Latent-space Dynamics for Reduced Deformable Simulation

Lawson Fulton[1,2], Vismay Modi[1], David Duvenaud[1], David I.W. Levin[1], Alec Jacobson[1]

[1] University of Toronto, Canada

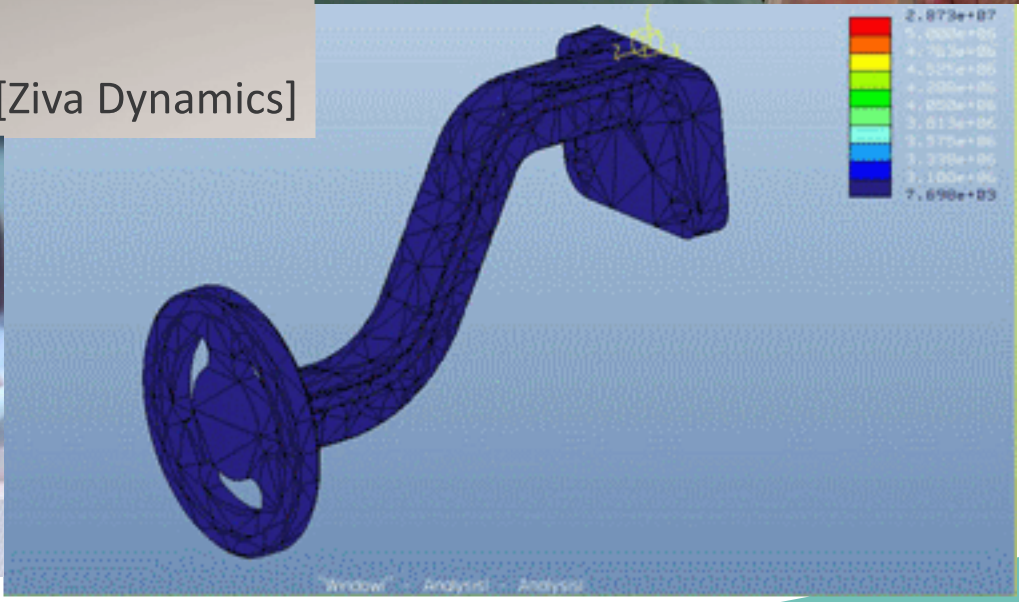
[2] MESH Consultants, Canada

EG2019

The 40° Annual Conference of the European Association for Computer Graphics

Why deformable simulation?

[Ziva Dynamics]

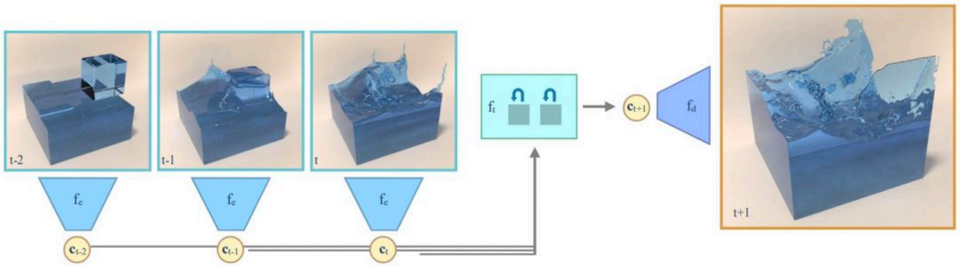


Research Question

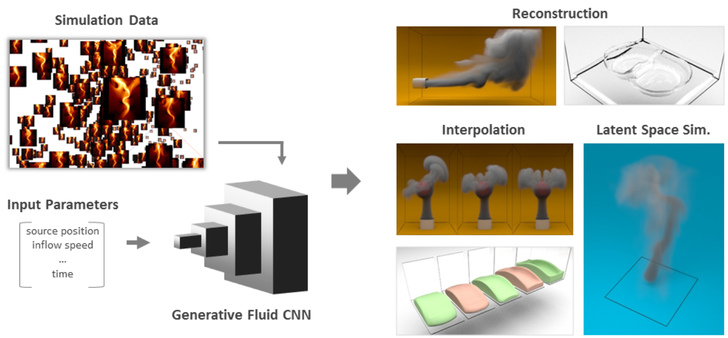
Can we use machine learning to accelerate hyperelastic simulation?

Related Work

Latent-space Physics: Towards Learning the Temporal Evolution of Fluid Flow
Wiewel et al. 2019



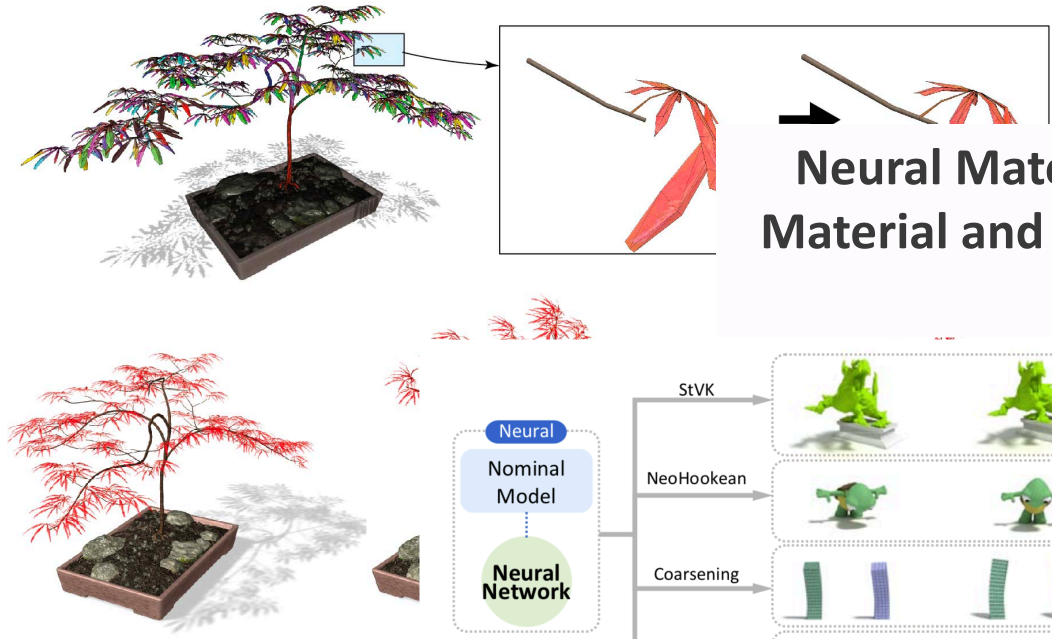
Deep Fluids – A Generative Network for Parameterized Fluid Simulations
Kim et al. 2019



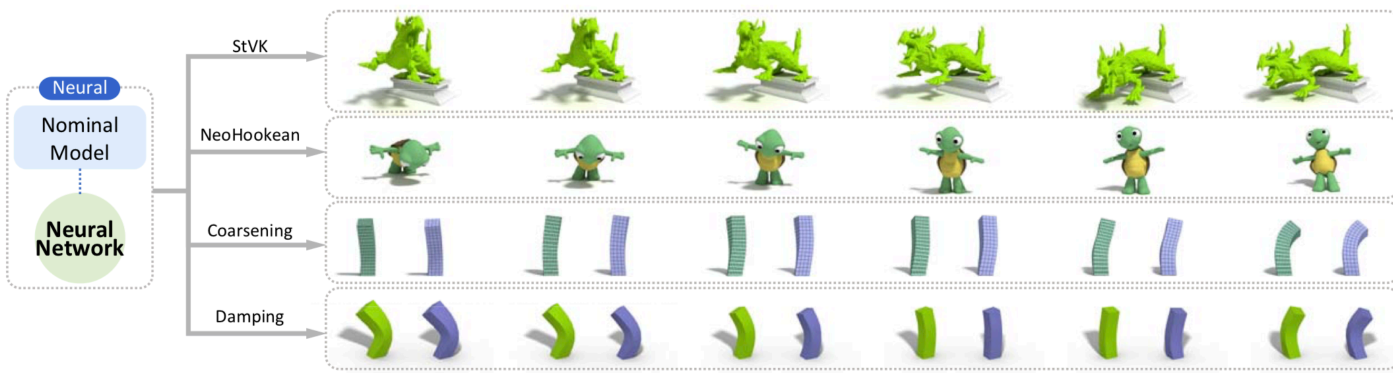
Learn how to *update* the latent state of a system

Related Work

DeepWarp: DNN-based Nonlinear Deformation
Luo et al. 2018



Neural Material: Learning Elastic Constitutive Material and Damping Models from Sparse Data
Wang et al. 2018



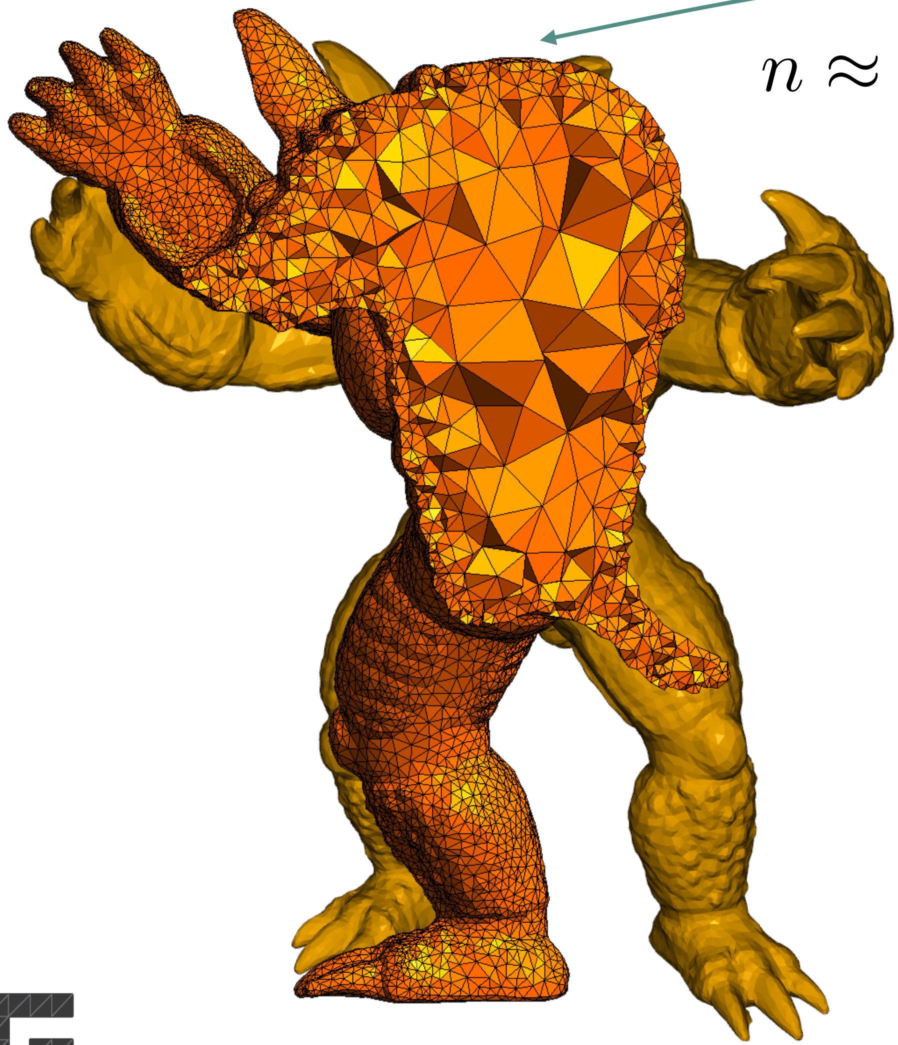
Learn *correction* to cheap simulation

Our Approach

Build on the vast literature of **Model Reduction**

Simulate in nonlinear latent space using the **true equations of motion**

First, why is it slow?



$n \approx 40,000$

$$\mathbf{u} = \underbrace{\begin{bmatrix} \mathbf{p}_1 \\ \mathbf{p}_2 \\ \vdots \\ \mathbf{p}_n \end{bmatrix}}_{\text{Vertex Displacements}}$$

Solving large differential equation

$$\mathbf{F}(\mathbf{u}) = \mathbf{M}\ddot{\mathbf{u}}$$

Solver

Fast and stable solution: Implicit Euler as a minimization problem

$$\overbrace{\mathbf{u}_{n+1}}^{\text{New configuration}} = \operatorname*{argmin}_{\mathbf{u}} \underbrace{V(\mathbf{u})}_{\text{Elastic Potential}} + \overbrace{I(\mathbf{u}, \underbrace{\mathbf{u}_n, \dot{\mathbf{u}}_n}_{\text{Previous State}})}^{\text{Inertia Term}}$$

$$\mathbf{u}_{n+1} = \operatorname*{argmin}_{\mathbf{u}} E(\mathbf{u})$$ Objective Function

Solve using pre-conditioned quasi-newton solver like L-BFGS

Existing Work: Model Reduction

$$\mathbf{u} \in \mathbb{R}^{40{,}000}$$

High Dimensional System

$$\mathbf{q} = \mathbf{U}^T \mathbf{u}$$

$$\mathbf{u} = \mathbf{U}\mathbf{q}$$

Low Dimensional System

$$\underbrace{\mathbf{q} \in \mathbb{R}^{\sim 60}}_{\text{Reduced Coordinates}}$$

Model Reduction

Replace high-dimensional problem with low-dimensional

$$\mathbf{u}_{n+1} = \operatorname*{argmin}_{\mathbf{u}} E(\mathbf{u})$$

Big

Becomes

$$\mathbf{q}_{n+1} = \operatorname*{argmin}_{\mathbf{q}} E(\mathbf{U}\mathbf{q})$$

Small

Static Solve Example

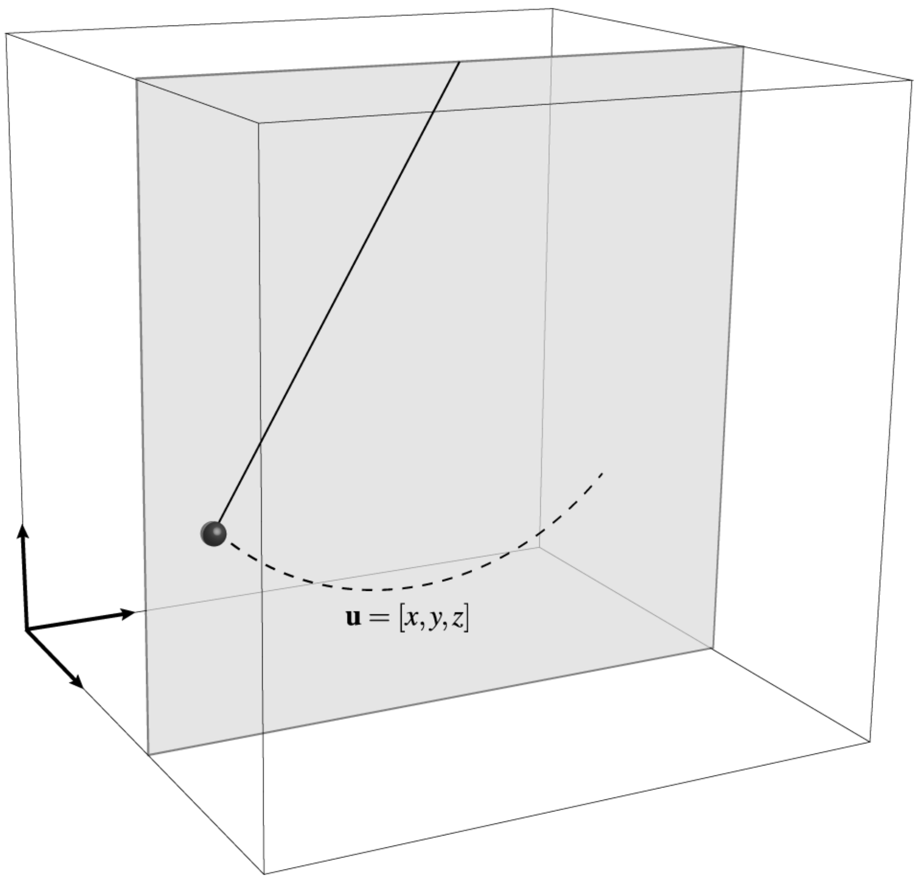
Full

Linear

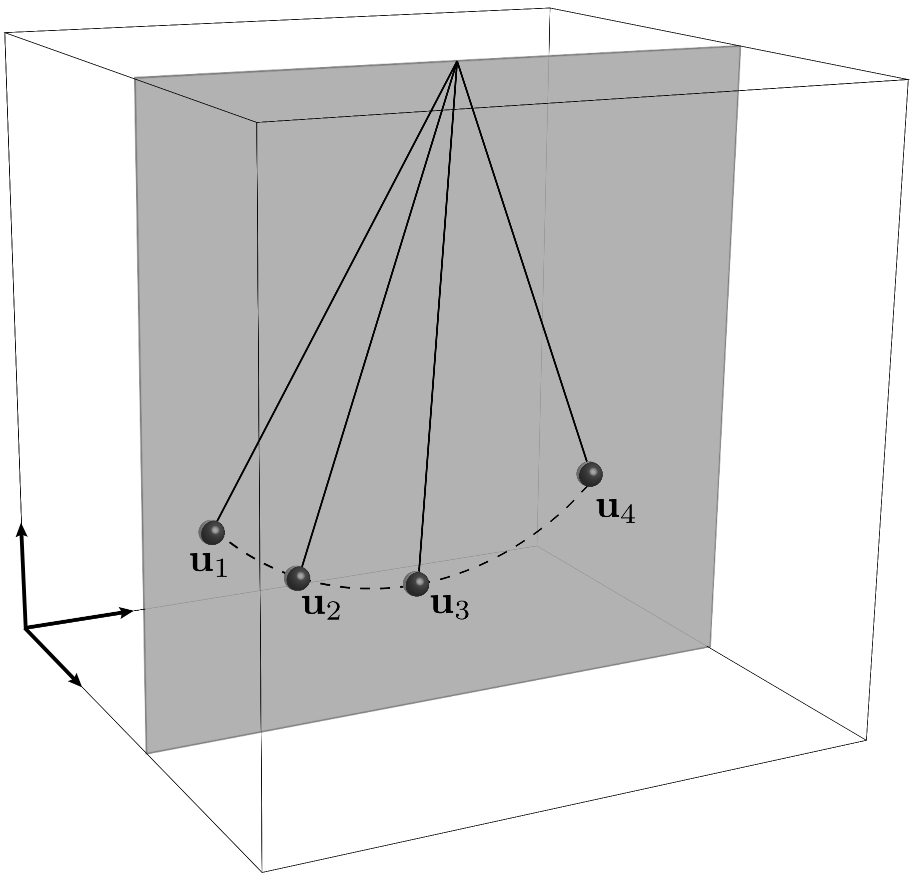
Iterations
0

Where does $\mathbf{U}$ come from?

Model Reduction - Example



Model Reduction - Example



Collect Snapshots

$$\mathbf{P} = [\mathbf{u}_1 \mathbf{u}_2 \mathbf{u}_3 \mathbf{u}_4]$$

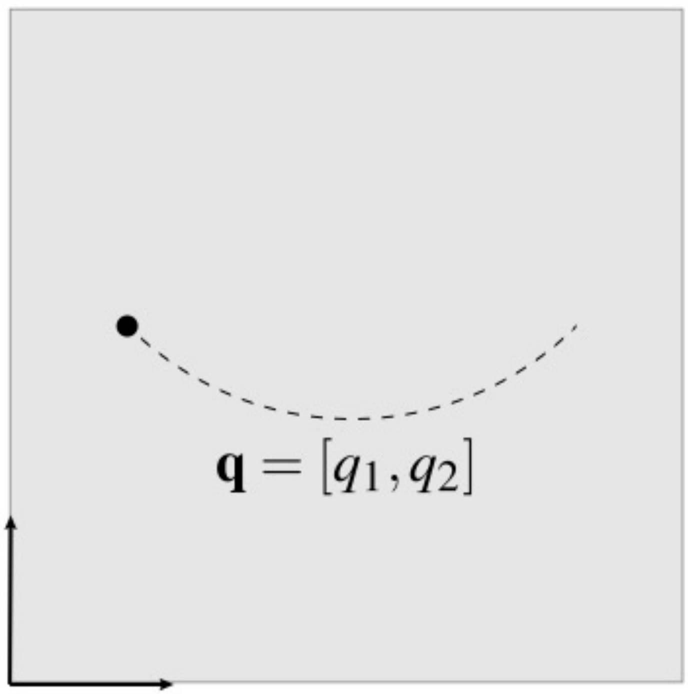
Perform PCA (via SVD)

$$\mathbf{P} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$$

Keep *k* largest eigen values

$$\mathbf{U} := \mathbf{U}_{1:k}$$

Model Reduction - Example



$$\mathbf{u} = \mathbf{U}\mathbf{q}$$

Collect Snapshots

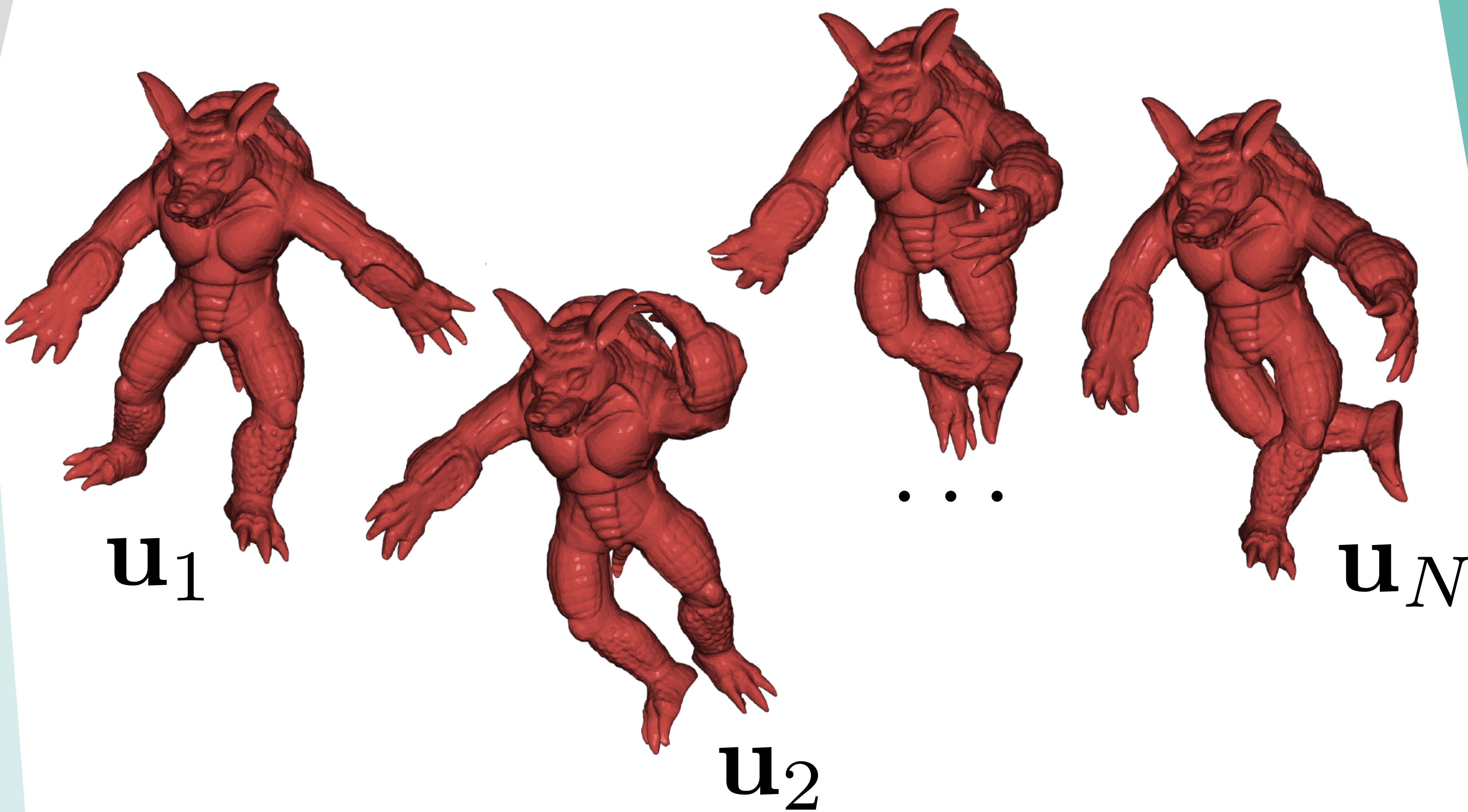
$$\mathbf{P} = [\mathbf{u}_1 \mathbf{u}_2 \mathbf{u}_3 \mathbf{u}_4]$$

Perform PCA (via SVD)

$$\mathbf{P} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$$

Keep *k* largest eigen values

$$\mathbf{U} := \mathbf{U}_{1:k}$$



$$\mathbf{U} = \mathrm{PCA}([\mathbf{u}_1 \dots \mathbf{u}_N], k)$$

$k = 62$

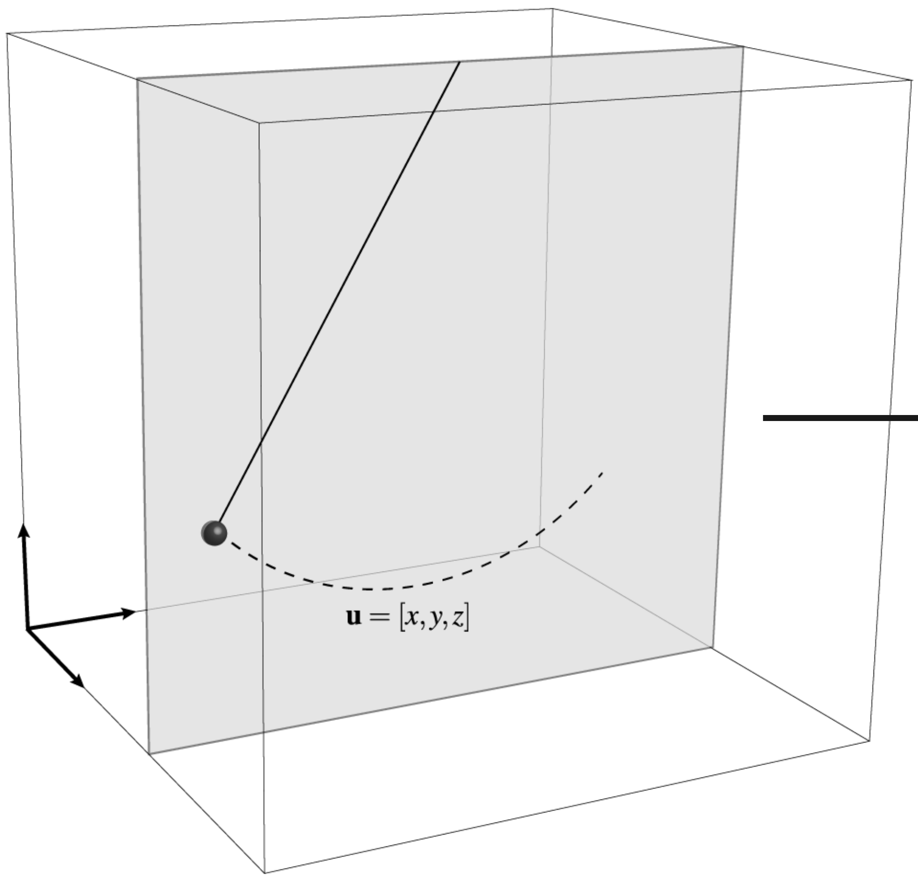
Limits to Linear Reduction

Full Space

Limits to Linear Reduction

6 Degrees of Freedom

Can we do better?



$$\mathbf{u} = \mathrm{nonlinear}(\mathbf{z})$$

$$|\mathbf{q}| > |\mathbf{z}|$$

Linear: 6 DOF

Nonlinear: 6 DOF

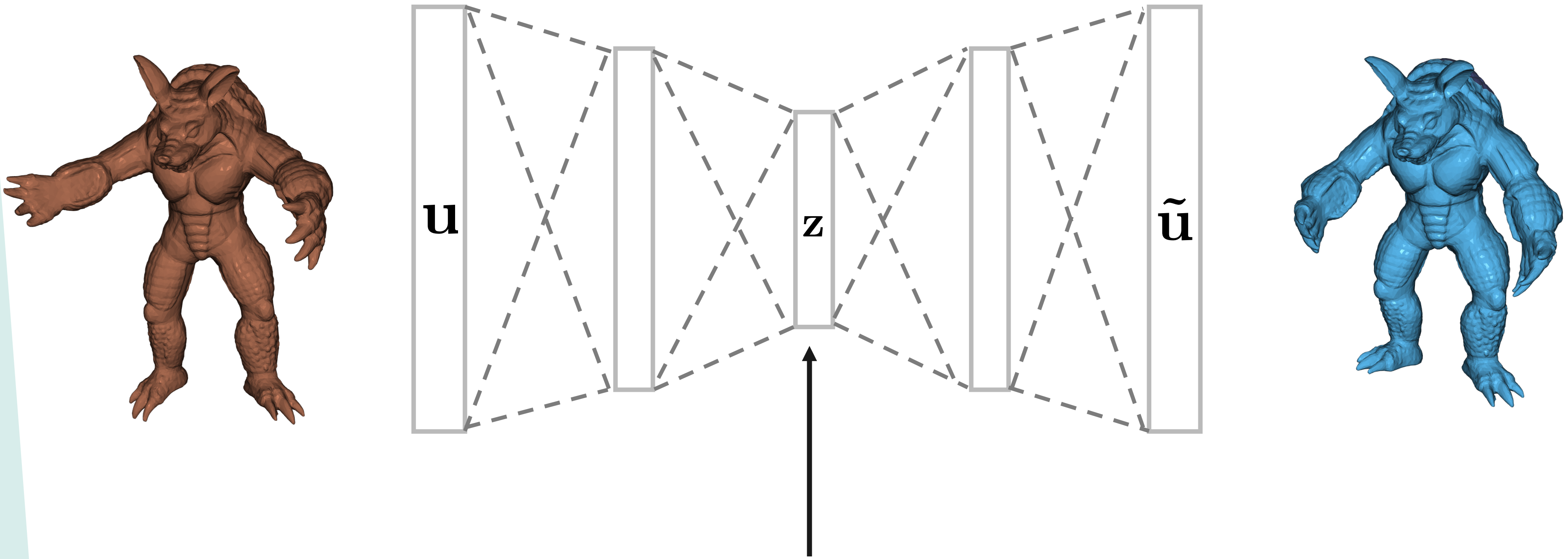
Our Contribution

Many possibilities for $\mathrm{nonlinear}(\mathbf{z})$

We use a neural network trained as an ***Autoencoder*** to create a unique $\mathrm{nonlinear}(\mathbf{z})$ for a given scenario

Autoencoders

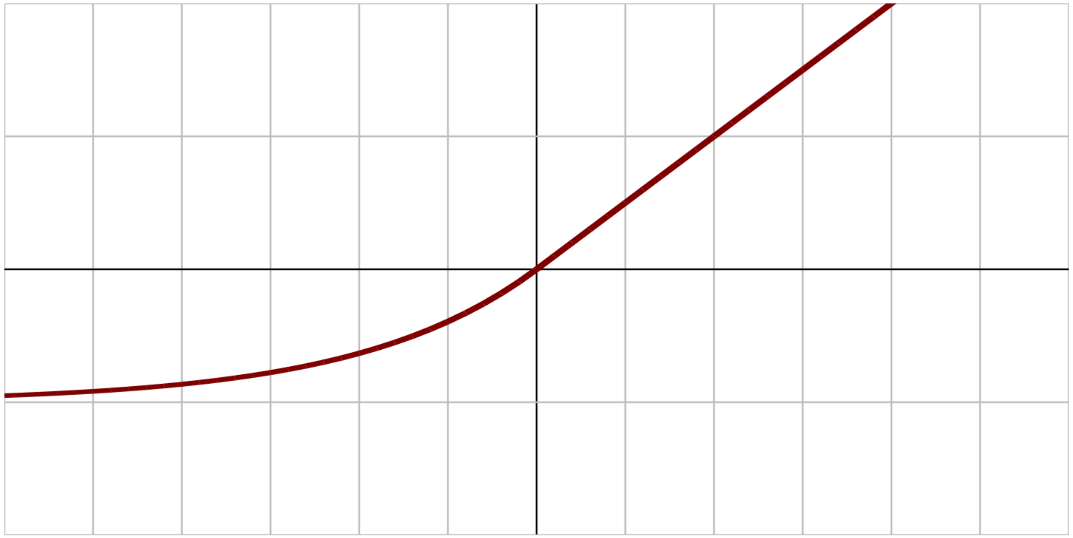
$$\mathrm{encode}(\mathbf{u}) = \mathbf{z} \qquad \mathrm{decode}(\mathbf{z}) = \tilde{\mathbf{u}}$$



Encoded “Latent” vector $\mathbf{z}$

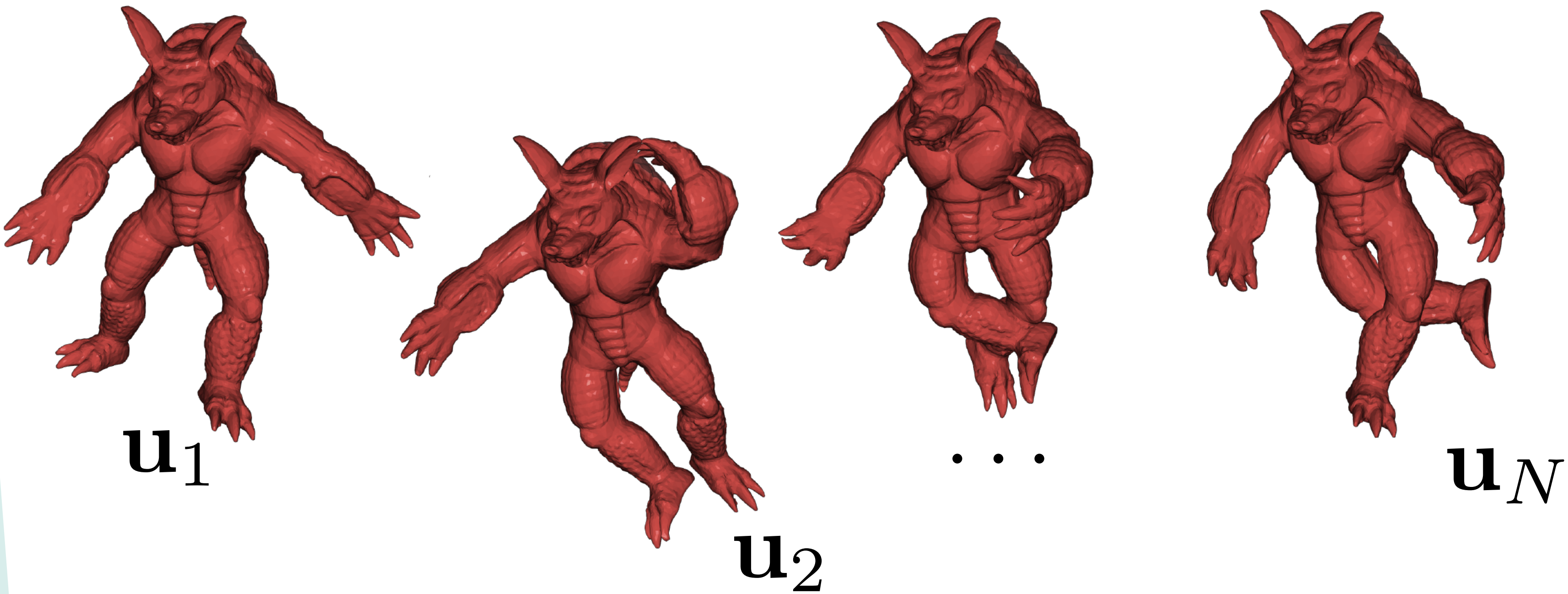
Decode is a sequence of function applications

$$\mathrm{decode}_k(\mathbf{z}) = \mathrm{activation}(\mathbf{z}^T \mathbf{W}_\theta + \mathbf{b})$$



$\mathrm{activation}(x)$

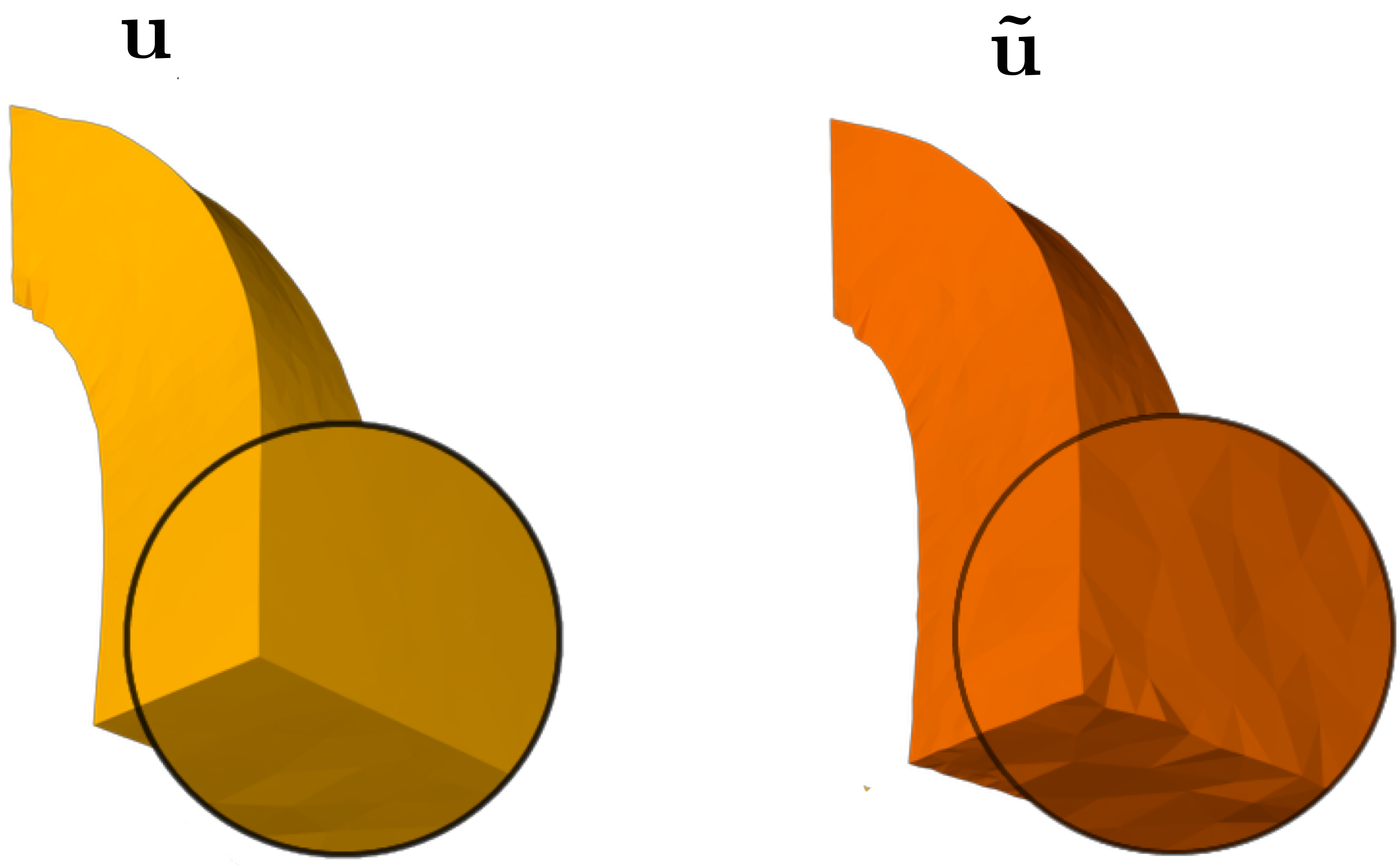
Optimize the weights $\mathbf{W}_\theta$ by automatic differentiation and gradient descent



$$\theta^* = \operatorname*{argmin}_{\theta} \sum_{i=1}^{N} \| \mathrm{decode}(\mathrm{encode}(\mathbf{u}_i)) - \mathbf{u}_i \|_2^2$$

Minimize Mean Squared Error with ADAM

Training directly on full mesh results in long training times and poor approximation



Previous work: last layer of network is **linear**, so just initialize it with PCA

We observe you can train directly in the PCA space and get equivalent results.

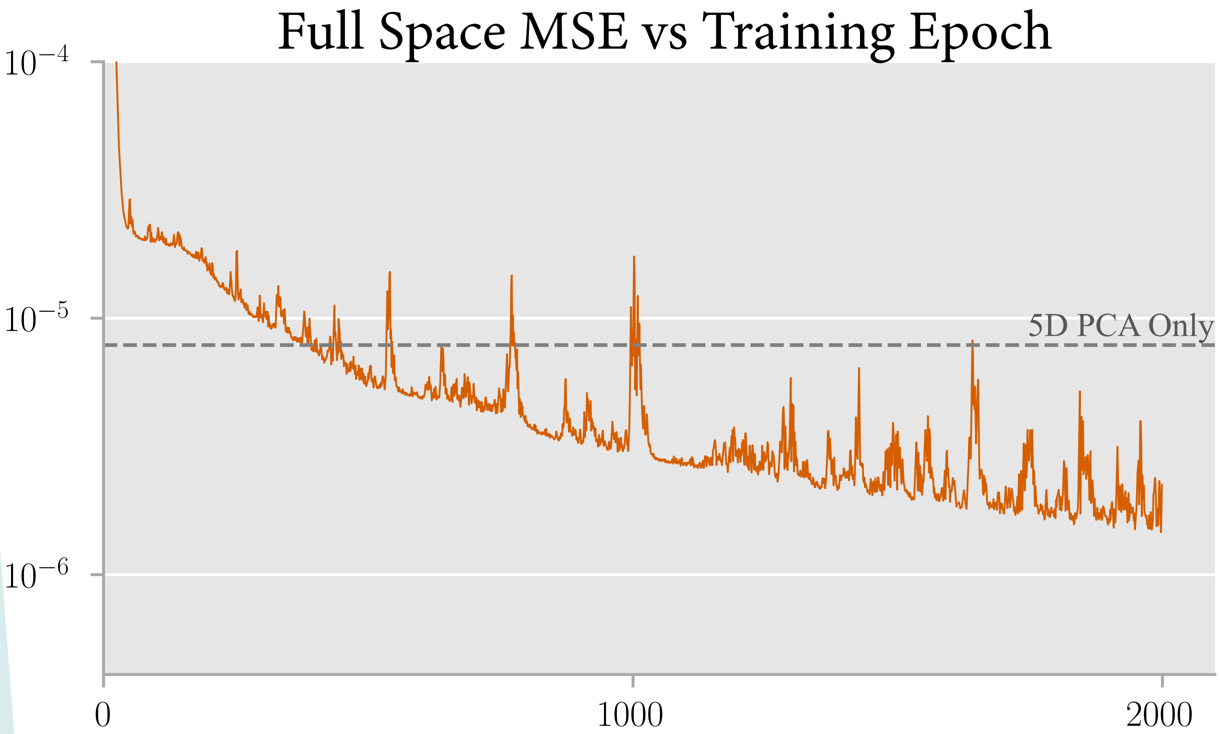
Our Training Pipeline

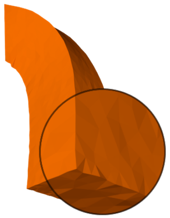
$$\mathbf{U} = \mathrm{PCA}([\mathbf{u}_1 \ldots \mathbf{u}_N], k)$$ Do PCA on snapshots

$$[\mathbf{q}_1 \ldots \mathbf{q}_N] = \mathbf{U}^T [\mathbf{u}_1 \ldots \mathbf{u}_N]$$ Project training samples

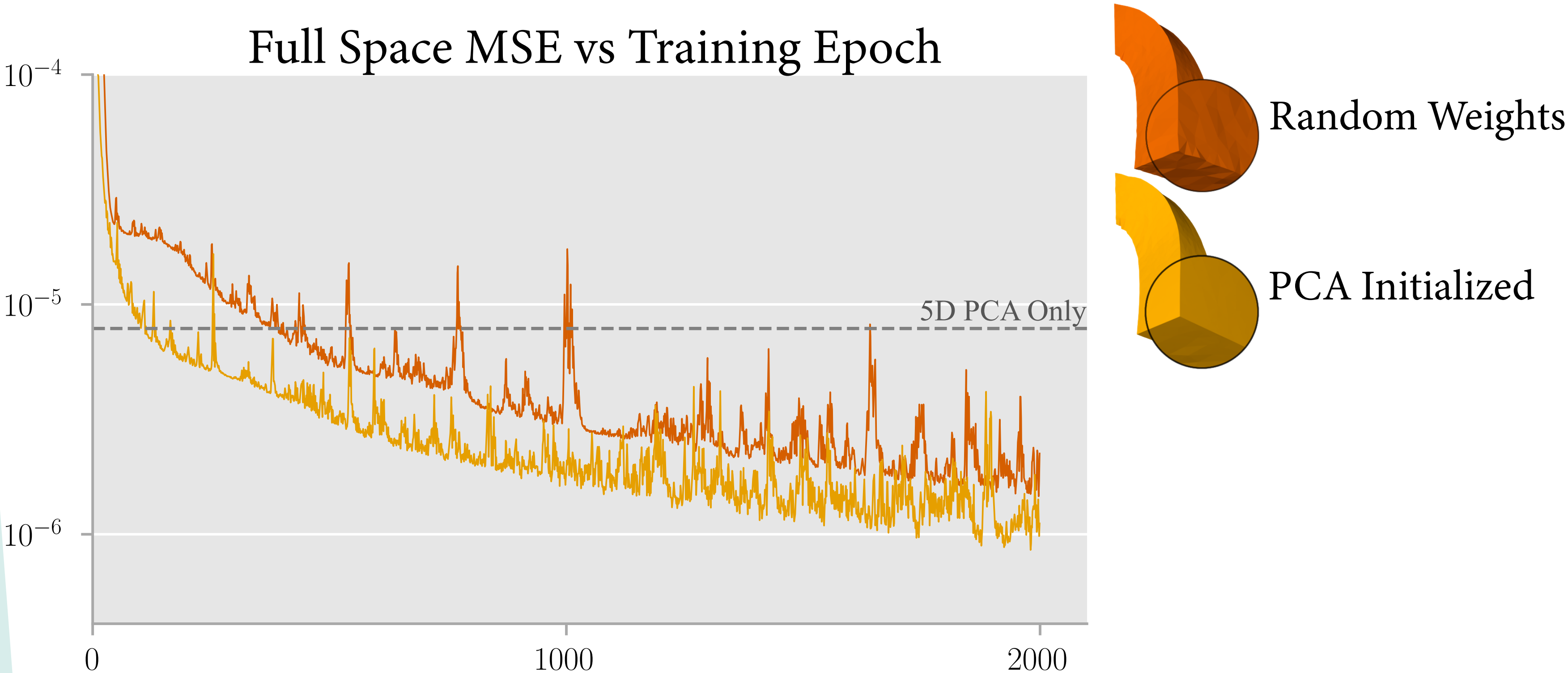
Train autoencoder to reduce the PCA coefficients further

$$\theta^* = \underset{\theta}{\operatorname{argmin}} \sum_{i=1}^{N} \| \mathrm{decode}(\mathrm{encode}(\mathbf{q}_i)) - \mathbf{q}_i \|_2^2$$

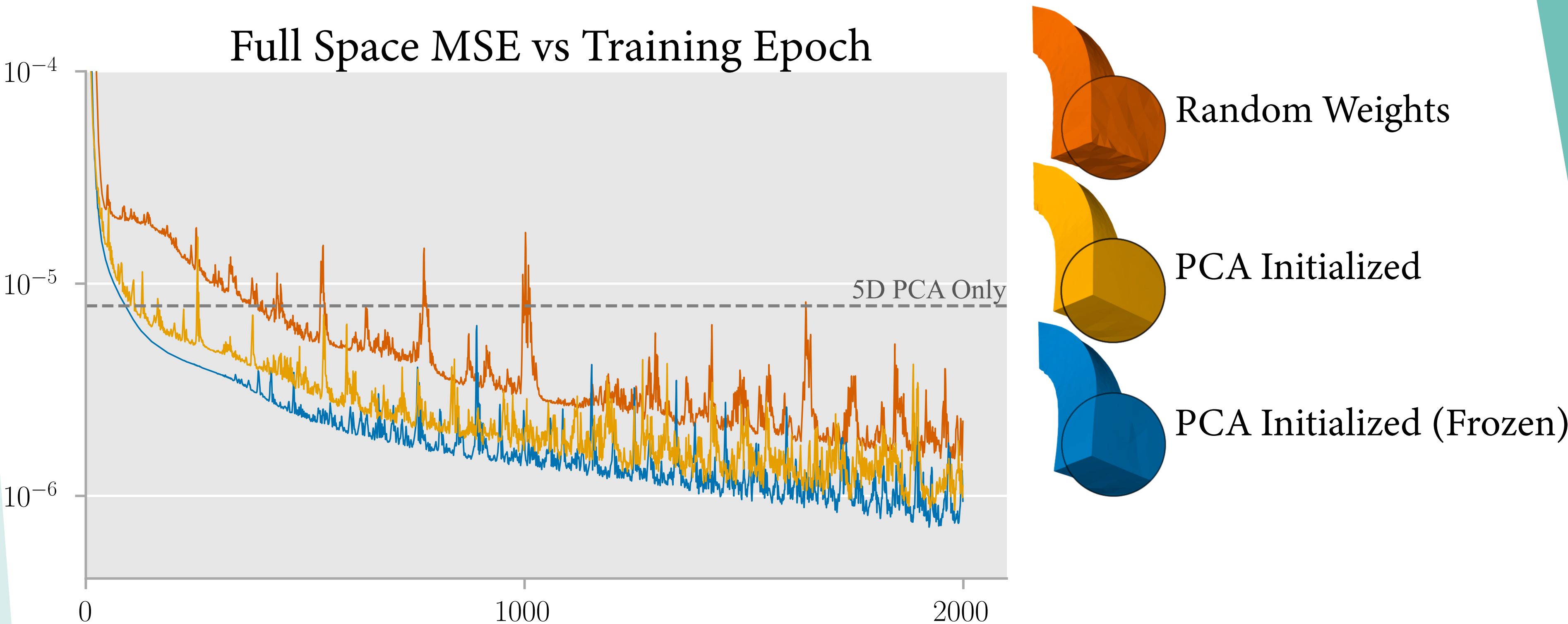




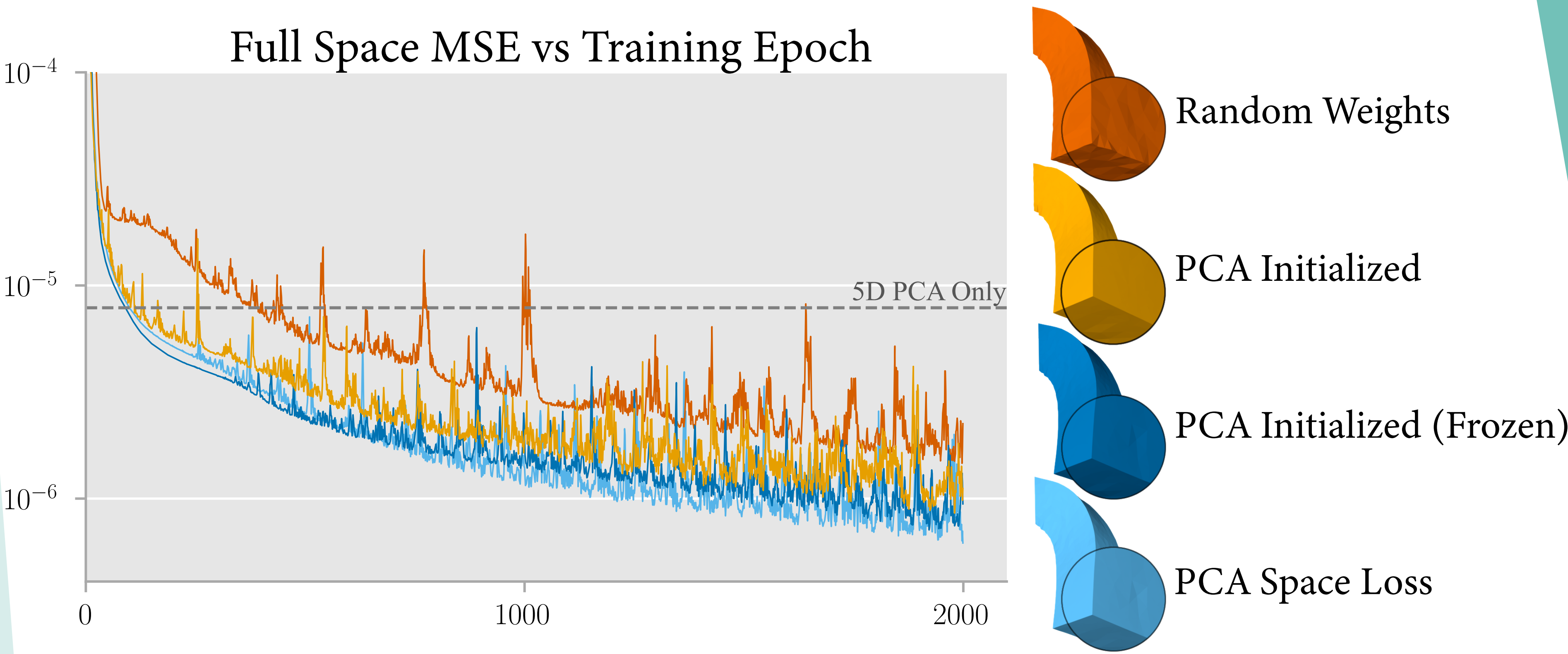
Random Weights

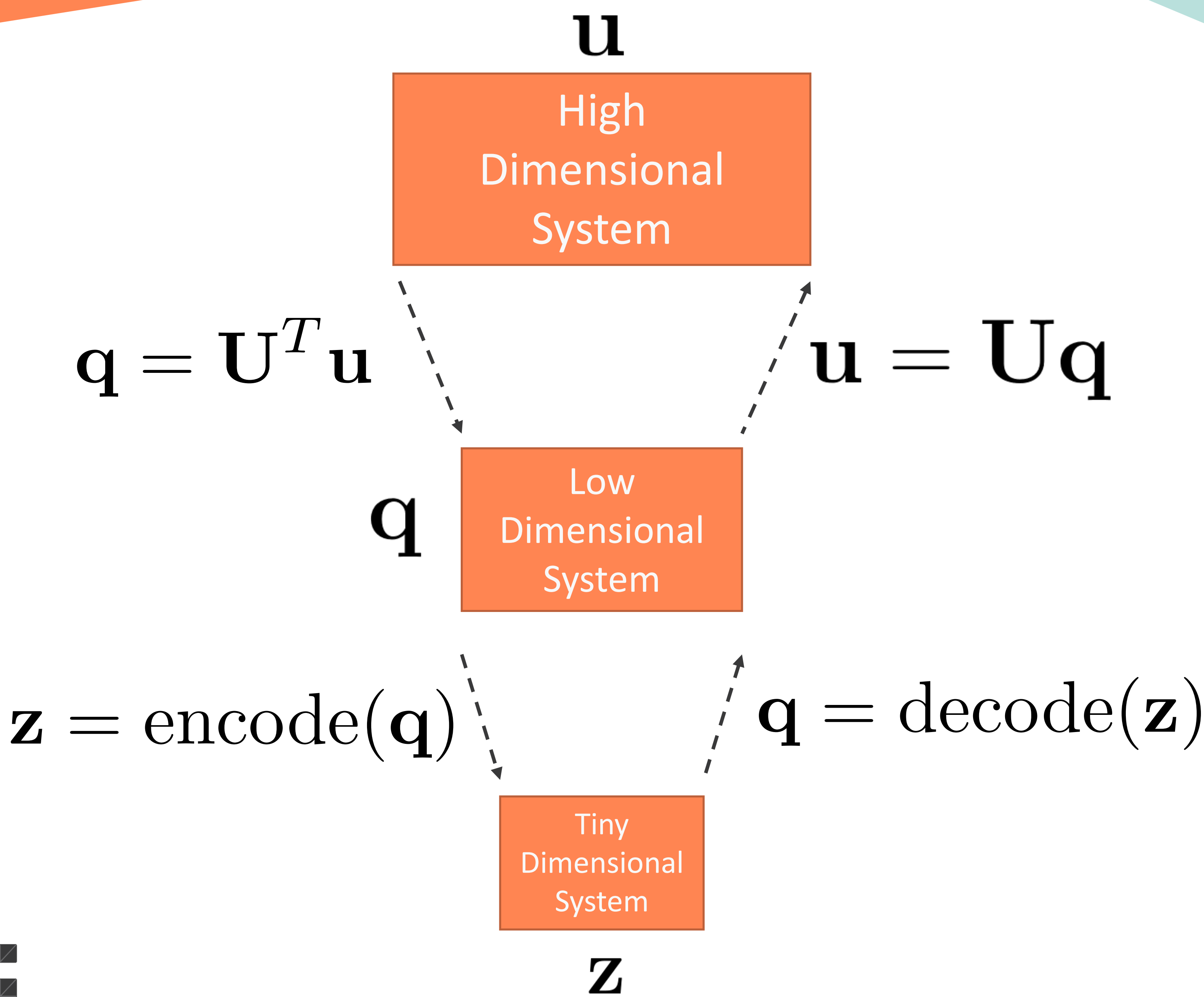


Full Space MSE vs Training Epoch
10^{-4}
10^{-5}
10^{-6}
5D PCA Only
0
1000
2000
Random Weights
PCA Initialized

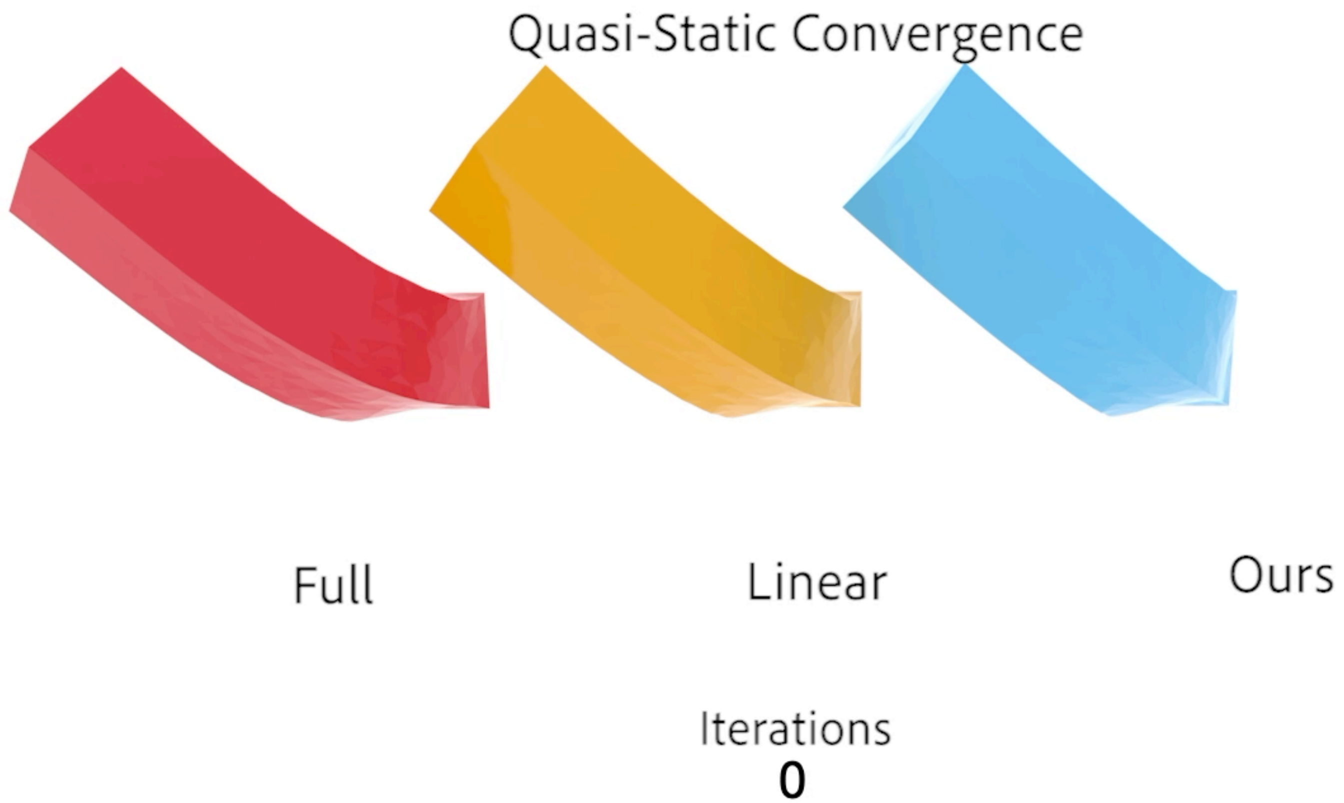


Full Space MSE vs Training Epoch
10^{-4}
10^{-5}
10^{-6}
5D PCA Only
0
1000
2000
Random Weights
PCA Initialized
PCA Initialized (Frozen)


Full Space MSE vs Training Epoch
10^{-4}
10^{-5}
10^{-6}
5D PCA Only
0
1000
2000
Random Weights
PCA Initialized
PCA Initialized (Frozen)
PCA Space Loss


$\mathbf{u}$
High Dimensional System
$\mathbf{q} = \mathbf{U}^T \mathbf{u}$
$\mathbf{u} = \mathbf{U}\mathbf{q}$
$\mathbf{q}$
Low Dimensional System
$\mathbf{z} = \mathrm{encode}(\mathbf{q})$
$\mathbf{q} = \mathrm{decode}(\mathbf{z})$
Tiny Dimensional System
$\mathbf{z}$

Convergence Rate



Latent Space Dynamics

$$\mathbf{u}_{n+1} = \underset{\mathbf{u}}{\operatorname{argmin}}\, E(\mathbf{u})$$

Big

Becomes

$$\mathbf{z}_{n+1} = \underset{\mathbf{z}}{\operatorname{argmin}}\, E(\mathbf{U}\, \mathrm{decode}(\mathbf{z}))$$

Small(er)

How do we make it fast?

Recall our objective function:

$$E(\mathbf{z}) = \underbrace{V(\mathbf{U}\,\mathrm{decode}(\mathbf{z}))}_{\text{Elastic Potential}} + \overbrace{I(\mathbf{U}\,\mathrm{decode}(\mathbf{z}), \mathbf{u}_n, \dot{\mathbf{u}}_n)}^{\text{Inertia Term}}$$

$$I = \frac{1}{2h^2} \left\| \mathbf{u} - \mathbf{u}_n - \dot{\mathbf{u}}_n h \right\|_{\mathbf{M}}^2$$

$$I = \frac{1}{2h^2} \left\| \mathbf{U}\, \mathrm{decode}(\mathbf{z}) - \mathbf{u}_n - \dot{\mathbf{u}}_n h \right\|_{\mathbf{M}}^2$$

Precompute $\mathbf{U}^T\mathbf{M}\mathbf{U}$ and only partially decode

$$I = \frac{1}{2h^2} \left\| \underbrace{\mathrm{decode}(\mathbf{z})}_{\text{Save as } \mathbf{q}_n \text{ for next timestep}} - \mathbf{q}_n - \dot{\mathbf{q}}_n h \right\|_{\mathbf{U}^T\mathbf{M}\mathbf{U}}^2$$

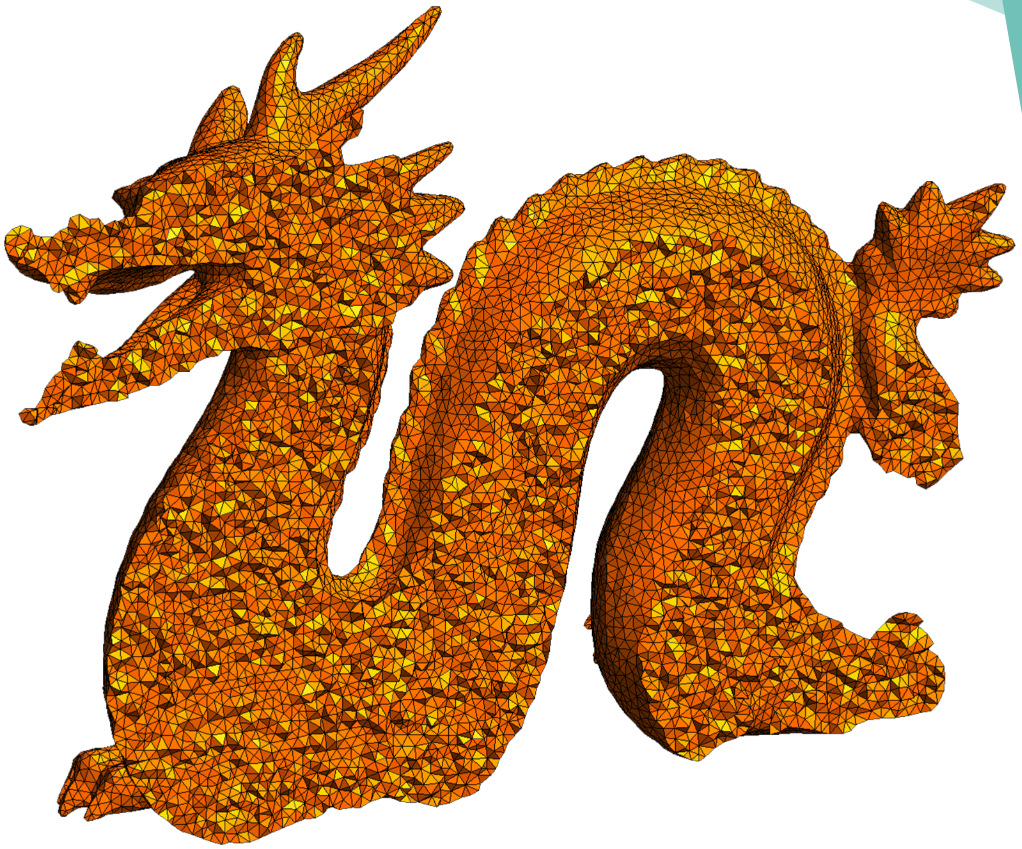
How do we make it fast?

Recall our objective function:

$$E(\mathbf{z}) = \underbrace{V(\mathbf{U}\,\mathrm{decode}(\mathbf{z}))}_{\text{Elastic Potential}} + \overbrace{I(\mathbf{U}\,\mathrm{decode}(\mathbf{z}), \mathbf{u}_n, \dot{\mathbf{u}}_n)}^{\text{Inertia Term}}$$

Cubature

$$V(\mathbf{u}) = \sum_{i=1}^{\#\,\mathrm{Tets}} V_i(\mathbf{u})$$

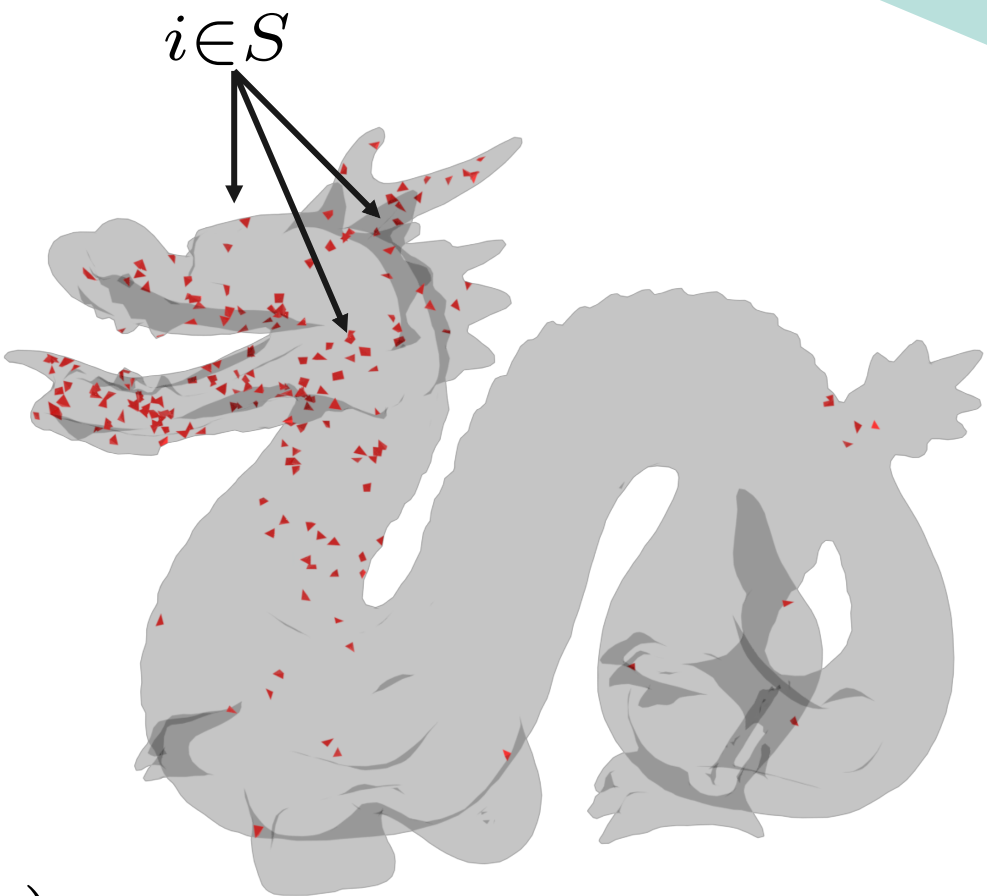


Cubature

$$V(\mathbf{u}) = \sum_{i=1}^{\#\,\mathrm{Tets}} V_i(\mathbf{u})$$

Approximate with weighted sum

$$V(\mathbf{u}) \approx \sum_{i \in S} w_i V_i(\mathbf{u})$$



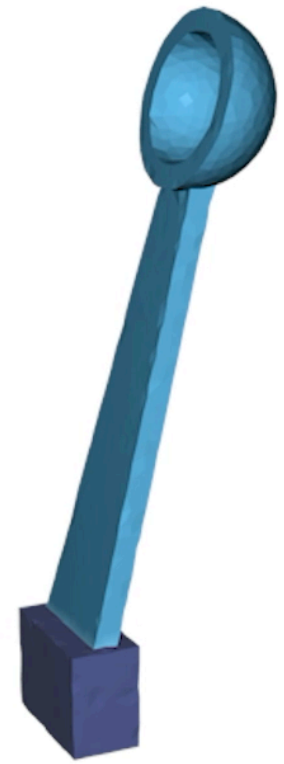
Use [An et al. 08]'s "Optimized Cubature"

Only fully-decode elements we need

Results: Stability

Single Cubature Point

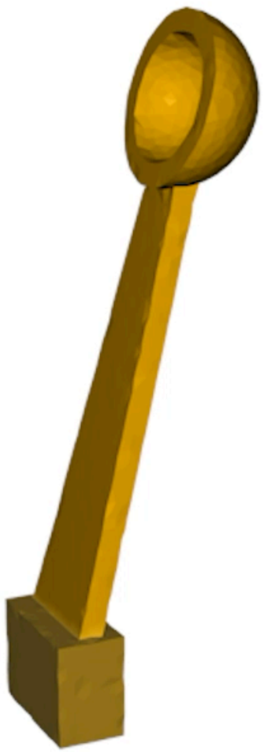
Results: Stability



2 dof Autoencoder subspace (ours)

SCREEN CAPTURE

Results: Stability



6 dof linear subspace

SCREEN CAPTURE

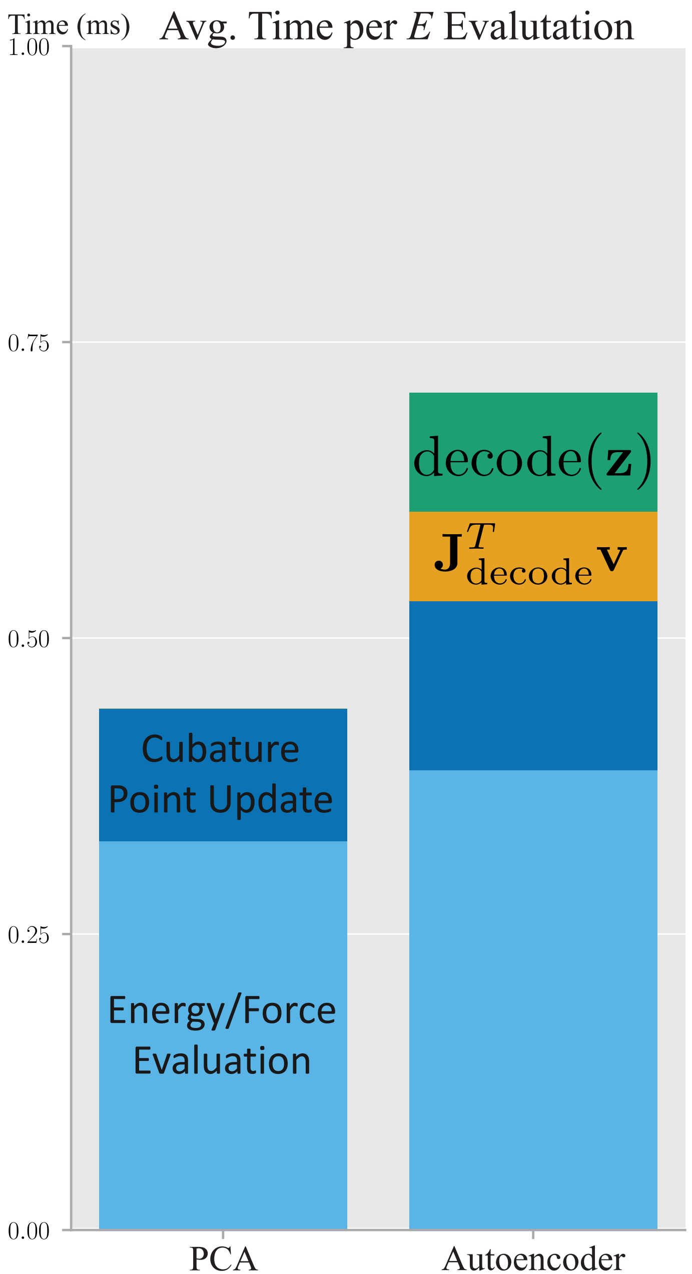
And finally $\nabla E(\mathbf{z})$

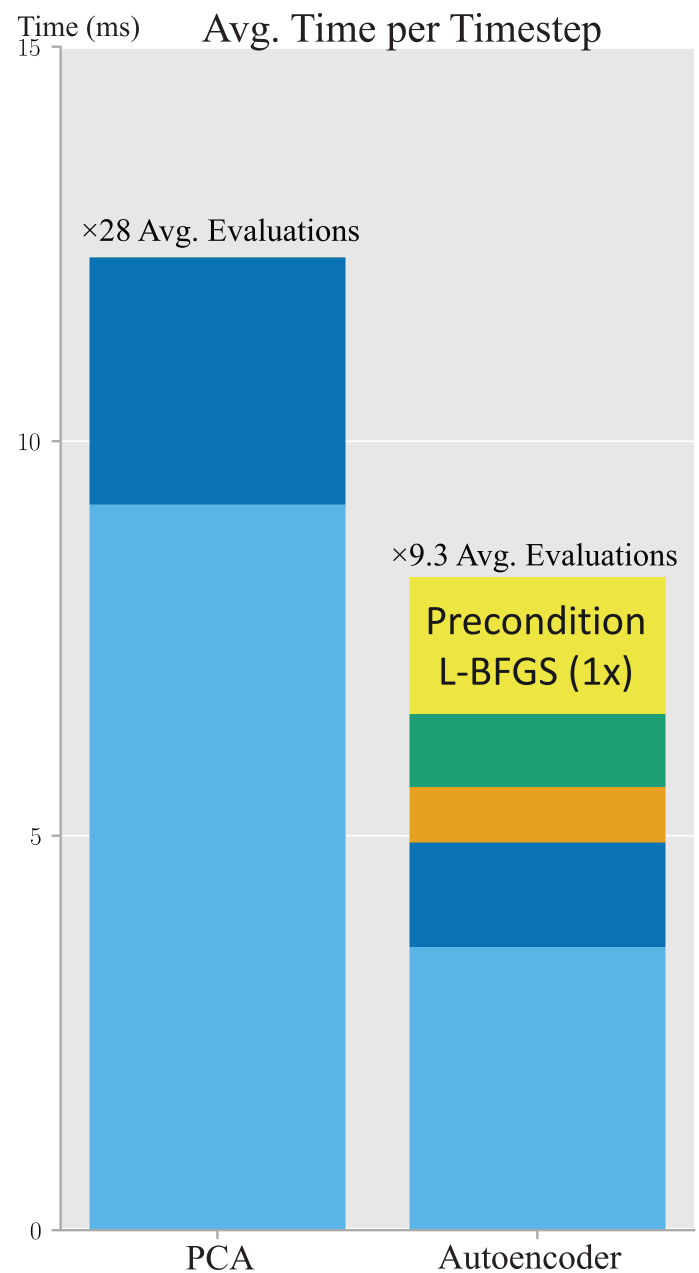
Only the gradient of our objective is required since using a quasi-Newton scheme

$$\nabla E(\mathbf{z}) = \mathbf{J}_{\text{decode}}^{T} \frac{\partial E}{\partial \mathbf{q}}$$

$\mathbf{J}_{\text{decode}}^{T}$ Non-constant Jacobian matrix of our autoencoder

Automatic differentiation allows us to evaluate $\mathbf{J}_{\text{decode}}^{T}\mathbf{v}$ with equivalent complexity as a single forward evaluation


Avg. Time per E Evaluation
Time (ms)
1.00
0.75
0.50
0.25
0.00
PCA
Autoencoder
Cubature Point Update
Energy/Force Evaluation
decode($\mathbf{z}$)
$\mathbf{J}_{\text{decode}}^T\mathbf{v}$


Avg. Time per Timestep
Time (ms)
15
10
5
0
PCA
Autoencoder
×28 Avg. Evaluations
×9.3 Avg. Evaluations
Precondition L-BFGS (1x)

Results: Performance

PCA - 62 dof

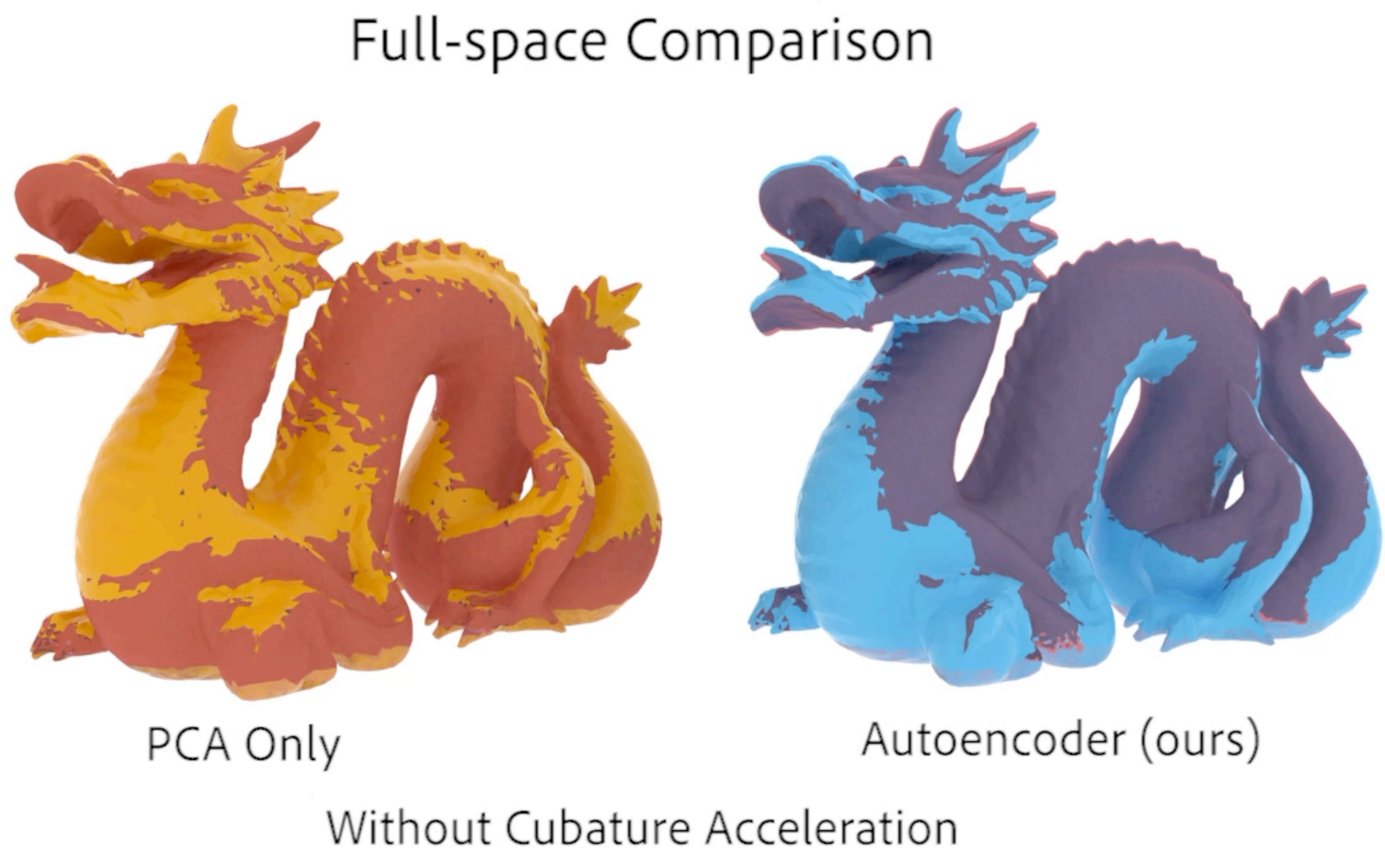
95Hz

Ours - 20 dof

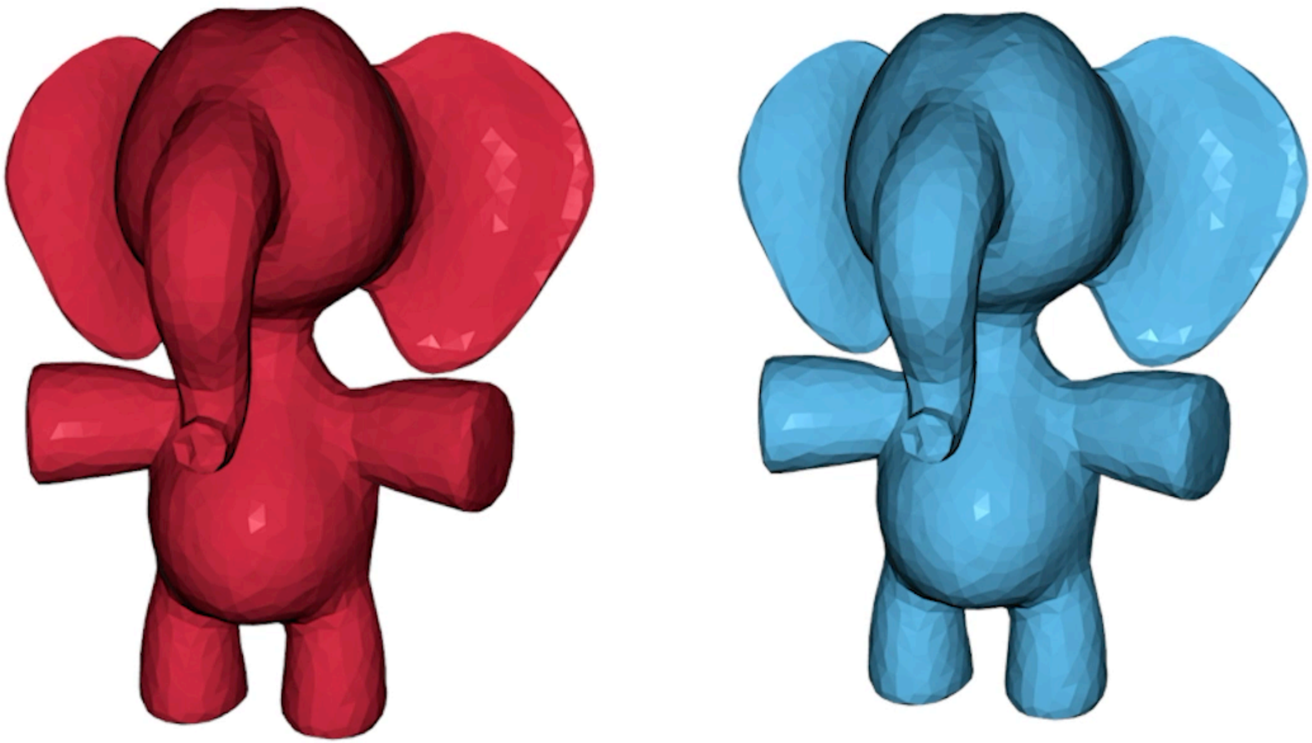
159 Hz

SCREEN CAPTURE

Results: Accuracy



Limitations



Summary

- Autoencoders can reduce system dimensionality further than linear alone.
- This reduction allows faster simulation
- Results are robust, even for small spaces and few cubature points.

Future Work

- Can we incorporate cubature into our method?

Future Work

- Can we incorporate cubature into our method?
- One network, many shapes?
- Automatic training data generation?

Acknowledgements

- NSERC Discovery Grants (RGPIN–2017–05235, RGPIN-2017-05524, RGPAS–2017–507938, RGPAS-2017-507909)
- Connaught Funds (NR2016–17)
- Canada Research Chairs Program
- Gifts from the Fields Institute, Adobe Systems Inc, Autodesk Inc, and MESH Inc.
- Sarah Kushner for help with figure creation

Thank you for listening!

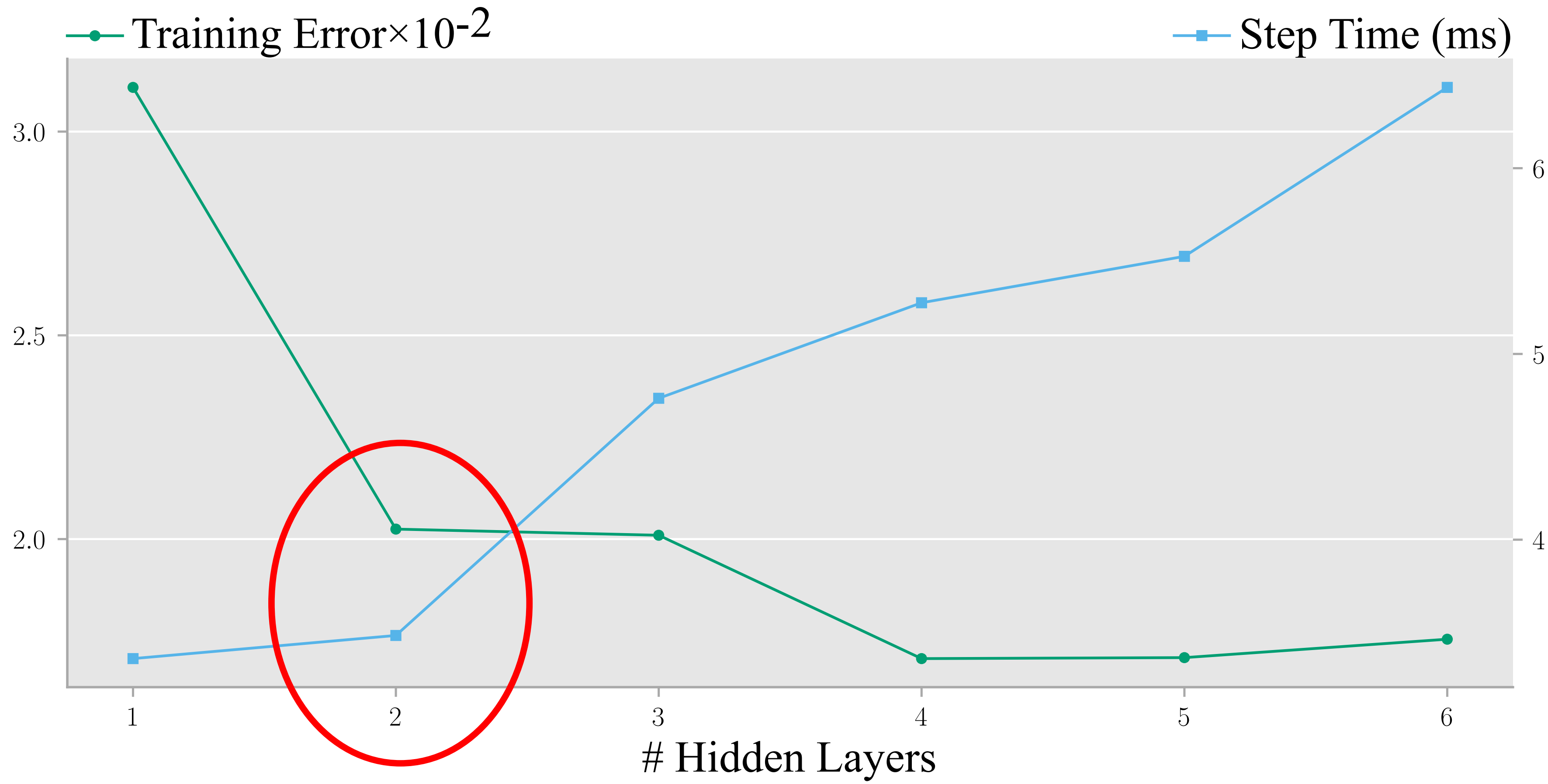
Latent-space Dynamics for Reduced Deformable Simulation

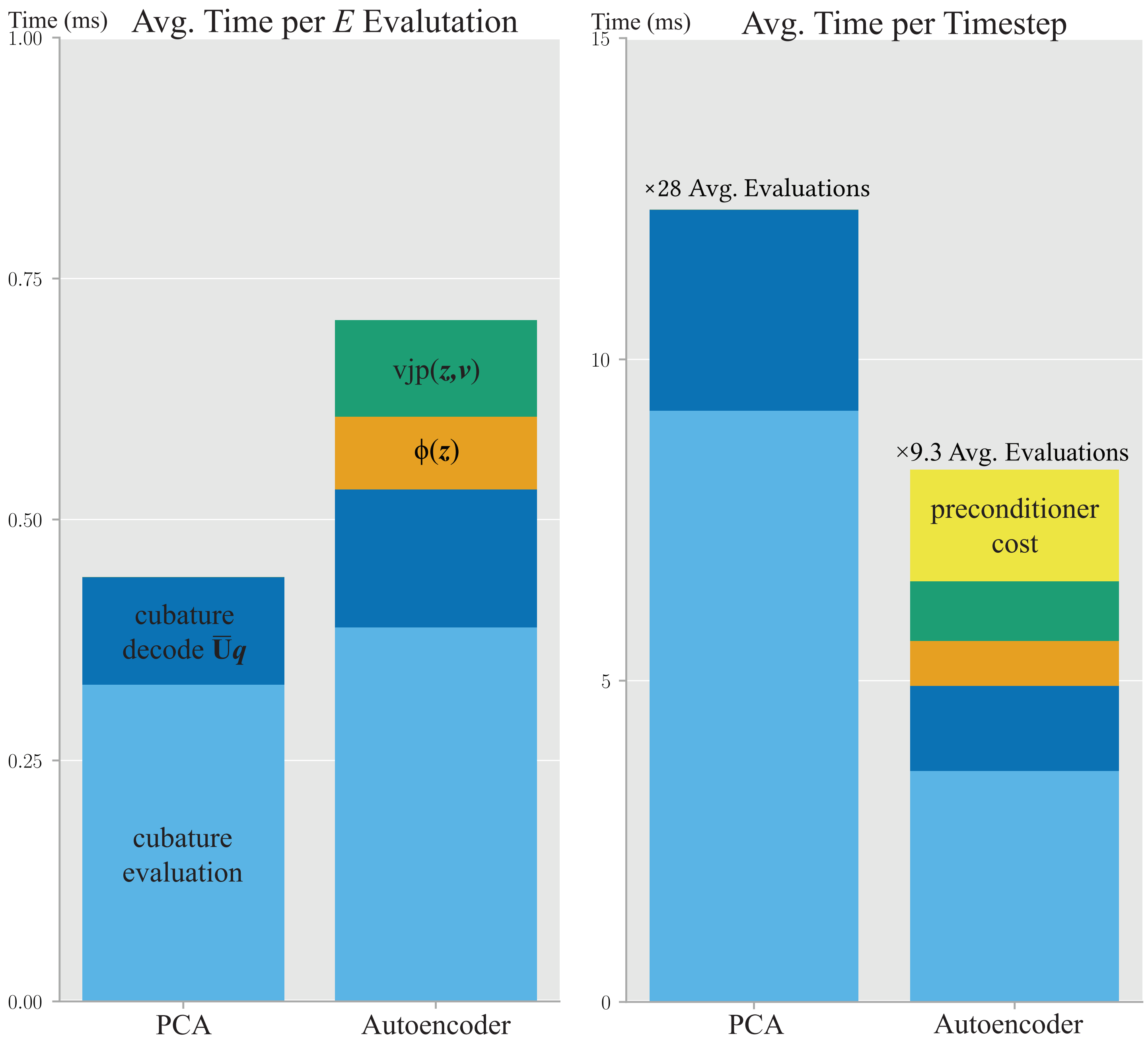
Lawson Fulton, Vismay Modi, David Duvenaud, David I.W. Levin, Alec Jacobson

University of Toronto

Contact: lawson@cs.toronto.edu

Project Page: bit.ly/2V3U9Kv


Training Error×10-2
Step Time (ms)
3.0
2.5
2.0
6
5
4
1
2
3
4
5
6
Hidden Layers



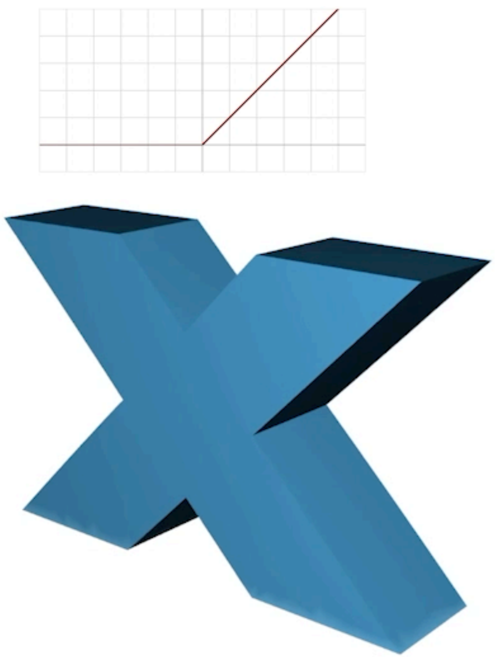
Time (ms)
Avg. Time per E Evalutation
1.00
0.75
0.50
0.25
0.00
vjp(z,v)
$\phi(z)$
cubature decode $\overline{\mathbf{U}}\boldsymbol{q}$
cubature evaluation
PCA
Autoencoder
Time (ms)
Avg. Time per Timestep
15
10
5
0
×28 Avg. Evaluations
×9.3 Avg. Evaluations
preconditioner cost
PCA
Autoencoder

Training Data

Training data generation

SCREEN CAPTURE

Choice of Activation



ReLU Activations

SCREEN CAPTURE

Preconditioner

$$\tilde{\mathbf{H}} = \mathbf{J}_{\mathbf{z}_n}^T \tilde{\mathbf{K}}_0 \mathbf{J}_{\mathbf{z}_n}$$

$$\tilde{\mathbf{K}}_0 = U^T \mathbf{K}_0 U$$

$$\mathbf{K}_0 = \frac{\partial^2 \mathbf{V}(\mathbf{0})}{\partial \mathbf{u}^2}$$