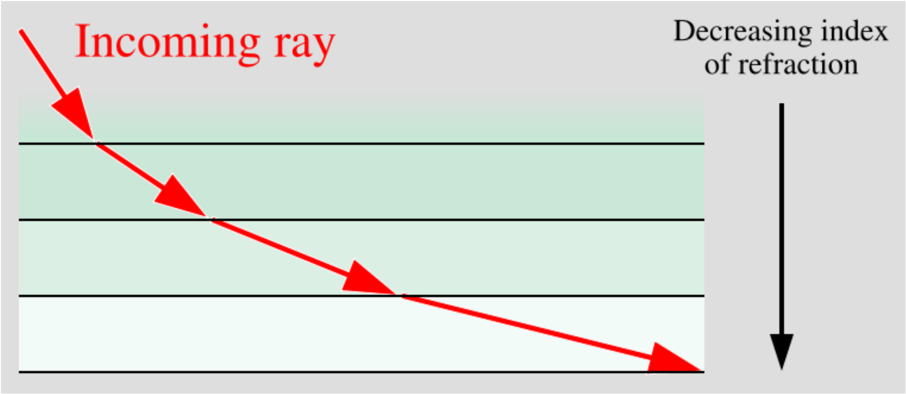
Ray Tracing

Computer Graphics
Instructor: Sungkil Lee

Overview

Overview

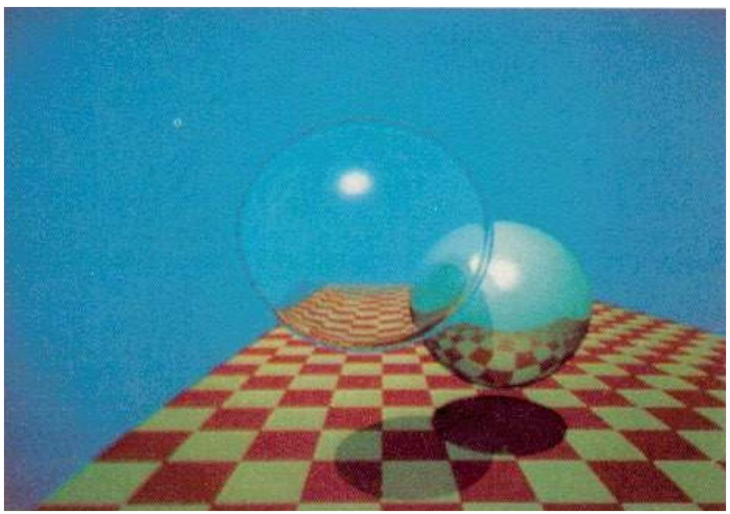
- **Ray tracing in physics**
  - a method for calculating the path of waves/particles through a system with regions of varying propagation velocity, absorption, and reflecting surfaces.
  - Ray tracing solves the problem by repeatedly advancing idealized narrow beams called **rays** through the medium by discrete amounts.
  - Simple problems can be analyzed using mathematics, but more detailed analysis can be performed by using a computer to propagate many rays.

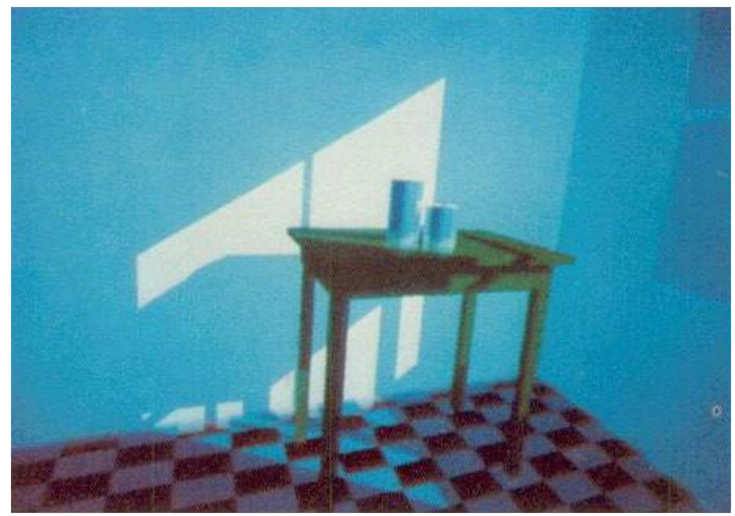


Ray tracing of a beam of light passing through a medium with changing refractive index.

Overview

- **Ray tracing in computer graphics**
  - The name stems from that of physics.
  - The ray tracing indicates a **simple ray tracing** that recursively considers only perfect reflection and refraction.
  - It is often referred to **Whitted ray tracing**.
    - An improved illumination model for shaded display [Turner Whitted, 1979]

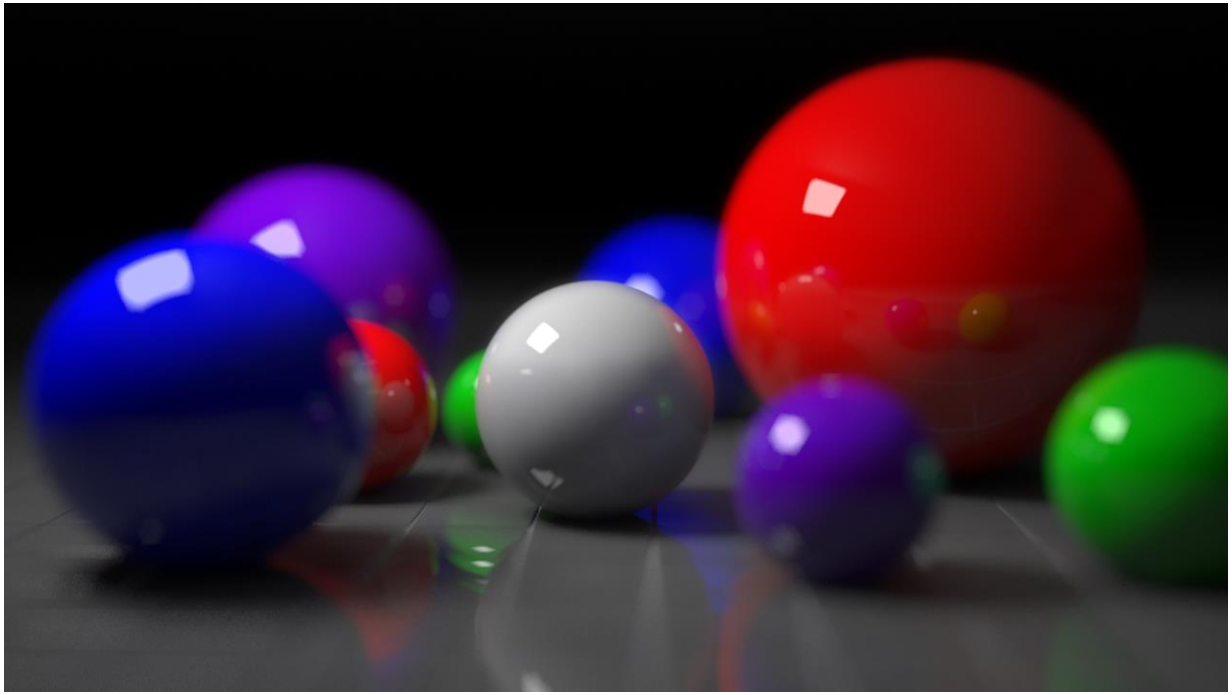




  - More general ray tracing is referred to "**path tracing**," which integrates all the types of reflections [Kajiya, 1986].

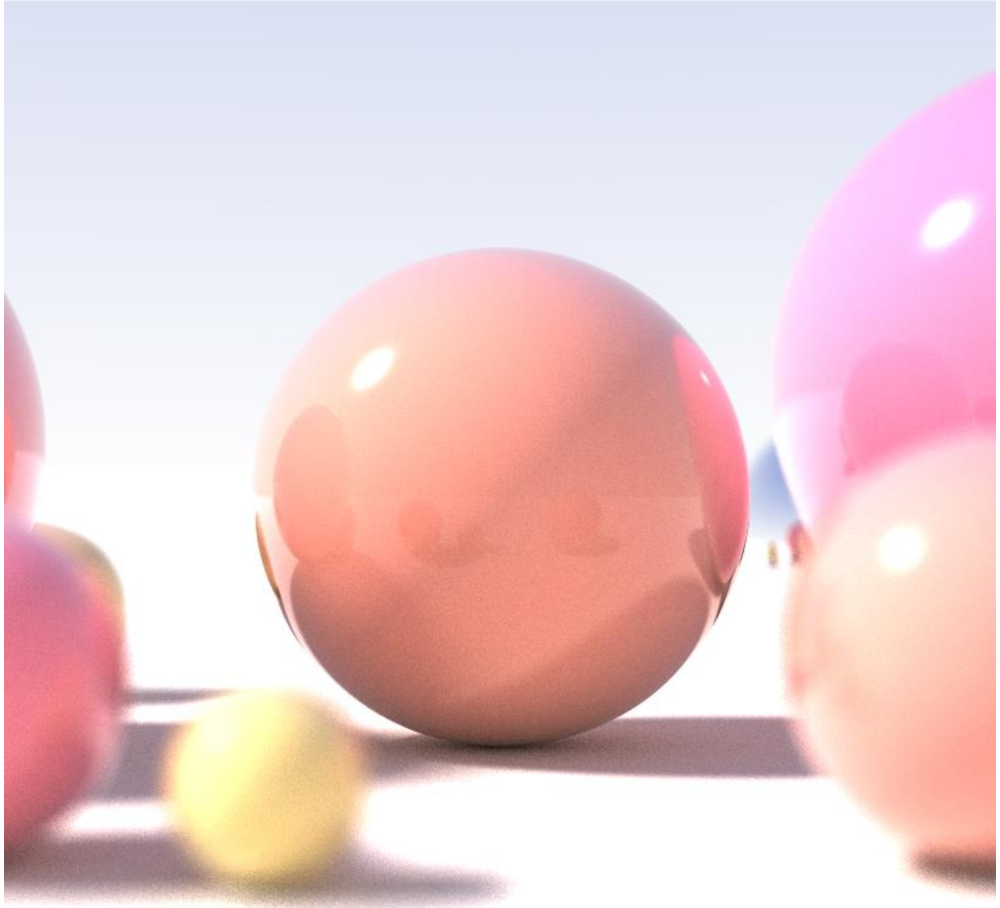
Overview

- **(Whitted) Ray tracing**
  - The most basic step before starting global illumination techniques in CG.
  - Approximate (in terms of illumination model; e.g., Phong model) but still better than pipeline/scanline approaches in quality.
  - Scene description uses not only polygonal meshes but also implicit representations (e.g., analytical spheres, surfaces, ...)



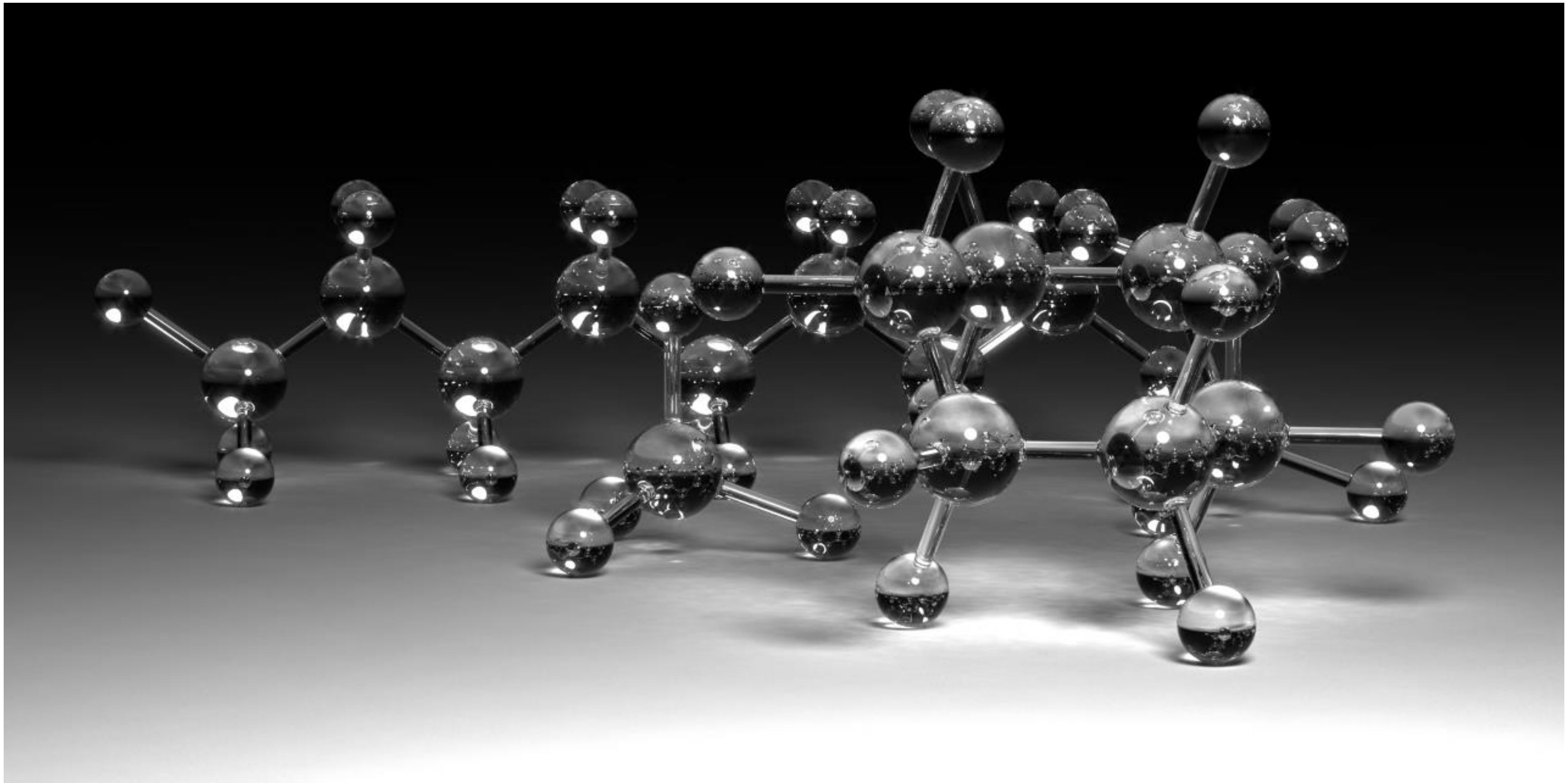
Overview

- **Effects in ray tracing (in Balls Example)**
  - Observe inter-object (mirror) reflection, soft shadows and depth-of-field (defocus blur) effect



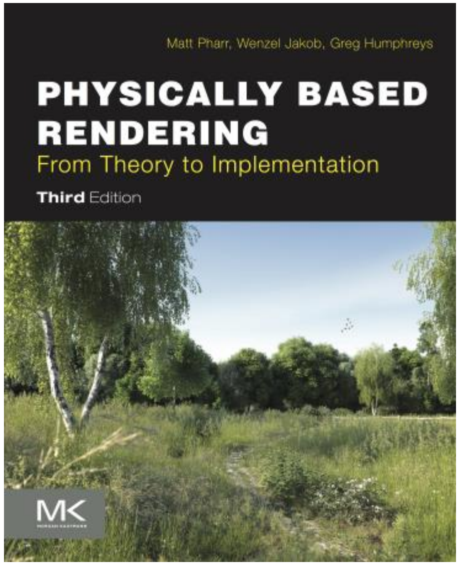
Overview

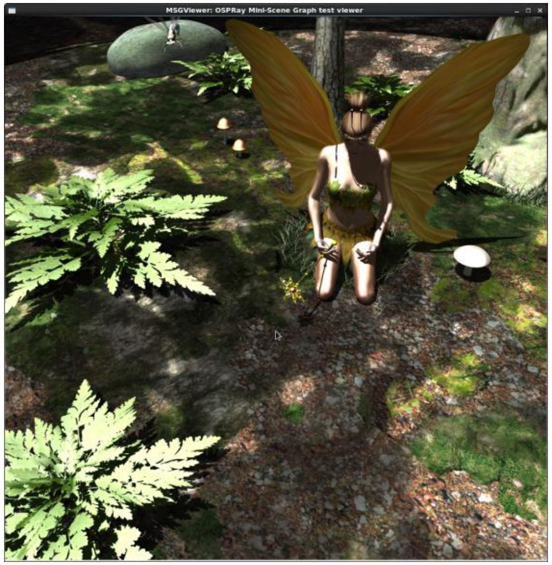
- **Effects in ray tracing (in Molecules Example)**
  - Observe the refraction inside the glass atoms and caustics (the concentration of rays to a small bright area)



Overview

- **Ray tracing implementation**
  - Usually implemented in software: slow and batch rendering
  - Common open-source software for ray tracing
    - PBRT, Povray, Intel Embree, Pixar's RenderMan, ...



PBRT book and the scenes from PovRay and Intel Embree

Overview

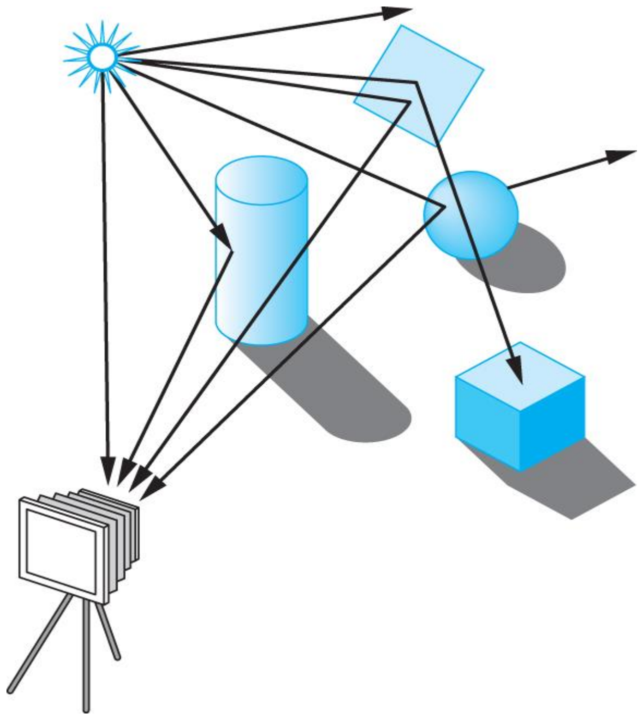
- **Ray tracing implementation**
  - Recent GPU implementations (e.g., NVIDIA RTX 2080)
    - DX12 RTX and Vulkan (since NVIDIA Volta arch.), which are the descendants of NVIDIA OptiX (written in CUDA) with ray tracing-specific APIs.


Unreal Engine demonstration with GPU Ray Tracing, *NVIDIA GTC 2018*

Ray Tracing Algorithms

Ray Tracing: Forward vs. Backward

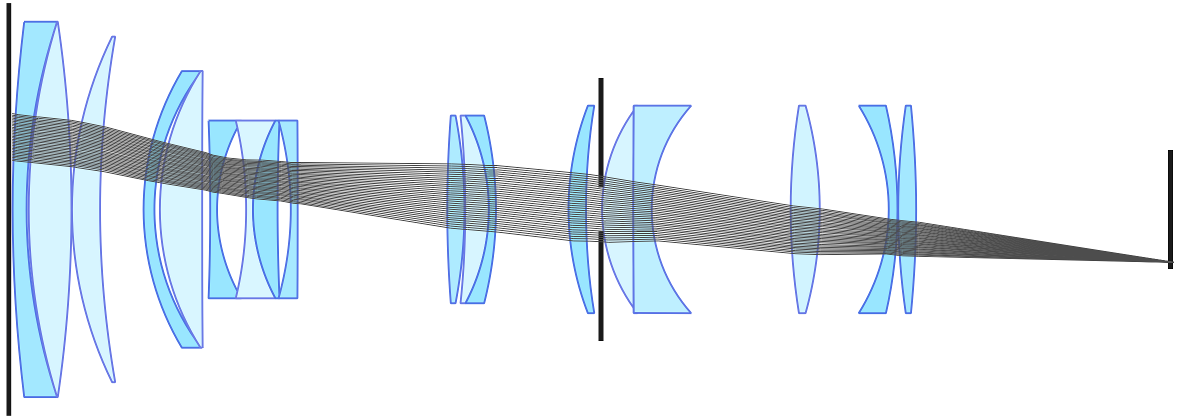
- **Forward ray tracing:**
  - Follow rays from a point light source
  - Can account for reflections and transmission
  - In general, computationally inefficient.
    - *most of rays are missing in camera*



- **An alternative for general cases:**
  - ***Backward*** ray tracing (or **ray casting**)

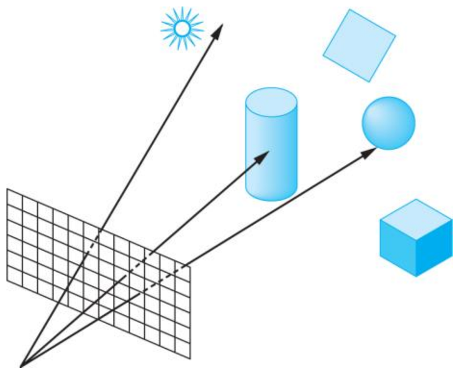
Ray Tracing: Forward vs. Backward

- **Forward ray tracing:**
  - Given a known ray path, the forward tracing can be useful.
    - e.g., optical ray tracing (Hullin et al., 2011)

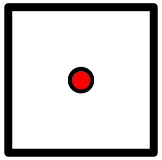


Backward Ray Tracing

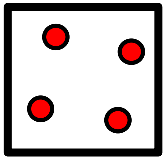
- **Only rays that reach the eye/camera matter.**
  - A ray goes in a ***reverse*** direction against the light path.



- **Need at least one ray/sample per pixel.**
  - More rays at different sampling locations suppress aliasing.



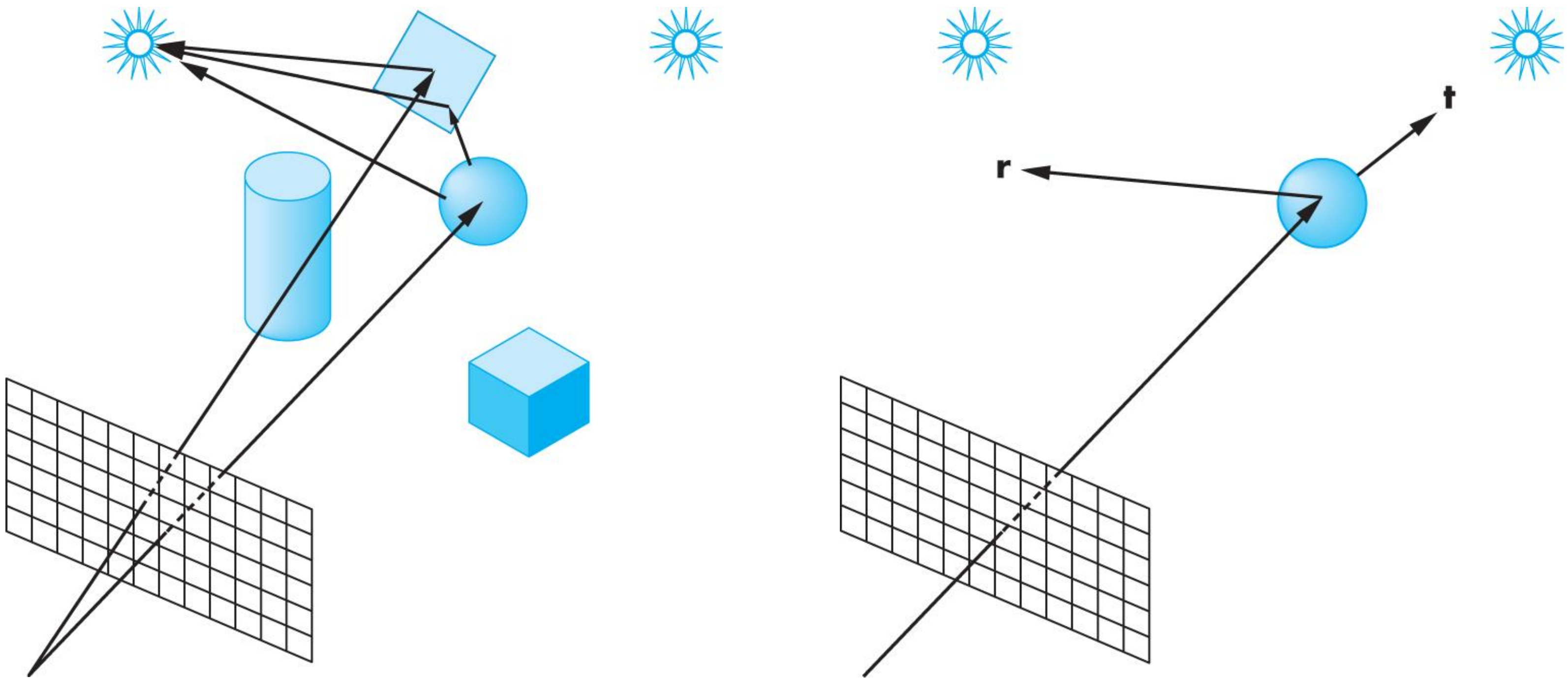
1 spp (samples per pixel)



4 spp

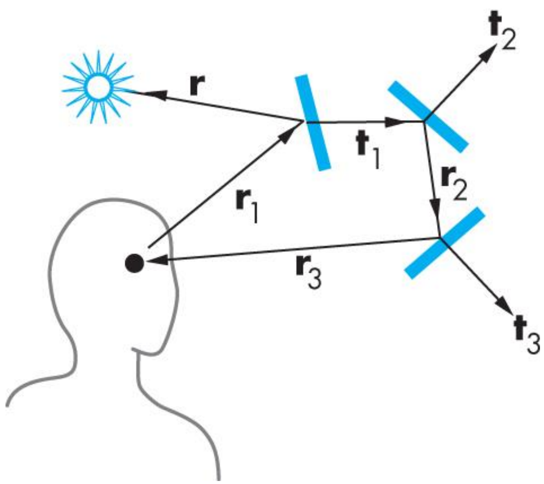
Reflection and Refraction

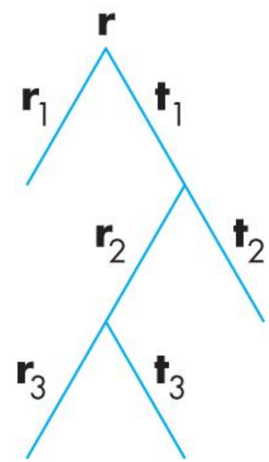
- **Reflection and refraction (transmission)**
  - In a simple ray tracing, only the mirror reflection is traced.



Ray Tree

- **A binary tree can express the recursive ray tracing**
  - Since there are only a single reflection, the binary tree is enough.
  - The tree is simple, so, we can recursively traverse the tree.

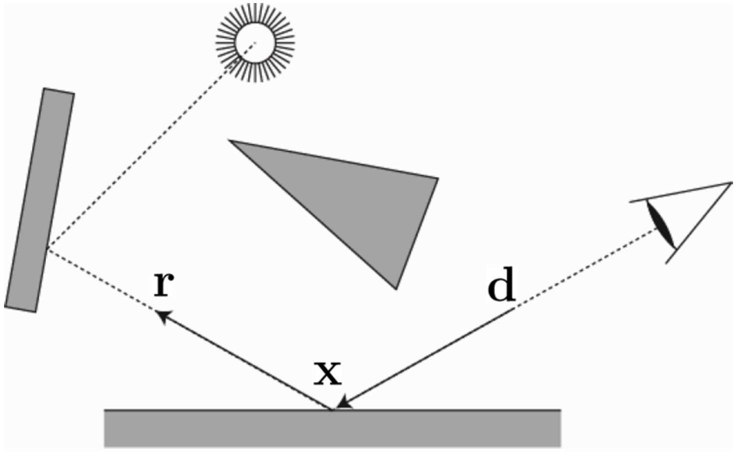




  - For the path tracing, the tree can expand beyond the binary tree.
  - Then, the recursive tracing might not fit with memory, which we need to come up with an alternative approach (e.g., Monte-Carlo solutions).

Reflection

- **Mirror/perfect reflection**

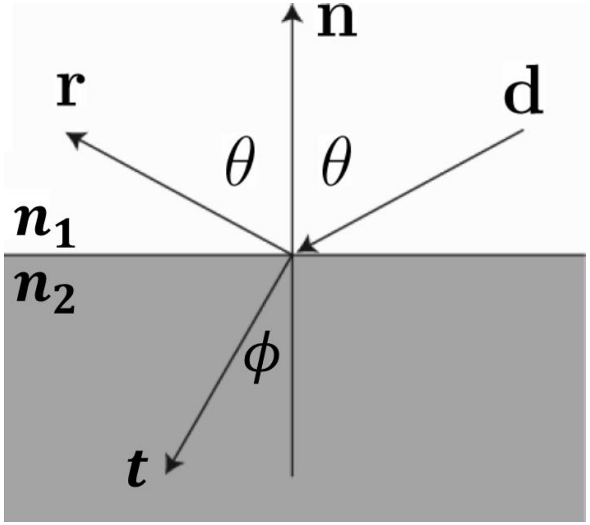


  - Recall how to derive a reflection vector in Phong illumination model.
  - Given the incoming direction $\mathbf{d}$ and the normal $\mathbf{n}$, the outgoing vector $\mathbf{r}$ is:

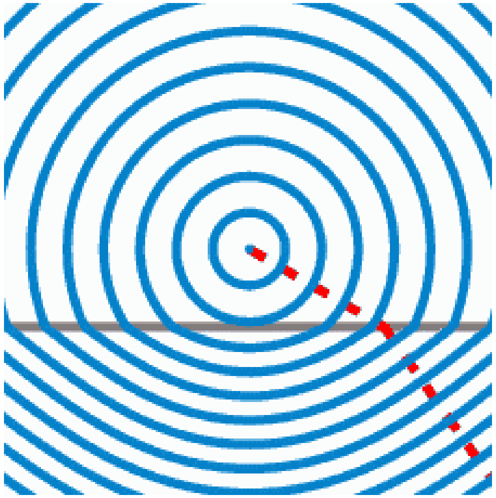
$$\mathbf{r} = \mathbf{d} - 2(\mathbf{d} \cdot \mathbf{n})\mathbf{n}$$

Refraction

- **Dielectric materials**
  - Insulators that can be polarized by an electric field.
  - Examples: diamond, glass, water, air
- **Snell's law**
  - Light travels slower at a denser dielectric medium than the vacuum.
  - Light is bent when it goes from one medium to another.
  - Given the refractive indices n_1 and n_2, the relationship is given by:



$$n_1 \sin\theta = n_2 \sin\phi$$



Ray Tracing Pseudocode

Pseudocode of Ray Tracing

- **Given a scene representation, a main function can be:**

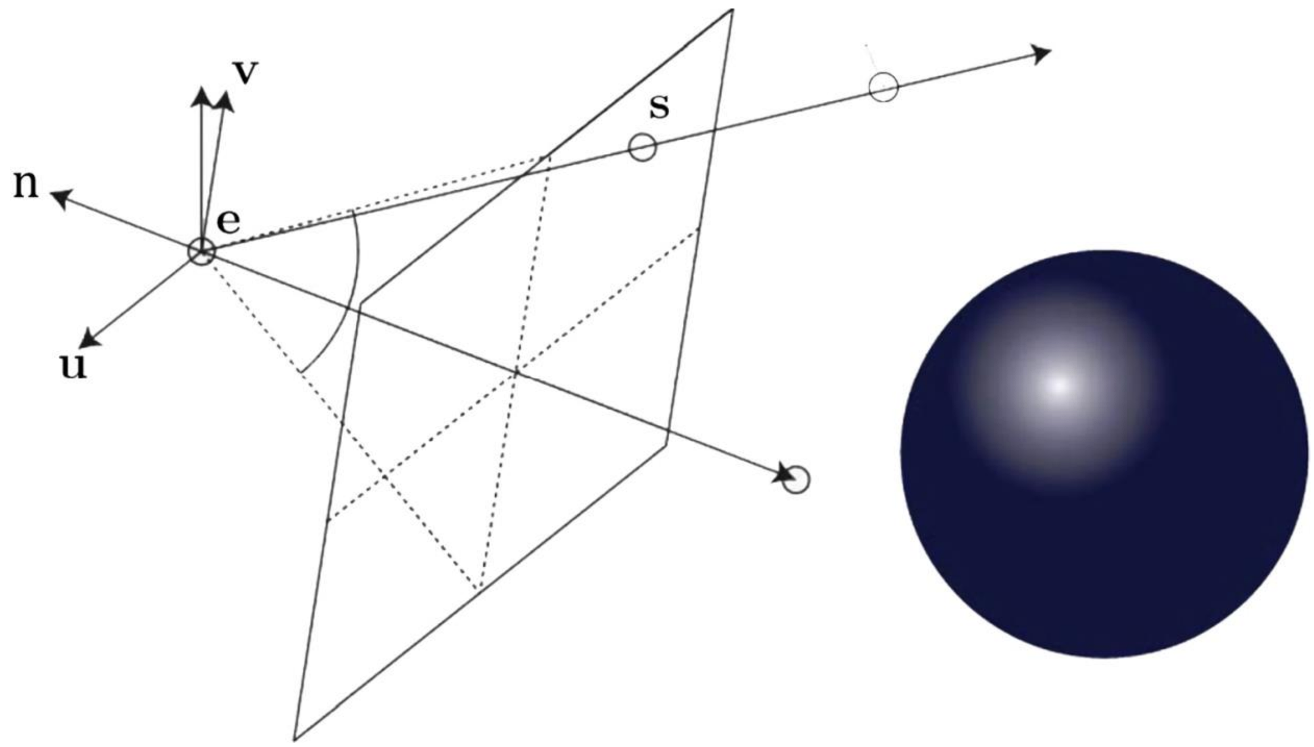
```
ray_trace()
{
    // acceleration structure (e.g., KD-tree or BVH)
    construct scene representation

    // main recursive loop
    for each pixel
    {
        generate primary ray
        color c = trace( primary ray, 0 )
        assign c to the current pixel
     }
}
```

Primary Rays

- **A primary ray p:**
  - emanates from e to s is defined as:

$$\boldsymbol{p}(t) = \boldsymbol{e} + t(\boldsymbol{s} - \boldsymbol{e})$$



Primary Rays

- **We derive primary rays in the camera frame, but transform them to the world frame.**

$$\boldsymbol{p}(t) = \boldsymbol{e} + t(\boldsymbol{s} - \boldsymbol{e})$$

  - Eye position $\boldsymbol{e}$: simply the origin at the camera frame
  - Given an screen coordinate $(u, v) \in [0,1]^2$ for perspective viewing,
    - a 3D pixel position $\boldsymbol{s}$ at $z = -1$ can be:

$$\boldsymbol{s}(x, y, z) = (\tan(0.5 \cdot fovx) * (u - 0.5), \tan(0.5 \cdot fovy) * (v - 0.5), -1)$$

  - Transform $\boldsymbol{s}$ and $\boldsymbol{e}$ to the world frame by multiplying inverse view matrix.
    - We prefer intersection tests in the world frame to in the camera frame.
    - Otherwise, all the objects require to be transformed to the camera frame (as done in the pipeline), but it's too costly in ray tracing.

Pseudocode

- **Given a scene representation, trace() can be:**
  - See the details in the following pages.

```
color trace( ray i, int step )
{
    if( step > max ) return background_color;

    status s = intersect(i,q); // q: output ray
    if(s==light_source)    return light_source_color;
    if(s==no_intersection) return background_color;

    vec3 n = get_face_normal(q); // do not use a vertex normal
    vec3 r = reflect(q,n);
    vec3 t = refract(q,n);

    color local = phong_shade(q,n,r);
    color reflected = trace(ray(q,r), step+1);
    color refracted = trace(ray(q,t), step+1);

    return local + reflected + refracted;
}
```

Stop Condition in Recursion

```
color trace( ray i, int step )
{
    if( step > max ) return background_color;
    ...
    status s = intersect(i,q); // q: output ray
    if(s==light_source)    return light_source_color;
    if(s==no_intersection) return background_color;
}
```

- **When to stop:**
  - Count steps (or recursion depth)
    - Typical option for implementation; e.g., 15 to 20
  - Ignore rays that go off to infinity or to light sources
    - Put large sphere around problem
  - Some light will be absorbed at each intersection
    - Track amount left

Intersection

- **Rays are tested with primitives to find intersections.**

```
color trace( ray i, int step )
{
    ...
    status s = intersect(r,q);
    ...
}
```

- **There are many solutions on various primitives.**
  - Refer to "Graphics Gems" Books or other notes.
  - An acceleration will be handled in the acceleration structures.

```
point intersect( ray i, out int status )
{
    // implemented for plane, triangles, sphere, surfaces
}
```

Ray-Triangle Intersection Test

- **Here, we see a way of testing intersection between a ray and a triangle.**
  - Parametric description of a triangle with triangle vertices $\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}$.

    $$\boldsymbol{p} \;=\; \boldsymbol{a} + \beta(\boldsymbol{b} - \boldsymbol{a}) + \gamma(\boldsymbol{c} - \boldsymbol{a}), \text{ where } \beta > 0, \gamma > 0, \beta + \gamma < 1$$
  - Ordering convention: counter-clockwise ordering of the vertices.
  - Intersection condition for the primary ray :

    $$\boldsymbol{e} + t\boldsymbol{d} \;=\; \boldsymbol{a} + \beta(\boldsymbol{b} - \boldsymbol{a}) + \gamma(\boldsymbol{c} - \boldsymbol{a})$$
  - The condition can be written in matrix form, and solve for the three unknowns β, γ, t, in matrix form:

$$\begin{bmatrix} a_x - b_x & a_x - c_x & d_x \\ a_y - b_y & a_y - c_y & d_y \\ a_z - b_z & a_z - c_z & d_z \end{bmatrix} \begin{bmatrix} \beta \\ \gamma \\ t \end{bmatrix} = \begin{bmatrix} a_x - e_x \\ a_y - e_y \\ a_z - e_z \end{bmatrix}$$

Reflection

- **Reflection is computed at the intersection**

```
color trace( ray i, int step )
{
    ...
    vec3 n = get_face_normal(q); // do not use a vertex normal
    vec3 r = reflect(q,n);
    vec3 t = refract(q,n);

    color local = phong_shade(q,n,r);
    color reflected = trace(ray(q,r), step+1);
    color refracted = trace(ray(q,t), step+1);
    return local + reflected + refracted;
}
```

  - For triangle, face normal is computed by the cross product from vertices.

- **Example implementation of** `reflect()`

```
// I: incident vector, N: normal
vec3 reflect( vec3 I, vec3 N ){ return I-N*dot(I,N)*2.0f; }
```

Transmission/Refraction

- **Some rays are refracted at intersection.**

```
color trace( ray i, int step )
{
    ...
    vec3 n = get_face_normal(q); // do not use a vertex normal
    vec3 r = reflect(q,n);
    vec3 t = refract(q,n); // need to provide refractive index
    color local = phong_shade(q,n,r);
    color reflected = trace(ray(q,r), step+1);
    color refracted = trace(ray(q,t), step+1);
    return local + reflected + refracted;
}
```

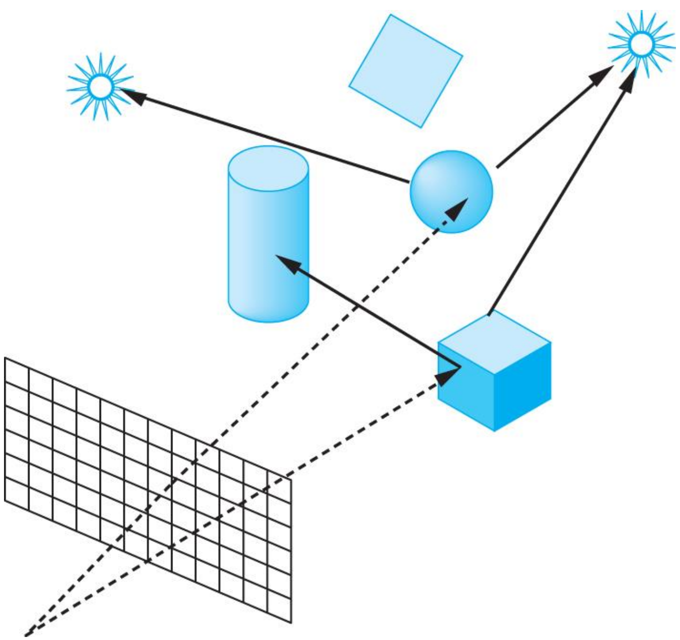
- **Example implementation of** `refract()`

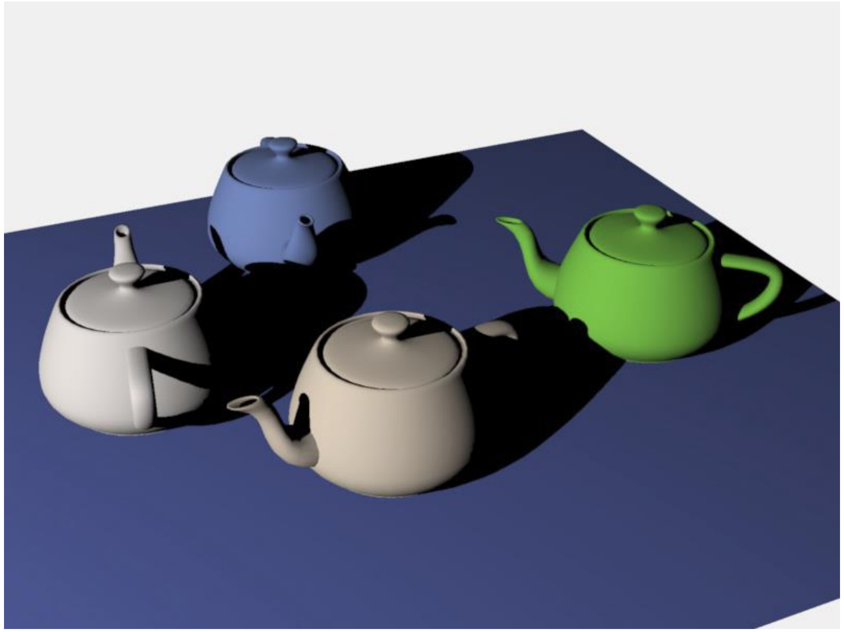
```
// I: incident vector, N: normal
vec3 refract( vec3 I, vec3 N, float eta /* = n0/n1 */ )
{
    float d=dot(I,N), k=1-eta*eta*(1-d*d);
    return k<0?0:(I*eta-N*(eta*d+sqrt(k)));
}
```

Shadows

Shadows

- **Shadow Rays**
  - Even if a point is visible, it will not be lit unless we can see a light source from that point.
  - Cast shadow rays to light sources for each intersection surface point.
    - For multiple light sources, we repeat the shadow ray casting.
    - Could be very slow with many light sources (i.e., soft shadows)

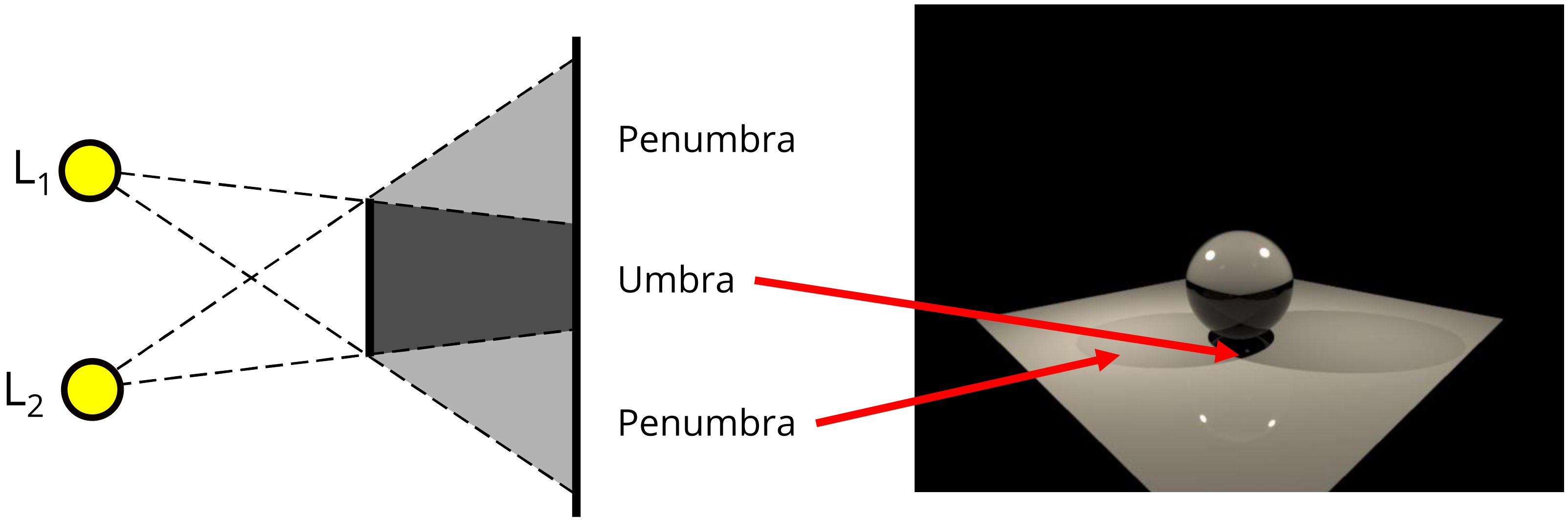




Example of hard shadows

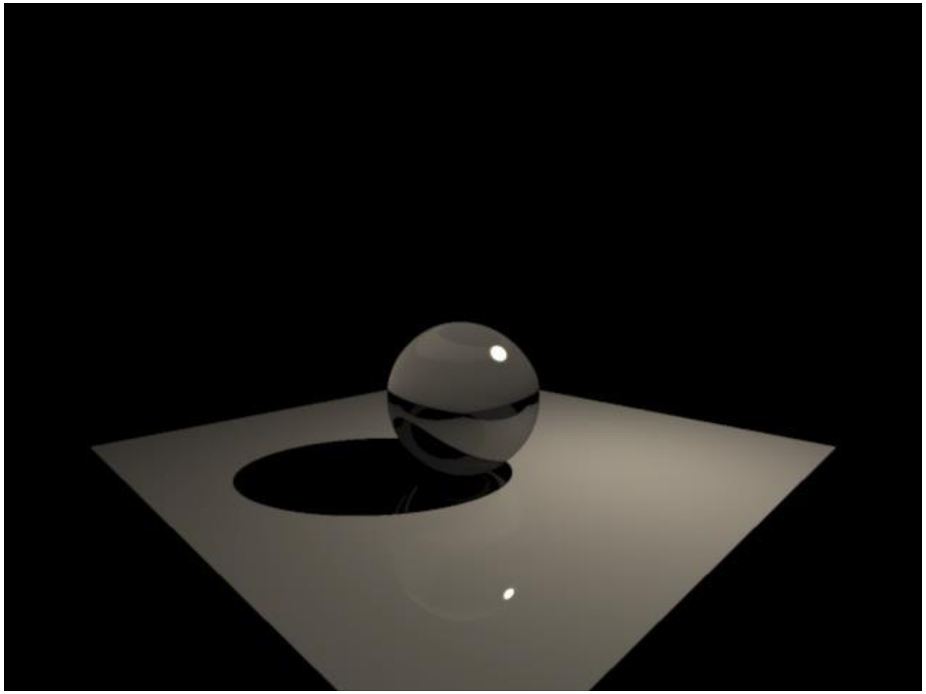
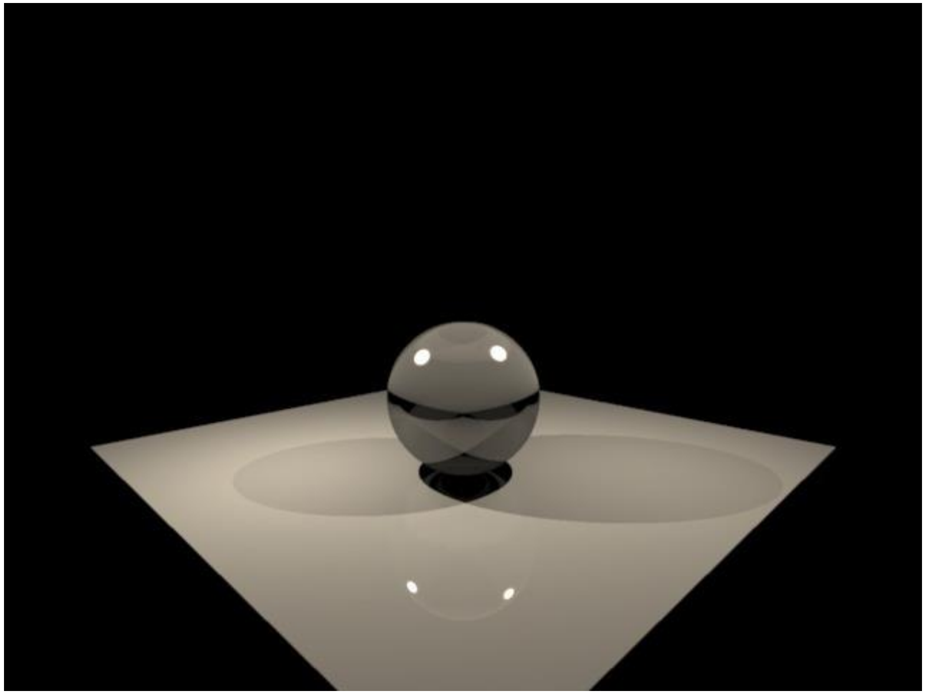
Soft Shadows

- **Soft shadows from multiple light sources**
  - ***Umbra***: Innermost/darkest part of a shadow, completely occluded from the light source
  - ***Penumbra***: the region in which only a portion of the light source is obscured by the occluders



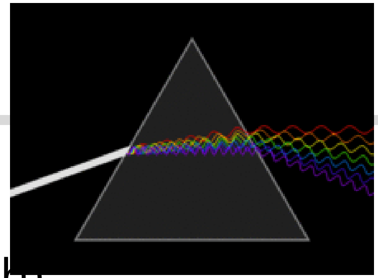
Examples

- **A glass sphere with one vs. two light sources**
  - Observe reflection and shadows (umbra and penumbra).

More on Reflection/Refraction

Dispersion

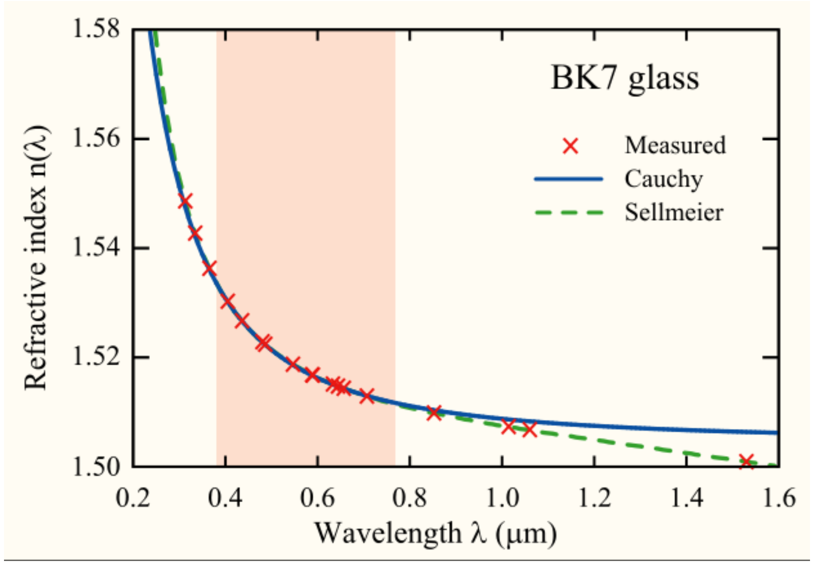


- **Refraction index varies with wavelength.**
  - Typically, 1.5 for glass at $\lambda = 587.6$ nm (reference wavelength)
    - but it may significantly change with other wavelengths.

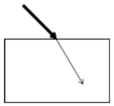
Dispersion

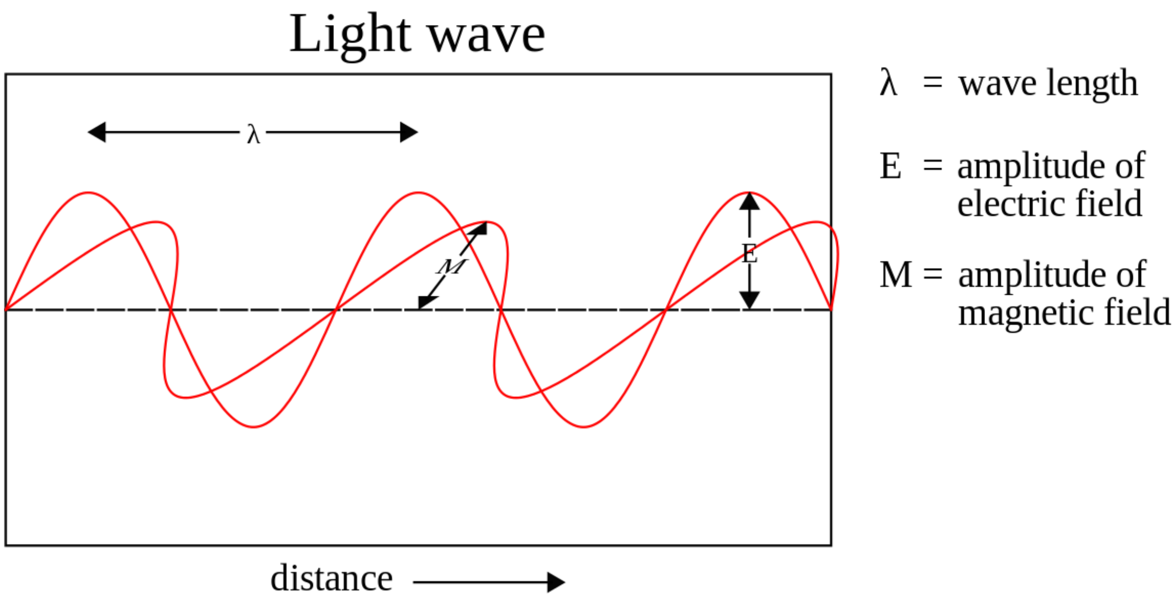
- **Refraction index varies with wavelength.**
  - Analytical formulae (Sellmeier or Cauchy equation) can be used to approximate dispersion.
  - Example: Sellmeier's equation requiring 6 coefficients

$$n^2(\lambda) = 1 + \frac{B_1\lambda^2}{\lambda^2 - C_1} + \frac{B_2\lambda^2}{\lambda^2 - C_2} + \frac{B_3\lambda^2}{\lambda^2 - C_3}$$

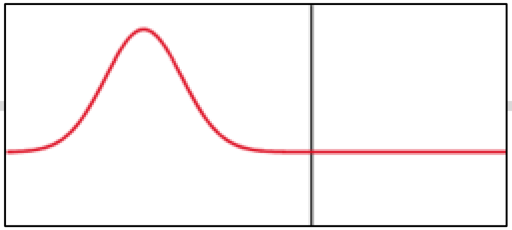


Fresnel Equations

- **For specular refraction (), it computes how much light is reflected and refracted.**
  - Derived from the electromagnetic wave equations
  - Involved polarization of the waves
  - While natural light is partially polarized, direct sunlight or reflections from dielectrics are highly polarized.



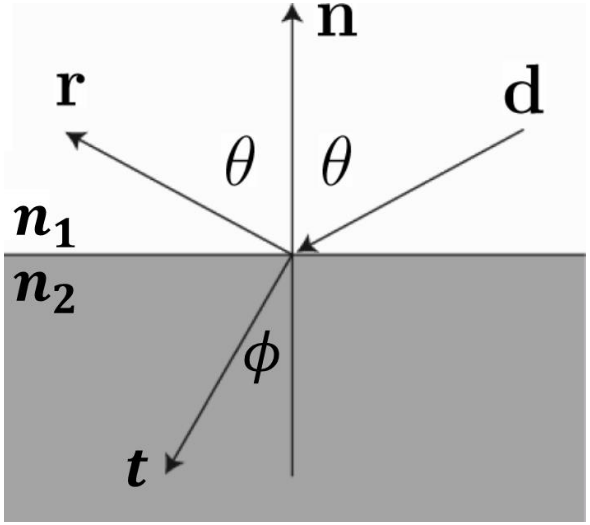
Fresnel Equations



- **Reflection of light is polarized**
  - *Fresnel equations* describe how much light is reflected and transmitted.
  - Reflectance for p- and s-polarized lights R_p and R_s are can be derived as:

$$R_p = \left(\frac{n_2 \cos\theta - n_1 \cos\phi}{n_2 \cos\theta + n_1 \cos\phi}\right)^2$$

$$R_s = \left(\frac{n_1 \cos\theta - n_2 \cos\phi}{n_1 \cos\theta + n_2 \cos\phi}\right)^2$$



  - The effective reflectance for natural lights can be the average of both:

$$R_{\text{effective}} = \frac{1}{2}\left(R_p + R_s\right)$$

$$T_{\text{effective}} = 1 - R_{\text{effective}}$$

Fresnel Equations

- **Schlick approximation**
  - The full equations are often too costly to evaluate, Schlick proposed the approximation below, using the reflectance at normal incidence.
  - Given n_2 and n_1, the reflectance at normal incidence (i.e., $\theta = \phi = 0$) is:

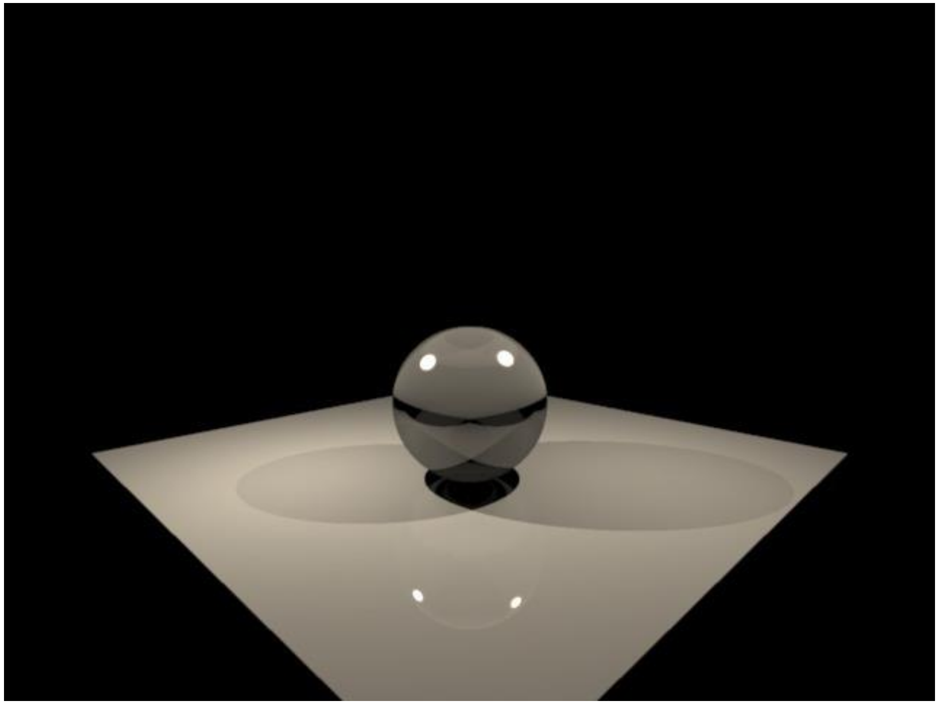
$$R_0 = \left(\frac{n_2 - n_1}{n_2 + n_1}\right)^2$$

  - Then, a reflectance at angle θ can be approximated as:

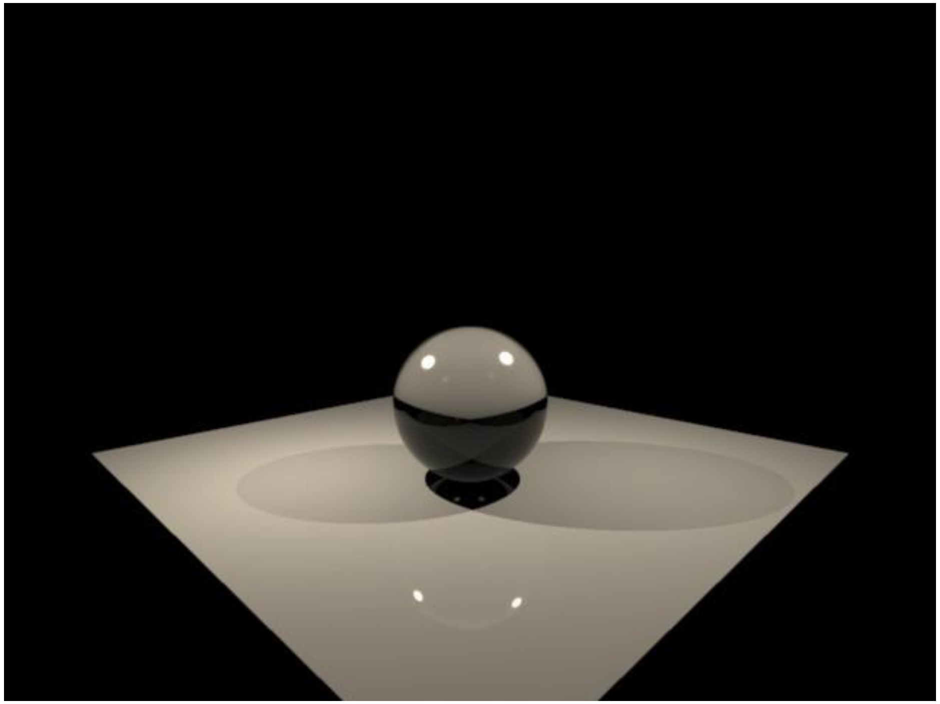
$$R(\theta) = R_0 + (1 - R_0)(1 - \cos\theta)^5$$

Fresnel Equations

- **Examples**



Without Fresnel reflection



With Fresnel reflection