DiffPD: Differentiable Projective Dynamics
Tao Du, Kui Wu, Pingchuan Ma, Sebastien Wah, Andrew Spielberg,
Daniela Rus, Wojciech Matusik
MIT CSAIL
Simulation platforms for learning and robotics

COMSOL
Issac Gym
PyBullet
Gazebo
REDMAX
MuJoCo
Simulation platforms for learning and robotics

Simulation platform

- Computational design
- Perception and sensing
- Planning and control
- Multi-agent collaboration
Simulation platforms for learning and robotics

Differentiable simulation platform

- Computational design
- Perception and sensing
- Planning and control
- Multi-agent collaboration
DiffPD: a differentiable soft-body simulator

A simulator that unlocks interesting downstream applications.
**DiffPD: a differentiable soft-body simulator**

A simulator that reveals interesting mathematical insights for developing differentiable simulators (more on this later).
Related work

Soft-body simulation

- Bouaziz 2014
- Liu 2017
- Ly 2020
- Macklin 2020

Differentiable physics

- Hahn 2019
- Hu 2019
- Qiao 2020
- Geilinger 2020
Method
Background: implicit time integration

Consider the $i$-th step of simulation with timestep $h$.

Input: nodal position $\mathbf{x}_i$ and velocity $\mathbf{v}_i$.

Output: new nodal position $\mathbf{x}_{i+1}$.

\[
\mathbf{x}_{i+1} = \mathbf{x}_i + h\mathbf{v}_{i+1}
\]
\[
\mathbf{v}_{i+1} = \mathbf{v}_i + h\mathbf{M}^{-1}[-\nabla E(\mathbf{x}_{i+1}) + \mathbf{f}_{\text{ext}}]
\]
Background: implicit time integration

Consider the $i$-th step of simulation with timestep $h$.

Input: nodal position $\mathbf{x}_i$ and velocity $\mathbf{v}_i$.

Output: new nodal position $\mathbf{x}_{i+1}$.

\[
\mathbf{x}_{i+1} = \mathbf{x}_i + h\mathbf{v}_{i+1}
\]
\[
\mathbf{v}_{i+1} = \mathbf{v}_i + h\mathbf{M}^{-1}[-\nabla E(\mathbf{x}_{i+1}) + \mathbf{f}_{\text{ext}}]
\]

mass matrix
Background: implicit time integration

Consider the $i$-th step of simulation with timestep $h$.

Input: nodal position $\mathbf{x}_i$ and velocity $\mathbf{v}_i$.

Output: new nodal position $\mathbf{x}_{i+1}$.

\[
\mathbf{x}_{i+1} = \mathbf{x}_i + h\mathbf{v}_{i+1}
\]

\[
\mathbf{v}_{i+1} = \mathbf{v}_i + hM^{-1}\left[\nabla E(\mathbf{x}_{i+1}) + \mathbf{f}_{\text{ext}}\right]
\]

- Mass matrix
- Internal force from elastic energy
Consider the $i$-th step of simulation with timestep $h$.

Input: nodal position $\mathbf{x}_i$ and velocity $\mathbf{v}_i$.

Output: new nodal position $\mathbf{x}_{i+1}$.

\[
\mathbf{x}_{i+1} = \mathbf{x}_i + h\mathbf{v}_{i+1}
\]
\[
\mathbf{v}_{i+1} = \mathbf{v}_i + h\mathbf{M}^{-1}[-\nabla E(\mathbf{x}_{i+1}) + \mathbf{f}_{\text{ext}}]
\]

- mass matrix
- internal force from elastic energy
- external force
Background: implicit time integration

Recast it as a saddle-point problem: find \( \nabla g(x_{i+1}) = 0 \) where

\[
g(x) := \frac{1}{2h^2} (x - y)^\top M(x - y) + E(x)
\]
Background: implicit time integration

Recast it as a saddle-point problem: find $\nabla g(x_{i+1}) = 0$ where

$$g(x) := \frac{1}{2h^2} (x - y)^\top M(x - y) + E(x)$$

$$y := x_i + h v_i + h^2 M^{-1} f_{\text{ext}}$$ is independent of $x$. 
Background: implicit time integration

Newton’s method: $\mathbf{x}^{k+1} = \mathbf{x}^k + \Delta \mathbf{x}^k$ where

$$\nabla^2 g(\mathbf{x}^k)\Delta \mathbf{x}^k = \nabla g(\mathbf{x}^k)$$

Bottleneck: solving the matrix $\nabla^2 g(\mathbf{x}^k)$:

$$\nabla^2 g(\mathbf{x}^k) = \frac{1}{h^2} \mathbf{M} + \nabla^2 E(\mathbf{x}^k)$$
Background: implicit time integration

Newton’s method: \( \mathbf{x}^{k+1} = \mathbf{x}^k + \Delta \mathbf{x}^k \) where

\[
\nabla^2 g(\mathbf{x}^k) \Delta \mathbf{x}^k = \nabla g(\mathbf{x}^k)
\]

Bottleneck: solving the matrix \( \nabla^2 g(\mathbf{x}^k) \):

\[
\nabla^2 g(\mathbf{x}^k) = \frac{1}{h^2} \mathbf{M} + \nabla^2 E(\mathbf{x}^k)
\]

requires recomputation whenever \( \mathbf{x}^k \) changes!
Background: differentiable simulation

Consider backpropagating loss $L$ in the $i$-th step of simulation.

$$\frac{\partial L}{\partial y} = \frac{\partial L}{\partial x_{i+1}} \frac{\partial x_{i+1}}{\partial y}$$

Recall that $\nabla g(x_{i+1}) = 0$. By differentiating it w.r.t. $y$ we have

$$\frac{\partial x_{i+1}}{\partial y} = \frac{1}{h^2} [\nabla^2 g(x_{i+1})]^{-1} M$$
Background: differentiable simulation

Putting everything together, we have \( \frac{\partial L}{\partial y} = \frac{1}{h^2} z^T M \) where

\[
\nabla^2 g(x_{i+1}) z = \left( \frac{\partial L}{\partial x_{i+1}} \right)^T
\]
Background: differentiable simulation

Putting everything together, we have \( \frac{\partial L}{\partial y} = \frac{1}{h^2} z^T M \) where

\[
\nabla^2 g(x_{i+1}) z = \left( \frac{\partial L}{\partial x_{i+1}} \right)^T
\]

We see that solving \( \nabla^2 g \), again, is the bottleneck.
Recap

Forward simulation: $\nabla^2 g(x^k)\Delta x^k = \nabla g(x^k)$.

Backpropagation: $\nabla^2 g(x_{i+1})z = \left(\frac{\partial L}{\partial x_{i+1}}\right)^\top$. 
Insight

Forward and backward computation share the same bottleneck.

Forward simulation: $\nabla^2 g(x^k) \Delta x^k = \nabla g(x^k)$.

Backpropagation: $\nabla^2 g(x_{i+1}) z = \left(\frac{\partial L}{\partial x_{i+1}}\right)^\top$. 
Insight

Efficient solvers for forward simulation exist.

Efficient forward simulation: \( \nabla^2 g(x^k) \Delta x^k = \nabla g(x^k) \).

Backpropagation: \( \nabla^2 g(x_{i+1}) z = \left( \frac{\partial L}{\partial x_{i+1}} \right)^T \).
Insight

Can we borrow them to build efficient backpropagation solver as well?

Efficient forward simulation: $\nabla^2 g(x^k) \Delta x^k = \nabla g(x^k)$.

Efficient backpropagation: $\nabla^2 g(x_{i+1}) z = \left( \frac{\partial L}{\partial x_{i+1}} \right)^T$. 
Simulation speedup: Projective Dynamics (PD)

Consider a special class of $E = \sum_c E_c$ where

$$E_c(x) := \min_{p_c \in M_c} ||G_c x - p_c||^2_2$$
Simulation speedup: Projective Dynamics (PD)

Consider a special class of $E = \sum_c E_c$ where

$$E_c(x) := \min_{p_c \in \mathcal{M}_c} \|G_c x - p_c\|_2^2$$

energy on each finite element
Simulation speedup: Projective Dynamics (PD)

Consider a special class of $E = \sum_c E_c$ where

$$E_c(x) := \min_{p_c \in \mathcal{M}_c} ||G_c x - p_c||_2^2$$

energy on each finite element, e.g., deformation gradient
Simulation speedup: Projective Dynamics (PD)

Consider a special class of $E = \sum_c E_c$ where

$$E_c(x) := \min_{p_c \in \mathcal{M}_c} \|G_c x - p_c\|_2^2$$

energy on each finite element, e.g., projection of local feature, e.g., deformation gradient local feature
Simulation speedup: Projective Dynamics (PD)

The saddle-point problem $\nabla g = 0$ is now modified accordingly:

$$\min_{x, \{p_c \in \mathcal{M}_c\}} \tilde{g}(x, \{p_c\})$$

where

$$\tilde{g}(x, \{p_c\}) := \frac{1}{2h^2} (x - y)^T M (x - y) + \sum_c \| G_c x - p_c \|^2_2$$
Simulation speedup: Projective Dynamics (PD)

The saddle-point problem $\nabla g = 0$ is now modified accordingly:

$$\min_{x, \{p_c \in \mathcal{M}_c\}} \frac{1}{2h^2} (x - y)^\top M (x - y) + \sum_c \|G_c x - p_c\|_2^2$$
Simulation speedup: Projective Dynamics (PD)

The saddle-point problem $\nabla g = 0$ is now modified accordingly:

$$\min_{\{p_c \in \mathcal{M}_c\}} \frac{1}{2h^2} (x - y)^T M (x - y) + \sum_c \| G_c x - p_c \|^2_2$$

The global step: fix $p_c$ and solve $x$, constant matrix in the quadratic form and can be prefactorized.
Simulation speedup: Projective Dynamics (PD)

The saddle-point problem $\nabla g = 0$ is now modified accordingly:

$$
\begin{align*}
\min_{x, \{p_c \in M_c\}} & \quad \frac{1}{2h^2} (x - y)^\top M (x - y) + \sum_c \|G_c x - p_c\|_2^2 \\
\end{align*}
$$

The global step: fix $p_c$ and solve $x$, constant matrix in the quadratic form and can be prefactorized.

The local step: fix $x$ and solve $p_c$, parallelizable among elements.
Backpropagation speedup: DiffPD

With PD, $\nabla^2 g$ becomes

$$\nabla^2 g(x) = \frac{1}{h^2} M + \sum_c G_c^T G_c - \sum_c G_c^T \frac{\partial p_c}{\partial x}$$

$$:= A - \Delta A$$
Backpropagation speedup: DiffPD

With PD, $\nabla^2 g$ becomes

$$\nabla^2 g(x) = \frac{1}{h^2} M + \sum_c G_c^T G_c - \sum_c G_c^T \frac{\partial p_c}{\partial x}$$

:= $A - \Delta A$

constant matrix and
source of efficiency
Backpropagation speedup: DiffPD

With PD, $\nabla^2 g$ becomes

$$\nabla^2 g(\mathbf{x}) = \frac{1}{h^2} \mathbf{M} + \sum_c \mathbf{G}_c^\top \mathbf{G}_c - \sum_c \mathbf{G}_c^\top \frac{\partial \mathbf{p}_c}{\partial \mathbf{x}}$$

$$:= \mathbf{A} - \Delta \mathbf{A}$$

constant matrix and residual that can be source of efficiency computed in parallel.
Backpropagation speedup: DiffPD

Recall the bottleneck:

$$\nabla^2 g(x_{i+1})z = b := \left( \frac{\partial L}{\partial x_{i+1}} \right)^T$$

With \( \nabla^2 g = A - \Delta A \) it becomes \((A - \Delta A)z = b\), or

$$Az = b + \Delta Az$$
Backpropagation speedup: DiffPD

Recall the bottleneck:

\[ \nabla^2 g(x_{i+1})z = b := \left( \frac{\partial L}{\partial x_{i+1}} \right)^T \]

With \( \nabla^2 g = A - \Delta A \) it becomes \( (A - \Delta A)z = b \), or

\[ Az = b + \Delta Az \]

The global step: constant \( A \), already factorized.
Backpropagation speedup: DiffPD

Recall the bottleneck:

$$\nabla^2 g(x_{i+1})z = b := \left( \frac{\partial L}{\partial x_{i+1}} \right)^T$$

With $\nabla^2 g = A - \Delta A$ it becomes $(A - \Delta A)z = b$, or

$$Az = b + \Delta Az$$

The global step: constant $A$, already factorized.  
The local step: parallelizable on elements.
Extension one: quasi-Newton speedup

We recall that Liu [2017] proposed a quasi-Newton approach to speed up PD even more. We can transfer it to backpropagation too.

Extension one: evaluation

Cantilever (8019 DoFs, 25 steps, 10ms timestep, no contact)
Extension two: boundary conditions

Consider the global step in PD:

$$Ax_{i+1} = \text{right-hand side from local step.}$$

Efficient solvers exist for $A$ with erased rows and columns:

$$(A + uv^T)^{-1} = A^{-1} - \frac{A^{-1}u(A^{-1}v)^T}{1 + v^TA^{-1}u}$$
Extension two: boundary conditions

Let’s take a look at its source of efficiency:

\[ (A + uv^T)^{-1} = A^{-1} - \frac{A^{-1}u(A^{-1}v)^T}{1 + v^TA^{-1}u} \]
Extension two: boundary conditions

Let’s take a look at its source of efficiency:

\[(A + uv^T)^{-1} = A^{-1} - \frac{A^{-1}u(A^{-1}v)^T}{1 + v^TA^{-1}u}\]
Extension two: boundary conditions

Let’s take a look at its source of efficiency:

$$(A + uv^T)^{-1} = A^{-1} - \frac{A^{-1}u(A^{-1}v)^T}{1 + v^TA^{-1}u}$$
Extension two: boundary conditions

Let’s take a look at its source of efficiency:

\[
(A + uv^T)^{-1} = A^{-1} - \frac{A^{-1}u(A^{-1}v)^T}{1 + v^TA^{-1}u}
\]

It turns out that we can transfer this idea to backpropagation too, which DiffPD used for contact handling.
Extension two: evaluation

Rolling sphere (2469 DoFs, 100 steps, 5ms timestep, with contact)
Insight, reiterated

Efficient forward simulation solvers can be transferred to efficient backpropagation solvers!
Applications
System identification

Goal: estimating the material parameters of a plant from its motion.

Input (ground truth)  Initial guess  After optimization
Initial state optimization

Goal: optimizing time-invariant actuation to reach the target.
Open-loop control

Goal: optimizing actuation to roll forward.

Initial guess

After optimization
Closed-loop control

Goal: optimizing a neural network controller so that the starfish rises.
Real-to-sim transfer

Goal: estimating scene parameters to reconstruct the balls’ motion.
User gallery: computer graphics

DiffPD are used in computational design of soft characters and cloth.

Design of soft underwater characters [Ma 2021]

Cloth simulation [Li 2022]

Ma et al. DiffAqua: A Differentiable Computational Design Pipeline for Soft Underwater Swimmers with Shape Interpolation. SIGGRAPH 2021
Li et al. DiffCloth: Differentiable Cloth Simulation with Dry Frictional Contact. TOG 2022
User gallery: robotics

DiffPD has also been used in modeling and controlling soft robots.

Soft robot study [Du 2021]

Du et al. Underwater Soft Robot Modeling and Control with Differentiable Simulation. RA-L 2021
User gallery: machine learning

DiffPD also attracts users from the learning community.

Ma et al. RISP: Rendering-Invariant State Predictor with Differentiable Simulation and Rendering for Cross-Domain Parameter Estimation. ICLR 2022
Nava et al. Fast Aquatic Swimmer Optimization with Differentiable Projective Dynamics and Neural Network Hydrodynamic Models. ICML 2022
Summary
Conclusions

Numerical techniques in forward simulation and backpropagation are two sides of the same coin.

Efficient forward simulation: \( \nabla^2 g(x^k) \Delta x^k = \nabla g(x^k) \).

Efficient backpropagation: \( \nabla^2 g(x_{i+1}) z = \left( \frac{\partial L}{\partial x_{i+1}} \right)^T \).
Conclusions

A fast, reliable differentiable soft-body simulator unlocks wide application in *graphics*, *robotics*, and *machine learning*. 
For more information

Project
http://diffpd.csail.mit.edu/

Code
https://github.com/mit-gfx/diff_pd_public

This work is sponsored by DARPA FA8750-20-C-0075, IARPA 2019-19020100001, and NSF 2106962.