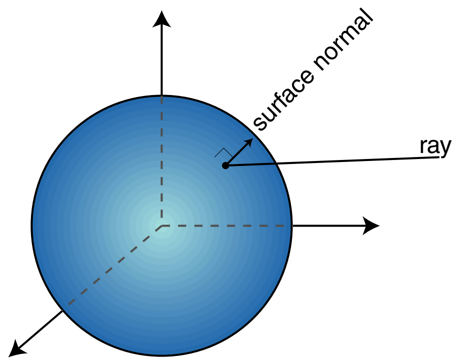
Raytracer project

Requirements:

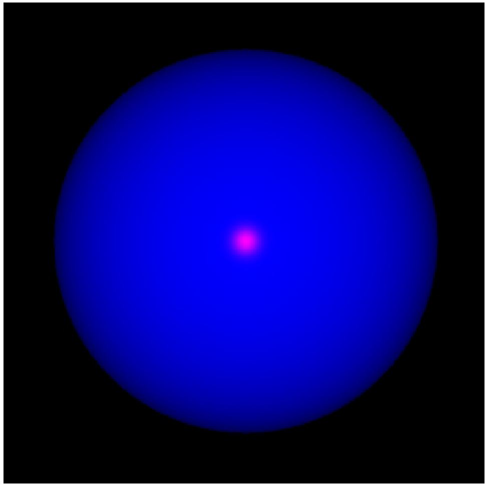
- Sphere intersection
- Phong model
- Light sources/shadows
- Reflection
- Refraction

Requirements: Sphere intersections



- You can assume that the sphere is the unit sphere centered at the origin; we take care of all the other transforms.
- All the excitement happens in `Sphere::intersectLocal`, which takes in a ray and returns *true* if an intersection is found. If an intersection exists, information about it (including the t value and the normal vector) are returned through the `isect` object.
- Don't forget about `RAY_EPSILON`!

Requirements: Phong shading

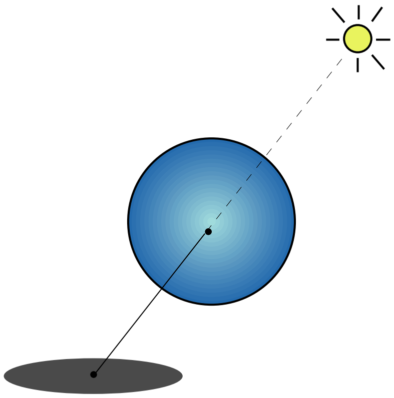


- Include all the terms in the Phong shading equation,

$$I = k_a + k_a I_a + \sum_j f_{\text{atten}}(d_j) I_{l_j} \left[k_d (\mathbf{N} \cdot \mathbf{L}_+ + k_s (\mathbf{V} \cdot \mathbf{R})_+^{n_s} \right]$$

- Everything you need to modify is in `Material::shade`. Remember to iterate over all lights! Look at `Material.h` for what the different material coefficients represent.
- You'll need to multiply the material's specular *exponent* by 128 in order to get the correct results.

Requirements: opaque shadows and lighting

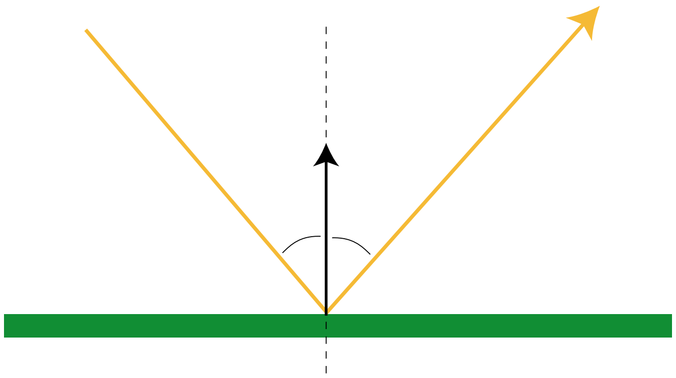


- Lighting and shadows are handled in `light.cpp` and `material.cpp`.
- Point lights should have intensity fall-off using the equation

$$f_{\text{atten}}(d) = \frac{1}{a + bd + cd^2}$$

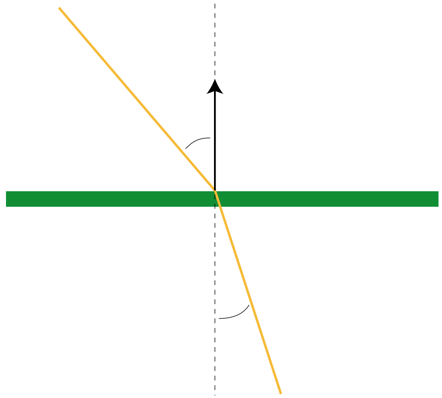
- Tracing shadow rays back to the light. Remember that point lights have a position in space!

Requirements: reflection



- You'll be working in `RayTracer.cpp` for both reflection and refraction.
- Make sure you remember to multiply the color returned by recursively tracing reflection rays by the surface's reflective coefficient.

Requirements: refraction



- Use the direction of the surface normal to determine whether the ray is entering or leaving the object.
- Be sure to consult the errata for the Watt book!

Vector math tips

Use the **`prod`** function for pointwise products, that is

$$\texttt{prod}\left(\begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}\right) = \begin{bmatrix} a_1 b_1 \\ a_2 b_2 \\ a_3 b_3 \end{bmatrix}$$