DigiPen
INSTITUTE OF TECHNOLOGY

CS300
Environment Mapping using Reflection Vector

M. C. Escher – Self Portrait

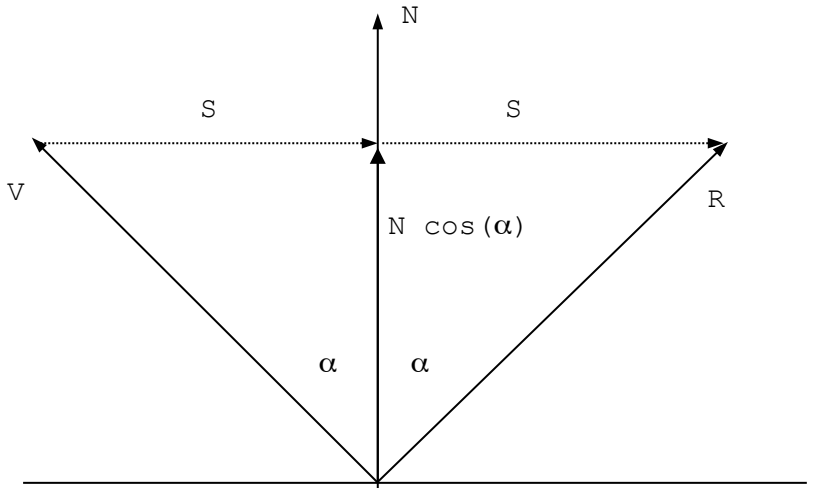
Environment Map / Reflection

- The goal is to render object as if it is perfectly reflective, so that the colors on the object's surface are those reflected to the eye from its surrounding.

- Reflection depends on:
  - Eye position
  - Surface position
  - Surface normal

Reflection Vector

- Where:
  - V is the vector toward the eye
  - N is the plane normal
  - R is the reflection vector



Reflection Vector *(cont'd)*

- R is the reflection vector of V and can be computed as follows:

$$R = N\cos(\alpha) + S$$

$$S = N\cos(\alpha) - V$$

- Substituting S, we have:

$$R = 2N\cos(\alpha) - V$$

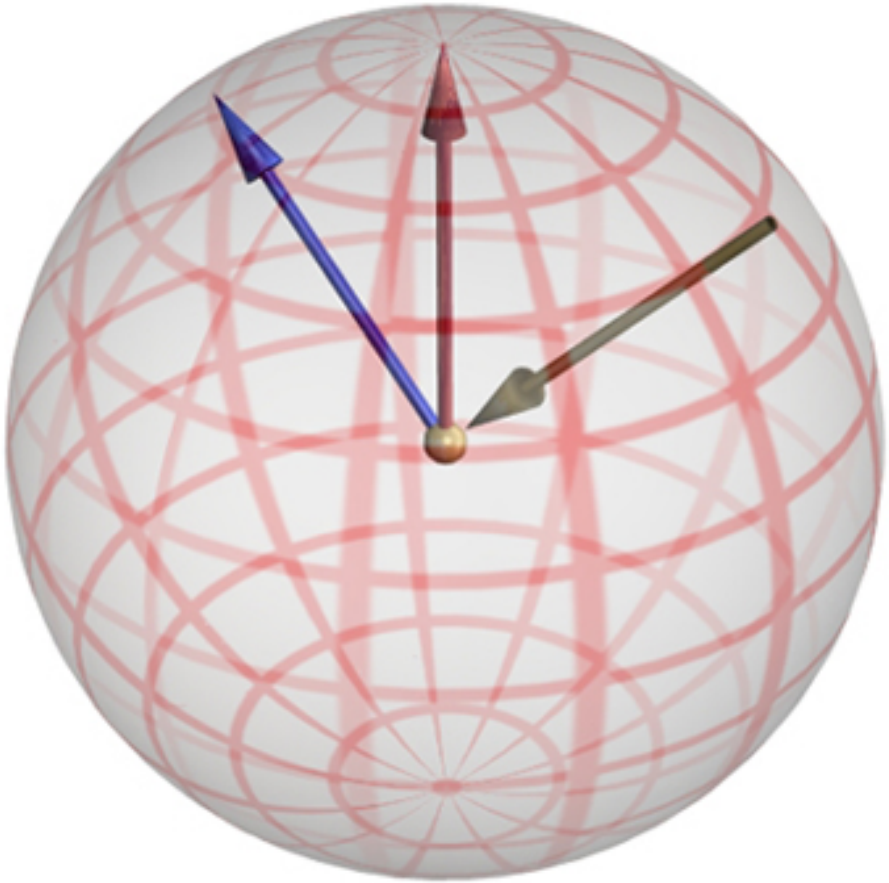
$$R = 2N(N.V) - V$$

Environment Map

- A map used to approximate the environment relative to a reflector.

- The environment map construction assumes that the reflective object is ‘small’ relative to its environment, i.e. the viewer and other objects being reflected are infinitely far away.

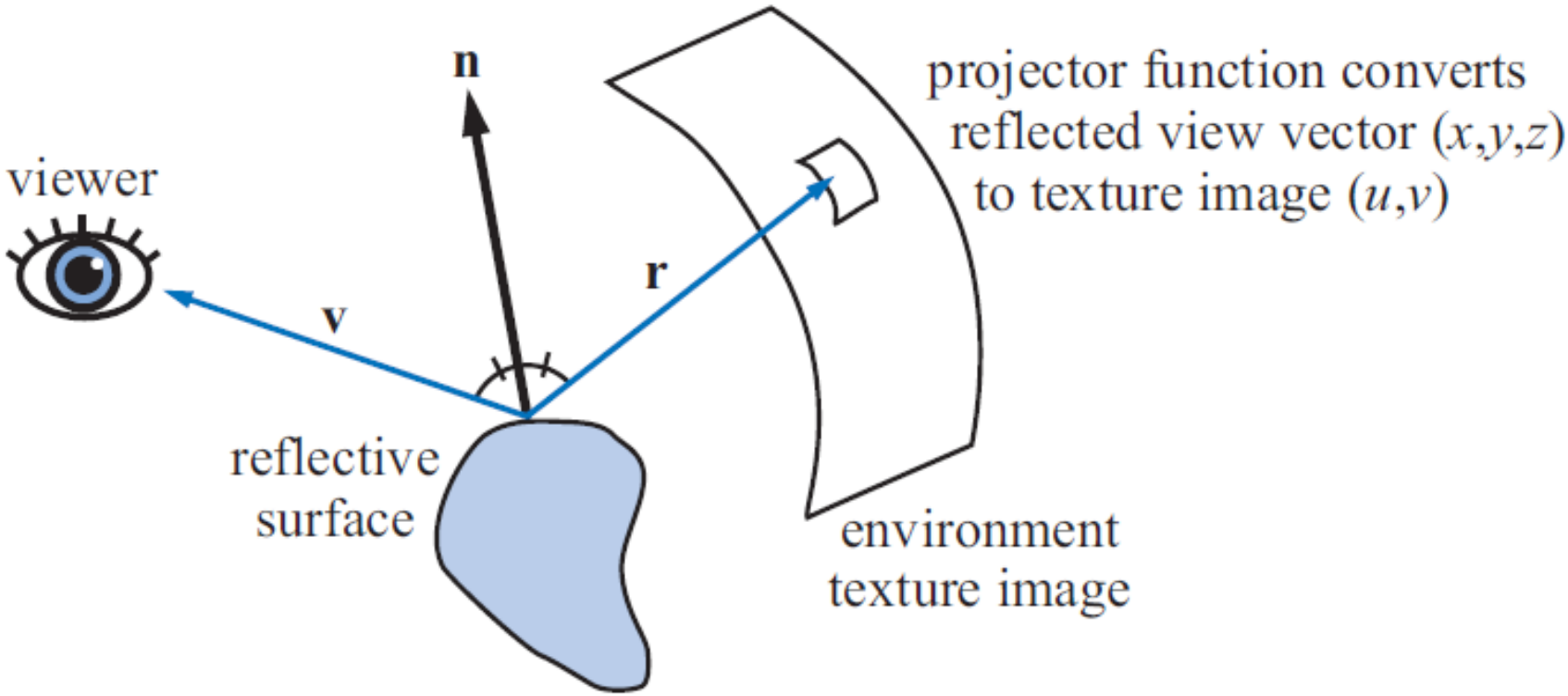
Environment and Object



Environment Map *(cont'd)*

- General process in environment mapping:
  - Load (or generate) the image(s) that represent the environment.
  - For each fragment on the reflective object:
    - Calculate the viewing vector
    - Calculate the normal vector
    - Calculate the reflection vector
    - Transform the reflection vector to texture coordinate
    - Use the texture coordinate to get the texel in the environment map

Environment Mapping



Environment Map *(cont'd)*

- An environment reflection needs position and direction
  - Position from renderer
  - Direction from normal and view direction

- References
  - Real-time Rendering, 3rd edition
  - http://www.opengl.org/resources/code/samples/sig99/advanced99/notes/node174.html

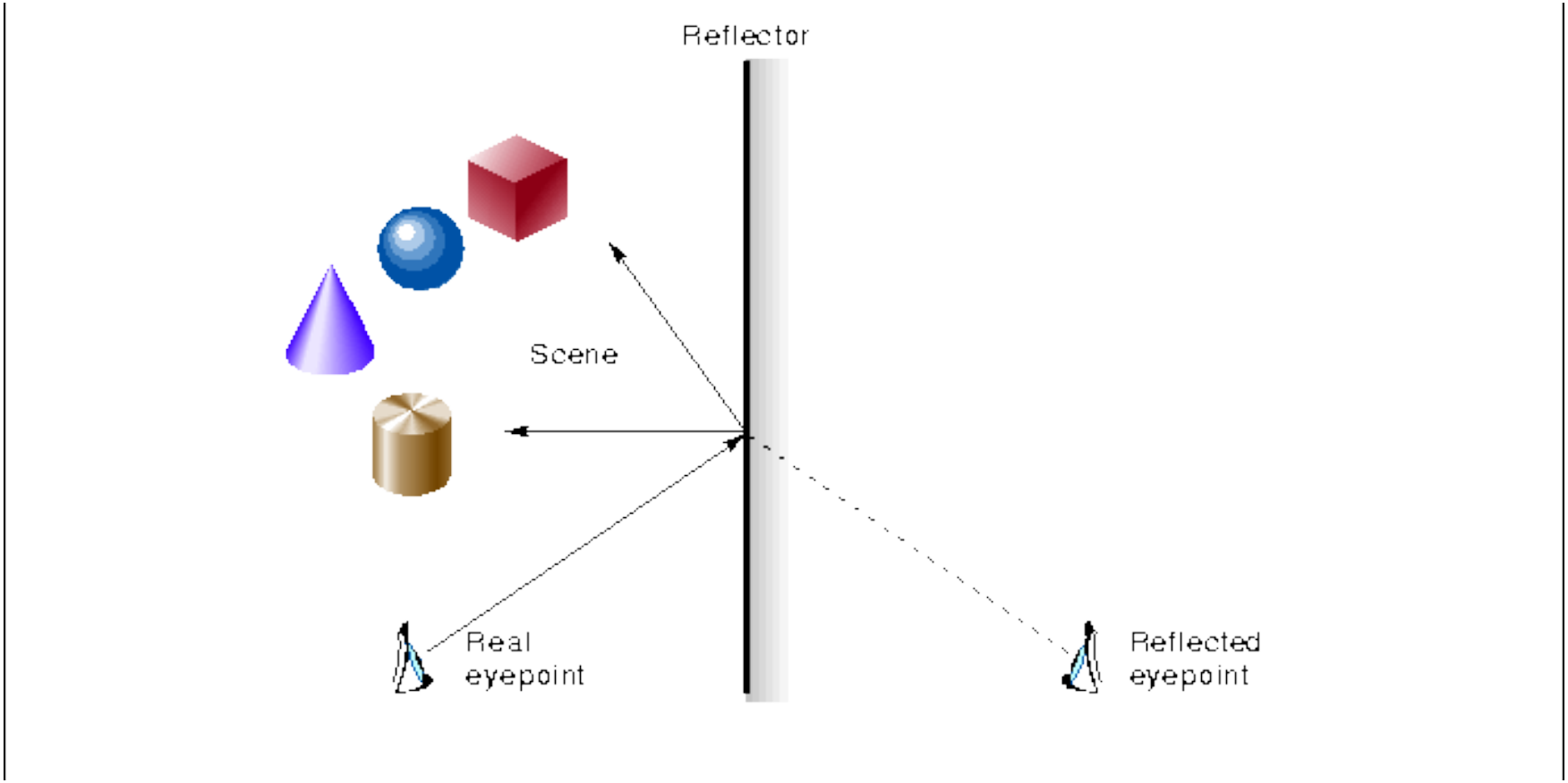
Environment Map *(cont'd)*

- Common environment map types:
  - Planar map
  - Polar spherical map
  - Spherical map
  - Cube maps
  - Dual parabolic maps

Planar Environment Map

- Used mainly for a planar reflector (mirror, calm water, etc).
- Process to generate the texture map:
  - Mirror the camera position and orientation (direction and up vector) along the reflector plane.
  - Flip the triangle winding order
    - Objects farther away from the camera should be visible
  - Render the scene from the new camera position to generate the environment map

Illustration



Planar Environment Map *(cont'd)*

- To calculate the texture coordinate:

$$s = (V_x / -V_z) * 0.5 + 0.5$$
$$t = (V_y / -V_z) * 0.5 + 0.5$$

- Where:
  - V is a vector from the camera to the vertex (or fragment) in camera space.

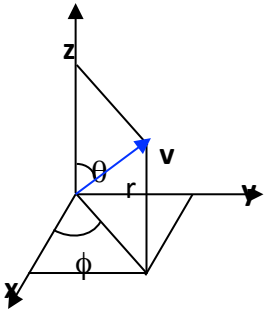
Planar Environment Map *(cont'd)*

- Pro:
  - Easy to generate
  - Only need one map
  - Very good approximation
- Con:
  - Very specific use
  - View dependent. Need to be regenerated for different viewer position or orientation.

Polar Spherical Environment Map

- Environment map is a texture that is wrapped around the sphere. Basically, each texel corresponds to a direction in polar coordinate.
- To generate the map, for each (u,v) in the texture coordinate, throw a ray along (ϕ, θ) and calculate its intersection with the environment.

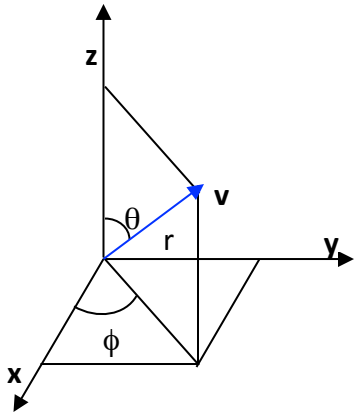
$$\phi = u * 2\pi$$
$$\theta = v * \pi$$



Polar Spherical Environment Map *(cont'd)*

- To calculate the texture coordinate from a reflection vector, use polar map inverse.

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



Polar Spherical Environment Map *(cont'd)*

- Pro:
  - Simple
  - First method for 3D coverage
  - View independent

- Con:
  - Distortion is very bad near the poles
  - Linear interpolation of texture coords fails when:
    - cross boundary
    - contain poles
  - Hard to generate in real-time

Spherical Environment Map

- The texture is a square texture where only the center circle is used.
- Few ways to generate the map:
  - Take a picture of a reflective sphere using a camera with infinite focal point.
  - A similar method used to generate the polar spherical environment map can be used.
  - Generated from cube maps by rendering the cube maps onto a distortion grid.

Light-probe

Spherical Environment Map (NVIDIA)

top

bottom

HDR Light Probe (pauldebevec.com)

Grace Cathedral, San Francisco
1000 × 1000
Dynamic range: 200,000:1

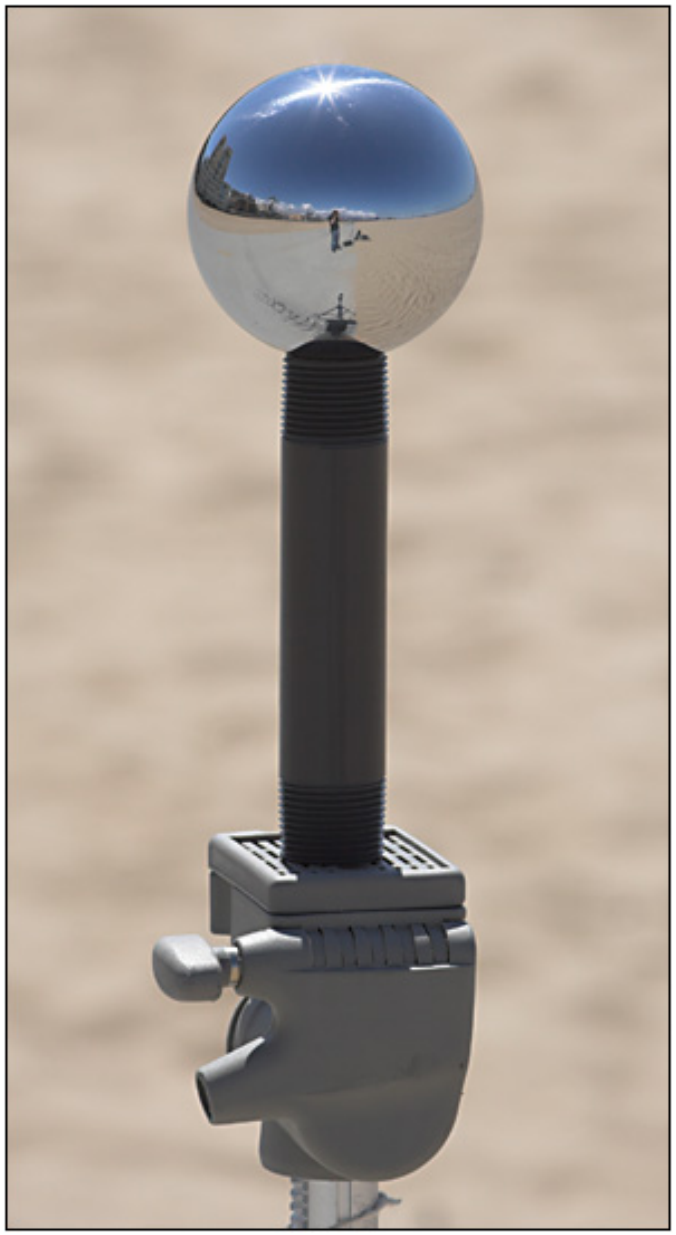
Angular map: .hdr I .float.gz I .pfm

Used as the illumination environment for Figure 6 of the SIGGRAPH 98 paper. Assembled from two radiance images of a mirrored sphere taken with a Sony VX1000 digital video camera (approx. ten image per sample.)

Eucalyptus Grove, UC Berkeley
900 × 900
Dynamic range: 5000:1

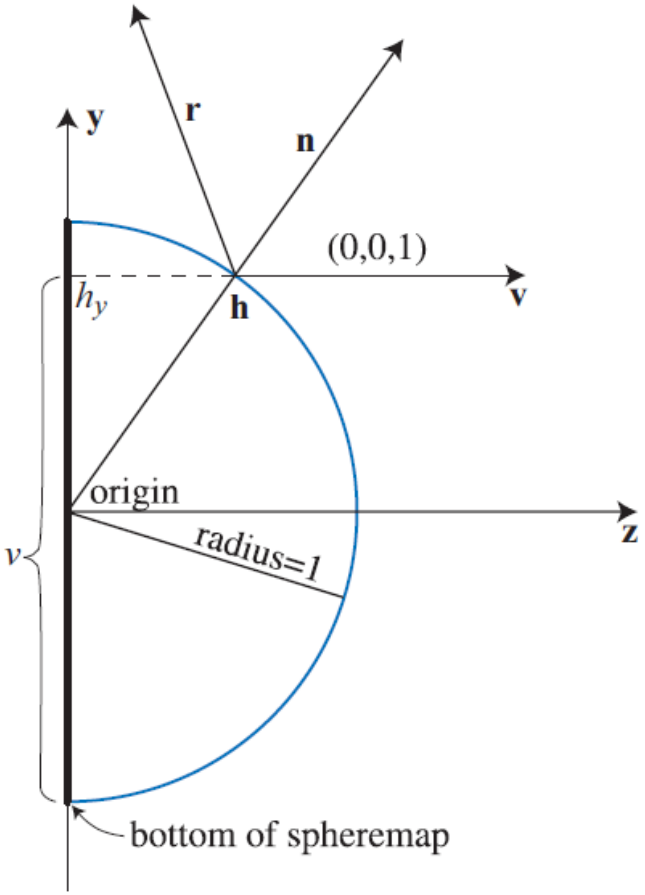
Angular map: .hdr I .float.gz I .pfm

Used as the illumination environment for Rendering with Natural Light; acquired similarly to the Grace Cathedral image..



Actual Light Probe

Calculating the Reflection Vector



Spherical Environment Map *(cont'd)*

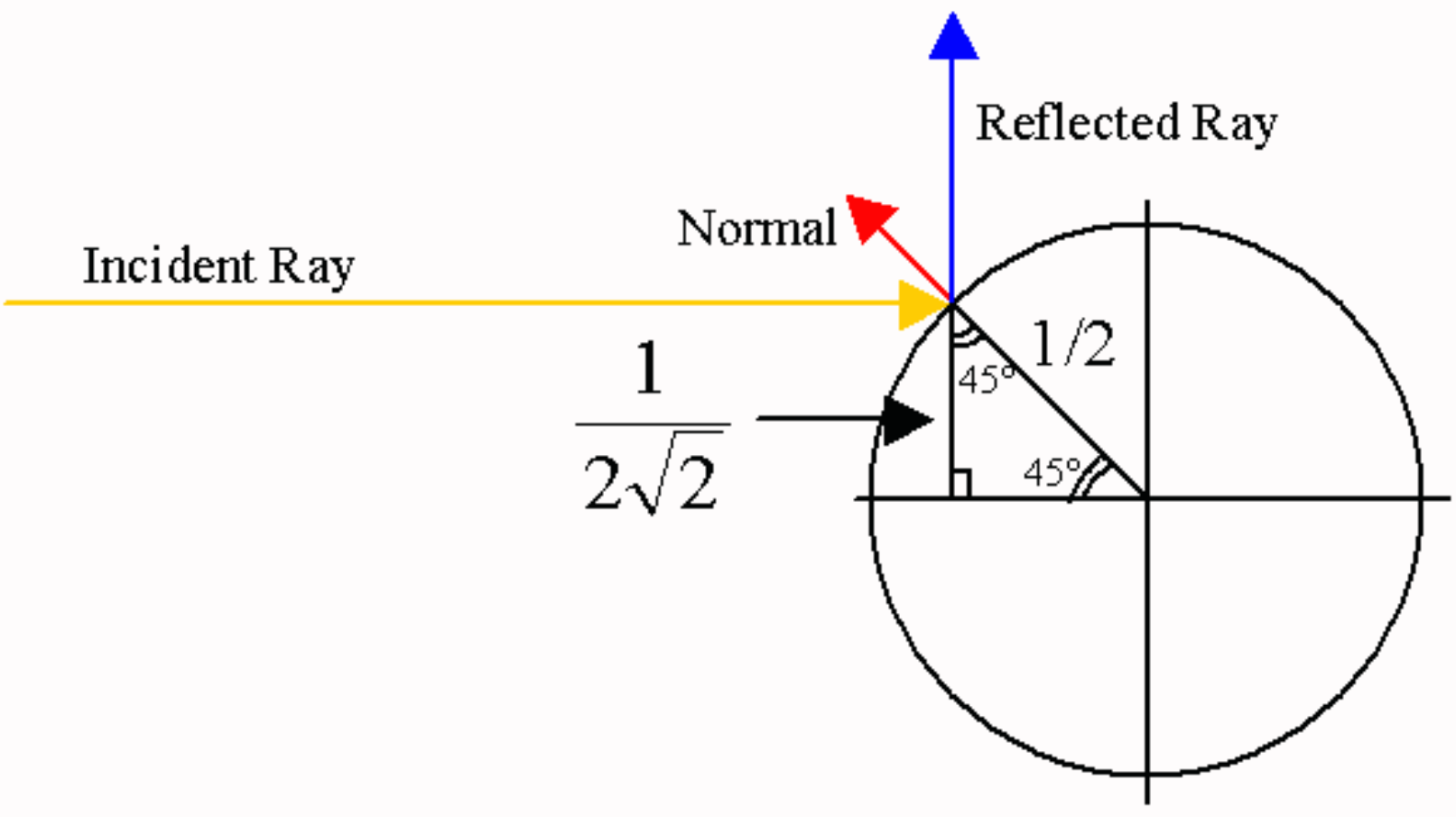
- To calculate the texture coordinate from a reflection vector (R_x, R_y, R_z):
  - View vector V = (0, 0, 1)
  - The normal is halfway between R and V

    `N = (R + V) / |R + V|`
  - The normal correspond to a position P on the sphere and P = N since the sphere's radius = 1.
  - So, can use (P_x, P_y) = (N_x, N_y) as (u', v').
  - The calculated (u', v') need to be mapped between [0, 1] to calculate the final (u, v).

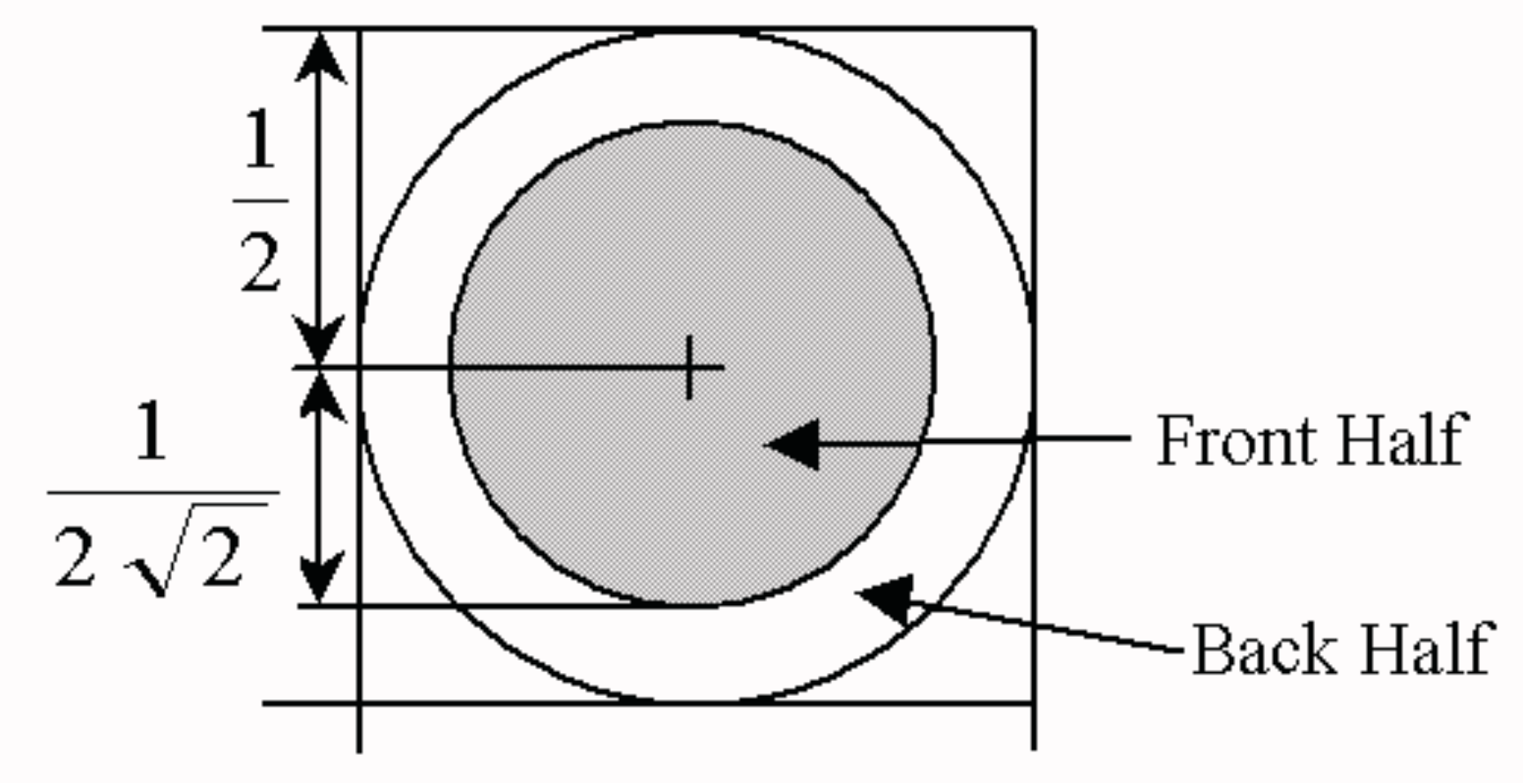
Spherical Environment Map *(cont'd)*

- Pro:
  - Can be generated with a camera
  - No pole/boundary interpolation problems
  - Needs one image
- Con:
  - View dependent
  - Linear interpolation is only an approximation
  - Hard to generate in real-time

Limitations of Spherical Mapping



Limited Resolution for "back" texels



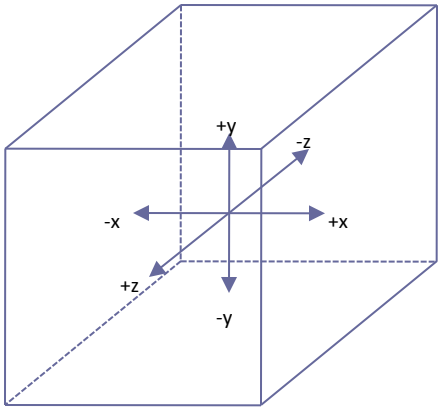
Spherical Environment Map *(cont'd)*

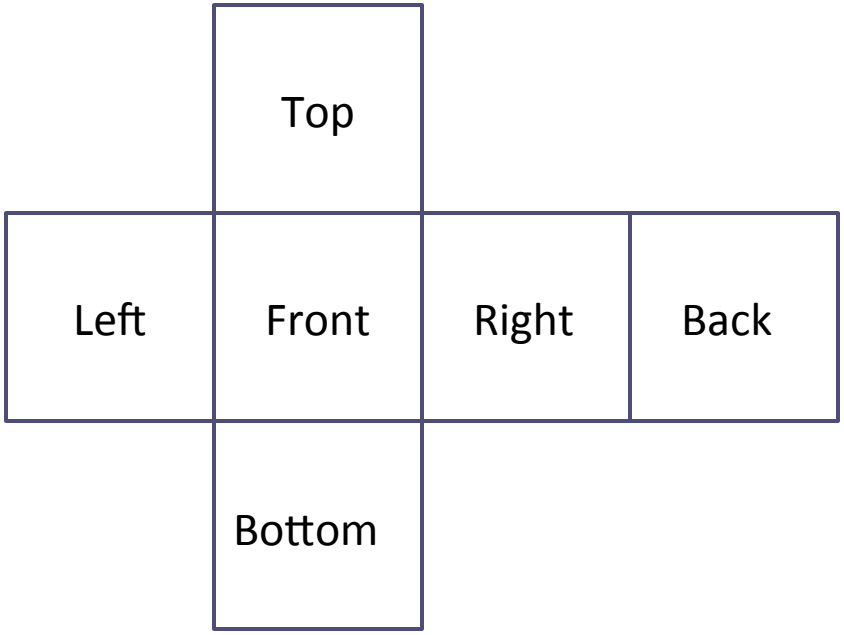
- Both Pro/Con
  - Low resolution on sphere edges.

- References
  - http://www.opengl.org/resources/code/samples/sig99/advanced99/notes/node176.html

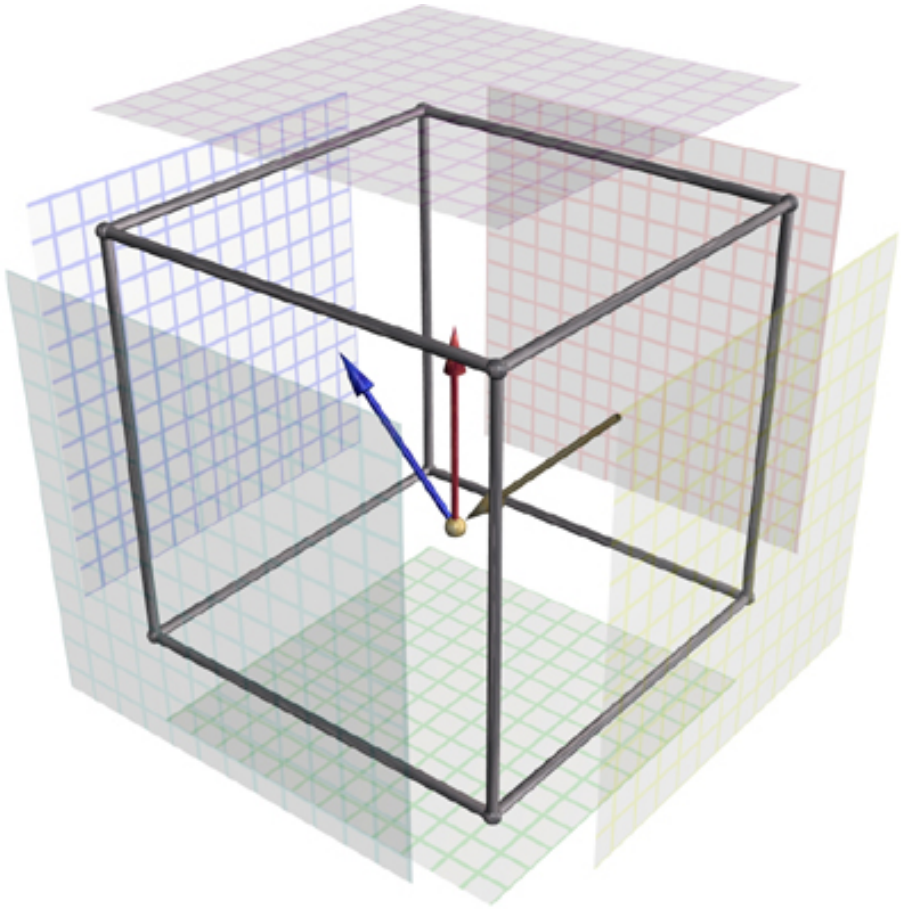
Cubic Environment Maps

- The environment maps are 6 square texture maps. One for each axis aligned directions.

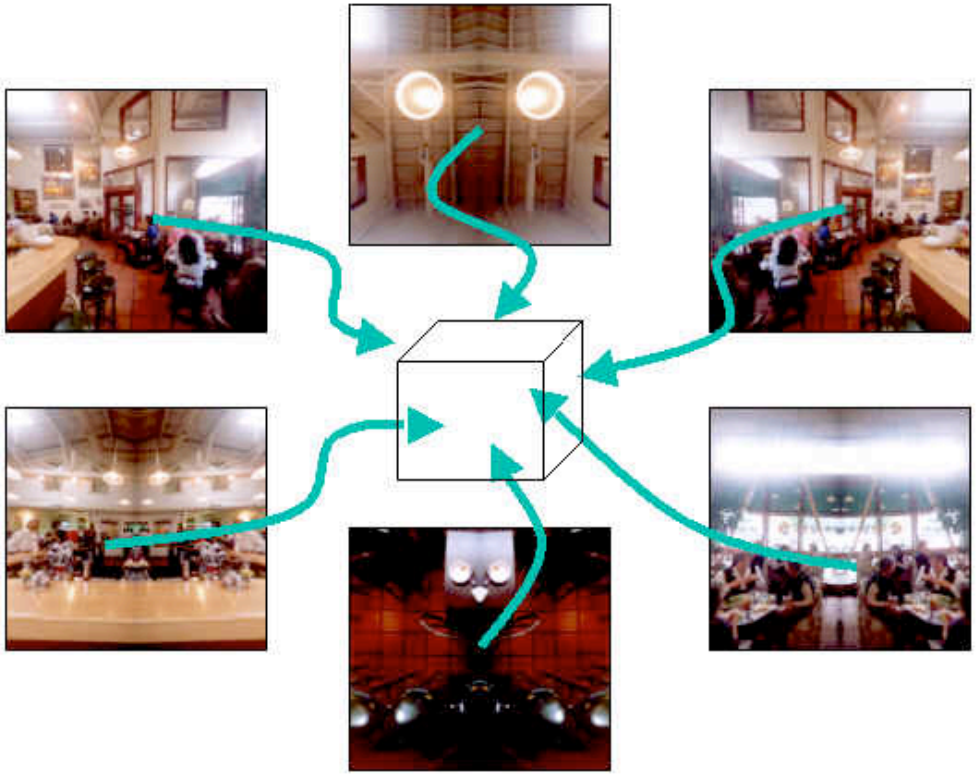




Environment and Object



Cubic Environment Maps



Spherical Map

Cubic Environment Maps *(cont'd)*

- Process to generate the maps:
  - Position the camera at the reflective object position.
  - Set the FOV to 90 degree and aspect ratio to 1.
  - For each map:
    - Orient the camera to point along the associated axis.
    - Render the scene

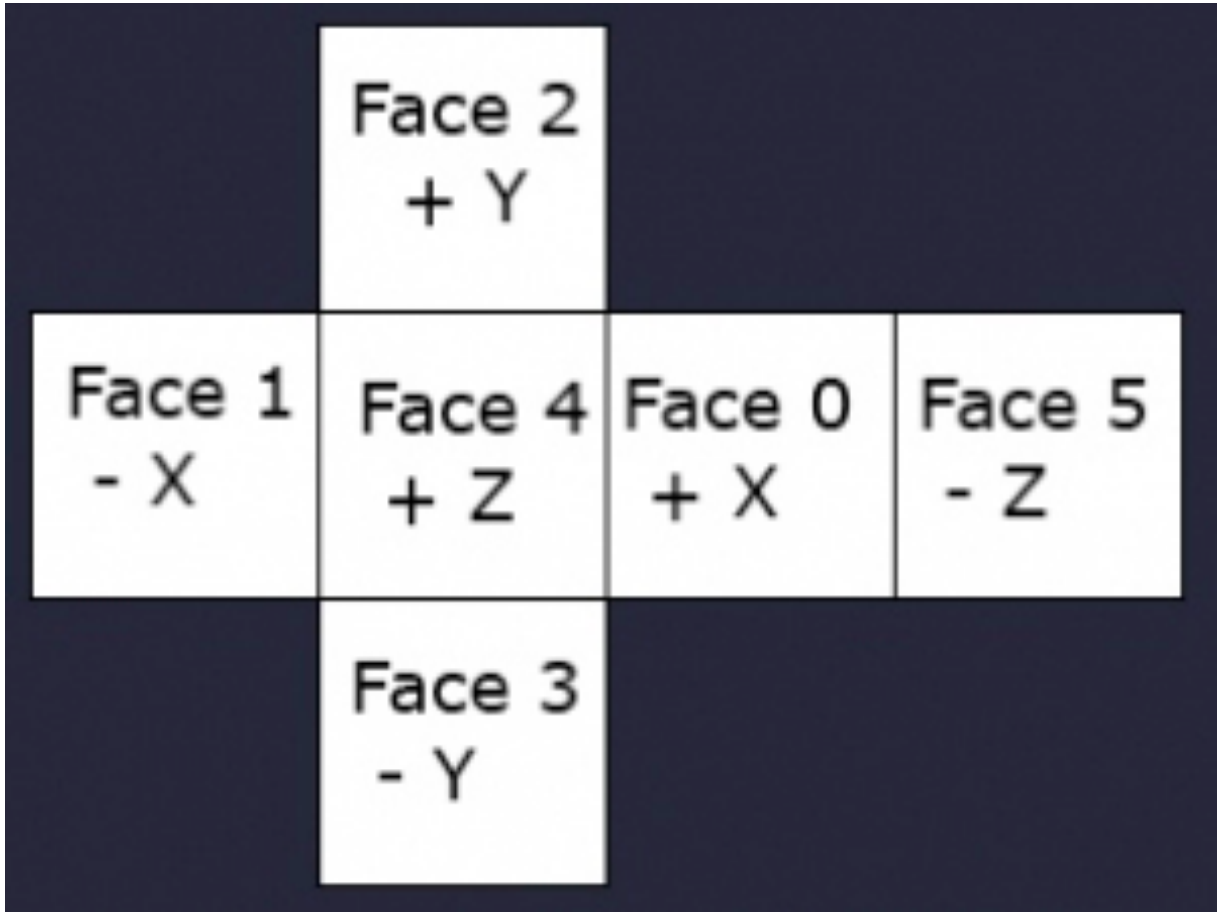
Cubic Environment Maps *(cont'd)*

- To calculate the texture coordinate from R:
  - Choose the texture map based on the greatest magnitude of R component.
  - Intersect R with the cube face

$$R' = R / \max(|Rx|, |Ry|, |Ry|)$$

  - Use the other 2 component that are not the greatest magnitude as the texture coordinate.

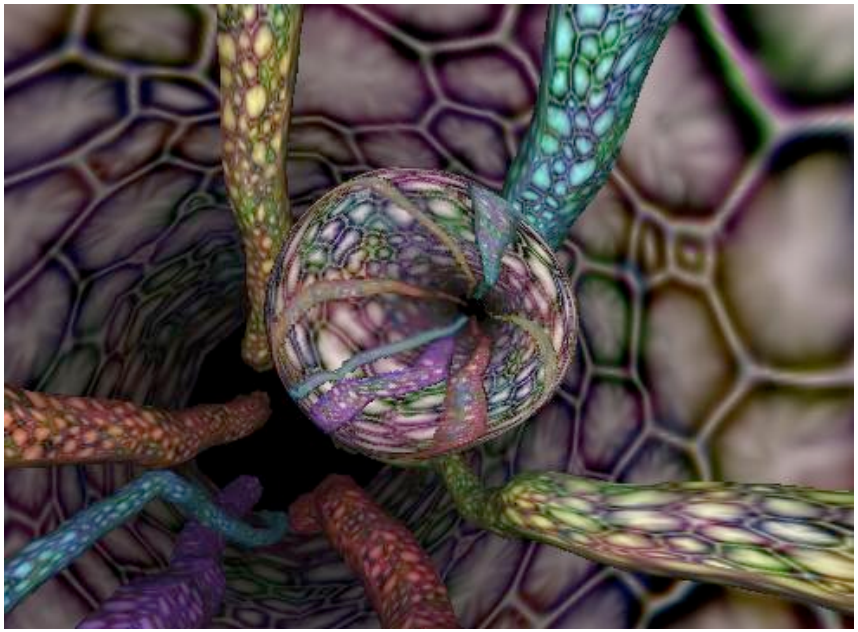
In pictures …



Cubic Environment Mapping

Cube Map

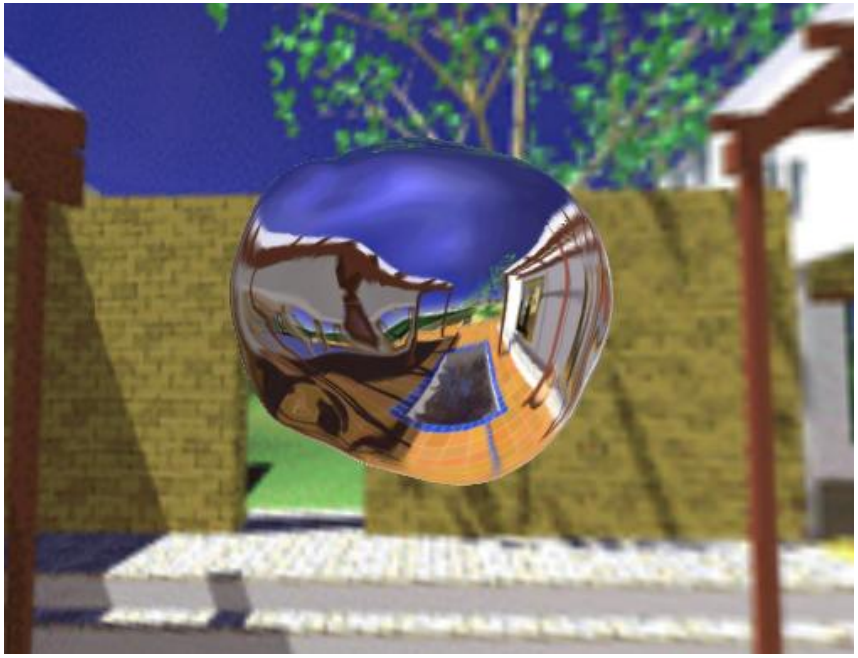
Result of rendering



Cubic Environment Mapping

Cube Map

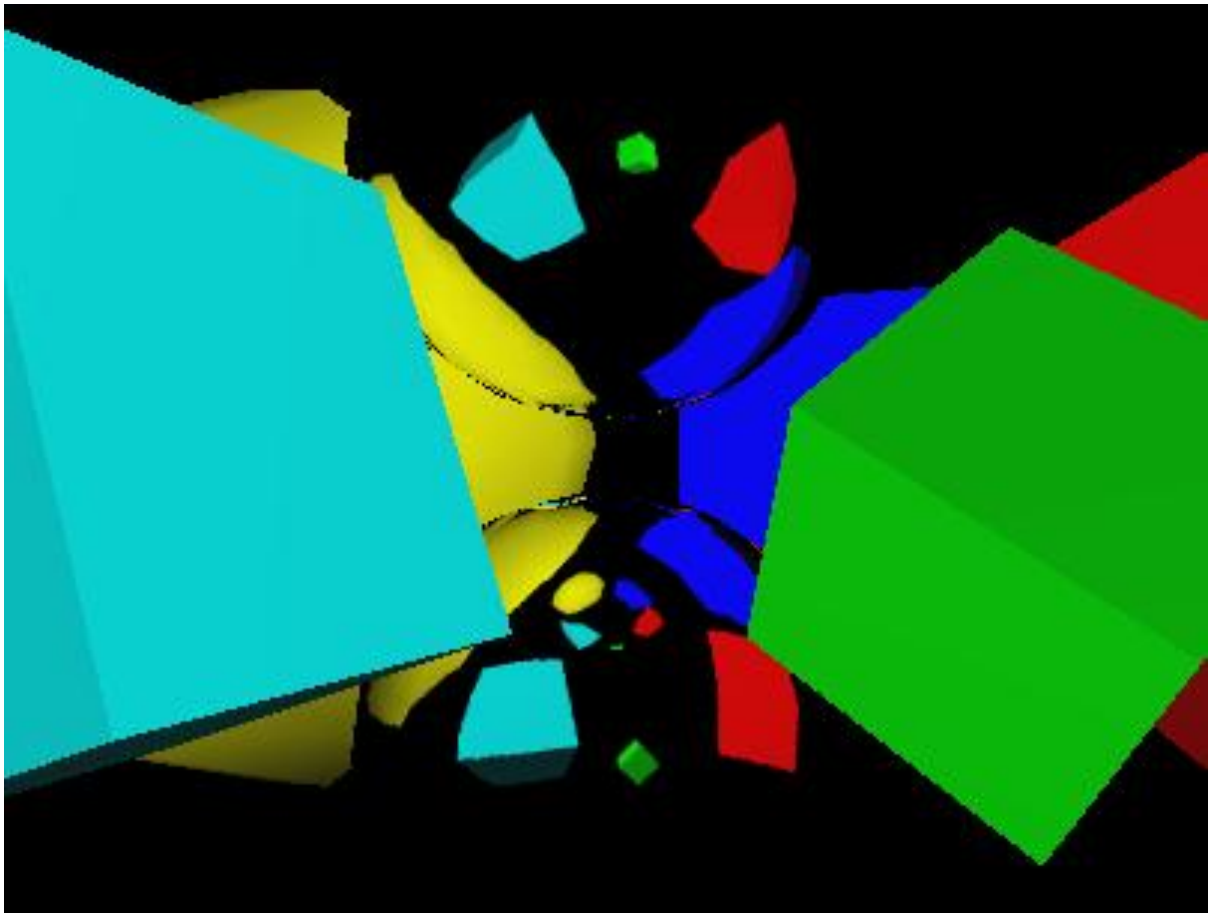
Result of rendering



Cubic Environment Maps *(cont'd)*

- Pro:
  - Little distortion
  - No pole/boundary interpolation problems
  - Can be generated programmatically
  - View independent
- Con:
  - Needs six images

Recursive Application!



18 passes!

Dual Parabolic Mapping

- Encodes the environment in two texture maps to retain resolution for view independence.
  - One for the front side
  - One for the back side

- Parabolic equations:
  - Front side:

$$f(x, y) = 0.5 - 0.5 * (x^2 + y^2)$$

  - Back side:

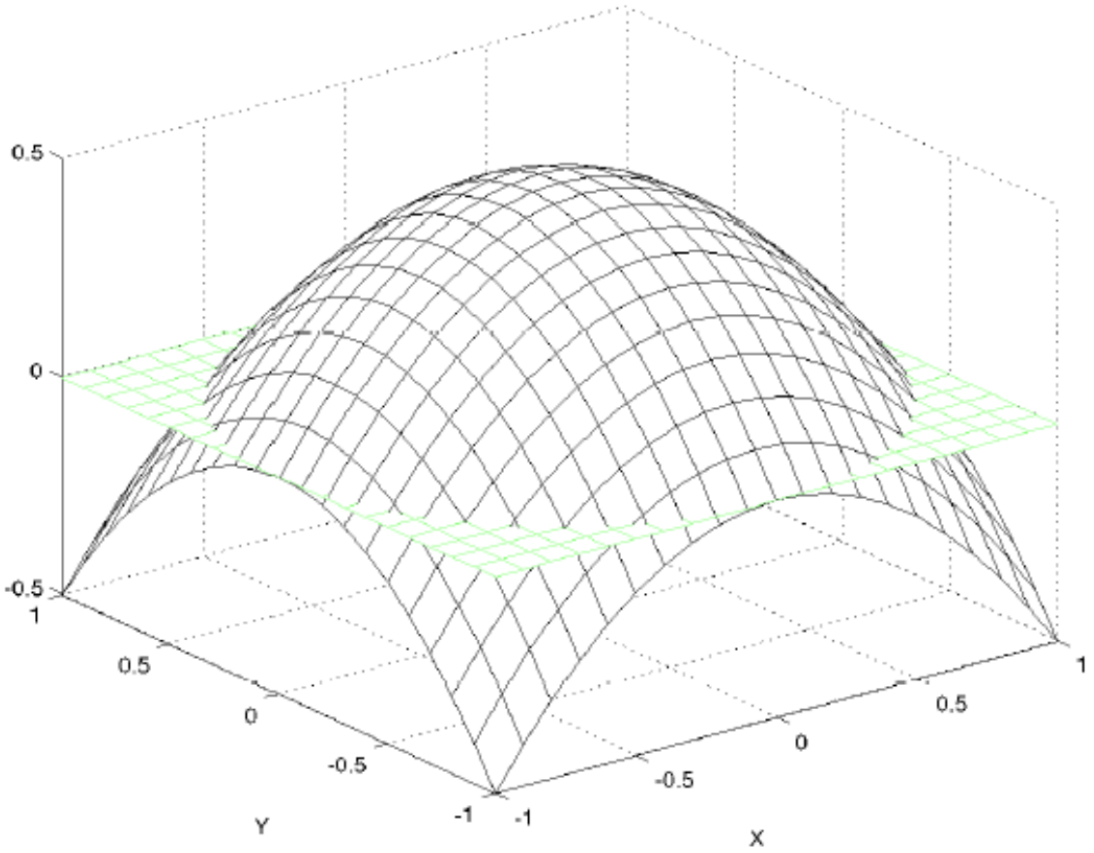
$$f(x, y) = -0.5 + 0.5 * (x^2 + y^2)$$

  - For both side:

$$x^2 + y^2 <= 1$$

$$f(x,y) = \frac{1}{2} - \frac{1}{2}(x^2 + y^2) \text{ for } x^2 + y^2 \leq 1$$

Parabolic Surface



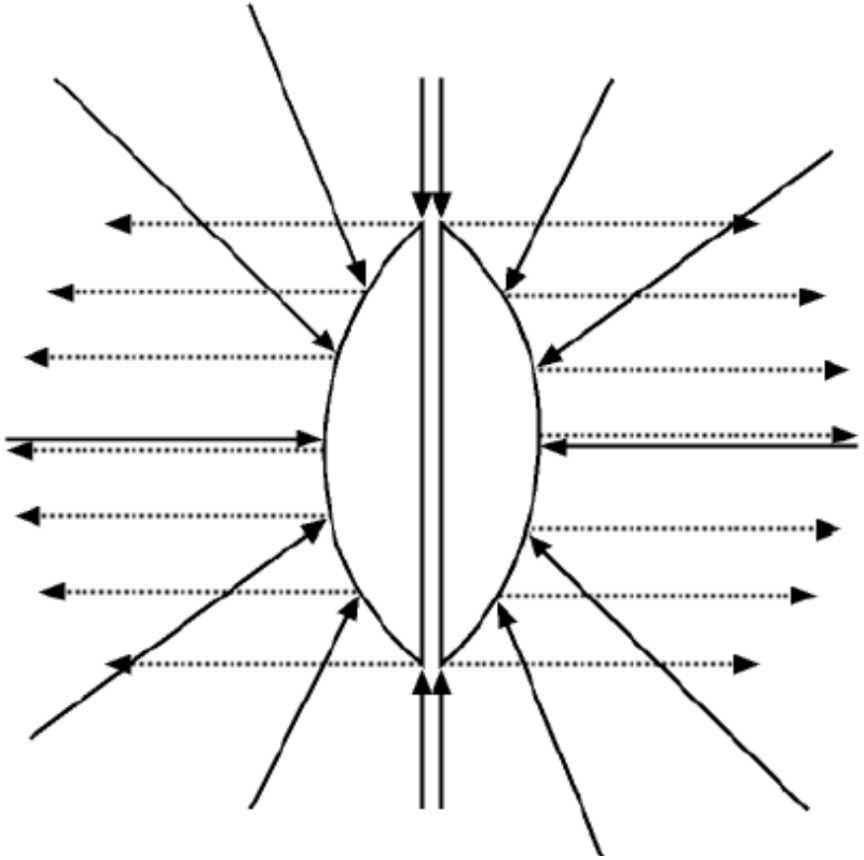
Deriving the Surface Normal

$$T_x = \frac{\partial P}{\partial x} = \left(1, 0, \frac{\partial f(x,y)}{\partial x}\right) = (1, 0, -x)$$

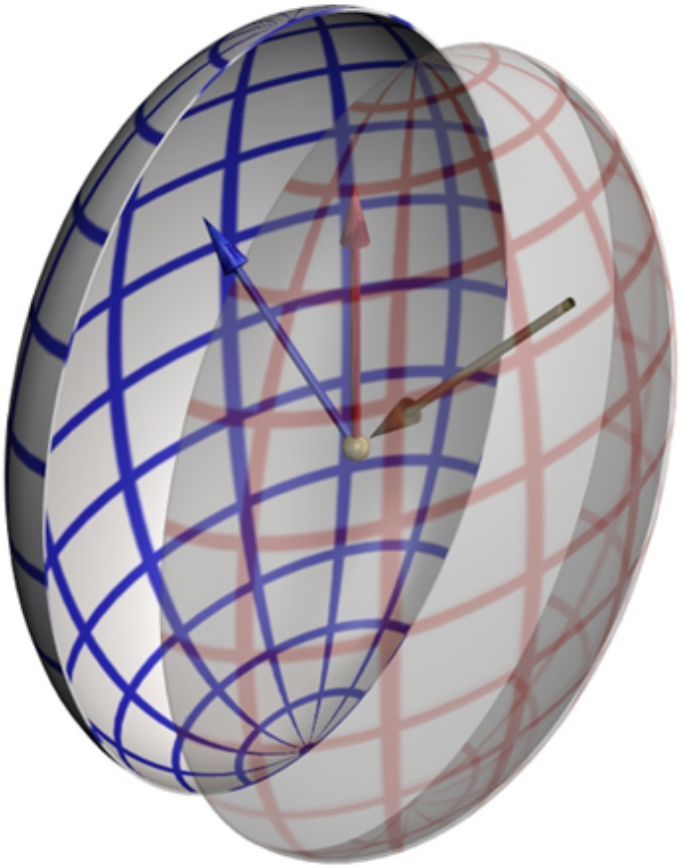
$$T_y = \frac{\partial P}{\partial y} = \left(0, 1, \frac{\partial f(x,y)}{\partial y}\right) = (0, 1, -y)$$

$$N_P = T_x \times T_y = (x, y, 1)$$

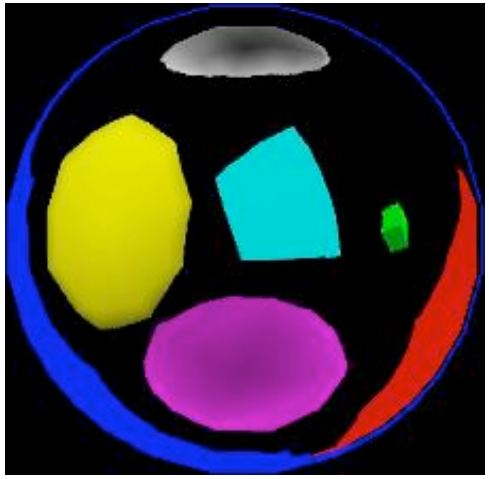
Dual Paraboloid Mapping



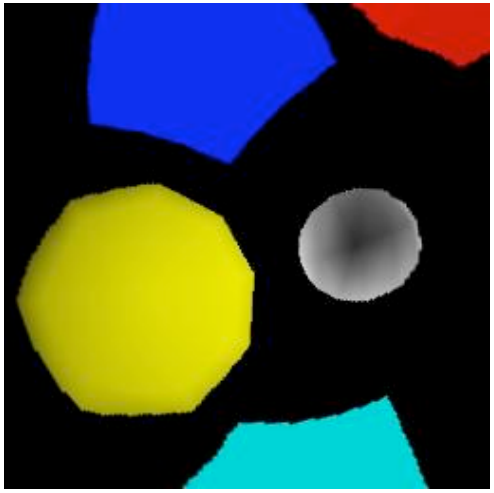
Environment and Object



Comparing Sphere Map and Parabolic Maps



Sphere map



front

back

Dual Parabolic Mapping *(cont'd)*

- Process to generate the parabolic map
  - Position the camera at the reflector position
  - For each side to be rendered:
    - Orient the camera
    - For each object in the scene
      - In vertex shader, distort the vertex position to its corresponding position on the texture map.
      - Render the object to the texture.

Dual Parabolic Mapping *(cont'd)*

- Calculating the texture coordinate from a reflection vector R:
  - Based on R_z choose the texture map
  - For the front one:
    - $u = (R_x / (R_z + 1)) * 0.5 + 0.5$
    - $v = (R_y / (R_z + 1)) * 0.5 + 0.5$
  - For the back one:
    - $u = (R_x / (1 - R_z)) * 0.5 + 0.5$
    - $v = (R_y / (1 - R_z)) * 0.5 + 0.5$

Dual Parabolic Mapping *(cont'd)*

- Pro:
  - No pole/boundary interpolation problems
  - View independent.
- Con:
  - Need two texture maps
- References
  - http://www.opengl.org/resources/code/samples/sig99/advanced99/notes/node184.html

Environment Mapping

- Reference
- http://developer.nvidia.com/cube-map-ogl-tutorial
- http://www.unc.edu/~zimmons/cs238/maps/environment.html