thrive
SIGGRAPH2019
LOS ANGELES • 28 JULY – 1 AUGUST

Introduction to Real-Time Ray Tracing

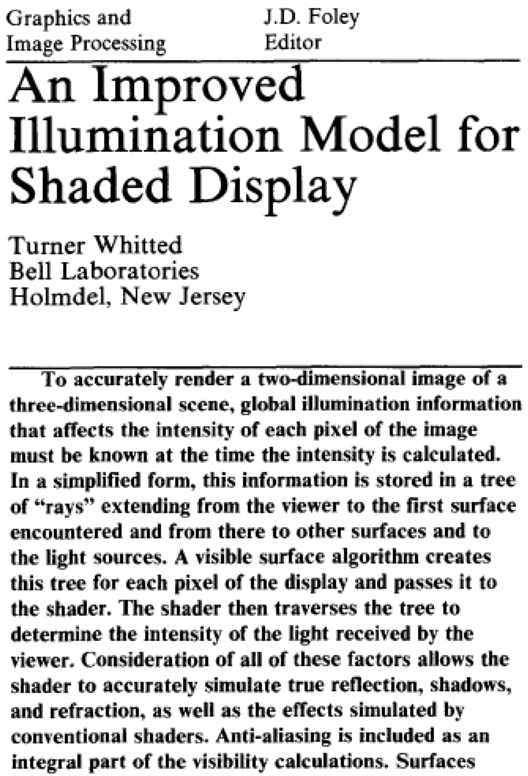
Peter Shirley
Chris Wyman
Morgan McGuire

NVIDIA

RAY TRACING: INTRODUCED BY TURNER WHITTED IN 1979

- Mirrors, glass, shadows
- Multi-sampled AA
- Hand-built BVH

- 1980s Glossy reflection, diffuse inter-reflection
- 2000s Movie rendering
- 2010s De-noising
- now interactive

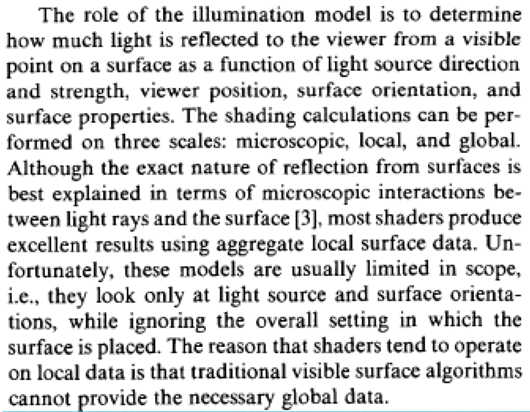


Graphics and Image Processing — J.D. Foley, Editor

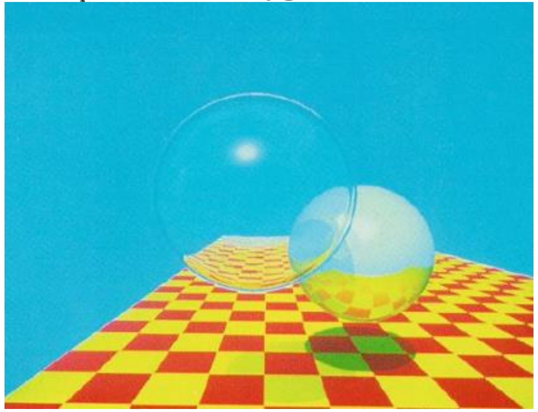
An Improved Illumination Model for Shaded Display

Turner Whitted
Bell Laboratories
Holmdel, New Jersey

To accurately render a two-dimensional image of a three-dimensional scene, global illumination information that affects the intensity of each pixel of the image must be known at the time the intensity is calculated. In a simplified form, this information is stored in a tree of "rays" extending from the viewer to the first surface encountered and from there to other surfaces and to the light sources. A visible surface algorithm creates this tree for each pixel of the display and passes it to the shader. The shader then traverses the tree to determine the intensity of the light received by the viewer. Consideration of all of these factors allows the shader to accurately simulate true reflection, shadows, and refraction, as well as the effects simulated by conventional shaders. Anti-aliasing is included as an integral part of the visibility calculations. Surfaces

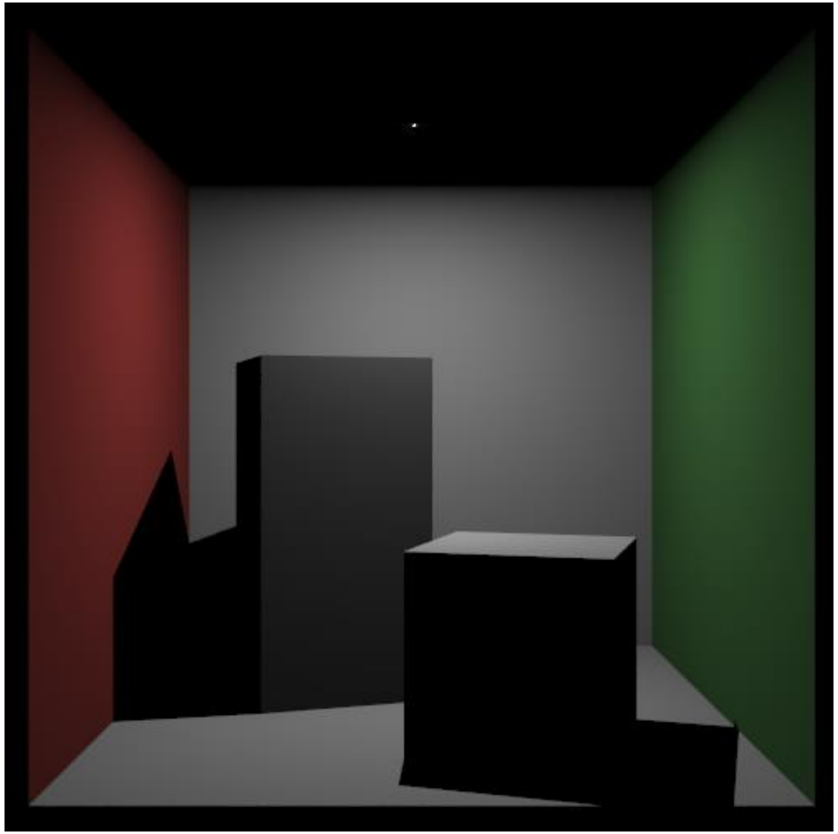


The role of the illumination model is to determine how much light is reflected to the viewer from a visible point on a surface as a function of light source direction and strength, viewer position, surface orientation, and surface properties. The shading calculations can be performed on three scales: microscopic, local, and global. Although the exact nature of reflection from surfaces is best explained in terms of microscopic interactions between light rays and the surface [3], most shaders produce excellent results using aggregate local surface data. Unfortunately, these models are usually limited in scope, i.e., they look only at light source and surface orientations, while ignoring the overall setting in which the surface is placed. The reason that shaders tend to operate on local data is that traditional visible surface algorithms cannot provide the necessary global data.

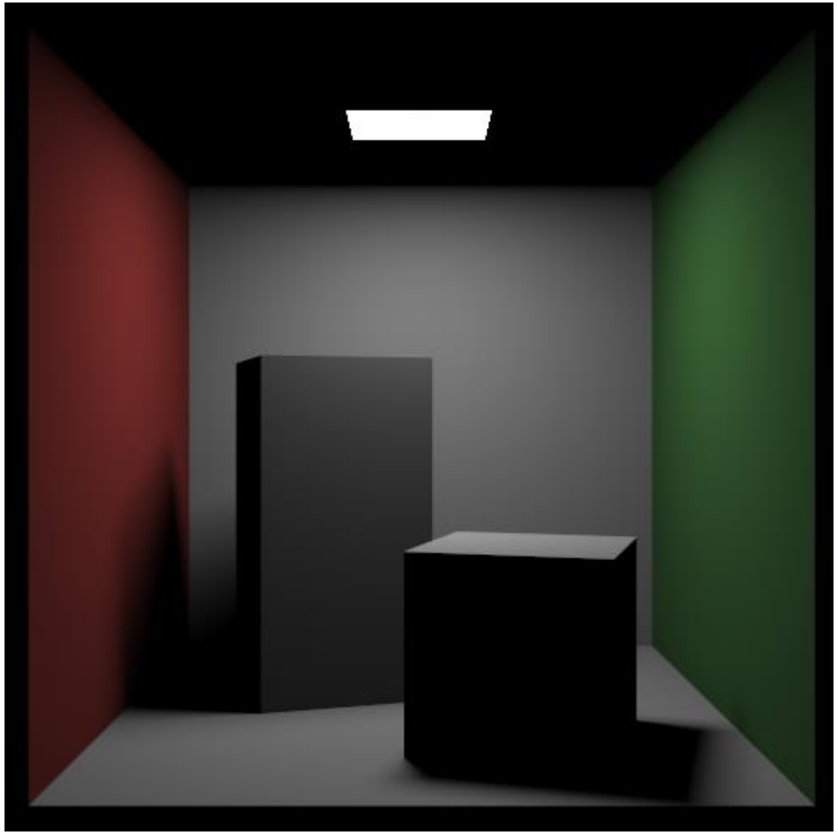


$$I = I_a + k_d \sum_{j=1}^{j=ls} (\bar{N}\cdot\bar{L}_j) + k_s \sum_{j=1}^{j=ls} (\bar{N}\cdot\bar{L}_j')^n, \qquad (1)$$

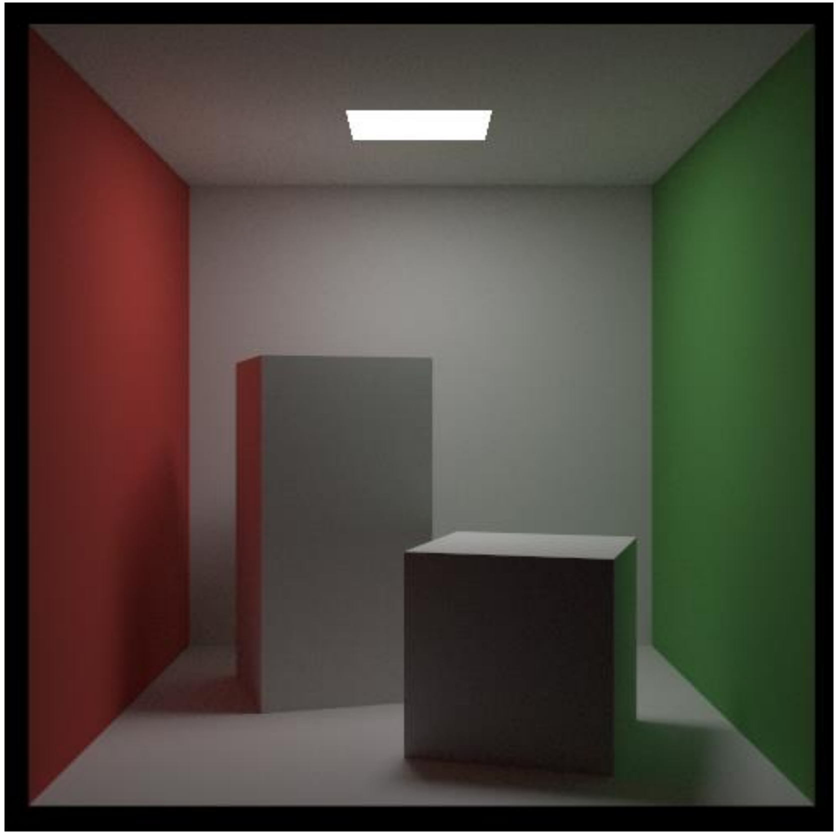
HARD SHADOWS



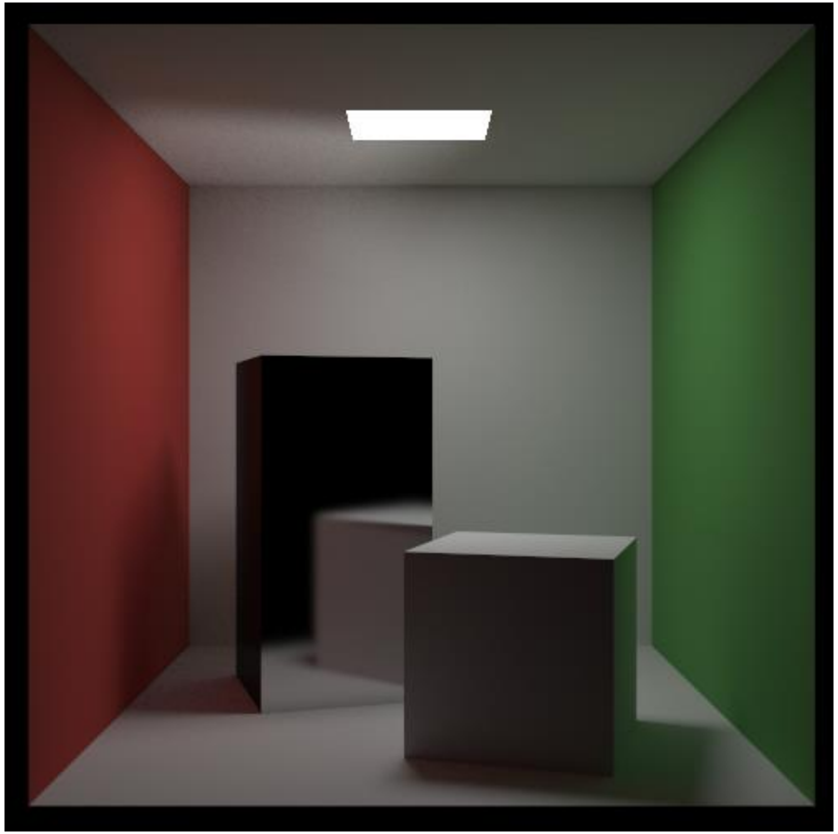
SOFT SHADOWS



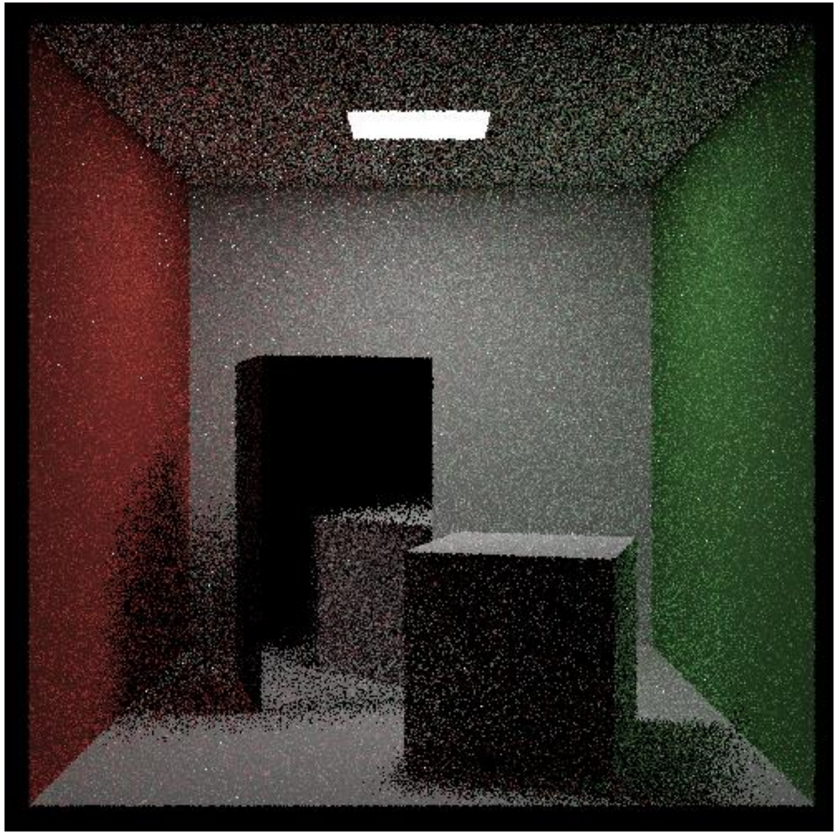
DIFFUSE INTER-REFLECTION



SPECULAR REFLECTION



ONE SAMPLE PER PIXEL



RAY TRACING ALGORITHMS

$$\xi_1 = F_G(\gamma') = \int_0^{\gamma'} f_G(\gamma)\, d\gamma = 2\gamma' - {\gamma'}^2$$

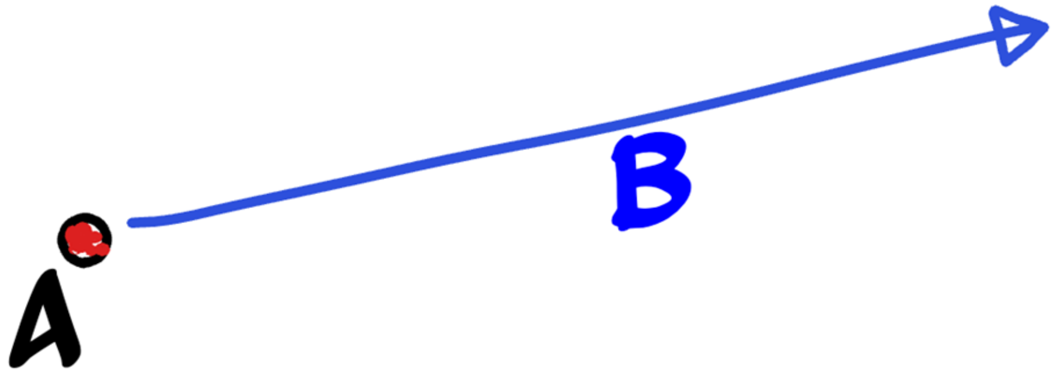
Solving for γ' we get $\gamma' = 1 - \sqrt{1 - \xi_1}$. For β'

$$\xi_2 = F_{B|G}(\beta'|\gamma') = \int_0^{\beta'} f_{B|G}(\beta'|\gamma')\, d\beta = \frac{\beta'}{1 - \gamma'}$$

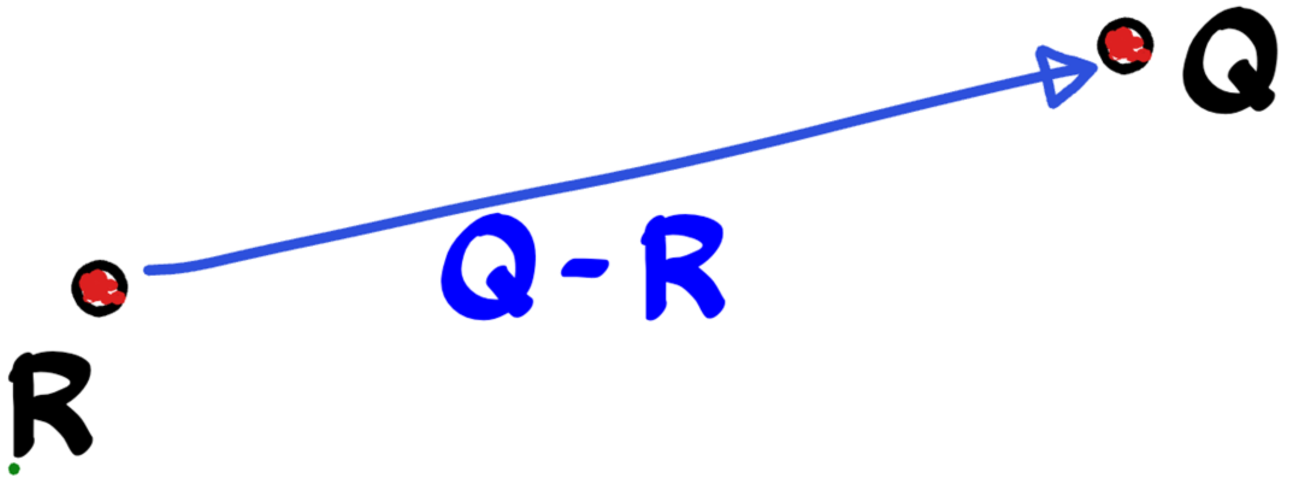
thus $\beta' = \xi_2(1 - \gamma') = \xi_2\sqrt{1 - \xi_1}$. Therefore

$$\mathbf{x}' = \mathbf{p}_0 + \xi_2\sqrt{1 - \xi_1}\left(\overrightarrow{\mathbf{p}_1 - \mathbf{p}_0}\right) + 1 - \sqrt{1 - \xi_1}\left(\overrightarrow{\mathbf{p}_2 - \mathbf{p}_0}\right)$$

A RAY IS A LOCATION AND A DIRECTION

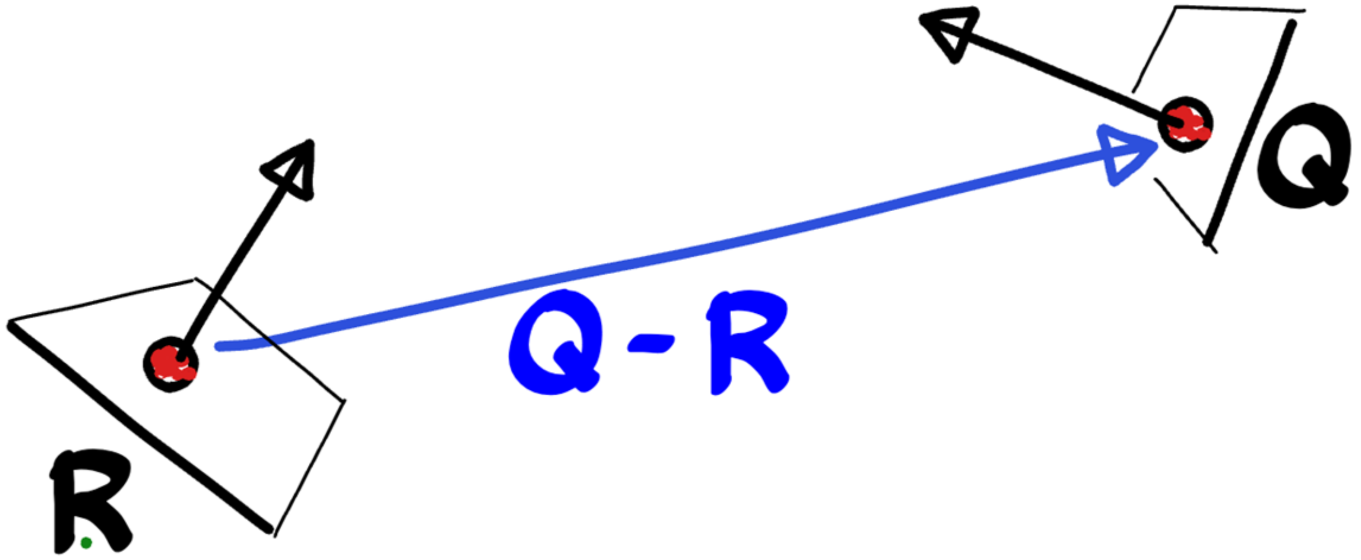


A RAY CAN TEST VISIBILITY BETWEEN TWO POINTS



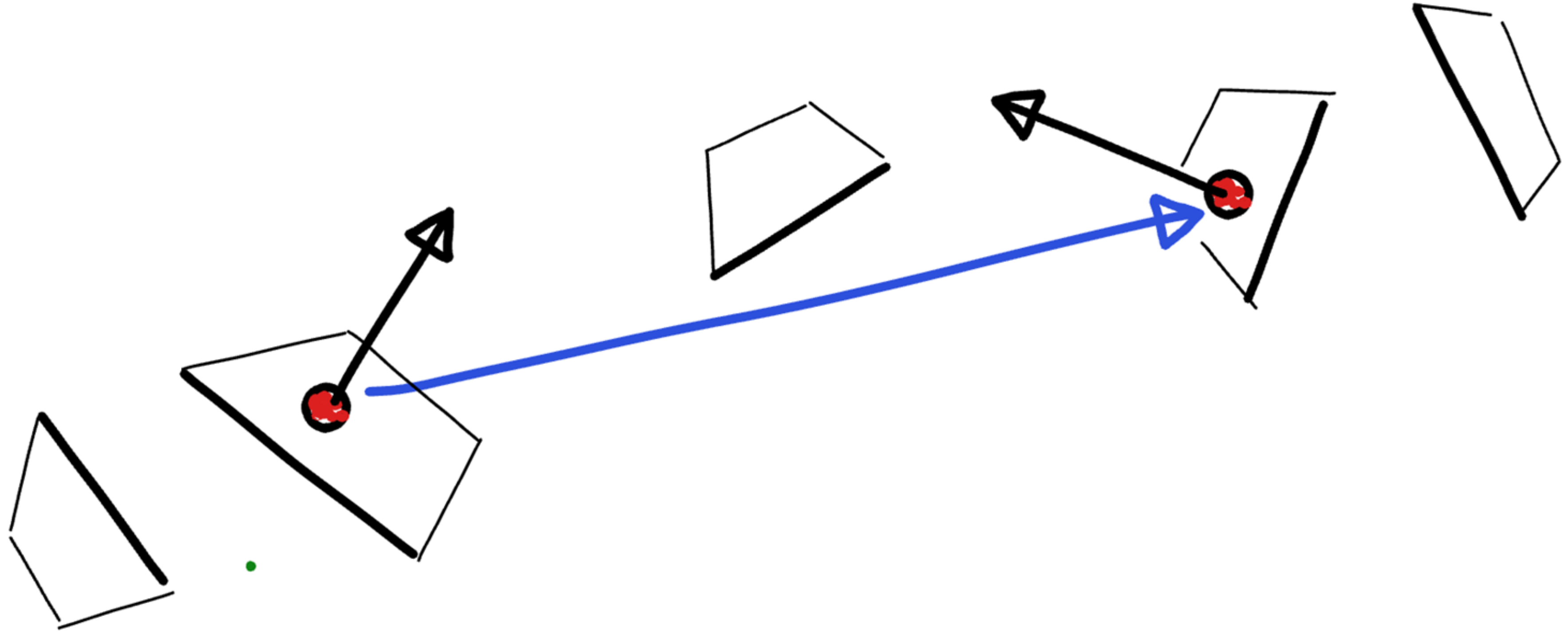
$$P(t) = R + t * (Q-R)$$

A RAY CAN TEST VISIBILITY BETWEEN TWO POINTS ON SURFACES



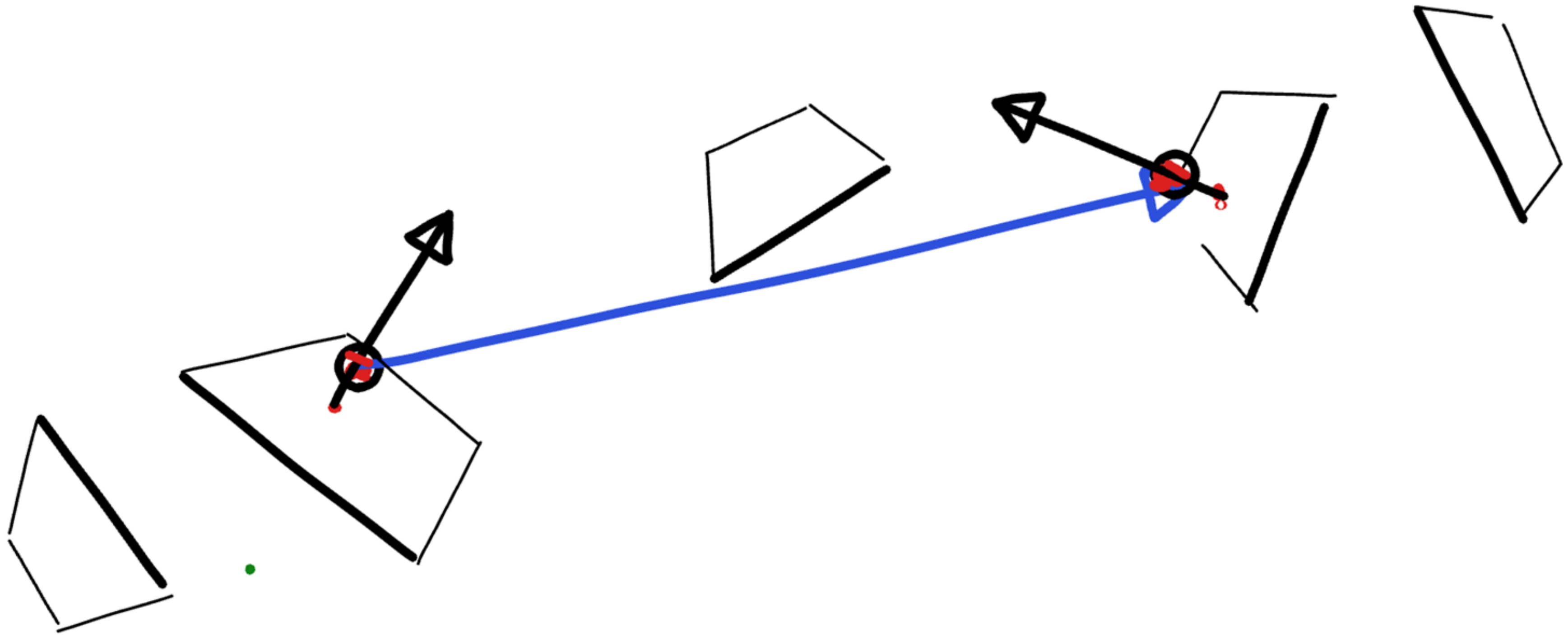
$$P(t) = R + t * (Q-R)$$

FLOATING POINT PAIN IN NECK



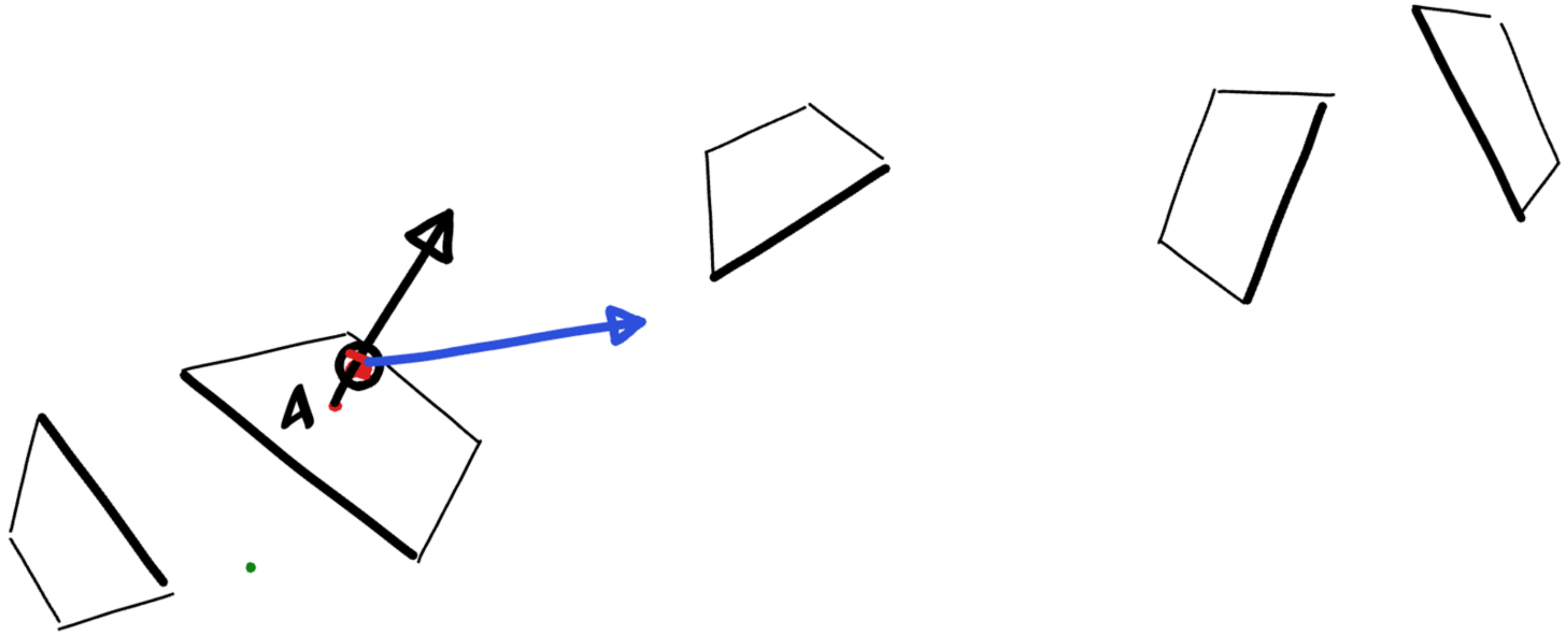
HITS AT T = -0.25, 0, 1, 1.25

FLOATING POINT PAIN IN NECK

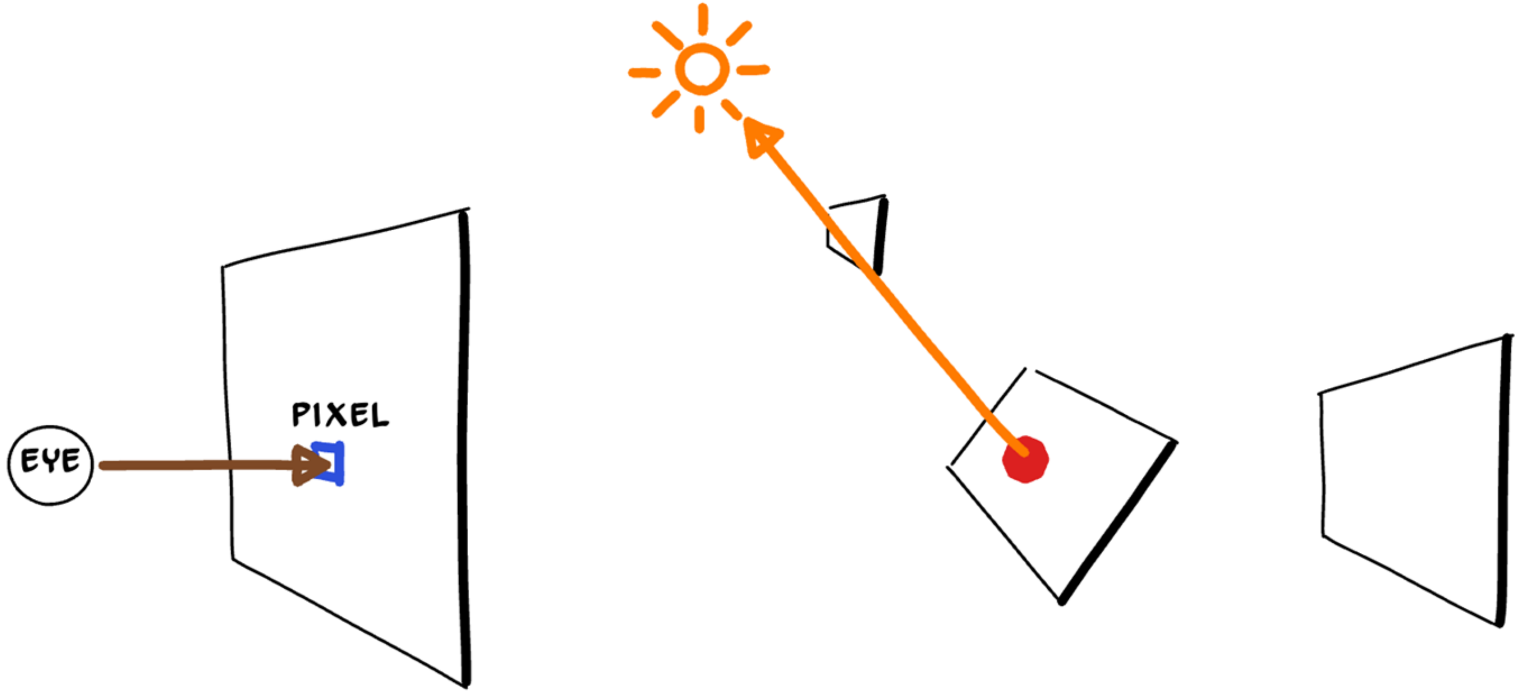


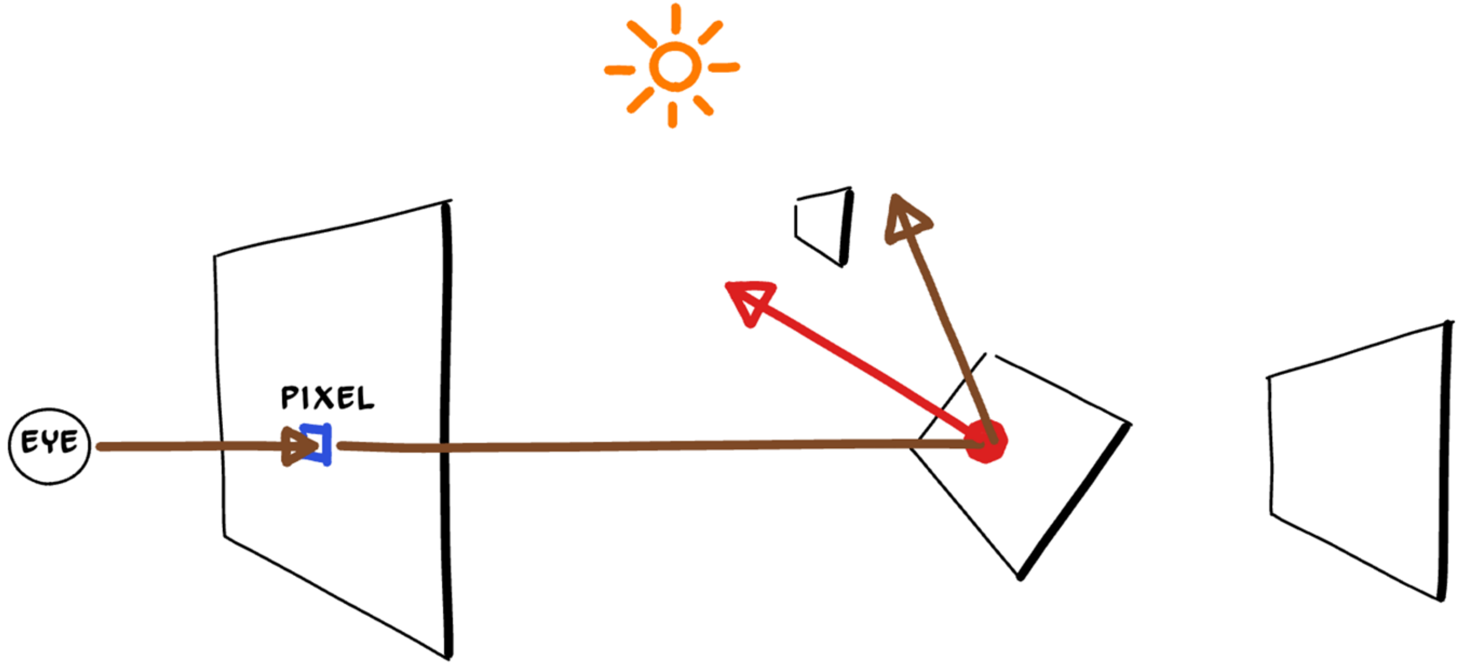
HITS AT T = -0.25, -0.001, 1.001, 1.25

PROPAGATING RAYS



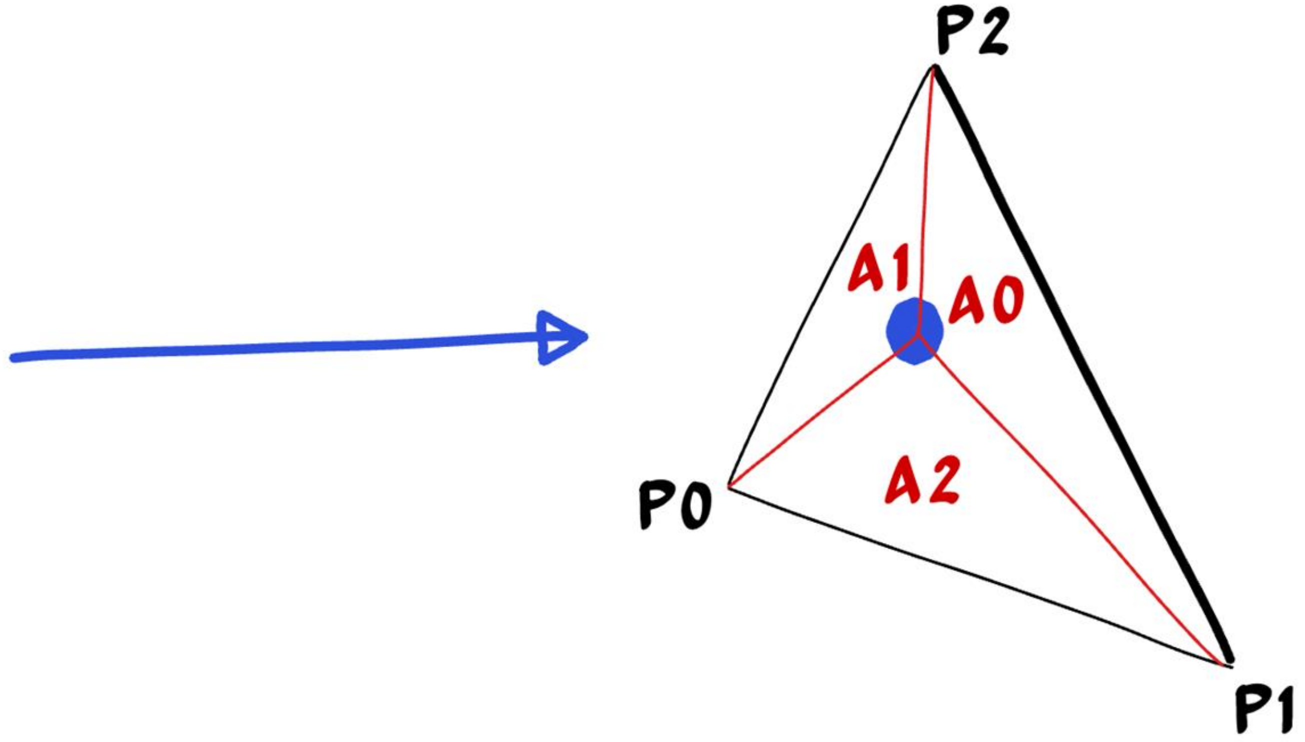
$$P(t) = A + t * B$$

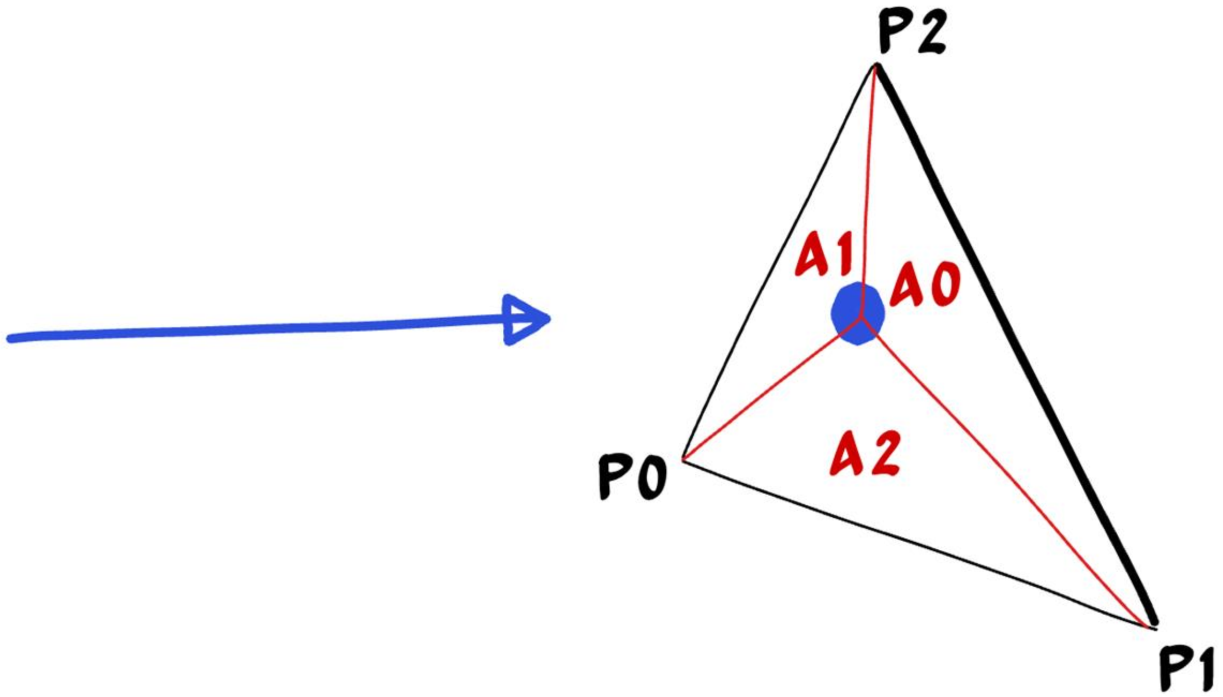

PIXEL
EYE


EYE
PIXEL

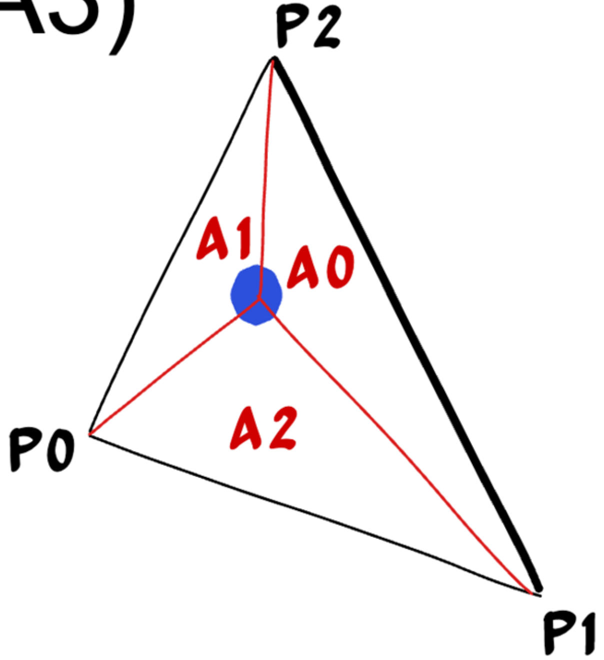
```
For each pixel_ij
   c_ij = radiance( ray(eye,pixel_center - eye)

rgb radiance(ray r)
    if Q = hit(r, 0.01, infinity)
       s = ray(Q, light-Q)
       if hit(s, 0.01,0.99)
           col = direct_light
       col += radiance(ray(Q, reflection))
       return col
    else
       return background(r.direction)
```

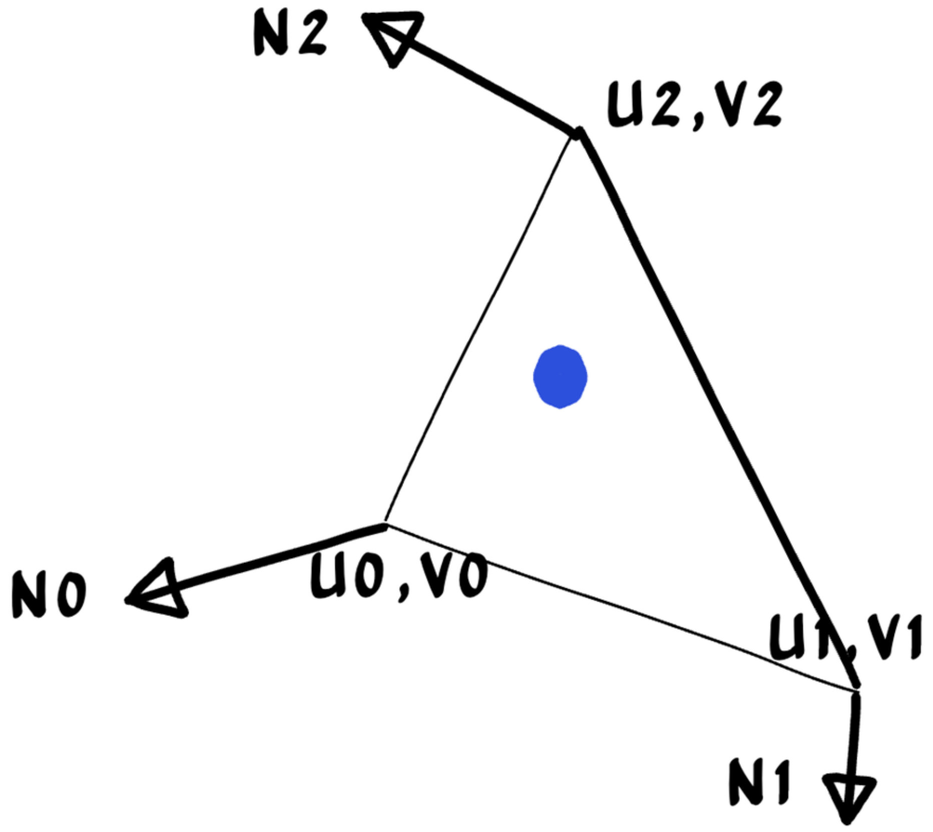

P2
A1
A0
A2
P0
P1


P2
A1
A0
A2
P0
P1

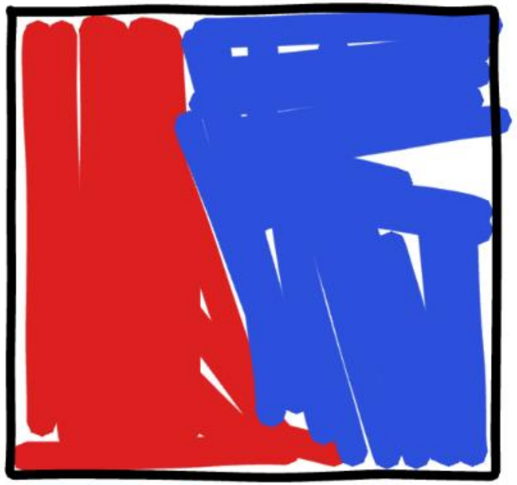
$$w1 = A1/A$$
$$= A1/(A1+A2+A3)$$

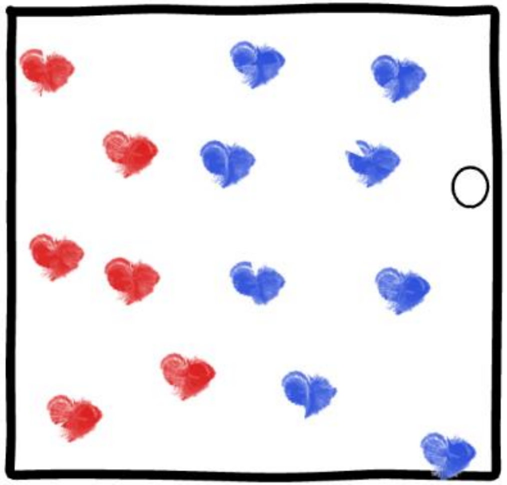


$$u = w0*u0 + w1*u1 + w2*u2$$

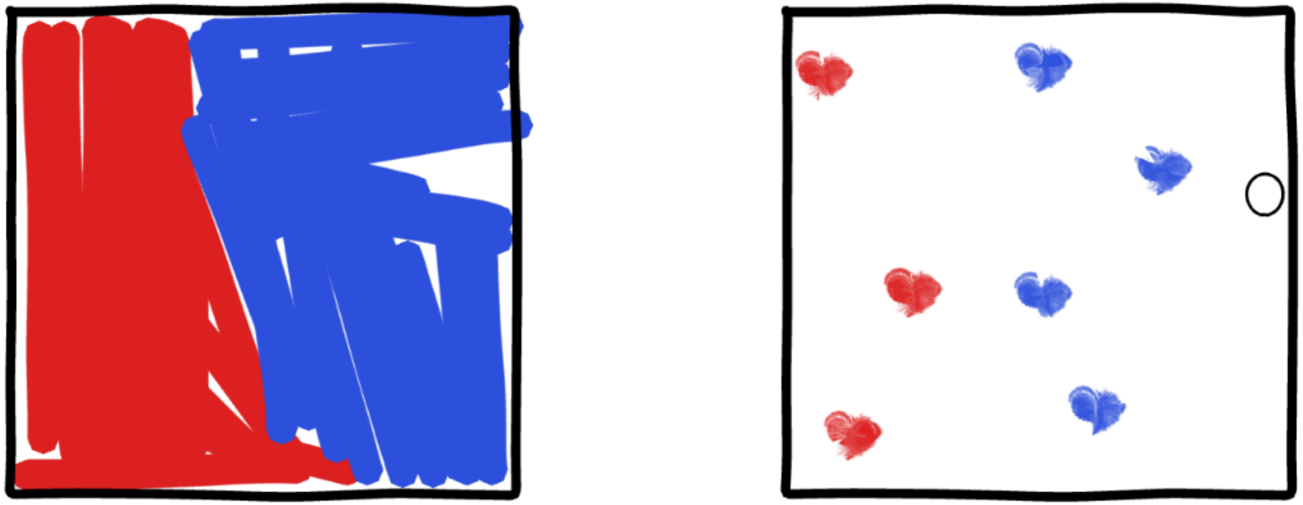


WEIGHTED AVERAGES EVERYWHERE

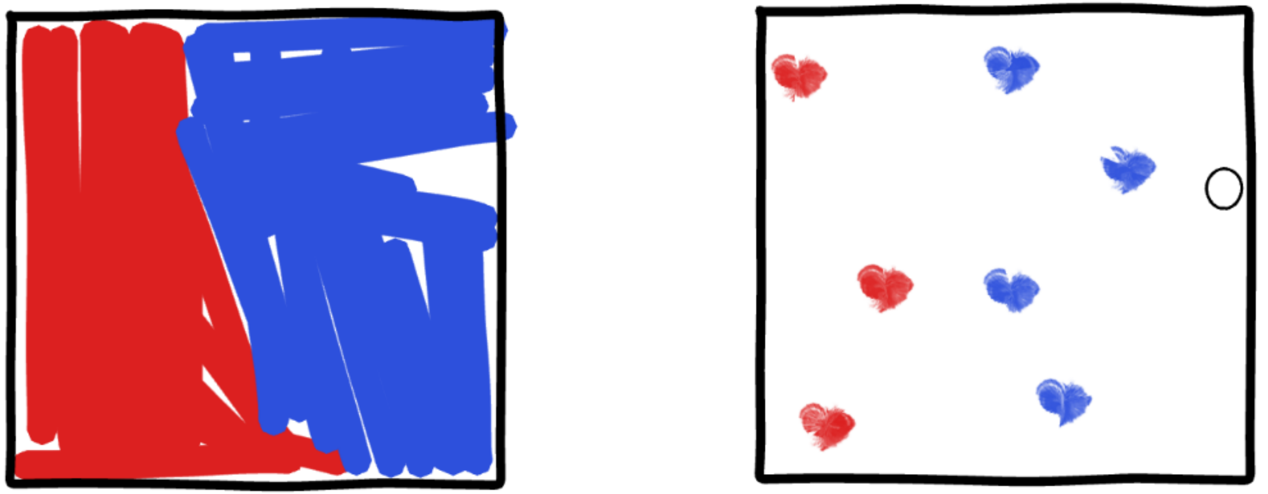




WEIGHTED AVERAGES EVERYWHERE

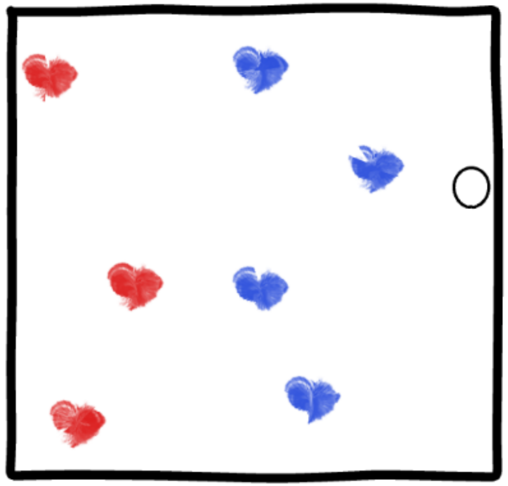


WEIGHTED AVERAGES EVERYWHERE



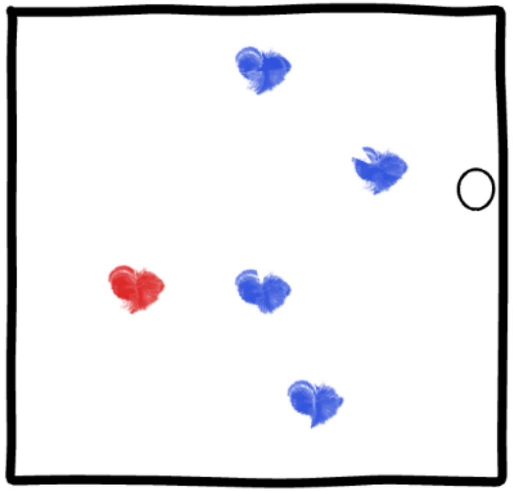
COLOR = $\dfrac{❤ + ❤ + ❤ + 💙 + 💙 + 💙 + 💙 + ○}{8}$

WEIGHTED AVERAGES EVERYWHERE



$$\text{COLOR} = \frac{7❤️ + ❤️ + ❤️ + 💙 + 💙 + 💙 + 💙 + ○}{14}$$

WEIGHTED AVERAGES EVERYWHERE



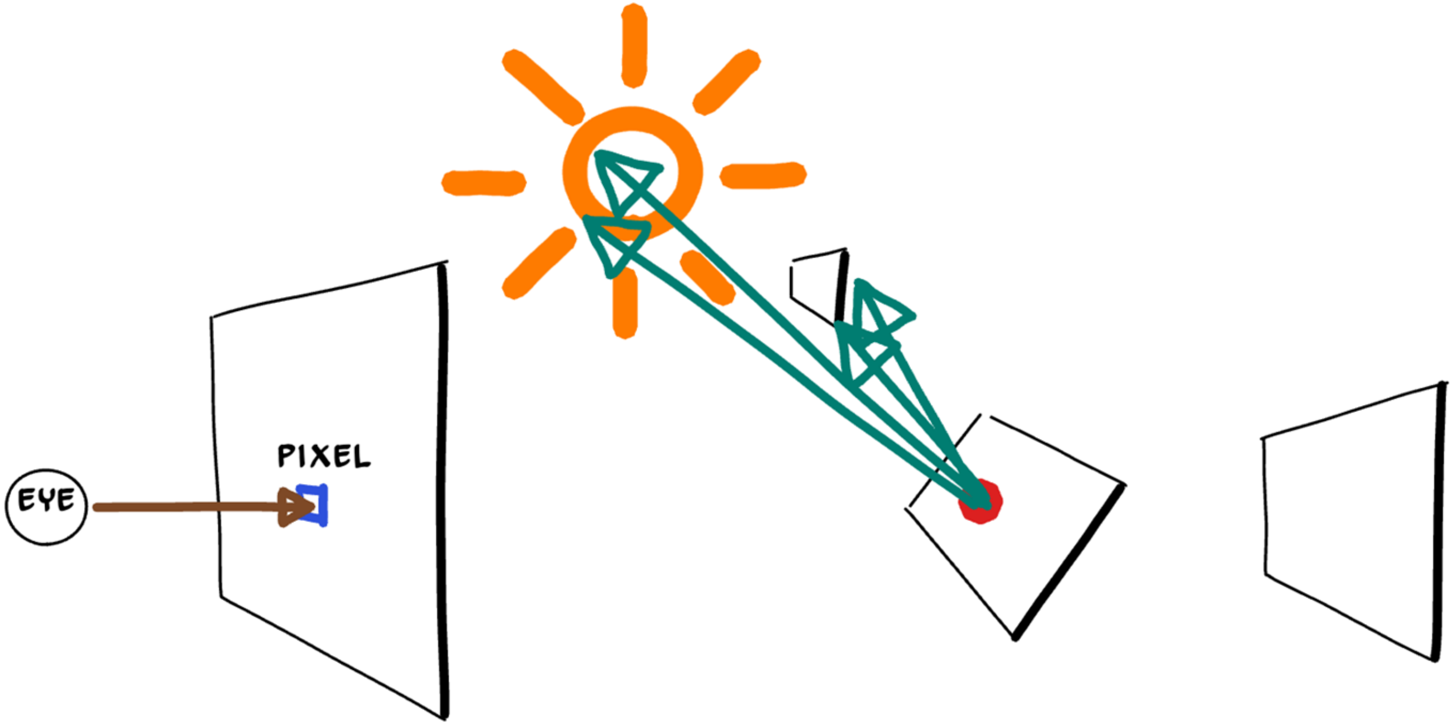
COLOR = (3❤️ + 💙 + 💙 + 💙 + 💙 + ○) / 8

SAMPLES CAN BE ANYTHING!
QMC, MONTE CARLO, BLUE NOISE

IF YOU SAMPLE NON-UNIFORMLY, THEN
WEIGHT THEM APPROPRIATELY

IF THIS MAKES YOUR PROGRAM BETTER
THEN YOU ARE IMPORTANCE SAMPLING

CHOOSING WEIGHTS EXACTLY RIGHT IS
"MONTE CARLO INTEGRATION"


PIXEL
EYE

Without random sampling

- **Q** = light center
- **P** = shaded point
- Shadow ray = **P** + t*(**Q-P**)
- Is there a hit for *t* in [0.001, 0.999]?

Without random sampling

Q = light center

P = shaded point

Shadow ray = **P** + t*(**Q-P**)

Is there a hit for *t* in [0.001, 0.999]?

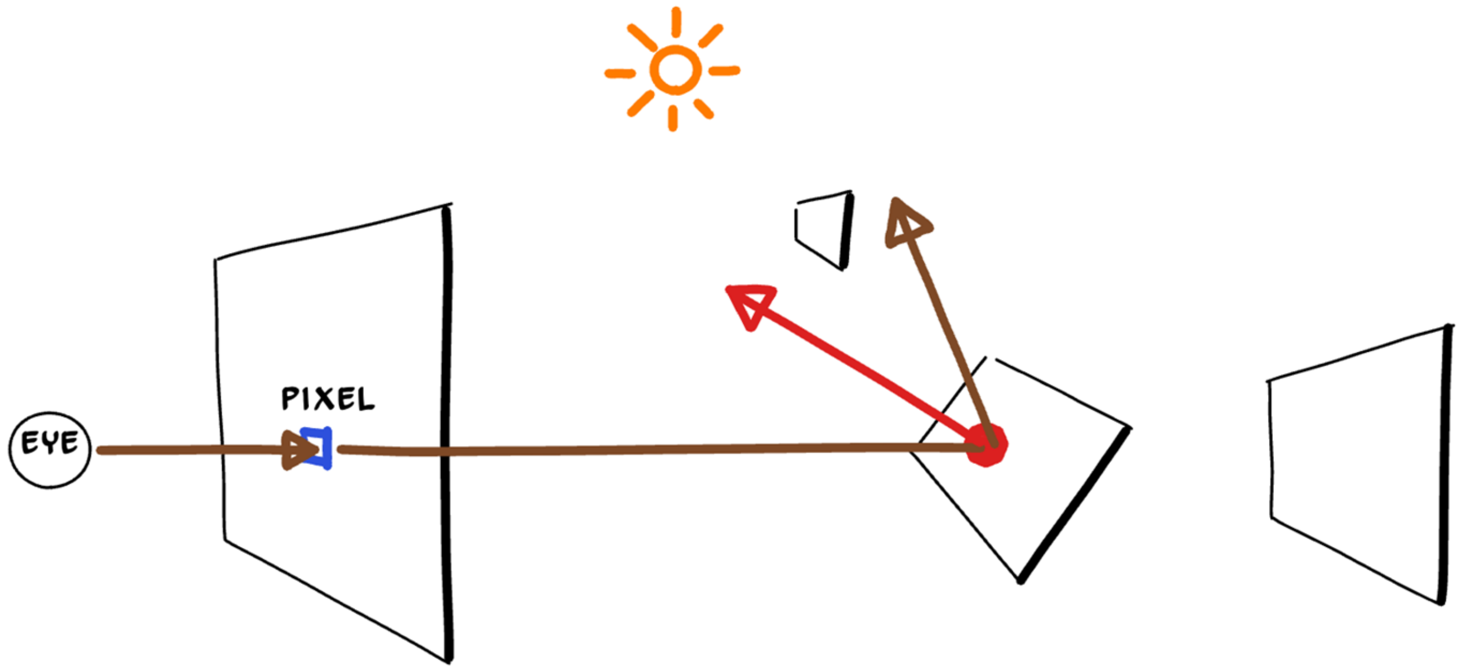
With random sampling

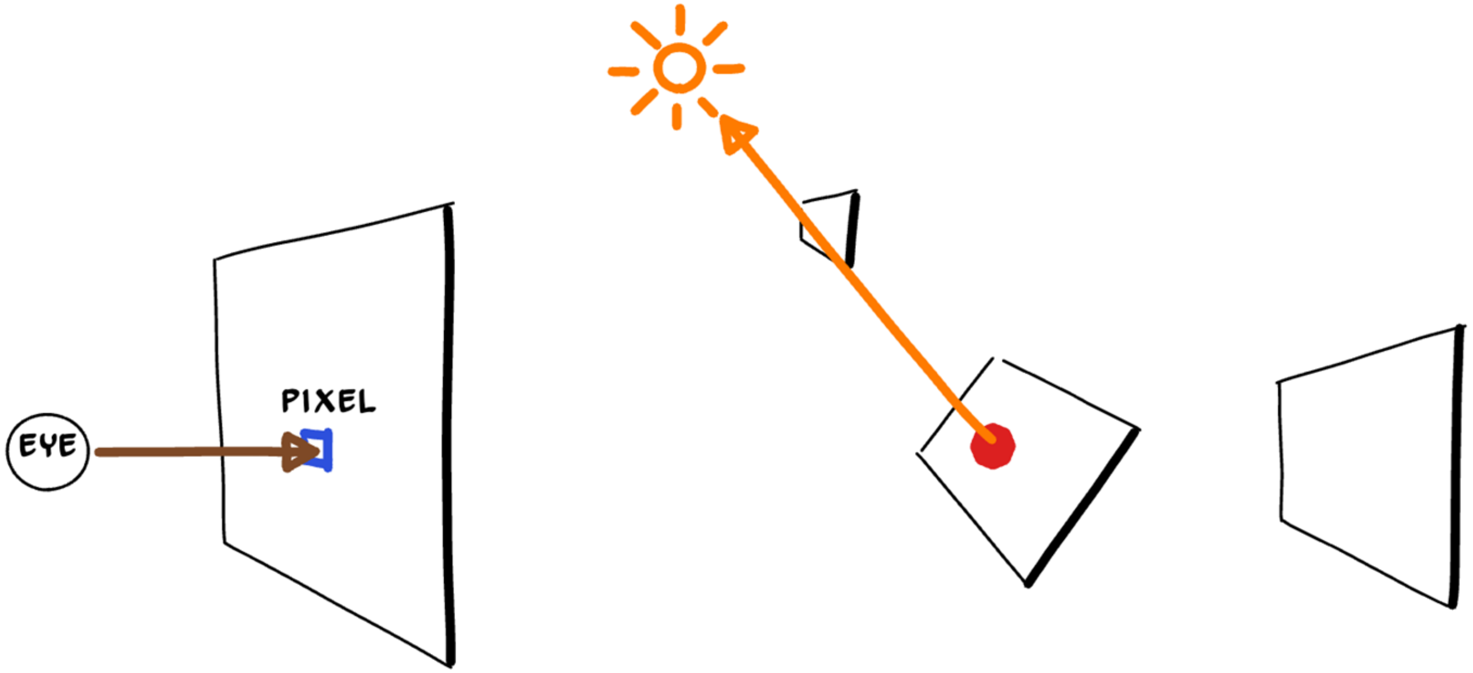
Q = light center + random_in_sphere_of_radius(0.3)

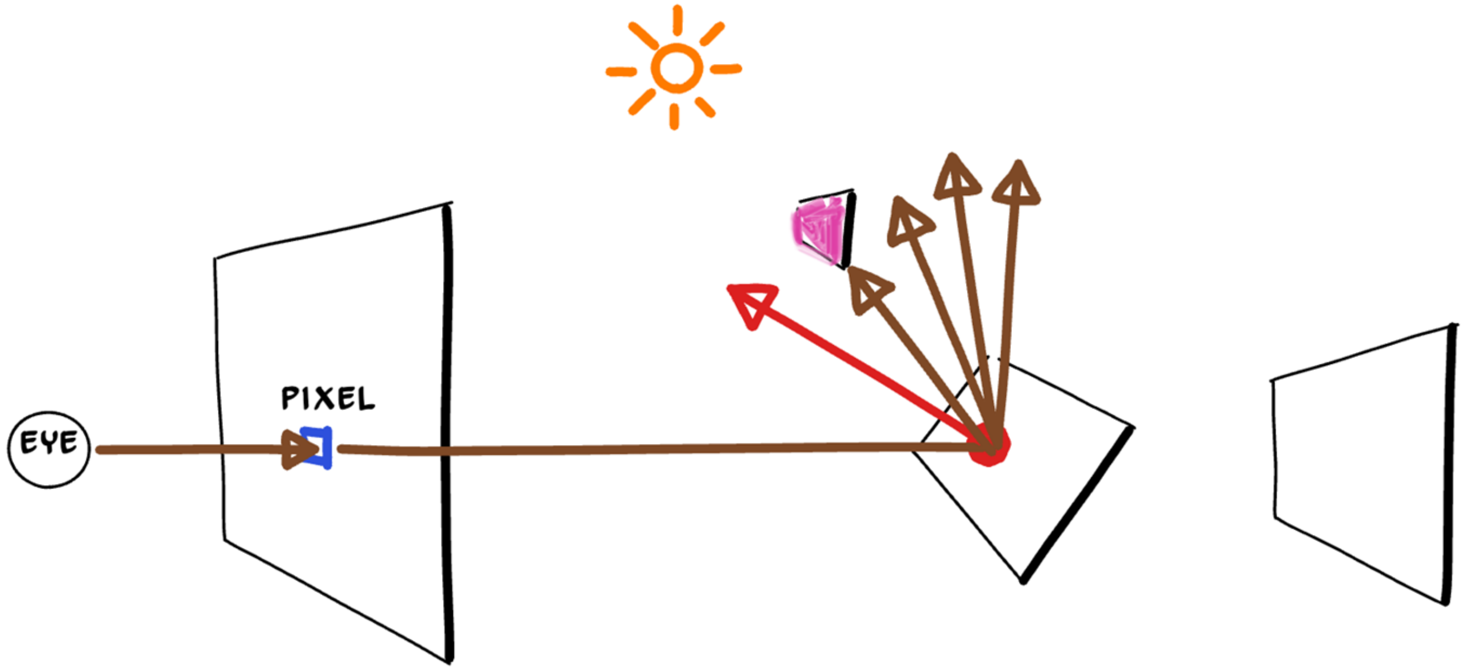
P = shaded point

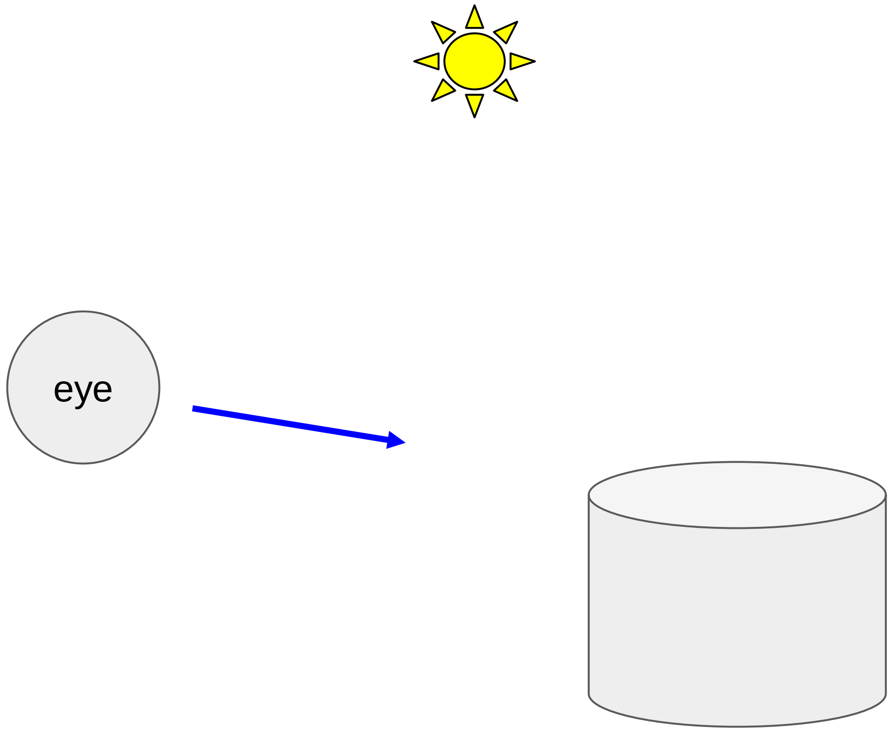
Shadow ray = **P** + t*(**Q-P**)

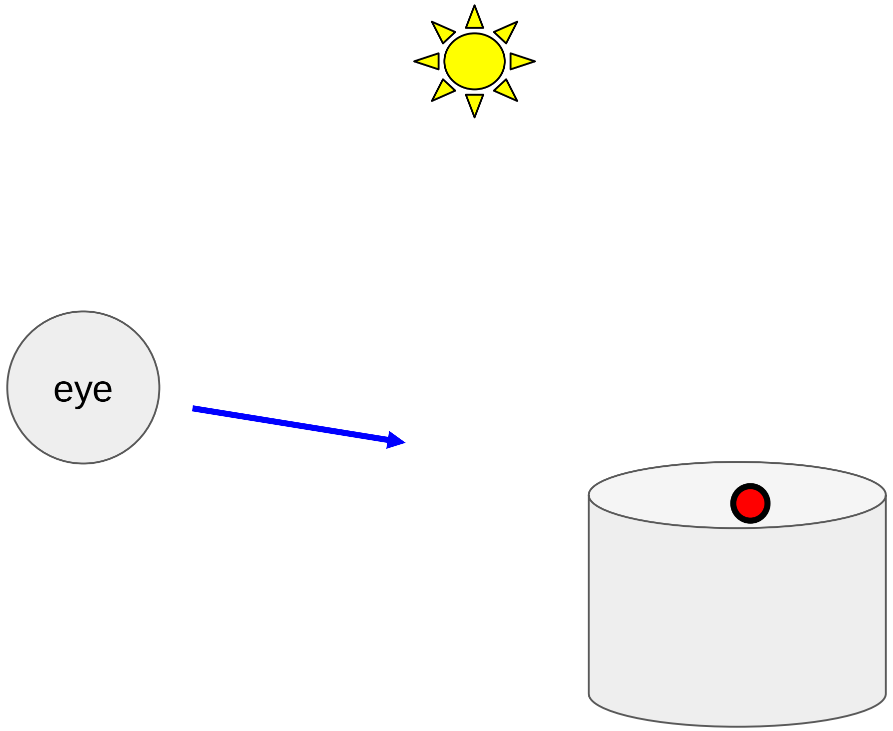
Is there a hit for *t* in [0.001, 0.999]?

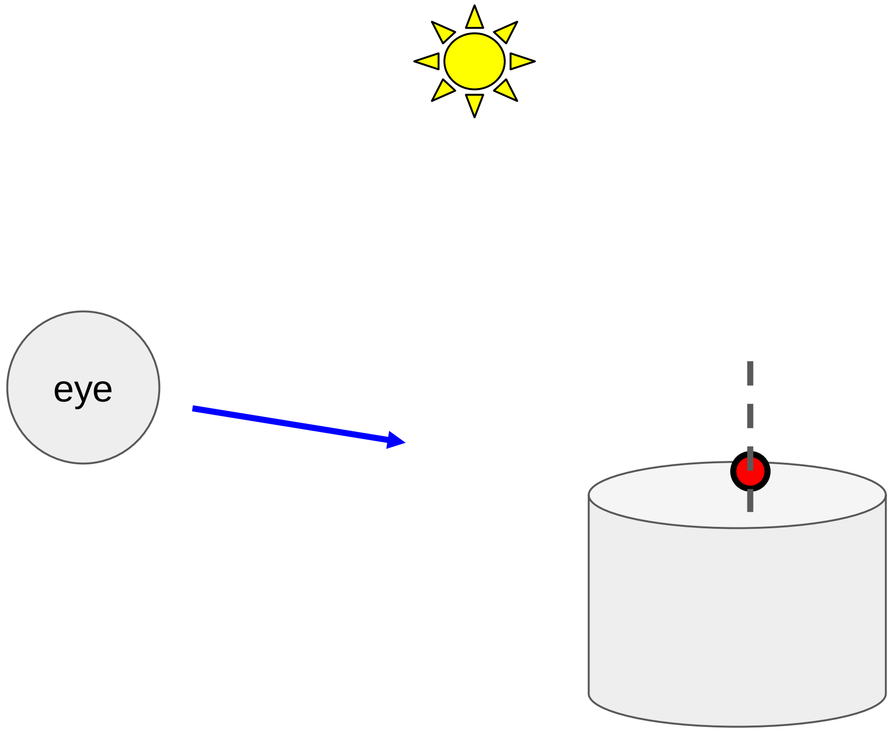

PIXEL
EYE

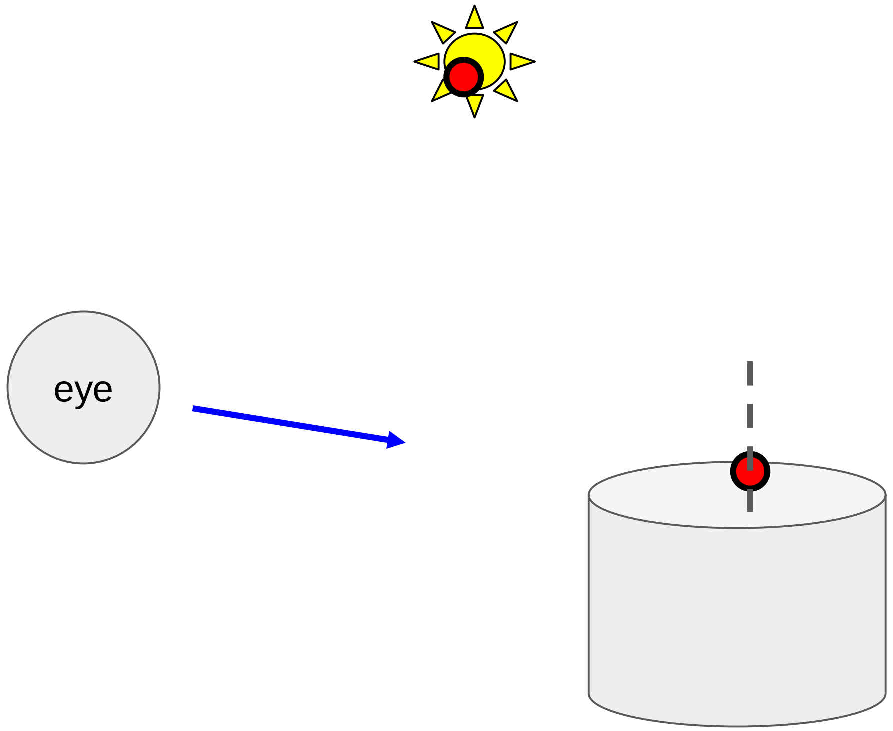

PIXEL
EYE

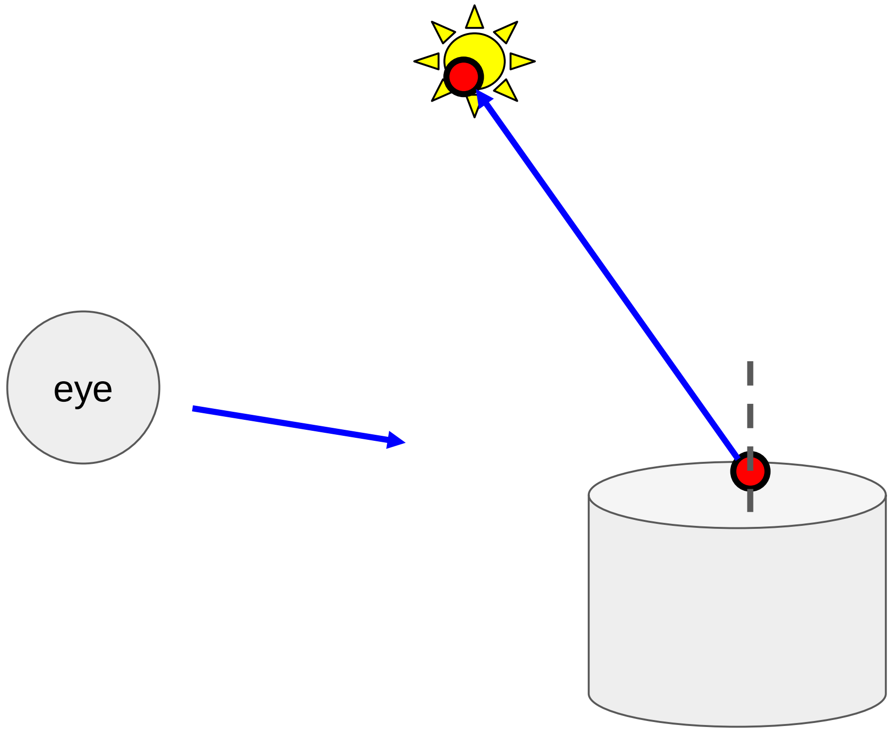

PIXEL
EYE

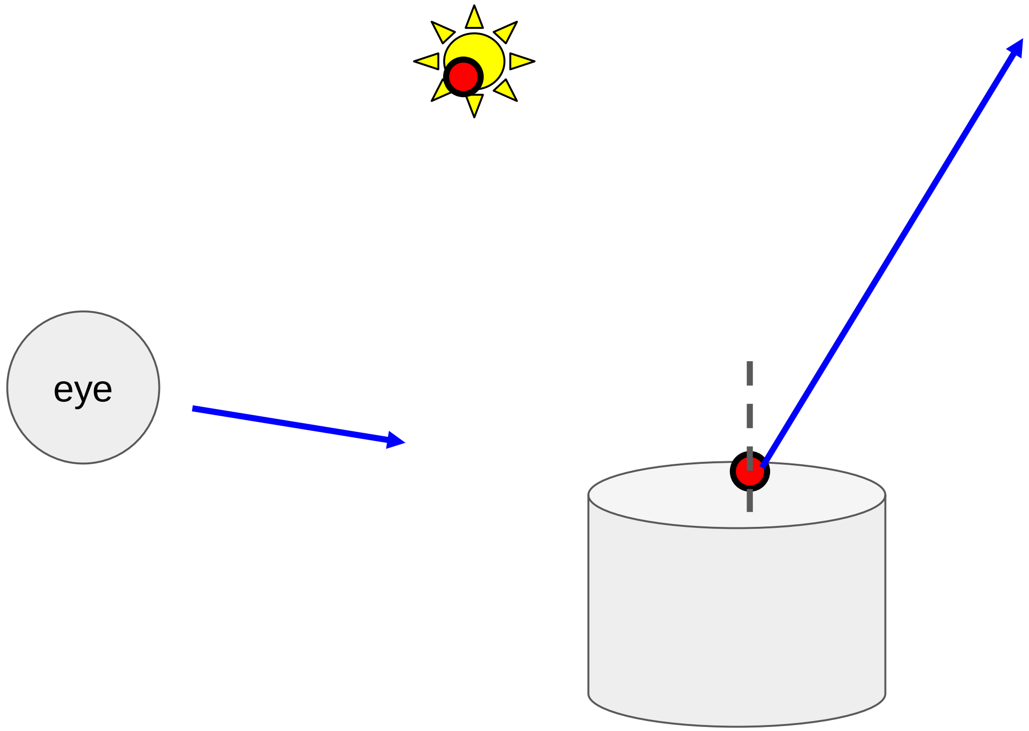

eye

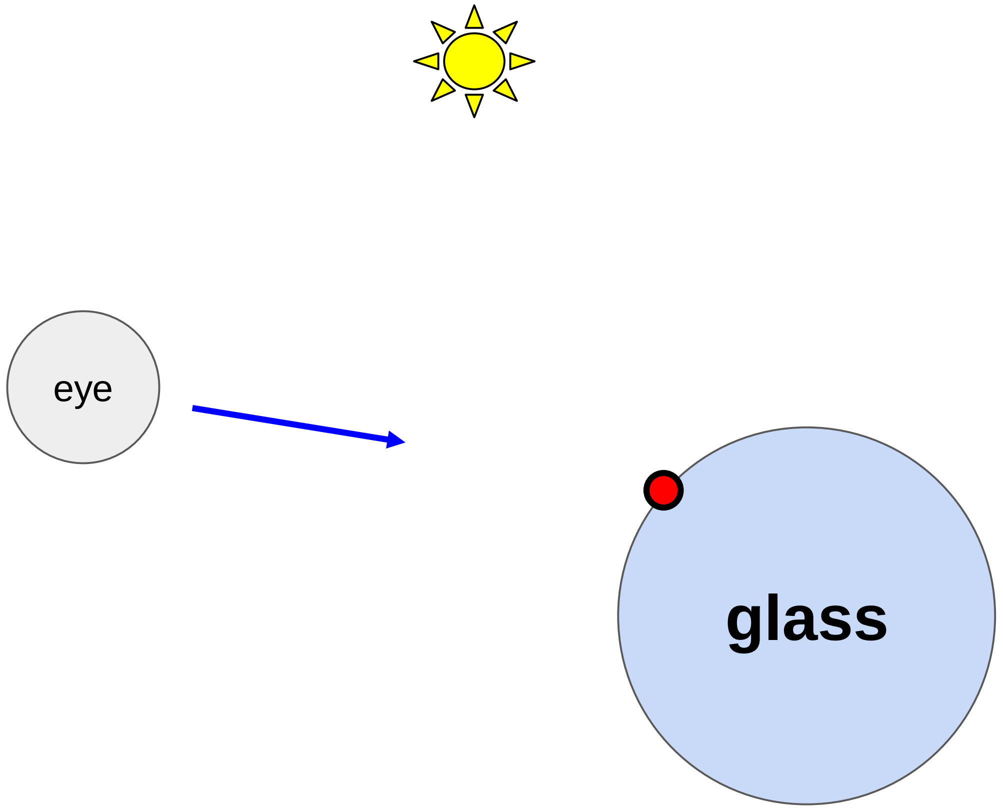

eye

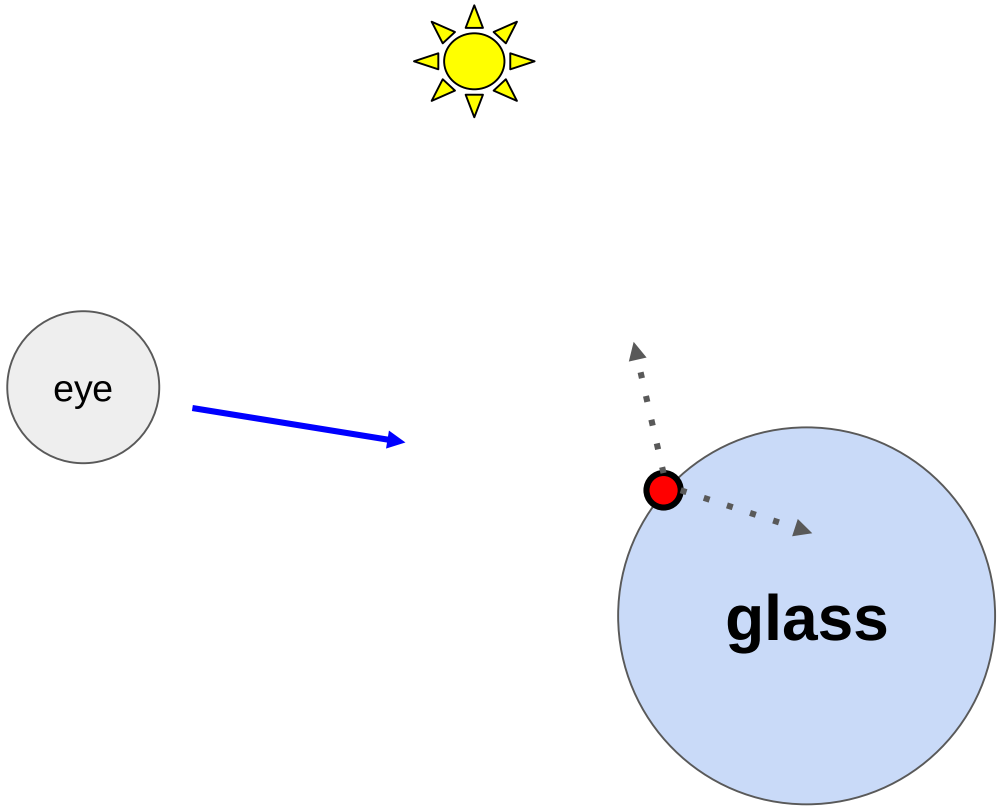

eye

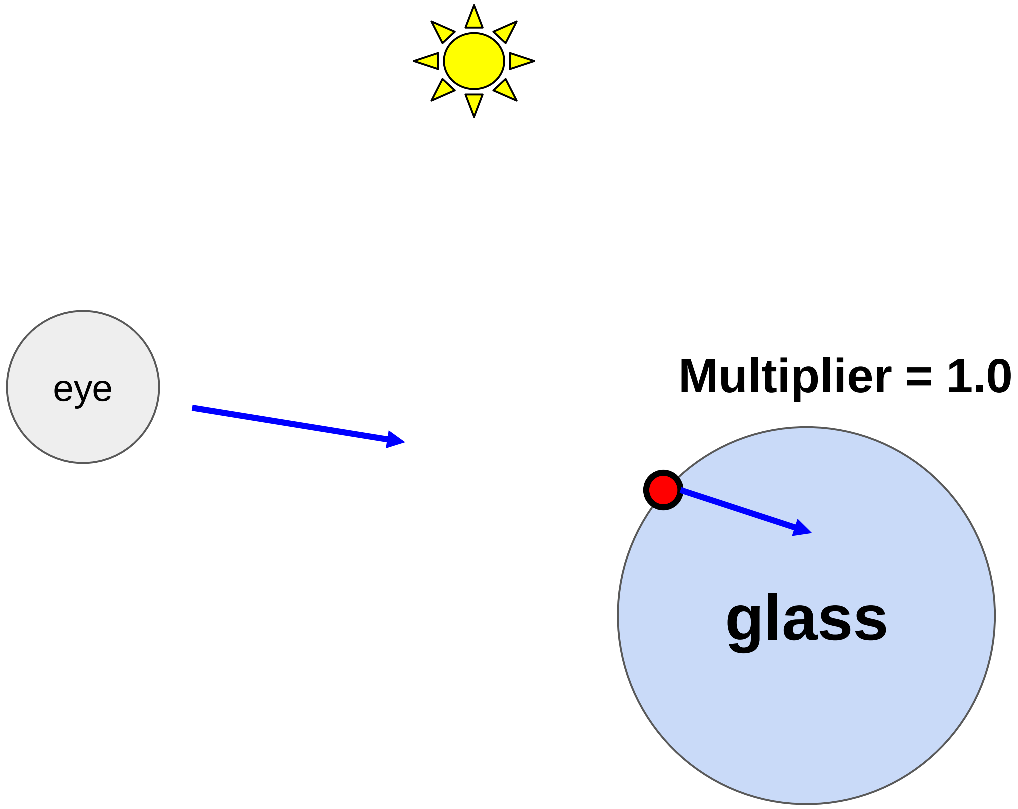

eye

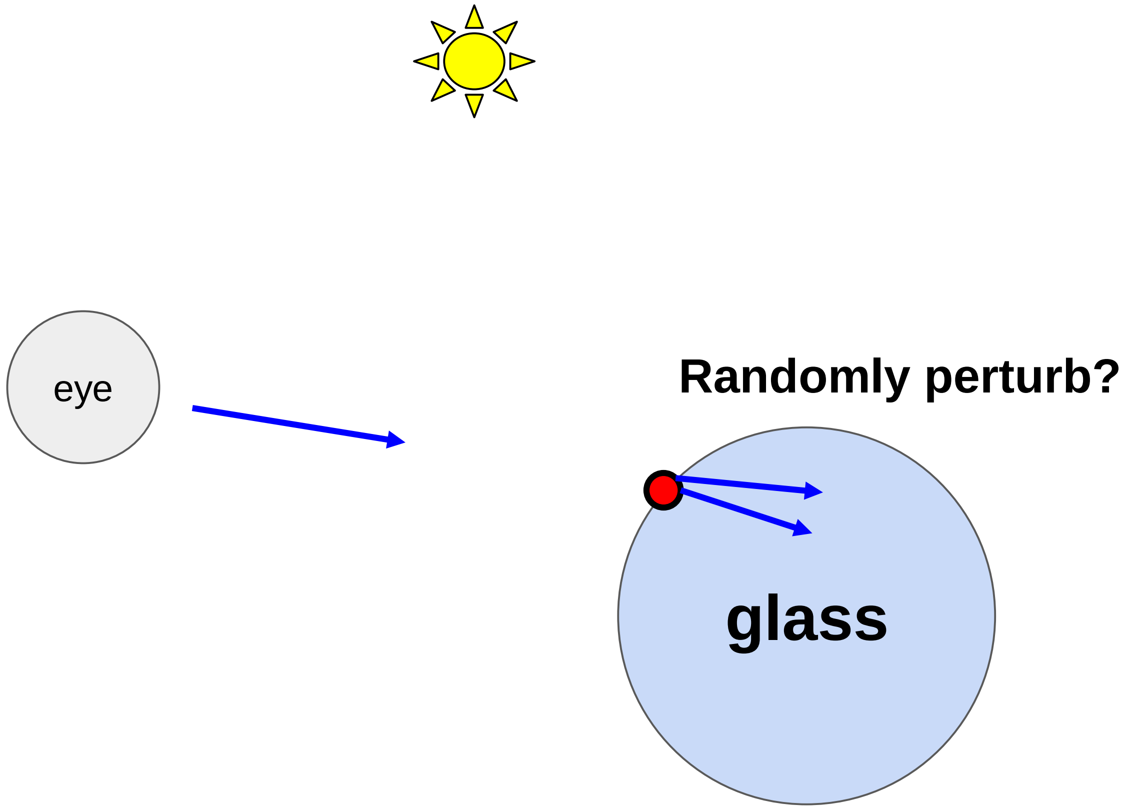

eye

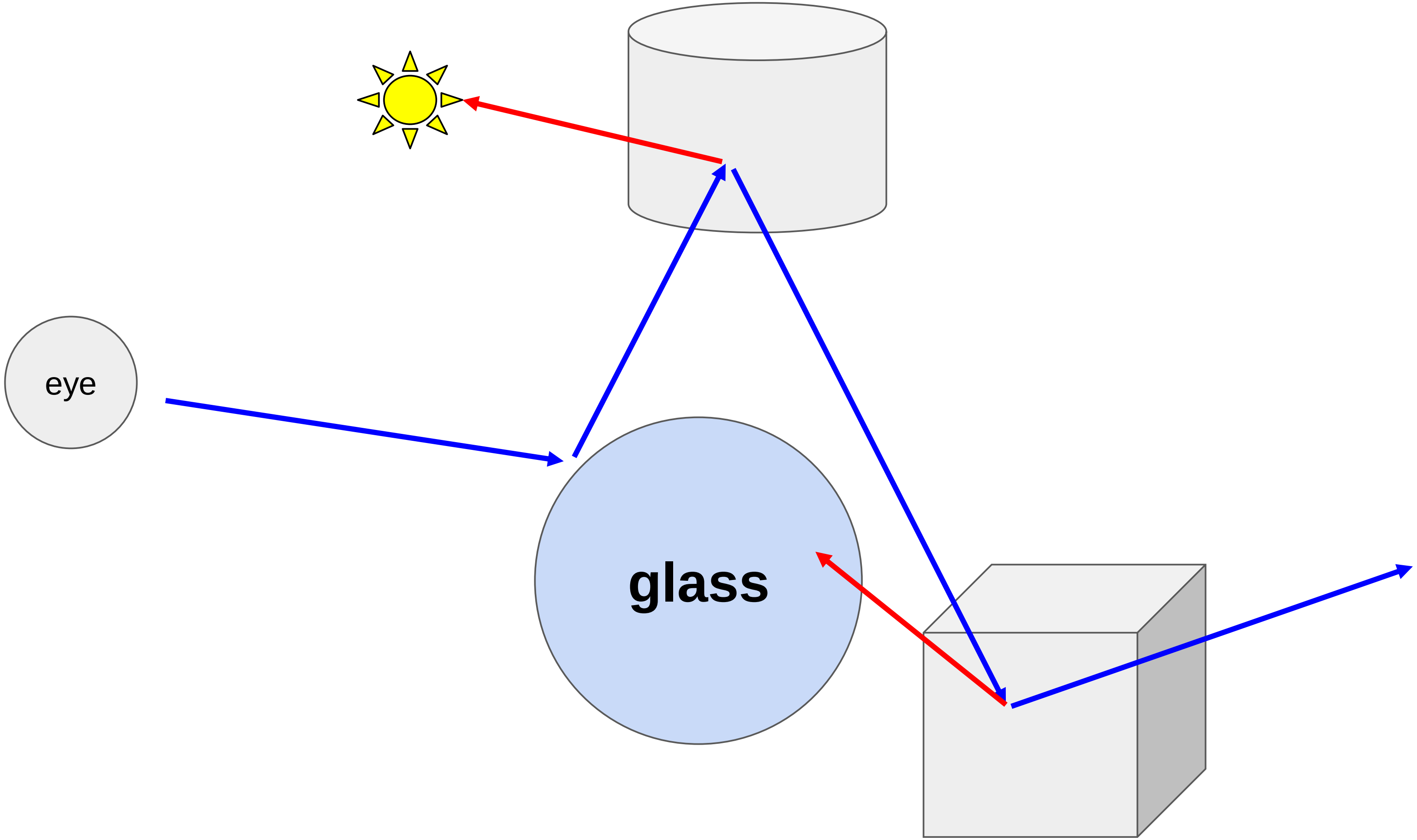

eye

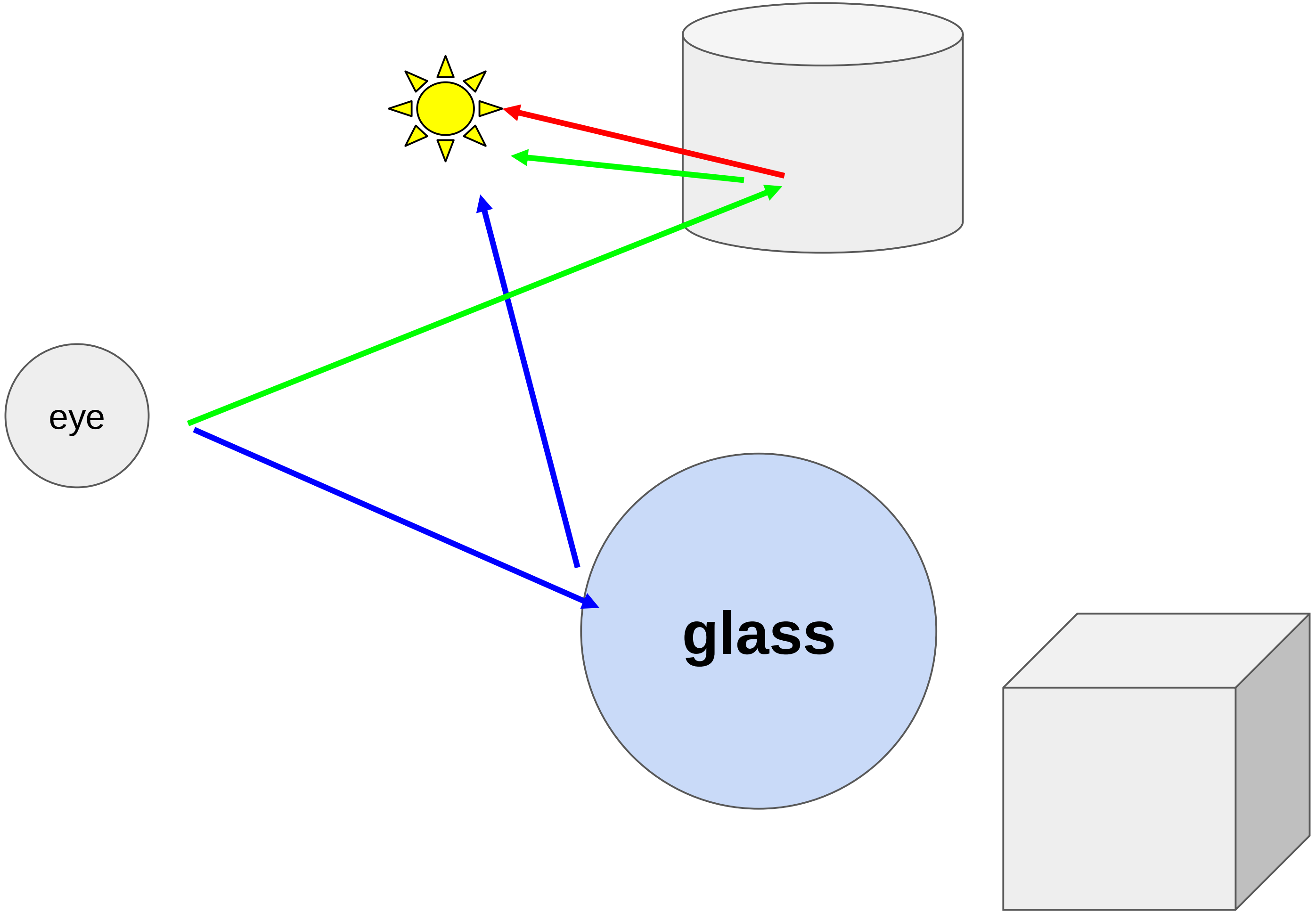

eye
glass


eye
glass

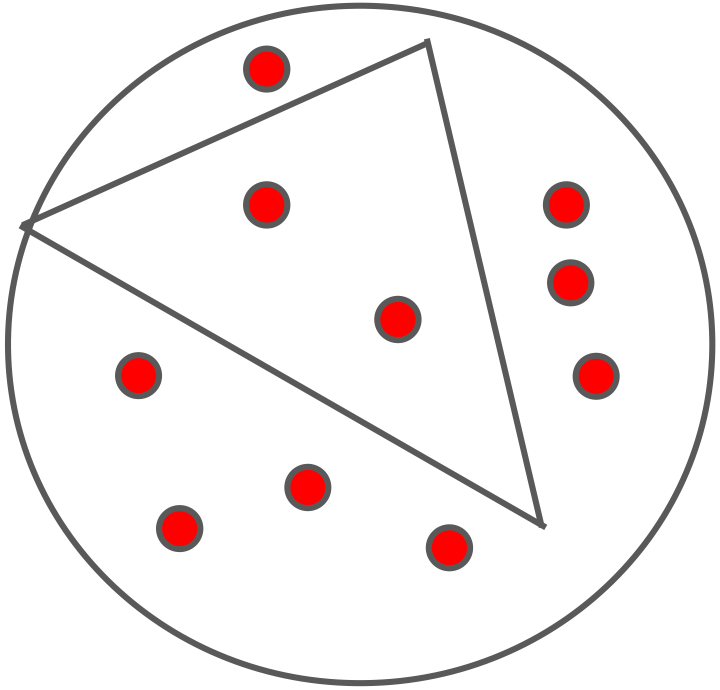

eye
Multiplier = 1.0
glass


eye
Randomly perturb?
glass

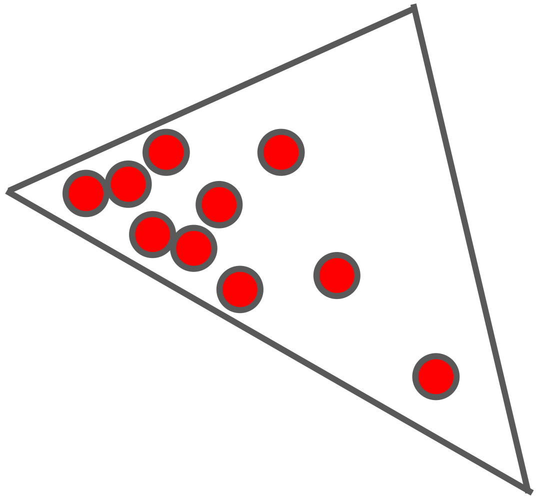

eye
glass


eye
glass

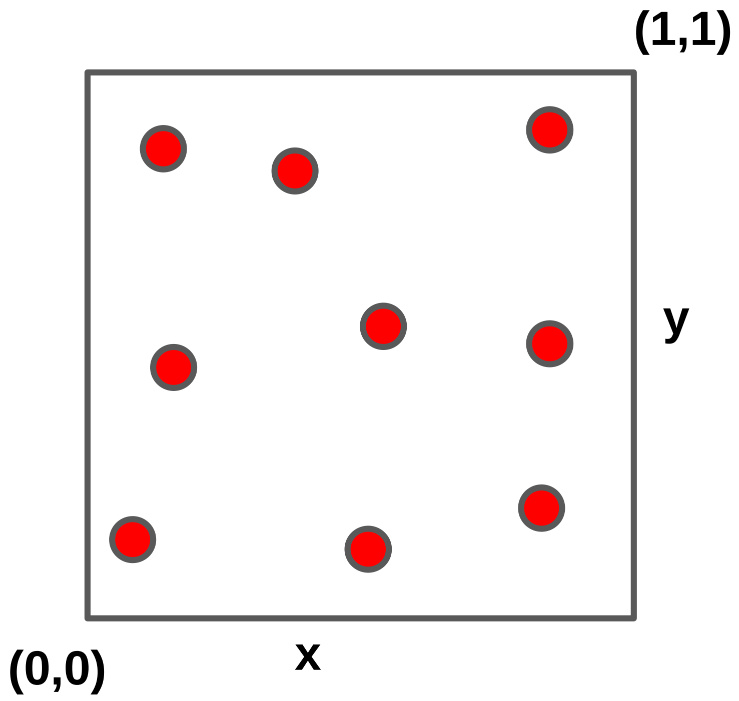
Random Points on Triangles

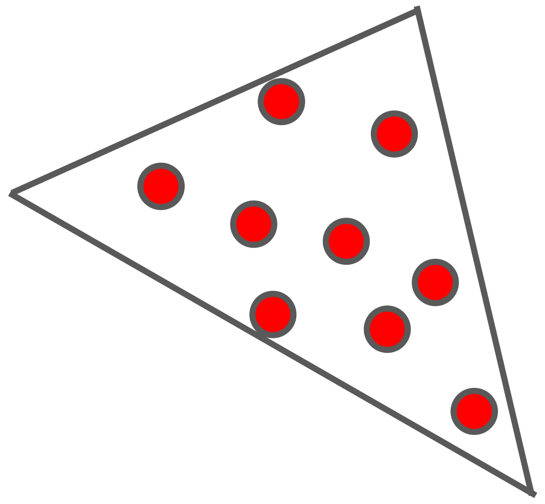


Random Points on Triangles



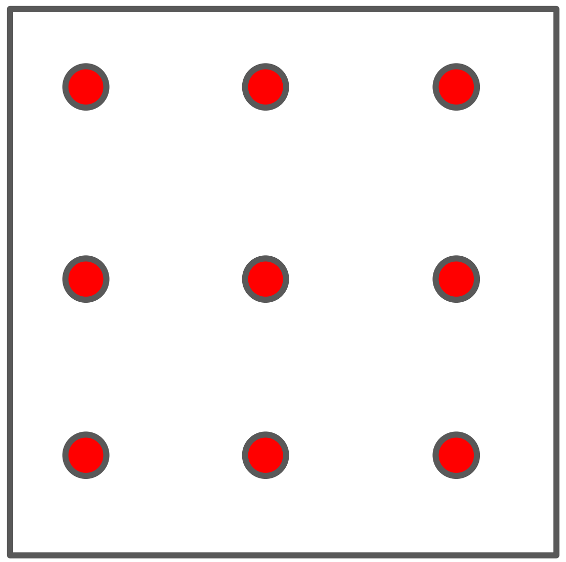
Random Points on Triangles

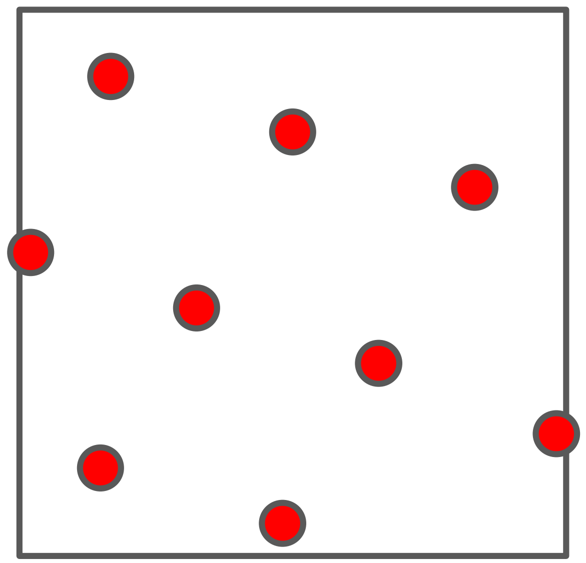


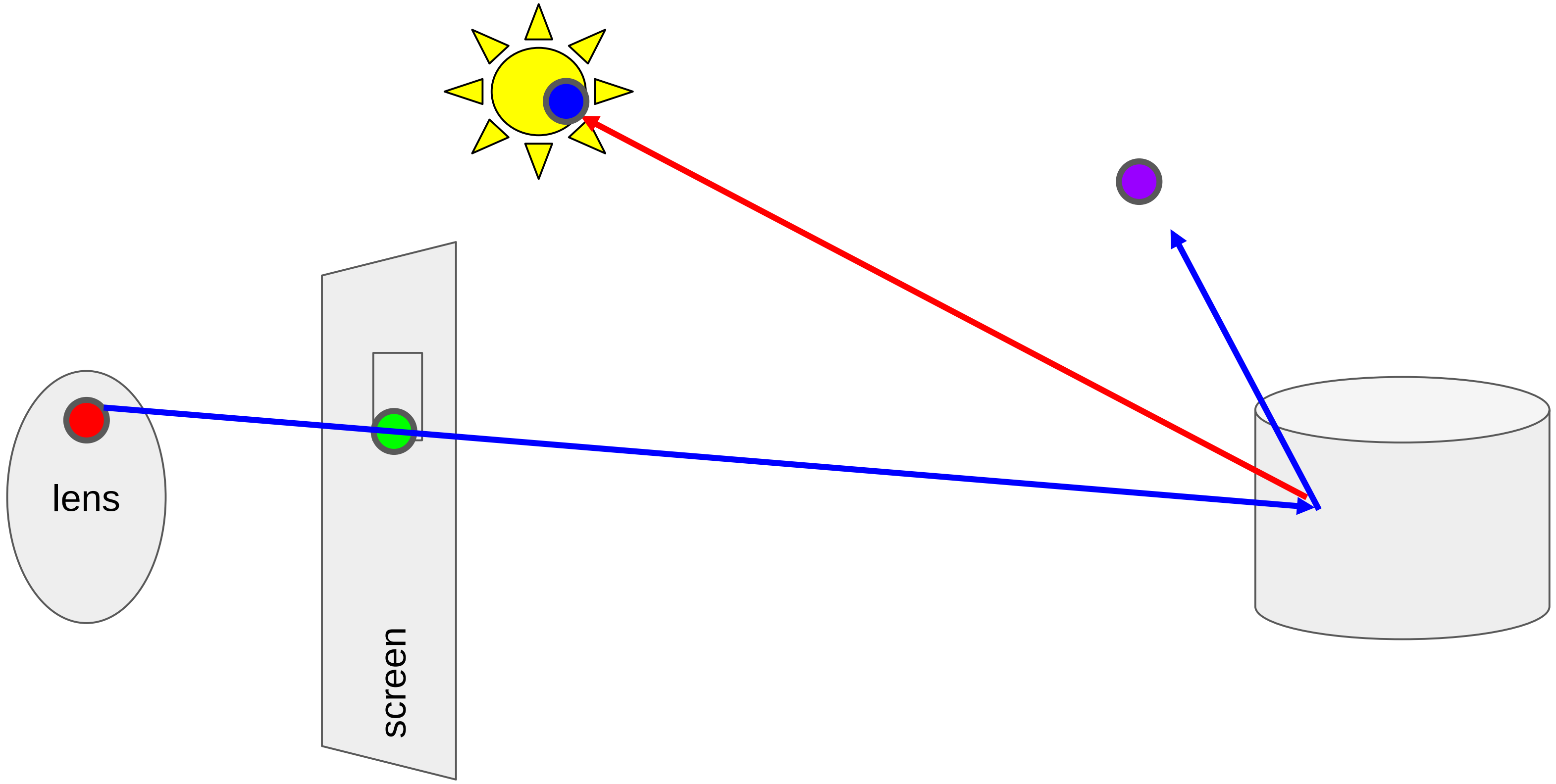


$$P = f(x,y)A + g(x,y)*B + (1-f(x,y)-g(x,y))*C$$

Uniform Points on Triangles






lens
screen

SUMMARY

RAYS ARE A DIRECTED LINE

QUERYING FOR HITS MAINLY:

1) DO TWO POINTS SEE EACH OTHER?
2) WHAT IS SEEN IN THAT DIRECTION?

FOR 2), YOU GET AUXILIARY INFO

COMPUTATION IS A WEIGHTED AVERAGED

DONT GET TOO UPTIGHT ABOUT PERFECT WEIGHTS YET-- BIAS IS NOT FATAL!

For debugging: brute force renderer

```
radiance(ray **r**)
    If **Q** = ray_hit(**r**)
        R = ray(**Q**, **N** + random_on_unit_sphere( )
        Return emitted + albedo*radiance(r)
    Else
        Return bg_color
```