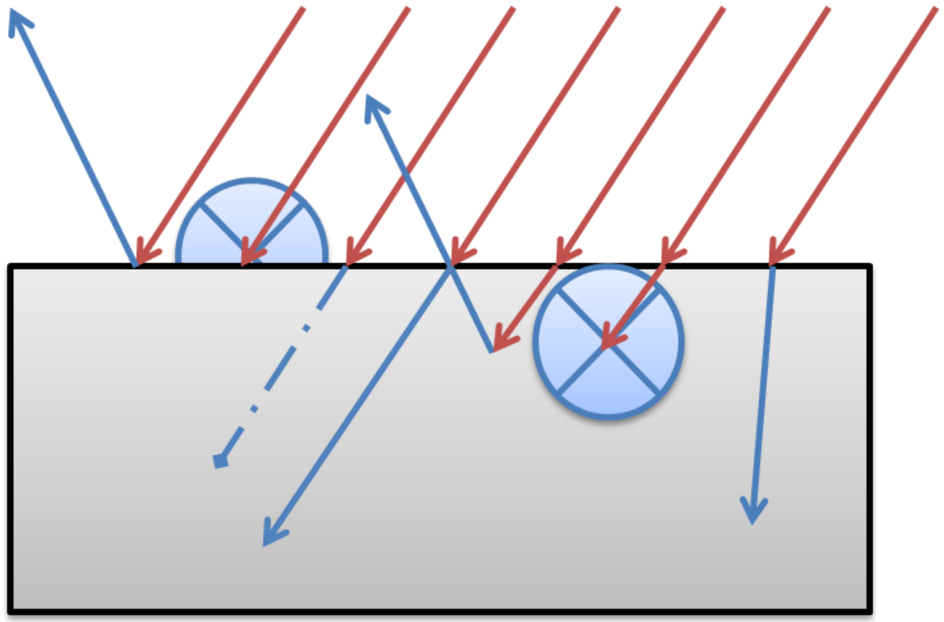
Transmittance Effects

CS7GV3 - Real-time Rendering

Light Interactions

1. Surface Reflectance
2. Surface Scattering (also reflectance)
3. Body Absorption
4. Transmission
5. Body Reflectance
6. Sub-surface scattering
7. Refraction



Refraction

- The change of direction in a light ray when crossing from one medium to another
  - Change in *optical density* causes change in velocities of the wave resulting in change of direction
- **Snell's law**: angle of incidence θ_1 and angle of refraction θ_2 are related by

$$\frac{\sin\theta_1}{\sin\theta_2} = \frac{v_1}{v_2} = \frac{n_1}{n_2} = IOR$$

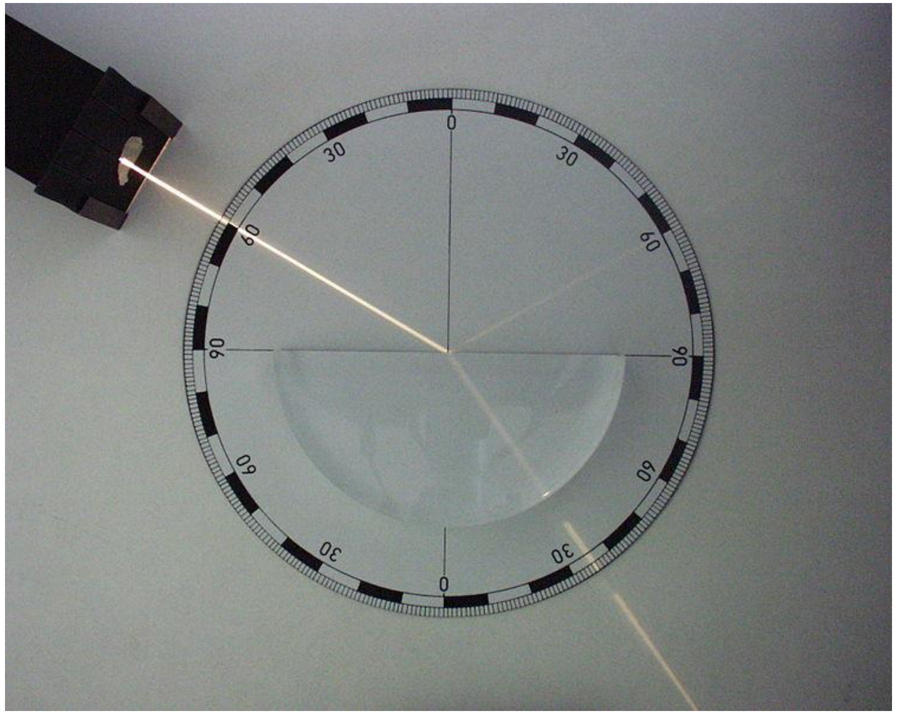
  - v_1, v_2 are velocities of the wave in respective medium and n_1, n_2 are the **refractive indices**
- Generally part of the wave is refracted and part of the wave is reflected – described by **Fresnel Equations**

Refraction

- Refraction direction ***t*** can be calculated as [Bec97]

$$\mathbf{t} = (w - k)\mathbf{N} - n\mathbf{L}$$

  - $n = n1/n2$ is the relative index of refraction
  - $w = n(\mathbf{L}.\mathbf{N})$
  - $k = n\sqrt{1 + (w - n)(w + n)}$



- GLSL provides a built in function for this
  - **T refract(T I, T N, float eta)**

Bec, Xavier, "Faster Refraction Formula, and Transmission Color Filtering," in *Ray Tracing News*, vol. 10, no. 1, January 1997 http://tog.acm.org/resources/RTNews/html/rtnv10n1.html#art3

Refraction Shader

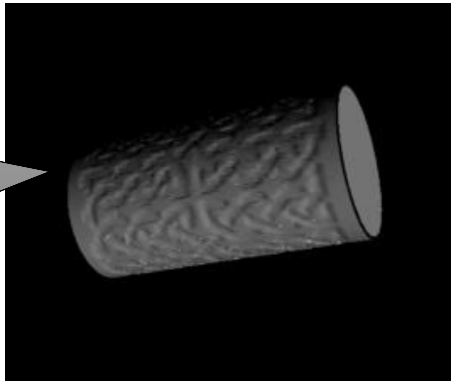
```
float eta = 0.8; // eta ratio
varying vec3 R; // refract vector
void main () {
// Create incident and normal vectors
vec4 V = gl_ModelViewMatrix * gl_Vertex ;
vec4 E = gl_ProjectionMatrixInverse
                                    * vec4 (0 ,0 , -1 ,0);
vec3 I = normalize (V.xyz - E.xyz);
vec3 N = normalize ( gl_Normal );
R = refract (I, N, eta);
gl_Position = ftransform ();
}
```

```
uniform samplerCube CubeMap ;

varying vec3 R; // refracted vector

void main () {
gl_FragColor = textureCube ( CubeMap , R);
}
```

Environment Mapping

- For transmission (and reflectance) effects, its important to have a reasonable detailed environment to reflect
- Actual geometry is expensive so an **Environment Map** is usually employed:
  - This is an approximation of incoming radiance
  - Modelled with some kind of look up function – as with texture mapping

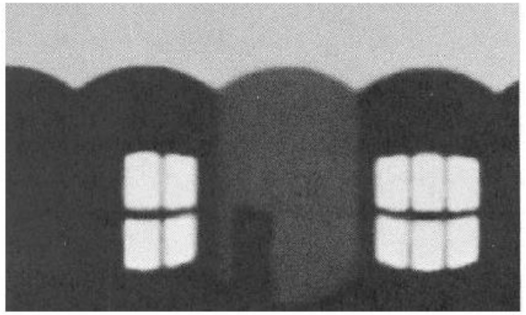
Shiny objects don’t look terribly shiny without a good model of variance in incoming radiance



Blinn and Newell Method

- First published technique
  - Use a lookup table (e.g texture) that essential represents radiance from all directions in a sphere
  - In 2D: a flattened out sphere like a *mercator* projection map stores the values
  - Access data through polar co-ordinates or **latitude and longitude**

Mercator mappings of the globe and a spherical environment map



Environment map in[Blinn Newel 76]

James F. Blinn and Martin E. Newell. 1976. Texture and reflection in computer generated images. *Commun. ACM* 19,

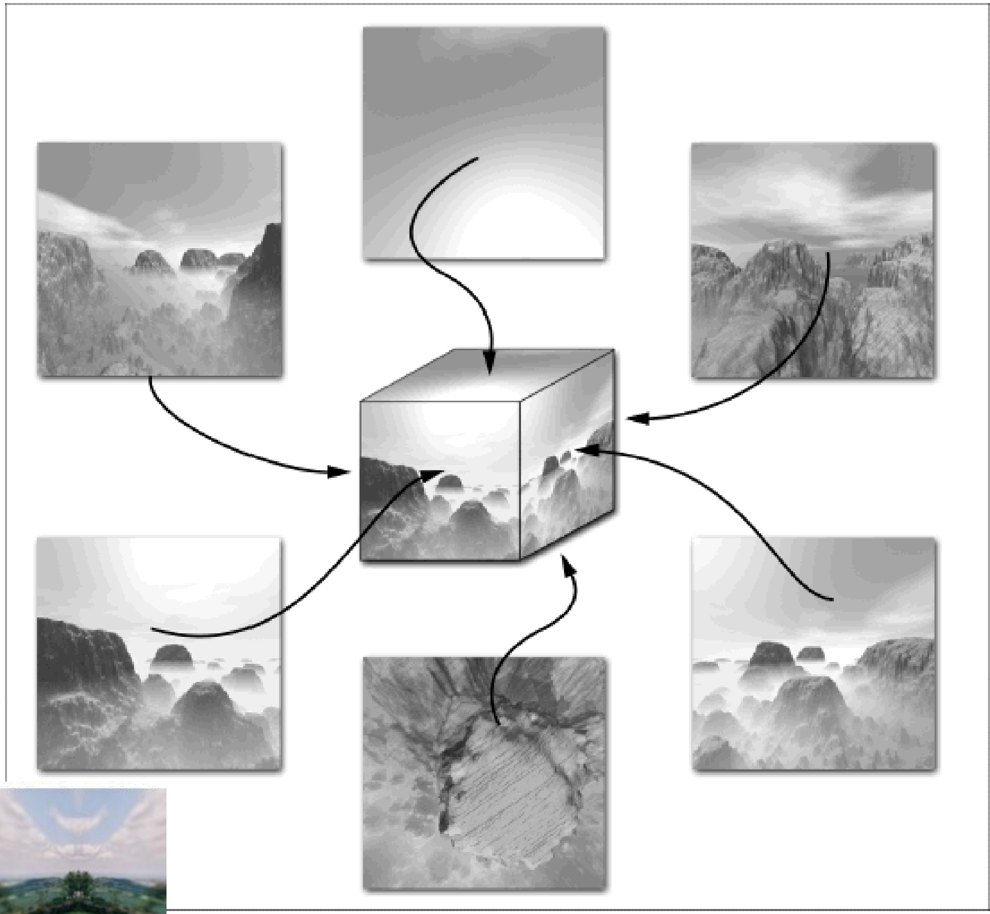
Sphere Map [Williams 83][Miller and Hoffman 84]

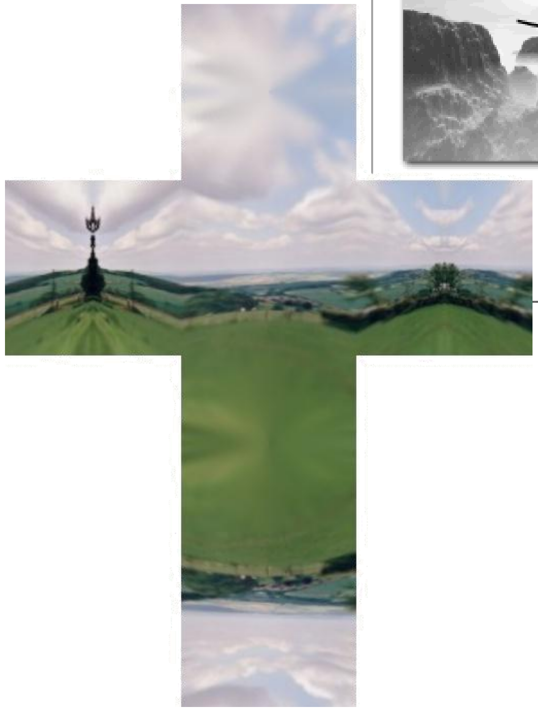
- First Environment Mapping technique generally supported in graphics hardware
  - Store radiance on an orthographically projected sphere
  - Can be obtained by photographing a reflective sphere
  - Sometimes referred to as a light probe
  - Sometimes referred to as a light probe

- Gene S. Miller and C. Robert Hoffman. **Illumination and Reflection Maps: Simulated Objects in Simulated and Real Environments.** *SIGGRAPH*1984.
- Lance Williams. **Pyramidal parametrics**. *SIGGRAPH* 1983.

Cubic Environment Map

- [Green 1986]
  - Most popular technique
  - Place camera in centre of the scene and project onto 6 cube directions
  - Advantages: view independent
  - Disadvantages: some seaming
- Can also be generated in real-time (see later lectures)

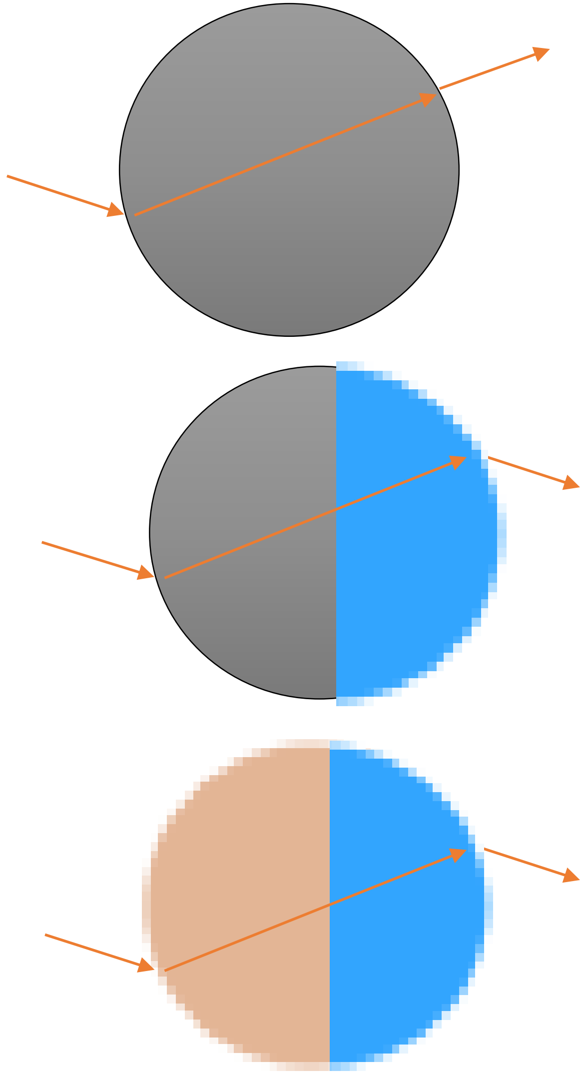




Ned Greene. 1986. Environment mapping and other applications of world projections. *IEEE Computer Graphics and Applications* journal. 6, 11 (November 1986)

Multiple Refractions

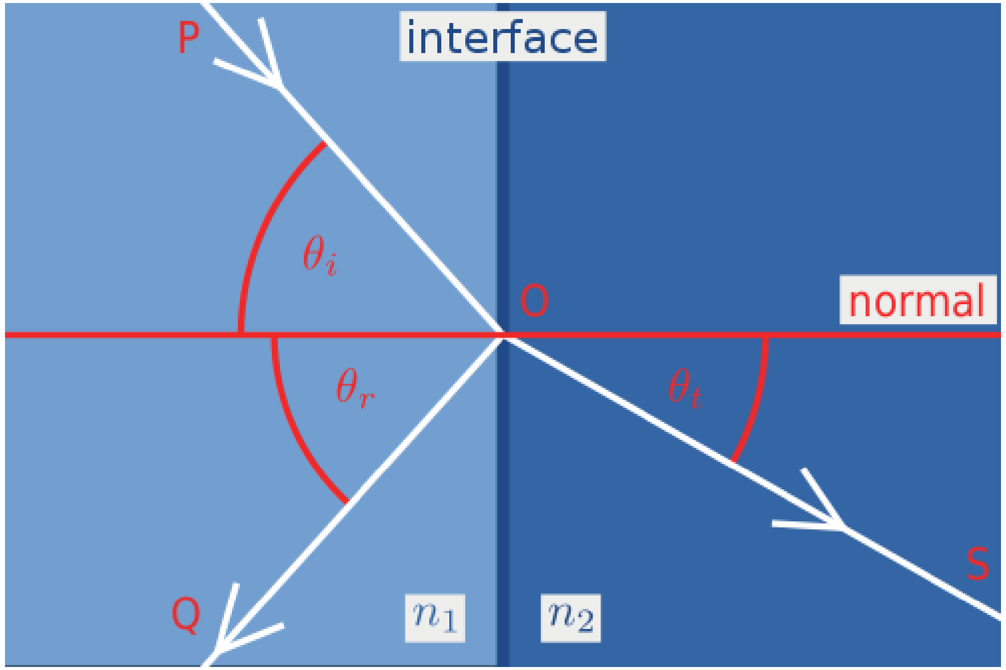
- Most real-time refraction does not take into account second redirection when object leaves the volume
  - Sometimes isn’t noticeable
- Oliveira & Brauwers technique:
  - Render backfaces and store depths and normal
  - Render frontfaces then refract
    - Treat backface z-values like a heightfield
    - March refracted ray to determine where it hits
- Davis & Wyman store both front and back as heightfield



- M. M. Oliviera & M. Brauwers: “Real-time Refraction Through Deformable Objects”, Symposium on Interactive 3D Graphics and Games 2007
- S.T. Davis & C. Wyman: “Interactive Refractions with Total Internal Reflection”, Graphics Interface 2007.

Fresnel Reflectance

- Describes surface Reflectance for a perfect surface
- When light hits surface between two media, reflectance AND transmission occur
- The amount of light reflected R_F is described by the Fresnel Equations
- This varies with angle of incidence θ



$$R_F(\theta) = \frac{1}{2}\frac{\sin^2(\phi-\theta)}{\sin^2(\phi+\theta)} - \frac{\tan^2(\phi-\theta)}{\tan^2(\phi-\theta)}$$

θ: incident angle
ϕ: asin(θ/n)
n: index of refraction

Fresnel Reflectance

- A simplified version of this equation is provided by Schlick

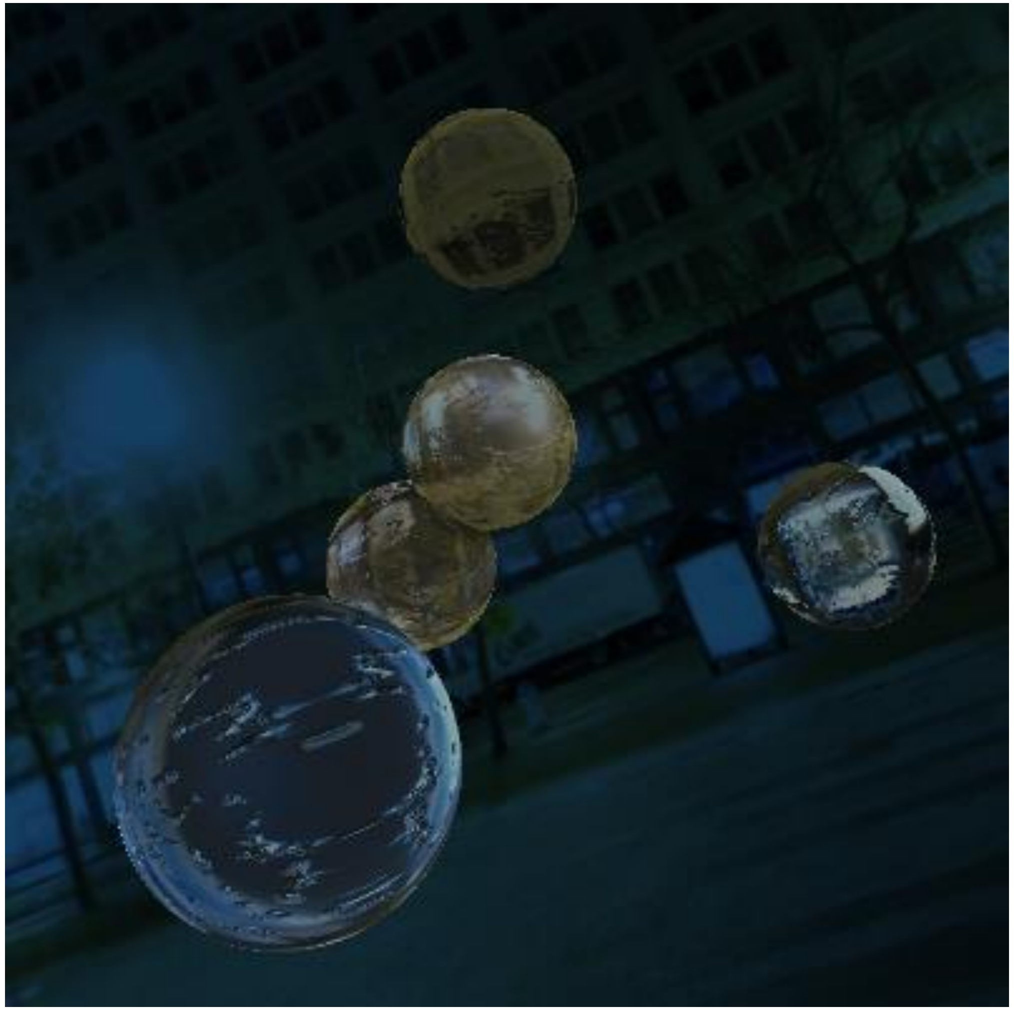
$$R_F(\theta) \approx R_F(0) + \left(1 - (\mathbf{H.N})\right)^5 \times \left(1 - R_F(0)\right)$$

- The transmitted flux is calculated as

$$L_t = \left(1 - R_F(\theta)\right)\frac{n_2^2}{n_1^2} L_i$$

Also see: http://en.wikipedia.org/wiki/Fresnel_equations

Fresnel With Chromatic Dispersion



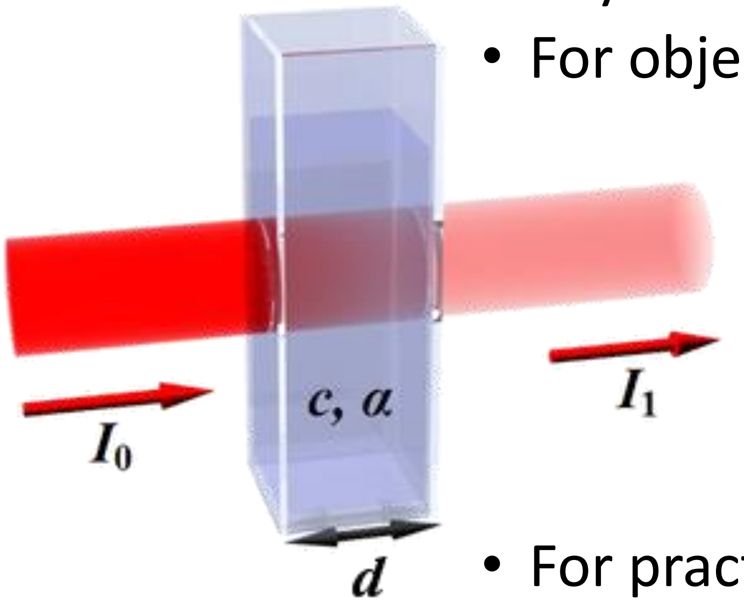
Refraction Only

With Chromatic Dispersion

Shader Setup for Illumination & Shading Models

- Vertex shader just sets up the principal vectors per-vertex for interpolation before the Fragment Shader
  - Inputs: Geometry set-up data
- Most of the work (with respect to Illumination Models) is typically done in the Fragment Shader
  - Inputs: reflectance & refraction variables; interpolated vectors for light, normal, view

Basic Transmittance



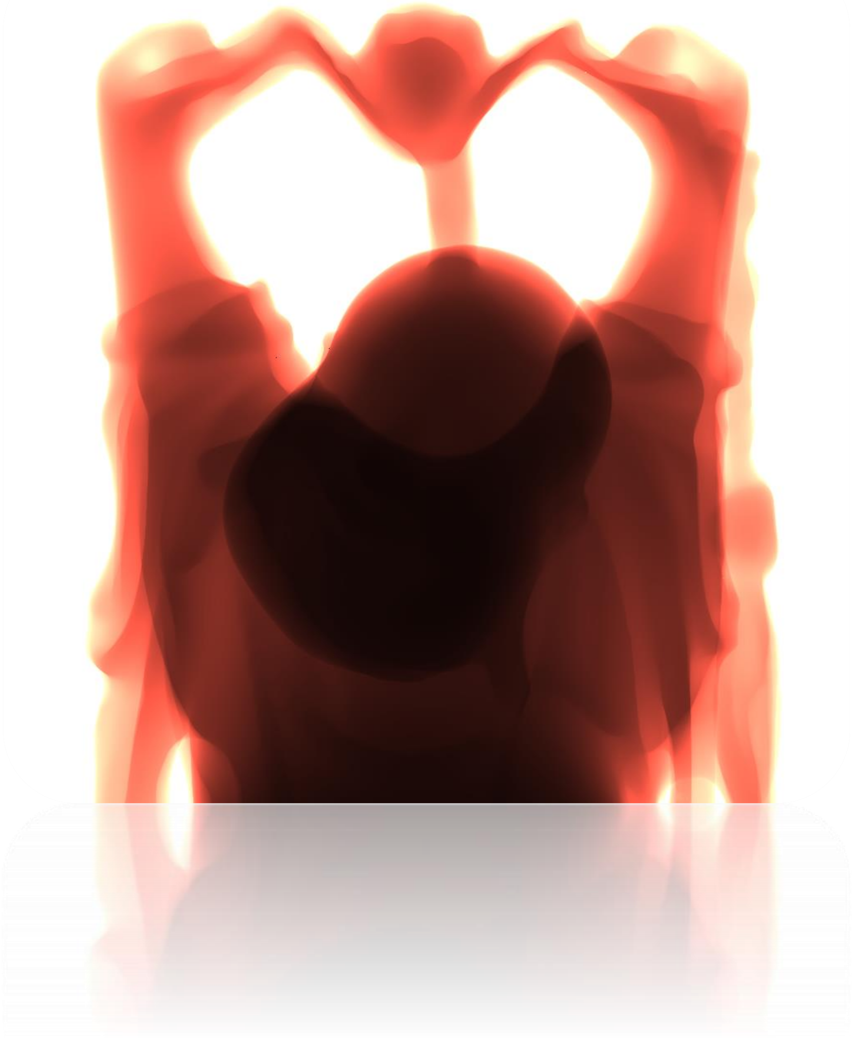
- Change in intensity or colour of light transmitted through a translucent object
  - Physically correct model should vary with thickness of object
  - For objects of varying thickness, the Baer-Lambert Law

$$T = e^{-\alpha' c d}$$

    - T : transmittance
    - α': absorption co-efficient
    - c : concentration of the absorbing material
    - d : is the distance travelled (thickness)
  - For practical rendering this could be simplified* to $T = e^{-\alpha' d_{user}}$
    - User specifies darkest transmittance filter color T for a given depth and calculate α'_R, α'_G, α'_B from this – use this at run time

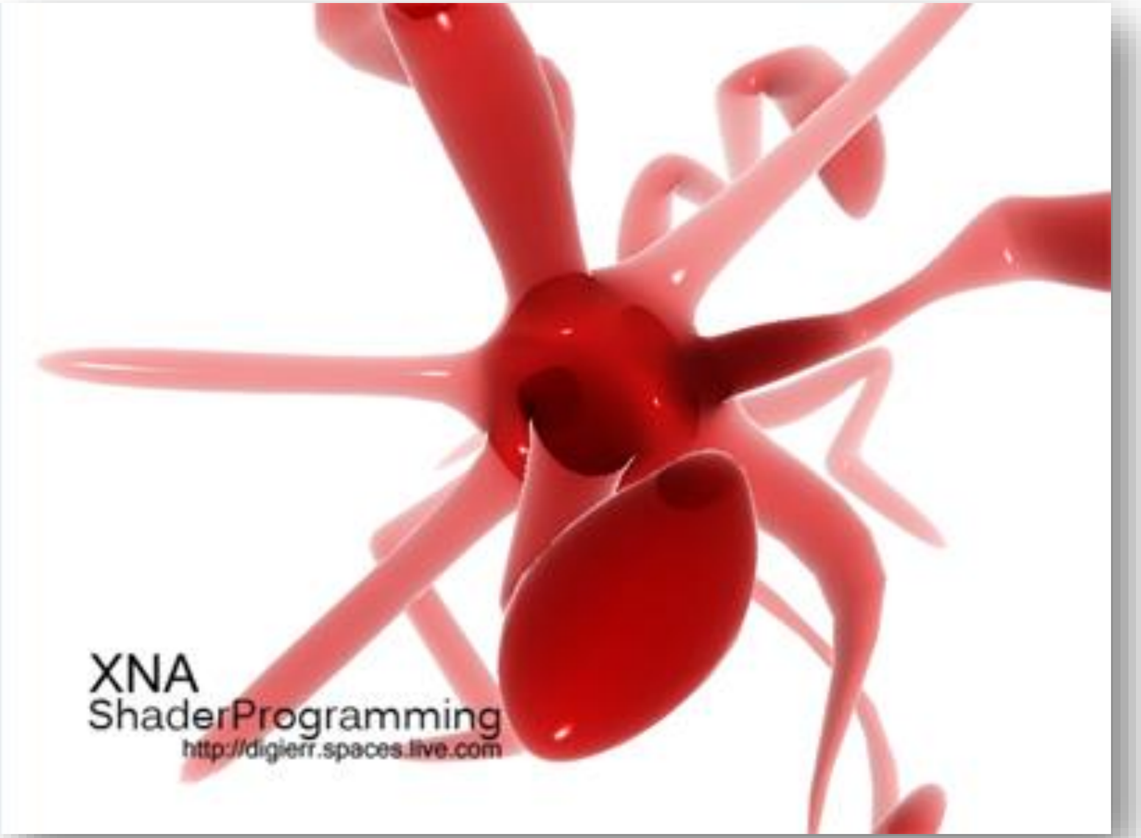
Transmittance

- Calculating d, for convex objects (2-pass):
  - Render object back face and store z_b value for each fragment
  - Render front faces and use difference in z_f
  - Then $d = \left| z_f - z_b \right|$
- For convex objects, need to apply *depth peeling*



Depth Peeling

Transmittance



For more details:

Some XNA Source available

http://xnameetingpoint.web.officelive.com/EnglishShader14.aspx

Transmittance

For more details: Some XNA Source available

http://xnameetingpoint.web.officelive.com/EnglishShader14.aspx

Transmittance

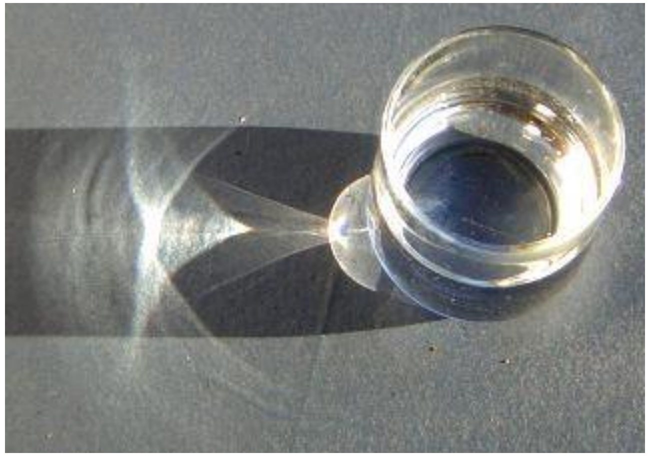
Can achieve some complex effects especially when combined with Fresnel

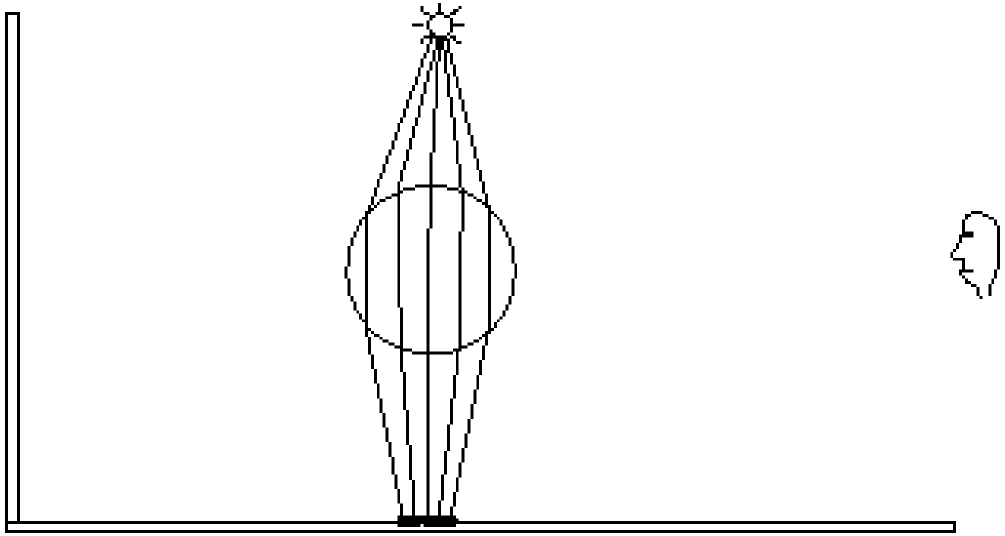
High absorption co-efficient

Low absorption co-efficient

From: Louis Bavoil, Steven P. Callahan, Aaron Lefohn, Joao L. D. Comba, and Claudio T. Silva. 2007. Multi-fragment effects on the GPU using the *k*-buffer. In *Proceedings of i3D 2007.*

Caustics

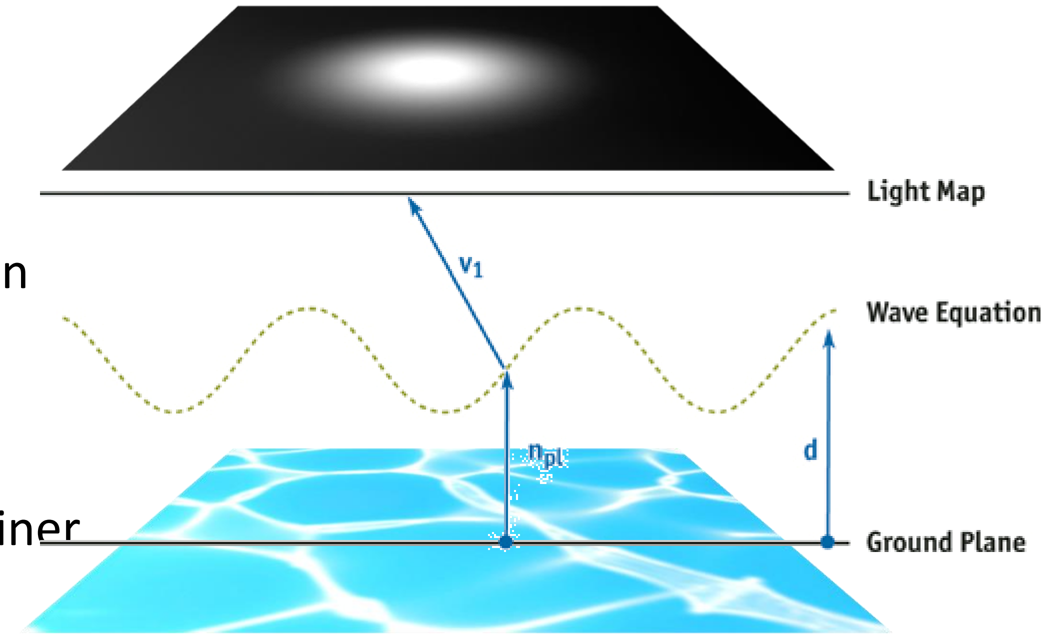




- Varying concentration of light resulting from refraction

Water Caustics

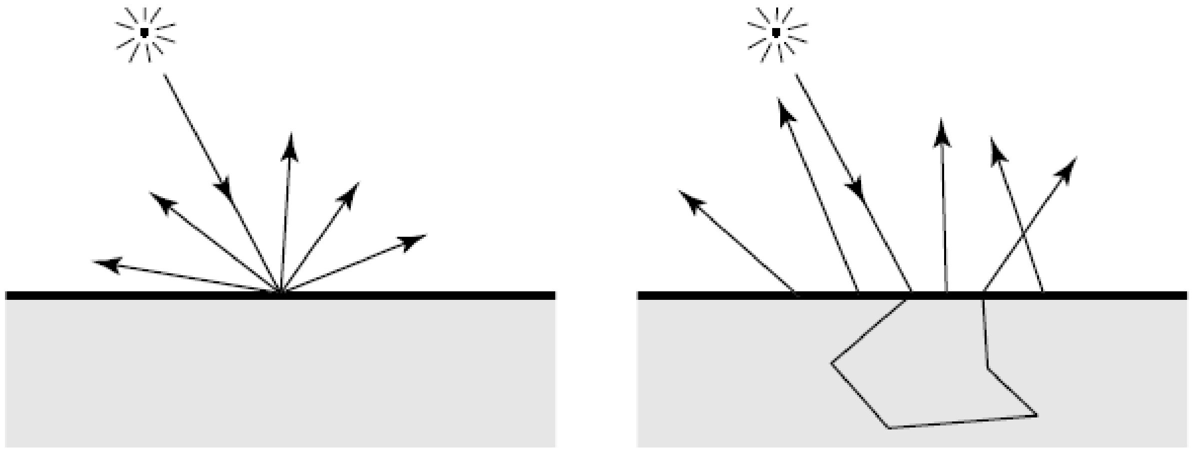
- Guardado & Sanchez-Crespo's Algorithm:
  - For each vertex in the fine mesh:
    - Send a vertical ray
    - Collide the ray with the ocean's mesh.
    - Compute the refracted ray using Snell's Law in reverse
    - Use the refracted ray to compute texture coordinates for the "Sun" map.
    - Apply texture coordinates to vertices in the finer mesh
  - Render the ocean surface



http://http.developer.nvidia.com/GPUGems/gpugems_ch02.html

Sub-surface Scattering

- If Insulator is non homogeneous
  - Transmitted light scatters (body reflectance)
  - Some light is absorbed
- Scattering **albedo** is ratio of absorption to scattering
- Direction of reflected light is not unifrom



BRDF

Real-time BSSRDF Approximation

Approximated Sub Surface Scattering in GLSL

http://machinesdontcare.wordpress.com/2008/10/29/subsurface-scatter-shader/