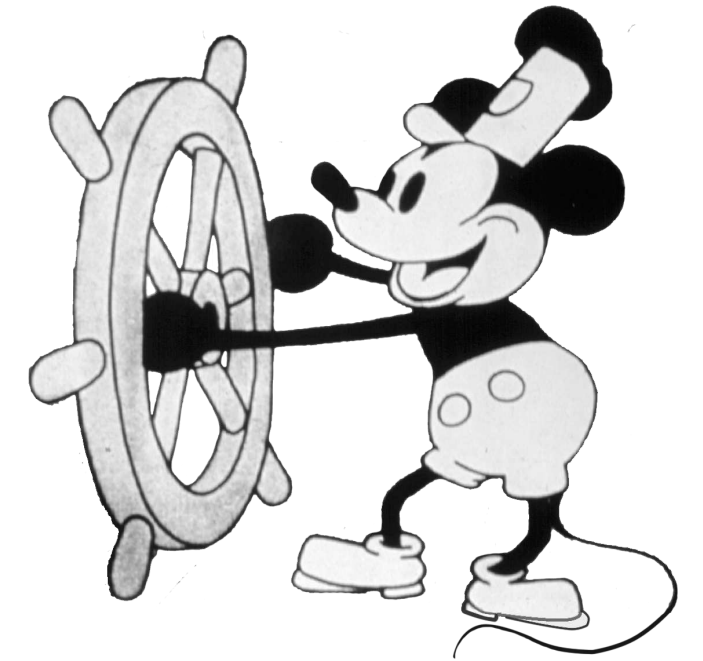




CSCI 461: Computer Graphics

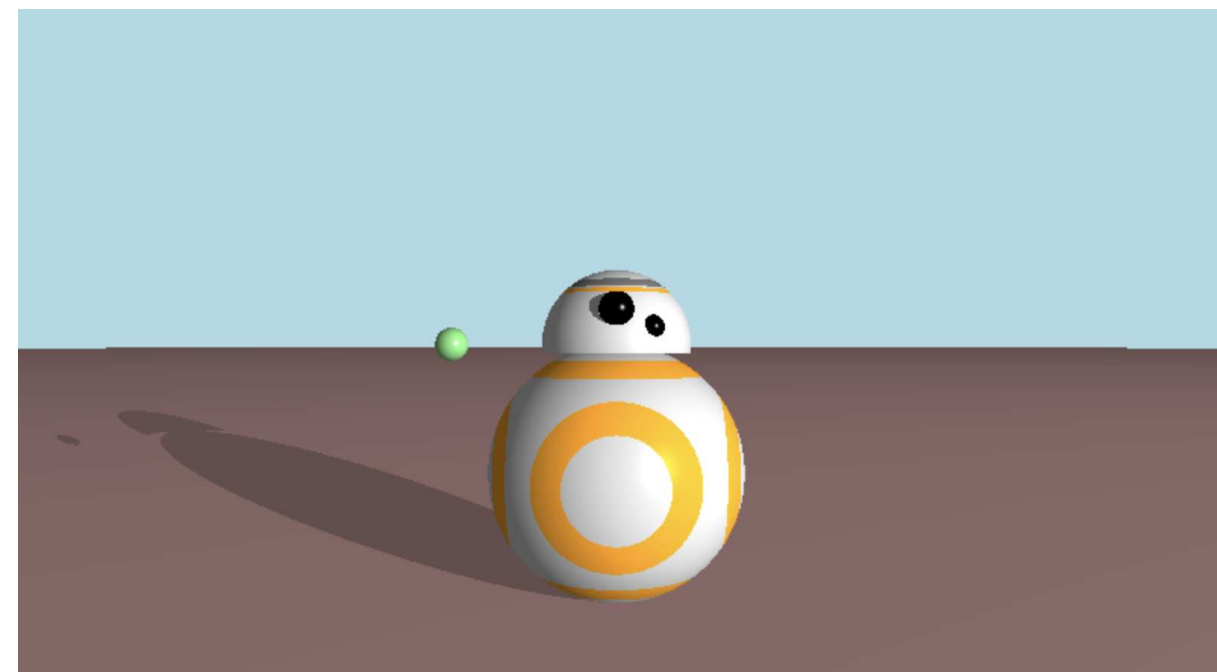
Middlebury College, Fall 2025

Lecture 04: Shading

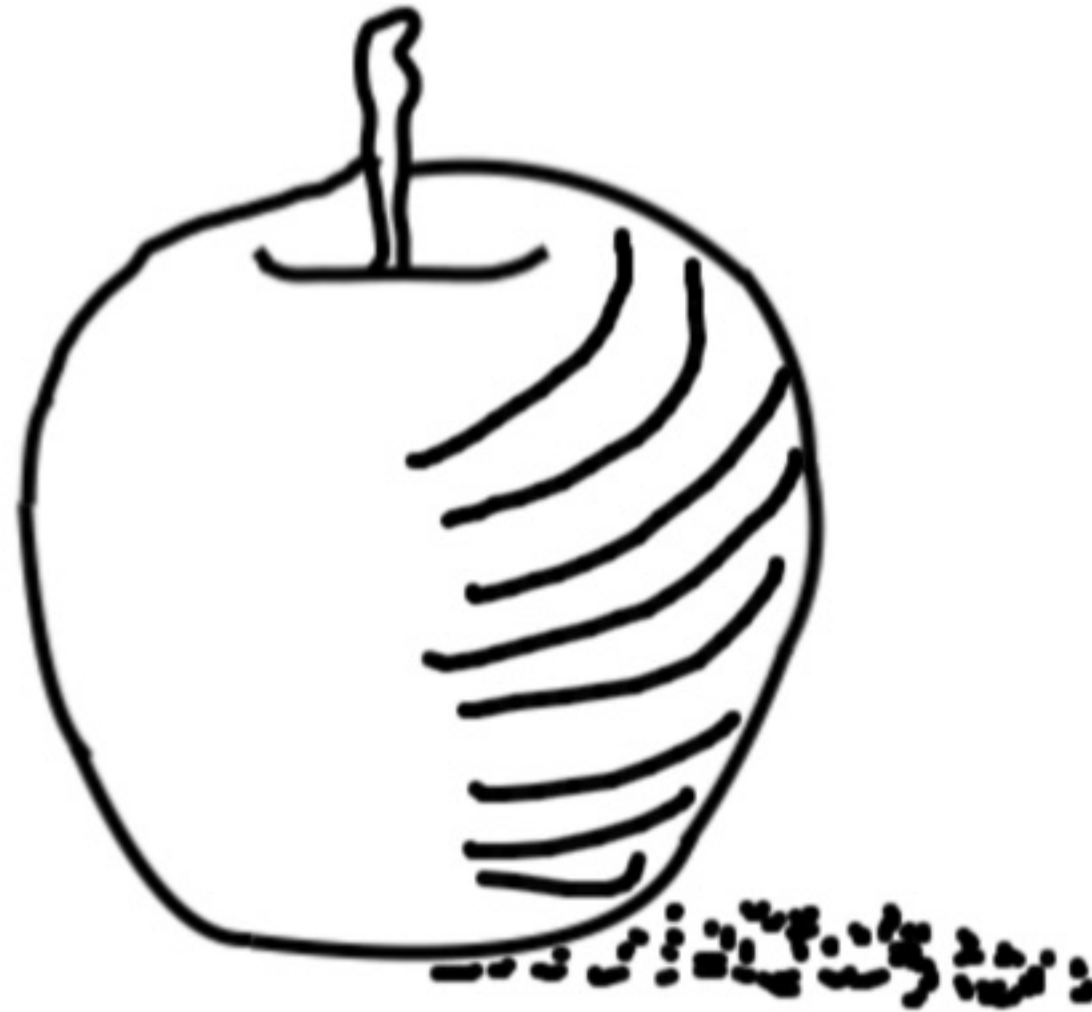
By the end of this lecture, you will be able to:

- compute the **ambient**, **diffuse** and **specular** contributions to the color at a surface point,
- cast a secondary ray to a light source to determine if a point is in a **shadow**,
- cast secondary rays off of **reflective** materials.

Our scenes have looked kind of 2d. Shading will make them look like they're in 3d!



Shading is the process of darkening areas that don't face a light source, and brightening those that do.



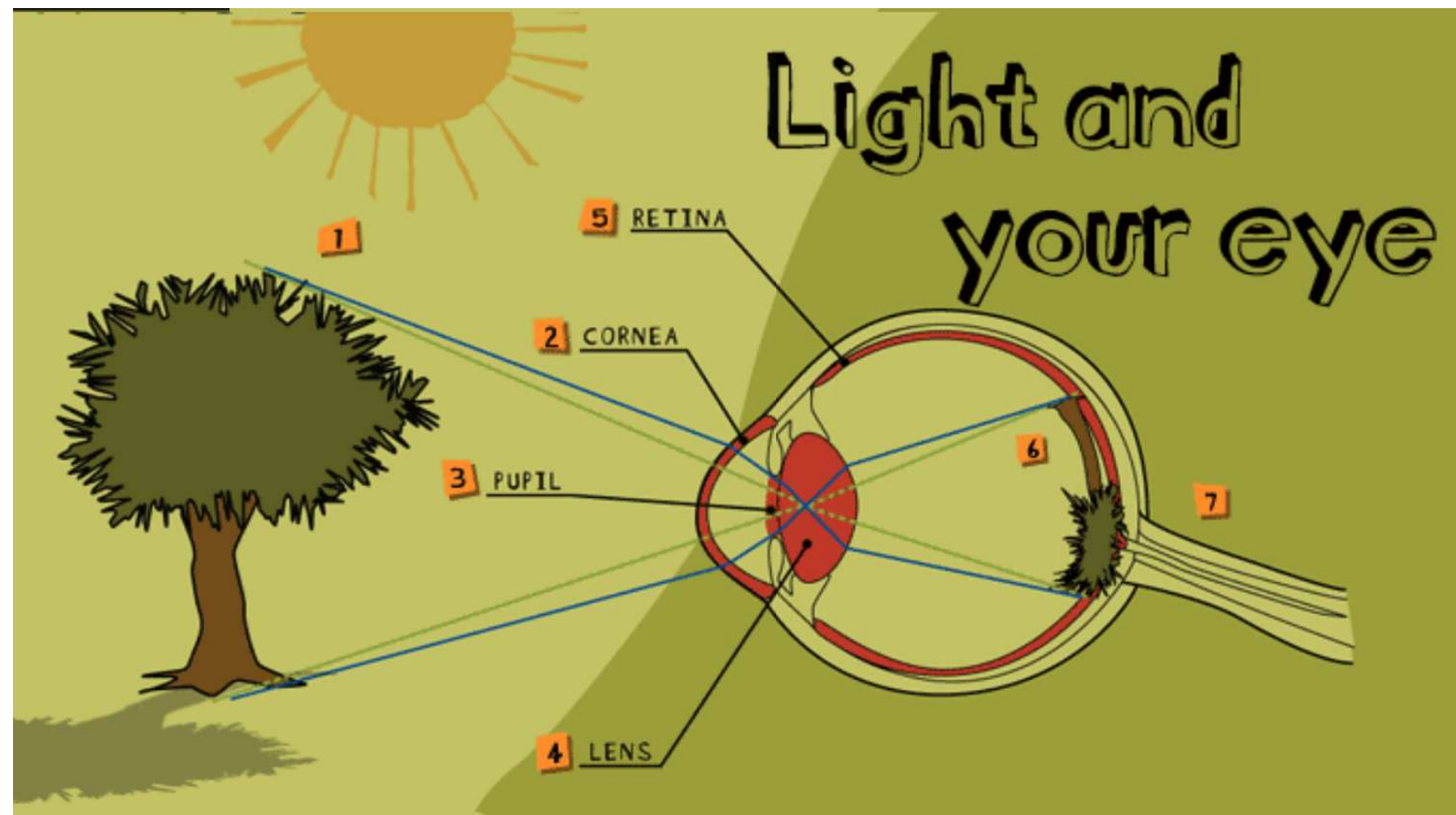
WARNING: shading versus reality!

- All of the shading in this chapter seems like enormous hacks. Is that true?

Yes. However, they are carefully designed hacks that have proven useful in practice. In the long run, we will probably have better-motivated algorithms that include physics, psychology, and tone-mapping. However, the improvements in image quality will probably be incremental.

From FAQ in Chapter 10 of *Fundamentals of Computer Graphics*.

How do we see color? Light travels from a source and reflects off of objects.

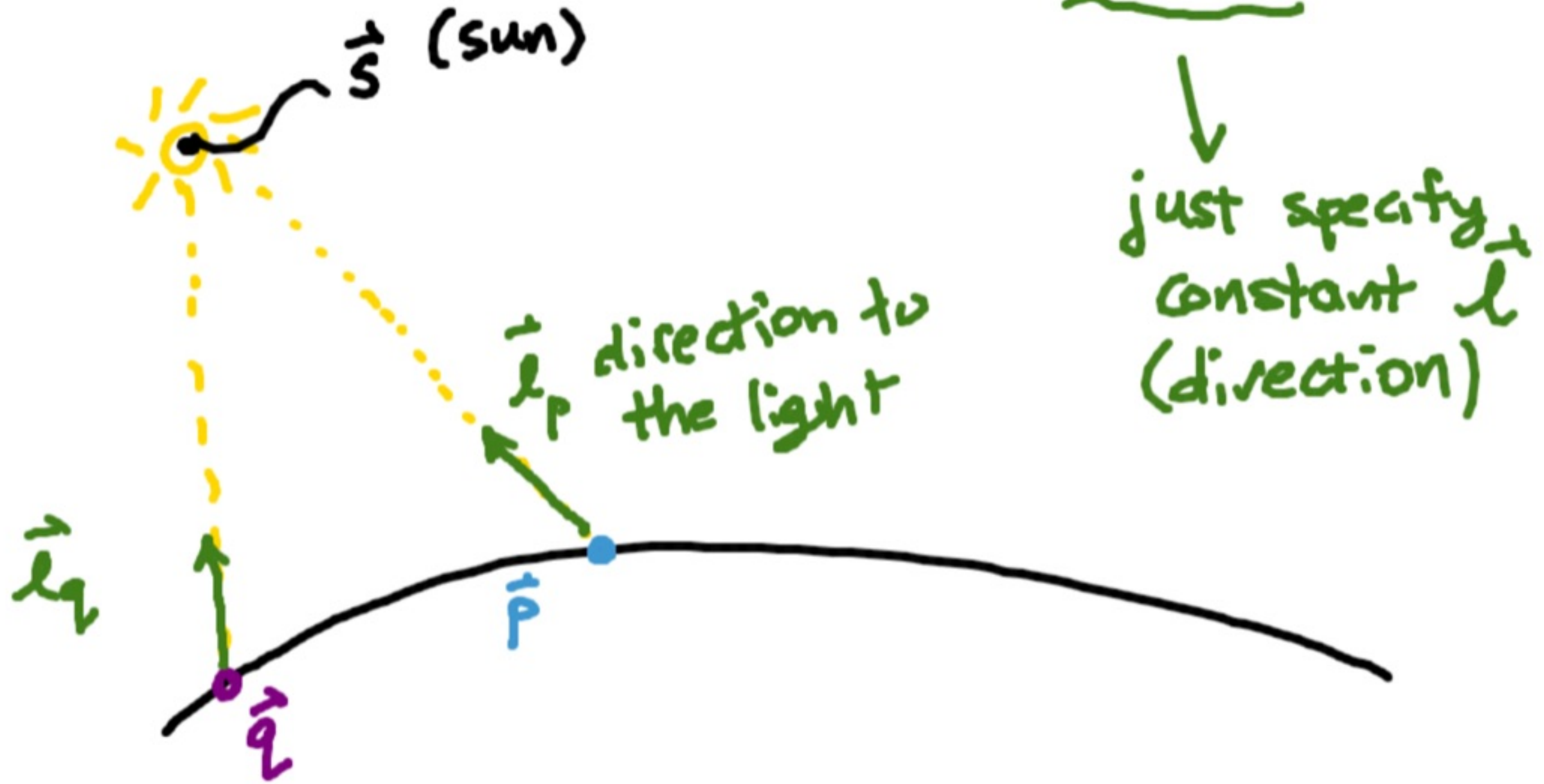


Two ingredients: light sources and materials.

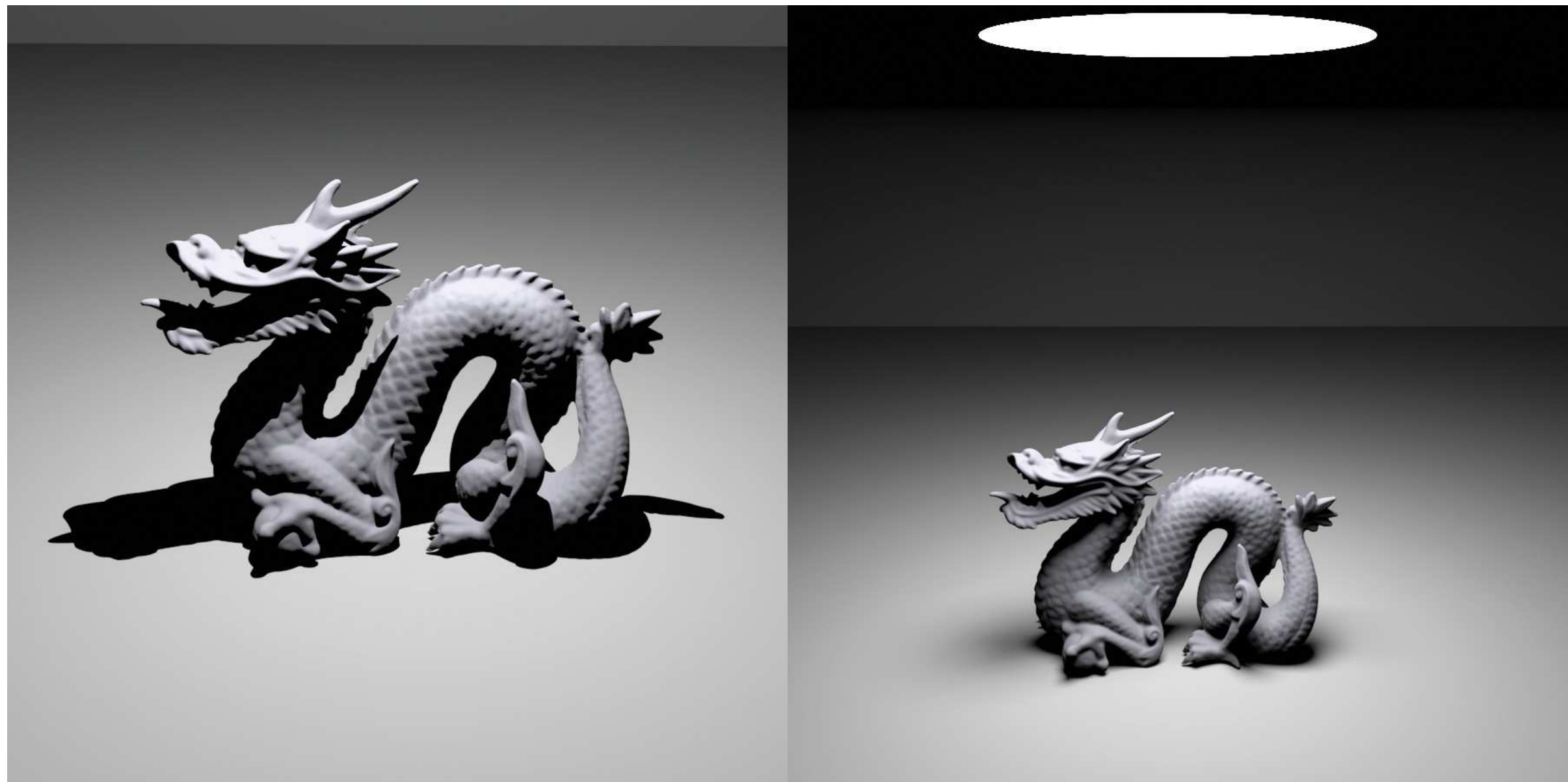
Ingredient #1: Light sources have a color, and a location or direction.



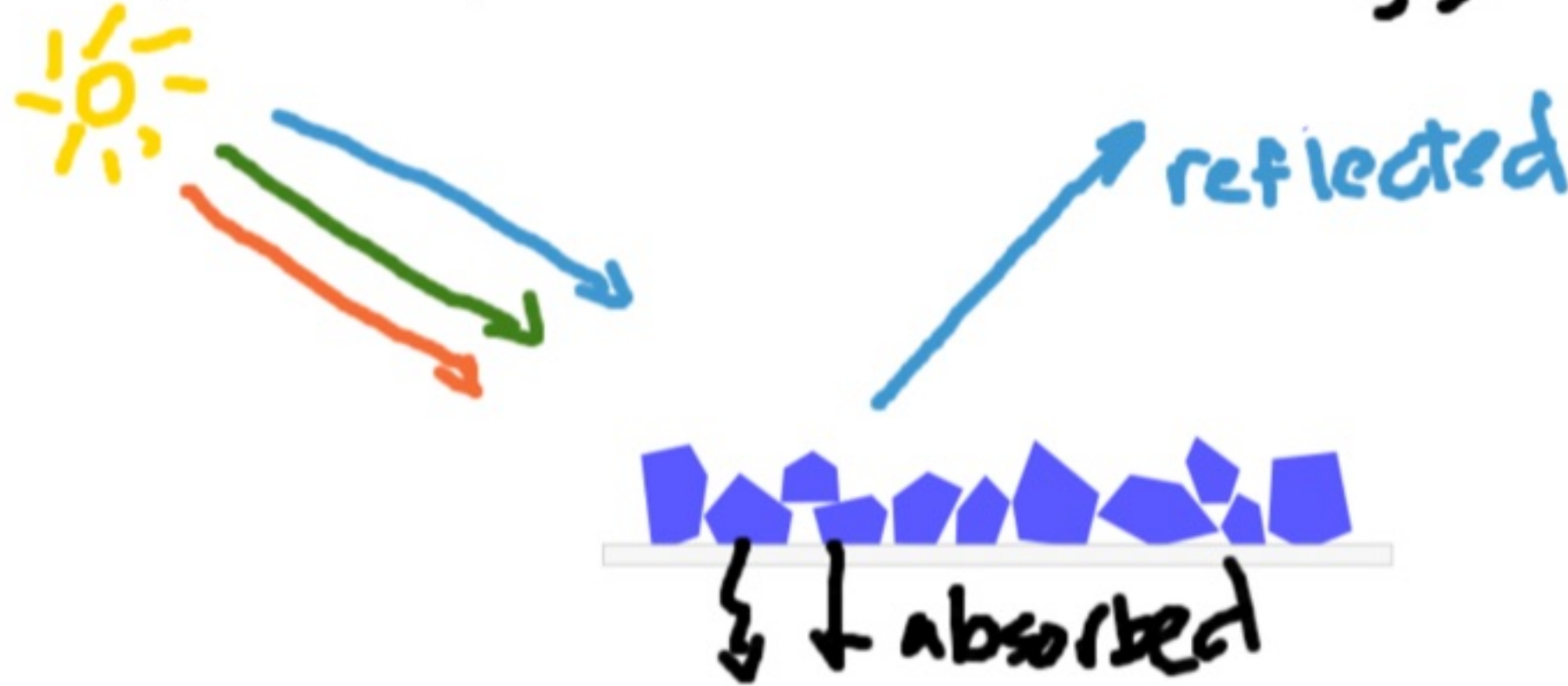
We'll mostly work with a mix of "point" and "directional" lights.



Area lights are more realistic, but harder (and more expensive) to model.



Ingredient #2: Materials reflect certain light components, absorb others.



Things we need to consider:

- **albedo:** fraction of incoming light diffusely reflected,
- **properties:** in what direction is light reflected?

We will look at matte-like, plastic-like, mirror-like and also translucent materials.

Our shading equation (*Phong Reflection Model*):
ambient + diffuse + specular contributions.

$$I_a + I_d + I_s = I$$

Works really well if we want to render plastic!

The ambient contribution (I_a) provides background lighting.

$$I_a = C_a K_m$$
$$= \begin{pmatrix} C_{a,r} K_{m,r} \\ C_{a,g} K_{m,g} \\ C_{a,b} K_{m,b} \end{pmatrix}$$

Handwritten annotations: $C_a = \begin{pmatrix} C_{a,r} \\ C_{a,g} \\ C_{a,b} \end{pmatrix}$ and $K_m = \begin{pmatrix} K_{m,r} \\ K_{m,g} \\ K_{m,b} \end{pmatrix}$ are shown above the equation. A bracket groups the components of the resulting vector.

componentwise multiplication.



$$C_a = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}$$

$I_a = \text{vec3.multiply}(\text{vec3.create()}, C_a, K_m)$

How does the angle between $\vec{n}$ and $\vec{l}$ influence the illumination?

As the angle θ between the surface normal vector $\vec{n}$ and the direction to the light $\vec{l}$ gets bigger, the illumination at the surface

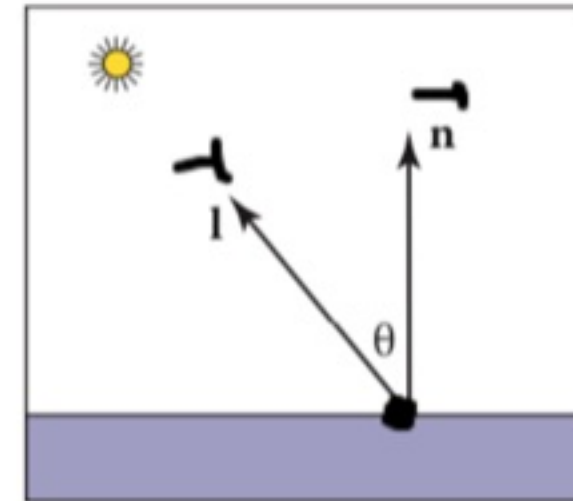
A. gets brighter.

B. gets darker.

C. does not change.

$\theta = 0^\circ$ brightest
 $\theta = 90^\circ$ darkest

Vote at Reactions page of [go/cs461](https://go.cs461.org).



22 71 40 0 0 0 13 0 1 1 2 1 5 3 1



shininess
exponent

[illegible]

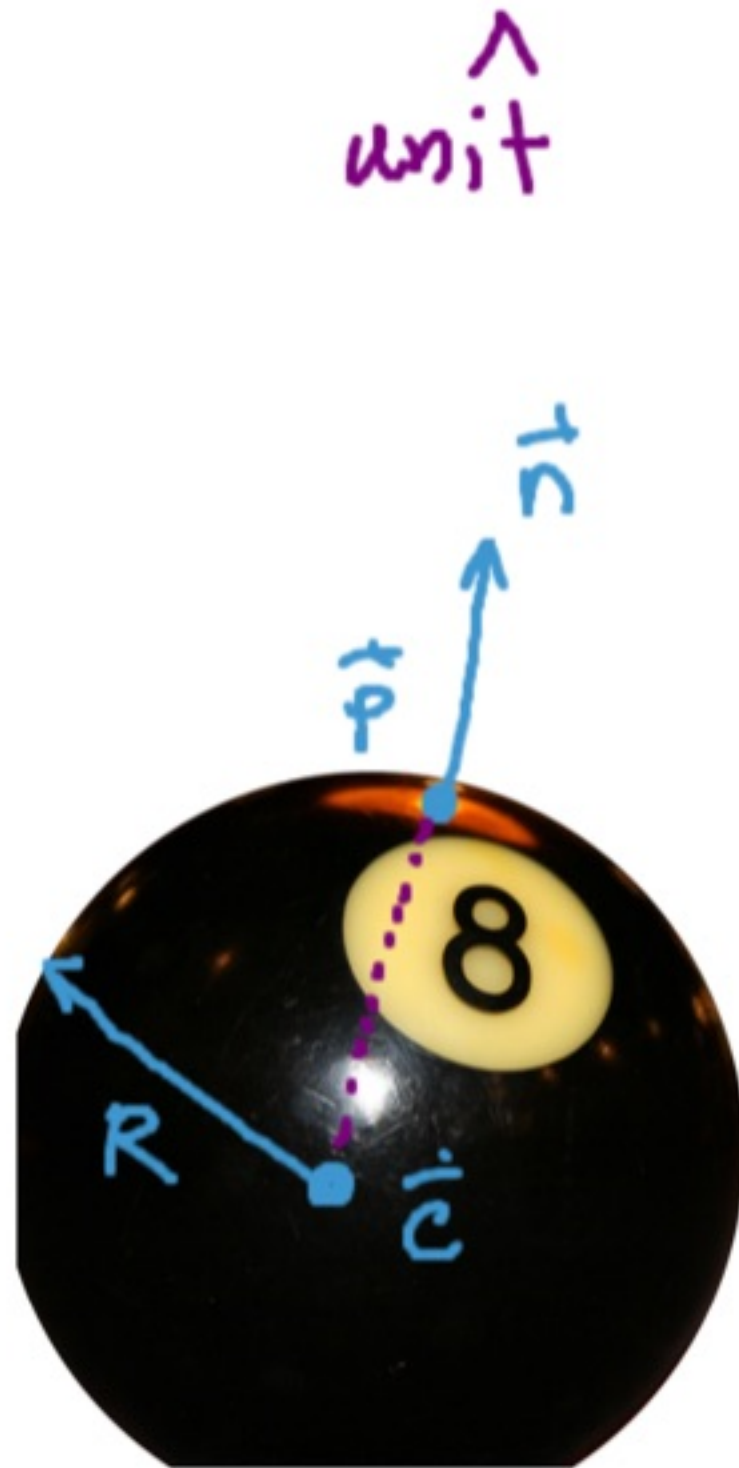


The Phong Reflection Model: how much a surface point is illuminated by light sources.

$$I = c_a k_m + c_l k_m \max \left(0, \vec{n} \cdot \vec{l} \right) + c_s k_s \max(0, \vec{v} \cdot \vec{r})^p$$

- $\vec{n}$: unit surface normal,
- $\vec{l}$: unit vector from surface point to light,
- $\vec{r} = -\vec{l} + 2(\vec{l} \cdot \vec{n})\vec{n}$ (reflection of $\vec{l}$ across $\vec{n}$),
- $\vec{v}$: unit vector from surface point to eye,
- p : shininess

Calculating normal vectors.



$$\hat{n}_{\text{sphere}} = \frac{\vec{P} - \vec{C}}{\|\vec{P} - \vec{C}\|} \quad \text{] what is this?}$$

vec3.normalize

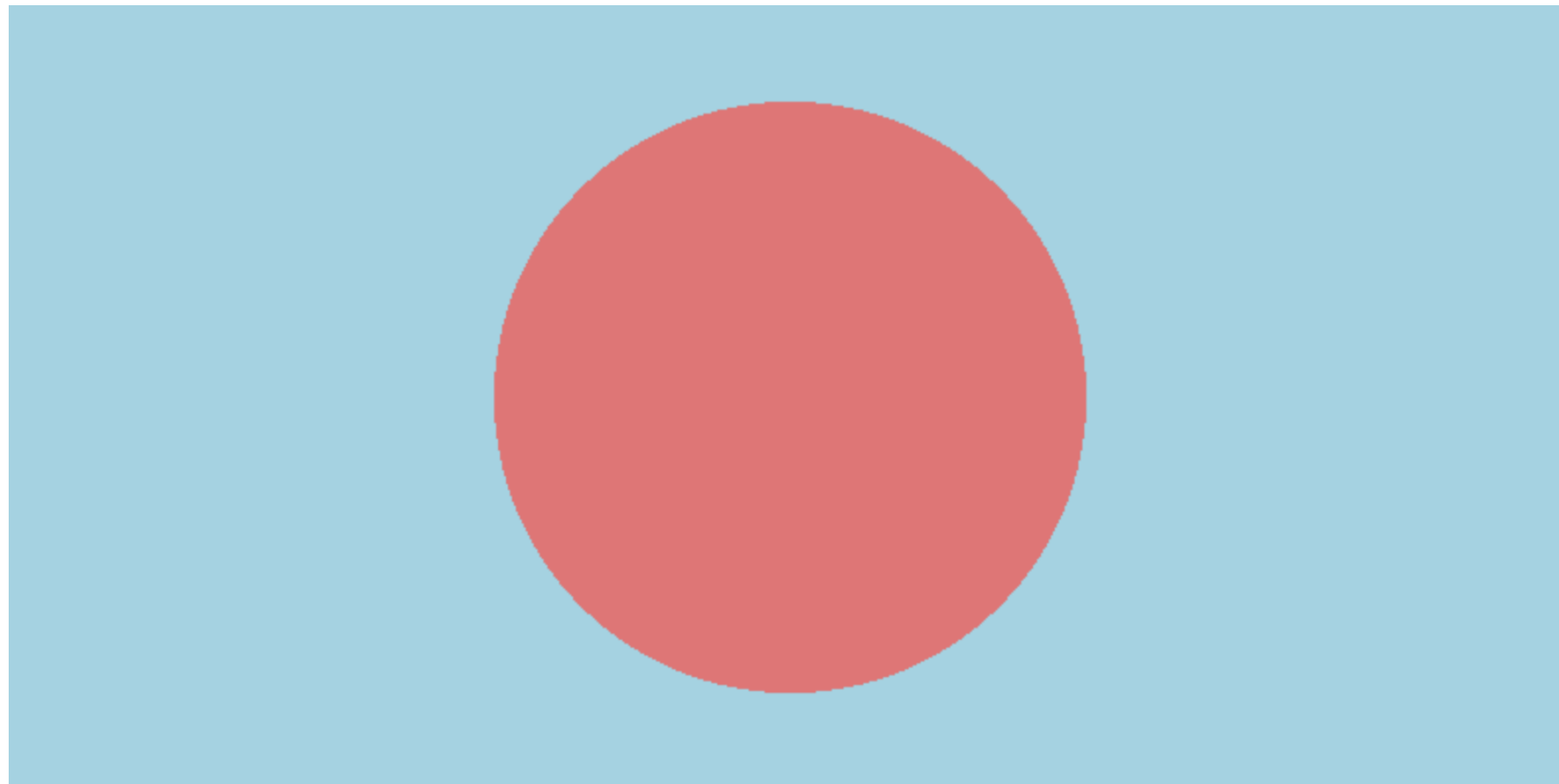
$$= \frac{\vec{P} - \vec{C}}{R} \quad \text{vec3.subtract}$$

/ R X

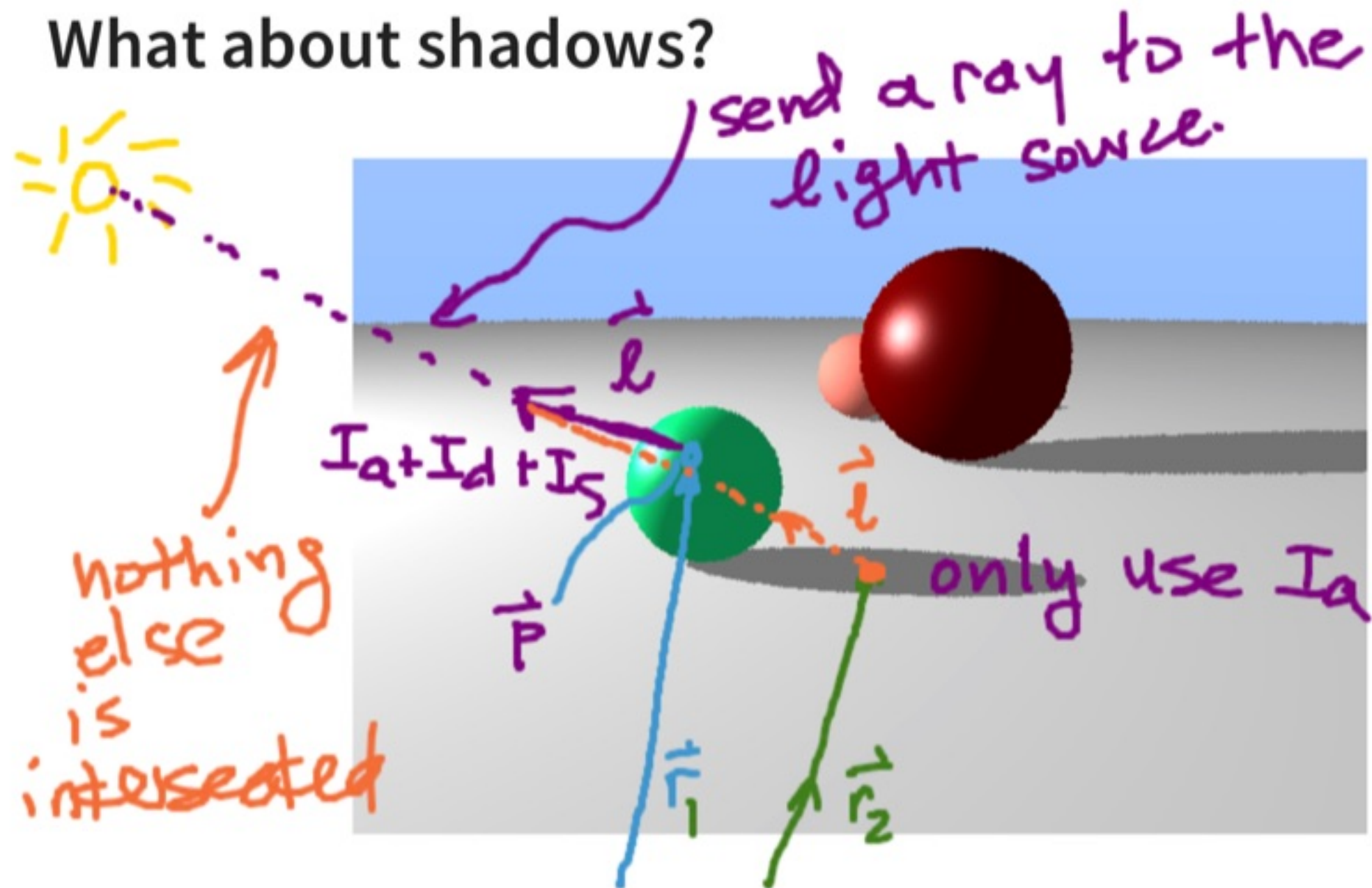
do not use vec3.divide
scale by $1/R$

Practice exercise!

Click to open the editor.

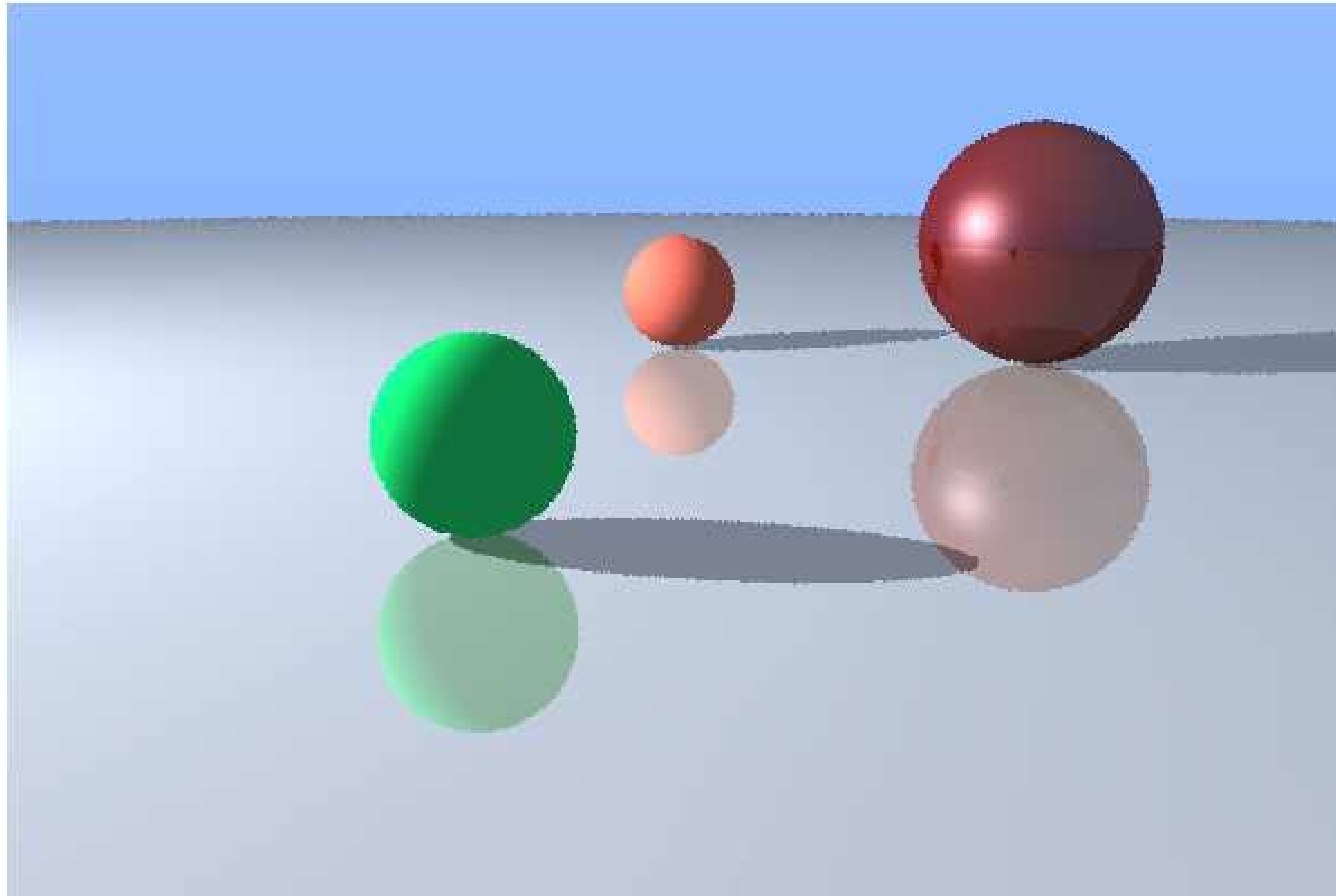


What about shadows?



Cast a ray from intersection point to light source.
No intersection? use I_a , otherwise add I_d and I_s .

What about mirrors?

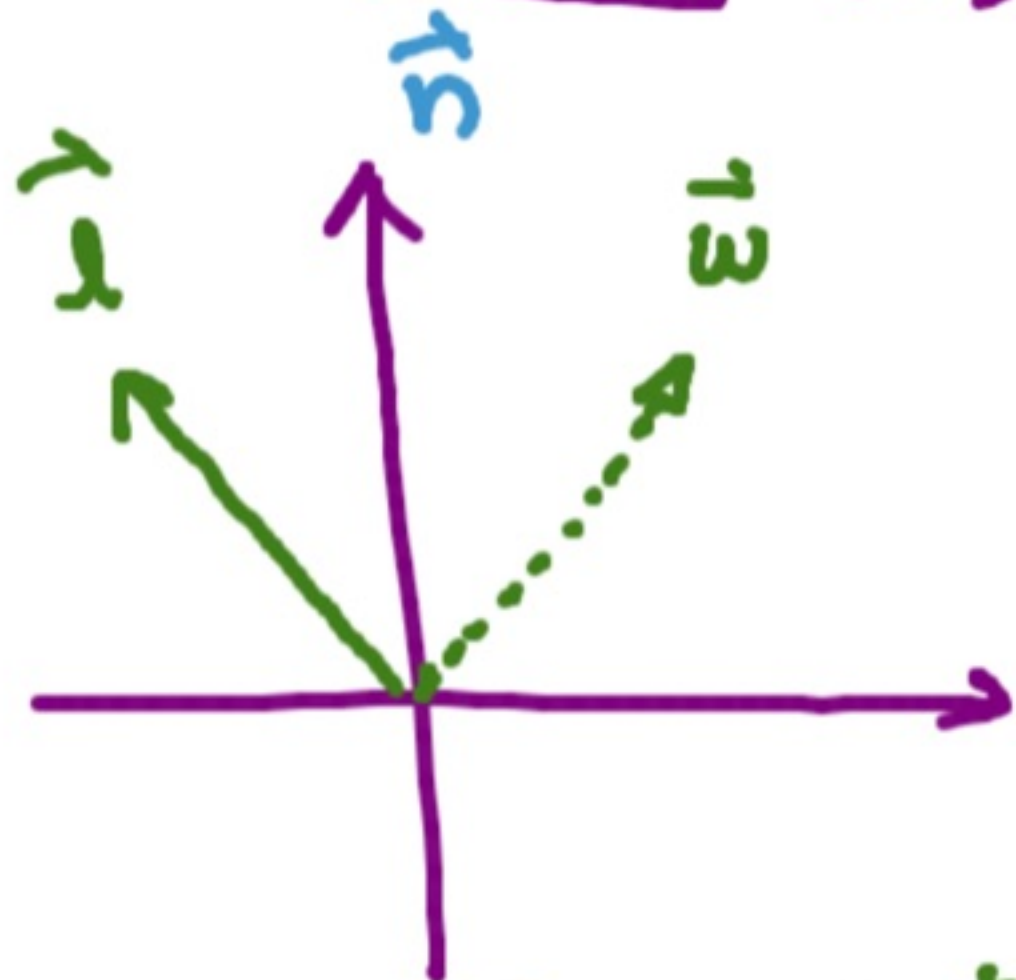


Cast a ray from intersection point in reflection direction (reflect ray direction across $\vec{n}$).
Calculate color recursively!

Computing the reflection direction.

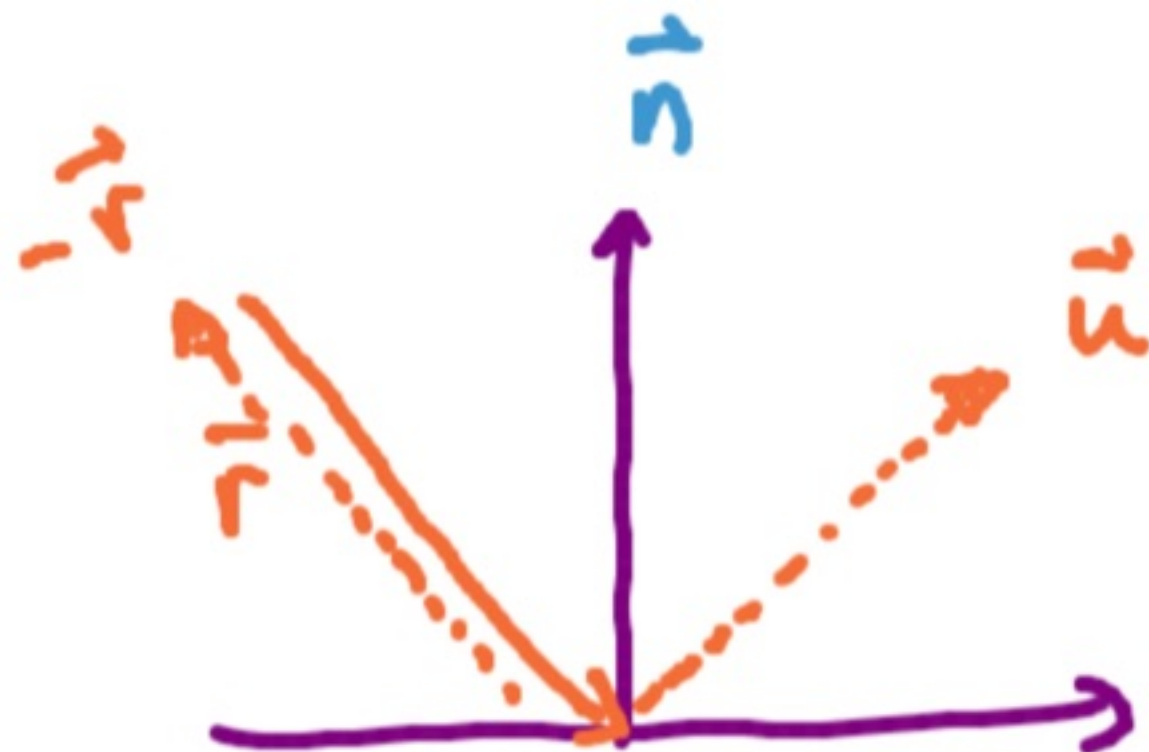
Computing the reflection direction.

specular (I_s)



$$\vec{\omega} = -\vec{x} + 2(\vec{n} \cdot \vec{x})\vec{n}$$

Mirror



$$\begin{aligned}\vec{u} &= -(-\vec{r}) + 2(-\vec{r} \cdot \vec{n})\vec{n} \\ \vec{u} &= \vec{r} - 2(\vec{r} \cdot \vec{n})\vec{n}\end{aligned}$$

Pseudocode for a ray tracer with what we covered today:

```
1 function computeColor(ray, currentDepth)
2
3   // initialization and check for maximum recursion depth (ray bounces)
4   color = background sky color
5   if (currentDepth > maxDepth) return color
6
7   // check if any intersection occurs
8   if (ray does not intersect any object) return color
9
10  // initialize to ambient color term
11  color = Ia
12
13  // check if this point is in the shadow of another object
14  shadowRay = new ray from intersection point to light
15  if (shadowRay intersects an object) return color // only keep ambient term
16
17  // not in a shadow: add other terms and check if secondary rays are needed
18  add diffuse (Id) and specular (Is) terms to color
19  if (intersection object is mirror) {
20    calculate reflectionDirection
21    reflectedRay = new ray from intersection point in reflectionDirection
22    reflectionColor = computeColor(reflectedRay, currentDepth + 1)
23    add reflectionColor to color (possibly scale)
24  }
25
26  return color
```


Limit number of bounces in recursive ray tracer.



Summary

- Calculate color = ambient + diffuse + specular terms.
- Good idea to write a general function to determine intersection of ray with objects in scene (this is why both **Sphere** and **Triangle** classes in Lab 2 had a similar **intersect** function).
- You will need to return information about the intersection as well (point, normal, material) - not just t anymore!
- See feedback on Lab 1: final submission due on 10/7.
- Complete pre-lab 3 by Thursday (before the lab).

