Environment/Reflection maps

Environment/reflection maps

- Load image(s) that represent the environment
- For each pixel showing a reflective object:
  - Compute object's normal
  - Compute reflection direction
  - Compute index(s) into environment map
  - Use texel to color pixel
- An environment reflection needs position and direction
  - Position from renderer
  - Direction from normal and view direction: $R = 2(N \cdot V)N - V$
- Simplify to use just direction
  - That's a reasonable approximation if object is
    - small, simple, and distant from other objects.

Polar Spherical texture environment map

- Texture: One full texture with polar distortion
- Map: Polar map inverse:

$$(u, v) = \left(\frac{\tan^{-1}(y/x)}{2\pi}, \frac{\cos^{-1} z}{\pi} \right)$$

- Pro:
  - Simple
  - Was first
- Con:
  - Distortion is very bad.
  - Linear interpolation of texture coords fails when:
    - cross boundary
    - contain poles

Cubic texture environment map

- Texture: 6 square texture maps for axis aligned directions.
- Map: 2 stage:
  - Stage 2: Cube to surface inverse: follow reflection direction
    - Reflection direction:
    - Project from center of cube to cube surface. by scaling R to

$$\frac{(r_x, r_y, r_z)}{max(|r_x|, |r_y|, |r_z|)}$$

  - Stage 1: Texture to cube inverse: Just six planar texture maps
    - Choose which of six from Stage 2 results
- Pro:
  - Little distortion
  - No pole/boundary interpolation problems
  - Can be generated programatically
  - Are view independent
- Con:
  - Needs six images

Spherical texture environment map

Texture: One full texture using center circle portion only

Map: For $R=(r_x, r_y, r_z)$

N is half-way between E=(0,0,-1) and R

$$(n_x, n_y, n_z) = \frac{(r_x, r_y, r_z - 1)}{\sqrt{r_x^2 + r_y^2 + (r_z - 1)^2}}$$

so a good choice of coordinates is

$$(u', v') = (n_x, n_y) = \frac{(r_x, r_y)}{\sqrt{r_x^2 + r_y^2 + (r_z - 1)^2}}$$

which maps all (unit) directions into a unit circle, and

$$(u, v) = \left(\frac{u'}{2} + \frac{1}{2},\ \frac{v'}{2} + \frac{1}{2}\right)$$

which maps $[-1,1] \times [-1,1] \rightarrow [0,1] \times [0,1]$

Pro:

Can be generated with a camera
or programatically followed by distortion calc
No pole/boundary interpolation problems
Needs one image.

Con:

Is view dependent
Linear interpolation is only an approximation.

Both Pro/Con

Low resolution on sphere edges.

Short cut:

Since scanline produces N, don't compute R then N then (u,v),
instead just $(u', v') = (n_x, n_y)$

OpenGL note

In class, the eye coordinate system looks along the +z axis, in OpenGL, the eye looks along the -z axis, so the above equation has $(r_z+1)^2$ instead of $(r_z-1)^2$ in the OpenGL manuals.

Paraboloid Mapping

Like sphere, but using two hemispheres to retain resolution for view independence.

$$(u', v') = \frac{(r_x, r_y)}{r_z \pm 1}$$

Features:

Needs two image.

Pro:

No pole/boundary interpolation problems
Is view independent.

Con:

Generate only by distorting calculation

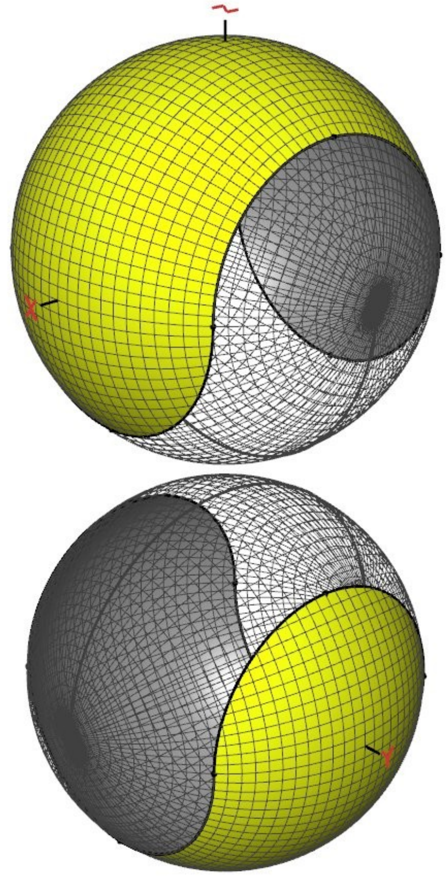
Tennisball map

Two maps of sphere, each with its bad areas (poles and seams) covered by the others nice areas (equator, non-seam). For a vector

$R = (a, b, c)$

$$C_1: \ (\theta, \varphi) = (\mathrm{asin}(b), \ \mathrm{atan2}(a, \quad c))$$

$$C_2: \ (\theta, \varphi) = (\mathrm{asin}(a), \ \mathrm{atan2}(b, -c))$$



For a given vector $R = (a, b, c)$, compute which component via

$$\mathrm{Component}(R) = \begin{cases} C_1: & c \geq 0 \ \text{and} \ |b| \leq \sqrt{2}/2 \\ C_1: & c \leq 0 \ \text{and} \ |a| \geq \sqrt{2}/2 \\ C_2: & \text{otherwise} \end{cases}$$

to get the parameter ranges

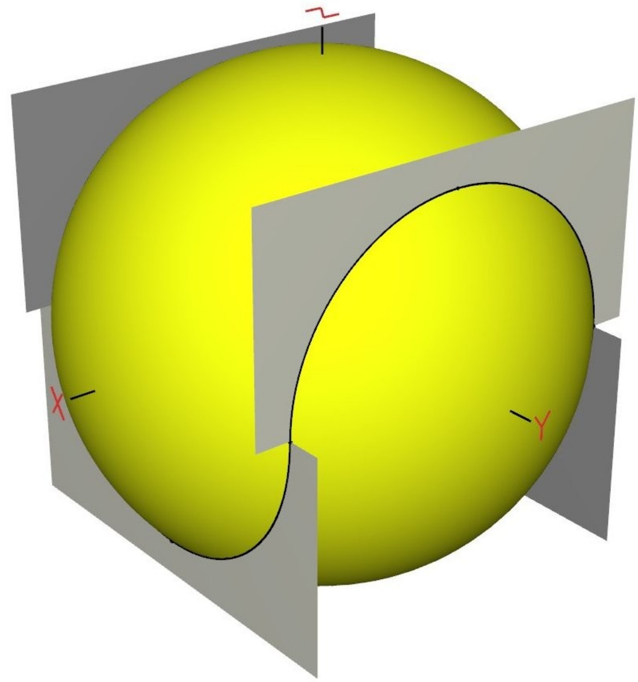
$$-\tfrac{\pi}{4} \leq \theta \leq \tfrac{\pi}{4}$$

$$-\tfrac{3\pi}{4} \leq \varphi \leq \tfrac{3\pi}{4}$$

For rendering into the map:

$$C_1: \ (\theta, \varphi, r) = (\mathrm{asin}(b), \ \mathrm{atan2}(a, \quad c), \ |R|)$$

$$C_2: \ (\theta, \varphi, r) = (\mathrm{asin}(a), \ \mathrm{atan2}(b, -c), \ |R|)$$


Z
X
Y