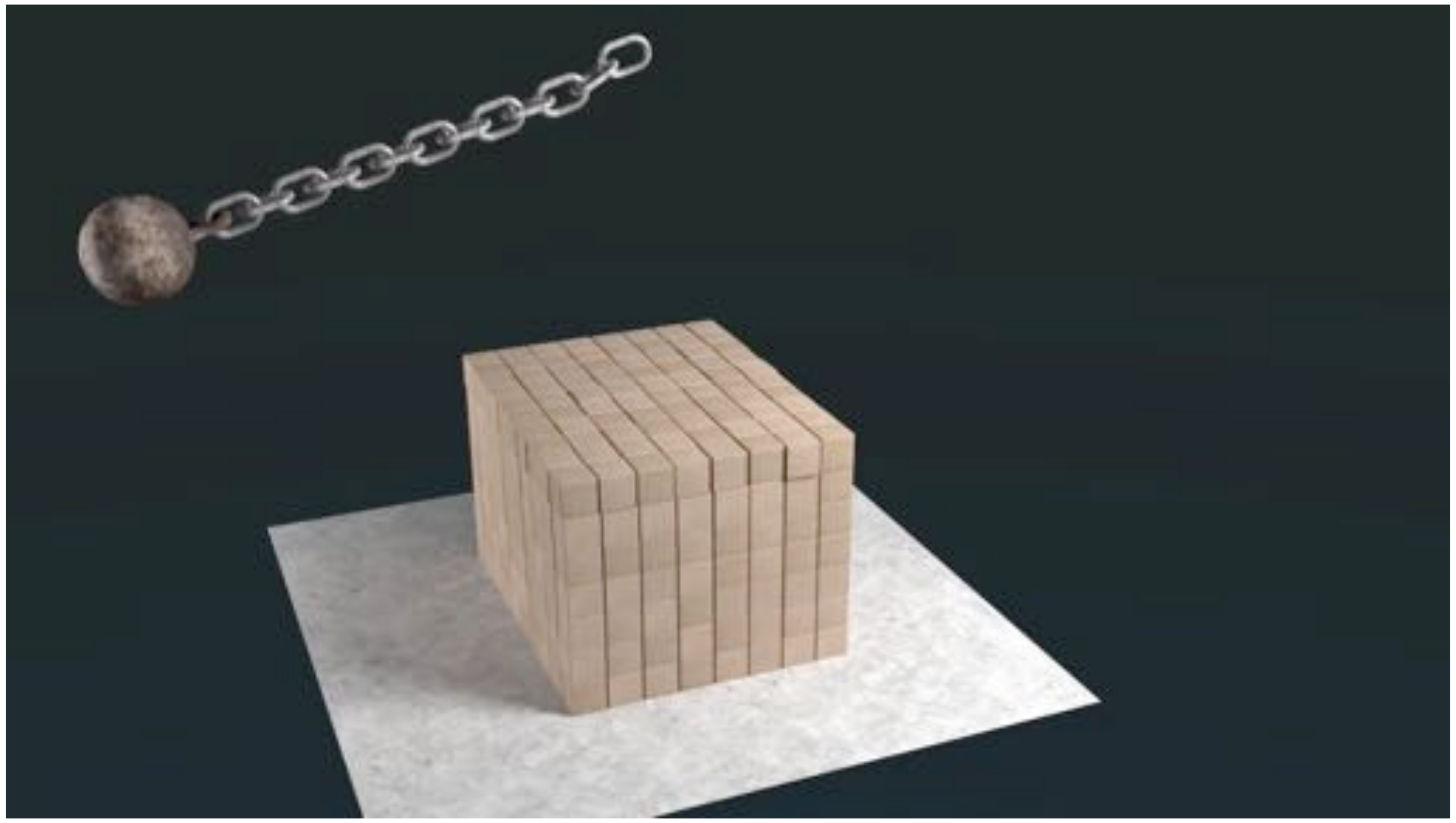
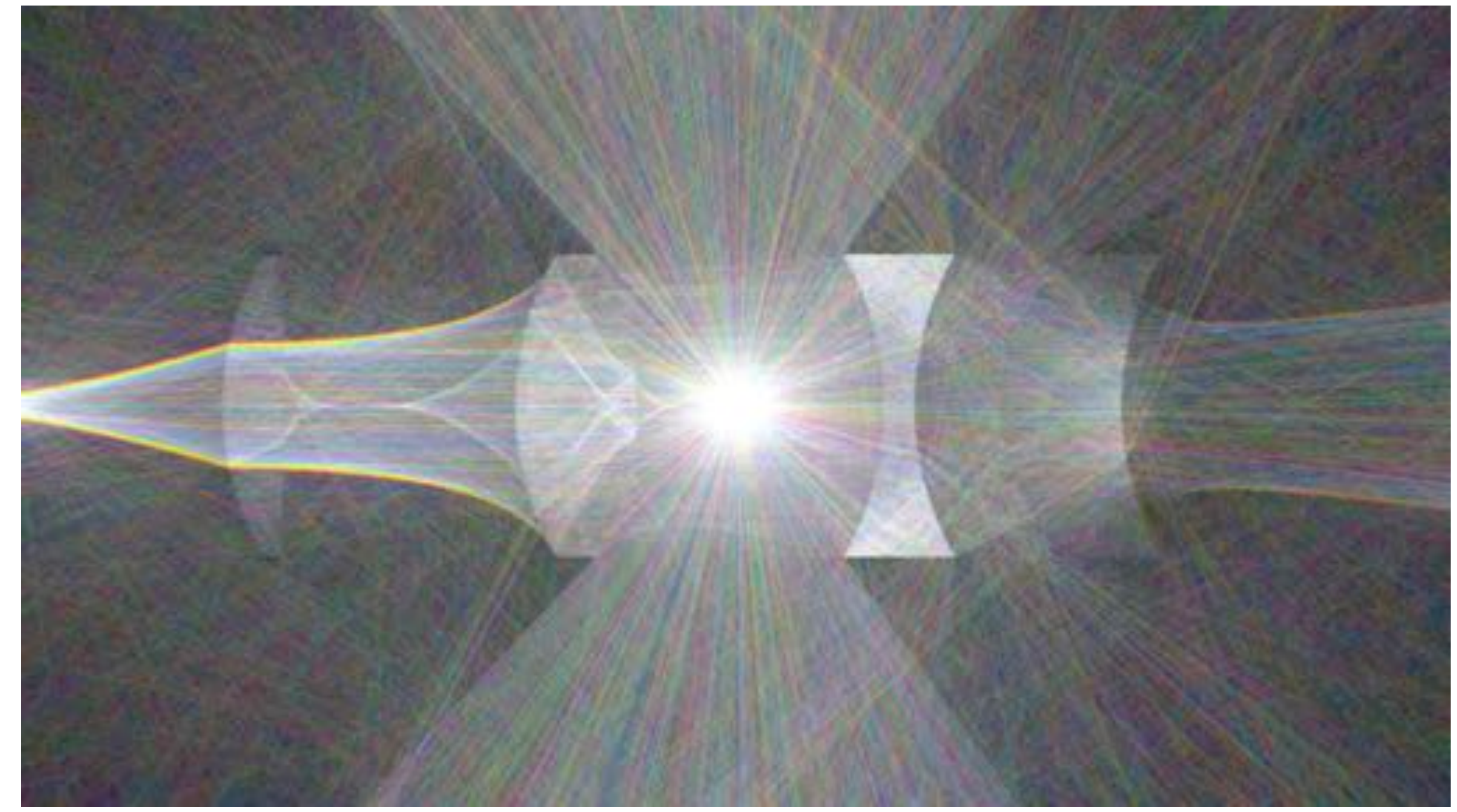
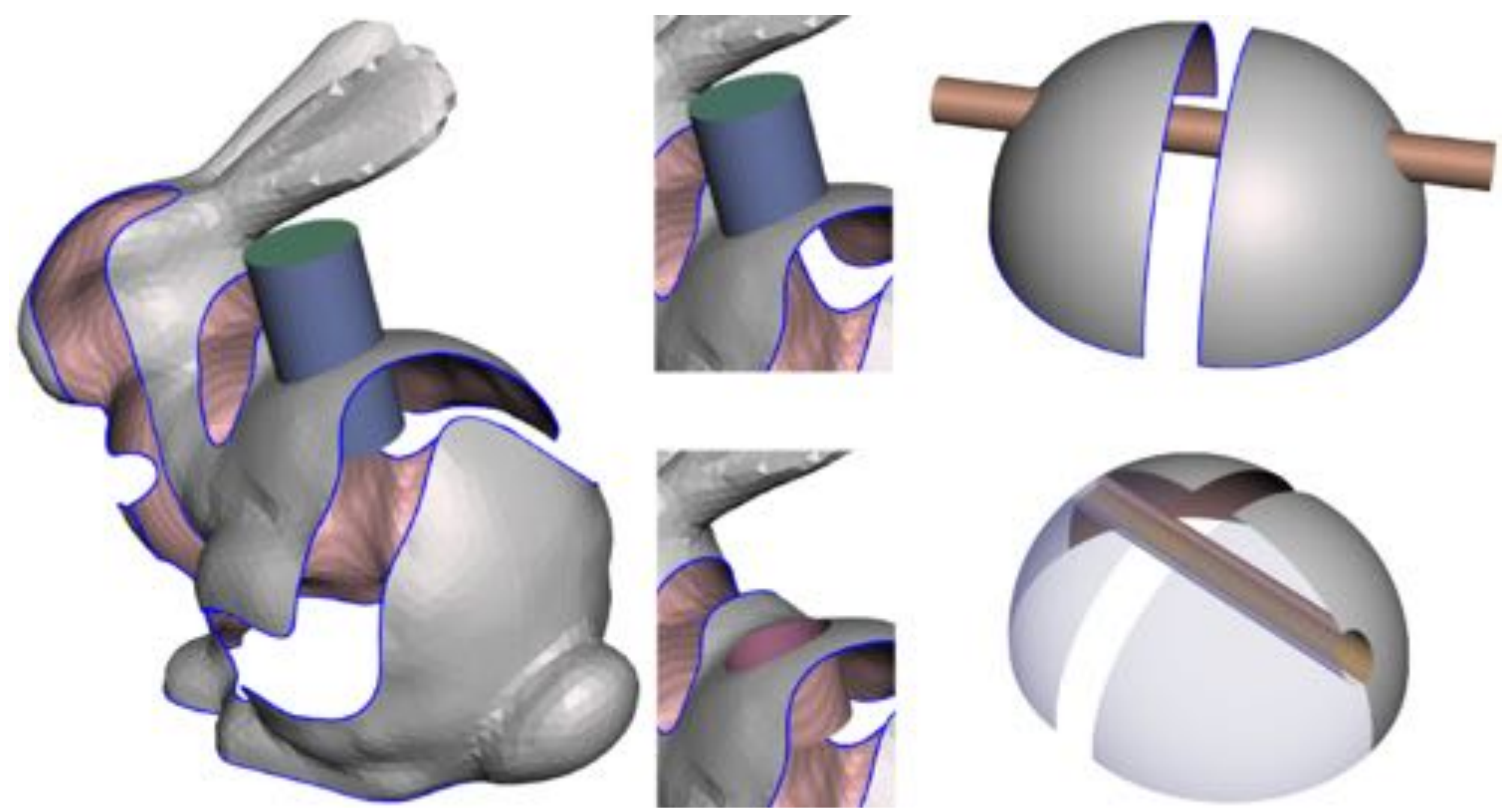
Geometric Queries

Computer Graphics
CMU 15-462/15-662

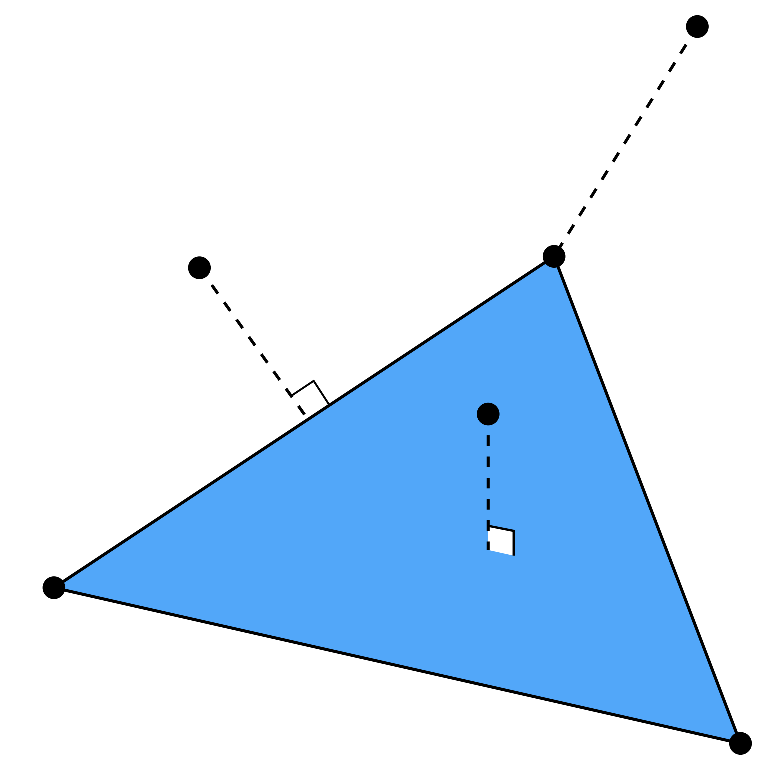
Geometric Queries—Motivation







Many types of geometric queries

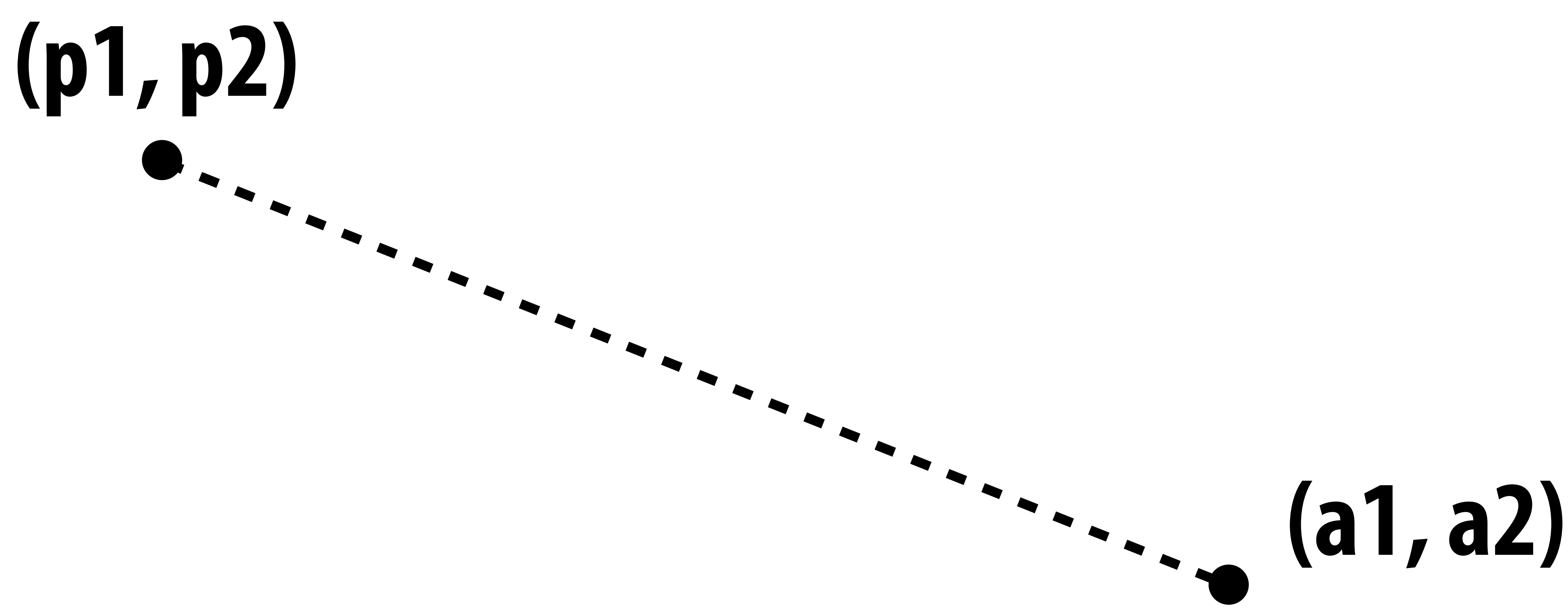


- Already identified need for "closest point" query
- Plenty of other things we might like to know:
  - Do two triangles intersect?
  - Are we inside or outside an object?
  - Does one object contain another?
  - ...

- Data structures we've seen so far not really designed for this...
- Need some new ideas!
- TODAY: come up with simple (read: slow) algorithms.
- NEXT TIME: intelligent ways to accelerate geometric queries.

Warm up: closest point on point

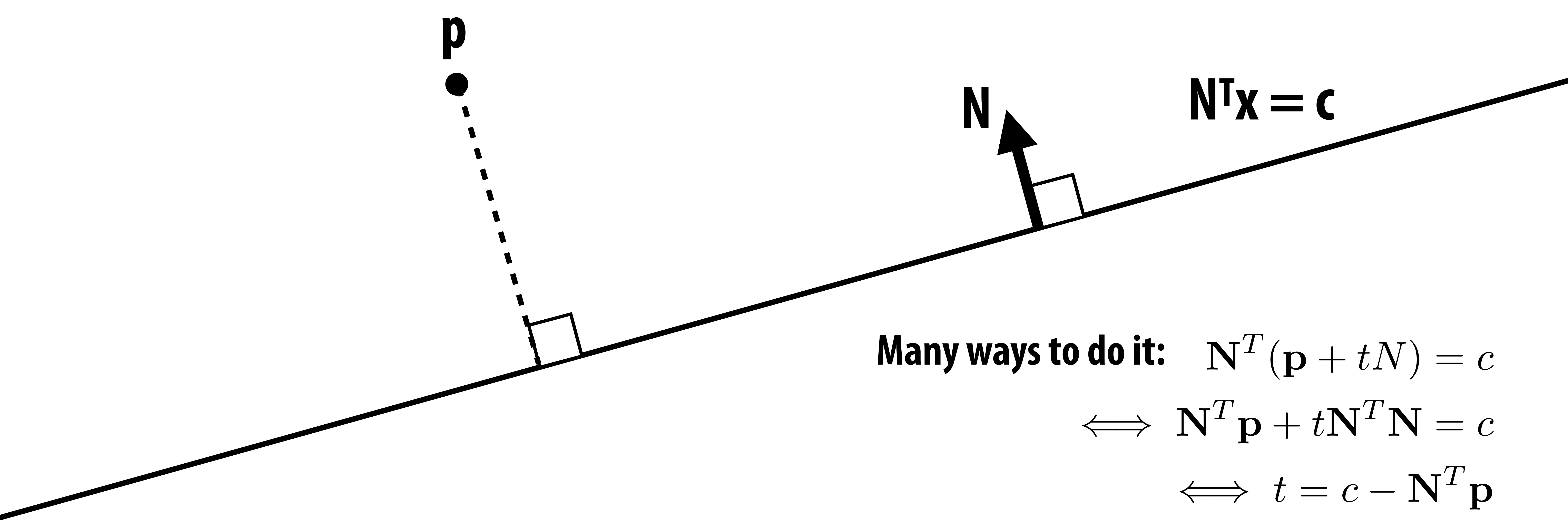
- Goal is to find the point on a mesh closest to a given point.
- Much simpler question: given a query point (p1,p2), how do we find the closest point on the point (a1,a2)?



Bonus question: what's the distance?

Slightly harder: closest point on line

- Now suppose I have a line $\mathbf{N}^\mathsf{T}\mathbf{x} = \mathbf{c}$, where **N** is the unit normal
- How do I find the point closest to my query point **p**?



Many ways to do it:

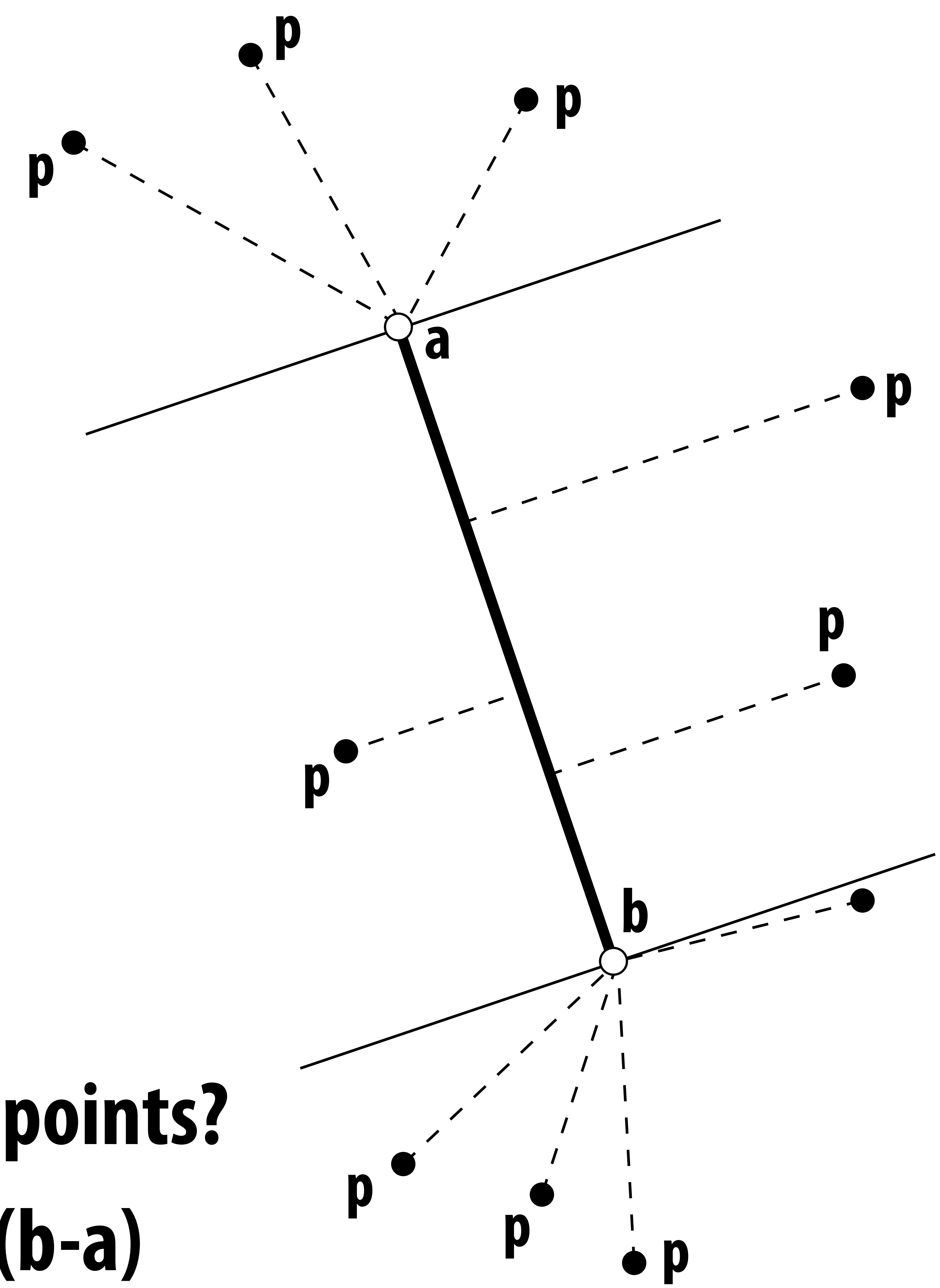
$$\mathbf{N}^T(\mathbf{p} + tN) = c$$

$$\iff \mathbf{N}^T\mathbf{p} + t\mathbf{N}^T\mathbf{N} = c$$

$$\iff t = c - \mathbf{N}^T\mathbf{p}$$

$$\Rightarrow \mathbf{p} + t\mathbf{N} = \boxed{\mathbf{p} + (c - \mathbf{N}^T\mathbf{p})\mathbf{N}}$$

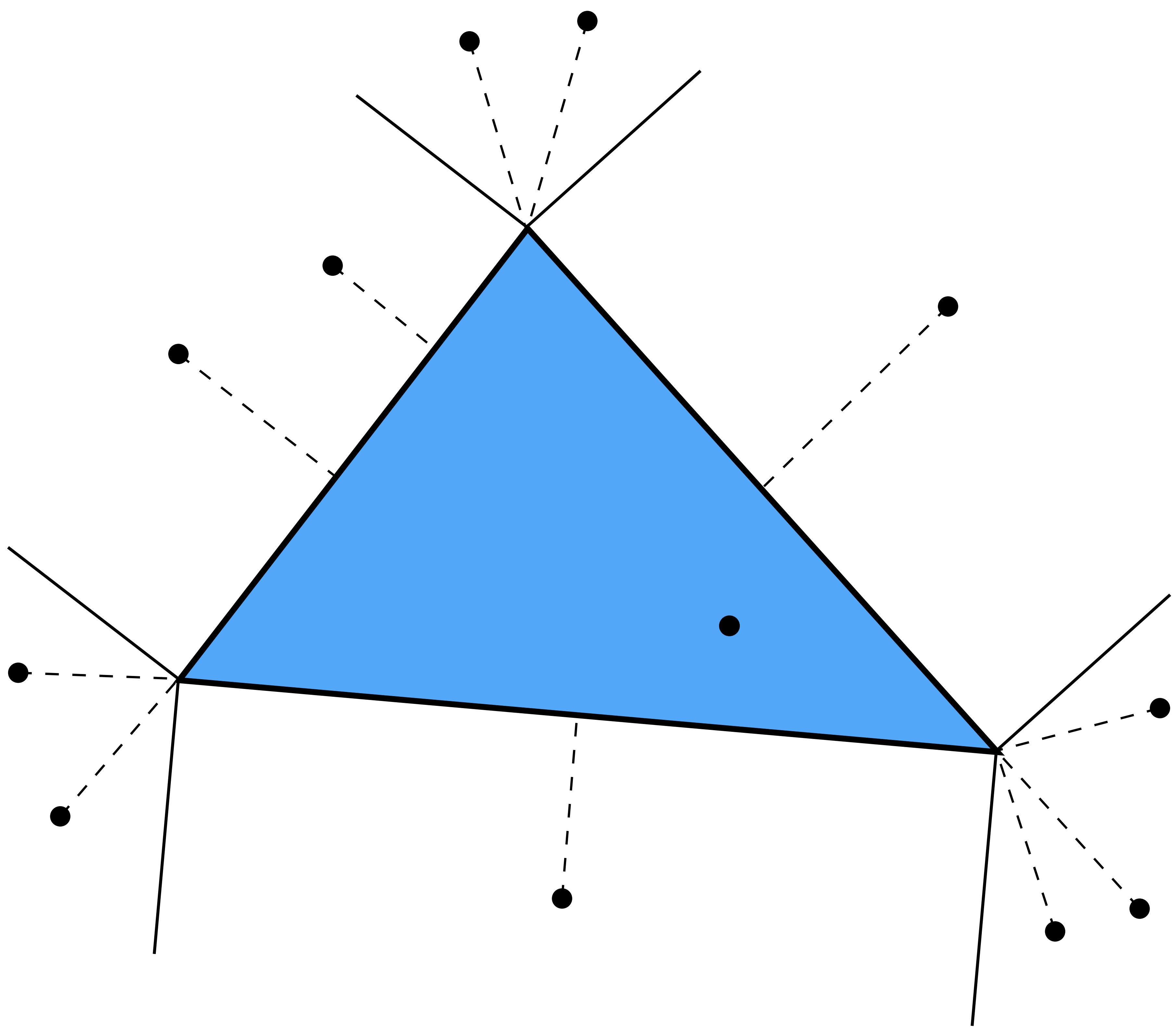
Harder: closest point on line segment



- Two cases: endpoint or interior
- Already have basic components:
  - point-to-point
  - point-to-line
- Algorithm?
  - find closest point on line
  - check if it's between endpoints
  - if not, take closest endpoint
- How do we know if it's between endpoints?
  - write closest point on line as a+t(b-a)
  - if t is between 0 and 1, it's inside the segment!

Even harder: closest point on triangle

- What are all the possibilities for the closest point?
- Almost just minimum distance to three segments:



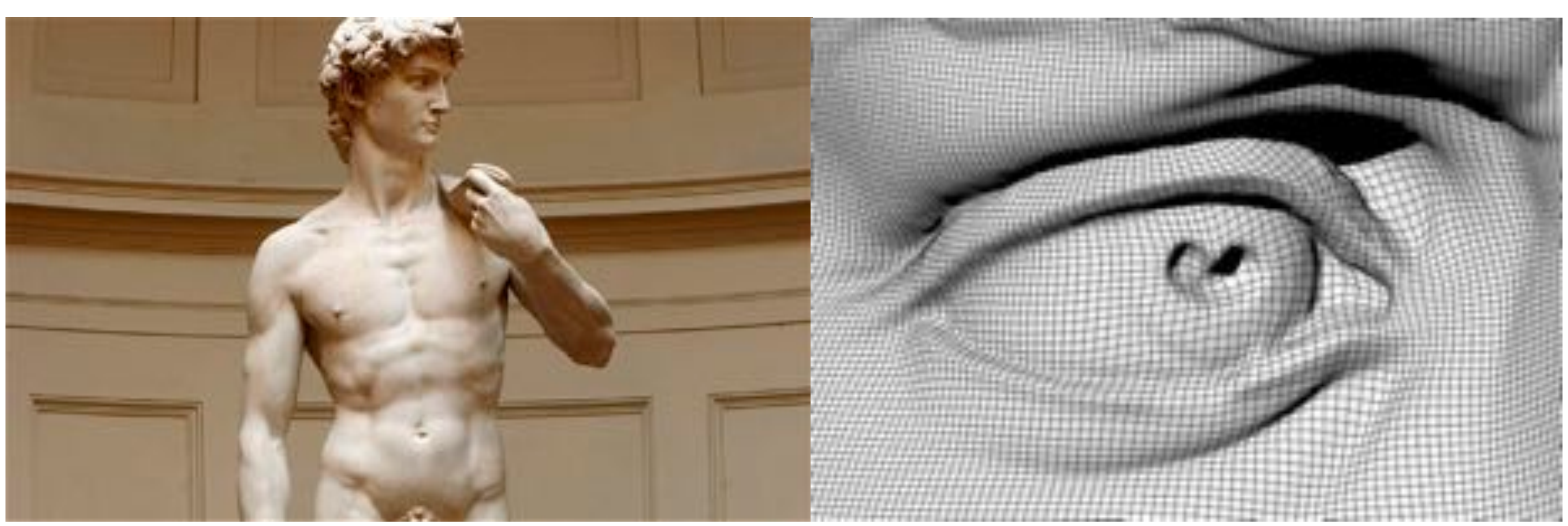
Q: What about a point inside the triangle?

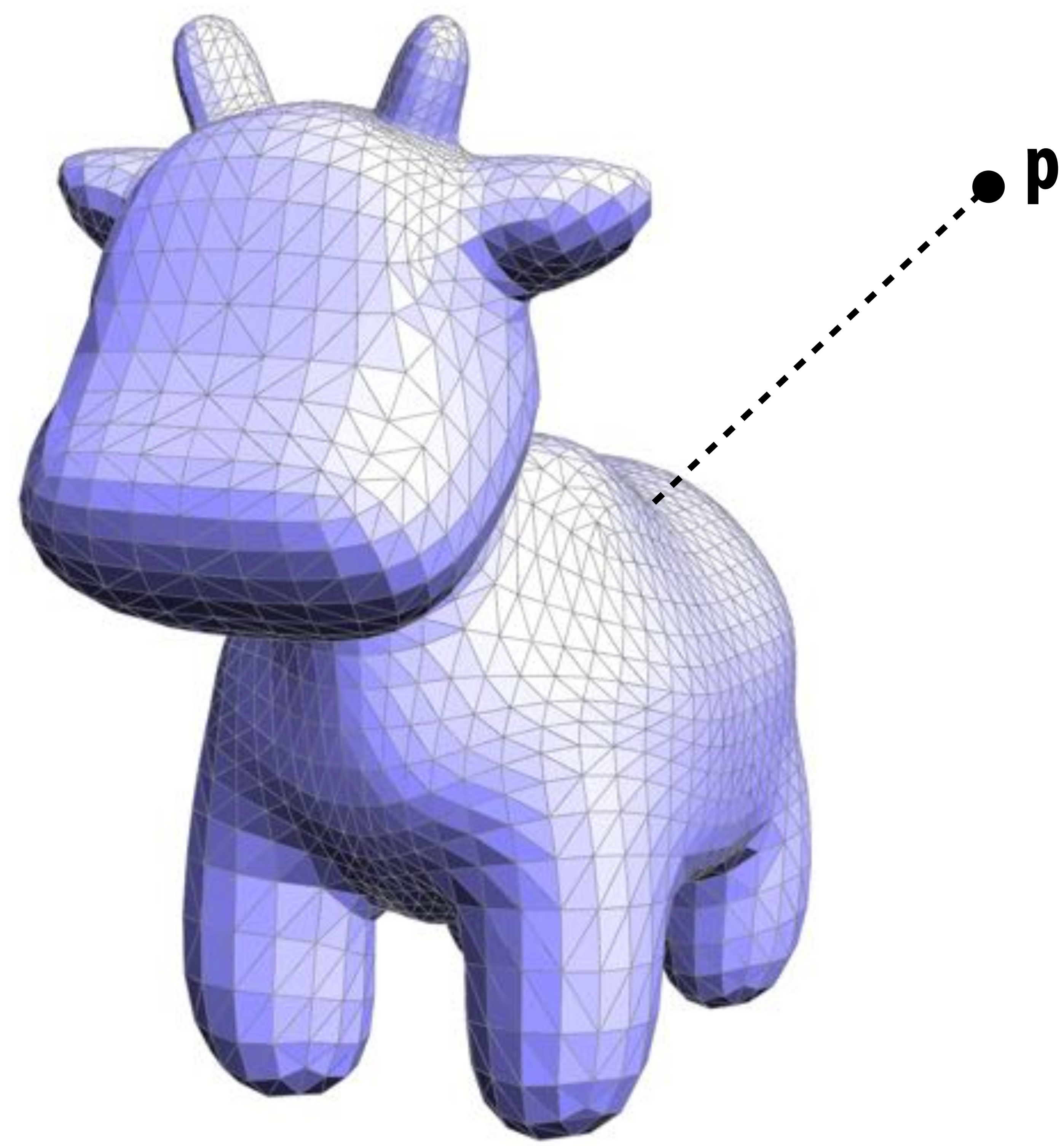
Closest point on triangle in 3D

- Not so different from 2D case
- Algorithm?
  - project onto plane of triangle
  - use half-space tests to classify point (vs. half plane)
  - if inside the triangle, we're done!
  - otherwise, find closest point on associated vertex or edge
- By the way, how do we find closest point on plane?
- Same expression as closest point on a line!
- E.g., $p + (c - N^T p) N$

Closest point on triangle mesh in 3D?

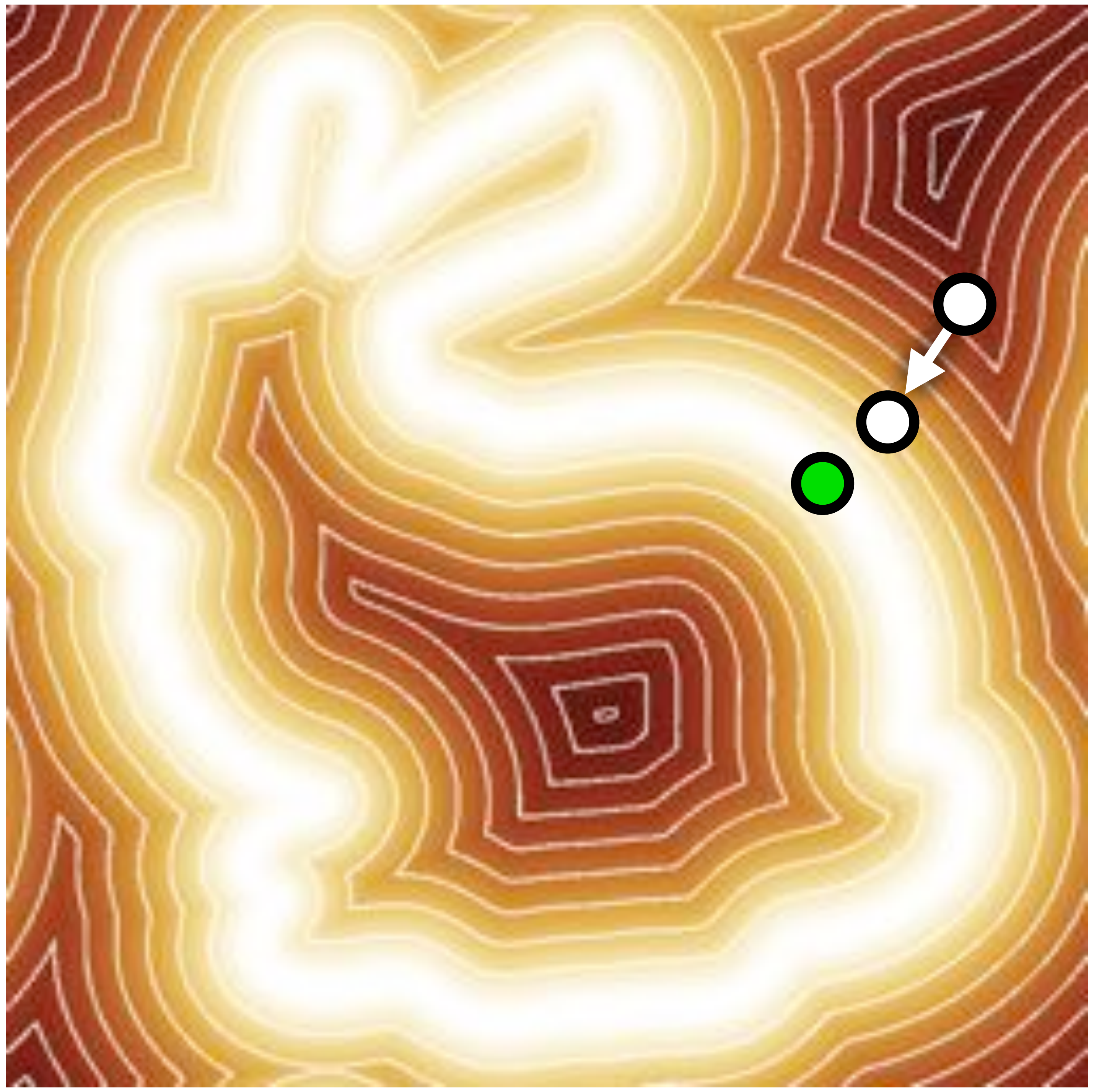
- Conceptually easy:
  - loop over all triangles
  - compute closest point to current triangle
  - keep globally closest point
- Q: What's the cost?
- What if we have billions of faces?
- NEXT TIME: Better data structures!





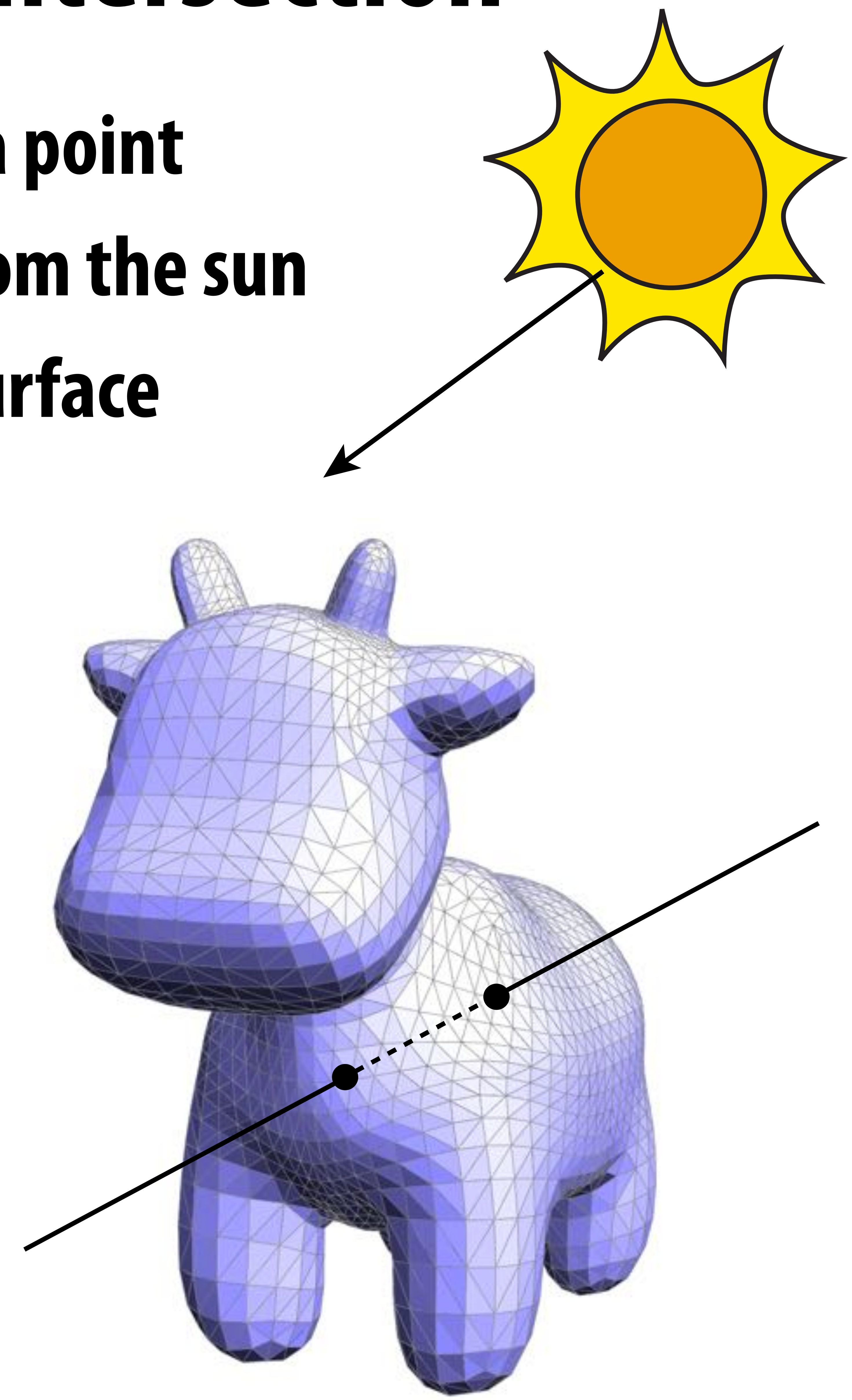
Closest point to implicit surface?

- If we change our representation of geometry, algorithms can change completely
- E.g., how might we compute the closest point on an implicit surface described via its distance function?
- One idea:
  - start at the query point
  - compute gradient of distance (using, e.g., finite differences)
  - take a little step (decrease distance)
  - repeat until we're at the surface (zero distance)
- Better yet: just store closest point for each grid cell! (speed/memory trade off)



Different query: ray-mesh intersection

- A "ray" is an oriented line starting at a point
- Think about a ray of light traveling from the sun
- Want to know where a ray pierces a surface
- Why?
  - GEOMETRY: inside-outside test
  - RENDERING: visibility, ray tracing
  - ANIMATION: collision detection
- Might pierce surface in many places!



Ray equation

- **Can express ray as**

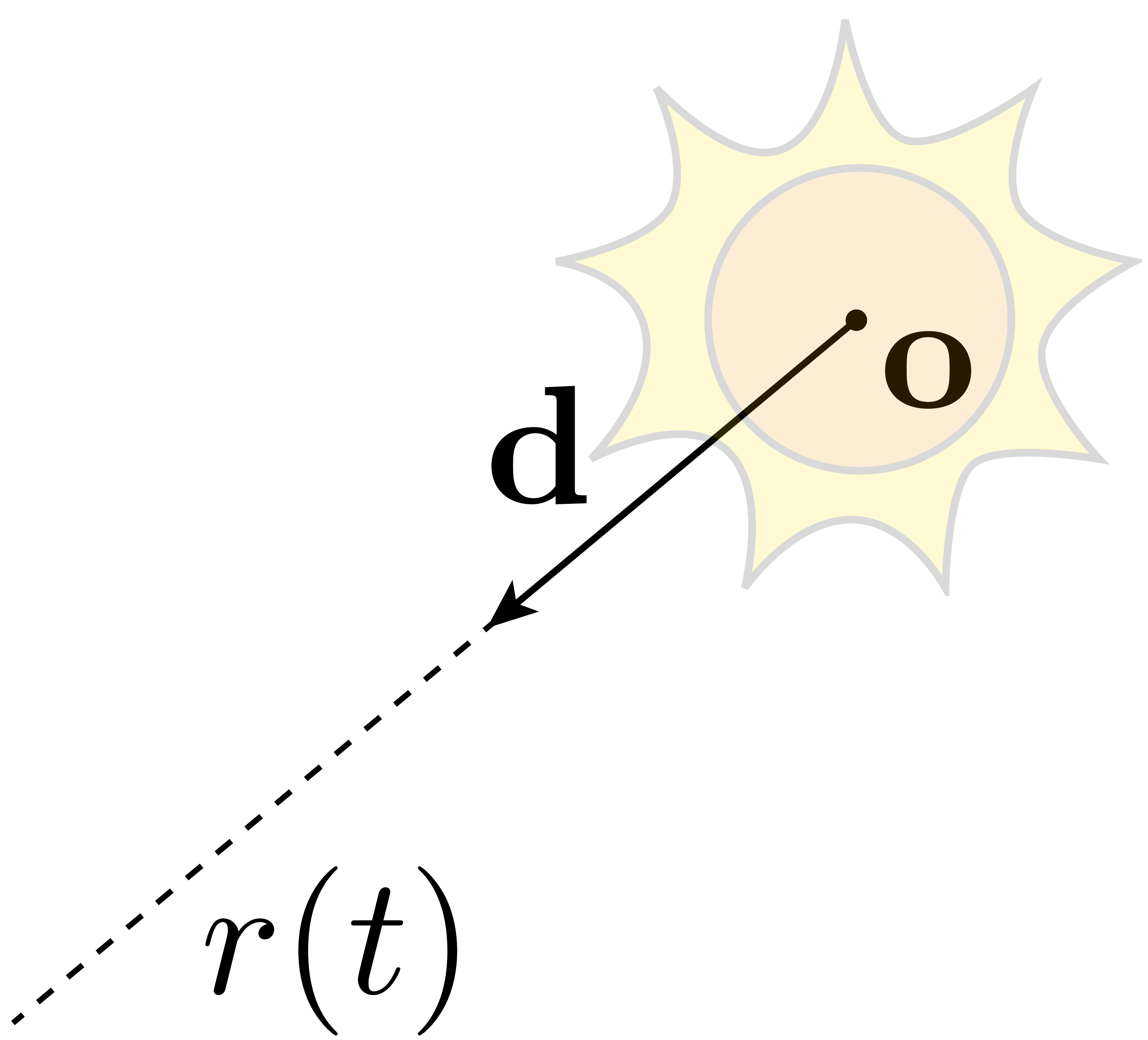
origin

unit direction

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

point along ray

"time"



Intersecting a ray with an implicit surface

- **Recall implicit surfaces: all points x such that f(x) = 0**
- **Q: How do we find points where a ray pierces this surface?**
- **Well, we know all points along the ray: r(t) = o + td**
- **Idea: replace "x" with "r" in 1st equation, and solve for t**
- **Example: unit sphere**

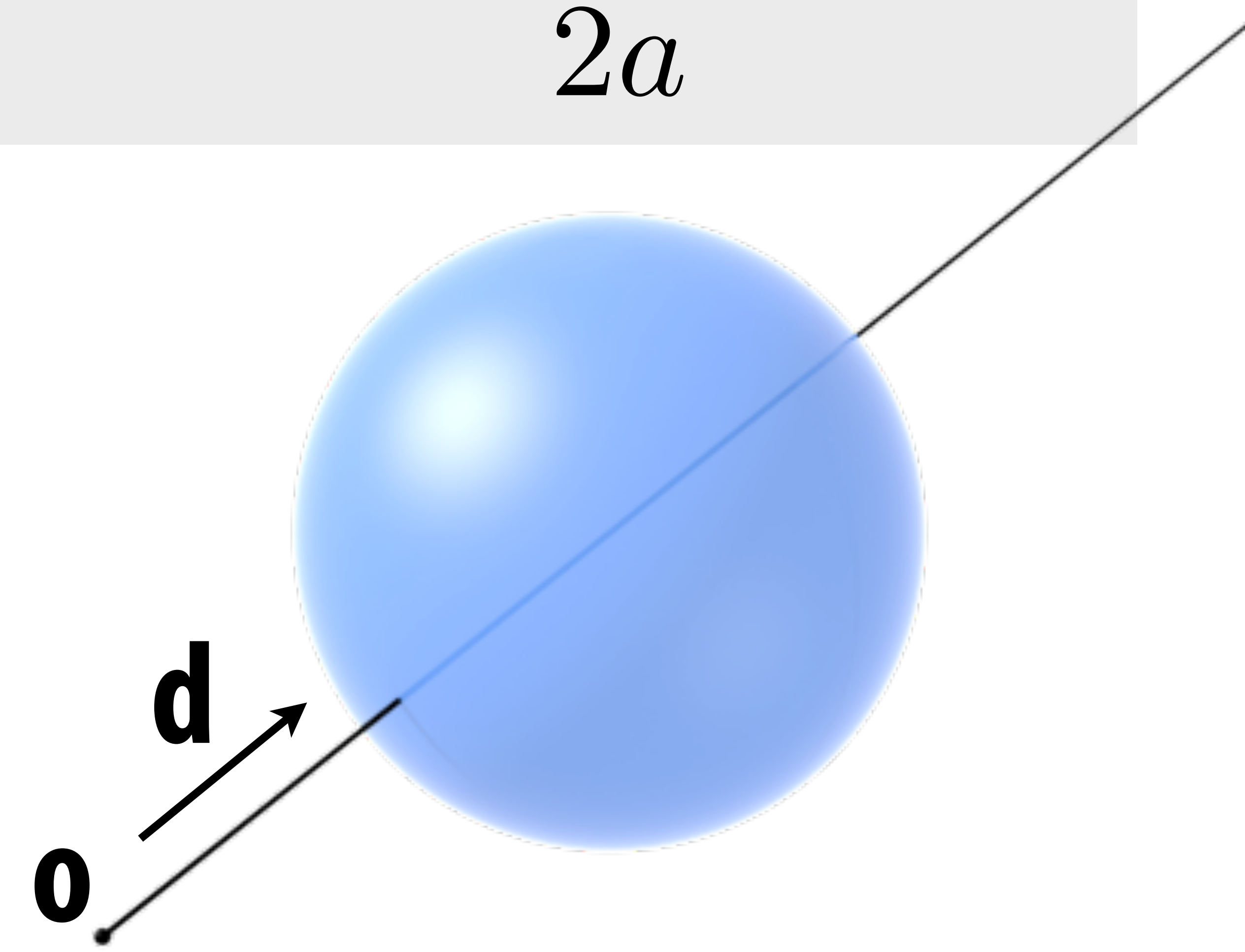
quadratic formula:

$$t = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

$$f(\mathbf{x}) = |\mathbf{x}|^2 - 1$$

$$\Rightarrow f(\mathbf{r}(t)) = |\mathbf{o} + t\mathbf{d}|^2 - 1$$

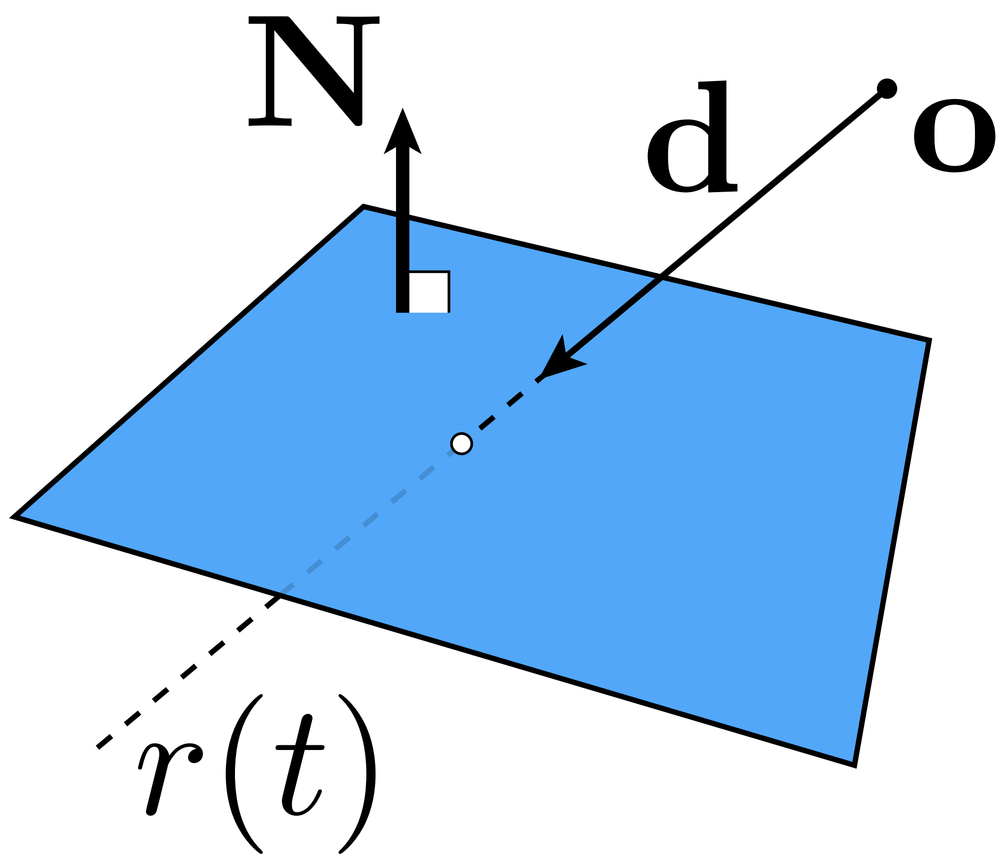
$$\underbrace{|\mathbf{d}|^2}_{a} t^2 + \underbrace{2(\mathbf{o}\cdot\mathbf{d})}_{b} t + \underbrace{|\mathbf{o}|^2 - 1}_{c} = 0$$



$$t = \boxed{-\mathbf{o}\cdot\mathbf{d} \pm \sqrt{(\mathbf{o}\cdot\mathbf{d})^2 - |\mathbf{o}|^2 + 1}}$$

Why two solutions?

Ray-plane intersection



- Suppose we have a plane $N^T x = c$
  - N - unit normal
  - c - offset
- How do we find intersection with ray $r(t) = o + td$?
- Key idea: again, replace the point x with the ray equation t:

$$\mathbf{N}^{\mathsf{T}}\mathbf{r}(t) = c$$

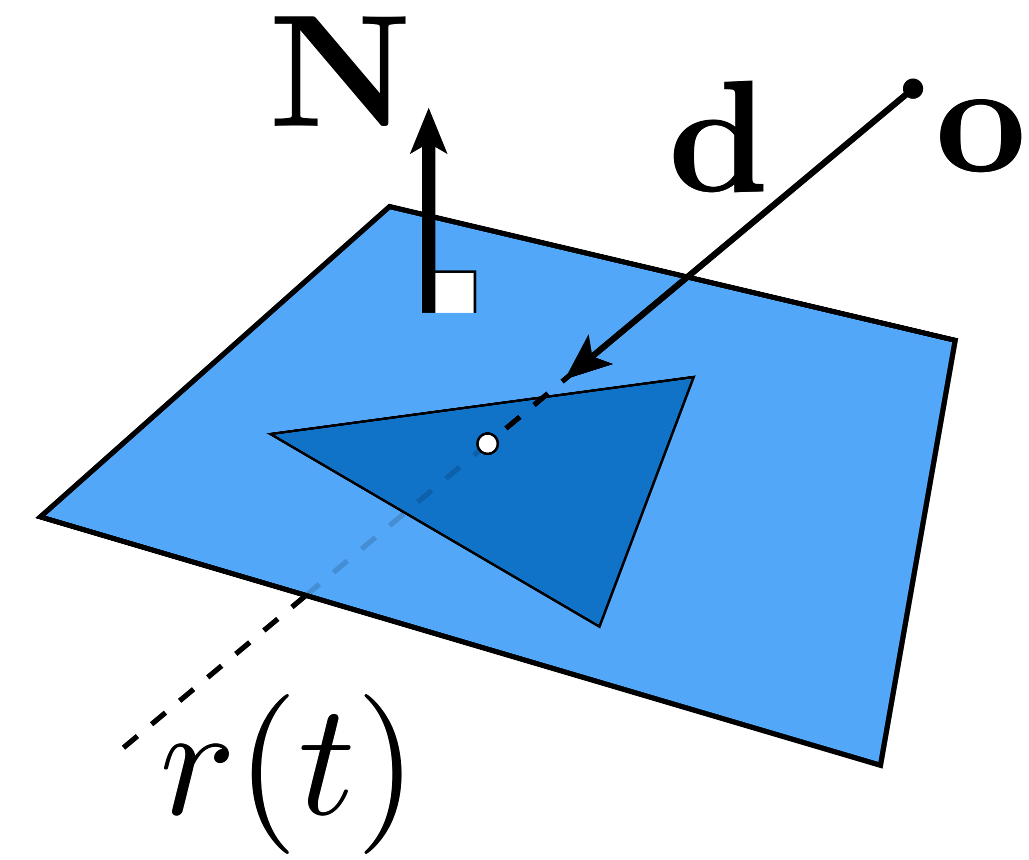
- Now solve for t:

$$\mathbf{N}^{\mathsf{T}}(\mathbf{o} + t\mathbf{d}) = c \qquad \Rightarrow t = \frac{c - \mathbf{N}^{\mathsf{T}}\mathbf{o}}{\mathbf{N}^{\mathsf{T}}\mathbf{d}}$$

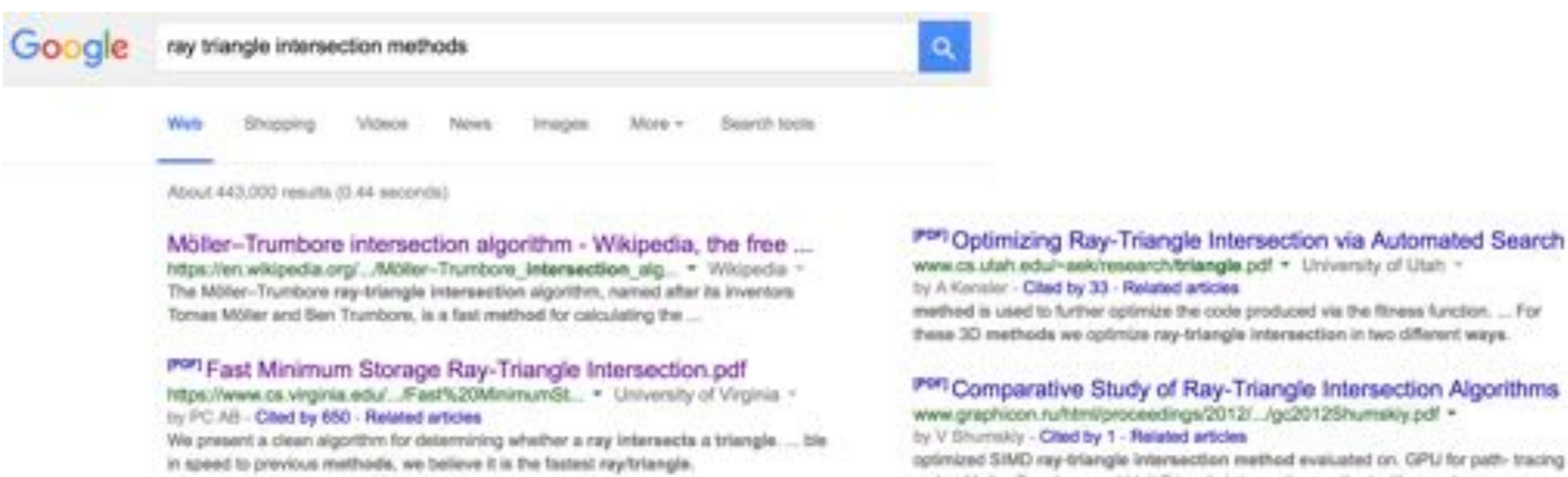
- And plug t back into ray equation:

$$r(t) = \mathbf{o} + \frac{c - \mathbf{N}^{\mathsf{T}}\mathbf{o}}{\mathbf{N}^{\mathsf{T}}\mathbf{d}}\mathbf{d}$$

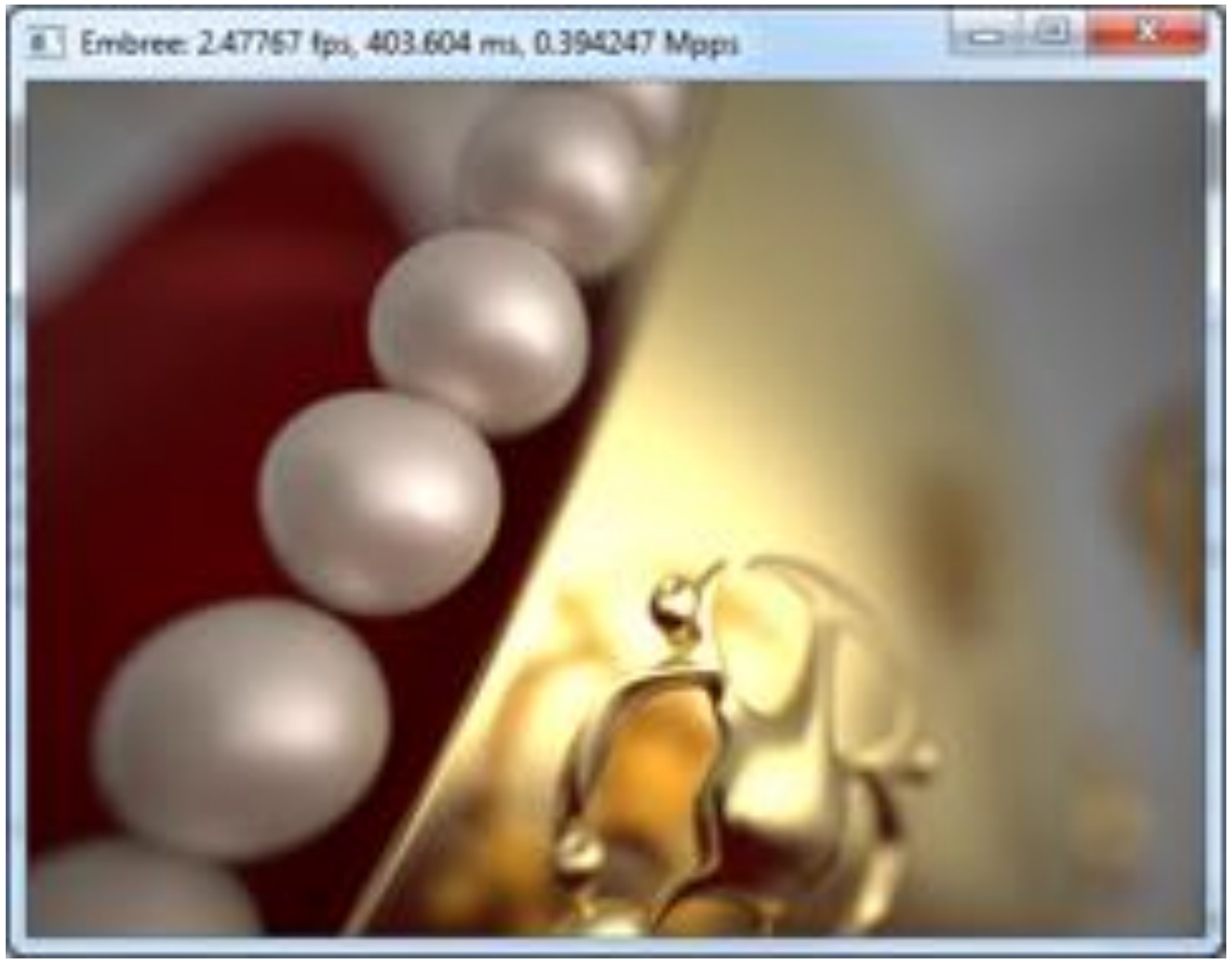
Ray-triangle intersection



- Triangle is in a plane...
- Not much more to say!
  - Compute ray-plane intersection
  - Q: What do we do now?
  - A: Why not compute barycentric coordinates of hit point?
  - If barycentric coordinates are all positive, point in triangle
- Actually, a lot more to say... if you care about performance!



Why care about performance?



Intel Embree

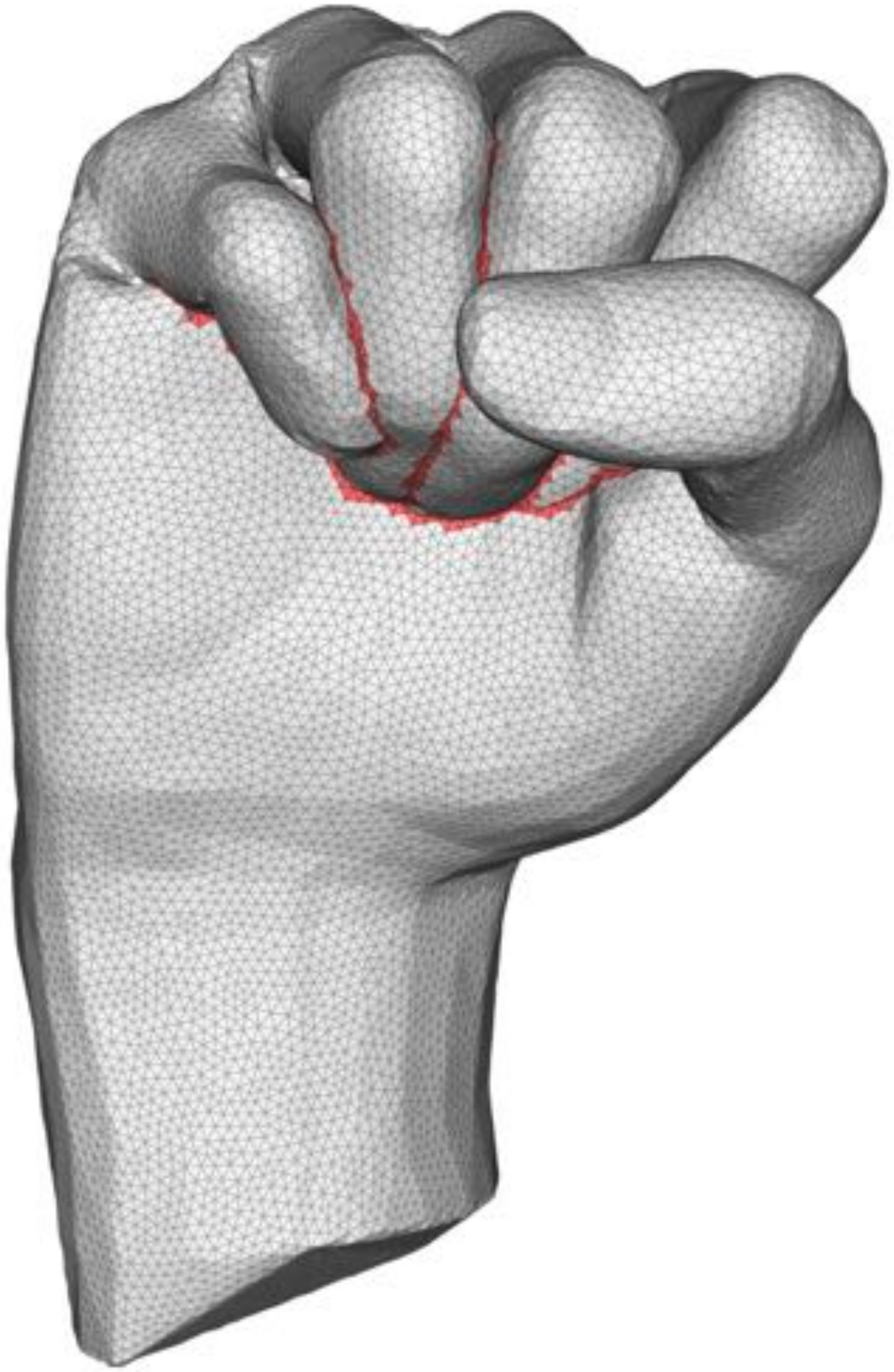
NVIDIA OptiX

Why care about performance?

"Brigade 3" real time path tracing demo

One more query: mesh-mesh intersection

- **GEOMETRY: How do we know if a mesh intersects itself?**
- **ANIMATION: How do we know if a collision occurred?**



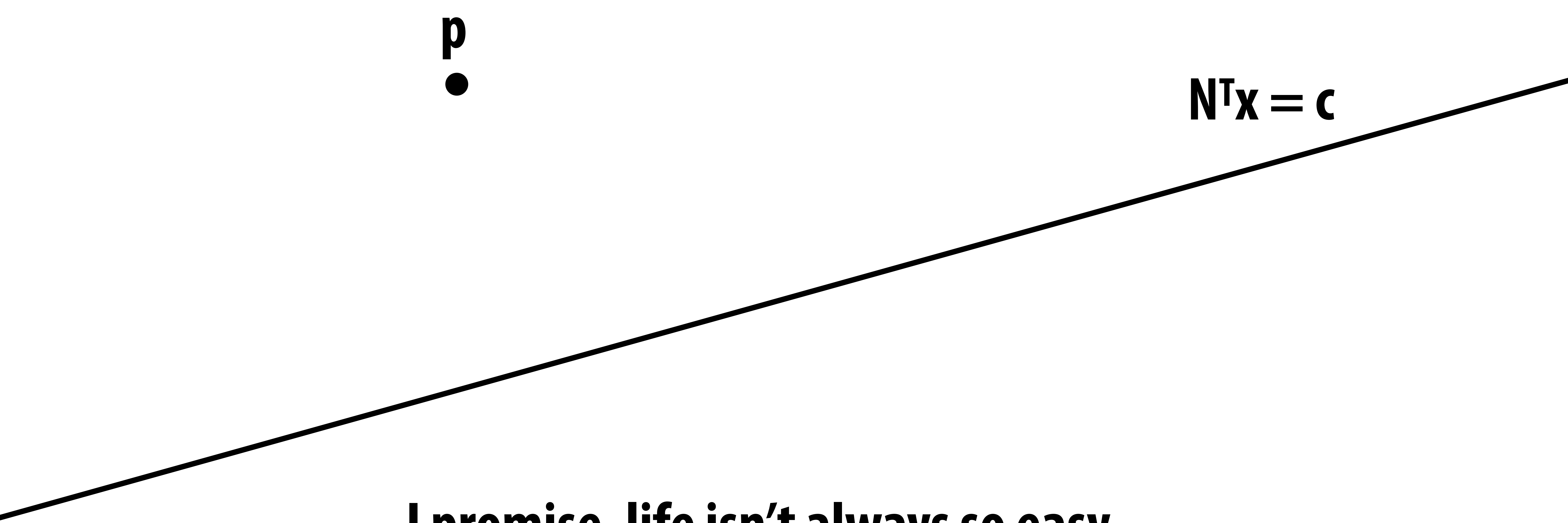
Warm up: point-point intersection

- Q: How do we know if p intersects a?
- A: ...check if they're the same point!

Sadly, life is not always so easy.

Slightly harder: point-line intersection

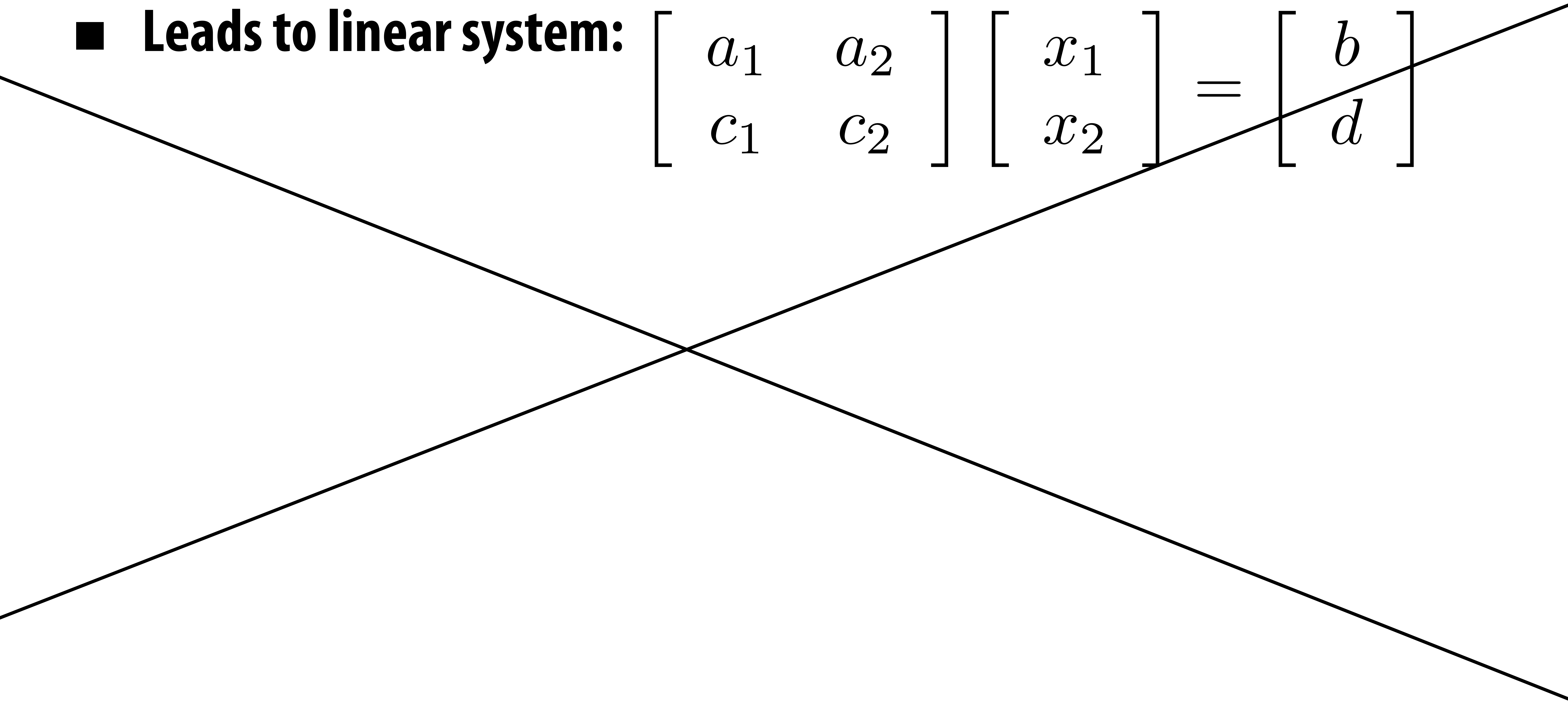
- Q: How do we know if a point intersects a given line?
- A: ...plug it into the line equation!



I promise, life isn't always so easy.

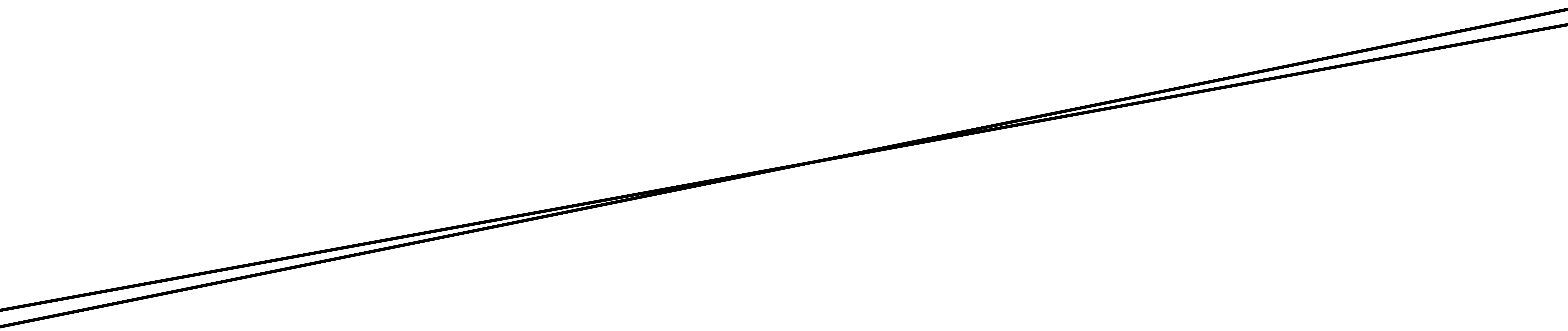
Finally interesting: line-line intersection

- **Two lines: ax=b and cx=d**
- **Q: How do we find the intersection?**
- **A: See if there is a simultaneous solution**
- **Leads to linear system:** $\begin{bmatrix} a_1 & a_2 \\ c_1 & c_2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} b \\ d \end{bmatrix}$



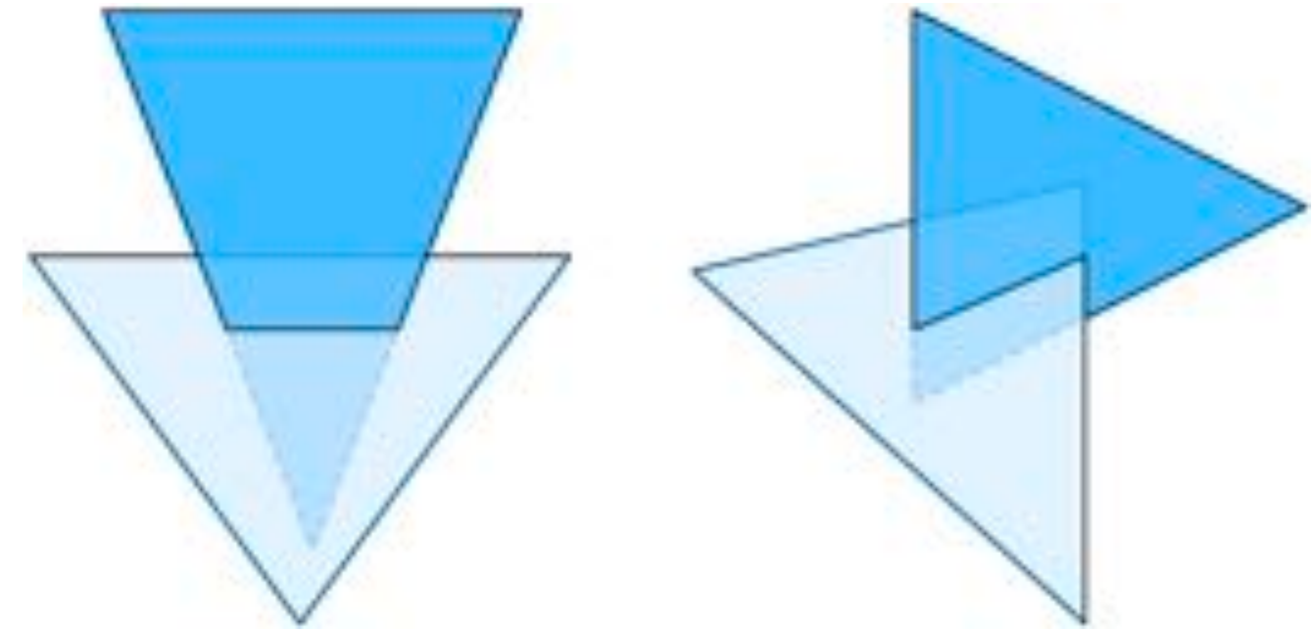
Degenerate line-line intersection?

- **What if lines are almost parallel?**
- **Small change in normal can lead to big change in intersection!**
- **Instability very common, very important with geometric predicates. Demands special care (e.g., analysis of matrix).**

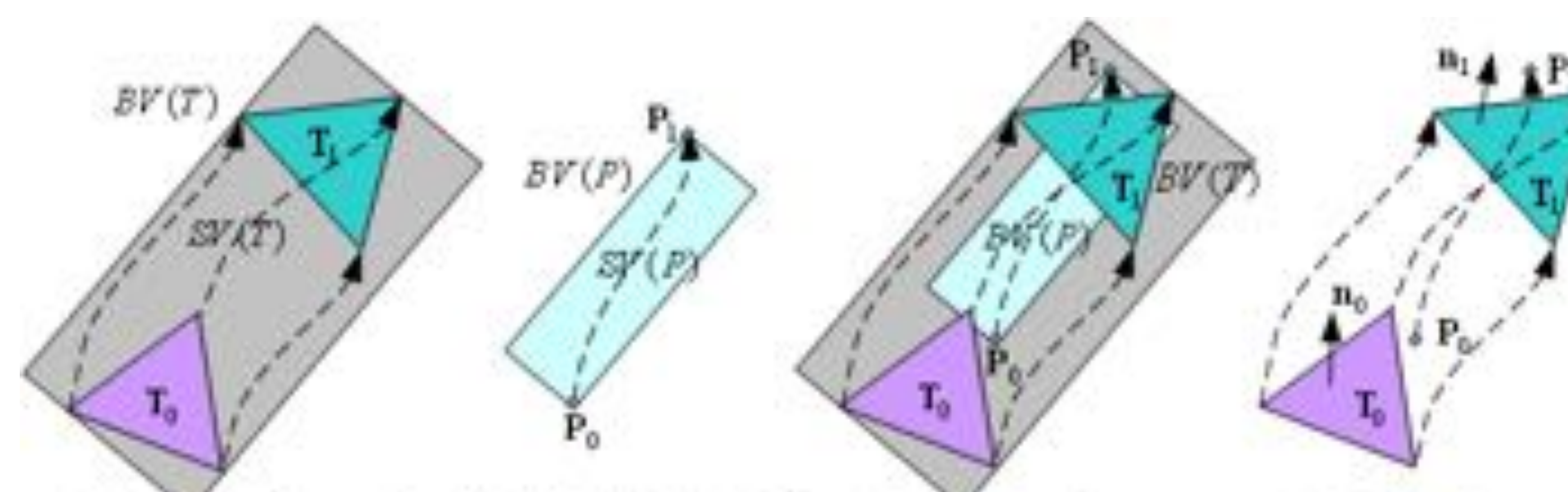


See for example Shewchuk, "*Adaptive Precision Floating-Point Arithmetic and Fast Robust Geometric Predicates*"

Triangle-Triangle Intersection?



- Lots of ways to do it
- Basic idea:
  - Q: Any ideas?
  - One way: reduce to edge-triangle intersection
  - Check if each line passes through plane
  - Then do interval test
- What if triangle is moving?
  - Important case for animation
  - Can think of triangles as prisms in time
  - Turns dynamic problem (nD + time) into purely geometric problem in (n+1)-dimensions



(a) Bounding volume of a deforming triangle (b) Bounding volume of a deforming vertex (c) Bounding volume test (d) Coplanarity test

Up Next: Spatial Acceleration Data Strucutres

- Testing every element is slow!
- E.g., linearly scanning through a list vs. binary search
- Can apply this same kind of thinking to geometric queries