Deep Fluids: A Generative Network for Parameterized Fluid Simulations

Fall 2020 Research Group Presentation

Skylar Wurster

EUROGRAPHICS 2019 / P. Alliez and F. Pellacini
(Guest Editors)

Volume 38 (2019), Number 2

Deep Fluids: A Generative Network for Parameterized Fluid Simulations

Byungsoo Kim[1], Vinicius C. Azevedo[1], Nils Thuerey[2], Theodore Kim[3], Markus Gross[1] and Barbara Solenthaler[1]

[1]ETH Zürich [2]Technical University of Munich [3]Pixar Animation Studios

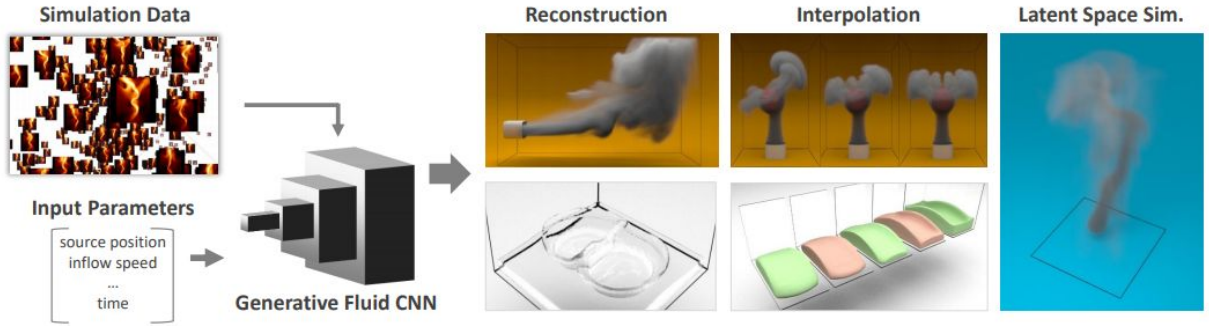


Figure 1: Our generative neural network synthesizes fluid velocities continuously in space and time, using a set of input simulations for training and a few parameters for generation. This enables fast reconstruction of velocities, continuous interpolation and latent space simulations.

Main contributions

1. A conditional autoencoder architecture that:
    a. Reaches up to 1400x compression on flow datasets

Main contributions

1. A conditional autoencoder architecture that:
    a. Reaches up to 1400x compression on flow datasets
    b. Generates fluid flow frames (through inference) up to 700x faster than direct numerical simulations (DNS)

Main contributions

1. A conditional autoencoder architecture that:
    a. Reaches up to 1400x compression on flow datasets
    b. Generates fluid flow frames (through inference) up to 700x faster than direct numerical simulations (DNS)
    c. Can approximate frames for unseen parameter settings (conditional generation)

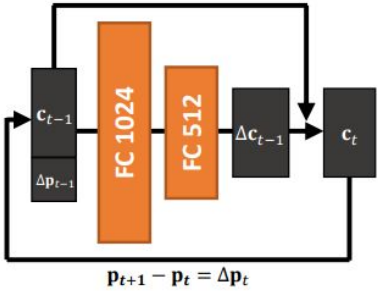
Main contributions

1. A conditional autoencoder architecture that:
    a. Reaches up to 1400x compression on flow datasets
    b. Generates fluid flow frames (through inference) up to 700x faster than direct numerical simulations (DNS)
    c. Can approximate frames for unseen parameter settings (conditional generation)
    d. Works on compressible and incompressible fluids in 2D and 3D

Main contributions

1. A conditional autoencoder architecture…

2. A small fully connected network that learns latent space advection to perform simulations completely in the latent space

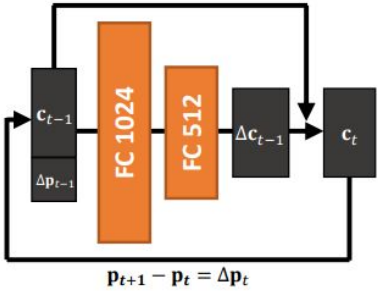


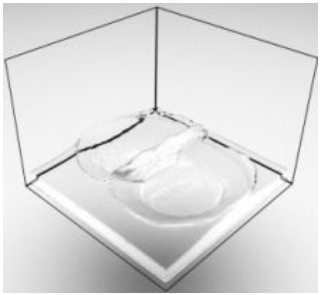
Main contributions

1. A conditional autoencoder architecture…

2. A small fully connected network that learns latent space advection to perform simulations completely in the latent space

3. A novel *stream function loss* that guarantees the output is physically accurate for incompressible fluids

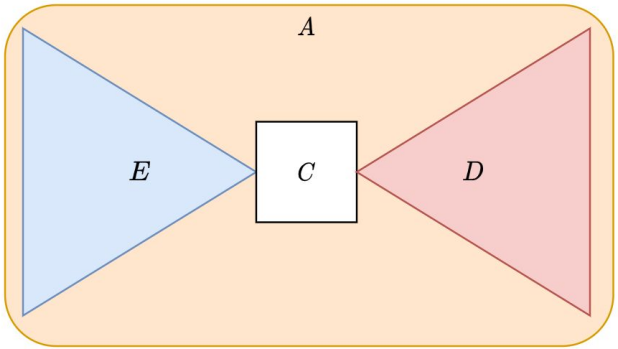




Background - Autoencoders

Autoencoders have two parts:

- Encoder (E): “compresses” input into a latent space that captures high level features about the data
- Decoder (D): “decompresses” the latent vector to recover the original ground truth



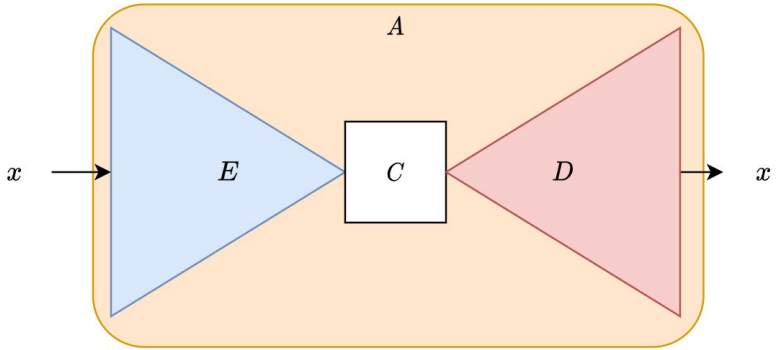
Background - Autoencoders

Autoencoders have two parts:

- Encoder (E): “compresses” input into a latent space that captures high level features about the data
- Decoder (D): “decompresses” the latent vector to recover the original ground truth

They are trained with this constraint (usually):

$$A(x) = D(E(x)) = x$$



Background - Autoencoders

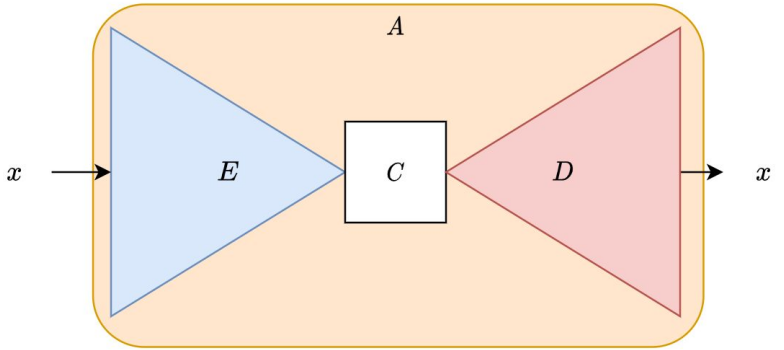
Autoencoders have two parts:

- Encoder (E): “compresses” input into a latent space that captures high level features about the data
- Decoder (D): “decompresses” the latent vector to recover the original ground truth

They are trained with this constraint (usually):

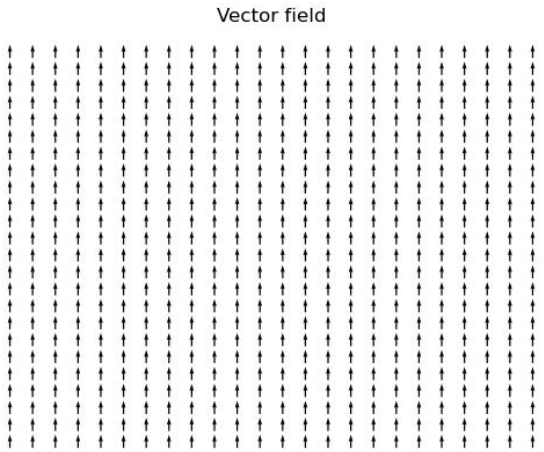
$$A(x) = D(E(x)) = x$$

Therefore, autoencoders are *unsupervised*.

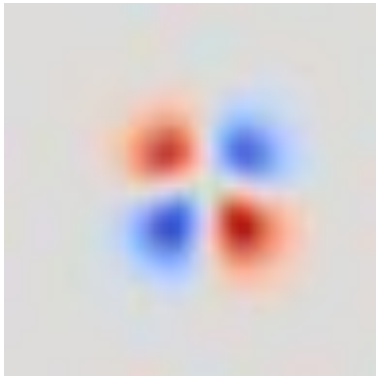


Background - Fluid representation

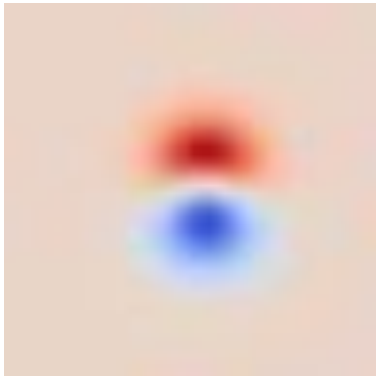
Vector field
2D: [2, x, y]



Matplotlib *quiver* function



Vorticity

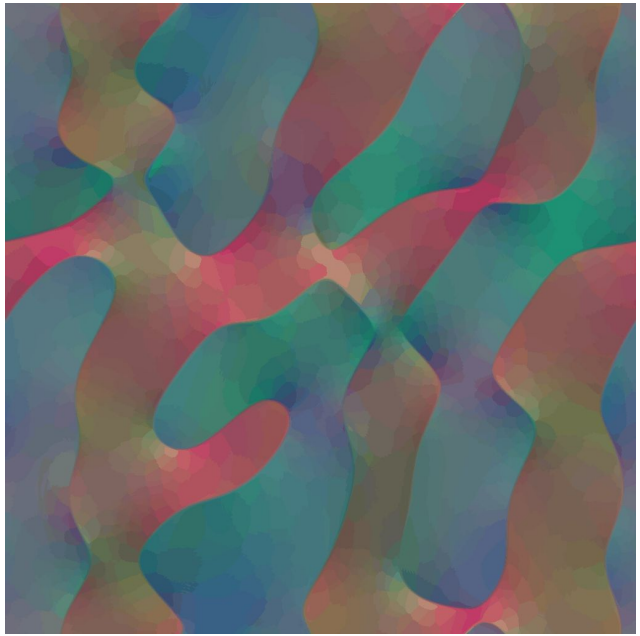


Speed

Background - Fluid representation

Vector field
2D: [2, x, y]
3D: [3, x, y, z]

John Hopkins University Turbulence data (single z-slice of 3D data)
Left: mixing
Right: isotropic
Velocity(u,v,w) directly mapped to RGB to visualize



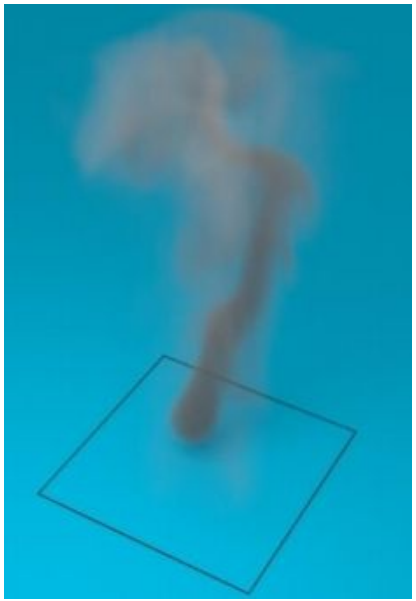
Background - Compressible and incompressible fluids

A property of fluid is *compressibility*.

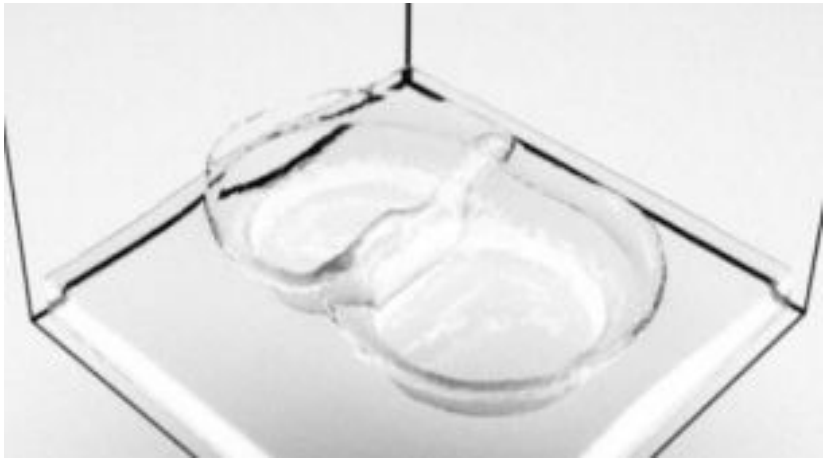
Background - Compressible and incompressible fluids

A property of fluid is *compressibility*.

Compressible



Incompressible

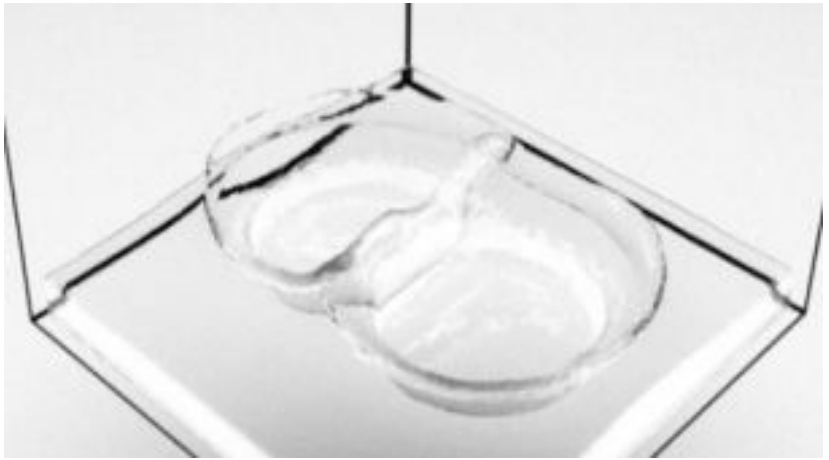


Background - Compressible and incompressible fluids

A property of fluid is *compressibility*.

Compressible

Incompressible

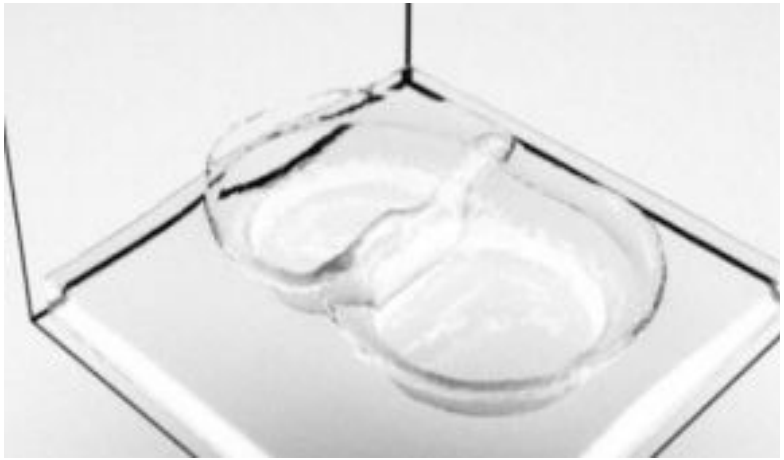


Background - Compressible and incompressible fluids

A property of fluid is *compressibility*.

Compressible

Incompressible



Background - Compressible and incompressible fluids

Incompressibility definition - ***divergence* free**

$$divV = \nabla \cdot V = 0$$

Background - Compressible and incompressible fluids

Incompressibility definition - ***divergence* free**

$$divV = \nabla \cdot V = 0$$

2D

$$\nabla \cdot V = \frac{\partial V_x}{\partial x} + \frac{\partial V_y}{\partial y}$$

3D

$$\nabla \cdot V = \frac{\partial V_x}{\partial x} + \frac{\partial V_y}{\partial y} + \frac{\partial V_z}{\partial z}$$

Background - Compressible and incompressible fluids

Incompressibility definition - ***divergence* free**

$$divV = \nabla \cdot V = 0$$

2D

$$\nabla \cdot V = \frac{\partial V_x}{\partial x} + \frac{\partial V_y}{\partial y}$$

3D

$$\nabla \cdot V = \frac{\partial V_x}{\partial x} + \frac{\partial V_y}{\partial y} + \frac{\partial V_z}{\partial z}$$

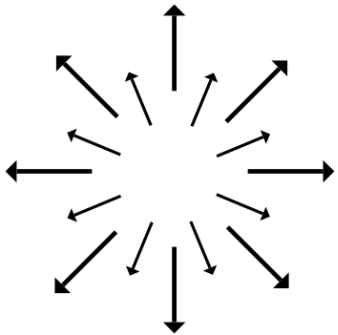
Divergence is a *scalar field* regardless of dimensionality

Background - Compressible and incompressible fluids

Incompressibility definition - ***divergence* free**

$$divV = \nabla \cdot V = 0$$

Positive divergence means the vector field is has a net negative flow out of the cell



$\partial/\partial x(\mathbf{V}_x) > 0$
$\partial/\partial y(\mathbf{V}_y) > 0$
$\nabla \bullet (\mathbf{V}) > 0$

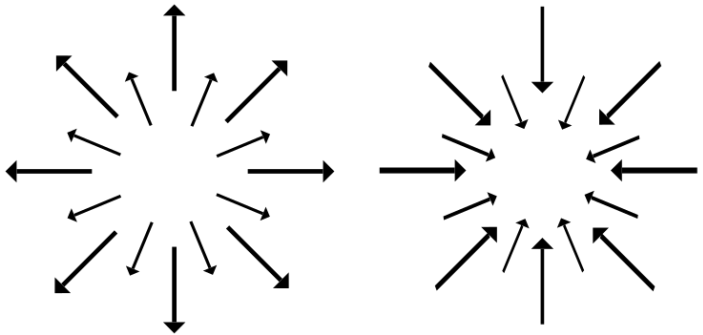
Background - Compressible and incompressible fluids

Incompressibility definition - ***divergence* free**

$$divV = \nabla \cdot V = 0$$

Positive divergence means the vector field is has a net negative flow out of the cell

Negative divergence means the VF has a net positive flow into the cell



$\partial/\partial x(\mathbf{V}_x) > 0$
$\partial/\partial y(\mathbf{V}_y) > 0$
$\nabla \bullet (\mathbf{V}) > 0$

$\partial/\partial x(\mathbf{V}_x) < 0$
$\partial/\partial y(\mathbf{V}_y) < 0$
$\nabla \bullet (\mathbf{V}) < 0$

Background - Compressible and incompressible fluids

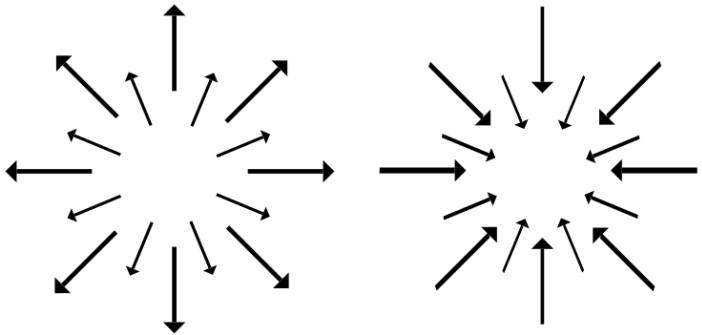
Incompressibility definition - ***divergence* free**

$$divV = \nabla \cdot V = 0$$

Positive divergence means the vector field is has a net negative flow out of the cell

Negative divergence means the VF has a net positive flow into the cell

Divergence free means no net flow through the cell. Makes sense, means no compression or lost matter for incompressible fluids

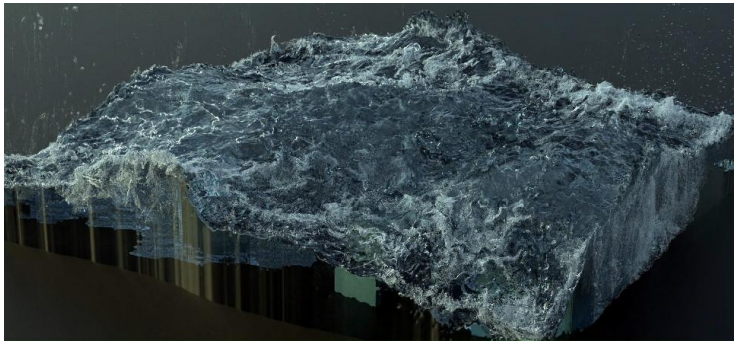


$\partial/\partial x\,(\mathbf{V}_x) > 0$
$\partial/\partial y\,(\mathbf{V}_y) > 0$
$\nabla \bullet (\mathbf{V}) > 0$

$\partial/\partial x\,(\mathbf{V}_x) < 0$
$\partial/\partial y\,(\mathbf{V}_y) < 0$
$\nabla \bullet (\mathbf{V}) < 0$

Deep Fluids Motivation

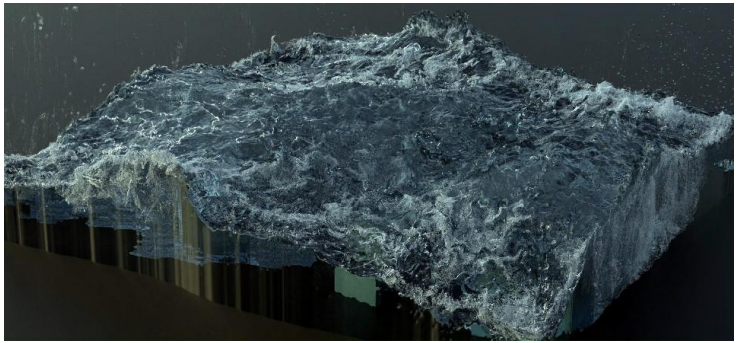
Fluid simulations can be expensive to simulate, so scientists are always looking for quicker ways to get data.



Deep Fluids Motivation

Fluid simulations can be expensive to simulate, so scientists are always looking for quicker ways to get data.

These come as something called “Reduced Order Methods” (ROMs)

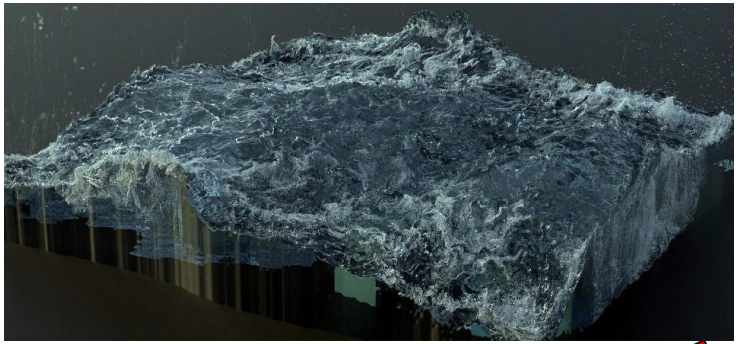


Deep Fluids Motivation

Fluid simulations can be expensive to simulate, so scientists are always looking for quicker ways to get data.

These come as something called “Reduced Order Methods” (ROMs)

These methods aim to speed up simulations by using simplified representations, which have all been linear this far.



$ gcc sim.c -o sim
$ ./sim.exe

$ python sim.py

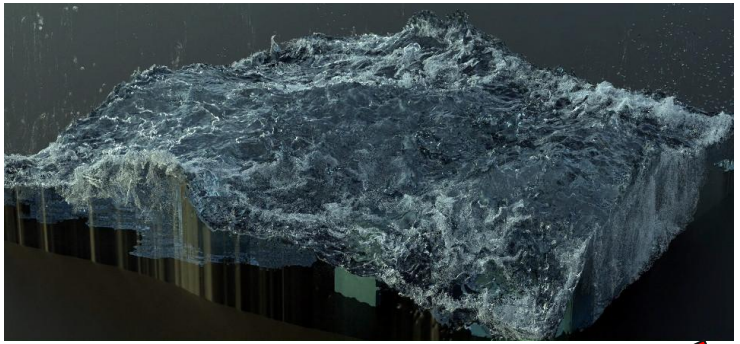
Deep Fluids Motivation

Fluid simulations can be expensive to simulate, so scientists are always looking for quicker ways to get data.

These come as something called “Reduced Order Methods” (ROMs)

These methods aim to speed up simulations by using simplified representations, which have all been linear this far.

A drawback of this is that fluid flow (Navier-Stokes equation) is highly non-linear.



$ gcc sim.c -o sim
$ ./sim.exe

$ python sim.py

Deep Fluids Motivation

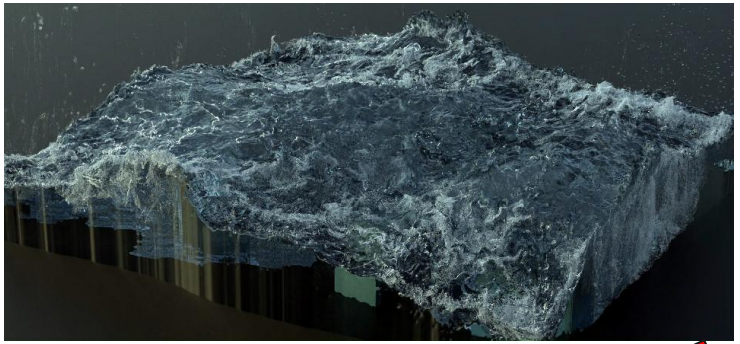
Fluid simulations can be expensive to simulate, so scientists are always looking for quicker ways to get data.

These come as something called “Reduced Order Methods” (ROMs)

These methods aim to speed up simulations by using simplified representations, which have all been linear this far.

A drawback of this is that fluid flow (Navier-Stokes equation) is highly non-linear.

The authors use a CNN-based approach as a novel non-linear ROM for fluid simulation.



$ gcc sim.c -o sim
$ ./sim.exe

$ python sim.py

Goal

The primary goal for Deep Fluids is to perform fluid flow simulation (approximation) efficiently (low computation/memory requirement). Both 2D and 3D incompressible and compressible fluids are desired.

To do this, they use a CNN-based approach as an ROM for fluid simulations.

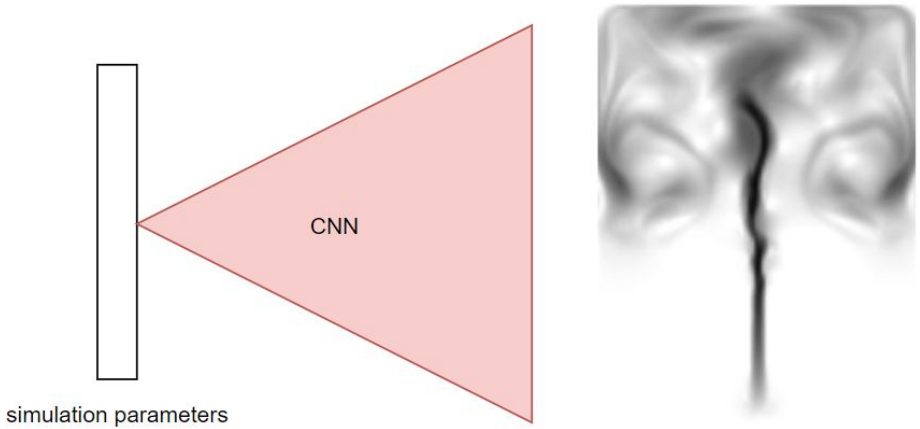
The authors suggest the non-linearities that neural networks have will allow the ROM to do well for the non-linear fluid data.

Primary goal is to train a CNN that approximates the original velocity field data set. The datasets can be parameterized (smoke cloud source, water drop starting position, etc…)

A generative fluid model

Given some parameters, we want to construct the vector field for the data.

Parameters might include:
Time, previous velocity field, viscosity, smoke plume origin, gravity, smoke speed, etc...

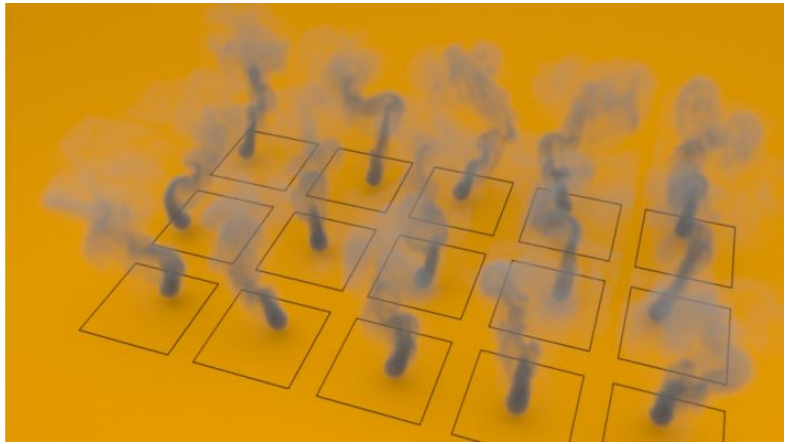


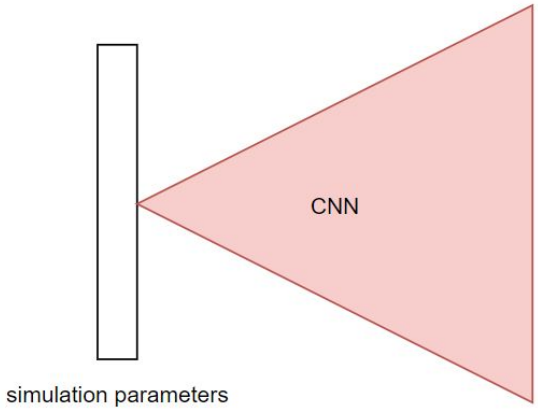
A generative fluid model

Given some parameters, we want to construct the vector field for the data.

Parameters might include:
Time, previous velocity field, viscosity, smoke plume origin, gravity, smoke speed, etc…

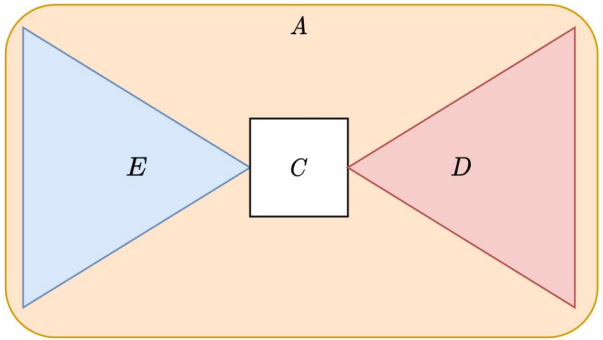
Problem: what if these change over time? For instance, if the smoke plume source moves around. Then the number of parameters grow linearly with the time, creating intractably large parameter space.





A generative fluid model

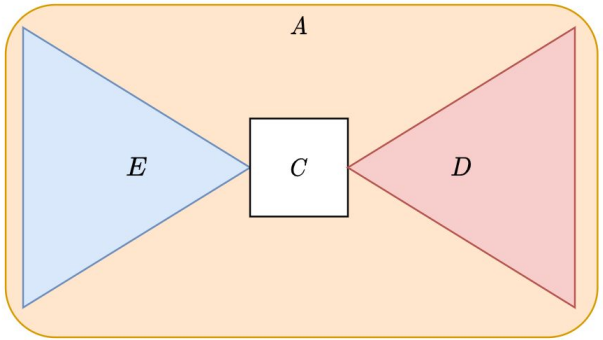
This is where the autoencoder is introduced - represent the parameters with a latent space representation.



A generative fluid model

This is where the autoencoder is introduced - represent the parameters with a latent space representation.

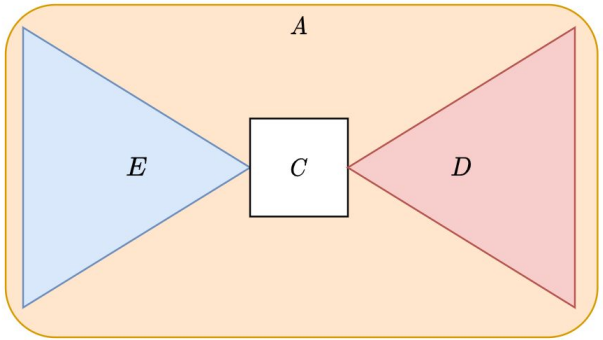
But, they still want manual control over some parameters, so they supervise some parameters in the latent space



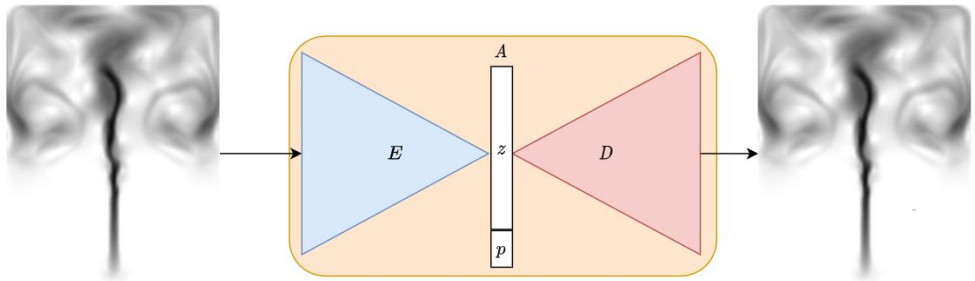
A generative fluid model

This is where the autoencoder is introduced - represent the parameters with a latent space representation.

But, they still want manual control over some parameters, so they supervise some parameters in the latent space

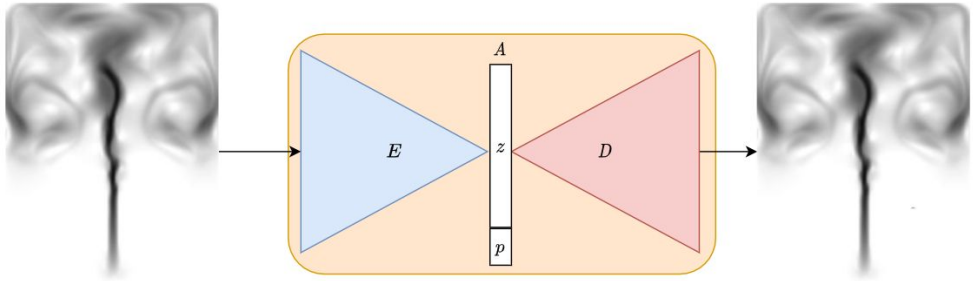


Solution:
C = [z, p]
z represents arbitrary features (unsupervised)
p is supervised and represents the smoke plume source (0.5)



A generative fluid model

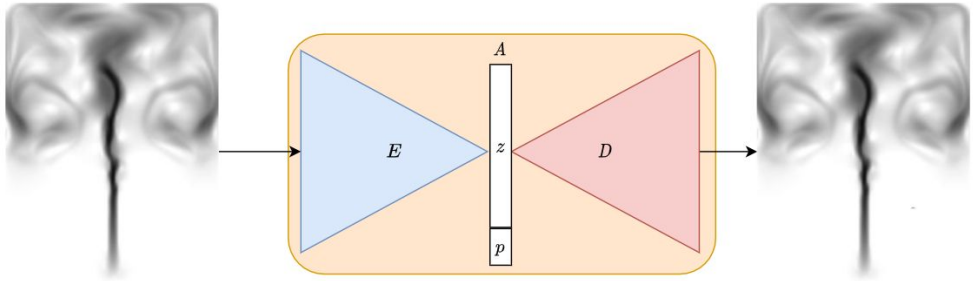
Now, each frame can be reduced to a latent space encoding parameters about the frame



This allows us to do our frame generation from the latent space, which is a compressed representation

A generative fluid model

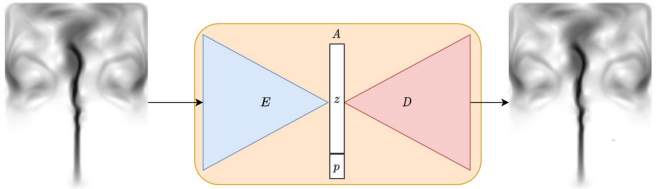
Now, each frame can be reduced to a latent space encoding parameters about the frame



This allows us to do our frame generation from the latent space, which is a compressed representation

But what about the next timesteps? How do we simulate the evolution of a fluid over time?

A generative fluid model

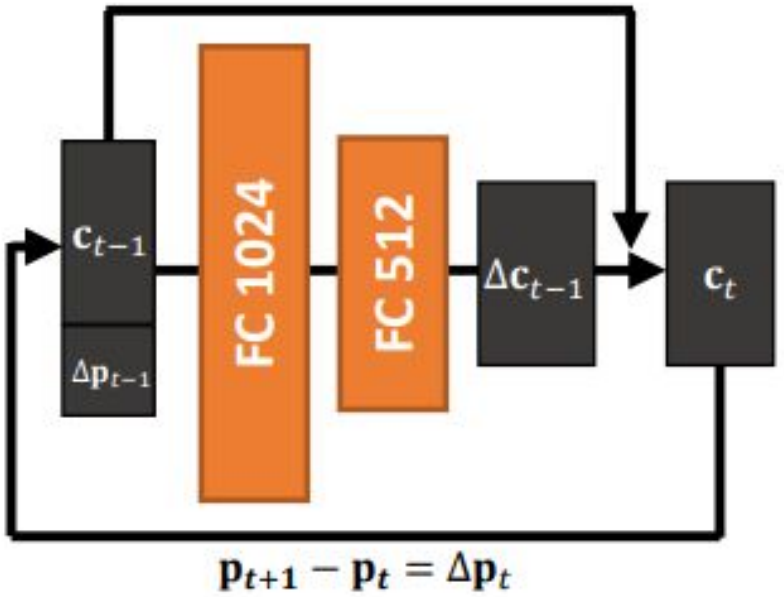


Time integration network

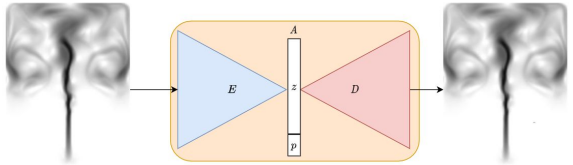
They create a second network that tries to replicate a simulation, but in the latent space.

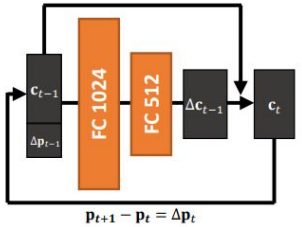
Learns the *change in the latent vector* given the current latent vector

Note: p is our *controlled* parameters that we must choose ourselves. That is why it is not a part of the output of the network. We append it ourselves.
This allows us to change some parameter over time to see different output.

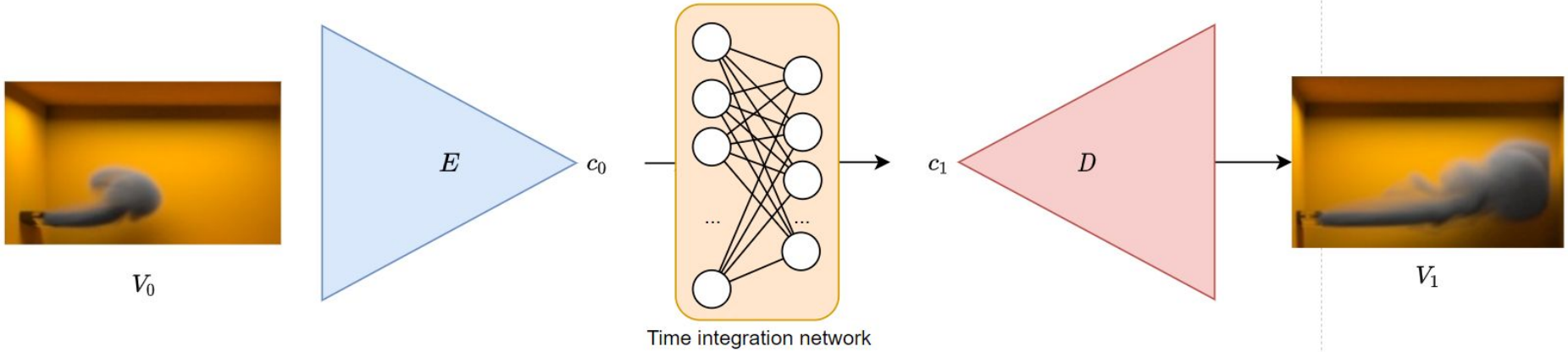


A generative fluid model

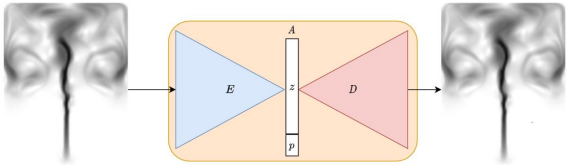


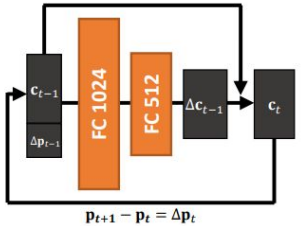


Big picture



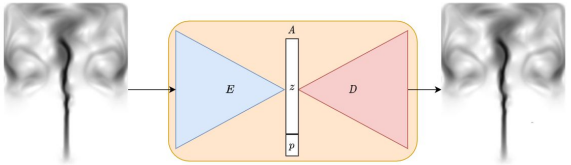
Training

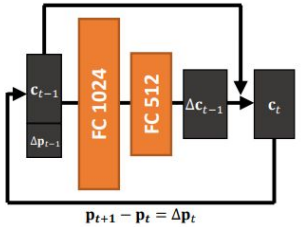




$$A(x) = D(E(x)) = x$$

Training



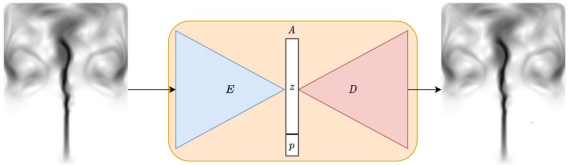


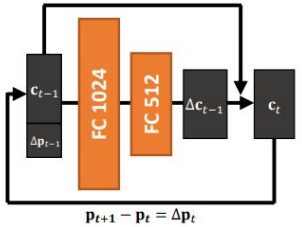
$$A(x) = D(E(x)) = x$$

Compressible fluid loss:

$$L(A(x)) = ||x - A(x)||_1$$

Training





$$A(x) = D(E(x)) = x$$

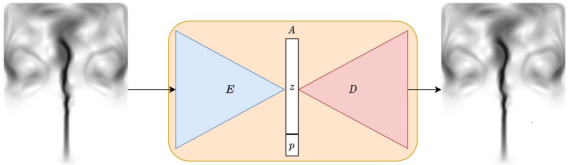
Compressible fluid loss:

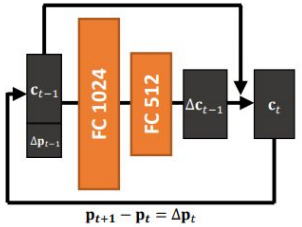
$$L(A(x)) = ||x - A(x)||_1$$

Incompressible fluid loss:

$$L(A(x)) = ||x - \nabla \times A(x)||_1$$

Training



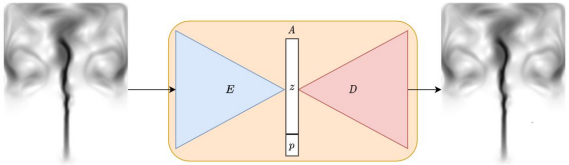


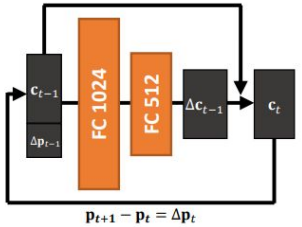
Incompressible fluid loss:

$$L(A(x)) = ||x - \nabla \times A(x)||_1$$

The curl of $A(x)$

Training





Incompressible fluid loss:

$$L(A(x)) = ||x - \nabla \times A(x)||_1$$

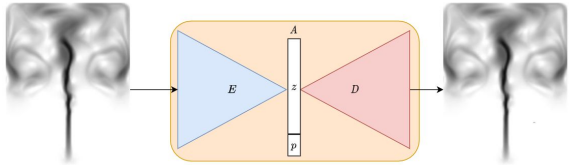
The curl of $A(x)$

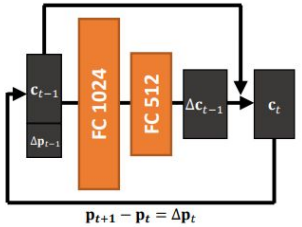
Curl in $\mathbb{R}^2$

$$curl(\vec{R}) = \nabla \times \vec{R} = \frac{\partial \vec{R}_y}{\partial x} - \frac{\partial \vec{R}_x}{\partial y}$$

Curl in R^2 is the rotation around the z-axis (scalar field)

Training



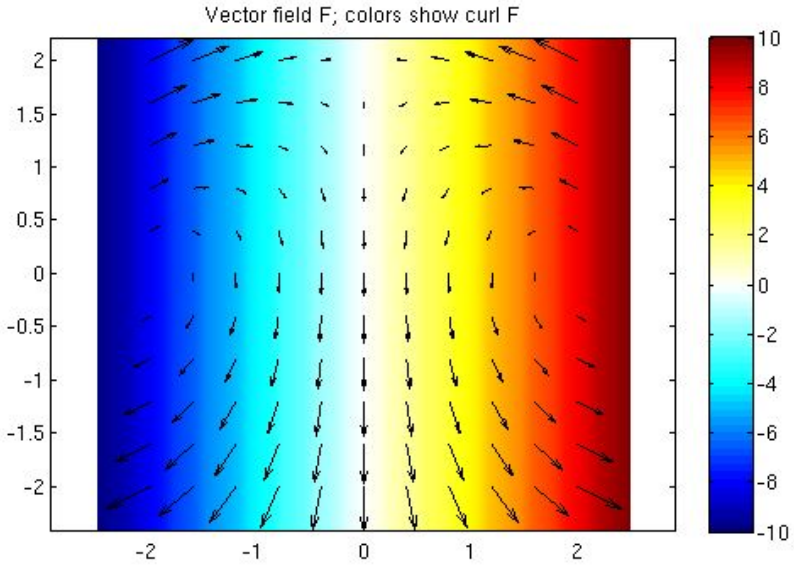


Curl in $\mathbb{R}^2$

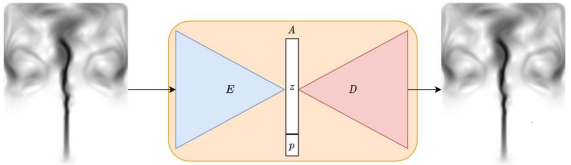
$$curl(\vec{R}) = \nabla \times \vec{R} = \frac{\partial \vec{R}_y}{\partial x} - \frac{\partial \vec{R}_x}{\partial y}$$

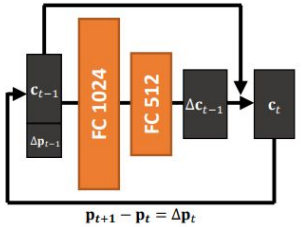
Curl in R^2 is the rotation around the z-axis

Positive curl is counter-clockwise rotation
Negative curl is clockwise rotation



Training





Incompressible fluid loss:

$$L(A(x)) = ||x - \nabla \times A(x)||_1$$

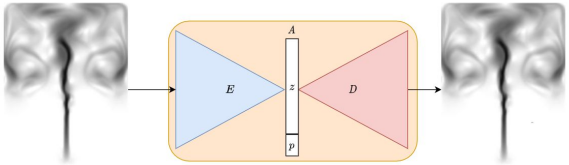
The curl of $A(x)$

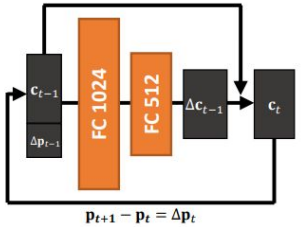
Curl in $\mathbb{R}^3$

$$curl(\vec{R}) = \nabla \times \vec{R} = [\frac{\partial \vec{R}_z}{\partial y} - \frac{\partial \vec{R}_y}{\partial z}, \frac{\partial \vec{R}_x}{\partial z} - \frac{\partial \vec{R}_z}{\partial x}, \frac{\partial \vec{R}_y}{\partial x} - \frac{\partial \vec{R}_x}{\partial y}]$$

Curl in R^3 is similar with each term identifying the rotation around some axis

Training





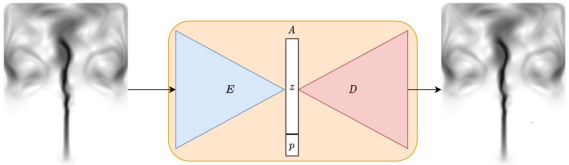
Incompressible fluid loss:

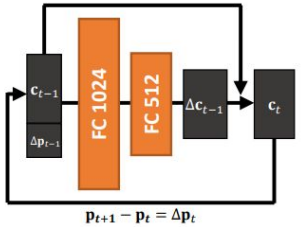
$$L(A(x)) = ||x - \nabla \times A(x)||_1$$

The curl of $A(x)$

So why use the curl of the output of the network?

Training



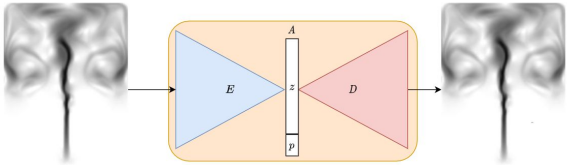


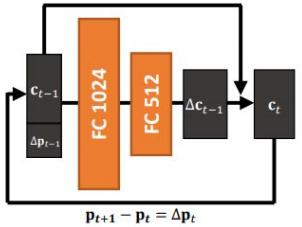
Identity:

The divergence of the curl of some vector field is always 0

$$\nabla \cdot (\nabla \times F) = 0$$

Training



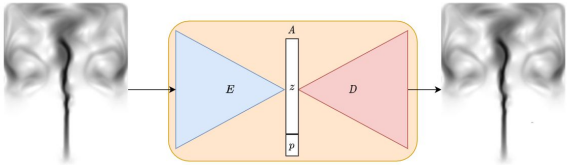


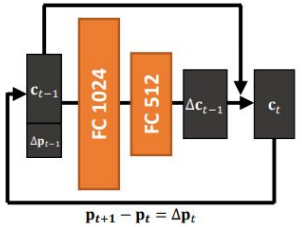
Identity:

The divergence of the curl of some vector field is always 0

$$\nabla \cdot (\nabla \times F) = 0$$

Training





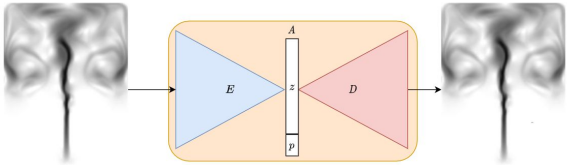
Identity:

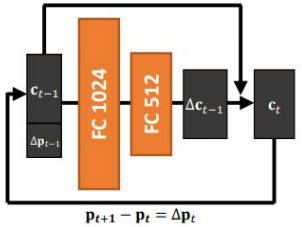
The divergence of the curl of some vector field is always 0

$$\nabla \cdot (\nabla \times F) = 0$$

$$\nabla \times \vec{V} = [\frac{\partial \vec{V}_z}{\partial y} - \frac{\partial \vec{V}_y}{\partial z}, \frac{\partial \vec{V}_x}{\partial z} - \frac{\partial \vec{V}_z}{\partial x}, \frac{\partial \vec{V}_y}{\partial x} - \frac{\partial \vec{V}_x}{\partial y}]$$

Training





Identity:

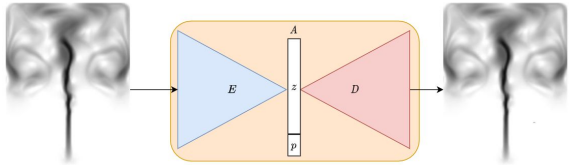
The divergence of the curl of some vector field is always 0

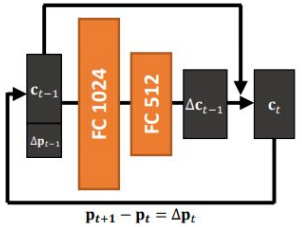
$$\nabla \cdot (\nabla \times F) = 0$$

$$\nabla \times \vec{V} = [\frac{\partial \vec{V}_z}{\partial y} - \frac{\partial \vec{V}_y}{\partial z}, \frac{\partial \vec{V}_x}{\partial z} - \frac{\partial \vec{V}_z}{\partial x}, \frac{\partial \vec{V}_y}{\partial x} - \frac{\partial \vec{V}_x}{\partial y}]$$

$$\nabla \cdot (\nabla \times V) = \nabla \cdot ([\frac{\partial \vec{V}_z}{\partial y} - \frac{\partial \vec{V}_y}{\partial z}, \frac{\partial \vec{V}_x}{\partial z} - \frac{\partial \vec{V}_z}{\partial x}, \frac{\partial \vec{V}_y}{\partial x} - \frac{\partial \vec{V}_x}{\partial y}])$$

Training





Identity:

The divergence of the curl of some vector field is always 0

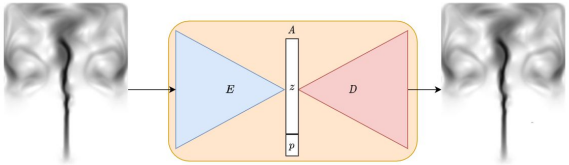
$$\nabla \cdot (\nabla \times F) = 0$$

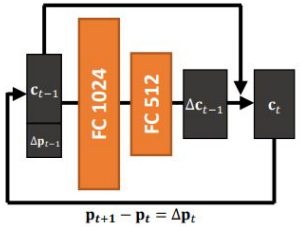
$$\nabla \times \vec{V} = [\frac{\partial \vec{V}_z}{\partial y} - \frac{\partial \vec{V}_y}{\partial z}, \frac{\partial \vec{V}_x}{\partial z} - \frac{\partial \vec{V}_z}{\partial x}, \frac{\partial \vec{V}_y}{\partial x} - \frac{\partial \vec{V}_x}{\partial y}]$$

$$\nabla \cdot (\nabla \times V) = \nabla \cdot ([\frac{\partial \vec{V}_z}{\partial y} - \frac{\partial \vec{V}_y}{\partial z}, \frac{\partial \vec{V}_x}{\partial z} - \frac{\partial \vec{V}_z}{\partial x}, \frac{\partial \vec{V}_y}{\partial x} - \frac{\partial \vec{V}_x}{\partial y}])$$

$$\nabla \cdot (\nabla \times V) = \frac{\partial(\frac{\partial \vec{V}_z}{\partial y} - \frac{\partial \vec{V}_y}{\partial z})}{\partial x} + \frac{\partial(\frac{\partial \vec{V}_x}{\partial z} - \frac{\partial \vec{V}_z}{\partial x})}{\partial y} + \frac{\partial(\frac{\partial \vec{V}_y}{\partial x} - \frac{\partial \vec{V}_x}{\partial y})}{\partial z}$$

Training





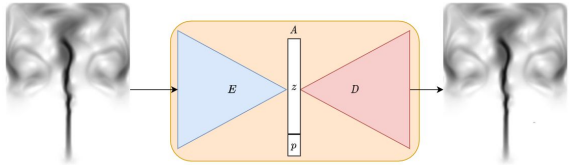
If we teach the network to generate this *stream function*, and then take the curl, the output is guaranteed to be divergence free

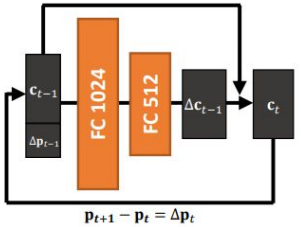
Thus, they call this a *stream function loss*.

$$\nabla \cdot (\nabla \times F) = 0$$

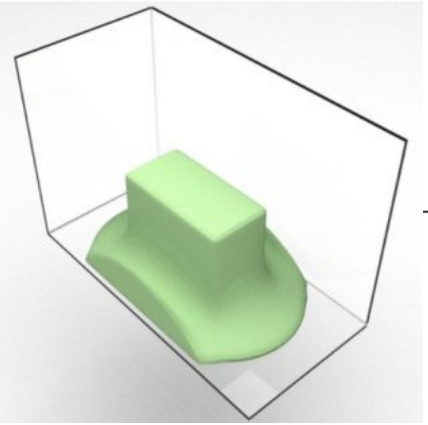
Stream function

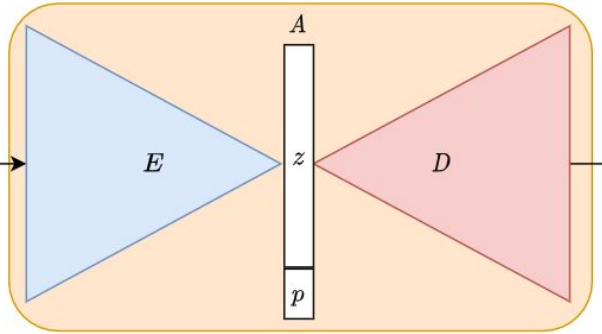
Training



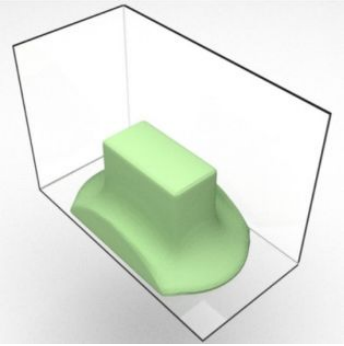


$$L(A(x)) = ||x - \nabla \times A(x)||_1$$

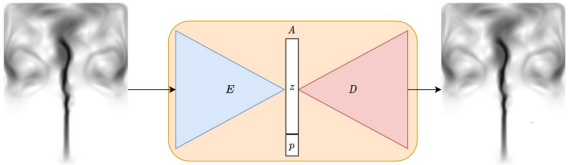


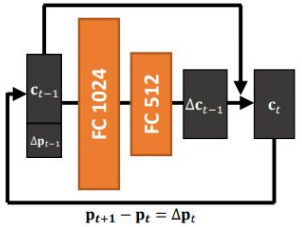


$$F \rightarrow \nabla \times F$$



Training



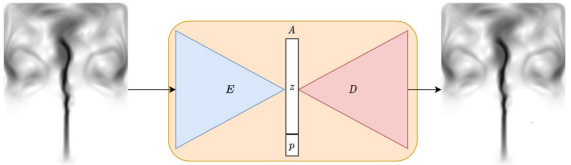


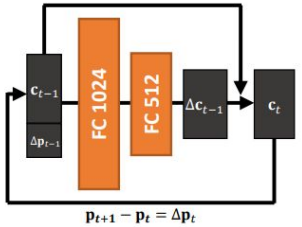
Gradient loss:

Velocity gradient encodes vorticity, shearing, and divergence.

$$L_{\nabla} = ||\nabla x - \nabla A(x)||_1$$

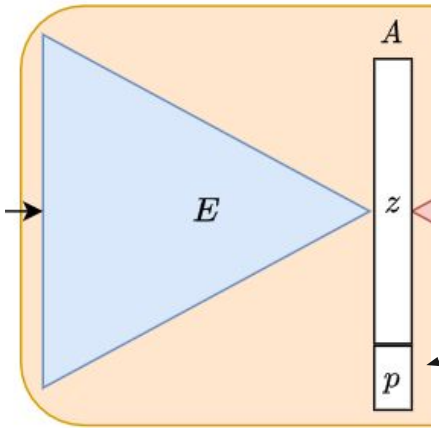
Training





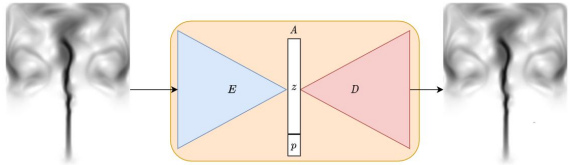
Controlled parameter loss:

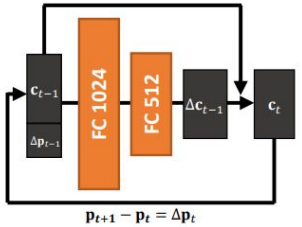
Force some parameters in the AE latent space to be controlled



$$L_p = ||E(x)_p - p||_2^2$$

Training





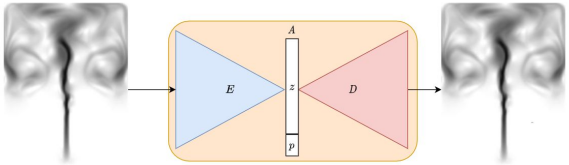
Full autoencoder loss

$$L = L_{L1} + L_{\nabla} + L_p$$

$$L_{L1} = ||x - A(x)||_1 \qquad L_{\nabla} = ||\nabla x - \nabla A(x)||_1 \qquad L_p = ||E(x)_p - p||_2^2$$

When training on incompressible fluids, define A(x) as the curl of the autoencoder output.

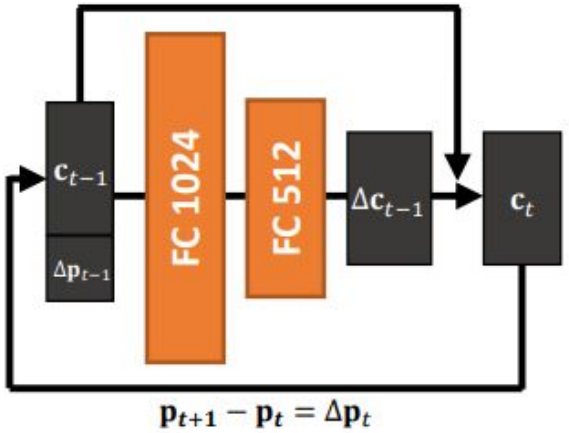
Training



Temporal inference network

w=30 is used

$$L_T(\mathbf{x}_t, \ldots, \mathbf{x}_{t+w-1}) = \frac{1}{w} \sum_{i=t}^{t+w-1} ||\Delta \mathbf{z}_i - T_i||_2^2,$$



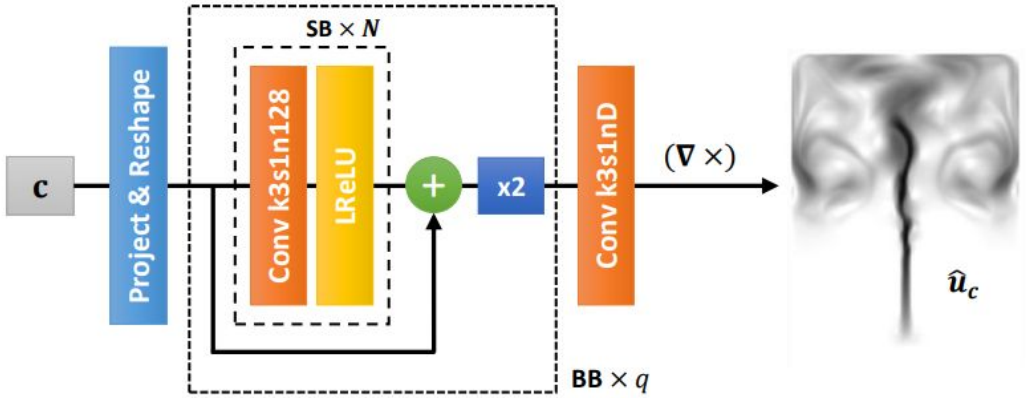
Architecture - Encoder/Decoder

Decoder architecture
(encoder is symmetric)

SB = Small Blocks
BB = Big Blocks

c = 16
N = 4 or 5
$q = \log_2(d_{max}) - 3$

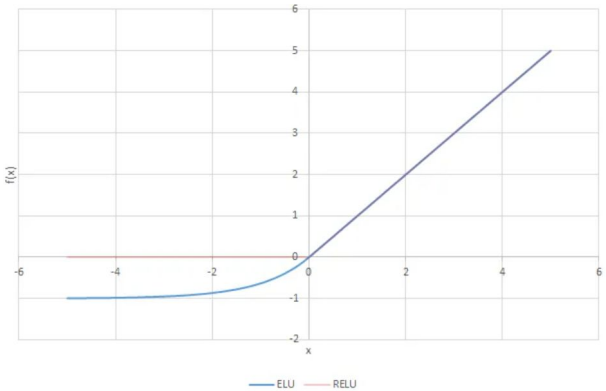
Ex. $d_{max} = 128 \rightarrow q = 7-3 = 4$

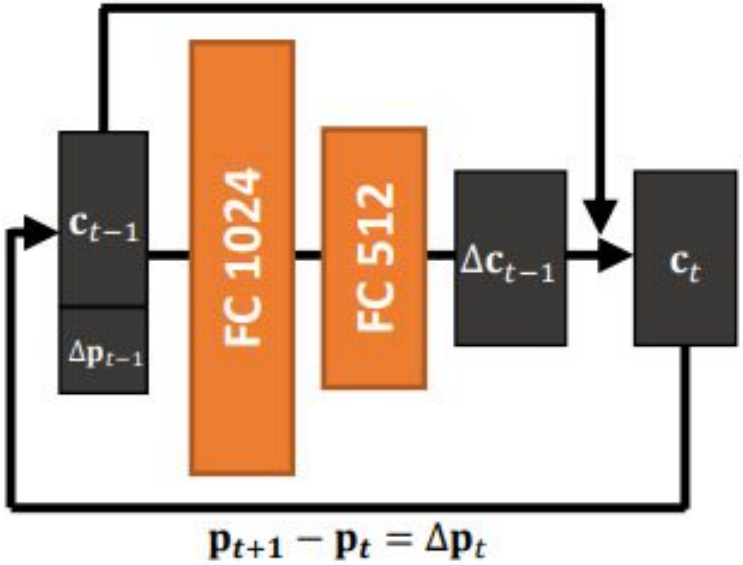


Architecture - Temporal inference network

3 fully connected

ELU activation function
Batchnorm and dropout layers (p=0.1)
to avoid overfitting



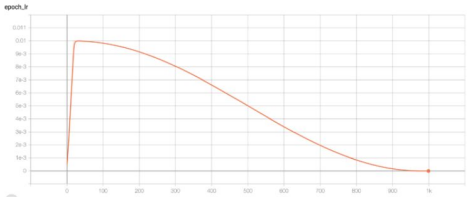


Results

Specifics:
300,000 iterations per AE network, 30,000 for time network
Learning rate scheduled with cosine annealing decay
Batch size: 1 for 3D, 8 for 2D

Sims performed in Mantaflow

Data is all normalized to [-1, 1] via maximum absolute value



Results - 2D smoke

Parameters: source width(5 samples) and source x position(21 samples)
Simulation length: 200 frames
Resolution: 96x128
Training data: 21000 unique frames

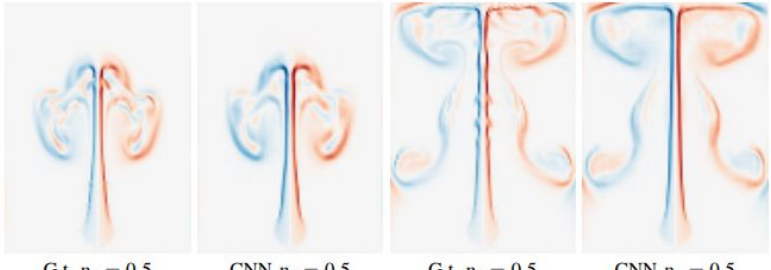
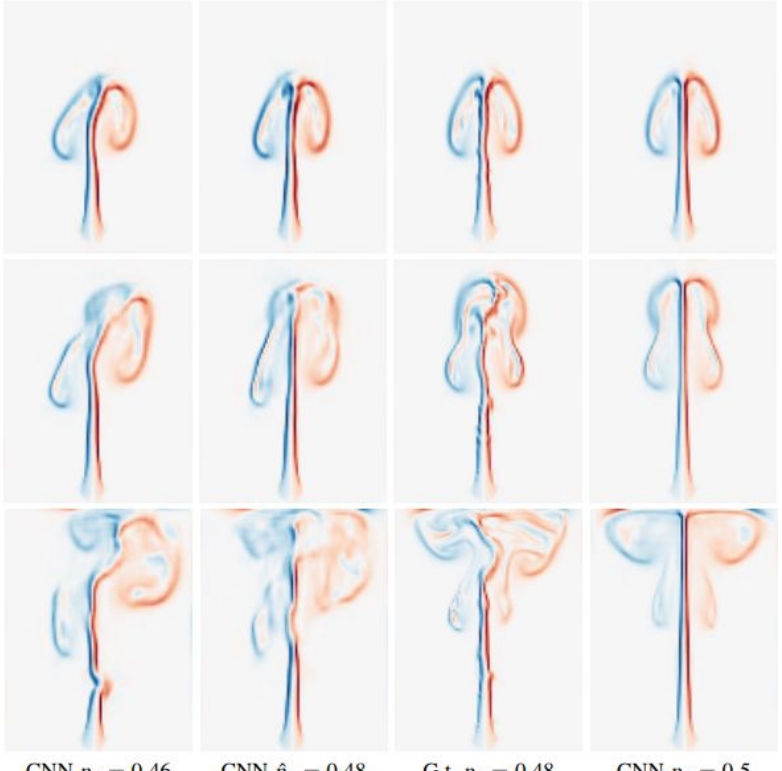


Figure 6: Vorticity plot of a 2-D smoke simulation with direct correspondences to the training data set for two different times. The RdBu colormap is used to show both the magnitude and the rotation direction of the vorticity (red: clockwise). Our CNN is able to closely approximate ground truth samples (G.t.).



Not in training data

Results - 3D smoke sphere obstacle

Parameters: sphere position(10 samples)
Resolution: 64x96x64
Simulation length: 660 frames

Not in training data

Results - 3D smoke inflow and buoyancy

Parameters: inflow velocities (5), buoyancy values (3)
Simulation length: 250 frames
Resolution: 112x64x32

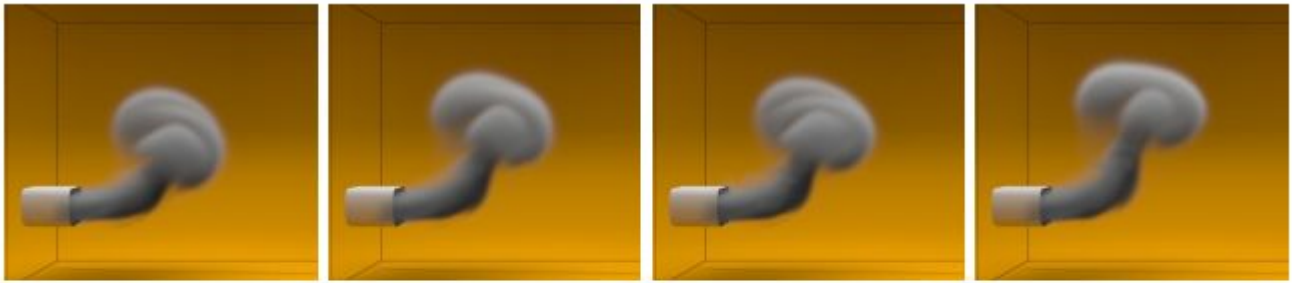


Figure 9: Reconstructions of the rising plume scene (left and right), reconstruction for an interpolated buoyancy value ($\hat{b} = 8 \times 10^{-4}$) (second image) and the corresponding ground truth (third image).

Results - 3D smoke rotating

Parameters: inflow velocities (5), buoyancy values (3)
Simulation length: 500 frames
Resolution: 48x72x48
Smoke rotates periodically every 100 frames

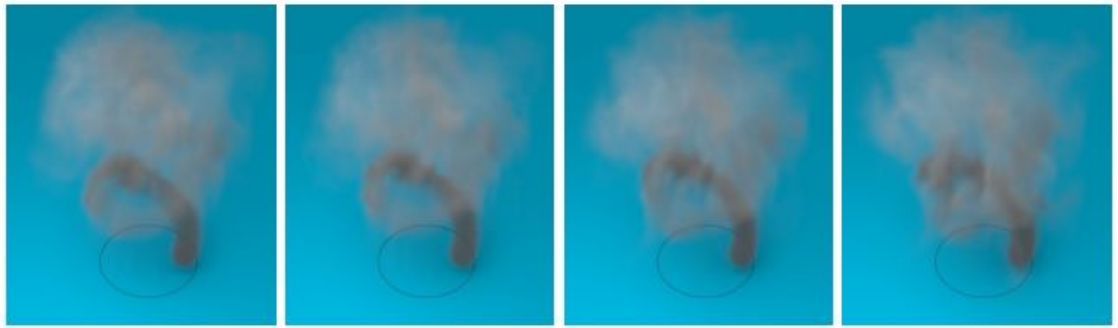


Figure 10: Time extrapolation results using our latent space integration network. The left image shows the last frame of the ground truth simulation. The subsequent images show results with time extrapolation of +20%, +60% and +100% of the original frames.

Results - 3D smoke moving

Simulation length: 400 frames
Resolution: 48x72x48
Number of sims: 200

Each sim moves the smoke source around randomly using Perlin noise. Evaluated by using time network to generate plausible flow.

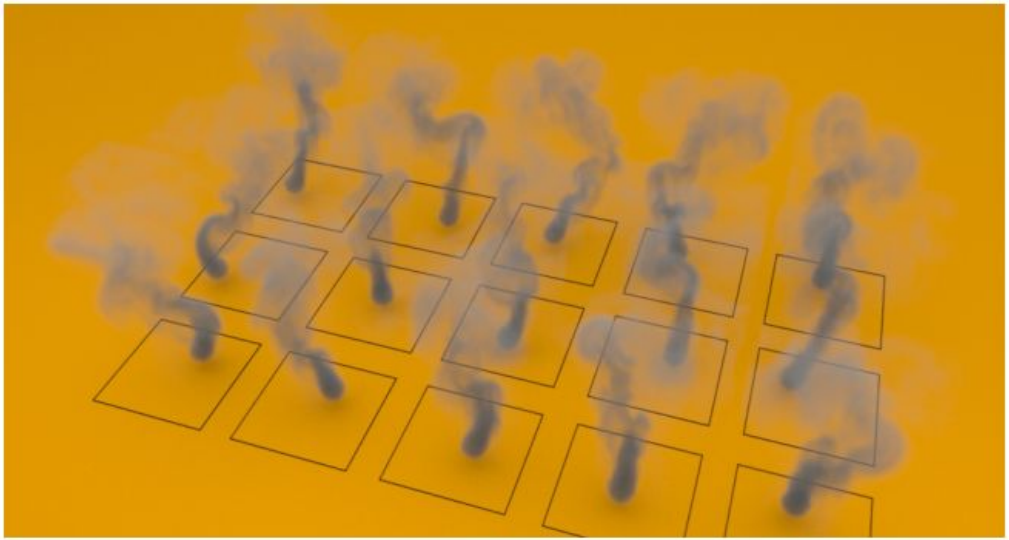


Figure 11: Example simulations of the moving smoke scene used for training the extended parameterization networks.

Results - 3D liquid spheres

Parameters: initial distance of the spheres (5), drop angles (10)
Simulation length: 150 frames
Resolution: 96x72x48

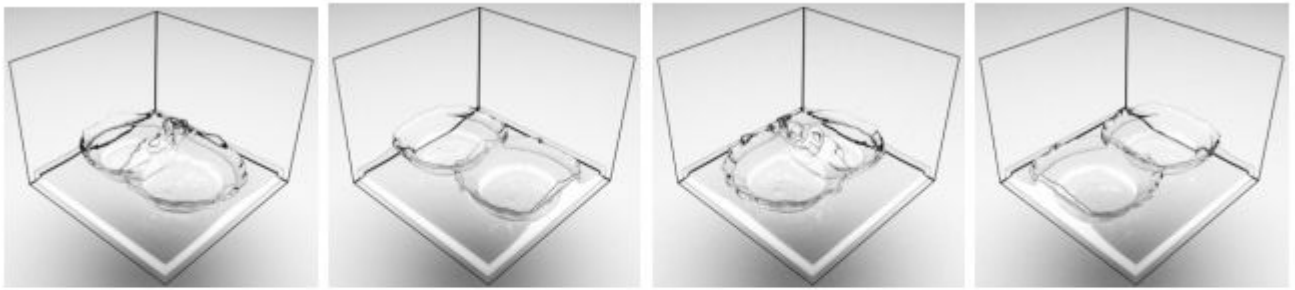


Figure 13: Training samples for the liquid spheres scene. In total we used 50 simulation examples with varying distances and angles.

Results - 3D liquid spheres

Parameters: initial distance of the spheres (5), drop angles (10)
Simulation length: 150 frames
Resolution: 96x72x48

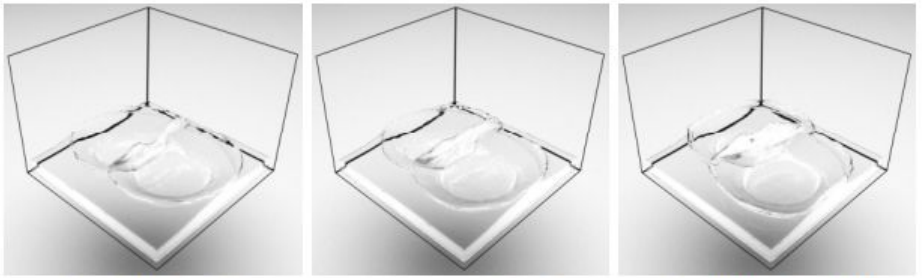


Figure 14: CNN-generated results with parameter interpolation for the liquid spheres example. While the far left and right simulations employ parameter settings that were part of the training data, the middle example represents a new in-between parameter point which is successfully reconstructed by our method.

Results - 3D liquid viscous dam break

Parameters: viscosity strength (4)
Simulation length: 150 frames
Resolution: 96x72x48

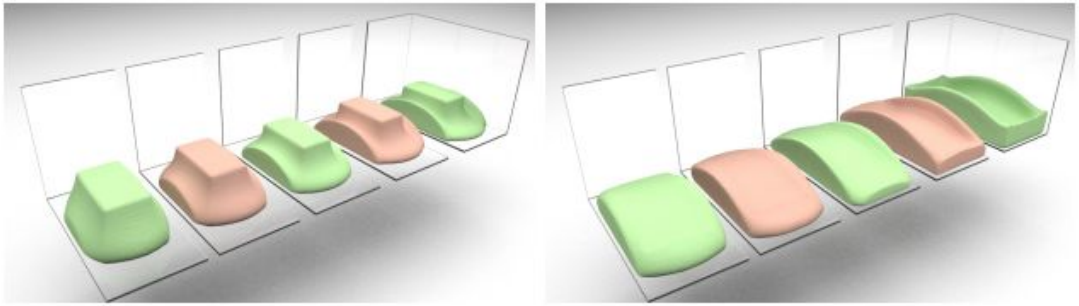
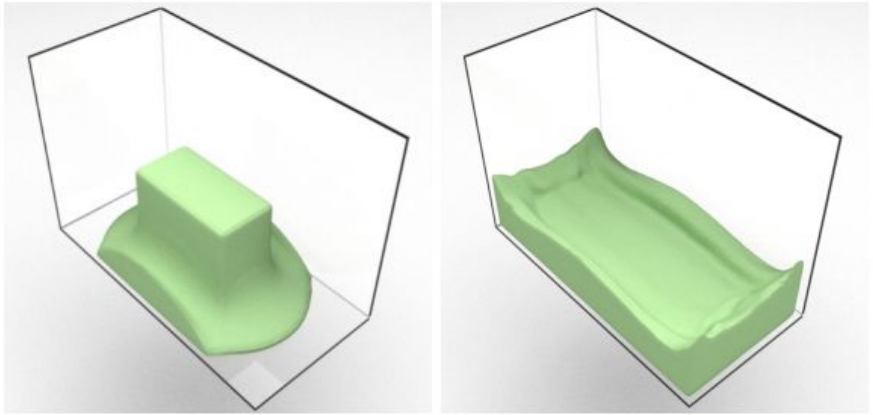


Figure 15: Snapshots of a CNN reconstructed dam break with different viscosity strengths for two different frames. Green liquids denote correspondences with ground truth ($\mu = 2 \times [10^{-4}, 10^{-3}, 10^{-2}]$, back to front) while pink ones are interpolated ($\hat{\mu} = 2 \times [5^{-3}, 5^{-2}]$, back to front).

Results - 3D liquid viscous dam break

Parameters: viscosity strength (4)
Simulation length: 150 frames
Resolution: 96x72x48



Frame 23, $\hat{\mu} = 1.04 \times 10^{-3}$ Frame 64, $\hat{\mu} = 7.14 \times 10^{-3}$

Figure 16: Reconstruction result using a time varying viscosity strength. In the first few frames the liquid quickly breaks into the container. As the simulation advances, the viscosity increases and the liquid sticks to a deformed configuration.

Results - training

- Batch normalization didn't help
- Other rescaling techniques might do better
  - Standardization, histogram equalization
- Training was stable for all trials
- Training took longer with liquid to have high quality surfaces than it took with the gases

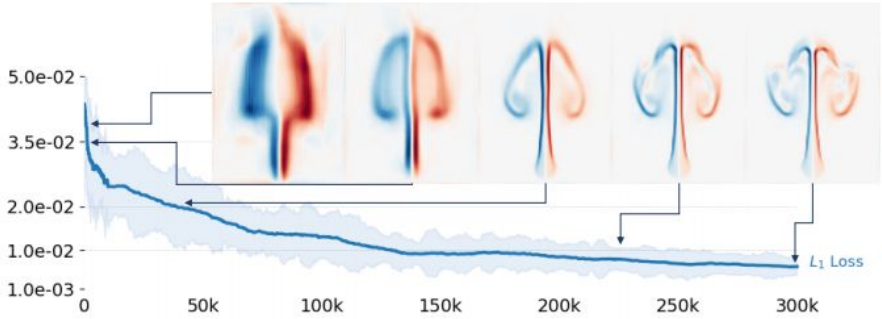


Figure 17: Convergence plot of the L_1 loss for the 2-D smoke sequence from Figure 7.

Results - compression and ROM

Efficiency:

- Up to 700x faster than simulating via CPU
- Can efficiently create many timesteps at once (batching) while a sim has to do one step at a time

Results - compression and ROM

Efficiency:

- Up to 700x faster than simulating via CPU
- Can efficiently create many timesteps at once (batching) while a sim has to do one step at a time

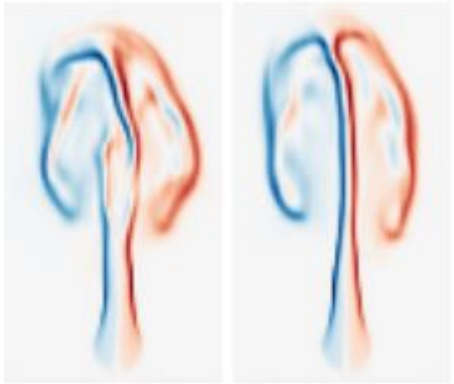
Compression:

- Compression rates of up to 1300x, based on the network taking a static 30MB.
  - Previous ROMs only achieved 14x.
- Outperforms FZIP (compression algorithm), which only achieves 4x with a mean absolute error comparable to Deep Fluids

Results - quality of reconstruction

- Highly dependent on training data and proximity to a nearby parameter setting.

Less training data



More training data

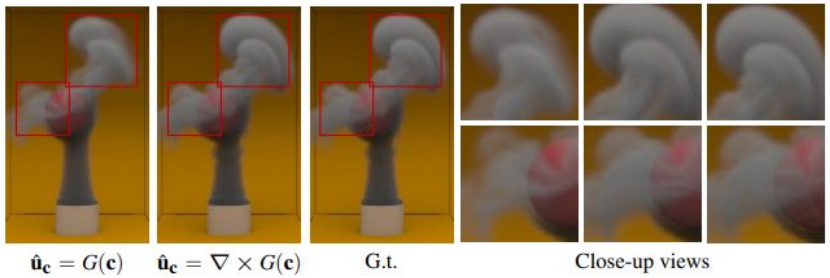


Figure 18: Comparisons of the results from networks trained on our compressible loss, incompressible loss and the ground truth, respectively. On the right sequence we show the highlighted images from the simulations on the left. We notice that the smoke patterns from the incompressible loss are closer to ground truth simulations.

- Training with incompressible loss gives better results.

Results - quality of reconstruction

- Attempts to not penetrate boundaries

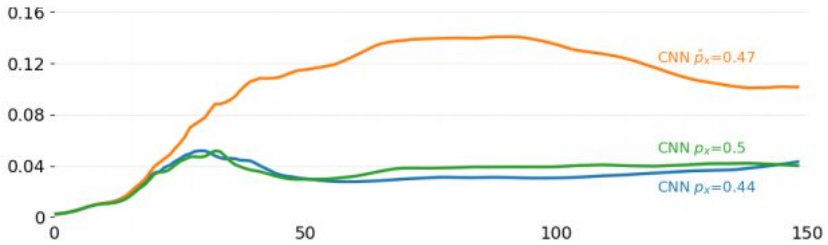


Figure 20: Mean absolute error plot of velocity penetration for the smoke obstacle example. Though errors in interpolated samples are a bit higher than those of reconstructed samples, they do not exceed 1% of the maximum absolute value of the data set.

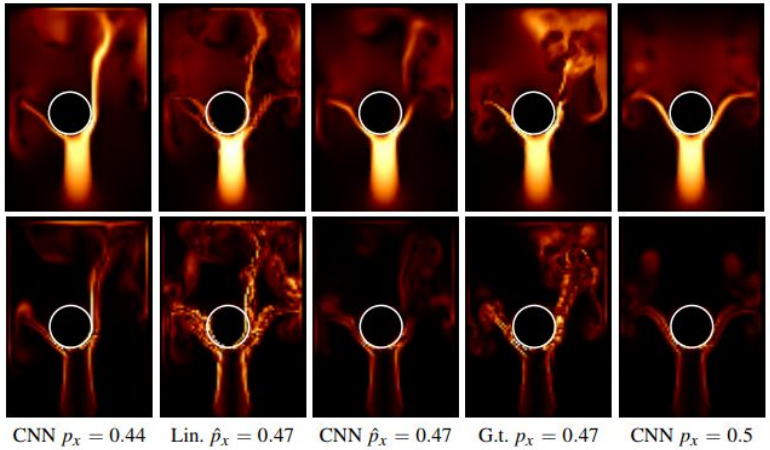


Figure 19: Slice views of the last row of Fig. 8. The color code represents the velocity (top) and vorticity (bottom) magnitudes. The second column shows a linear interpolation of the input. Despite the absence of any constraints on boundary conditions, our method (third column) preserves the shape of the original sphere obstacle, and yields significantly better results than the linear interpolation.

Results - summary

Scene	Grid Resolution	# Frames	Simulation Time (s)	Eval. Time (ms) [Batch]	**Speed Up** (×)	Data Set Size (MB)	Network Size (MB)	**Compression Ratio**	Training Time (h)
Smoke Plume	96×128	21,000	0.033	0.052 [100]	635	2064	12	172	5
Smoke Obstacle	$64 \times 96 \times 64$	6,600	0.491	0.999 [5]	513	31143	30	1038	74
Smoke Inflow	$112 \times 64 \times 32$	3,750	0.128	0.958 [5]	128	10322	29	356	40
Liquid Drops	$96 \times 48 \times 96$	7,500	0.172	1.372 [3]	125	39813	30	**1327**	134
Viscous Dam	$96 \times 72 \times 48$	600	0.984	1.374 [3]	**716**	2389	29	82	100
Rotating Smoke	$48 \times 72 \times 48$	500	0.08	0.52 [10]	308	995	38	26	49
Moving Smoke	$48 \times 72 \times 48$	80,000	0.08	0.52 [10]	308	159252	38	4191*	49

Table 1: Statistics for training data sets and our CNN. Note that simulation excludes advection and is done on the CPU, while network evaluation is executed on the GPU with batch sizes noted in brackets. In case of liquids, the conjugate gradient residual threshold is set to $1e^{-3}$, while for smoke it is $1e^{-4}$. For the Rotating and Moving Smoke scenes, the numbers for training time and network size include both the autoencoder and latent space integration networks.

* We optimize the network for subspace simulations rather than the quality of reconstruction, so we do not take this number into account when evaluating the maximal compression ratio.

Limitations

- Extrapolation is limited with the time network
- Cannot recreate arbitrary velocity fields that aren't similar to the training dataset
- No physical constraints on boundaries
- Temporal jittering due to no gradient loss
- Number of control parameters is limited
- No quantitative metrics shown
- Artifacts appear when not near trained parameter settings

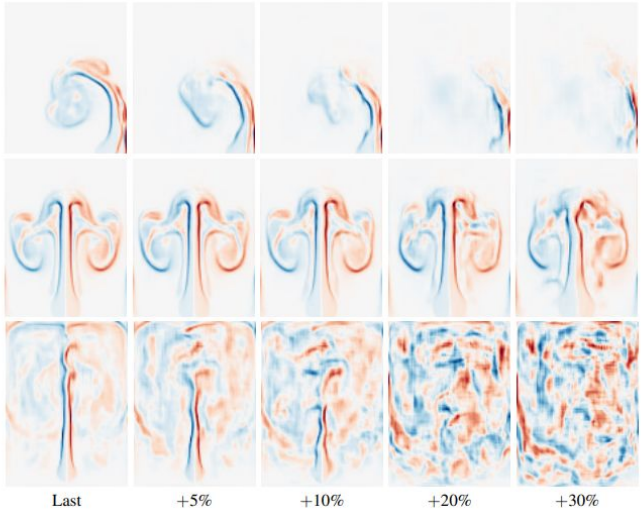


Figure 21: 2-D smoke plume extrapolation results (from t. to b.: position, inflow width, time) where only the generative network is used. Plausible results can be observed for up to 10% extrapolation.

The End