Texture Mapping

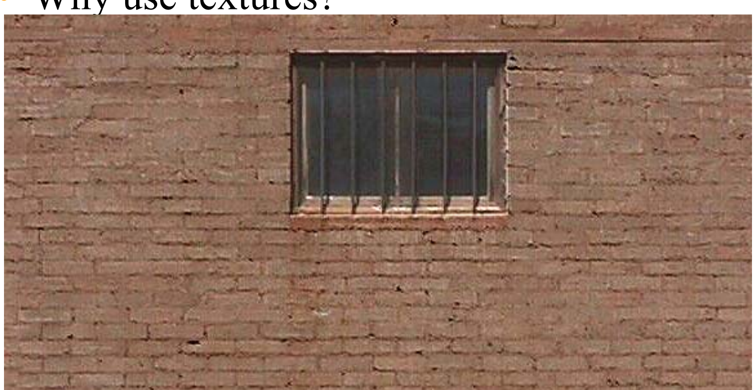
Roger Crawfis
Ohio State University

Outline

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

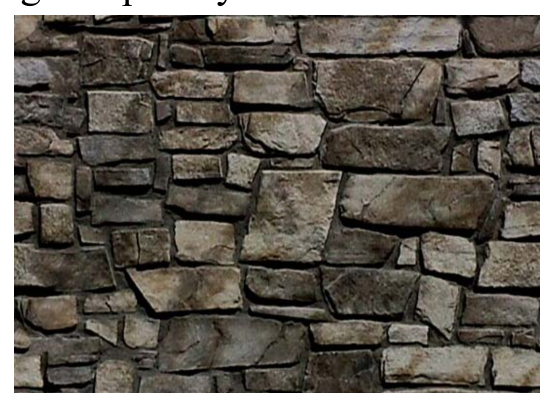
Texture Mapping

- Why use textures?



Texture Mapping

- Modeling complexity



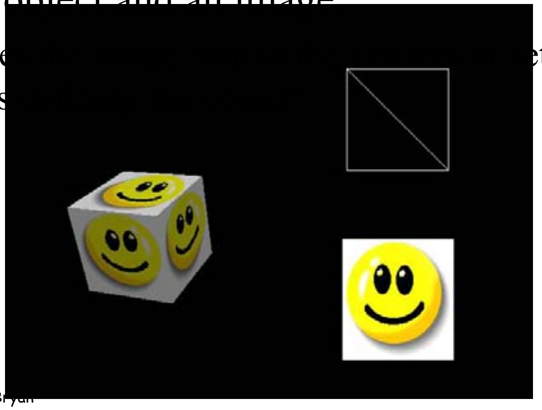
Quote

"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

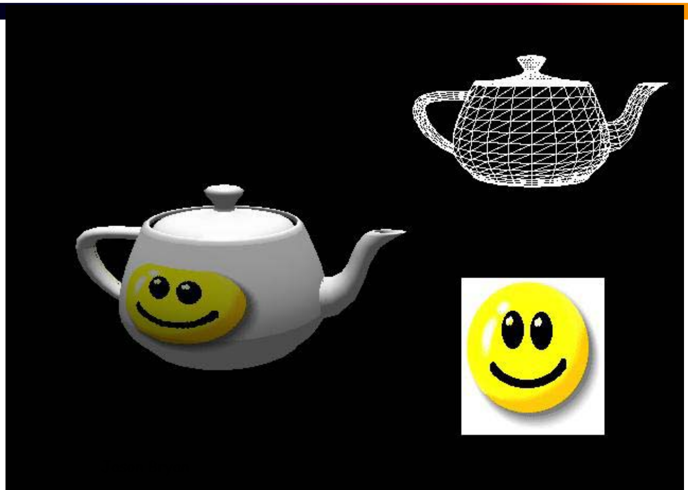
Texture Mapping

- Given an object and an image:
  - How doe … et of points …

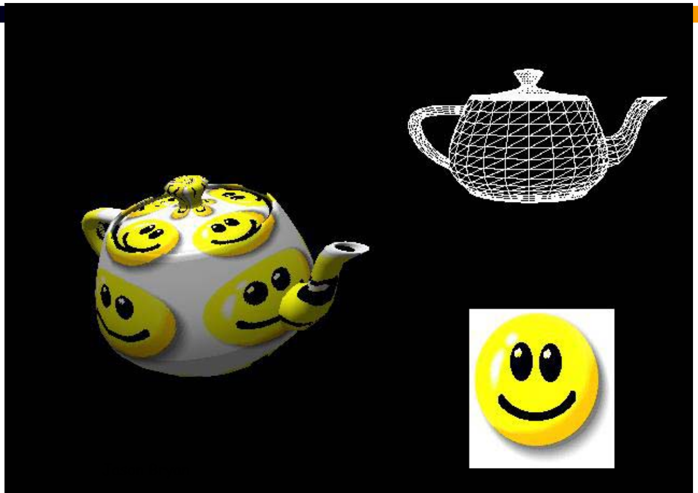


Jason Bryan

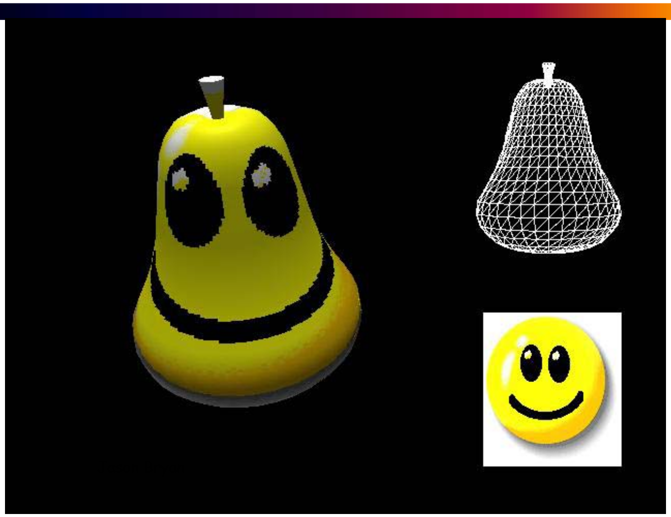
Texture Mapping



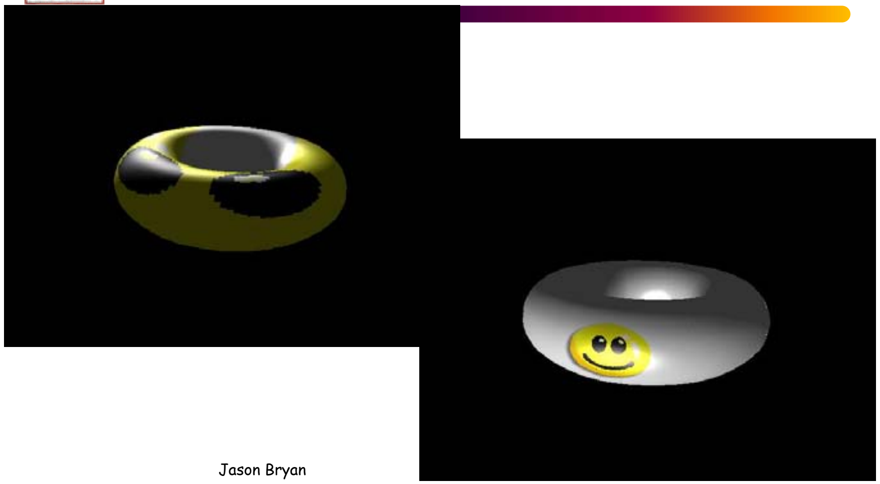
Texture Mapping



Texture Mapping

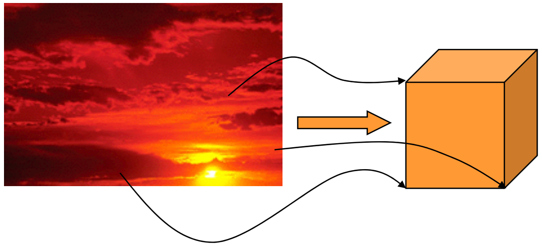


Texture Mapping



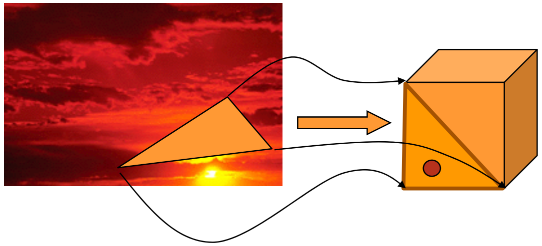
Texture Mapping

- 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

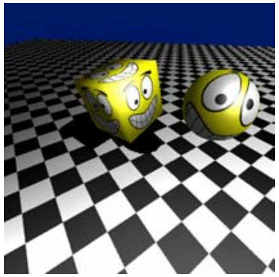
Checkerboard

Scale: s= 10

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

Else

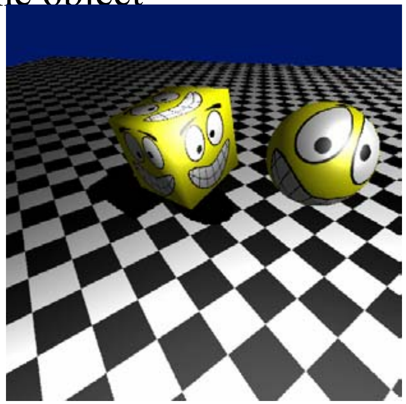
    texture(u,v) = 1; // white



RGB Textures

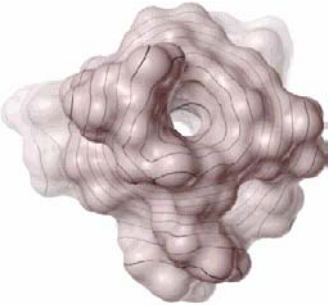
- Places an image on the object

Camuto 1998



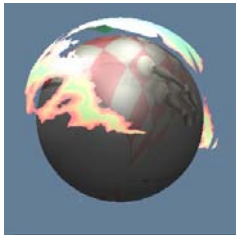
Intensity Modulation Textures

- Multiply the objects color by that of the texture.



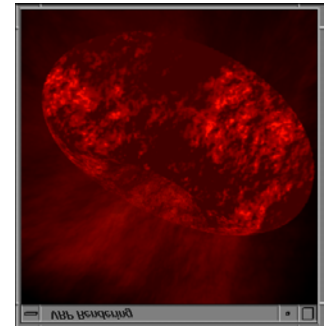
Opacity Textures

- A binary mask, really redefines the geometry.



Color Index Textures

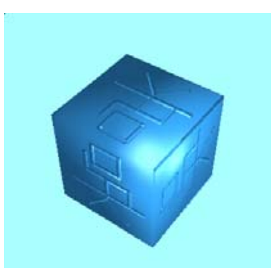
- New Microsoft Extension for 8-bit textures.
- Also some cool new extensions to SGI's OpenGL to perform table look-ups after the texture samples have been computed.

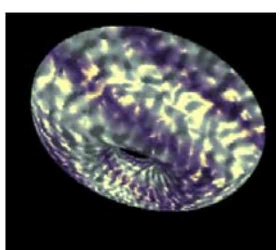


Lao 1998

Bump Mapping

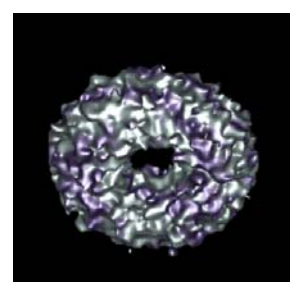
- This modifies the surface normals.
- More on this later.





Displacement Mapping

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



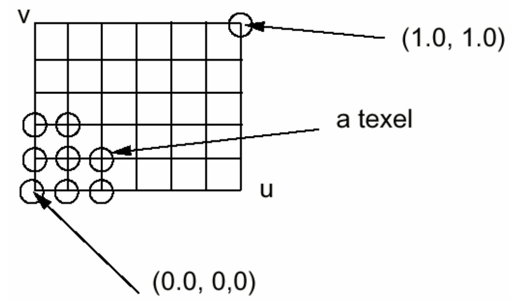
Lao 1998

Reflection Properties

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

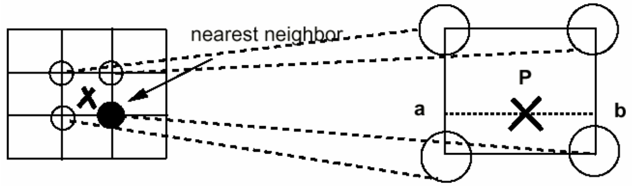
Texture and Texel

- 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

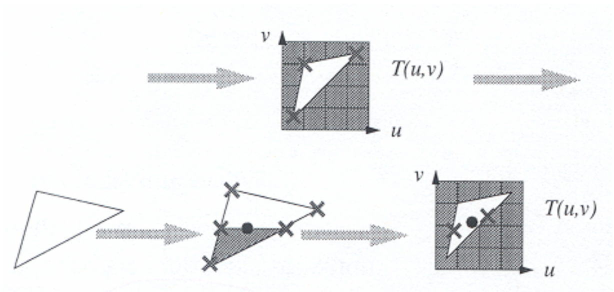


Image space scan

For each scanline, y

For each pixel, x

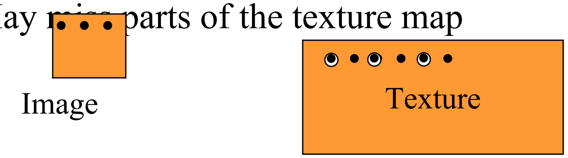
compute u(x,y) and v(x,y)

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

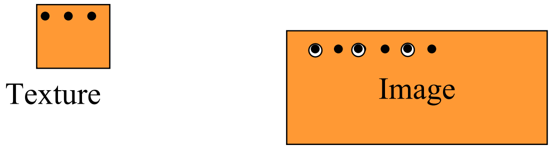
compute x(u,v) and y(u,v)

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)*?

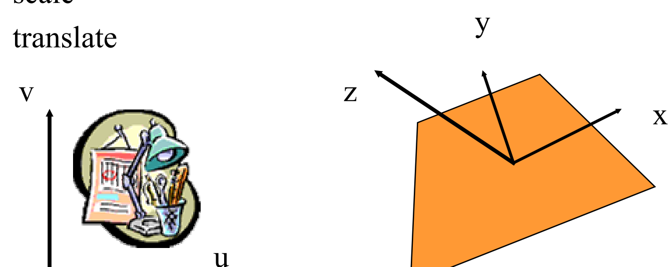
- Higher-order interpolation
  - **F**(u,v) = $\sum_i \sum_j$ **F**[i,j] *h(u,v)*
  - *h(u,v) is called the reconstruction kernel*
    - *Guassian*
    - *Sinc function*
    - *splines*
  - Like linear interpolation, need to find neighbors.
    - Usually four to sixteen

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

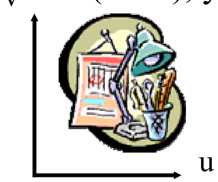
Texture Parameterization

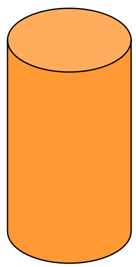
- Mapping to a 3D Plane
  - Simple Affine transformation
    - rotate
    - scale
    - translate



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)



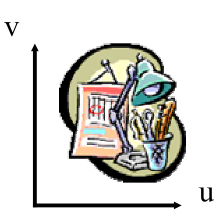


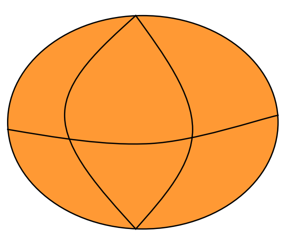
Texture Parameterization

- 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)

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

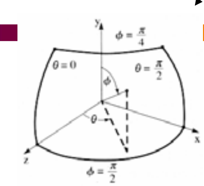
$y(\theta,\phi) = \cos\phi$

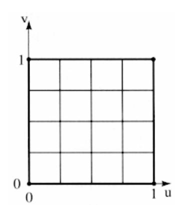
$z(\theta,\phi) = \cos\theta \sin\phi$

$0 \le \theta \le \pi/2$

$\pi/4 \le \phi \le \pi/2$

Part of a sphere



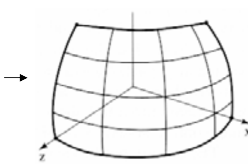


$(u,v) = (0,0) \Leftrightarrow (\theta,\phi) = (0, \pi/2)$

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

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

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



- Setup up surface, define correspondence, and voila!

Example Continued

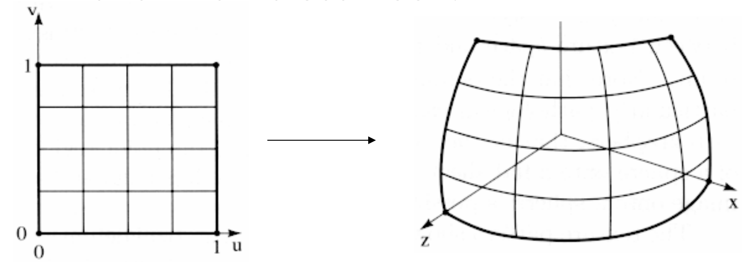
- Can even solve for (θ,ϕ) and (u,v)
  - $A=\pi/2$, $B=0$, $C=-\pi/4$, $D=\pi/2$

[redacted]

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

All Is Not Good

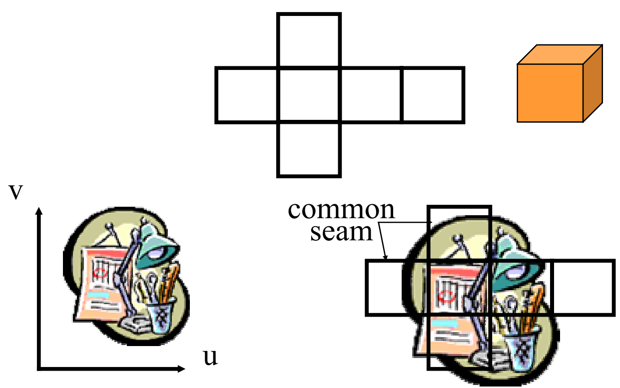
- Let's take a closer look:



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

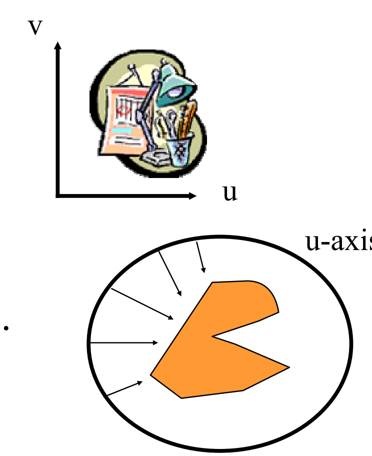
Texture Parameterization

- Mapping to a Cube



Two-pass Mappings

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



S and O Mapping

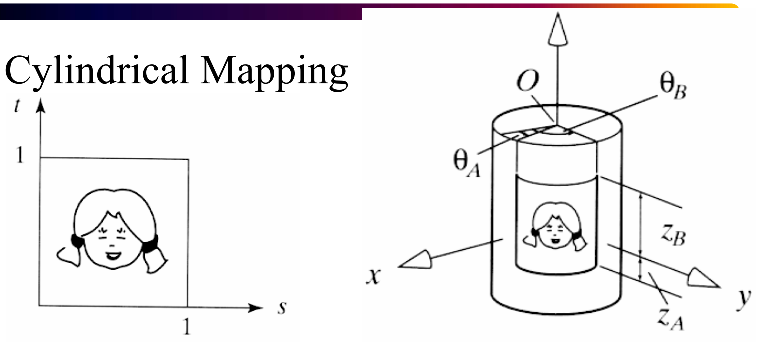
- Pre-distort the texture by mapping it onto a simple surface like a plane, cylinder, sphere, or box
- Map the result of that onto the surface
- Texture → Intermediate is S mapping
- Intermediate → Object is O mapping

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

Texture space ⟶ Intermediate space ⟶ Object space

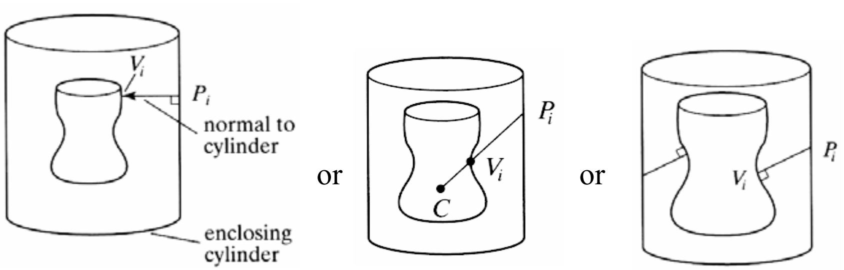
S Mapping Example

- Cylindrical Mapping



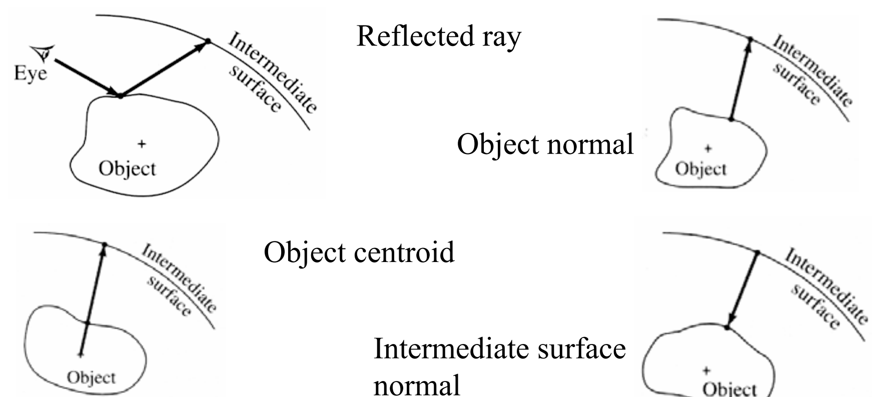
O Mapping

- A method to relate the surface to the cylinder



O Mappings Cont'd

- Bier and Sloan defined 4 main ways



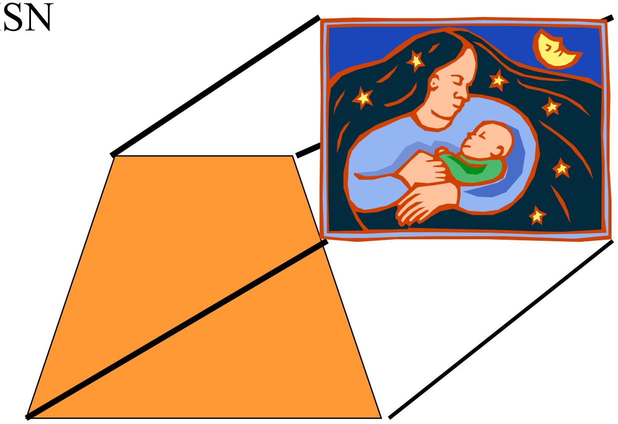
Texture Parameterization

- 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



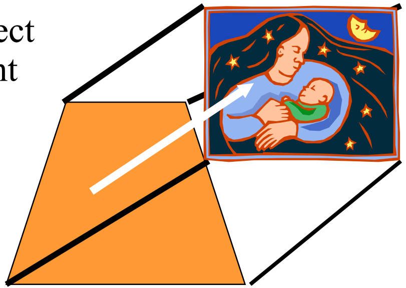
Texture Parameterization

- 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

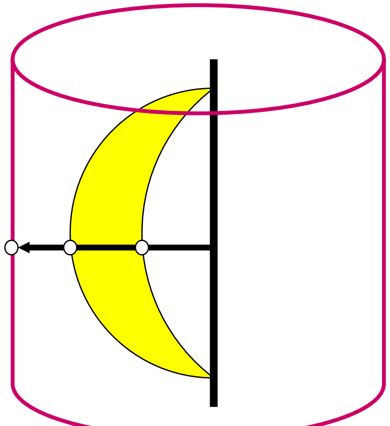
Texture Parameterization

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

Watt

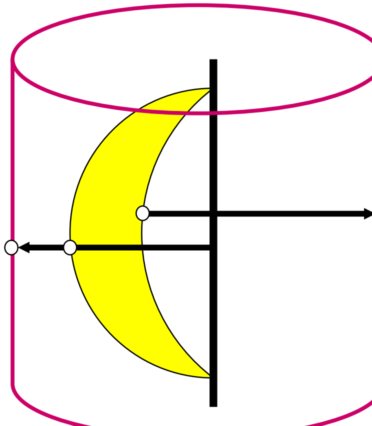
Texture Parameterization

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



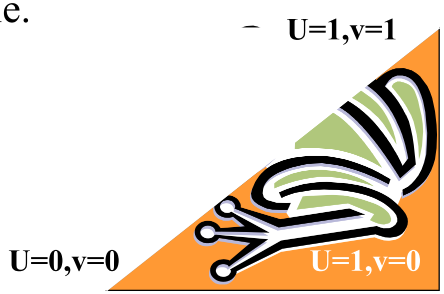
Texture Parameterization

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



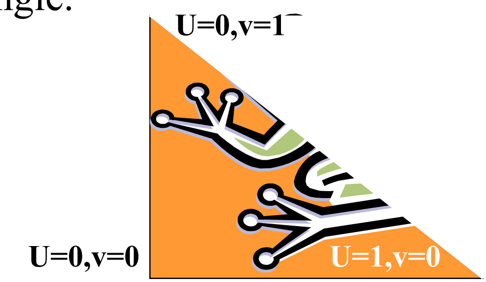
Triangle Mapping

- 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.

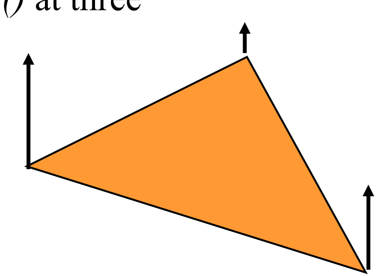


Triangle Mapping

- 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

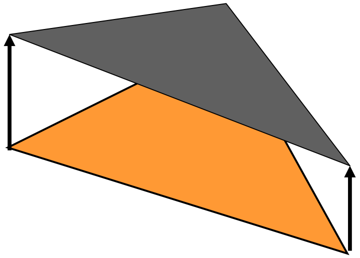
- A linear function in 3D is defined as
  - $f(x,y,z) = Ax + By + Cz + D$
- Note, four points uniquely determine this equation, hence a tetrahedron has a unique linear function through it.
- Taking a slice plane through this gives us a linear function on the plane.

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

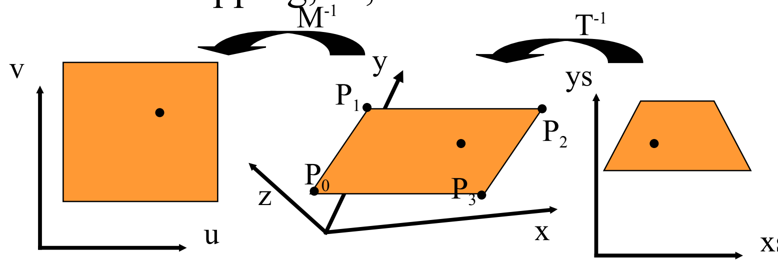
- We get a similar set of equations for *v(x,y,z)*.
- Note, that if the points lie in a plane parallel to the *xz* or *yz*-planes, then *z* is undefined.
- We should then solve the plane equation for *y* or *x*, respectively.
- For robustness, solve the plane equation for the term with the highest coefficient.

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:

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

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

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

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

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

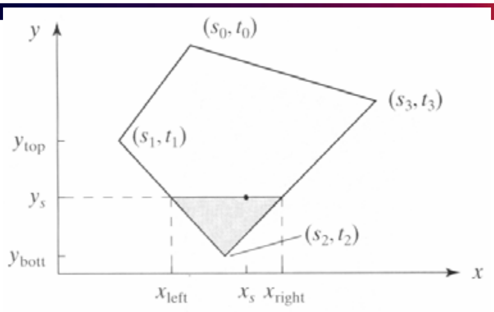
Solve for *s* and *t* using two of the first three equations.

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

Degenerate Solutions

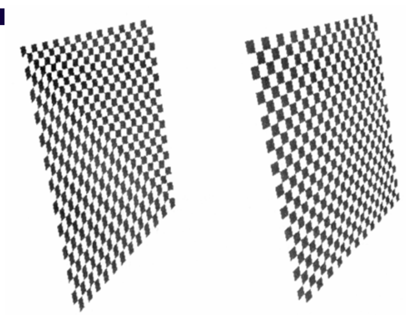
- When mapping a square texture to a rectangle, the solutions will be linear.
  - The quadratic will simplify to a linear equation.
  - s(x,y) = s(x), or s(y).
  - You need to check for these conditions.

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);

OpenGL functions

- *During rendering, give the cartesian coordinates and the texture coordinates for each vertex.*

```
glBegin (GL_QUADS);
    glTexCoord2f (0.0, 0.0);
    glVertex3f (0.0, 0.0, 0.0);
    glTexCoord2f (1.0, 0.0);
    glVertex3f (10.0, 0.0, 0.0);
    glTexCoord2f (1.0, 1.0);
    glVertex3f (10.0, 10.0, 0.0);
    glTexCoord2f (0.0, 1.0);
    glVertex3f (0.0, 10.0, 0.0);
glEnd ();
```

OpenGL Functions

- Nate Miller's pages
  - **OpenGL Texture Mapping : An Introduction**
  - **Advanced OpenGL Texture Mapping**

OpenGL functions

- Automatic texture coordinate generation
  - *Void glTexGenf(coord, pname, param)*
    - *Coord:*
      - *GL_S, GL_T, GL_R, GL_Q*
    - *Pname*
      - *GL_TEXTURE_GEN_MODE*
      - *GL_OBJECT_PLANE*
      - *GL_EYE_PLANE*
    - *Param*
      - *GL_OBJECT_LINEAR*
      - *GL_EYE_LINEAR*
      - *GL_SPHERE_MAP*

OpenGL Texture Mapping

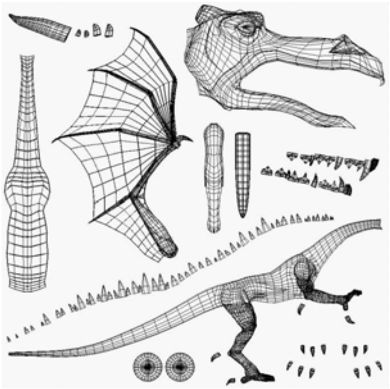
- Add slides on Decal vs. Blend vs. Modulate, …

3D Paint Systems

- Determine a texture parameterization to a blank image.
  - Usually not continuous
  - Form a texture map atlas
- Mouse on the 3D model and paint the texture image.
- Deep Paint 3D demo

Texture Atlas

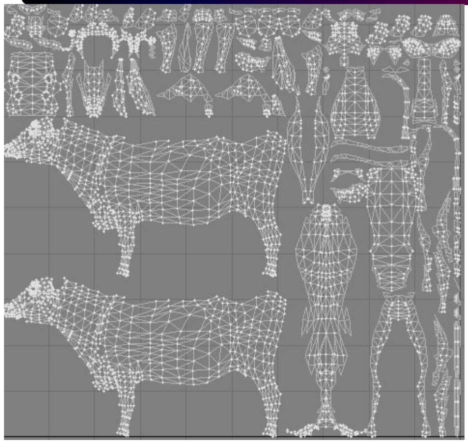
- Find patches on the 3D model
- Place these (map them) on the texture map image.
- Space them apart to avoid neighboring influences.



Texture Atlas

- Add the color image (or bump, …) to the texture map.
- Each polygon, thus has two sets of coordinates:
  - *x,y,z* world
  - *u,v* texture

Example 2



3D Paint

- To interactively paint on a model, we need several things:
  1. Real-time rendering.
  2. Translation from the mouse pixel location to the texture location.
  3. Paint brush style depositing on the texture map.

3D Paint

- Translating from pixel space to texture space.
  - We know a polygon's projection to the image.
  - We know a polygon's projection to the texture.
- The question is, which polygons are covered by our virtual paintbrush?
  - Picking or ray intersections
  - Item Buffers (also called Id- or Object-buffers)

Item Buffers

- If you have less than 2**24 polygons in your object, give each polygon a unique color.
- Turn off shading and lighting.
- Read out the sub-image that the brush covers.
- For each polygon id (and xs,ys,zs position), determine the mapping to texture space and the portion of the brush to paint.

3D Paint

- Problems:
  - Mip-mapping leads to overlapped regions.
  - Large spaces in the texture map to avoid overlap really wastes texture space.
  - A common vertex may need to be repeated with different texture coordinates.

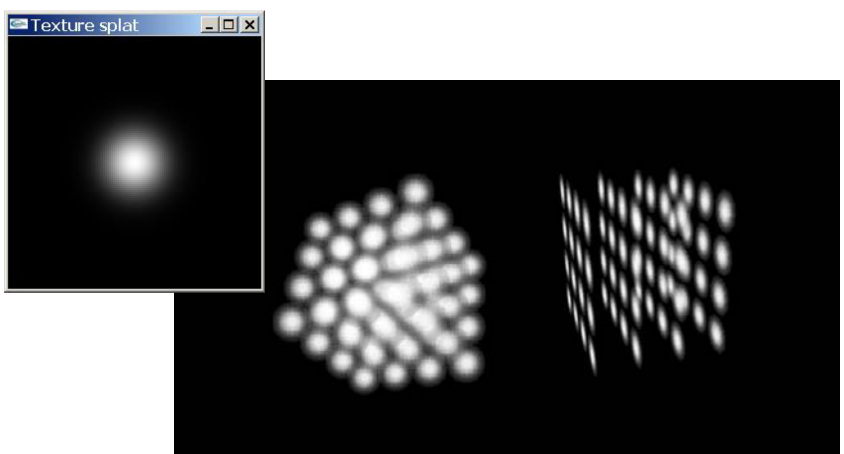
Sprites and Billboards

- Sprites – usually refer to 2D animated characters that move across the screen.
  - Like Pacman 
- Three types (or styles) of billboards
  - Screen-aligned (parallel to top of screen)
  - World aligned (allows for head-tilt)
  - Axial-aligned (not parallel to the screen)

Creating Billboards in OpenGL

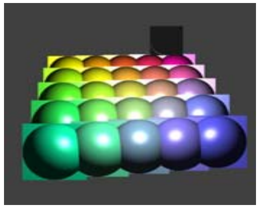
- Annotated polygons do not exist with OpenGL 1.3 directly.
- If you specify the billboards for one viewing direction, they will not work when rotated.

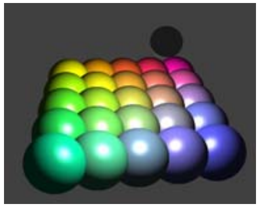
Example



Example 2

- The alpha test is required to remove the background.
- More on this example when we look at depth textures.



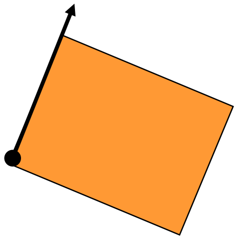


Re-orienting

- Billboards need to be re-oriented as the camera moves.
- This requires immediate mode (or a vertex shader program).
- Can either:
  - Recalculate all of the geometry.
  - Change the transformation matrices.

Re-calculating the Geometry

- Need a projected point (say the lower-left), the projected up-direction, and the projected scale of the billboard.
- Difficulties arise if we are looking directly at the ground plane.

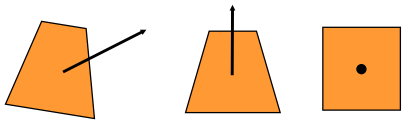


Undo the Camera Rotations

- Extract the projection and model view matrices.
- Determine the *pure rotation* component of the combined matrix.
- Take the inverse.
- Multiply it by the current model-view matrix to undo the rotations.

Screen-aligned Billboards

- Alternatively, we can think of this as two rotations.
- First rotate around the *up*-vector to get the normal of the billboard to point towards the eye.
- Then rotate about a vector perpendicular to the new normal orientation and the new *up*-vector to align the top of the sprite with the edge of the screen.
- This gives a more spherical orientation.
  - Useful for placing text on the screen.



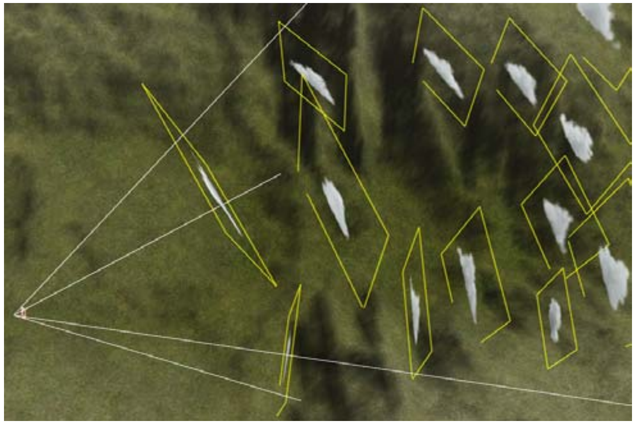
World Aligned Billboards

- Allow for a final rotation about the eye-space *z*-axis to orient the billboard towards some world direction.
- Allows for a head tilt.

Example

Lastra

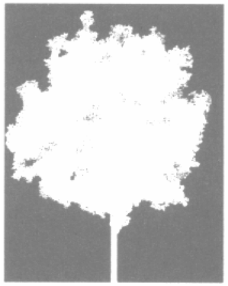
Example



Lastra

Axial-Aligned Billboards

- The *up*-vector is constrained in world-space.
- Rotation about the up vector to point normal towards the eye as much as possible.
- Assuming a ground plane, and always perpendicular to that.
- Typically used for trees.

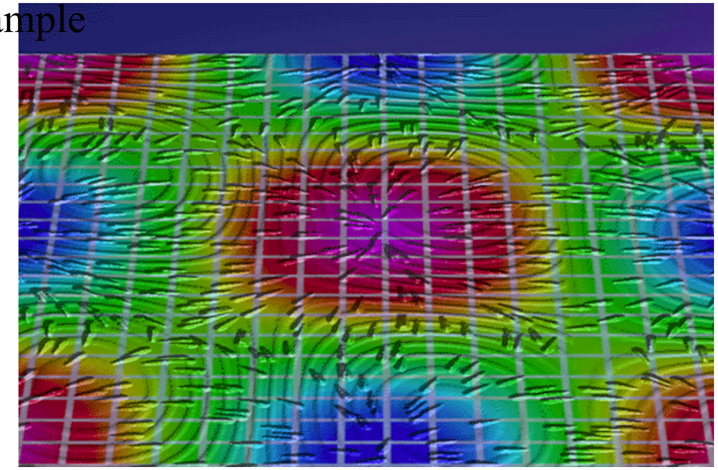


Bump Mapping

- 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.

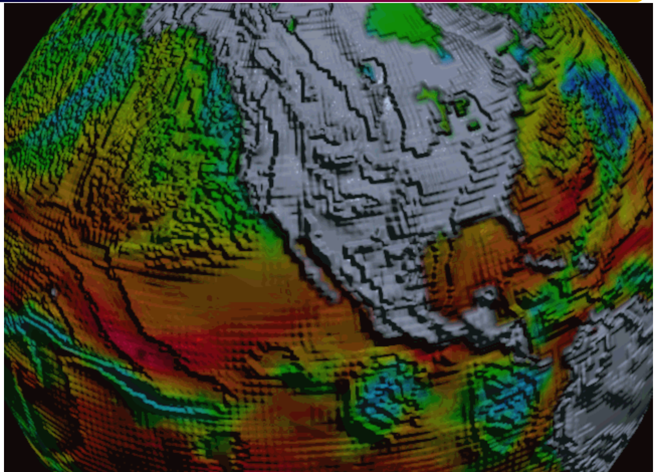
Bump Mapping

- Example



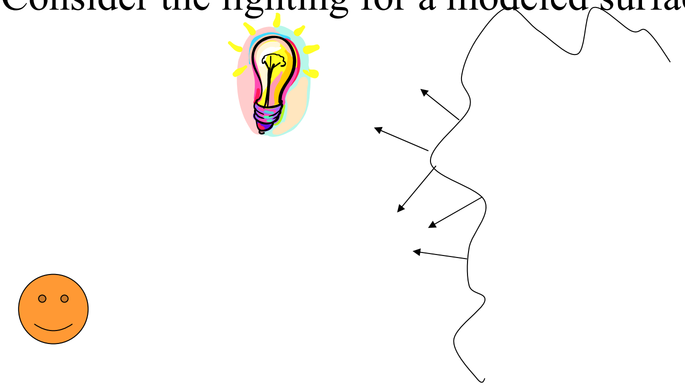
Crawfis 1991

Bump Mapping



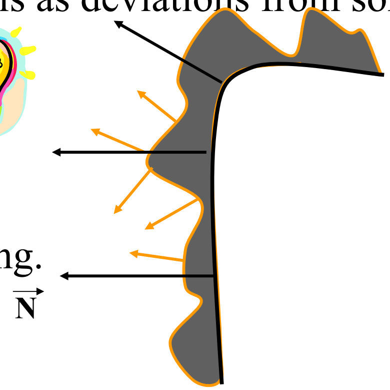
Bump Mapping

- Consider the lighting for a modeled surface.



Bump Mapping

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



Bump Mapping

- Assumption: small deviations in the normal direction to the surface.

$$\boxed{\vec{\mathbf{X}} = \vec{\mathbf{X}} + \mathrm{B}\,\vec{\mathbf{N}}}$$

Where B is defined as a 2D function parameterized over the surface:

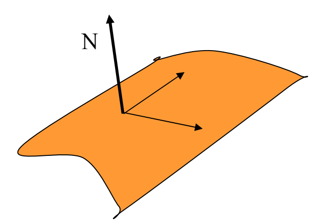
$$B = f(u,v)$$

Bump Mapping

- 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 $\vec{\mathbf{O}}(u,v)$

Bump Mapping

- Define the tangent plane to the surface at a point (u,v) by using the two vectors O_u and O_v, resulting from the partial derivatives.
- The normal is then given by:
  - $\mathbf{N} = \mathbf{O_u} \times \mathbf{O_v}$



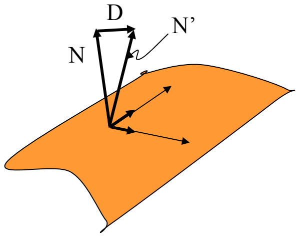
Bump Mapping

- The new surface positions are then given by:
  - $\mathbf{O}'(u,v) = \mathbf{O}(u,v) + B(u,v)\,\mathbf{N}$
  - **Where,** $\mathbf{N} = \mathbf{N} / |\mathbf{N}|$
- Differentiating leads to:
  - $\mathbf{O}'_u = \mathbf{O}_u + B_u\,\mathbf{N} + B\,(\mathbf{N})_u \quad \approx \mathbf{O}'_u = \mathbf{O}_u + B_u\,\mathbf{N}$
  - $\mathbf{O}'_v = \mathbf{O}_v + B_v\,\mathbf{N} + B\,(\mathbf{N})_v \quad \approx \mathbf{O}'_v = \mathbf{O}_v + B_v\,\mathbf{N}$

  If B is small.

Bump Mapping

- This leads to a new normal:
  - $\mathbf{N}'(u,v) = \mathbf{O}_u \times \mathbf{O}_v - B_u(\mathbf{N} \times \mathbf{O}_v) + B_v(\mathbf{N} \times \mathbf{O}_u) + B_u B_v(\mathbf{N} \times \mathbf{N})$
  - $= \mathbf{N} - B_u(\mathbf{N} \times \mathbf{O}_v) + B_v(\mathbf{N} \times \mathbf{O}_u)$
  - $= \mathbf{N} + \mathbf{D}$



Bump Mapping

- For efficiency, can store B_u and B_v in a 2-component texture map.
  - This is commonly called a *offset vector* map.
  - Note: It is oriented in tangent-space, not normal space.
- The cross products are geometry terms only.
- **N'** will of course need to be normalized after the calculation and before lighting.
  - This floating point square root and division makes it difficult to embed into hardware.

Bump Mapping

- An alternative representation of bump maps can be viewed as a *rotation* of the normal.
- The rotation axis is the cross-product of N and N'.

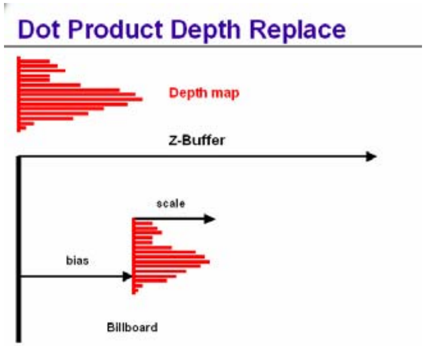
$$\vec{A} = \vec{N} \otimes \vec{N}' = \vec{N} \otimes \left(\vec{N} + \vec{D}\right) = \vec{N} \otimes \vec{D}$$

Bump Mapping

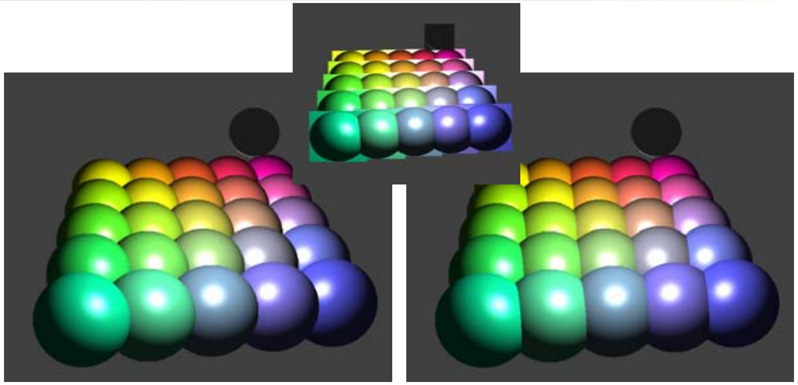
- We can store:
  - The height displacement
  - The offset vectors in tangent space
  - The rotations in tangent space
    - Matrices
    - Quaternians
    - Euler angles
- Object dependent versus reusable.

Z Texture

- GeForce 3 allows pseudo-depth textures to get rid of the smoothness of the bump-mapped surface silhouettes.



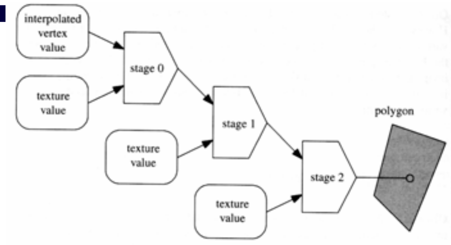
Imposters with Depth



Multi-texture

- Originally you would send the geometry down, transform it, shade it, texture it, and THEN blend it with whatever is in the framebuffer
- Multi-texture keeps the geometry and applies more texture operations before it dumps it to the framebuffer.

Multitexture II



- Rasterization is even more important now!
  - Doubling pixels will result in bright spots
- Be careful the order in which you blend

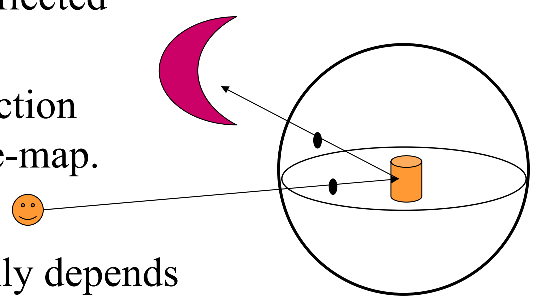
Blending Example

- From Kenny
- Given: Polygons A, B, C; polygon opacity factors: K_A, K_B, K_C; and polygon intensities (perhaps RGB triplets: I_A, I_B, I_C)
- rI_K = resulting intensity at polygon K
- $rI_A = (1-K_A)I_A + K_A rI_B$
  $rI_B = (1-K_B)I_B + K_B rI_C$
  $rI_A = (1-K_A)I_A + K_A[(1-K_B)I_B + K_B I_C]$ <- A to B to C
- $rI_B = (1-K_B)I_B + K_B[(1-K_A)I_A + K_A I_C$ <- B to A to C

- $I_A = 1,\ I_B = 0.5,\ K_A = 0.2,\ K_B = 0.5 : (I_C = 1)$
  $rI_A = (1 - 0.2)(1) + (0.2)[(1 - 0.5)(0.5) + (0.5)(1)] =$ **0.95**
  $rI_B = (1 - 0.5)(0.5) + (0.5)[(1 - 0.2)(1) + (0.2)(1)] =$ **0.75**
- Blending order matters! Should sort!

Environment Mapping

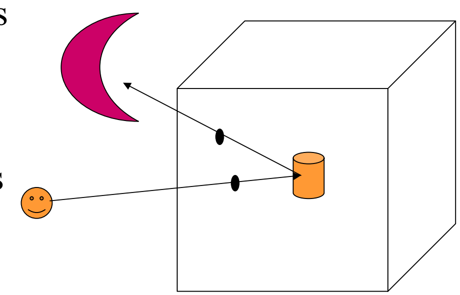
- Determine reflected ray.
- Look-up direction from a sphere-map.



- Reflection only depends on the direction, not the position.

Environment Mapping

- We can also encode the reflected directions using several other formats.
- Greene, et al suggested a cube. This has the advantage that it can be constructed by six normal renderings.

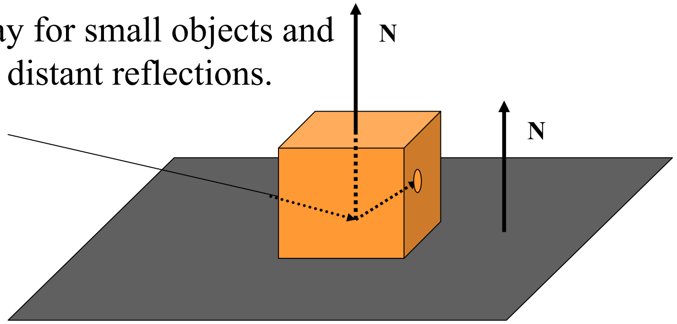


Environment Mapping

- 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.

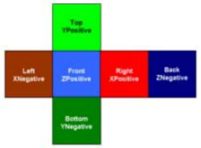
Environment Mapping

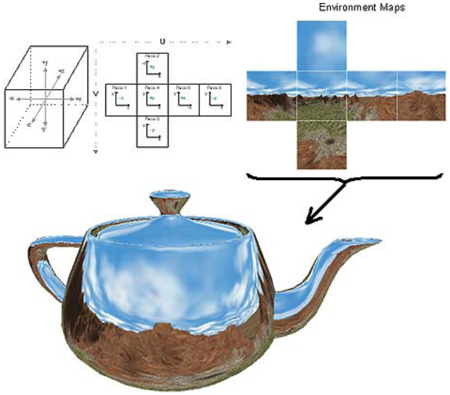
- Problems:
  - Reflection is about object's centroid.
  - Okay for small objects and and distant reflections.



Environment Mapping

- Cube Mapping





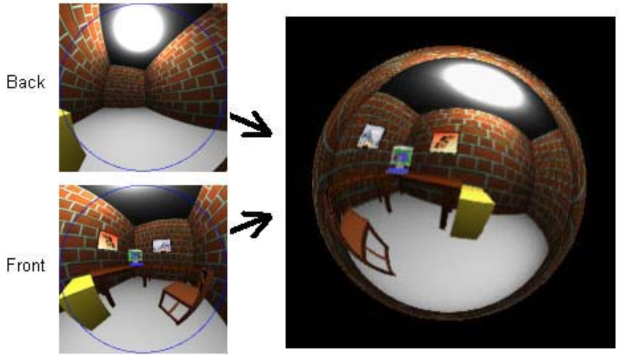
Cubic Environment Mapping

Environment Mapping

- Sphere mapping
  - Unpeel the sphere, such that the outer radius of the circle is the back part of the sphere

Environment Mapping

- Dual Paraboloid
  - Multi-textured or multi-pass



Environment Mapping

- Applications
  - Specular highlights
  - Multiple light sources
  - Reflections for shiny surfaces
  - Irradiance for diffuse surfaces

Chrome Mapping

- Cheap environment mapping
- Material is very glossy, hence perfect reflections are not seen.
- Index into a pre-computed view independent texture.
- Reflection vectors are still view dependent.

Chrome Mapping

- Usually, we set it to a very blurred landscape image.
  - Brown or green on the bottom
  - White and blue on the top.
  - Normals facing up have a white/blue color
  - Normals facing down on average have a brownish color.

Chrome Mapping

- Also useful for things like fire.
- The major point, is that it is not important what actually is shown in the reflection, only that it is view dependent.

CIS 781

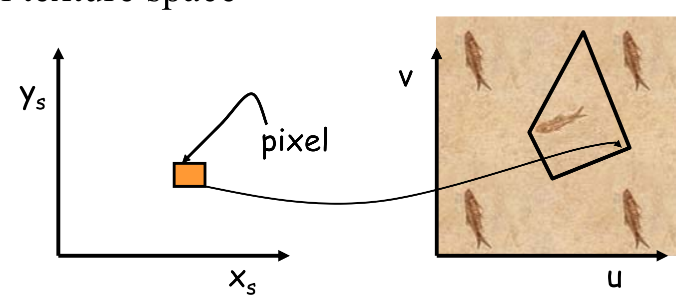
Anti-aliasing for Texture Mapping

Quality considerations

- So far we just mapped one point
  - results in bad aliasing (resampling problems)
- We really need to integrate over polygon
- Super-sampling is not a very good solution
  - Dependent on area of integration.
  - Can be quit large for texture maps.
- Most popular (easiest) - mipmaps

Quality considerations

- Pixel area maps to “weird” (warped) shape in texture space



Quality Considerations

- We need to:
  - Calculate (or approximate) the integral of the texture function under this area
  - Approximate:
    - Convolve with a wide filter around the center of this area
    - Calculate the integral for a similar (but simpler) area.

Quality Considerations

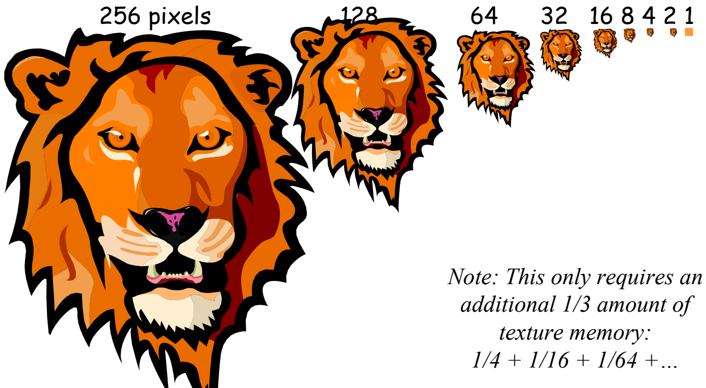
- The area is typically approximated by a rectangular region (found to be good enough for most applications)
- Filter is typically a box/averaging filter - other possibilities
- How can we pre-compute this?

Mip-maps

- Mipmapping was invented in 1983 by Lance Williams
  - Multi in parvo "many things in a small place"

Mip-maps

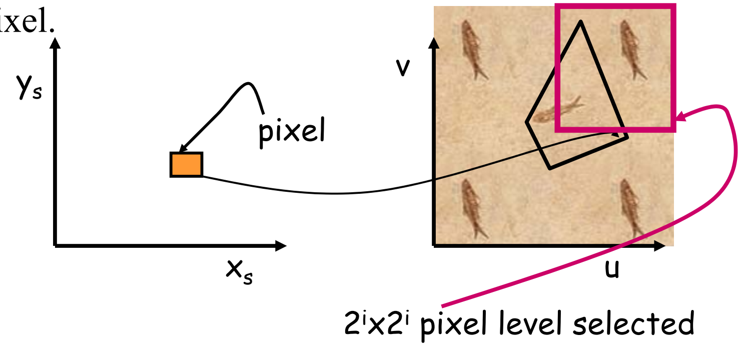
- An image-pyramid is built.



Note: This only requires an additional 1/3 amount of texture memory:
1/4 + 1/16 + 1/64 +…

Mip-maps

- Find level of the mip-map where the area of each mip-map pixel is closest to the area of the mapped pixel.



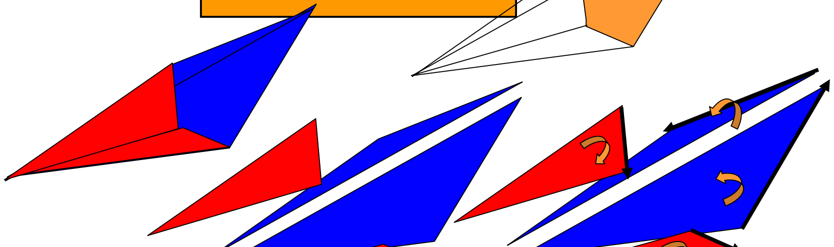
Mip-maps

- Mip-maps are thus indexed by *u, v,* and the level, or amount of *compression*, *d*.
- The compression amount, *d,* will change according to the *compression* of the texels to the pixels, and for mip-maps can be approximated by:
  - *d* = *sqrt*(Area of pixel in *uv*-space)
  - The *sqrt* is due to the assumption of uniform compression in mip-maps.

Review: Polygon Area

- Recall how to calculate the area of a 2D polygon (in this case, the quadrilateral of the mapped pixel corners).

$$A = \frac{1}{2}\sum_{i=0}^{n-1}\left(x_i y_{i+1} - x_{i+1} y_i\right)$$

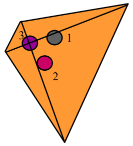


Mip-maps

- William's algorithm
  - Take the difference between neighboring *u* and *v* coordinates to approximate derivatives across a screen step in *x* or *y*
  - Derive a mipmap level from them by taking the maximum distortion
    - Over-blurring

Mip-maps

- The texel location can be determined to be either:
  1. The *uv*-mapping of the pixel center.
  2. The average *u* and *v* values from the projected pixel corners (the centroid).
  3. The diagonal crossing of the projected quadrilateral.



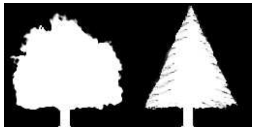
- However, there are only so many mip-map centers.

Mip-maps

- Pros
  - Easy to calculate:
    - Calculate pixels area in texture space
    - Determine mip-map level
    - Sample or interpolate to get color
- Cons
  - Area not very close – restricted to square shapes (64x64 is far away from 128x128).
  - Location of area is not very tight - shifted.

Note on Alpha

- Alpha can be averaged just like rgb for texels
- Watch out for borders though if you interpolate



Specifying Mip-map levels

- OpenGL allows you to specify each level individually (see *glTexImage2D* function).
- The GLU routine *gluBuild2Dmipmaps()* routine offers an easy interface to averaging the original image down into its mip-map levels.
- You can (and probably should) recalculate the texture for each level.

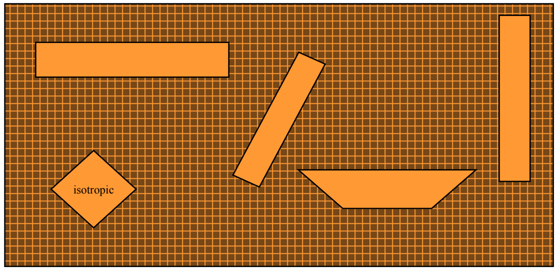
Warning: By default, the filtering assumes mip-mapping. If you do not specify all of the mip-map levels, your image will probably be black.

Higher-levels of the Mip-map

- Two considerations should be made in the construction of the higher-levels of the mip-map.
    1. Filtering – simple averaging using a box filter, apply a better low-pass filter.
    2. Gamma correction – by taking into account the perceived brightness, you can maintain a more consistent effect as the object moves further away.

Anisotropic Filtering

- A pixel may rarely project onto texture space affinely.
- There may be large distortions in one direction.



Anisotropic Filtering

- Multiple mip-maps or *Ripmaps*
- Summed Area Tables (SAT)
- Multi-sampling for anisotropic texture filtering.
- EWA filter

Ripmaps

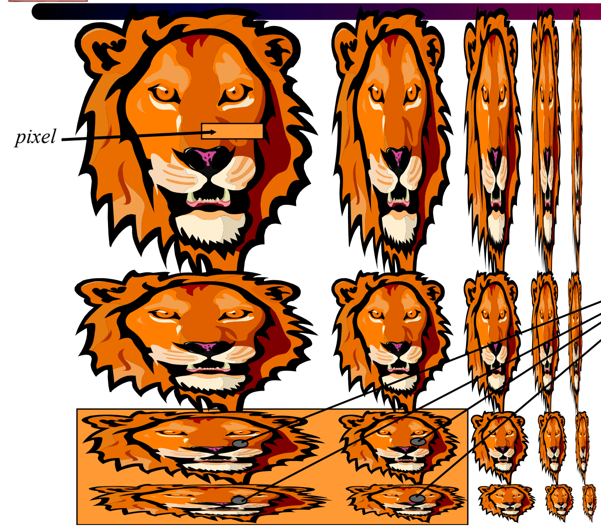
- Scale by half by x across a row.
- Scale by half in y going down a column.
- The diagonal has the equivalent mip-map.

- Four times the amount of storage is required.

Ripmaps

- To use a ripmap, we use the pixel's extents to determine the appropriate compression ratios.
- This gives us the four neighboring maps from which to sample and interpolate from.

Ripmaps



Compression in *u* is 1.7
Compression in *v* is 6.8

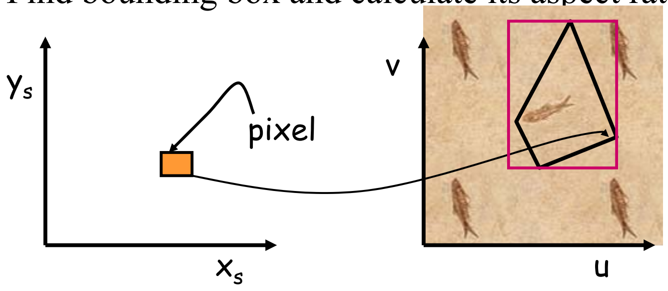
Determine weights from each sample

Summed Area Table (SAT)

- Use an axis aligned rectangle, rather than a square
- Pre-compute the sum of all texels to the left and below for each texel location
  - For texel (u,v), replace it with:
    sum (texels($i=0\ldots u, j=0\ldots v$))

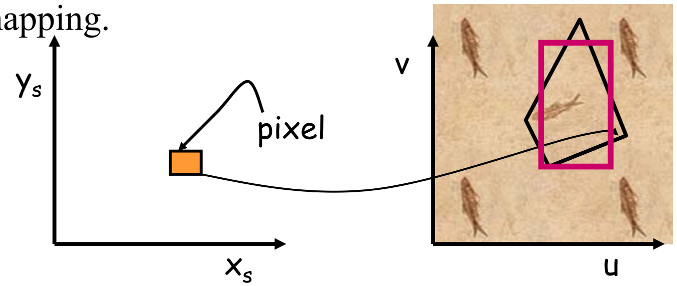
Summed Area Table (SAT)

- Determining the rectangle:
  - Find bounding box and calculate its aspect ratio



Summed Area Table (SAT)

- Determine the rectangle with the same aspect ratio as the bounding box and the same area as the pixel mapping.

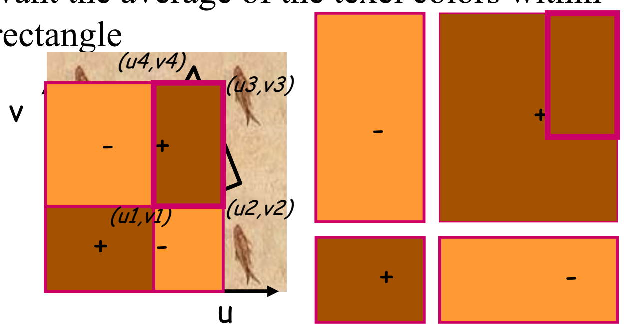


Summed Area Table (SAT)

- Center this rectangle around the bounding box center.
- Formula:
  - Area = aspect_ratio*x*x
  - Solve for x – the width of the rectangle
- Other derivations are also possible using the aspects of the diagonals, …

Summed Area Table (SAT)

- Calculating the color
  - We want the average of the texel colors within this rectangle



Summed Area Table (SAT)

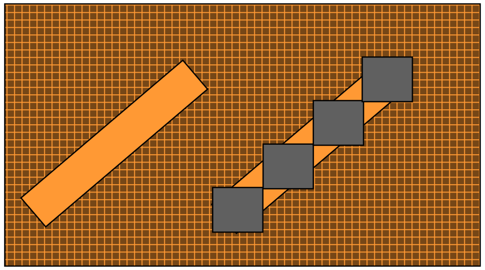
- To get the average, we need to divide by the number of texels falling in the rectangle.
  - Color = SAT(u3,v3)-SAT(u4,v4)-SAT(u2,v2)+SAT(u1,v1)
  - Color = Color / ((u3-u1)*(v3-v1))
- This implies that the values for each texel may be very large:
  - For 8-bit colors, we could have a maximum SAT value of 255*nx*ny
  - 32-bit pixels would handle a 4kx4k texture with 8-bit values.
  - RGB images imply 12-bytes per pixel.

Summed Area Table (SAT)

- Pros
  - Still relatively simple
    - Calculate four corners of rectangle
    - 4 look-ups, 5 additions, 1 multiply and 1 divide.
  - Better fit to area shape
  - Better overlap
- Cons
  - Large texel SAT values needed.
  - Still not a perfect fit to the mapped pixel.
  - The divide is expensive in hardware.

Anisotropic Mip-mapping

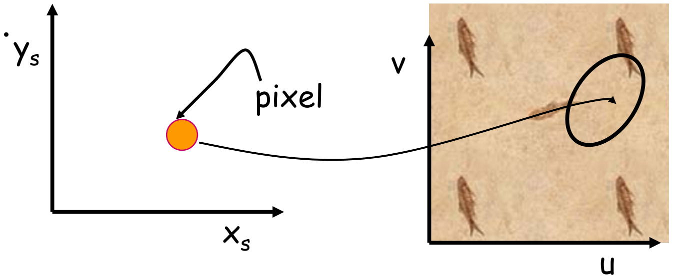
- Uses parallel hardware to obtain multiple mip-map samples for a fragment.
- A lower-level of the mip-map is used.
- Calculate *d* as the minimum length, rather than the maximum.



Anisotropic Mip-mapping

Elliptical Weighted Average (EWA) Filter

- Treat each pixel as circular, rather than square.
- Mapping of a circle is elliptical in texel space.



EWA Filter

- Precompute?
- Can use a better filter than a box filter.
- Heckbert chooses a Gaussian filter.

EWA Filter

- Calculating the Ellipse
- Scan converting the Ellipse
- Determining the final color (normalizing the value or dividing by the weighted area).

EWA Filter

- Calculating the ellipse
  - We have a circular function defined in (x,y).
  - Filtering that in texture space h(u,v).
  - (u,v) = T(x,y)
  - Filter: h(T(x,y))

EWA Filter

- Ellipse:
  - $\phi(u,v) = Au^2 + Buv + Cv^2 = F$
  - (u,v) = (0,0) at center of the ellipse
    - $A = v_x^2 + v_y^2$
    - $B = -2(u_x v_y + u_y v_x)$
    - $C = u_x^2 + u_y^2$
    - $F = u_x v_y + u_y v_x$

EWA Filter

- Scan converting the ellipse:
  - Determine the bounding box
  - Scan convert the pixels within it, calculating $\phi(u,v)$.
  - If $\phi(u,v) < F$, weight the underlying texture value by the filter kernel and add to the sum.
  - Also, sum up the filter kernel values within the ellipse.

EWA Filter

- Determining the final color
  - Divide the weighted sum of texture values by the sum of the filter weights.

EWA Filter

- What about large areas?
  - If m pixels fall within the bounding box of the ellipse, then we have $O(n^2m)$ algorithm for an nxn image.
  - m maybe rather large.
- We can apply this on a mip-map pyramid,rather than the full detailed image.
  - Tighter-fit of the mapped pixel
  - Cross between a box filter and gaussian filter.
  - Constant complexity - $O(n^2)$

CIS 781

Procedural and Solid Textures

Procedural Textures

- Introduced by Perlin and Peachey (Siggraph 1989)
- Look for book by Ebert et al: V "Texturing and Modeling: A Procedural Approach"
- It's a 3D texturing approach (can be used in 2D of course)

Procedural Textures

- Gets around a bunch of problems of 2D textures
  - Deformations/compressions
  - Worrying about topology
  - Excessively large texture maps
- In 3D, analogous to sculpting or carving

3D Texture Mapping

- Much simpler than 2D texture mapping:
  - $u = x$
  - $v = y$
  - $w = z$

Procedural Textures

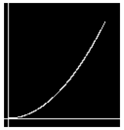
- 2D Brick
- 1D sin-wave example: (Excel spreadsheet)

Procedural Textures

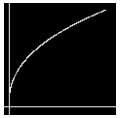
- Object Density Function D(x)
  - defines an object, e.g. implicit description or inside/outside etc.
- Density Modulation Function (DMF) f_i
  - position dependent
  - position independent
  - geometry dependent
- Hyper-texture:
  $H(D(x),x) = f_n(\ldots f_2(f_1(D(x))))$

Procedural Textures

- Base DMF's:
  - bias



b = 0.25



b = 0.75

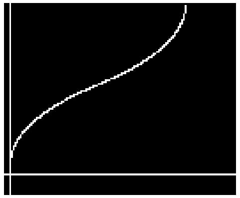
- used to bend the Density function either upwards or downwards over the [0,1] interval. The rules the bias function has to follow are:
  bias(b,0)=0
  bias(b,.5)=b
  bias(b,1)=1
- The following function exhibits those properties:
- bias(b,t) = t^(ln(b)/ln(0.5))

Procedural Textures

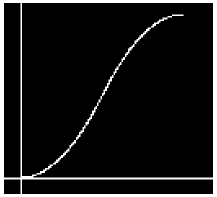
- Gain
  - The gain function is used to help shape how fast the midrange of an objects soft region goes from 0 to 1. A higher gain value means the a higher rate in change. The rules of the gain function are as follows:
    gain(g,0)=0
    gain(g,1/4)=(1-g)/2
    gain(g,1/2)=1/2
    gain(g,3/4)=(1+g)/2
    gain(g,1)=1

Procedural Textures

- Gain
  - The gain function is defined as a spline of two bias curves: gain(g,t)= if (t<0.5) then bias(1-g,2*t) else 1-bias(1-g,2-2*2t)/2



G = 0.25



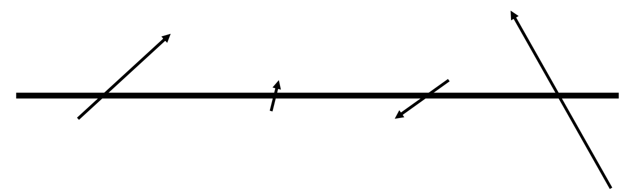
G = 0.75

Procedural Textures

- Noise
  - some strange realization that gives smoothed values between -1 and 1
  - creates a random gradient field G[i,j,k] (using a 3 step monte carlo process separate for each coordinate)

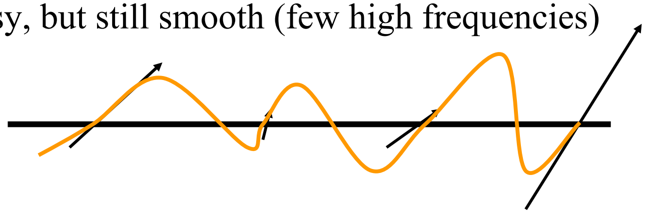
Noise

- Set all integer lattice values to zero
- Randomly assign gradient (tangent) vectors



Simple Noise

- Hermite spline interpolation
- Oscillates about once per coordinate
- Noisy, but still smooth (few high frequencies)



Procedural Textures

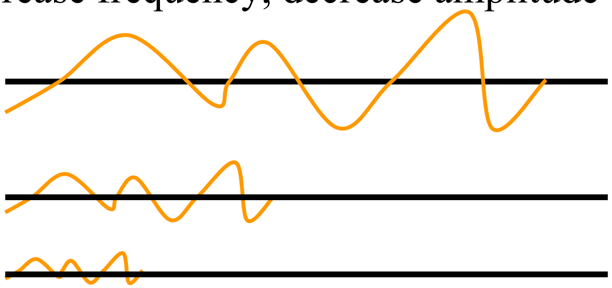
- Noise
  - for an entry (x,y,z) - he does a cubic interpolation step between the dotproduct of G and the offset of the 8 neighbors of G of (x,y,z):

Procedural Textures

- turbulence
  - creates “higher order” noise - noise of higher frequency, similar to the fractal brownian motion:

Turbulence

- Increase frequency, decrease amplitude

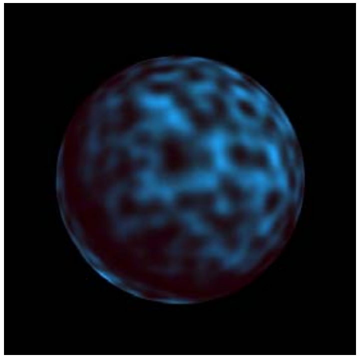


Turbulence

- The abs() adds discontinuities in the first derivative.
- Limit the sum to the Nyquist limit of the current pixel sampling.

Procedural Textures

- Effects (on colormap):
  - noise:



Procedural Textures

- Effects (on colormap):
  - sum 1/f(noise):

Procedural Textures

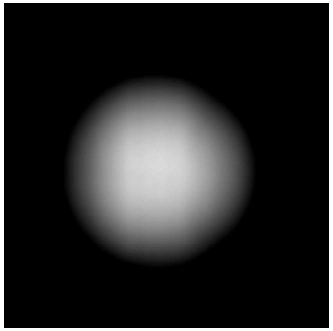
- Effects (on colormap):
  - sum 1/f(|noise|):

Procedural Textures

- Effects (on colormap):
  - sin(x + sum 1/f(|noise|)):

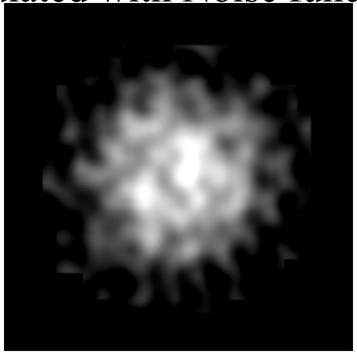
Density Function Models

- Radial Function (2D Slice)



Density Function Models

- Modulated with Noise function.

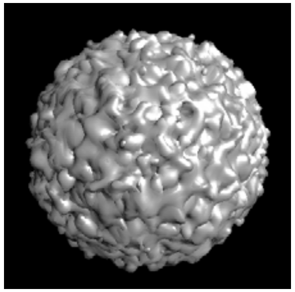


Density Function Models

- Thresholded (iso-contour or step function).

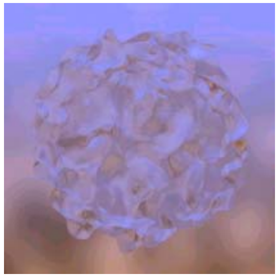
Density Function Models

- Volume Rendering of Hypertextured Sphere

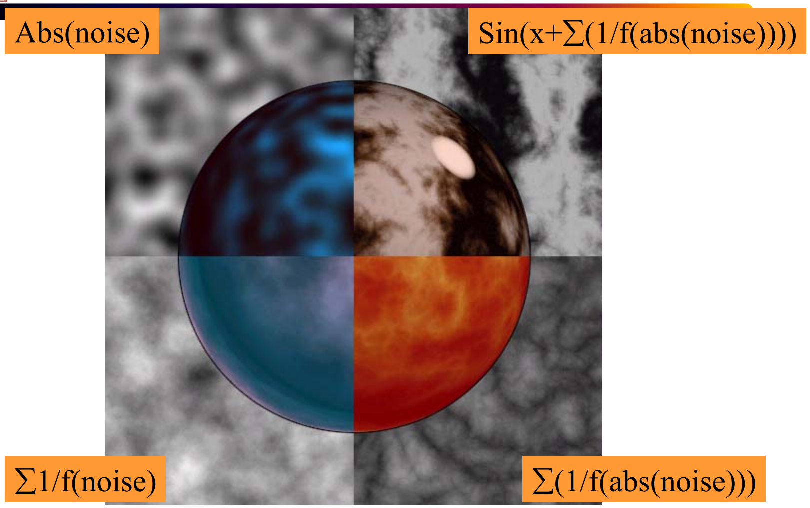


Procedural Textures

- Effects:
  - noisy sphere
    - modify amplitude/frequency
    - (Perlins fractal egg)

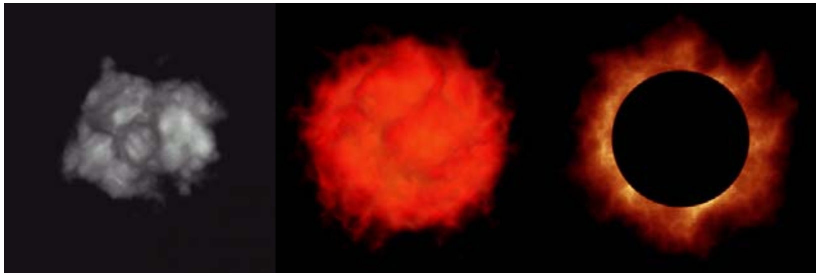


Procedural Textures



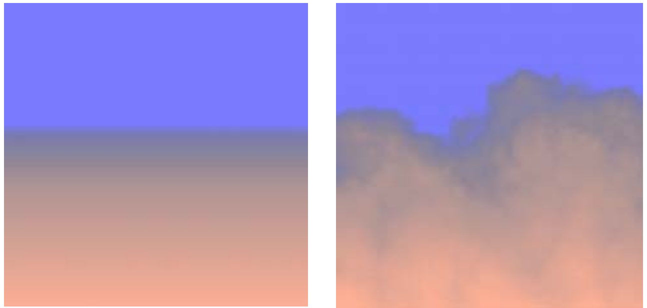
Procedural Textures

- Effects:
  - marble
    - marble(x) = m_color(sin(x+turbulence(x)))
  - fire



Procedural Textures

- Effects:
  - clouds
    - noise translates in x,y



Procedural Textures

- Many other effects!!
  - Wood,
  - fur,
  - facial animation,
  - etc.

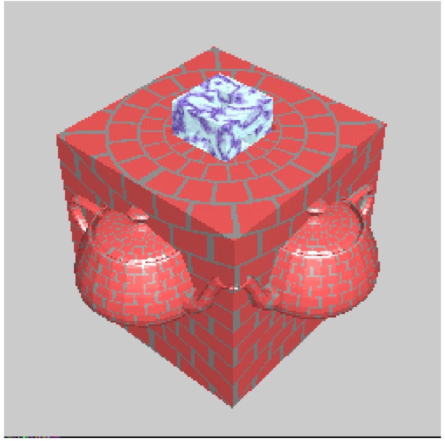
Procedural Textures

- Rendering
  - solid textures
    - keeps original surface
    - map (x,y,z) to (u,v,w)
  - hypertexture
    - changes surface as well (density function)
    - volume rendering approach
    - I.e. discrete ray caster

Simple mesh for tile.

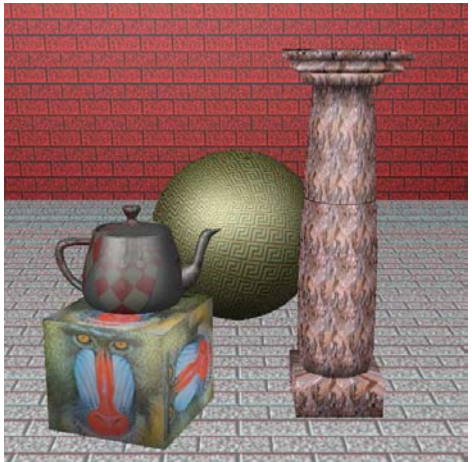
So-so marble

Brick with mortar



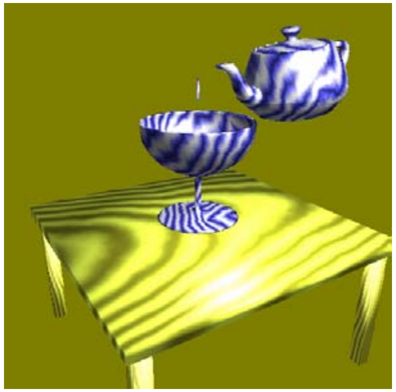
Better marble

More bricks and other texture



Simple Wood

Better Wood

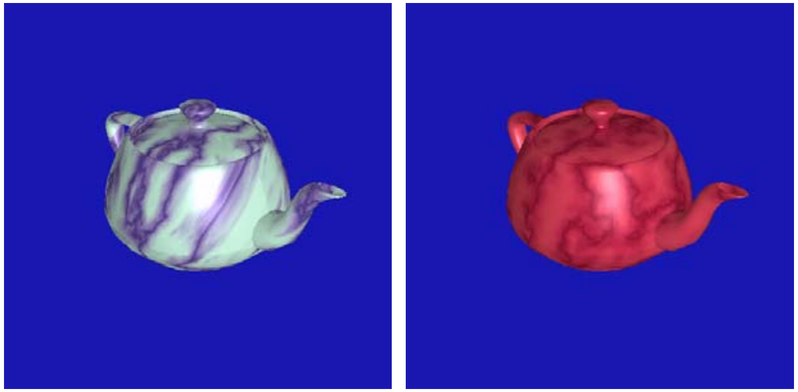


Better marble
as well.

Decent Marble

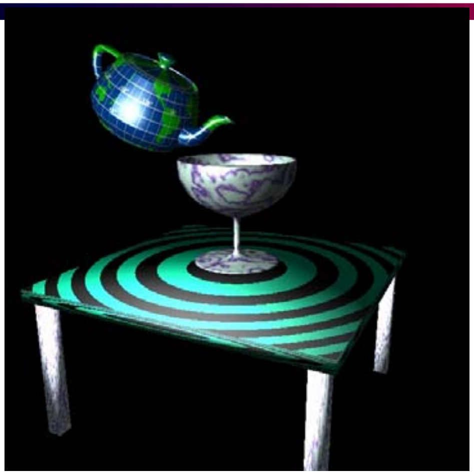
Not so good fire.

More Marble



Decent Fire

More examples



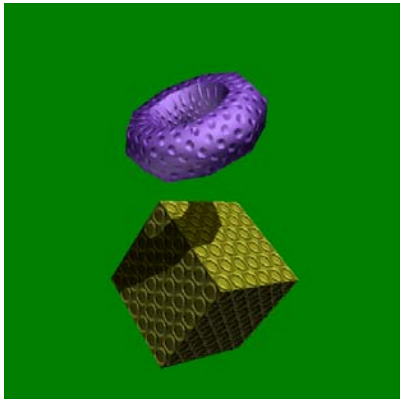
Bump Mapping

- Procedurally bump mapped object

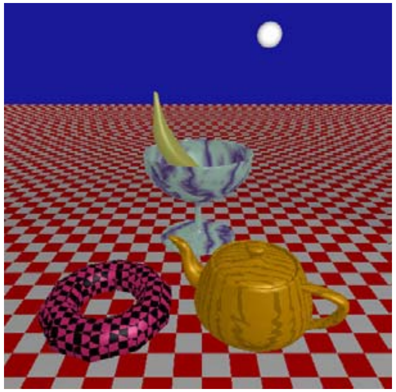
Bump Mapping

- Bump map based on a simple image or procedure. Cylindrical texture space used.

More examples



Anti-aliasing?



White noise

More Scene Lay-outs