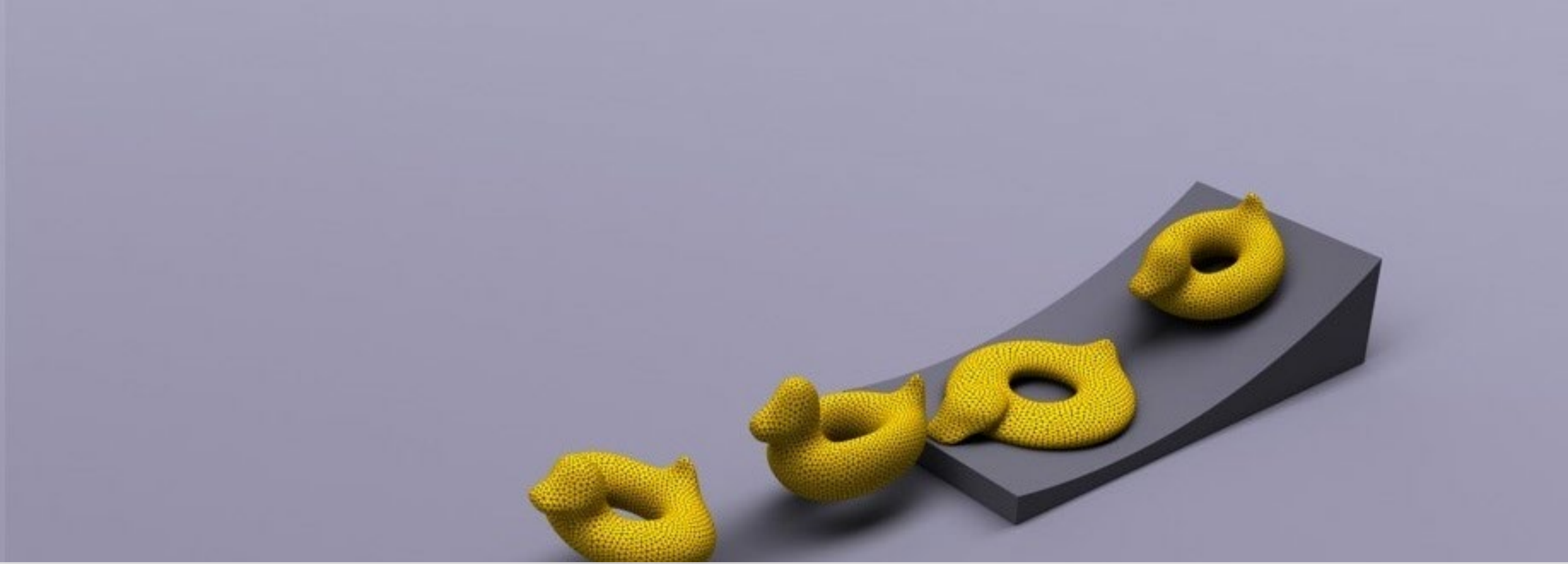


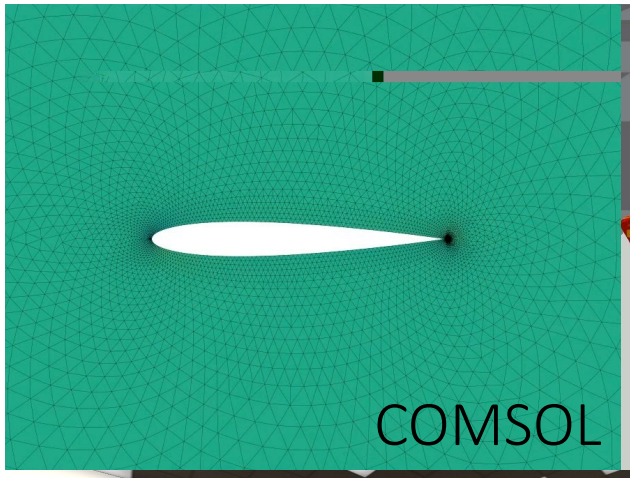
DiffPD: Differentiable Projective Dynamics

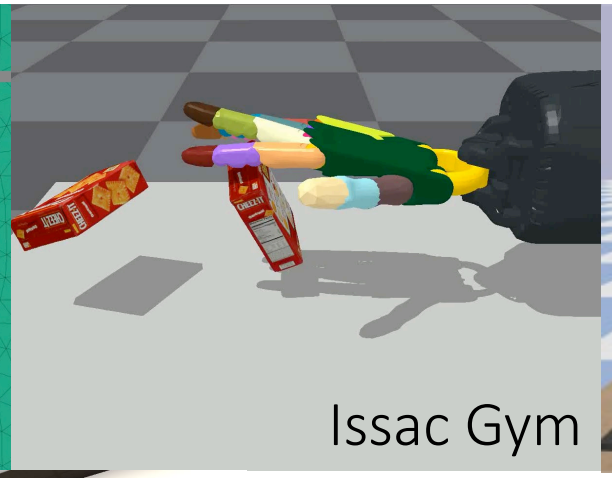
Tao Du, Kui Wu, Pingchuan Ma, Sebastien Wah, Andrew Spielberg,

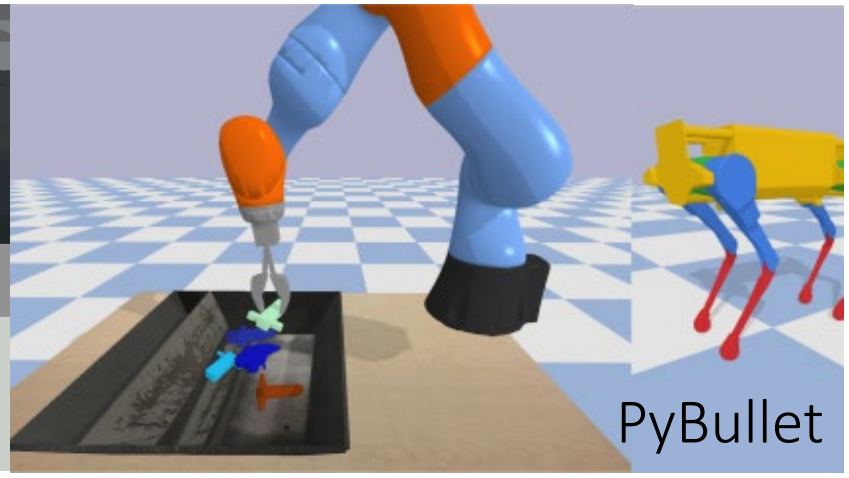
Daniela Rus, Wojciech Matusik

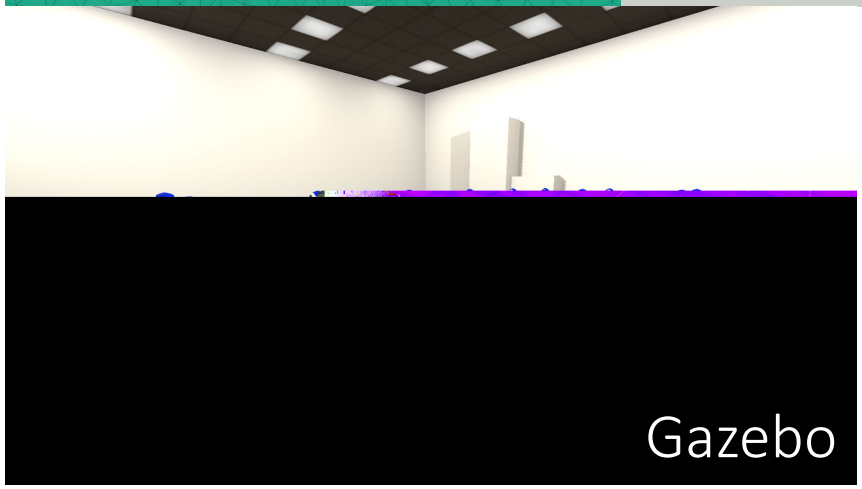
MIT CSAIL

Simulation platforms for learning and robotics









REDMAX

MuJoCo

Method

Background: implicit time integration

Consider the ٰΩ-th step of simulation with timestep ٰΩ.

Input: nodal position ȯ۳ and velocity ٰI۳.

Output: new nodal position ȯ۳ٹ۳.

$$\text{ȯ}_{\text{۳ٹ۳}} = \text{ȯ}_{\text{۳}} + \text{ٰΩ}\,\text{ٰI}_{\text{۳ٹ۳}}$$

$$\text{ٰI}_{\text{۳ٹ۳}} = \text{ٰI}_{\text{۳}} + \text{ٰΩ}\,\text{ٰE}^{\text{ٹ}\,\text{ٹ}}\left[\text{سر}\ \text{ٰO}(\text{ȯ}_{\text{۳ٹ۳}}) + \text{ٰH}_{\text{ڱڱڈ}}\right]$$

Background: implicit time integration

Consider the n-th step of simulation with timestep Δt.

Input: nodal position x_n and velocity v_n.

Output: new nodal position x_{n+1}.

$$x_{n+1} = x_n + \Delta t v_{n+1}$$
$$v_{n+1} = v_n + \Delta t M^{-1}[f_{int}(x_{n+1}) + f_{ext}]$$

mass matrix

Background: implicit time integration

Consider the n-th step of simulation with timestep Δt.

Input: nodal position x_n and velocity v_n.

Output: new nodal position x_{n+1}.

$$x_{n+1} = x_n + \Delta t v_{n+1}$$
$$v_{n+1} = v_n + \Delta t M^{-1}[-\nabla E(x_{n+1}) + f_{ext}]$$

mass matrix

internal force from elastic energy

Background: implicit time integration

Consider the n-th step of simulation with timestep h.

Input: nodal position x_n and velocity v_n.

Output: new nodal position x_{n+1}.

$$x_{n+1} = x_n + h v_{n+1}$$
$$v_{n+1} = v_n + h M^{-1}[-\nabla E(x_{n+1}) + f_{ext}]$$

mass matrix

internal force from elastic energy

external force

Background: implicit time integration

Recast it as a saddle-point problem: find $\text{Ώ}(\text{ȯ}_{\text{ڀٿ}}) = \text{ج}$ where

$$\text{Ώ}(\text{ȯ}) \;\; \text{ḧ} \;\; \frac{1}{2\text{Ώ}_{\text{ڀ}}}(\text{ȯ}\ \text{سر}\ \text{ȯ})^{\text{ڎ}}\ \text{Ἑ}(\text{ȯ}\ \text{سر}\ \text{ȯ}) + \text{Ὁ}(\text{ȯ})$$

Background: implicit time integration

Recast it as a saddle-point problem: find $\text{Ω}(\text{ȯ}_{\text{ڀٿ}}) = \text{ج}$ where

$$\text{Ω}(\text{ȯ})\ \ddot{\text{h}}\ \frac{1}{2\text{Ω}^{\text{ڀ}}}(\text{ȯ}\ \text{سر}\ \text{ȯ})^{\text{ڙ}}\ \text{E}(\text{ȯ}\ \text{سر}\ \text{ȯ}) + \text{O}(\text{ȯ})$$

$\text{ȯ}\ \ddot{\text{h}}\ \text{ȯ}_{\text{س}} + \text{Ω}^{\text{ٓ}}\text{I}_{\text{س}} + \text{Ω}^{\text{ڀ}}\text{E}^{\text{ٿ}\ \text{ٻ}}\text{H}^{\text{ٽ}}_{\text{ڳڱڙ}}$ is independent of ȯ.

Background: implicit time integration

Newton's method: ȯ$^{\text{ش ٹ پ}}$ = ȯ^{ش} + ɜ̃ȯ^{ش} where

$$\text{ʼ}\Omega\left(\text{ȯ}^{\text{ش}}\right)\text{ɜ̃ȯ}^{\text{ش}} = \text{ʼ}\Omega\left(\text{ȯ}^{\text{ش}}\right)$$

Bottleneck: solving the matrix $\text{ʼ}\Omega\left(\text{ȯ}^{\text{ش}}\right)$:

$$\text{ʼ}\Omega\left(\text{ȯ}^{\text{ش}}\right) = \frac{1}{\text{ʼ}\Omega\text{پ}}\text{ʼE} + \text{ʼO}\left(\text{ȯ}^{\text{ش}}\right)$$

Background: implicit time integration

Newton's method: $\text{ȯ}^{\text{ش}\text{ٹ}\text{پ}} = \text{ȯ}^{\text{ش}} + \text{ɜ}\text{ȯ}^{\text{ش}}$ where

$$\text{ٮ}\text{Ω}\left(\text{ȯ}^{\text{ش}}\right)\text{ɜ}\text{ȯ}^{\text{ش}} = \text{ٮ}\text{Ω}\left(\text{ȯ}^{\text{ش}}\right)$$

Bottleneck: solving the matrix $\text{ٮ}\text{Ω}\left(\text{ȯ}^{\text{ش}}\right)$:

$$\text{ٮ}\text{Ω}\left(\text{ȯ}^{\text{ش}}\right) = \frac{1}{\text{ٮ}\text{Ω}^{\text{پ}}}\text{ٮE} + \text{ٮO}\left(\text{ȯ}^{\text{ش}}\right)$$

requires recomputation whenever $\text{ȯ}^{\text{ش}}$ changes!

Background: differentiable simulation

Consider backpropagating loss ὐ in the ῞Ω-th step of simulation.

$$\frac{\text{⍑ ⍑ ⍑}}{\text{⍑ ⍑ ⍑}} = \frac{\text{⍑ ⍑ ⍑}}{\text{⍑ ⍑ ⍑}} \frac{\text{⍑ ⍑ ⍑}}{\text{⍑ ⍑ ⍑}}$$

Recall that ῞Ω(⍑ ⍑ ⍑) = ج. By differentiating it w.r.t. ⍑ we have

$$\frac{\text{⍑ ⍑ ⍑}}{\text{⍑ ⍑ ⍑}} = \frac{1}{\text{῞Ω ⍑}} \left[\text{῞Ω(⍑ ⍑ ⍑)} \right] \text{⍑ ⍑ ⍑ E}$$

Background: differentiable simulation

Putting everything together, we have $\frac{\partial \mathcal{L}}{\partial \theta} = \frac{\partial \mathcal{L}}{\partial s'} \frac{\partial s'}{\partial \theta}$ where

$$\frac{\partial s'}{\partial \theta}(\theta_{sim}) = \left(\frac{\partial s'}{\partial \theta} \right)^{T}$$

Background: differentiable simulation

Putting everything together, we have $\frac{\text{ج}\text{لإ}}{\text{ó}\text{لإ}} = \frac{\text{ٮ}}{\text{ٮ}\text{س}}\text{ö}^{\text{ڑ}}\,\text{ʿE}$ where

$$\text{ٮ}\text{ʿΩ}(\text{ó}_{\text{ٮ}\text{ٹ}\text{س}})\text{ö} = \left(\frac{\text{ٮ}\text{ö}}{\text{ٮ}\text{ó}_{\text{ٮ}\text{ٹ}\text{س}}}\right)^{\text{ڑ}}$$

We see that solving $\text{ٮ}\text{ʿΩ}$, again, is the bottleneck.

Recap

Forward simulation: $\text{ٮٚ}\Omega(\text{ٮٚ}^{\text{ش}})\text{ٮٚ}\text{ٮٚ}^{\text{ش}} = \text{ٮٚ}\Omega(\text{ٮٚ}^{\text{ش}})$.

Backpropagation: $\text{ٮٚ}\Omega(\text{ٮٚ}_{\text{ٮٚ}\text{ٮٚ}})\text{ٮٚ} = \left(\frac{\text{ج}\text{لإ}}{\text{ٮٚ}\text{ٮٚ}\text{لإ}}\right)^{\text{ڑ}}$.

Insight

Forward and backward computation share the same bottleneck.

Forward simulation: $\text{ٱ}\Omega(\text{ٱ}^{\text{ش}})\text{ٱ}^{\text{ش}} = \text{ٱ}\Omega(\text{ٱ}^{\text{ش}})$.

Backpropagation: $\text{ٱ}\Omega(\text{ٱ}_{\text{ٱ}})\text{ٱ} = \left(\frac{\text{ج}}{\text{ٱ}_{\text{ٱ}}}\right)^{\text{ڑ}}$.

Insight

Efficient solvers for forward simulation exist.

Efficient forward simulation: $\text{ٮٚ}\Omega(\text{ٮٚ}^{\text{ش}})\text{ٮٚ}^{\text{ش}} = \text{ٮٚ}\Omega(\text{ٮٚ}^{\text{ش}})$.

Backpropagation: $\text{ٮٚ}\Omega(\text{ٮٚ}_{\text{ٮٚ}\text{ٮٚ}})\text{ٮٚ} = \left(\frac{\text{ج}\text{لإ}}{\text{ٮٚ}_{\text{ٮٚ}}\text{لإ}}\right)^{\text{ڑ}}$.

Insight

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

Efficient forward simulation: $\text{ٱ}\Omega(\text{ٱ}^{\text{ش}})\text{ٱ}^{\text{ش}} = \text{ٱ}\Omega(\text{ٱ}^{\text{ش}})$.

Efficient backpropagation: $\text{ٱ}\Omega(\text{ٱ}_{\text{ٱ}}^{\text{ٱ}})\text{ٱ} = \left(\frac{\text{ٱ}^{\text{ٱ}}}{\text{ٱ}^{\text{ٱ}}}\right)^{\text{ٱ}}$.

Simulation speedup: Projective Dynamics (PD)

Consider a special class of $\text{Ȯ} = \text{B}_{\text{ر}}\,\text{Ȯ}_{\text{ر}}$ where

$$\text{Ȯ}_{\text{ر}}(\text{ȯ})\ \ \ddot{\text{h}}\ \min_{\text{ʼI}_{\text{يز}}\ \text{N}\ \text{ה}_{\text{يز}}} \|\text{ɛ̇}_{\text{ر}}\,\text{ȯ}\ \text{س}\ \text{ʼI}_{\text{ر}}\|_{\text{ٮٮٮ}}$$

Simulation speedup: Projective Dynamics (PD)

Consider a special class of $\text{Ȯ} = \text{B}_{\text{ر}}\,\text{Ȯ}_{\text{ر}}$ where

$$\text{Ȯ}_{\text{ر}}(\text{ȯ}) \;\; \text{ḧ} \;\; \min_{\text{ʼI}_{\text{يز}}\,\text{N}\,\text{ה}_{\text{يز}}} \|\text{ɛ̇}_{\text{ر}}\,\text{ȯ}\,\text{س}\,\text{ʼI}_{\text{ر}}\|_{\text{ڀ}}^{\text{ڀ}}$$

energy on each
finite element

Simulation speedup: Projective Dynamics (PD)

Consider a special class of $\mathcal{O} = B_{\text{ر}} \mathcal{O}_{\text{ر}}$ where

$$\mathcal{O}_{\text{ر}}(\mathring{o}) \ \ddot{h} \ \min_{\mathcal{I}_{\text{يز}}^{N} \ \text{ה يز}} \| \dot{\varepsilon}_{\text{ر}} \mathring{o} \ \text{سر} \ \mathcal{I}_{\text{ر}} \|_{\text{ڀڀ}}$$

energy on each finite element

local feature, e.g., deformation gradient

Simulation speedup: Projective Dynamics (PD)

Consider a special class of $\mathcal{O} = B_{\text{ر}} \mathcal{O}_{\text{ر}}$ where

$$\mathcal{O}_{\text{ر}}(\text{ȯ}) \ \ddot{\text{h}} \ \min_{\text{'I}_{\text{يز}}^{\text{N}} \text{ ה يز}} \| \text{ɛ̇}_{\text{ر}} \text{ȯ} \text{ س } \text{'I}_{\text{ر}} \|_{\text{ٮٮ}}^{\text{ٮٮ}}$$

energy on each finite element

local feature, e.g., deformation gradient

projection of local feature

Simulation speedup: Projective Dynamics (PD)

The saddle-point problem $\ddot{\Omega} = ج$ is now modified accordingly:

$$\min_{\dot{o},\{\text{'I}_{\text{يز}}\ \text{ה}\ \text{يز}\}} \ddot{\text{ﻫﻲ}}(\dot{o},\{\text{'I}_{\text{ر}}\})$$

where

$$\ddot{\text{ﻫﻲ}}(\dot{o},\{\text{'I}_{\text{ر}}\}) \ \ddot{\text{h}} \ \frac{1}{2\ddot{\Omega}^{\text{پ}}}(\dot{o}\ \text{سر}\ \dot{o})^{\text{ڻ}}\ \text{'E}(\dot{o}\ \text{سر}\ \dot{o}) + \text{ضحي}_{\text{ر}}\ ||\text{ع}_{\text{ر}}\dot{o}\ \text{سر}\ \text{'I}_{\text{ر}}||^{\text{پ}}_{\text{پ}}$$

Simulation speedup: Projective Dynamics (PD)

The saddle-point problem ‘Ω = ج is now modified accordingly:

$$\min_{\text{ỏ},\{\text{'I}_{\text{يز}}\ \text{N}\ \text{ה}_{\text{يز}}\}} \frac{1}{2\text{''Ω}_{\text{پ}}} (\text{ỏ}\ \text{سر}\ \text{ỏ})^{\text{ڑ}}\ \text{'E}(\text{ỏ}\ \text{سر}\ \text{ỏ}) + \text{ضحي}_{\text{ر}}\ ||\text{ٔ}_{\text{ر}}\text{ỏ}\ \text{سر}\ \text{'I}_{\text{ر}}||_{\text{پ}}^{\text{پ}}$$

Simulation speedup: Projective Dynamics (PD)

The saddle-point problem $\mathbf{A}\mathbf{x} = \mathbf{b}$ is now modified accordingly:

$$\min_{\mathbf{x},\{\mathbf{z}_i\}} \frac{1}{2}\mathbf{x}^T\mathbf{A}\mathbf{x} - \mathbf{x}^T\mathbf{b} + \frac{w}{2}\sum_i \|\mathbf{C}_i\mathbf{x} - \mathbf{z}_i\|^2$$

The global step: fix $\mathbf{z}_i$ and solve $\mathbf{x}$, constant matrix in the quadratic form and can be prefactorized.

Simulation speedup: Projective Dynamics (PD)

The saddle-point problem $\mathbf{A}\mathbf{x} = \mathbf{b}$ is now modified accordingly:

$$\min_{\mathbf{x},\{\mathbf{z}_i \in \mathcal{C}_i\}} \frac{1}{2}\mathbf{x}^T\mathbf{A}\mathbf{x} - \mathbf{x}^T\mathbf{b} + \frac{w}{2}\sum_i \|\mathbf{S}_i\mathbf{x} - \mathbf{z}_i\|_2^2$$

The global step: fix $\mathbf{z}_i$ and solve $\mathbf{x}$, constant matrix in the quadratic form and can be prefactorized.

The local step: fix $\mathbf{x}$ and solve $\mathbf{z}_i$, parallelizable among elements.

Backpropagation speedup: DiffPD

With PD, ‘Ω becomes

‘Ω(ȯ) = 1/‘Ω + ضحي ر ٮُر ٮٔ ر سر ضحي ر ٮٔ ٮُر ‘I ر ‘ȯ

ḧ ‘A سر ɜ‘A

Backpropagation speedup: DiffPD

With PD, ᵕᵕΩ becomes

$$\text{ᵕᵕΩ}(\text{ȯ}) = \frac{1}{\text{ᵕΩᵕ}}\,\text{ʿE} + \text{ضحي}_{\text{ر}}\ \text{ʿɛ}_{\text{ر}}^{\text{ڑ}}\ \text{ʿɛ}_{\text{ر}}\ \text{سر}\ \text{ضحي}_{\text{ر}}\ \text{ʿɛ}_{\text{ر}}^{\text{ڑ}}\ \frac{\text{ʾI}_{\text{ر}}}{\text{ȯ}}$$

$$\text{ʿA}\ \text{سر}\ \text{ɜʿA}\ \ddot{\text{h}}$$

constant matrix and
source of efficiency

Backpropagation speedup: DiffPD

With PD, ᵕᵕΩ becomes

$$ᵕᵕΩ(ȯ) = \frac{1}{ᵕΩ}ʿE + ضحي ر\ ɛ̇ ڑ ɛ̇ ر\ سر\ ضحي ر\ ɛ̇ ڑ \frac{ʾI ر\ ┬}{ȯ\ ┬}$$

$$ḧ\ ʿA\ سر\ ɜʿA$$

constant matrix and source of efficiency

residual that can be computed in parallel.

Backpropagation speedup: DiffPD

Recall the bottleneck:

$$\text{ᵕ̈Ω}(\text{ȯ}_{\text{ᵕ̈ᵗᵗ}})\text{ȯ} = \text{ᵕ̈H}\ \ddot{\text{h}}\ \left(\frac{\text{ᵕ̈ᵗ}}{\text{ᵕ̈ᵗ}}\right)^{\text{ᵗ}}$$

With $\text{ᵕ̈Ω} = \text{ᵕ̈A}$ سر ɜᵕ̈A it becomes $(\text{ᵕ̈A}$ سر ɜ$\text{ᵕ̈A})\text{ȯ} = \text{ᵕ̈H}$, or

$$\text{ᵕ̈Aȯ} = \text{ᵕ̈H} + \text{ɜᵕ̈Aȯ}$$

Backpropagation speedup: DiffPD

Recall the bottleneck:

$$\text{ﾞﾞﾛ}(\text{ﾟﾟﾛﾟﾟ})\text{ﾟﾟ} = \text{ﾞﾞﾄ}\ \ddot{\text{h}}\ \left(\frac{\text{ﾟﾟ}\ \text{ﾟﾟ}}{\text{ﾟﾟﾛﾟﾟ}}\right)^{\text{ﾟﾟ}}$$

With $\text{ﾞﾞﾛ} = \text{ﾞﾞﾄ}\ \text{ﾟﾟﾛ}\ \text{ﾟﾟﾛ}$ it becomes $(\text{ﾞﾞﾄ}\ \text{ﾟﾟﾛ}\ \text{ﾟﾟﾛ})\text{ﾟﾟ} = \text{ﾞﾞﾄ}$, or

$$\text{ﾞﾞﾄﾟﾟ} = \text{ﾞﾞﾄ} + \text{ﾟﾟﾛﾟﾟ}$$

The global step: constant ﾞﾞﾄ,
already factorized.

Backpropagation speedup: DiffPD

Recall the bottleneck:

$$\text{ﾟﾞﾟ}\Omega(\text{ȯ}_{\text{ﾟﾟﾟﾟﾟﾟ}})\text{ȯ} = \text{ʼH}\ \ddot{\text{h}}\left(\frac{\text{ﾟ}\ \text{ʊ̀}}{\text{ﾟ}\ \text{ȯ}_{\text{ﾟﾟﾟﾟﾟﾟ}}}\right)^{\text{ﾟ}}$$

With $\text{ﾟﾞﾟ}\Omega = \text{ʼA}\ \text{ﾟ}\ \text{ﾟ}\ \text{ʼA}$ it becomes $(\text{ʼA}\ \text{ﾟ}\ \text{ﾟ}\ \text{ʼA})\text{ȯ} = \text{ʼH}$, or

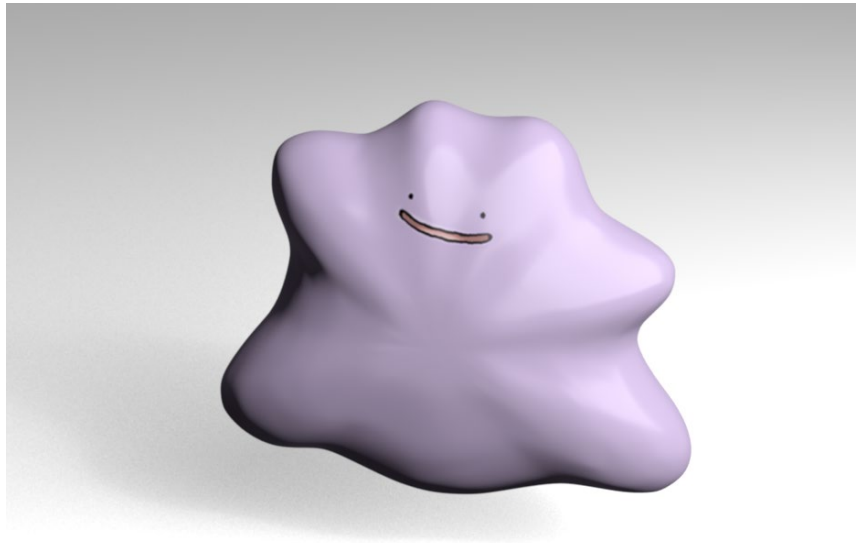
$$\text{ʼAȯ} = \text{ʼH} + \text{ﾟ}\text{ʼAȯ}$$

The global step: constant ʼA, already factorized.

The local step: parallelizable on elements.

Extension one: quasi-Newton speedup

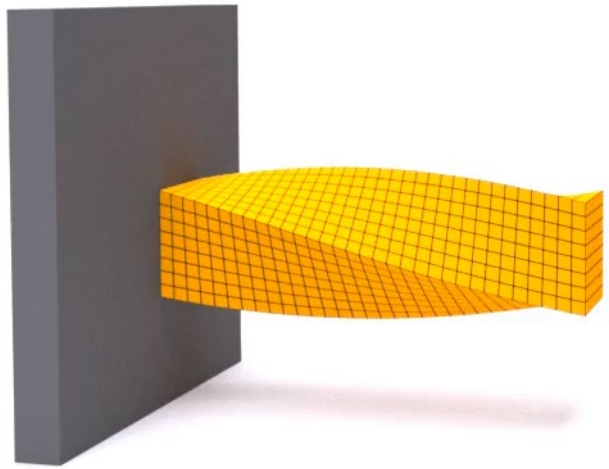
We recall that Liu [2017] proposed a quasi-Newton approach to speed up PD even more. We can transfer it to backpropagation too.

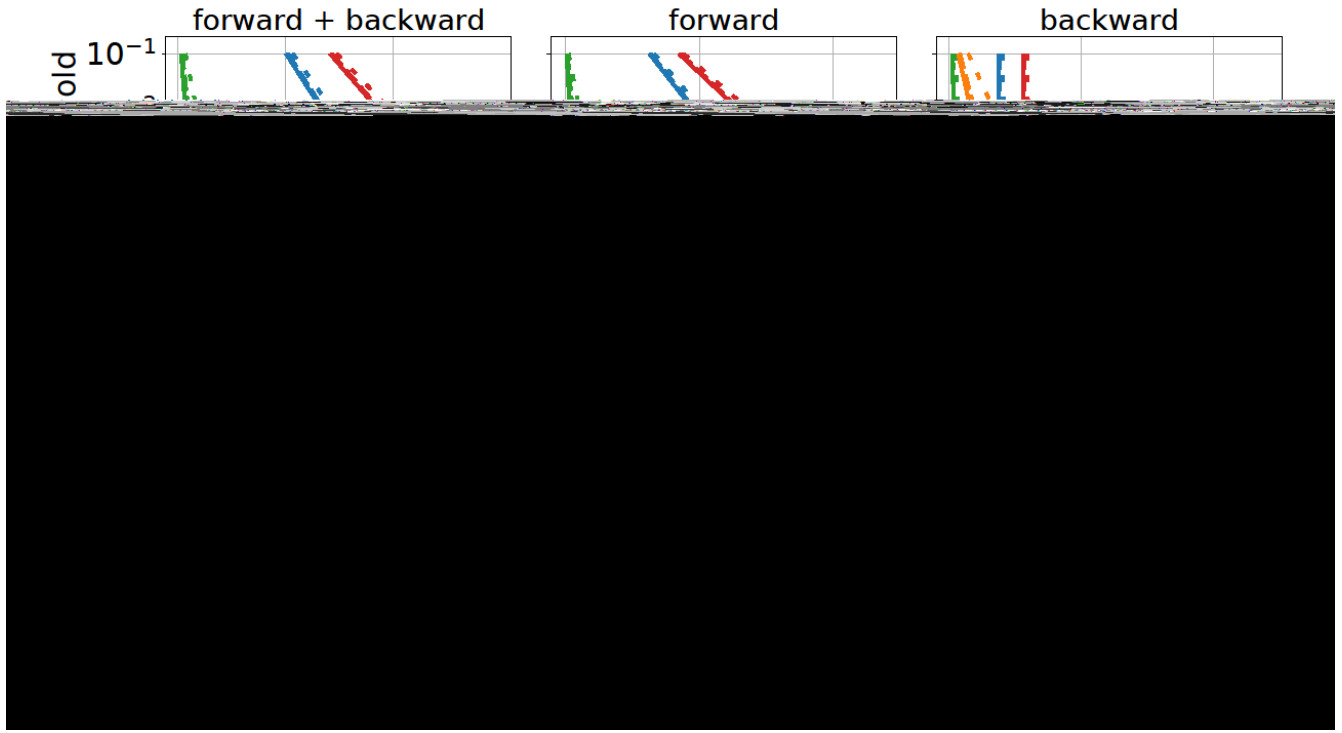


Liu et al. *Quasi-Newton Methods for Real-time Simulation of Hyperelastic Materials*. TOG 2017

Extension one: evaluation

Cantilever (8019 DoFs, 25 steps, 10ms timestep, no contact)





Extension two: boundary conditions

Consider the global step in PD:

$$\text{Ӓ}\text{ȯ}_{\text{ڀٹ۳}} = \text{right-hand side from local step.}$$

Efficient solvers exist for Ӓ with erased rows and columns:

$$(\text{Ӓ} + \text{ʼI}\tilde{\text{ʼI}}^{\text{ڻ}})^{\text{ڀ}} = \text{Ӓ}^{\text{ڀ}}\ \text{س}\ \frac{\text{Ӓ}^{\text{ڀ}}\text{ʼI}(\text{Ӓ}^{\text{ڀ}}\tilde{\text{ʼI}})^{\text{ڻ}}}{1 + \tilde{\text{ʼI}}^{\text{ڻ}}\text{Ӓ}^{\text{ڀ}}\text{ʼI}}$$

Extension two: boundary conditions

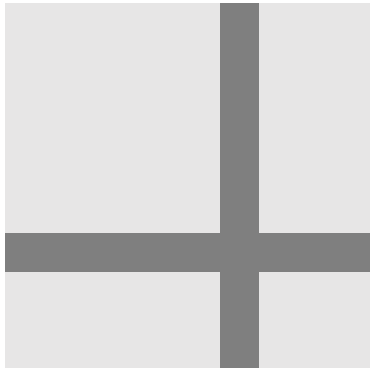
Let’s take a look at its source of efficiency:

$$(\text{“A} + \text{“I}\tilde{\text{’I}}^{\text{ڑ}})^{\text{ڕ}\text{ڕ}} = \text{“A}^{\text{ڕ}\text{ڕ}}\,\text{سر}\,\frac{\text{“A}^{\text{ڕ}\text{ڕ}}\text{“I}(\text{“A}^{\text{ڕ}\text{ڕ}}\tilde{\text{’I}})^{\text{ڑ}}}{1 + \tilde{\text{’I}}^{\text{ڑ}}\text{“A}^{\text{ڕ}\text{ڕ}}\text{“I}}$$

Extension two: boundary conditions

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

$$(\text{“A} + \text{“I}\tilde{\text{’I}}^{\text{ڑ}})^{\text{ڗ}\,\text{ڕ}} = \text{“A}^{\text{ڗ}\,\text{ڕ}}\ \text{سر}\ \frac{\text{“A}^{\text{ڗ}\,\text{ڕ}}\text{“I}(\text{“A}^{\text{ڗ}\,\text{ڕ}}\tilde{\text{’I}})^{\text{ڑ}}}{1 + \tilde{\text{’I}}^{\text{ڑ}}\text{“A}^{\text{ڗ}\,\text{ڕ}}\text{“I}}$$



Extension two: boundary conditions

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

$$(\text{“A} + \text{“I}\tilde{\text{’I}}^{\text{ڑ}})^{\text{ڕ}\text{ڕ}} = \text{“A}^{\text{ڕ}\text{ڕ}} \text{ سر } \frac{\text{“A}^{\text{ڕ}\text{ڕ}}\text{“I}(\text{“A}^{\text{ڕ}\text{ڕ}}\tilde{\text{’I}})^{\text{ڑ}}}{1 + \text{“I}^{\text{ڑ}}\text{“A}^{\text{ڕ}\text{ڕ}}\text{“I}}$$

Extension two: boundary conditions

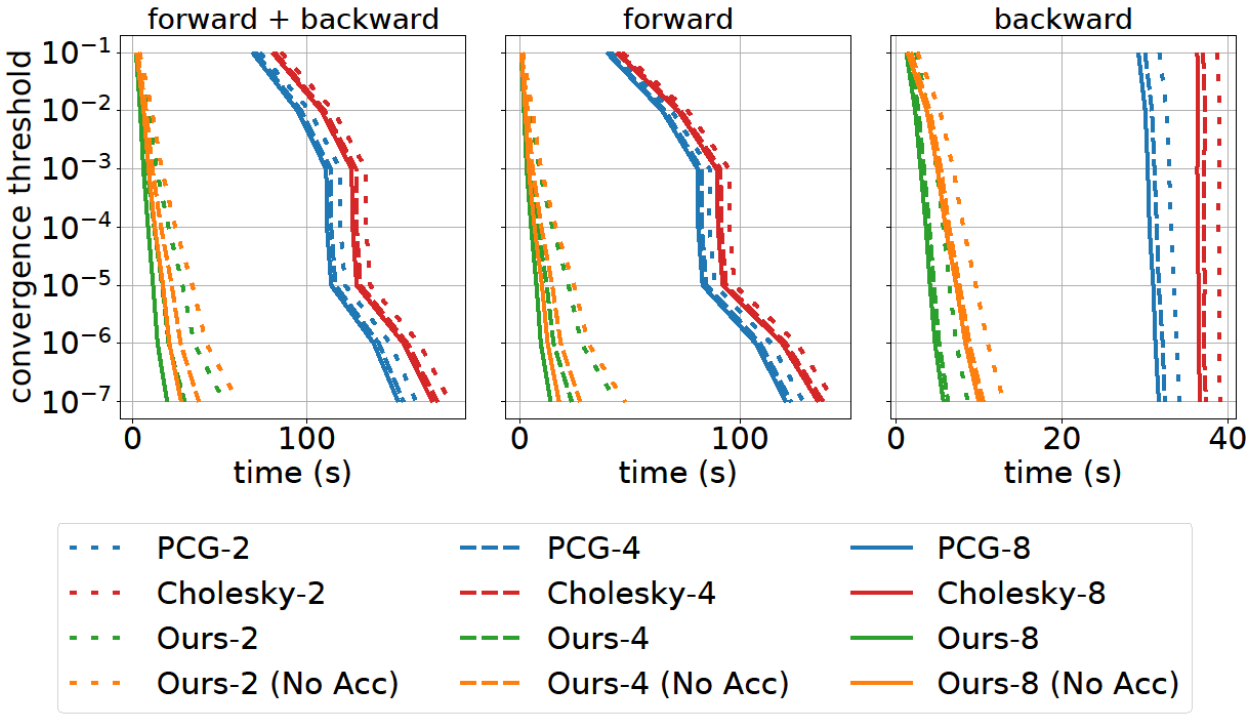
Let’s take a look at its source of efficiency:

$$(\mathbf{A} + \mathbf{I}\tilde{\mathbf{I}}^{T})^{-1} = \mathbf{A}^{-1} - \frac{\mathbf{A}^{-1}\mathbf{I}(\mathbf{A}^{-1}\tilde{\mathbf{I}})^{T}}{1 + \tilde{\mathbf{I}}^{T}\mathbf{A}^{-1}\mathbf{I}}$$

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**)



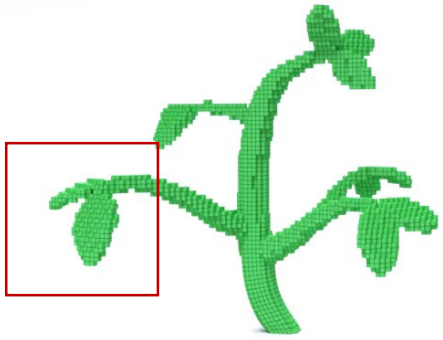
Insight, reiterated

Efficient forward simulation solvers can be transferred to efficient backpropagation solvers!

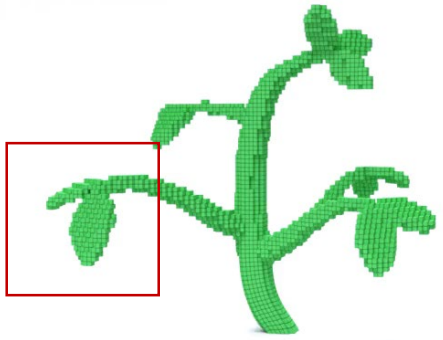
Applications

System identification

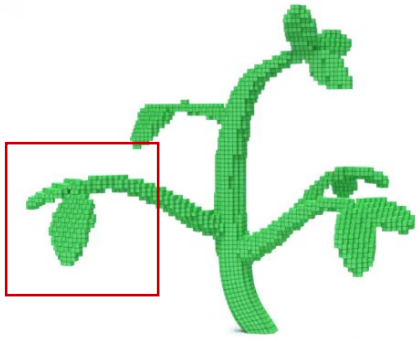
Goal: estimating the material parameters of a plant from its motion.



Input (ground truth)



Initial guess



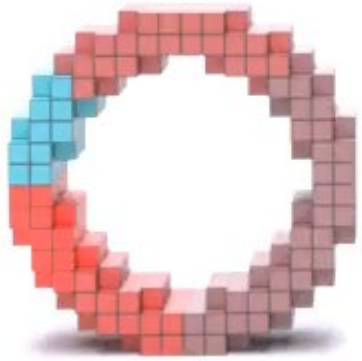
After optimization

Initial state optimization

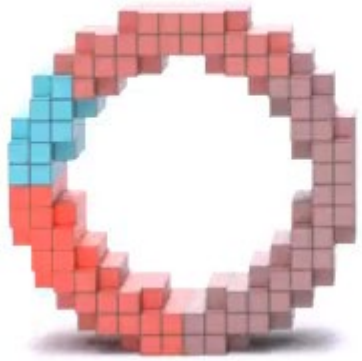
Goal: optimizing time-invariant actuation to reach the target.

Open-loop control

Goal: optimizing actuation to roll forward.



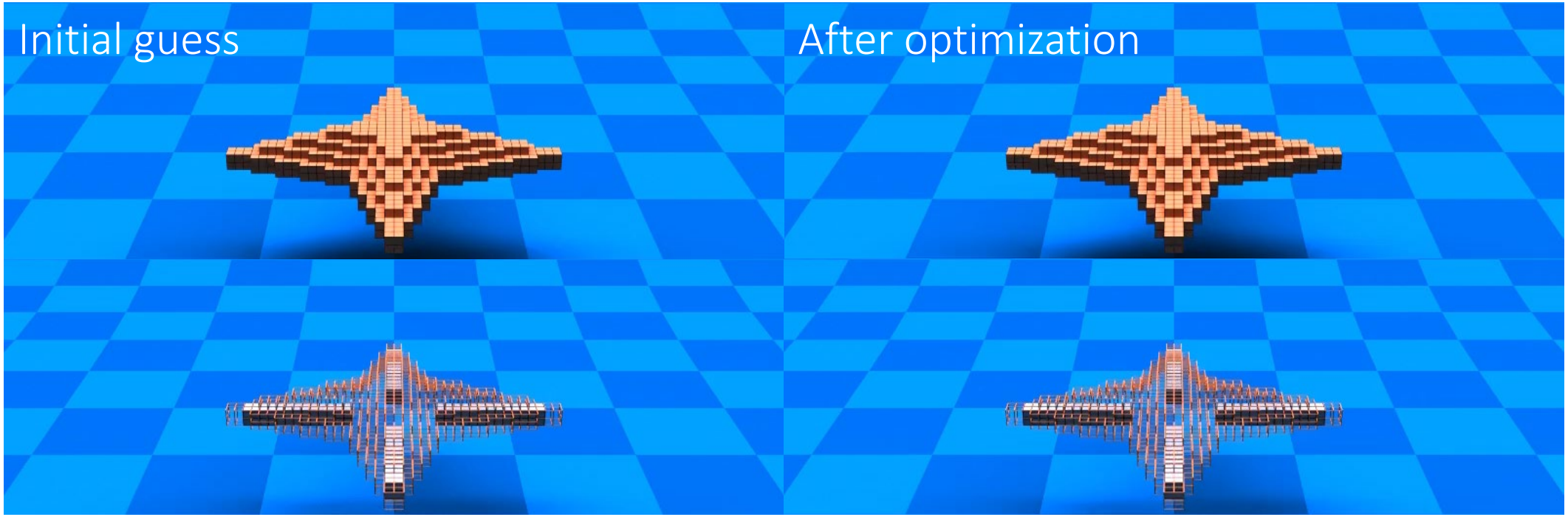
Initial guess



After optimization

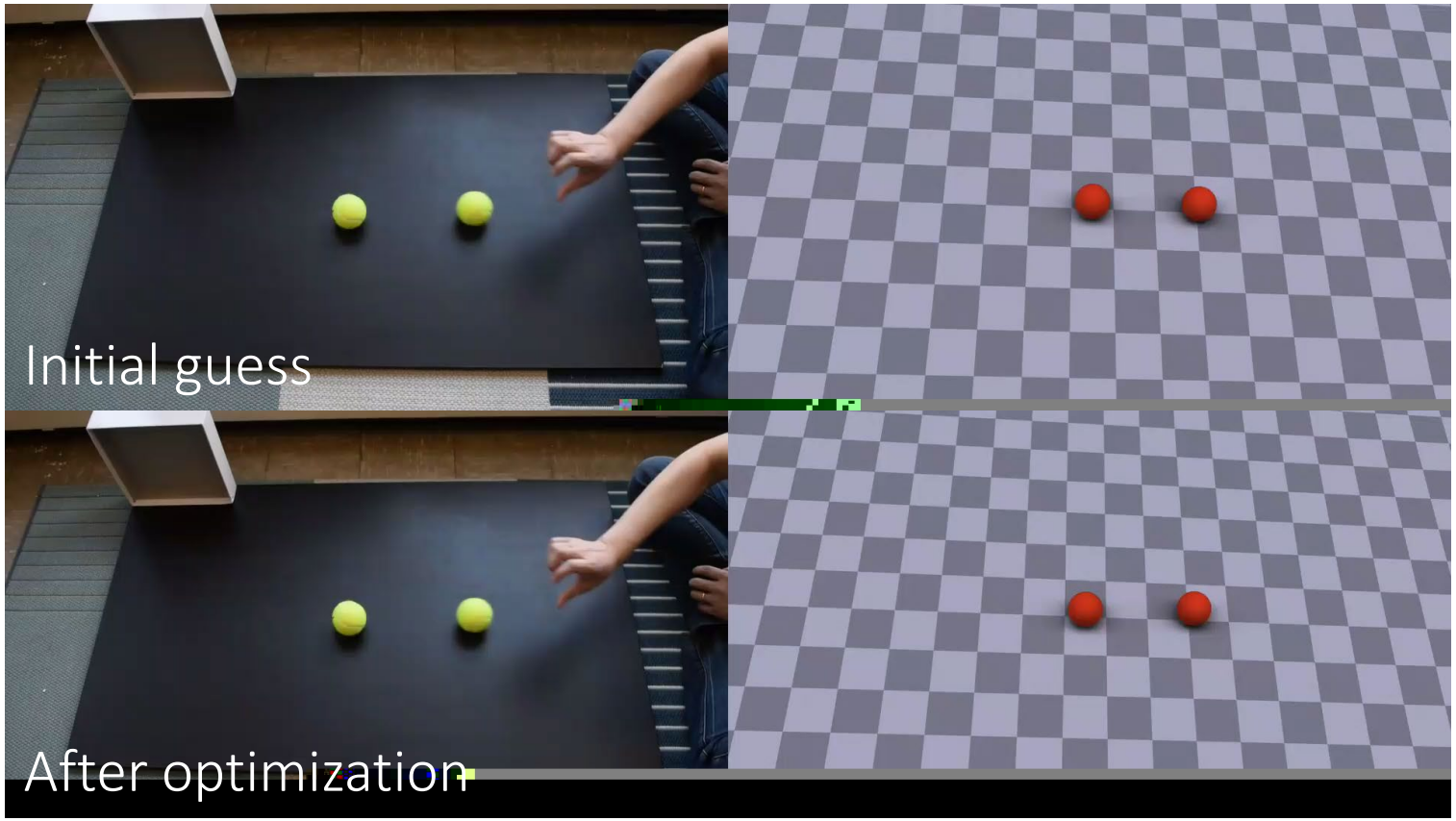
Closed-loop control

Goal: optimizing a neural network controller so that the starfish rises.



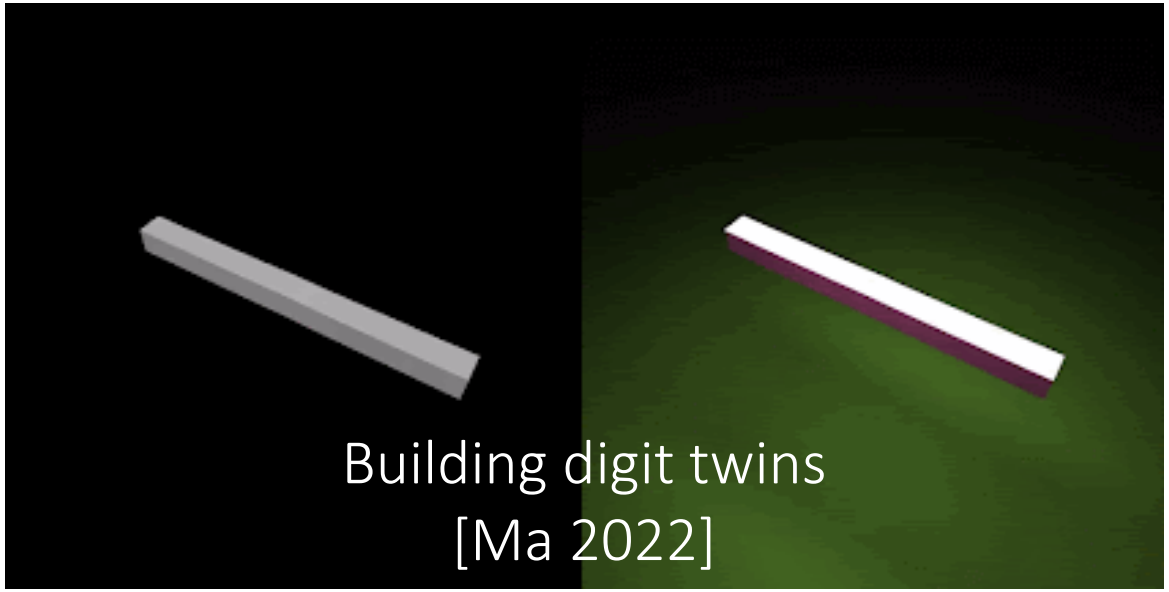
Real-to-sim transfer

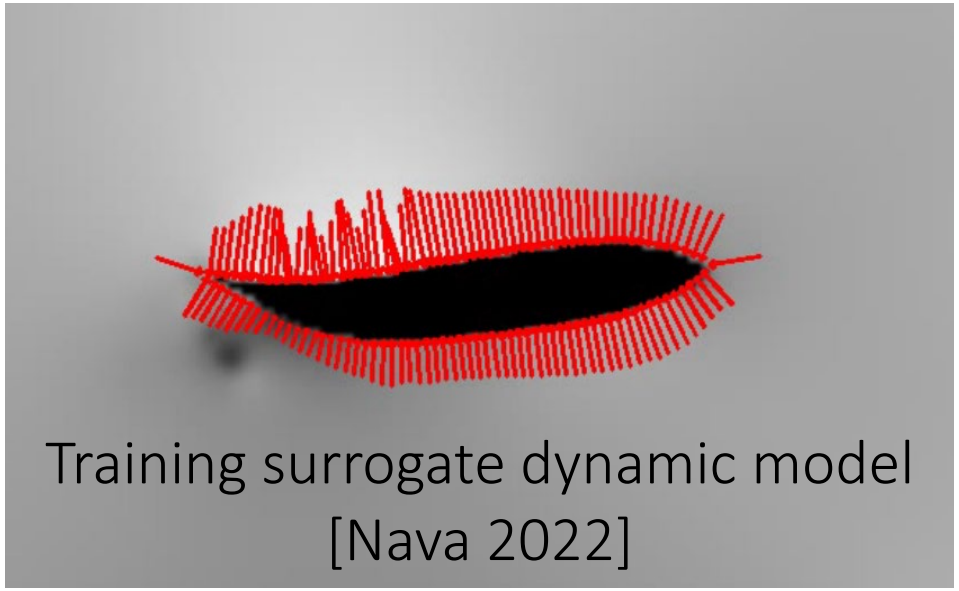
Goal: estimating scene parameters to reconstruct the balls' motion.



User gallery: 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

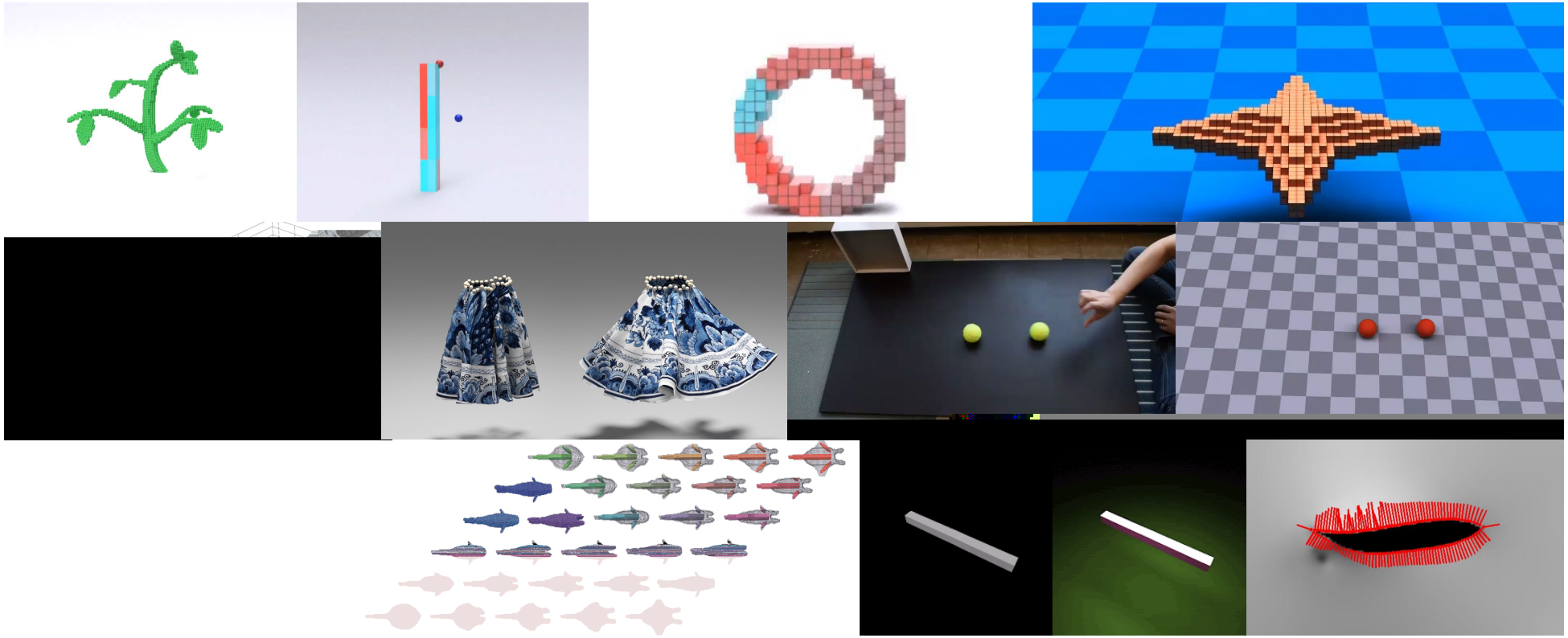
Numerical techniques in forward simulation and backpropagation are two sides of the same coin.

Efficient forward simulation: [illegible]

Efficient backpropagation: [illegible]

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.