Texture Maps

Map an image onto a surface

- Need
  - texture
  - attribute
  - a map
- Texture
  - always rectangular
  - usually an image (1,2,3 or 4 channels)
  - may be a vector field
  - may be a coordinate frame
- Attribute to be modified:
  - texture RGB $\Rightarrow$ screen (e.g., mapping image onto TV screen)
  - texture RGB $\Rightarrow K_d \Rightarrow$ normal lighting
  - texture I $\Rightarrow$ scales K_d
  - texture A sets $\Rightarrow$ surface alpha (billboarding clouds)
  - texture RGBA $\Rightarrow K_d$, surface alpha (billboarding trees, text in space)
  - texture RGBA $\Rightarrow K_d$, texture alpha (decal on surface)
  - texture height map $\Rightarrow$ surface normal perturbation (bump maps)
  - texture warped environment map $\Rightarrow$ reflection map
  - texture I $\Rightarrow$ phong shininess parameter
  - texture depth $\Rightarrow$ shadows (with projective map)
- Map
  - from texture to surface
  - must be invertible
  - is onto surface in world, model, or even projection coordinates
  - may be built in pieces
  - may be specified only at vertices and interpolated across polygons

Texture coordinate space

Texture-space is **always** [0,1] x [0,1]
Texture parameters are u and v
Lookup of a texture value (u,v) in texture-coordinate space:
- index texel at (round(uW), round(vH))

Map to surface in world/model/projection 3D space

Is a function from $f:(u,v)\rightarrow(x,y,z)$

Must be invertible (see below)
Are often two-stage maps (more on that later)

Texture lookup in scan conversion alg

When a pixel is determined to be visible:
- Scanline alg knows pixel-space (x,y,z)
- Map pixel-space (x,y,z) $\Rightarrow$ world/model (using $(VPS)^{-1}$)
- World/model $\Rightarrow$ texture-space (using f^{-1})
- Look up texture image value and use it to color pixel.

Texture coordinates and their interpolation

Inform graphics pipeline of f and f^{-1} at vertices only
- $T_i = (u_i, v_i) = f^{-1}(V_i)$ for all vertices V_i

Require scan conversion to interpolate (u,v) values between vertices.
Beware of perspective problems.

Perspective correction

Must interpolate in pre-perspective-division coordinates
For a point (x,y,z,w) with tex coordinate (u,v)
- map (u,v) to (u/w, v/w, 1/w) in post-perspective space

Interpolate all three coords normally to get some triple (u', v', w')
At texture lookup time
- map (u', v', w') to (u,v)=(u'/w', v'/w') get back into pre-perspective

Boundary considerations

Clamped:
- clamp to [0,1]x[0,1] to index map

Extra boundary pixels
- size is (n+2)x(m+2) with extra border rows/cols

Extra background color
- Accessed when tex coords are out of bounds

Wrap:
- ignore integer portion

Reflect:
- reverse fractional portion when integer portion is even

Reflect once (in DirectX)

Two-stages maps

Easily invertible map from (u,v) to an intermediate surface

Possible intermediate surfaces: sphere, cylinder, plane, cube

Map the intermediate surface to the real surface

Hint: we only need the inverse of this map!

Possible maps (inverted): centroid, object normal, reflected ray

Example: Spherical intermediate surface and centroid projection:

Part 1:

$$S(u,v) = (\sin \pi v \ \cos 2\pi u, \ \sin \pi v \ \sin 2\pi u, \ \cos \pi v)$$

To invert, let

$$(x,y,z) = (\sin \pi v \ \cos 2\pi u, \ \sin \pi v \ \sin 2\pi u, \ \cos \pi v)$$

and solve for u and v:

$$z = \cos \pi v \quad \Rightarrow \quad v = \frac{\cos^{-1} z}{\pi}$$

In C/C++ use **acos** which returns $0 ... \pi$

We can ignore height (z-axis) to get

$$(x,y) = (\cos 2\pi u, \ \sin 2\pi u)$$

so

$$\frac{y}{x} = \frac{\sin \pi v \ \sin 2\pi u}{\sin \pi v \ \cos 2\pi u} = \frac{\sin 2\pi u}{\cos 2\pi u} = \tan 2\pi u$$

and

$$u = \frac{\tan^{-1}(y/x)}{2\pi}$$

In C/C++ use **atan2(y,x)** which returns $0 ... 2\pi$,
and does not lose the signs in y/x.

Part 2:

Given a point $P = (x,y,z)$

on the object,

$\frac{P}{|P|}$ is on the spherical intermediate surface.

Aliasing/Sampling problem

To avoid sampling problem we must:

- consider pixels as regions
- map pixel region back to texture space
  - (will be an arbitrary quadrilateral)
- Lookup and compute average of all texture pixels in quadrilateral
  - Weighted by quadrilateral's area of intersection with the pixel.
- Yuck! That's too hard

Simpler:

- Approximate quadrilateral with a pixel aligned rectangle.
- Average is now a simple average.

Aliasing: Summed Area Table

Given a texture image T , create another S of the same size:

Each pixel in S is computed by summing over a rectangle of T,

$$S(p,q) = \sum_{i=0}^{p} \sum_{j=0}^{q} T(i,j)$$

Then the sum of T over some rectangle $[u_{0,} u_1] \times [v_{0,} v_1]$

can be computes with four lookups in S:

$$S(u_{1,} v_1) - S(u_0-1, v_1) - S(u_{1,} v_0-1) + S(u_0-1, v_0-1)$$

Aliasing: MIP maps (Much in Little; Latin: *Multo Im Parvo*)

Preprocess texture map to MIP map structure

- Filter original image to half size, quarter size, ...
  - where each pixel is an average of 2x2 block of pixels
  - (or use a better filter than the box filter)
- After filtering, each pixel in level d accounts for 2^{2d} pixels in the original
  - Level 0: 1x1 = 1
  - Level 1: 2x2 = 4
  - Level 2: 4x4 = 16
  - ...
  - Level d: $2^d \mathrm{x} 2^d = 2^{2d}$

Indexing the mipmap: given a pixel:

- Inverse map pixel to quad in texture space
  - Let T = center position of quad
    A = area of quad,
  - and convert from $[0,1]\times[0,1]$ space to $[0,W]\times[0,H]$ with $A \cdot W \cdot H$
- Choose level d where pixels account for AWH original pixels

$$2^{2d} = A \cdot W \cdot H \quad \text{so} \quad d = \frac{\log_2 A \cdot W \cdot H}{2}$$

- Access pixel T in level d , or
- If d is fractional, blend proportionately between two levels.

Size of MIP map is merely 4/3 of the original image:

Because $$1 + \frac{1}{4} + \frac{1}{16} + \frac{1}{64} + \ldots = \frac{4}{3}$$

Compute a quad from the scanline algorithm

The scanline algorithm does not keep enough info around to compute the quad.

But it can approximate its area:

Let the values in the scanline algorithm be named

T: The texture coordinate computed at each pixel.

E: The scanline-to-scanline change in T along an edge $\Delta T / \Delta y$

P: The pixel-to-pixel change in T along an scanline $\Delta T / \Delta x$

Because of perspective correction, these values are all triples:

$$T = (T_u, T_v, T_w)$$
$$E = (E_u, E_v, E_w)$$
$$P = (P_u, P_v, P_w)$$

After homogeneous division by T_w these become

$$\bar{E} = (E_u/T_w, E_v/T_w)$$
$$\bar{P} = (P_u/T_w, P_v/T_w)$$

The pixel preimage quad is approximated by the parallegram $\bar{E}\bar{P}$, whose area is

$$A = \begin{vmatrix} E_u/T_w & E_v/T_w \\ P_u/T_w & P_b/T_w \end{vmatrix}$$

Magnification

When a texture is viewed closely, **pixelation** occurs.

Given texture coordinate (u,v), and texture T, convert to texel index

$$(u_i, v_i) = (uW, vH)$$

Nearest neighbor:

$$T(\text{round}(u_i), \text{round}(v_i))$$

Bilinear interpolation:

Lower texel indices: $(u_0, v_0) = (\lfloor u_i \rfloor, \lfloor v_i \rfloor)$

Upper texel indices: $(u_1, v_1) = (u_0+1, v_0+1)$

Fractional texel indices: $(u_f, v_f) = (u_i - u_0, v_i - v_0)$

Return blended value:

$$(1-u_f)(1-v_f)T(u_0,v_0) + u_f(1-v_f)T(u_1,v_0) + (1-u_f)v_f T(u_0,v_1) + u_f v_f T(u_1,v_1)$$

Anisotropic problem

When the back transformed pixel results in a no-where near square quad:

both S.A.T. And MipMap cause too much blurring.

So split long quad into multiple square-ish regions, lookup each, and average.