maths and algorithms behind photo-realistic graphics

dr. jon denning
assistant professor of cse
taylor university

frank s. brenneman lecture series

warning: there will be equations

do not be afraid of them

goal of photo-realism: produce a computer-generated image that is indistinguishable from an actual photograph

[bertil pinewood chair, 2006, ikea link]


Mikhalenko_Art 2015


VALKYRIE

[valkyrie link]


VALKYRIE

[frank tzeng link]

[luc bégin link]

[jungwon park link]

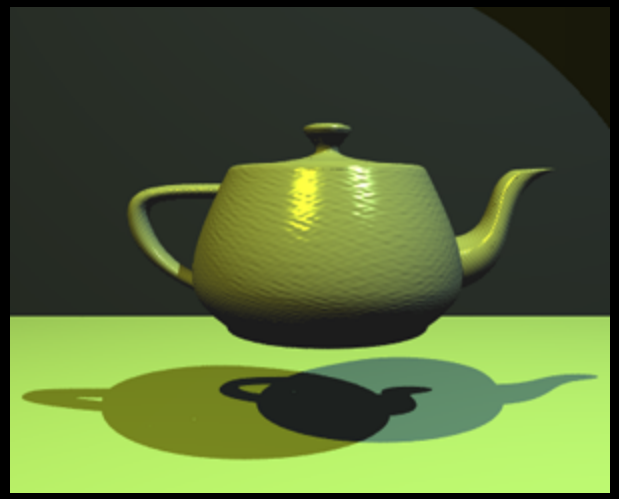
[chris jones link]

[andrew price link]

[youyudetun, link]

the good

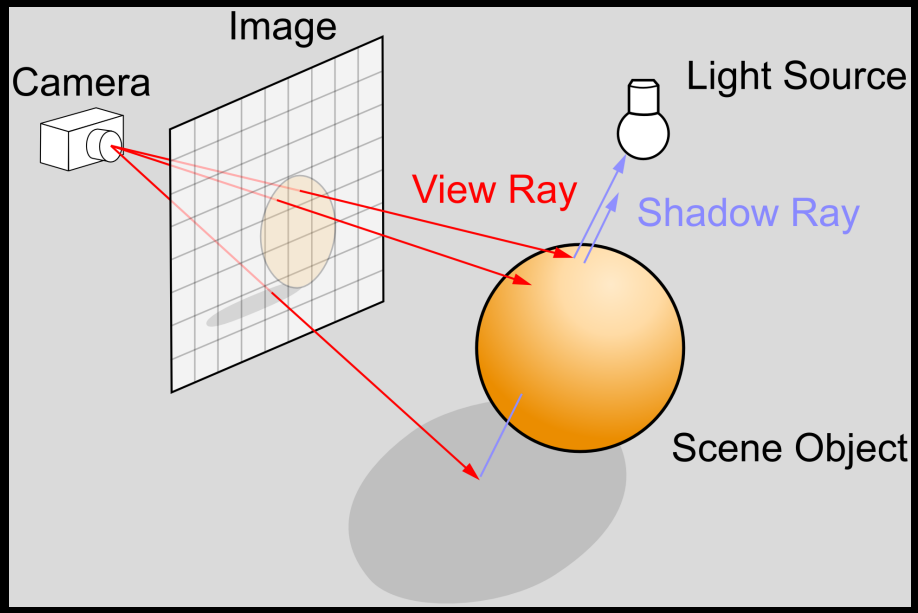
today we use the same principles behind the earliest cg



[martin newell, link]

rendering system : light simulation

emit from source, enter camera, bounce around scene



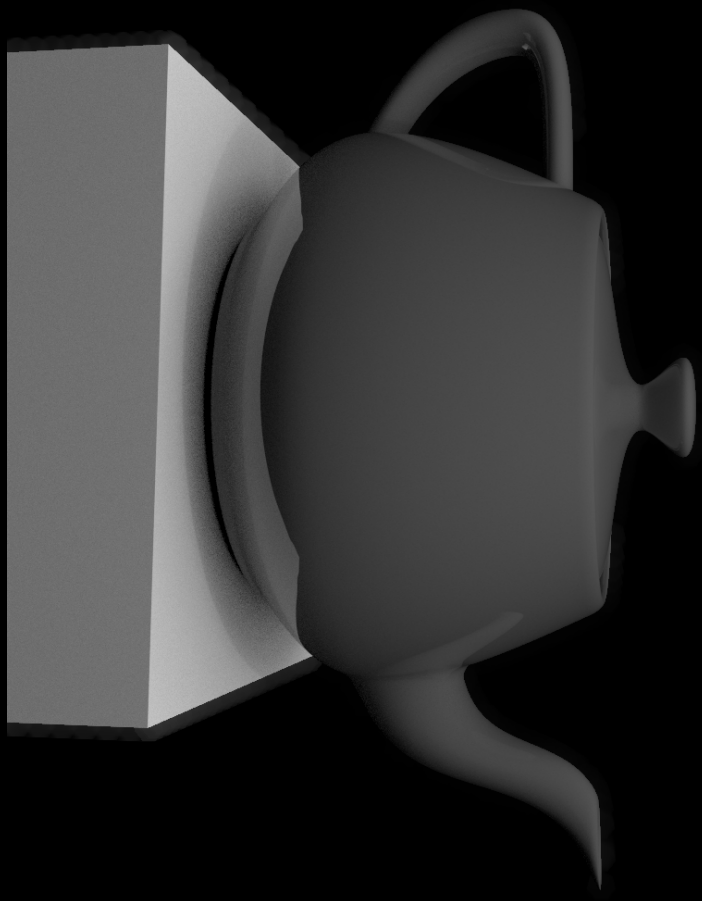
[henrik link]

modeling quality $\Rightarrow$ render quality

(light source, camera, materials, surfaces)

modeling light sources is relatively easy

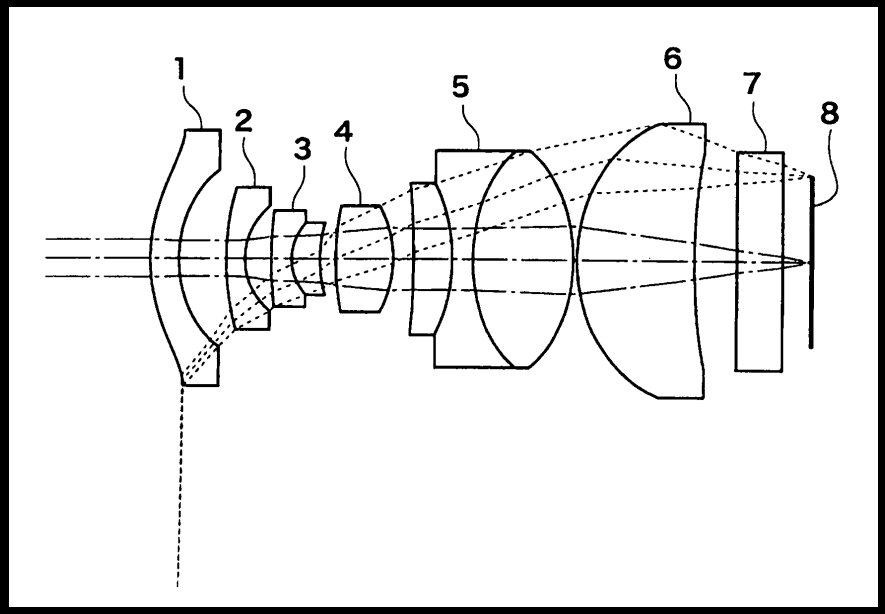
[paul debevec link]



[oscar leif link, paul debevec link]

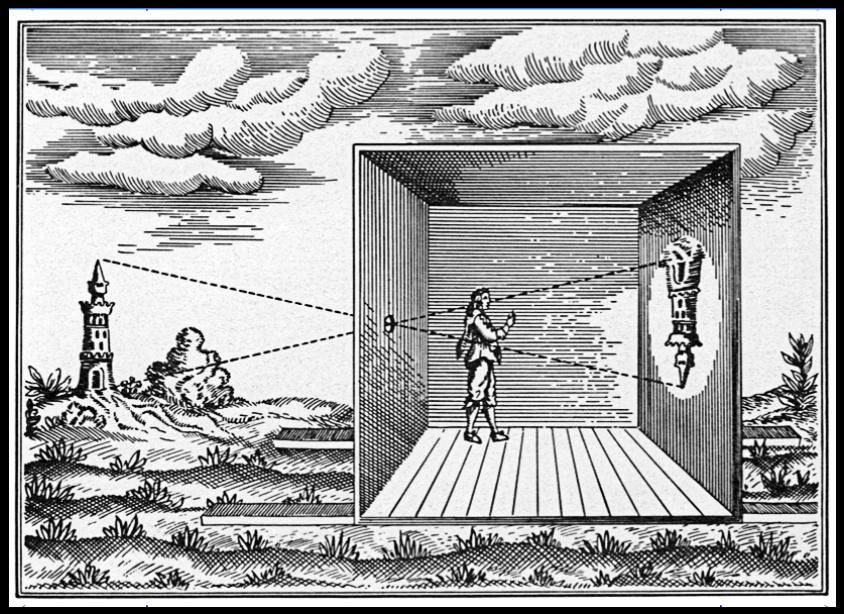
[oscar leif link, paul debevec link]

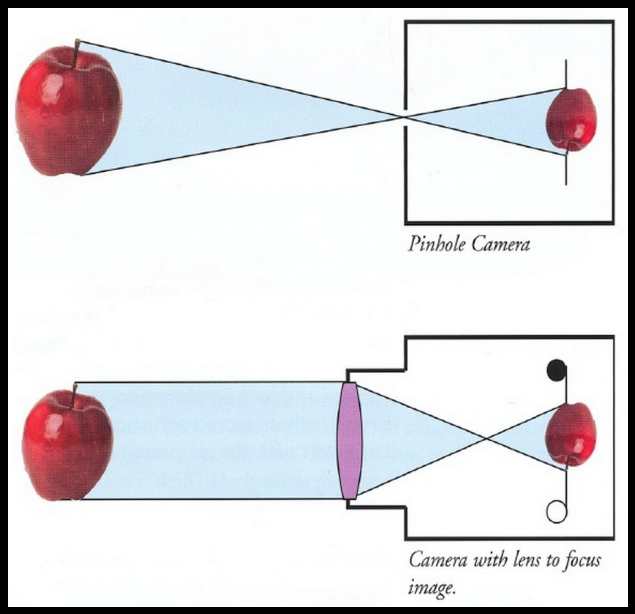
modeling camera lens system can be complex



[duncan meeder, petapixel link, koichi wakamiya link]

pinhole works surprisingly well, much easier to control





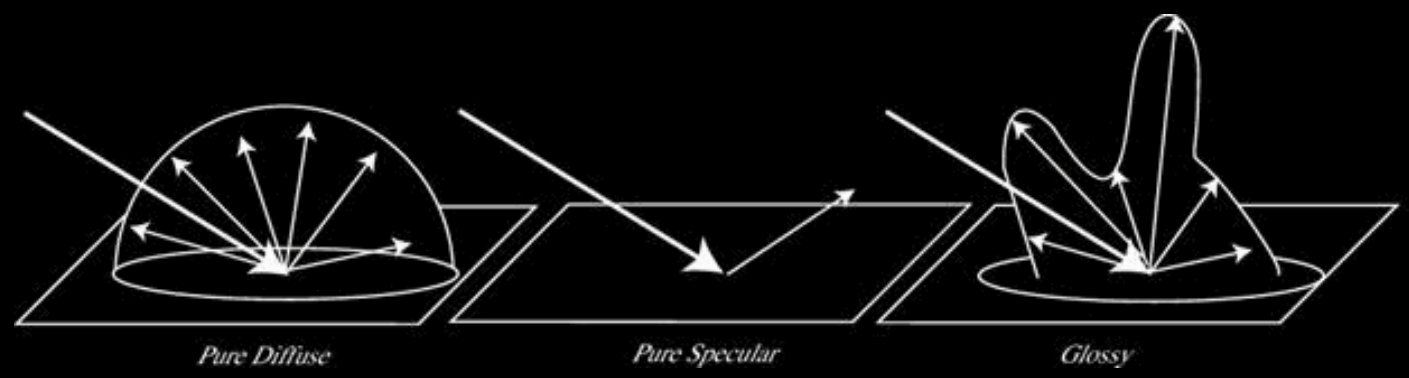
[original author unknown link, john munno link]

modeling materials is not difficult

[marc tardif link]

light reflects off surface in all directions at different amounts

materials describe how much light bounces

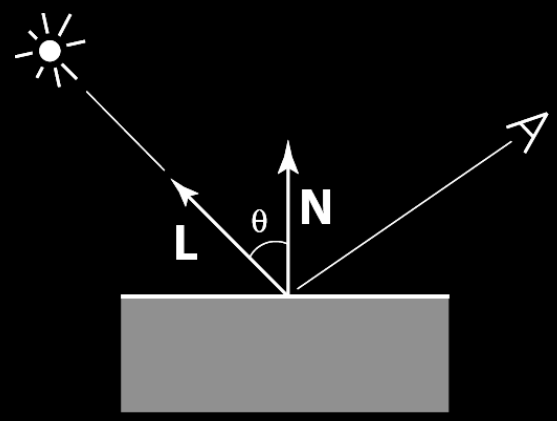


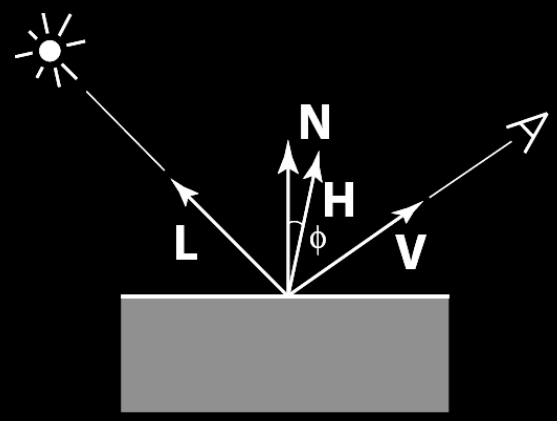
[andrey s. lebedev link]

we can faithfully describe almost any material as a math function (ρ) with just a few parameters

diffuse $\rho(\ldots) = \dfrac{R_d}{\pi}$

specular $\rho(\ldots) = R_s(\mathbf{h} \cdot \mathbf{n})^p$



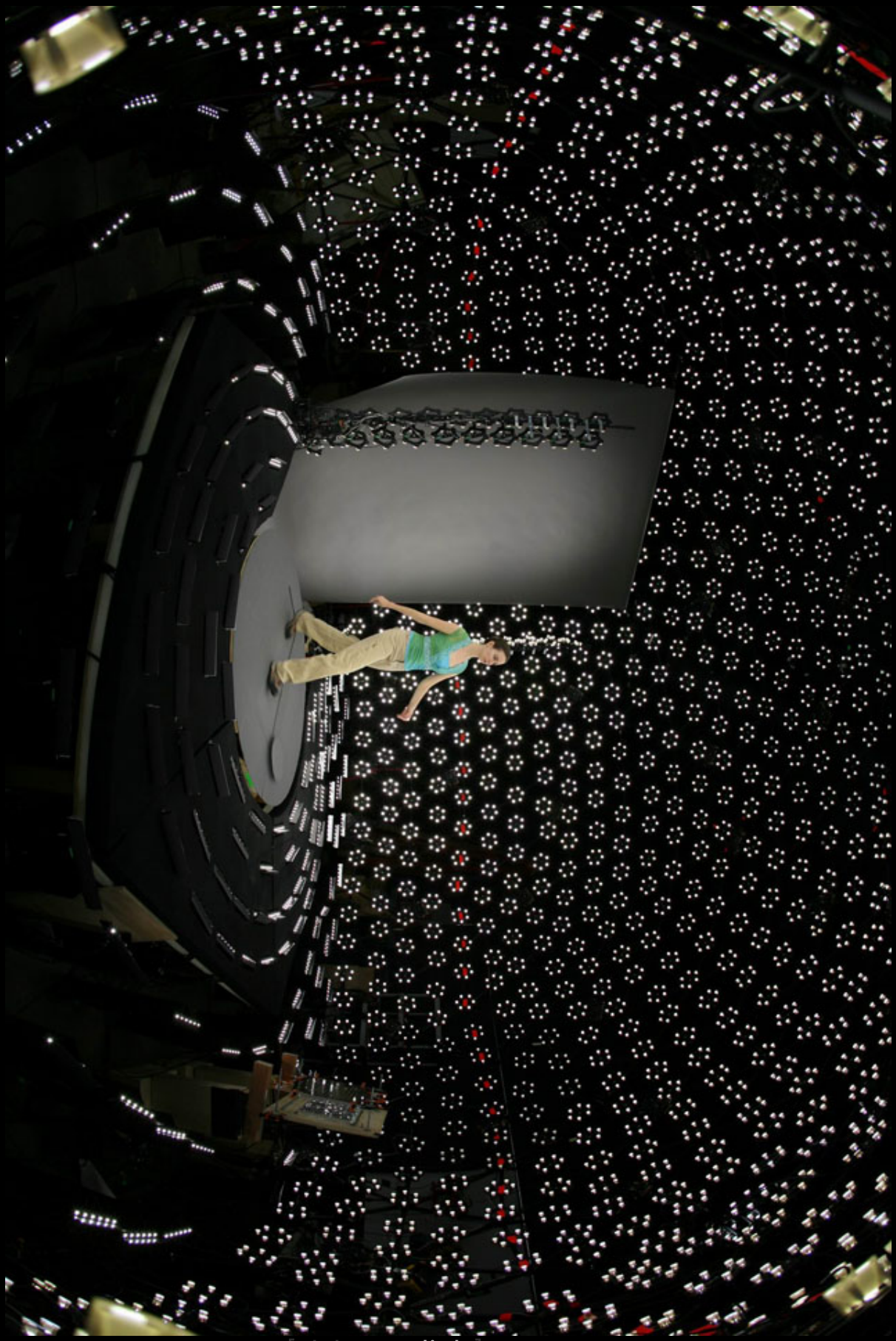


$$
\begin{aligned}
\rho(\ldots) &= a \cdot b \cdot F(\omega_i \cdot \mathbf{h}) \\
a &= \frac{\sqrt{(n_u + 1)(n_v + 1)}}{8\pi} \\
b &= \frac{(\mathbf{n} \cdot \mathbf{h})^{n_u \cos^2 \phi + n_v \sin^2 \phi}}{(\mathbf{h} \cdot \omega_i) \max(\cos\theta_i, \cos\theta_o)} \\
F(\omega_i \cdot \mathbf{h}) &= R_s + (1 - R_s)(1 - (\omega_i \cdot \mathbf{h}))^5
\end{aligned}
$$

(schlick's approximation to the fresnel equation)

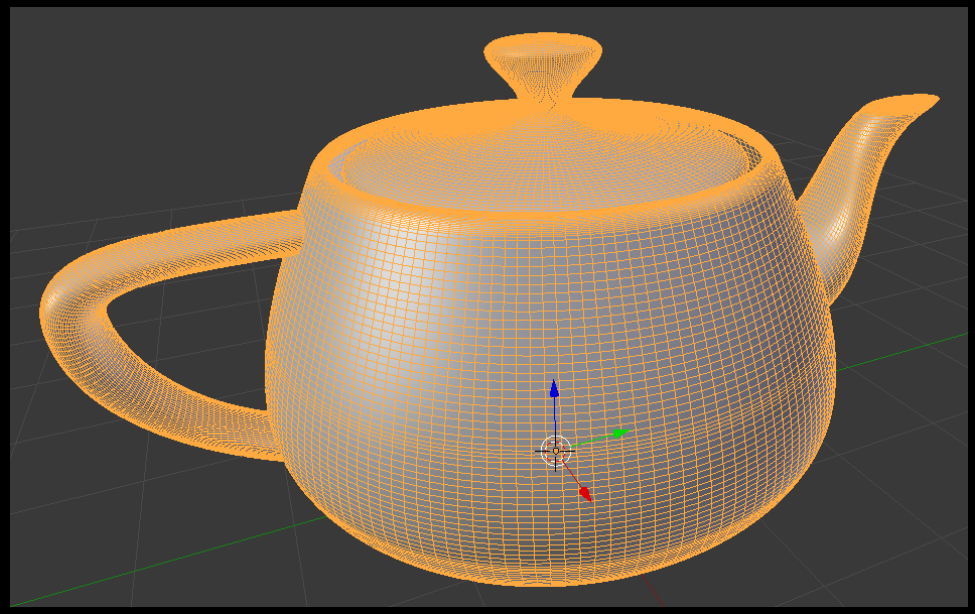
[jensen]

[jensen]



[debevec, link]

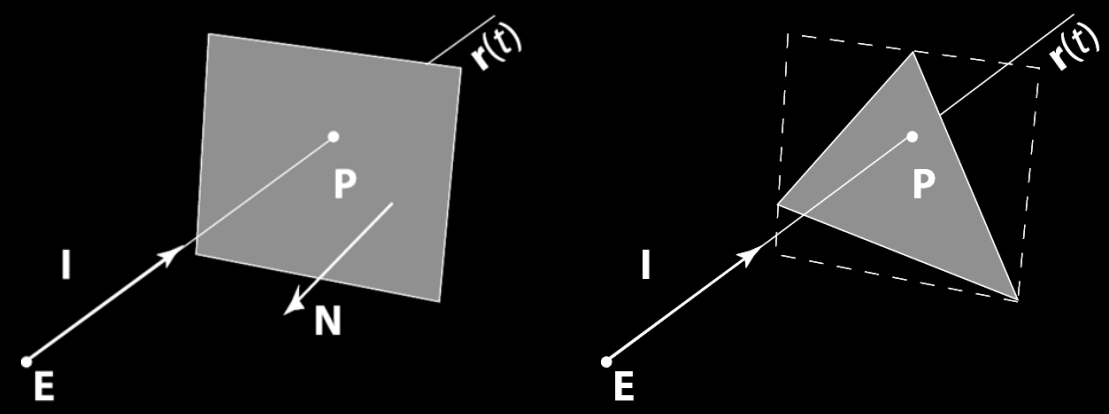
modeling surfaces is straightforward



use triangles to describe any surface to arbitrary precision

using ray for light and triangle for surface,
simulation involves simple algebra

ray	$P = P_o + tV$
plane	$P \cdot N + d = 0$
ray-plane	$t = -(P_o \cdot N + d)/(V \cdot N)$



the rendering equation puts the parts together

all photo-realistic graphics engines solve this equation

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

emit (L_e), camera (L_o), bounce (ρ, L_i, L_o)

the only thing remaining is doing the light simulation

solve the rendering equation

"what do we do now sheriff?"

"now, we *render*"

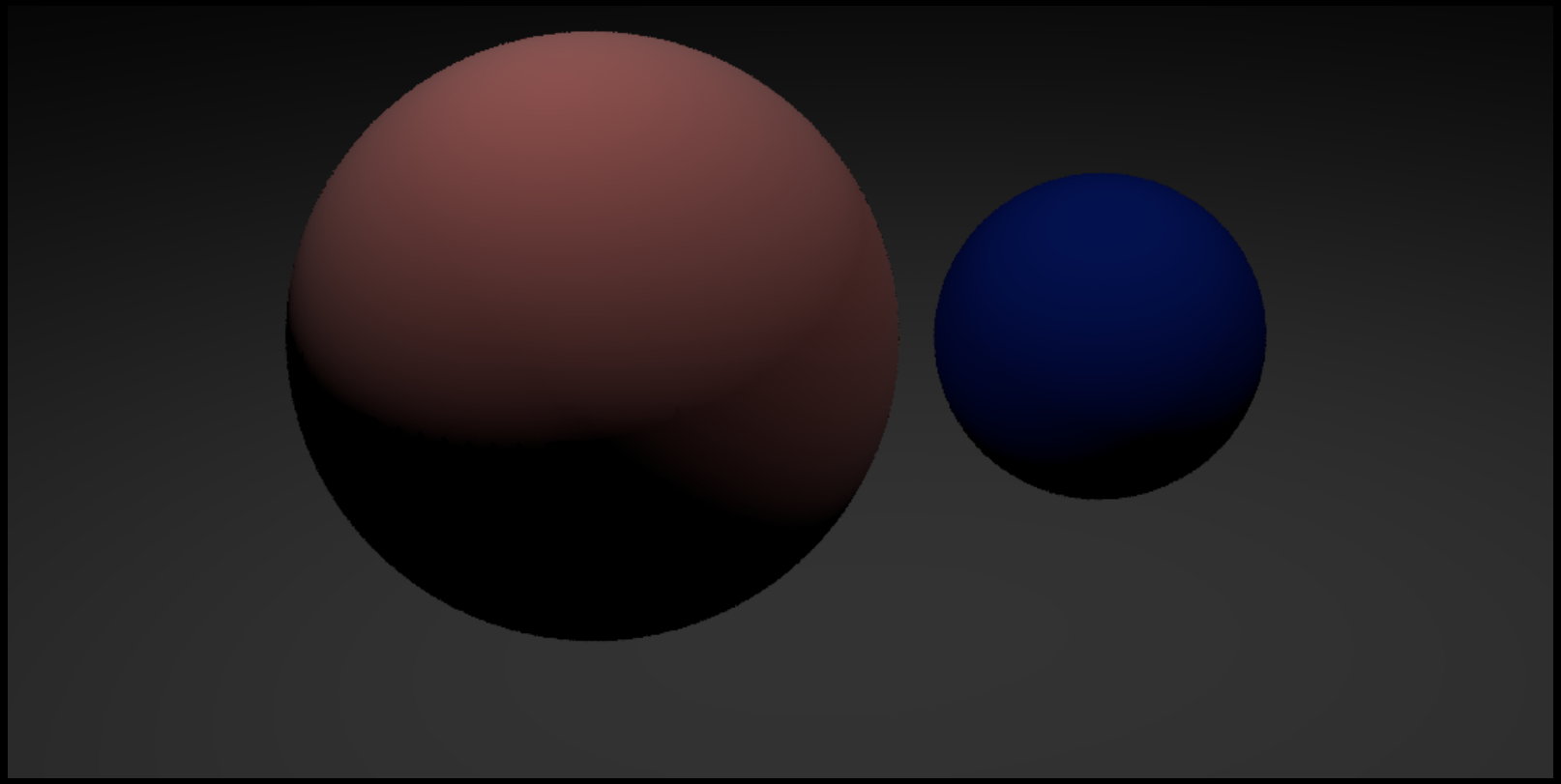
the bad

light simulation isn't cheap

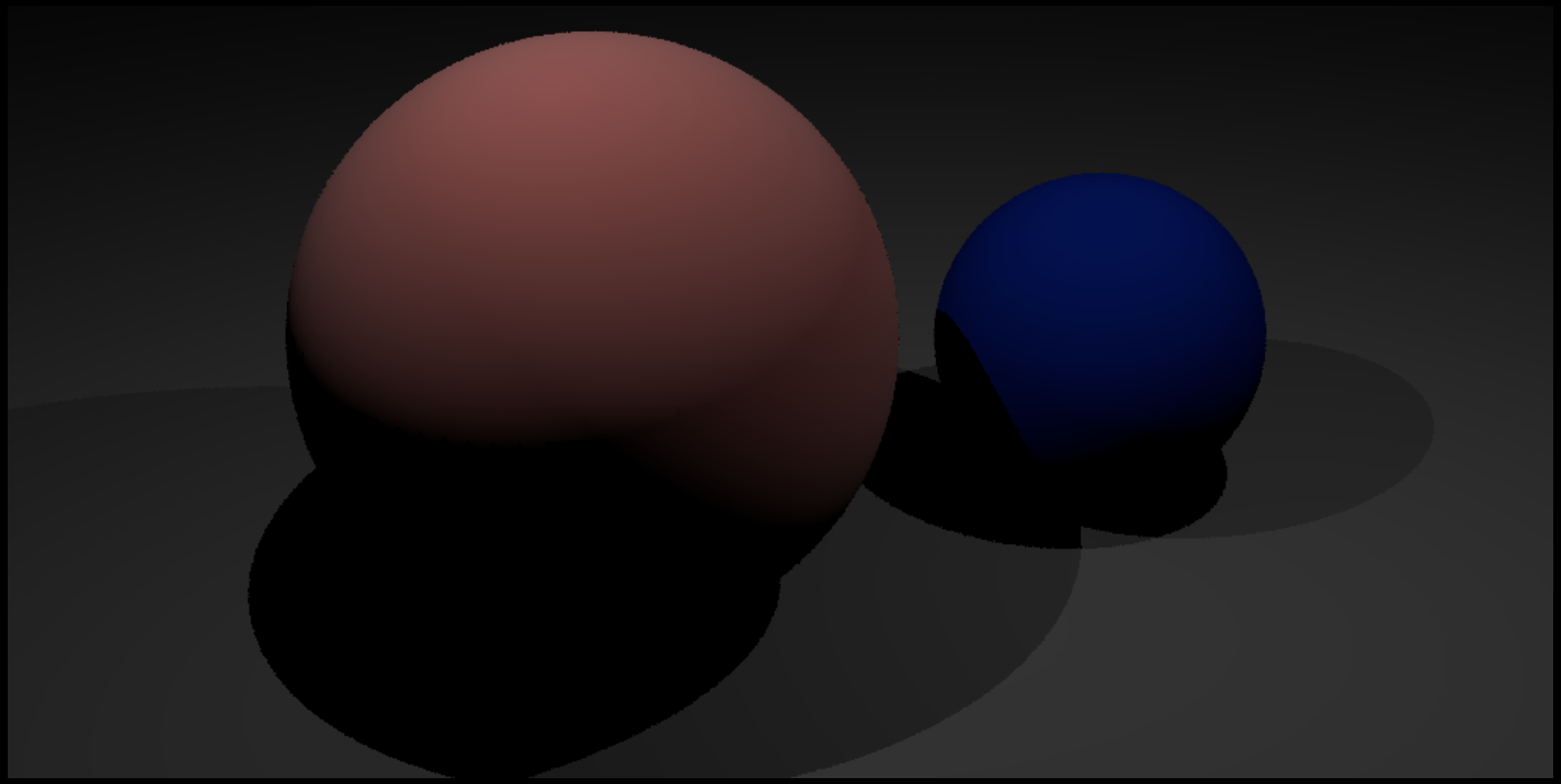
steps toward photo-realism causes computation time to grow

for example, let's look at rendering an image
two spheres on a plane with two lights

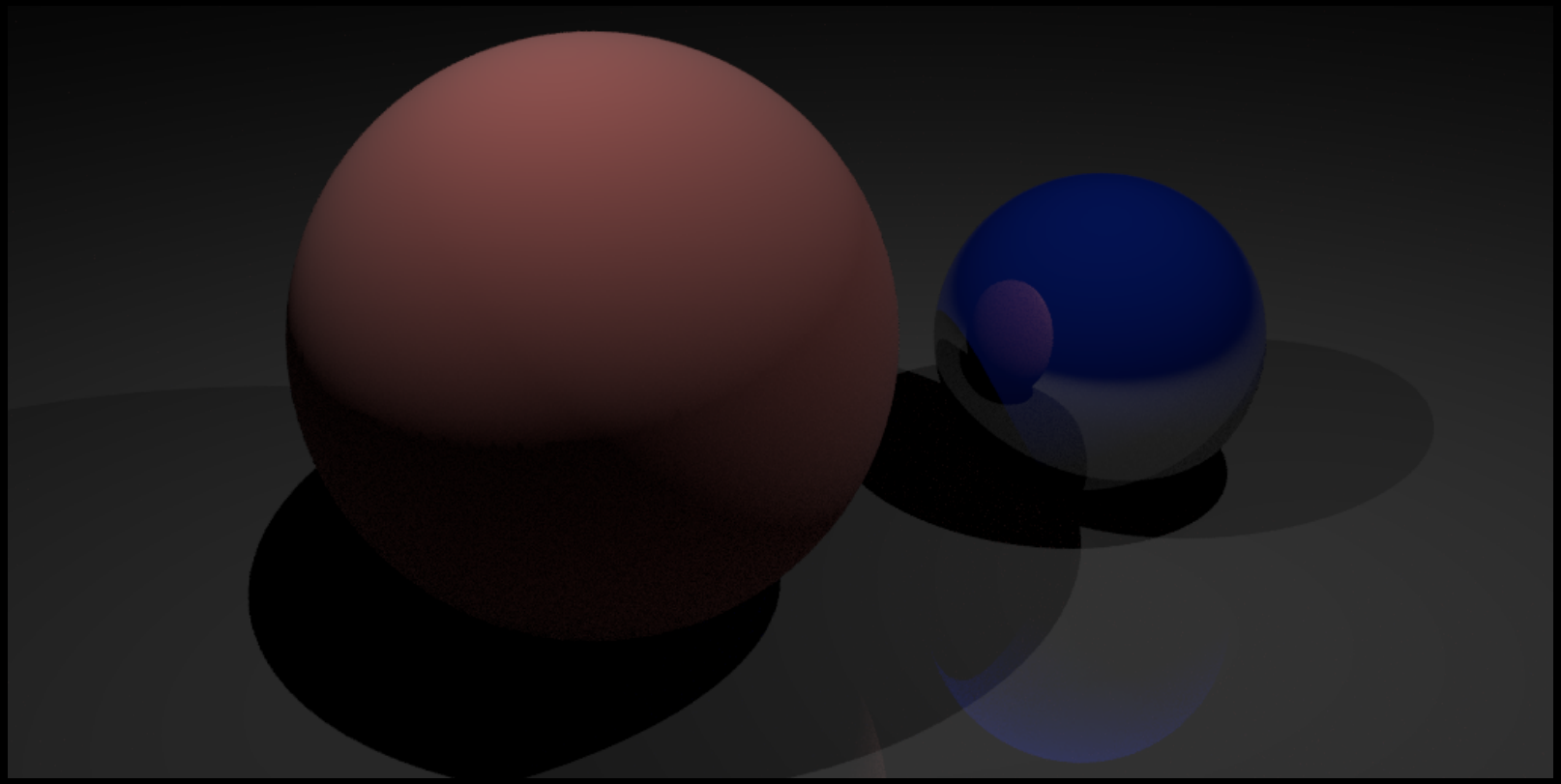
note: following simulations are already "smart"



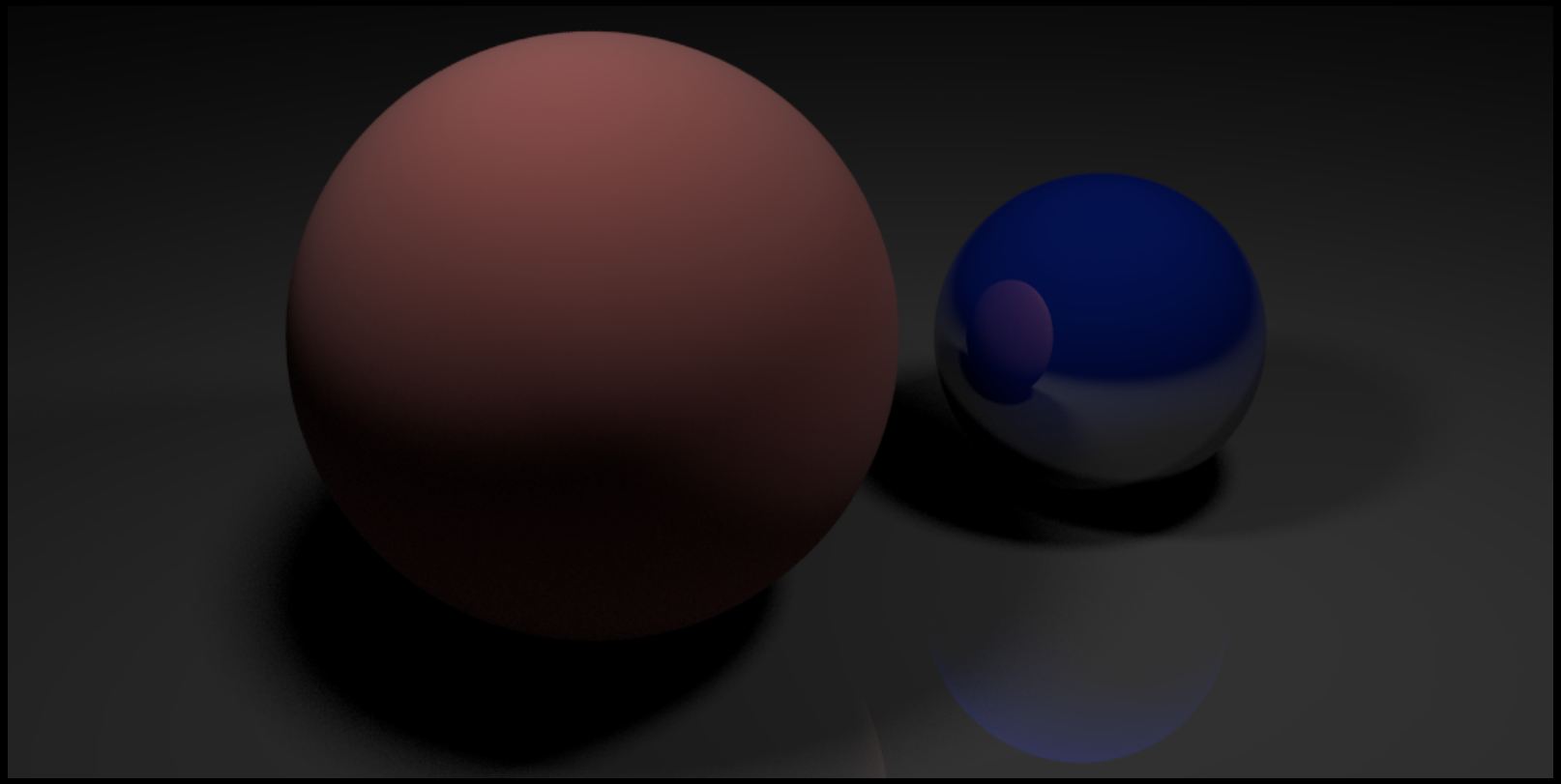
simple
< 1 second



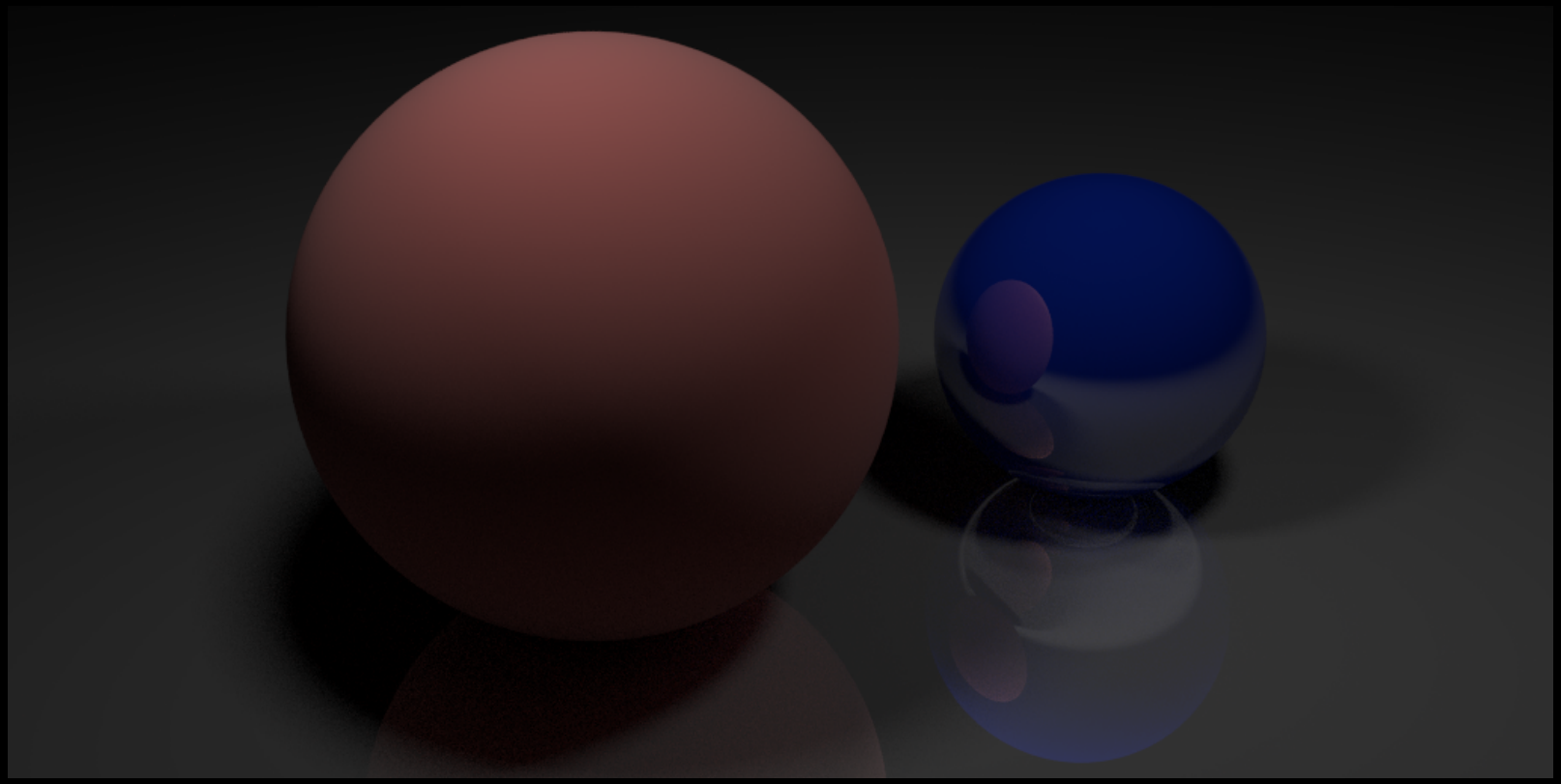
shadows
1 second



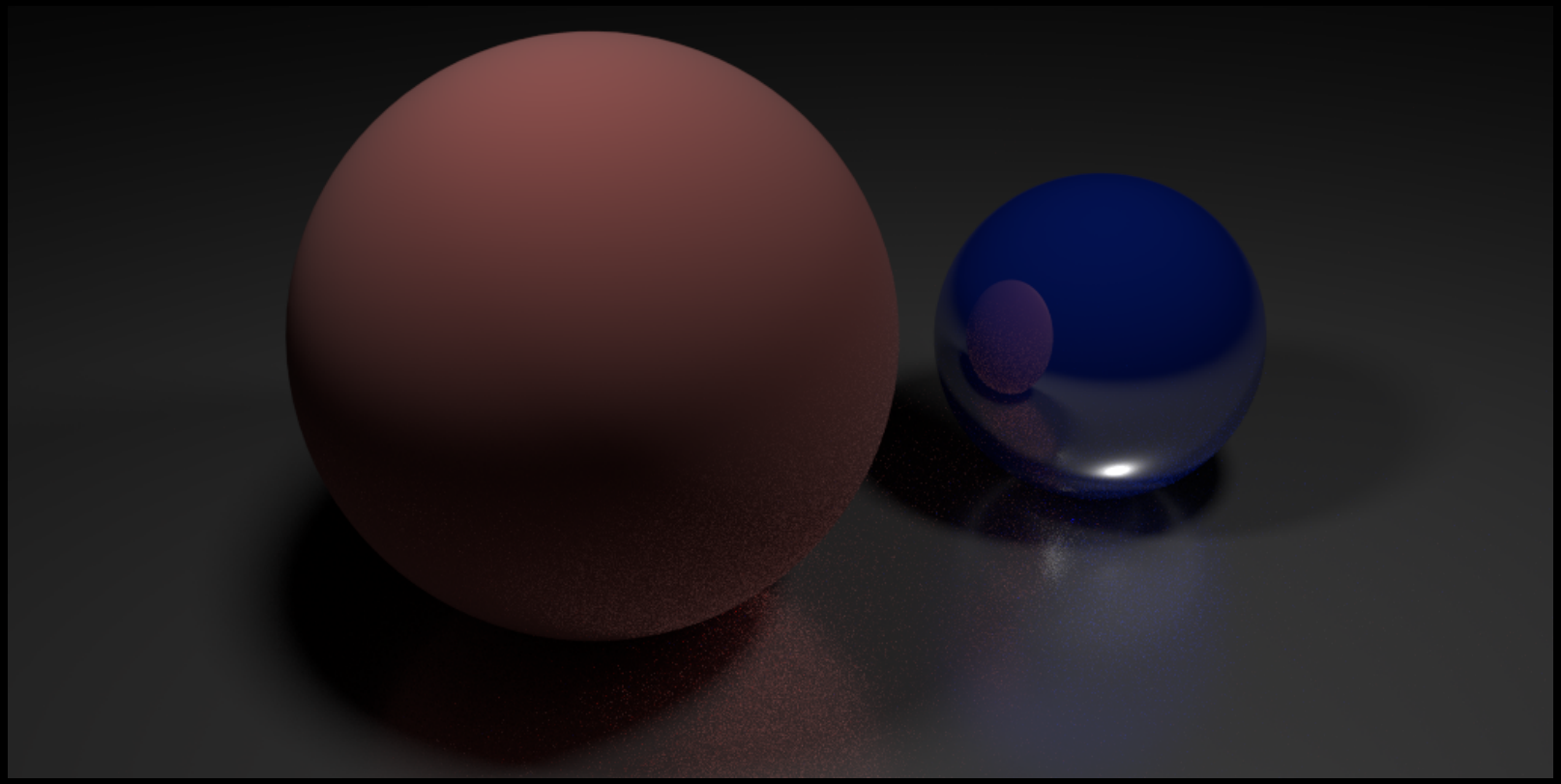
mirror reflection
7 seconds



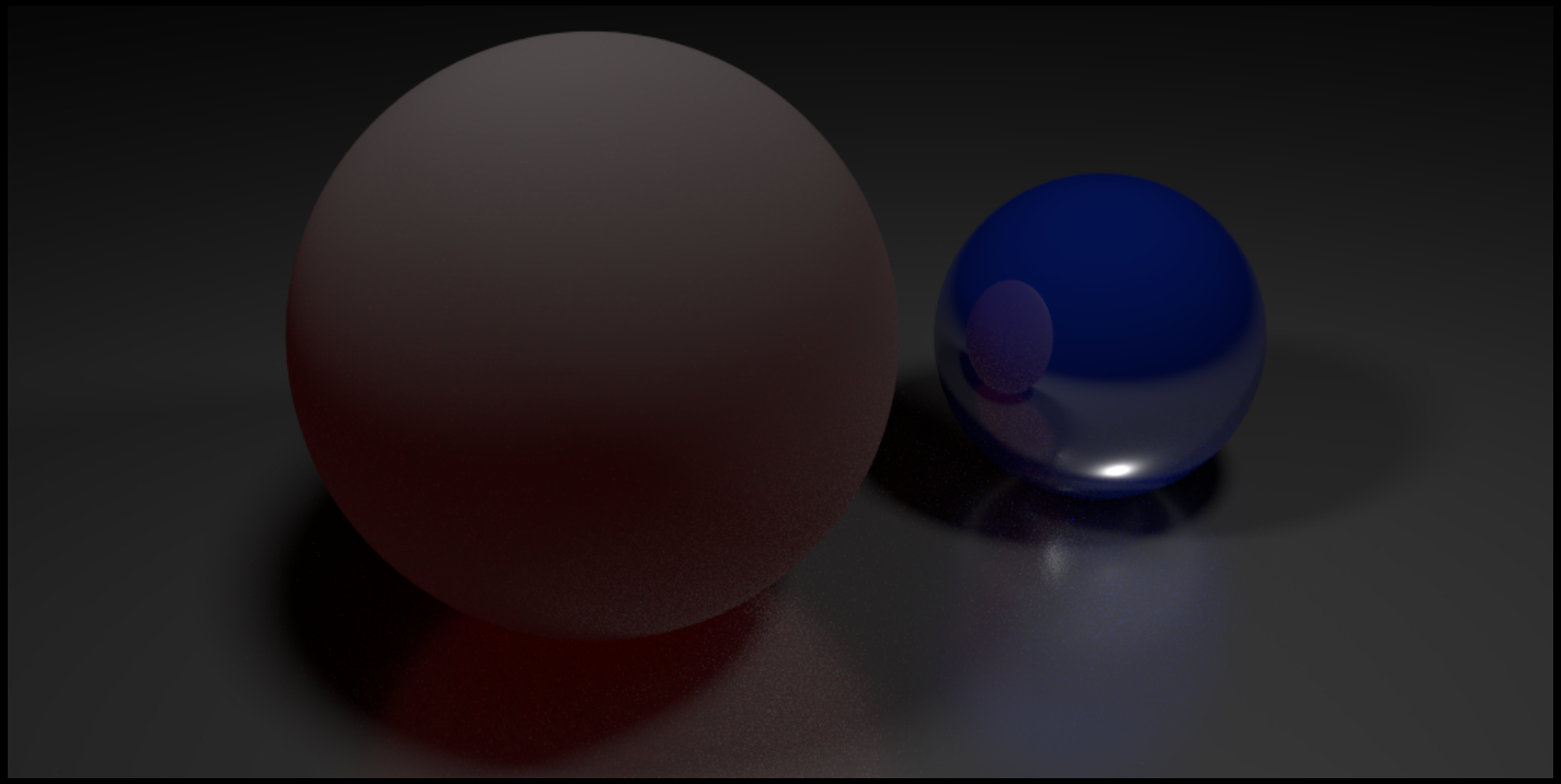
soft shadows
30 seconds



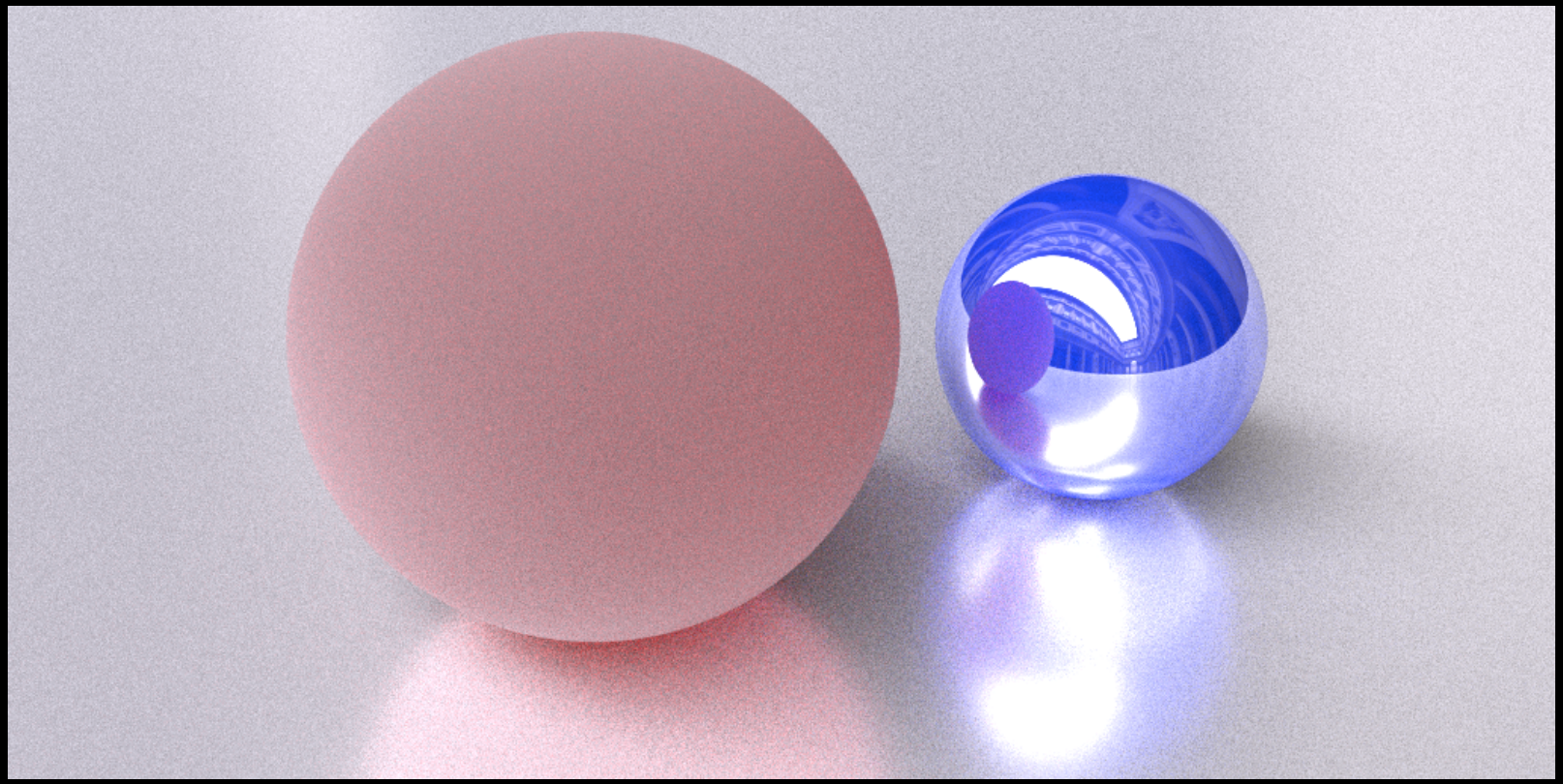
multiple bounces
32 seconds



glossy reflection
258 seconds



translucency
681 seconds



environment
776 seconds

and it only gets worse from here

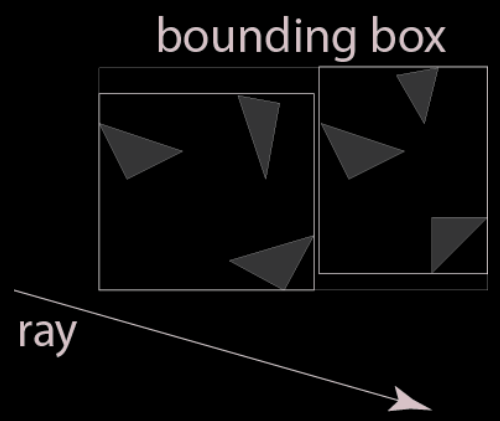
adding more realism involves computing more light bounces

the weird

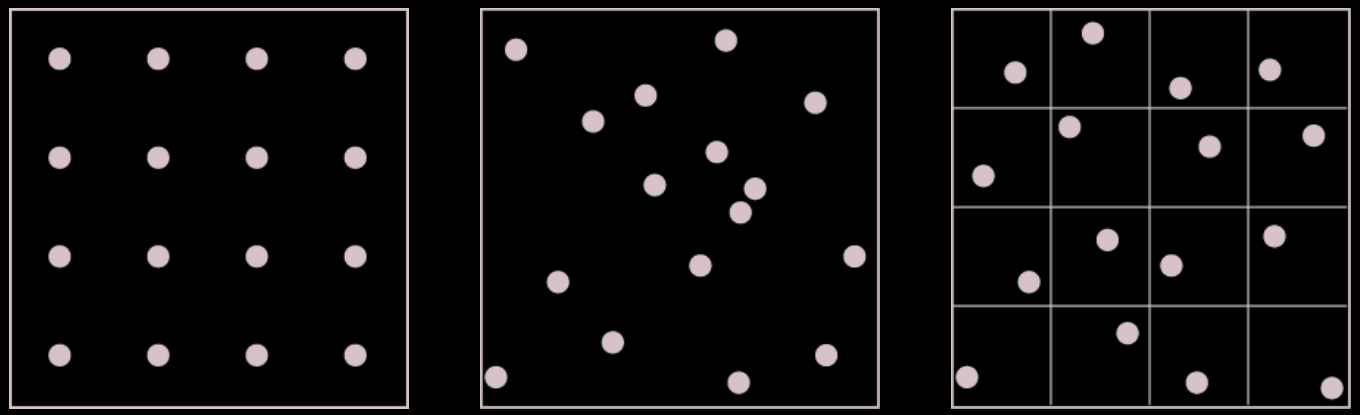
we could use approximations for the rendering equation
but the final image will not be accurate

the following are a few different ways
to improve render times
without changing final image

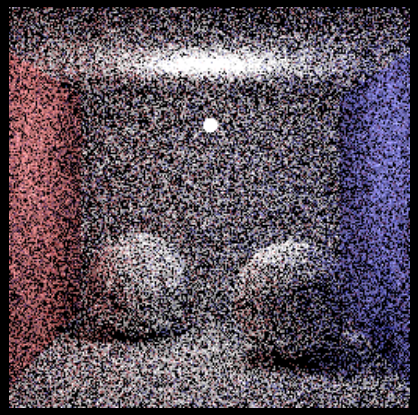
put triangles into boxes

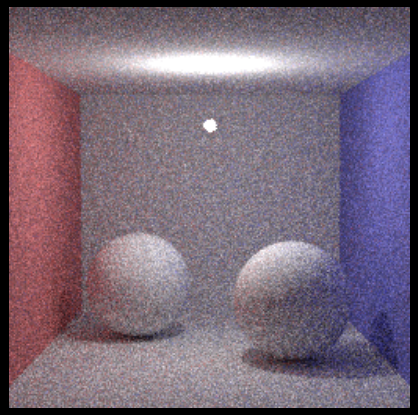


throw darts



roll multiple unfair dice





[merity et al. link]

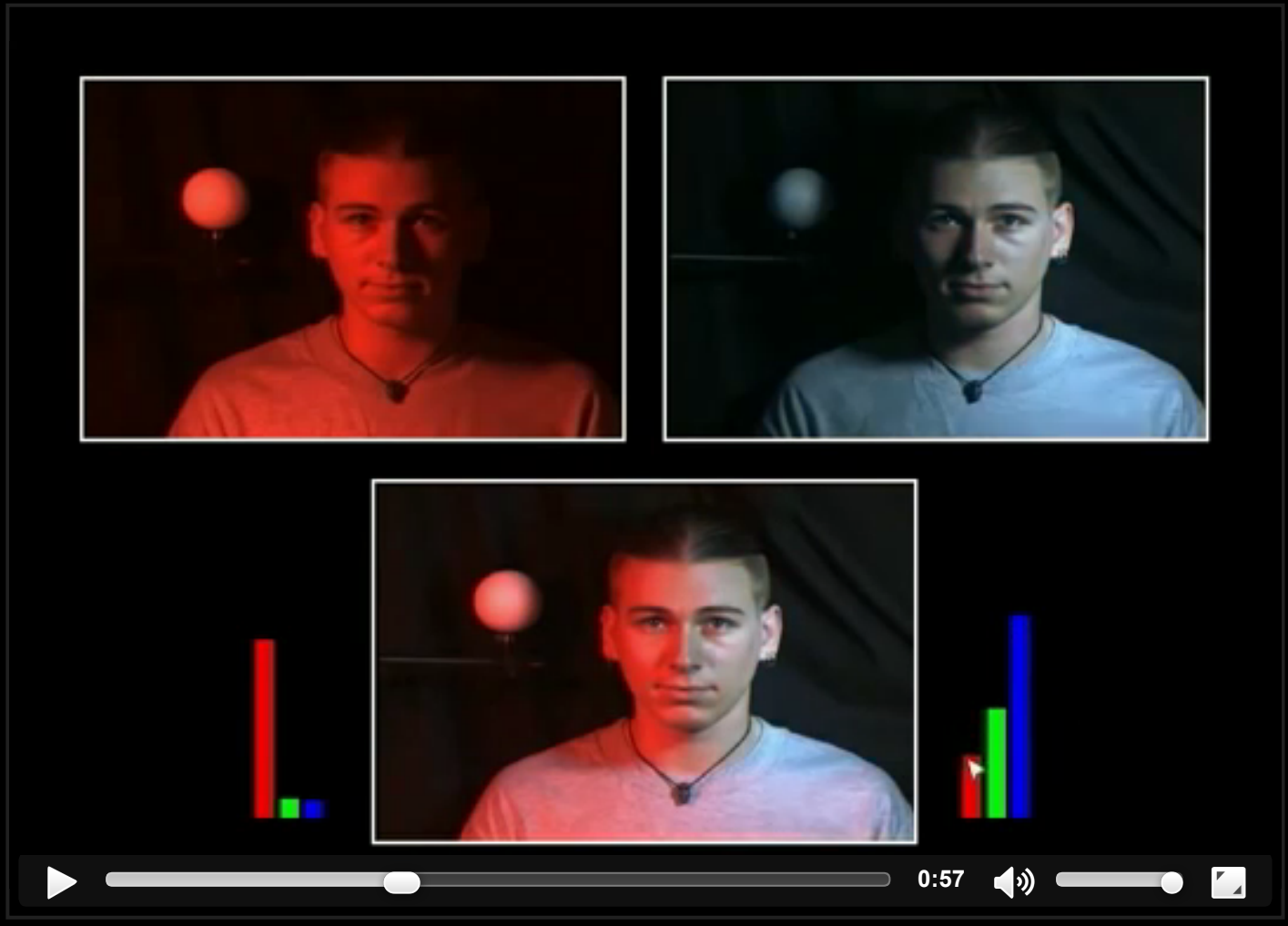
the end

[the third and the seventh, by alex roman link]

[eye piece, by chris jones link]

[snappers facial rig, by snappers mocap link]

[roundtrip, by david gruwier link]



[light stage, hawkins et al. link]