Introduction to Computer Graphics

Practical Session 3 - Lighting

Tizian Zeltner
Realistic Graphics Lab

Outline of today's lecture

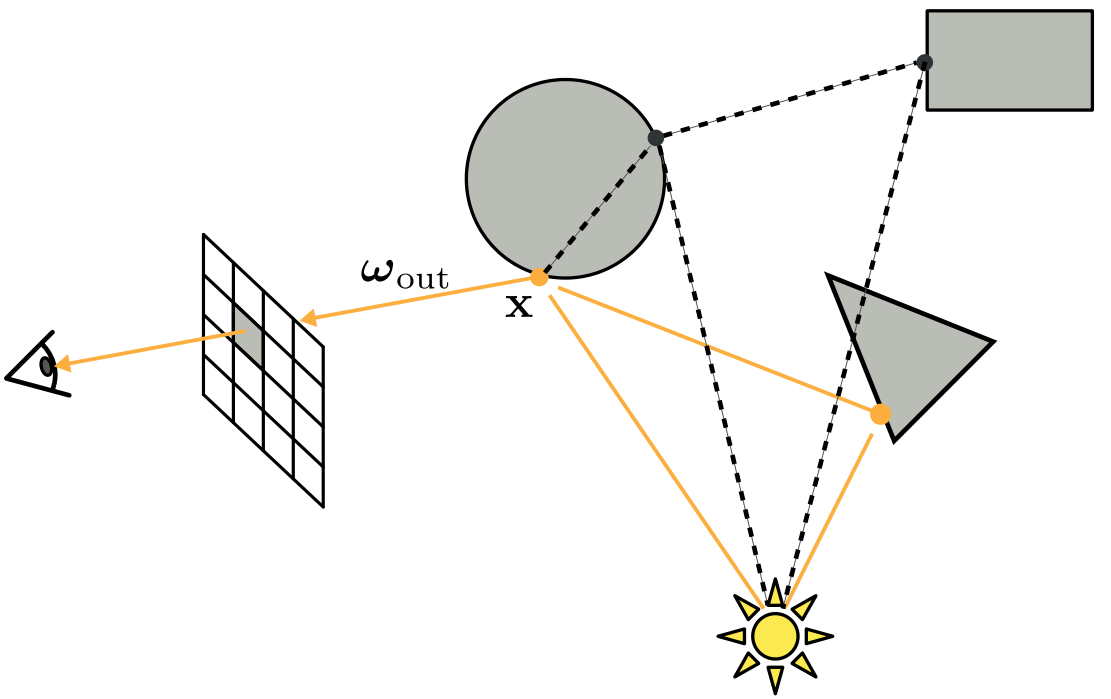
- Assignment 2
- Phong lighting
- Shadows
- Reflections

Relevant components in Framework

- Everything takes place in `Scene.h` / `Scene.cpp`
- Check variables, functions, comments in `Scene.h`
- Check how `render()`, `intersect()`, `read()` are implemented
- Fill in missing code in `lighting()` and `trace()`

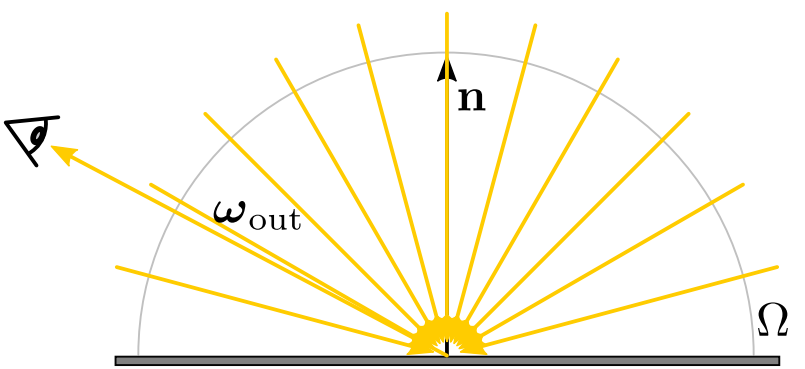
Surface Reflectance

- How much light is leaving point $\mathbf{x}$ in direction ω_{out}?



Surface Reflectance

- How much light is leaving point $\mathbf{x}$ in direction ω_{out}?

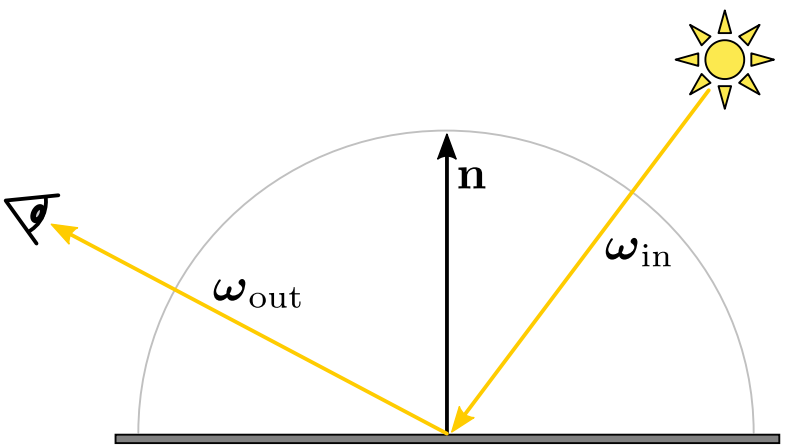


- Collect incoming light L_{in} from all directions $\omega_{\text{in}} \in \Omega$

$$L_{\text{out}}(\omega_{\text{out}}) = \int_{\Omega} f(\omega_{\text{in}}, \omega_{\text{out}})\, L_{\text{in}}(\omega_{\text{in}})\, \cos(\theta_{\text{in}})\, \mathrm{d}\omega_{\text{in}}$$

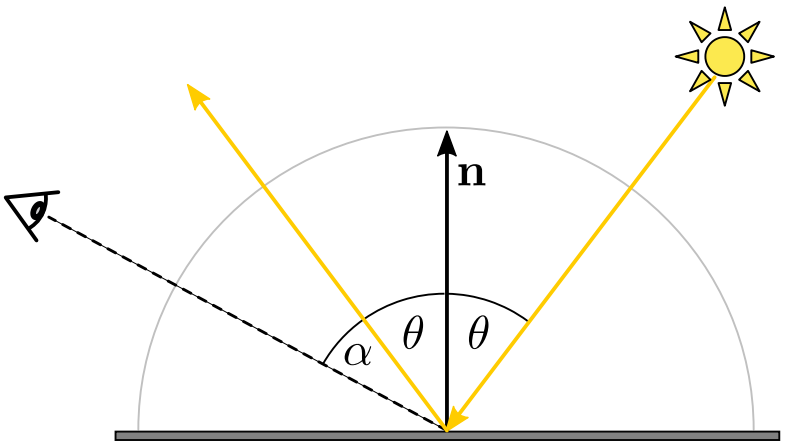
Surface Reflectance

- How much light coming in from direction ω_{in} is reflected out in direction ω_{out}?



- Determined by the object's *BRDF* $f(\omega_{\text{in}}, \omega_{\text{out}})$
  - Bidirectional Reflectance Distribution Function
  - General description of an object's material

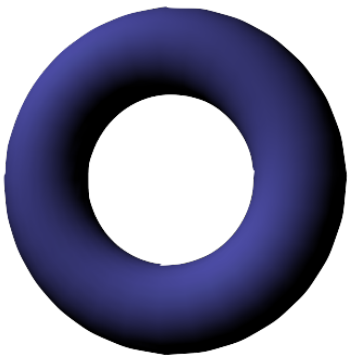
Phong Lighting Model



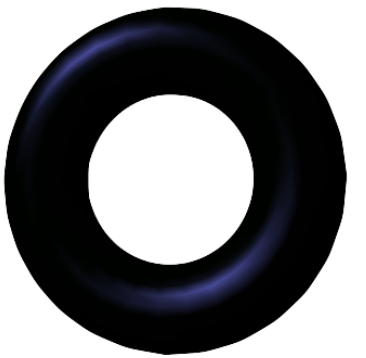
$$I = I_a m_a + I_l \left(m_d \left(\mathrm{n} \cdot \mathrm{l} \right) + m_s (\mathrm{r} \cdot \mathrm{v})^s \right)$$

Phong Lighting Model

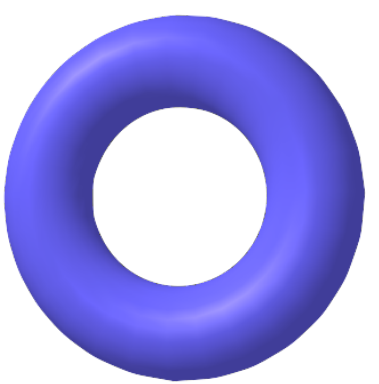
ambient: $I_a m_a$



diffuse: $I_l m_d \left(\mathrm{n} \cdot \mathrm{l}\right)$



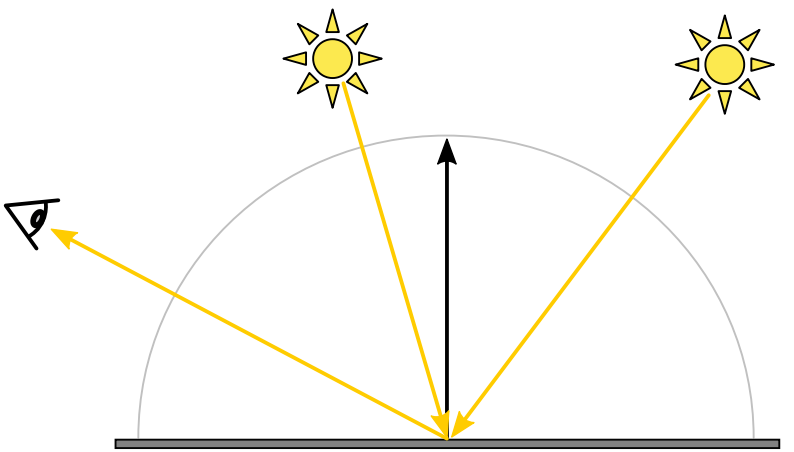
specular: $I_l m_s (\mathrm{r} \cdot \mathrm{v})^s$



$$I = I_a m_a + I_l \left(m_d \left(\mathrm{n} \cdot \mathrm{l}\right) + m_s (\mathrm{r} \cdot \mathrm{v})^s \right)$$

Multiple Light Sources

We assumed linear superposition of light contributions and therefore can simply sum over all light sources



$$I = I_a m_a + \sum_l I_l \left(m_d \left(\mathrm{n} \cdot \mathrm{l}_l \right) + m_s (\mathrm{r}_l \cdot \mathrm{v})^s \right)$$

In the framework...

$$I = I_a m_a + I_l \left(m_d \left(\mathrm{n} \cdot \mathrm{l} \right) + m_s (\mathrm{r} \cdot \mathrm{v})^s \right)$$

- Ambient light intensity of scene is stored in `ambience`
- Each material has properties `ambient`, `diffuse`, `specular`, and `shininess`. See `material.h`.
- Each light has property `color` that stores its intensity, and `position` that stores location in the scene. See `Light.h`.

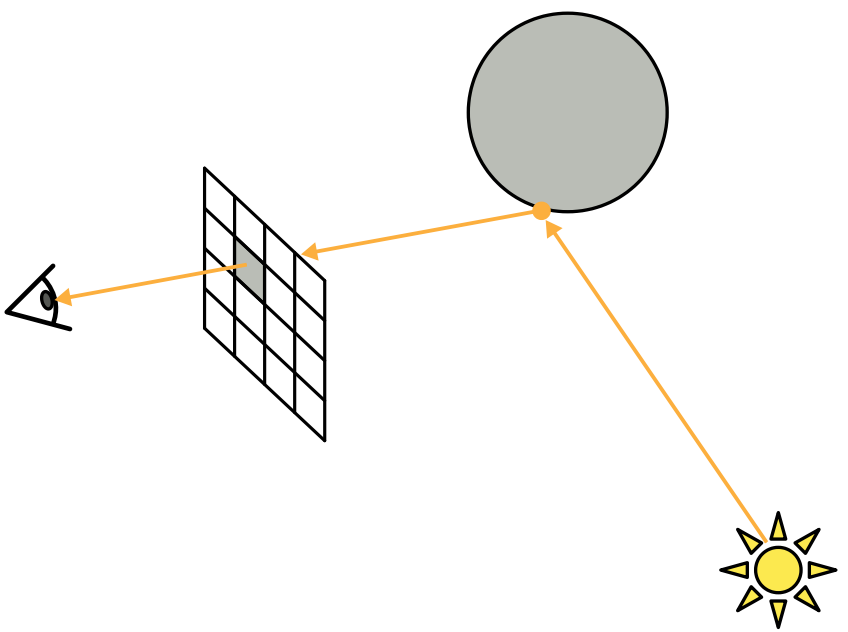
TODO 1.1

In function `lighting()`:

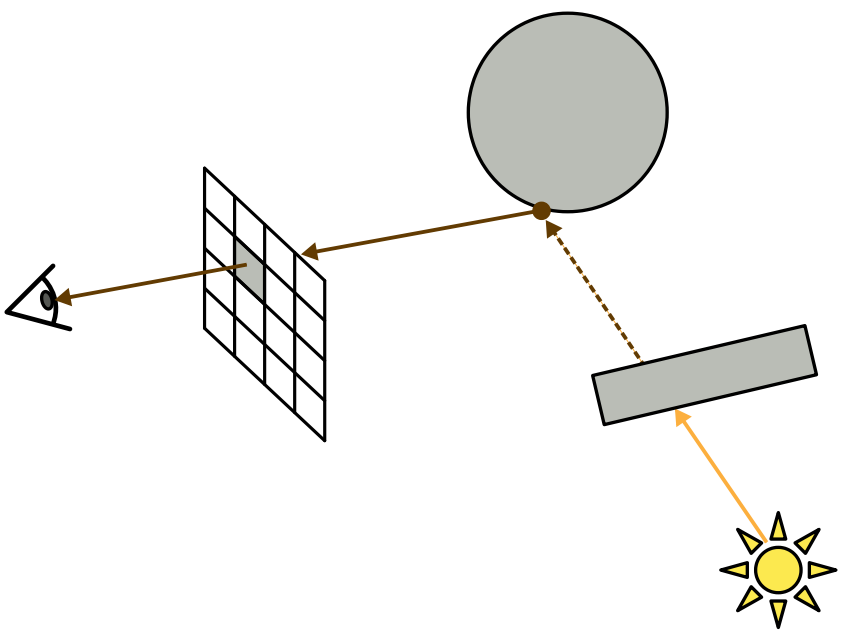
- Compute ambient term and store it in `color`
- For each light source
  - Compute diffuse term (and add the contribution to `color`)
  - Compute specular term (and add the contribution to `color`)

Shadows

- Send *shadow ray* from intersection point to light source.
- Discard diffuse and specular contribution if light source is blocked by another object.



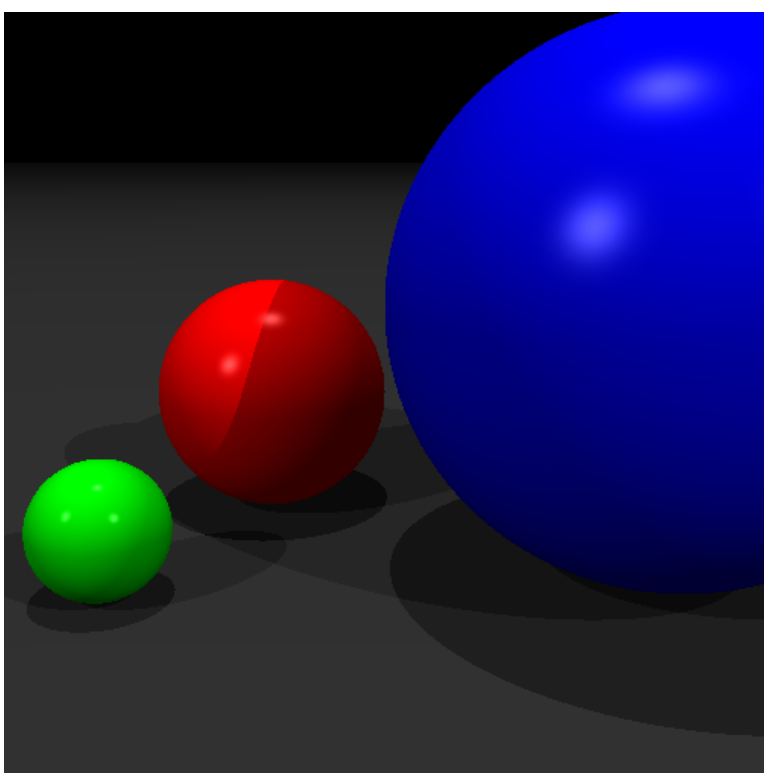
Point in light: ambient + diffuse + specular



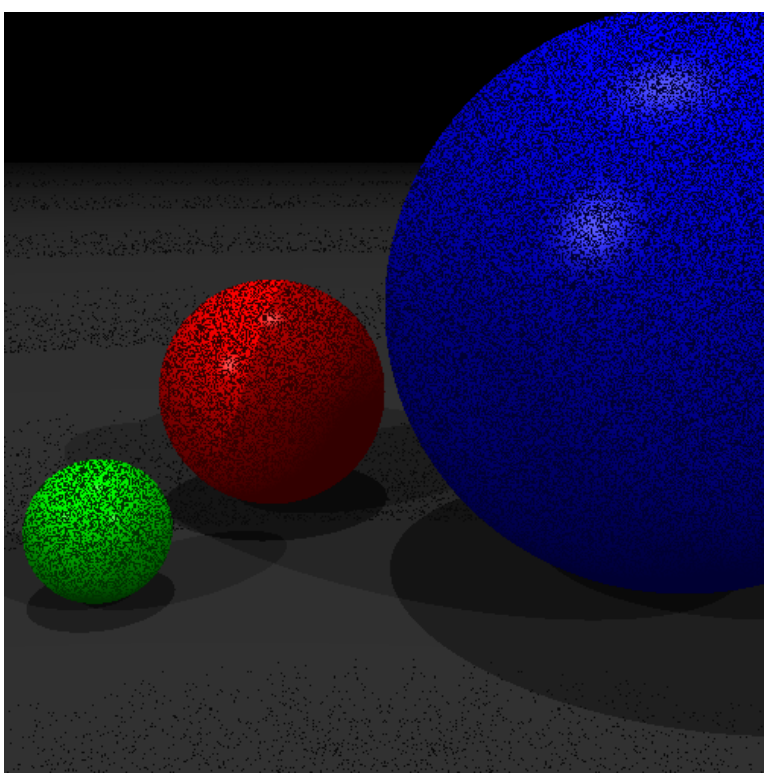
Point In shadow: ambient lighting only

Shadows

- Send *shadow ray* from intersection point to light source.
- Discard diffuse and specular contribution if light source is blocked by another object.



we want this

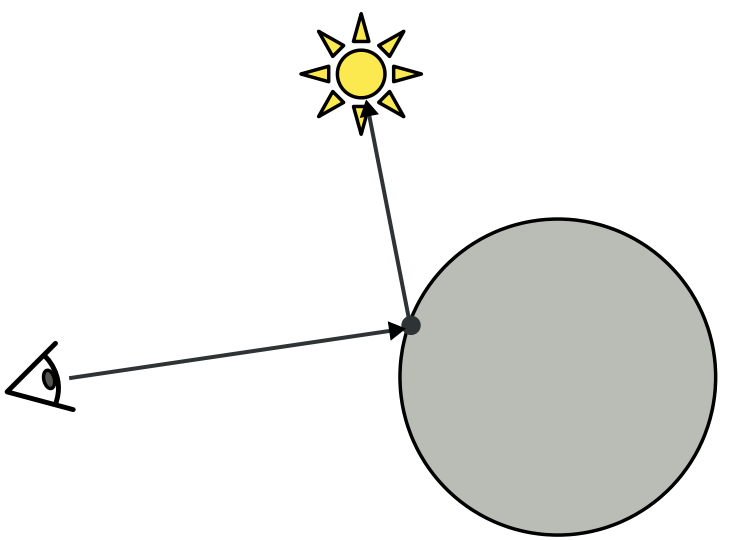


but we get this

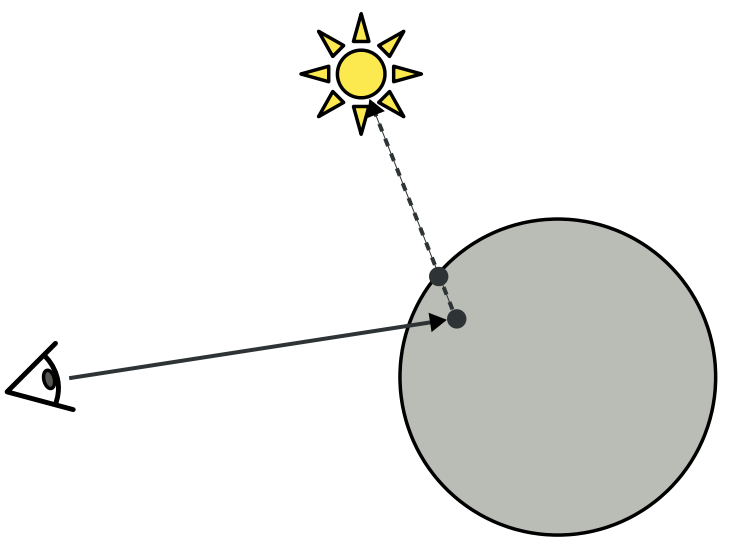
Why??

Shadows

- Floating point errors might lead to erroneous self-shadowing (*shadow acne*).
  - Solution 1: Discard secondary intersection points that are too close.
    - in our implementation: slightly displace ray origin along new ray direction.
  - Solution 2: Offset primary intersection point along surface normal.



exact computation



precision problems

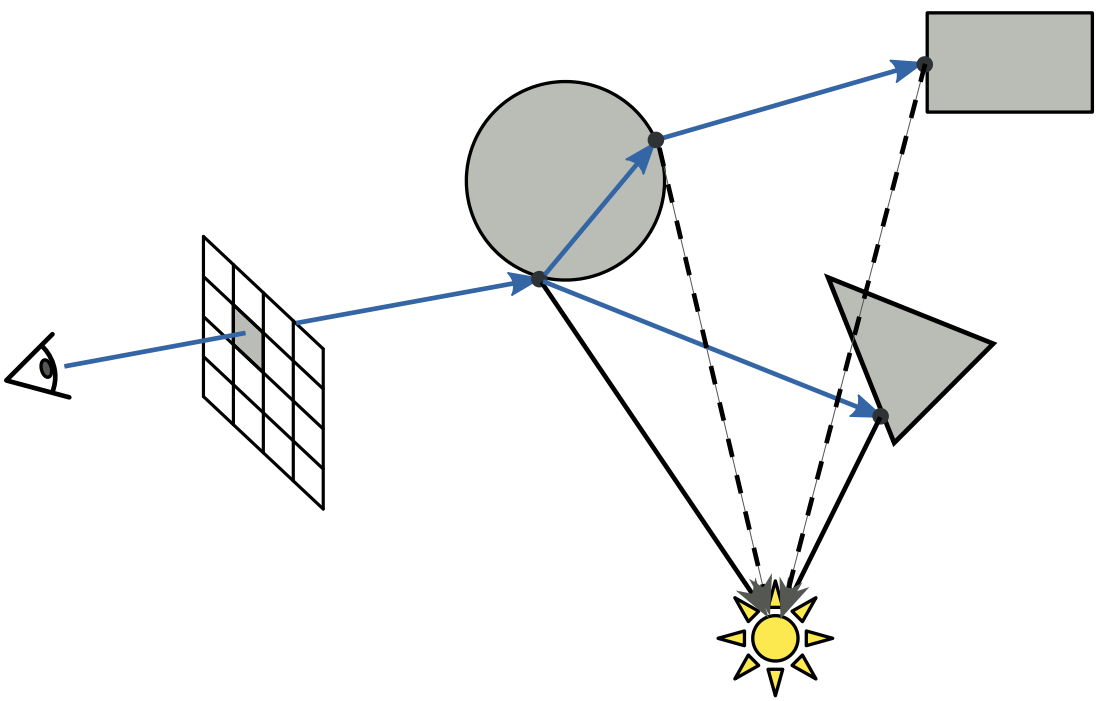
TODO 1.2

In function `lighting()`:

- Send out a *shadow ray* from intersection point to light source
  - Remember to use small displacement
- Discard diffuse and specular contribution if light source is blocked by another object

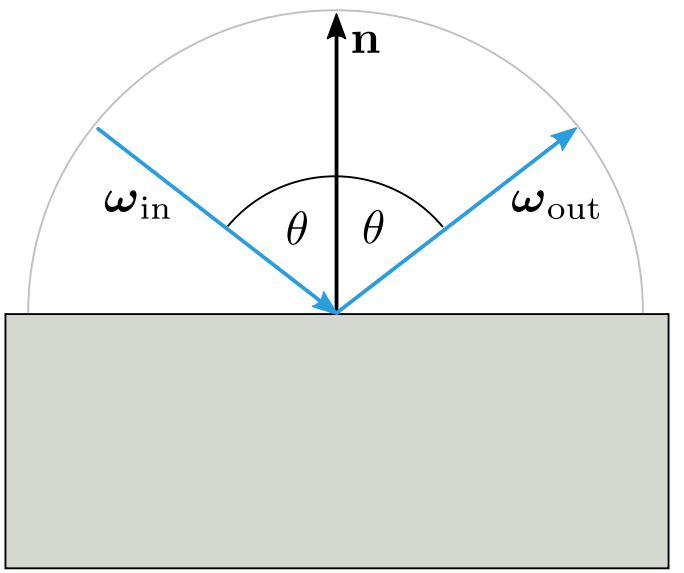
Recursive Ray Tracing

- At each intersection point, reflect and/or refract incoming viewing ray at surface normal, and trace child rays *recursively*.



Reflections

- At each intersection point, reflect and/or refract incoming viewing ray at surface normal, and trace child rays *recursively*.



$$\omega_{\text{out}} = \left(\mathrm{I} - 2\mathrm{n}\mathrm{n}^{\mathsf{T}}\right)\omega_{\text{in}}$$

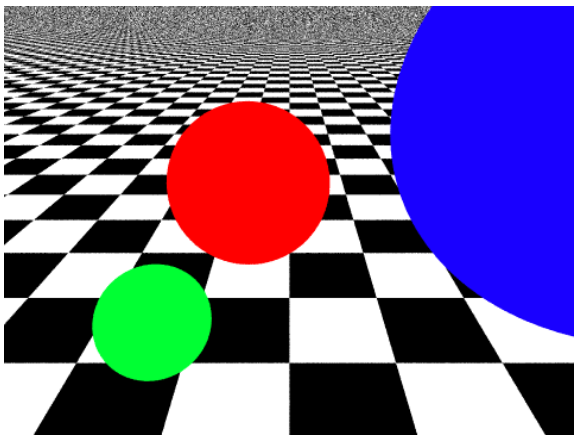
Hint: Remember precision issue. You need to offset ray origin!

TODO 2

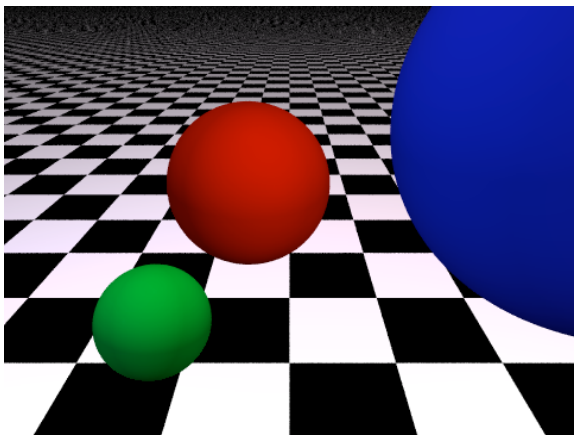
In function `trace()`:

- Compute *reflected ray*
  - *Hint*: `reflect()` in `vec3.h`
- Compute final return color using linear interpolation
  - $\text{color} = (1 - \alpha) \cdot \text{color} + \alpha \cdot \text{reflected\_color}$
  - reflected_color computed with recursive call
  - α is the `mirror` property in `Material.h`
- Use `max_depth` as stopping criteria

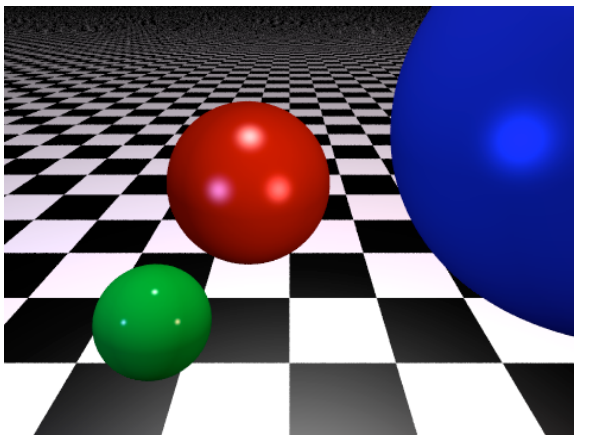
Lighting Computations



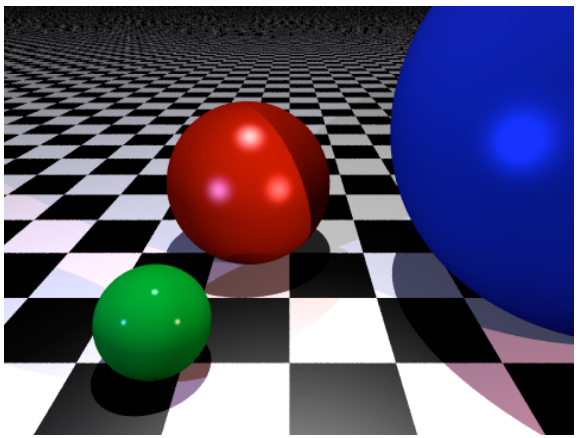
ambient



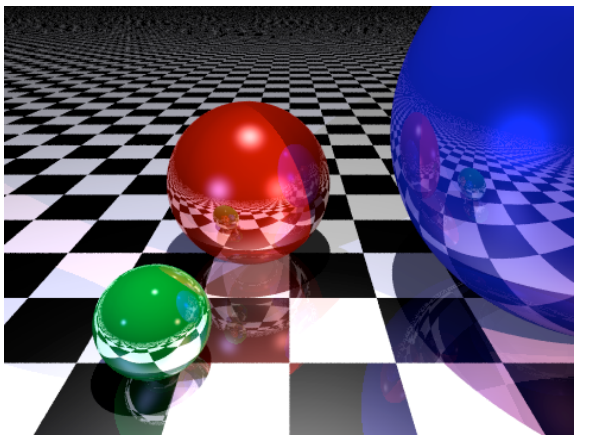
+diffuse



+specular



+shadows



+reflections

Questions?

- Ask them now or…
  - Post to the Moodle forum (no partial solutions!)
  - Email us at icg19@groupes.epfl.ch