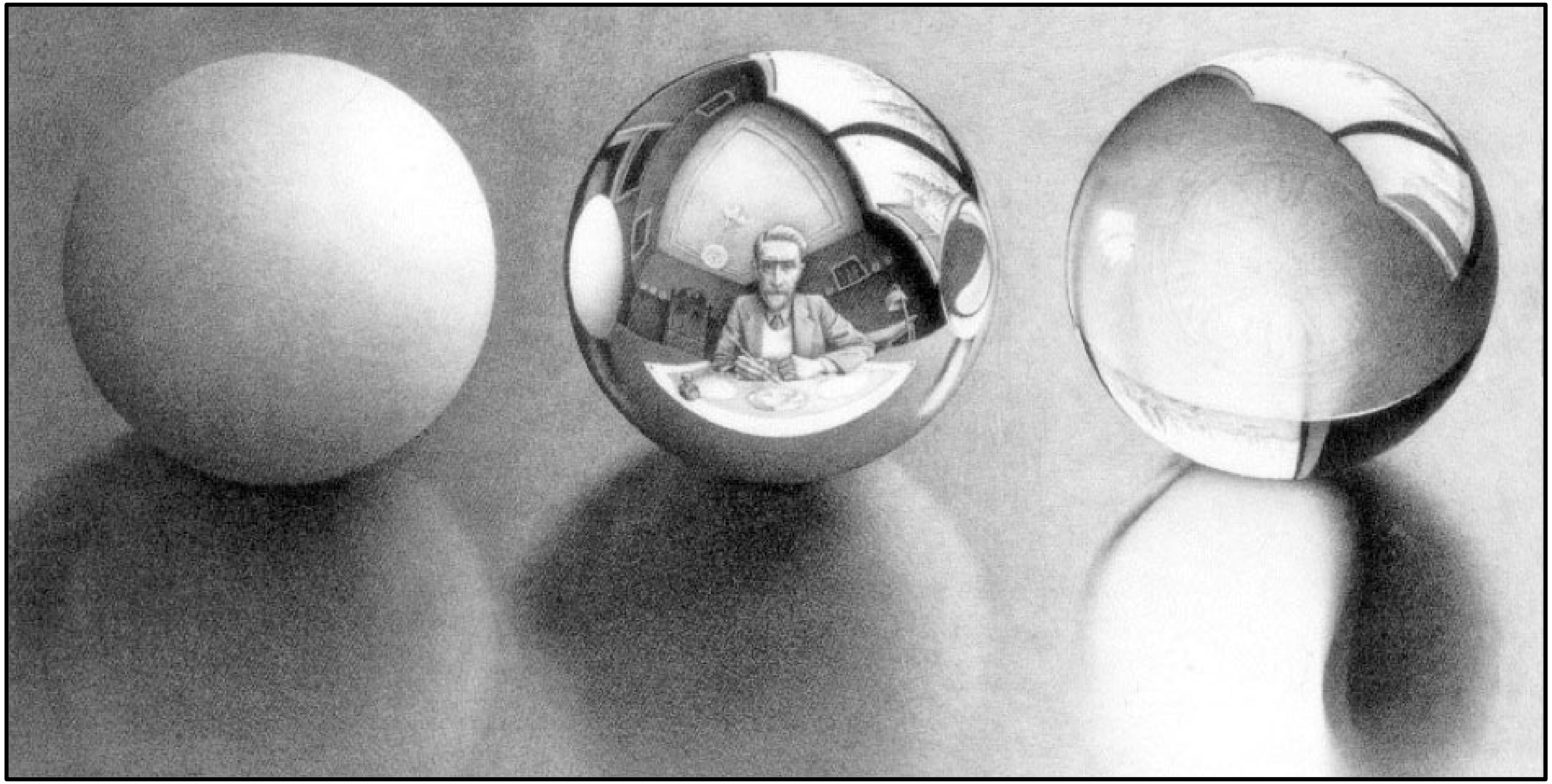
Ray tracing and simple shading



15-468, 15-668, 15-868
Physics-based Rendering
Spring 2021, Lecture 3

http://graphics.cs.cmu.edu/courses/15-468

Course announcements

- Take-home quiz 1 will be posted **tonight**, Tuesday 2/9, and will be due a week later.

- Programming assignment 1 will be posted on Friday 2/12, will be due two weeks later.
  - We will post an *ungraded programming assignment 0* tomorrow, Wednesday 2/10.
  - Used to set up our rendering environment and github-based submission system.
  - Should take no more than 1 – 2 hours max.

- Office hours *for the rest of the semester*:
  - Bailey, Tuesdays 3 – 5 pm ET.
  - Yannis, Thursdays 2 – 4 pm ET.
  - Zoom details on Piazza and Canvas.
  - Yannis may schedule additional office hours on Fridays.

Overview of today's lecture

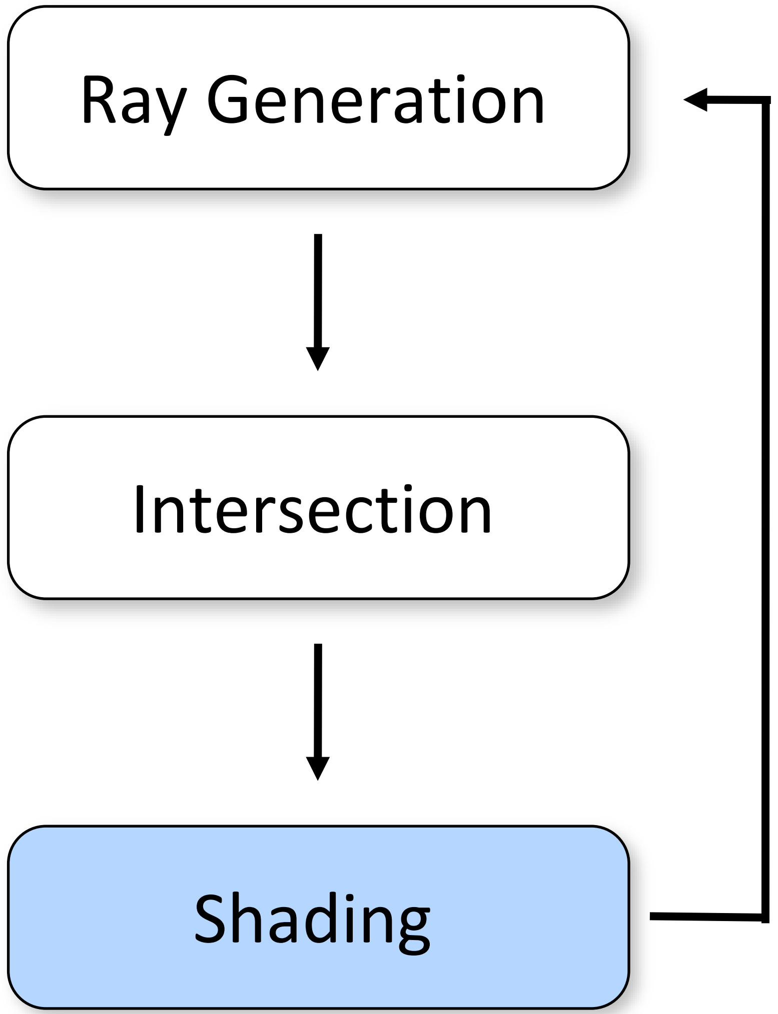
- Leftover from previous lecture: intersections, meshes, acceleration structures.
- Basics of shading.
- Basic reflection models.

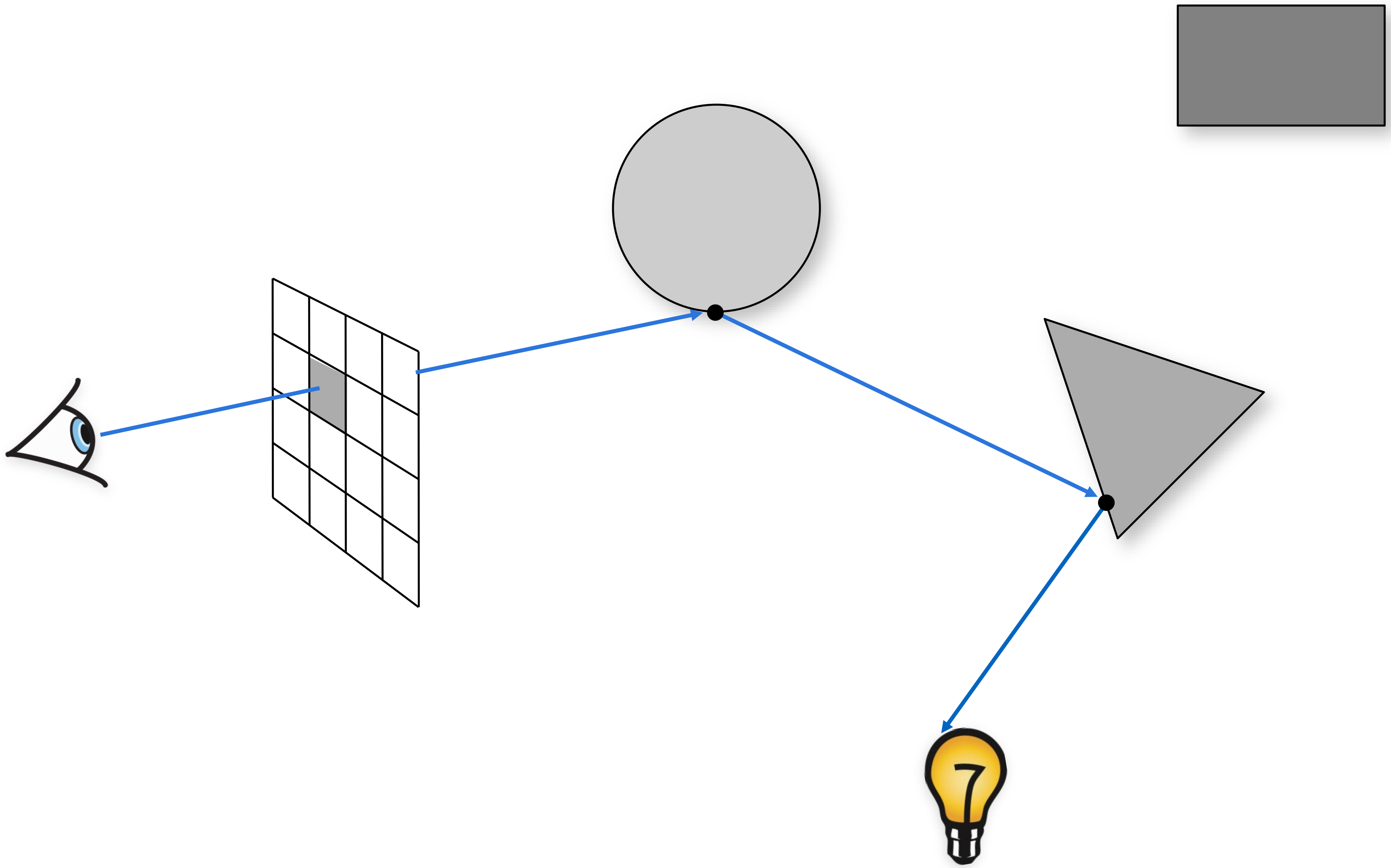
Slide credits

Most of these slides were directly adapted from:

- Wojciech Jarosz (Dartmouth).

Recap: Raytracing





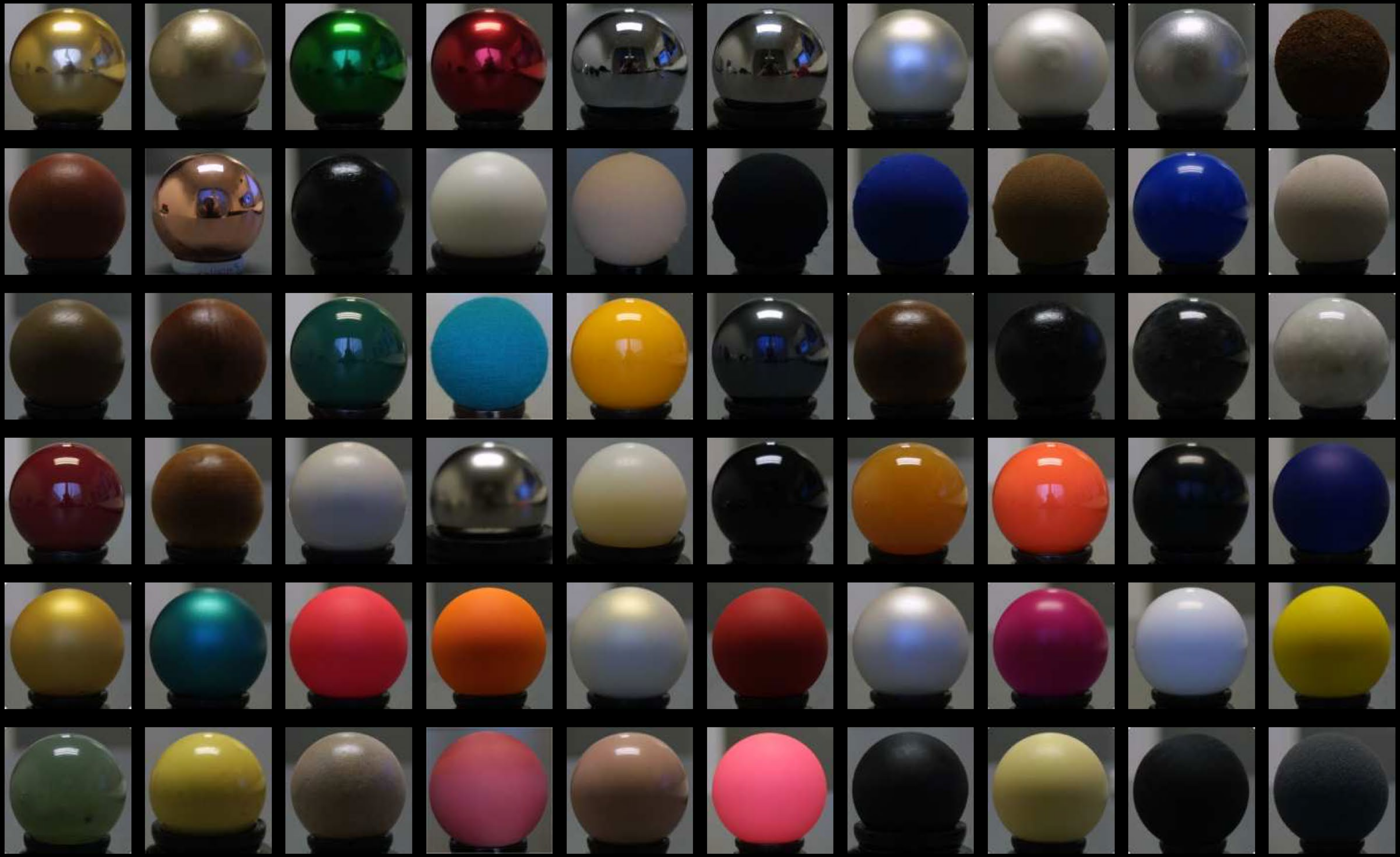
Shading

When ray hits a surface we perform *lighting/shading*

Determine “what color/light should we see at this location?”

Surfaces can scatter and/or emit light

- Surface emits light? just return emitted color (determined by the material)
- Surface scatters/reflects/refracts light? (recursively) trace a ray in a scattering direction (determined by the underlying material)



Overview

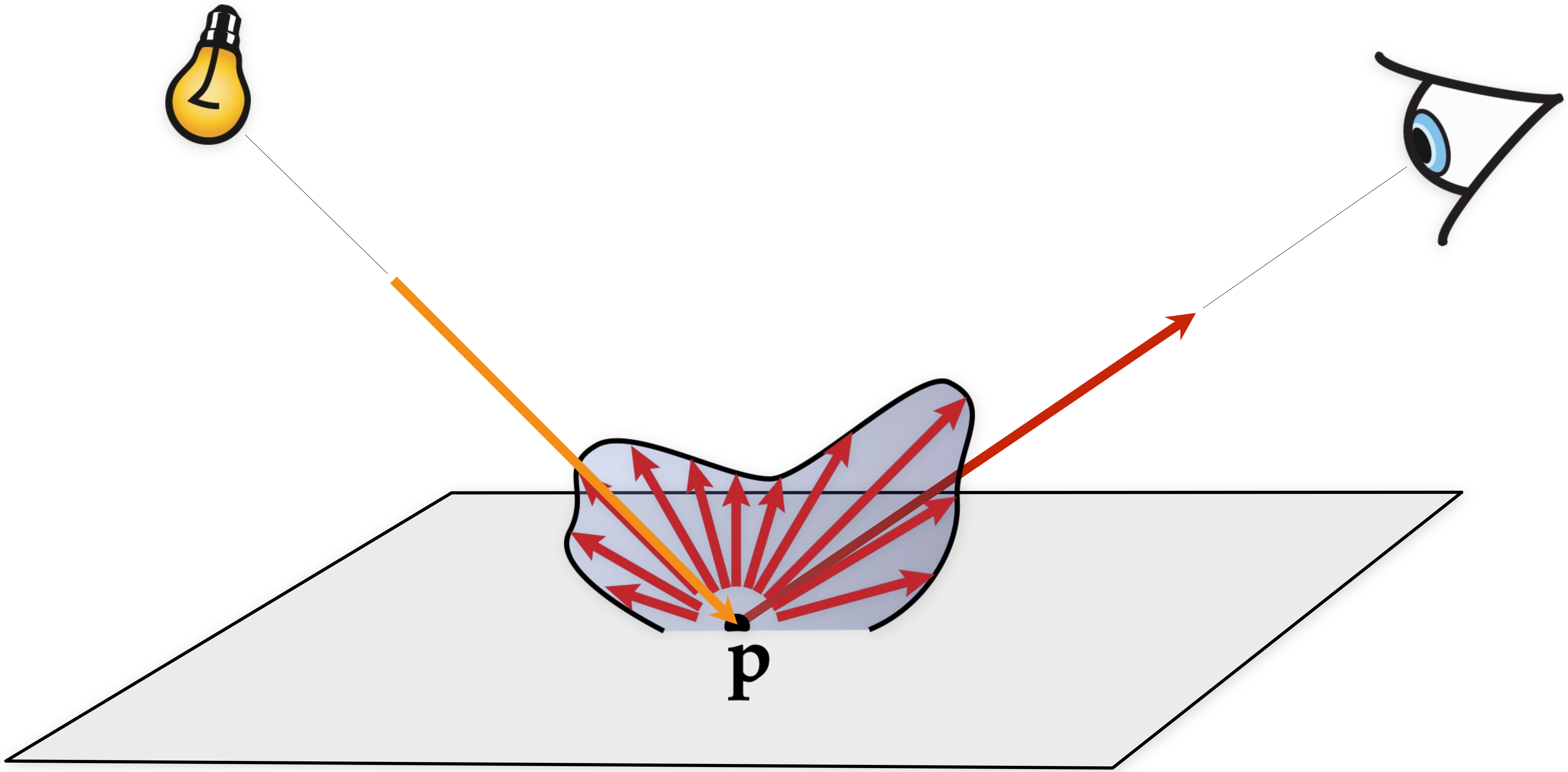
Diffuse shading

Specular reflection

Refraction

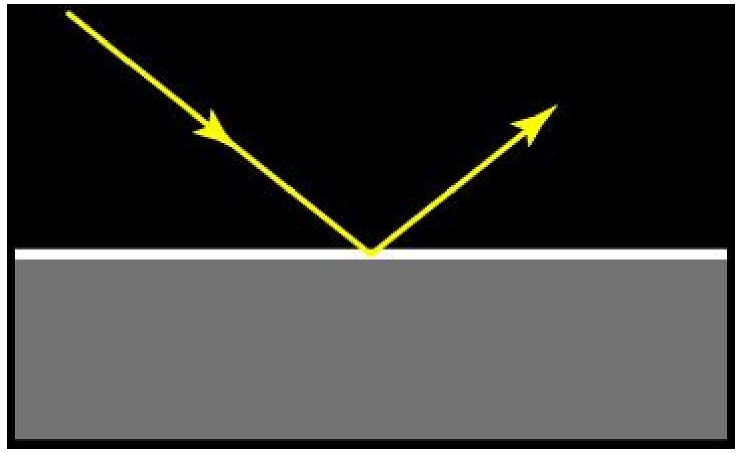
Diffuse emission

Light-material interactions

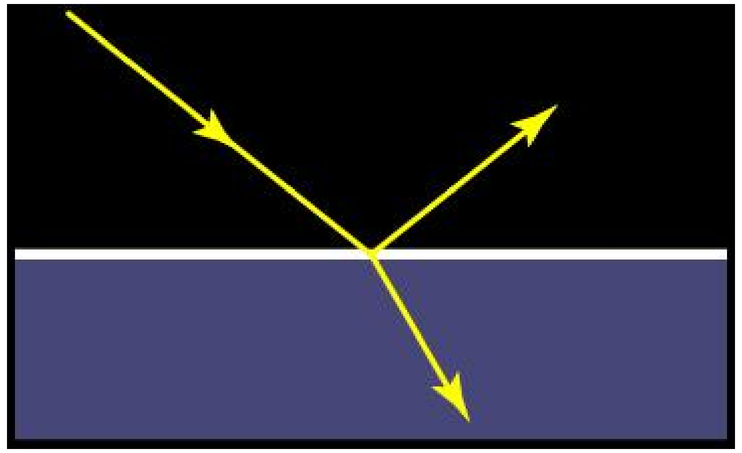


Real-world materials

Metals

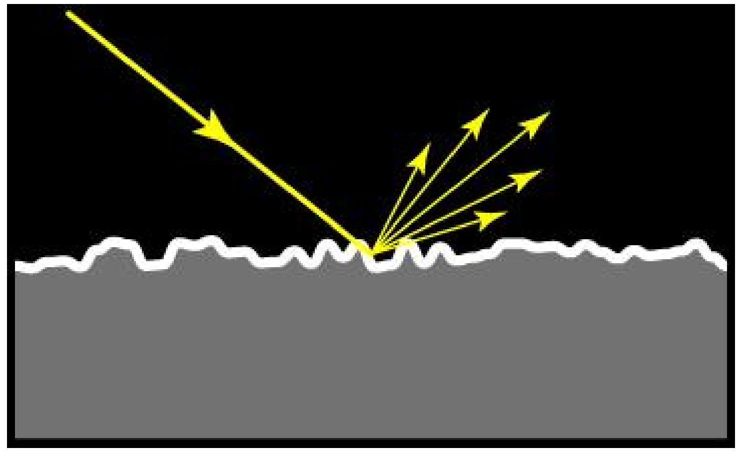


Dielectric

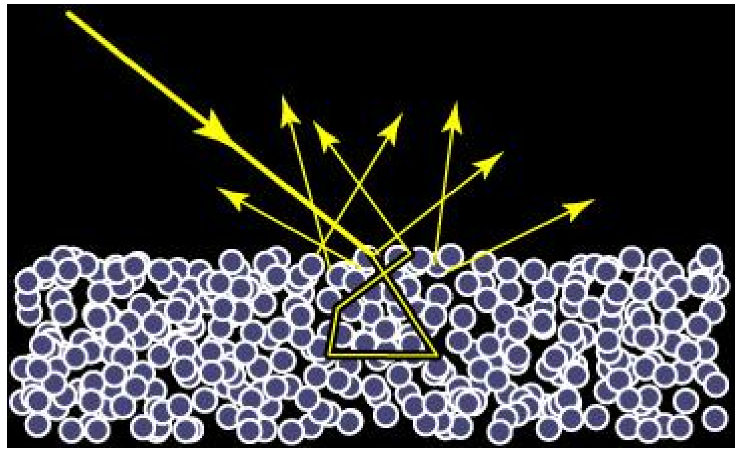


Real-world materials

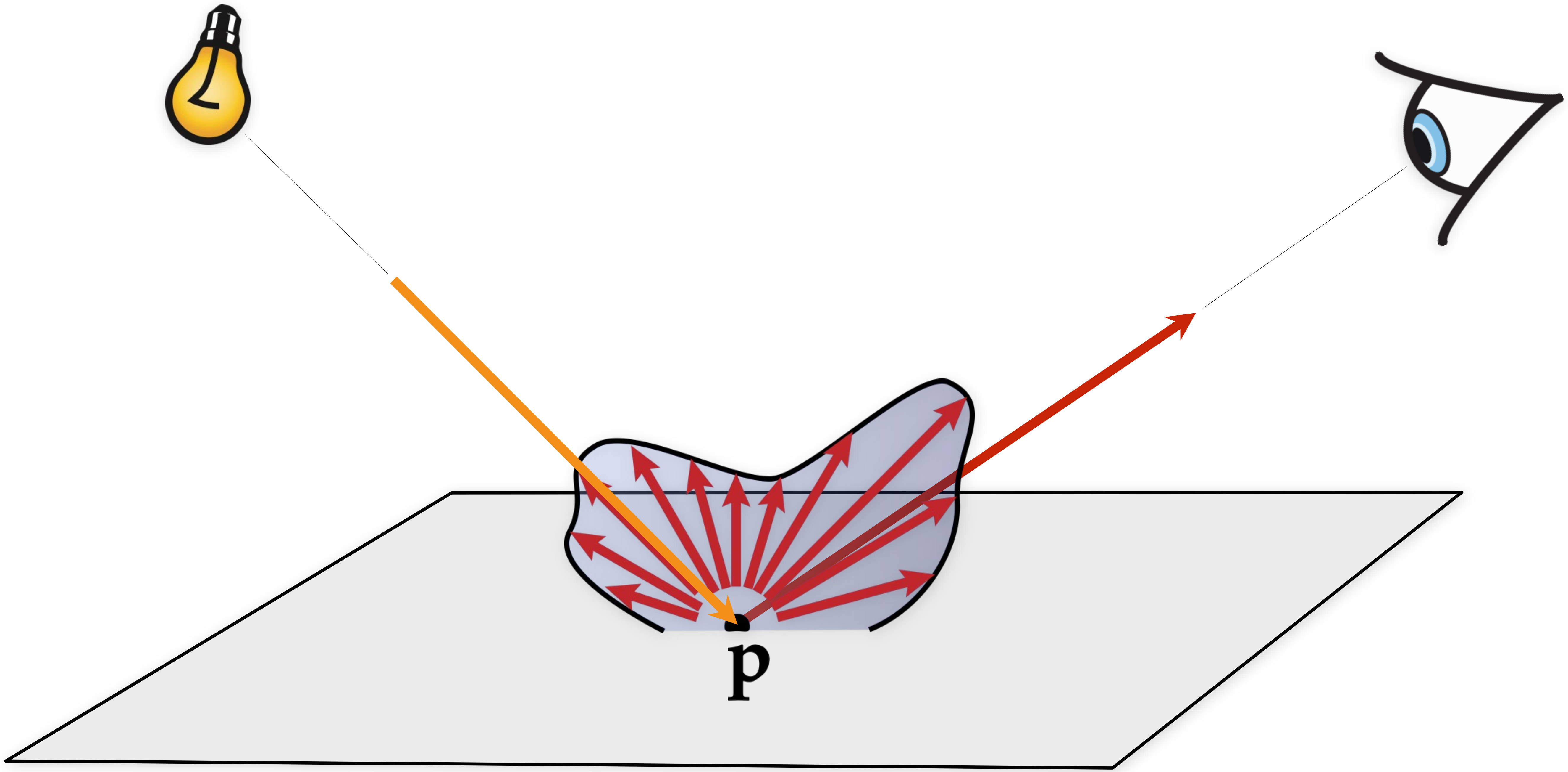
Metals



Dielectric



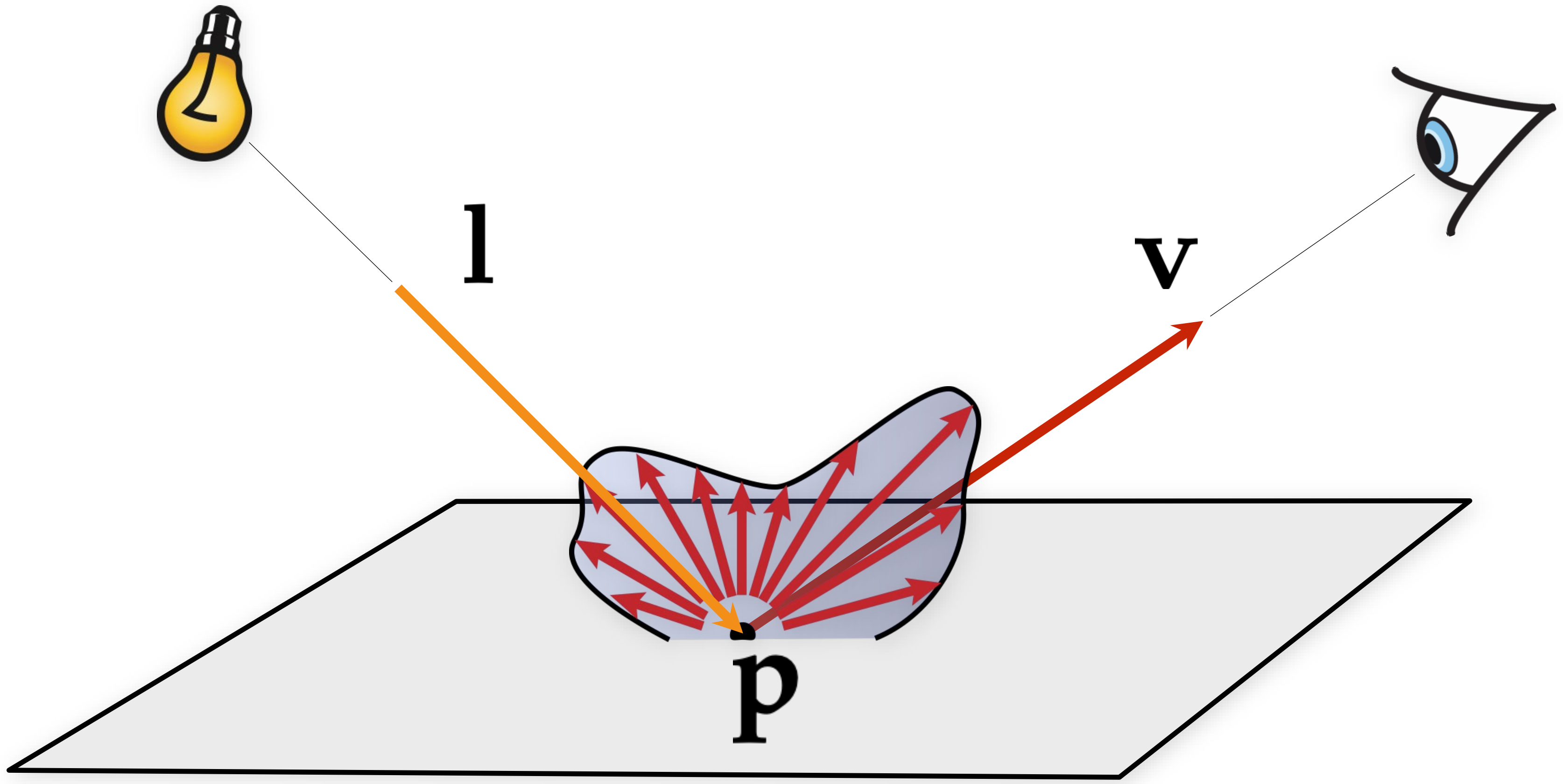
Light-material interactions



The BSDF

Bidirectional **S**cattering **D**istribution **F**unction

- informally: how much the material scatters light coming from one direction $\mathbf{l}$ into some other direction $\mathbf{v}$, at each point $\mathbf{p}$



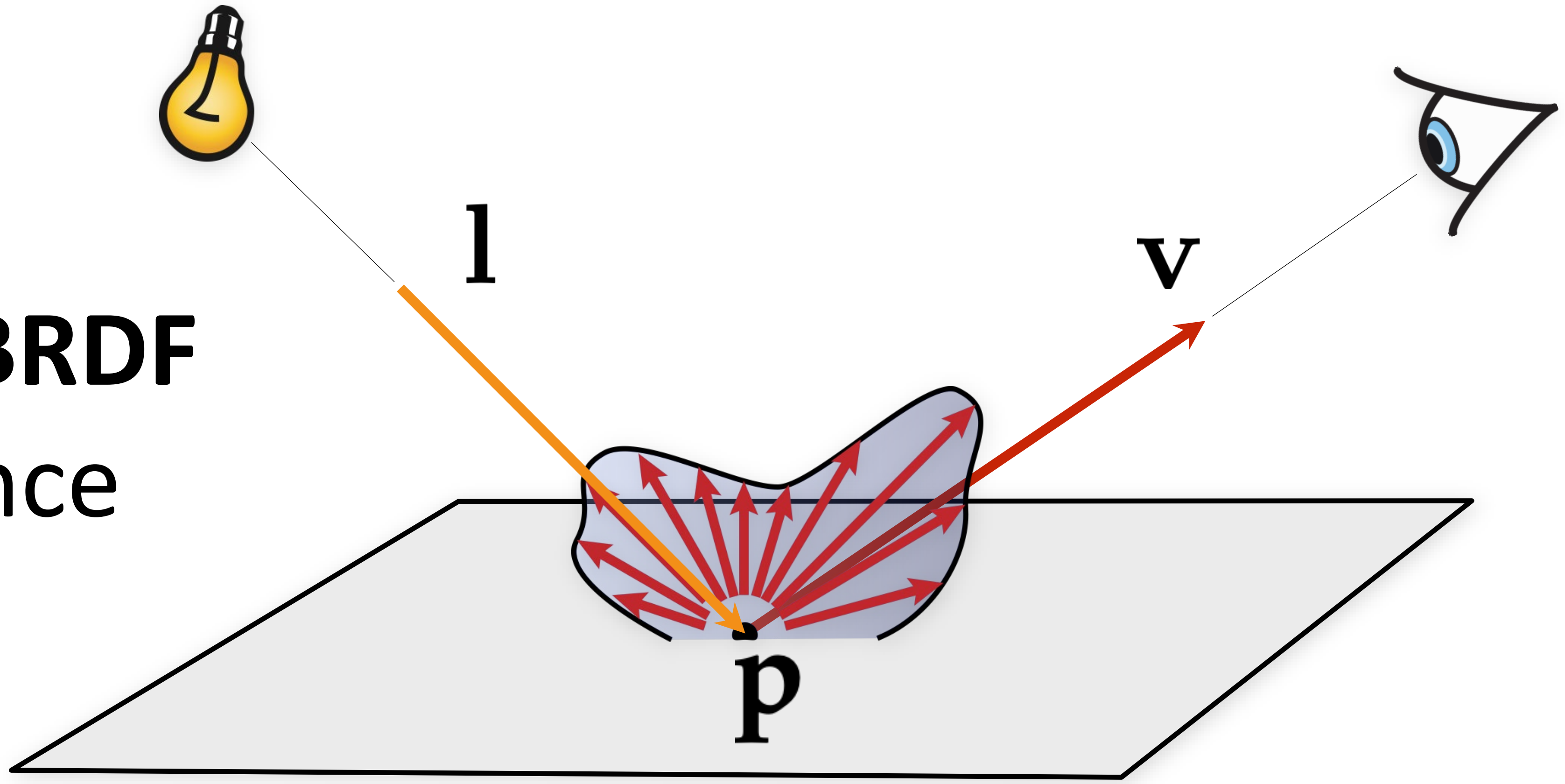
The BSDF

Bidirectional **S**cattering **D**istribution **F**unction

- informally: how much the material scatters light coming from one direction $\mathbf{l}$ into some other direction $\mathbf{v}$, at each point $\mathbf{p}$

You'll also see **BRDF**

R for reflectance



Idealized material models

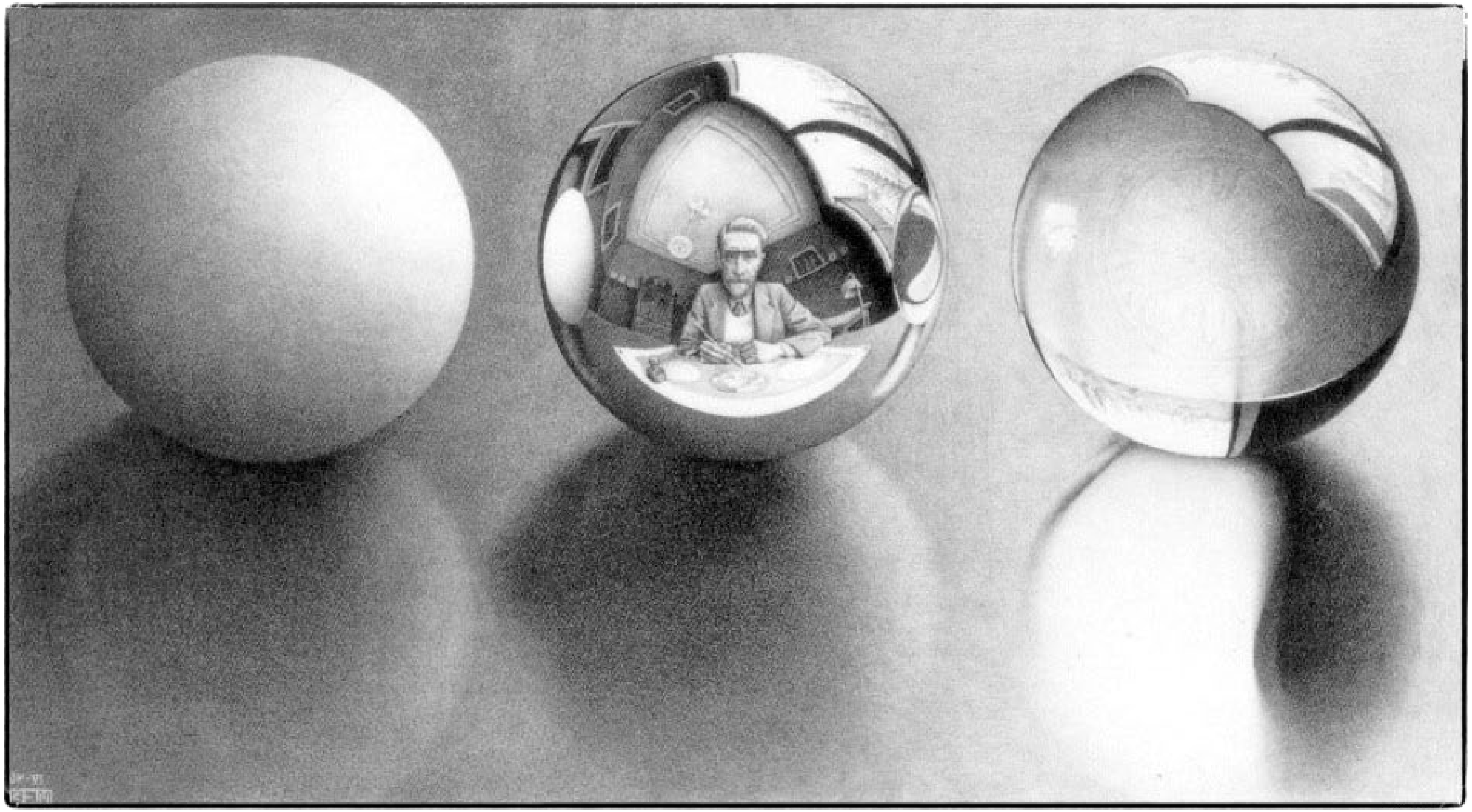
Diffuse reflection

- light is reflected in all directions
- colored by surface color

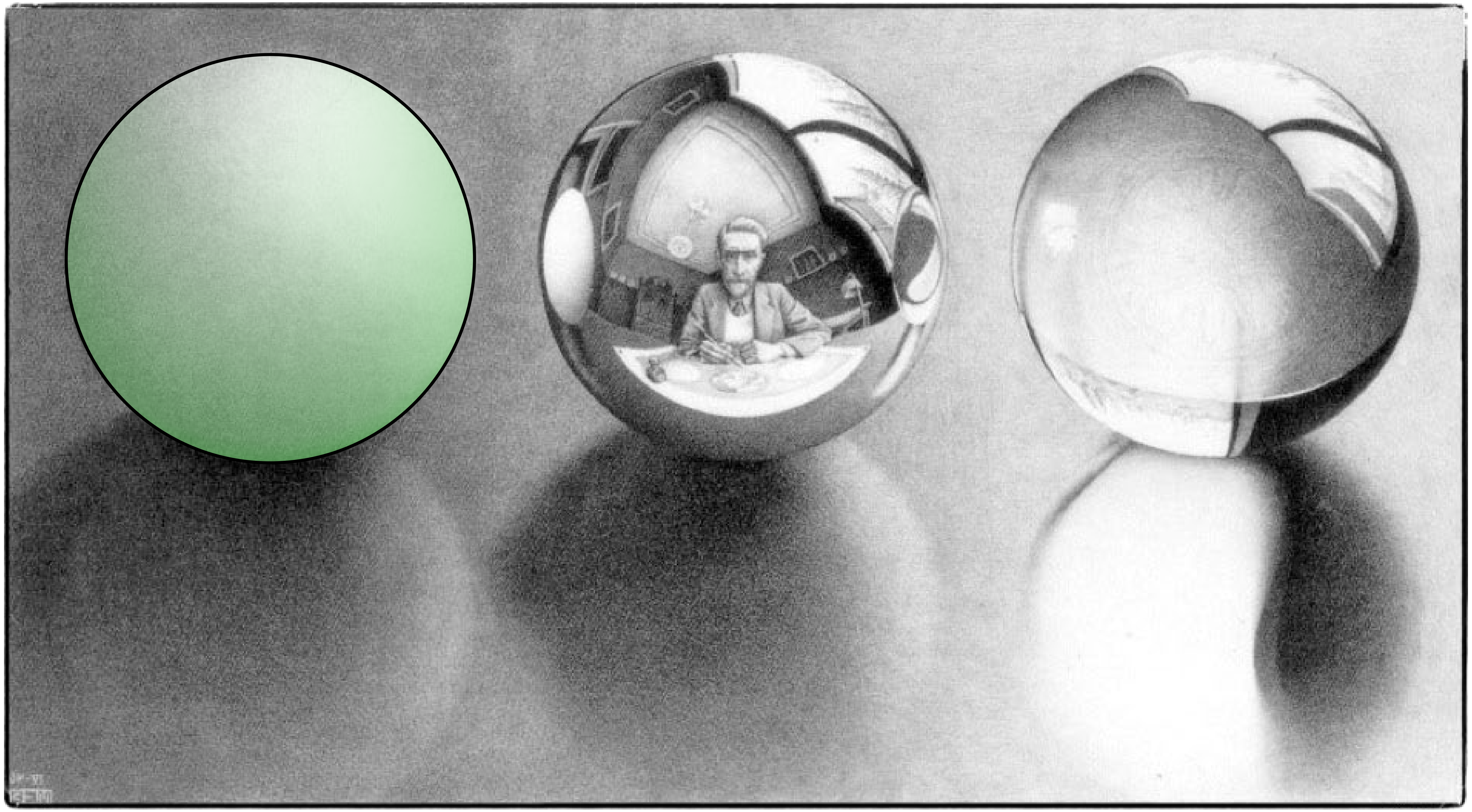
Smooth specular reflection/refraction
(e.g. chrome, glass, glaze/varnish)

- light reflected/refracted only in a single direction
- colored by source color

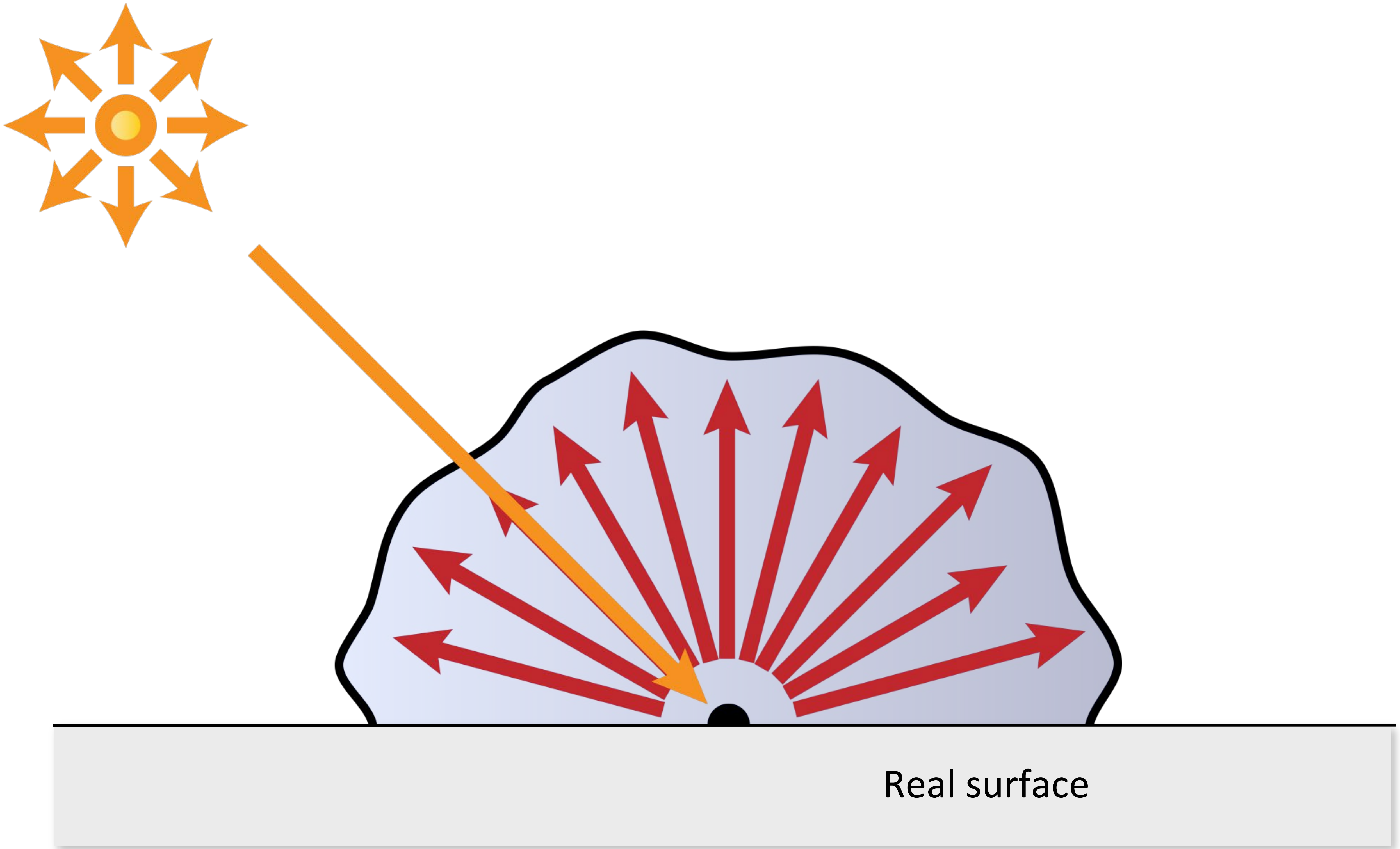
Idealized materials



Diffuse reflection

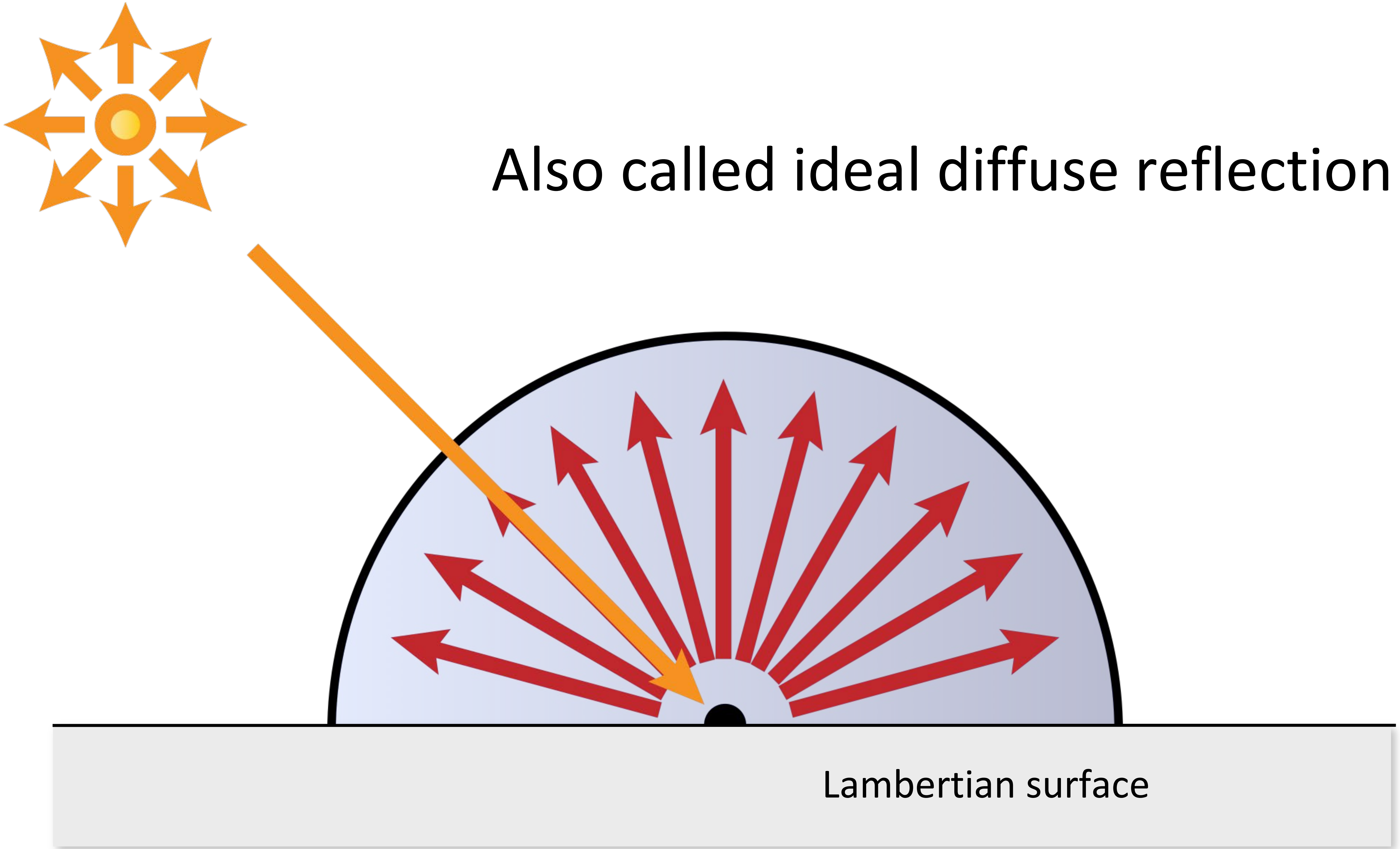


Diffuse reflection



Lambertian reflection

Also called ideal diffuse reflection



Basic Ray Tracing Pipeline

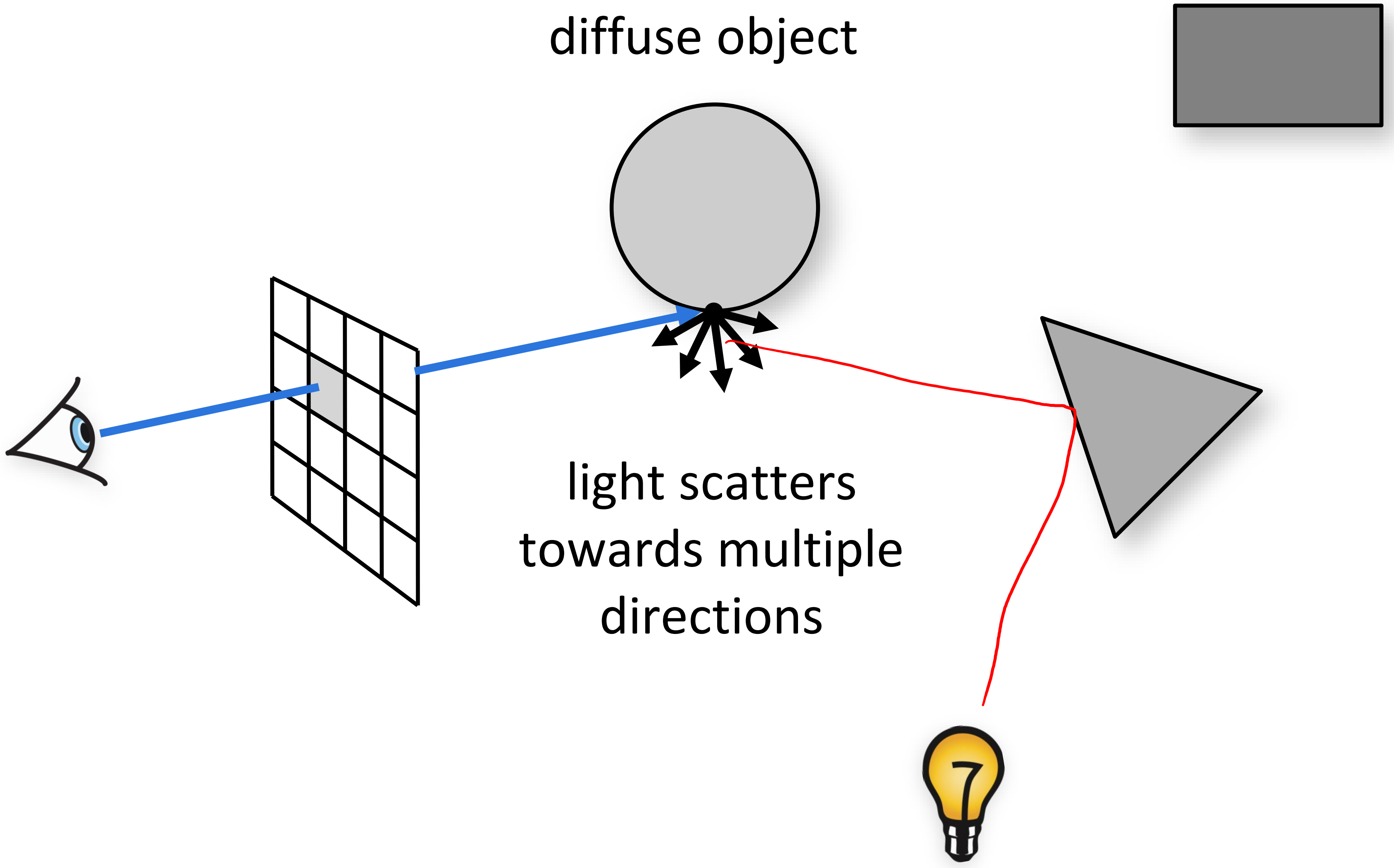
Ray Generation

↓

Intersection

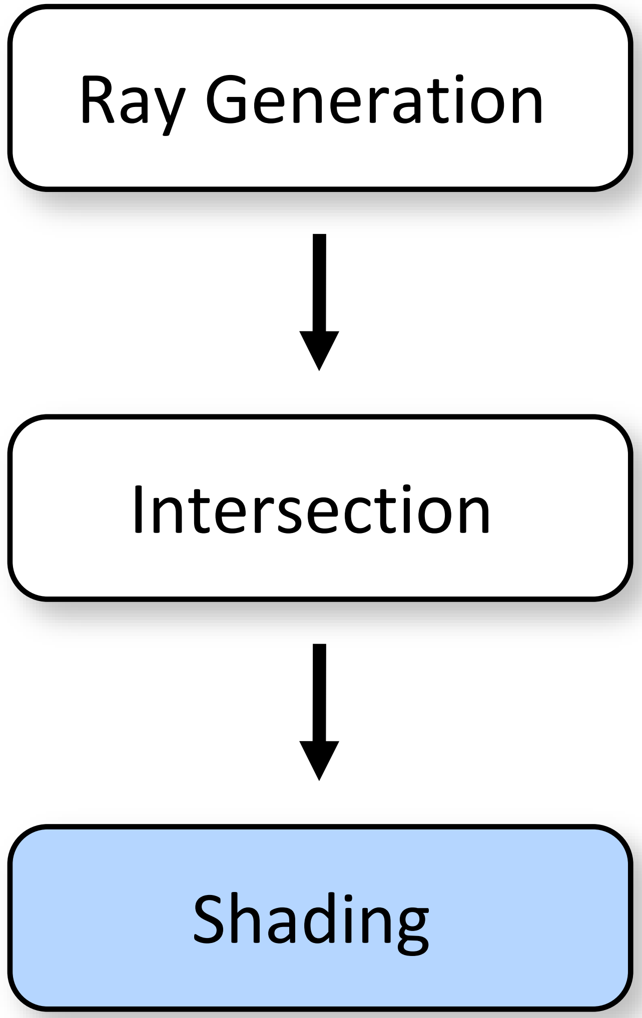
↓

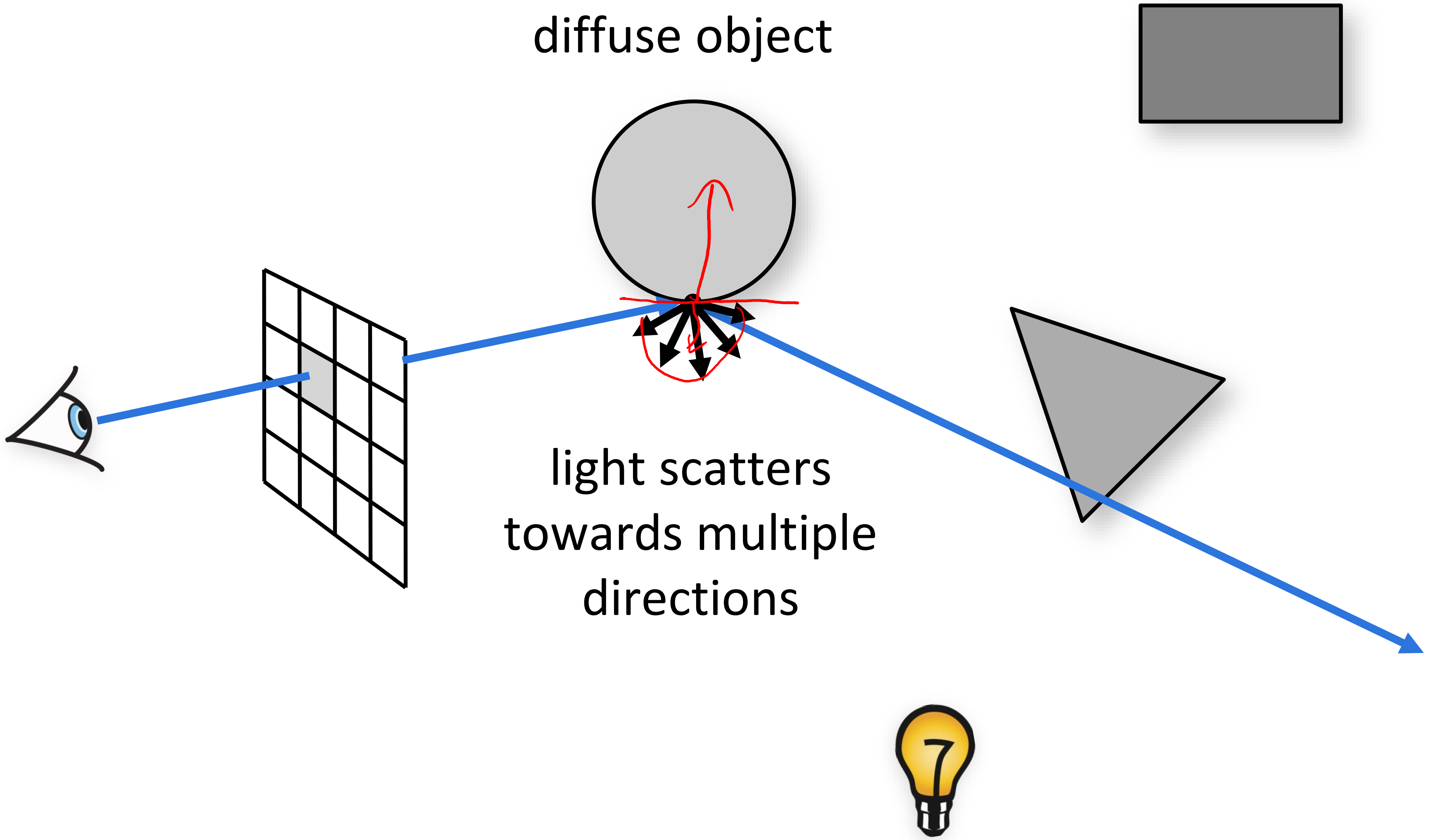
Shading



What direction should we trace a new ray towards?

Basic Ray Tracing Pipeline





What direction should we trace a new ray towards?

- Pick a direction at random!

Sampling Lambertian scattering

From what distribution should we sample directions?

Sampling Lambertian scattering

From what distribution should we sample directions?

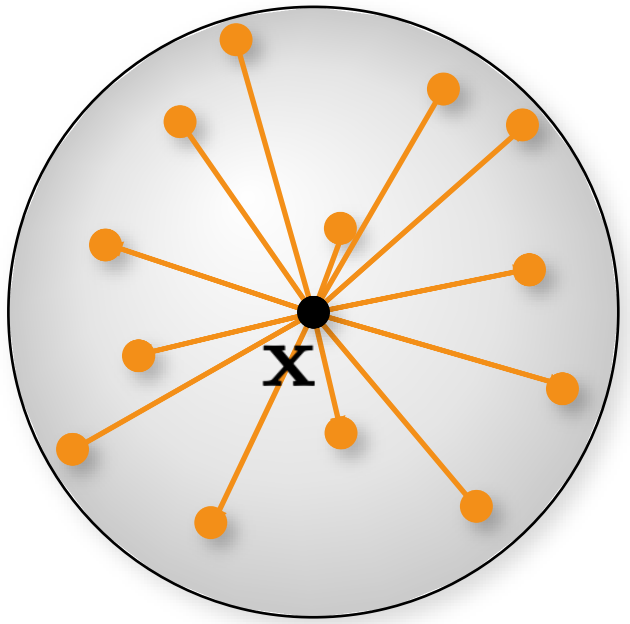
- Probability proportional to $\cos(\hat{n} \cdot \widehat{\omega})$.
- Even though BSDF scatters to all directions uniformly, we need to account for foreshortening.

How do we sample directions based on this *cosine-weighted* distribution?

Sampling Lambertian scattering

Generate points on sphere

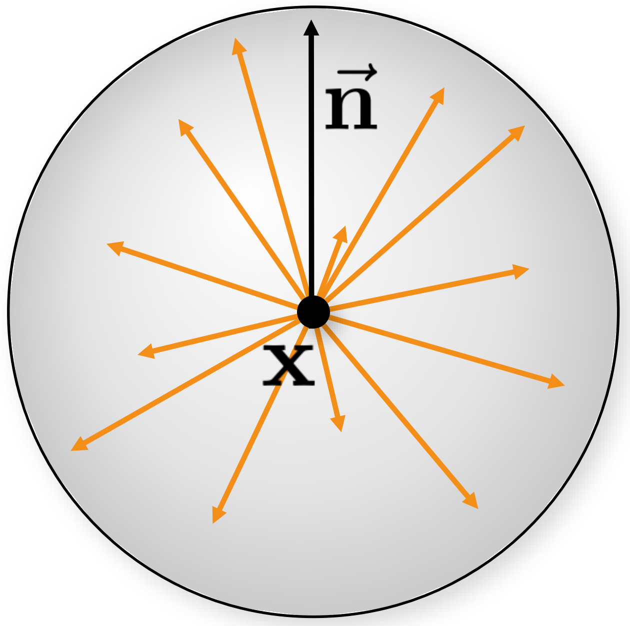
(unit directions)



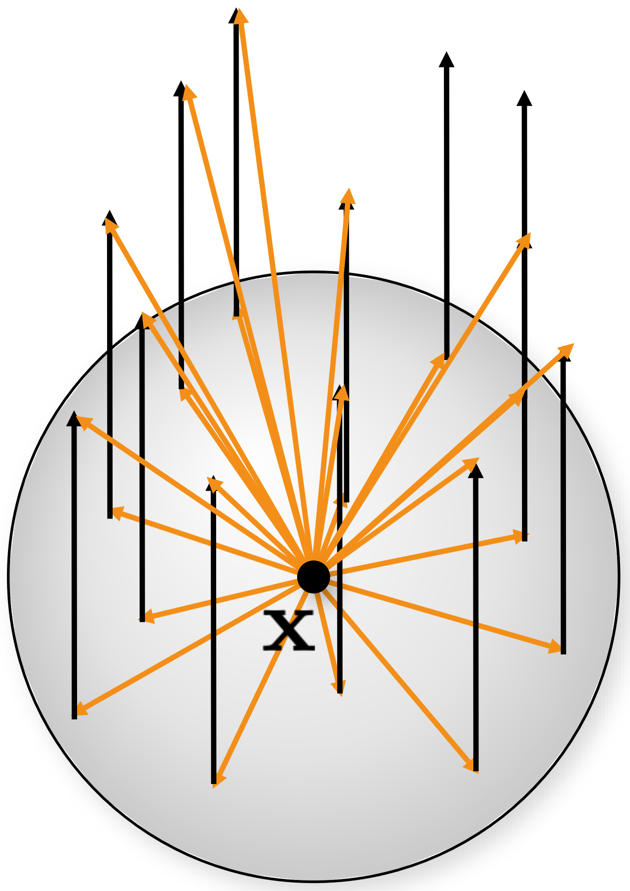
Sampling Lambertian scattering

Generate points on sphere
(unit directions)

unit normal



Add unit normal



Sampling Lambertian scattering

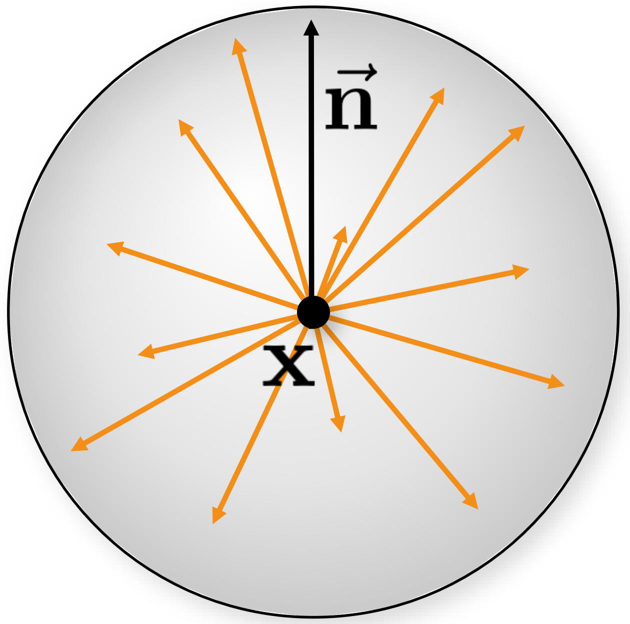
Generate points on sphere

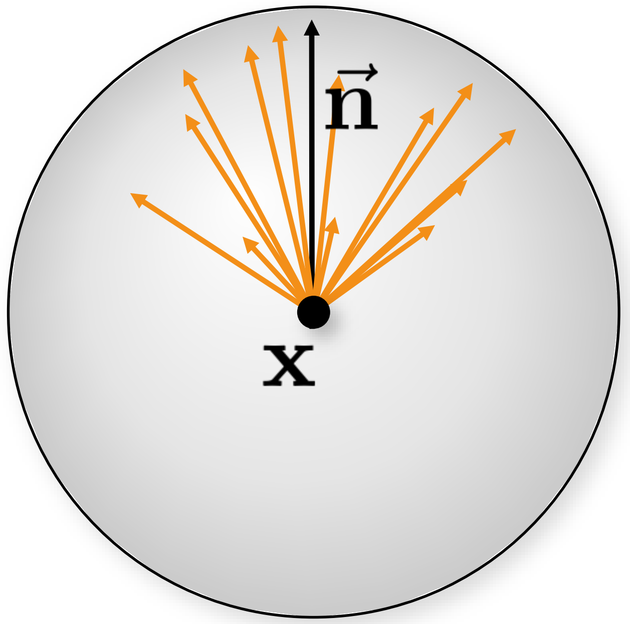
(unit directions)

Add unit normal

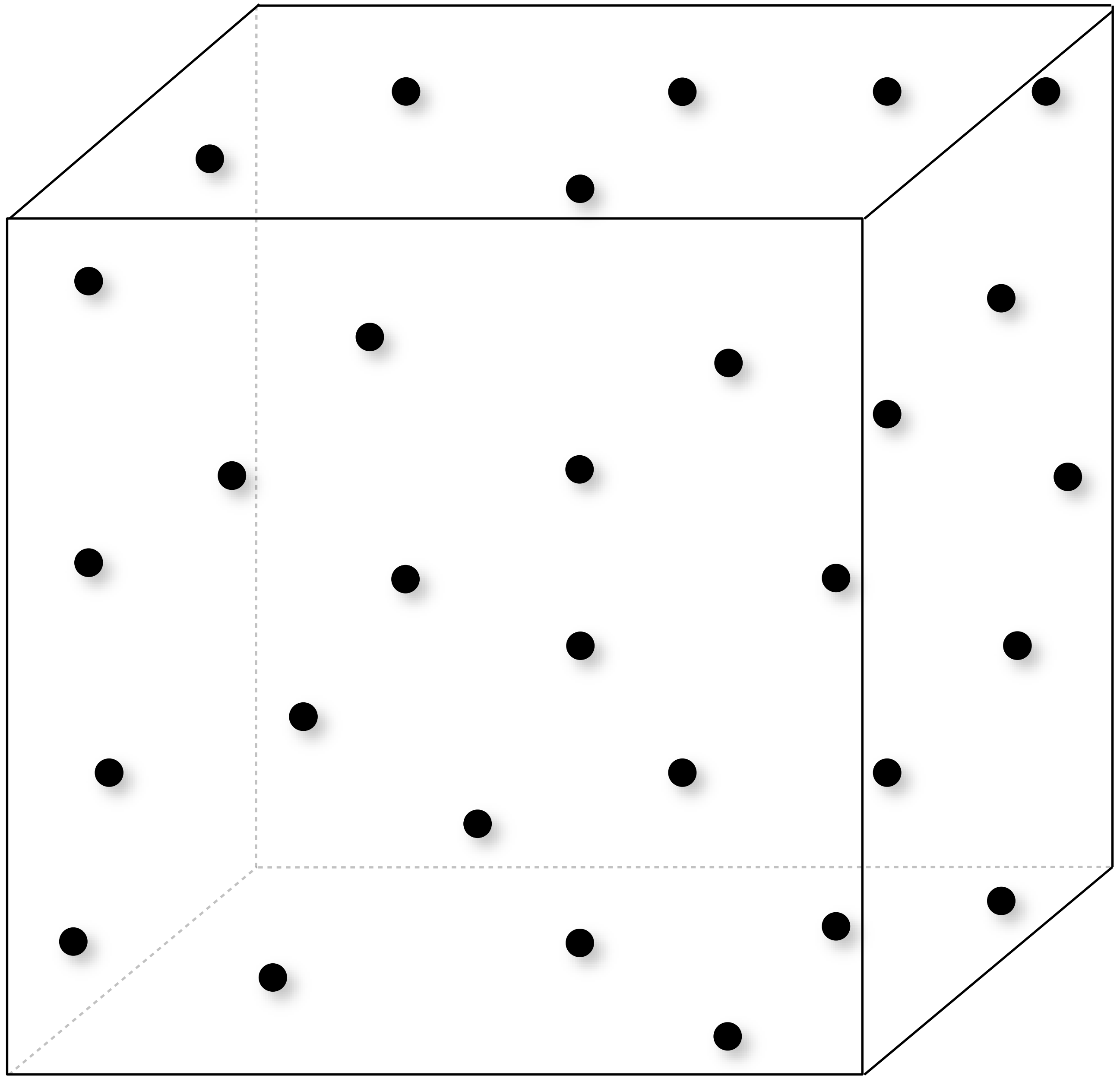
normalize

unit normal



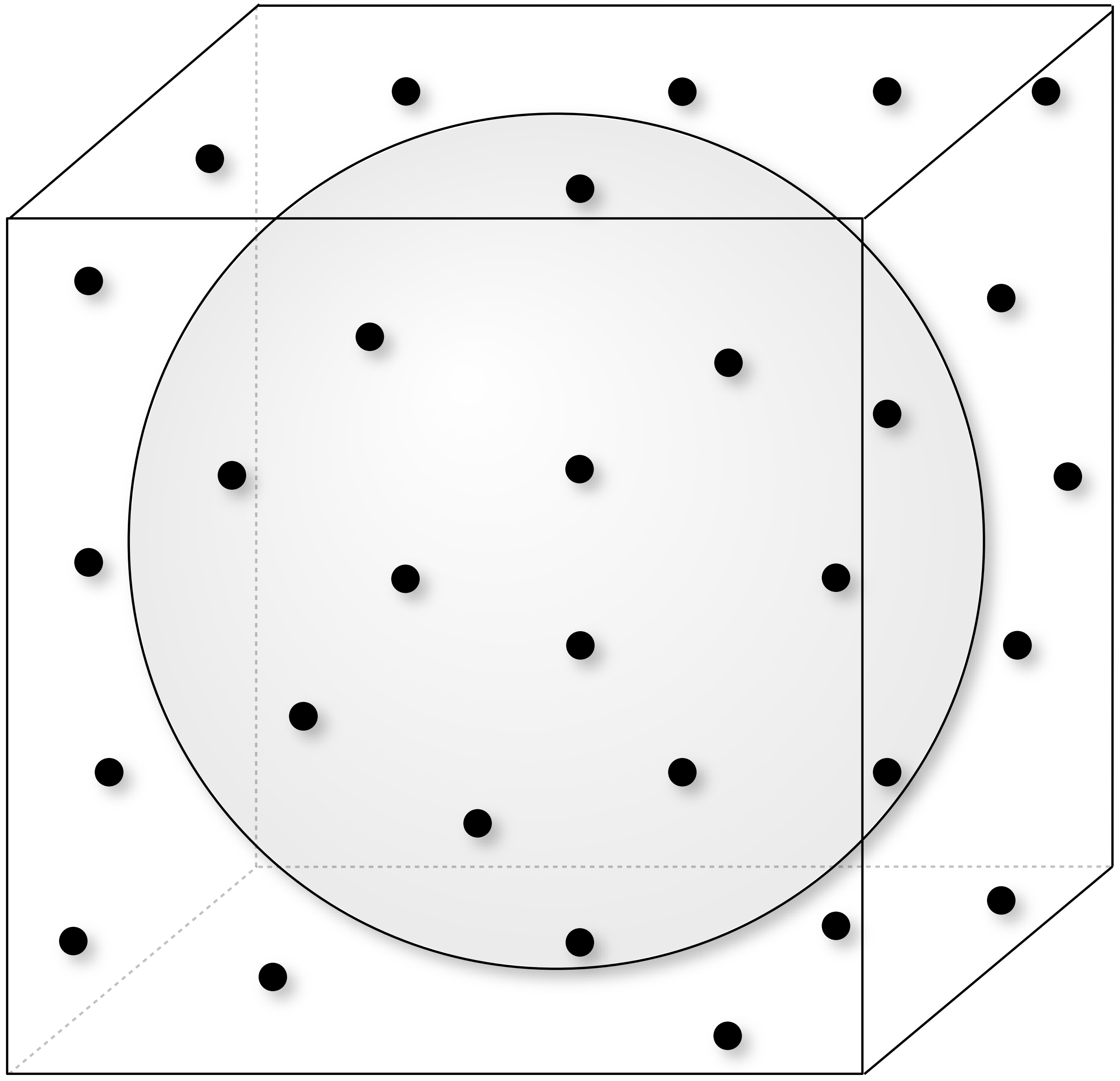


Rejection Sampling a Sphere



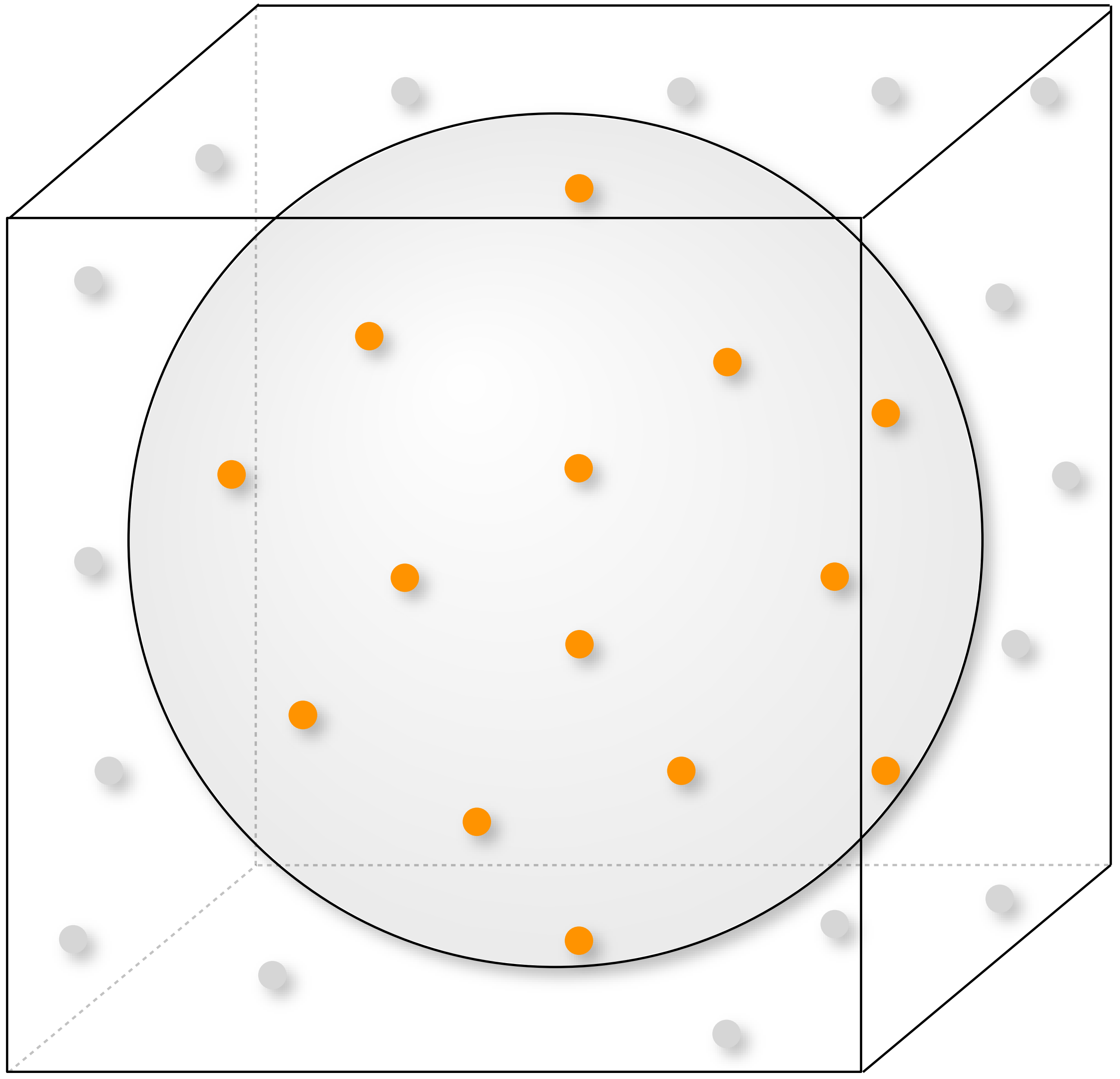
```
Vector3D v;


 v.x = 1-2*randf();
 v.y = 1-2*randf();
 v.z = 1-2*randf();
```

Rejection Sampling a Sphere



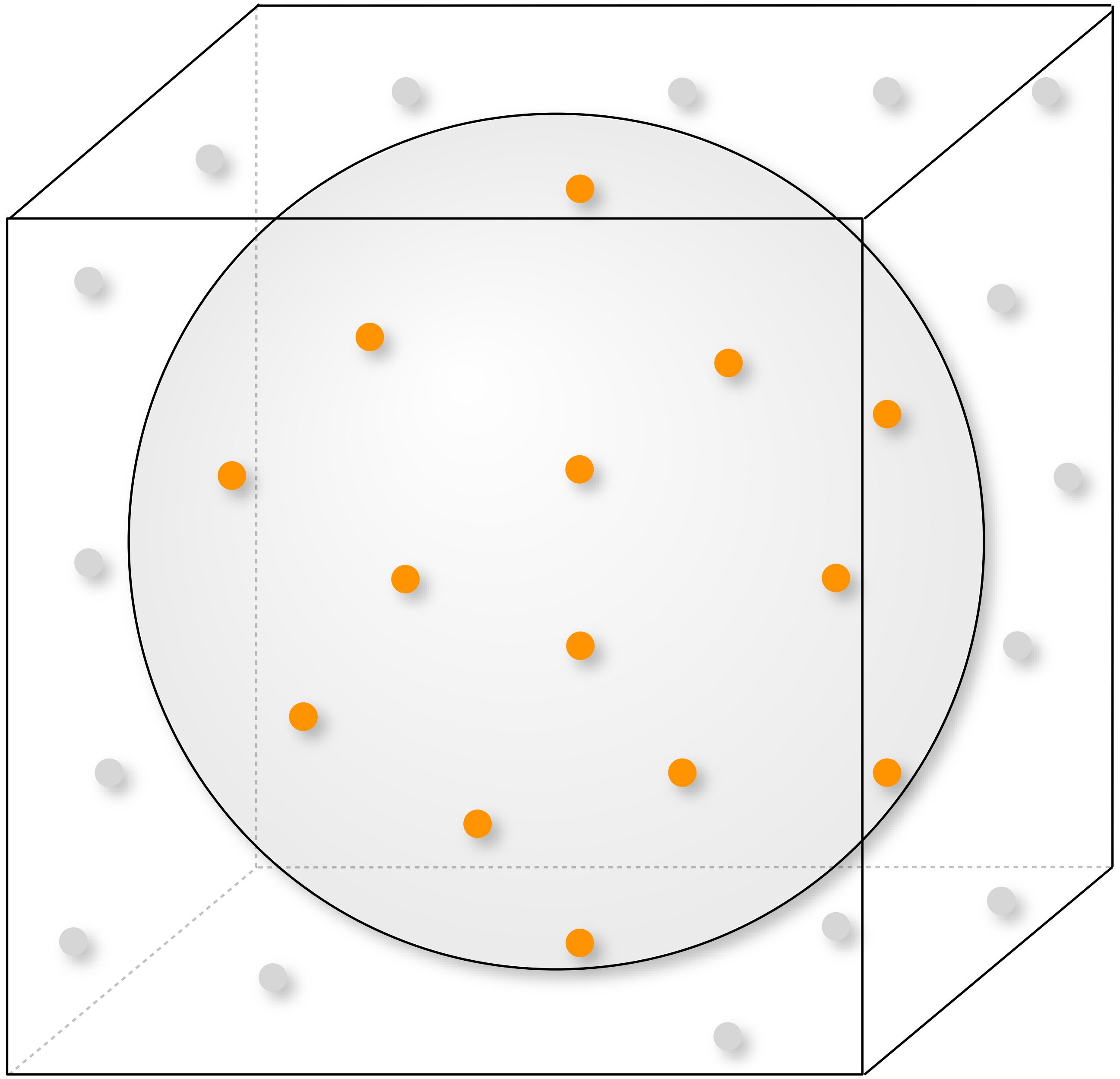
```
Vector3D v;
do
{
    v.x = 1-2*randf();
    v.y = 1-2*randf();
    v.z = 1-2*randf();
} while(v.length²() > 1)
```

Rejection Sampling a Sphere



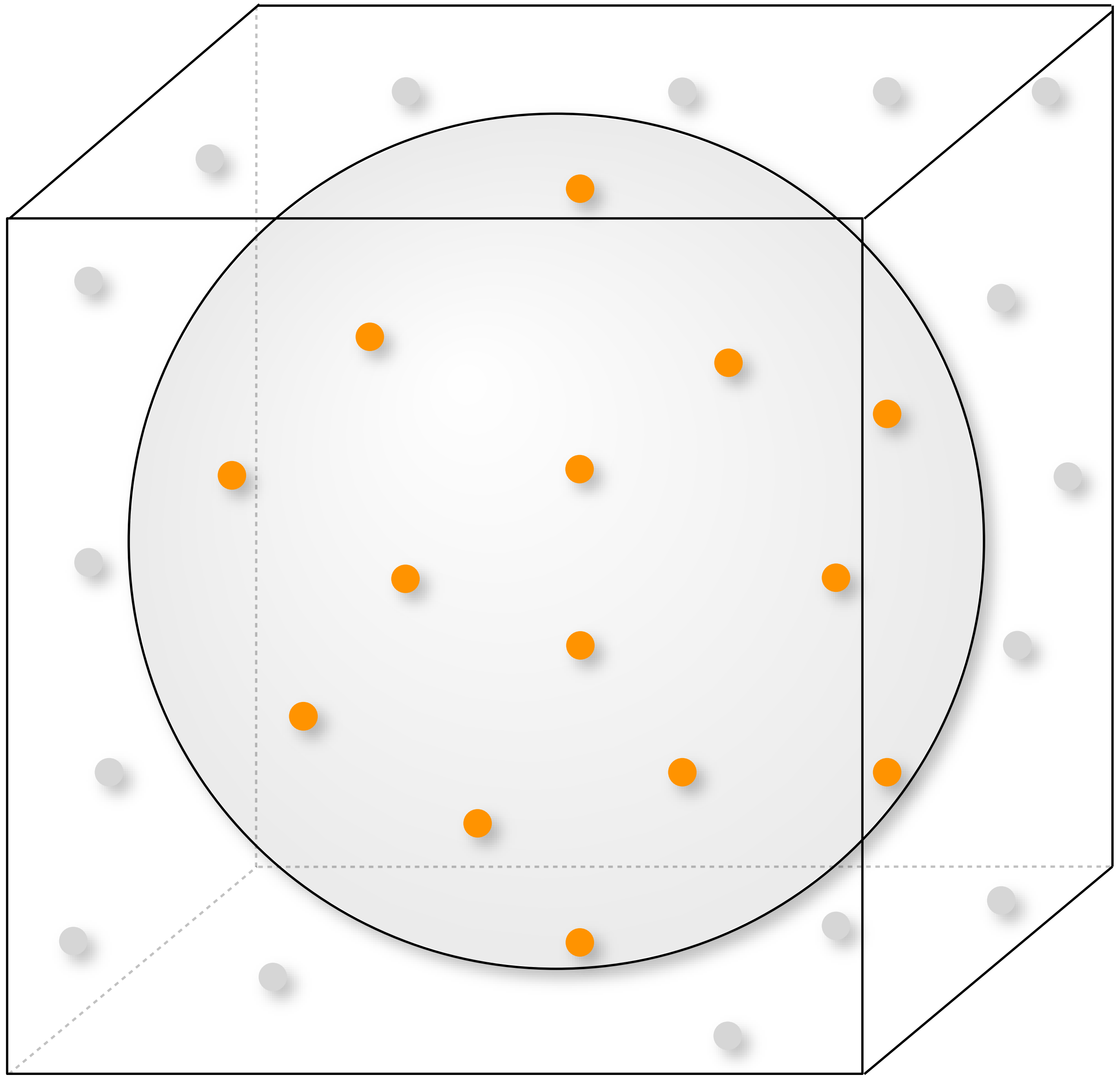
```
Vector3D v;
do
{
    v.x = 1-2*randf();
    v.y = 1-2*randf();
    v.z = 1-2*randf();
} while(v.length²() > 1)
```

Rejection Sampling a Sphere



```
Vector3D v;
do
{
    v.x = 1-2*randf();
    v.y = 1-2*randf();
    v.z = 1-2*randf();
} while(v.length²() > 1)
// Project onto sphere
v /= v.length();
```

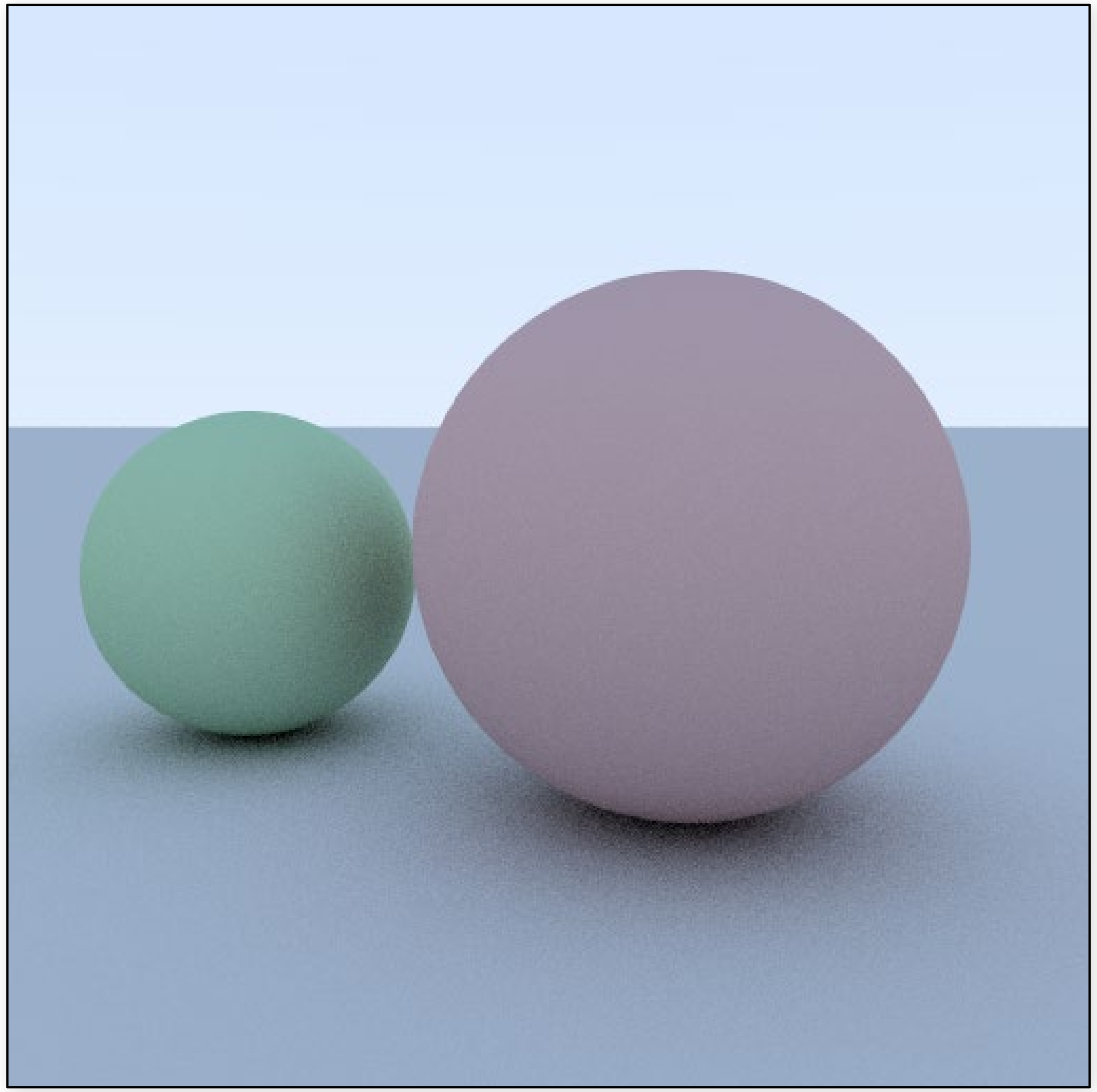
Sampling a Sphere using normal samples



```
Vector3D v;
v.x = randnf();
v.y = randnf();
v.z = randnf();
// Project onto sphere
v /= v.length();
```

- No rejection sampling required (no `while` loop).
- Need to use normal, rather than uniform, samples.

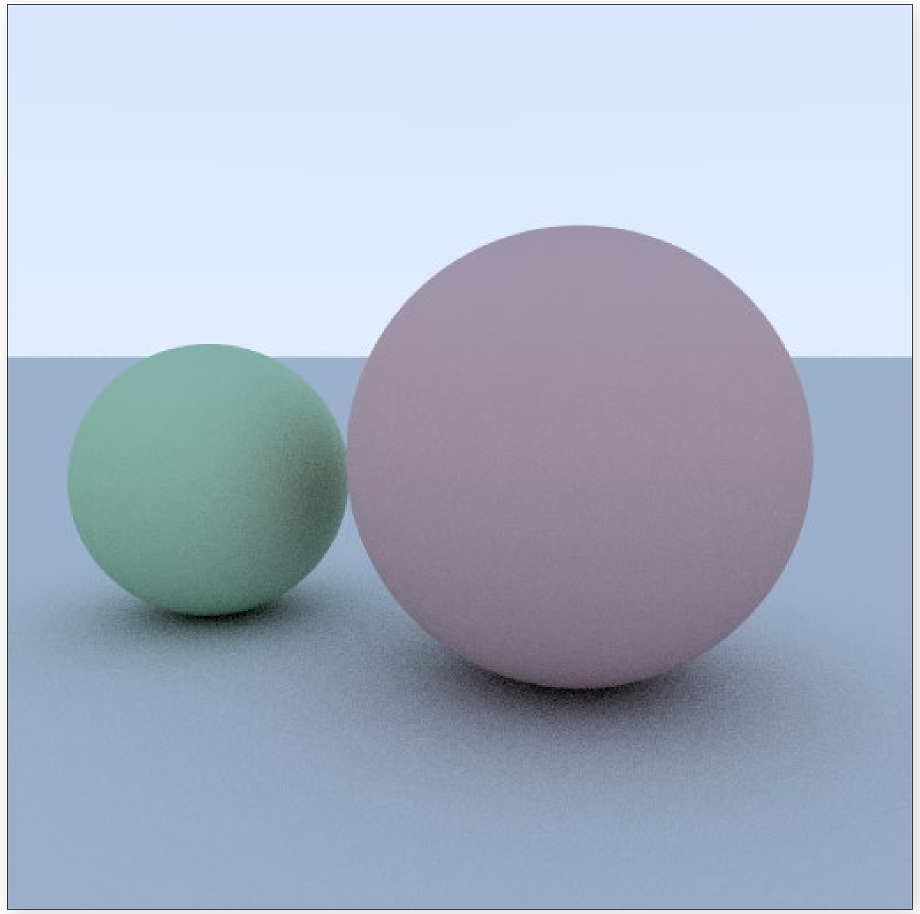
Diffuse shading



After a slide by Steve Marschner

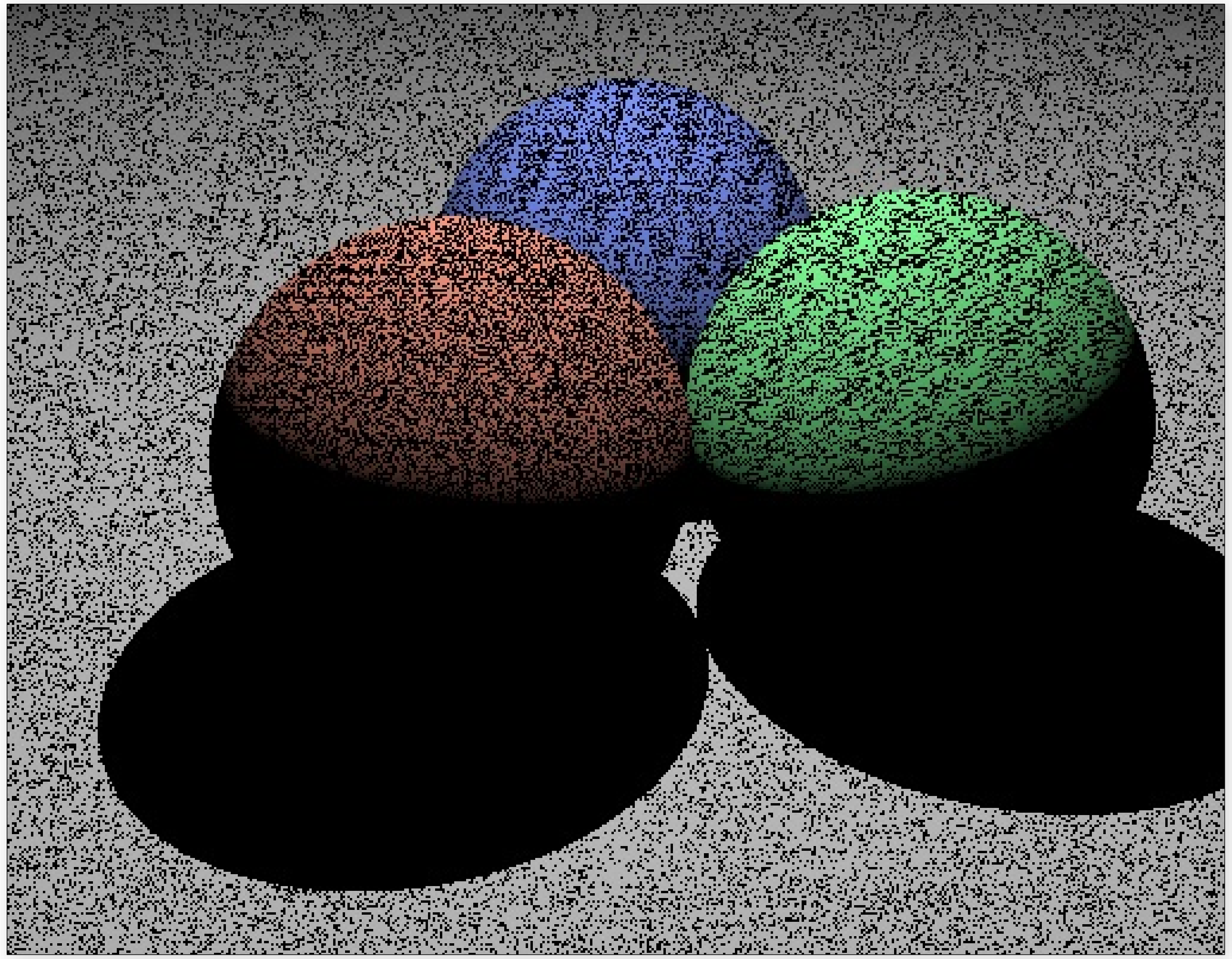
Image so far

```
Scene::trace(Ray ray)
   hit = surfaces.intersect(ray);
   if hit
      [col, sRay] = hit->mat->scatter(ray)
      return col * trace(sRay);
   else
      return backgroundColor;
```



Rounding errors

Don't fall victim to one of the classic blunders:



affectionately called
"shadow acne"

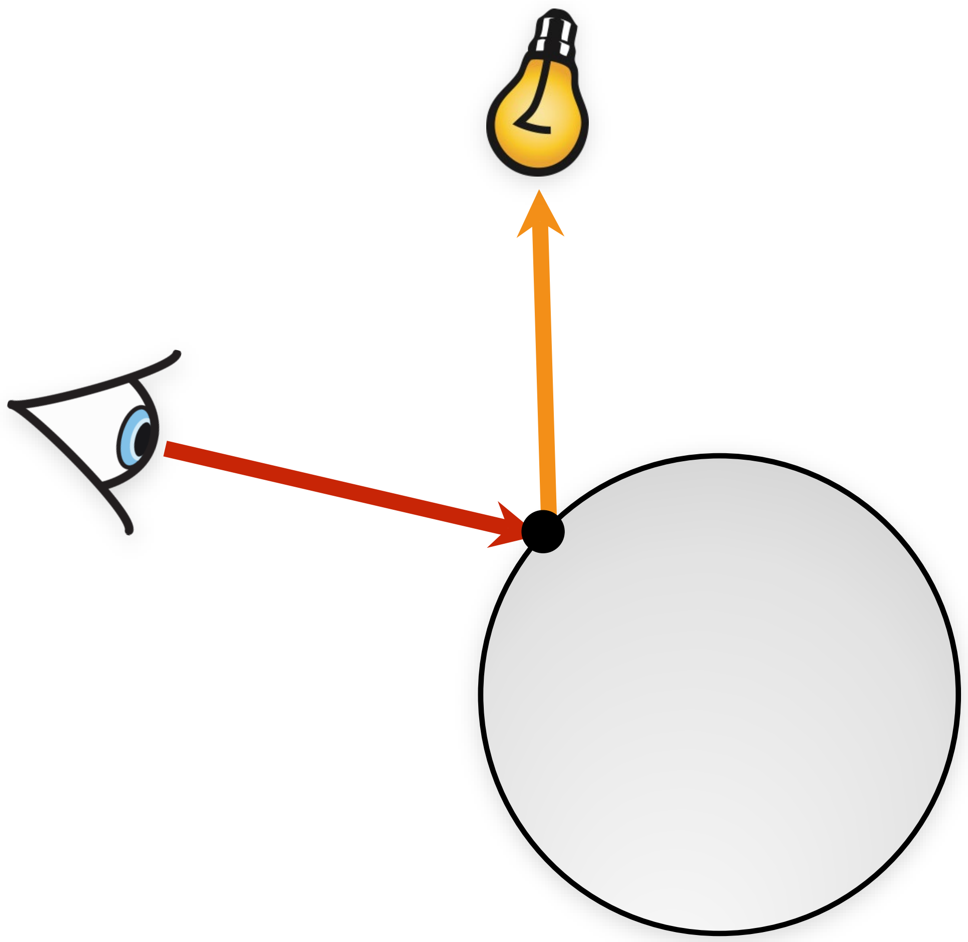
What's going on?

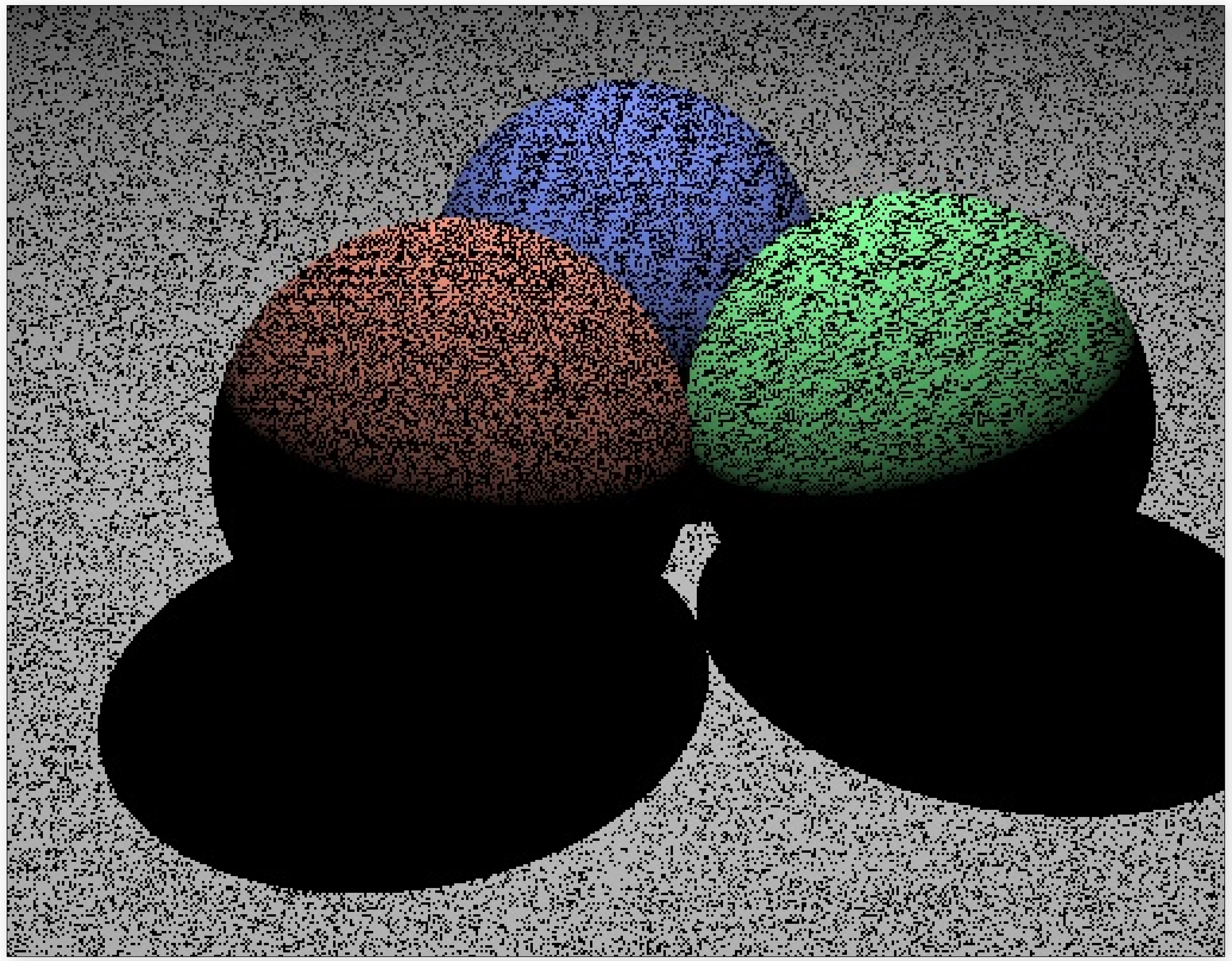
- hint: at what *t* does a recursive ray intersect the surface you're shading?

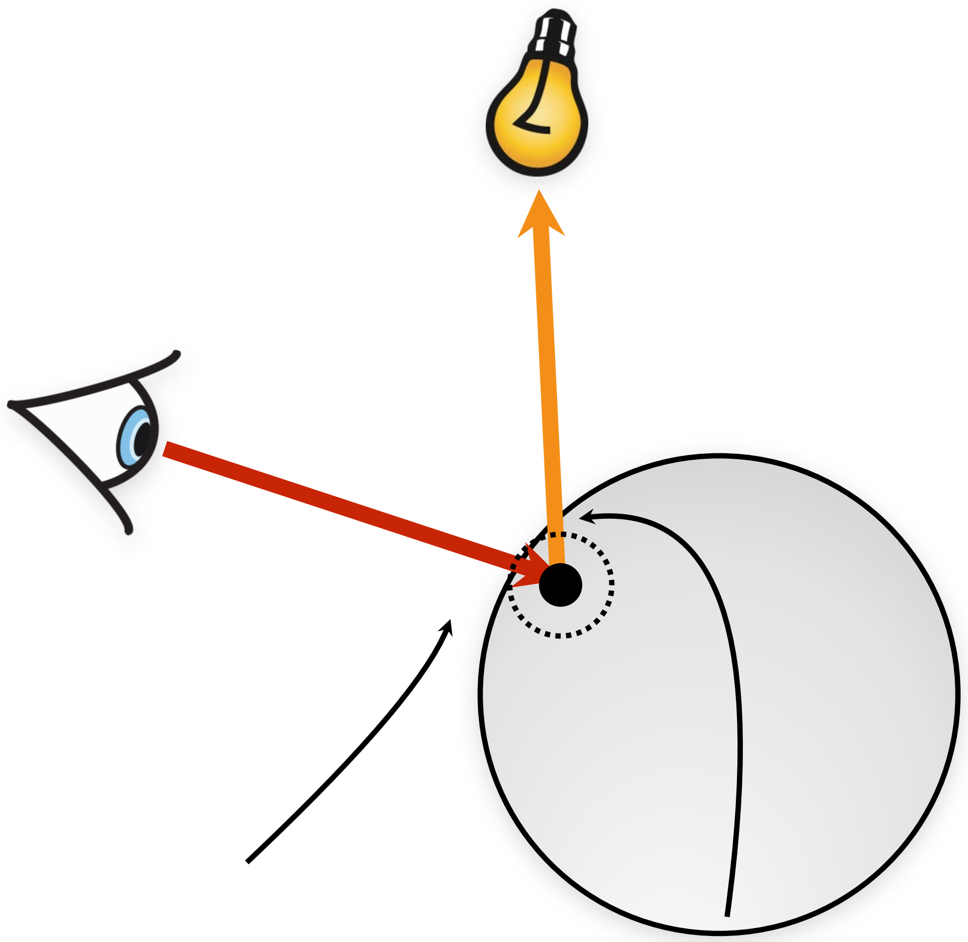
After a slide by Steve Marschner

Rounding errors

Don't fall victim to one of the classic blunders:







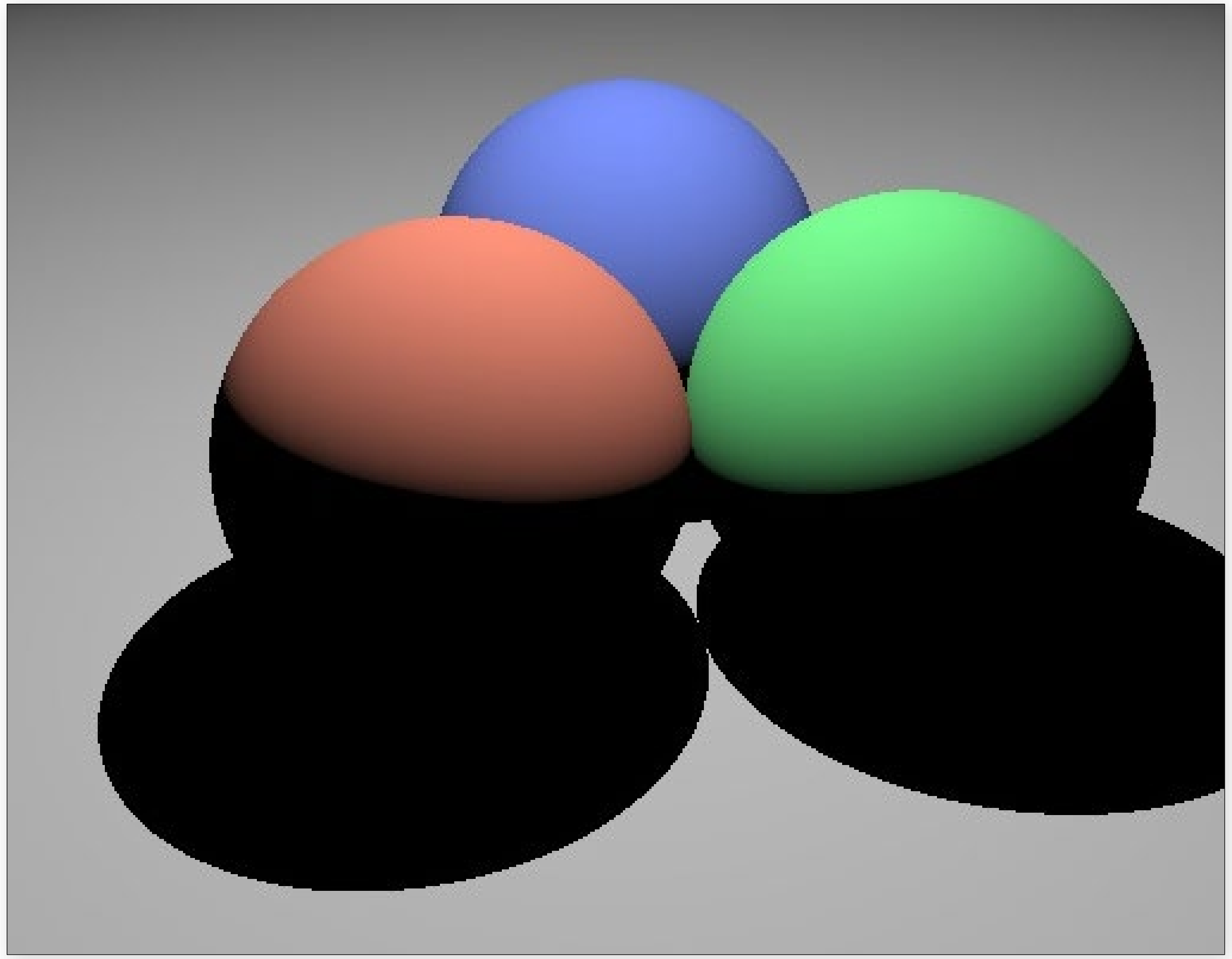
intersection inside due to floating point error

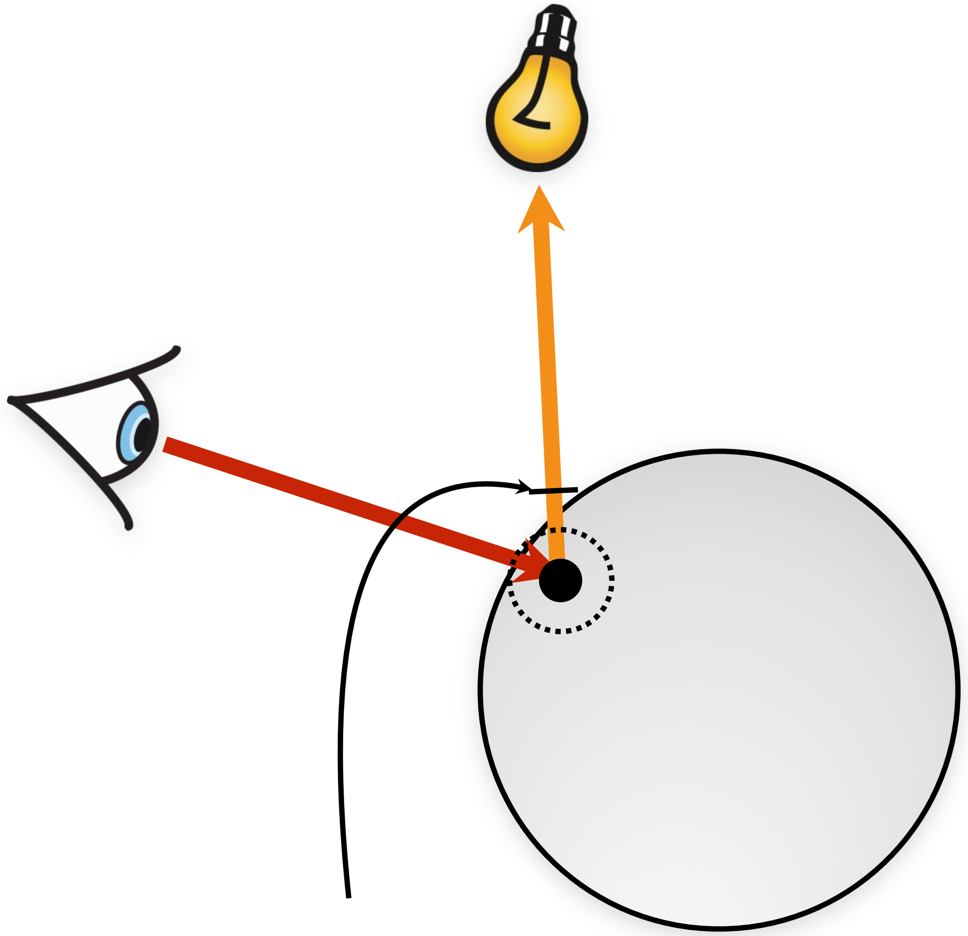
ray blocked by self-intersection

After a slide by Steve Marschner

Shadow rounding errors

Solution: recursive rays start a tiny distance from the surface

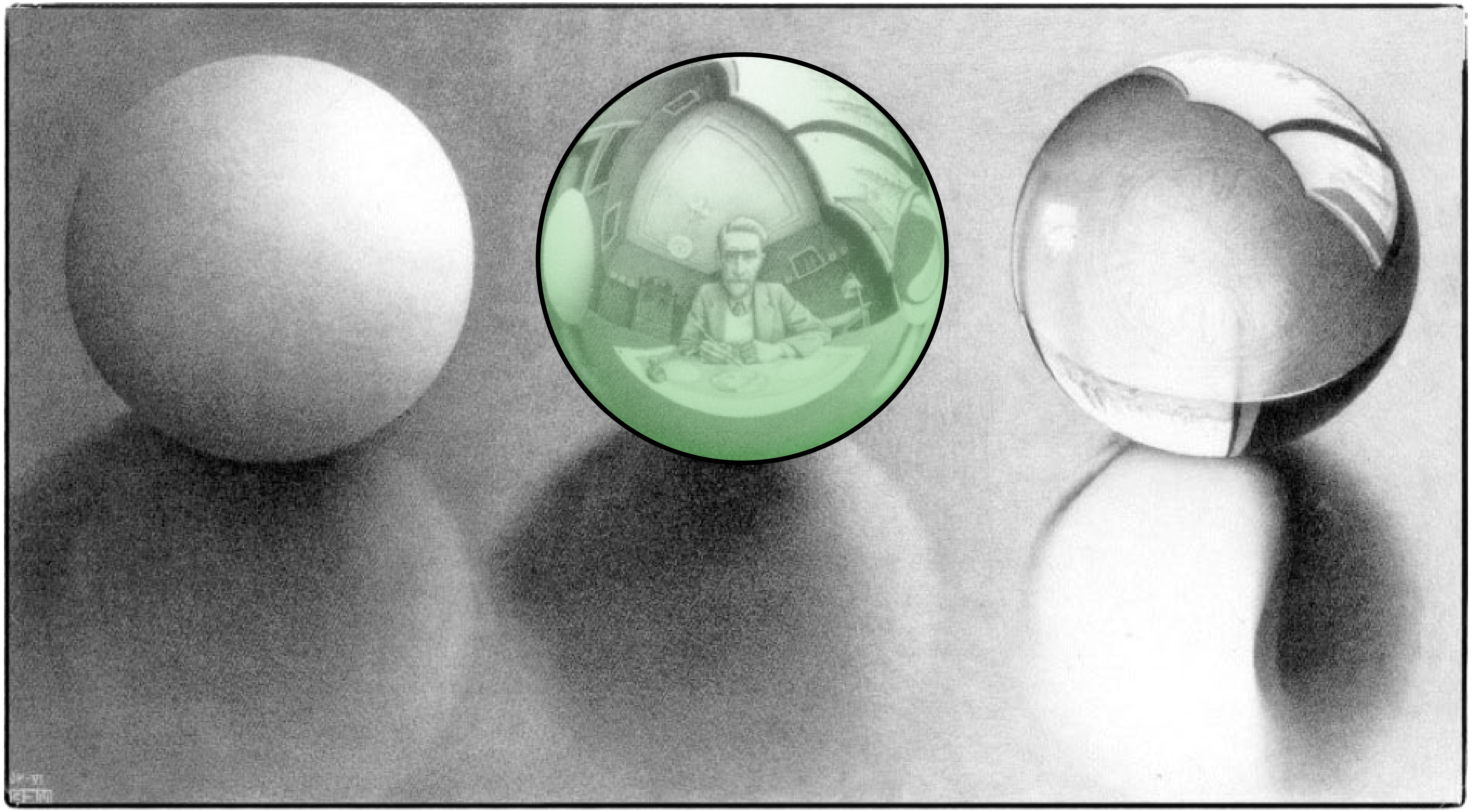




Do this by limiting the *t* range

After a slide by Steve Marschner

Specular/Mirror reflection



Mirror reflection

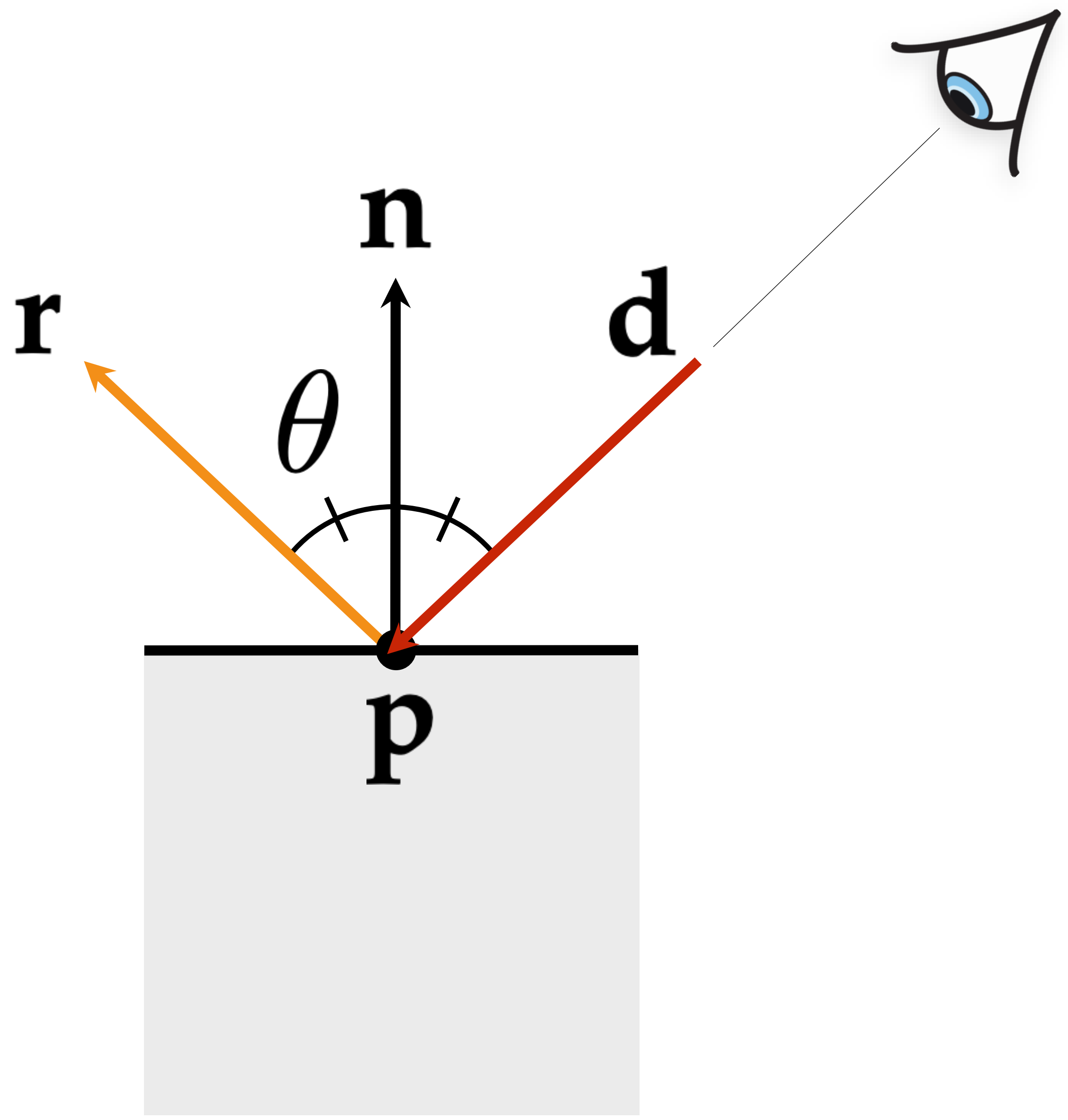
Evaluated by tracing a new ray:

$$L_r = k_r\,\textbf{trace}(\mathbf{p}, \mathbf{r})$$

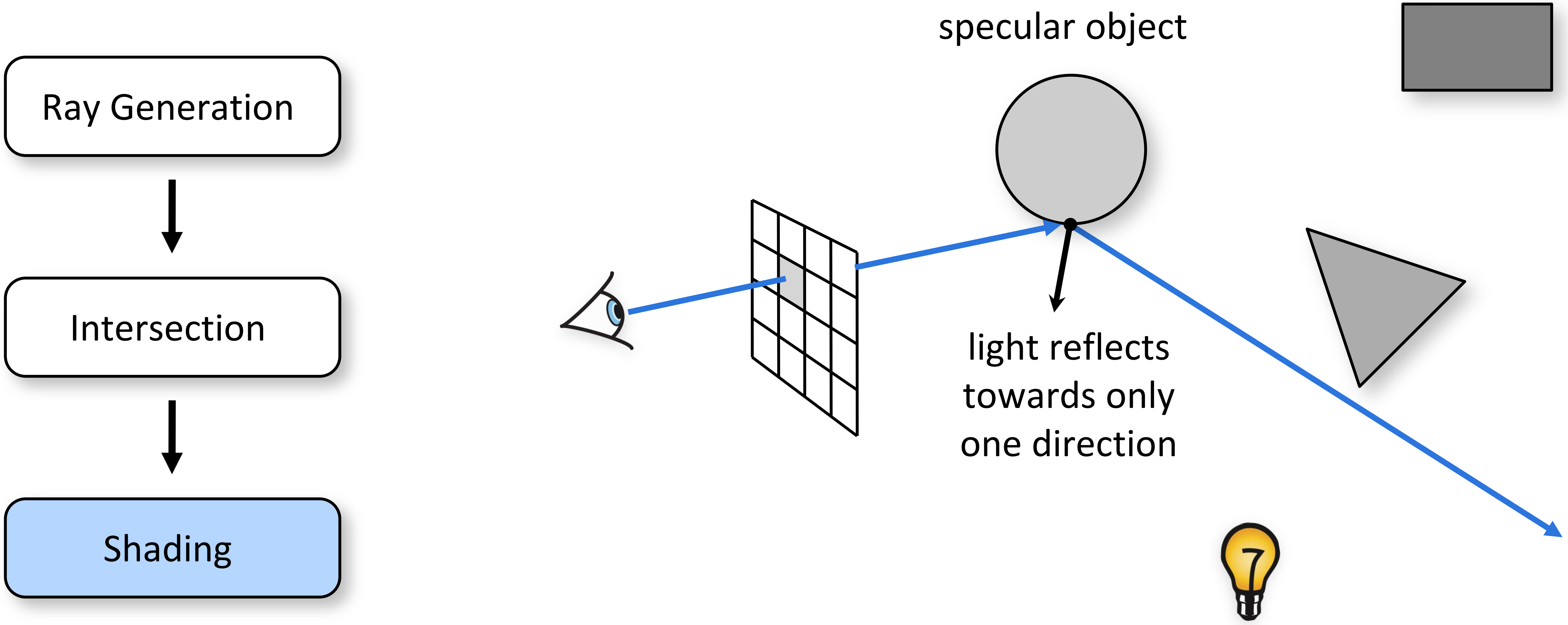
reflected light

reflection scaled down/tinted

recursive call



Basic Ray Tracing Pipeline



What direction should we trace a new ray towards?

- Just use law of mirror reflection, no need for random selection!

Mirror reflection

Consider perfectly shiny surface

- there’s a reflection of other objects

Can render this using recursive ray tracing

- to find out mirror reflection color ask:
  “what color is seen from surface point in reflection direction?”

Mirror reflection

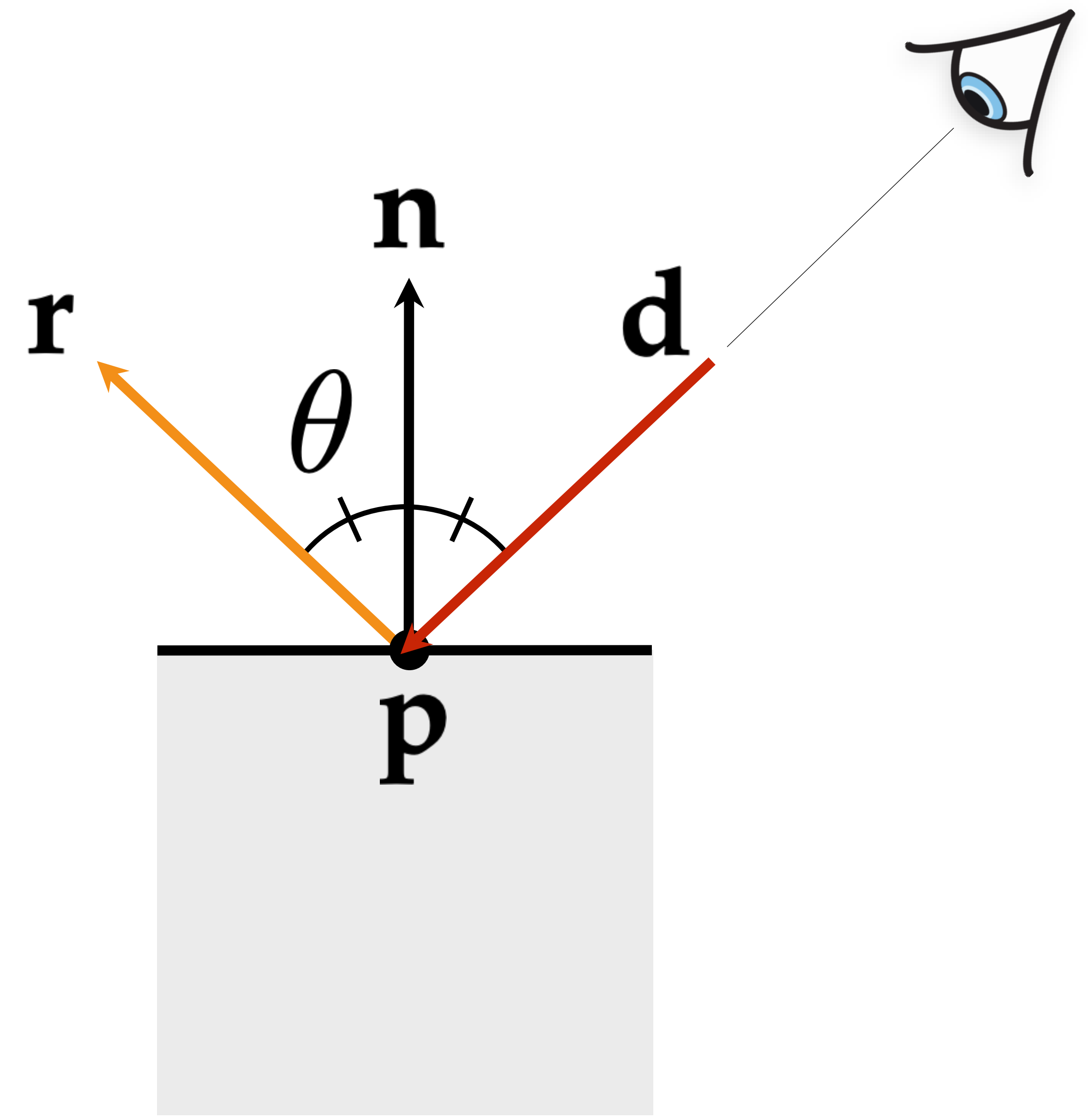
Evaluated by tracing a new ray:

$$L_r = k_r \mathrm{trace}(\mathbf{p}, \mathbf{r})$$

reflected light

reflection scaled down/tinted

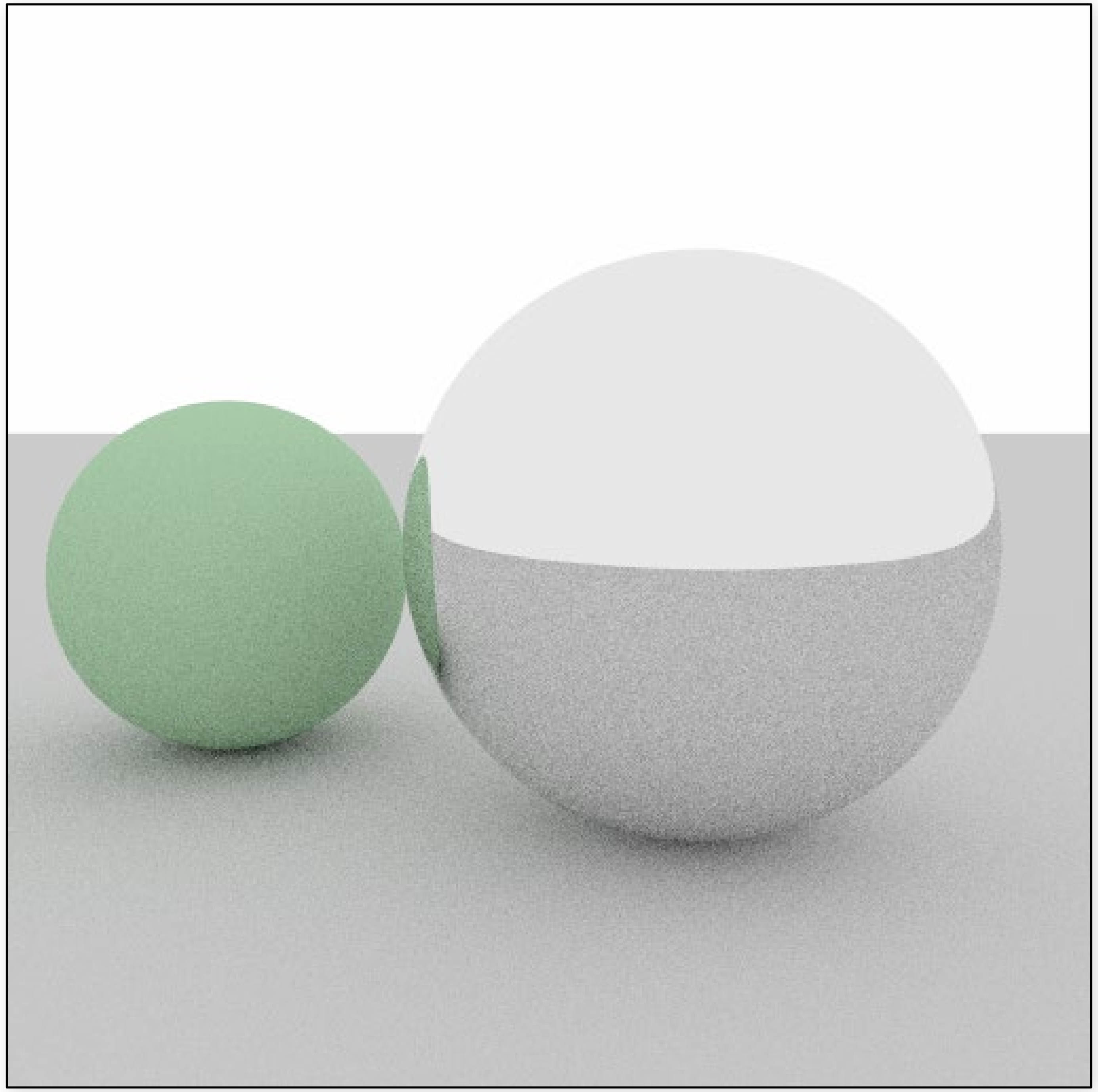
recursive call



Implementation details:

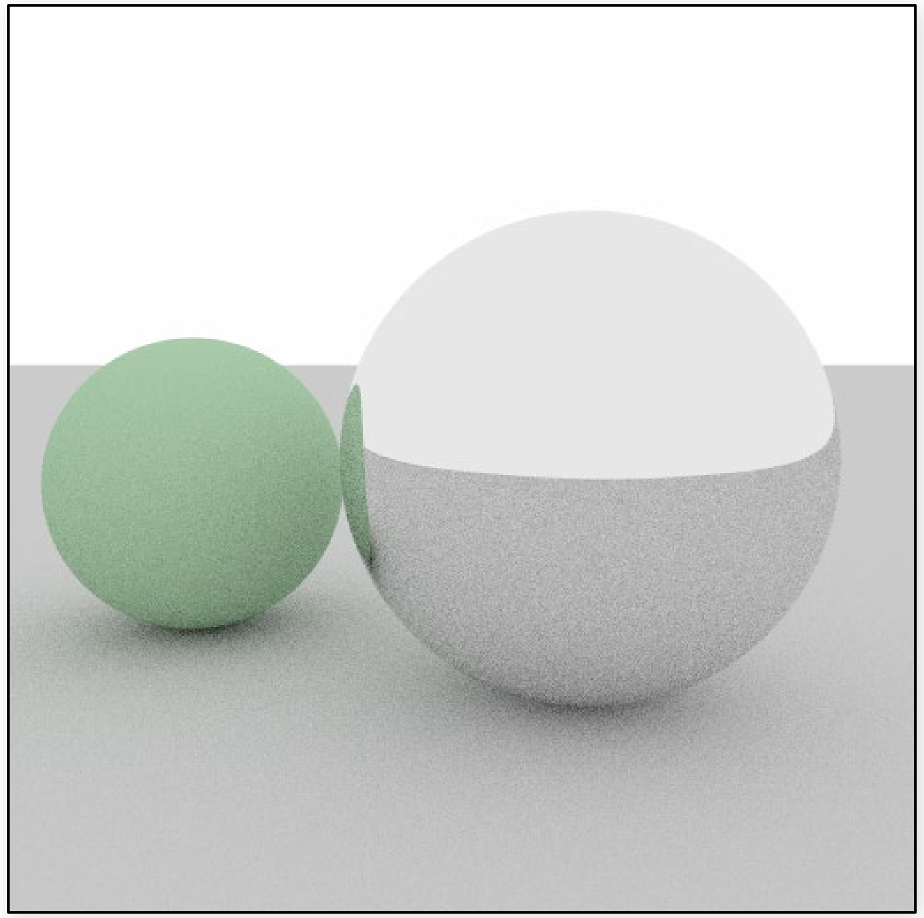
- don't self-intersect ($t_{\min} > \epsilon$)
- don't recurse indefinitely

Diffuse & mirror spheres



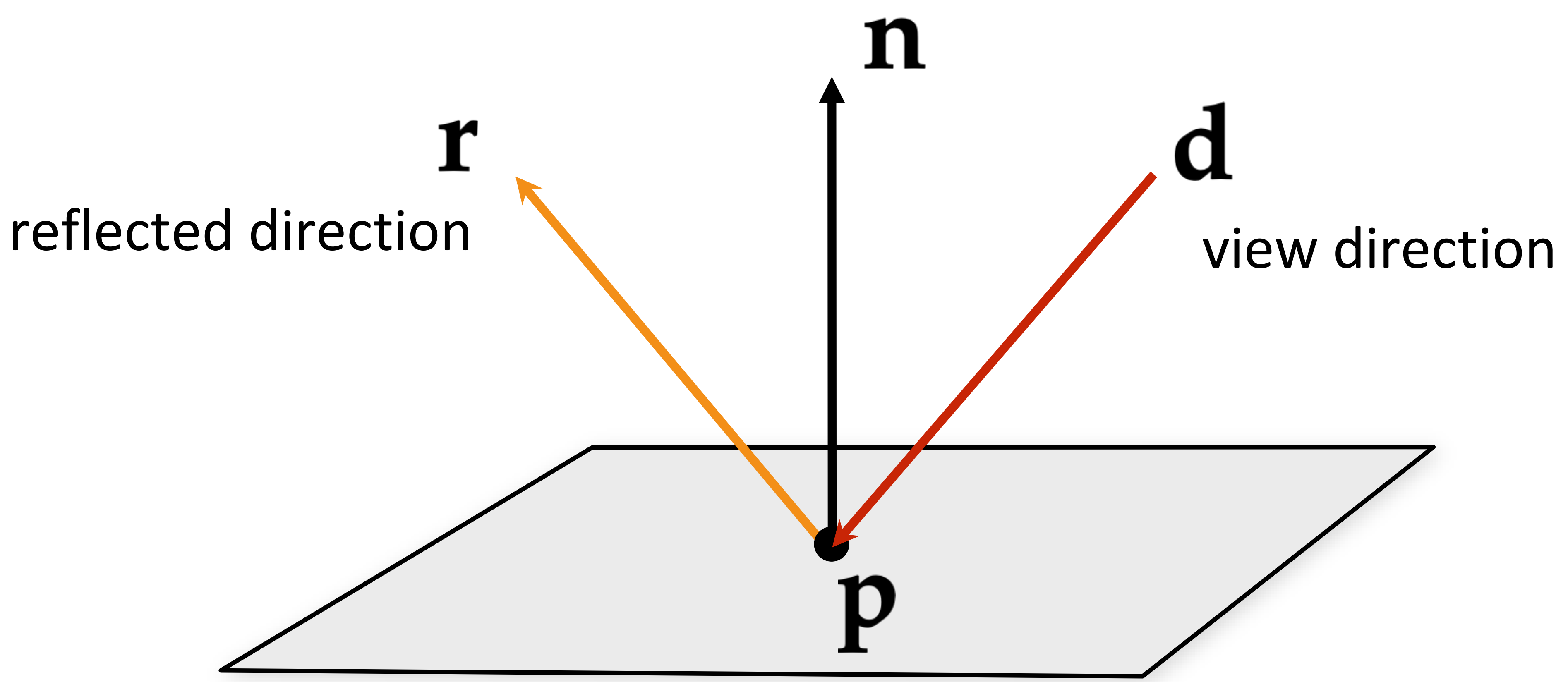
Same pseudo-code

```
Scene::trace(Ray ray)
   hit = surfaces.intersect(ray);
   if hit
      [col, sRay] = hit->mat->scatter(ray)
      return col * trace(sRay);
   else
      return backgroundColor;
```



Mirror reflection

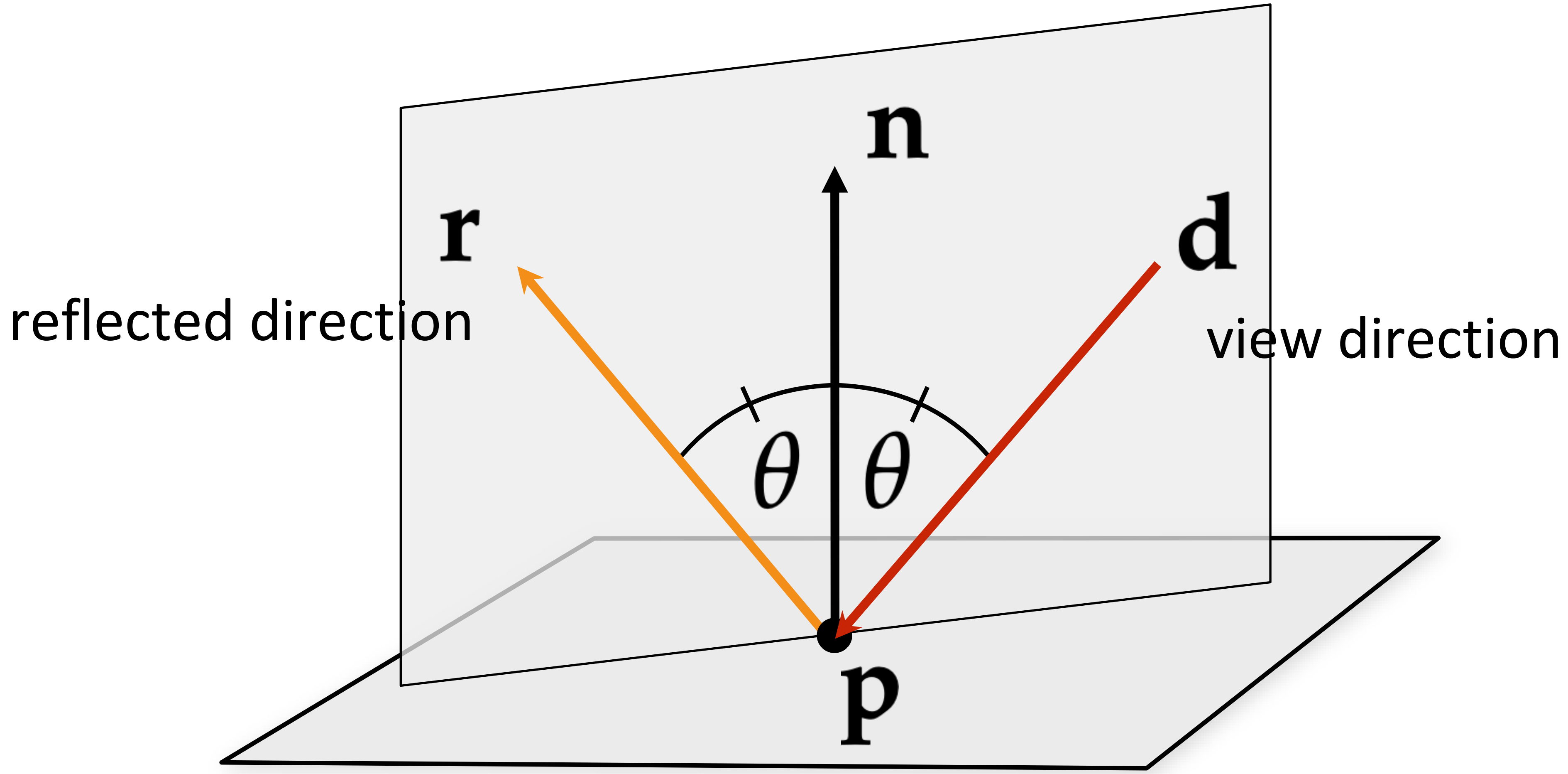
Assume $\mathbf{n}$ is unit length



What <u>two</u> properties defined reflection direction?

Mirror reflection

Assume $\mathbf{n}$ is unit length

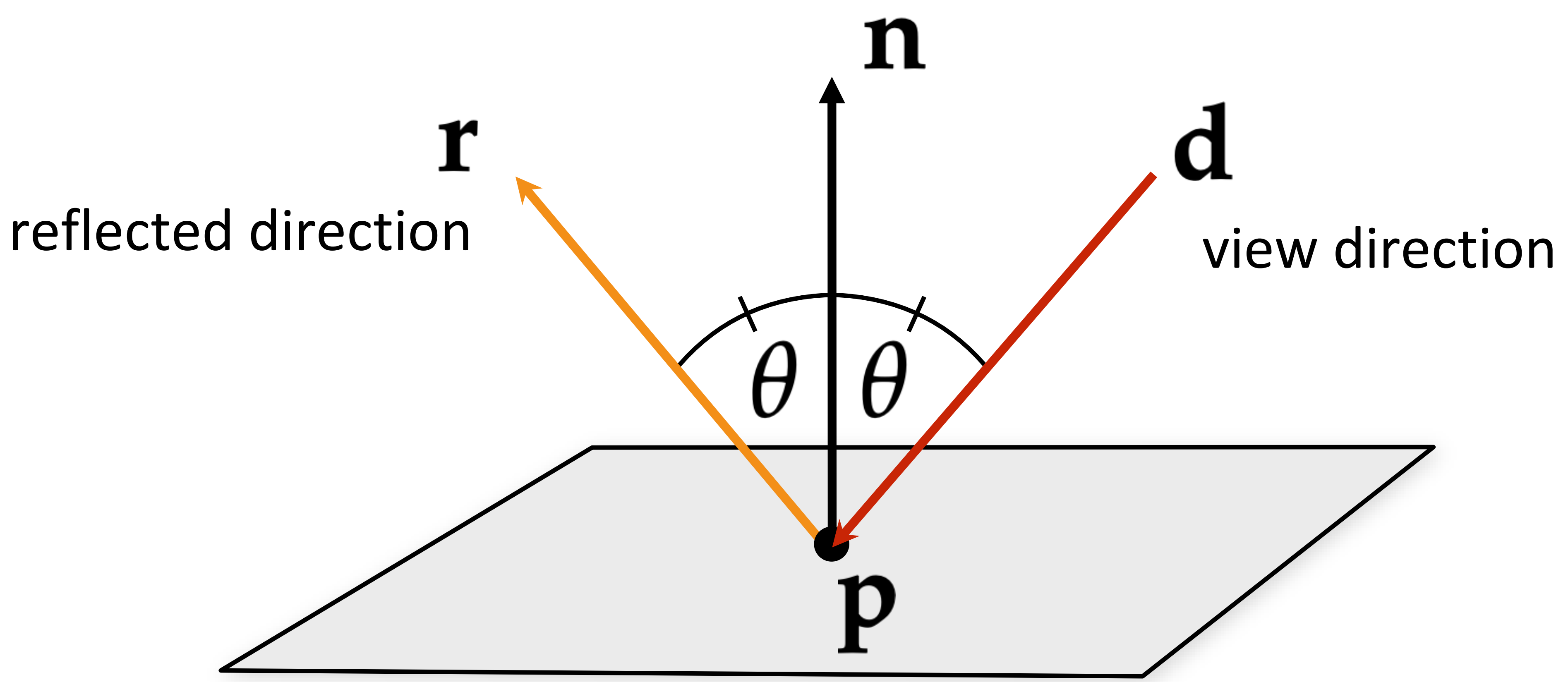


What <u>two</u> properties defined reflection direction?

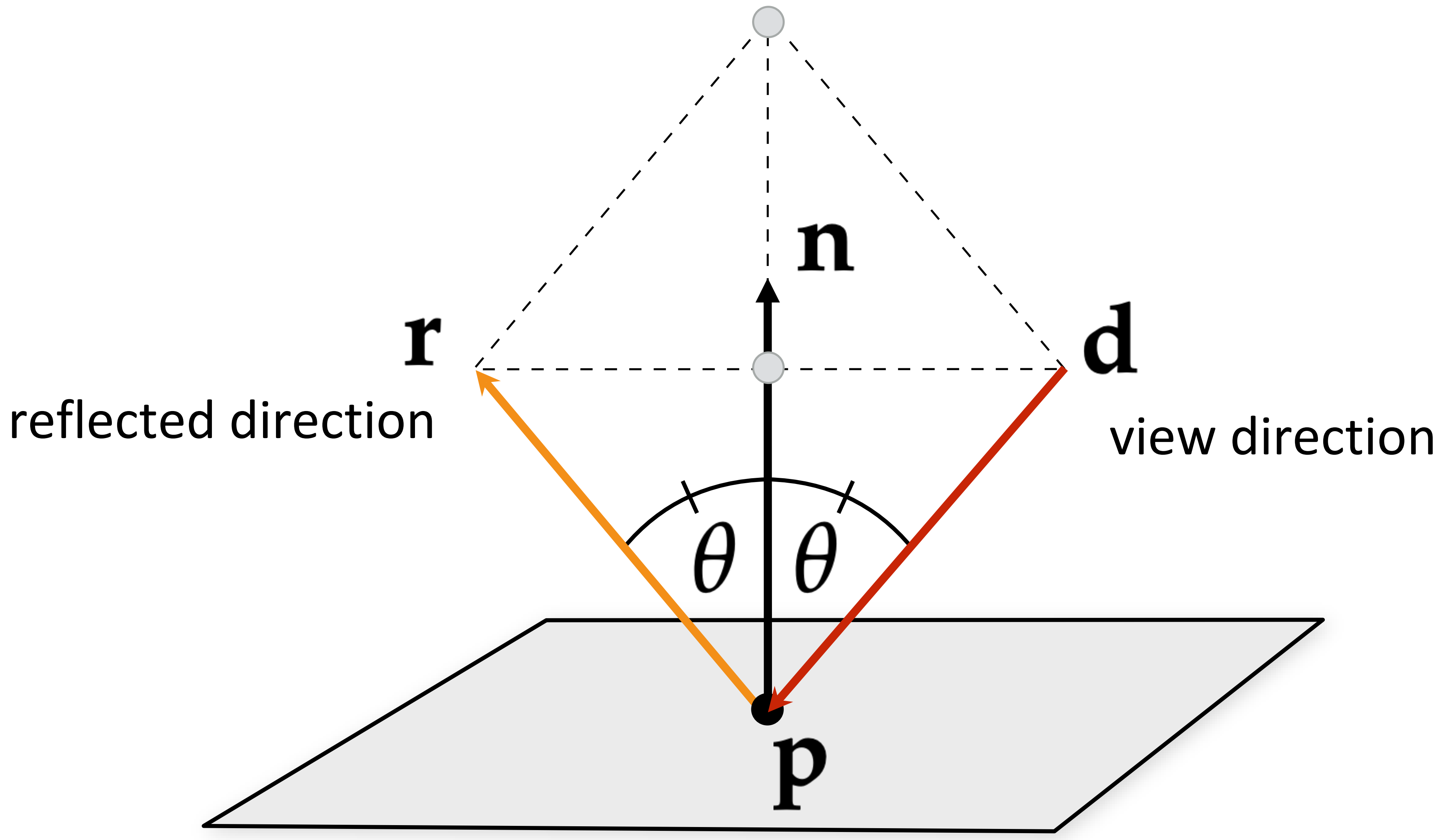
- co-planar view direction, reflected direction, and normal direction
- equal angles between normal-view directions, and normal-reflected directions

Mirror reflection

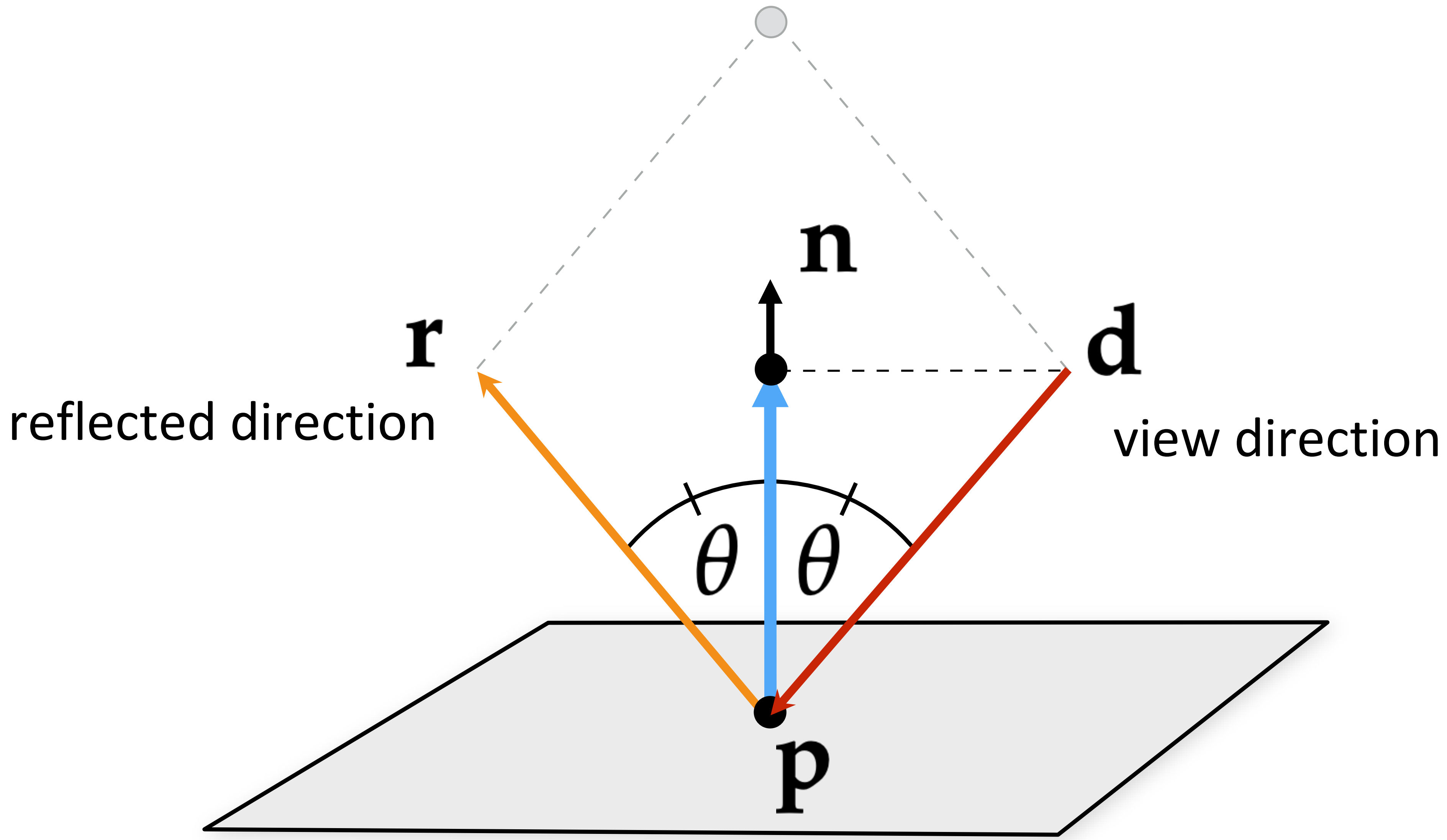
Assume $\mathbf{n}$ is unit length



Mirror reflection

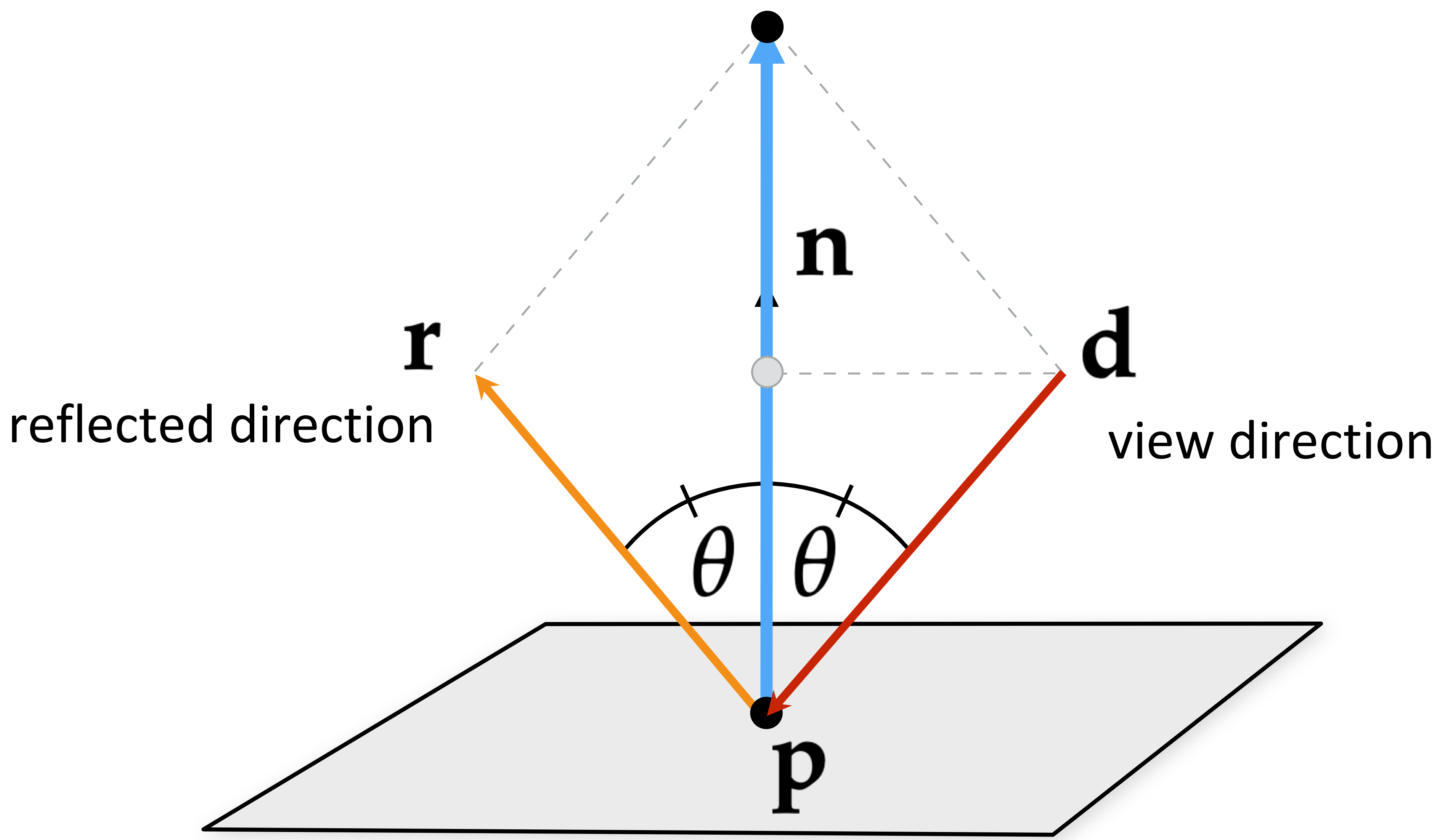


Mirror reflection



$$-\mathbf{n}(\mathbf{n} \cdot \mathbf{d})$$

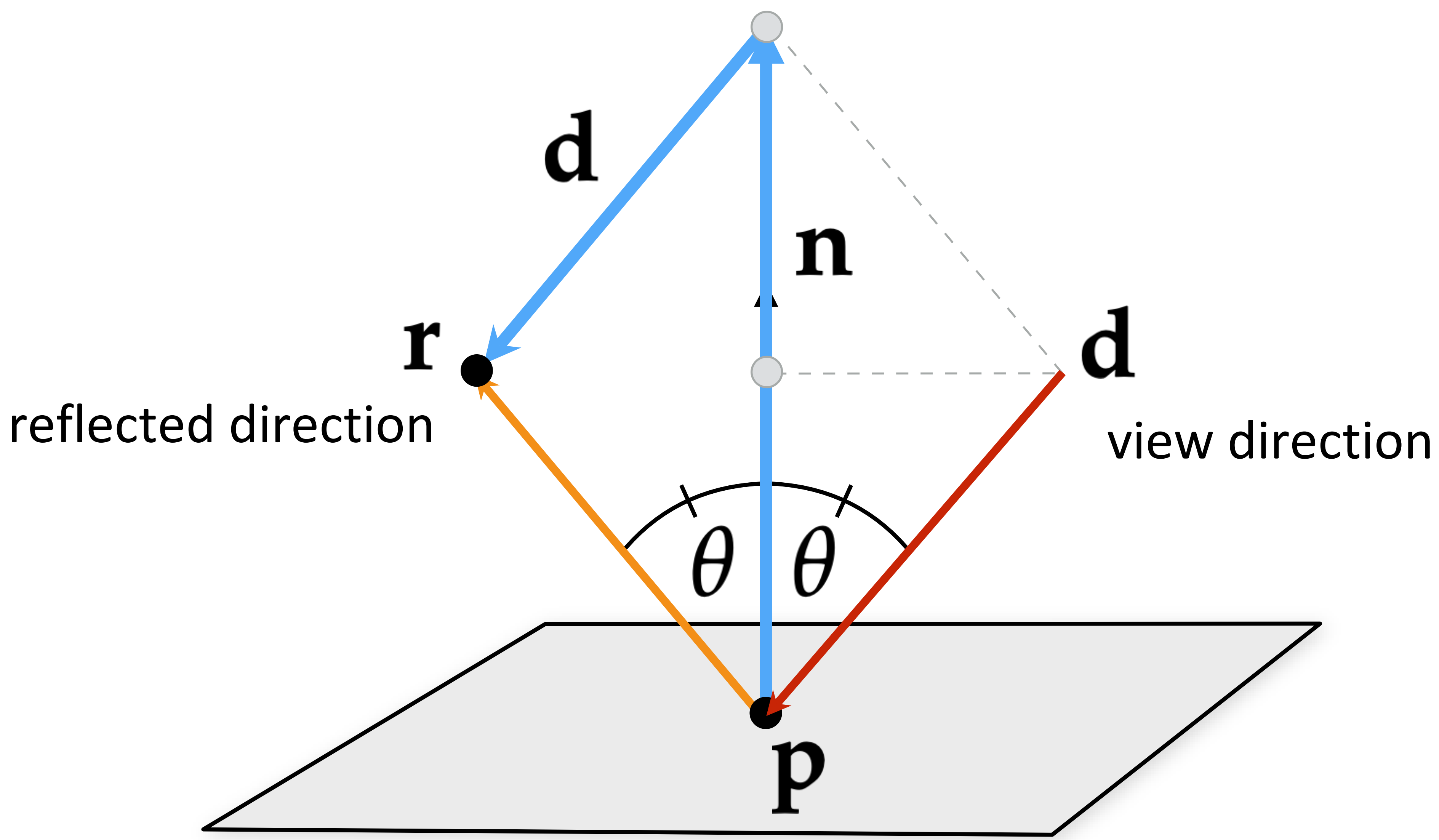
Mirror reflection



$$-2\mathbf{n}(\mathbf{n}\cdot\mathbf{d})$$

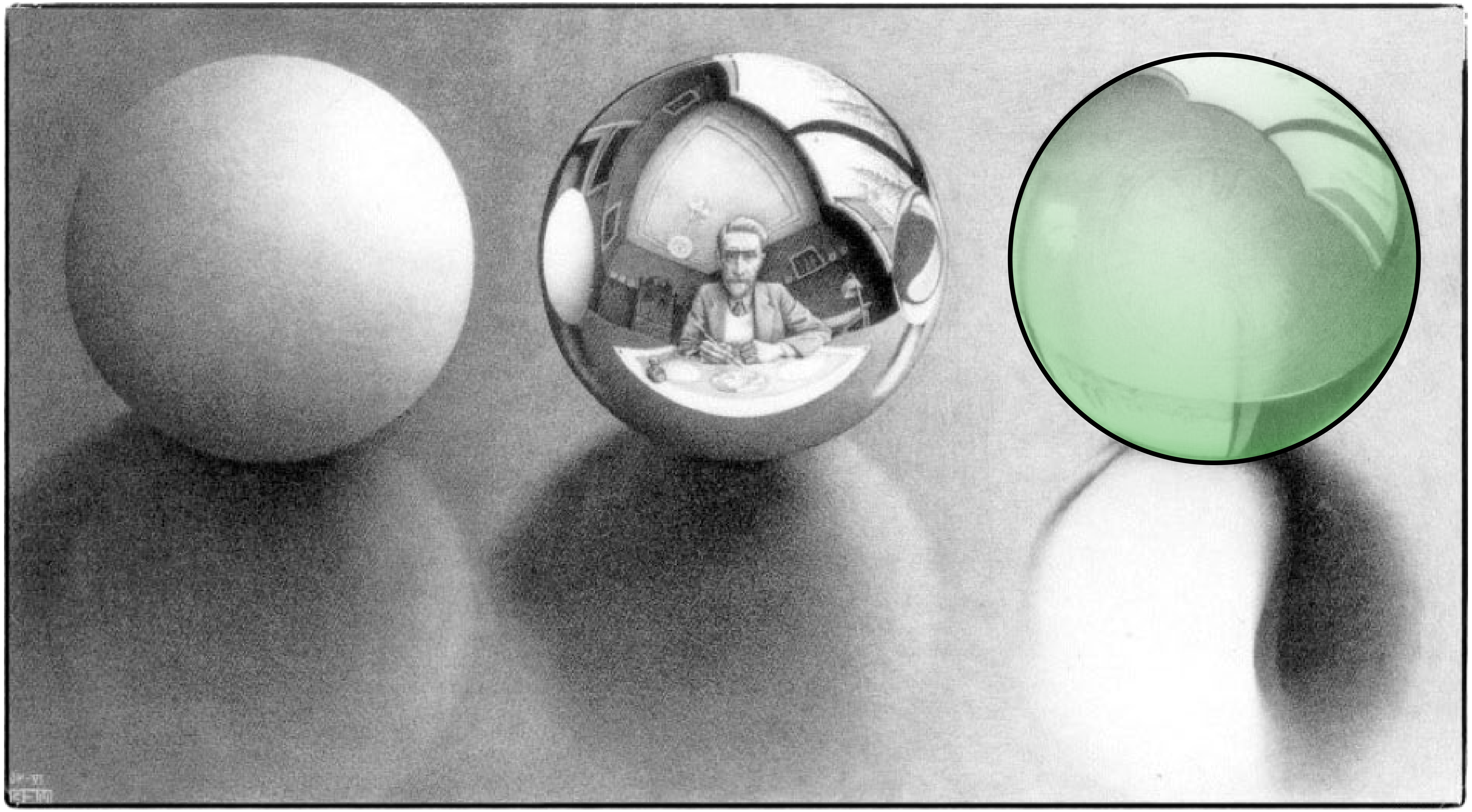
Mirror reflection

Assumes $\mathbf{n}$ is unit length



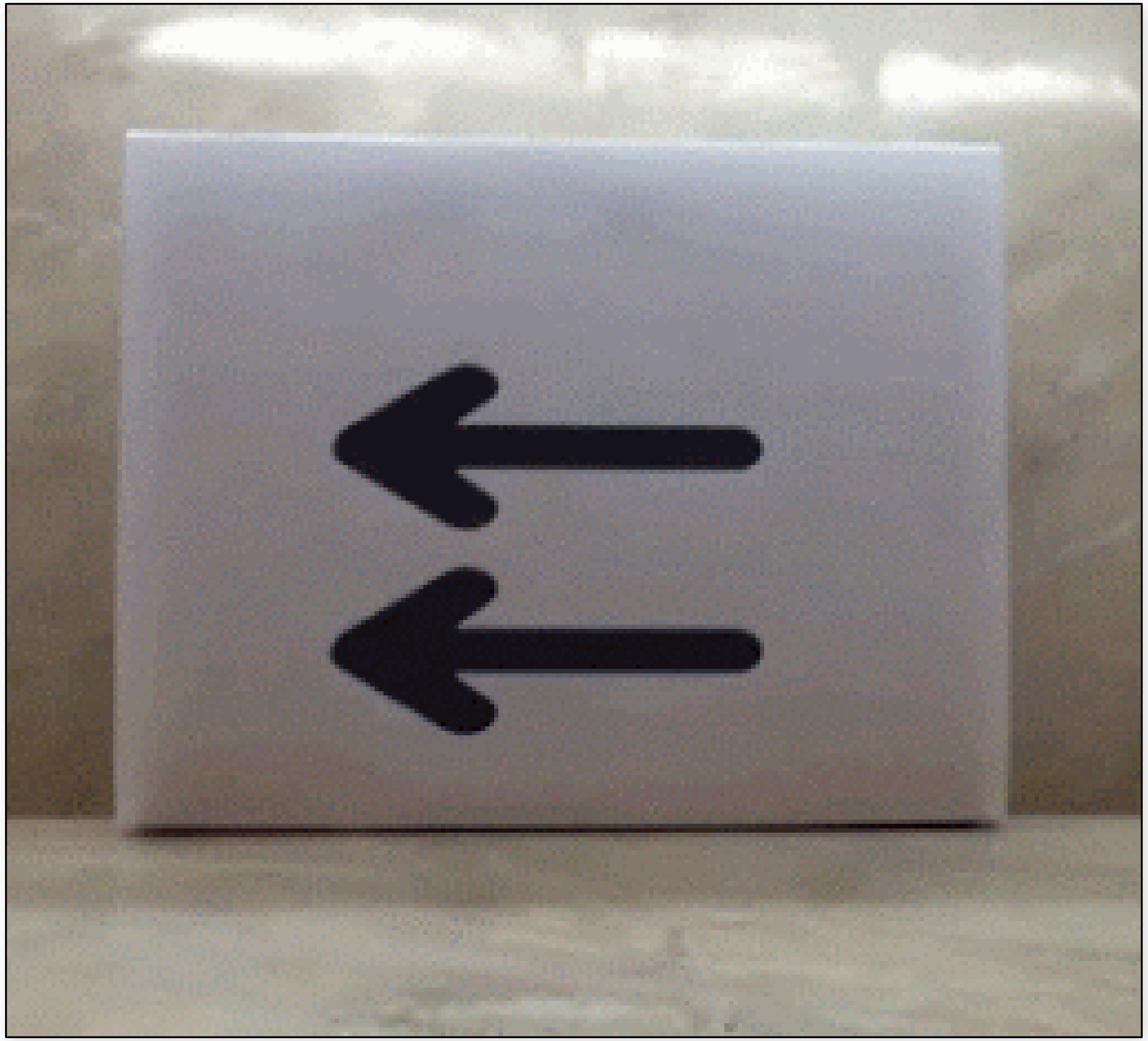
$$\mathbf{r} = -2\mathbf{n}(\mathbf{n} \cdot \mathbf{d}) + \mathbf{d}$$

Specular refraction



Refraction

Refraction



Index of Refraction

Speed of light in vacuum / speed of light in medium

Some values of η	
Vacuum	1
Air at STP	1.00029
Ice	1.31
Water	1.33
Crown glass	1.52 - 1.65
Diamond	2.417

Specular transmission/refraction

Materials like water, glass, etc., also refract/blend light

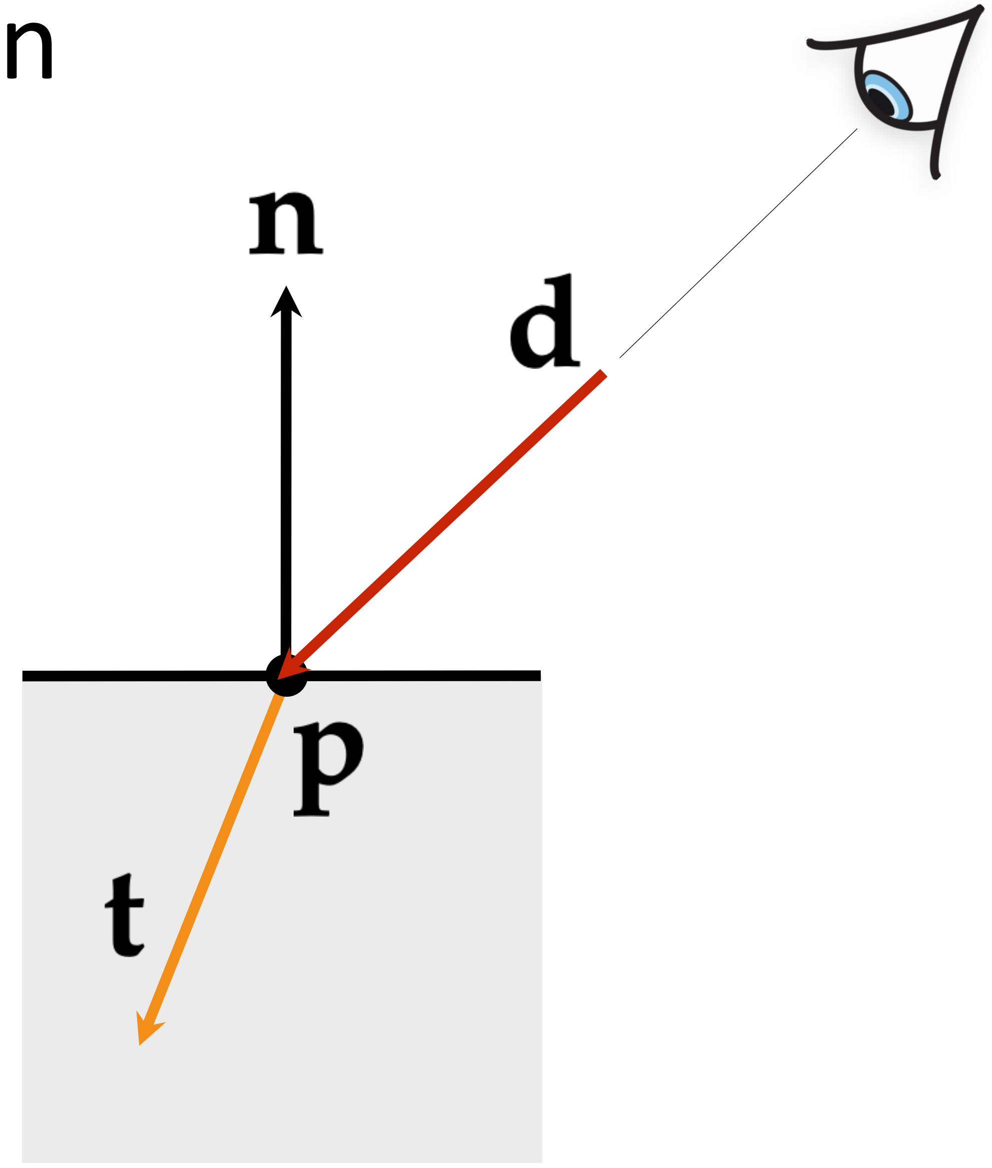
Trace a recursive ray in the refraction direction

$$L_t = k_t \mathrm{trace}(\mathbf{p}, \mathbf{t})$$

refracted light

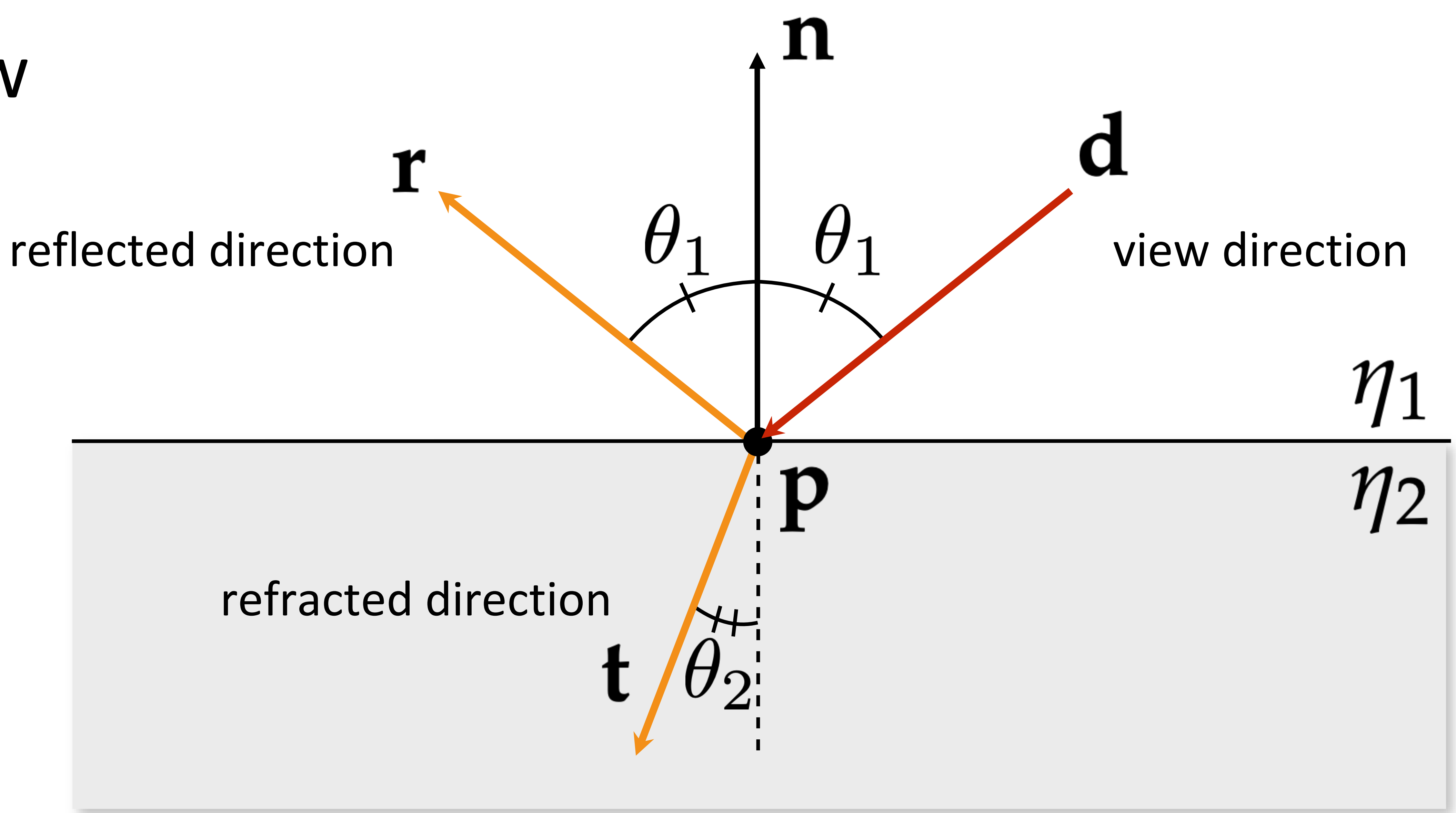
refraction scaled down

recursive call



Specular transmission/refraction

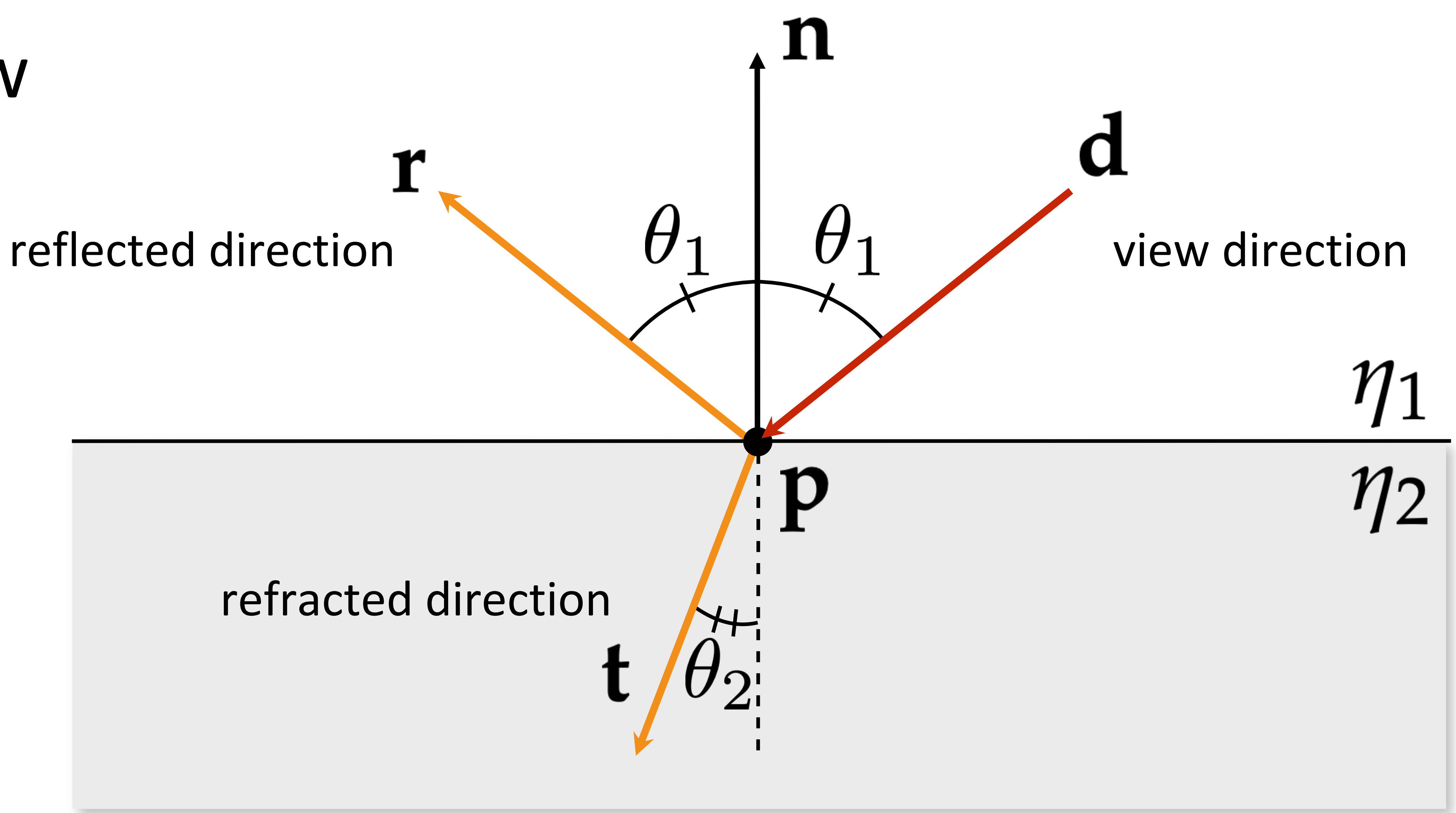
Snell's law



$$\eta_1 \sin\theta_1 = \eta_2 \sin\theta_2$$

Specular transmission/refraction

Snell's law



$$\mathbf{t} = \eta_1/\eta_2\left(\mathbf{d} - (\mathbf{d}\cdot\mathbf{n})\,\mathbf{n}\right) - \mathbf{n}\sqrt{1 - \eta_1^2/\eta_2^2\left(1 - (\mathbf{d}\cdot\mathbf{n})^2\right)}$$

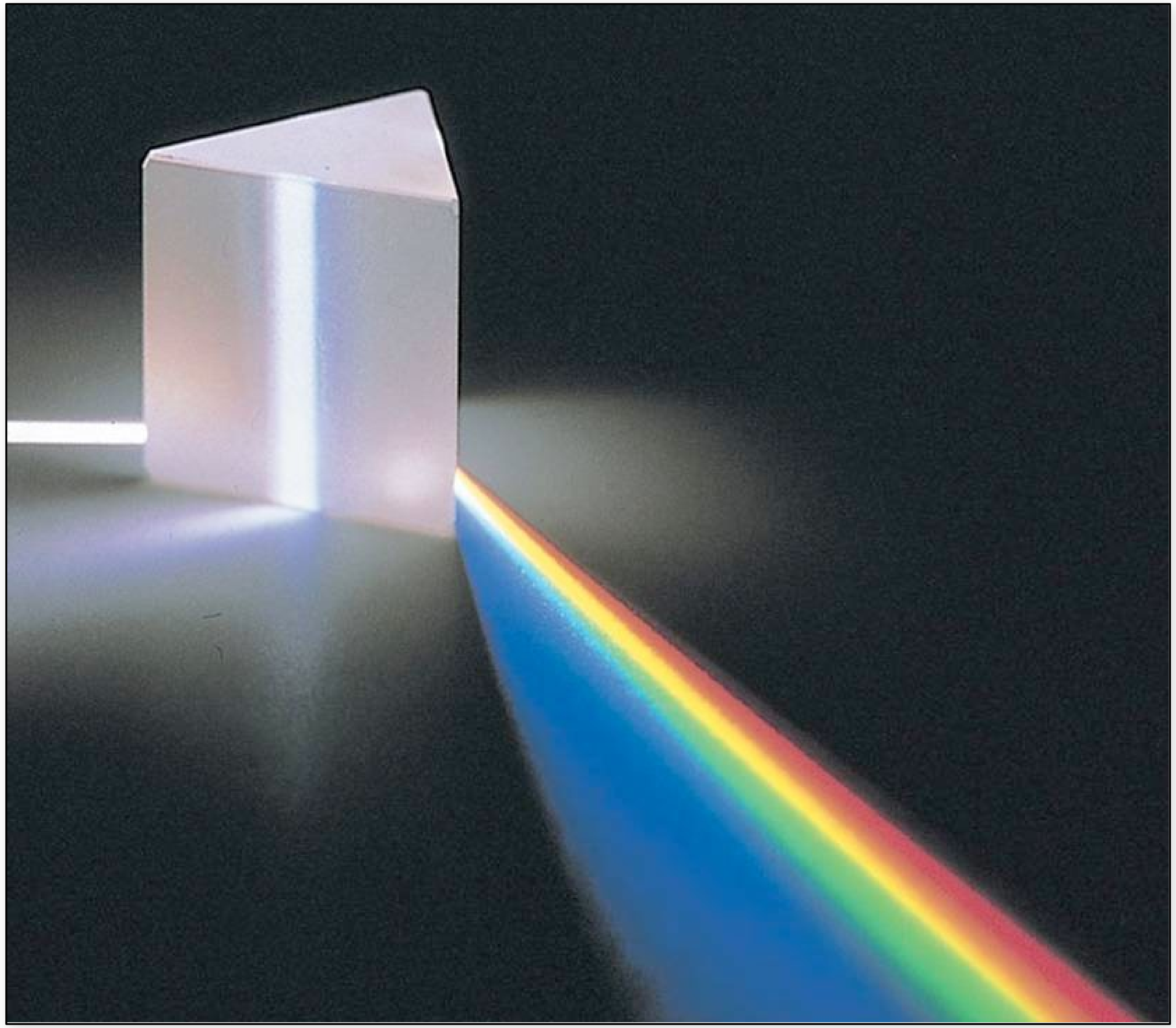
Index of Refraction

Speed of light in vacuum / speed of light in medium

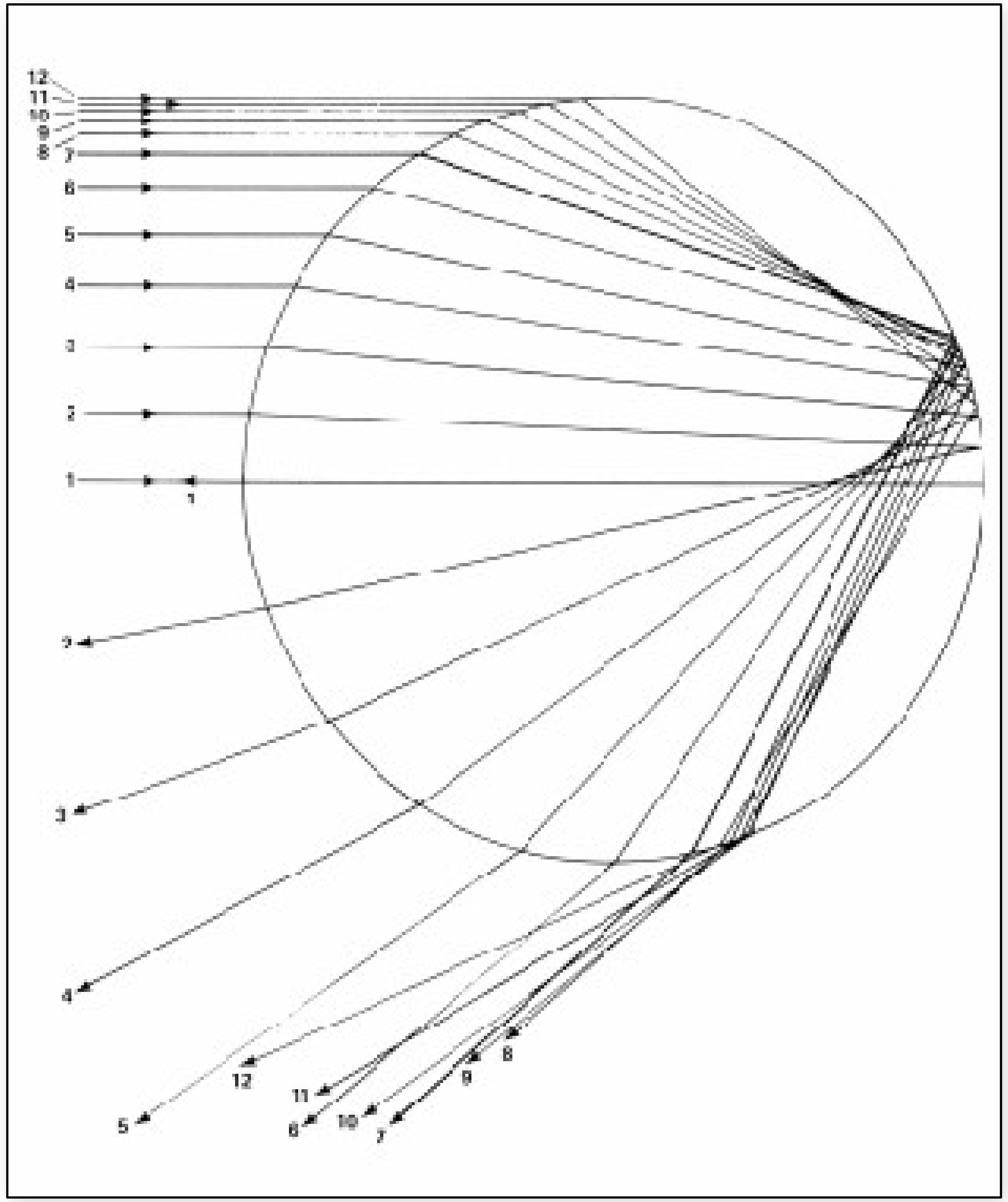
Some values of η	
Vacuum	1
Air at STP	1.00029
Ice	1.31
Water	1.33
Crown glass	1.52 - 1.65
Diamond	2.417

These are actually wavelength dependent!

Dispersion



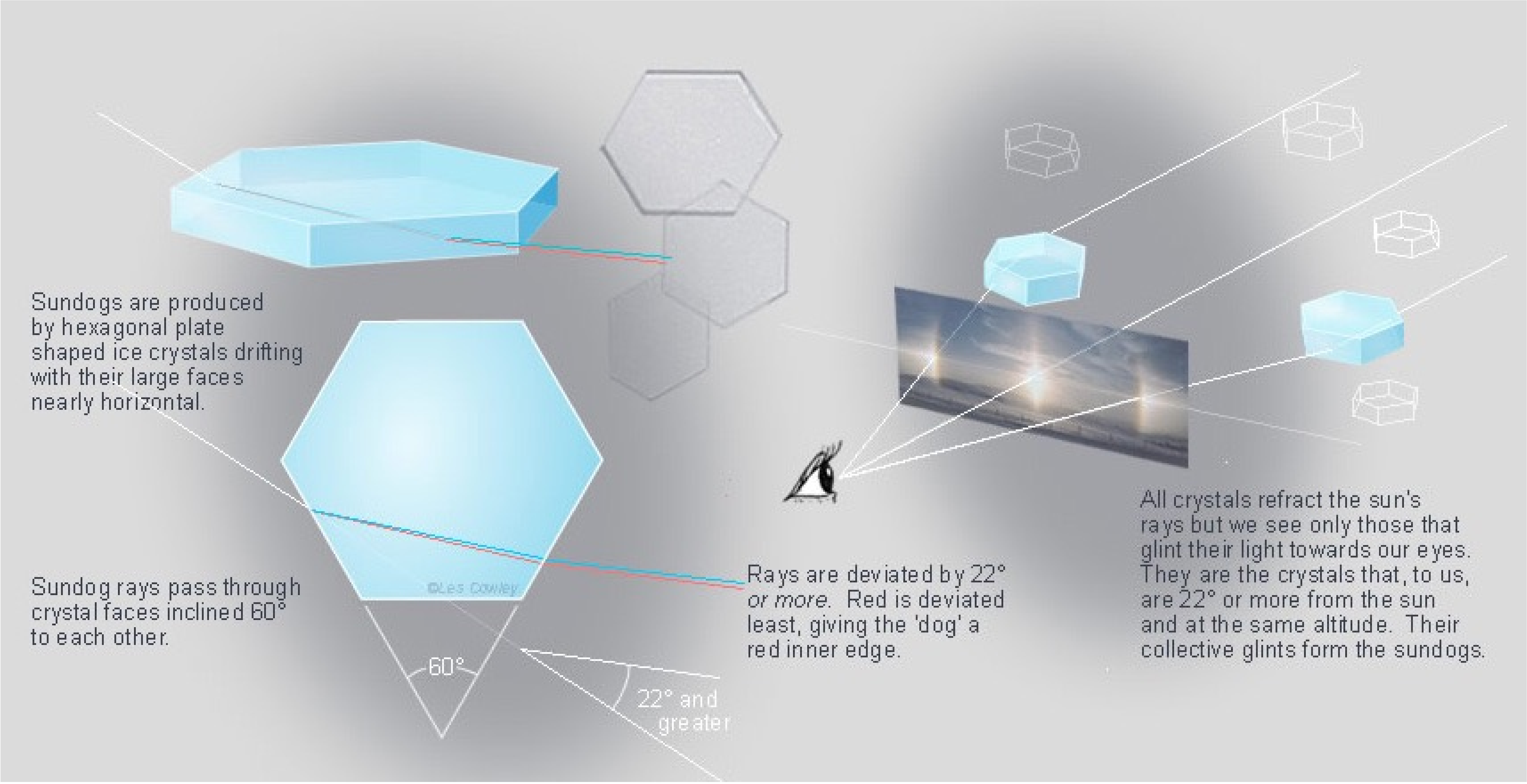
Refraction in a Waterdrop



Double rainbow all the way across the sky!

Dispersion: “Halos” and “Sun dogs”

Halos and Sundogs



Dispersion


denfo.ru

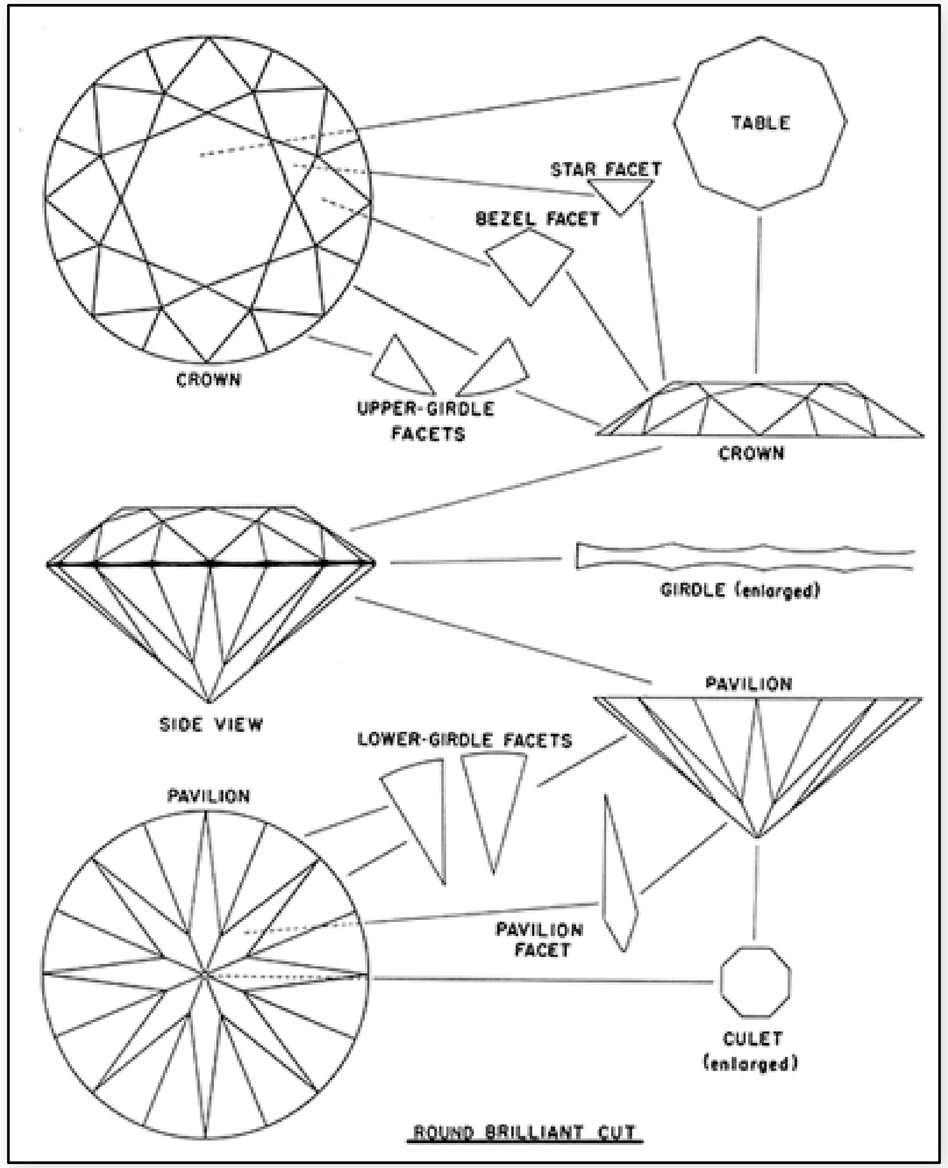
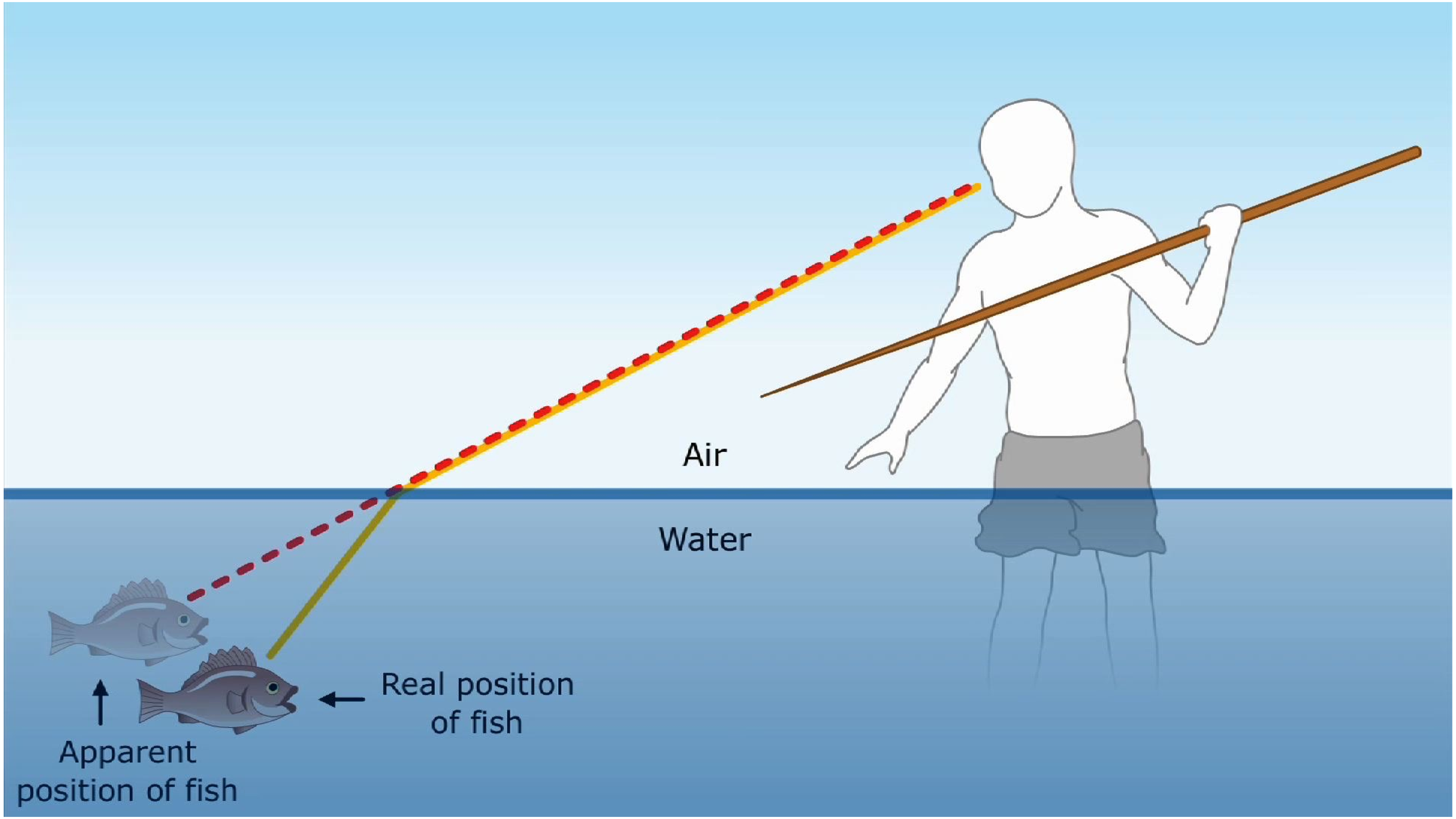


TABLE
STAR FACET
BEZEL FACET
CROWN
UPPER-GIRDLE FACETS
CROWN
GIRDLE (enlarged)
SIDE VIEW
PAVILION
LOWER-GIRDLE FACETS
PAVILION
PAVILION FACET
CULET (enlarged)
ROUND BRILLIANT CUT

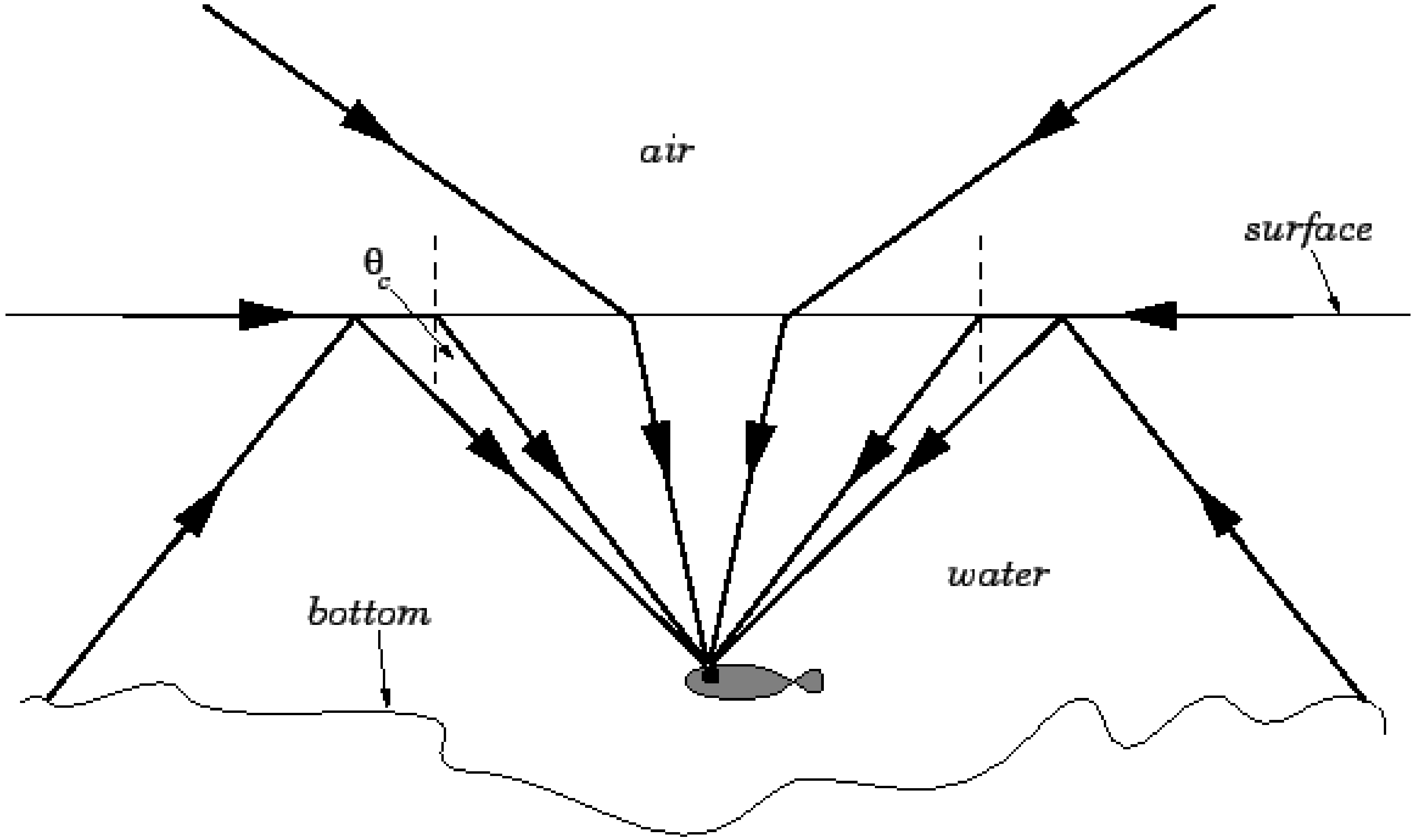
Practical consequences of refraction



What is this dark circle?

source: mrreid.org

What is this dark circle?

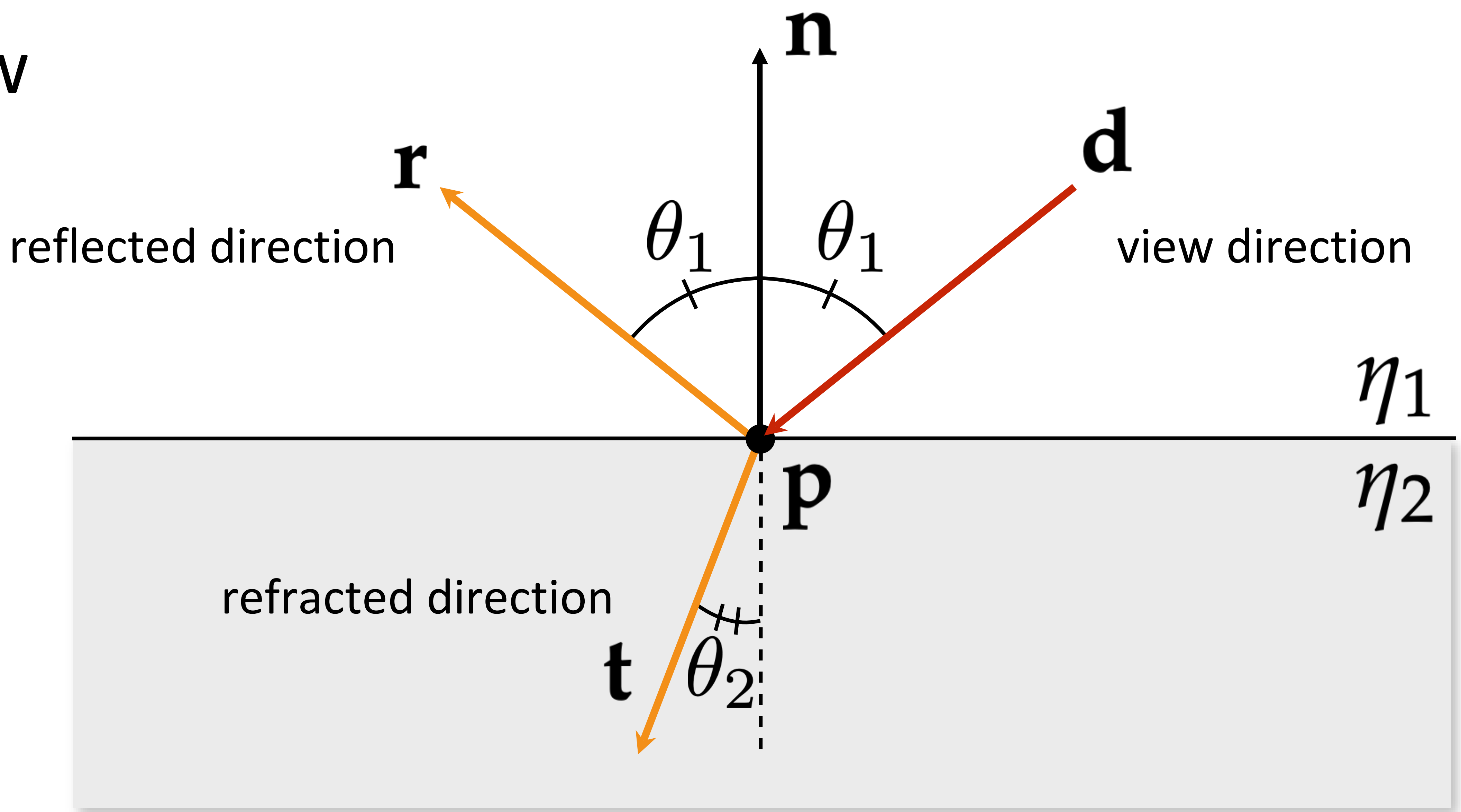


Called
"Snell's window"

Caused by total internal reflection

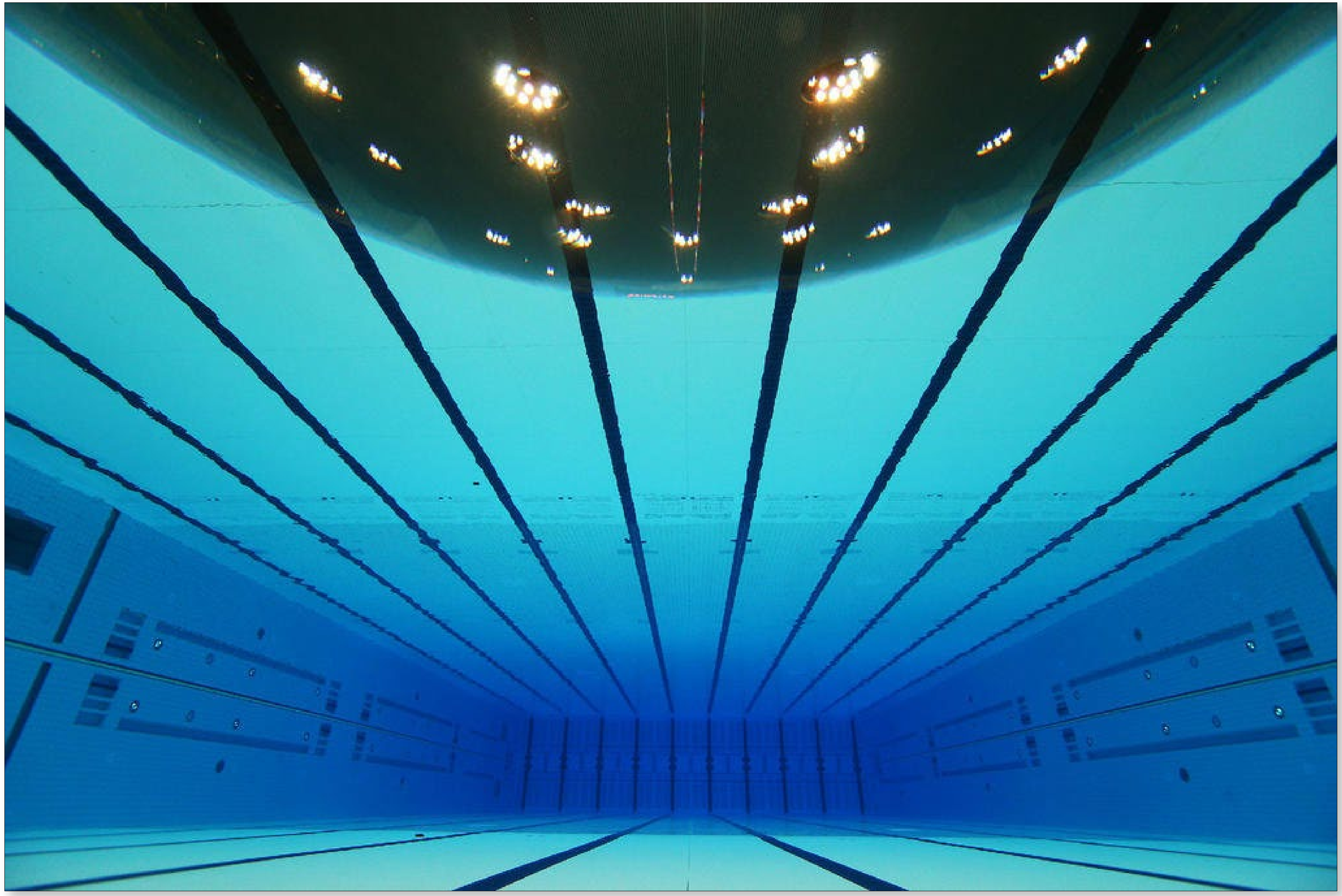
Recall...

Snell's law



$$\mathbf{t} = \eta_1/\eta_2 \left(\mathbf{d} - (\mathbf{d}\cdot\mathbf{n})\,\mathbf{n}\right) - \mathbf{n}\sqrt{1 - \eta_1^2/\eta_2^2\left(1 - (\mathbf{d}\cdot\mathbf{n})^2\right)}$$

Total Internal Reflection



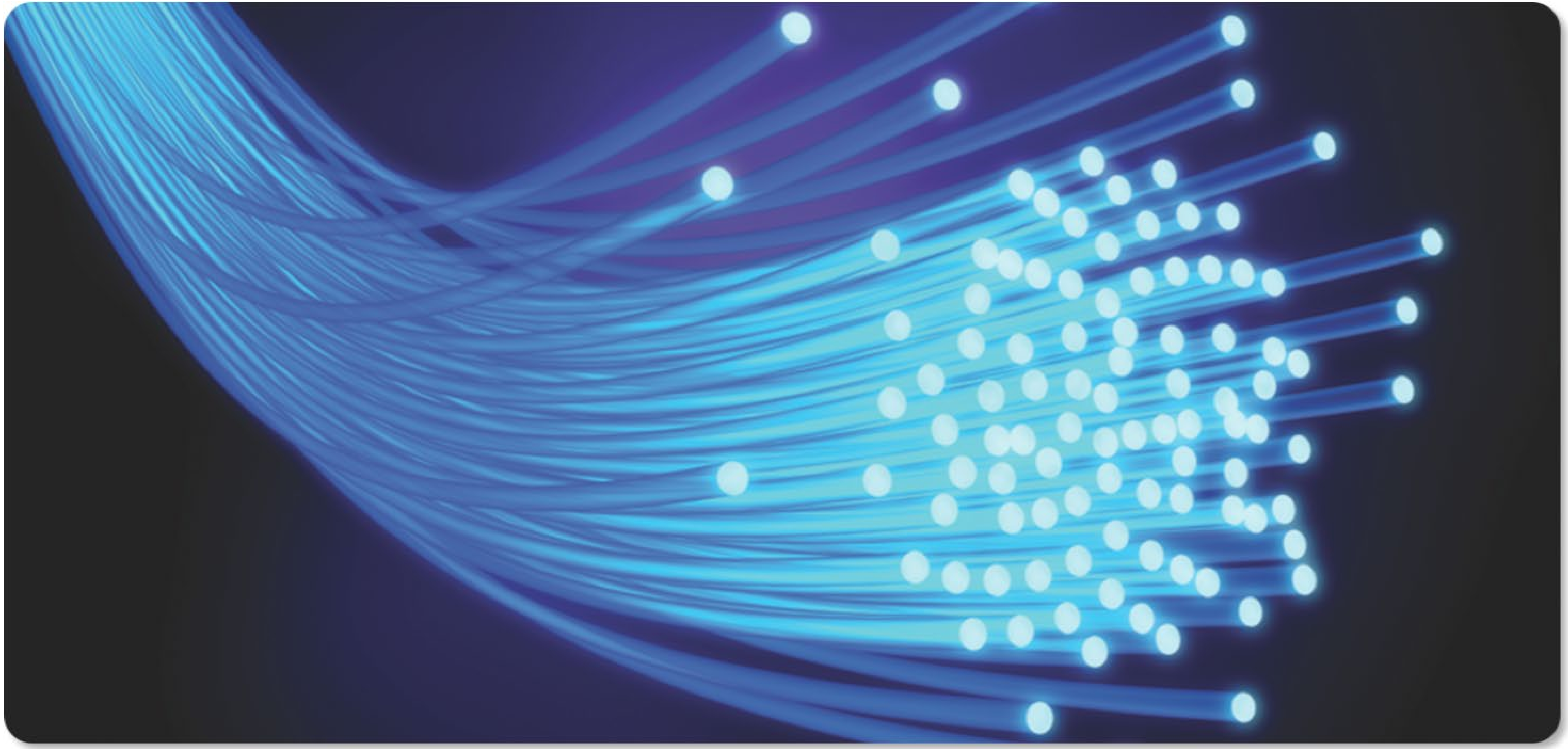
source: mrreid.org

Total Internal Reflection

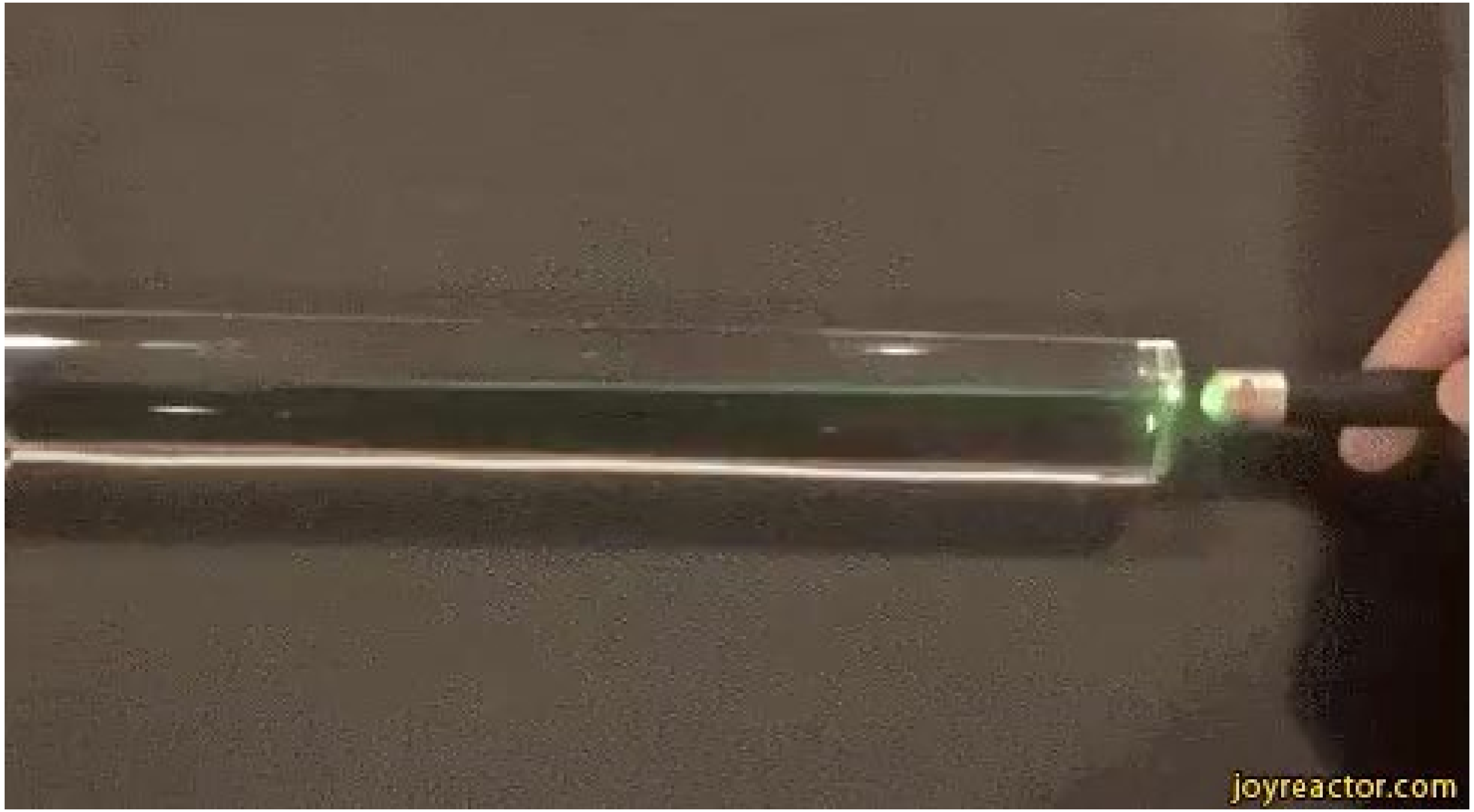
Total Internal Reflection

Can only happen when the ray starts in the higher index medium

Total Internal Reflection



Total Internal Reflection

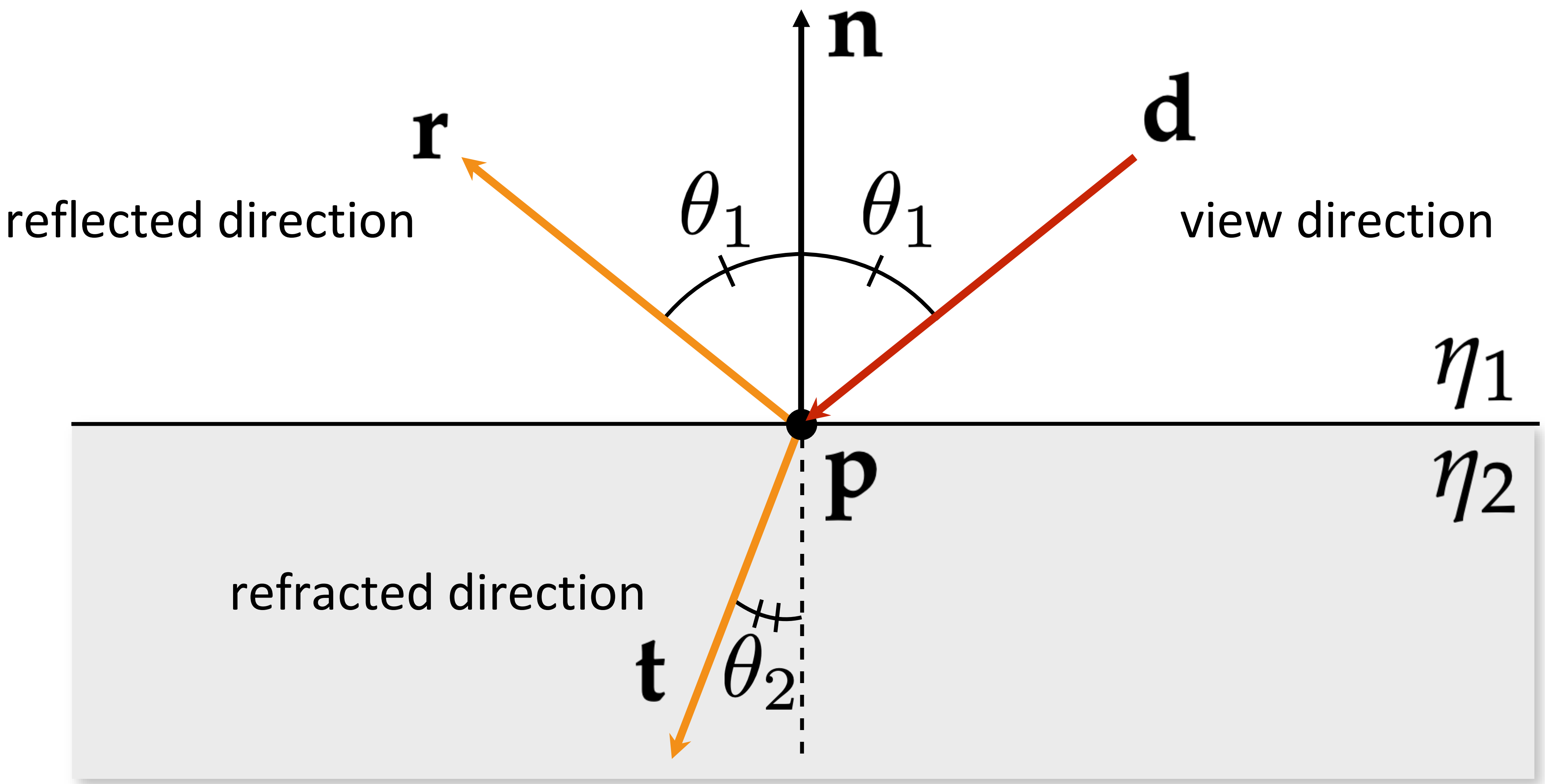


source: imgur.com

Reflection vs. Refraction

How much light is reflected vs. refracted?

- in reality determined by “Fresnel equations”



Fresnel Equations

Reflection and *refraction* from smooth *dielectric* (e.g. glass) surfaces

Reflection from *conducting* (e.g. metal) surfaces

Derived from Maxwell equations

Involves polarization of the wave

Fresnel Equations for Dielectrics

Reflection of light polarized parallel and perpendicular to the plane of refraction

$$\rho_{||} = \frac{\eta_2 \cos\theta_1 - \eta_1 \cos\theta_2}{\eta_2 \cos\theta_1 + \eta_1 \cos\theta_2}$$

$$\rho_{\perp} = \frac{\eta_1 \cos\theta_1 - \eta_2 \cos\theta_2}{\eta_1 \cos\theta_1 + \eta_2 \cos\theta_2}$$

reflected: $F_r = \frac{1}{2}\left(\rho_{||}^2 + \rho_{\perp}^2\right)$

refracted: $F_t = 1 - F_r$

What's happening in this photo?

source: flickr user neofob

Polarizing Filter

Polarization

Without Polarizer

With Polarizing Filter

source: photography.ca

Polarization

Without Polarizer

With Polarizing Filter

source: wikipedia

Effect of Polarization

Effect of Polarization

Fresnel Equations for Dielectrics

Reflection of light polarized parallel and perpendicular to the plane of refraction

$$\rho_{||} = \frac{\eta_2 \cos\theta_1 - \eta_1 \cos\theta_2}{\eta_2 \cos\theta_1 + \eta_1 \cos\theta_2}$$

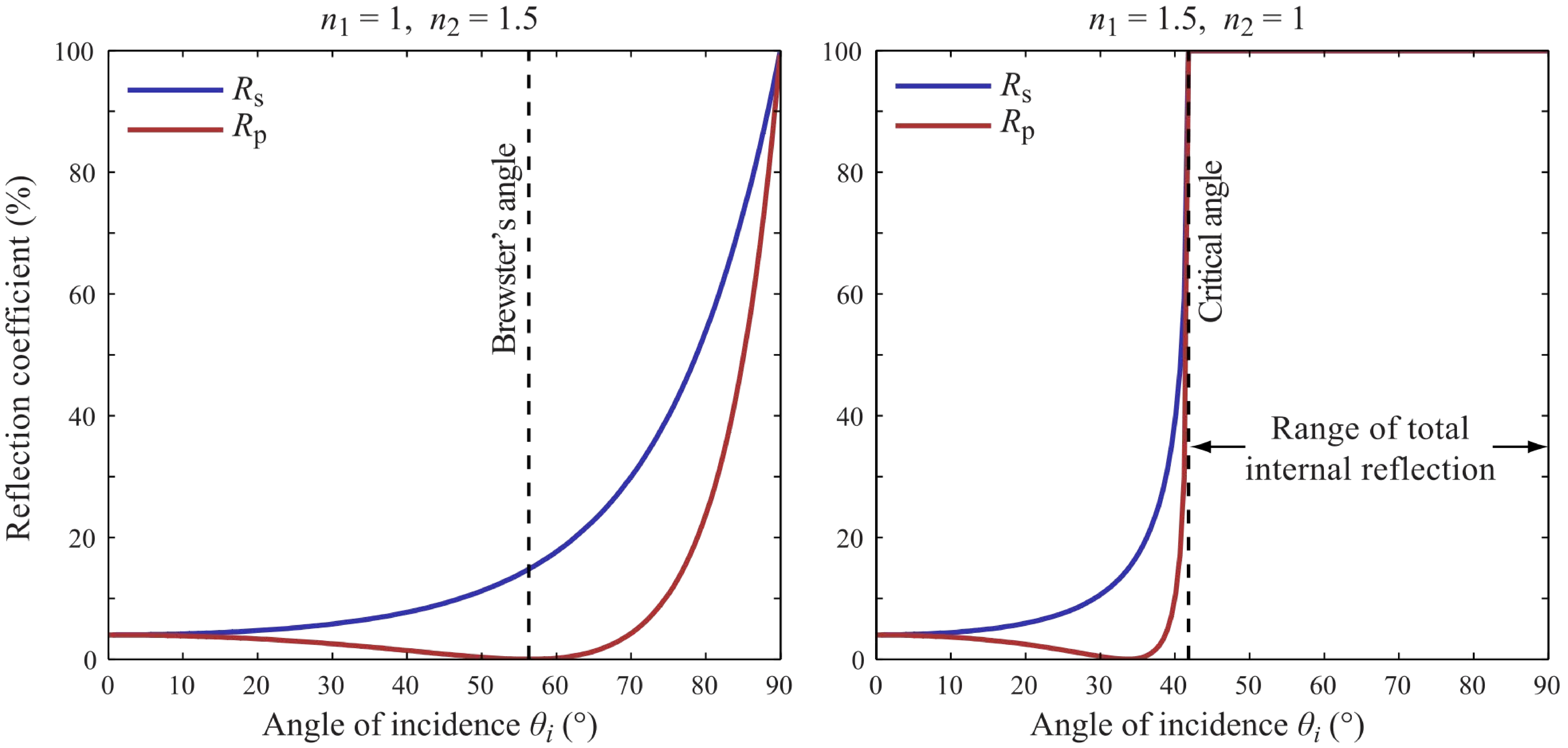
$$\rho_{\perp} = \frac{\eta_1 \cos\theta_1 - \eta_2 \cos\theta_2}{\eta_1 \cos\theta_1 + \eta_2 \cos\theta_2}$$

reflected: $F_r = \frac{1}{2}\left(\rho_{||}^2 + \rho_{\perp}^2\right)$

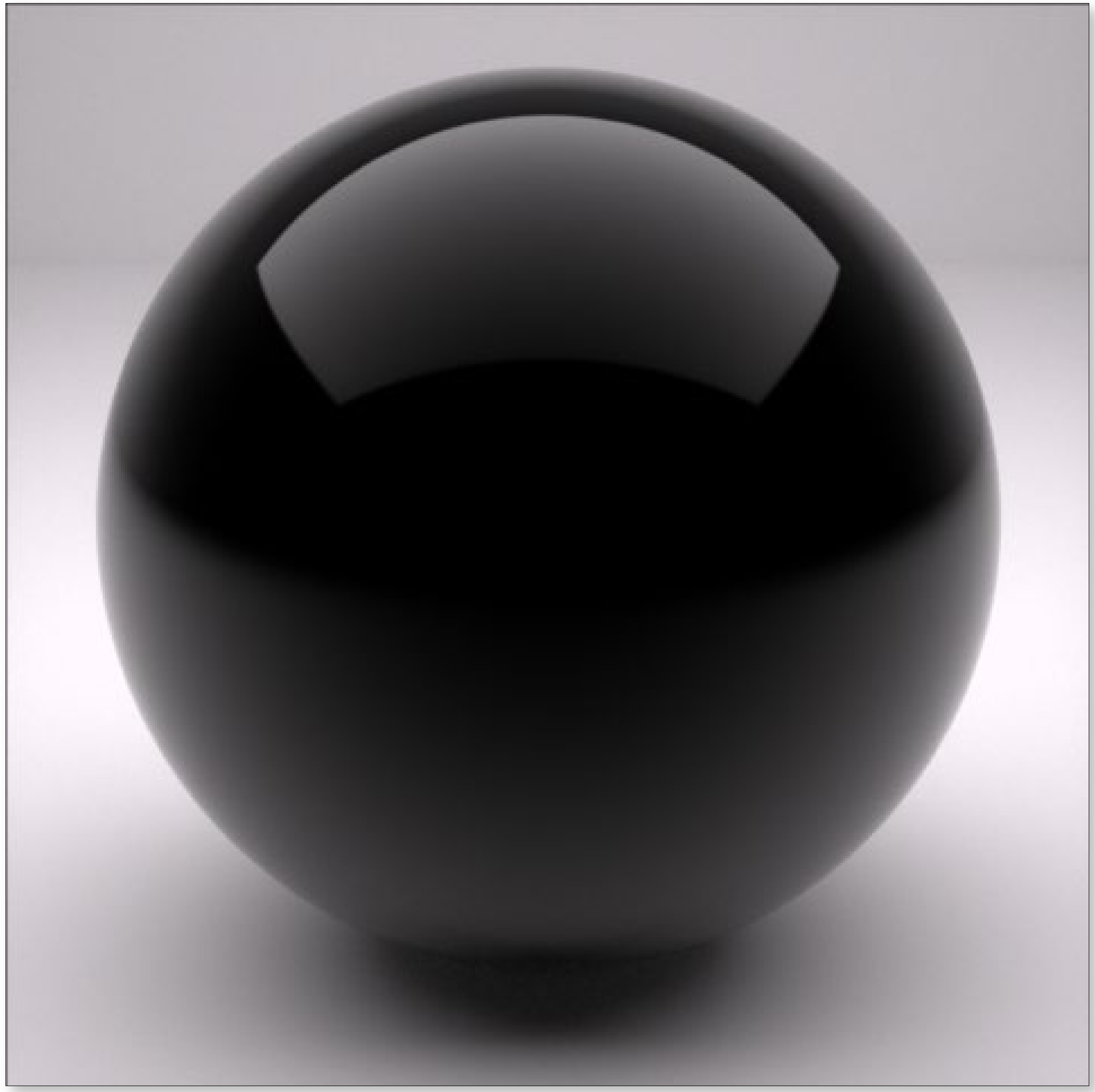
refracted: $F_t = 1 - F_r$

- The Shirley book uses a faster approximation (Schlick), but to get full accuracy you'd need to use these equations

Fresnel equations for glass



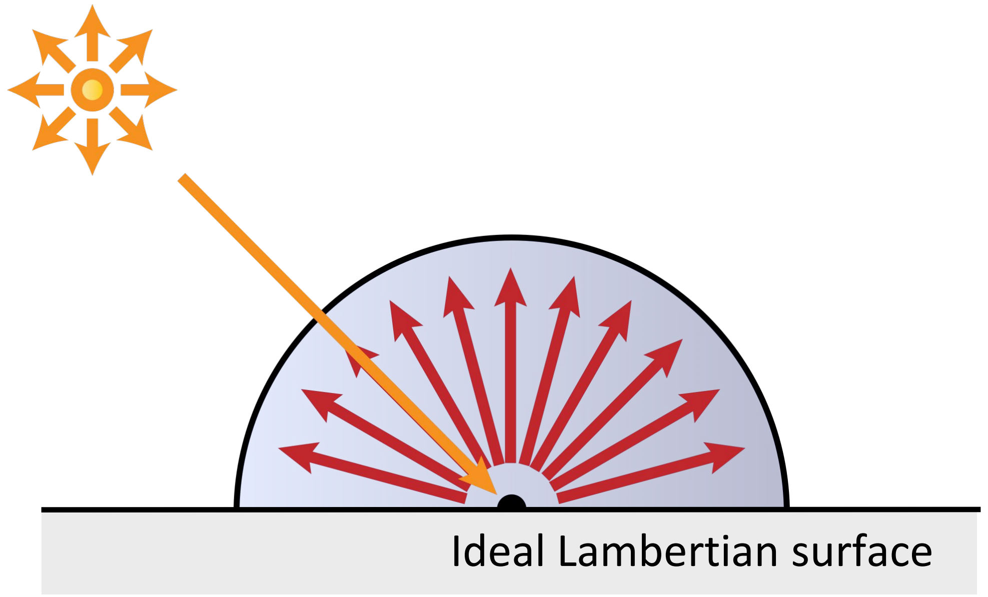
Fresnel Reflection



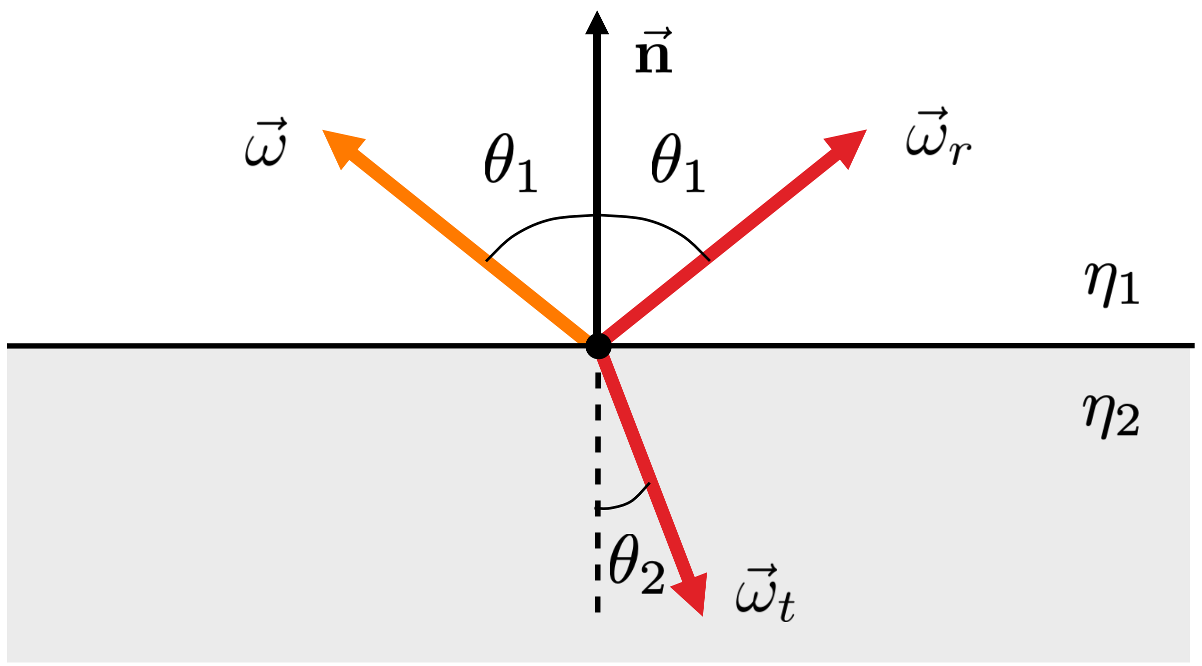
Fresnel reflection/refraction

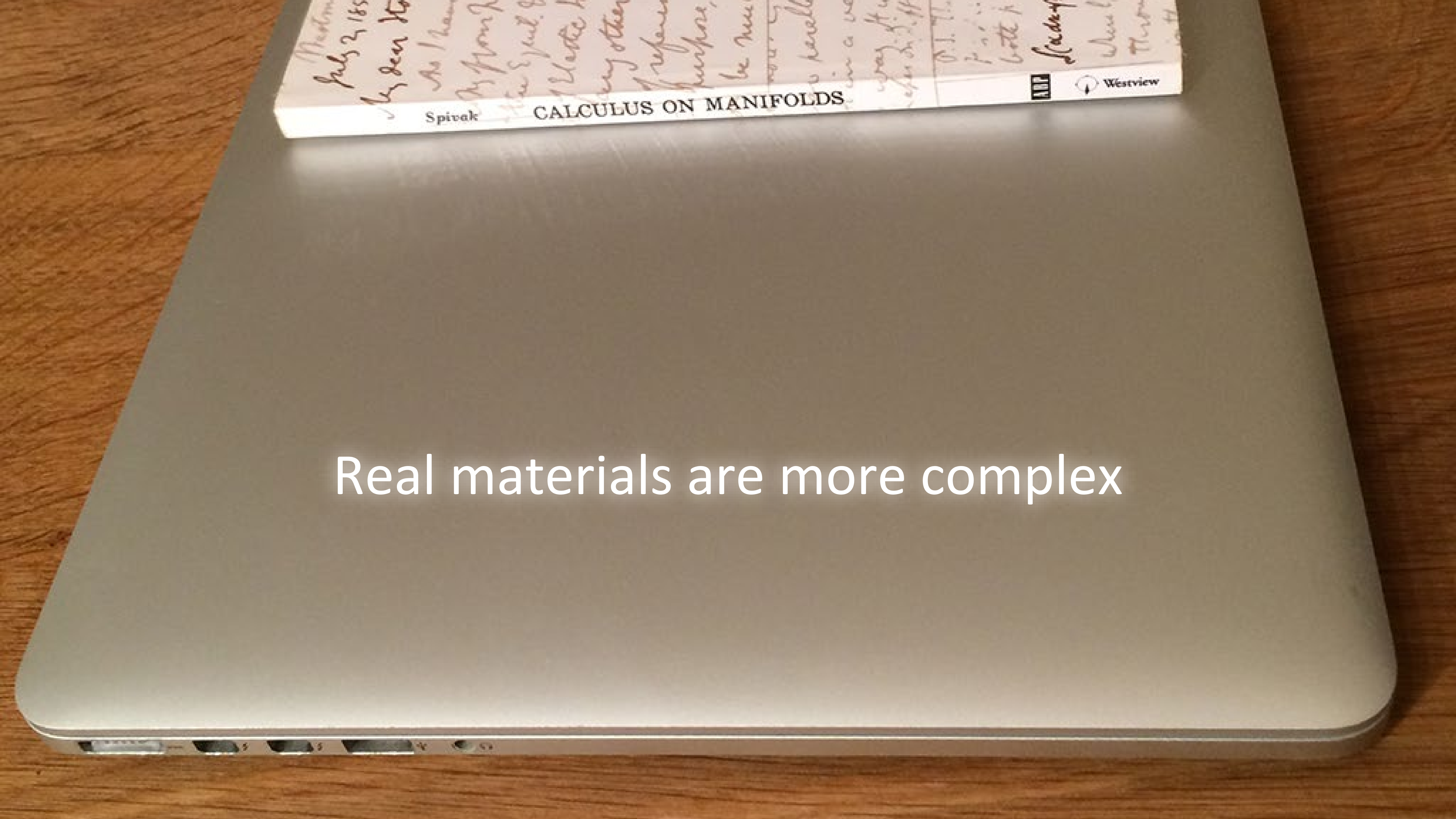
So Far: Idealized BRDF Models

Diffuse



Specular Reflection and Refraction





Real materials are more complex

Rough materials

In reality, most materials are neither perfectly diffuse nor specular, but somewhere in between

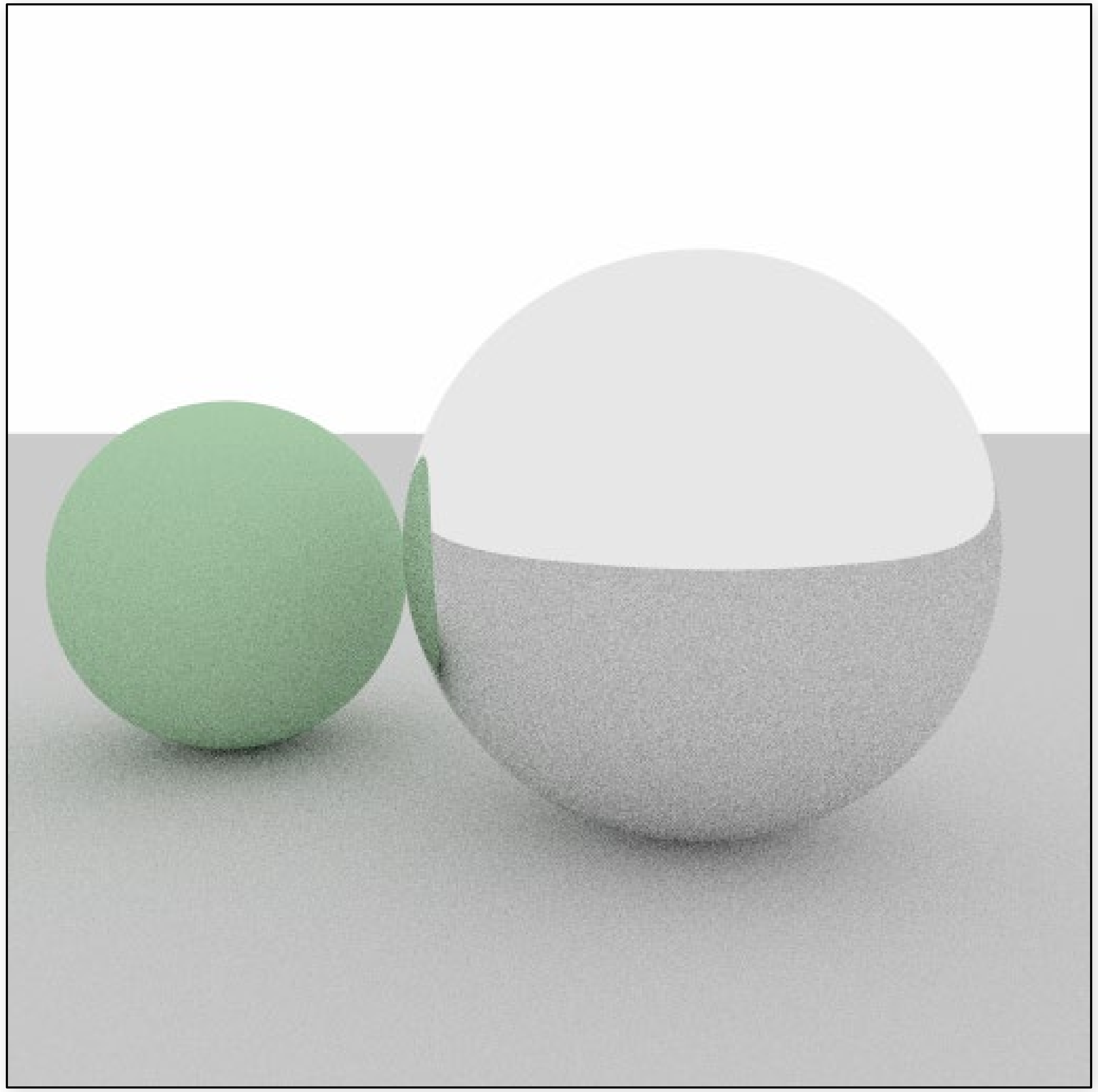
- Imagine a shiny surface scratched up at a microscopic level

We will look at a more principled way to handle this later.

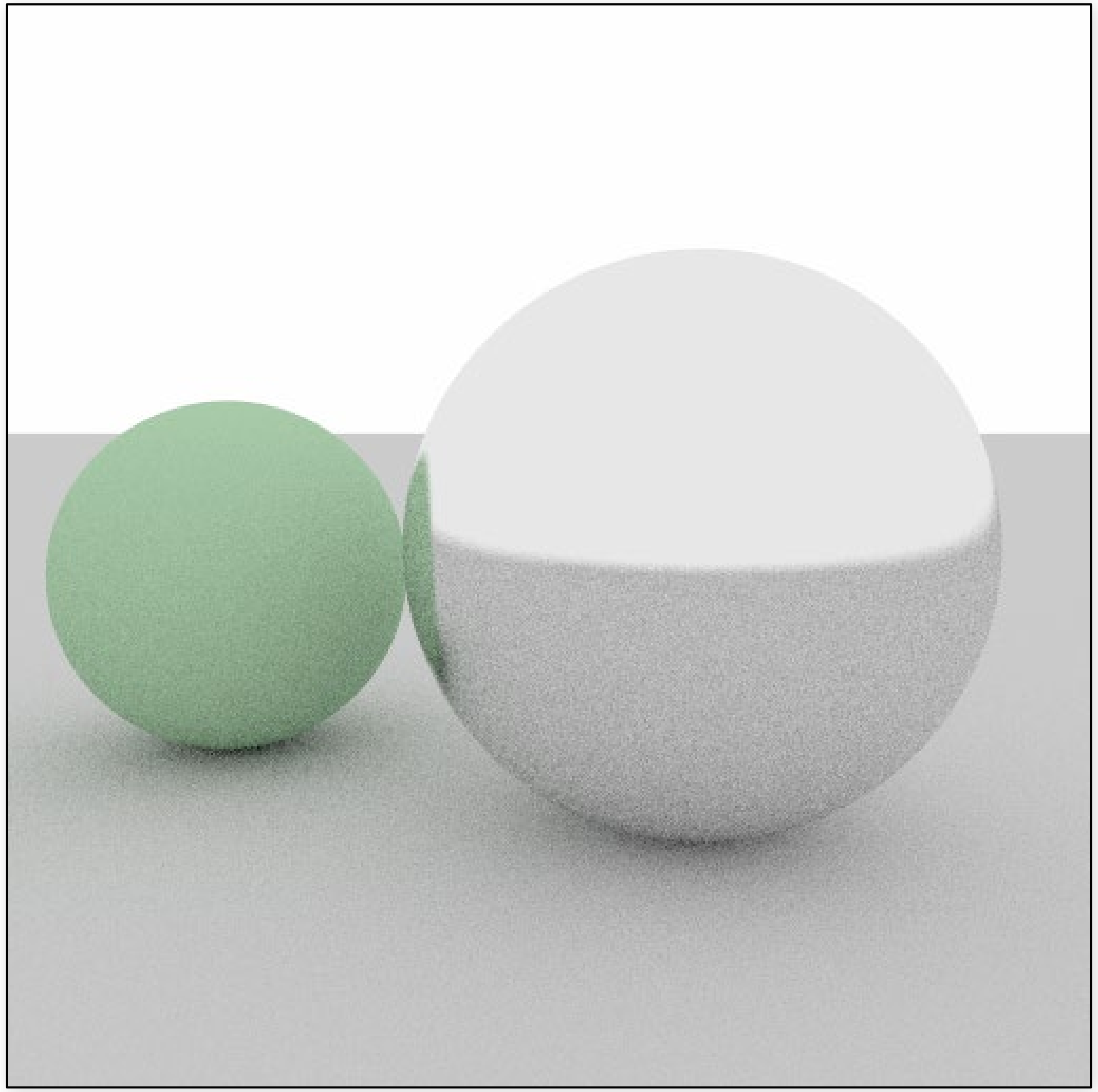
For now, we can easily approximate one important characteristic: blurred reflections

- Compute reflection direction, then add a random offset to it
- Sample random offset from sphere. Scale it to increase/decrease fuzziness

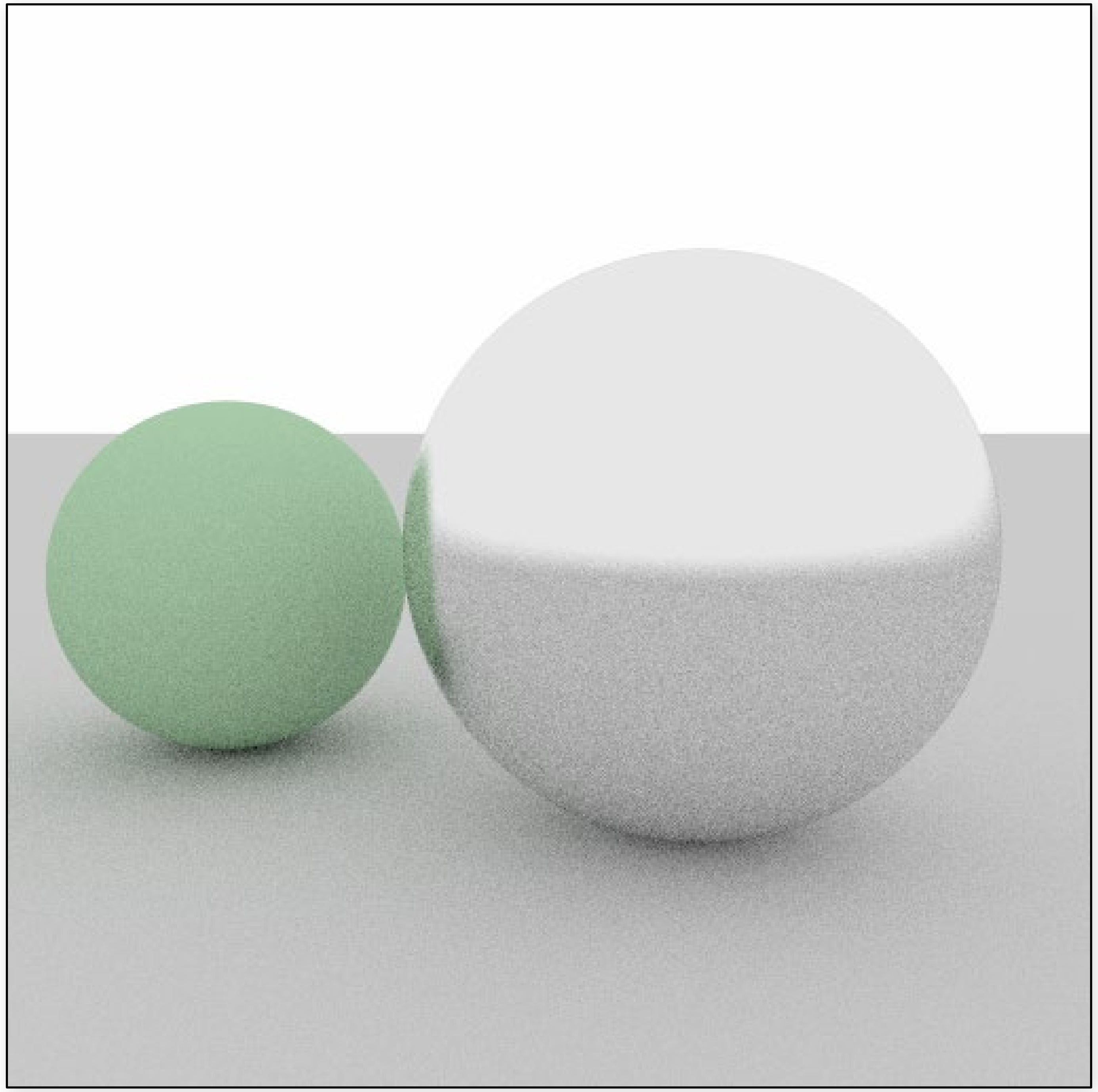
Diffuse & mirror spheres



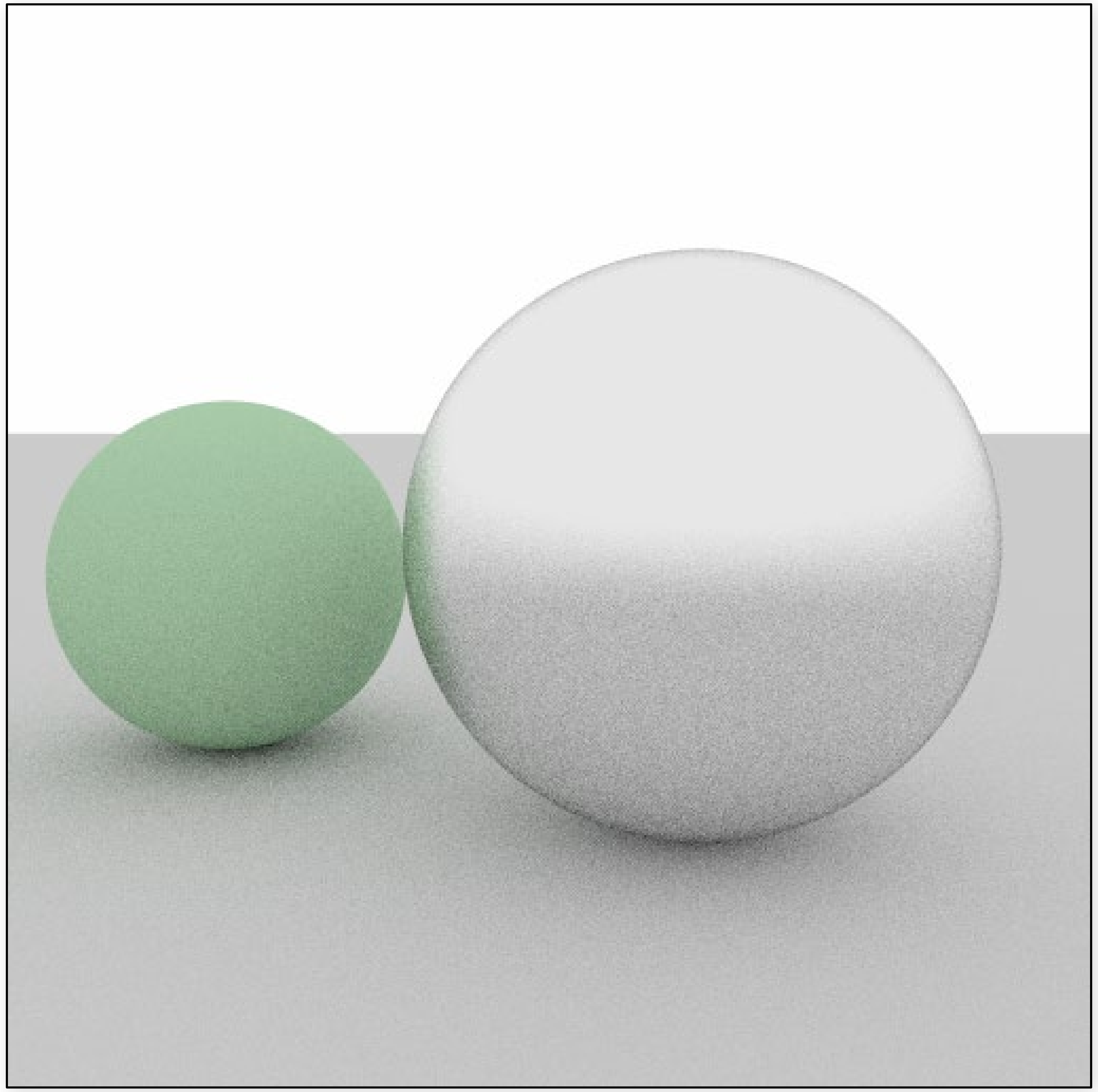
Diffuse & rough mirror spheres



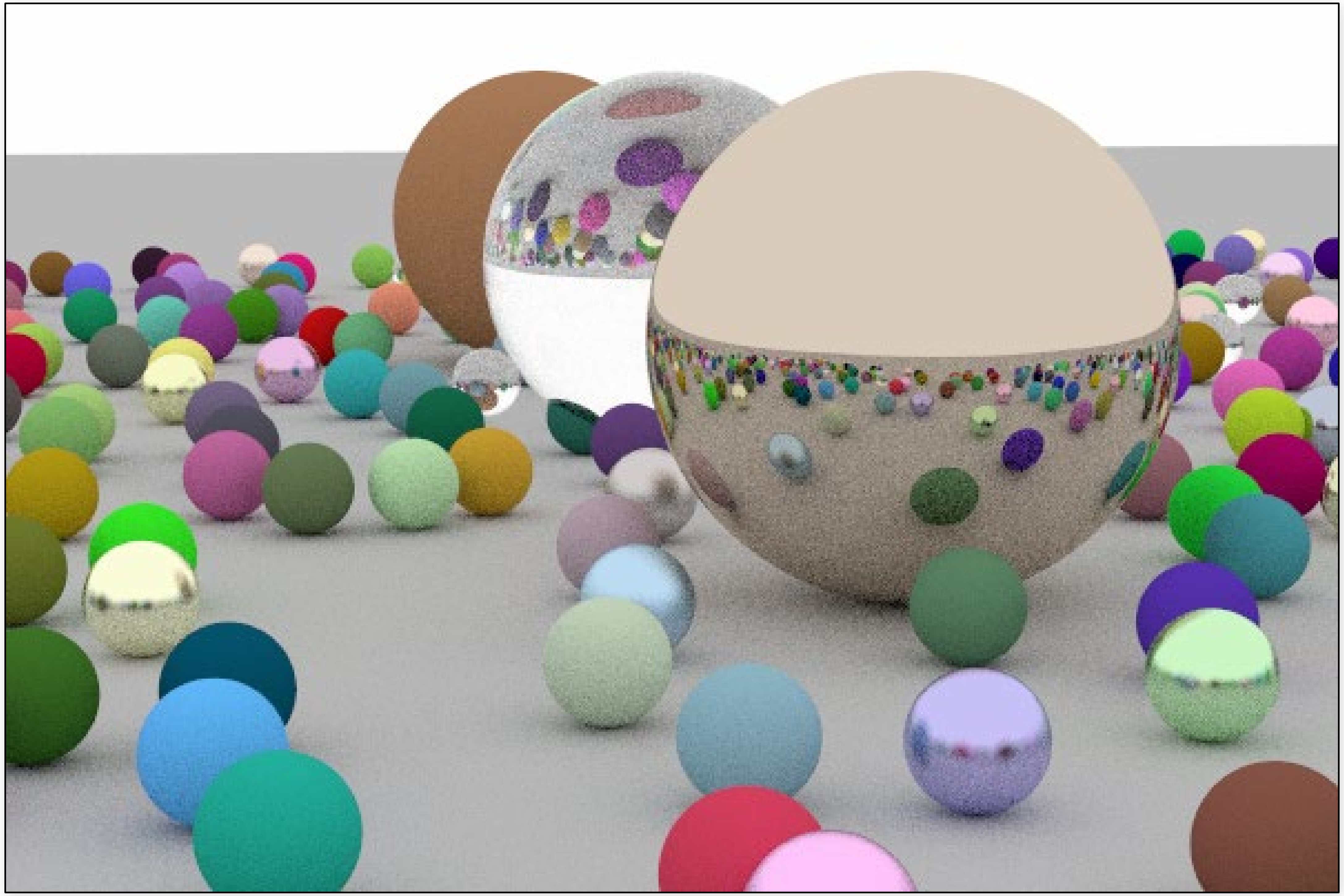
Diffuse & rough mirror spheres



Diffuse & rough mirror spheres



Putting it together



Lighting

Lighting

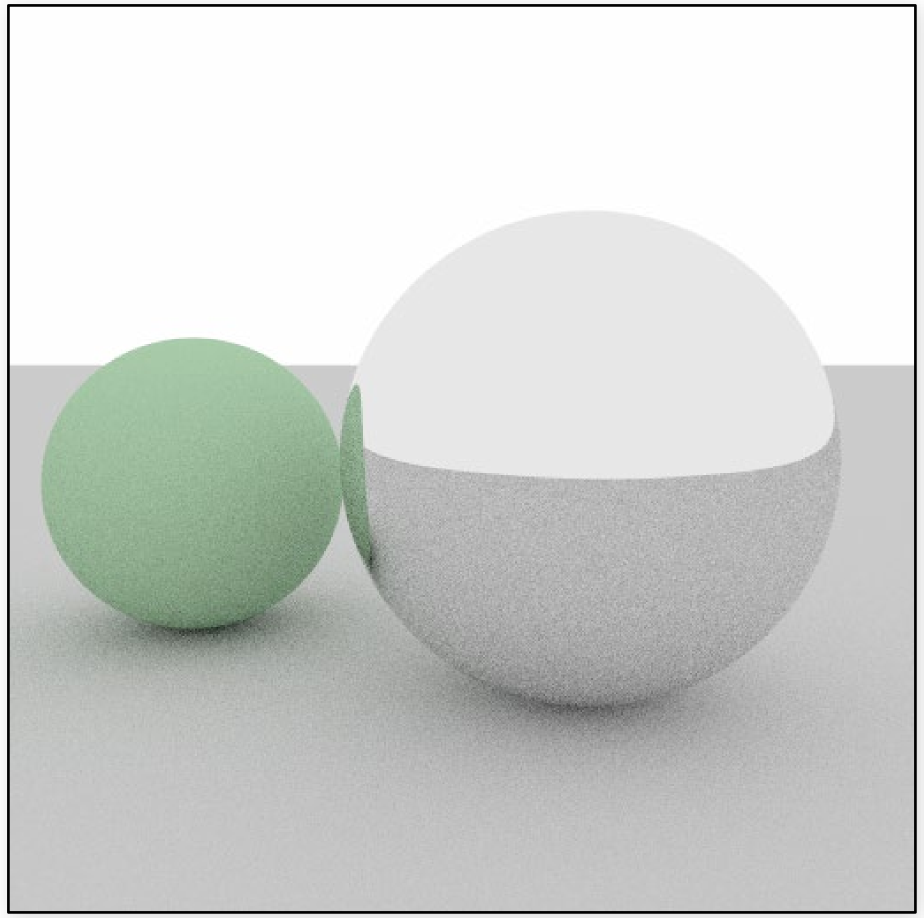
So far, the sky or background has been the only source of emitted light

But we can easily make any surface a light source!

- Just return an emitted color when a ray hits that surface
- Add a function to material that returns emitted color
    - Returns black (all zeros) for regular (non-emissive) surfaces
    - Color will often be greater than (1,1,1)
- Also possible for surfaces to emit & scatter (but not common)

Pseudo-code

```
Scene::trace(Ray ray)
   hit = surfaces.intersect(ray);
   if hit
      [col, sRay] = hit->mat->scatter(ray)
      return col * trace(sRay);
   else
      return backgroundColor;
```



Pseudo-code

```
Scene::trace(Ray ray)
    hit = surfaces.intersect(ray);
    if hit
        emit = hit->mat->emit(ray)
        [col, sRay] = hit->mat->scatter(ray)
        return emit + col * trace(sRay);
    else
        return backgroundColor;
```