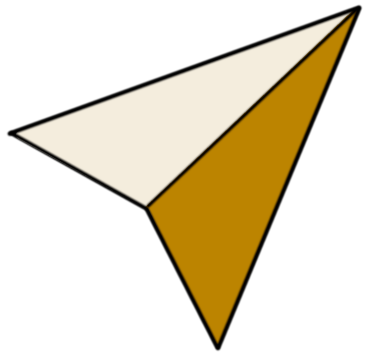


COMP4610/COMP6461

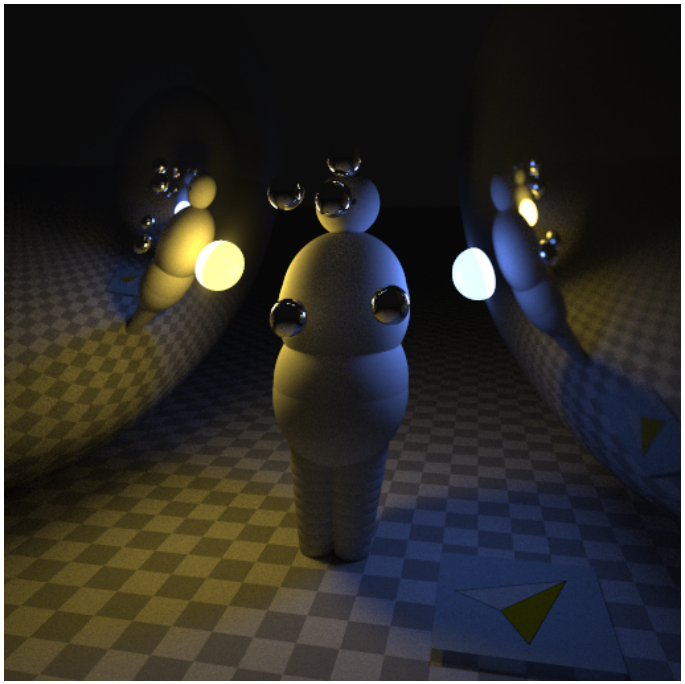
Week 10 - Advanced Lighting and Ray Tracing

<Print version>

Admin

Lab-5

- This lab involves writing a **ray-tracer**.
- You will need to implement the (Blinn) **Phong Lighting model**.



A slightly modified version of a Lab-5 solution.

BRDF

BRDF

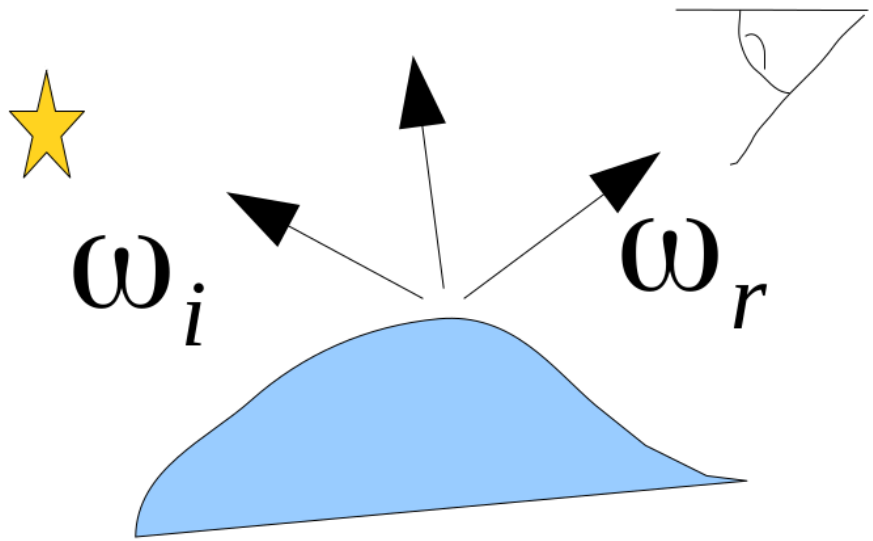
The bidirectional reflectance distribution function (BRDF) characterizes the amount of light reflected on a surface as the lights incident direction and observation direction changes. The function is often denoted:

$$f_r(\omega_i, \omega_f) \tag{1}$$

where ω_i is the unit length vector that points to the light, and ω_r is the unit vector that points to the observer (with respect to the surface normal).

BRDF

The BRDF is a ratio of the reflected radiance to the incident irradiance and has units sr^{-1} (where sr is steradians or the solid angle).



BRDF is a property of a (real) materials surface that can be measured, and then modeled.

Better Material Modeling

As we model a surface of a material the diffuse and specular aspects can be separated and modeled using different approaches. The colour can be additively combined.

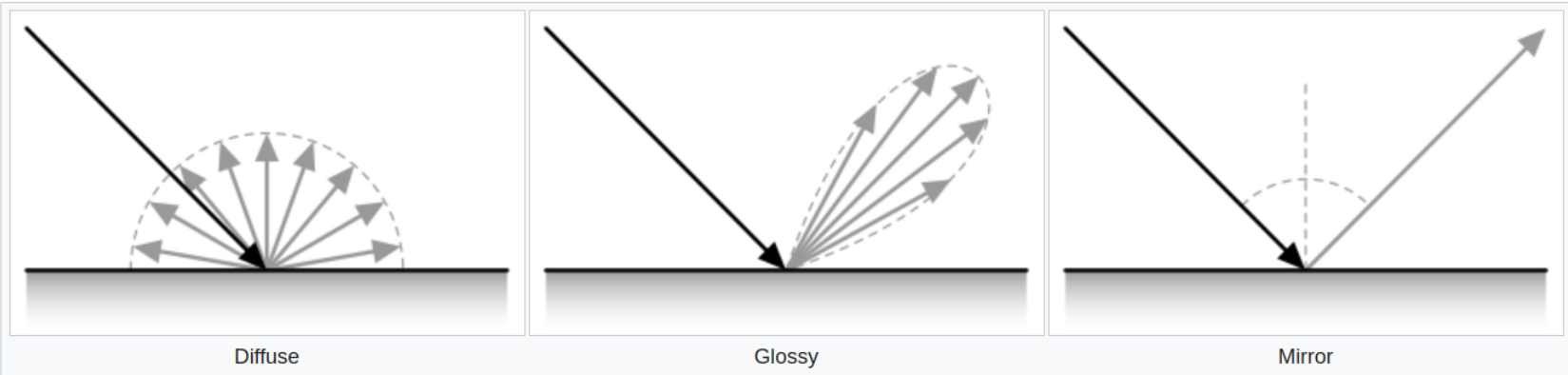
The Blinn-Phong approach is a simple and a good starting point for modeling surface material, however, it is somewhat limited and does not model some surfaces well. (e.g. with metal surfaces the amount of reflection is dependent on the angle of the light source, or object like the moon or a tennis ball their intensity is more uniform rather than be Lambertian in terms of the diffuse aspects)

One could empirically measure the BRDF of a material and then use this to model the material.

There as also more complex aspects that may need to be considered such as: subsurface scattering (skin), bumpy material, scratched surfaces (stainless steel), wave properties (surface of a CD).

BRDF

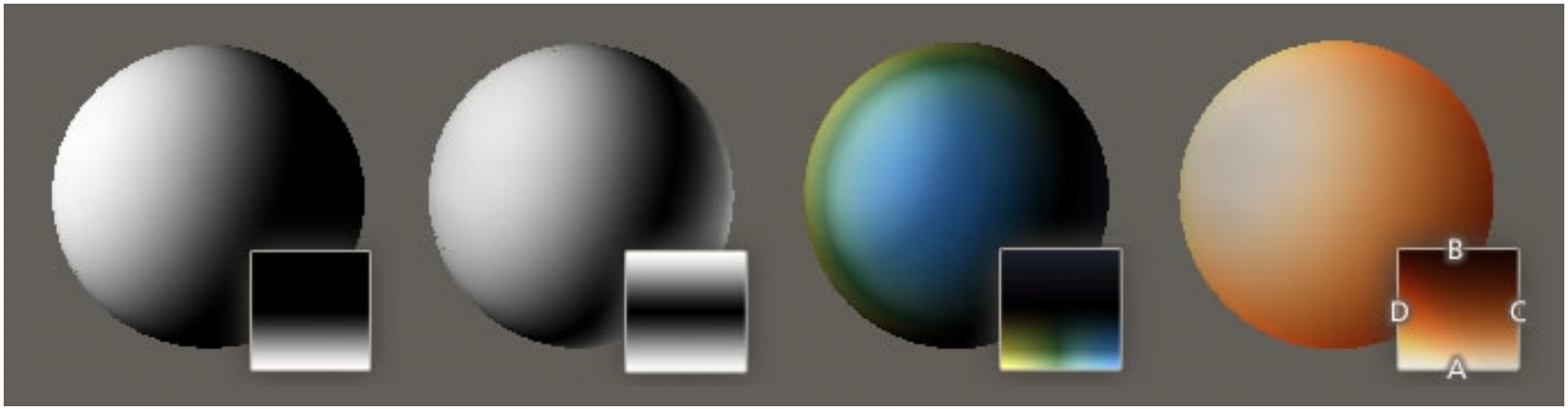
Many the of the lighting models we have studies can be represented by BRDF functions.



Credit: https://en.wikipedia.org/wiki/Bidirectional_reflectance_distribution_function

BRDF Maps

If we consider only the angle of incidence, and the angle of reflection, we can use a 2D map to store a BRDF function.



Credit: http://wiki.polycount.com/wiki/BDRF_map

Plausiable BRDF

There are some properties that a BRDF must have to make it physically plausible. For example it must obey the following rules:

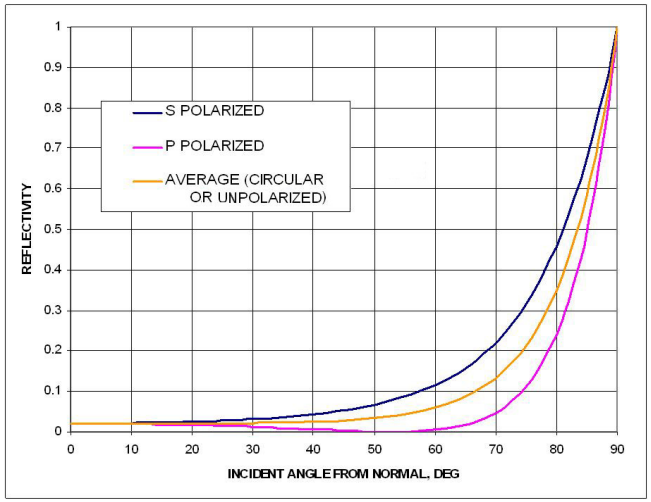
- Positivity. $f_r(\omega_r, \omega_i) \geq 0$
- Energy conserving. $\int_\Omega f_r(\omega_r, \omega_i) \cos\theta_r d\omega_r \leq 1 (\forall \omega_i)$
- Helmholtz reciprocity: $f_r(\omega_r, \omega_i) = f_r(\omega_i, \omega_r)$

Of cause, you can always create BRDF functions that do not follow these rules, they just would not be able to exist in reality.

Fresnel Effects

Surfaces, such as metal, water, glass, etc., have Fresnel effects at **glazing angles**. This is where the amount of light reflected increases (in a non-linear fashion) as the angle of the incident light increases.

If you graph the amount of light reflect at different angles you could use these graphs to better model materials. Blinn-Phong approach does not capture this effect, instead to model this we use the Cook-Torrance model.



Source https://en.wikipedia.org/wiki/Reflectance

Rough Surfaces

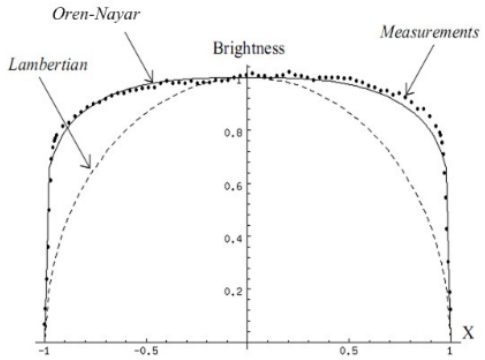
The **Lambertian model** makes an assumption that the intensity of a surface is the same from all viewing angles. Although a good approximation, particularly for very smooth surfaces, this is generally not the case for the diffuse aspects of rough surfaces.

So for rough surfaces, such as the moon or a tennis ball, the **Oren-Nayar** Model will produce more realistic images.

Real Image

Lambertian Model

Oren-Nayar Model

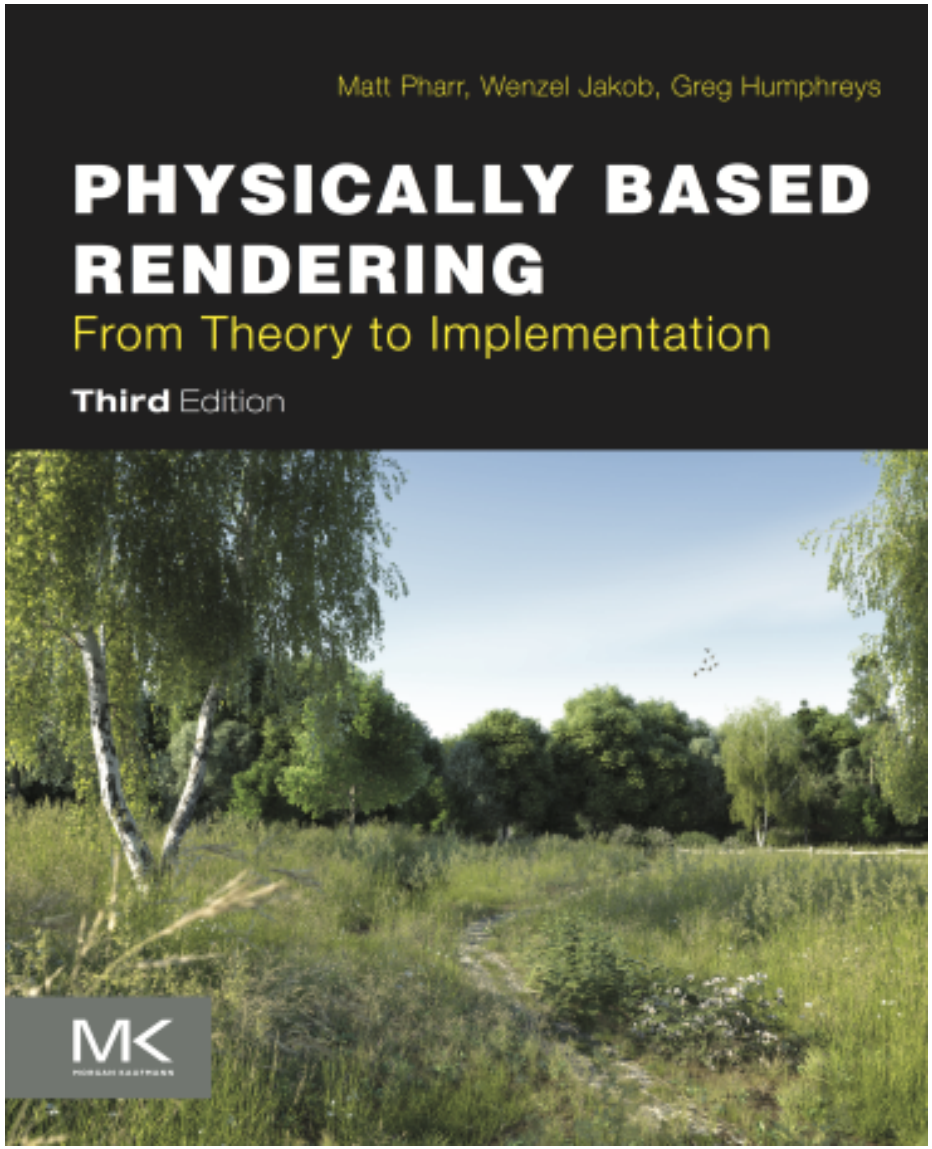


Credit https://en.wikipedia.org/wiki/Oren%E2%80%93Nayar_reflectance_model

Physical Based Rendering (PBR)

- PBR is an attempt to simulate real lighting using the principles of physics. This has only become popular recently (last 10-years) due to the computation required.
- Ideally we no longer want to distinguish between diffuse non-reflective light and specular reflective light. Everything is reflective... it's just a matter of how much.
- The end result is 'one shader to rule them all' that produces quite good results and can take as input real-world measurements of surface materials.
- The downside is that it is quite complicated (I've seen PBR shaders that are 100's of lines of code).

PBR Book



The ANU library has (digital) copies of this book.

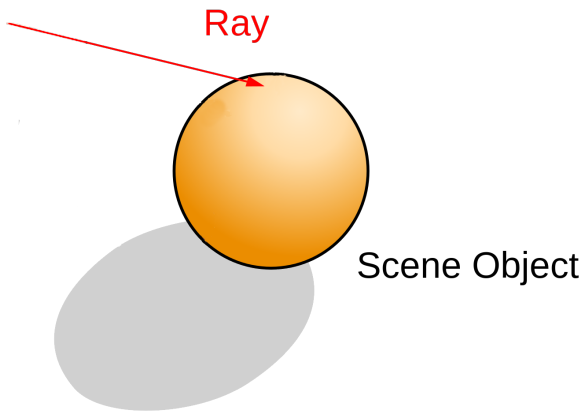
Ray Tracing

Overview

- Ray Tracing is a very old idea.
- Ray Tracing is often easier to implement than rasterized 3D.
- The basic idea is calculate lighting by tracing light rays, rather than a 'bag of tricks'.
- Used in high end rendering engines (e.g. Octane)
- The downside is it's (very) slow.

Ray Casting

Ray Casting - Shoot a ray through the scene and return the first object hit. This can be useful for non-graphical applications as well (e.g. collisions).

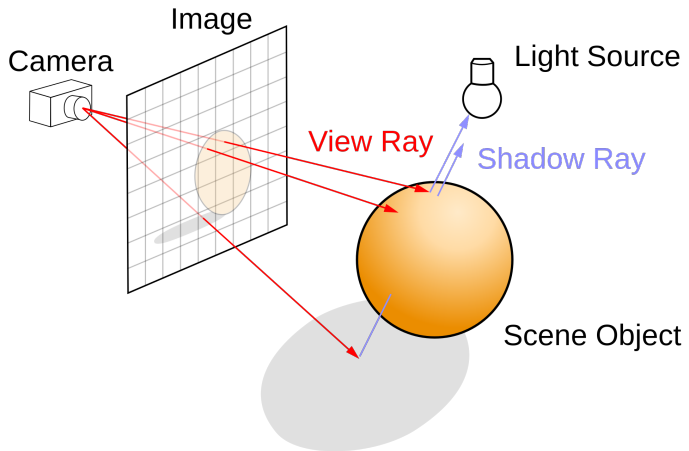


Casting a ray through a scene.
Credit https://en.wikipedia.org/wiki/Ray_tracing_(graphics)

Ray Tracing

Ray Tracing - Shoot a ray of light from the eye, through a pixel's center. When it hits a surface cast the following rays...

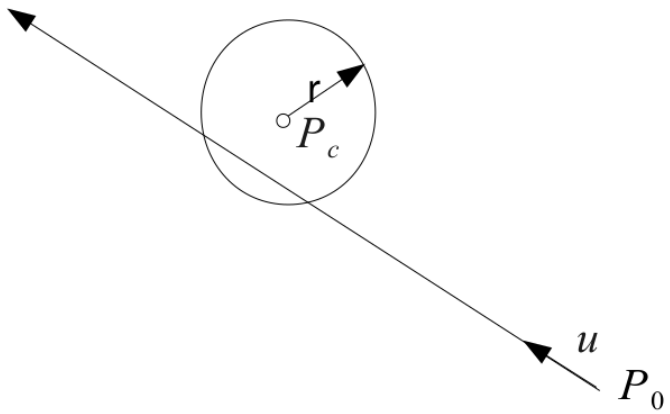
- A single ray to each light source, to test if patch is in shadow or not. If not calculate lighting according to a reflection model.
- (if needed) A single ray *refracted* through the object to for transparent surfaces (e.g. water).
- (if needed) A single ray *reflected* off the surface for shiny surfaces.



Ray Tracing. https://en.wikipedia.org/wiki/Ray_tracing_(graphics)

Ray-Sphere Intersection

Spheres are very simple to trace against (simpler than triangles or boxes). And are therefore often used as bounds for more complex objects.



$$|P - P_c| = r \quad \leftarrow \text{sphere}$$

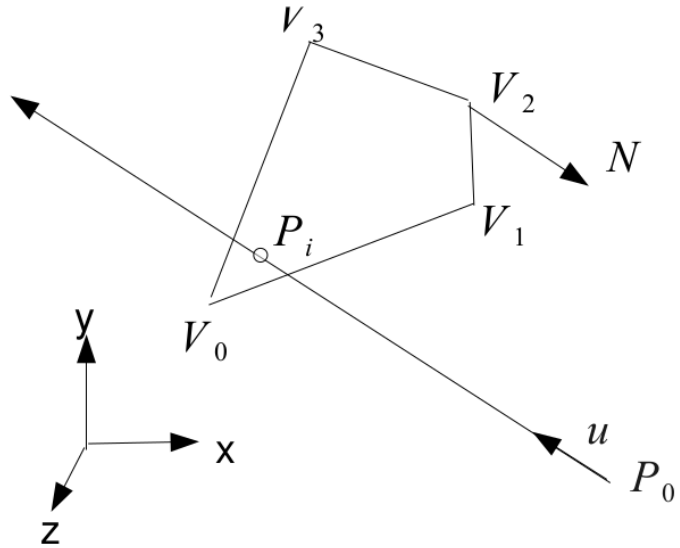
$$P = P_o + s\,\boldsymbol{u} \quad \leftarrow \text{ray}$$

Let $\Delta P = P_o - P_c$

$$s = -u.\Delta P \pm \sqrt{(u.\Delta P)^2 - |\Delta P|^2 + r^2}$$

Ray-Polygon Intersection

Rays are first intersected with the polygon's plane. Then we just need to check if the point found is inside or outside the polygon.



$P = Po + s\,\boldsymbol{u}$ ← ray

$N \,.\, P = d$ ← plane

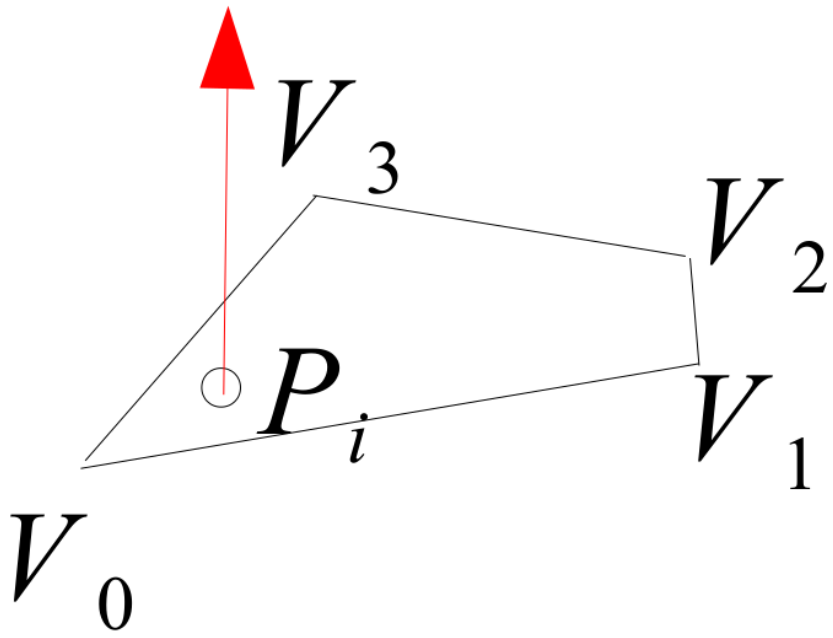
Solving these gives us: $s = \dfrac{d - N \,.\, P_0}{N \,.\, u}$

So the point of intersection is:

$$P_i = Po + \left(\frac{d - N \,.\, P_0}{N \,.\, u}\right)\boldsymbol{u}$$

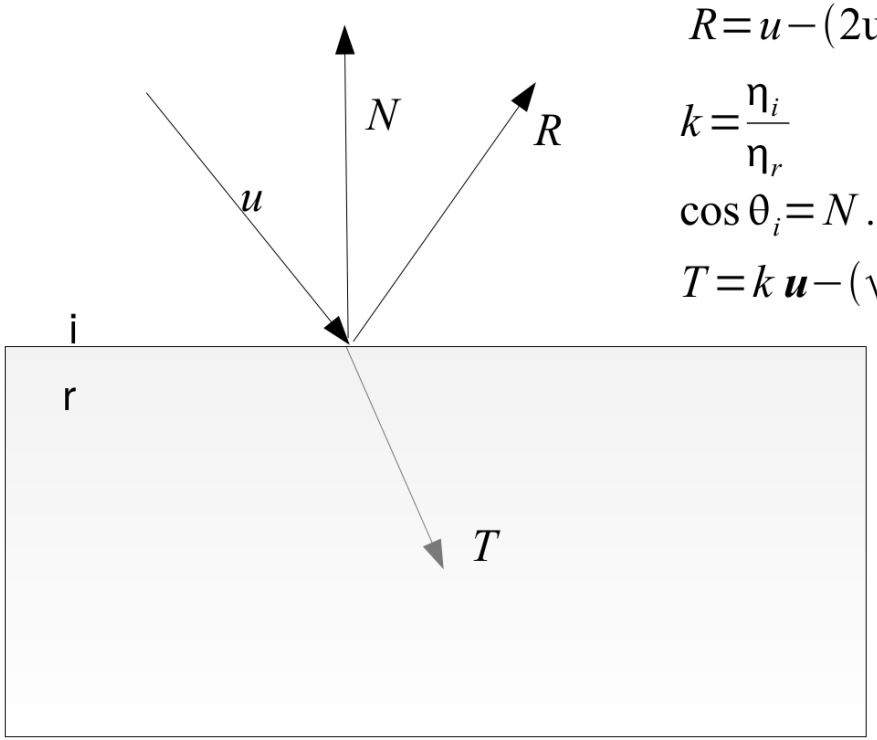
Ray-Polygon Intersection

Vertices of the polygon, along with the intersection point are then projected onto the polygon's plane. Then we perform an inside-outside test.



Reflection and Refraction

Given unit vectors u and N the reflected vector R can be calculated. The transmission direction T also requires the indices of refraction. Fresnel equations can be used to find the intensity of light.



$$R = u - (2\mathrm{u} \,.\, N)\, N$$

$$k = \frac{\eta_i}{\eta_r}$$

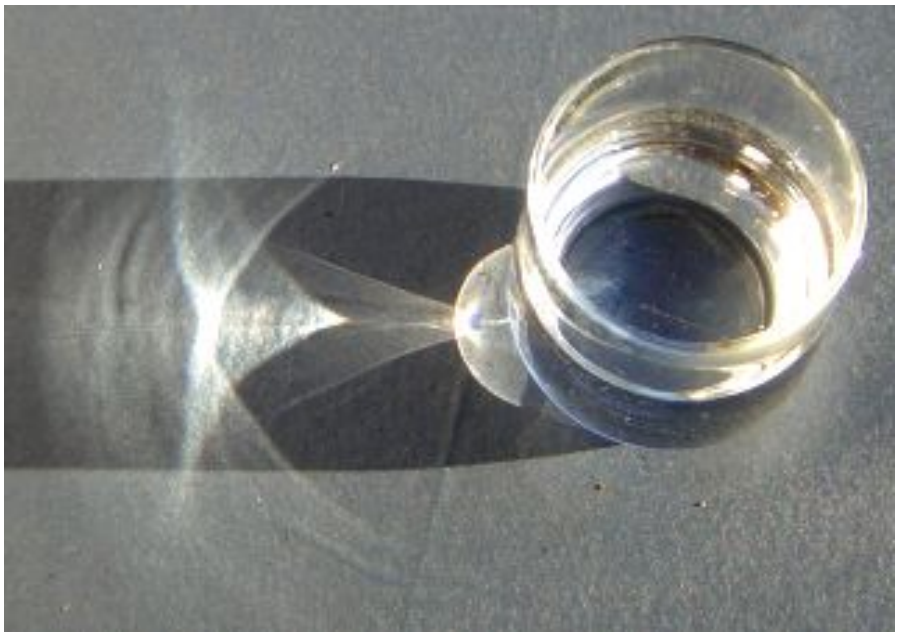
Snell's Law

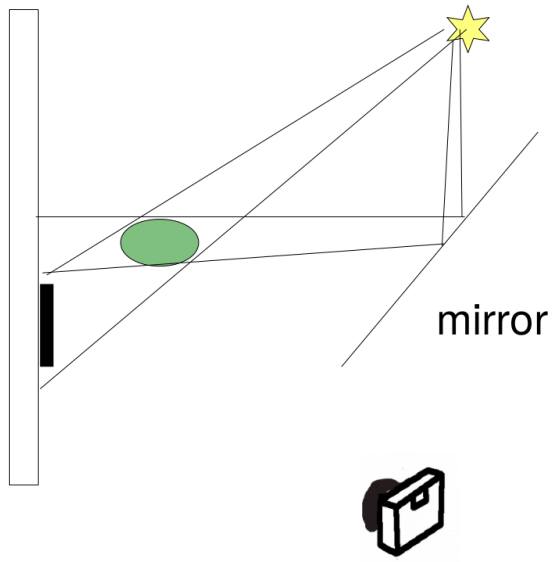
$$\cos\theta_i = N \,.\, (-\boldsymbol{u})$$

$$T = k\,\boldsymbol{u} - (\sqrt{1 - k^2(1 - \cos^2\theta_i)} - k\cos\theta_i)\, N$$

Limitations of Ray Tracing

- Basic ray tracing still uses a simple ambient lighting model.
- (indirect) Shadows from reflected or refracted sources are not visible.
- Caustics not visible.

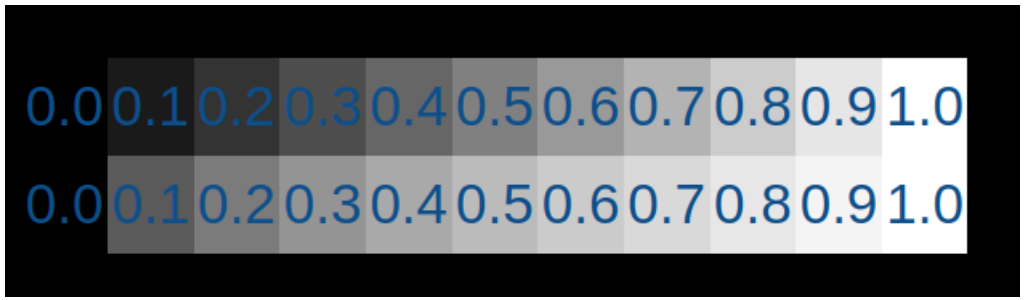




Aside: Gamma Space vs Linear Space

Gamma curves

- Most monitors display color using **sRGB**.
- This involves a complex relationship between the RGB values and the actual luminance output by the monitor.
- The mapping from RGB values to luminance is not **linear**. Therefore if you take two colors and average them you get a new colour that is *not* a blend of the two.
- Instead you need to 'undo' the gamma transform, apply your operation in linear space, then 'redo' the gamma transform.



Top, intensity non linear. **Bottom**, intensity is linear.

Example

Consider a two values, stored in gamma space with $\gamma = 2.0$

$$l_1 = 0.5, l_2 = 1.0$$

If we add them together naively we have.

$$l_{\text{gamma}} = 0.5 + 1.0 = 1.5$$

Which would produce a luminance $1.5^2 = 2.25$, however what we should do is

$$l_{\text{linear}} = 0.5^2 + 1.0^2 = 1.25 \neq 2.25$$

Typically this step is ignored, and all arithmetic is performed in **gamma space**, rather than **linear space**. In many cases you can get away with this, but it will produces non-realistic results.

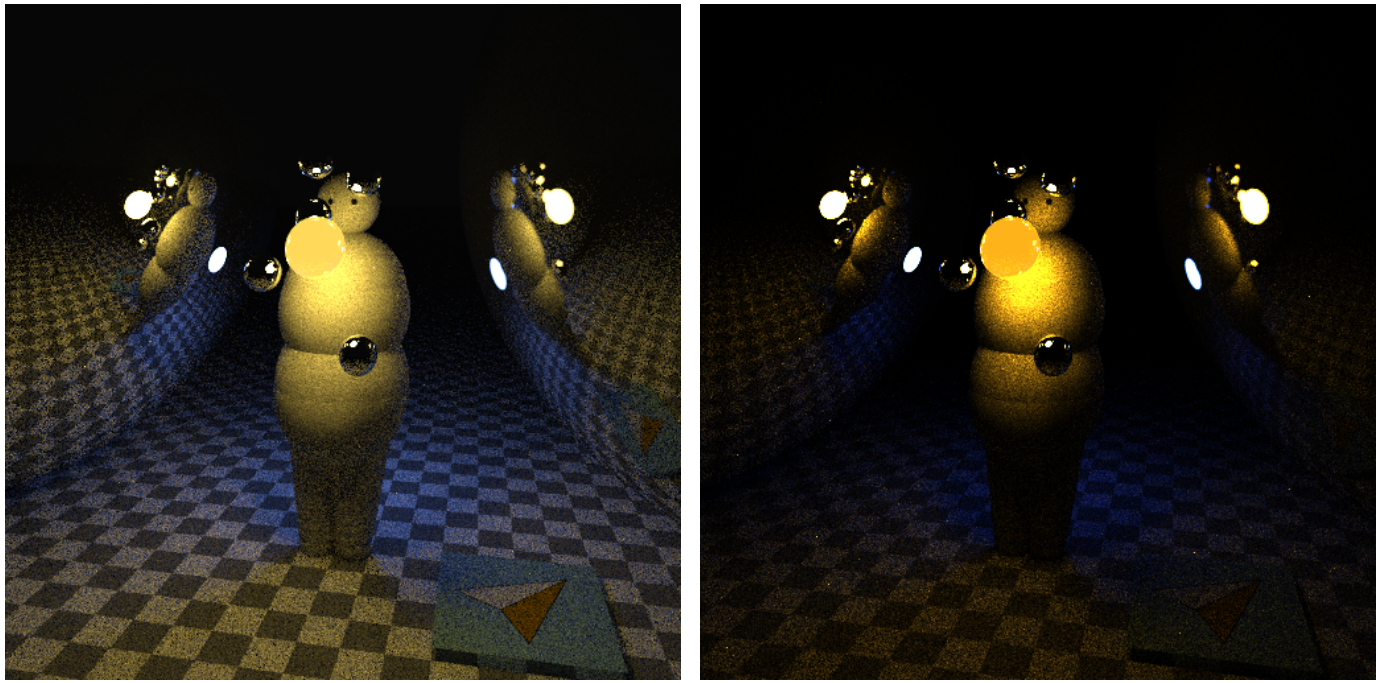
Linear Space Processing

To process operations correctly there are two options. One is to assume everything is **gamma** space, then convert, transform, and convert back again. This can be a bit slow, but is usually how things are done.

The other is to keep everything in linear space, so no transforms are needed, then convert once to gamma space at the end. This method requires converting images to linear space on loading, and also requires 32-bit floating point for intermediate RGB values. For this reason it is not used much.

Raytracer Example

In my solution to lab-5, I perform all operations in linear space (this is not required), then transform them once at the end. This gives much softer lighting.



Left: Linear space lighting, **Right**: Lighting (incorrectly) performed in gamma space.