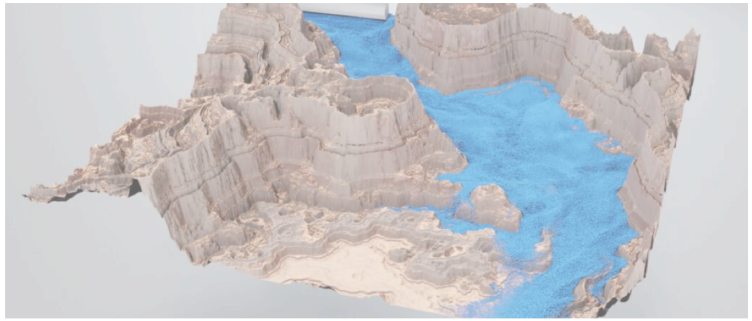
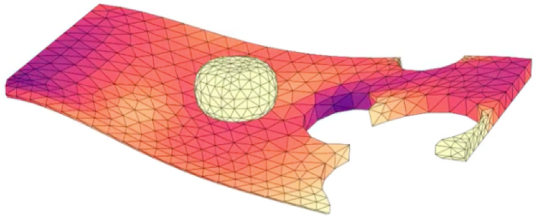
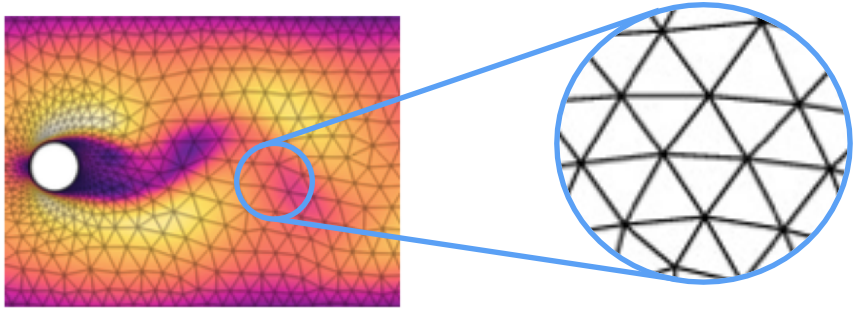
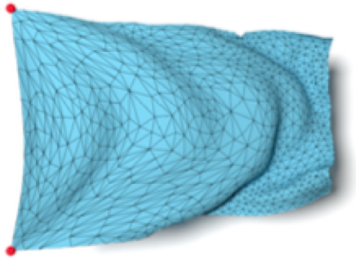




GNNs for Particle- and Mesh-Based Simulation (II)





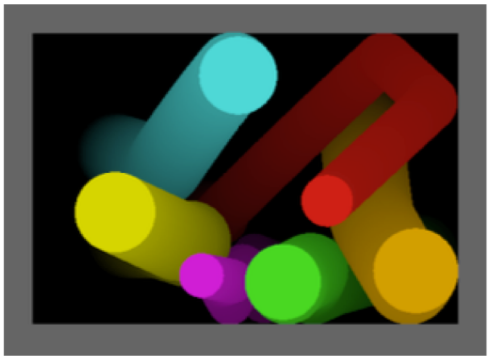
Dr. Mario Lino, m.lino@tum.de

Particle-based simulation with GNNs

Topics that we will cover:

1. Interaction Networks for Learning about Objects, Relations and Physics
2. Learning to Simulate Complex Physics with Graph Networks
3. Lagrangian Fluid Simulation with Continuous Convolutions
4. Inverse Design for Fluid-Structure Interactions using Graph Network Simulators
5. Guaranteed Conservation of Momentum for Learning Particle-based Fluid Dynamics

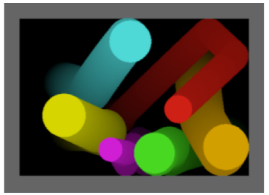
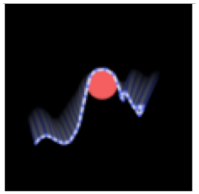
Interaction Networks for Learning about Objects, Relations and Physics



Battaglia et al., 2016

Interaction Networks for Learning about Physics

- First work to represent systems of particles as graphs:
  - The nodes represent the particles
  - The edges represent their relationship (e.g., springs, gravitational attraction, distance to collision)

- The message function models the interaction between particles.
- The node-update function models particle dynamics.
- Experimented with: N-body problem, rigid-body collision and discretised string.
- Proved extrapolation to larger systems and thousands of future states.
- Scale to up to 12 particles.

Interaction Networks for Learning about Physics

- Numerical models for particle systems account for **the interactions between particles** and **how these impact the state of each particle**.
- For example, in the the N-body problem:
  - Every particle j attracts each particle i:

$$\ddot{\boldsymbol{x}}_i^n = \sum_{j \neq i} \frac{G\, m_j}{|\boldsymbol{x}_j^n - \boldsymbol{x}_i^n|^3} (\boldsymbol{x}_j^n - \boldsymbol{x}_i^n)$$

Aggregation **Message**

  - The position of each particle is updated based on their current state and this interaction (**node update**):

$$\boldsymbol{x}_i^{n+1} = \boldsymbol{x}_i^n + \Delta t \dot{\boldsymbol{x}}_i^n + \Delta t^2 \ddot{\boldsymbol{x}}_i^n / 2$$

- These dynamics could be modelled by a Message Passing layer.

Interaction Networks for Learning about Physics

Experiments

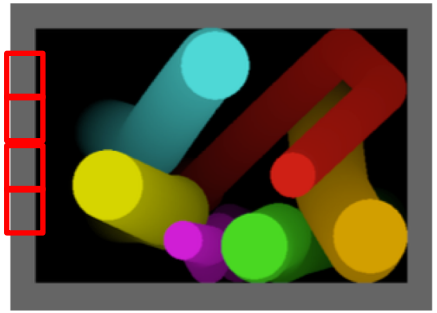
- N-body problem:
  - $|V|$ = N
  - $|E|$ = $|V|$ x ($|V|$ - 1)
  - Input node attributes: velocity and mass
  - Input edge attributes: relative position
  - Training with $|V|$ = 6, testing with $|V|$ = 3, 6 and 9

* The inputs are translation invariant

Interaction Networks for Learning about Physics

Experiments

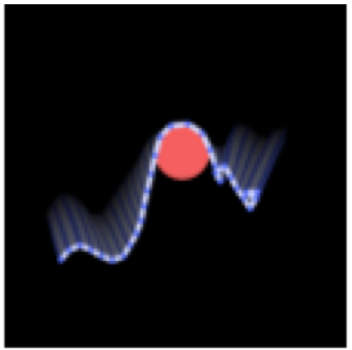
- Bouncing balls:
  - $|V|$ = Number of balls + Number of rectangles discretising the walls
  - $|E|$ = $|V|$ x ($|V|$ - 1)
  - Input node attributes: velocity, inverse of the mass, radius and node type
  - Input edge attributes: relative position and coefficient of restitution
  - Training with 6 balls and testing with 3, 6 and 9 balls



Interaction Networks for Learning about Physics

Experiments

- Discretised string:
  - $|V|$ = Number of nodes on the string and one central ball
  - $|E|$ = 2 x ($|V|$ - 2) + 2 ($|V|$ - 1) (spring edges + spring-obstacle edges)
  - Input node attributes: velocity, inverse of the mass, gravity and node type
  - Input edge attributes: relative position and relation type



Interaction Networks for Learning about Physics

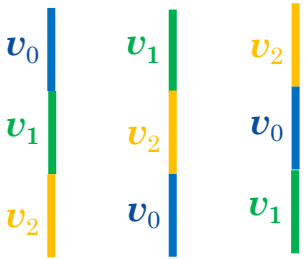
Experiments

- The output is the velocity at the next time-point, $\dot{\boldsymbol{x}}_i^{n+1}$
- Trained to minimise the MSE of $\dot{\boldsymbol{x}}_i^{n+1}$
- This velocity is used to update the position: $\boldsymbol{x}_i^{n+1} = \boldsymbol{x}_i^n + \Delta t \dot{\boldsymbol{x}}_i^{n+1}$
- Rollouts are produced by updating the input attributes and evaluating iteratively the model.
- The simulations were rolled out for thousands of time-steps during training
- **Message function**: MLP with four hidden layer (150 neurons each) and 50 neurons in the output layer.
- **Node-update function**: MLP with one hidden layer (with 100 neurons) and 2 neurons in the output layer (x and y component of the velocity).

Any baseline to compare with?

GNNs vs MLPs

- MP-based GNNs does not constrain the size of the system to a fix number of nodes.
- MP is permutation invariant. MLPs are not: the feature vectors can be stacked in many different orders. For instance, these three input vectors represent the same system, but nothing guarantees that the velocity predicted for each node is the same in every case.



- MP is translation invariant if we use the relative position between nodes as edge input.
- Note: in MP, the result from the aggregation must be permutation invariant. Thus, the aggregated message cannot be directly computed, for instance, like this:

$$\bar{\boldsymbol{m}}_0 \leftarrow \text{MLP}\left([\boldsymbol{v}_0|\boldsymbol{v}_1|\boldsymbol{e}_{10}|\boldsymbol{v}_2|\boldsymbol{e}_{20}]\right) \quad \text{or} \quad \bar{\boldsymbol{m}}_0 \leftarrow \text{MLP}\left([\boldsymbol{v}_0|\boldsymbol{v}_2|\boldsymbol{e}_{20}|\boldsymbol{v}_1|\boldsymbol{e}_{10}]\right)$$

Interaction Networks for Learning about Physics

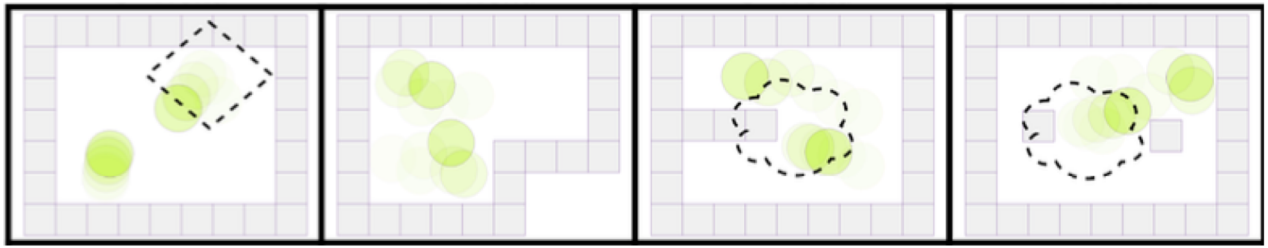
Experiments

- Baseline: MLP which took as input a flattened vector of all of the input data.
- The MLP works only with a fixed number of particles and edges

Domain	Constant velocity	Baseline	Dynamics-only IN	IN
n-body	82	79	76	**0.25**
Balls	0.074	0.072	0.074	**0.0020**
String	0.018	0.016	0.017	**0.0011**

Interaction Networks for Learning about Physics

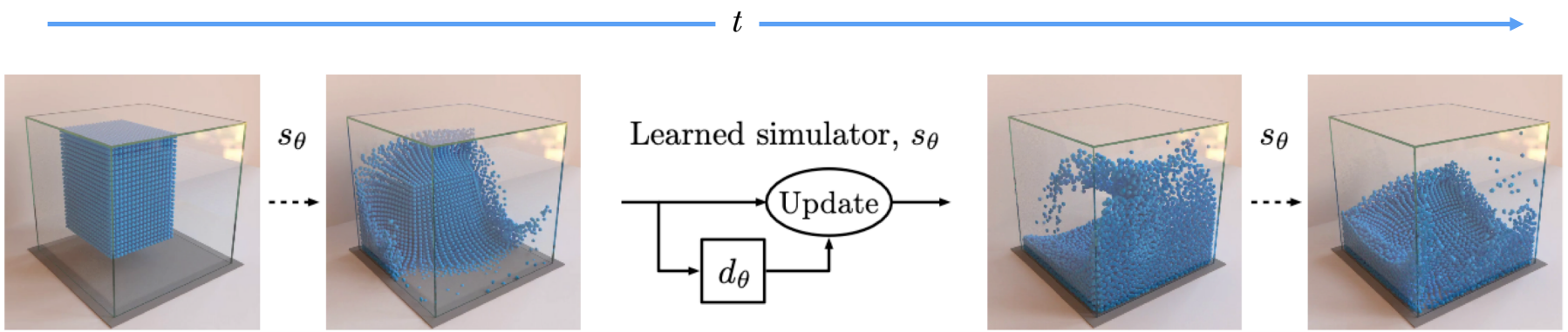
- Similar work: *A Compositional Object-Based Approach to Learning Physical Dynamics,* Chang et al., 2016.
- They arrived to similar conclusions independently.



Note on Time Predictions

- When evaluating a model autoregressively to produce rollouts the error accumulates and the simulation can divergence in a few time-steps (network evaluations).
- In this work, they mitigated this by adding Gaussian noise to the input positions and velocities (similar to label smoothing LLM).

Learning to Simulate Complex Physics with Graph Networks



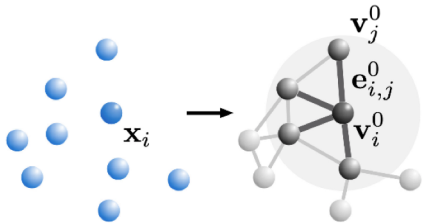
Sanchez-Gonzalez et al., 2020

Learning to Simulate Complex Physics with GNs

- Scaled the previous work to >10k nodes/particles.
- This allowed to simulate continuous systems – fluids, sand and goop – discretised into thousands of Lagrangian particles (Similarity with Smoothed Particle Hydrodynamics).
- This required ~10 MP layers.

Learning to Simulate Complex Physics with GNs

- The edges were creating by directing an edge to each node from all the nodes closer than a distance R (hyperparameter).

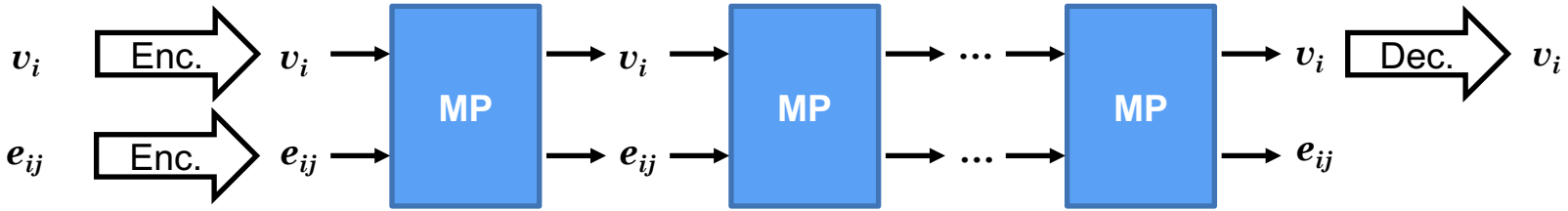


- Input node-features: velocity at the last 5 time-points, material type (water, sand, goop or boundary).
- Input edge-features: relative position, $\boldsymbol{x}_i - \boldsymbol{x}_j$, and distance , $\|\boldsymbol{x}_i - \boldsymbol{x}_j\|$.
- Output edge features: acceleration at the current point, $\ddot{\boldsymbol{x}}_i^n$.
- The velocities and positions are updated according to:

$$\dot{\boldsymbol{x}}_i^{n+1} = \dot{\boldsymbol{x}}_i^n + \Delta t \cdot \ddot{\boldsymbol{x}}_i^n \qquad \boldsymbol{x}_i^{n+1} = \boldsymbol{x}_i^n + \Delta t \cdot \dot{\boldsymbol{x}}_i^{n+1}$$

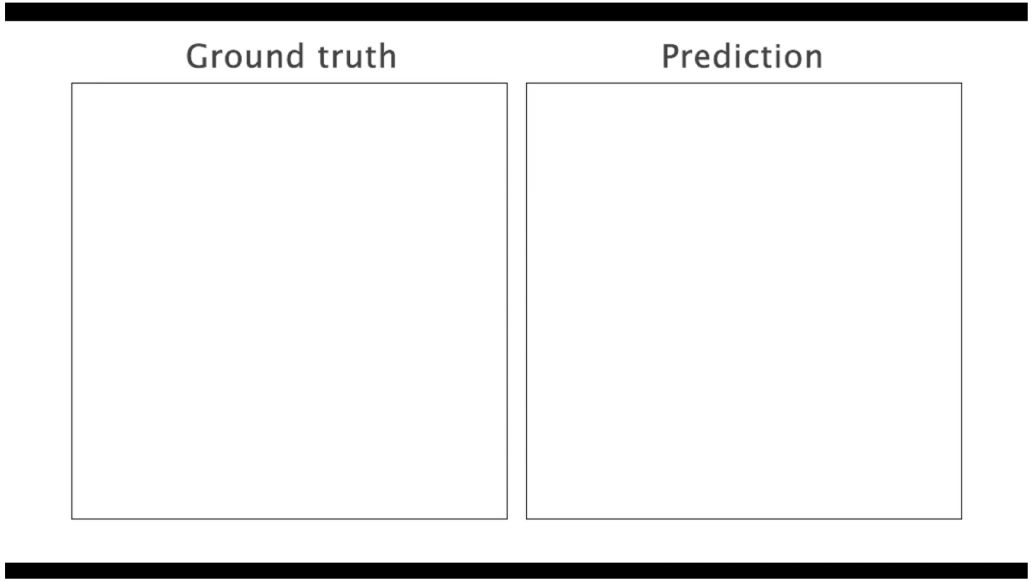
Learning to Simulate Complex Physics with GNs

- This GNN architecture has become widely used for particle based simulation:
  - Node encoder: an **MLP or linear layer** applied to each input node-features. It projects the node features to a 128-dimensional space.
  - Edge encoder: an **MLP or linear layer** applied to each input edge-features.
  - Propagator: $\boldsymbol{M}$ **sequential MP layers**. They employed MP layer with edge update. The node and edge functions are MLPs with 2 hidden layers and 128 neurons, followed by Layer Normalisation.
  - Node decoder: an **MLP or linear layer**. It projects the node features to 2 or 3-dimensional space (components of the acceleration).



Learning to Simulate Complex Physics with GNs

- Training for one-step predictions (MSE of acceleration) on systems with $O(10^3)$ nodes.
- Inference rollouts with length $O(10^3)$ and $O(10^4)$ nodes.



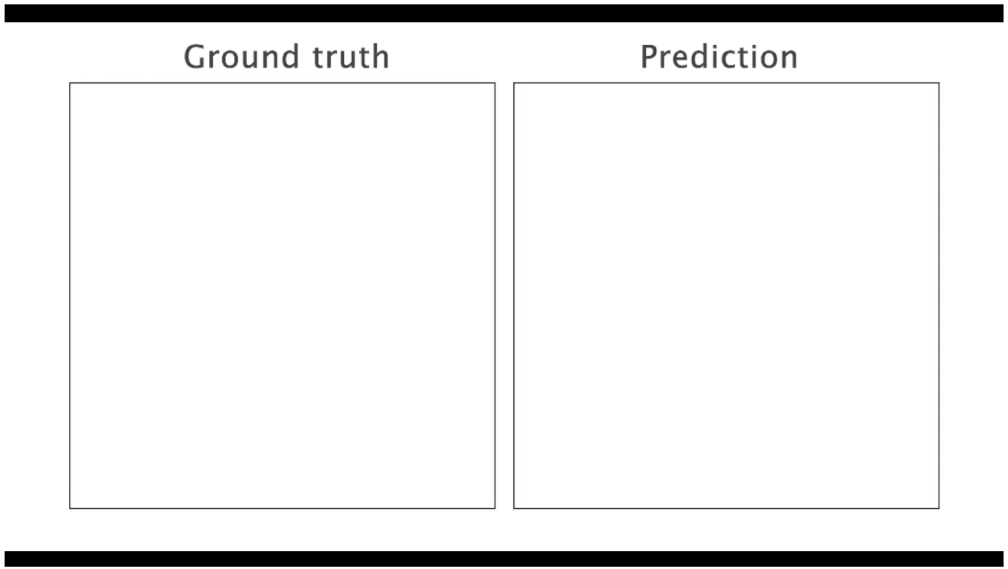
Training:
- 1x1 domain
- 2.5k particles
- 600 individual steps

Inference:
- 2x2 domain
- 28k particles
- 2500-steps rollout

Motivated by: compositional approach of MP, weight sharing between nodes and edges, translation invariance, noise-added to input positions and velocities.

Learning to Simulate Complex Physics with GNs

- Also learned the interactions between different types of materials.



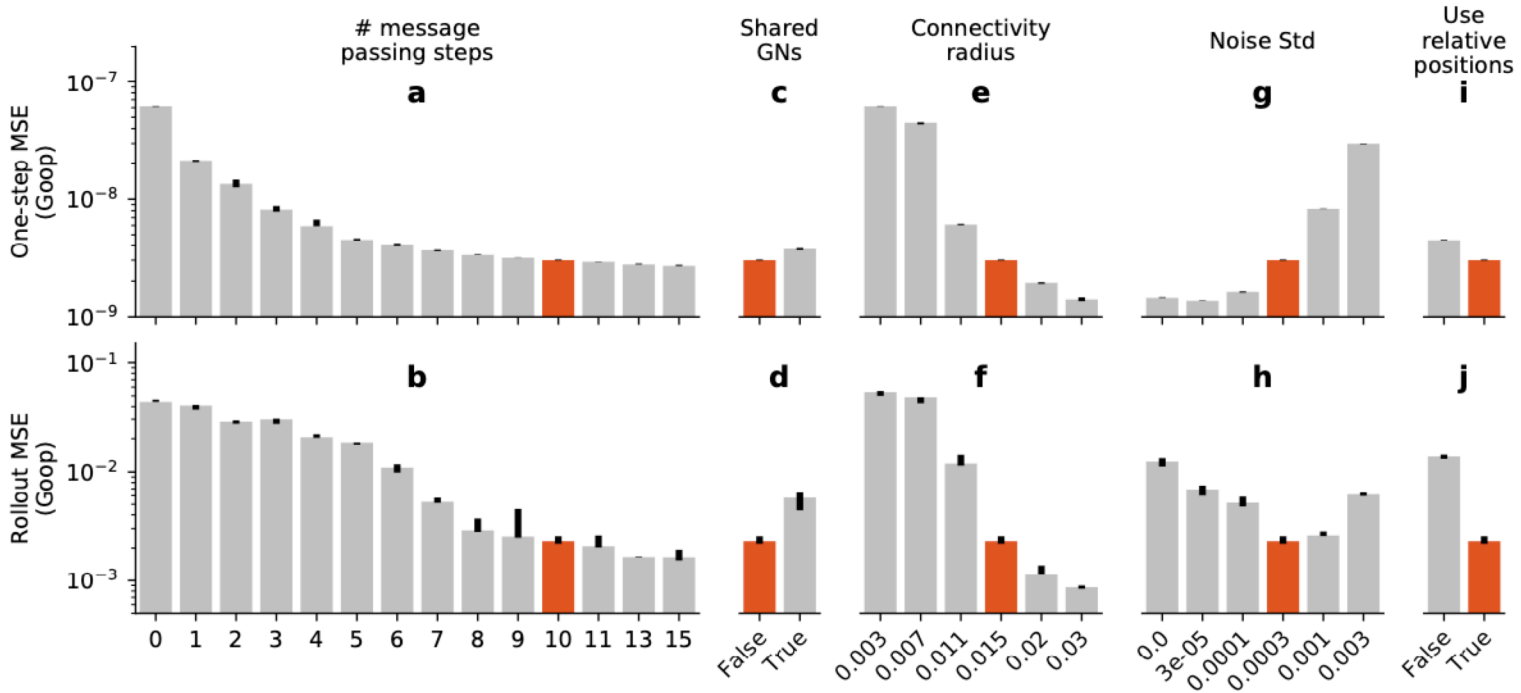
Training:
- 2k particles
- 1000 individual steps

Inference:
- 4.5k particles
- 2000-steps rollout

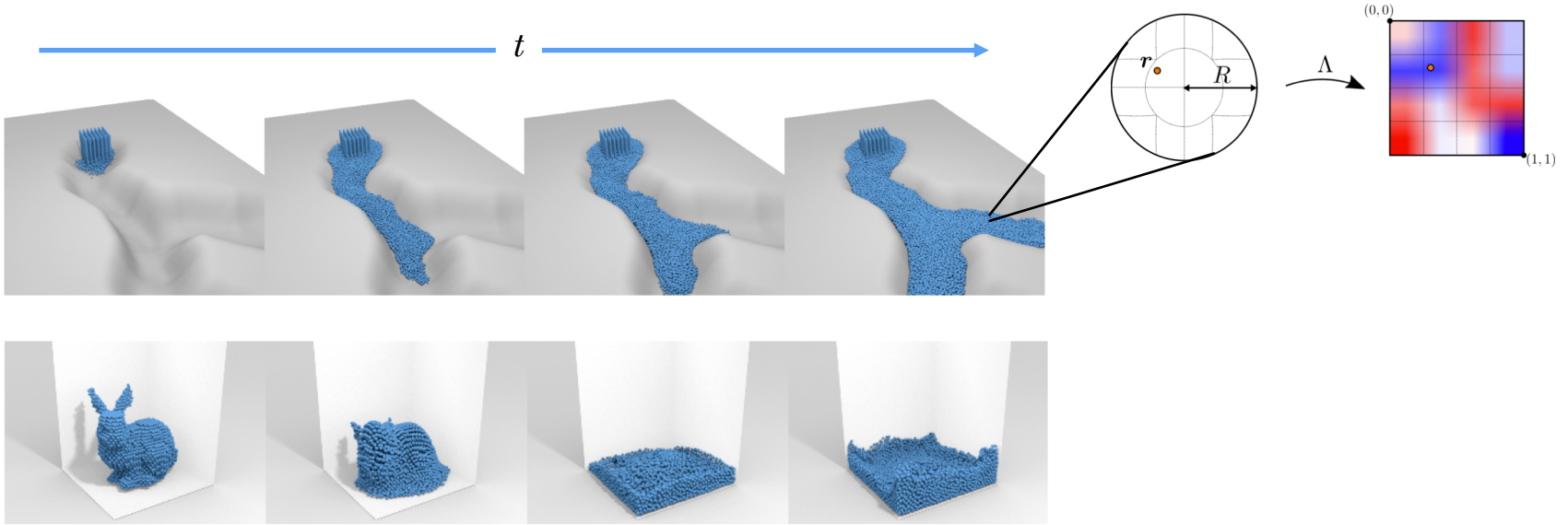
Motivated by: compositional approach of MP

Learning to Simulate Complex Physics with GNs

- Ablations on the one-steps and the rollout error:

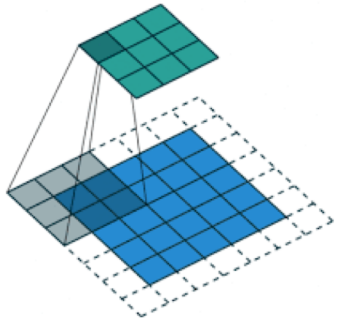


Lagrangian Fluid Simulation with Continuous Convolutions



Ummenhofer et al., 2020

Lagrangian Fluid Simulation with CConvs



- Discrete convolution: Discrete input f and filter g

$$(f * g)(\boldsymbol{x}) = \sum_{\tau \in \Omega} f(\boldsymbol{x} + \tau) g(\tau)$$

- Continuous convolution: Continuous input f and filter g

$$(f * g)(\boldsymbol{x}) = \int_{\Omega} f(\boldsymbol{x} + \tau) g(\tau)\, d\tau$$

- This work adapts continuous convolutions to unstructured point clouds: The input is defined on a finite number of points that do not lie on a grid and the filter is a continuous function defined on a sphere of radius R (compact support).

$$(f * g)(\boldsymbol{x}_j) = \sum_{i \in \mathcal{N}_j^-} a\left(||\boldsymbol{x}_i - \boldsymbol{x}_j||_2^2\right)\ f_i\ g\left(\Lambda(\boldsymbol{x}_i - \boldsymbol{x}_j)\right)$$

Lagrangian Fluid Simulation with CConvs

Elements of their Continuous Convolution (CConv)

Neighbour attribute

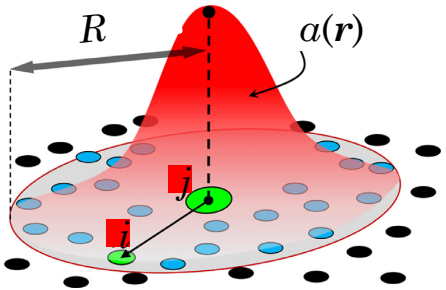
$$(f * g)(\boldsymbol{x}_j) = \sum_{i \in \mathcal{N}_j^-} a\left(||\boldsymbol{x}_i - \boldsymbol{x}_j||_2^2\right)\ f_i\ g\left(\Lambda(\boldsymbol{x}_i - \boldsymbol{x}_j)\right)$$

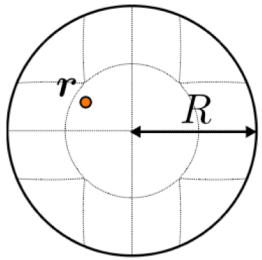
Spherical filter

Mapping from a sphere to a cube

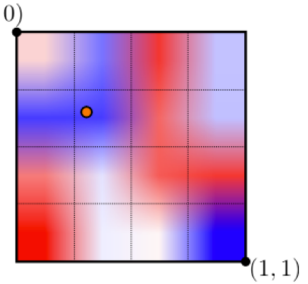
$$a(r) = \left(1 - \frac{r^2}{R^2}\right)^3$$

It ensure a smooth response under varying particle neighborhoods





Λ



The weights are stored on a grid on the cube and interpolated on the rest of point

$$\Lambda_{\text{ball}\to\text{cyl}}(\mathbf{r}) = \begin{cases} (0,0,0) & \text{if } \|\mathbf{r}\|_2 = 0 \\ \left(x\frac{\|\mathbf{r}\|_2}{\|(x,y)\|_2},\ y\frac{\|\mathbf{r}\|_2}{\|(x,y)\|_2},\ \frac{3}{2}z\right) & \text{if } \frac{5}{4}z^2 \le x^2 + y^2 \\ \left(x\sqrt{\frac{3\|\mathbf{r}\|_2}{\|\mathbf{r}\|_2+|z|}},\ y\sqrt{\frac{3\|\mathbf{r}\|_2}{\|\mathbf{r}\|_2+|z|}},\ \text{sign}(z)\|\mathbf{r}\|_2\right) & \text{else.} \end{cases}$$

$$\Lambda_{\text{cyl}\to\text{cube}}(\mathbf{r}) = \begin{cases} (0,0,z) & \text{if } x = 0, y = 0 \\ \left(\text{sign}(x)\|(x,y)\|_2,\ \frac{4}{\pi}\,\text{sign}(x)\|(x,y)\|_2 \arctan\frac{y}{x},\ z\right) & \text{if } |y| \le |x| \\ \left(\frac{4}{\pi}\,\text{sign}(y)\|(x,y)\|_2 \arctan\frac{x}{y},\ \text{sign}(y)\|(x,y)\|_2,\ z\right) & \text{else.} \end{cases}$$

Lagrangian Fluid Simulation with CConvs

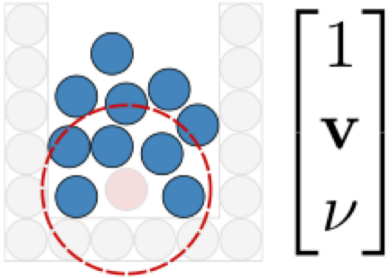
- The CConv is inspired by the SPH discretisation: $f_j = \sum_{i \in \mathcal{N}_j^-} V_i \; f_i \; W\left(||\boldsymbol{x}_i - \boldsymbol{x}_j||_2^2\right)$
- Multiple input and output channels were used ($G : \mathbb{R}^3 \rightarrow \mathbb{R}^{C_{\text{in}} \times C_{\text{out}}}$) and a linear layer was added for the central node:

$$\boldsymbol{v}_j \leftarrow W\boldsymbol{v}_j + \sum_{i \in \mathcal{N}_j^-} a\left(||\boldsymbol{x}_i - \boldsymbol{x}_j||_2^2\right) \; \boldsymbol{v}_i \; G\left(\Lambda(\boldsymbol{x}_i - \boldsymbol{x}_j)\right)$$

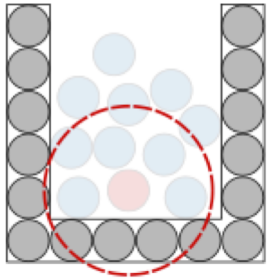
- This can be written as message passing.
- Hyperparameters: number of channels, R and the kernel resolution.
- As opposed to the work in Sanchez-Gonzalez et al., the edges are not updated and MP is linear.
- This makes CConv more efficient but also less accurate for a fixed number of MP steps and radius R.

Lagrangian Fluid Simulation with CConvs

- The external forces (i.e., gravity) are directly applied to each node and the GNN processes only the internal forces.
- The use heterogeneous graphs with two types of nodes:
  1. Fluid-type nodes. Attributes: velocity and viscosity.
  2. Solid-type nodes. Attributes: wall normal vector.



$$\begin{bmatrix} 1 \\ \mathbf{v} \\ \nu \end{bmatrix}$$



$$\begin{bmatrix} \mathbf{n} \end{bmatrix}$$

- There are not edge attributes.

Lagrangian Fluid Simulation with CConvs

Time stepping:

1. Compute intermediate positions and velocities by applying the external forces:

$$\mathbf{v}_i^{n*} = \mathbf{v}_i^n + \Delta t\, \mathbf{a}_{\text{ext}}$$

$$\mathbf{x}_i^{n*} = \mathbf{x}_i^n + \Delta t\, \frac{\mathbf{v}_i^n + \mathbf{v}_i^{n*}}{2}$$

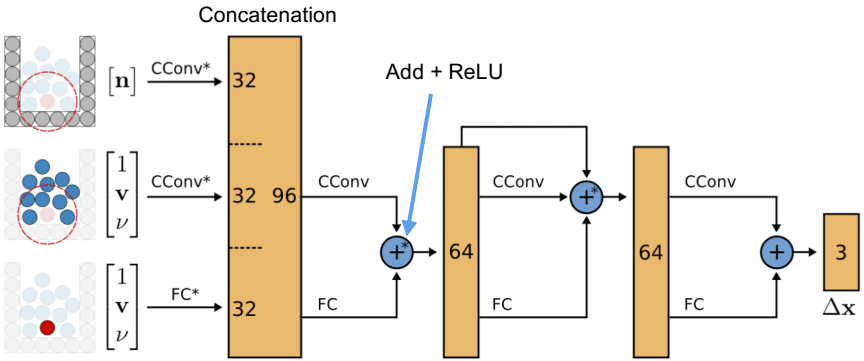
2. Evaluate the GNN with this intermediate positions and velocities to obtain as output a position correction, $\Delta \boldsymbol{x}$. This correction accounts for the particle interactions.
3. Update the particles' position and velocities with this correction:

$$\mathbf{x}_i^{n+1} = \mathbf{x}_i^{n*} + \Delta \mathbf{x}_i$$

$$\mathbf{v}_i^{n+1} = \frac{\mathbf{x}_i^{n+1} - \mathbf{x}_i^n}{\Delta t}$$

Lagrangian Fluid Simulation with CConvs

Model architecture:



- Four CConv layers (C_{out} = 32, 64, 64 and 3) are applied sequentially.
- To account for the solid-type nodes, the first layer is modified:

$$\boldsymbol{v}_j \leftarrow \left[W\boldsymbol{v}_j \,\middle|\, \sum_{i \in \mathcal{N}_j^{-},F} a\left(||\boldsymbol{x}_i - \boldsymbol{x}_j||_2^2\right)\, \boldsymbol{v}_i\, G^f\left(\Lambda(\boldsymbol{x}_i - \boldsymbol{x}_j)\right) \,\middle|\, \sum_{i \in \mathcal{N}_j^{-},S} a\left(||\boldsymbol{x}_i - \boldsymbol{x}_j||_2^2\right) \boldsymbol{v}_i\, G^s\left(\Lambda(\boldsymbol{x}_i - \boldsymbol{x}_j)\right) \right]$$

Lagrangian Fluid Simulation with CConvs

Loss function

$$\mathcal{L}^{n+1} = \sum_{i=1}^{N} \phi_i \left\| \mathbf{x}_i^{n+1} - \hat{\mathbf{x}}_i^{n+1} \right\|_2^{\gamma}$$

$$\phi_i = \exp(-\tfrac{1}{c}|\mathcal{N}(\mathbf{x}_i^{n*})|)$$

- Individual weight for each particle.
- It emphasizes the loss for particles with fewer neighbours.
- These are the most important: particles close to the surface or near the walls.
- c = 40 (average number of neighbours)

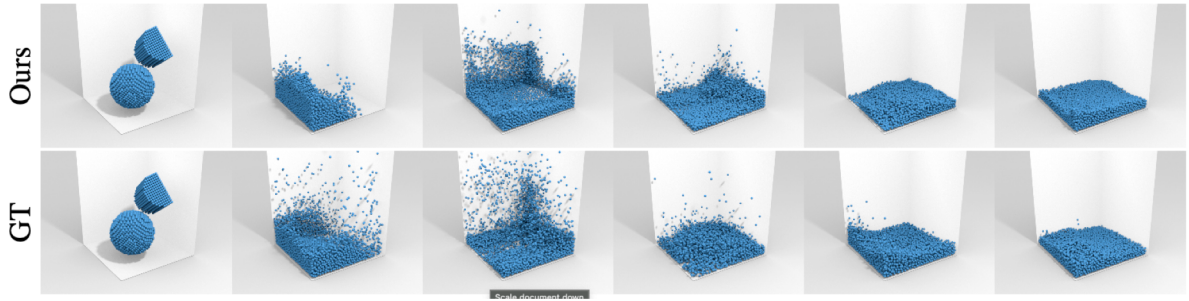
- γ = 0.5
- It makes the loss function more sensitive to small particle motions → improve visual fidelity for small fluid flows.

$$\mathcal{L} = \mathcal{L}^{n+1} + \mathcal{L}^{n+2}$$

During training the particles' positions is predicted for two timesteps.

Lagrangian Fluid Simulation with CConvs

- Trained with simulations on box-like containers.
- Tested with different initial location of the particles.

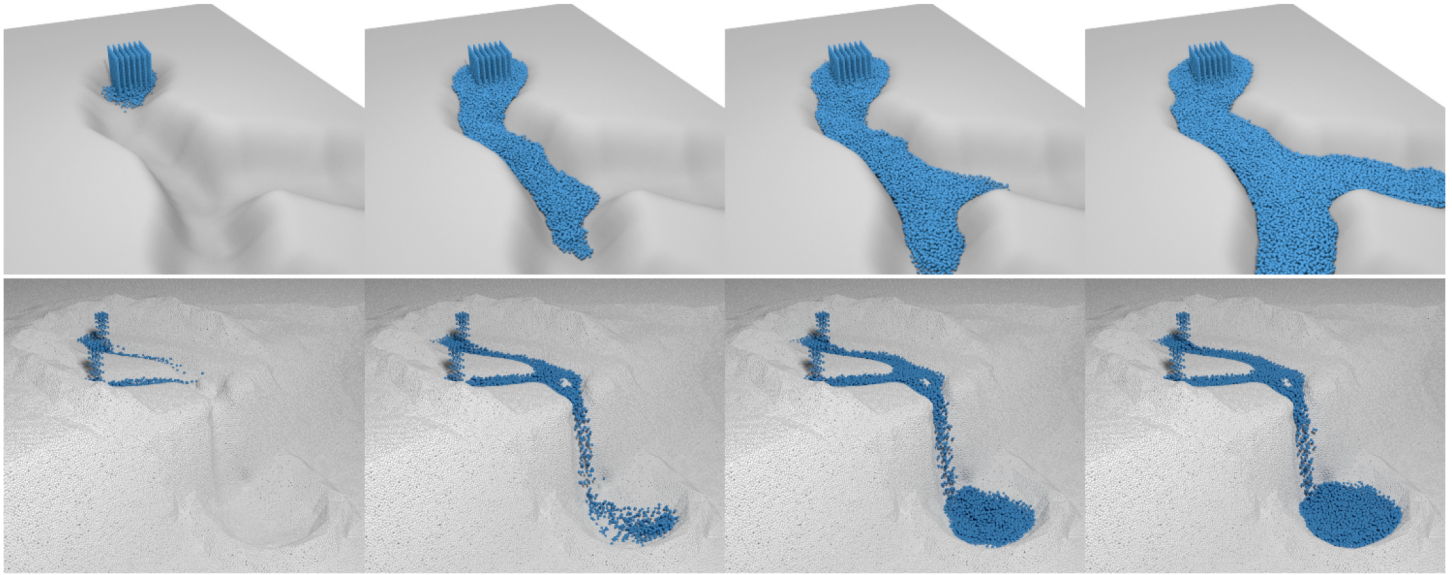


- Ablation study justifying design choices:

Method	Average error (mm)		Seq. end error (mm)		Average distance to
	$n+1$	$n+2$	$n+1$	$n+2$	closest point d^n (mm)
Ours	**0.67**	**1.87**	**0.25**	**0.74**	**30.63**
Ours w/o interpolation	0.79	2.24	0.30	0.89	32.39
Ours w/o window	0.77	2.21	0.30	0.89	31.77
Ours w/ naïve loss	0.69	**1.86**	0.27	0.77	**30.35**
Ours w/o FC	0.75	2.17	0.27	0.80	32.49

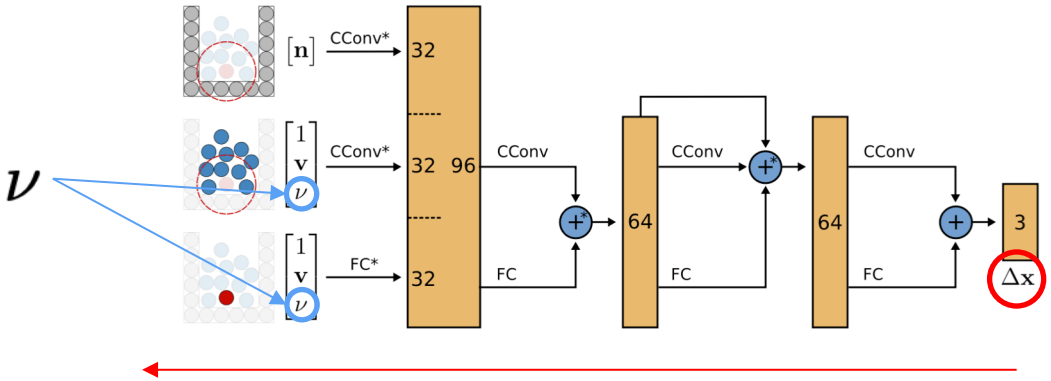
Lagrangian Fluid Simulation with CConvs

- Showed extrapolation to new viscosities and generalisation to non-box-contained flows.



Lagrangian Fluid Simulation with CConvs

- An advantage of DL-based solvers is that they are differentiable.
- This can be used to solve inverse problems.
- In this work they freeze the GNN weights, and, given the position of the particles at two consecutive time-points, backpropagation can be used to iteratively update the viscosity (node feature):

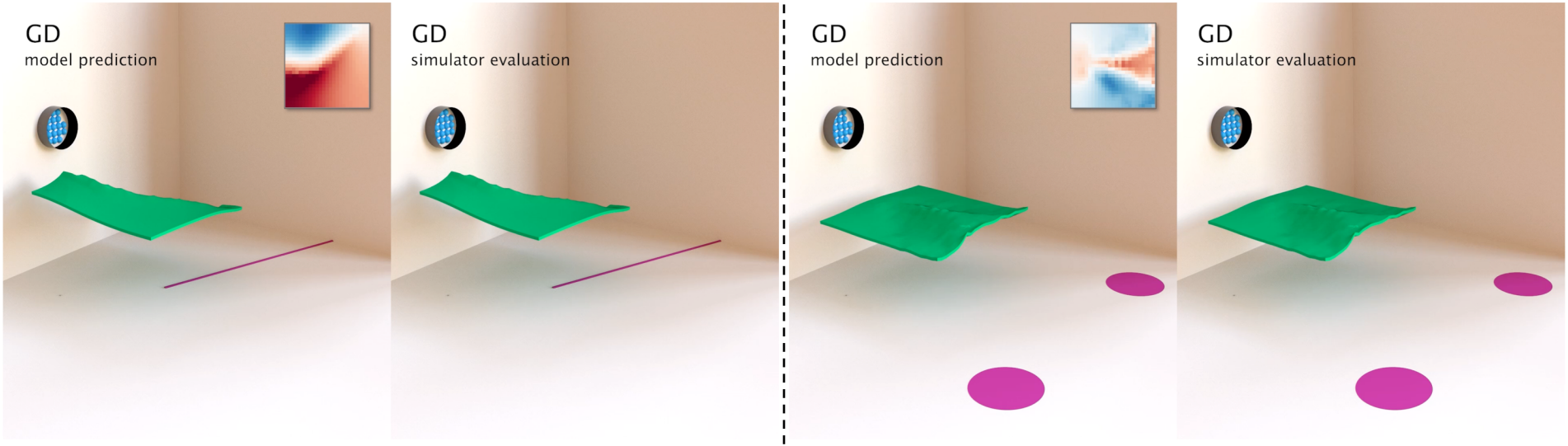


$$\nu \leftarrow \nu - \lambda \frac{\partial \mathcal{F}}{\partial \nu}$$

$$\mathcal{F} := \left| \Delta \boldsymbol{x}_{\text{observed}} - \Delta \boldsymbol{x}_{\text{GNN}} \right|$$

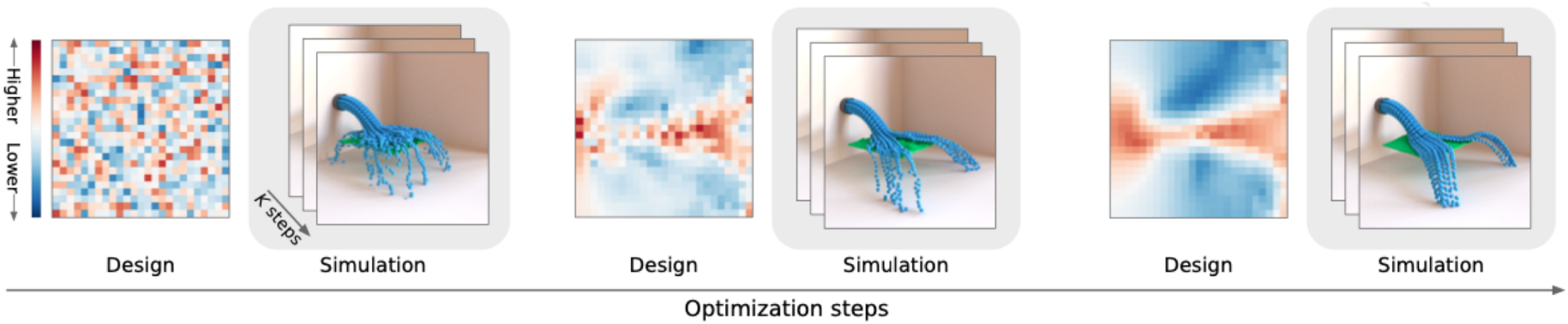
Inverse Design using GNN Simulators

- The application of GNNs to inverse problems was explored in deeper detail in *Inverse Design for Fluid-Structure Interactions using Graph Network Simulators* by Allen et al.



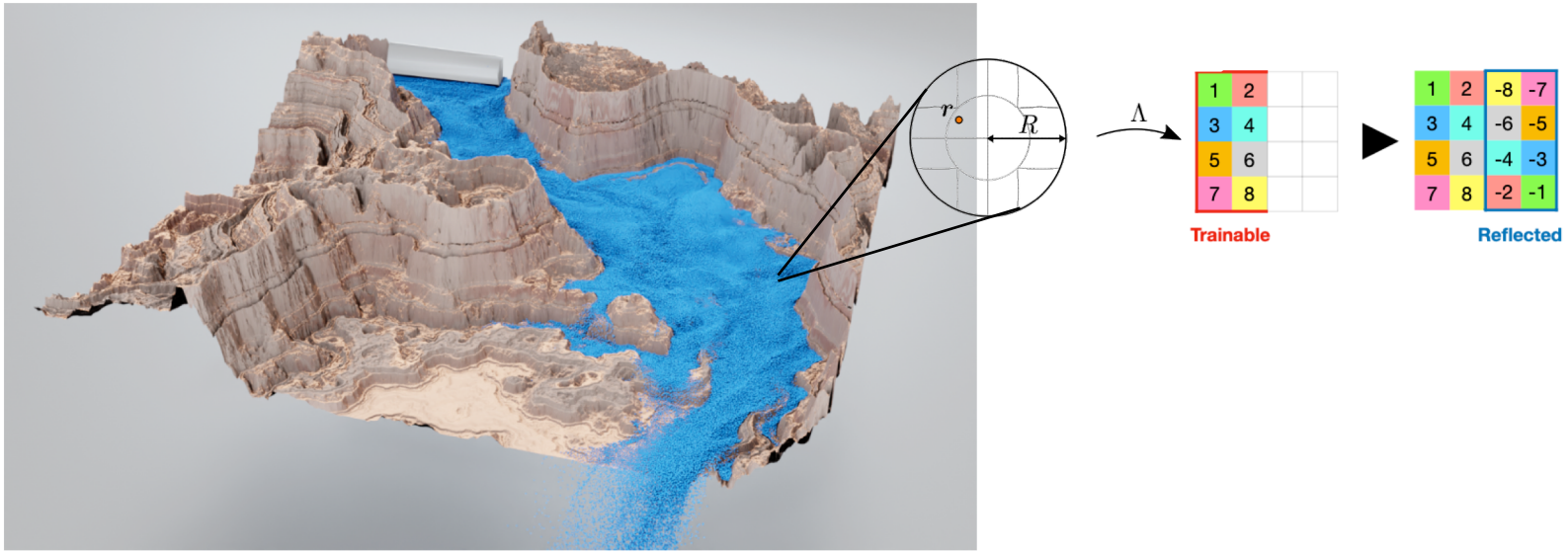
Inverse Design using GNN Simulators

- The *learnable parameters* are the elevation of the wall points.
- A simulation is ran for k steps. Then an objective function is evaluated.
- The gradients of the objective function are computed by backpropagation, which involves backpropagating trough the model multiple times.



- This gradient-based optimization converges in few design iterations but it is expensive.

Guaranteed Conservation of Momentum for Learning Particle-based Fluid Dynamics



Prantl et al., 2022

Guaranteed Conservation of Momentum

Some high-school physics…

- Momentum of a discrete system S: $\mathcal{P}_S = \sum_{i \in S} p_i = \sum_{i \in S} m_i \, v_i$

- The linear momentum of each particle i can be modified by the action of external forces and by the action of internal forces from other particles:

$$\delta p_i = \left(F_{\text{ext},i} + \sum_{j \in S} F_{j,i} \right) \delta t$$

- According to Newton's third law of motion (*action-reaction*), the sum of all the internal forces is zero

$$F_{j,i} = -F_{i,j} \rightarrow \sum_{i \in S} \sum_{j \in S} F_{j,i} = 0$$

Guaranteed Conservation of Momentum

$$F_{j,i} = -F_{i,j} \rightarrow \sum_{i \in S} \sum_{j \in S} F_{j,i} = 0$$

- So, **the total change in momentum of a system can only be caused external forces, not by internal forces:**

$$\delta \mathcal{P}_S = \sum_{i \in S} \delta p_i = \sum_{i \in S} \left(F_{\text{ext},i} + \sum_{j \in S} F_{j,i} \right) \delta t = \sum_{i \in S} F_{\text{ext},i} \ \delta t$$

- Strategy to conserve momentum: embed the action-reaction principle into the model
- Embedding physic principles into a DL model can help to use its weights more efficiently.

Guaranteed Conservation of Momentum

- As in *Lagrangian Fluid Simulation with CConvs,* the external forces (accelerations) are applied first and the obtained positions and velocities are corrected by the GNN.

$$\mathbf{v}_i^{n*} = \mathbf{v}_i^n + \Delta t\, \mathbf{a}_{\text{ext}} \qquad \mathbf{x}_i^{n+1} = \mathbf{x}_i^{n*} + \Delta \mathbf{x}_i$$

GNN output

$$\mathbf{x}_i^{n*} = \mathbf{x}_i^n + \Delta t\, \frac{\mathbf{v}_i^n + \mathbf{v}_i^{n*}}{2} \qquad \mathbf{v}_i^{n+1} = \frac{\mathbf{x}_i^{n+1} - \mathbf{x}_i^n}{\Delta t}$$

- Since the internal forces are handled by the GNN model, this one should ensure the conservation of momentum.
- The position correction, $\Delta \boldsymbol{x_i}$, predicted by the GNN for each particle must respect Newton's third law to guarantee conservation of momentum:

$$F_{j,i} = -F_{i,j} \rightarrow m_i \frac{\Delta \boldsymbol{x}_{j,i}}{\Delta t^2} = -m_j \frac{\Delta \boldsymbol{x}_{i,j}}{\Delta t^2} \rightarrow \Delta \boldsymbol{x}_{j,i} = -\Delta \boldsymbol{x}_{i,j}$$

Guaranteed Conservation of Momentum

- The only condition is $\Delta \boldsymbol{x}_{j,i} = -\Delta \boldsymbol{x}_{i,j}$
- The contribution of node i to the position correction of node j appears in the message sent from node j to node i in the last MP layer:

$$\Delta \boldsymbol{x}_j = \sum_{i \in S} \Delta \boldsymbol{x}_{i,j} = \sum_{i \in \mathcal{N}_j^-} \Delta \boldsymbol{x}_{i,j} = \sum_{i \in \mathcal{N}_j^-} m_{j \leftarrow i}$$

- In the CConv:

$$m_{j \leftarrow i}^{\text{CConv}} = a\left(||\boldsymbol{x}_i - \boldsymbol{x}_j||_2^2\right) \; \boldsymbol{f}_i \; G\left(\Lambda(\boldsymbol{x}_i - \boldsymbol{x}_j)\right)$$

- In the CConv $m_{j \leftarrow i}^{\text{CConv}} \neq -m_{j \leftarrow i}^{\text{CConv}}$, so momentum is not conserved.

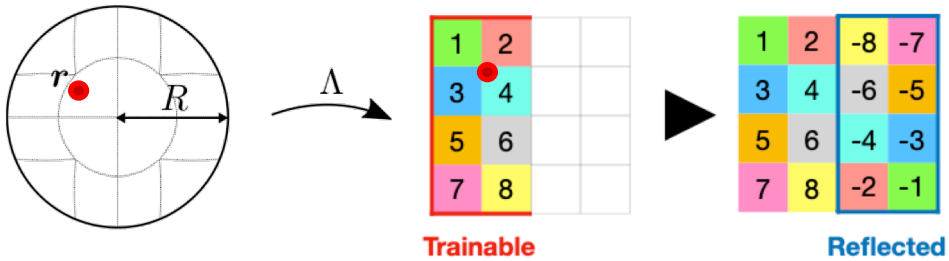
Guaranteed Conservation of Momentum

- Addressed by redefining the CConv message to be antisymmetric:

$$\Delta \boldsymbol{x}_j = \sum_{i \in \mathcal{N}_j^-} a\left(||\boldsymbol{x}_i - \boldsymbol{x}_j||_2^2\right) \; (\boldsymbol{f}_j + \boldsymbol{f}_i) \; G_s\left(\Lambda(\boldsymbol{x}_i - \boldsymbol{x}_j)\right)$$

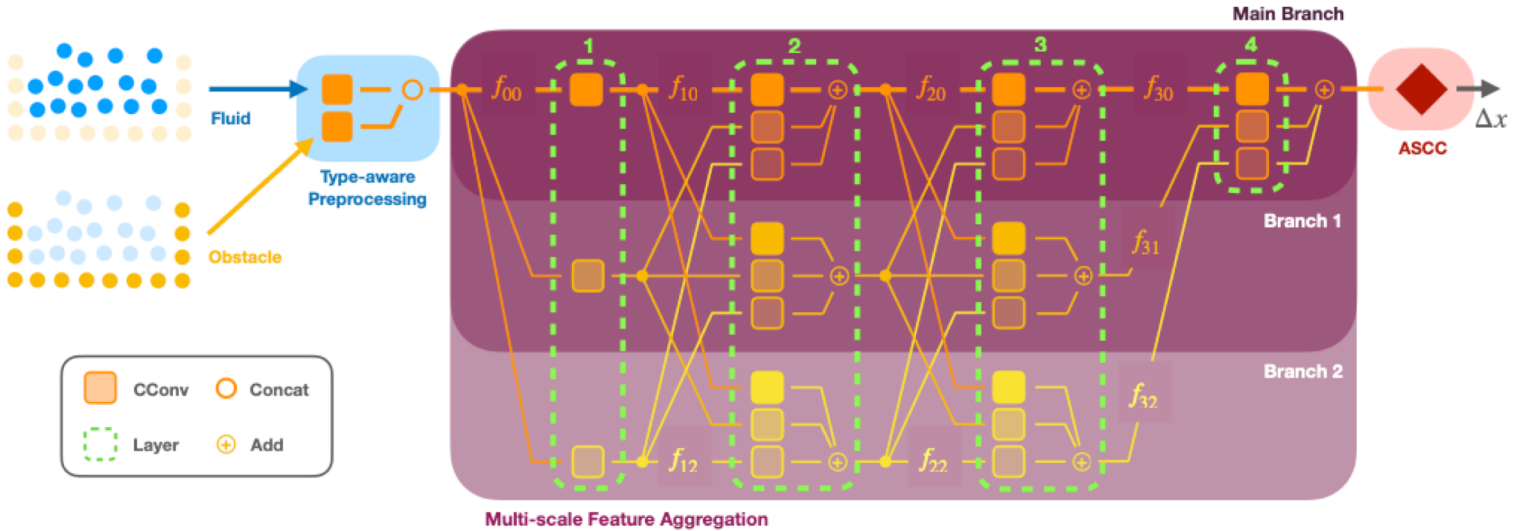
where $G_s : \mathbb{R}^3 \rightarrow \mathbb{R}^{C_{\text{in}} \times C_{\text{out}}}$ returns C_{in} x C_{out} anti-symmetric continuous filters.

- Only a half of the parameters of each filter are learnt.



Guaranteed Conservation of Momentum

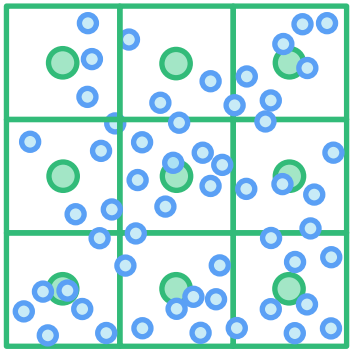
- This MP layer is known as **Anti-Symmetric CConv (ASCC)**.
- The ASCC is only needed for the last layer to guarantee momentum conservation.
- The previous four layers are **multi-scale CConvs**.



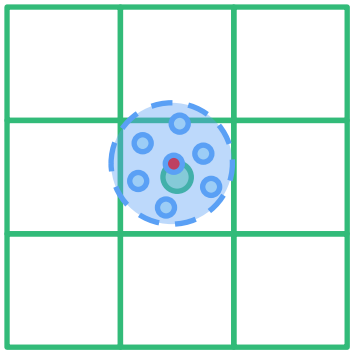
Guaranteed Conservation of Momentum

Multi-scale CConv: A CConv applied between L scales or levels of resolution to increase the receptive field of each particle.

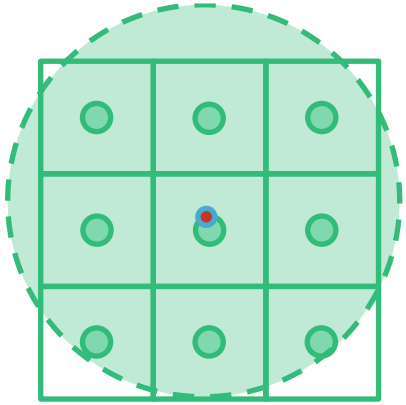
1. Node downsampling. Several possible strategies: graph contraction, furthest point sampling, **voxelisation**, etc.
2. CConv to each scale α from each scale β: $(\boldsymbol{f}_\alpha)_j \leftarrow \sum_\beta \mathrm{CConv}_{\beta\alpha}(V^\beta)$



Voxelisation



CConv_{00} +



CConv_{10}

Example of CConv to level 0 from levels 0 and 1 ($\alpha = 0$ and $\beta = 0,1$).

Note: each level needs its own R.

Guaranteed Conservation of Momentum

- The enforcement of momentum conservation resulted in average improvement of 50%
- The use of multi-scale CConvs further contributed to improve the accuracy, but in a much smaller percentage.
- The proposed model outperform baseline models in generalisation tasks.
- Thanks to its good generalisation it could be applied to large-scale 3D problems, which need high accuracy to remain stable for long rollouts.

G.T. PointNet CConv GNS ASCC