CSC 317

Tutorial: Ray Tracing

Prerequisites - all from previous 2 assignments

src/Plane.cpp,

src/Sphere.cpp,

src/Triangle.cpp,

src/TriangleSoup.cpp,

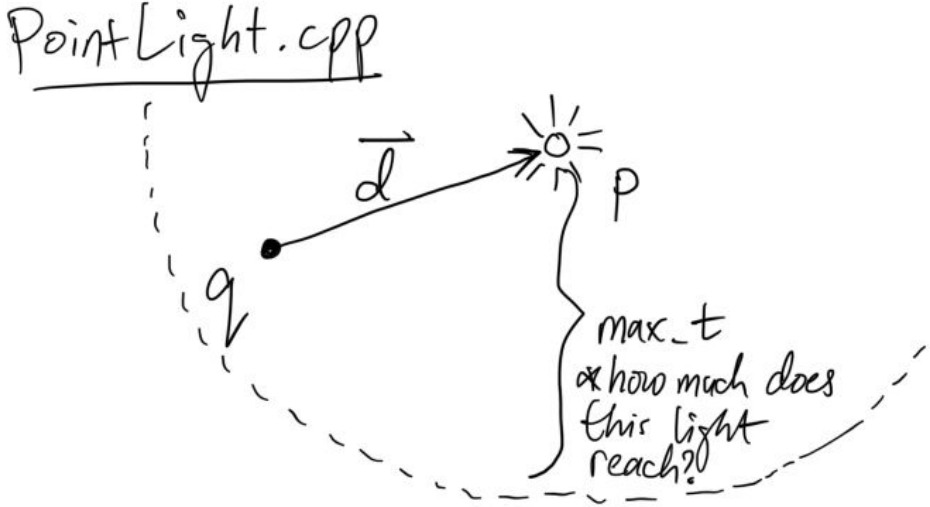
src/first_hit.cpp,

src/viewing_ray.cpp,

src/write_ppm.cpp

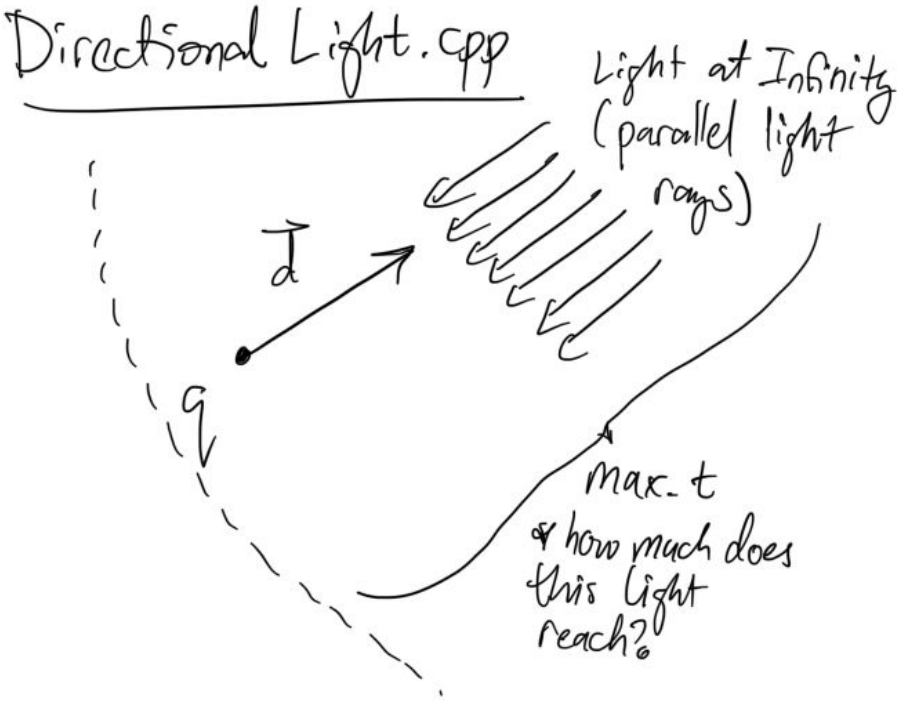
PointLight.cpp

- Output:
  - d: vector direction **from** query point q **towards** this light's location p; unnormalized
  - max_t: maximum (parametric) extent of this point light (ie how far this light can reach); parametric relative to d
- Input:
  - q: query point in space
- Check .h file for what members are defined



DirectionalLight.cpp

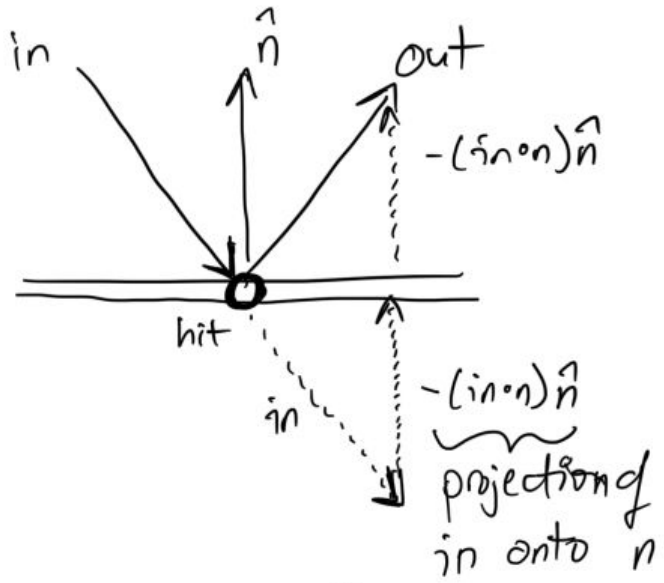
- Output:
  - d: vector direction **from** query point q **towards** this light's location (at infinity); normalized
  - max_t: maximum (parametric) extent of this point light (ie how far this light can reach); parametric relative to d
- Input:
  - q: query point in space
- Check .h file for what members are defined



*Note: #include <limits> has std::numeric_limits<T>::infinity()

reflect.cpp

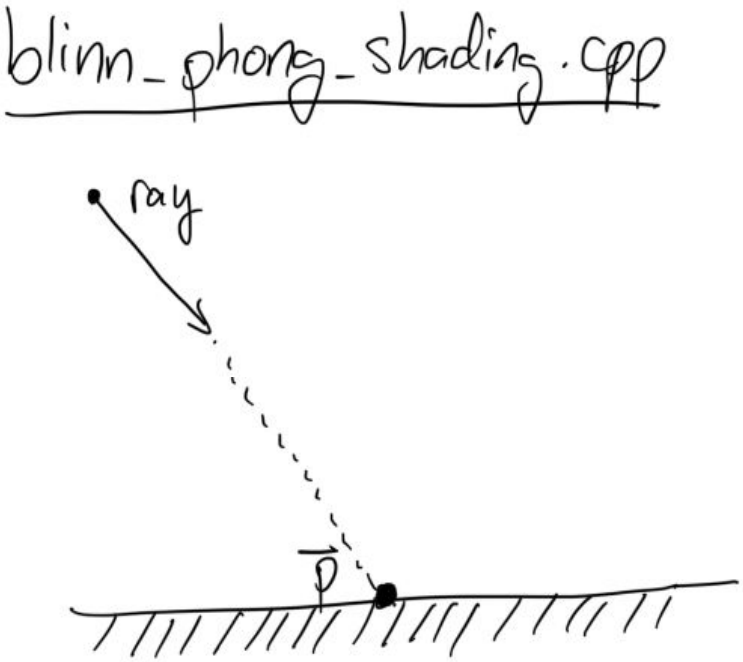
reflect.cpp



$in - 2(in \cdot n)\hat{n} = out$

* DO NOT NORMALIZE
  we want out to be the same length as in

blinn_phong_shading.cpp

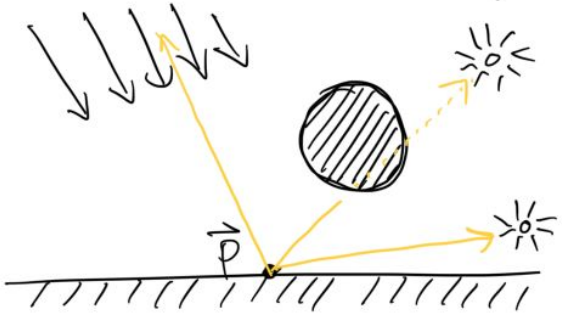


blinn_phong_shading.cpp

First find intersection point p

$$\vec{p} = \overrightarrow{ray.origin} + t * \overrightarrow{ray.direction}$$

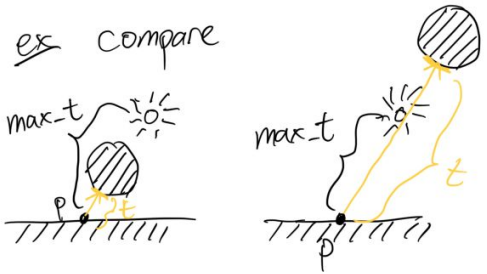
Now, we want to find the contribution from each light



For each light

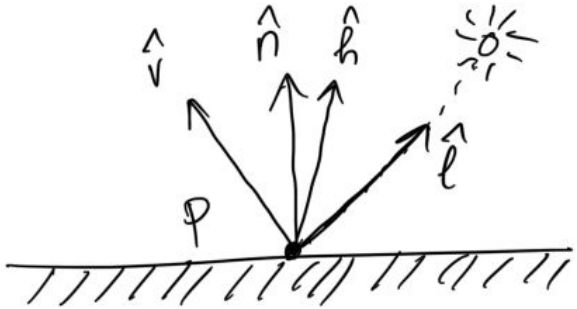
* check if we are in shadow
  - ↳ construct a "shadow ray" from $\vec{p}$ pointing towards the light
  - ↳ if the shadow ray hits another object before it reaches the light (ie use max_t) then we ignore that light.

ex compare



blinn_phong_shading.cpp

If the point p is lit:



$\hat{v}$: view vector
(ie ray.direction normalized)
* note the sign, ray.direction points towards p; blinn-phong usually written with $\hat{v}$ pointing away

$\hat{l}$: light vector
(ie same as shadow ray direction normalized)

$\hat{h}$: halfway vector

$$\frac{\hat{l} + \hat{v}}{\|\hat{l} + \hat{v}\|}$$ ← normalized.

blinn_phong_shading.cpp

$L \mathrel{+}= k_d * \max(0, n^T \ell) * \text{light colour}$

diffuse light

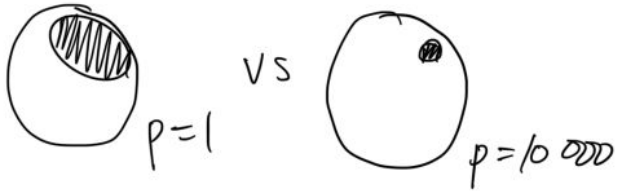
↳ depends on angle between $\hat{n}$ and $\hat{\ell}$

$L \mathrel{+}= k_s * \max(0, n^T h)^p * \text{light colour}$

specular light

p controls object shininess

phong exponent



blinn_phong_shading.cpp

ALSO INCLUDE

L += ka * ambient_factor

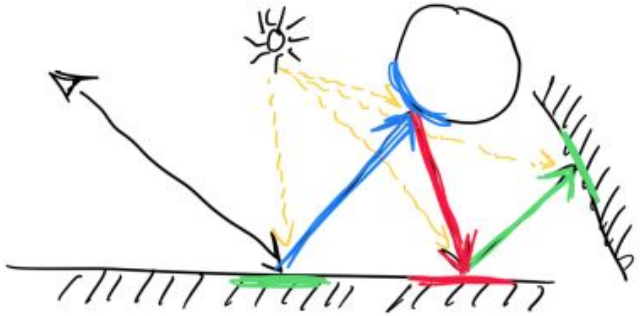
ambient_light (use 0.1)

see .h file

& include only once
(not for every light)
the base colour of the
material if entirely in shadow.

raycolor.cpp

We want to allow for recursive reflections



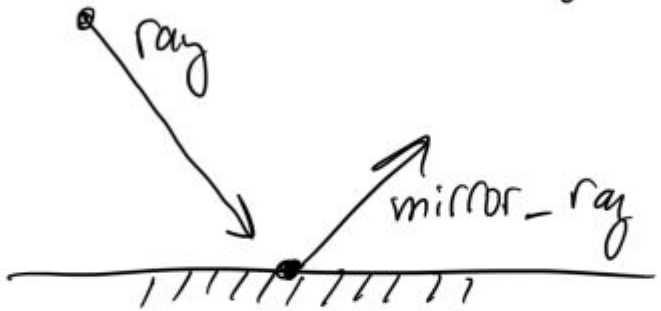
and at each point of intersection compute the light colour

* Use first_hit to look for intersections
  - ↳ if intersection found:
    - ↳ get colour using blinn_phong
    - ↳ make mirror ray
    - ↳ recurse
      - (ie call raycolor then add km * returned_colour to colour)

mirror coefficients

raycolor.cpp

To construct mirror ray:

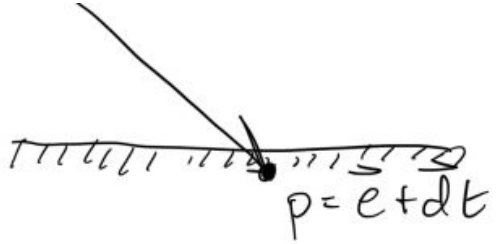


mirror_ray.origin = ray.origin + t* ray.direct
mirror_ray.direction = reflect ()

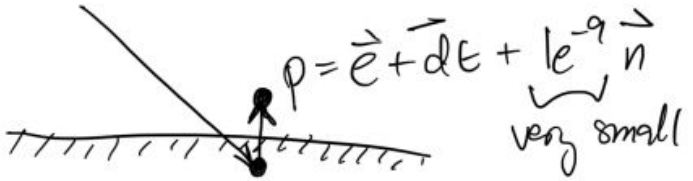
FOR

mirror_ray.origin
shadow_ray.origin

you might need to add a fudge factor since floating point rounding could cause you to be under the surface



Just add a very small surface normal



to guarantee you are outside the surface.