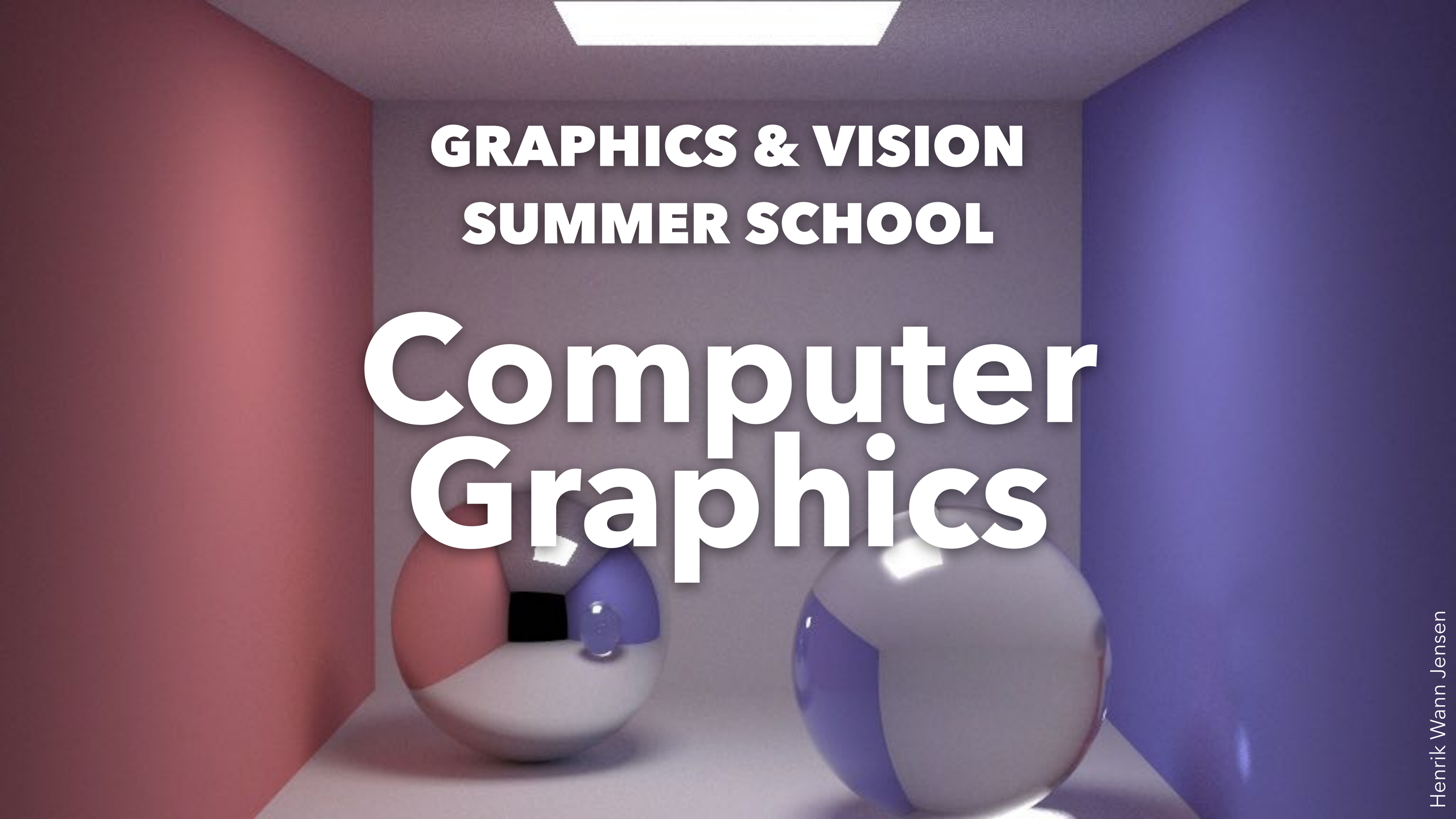
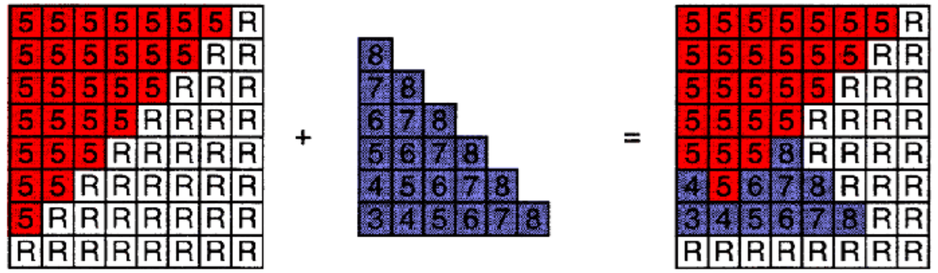
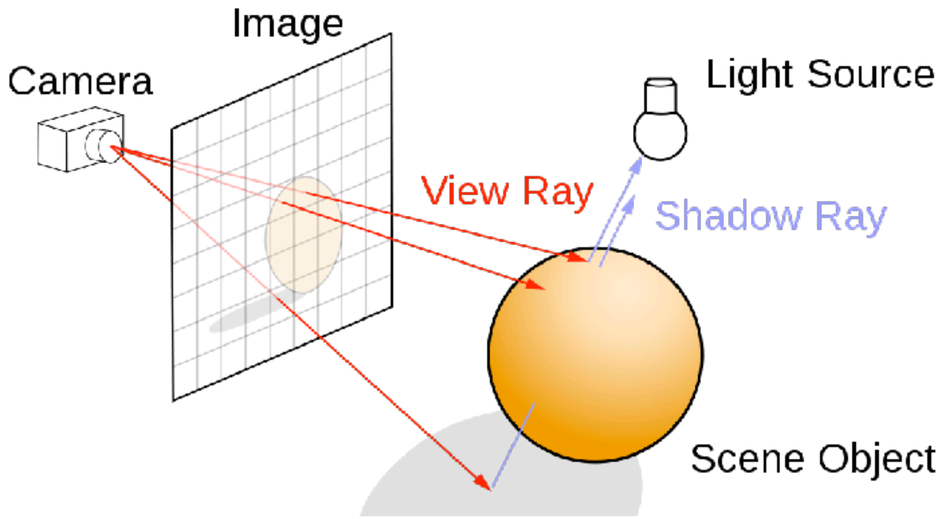

GRAPHICS & VISION
SUMMER SCHOOL
Computer
Graphics
Henrik Wann Jensen

Rasterization vs. Ray tracing

for each *shape*:
    for each *sample*:
        get *point* where *shape* covers *sample*
        if *point* is closest point seen by *sample*:
            sample.colour = shade(*point*)



for each *sample*:
    for each *shape*:
        get *point* where *shape* covers *sample*
        if *point* is closest point seen by *sample*:
            sample.colour = shade(*point*)



Ray tracing

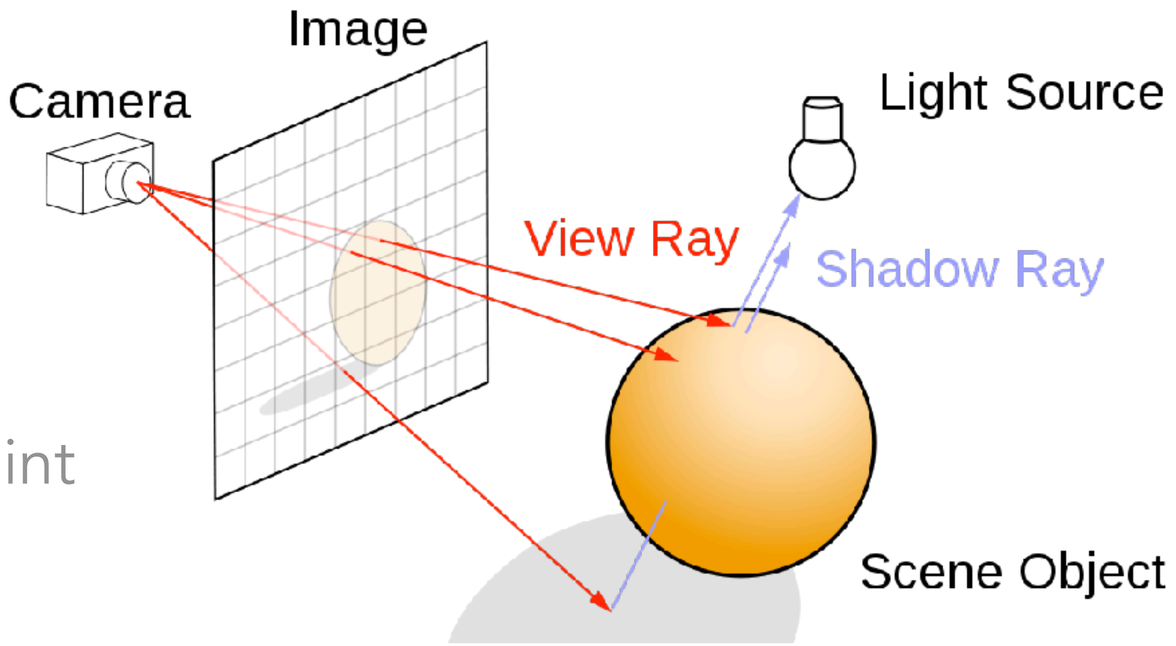
For each sample (x, y):

Generate eye ray $\mathbf{r}(t) = \mathbf{o} + t\mathbf{d}$

Find the closest intersection

Get shaded colour at intersection point

Set sample colour to it



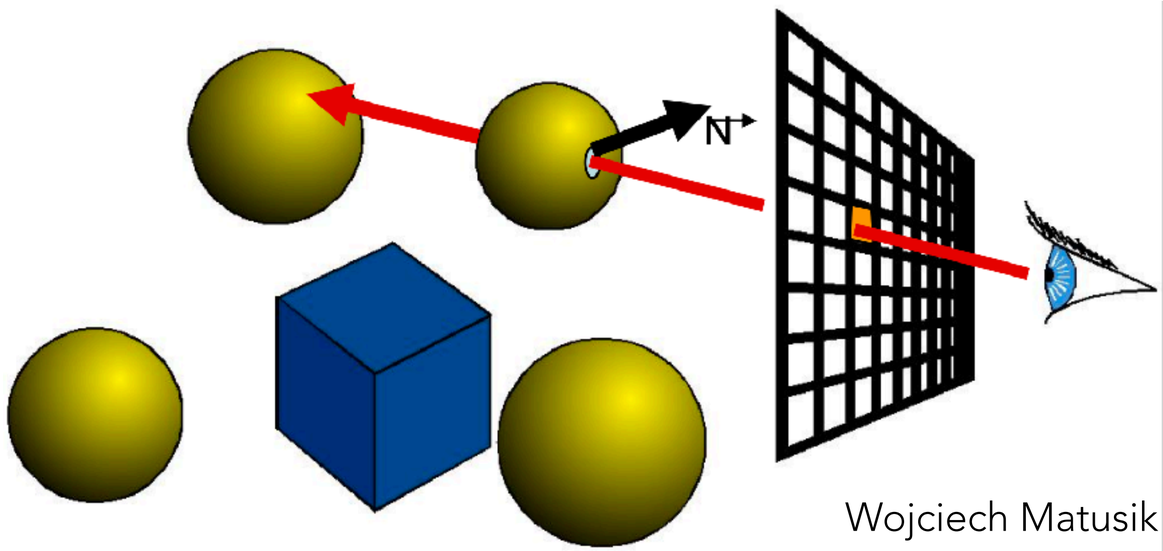
Ray-surface intersection

Given a ray $\mathbf{r}(t) = \mathbf{o} + t\mathbf{d}$, find closest intersection i.e. minimum t

Return info needed for shading:

- Position **p**
- Normal **n**
- Object ID / material properties

(Roughly the same data you would need in a fragment shader)

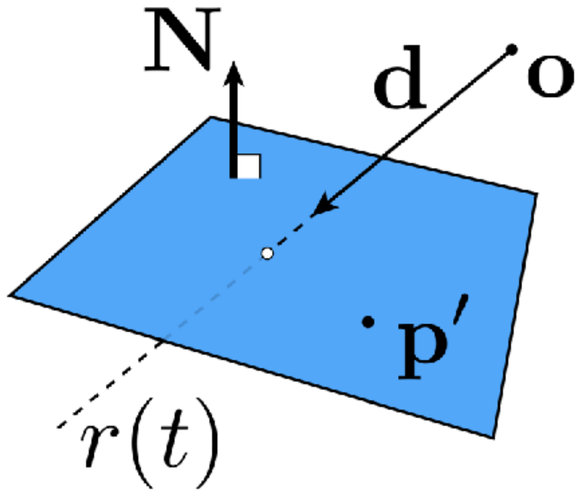


Wojciech Matusik

Ray-plane intersection

Plane equation: $\mathbf{n} \cdot (\mathbf{p} - \mathbf{p}_0) = 0$

any known point on the plane

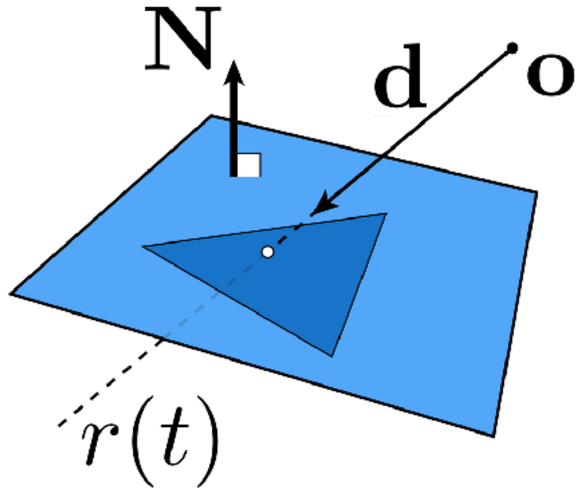


$$\mathbf{n} \cdot (\mathbf{o} + t\mathbf{d} - \mathbf{p}_0) = 0$$

$$t = (\mathbf{n} \cdot (\mathbf{p}_0 - \mathbf{o}))/(\mathbf{n} \cdot \mathbf{d})$$

Ray-triangle intersection

Intersect ray with plane, then check if it is inside triangle?



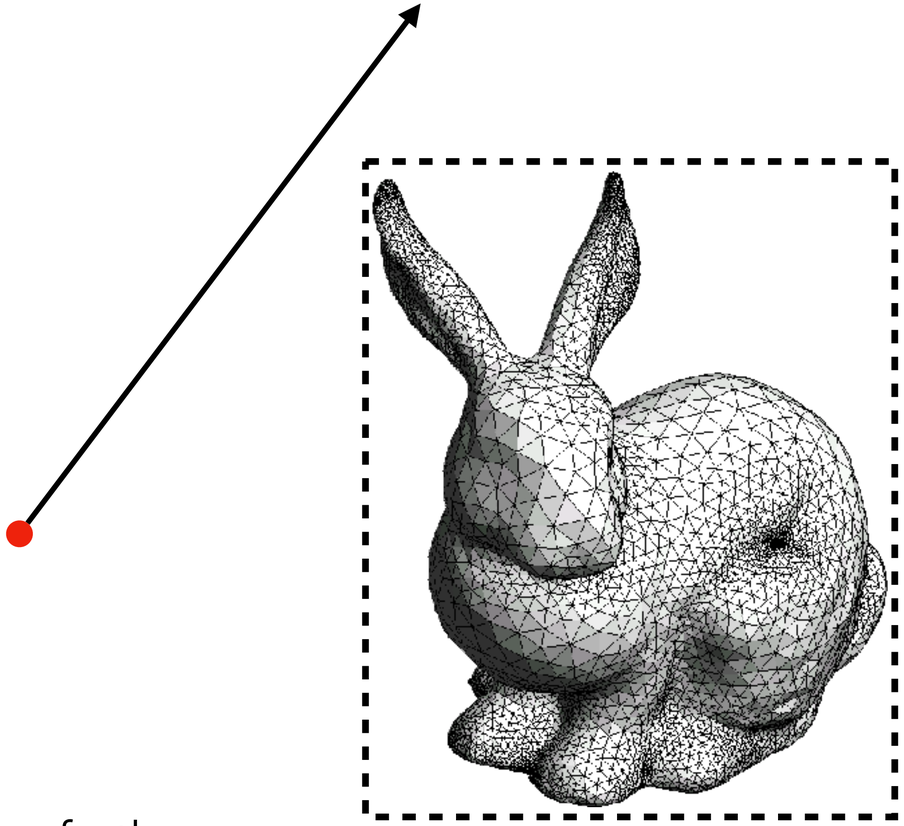
Ray-mesh intersection

Naïve approach: Test ray with all triangles, return the earliest hit.

Cost = *O*(#triangles)! Can we speed it up?

Construct a conservative **bounding volume**: all mesh triangles lie inside it

Super easy to reject rays that don't come close to intersecting the mesh.

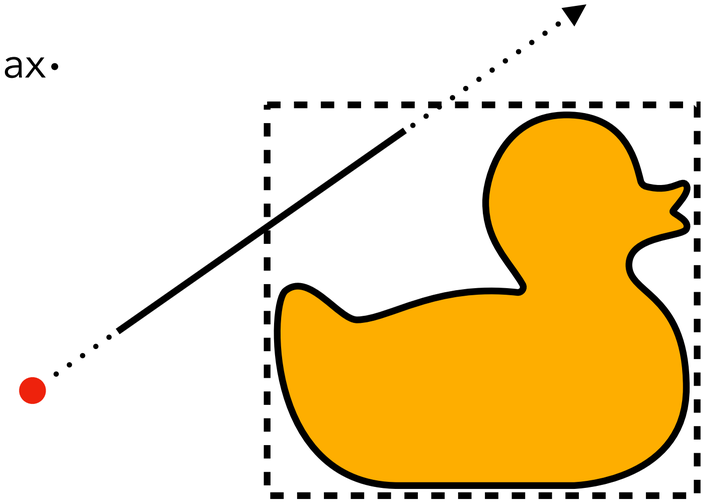


In practice, we use bounding volume **hierarchies** to speed things up further.

Object-oriented raytracer design

We can ray trace any shape as long as it provides the following methods:

- `bool hit(Ray` $\mathbf{o} + t\mathbf{d}$`, real` t_{min}`, real` t_{max}`, HitRecord &rec)`
  - Only consider intersections in the range $t_{\text{min}} \leq t \leq t_{\text{max}}$.
    Usually [0, ∞] for eye rays
  - If hit, write the position, normal, material, etc.
    into the `HitRecord`
- `Box bounding_box()`
  - For early exit



Ray tracing

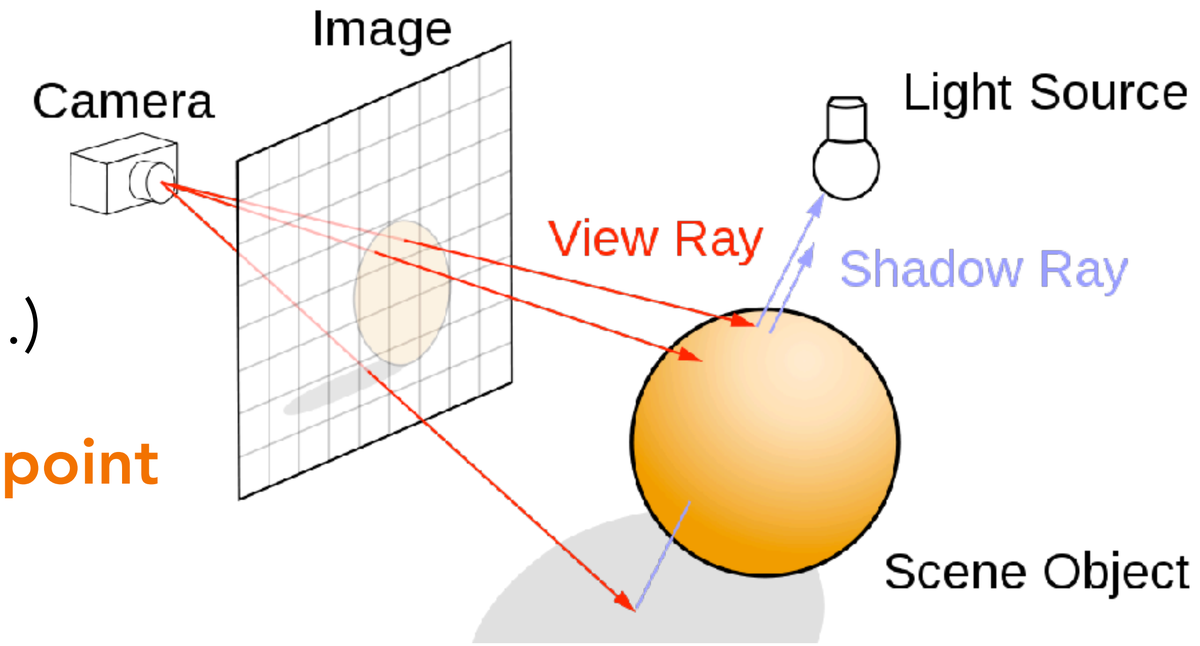
For each sample (x, y):

Generate eye ray $\mathbf{r}(t) = \mathbf{o} + t\mathbf{d}$

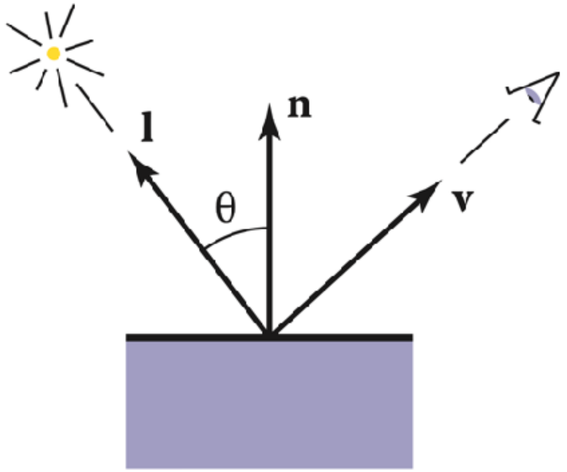
Find the closest intersection ($\mathbf{p}$, $\mathbf{n}$, …)

Get shaded colour at intersection point

Set sample colour to it



Shading the intersection point



Now we have position **p**, normal **n**, etc.
We can compute other vectors **v**, $\boldsymbol{\ell}$, **h**, etc. as usual

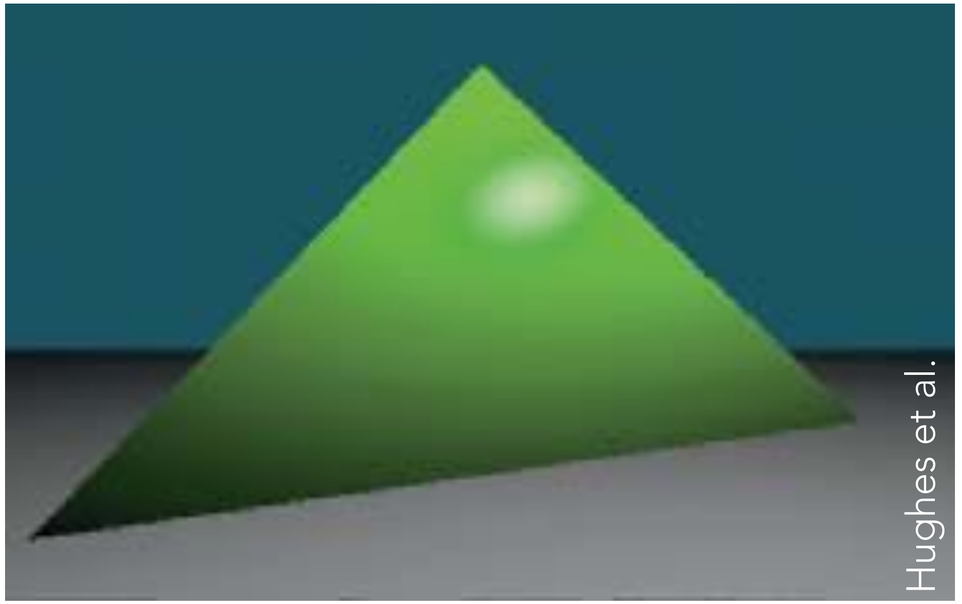
Apply your favourite reflection model, e.g. Blinn-Phong:

$$L = k_a I_a + \sum I_i \underbrace{(k_d \max(0, \mathbf{n} \cdot \boldsymbol{\ell}_i) + k_s \max(0, \mathbf{n} \cdot \mathbf{h}_i)^p)}_{f(\boldsymbol{\ell}_i, \mathbf{v})}$$

$f(\boldsymbol{\ell}, \mathbf{v})$ = what fraction of incident light from direction $\boldsymbol{\ell}$ is reflected to direction **v**

So far, what we have done is **ray casting**: shoot just one ray from the eye.

This should give us **exactly the same image** as we'd get from rasterization!



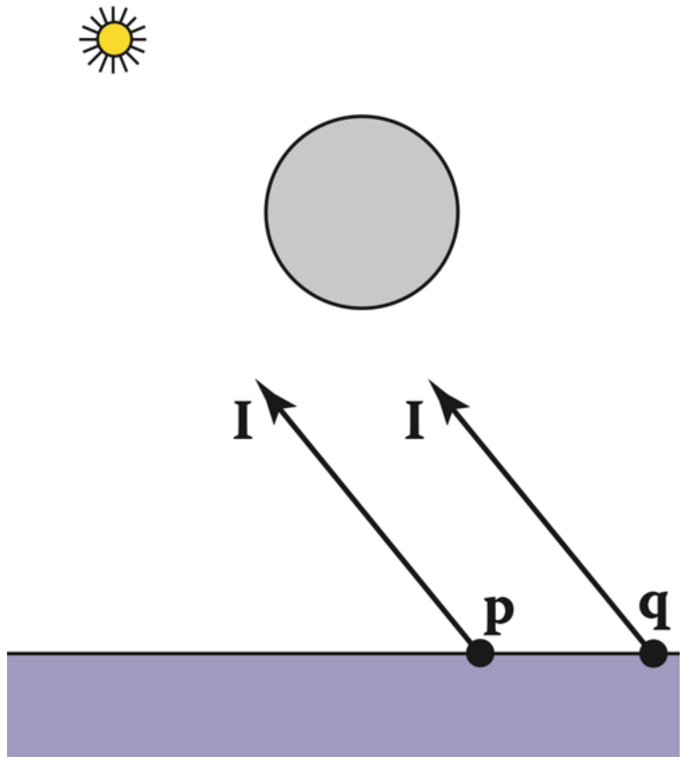
Now we will also trace secondary rays to get additional effects.

$$L = k_a I_a + \sum I_i f(\boldsymbol{\ell}_i, \mathbf{v})$$

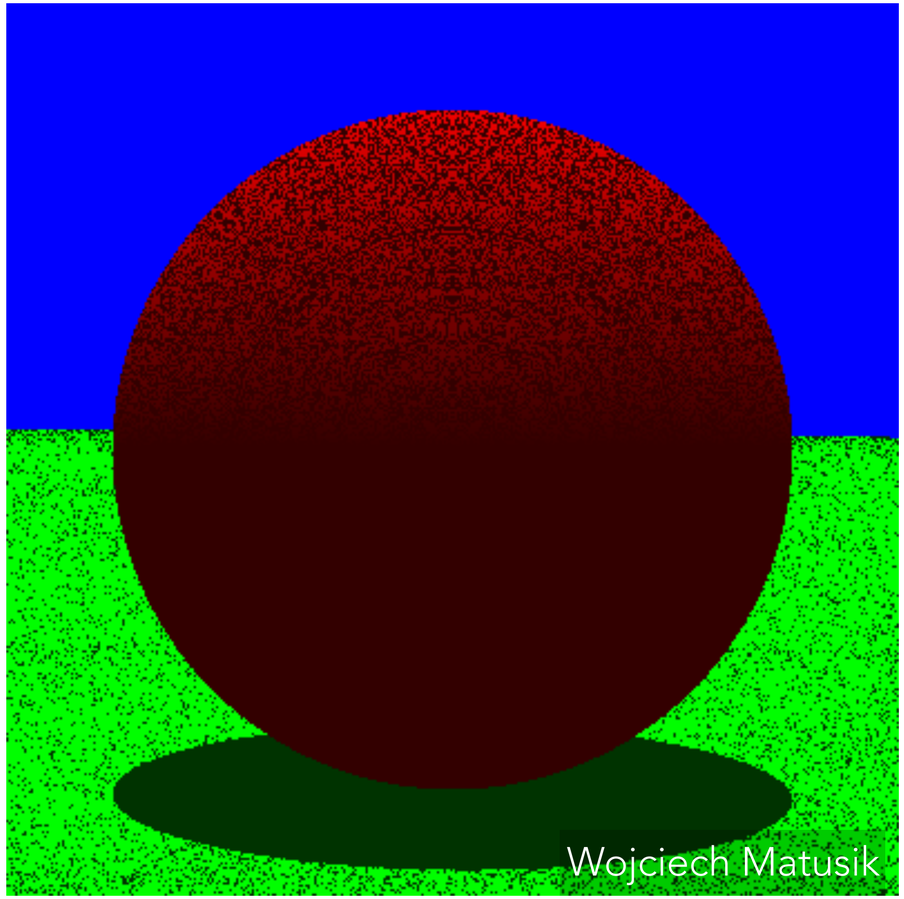
Light should only be included if it is **visible** from **p**

- Shoot a "shadow ray" $\mathbf{p} + t\boldsymbol{\ell}$ towards light source
- If no intersection within interval $[0, t_{light}]$, add its contribution to L

Why $[0, t_{light}]$ instead of $[0, \infty]$?



Marschner & Shirley

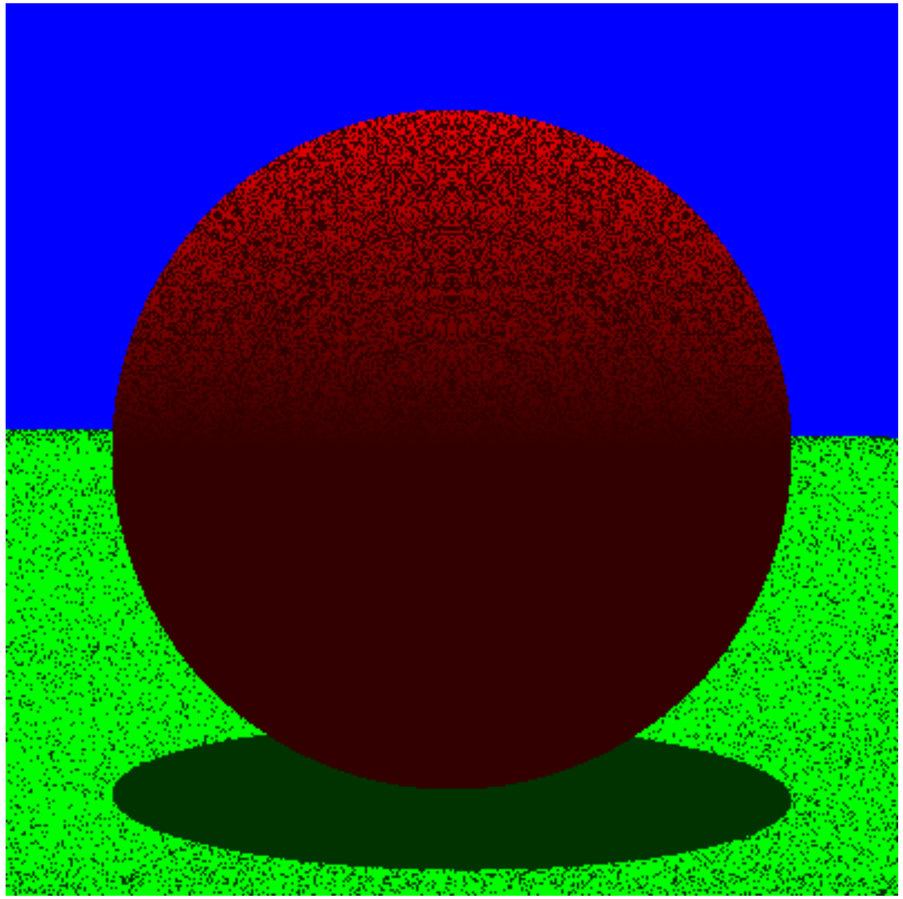


"**Shadow acne**"

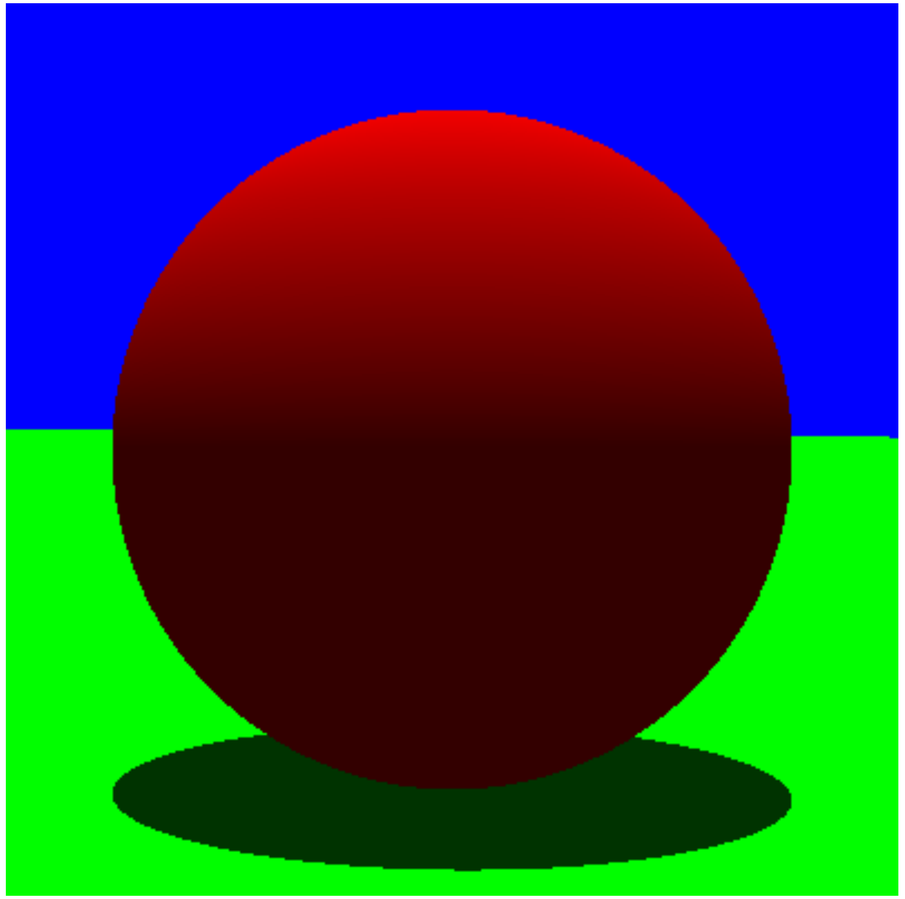
Why does this happen?

- **p** lies on surface
- **p** + $t\boldsymbol{\ell}$ intersects surface at $t = 0$
- Floating-point arithmetic may give e.g. $t = 0.000001$
- **p** thinks it's shadowed… by itself!

Solution: Pick a small positive ε ("bias") and only look for intersections in $[\varepsilon, t_{\text{light}}]$.

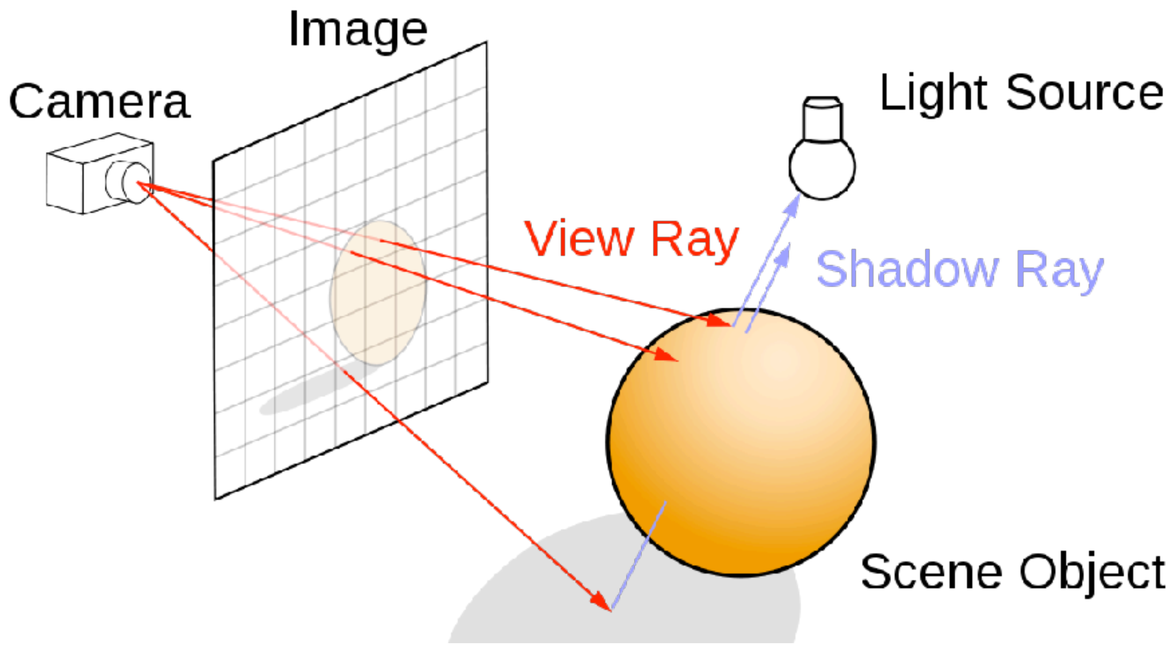


Without bias



With bias

Ray tracing



For each sample (*x*, *y*):

ray = makeRay(*camera*, *x*, *y*)

hit = castRay(*ray*, *scene*)

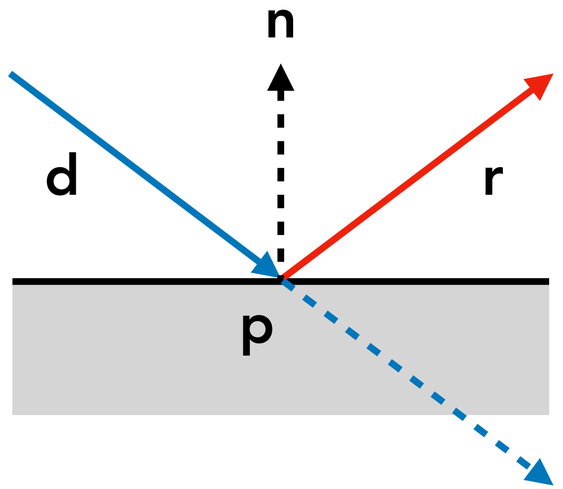
color = shade(*hit*, *scene*)

color = traceRay(*ray*, *scene*)

image[*x*, *y*] = *color*

Reflection

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



Perfect mirror:

$$L = \text{traceRay}(\mathbf{p}, \mathbf{r}, \textit{scene})$$

Reflective surface:

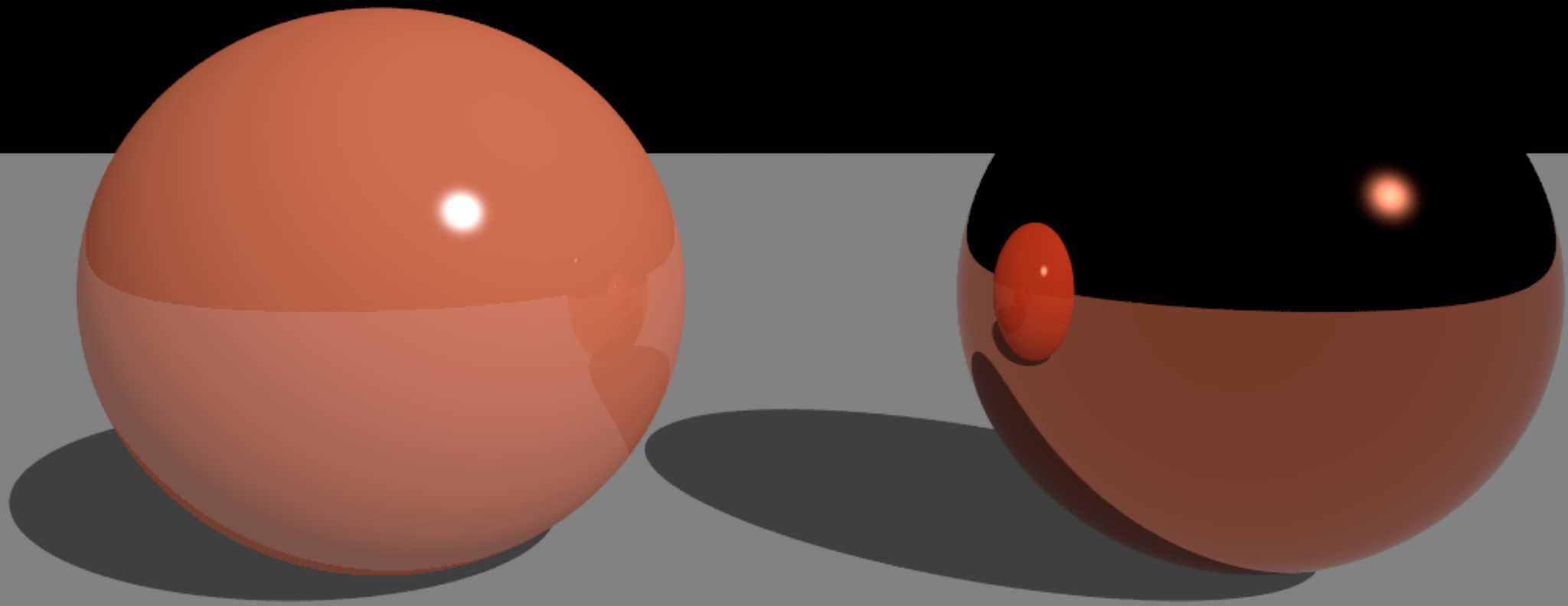
Actually, k_r depends on n · d for many materials…

$$L = [\ldots\text{Blinn-Phong}\ldots] + k_r\,\text{traceRay}(\mathbf{p}, \mathbf{r}, \textit{scene})$$

Again, don’t forget ray bias!

k_a, k_d coloured
k_s, k_r colourless

k_a, k_d zero
k_s, k_r coloured



Plastic

Metallic

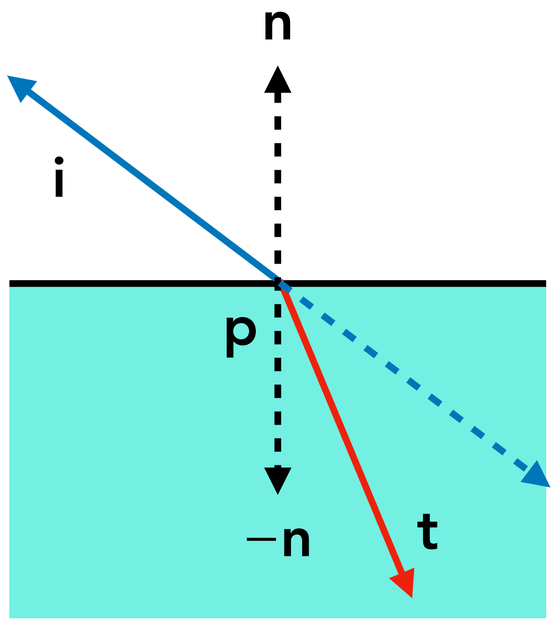
Transmission

Suppose ray passes from one material to another with different indices of refraction.

Snell's law: $\eta_i \sin \theta_i = \eta_t \sin \theta_t$

$$\mathbf{t} = \left(\eta_r (\mathbf{n} \cdot \mathbf{i}) - \sqrt{1 - \eta_r^2 \left(1 - (\mathbf{n} \cdot \mathbf{i})^2\right)} \right) \mathbf{n} - \eta_r \mathbf{i}$$

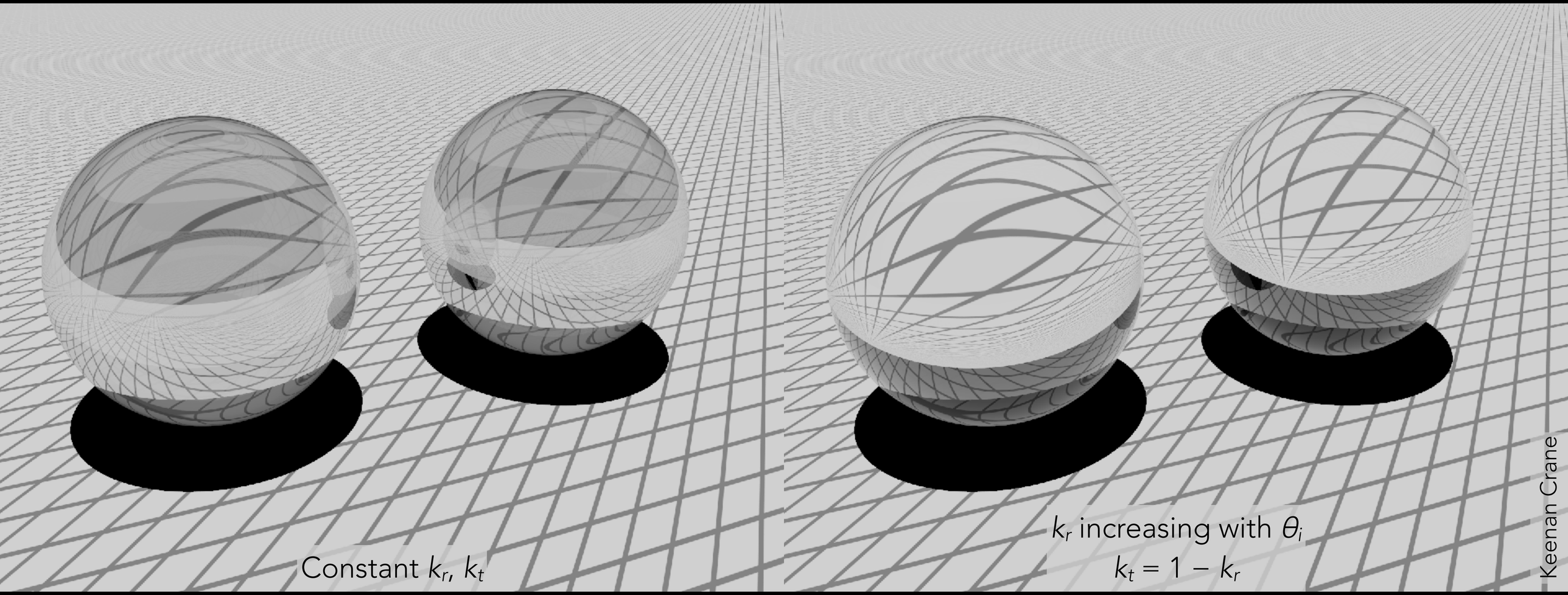
where $\eta_r = \eta_i / \eta_t$.

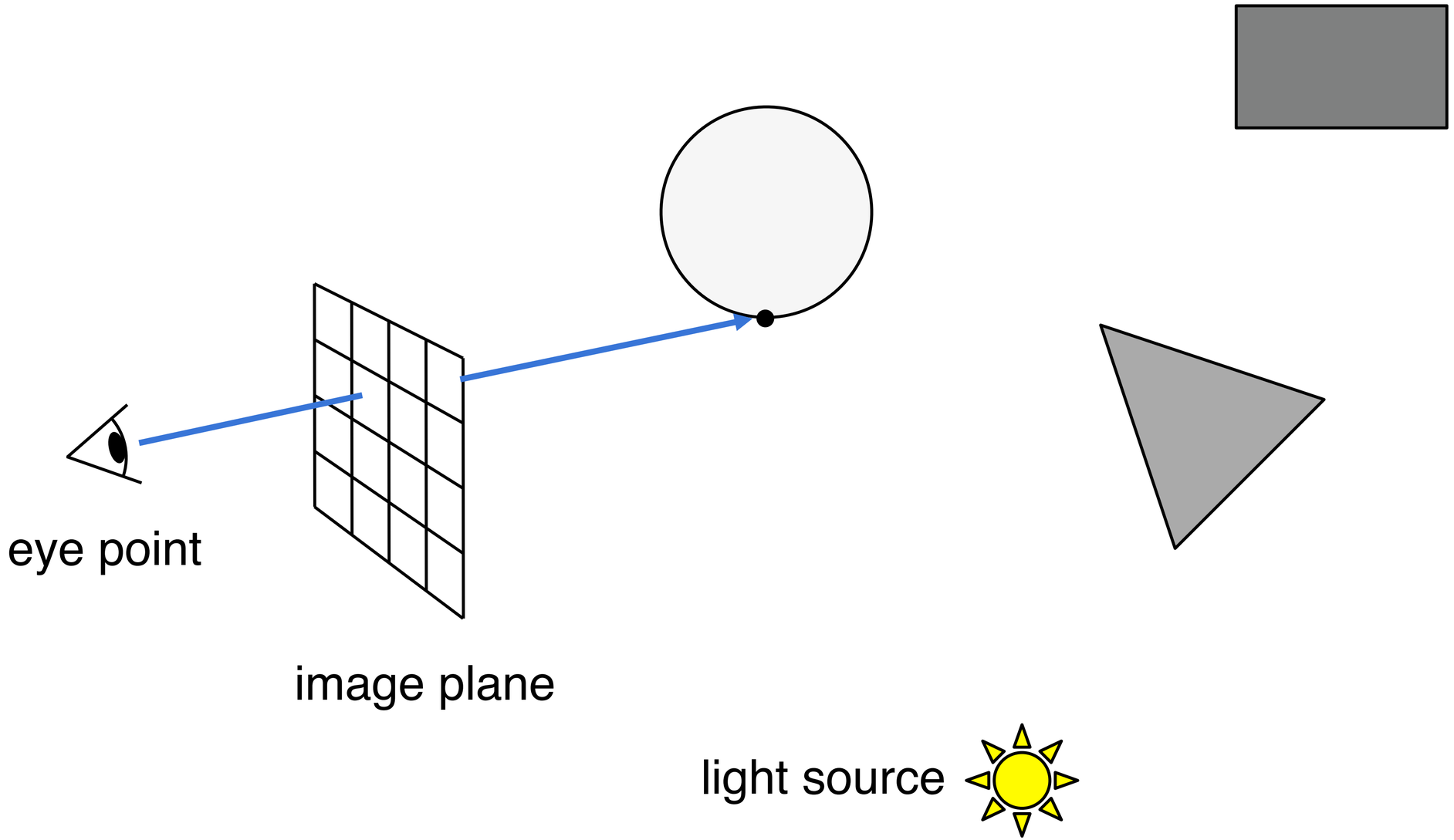


$$L = [\ldots] + k_t \, \text{traceRay}(\mathbf{p}, \mathbf{t}, \textit{scene})$$

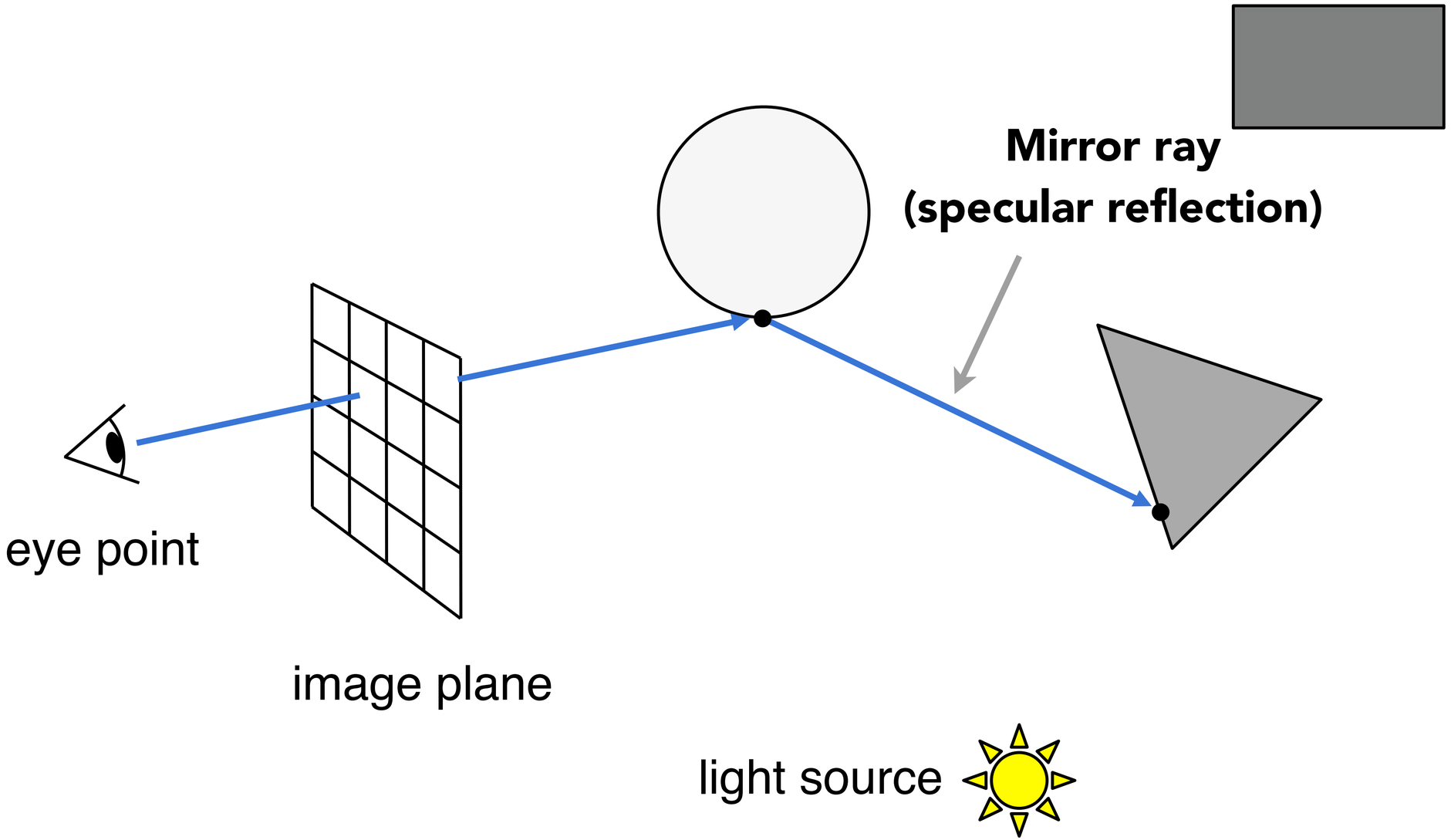
Also depends on n · d

Fresnel effect

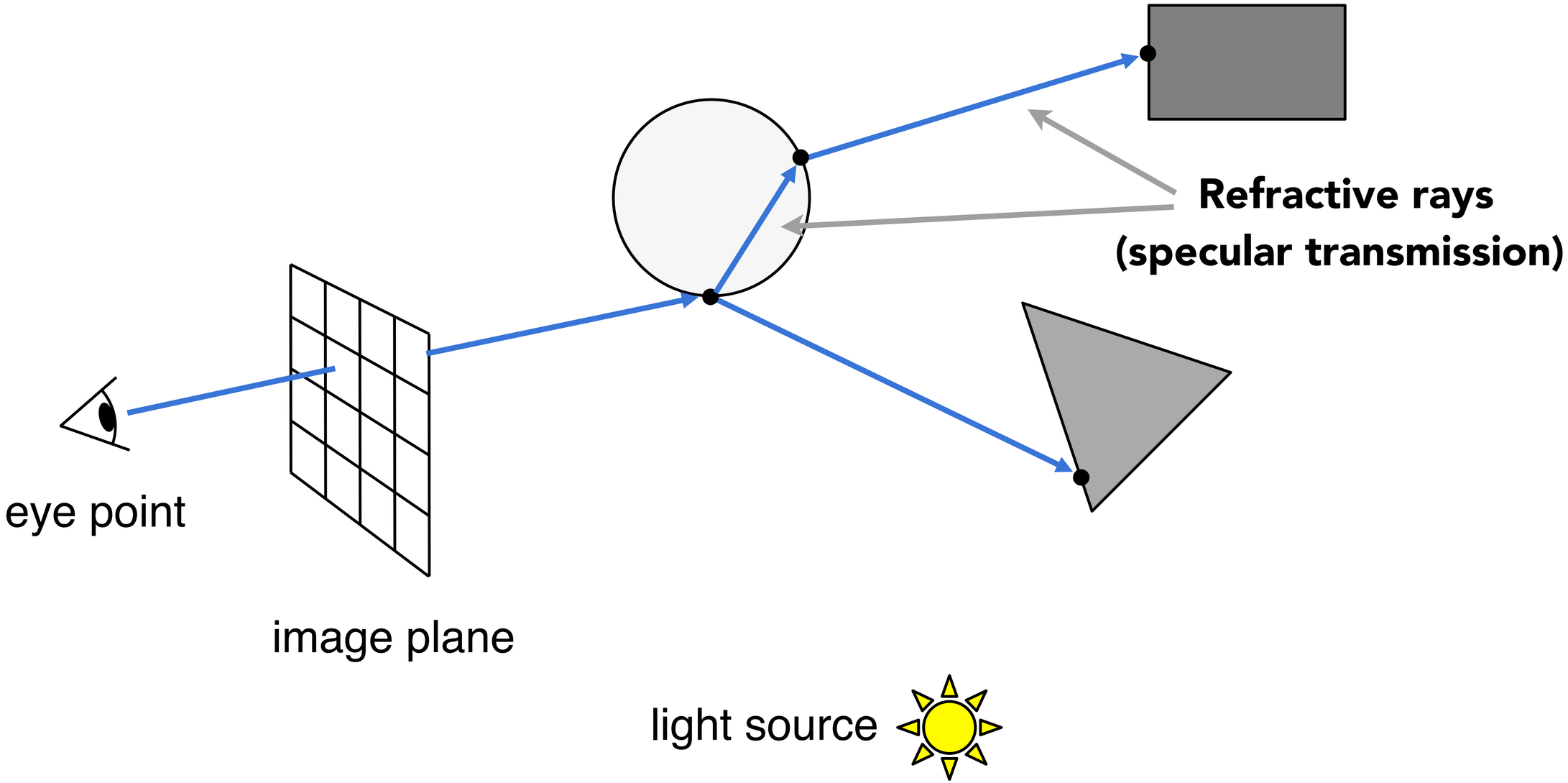

Constant k_r, k_t
k_r increasing with θ_i
$k_t = 1 - k_r$
Keenan Crane



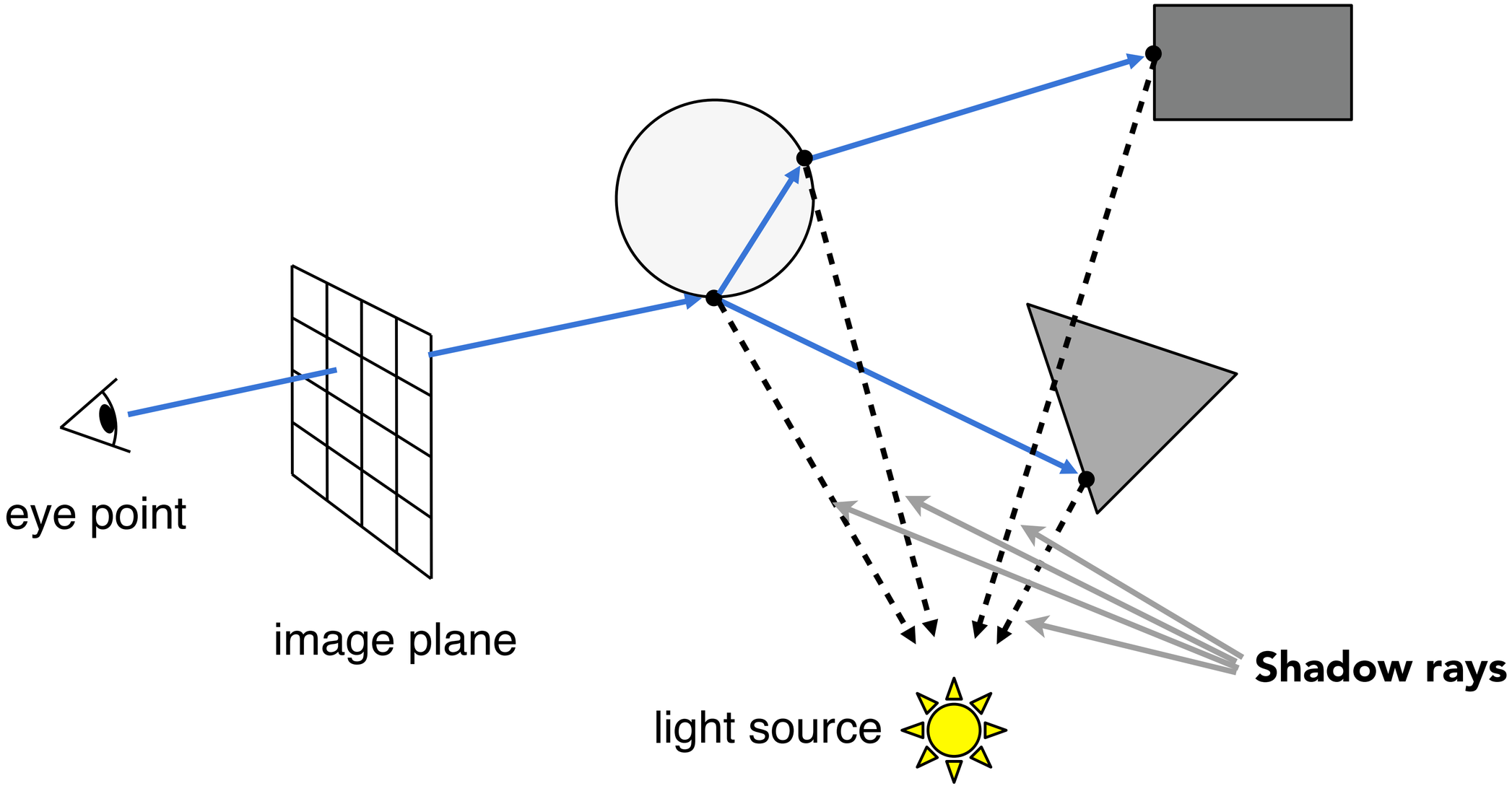
Kanazawa and Ng



Kanazawa and Ng

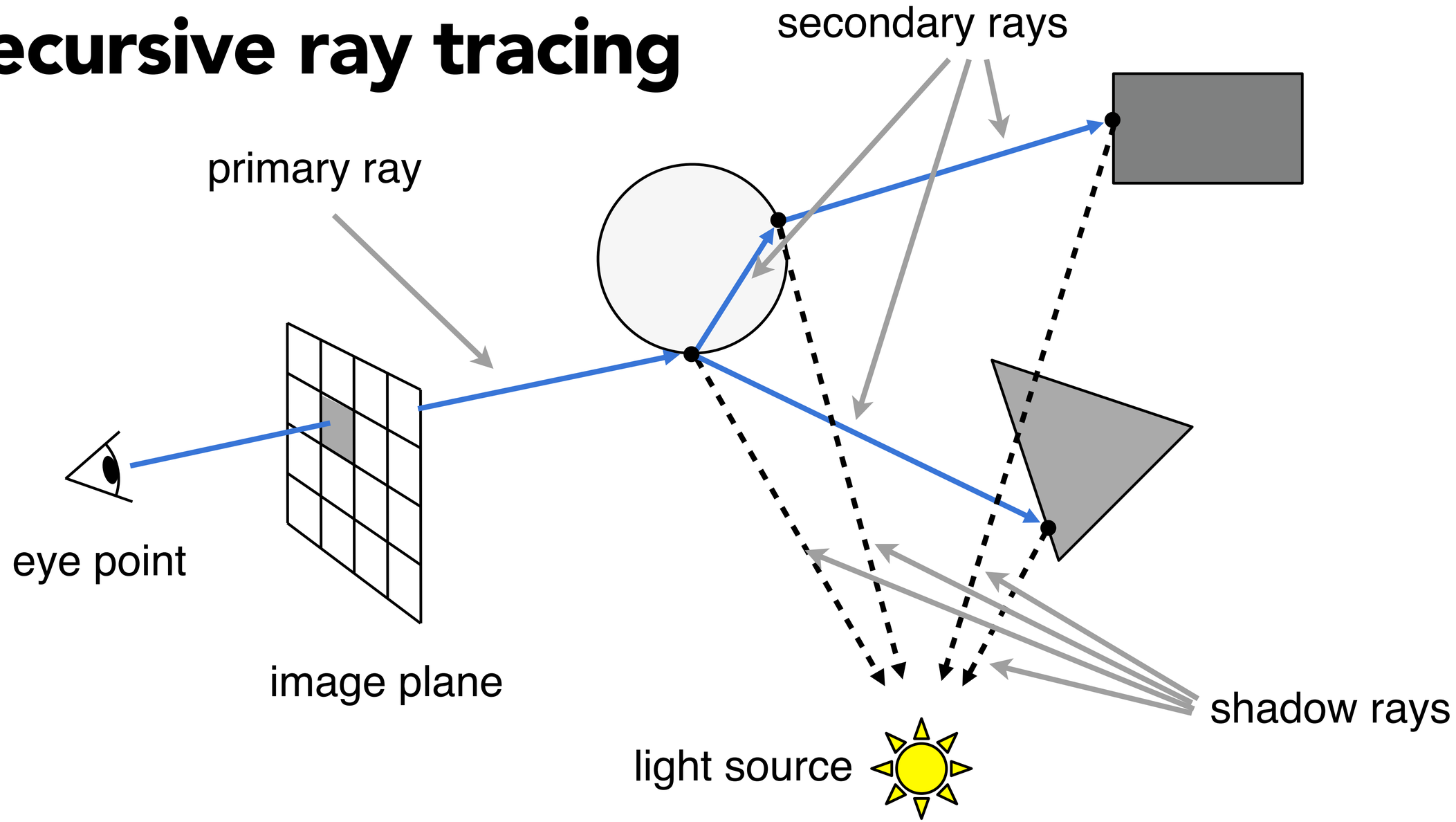


Kanazawa and Ng

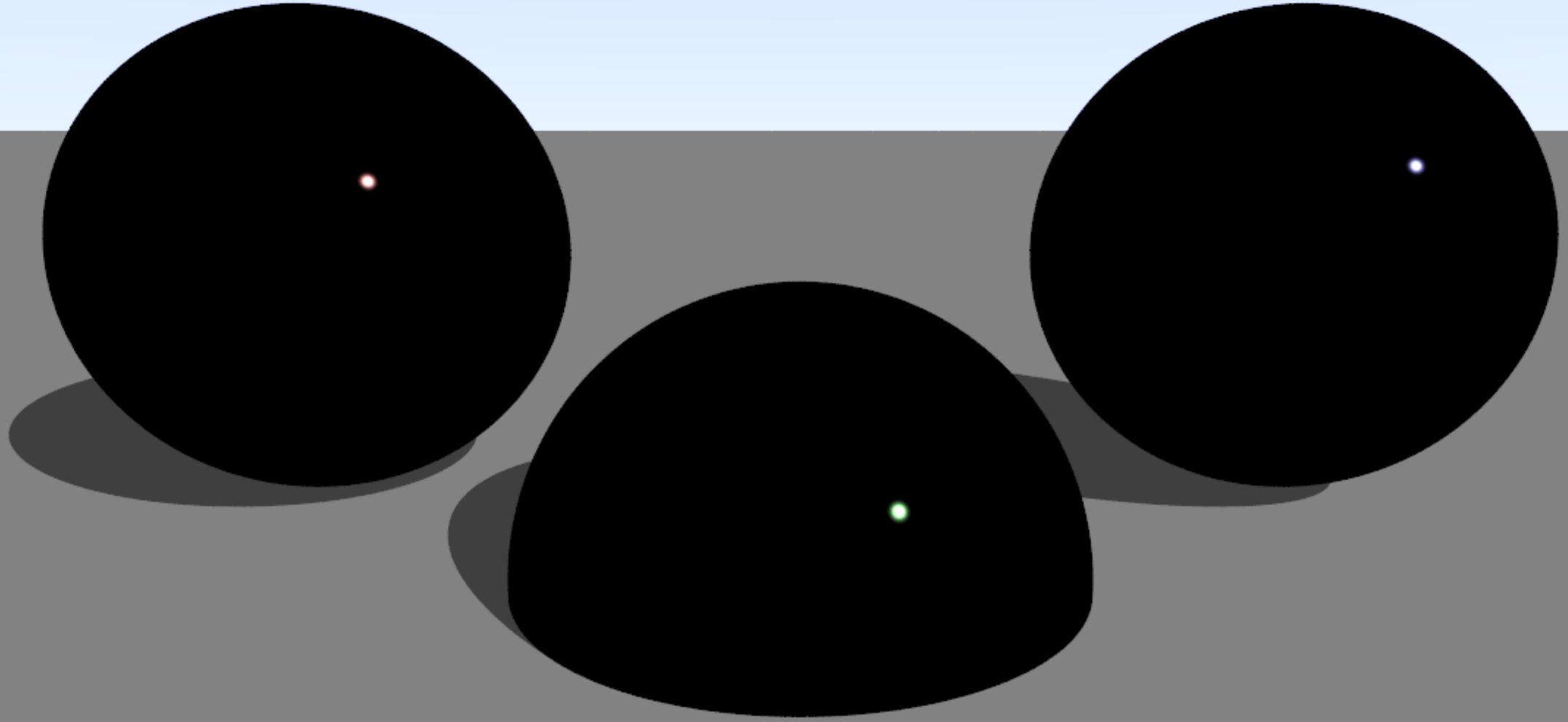


Kanazawa and Ng

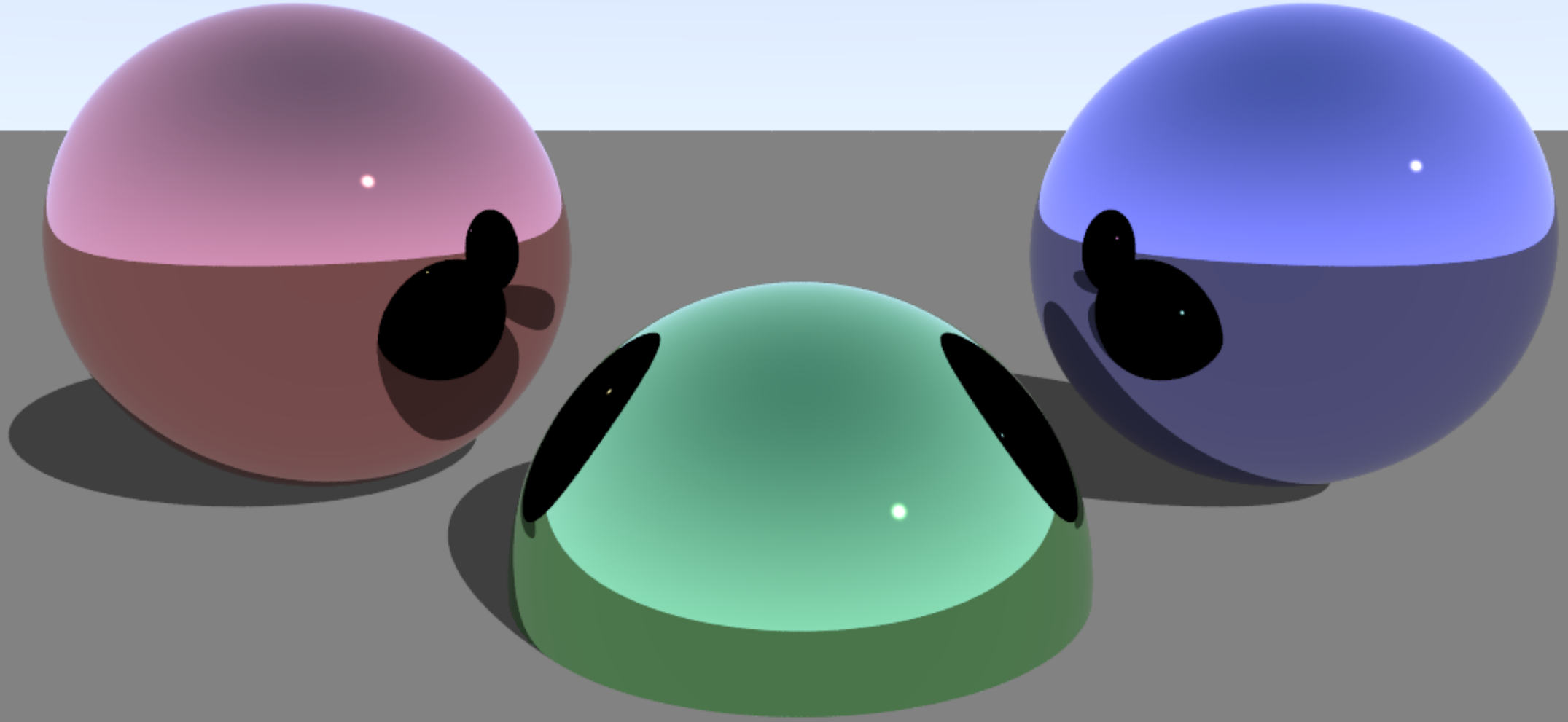
Recursive ray tracing



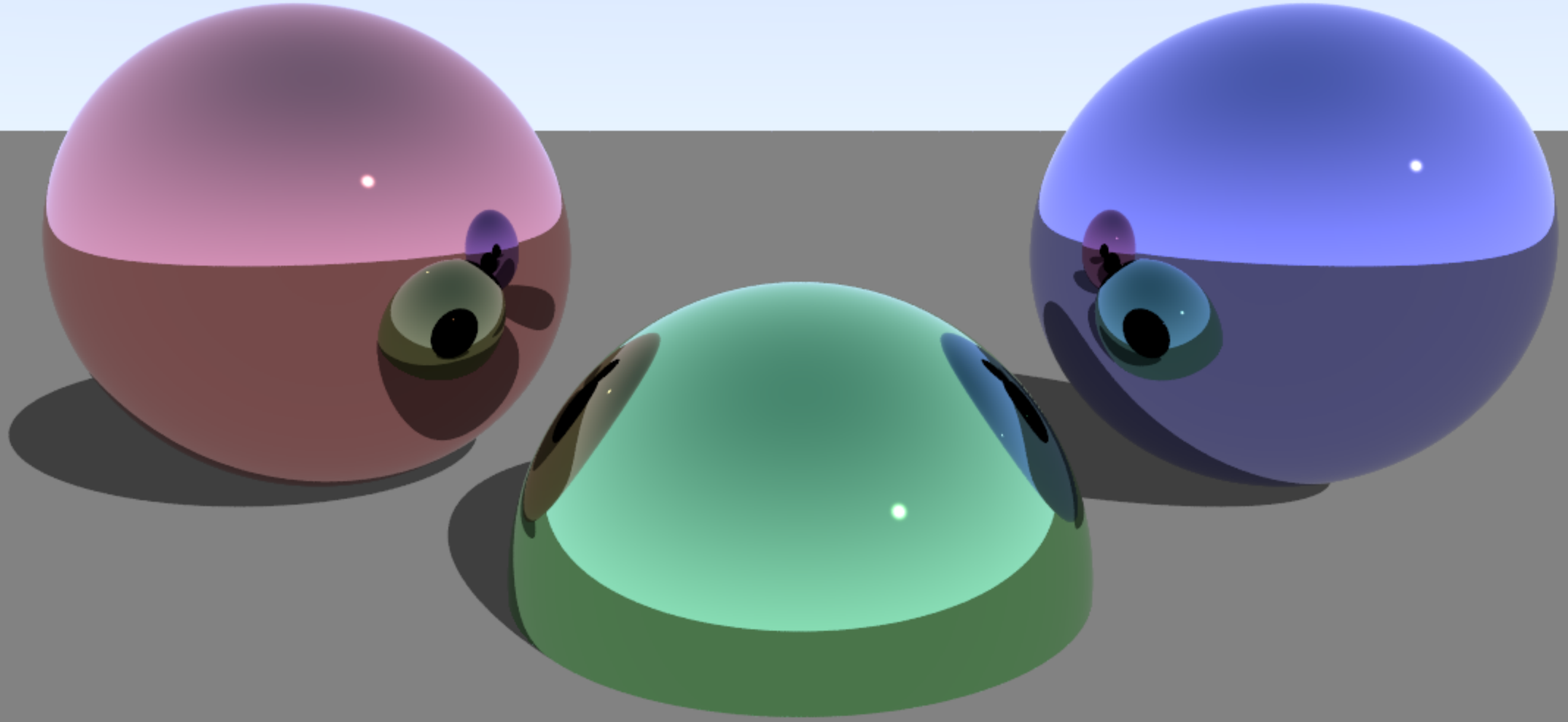
Kanazawa and Ng



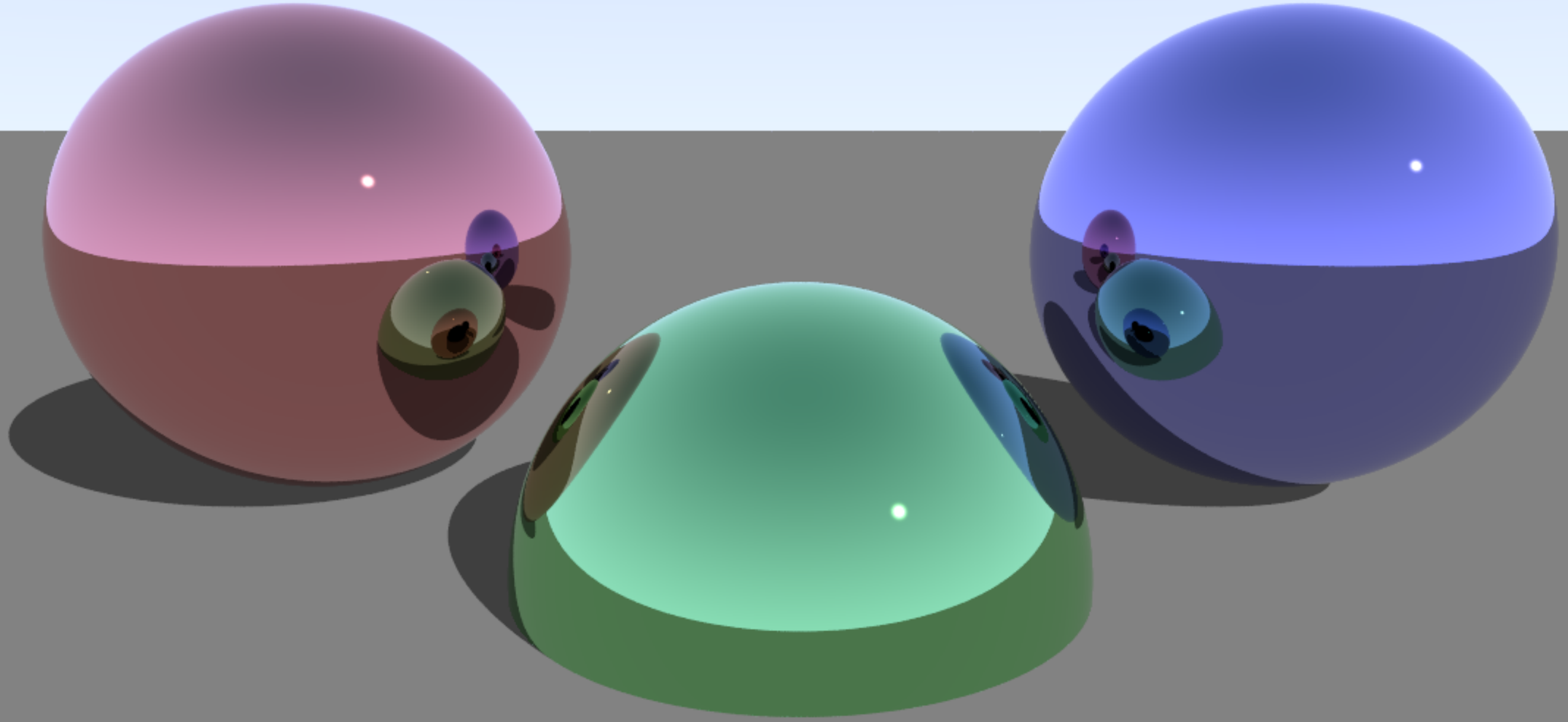
Recursion depth: 1



Recursion depth: 2



Recursion depth: 3

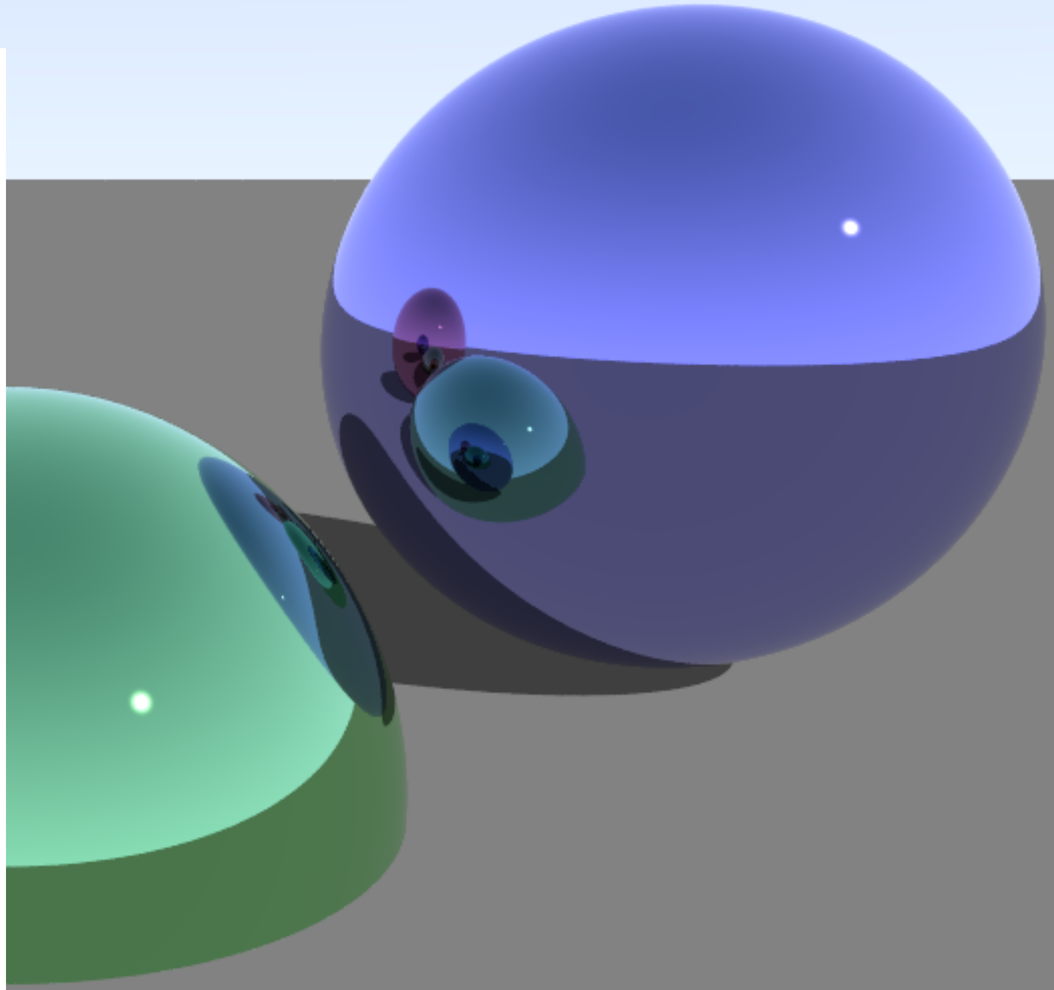


Recursion depth: 4

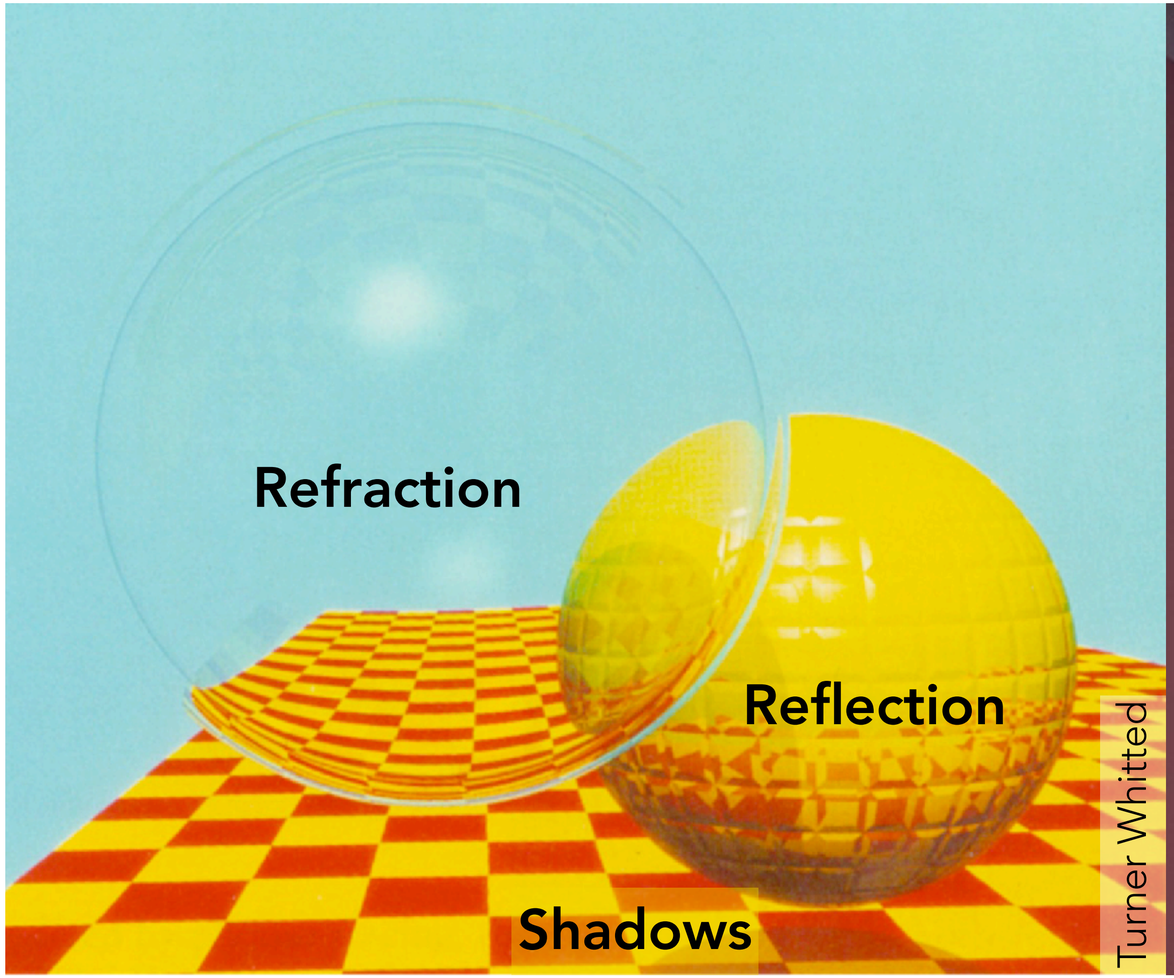
When to terminate the recursion?

- **Fixed:** stop if recursion depth > max
- **Adaptive:** stop if contribution of ray to final pixel colour < threshold
- …or whichever comes first

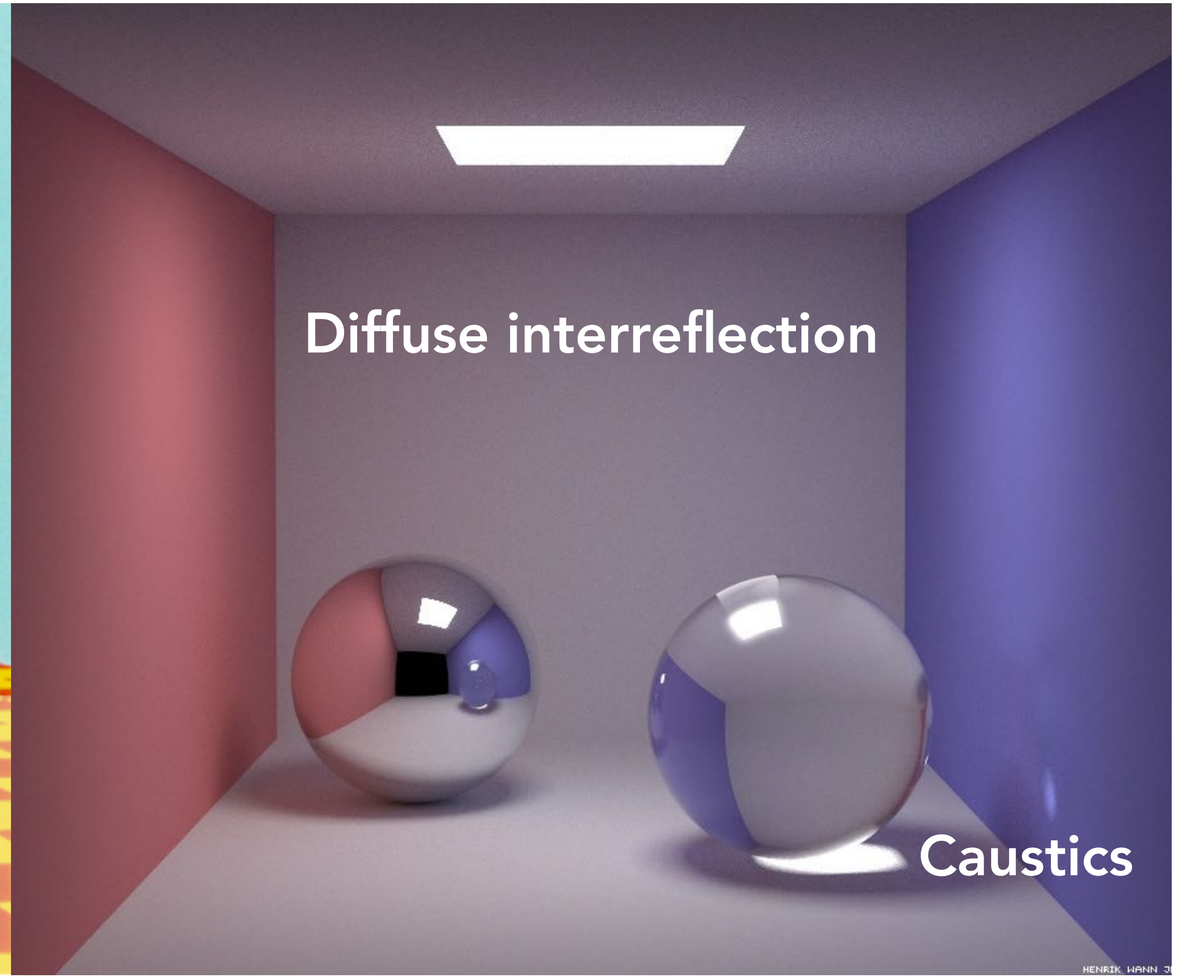
On termination, return… diffuse colour? background colour? black?



What ray tracing can and can’t do

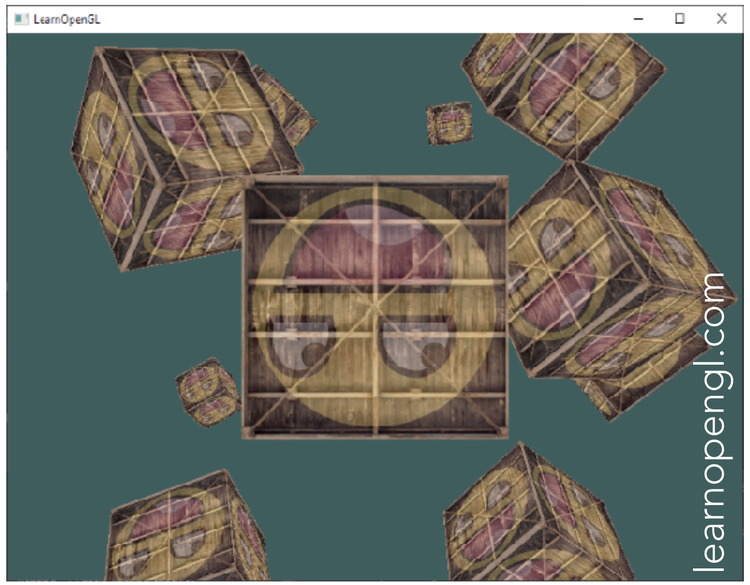


Turner Whitted



Henrik Wann Jensen

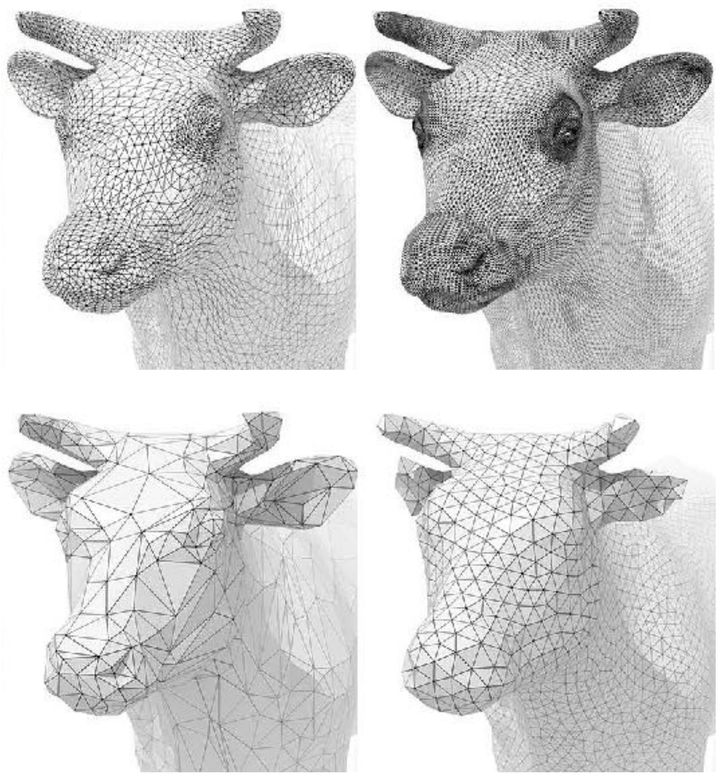
So far, we have learned how to make crude pictures of polygonal shapes.



How would we make photorealistic movies of complicated shapes?

RENDERING **ANIMATION** **MODELING**

Tour of deeper aspects of computer graphics

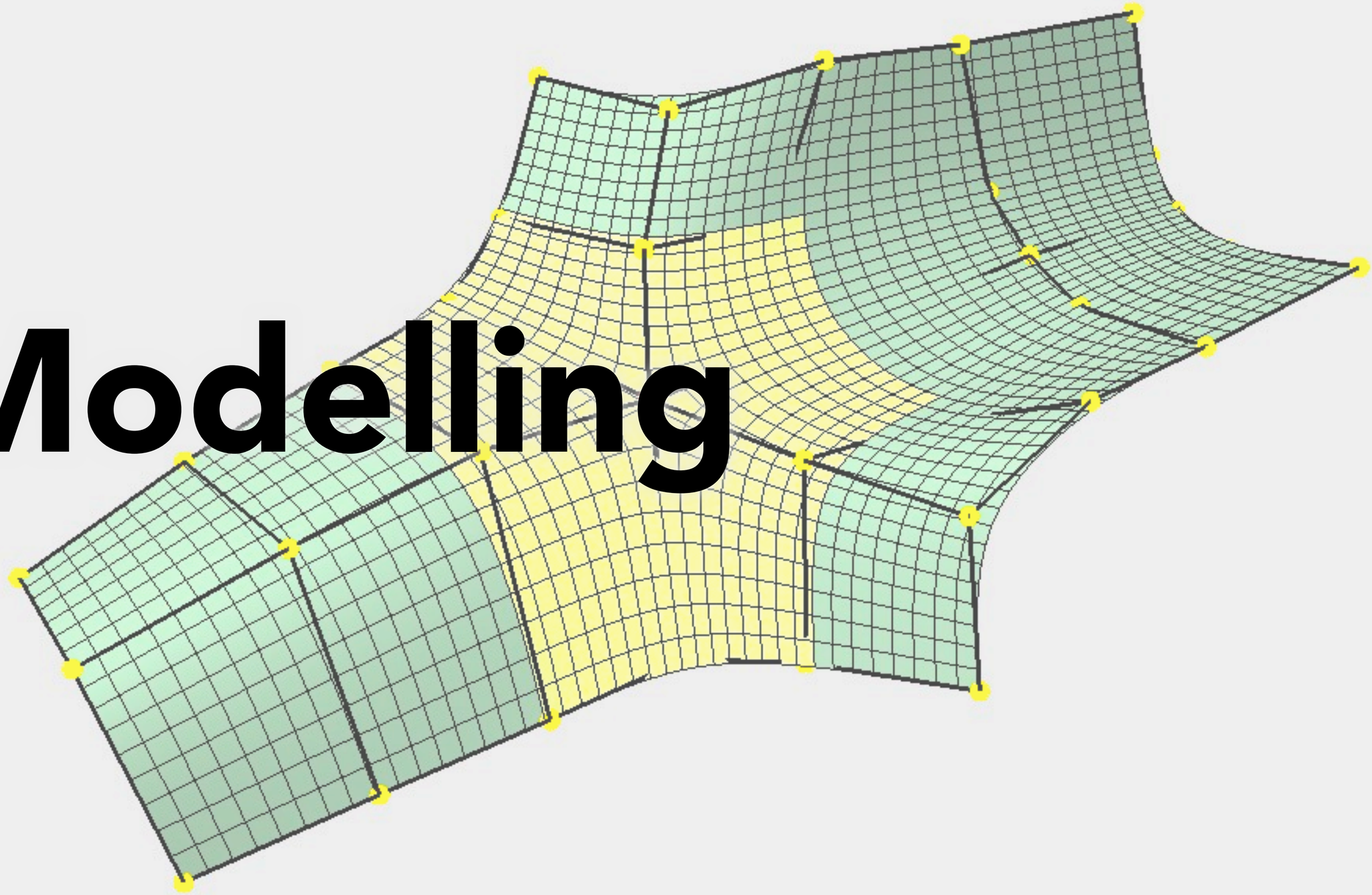


Modelling

Rendering

Animation

Modelling

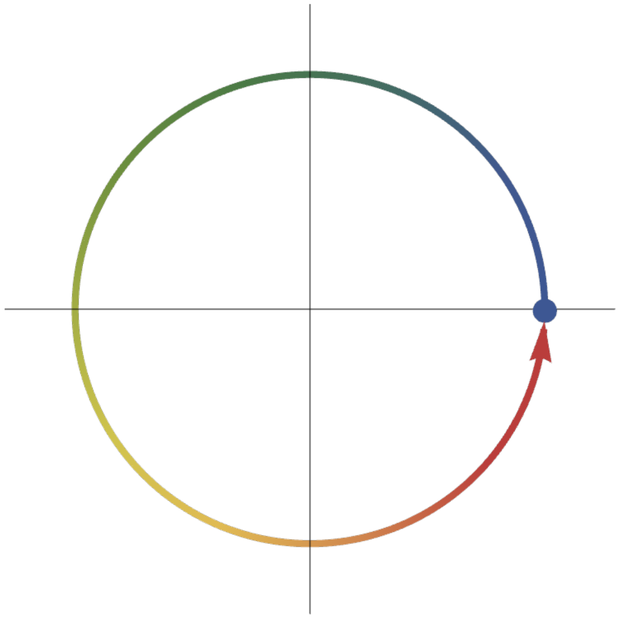


Pixar

How to define a unit circle in 2D?

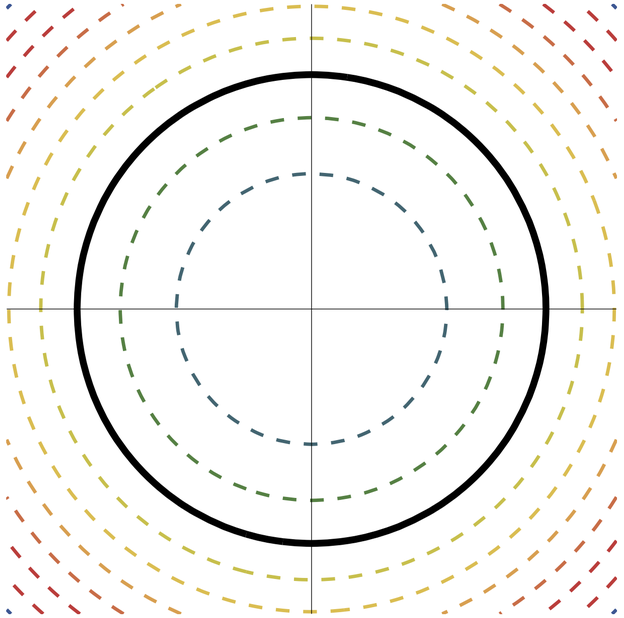
Explicit:

$$\{(\cos\theta, \sin\theta): 0 \leq \theta < 2\pi\}$$



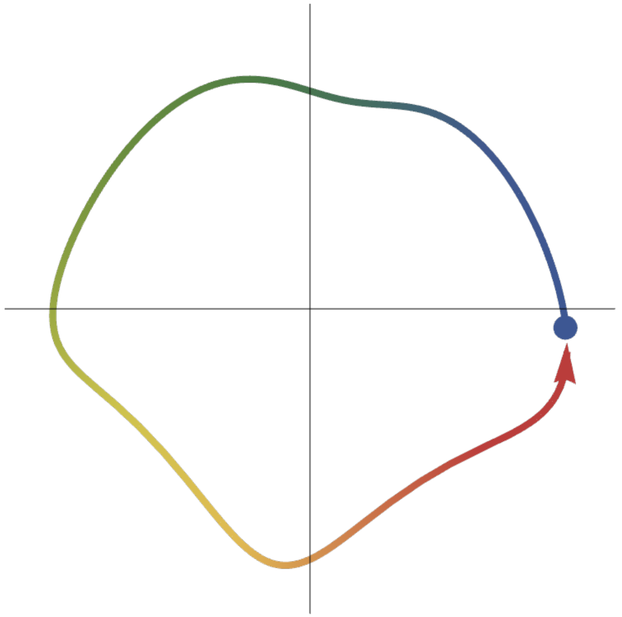
Implicit:

$$\{(x, y): x^2 + y^2 - 1 = 0\}$$



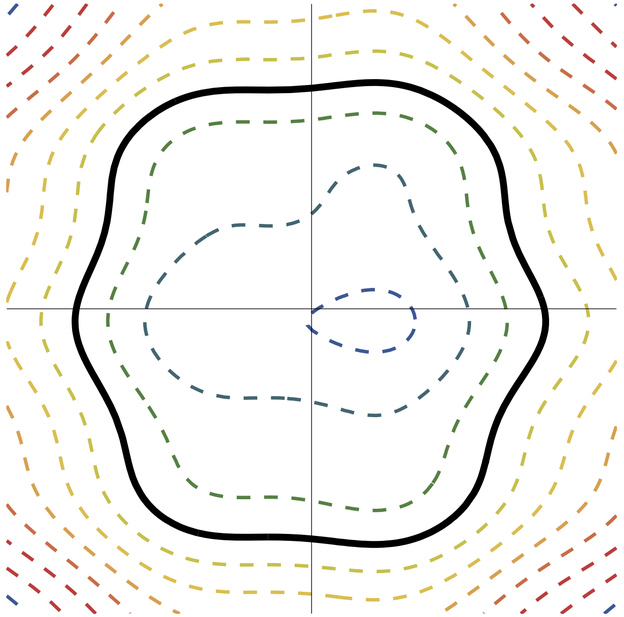
Explicit:

$\{(x(t), y(t)): t \in [a, b]\}$



Implicit:

$\{(x, y): f(x, y) = 0\}$

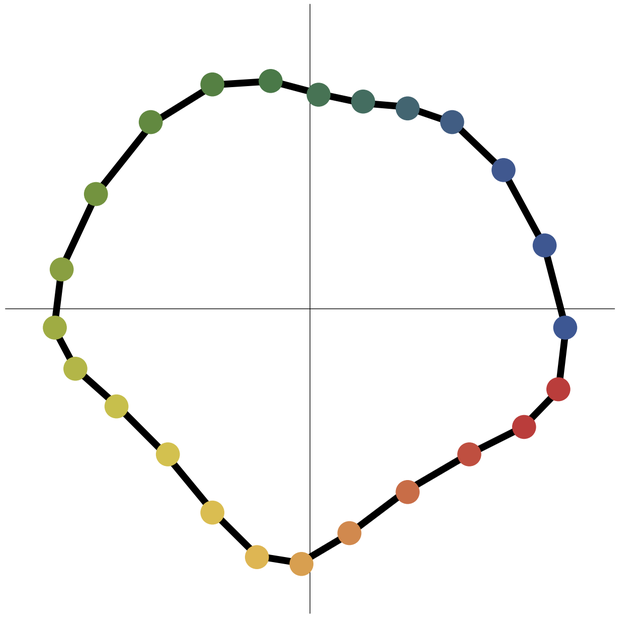


When is it easy to **generate** an arbitrary point on the curve?

When is it easy to **test** if a given point lies on the curve?

How to **draw** a curve given in one of these forms?

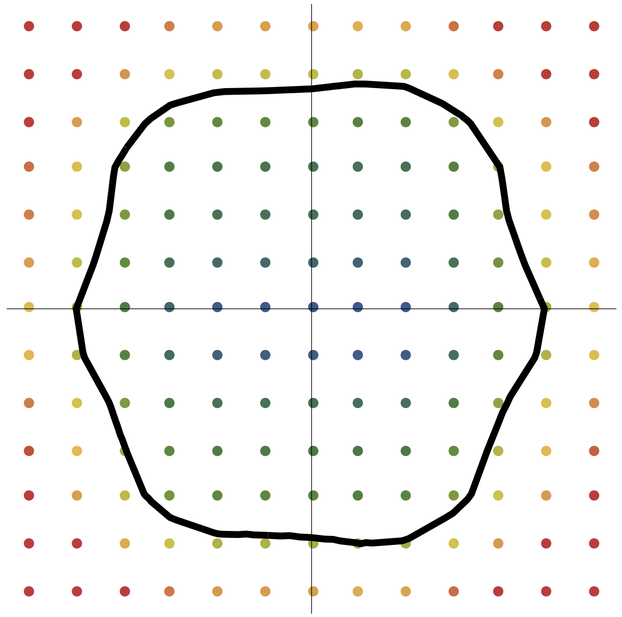
$\{(x(t), y(t)): t \in [a, b]\}$



Sample points at various values of t

Connect by polyline

$\{(x, y): f(x, y) = 0\}$



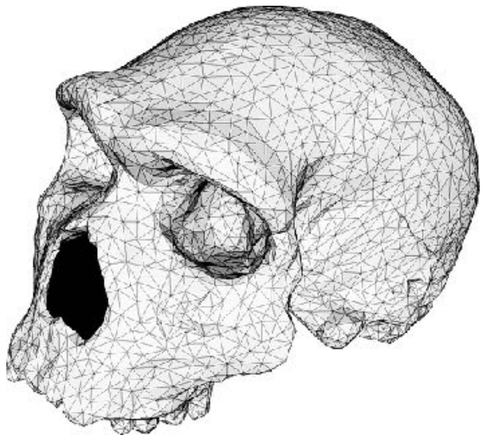
Sample f at various points (x, y)

Draw boundary between + and − points

Representing geometry in 3D

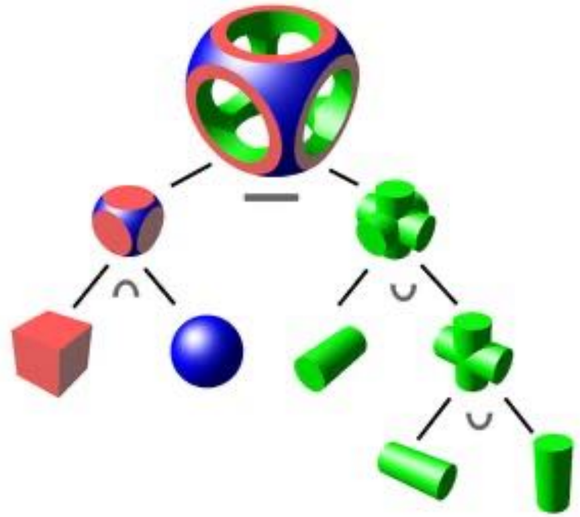
Explicit:

- Polygon meshes
- Parametric curves and surfaces
- Subdivision surfaces
- Point clouds

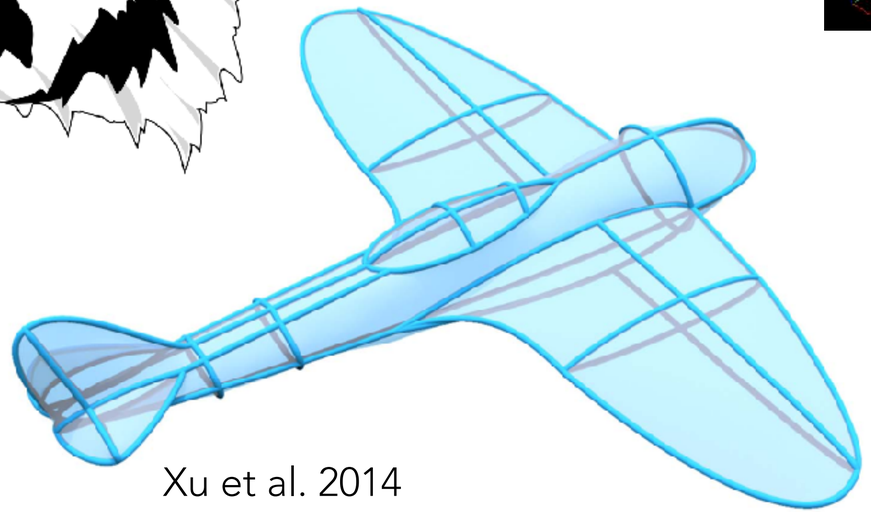


Implicit:

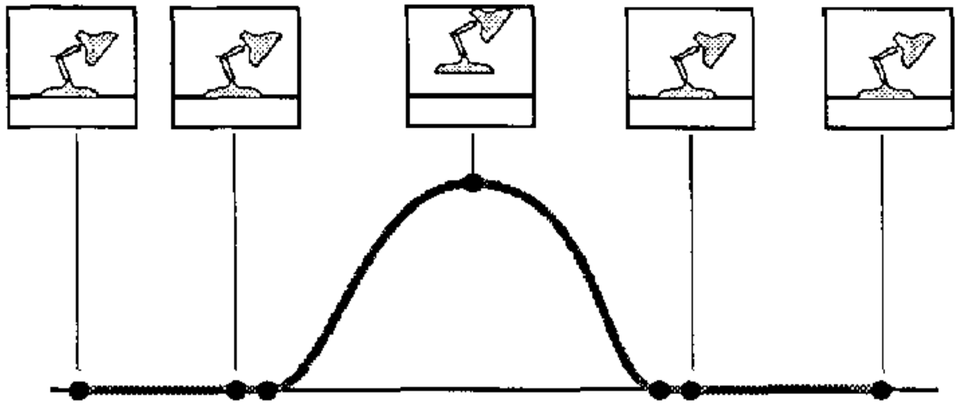
- Algebraic surfaces, distance fields
- Constructive solid geometry
- “Blobby” surfaces
- Level sets



Curves

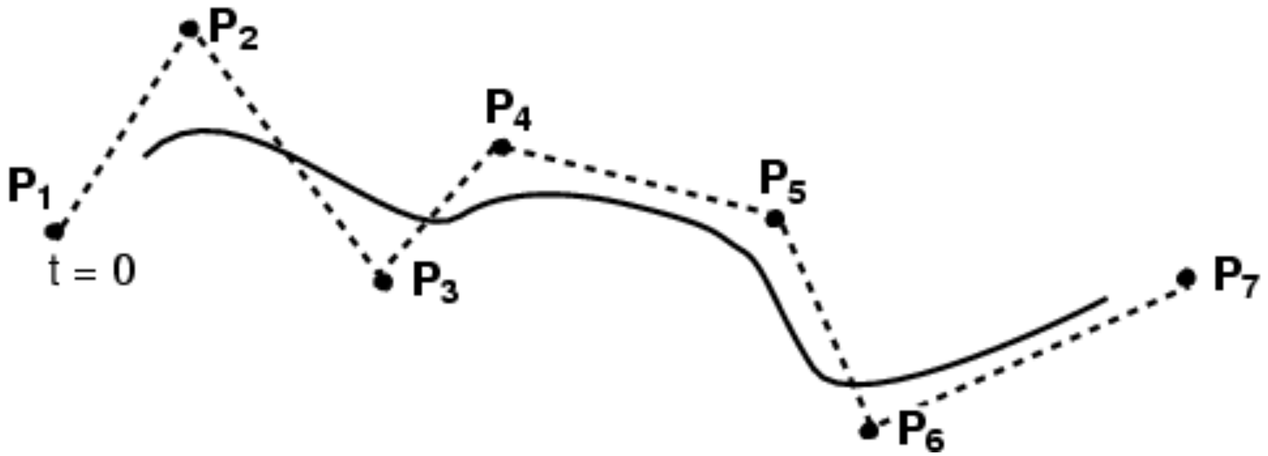


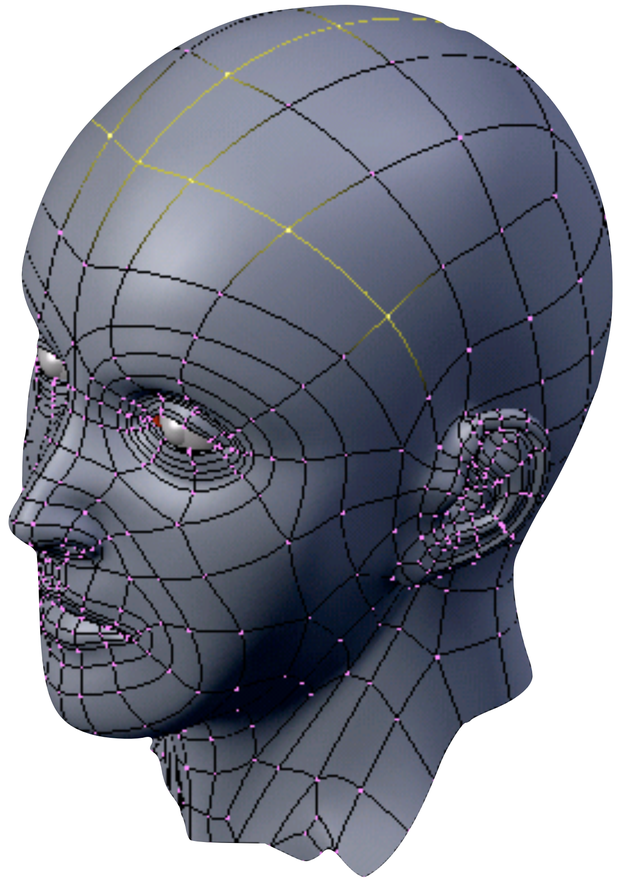
Xu et al. 2014



Lasseter 1987

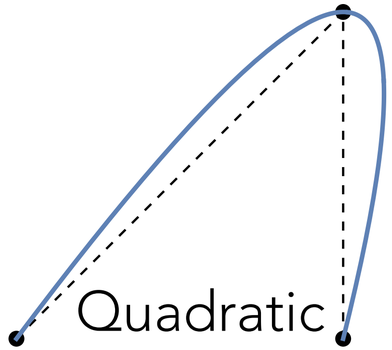
We will retain the same “interface” as polylines: user specifies a sequence of points. Now we want to define a smooth curve based on them.

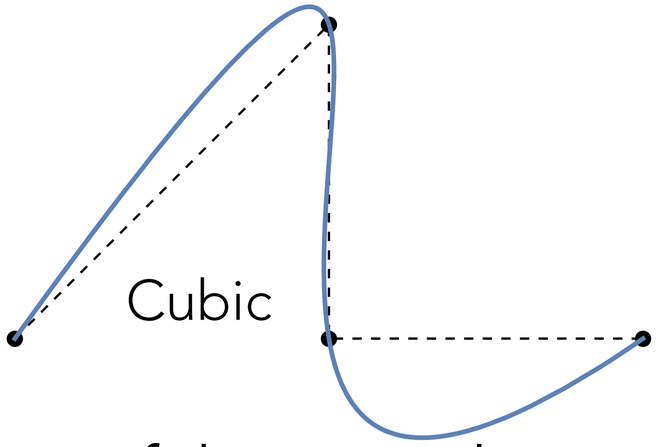




Usually define parametrically: $x(t)$, $y(t)$ where
x, y are piecewise polynomial functions a.k.a. **splines**

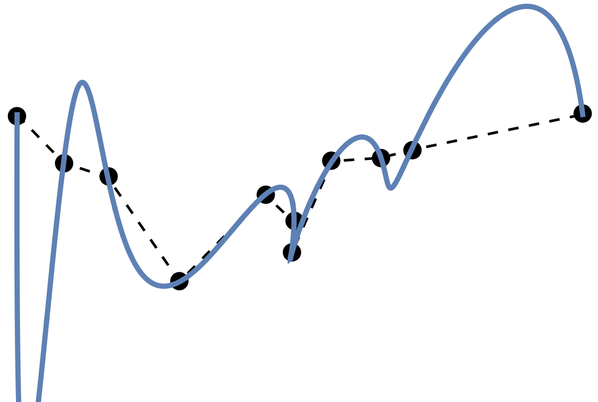
You all probably already know one way to fit a smooth function through multiple points: **polynomial interpolation**.





Hard to control: curve goes beyond the range of the control points

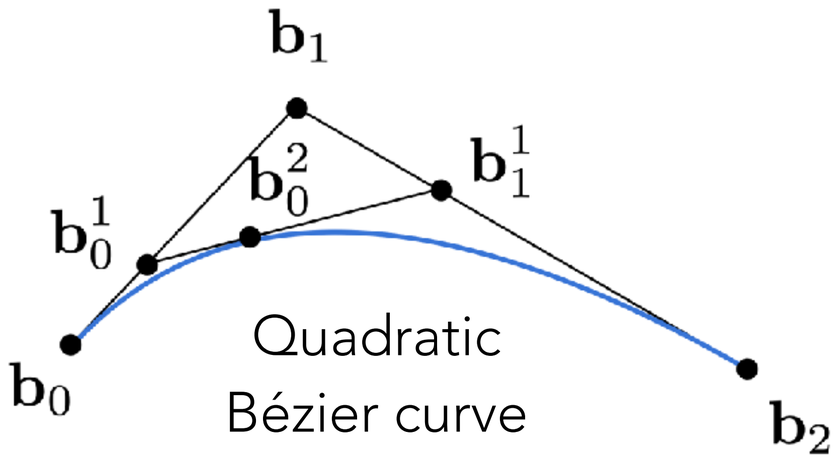
Very unstable for higher degrees!



Bézier curves

How can we guarantee the curve stays within the range of the control points?

Construct the curve by recursive interpolation: **de Casteljau's algorithm** a.k.a. "**corner cutting**"



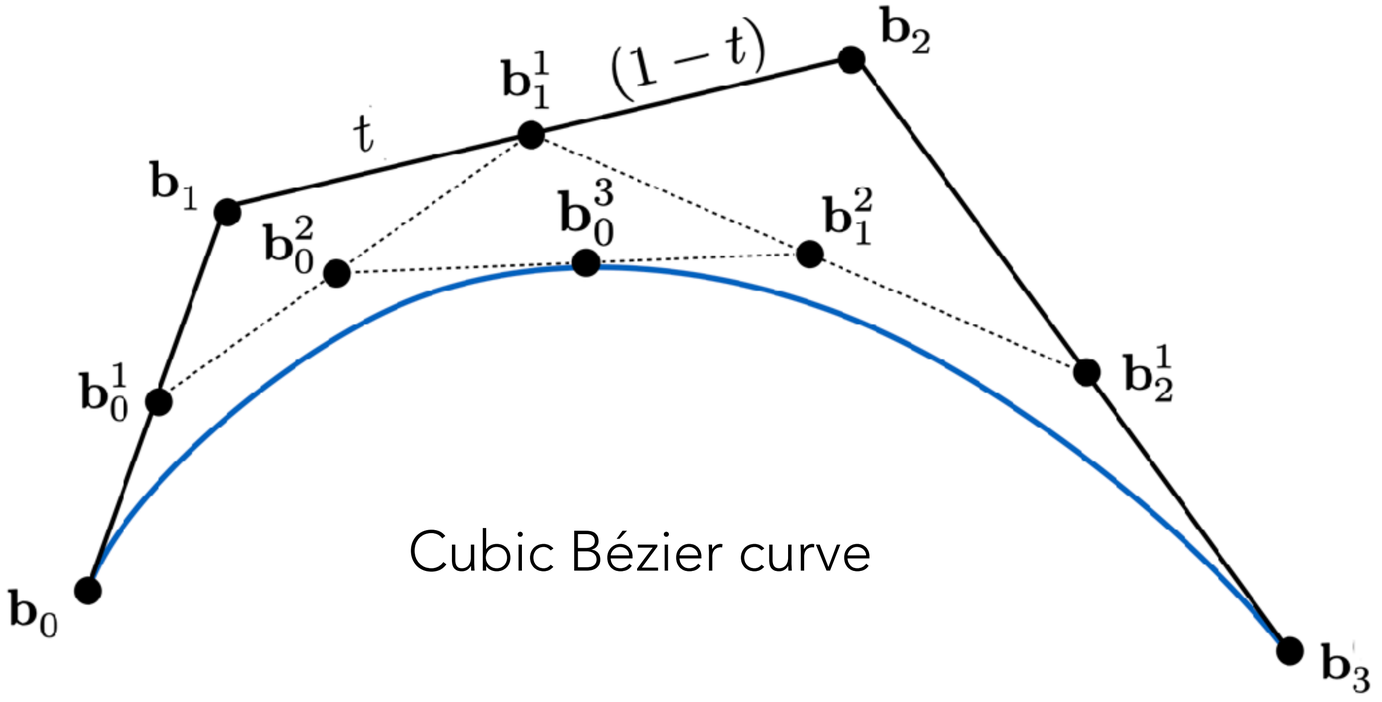
$$\mathbf{b}_0^1 = \text{lerp}(t, \mathbf{b}_0, \mathbf{b}_1)$$
$$\mathbf{b}_1^1 = \text{lerp}(t, \mathbf{b}_1, \mathbf{b}_2)$$

$$\mathbf{b}_0^2 = \text{lerp}(t, \mathbf{b}_0^1, \mathbf{b}_1^1)$$

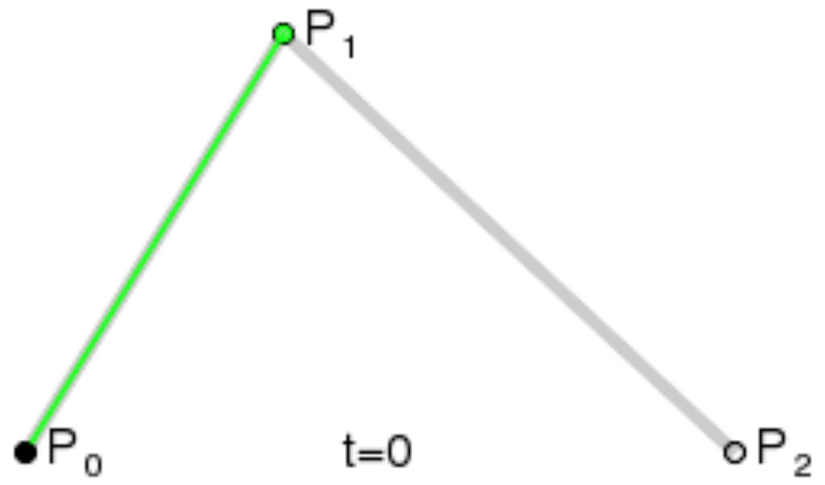
$$\mathbf{b}_0^1 = \text{lerp}(t, \mathbf{b}_0, \mathbf{b}_1)$$
$$\mathbf{b}_1^1 = \text{lerp}(t, \mathbf{b}_1, \mathbf{b}_2)$$
$$\mathbf{b}_2^1 = \text{lerp}(t, \mathbf{b}_2, \mathbf{b}_3)$$

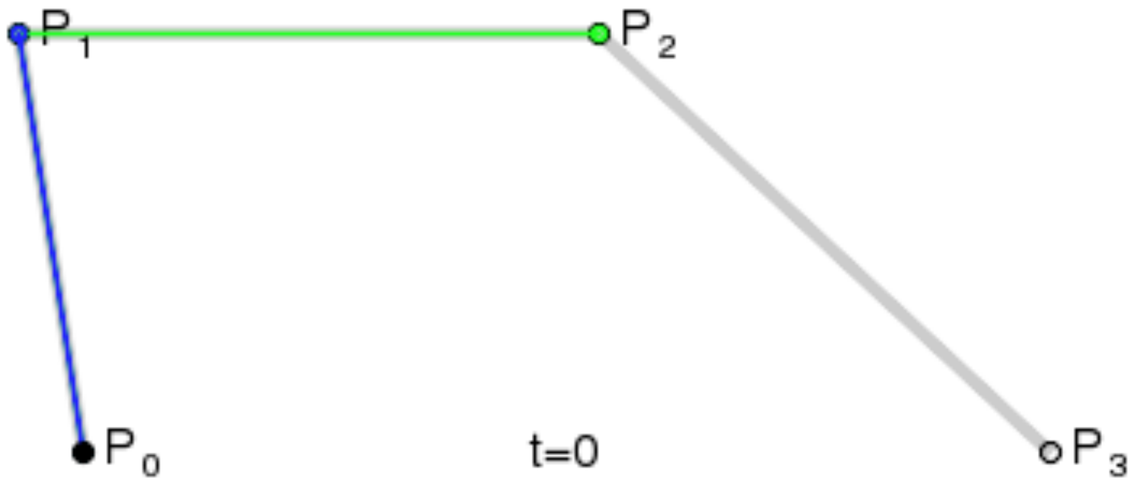
$$\mathbf{b}_0^2 = \text{lerp}(t, \mathbf{b}_0^1, \mathbf{b}_1^1)$$
$$\mathbf{b}_1^2 = \text{lerp}(t, \mathbf{b}_1^1, \mathbf{b}_2^1)$$

$$\mathbf{b}_0^3 = \text{lerp}(t, \mathbf{b}_0^2, \mathbf{b}_1^2)$$

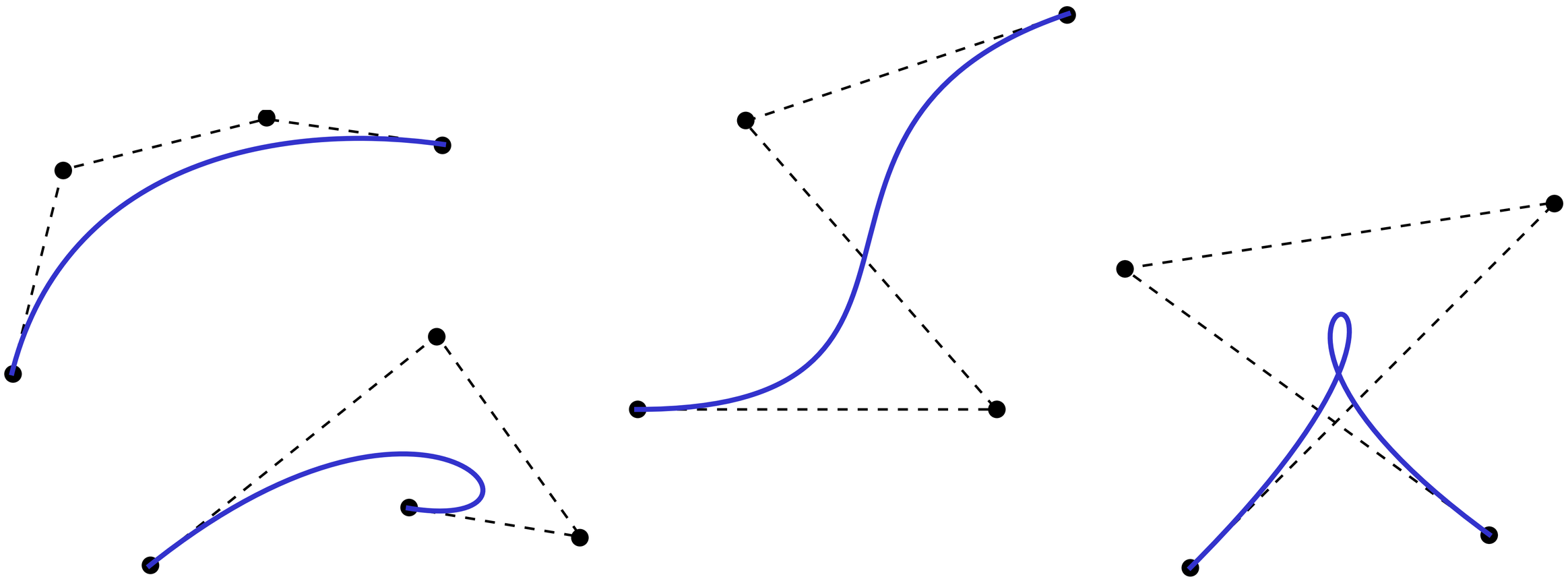


Cubic Bézier curve


P1
P0
t=0
P2

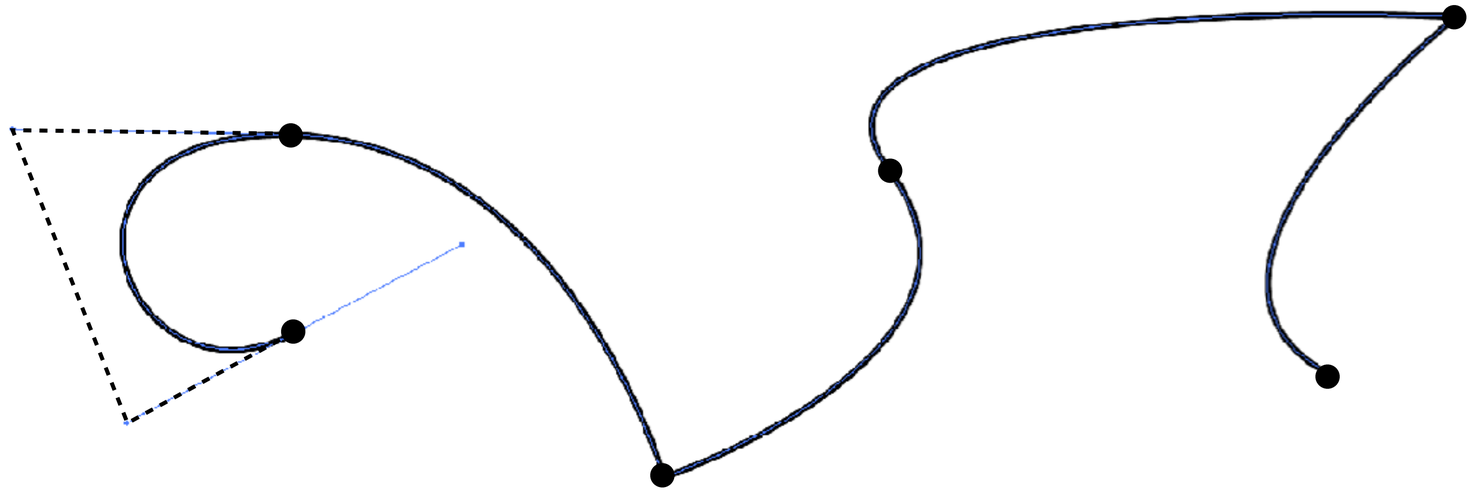

P1
P2
P0
t=0
P3

No longer interpolation but **approximation**: Curve is influenced by the control points but does not pass through them



Piecewise Bézier curves (Bézier splines)

Chain together multiple Bézier curves of low degree (usually cubic)



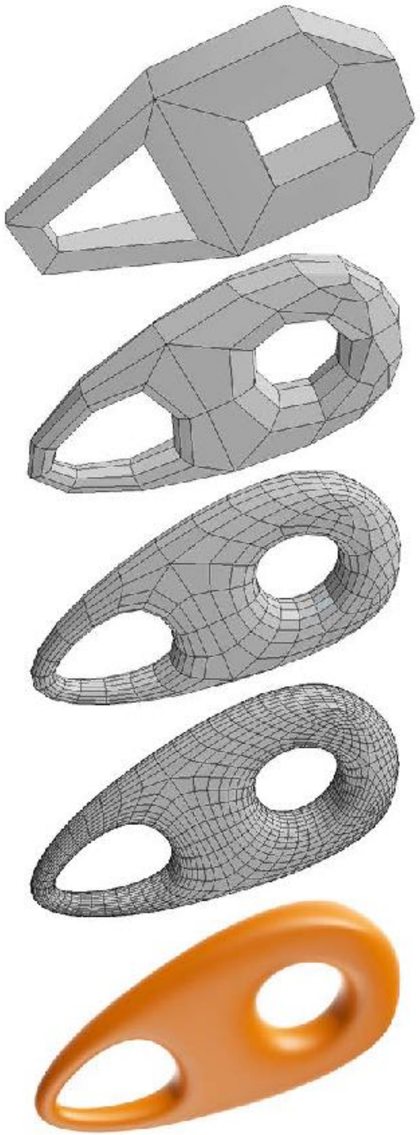
Now we have local control: each control point only affects one or two segments

Used basically everywhere (fonts, paths, Illustrator, PowerPoint, …)

Another strategy to create smooth shapes from a coarse mesh of control points: **subdivision**

- Split each element by inserting new vertices
- Update positions of all vertices by local averaging
- Repeat…

The desired shape is what we converge to in the limit.

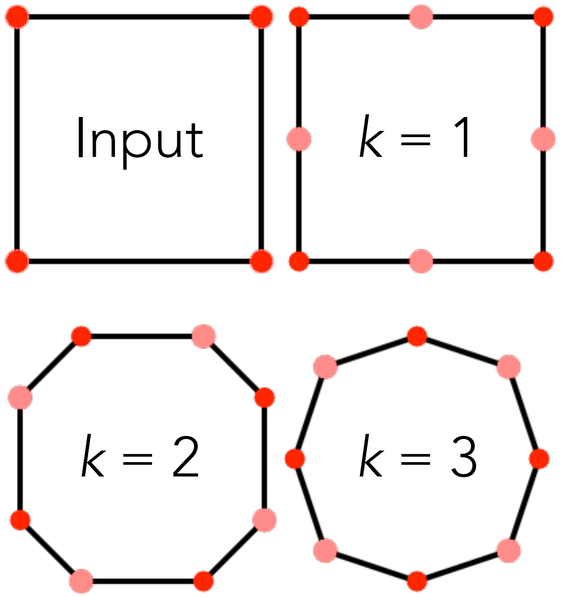


Subdivision curves

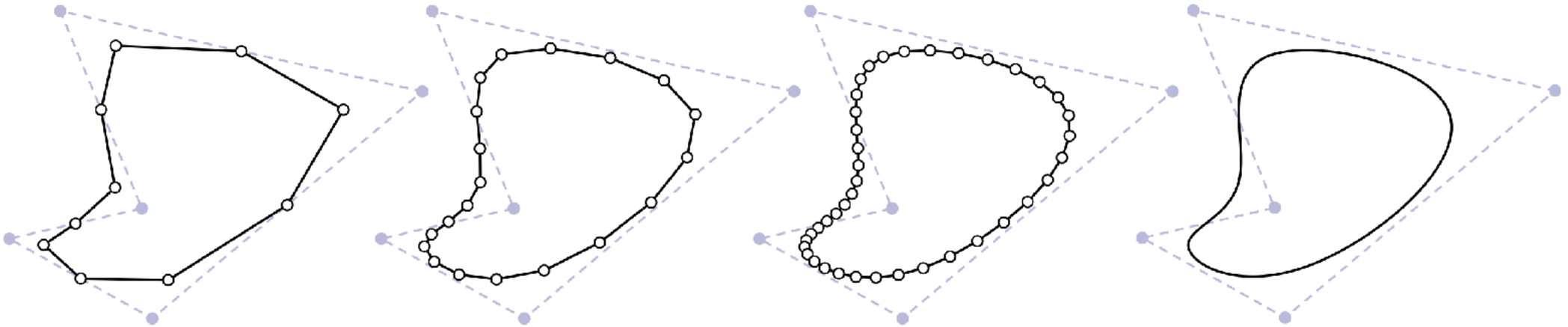
One possible method: Lane-Riesenfeld

- Insert midpoint of each edge
- Repeat $k-1$ times: Average adjacent vertices

Limit is a degree-k B-spline!



gilgamec

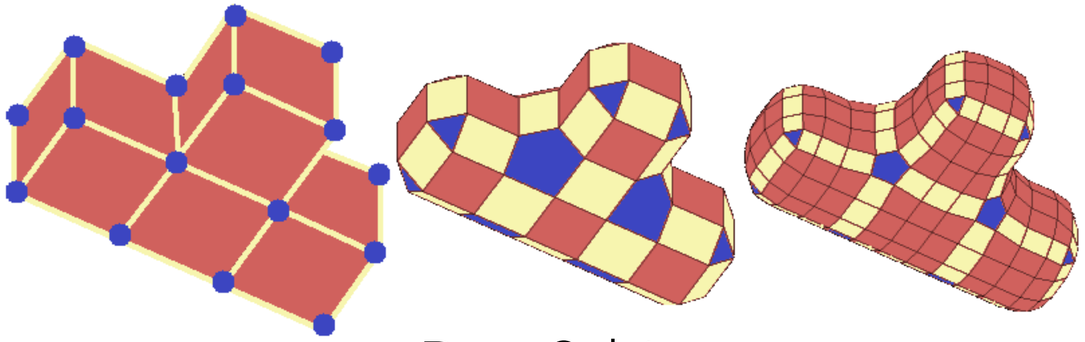


Keenan Crane

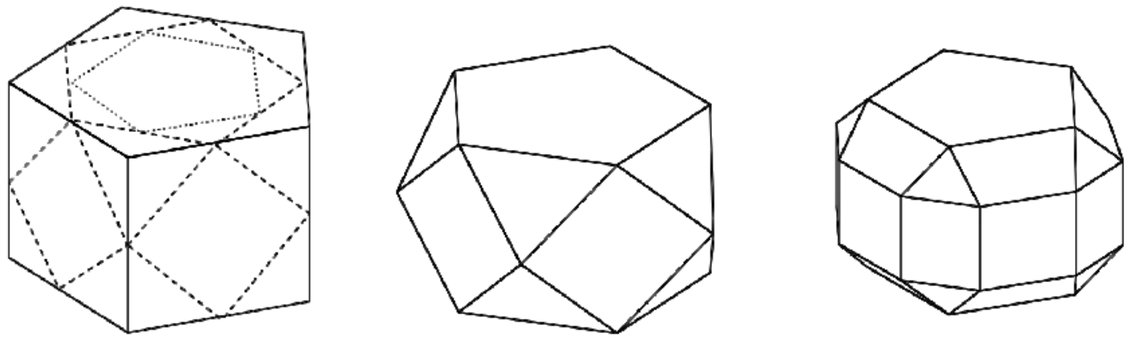
Subdivision surfaces

Connectivity of surfaces is more complicated. Many different subdivision schemes are possible:

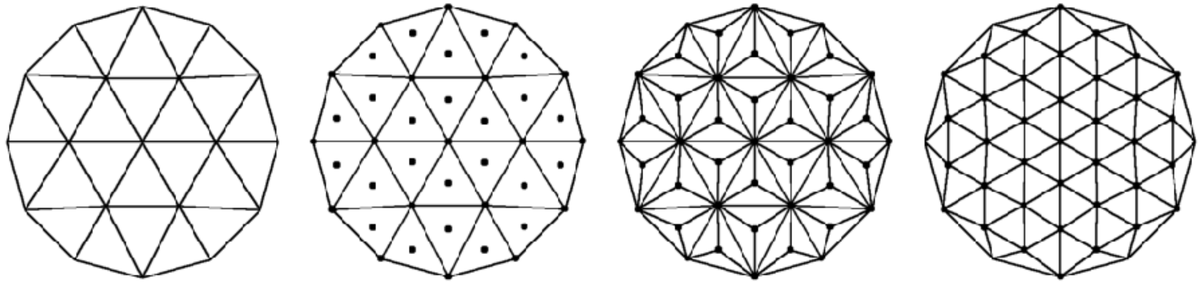
- **General polygon meshes:** Catmull-Clark, Doo-Sabin, mid-edge [Peters & Reif], …
- **Triangle meshes:** Loop, modified butterfly [Zorin et al.], Sqrt(3) [Kobbelt], …



Doo-Sabin



mid-edge



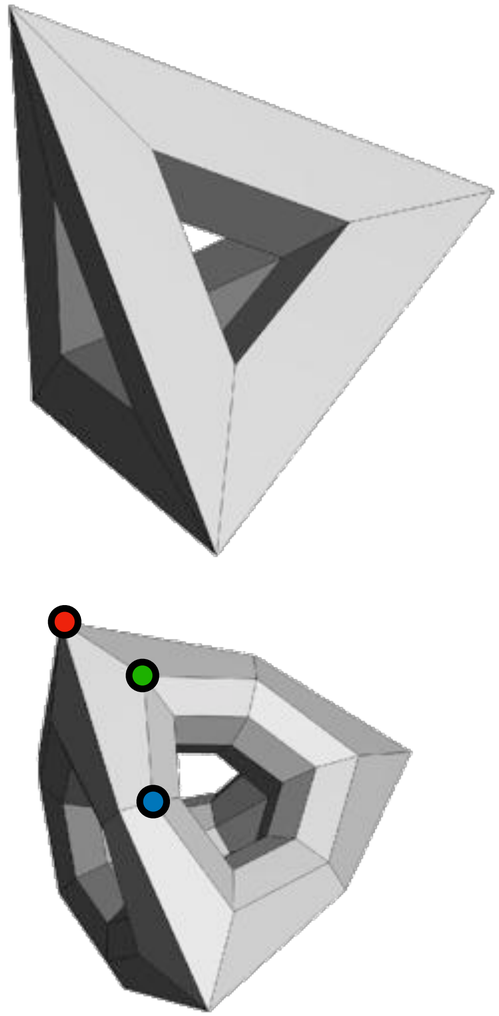
Sqrt(3)

Catmull-Clark subdivision

Split each *n*-sided face into *n* quads

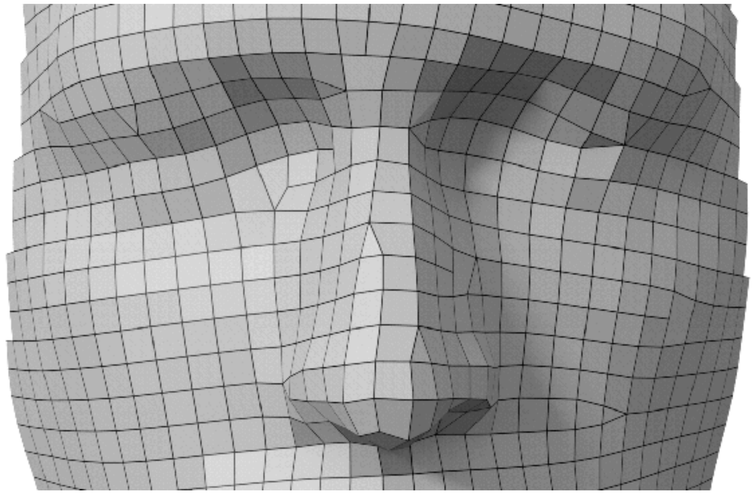
Update vertex positions by averaging:

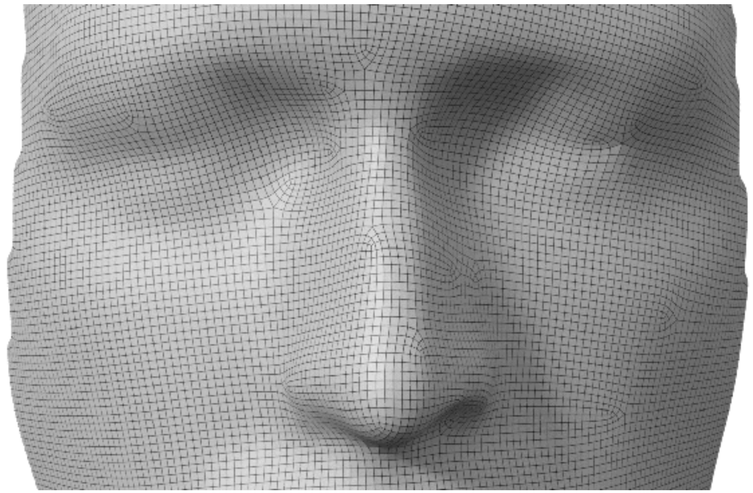
- New face point = average of old face vertices
- New edge point = average of 2 old vertices and 2 new face points
- Updated vertex = $\frac{1}{n}(Q + 2R + (n-3)S)$
  where Q = average of n new face points,
  R = average of n new edge points, S = old vertex

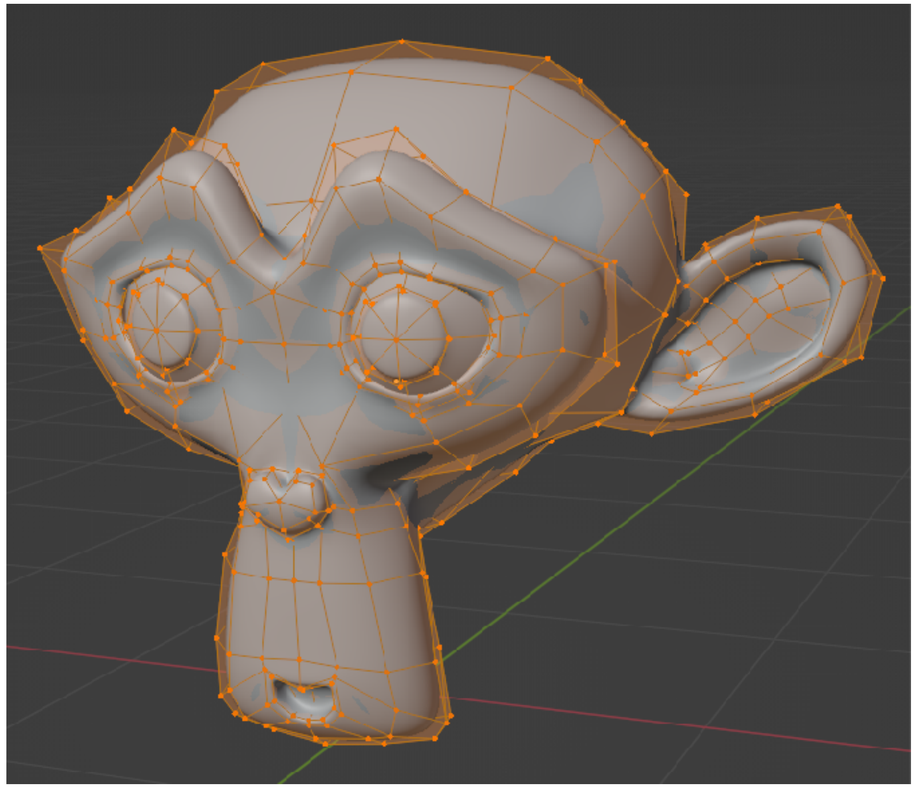


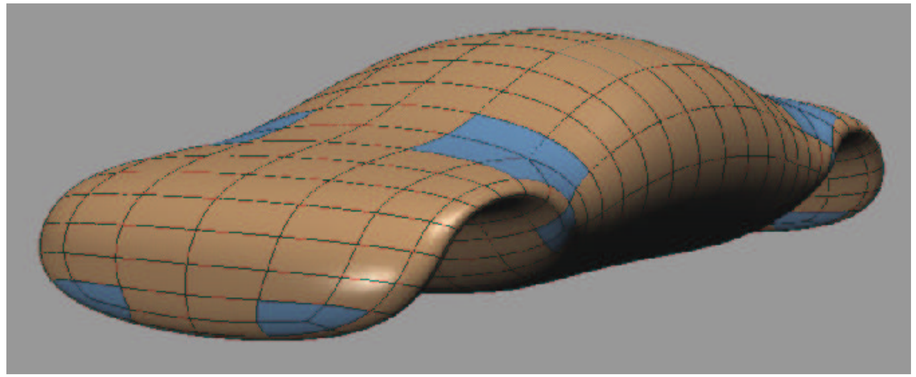
Pixar

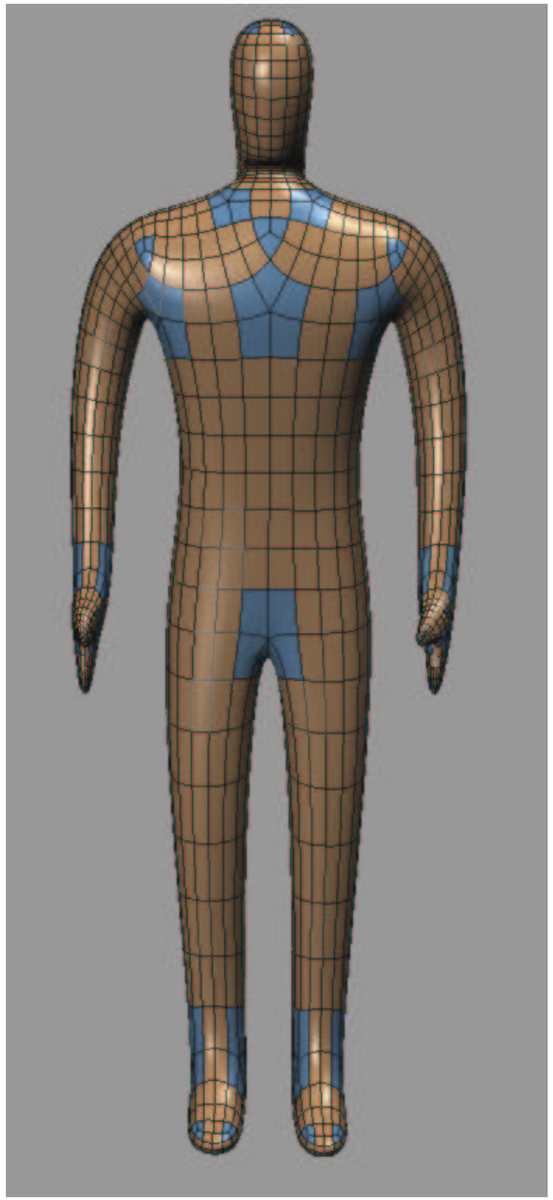
Examples





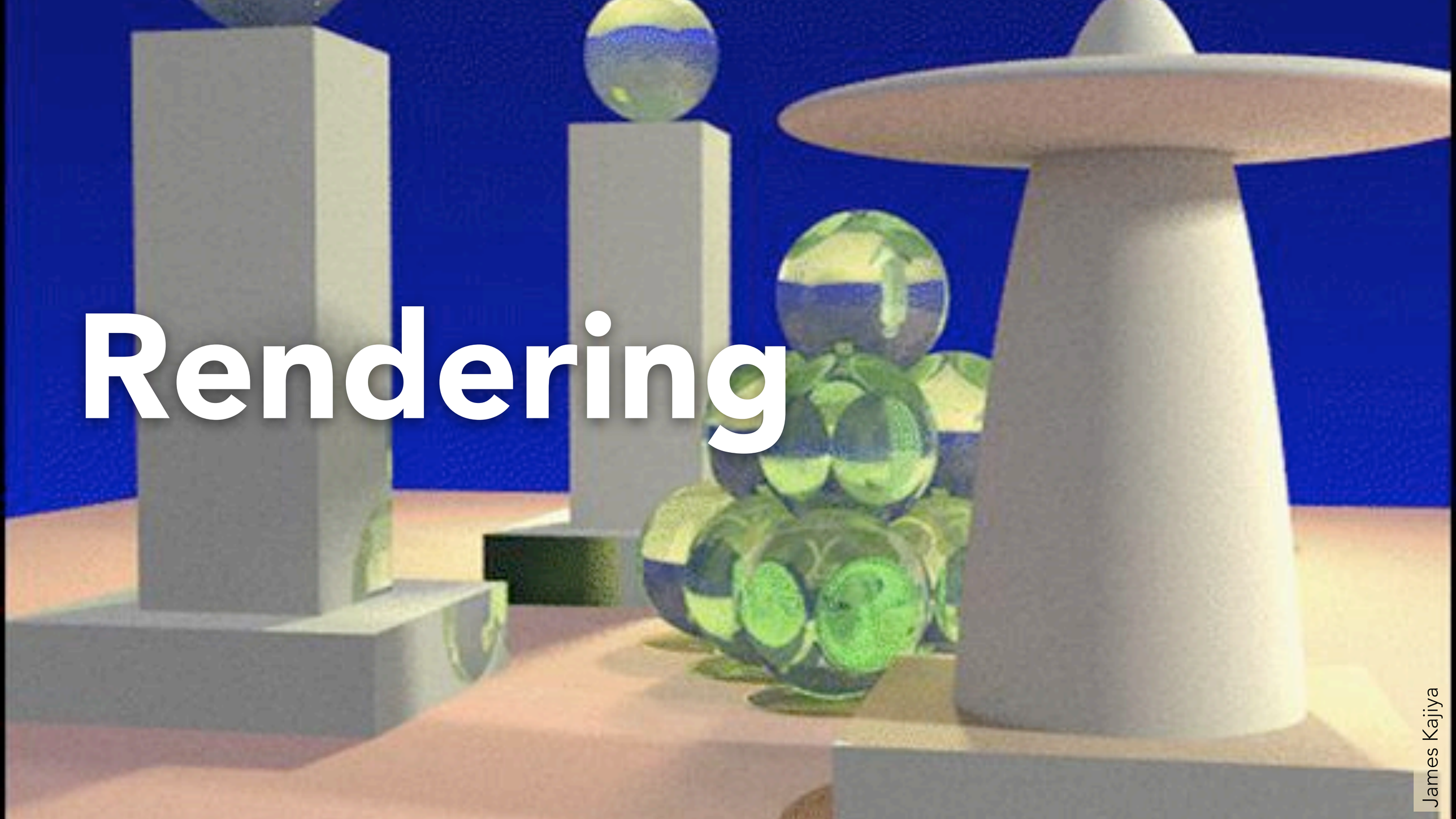






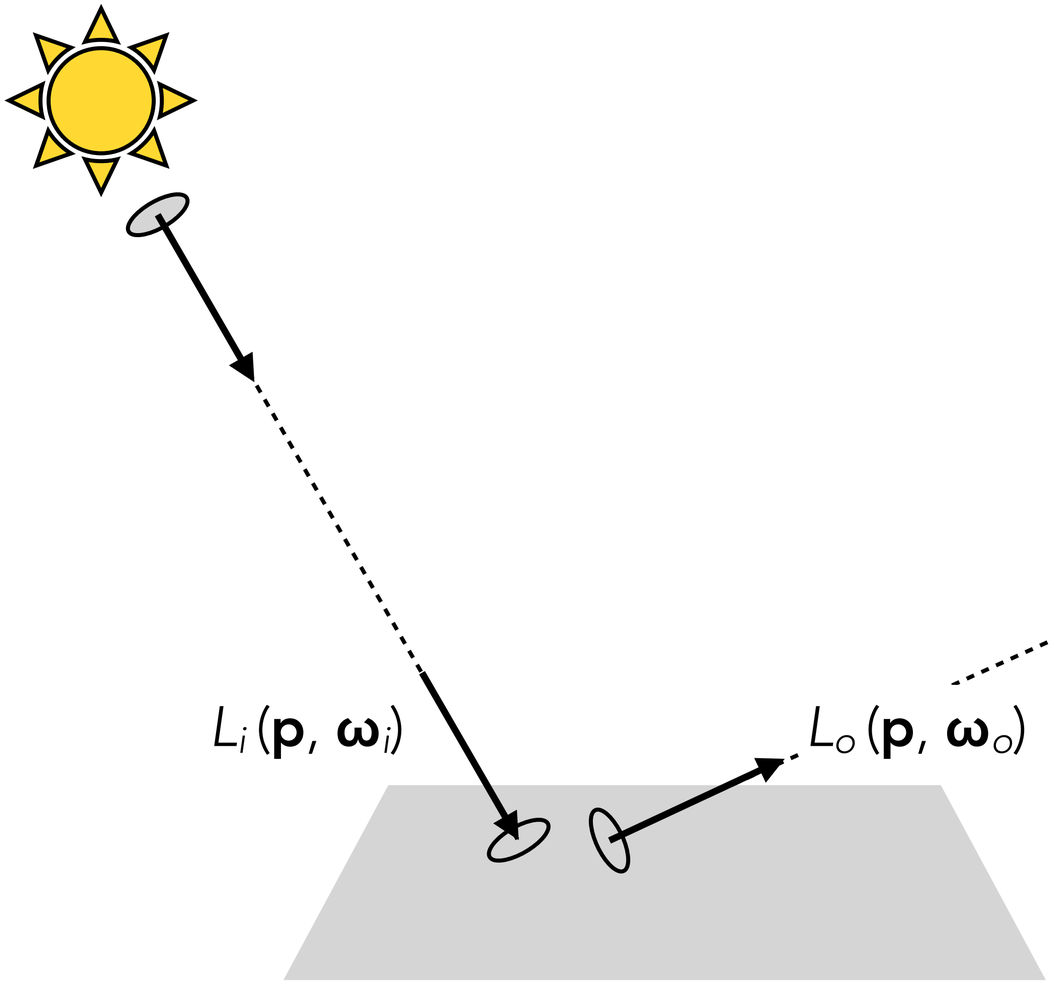
Break

Rendering



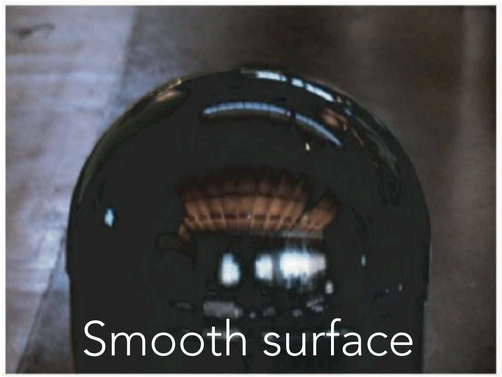
James Kajiya

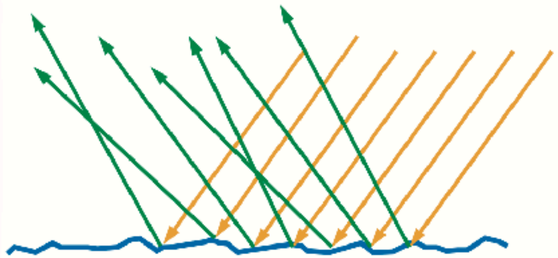
Rendering

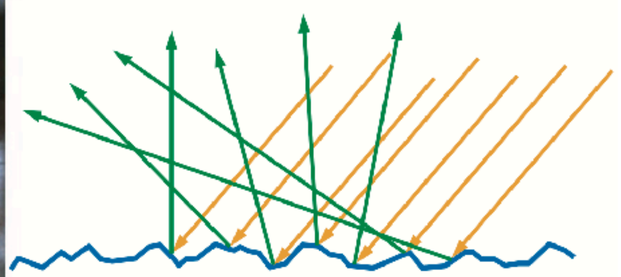


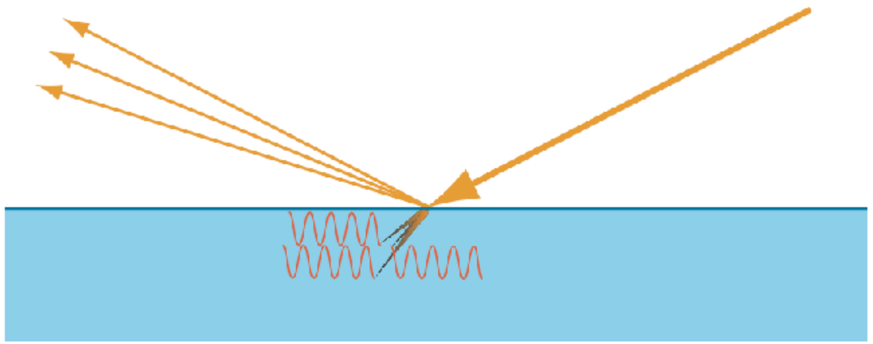
Our goal: given light sources and scene geometry, find amount of light (i.e. **radiance**) incident on camera.

To do this, we need to know how surfaces transform **incident** radiance into **exitant** radiance

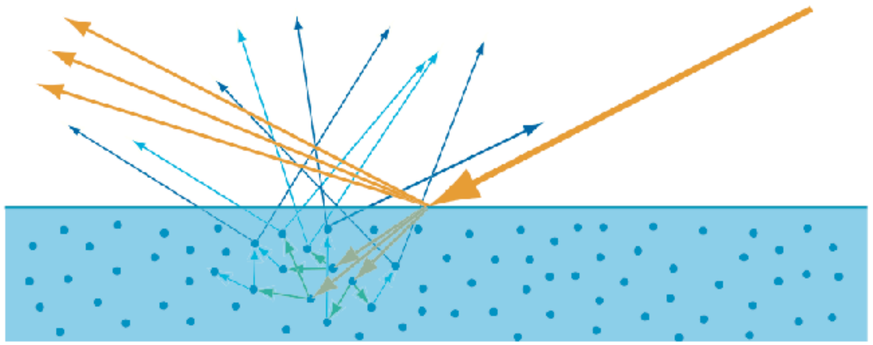






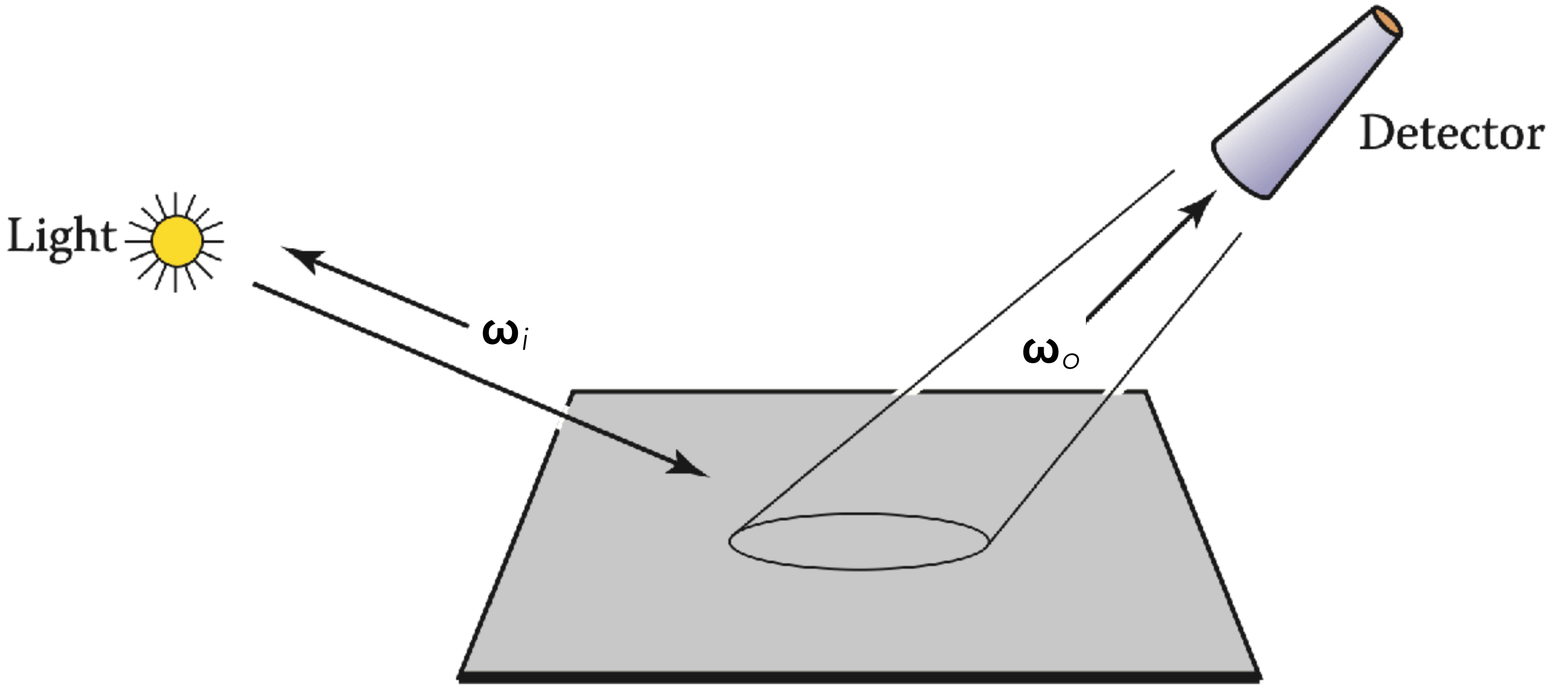


Metals



Non-metals

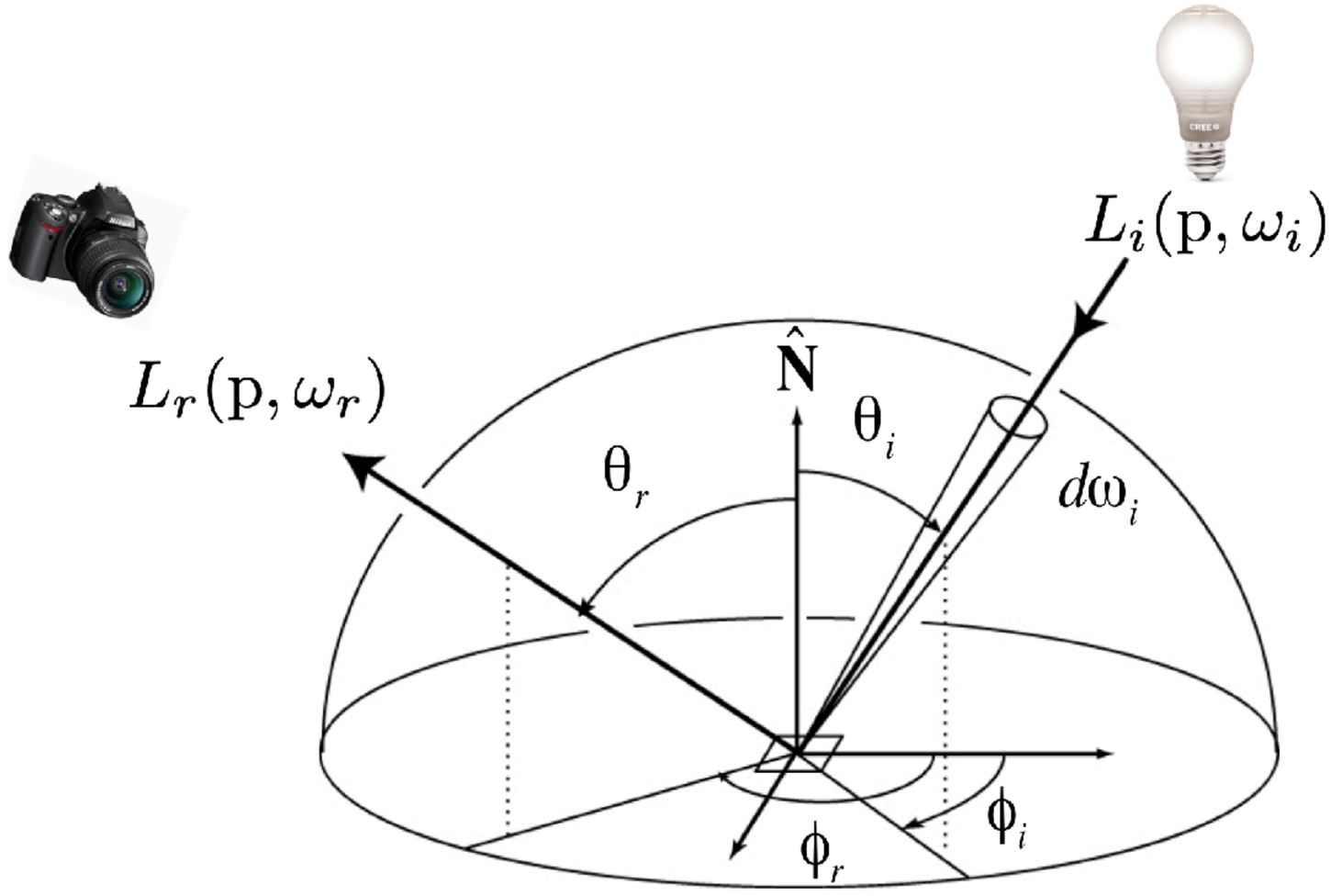
The BRDF



Fundamentals of Computer Graphics

Bidirectional Reflectance Distribution Function

$$f_r(\omega_i \rightarrow \omega_o) = L_o(\mathbf{x}, \omega_o)/E(\mathbf{x})$$



$$L_r(\mathbf{x}, \boldsymbol{\omega}_r) = \int_{H^2} f_r(\boldsymbol{\omega}_i \rightarrow \boldsymbol{\omega}_r)\, L_i(\mathbf{x}, \boldsymbol{\omega}_i) \cos(\theta_i)\, \mathrm{d}\boldsymbol{\omega}_i$$

Lambertian (diffuse) material

Simplest possible model: BRDF is a constant!

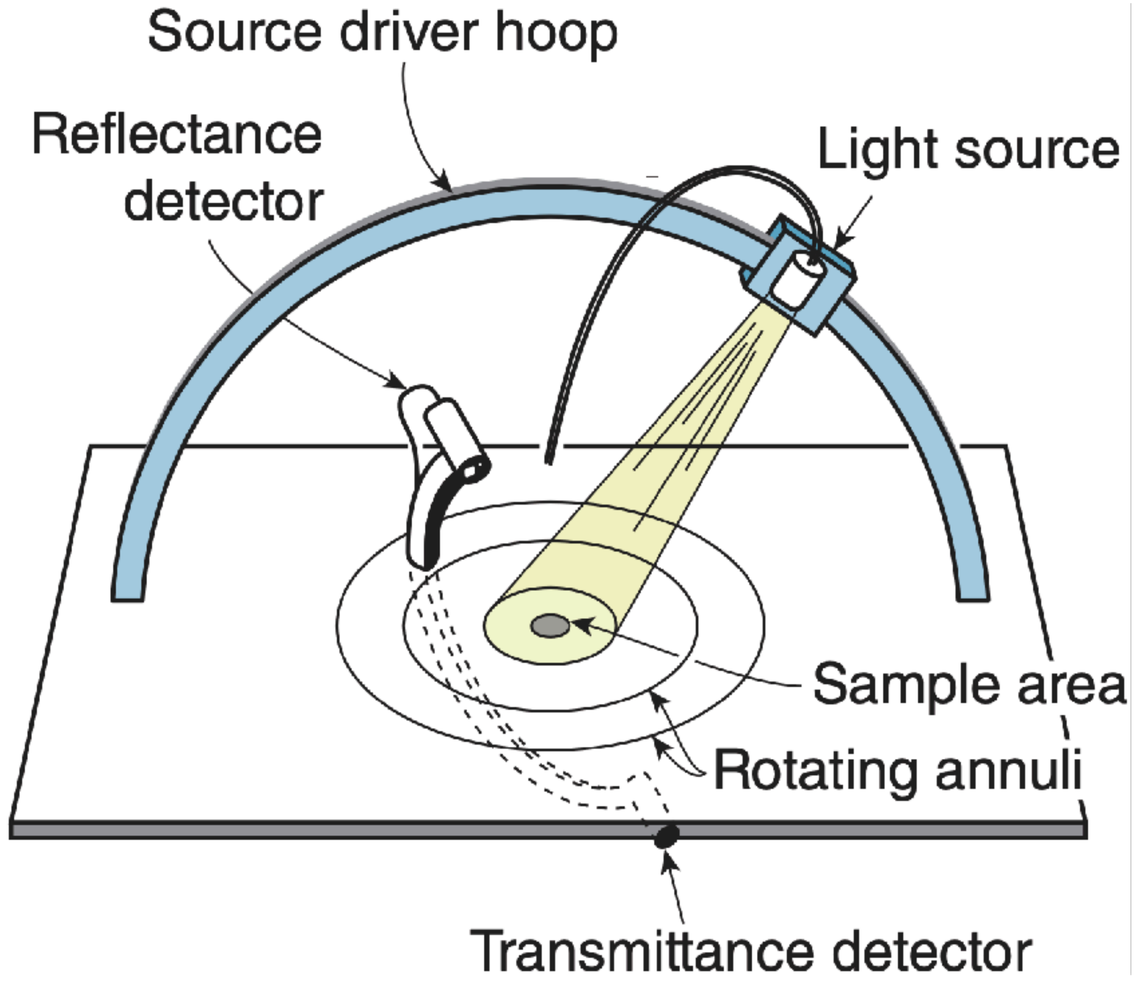
$$L_o(\boldsymbol{\omega}_o) = \int_{H^2} f_r \, L_i(\boldsymbol{\omega}_i) \cos(\theta_i) \, d\boldsymbol{\omega}_i$$

$$= f_r \, E_i$$

To conserve energy, $f_r = \rho/\pi$ where **albedo** ρ is ≤ 1

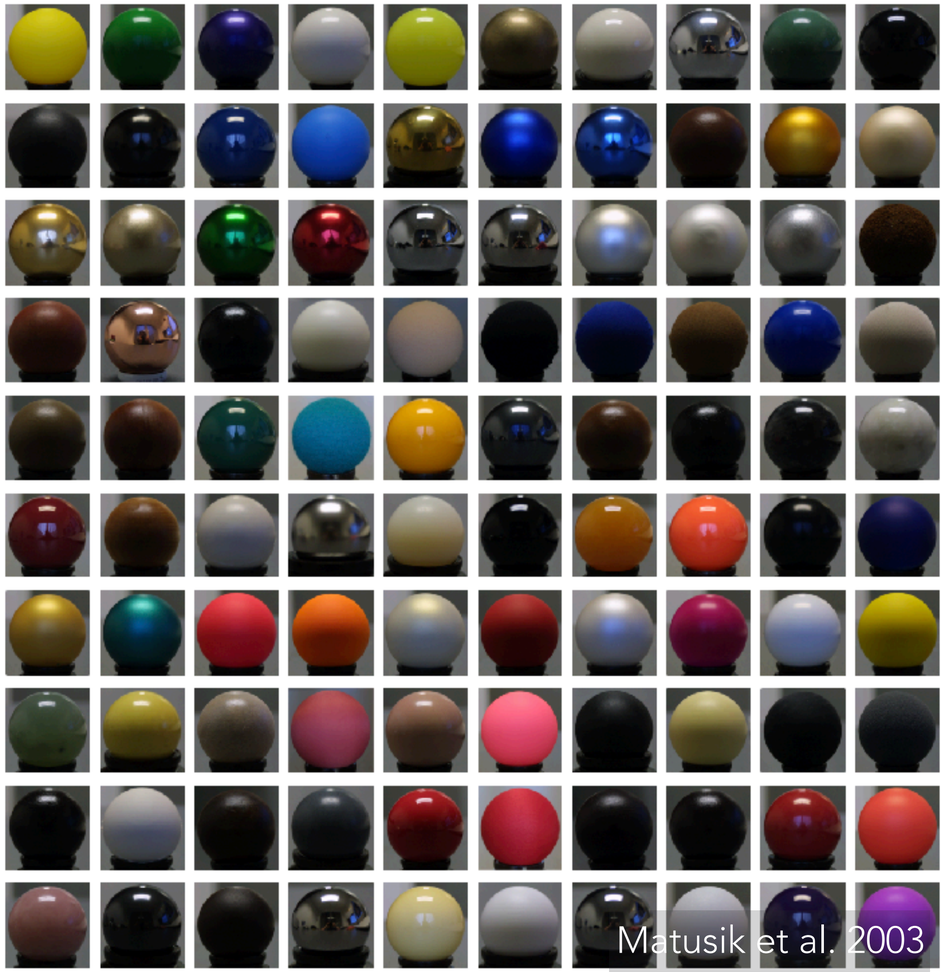
Why? For constant radiance L, total flux density = $L\,\pi$

BRDF acquisition: Gonioreflectometer



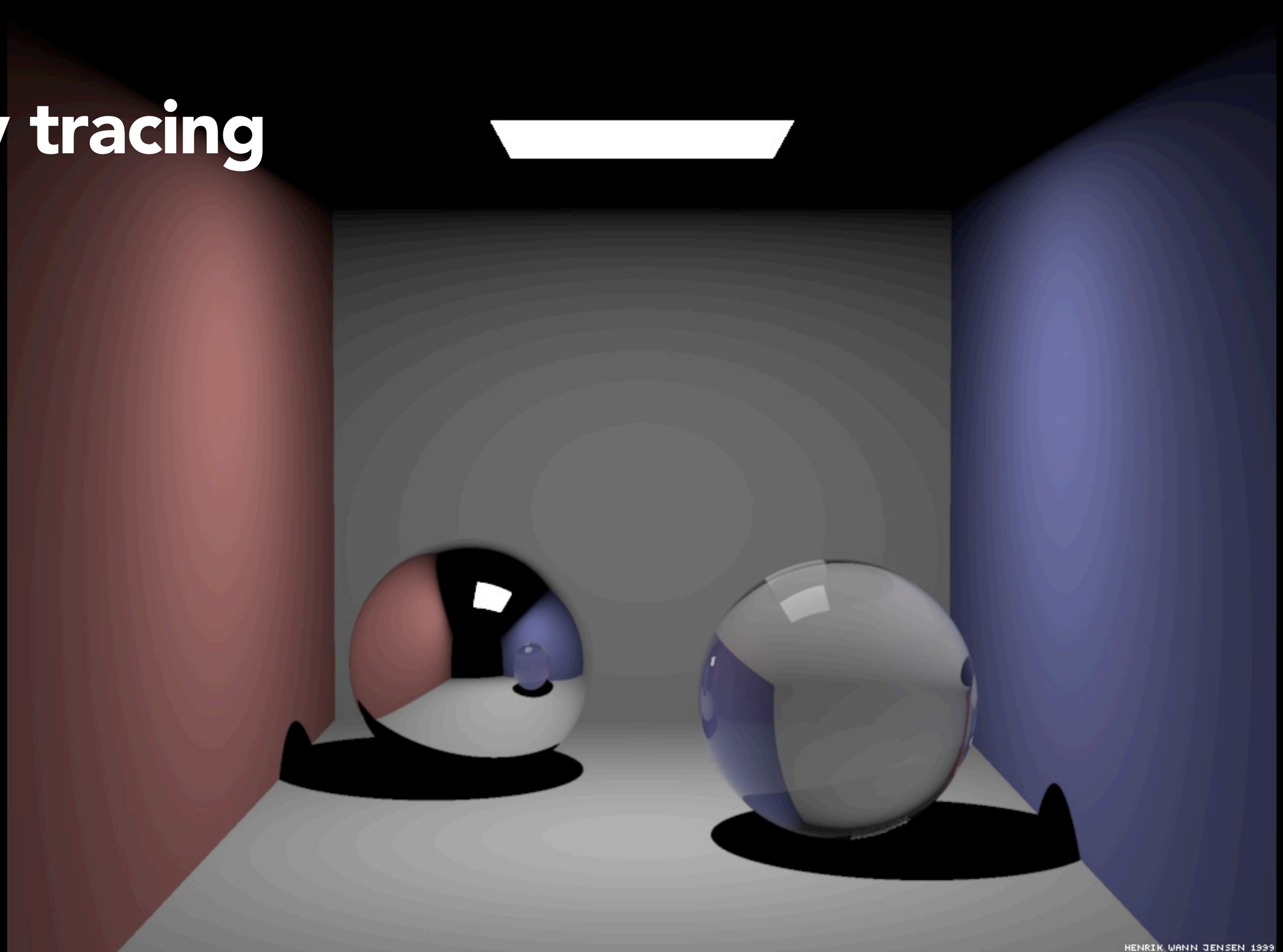
Computer Graphics: Principles and Practice

MERL BRDF database



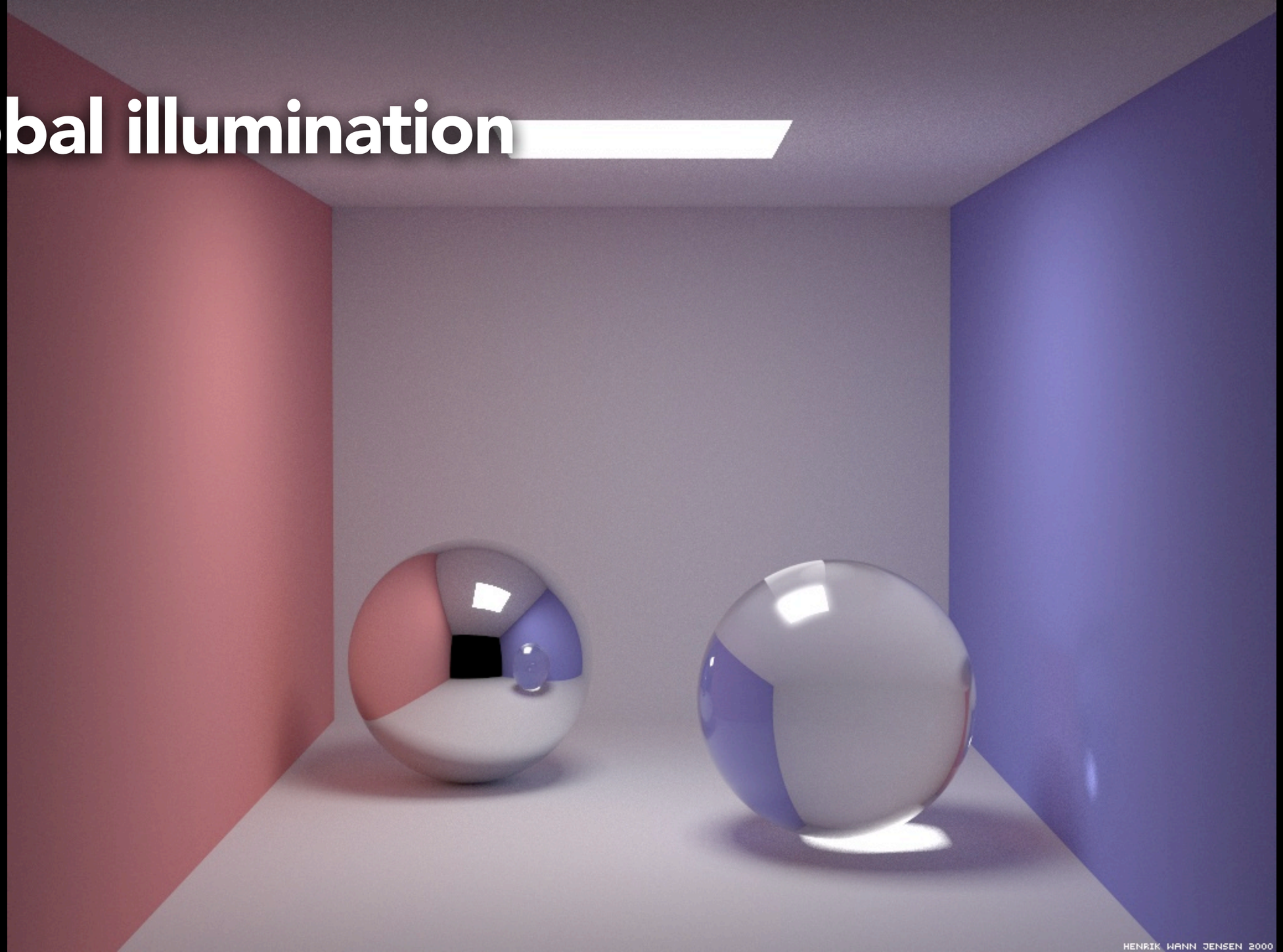
Matusik et al. 2003

Ray tracing



Henrik Wann Jensen

Global illumination

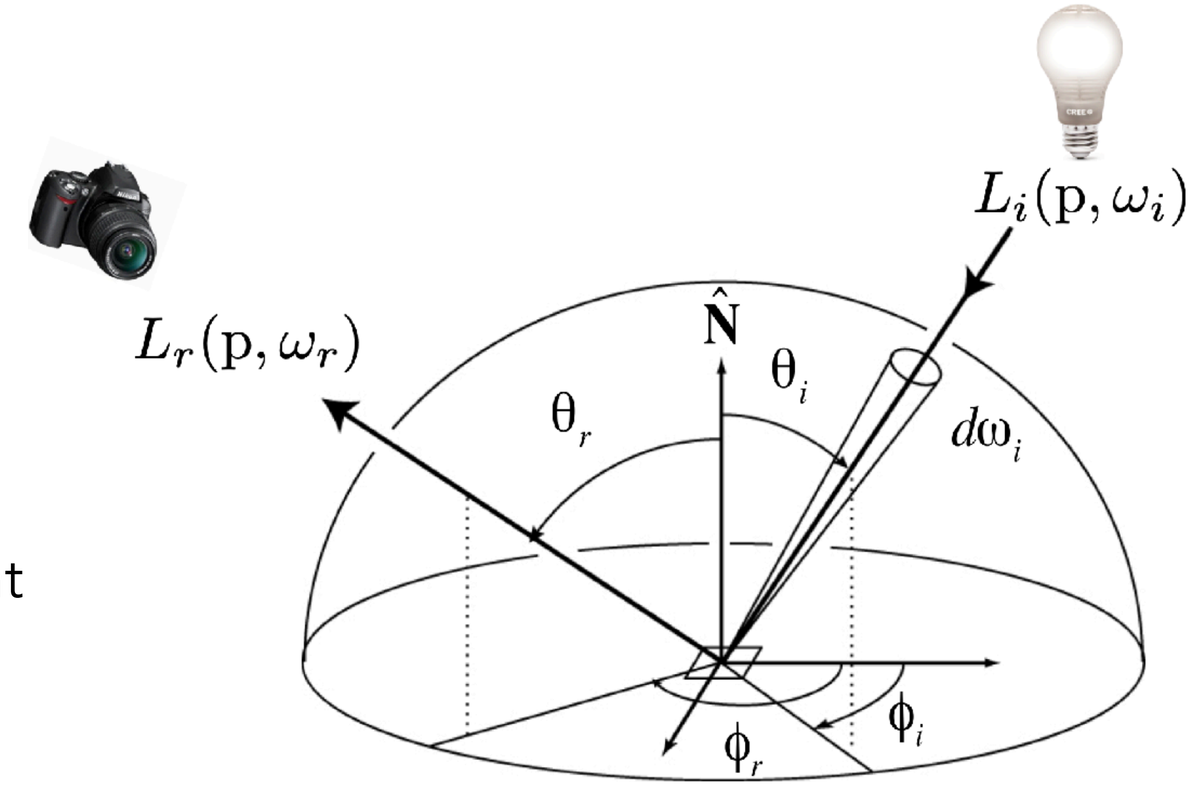


Henrik Wann Jensen

Ray tracing revisited

For each sample:

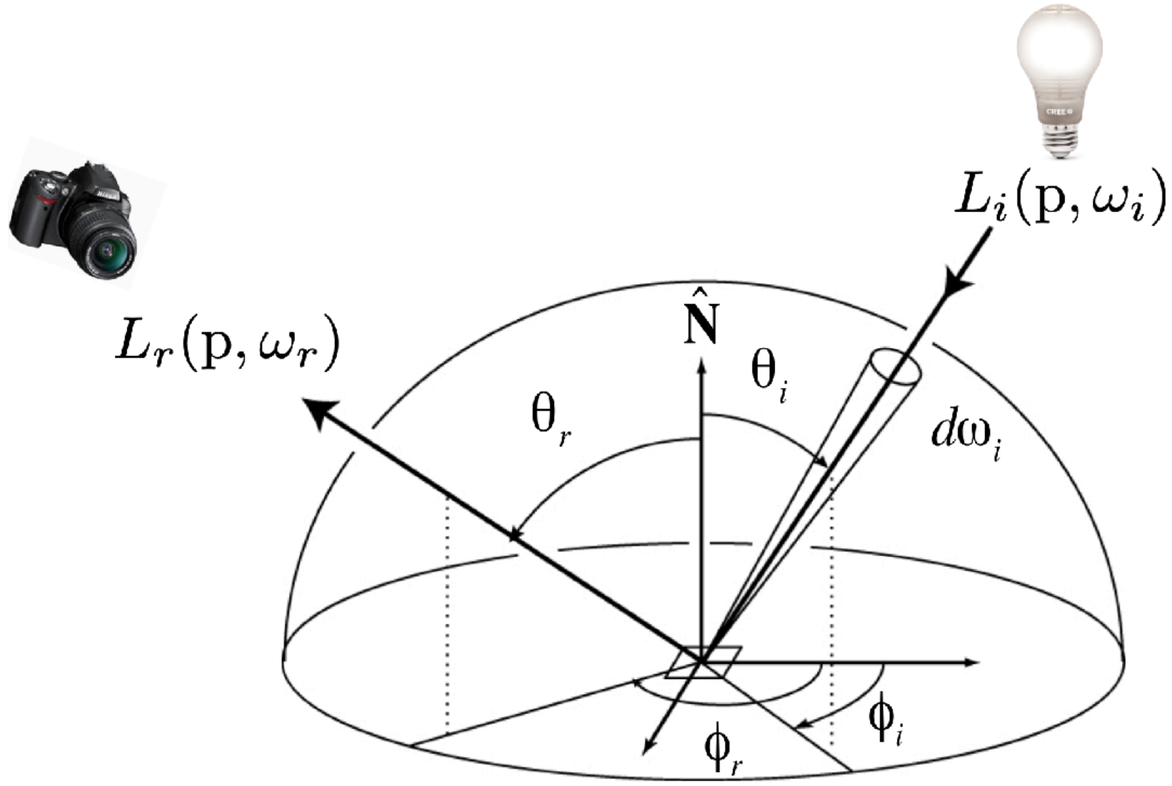
- Cast a ray into the scene
- Find the closest intersection
- Get **exitant radiance** at intersection point
- Set sample colour to it



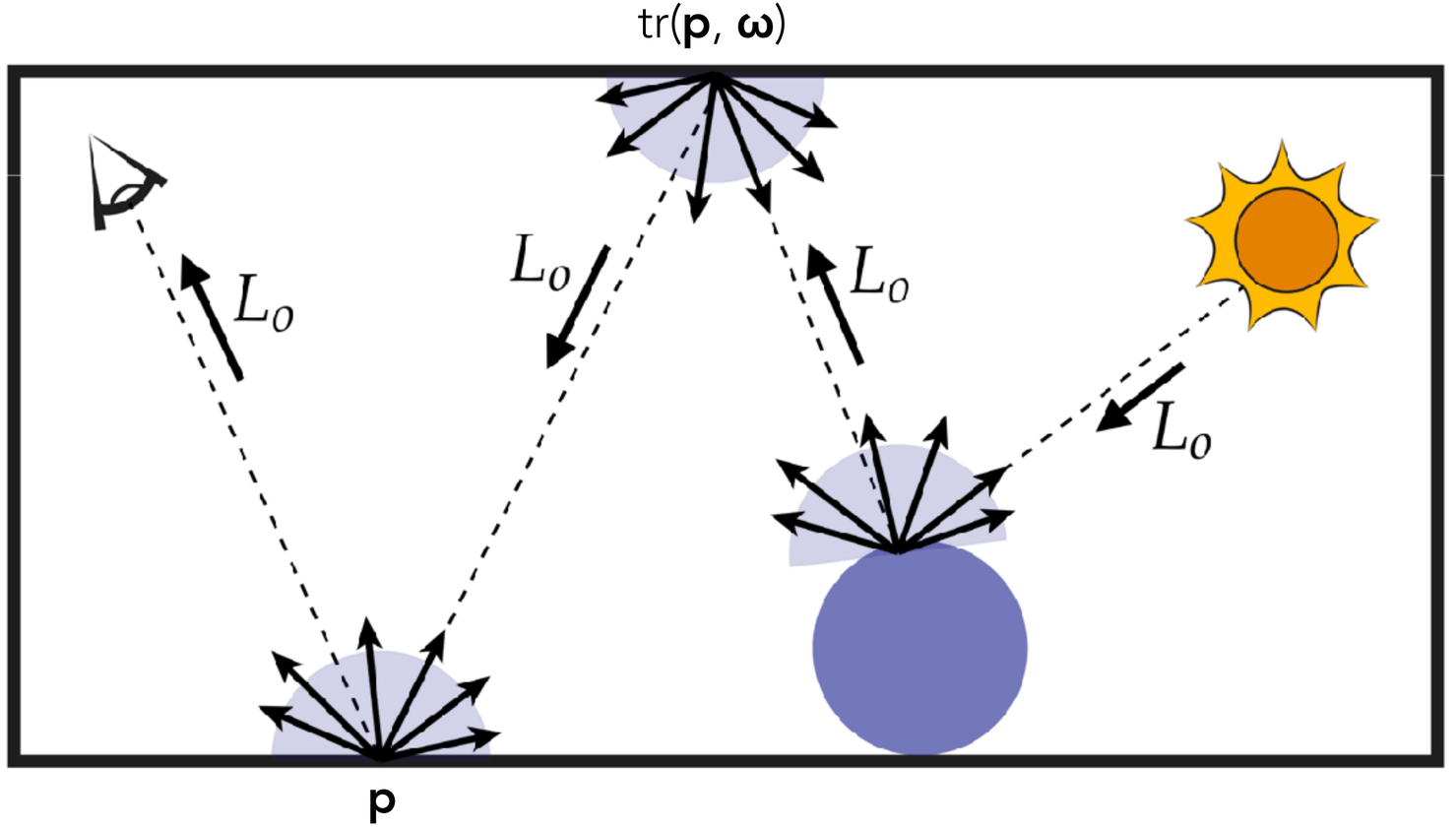
$$L_o(\mathbf{p}, \boldsymbol{\omega}_o) = L_e(\mathbf{p}, \boldsymbol{\omega}_o) + \int_{H^2} f_r(\mathbf{p}, \boldsymbol{\omega}_i \rightarrow \boldsymbol{\omega}_o)\, L_i(\mathbf{p}, \boldsymbol{\omega}_i) \cos(\theta_i)\, \mathrm{d}\boldsymbol{\omega}_i$$

$$L_o(\mathbf{p}, \boldsymbol{\omega}_o) = L_e(\mathbf{p}, \boldsymbol{\omega}_o) + \int_{H^2} f_r(\mathbf{p}, \boldsymbol{\omega}_i \rightarrow \boldsymbol{\omega}_o)\, L_i(\mathbf{p}, \boldsymbol{\omega}_i) \cos(\theta_i)\, \mathrm{d}\boldsymbol{\omega}_i$$

- How to evaluate incident radiance from **any** direction (not just light sources)?
- How to compute the integral over a hemisphere?



What is $L_i(\mathbf{p}, \boldsymbol{\omega}_i)$? Simply exitant radiance from somewhere else!



Keenan Crane

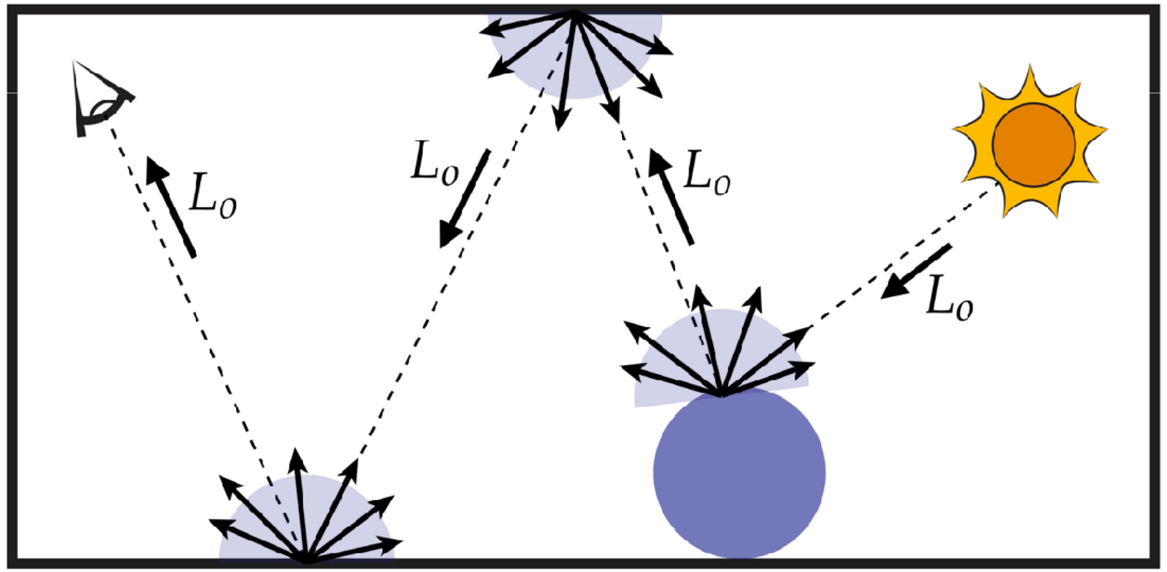
Define tr($\mathbf{p}$, $\boldsymbol{\omega}$) as the first surface point hit by the ray $\mathbf{p} + t\boldsymbol{\omega}$.

$$L_i(\mathbf{p}, \boldsymbol{\omega}_i) = L_o(\text{tr}(\mathbf{p}, \boldsymbol{\omega}_i), -\boldsymbol{\omega}_i)$$

$$L_o(\mathbf{p}, \boldsymbol{\omega}_o) = L_e(\mathbf{p}, \boldsymbol{\omega}_o) + \int_{H^2} f_r(\mathbf{p}, \boldsymbol{\omega}_i \rightarrow \boldsymbol{\omega}_o)\, L_o(\mathrm{tr}(\mathbf{p}, \boldsymbol{\omega}_i), -\boldsymbol{\omega}_i) \cos(\theta_i)\, \mathrm{d}\boldsymbol{\omega}_i$$

This is an **integral equation**!
Unknown quantity L_o on both sides

Like ray tracing, we'll evaluate it recursively



Quick probability recap

If X is a random variable with probability distribution $p(x)$, its **expected value** or **expectation** is

$$E[X] = \sum x_i \, p_i \qquad \text{(discrete)}$$

$$E[X] = \int x \, p(x) \, dx \qquad \text{(continuous)}$$

Expectation is **linear**:

- $E[X_1 + X_2] = E[X_1] + E[X_2]$
- $E[aX] = a \, E[X]$

The basic Monte Carlo method

If X is uniformly distributed in [a, b], then

$$E[f(X)] = \frac{1}{b-a}\int_a^b f(x)\,\mathrm{d}x$$

So, if I take N independent samples of X,

$$\frac{1}{N}\sum_{i=1}^{N} f(x_i) \approx E[f(X)] = \frac{1}{b-a}\int_a^b f(x)\,\mathrm{d}x$$

$$\int_a^b f(x)\,\mathrm{d}x \approx \frac{b-a}{N}\sum_{i=1}^{N} f(x_i)$$

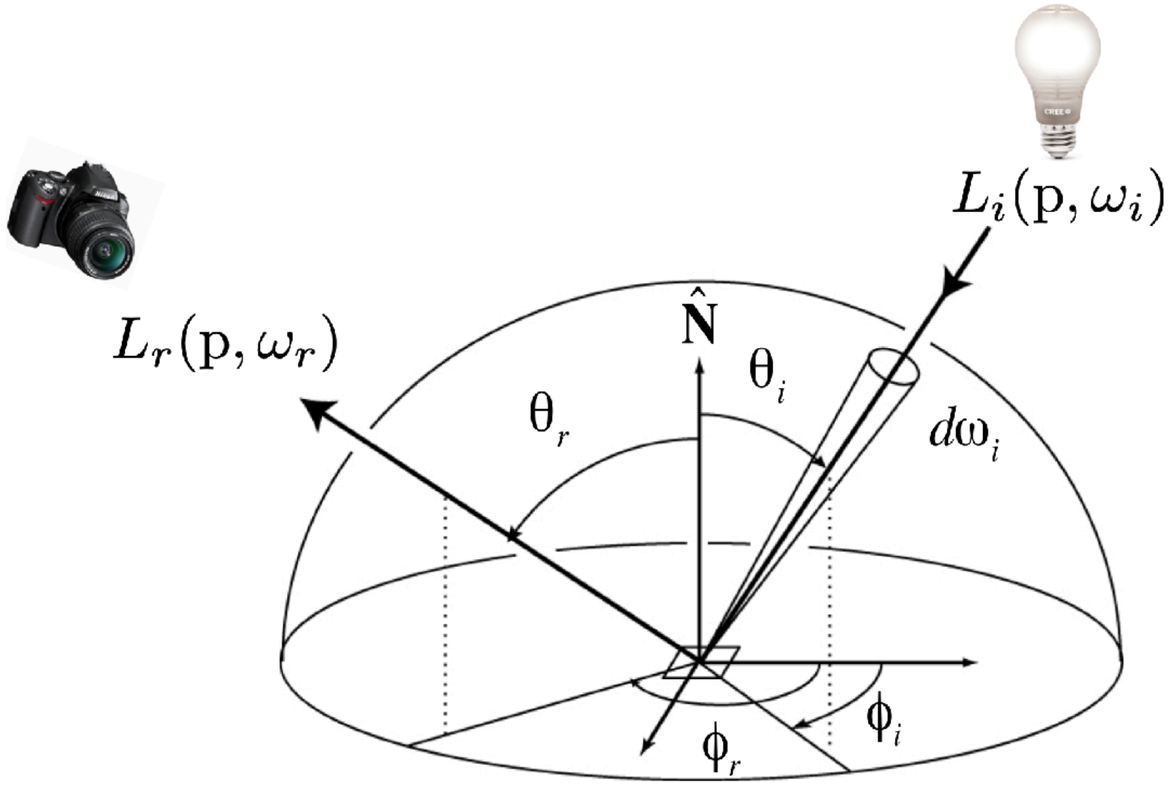
Interpretation:
Integral = average value × domain size

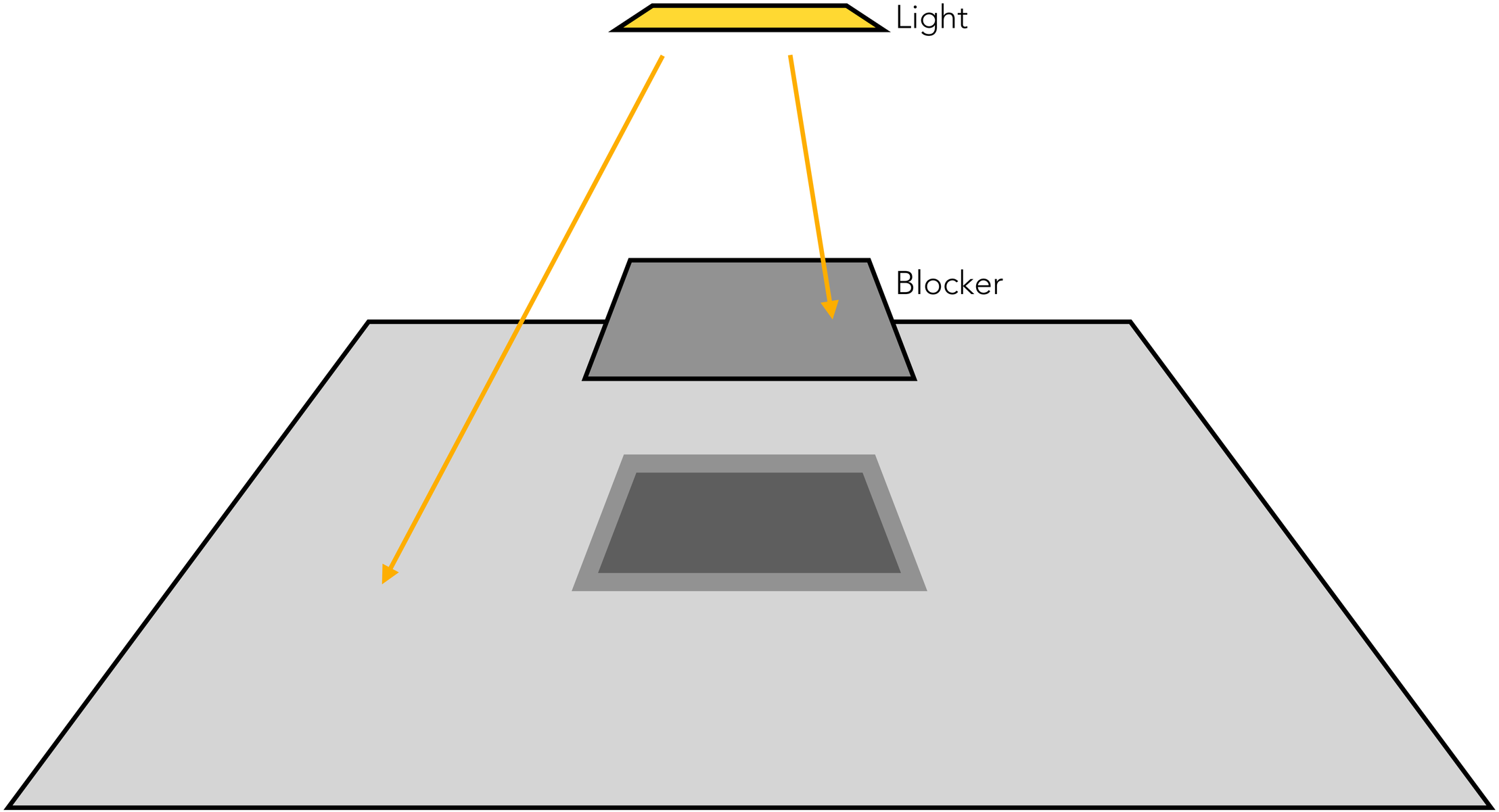
Monte Carlo rendering

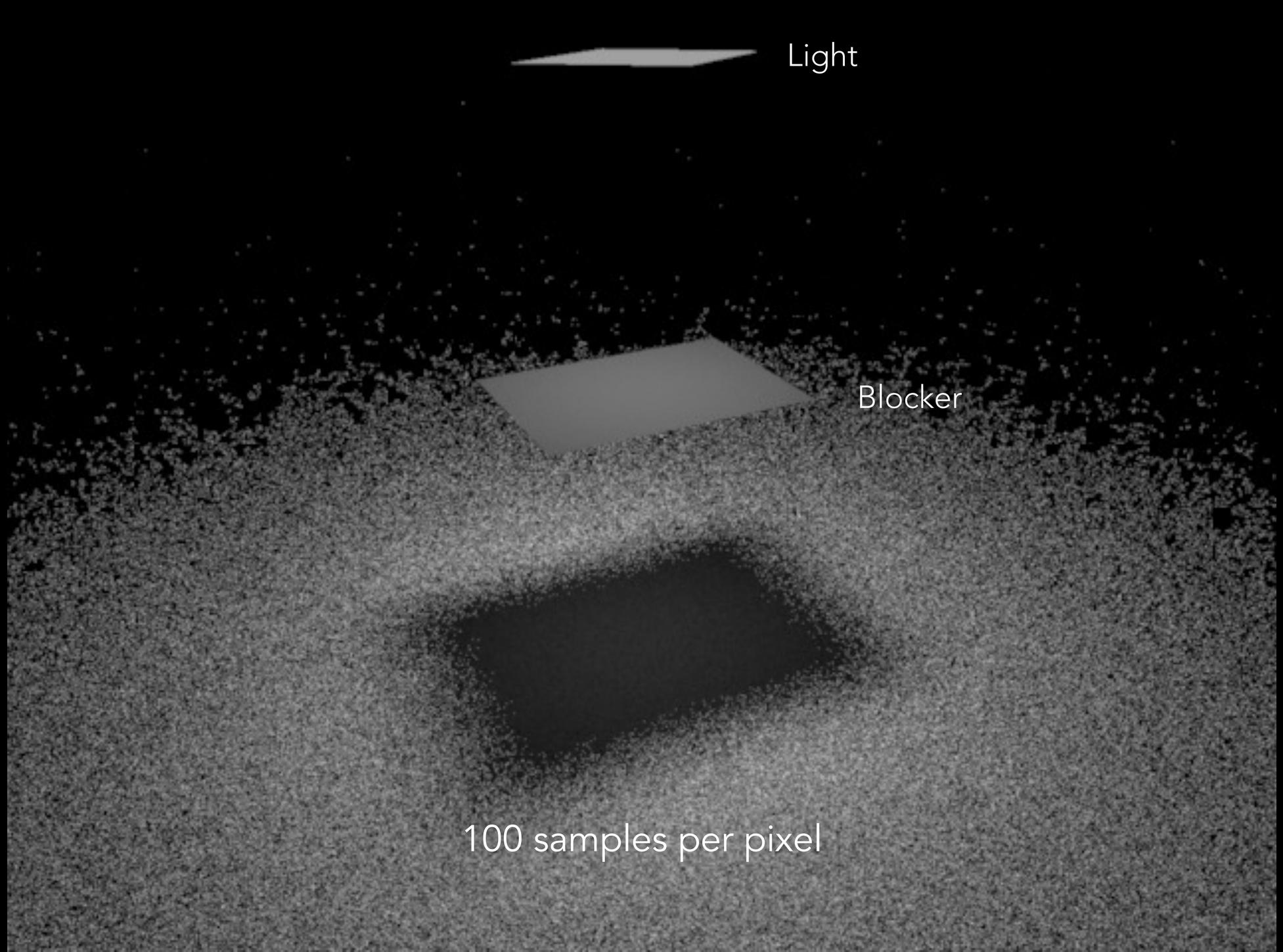
We need to estimate the reflectance integral $\int_{H^2} f_r(\mathbf{p}, \boldsymbol{\omega}_i \rightarrow \boldsymbol{\omega}_o)\, L_i(\mathbf{p}, \boldsymbol{\omega}_i) \cos(\theta_i)\, d\boldsymbol{\omega}_i$

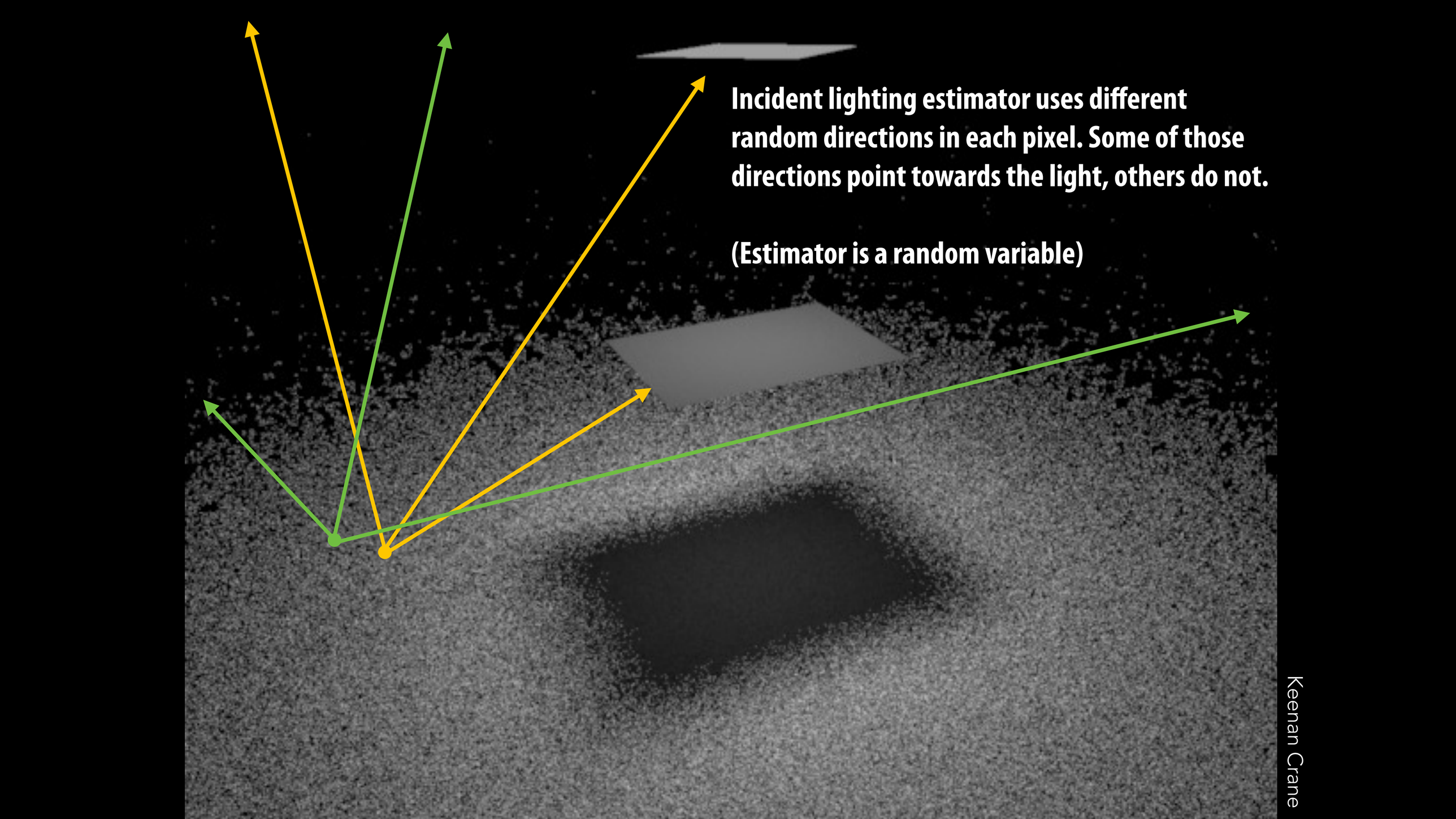
With Monte Carlo, it's easy:

- Uniformly sample hemisphere of incident directions: $\mathbf{X}_i \sim U(H^2)$, probability density $p(\boldsymbol{\omega}) = 1/(2\pi)$
- Evaluate integrand $Y_i = f_r(\mathbf{p}, \mathbf{X}_i \rightarrow \boldsymbol{\omega}_o)\, L_i(\mathbf{p}, \mathbf{X}_i) \cos(\theta_i)$
- MC estimator is simply $F_N = 2\pi/N \sum Y_i$




Light
Blocker

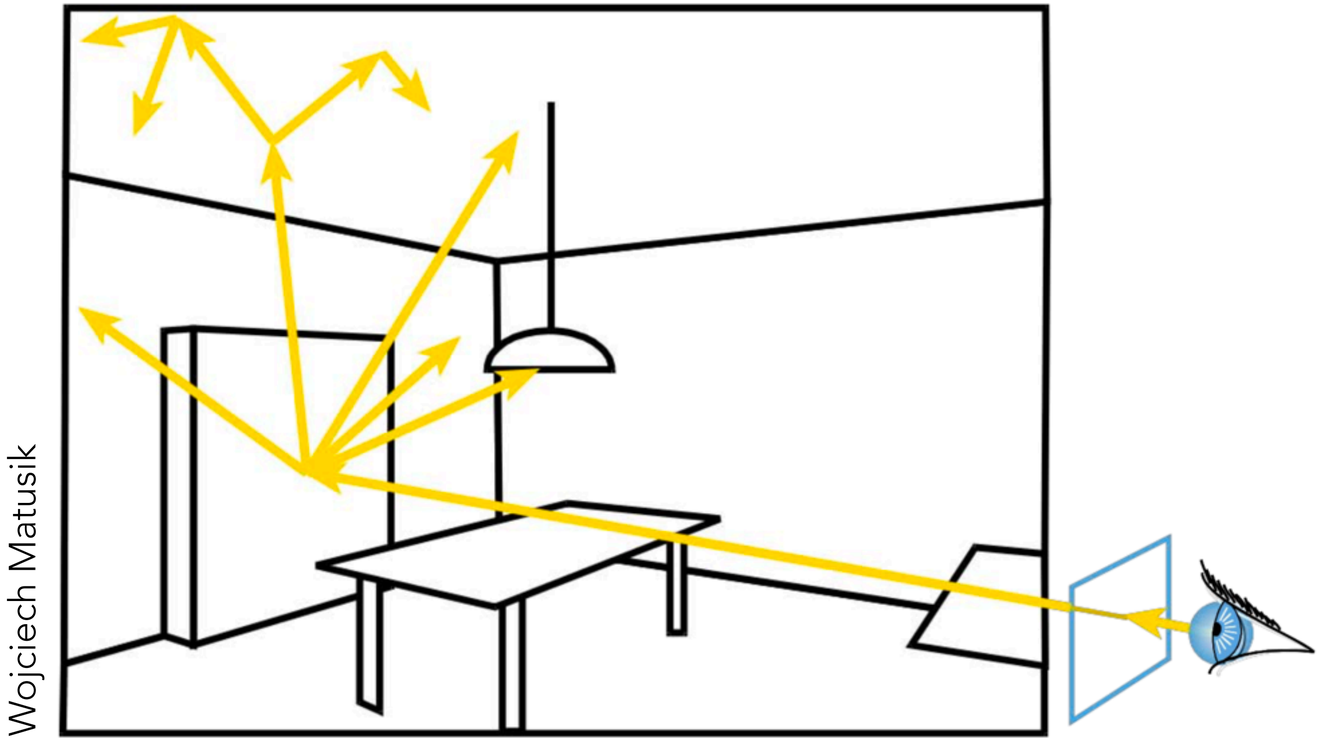

Light
Blocker
100 samples per pixel


Incident lighting estimator uses different random directions in each pixel. Some of those directions point towards the light, others do not.
(Estimator is a random variable)
Keenan Crane

$$L_o(\mathbf{p}, \boldsymbol{\omega}_o) = L_e(\mathbf{p}, \boldsymbol{\omega}_o) + \int_{H^2} f_r(\mathbf{p}, \boldsymbol{\omega}_i \rightarrow \boldsymbol{\omega}_o)\, L_i(\mathbf{p}, \boldsymbol{\omega}_i) \cos(\theta_i)\, \mathrm{d}\boldsymbol{\omega}_i$$

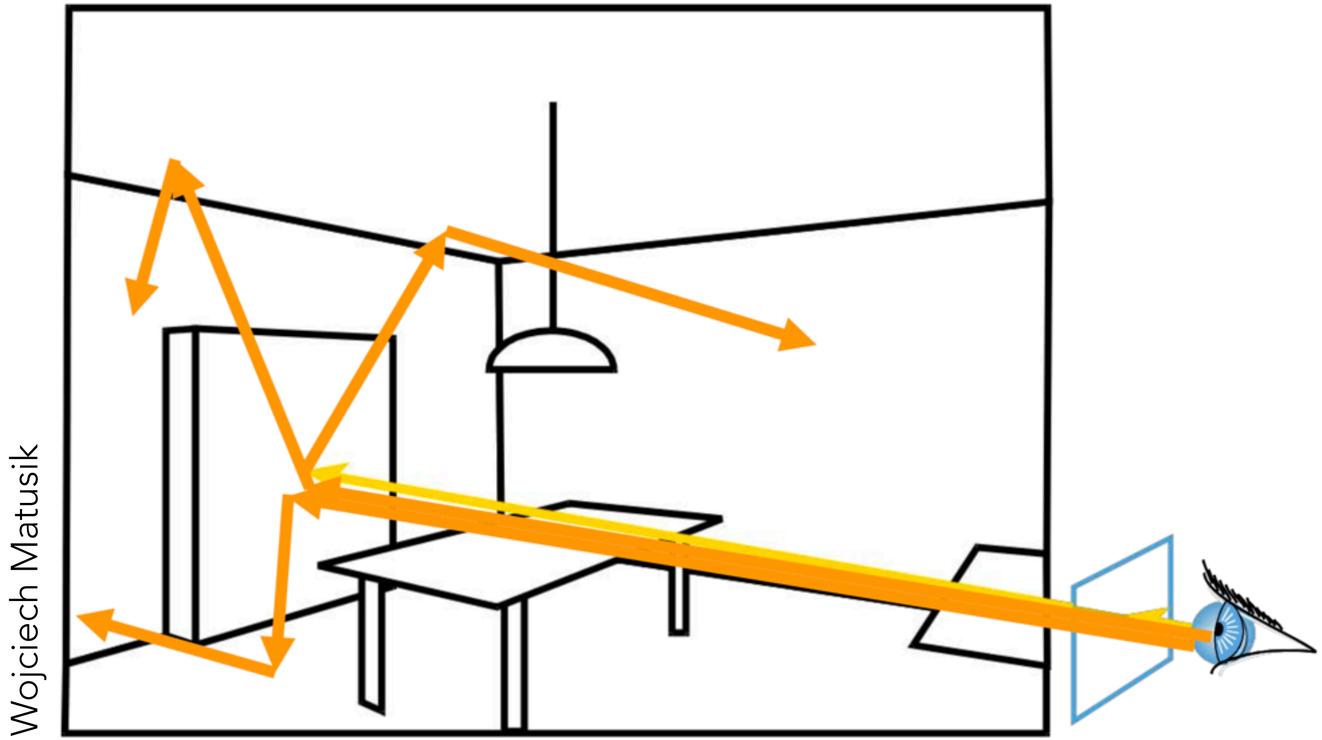
```
incidentRadiance(x, ω):
    p = intersectScene(x, ω)
    L = p.emittedLight(−ω)
    for i = 1, …, N:
        ωi = sampleDirection(p.normal)
        L += incidentRadiance(p, ωi) * p.BRDF(ωi, −ω) * cos_θi * 2π / N
    return L
```

Problem: Exponential increase in number of samples per bounce



Wojciech Matusik

Solution: Just take one recursive sample per bounce!



But take many samples per pixel, and average them.

incidentRadiance(**x**, **ω**):

p = intersectScene(**x**, **ω**)

L = **p**.emittedLight(−**ω**)

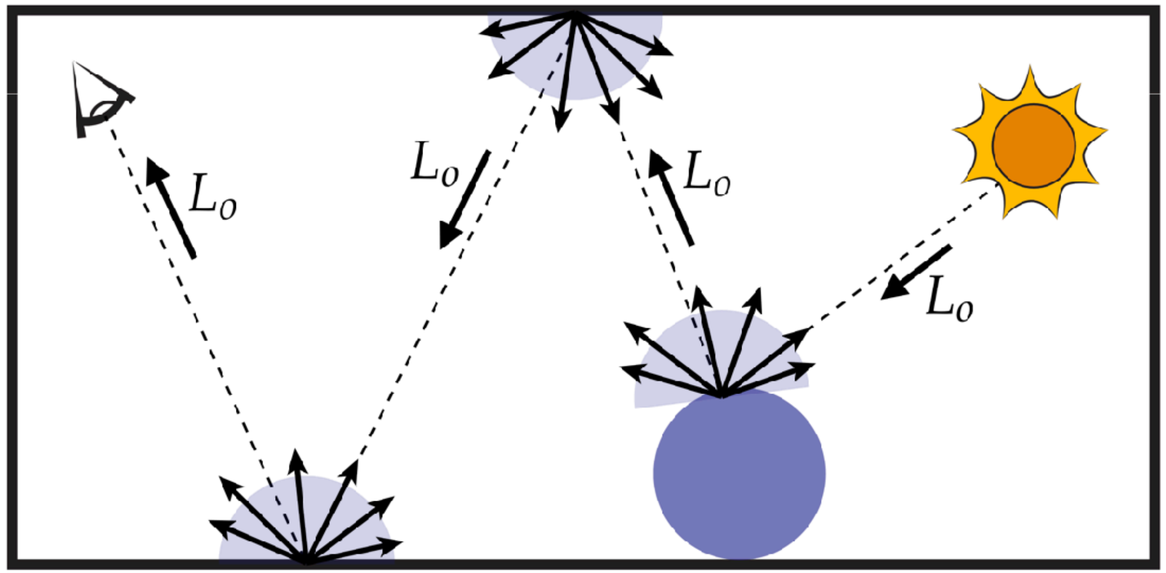
ω*i* = sampleDirection(**p**.normal)

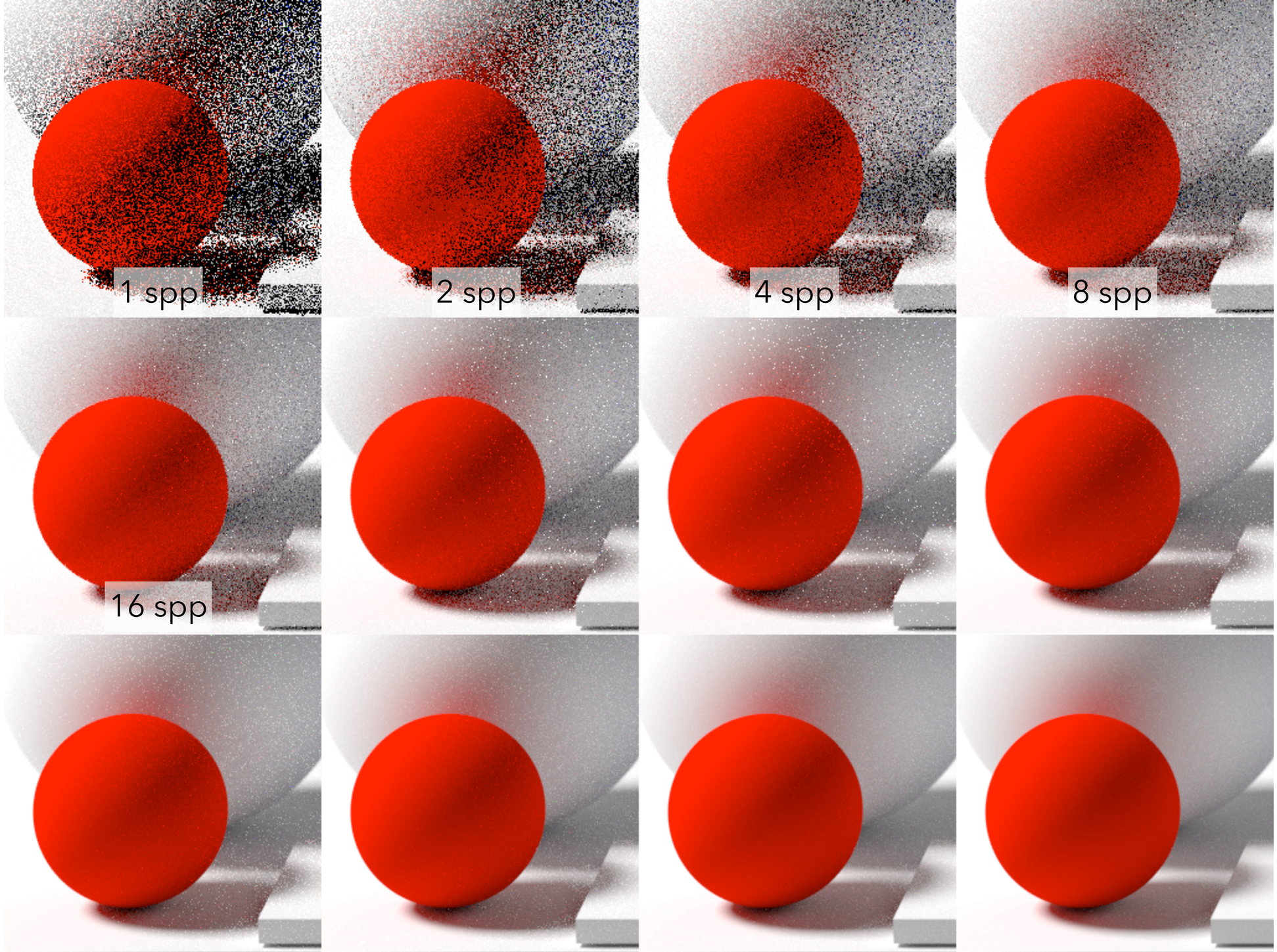
L += incidentRadiance(**p**, **ω***i*) * **p**.BRDF(**ω***i*, −**ω**) * cos_*θi* * 2π

return *L*

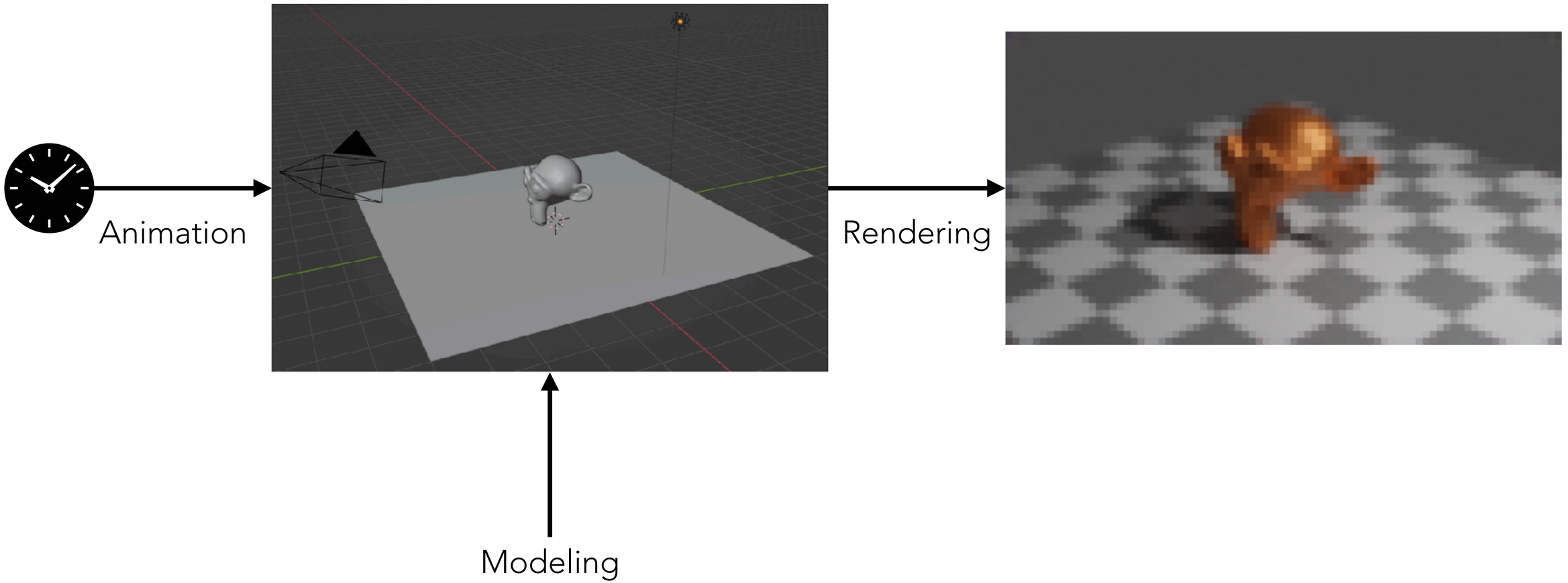
This is called **path tracing**.

Each sample is tracing **one** possible path between the eye and a light source

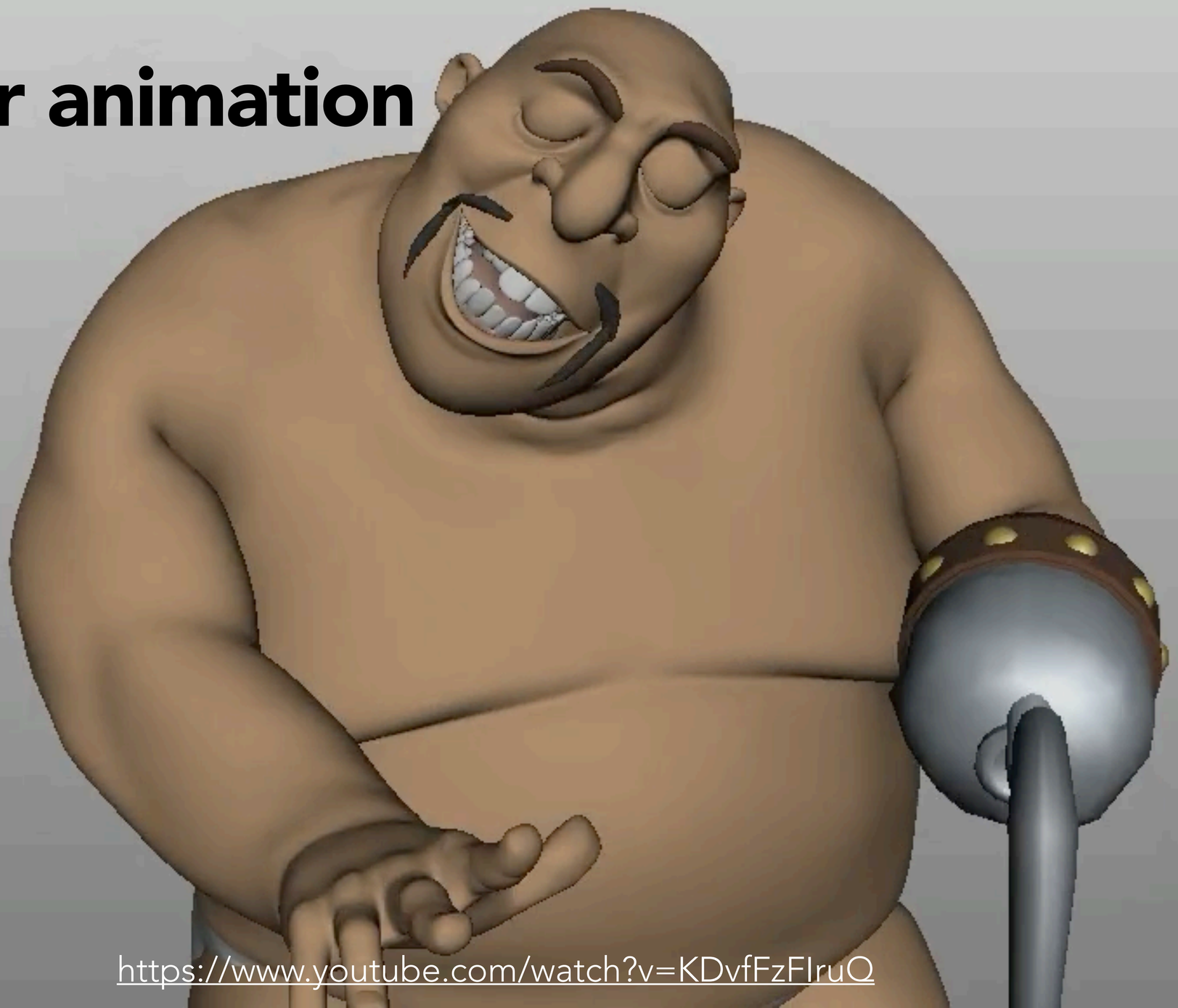



1 spp
2 spp
4 spp
8 spp
16 spp

Animation

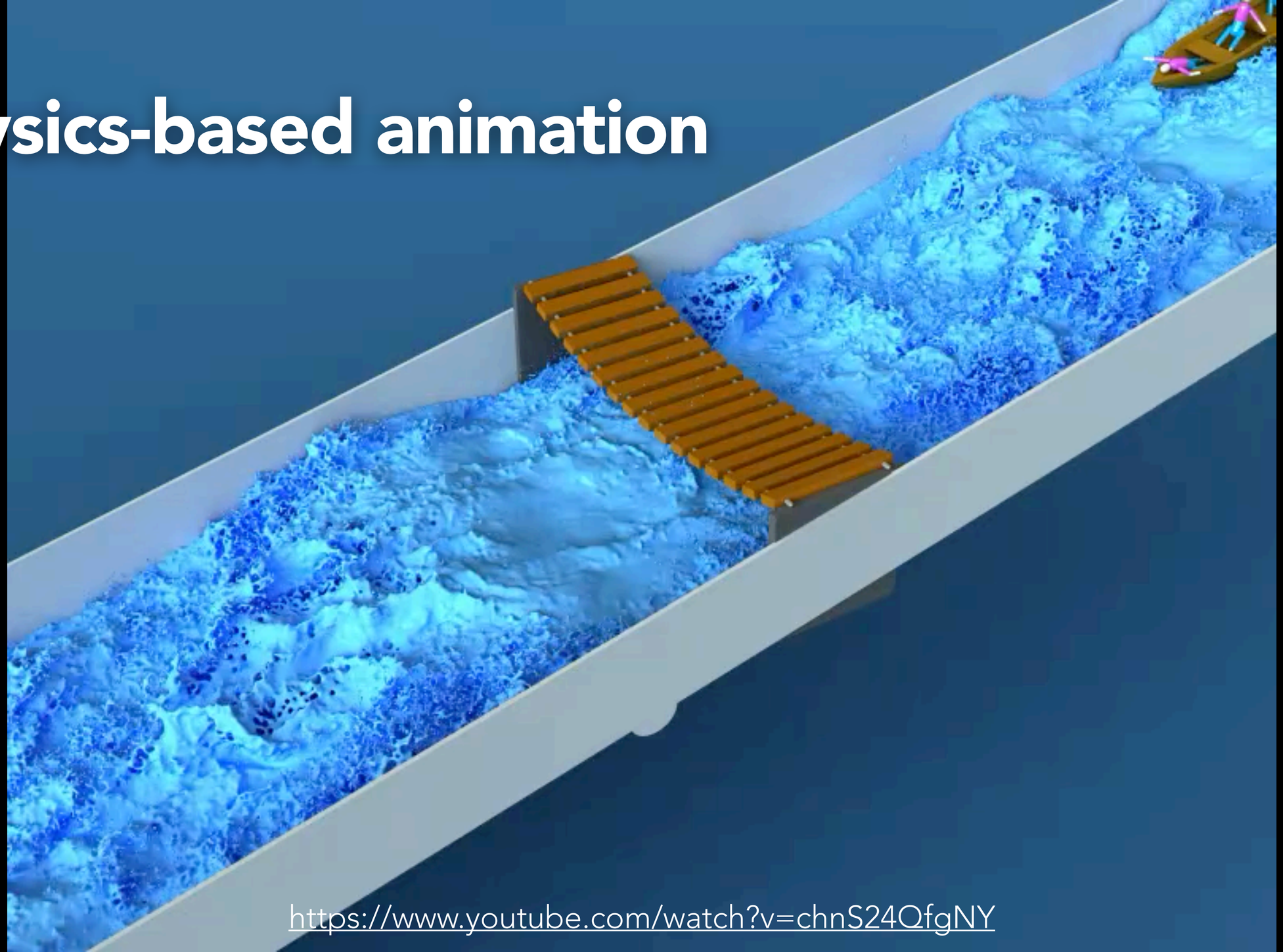

Animation
Rendering
Modeling

Character animation



https://www.youtube.com/watch?v=KDvfFzFIruQ

Physics-based animation



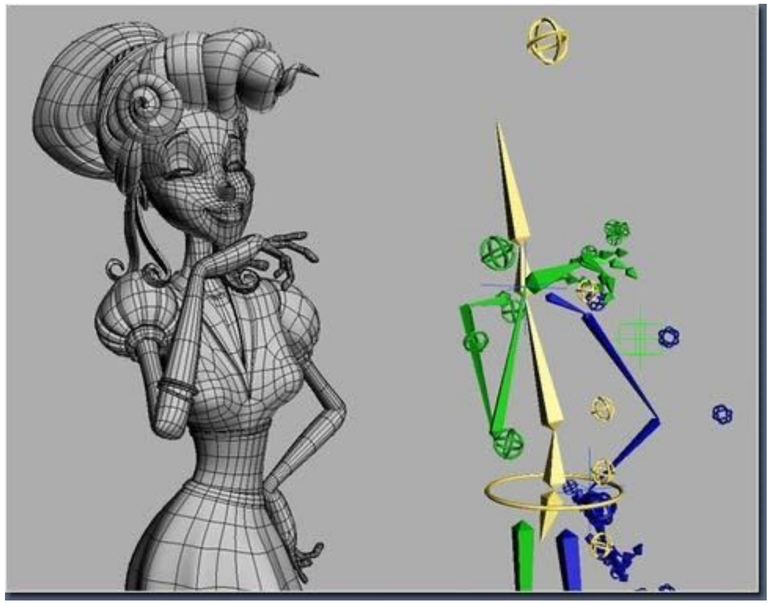
https://www.youtube.com/watch?v=chnS24QfgNY

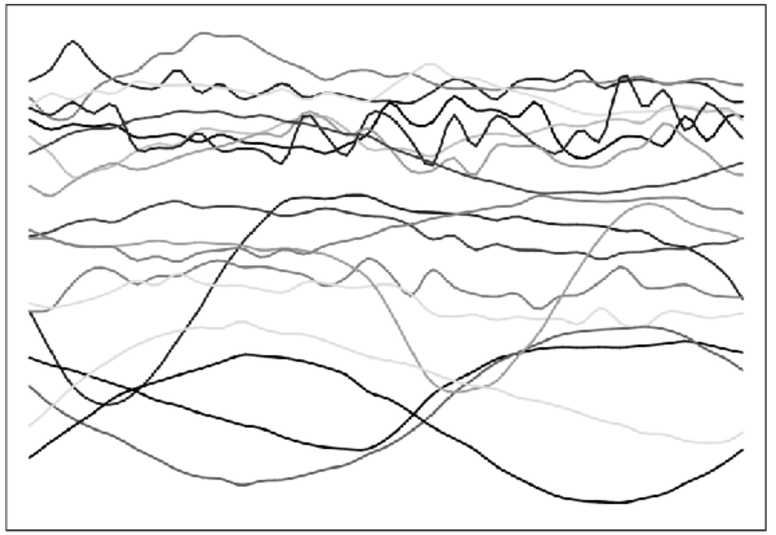
Akinci et al. 2012

Animation is defined through a set of **animation controls** (**degrees of freedom**) whose values vary with time

For example:

- Character: joint angles, etc.
- Rigid body: translation and rotation
- Liquid: position/velocity of all particles(!)





Types of animation techniques

More artistic control

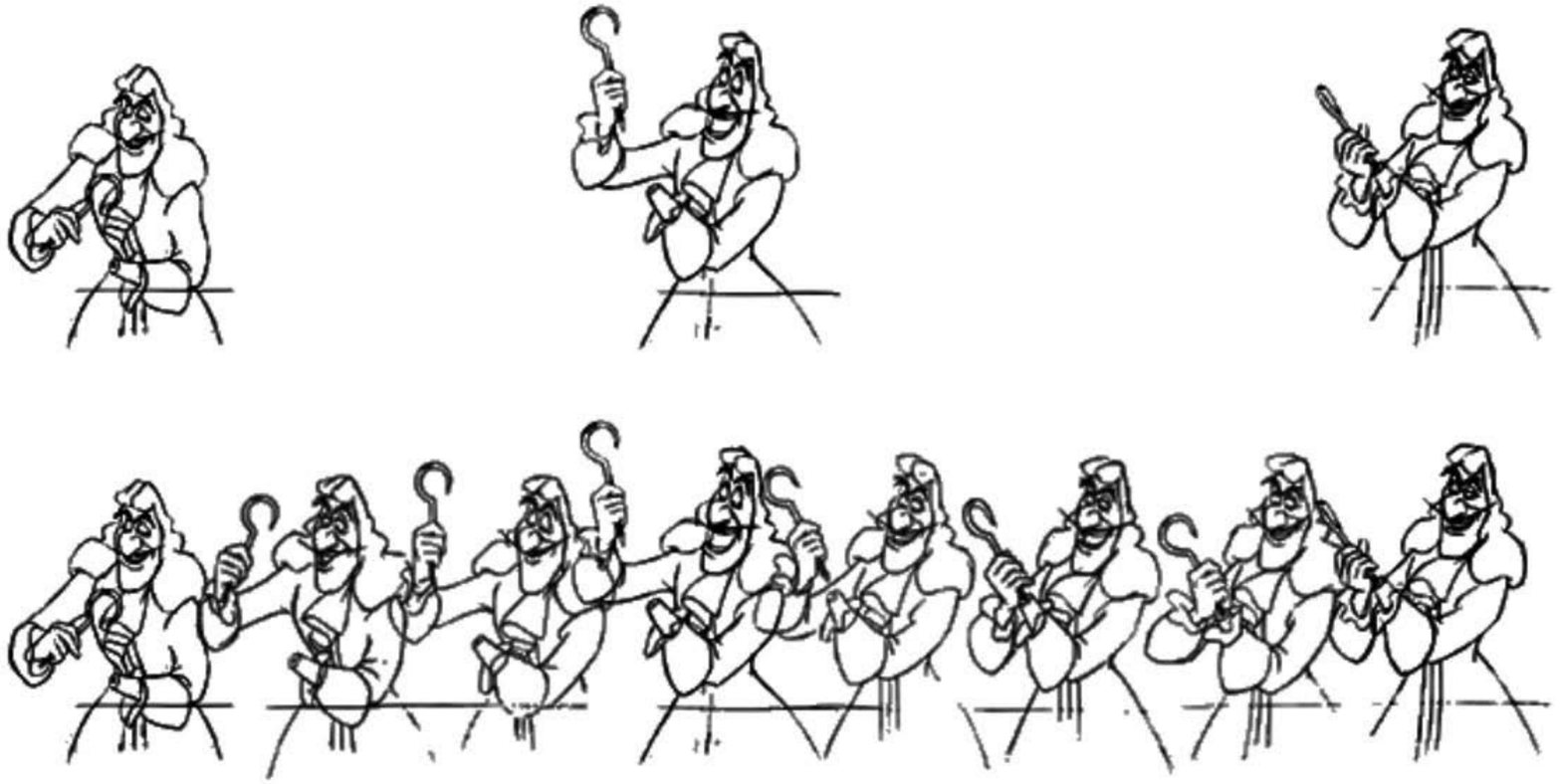
- Artist-specified
  (e.g. keyframing)
- Data-driven
  (e.g. motion capture)
- Procedural
  (e.g. simulation)

Less manual effort

Keyframe animation

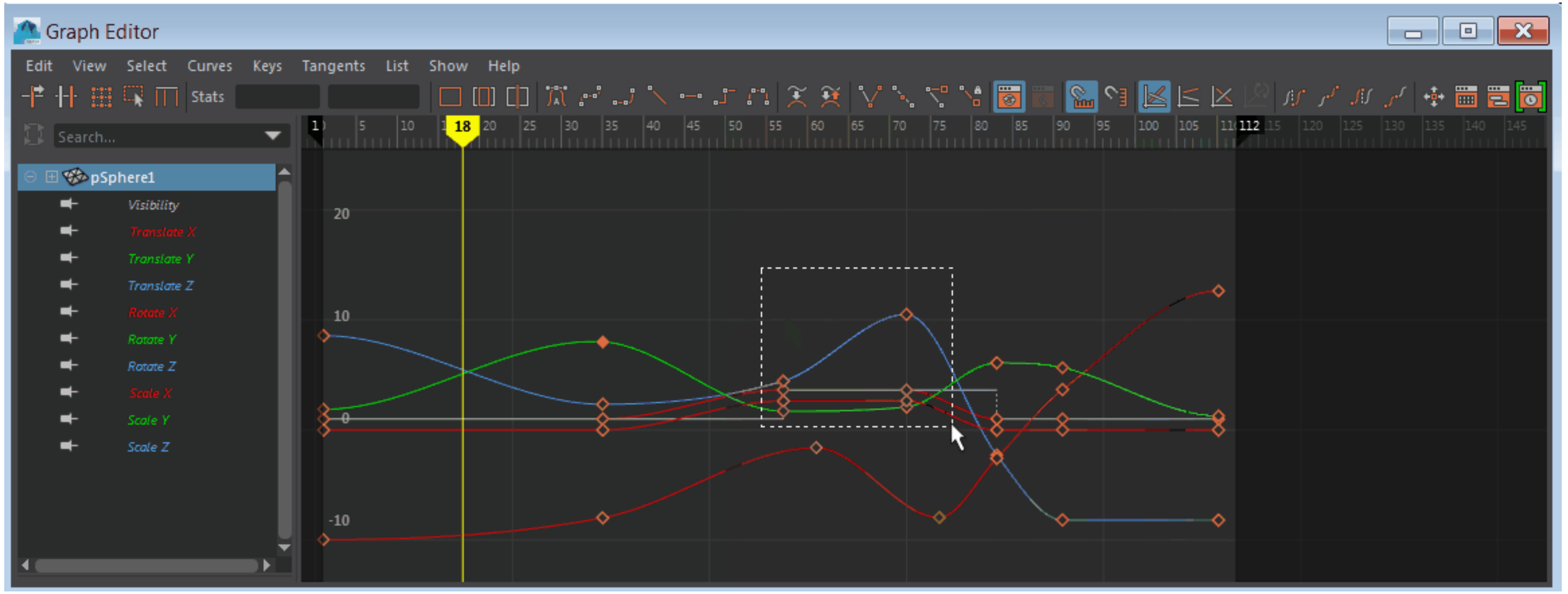
In traditional (hand-drawn animation:

- Lead animator creates **keyframes**
- Assistant creates in-between frames (“**tweening**”)

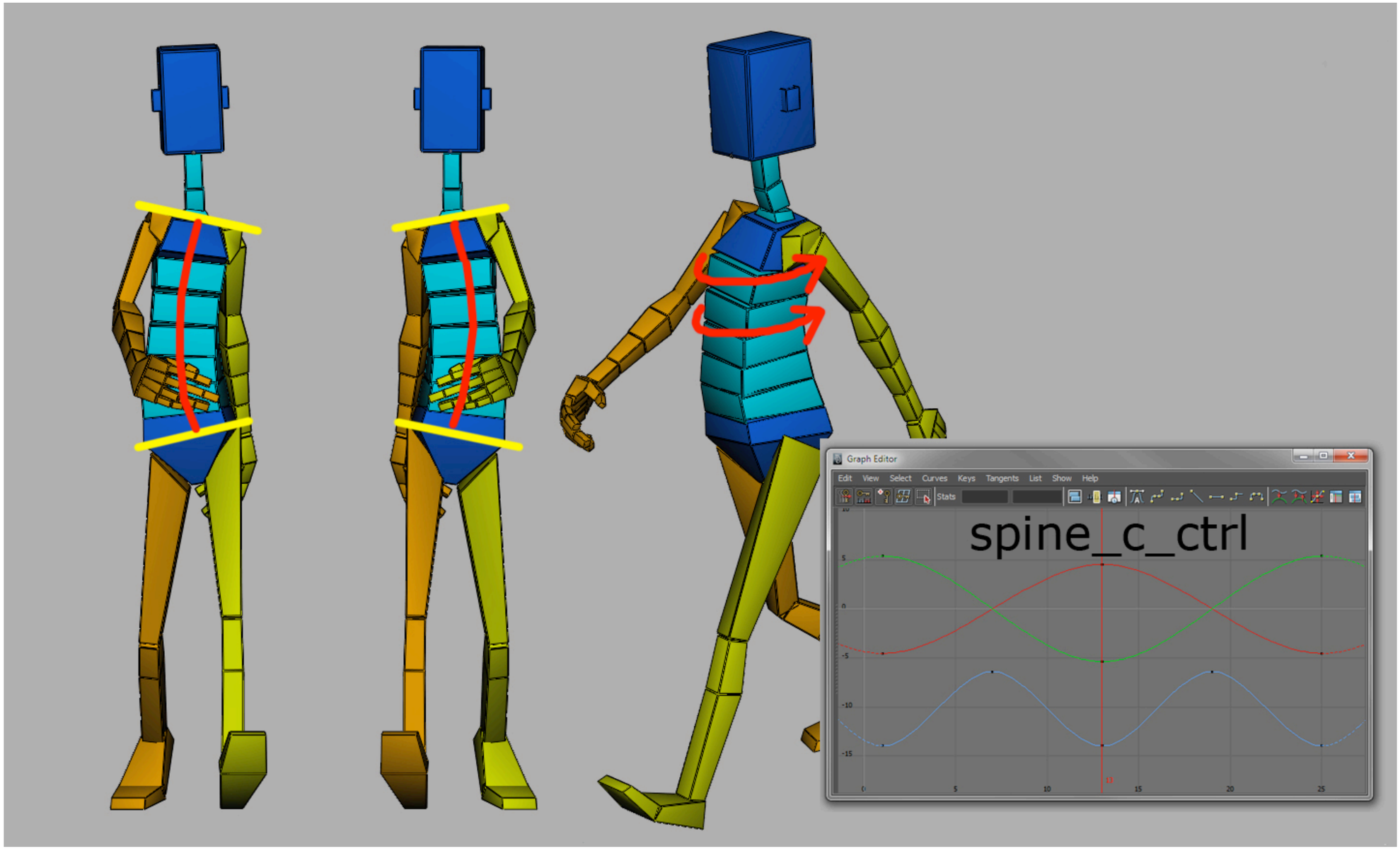


Thomas & Johnston, *The Illusion of Life*

In computer animation, keyframes = control points, tweening = splines!



Autodesk Maya's Graph Editor


Graph Editor
Edit View Select Curves Keys Tangents List Show Help
Stats
spine_c_ctrl

Motion capture


Optical


Magnetic


Mechanical

CAESAR

https://www.youtube.com/watch?v=4NU9ikjqjC0

Physics-based animation (a.k.a. simulation)

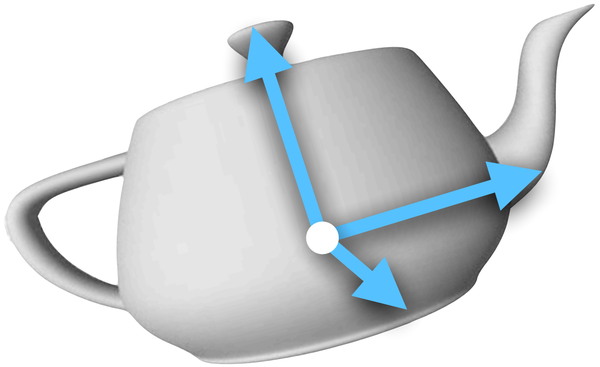
Solve the **equations of motion** to automatically get physically realistic motion.

e.g. **Rigid bodies**

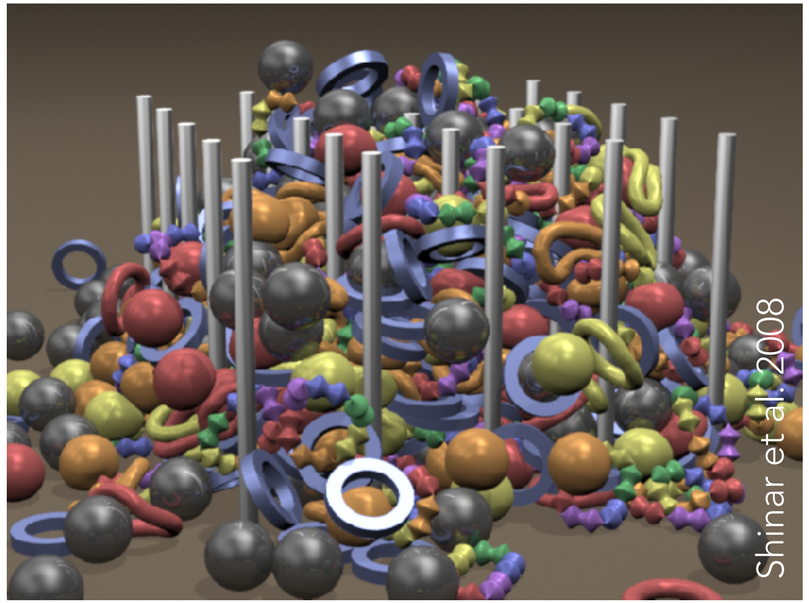
- Degrees of freedom: position, rotation

$$\frac{\mathrm{d}^2\mathbf{x}}{\mathrm{d}t^2} = \mathbf{f}_{\text{ext}}/m$$

$$\frac{\mathrm{d}^2\mathbf{R}}{\mathrm{d}t^2} = \cdots$$



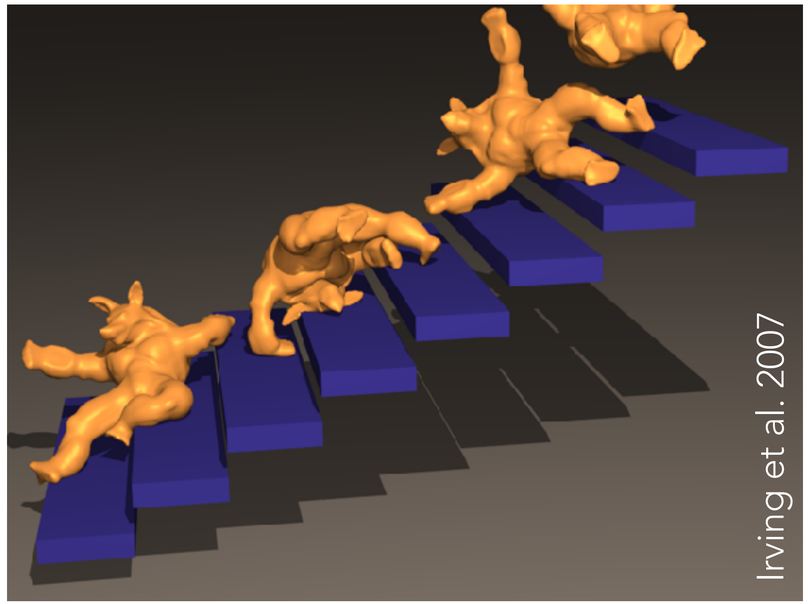
- Challenges: collisions, frictional contact, stacking



Deformable bodies, cloth, etc.

Every vertex can move independently! But deformation causes internal elastic forces

- Physically accurate: **finite element method**
- Simplified approximation: **mass-spring systems**
  (just a bunch of particles
  connected by springs)

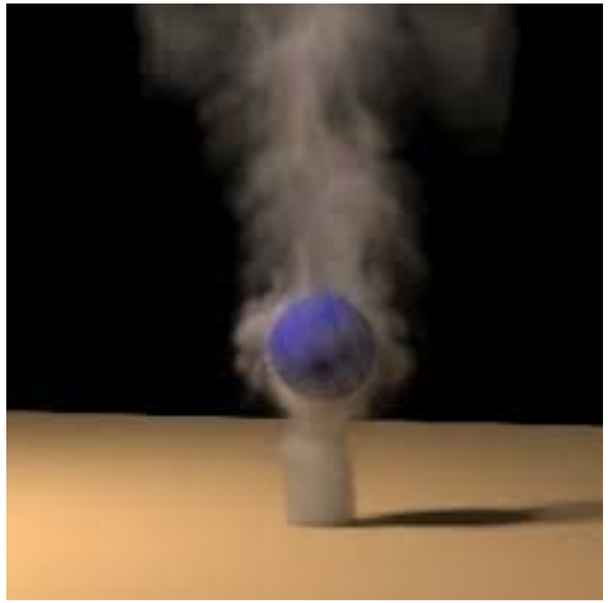


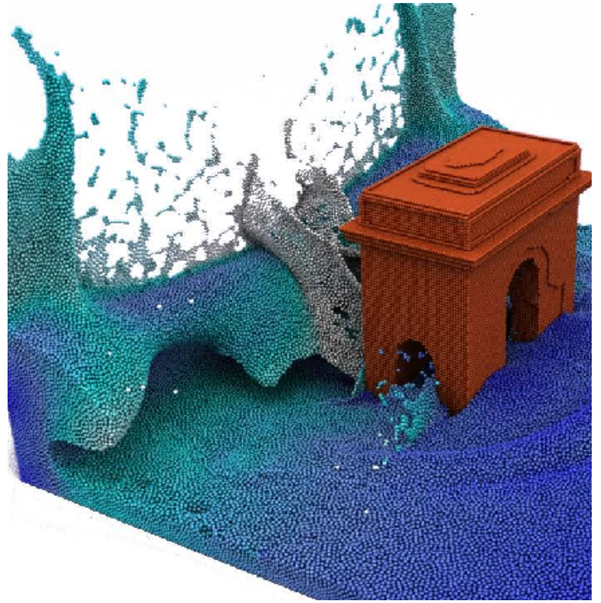
Fluids (smoke, water, fire, etc.)

Described by systems of partial differential equations

Velocity field $\mathbf{v}(\mathbf{x}, t)$: every point has its own velocity!

$$\frac{\partial \mathbf{v}}{\partial t} = \left[\ldots \text{ something involving } \mathbf{v}, \frac{\partial \mathbf{v}}{\partial \mathbf{x}}, \text{ etc. } \ldots \right]$$





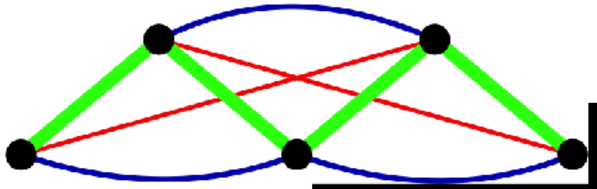
Physics in character animation

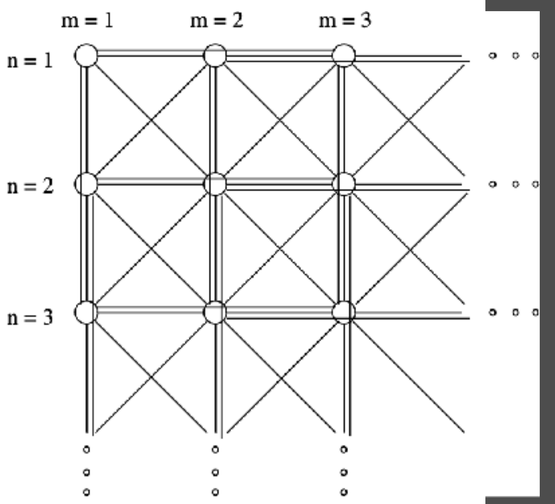
- Flesh
- Hair
- Clothing
- …

https://vimeo.com/245424174

Smith et al. 2018

Mass-spring systems





Recall springs in 1 dimension from physics classes.

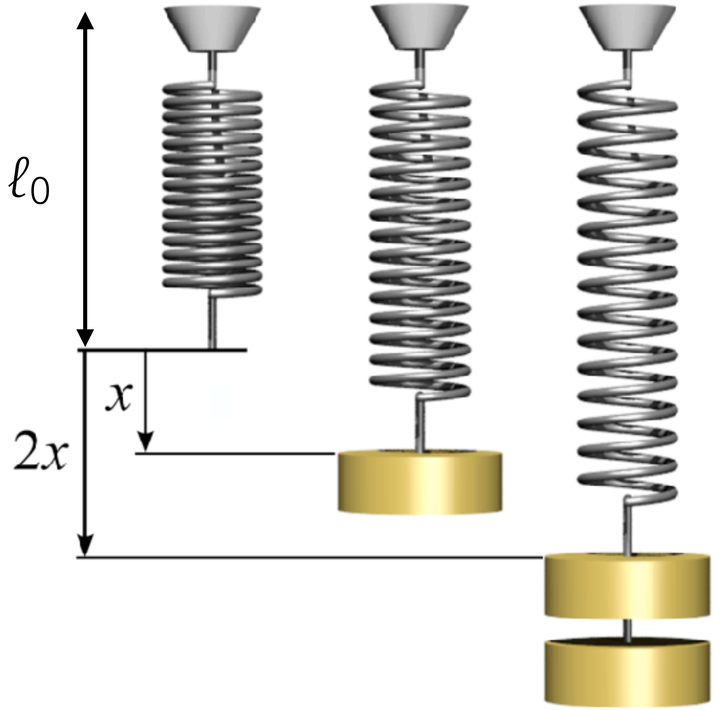
Hooke's law: force is proportional to displacement

$$F = -k\,x = -k\,(\ell - \ell_0)$$

Potential energy:

$$U = \tfrac{1}{2}\,k\,(\ell - \ell_0)^2$$

In fact $F = -\mathrm{d}U/\mathrm{d}\ell$



In 3D, suppose a spring connects particles i and j. What should be the force $\mathbf{f}_{ij}$ on i due to j?

Let's first define the potential:

$$U = \tfrac{1}{2}\, k\, (\|\mathbf{x}_i - \mathbf{x}_j\| - \ell_0)^2$$

Then $\mathbf{f}_{ij} = -\partial U / \partial \mathbf{x}_i \Rightarrow$

$$\mathbf{f}_{ij} = -k\, (\|\mathbf{x}_i - \mathbf{x}_j\| - \ell_0)\, \frac{\mathbf{x}_i - \mathbf{x}_j}{\|\mathbf{x}_i - \mathbf{x}_j\|}$$

$$= -k\, (\|\mathbf{x}_{ij}\| - \ell_0)\, \hat{\mathbf{x}}_{ij}$$

Similarly $\mathbf{f}_{ji} = -\partial U / \partial \mathbf{x}_j$ (but it's also just $-\mathbf{f}_{ij}$)

Exercise: Derive this expression from $-\partial U / \partial \mathbf{x}_i$. Optional: Look up multivariable calculus identities, chain rule, etc. so you don't have to differentiate componentwise.

Time stepping

Equations of motion:

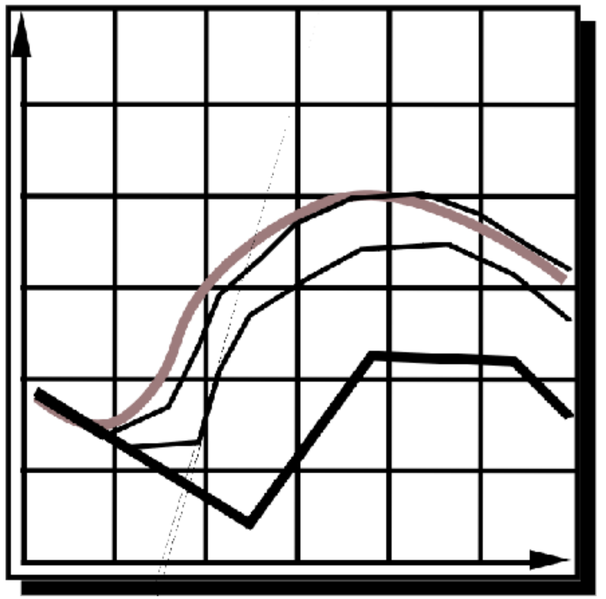
$$\frac{\mathrm{d}\mathbf{x}_i}{\mathrm{d}t} = \mathbf{v}_i \qquad \frac{\mathrm{d}\mathbf{v}_i}{\mathrm{d}t} = m_i^{-1} \sum \mathbf{f}_{ij}$$

Take small **time steps** from time t_0 to t_1, then t_1 to t_2, and so on…

$$\mathbf{v}_i(t_{n+1}) \approx \mathbf{v}_i(t_n) + \Delta t\, m_i^{-1} \sum \mathbf{f}_{ij}(t_n)$$

$$\mathbf{x}_i(t_{n+1}) \approx \mathbf{x}_i(t_n) + \Delta t\, \mathbf{v}_i(t_{n+1})$$

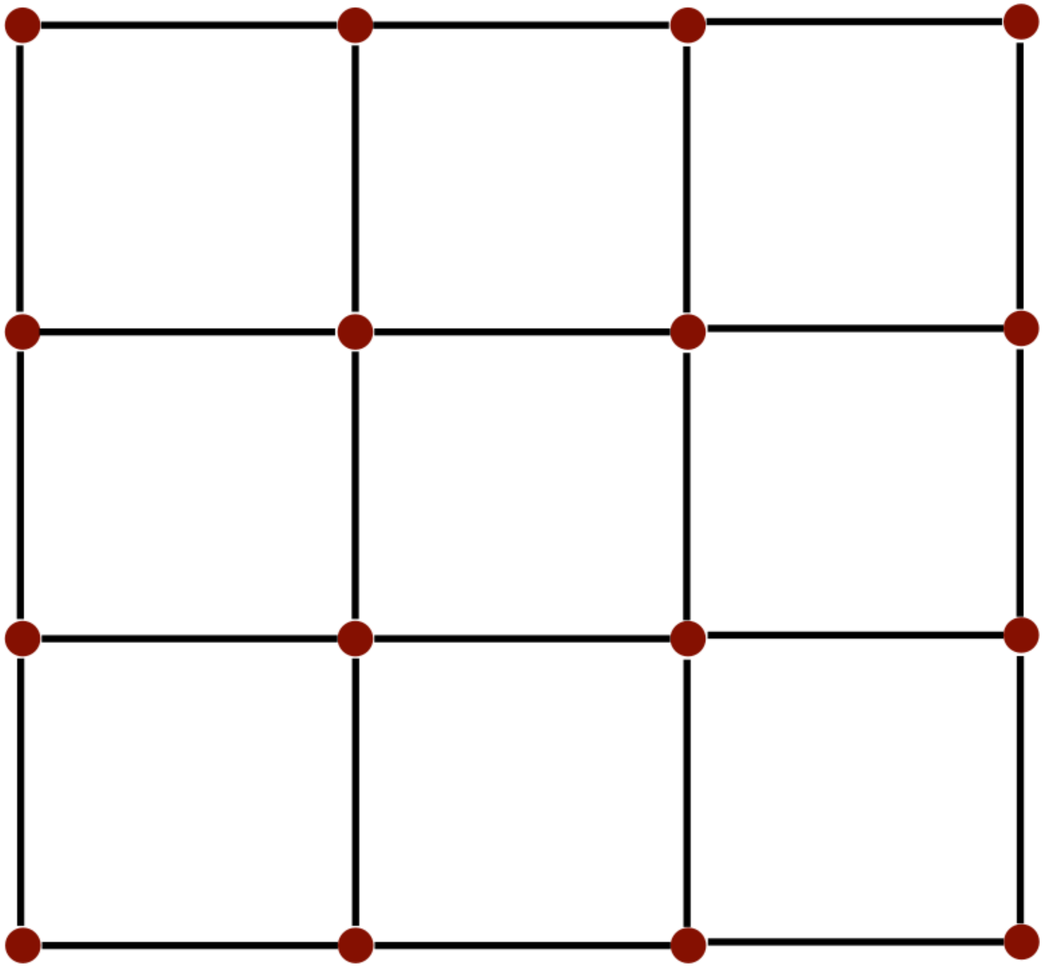
Tradeoff: Smaller Δt → more accurate, but more computation to reach any desired t.



Witkin & Baraff 2001

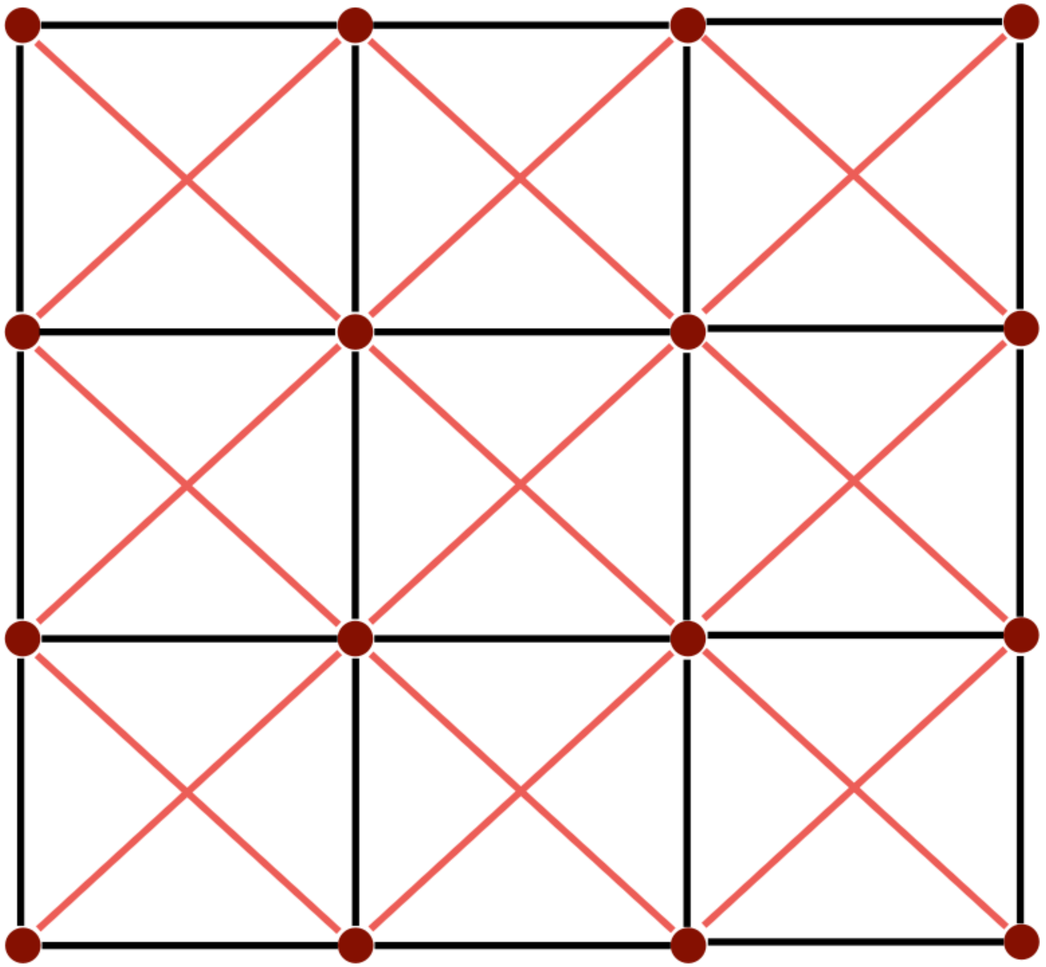
How to model a rectangular sheet of cloth?

- Structural springs



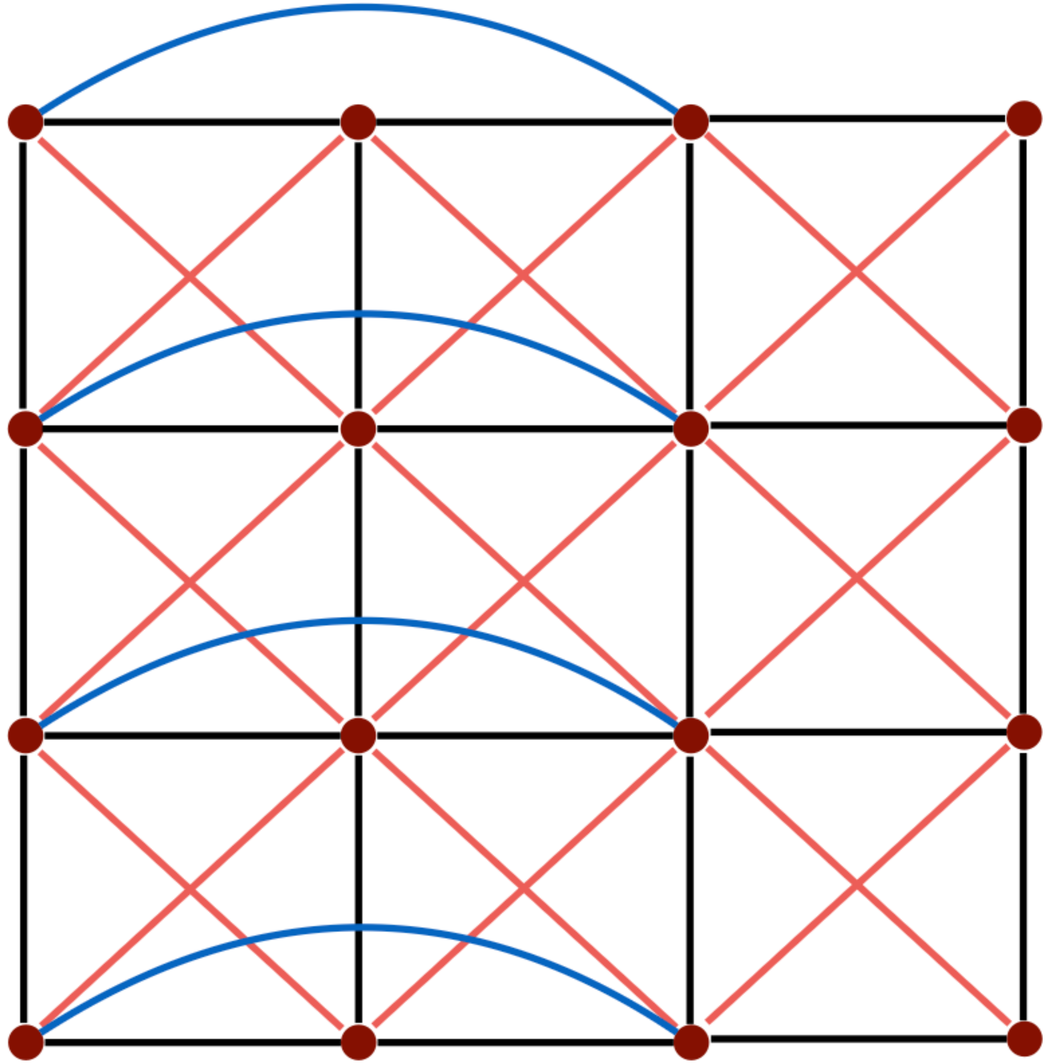
How to model a rectangular sheet of cloth?

- Structural springs
- Shear springs



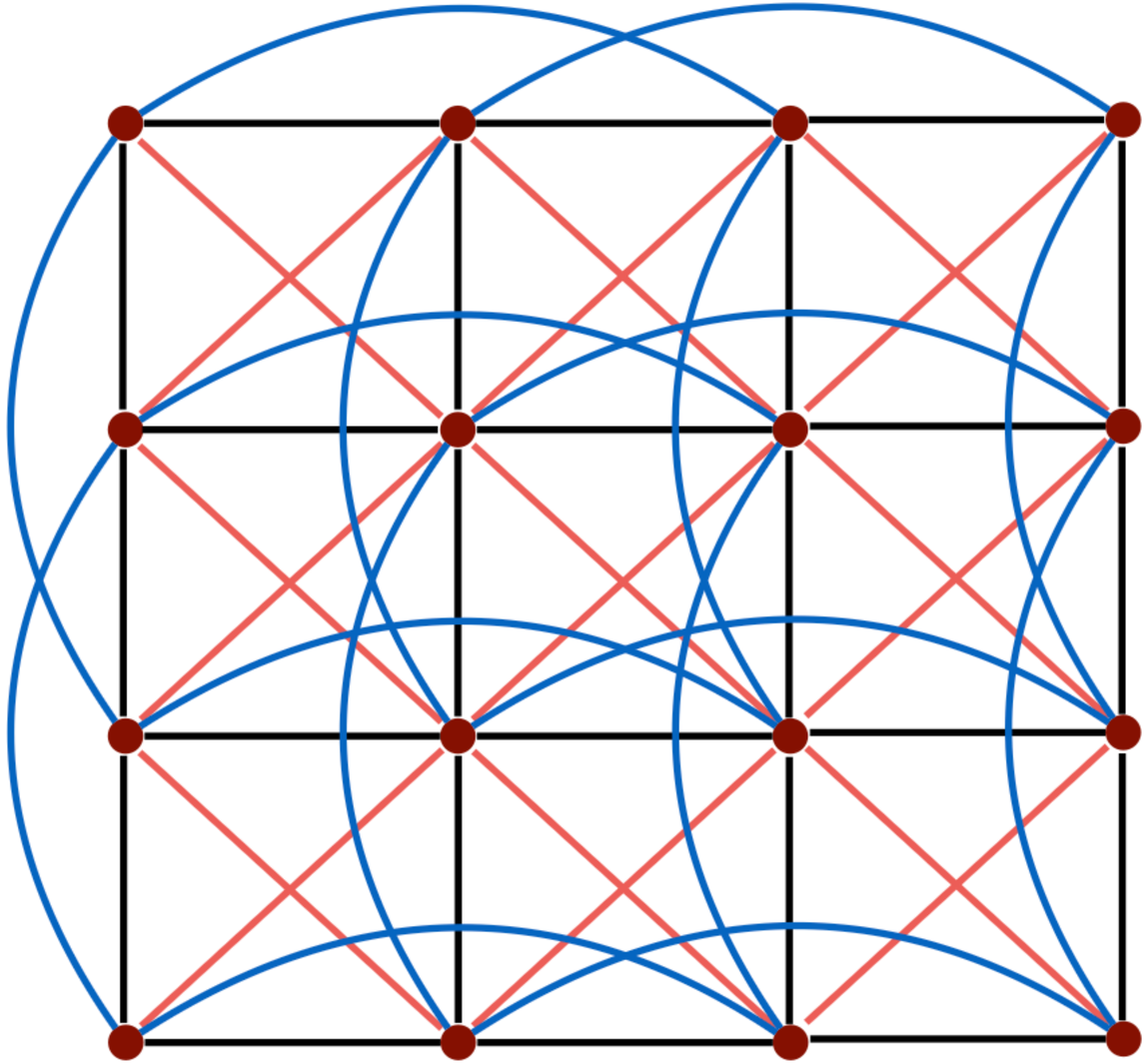
How to model a rectangular sheet of cloth?

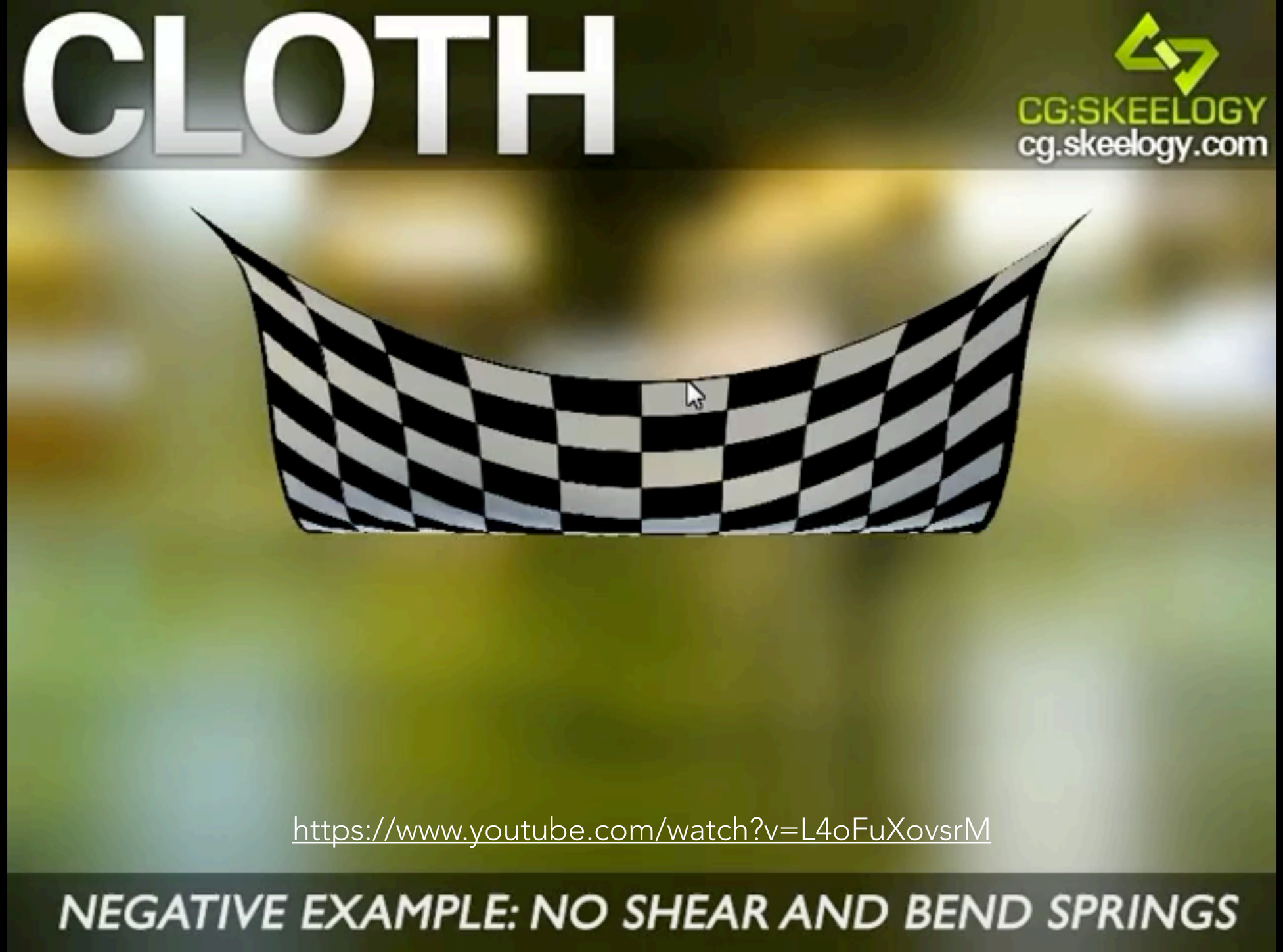
- Structural springs
- Shear springs
- Bending springs



How to model a rectangular sheet of cloth?

- Structural springs
- Shear springs
- Bending springs

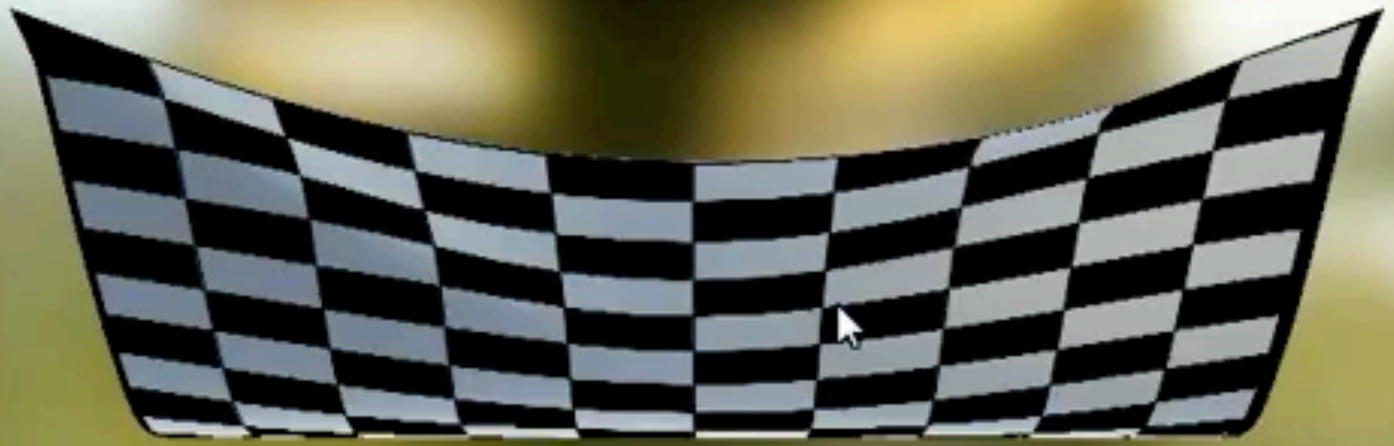



CLOTH
CG:SKEELOGY
cg.skeelogy.com
https://www.youtube.com/watch?v=L4oFuXovsrM
NEGATIVE EXAMPLE: NO SHEAR AND BEND SPRINGS

Skeel Lee

CLOTH

CG:SKEELOGY
cg.skeelogy.com



https://www.youtube.com/watch?v=RMqgajfZSvY

NEGATIVE EXAMPLE: NO BEND SPRINGS

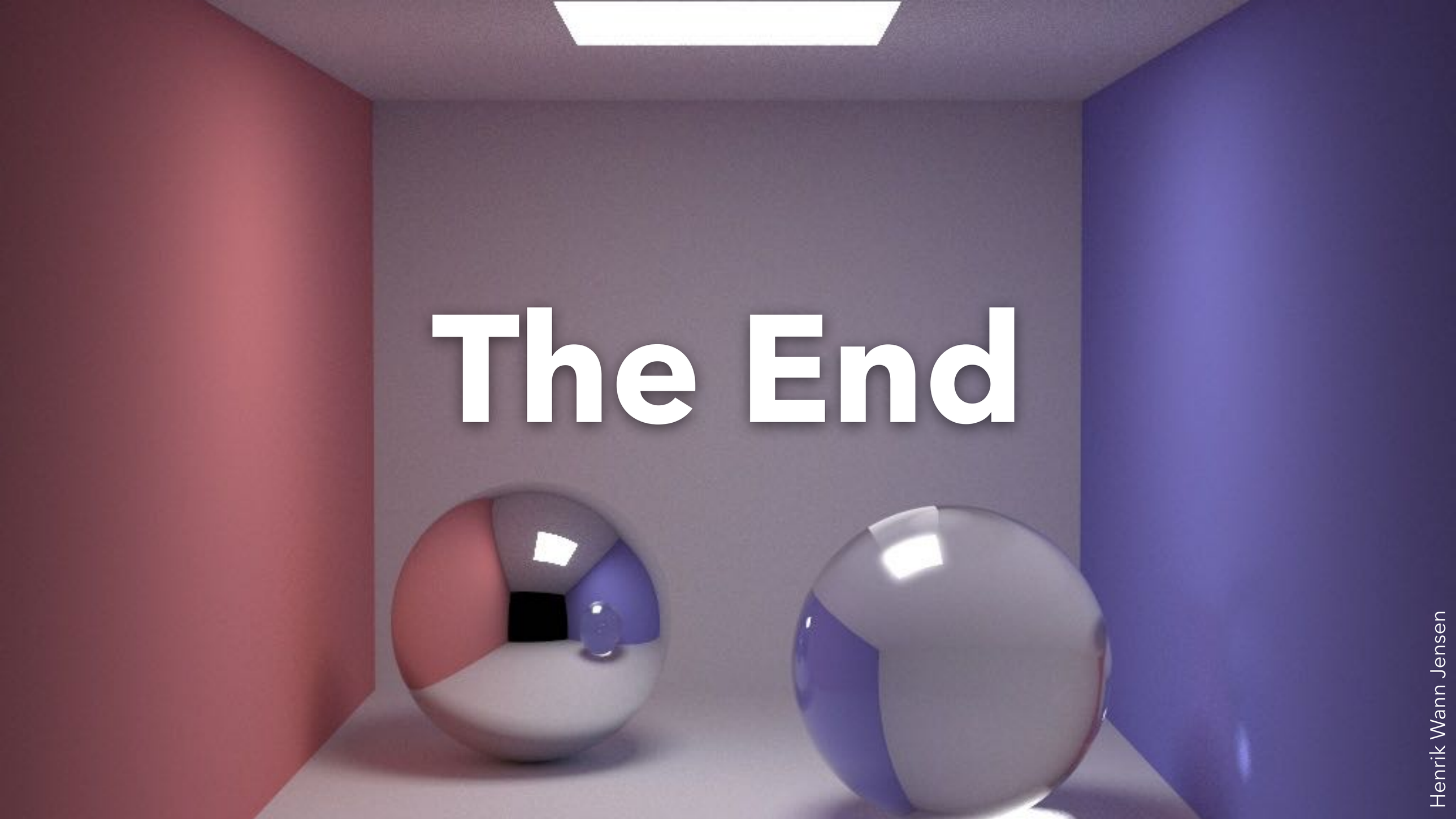
Skeel Lee

CLOTH

CG:SKEELOGY
cg.skeelogy.com

https://www.youtube.com/watch?v=JncQOCg9FS0

Skeel Lee


The End
Henrik Wann Jensen