Lecture 7

Deformable Models

FUNDAMENTALS OF COMPUTER GRAPHICS

Animation & Simulation

Stanford CS248B, Fall 2023

PROFS. KAREN LIU & DOUG JAMES

Announcements

- 3 Programming Assignments:
  - P1: Pinball!
    - Due on Tues, Oct 24
  - P2: Deformable collision processing
    - Out on Tues, Oct 24
    - Due on Thurs, Nov 9
  - P3: Inverse Kinematics
- HW2: Collisions
  - Out Thurs, Oct 19
  - Due Mon, Oct 30

Stanford CS 248B Fall 2022 – Programming Assignment #1

Pinball!

Colliding Particles for Fun

Summary

In your first programming assignment you will design, model, simulate and play a virtual game of pinball. The main challenge is to simulate the ball motion subject to collisions with an environment composed of rigid obstacles (mostly static but some moving). The environment will be modeled using simple 2D shape primitives, for which we can use signed-distance fields (SDFs) to help process collisions robustly, even for fast-moving balls. Your implementation will be done in Open Processing (OP), and you and your classmates will be able to share and play each others' games (without sharing the code). Yes, this will be a groovy start to CS248B.

https://en.wikipedia.org/wiki/Pinball

2D Particle Simulation

For this first assignment, we will model the ball using a simple 2D point-like circular primitive (with unit mass, m=1, radius r, position **p**, and velocity **v**) falling under a small gravitational acceleration, **g**. This assumption ignores the more complex dynamics of a 3D spherical balling with rolling and frictional contact. The main challenge is not the time-stepping of the free-flight motion, but the detection and robust processing of ball-object collisions. As discussed in class, you can use a Symplectic Euler integrator for the ball dynamics, and use impulses to modify the velocity to resolve collisions. In difficult cases, such as for fast-moving balls, you will need to perform continuous collision processing to avoid missing collisions or having objects interpenetrate (more on that later).

The starter code includes a `Ball class` that can `draw()` a circle, and has a `timestep(dt)` method that performs discrete collision detection using the scene's signed-distance field, and applies a suitable collision impulse. You will need to modify the time-stepping scheme to handle collisions more robustly, as discussed below.

Announcements

- Re: Pinball questions and feedback
- Suggestion:
  - Do non-ray-marching parts first
  - Then do ray marching
- Ray marching:
  - Process all collisions on each timestep
  - Multiple collisions possible per timestep
    - May want to limit # for performance
  - Tip: Debug on simple scene first
    - 1D test problem w/ known solution
    - E.g., pinball in empty box
  - Discussion of depthMin and vn>0

Stanford CS 248B Fall 2022 – Programming Assignment #1

Pinball!

Colliding Particles for Fun

Summary

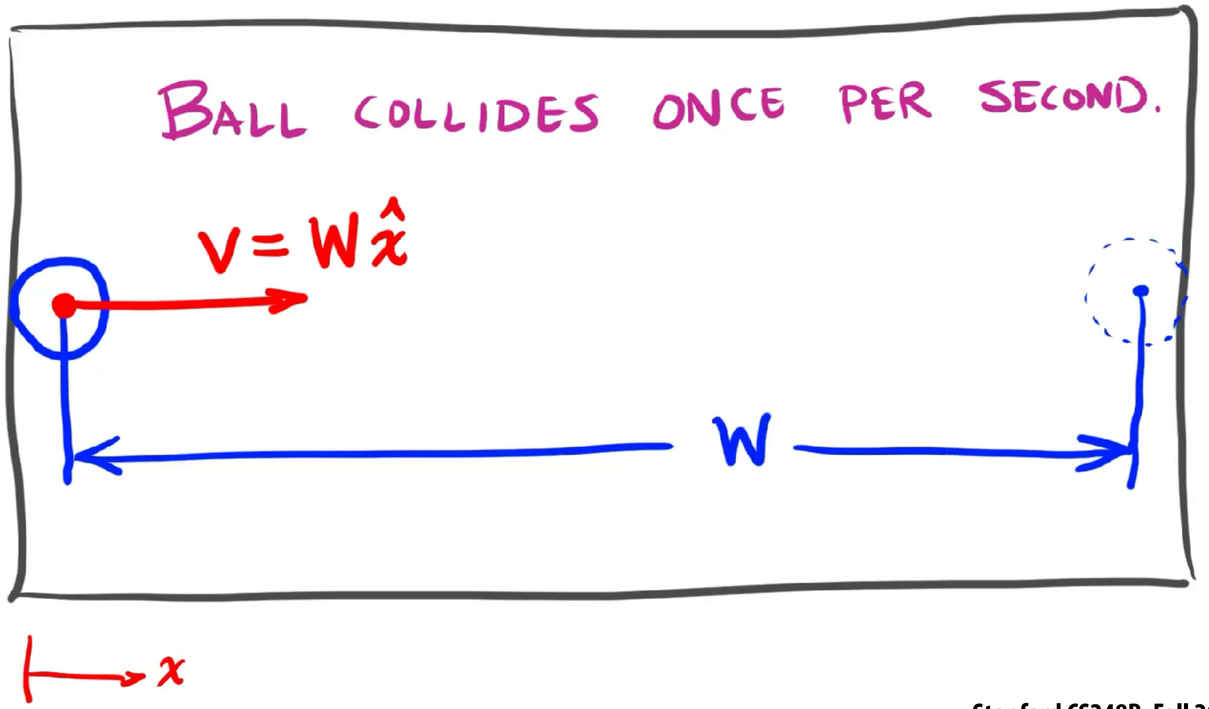
In your first programming assignment you will design, model, simulate and play a virtual game of pinball. The main challenge is to simulate the ball motion subject to collisions with an environment composed of rigid obstacles (mostly static but some moving). The environment will be modeled using simple 2D shape primitives, for which we can use signed-distance fields (SDFs) to help process collisions robustly, even for fast-moving balls. Your implementation will be done in Open Processing (OP), and you and your classmates will be able to share and play each others' games (without sharing the code). Yes, this will be a groovy start to CS248B.

https://en.wikipedia.org/wiki/Pinball

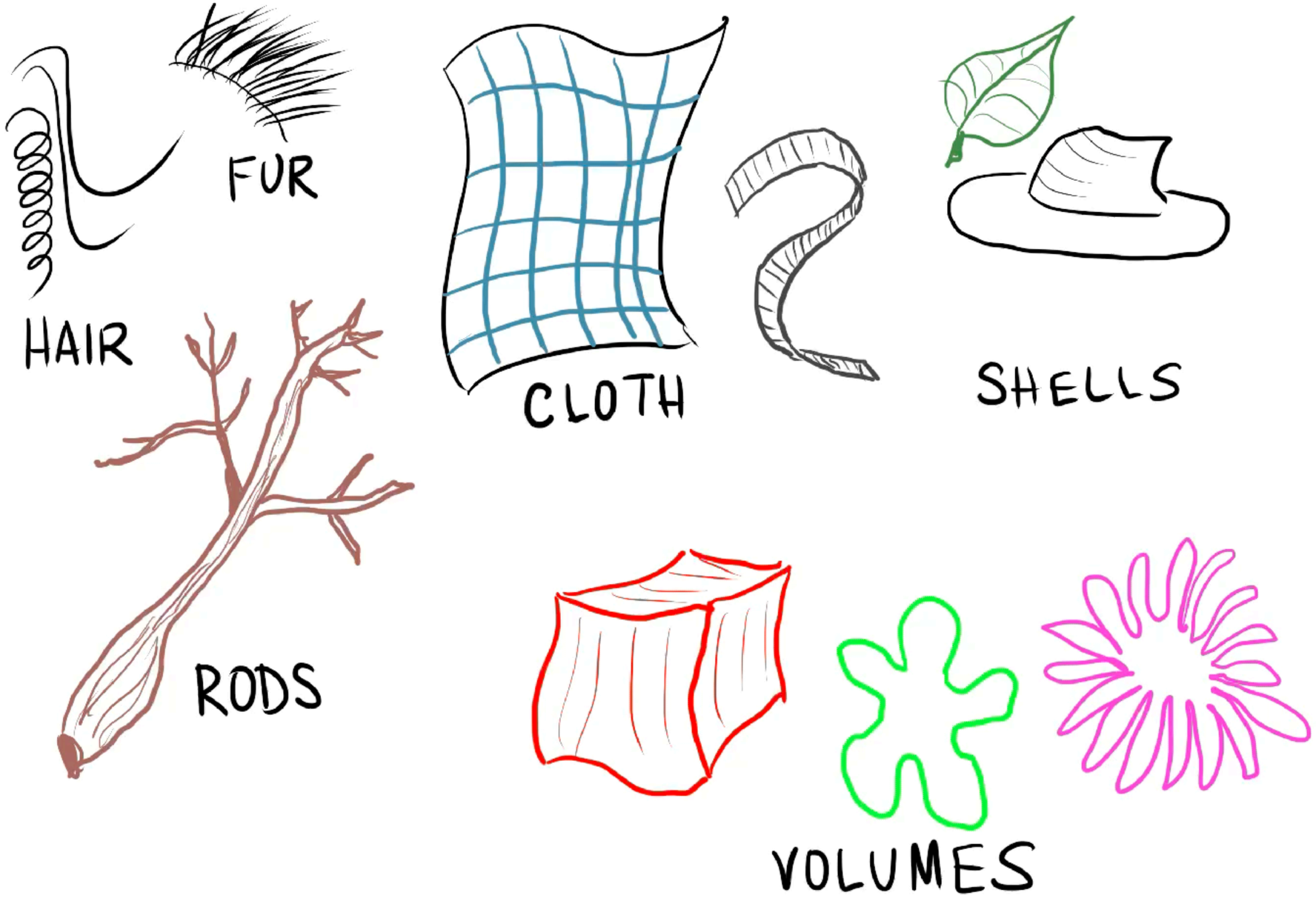
2D Particle Simulation

For this first assignment, we will model the ball using a simple 2D point-like circular primitive (with unit mass, m=1, radius r, position **p**, and velocity **v**) falling under a small gravitational acceleration, **g**. This assumption ignores the more complex dynamics of a 3D spherical balling with rolling and frictional contact. The main challenge is not the time-stepping of the free-flight motion, but the detection and robust processing of ball-object collisions. As discussed in class, you can use a Symplectic Euler integrator for the ball dynamics, and use impulses to modify the velocity to resolve collisions. In difficult cases, such as for fast-moving balls, you will need to perform continuous collision processing to avoid missing collisions or having objects interpenetrate (more on that later).

The starter code includes a `Ball class` that can `draw()` a circle, and has a `timestep(dt)` method that performs discrete collision detection using the scene's signed-distance field, and applies a suitable collision impulse. You will need to modify the time-stepping scheme to handle collisions more robustly, as discussed below.



Physics-based deformable models in graphics



Modeling deformation forces

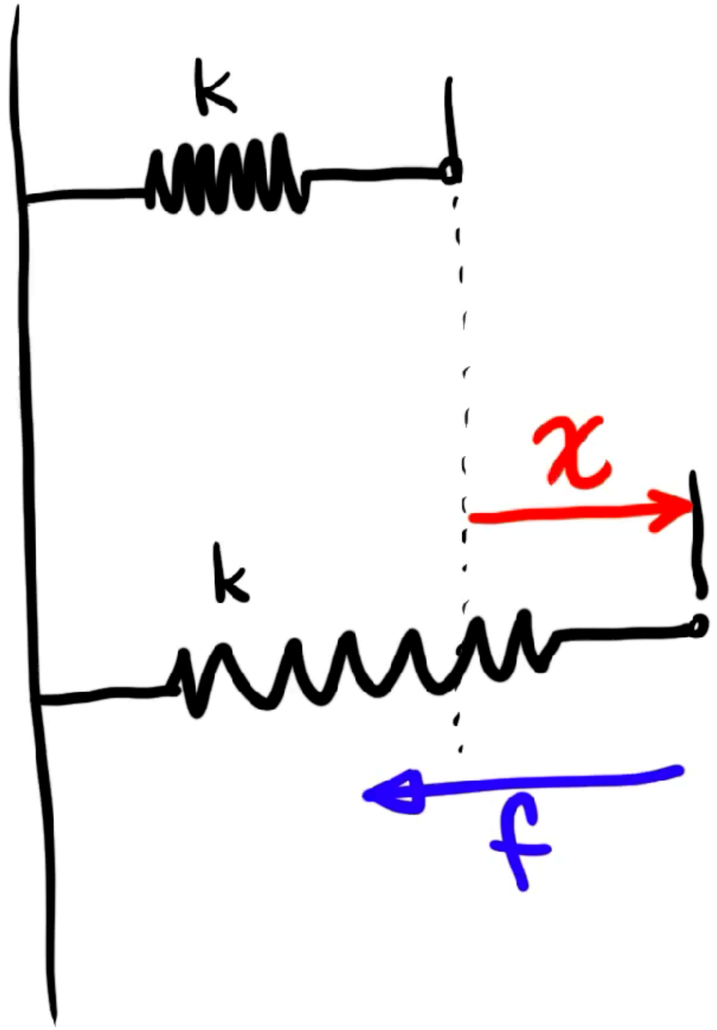
Reference:

David Baraff and Andrew Witkin, Physically Based Modeling, Online SIGGRAPH 2001 Course Notes, 2001.

- https://graphics.pixar.com/pbm2001
- Particle System Dynamics,
- Cloth and Fur Energy Functions

Recall: Hooke's law for 1D linear springs

- 1D elastic force



$$f = -k\chi$$

spring stiffness

displacement

Recall: Hooke's law for 1D linear springs

- Potential energy model

$$E(x) = \frac{1}{2} k x^2$$

$$f(x) = -\frac{d}{dx} E(x) = -kx$$

Conservative forces (preserve energy)

$$\text{WORK} = \int_{x_i}^{x_f} f(x)\, dX = -\int_{x_i}^{x_f} \frac{dE}{dX}\, dX = E(x_i) - E(x_f)$$

Hooke's Law for nD linear springs (zero rest-length)

$$E(p) = \frac{1}{2} k \| p - \bar{p} \|^2$$

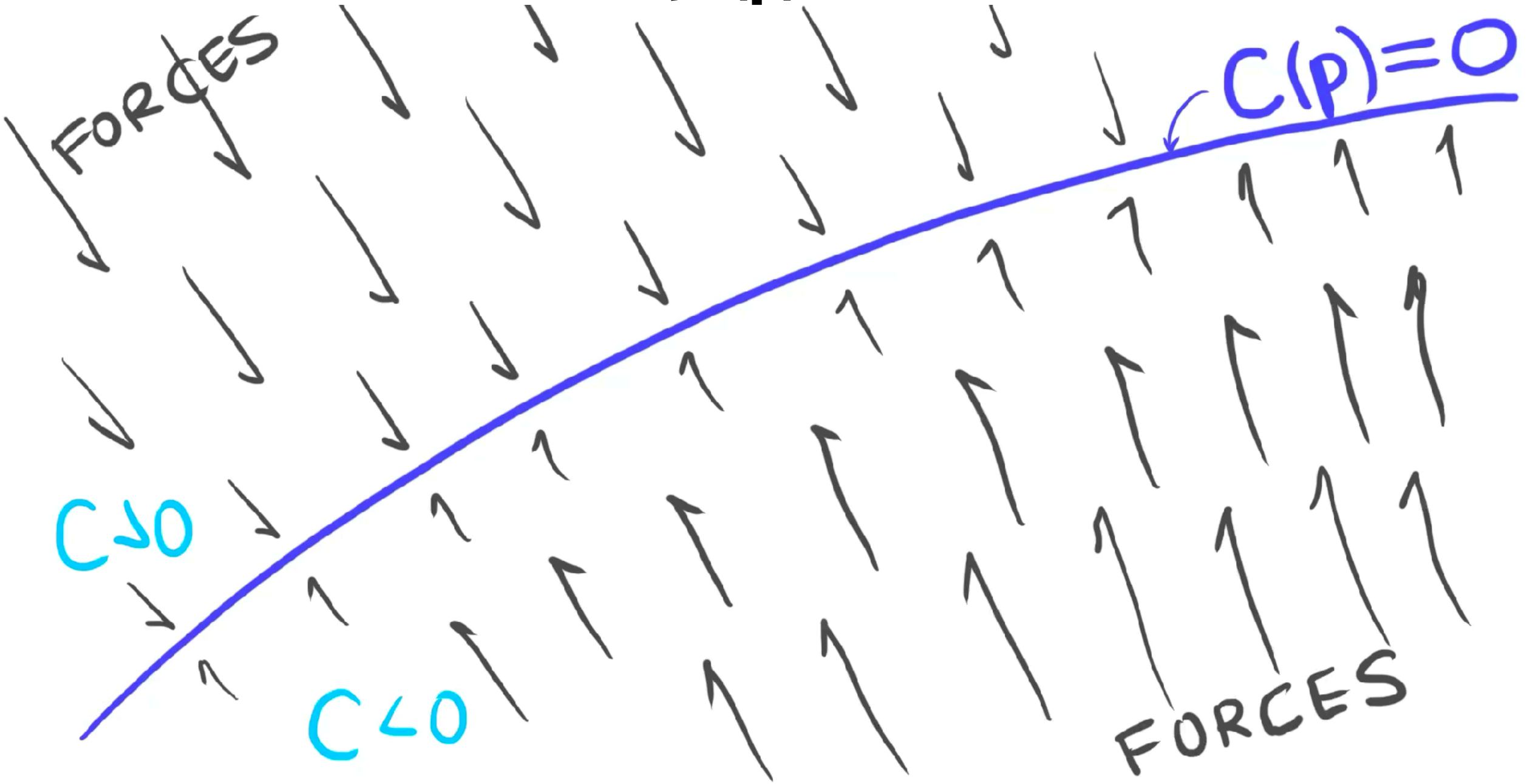
↑ rest position of particle

$$f(p) = -\frac{\partial}{\partial p} E(p) = -\frac{1}{2} k \frac{\partial}{\partial p} \| p - \bar{p} \|^2$$

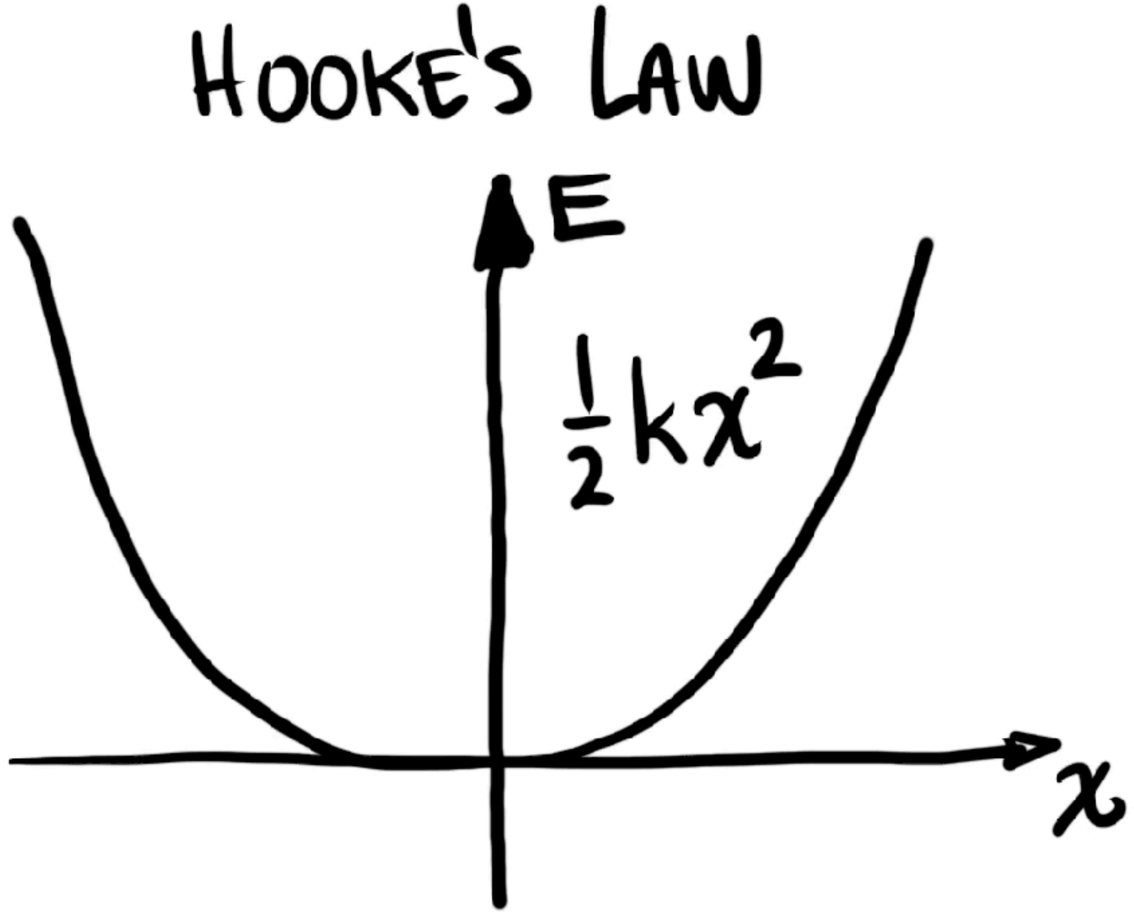
$$= -k(p - \bar{p})$$

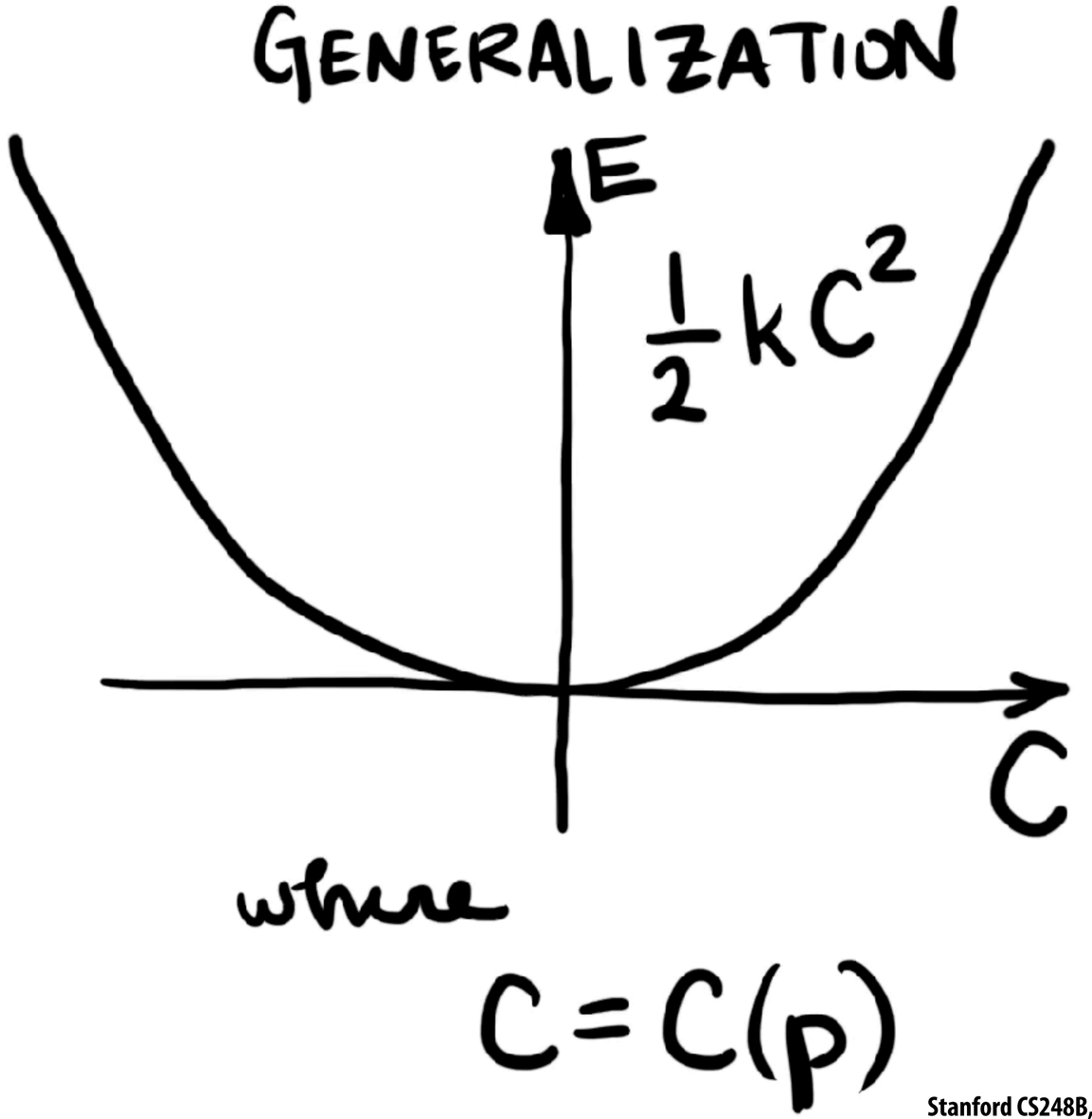
(pulls p back to $\bar{p}$)

Soft Constraint Functions, C(p)



Generalizing Hooke's Law: From $x \to 0$ to $C(\mathbf{p}) \to 0$.





Generalized spring model (one constraint)

$C(p) \in \mathbb{R}$: Generalized displacement

(soft constraint function)

Want $C \to 0$

$$E(p) = \frac{1}{2} k C^2(p)$$

$$f_i(p) = -\nabla_{p_i} E(p) = -\frac{\partial E(p)}{\partial p_i} = -k\, C\, \frac{\partial C}{\partial p_i}$$

stiffness · displacement · direction

Generalized spring model (M constraints)

Consider M constraint functions,

$$\mathbf{C} = \begin{pmatrix} C_1(\mathbf{p}) \\ C_2(\mathbf{p}) \\ \vdots \\ C_M(\mathbf{p}) \end{pmatrix} : \mathbb{R}^{nN} \longrightarrow \mathbb{R}^M$$

Generalized spring model (M constraints)

TOTAL ENERGY IS SUM OF SPRING ENERGIES:

$$E(p) = \sum_{i=1}^{M} \frac{1}{2} k_i C_i^2(p) \overset{k=k_i}{=\!=} \frac{1}{2} k\, C \cdot C$$

$$= \frac{1}{2} k\, C^T C = \frac{1}{2} k \|C\|^2$$

Generalized spring model (M constraints)

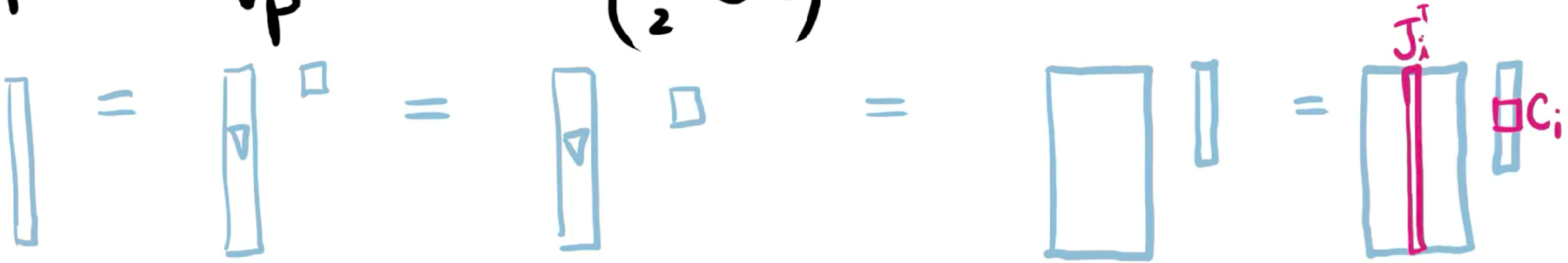
Force on particle i,

$$f_i = -\frac{\partial E}{\partial p_i} = -\frac{\partial}{\partial p_i}\left(\frac{1}{2}k\sum_{j=1}^{M} C_j^2(p)\right) = -k\sum_{j=1}^{M} C_j \frac{\partial C_j}{\partial p_i}$$

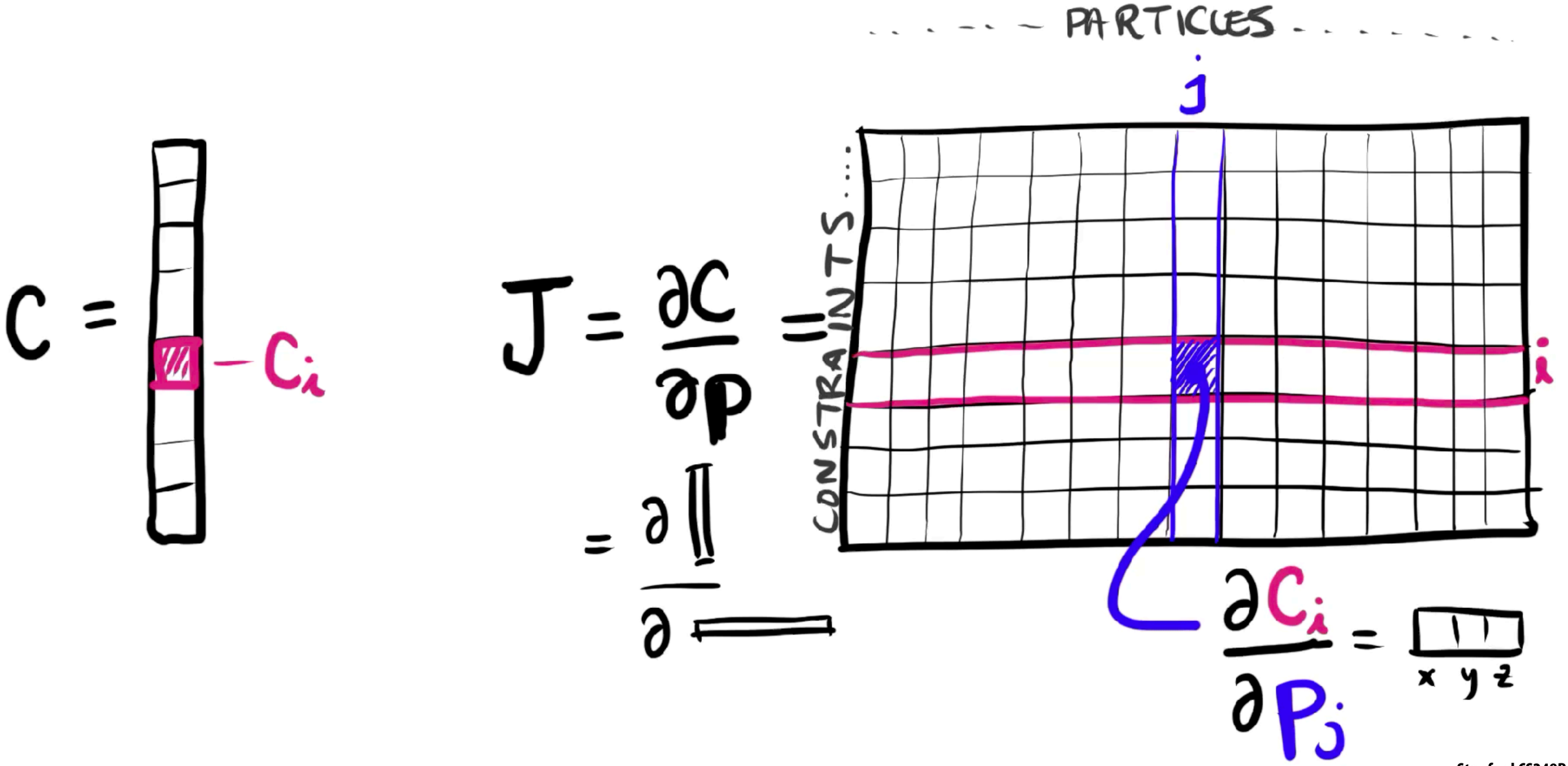
stiffness, displacement, direction

Force on particle system

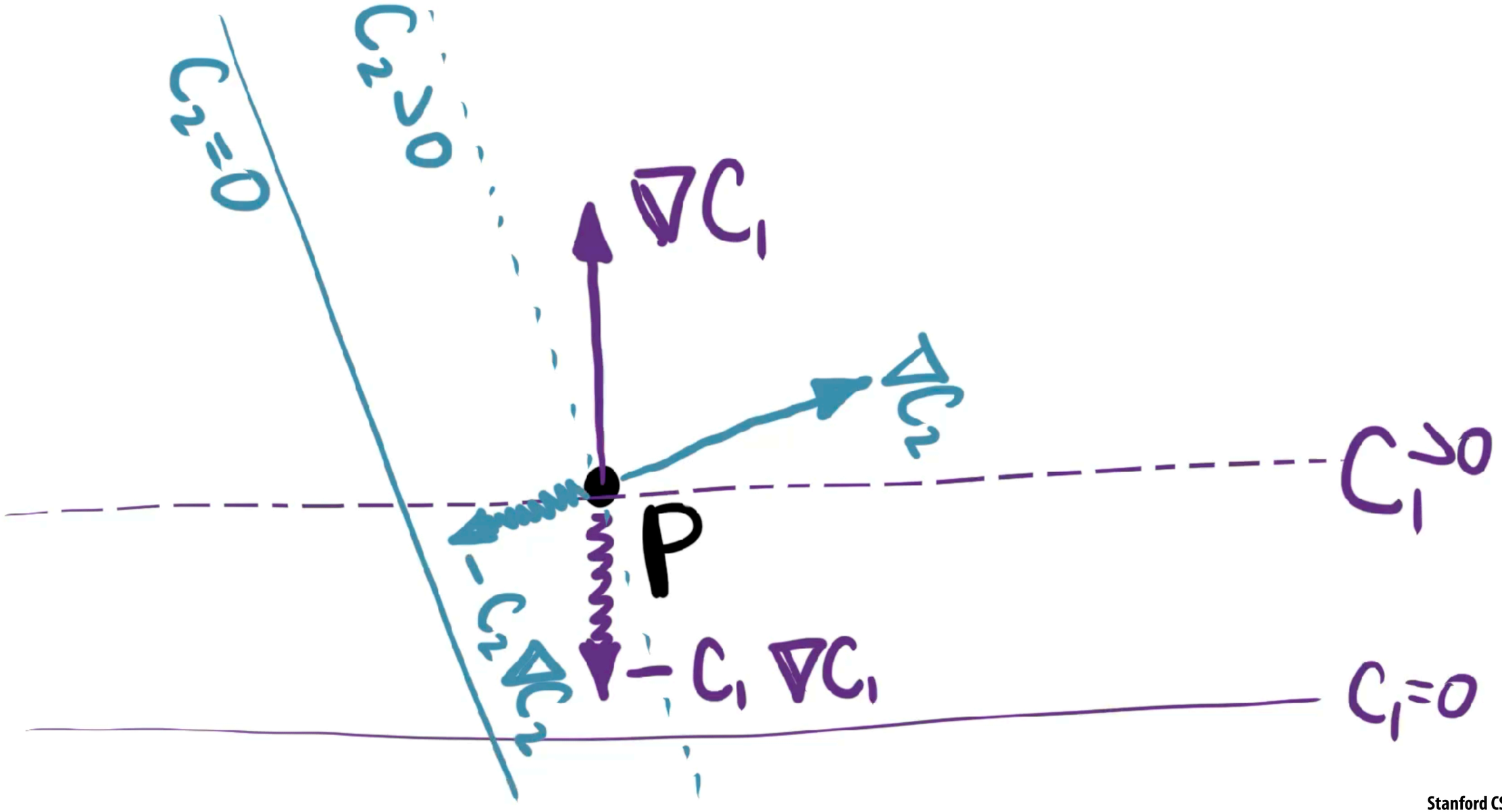
$$f = -\nabla_p E = -\nabla\left(\frac{k}{2} C^T C\right) = -k\,\nabla C^T C = -k J^T C$$



Constraint vector and Jacobian matrix



Geometric picture of generalized spring forces



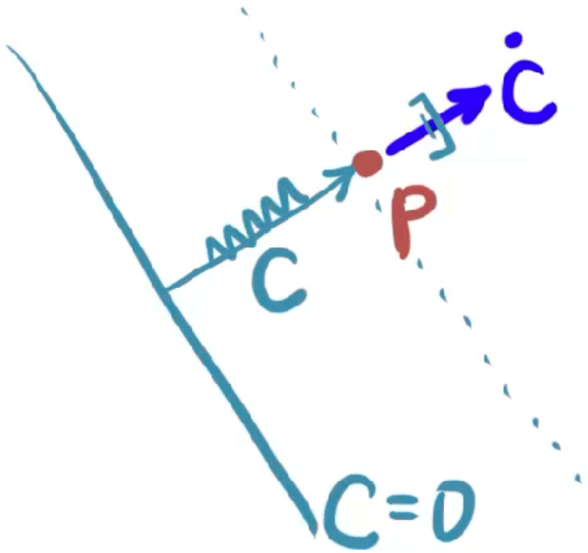
Generalized damped springs

Hooke's law
+ damping

$$f = -k_s x - k_d \dot{x}$$

Generalization
of damped spring

$$f = -J^T(k_s C + k_d \dot{C})$$



Taking derivatives

of things we'll need

PIXAR

FEATURE FILMS SHORT FILMS TECHNOLOGY CAREERS EXTRAS ABOUT

AWWW...DON'T CRY.

Notation: Gradient w.r.t. particle positions

Particle position variables:

$$p_i, \; i = 1, \ldots, N \qquad \text{where} \qquad p_i = \begin{pmatrix} x_i \\ y_i \\ z_i \end{pmatrix} \in \mathbb{R}^3 \qquad (n = 3)$$

Gradient w.r.t. particle position:

$\frac{\partial}{\partial p_i}$ or ∇_{p_i} or ∂_i is shorthand for $\begin{pmatrix} \frac{\partial}{\partial x_i} \\ \frac{\partial}{\partial y_i} \\ \frac{\partial}{\partial z_i} \end{pmatrix}$ OR $\equiv \begin{pmatrix} \frac{\partial}{\partial x_i} & \frac{\partial}{\partial y_i} & \frac{\partial}{\partial z_i} \end{pmatrix}$

- it's 3 derivatives in 3D.
- arrange them however is convenient.

Gradient of a constant (!)

Function

$$g(P) = c \in \mathbb{R}^1$$

Gradient w.r.t. $P_i \in \mathbb{R}^3$:

$$\partial_{P_i} g(P) = \partial_{P_i} c = \vec{0} \in \mathbb{R}^3 \qquad \text{e.g., } (0\ 0\ 0)$$

Gradient of some particle's position

Function:

$$g(P) = P_j = \square\!\square\!\square \in \mathbb{R}^3$$

$j \in \{1, 2, \ldots, N\}$ dummy index variable

Gradient w.r.t. P_i:

$$\partial_{P_i} g = \frac{\partial P_j}{\partial P_i} = \frac{\partial \square\!\square\!\square}{\partial \square\!\square\!\square} = \begin{bmatrix} \frac{\partial x_j}{\partial x_i} & \frac{\partial x_j}{\partial y_i} & \frac{\partial x_j}{\partial z_i} \\ \frac{\partial y_j}{\partial x_i} & \frac{\partial y_j}{\partial y_i} & \frac{\partial y_j}{\partial z_i} \\ \frac{\partial z_j}{\partial x_i} & \frac{\partial z_j}{\partial y_i} & \frac{\partial z_j}{\partial z_i} \end{bmatrix} = \delta_{ij} I_3 = \delta_{ij} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

"WHY?"

where $\delta_{ij} = \begin{cases} 1, & i = j \\ 0, & i \neq j \end{cases}$ Krönecker delta.

Gradient of position difference

Function:

$$g(\mathbf{p}) = \mathbf{p}_k - \mathbf{p}_\ell \quad \text{for some } k \neq \ell.$$

Gradient:

$$\frac{\partial g}{\partial \mathbf{p}_i} = \partial_i (\mathbf{p}_k - \mathbf{p}_\ell) = \partial_i \mathbf{p}_k - \partial_i \mathbf{p}_\ell$$

$$= (\delta_{ik} - \delta_{i\ell}) \mathbf{I}_3$$

Gradient of distance squared (e.g., springs)

Function:

$$g(\mathbf{p}) = \|\mathbf{p}_k - \mathbf{p}_\ell\|^2 = (\mathbf{p}_k - \mathbf{p}_\ell)^T(\mathbf{p}_k - \mathbf{p}_\ell)$$

Gradient:

$$\partial_i g = \left(\partial_i(\mathbf{p}_k - \mathbf{p}_\ell)\right)^T(\mathbf{p}_k - \mathbf{p}_\ell) + (\mathbf{p}_k - \mathbf{p}_\ell)^T\left(\partial_i(\mathbf{p}_k - \mathbf{p}_\ell)\right)$$

$$= 2\left(\partial_i(\mathbf{p}_k - \mathbf{p}_\ell)\right)^T(\mathbf{p}_k - \mathbf{p}_\ell)$$

$$= 2\left[(\delta_{ik} - \delta_{i\ell})\mathbb{I}_3\right](\mathbf{p}_k - \mathbf{p}_\ell)$$

$$= 2\,(\delta_{ik} - \delta_{i\ell})(\mathbf{p}_k - \mathbf{p}_\ell)$$

Gradient of distance (e.g., springs)

Function: $g(p) = \|p_k - p_l\| = \left(\|p_k - p_l\|^2 \right)^{1/2}$

Gradient:

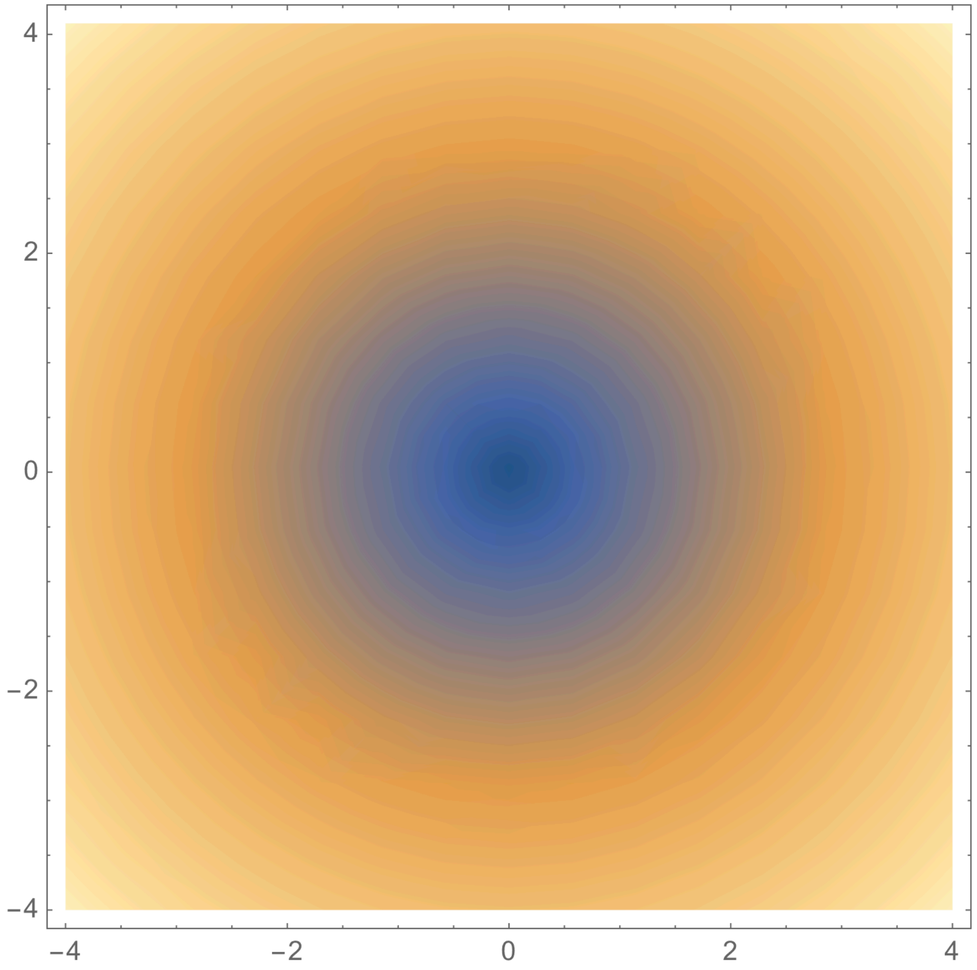
$$\partial_i g(p) = \partial_i \left(\|p_k - p_l\|^2 \right)^{1/2} = \frac{1}{2} \left(\|p_k - p_l\|^2 \right)^{-1/2} \left(\partial_i \|p_k - p_l\|^2 \right)$$

($1/R_{kl}$; from earlier)

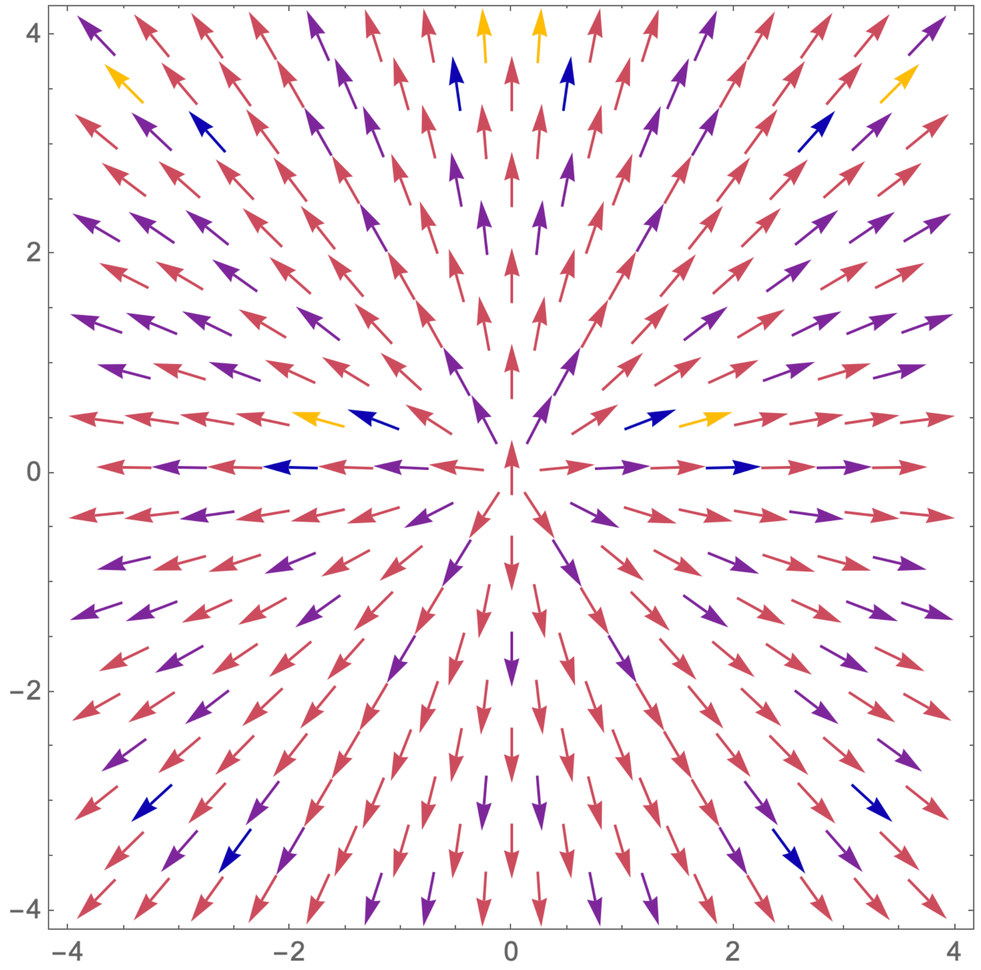
$$= \frac{1}{2} \frac{1}{\|p_k - p_l\|} \left(2 \left(\delta_{ik} - \delta_{il} \right)(p_k - p_l) \right)$$

$$= \left(\delta_{ik} - \delta_{il} \right) \underbrace{\frac{(p_k - p_l)}{\|p_k - p_l\|}}_{\text{unit vector, } \hat{R}_{kl}} = \left(\delta_{ik} - \delta_{il} \right) \hat{R}_{kl}$$

Gradient of distance (e.g., springs)



R



$\hat{R}$

Gradient of distance to power α (e.g., gravity, bending)

Function: $g(p) = R^\alpha$ for $\alpha \neq 0$ & $R = \|p_k - p_\ell\|$.

($p_k - p_\ell$ = $\vec{R}$)

Gradient:

$$\partial_i g = \partial_i R^\alpha = \alpha R^{\alpha-1} (\partial_i R)$$ (from before)

$$= \alpha R^{\alpha-1} (\delta_{ik} - \delta_{i\ell}) \hat{R}$$

$$= \alpha (\delta_{ik} - \delta_{i\ell}) R^{\alpha-2} \vec{R}$$

$$= \alpha \|p_k - p_\ell\|^{\alpha-2} (p_k - p_\ell)(\delta_{ik} - \delta_{i\ell})$$

Gradient of unit distance vector (e.g., bending)

Function: $g(p) = \hat{R} = \frac{\vec{R}}{R} = \frac{(p_k - p_l)}{\|p_k - p_l\|}$

Gradient:

$$\partial_i g = \partial_i \hat{R} = \partial_i(\vec{R} \cdot R^{-1}) = (\partial_i \vec{R})\, R^{-1} + \vec{R}\,(\partial_i R^{-1})$$

$$= \left[(\delta_{ik} - \delta_{il})\, I_3\right] R^{-1} + \vec{R}\left[-\frac{(\delta_{ik} - \delta_{il})}{R^2}\, \hat{R}^T\right]$$

$$= \frac{(\delta_{ik} - \delta_{il})}{R}\left[I_3 - \hat{R}\,\hat{R}^T\right]$$

(a rank-2 matrix in 3D)

Some deformation forces

Zero rest-length spring between two particles

Energy: $E(p) = \frac{1}{2} k \| p_k - p_\ell \|^2$

Force on particle i:

from before

$$f_i = -\partial_i E = -\frac{1}{2} k \left(\partial_i \| p_k - p_\ell \|^2 \right)$$

$$= -\frac{1}{2} k \left[2 \left(\delta_{ik} - \delta_{i\ell} \right) \left(p_k - p_\ell \right) \right]$$

$$= -k \left(\delta_{ik} - \delta_{i\ell} \right) \left(p_k - p_\ell \right)$$

$$\Longrightarrow f_k = -f_\ell \quad \ddot\smile$$

Linear spring with finite rest-length, L

Energy: $E(p) = \frac{1}{2} k \left(\underbrace{\| p_k - p_\ell \|}_{R} - L \right)^2$

Force on particle i:

$$f_i = -\partial_i E = -\frac{1}{2} k \left(\partial_i (R-L)^2 \right) = -\frac{1}{2} k \left(2 (R-L) \overbrace{\partial_i R}^{\text{from before}} \right)$$

$$= -k (\delta_{ik} - \delta_{i\ell})(R-L)\hat{R}$$

$$= -k (\delta_{ik} - \delta_{i\ell})(\| p_k - p_\ell \| - L) \frac{(p_k - p_\ell)}{\| p_k - p_\ell \|}$$

$$\Longrightarrow f_k = -f_\ell$$

Damped linear spring with finite rest-length, L

Recall generalized damped spring force:

$$f = -J^T(k_s C + k_d \dot{C})$$

By inspection, we have

$$f_i = -(\delta_{ik} - \delta_{il})\,\hat{R}_{kl}\left\{ k_s \overbrace{(R_{kl} - L)}^{C} + k_c \overbrace{\dot{R}_{kl}}^{\dot{C}} \right\}$$

where

$$\dot{R}_{kl} = \frac{d}{dt} R_{kl}(p) = \sum_{i=k,l} \partial_i R_{kl}(p) \cdot \frac{dp_i}{dt}$$

($\frac{dp_i}{dt} = v_i$)

$$= \sum_{i=k,l} (\delta_{ik} - \delta_{il})\,\hat{R}_{kl} \cdot v_i = (v_k - v_l) \cdot \hat{R}_{kl} = \left(\begin{array}{c}\text{relative velocity along} \\ \text{spring}\end{array}\right)$$

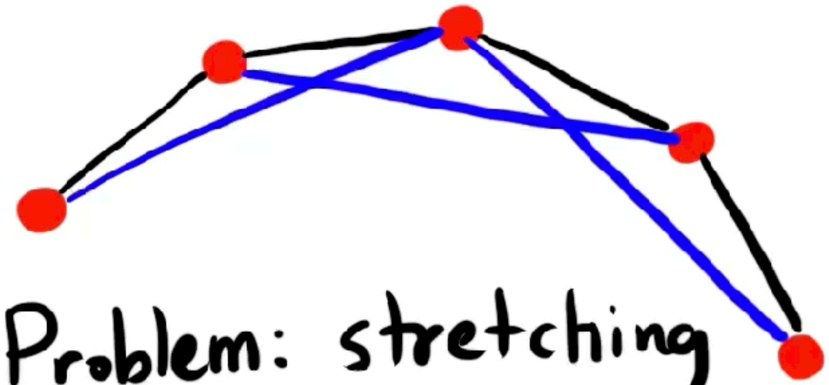
Hair bending forces

"Sulley" [Monsters, Inc., Pixar]

Hair bending force (flat rest shape)

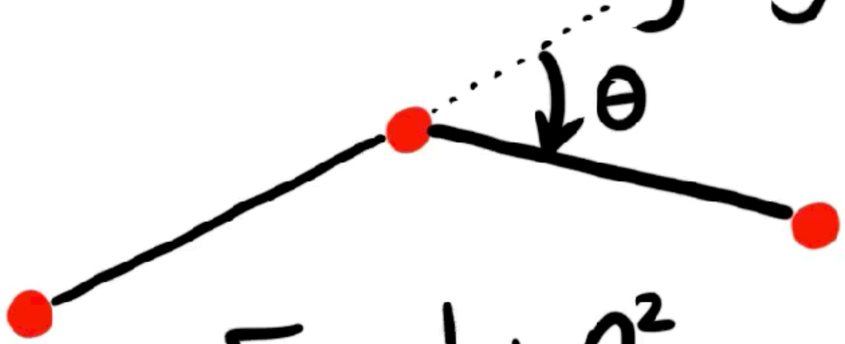
Bending models.

Simple but hacky:
add linear springs.



Problem: stretching and bending forces fight.

Better: Penalize bending angles.



$$E = \frac{1}{2} k \theta^2 \quad \text{per joint}$$

Easier to differentiate

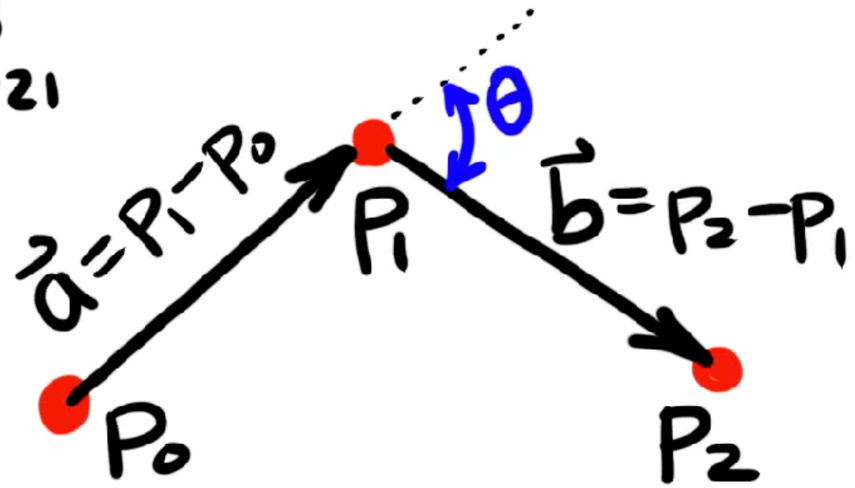
$$k \sin^2 \frac{\theta}{2} = \frac{k}{2}(1 - \cos\theta)$$

or just $$E = -\frac{k}{2} \cos\theta$$

Hair bending force (flat rest shape)

$$E = -\frac{k}{2}\,\hat{a}\cdot\hat{b} = -\frac{k}{2}\cos\theta = -\frac{k}{2}\,\hat{R}_{10}\cdot\hat{R}_{21}$$

$$f_i = -\partial_i E$$



Hair bending force (flat rest shape)

- **Symbolic differentiation (Mathematica). Easy & correct, but cumbersome expressions.**

```
In[22]:= Rhat[x_, y_] := {x, y} / Sqrt[x^2 + y^2];

In[23]:= Energy = -(k/2) Dot[Rhat[x1 - x0, y1 - y0], Rhat[x2 - x1, y2 - y1]]
```

$$\text{Out[23]=} \quad -\frac{1}{2}\,k\left(\frac{(-x0+x1)\,(-x1+x2)}{\sqrt{(-x0+x1)^2+(-y0+y1)^2}\,\sqrt{(-x1+x2)^2+(-y1+y2)^2}}+\frac{(-y0+y1)\,(-y1+y2)}{\sqrt{(-x0+x1)^2+(-y0+y1)^2}\,\sqrt{(-x1+x2)^2+(-y1+y2)^2}}\right)$$

```
In[24]:= f0 = D[Energy, {{x0, y0}}]
```

$$\text{Out[24]=} \quad \left\{-\frac{1}{2}\,k\left(\frac{(-x0+x1)^2\,(-x1+x2)}{\left((-x0+x1)^2+(-y0+y1)^2\right)^{3/2}\sqrt{(-x1+x2)^2+(-y1+y2)^2}}-\frac{-x1+x2}{\sqrt{(-x0+x1)^2+(-y0+y1)^2}\,\sqrt{(-x1+x2)^2+(-y1+y2)^2}}+\frac{(-x0+x1)\,(-y0+y1)\,(-y1+y2)}{\left((-x0+x1)^2+(-y0+y1)^2\right)^{3/2}\sqrt{(-x1+x2)^2+(-y1+y2)^2}}\right),\right.$$
$$\left.-\frac{1}{2}\,k\left(\frac{(-x0+x1)\,(-x1+x2)\,(-y0+y1)}{\left((-x0+x1)^2+(-y0+y1)^2\right)^{3/2}\sqrt{(-x1+x2)^2+(-y1+y2)^2}}+\frac{(-y0+y1)^2\,(-y1+y2)}{\left((-x0+x1)^2+(-y0+y1)^2\right)^{3/2}\sqrt{(-x1+x2)^2+(-y1+y2)^2}}-\frac{-y1+y2}{\sqrt{(-x0+x1)^2+(-y0+y1)^2}\,\sqrt{(-x1+x2)^2+(-y1+y2)^2}}\right)\right\}$$

```
In[25]:= f1 = D[Energy, {{x1, y1}}]
```

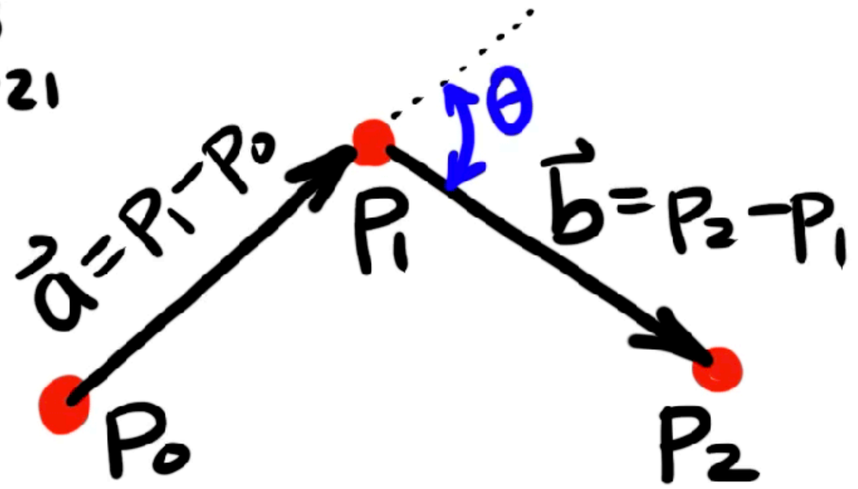
$$\text{Out[25]=} \quad \left\{-\frac{1}{2}\,k\left(\frac{(-x0+x1)\,(-x1+x2)^2}{\sqrt{(-x0+x1)^2+(-y0+y1)^2}\,\left((-x1+x2)^2+(-y1+y2)^2\right)^{3/2}}+\frac{(-x1+x2)\,(-y0+y1)\,(-y1+y2)}{\sqrt{(-x0+x1)^2+(-y0+y1)^2}\,\left((-x1+x2)^2+(-y1+y2)^2\right)^{3/2}}-\frac{(-x0+x1)^2\,(-x1+x2)}{\left((-x0+x1)^2+(-y0+y1)^2\right)^{3/2}\sqrt{(-x1+x2)^2+(-y1+y2)^2}}-\right.$$
$$\left.\frac{-x0+x1}{\sqrt{(-x0+x1)^2+(-y0+y1)^2}\,\sqrt{(-x1+x2)^2+(-y1+y2)^2}}+\frac{-x1+x2}{\sqrt{(-x0+x1)^2+(-y0+y1)^2}\,\sqrt{(-x1+x2)^2+(-y1+y2)^2}}-\frac{(-x0+x1)\,(-y0+y1)\,(-y1+y2)}{\left((-x0+x1)^2+(-y0+y1)^2\right)^{3/2}\sqrt{(-x1+x2)^2+(-y1+y2)^2}}\right),$$
$$-\frac{1}{2}\,k\left(\frac{(-x0+x1)\,(-x1+x2)\,(-y1+y2)}{\sqrt{(-x0+x1)^2+(-y0+y1)^2}\,\left((-x1+x2)^2+(-y1+y2)^2\right)^{3/2}}+\frac{(-y0+y1)\,(-y1+y2)^2}{\sqrt{(-x0+x1)^2+(-y0+y1)^2}\,\left((-x1+x2)^2+(-y1+y2)^2\right)^{3/2}}-\frac{(-x0+x1)\,(-x1+x2)\,(-y0+y1)}{\left((-x0+x1)^2+(-y0+y1)^2\right)^{3/2}\sqrt{(-x1+x2)^2+(-y1+y2)^2}}-\right.$$
$$\left.\frac{-y0+y1}{\sqrt{(-x0+x1)^2+(-y0+y1)^2}\,\sqrt{(-x1+x2)^2+(-y1+y2)^2}}-\frac{(-y0+y1)^2\,(-y1+y2)}{\left((-x0+x1)^2+(-y0+y1)^2\right)^{3/2}\sqrt{(-x1+x2)^2+(-y1+y2)^2}}+\frac{-y1+y2}{\sqrt{(-x0+x1)^2+(-y0+y1)^2}\,\sqrt{(-x1+x2)^2+(-y1+y2)^2}}\right)\right\}$$

```
In[26]:= f2 = D[Energy, {{x2, y2}}]
```

$$\text{Out[26]=} \quad \left\{-\frac{1}{2}\,k\left(-\frac{(-x0+x1)\,(-x1+x2)^2}{\sqrt{(-x0+x1)^2+(-y0+y1)^2}\,\left((-x1+x2)^2+(-y1+y2)^2\right)^{3/2}}-\frac{(-x1+x2)\,(-y0+y1)\,(-y1+y2)}{\sqrt{(-x0+x1)^2+(-y0+y1)^2}\,\left((-x1+x2)^2+(-y1+y2)^2\right)^{3/2}}+\frac{-x0+x1}{\sqrt{(-x0+x1)^2+(-y0+y1)^2}\,\sqrt{(-x1+x2)^2+(-y1+y2)^2}}\right),\right.$$
$$\left.-\frac{1}{2}\,k\left(-\frac{(-x0+x1)\,(-x1+x2)\,(-y1+y2)}{\sqrt{(-x0+x1)^2+(-y0+y1)^2}\,\left((-x1+x2)^2+(-y1+y2)^2\right)^{3/2}}-\frac{(-y0+y1)\,(-y1+y2)^2}{\sqrt{(-x0+x1)^2+(-y0+y1)^2}\,\left((-x1+x2)^2+(-y1+y2)^2\right)^{3/2}}+\frac{-y0+y1}{\sqrt{(-x0+x1)^2+(-y0+y1)^2}\,\sqrt{(-x1+x2)^2+(-y1+y2)^2}}\right)\right\}$$

Hair bending force (flat rest shape)

$$E = -\frac{k}{2}\hat{a}\cdot\hat{b} = -\frac{k}{2}\cos\theta = -\frac{k}{2}\hat{R}_{10}\cdot\hat{R}_{21}$$



$$f_i = -\partial_i E = +\frac{k}{2}\partial_i(\hat{a}\cdot\hat{b})$$

$$= \frac{k}{2}\left[(\partial_i\hat{a})\cdot\hat{b} + \hat{a}\cdot(\partial_i\hat{b})\right]$$

recall $\partial_i \hat{R}_{kl} = \dfrac{(\delta_{ik}-\delta_{il})}{R_{kl}}\left[I - \hat{R}_{kl}\hat{R}_{kl}^T\right]$

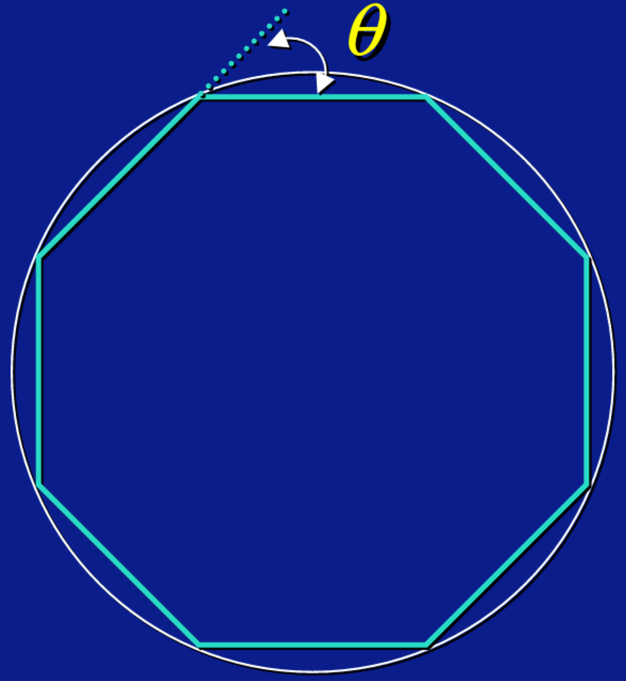
$$= \frac{k}{2}\left[\frac{(\delta_{i1}-\delta_{i0})}{a}\left(\hat{b}-\hat{a}(\hat{a}\cdot\hat{b})\right) + \frac{(\delta_{i2}-\delta_{i1})}{b}\left(\hat{a}-\hat{b}(\hat{b}\cdot\hat{a})\right)\right]$$

$$\Rightarrow f_0 = -\frac{k}{2a}\left(\hat{b}-\hat{a}(\hat{a}\cdot\hat{b})\right),\ f_2 = \frac{k}{2b}\left(\hat{a}-\hat{b}(\hat{a}\cdot\hat{b})\right),\ f_1 = -f_0-f_2$$

Hair bending force (flat rest shape)

- Length-dependent bending stiffness
- Reference: Kass

Discretization



Consider a discretized circle.

$$E = \frac{1}{2} k \sum \theta^2$$

Again, constant energy implies:

$k \propto n$ or $k_i \propto \frac{1}{l_i}$

Bending forces for cloth animation

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, unbanded 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

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

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 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.[1] 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 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.[2] 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 quartic 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

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.

[1] Additional use of implicit methods in animation and dynamics work includes Kass and Miller [8], Terzopoulos and Qin [18], and Tu [19].

[2] Even worse, the number of time steps per frame tends to increase along with the problem size, for an explicit method. Cloth simulations of size n—meaning $\mathbf{x} \in \mathbb{R}^{O(n)}$—generally require $O(n)$ explicit steps per unit simulated time. Because the cost of an explicit step is also $O(n)$ (setting aside complications such as collision detection for now) explicit methods for cloth require time $O(n^2)$—or worse.

SIGGRAPH 98, Orlando, July 19–24 COMPUTER GRAPHICS Proceedings, Annual Conference Series, 1998

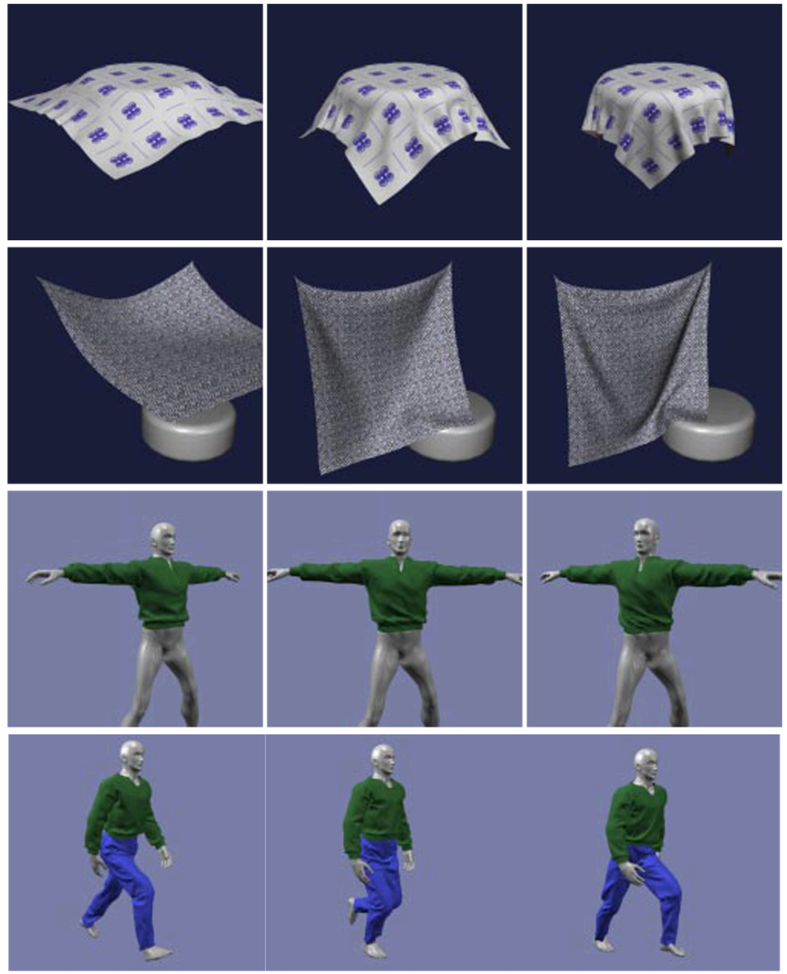
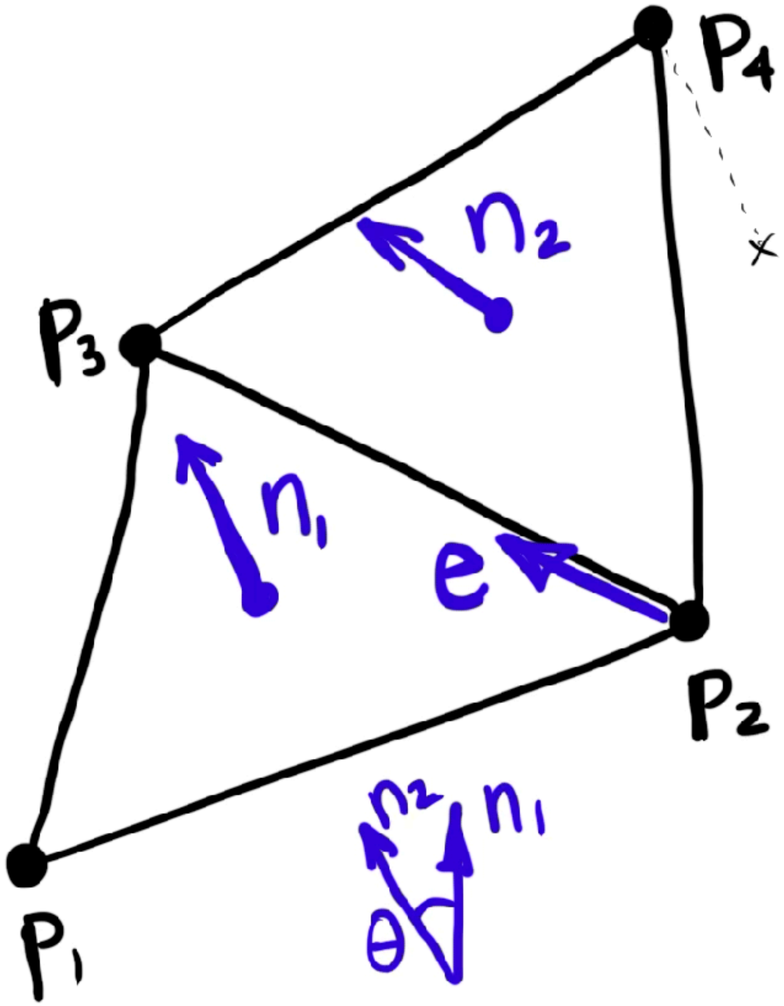


Figure 1 (top row): Cloth draping on cylinder; frames 8, 13 and 35. Figure 2 (second row): Sheet with two fixed particles; frames 10, 29 and 67. Figure 3 (third row): Shirt on twisting figure; frames 1, 24 and 46. Figure 4 (bottom row): Walking man; frames 30, 45 and 58.

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.

Bending force models for cloth and shells



Let $n_1 = \dfrac{(p_2 - p_1) \times (p_3 - p_1)}{\|(p_2 - p_1) \times (p_3 - p_1)\|}$

$n_2 = \dfrac{(p_4 - p_2) \times (p_3 - p_2)}{\|(p_4 - p_2) \times (p_3 - p_2)\|}$

$e = (p_3 - p_2) / \|p_3 - p_2\|$

Then the bending angle θ is s.t.

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

$\Rightarrow E(p) = \frac{1}{2} k C^2(p)$ where $C(p) = \theta(p) - \theta_0$ ← rest angle

Lecture 8

Deformable Models (Part 2)

Fundamentals of Computer Graphics

Animation & Simulation

Stanford CS248B, Fall 2023

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 $D(p) \simeq 0$ 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 $\frac{\partial D(p)}{\partial x} = \frac{D(p+h\hat{x})-D(p-h\hat{x})}{2\,h}$
  - The y-component is $\frac{\partial D(p)}{\partial y} = \frac{D(p+h\hat{y})-D(p-h\hat{y})}{2\,h}$
  - 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 ∇f

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

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

- The triangle stencil for ∇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.

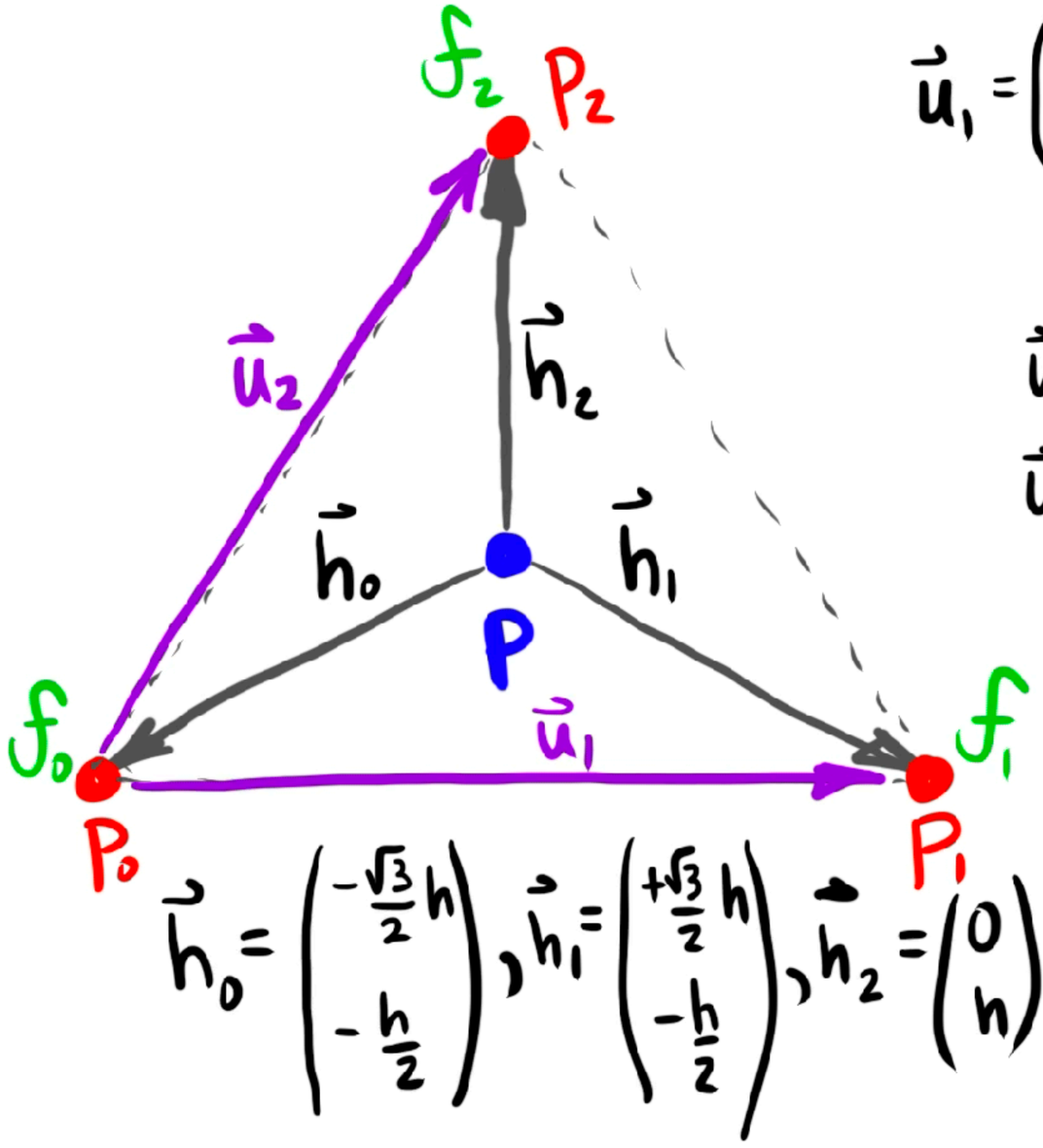
$$f(\mathbf{p}+\Delta\mathbf{p}) \approx f(\mathbf{p}) + \mathbf{g}^T\Delta\mathbf{p} = f(\mathbf{p}) + g_x\,\delta x + g_y\,\delta y$$

Rearranging

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

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

"Triangle trick" for 2D centered differences of ∇f



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

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

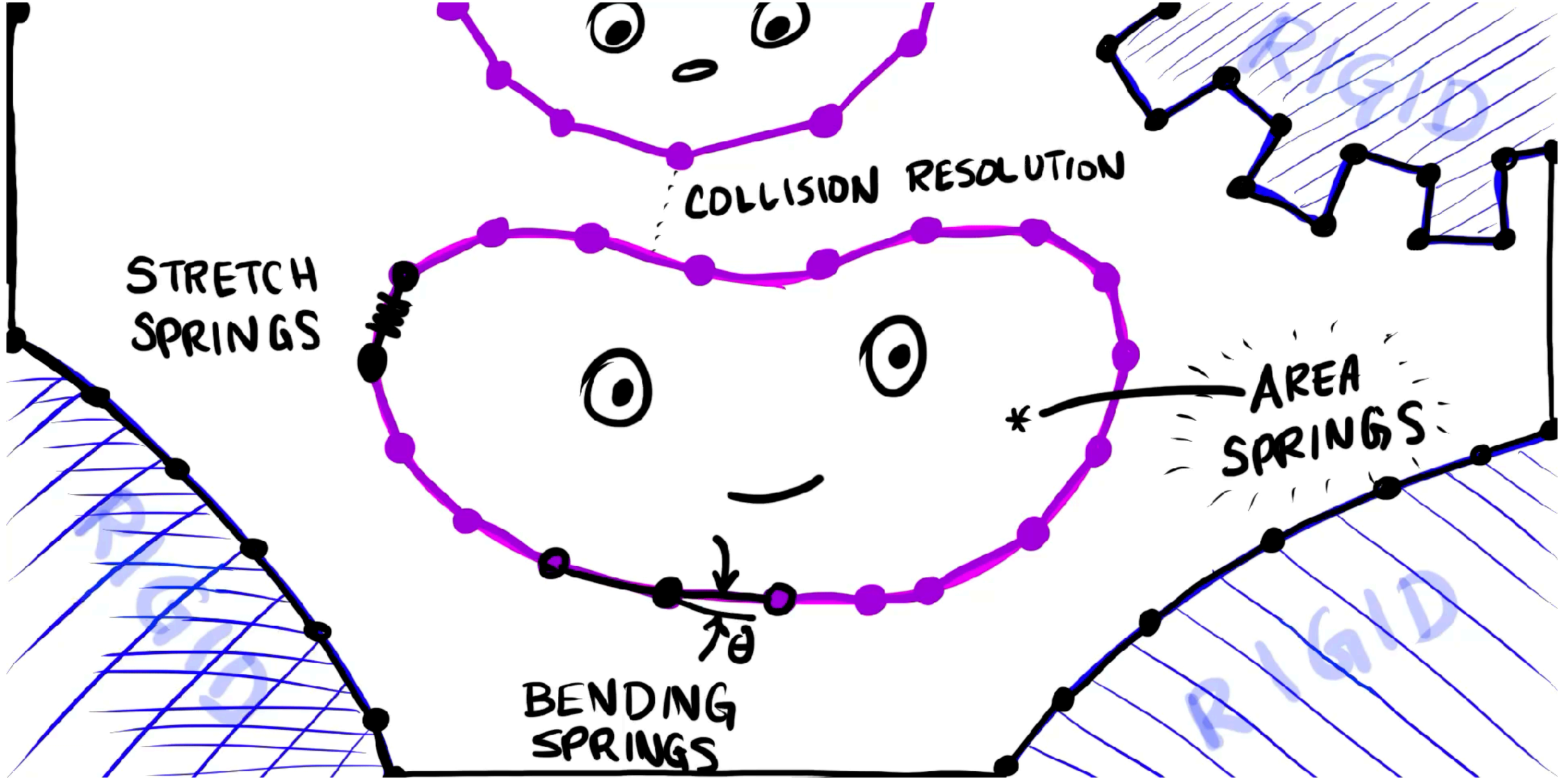
$$\Rightarrow \boxed{\vec{g} = U^{-1}\Delta f} = \begin{bmatrix} \sqrt{3}h & 0 \\ \frac{\sqrt{3}h}{2} & \frac{3h}{2} \end{bmatrix}^{-1} \Delta f$$

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

$$\vec{h}_0 = \begin{pmatrix} -\frac{\sqrt{3}}{2}h \\ -\frac{h}{2} \end{pmatrix}, \quad \vec{h}_1 = \begin{pmatrix} +\frac{\sqrt{3}}{2}h \\ -\frac{h}{2} \end{pmatrix}, \quad \vec{h}_2 = \begin{pmatrix} 0 \\ h \end{pmatrix}$$

Other force models

P2: Deformable blob simulation



Controlling Compression in 2D: Area springs

REST SHAPE
(UNDEFORMED)

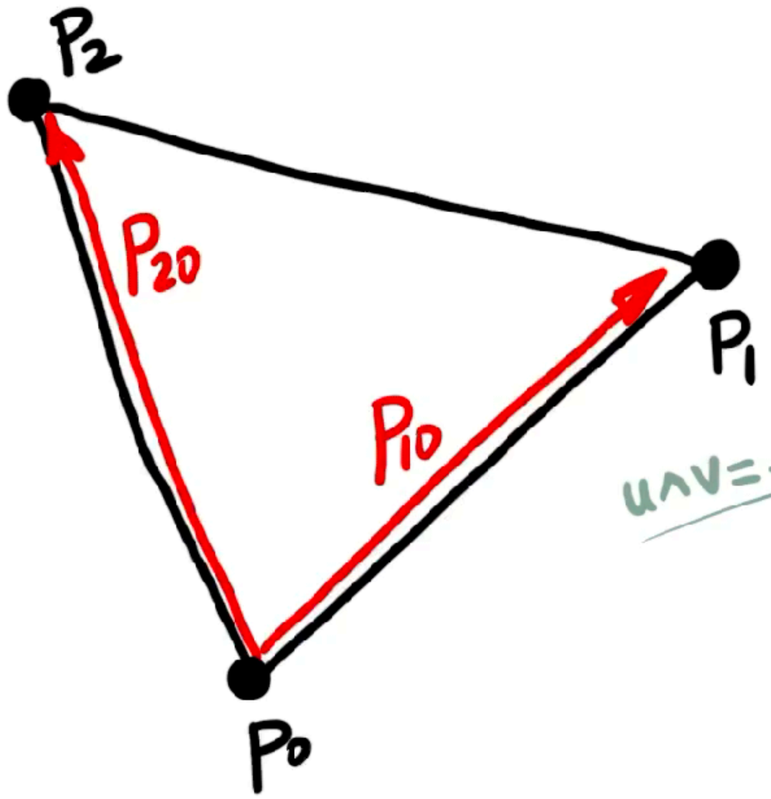
DEFORMED

POTENTIAL ENERGY: $E = \frac{1}{2} k \left(A(p) - A_0 \right)^2$

ELASTIC FORCE: $f_i = -k \left(A - A_0 \right) \partial_i A(p)$

Area of a 2D polygon

Consider the area of a triangle:



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

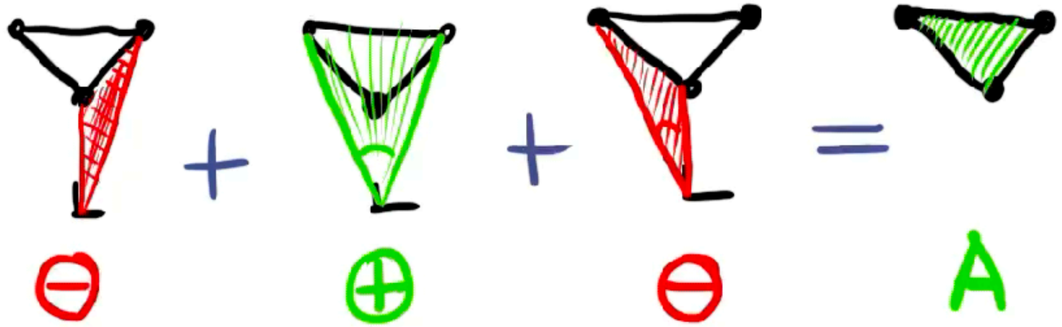
$$2A = p_{10} \wedge p_{20}$$

$$= (p_1 - p_0) \wedge (p_2 - p_0)$$

$$= p_1 \wedge p_2 - p_1 \wedge p_0 - p_0 \wedge p_2 + p_0 \wedge p_0 \quad (p_0 \wedge p_0 = 0)$$

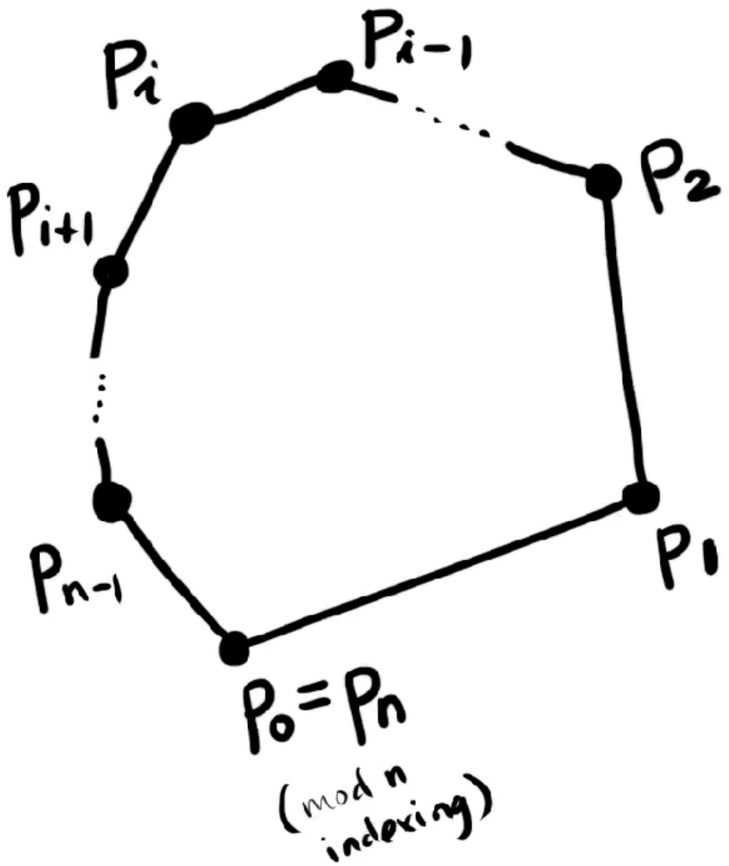
$$u \wedge v = -v \wedge u$$

$$= p_0 \wedge p_1 + p_1 \wedge p_2 + p_2 \wedge p_0$$



Area of a 2D polygon

Generalizing to a 2D polygon:



$$2A = \sum_{i=0}^{n-1} P_i \wedge P_{i+1}$$

$$= \ldots + P_{i-1} \wedge P_i + P_i \wedge P_{i+1} + \ldots$$

$$= \ldots + P_{i-1} \wedge P_i - P_{i+1} \wedge P_i + \ldots$$

$$= \ldots + (P_{i-1} - P_{i+1}) \wedge P_i + \ldots$$

$$\Longrightarrow \partial_i A = ?$$

Gradient of area of a 2D polygon

$$\partial_i A = \partial_i \left\{ \frac{(p_{i-1} - p_{i+1})}{2} \wedge p_i \right\}$$

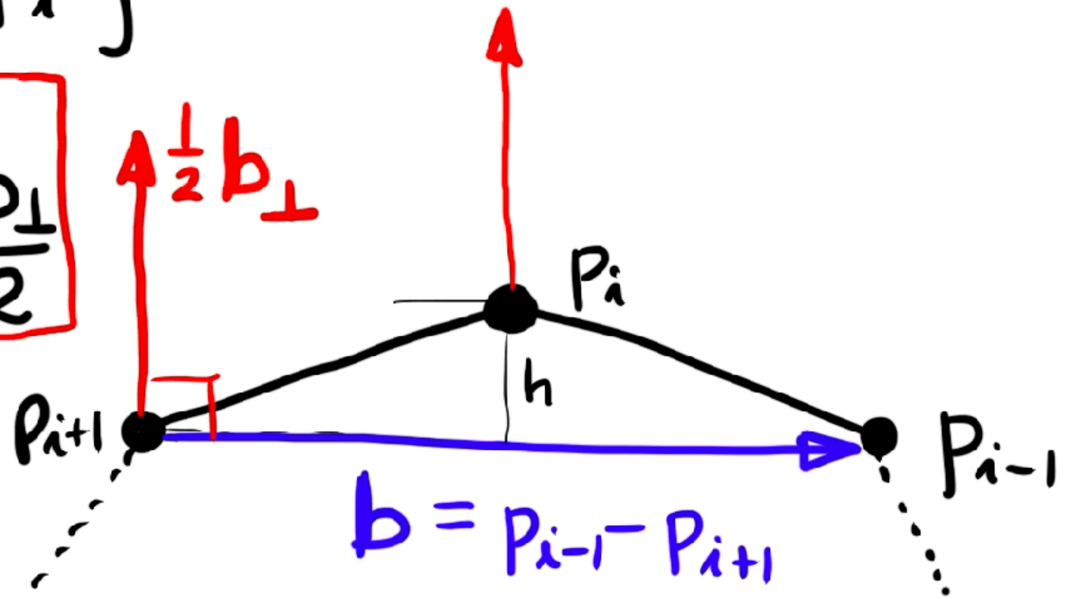
recall that $u \wedge v = u_\perp^T v = \begin{pmatrix} -u_y \\ u_x \end{pmatrix} \cdot \begin{pmatrix} v_x \\ v_y \end{pmatrix}$

$$= \partial_i \left\{ \frac{1}{2} (p_{i-1} - p_{i+1})_\perp^T \, p_i \right\}$$

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

Recall

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



Controlling Compression in 2D: Area springs

REST SHAPE
(UNDEFORMED)

DEFORMED

POTENTIAL ENERGY: $E = \frac{1}{2} k \left(A(p) - A_0 \right)^2$

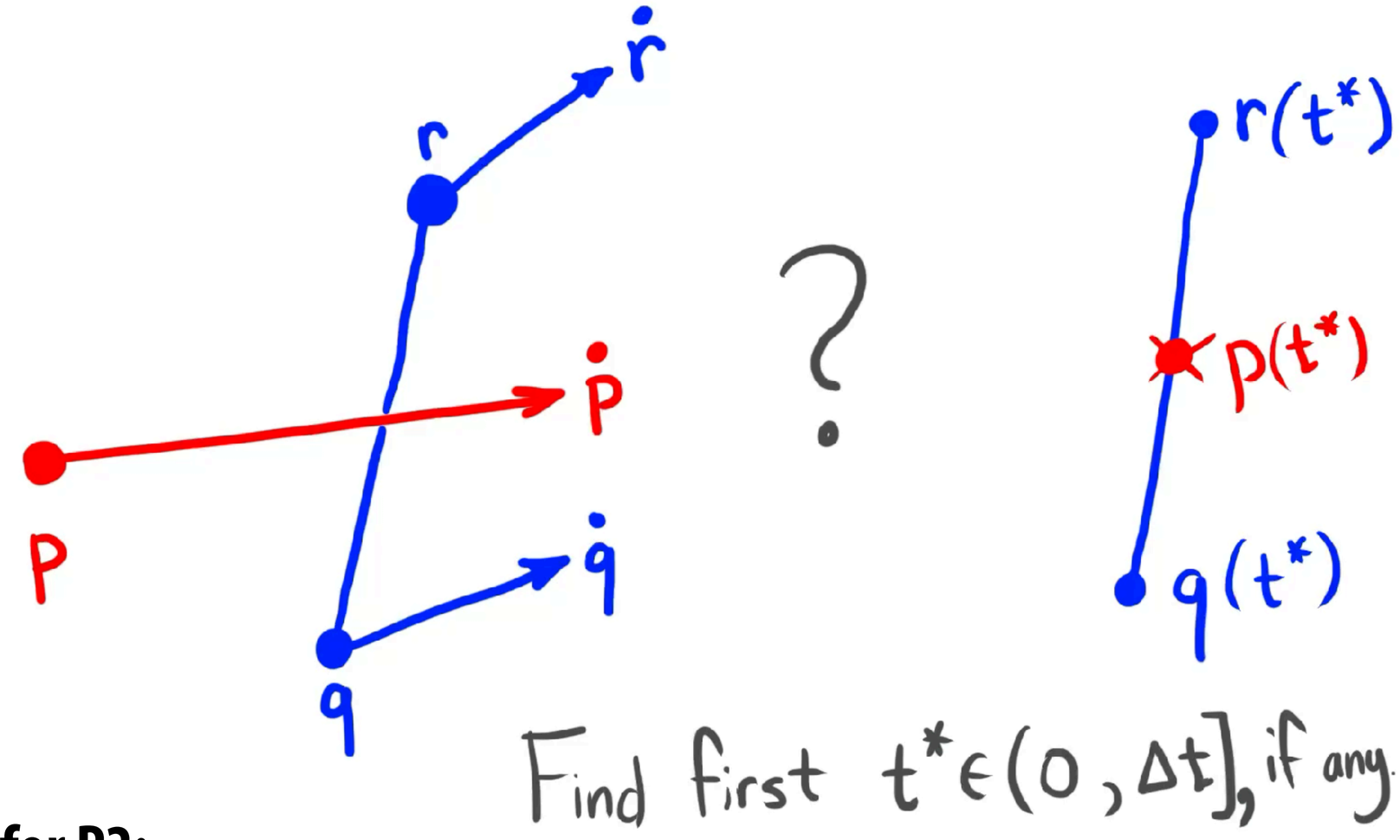
ELASTIC FORCE: $f_i = -k \left(A - A_0 \right) \partial_i A(p)$

Deformable collision detection

Time of collision (ToC)
Root-finding methods

RECALL:

CCD Test: Point-Edge (2D)

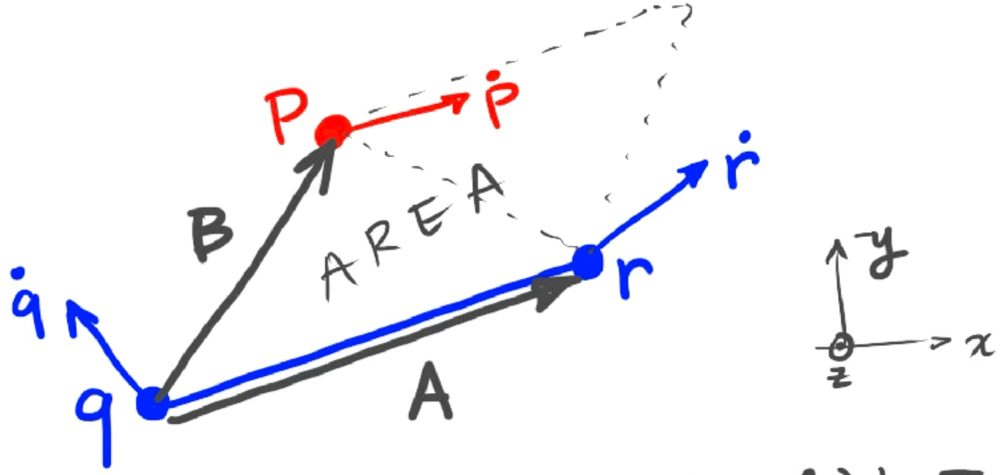


- **Approach for P2:**
  - **Find time when area of pt-edge triangle is zero, then check if pt on edge.**

RECALL:

CCD Test: Point-Edge (2D)

Collinearity test



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

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

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

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

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

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., $|a| \to 0$)

Robust solution:

(ref. Numerical Recipes in C)

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

if $(D > 0)$

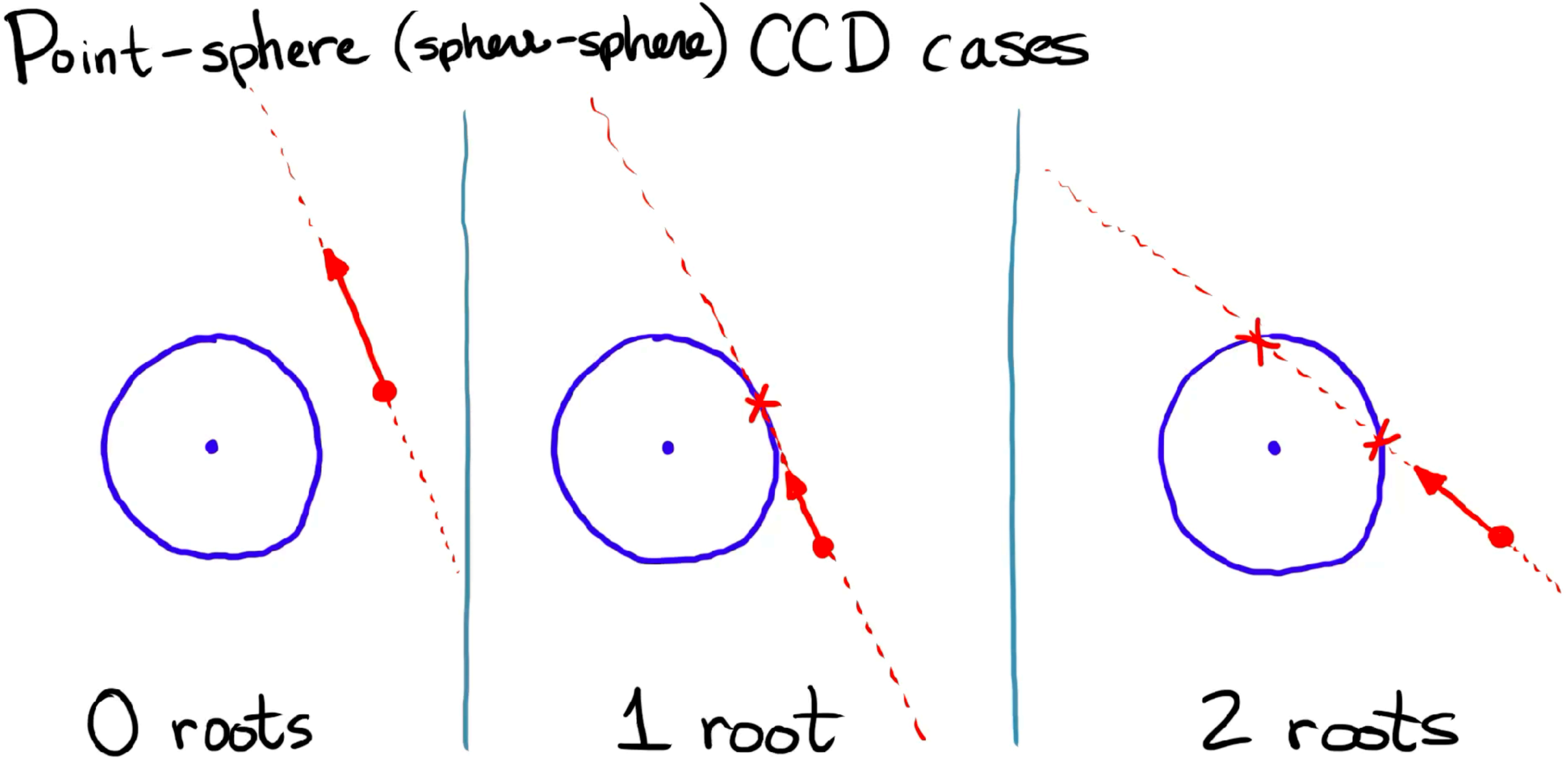
$r = -\frac{1}{2}\left(b + \text{sgn}(b)\sqrt{D}\right)$

$t_1 = r/a$

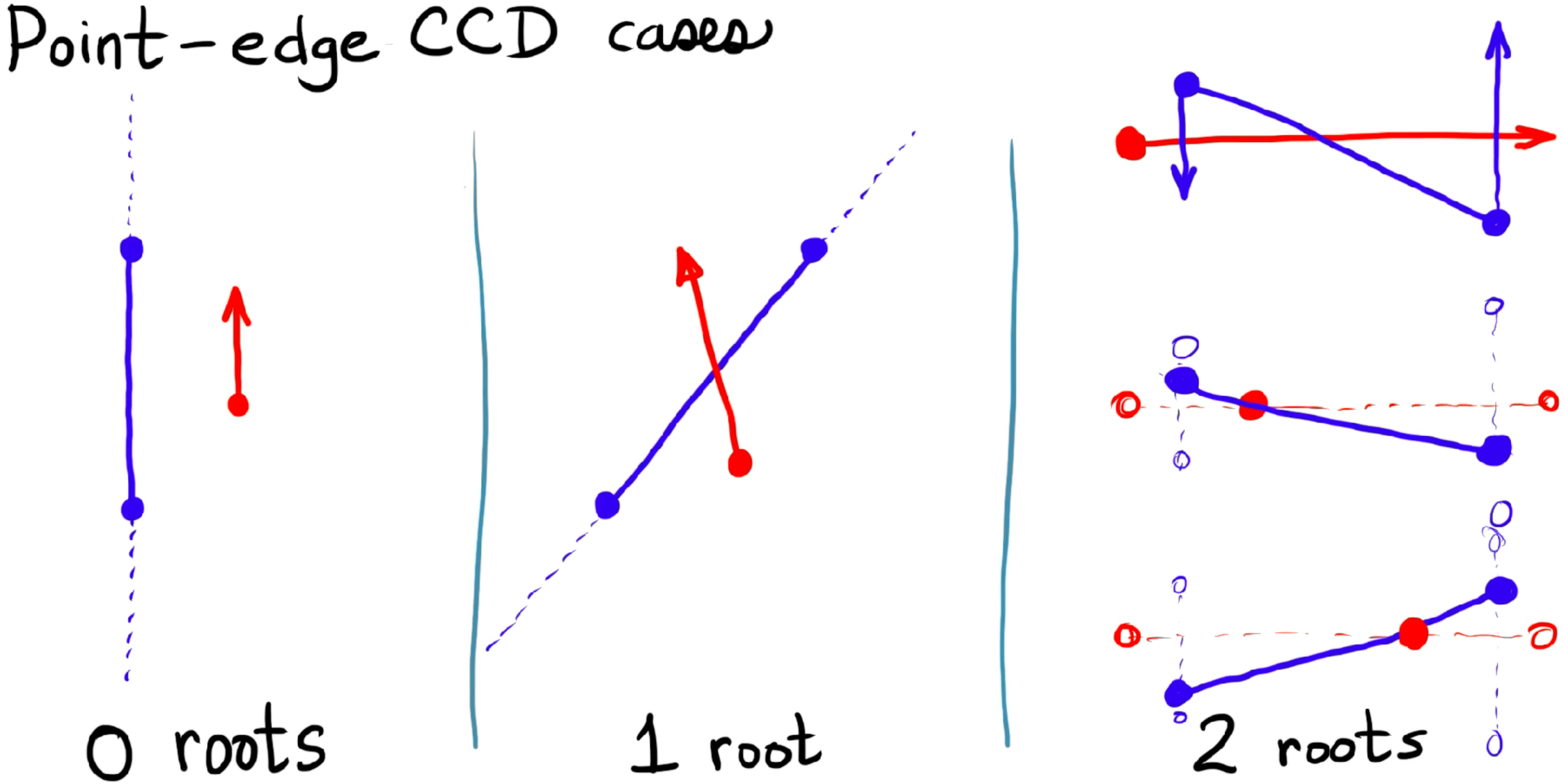
$t_2 = c/r$

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

Geometric interpretation of number of real roots



Geometric interpretation of number of real roots



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

IMPORTANT CCD TESTS FOR CLOTH & HAIR

Linear motion

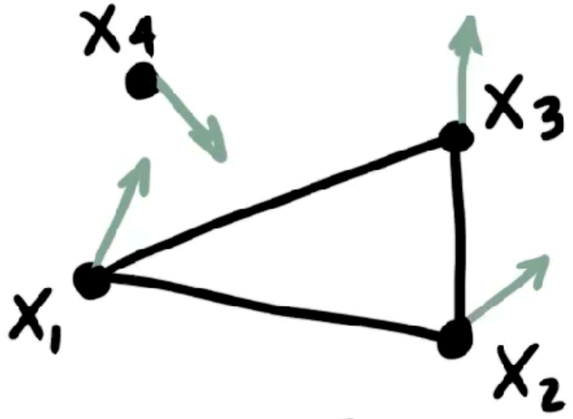
$$x_i + t v_i$$

Let

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

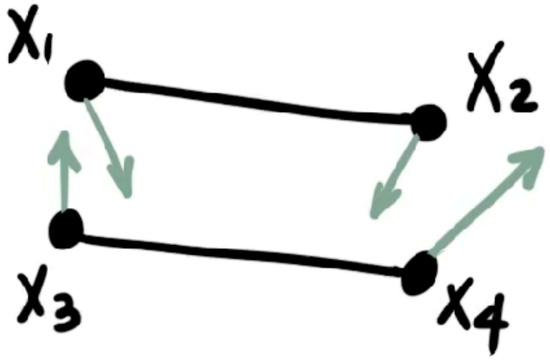
$$v_{ij} \equiv v_i - v_j$$

COPLANARITY TEST



POINT-TRIANGLE

OR



POINT-EDGE

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

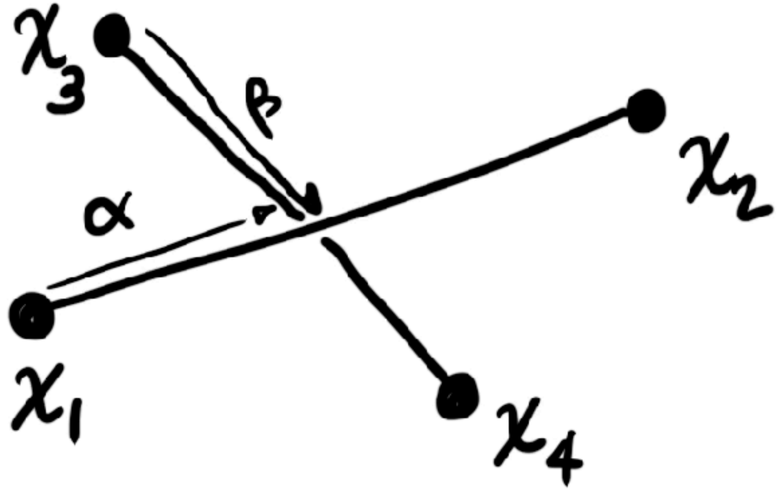
$\Longrightarrow$

- 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



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

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

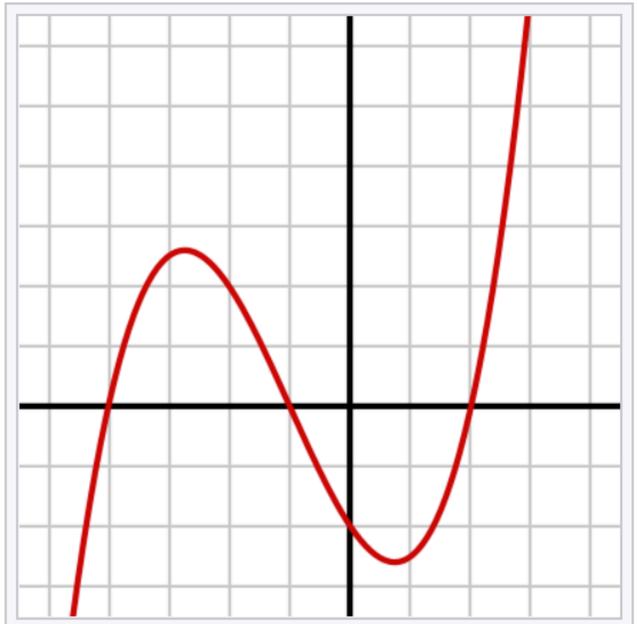
Solve in a least-squares sense: $Ax = b \longrightarrow A^T A\, x = A^T b$:

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

RECALL:

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

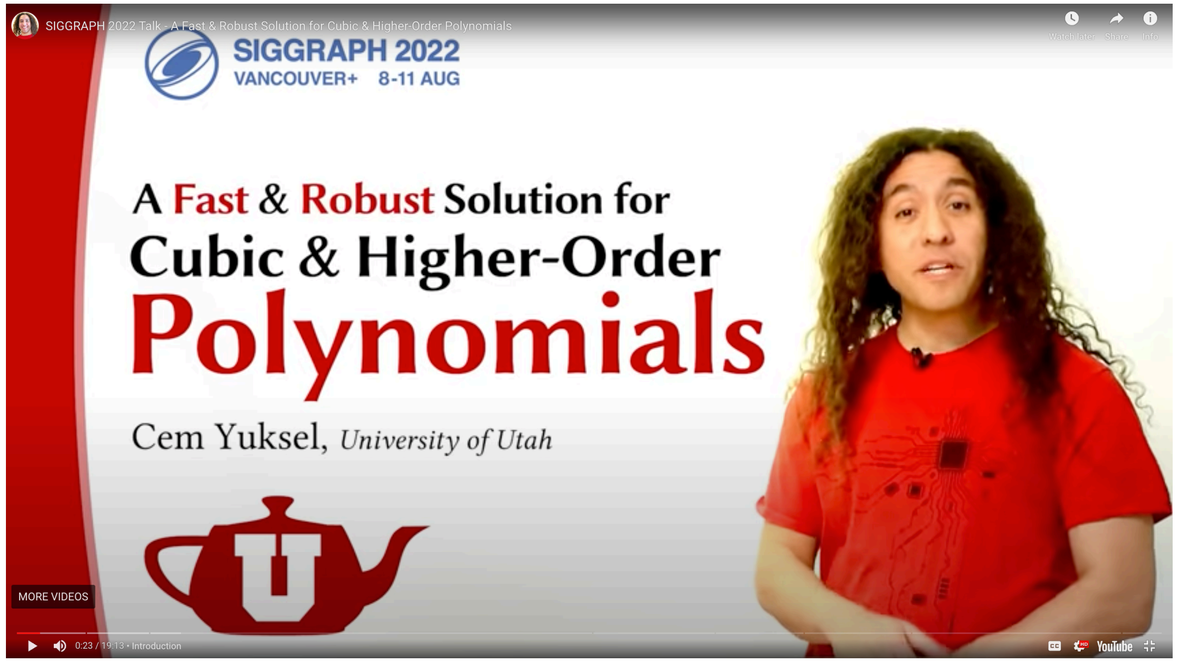
- Must find the roots of polynomial equations to find the ToC
- Can be numerically tricky for arbitrary computer-generated polynomials from animation
- Cloth animation can generate billions of CCD tests.
  - Many cubic polynomials with ill-conditioned coefficients
- One-in-a-billion bugs are a show stopper.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

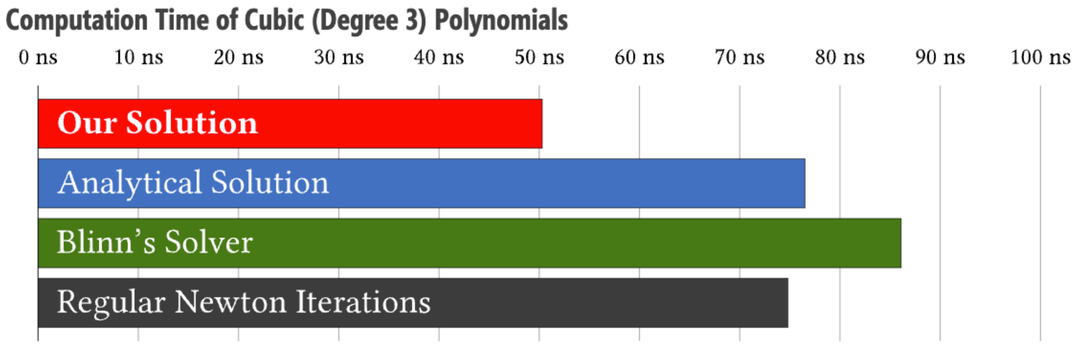


Graph of a cubic function with 3 real roots (where the curve crosses the horizontal axis at $y = 0$). The case shown has two critical points. Here the function is $f(x) = (x^3 + 3x^2 - 6x - 8)/4$.

Robust root finding: Polynomial equations

- See this recent and clear talk by Cem Yuksel





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

Lecture 8

Deformable Models (Part 3)

Fundamentals of Computer Graphics

Animation & Simulation

Stanford CS248B, Fall 2023

Profs. Karen Liu & Doug James

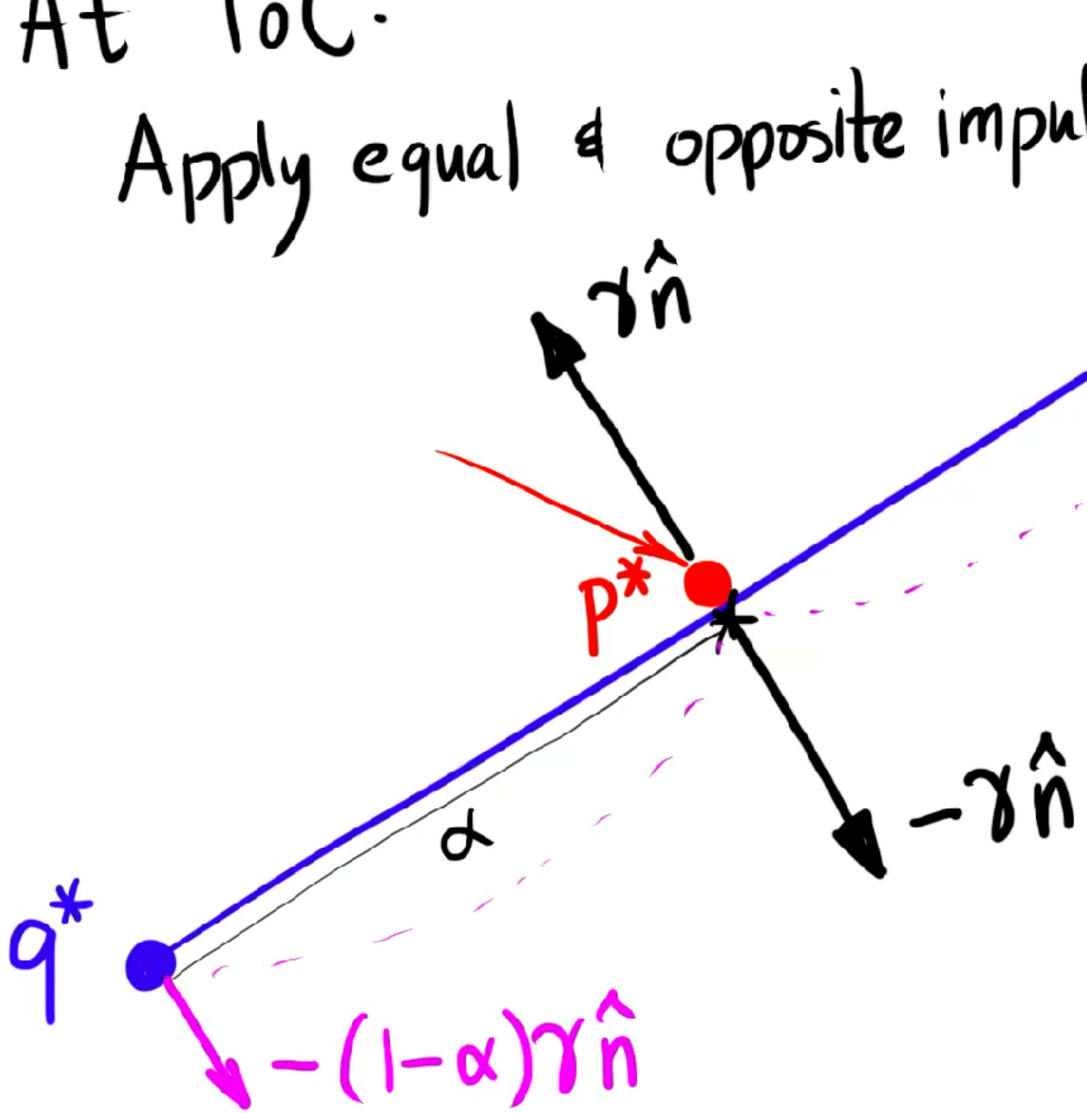
Deformable collision resolution

Computing impulses and penalty forces

2D point-edge collision: Impulse calculation

At ToC:

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



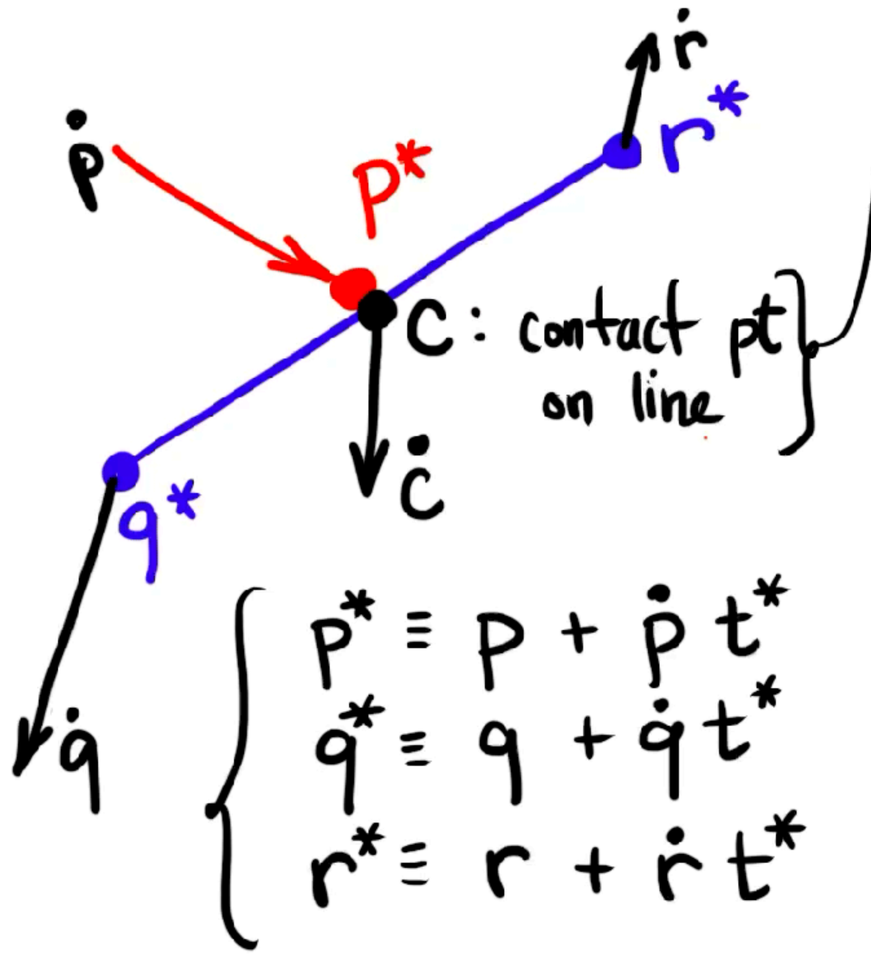
- Distribute impulse on edge to vertices using barycentric weighting.
- Set γ so that

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

2D point-edge collision: Normal calculation

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

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



$$\begin{cases} p^* \equiv p + \dot{p}\, t^* \\ q^* \equiv q + \dot{q}\, t^* \\ r^* \equiv r + \dot{r}\, t^* \end{cases}$$

$$c = q^* + \alpha(r^* - q^*) \quad \text{for } \alpha \in [0,1].$$

$$\Rightarrow \dot{c} = \dot{q} + \alpha(\dot{r} - \dot{q}) \quad \text{(contact pt velocity)}$$

Relative contact velocity: $v = \dot{p} - \dot{c}$

Candidate contact normal: $n_0 = \overrightarrow{qr}_\perp = \begin{pmatrix} q_y - r_y \\ r_x - q_x \end{pmatrix}$

Set normal sign s.t. $v_n < 0$:

$$\hat{n} = \hat{n}_0 \cdot \mathrm{sgn}(-v \cdot \hat{n}_0)$$

2D point-edge collision: Impulse calculation

Post-impulse particle velocities

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

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

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

$$\dot{c}^+ = \dot{c}^- + \Delta\dot{c} = (1-\alpha)\dot{q}^+ + \alpha\dot{r}^+$$

$$= \left(\bar{\alpha}\dot{q}^- + \alpha\dot{r}^-\right) + \underbrace{\left(\bar{\alpha}\Delta\dot{q} + \alpha\Delta\dot{r}\right)}_{\Delta\dot{c}}$$

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

2D point-edge collision: Impulse calculation

SOLVE FOR γ USING RESTITUTION HYPOTHESIS:

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

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

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

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

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

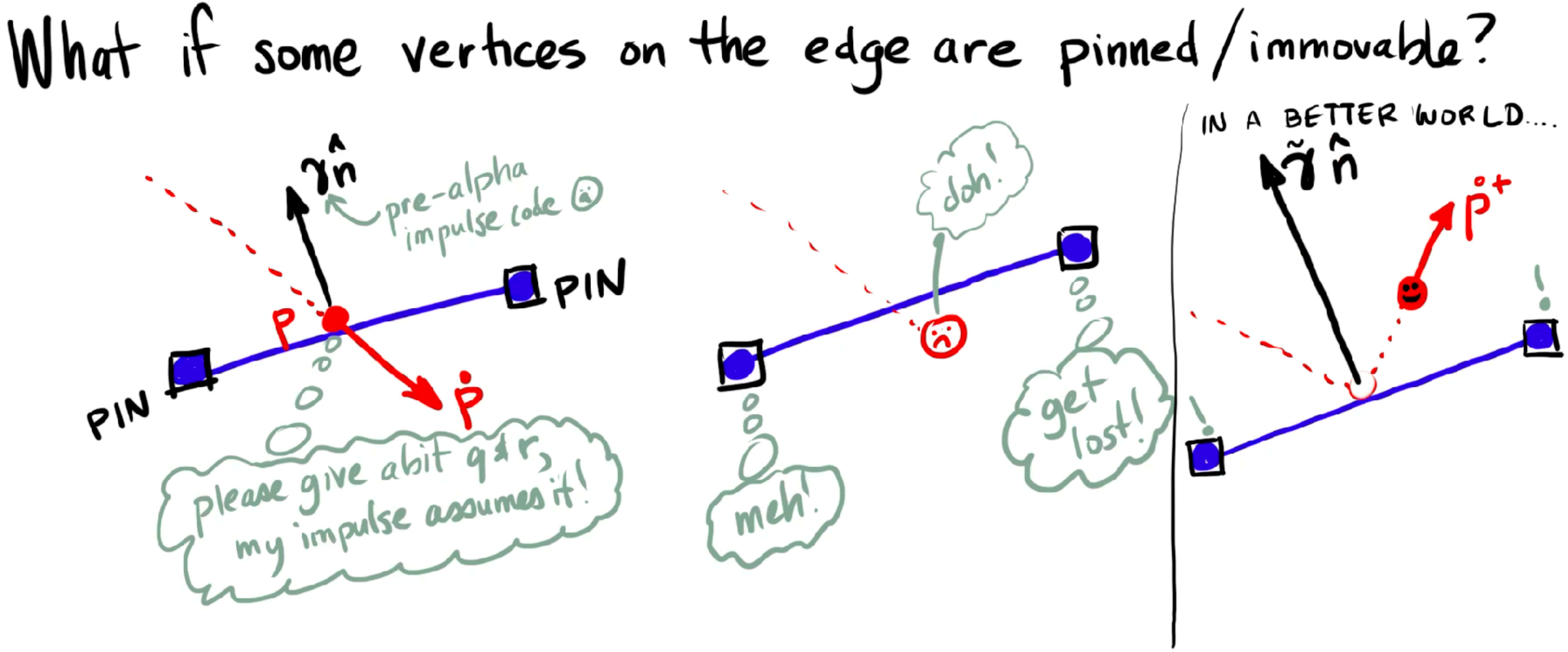
$$\Rightarrow \gamma = \frac{-(1+\varepsilon)\,v_n^-}{\frac{1}{m_p} + \frac{\bar{\alpha}^2}{m_q} + \frac{\alpha^2}{m_r}} \longrightarrow \boxed{\gamma = (1+\varepsilon)\,m_{eff}\,(-v_n^-)} \geq 0$$

$\sim m_{eff}^{-1}$

Inverse-mass filtering

Dealing with friends who ~~think they~~ have infinite mass

Impulses with pinned particles



Inverse-mass filtering (a really useful trick)

Use infinite mass for pinned/immovable particles:

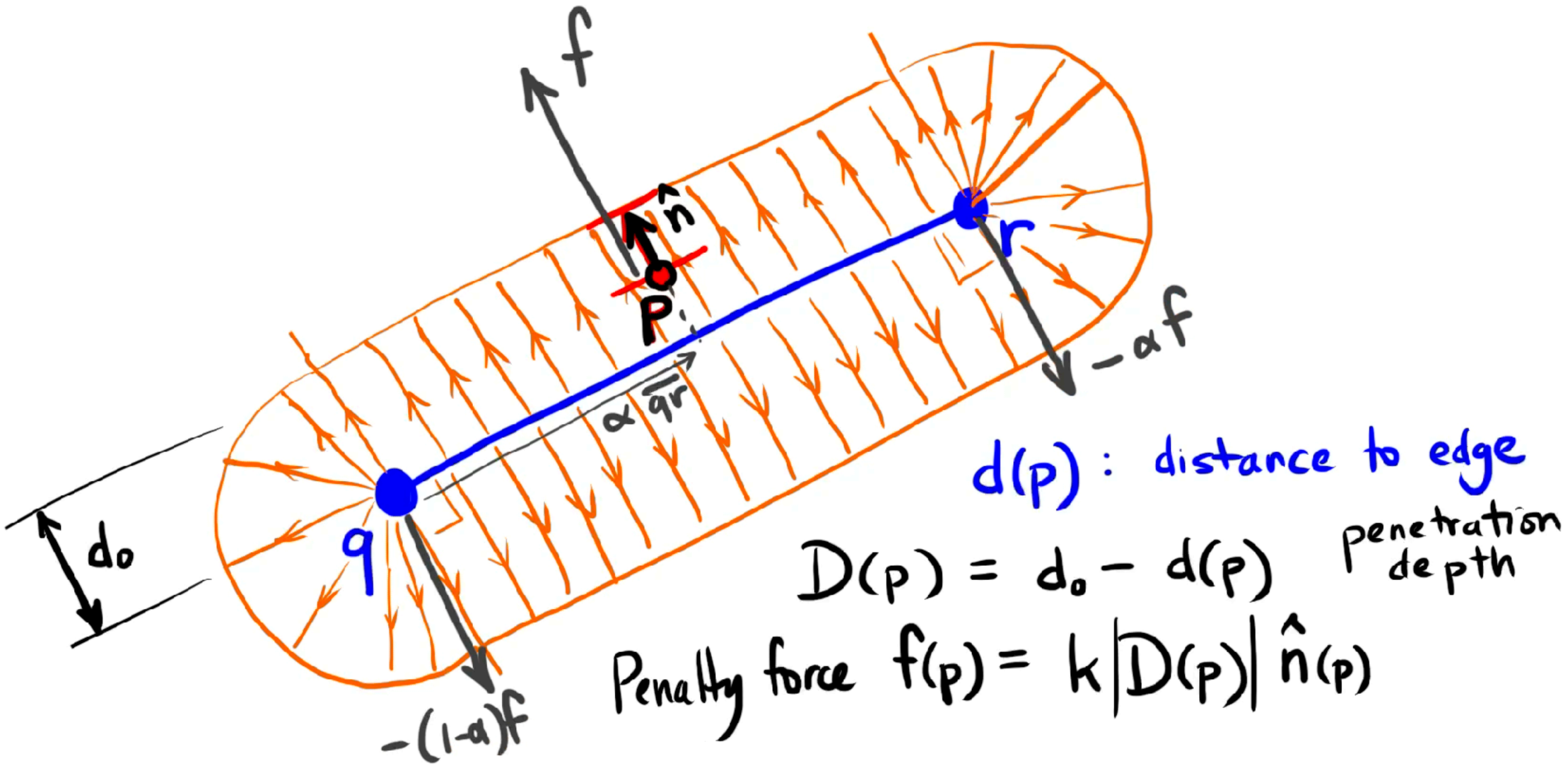
INVERSE MASS: $w_i = \begin{cases} 0 & \text{, if particle "i" doesn't want to play today,} \\ \frac{1}{m_i} & \text{, otherwise} \end{cases}$

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

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

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

Penalty forces: 2D point-edge collisions



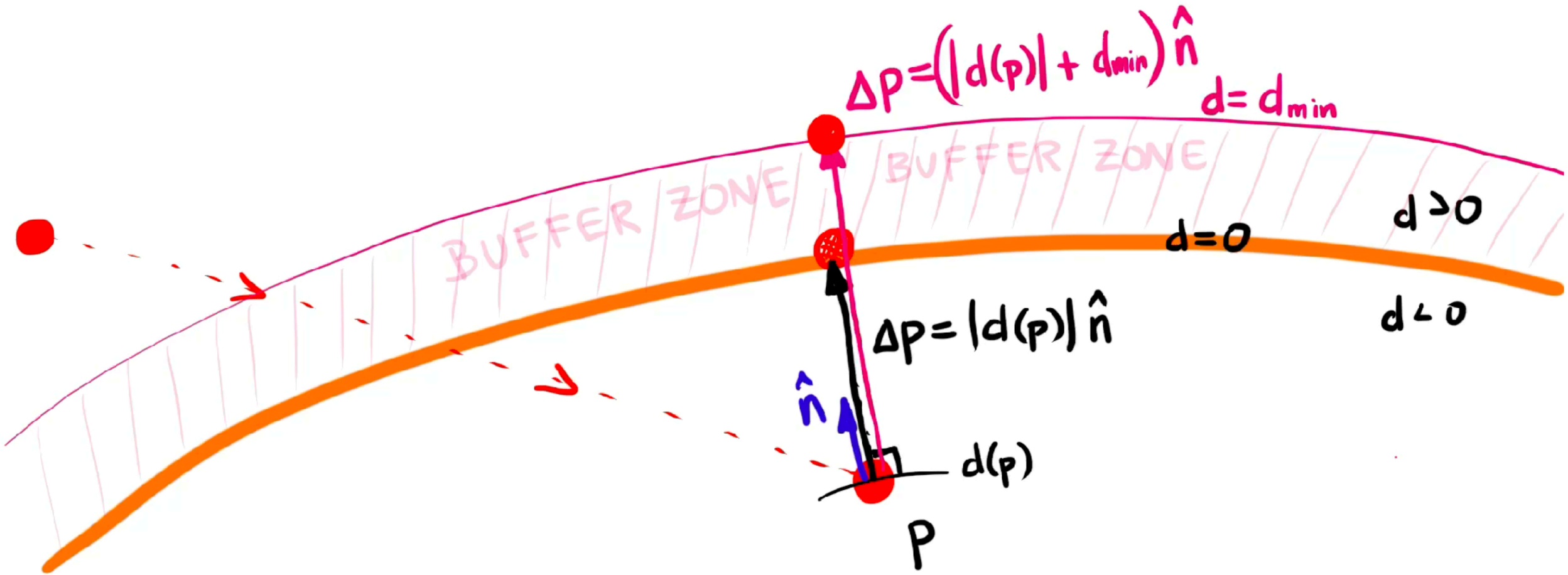
$d(p)$: distance to edge

$D(p) = d_0 - d(p)$ penetration depth

Penalty force $f(p) = k|D(p)|\,\hat{n}(p)$

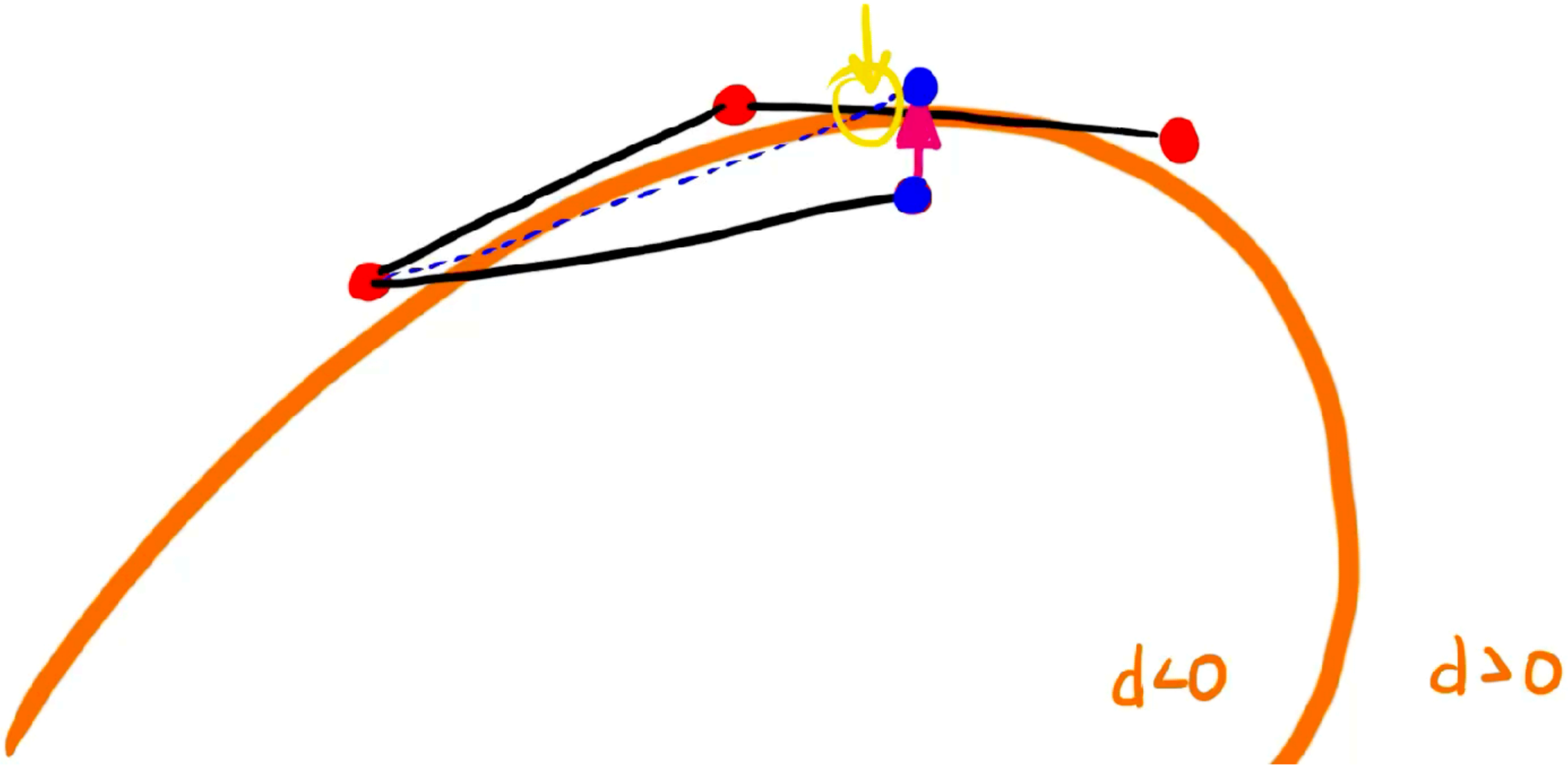
Removing interpenetration with SDFs

- **Position-level correction: takes** $d(\mathbf{p}) < 0$ **to** $d(\mathbf{p} + \Delta\mathbf{p}) = d_{min} > 0$



Removing interpenetration with SDFs

- Warning: Position-level corrections can introduce other interpenetrations
- Example: Edge-edge overlaps introduced by particle position correction



Iterative collision resolution with impulses

Recall: Symplectic Euler integrator with filters

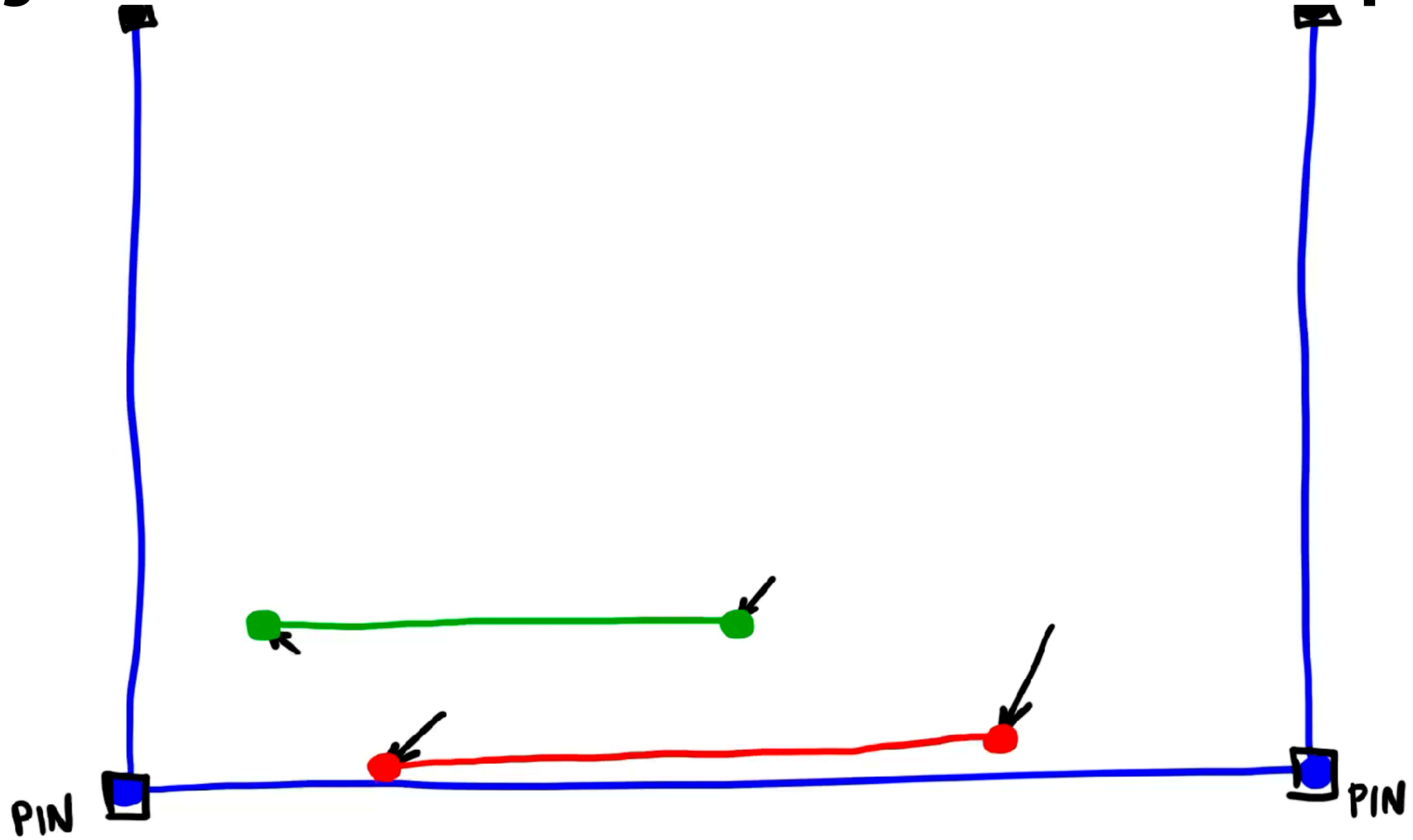
1. **Accumulate forces:** $\mathbf{f}$ (springs, gravity, drag, etc.)
2. **Evaluate accelerations:** $\mathbf{a} = \mathbf{M}^{-1}\mathbf{f}$
   - Optional: Inverse mass filtering for pinned particles
3. **Timestep velocities:** $\mathbf{v} \mathrel{+}= \Delta t\ \mathbf{a}$
   - Optional: Filter velocities for collisions and constraints
4. **Timestep positions:** $\mathbf{p} \mathrel{+}= \Delta t\ \mathbf{v}$
   - Optional: Filter positions

Integrator with iterative collision resolution w/ impulses

- Velocity filter for collision impulses
  - Freeze particle positions
  - Apply impulses until no collisions detected (CCD) on timestep

```
C = detectPtLineCollisions();
While(|C| > 0) {
  applyCollisionImpulses(C);     // modify v only
  C = detectPtLineCollisions(); // update C
}
```

Integrator with iterative collision resolution w/ impulses



Integrator with iterative collision resolution w/ impulses

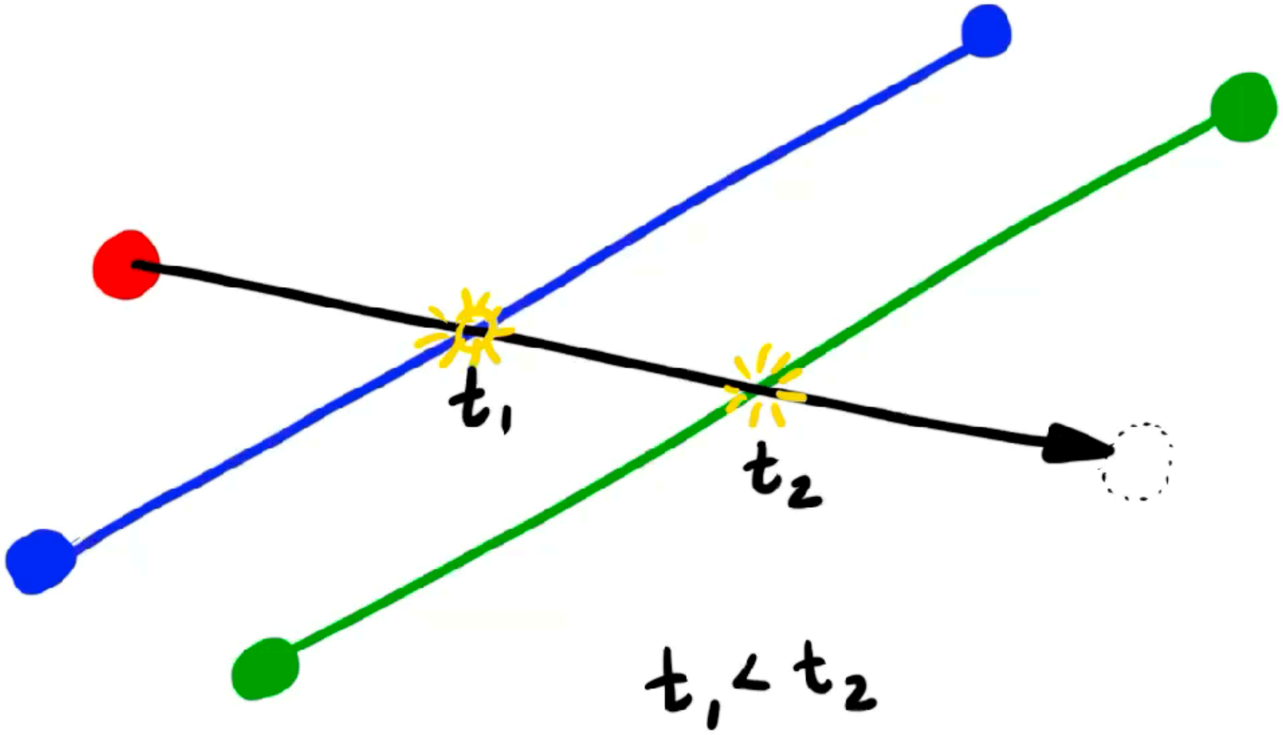
- Velocity filter for collision impulses
  - Freeze particle positions
  - Apply impulses until no collisions detected (CCD) on timestep

```
C = detectPtLineCollisions();
While(|C| > 0) {
  applyCollisionImpulses(C);     // modify v only
  C = detectPtLineCollisions(); // update C
}
```

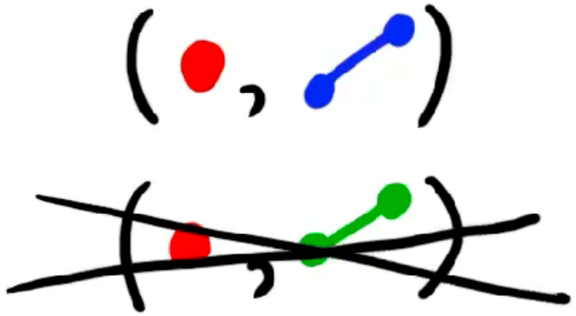
- Issues:
  - Convergence can be slow, so use impulses as a last resort.
    - Lower coefficients of restitution ensure energy loss and faster convergence.
  - No conservative advancement, so possible out-of-order collision resolution.

Integrator with iterative collision resolution w/ impulses

Temporal order matters



OVERLAPS



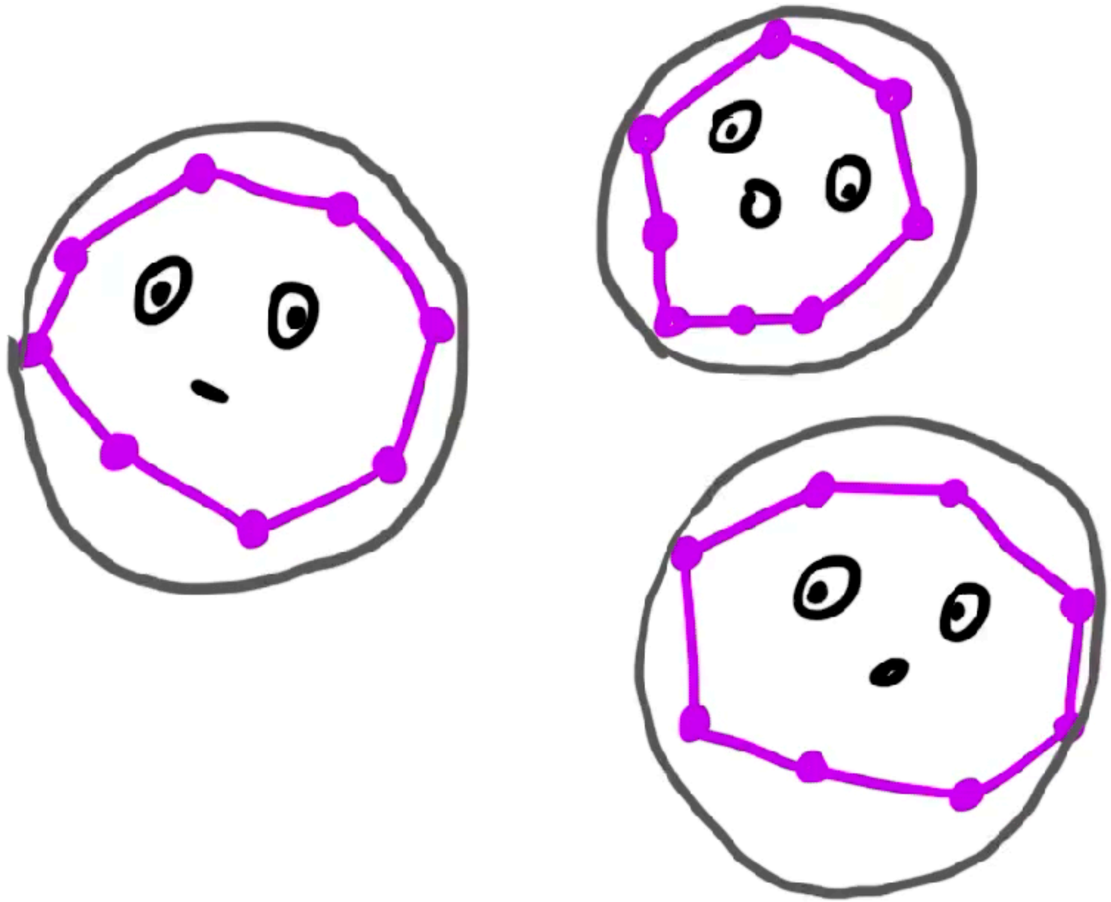
Process early collision (t_1) first. Why?

Point-edge broad-phase CD

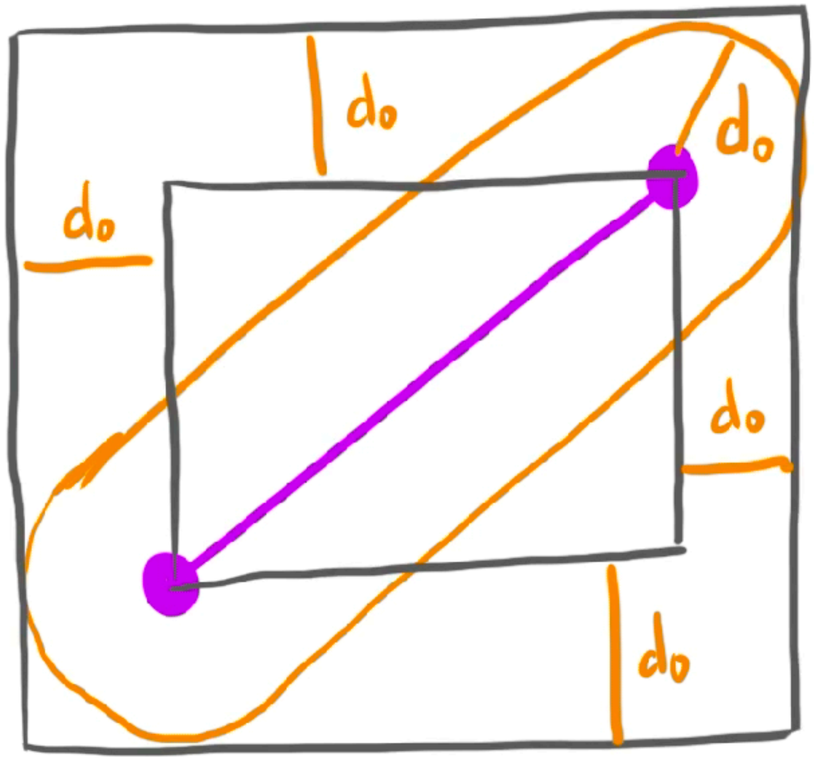
Broad phase CD for point-edge collisions

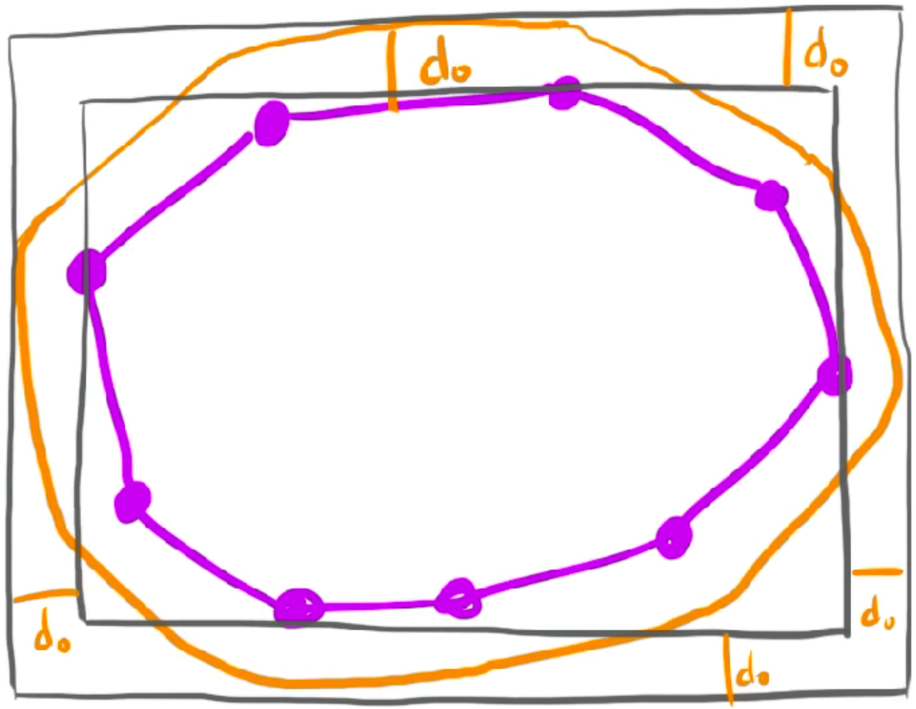
- Two broad-phase problems:
  1. Discrete proximity for penalty forces
  2. Continuous CD for collision impulses
- Not that many blobs
- Bounding volumes effective here

Blob bounding volumes: AABBs or Disks

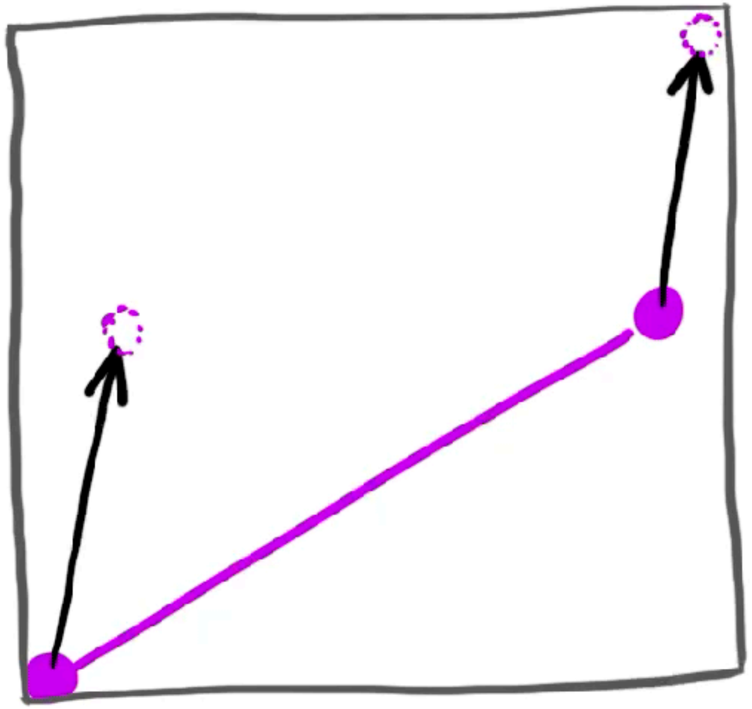


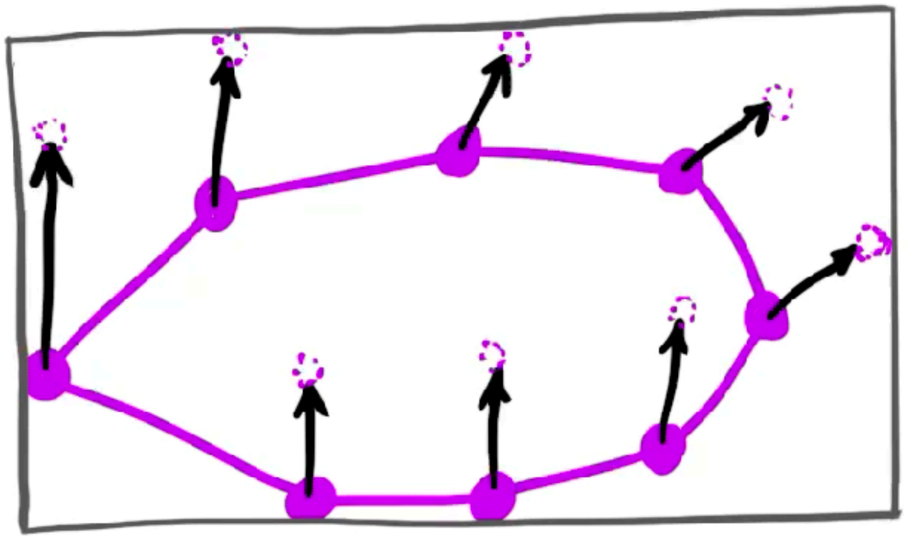
Bounding blobs: Expand for point-edge penalty forces





Bounding blobs: Expand for point-edge CCD





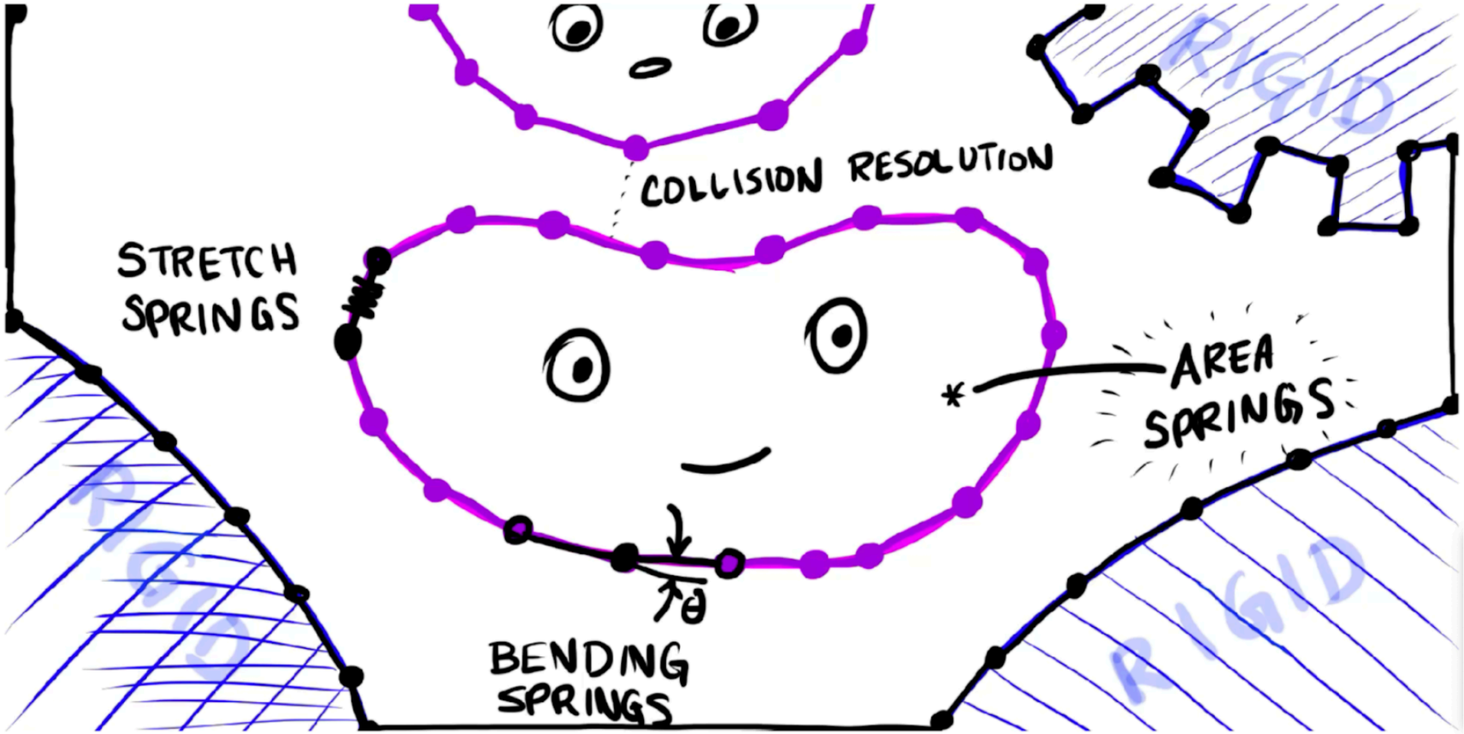
Stanford CS 248B Fall 2023 – Programming Assignment #2

Attack of the Blobs!

Deformable Collision Processing

Summary

In your second programming assignment you will simulate 2D deformable “blobs” and process collisions between them and with a rigid environment. Warning: There are going to be a LOT of blobs–see the title. So you will need to make your simulation not too slow (it needn’t run in real time), and also robust so that you don’t miss any collisions. Along the way, you will model and animate the dynamics of some lovely little deforming blobs filling up a sterile world (a.k.a. “Attack of the Blobs!”).



Implicit integration

Important for cloth animation

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, unbanded 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

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

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 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.[1] 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 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.[2] 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 quartic 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

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.

[1] Additional use of implicit methods in animation and dynamics work includes Kass and Miller [8], Terzopoulos and Qin [18], and Tu [19].

[2] Even worse, the number of time steps per frame tends to increase along with the problem size, for an explicit method. Cloth simulations of size n—meaning $\mathbf{x} \in \mathbb{R}^{O(n)}$—generally require $O(n)$ explicit steps per unit simulated time. Because the cost of an explicit step is also $O(n)$ (setting aside complications such as collision detection for now) explicit methods for cloth require time $O(n^2)$—or worse.

SIGGRAPH 98, Orlando, July 19–24 COMPUTER GRAPHICS Proceedings, Annual Conference Series, 1998

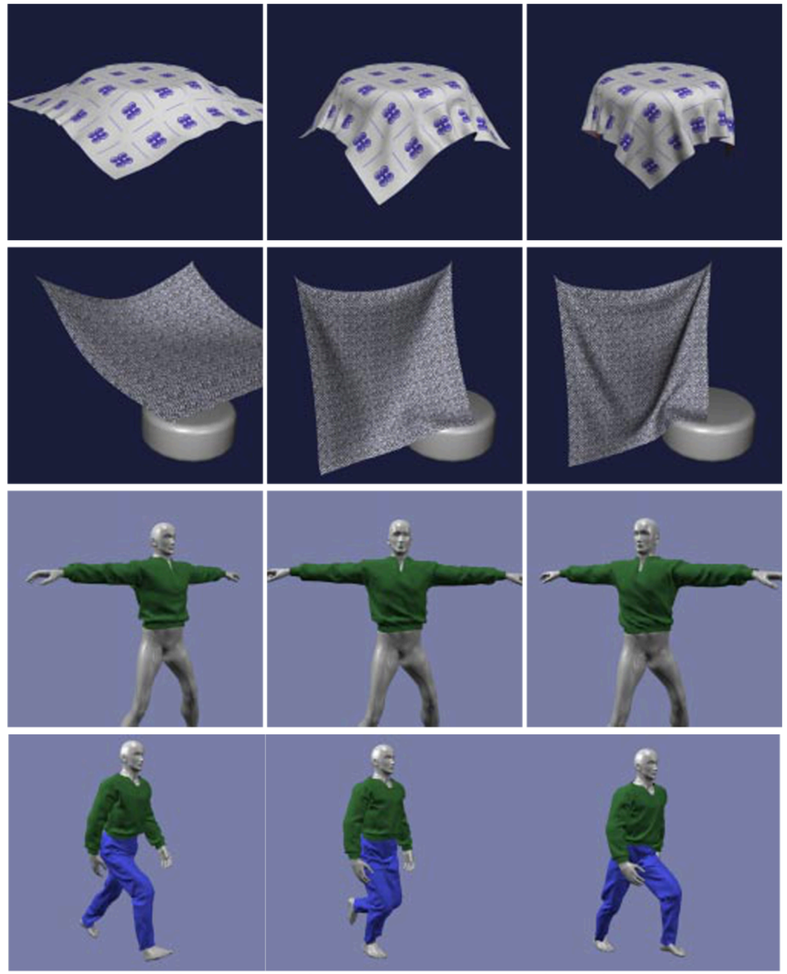


Figure 1 (top row): Cloth draping on cylinder; frames 8, 13 and 35. Figure 2 (second row): Sheet with two fixed particles; frames 10, 29 and 67. Figure 3 (third row): Shirt on twisting figure; frames 1, 24 and 46. Figure 4 (bottom row): Walking man; frames 30, 45 and 58.

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.

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

EQUATIONS OF MOTION:

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

Assume that $\mathbf{f} = \mathbf{f}(\mathbf{p}, \mathbf{v})$.

Backward Euler update:

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

Linearize $\mathbf{f}$ about current $(\mathbf{p},\mathbf{v})$ configuration, $(\mathbf{p}_0, \mathbf{v}_0)$:

$$\mathbf{f}(\mathbf{p}_0+\Delta\mathbf{p}, \mathbf{v}_0+\Delta\mathbf{v}) \approx \mathbf{f}(\mathbf{p}_0,\mathbf{v}_0) + \left.\frac{\partial \mathbf{f}}{\partial \mathbf{p}}\right|_{\substack{\mathbf{p}=\mathbf{p}_0 \\ \mathbf{v}=\mathbf{v}_0}} \Delta\mathbf{p} + \left.\frac{\partial \mathbf{f}}{\partial \mathbf{v}}\right|_{\substack{\mathbf{p}=\mathbf{p}_0 \\ \mathbf{v}=\mathbf{v}_0}} \Delta\mathbf{v}$$

$$\equiv \mathbf{f}_0 + \frac{\partial \mathbf{f}}{\partial \mathbf{p}}_0 \Delta\mathbf{p} + \frac{\partial \mathbf{f}}{\partial \mathbf{v}}_0 \Delta\mathbf{v}$$

PIXAR

FEATURE FILMS SHORT FILMS TECHNOLOGY CAREERS EXTRAS ABOUT

AWWW...DON'T CRY.

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

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 1st row $\Delta p = h(v_0 + \Delta v)$ into 2nd 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 Δv:

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

TYPICALLY A LARGE, SPARSE, SYMMETRIC MATRIX.
USUALLY POSITIVE DEFINITE IF h SMALL ENOUGH ☺

$$A\,\Delta v = b \xrightarrow{\text{solve}} \Delta v \longrightarrow \Delta p = h(v_0 + \Delta v)$$

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

- **Backward Differentiation Formula (BDF) methods**
  - **e.g., BDF2, second-order accurate scheme**

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

- **Trapezoidal rule, second-order accurate scheme**

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

- **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

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

For particle systems,

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

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

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

← SLIGHTLY DIFFERENT FROM BACKWARD EULER, & SECOND-ORDER ACCURATE (*)