Environment Mapping

CS 418: Interactive Computer Graphics

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

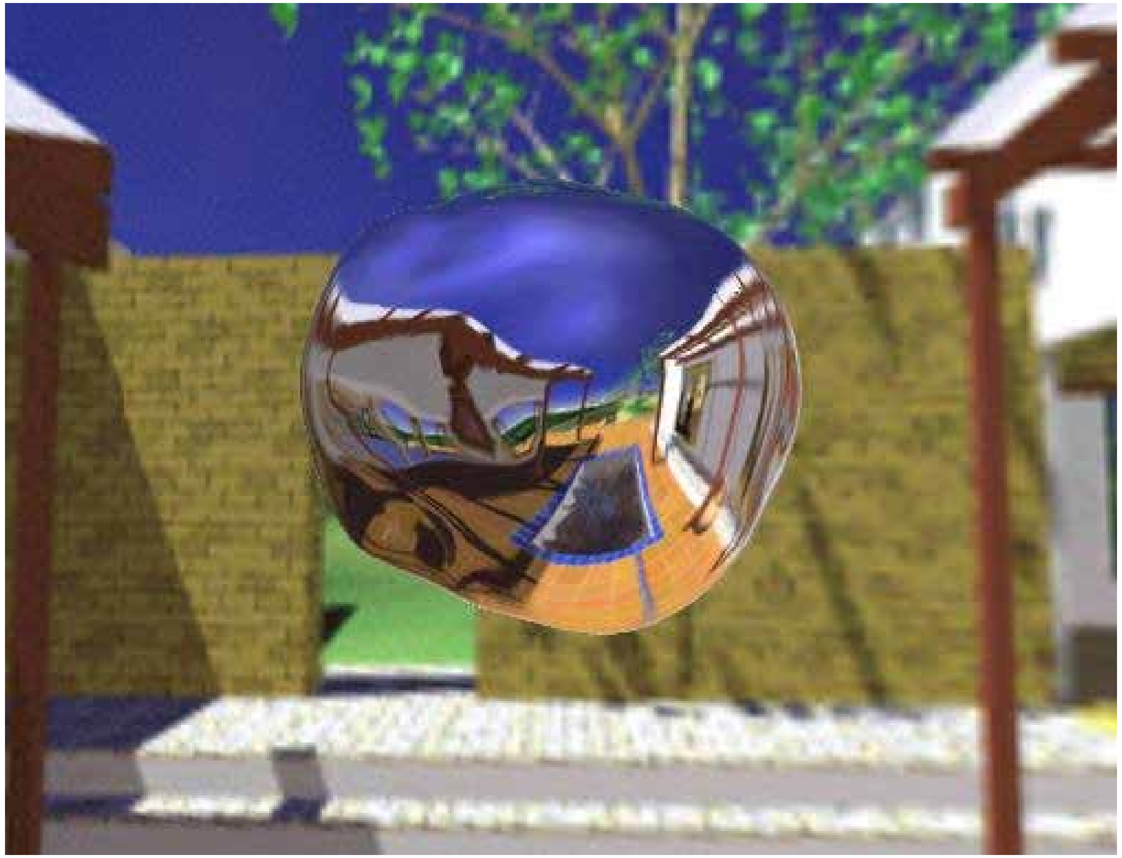
Eric Shaffer

Environment Mapping

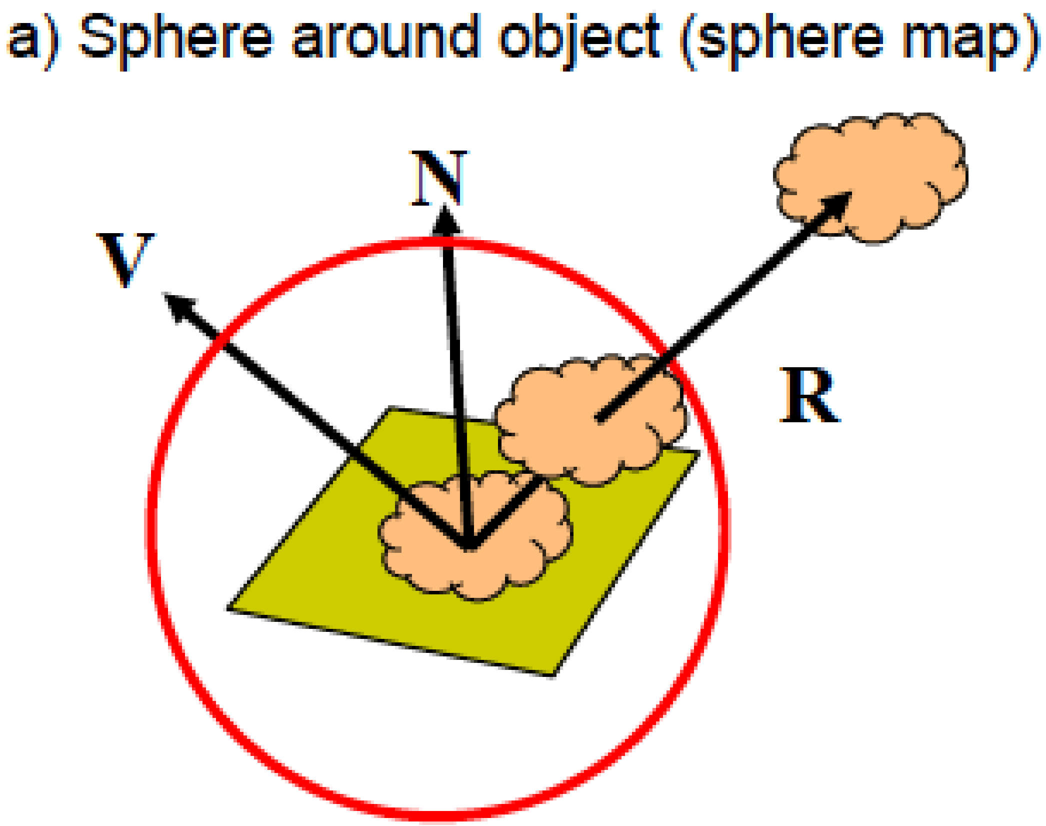
- How can we render reflections with a rasterization engine?
  - When shading a fragment, usually don't know other scene geometry
  - Answer: use texture mapping....

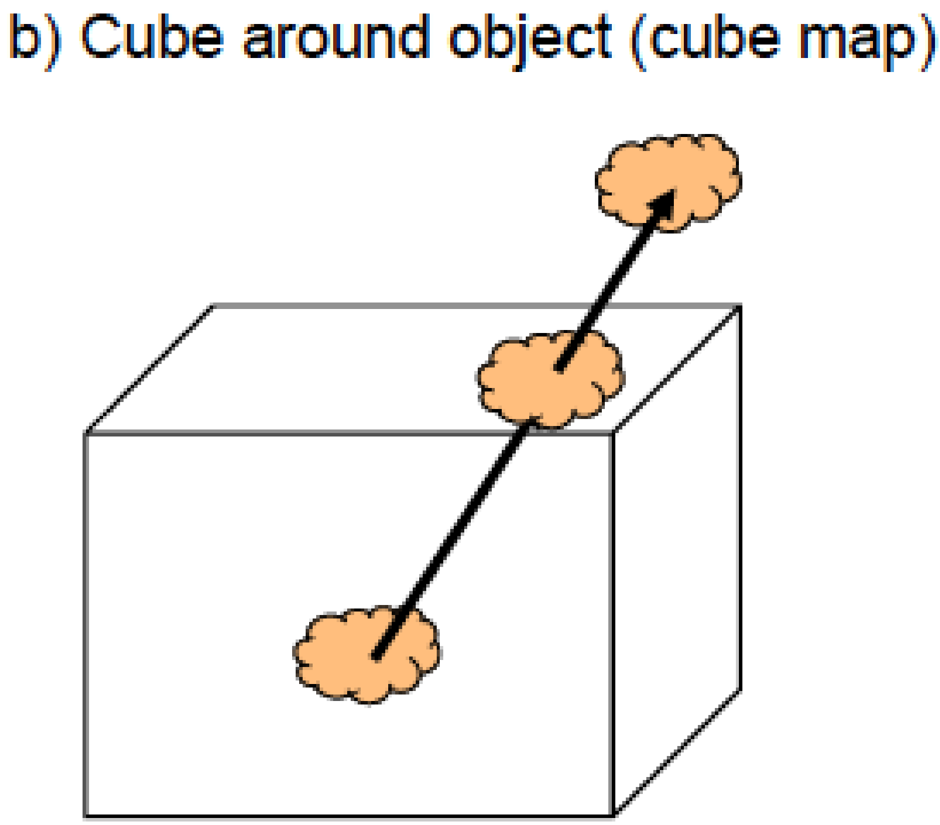
- Create a texture of the environment
  - Map it onto mirror object surface

- Any suggestions how generate (u,v)?



Types of Environment Maps





Sphere Mapping

- Classic technique…
- Not supported by WebGL

OpenGL supports sphere mapping

Requires a “circular” texture map

Equivalent to an image taken with a fisheye lens

Sphere Mapping Example

Sphere Mapping Limitations

- Visual artifacts are common
- Sphere mapping is view dependent
- Acquisition of images non-trivial
  - Need fisheye lens
    - Or render from fisheye lens
  - Cube maps are easier to acquire

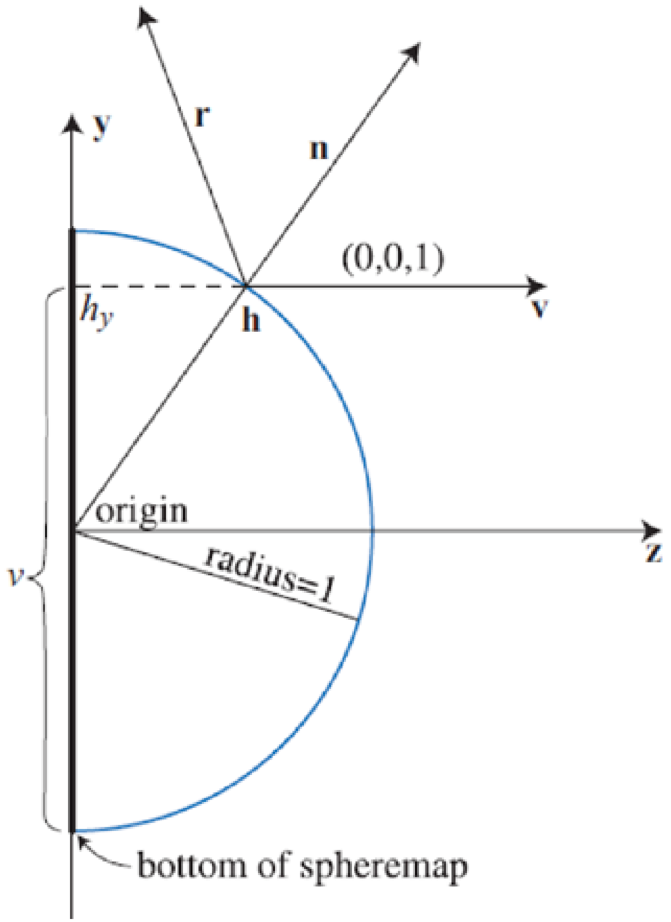
Acquiring a Sphere Map....

- Take a picture of a shiny sphere in a real environment

- Or render the environment into a texture (see next slide)

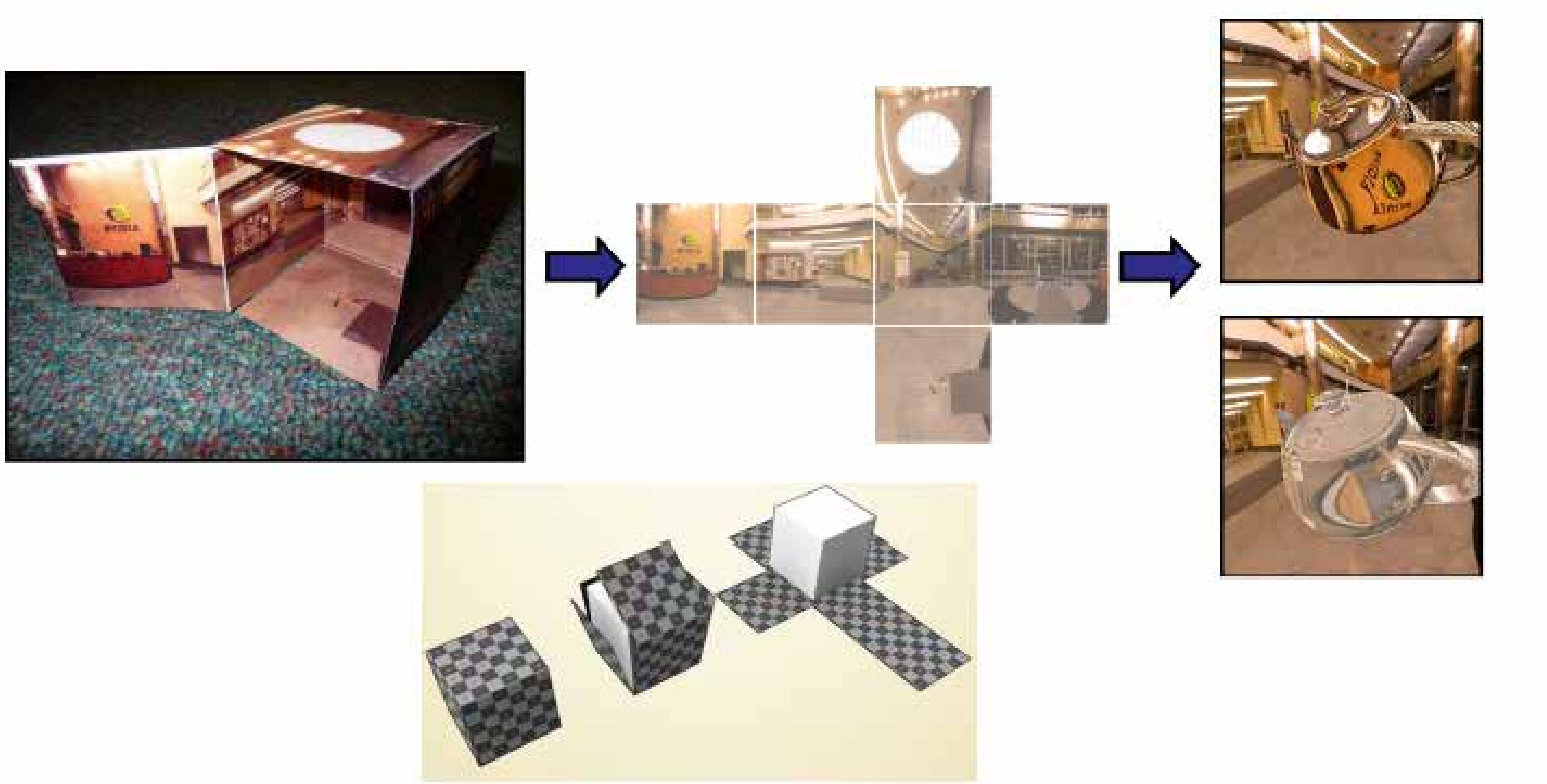
Why View Dependent?

- Conceptually a sphere map is generated like ray-tracing
- Records reflection under orthographic projection
  - From a given view direction
- What is a drawback of this?



Cube Map

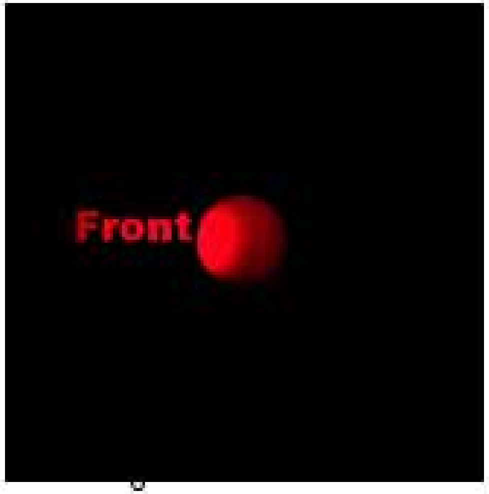
Cube mapping takes a different approach....
Imagine an object is in a box...and you can see the environment through that box

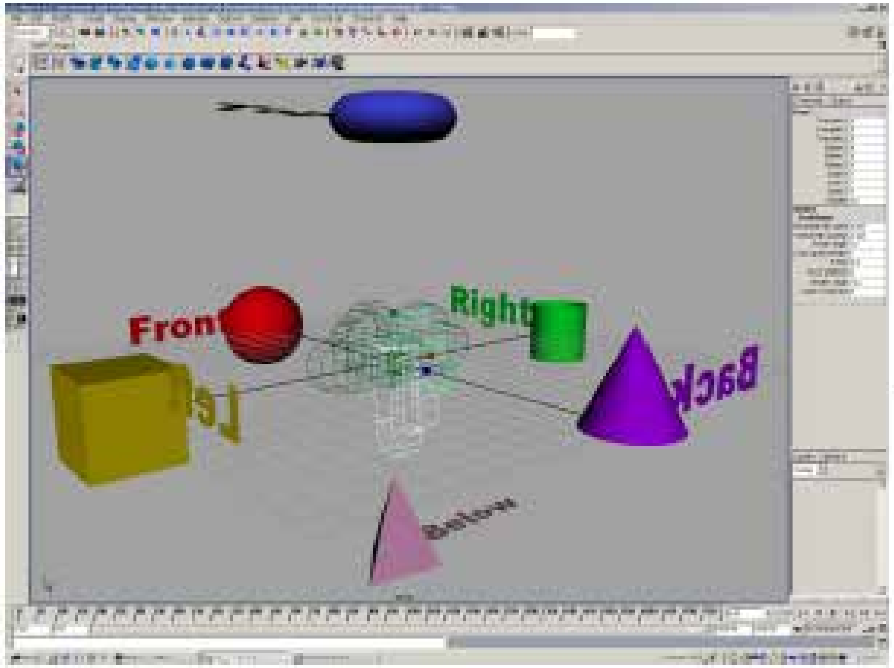


Forming a Cube Map

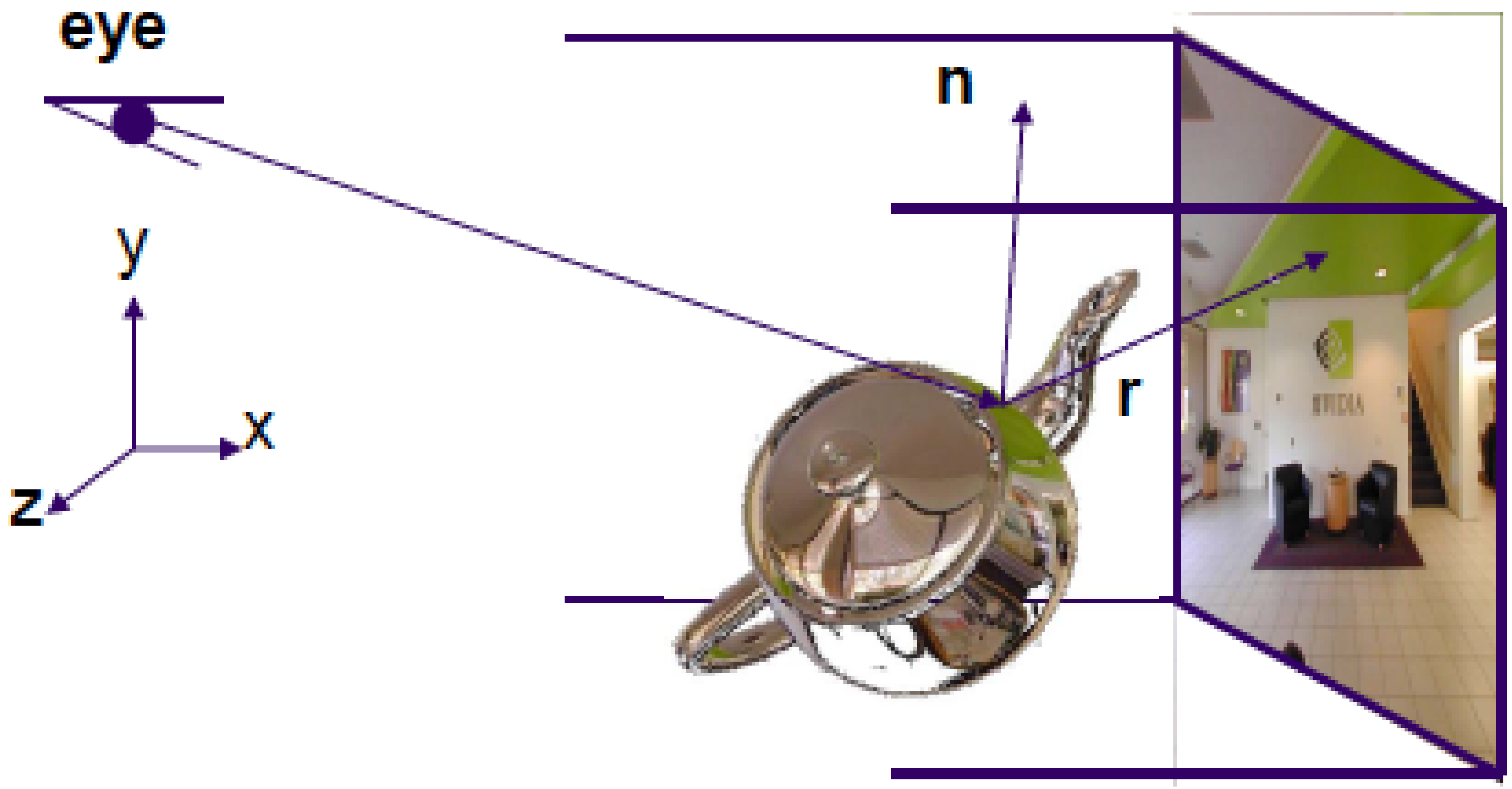
A cube map requires 6 images

Each covers a 90 degree angle from the center of the cube





Index using the Reflection Vector

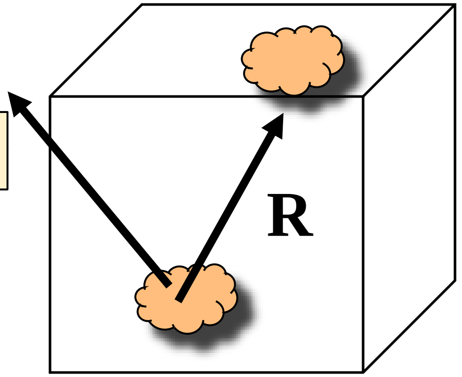


How Does WebGL Index into Cube Map?

- To access the cube map you compute
  R = 2(**N·V**)**N**-**V**

- Then, in your shader

```
vec4 texColor = textureCube(texMap, R);
```



- How does WebGL compute the index?

- Assume object at origin
- Largest magnitude component of R determines face of cube
- Other two components give texture coordinates

Indexing into a Cube Map

```
void convert_xyz_to_cube_uv(float x, float y, float z, int *index, float *u, float *v)
{
  float absX = fabs(x);
  float absY = fabs(y);
  float absZ = fabs(z);

  int isXPositive = x > 0 ? 1 : 0;
  ...
  float maxAxis, uc, vc;

  // POSITIVE X
  if (isXPositive && absX >= absY && absX >= absZ) {
    // u (0 to 1) goes from +z to -z
    // v (0 to 1) goes from -y to +y
    maxAxis = absX;
    uc = -z;
    vc = y;
    *index = 0;
  }

...

// Convert range from -1 to 1 to 0 to 1
  *u = 0.5f * (uc / maxAxis + 1.0f);
  *v = 0.5f * (vc / maxAxis + 1.0f);
}
```

Example

- R= (-4,3,-1)
- Normalize so max value has magnitude of 1
  R=(-1, ¾ , - ¼)
  - Remap texture coordinates…x,y,z are in [-1,1]
  - Need them on [0,1]
    - v = ½ + ½ x ¾ = 0.875
    - u = ½ + ½ x -¼ = 0.375
- Use face x = -1
- Texture coordinates of (u,v) = (0.375, 0.875)

WebGL Implementation

- WebGL supports only cube maps
  - vec4 texColor = textureCube(mycube, texcoord);
  - desktop OpenGL also supports sphere maps
- First must form map
  - Use images from a real camera
  - Form images with WebGL
- Texture map it to object

Vertex Shader

```
varying vec3 R;
attribute vec4 vPosition;
attribute vec4 vNormal;
uniform mat4 modelViewMatrix;
uniform mat4 projectionMatrix;
void main(){
   //...other code
   gl_Position = projectionMatrix*ModelViewMatrix*vPosition;
   vec4 eyePos  = ModelViewMatrix*vPosition;
   vec4 N = ModelViewMatrix*vNormal;
    R = reflect(eyePos.xyz, N.xyz);   }
```

Fragment Shader

```
precision mediump float;

varying vec3 R;
uniform samplerCube texMap;

void main()
{
    vec4 texColor = textureCube(texMap, R);
    gl_FragColor = texColor;
}
```

Limitations

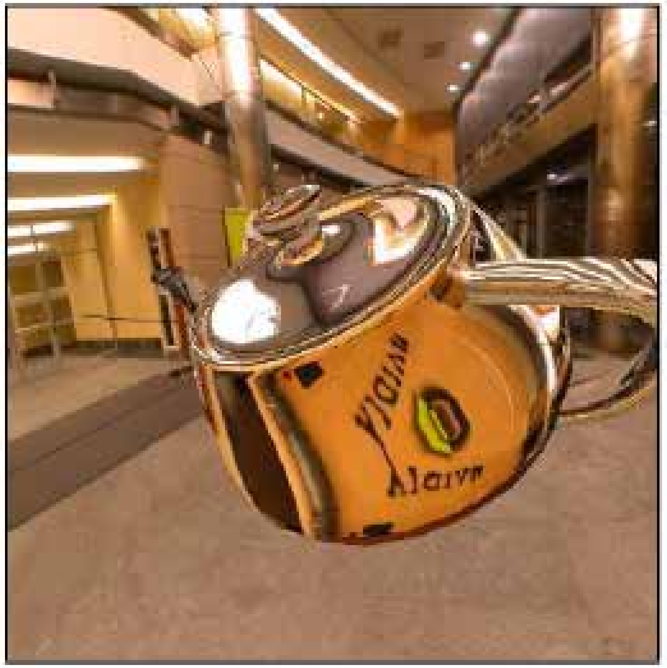
- What do you not see here that you should?

Issues

- Assumes environment is very far from object
  - (equivalent to the difference between near and distant lights)
- Object cannot be concave (no self reflections possible)
- No reflections between objects

Refraction

- Can also use cube map for refraction (transparent)



Reflection

Refraction

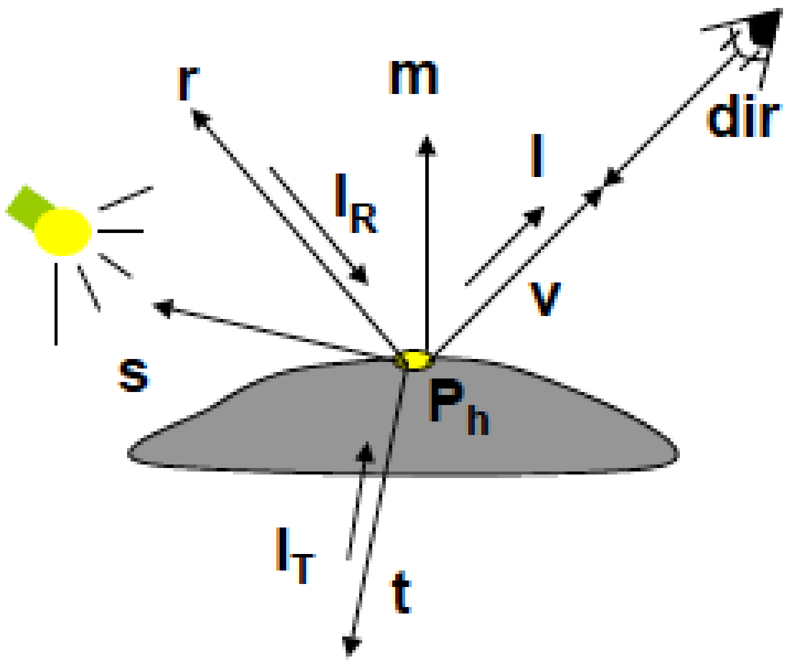
Refraction

Reflection

Refraction

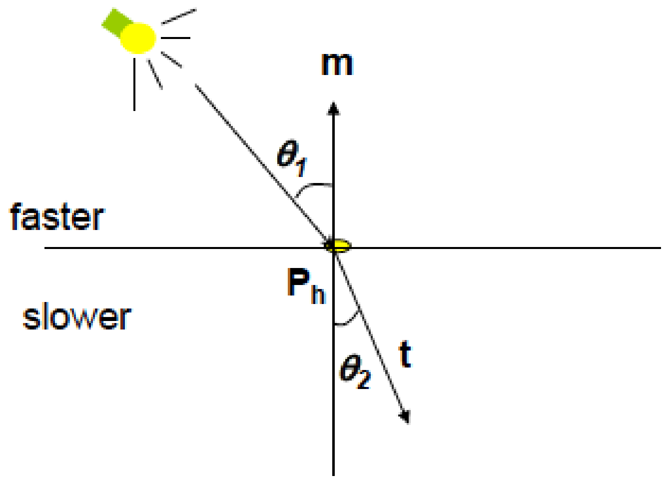
Need to Compute Refraction Vector

$$I = I_{amb} + I_{diff} + I_{spec} + I_{refl} + I_{tran}$$



Snell’s Law

- Transmitted direction obeys **Snell’s law**
- Snell’s law: relationship holds in diagram below



$$\frac{\sin(\theta_2)}{c_2} = \frac{\sin(\theta_1)}{c_1}$$

c_1, c_2 are speeds of light in medium 1 and 2

Medium is Important

- If ray goes from faster to slower medium, ray is bent **towards** normal
- If ray goes from slower to faster medium, ray is bent **away** from normal

In GLSL, the **`refract`** function expects the index of refraction to be specified as c1/c2 where:

C1 is the index of the outside medium
C2 is the index of the inside medium

So, to go from air to water you would call:

```
T=refract(V,N, 1.00/1.33)
```

Material	Refractive index
Air	1.00
Water	1.33
Ice	1.309
Glass	1.52
Diamond	2.42

The refractive index of a material is the ratio of the speed of in a vacuum to the speed of light in the medium.

For example, the speed of light through water is about ¾ the speed of light in vacuum so we have:

$$\eta = \frac{c}{\frac{3c}{4}}$$

Refraction Vertex Shader

```
void main() {
    gl_Position = Projection*ModelView*vPosition;
    vec4 eyePos  = vPosition;                // calculate view vector V
    vec4 NN = ModelView*Normal;        // transform normal
    vec3 N =normalize(NN.xyz);          // normalize normal
    T = refract(eyePos.xyz, N, iorefr);        // calculate refracted vector T
}
```

Was previously R = reflect(eyePos.xyz, N);

T is a varying….

Also eyePos.xyz needs to be the normalized view direction

Refraction Fragment Shader

```
void main()
{
    vec4 refractColor = textureCube(RefMap, T);   // look up texture map using T
    refractcolor = mix(refractcolor, WHITE, 0.3);  // mix pure color with 0.3 white

   gl_FragColor = texColor;
}
```

T is a varying….
RefMap is a uniform

What’s Wrong with this Code?

- From an actual published book…which has some good stuff in it:

7. And then in the fragment shader's main function, add the code to actually sample the cubemap and blend it with the base texture:

```
gl_FragColor = texture2D(uSampler, vTextureCoord) * textureCube(uCubeSampler, vVertexNormal);
```

8. We should now be able to reload the file in a browser and see the scene shown in the next screenshot: