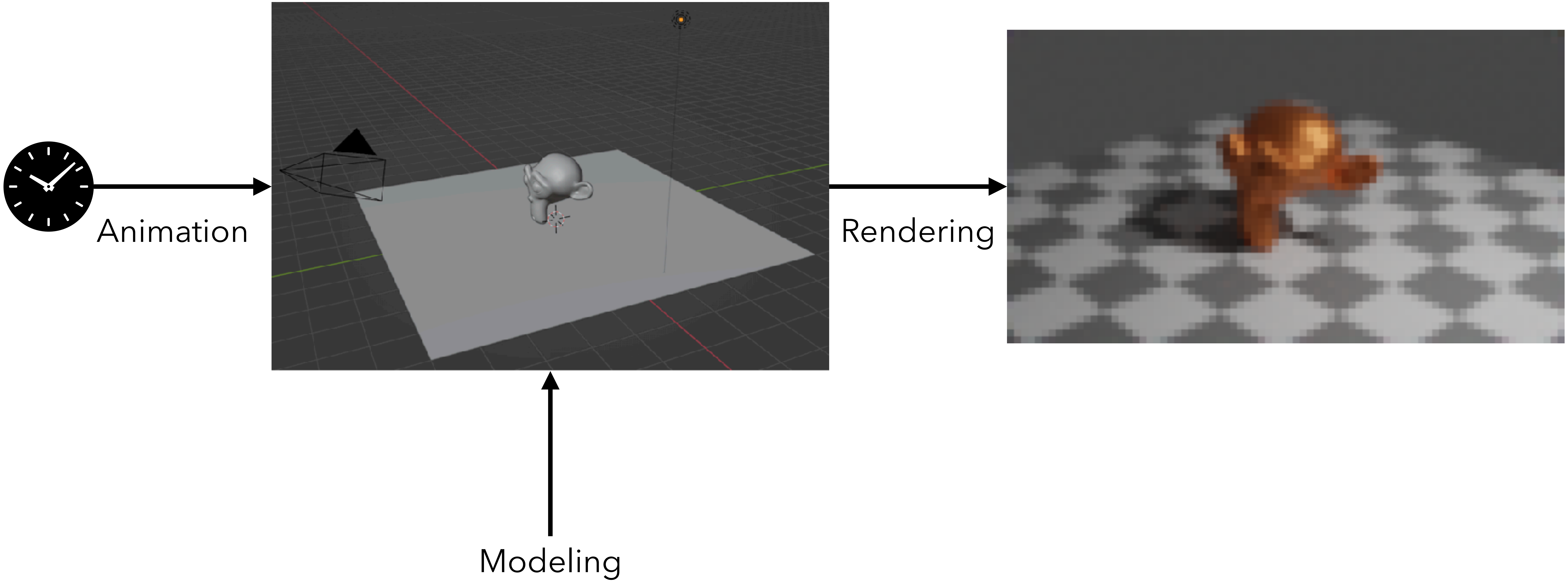
COV877: Differentiable Graphics for Vision and Learning

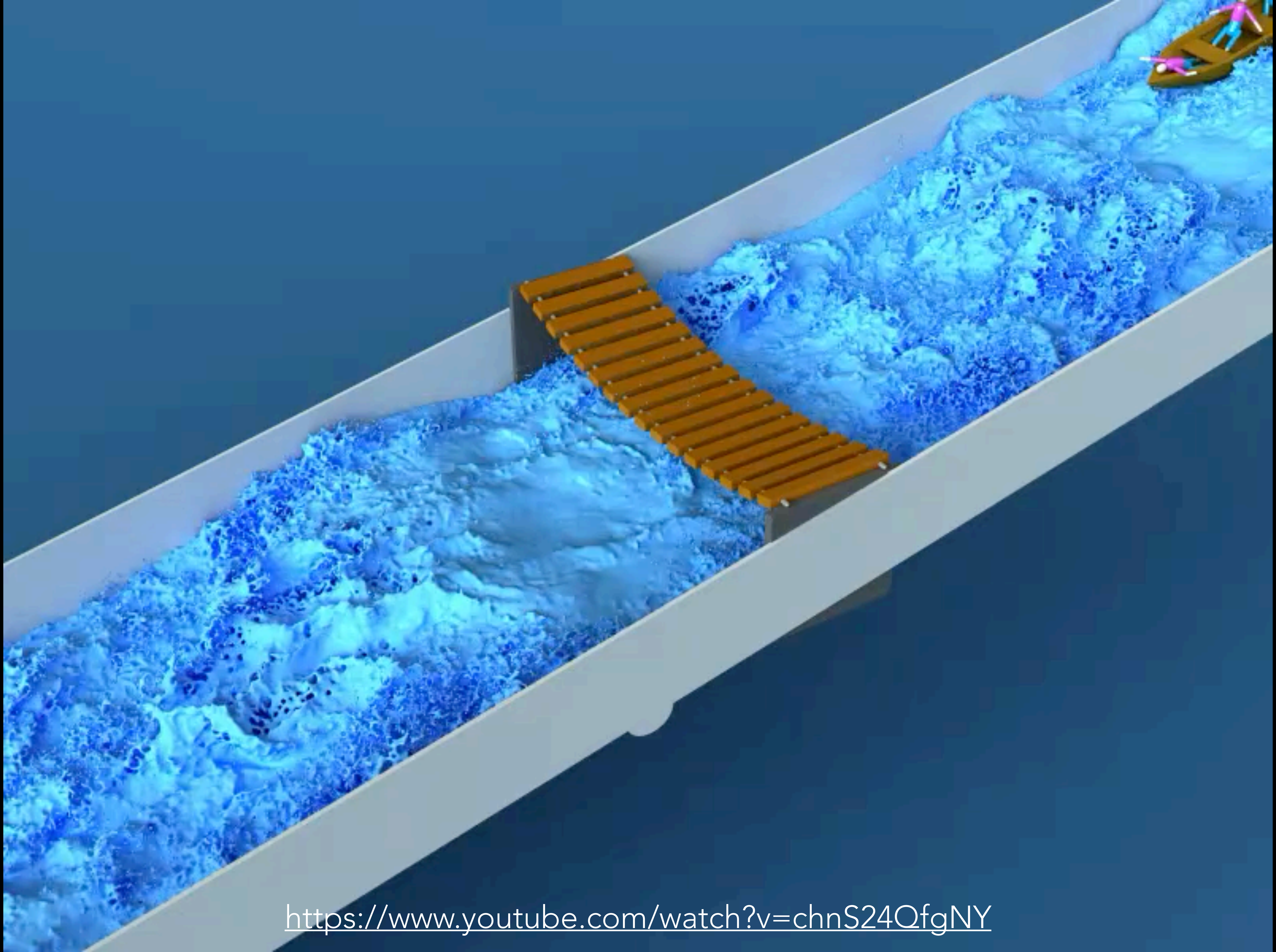
# 4. Differentiable Simulation

# Animation



# CAESAR

https://www.youtube.com/watch?v=4NU9ikjqjC0



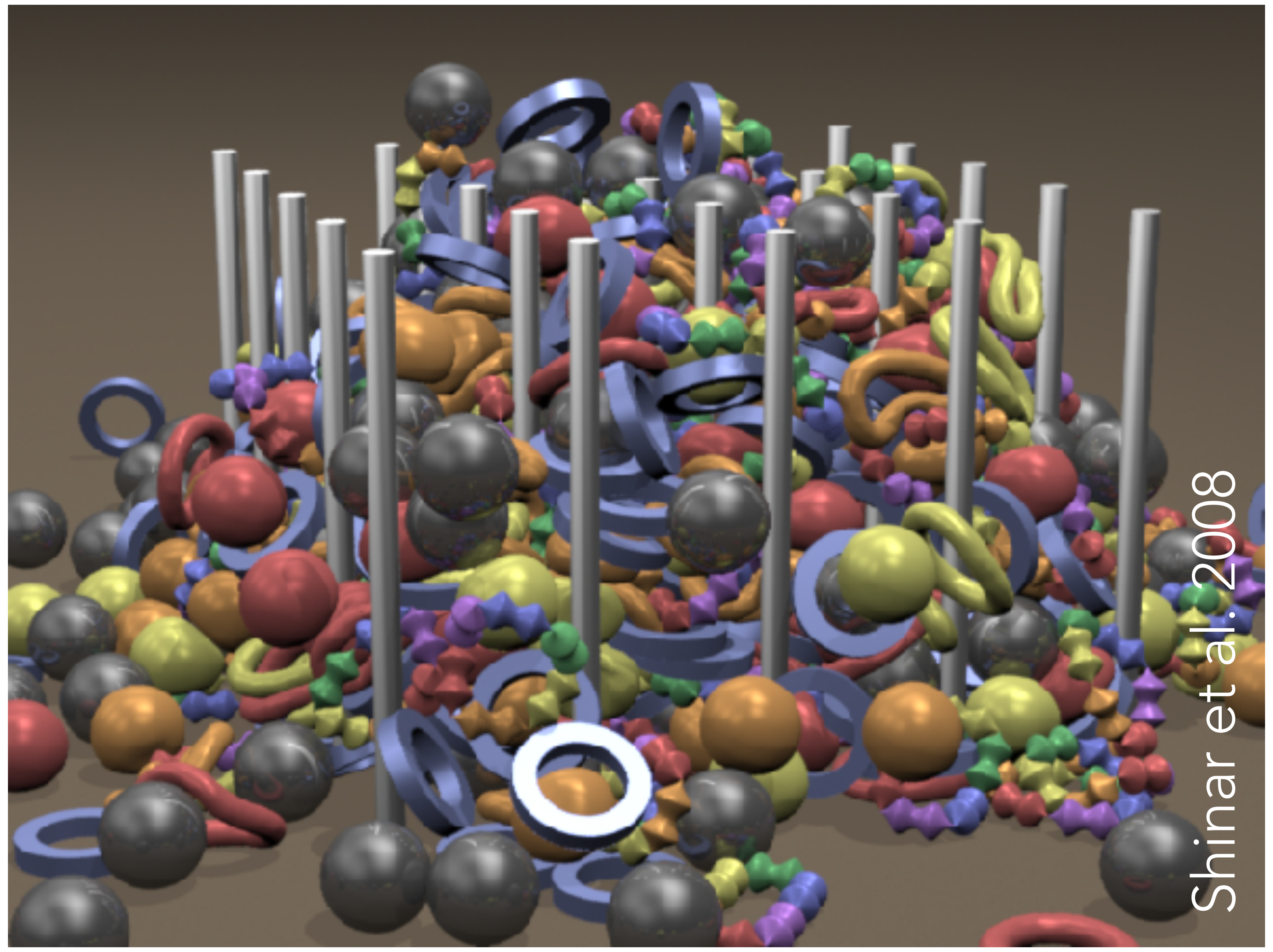
https://www.youtube.com/watch?v=chnS24QfgNY

Akinci et al. 2012

# Simulation

What makes the motion of a physical object look real?

$$\mathbf{F} = ma$$





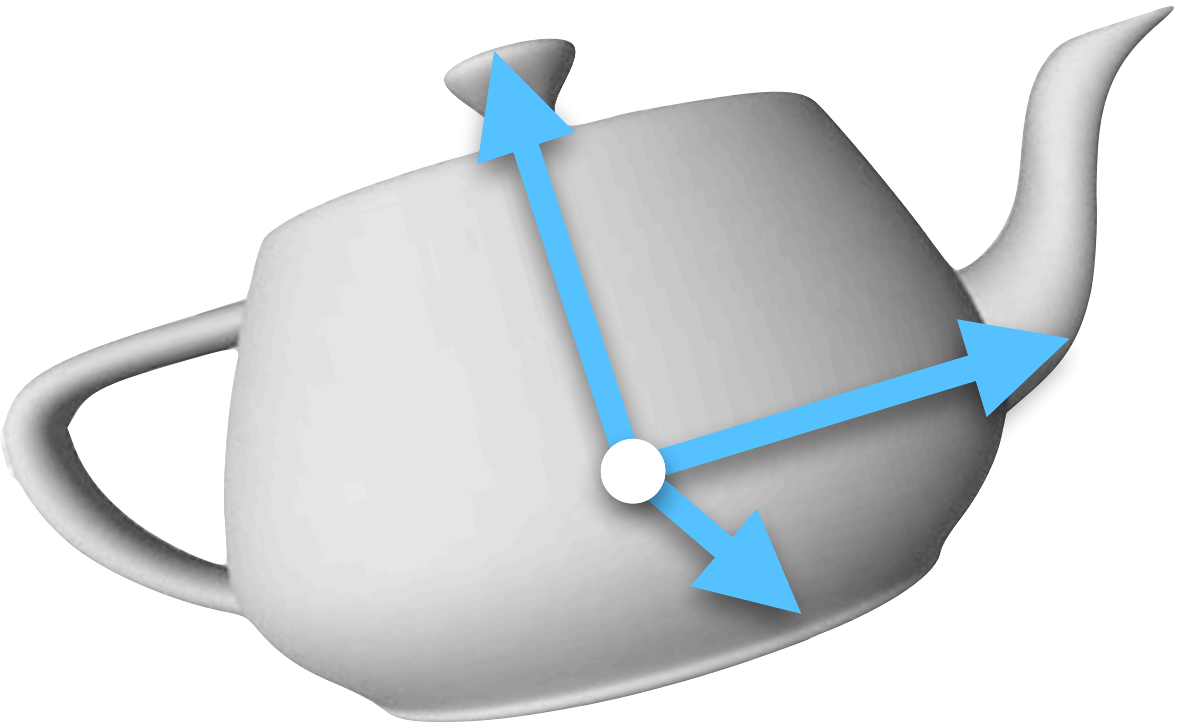
Solve the equations of motion to automatically get physically realistic motion.

e.g. **Rigid bodies**

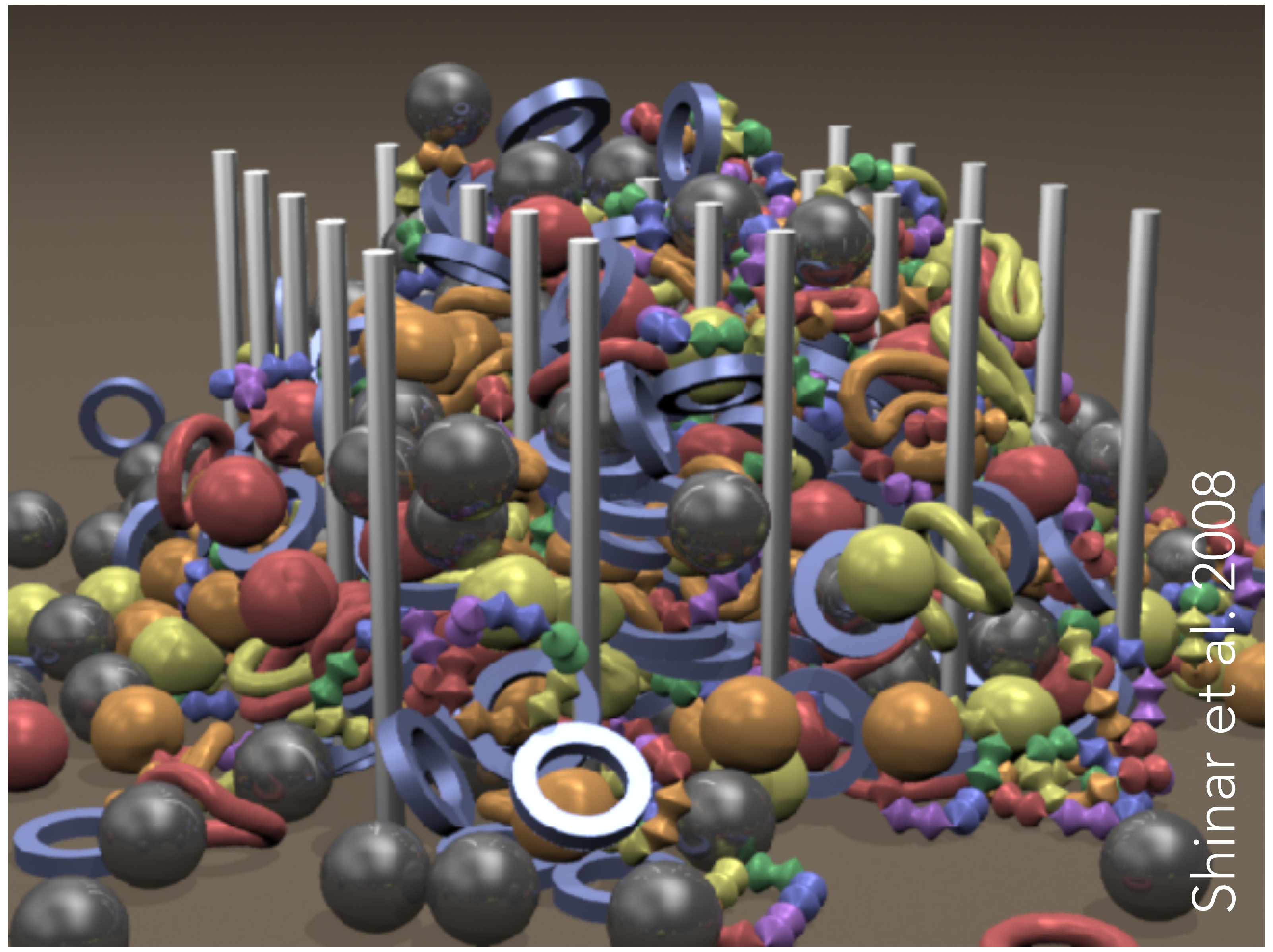
- Degrees of freedom: position, rotation

$$\frac{d^2\mathbf{x}}{dt^2} = \mathbf{f}_{ext}/m$$

$$\frac{d^2\mathbf{R}}{dt^2} = \cdots$$



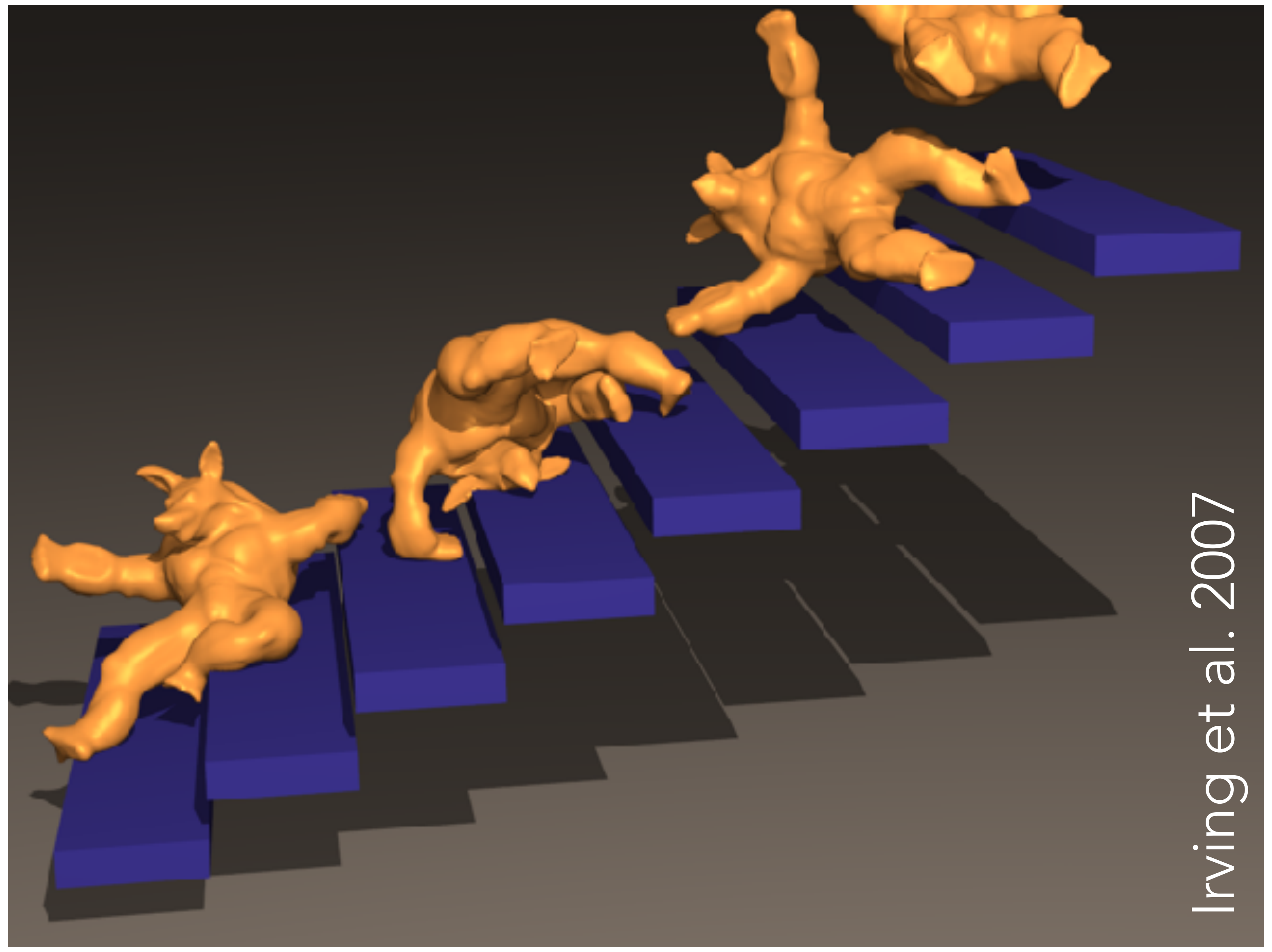
- Challenges: collisions, frictional contact, stacking



# Deformable bodies, cloth, etc.

Every vertex can move independently! But deformation causes internal elastic forces

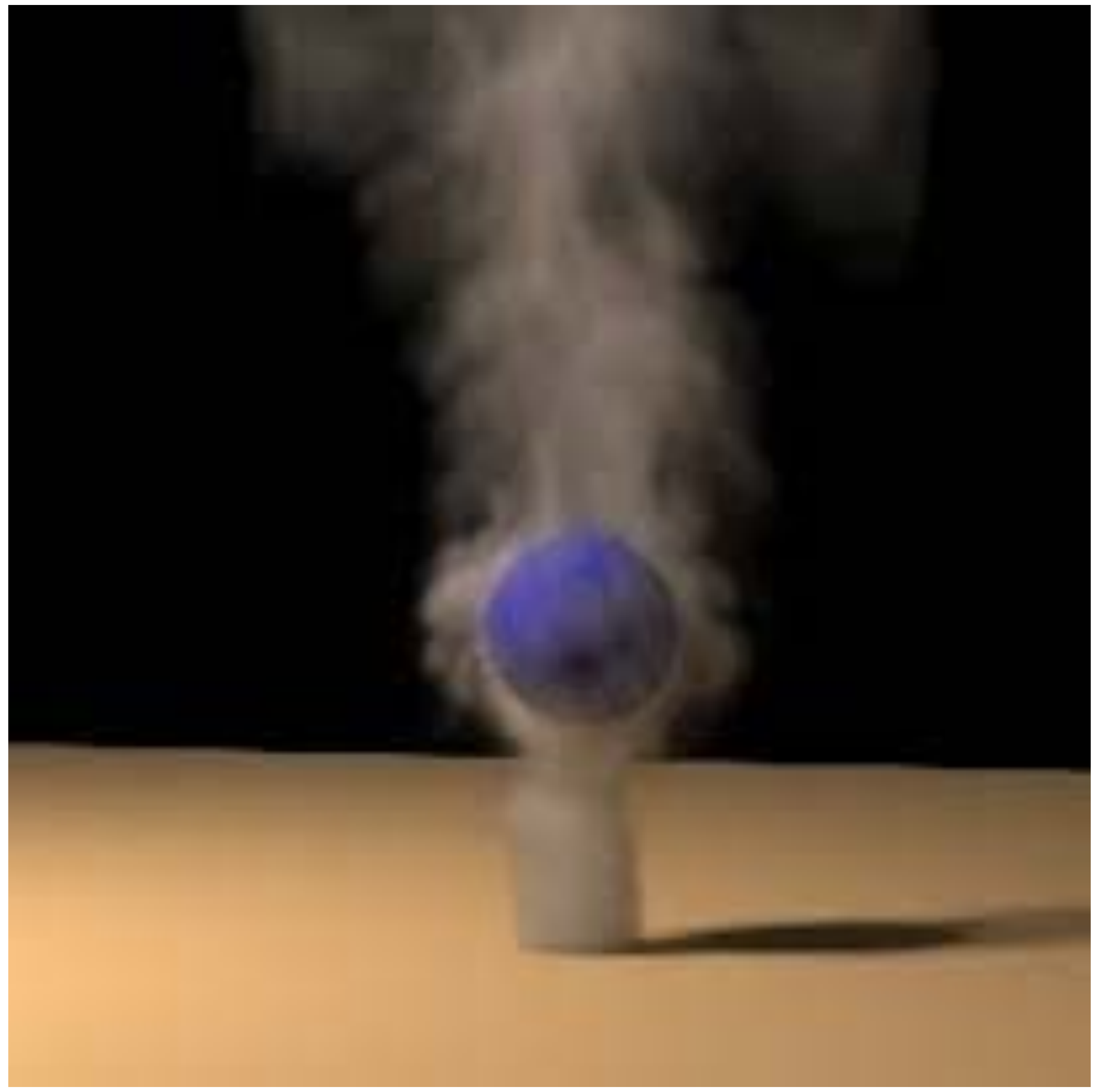
- Physically accurate: **finite element method**
- Cheap approximation: **mass-spring systems** (just a bunch of particles and 1D springs)

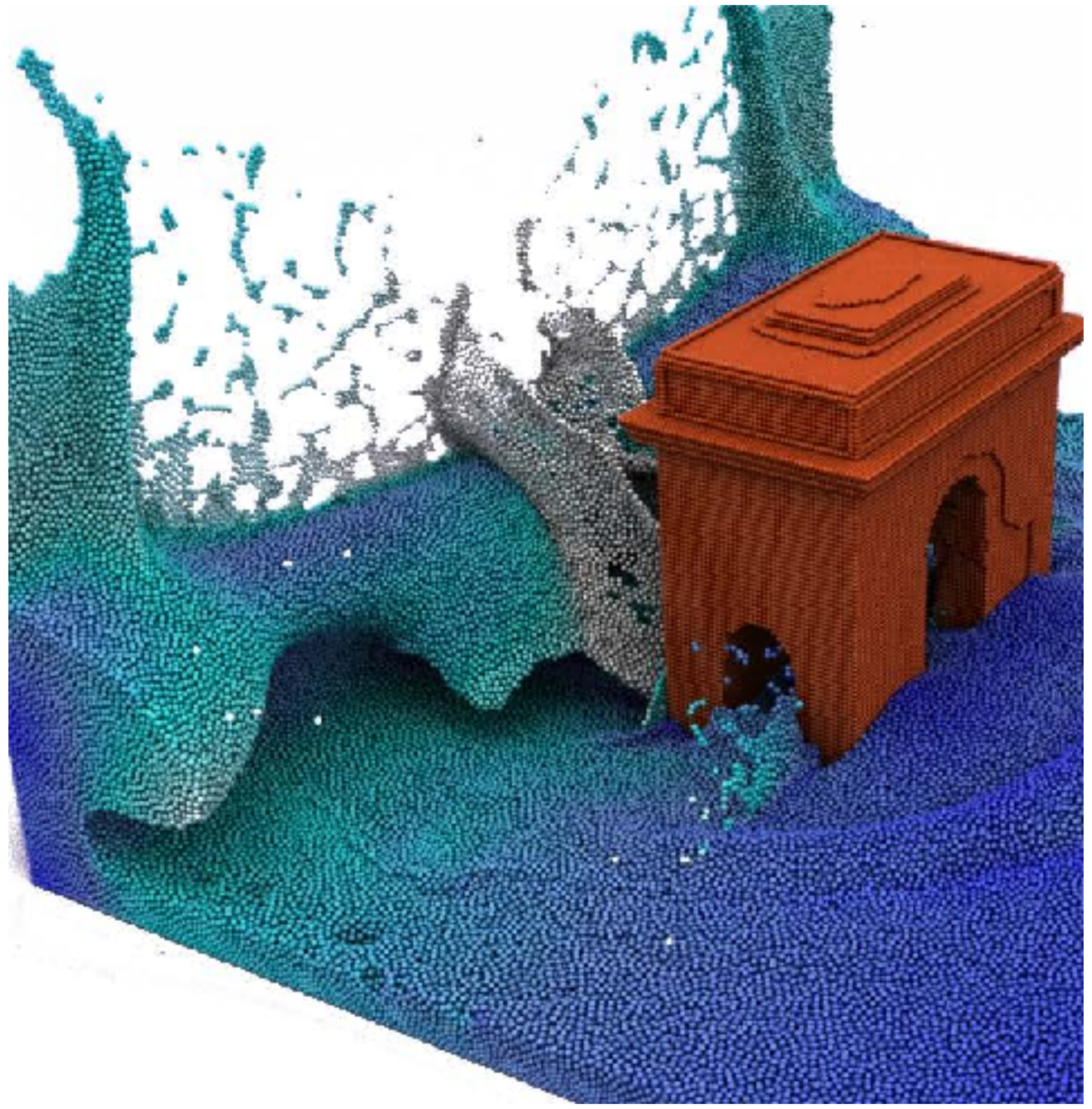



# Fluids (smoke, water, fire, etc.)

Described by the Navier-Stokes equations (system of partial differential equations)

Velocity field **v**(**x**): every point has its own velocity!



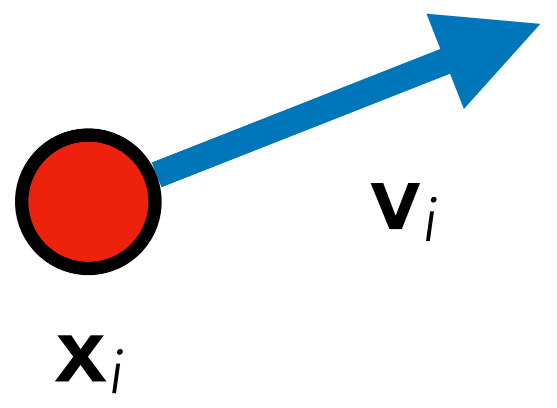


Let’s start simple…

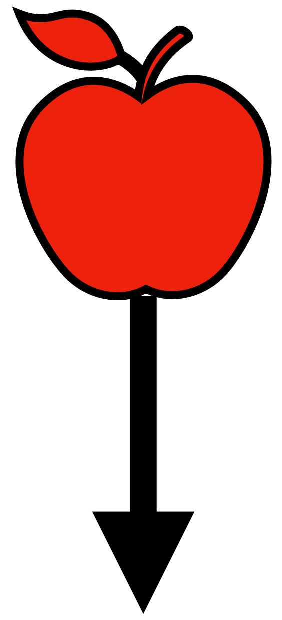
**Particle system** = collection of (usually non-interacting) particles in motion

Each particle is a point mass

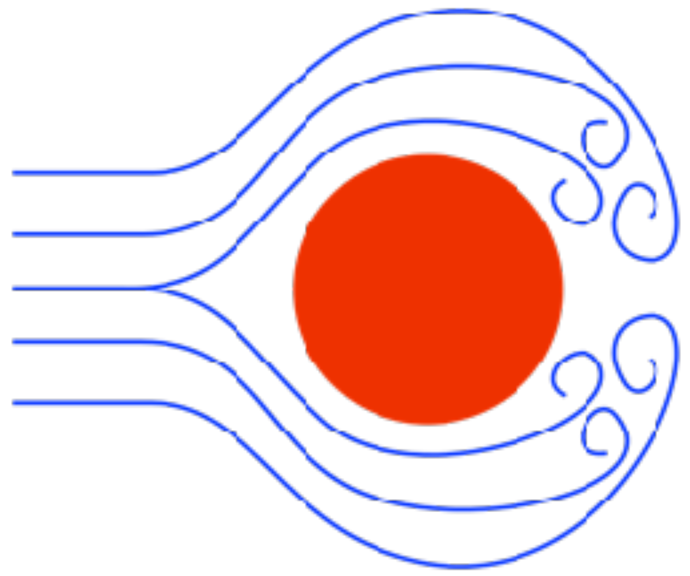
- Fixed: mass $m_i$
- Variable state: position $\mathbf{x}_i$, velocity $\mathbf{v}_i$



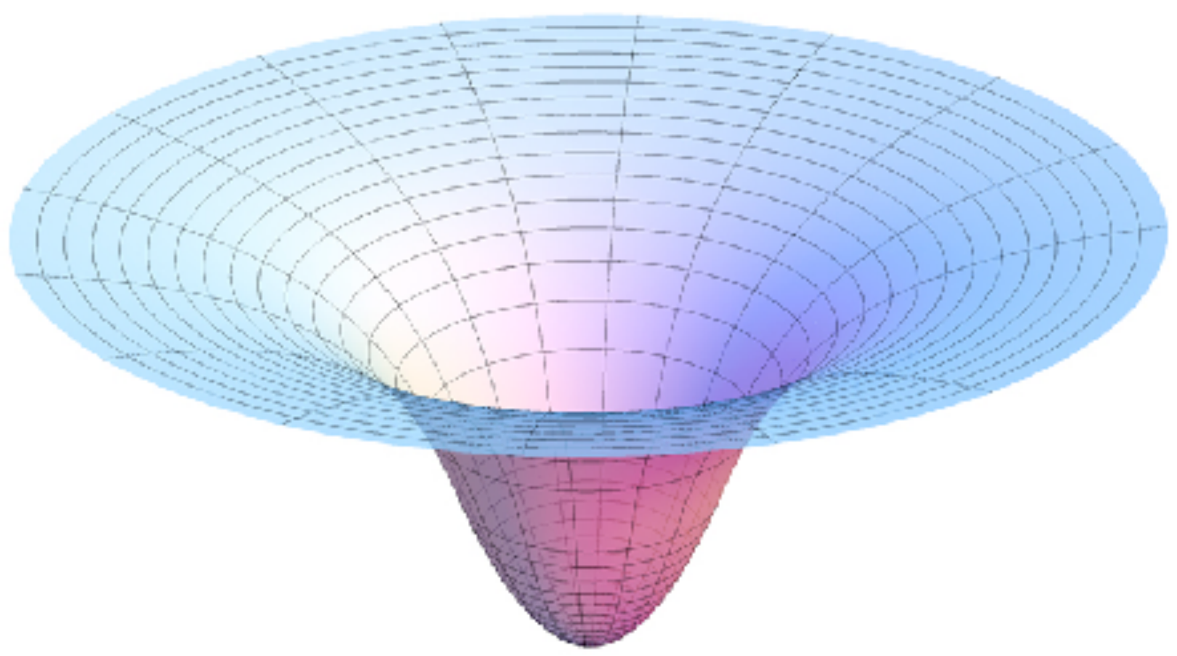

Affected by some forces $\mathbf{f}_i = \mathbf{f}(t, \mathbf{x}_i(t), \mathbf{v}_i(t))$



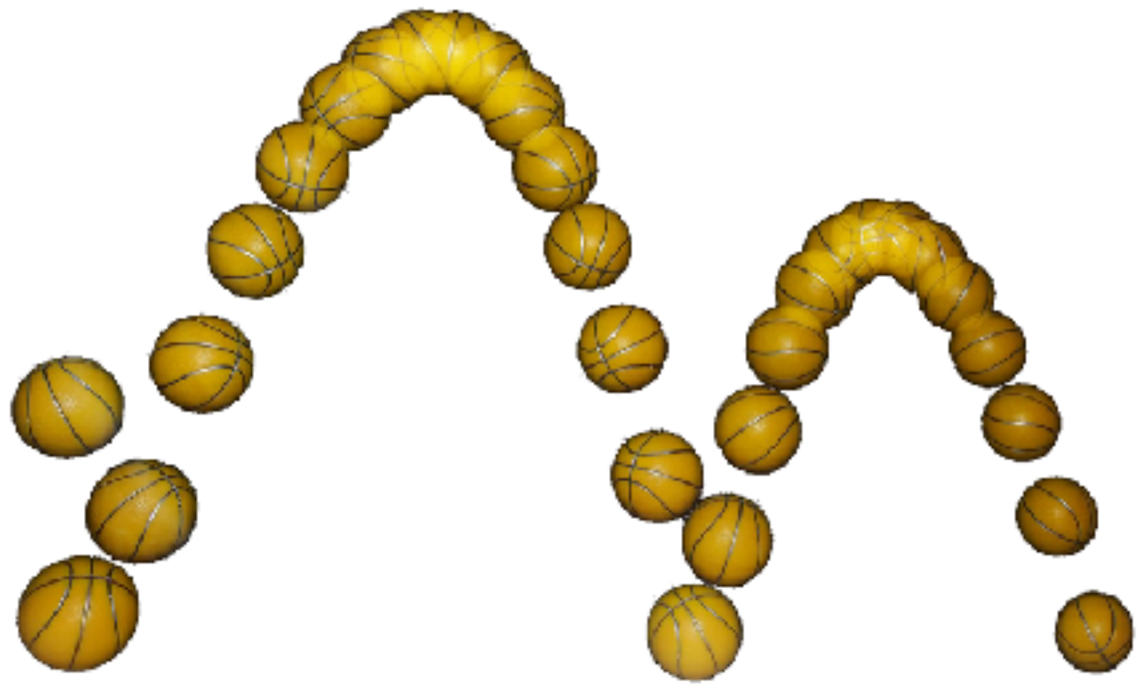
Gravity
$\mathbf{f} = m\mathbf{g}$



Wind / drag
$\mathbf{f} = -k_d(\mathbf{v} - \mathbf{v}_{\text{air}})$



Spatial fields
$\mathbf{f} = \mathbf{f}(\mathbf{x})$



Collisions
$\mathbf{f} = \ldots$TBD

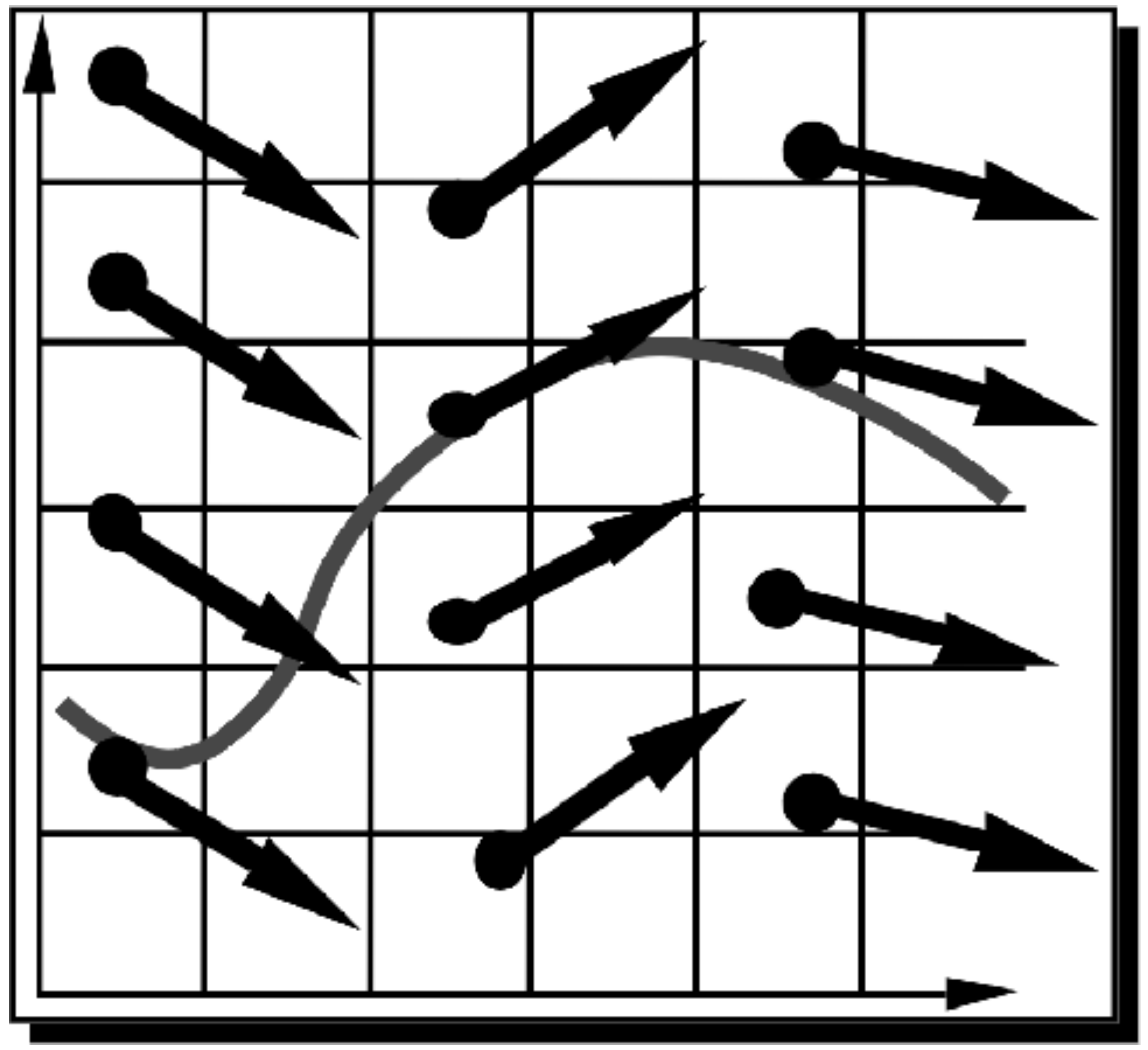
Equations of motion: $\mathbf{f} = m\mathbf{a}$ (where $\mathbf{f}$ is total force) so…

$$\frac{\mathrm{d}^2\mathbf{x}(t)}{\mathrm{d}t^2} = m^{-1}\,\mathbf{f}(t, \mathbf{x}(t), \mathbf{v}(t))$$

For each emitted particle, we know initial position $\mathbf{x}(0)$ and velocity $\mathbf{v}(0)$. How to find $\mathbf{x}(t)$, $\mathbf{v}(t)$ at any future time $t$?

In general, no closed form unless $\mathbf{f}$ is very simple!

Like with rendering, we need a numerical method…



Witkin & Baraff 1997

# Time stepping

Idea: Given a known state ($\mathbf{x}(t)$, $\mathbf{v}(t)$), estimate a near future state ($\mathbf{x}(t+\Delta t)$, $\mathbf{v}(t+\Delta t)$).

Then we can iterate: $(\mathbf{x}(0), \mathbf{v}(0)) \to (\mathbf{x}(\Delta t), \mathbf{v}(\Delta t)) \to (\mathbf{x}(2\Delta t), \mathbf{v}(2\Delta t)) \to (\mathbf{x}(3\Delta t), \mathbf{v}(3\Delta t)) \to \cdots$

$$\frac{\mathrm{d}\mathbf{x}(t)}{\mathrm{d}t} = \mathbf{v}(t)$$

$$\frac{\mathrm{d}\mathbf{v}(t)}{\mathrm{d}t} = m^{-1}\,\mathbf{f}(t, \mathbf{x}(t), \mathbf{v}(t))$$

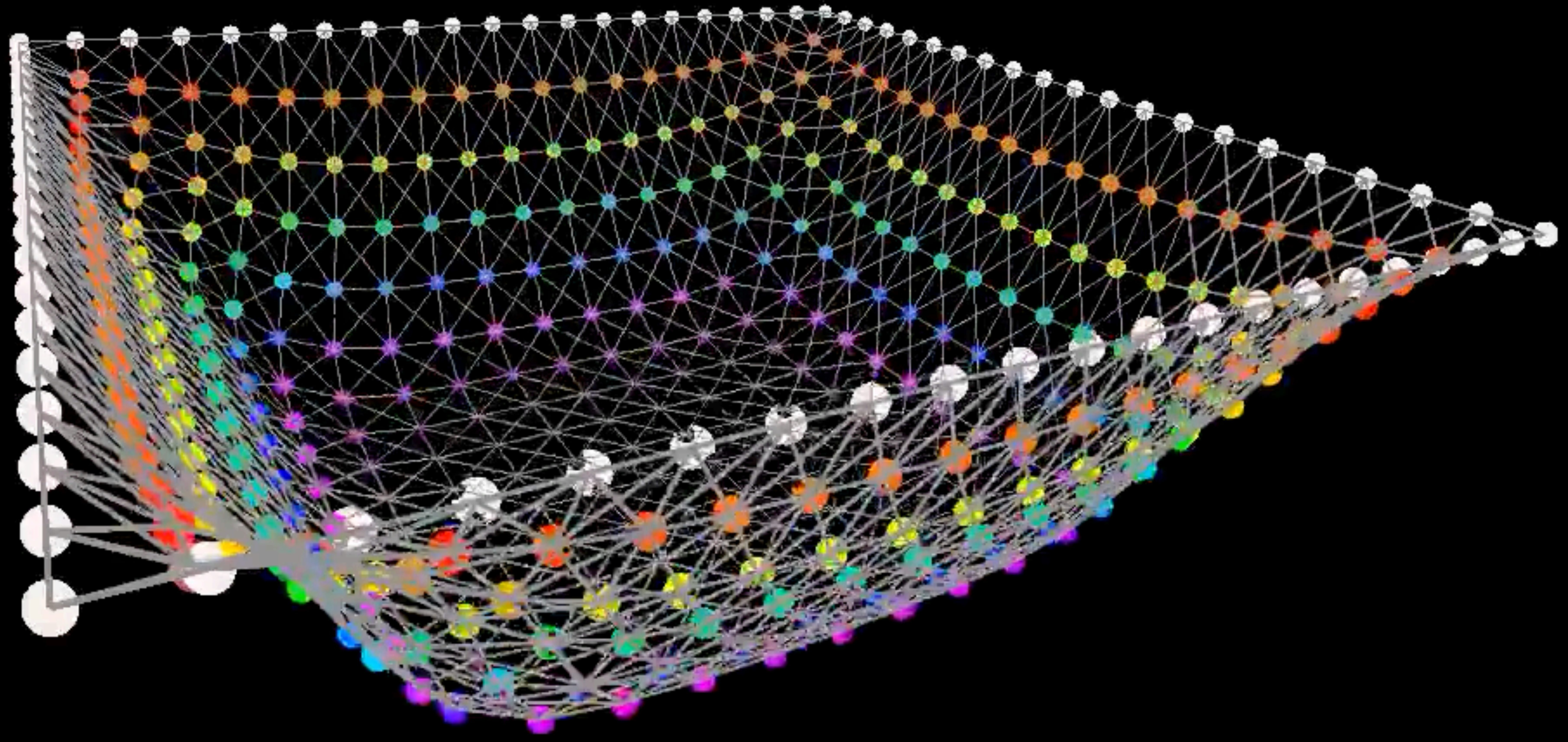
Simplest strategy:

$$\mathbf{v}(t + \Delta t) = \mathbf{v}(t) + m^{-1}\,\mathbf{f}(t, \mathbf{x}(t), \mathbf{v}(t))\,\Delta t$$

$$\mathbf{x}(t + \Delta t) = \mathbf{x}(t) + \mathbf{v}(t + \Delta t)\,\Delta t$$

because we already have it from the previous step

# Mass-spring systems



https://www.youtube.com/watch?v=ib1vmRDs8Vw

Paul Nathan

In 3D, suppose a spring of length $\ell_0$ and stiffness $k_s$ connects particles $i$ and $j$.
What should be the force $\mathbf{f}_{ij}$ on $i$ due to $j$?

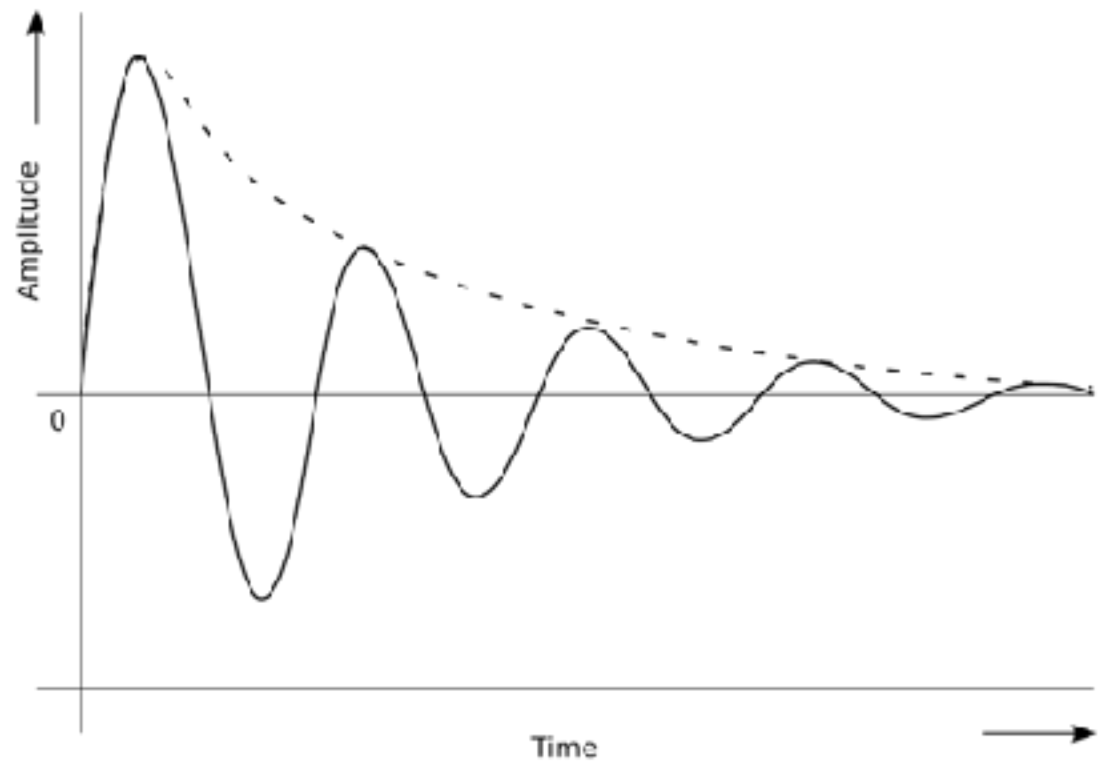
Let's first define the potential energy:

$$U = \tfrac{1}{2}\, k_s\, (\|\mathbf{x}_i - \mathbf{x}_j\| - \ell_0)^2$$

Then $\mathbf{f}_{ij} = -\partial U / \partial \mathbf{x}_i \Rightarrow$

$$\mathbf{f}_{ij} = -k_s\, (\|\mathbf{x}_i - \mathbf{x}_j\| - \ell_0)\, \frac{\mathbf{x}_i - \mathbf{x}_j}{\|\mathbf{x}_i - \mathbf{x}_j\|}$$

$$= -k_s\, (\|\mathbf{x}_{ij}\| - \ell_0)\, \hat{\mathbf{x}}_{ij}$$




Similarly $\mathbf{f}_{ji} = -\partial U / \partial \mathbf{x}_j$ (but it's also just $-\mathbf{f}_{ij}$)

Also add a damping force $\mathbf{f}_{ij} = -k_d\, (\mathbf{v}_{ij} \cdot \hat{\mathbf{x}}_{ij})\, \hat{\mathbf{x}}_{ij}$ to dissipate energy

Sum of contributions from *all* incident springs.
May depend on $\mathbf{x}_1(t)$, $\mathbf{v}_1(t)$, $\mathbf{x}_2(t)$, $\mathbf{v}_2(t)$, …!

How to compute? Same strategy:

$$\mathbf{v}_i(t + \Delta t) = \mathbf{v}_i(t) + m_i^{-1}\,\mathbf{f}_i(t)\,\Delta t$$

$$\mathbf{x}_i(t + \Delta t) = \mathbf{x}_i(t) + \mathbf{v}_i(t + \Delta t)\,\Delta t$$

Pseudocode:

```
for each particle p:
  p.f = 0
for each force object F:
  for each particle p affected by F:
    p.f += force on p due to F
for each particle p:
  p.v += p.f/p.m * dt
  p.x += p.v * dt
```

Simpler with generalized coordinates:

$$\mathbf{q} = \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \vdots \\ \mathbf{x}_n \end{bmatrix}, \quad \mathbf{v} = \begin{bmatrix} \mathbf{v}_1 \\ \mathbf{v}_2 \\ \vdots \\ \mathbf{v}_n \end{bmatrix}$$

Then

$$\frac{\mathrm{d}^2\mathbf{q}(t)}{\mathrm{d}t^2} = \begin{bmatrix} m_1^{-1}\mathbf{f}_1(t,\mathbf{q},\mathbf{v}) \\ m_2^{-1}\mathbf{f}_2(t,\mathbf{q},\mathbf{v}) \\ \vdots \\ m_n^{-1}\mathbf{f}_n(t,\mathbf{q},\mathbf{v}) \end{bmatrix} = \begin{bmatrix} m_1\mathbf{I} & & & \\ & m_2\mathbf{I} & & \\ & & \ddots & \\ & & & m_n\mathbf{I} \end{bmatrix}^{-1} \begin{bmatrix} \mathbf{f}_1(t,\mathbf{q},\mathbf{v}) \\ \mathbf{f}_2(t,\mathbf{q},\mathbf{v}) \\ \vdots \\ \mathbf{f}_n(t,\mathbf{q},\mathbf{v}) \end{bmatrix}$$

Now we're solving for the evolution of a single (though $3n$-dimensional!) vector

Generalized coordinates:

$$\frac{\mathrm{d}^2\mathbf{q}(t)}{\mathrm{d}t^2} = \mathbf{M}^{-1}\,\mathbf{f}(t, \mathbf{q}, \mathbf{v})$$

$$\Downarrow$$

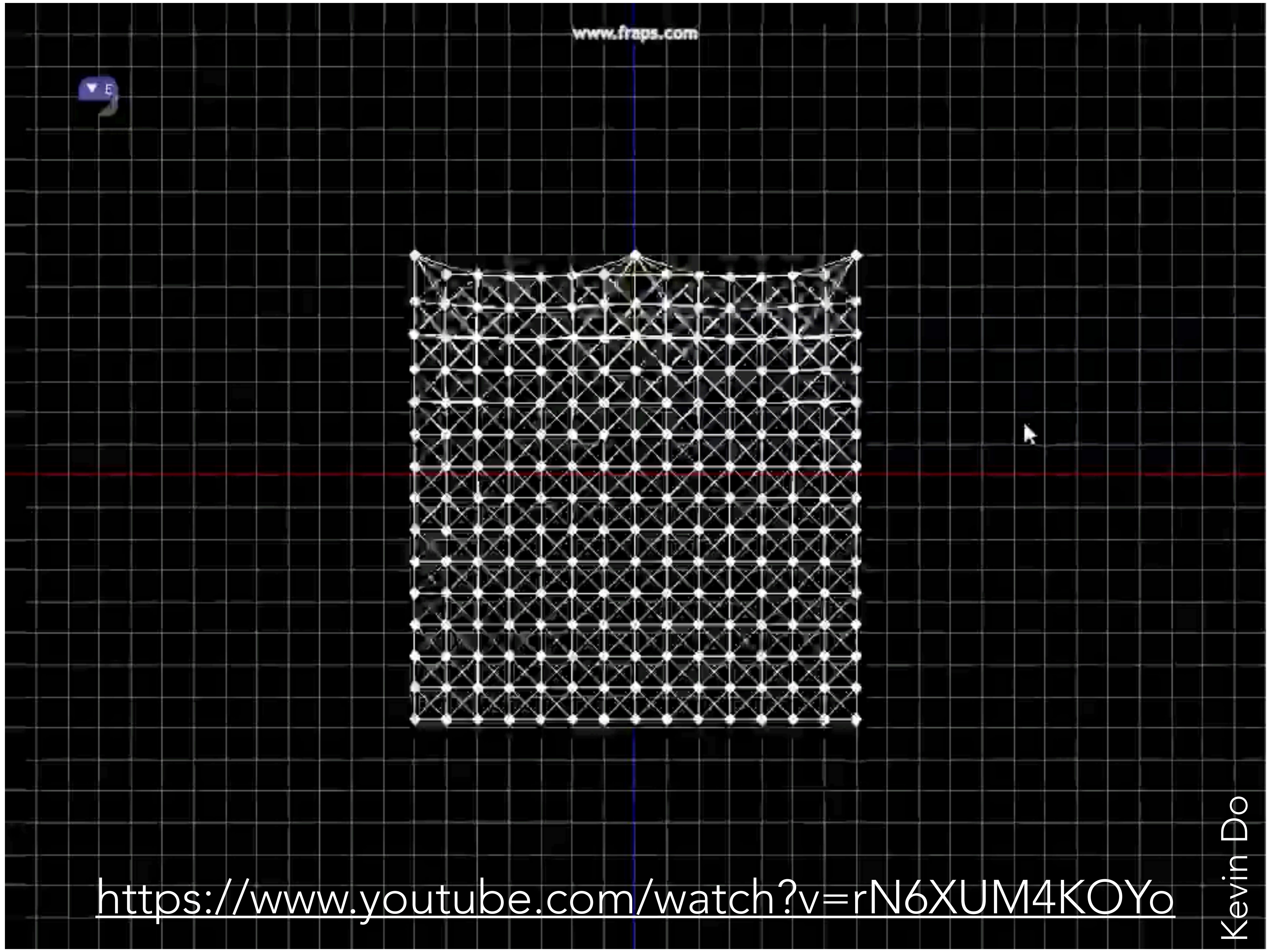
$$\mathbf{v}(t + \Delta t) = \mathbf{v}(t) + \mathbf{M}^{-1}\,\mathbf{f}(t, \mathbf{q}, \mathbf{v})\,\Delta t$$
$$\mathbf{q}(t + \Delta t) = \mathbf{q}(t) + \mathbf{v}(t + \Delta t)\,\Delta t$$

Simple! And generalizes to other things (e.g. rigid bodies) with few changes

Here's a problem you'll encounter:

Sometimes your simulation **blows up** for no apparent reason!

Why?



We have an ordinary differential equation

$$\ddot{\mathbf{q}} = \mathbf{M}^{-1}\,\mathbf{f}(t, \mathbf{q}, \dot{\mathbf{q}})$$

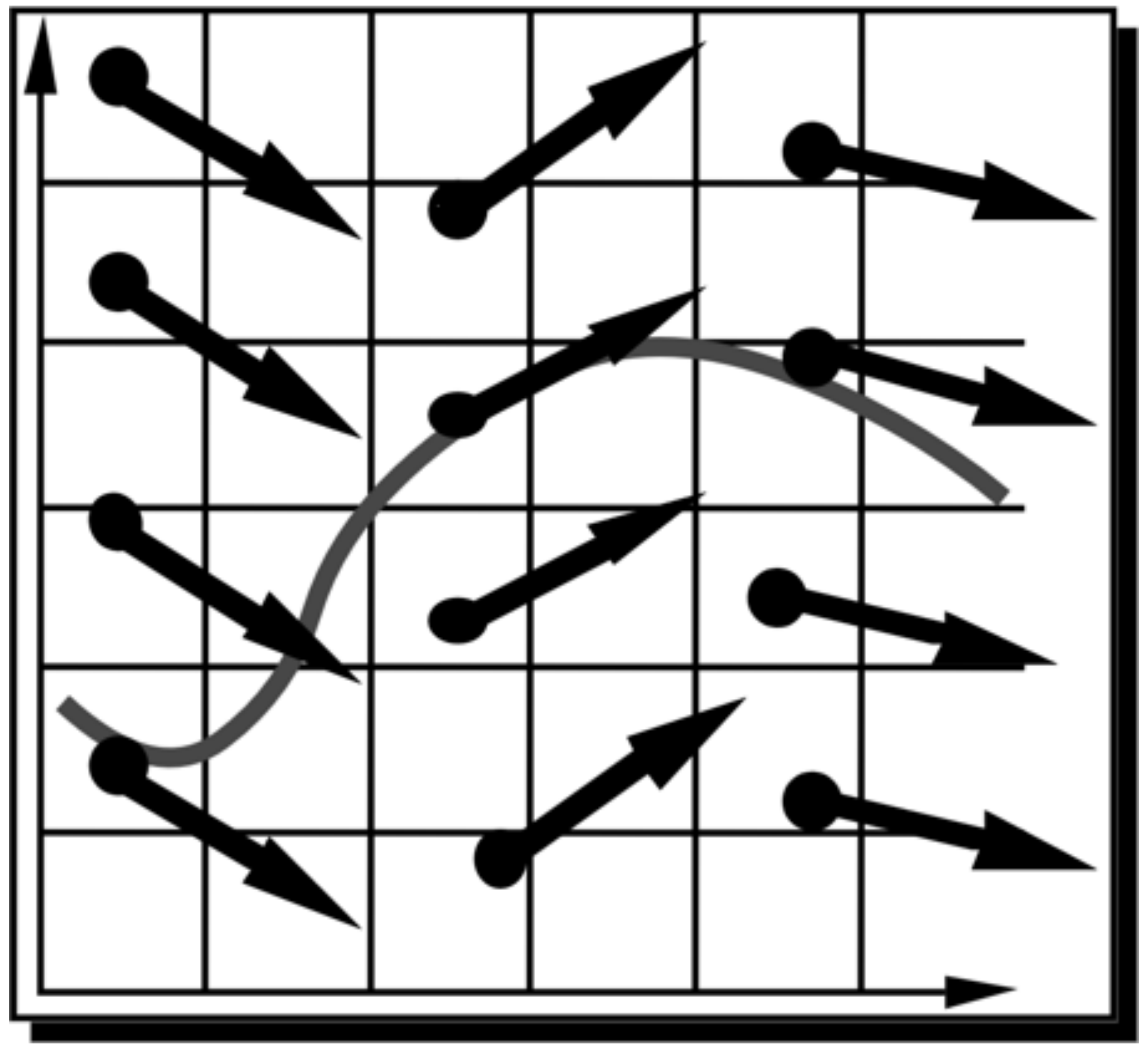
and are trying to solve an initial value problem:

Given $\mathbf{q}(0)$, $\dot{\mathbf{q}}(0)$, find $\mathbf{q}(t)$, $\dot{\mathbf{q}}(t)$ for $t > 0$.

Let's start by understanding this for a simple 1st-order ODE:

$$\dot{x}(t) = \phi(t, x(t))$$

Like a leaf in a river: if you are at position $x$ at time $t$, your velocity is $\phi(t, x)$



Witkin & Baraff 2001

# Explicit vs. implicit time integration

$$\dot{x}(t) = \phi(t, x(t))$$

- Simplest strategy: forward Euler method

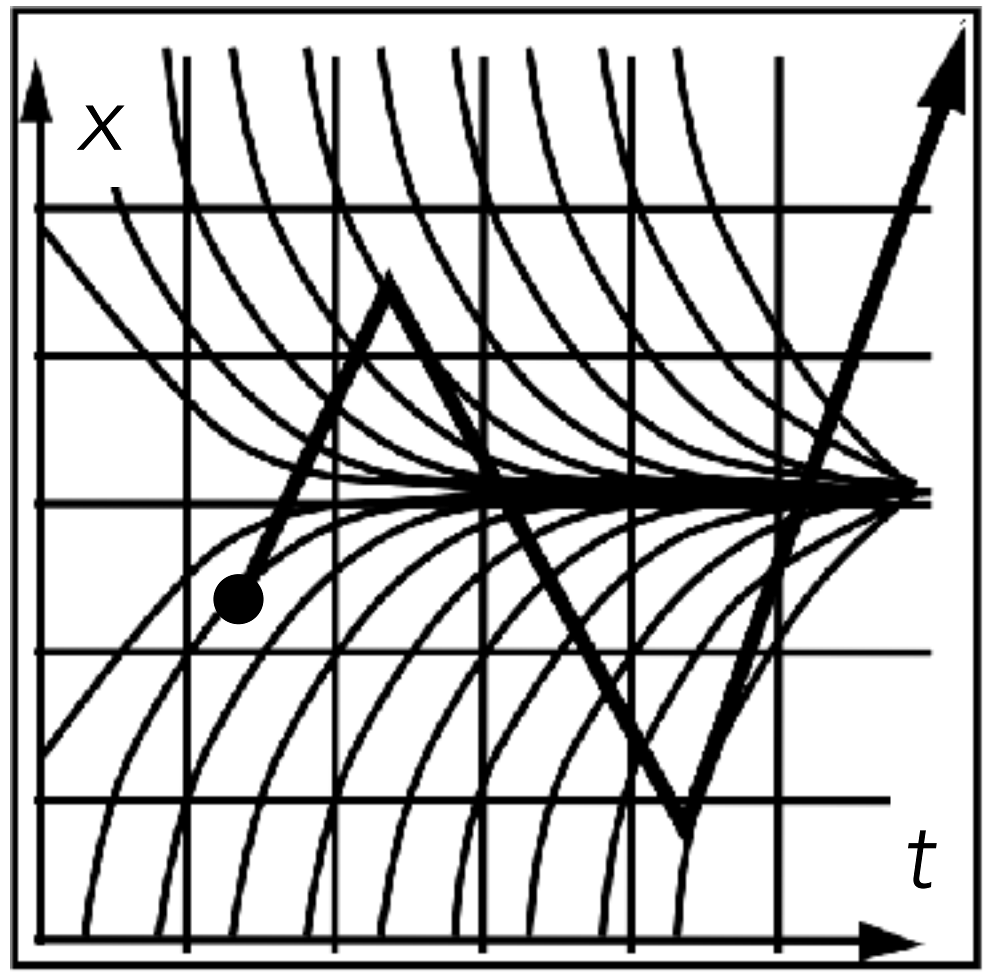
$$x_{n+1} = x_n + \phi(t_n, x_n)\,\Delta t$$

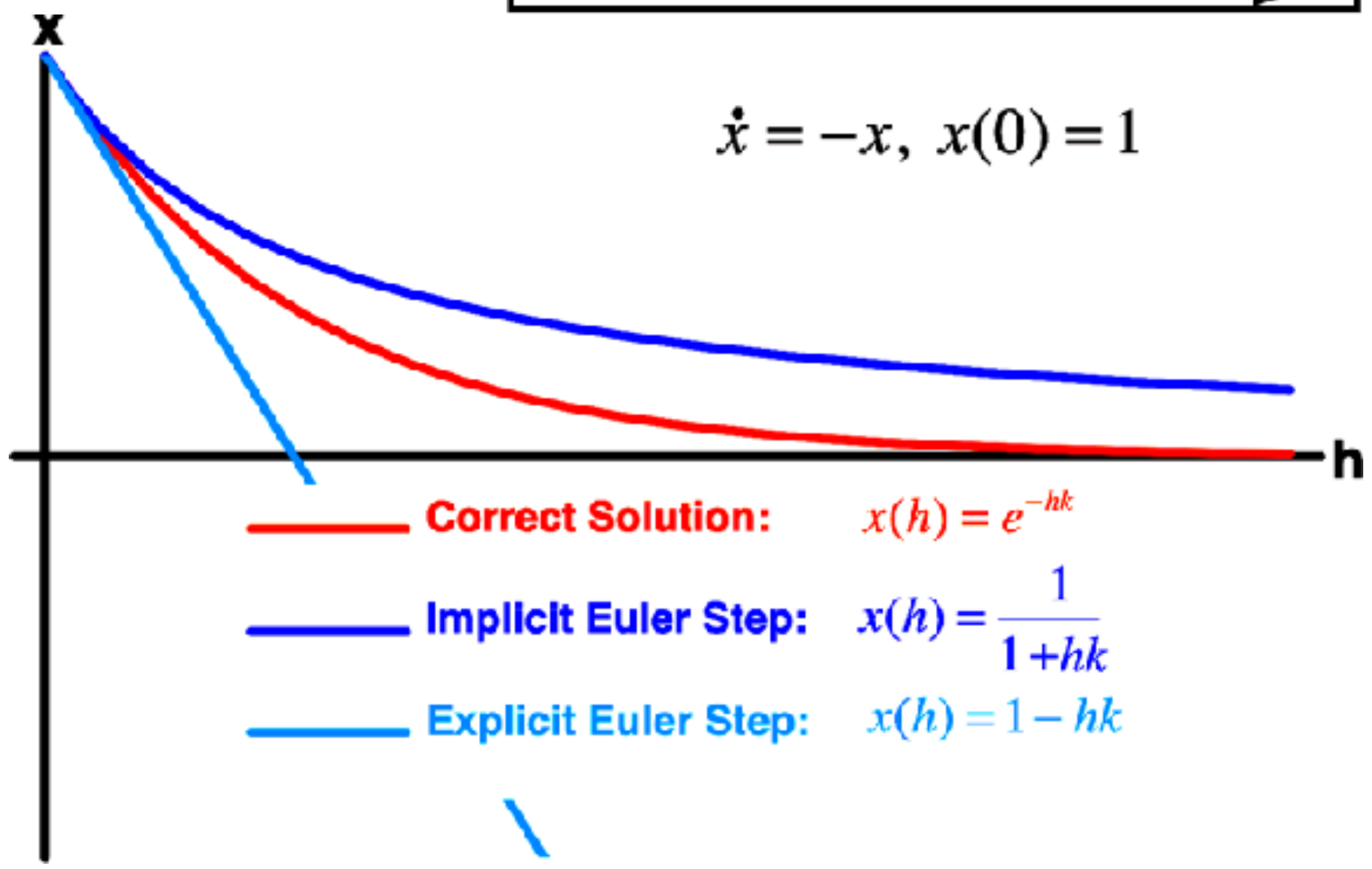
Tends to blow up if $\Delta t$ is too large

- Backward Euler:

$$x_{n+1} = x_n + \phi(t_{n+1}, x_{n+1})\,\Delta t$$

Implicit method: unknown $x_{n+1}$ appears on both sides!
But unconditionally stable for any $\Delta t$




Witkin & Baraff 2001

How do we apply all this to our 2nd-order ODE, $\ddot{\mathbf{q}} = \mathbf{M}^{-1}\,\mathbf{f}(\mathbf{q}, \dot{\mathbf{q}})$?

Reduce to 1st-order:

$$\dot{\mathbf{q}} = \mathbf{v}$$
$$\dot{\mathbf{v}} = \mathbf{M}^{-1}\,\mathbf{f}(\mathbf{q}, \mathbf{v})$$

Forward Euler:

$$\mathbf{q}_{n+1} = \mathbf{q}_n + \mathbf{v}_n\,\Delta t$$
$$\mathbf{v}_{n+1} = \mathbf{v}_n + \mathbf{M}^{-1}\,\mathbf{f}(\mathbf{q}_n, \mathbf{v}_n)\,\Delta t$$

Backward Euler:

$$\mathbf{q}_{n+1} = \mathbf{q}_n + \mathbf{v}_{n+1}\,\Delta t$$
$$\mathbf{v}_{n+1} = \mathbf{v}_n + \mathbf{M}^{-1}\,\mathbf{f}(\mathbf{q}_{n+1}, \mathbf{v}_{n+1})\,\Delta t$$

$$\mathbf{q}_{n+1} = 2\mathbf{q}_n - \mathbf{q}_{n-1} + \mathbf{M}^{-1}\,\mathbf{f}(\mathbf{q}_{n+1}, (\mathbf{q}_{n+1} - \mathbf{q}_n)/\Delta t)\,\Delta t^2$$

# Newton's method

How to solve a nonlinear system of equations $f(x) = 0$?

Start with a guess: $\tilde{x}$.

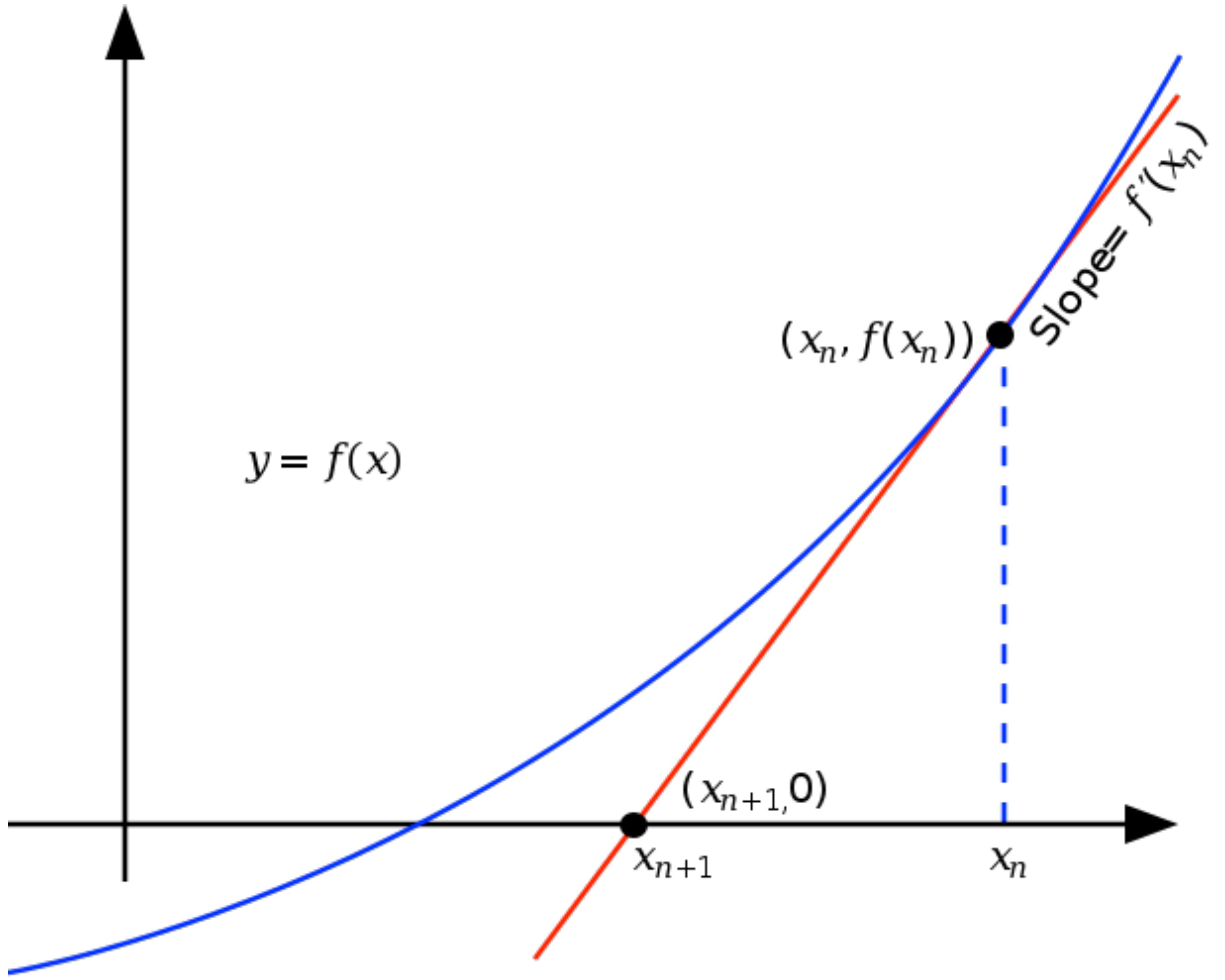
1. Approximate the problem near the guess:

$$0 = f(\tilde{x} + \Delta x) \approx f(\tilde{x}) + f'(\tilde{x})\,\Delta x$$

2. Solve the approximation exactly:

$$\Delta x = -(f'(\tilde{x}))^{-1} f(\tilde{x})$$

3. Improve the guess and repeat: $\tilde{x} \leftarrow \tilde{x} + \Delta x$



$$\mathbf{q}_{n+1} = \mathbf{q}_n + \mathbf{v}_{n+1}\,\Delta t$$
$$\mathbf{v}_{n+1} = \mathbf{v}_n + \mathbf{M}^{-1}\,\mathbf{f}(\mathbf{q}_{n+1}, \mathbf{v}_{n+1})\,\Delta t$$

Pick a guess $(\tilde{\mathbf{q}}, \tilde{\mathbf{v}})$. A natural choice is to start with $\tilde{\mathbf{q}} = \mathbf{q}_n$, $\tilde{\mathbf{v}} = \mathbf{v}_n$.

1. Approximate the problem:

$$(\tilde{\mathbf{q}}+\Delta\mathbf{q}) = \mathbf{q}_n + (\tilde{\mathbf{v}}+\Delta\mathbf{v})\,\Delta t$$

$$(\tilde{\mathbf{v}}+\Delta\mathbf{v}) = \mathbf{v}_n + \mathbf{M}^{-1}\,\mathbf{f}(\tilde{\mathbf{q}}+\Delta\mathbf{q}, \tilde{\mathbf{v}}+\Delta\mathbf{v})\,\Delta t$$

where $\mathbf{f}(\tilde{\mathbf{q}}+\Delta\mathbf{q}, \tilde{\mathbf{v}}+\Delta\mathbf{v}) \approx \mathbf{f}(\tilde{\mathbf{q}}, \tilde{\mathbf{v}}) + \frac{\partial\mathbf{f}}{\partial\mathbf{q}}(\tilde{\mathbf{q}}, \tilde{\mathbf{v}})\,\Delta\mathbf{q} + \frac{\partial\mathbf{f}}{\partial\mathbf{v}}(\tilde{\mathbf{q}}, \tilde{\mathbf{v}})\,\Delta\mathbf{v}$

2. Now the system is linear in $(\Delta\mathbf{q}, \Delta\mathbf{v})$. Plug into any linear solver. (Can simplify a bit first…)

**Note:** To carry this out, we must able to evaluate the **force Jacobians** $\frac{\partial\mathbf{f}}{\partial\mathbf{q}}$ and $\frac{\partial\mathbf{f}}{\partial\mathbf{v}}$.

# Rigid bodies

Degrees of freedom: Center of mass position **x**, rotation (matrix **R** or quaternion **q**)
…Basically just the body's coordinate system

Kinematics:

- (Linear) velocity: $\dot{\mathbf{x}} = \mathbf{v}$
- Angular velocity: $\boldsymbol{\omega}$

$$\dot{\mathbf{R}} = \begin{bmatrix} 0 & -\omega_z & \omega_y \\ \omega_z & 0 & -\omega_x \\ -\omega_y & \omega_x & 0 \end{bmatrix} \mathbf{R} \quad \text{or} \quad \dot{\mathbf{q}} = \frac{1}{2} \begin{bmatrix} q_x & -q_y & -q_z \\ q_w & q_z & -q_y \\ -q_z & q_w & q_x \\ q_y & -q_x & q_w \end{bmatrix} \boldsymbol{\omega}$$

Dynamics:

$$\dot{\mathbf{v}} = m^{-1}\,\mathbf{f}$$

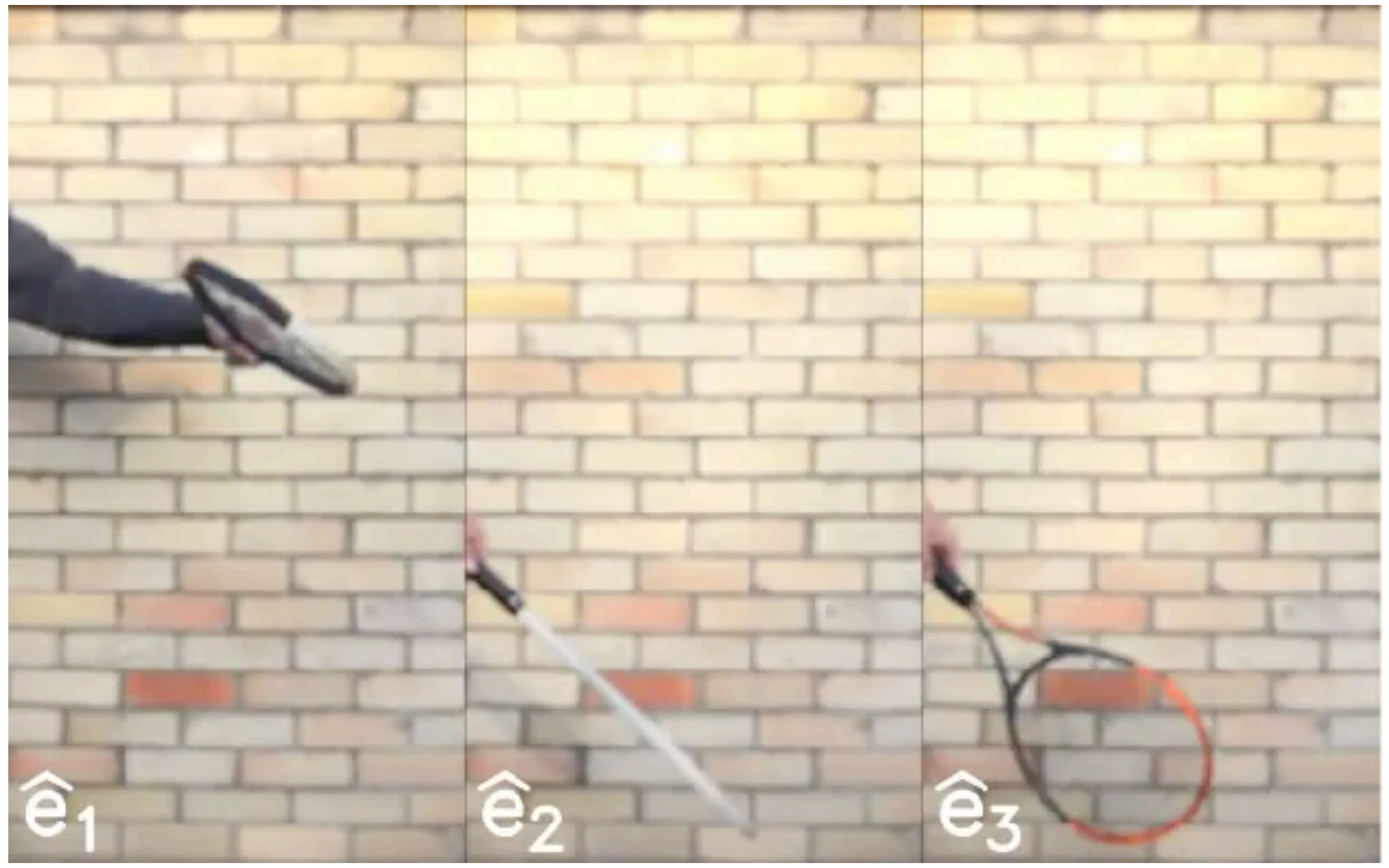
$$\dot{\boldsymbol{\omega}} = \mathbf{I}^{-1}\,(\boldsymbol{\tau} - \boldsymbol{\omega} \times \mathbf{I}\,\boldsymbol{\omega})$$

where $\mathbf{I}$ = moment of inertia, $\boldsymbol{\tau}$ = net torque = $\sum (\mathbf{p}_i - \mathbf{x}) \times \mathbf{f}_i$

$\boldsymbol{\omega} \times \mathbf{I}\,\boldsymbol{\omega}$ = "gyroscopic term" that makes things tumble

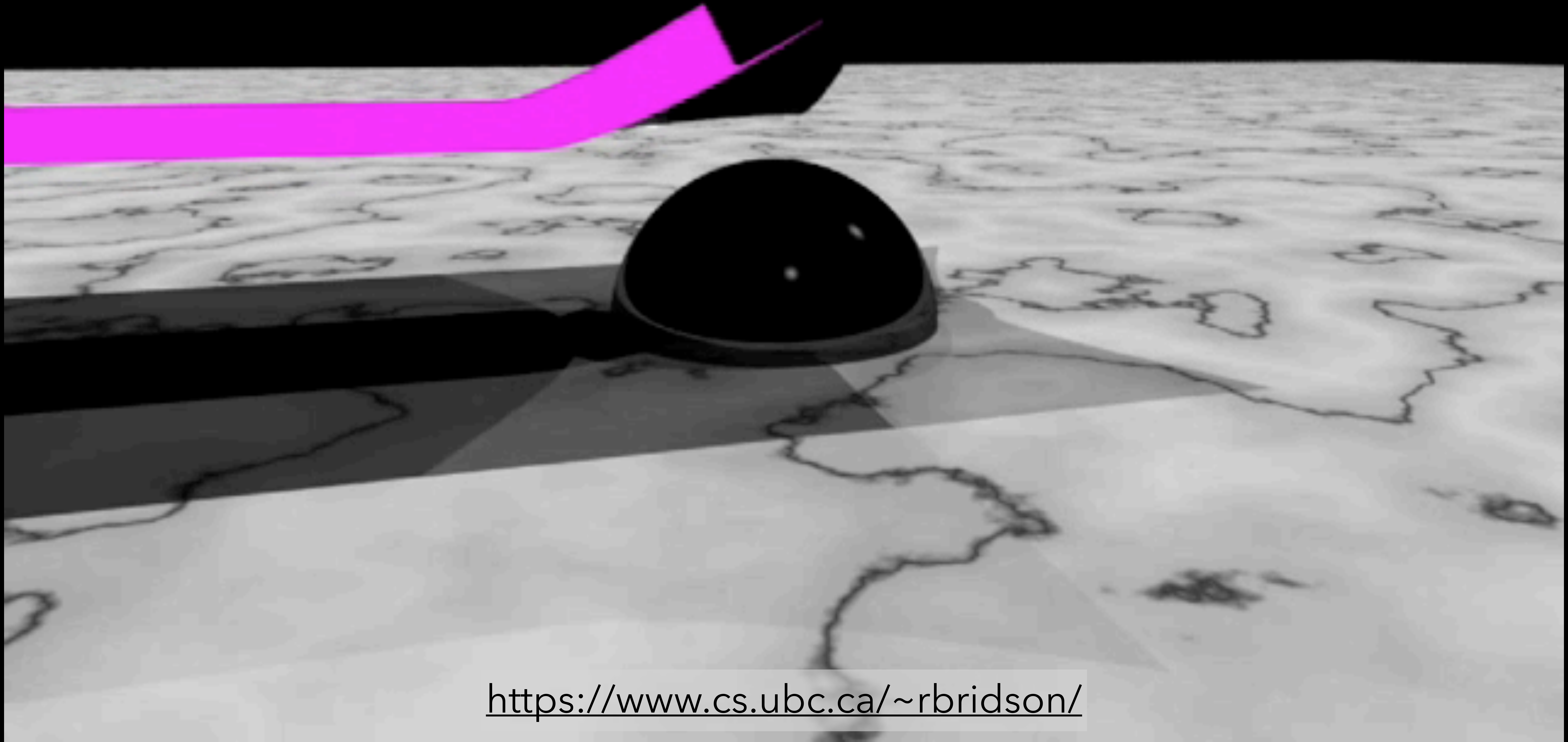
Simulation loop:

- Sum up forces $\mathbf{f}$ and torques $\boldsymbol{\tau}$
- Update velocities $\mathbf{v}$, $\boldsymbol{\omega}$
- Update DOFs $\mathbf{x}$, $\mathbf{q}$. Don't forget to normalize $\mathbf{q}$




https://commons.wikimedia.org/wiki/File:Tennis_racket_theorem.gif

# Collisions

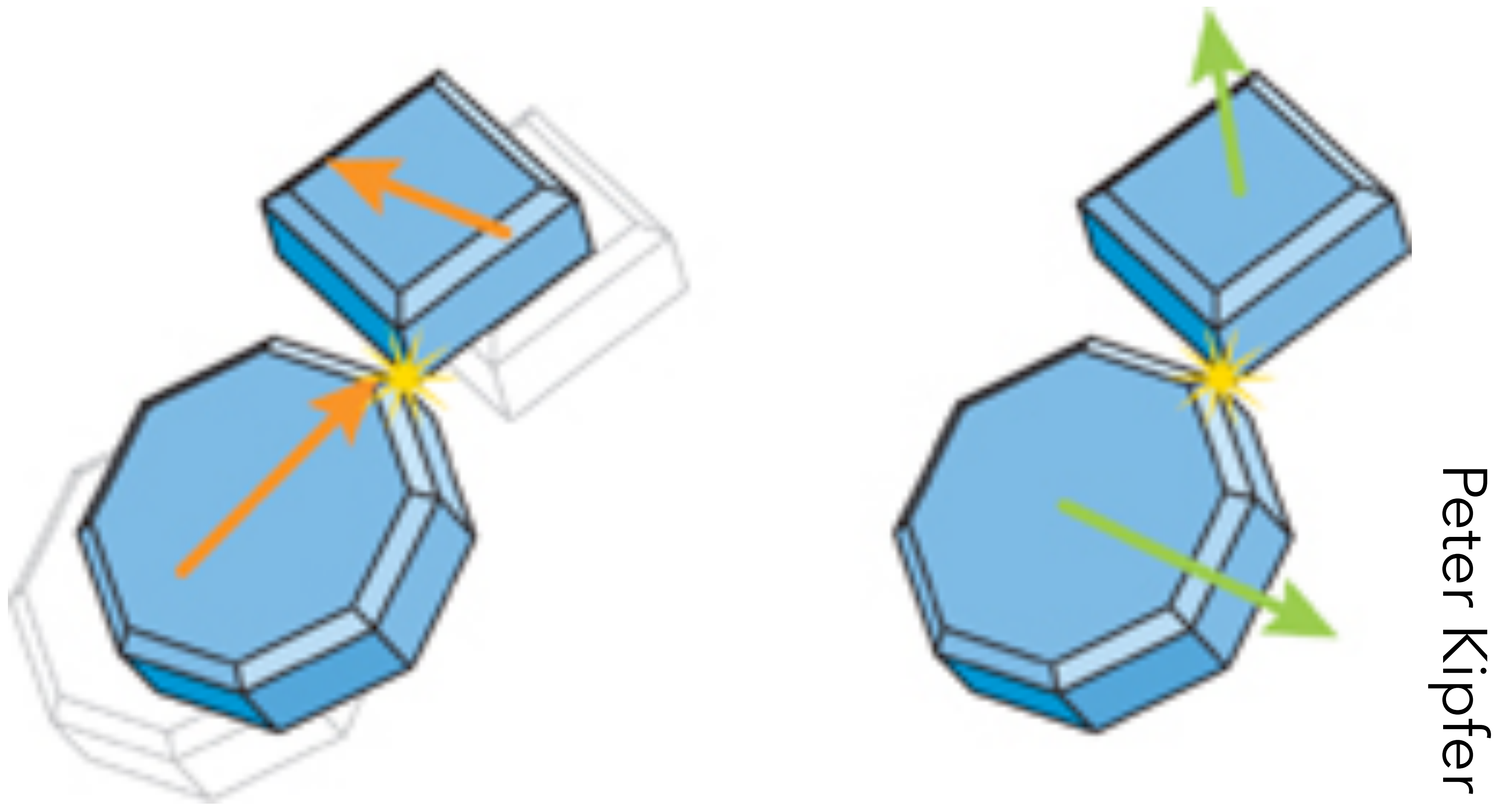


https://www.cs.ubc.ca/~rbridson/

Bridson et al. 2002

**Collision detection:** find out which particles / bodies / etc. are colliding

Purely a geometric problem

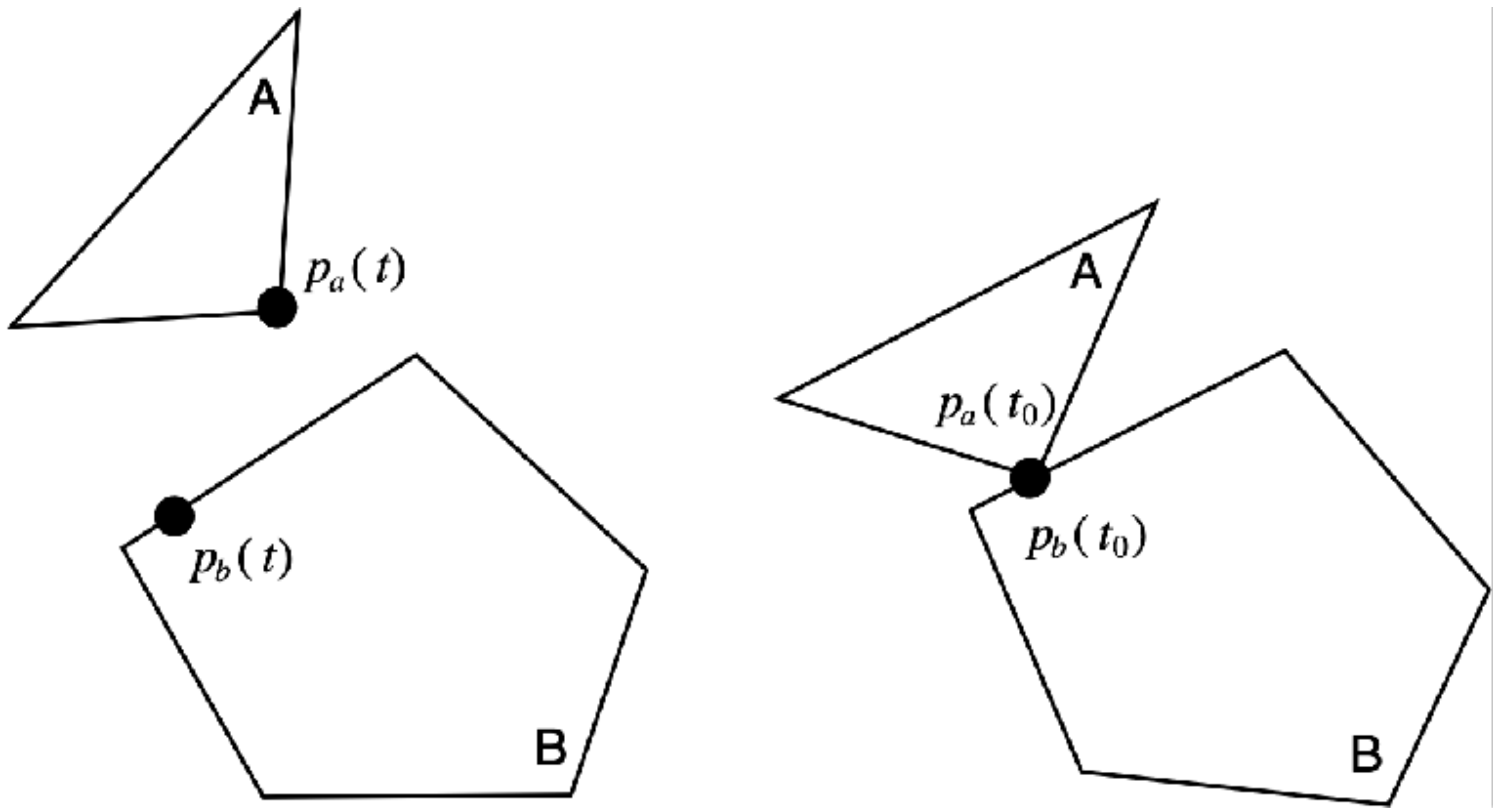



**Collision response:** figure out how to update their velocities / positions

Involves physics of contact forces, friction, etc.

Output of collision detection: **contact pairs**

- Point $\mathbf{p}_a$ on one body
- Point $\mathbf{p}_b$ on other body
- Contact normal $\mathbf{n}$
- Time of impact $t^*$

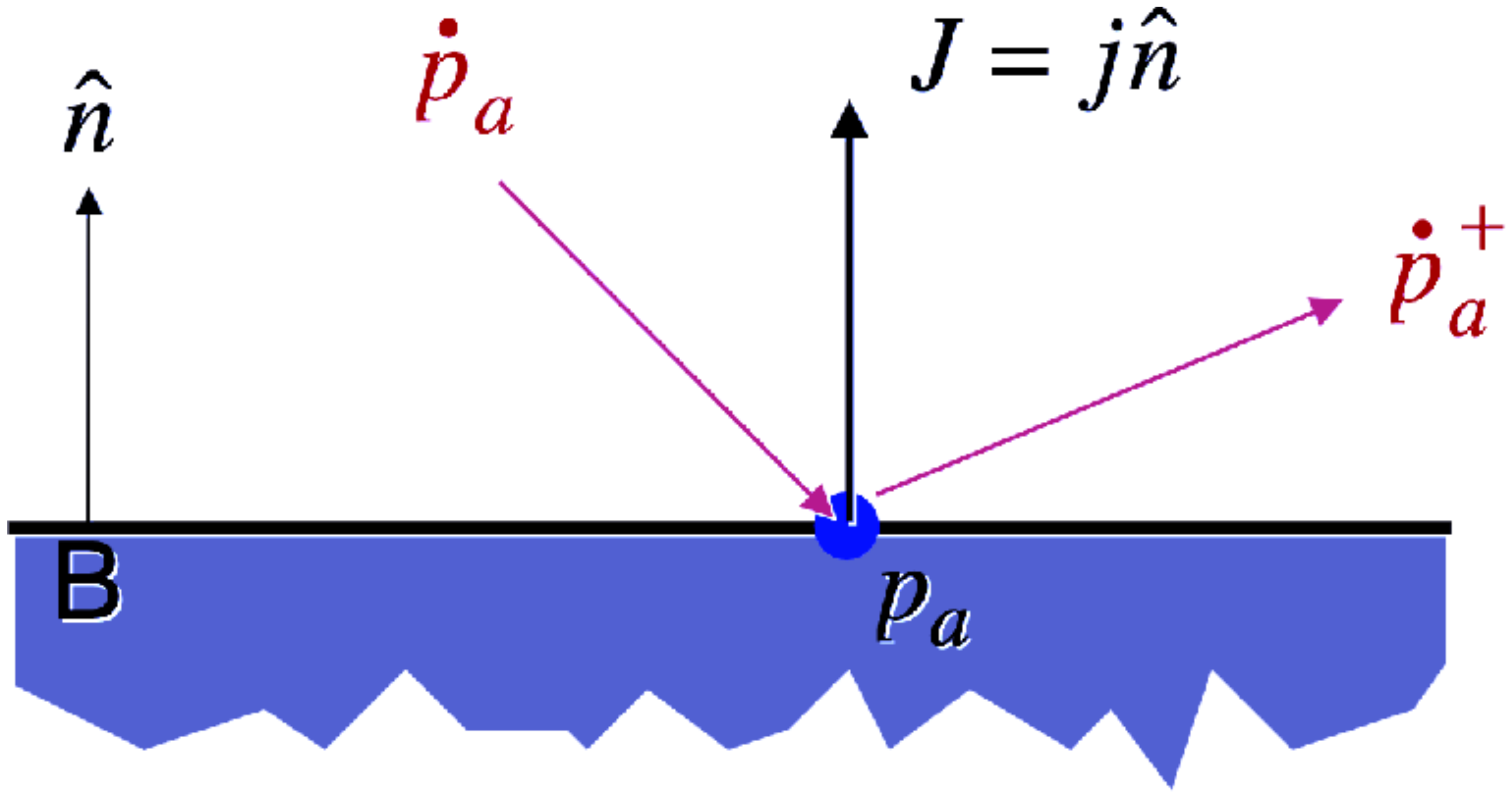



Witkin & Baraff 2001

# Collision resolution

Two components:

- Normal force (prevents interpenetration)
- Frictional force (opposes tangential sliding)



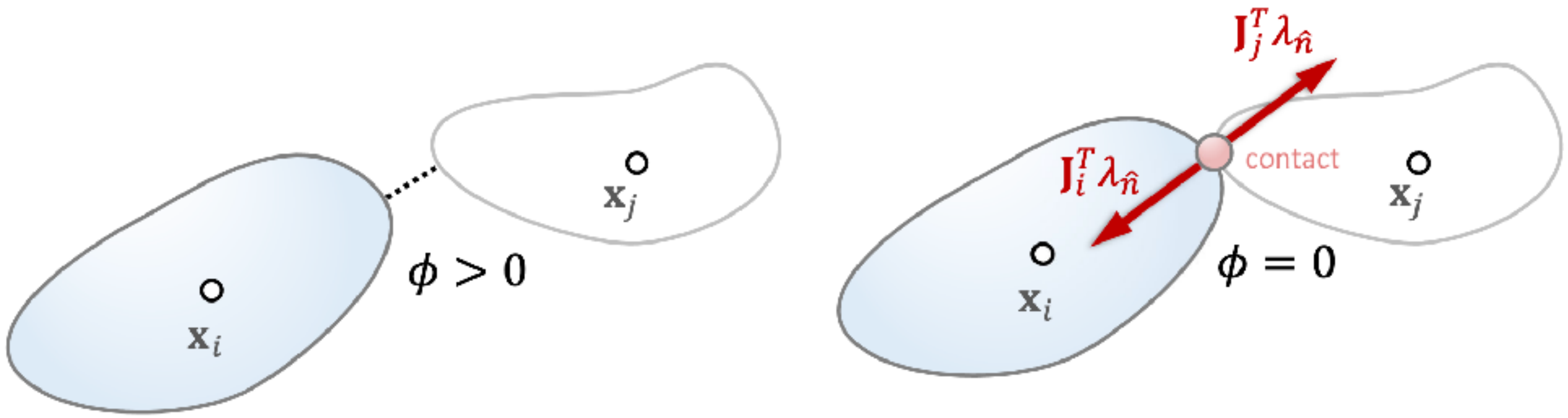

Witkin & Baraff 2001

Actually, collision forces change velocity over
an extremely very short time → treat as an instantaneous impulse

$$\mathbf{v}^+ = \mathbf{v} + m^{-1}\,\mathbf{j}$$

The normal component is like a constraint that prevents interpenetration.

Define a **gap function** $\varphi(\mathbf{q})$ which measures the distance between the bodies




Andrews et al. 2002

Constraint: $\varphi(\mathbf{q}) \geq 0$

Normal impulse: $\mathbf{j} = \lambda\, \nabla\varphi(\mathbf{q})$, $\lambda \geq 0$ (no sticking)

**Complementarity**: if $\varphi(\mathbf{q}) > 0$ then $\lambda = 0$, if $\lambda > 0$ then $\varphi(\mathbf{q}) = 0$

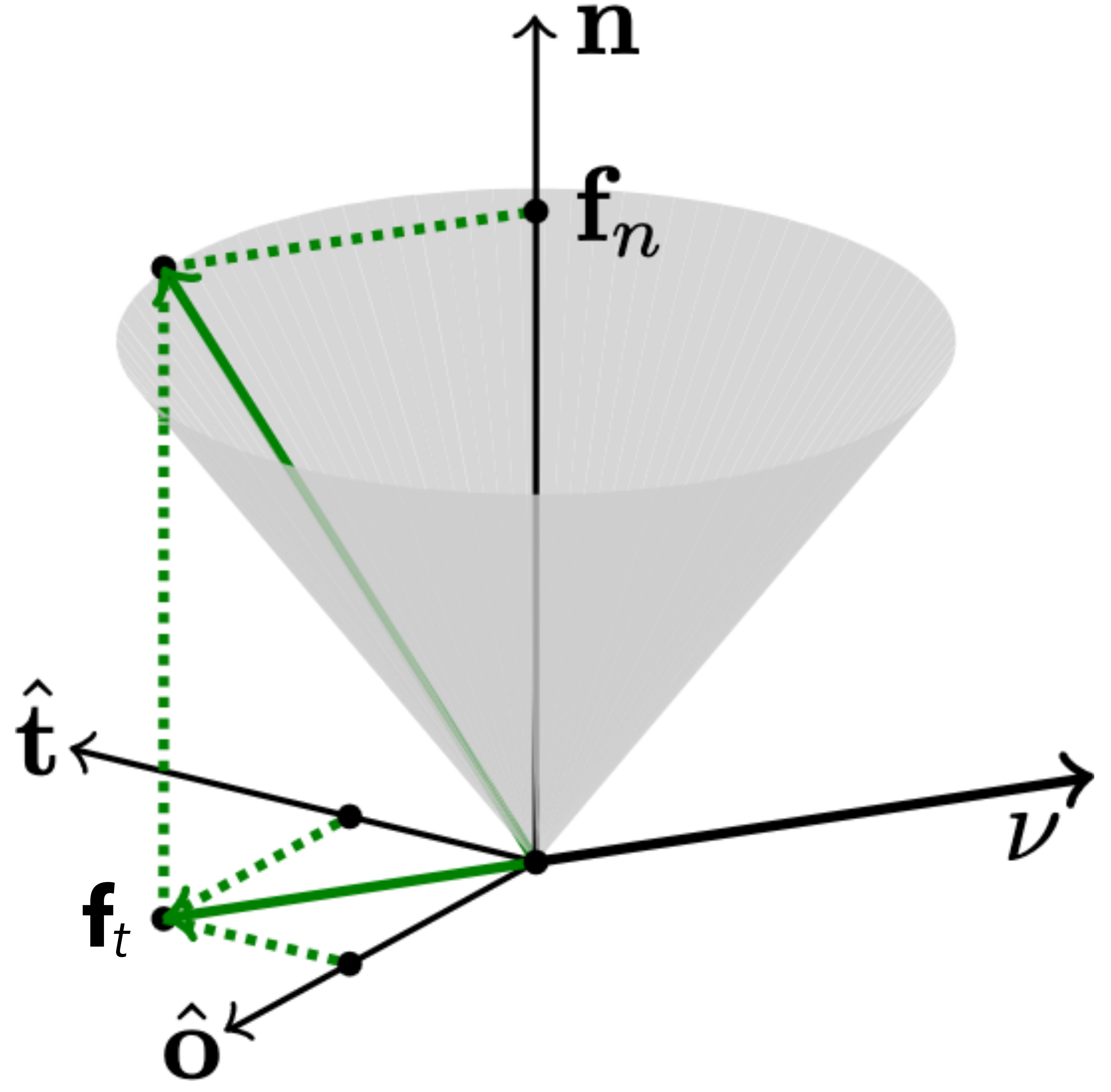
$$0 \leq \varphi(\mathbf{q}) \quad \perp \quad \lambda \geq 0$$

Friction is described by Coulomb's law

$$\|\mathbf{f}_t\| \leq \mu\, \mathbf{f}_n$$

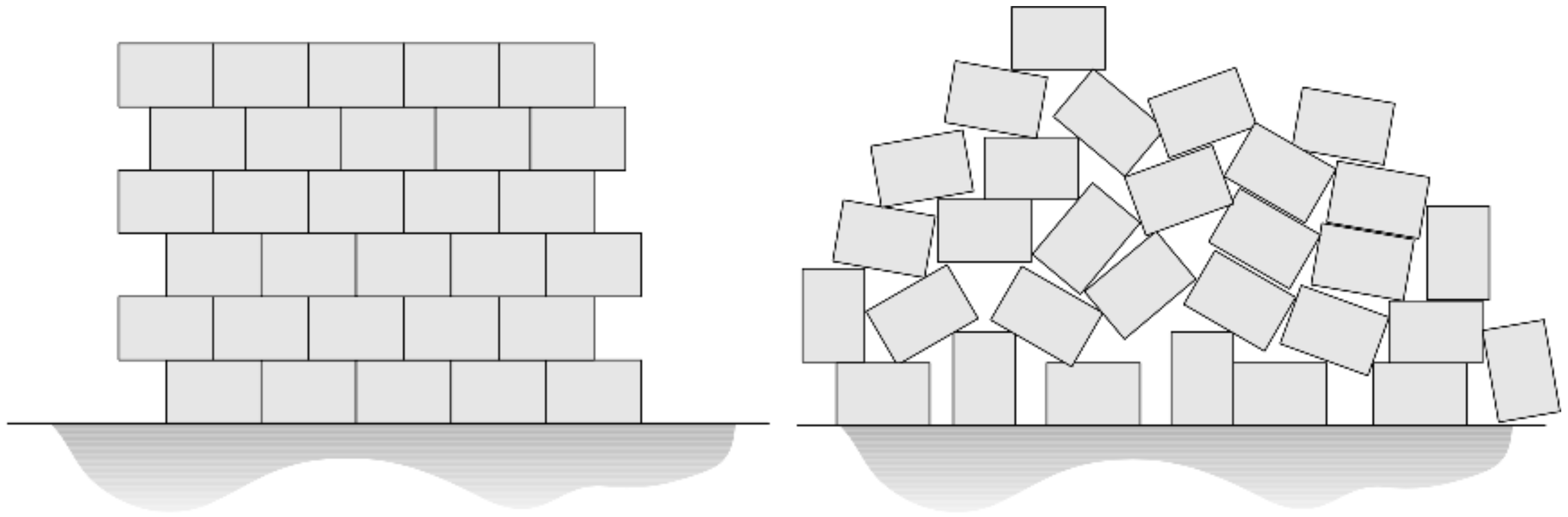
**Maximum dissipation principle**: Frictional force takes the value which dissipates as much kinetic energy as possible.

1. If $\|\mathbf{v}_t\| > 0$ (slipping) then $\mathbf{f}_t = -(\mu\, \mathbf{f}_n)\, \hat{\mathbf{v}}_t$
2. If $\|\mathbf{v}_t\| = 0$ (sticking) then $\mathbf{f}_t$ is any force in friction cone

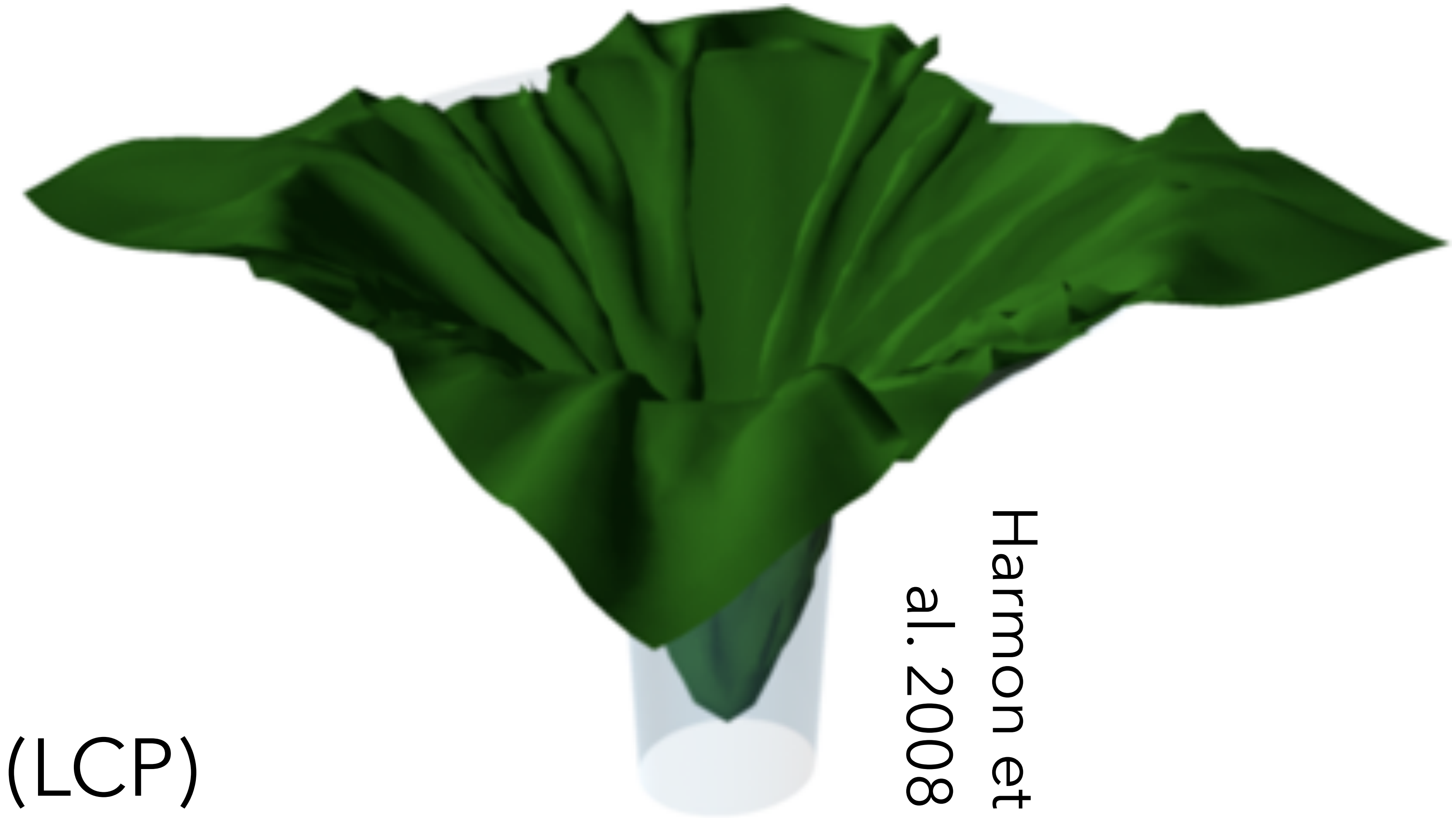



Bender et al. 2012

# Multi-contact problems (harder!)

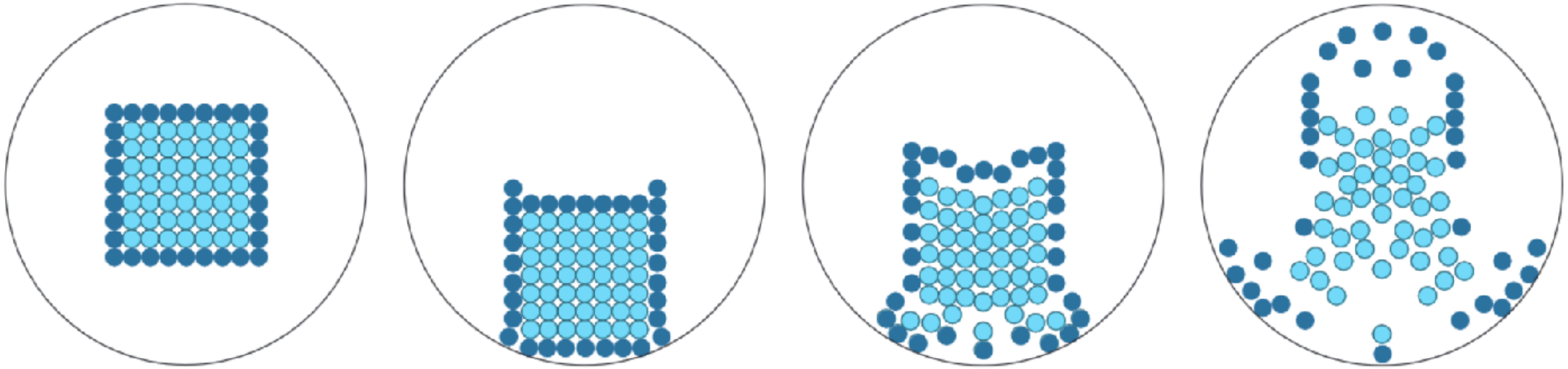


Erleben 2007



Harmon et al. 2008

Often modeled as a linear complementarity problem (LCP)

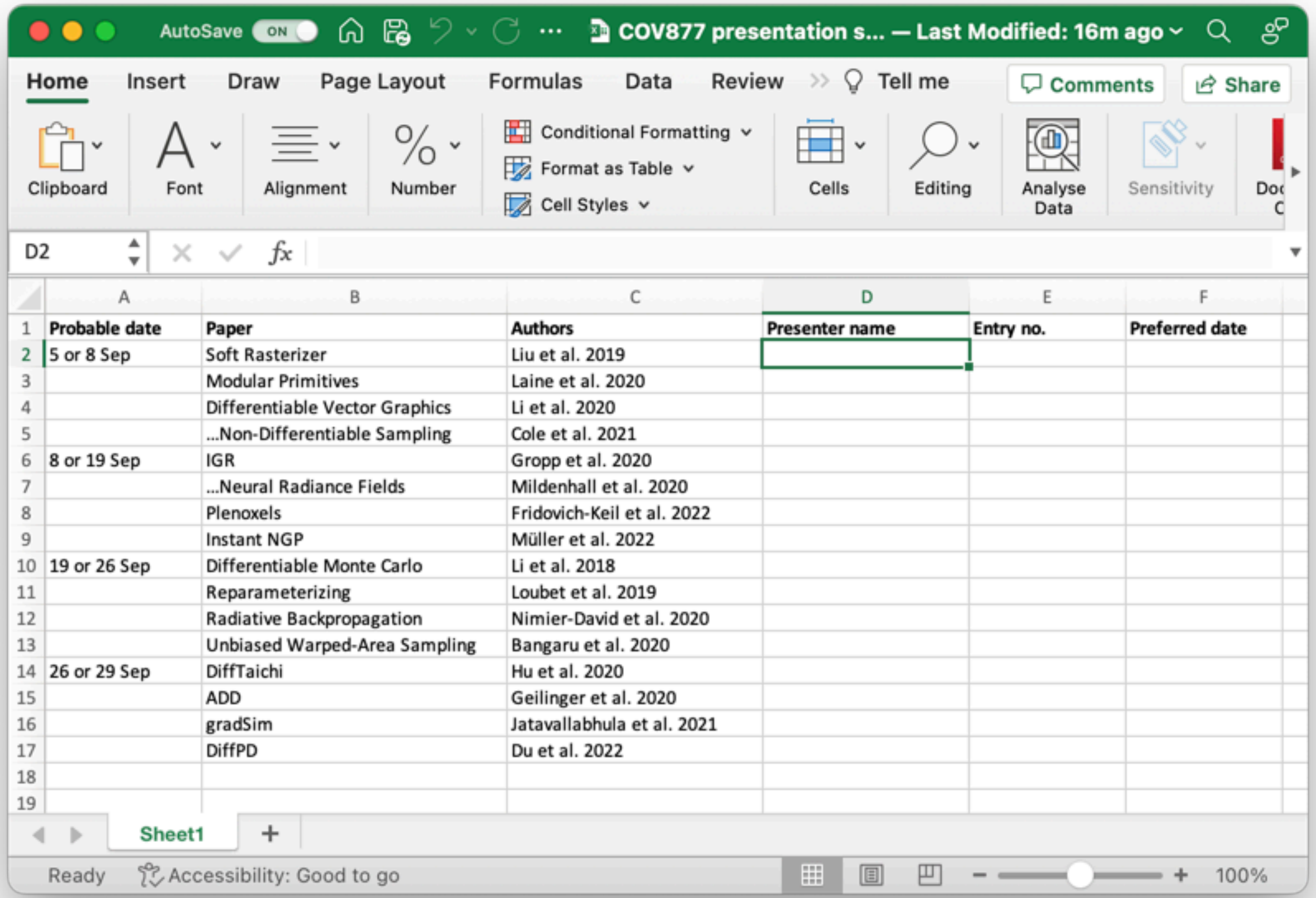


Smith et al. 2012

# Differentiable simulation

**Reminder:**

- Sign-up sheet posted on Teams
- Enter your name by end of today! Late sign-ups will be forced to present next week itself :)



| | A | B | C | D | E | F |
|---|---|---|---|---|---|---|
| 1 | Probable date | Paper | Authors | Presenter name | Entry no. | Preferred date |
| 2 | 5 or 8 Sep | Soft Rasterizer | Liu et al. 2019 | | | |
| 3 | | Modular Primitives | Laine et al. 2020 | | | |
| 4 | | Differentiable Vector Graphics | Li et al. 2020 | | | |
| 5 | | ...Non-Differentiable Sampling | Cole et al. 2021 | | | |
| 6 | 8 or 19 Sep | IGR | Gropp et al. 2020 | | | |
| 7 | | ...Neural Radiance Fields | Mildenhall et al. 2020 | | | |
| 8 | | Plenoxels | Fridovich-Keil et al. 2022 | | | |
| 9 | | Instant NGP | Müller et al. 2022 | | | |
| 10 | 19 or 26 Sep | Differentiable Monte Carlo | Li et al. 2018 | | | |
| 11 | | Reparameterizing | Loubet et al. 2019 | | | |
| 12 | | Radiative Backpropagation | Nimier-David et al. 2020 | | | |
| 13 | | Unbiased Warped-Area Sampling | Bangaru et al. 2020 | | | |
| 14 | 26 or 29 Sep | DiffTaichi | Hu et al. 2020 | | | |
| 15 | | ADD | Geilinger et al. 2020 | | | |
| 16 | | gradSim | Jatavallabhula et al. 2021 | | | |
| 17 | | DiffPD | Du et al. 2022 | | | |
| 18 | | | | | | |
| 19 | | | | | | |

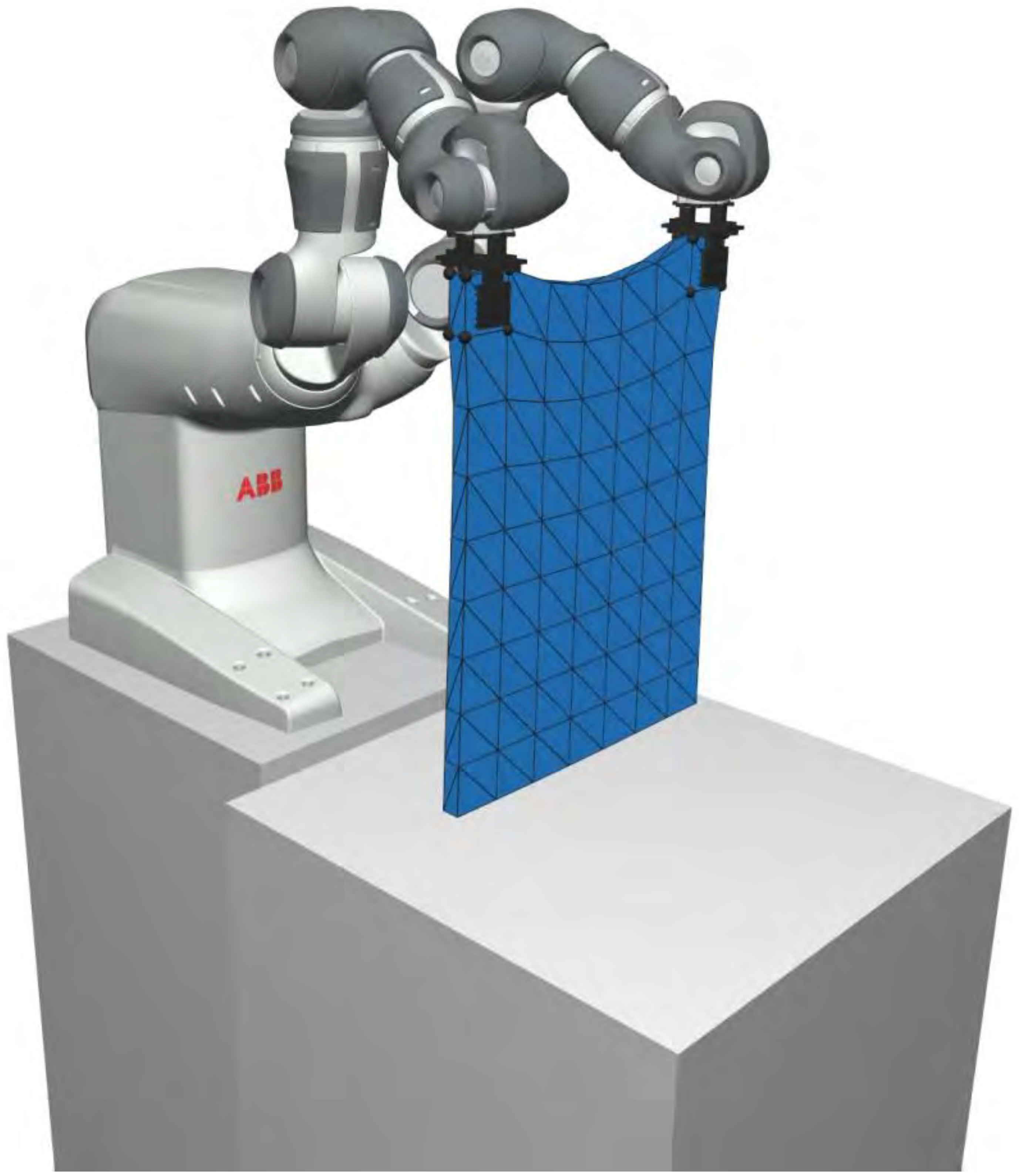
Suppose we want to do quasistatics: Given the parameters **p**, what is the equilibrium configuration of the body $\mathbf{x}^*$?

Simulator gives us forces $\mathbf{f}(\mathbf{x}; \mathbf{p})$

Equilibrium configuration is implicitly defined by

$$\mathbf{f}(\mathbf{x}^*; \mathbf{p}) = \mathbf{0}$$

How to find **p** to to minimize some objective $O(\mathbf{x}^*, \mathbf{p})$?




Coros et al. 2021

# Implicit differentiation

$$\mathbf{f}(\mathbf{x}^*; \mathbf{p}) = \mathbf{0}$$

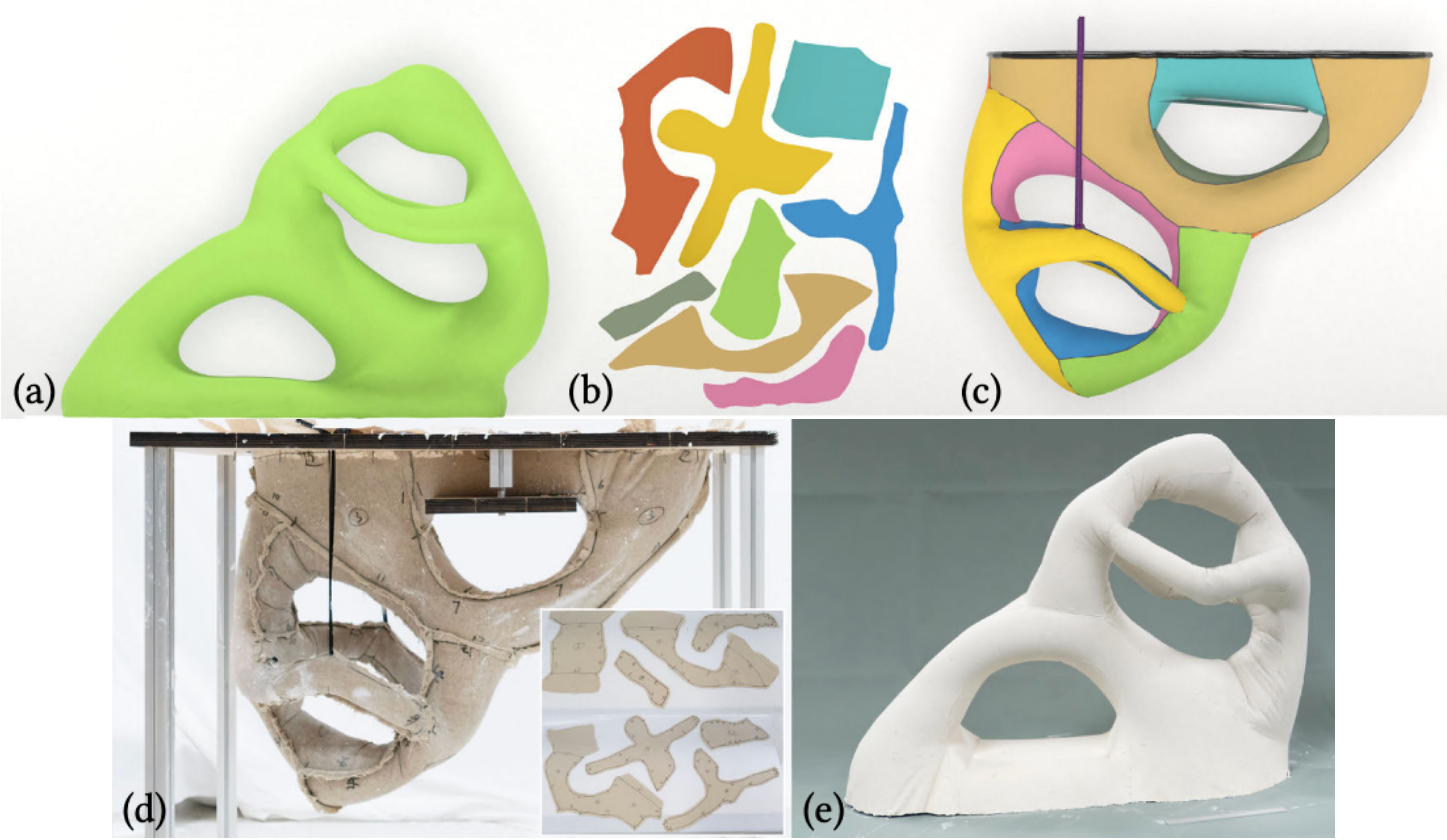
Differentiate both sides with respect to **p**:

$$\frac{\mathrm{d}}{\mathrm{d}\mathbf{p}}\mathbf{f}(\mathbf{x}^*; \mathbf{p}) = \mathbf{0} = \frac{\partial \mathbf{f}}{\partial \mathbf{x}}\frac{\mathrm{d}\mathbf{x}^*}{\mathrm{d}\mathbf{p}} + \frac{\partial \mathbf{f}}{\partial \mathbf{p}}$$

$$\frac{\mathrm{d}\mathbf{x}^*}{\mathrm{d}\mathbf{p}} = -\left(\frac{\partial \mathbf{f}}{\partial \mathbf{x}}\right)^{-1}\frac{\partial \mathbf{f}}{\partial \mathbf{p}}$$

So now we can get the gradient of the objective $O(\mathbf{x}^*, \mathbf{p})$:

$$\frac{\mathrm{d}O}{\mathrm{d}\mathbf{p}} = \frac{\partial O}{\partial \mathbf{x}^*}\frac{\mathrm{d}\mathbf{x}^*}{\mathrm{d}\mathbf{p}} + \frac{\partial O}{\partial \mathbf{p}}$$

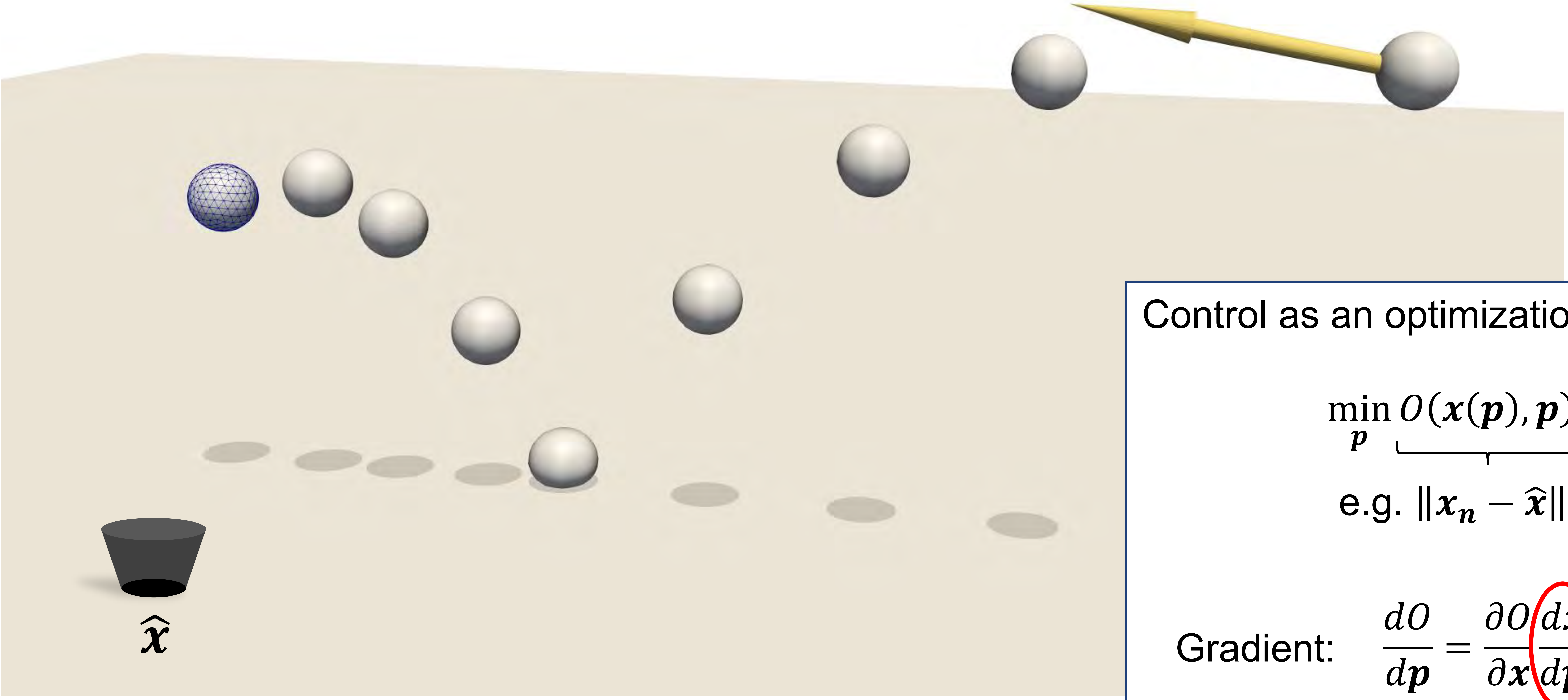



Zhang et al., “Computational Design of Fabric Formwork”, SIGGRAPH 2019

## What about dynamics?

Trajectory $\mathbf{x}(\mathbf{p}) = [\mathbf{x}_0(\mathbf{p}), \mathbf{x}_1(\mathbf{p}), \ldots, \mathbf{x}_n(\mathbf{p})]$

input parameters $\boldsymbol{p}$
(e.g. initial velocity)




Control as an optimization problem:

$$\min_{\boldsymbol{p}} \underbrace{O(\boldsymbol{x}(\boldsymbol{p}), \boldsymbol{p})}$$

e.g. $\|\boldsymbol{x_n} - \hat{\boldsymbol{x}}\|_2^2$

Gradient: $$\frac{dO}{d\boldsymbol{p}} = \frac{\partial O}{\partial \boldsymbol{x}} \frac{d\boldsymbol{x}}{d\boldsymbol{p}} + \frac{\partial O}{\partial \boldsymbol{p}}$$

Coros et al. 2021

Simulation output:

$$\boldsymbol{x} = \begin{bmatrix} \\ \\ \\ \\ \end{bmatrix} \rightarrow \underbrace{\begin{bmatrix} \mathbf{M}\ddot{\boldsymbol{x}}_1 - F(\boldsymbol{x}_1, \boldsymbol{p}) \\ \mathbf{M}\ddot{\boldsymbol{x}}_2 - F(\boldsymbol{x}_2, \boldsymbol{p}) \\ \vdots \\ \mathbf{M}\ddot{\boldsymbol{x}}_n - F(\boldsymbol{x}_n, \boldsymbol{p}) \end{bmatrix}}_{\boldsymbol{G}(\boldsymbol{x}(\boldsymbol{p}), \boldsymbol{p})}$$

Where:

- $\boldsymbol{p}$ is the input driving the simulation
- what we want is $\frac{d\boldsymbol{x}}{d\boldsymbol{p}}$
- $\boldsymbol{x}(\boldsymbol{p})$ does not have an analytic form

But:

- for *any* $\boldsymbol{p}$, we compute $\boldsymbol{x}(\boldsymbol{p})$ such that $\boldsymbol{G}(\boldsymbol{x}(\boldsymbol{p}), \boldsymbol{p}) = 0$

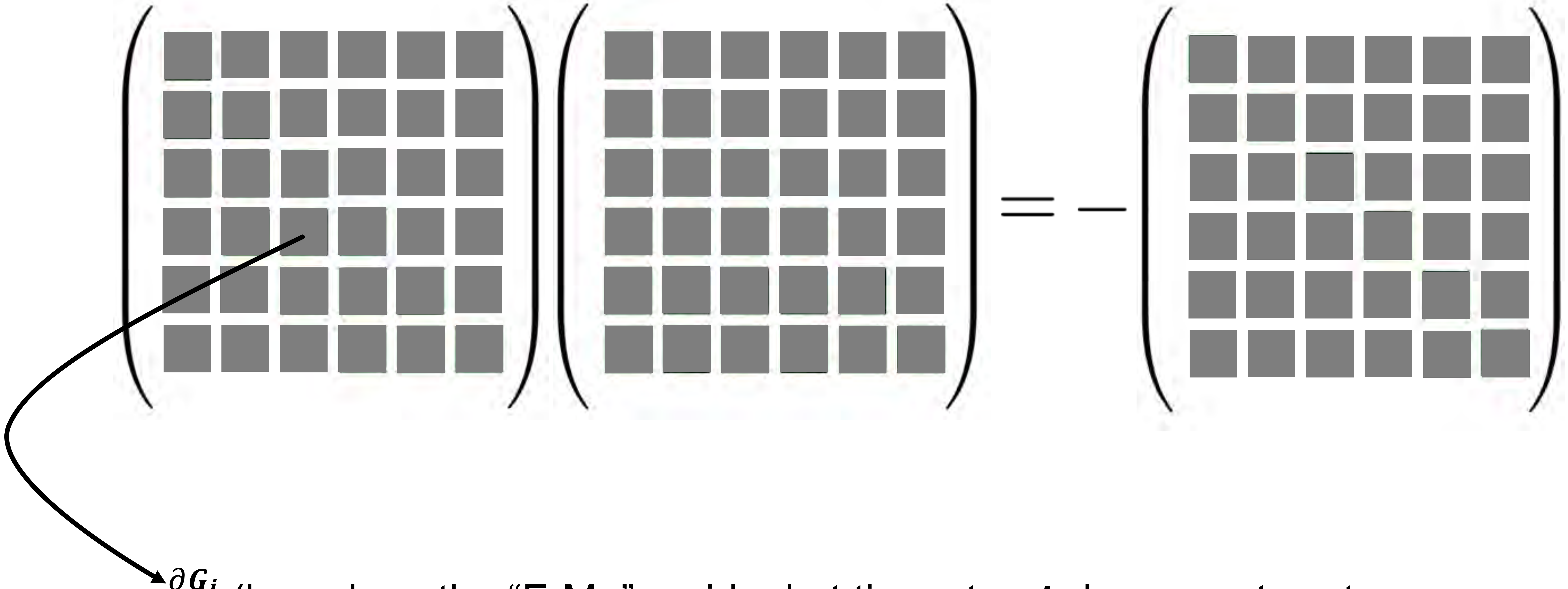
Coros et al. 2021

$$\boldsymbol{G}(\boldsymbol{x}(\boldsymbol{p}),\boldsymbol{p}) = \boldsymbol{0}, \forall \boldsymbol{p}$$

$$\frac{d\boldsymbol{G}}{d\boldsymbol{p}} = \boldsymbol{0} = \frac{\partial \boldsymbol{G}}{\partial \boldsymbol{x}}\frac{d\boldsymbol{x}}{d\boldsymbol{p}} + \frac{\partial \boldsymbol{G}}{\partial \boldsymbol{p}}$$

$$\frac{d\boldsymbol{x}}{d\boldsymbol{p}} = -\left(\frac{\partial \boldsymbol{G}}{\partial \boldsymbol{x}}\right)^{-1}\frac{\partial \boldsymbol{G}}{\partial \boldsymbol{p}}$$

$$\frac{\partial \boldsymbol{G}}{\partial \boldsymbol{x}} \qquad \frac{\mathrm{d}\boldsymbol{x}}{\mathrm{d}\boldsymbol{p}} \qquad \frac{\partial \boldsymbol{G}}{\partial \boldsymbol{p}}$$



$\frac{\partial \boldsymbol{G}_i}{\partial \boldsymbol{x}_j}$ (how does the “F-Ma” residual at time step $\boldsymbol{i}$ change wrt system configuration at time step $\boldsymbol{j}$)

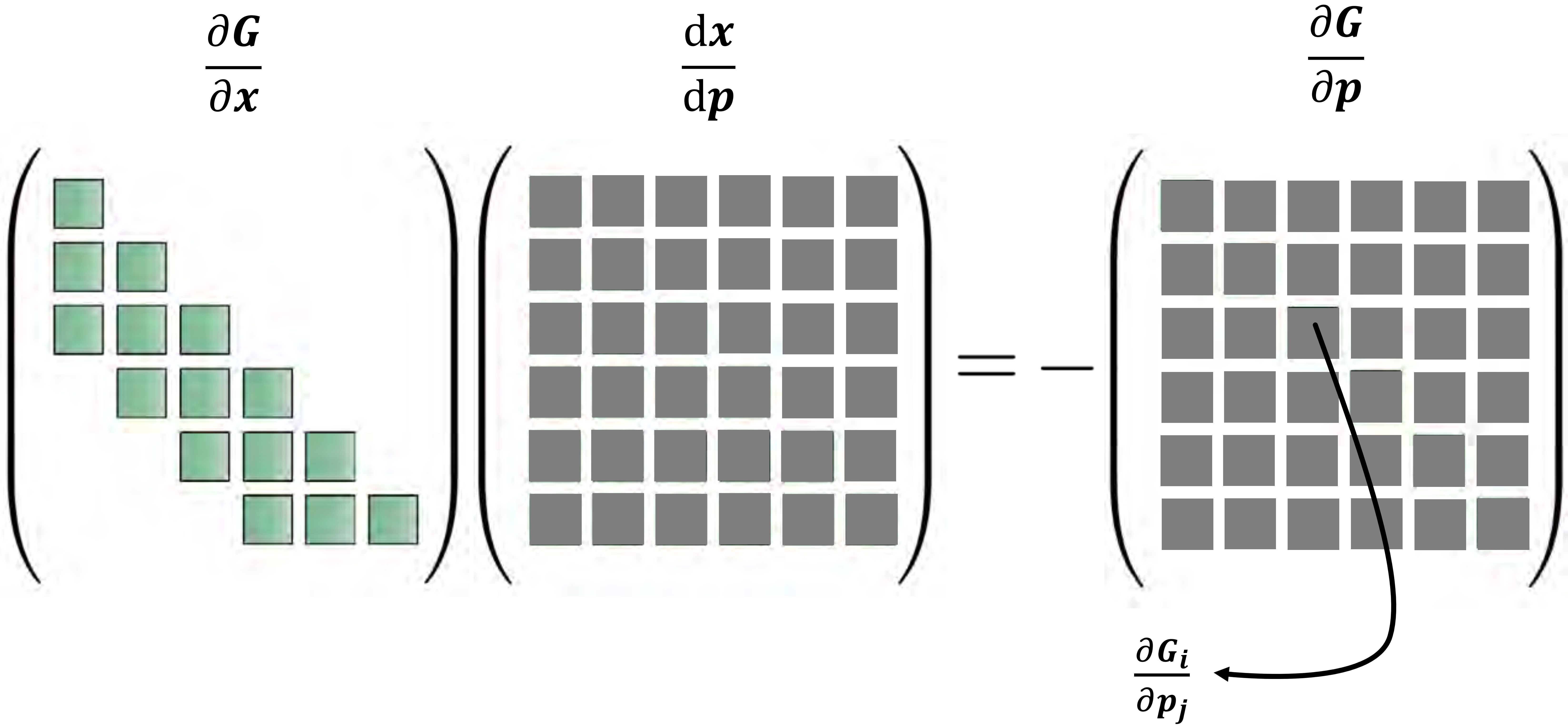
$$\boldsymbol{G}_{\mathrm{k}} = M\frac{\boldsymbol{x}_k - 2\boldsymbol{x}_{k-1} + \boldsymbol{x}_{k-2}}{h^2} - F(\boldsymbol{x}_k, \boldsymbol{p})$$

Coros et al. 2021

$$\boldsymbol{G}(\boldsymbol{x}(\boldsymbol{p}), \boldsymbol{p}) = \boldsymbol{0}, \forall \boldsymbol{p}$$

$$\frac{d\boldsymbol{G}}{d\boldsymbol{p}} = \boldsymbol{0} = \frac{\partial \boldsymbol{G}}{\partial \boldsymbol{x}} \frac{d\boldsymbol{x}}{d\boldsymbol{p}} + \frac{\partial \boldsymbol{G}}{\partial \boldsymbol{p}}$$

$$\frac{d\boldsymbol{x}}{d\boldsymbol{p}} = -\left(\frac{\partial \boldsymbol{G}}{\partial \boldsymbol{x}}\right)^{-1} \frac{\partial \boldsymbol{G}}{\partial \boldsymbol{p}}$$




(how does the "F-Ma" residual at time step $\boldsymbol{i}$ change wrt the j[th] input parameter $\boldsymbol{p_j}$)

Coros et al. 2021

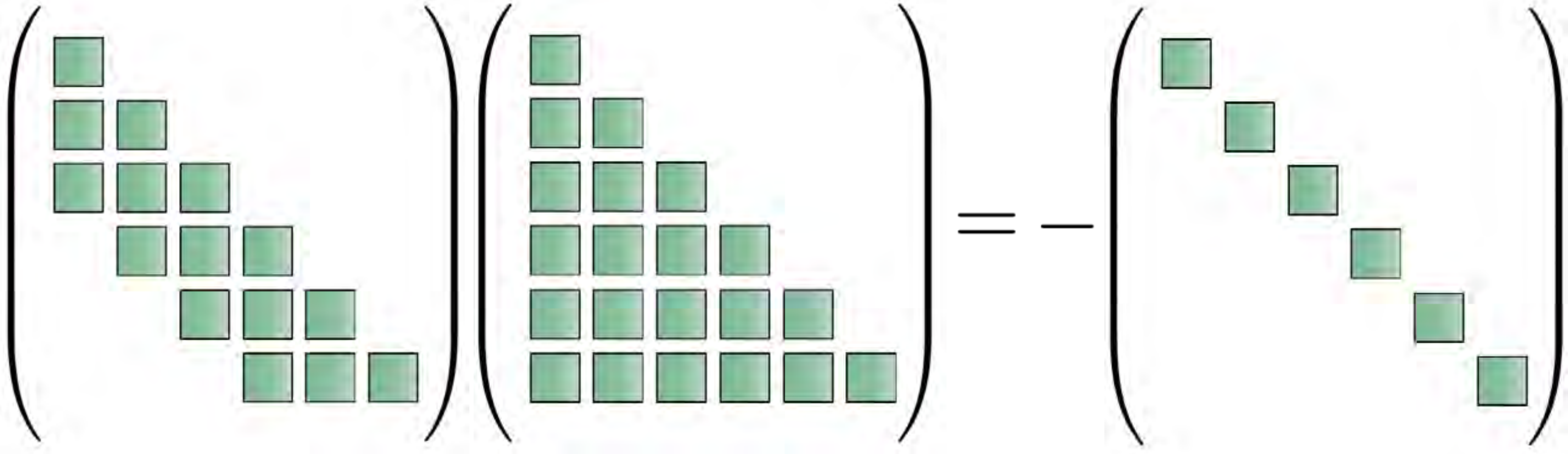
**Example:** if input parameters are actuation forces at each time step, $\mathbf{p} = [\mathbf{f}_0^{act}, \mathbf{f}_1^{act}, \ldots, \mathbf{f}_n^{act}]$

$$\boldsymbol{G}(\boldsymbol{x}(\boldsymbol{p}), \boldsymbol{p}) = \mathbf{0}, \forall \boldsymbol{p}$$

$$\frac{d\boldsymbol{G}}{d\boldsymbol{p}} = \mathbf{0} = \frac{\partial \boldsymbol{G}}{\partial \boldsymbol{x}} \frac{d\boldsymbol{x}}{d\boldsymbol{p}} + \frac{\partial \boldsymbol{G}}{\partial \boldsymbol{p}}$$

$$\frac{d\boldsymbol{x}}{d\boldsymbol{p}} = -\left(\frac{\partial \boldsymbol{G}}{\partial \boldsymbol{x}}\right)^{-1} \frac{\partial \boldsymbol{G}}{\partial \boldsymbol{p}}$$

$$\frac{\partial \boldsymbol{G}}{\partial \boldsymbol{x}} \qquad \frac{\mathrm{d}\boldsymbol{x}}{\mathrm{d}\boldsymbol{p}} \qquad \frac{\partial \boldsymbol{G}}{\partial \boldsymbol{p}}$$



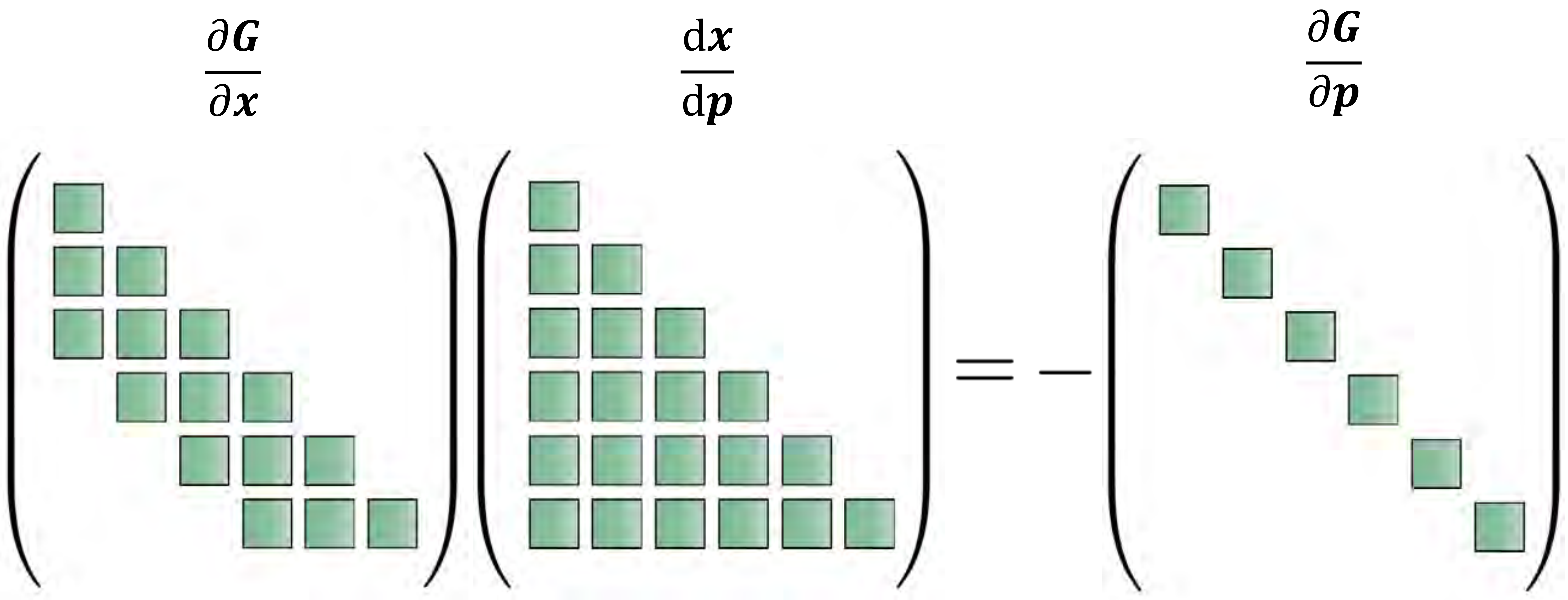
because $\mathbf{G}_i = \mathbf{M}(\mathbf{x}_i - 2\mathbf{x}_{i-1} + \mathbf{x}_{i-2})/h^2 - (\mathbf{F}(\mathbf{x}_i) + \mathbf{f}_i^{act})$

Coros et al. 2021

$$\boldsymbol{G}(\boldsymbol{x}(\boldsymbol{p}), \boldsymbol{p}) = \boldsymbol{0}, \forall \boldsymbol{p}$$

$$\frac{d\boldsymbol{G}}{d\boldsymbol{p}} = \boldsymbol{0} = \frac{\partial \boldsymbol{G}}{\partial \boldsymbol{x}} \frac{d\boldsymbol{x}}{d\boldsymbol{p}} + \frac{\partial \boldsymbol{G}}{\partial \boldsymbol{p}}$$

$$\frac{d\boldsymbol{x}}{d\boldsymbol{p}} = -\left(\frac{\partial \boldsymbol{G}}{\partial \boldsymbol{x}}\right)^{-1} \frac{\partial \boldsymbol{G}}{\partial \boldsymbol{p}}$$




Still very expensive if we have many DOFs, many time steps, and many parameters!

If we just want the gradient with respect to some scalar objective/score $s(\mathbf{x})$, there should be a way to do backpropagation / reverse mode…

Coros et al. 2021

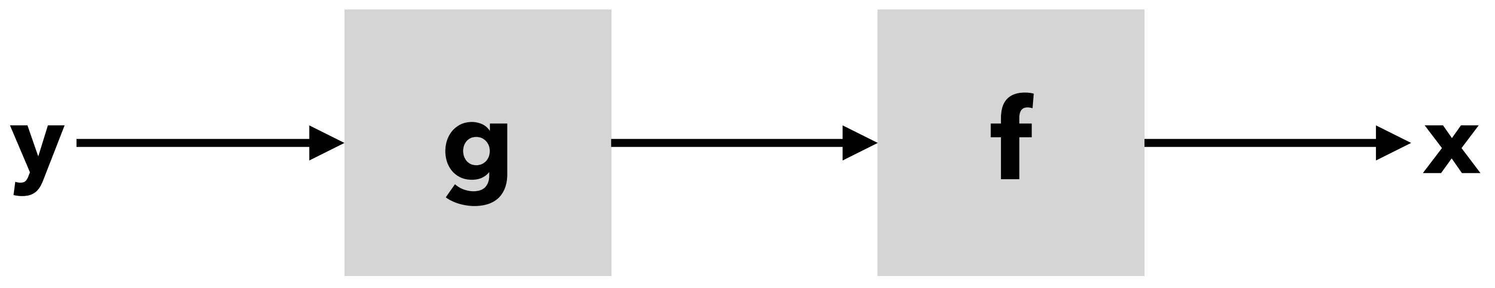
# Adjoint variables

Quick notational convenience: We'll need the gradient of the score $s(\mathbf{x})$ with respect to various intermediate variables **y**, **z**, etc.

Recall $\dfrac{\partial s}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial s}{\partial x_1} & \frac{\partial s}{\partial x_2} & \cdots & \frac{\partial s}{\partial x_n} \end{bmatrix}$

Define the adjoint $\mathbf{x}^* = \left( \dfrac{\partial s}{\partial \mathbf{x}} \right)^T = \nabla_{\mathbf{x}} s$

If $\mathbf{x} = \mathbf{f}(\mathbf{g}(\mathbf{y}))$, then




$$\frac{\partial s}{\partial \mathbf{y}} = \frac{\partial s}{\partial \mathbf{x}} \frac{\partial \mathbf{f}}{\partial \mathbf{g}} \frac{\partial \mathbf{g}}{\partial \mathbf{y}}$$

$$\mathbf{y}^* = \mathbf{J_g}^\top \mathbf{J_f}^\top \mathbf{x}^*$$

Coros et al. 2021

# (Discrete) adjoint method

- Replace ODE with time-stepping equations:

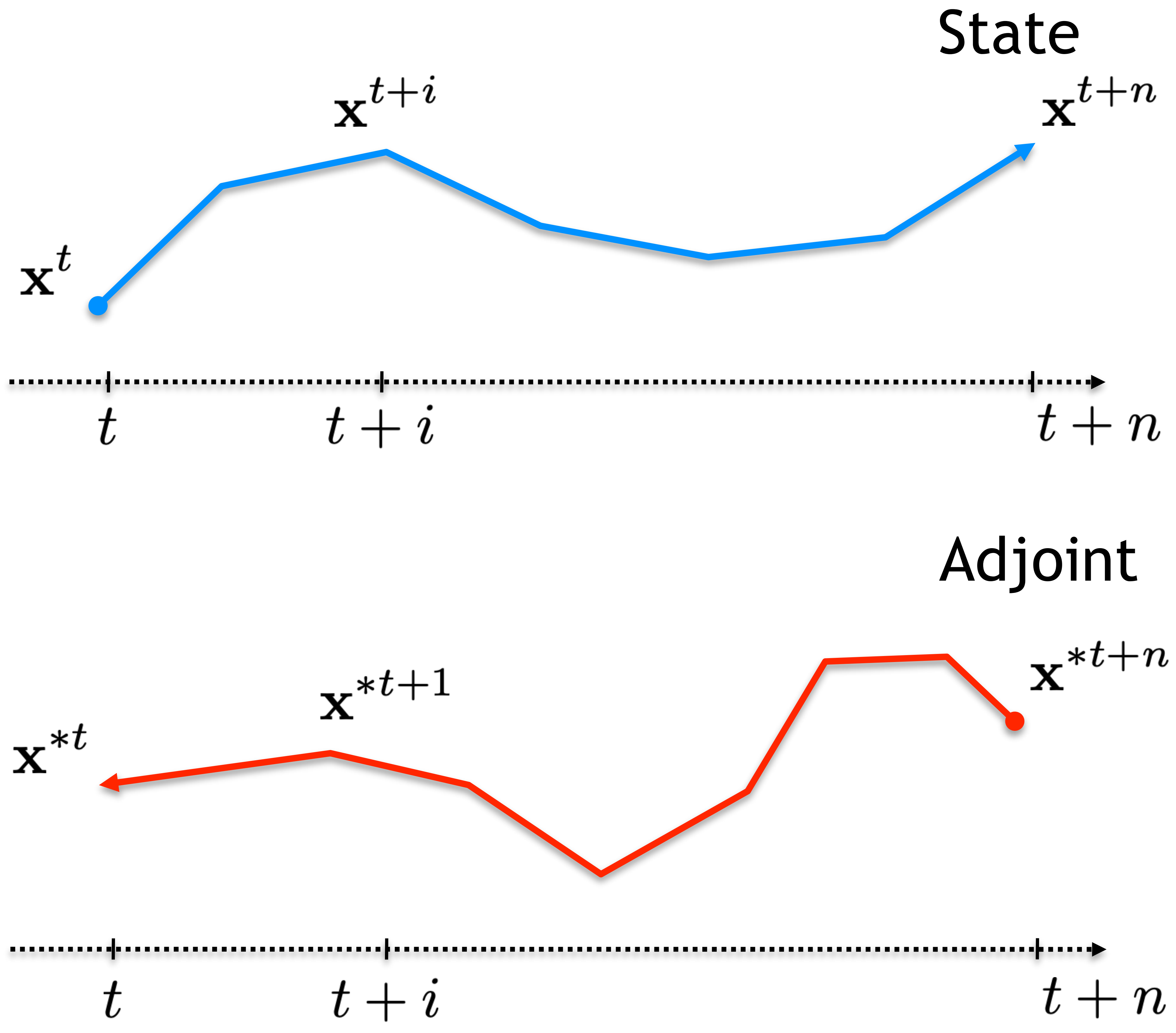
$$\mathbf{x}^{t+1} = f(\mathbf{x}^t)$$

- Discrete trajectory + loss:

$$s(\mathbf{x}^{t+n}) = s(f(f(f(\mathbf{x}^t))$$

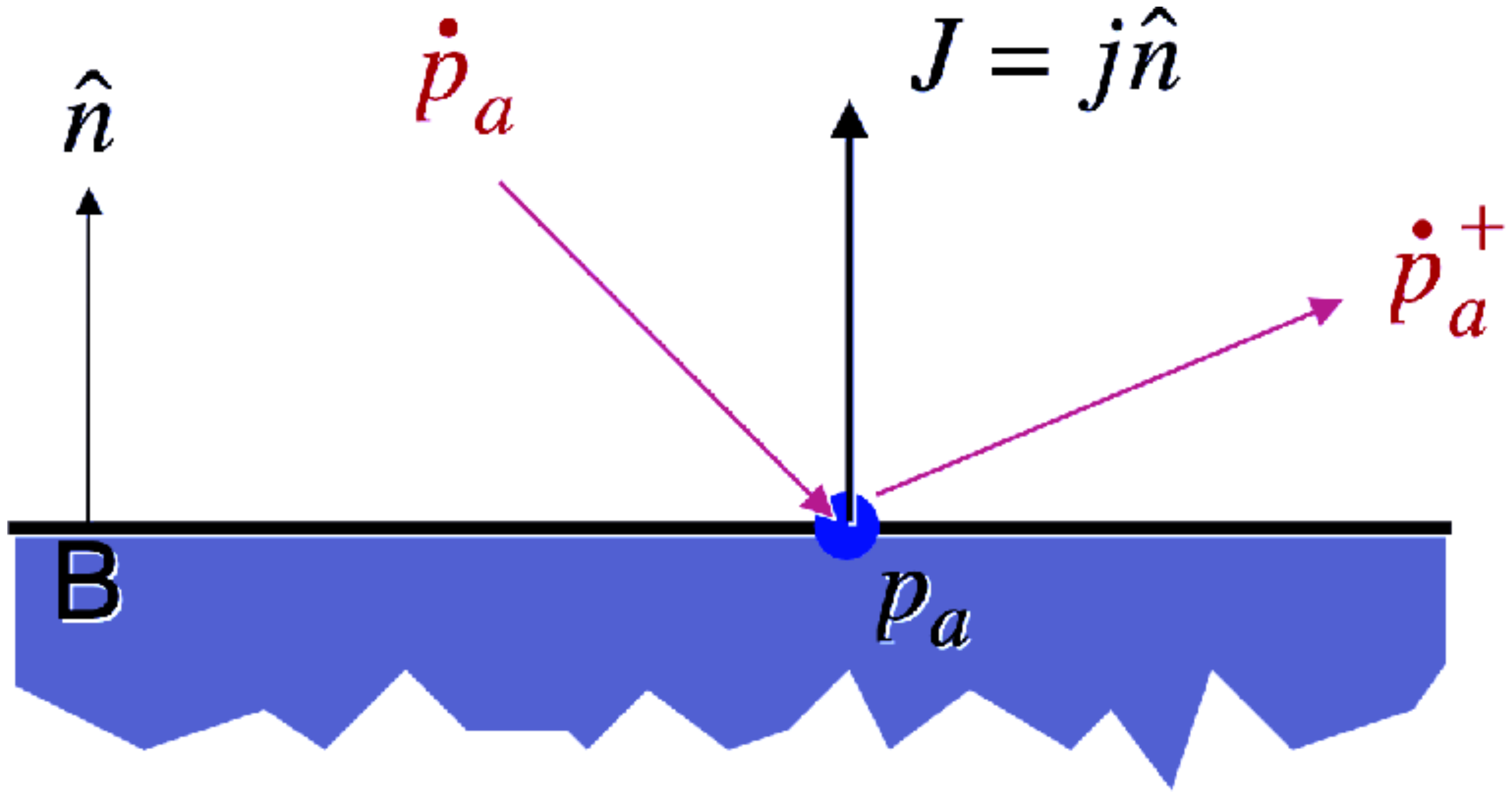
- Apply chain rule:

$$\mathbf{x}^{*^t} = \frac{\partial s}{\partial \mathbf{x}}\bigg|_{t+0}^T = \frac{\partial f}{\partial \mathbf{x}}\bigg|_{t+0}^T \cdot \frac{\partial f}{\partial \mathbf{x}}\bigg|_{t+1}^T \cdot \frac{\partial f}{\partial \mathbf{x}}\bigg|_{t+2}^T \cdot \frac{\partial s}{\partial \mathbf{x}}\bigg|_{t+3}^T$$




Coros et al. 2021

# Collisions

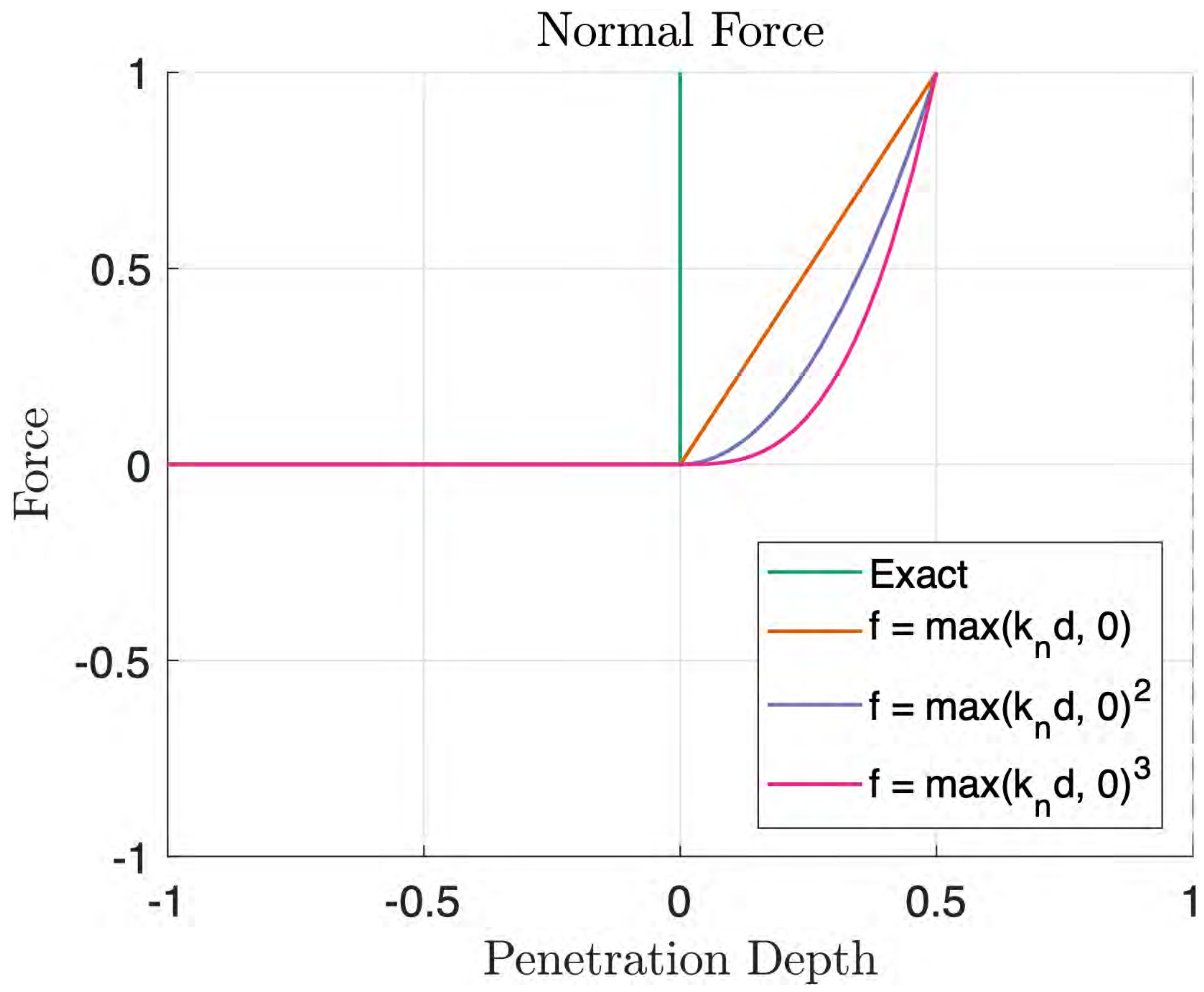


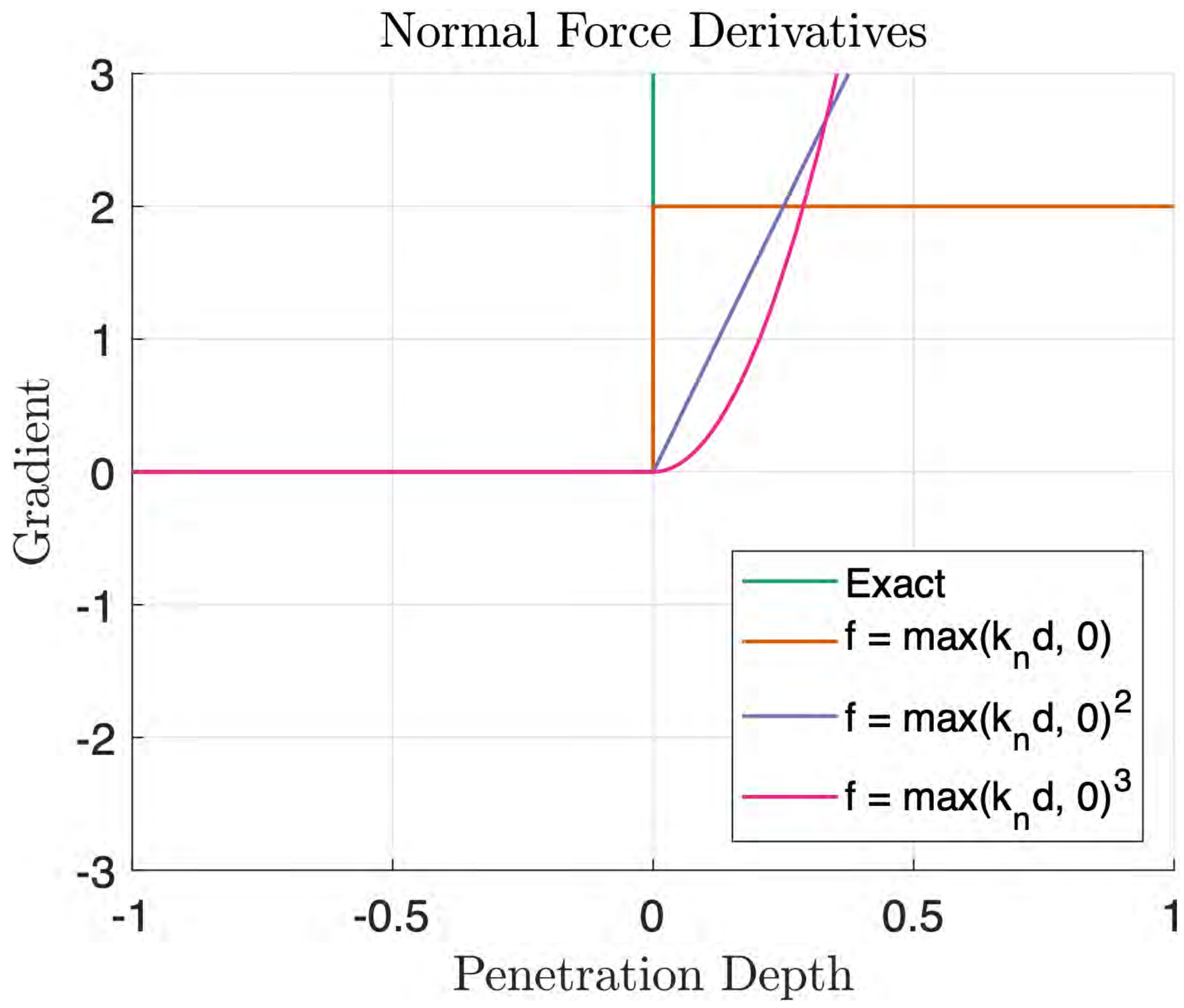

Witkin & Baraff 2001

**Problem:** Collisions are nonsmooth events!

Both normal and frictional force change nonsmoothly with position/velocity

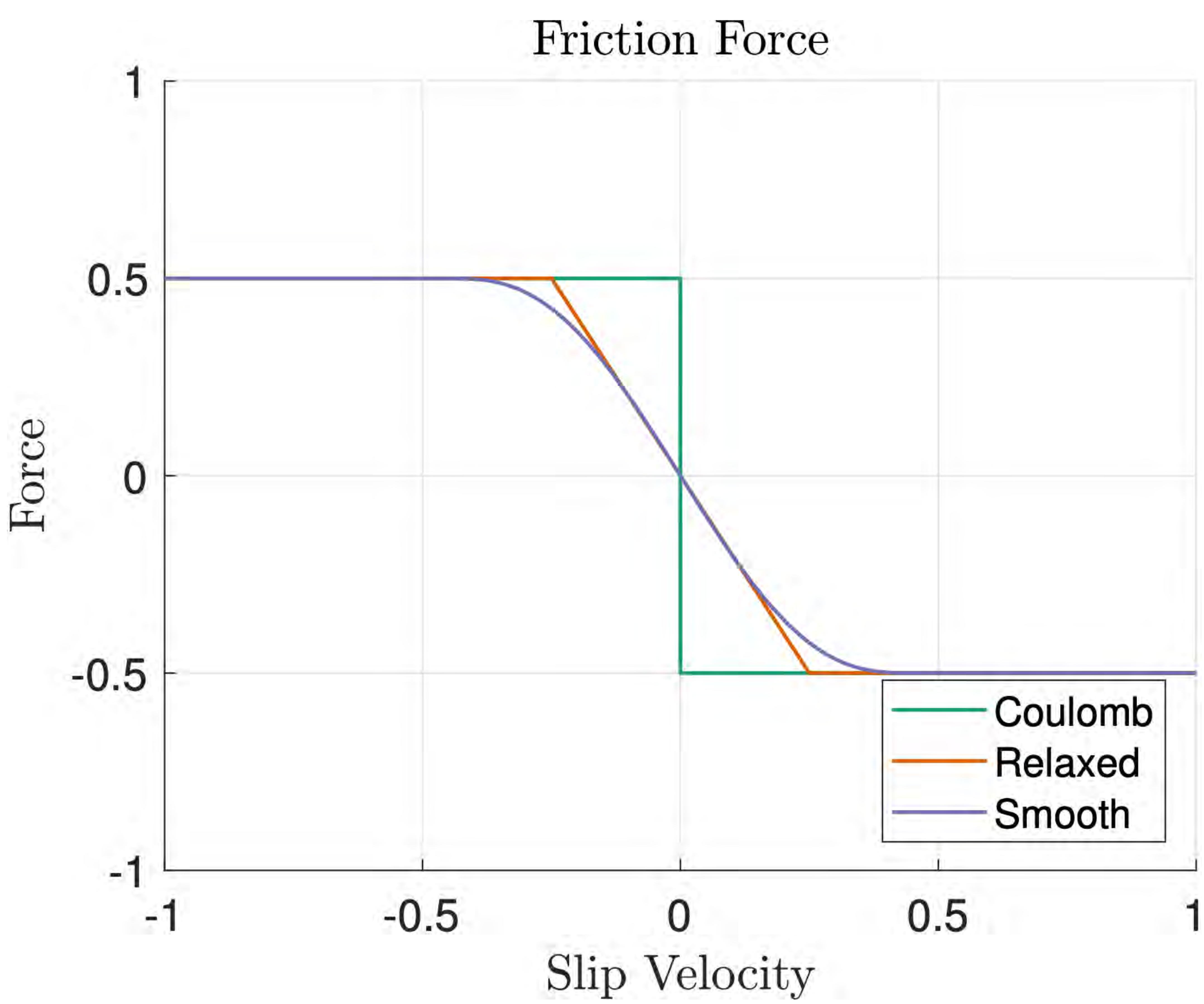
# Smoothed contact

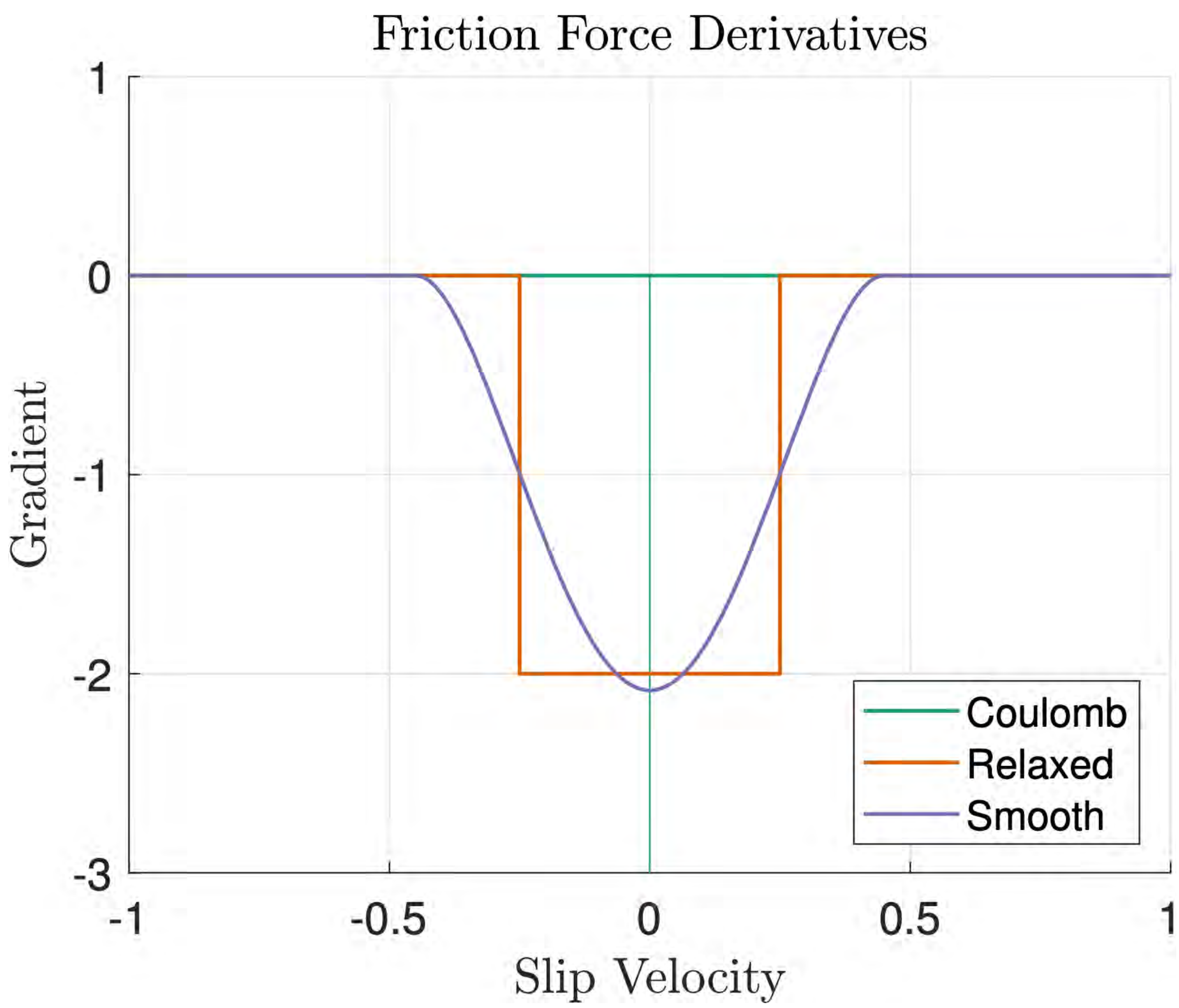



Coros et al. 2021

# Smoothed contact

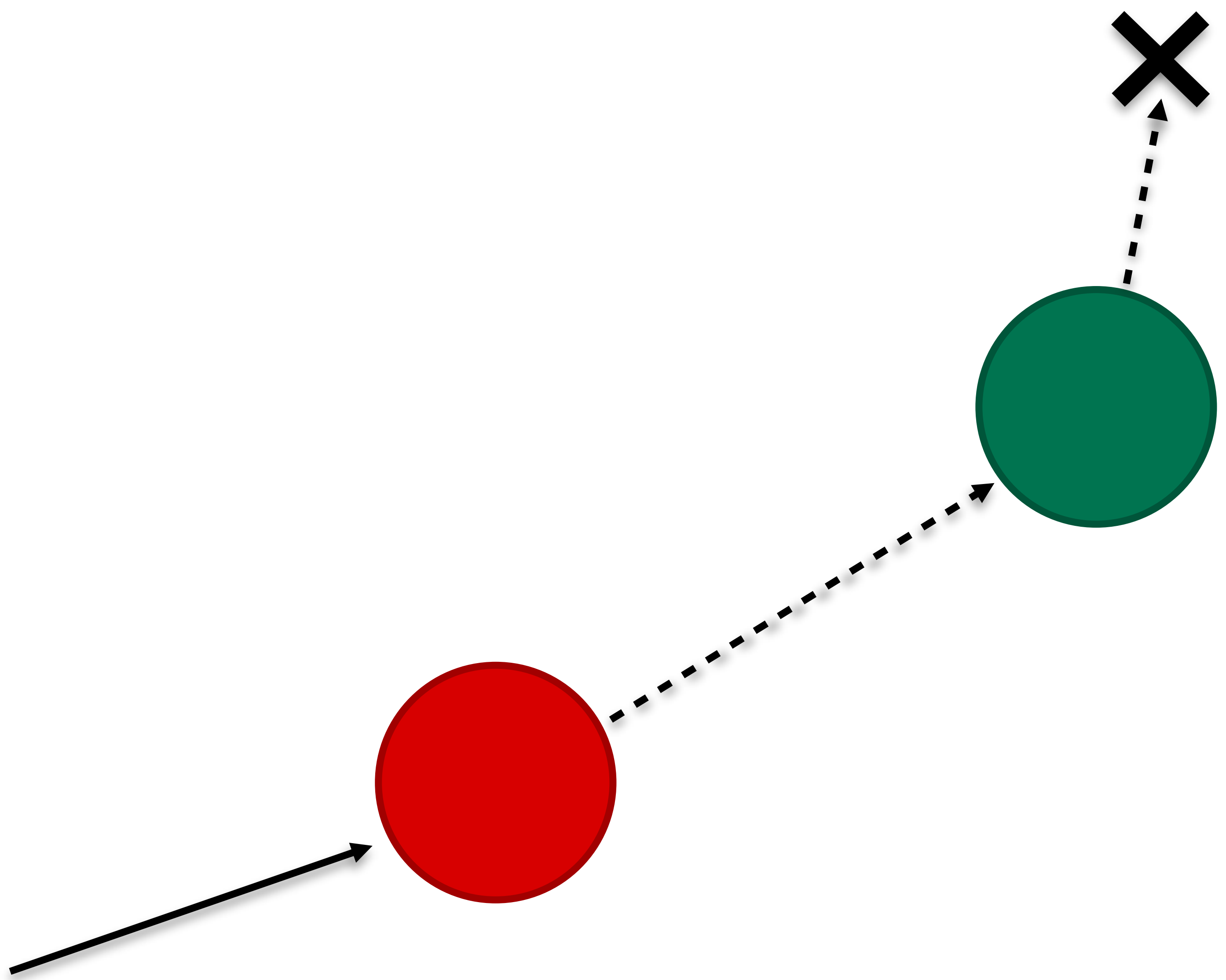


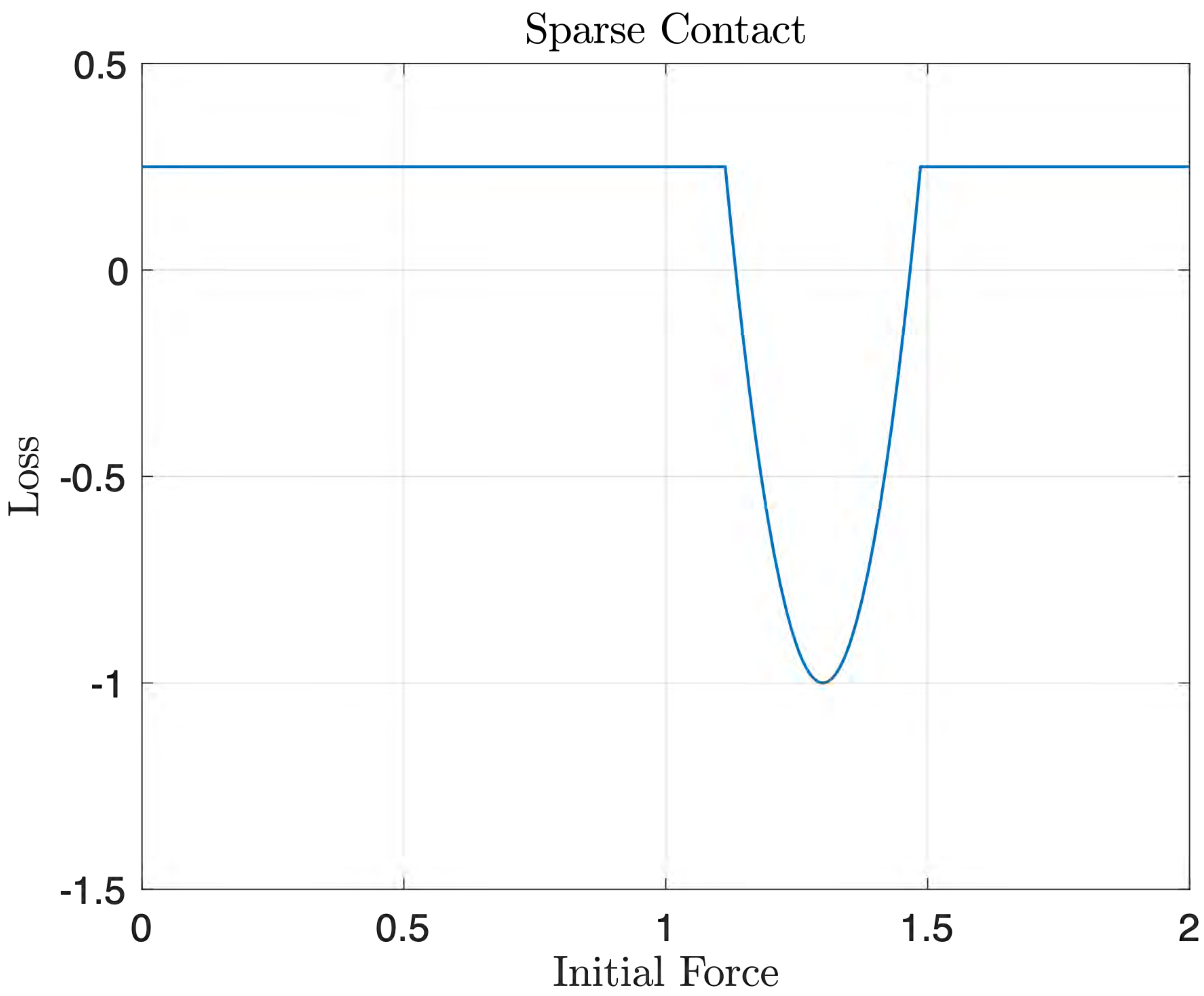




Coros et al. 2021

# Contact sparseness

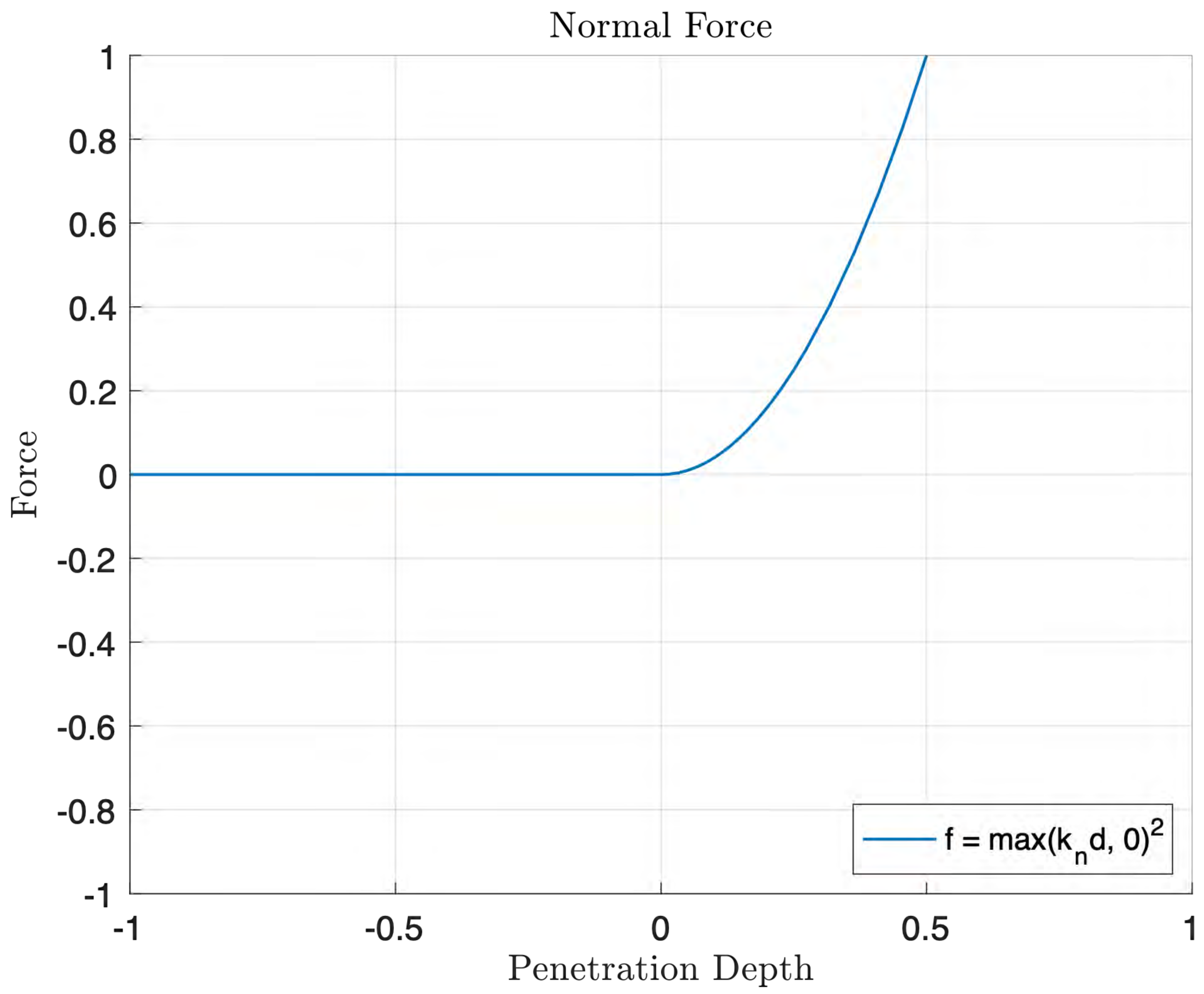
- No gradient information until contact
- Optimization stuck at local minima

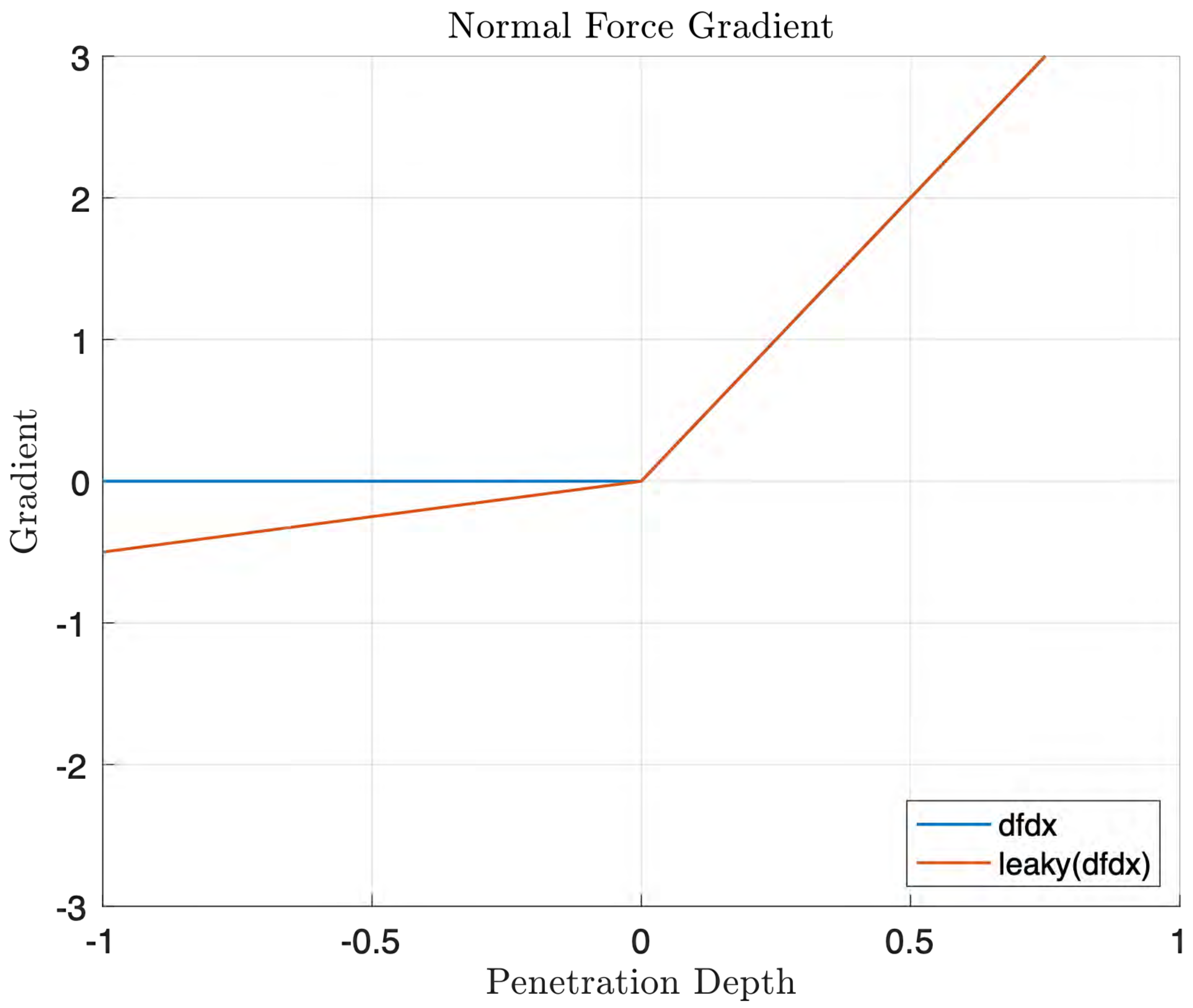





Coros et al. 2021

# Solution: leaky gradients

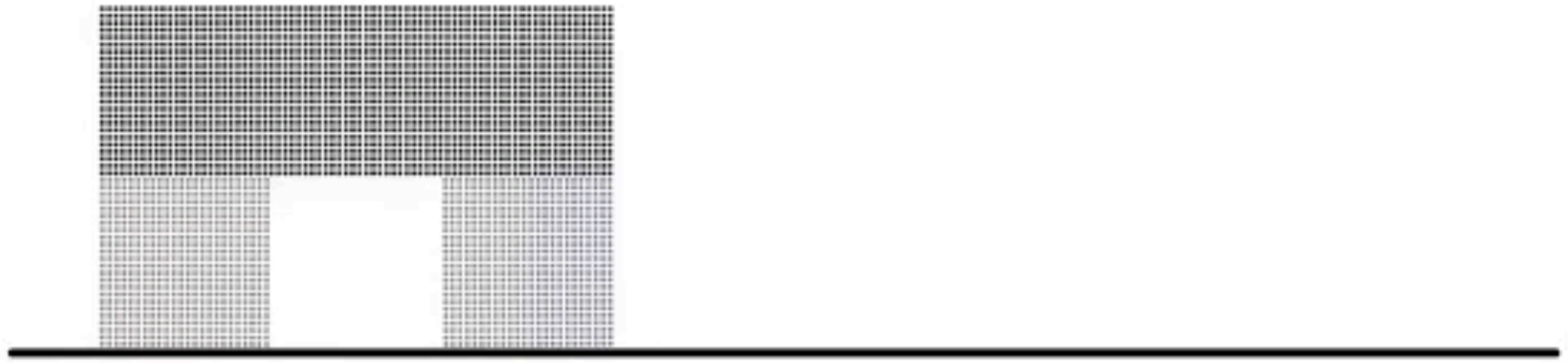


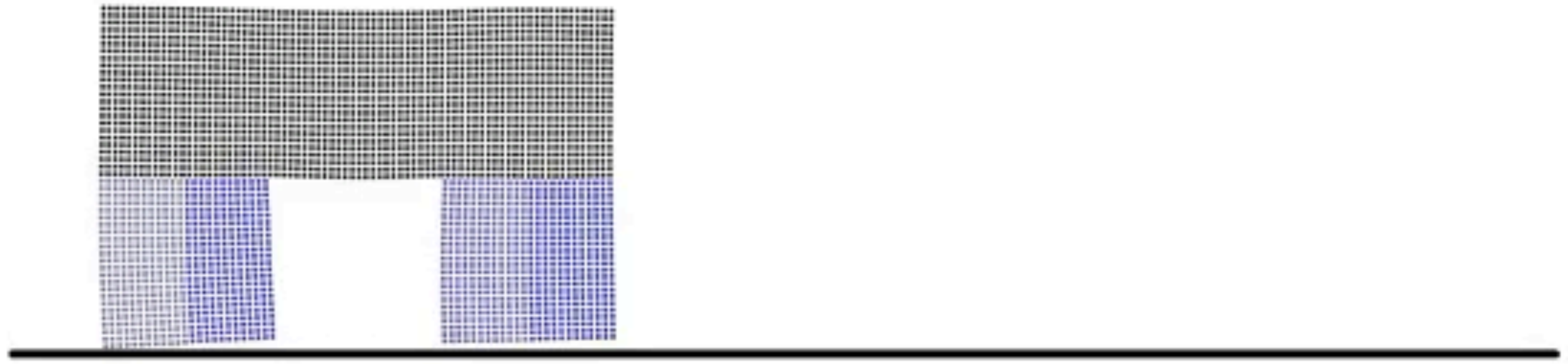

Coros et al. 2021

# Differentiable Elastic Object Simulation

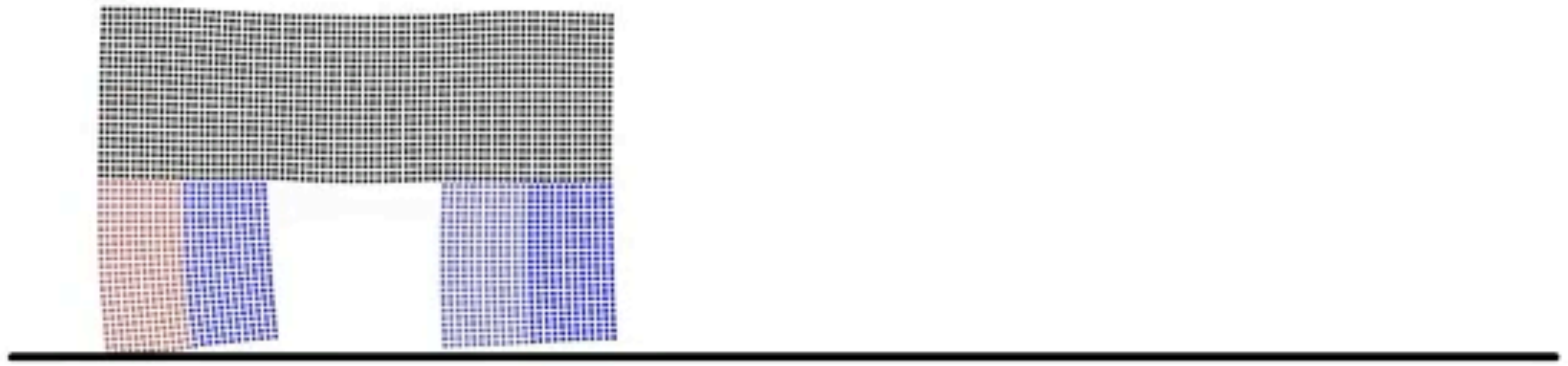
Iteration 0



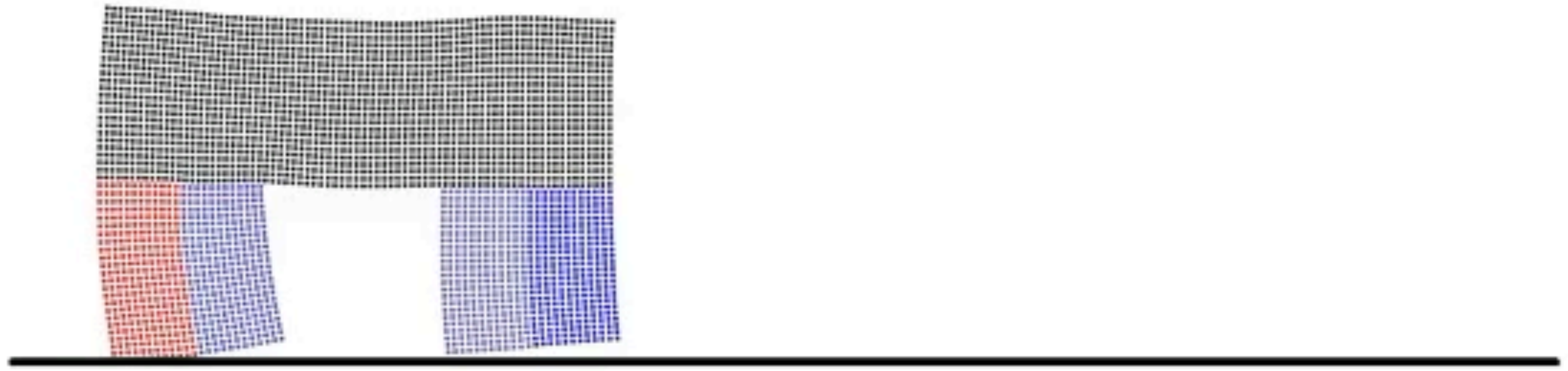
Iteration 20



Iteration 40



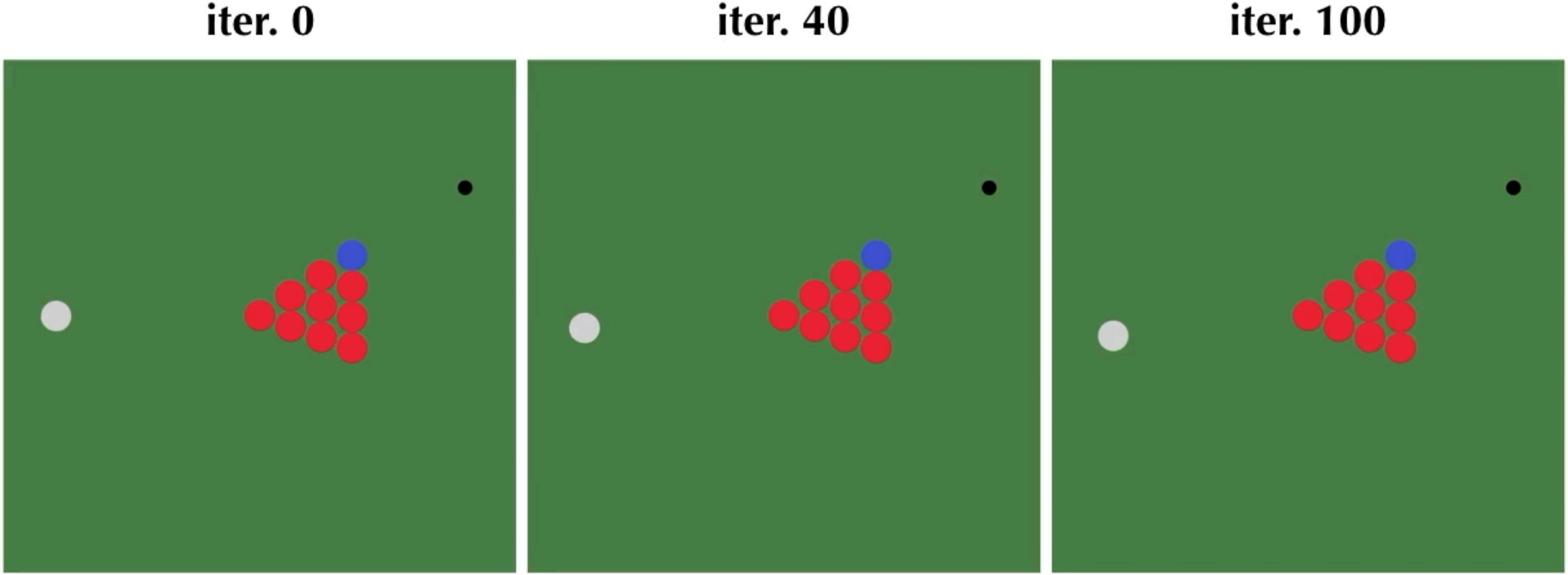
Iteration 80



Continuum modeled with both particles and grids. Open-loop controller.
4.2x shorter code than ChainQueen [Hu et al. ICRA 2019]; 188x faster than TensorFlow.
1024 time steps, 80 gradient descent iter. Run time=2min. Red=extension blue=contraction.
Reproduce: `python3 diffmpm.py`

Hu et al., "DiffTaichi", ICLR 2020

# Differentiable Billiard Simulation




Optimize the **initial position** and **velocity** of the white ball so that the blue ball goes to the black destination

Reproduce: `python3 billiards.py`

Hu et al., "DiffTaichi", ICLR 2020

# Motion capture data


Geilinger et al., “ADD: Analytically Differentiable Dynamics…”, SIGGRAPH Asia 2020


Geilinger et al., "ADD: Analytically Differentiable Dynamics…", SIGGRAPH Asia 2020


Geilinger et al., “ADD: Analytically Differentiable Dynamics…”, SIGGRAPH Asia 2020

# Acknowledgements

Many of these slides are based on the following source:

- Coros et al., *Differentiable Simulation*, SIGGRAPH 2021