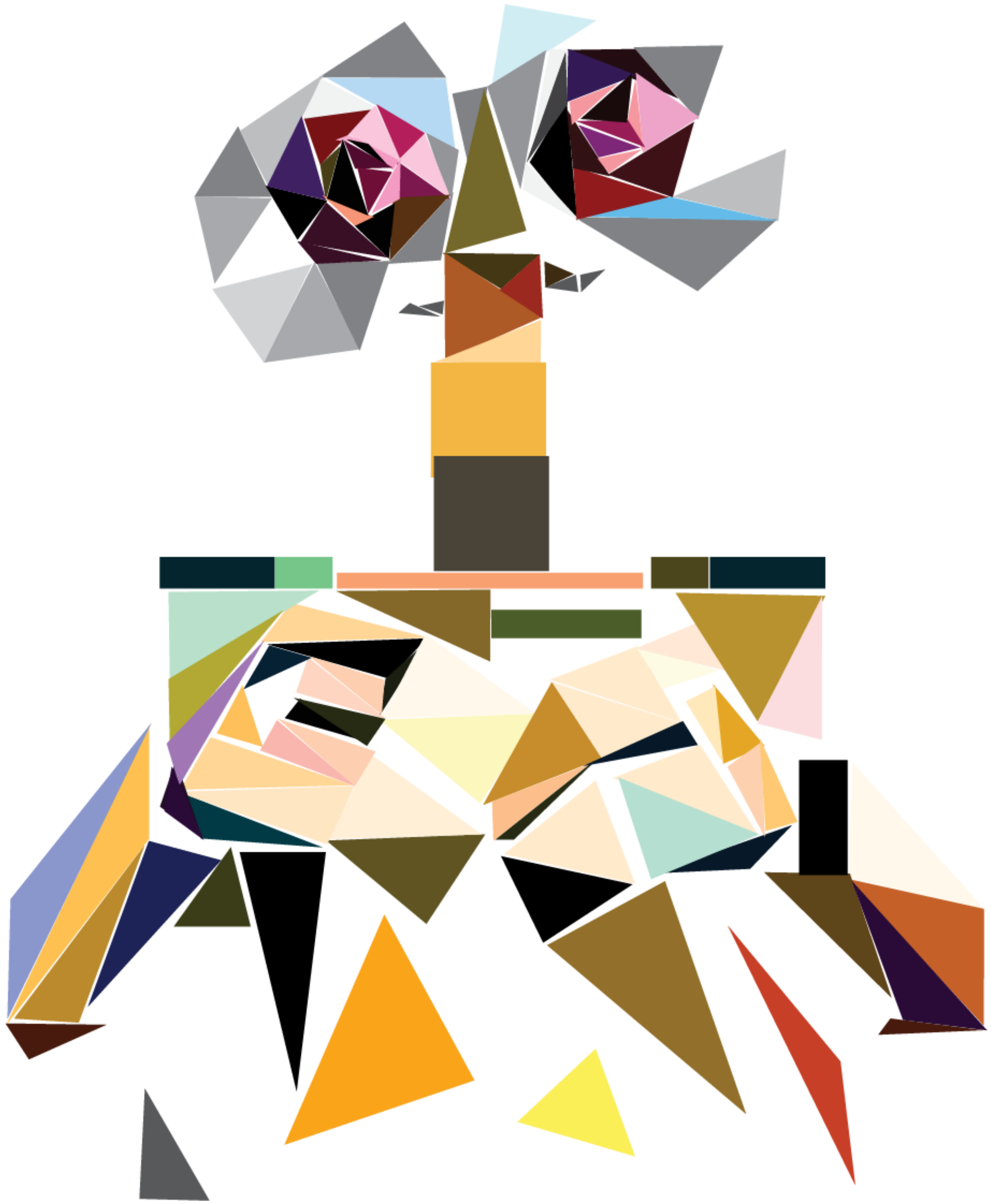
Lecture 8:

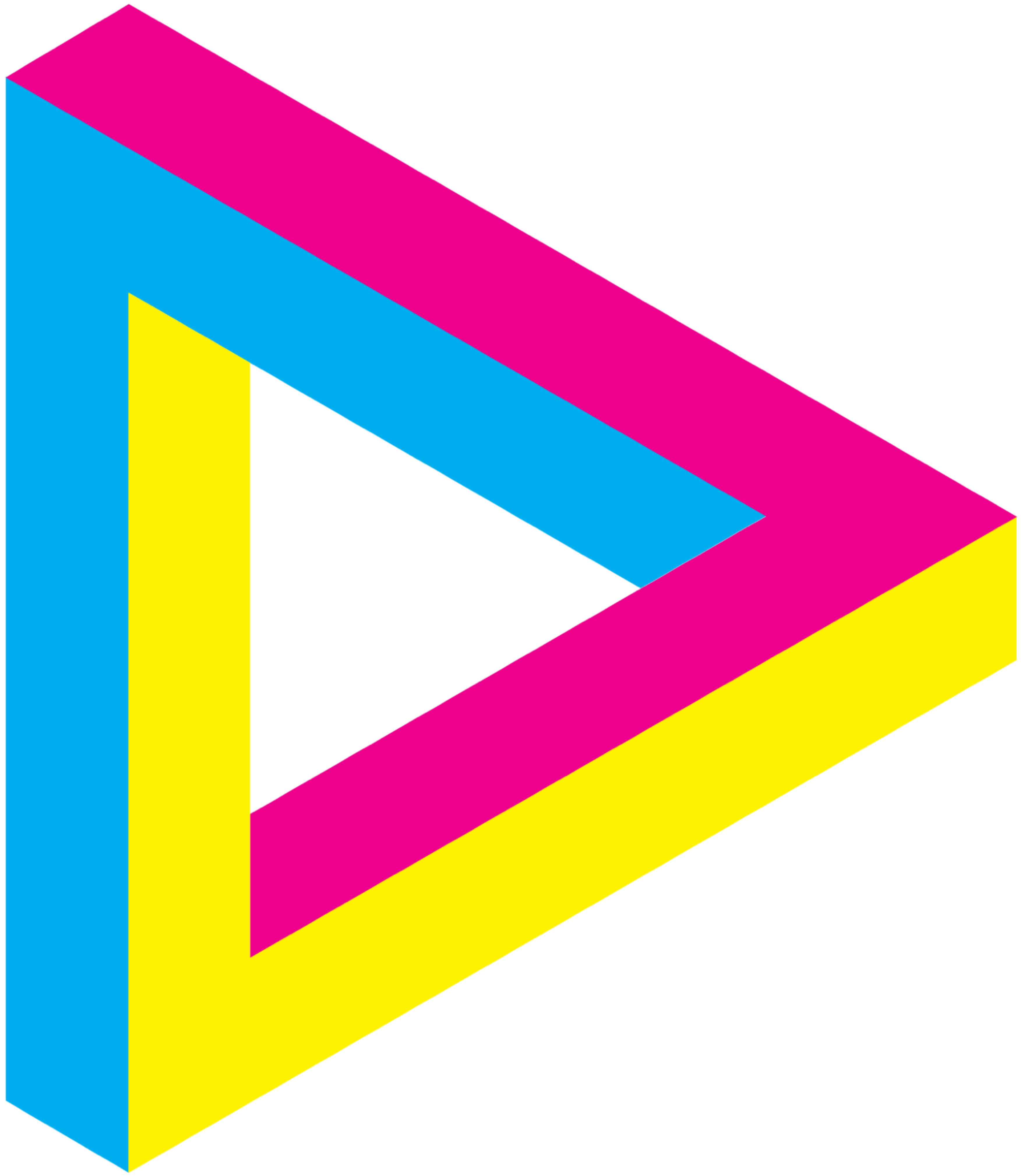
Geometric Queries

Interactive Computer Graphics
Stanford CS248, Spring 2018

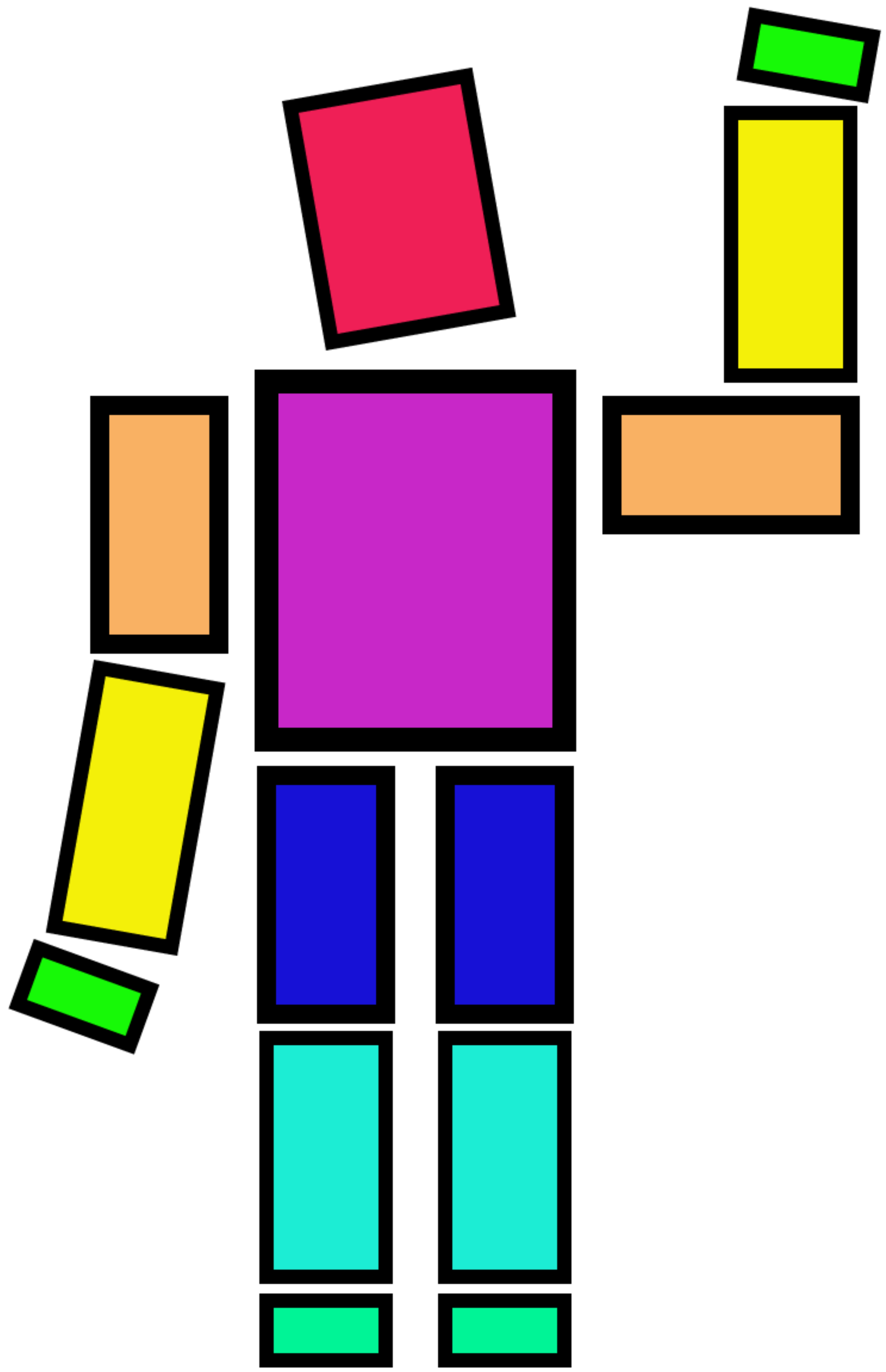
Great SVGs from assignment 1



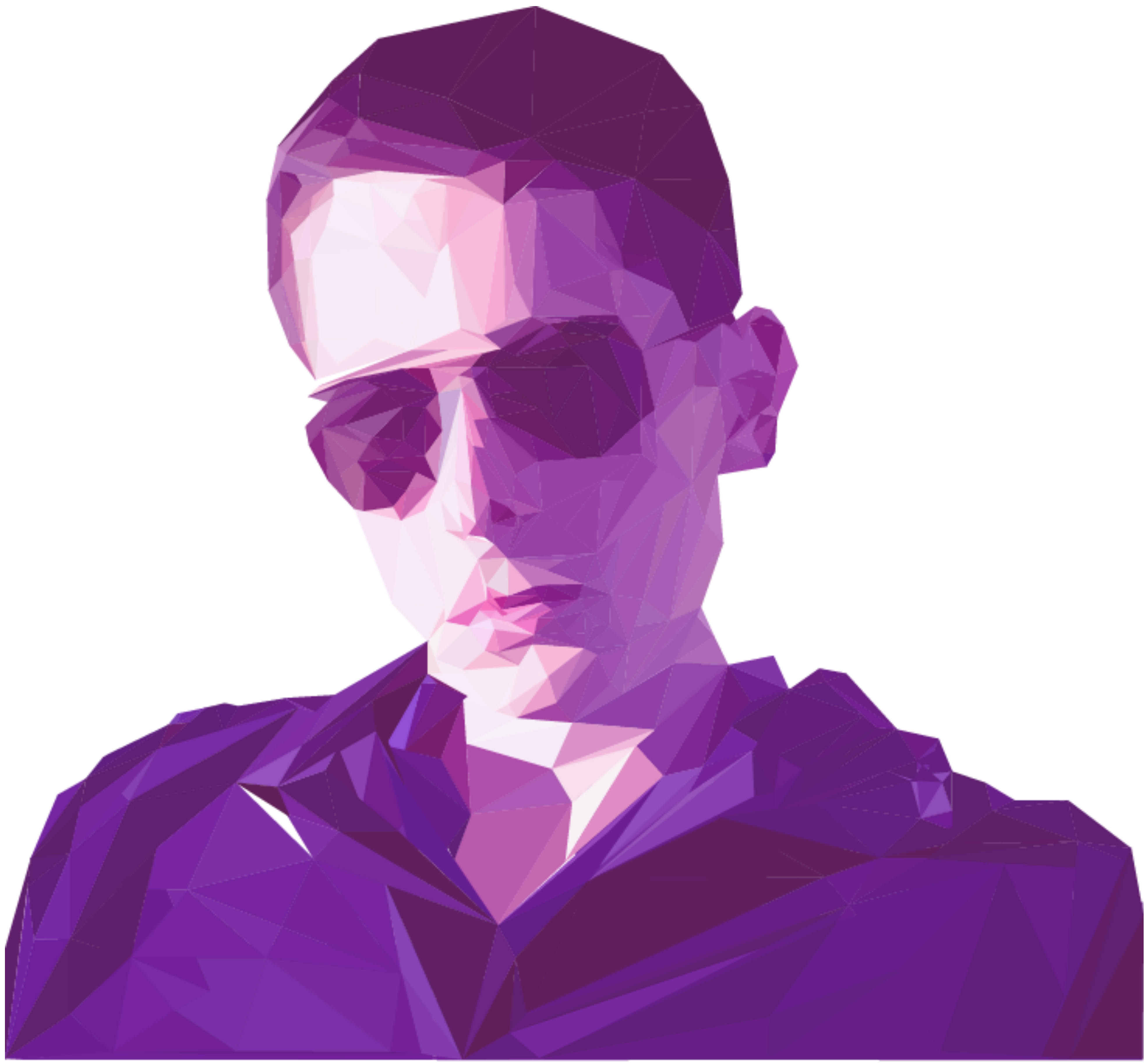
Great SVGs from assignment 1



Great SVGs from assignment 1

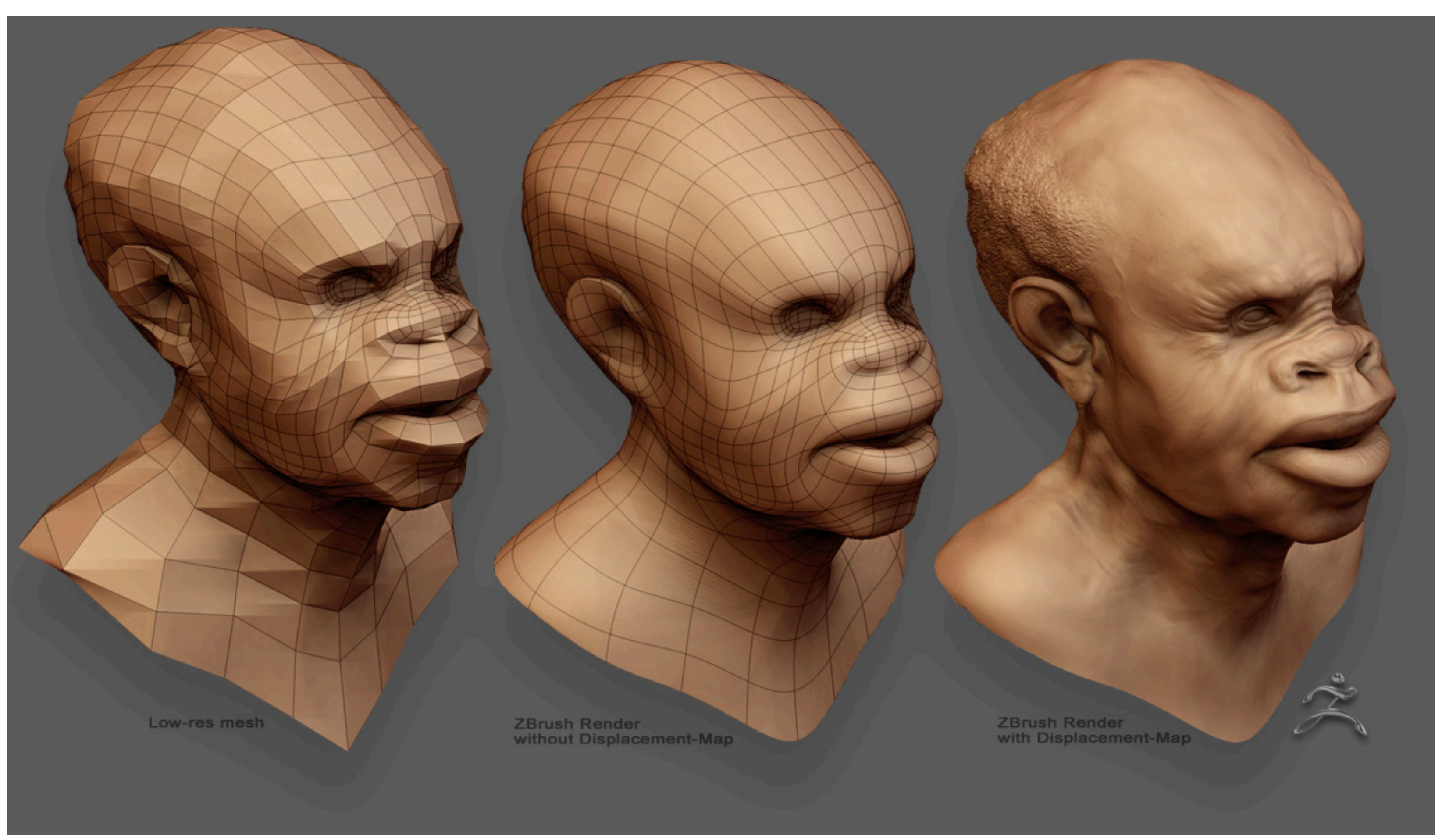


Great SVGs from assignment 1



Geometric queries — motivation

Intersecting rays and triangles
(ray tracing)

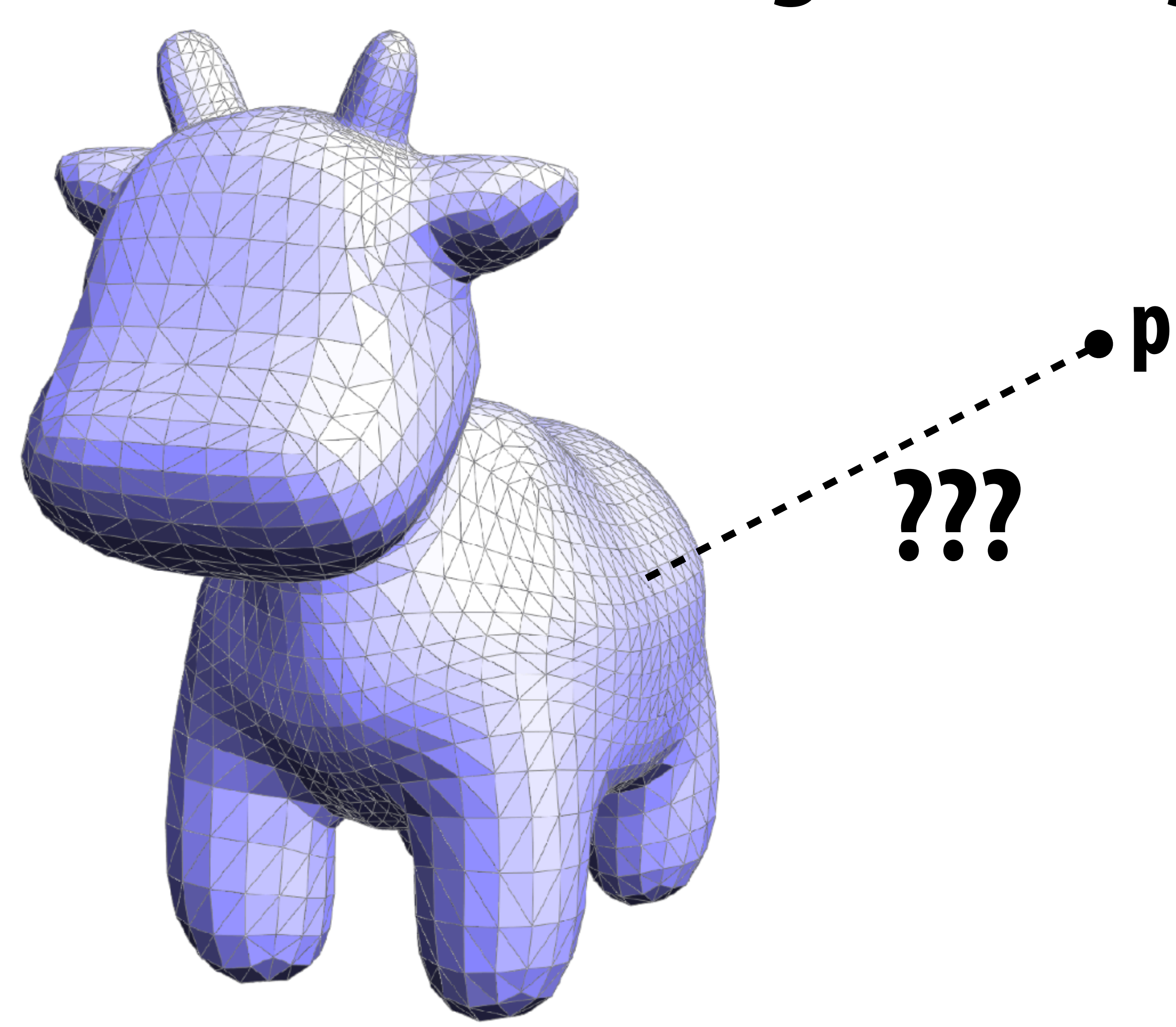


Closest point on surface queries

Intersecting triangles (collisions)

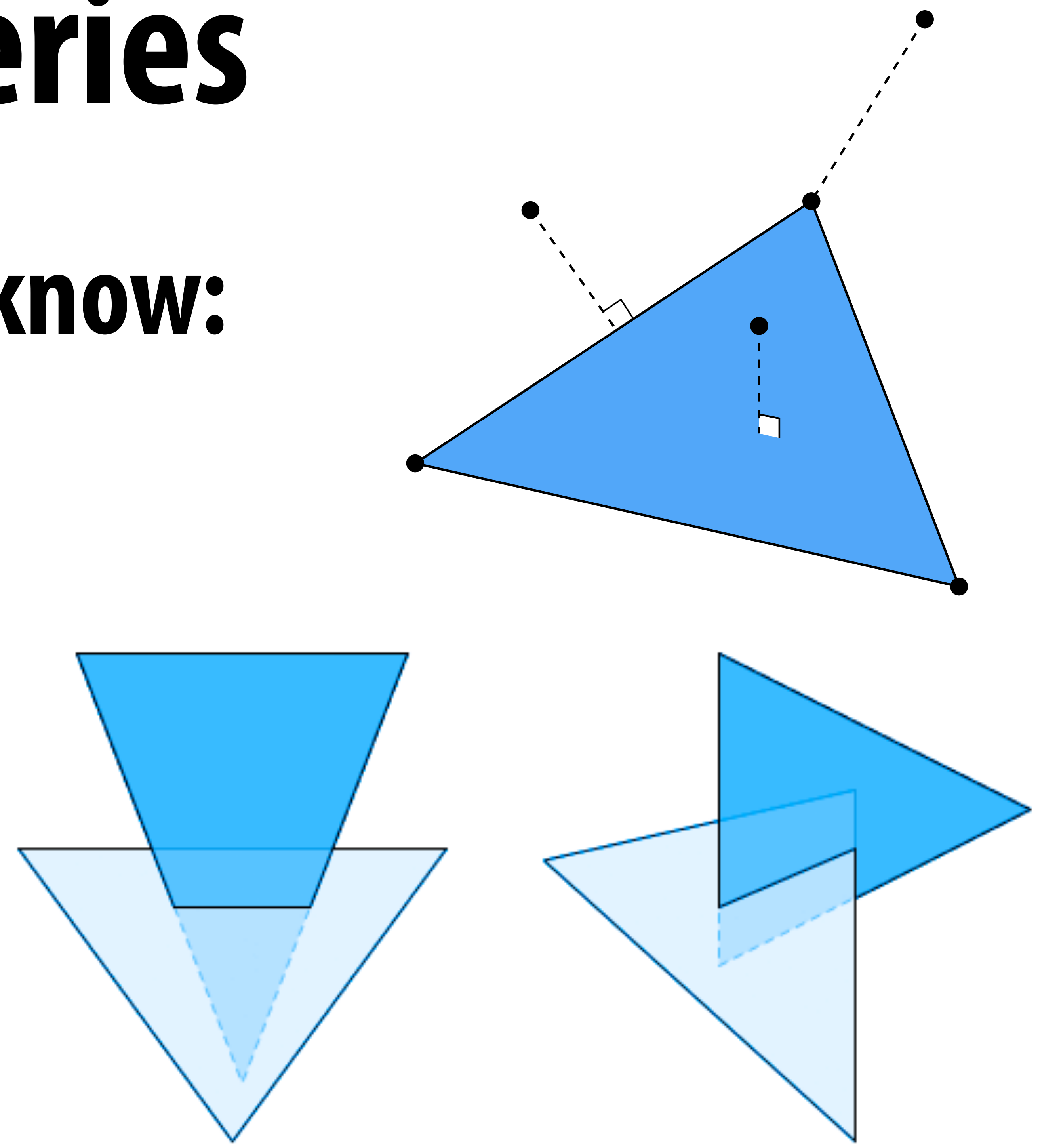
Example: closest point queries

- Q: Given a point, in space (e.g., a new sample point), how do we find the closest point on a given surface?
- Q: Does implicit/explicit representation make this easier?
- Q: Does our half-edge data structure help?
- Q: What's the cost of the naïve algorithm?
- Q: How do we find the distance to a single triangle anyway?



Many types of geometric queries

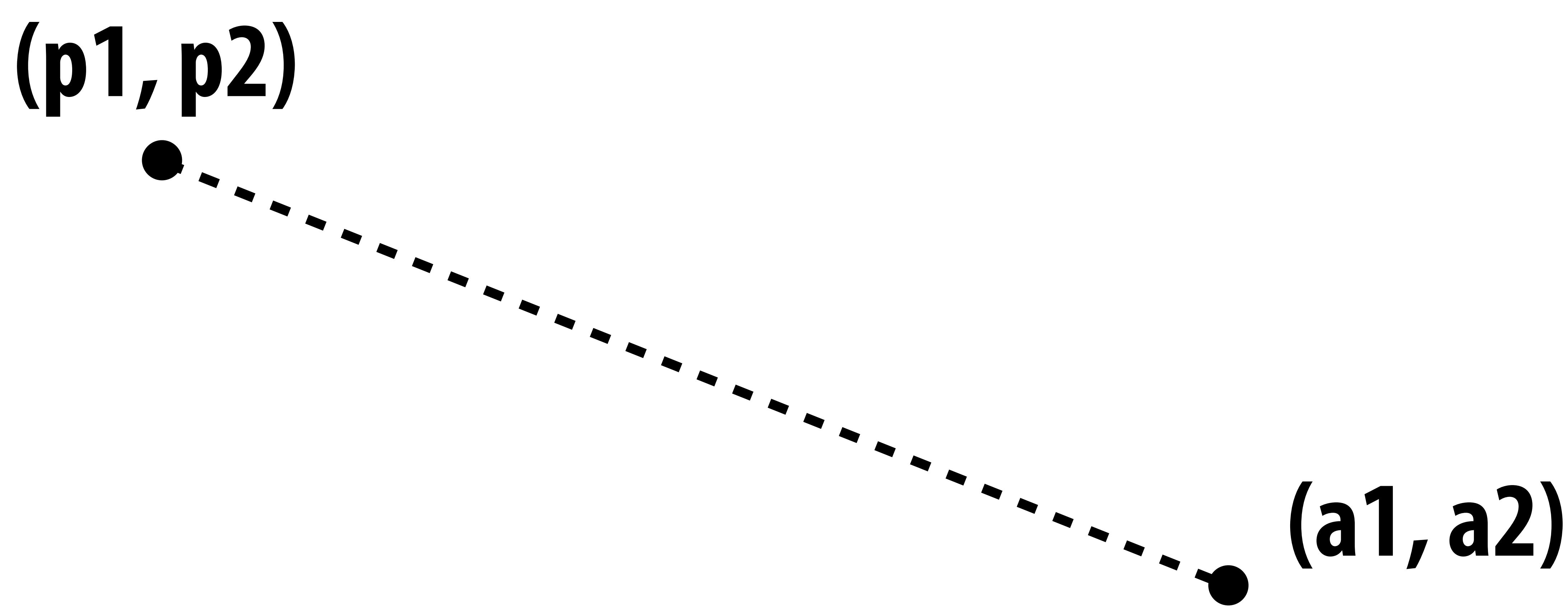
- Plenty of other things we might like to know:
  - Do two triangles intersect?
  - Are we inside or outside an object?
  - Does one object contain another?
  - ...



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

Warm up: closest point on point

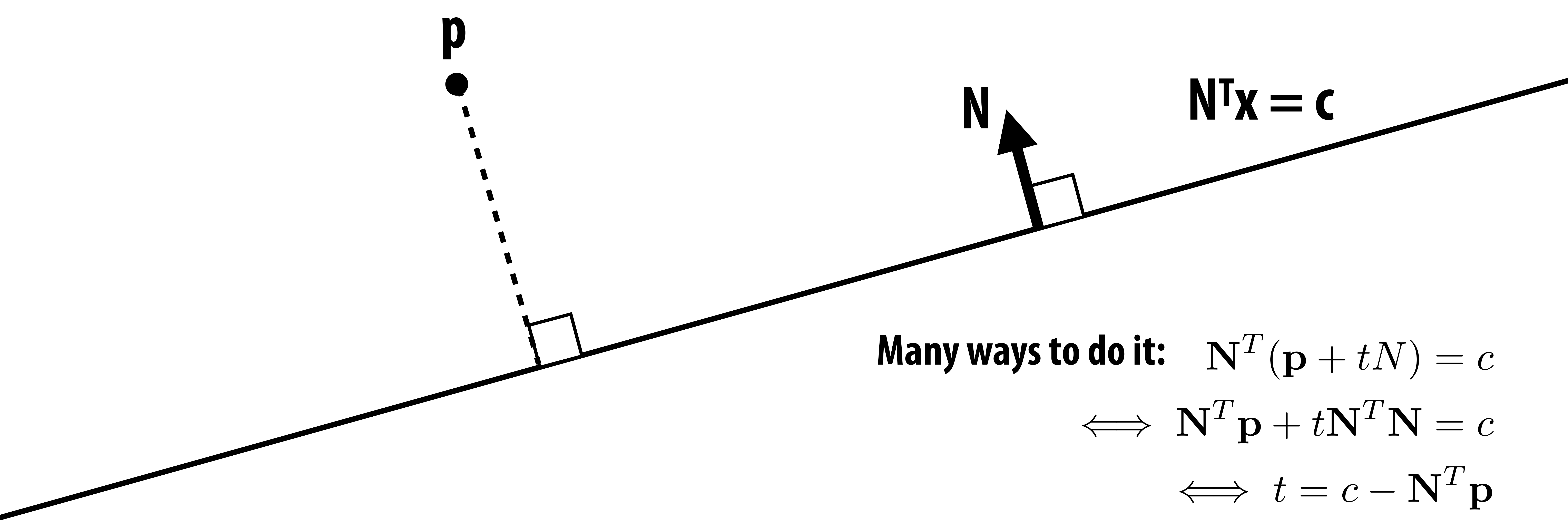
- *G*iven a query point (p1,p2), how do we find the closest point on the point (a1,a2)?



Bonus question: what's the distance?

Slightly harder: closest point on line

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

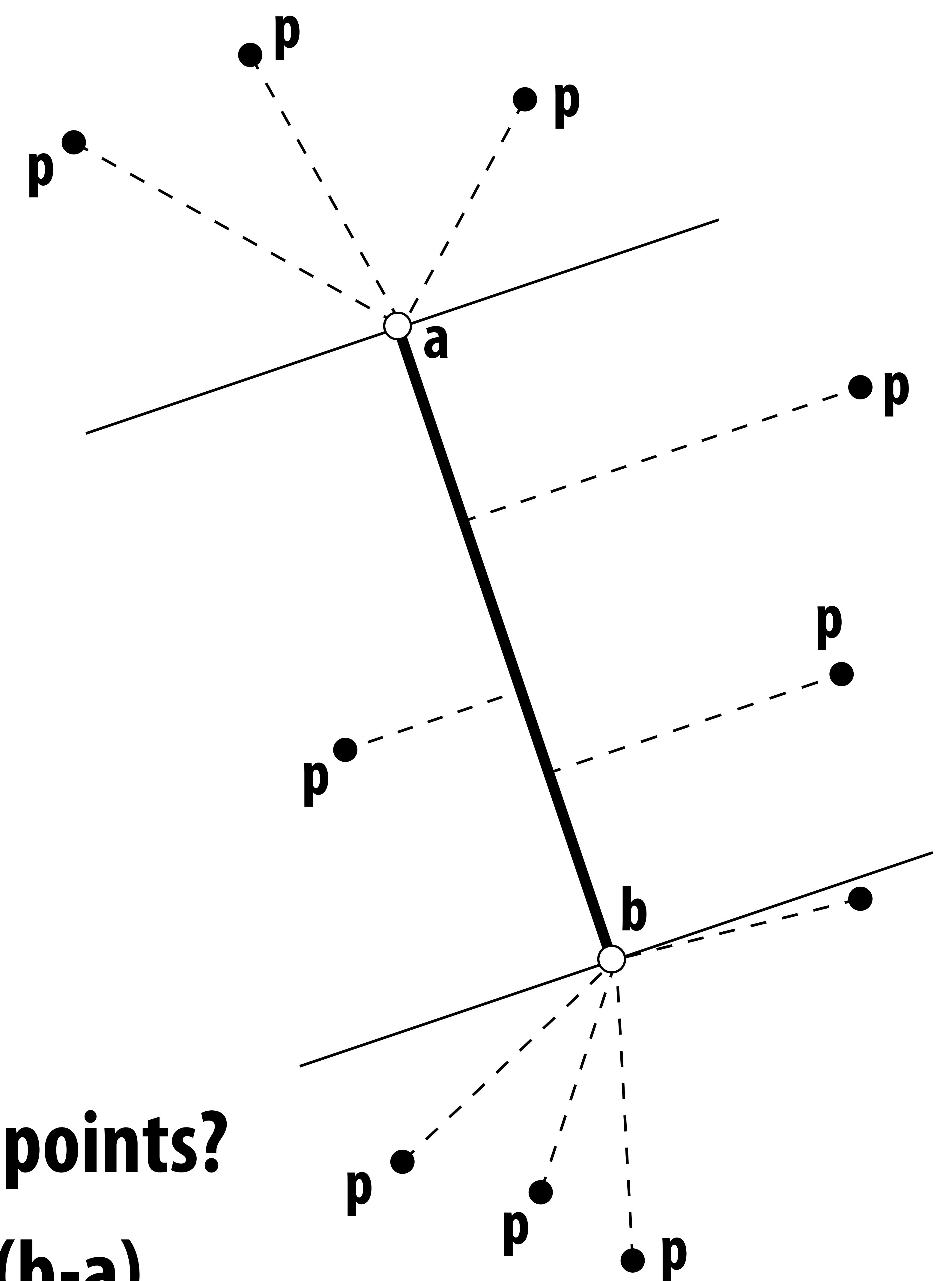


Many ways to do it:

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

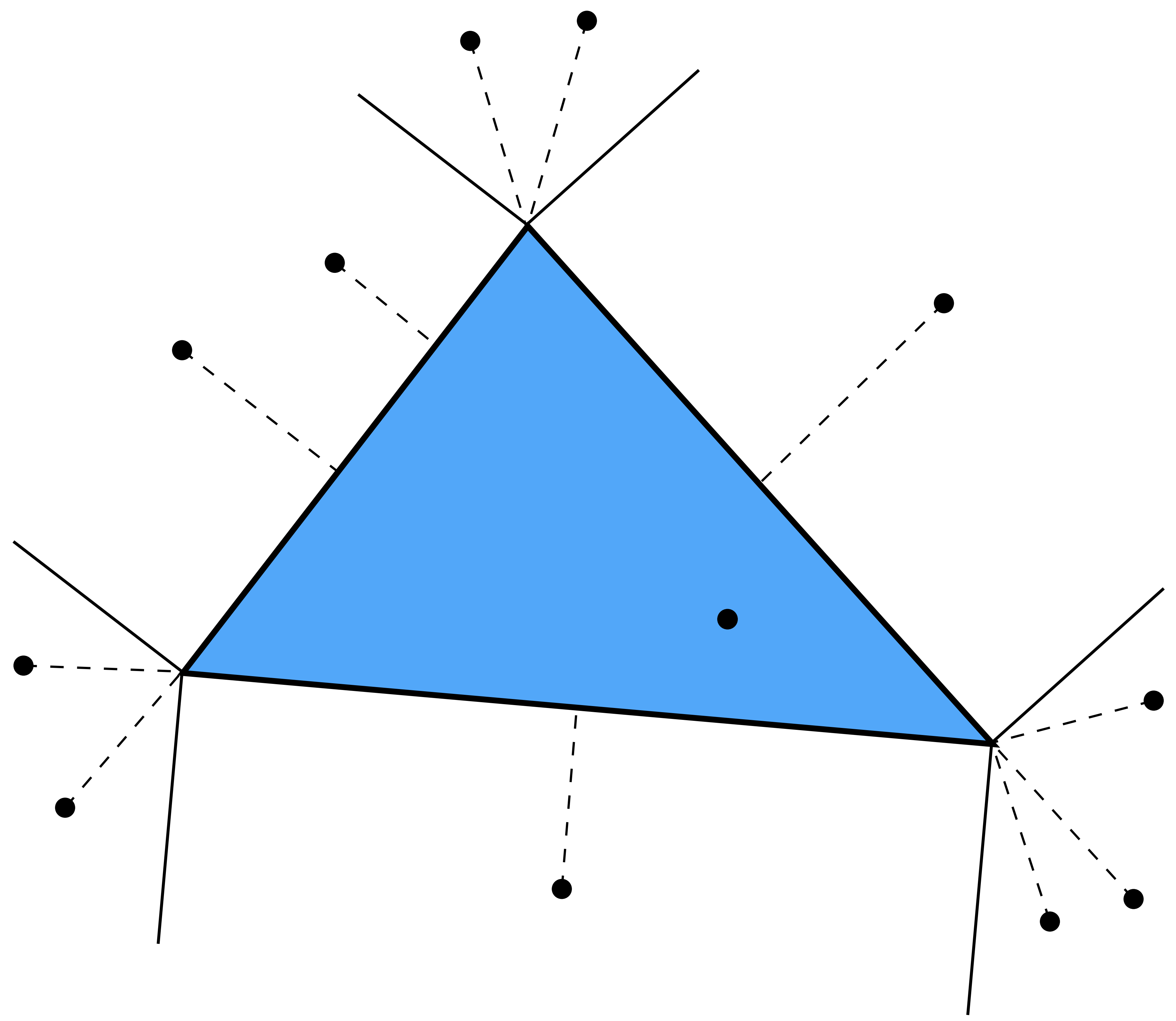
Harder: closest point on line segment

- Two cases: endpoint or interior
- Already have basic components:
  - point-to-point
  - point-to-line
- Algorithm?
  - find closest point on line
  - check if it is 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 line segments:



Q: What about a point inside the triangle?

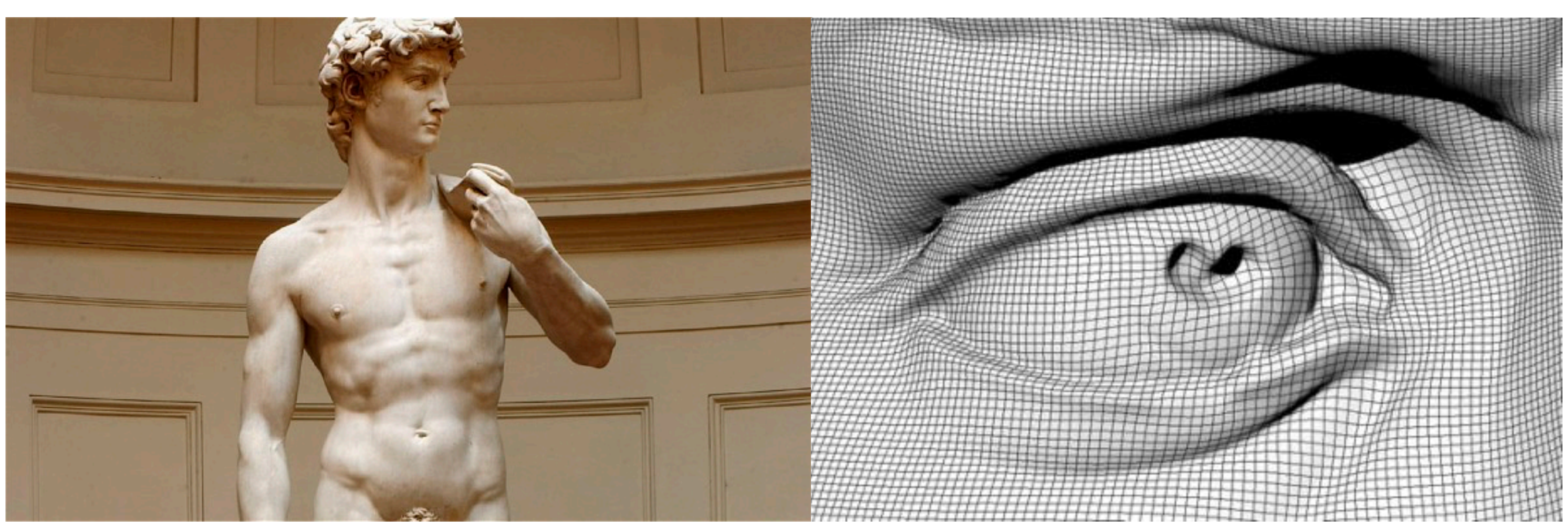
Closest point on triangle in 3D

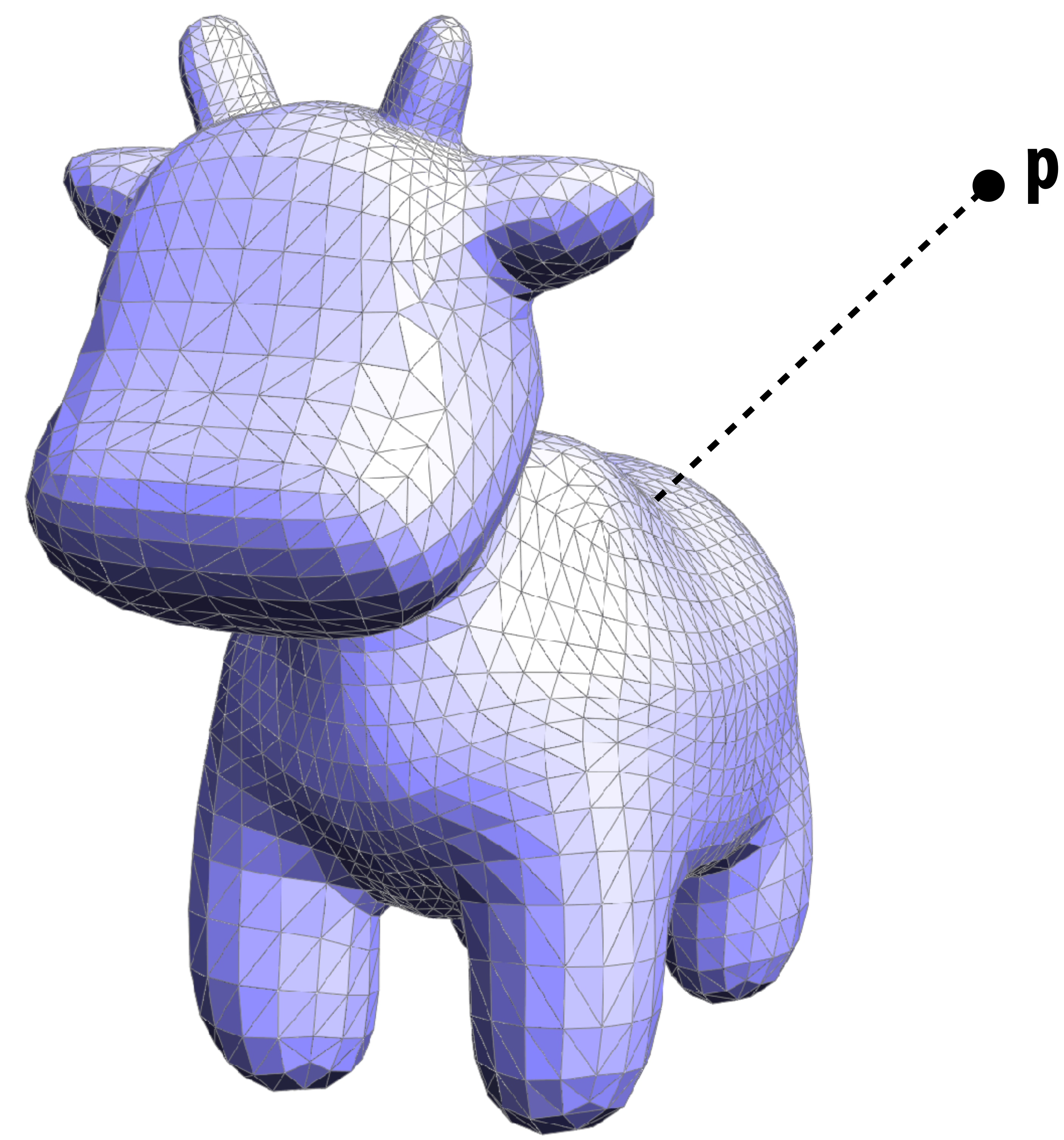
- Not so different from 2D case
- Algorithm:
  - Project point onto plane of triangle
  - Use half-*space* tests to classify point (vs. half plane)
  - If inside the triangle, we're done!
  - Otherwise, find closest point on associated vertex or edge

- By the way, how do we find closest point on plane?
- Same expression as closest point on a line! $p + (c - N^T p) N$

Closest point on triangle *mesh* in 3D?

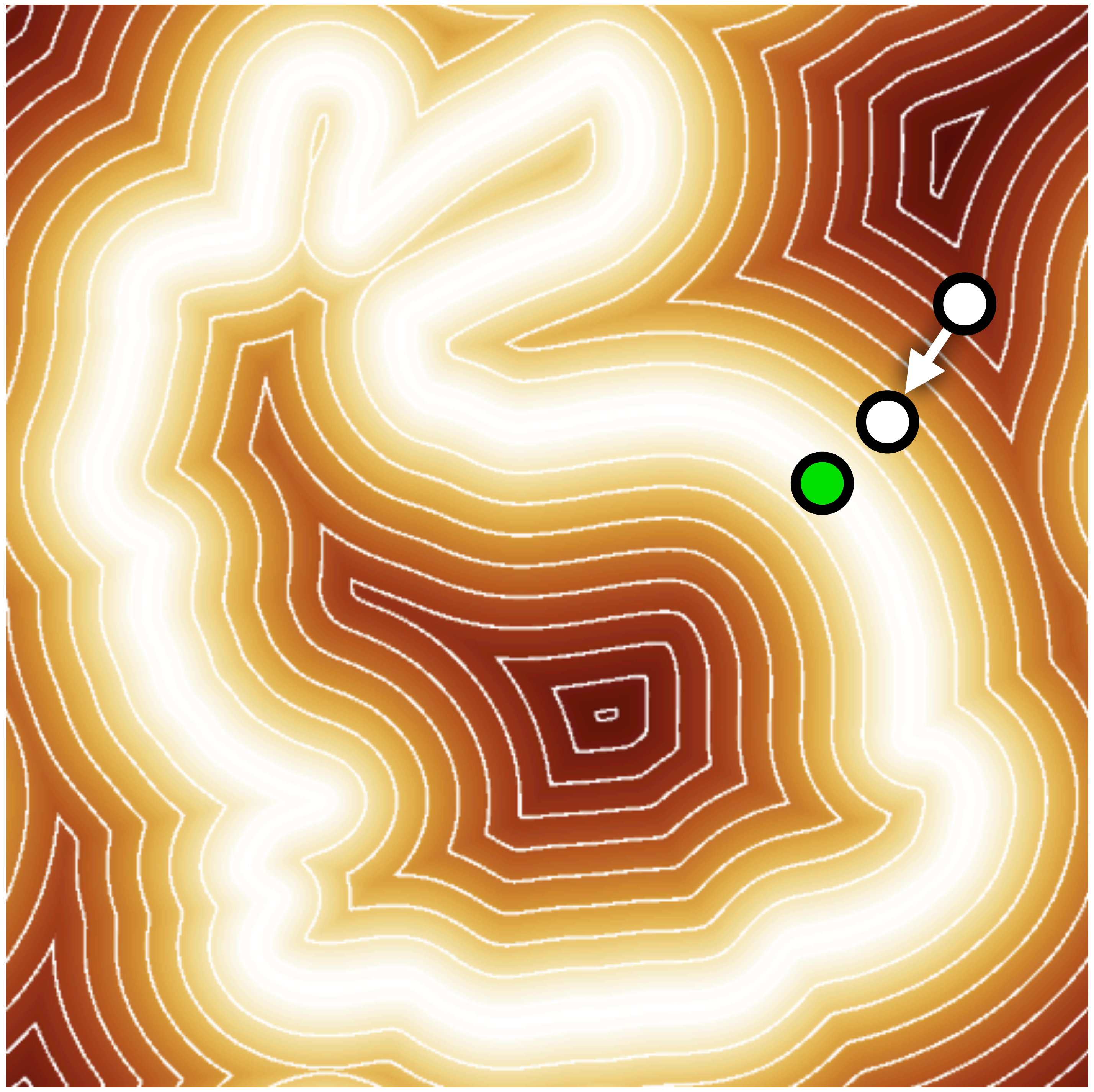
- **Conceptually easy:**
  - loop over all triangles
  - compute closest point to current triangle
  - keep globally closest point
- **Q: What's the cost?**
- **What if we have *billions* of faces?**
- **NEXT TIME: Better data structures!**





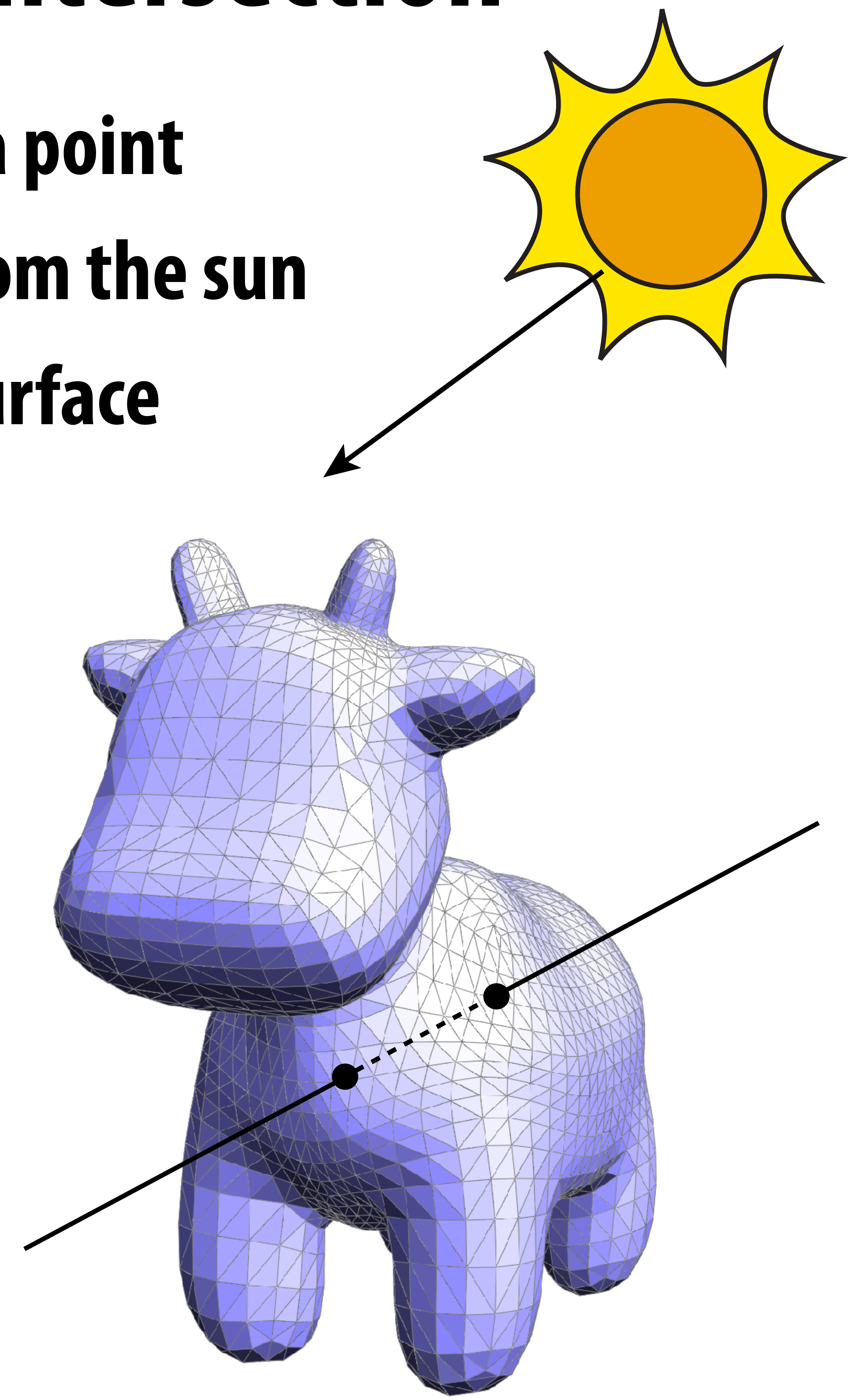
Closest point to *implicit* surface?

- 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

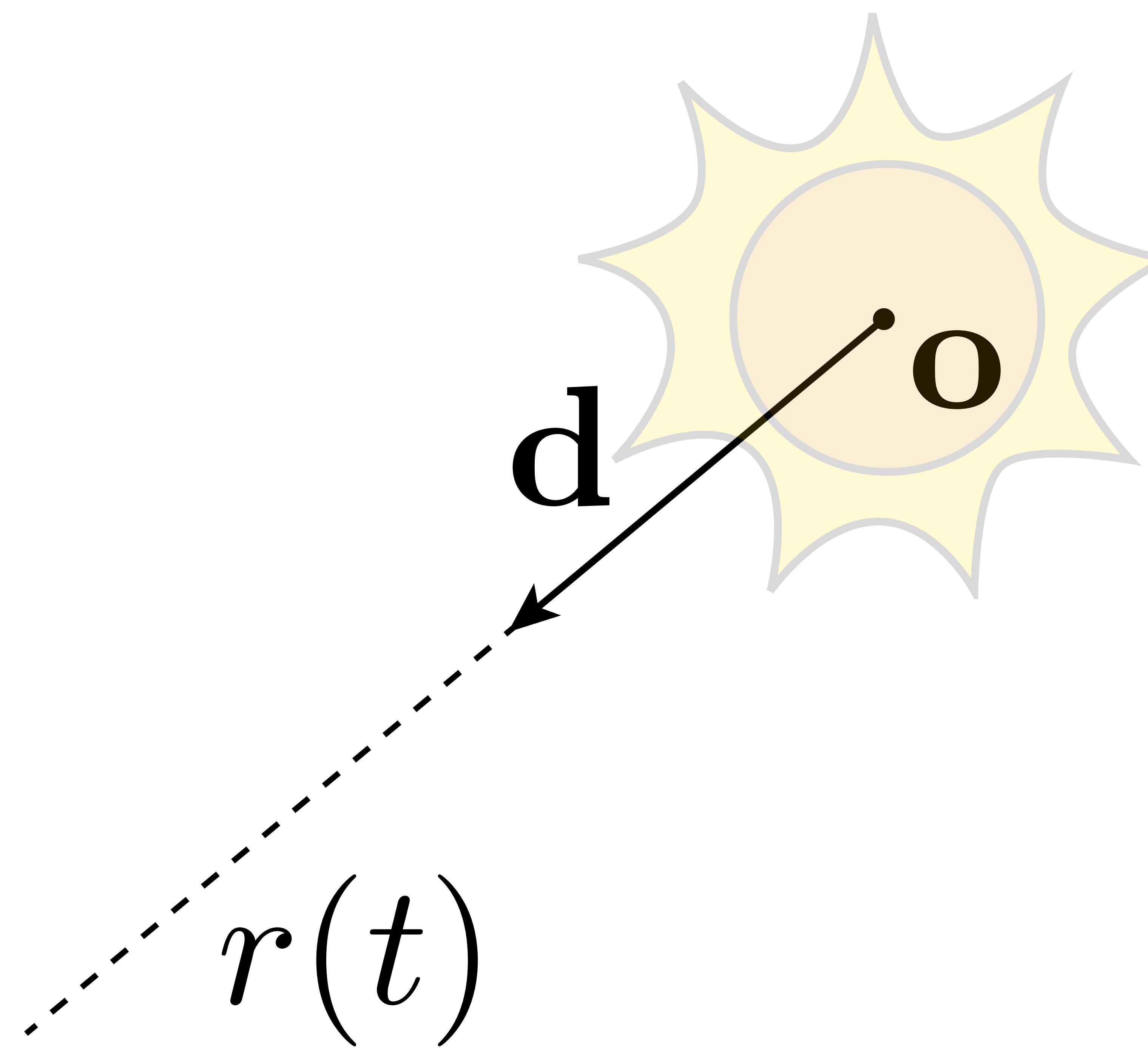
origin

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

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

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

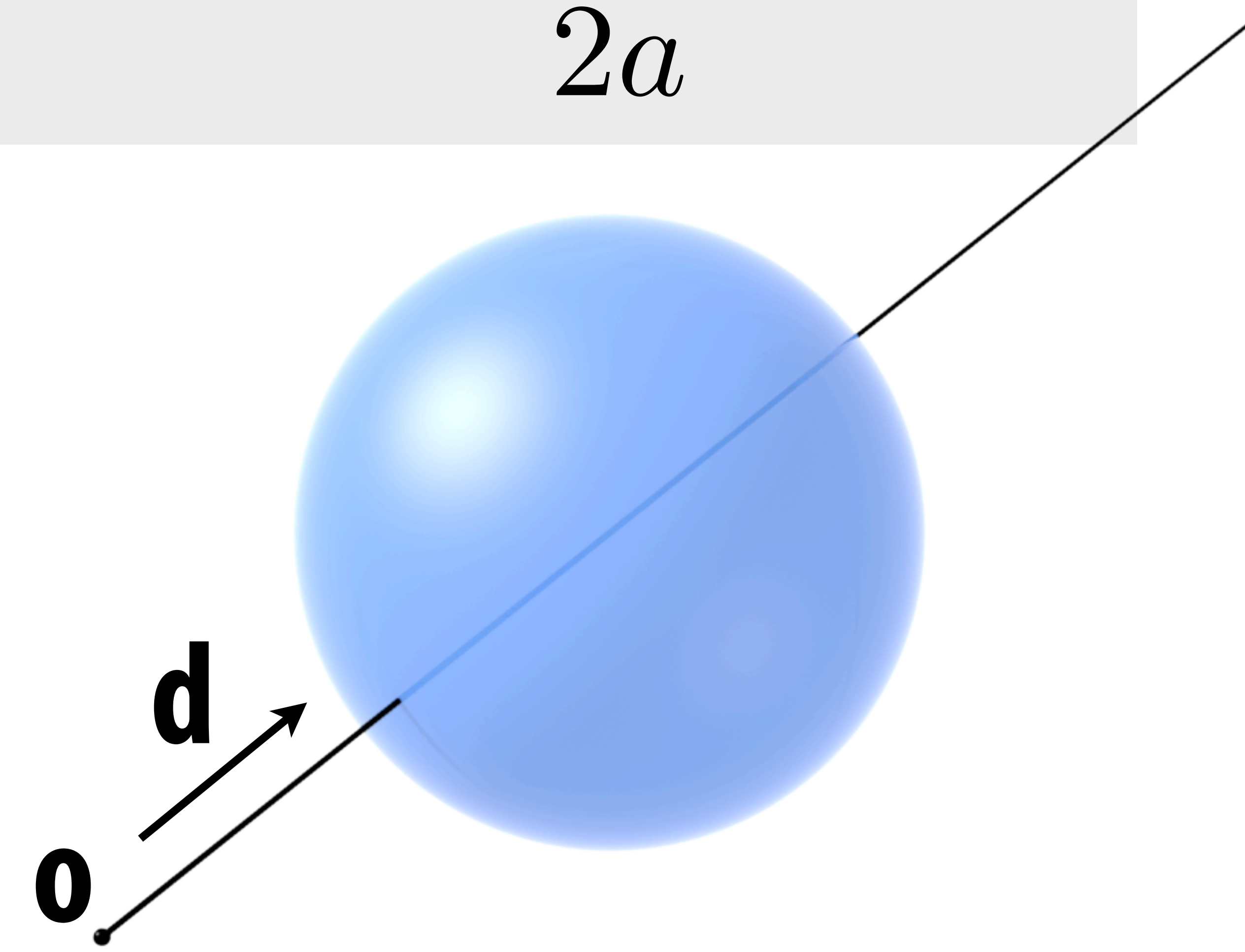
$$\underbrace{|\mathbf{d}|^2}_{a} t^2 + \underbrace{2(\mathbf{o} \cdot \mathbf{d})}_{b} t + \underbrace{|\mathbf{o}|^2 - 1}_{c} = 0$$

Note: $|\mathrm{d}|^2 = 1$ since d is a unit vector

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

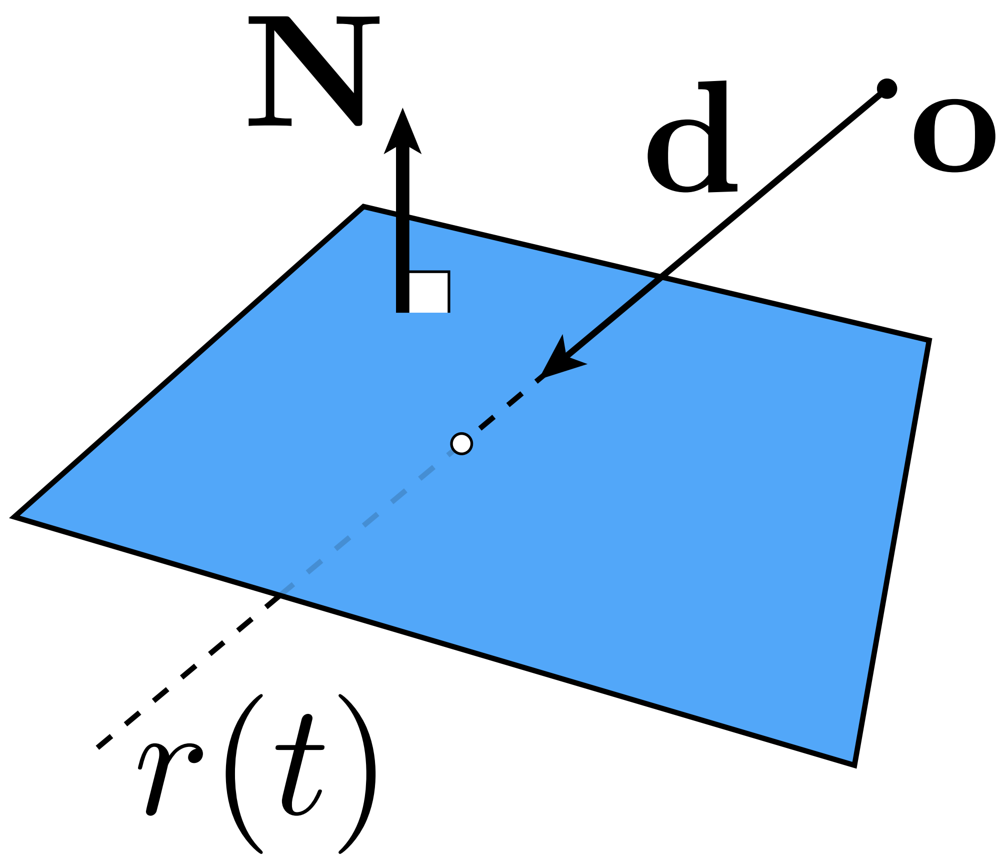
quadratic formula:

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



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:

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

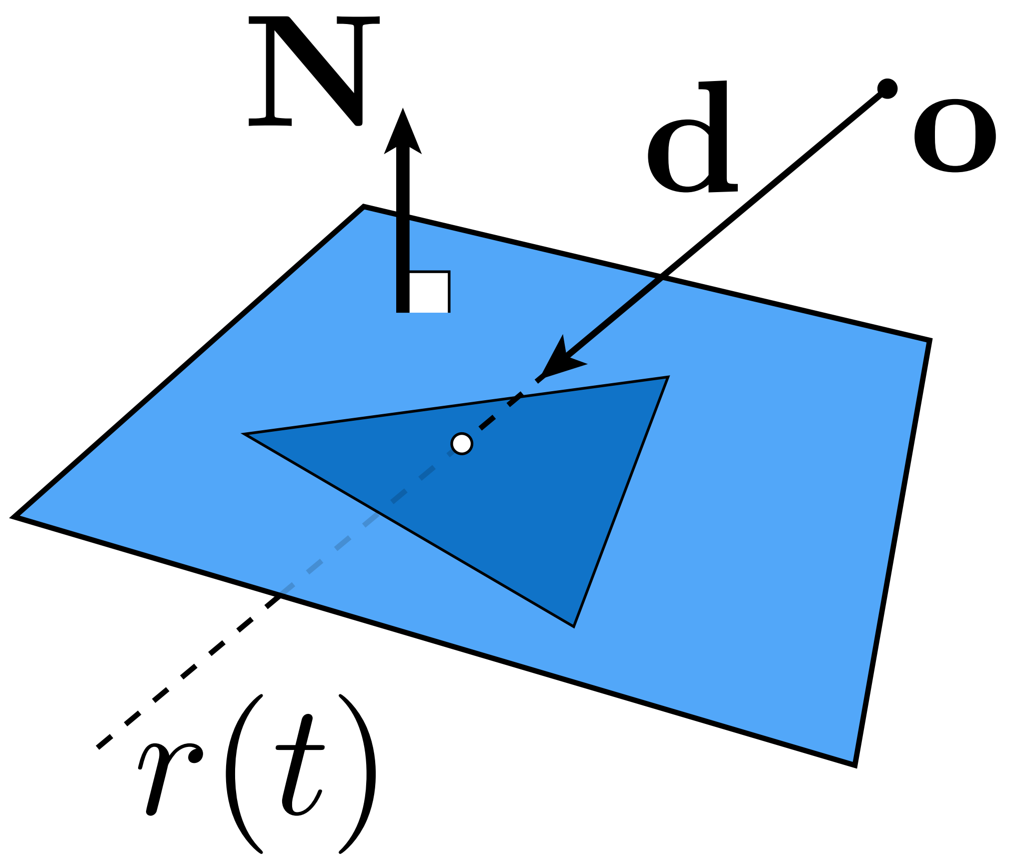
- Now solve for t:

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

- And plug t back into ray equation:

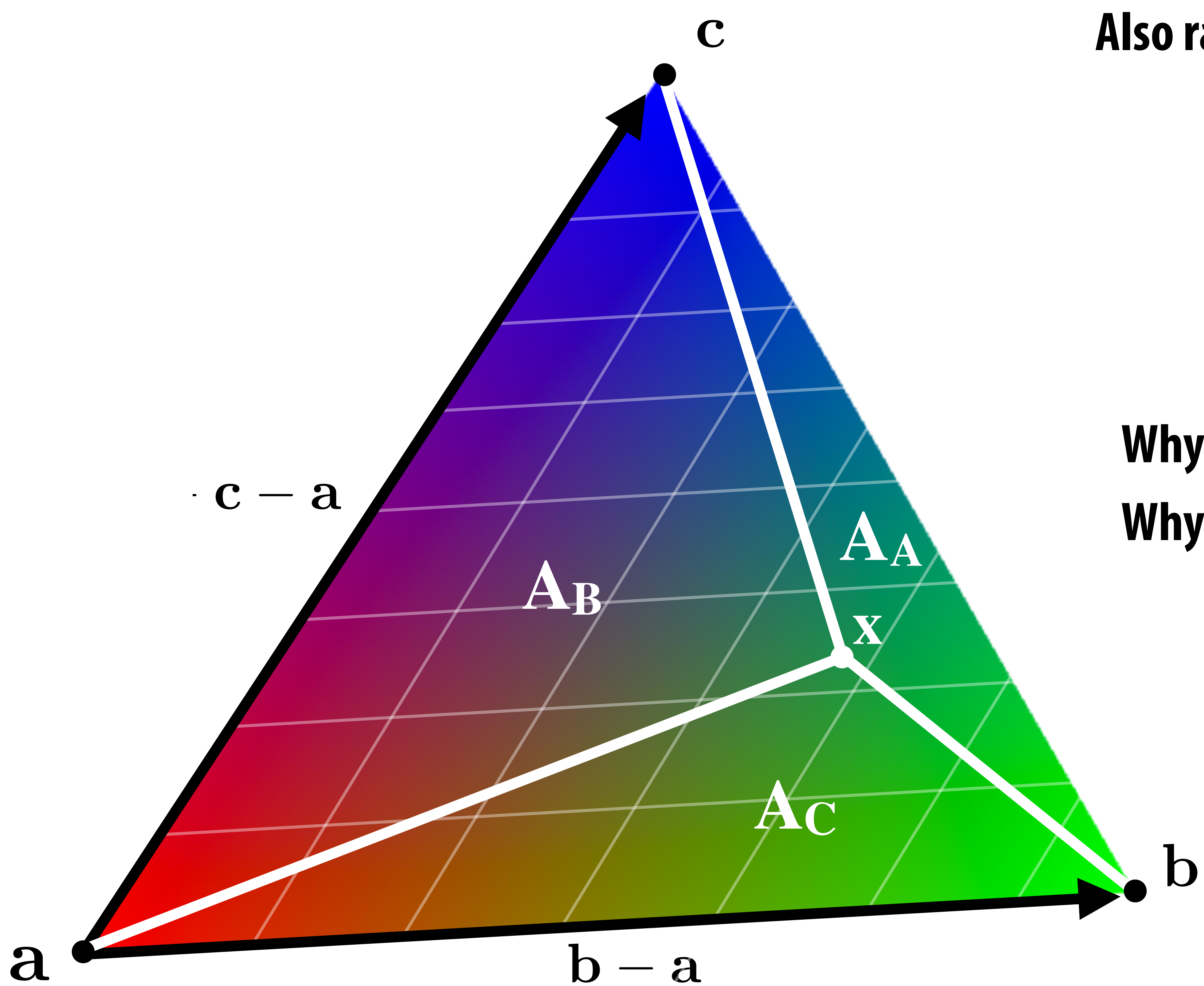
$$r(t) = \mathbf{o} + \frac{c - \mathbf{N}^\mathsf{T}\mathbf{o}}{\mathbf{N}^\mathsf{T}\mathbf{d}}\mathbf{d}$$

Ray-triangle intersection



- Triangle is in a plane...
- Algorithm:
  - Compute ray-plane intersection
  - Q: What do we do now?

Barycentric coordinates (as ratio of areas)



Also ratio of *signed* areas:

$$\alpha = A_A/A$$

$$\beta = A_B/A$$

$$\gamma = A_C/A$$

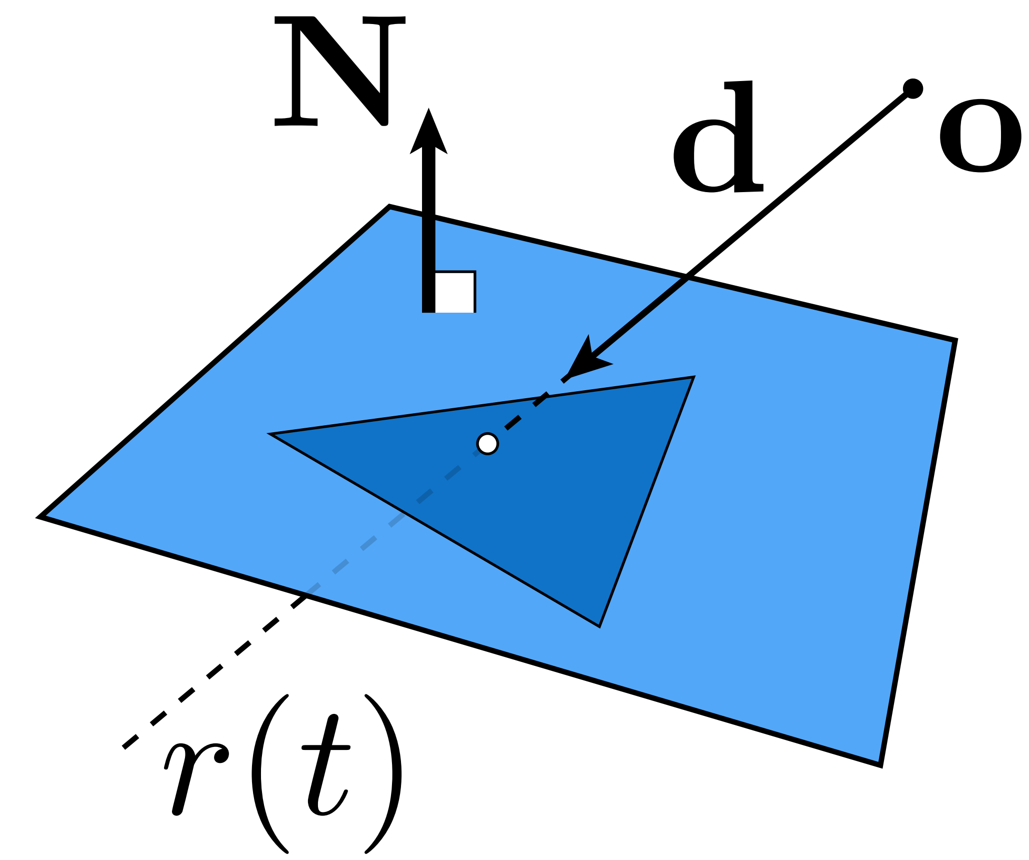
Why must coordinates sum to one?

Why must coordinates be between 0 and 1?

Useful: Heron's formula:

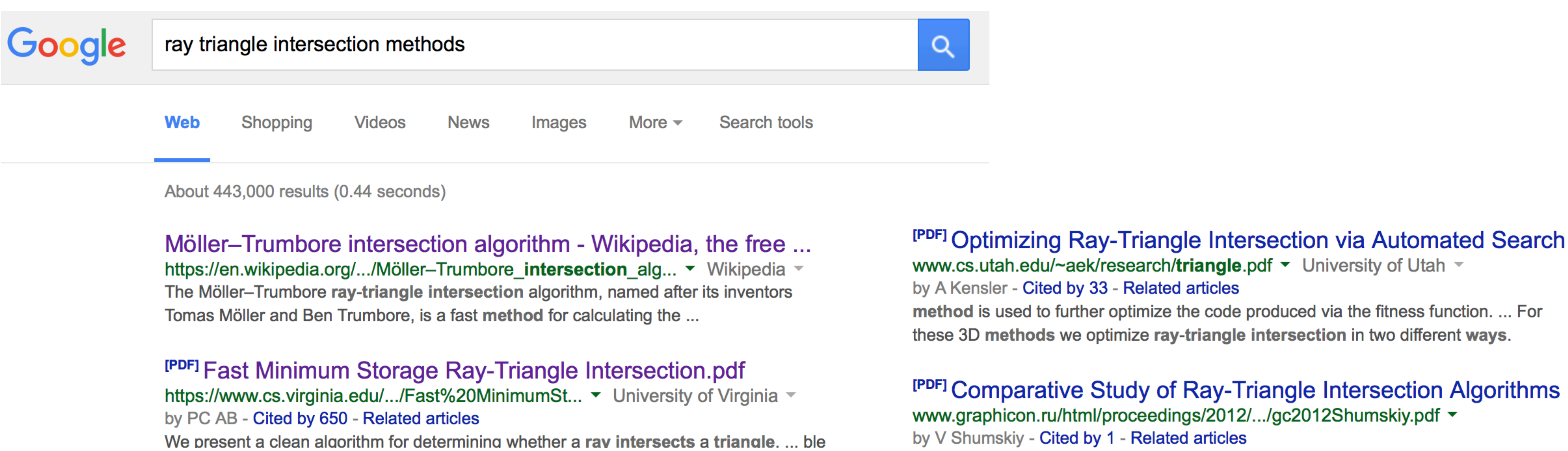
$$A_C = \frac{1}{2}(\mathbf{b} - \mathbf{a}) \times (\mathbf{x} - \mathbf{a})$$

Ray-triangle intersection



- **Algorithm:**
  - Compute ray-plane intersection
  - Q: What do we do now?
  - A: Compute barycentric coordinates of hit point?
  - If barycentric coordinates are all positive, point is in triangle

- **Many different techniques if you care about efficiency**



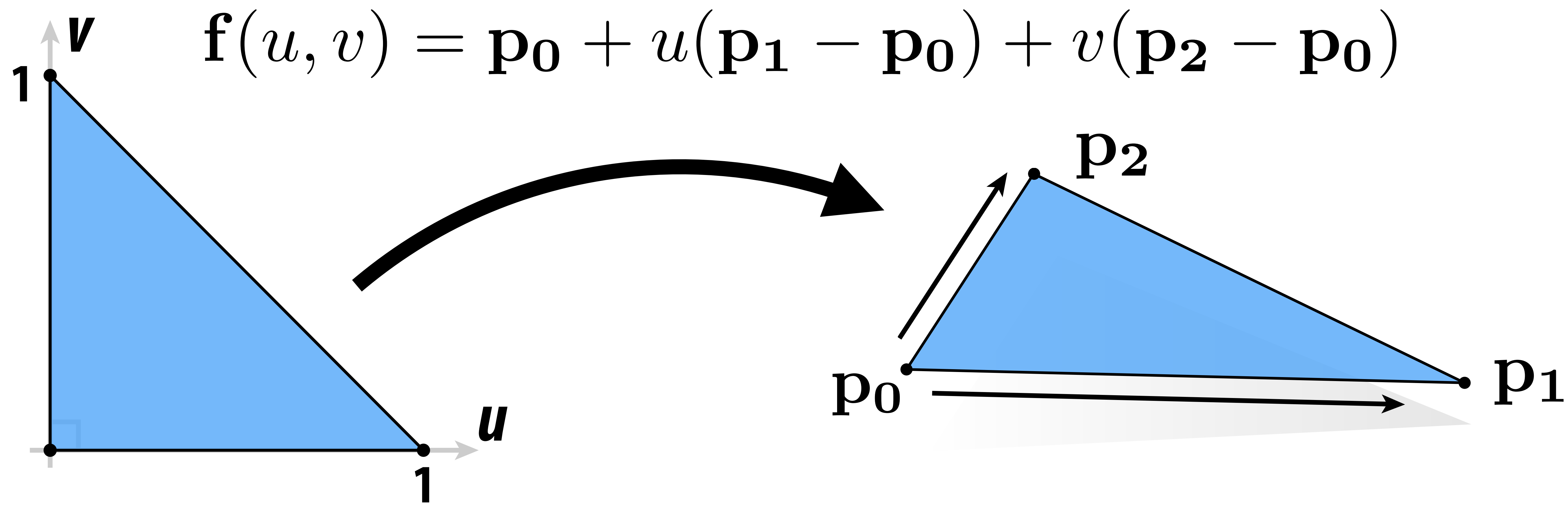
Another way: ray-triangle intersection

- **Parameterize triangle given by vertices $\mathbf{p_0}, \mathbf{p_1}, \mathbf{p_2}$ using barycentric coordinates**

$$f(u,v) = (1-u-v)\mathbf{p_0} + u\mathbf{p_1} + v\mathbf{p_2}$$

- **Can think of a triangle as an affine map of the unit triangle**

$$\mathbf{f}(u,v) = \mathbf{p_0} + u(\mathbf{p_1} - \mathbf{p_0}) + v(\mathbf{p_2} - \mathbf{p_0})$$



Another way: ray-triangle intersection

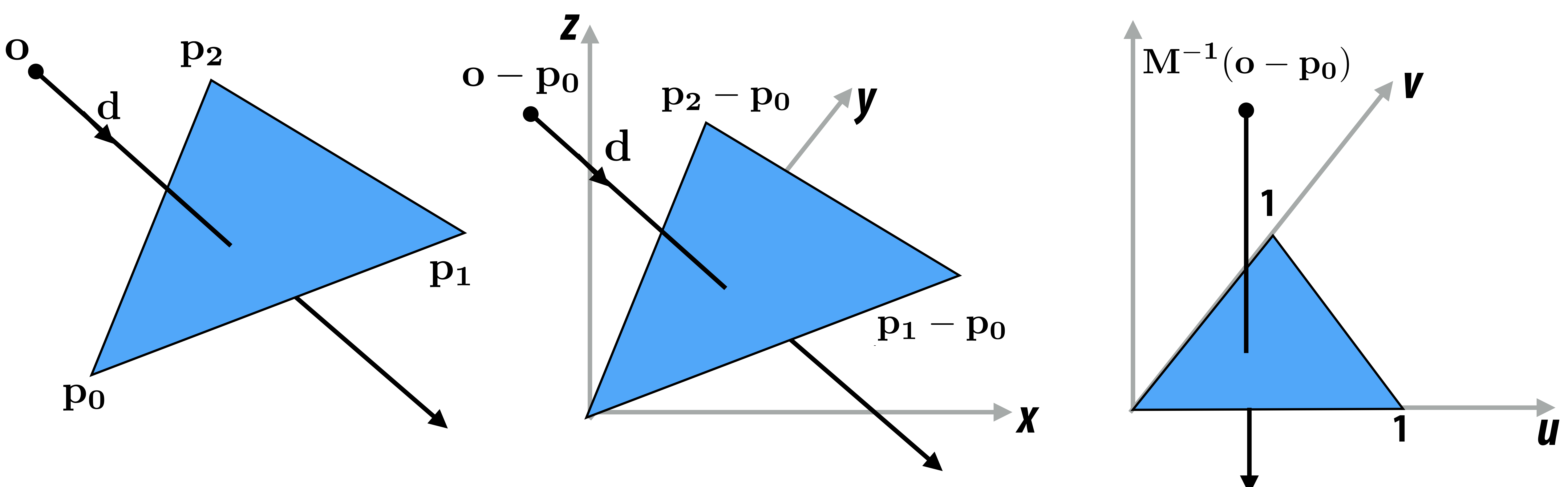
Plug parametric ray equation directly into equation for points on triangle:

$$\mathbf{p}_0 + u(\mathbf{p}_1 - \mathbf{p}_0) + v(\mathbf{p}_2 - \mathbf{p}_0) = \mathbf{o} + t\mathbf{d}$$

Solve for u, v, t:

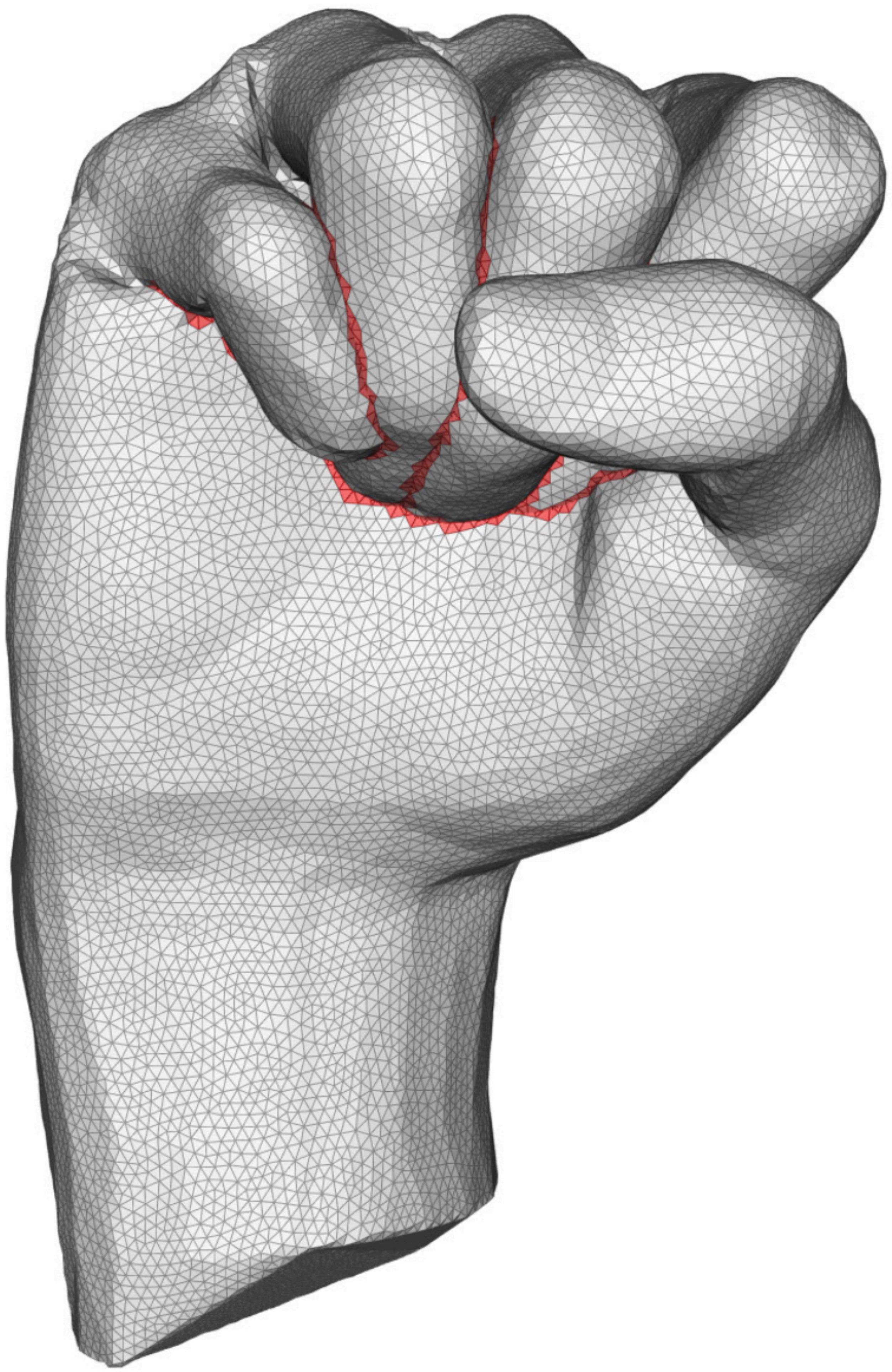
$$\underbrace{\begin{bmatrix}\mathbf{p}_1 - \mathbf{p}_0 & \mathbf{p}_2 - \mathbf{p}_0 & -\mathbf{d}\end{bmatrix}}_{\mathbf{M}} \begin{bmatrix} u \\ v \\ t \end{bmatrix} = \mathbf{o} - \mathbf{p}_0$$

M^{-1} **transforms triangle back to unit triangle in u,v plane, and transforms ray's direction to be orthogonal to plane. It's a point in 2D triangle test now!**



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?**

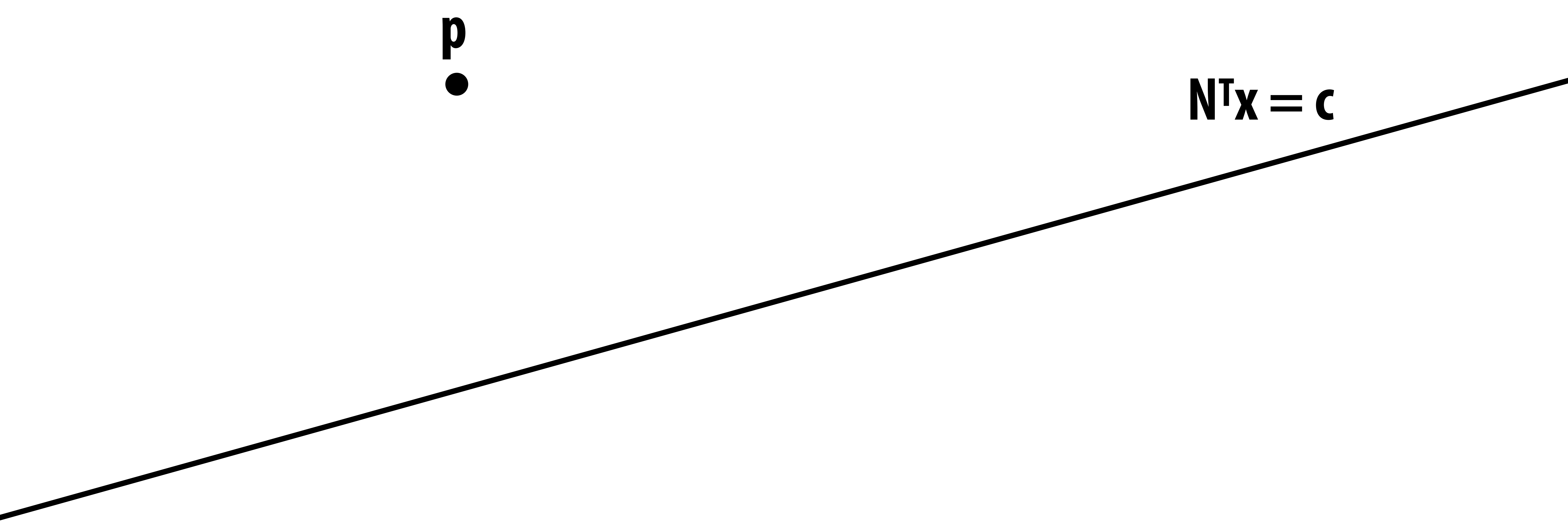


Warm up: point-point intersection

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

Slightly harder: point-line intersection

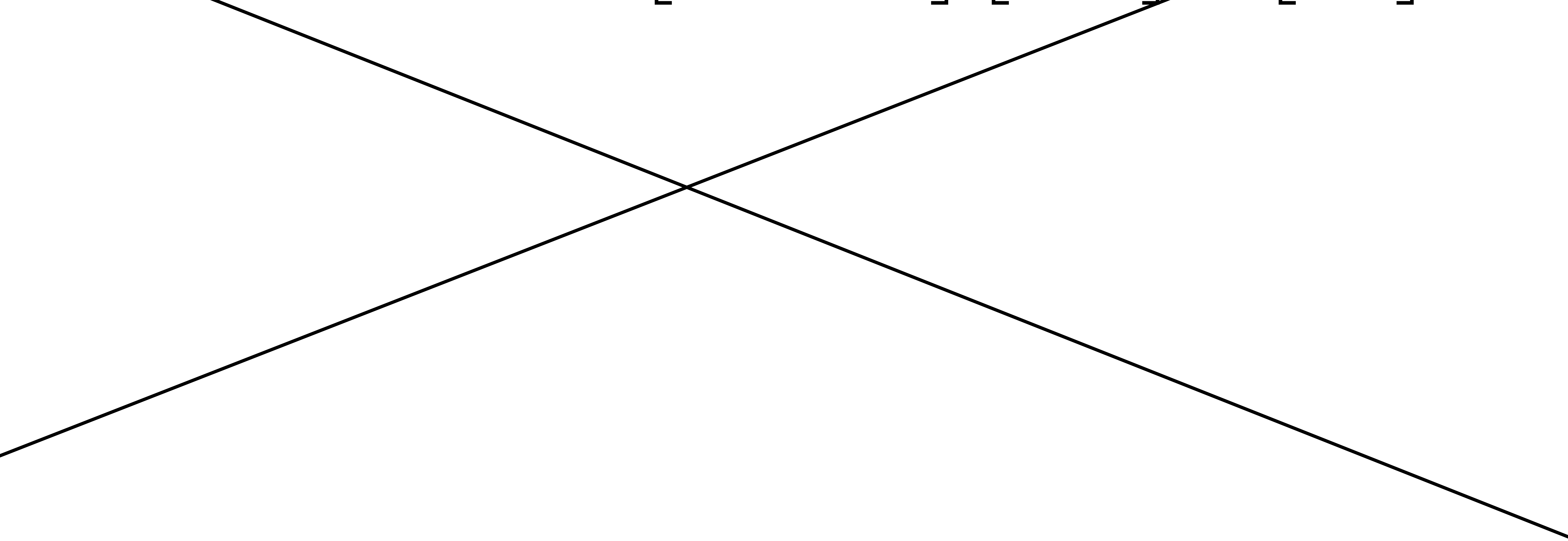
- Q: How do we know if a point intersects a given line?
- A: ...plug it into the line equation!



Line-line intersection

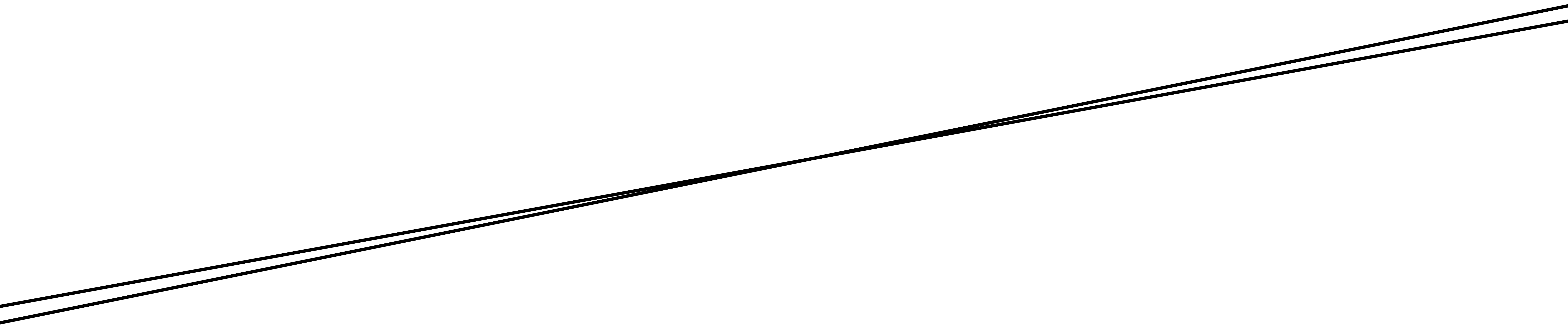
- **Two lines: ax=b and cx=d**
- **Q: How do we find the intersection?**
- **A: See if there is a simultaneous solution**
- **Leads to linear system:**

$$\begin{bmatrix} a_1 & a_2 \\ c_1 & c_2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} b \\ d \end{bmatrix}$$



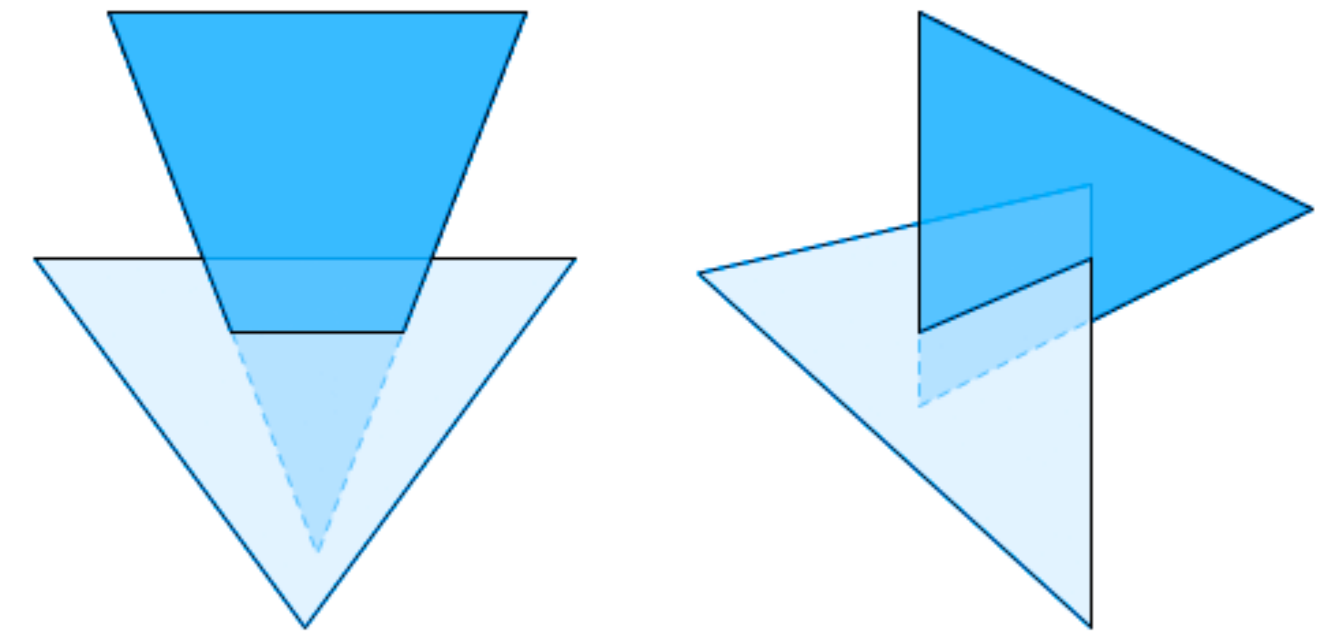
Degenerate line-line intersection?

- 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).

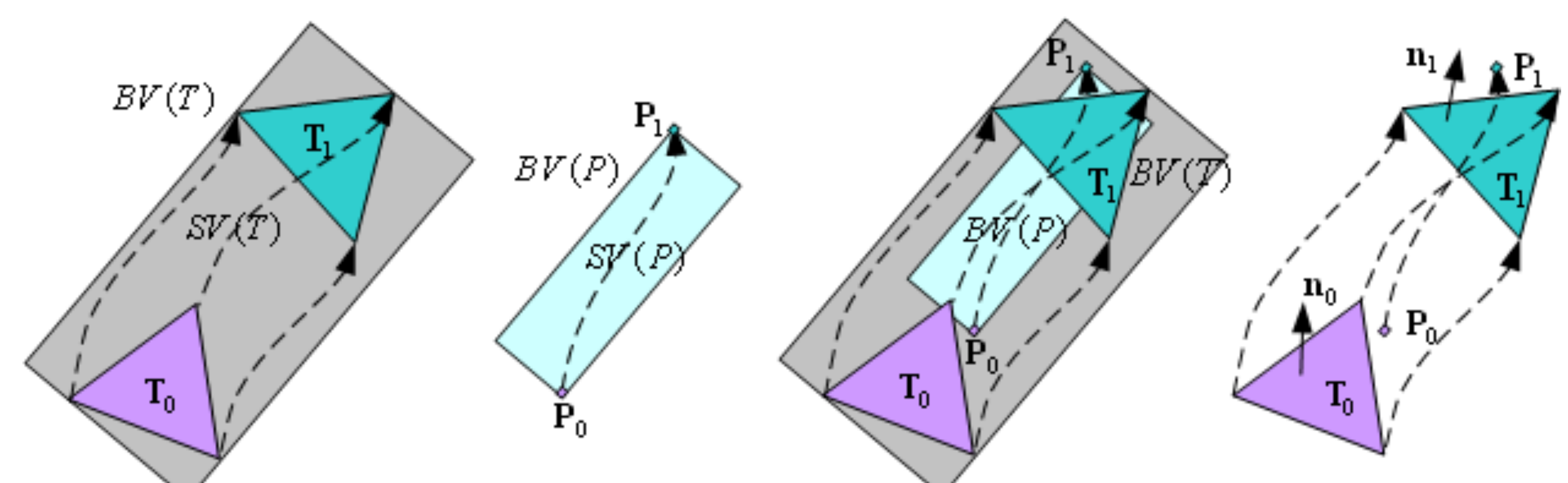


See for example Shewchuk, “*Adaptive Precision Floating-Point Arithmetic and Fast Robust Geometric Predicates*”

Triangle-triangle intersection?



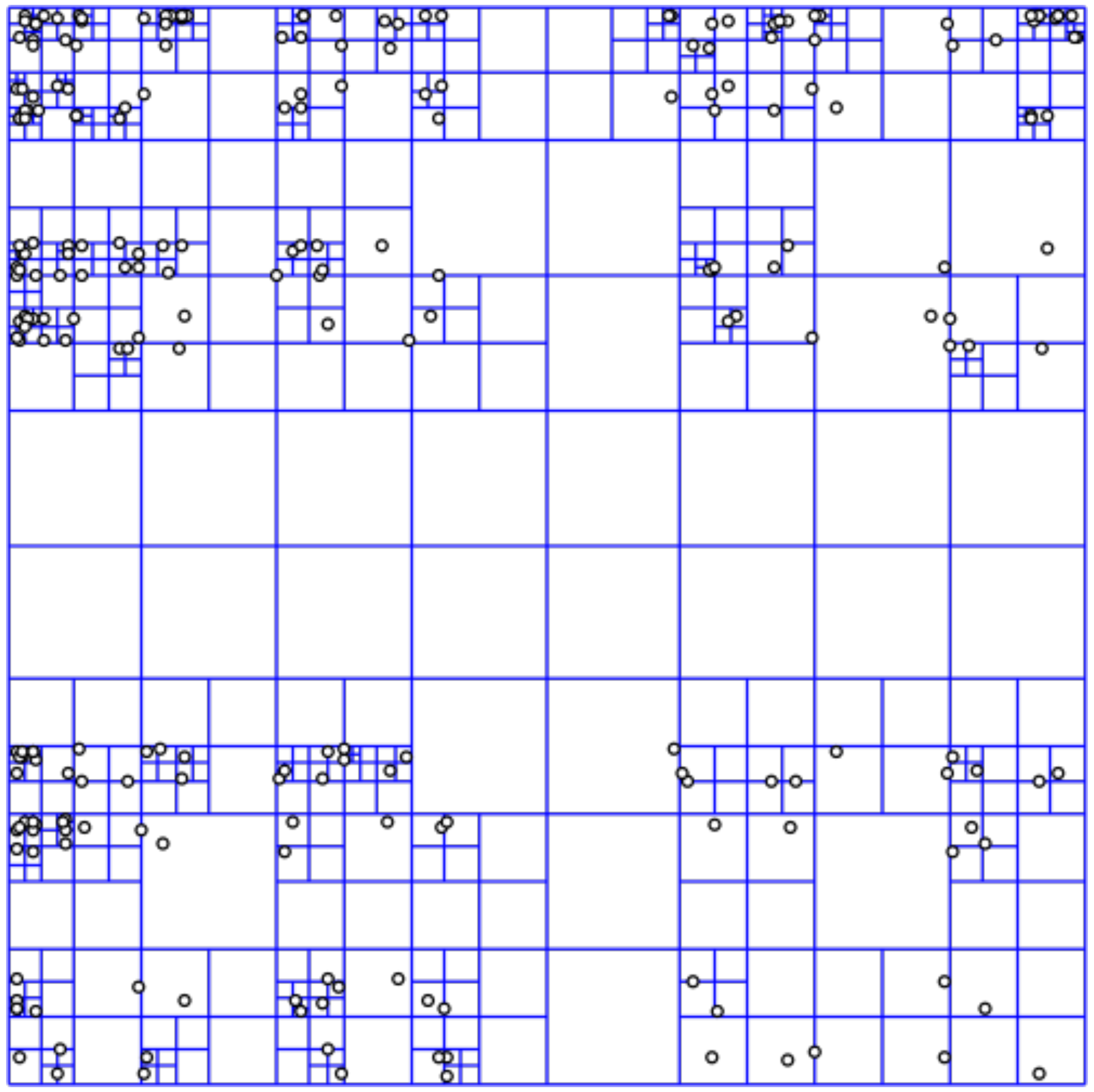
- Lots of ways to do it
- Basic idea:
  - Q: Any ideas?
  - One way: reduce to edge-triangle intersection
  - Check if each line passes through plane (ray-triangle)
  - Then do interval test
- What if triangle is *moving*?
  - Important case for animation
  - Can think of triangles as *prisms* in time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

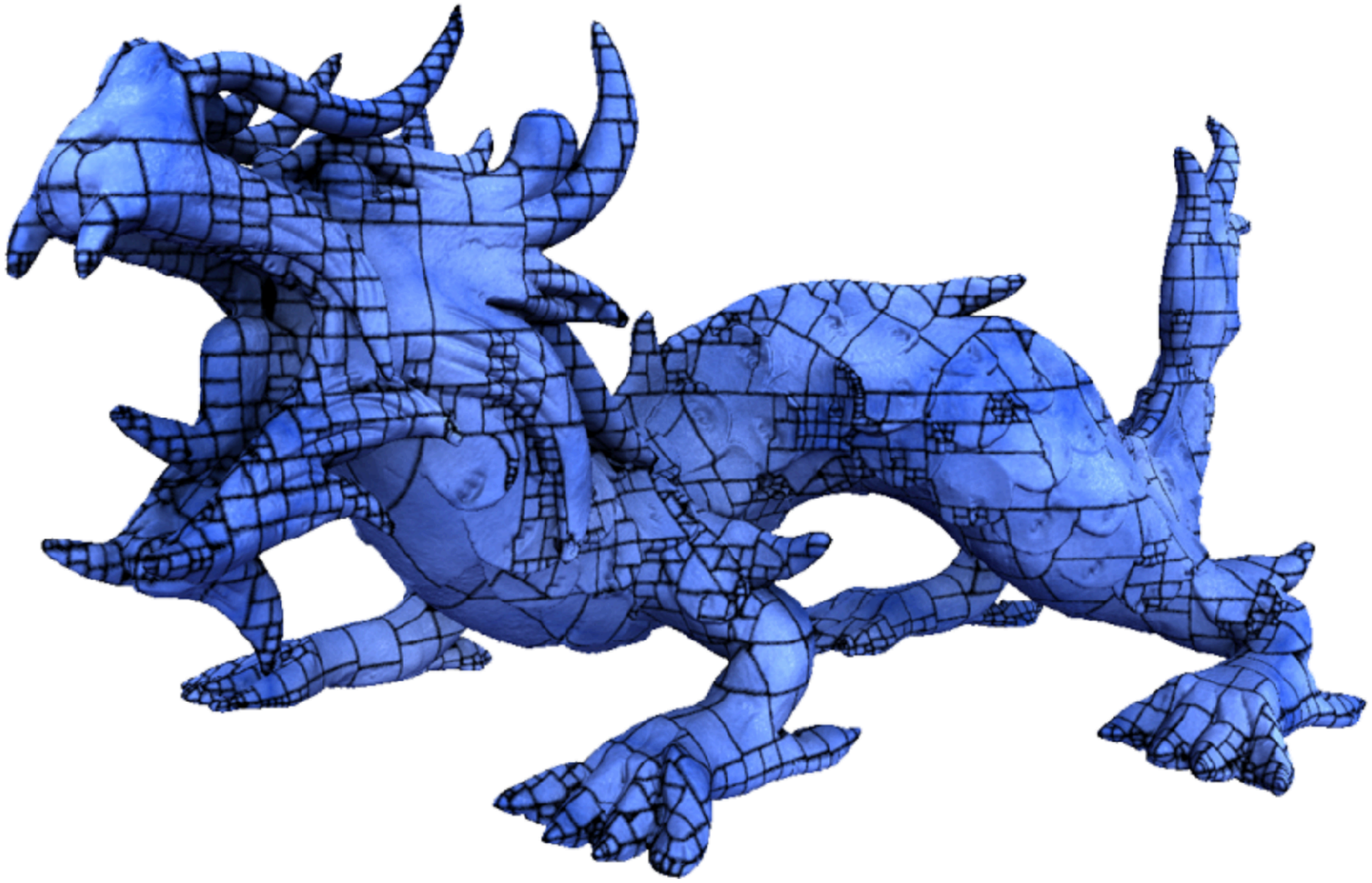


(a) Bounding volume of a deforming triangle (b) Bounding volume of a deforming vertex (c) Bounding volume test (d) Coplanarity test

Next time: spatial acceleration data structures

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





Acknowledgements

- Thanks to Keenan Crane for presentation resources