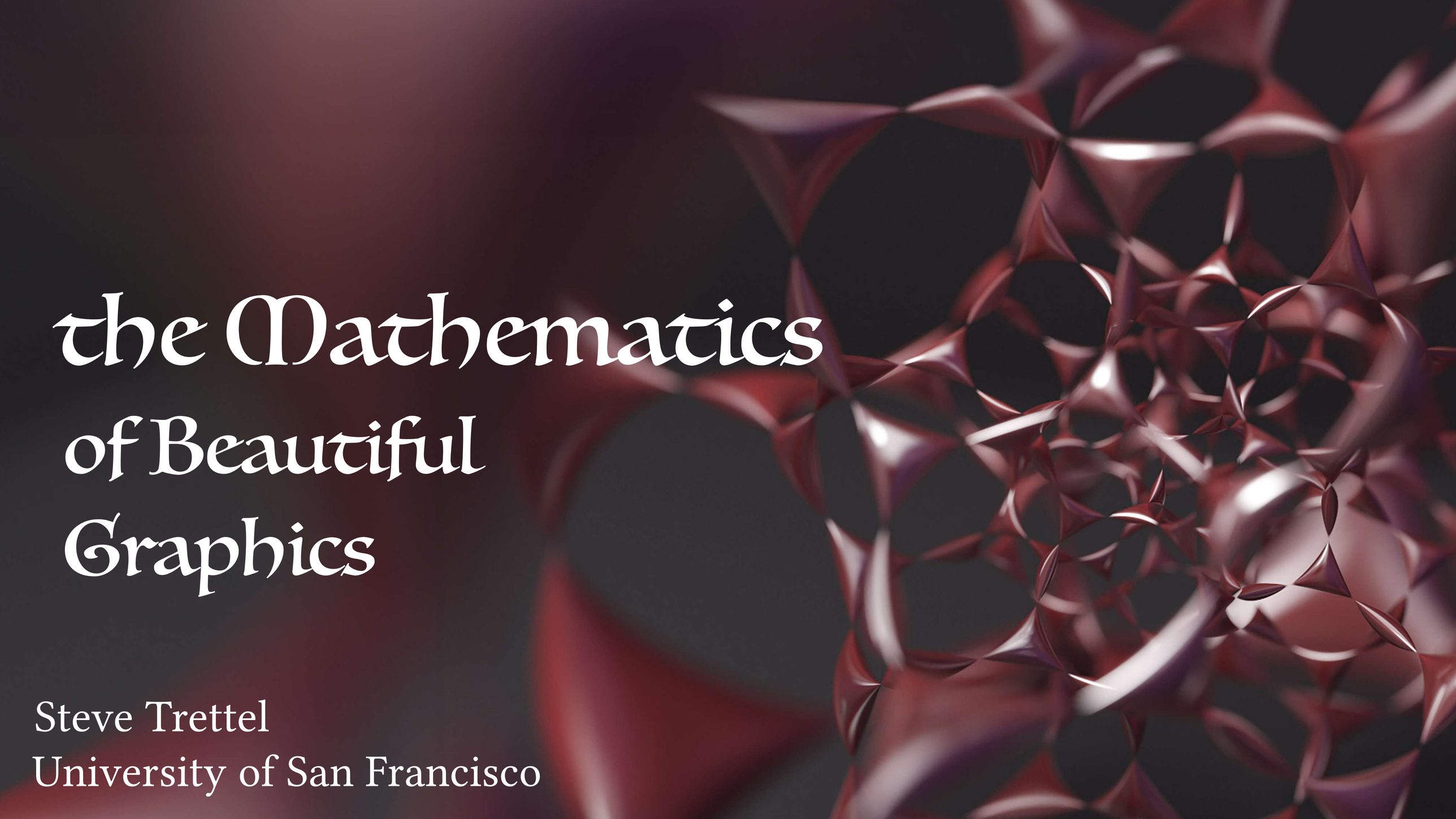
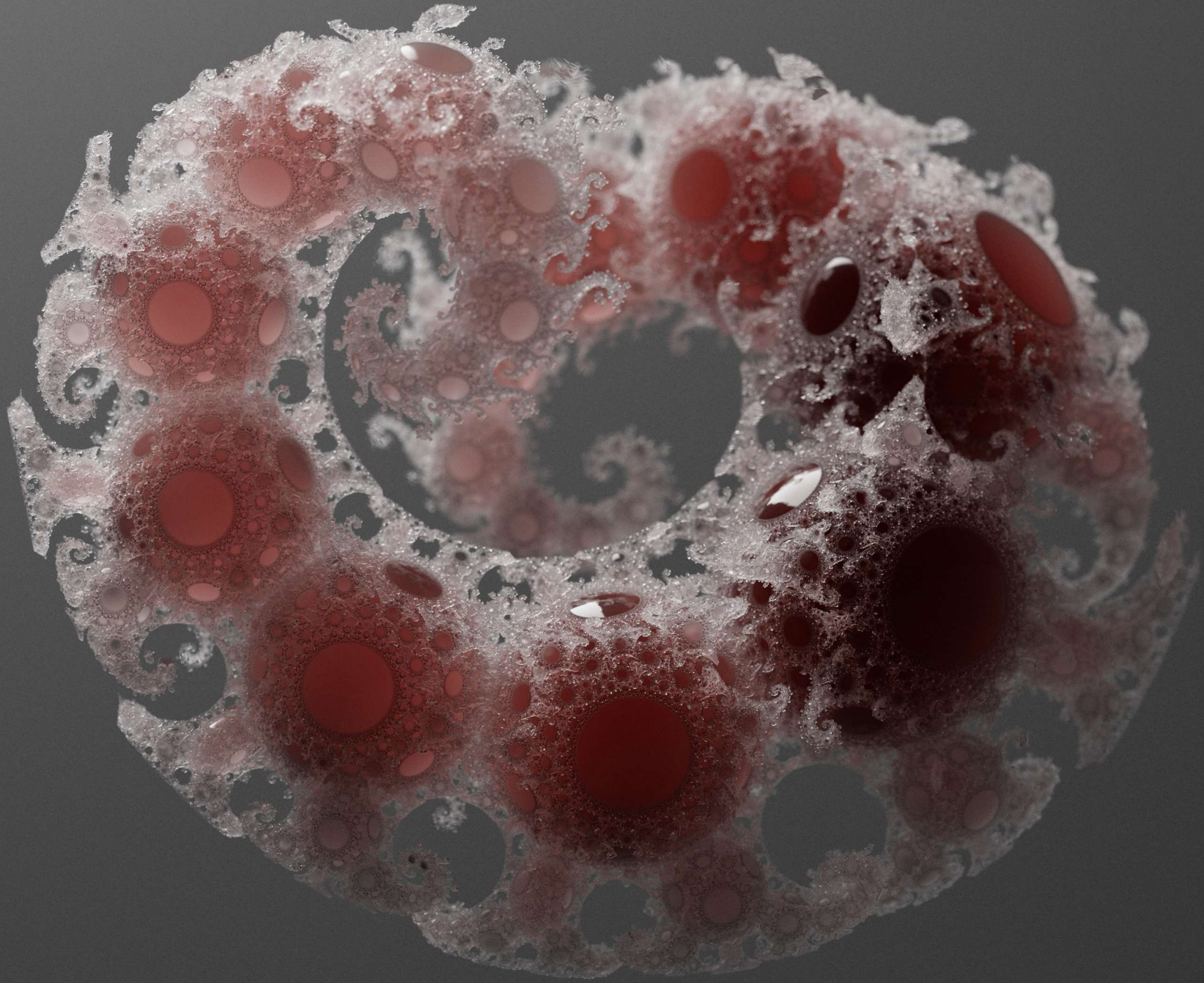
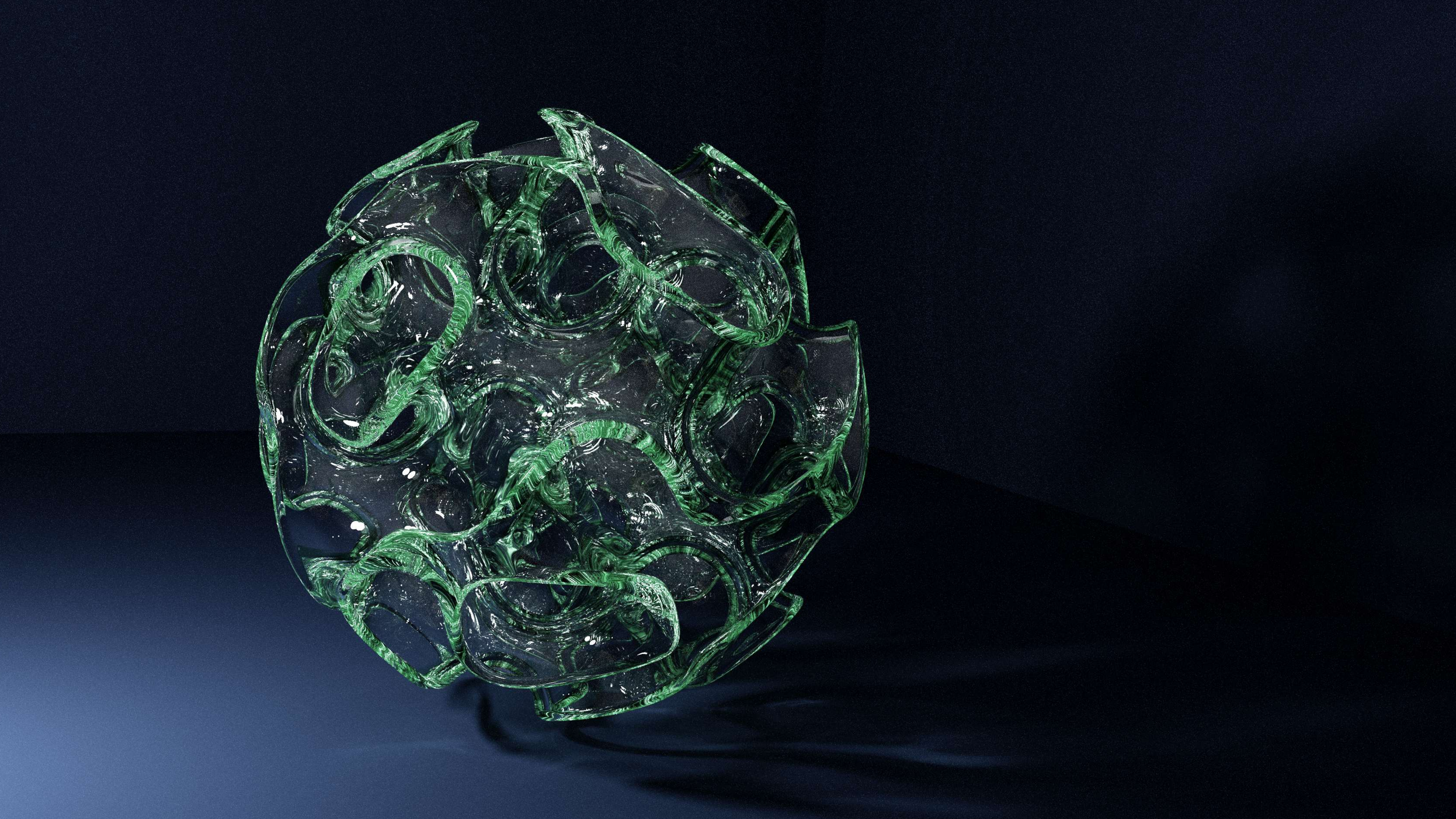
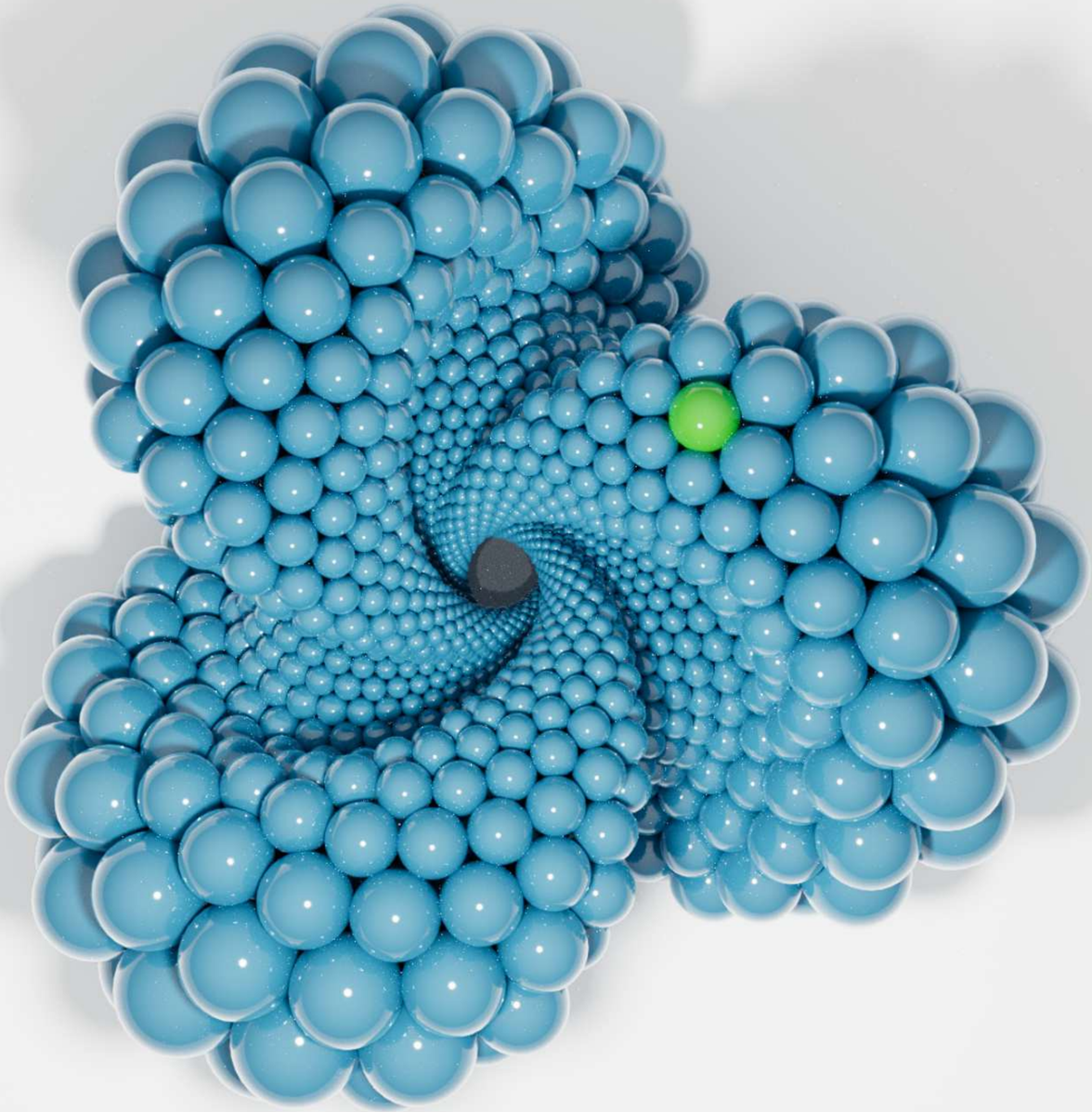


the Mathematics of Beautiful Graphics

Steve Trettel
University of San Francisco





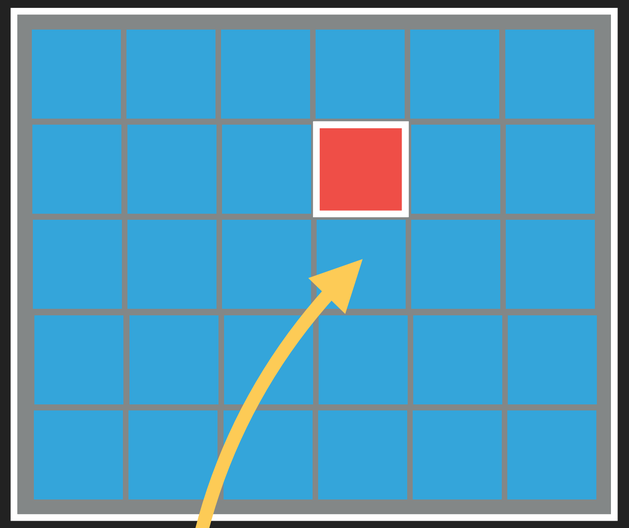


Idea 1:

Pixel by Pixel

Simulating Light Rays by Parametric Lines


Images are
made of pixels

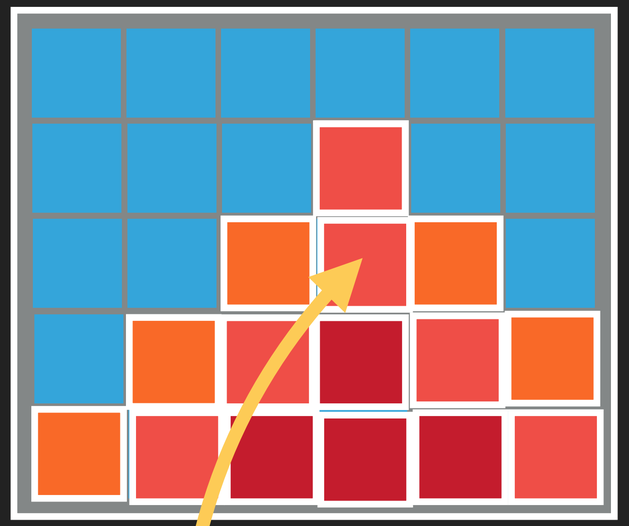


Pixel (i, j)

Computer Program

Input: Pixel Coordinates (i, j)

Output: Pixel Color (r, g, b)



Pixel (i, j)

Computer Program

Input: Pixel Coordinates (i, j)

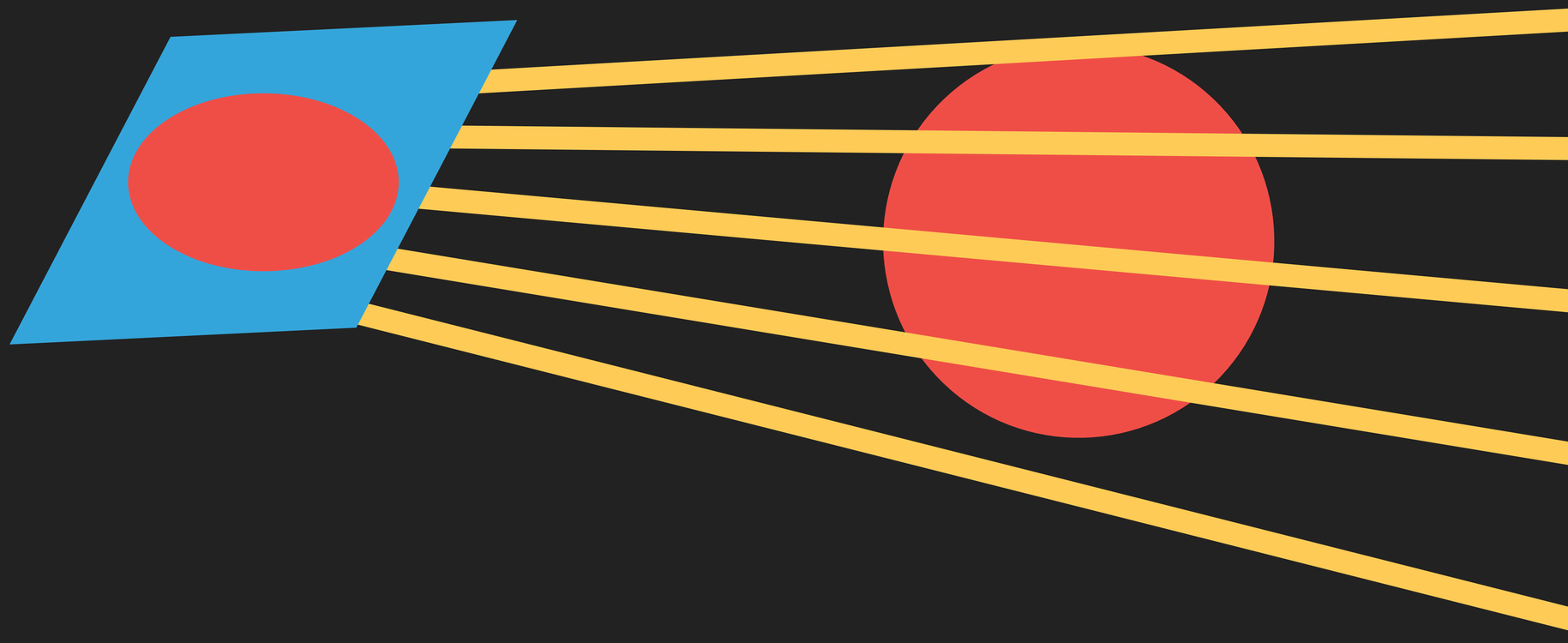
Output: Pixel Color (r, g, b)

MATH TIME!

Tracing lines in 3D just needs vector addition and scalar multiplication

$$\ell_v(t) = p + t\vec{v}$$

The color function can be built from tracing lines into a 3D scene.

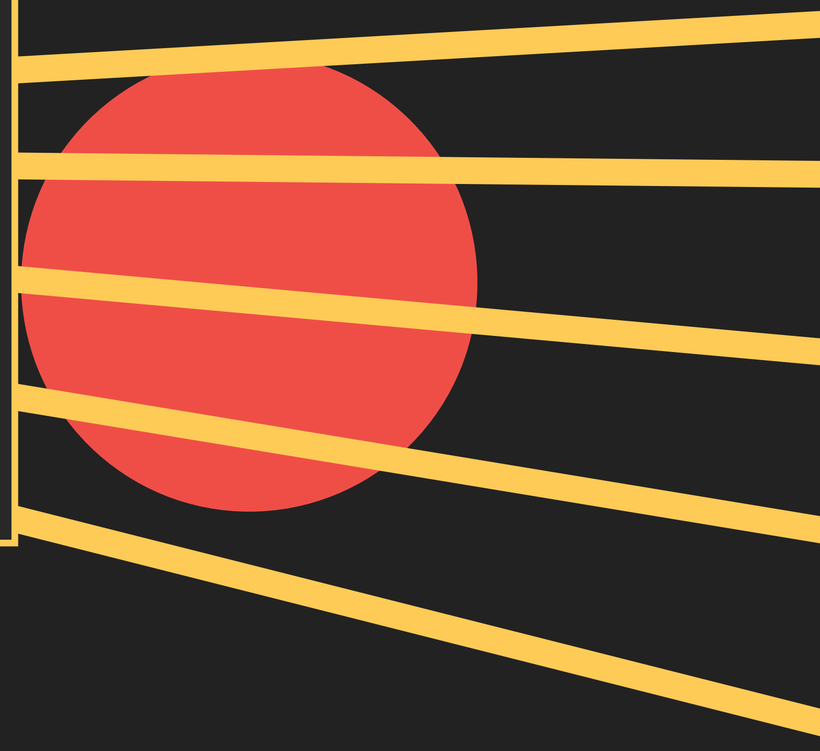


MATH TIME!

Tracing lines in 3D just needs vector addition and scalar multiplication

$$\ell_v(t) = p + t\vec{v}$$

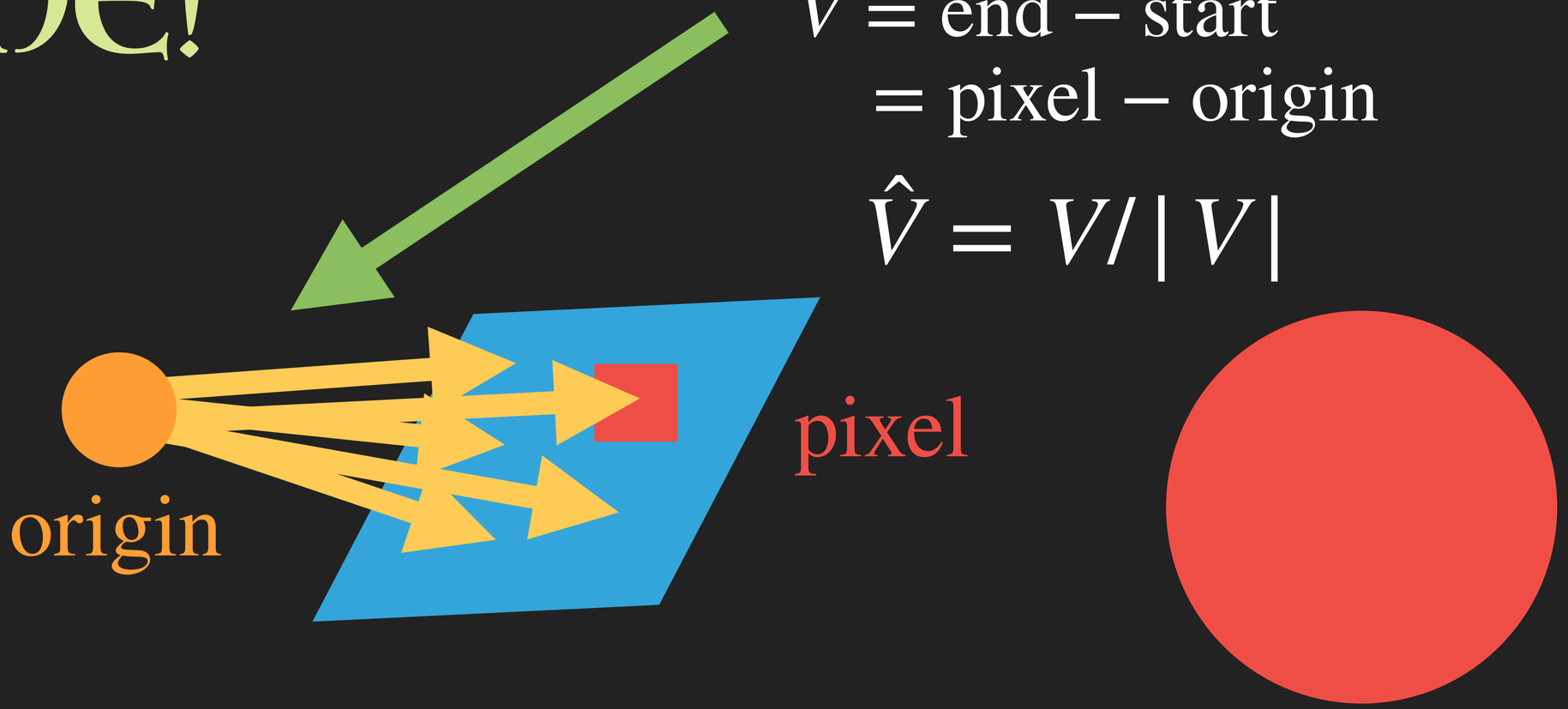
```
void flow( Vector vec, float t ){

    vec.pos += t* vec.dir;
}
```

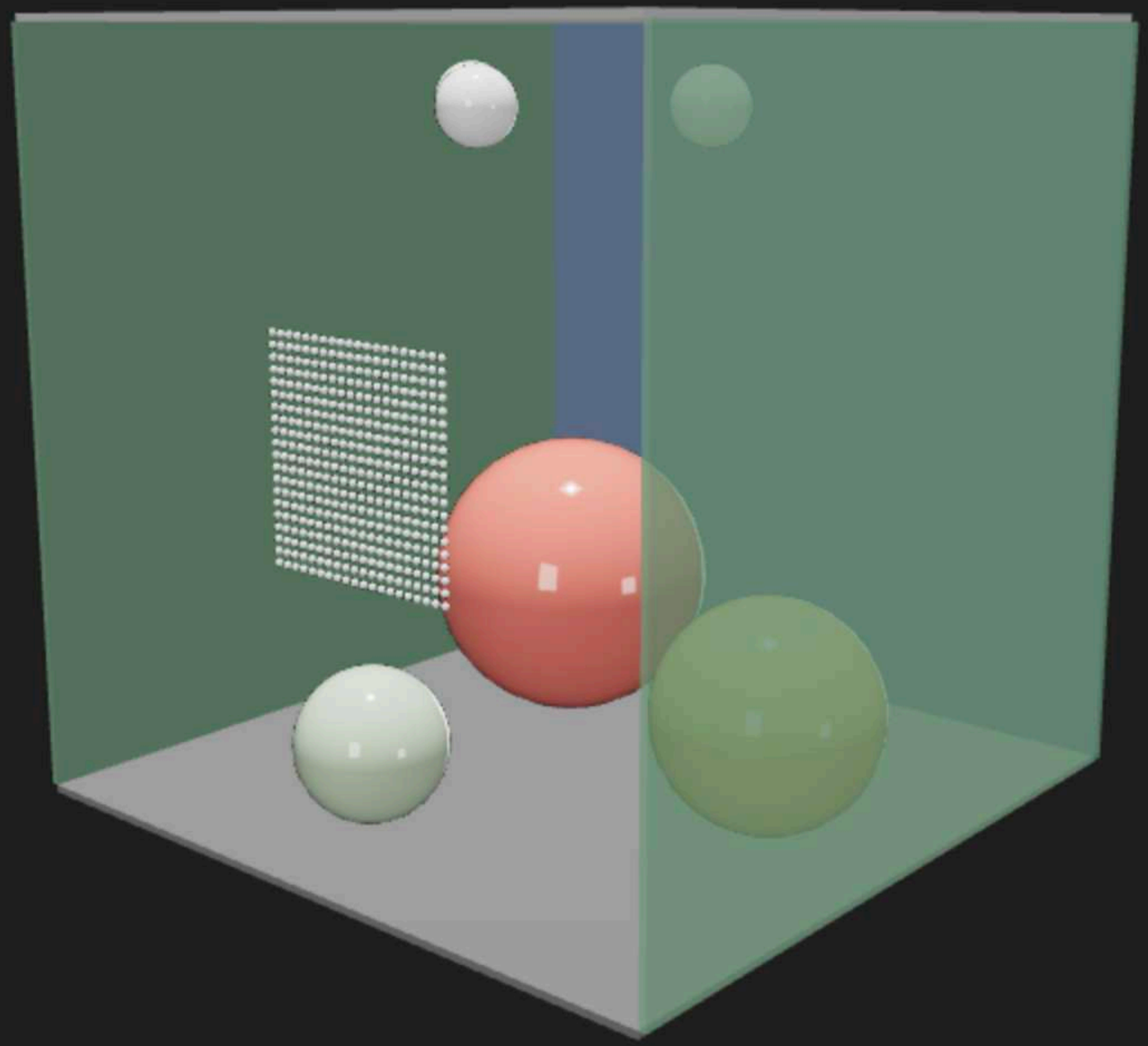


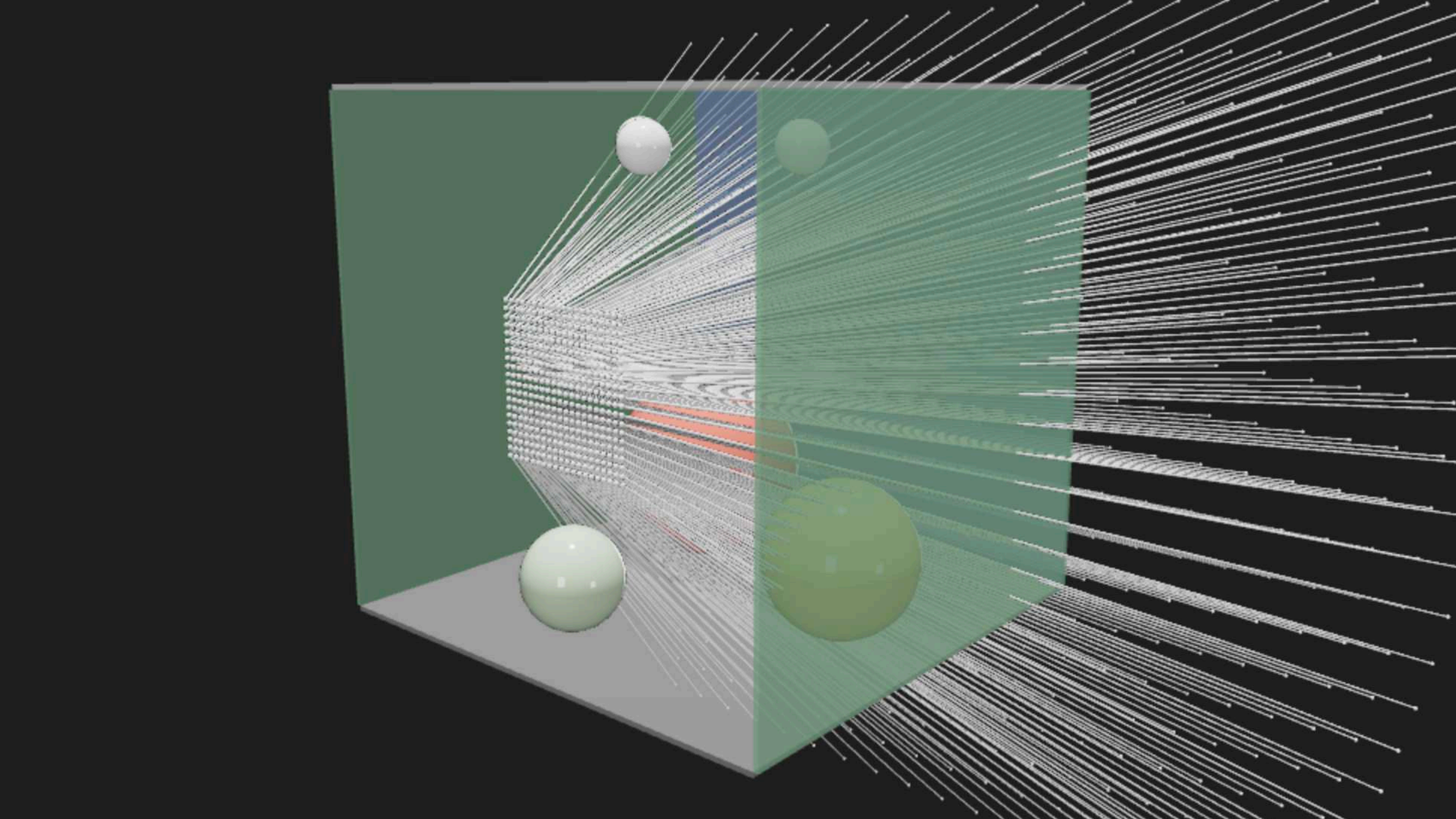
MATH TIME!

Calculating the initial direction just requires vector subtraction and forming unit vectors

$$V = \text{end} - \text{start}$$
$$= \text{pixel} - \text{origin}$$
$$\hat{V} = V/|V|$$







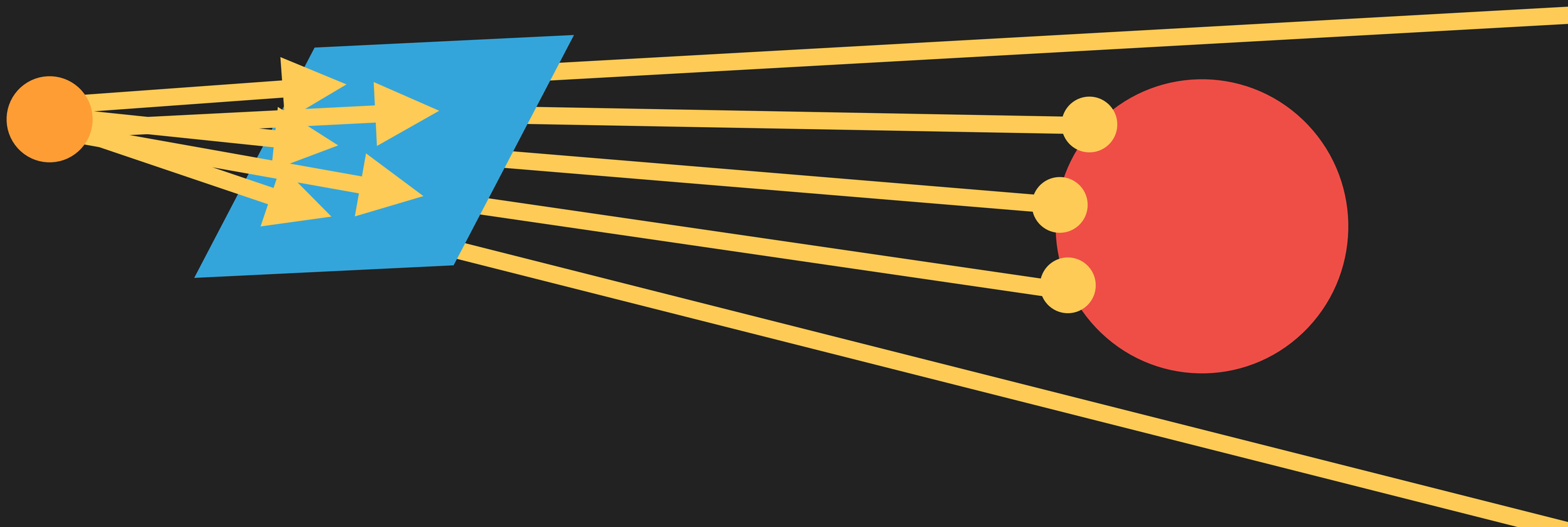
Idea II:

Where to Stop

Intersections between light rays and objects

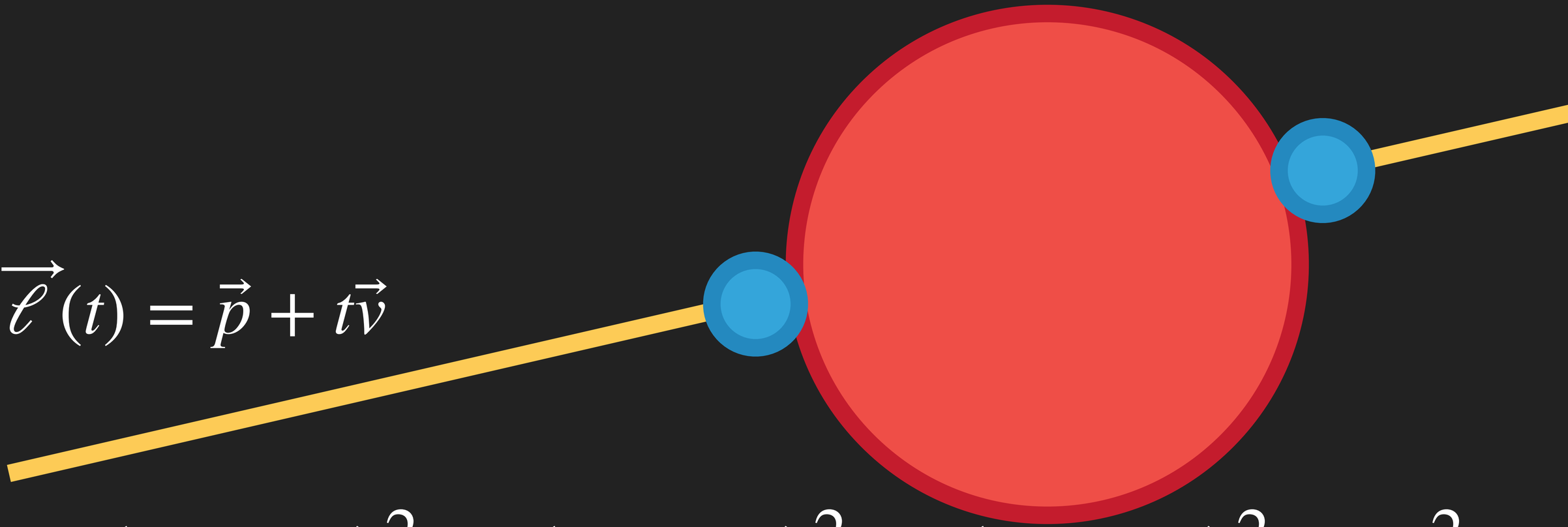
How do we stop when we hit an object?

How do we stop when we hit an object?



Using the vector equation of our line to find intersections

$$f(\ell(t)) = R^2$$



$$\overrightarrow{\ell}(t) = \vec{p} + t\vec{v}$$

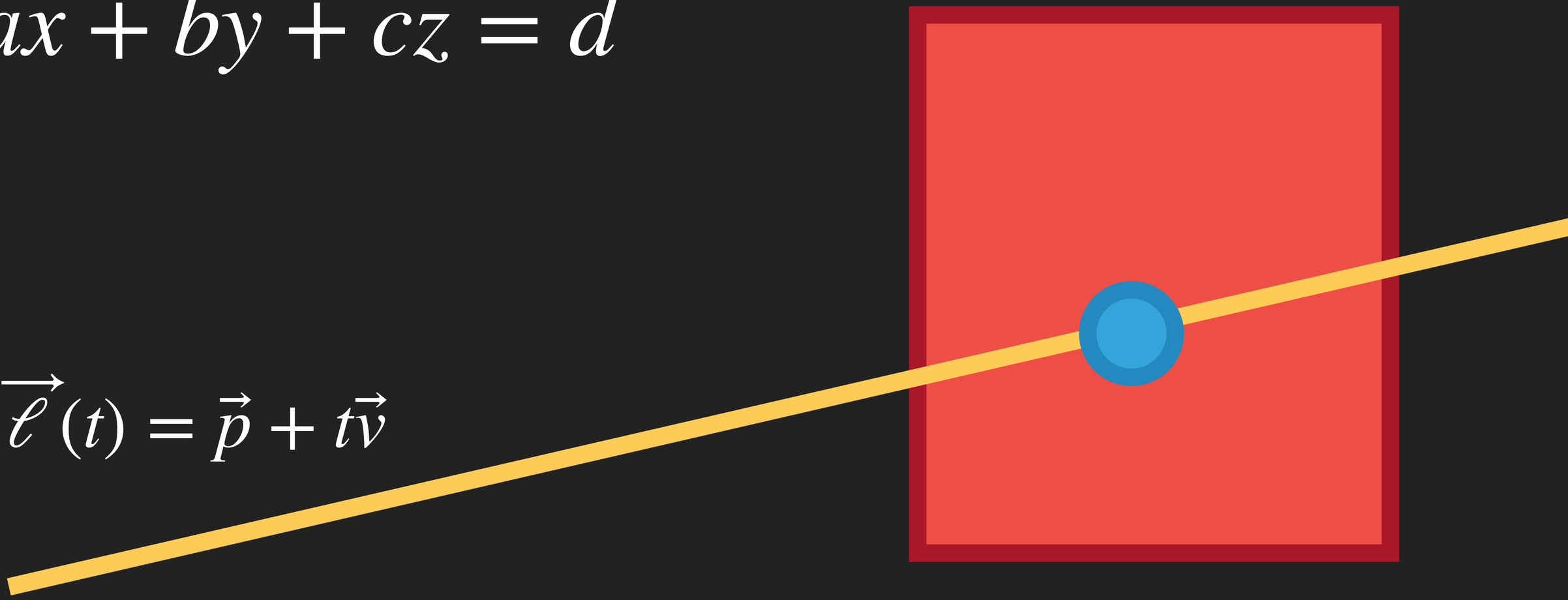
$$(x - c_1)^2 + (y - c_2)^2 + (z - c_3)^2 = r^2$$

Distance to a plane

$$ax + by + cz = d$$

$$f(\ell(t)) = d$$

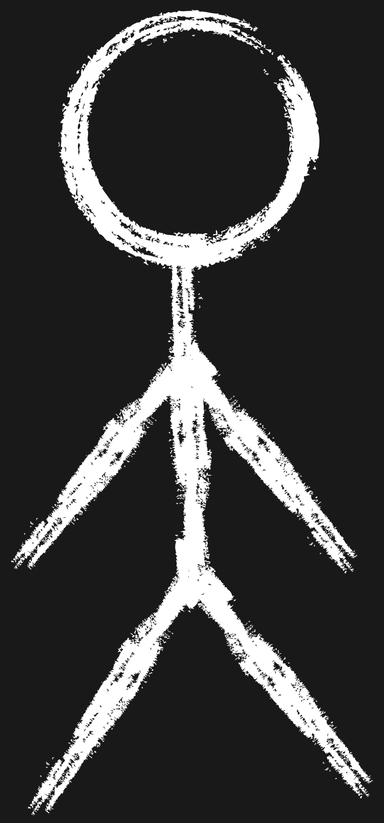
$$\overrightarrow{\ell}(t) = \vec{p} + t\vec{v}$$



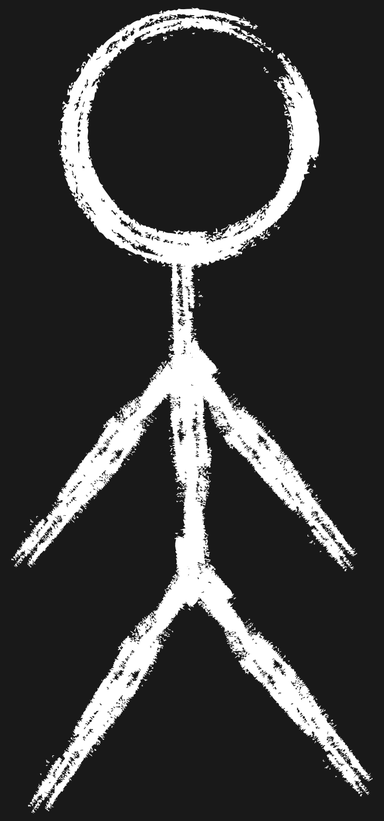
Using more complicated equations

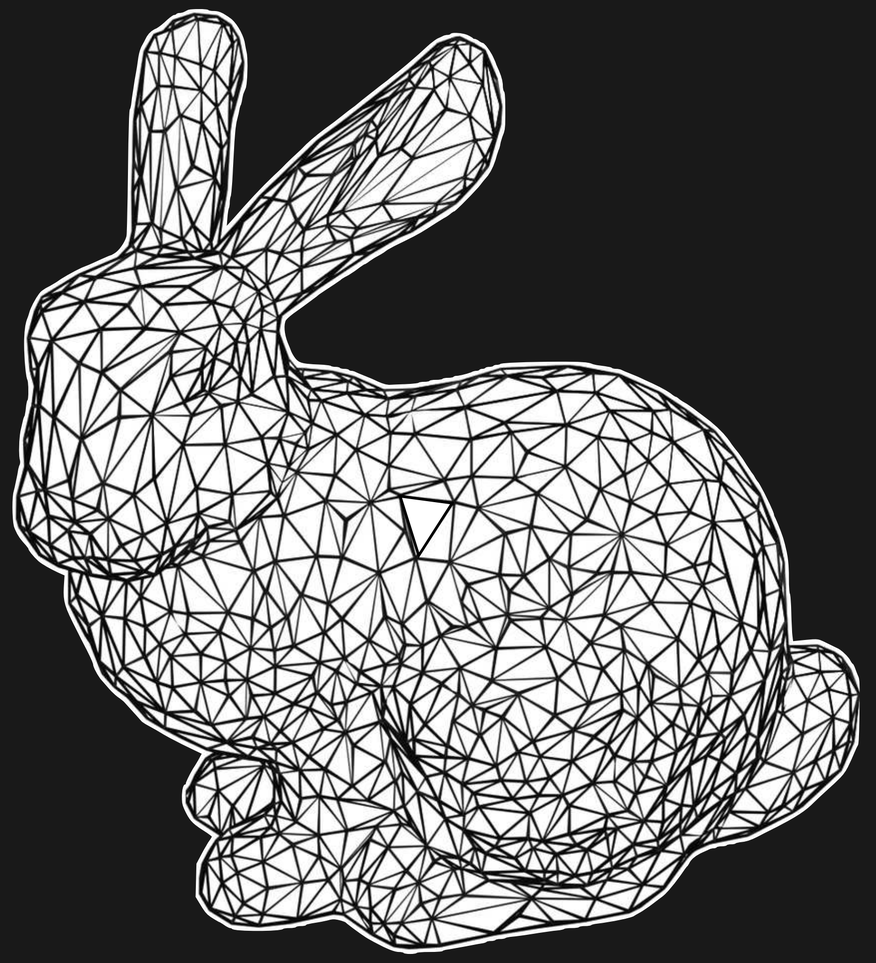
$$64(x-w)[x^4-4x^3w-10x^2y^2-4x^2w^2$$
$$+16xw^3-20xy^2w+5y^4+16w^4-20y^2w^2]$$
$$-5\sqrt{5-\sqrt{5}}\left(2z-\sqrt{5-\sqrt{5}}w\right)\left[4(x^2+y^2+z^2)+(1+3\sqrt{5})w^2\right]^2$$
$$=0$$

Using Lots of Planes

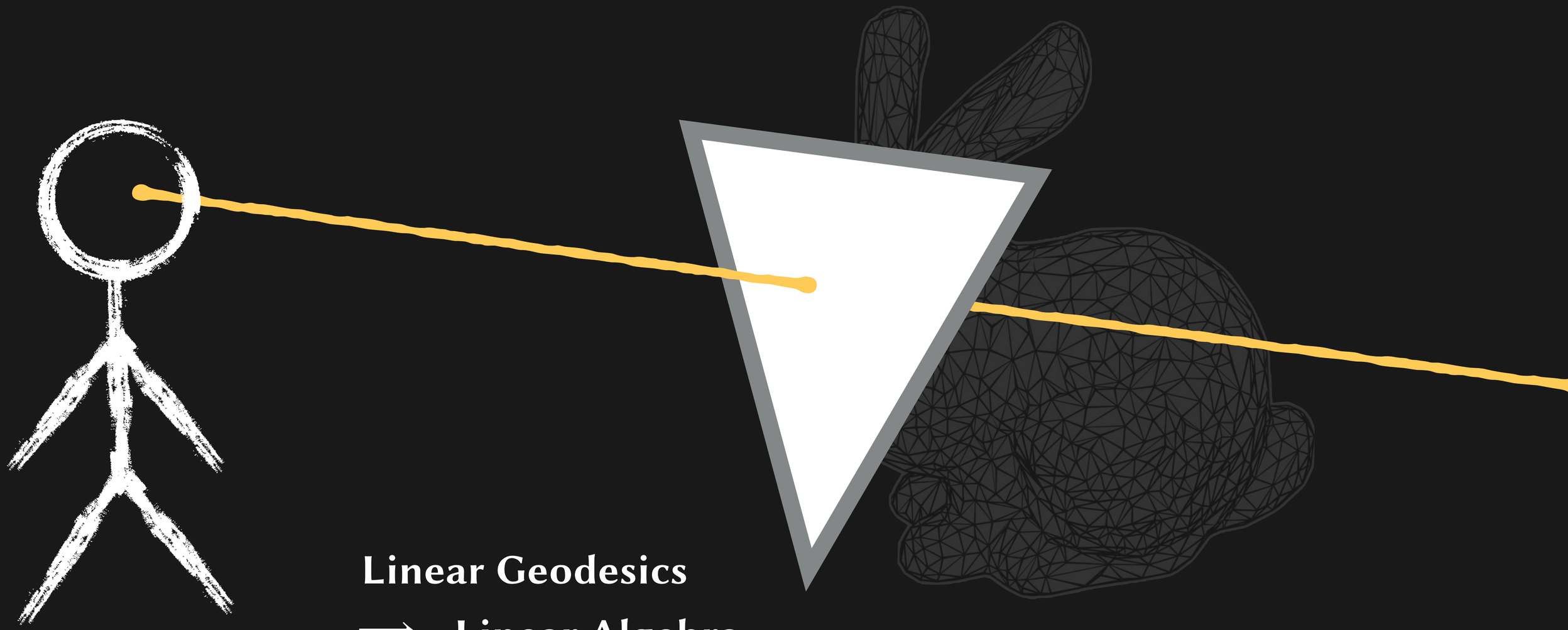


Using Lots of Planes





Using Lots of Planes



Linear Geodesics

$\Longrightarrow$ **Linear Algebra**

A simple scene:
Five walls and three balls

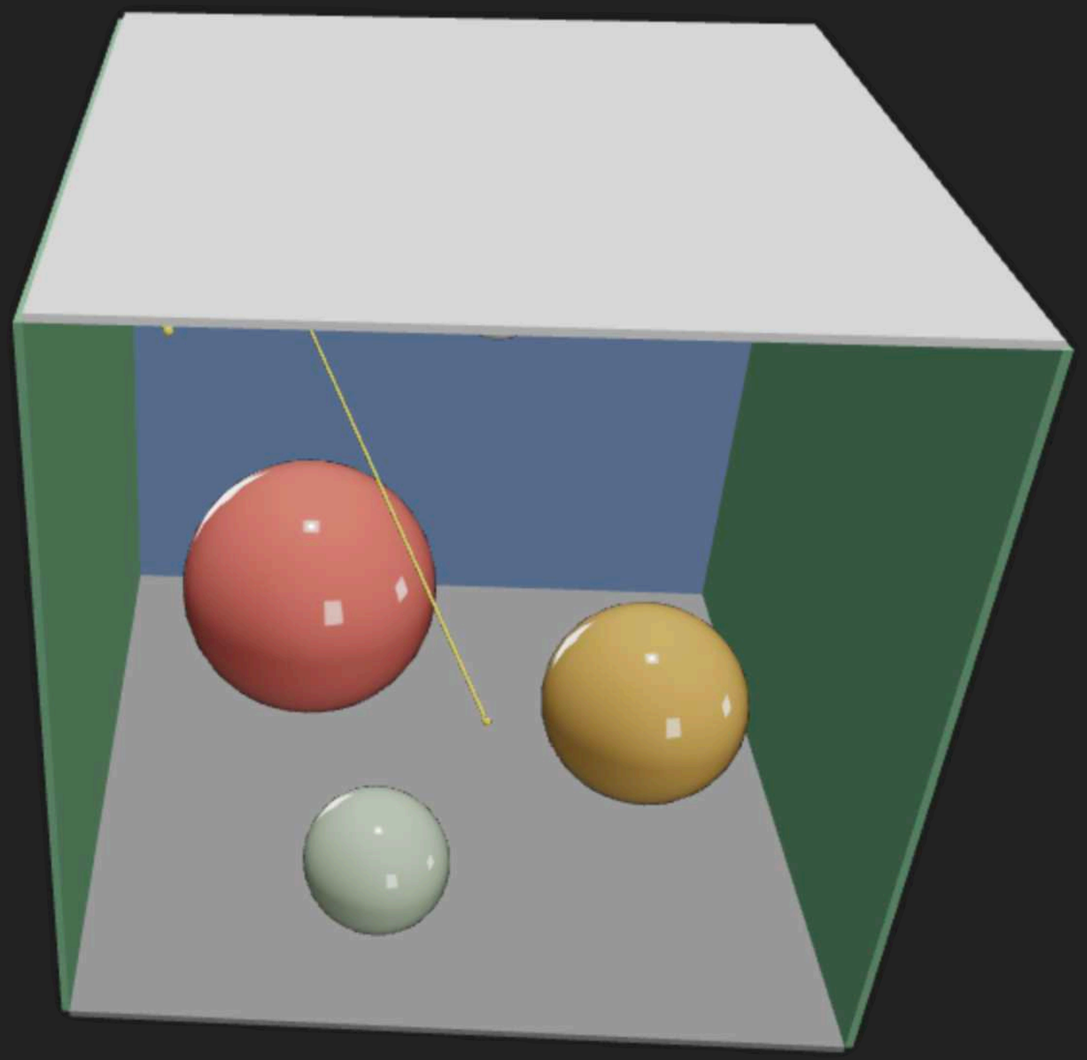
$$t_1 = \text{dist}(\text{wall}_1) \quad t_2 = \text{dist}(\text{wall}_2)$$

$$t_3 = \text{dist}(\text{wall}_3) \quad t_4 = \text{dist}(\text{wall}_4)$$

$$t_5 = \text{dist}(\text{wall}_5) \quad t_6 = \text{dist}(\text{ball}_1)$$

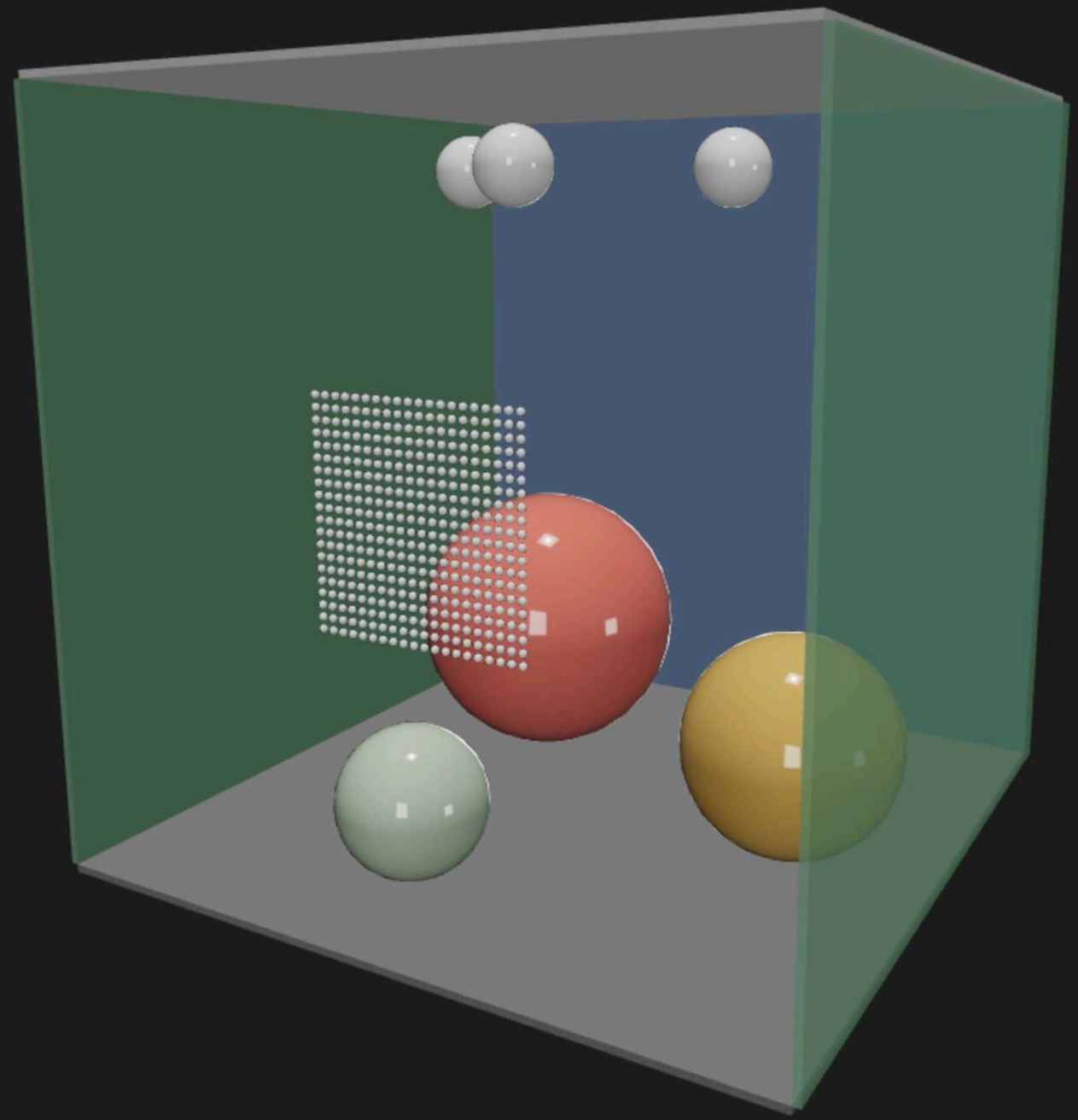
$$t_7 = \text{dist}(\text{ball}_2) \quad t_8 = \text{dist}(\text{ball}_3)$$

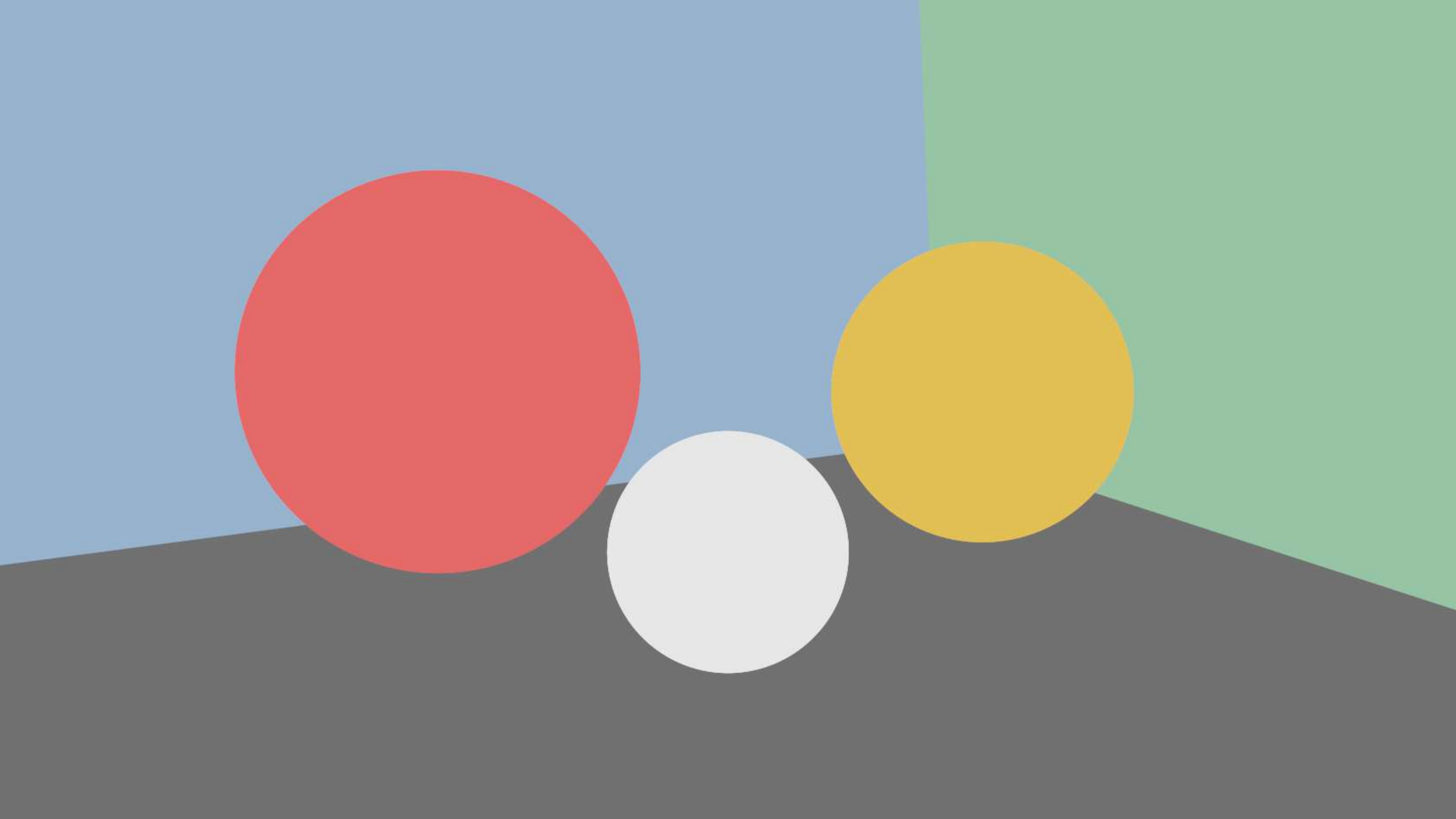
$$t = \min\{t_1, t_2, t_3, t_4, t_5, t_6, t_7, t_8\}$$



it works!

Now - just grab the color of the object we hit, and return it for that pixel?





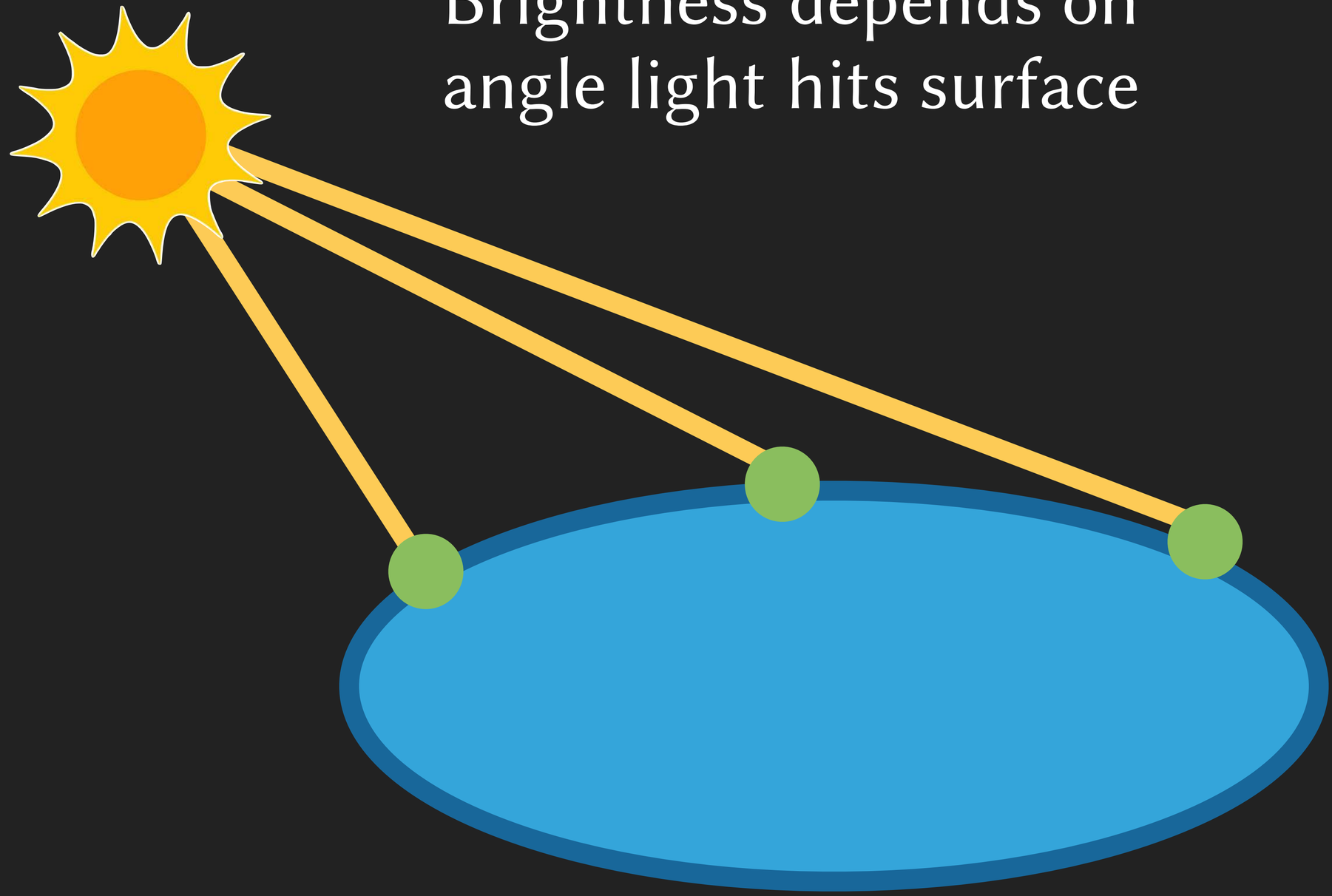
Idea III:

Finding Light

Vectors and dot products for reflections and shadows

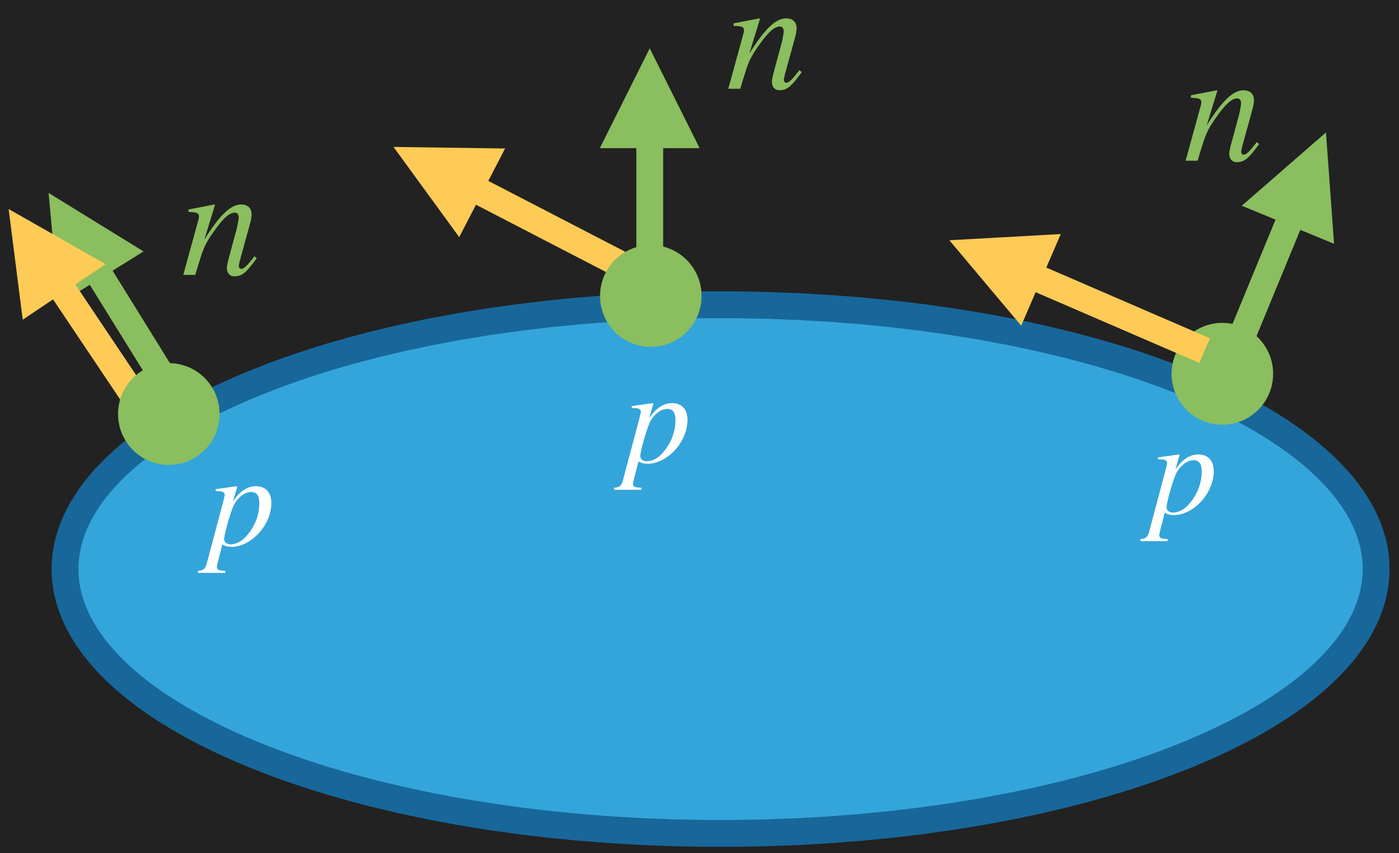
MATH TIME!

Brightness depends on angle light hits surface



MATH TIME!

Brightness depends on angle light hits surface



Brightness

$$\cos\theta = \text{toLight} \cdot n$$

$$\text{toLight} = \frac{L - p}{\|L - p\|}$$

MATH TIME

Brightness depends on angle light hits surface

Brig

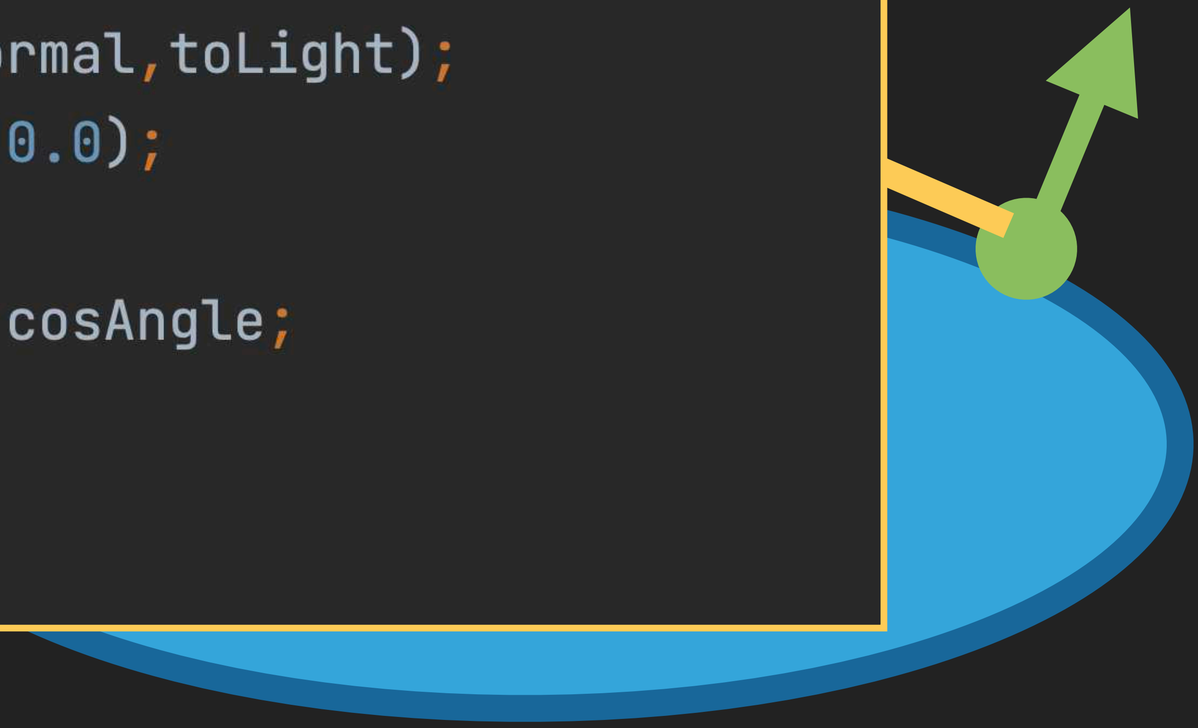
cos

toLi

toLight

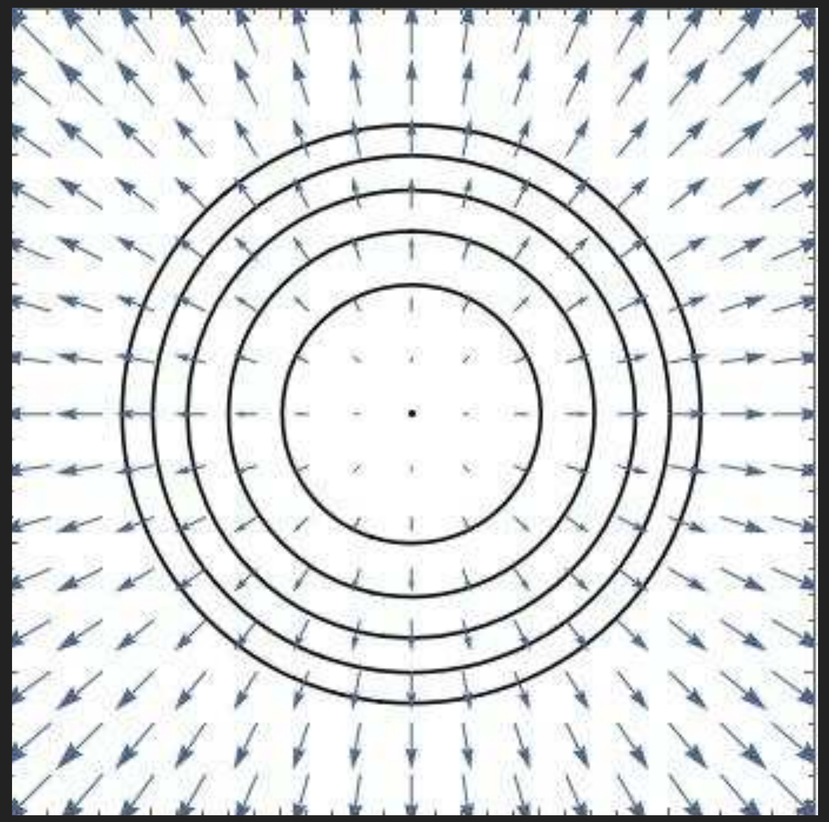
$\|L - p\|$

```
vec3 lighting( Vector normal, Vector toLight ){

    //angle
    float cosAngle = dot(normal,toLight);
    cosAngle=max(cosAngle, 0.0);

    return materialColor * cosAngle;

}
```



MATH TIME!

The normal vector is the gradient of the distance function!



$$\nabla f = \left\langle \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \right\rangle$$

$$D_{\vec{v}} f = \nabla f \cdot v$$

$$0 = \nabla f \cdot v$$

$$\nabla f \perp v$$

MATH TIME

```
Vector normal( Vector pos ){

    ex = vec3(epsilon, 0, 0);
    ey = vec3(0, epsilon, 0);
    ez = vec3(0, 0, epsilon);

    dx = distance(pos + ex) - distance(pos);
    dy = distance(pos + ey) - distance(pos);
    dz = distance(pos + ez) - distance(pos);

    Vector gradient = Vector(dx,dy,dz);

    return normalize(gradient);

}
```

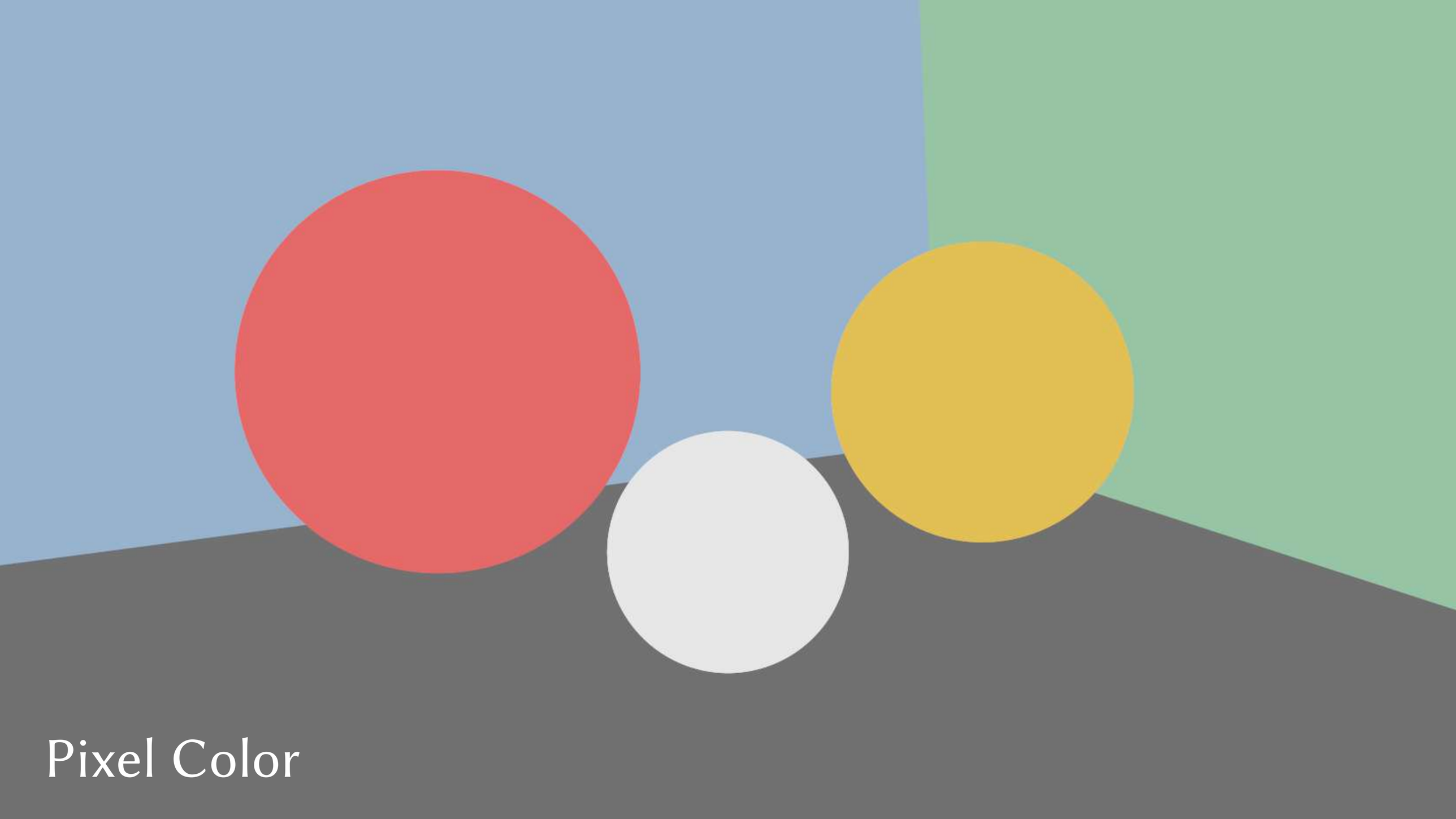
adient

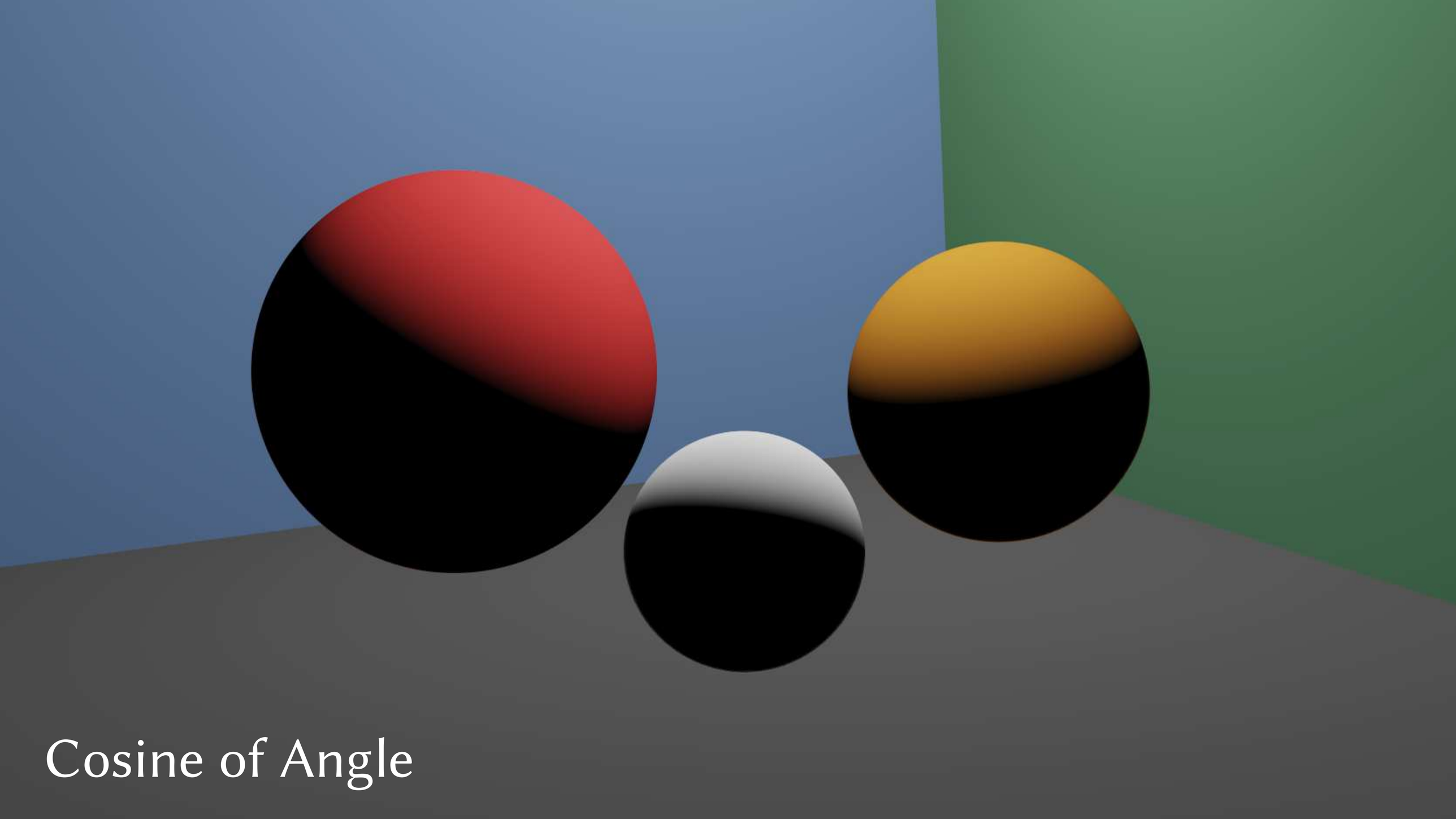
$$\left\langle \frac{\partial f}{y}, \frac{\partial f}{\partial z} \right\rangle$$

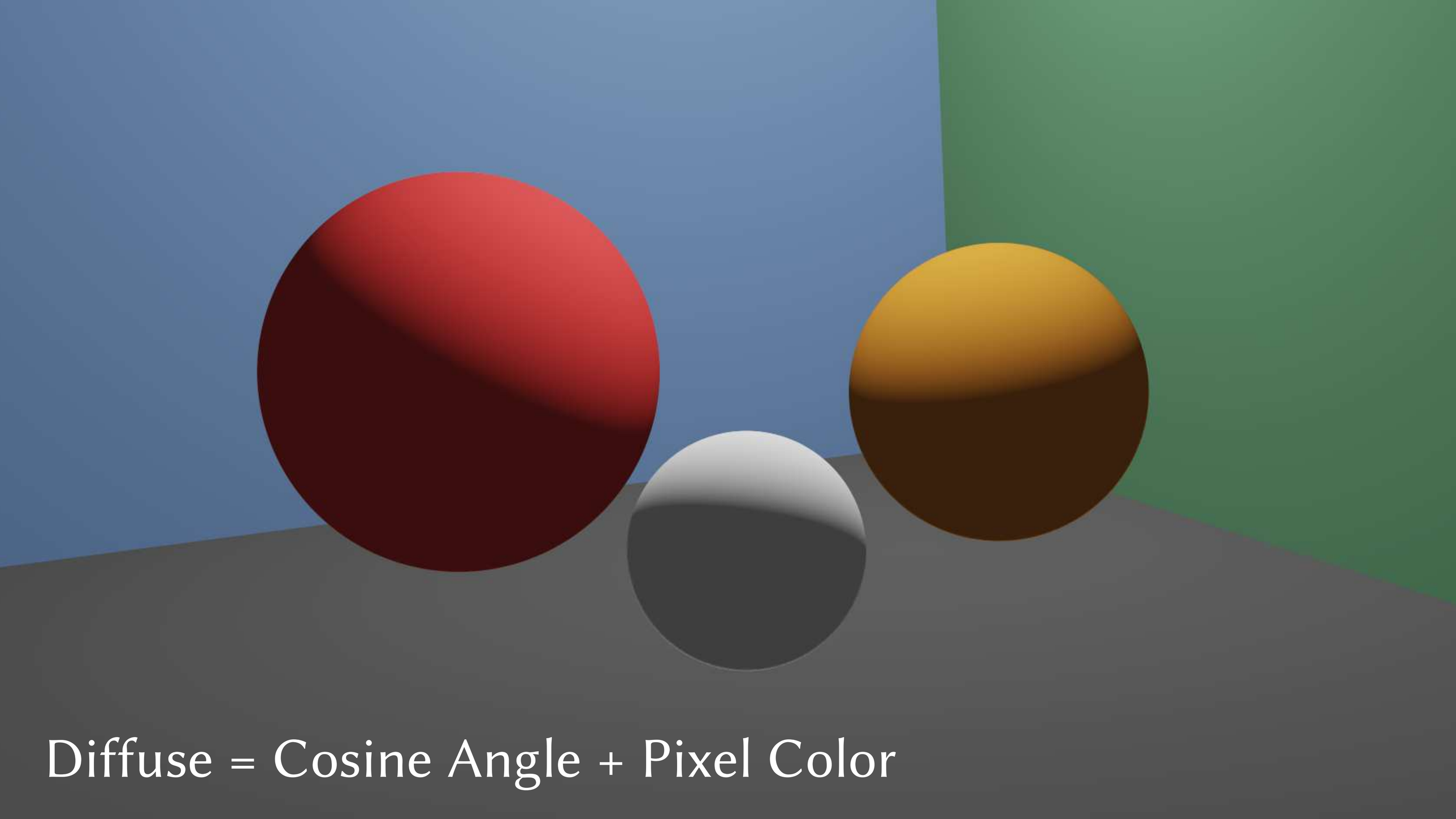
$f \cdot v$

$f \cdot v$

v


Pixel Color


Cosine of Angle


Diffuse = Cosine Angle + Pixel Color

This is still not looking that realistic.... What are we missing?

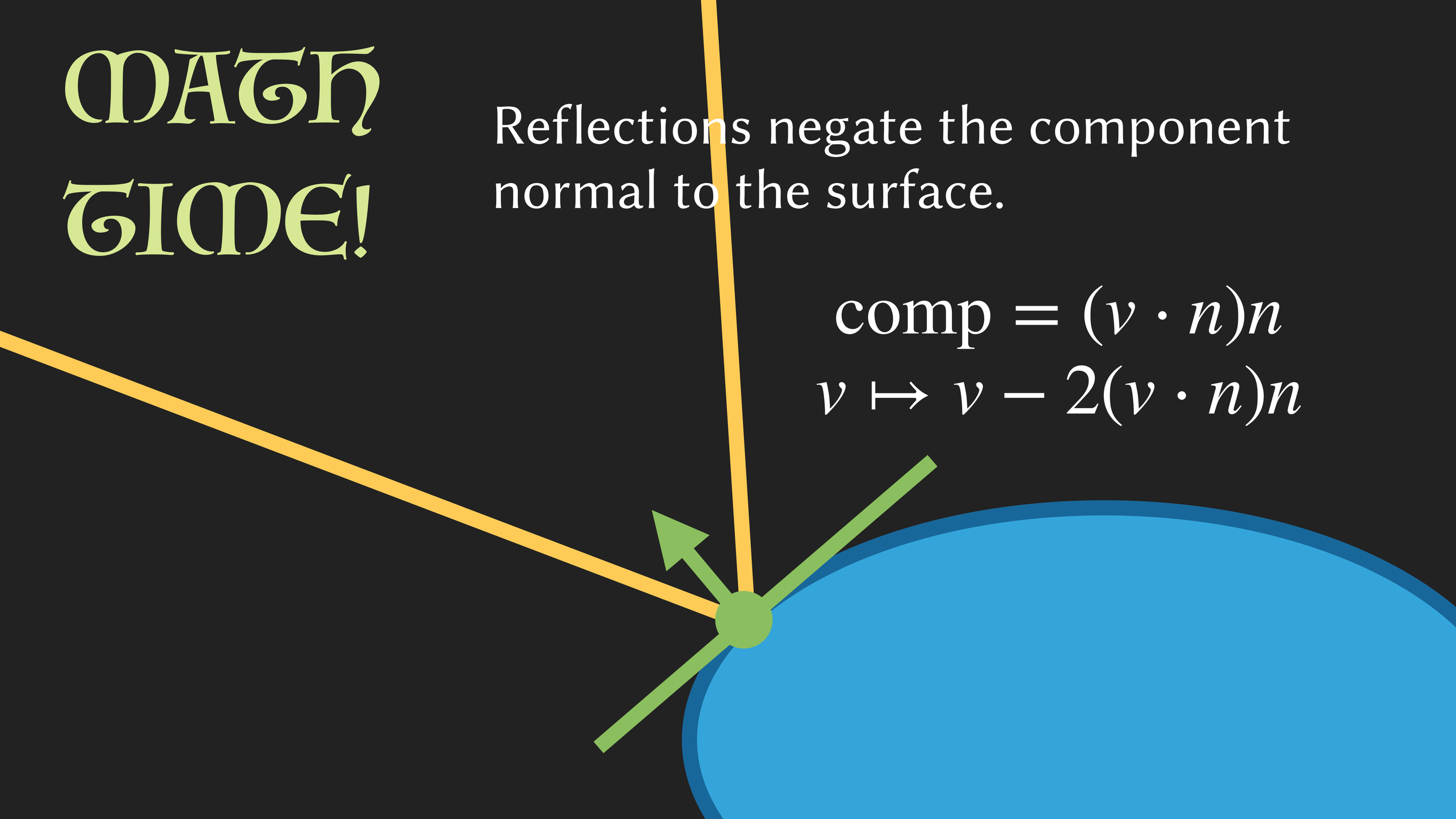
Diffuse = Cosine Angle + Pixel Color

MATH TIME!

Reflections negate the component normal to the surface.

$$\text{comp} = (v \cdot n)n$$

$$v \mapsto v - 2(v \cdot n)n$$



MATH TIC

Reflections negate the component

```
Vector reflect(Vector v, Vector n){

    Vector comp = 2.* dot(v, n) * n;
    Vector refl = subtract(v, comp);

    return refl;

}
```

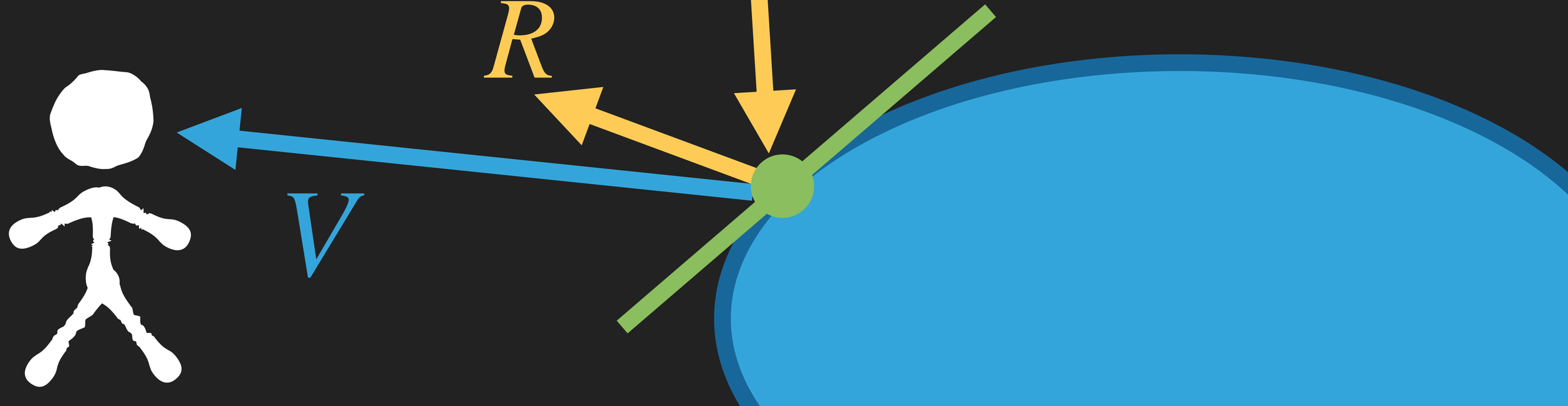
n

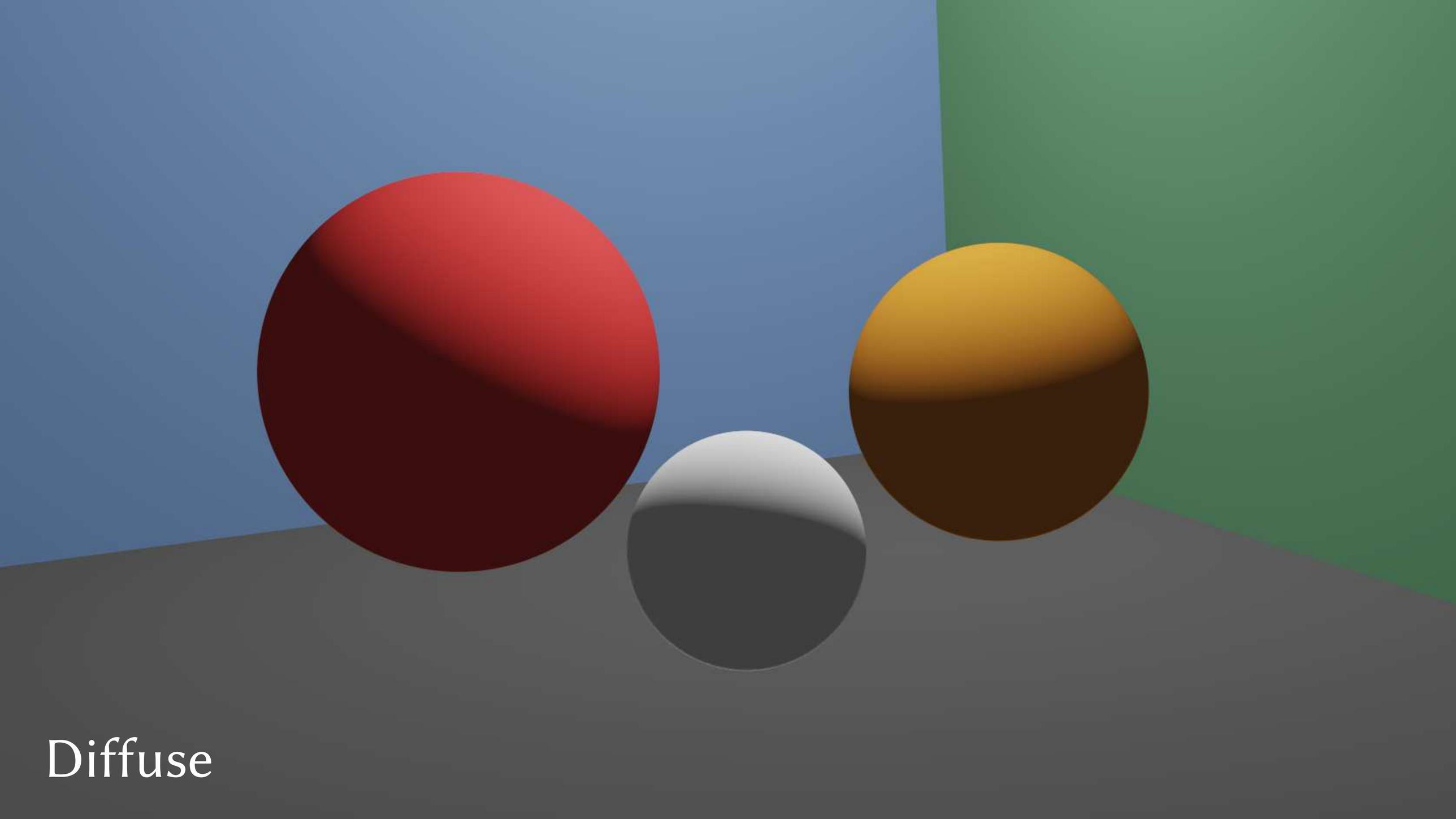
$)n$

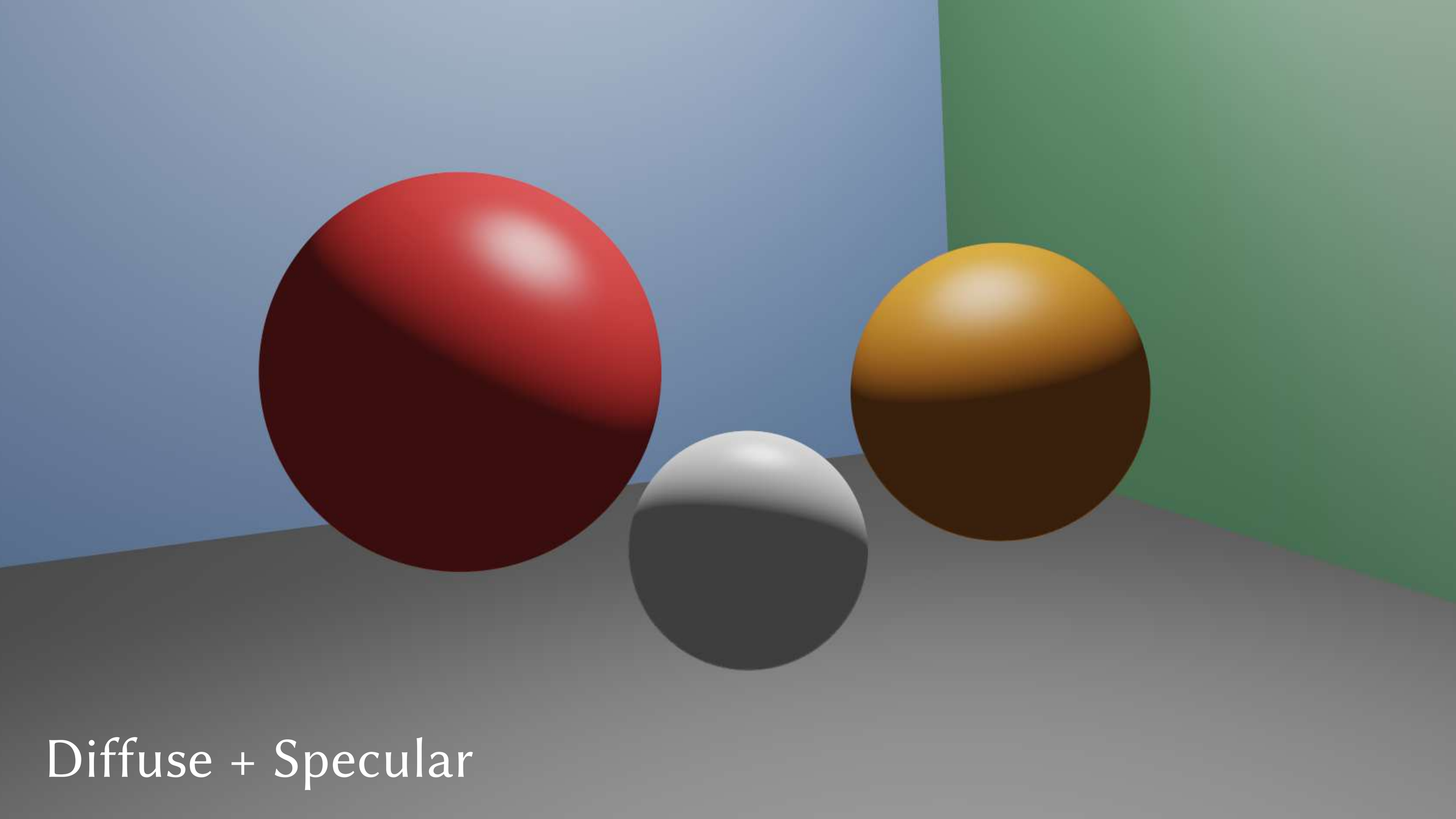
MATH TIME!

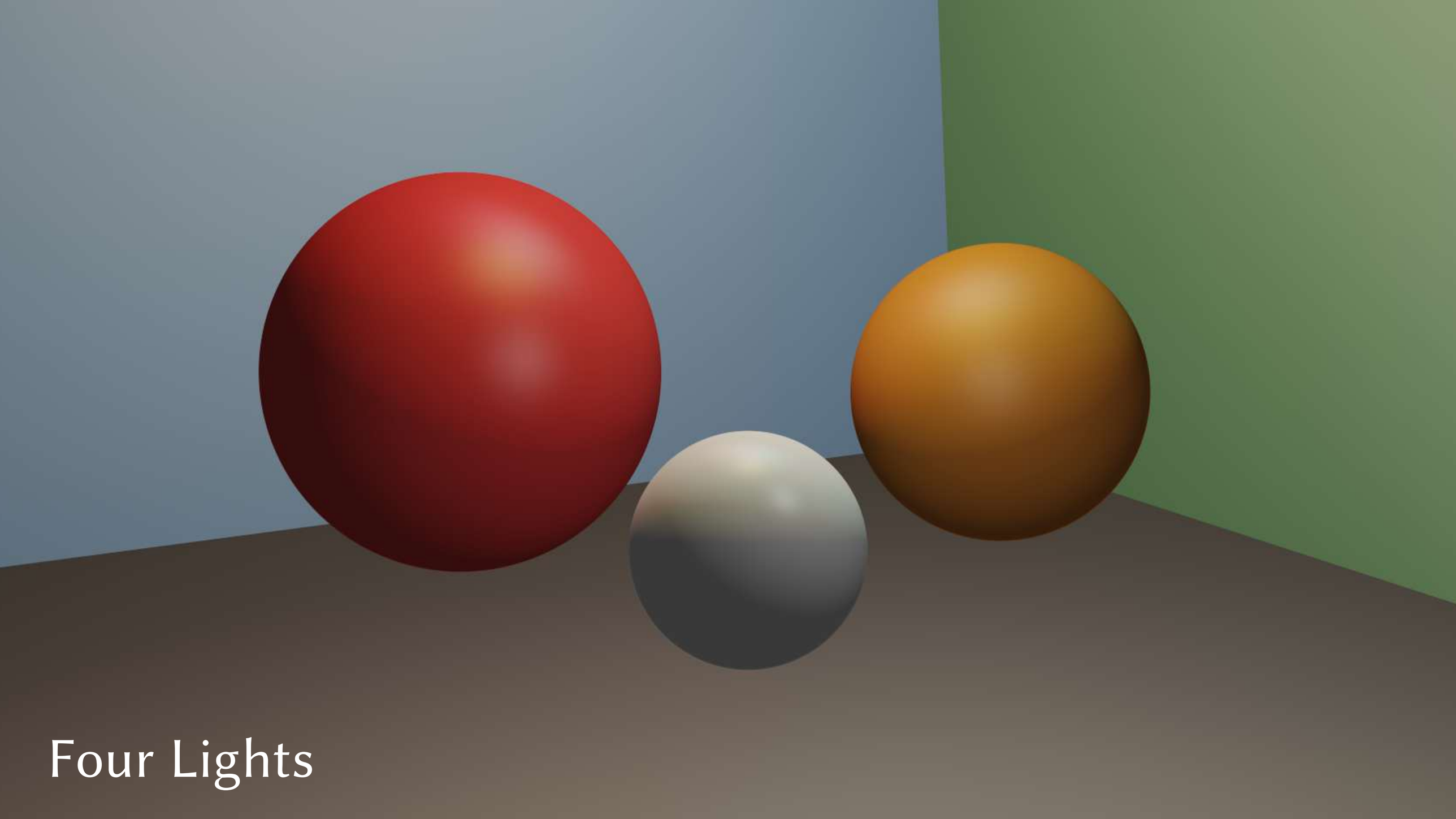
Specularity is proportional to how closely R and V are aligned

$$\text{Spec} = \hat{R} \cdot \frac{V}{\|V\|}$$




Diffuse

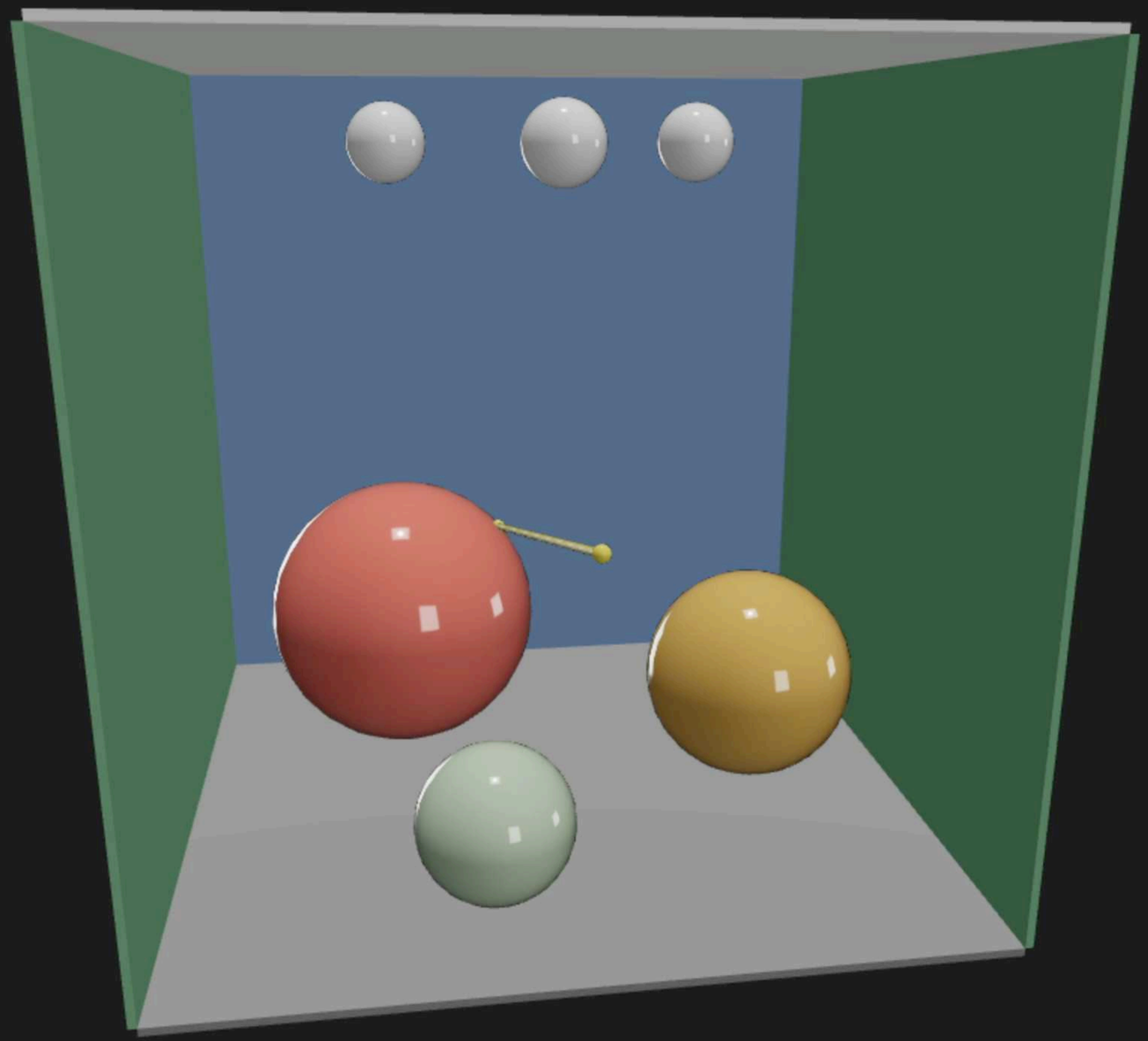

Diffuse + Specular


Four Lights

Idea IV:

Adding it all Up

Realistic lighting and integration

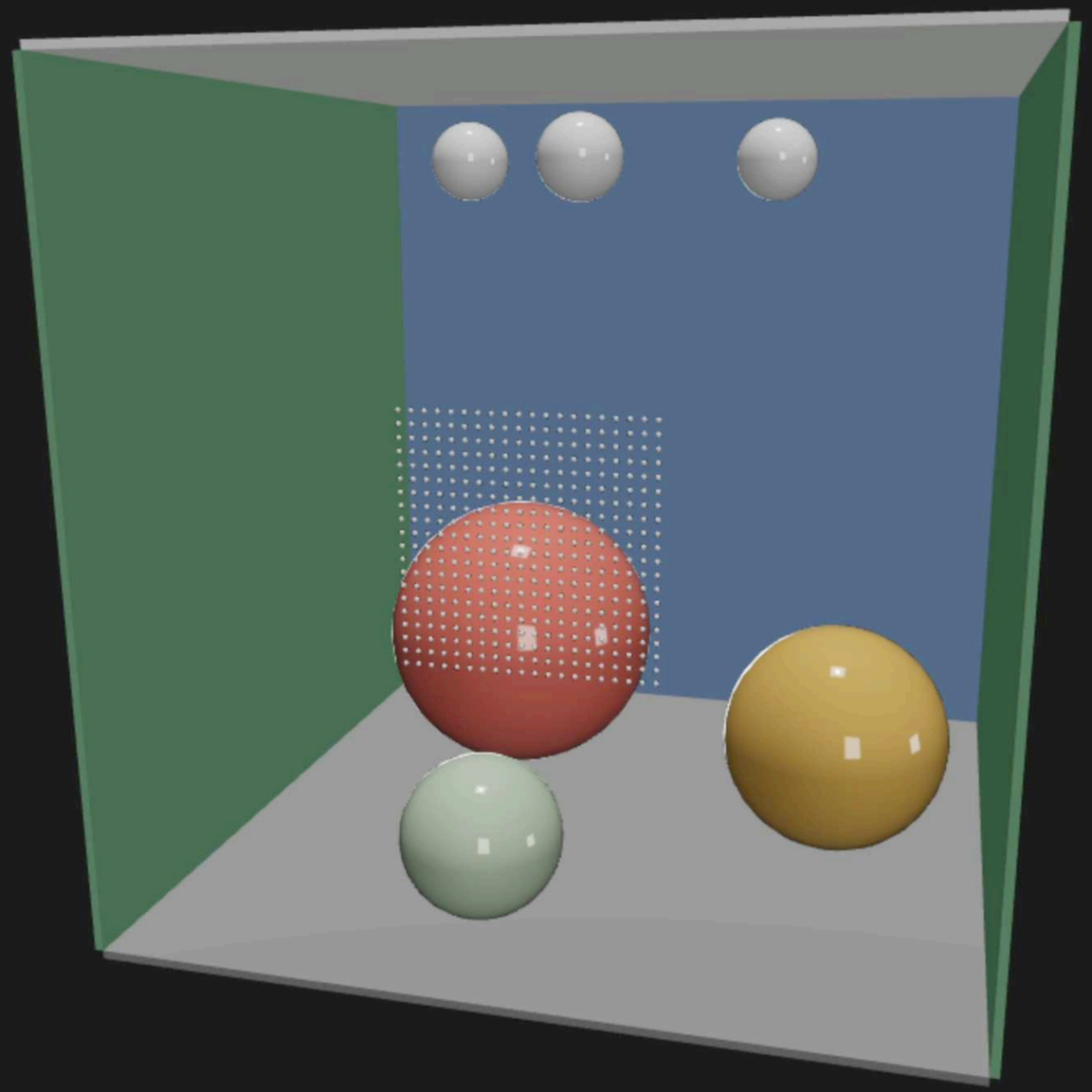


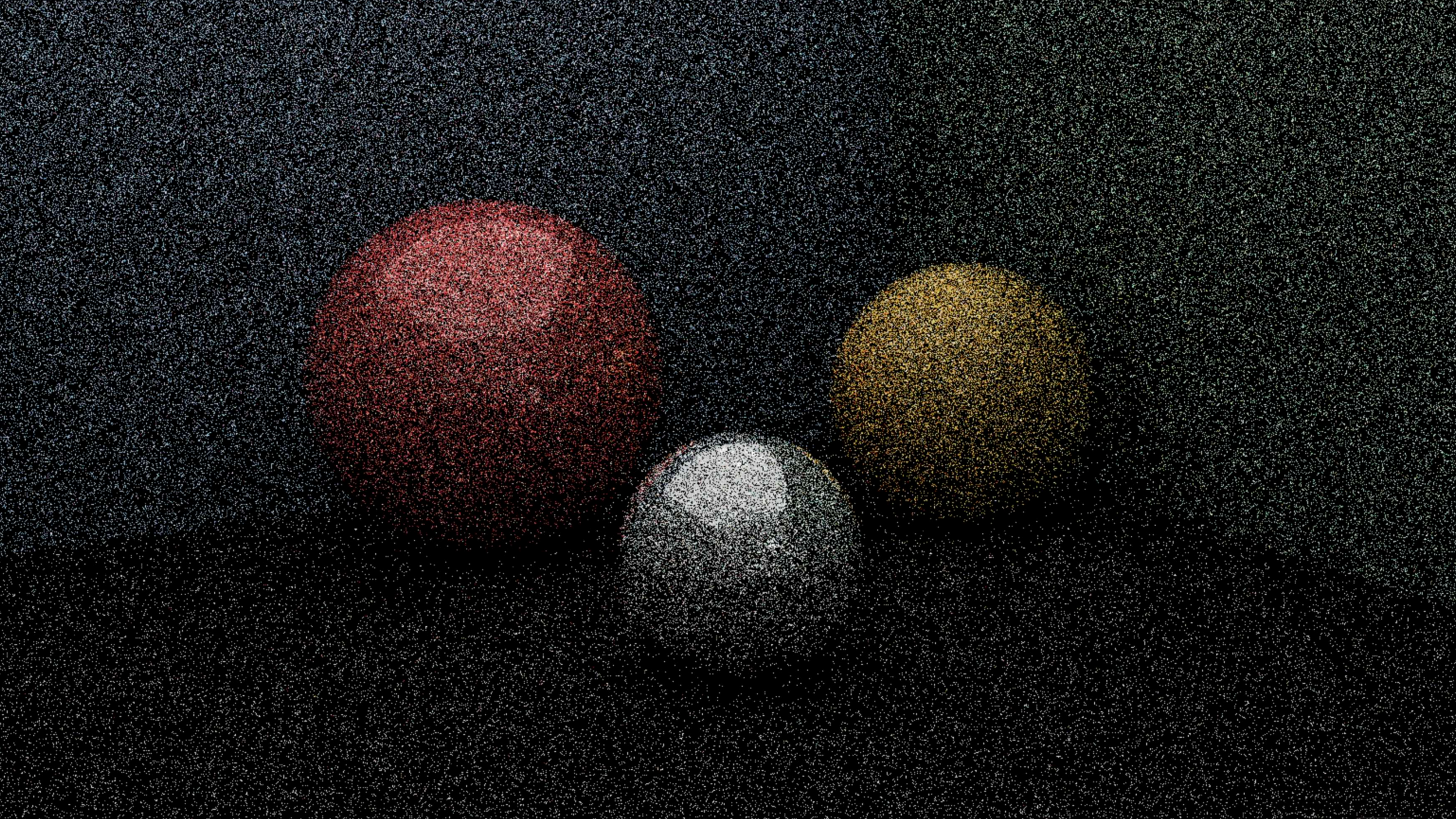
Reflections require us to continue the ray....

Idea: what if we just kept going, until we hit a light source?

Simulation:

Send out one such trace per pixel and collect colors of objects hit along the way





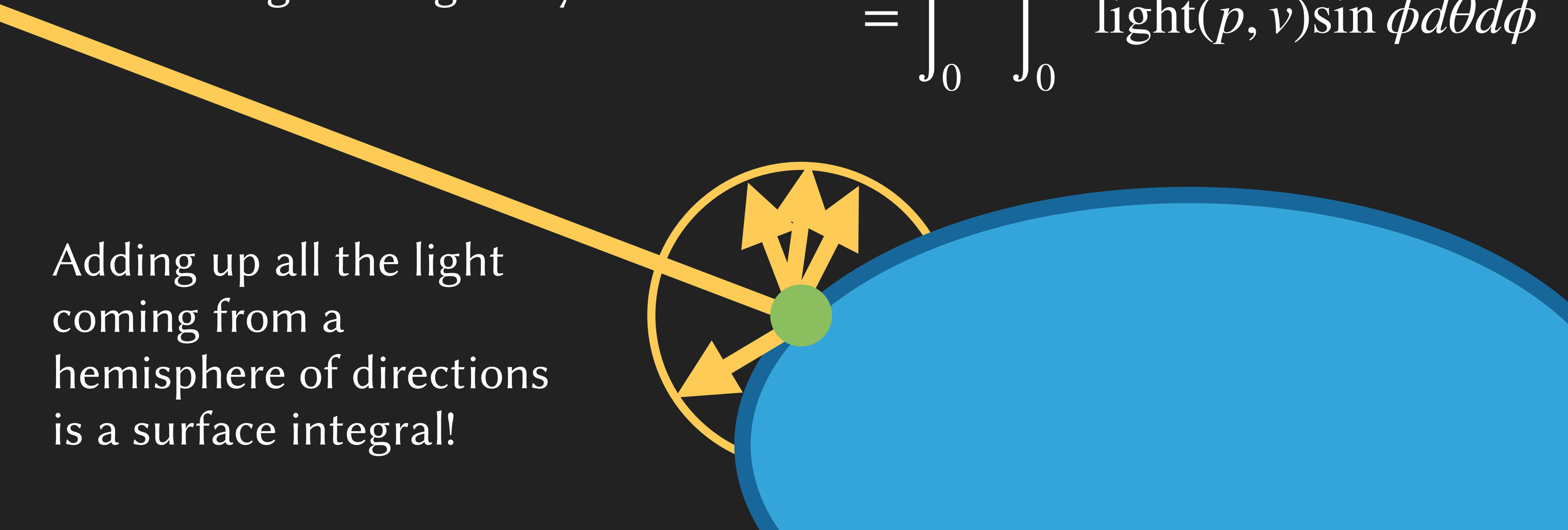
Why so noisy?!

At each bounce theres a 2D space of directions to sample: we need to follow light along every one of these

$$\iint_H \text{light}(p, v)\, dS$$

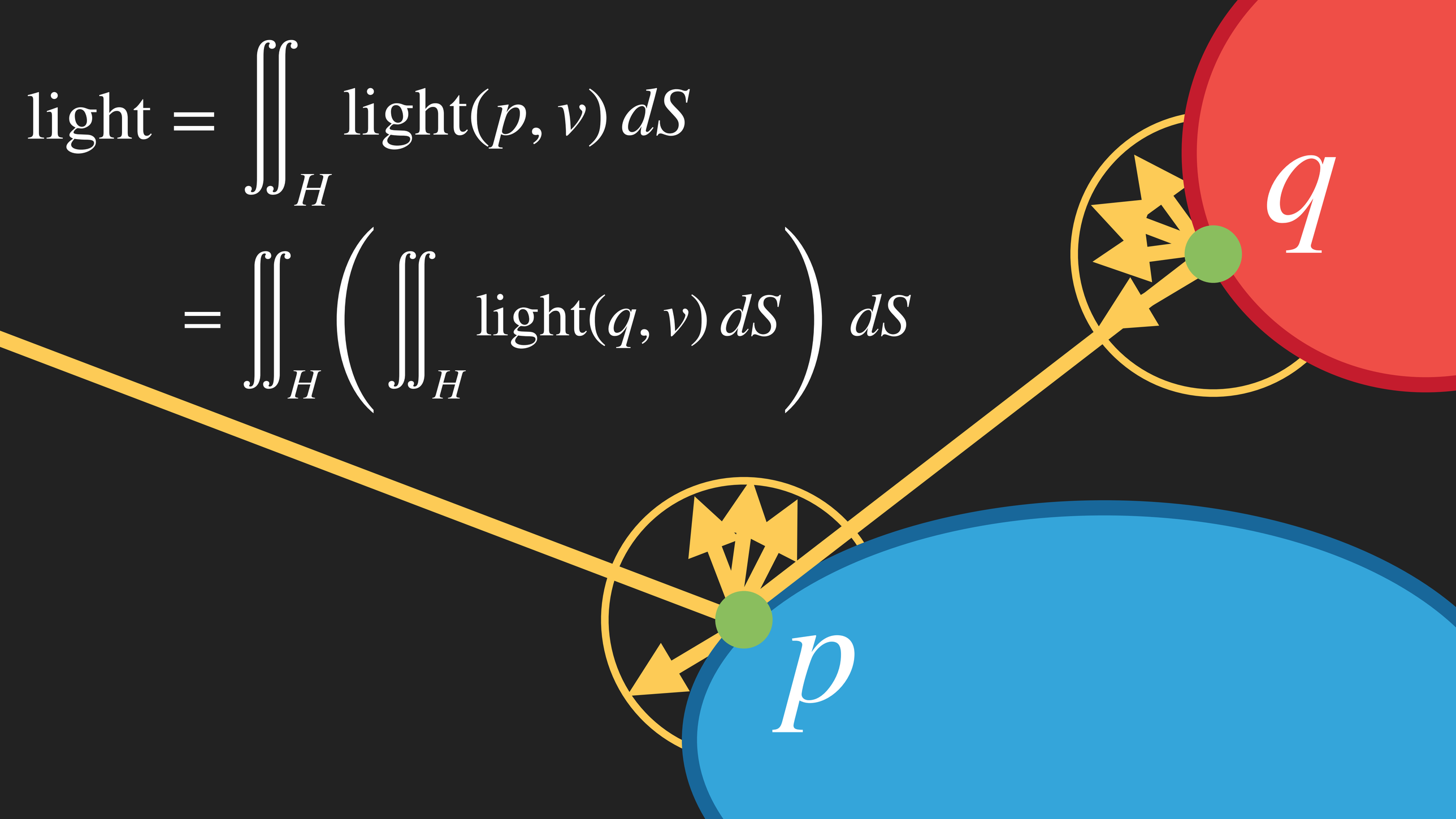
$$= \int_0^{\pi/2} \int_0^{2\pi} \text{light}(p, v) \sin\phi d\theta d\phi$$

Adding up all the light coming from a hemisphere of directions is a surface integral!



$$\text{light} = \iint_H \text{light}(p, v)\, dS$$

$$= \iint_H \left(\iint_H \text{light}(q, v)\, dS \right) dS$$



$$\text{light} = \iint_H \text{light}(p, v)\, dS$$

$$= \iint_H \left(\iint_H \text{light}(q, v)\, dS \right) dS$$

$$= \iint_H \left(\iint_H \left(\iint_H \text{light}(r, v)\, dS \right) dS \right) dS$$

$$\bullet \bullet \bullet$$

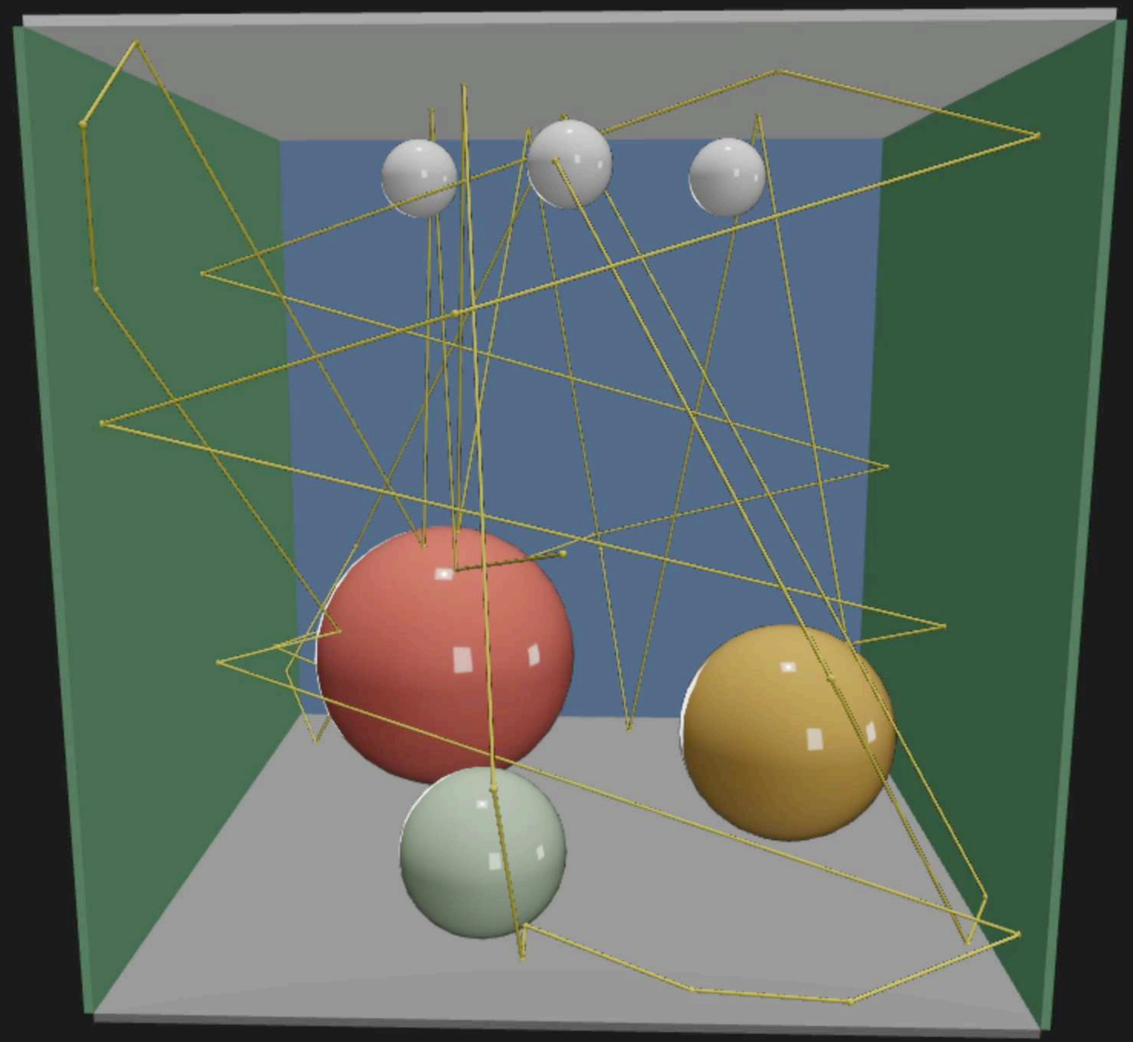
$$= \iint_H \iint_H \iint_H \cdots \iint_H \iint_H \text{light}\, dS\, dS\, dS\, \cdots\, dS\, dS$$

How do we do such a big integral?

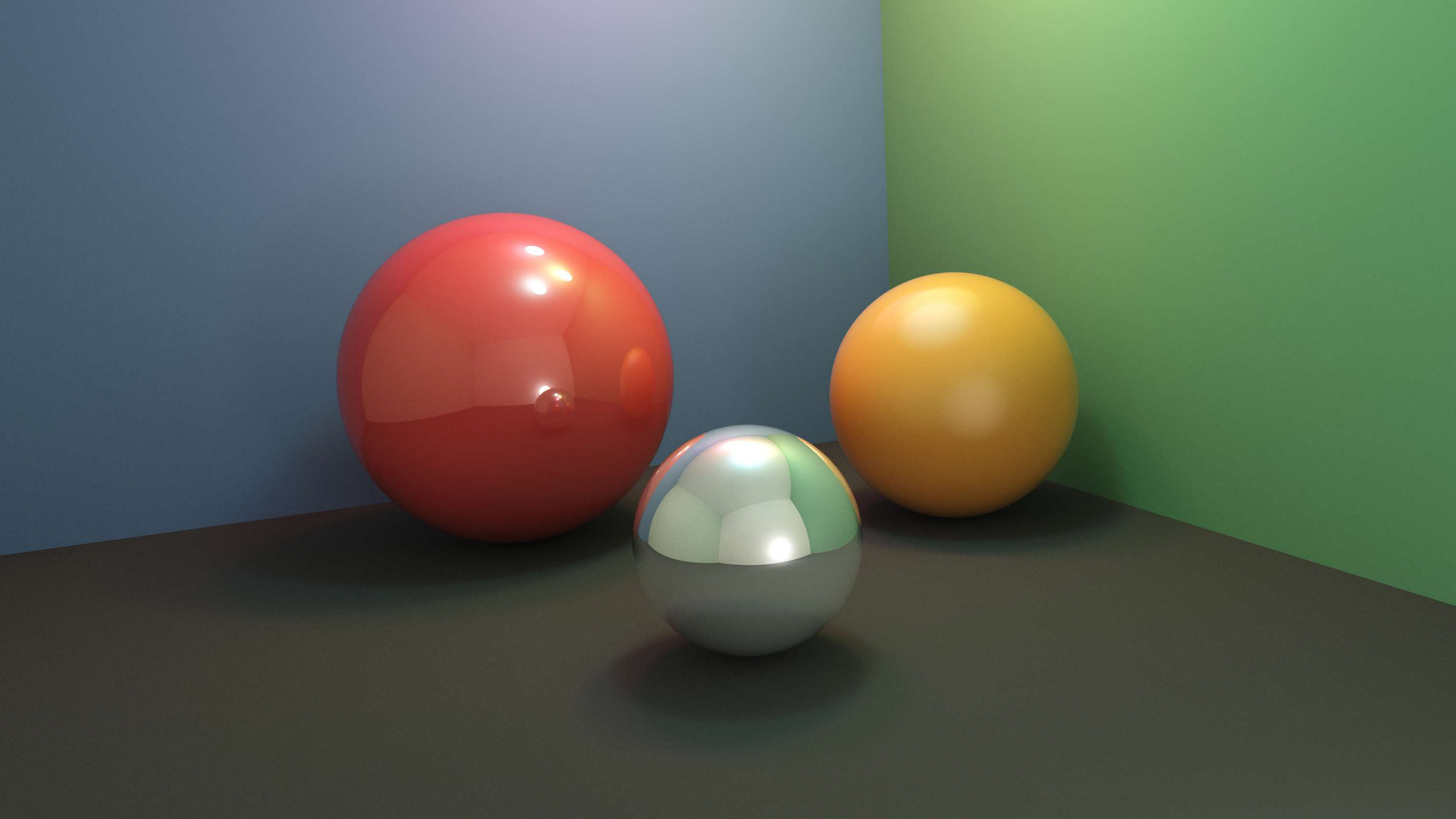
Riemann Sums!*

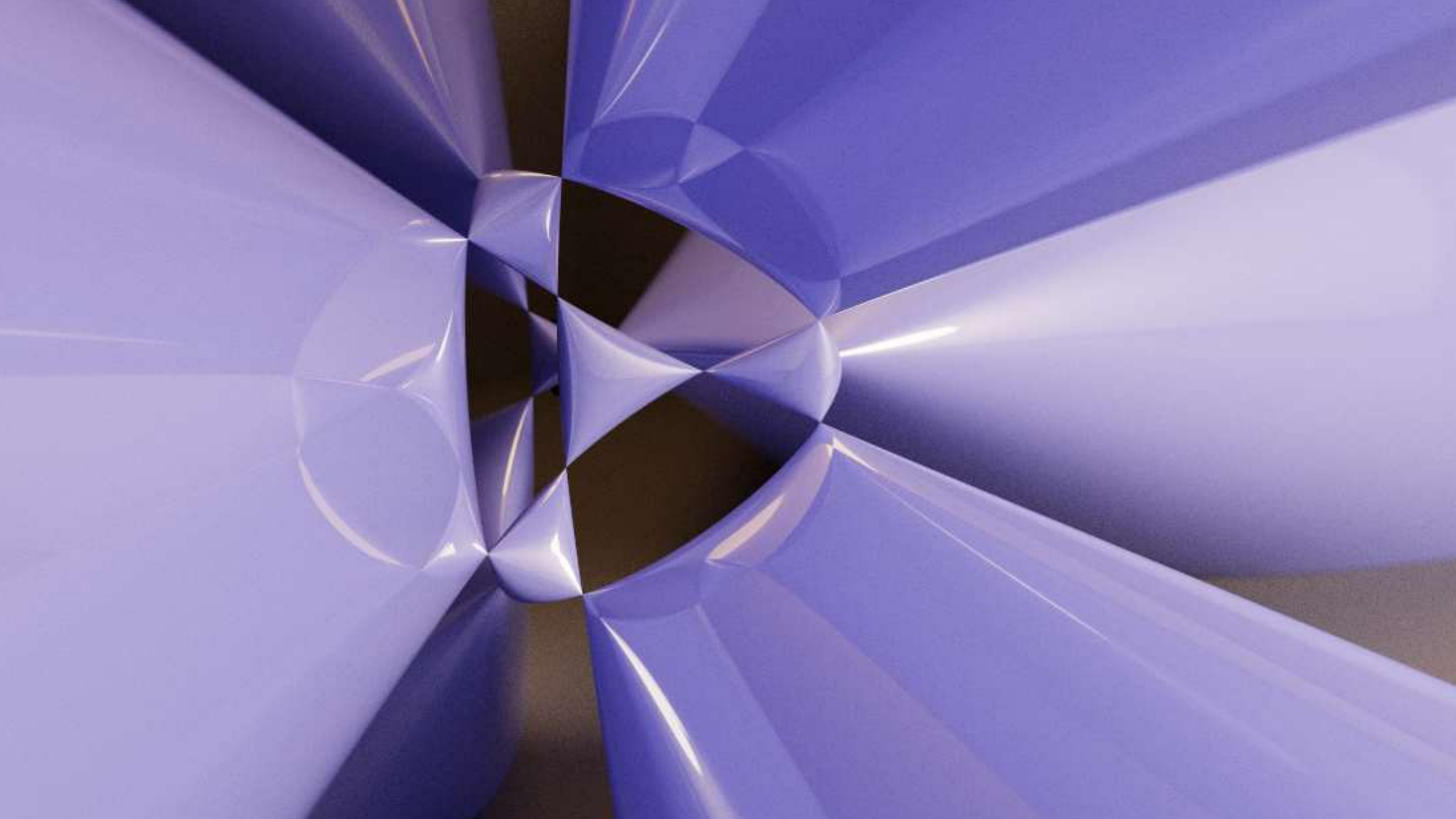
Each term is one path: add a lot of these up

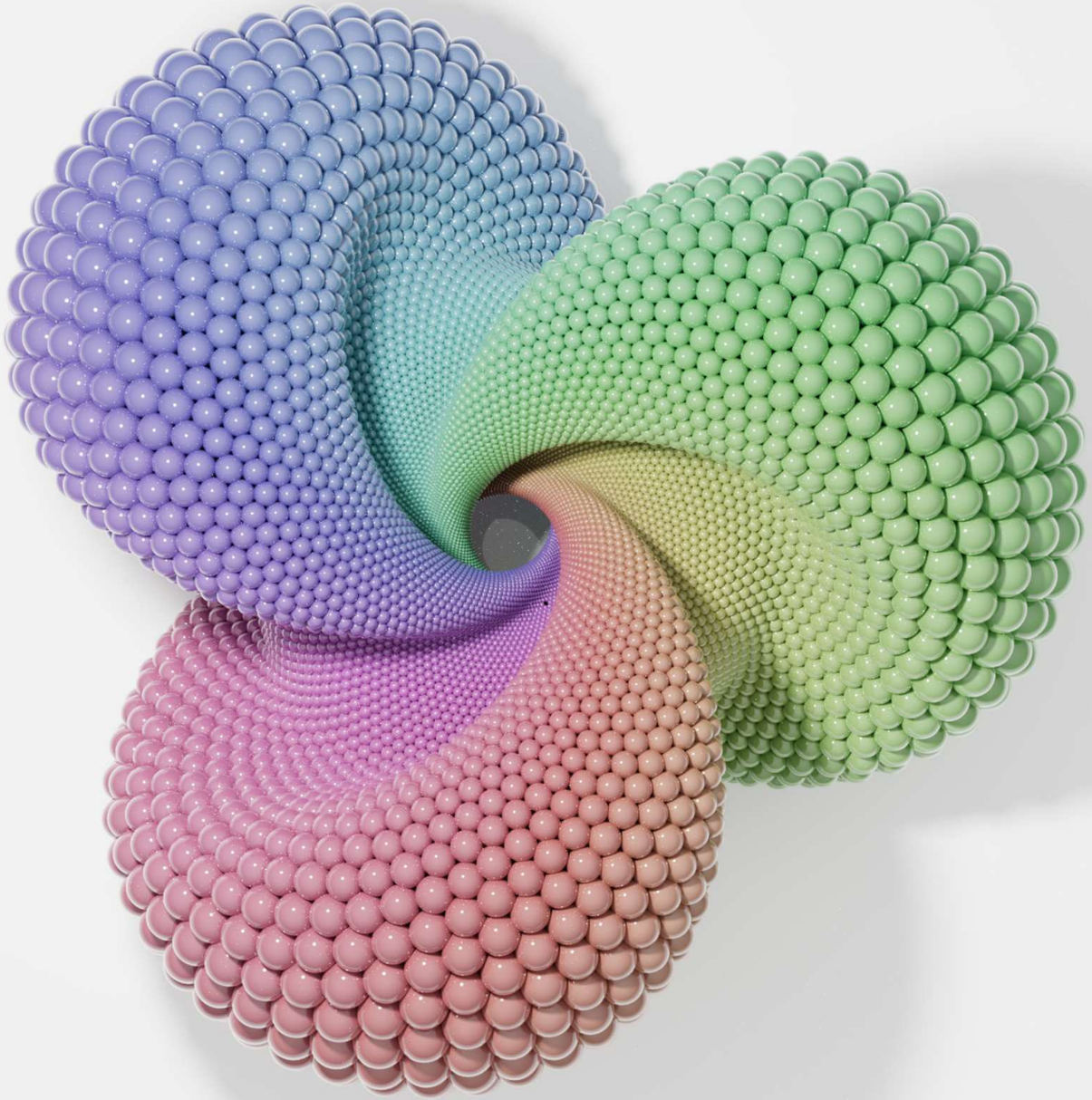
$$\text{light}(i,j) = \sum_{\text{path}} \text{light}(\text{path})$$



**Really: Monte Carlo Methods*







Idea V:

Modeling Materials

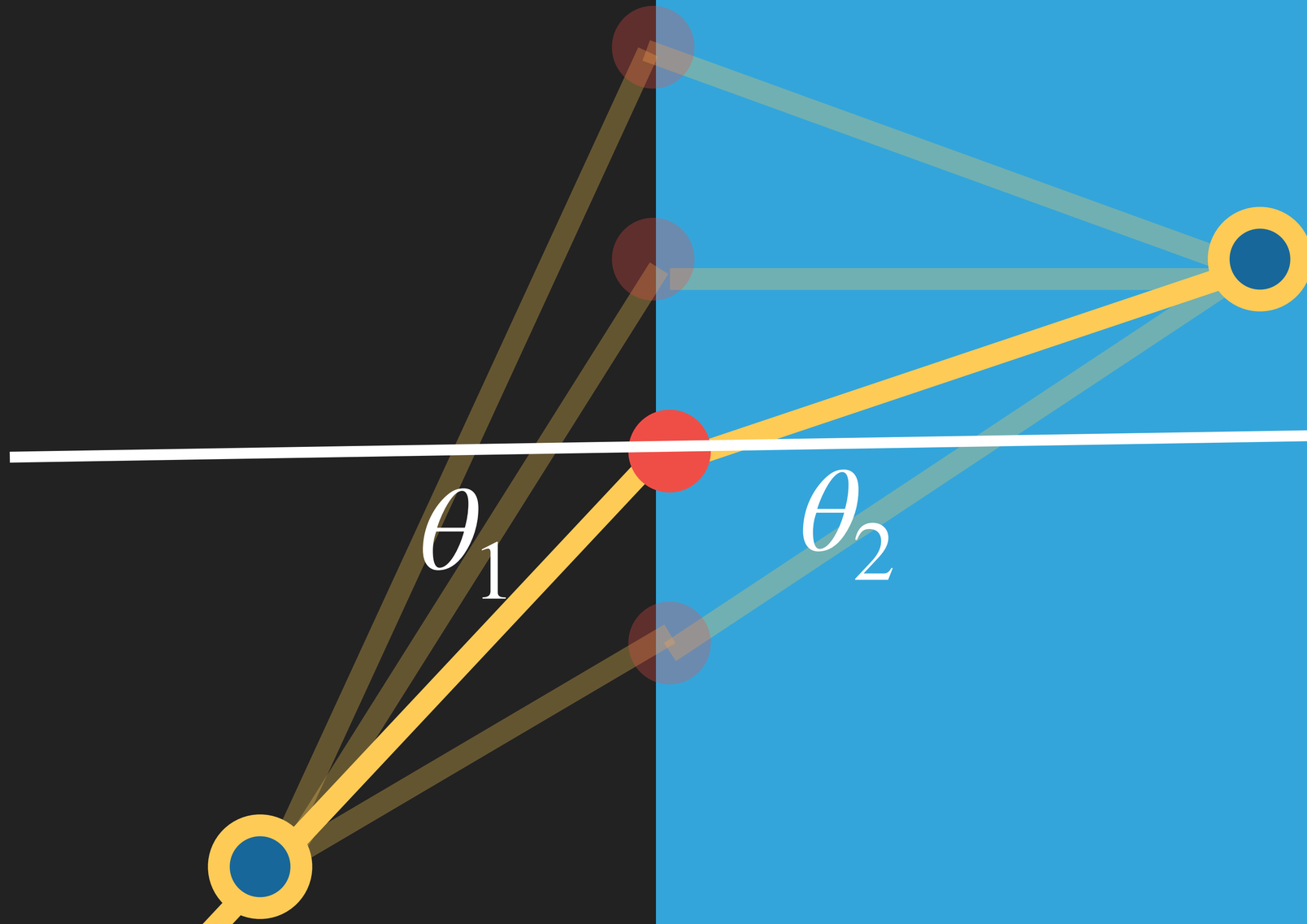
Optimization and Random Walks

How do we simulate glass like this?

Light has to pass through, and it also bends. But by how much?

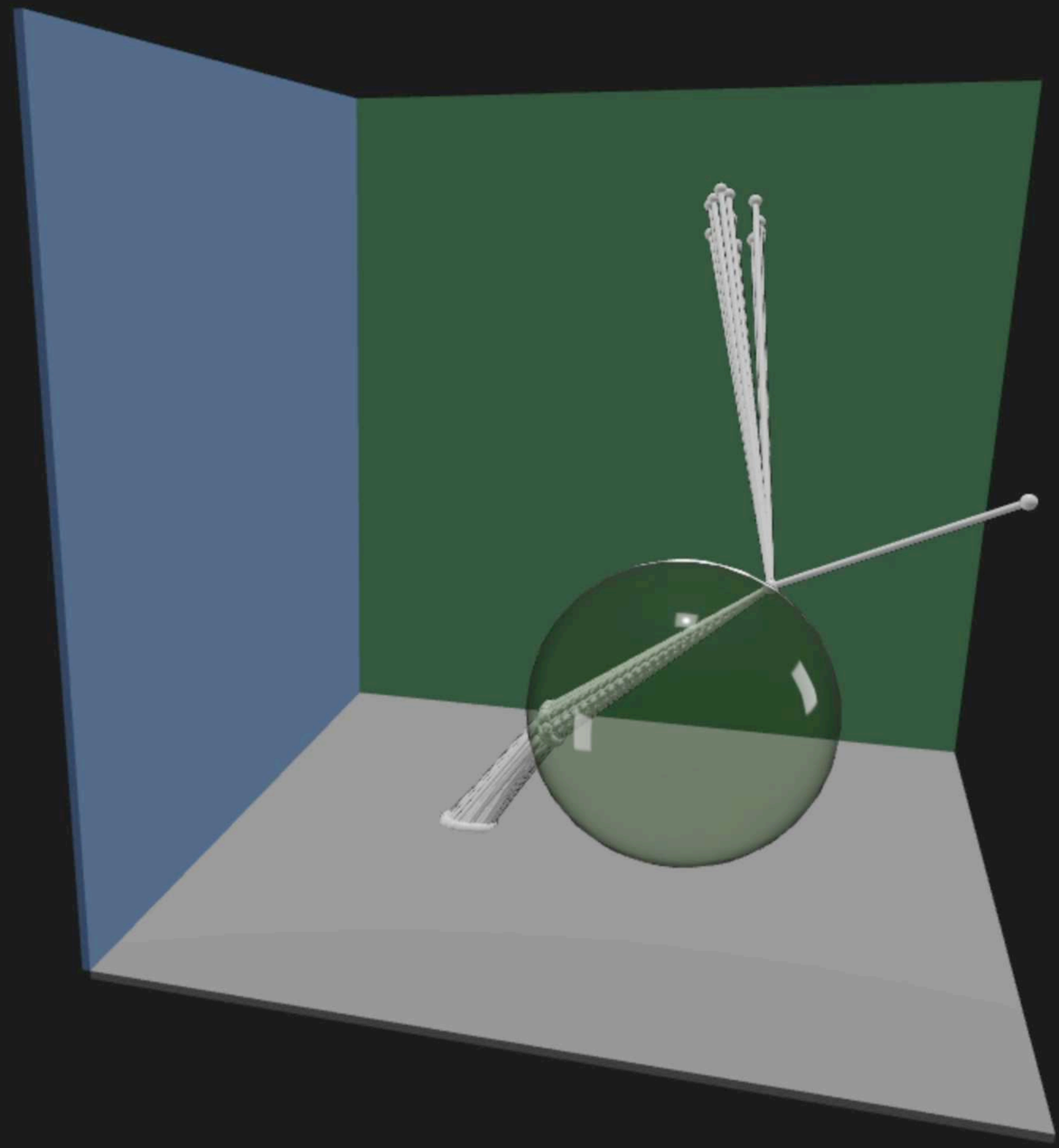
Snell's Law:

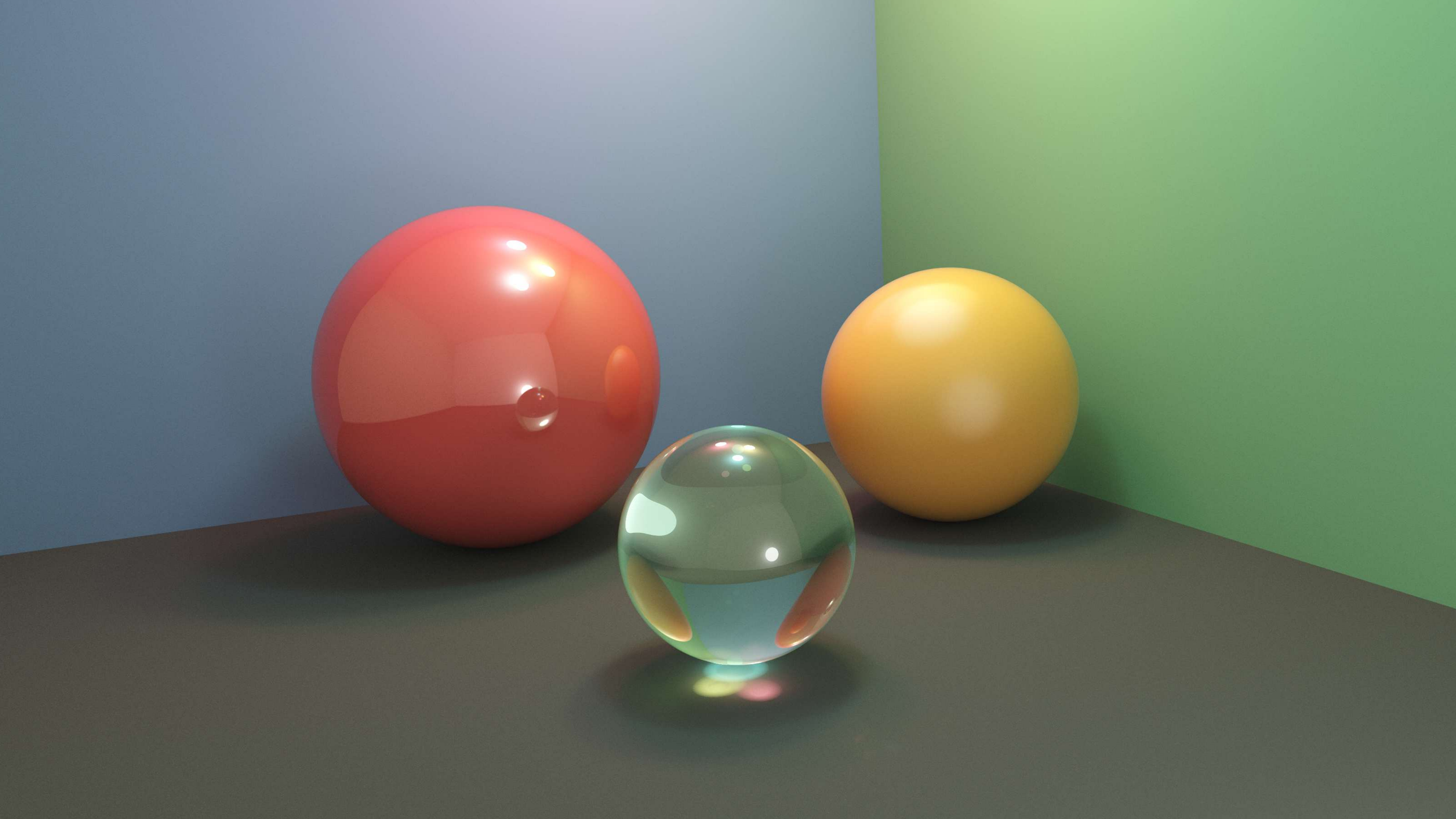
$$\frac{\sin\theta_1}{\sin\theta_2} = \frac{v_1}{v_2}$$

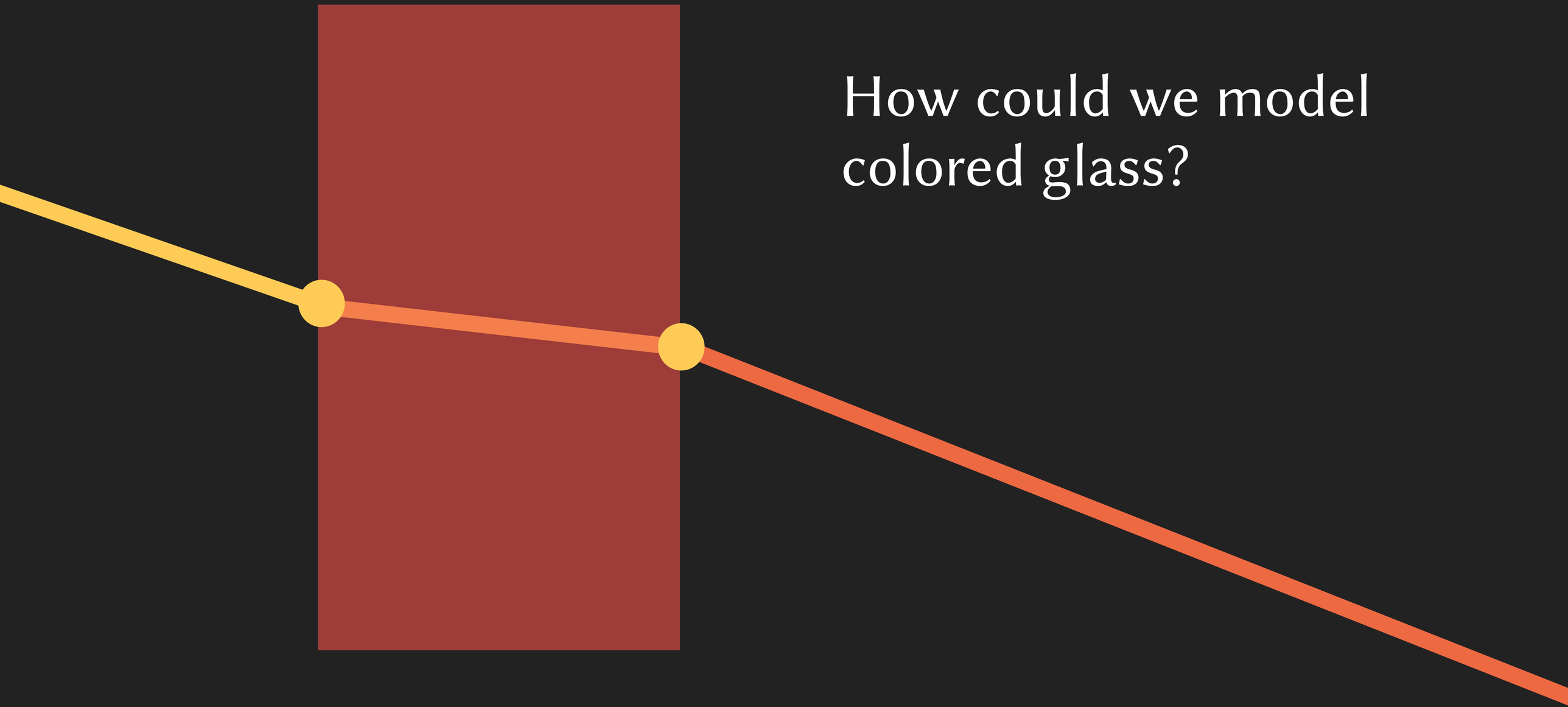


Simulatio nit works!

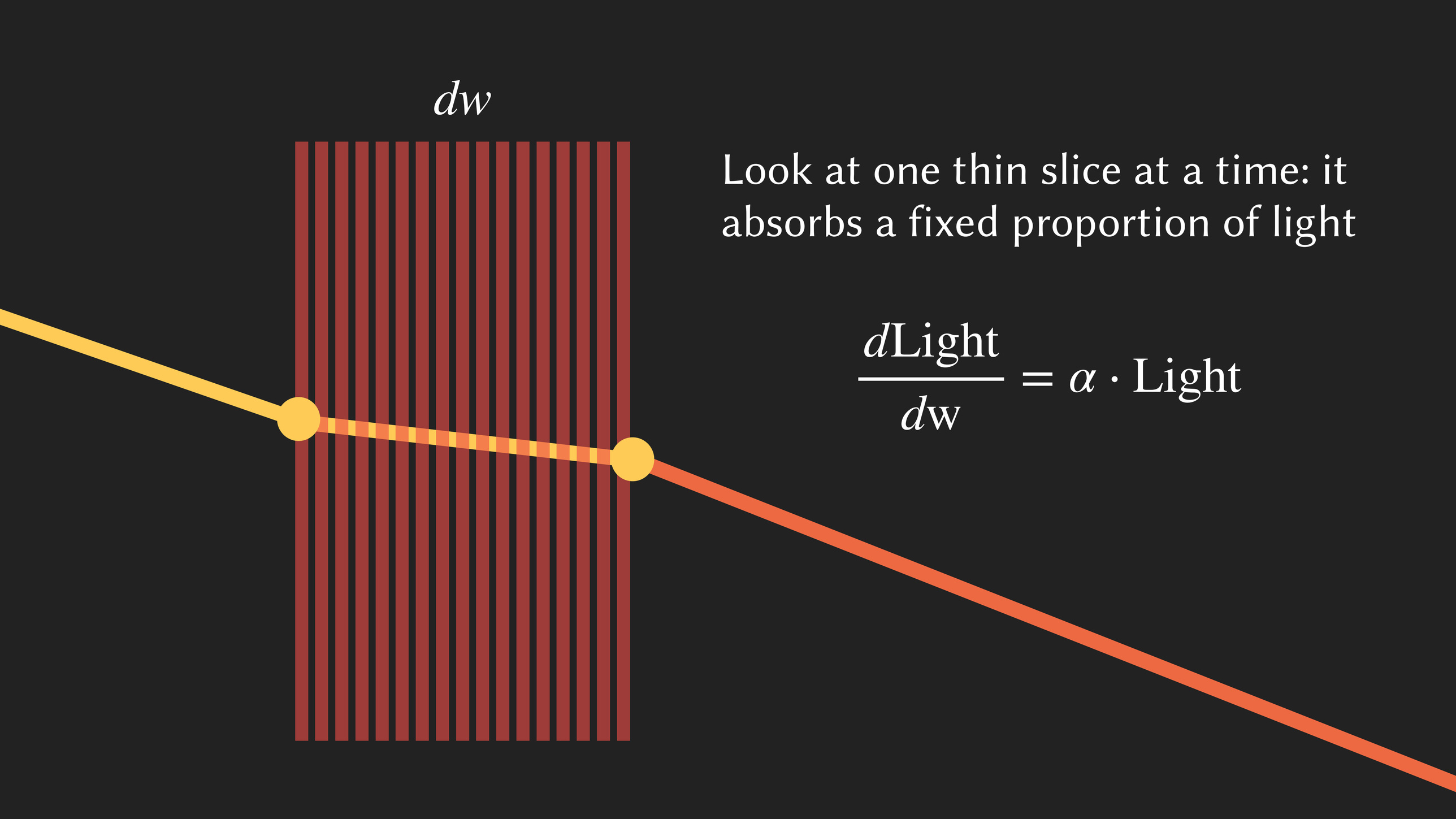
Light bends through, instead of bouncing off a clear object.





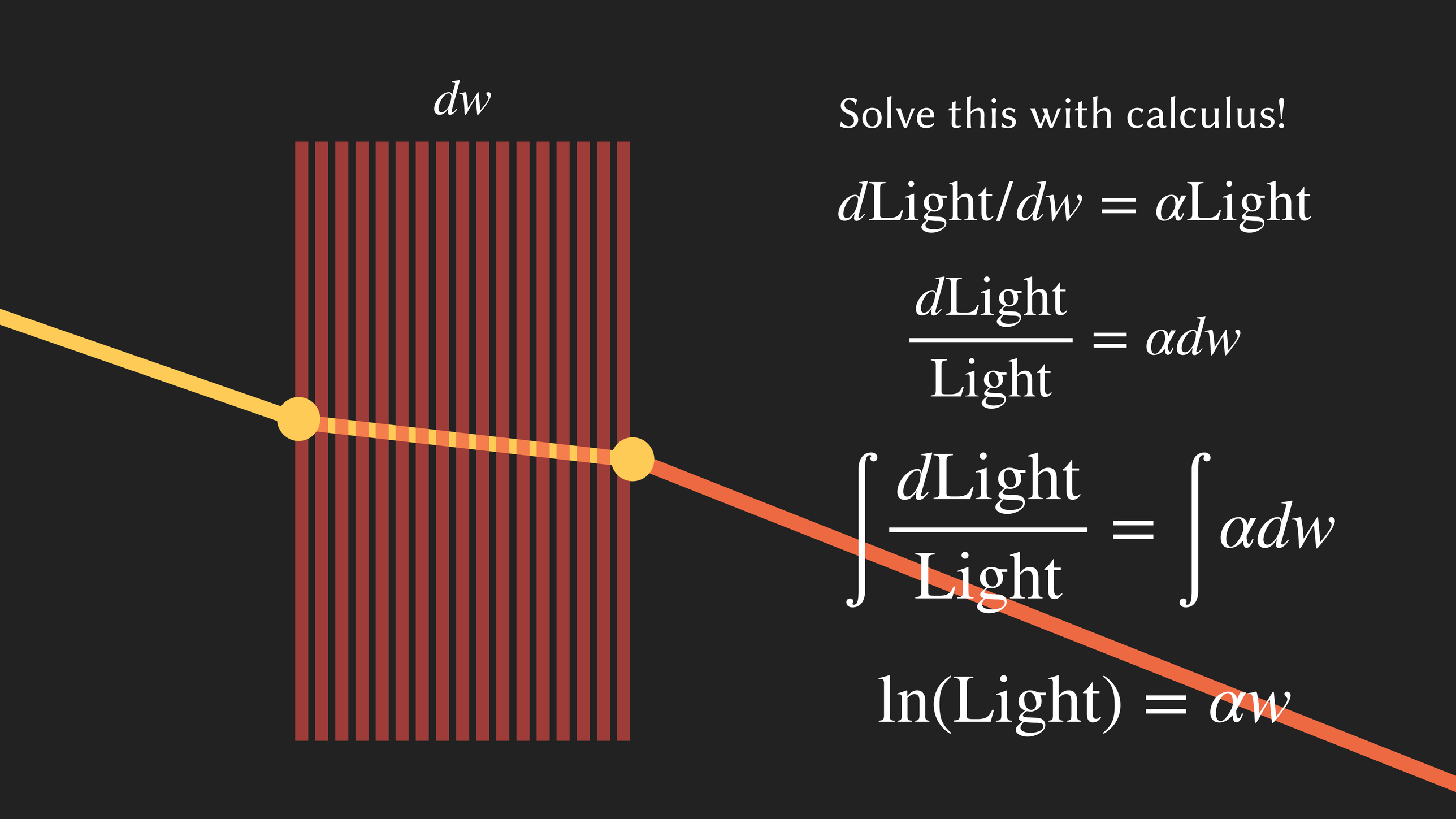


How could we model colored glass?



Look at one thin slice at a time: it absorbs a fixed proportion of light

$$\frac{d\text{Light}}{d\text{w}} = \alpha \cdot \text{Light}$$



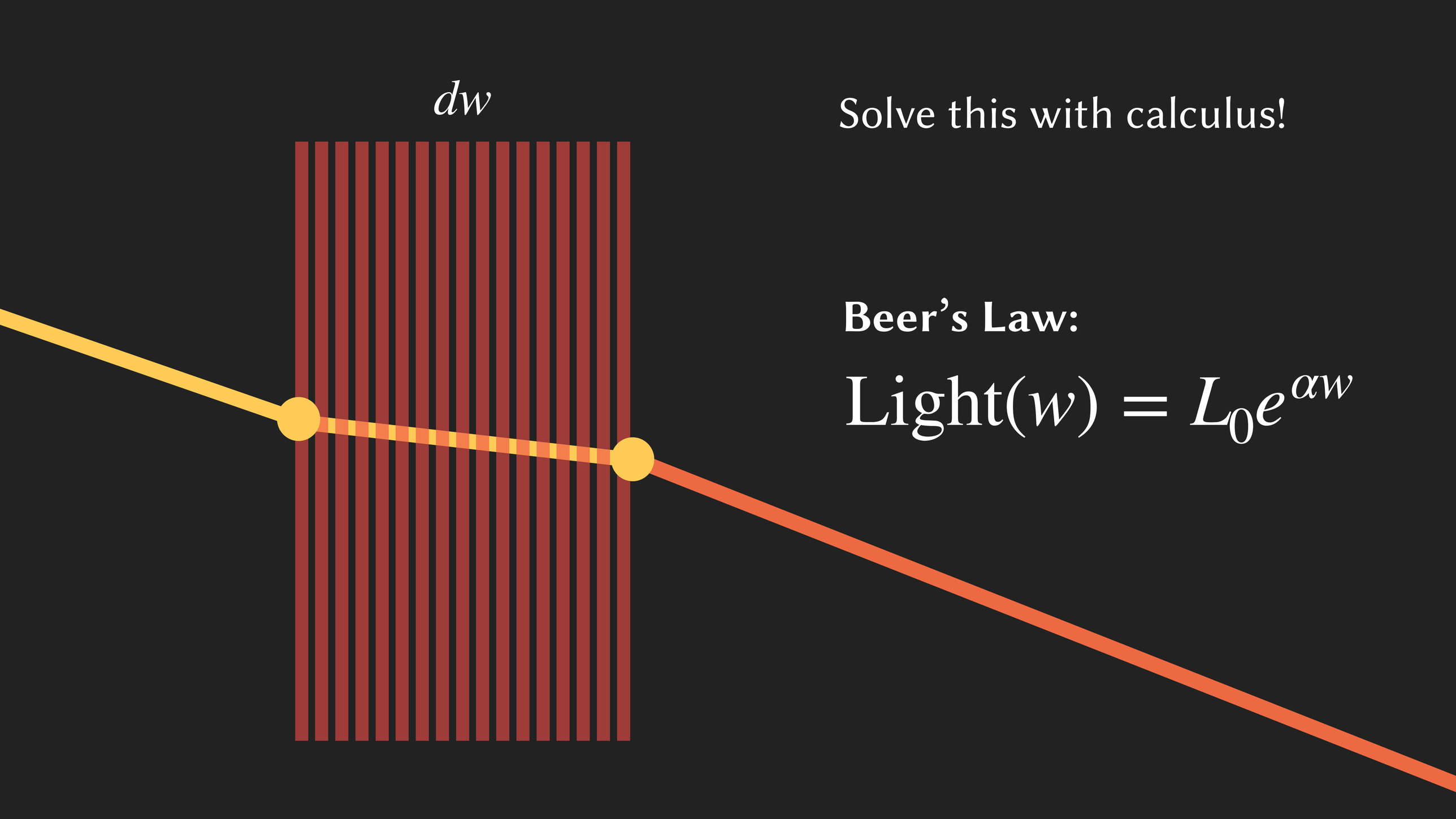
Solve this with calculus!

$$d\text{Light}/dw = \alpha\text{Light}$$

$$\frac{d\text{Light}}{\text{Light}} = \alpha dw$$

$$\int \frac{d\text{Light}}{\text{Light}} = \int \alpha dw$$

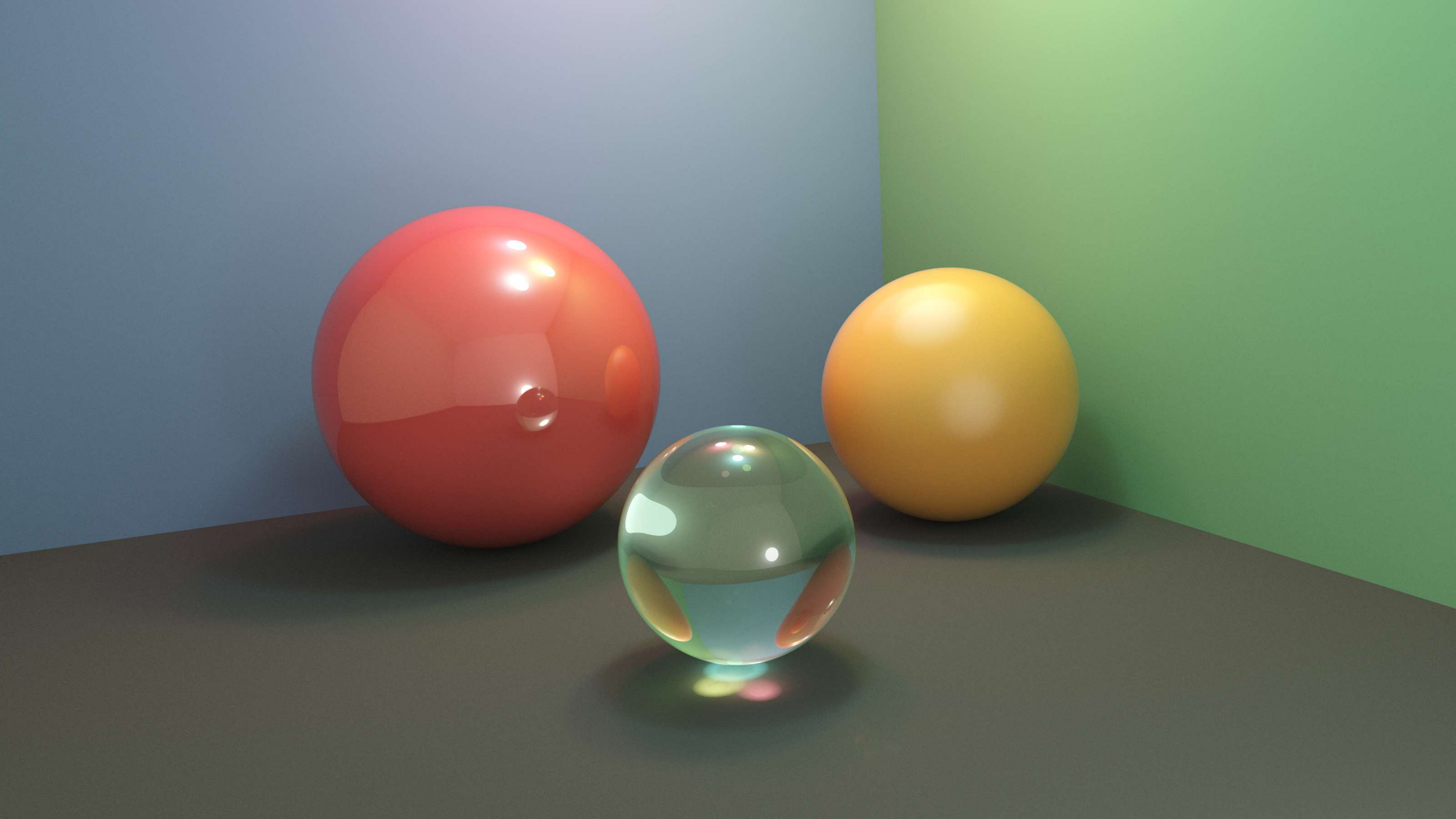
$$\ln(\text{Light}) = \alpha w$$

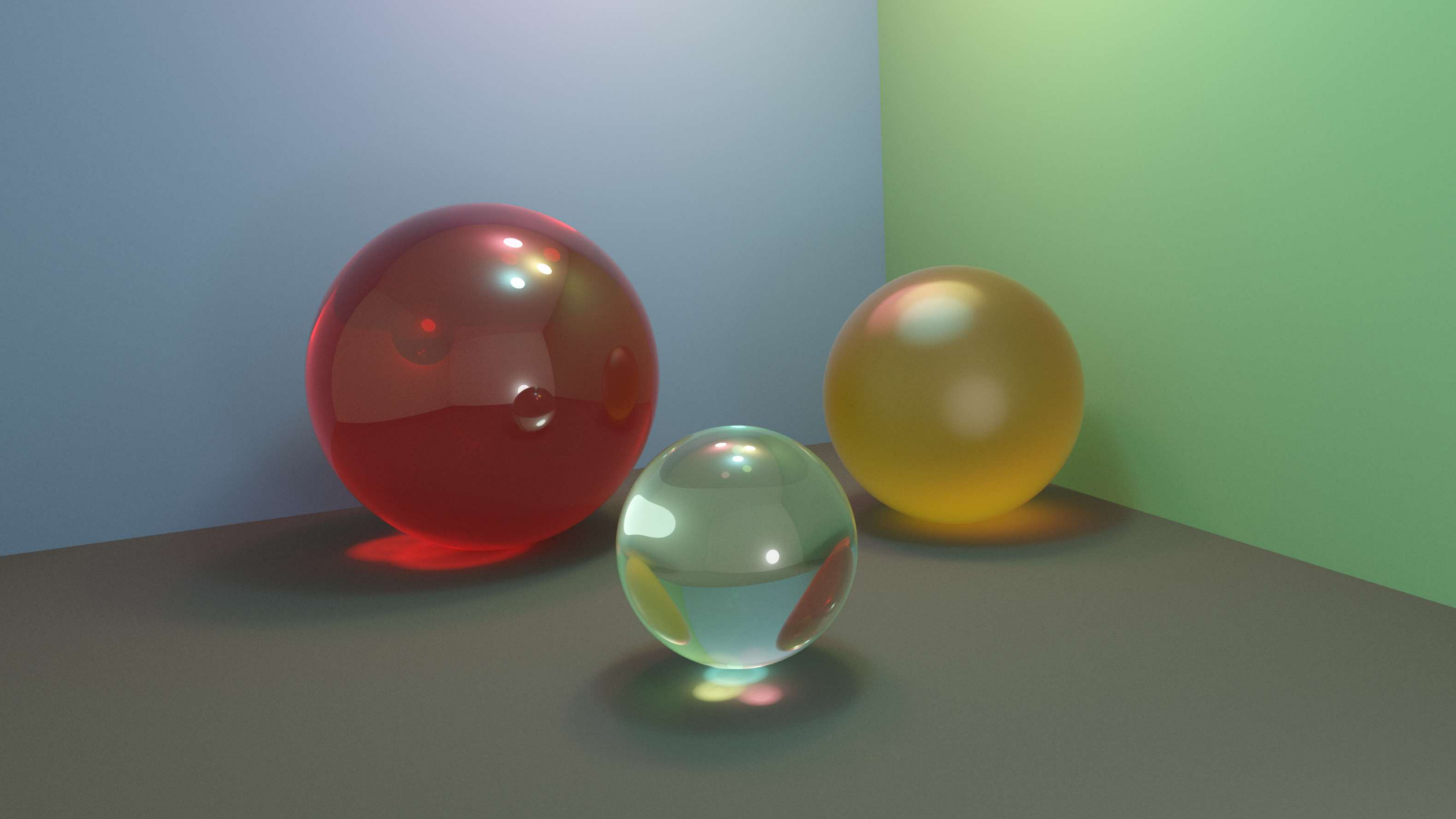


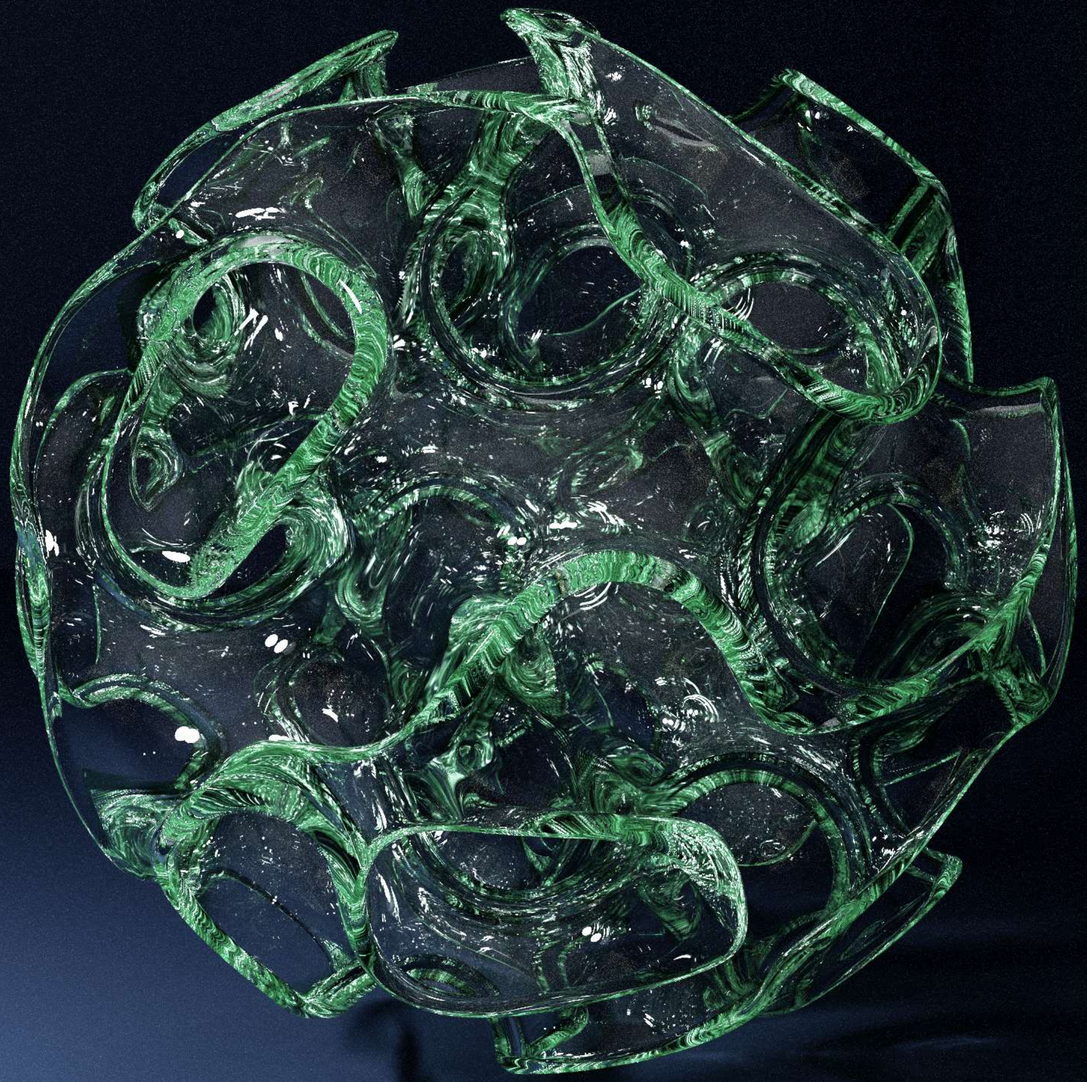
Solve this with calculus!

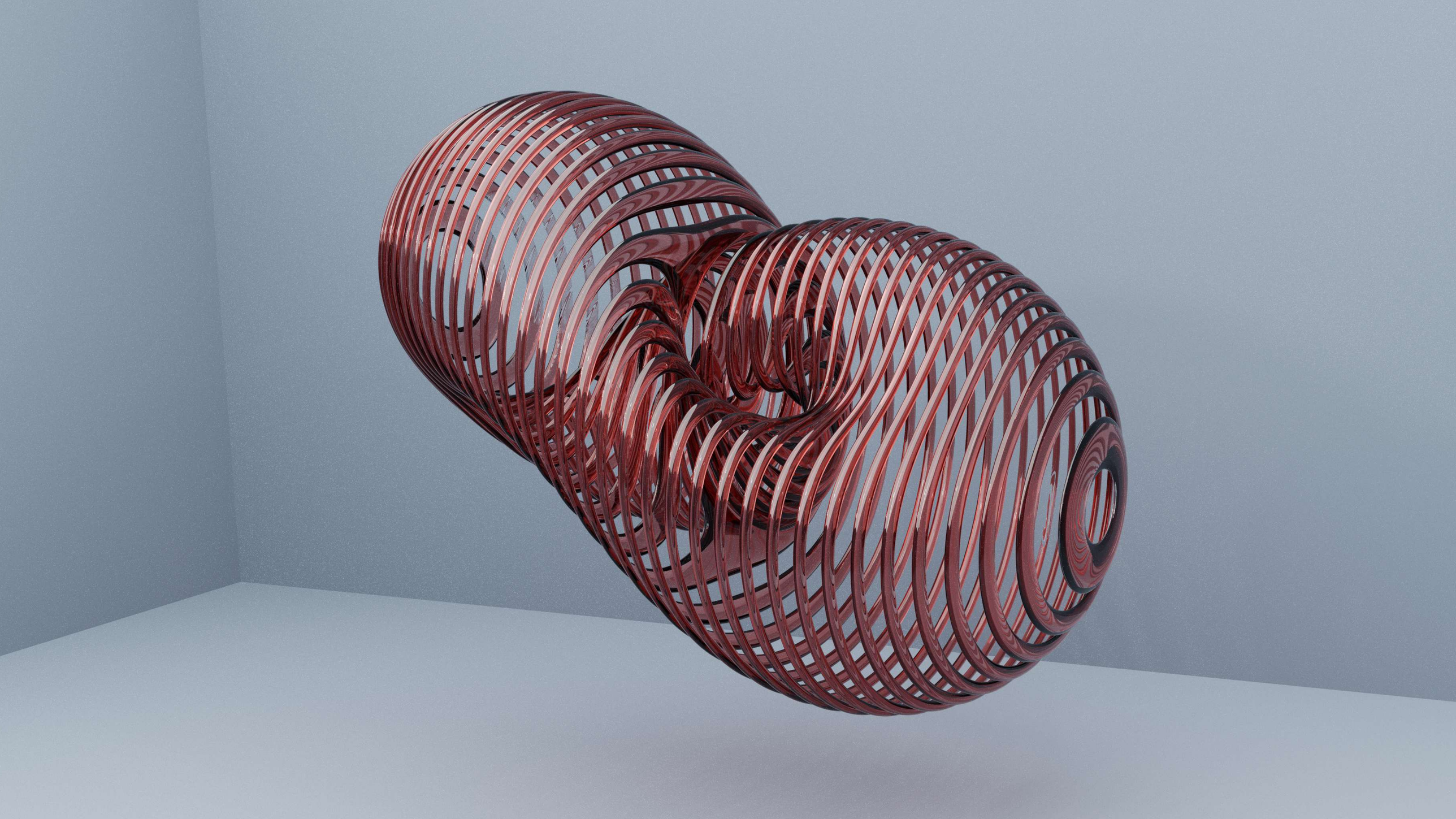
Beer's Law:

$$\text{Light}(w) = L_0 e^{\alpha w}$$









Most materials aren’t perfectly clear.

Light can scatter on the inside before bouncing back out.

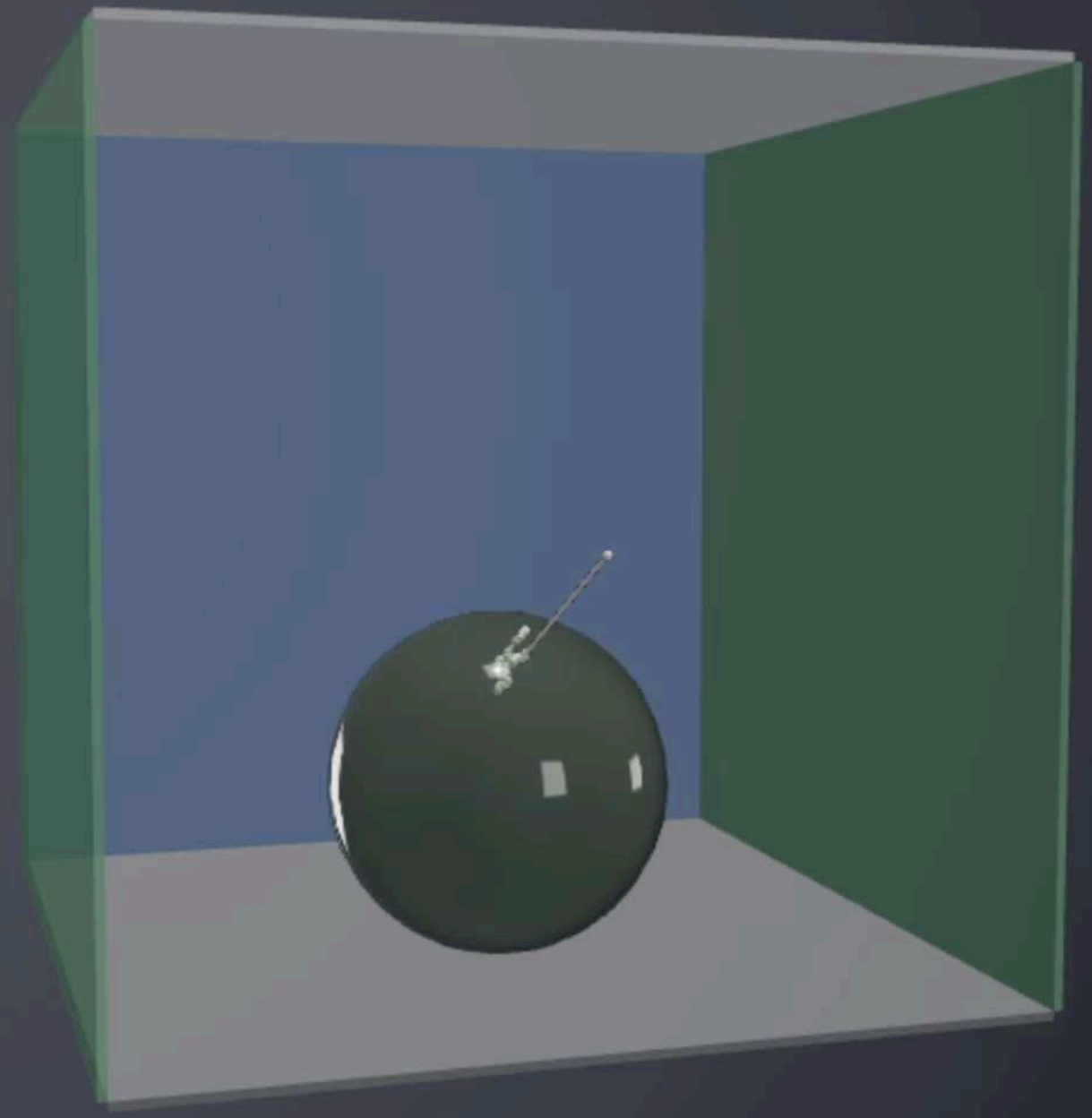
Light scatter is modeled by a *random walk*

Light travels a random distance

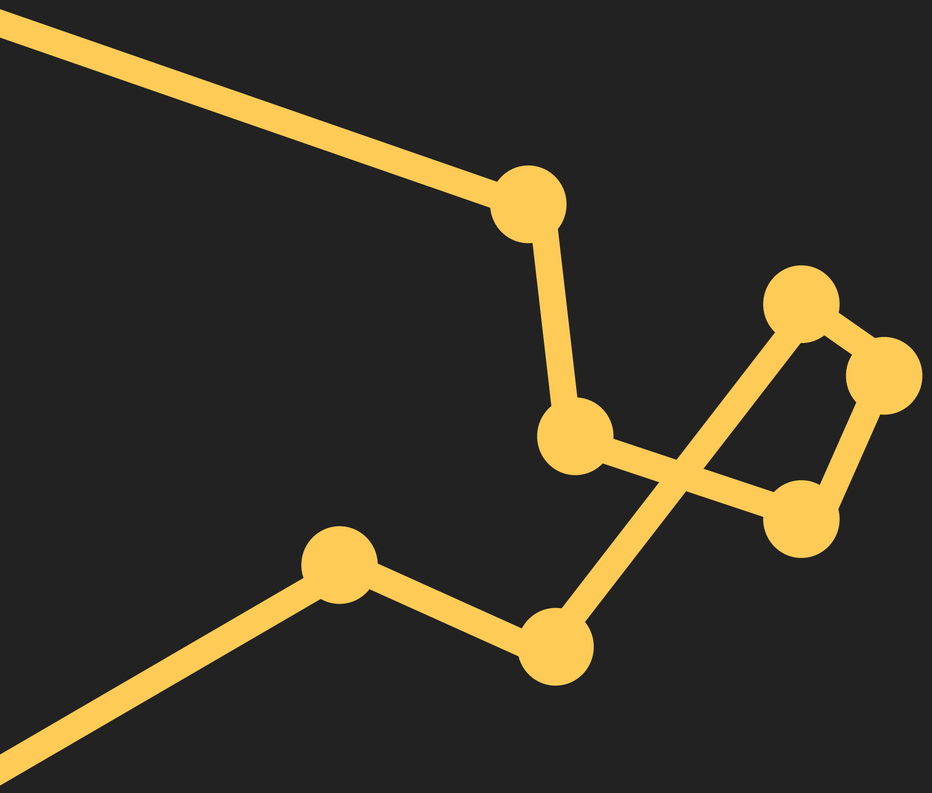
Then bounces in a random direction

Each ray follows a random path through the material

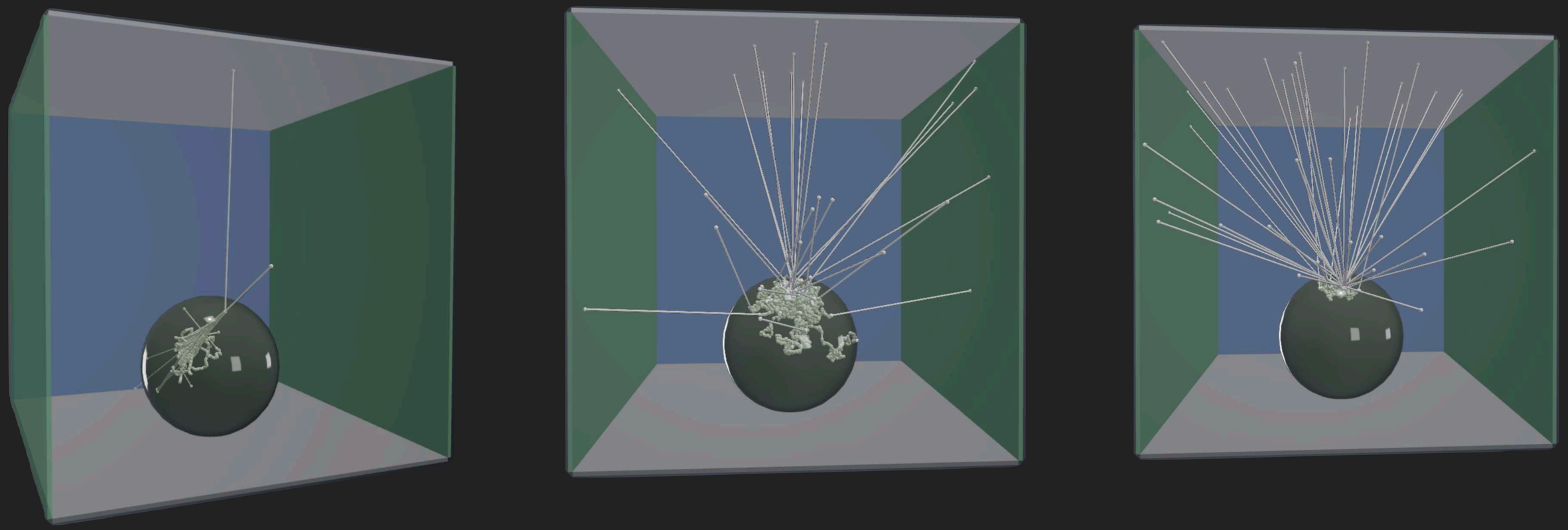
Adding up lots of these paths is a Riemann sum for an integral!



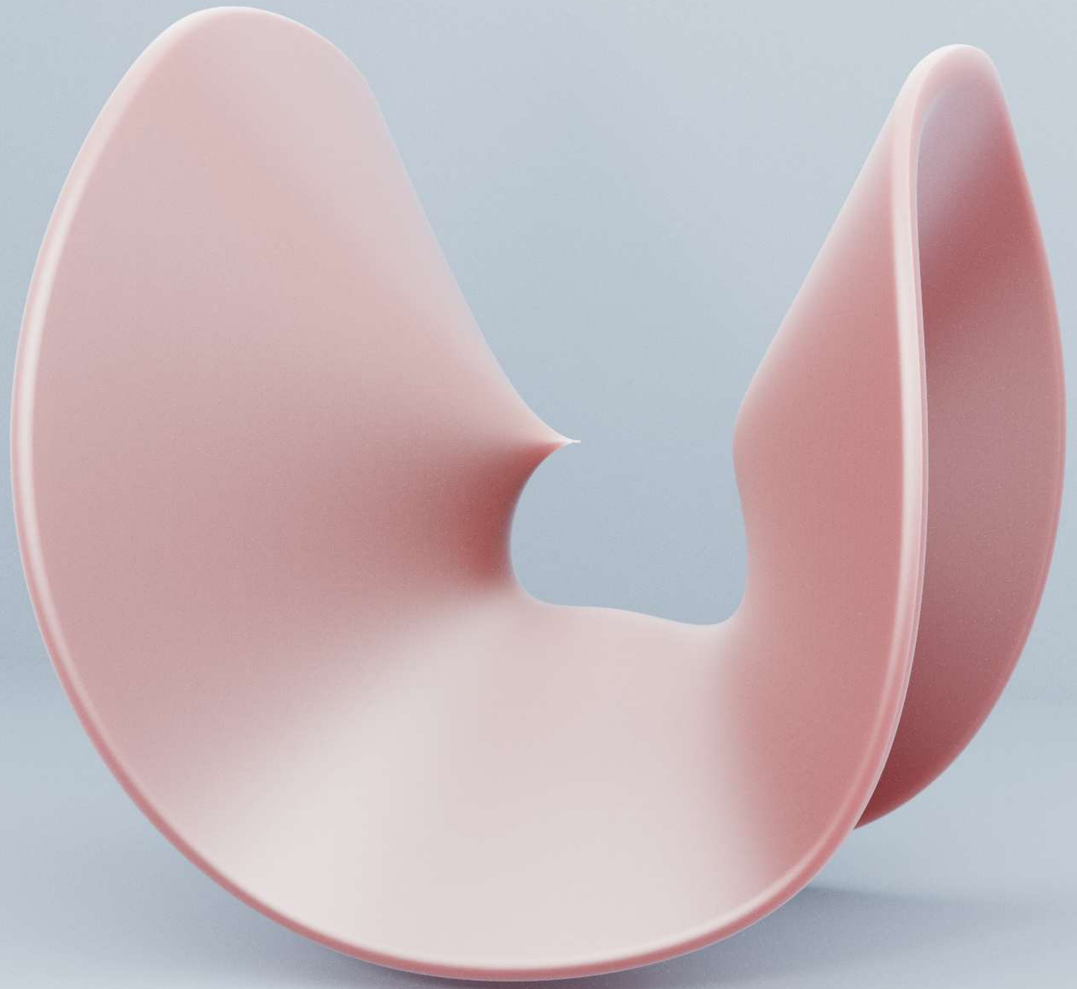
$$\text{Light} = \iiint_S \iiint_S \iiint_S \iiint_S \iiint_S \iiint_S \iiint_S$$

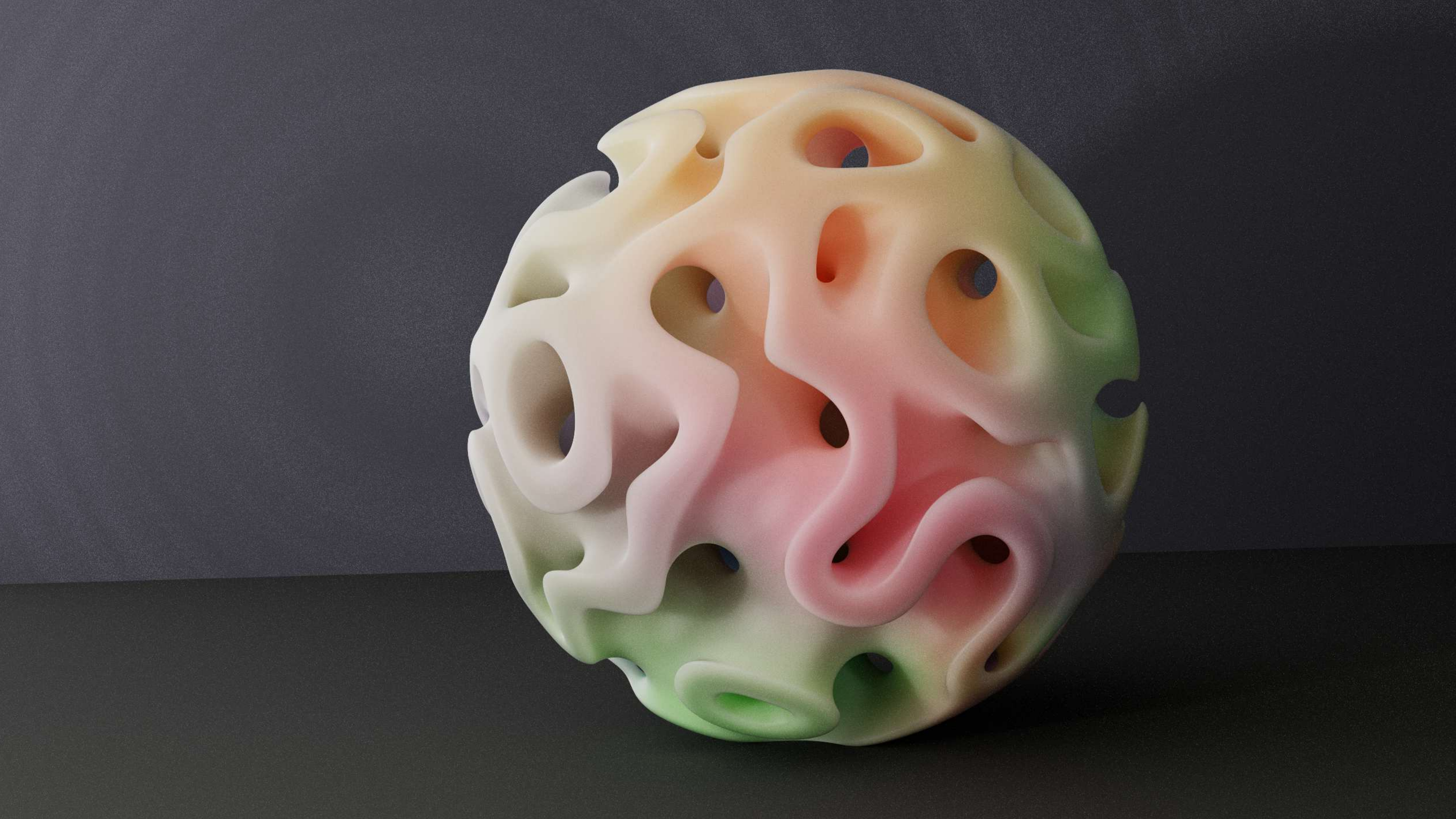


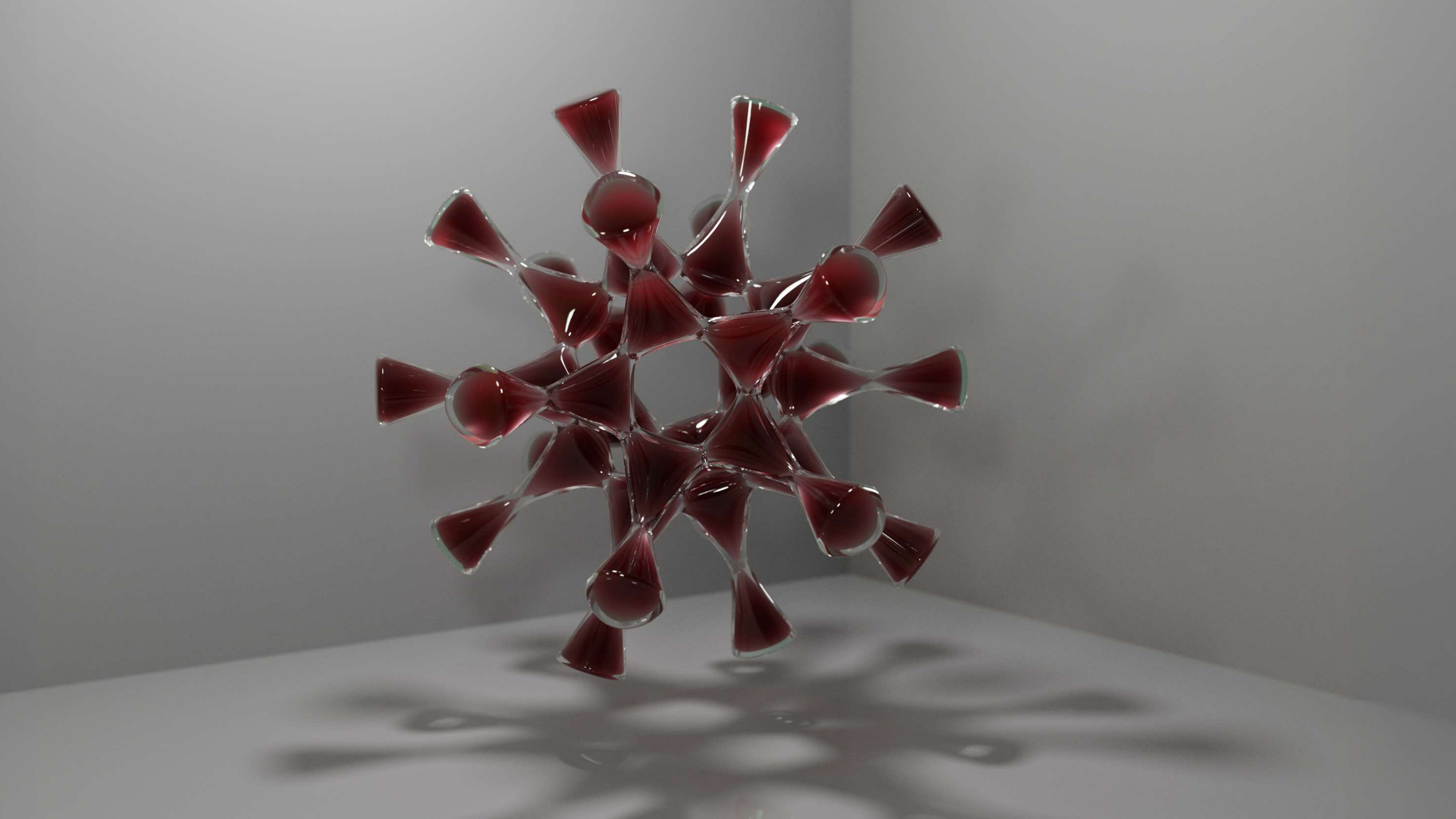
Each integral is over a probability distribution on the sphere, for direction of scattering

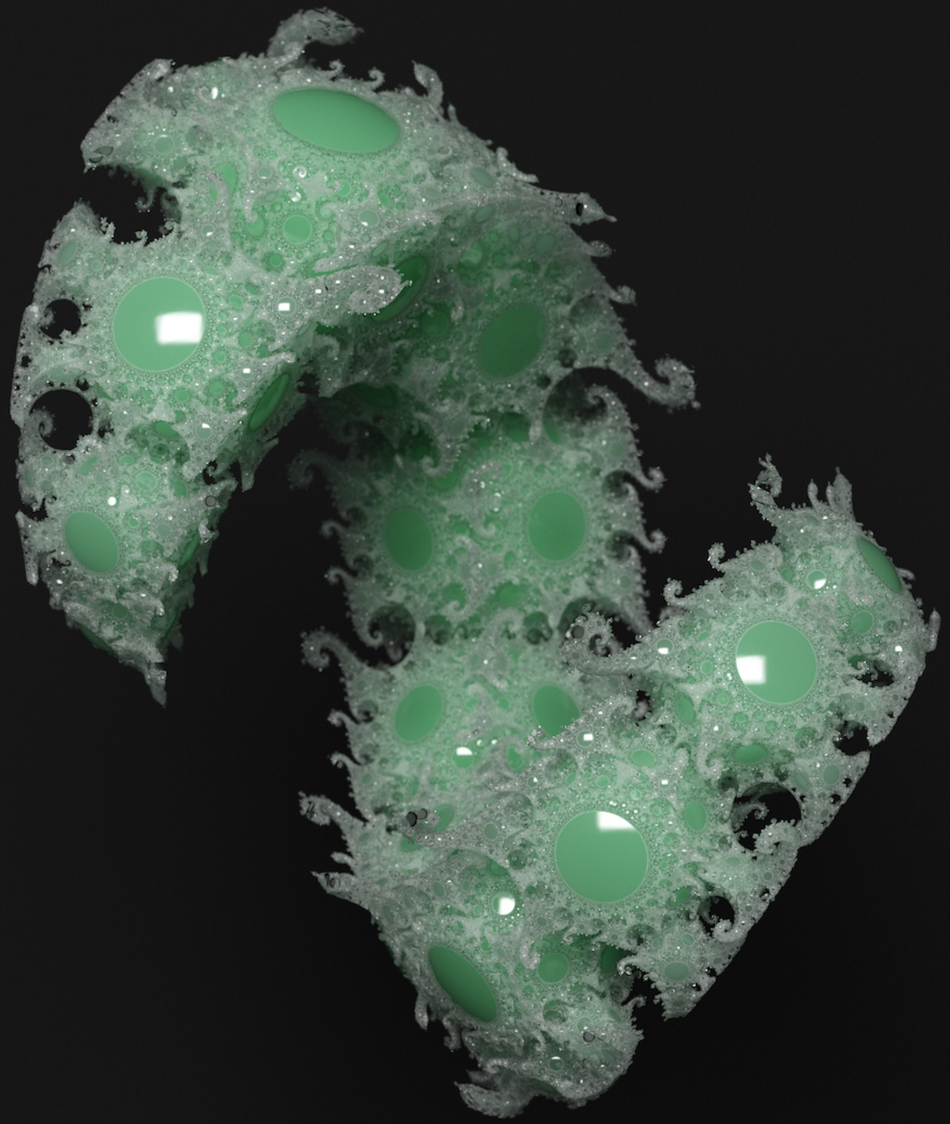


Differences in internal scattering leads to very different material properties







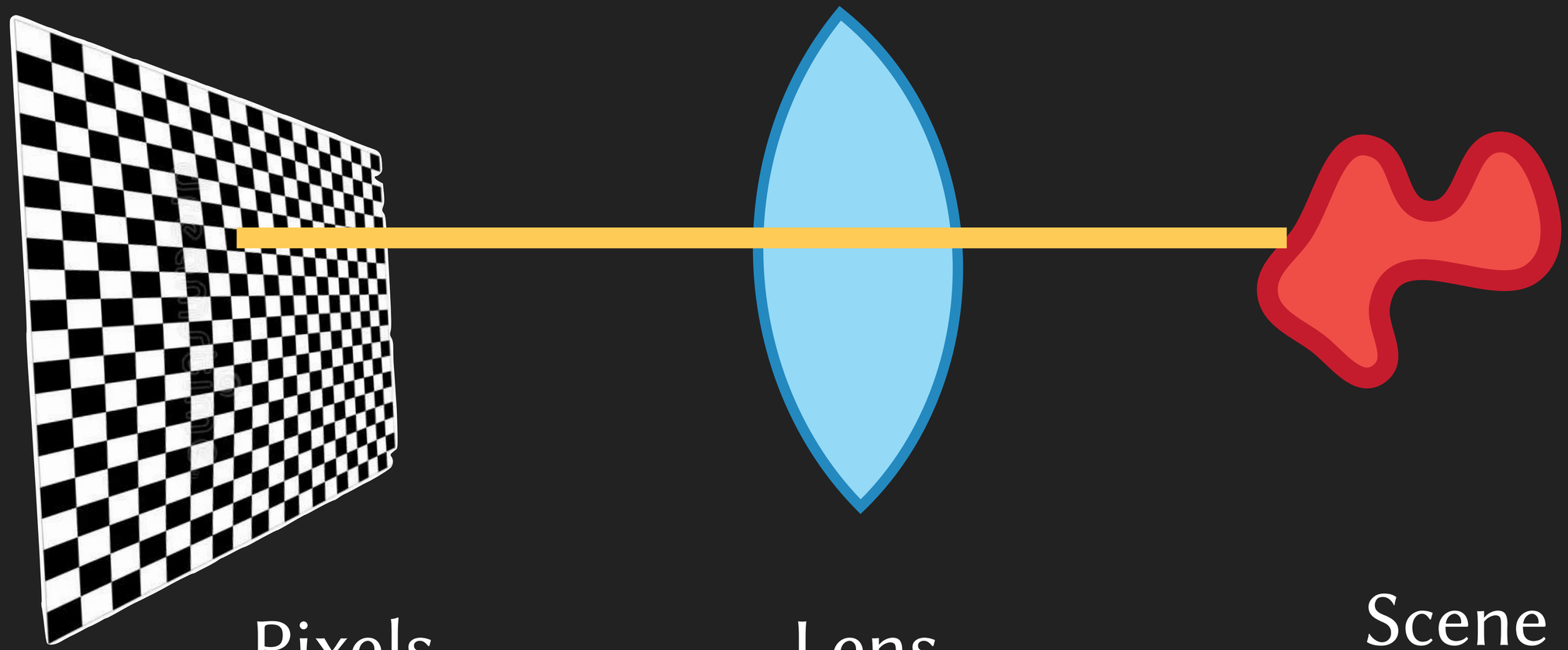


Idea VI:

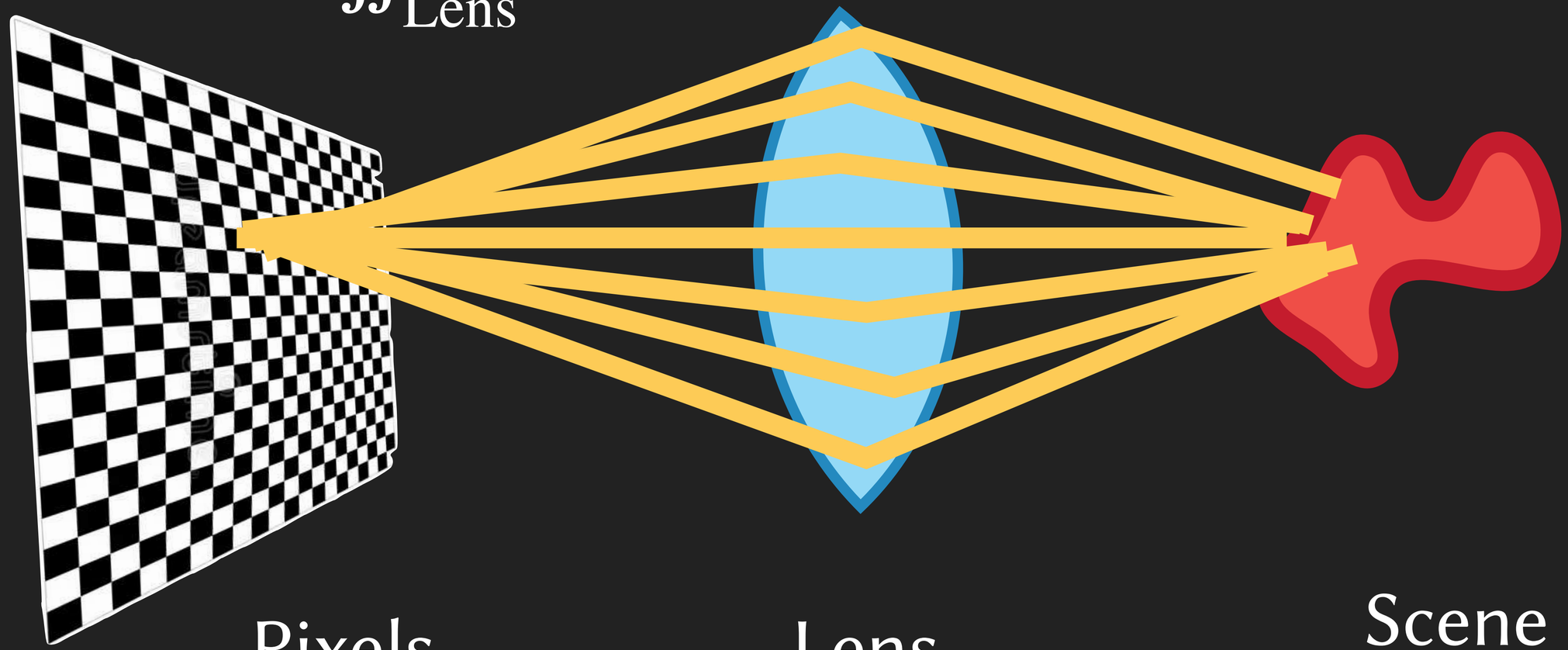
Lenses & Optics

Surface integrals and depth of field


Pixels
Scene


Pixels
Lens
Scene

$$\text{Pixel} = \iint_{\text{Lens}} \text{Light}\, dS$$



Pixels

Lens

Scene