Learning to Simulate Complex Physics with Graph Networks

Alvaro Sanchez-Gonzalez, Jonathan Godwin, Tobias Pfaff, Rex Ying, Jure Leskovec, Peter W. Battaglia (ICML 2020)

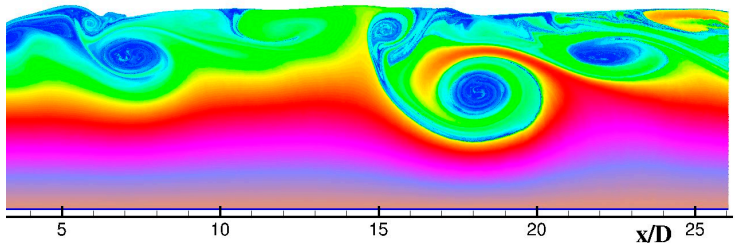
Presented by: Federico Berto (20204817)
Date: 2021-10-28

Contents

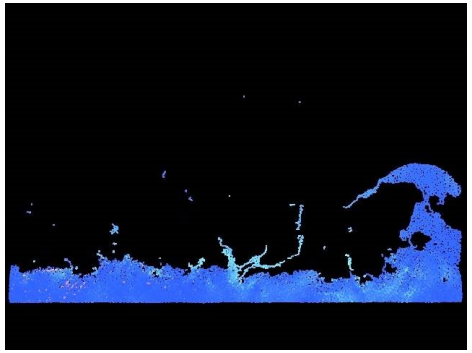
Introduction

Emulating Physics: Particle Simulators

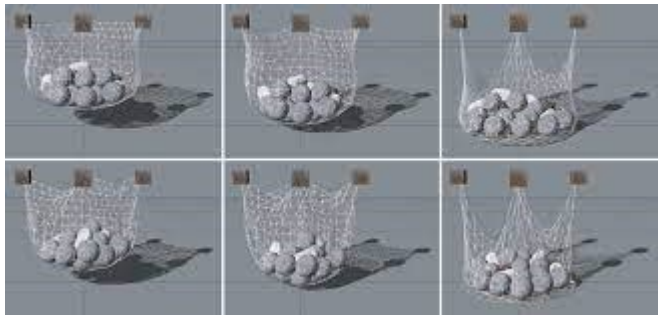
- **Smoothed Particle Hydrodynamics (SPH)**: a meshfree Lagrangian method for simulating mechanics of media such as solids and fluid flows. Good with many fields, but does not deal well with boundary conditions



- **Material Point Method (MPM):** meshfree Lagrangian method which divides the continuum in smaller Lagrangian points called *material points.* Good for dealing with cracks and discontinuities, but more computationally expensive than FEM (Finite Element Method)



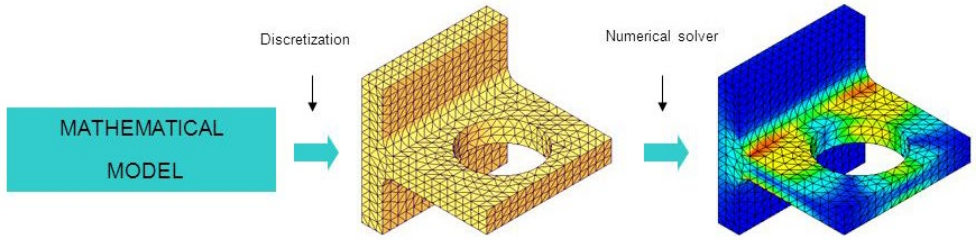
- **Position Based Dynamics (PBD):** meshfree methods which directly works on positions. Works well with constraints and it is modular, but it is not physically accurate



Emulating Physics: Other Examples

- **Finite Element Method (FEM)**: a mesh-based algorithm for solving PDEs. It can handle a variety of problems and geometries, but it gives only approximate solutions and parameter changes can result in fatal errors

- **Other Simulators:** classical (e.g. FVM, ARCSim), deep-learning based (e.g. PINNs) and more



→ *Takeaway 1*: many very well engineered simulators, each one with its pros and cons. No "one-fits-all" type!

→ *Takeaway 2*: science and technology need simulators for many purposes, from medical simulations, risk assessment, heat spread and more. Simulation is fundamental for engineering!

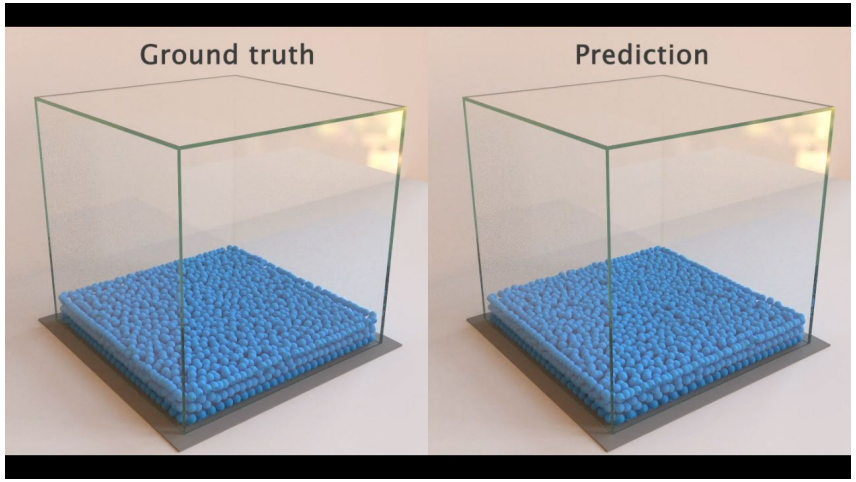
Why learn simulation?

Engineered simulators

- Substantial effort to build
- Substantial resources to run
- Only as accurate as the model
- Not always suitable for solving inverse problems

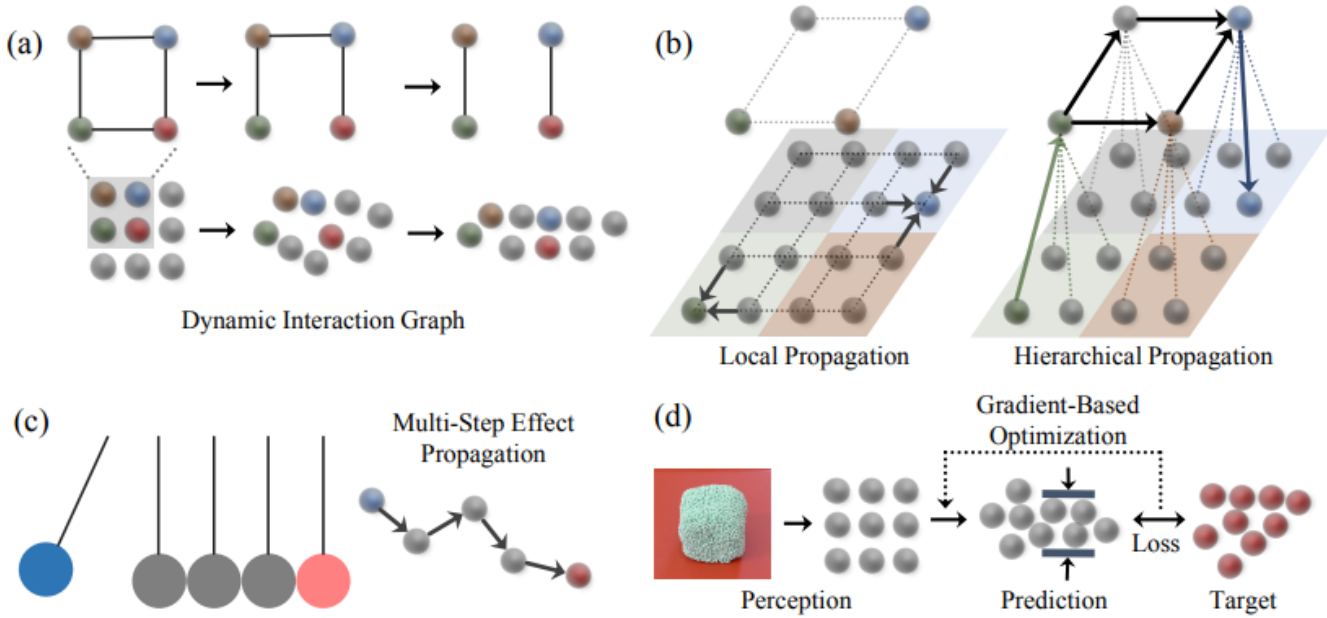
Learned simulators

- Reusable general architectures
- Can be directly optimized for efficiency
- Can be as accurate as the available data
- Gradient-based search for control, inference, etc



Related Work

Dynamic Particle Interaction Networks (DPI)

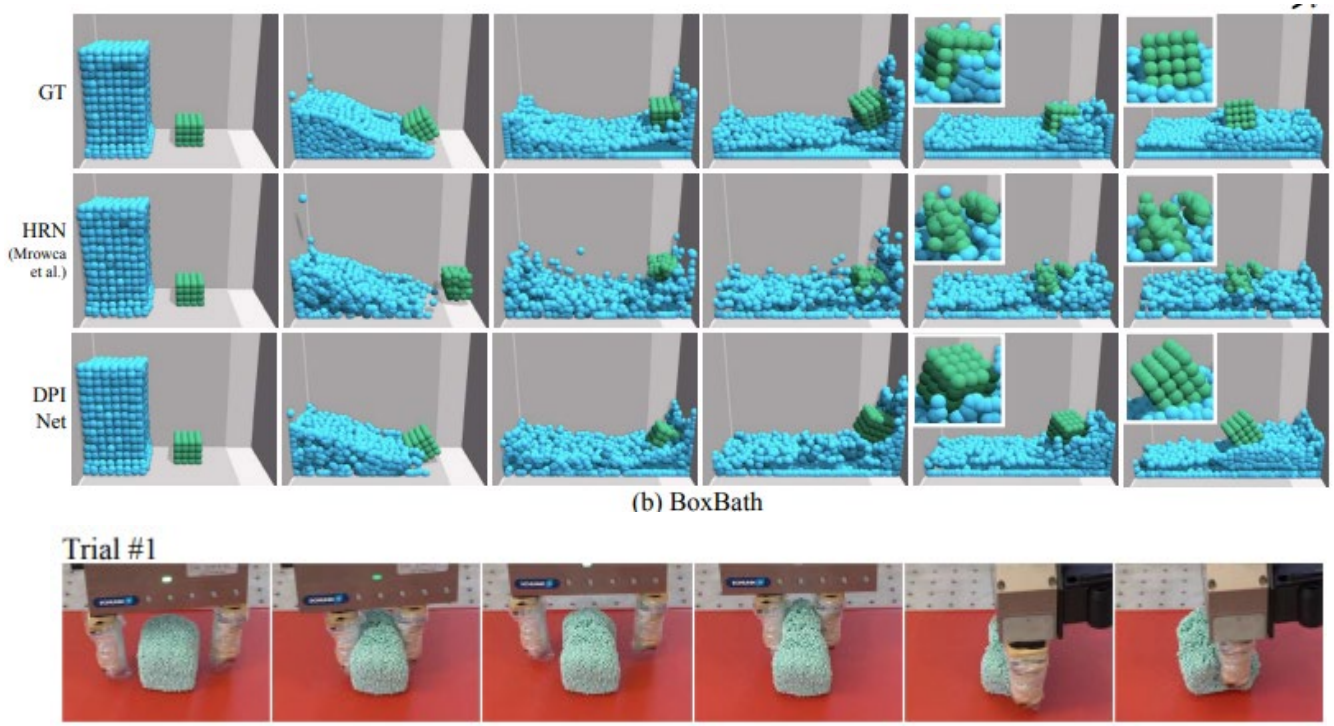


DPI can learn particles interaction by:

- Creating graph on the fly
- Local propagation
- Hierarchical propagation (for long-range predictions)

Li, Yunzhu, et al. "Learning particle dynamics for manipulating rigid bodies, deformable objects, and fluids." *arXiv preprint arXiv:1810.01566* (2018).

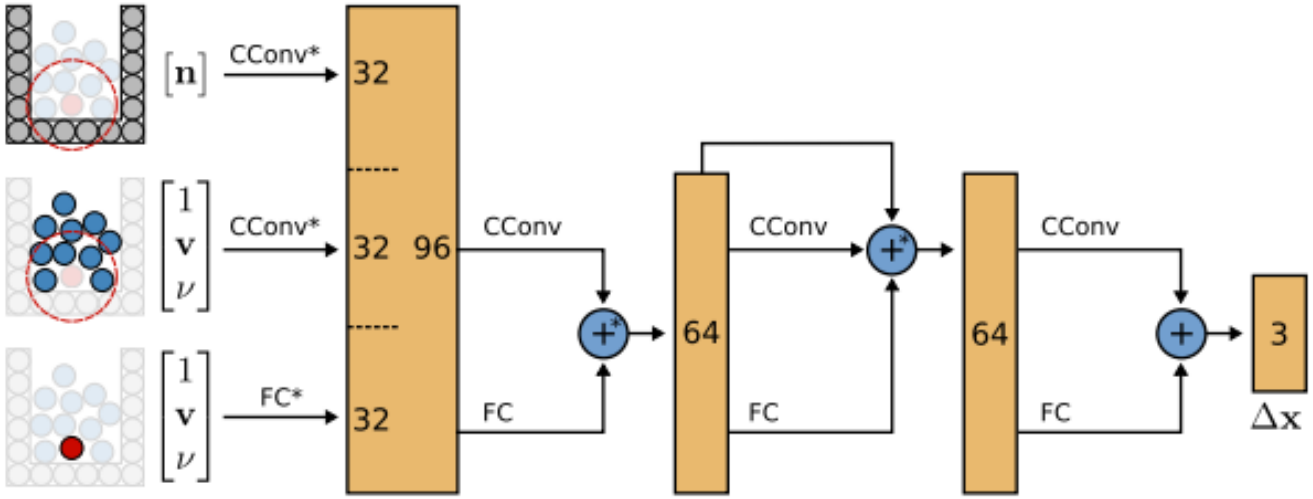
Dynamic Particle Interaction Networks (DPI) - continued



- By imposing constraints, we can simulate interactions of solids and liquids
- We can use the simulator to manipulate non-solid objects, e.g. rice

Li, Yunzhu, et al. "Learning particle dynamics for manipulating rigid bodies, deformable objects, and fluids." *arXiv preprint arXiv:1810.01566* (2018).

Continuous Convolution Networks (CConv-Nets)



- Use both continuous convolutions and fully connected nets on:
  - Static particles (CConv)
  - Dynamic particles (CConv)
  - Each particle (FC)

- Obtain displacement as output

Ummenhofer, Benjamin, et al. "Lagrangian fluid simulation with continuous convolutions." *International Conference on Learning Representations*. 2019.

Continuous Convolution Networks (CConv-Nets) - continued

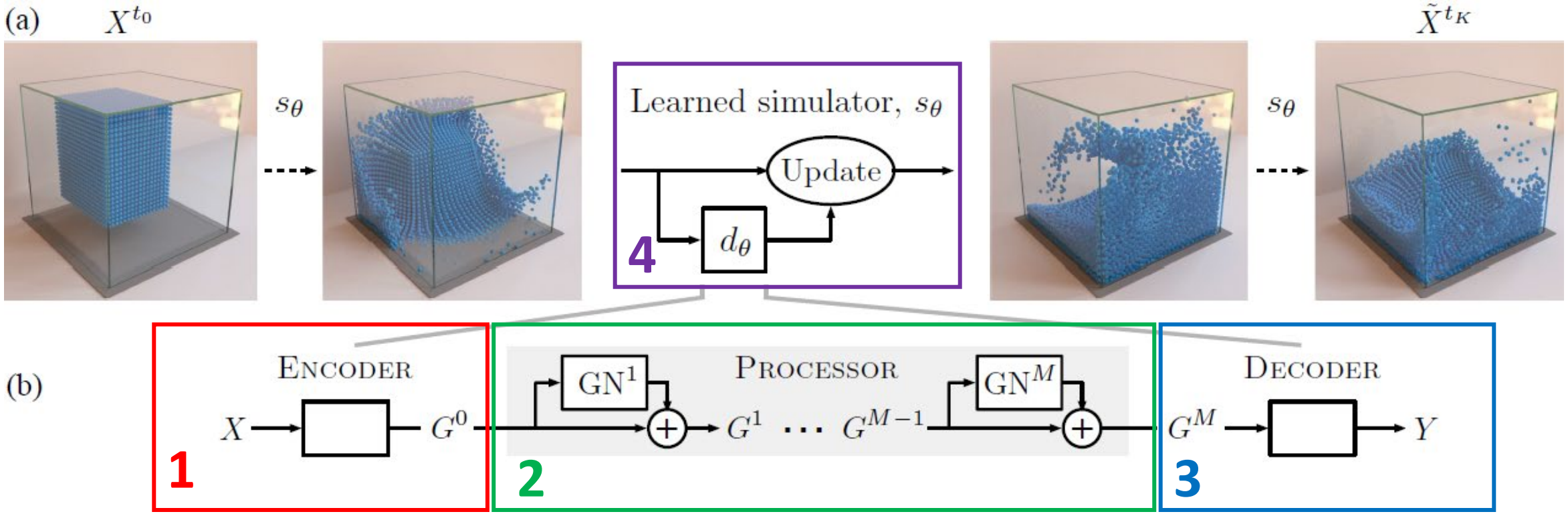
	Method	Average pos error (mm) $n+1$	Average pos error (mm) $n+2$	Average distance to closest point d^n (mm)	Frame inference time (ms)
DPI DamBreak	DPI-Nets	12.73	25.38	22.07	202.56
	SPNets Convs	–	–	–	1058.46
	PCNN Convs	0.72	1.67	19.79	187.34
	SplineCNN Convs	0.71	1.65	170.20	67.67
	KPConv	2.49	7.05	unstable	47.96
	Ours	**0.62**	**1.49**	**16.98**	**12.01**
Our data (6K particles)	DPI-Nets	26.19	51.77	unstable	305.55
	SPNets Convs	–	–	–	784.35
	PCNN Convs	0.67	1.87	32.51	319.17
	SplineCNN Convs	0.68	1.93	unstable	281.92
	KPConv	1.65	4.54	unstable	57.89
	Ours	**0.56**	**1.51**	**29.50**	**16.47**

- Cconv-Nets can produce reliable fluid simulations
- Inference time is also low compared to other methods

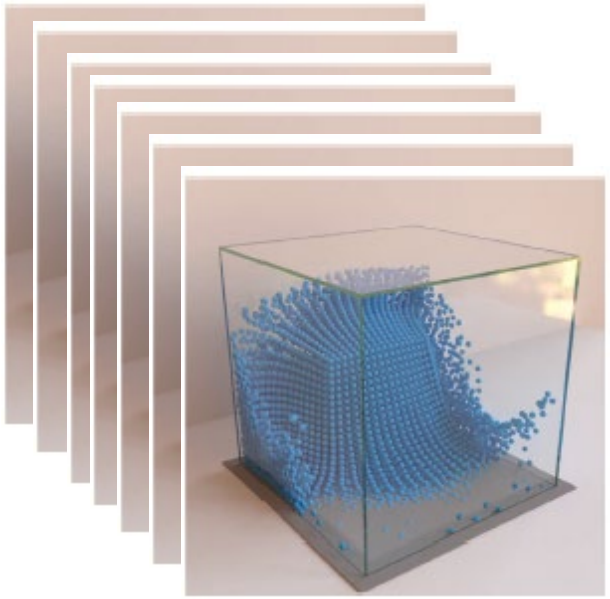
Ummenhofer, Benjamin, et al. "Lagrangian fluid simulation with continuous convolutions." *International Conference on Learning Representations*. 2019.

Methodology: Graph Network-Based Simulator (GNS)

GNS: Graph Network-based Simulator Overview



1 – Inputs and Encoder

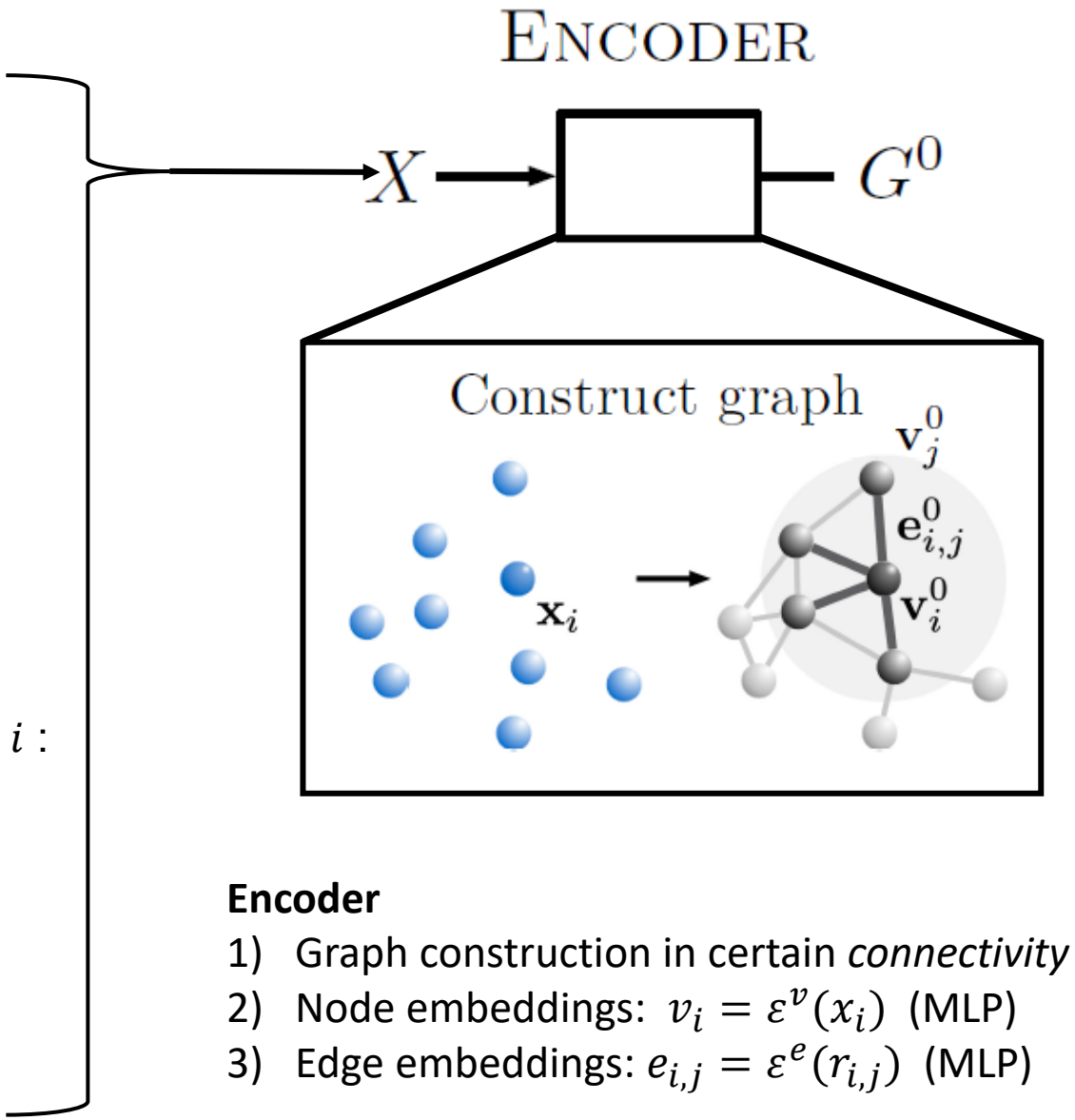


Input Features x_i for each particle i :

- Position p_i
- Last 5 velocities $\dot{p}_i$
- Particle type f_i

Resulting vector:

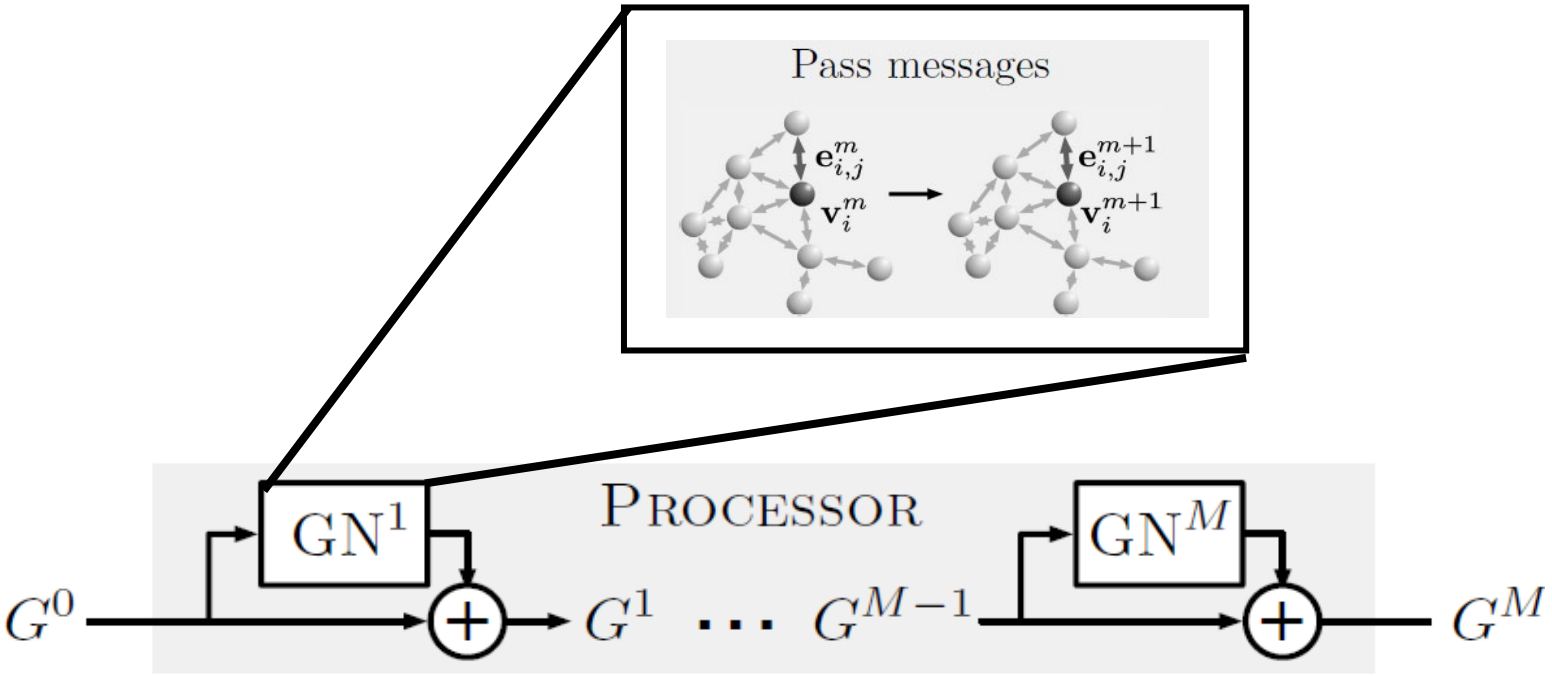
$$x_i^{t_k} = [p_i^{t_k}, \dot{p}_i^{t_{k-C+1}}, \ldots, \dot{p}_i^{t_k}, f_i]$$



Encoder

1) Graph construction in certain *connectivity radius R*
2) Node embeddings: $v_i = \varepsilon^v(x_i)$ (MLP)
3) Edge embeddings: $e_{i,j} = \varepsilon^e(r_{i,j})$ (MLP)

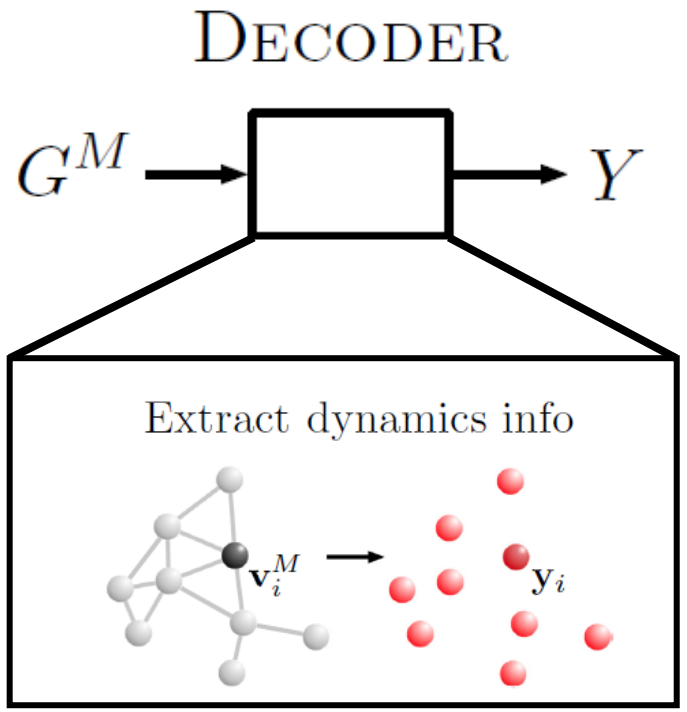
2- Processor and Message Passing



Processor

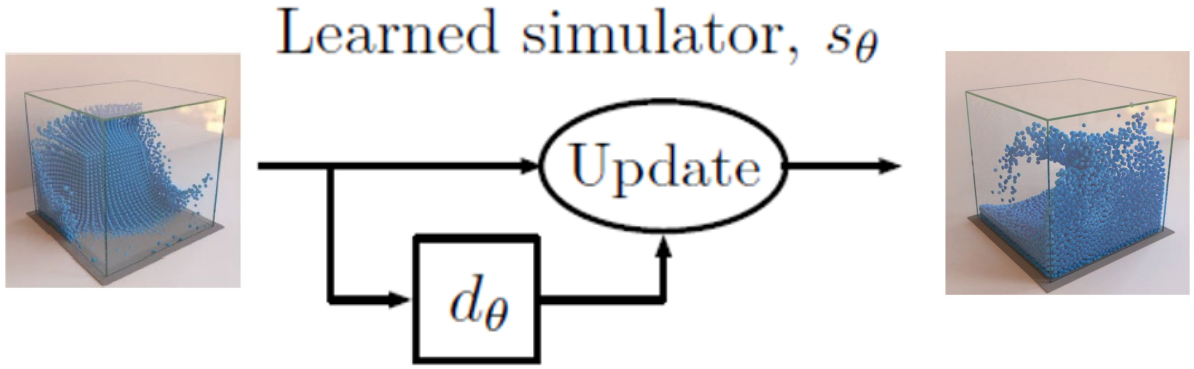
1) Message passing in the latent graphs $G^{m+1} = GN^{m+1}(G^m)$
   where node, edge update functions are MLPs
2) Updated Latent graphs $\mathcal{G} = (G^1, \ldots, G^M)$
3) Total update: $G^M = PROCESSOR(G^0)$

3 – Decoder and 4 - Euler Step



Decoder: extract dynamics information (e.g., acceleration)

- Decoder learned function: δ^v (MLP)
- Output $Y = \delta^v(v_i^M)$

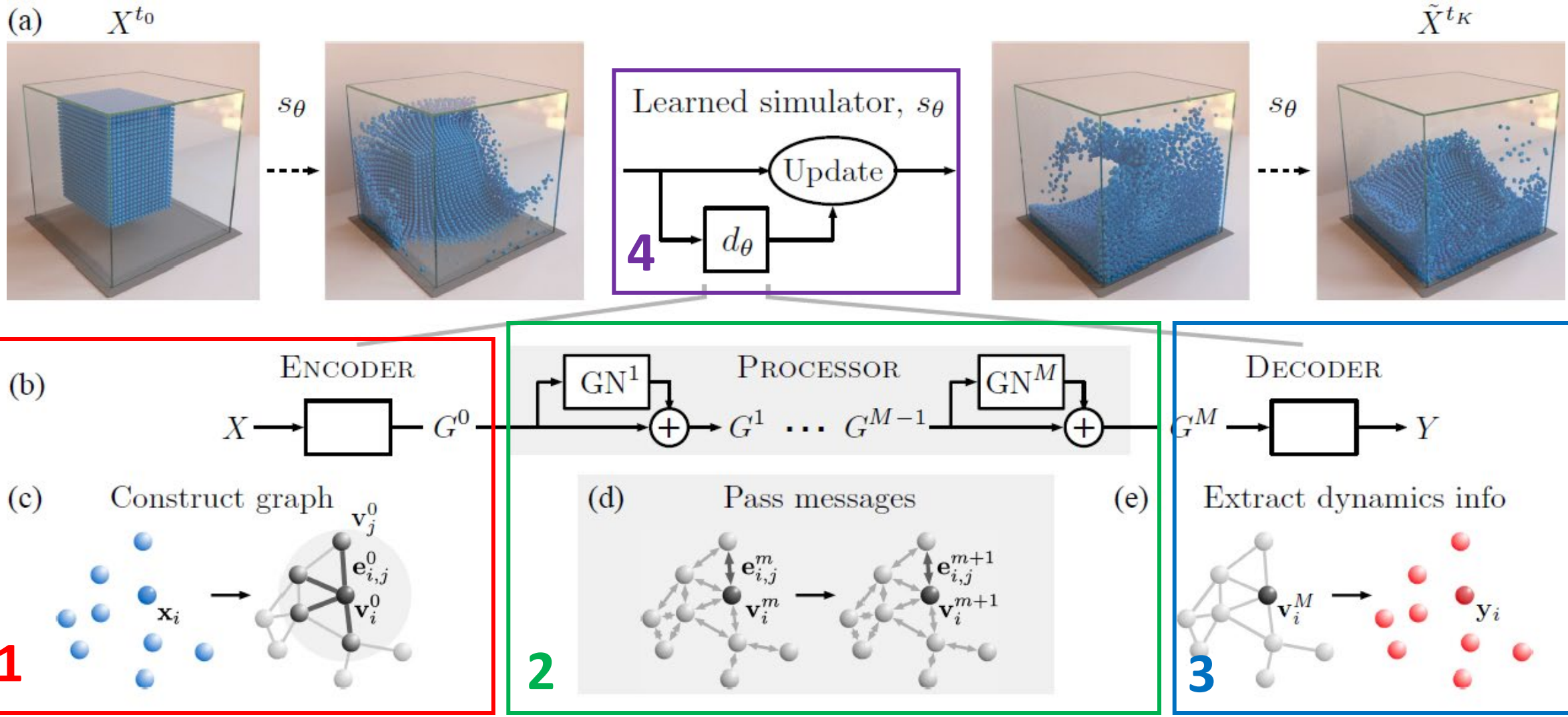


Euler Step:
Use the semi-implicit Euler scheme to update final positions:

$$\dot{\boldsymbol{p}}^{t_{k+1}} = \dot{\boldsymbol{p}}^{t_k} + \Delta t \cdot \ddot{\boldsymbol{p}}^{t_k}$$
$$\boldsymbol{p}^{t_{k+1}} = \boldsymbol{p}^{t_k} + \Delta t \cdot \dot{\boldsymbol{p}}^{t_{k+1}}$$

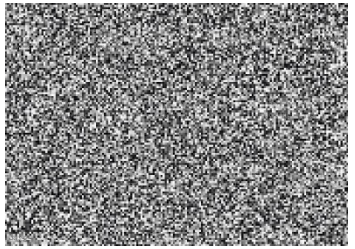
Semi-implicit: the new velocity influences directly the new position (in the explicit update $\Delta t \cdot \dot{\boldsymbol{p}}^{t_k}$ is used)

GNS: Graph Network-based Simulator Revised Overview

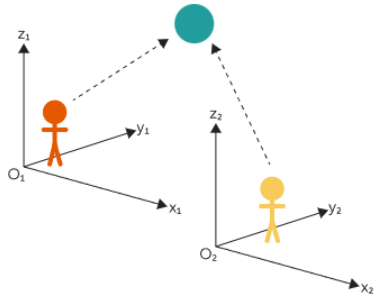


Additional Elements

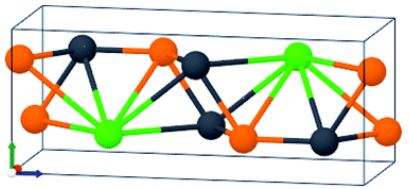
- **Noise**: *random walk noise* (accumulate noise as in real rollout) added during training to render model more robust → Choice of noise as $\mathcal{N}(0, \sigma_v = 0.0003)$ on positions



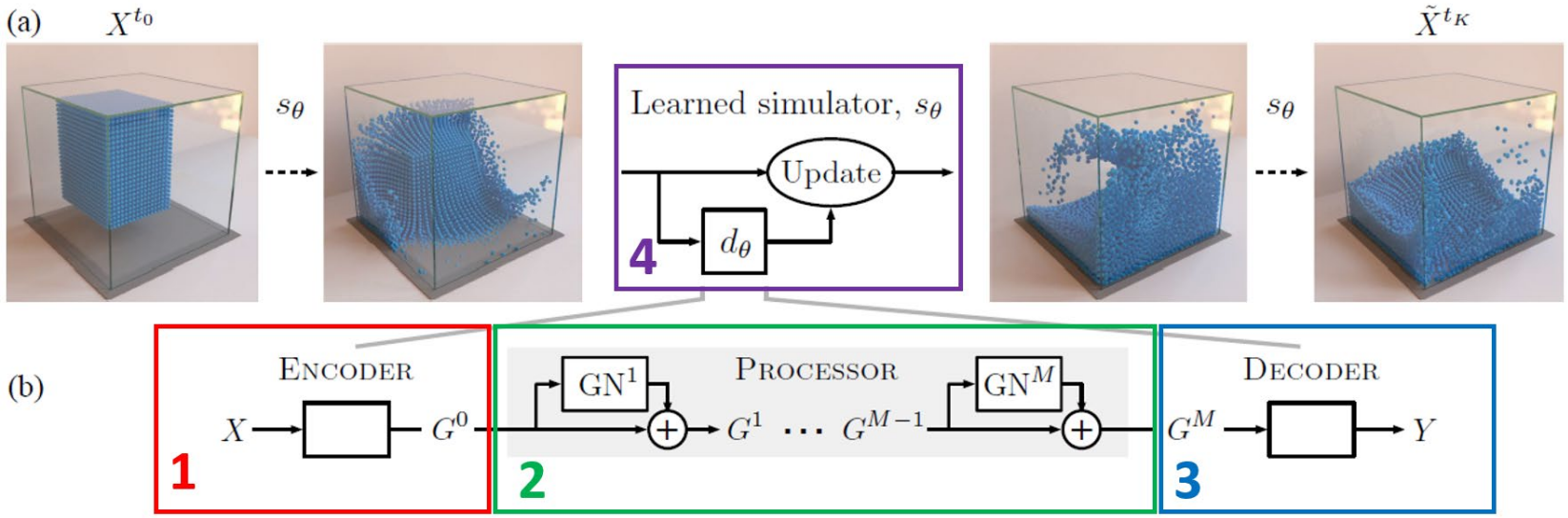
- **Relative encoder**: instead of using absolute information on position, use relative information → inductive bias on invariance



- **Global features:** features that can be present and added as node inputs → representation of external forces and movements



GNS Training



The loss function can be simply an L_2 loss between target and predicted positions:

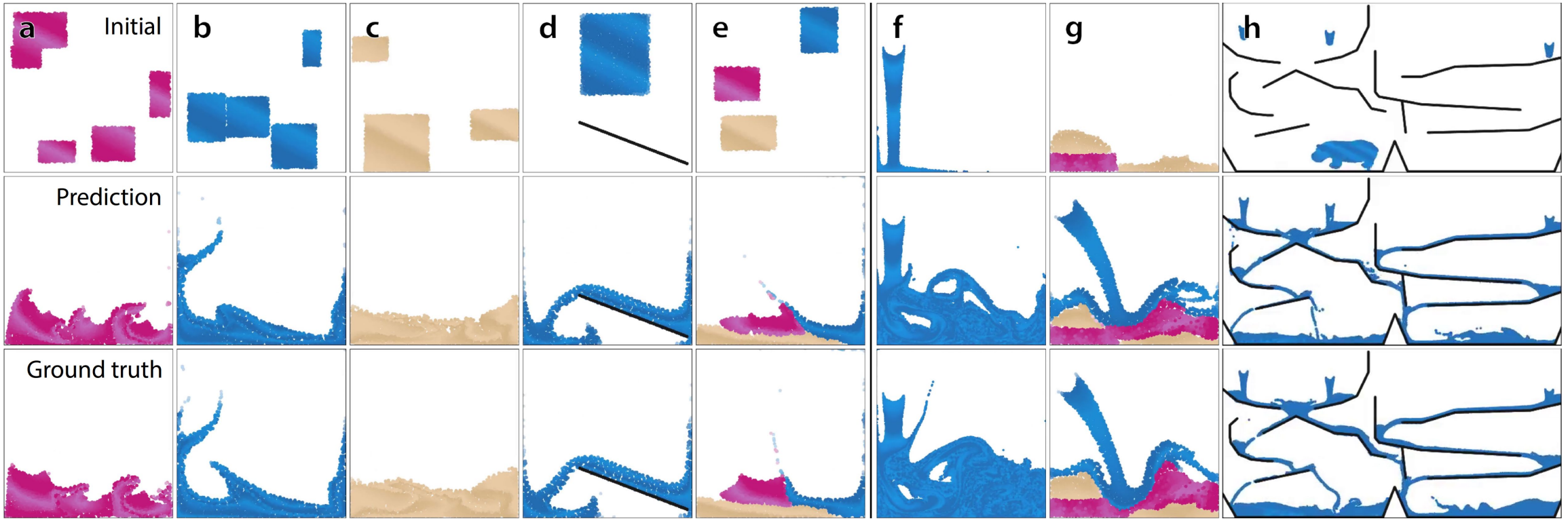
$$L = \|x, \tilde{x}\|^2$$

Training the model means minimizing the loss function over multiple trajectories, hence:

$$\min_\theta \mathbb{E}_{\mathbb{P}(X^{t_{o:K}})} L\left(X^{t_{1:k}}, \tilde{X}^{t_{1:K}}_{s_\theta, X^{t_0}}\right)$$

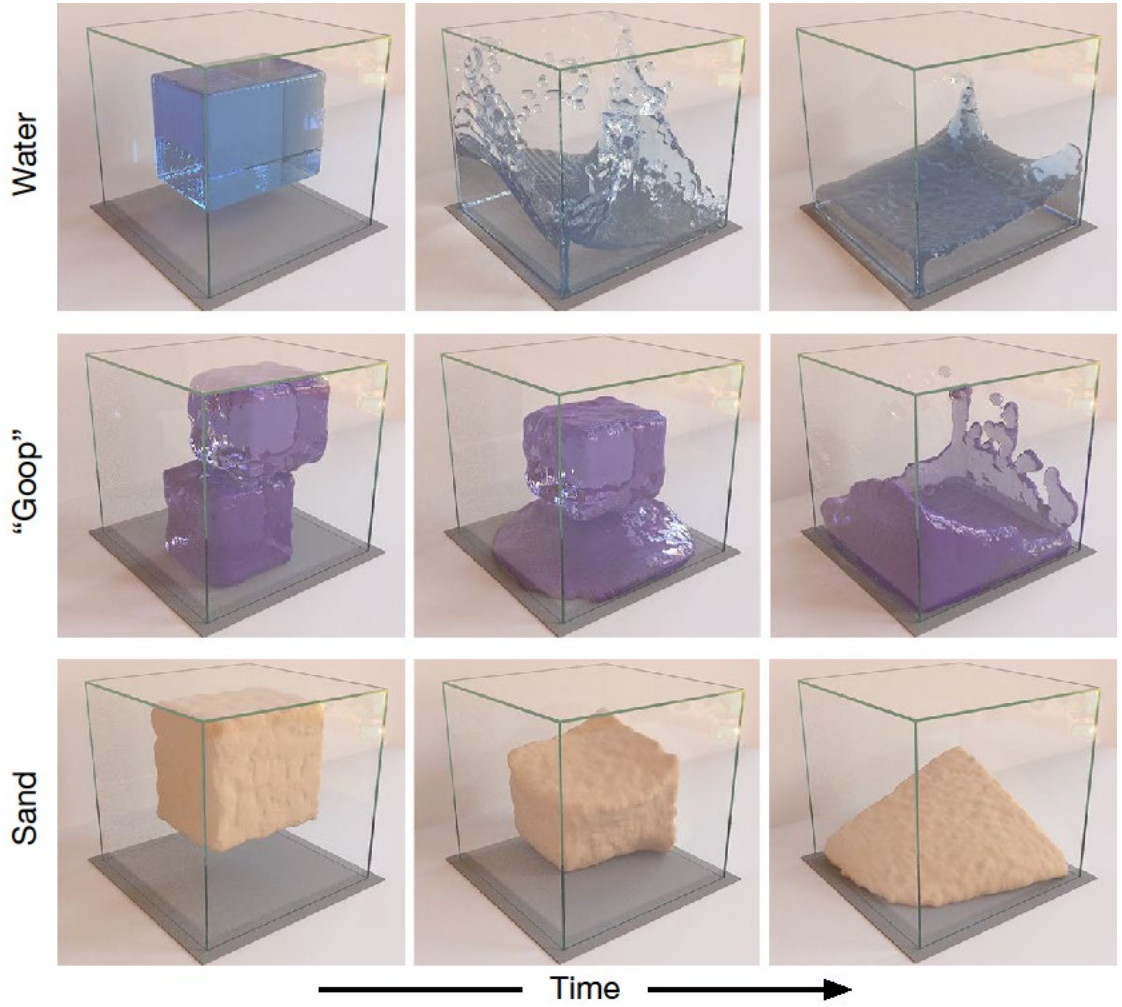
Experiments

Experiment Setup and Generalization



→ GNS is able to generalize to more particle number at different positions, for longer rollouts and even for more complex domains with e.g. more ramps or interaction with multiple particle types

Different and Complex Domains



→ GNS is able to generalize to more particle number at different positions, for longer rollouts and even in 3D domains

Metrics: MSE, OT, MMD

- **Mean Squared Error (MSE):**

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

→ Problem: if predicted positions for particles A and B exactly match true positions of particles B and A, respectively, the MSE could be high, even though the predicted and true distributions of particles match!

- **Optimal Transport (OT):** uses 2D or 3D Wasserstein distance and approximated by the Sinkhorn Algorithm[1]. Intuitively, this metric represents the minimum "cost" of moving a distribution (e.g. mass of liquid) and thus more suitable for particle distribution problems

- **Maximum Mean Discrepancy (MMD):** defined by the idea of representing distances between distributions as distances between *mean embeddings* of features[2]. Similarly to OT, it is permutation-invariant and suitable to particle distribution measurement

[1]Villani, Cédric. *Topics in optimal transportation*. Vol. 58. American Mathematical Soc., 2021.

[2]Gretton, Arthur, et al. "A kernel two-sample test." *The Journal of Machine Learning Research* 13.1 (2012): 723-773.

Metrics: Errors on Rollouts

Experimental domain	N	K	1-step ($\times 10^{-9}$)	Rollout ($\times 10^{-3}$)
WATER-3D (SPH)	13k	800	8.66	10.1
SAND-3D	20k	350	1.42	0.554
GOOP-3D	14k	300	1.32	0.618
WATER-3D-S (SPH)	5.8k	800	9.66	9.52
BOXBATH (PBD)	1k	150	54.5	4.2
WATER	1.9k	1000	2.82	17.4
SAND	2k	320	6.23	2.37
GOOP	1.9k	400	2.91	1.89
MULTIMATERIAL	2k	1000	1.81	16.9
FLUIDSHAKE	1.3k	2000	2.1	20.1
WATERDROP	1k	1000	1.52	7.01
WATERDROP-XL	7.1k	1000	1.23	14.9
WATERRAMPS	2.3k	600	4.91	11.6
SANDRAMPS	3.3k	400	2.77	2.07
RANDOMFLOOR	3.4k	600	2.77	6.72
CONTINUOUS	4.3k	400	2.06	1.06

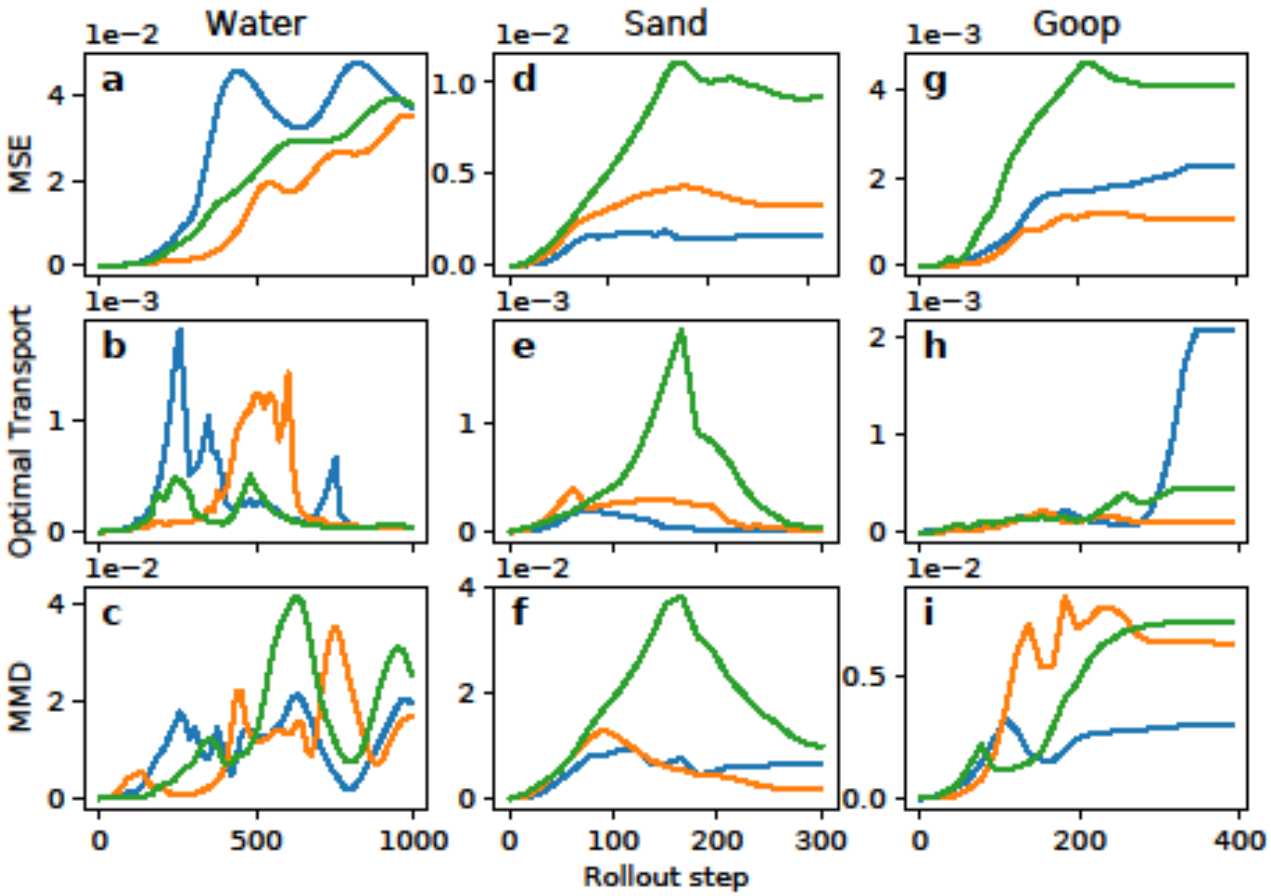
Metrics: More Details on Problem Size

Domain	Simulator (Dim.)	Max. # particles (approx)	Trajectory length	# trajectories (Train/ Validation/ Test)	Δt [ms]	Connectivity radius	Max. # edges (approx)
WATER-3D	SPH (3D)	13k	800	1000/100/100	5	0.035	230k
SAND-3D	MPM (3D)	20k	350	1000/100/100	2.5	0.025	320k
GOOP-3D	MPM (3D)	14k	300	1000/100/100	2.5	0.025	230k
WATER-3D-S	SPH (3D)	5.8k	800	1000/100/100	5	0.045	100k
BOXBATH	PBD (3D)	1k	150	2700/150/150	16.7	0.08	17k
WATER	MPM (2D)	1.9k	1000	1000/30/30	2.5	0.015	27k
SAND	MPM (2D)	2k	320	1000/30/30	2.5	0.015	21k
GOOP	MPM (2D)	1.9k	400	1000/30/30	2.5	0.015	19k
MULTIMATERIAL	MPM (2D)	2k	1000	1000/100/100	2.5	0.015	25k
FLUIDSHAKE	MPM (2D)	1.3k	2000	1000/100/100	2.5	0.015	20k
FLUIDSHAKE-BOX	MPM (2D)	1.5k	1500	1000/100/100	2.5	0.015	19k
WATERDROP	MPM (2D)	1k	1000	1000/30/30	2.5	0.015	12k
WATERDROP-XL	MPM (2D)	7.1k	1000	1000/100/100	2.5	0.01	210k
WATERRAMPS	MPM (2D)	2.3k	600	1000/100/100	2.5	0.015	26k
SANDRAMPS	MPM (2D)	3.3k	400	1000/100/100	2.5	0.015	32k
RANDOMFLOOR	MPM (2D)	3.4k	600	1000/100/100	2.5	0.015	44k
CONTINUOUS	MPM (2D)	4.3k	400	1000/100/100	2.5	0.015	47k

MSE, OT, MMD

Domain	Mean Squared Error		Optimal Transport		Maximum Mean Discrepancy ($\sigma = 0.1$)	
	One-step $\times 10^{-9}$	Rollout $\times 10^{-3}$	One-step $\times 10^{-9}$	Rollout $\times 10^{-3}$	One-step $\times 10^{-9}$	Rollout $\times 10^{-3}$
WATER-3D	8.66	10.1	26.5	0.165	7.32	0.368
SAND-3D	1.42	0.554	4.29	0.138	11.9	2.67
GOOP-3D	1.32	0.618	4.05	0.208	22.4	5.13
WATER-3D-S	9.66	9.52	29.9	0.222	6.9	0.192
BOXBATH	54.5	4.2	–	–	–	–
WATER	2.82	17.4	6.19	0.468	10.6	7.66
SAND	6.23	2.37	11.8	0.193	32.6	6.79
GOOP	2.91	1.89	6.14	0.419	20.3	7.76
MULTIMATERIAL	1.81	16.9	–	–	–	–
FLUIDSHAKE	2.1	20.1	4.13	0.591	12.1	9.84
FLUIDSHAKE-BOX	1.33	4.86	–	–	–	–
WATERDROP	1.52	7.01	3.31	0.273	11.1	5.99
WATERDROP-XL	1.23	14.9	3.03	0.209	11.6	4.34
WATERRAMPS	4.91	11.6	10.3	0.507	14.3	7.68
SANDRAMPS	2.77	2.07	6.13	0.187	15.7	4.81
RANDOMFLOOR	2.77	6.72	5.8	0.276	14	3.53
CONTINUOUS	2.06	1.06	4.63	0.0709	23.1	3.57

MSE, OT, MMD Over Rollouts

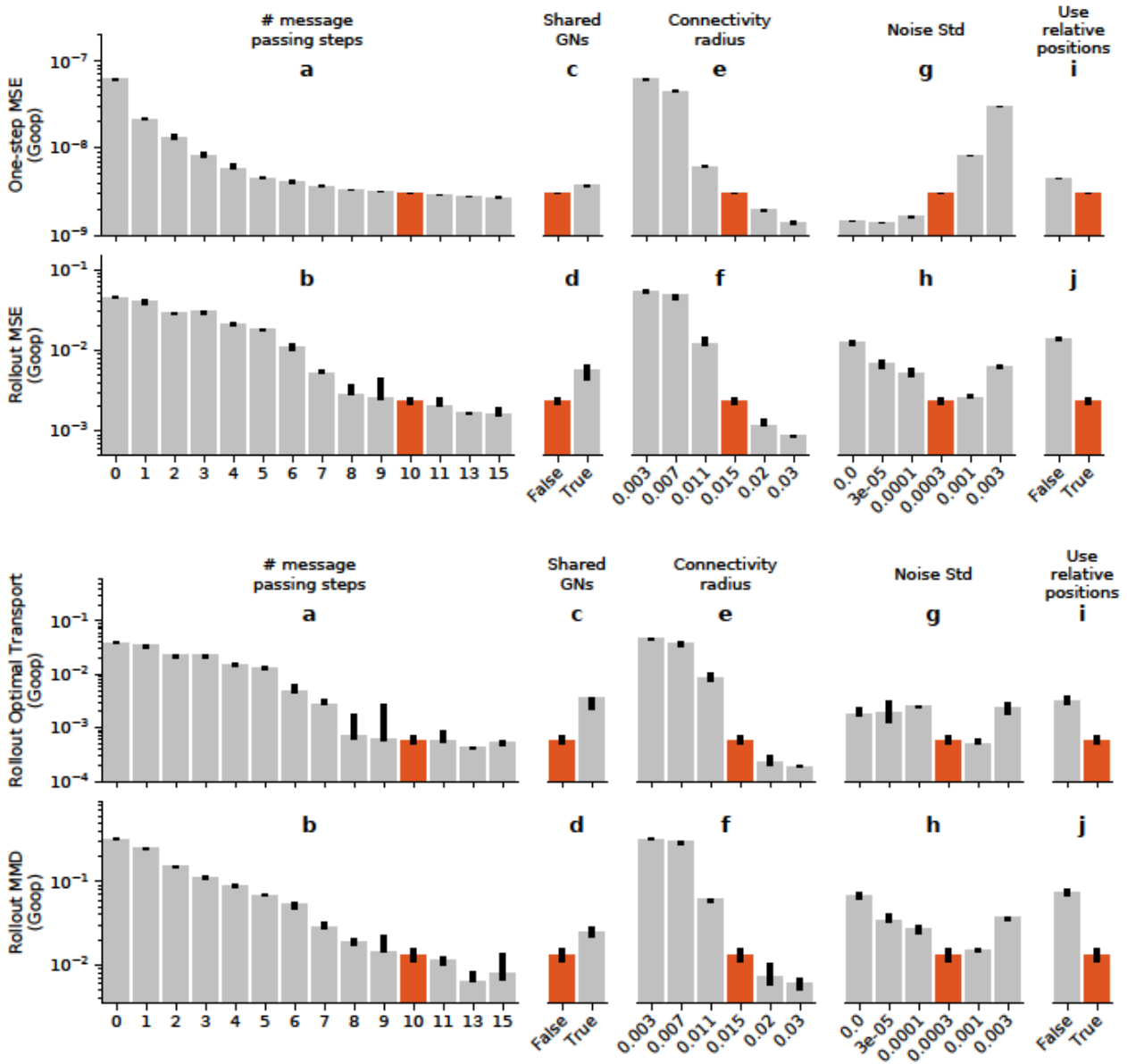


Main Ablation Study

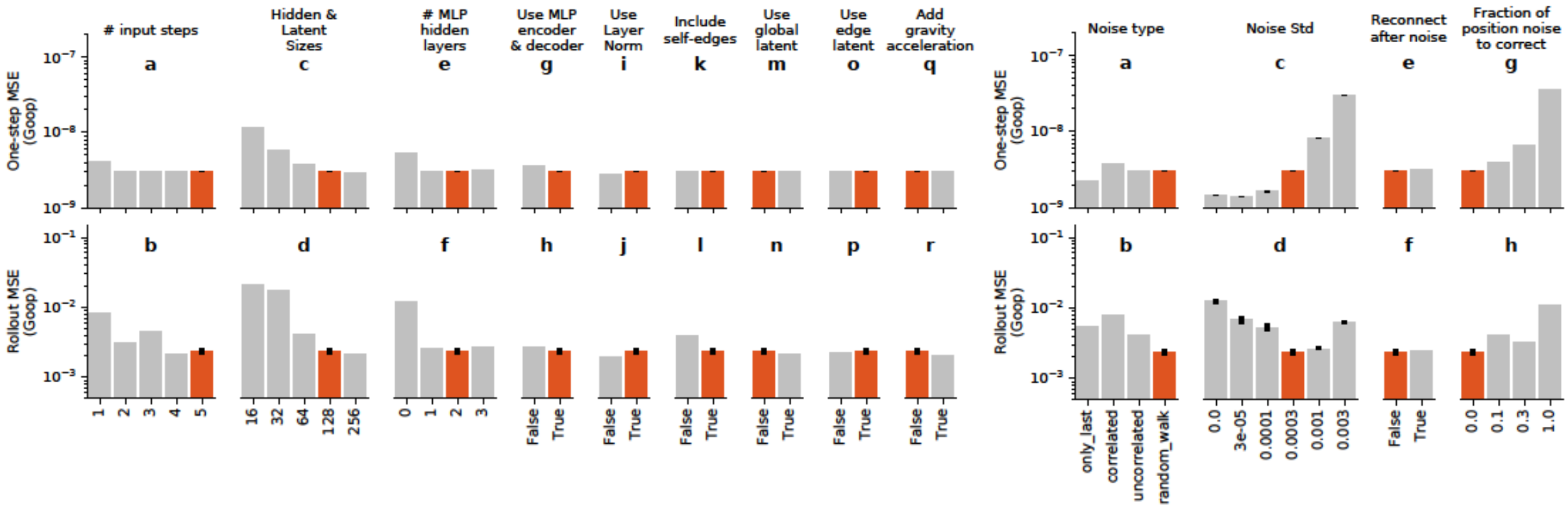
Key architectural choices:

1. Number of message passing steps (more steps: longer interactions)
2. Unshared Processor GN parameters (deeper network)
3. Connectivity radius (larger radius: more edges and more interaction)
4. Scale of the noise (noise is useful for robustness)
5. Relative encoder (relative learns invariance inductive bias)

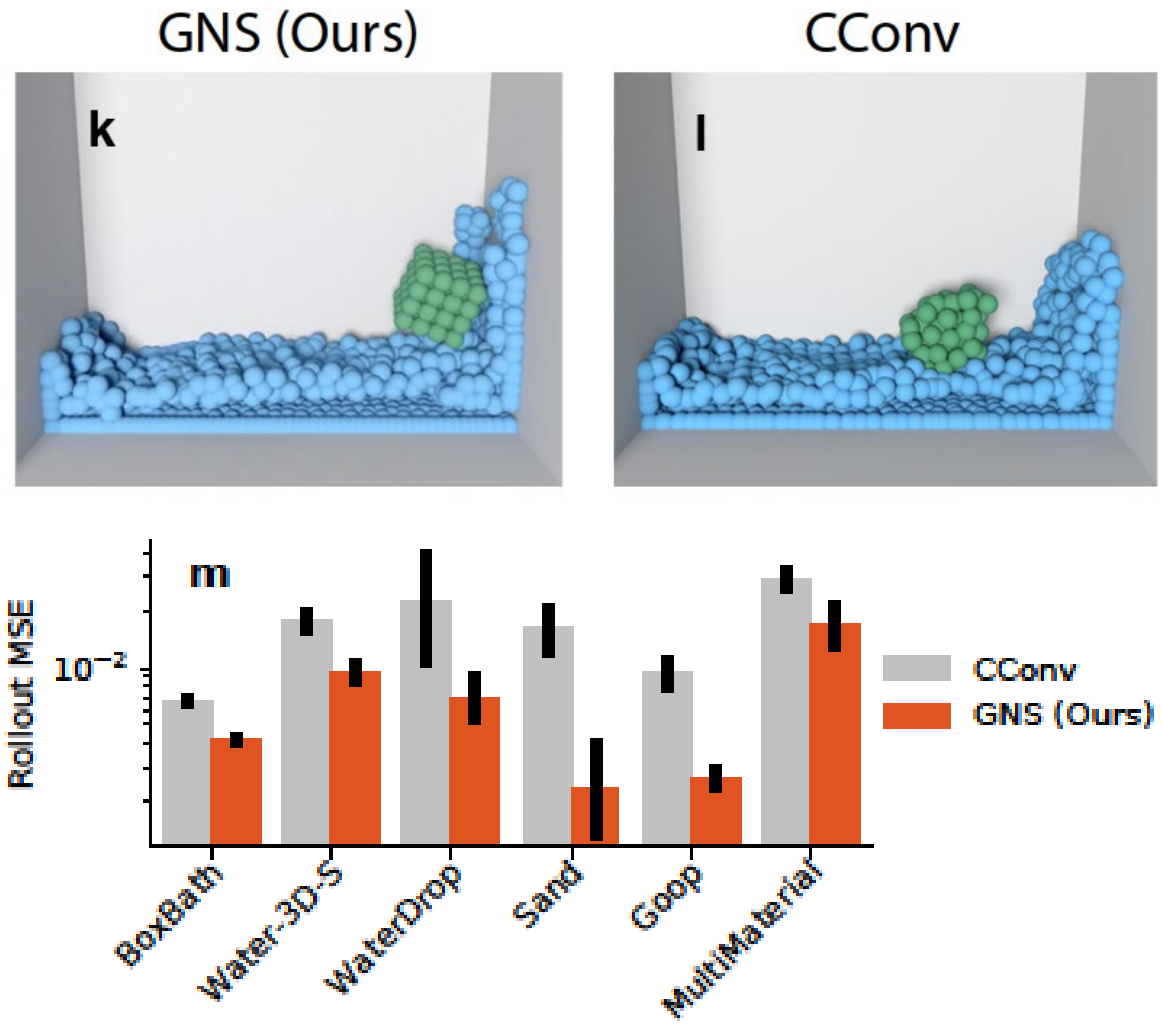
→ These properties have interesting physical intuition behind!



Ablation Study - Supplementary



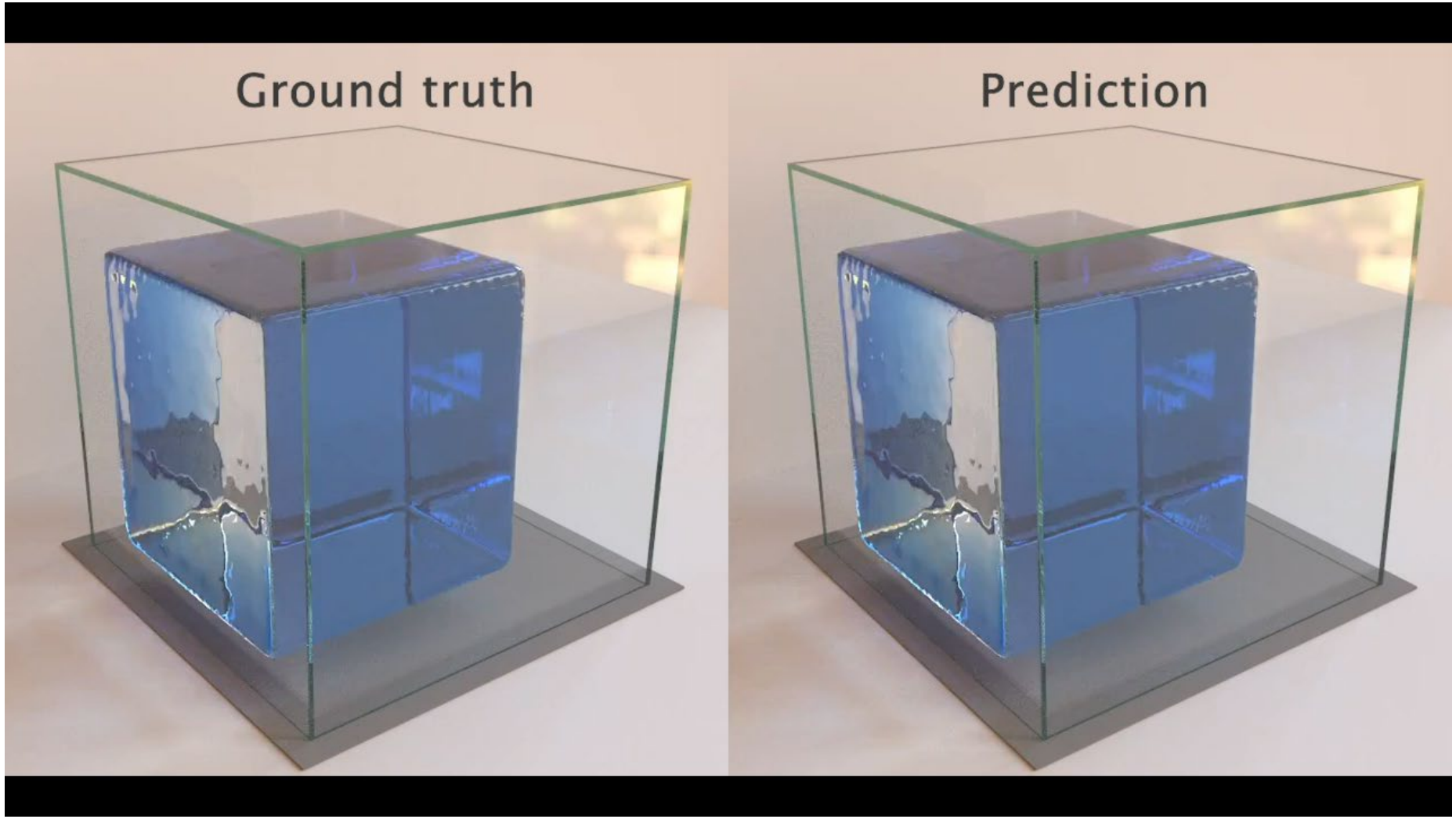
Comparison to CConv



Experiments – Videos*

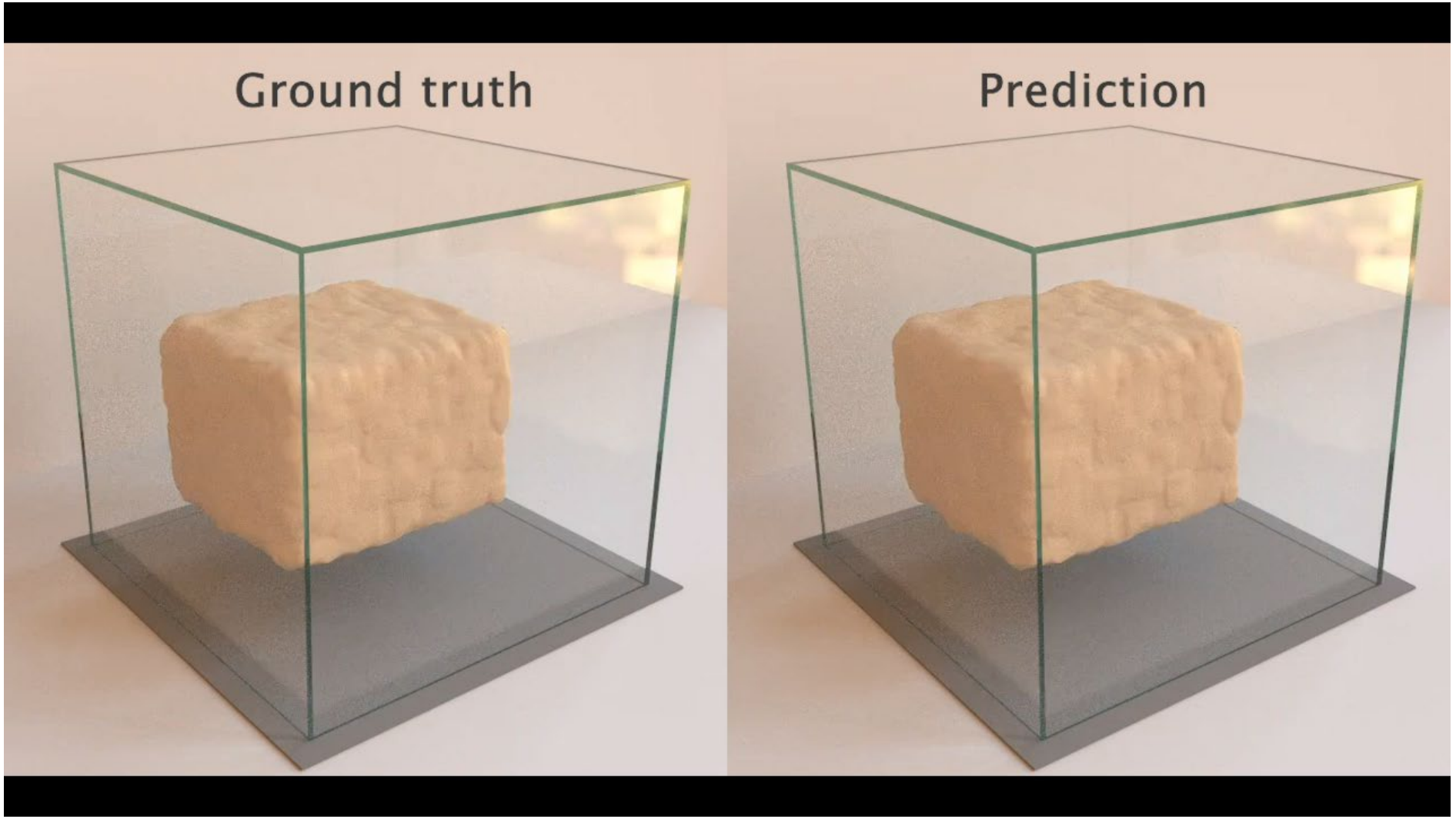
*These may not be visualized well in the .pptx – please use the provided links ☺

Water-3D



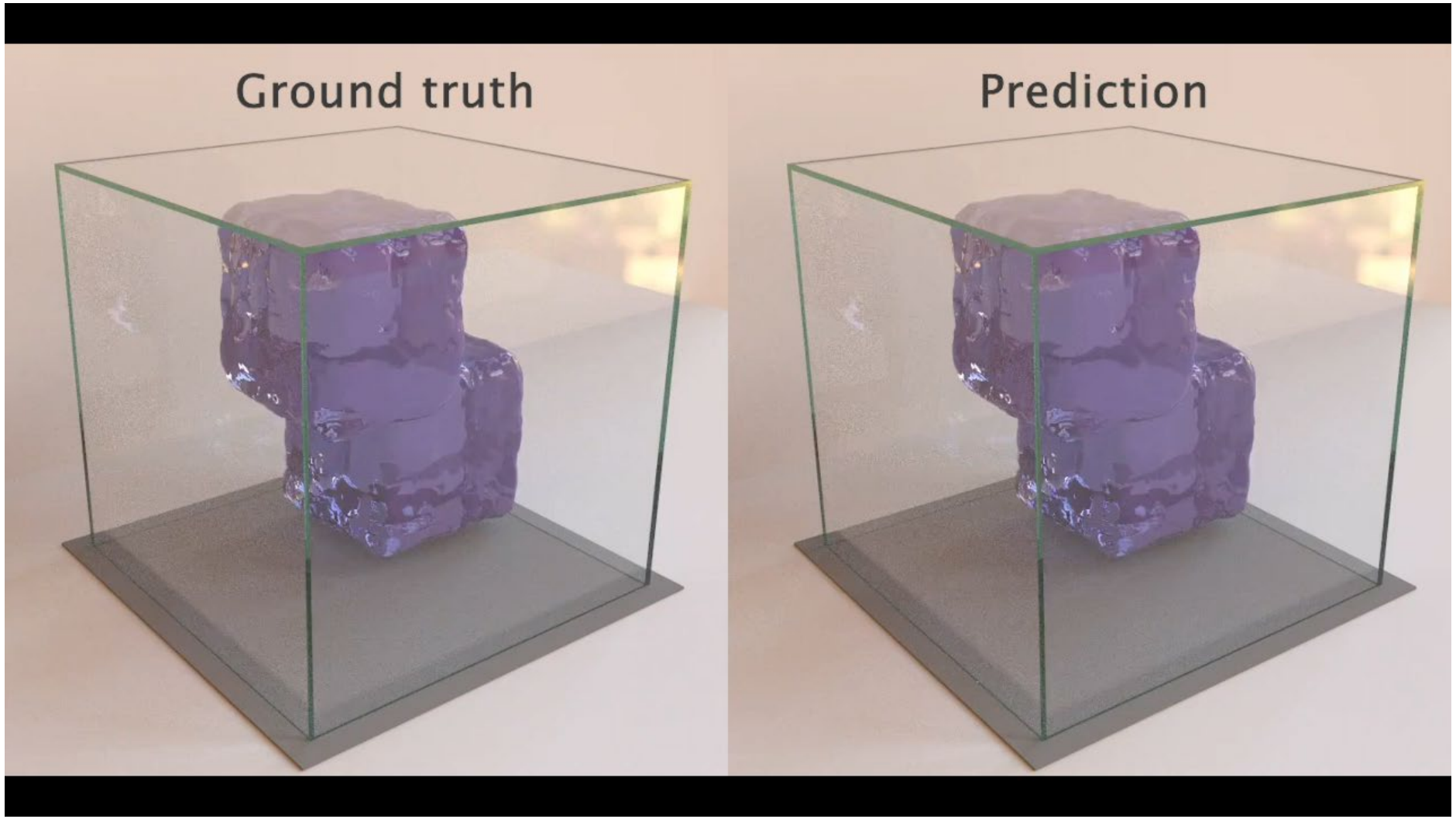
Link: https://sites.google.com/view/learning-to-simulate#h.p_hjnaJ6k8y0wo

Sand-3D



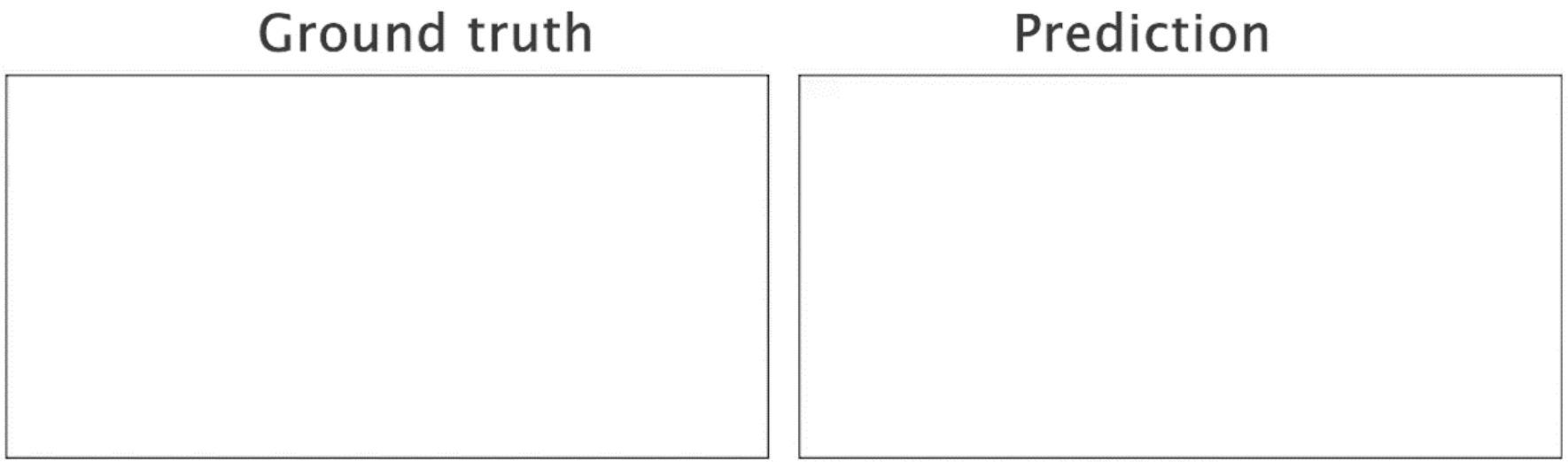
Link: https://sites.google.com/view/learning-to-simulate#h.p_n6EtI_-yN6R-

Goop-3D



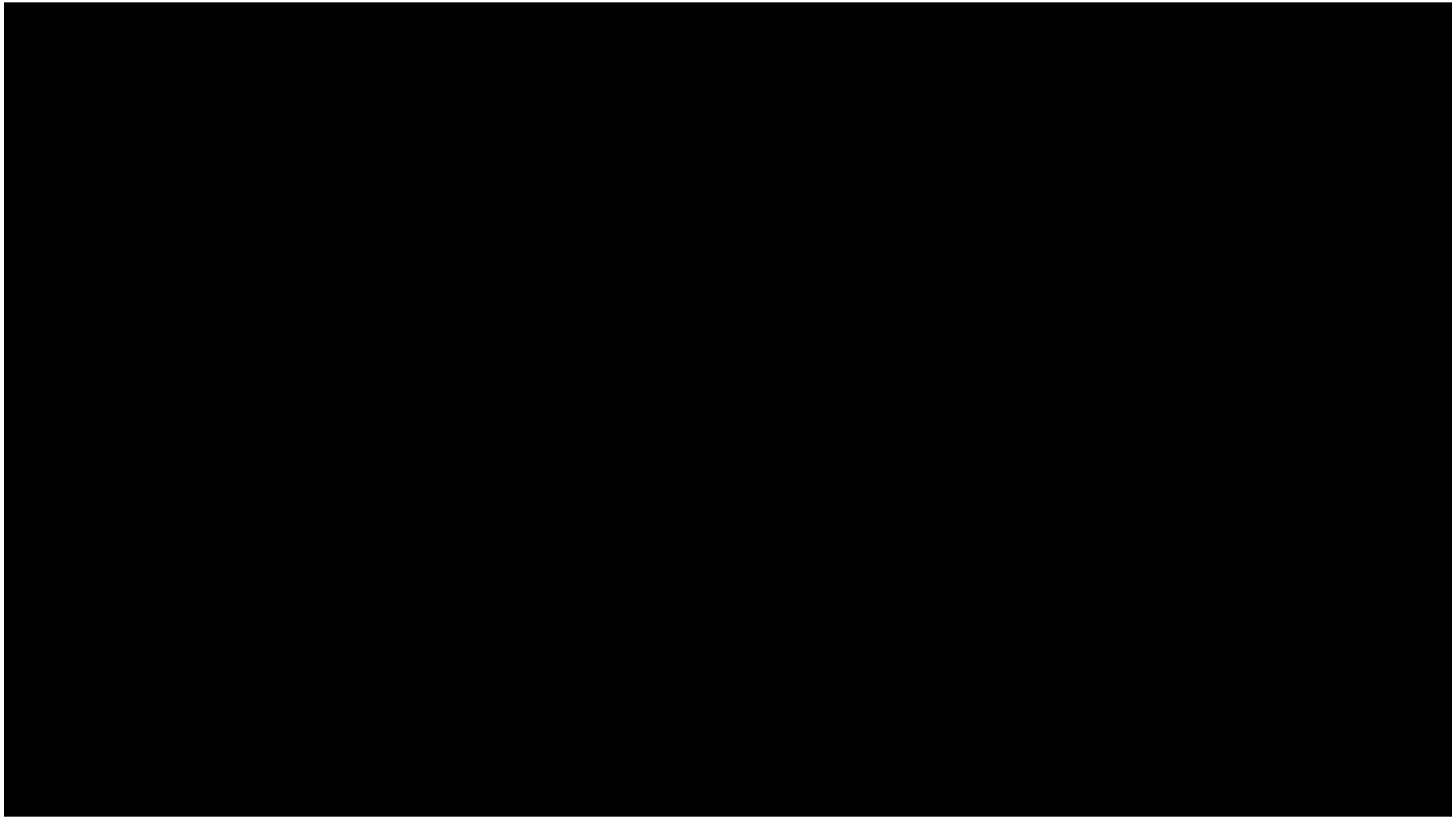
Link: https://sites.google.com/view/learning-to-simulate#h.p_1cev_oXpO9yG

Generalization over More Complex Domains



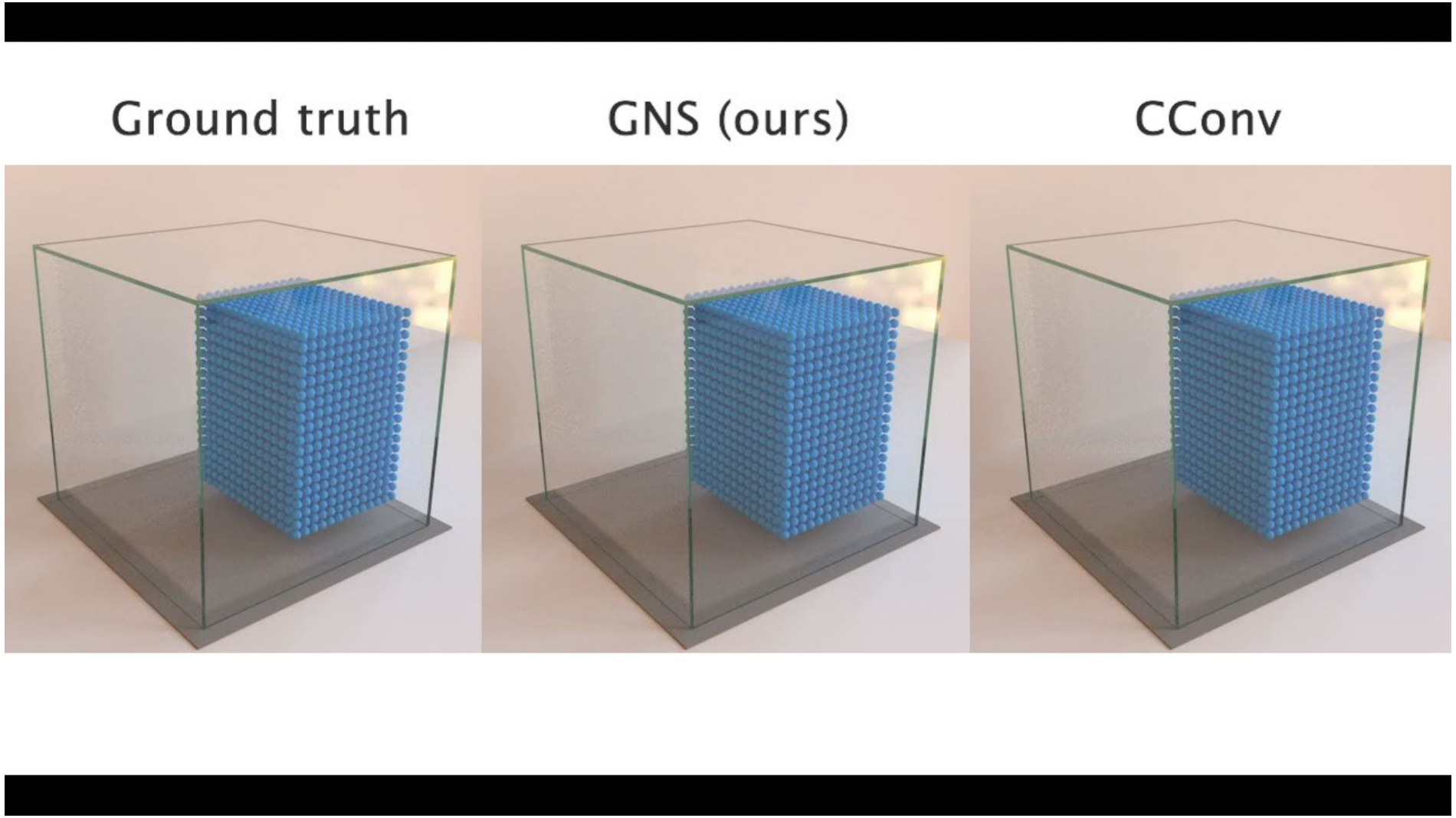
Link: https://sites.google.com/view/learning-to-simulate#h.p_oVm-SKnnQ_p9

Baseline Comparison: DPI



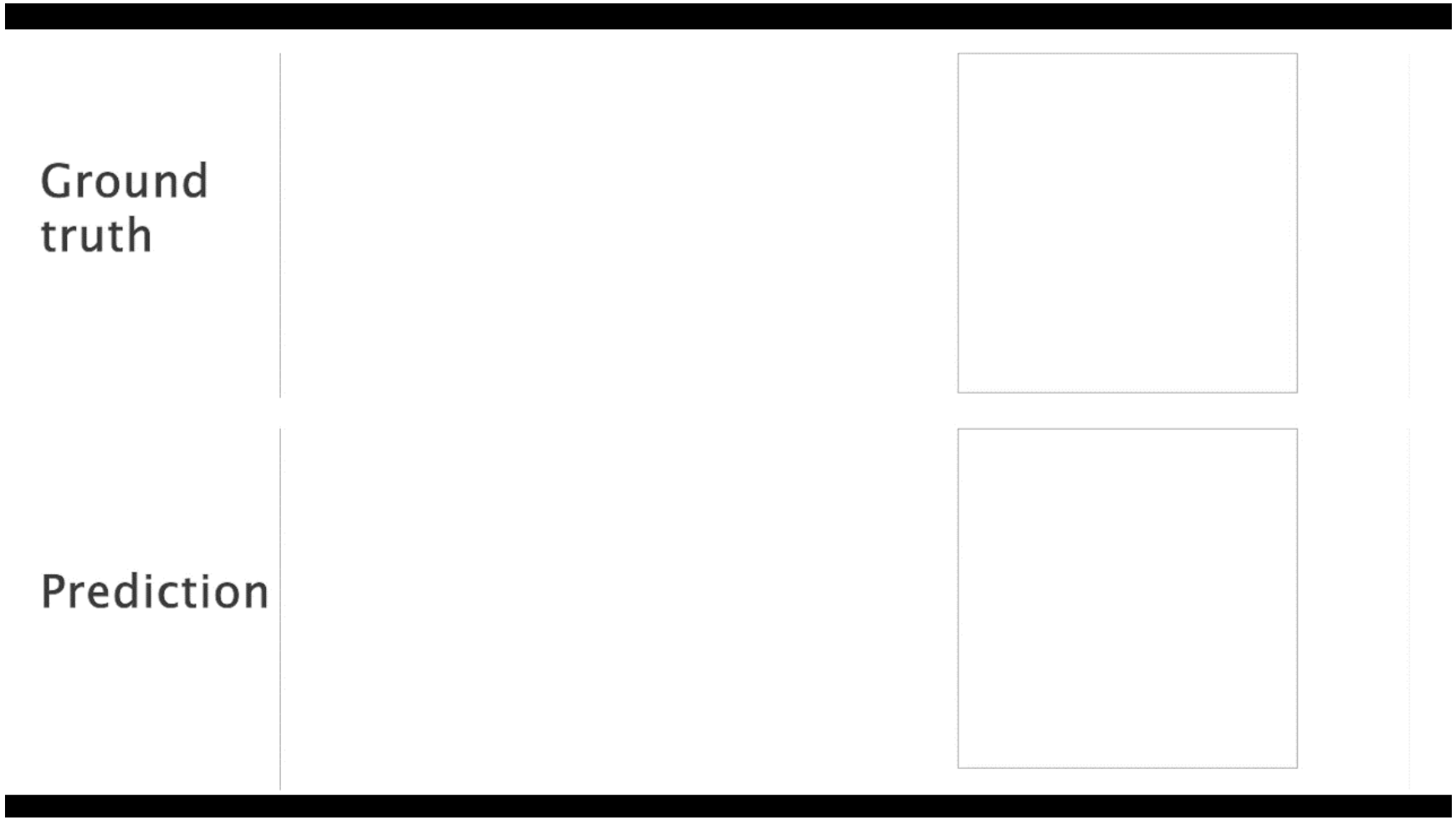
Link: https://sites.google.com/view/learning-to-simulate#h.p_qUqtrBIqti4G

Baseline Comparison: CConv



Link: https://sites.google.com/view/learning-to-simulate#h.p_rGByTjfDcDx9

Bonus: Examples of Fails



Link: https://sites.google.com/view/learning-to-simulate#h.p_GYmSoIisBUUX

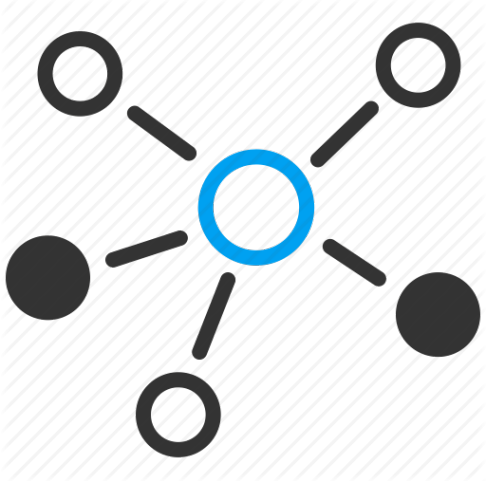
Conclusions

Conclusions: a Physical Interpretation

Why does the module work?

Inductive Biases

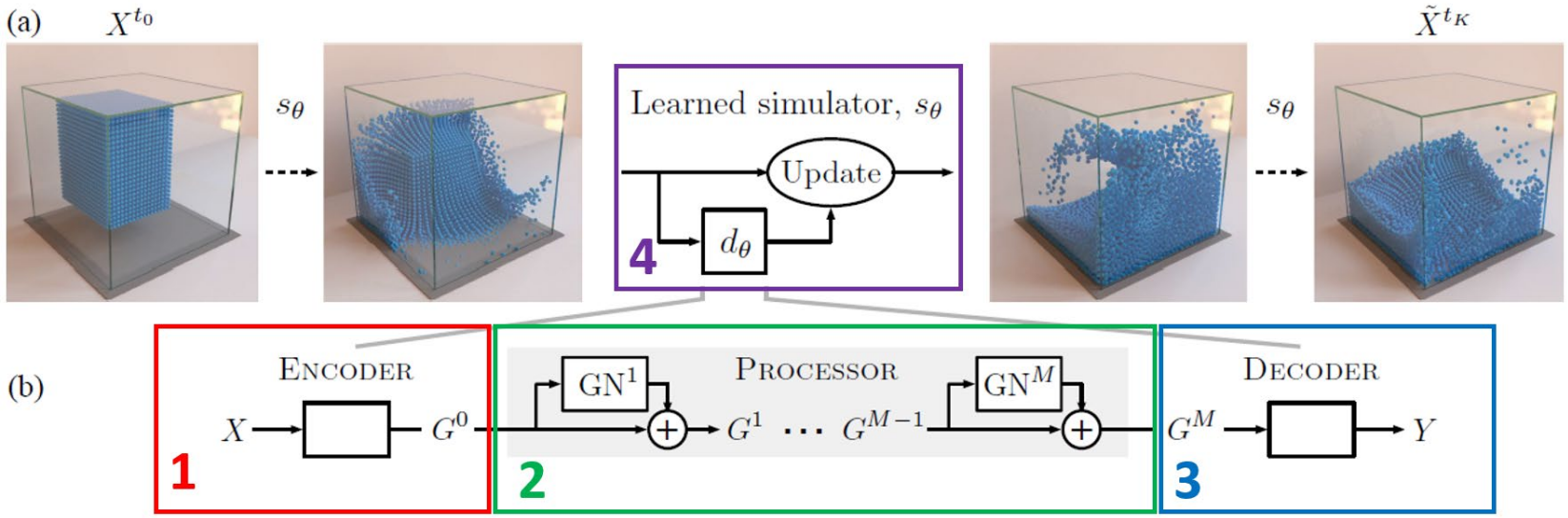
- *Shared node, edge functions:* dynamics is the same for all particles
- *Relative encoding:* absolute position is never observed, only position difference to neighbors



Effects

- Reduced input space, observing translational symmetry
- Effectively more training data each particle is a “sample”
- Less likely to overfit, more likely to generalize out-of-distribution

Conclusions: Summary

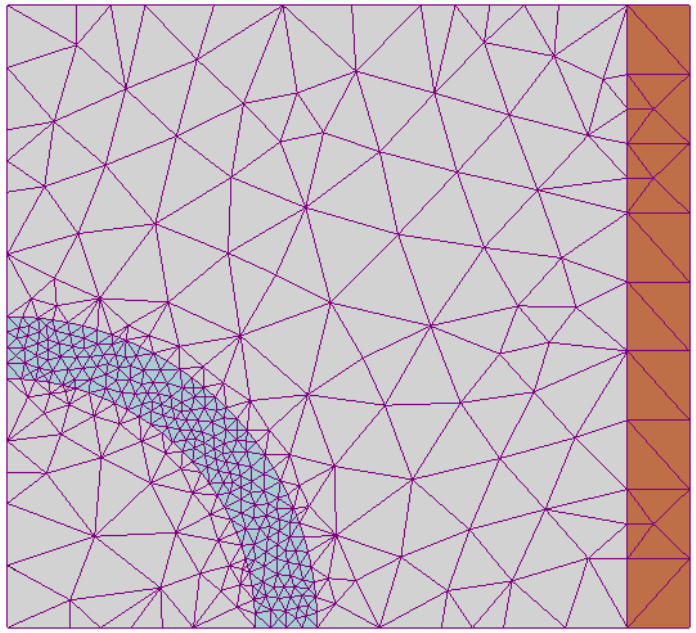


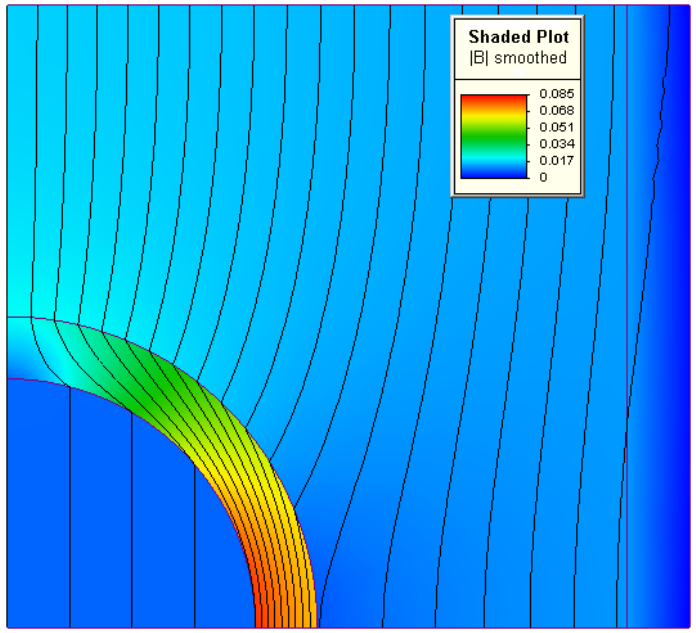
The paper introduces the **GNS (Graph Network-based Simulator)** architecture for learning to simulate particle systems:

1. High-quality general-purpose learned simulator
2. Strong generalization to larger domains, different initial conditions, setups etc
3. Paves the way for high-performance science and engineering applications

Limitations

1. Limitations – No Leverage on FEM Method



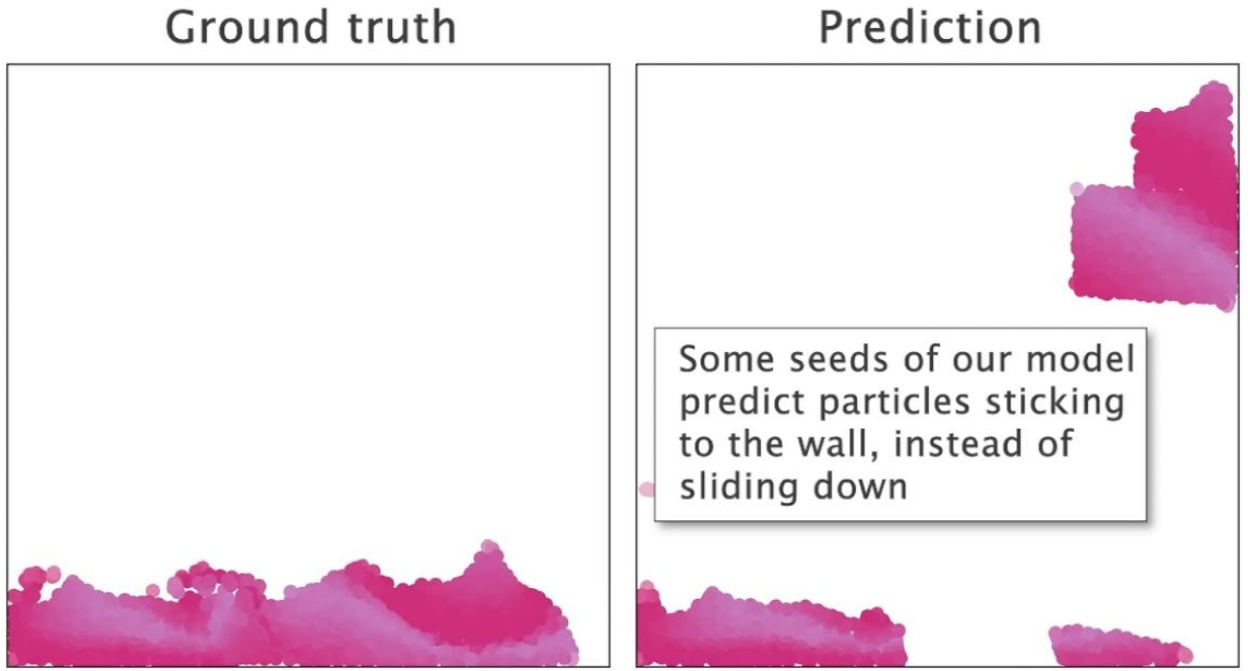


The Finite Element Method (FEM) inherently creates a graph (*mesh*) in order to solve PDEs (Partial Differential Equations)

→ The method could be much more suitable for such inherently-geometrical problems!

Images source: Wikipedia

2. Limitations – Handling Elasticity



Elasticity is problematic: particle interactions with this property are not easy to learn and are briefly mentioned in the paper

→ Problem: no variables explicitly describing physical states, such as energies!

3. Limitations - GNS Time per Step

Domain	Simulator (Dim.)	Mean # particles per graph (approx)	Mean # edges per graph (approx)	Simulator time per step [s]	Learned GNS time per step including neighborhood computation [s] (relative to simulator)
WATER-3D	SPH (3D)	7.8k	110k	0.104	0.358 (345%)
SAND-3D	MPM (3D)	9.8k	140k	0.221	0.336 (152%)
GOOP-3D	MPM (3D)	7.8k	120k	0.199	0.247 (124%)
WATER-3D-S	SPH (3D)	3.8k	55k	0.053	0.0683 (129%)
BOXBATH	PBD (3D)	1k	15k	–	0.0475 (–)
WATER	MPM (2D)	1.1k	12k	0.037	0.0579 (156%)
SAND	MPM (2D)	1.2k	11k	0.045	0.048 (107%)
GOOP	MPM (2D)	1k	9.2k	0.04	0.0301 (75.1%)
MULTIMATERIAL	MPM (2D)	1.6k	16k	0.049	0.0595 (121%)
FLUIDSHAKE	MPM (2D)	1.3k	13k	0.039	0.0257 (65.8%)
FLUIDSHAKE-BOX	MPM (2D)	1.4k	13k	0.048	0.133 (277%)
WATERDROP	MPM (2D)	0.6k	4.8k	0.05	0.0256 (51.3%)
WATERDROP-XL	MPM (2D)	4.3k	83k	0.166	0.192 (116%)
WATERRAMPS	MPM (2D)	1.5k	13k	0.071	0.0506 (71.3%)
SANDRAMPS	MPM (2D)	2.3k	21k	0.077	0.0691 (89.8%)
RANDOMFLOOR	MPM (2D)	2.3k	24k	0.076	0.0634 (83.4%)
CONTINUOUS	MPM (2D)	2.4k	22k	0.072	0.0919 (128%)

Usually more expensive than the actual simulator!

GNS Time per Step – No Neighborhood Computation

Domain	# particles in batch	# edges in batch	Learned GNS time per step without neighborhood computation [s] (relative to learned GNS in previous table)[8]
WATER-3D	14k	245k	0.071 (19.8%)
SAND-3D	19k	320k	0.086 (25.6%)
GOOP-3D	15k	230k	0.109 (44.2%)
WATER-3D-S	6k	120k	0.04 (58.6%)
BOXBATH	1k	18k	0.017 (35.8%)
WATER	2k	31k	0.025 (43.2%)
SAND	2k	21k	0.018 (37.5%)
GOOP	2k	21k	0.019 (63.2%)
MULTIMATERIAL	2k	27k	0.018 (30.3%)
FLUIDSHAKE	1.4k	23k	0.017 (66.2%)
FLUIDSHAKE-BOX	1.5k	20k	0.019 (14.3%)
WATERDROP	2k	18k	0.023 (89.7%)
WATERDROP-XL	8k	300k	0.057 (29.7%)
WATERRAMPS	2.5k	28k	0.017 (33.6%)
SANDRAMPS	3.5k	35k	0.023 (33.3%)
RANDOMFLOOR	3.5k	46k	0.023 (36.3%)
CONTINUOUS	5k	50k	0.033 (35.9%)

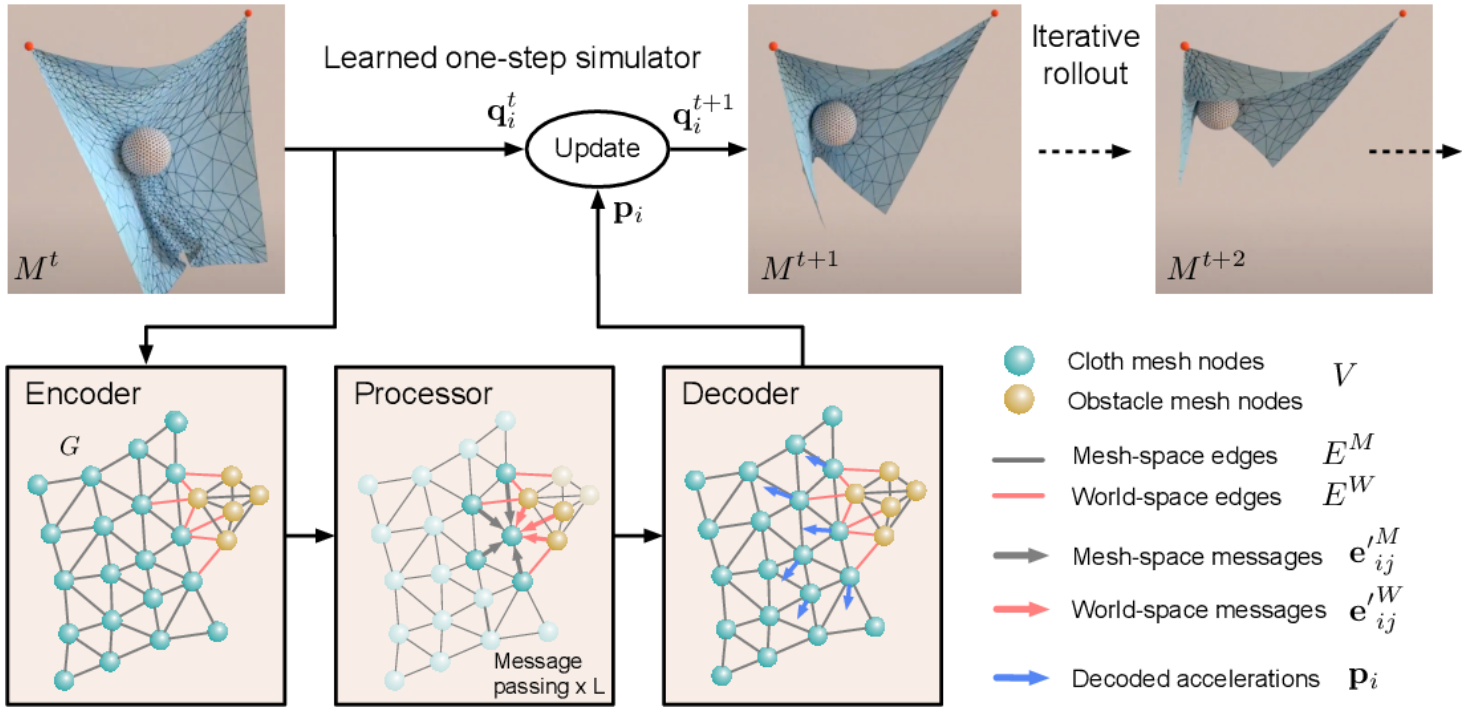
Graph creation with neighborhood computation seems to be the most expensive part!

Future Work and Research Ideas

1. Solutions – Follow Up Paper

- The follow-up work deals with mesh-based learned simulations
- Inference time is much better in this setup[1]

→ Don't worry, this paper will be covered by another student in this course! ☺

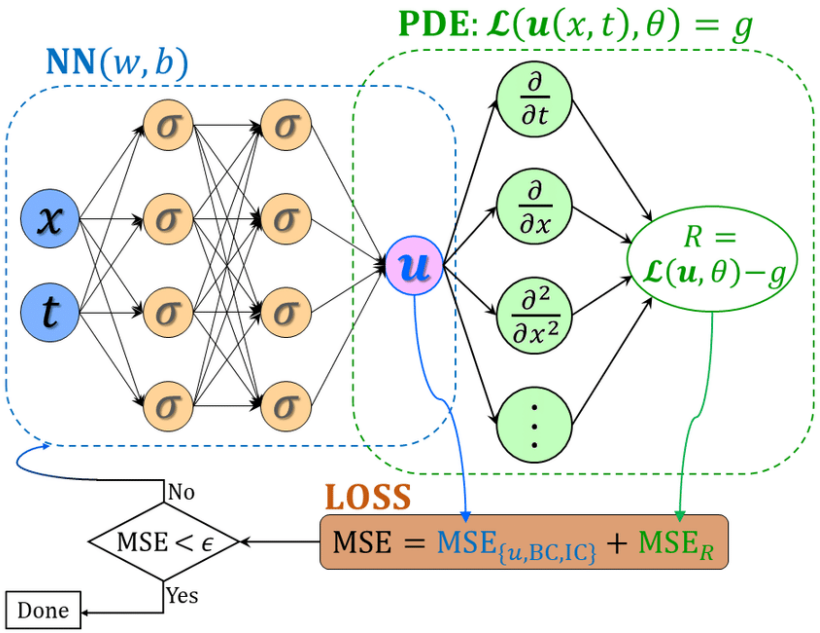


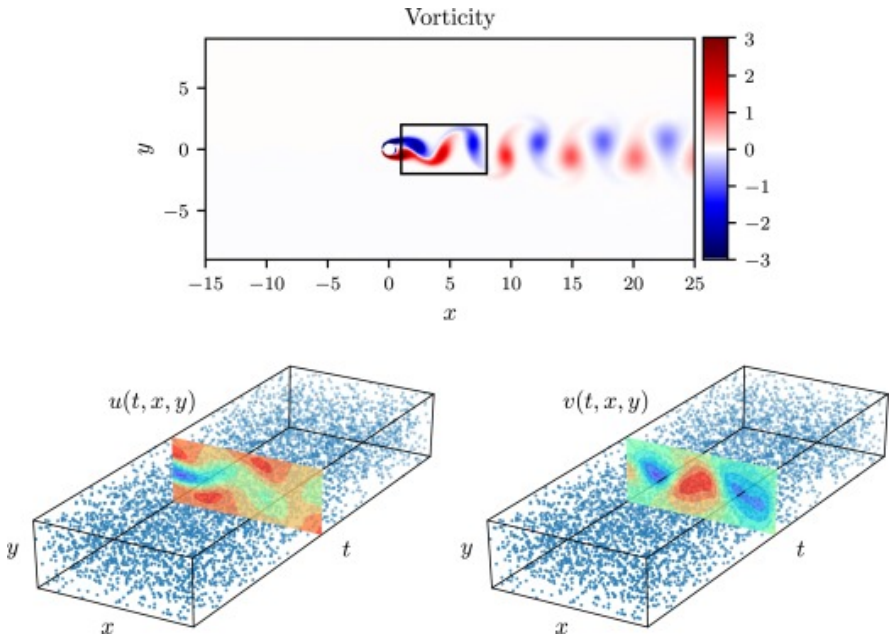
[1] Pfaff, Tobias, et al. "Learning mesh-based simulation with graph networks." *arXiv preprint arXiv:2010.03409* (2020).

2. Ideas for Potential Solution – Enforcing Physical Laws

Dealing with certain problems such as elasticity requires e.g. energy conservation. We can deal with it by:

- Adding a *memory* to the graph structure so that certain information is not lost
- Enforcing conditions based on a domain knowledge such as in PINNs[1]

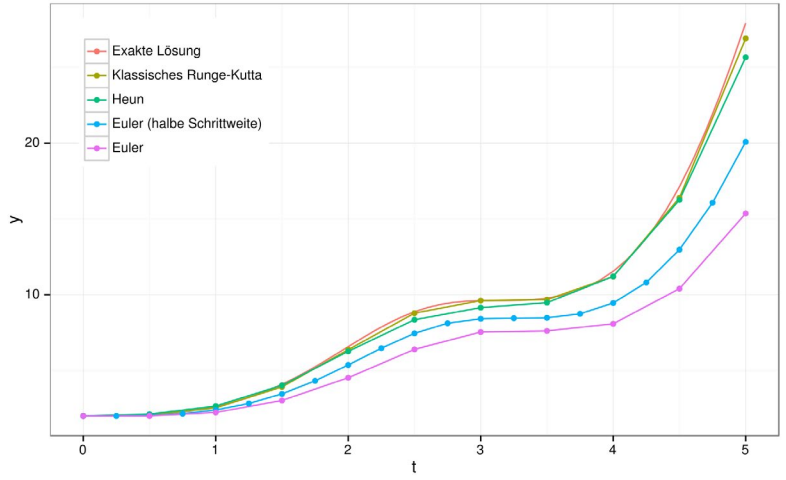


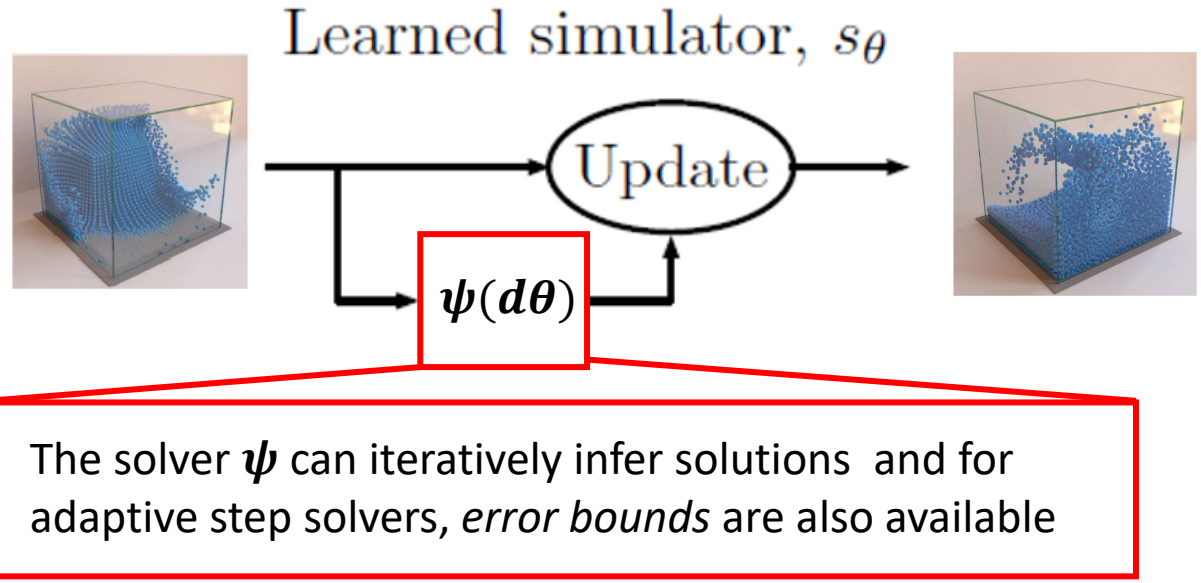


[1] Raissi, Maziar, Paris Perdikaris, and George E. Karniadakis. "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations." *Journal of Computational Physics* 378 (2019): 686-707.

3. Ideas for Potential Solution – Adaptive Step-Size Integrators

We still have a major problem to tackle: creating the graphs on the fly takes a ton of computations!
However, latent graph inference is relatively “cheap”
→ We could resort to other integration schemes such as adaptive-step solvers [1]
→ Result: larger step size (less graph creation steps), cheaper inference steps!





[1] Dormand, John R., and Peter J. Prince. "A family of embedded Runge-Kutta formulae." *Journal of computational and applied mathematics* 6.1 (1980): 19-26.

Thanks for your attention!

Presented by: Federico Berto (20204817)
Date: 2021-10-28