

## **Lecture 8**

# **Deformable Models (Part 2)**

---

**FUNDAMENTALS OF COMPUTER GRAPHICS  
Animation & Simulation**

**Stanford CS248B, Fall 2022**

**PROFS. KAREN LIU & DOUG JAMES**

# Announcements

- Written HW2 out
- P1 Pinball due next Tuesday
- P2 out next week

# Re: P1: Pinball: "Triangle trick" for finite difference gradient

## Contact Normals: Finite Difference Approximation to 2D SDF Gradient

The normal at the ball-environment contact point is the same as normal at the ball-center point with an environment inflated by  $r$ . It is sufficient to compute the gradient of  $D$  at  $p$ , and use that as the unnormalized outward normal. Since computing analytical SDF gradients is not always easy or possible, it is common practice to use finite difference approximations of which there are multiple choices:

- **One-sided finite differences** (implemented in the starter code as `normalFD1(p)`) are simple but only first-order accurate,  $O(h)$ :
  - The x-component is  $\frac{\partial D(p)}{\partial x} = \frac{D(p+h\hat{x}) - D(p)}{h}$
  - The y-component is  $\frac{\partial D(p)}{\partial y} = \frac{D(p+h\hat{y}) - D(p)}{h}$
  - Note that when in contact, we have  $D(p) \approx 0$  and so there is essentially only one additional SDF evaluation for each component at a small distance offset,  $h$ .
- **Two-sided finite differences** (you should implement in your code as `normalFD2(p)`) are more accurate,  $O(h^2)$ , but require two function evaluations:
  - The x-component is  $\frac{\partial D(p)}{\partial x} = \frac{D(p+h\hat{x}) - D(p-h\hat{x})}{2h}$
  - The y-component is  $\frac{\partial D(p)}{\partial y} = \frac{D(p+h\hat{y}) - D(p-h\hat{y})}{2h}$
  - Implement this in the starter code. These will help give you more contact precision for detailed geometry and smaller pin balls.
- **Triangle trick:** FYI, there is a way to compute second-order accurate,  $O(h^2)$ , gradients with only 3 function evaluations with a triangle stencil. The approach is analogous to how a tetrahedral sampling is used to compute the 3D gradient for normals (see <https://iquilezles.org/articles/normalsSDF>). If you are particularly interested, try doing the triangle version instead of `normalFD2`.

# "Triangle trick" for 2D centered differences of $\nabla f$

- The second-order accurate centered-difference approximation takes 4 function evals:

$$\nabla f(\mathbf{p}) = \frac{1}{2h} \begin{pmatrix} f(\mathbf{p} + h\hat{\mathbf{x}}) - f(\mathbf{p} - h\hat{\mathbf{x}}) \\ f(\mathbf{p} + h\hat{\mathbf{y}}) - f(\mathbf{p} - h\hat{\mathbf{y}}) \end{pmatrix} + O(h^2), \quad \text{where } \mathbf{p} \in \mathbb{R}^2.$$

- The triangle stencil for  $\nabla f$  only takes 3 function evals and is also  $O(h^2)$ 
  - Similar idea for tetrahedral sampling of 3D gradients
    - See <https://iquilezles.org/articles/normalsSDF>
- Idea: Linear approximation determined by just 3 values.

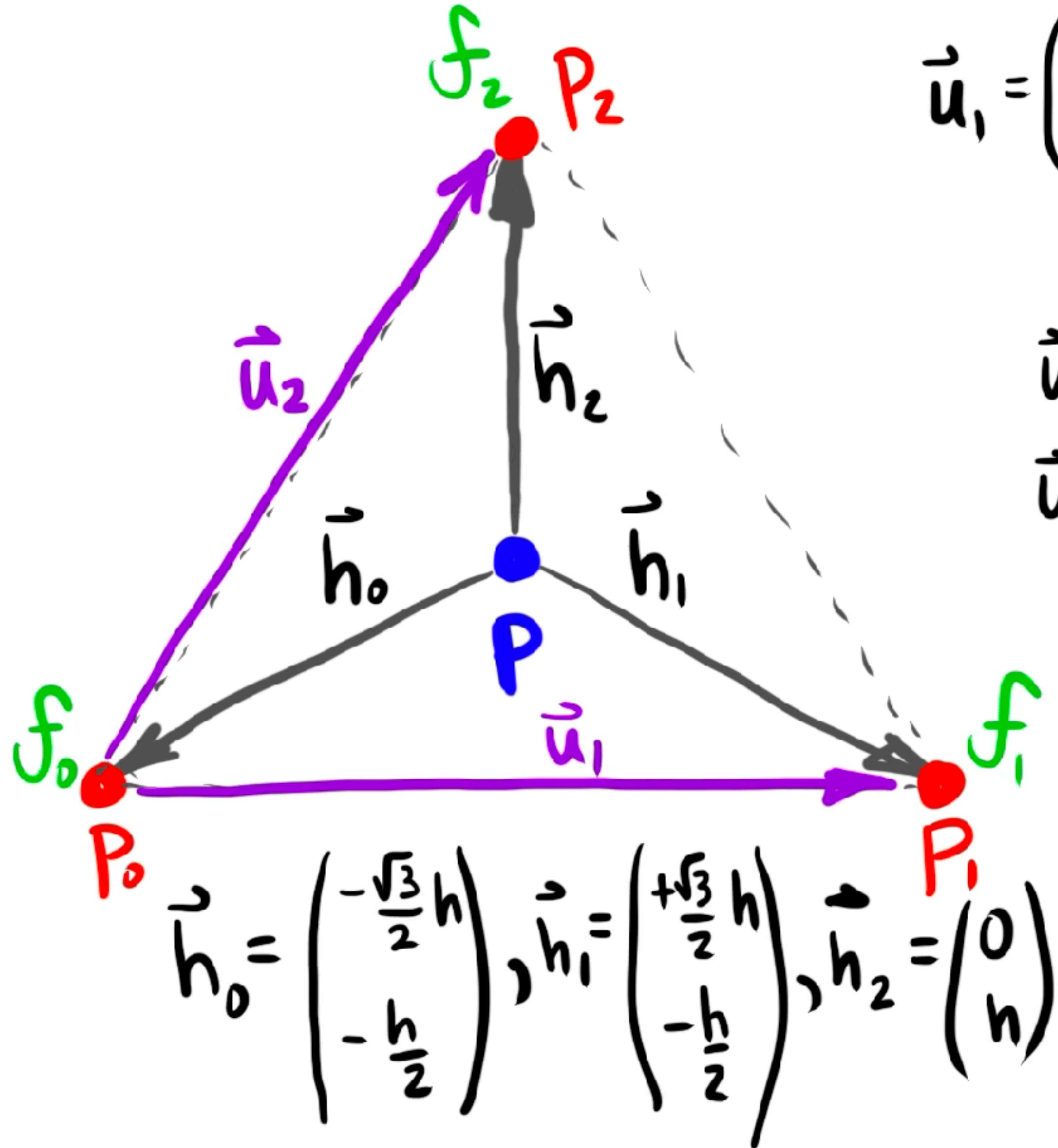
$$f(\mathbf{p} + \Delta\mathbf{p}) \approx f(\mathbf{p}) + \mathbf{g}^T \Delta\mathbf{p} = f(\mathbf{p}) + g_x \delta x + g_y \delta y$$

Rearranging

$$\mathbf{g}^T \Delta\mathbf{p} = \Delta f = f(\mathbf{p} + \Delta\mathbf{p}) - f(\mathbf{p})$$

Just need to know two  $\Delta f$  estimates to solve for the two unknowns,  $\mathbf{g} \in \mathbb{R}^2$ .

# "Triangle trick" for 2D centered differences of $\nabla f$



$$\vec{u}_1 = \begin{pmatrix} \sqrt{3}h \\ 0 \end{pmatrix}, \quad \vec{u}_2 = \begin{pmatrix} \frac{\sqrt{3}}{2}h \\ \frac{3}{2}h \end{pmatrix}, \quad \Delta f_i \equiv f_i - f_0$$

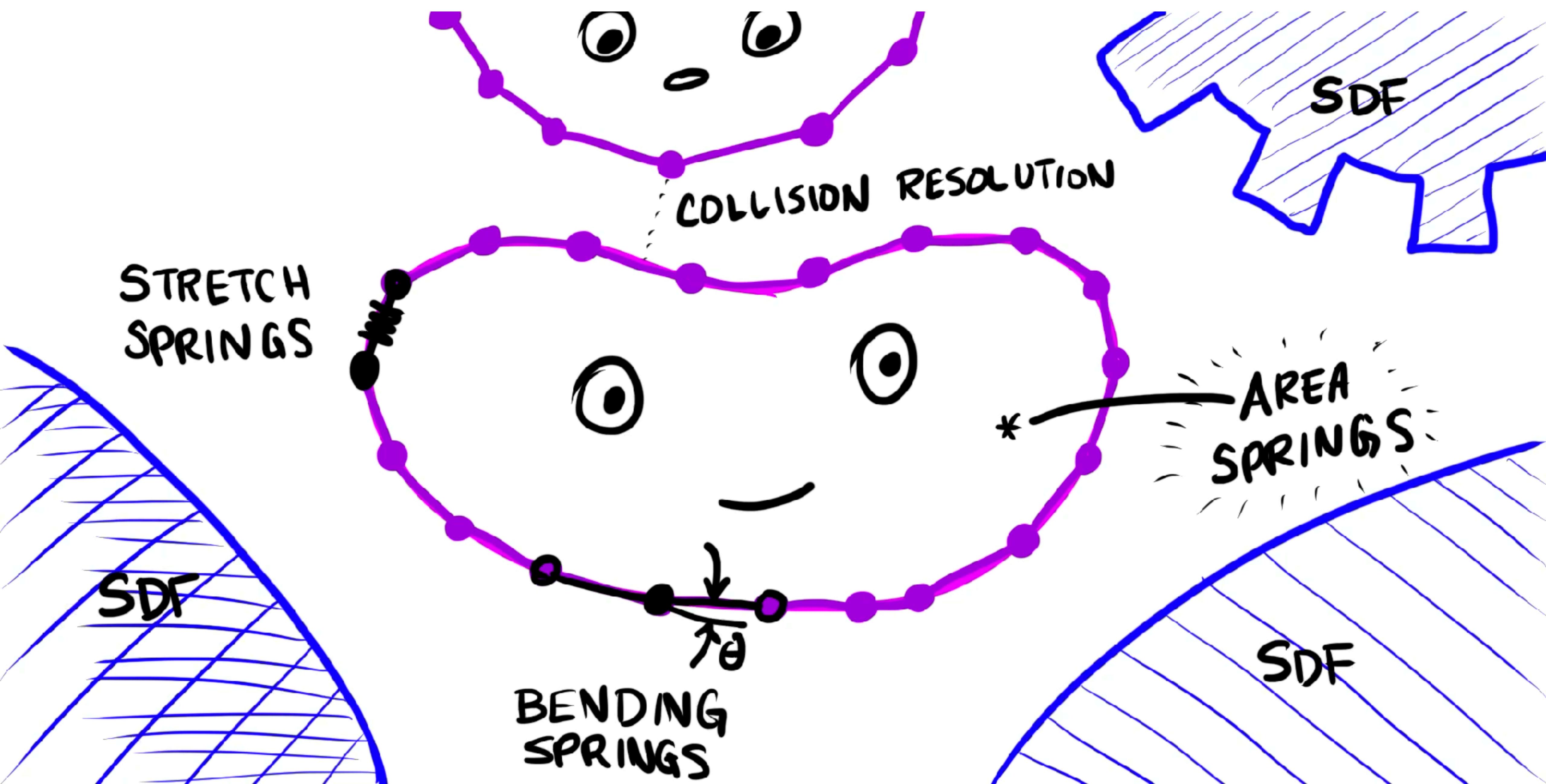
$$\begin{aligned} \vec{u}_1^\top \vec{g} &= \Delta f_1 \\ \vec{u}_2^\top \vec{g} &= \Delta f_2 \end{aligned} \quad \left\{ \begin{bmatrix} -\vec{u}_1^\top \\ -\vec{u}_2^\top \end{bmatrix} \vec{g} = U \vec{g} = \vec{\Delta f} \right.$$

$$\Rightarrow \boxed{\vec{g} = U^{-1} \vec{\Delta f} = \begin{bmatrix} \sqrt{3}h & 0 \\ \frac{\sqrt{3}h}{2} & \frac{3h}{2} \end{bmatrix}^{-1} \vec{\Delta f}}$$

$$= \frac{1}{h} \begin{pmatrix} \frac{1}{\sqrt{3}} & 0 \\ -\frac{1}{3} & \frac{2}{3} \end{pmatrix} \vec{\Delta f} = \boxed{\begin{pmatrix} \frac{\Delta f_1}{\sqrt{3}h} \\ -\frac{\Delta f_1 + 2\Delta f_2}{3h} \end{pmatrix}}$$

# **Other force models**

# P2: Deformable blob simulation



# Controlling Compression in 2D: Area springs

REST SHAPE  
(UNDEFORMED)



DEFORMED



POTENTIAL  
ENERGY:

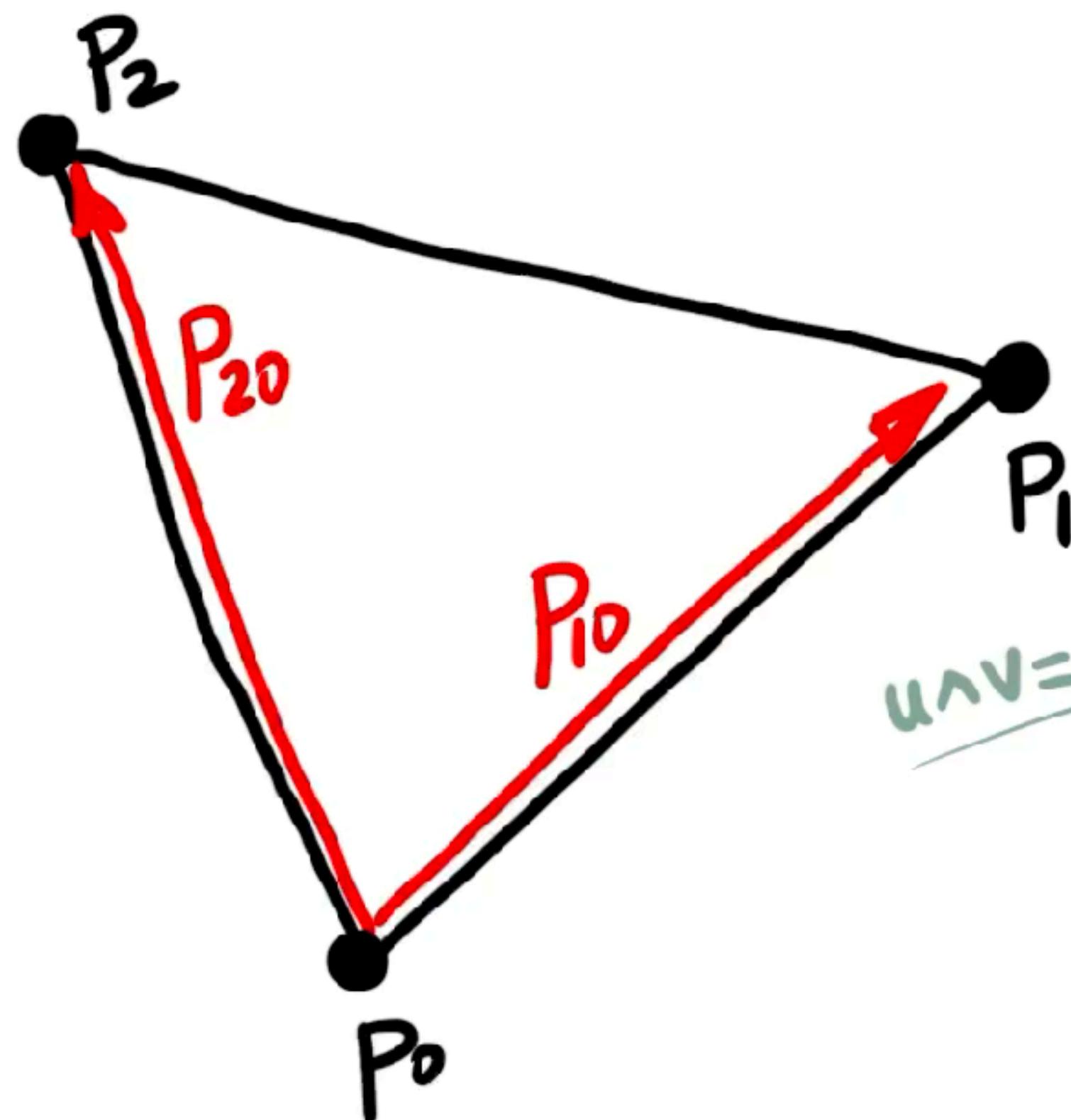
$$E = \frac{1}{2} k (A_{(P)} - A_0)^2$$

ELASTIC FORCE:

$$f_i = -k (A - A_0) \underline{\partial_i A_{(P)}}$$

# Area of a 2D polygon

Consider the area of a triangle:

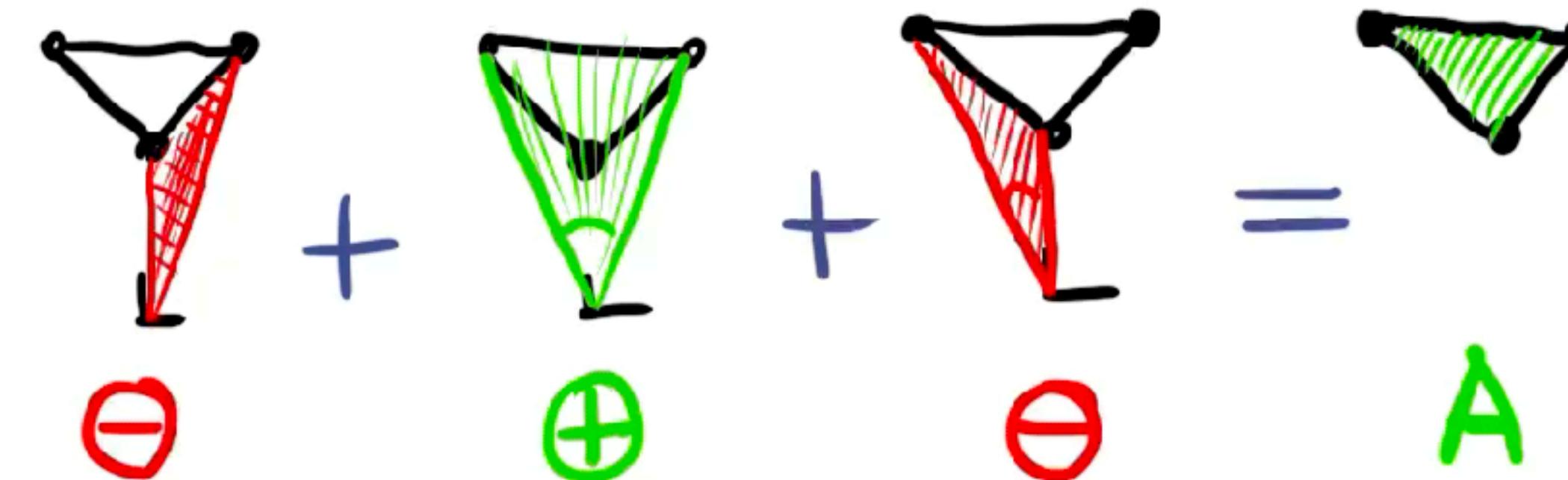


$$2A = \overrightarrow{P_{10}} \wedge \overrightarrow{P_{20}}$$
$$= (\vec{P}_1 - \vec{P}_0) \wedge (\vec{P}_2 - \vec{P}_0)$$

$\vec{u} \wedge \vec{v} \equiv u_x v_y - u_y v_x$  $= \begin{pmatrix} -u_y \\ u_x \end{pmatrix}^T \begin{pmatrix} v_x \\ v_y \end{pmatrix}$  $= \vec{u}_{\perp} \cdot \vec{v}$

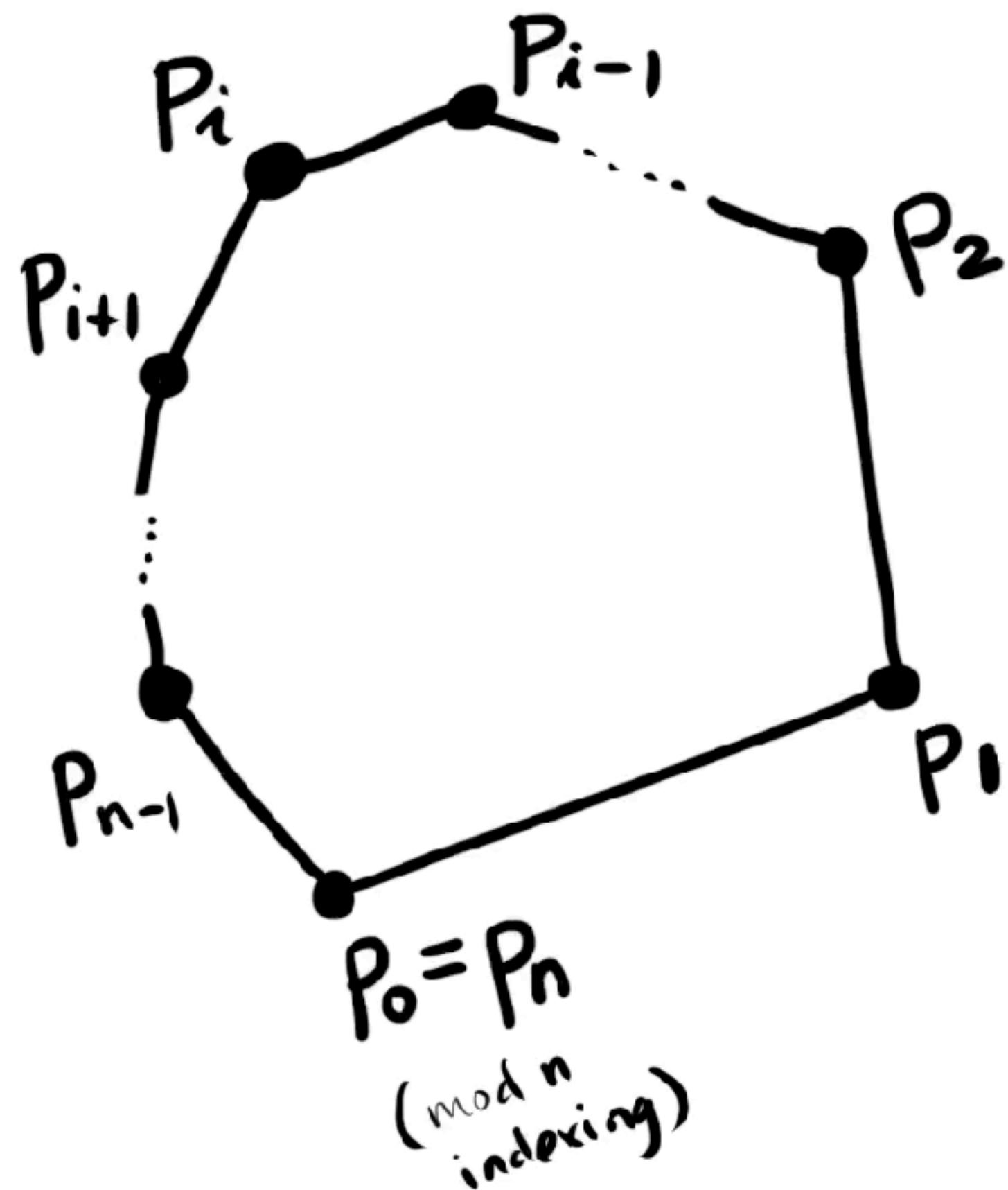
$$= \vec{P}_1 \wedge \vec{P}_2 - \vec{P}_1 \wedge \vec{P}_0 - \vec{P}_0 \wedge \vec{P}_2 + \vec{P}_0 \wedge \vec{P}_1$$

$$= \vec{P}_0 \wedge \vec{P}_1 + \vec{P}_1 \wedge \vec{P}_2 + \vec{P}_2 \wedge \vec{P}_0$$



# Area of a 2D polygon

Generalizing to a 2D polygon:



$$\begin{aligned} 2A &= \sum_{i=0}^{n-1} P_i \wedge P_{i+1} \\ &= \dots + P_{i-1} \wedge P_i + P_i \wedge P_{i+1} + \dots \\ &= \dots + P_{i-1} \wedge P_i - P_{i+1} \wedge P_i + \dots \\ &= \dots + (P_{i-1} - P_{i+1}) \wedge P_i + \dots \\ \Rightarrow \partial_i A &=? \end{aligned}$$

# Gradient of area of a 2D polygon

$$\partial_i A = \partial_i \left\{ \frac{1}{2} (\underline{P_{i-1} - P_{i+1}}) \wedge P_i \right\}$$

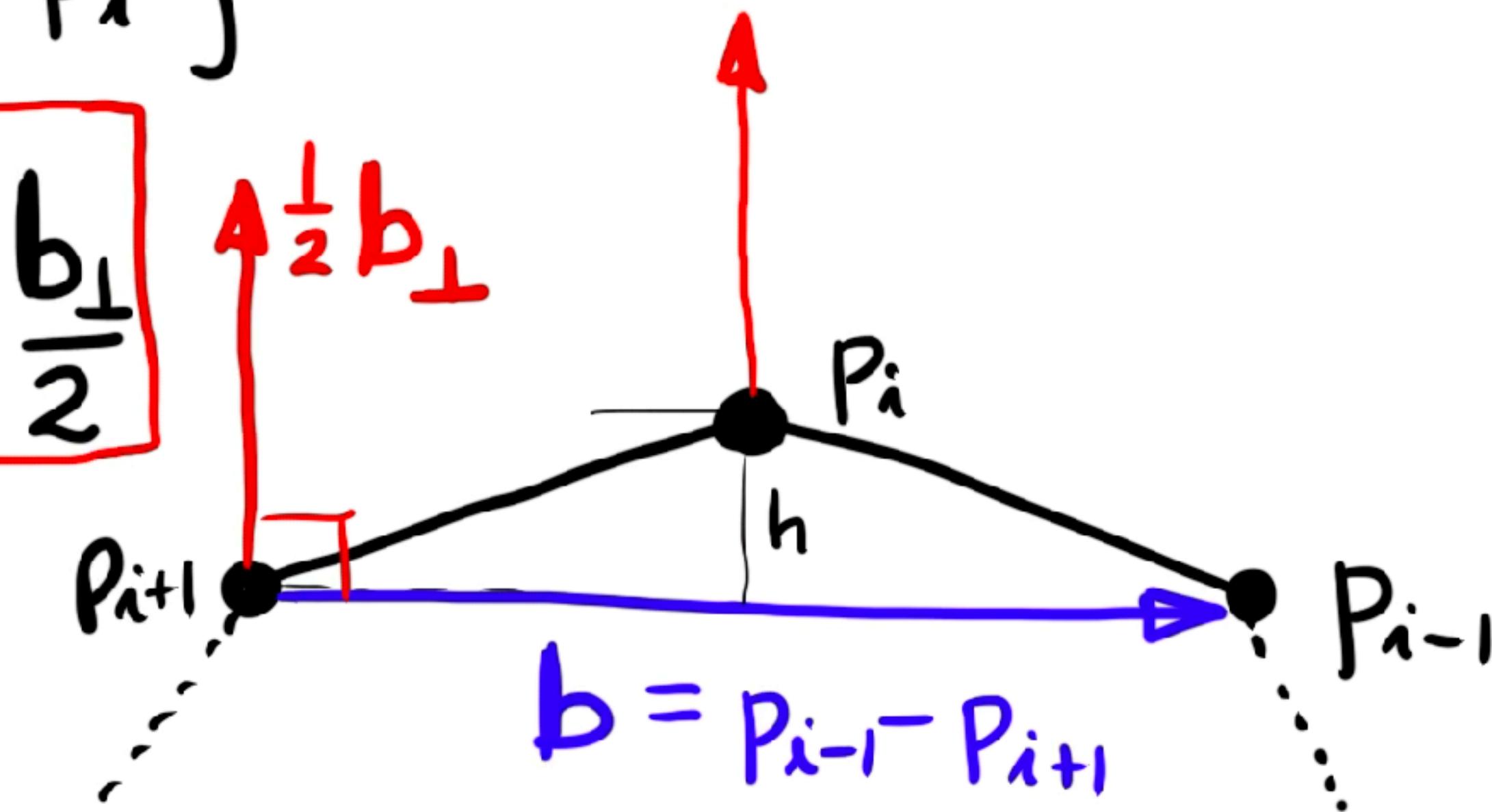
recall that  $u \wedge v = u^\top v = \begin{pmatrix} -u_y \\ u_x \end{pmatrix} \cdot \begin{pmatrix} v_x \\ v_y \end{pmatrix}$

$$= \partial_i \left\{ \frac{1}{2} (\underline{P_{i-1} - P_{i+1}})_\perp^\top P_i \right\}$$

$$\boxed{\partial_i A = \frac{1}{2} (\underline{P_{i-1} - P_{i+1}})_\perp = \frac{b_\perp}{2}}$$

Recall

$$A_\Delta = \frac{1}{2} b h \rightarrow \frac{\partial A_\Delta}{\partial h} = \frac{b}{2}$$



# Controlling Compression in 2D: Area springs

REST SHAPE  
(UNDEFORMED)



DEFORMED



POTENTIAL  
ENERGY:

$$E = \frac{1}{2} k (A_{(P)} - A_0)^2$$

ELASTIC FORCE:

$$f_i = -k (A - A_0) \underline{\partial_i A_{(P)}}$$

# Bending forces for cloth animation

SIGGRAPH 98, Orlando, July 19–24

COMPUTER GRAPHICS Proceedings, Annual Conference Series, 1998

SIGGRAPH 98, Orlando, July 19–24

COMPUTER GRAPHICS Proceedings, Annual Conference Series, 1998

## Large Steps in Cloth Simulation

David Baraff Andrew Witkin

Robotics Institute  
Carnegie Mellon University

### Abstract

The bottle-neck in most cloth simulation systems is that time steps must be small to avoid numerical instability. This paper describes a cloth simulation system that can stably take large time steps. The simulation system couples a new technique for enforcing constraints on individual cloth particles with an implicit integration method. The simulator models cloth as a triangular mesh, with internal cloth forces derived using a simple continuum formulation that supports modeling operations such as local anisotropic stretch or compression; a unified treatment of damping forces is included as well. The implicit integration method generates a large, unbandaged sparse linear system at each time step which is solved using a modified conjugate gradient method that simultaneously enforces particles' constraints. The constraints are always maintained exactly, independent of the number of conjugate gradient iterations, which is typically small. The resulting simulation system is significantly faster than previous accounts of cloth simulation systems in the literature.

**Keywords**—Cloth, simulation, constraints, implicit integration, physically-based modeling.

### 1 Introduction

Physically-based cloth animation has been a problem of interest to the graphics community for more than a decade. Early work by Terzopoulos *et al.* [17] and Terzopoulos and Fleischer [15, 16] on deformable models correctly characterized cloth simulation as a problem in deformable surfaces, and applied techniques from the mechanical engineering and finite element communities to the problem. Since then, other research groups (notably Carignan *et al.* [4] and Volino *et al.* [20, 21]; Breen *et al.* [3]; and Eberhardt *et al.* [5]) have taken up the challenge of cloth.

Although specific details vary (underlying representations, numerical solution methods, collision detection and constraint methods, etc.), there is a deep commonality amongst all the approaches: physically-based cloth simulation is formulated as a time-varying partial differential equation which, after discretization, is numerically solved as an ordinary differential equation

$$\ddot{\mathbf{x}} = \mathbf{M}^{-1} \left( -\frac{\partial E}{\partial \mathbf{x}} + \mathbf{F} \right). \quad (1)$$

In this equation the vector  $\mathbf{x}$  and diagonal matrix  $\mathbf{M}$  represent the geometric state and mass distribution of the cloth,  $E$ —a scalar function

Author affiliation (September 1998): David Baraff, Andrew Witkin, Pixar Animation Studios, 1001 West Cutting Blvd., Richmond, CA 94804. Email: deb@pixar.com, aw@pixar.com.

This is an electronic reprint. Permission is granted to copy part or all of this paper for noncommercial use provided that the title and this copyright notice appear. This electronic reprint is ©1998 by CMU. The original printed paper is ©1998 by the ACM.

of  $\mathbf{x}$ —yields the cloth's internal energy, and  $\mathbf{F}$  (a function of  $\mathbf{x}$  and  $\dot{\mathbf{x}}$ ) describes other forces (air-drag, contact and constraint forces, internal damping, etc.) acting on the cloth.

In this paper, we describe a cloth simulation system that is much faster than previously reported simulation systems. Our system's faster performance begins with the choice of an *implicit* numerical integration method to solve equation (1). The reader should note that the use of implicit integration methods in cloth simulation is far from novel: initial work by Terzopoulos *et al.* [15, 16, 17] applied such methods to the problem.<sup>1</sup> Since this time though, research on cloth simulation has generally relied on *explicit* numerical integration (such as Euler's method or Runge-Kutta methods) to advance the simulation, or, in the case of energy minimization, analogous methods such as steepest-descent [3, 10].

This is unfortunate. Cloth strongly resists stretching motions while being comparatively permissive in allowing bending or shearing motions. This results in a "stiff" underlying differential equation of motion [12]. Explicit methods are ill-suited to solving stiff equations because they require many small steps to stably advance the simulation forward in time.<sup>2</sup> In practice, the computational cost of an explicit method greatly limits the realizable resolution of the cloth. For some applications, the required spatial resolution—that is, the dimension  $n$  of the state vector  $\mathbf{x}$ —can be quite low: a resolution of only a few hundred particles (or nodal points, depending on your formulation/terminology) can be sufficient when it comes to modeling flags or tablecloths. To animate clothing, which is our main concern, requires much higher spatial resolution to adequately represent realistic (or even semi-realistic) wrinkling and folding configurations.

In this paper, we demonstrate that implicit methods for cloth overcome the performance limits inherent in explicit simulation methods. We describe a simulation system that uses a triangular mesh for cloth surfaces, eliminating topological restrictions of rectangular meshes, and a simple but versatile formulation of the internal cloth energy forces. (Unlike previous metric-tensor-based formulations [15, 16, 17, 4] which model some deformation energies as quadratic functions of positions, we model deformation energies only as quadratic functions with suitably large scaling. Quadratic energy models mesh well with implicit integration's numerical properties.) We also introduce a simple, unified treatment of damping forces, a subject which has been largely ignored thus far. A key step in our simulation process is the solution of an  $O(n) \times O(n)$  sparse linear system, which arises from the implicit integration method. In this respect, our implementation differs greatly from the implementation by Terzopoulos *et al.* [15, 17], which for large simulations

SIGGRAPH 98, Orlando, July 19–24

COMPUTER GRAPHICS Proceedings, Annual Conference Series, 1998

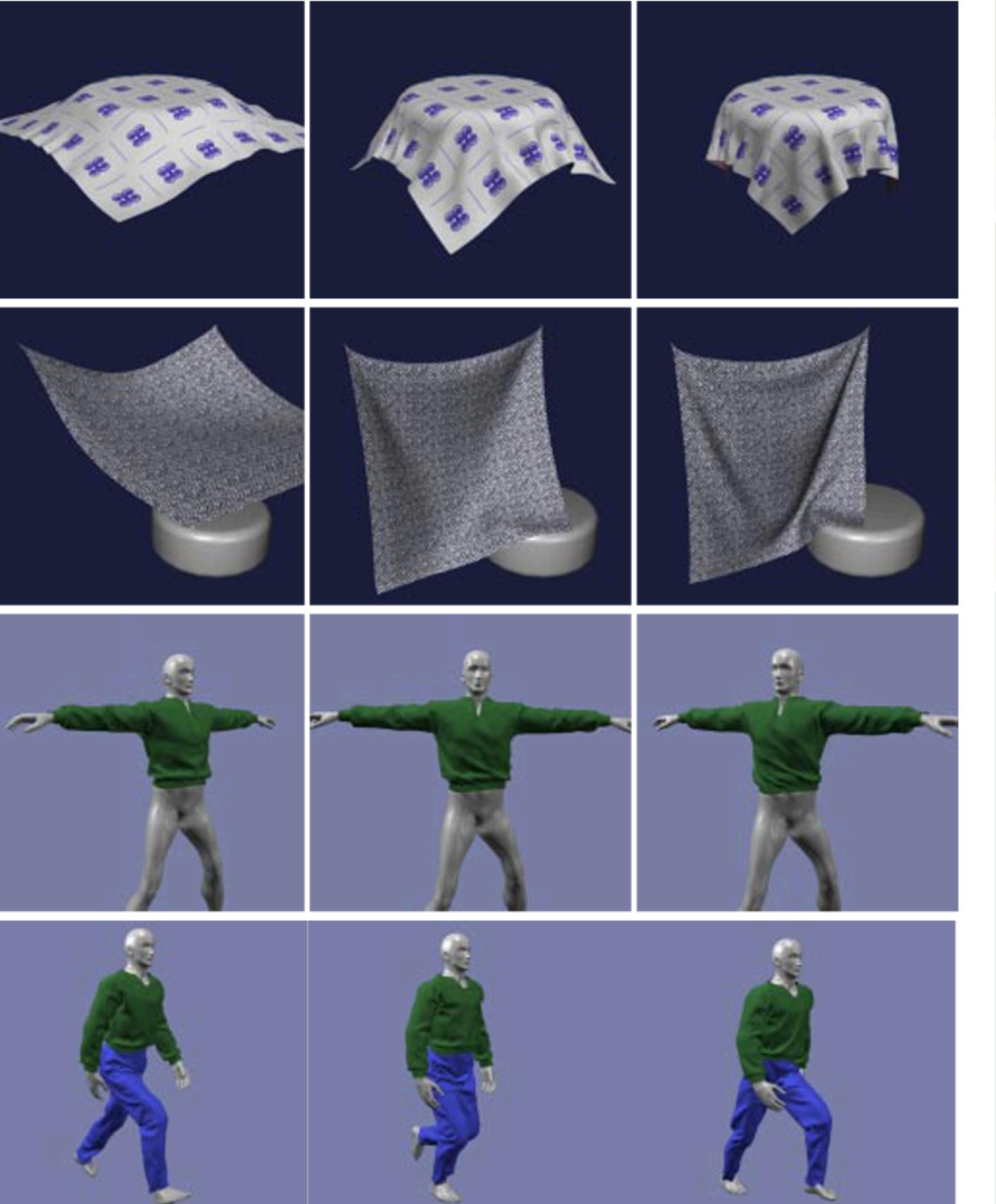


Figure 1 (top row): Cloth draping on cylinder; frames 8, 13 and 35. Figure 2 (second row): Sheet with two fixed particles; frames 10, 29 and 67. Figure 3 (third row): Shirt on twisting figure; frames 1, 24 and 46. Figure 4 (bottom row): Walking man; frames 30, 45 and 58.

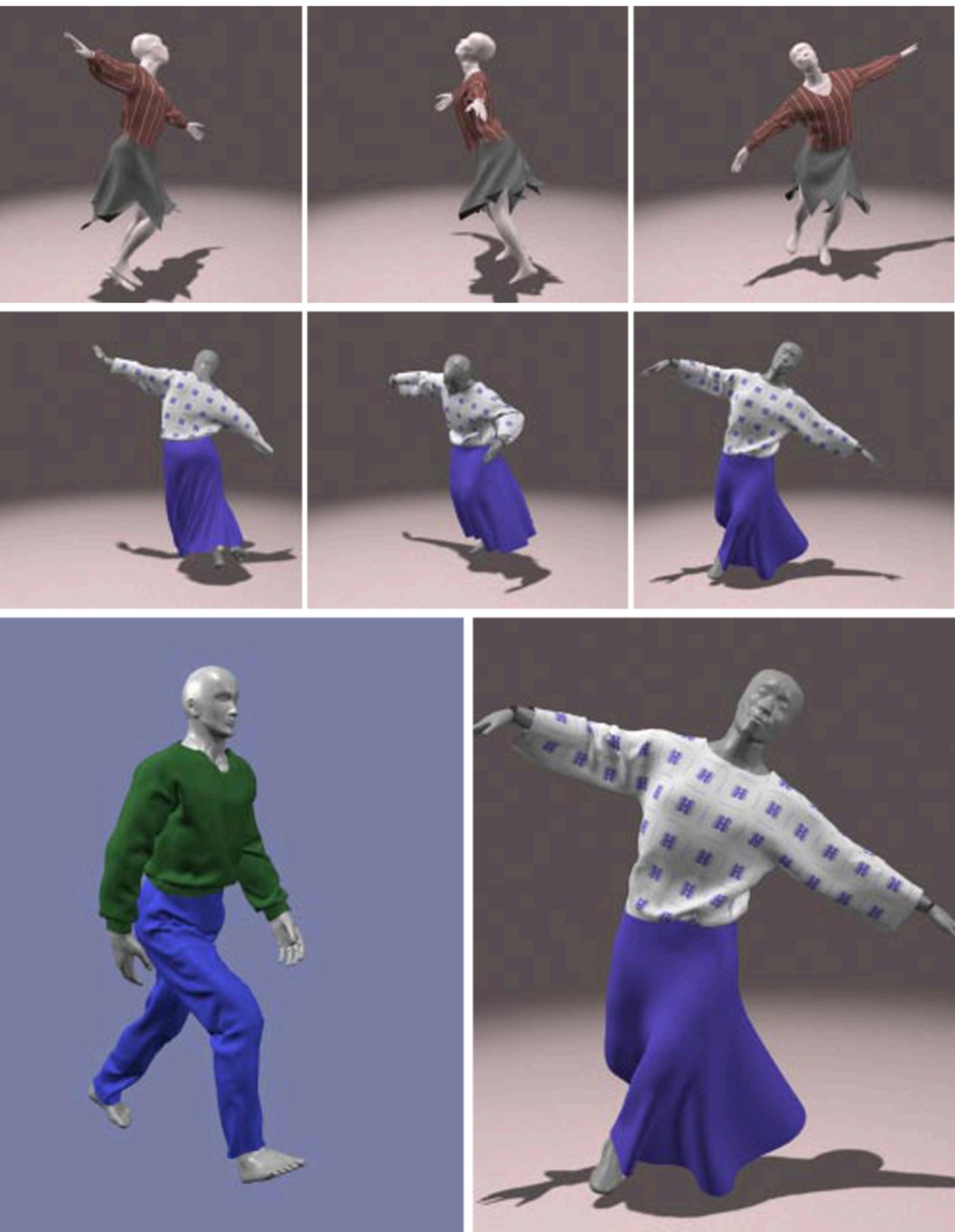
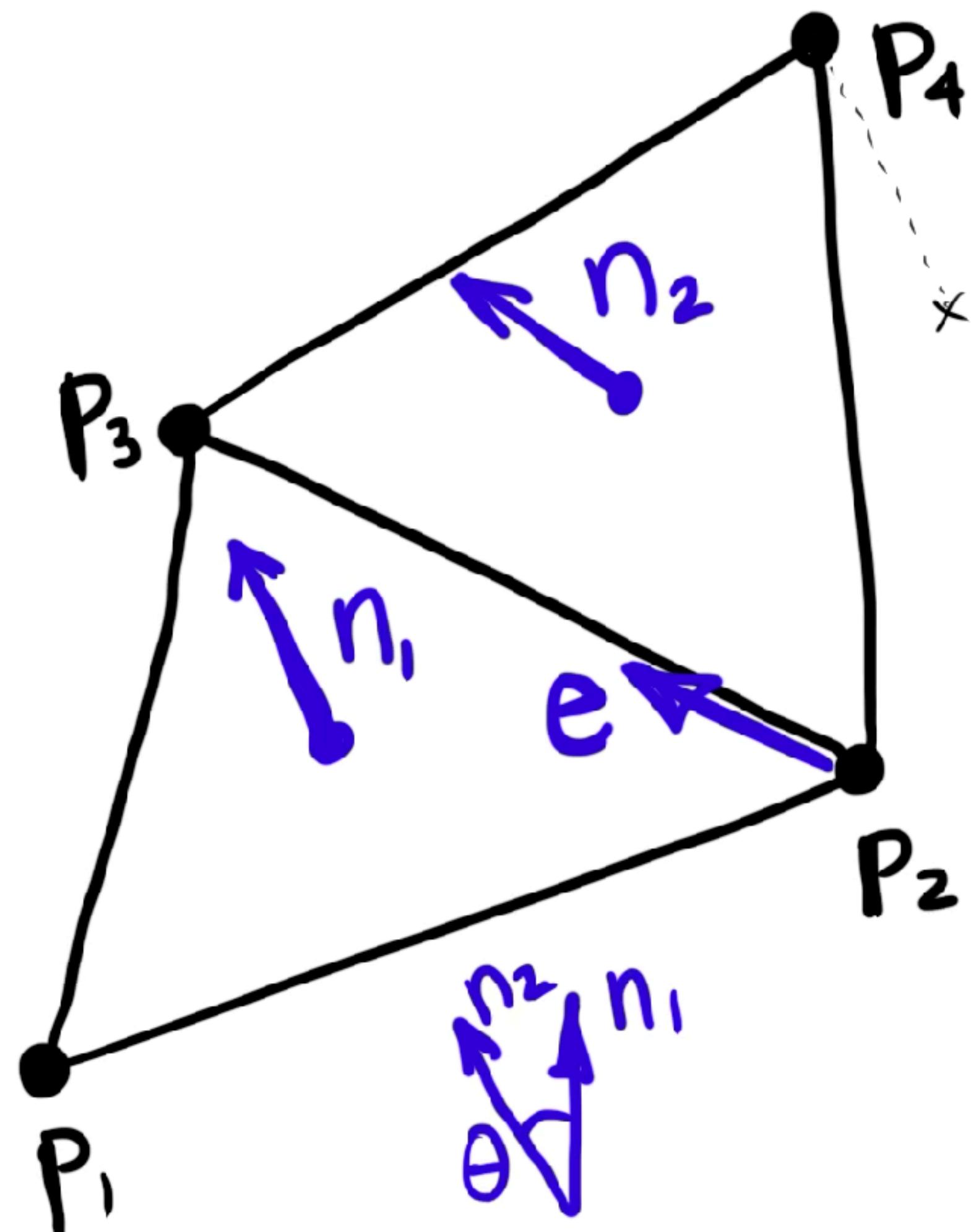


Figure 5 (top row): Dancer with short skirt; frames 110, 136 and 155. Figure 6 (middle row): Dancer with long skirt; frames 185, 215 and 236. Figure 7 (bottom row): Closeups from figures 4 and 6.

# Bending force models for cloth and shells



Let  $n_1 = \frac{(P_2 - P_1) \times (P_3 - P_1)}{\|(P_2 - P_1) \times (P_3 - P_1)\|}$

$$n_2 = \frac{(P_1 - P_2) \times (P_3 - P_2)}{\|(P_1 - P_2) \times (P_3 - P_2)\|}$$

$$e = (P_3 - P_2) / \|P_3 - P_2\|$$

Then the bending angle  $\Theta$  is s.t.

$$\sin \Theta = (n_1 \times n_2) \cdot e, \quad \cos \Theta = n_1 \cdot n_2$$

$$\Rightarrow E(P) = \frac{1}{2} k C^2(P) \text{ where } C(P) = \Theta(P) - \Theta_0$$

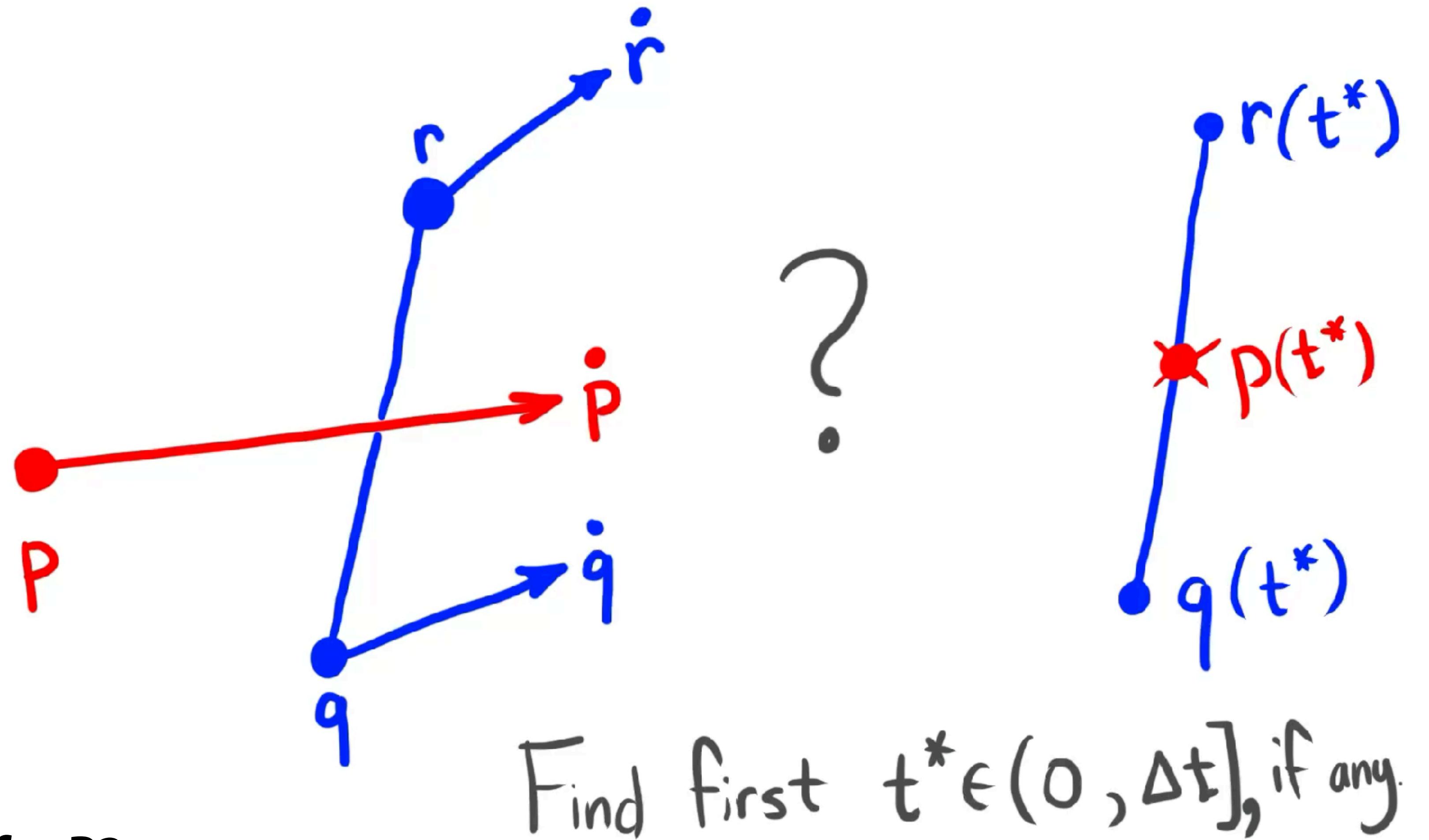
↑ rest angle

# **Deformable collision detection**

**Time of collision (ToC)**  
**Root-finding methods**

RECALL:

# CCD Test: Point-Edge (2D)



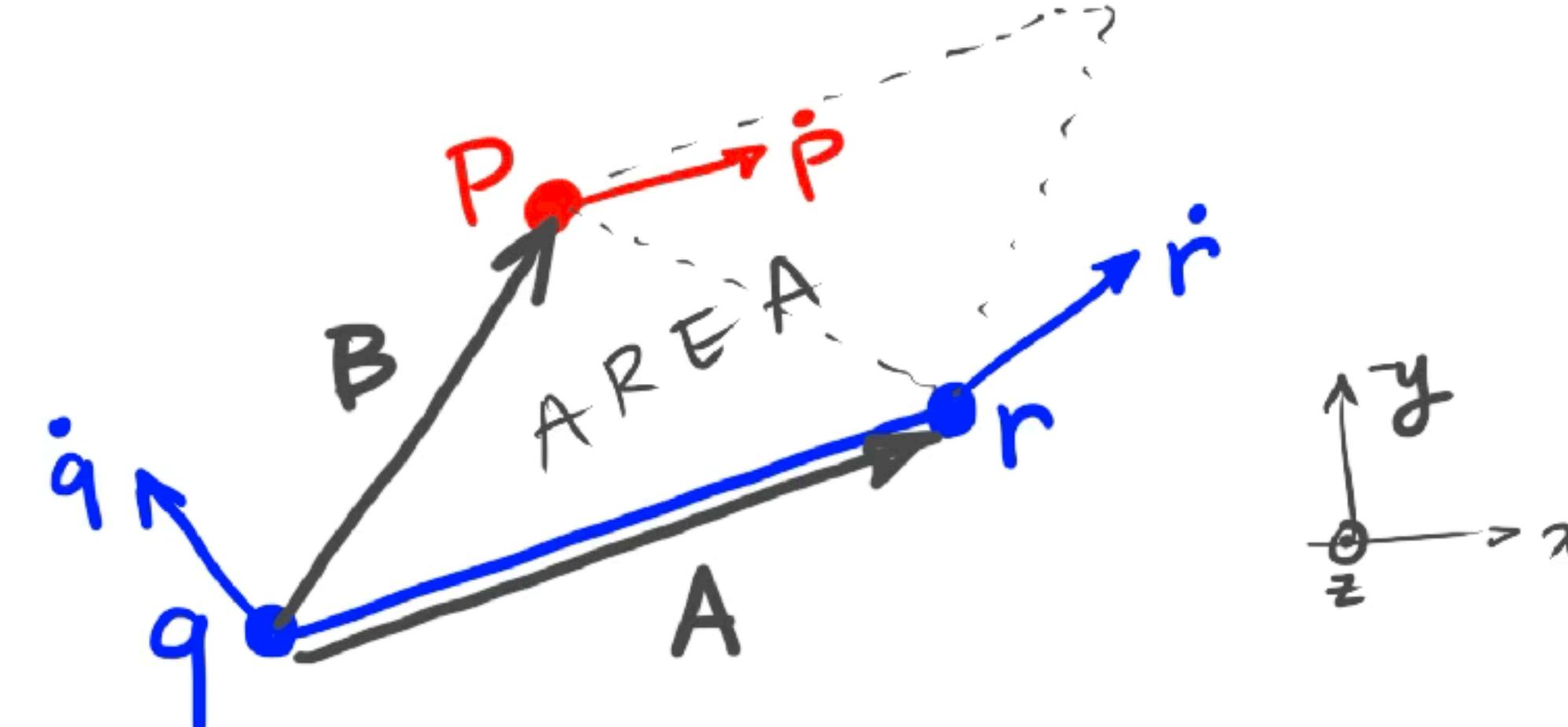
## ■ Approach for P2:

- Find time when area of pt-edge triangle is zero, then check if pt on edge.

**RECALL:**

CCD Test: Point-Edge (2D)

# Collinearity test



$$A(t) \equiv (r + \dot{r}t) - (q + \dot{q}t) = (r - q) + (\dot{r} - \dot{q})t \equiv a + \dot{a}t$$

$$B(t) \equiv (p + \dot{p}t) - (q + \dot{q}t) = (p - q) + (\dot{p} - \dot{q})t \equiv b + \dot{b}t$$

$$\text{"AREA}(t)\text{"} = (A(t) \times B(t))_z \equiv A_x B_y - A_y B_x \equiv A \wedge B$$

$$= (a + \dot{a}t) \wedge (b + \dot{b}t)$$

$$= (\dot{a} \wedge \dot{b}) t^2 + (\dot{a} \wedge b + a \wedge \dot{b}) t + (a \wedge b)$$

Find roots of  $\text{AREA}(t^*) = 0$ .

Find first TOC s.t.  $t^* \in (0, \Delta t]$  AND  $p(t^*)$  is on  $\overline{q(t^*) r(t^*)}$ .

# Robust root finding: Quadratic equation

Find real roots of  $0 = at^2 + bt + c$

Can assume that  $a \neq 0$  &  $b \neq 0$  (Why?)

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



Poorly behaved numerically  
(e.g.,  $|a| \rightarrow 0$ )

Robust solution:

(ref. Numerical Recipes in C)

Compute determinant,  $D = b^2 - 4ac$ .

if ( $D > 0$ )

$$r = -\frac{1}{2} (b + \text{sgn}(b)\sqrt{D})$$

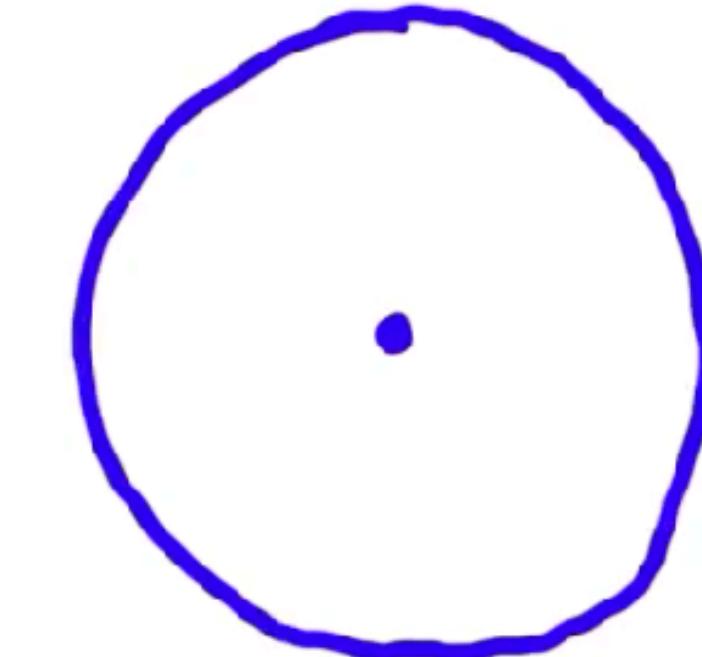
$$t_1 = r/a$$

$$t_2 = c/r$$

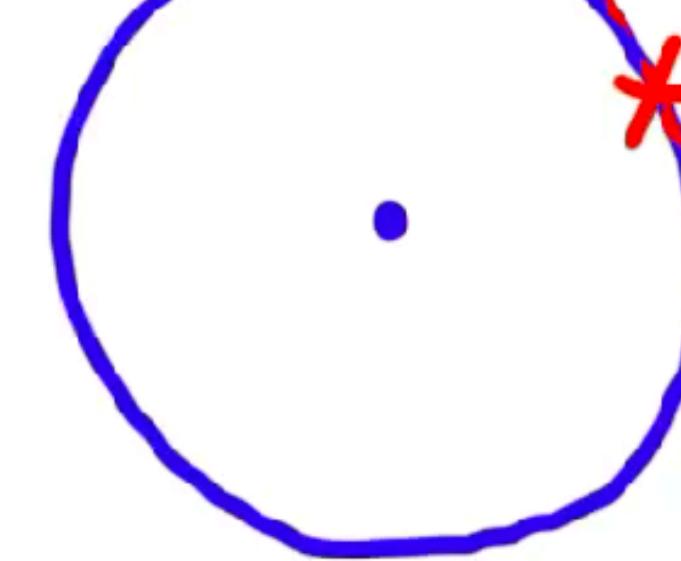
} Pt-Edge CCD: Find first root  
on  $(0, \Delta t]$  that is also  
on the line.

# Geometric interpretation of number of real roots

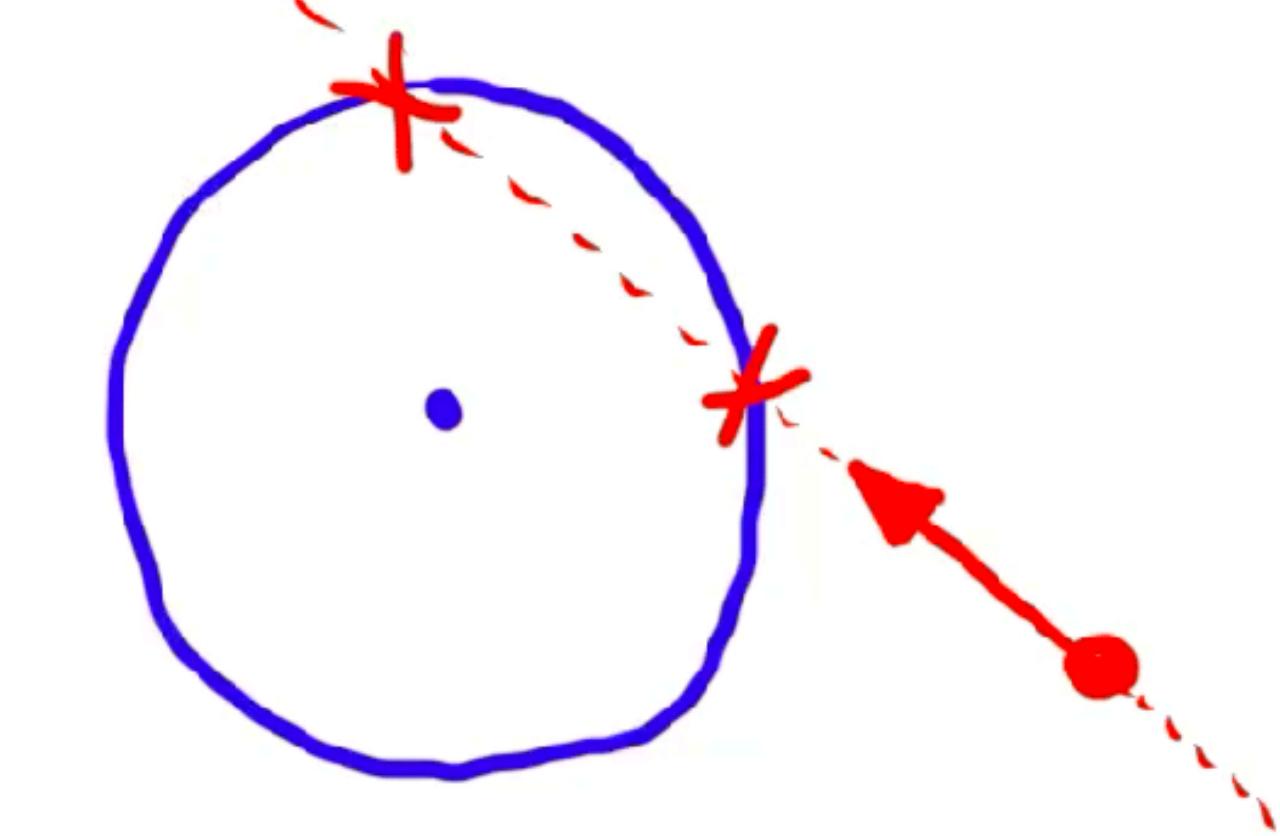
Point-sphere (sphere-sphere) CCD cases



0 roots



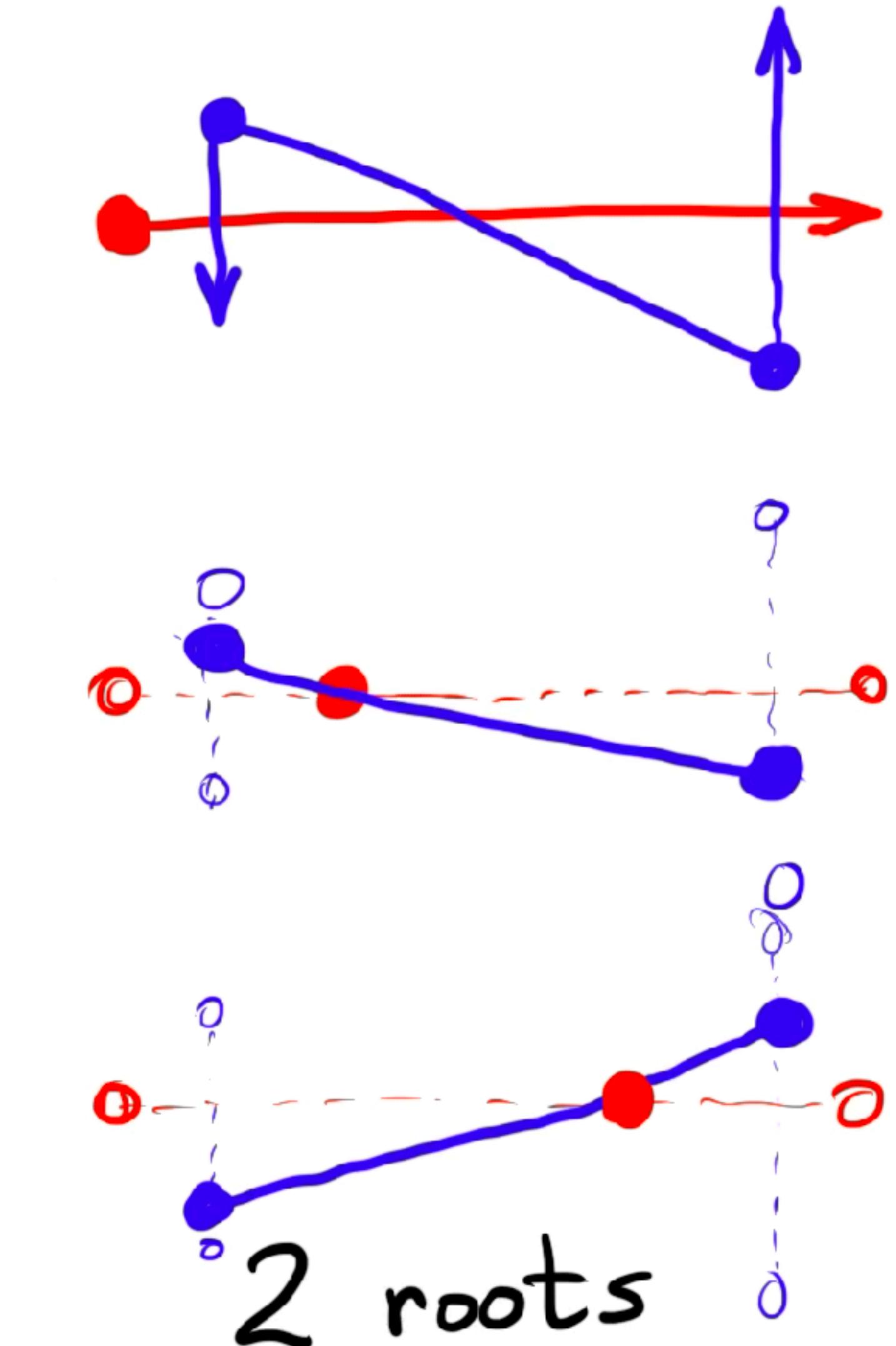
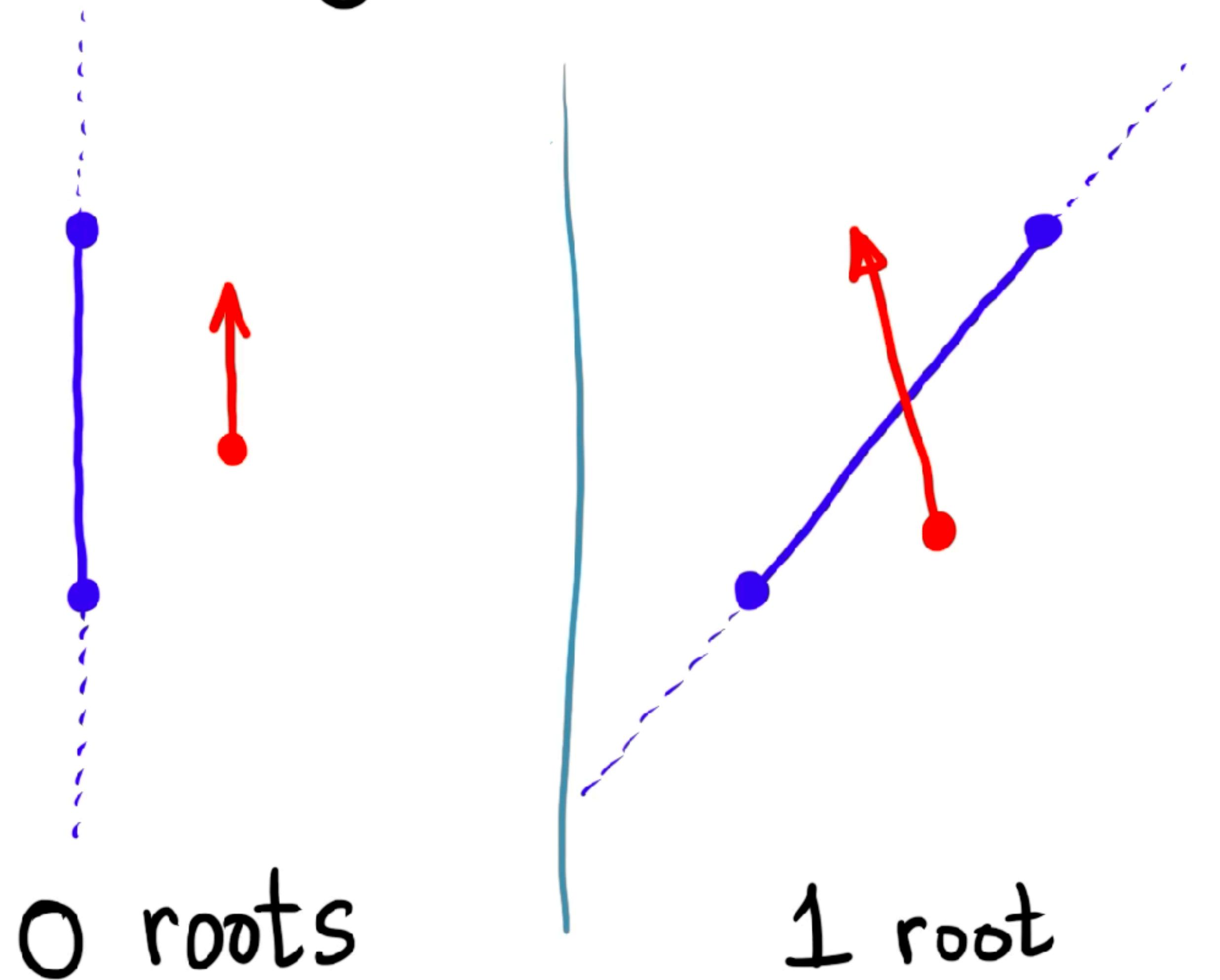
1 root



2 roots

# Geometric interpretation of number of real roots

Point - edge CCD cases



# 3D CCD: Point-triangle and edge-edge tests

IMPORTANT CCD TESTS  
FOR CLOTH & HAIR

Linear motion

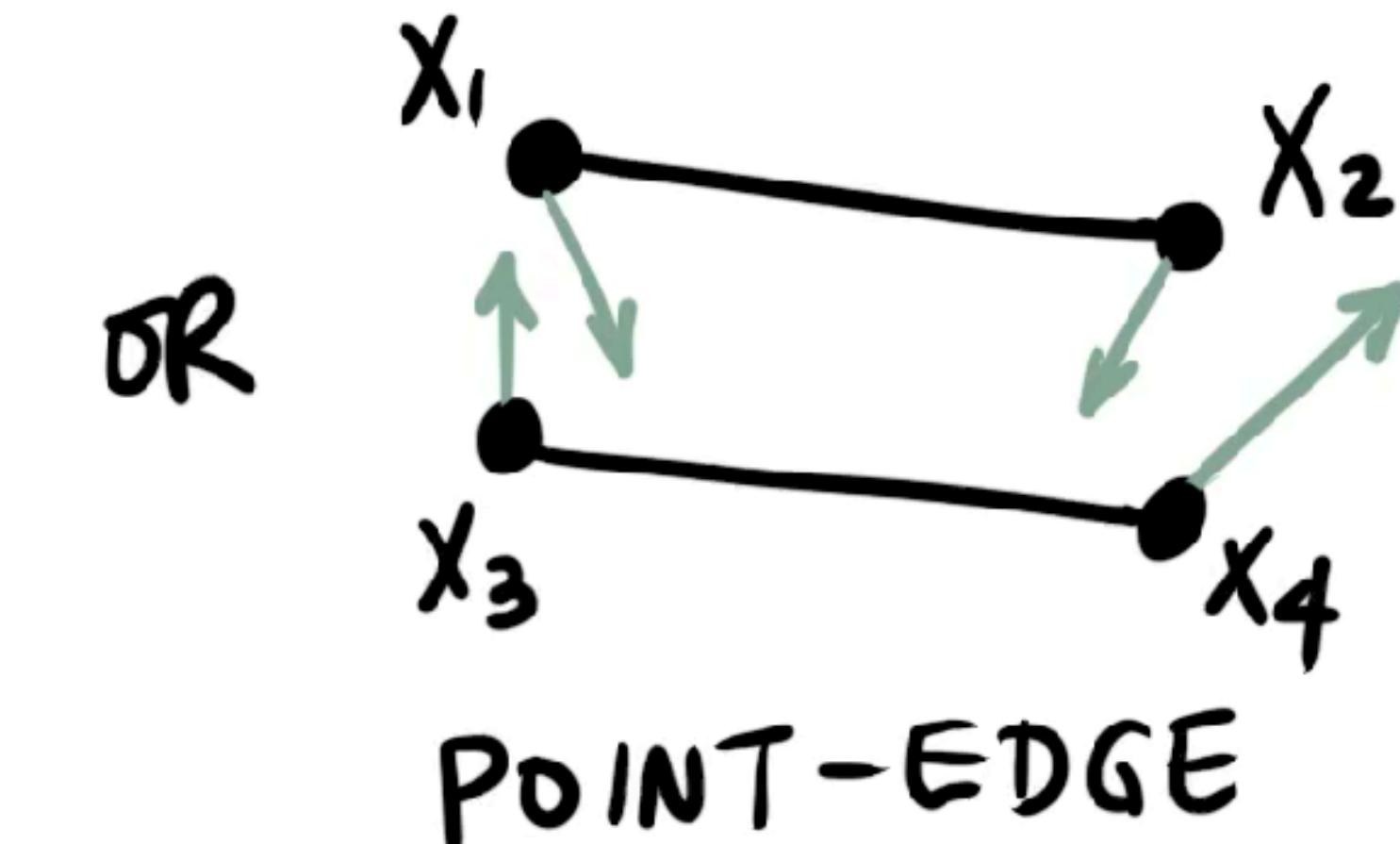
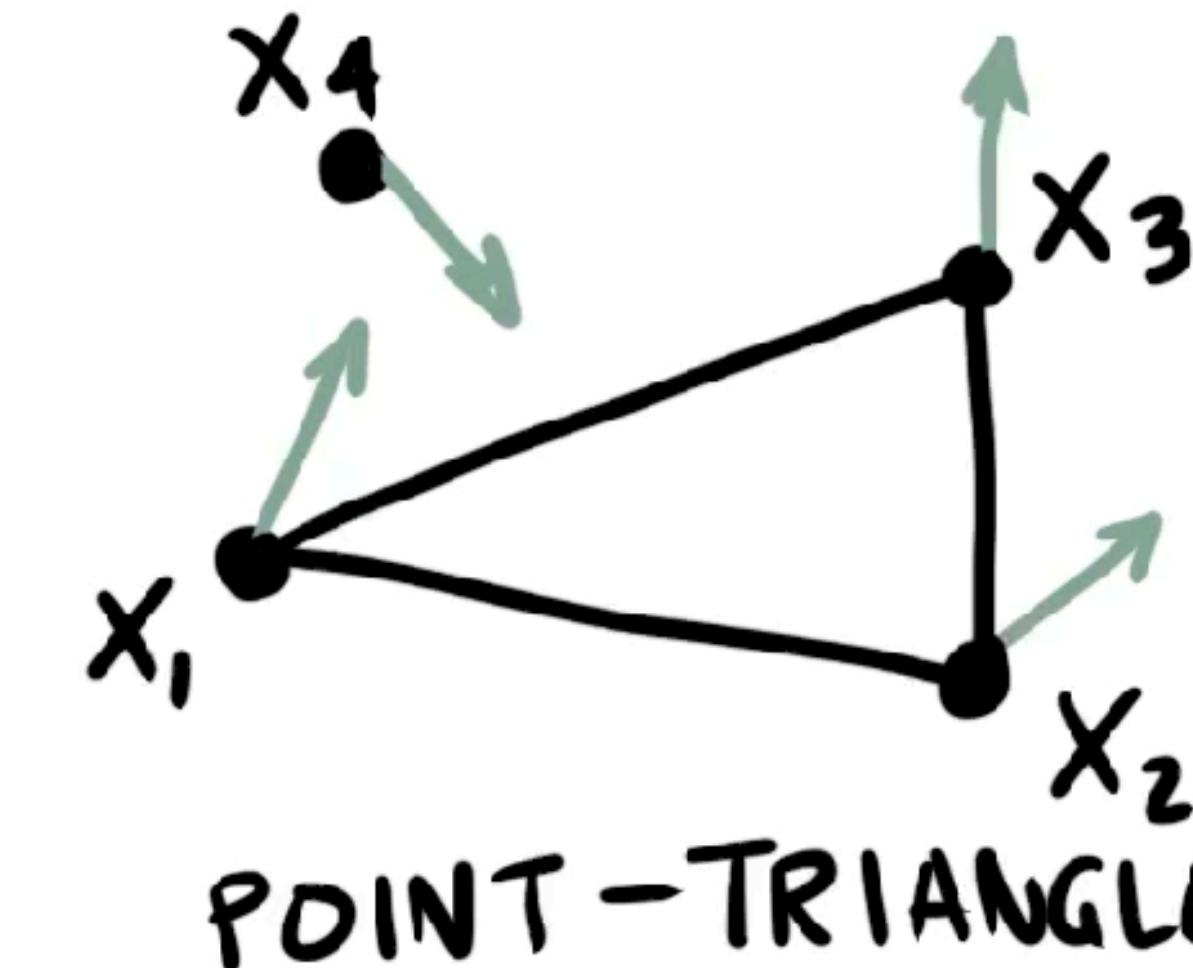
$$x_i + t v_i$$

Let

$$x_{ij} \equiv x_i - x_j$$

$$v_{ij} \equiv v_i - v_j$$

COPLANARITY TEST  $\rightarrow$



OR

$$(x_{21} + t v_{21}) \times (x_{31} + t v_{21}) \cdot (x_{41} + t v_{41}) = 0$$

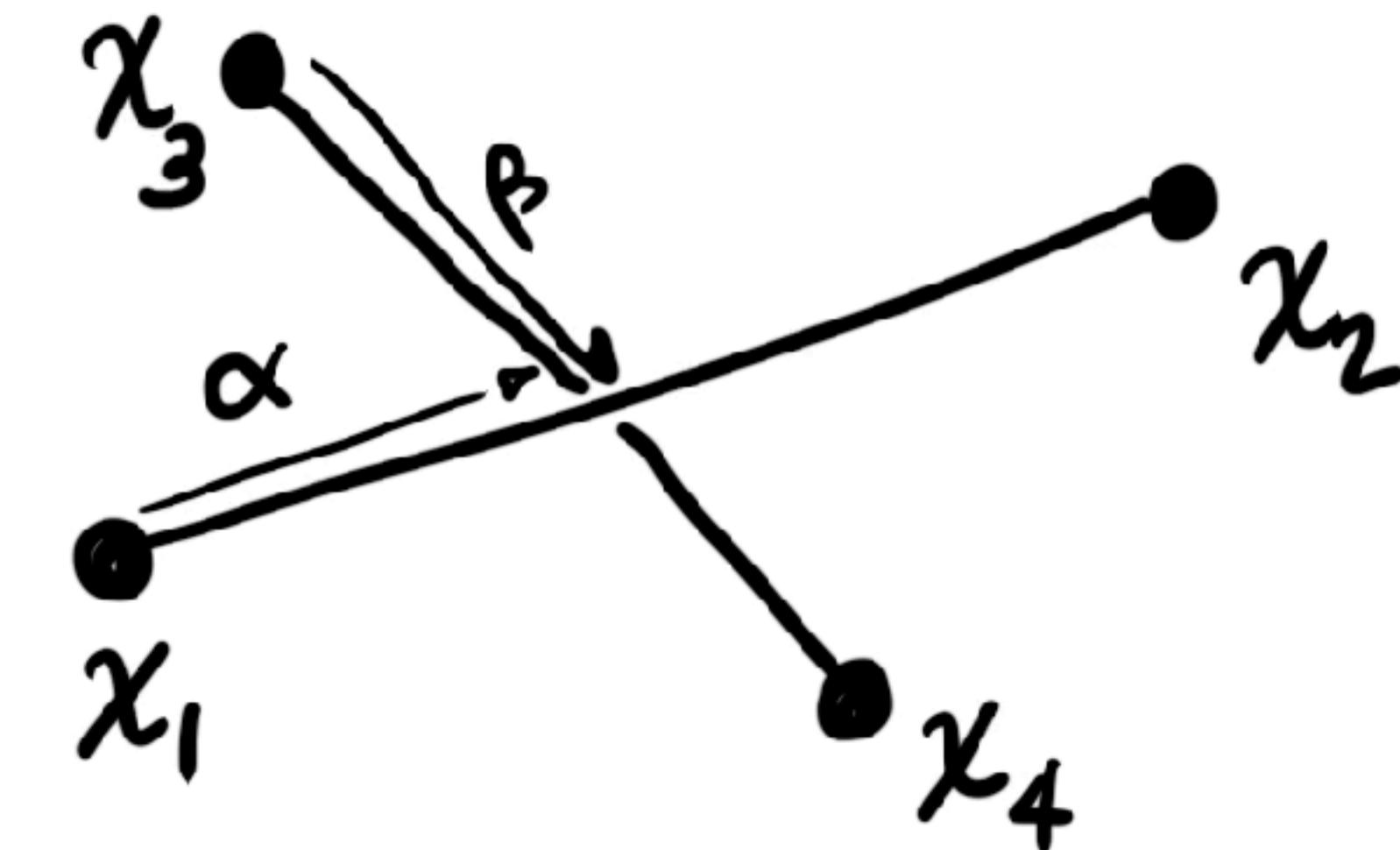
- $\Rightarrow$
- Cubic equation in  $t$
  - Find first ToC  $t^* \in (0, \Delta t]$   
s.t. objects touch.
  - Watch rounding & numerical errors!

# 3D CCD: Edge-edge test: Check if touching at ToC

"ON EDGES" TEST @ ToC

Let geometry at ToC be

$$x_1 + \alpha x_{21} = x_3 + \beta x_{43}$$



$$\begin{bmatrix} 1 & 1 \\ x_{21} & x_{34} \end{bmatrix} \begin{pmatrix} \alpha \\ \beta \end{pmatrix} = x_{31} \quad (3 \text{ eqns with 2 unknowns})$$

Solve in a least-squares sense:  $Ax = b \rightarrow A^T A x = A^T b$ :

$$\begin{bmatrix} (x_{21} \cdot x_{21}) & (x_{21} \cdot x_{34}) \\ (x_{34} \cdot x_{21}) & (x_{34} \cdot x_{34}) \end{bmatrix} \begin{pmatrix} \alpha \\ \beta \end{pmatrix} = \begin{pmatrix} x_{21} \cdot x_{31} \\ x_{34} \cdot x_{31} \end{pmatrix} \implies \alpha^*, \beta^* \in [0, 1] ?$$

**RECALL:**

# Time-of-collision (ToC): Robust root-finding methods

- Must find the roots of polynomial equations to find the ToC
- Can be numerically tricky for arbitrary computer-generated polynomials from animation
- Cloth animation can generate billions of CCD tests.
  - Many cubic polynomials with ill-conditioned coefficients
- One-in-a-billion bugs are a show stopper.
- Codes must be fast AND highly reliable.
  - Analytical methods are not useful in practice.



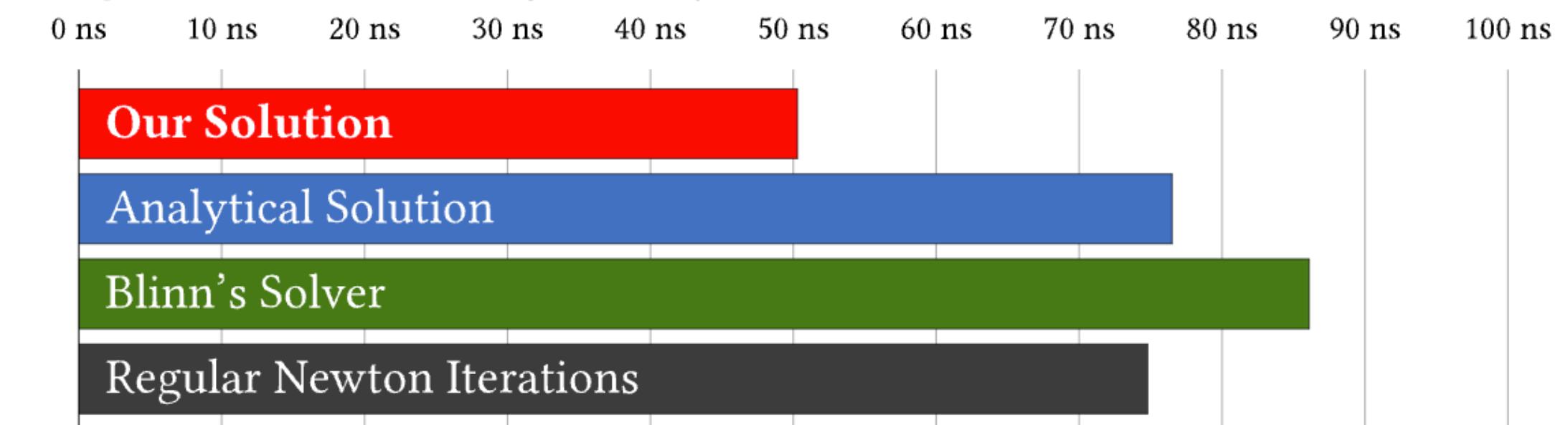
Graph of a cubic function with 3 **real roots** (where the curve crosses the horizontal axis at  $y = 0$ ). The case shown has two **critical points**. Here the function is  $f(x) = (x^3 + 3x^2 - 6x - 8)/4$ .

# Robust root finding: Polynomial equations

- See this recent and clear talk by Cem Yuksel



Computation Time of Cubic (Degree 3) Polynomials



<http://www.cemyuksel.com/research/polynomials>

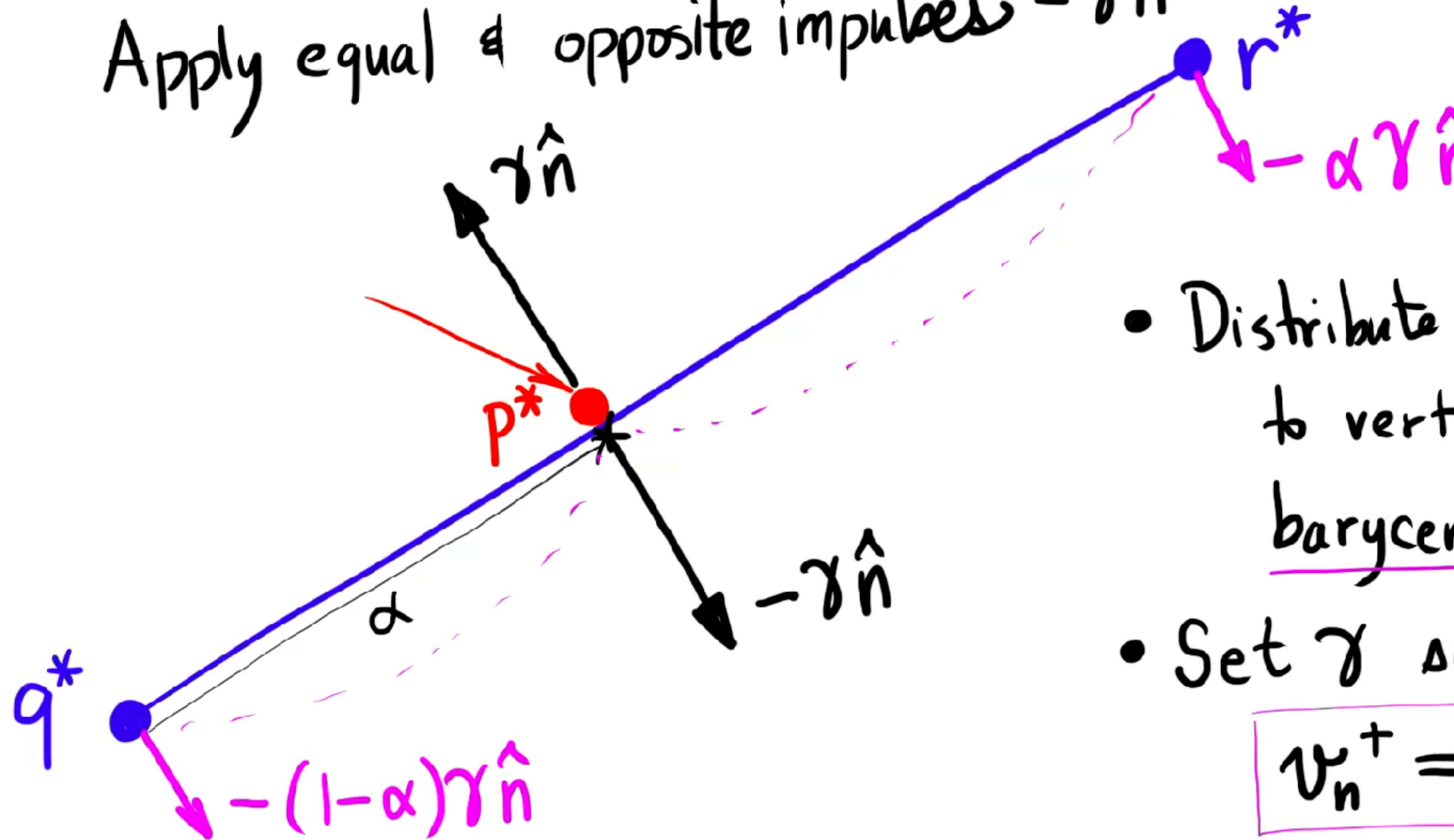
# **Deformable collision resolution**

**Computing impulses and penalty forces**

# 2D point-edge collision: Impulse calculation

At ToC:

Apply equal & opposite impulses  $\pm \gamma \hat{n}$



- Distribute impulse on edge to vertices using barycentric weighting.

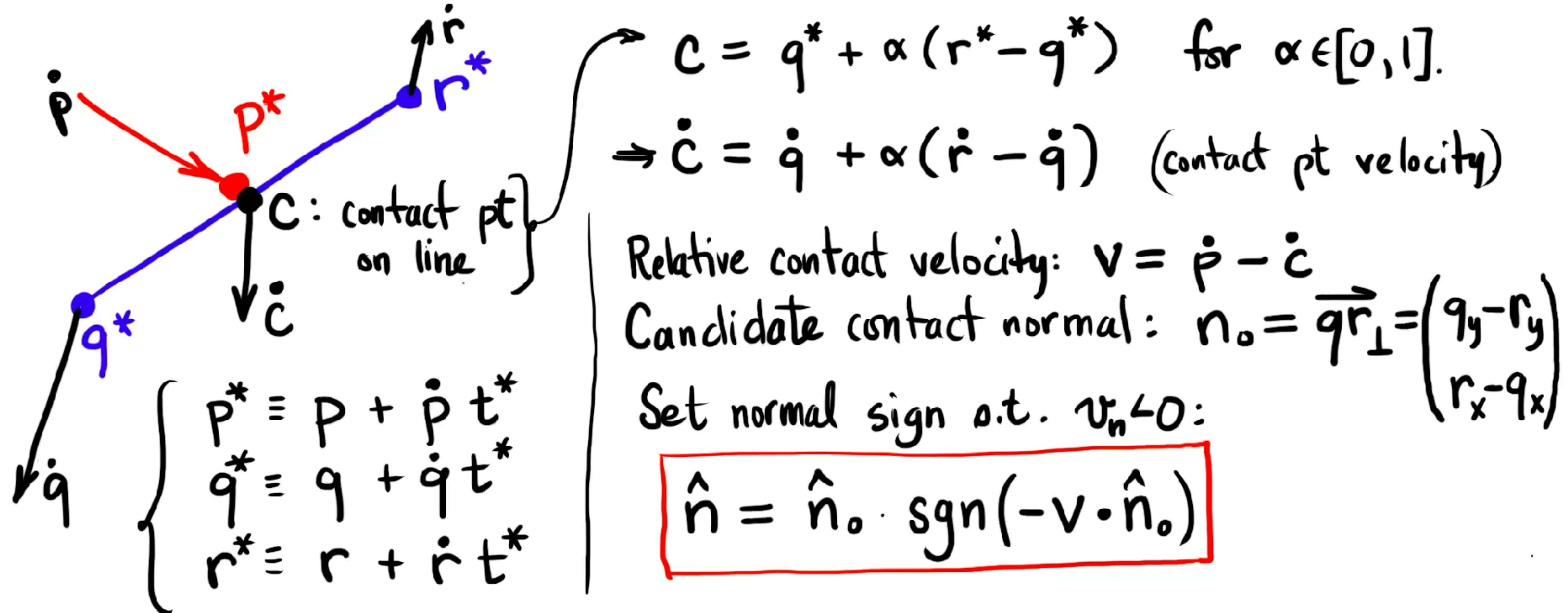
- Set  $\gamma$  so that

$$v_n^+ = -\epsilon v_n^-$$

# 2D point-edge collision: Normal calculation

Need contact normal at ToC s.t.  $v_n < 0$ . How?

At ToC we have ( $t = t^* \in (0, \Delta t]$ )



# 2D point-edge collision: Impulse calculation

Post-impulse particle velocities

$$\dot{\vec{p}}^+ = \dot{\vec{p}}^- + \Delta\dot{\vec{p}} = \dot{\vec{p}}^- + \frac{\gamma}{m_p} \hat{n}$$

$$\dot{\vec{q}}^+ = \dot{\vec{q}}^- + \Delta\dot{\vec{q}} = \dot{\vec{q}}^- - (1-\alpha) \frac{\gamma}{m_q} \hat{n}$$

$$\dot{\vec{r}}^+ = \dot{\vec{r}}^- + \Delta\dot{\vec{r}} = \dot{\vec{r}}^- - \alpha \frac{\gamma}{m_r} \hat{n}$$

$$\begin{aligned}\dot{\vec{c}}^+ &= \dot{\vec{c}}^- + \Delta\dot{\vec{c}} = (1-\alpha) \dot{\vec{q}}^+ + \alpha \dot{\vec{r}}^+ \\ &= (\bar{\alpha} \dot{\vec{q}}^- + \alpha \dot{\vec{r}}^-) + (\bar{\alpha} \Delta\dot{\vec{q}} + \alpha \Delta\dot{\vec{r}})\end{aligned}$$

$\bar{\alpha} \equiv 1 - \alpha$   
(I'm that lazy!)

# 2D point-edge collision: Impulse calculation

SOLVE FOR  $\gamma$  USING RESTITUTION HYPOTHESIS:

$$v_n^+ = -\varepsilon v_n^-$$

$$(\dot{p}^+ - \dot{c}^+) \cdot \hat{n} = -\varepsilon (\dot{p}^- - \dot{c}^-) \cdot \hat{n}$$

$$(\dot{p}_n^- - \dot{c}_n^-) + (\Delta \dot{p}_n - \Delta \dot{c}_n) = -\varepsilon (\dot{p}_n^- - \dot{c}_n^-)$$

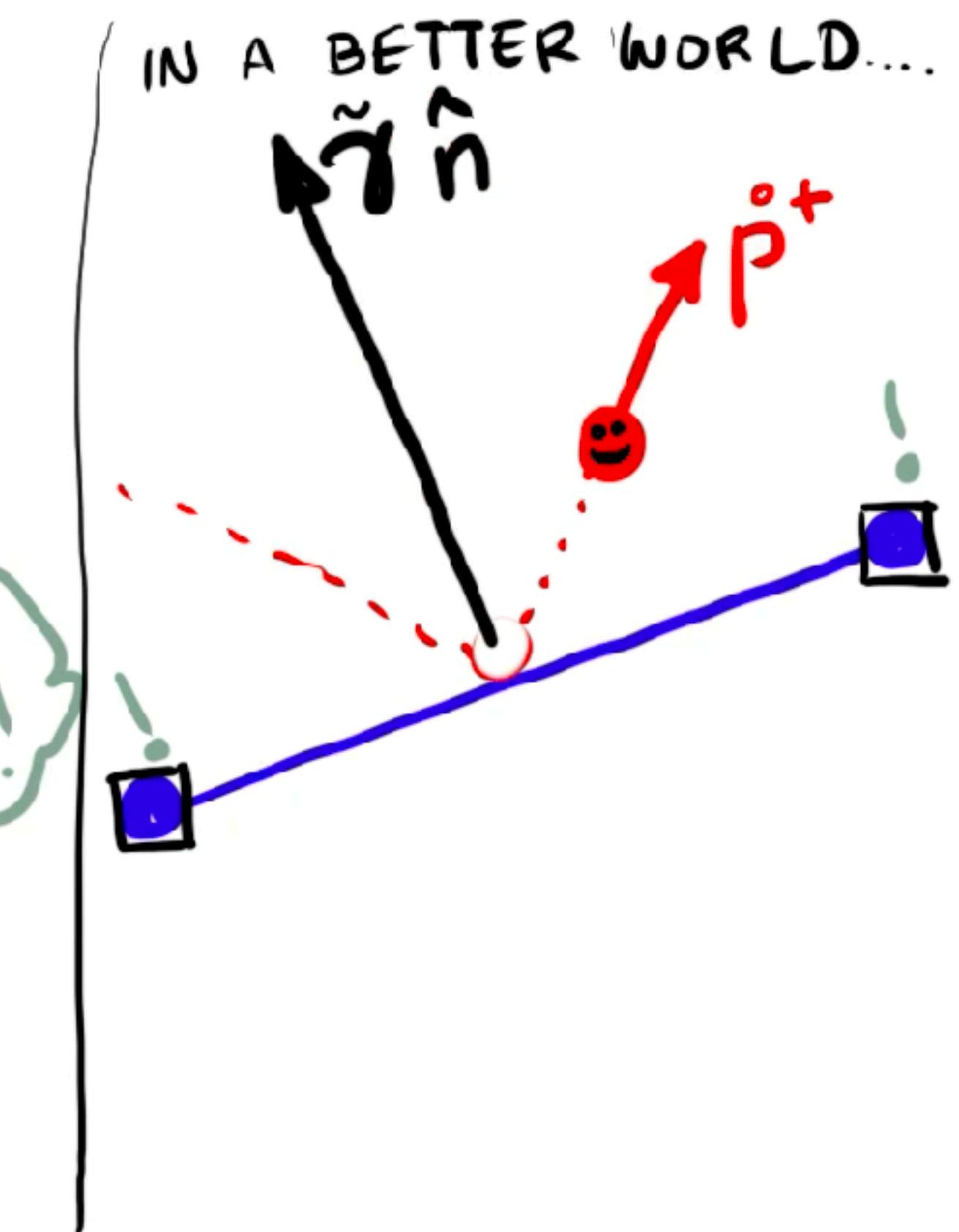
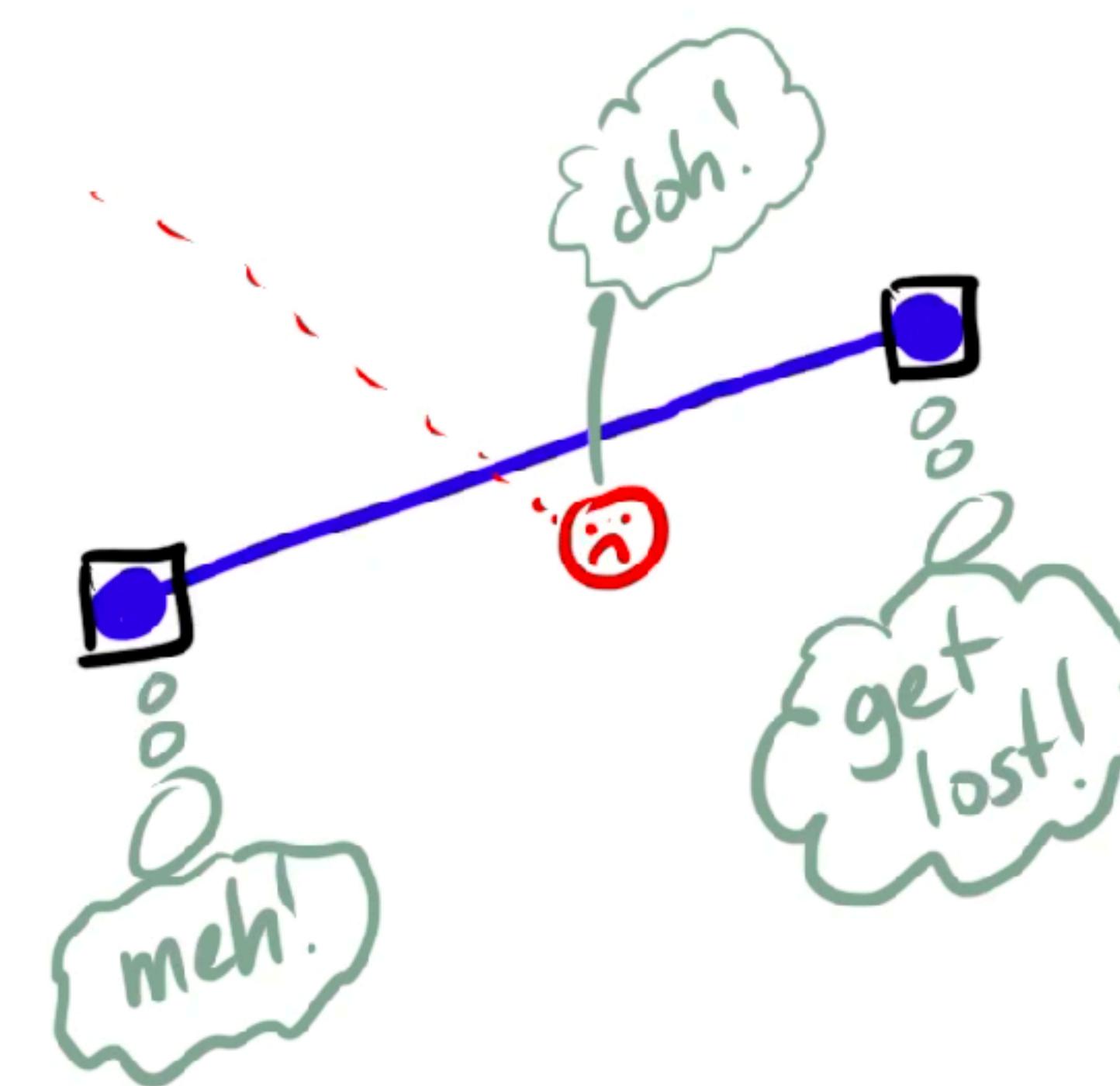
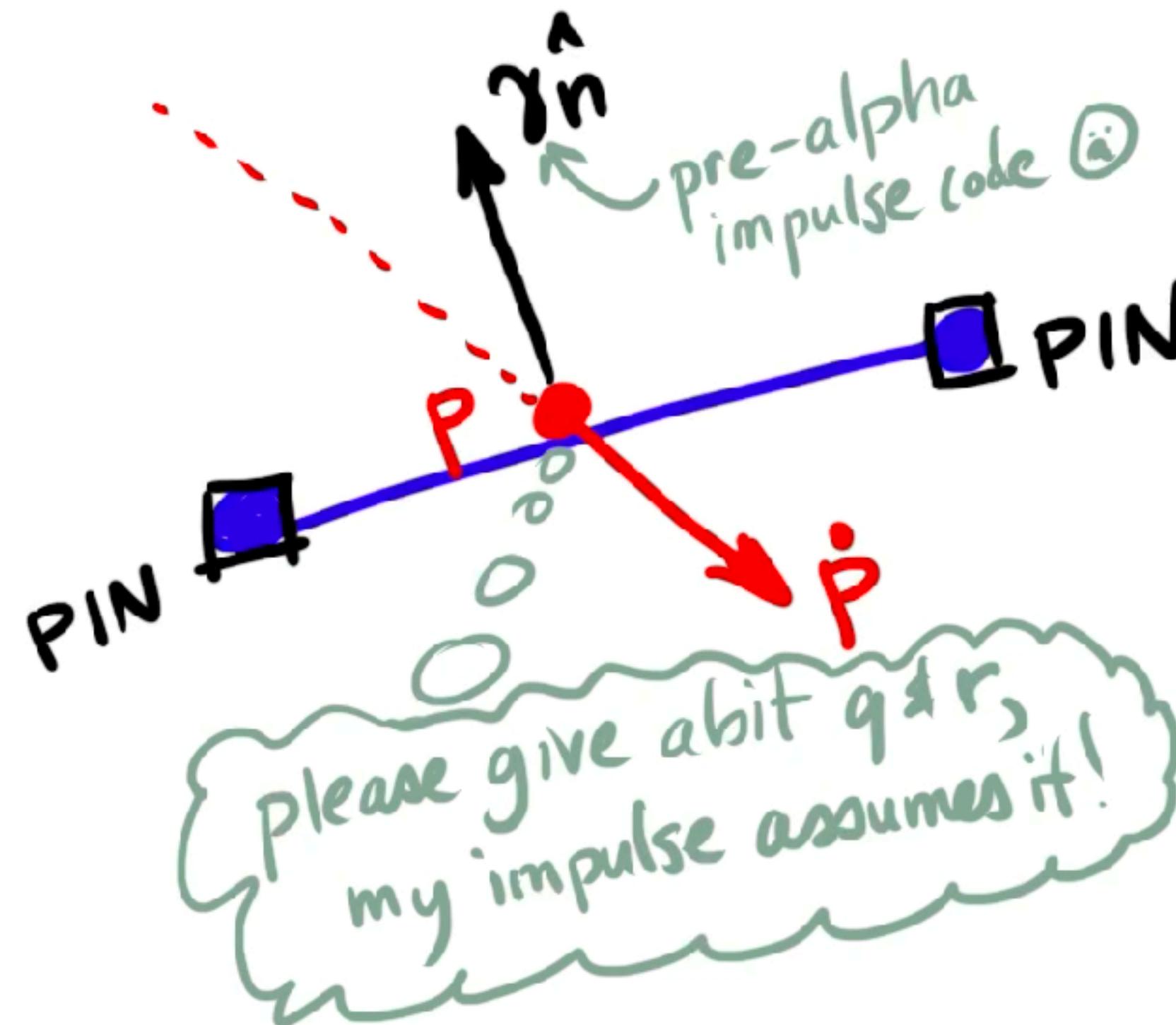
$$\frac{\gamma}{m_p} - \left( \bar{\alpha} \Delta \dot{q}_n + \alpha \Delta \dot{r}_n \right) = -(1+\varepsilon) (\dot{p}_n^- - \dot{c}_n^-)$$

$$\frac{\gamma}{m_p} - \left( \bar{\alpha} \left( -\bar{\alpha} \frac{\gamma}{m_q} \right) + \alpha \left( -\alpha \frac{\gamma}{m_r} \right) \right) = -(1+\varepsilon) v_n^-$$

$$\Rightarrow \gamma = \frac{-(1+\varepsilon) v_n^-}{\frac{1}{m_p} + \frac{\bar{\alpha}^2}{m_q} + \frac{\alpha^2}{m_r}} \xrightarrow{\sim m_{\text{eff}}^{-1}} \boxed{\gamma = (1+\varepsilon) m_{\text{eff}} (-v_n^-)} \rightarrow 0.$$

# Impulses with pinned particles

What if some vertices on the edge are pinned / immovable?



# Inverse-mass filtering (a really useful trick)

Use infinite mass for pinned/immovable particles:

INVERSE MASS:  $w_i = \begin{cases} 0, & \text{if particle "i" doesn't want to play today,} \\ \frac{1}{m_i}, & \text{otherwise} \end{cases}$

- Filters velocity updates:  $\Delta v_i = \frac{\Delta t f_i}{m_i} \implies \Delta v_i = \Delta t w_i f_i$
- Modified impulse calculation:

$$\gamma = \frac{(1+\varepsilon)(-\bar{v_n})}{w_p + \bar{\alpha}^2 w_q + \alpha^2 w_r} = (1+\varepsilon) m_{\text{eff}} (-\bar{v_n})$$

where  $m_{\text{eff}} = (w_p + \bar{\alpha}^2 w_q + \alpha^2 w_r)^{-1}$

# Impulse calculation: 3D point-triangle collision

- Similar derivation

# Impulse calculation: 3D edge-edge collision

- Similar derivation

# Implicit integration

# Important for cloth animation

SIGGRAPH 98, Orlando, July 19–24

COMPUTER GRAPHICS Proceedings, Annual Conference Series, 1998

SIGGRAPH 98, Orlando, July 19–24

COMPUTER GRAPHICS Proceedings, Annual Conference Series, 1998

## Large Steps in Cloth Simulation

David Baraff Andrew Witkin

Robotics Institute  
Carnegie Mellon University

### Abstract

The bottle-neck in most cloth simulation systems is that time steps must be small to avoid numerical instability. This paper describes a cloth simulation system that can stably take large time steps. The simulation system couples a new technique for enforcing constraints on individual cloth particles with an implicit integration method. The simulator models cloth as a triangular mesh, with internal cloth forces derived using a simple continuum formulation that supports modeling operations such as local anisotropic stretch or compression; a unified treatment of damping forces is included as well. The implicit integration method generates a large, unbandaged sparse linear system at each time step which is solved using a modified conjugate gradient method that simultaneously enforces particles' constraints. The constraints are always maintained exactly, independent of the number of conjugate gradient iterations, which is typically small. The resulting simulation system is significantly faster than previous accounts of cloth simulation systems in the literature.

**Keywords**—Cloth, simulation, constraints, implicit integration, physically-based modeling.

### 1 Introduction

Physically-based cloth animation has been a problem of interest to the graphics community for more than a decade. Early work by Terzopoulos *et al.* [17] and Terzopoulos and Fleischer [15, 16] on deformable models correctly characterized cloth simulation as a problem in deformable surfaces, and applied techniques from the mechanical engineering and finite element communities to the problem. Since then, other research groups (notably Carignan *et al.* [4] and Volino *et al.* [20, 21]; Breen *et al.* [3]; and Eberhardt *et al.* [5]) have taken up the challenge of cloth.

Although specific details vary (underlying representations, numerical solution methods, collision detection and constraint methods, etc.), there is a deep commonality amongst all the approaches: physically-based cloth simulation is formulated as a time-varying partial differential equation which, after discretization, is numerically solved as an ordinary differential equation

$$\ddot{\mathbf{x}} = \mathbf{M}^{-1} \left( -\frac{\partial E}{\partial \mathbf{x}} + \mathbf{F} \right). \quad (1)$$

In this equation the vector  $\mathbf{x}$  and diagonal matrix  $\mathbf{M}$  represent the geometric state and mass distribution of the cloth,  $E$ —a scalar function

Author affiliation (September 1998): David Baraff, Andrew Witkin, Pixar Animation Studios, 1001 West Cutting Blvd., Richmond, CA 94804. Email: deb@pixar.com, aw@pixar.com.

This is an electronic reprint. Permission is granted to copy part or all of this paper for noncommercial use provided that the title and this copyright notice appear. This electronic reprint is ©1998 by CMU. The original printed paper is ©1998 by the ACM.

of  $\mathbf{x}$ —yields the cloth's internal energy, and  $\mathbf{F}$  (a function of  $\mathbf{x}$  and  $\dot{\mathbf{x}}$ ) describes other forces (air-drag, contact and constraint forces, internal damping, etc.) acting on the cloth.

In this paper, we describe a cloth simulation system that is much faster than previously reported simulation systems. Our system's faster performance begins with the choice of an *implicit* numerical integration method to solve equation (1). The reader should note that the use of implicit integration methods in cloth simulation is far from novel: initial work by Terzopoulos *et al.* [15, 16, 17] applied such methods to the problem.<sup>1</sup> Since this time though, research on cloth simulation has generally relied on *explicit* numerical integration (such as Euler's method or Runge-Kutta methods) to advance the simulation, or, in the case of energy minimization, analogous methods such as steepest-descent [3, 10].

This is unfortunate. Cloth strongly resists stretching motions while being comparatively permissive in allowing bending or shearing motions. This results in a "stiff" underlying differential equation of motion [12]. Explicit methods are ill-suited to solving stiff equations because they require many small steps to stably advance the simulation forward in time.<sup>2</sup> In practice, the computational cost of an explicit method greatly limits the realizable resolution of the cloth. For some applications, the required spatial resolution—that is, the dimension  $n$  of the state vector  $\mathbf{x}$ —can be quite low: a resolution of only a few hundred particles (or nodal points, depending on your formulation/terminology) can be sufficient when it comes to modeling flags or tablecloths. To animate clothing, which is our main concern, requires much higher spatial resolution to adequately represent realistic (or even semi-realistic) wrinkling and folding configurations.

In this paper, we demonstrate that implicit methods for cloth overcome the performance limits inherent in explicit simulation methods. We describe a simulation system that uses a triangular mesh for cloth surfaces, eliminating topological restrictions of rectangular meshes, and a simple but versatile formulation of the internal cloth energy forces. (Unlike previous metric-tensor-based formulations [15, 16, 17, 4] which model some deformation energies as quadratic functions of positions, we model deformation energies only as quadratic functions with suitably large scaling. Quadratic energy models mesh well with implicit integration's numerical properties.) We also introduce a simple, unified treatment of damping forces, a subject which has been largely ignored thus far. A key step in our simulation process is the solution of an  $O(n) \times O(n)$  sparse linear system, which arises from the implicit integration method. In this respect, our implementation differs greatly from the implementation by Terzopoulos *et al.* [15, 17], which for large simulations

SIGGRAPH 98, Orlando, July 19–24

COMPUTER GRAPHICS Proceedings, Annual Conference Series, 1998

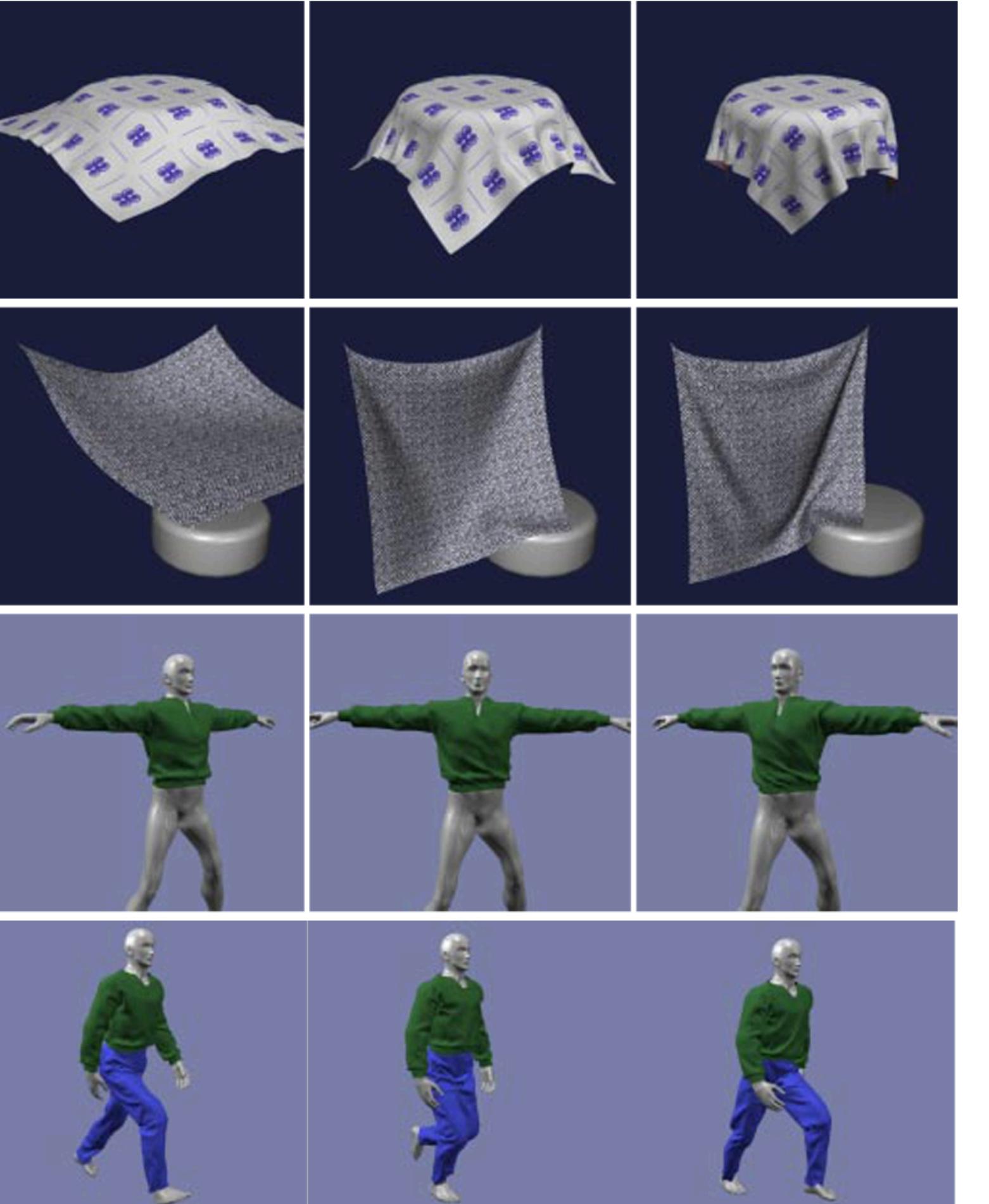


Figure 1 (top row): Cloth draping on cylinder; frames 8, 13 and 35. Figure 2 (second row): Sheet with two fixed particles; frames 10, 29 and 67. Figure 3 (third row): Shirt on twisting figure; frames 1, 24 and 46. Figure 4 (bottom row): Walking man; frames 30, 45 and 58.

SIGGRAPH 98, Orlando, July 19–24

COMPUTER GRAPHICS Proceedings, Annual Conference Series, 1998

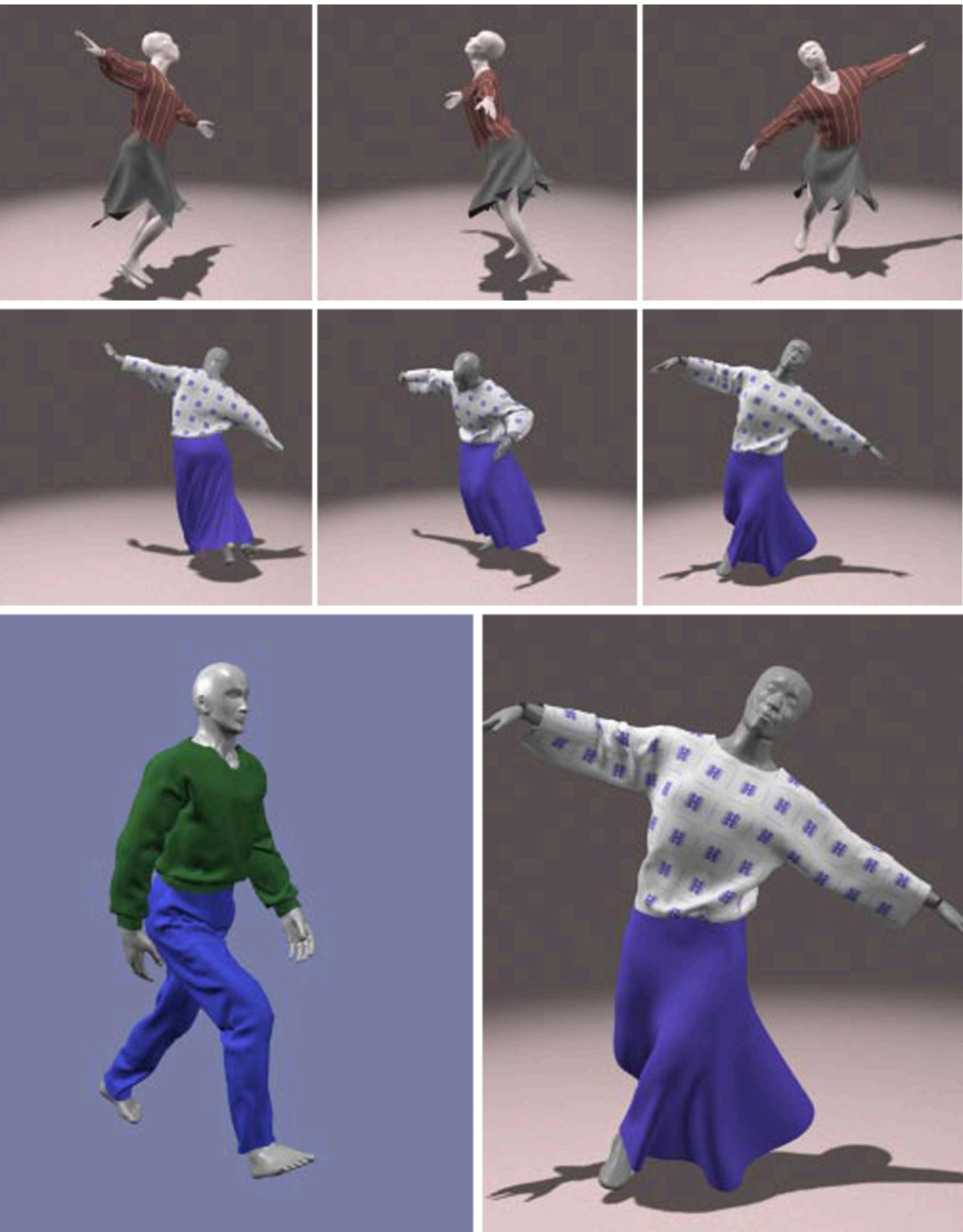


Figure 5 (top row): Dancer with short skirt; frames 110, 136 and 155. Figure 6 (middle row): Dancer with long skirt; frames 185, 215 and 236. Figure 7 (bottom row): Closeups from figures 4 and 6.



A W W W . . . D O N ' T C R Y .

It's just ~~some~~ derivatives!  
*more*

# Backward Euler (a.k.a. Implicit Euler) for particle systems

EQUATIONS OF MOTION:

$$\frac{d}{dt} \begin{pmatrix} p \\ v \end{pmatrix} = \begin{pmatrix} v \\ M^{-1}f \end{pmatrix}$$

Assume that  $f = f(p, v)$ .

Backward Euler update:

$$\begin{pmatrix} \Delta p \\ \Delta v \end{pmatrix} = h \begin{pmatrix} v + \Delta v \\ M^{-1} f(p + \Delta p, v + \Delta v) \end{pmatrix}$$

Linearize  $f$  about current  $(p, v)$  configuration,  $(p_0, v_0)$ :

$$f(p_0 + \Delta p, v_0 + \Delta v) \approx f(p_0, v_0) + \left. \frac{\partial f}{\partial p} \right|_{\substack{p=p_0 \\ v=v_0}} \Delta p + \left. \frac{\partial f}{\partial v} \right|_{\substack{p=p_0 \\ v=v_0}} \Delta v$$

$$\equiv f_0 + \left. \frac{\partial f_0}{\partial p} \right|_{p=p_0} \Delta p + \left. \frac{\partial f_0}{\partial v} \right|_{v=v_0} \Delta v$$

# Backward Euler (a.k.a. Implicit Euler) for particle

$$\Rightarrow \begin{pmatrix} \Delta P \\ M \Delta V \end{pmatrix} = h \begin{pmatrix} v_0 + \Delta V \\ f_0 + \frac{\partial f_0}{\partial P} \Delta P + \frac{\partial f_0}{\partial V} \Delta V \end{pmatrix}$$

Substituting 1<sup>st</sup> row  $\Delta P = h(v_0 + \Delta V)$  into 2<sup>nd</sup> row:

$$M \Delta V = h \left( f_0 + \frac{\partial f_0}{\partial P} \Delta P + \frac{\partial f_0}{\partial V} \Delta V \right)$$

SOLVING FOR  $\Delta V$ :

$$\left[ M - h \frac{\partial f_0}{\partial V} - h^2 \frac{\partial f}{\partial P} \right] \Delta V = h \left( f_0 + h \frac{\partial f}{\partial P} v_0 \right)$$

TYPICALLY A  
LARGE, SPARSE,  
SYMMETRIC MATRIX.

USUALLY POSITIVE DEFINITE IF  $h$  SMALL ENOUGH 😊

$$\boxed{A} \Delta V = b \xrightarrow{\text{solve}} \Delta V \rightarrow \Delta P = h(v_0 + \Delta V)$$

# Other implicit integrators of interest for $y' = f(t, y)$

## ■ Backward Differentiation Formula (BDF) methods

- e.g., BDF2, second-order accurate scheme

$$y_{n+2} - \frac{4}{3}y_{n+1} + \frac{1}{3}y_n = \frac{2}{3}hf(t_{n+2}, y_{n+2})$$

## ■ Trapezoidal rule, second-order accurate scheme

$$y_{n+1} = y_n + \frac{1}{2}h \left( f(t_n, y_n) + f(t_{n+1}, y_{n+1}) \right)$$

## ■ Newmark family of integrators

- Popular in structural mechanics
- Include symplectic integrators
- Both explicit and implicit schemes

# Linearized systems often require similar matrices

TRAPEZOID INTEGRATION IS IMPLICIT

$$y' = f(y) \longrightarrow y_{n+1} = y_n + \frac{h}{2} (f(y_n) + f(y_{n+1}))$$

For particle systems,

$$\begin{pmatrix} \Delta p \\ \Delta v \end{pmatrix} = \frac{h}{2} \left( M^{-1} f_0 + M^{-1} f(p_0 + \Delta p, v_0 + \Delta v) \right)$$

$$\begin{pmatrix} \Delta p \\ M \Delta v \end{pmatrix} \approx \frac{h}{2} \left( \begin{matrix} 2v_0 + \Delta v \\ 2f_0 + \frac{\partial f}{\partial p} \Delta p + \frac{\partial f}{\partial v} \Delta v \end{matrix} \right)$$

$$\Rightarrow \left[ M - \frac{h}{2} \frac{\partial f_0}{\partial p} - \frac{h^2}{2} \frac{\partial f_0}{\partial v} \right] \Delta v = \left( h f_0 + \frac{h^2}{2} \frac{\partial f_0}{\partial p} v_0 \right) \Rightarrow \Delta v \Rightarrow \Delta p.$$

SLIGHTLY DIFFERENT FROM  
BACKWARD EULER, &  
SECOND-ORDER ACCURATE