find the texture coordinates at each point within the triangle.



Triangle Mapping

- Given: a triangle with texture coordinates at each vertex.
- Find the texture coordinates at each point within the triangle.

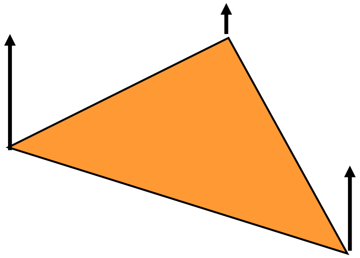


Triangle Mapping

- Triangles define linear mappings.
- u(x,y,z) = Ax + By + Cz + D
- v(x,y,z) = Ex + Fy + Gz + H
- Plug in the each point and corresponding texture coordinate.
- Three equations and three unknowns
- Need to handle special cases: u==u(x,y) or v==v(x), etc.

Triangle Interpolation

- The equation: *f(x,y)* = A*x*+B*y*+C defines a linear function in 2D.
- Knowing the values of *f()* at three locations gives us enough information to solve for A, B and C.
- Provided the triangle lies in the xy-plane.



Triangle Interpolation

- We need to find two 3D functions: *u(x,y,z)* and *v(x,y,z)*.
- However, there is a relationship between x, y and z, so they are not independent.
- The plane equation of the triangle yields:

  $z = Ax + By + D$

Triangle Interpolation

- A linear function in 3D is defined as
  - f(x,y,z) = Ax + By + Cz + D
- Note, four points uniquely determine this equation, hence a tetrahedron has a unique linear function through it.
- Taking a slice plane through this gives us a linear function on the plane.

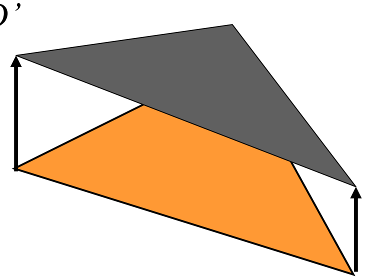
Triangle Interpolation

- Plugging in z from the plane equation.

  $f(x,y,z) = Ax + By + C(Ex+Fy+G) + D$

  $= A'x + B'y + D'$

- For u, we are given:



Triangle Interpolation

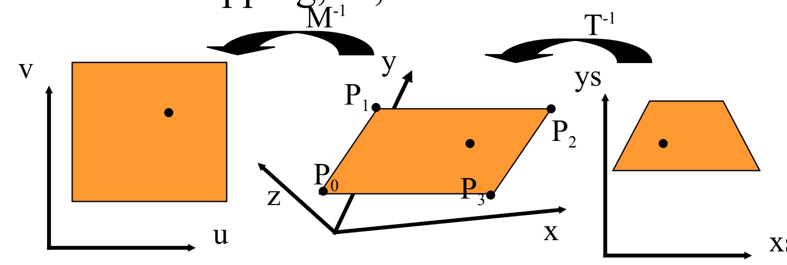
- We get a similar set of equations for *v(x,y,z)*.
- Note, that if the points lie in a plane parallel to the *xz* or *yz*-planes, then *z* is undefined.
- We should then solve the plane equation for *y* or *x*, respectively.
- For robustness, solve the plane equation for the term with the highest coefficient.

Quadrilateral Mapping

- Given: four texture coordinates on four vertices of a quadrilateral.
- Determine the texture coordinates throughout the quadrilateral.

Inverse Bilinear Interpolation

- Given a quadrilateral with texture coordinates at each vertex
- The exact mapping, M, is unknown



Inverse Bilinear Interpolation

- Given:
  - (x_0,y_0,u_0,v_0)
  - (x_1,y_1,u_1,v_1)
  - (x_2,y_2,u_2,v_2)
  - (x_3,y_3,u_3,v_3)
  - (xs,ys,zs) - The screen coords. w/depth
  - T^{-1}
- Calculate (xt,yt,zt) from T^{-1}*(xs,ys,zs)

Inverse Bilinear Interpolation

Barycentric Coordinates:

$x(s,t) = x_0(1-s)(1-t) + x_1(s)(1-t) + x_2(s)(t) + x_3(1-s)(t) = xt$

$y(s,t) = y_0(1-s)(1-t) + y_1(s)(1-t) + y_2(s)(t) + y_3(1-s)(t) = yt$

$z(s,t) = z_0(1-s)(1-t) + z_1(s)(1-t) + z_2(s)(t) + z_3(1-s)(t) = zt$

$u(s,t) = u_0(1-s)(1-t) + u_1(s)(1-t) + u_2(s)(t) + u_3(1-s)(t)$

$v(s,t) = v_0(1-s)(1-t) + v_1(s)(1-t) + v_2(s)(t) + v_3(1-s)(t)$

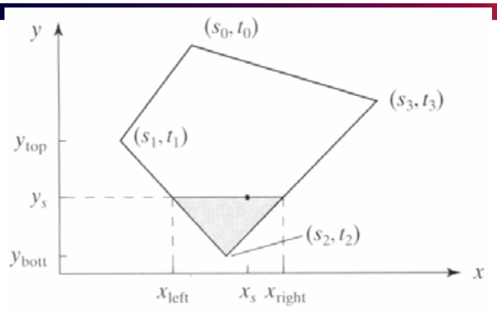
Solve for *s* and *t* using two of the first three equations.

This leads to a quadratic equation, where we want the root between zero and one.

Degenerate Solutions

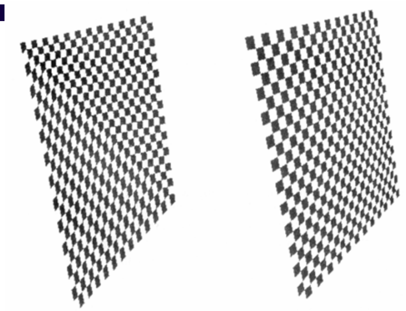
- When mapping a square texture to a rectangle, the solutions will be linear.
  - The quadratic will simplify to a linear equation.
  - s(x,y) = s(x), or s(y).
  - You need to check for these conditions.

Bilinear Interpolation



- Linearly interpolate each edge
- Linearly interpolate (u1,v1),(u2,v2) for each scan line

Uh oh!



- We failed to take into account perspective foreshortening
- Linearly interpolating doesn’t follow the object

What Should We Do?

- If we march in equal steps in screen space (in a line say) then how to do move in texture space?
- Must take into account perspective division

Interpolating Without Explicit Inverse Transform

- Scan-conversion and color/z/normal interpolation take place in screen space
- What about texture coordinates?
  - Do it in clip space, or homogenous coordinates

In Clip space

- Two end points of a line segment (scan line)

$$\mathbf{Q}_1 = (x_1, y_1, z_1, w_1) \qquad \mathbf{Q}_2 = (x_2, y_2, z_2, w_2)$$

- Interpolate for a point Q in-between

$$\mathbf{Q} = (1 - t)\mathbf{Q}_1 + t\mathbf{Q}_2$$

In Screen Space

- From the two end points of a line segment (scan line), interpolate for a point Q in-between: $$\mathbf{Q}^s = (1 - t^s)\mathbf{Q}_1^s + t^s\mathbf{Q}_2^s$$
- Where: $\mathbf{Q}_1^s = \mathbf{Q}_1/w_1$ and $\mathbf{Q}_2^s = \mathbf{Q}_2/w_2$
- Easy to show: in most occasions, t and t^s are different

From t^s to t

- Change of variable: choose
  - a and b such that $1 - t^s = a/(a + b)$, $t^s = b/(a + b)$
  - A and B such that $(1 - t) = A/(A + B)$, $t = B/(A + B)$.
- Easy to get $\mathbf{Q}^s = \dfrac{a\mathbf{Q}_1/w_1 + b\mathbf{Q}_2/w_2}{(a + b)} = \dfrac{A\mathbf{Q}_1 + B\mathbf{Q}_2}{Aw_1 + Bw_2}$
- Easy to verify: $A = aw_2$ and $B = bw_1$ is a solution

Texture Coordinates

- All such interpolation happens in homogeneous space.
- Use A and B to linearly interpolate texture coordinates
- The homogeneous texture coordinate is: (u,v,1)

Homogeneous Texture Coordinates

- $u^l = A/(A+B)\ u_1^l + B/(A+B)u_2^l$
- $w^l = A/(A+B)\ w_1^l + B/(A+B)w_2^l = 1$
- $u = u^l/w^l = u^l = (Au_1^l + Bu_2^l)/(A + B)$
- $u = (au_1^l + Bu_2^l)/(A + B)$
- $u = (au_1^l/w_1^l + bu_2^l/w_2^l\ )/(a\ ^1/w_1^l + b\ ^1/w_2^l)$

Homogeneous Texture Coordinates

- The homogeneous texture coordinates suitable for linear interpolation in screen space are computed simply by
  - Dividing the texture coordinates by screen w
  - Linearly interpolating (u/w,v/w,1/w)
  - Dividing the quantities u/w and v/w by 1/w at each pixel to recover the texture coordinates

OpenGL functions

- *During initialization read in or create the texture image and place it into the OpenGL state.*

```
glTexImage2D (GL_TEXTURE_2D, 0, GL_RGB,
  imageWidth, imageHeight, 0, GL_RGB,
  GL_UNSIGNED_BYTE, imageData);
```

- *Before rendering your textured object, enable texture mapping and tell the system to use this particular texture.*

```
glBindTexture (GL_TEXTURE_2D, 13);
```