Lecture 8

# Deformable Models (Part 3)

---

Fundamentals of Computer Graphics

## Animation & Simulation

Stanford CS248B, Fall 2022

Profs. Karen Liu & Doug James

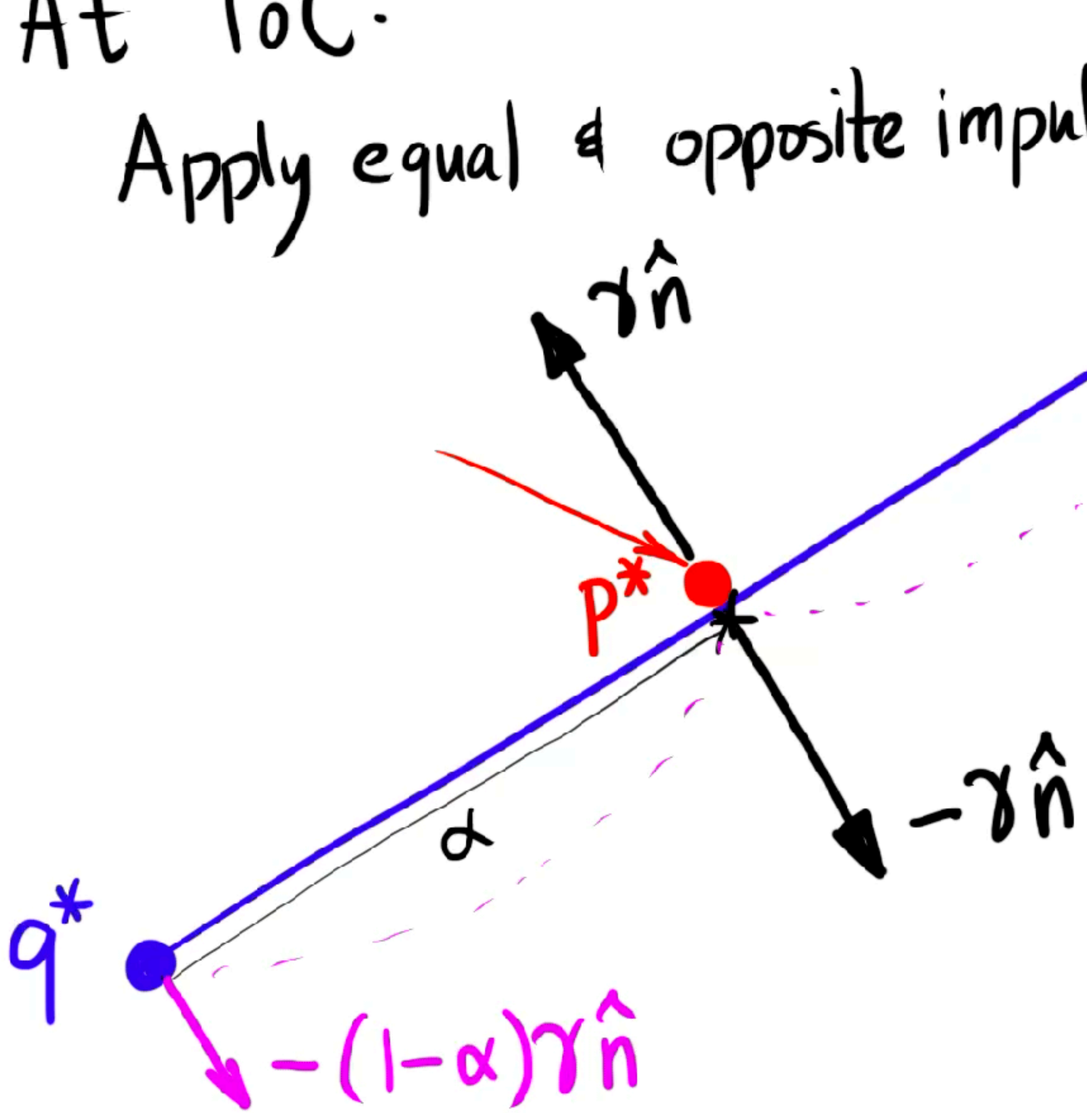
# Deformable collision resolution

## Computing impulses and penalty forces

# 2D point-edge collision: Impulse calculation

At ToC:

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



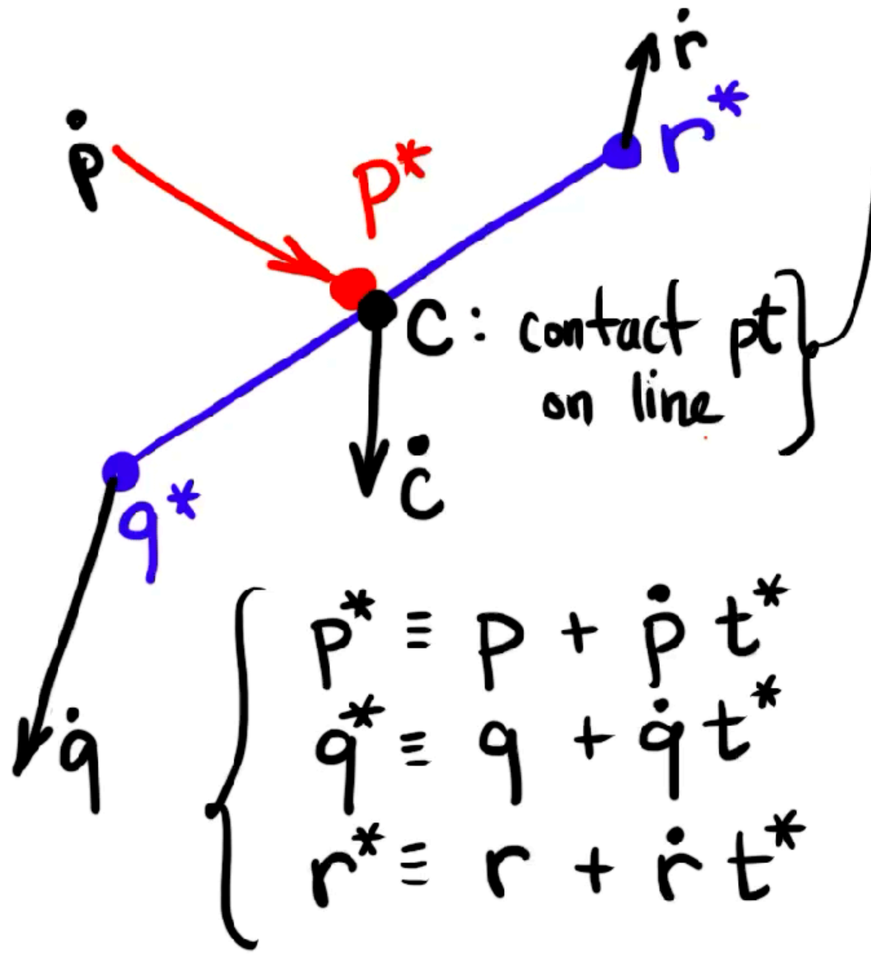

- Distribute impulse on edge to vertices using barycentric weighting.
- Set $\gamma$ so that

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

# 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 $\gamma$ USING RESTITUTION HYPOTHESIS:

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

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

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

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

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

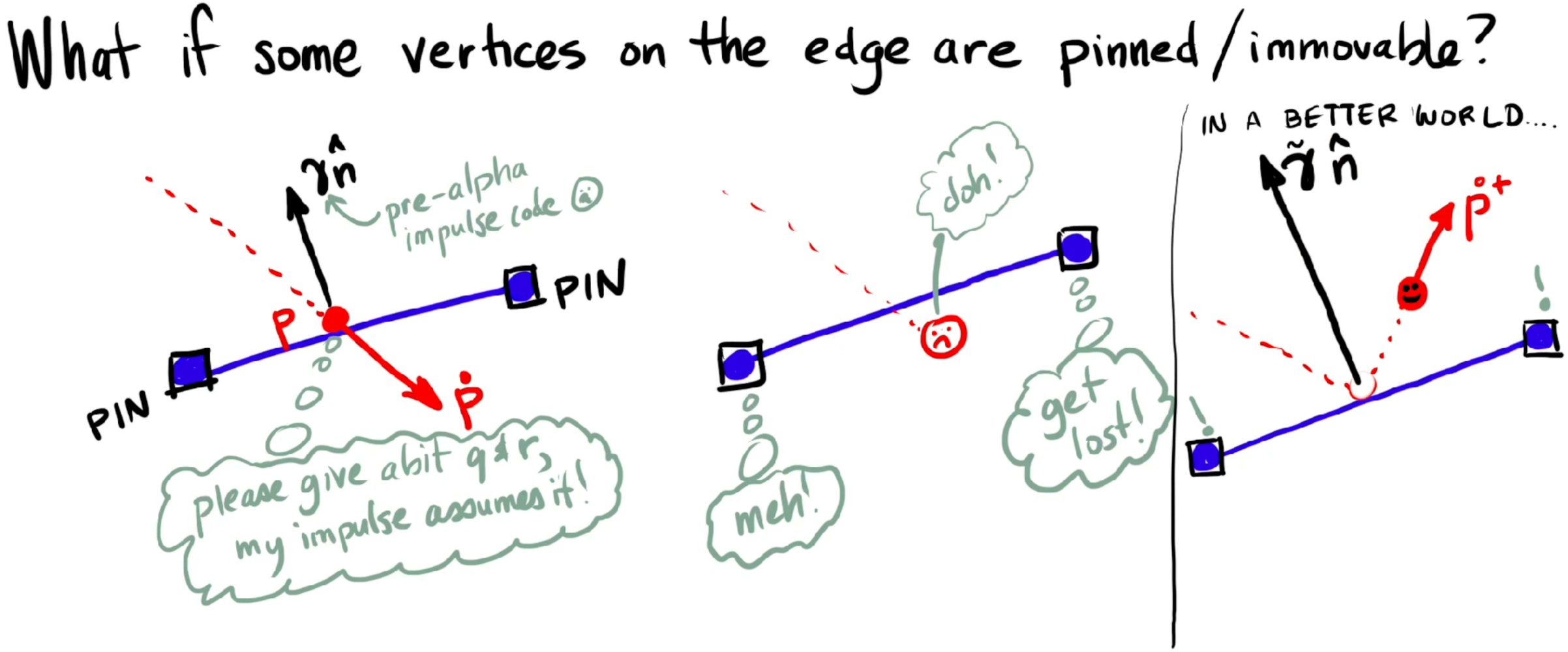
$$\Rightarrow \gamma = \frac{-(1+\varepsilon)\,v_n^-}{\frac{1}{m_p} + \frac{\bar{\alpha}^2}{m_q} + \frac{\alpha^2}{m_r}} \longrightarrow \boxed{\gamma = (1+\varepsilon)\,m_{eff}\,(-v_n^-)} \geq 0$$

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

# 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:

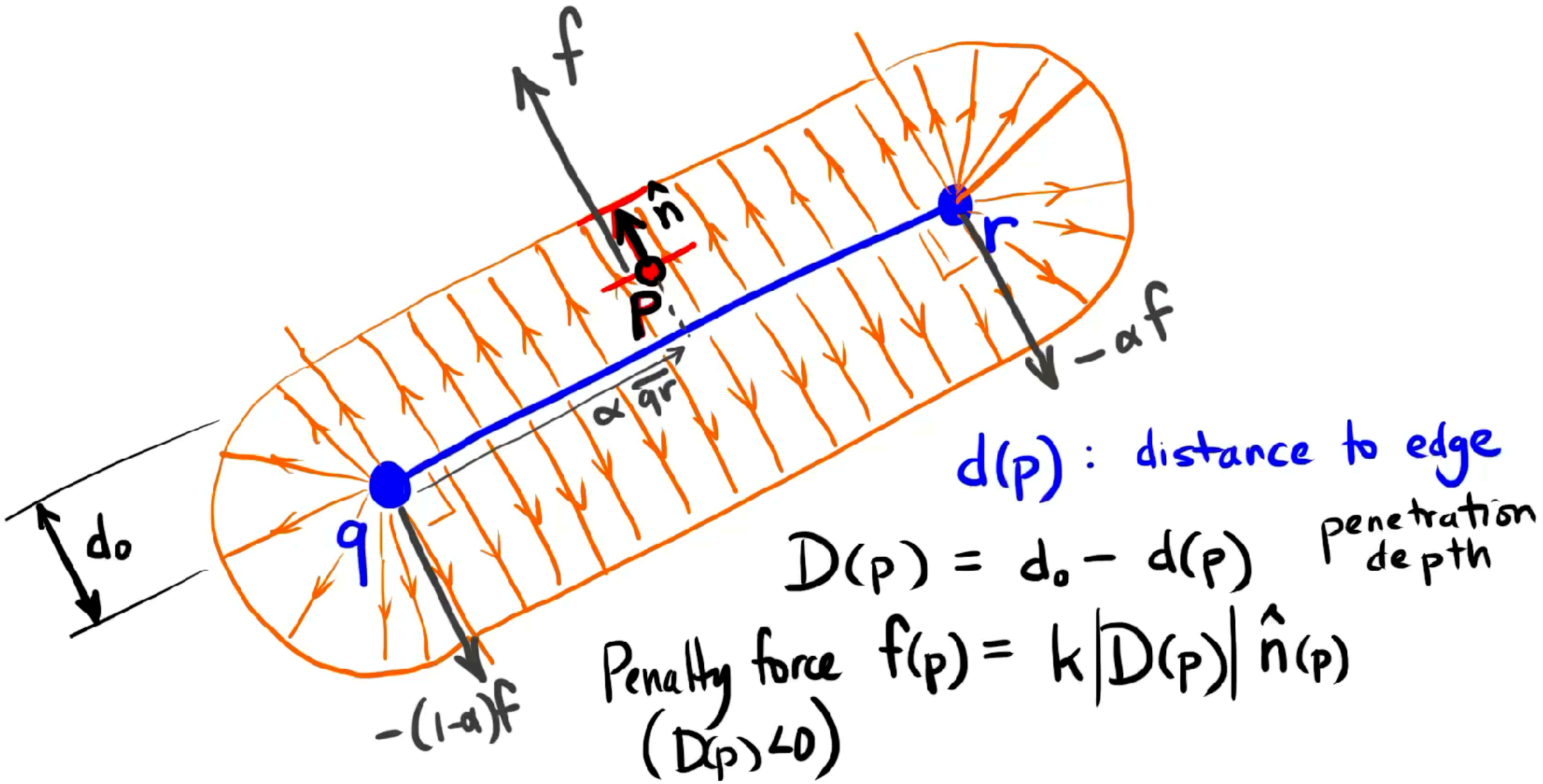
INVERSE MASS: $w_i = \begin{cases} 0 & \text{, if particle "i" doesn't want to play today,} \\ \frac{1}{m_i} & \text{, otherwise} \end{cases}$

- 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

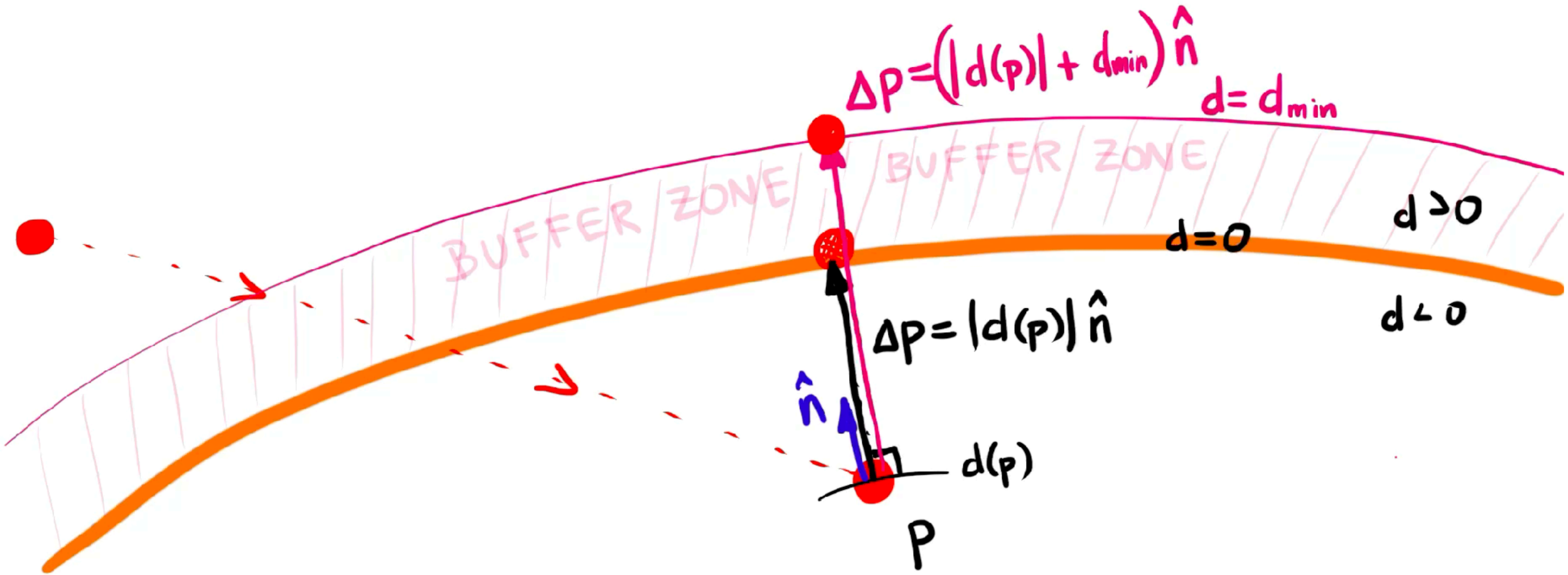
$D(p) = d_0 - d(p)$ penetration depth

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

$(D(p) < 0)$

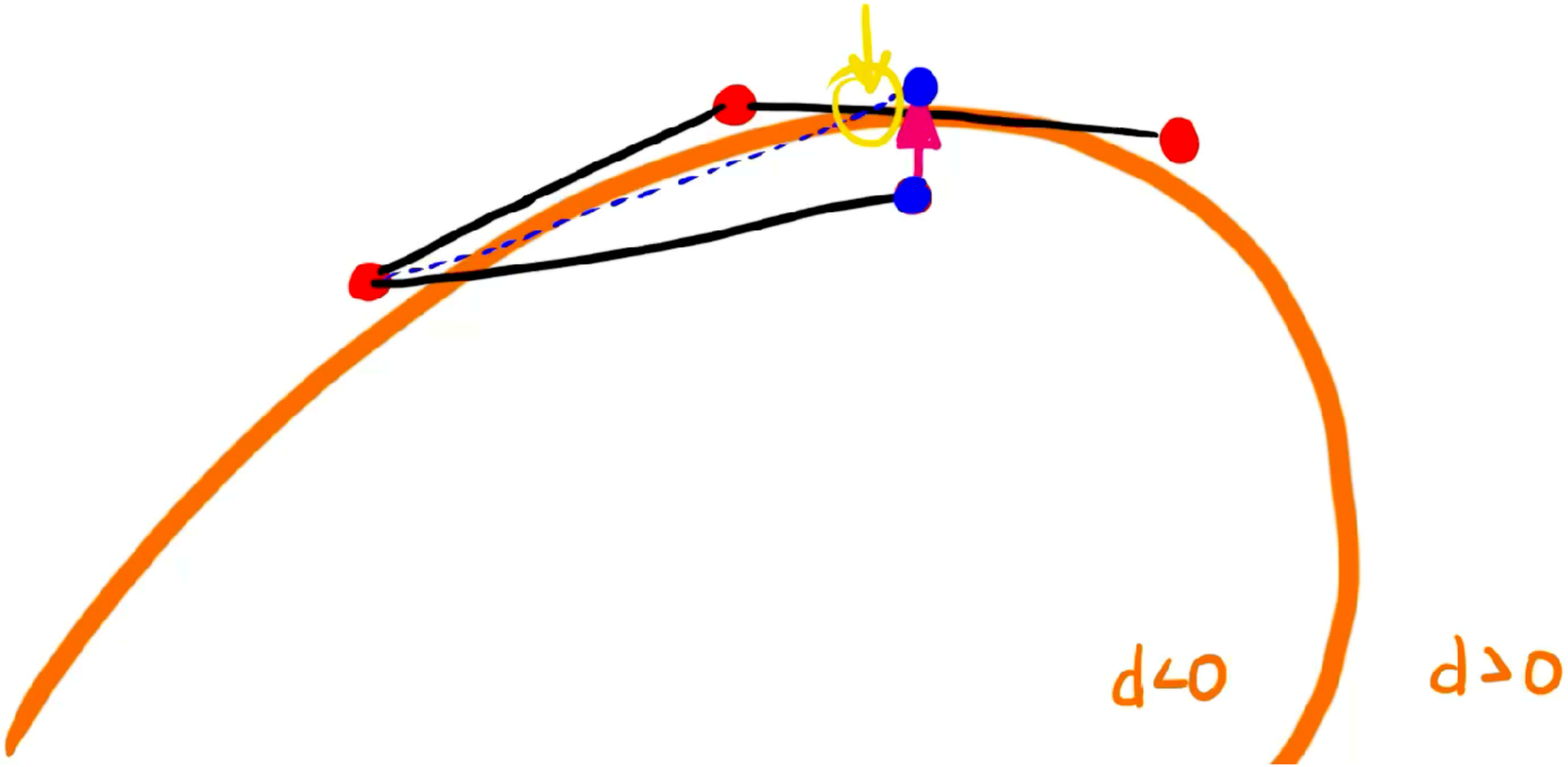
# Removing interpenetration with SDFs

- **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

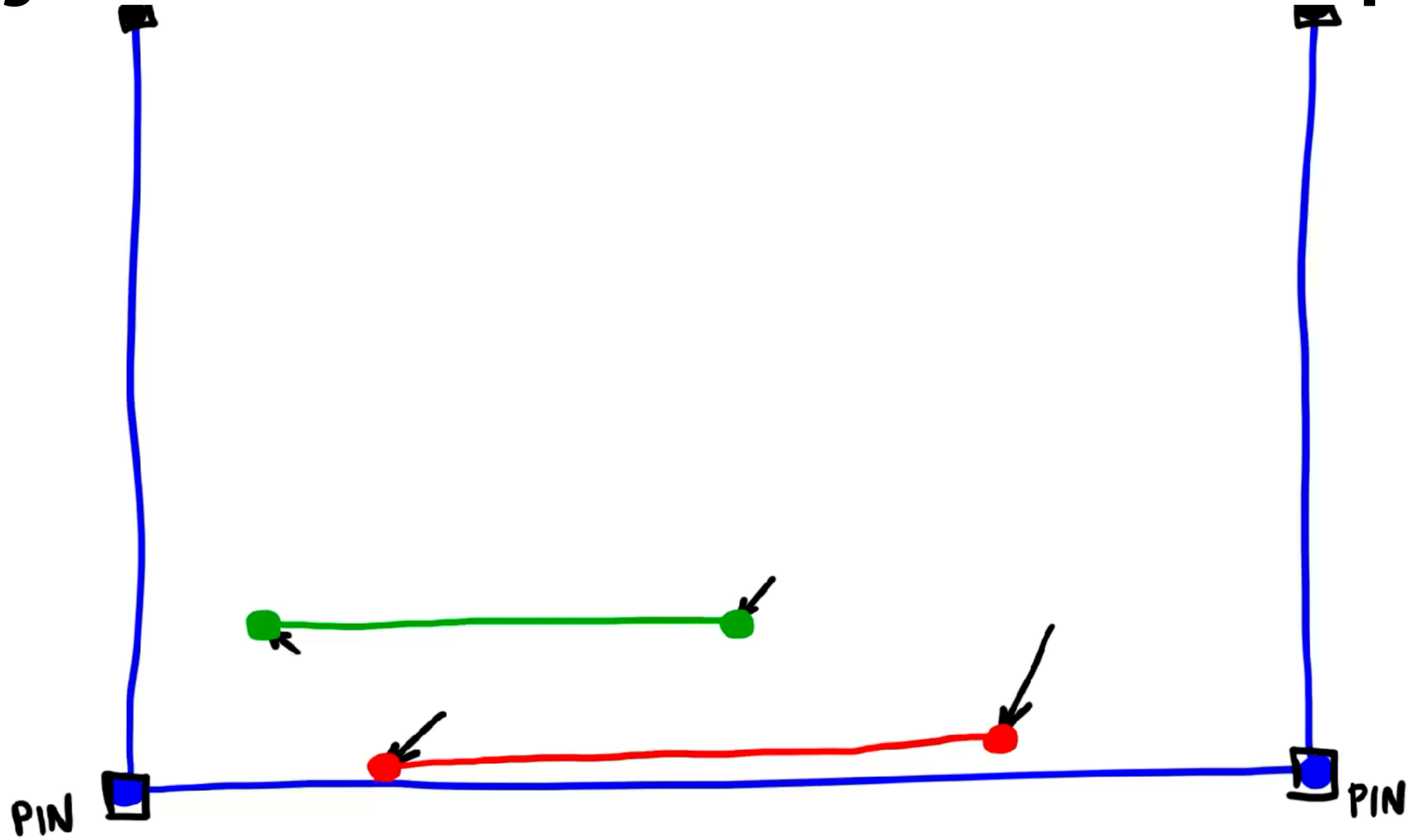
1. **Accumulate forces:** $\mathbf{f}$ (springs, gravity, drag, etc.)
2. **Evaluate accelerations:** $\mathbf{a} = \mathbf{M}^{-1}\mathbf{f}$
   - Optional: Inverse mass filtering for pinned particles
3. **Timestep velocities:** $\mathbf{v} \mathrel{+}= \Delta t\ \mathbf{a}$
   - Optional: Filter velocities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

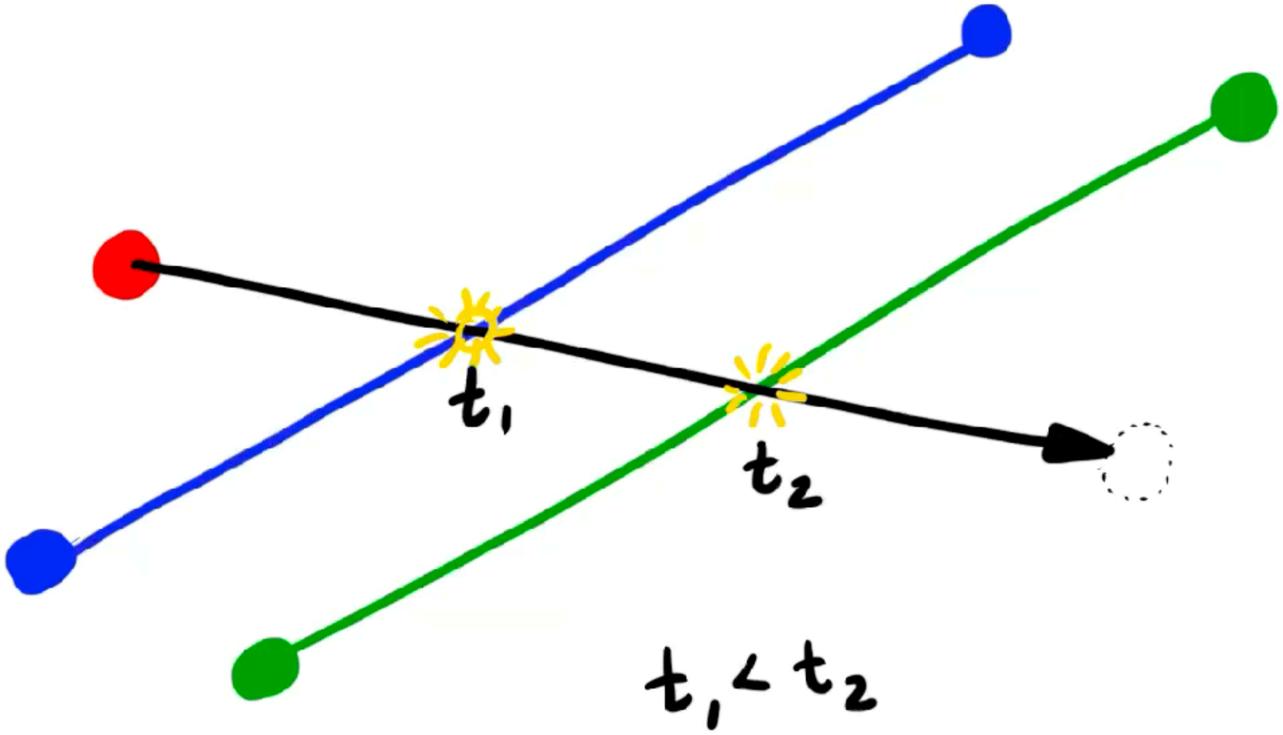
- Velocity filter for collision impulses
  - Freeze particle positions
  - Apply impulses until no collisions detected (CCD) on timestep

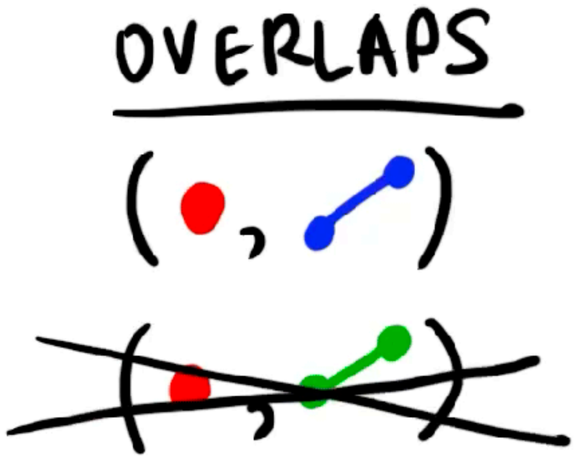
```
C = detectPtLineCollisions();
While(|C| > 0) {
  applyCollisionImpulses(C);     // modify v only
  C = detectPtLineCollisions(); // update C
}
```

- 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



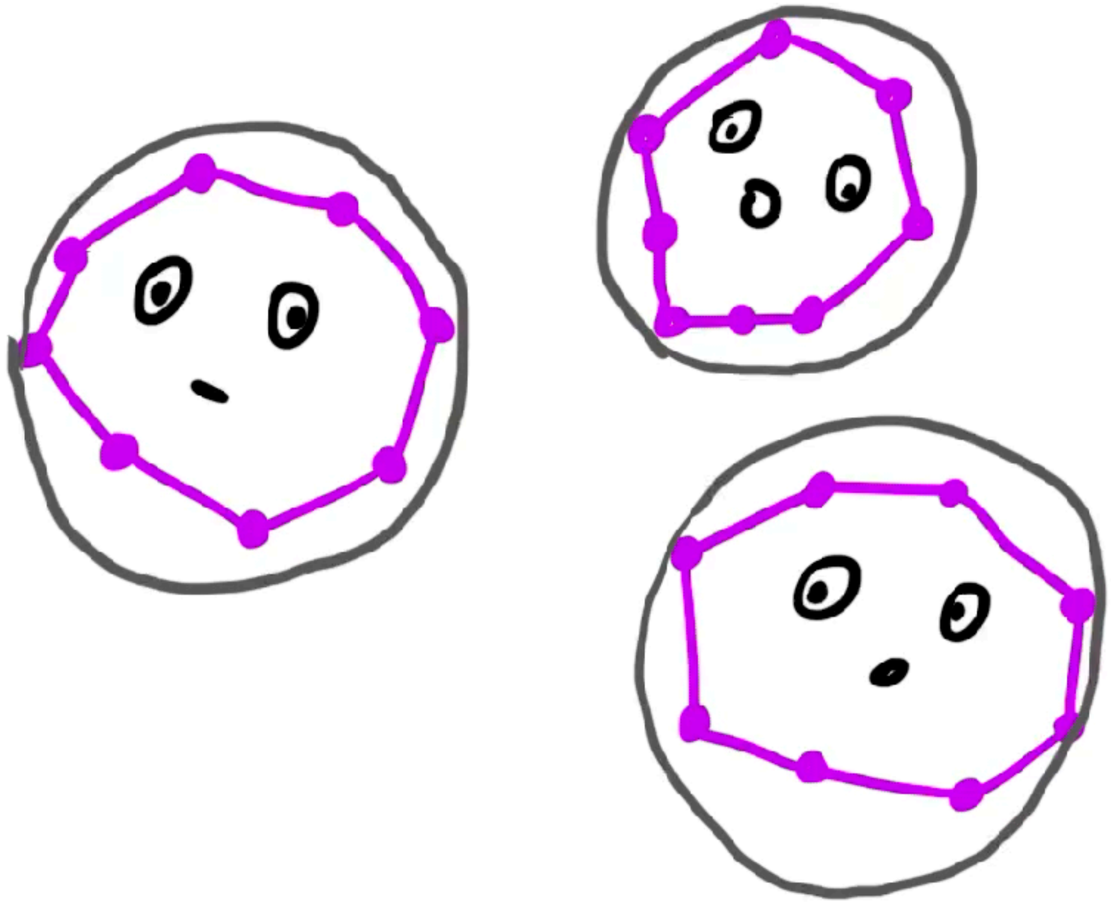

Process early collision ($t_1$) first. Why?

# Point-edge broad-phase CD

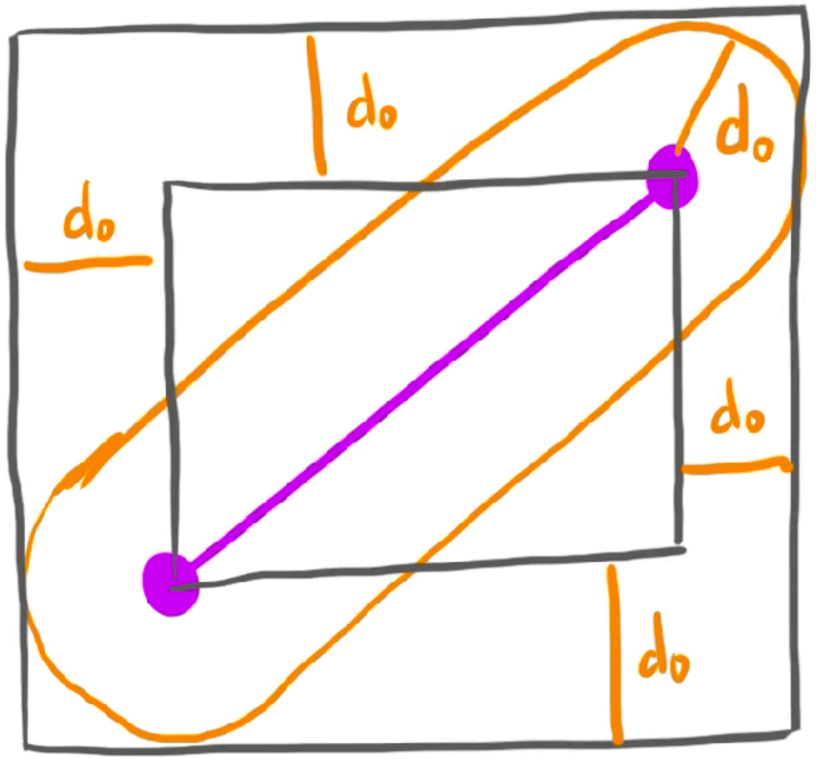
# Broad phase CD for point-edge collisions

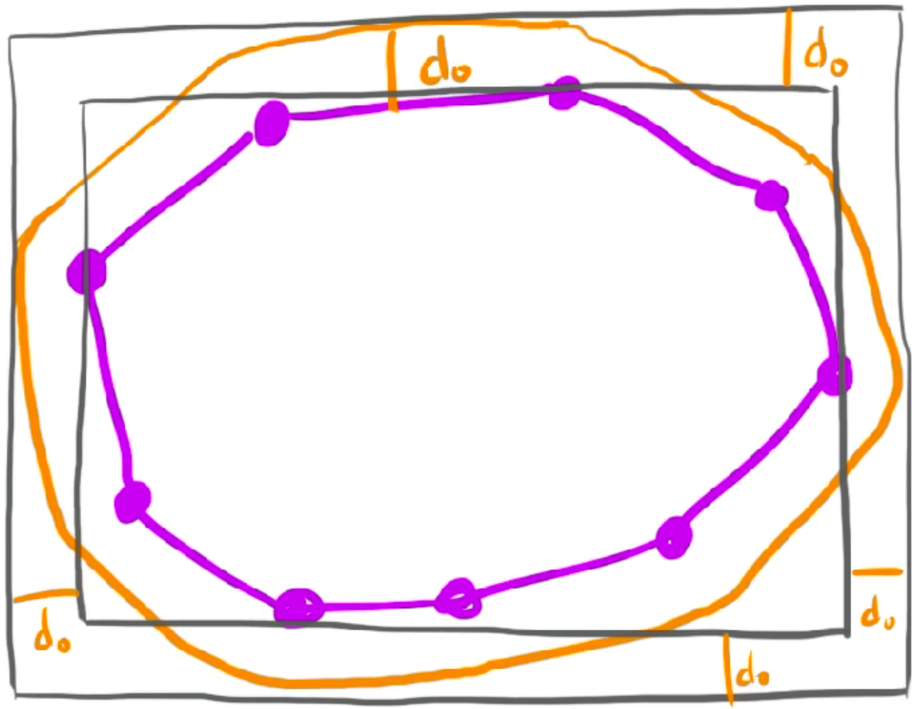
- Two broad-phase problems:
  1. Discrete proximity for penalty forces
  2. Continuous CD for collision impulses
- Not that many blobs
- Bounding volumes effective here

# Blob bounding volumes: AABBs or Disks

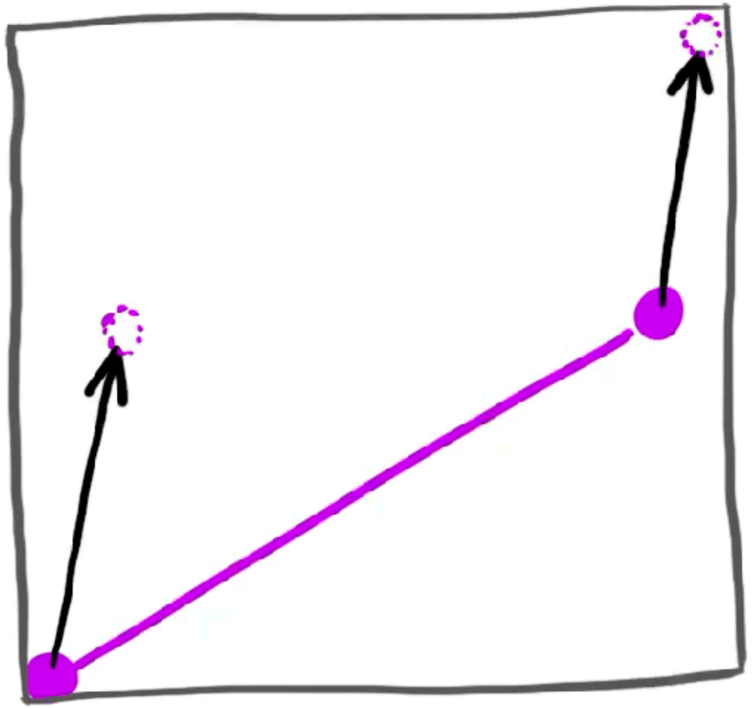


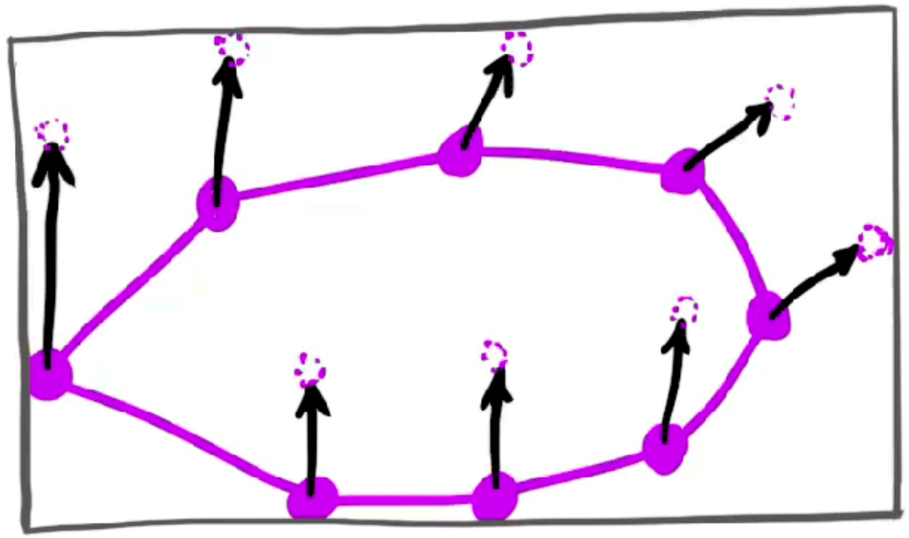
# Bounding blobs: Expand for point-edge penalty forces



# Bounding blobs: Expand for point-edge CCD





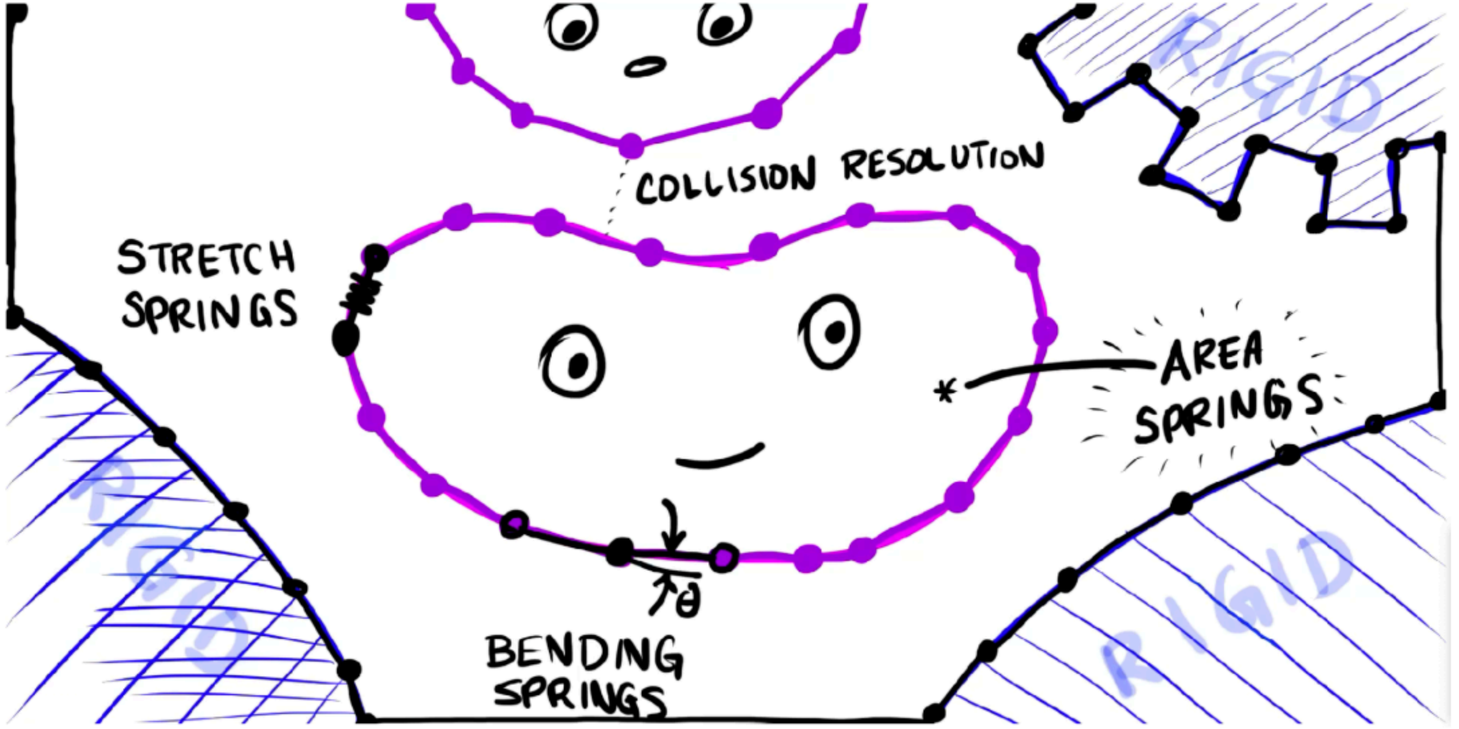
**Stanford CS 248B Fall 2022 – Programming Assignment #2**

# 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

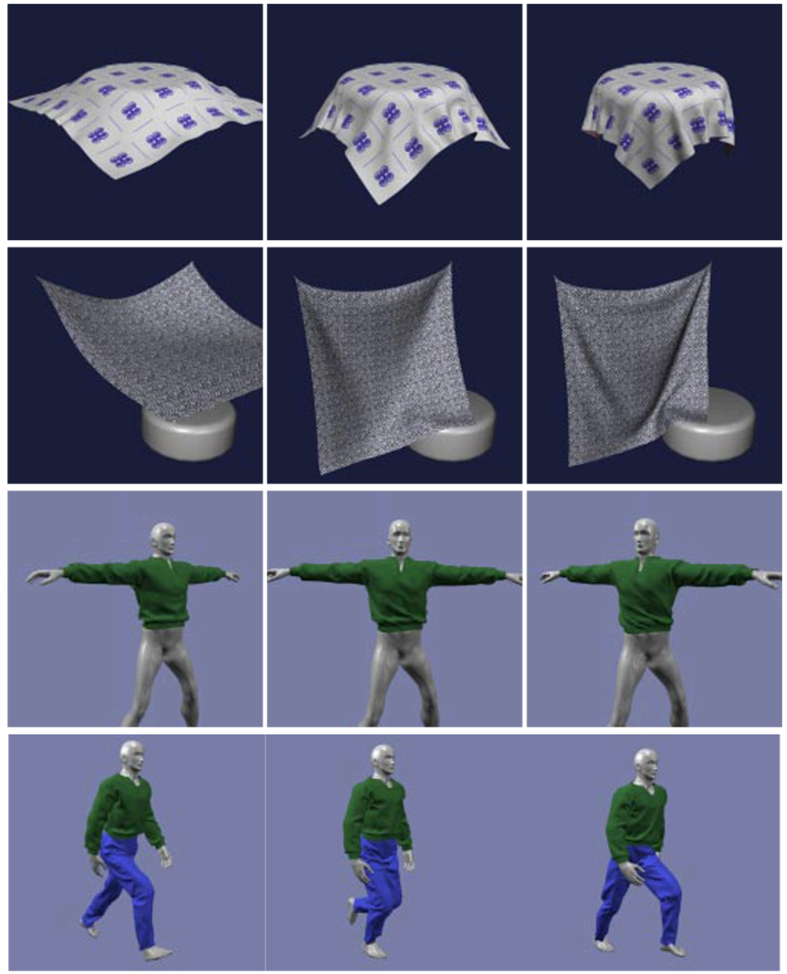


**Figure 1 (top row): Cloth draping on cylinder; frames 8, 13 and 35. Figure 2 (second row): Sheet with two fixed particles; frames 10, 29 and 67. Figure 3 (third row): Shirt on twisting figure; frames 1, 24 and 46. Figure 4 (bottom row): Walking man; frames 30, 45 and 58.**

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 $\Delta v$:

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

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

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

# 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 (*)