

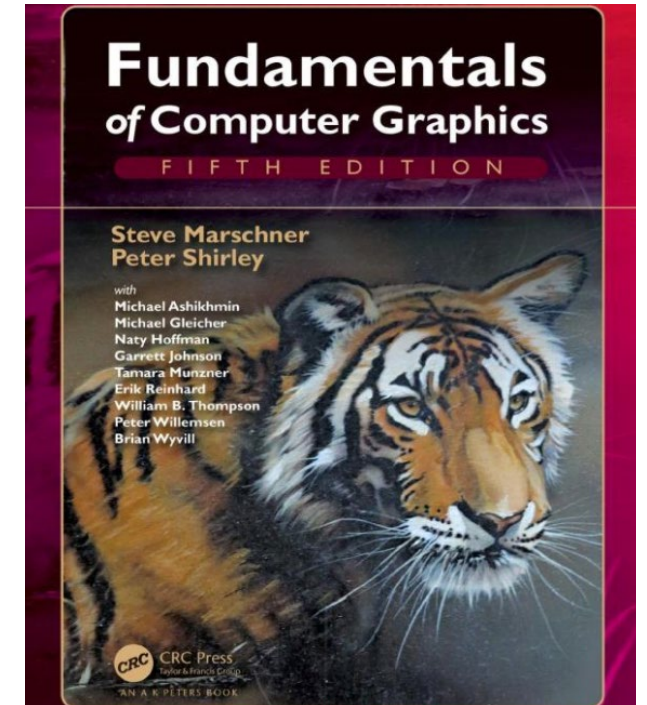


Introduction to Ray Tracing

CS 415: Game Development

Professor Eric Shaffer

Acknowledgments



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

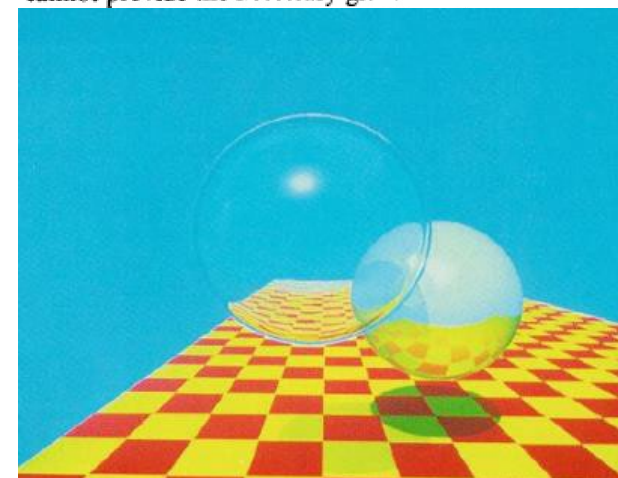
J.D. Foley
Editor

An Improved Illumination Model for Shaded Display

Turner Whitted
Bell Laboratories
Holmdel, New Jersey

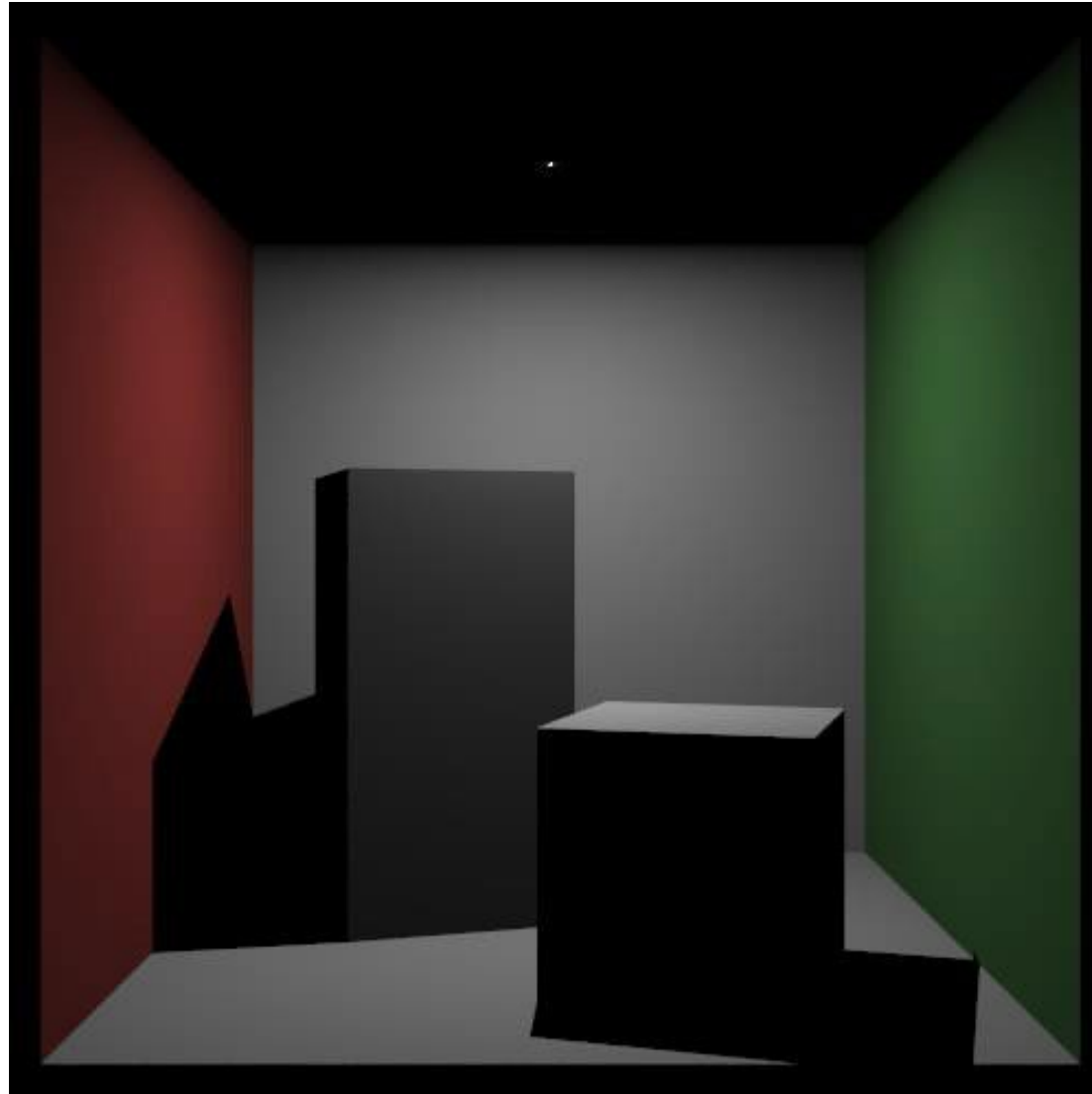
To accurately render a two-dimensional image of a three-dimensional scene, global illumination information that affects the intensity of each pixel of the image must be known at the time the intensity is calculated. In a simplified form, this information is stored in a tree of "rays" extending from the 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.

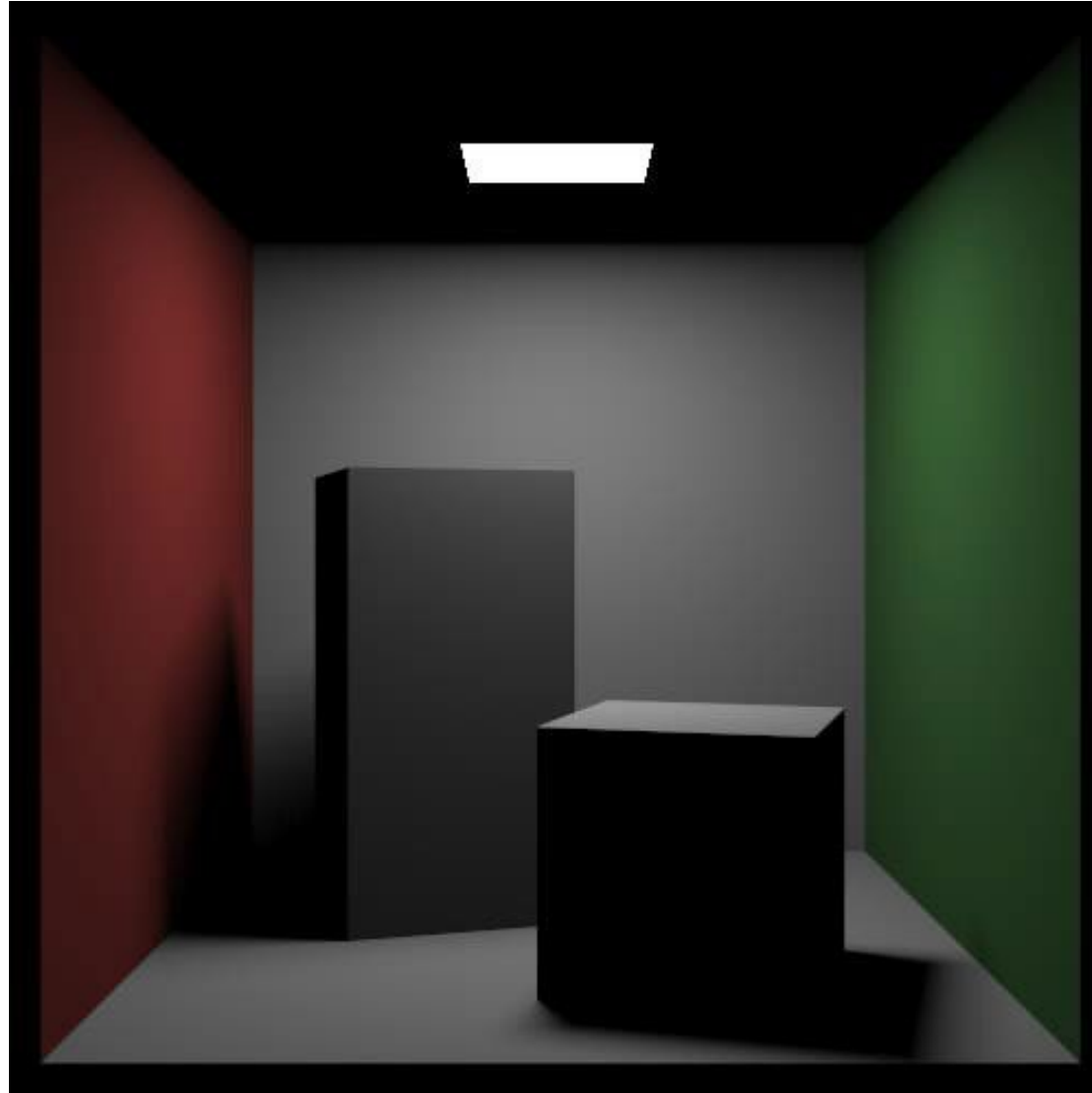


$$I = I_a + k_d \sum_{j=1}^{j=ls} (\vec{N} \cdot \vec{L}_j) + k_s \sum_{j=1}^{j=ls} (\vec{N} \cdot \vec{L}_j)^n, \quad (1)$$

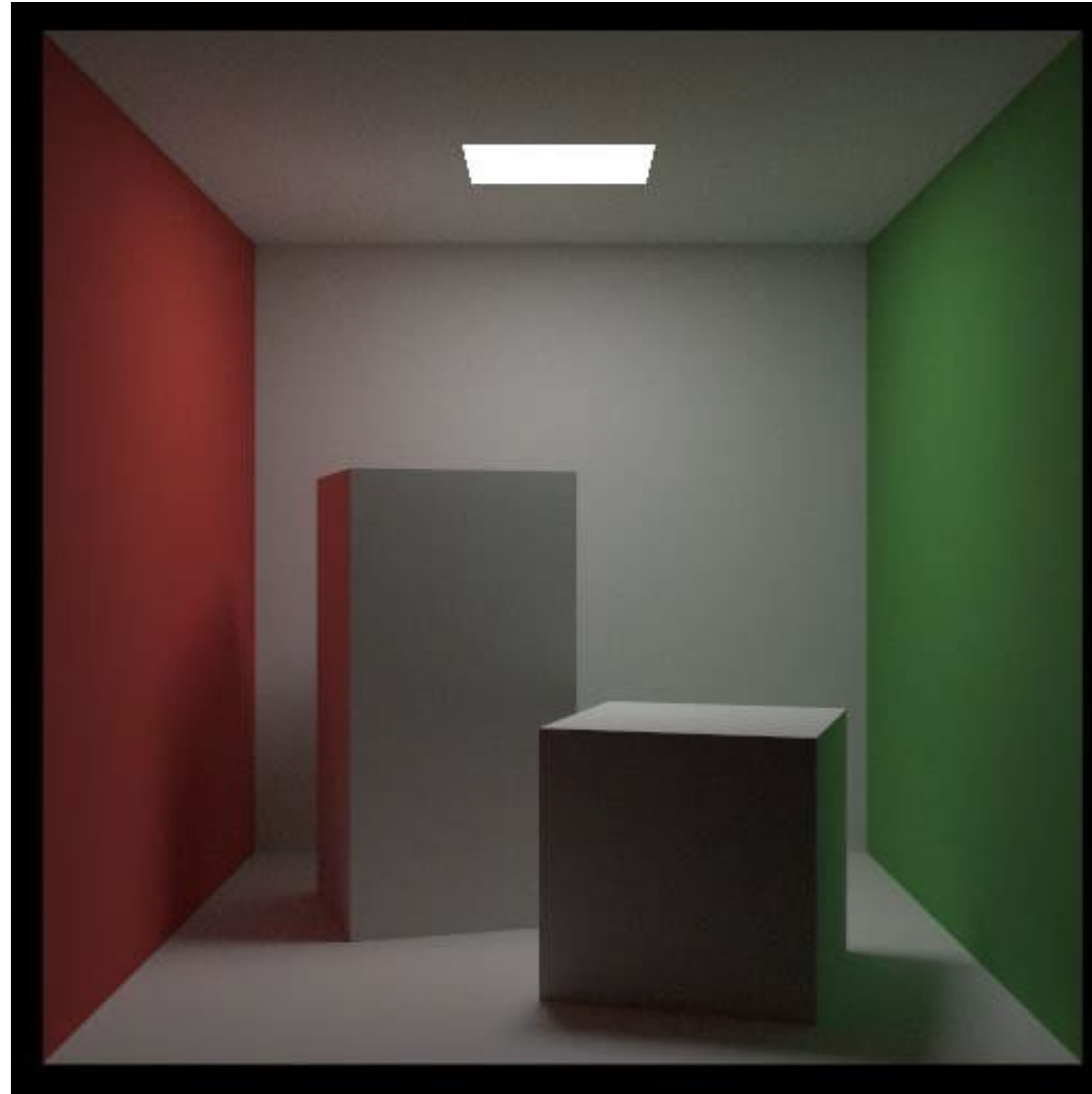
HARD SHADOWS



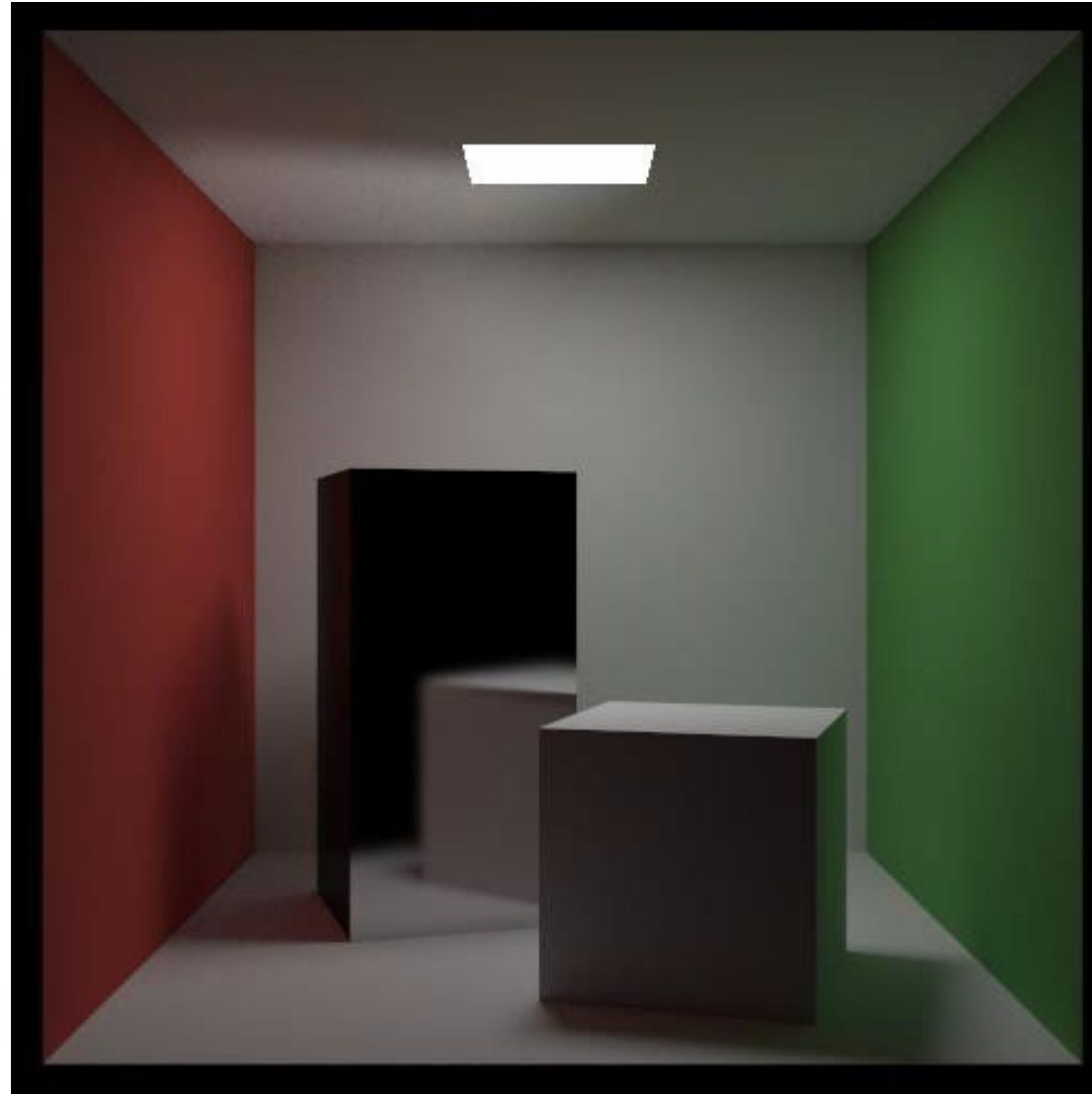
SOFT SHADOWS



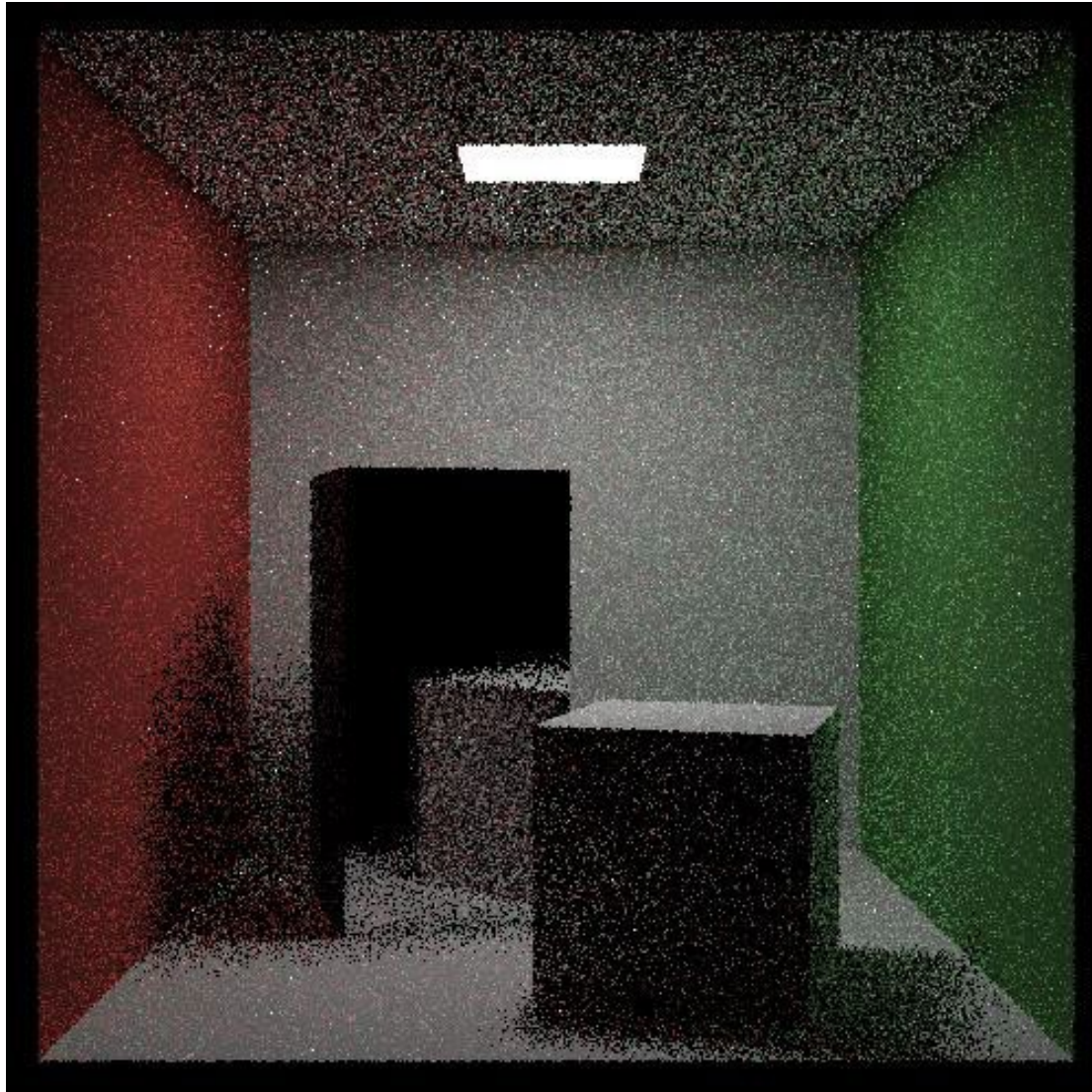
DIFFUSE INTER-REFLECTION



SPECULAR REFLECTION

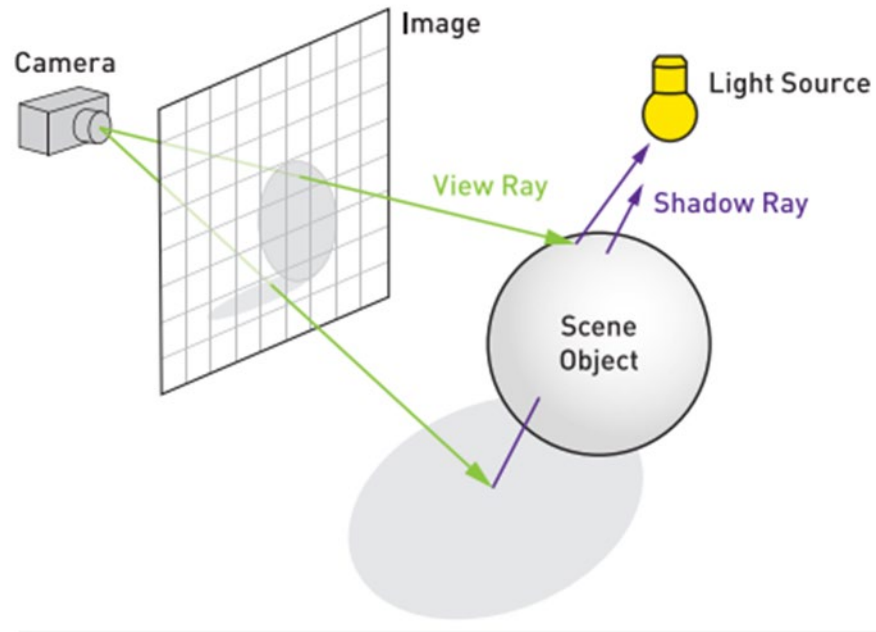


ONE SAMPLE PER PIXEL



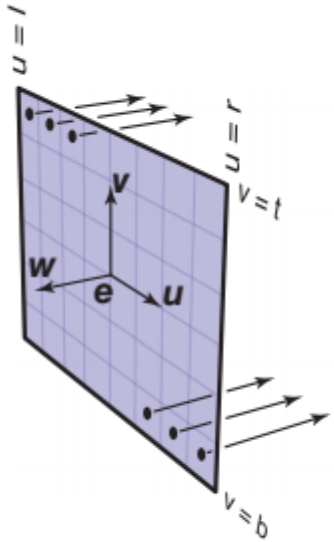
What does it
mean to take
one sample
per pixel?

Sampling = Casting a Light Ray into the Scene

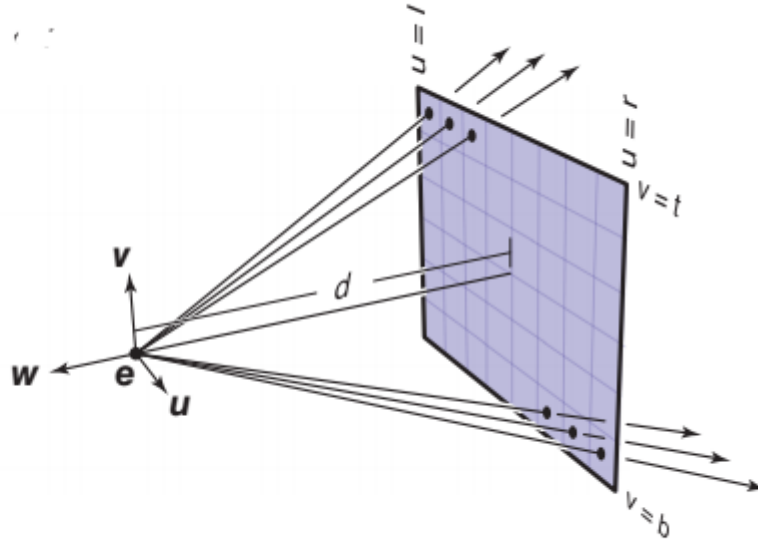


All these things...camera...lights...objects
...image plane...pixels...have locations and
orientations in a virtual world that we are
rendering

Sampling = Casting a Light Ray into the Scene

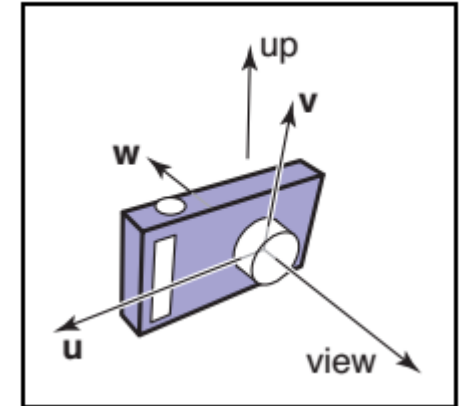


Parallel projection
same direction, different origins

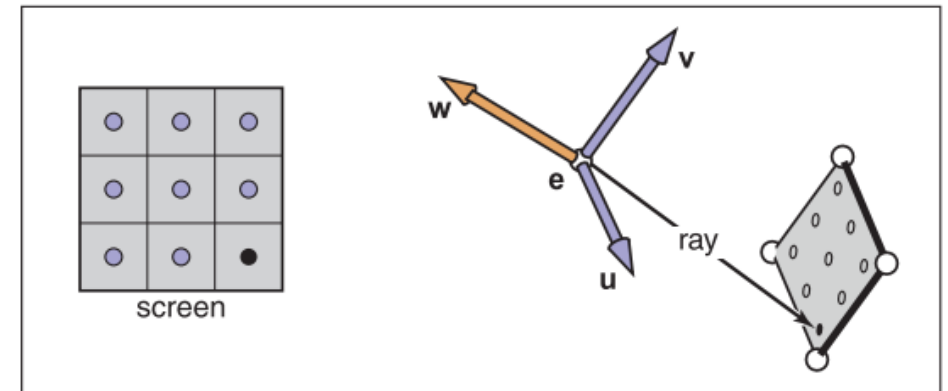


Perspective projection
same origin, different directions

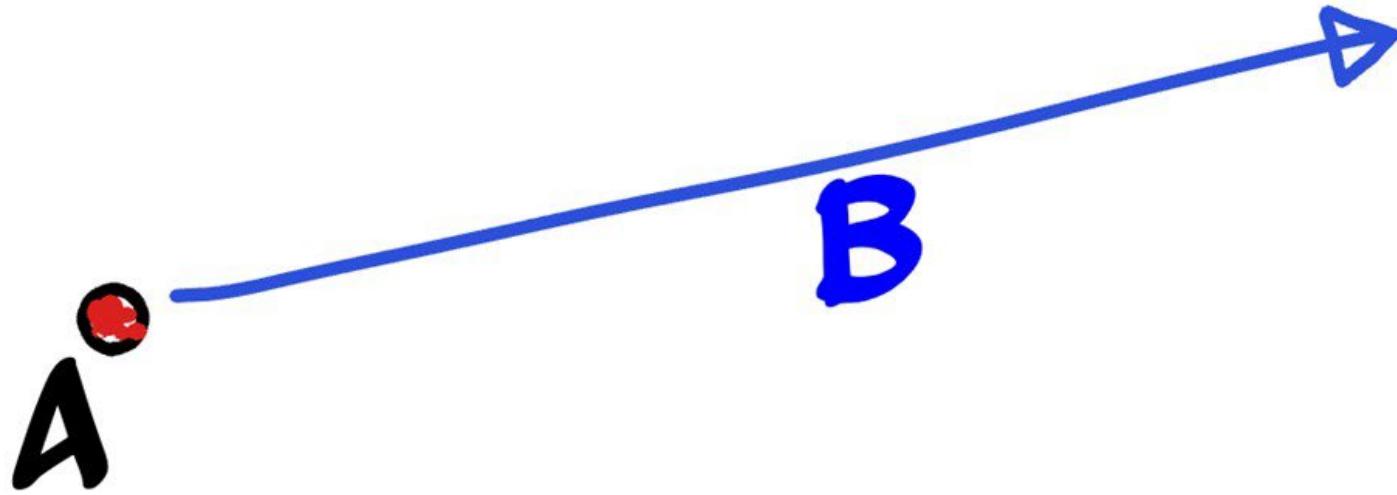
Camera has its own frame...allows you to do calculations from origin of that frame...simpler



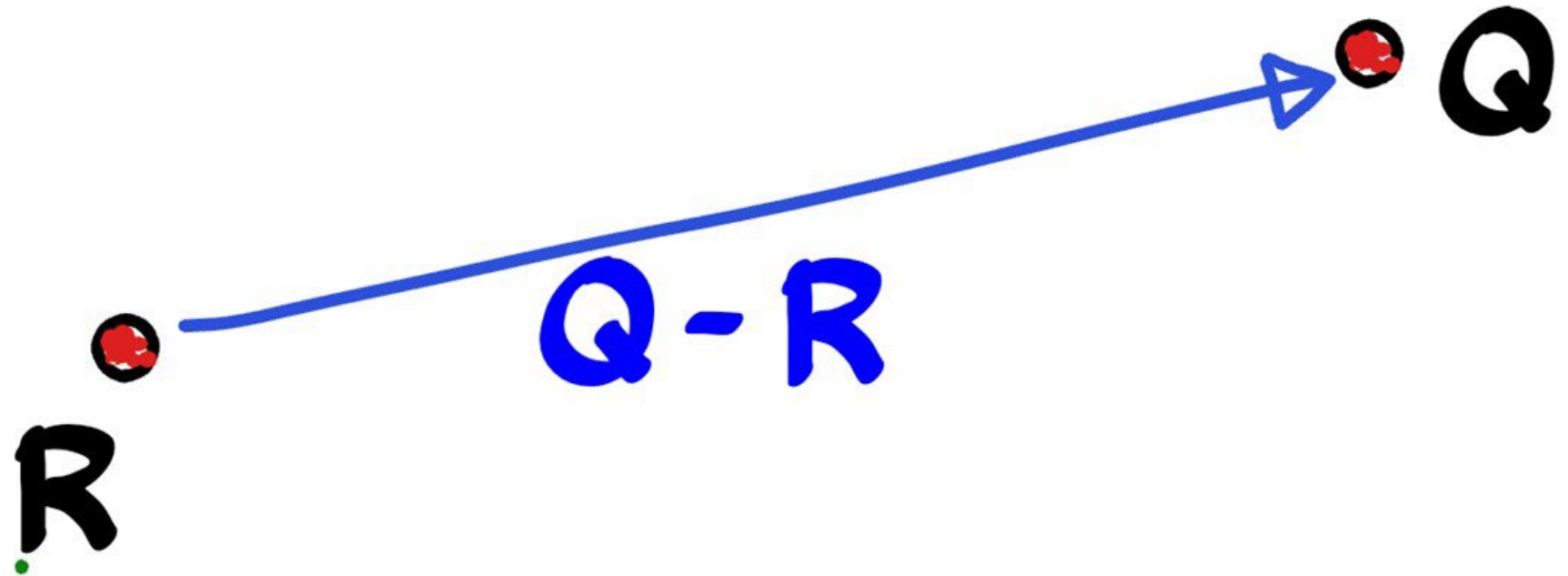
The sample points on the screen are mapped to a similar array on the 3D window. A viewing ray is sent to each of these locations.



A RAY IS A LOCATION AND A DIRECTION

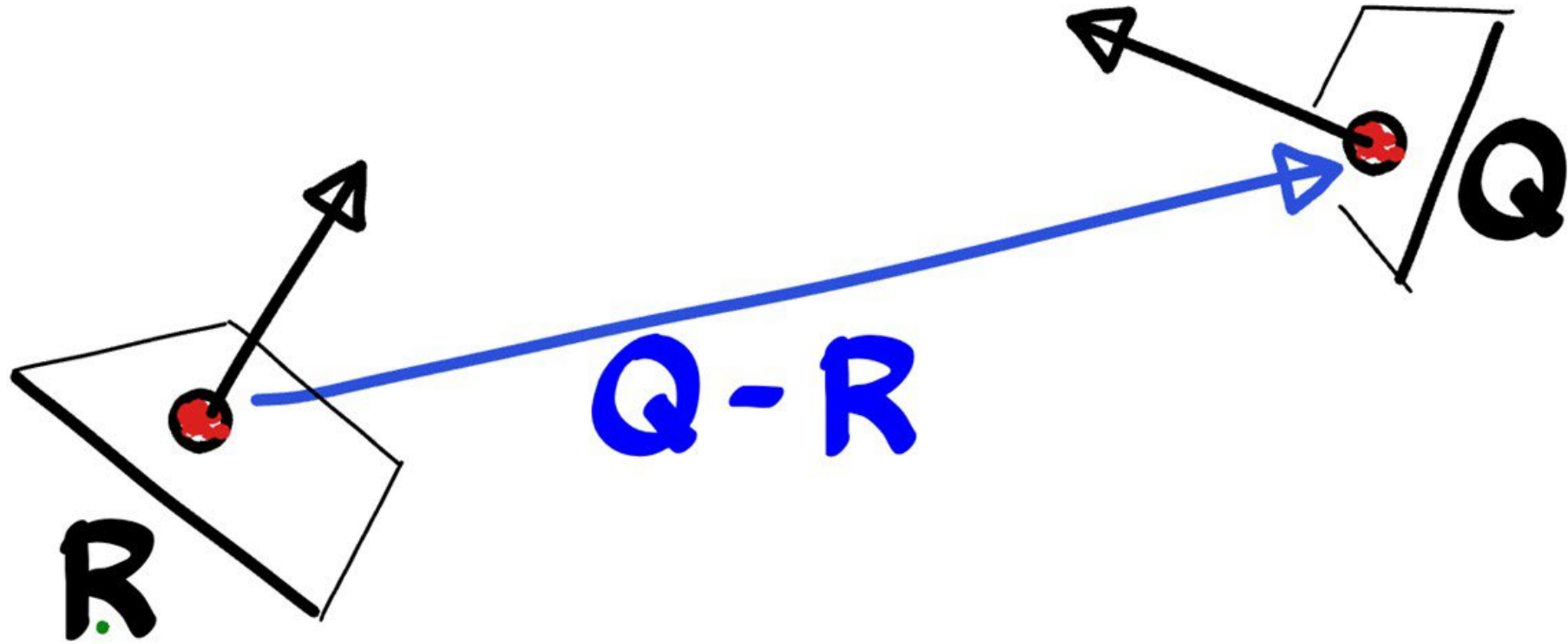


A RAY CAN TEST VISIBILITY
BETWEEN TWO POINTS



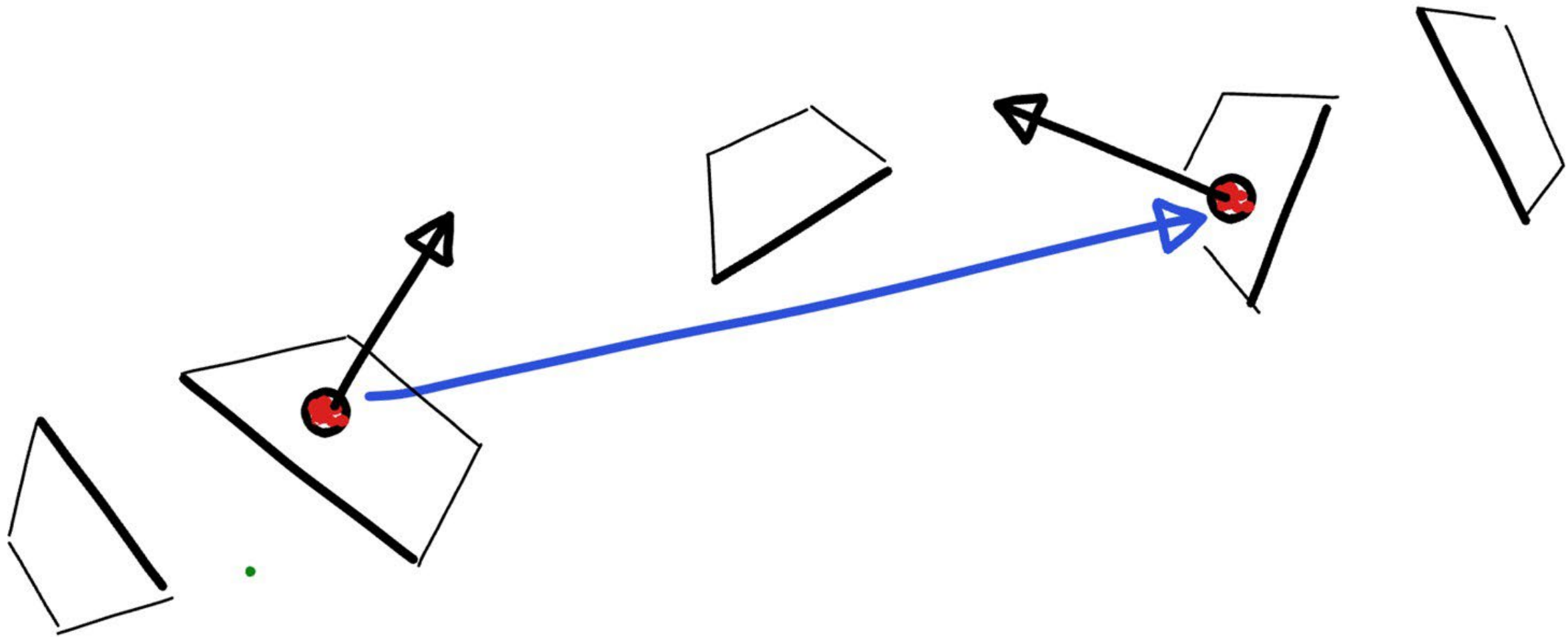
$$P(t) = R + t * (Q - R)$$

A RAY CAN TEST VISIBILITY
BETWEEN TWO POINTS ON SURFACES



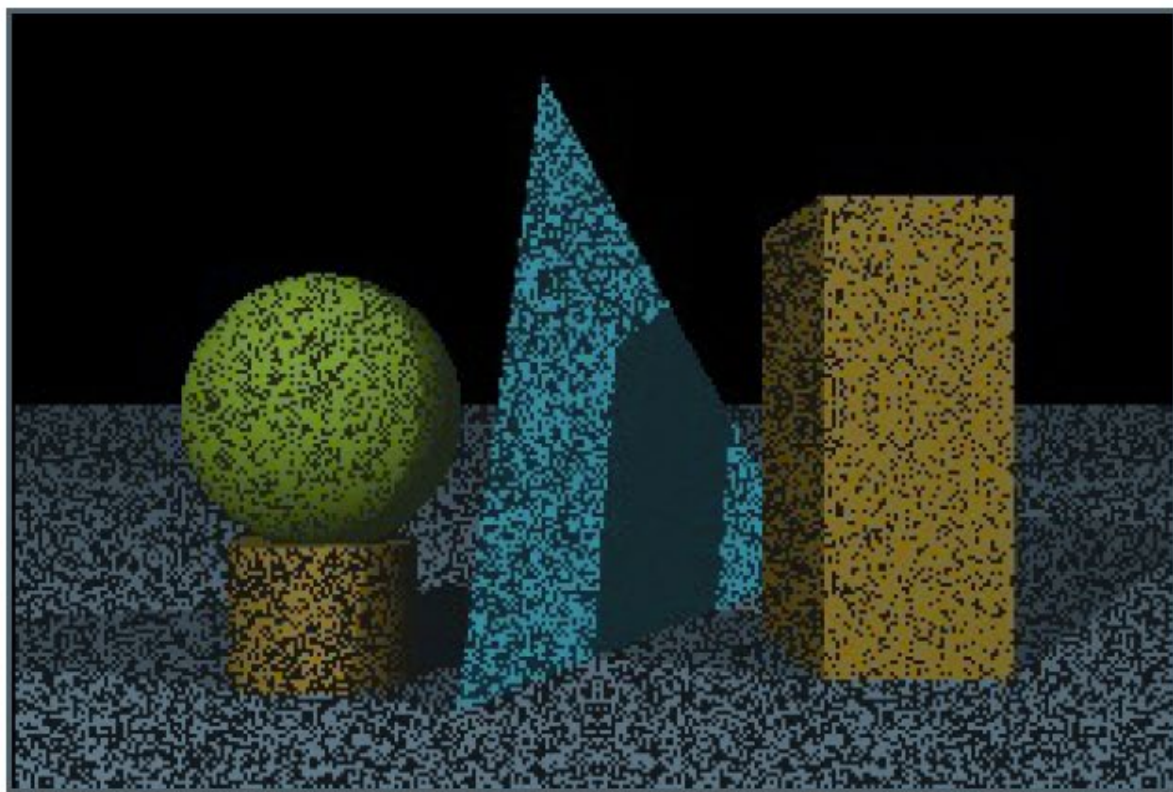
$$P(t) = R + t * (Q-R)$$

FLOATING POINT PAIN IN NECK

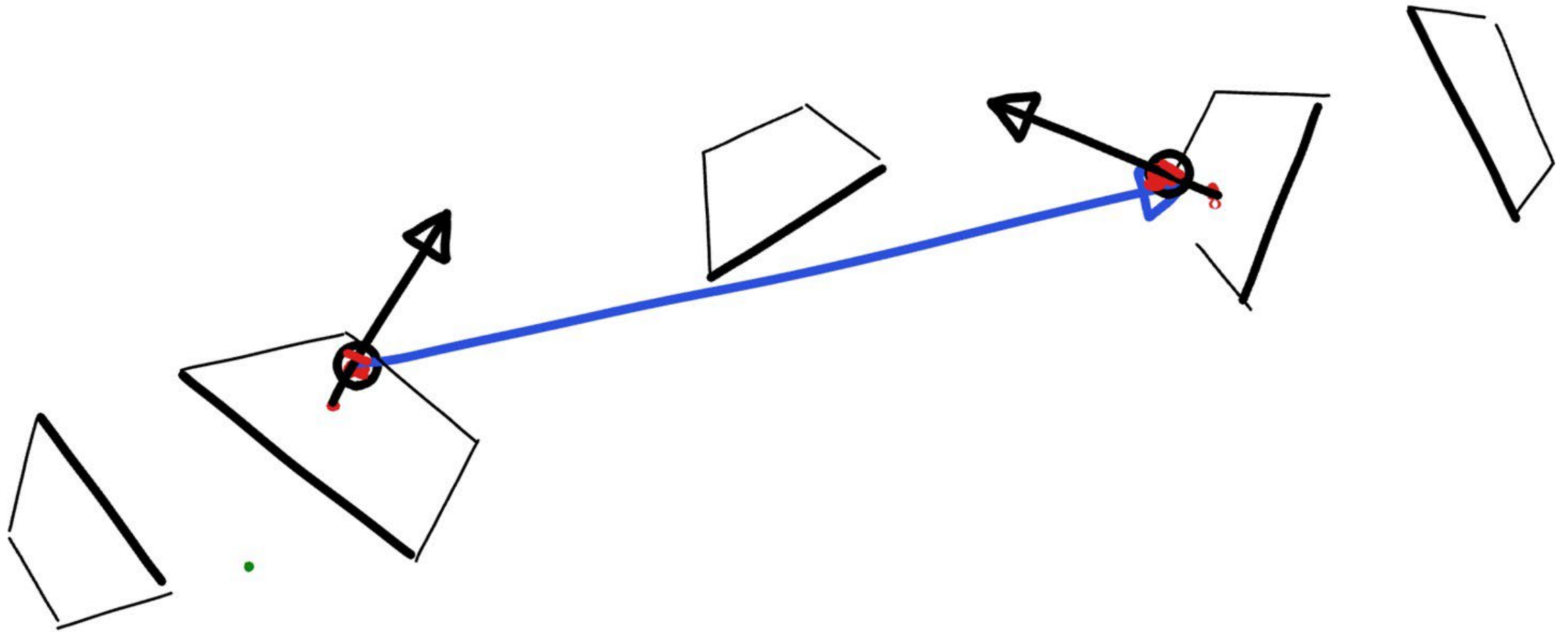


HITS AT $T = -0.25, 0, 1, 1.25$

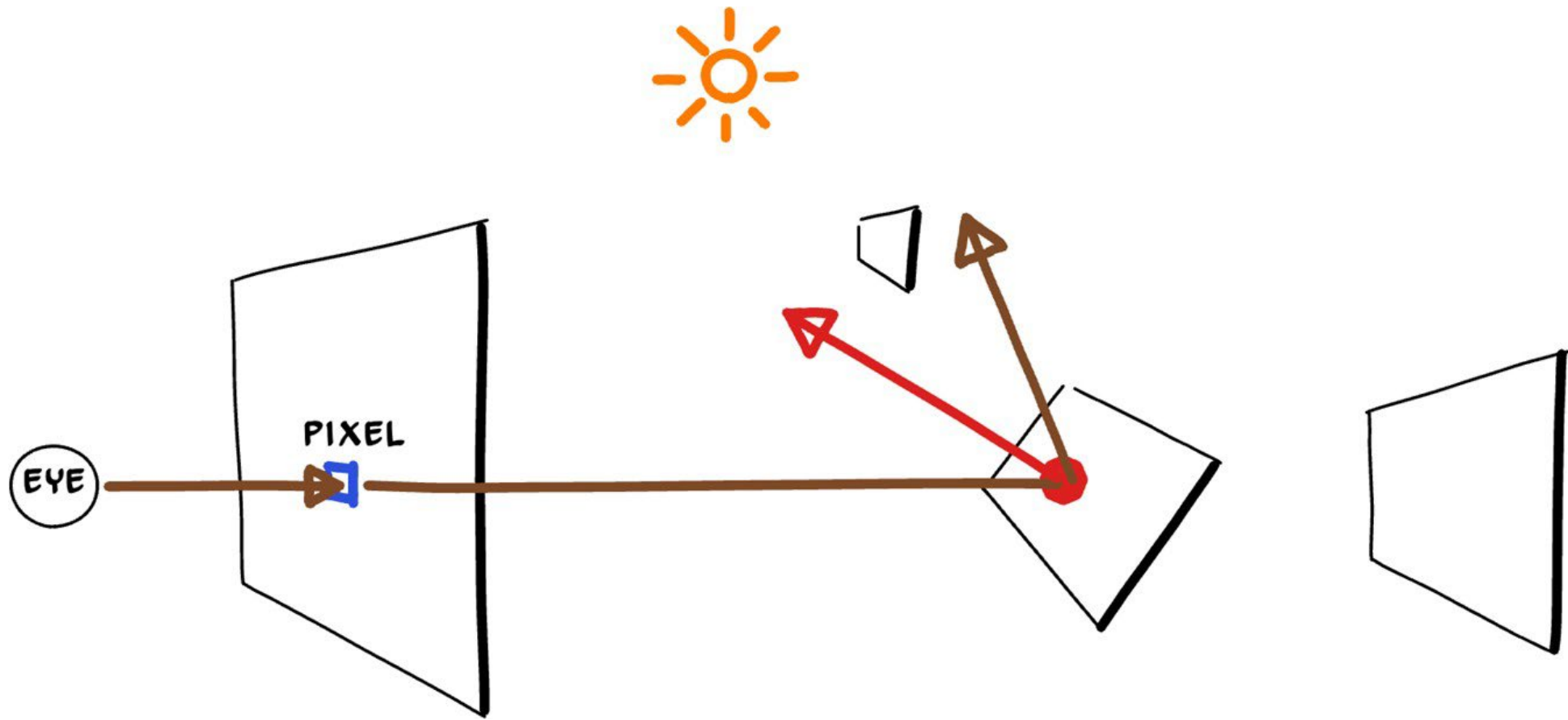
What Went Wrong?



FLOATING POINT PAIN IN NECK



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



```
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)
```


Ray-Object Intersection

4.4 Ray-Object Intersection

Once we've generated a ray $\mathbf{e} + t\mathbf{d}$, we next need to find the first intersection with any object where $t > 0$. In practice, it turns out to be useful to solve a slightly more general problem: find the first intersection between the ray and a surface that occurs at a t in the interval $[t_0, t_1]$. The basic ray intersection is then the case where $t_0 = 0$ and $t_1 = +\infty$. We solve this problem for both spheres and triangles. In the next section, multiple objects are discussed.

$$\begin{aligned}f(x, y, z) &= 0 \\f(p) &= 0 \\f(o + td) &= 0\end{aligned}$$

We can intersect an implicit surface with a ray

Just find the point p on the ray that satisfies the equation for the implicit surface

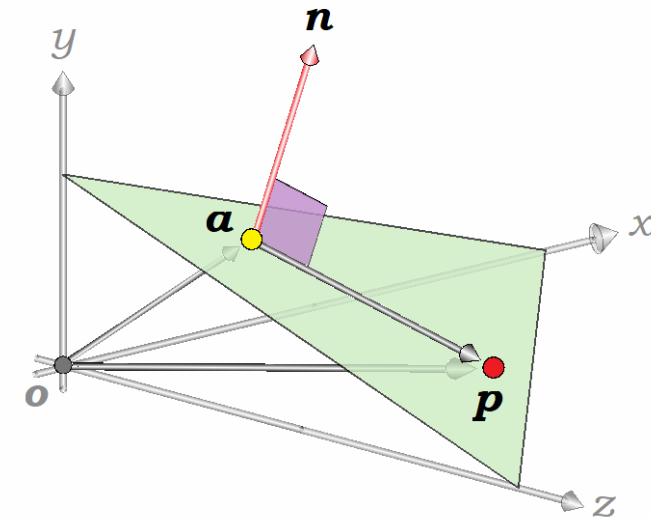
Plane Equation

$$Ax + By + Cz + D = 0$$

$$(p - a) \cdot n = 0$$

Here a is a point on the plane and n is the normal

All points p that satisfy the equation form the plane



Ray-Plane Intersection

Use the plane equation with normal \mathbf{n} and point on the plane \mathbf{a}
Solve for t ...that value generates a point on both the plane and ray

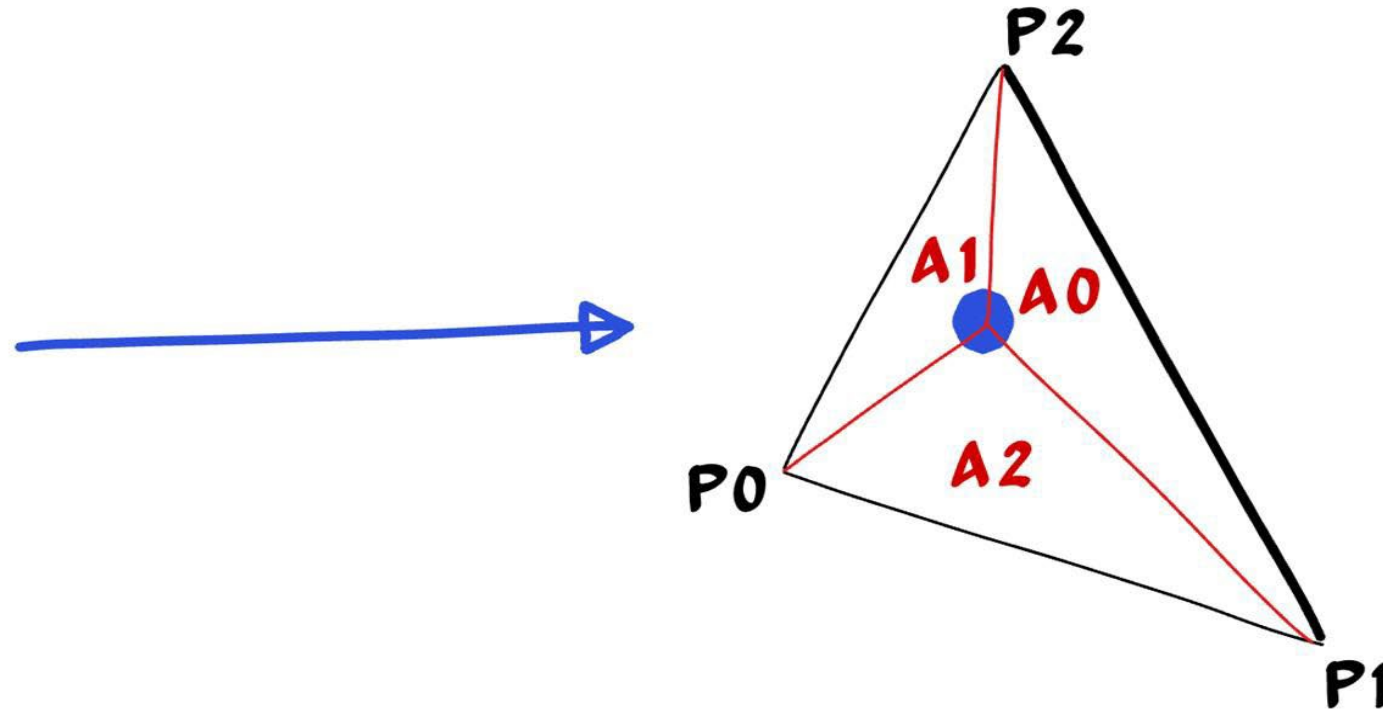
$$\begin{aligned}(\mathbf{p} - \mathbf{a}) \cdot \mathbf{n} &= 0 \\(\mathbf{o} + t\mathbf{d} - \mathbf{a}) \cdot \mathbf{n} &= 0 \\t &= ((\mathbf{a} - \mathbf{o}) \cdot \mathbf{n}) / (\mathbf{d} \cdot \mathbf{n})\end{aligned}$$

What happens if \mathbf{d} is parallel to the plane?

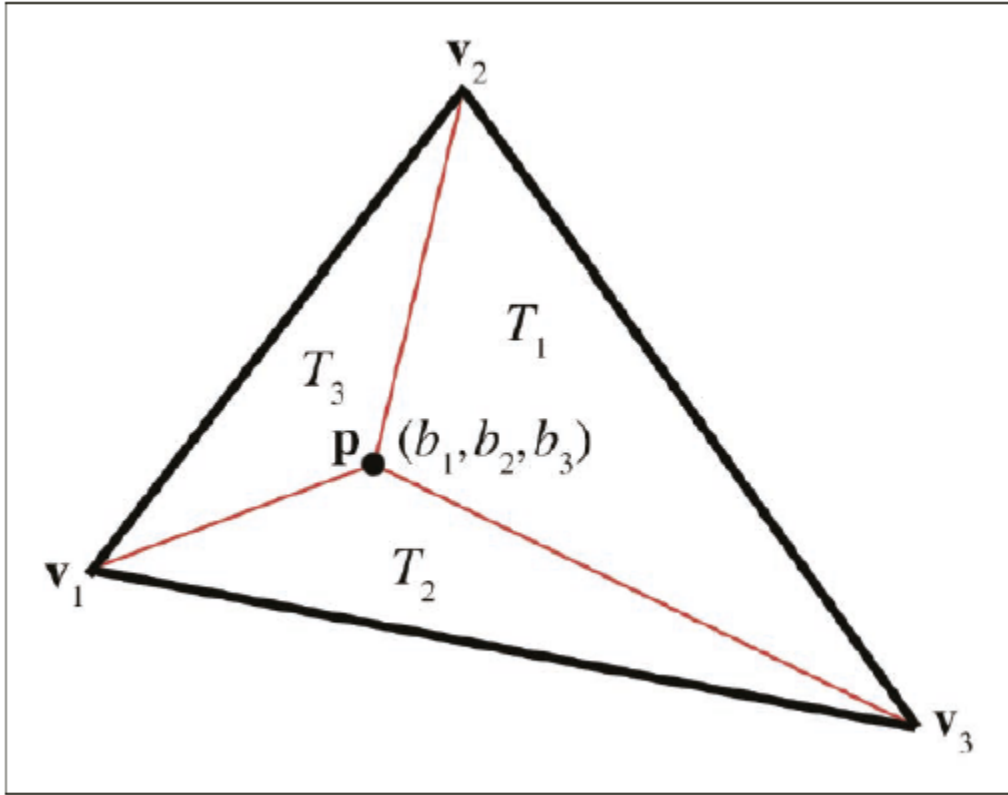
How do you know if the hit happens in front or behind the viewer?

So..How About Triangles?

Anyone know one way of extending ray-plane for triangles?



Computing Barycentric Coordinates for Triangles

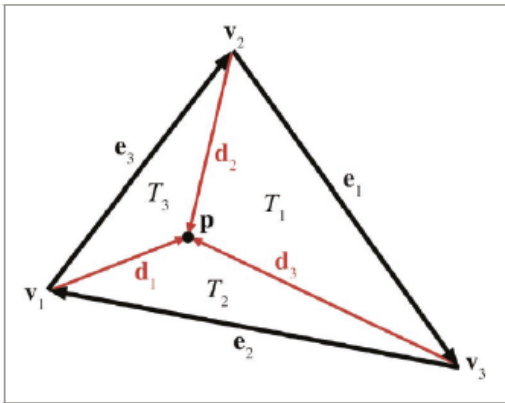


Coordinates are the signed area of the opposite subtriangle divided by area of the triangle

$$b_1 = A(T_1)/A(T), \quad b_2 = A(T_2)/A(T), \quad b_3 = A(T_3)/A(T)$$

Can Compute in 3D....

Don't worry about this math...



$$\begin{aligned} \mathbf{e}_1 &= \mathbf{v}_3 - \mathbf{v}_2, \\ \mathbf{d}_1 &= \mathbf{p} - \mathbf{v}_1, \end{aligned}$$

$$\begin{aligned} \mathbf{e}_2 &= \mathbf{v}_1 - \mathbf{v}_3, \\ \mathbf{d}_2 &= \mathbf{p} - \mathbf{v}_2, \end{aligned}$$

$$\begin{aligned} \mathbf{e}_3 &= \mathbf{v}_2 - \mathbf{v}_1, \\ \mathbf{d}_3 &= \mathbf{p} - \mathbf{v}_3. \end{aligned}$$

$$\hat{\mathbf{n}} = \frac{\mathbf{e}_1 \times \mathbf{e}_2}{\|\mathbf{e}_1 \times \mathbf{e}_2\|}.$$

$$A(T) = ((\mathbf{e}_1 \times \mathbf{e}_2) \cdot \hat{\mathbf{n}})/2,$$

$$A(T_1) = ((\mathbf{e}_1 \times \mathbf{d}_3) \cdot \hat{\mathbf{n}})/2,$$

$$A(T_2) = ((\mathbf{e}_2 \times \mathbf{d}_1) \cdot \hat{\mathbf{n}})/2,$$

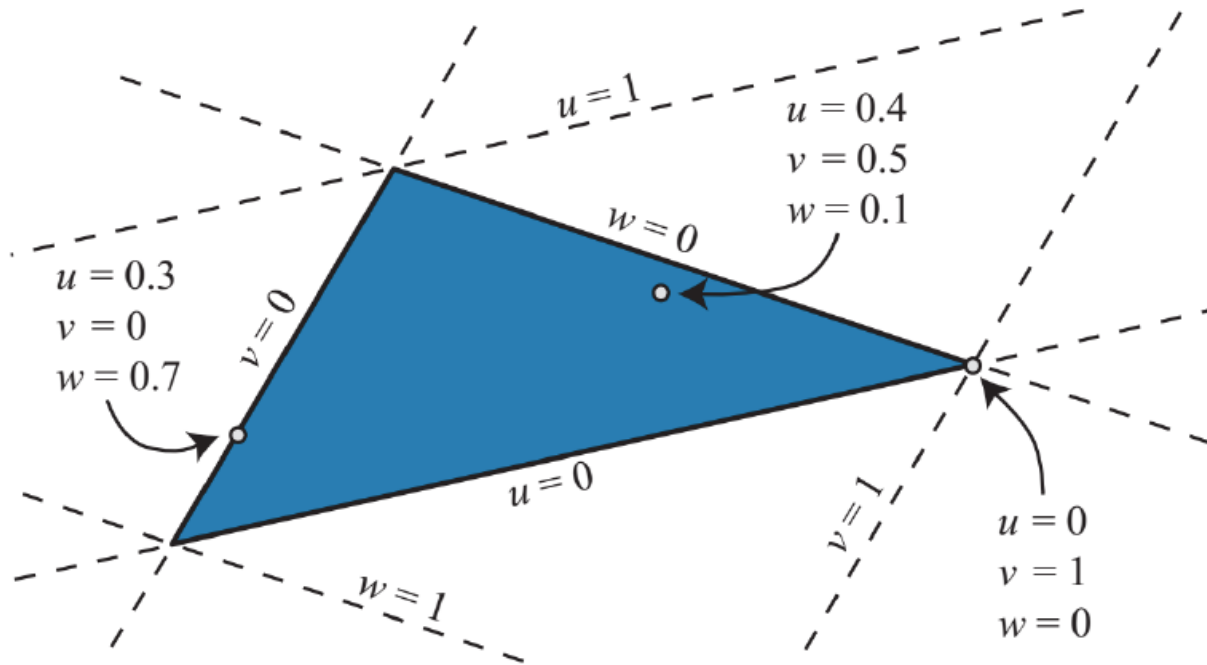
$$A(T_3) = ((\mathbf{e}_3 \times \mathbf{d}_2) \cdot \hat{\mathbf{n}})/2.$$

$$b_1 = A(T_1)/A(T) = \frac{(\mathbf{e}_1 \times \mathbf{d}_3) \cdot \hat{\mathbf{n}}}{(\mathbf{e}_1 \times \mathbf{e}_2) \cdot \hat{\mathbf{n}}},$$

$$b_2 = A(T_2)/A(T) = \frac{(\mathbf{e}_2 \times \mathbf{d}_1) \cdot \hat{\mathbf{n}}}{(\mathbf{e}_1 \times \mathbf{e}_2) \cdot \hat{\mathbf{n}}},$$

$$b_3 = A(T_3)/A(T) = \frac{(\mathbf{e}_3 \times \mathbf{d}_2) \cdot \hat{\mathbf{n}}}{(\mathbf{e}_1 \times \mathbf{e}_2) \cdot \hat{\mathbf{n}}}.$$

Point in Triangle Test



How can we test if a point is in a triangle?

Anti-Aliasing

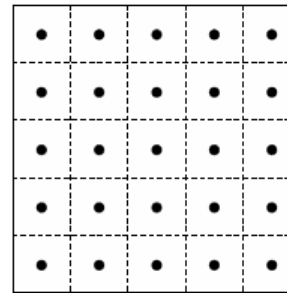
Often need multiple samples per pixel avoid jagged appearance



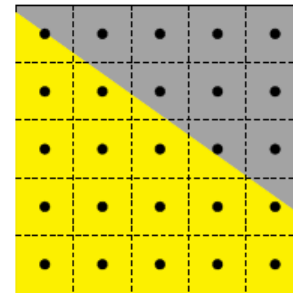
Filthy Jagged
Original
Emulation

Glorious Anti-
Aliased PC
Emulation

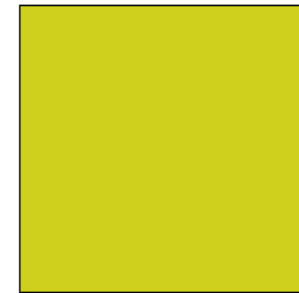
One remedy to aliasing is to shoot more rays per-pixel



Sampling pattern
on a single pixel



Pixel on the edge



Final color

Uniform sub-grid is
not ideal...lots of
different sampling
have been proposed
and used

The Rendering Equation

directions for making pictures using numbers
(explained using only the ten hundred words people use most often)

the light that comes from an interesting direction towards the position on the stuff

direction towards the eye

position on the stuff

light made by the stuff (sometimes because it is very hot)

the answer to how much light from an interesting direction that will keep going in the direction towards the eye, after hitting stuff at the position (this is easy for mirrors, not so easy for everything else)

stuff

how much the light becomes less bright because the stuff leans away from the interesting direction

light that leaves the position on the stuff and reaches the eye

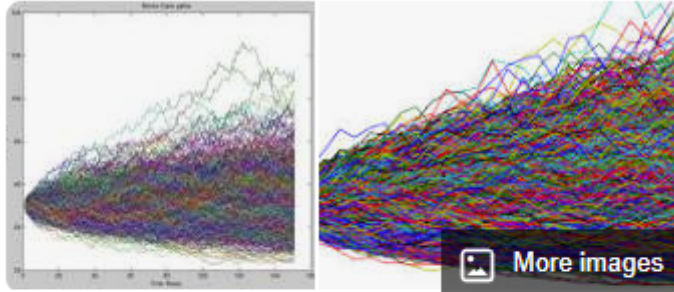
light can be added said a man who sat under a tree many years ago

for lots of interesting directions inside half a ball facing up from the stuff, add up all the answers in between

$$L_o(x, \omega_o) = L_e(x, \omega_o) + \int_{\Omega} f_r(x, \omega_i \rightarrow \omega_o) L_i(x, \omega_i) (\omega_i \cdot n) d\omega_i$$

A ray tracer is actually solving this equation

Monte Carlo Ray Tracing

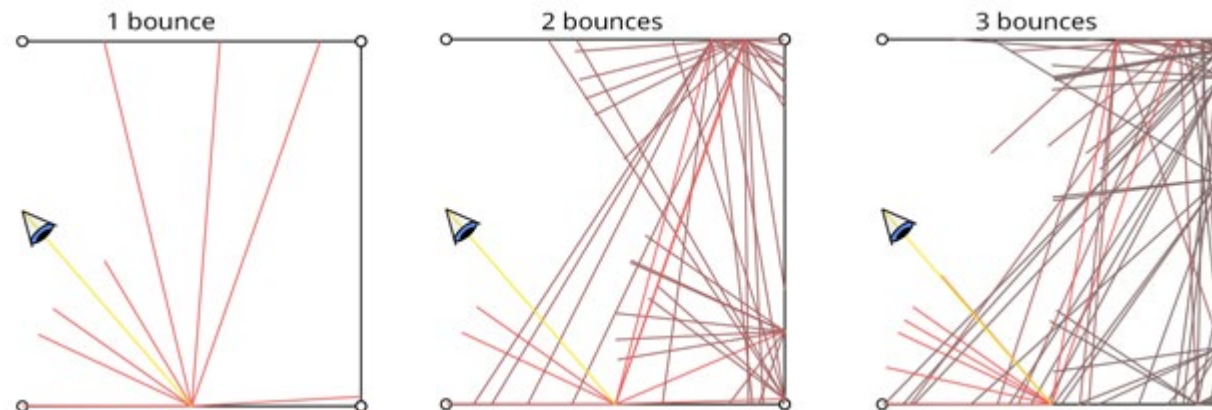


About

Monte Carlo methods, or Monte Carlo experiments, are a broad class of computational algorithms that rely on repeated random sampling to obtain numerical results. The underlying concept is to use randomness to solve problems that might be deterministic in principle. [Wikipedia](#)

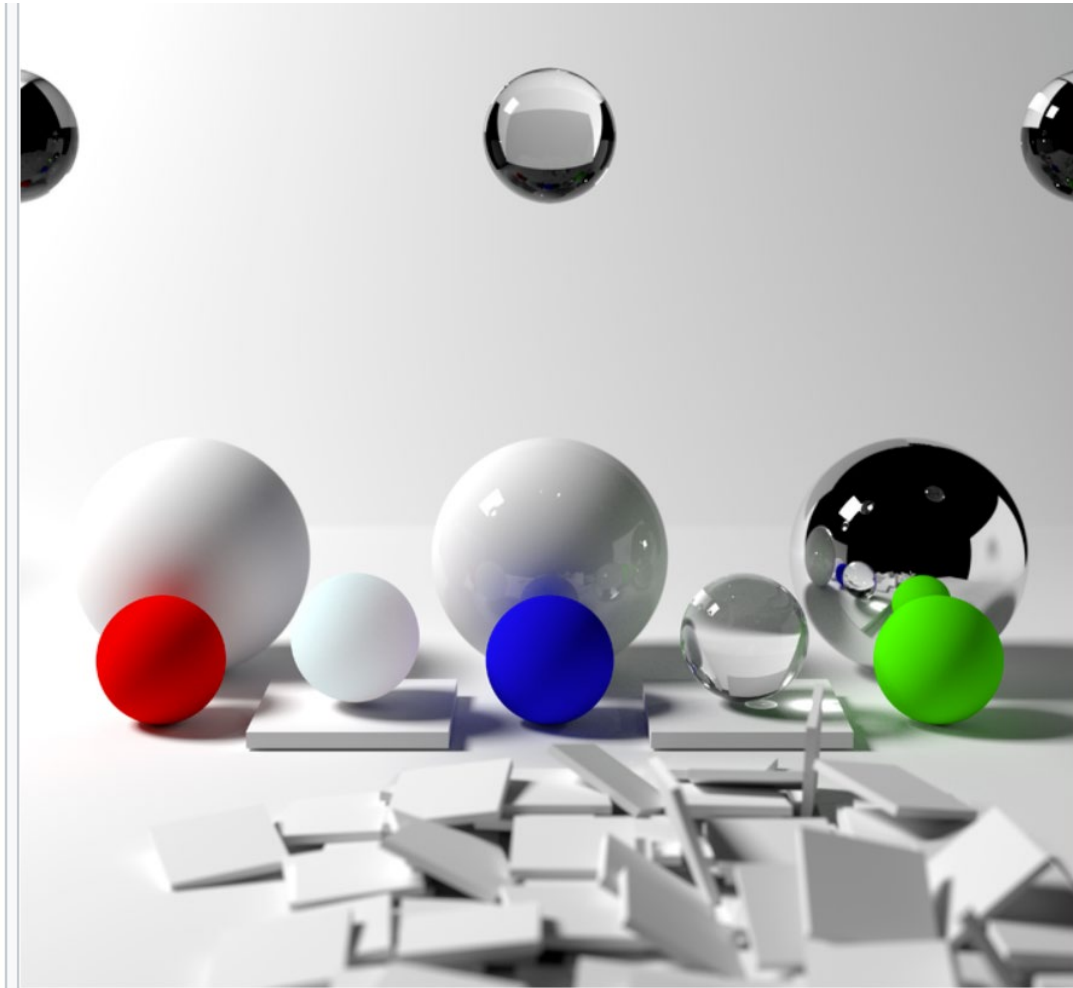
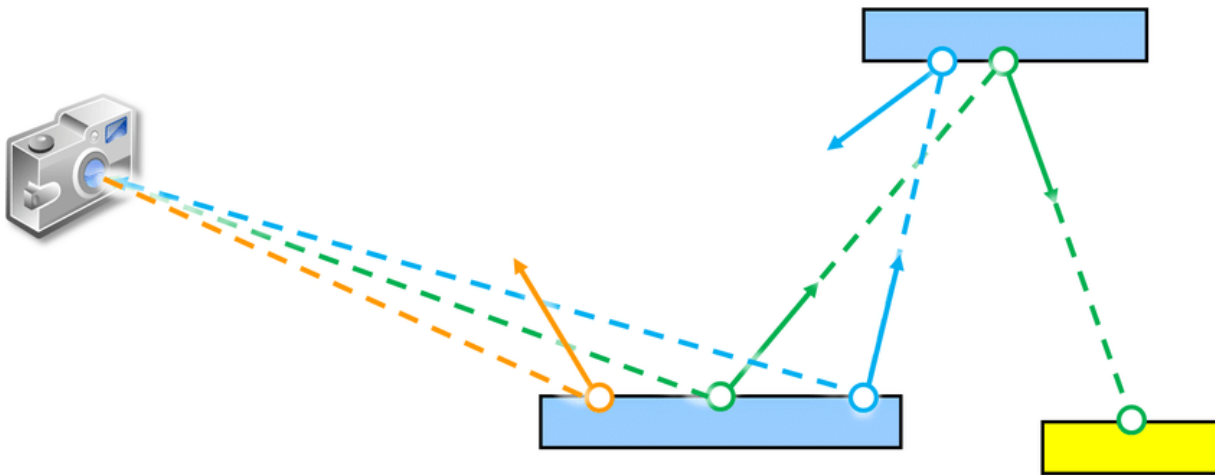
Approximate the solution to the integral using random sampling

Generating samples in a principled way allows faster convergence and correct(ish) results



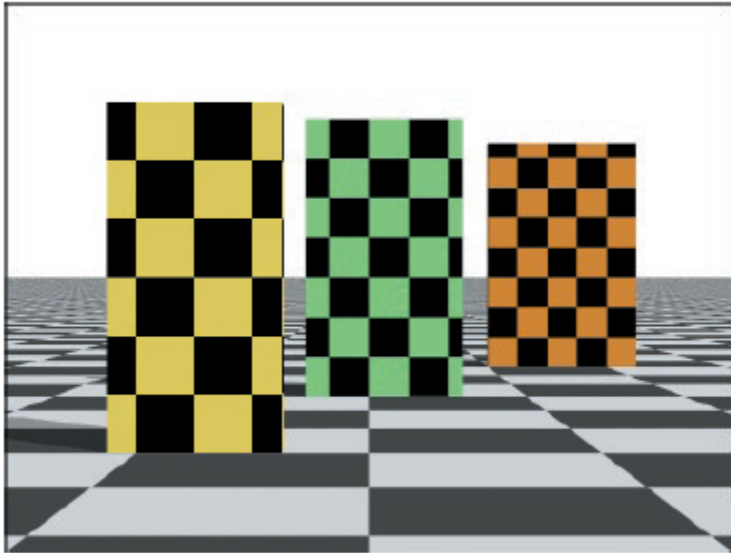
Path Tracing

Path tracing is a type of ray tracing. When using path tracing for rendering, the rays only produce a single ray per bounce.

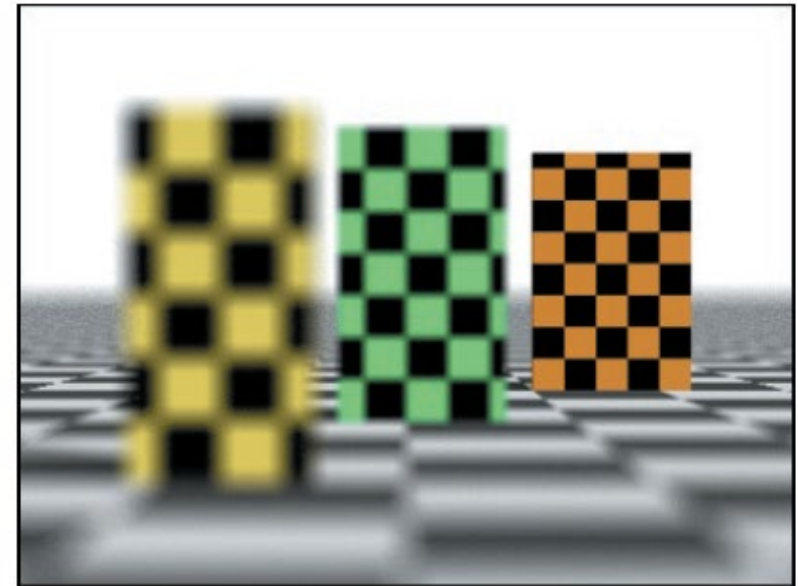


An image rendered using path tracing, demonstrating notable features of the technique

More Rays = More Effects



Default Rendering




Depth of Field

Can also do motion blur....

Ambient Occlusion

Ambient Occlusion NVIDIA

- Gives perceptual clues of depth, curvature and spatial proximity

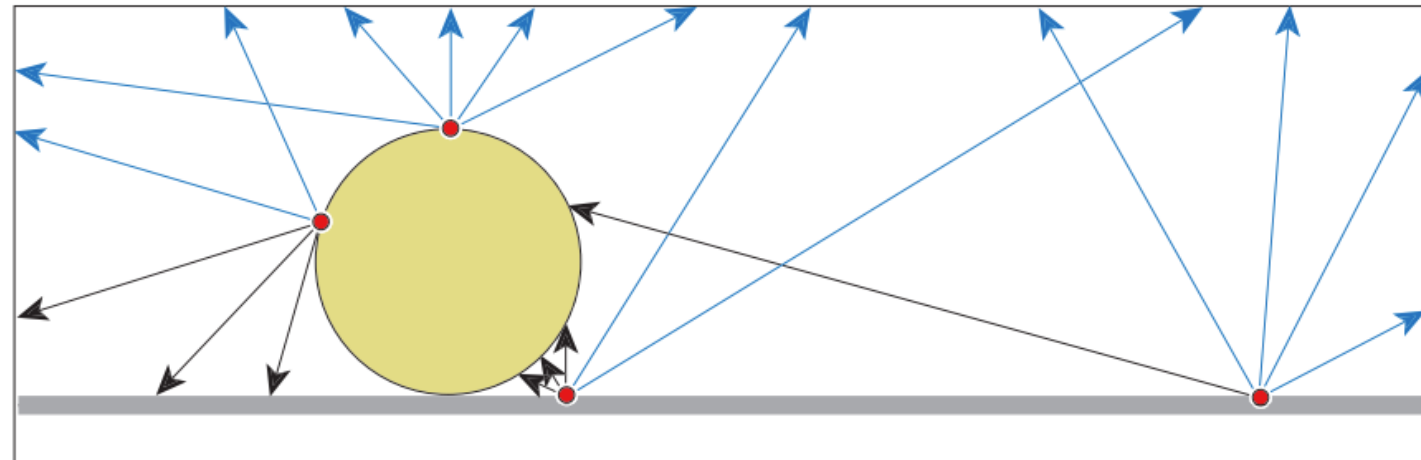
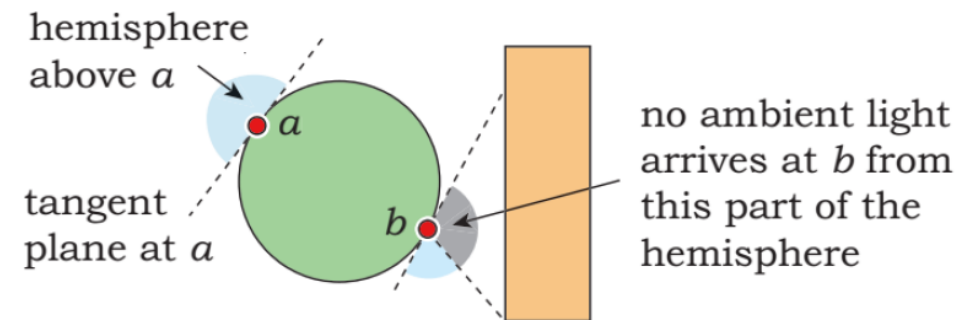


Without AO With our AO

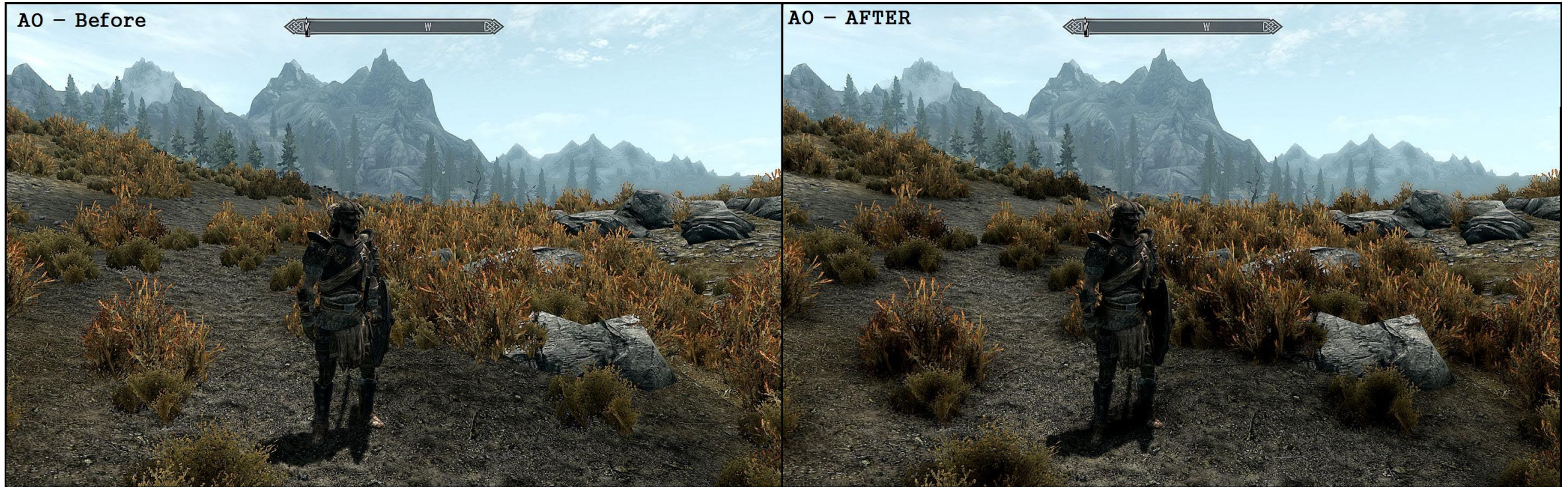
SIGGRAPH2008

We make ambient illumination non-constant

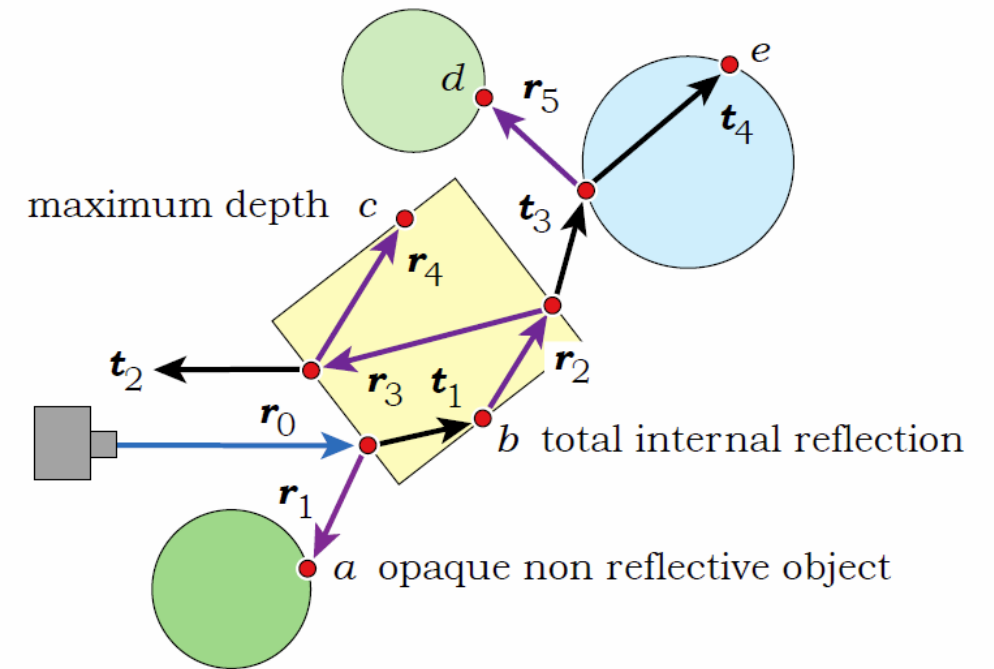
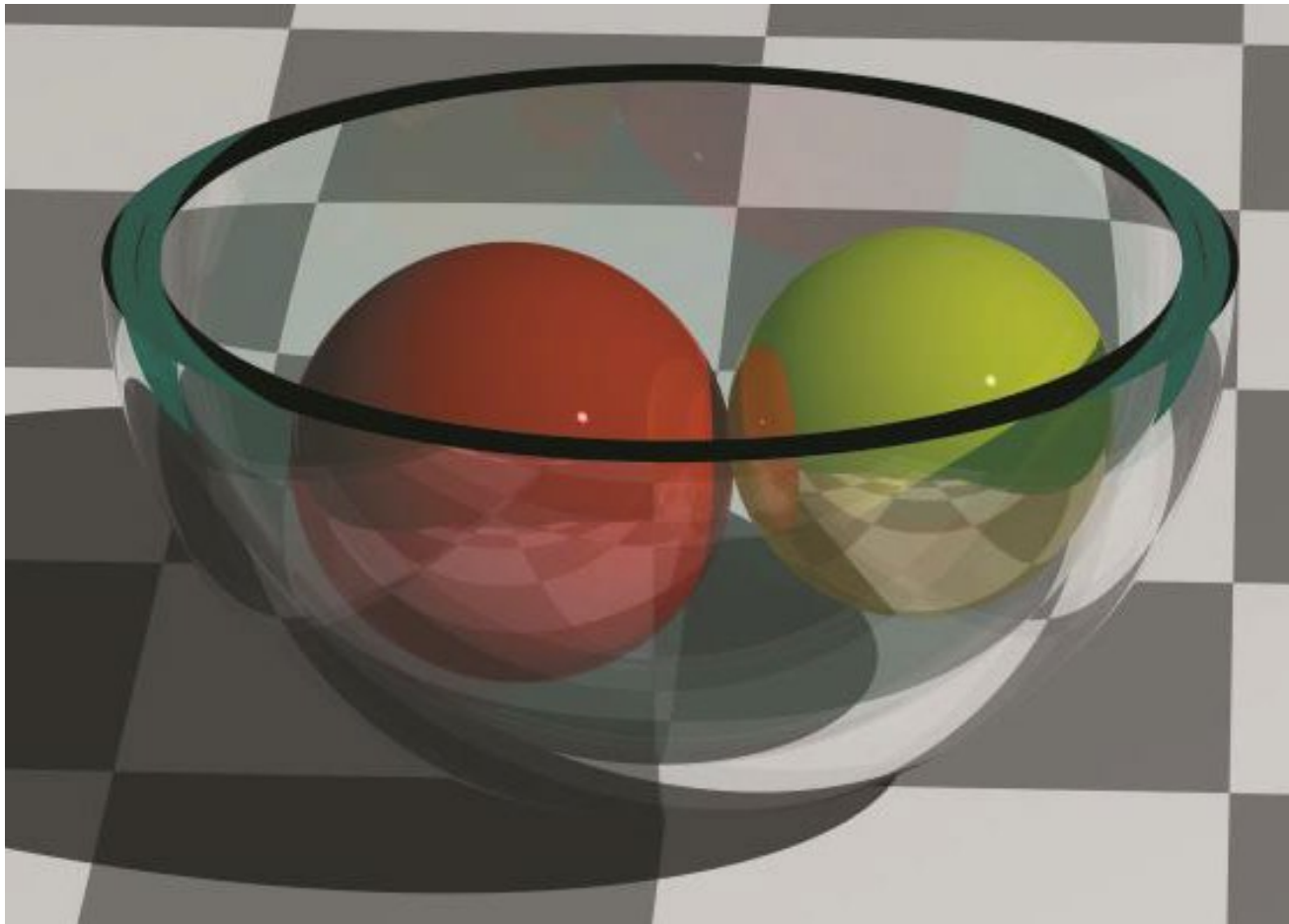
We base the illumination how much of hemisphere is not blocked



Ambient Occlusion



Transparency



Hybrid Rendering



Real-time ray tracing is still computationally intense...

Often just used for specific effects in games (e.g. reflections).

How can you combine ray-tracing and rasterization? Ideas?