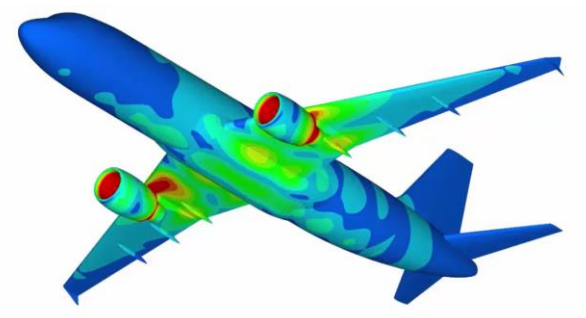
Towards Real-time Simulation of Hyperelastic Materials

A Dissertation Presentation

Tiantian Liu

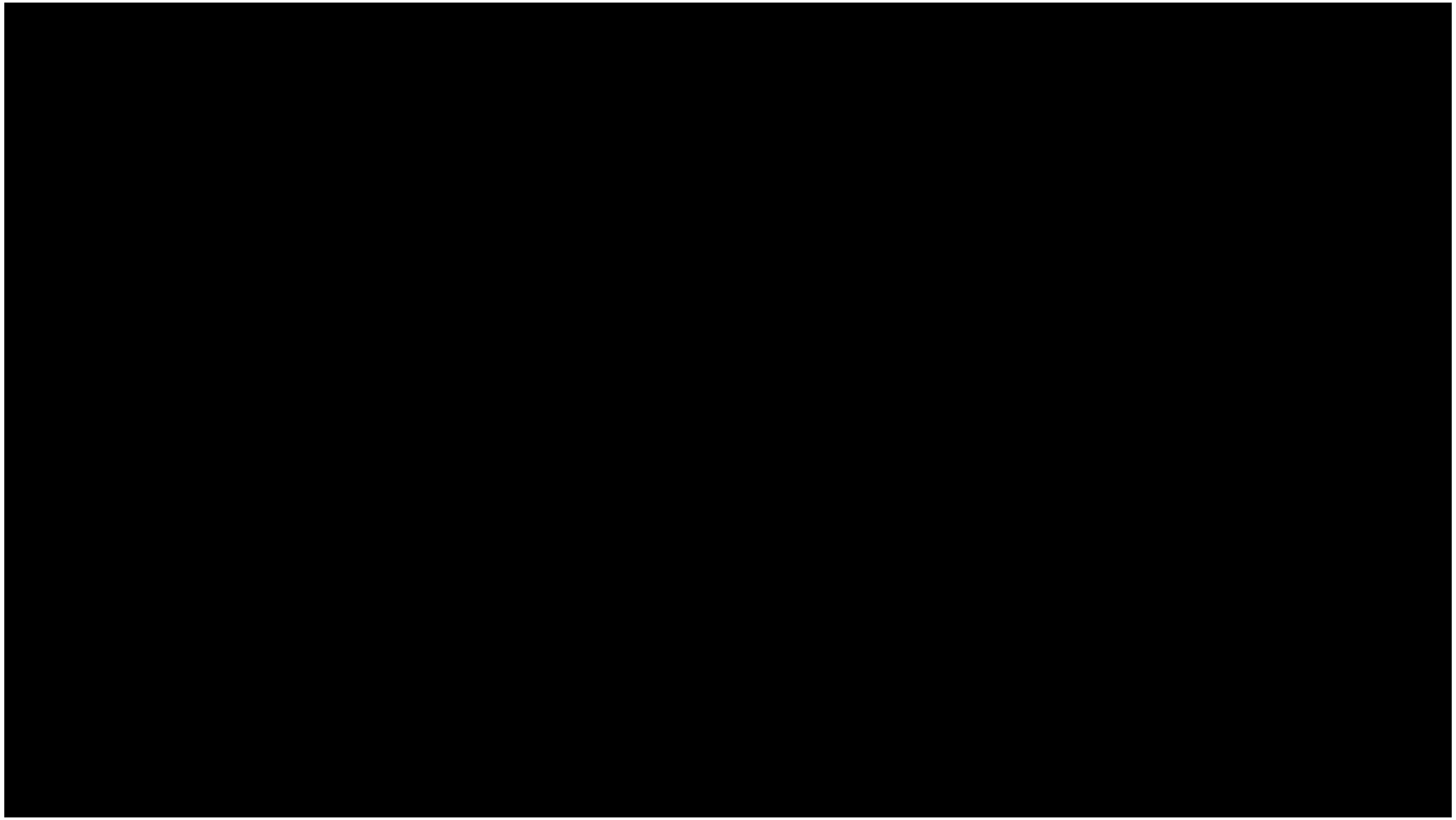
April 24th 2018

Penn
University of Pennsylvania



Deformable Body Simulation

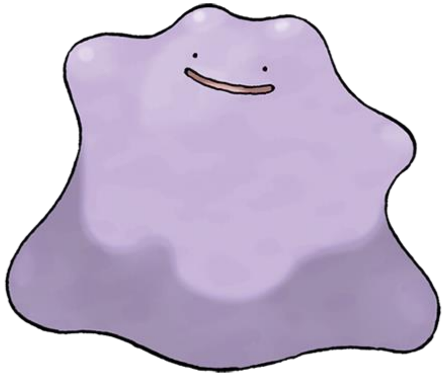
Limited Human Interactivity in Mixed Reality Environments



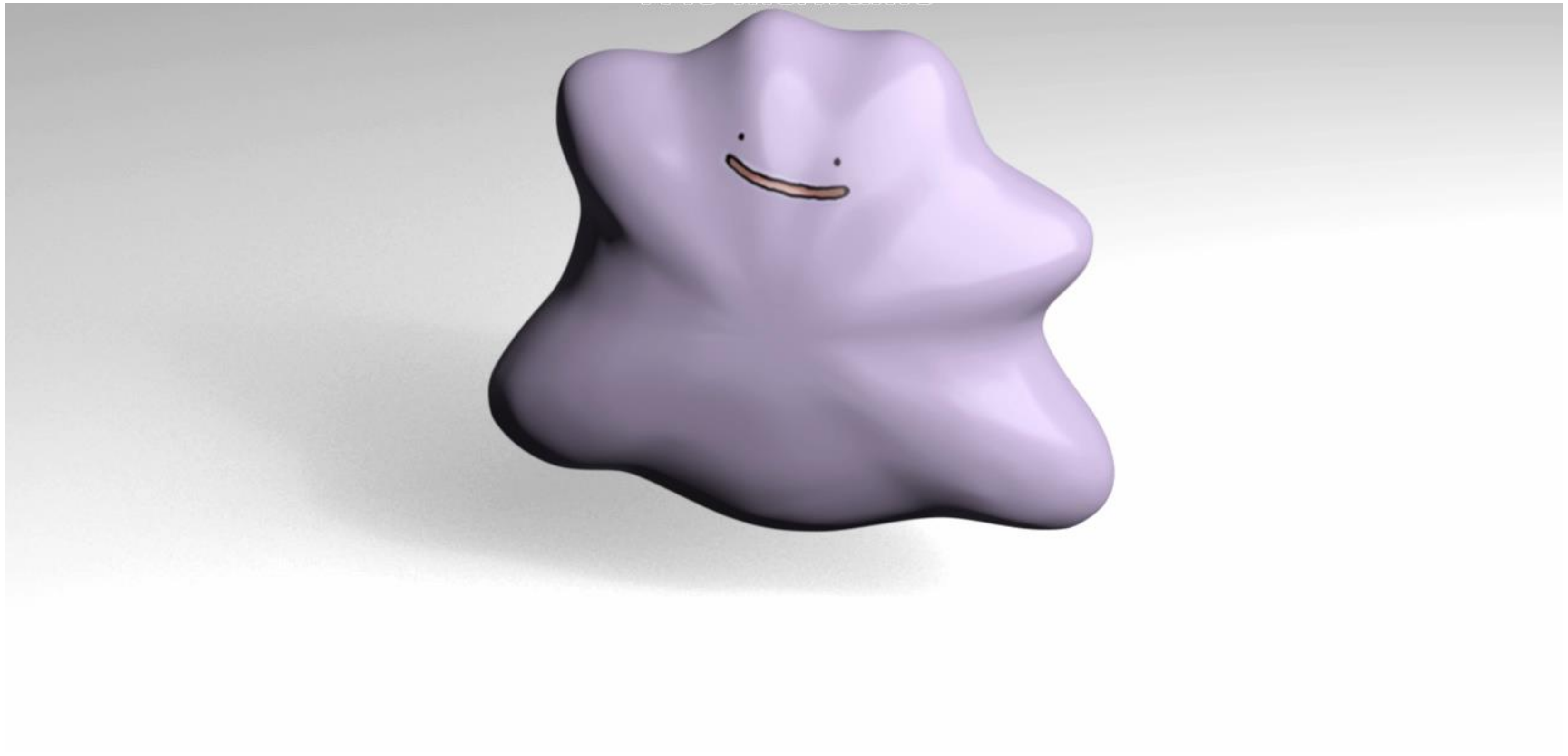
Limited Materials in Real-time Simulators

[Bai et al. 2016]

Robotics

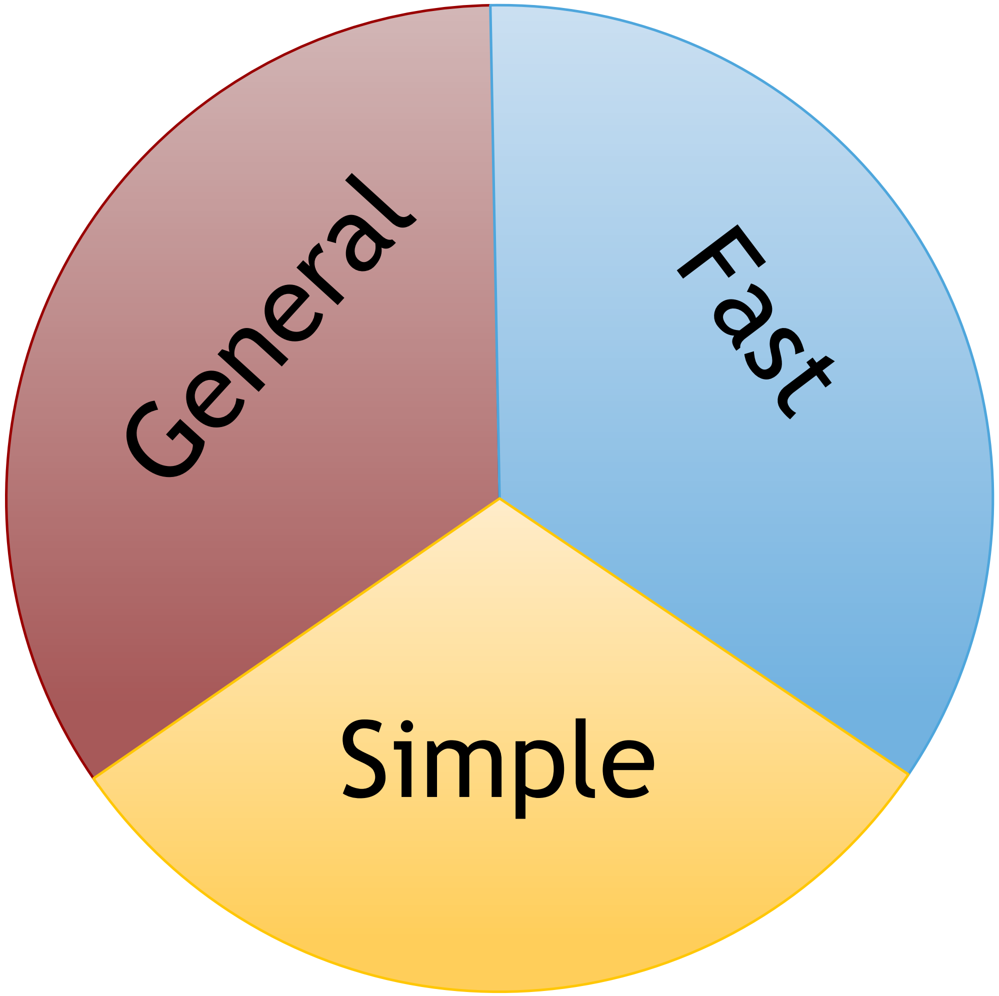


Rigid Body v.s. Deformable Body

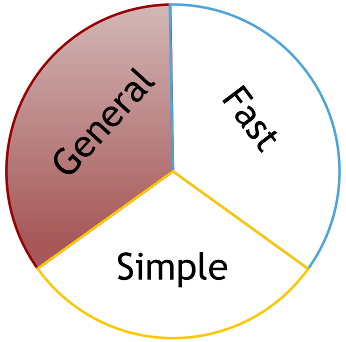


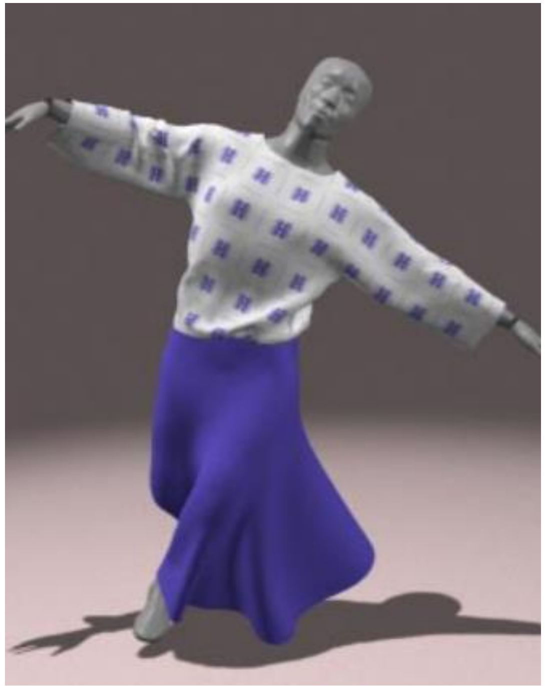
Goal: Fast simulation of general deformable objects

Goal: Fast simulation of general deformable objects



Related Work: Classic work



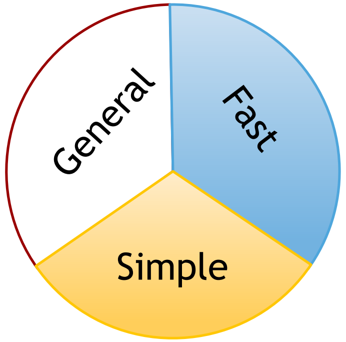


[Baraff and Witkin 1998]

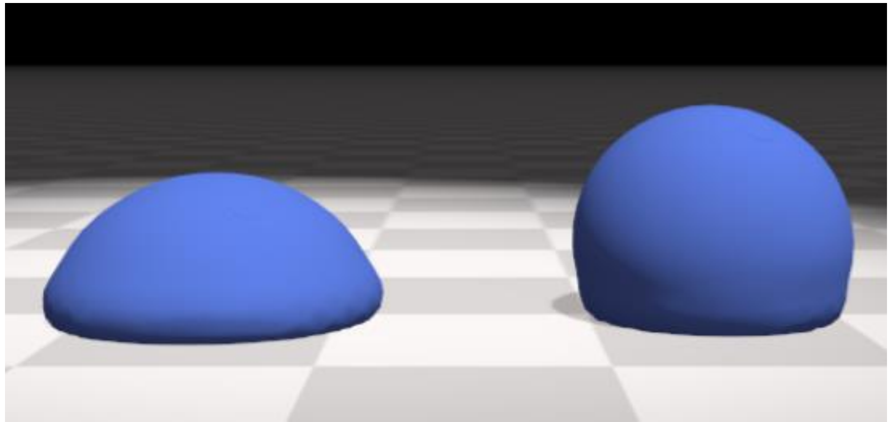
[Goldenthal et al. 2007]

[Tournier et al. 2015]

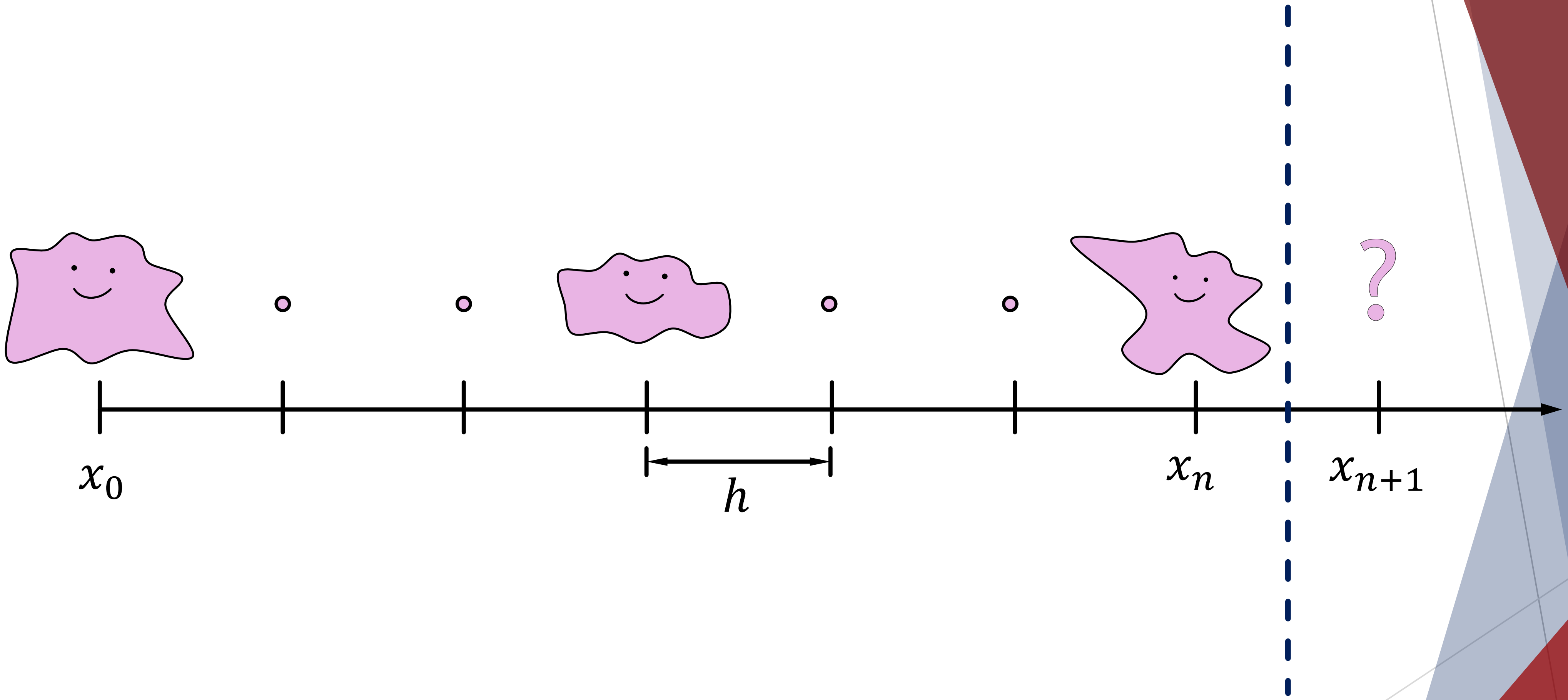
Related Work: Position Based Dynamics




[Müller et al. 2007]


[Macklin et al. 2016]

Simulation: Prediction of Future



Temporal Discretization

- Newton's 2nd Law of Motion
  - $x_{n+1} = x_n + \int_{t_n}^{t_n+h} v(t)dt$
  - $v_{n+1} = v_n + \int_{t_n}^{t_n+h} M^{-1}\left(f_{int}\left(x(t)\right) + f_{ext}\right)dt$

Temporal Discretization

- Implicit Euler Integration
  - $x_{n+1} = x_n + hv_{n+1}$
  - $v_{n+1} = v_n + hM^{-1}(f_{int}(x_{n+1}) + f_{ext})$

  - $\underline{x_{n+1}} = \underbrace{x_n + hv_n + h^2M^{-1}f_{ext}}_{y} + h^2M^{-1}f_{int}(\underline{x_{n+1}})$

Temporal Discretization

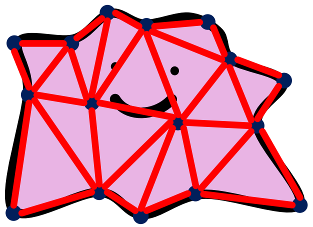
- Variational Implicit Euler
  - $$x_{n+1} = \underbrace{argmin_x \underbrace{\frac{1}{2h^2}\|x - y\|_M}_{\text{inertia}} + \underbrace{E(x)}_{\text{elasticity}}}_{g(x)}$$

Typical workflow of a deformable body simulation

Spatial Discretization

Temporal Discretization

Numerical Solution

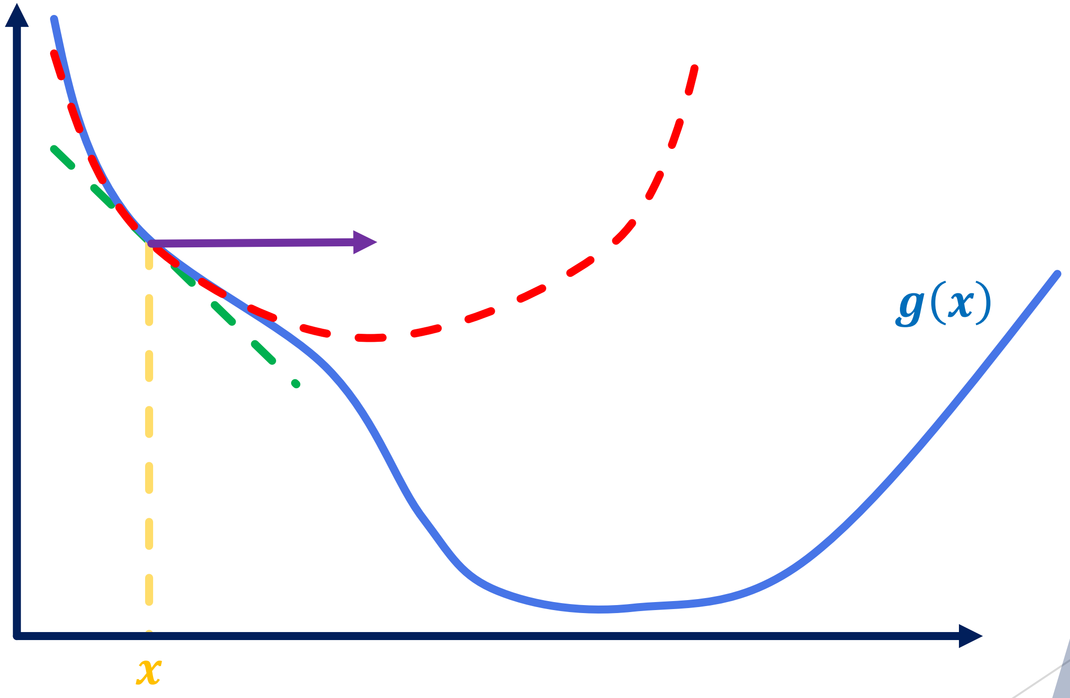


$$\min_{x} \frac{1}{2h^2} \|x - y\|_M + E(x)$$

?

Numerical Solution

$$\Delta x = -[\nabla^2 g(x)]^{-1} \nabla g(x)$$



Numerical Solution: Newton's Method

$$\min_{x} \underbrace{\frac{1}{2h^2}\|x-y\|_M}_{\text{☺}} + \underbrace{E(x)}_{\text{☹}}$$

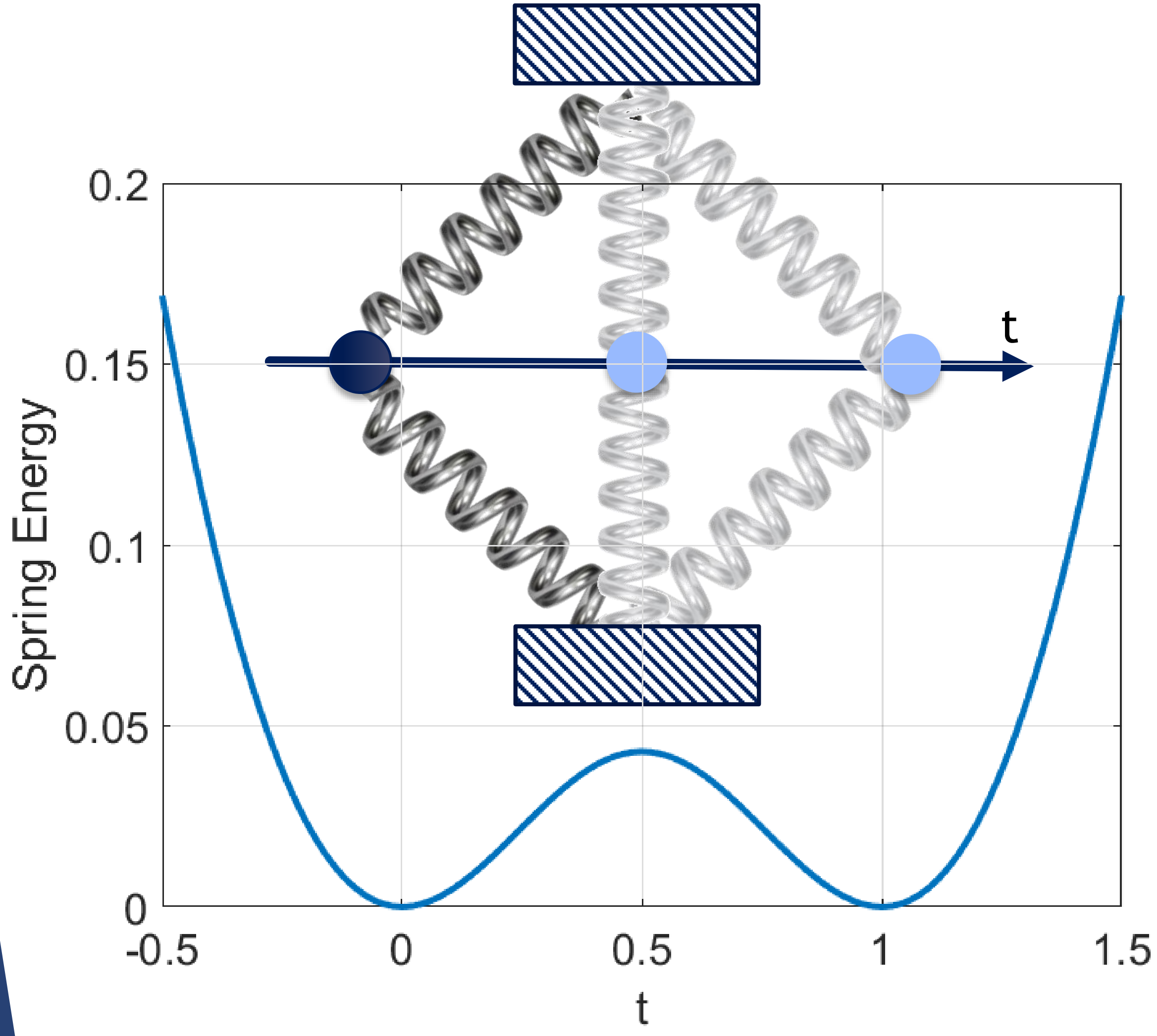
☹ ▶ Slow

- $\nabla^2 E$ depends on x

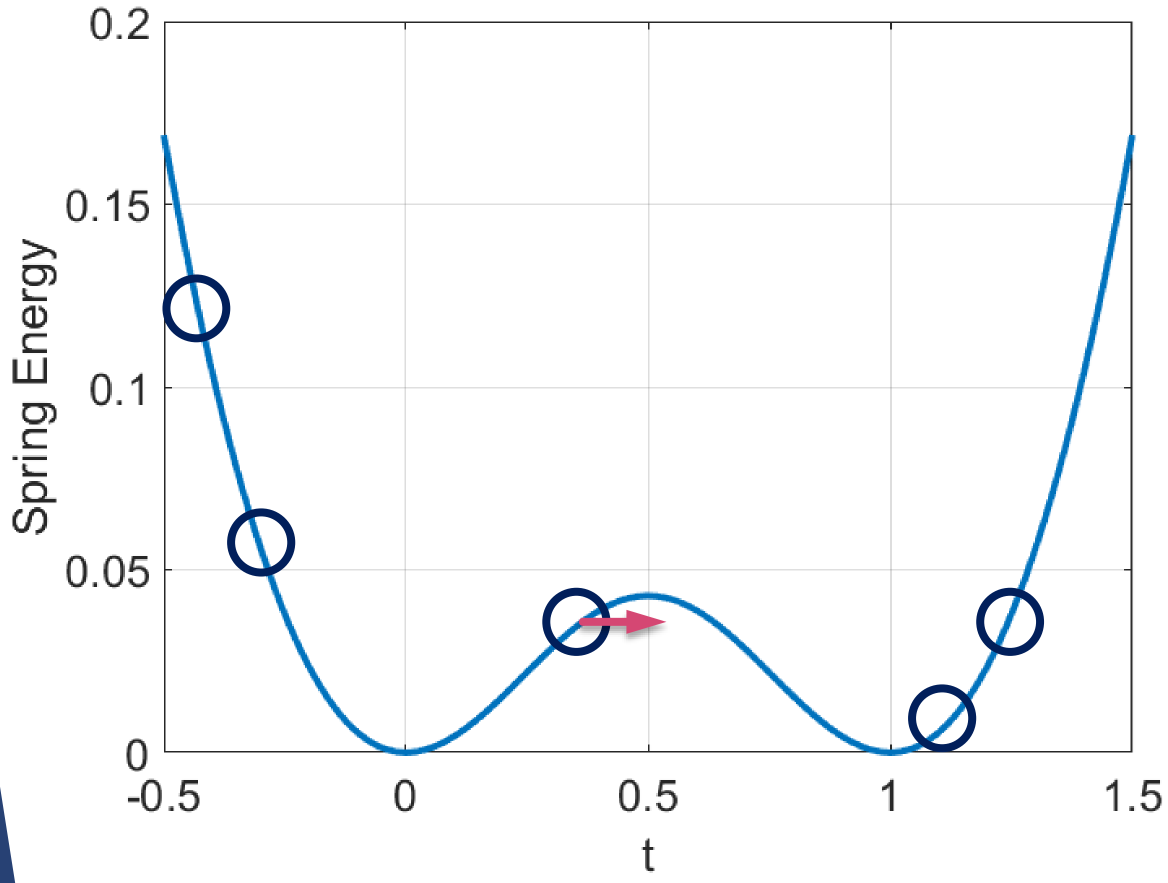
☹ ▶ Non-convex

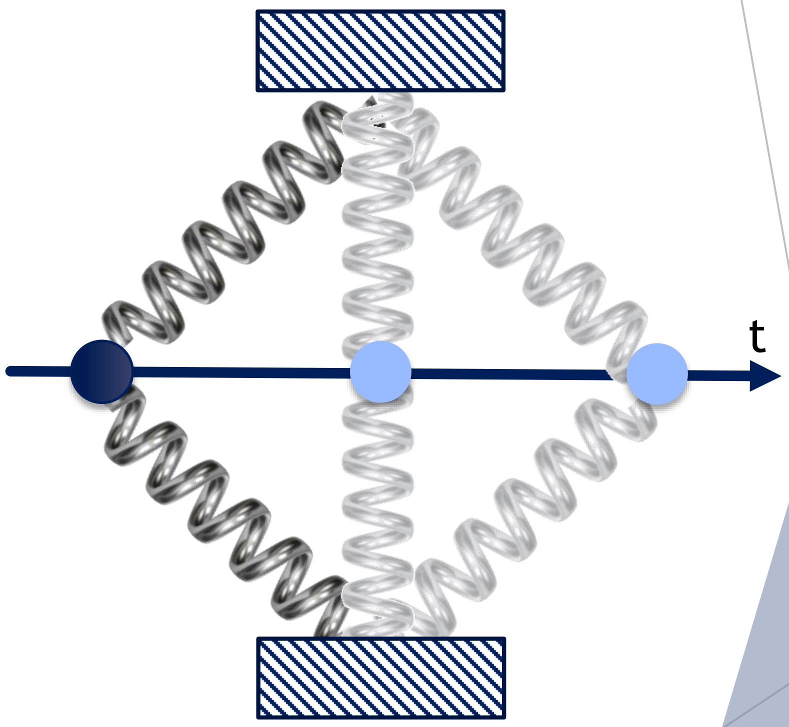
- The Hessian $M/h^2 + \nabla^2 E$ can be indefinite

Non-convex Potential



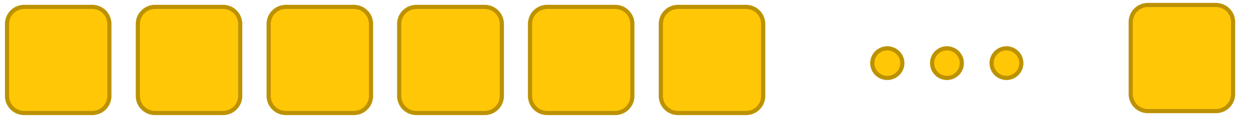
Numerical Solution: Newton's Method





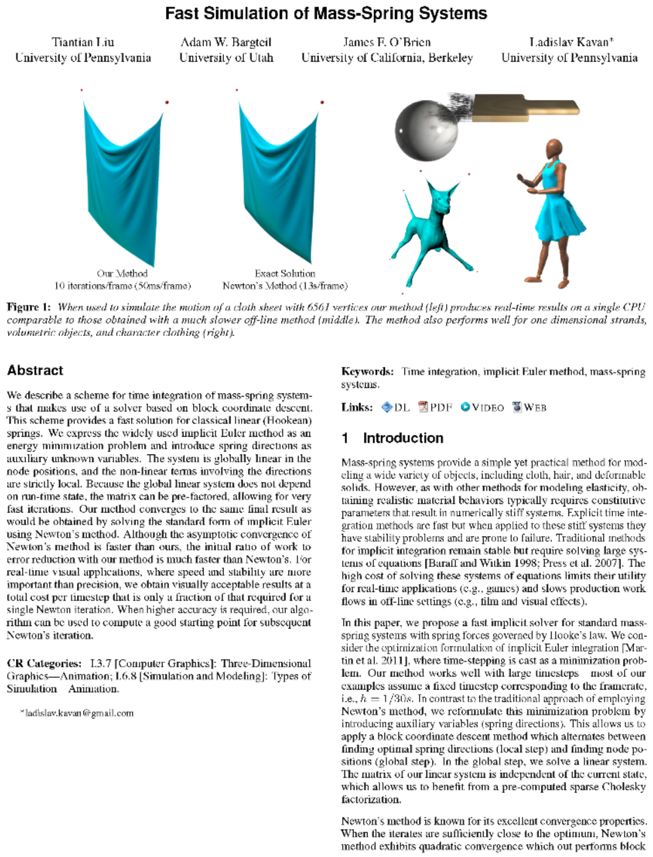
Ideal Numerical Problems

Large Convex Quadratic Problem
(Ideally with Constant System Matrix)



Many Small Non-convex Problems
(Ideally Independent)

Mass-spring Systems



Fast Simulation of Mass-Spring Systems

Tiantian Liu, University of Pennsylvania
Adam W. Bargteil, University of Utah
James F. O'Brien, University of California, Berkeley
Ladislav Kavan*, University of Pennsylvania

Our Method
10 iterations/frame (50ms/frame)

Exact Solution
Newton's Method (13s/frame)

Figure 1: *When used to simulate the motion of a cloth sheet with 6561 vertices our method (left) produces real-time results on a single CPU comparable to those obtained with a much slower off-line method (middle). The method also performs well for one dimensional strands, volumetric objects, and character clothing (right).*

Abstract

We describe a scheme for time integration of mass-spring systems that makes use of a solver based on block coordinate descent. This scheme provides a fast solution for classical linear (Hookean) springs. We express the widely used implicit Euler method as an energy minimization problem and introduce spring directions as auxiliary unknown variables. The system is globally linear in the node positions, and the non-linear terms involving the directions are strictly local. Because the global linear system does not depend on run-time state, the matrix can be pre-factored, allowing for very fast iterations. Our method converges to the same final result as would be obtained by solving the standard form of implicit Euler using Newton's method. Although the asymptotic convergence of Newton's method is faster than ours, the initial ratio of work to error reduction with our method is much faster than Newton's. For real-time visual applications, where speed and stability are more important than precision, we obtain visually acceptable results at a total cost per timestep that is only a fraction of that required for a single Newton iteration. When higher accuracy is required, our algorithm can be used to compute a good starting point for subsequent Newton's iteration.

CR Categories: I.3.7 [Computer Graphics]: Three-Dimensional Graphics—Animation; I.6.8 [Simulation and Modeling]: Types of Simulation Animation.

*ladislav.kavan@gmail.com

Keywords: Time integration, implicit Euler method, mass-spring systems.

Links: DL PDF VIDEO WEB

1 Introduction

Mass-spring systems provide a simple yet practical method for modeling a wide variety of objects, including cloth, hair, and deformable solids. However, as with other methods for modeling elasticity, obtaining realistic material behaviors typically requires constitutive parameters that result in numerically stiff systems. Explicit time integration methods are fast but when applied to these stiff systems they have stability problems and are prone to failure. Traditional methods for implicit integration remain stable but require solving large systems of equations [Baraff and Witkin 1998; Press et al. 2007]. The high cost of solving these systems of equations limits their utility for real-time applications (e.g., games) and slows production work flows in off-line settings (e.g., film and visual effects).

In this paper, we propose a fast implicit solver for standard mass-spring systems with spring forces governed by Hooke's law. We consider the optimization formulation of implicit Euler integration [Martin et al. 2011], where time-stepping is cast as a minimization problem. Our method works well with large timesteps most of our examples assume a fixed timestep corresponding to the framerate, i.e., $h = 1/30s$. In contrast to the traditional approach of employing Newton's method, we reformulate this minimization problem by introducing auxiliary variables (spring directions). This allows us to apply a block coordinate descent method which alternates between finding optimal spring directions (local step) and finding node positions (global step). In the global step, we solve a linear system. The matrix of our linear system is independent of the current state, which allows us to benefit from a pre-computed sparse Cholesky factorization.

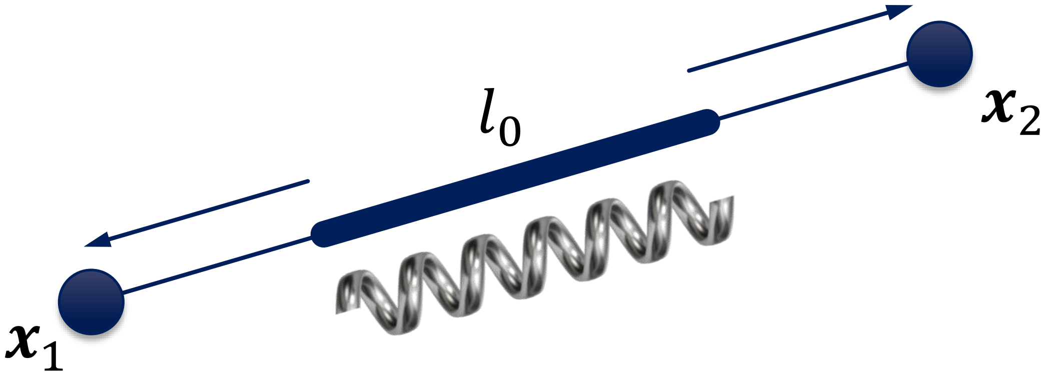
Newton's method is known for its excellent convergence properties. When the iterates are sufficiently close to the optimum, Newton's method exhibits quadratic convergence which out performs block

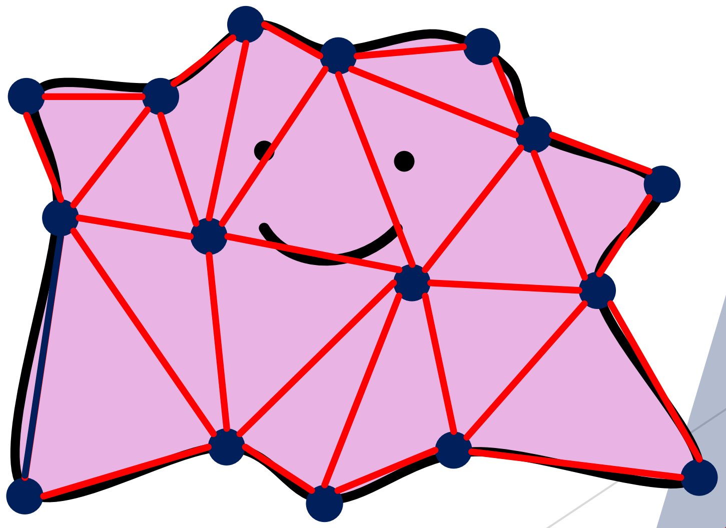
Fast Simulation of Mass-Spring Systems
Tiantian Liu, Adam W. Bargteil, James F. O'Brien, Ladislav Kavan
ACM Transactions on Graphics 32(6) [Proceedings of SIGGRAPH Asia], 2013

Mass-spring System: Basis

Hooke's Law:

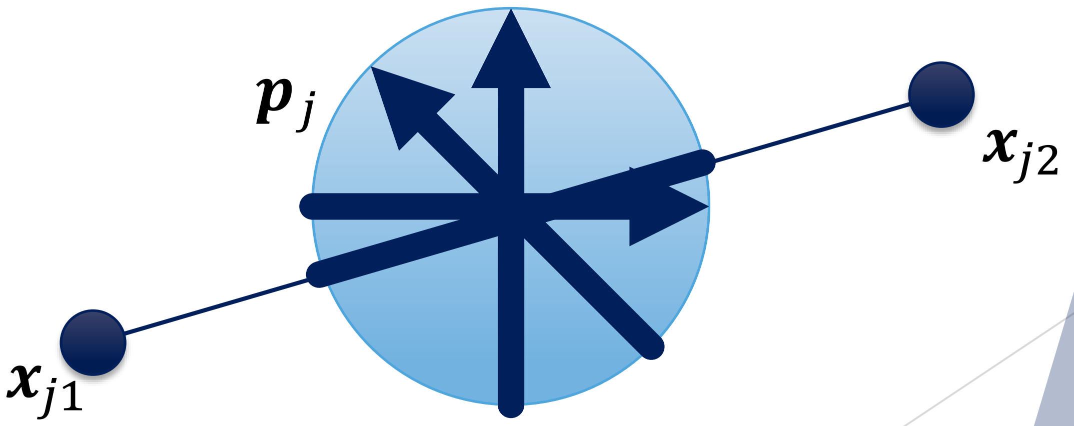
$$E(\boldsymbol{x_1}, \boldsymbol{x_2}) = \frac{1}{2}k(\|\boldsymbol{x_1} - \boldsymbol{x_2}\| - l_0)^2$$





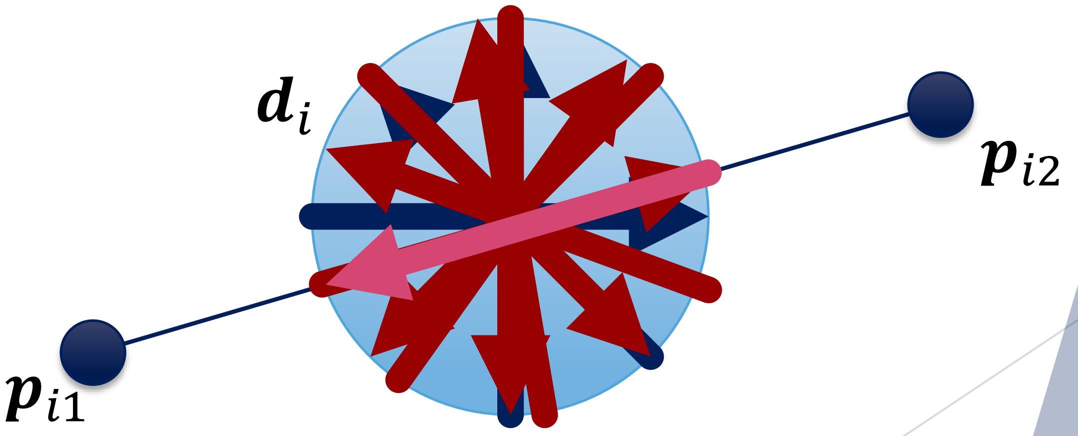
Hooke's Law with auxiliary variables

- For the j-th spring:
  - $E_{\mathrm{j}}(\boldsymbol{x}) = \frac{1}{2} k_{\mathrm{j}}\left(\|\boldsymbol{x}_{\mathrm{j}1} - \boldsymbol{x}_{\mathrm{j}2}\| - \mathrm{l}_{\mathrm{j}0}\right)^2$
- Introduce auxiliary variable $\mathbf{p}_{\mathrm{j}}$ where $\|\mathbf{p}_{\mathrm{j}}\| = \mathrm{l}_{\mathrm{j}0}$



Hooke's Law with auxiliary variables

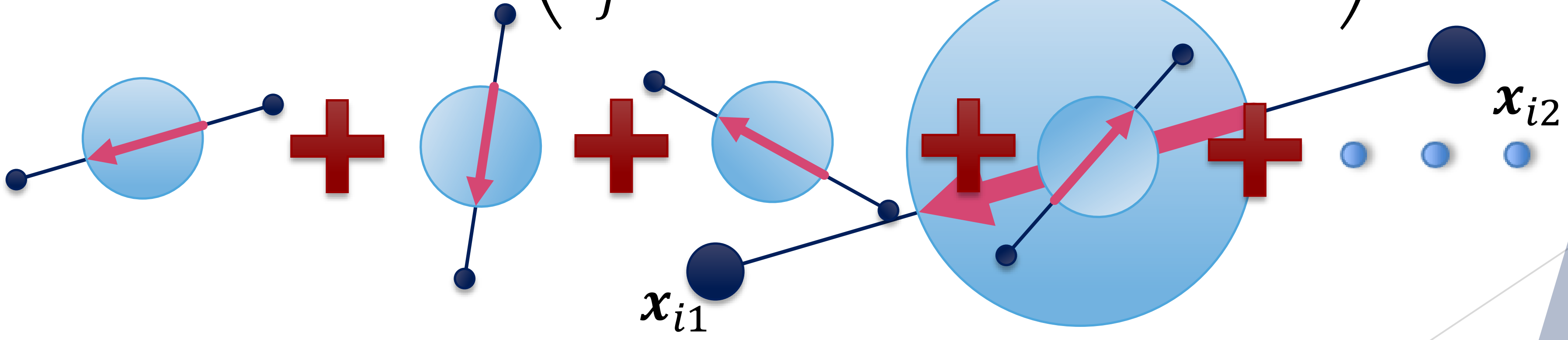
- $\min_{\|\boldsymbol{p}_j\|=l_{j0}} \boxed{\left(\frac{1}{2}k_j\left\|\boldsymbol{x}_{j1}-\boldsymbol{x}_{j2}-\boldsymbol{p}_j\right\|^2\right)} = \frac{1}{2}k_i\left(\left\|\boldsymbol{x}_{j1}-\boldsymbol{x}_{j2}\right\|-l_{j0}\right)^2$
  - When $\boldsymbol{p}_j = l_{j0}\frac{\boldsymbol{x}_{j1}-\boldsymbol{x}_{j2}}{\left\|\boldsymbol{x}_{j1}-\boldsymbol{x}_{j2}\right\|}$



Hooke's Law with auxiliary variables

$$E(\boldsymbol{x}) = \sum_j \left(\min_{\|\boldsymbol{p}_j\| = l_{0j}} \left(\frac{1}{2} k_j \left\| \boldsymbol{x}_{j1} - \boldsymbol{x}_{j2} - \boldsymbol{p}_j \right\|^2 \right) \right)$$

$$E(\boldsymbol{x}) = \min_{\boldsymbol{p} \in \mathcal{M}} \left(\sum_j \left(\frac{1}{2} k_j \left\| \boldsymbol{x}_{j1}^T - \boldsymbol{x}_{j2}^T - \boldsymbol{p}_j^T \right\|^2 \right) \right)$$



Hooke's Law with auxiliary variables

$\boldsymbol{x}_{j1}^T - \boldsymbol{x}_{j2}^T$: Discrete Shape Descriptor

$$\boldsymbol{x}_{j1}^T - \boldsymbol{x}_{j2}^T = \boldsymbol{G}_{\boldsymbol{j}}^{\boldsymbol{T}} \boldsymbol{x} \qquad \begin{array}{cl} \left[\begin{matrix} 0 \\ 0 \\ \vdots \\ 1 \\ \vdots \\ -1 \\ \vdots \\ 0 \\ 0 \end{matrix}\right] & \begin{matrix} \\ \\ \\ \boldsymbol{j1} \\ \\ \boldsymbol{j2} \\ \\ \\ \\ \end{matrix} \end{array} \qquad \begin{bmatrix} \boldsymbol{x}_1^T \\ \boldsymbol{x}_2^T \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \boldsymbol{x}_n^T \end{bmatrix}$$

$\boldsymbol{G}_{\boldsymbol{j}} \in \mathbb{R}^{\boldsymbol{n}\times\mathbf{1}}$ $\qquad$ $\boldsymbol{x} \in \mathbb{R}^{\boldsymbol{n}\times\mathbf{3}}$

Hooke's Law with auxiliary variables

$\boldsymbol{p}_j^T$: Projection

$$\boldsymbol{p}_j^T = \boldsymbol{S}_j^T \boldsymbol{p}$$

$$\left[\begin{array}{c} 0 \\ 0 \\ \vdots \\ 1 \\ \vdots \\ \vdots \\ \vdots \\ 0 \\ 0 \end{array}\right]\begin{array}{c} \\ \\ \\ \boldsymbol{j} \\ \\ \\ \\ \\ \\ \end{array} \qquad \begin{bmatrix} \boldsymbol{p}_1^T \\ \boldsymbol{p}_2^T \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \boldsymbol{p}_n^T \end{bmatrix}$$

$$\boldsymbol{S}_j \in \mathbb{R}^{\boldsymbol{m}\times\mathbf{1}} \qquad \boldsymbol{p} \in \mathbb{R}^{\boldsymbol{m}\times\mathbf{3}}$$

Hooke’s Law with auxiliary variables

$$E(\boldsymbol{x}) = \min_{\boldsymbol{p}\in\mathcal{M}} \left(\sum_j \left(\frac{1}{2} k_j \left\| \boldsymbol{x}_{j1}^T - \boldsymbol{x}_{j2}^T - \boldsymbol{p}_j^T \right\|^2 \right) \right)$$

$$E(\boldsymbol{x}) = \min_{\boldsymbol{p}\in\mathcal{M}} \frac{1}{2} tr\left(\boldsymbol{x}^T \underbrace{\left(\sum_j k_j \boldsymbol{G}_j \boldsymbol{G}_j^T \right)}_{\boldsymbol{L}} \boldsymbol{x} \right) - tr\left(\boldsymbol{x}^T \underbrace{\left(\sum_j k_j \boldsymbol{G}_j \boldsymbol{S}_j^T \right)}_{\boldsymbol{J}} \boldsymbol{p} \right) + \boldsymbol{C}$$

$$E(\boldsymbol{x}) = \min_{\boldsymbol{p}\in\mathcal{M}} \frac{1}{2} tr(\boldsymbol{x}^T \boldsymbol{L} \boldsymbol{x}) - tr(\boldsymbol{x}^T \boldsymbol{J} \boldsymbol{p}) + C$$

Variational Time Integration with Auxiliary Variable

$$\min_{x} \frac{1}{2h^2} tr\left((x-y)^T \boldsymbol{M}(x-y)\right) + E(x)$$

$$\min_{x\in\mathbb{R}^{n\times 3}, \boldsymbol{p}\in\mathcal{M}} \frac{1}{2h^2} tr\left((x-y)^T \boldsymbol{M}(x-y)\right) + \frac{1}{2} tr(\boldsymbol{x}^T \boldsymbol{L}\boldsymbol{x}) - tr(\boldsymbol{x}^T \boldsymbol{J}\boldsymbol{p}) + C$$

$$\boldsymbol{p} \in \mathcal{M} \Longleftrightarrow \|\boldsymbol{p}_{\boldsymbol{j}}\| = l_{j0}$$

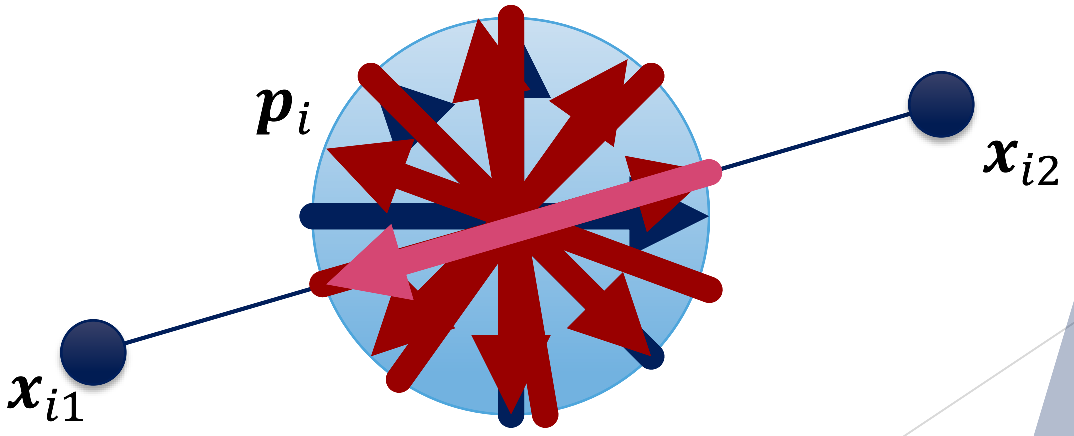
Optimization

$$\min_{x \in \mathbb{R}^{n\times 3}, \boldsymbol{p} \in \mathcal{M}} \frac{1}{2h^2} tr\left((x-y)^T \boldsymbol{M}(x-y)\right) + \frac{1}{2} tr(\boldsymbol{x}^T \boldsymbol{L}\boldsymbol{x}) - tr(\boldsymbol{x}^T \boldsymbol{J}\boldsymbol{p}) + C$$

- $\boldsymbol{M}, \boldsymbol{L}, \boldsymbol{J}, \boldsymbol{c}$ does not depend on $\boldsymbol{x}$ or $\boldsymbol{p}$
- If we fix $\boldsymbol{x}$ -> easy to solve for $\boldsymbol{p}$
- If we fix $\boldsymbol{p}$ -> easy to solve for $\boldsymbol{x}$
- Invites alternate solver (local/global)

Local Step

- For each spring, project to unit length using the current $\boldsymbol{x}$ to find $\boldsymbol{p_i}$
  - Trivially Parallelizable



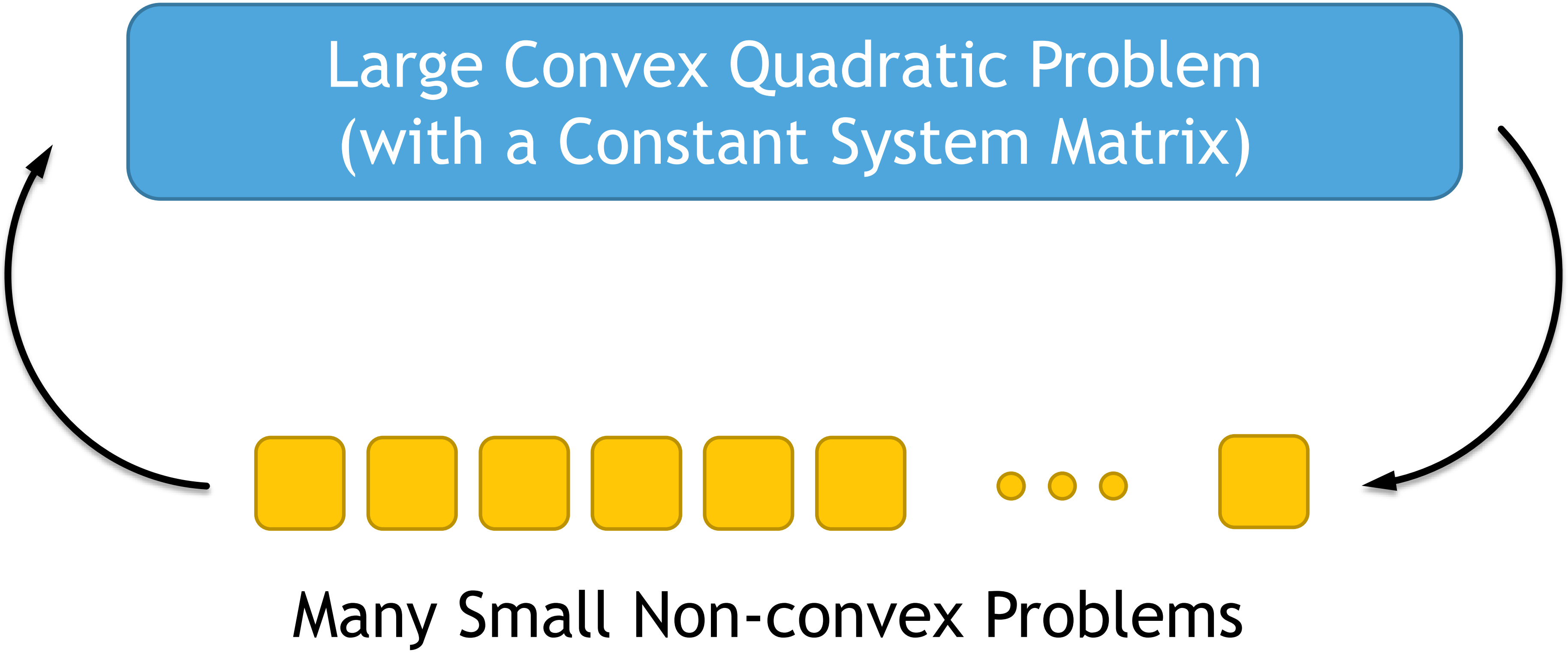
Global Step

$$\min_{x\in\mathbb{R}^{n\times 3},\boldsymbol{p}\in\mathcal{M}} \frac{1}{2h^2} tr\left((x-y)^T \boldsymbol{M}(x-y)\right) + \frac{1}{2} tr(\boldsymbol{x}^T \boldsymbol{L}\boldsymbol{x}) - tr(\boldsymbol{x}^T \boldsymbol{J}\boldsymbol{p}) + C$$

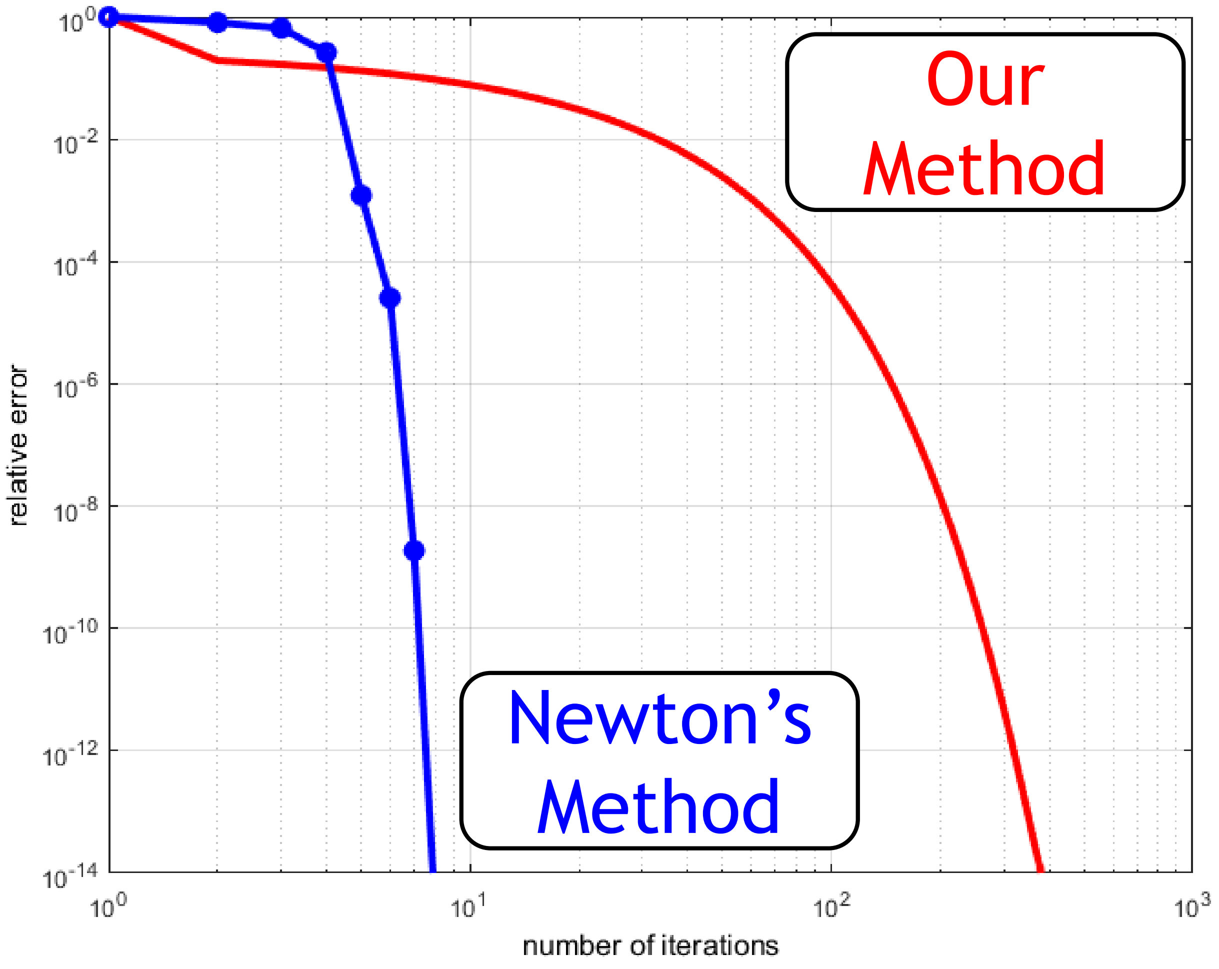
$$Fix\ \boldsymbol{p}: \boldsymbol{x}^* = \left(\frac{\boldsymbol{M}}{h^2} + \boldsymbol{L}\right)^{-1} \left(\frac{\boldsymbol{M}}{h^2}\boldsymbol{y} + \boldsymbol{J}\boldsymbol{p}\right)$$

- System matrix $(\mathbf{M}/\mathrm{h}^2 + \boldsymbol{L})$ is:
  - Independent of $\boldsymbol{x}$ and $\boldsymbol{p}$ (Constant)
  - Positive Definite
- Thus can be **pre-factorized** (using e.g. Cholesky)

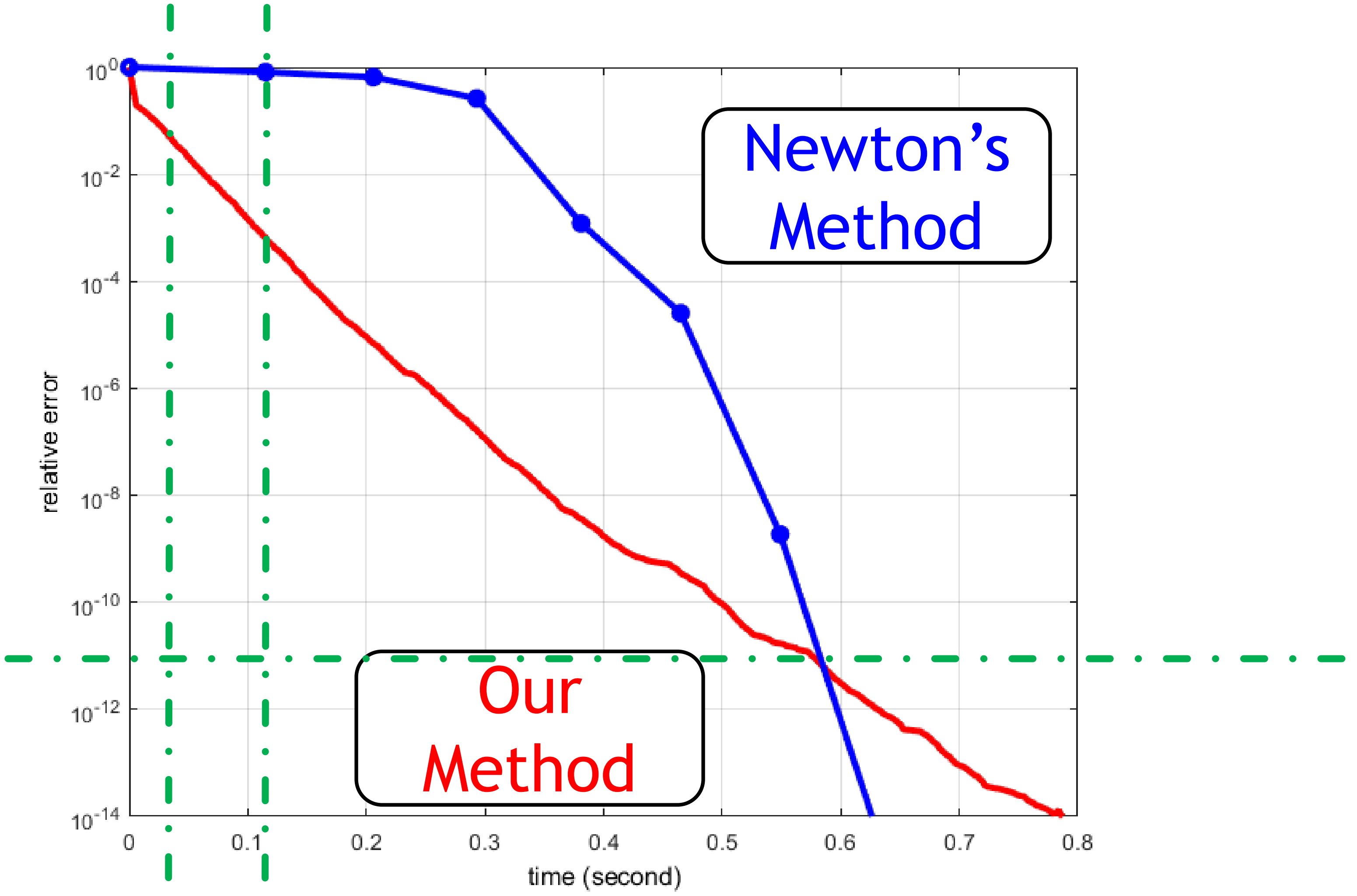
Alternating Solver



Performance



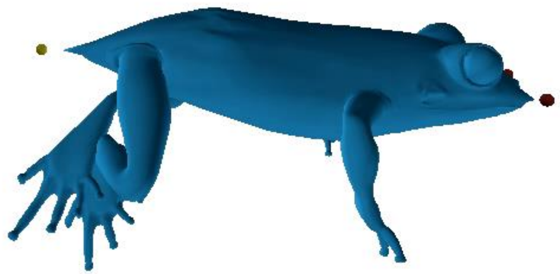
Performance



Results: Mass-spring Systems

Results: Mass-spring Systems

Results: Mass-spring Systems



Remark: Fast Mass-spring Systems

$$\min_{x} \frac{1}{2h^2} tr\left((x-y)^T \boldsymbol{M}(x-y)\right) + E(x)$$

$$\frac{1}{2} k_j \left(\|\boldsymbol{x}_{j1} - \boldsymbol{x}_{j2}\| - l_{j0}\right)^2 = \min_{\|\boldsymbol{p}_j\| = l_{j0}} \left(\frac{1}{2} k_j \|\boldsymbol{x}_{j1} - \boldsymbol{x}_{j2} - \boldsymbol{p}_j\|^2\right)$$

$$\min_{x \in \mathbb{R}^{n\times 3}, \boldsymbol{p} \in \mathcal{M}} \frac{1}{2h^2} tr\left((x-y)^T \boldsymbol{M}(x-y)\right) + \frac{1}{2} tr(\boldsymbol{x}^T \boldsymbol{L} \boldsymbol{x}) - tr(\boldsymbol{x}^T \boldsymbol{J} \boldsymbol{p}) + C$$

Remark: Fast Mass-spring Systems

Formulate IE as an Optimization Problem

$$\min_{x} \frac{1}{2h^2}(x-y)^T M(x-y) + E(x)$$

Formulate Hooke's Law with an Auxiliary Variable

$$\frac{1}{2}k_j\left(\|p_{j1}-p_{j2}\| - l_{j0}\right)^2 = \min_{\|p_j\|=l_{j0}} \left(\frac{1}{2}k_j\|p_{j1}-p_{j2}-p_j\|^2\right)$$

Local/global Solve

$$\min_{x\in\mathbb{R}^{n\times 3},\, p\in M} \frac{1}{2h^2} tr\left((x-y)^T M(x-y)\right) + \frac{1}{2}tr(x^T L x) - tr(x^T J p) + C$$

Remark: Fast Mass-spring Systems

$$\min_{x} \frac{1}{2h^2} tr\left((x-y)^T \boldsymbol{M}(x-y)\right) + E(x)$$

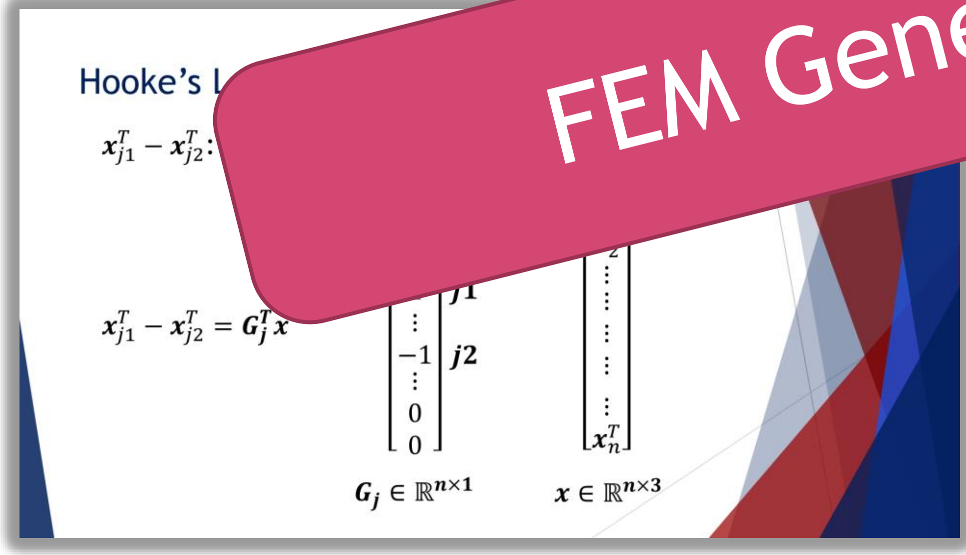
$$\frac{1}{2}k_j\left(\|\boldsymbol{x}_{j1}-\boldsymbol{x}_{j2}\| - l_{j0}\right)^2 = \boxed{\min_{\|\boldsymbol{p}_j\|=l_{j0}} \left(\frac{1}{2}k_j\|\boldsymbol{x}_{j1}-\boldsymbol{x}_{j2}-\boldsymbol{p}_j\|^2\right)}$$

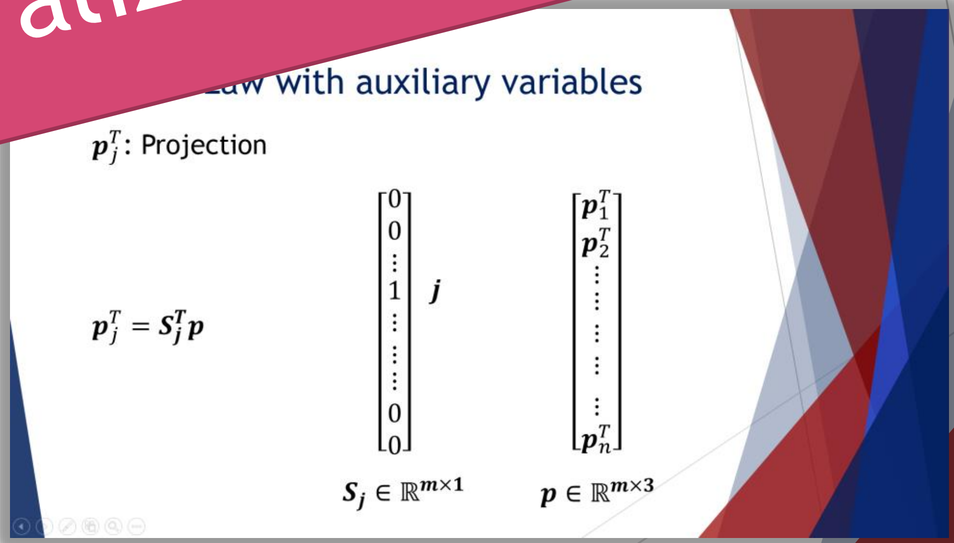
$$\min_{x\in\mathbb{R}^{n\times 3},\boldsymbol{p}\in\mathcal{M}} \frac{1}{2h^2} tr\left((x-y)^T \boldsymbol{M}(x-y)\right) + \frac{1}{2}tr(\boldsymbol{x}^T\boldsymbol{L}\boldsymbol{x}) - tr(\boldsymbol{x}^T\boldsymbol{J}\boldsymbol{p}) + C$$

Remark: Fast Mass-spring Systems

$$\min_{\|\boldsymbol{p}_j\|=l_{j0}} \left(\frac{1}{2}k_j\left\|\boldsymbol{x}_{j1}-\boldsymbol{x}_{j2}-\boldsymbol{p}_j\right\|^2\right)$$

FEM Generalization?





Projective Dynamics

Projective Dynamics: Fusing Constraint Projections for Fast Simulation

Sofien Bouaziz[*] EPFL | Sebastian Martin[†] VM Research | Tiantian Liu[‡] University of Pennsylvania | Ladislav Kavan[§] University of Pennsylvania | Mark Pauly[¶] EPFL

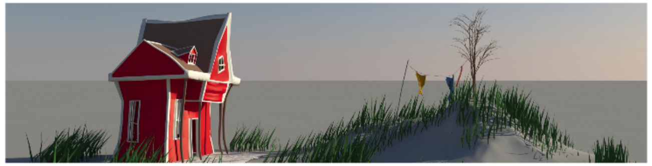


Figure 1: *We propose a new "projection-based" implicit Euler integrator that supports a large variety of geometric constraints in a single physical simulation framework. In this example, all the elements including building, grass, tree, and clothes (49k DoFs, 43k constraints), are simulated at 3.1ms/iteration using 10 iterations per frame (see also accompanying video).*

Abstract

We present a new method for implicit time integration of physical systems. Our approach builds a bridge between nodal Finite Element methods and Position Based Dynamics, leading to a simple, efficient, robust, yet accurate solver that supports many different types of constraints. We propose specially designed energy potentials that can be solved efficiently using an alternating optimization approach. Inspired by continuum mechanics, we derive a set of continuum-based potentials that can be efficiently incorporated within our solver. We demonstrate the generality and robustness of our approach in many different applications ranging from the simulation of solids, cloths, and shells, to example-based simulation. Comparisons to Newton-based and Position Based Dynamics solvers highlight the benefits of our formulation.

CR Categories: I.3.7 [Computer Graphics]: Three-Dimensional Graphics—Animation; I.6.8 [Simulation and Modeling]: Types of Simulation—Animation

Keywords: physics-based animation, implicit Euler method, position based dynamics, continuum mechanics.

Links: DL PDF

[*]sofien.bouaziz@epfl.ch
[†]sebimartin@gmail.com
[‡]ltt1598@gmail.com
[§]ladislav.kavan@gmail.com
[¶]mark.pauly@epfl.ch

1 Introduction

Physics-based simulation of deformable material has become an indispensable tool in many areas of computer graphics. Virtual worlds, and more recently character animations, incorporate sophisticated simulations to greatly enhance visual experience, e.g., by simulating muscles, fat, hair, clothing, or vegetation. These models are often based on finite element discretizations of continuum-mechanics formulations, allowing highly *accurate* simulation of complex nonlinear materials.

Besides realism and accuracy, a number of other criteria are also important in computer graphics applications. By *generality* we mean the ability to simulate a large spectrum of behaviors, such as different types of geometries (solids, shells, rods), different material properties, or even art-directable extensions to classic physics-based simulation. *Robustness* refers to the capability to adequately handle difficult configurations, including large deformations, degenerate geometries, and large time steps. Robustness is especially important in real-time applications where there is no "second chance" to re-run a simulation, such as in computer games or medical training simulators. The *simplicity* of a solver is often important for its practical relevance. Building on simple, easily understandable concepts – and the resulting lightweight codebases – eases the maintenance of simulators and makes them adaptable to specific application needs. *Performance* is a critical enabling criterion for realtime applications. However, performance is no less important in offline simulations, where the turnaround time for testing new scenes and simulation parameters should be minimized.

Current continuum mechanics approaches often have unfavorable trade-offs between these criteria for certain computer graphics applications, which led to the development of alternative methods, such as Position Based Dynamics (PBD). Due to its generality, simplicity, robustness, and efficiency, PBD is now implemented in a wide range of high-end products including PhysX, Havok Cloth, Maya nCloth, and Bullet. While predominantly used in realtime applications, PBD is also often used in offline simulation. However, the desirable qualities of PBD come at the cost of limited accuracy, because PBD is not rigorously derived from continuum mechanical principles.

We propose a new implicit integration solver that bridges the gap

Projective Dynamics: Fusing Constraint Projections for Fast Simulation
Sofien Bouaziz, Sebastian Martin, Tiantian Liu, Ladislav Kavan, Mark Pauly
ACM Transactions on Graphics 33(4) [Proceedings of SIGGRAPH], 2014

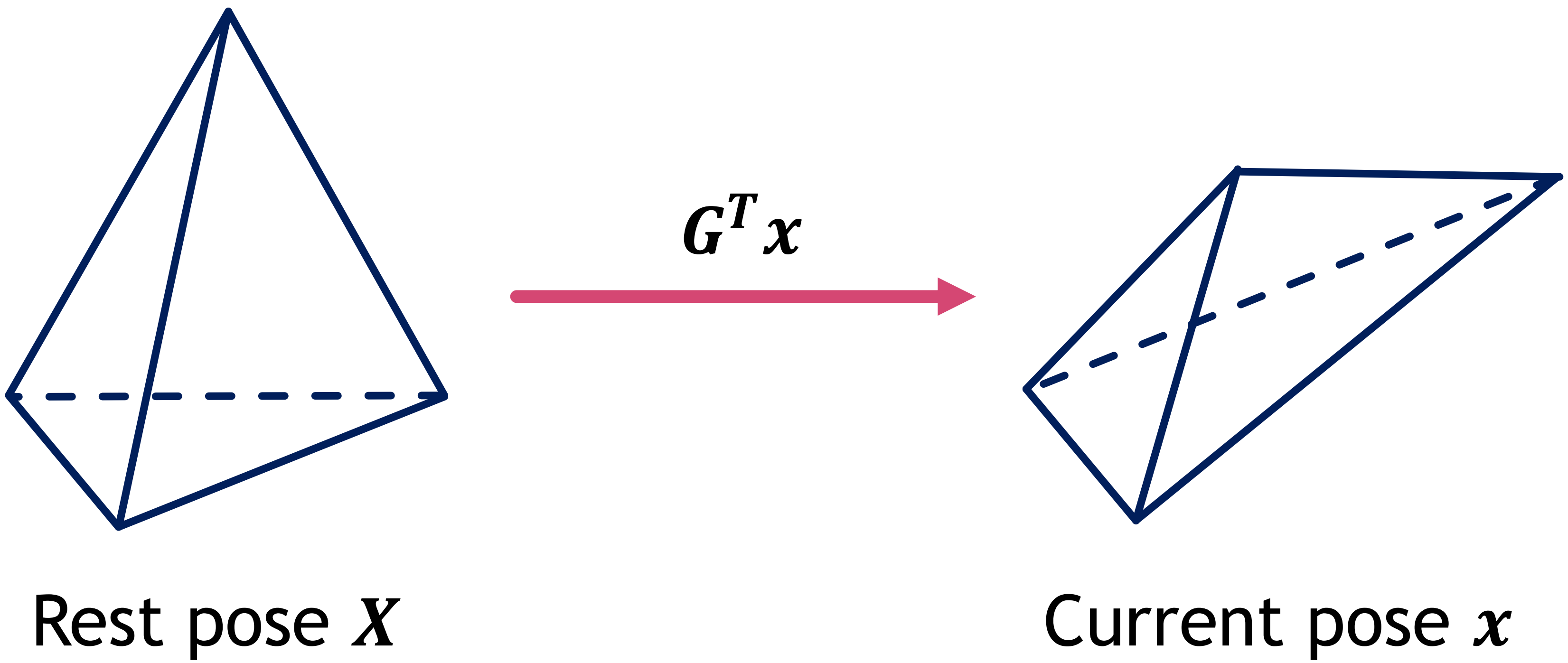
Key Idea of Fast Mass-Spring Systems

$$E(\boldsymbol{x}) = \min_{\boldsymbol{p}\in\mathcal{M}} \left(\sum_j \left(w_j \left\| \boldsymbol{G}_j^T \boldsymbol{x} - \boldsymbol{p}_j \right\|^2 \right) \right)$$

$$\left\| \mathit{Discrete\ Shape\ Descriptor} - \mathit{Projection} \right\|^2$$

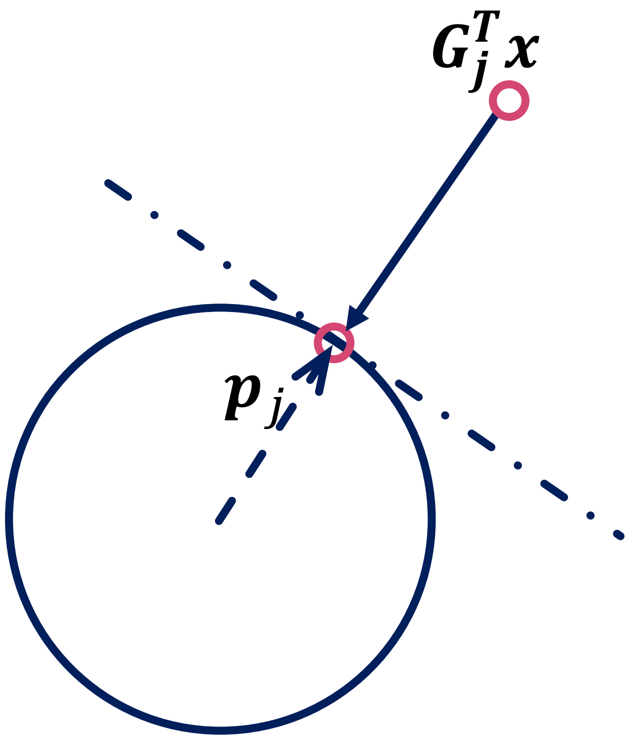
Other Discrete Shape Descriptors

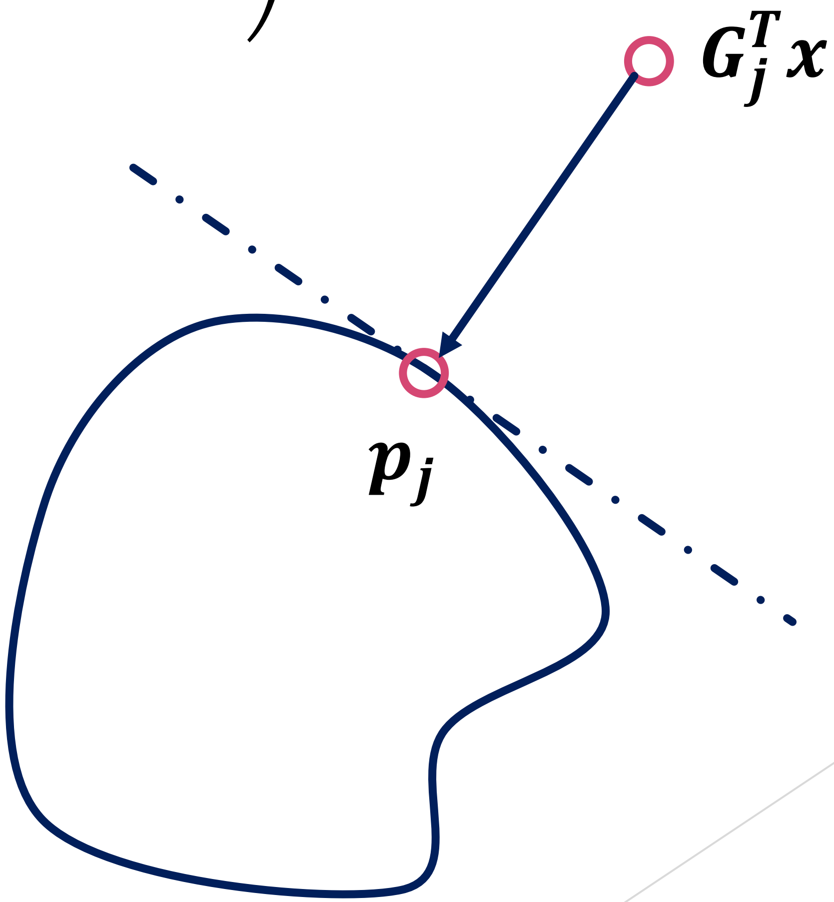
$$E(\boldsymbol{x}) = \min_{\boldsymbol{p}\in\mathcal{M}} \left(\sum_j \left(w_j \left\| \boldsymbol{G}_j^T \boldsymbol{x} - \boldsymbol{p}_j \right\|^2 \right) \right)$$



Other Manifolds

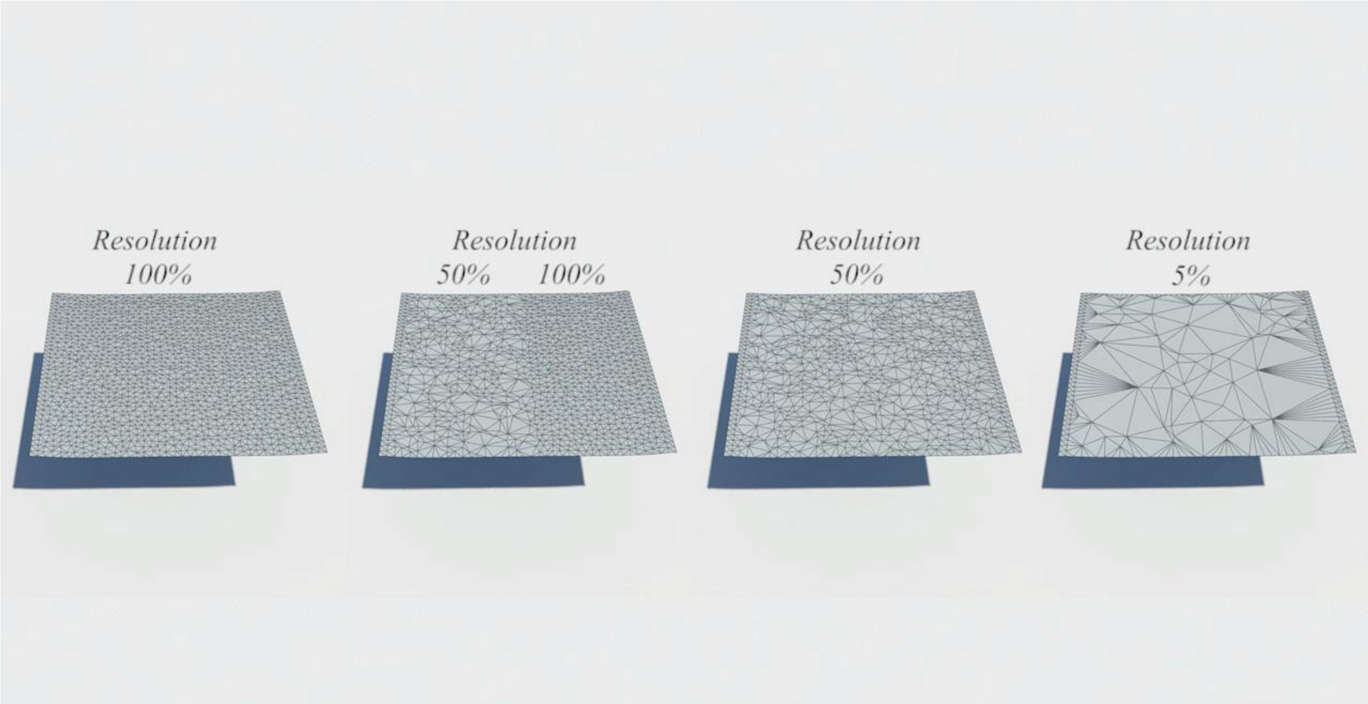
$$E(\boldsymbol{x}) = \min_{\boldsymbol{p}\in\mathcal{M}} \left(\sum_j \left(w_j \left\| \boldsymbol{G}_j^T \boldsymbol{x} - \boldsymbol{p}_j \right\|^2 \right) \right)$$





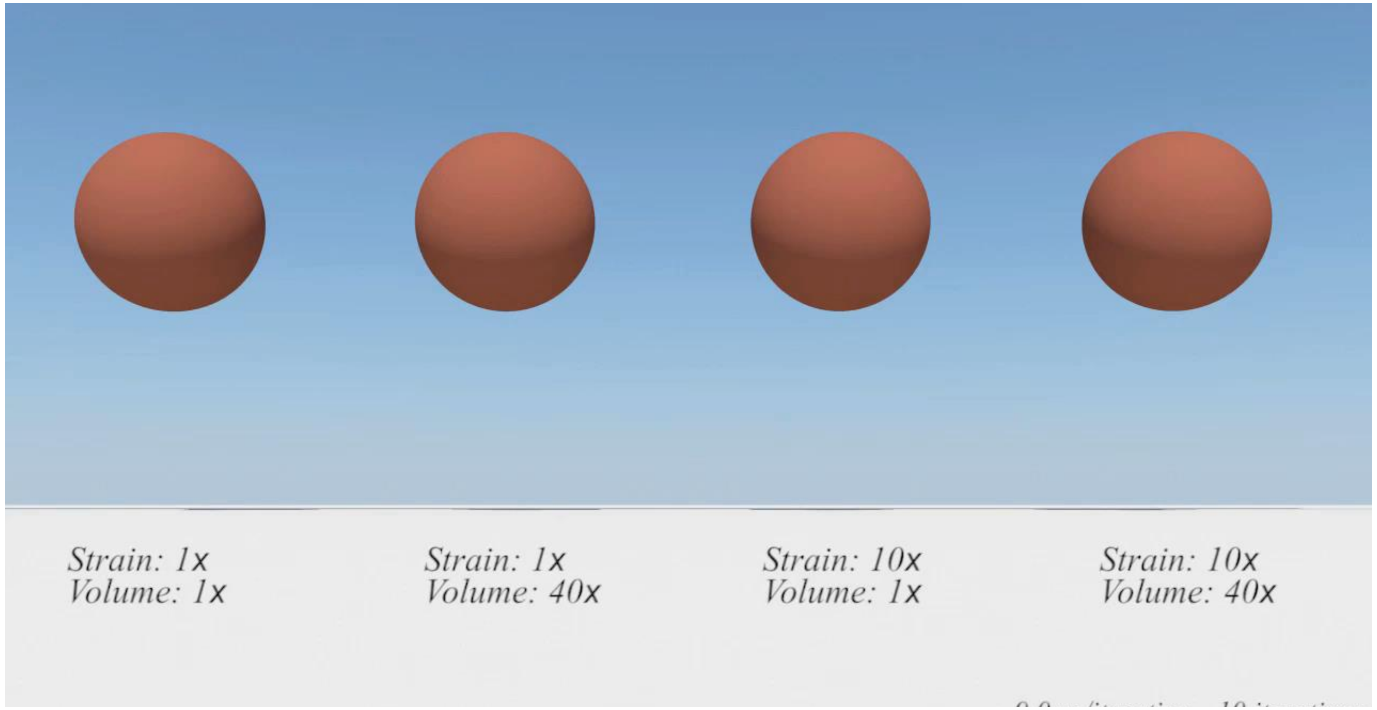
Intuitive Projection Manifold: SO(3)

- SO(3) … Best Fit Rotation Matrix
- “As Rigid As Possible” [Chao et al. 2010]

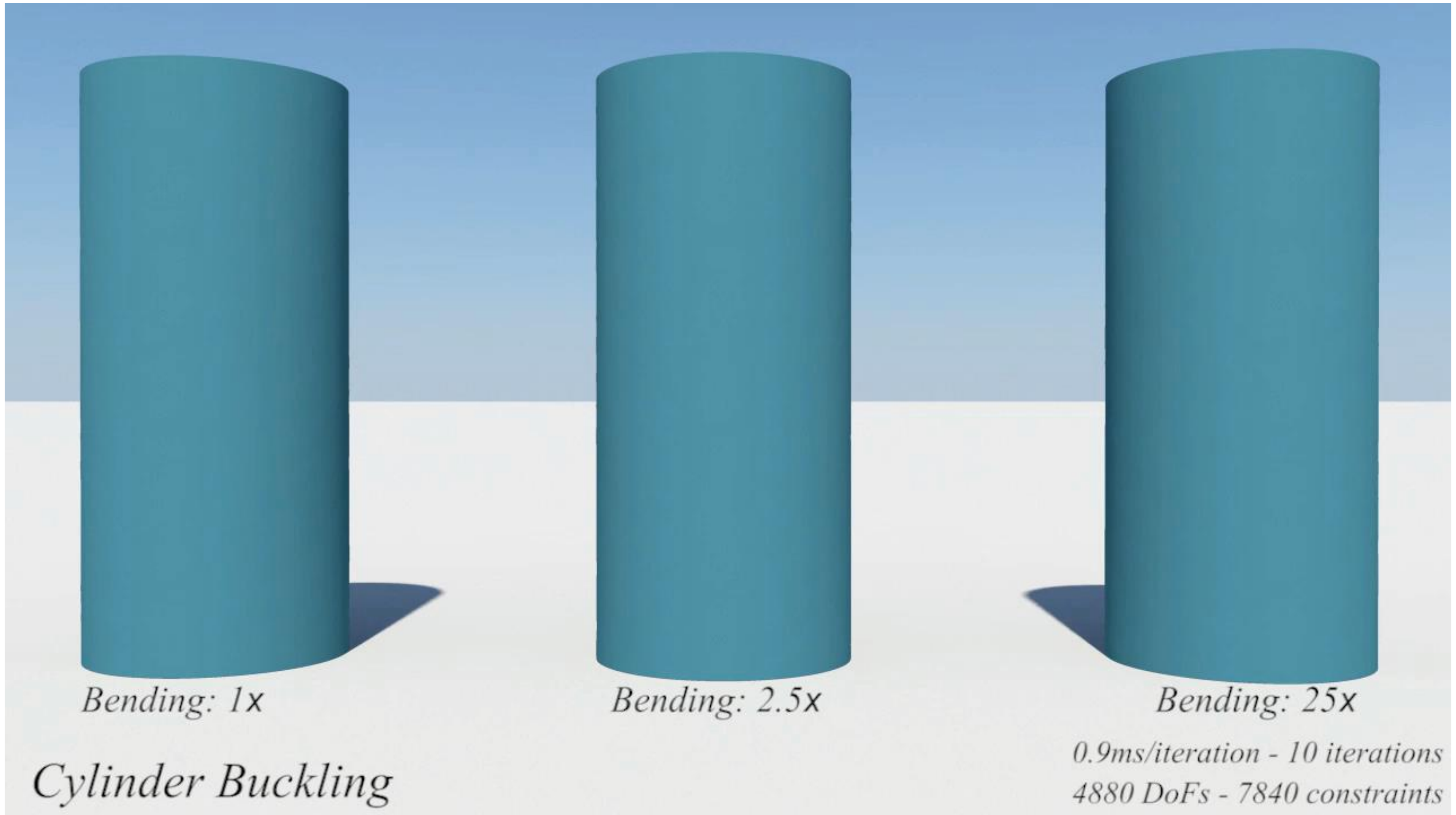


Intuitive Projection Manifold: SL(3)

- SL(3) … Group of Matrices with det = 1
- Volume Preservation



More Discrete Shape Descriptors:
Laplace-Beltrami operator



Results: Projective Dynamics

Remark: Projective Dynamics

$$E(\boldsymbol{x}) = \min_{\boldsymbol{p}\in\mathcal{M}} \left(\sum_j \left(w_j \left\| \boldsymbol{G}_j^T \boldsymbol{x} - \boldsymbol{p}_j \right\|^2 \right) \right)$$

$$\min_{x\in\mathbb{R}^{n\times 3}, \boldsymbol{p}\in\mathcal{M}} \frac{1}{2h^2} tr\left((x-y)^T \boldsymbol{M}(x-y)\right) + \frac{1}{2} tr(\boldsymbol{x}^T \boldsymbol{L}\boldsymbol{x}) - tr(\boldsymbol{x}^T \boldsymbol{J}\boldsymbol{p}) + C$$

- Like before, $\boldsymbol{M}, \boldsymbol{L}, \boldsymbol{J}, \boldsymbol{c}$ does not depend on $\boldsymbol{x}$ and $\boldsymbol{p}$
- If we fix $\boldsymbol{x}$ -> easy to solve for $\boldsymbol{p}$: Projection
- If we fix $\boldsymbol{p}$ -> easy to solve for $\boldsymbol{x}$: $\boldsymbol{x}^* = \left(\frac{\boldsymbol{M}}{h^2} + \boldsymbol{L}\right)^{-1} \left(\frac{\boldsymbol{M}}{h^2}\boldsymbol{y} + \boldsymbol{J}\boldsymbol{p}\right)$

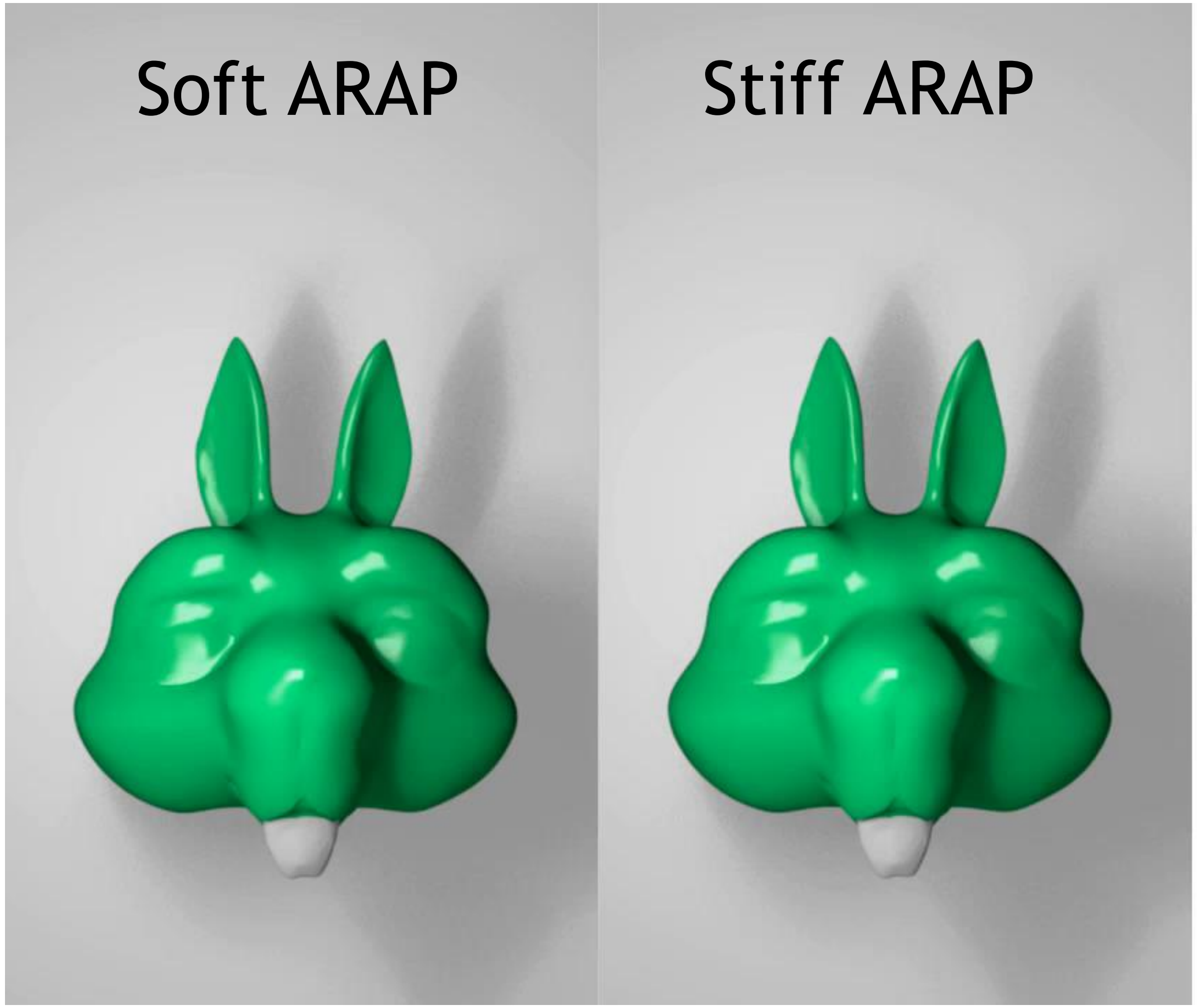
Limitation: Projective Dynamics

$$E(\boldsymbol{x}) = \min_{\boldsymbol{p}\in\mathcal{M}} \left(\sum_j \left(w_j \underbrace{\left\| \boldsymbol{G}_j^T \boldsymbol{x} - \boldsymbol{p}_j \right\|^2}_{} \right) \right)$$

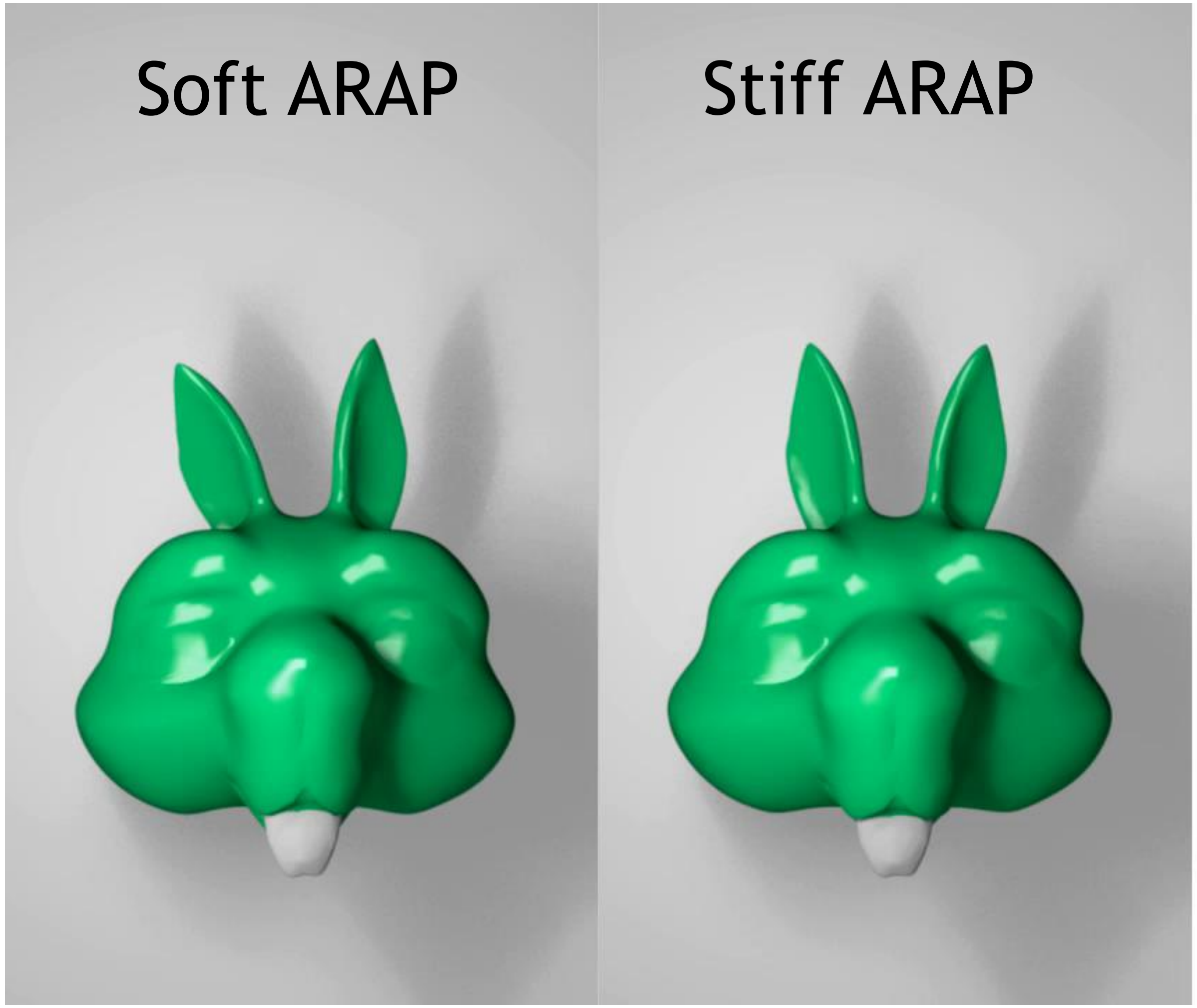
$$\| \mathit{Discrete\ Shape\ Descriptor} - \mathit{Projection} \|^2$$

Special Requirement for the Energy Representation

More Materials?



Spline-Based Materials [Xu et al. 2015]



Polynomial Material [Xu et al. 2015]

Quasi-Newton Methods for Real-Time Simulation of Hyperelastic Materials

23

Quasi-Newton Methods for Real-Time Simulation of Hyperelastic Materials

TIANTIAN LIU
University of Pennsylvania
SOFIEN BOUAZIZ
École polytechnique fédérale de Lausanne
and
LADISLAV KAVAN
University of Utah

We present a new method for real-time physics-based simulation supporting many different types of hyperelastic materials. Previous methods such as Position-Based or Projective Dynamics are fast but support only a limited selection of materials; even classical materials such as the Neo-Hookean elasticity are not supported. Recently, Xu et al. [2015] introduced new "spline-based materials" that can be easily controlled by artists to achieve desired animation effects. Simulation of these types of materials currently relies on Newton's method, which is slow, even with only one iteration per timestep. In this article, we show that Projective Dynamics can be interpreted as a quasi-Newton method. This insight enables very efficient simulation of a large class of hyperelastic materials, including the Neo-Hookean, spline-based materials, and others. The quasi-Newton interpretation also allows us to leverage ideas from numerical optimization. In particular, we show that our solver can be further accelerated using L-BFGS updates (Limited-memory Broyden-Fletcher-Goldfarb-Shanno algorithm). Our final method is typically more than 10 times faster than one iteration of Newton's method without compromising quality. In fact, our result is often more accurate than the result obtained with one iteration of Newton's method. Our method is also easier to implement, implying reduced software development costs.

Categories and Subject Descriptors: I.3.7 [**Computer Graphics**]: Three-Dimensional Graphics and Realism—*Animation*

General Terms: Physics-based Animation

Additional Key Words and Phrases: Physics-based animation, material models, numerical optimization

Authors' addresses: L. Kavan (correspondence author), School of Computing, University of Utah, Salt Lake City, UT; email: ladislav.kavan@gmail.com.

DOI: http://dx.doi.org/10.1145/2990496

ACM Reference Format:

T. Liu, S. Bouaziz, and L. Kavan. 2017. Quasi-newton methods for real-time simulation of hyperelastic materials. ACM Trans. Graph. 36, 3, Article 23 (May 2017), 16 pages.
DOI: http://dx.doi.org/10.1145/2990496

1. INTRODUCTION

Physics-based animation is an important tool in computer graphics, even though creating visually compelling simulations often requires a lot of patience. Waiting for results is not an option in real-time simulations, which are necessary in applications such as computer games and training simulators (e.g., surgery simulators). Previous methods for real-time physics such as Position-Based Dynamics [Müller et al. 2007] or Projective Dynamics [Bouaziz et al. 2014] have been successfully used in many applications and commercial products, despite the fact that these methods support only a restricted set of material models. Even classical models from continuum mechanics, such as the Neo-Hookean, St. Venant-Kirchoff, or Mooney-Rivlin materials, are not supported by Projective Dynamics. We tried to emulate their behavior with Projective Dynamics, but despite our best efforts, there are still obvious visual differences when compared to simulations with the original nonlinear materials.

The advantages of more general material models were nicely demonstrated in the recent work of Xu et al. [2015], who proposed a new class of spline-based materials particularly suitable for physics-based animation. Their user-friendly spline interface enables artists to easily modify material properties in order to achieve desired animation effects. However, their system relies on Newton's method, which is slow, even if the number of Newton's iterations per frame is limited to one. Our method enables fast simulation of spline-based materials, combining the benefits of artist-friendly material interfaces with the advantages of fast simulation, such as rapid iterations and/or higher resolutions.

Physics-based simulation can be formulated as an optimization problem where we minimize a multivariate function g. Newton's method minimizes g by performing descent along direction $-(\nabla^2 g)^{-1}\nabla g$, where $\nabla^2 g$ is the Hessian matrix, and ∇g is the gradient. One problem of Newton's method is that the Hessian $\nabla^2 g$ can be indefinite, in which case Newton's direction could erroneously *increase* g. This undesired behavior can be prevented by so-called definiteness fixes [Teran et al. 2005; Nocedal and Wright 2006]. While definiteness fixes require some computational overheads, the slow speed of Newton's method is mainly caused by the fact that

Quasi-Newton Methods for Real-Time Simulation of Hyperelastic Materials
Tiantian Liu, Sofien Bouaziz, Ladislav Kavan
ACM Transactions on Graphics 36(3) [Presented at SIGGRAPH],2017.

30 JULY—3 AUGUST Los Angeles
SIGGRAPH2017

Reformulation of Projective Dynamics

$$\min_{x\in\mathbb{R}^{n\times 3},\boldsymbol{p}\in\boldsymbol{\mathcal{M}}} \frac{1}{2h^2} tr\left((x-y)^T \boldsymbol{M}(x-y)\right) + \frac{1}{2} tr(\boldsymbol{x}^T \boldsymbol{L}\boldsymbol{x}) - tr(\boldsymbol{x}^T \boldsymbol{J}\boldsymbol{p}) + C$$

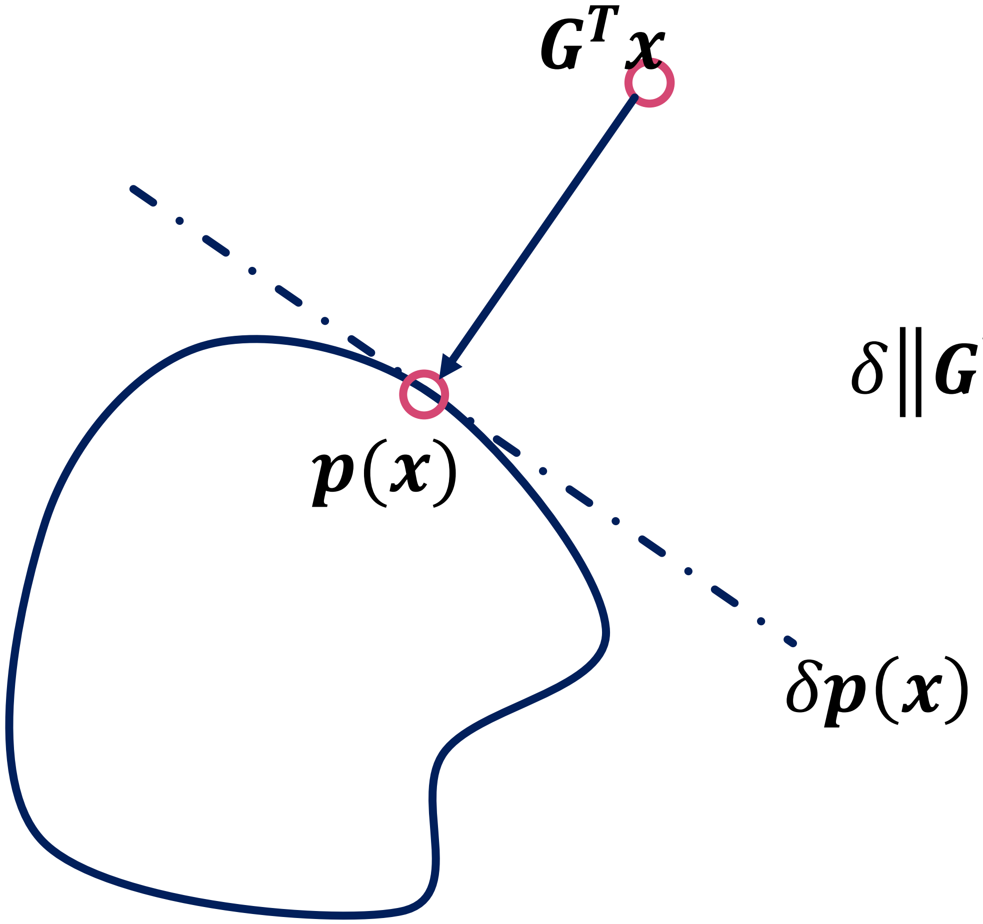
$$\min_{x\in\mathbb{R}^{n\times 3}} \underbrace{\frac{1}{2h^2} tr\left((x-y)^T \boldsymbol{M}(x-y)\right) + \frac{1}{2} tr(\boldsymbol{x}^T \boldsymbol{L}\boldsymbol{x}) - tr(\boldsymbol{x}^T \boldsymbol{J}\boldsymbol{p}(\boldsymbol{x})) + \frac{1}{2} tr\left(\boldsymbol{p}(\boldsymbol{x})^T \boldsymbol{S}\boldsymbol{p}(\boldsymbol{x})\right)}_{\boldsymbol{g}(\boldsymbol{x})}$$

Reformulation of Projective Dynamics

$$\min_{x\in\mathbb{R}^{n\times 3}} \underbrace{\frac{1}{2h^2}tr\left((x-y)^T\boldsymbol{M}(x-y)\right)+\frac{1}{2}tr(\boldsymbol{x}^T\boldsymbol{L}\boldsymbol{x})-tr(\boldsymbol{x}^T\boldsymbol{J}\boldsymbol{p}(\boldsymbol{x}))+\frac{1}{2}tr\left(\boldsymbol{p}(\boldsymbol{x})^T\boldsymbol{S}\boldsymbol{p}(\boldsymbol{x})\right)}_{\boldsymbol{g}(\boldsymbol{x})}$$

$$\nabla\boldsymbol{g}(\boldsymbol{x})=\boxed{\frac{\boldsymbol{M}}{h^2}(\boldsymbol{x}-\boldsymbol{y})+\boldsymbol{L}\boldsymbol{x}-\boldsymbol{J}\boldsymbol{p}(\boldsymbol{x})}+\underbrace{\boxed{\frac{\partial\boldsymbol{p}(\boldsymbol{x})}{\partial\boldsymbol{x}}:(\boldsymbol{S}\boldsymbol{p}(\boldsymbol{x})-\boldsymbol{J}^T\boldsymbol{x})}}_{0}$$

Projection Differential



$$\delta\left\|\boldsymbol{G}^T\boldsymbol{x}-\boldsymbol{p}(\boldsymbol{x})\right\|^2=\left(\boldsymbol{G}^T\boldsymbol{x}-\boldsymbol{p}(\boldsymbol{x})\right)^T\boldsymbol{G}^T\delta\boldsymbol{x}$$

$$\boxed{\cancel{-\delta\boldsymbol{p}(\boldsymbol{x})^T\left(\boldsymbol{G}^T\boldsymbol{x}-\boldsymbol{p}(\boldsymbol{x})\right)}}$$

Reformulation of Projective Dynamics

$$\underbrace{\min_{x\in\mathbb{R}^{n\times 3}} \frac{1}{2h^2} tr\left((x-y)^T \boldsymbol{M}(x-y)\right) + \frac{1}{2} tr(\boldsymbol{x}^T \boldsymbol{L}\boldsymbol{x}) - tr(\boldsymbol{x}^T \boldsymbol{J}\boldsymbol{p}(\boldsymbol{x})) + \frac{1}{2} tr\left(\boldsymbol{p}(\boldsymbol{x})^T \boldsymbol{S}\boldsymbol{p}(\boldsymbol{x})\right)}_{\boldsymbol{g}(\boldsymbol{x})}$$

$$\nabla \boldsymbol{g}(\boldsymbol{x}) = \frac{\boldsymbol{M}}{h^2}(\boldsymbol{x}-\boldsymbol{y}) + \boldsymbol{L}\boldsymbol{x} - \boldsymbol{J}\boldsymbol{p}(\boldsymbol{x}) + \cancel{\frac{\partial \boldsymbol{p}(\boldsymbol{x})}{\partial \boldsymbol{x}} : (\boldsymbol{S}\boldsymbol{p}(\boldsymbol{x}) - \boldsymbol{J}^T\boldsymbol{x})}$$

$$(\frac{\boldsymbol{M}}{h^2} + \mathbf{L})^{-1} \nabla \boldsymbol{g}(\boldsymbol{x}) = \boldsymbol{x} - \underbrace{\left(\frac{\boldsymbol{M}}{h^2} + \boldsymbol{L}\right)^{-1} \left(\frac{\boldsymbol{M}}{h^2}\boldsymbol{y} + \boldsymbol{J}\boldsymbol{p}\right)}_{\boldsymbol{x}^*}$$

$$\boldsymbol{x}^* = \boldsymbol{x} - (\boldsymbol{M}/h^2 + \boldsymbol{L})^{-1} \nabla \boldsymbol{g}(\boldsymbol{x})$$

Reformulation of Projective Dynamics

Compare to one Newton step:

$$\boldsymbol{x}^* = \boldsymbol{x} - \boldsymbol{\alpha}[\nabla^2 g(\boldsymbol{x})\ ]^{-1}\nabla g(\boldsymbol{x})$$

- α: Step size, usually decided by linesearch, typical value is 1.
- $\nabla^2 g(\boldsymbol{x})$: Hessian Matrix, $\boldsymbol{M}/h^2 + \nabla^2 E(\boldsymbol{x})$

$$\boldsymbol{x}^* = \boldsymbol{x} - (\boldsymbol{M}/h^2 + \boldsymbol{L})^{-1}\nabla \boldsymbol{g}(\boldsymbol{x})$$

Quasi-Newton Formulation

$$\boldsymbol{x}^* = \boldsymbol{x} - \alpha(\boldsymbol{M}/h^2 + \boldsymbol{L})^{-1}\nabla\boldsymbol{g}(\boldsymbol{x})$$

$$\alpha = 1$$

Projective Dynamics:
A Quasi Newton method applied
on a special type of energy

Supporting More General Materials

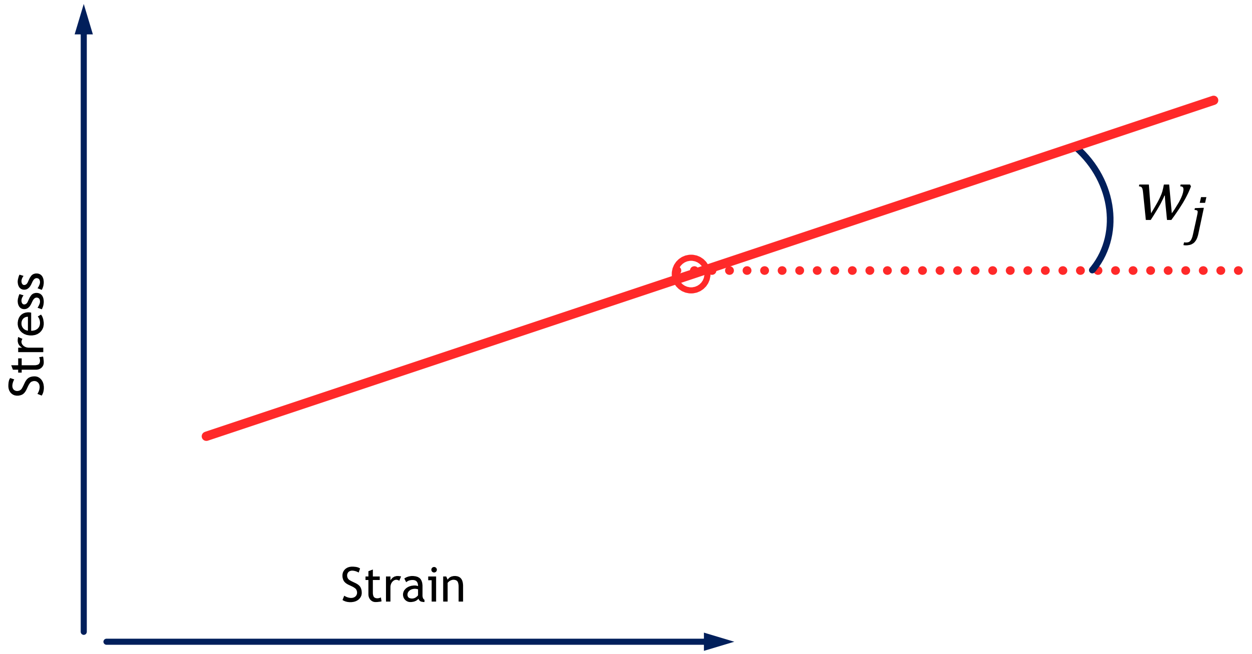
$$\boldsymbol{x}^* = \boldsymbol{x} - \alpha(\boldsymbol{M}/h^2 + \boldsymbol{L})^{-1}\nabla\boldsymbol{g}(\boldsymbol{x})$$

This quasi-Newton formulation can be used for any hyperelastic material, but:

- We need to do line-search
  - $\alpha = 1$ only works for Projective Dynamics
- We need to define the proper weights w_i
  - $\boldsymbol{M}/h^2 + \sum_j \boxed{w_j} \boldsymbol{G}_j \boldsymbol{G}_j^T$

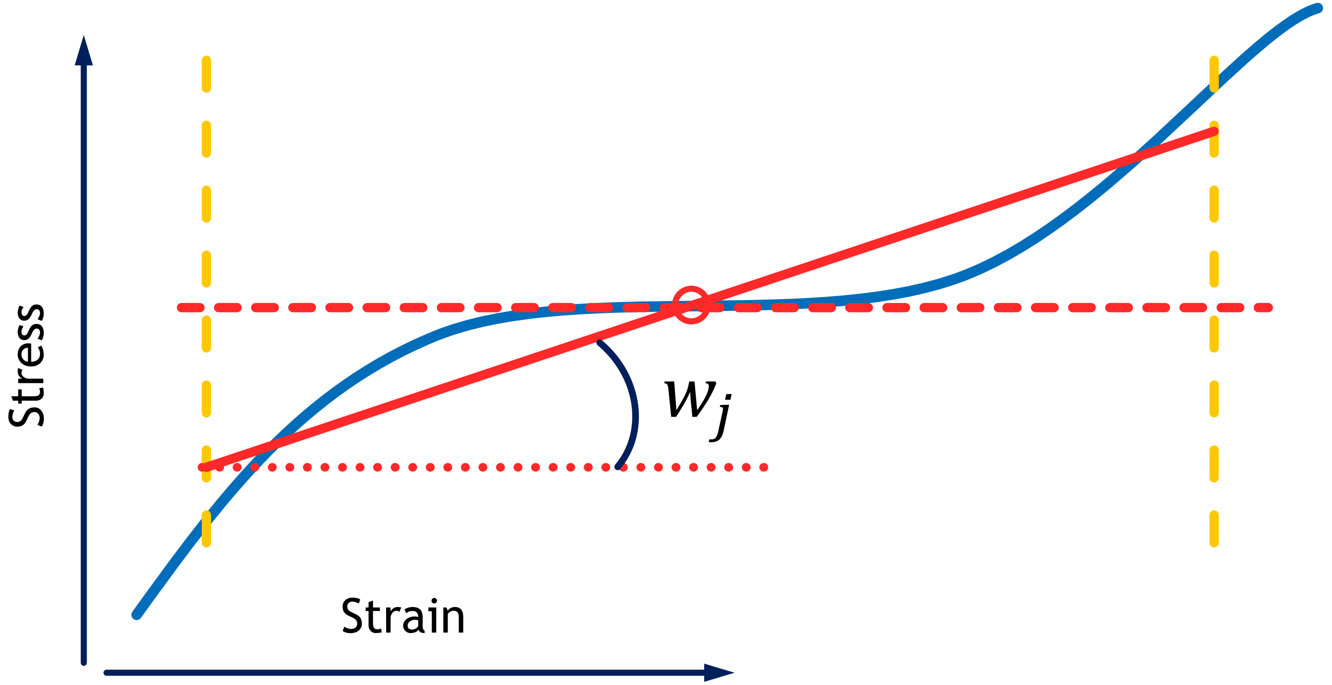
Strain-Stress Curve for PD

- $\boldsymbol{M}/h^2 + \sum_j \boxed{w_j} \boldsymbol{G}_j \boldsymbol{G}_j^T$

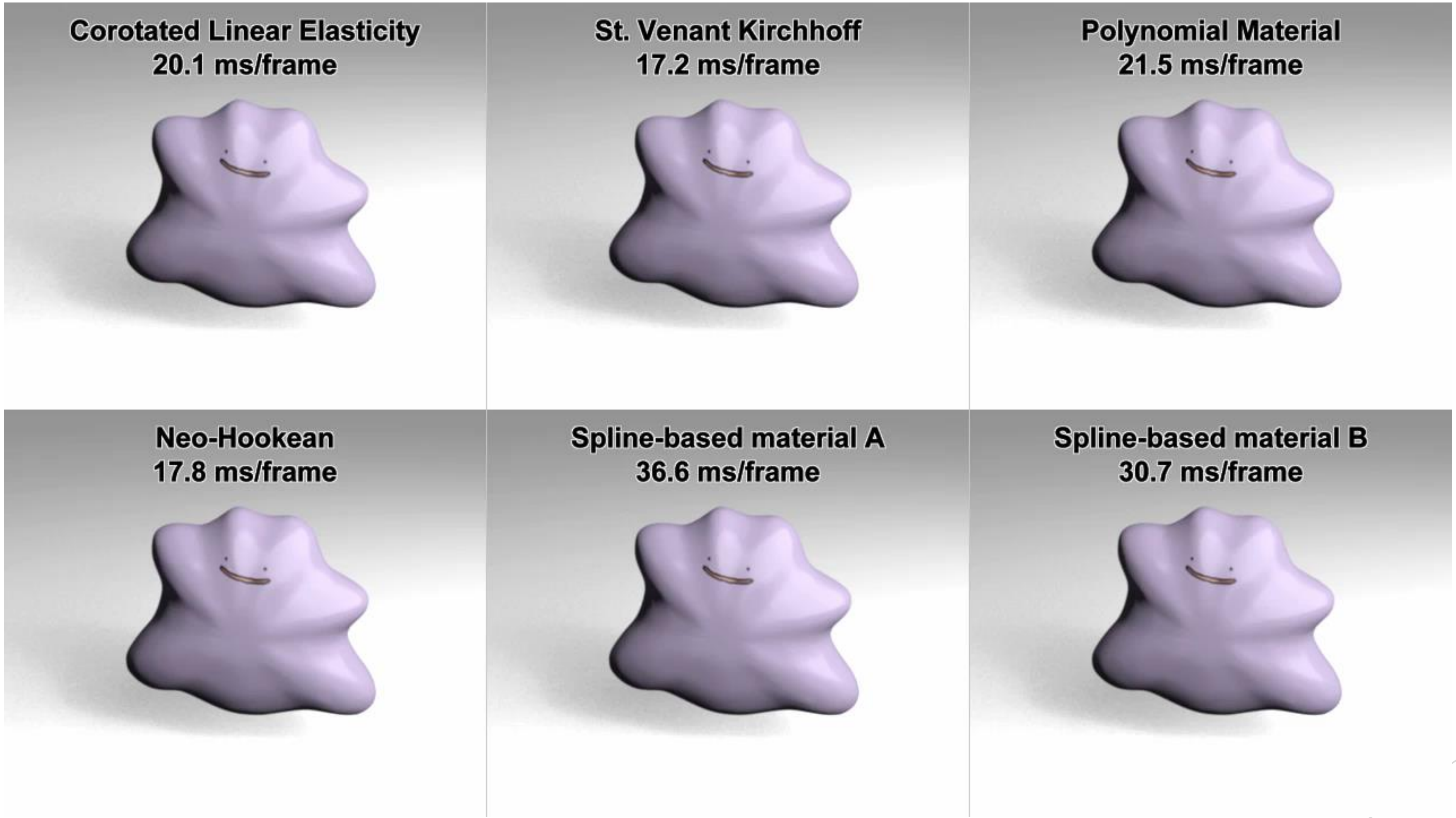


Supporting More General Materials

- $\boldsymbol{M}/h^2 + \sum_j \boxed{w_j} \boldsymbol{G}_j \boldsymbol{G}_j^T$



Supporting More General Materials



Quasi-Newton Algorithm

Algorithm 3: Quasi-Newton Solver with Backtracking Line Search.

$\mathbf{x}^{(1)} := \mathbf{y}$;

$g(\mathbf{x}^{(1)}) := \texttt{evalObjective}(\mathbf{x}^{(1)})$

for $k = 1, \ldots, \text{numIterations}$ **do**

  $\nabla g(\mathbf{x}^{(k)}) := \texttt{evalGradient}(\mathbf{x}^{(k)})$

  $\delta\mathbf{x} := -(\mathbf{M}/h^2 + \mathbf{L})^{-1} \nabla g(\mathbf{x}^{(k)})$

  $\alpha := 1/\beta$

  repeat

    $\alpha := \beta\alpha$

    $\mathbf{x}^{(k+1)} := \mathbf{x}^{(k)} + \alpha\delta\mathbf{x}$

    $g(\mathbf{x}^{(k+1)}) := \texttt{evalObjective}(\mathbf{x}^{(k+1)})$

  until $g(\mathbf{x}^{(k+1)}) \leq g(\mathbf{x}^{(k)}) + \gamma\alpha \operatorname{tr}((\nabla g(\mathbf{x}^{(k)}))^\mathsf{T} \delta\mathbf{x})$;

end

Compute Gradient

Quasi-Newton Algorithm

Algorithm 3: Quasi-Newton Solver with Backtracking Line Search.

$\mathbf{x}^{(1)} := \mathbf{y};$

$g(\mathbf{x}^{(1)}) := \texttt{evalObjective}(\mathbf{x}^{(1)})$

for $k = 1, \ldots, \text{numIterations}$ **do**

$\quad \nabla g(\mathbf{x}^{(k)}) := \texttt{evalGradient}(\mathbf{x}^{(k)})$

$\quad \delta\mathbf{x} := -(\mathbf{M}/h^2 + \mathbf{L})^{-1} \nabla g(\mathbf{x}^{(k)})$

$\quad \alpha := 1/\beta$

$\quad$ **repeat**

$\quad\quad \alpha := \beta\alpha$

$\quad\quad \mathbf{x}^{(k+1)} := \mathbf{x}^{(k)} + \alpha\delta\mathbf{x}$

$\quad\quad g(\mathbf{x}^{(k+1)}) := \texttt{evalObjective}(\mathbf{x}^{(k+1)})$

$\quad$ **until** $g(\mathbf{x}^{(k+1)}) \leq g(\mathbf{x}^{(k)}) + \gamma\alpha \operatorname{tr}((\nabla g(\mathbf{x}^{(k)}))^{\mathsf{T}} \delta\mathbf{x});$

end

Evaluate Descent Direction

Quasi-Newton Algorithm

Algorithm 3: Quasi-Newton Solver with Backtracking Line Search.

$\mathbf{x}^{(1)} := \mathbf{y};$

$g(\mathbf{x}^{(1)}) := \texttt{evalObjective}(\mathbf{x}^{(1)})$

for $k = 1, \ldots,$ numIterations **do**

$\quad \nabla g(\mathbf{x}^{(k)}) := \texttt{evalGradient}(\mathbf{x}^{(k)})$

$\quad \delta\mathbf{x} := -(\mathbf{M}/h^2 + \mathbf{L})^{-1} \nabla g(\mathbf{x}^{(k)})$

$\quad \alpha := 1/\beta$

$\quad$ **repeat**

$\quad\quad \alpha := \beta\alpha$

$\quad\quad \mathbf{x}^{(k+1)} := \mathbf{x}^{(k)} + \alpha\delta\mathbf{x}$

$\quad\quad g(\mathbf{x}^{(k+1)}) := \texttt{evalObjective}(\mathbf{x}^{(k+1)})$

$\quad$ **until** $g(\mathbf{x}^{(k+1)}) \leq g(\mathbf{x}^{(k)}) + \gamma\alpha\, \mathrm{tr}((\nabla g(\mathbf{x}^{(k)}))^\mathsf{T} \delta\mathbf{x});$

end

Line Search

Quasi-Newton Algorithm

Algorithm 3: Quasi-Newton Solver with Backtracking Line Search.

$\mathbf{x}^{(1)} := \mathbf{y};$

$g(\mathbf{x}^{(1)}) := \texttt{evalObjective}(\mathbf{x}^{(1)})$

for $k = 1, \ldots, \mathrm{numIterations}$ **do**

$\nabla g(\mathbf{x}^{(k)}) := \texttt{evalGradient}(\mathbf{x}^{(k)})$

$\delta\mathbf{x} := -(\mathbf{M}/h^2 + \mathbf{L})^{-1} \nabla g(\mathbf{x}^{(k)})$

$\alpha := 1/\beta$

repeat

$\alpha := \beta\alpha$

$\mathbf{x}^{(k+1)} := \mathbf{x}^{(k)} + \alpha\delta\mathbf{x}$

$g(\mathbf{x}^{(k+1)}) := \texttt{evalObjective}(\mathbf{x}^{(k+1)})$

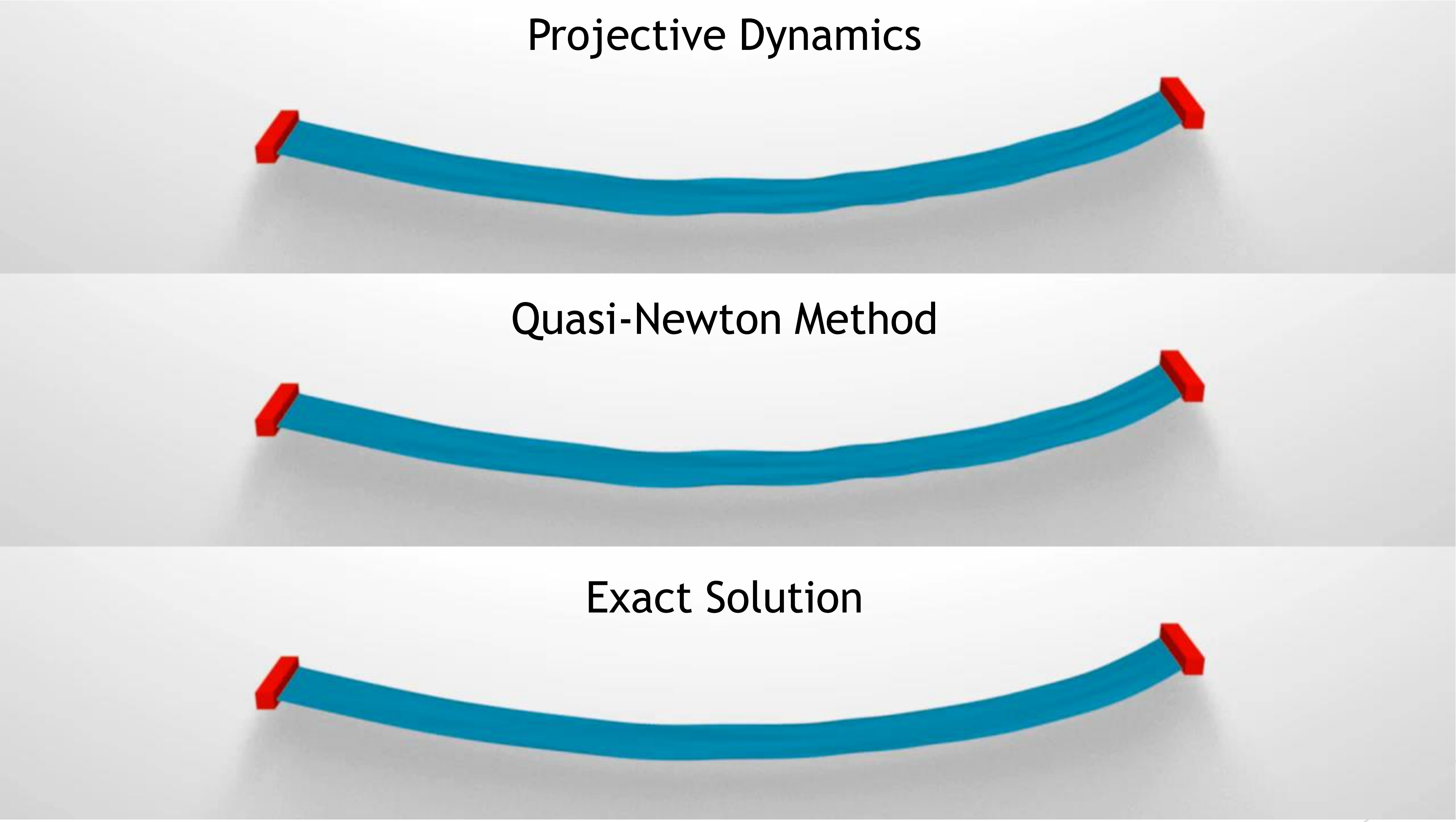
until $g(\mathbf{x}^{(k+1)}) \leq g(\mathbf{x}^{(k)}) + \gamma\alpha\, \mathrm{tr}((\nabla g(\mathbf{x}^{(k)}))^{\mathsf{T}} \delta\mathbf{x});$

end

We can do more

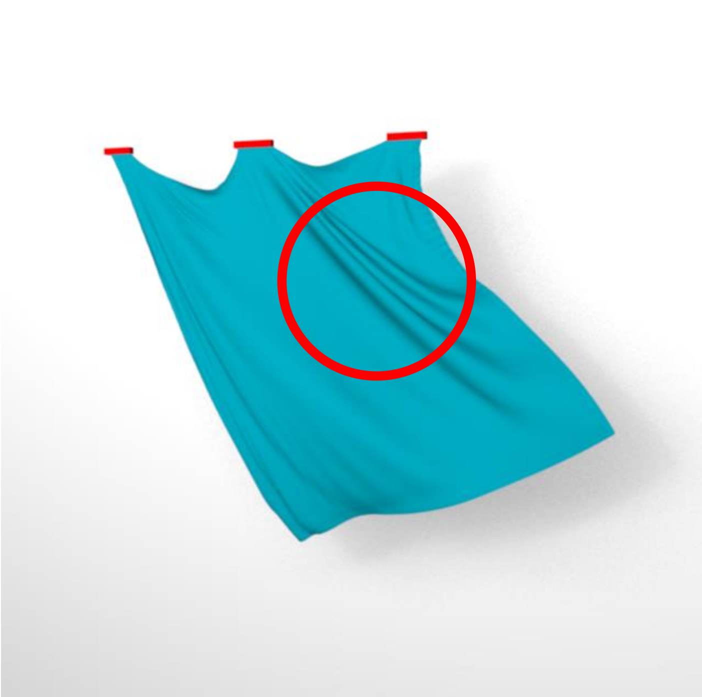
Broyden, Fletcher, Goldfarb, Shanno

L-BFGS Acceleration

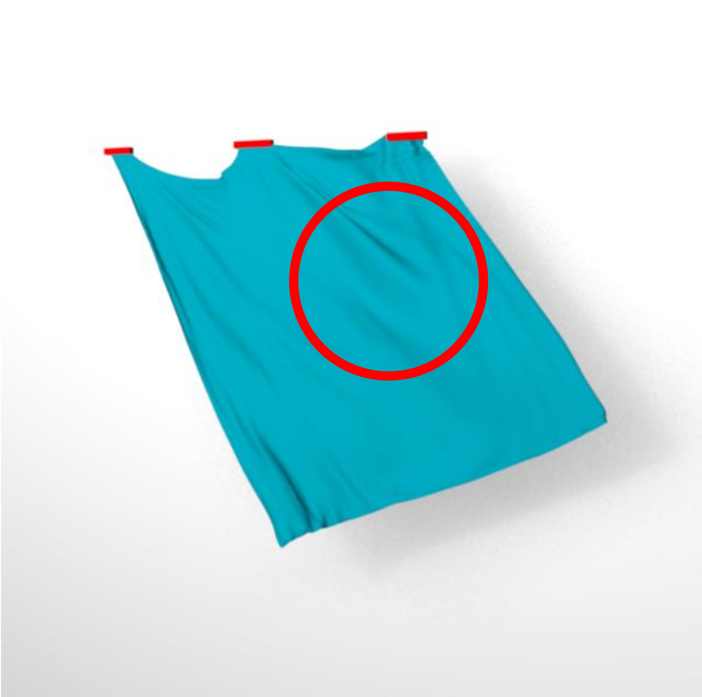


L-BFGS Acceleration

Quasi-Newton Method



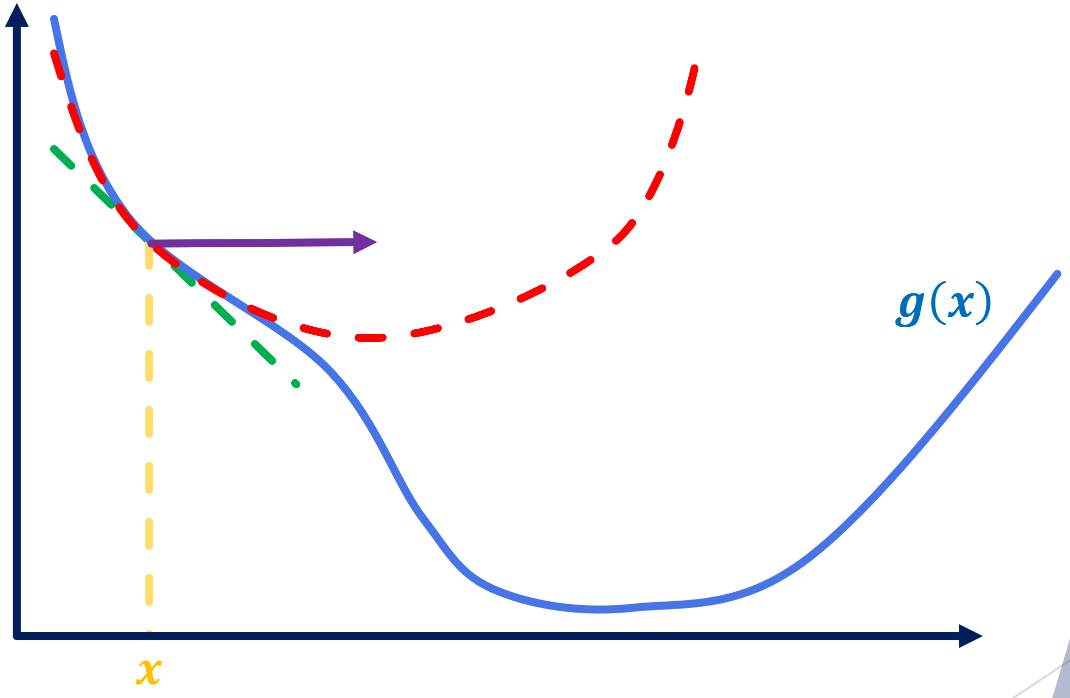
Projective Dynamics



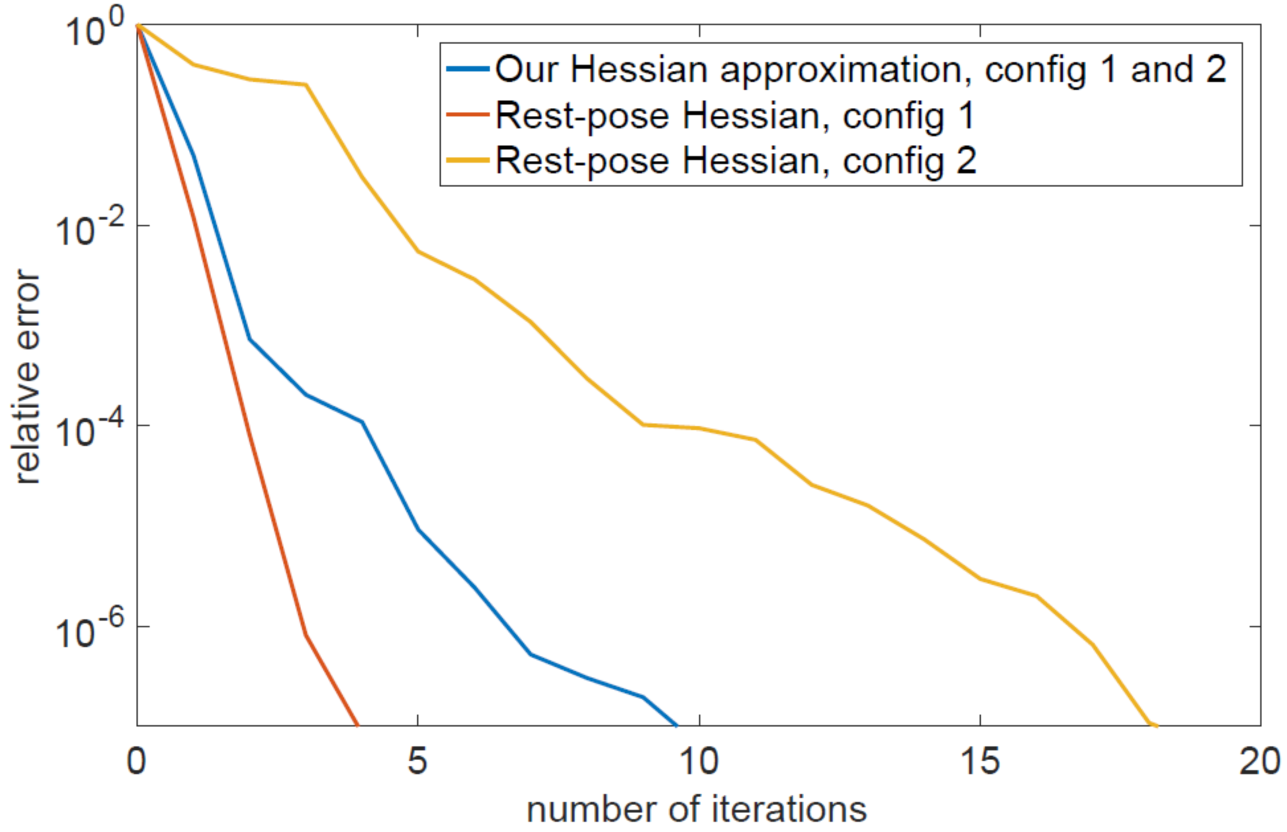
Core of Quasi-Newton Methods

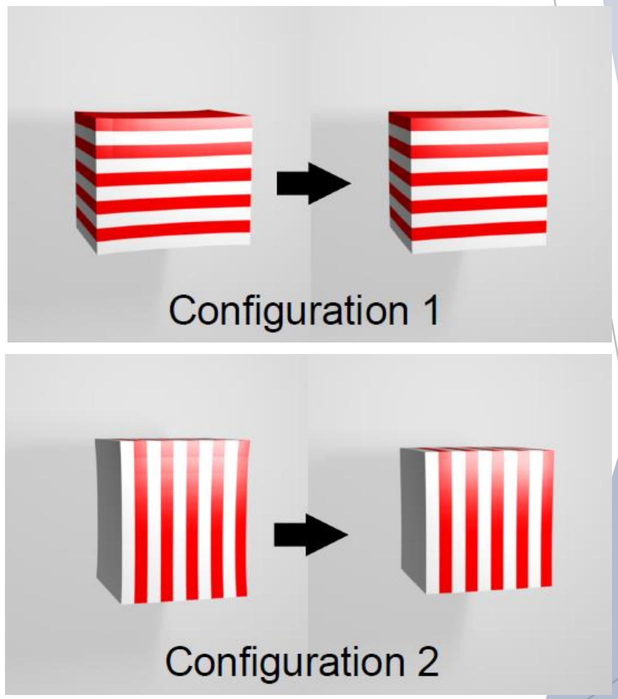
$$\Delta x = -[\quad A \quad]^{-1} \nabla g(x)$$

$$\frac{M}{h^2} + L$$

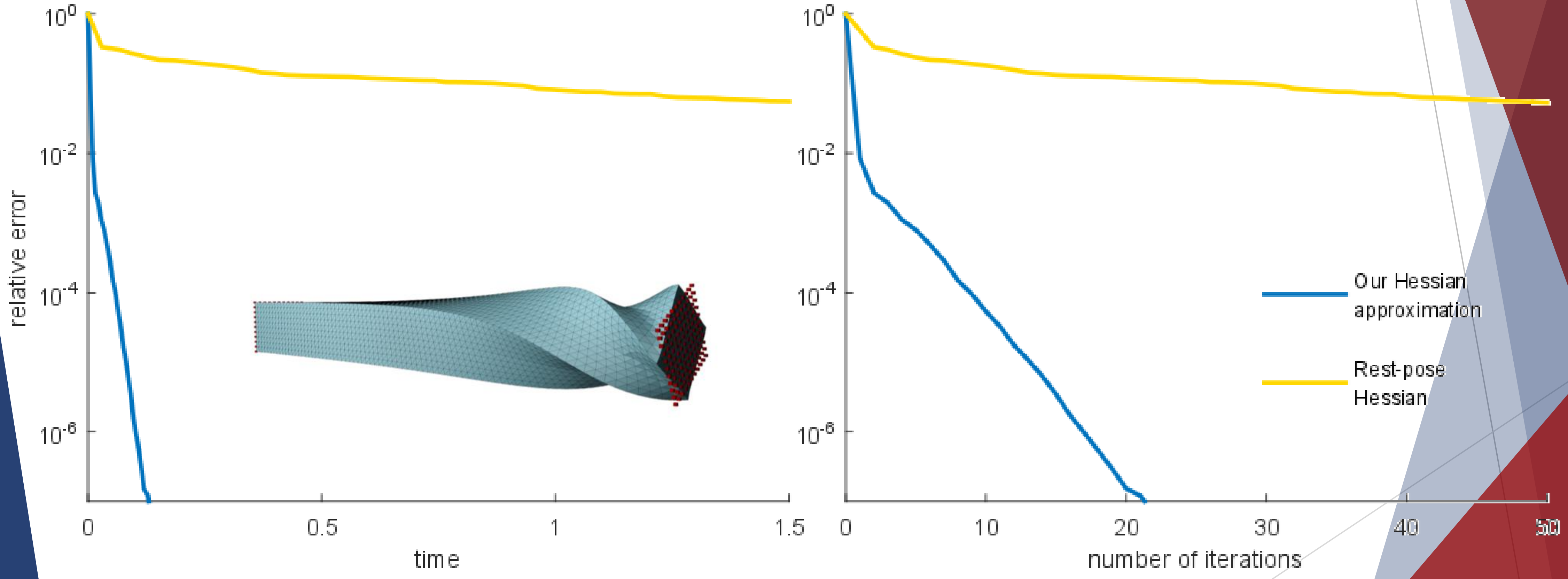


L-BFGS with rest-pose Hessian

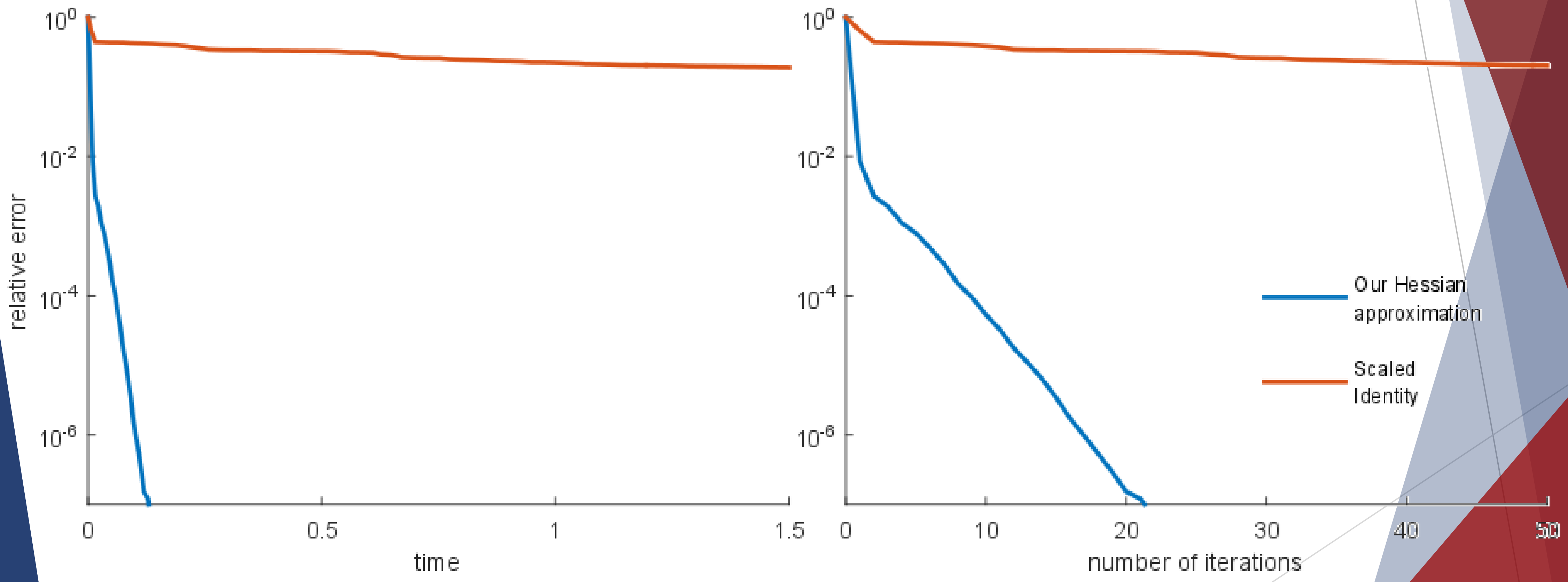




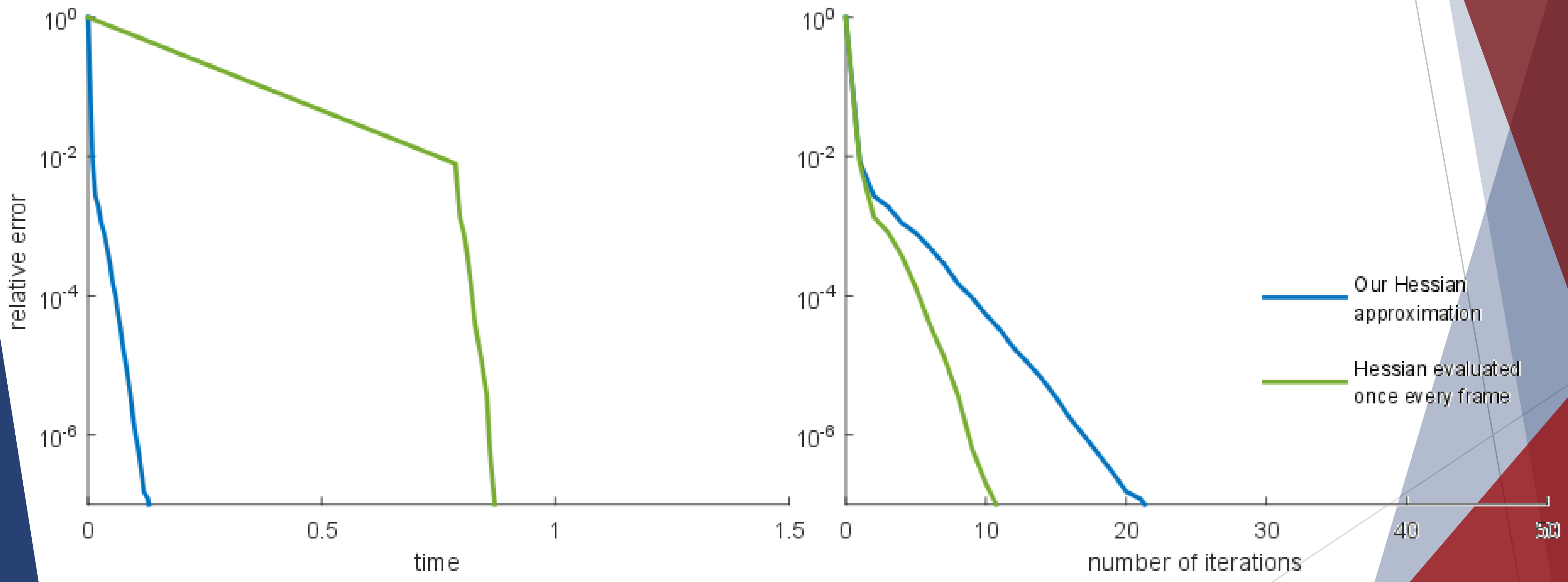
L-BFGS with rest-pose Hessian



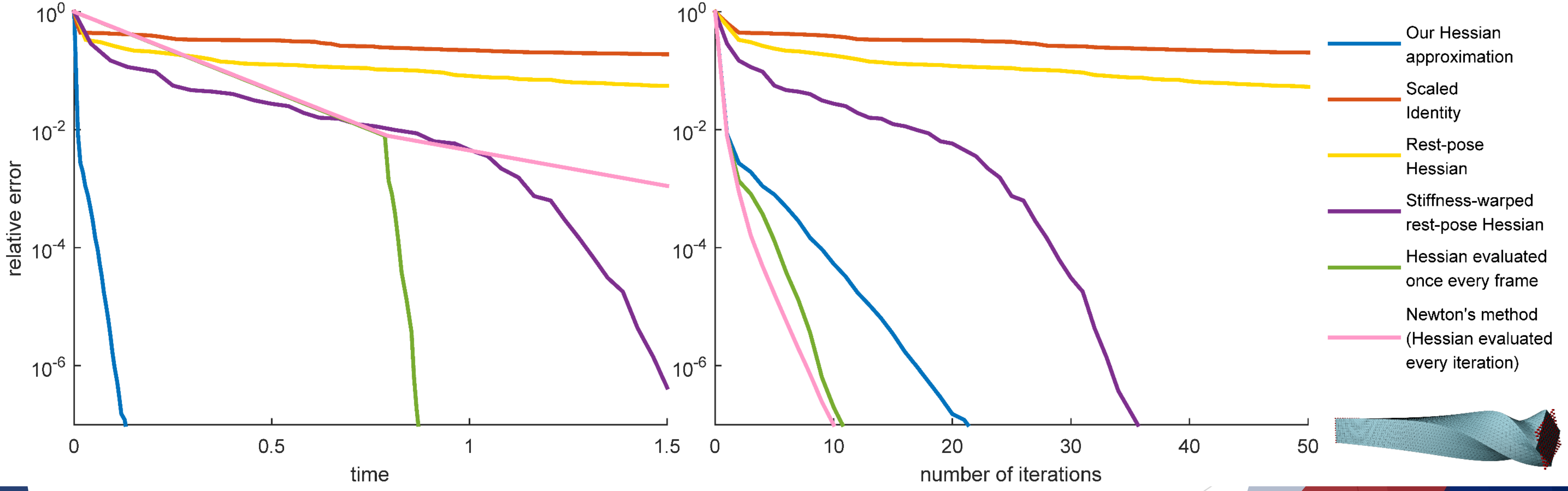
L-BFGS with Scaled Identity



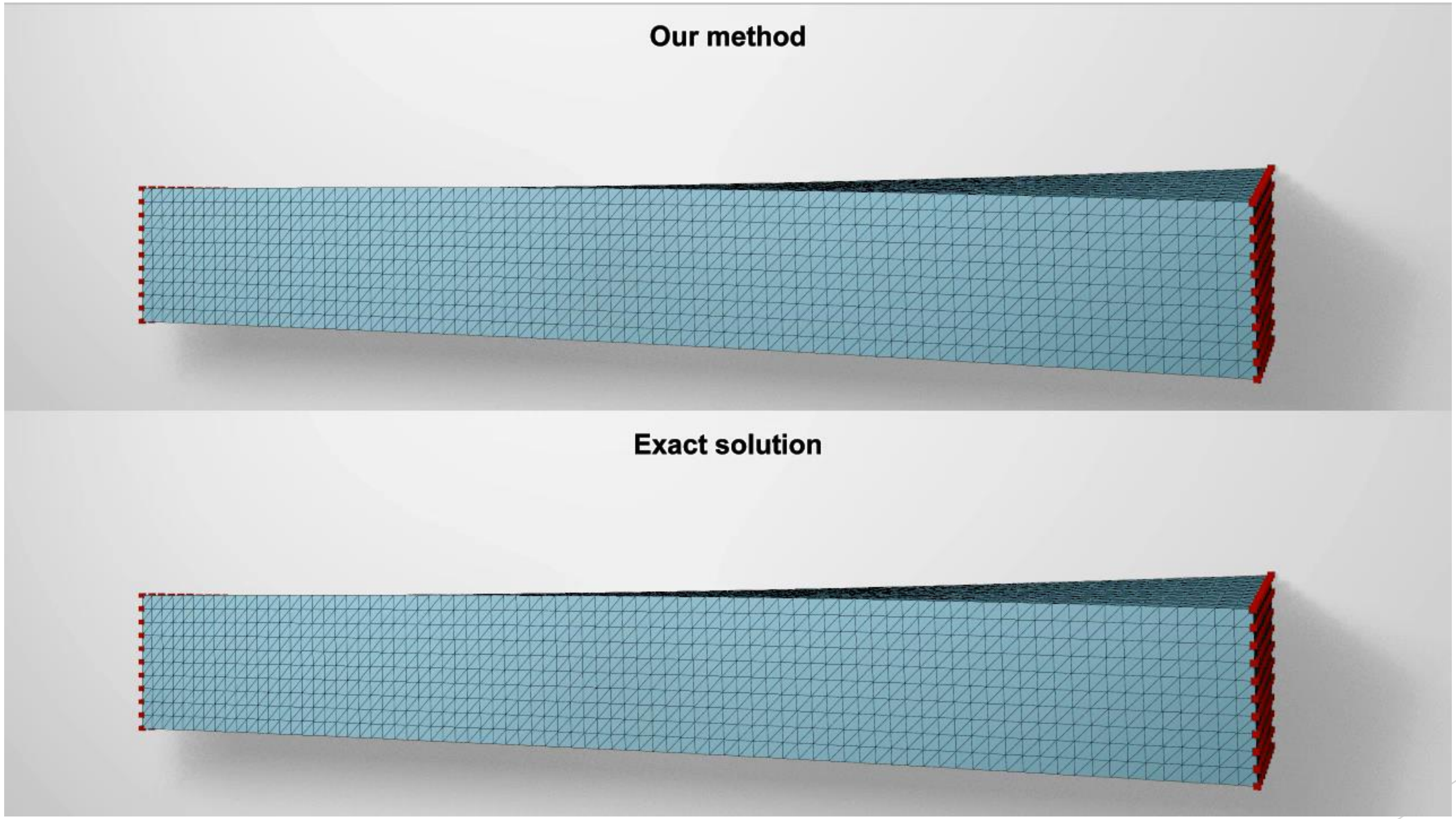
L-BFGS with updating Hessian



Performance of L-BFGS family

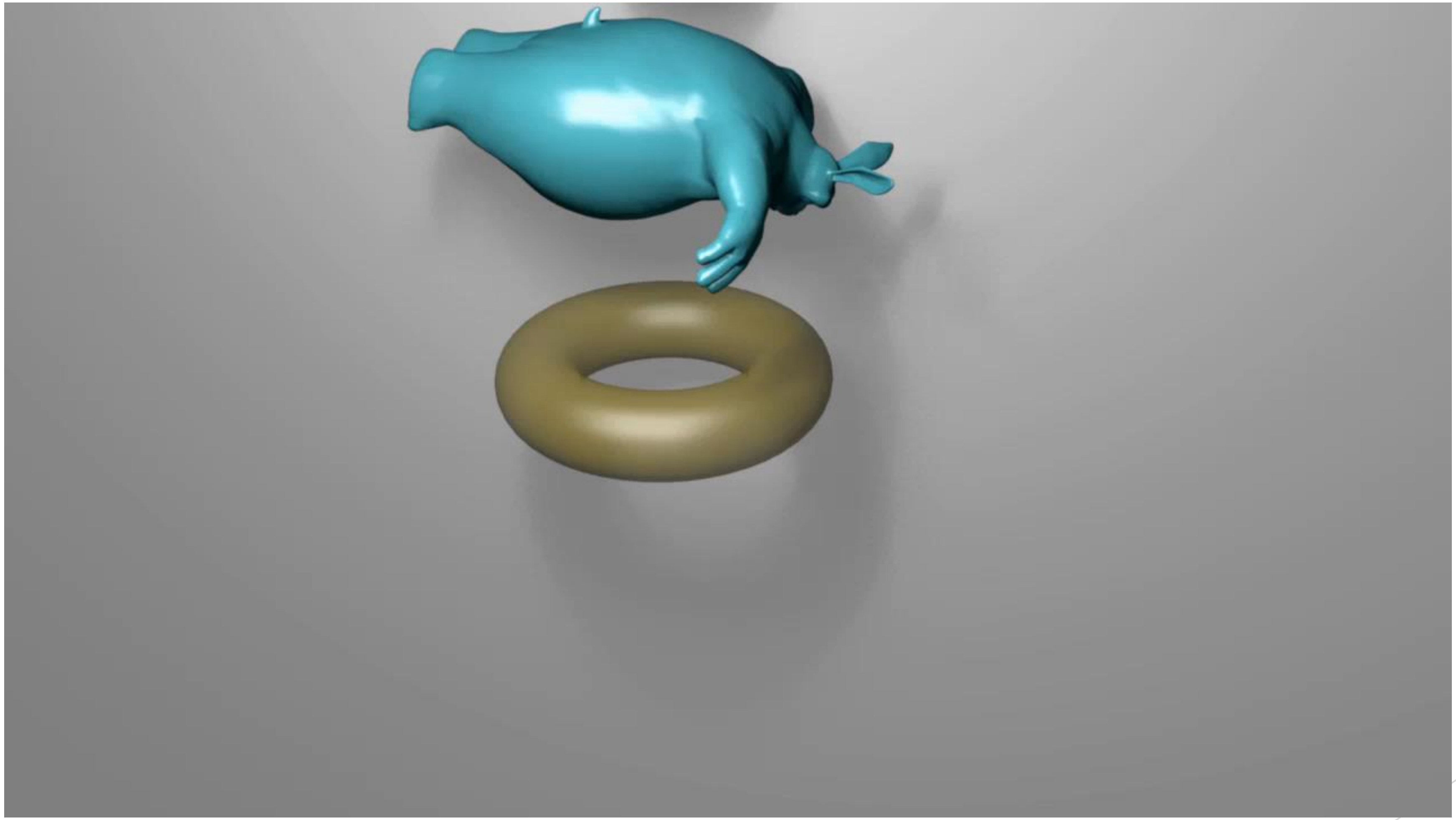


Results: Accuracy

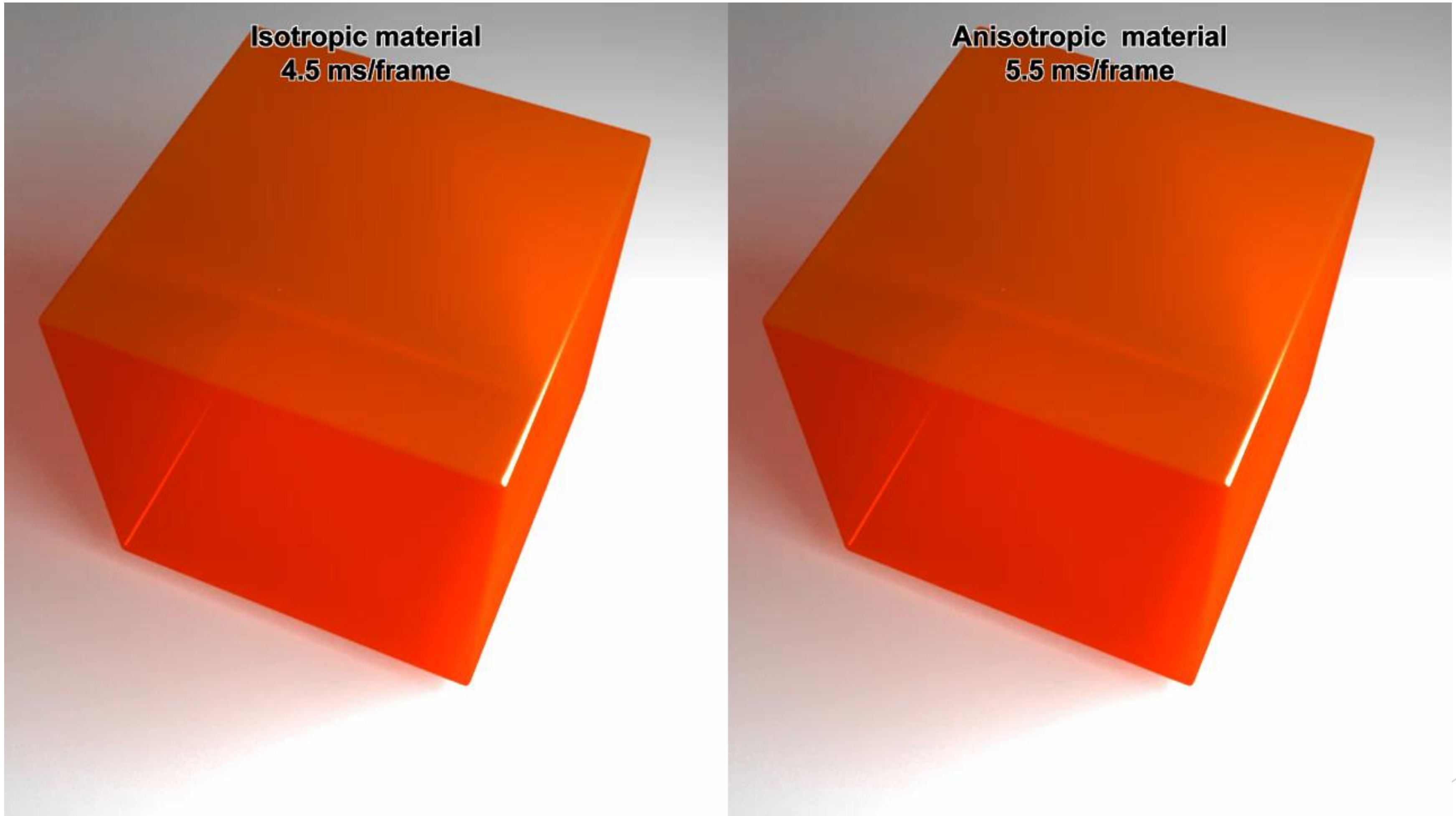


Results: Robustness

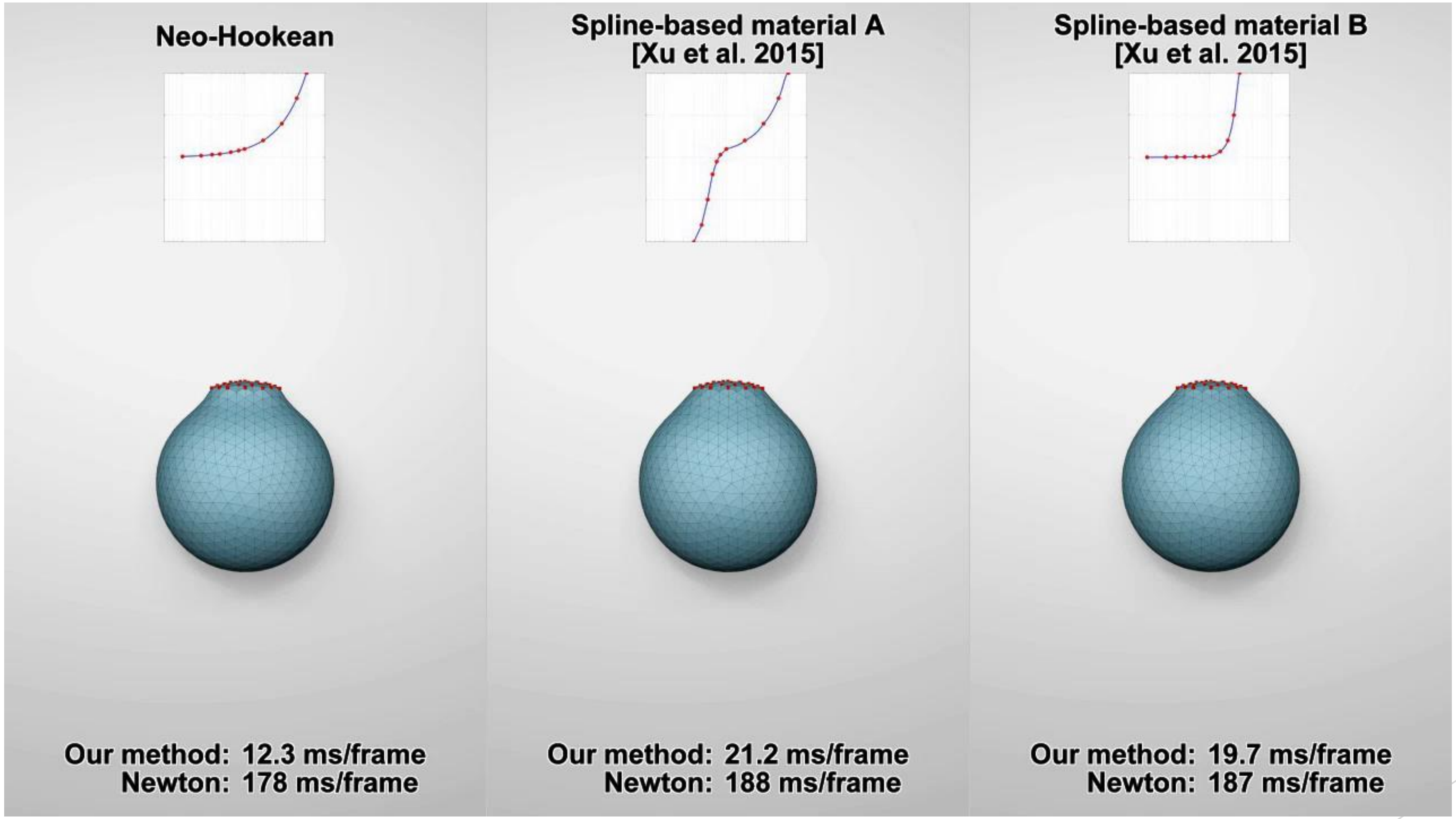
Results: Collision



Results: Anisotropy

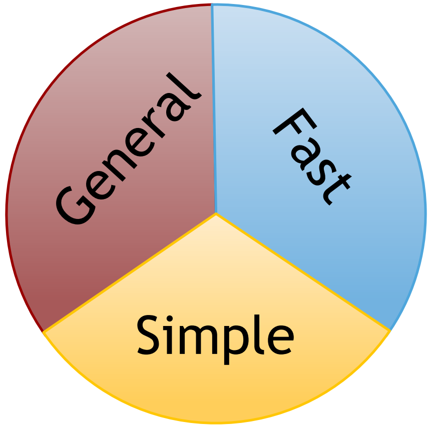


Results: Spline-Based Materials



Remark

- Our method is:
  - General: supports a variety types of hyperelastic materials
  - Fast: >10x faster compared to Newton’s method to achieve similar accuracy level
  - Simple: avoids Hessian computation, avoids definiteness fix



Towards Real-time Simulation of Deformable Objects:

Generalization of Spatial Discretization Models

Fast Mass Spring System → Projective Dynamics

Towards Real-time Simulation of Deformable Objects:

Generalization of Material Models + Acceleration

Towards Real-time Simulation of Deformable Objects:

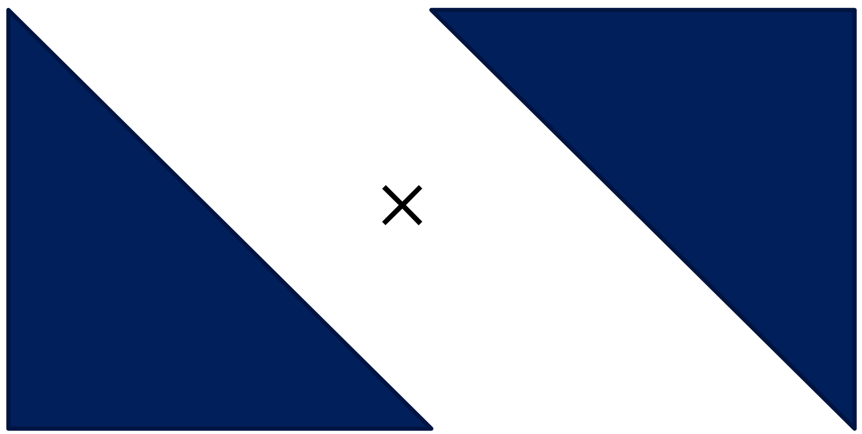
What's Next?

Quasi-Newton Methods → ?

Core of Our Methods

$$\Delta \boldsymbol{x} = -\left[\frac{\boldsymbol{M}}{\boldsymbol{h}^2} + \boldsymbol{L}\right]^{-1} \nabla \mathrm{g}(\mathbf{x})$$

$$\frac{\boldsymbol{M}}{\boldsymbol{h}^2} + \boldsymbol{L} =$$

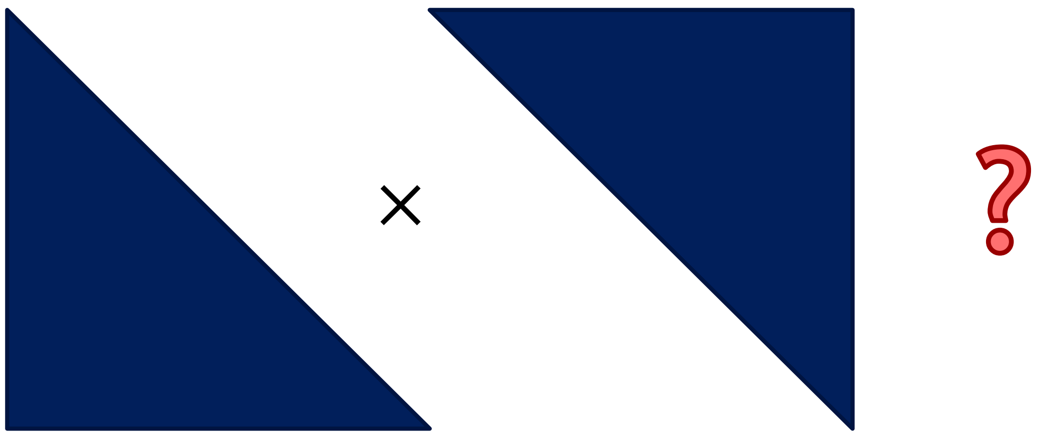


Core of Our Methods

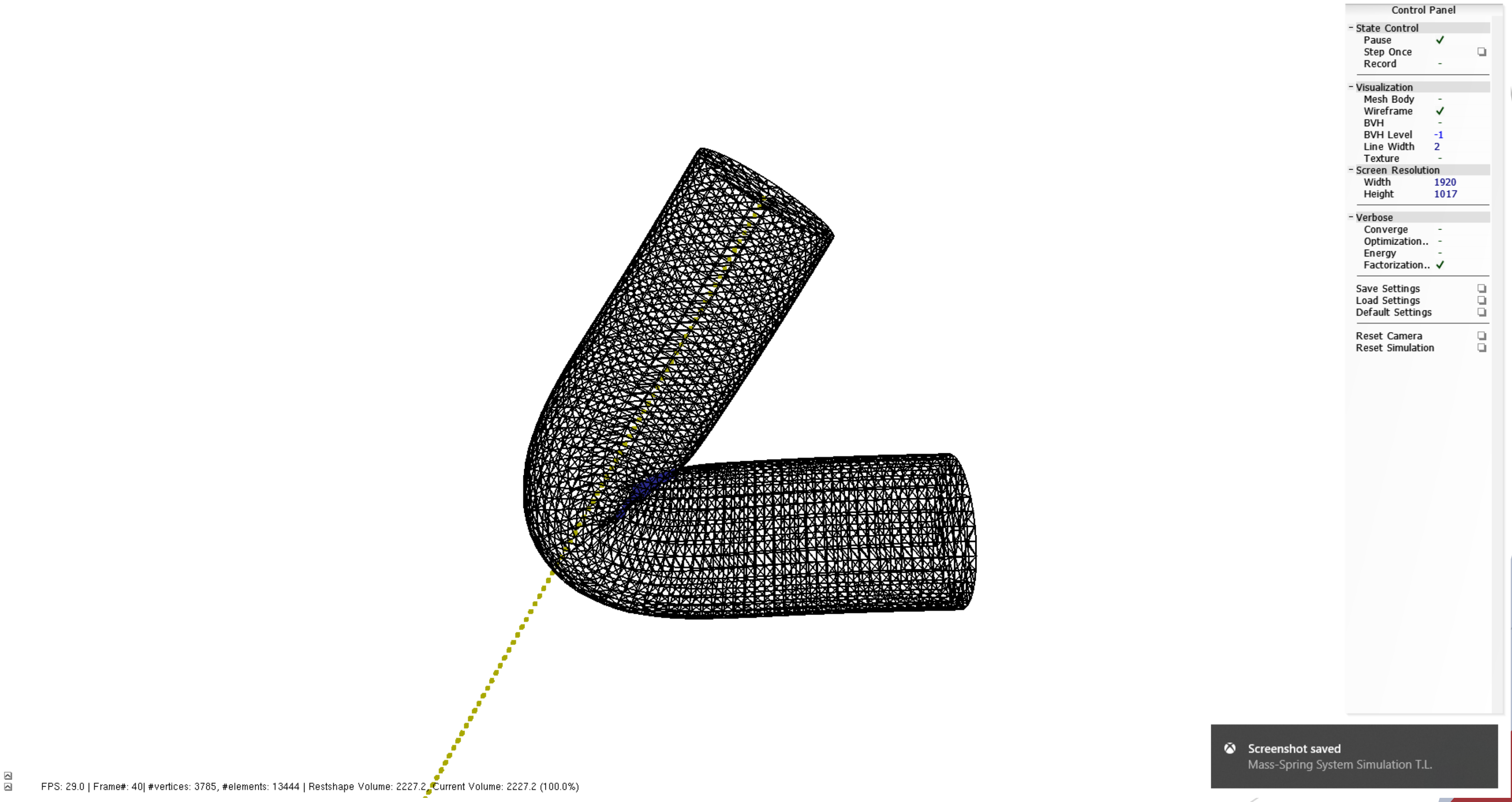
$$\Delta \boldsymbol{x} = -\left[\frac{\boldsymbol{M}}{\boldsymbol{h}^2} + \boldsymbol{L}\right]^{-1} \nabla \mathrm{g}(\mathbf{x})$$

Time Varying Events

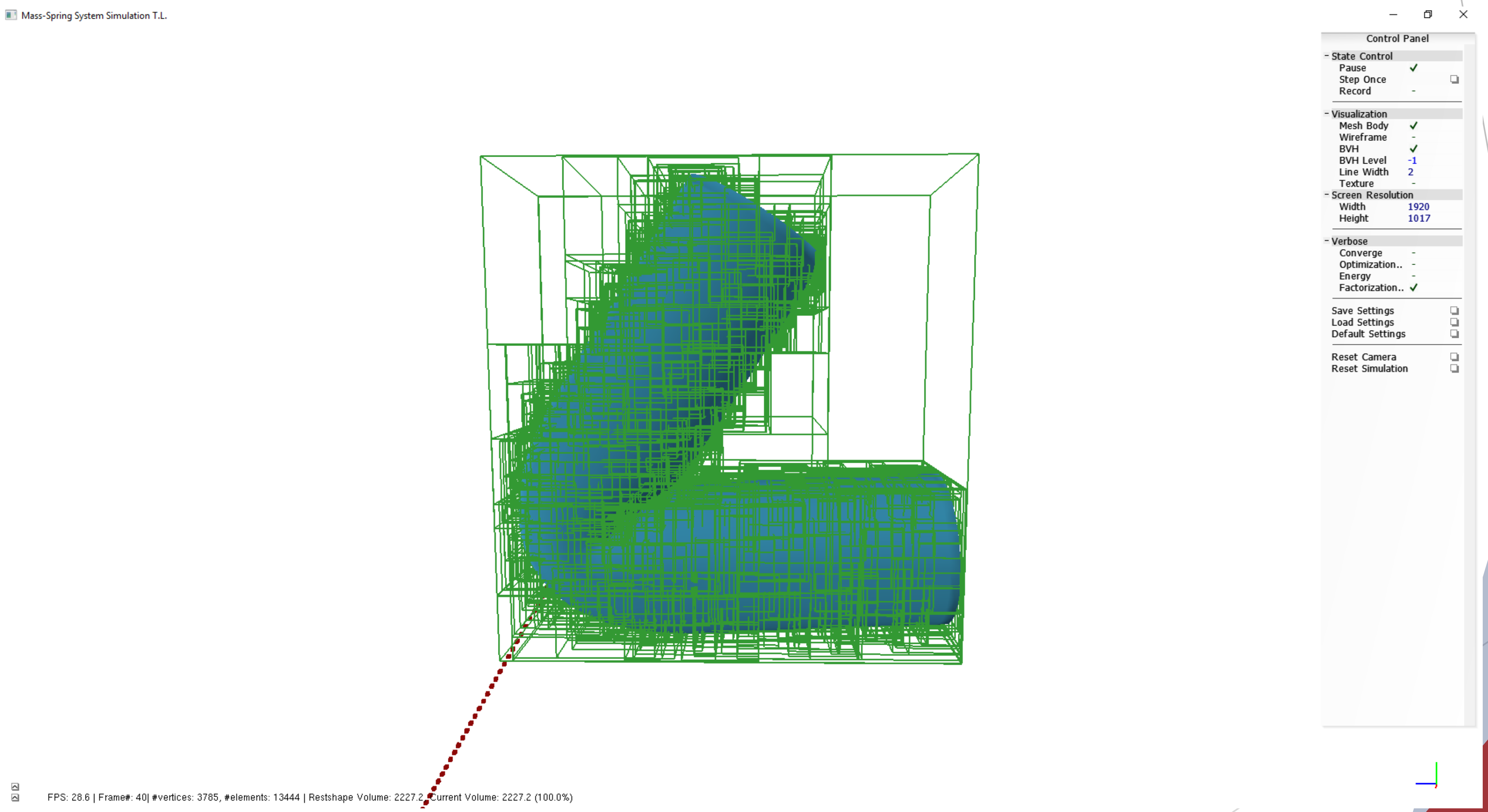
- Collisions
- Tearing or Cutting



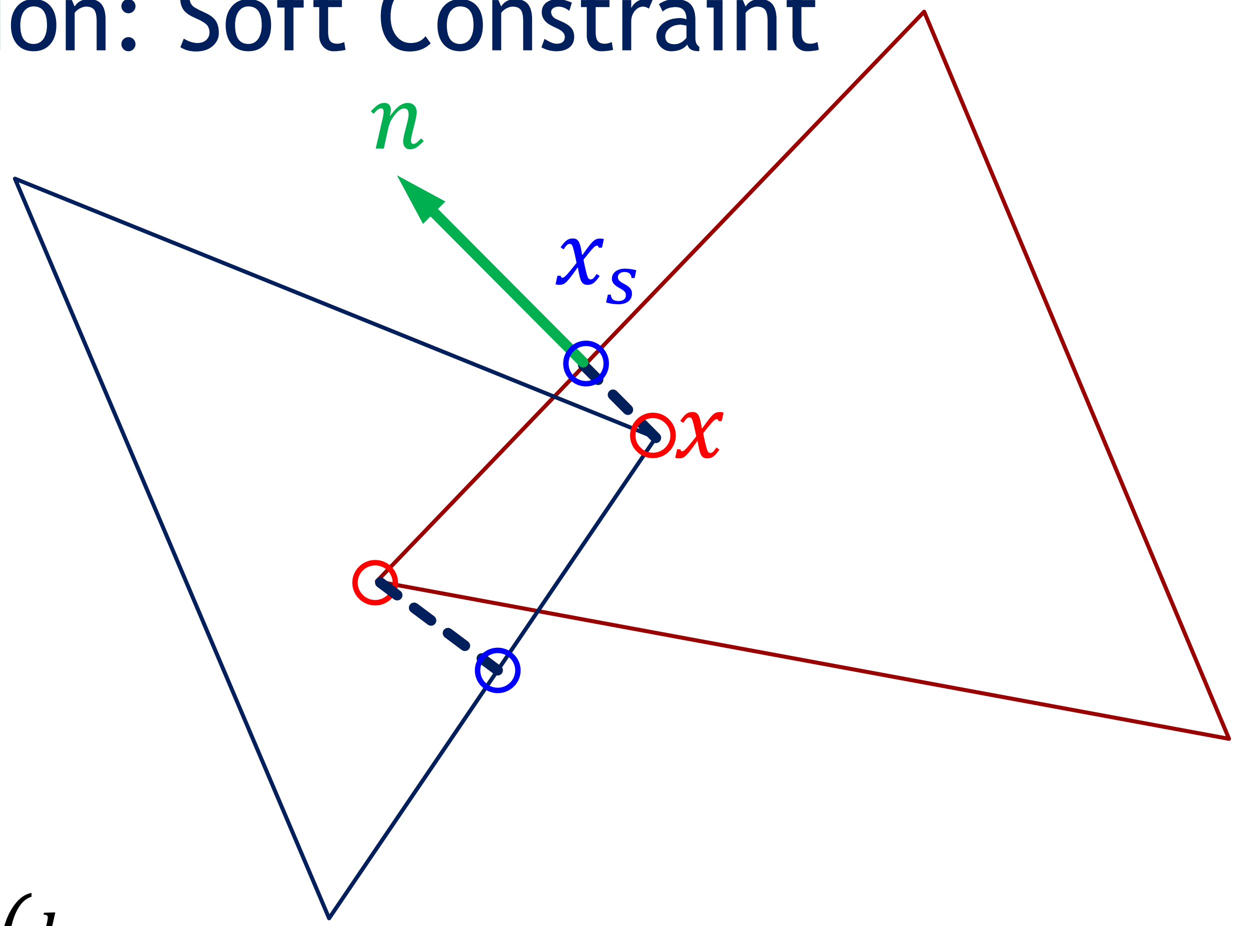
Collisions


Control Panel
State Control
Pause
Step Once
Record
Visualization
Mesh Body
Wireframe
BVH
BVH Level -1
Line Width 2
Texture
Screen Resolution
Width 1920
Height 1017
Verbose
Converge
Optimization..
Energy
Factorization..
Save Settings
Load Settings
Default Settings
Reset Camera
Reset Simulation
Screenshot saved
Mass-Spring System Simulation T.L.
FPS: 29.0 | Frame#: 40| #vertices: 3785, #elements: 13444 | Restshape Volume: 2227.2, Current Volume: 2227.2 (100.0%)

Collisions


Mass-Spring System Simulation T.L.
Control Panel
State Control
Pause
Step Once
Record
Visualization
Mesh Body
Wireframe
BVH
BVH Level -1
Line Width 2
Texture
Screen Resolution
Width 1920
Height 1017
Verbose
Converge
Optimization..
Energy
Factorization..
Save Settings
Load Settings
Default Settings
Reset Camera
Reset Simulation
FPS: 28.6 | Frame#: 40| #vertices: 3785, #elements: 13444 | Restshape Volume: 2227.2 Current Volume: 2227.2 (100.0%)

Collision: Soft Constraint



$$E_{col} = \begin{cases} \frac{k_{col}}{2}\left((x - x_s)^T n\right)^2 & , if\,(x - x_s)^T n < 0 \\ 0 & , otherwise \end{cases}$$

Collision: Soft Constraint

$$E_{col} = \begin{cases} \frac{k_{col}}{2}\left((x - x_s)^T n\right)^2 & , if\,(x - x_s)^T n < 0 \\ 0 & , otherwise \end{cases}$$

$$\nabla E_{col} = \begin{cases} k_{col}\left((x - x_s)^T n\right) n & , if\,(x - x_s)^T n < 0 \\ 0 & , otherwise \end{cases}$$

$$\nabla^2 E_{col} = \begin{cases} k_{col} n n^T & , if\,(x - x_s)^T n < 0 \\ 0 & , otherwise \end{cases}$$

Quasi-Newton Algorithm with Collisions

Algorithm 3: Quasi-Newton Solver with Backtracking Line Search.

$\mathbf{x}^{(1)} := \mathbf{y};$

$g(\mathbf{x}^{(1)}) :=$ `evalObjective`$(\mathbf{x}^{(1)})$

for $k = 1, \ldots,$ numIterations **do**

  $\nabla g(\mathbf{x}^{(k)}) :=$ `evalGradient`$(\mathbf{x}^{(k)})$ ∇E_{col}

  $\delta\mathbf{x} := -(\mathbf{M}/h^2 + \mathbf{L})^{-1}\nabla g(\mathbf{x}^{(k)})$

  $\alpha := 1/\beta$ 0

  repeat

    $\alpha := \beta\alpha$

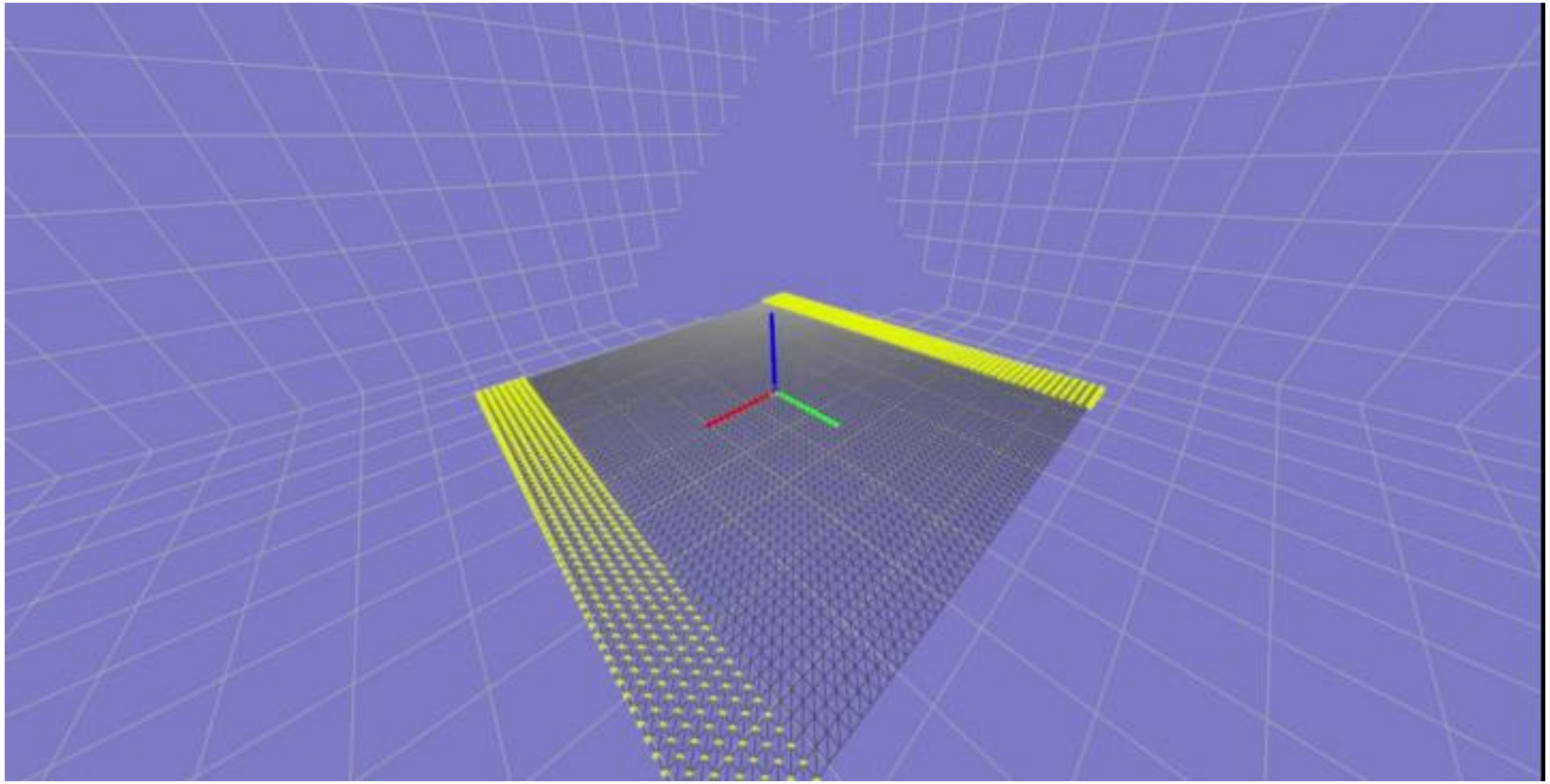
    $\mathbf{x}^{(k+1)} := \mathbf{x}^{(k)} + \alpha\delta\mathbf{x}$

    $g(\mathbf{x}^{(k+1)}) :=$ `evalObjective`$(\mathbf{x}^{(k+1)})$ E_{col}

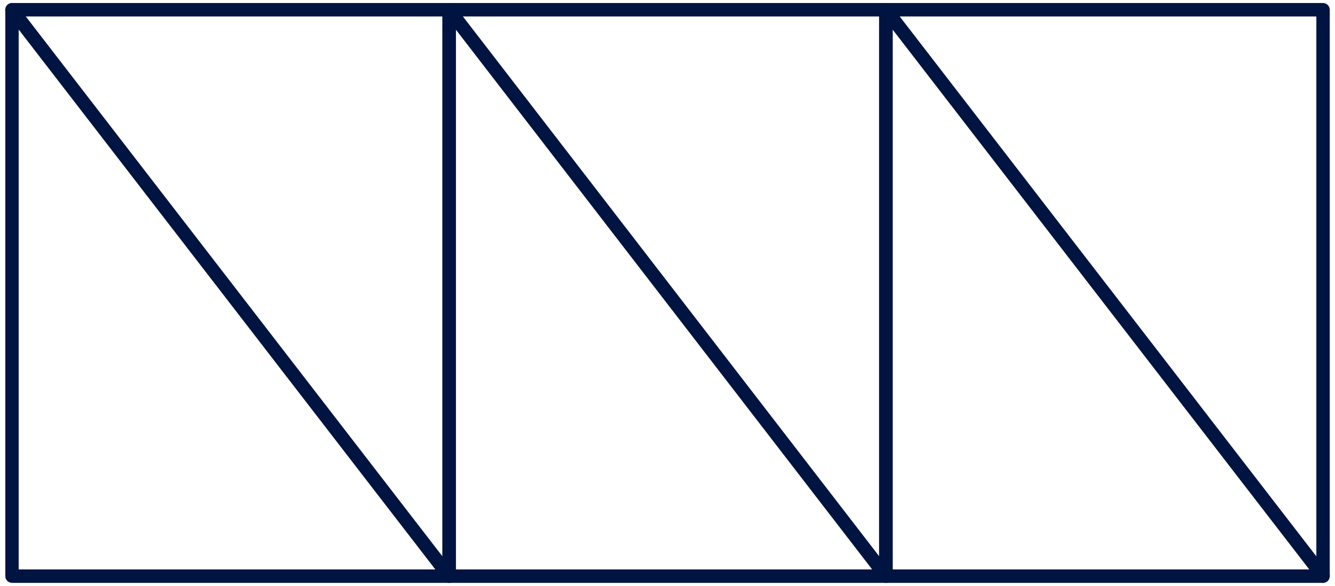
  until $g(\mathbf{x}^{(k+1)}) \leq g(\mathbf{x}^{(k)}) + \gamma\alpha\,\mathrm{tr}((\nabla g(\mathbf{x}^{(k)}))^{\mathsf{T}}\delta\mathbf{x});$

end

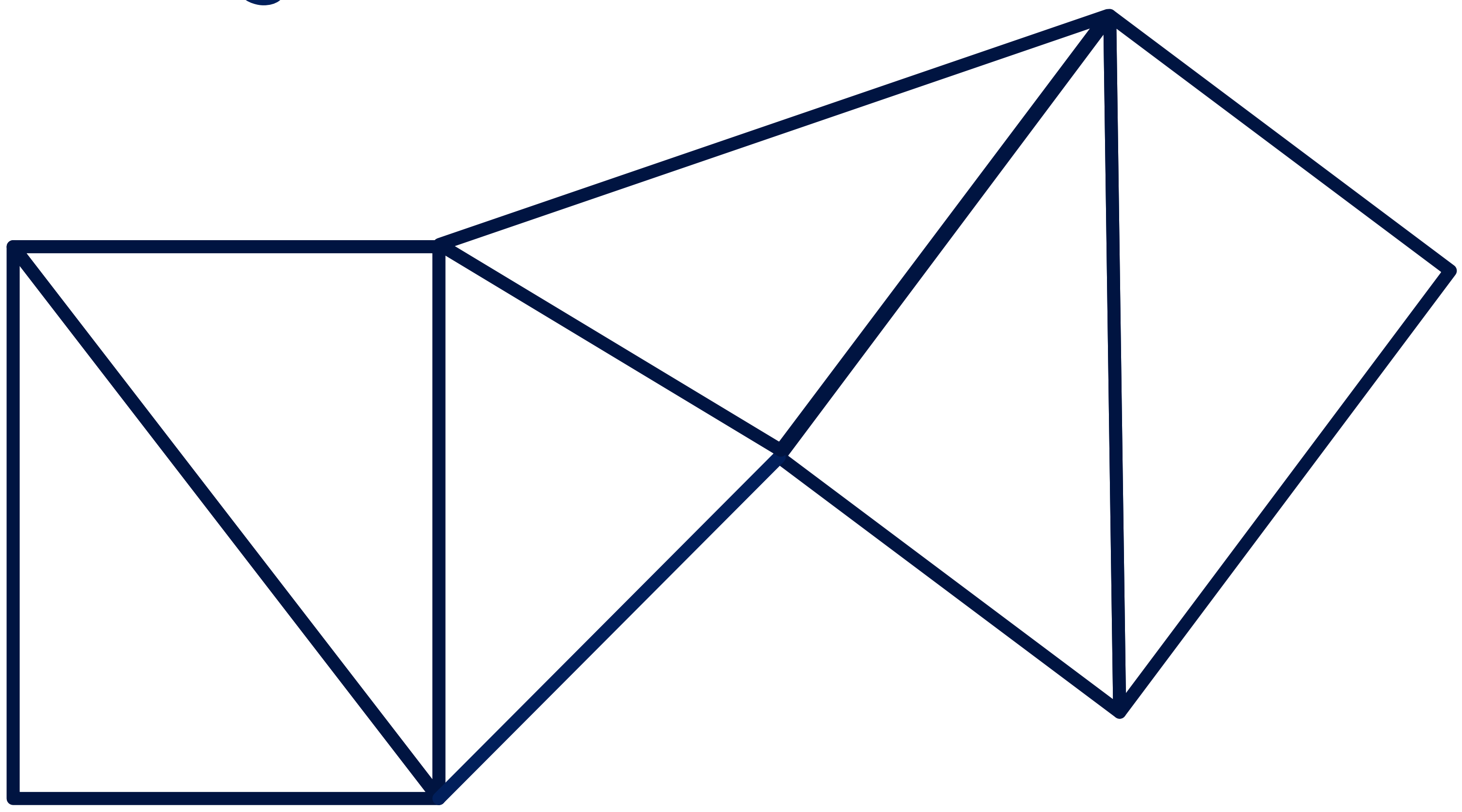
Tearing



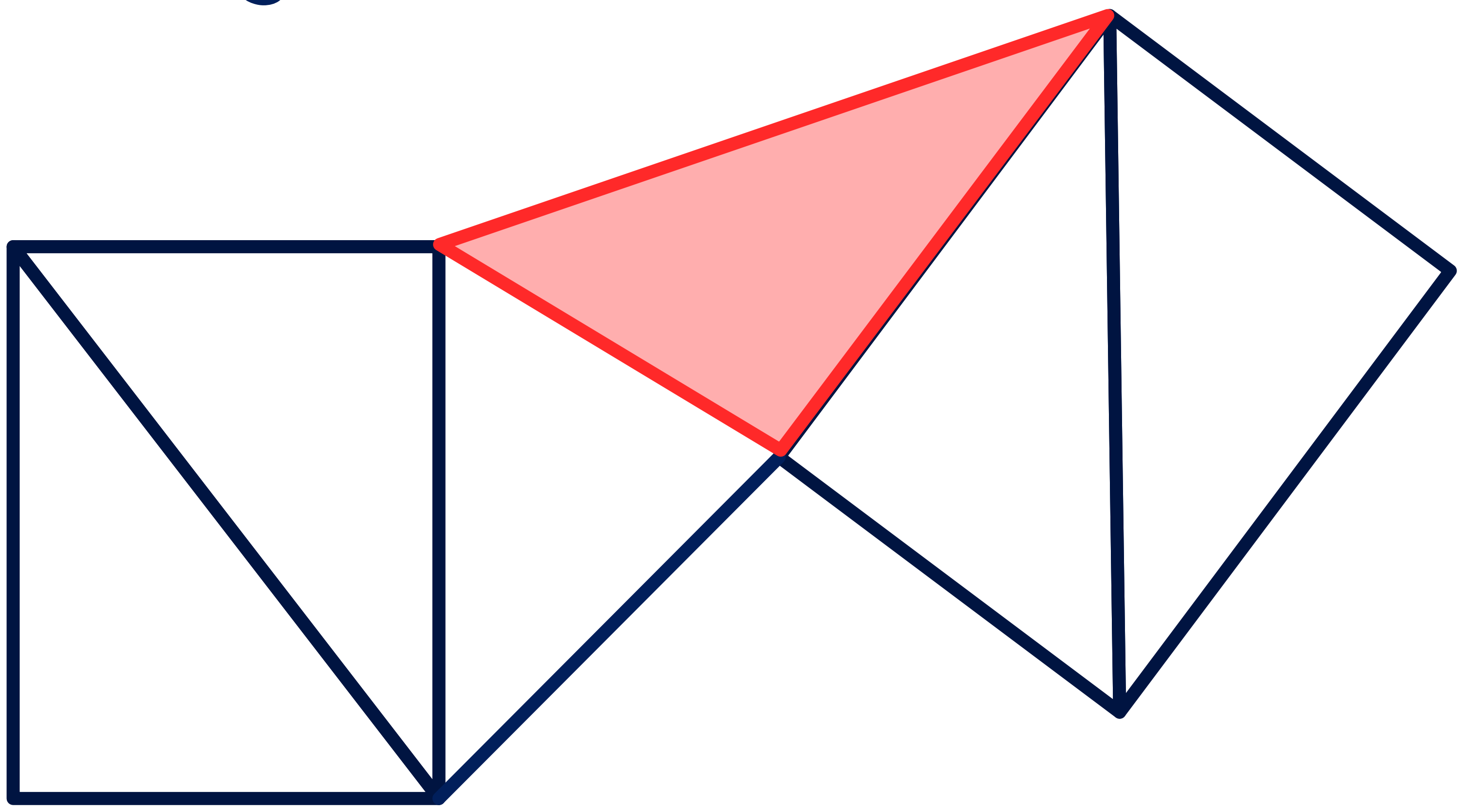
Tearing



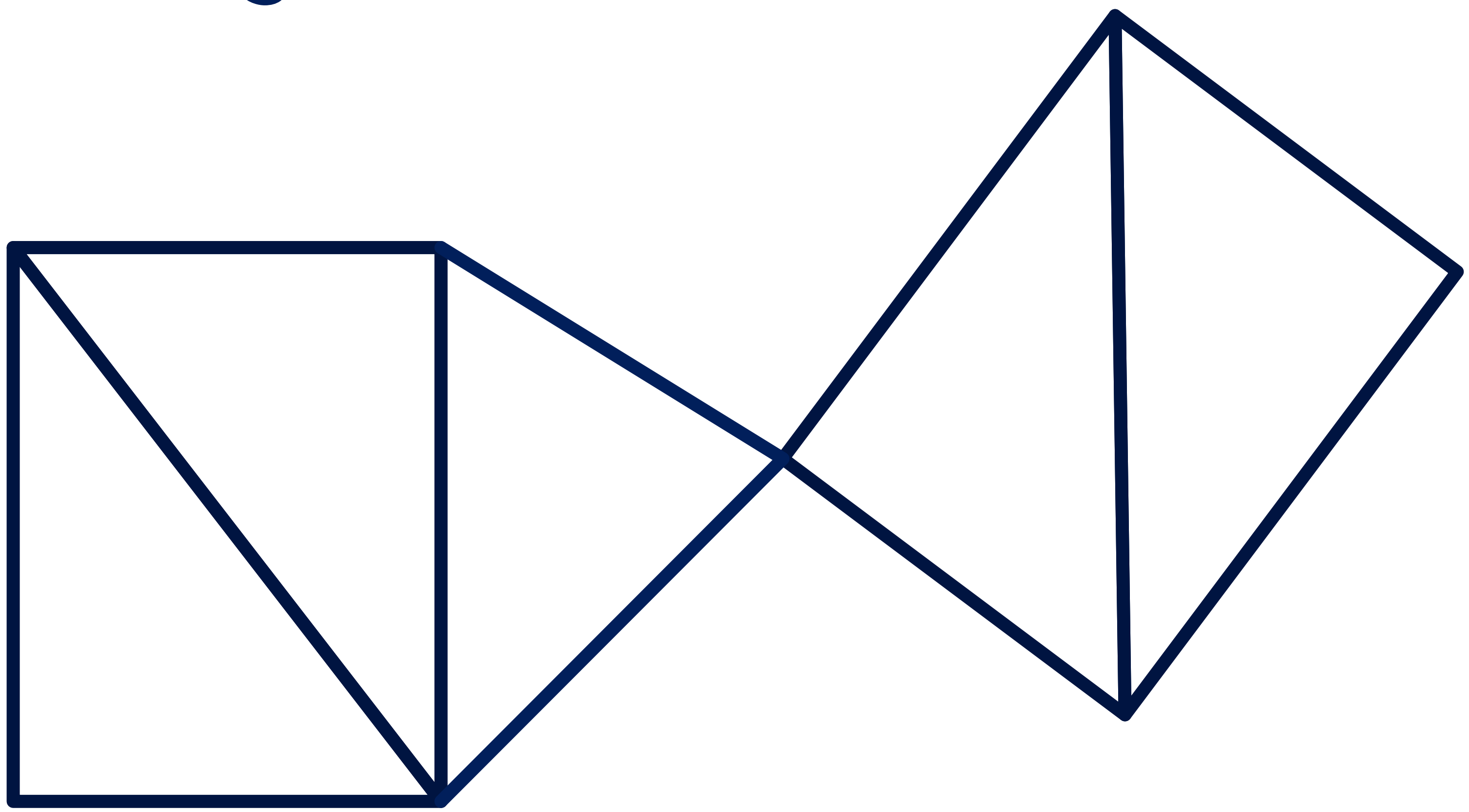
Tearing



Tearing



Tearing



Quasi-Newton Algorithm with Tearing

Algorithm 3: Quasi-Newton Solver with Backtracking Line Search.

$\mathbf{x}^{(1)} := \mathbf{y};$

$g(\mathbf{x}^{(1)}) := \texttt{evalObjective}(\mathbf{x}^{(1)})$

for $k = 1, \ldots,$ numIterations **do**

$\nabla g(\mathbf{x}^{(k)}) := \texttt{evalGradient}(\mathbf{x}^{(k)})$ 0

$\delta\mathbf{x} := -(\mathbf{M}/h^2 + \mathbf{L})^{-1} \nabla g(\mathbf{x}^{(k)})$

$\alpha := 1/\beta$ Original L

repeat

$\alpha := \beta\alpha$

$\mathbf{x}^{(k+1)} := \mathbf{x}^{(k)} + \alpha\delta\mathbf{x}$

$g(\mathbf{x}^{(k+1)}) := \texttt{evalObjective}(\mathbf{x}^{(k+1)})$ 0

until $g(\mathbf{x}^{(k+1)}) \leq g(\mathbf{x}^{(k)}) + \gamma\alpha \operatorname{tr}((\nabla g(\mathbf{x}^{(k)}))^{\mathsf{T}} \delta\mathbf{x});$

end

Parallelization

- Local Step / Gradient Evaluation Step ?
  - Yes
- Global Step / Decent Direction Evaluation ?
  - Dependents on the Linear Solver

Choice of Linear Solver: Direct Solver

Pros:
Accurate
Fast in CPU if Prefactorized

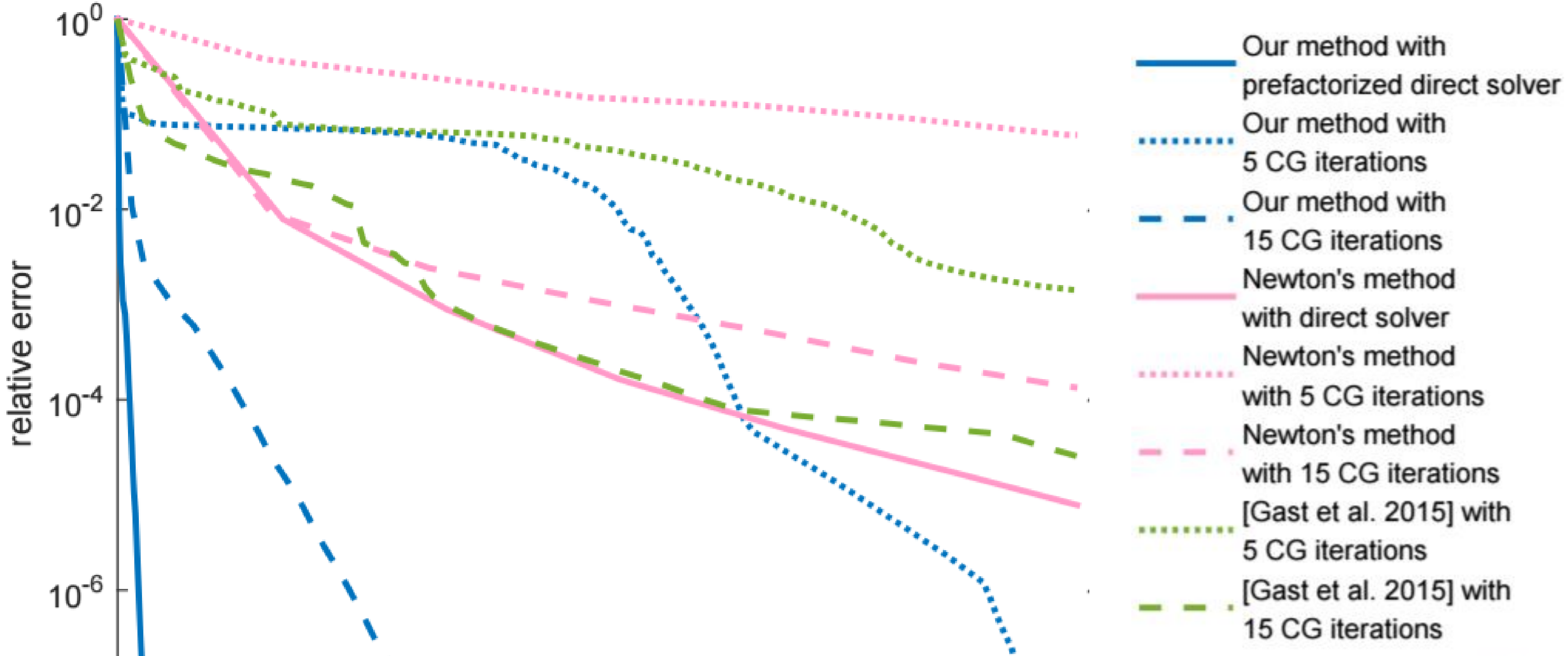
Cons:
Hard to Parallelize
Memory Consuming

Choice of Linear Solver: Iterative Solvers

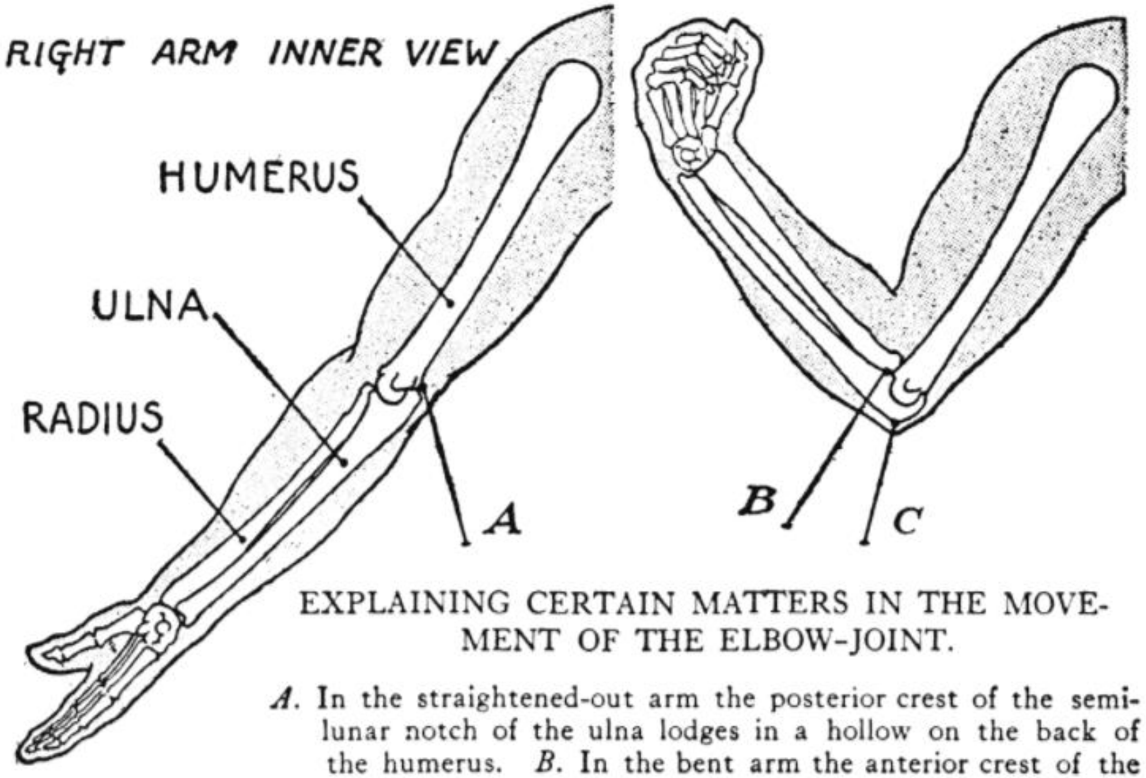
[Wang 2015]

[Fratarcangeli et al. 2016]

Choice of Linear Solver: CG



Simulating Stiff/Rigid Materials



EXPLAINING CERTAIN MATTERS IN THE MOVEMENT OF THE ELBOW-JOINT.

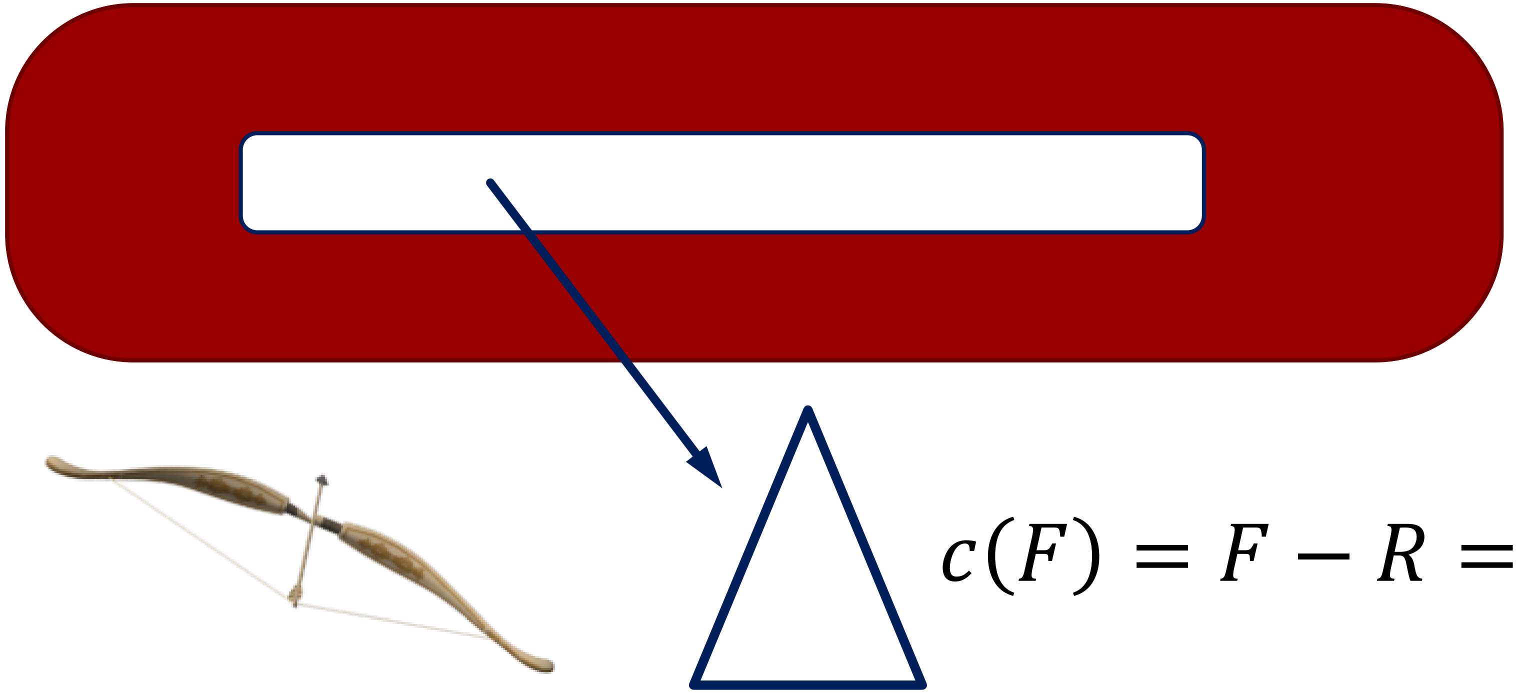
A. In the straightened-out arm the posterior crest of the semilunar notch of the ulna lodges in a hollow on the back of the humerus. *B*. In the bent arm the anterior crest of the notch lodges in a hollow on the front of the humerus. *C*. The point of the elbow, very conspicuous in the bent arm.

Simulating Stiff/Rigid Materials

[Image courtesy of FistfulOfTalent.com]

Increasing Stiffness Directly?

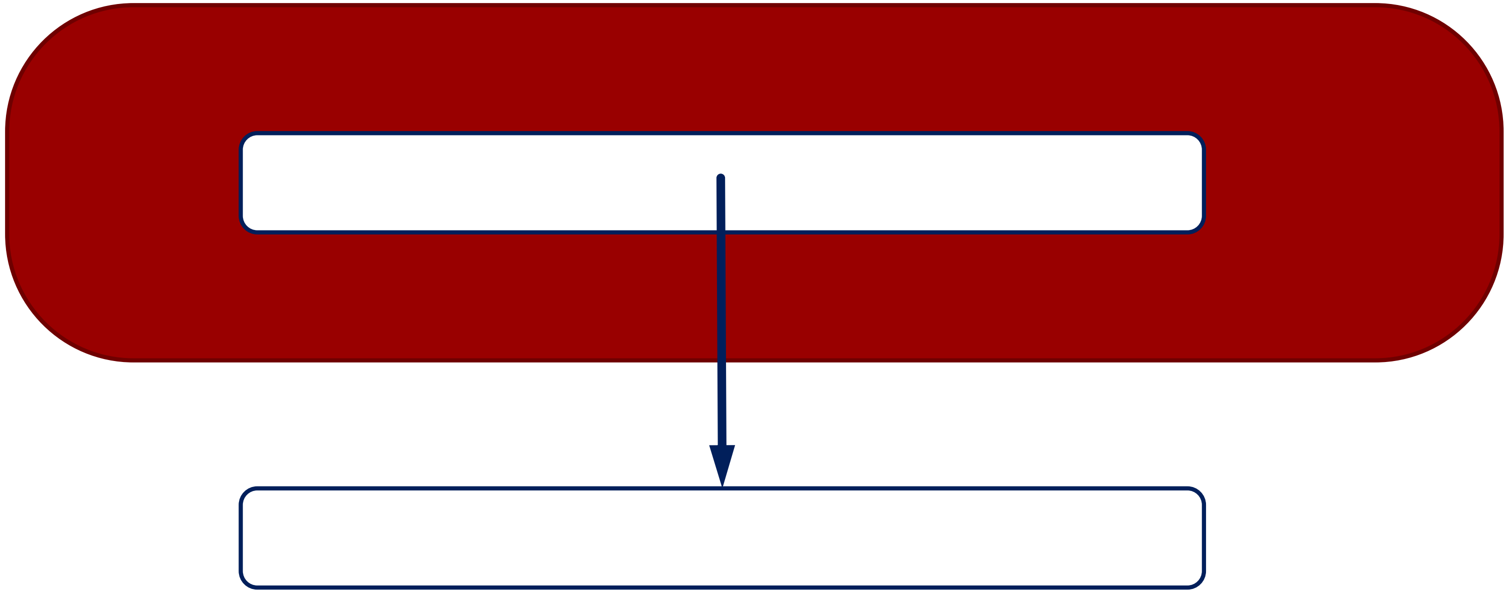
Using Hard Constraints



$$c(F) = F - R = 0$$

[Tournier et al. 2015]

Using Hard Constraints (Cont'd)



$$x = RX + t$$

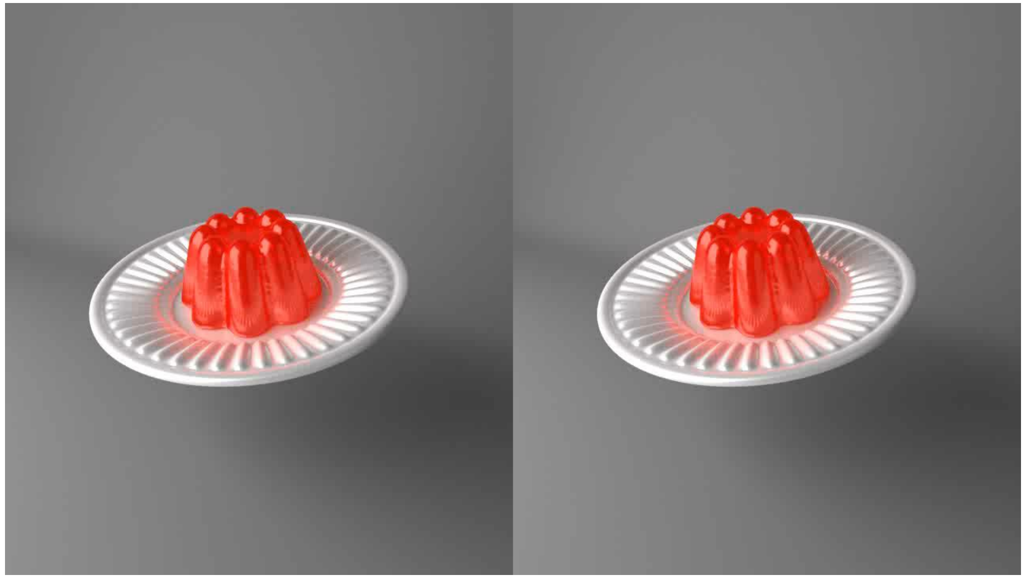
Damping

- Current damping model: post-processing models – Ether drag, PBD damping

[Li et al. 2018]

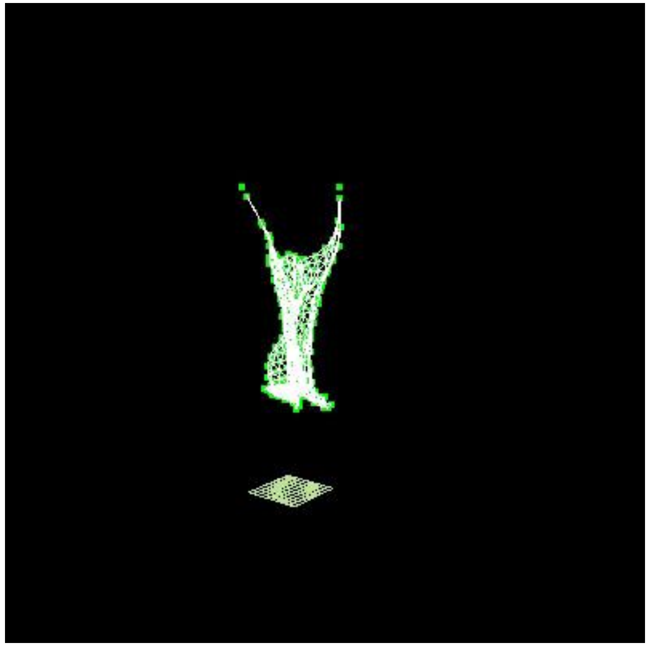
Other Time Integrators

- More vivid motion?
  - Other Integrators
    - Implicit Midpoint
    - Newmark-Beta
    - BDF2
    - [Bathe 2007] Integrator
  - Energy Momentum Methods
    - [Dimitar et al. 2018]



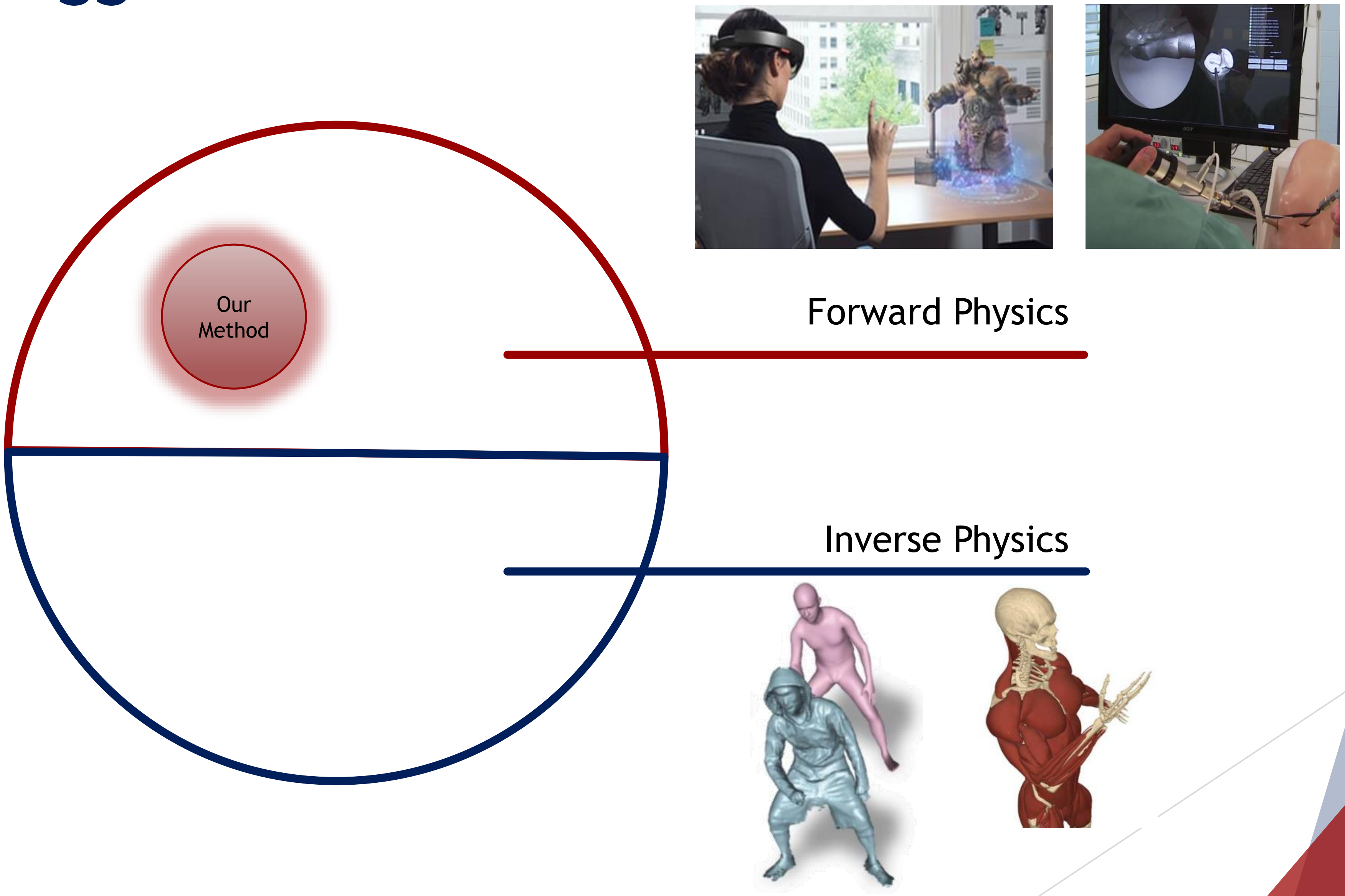
What’s Next?

- Bring Machine Learning to Physics?

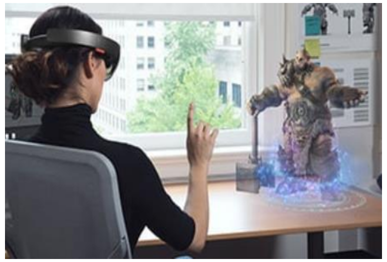


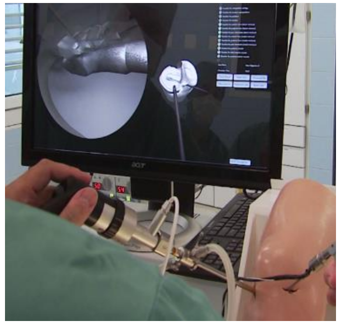
[Video courtesy of Junior Rojas]

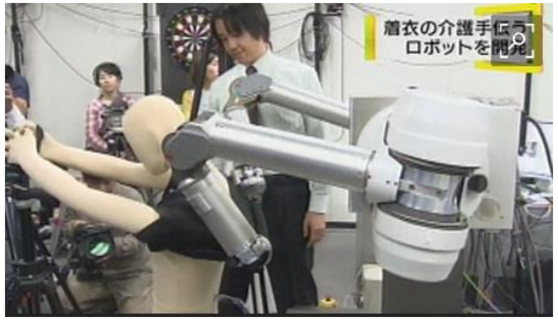
A Bigger Picture

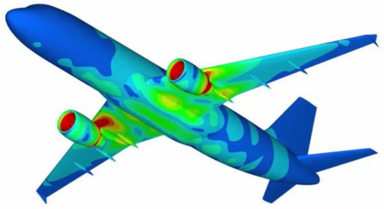


A Bigger Picture









Phys-based Simulation

Thank You