Technische Universität München
TUM School of Computation, Information and Technology

Deep Learning Methods for Simulation of Liquids

Lukas Prantl

Vollständiger Abdruck der von der TUM School of Computation, Information and Technology der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitz:		Prof. Dr. Stephan Günnemann
Prüfer*innen der Dissertation:	1.	Prof. Dr.-Ing. Nils Thuerey
	2.	Prof. Dr. Jan Bender

Die Dissertation wurde am 10.10.2022 bei der Technischen Universität München eingereicht und durch die TUM School of Computation, Information and Technology am 04.04.2023 angenommen.

For peace

Abstract

Liquids, or fluids in general, play a significant role in science and engineering. We also encounter them in everyday life, like in movies or computer games, where they are conspicuous for their chaotic and complex physical behavior. An efficient and accurate simulation remains a major scientific challenge and is part of ongoing research. This thesis proposes new approaches to liquid reconstruction using deep learning. We focus on the complex behavior of gas-liquid interfaces, which comprise a large part of classical liquid scenes. The goal of our methods are fast and easily controllable fluid approximations for computer graphics.

We target a reduced representation of spatio-temporal surface data in the first part of the work. This reduced representation gives the possibility of generating interactive fluid data in a short amount of time. The capacities of our method are determined by a selected set of precomputed data which is interpolated by a deep learning-based approach, allowing for the generation of unseen data within the selected parameter space. Our work shows that the usage of neural networks is essential in this process. The underlying algorithm for interpolating the data is linear, but by extending it with neural networks, the highly nonlinear behavior from the liquids can be approximated appropriately. The final method allows for generating the desired surface quickly. To demonstrate the effectiveness of our method, we have implemented a mobile application based on our method that allows real-time interactions with complex liquid effects.

A limitation of our first method is the lack of generalization to data outside the chosen parameter space. Therefore, in our second work we aim at a method with better generalizability. For this, we target a super-resolution technique for particle-based 3D sequences. The motivation here is that high-resolution simulations are generally very costly to generate due to the high time complexity of conventional solvers. Therefore, we want to enhance rapidly generatable low-resolution simulations using a neural network to approximate a high-resolution variant. Neural networks can be evaluated in linear time $\mathcal{O}(n)$ and are thus not affected as much by dimensionality. For this purpose, we present a generative model for point clouds that provides temporally coherent results. We use a new temporal loss function that considers the higher temporal derivatives of the point positions. We combine these techniques with a truncation approach to flexibly adjust the amount of generated particles and show the generalizability of our approach by inferring large deforming point sets from different test sets. The truncation approach, combined with the ability to evaluate the model only for local subsets of the input, allows us to adaptively apply supersampling only to the surface.

In summary, our two methods permit temporally coherent and fast reconstruction of liquid behavior. The focus on liquids inspired us to develop efficient adaptive methods, but they can also be easily extended to flows in general. The results of our methods show that deep learning-based methods are promising, and we hope they are an inspiration for future work.

Zusammenfassung

Flüssigkeiten, oder im Allgemeinen Strömungen, spielen eine große Rolle in der Wissenschaft und im Ingenieurwesen. Sie begegnen uns aber auch in unserem alltäglichen Leben, wie in Filmen oder Computerspielen, wo sie durch ein chaotisches und komplexes physikalisches Verhalten auffallen. Eine effiziente und akkurate Simulation stellt die Wissenschaft weiterhin vor eine große Herausforderung und ist Teil von aktuellen Studien. Diese Thesis präsentiert neue Deep Learning-basierte Ansätze zur Rekonstruktion von Flüssigkeiten. Wir fokussieren uns dabei auf das komplexe Verhalten der Schnittstelle zwischen Luft und Flüssigkeit, die einen Großteil von klassischen Flüssigkeitsszenen ausmacht. Das Ziel unserer Methoden sind schnelle und leicht kontrollierbare Flüssigkeitapproximationen für die Computergrafik.

Im ersten Teil der Arbeit zielen wir auf eine reduzierte Repräsentation von räumlich-zeitlichen Oberflächendaten ab. Der Vorteil dieser reduzierten Repräsentation ist, dass man unter geringem Zeitaufwand interaktive Strömungsdaten generieren kann. Die Kapazitäten unserer Methode werden dabei von einer gewählten Menge an vorberechneten Daten bestimmt. Die Auswahl der Flüssigkeitsdaten wird durch einen Deep Learning-basierten Ansatz interpoliert, wodurch innerhalb des gewählten Parameterraumes beliebig viele neue Daten generiert werden können. Wie wir in unserer Arbeit zeigen, ist die Verwendung von neuronalen Netzwerken dabei essentiell. Der zugrundeliegende Algorithmus für die Interpolation der Daten ist linear. Erst durch die Erweiterung mit neuronalen Netzwerken kann das stark nichtlineare Verhalten von den Fluiden richtig approximiert werden. Die finale Methode erlaubt es uns schnell eine gewünschte Oberfläche zu generieren. Dementsprechend haben wir auf Basis unserer Methode eine mobile Applikation implementiert, die Echtzeitinteraktionen mit komplexen Flüssigkeitseffekten ermöglicht.

Eine Limitierung unser ersten Methode ist die fehlende Generalisierung auf Daten außerhalb des gewählten Parameterraumes. Deshalb zielen wir in unserer zweiten Arbeit auf eine Methode mit besserer Generalisierbarkeit ab. Dafür arbeiten wir mit einer Super-Resolution-Technik für partikel-basierte 3D Sequenzen. Die Motivation dabei ist, dass durch die hohe Zeitkomplexität von herkömmlichen Solvern hochauflösende Simulationen generell sehr kostenaufwendig zu generieren sind. Deshalb wollen wir schnell generierbare, niedrig-aufgelöste Simulationen anhand eines neuronalen Netzwerks aufwerten und die hochauflösende Variante damit approximieren. Neuronale Netzwerke lassen sich in linearer Zeit $\mathcal{O}(n)$ auswerten und werden somit nicht so stark von der Daten-Dimensionalität beeinflusst. Wir präsentieren dafür ein generatives Modell für Punktwolken, welches zeitlich kohärente Ergebnisse liefert. Für die zeitliche Kohärenz schlagen wir eine neue temporale Verlustfunktion vor, die die höheren zeitlichen Ableitungen der Punktpositionen berücksichtigt. Wir kombinieren diese Techniken mit einer dynamischen Maskierung zur flexiblen Anpassung der Größe der generierten Punktdaten und zeigen die Flexibiltät unseres Ansatzes mit der Inferenz von großen, deformierenden Punktmengen aus verschiedenen Testszenarien. Die Maskierung in Kombination mit der Möglichkeit, das Modell nur für lokale Teilbereiche des Inputs auszuwerten, erlaubt das Supersampling zudem adaptiv nur auf die Oberfläche anzuwenden.

Zusammengefasst ermöglichen unsere beiden Methoden eine zeitlich kohärente und schnelle Rekonstruktion von Flüssigkeiten. Der Fokus auf Flüssigkeiten inspirierte uns dabei zu effizienten und adaptiven Methoden, die aber auch leicht für Strömungen im Allgemeinen angepasst werden können. Die Ergebnisse unserer Methoden zeigen, dass Deep Learning-basierte Methoden vielversprechend sind und wir hoffen, dass sie eine Inspiration für zukünftige Arbeiten sind.

Acknowledgments

I sincerely thank everyone who has supported me in my doctoral studies. The support has made the journey a wonderful time and helped me through harder times. First and foremost, I would like to express my deepest gratitude to my supervisor Prof. Nils Thuerey, who inspired me to choose the path I did and supported me actively. He helped me with his advice throughout my journey, from my bachelor thesis all the way to this doctoral thesis. He gave me freedom in my decisions and made me feel I was being taken seriously. He was patient at times when the research did not go as hoped and helped me in every possible way when the time was running out before a deadline. Finally, I would like to thank him for the regular meetings. Such a high time commitment to supervision cannot be taken for granted. Among other things, this was reflected in his high level of interest and understanding of my research.

A big thanks also go to Prof. Jan Bender, who supported me with his knowledge during my research. His advice in the field of particle-based fluid simulations deemed very beneficial. In particular, I would like to thank him and Prof. Stephan Günnemann for taking over the part of my defense committee. Besides Prof. Jan Bender, I am also grateful to the other co-authors who gave me advice and active help. First of all, Benjamin Ummenhofer, followed by Boris Bonev, Nuttapong Chentanez, Stefan Jeschke, Prof. Vladlen Koltun, and Tassilo Kugelstadt. I enjoyed the collaboration very much and found it very enriching.

I am also very grateful for my colleagues in the department who made the time during the doctoral studies very enjoyable and were a great moral support. First, I want to thank Björn and Stephan for proofreading my thesis, which helped me immensely. Special thanks also go to my office-mates, Steffen and Patrick, as well as to Marie-Lena, Mathias, Michael, Alex, Philipp, You, Rachel, and Kiwon, who provided an excellent start to my doctoral studies. I also had the pleasure of working with Georg, Erik, Nilam, Stephan, Kevin, Behdad, Christian, Sebastian, Liwei, Brener, Benjamin, Robin, Shuvayan, Rene, and Wei, as well as Prof. Rüdiger Westermann, Prof. Matthias Niessner, Prof. Angela Dai, Justus, Andreas, Dejan, Manuel, Christoph, Fatemeh, Josef, and Ludwic. I enjoyed the time together and the discussions during lunch breaks and countless coffee breaks. A special thanks also goes to Susanne, who was very great support and always spread positive vibes. The same goes for Sebastian through his support as administrator. You have all made my time at the university unforgettable.

Last but not least, I would, of course, like to thank my family and friends. First and foremost, my partner Elisa, was my most considerable mental support, especially when things didn't go as planned. I would also like to thank my parents, Ursula and Günther, and my brother Martin. They have always supported me without ifs and buts in everything I have done, and without them, I would not be where I am today. Lastly, I want to mention my friends, who provided the needed balance to my studies during my free time. Thank you very much.

Contents

1 Introduction

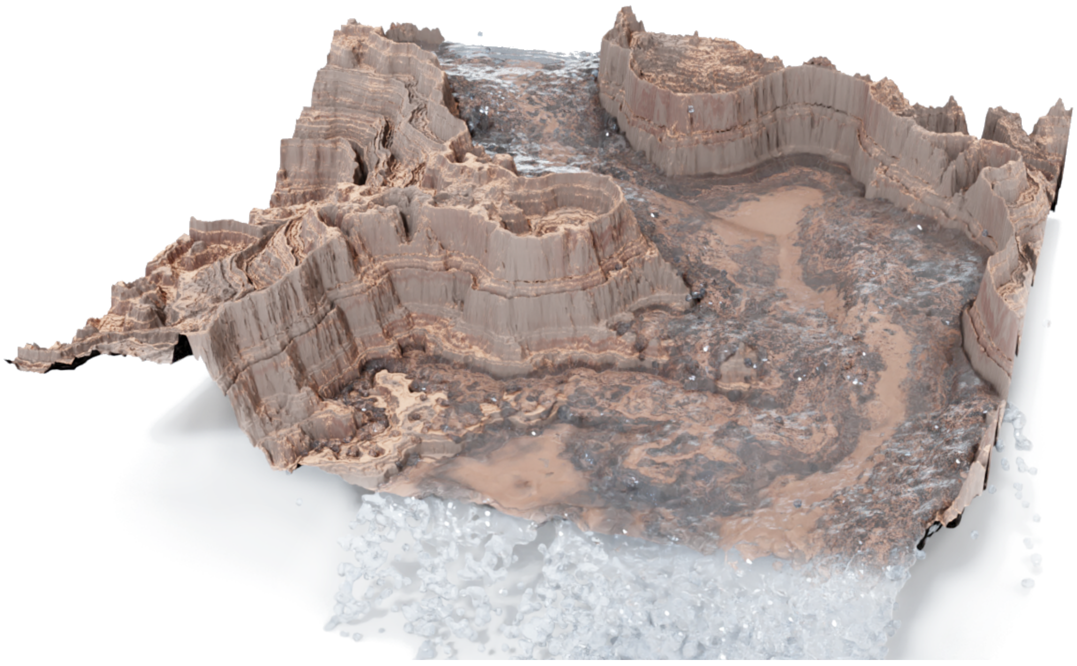
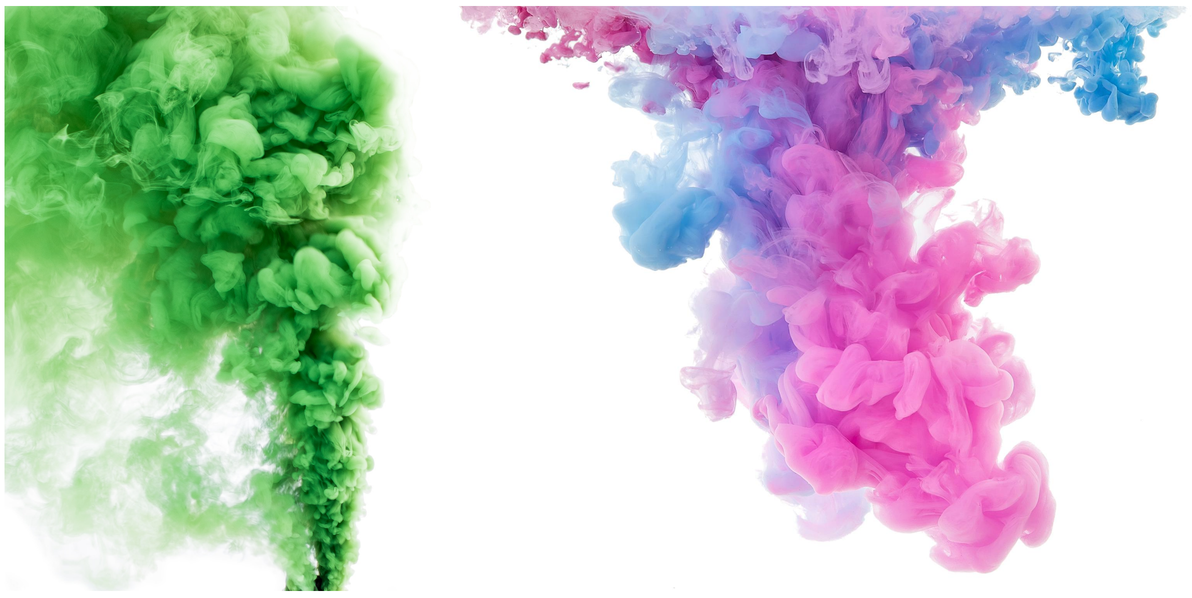


Figure 1.1.: Results based on a deep learning-based fluid simulation. The number of particles is about 150k.

Water, and liquids in general, are ubiquitous in our world. The simulation of liquids is widely applicable and is of great relevance in our everyday life. Be it for engineering applications, the simulation of weather and climate, or predicting the evolution and impact of liquids in natural phenomena, e.g., tsunamis and floods. A visually appealing and interactive implementation of liquid behavior also plays a significant role in computer graphics. There it is used to generate special effects in movies or games.



(a) Smoke plume[1]. **(b)** Ink drops in water[2].

Figure 1.2.: Comparison between smoke and ink in water.

Liquids are subject to the physics of fluids, which includes gas and smoke, but also astrophysical phenomena. Comparing a cloud of smoke (Figure 1.2a) with a drop of ink in a pool of water (Figure 1.2b), the similarity of behavior is clearly visible. In practice, however, liquids are usually simulated in combination with air, creating a phase boundary between water and the air medium (Figure 1.1). Gas simulations, in contrast, are mostly single-phase simulations, where the domain is filled with one single fluid medium, such as air. The phase boundary makes liquids a callous class of physics problems because the constantly changing boundary conditions at the liquid-gas interface result in a complex range of surface motions and configurations. In computer graphics we are particularly interested in surface behavior, as it plays a significant visual role due to striking optical effects, such as reflections and refractions. On the other hand, surface behavior is also crucial for the liquid interaction with solid bodies, e.g., ships on the sea or breaking waves. This observation incentivizes us to address liquid simulations with an increased focus on the surface to optimize the efficiency of our methods. To achieve that, we developed adaptive approaches, which can also be adapted for general fluid simulations or even for another class of materials, e.g., mucus or similar elastoplastic substances.

[1]Photo by *Jeremy Bishop* on *Unsplash*.
[2]Photo by *mystraysoul* on *Pixabay*.

From a numerical perspective, solving the underlying differential equation of the fluid behavior is highly complex. An accurate and at the same time fast solution to fluid simulations is still an open research problem [Wu+18; AHA12; Kos+22; NB16]. Most modern approaches provide robust and elaborate solutions but require us to formulate trade-offs between accuracy and speed. In addition, the minimum resource requirements for liquid simulations are still relatively high, limiting their use in practice. For example, due to the restricted resource availability, there are hardly any physics-based fluid simulations in interactive applications, such as computer games. The necessity of fast solutions forces game developers to replace liquid simulations with rough approximations with low accuracy and high limitations. Even in generating special effects for movies, where there is less need for resource-efficient methods, a fast, scalable, and robust approach for prototyping and testing is beneficial. Due to this, the focus of more recent approaches is based on new advances in deep learning to efficiently approximate parts of the complex behavior of fluids and thus optimize accuracy and speed even further.

1.1. Contributions

In this thesis, we present different methods for approximating liquid behavior. We examine at the problem from a computer graphics point of view and focus mainly on an unconditionally stable, fast, interactive, and visually compelling reconstruction. Conventional numerical solutions typically require a trade-off between accuracy and speed. To alleviate this problem, we build on recent advances in deep learning. The idea is to replace complex and time-consuming steps with a learnable non-linear function that approximates the process. The properties of deep neural networks used in this process allow the approximation of complex non-linear dynamics exhibited by fluids. Our methods are based on geometric deep learning approaches, where we consider the liquid as a spatio-temporal volume that needs to be reconstructed. In addition, we focus on surface behavior, as it mainly determines the visual appearance; the internal dynamics usually play only a minor role. The prioritization of surface reconstruction simplifies the problem without compromising the visual quality of the results. As we will show, the presented methods can outperform conventional methods in terms of stability, accuracy, and speed. In addition, our methods bring a form of controllability that directs the fluid's behavior. We will discuss the presented methods in more detail in the following paragraphs.

Generating Liquid Simulations with Deformation-aware Neural Networks [PBT19] This work presents a novel approach to capturing parameterized spaces of liquid behavior based on space-time deformations. We represent a single 3D input surface sequence as a four-dimensional signed-distance function (SDF). We deform the surfaces in space and time with learned deformations to generate new unseen fluid data. To calculate and represent these deformations efficiently, we take a two-stage approach: First, we span the sides of the original parameter region with pre-computed deformations based on optical flow and infer a suitable weighting function. In the second step, we synthesize a dense deformation field for refinement. As both the parameter weighting problem and the deformation synthesis are highly non-linear, we demonstrate that neural networks are a particularly suitable solver for robustly finding solutions.

Tranquil Clouds: Neural Networks for Learning Temporally Coherent Features in Point Clouds [Pra+20] In the second paper, we consider general spatio-temporal volume generation. Unlike the first work of this thesis, we do not consider implicit data in the form of grids but an explicit, particle-based representation. Given a sequence of point clouds, our method allows the generation of a temporally coherent, higher-resolution point cloud. Thereby, details learned in training are generated based on the spatio-temporal dynamics of the input and previously generated frames. To further reduce the computational task, the method limits the details' generation only to the volume's surface. Besides liquids, the geometric-based approach allows various possible applications involving spatio-temporal point clouds, as we will show in our paper.

1.2. List of Publications

Our two works accompanying the thesis have been published as full papers at the peer-reviewed *International Conference on Learning Representations* (ICLR):

- Lukas Prantl, Boris Bonev, and Nils Thuerey. "Generating Liquid Simulations with Deformation-aware Neural Networks". In: International Conference on Learning Representations. Feb. 20, 2019
- Lukas Prantl, Nils Thuerey, Nuttapong Chentanez, and Stefan Jeschke. "Tranquil Clouds: Neural Networks for Learning Temporally Coherent Features in Point Clouds". In: International Conference on Learning Representations. Mar. 11, 2020

1.3. Outline

The thesis is divided as follows. Chapter 2 deals with relevant background information and related work that plays a role in our methods. It intends to give a general overview of the current state of deep learning and fluid simulations and shows where our work is positioned in it. In Chapter 3, on the other hand, we discuss the methods we have presented. We briefly discuss our approaches and put them in a shared context. In Chapter 4, we list the publications again as an overview. Finally, we discuss the results in Chapter 5 and a potential outlook for future work based on the proposed methods.

2 Fundamentals and Related Work

The behavior of fluids relevant to computer graphics can be defined by the incompressible Navier-Stokes equations:

$$\frac{\partial \vec{u}}{\partial t} + \vec{u} \cdot \nabla \vec{u} + \frac{1}{\rho} \nabla p = \vec{g} + \nu \Delta \vec{u} \tag{2.1}$$

$$\nabla \cdot \vec{u} = 0 \tag{2.2}$$

where $\vec{u}$ corresponds to the velocity of the fluid, ρ is the density, p the pressure, ν the kinematic viscosity, and $\vec{g}$ stands for external forces such as gravity. We can divide the equation into two essential parts, the momentum equation (Eq. 2.1) and the incompressibility condition (Eq. 2.2).

The momentum equation describes how the fluid accelerates under the effect of different external forces $\vec{g}$. Decisive for the forces acting on the fluid is the direct effect of the external forces $m\vec{g}$, with m corresponding to the mass. In addition, the fluid parts induce internal forces themselves. A part of them consists of forces caused by the pressure gradient ∇p. Areas with high pressure push into areas with low pressure. Apart from that, there are forces caused by viscosity $\nu \Delta \vec{u}$. We can interpret viscosity as an internal frictional force of the fluid. For instance, fluids with high viscosity, such as honey or tar, tend to have a higher resistance to deformation. In practice, viscosity is usually ignored for fluids with negligible small viscosity, such as water. The second part of the equation is the incompressibility condition (Eq. 2.2). Incompressibility refers to the conservation of the volume of fluids. It is essential to know that fluids do change their volume in reality. However, the changes in volume are very tiny and have almost no effect on how fluids move at a macroscopic level, especially for liquids. Since the calculation of compressible fluids is also very expensive and complex, we impose the simplifying assumption that fluids are incompressible in most cases. Incompressibility is obtained by conditioning the velocity field to be divergence-free, attained by the pressure. To derive the desired pressure value, we relate the incompressibility condition with the pressure from the momentum equation:

$$\nabla \cdot \frac{1}{\rho} \nabla p = -\nabla \cdot (\vec{u} \cdot \nabla \vec{u} + \vec{g}). \tag{2.3}$$

Here the viscosity was already omitted for simplification. For a more detailed derivation and explanation of the Navier-Stokes equation for fluid simulations, we refer to Bridson [Bri15].

2.1. Fluid Simulation

Numerical computation of fluid-based dynamics is a long and extensively treated topic in research [HER55; Wil06]. The first important step in a numerical simulation is the data's discretization and representation. For this, we roughly distinguish between Lagrangian and Eulerian viewpoints. Lagrangian methods correspond to particle systems, which represent the fluid data explicitly in the form of moving particles. In contrast, the Eulerian viewpoint uses a spatially fixed grid, where the fluid properties are stored in the particular cells through which the fluid flows. Both representations have certain advantages and disadvantages, with none dominating the current literature or practical applications.

The basic method for solving Lagrangian-based methods stems from astrophysics [GM77] and is known as *Smoothed Particle Hydrodynamics* (SPH). Many additional works have proposed improvements of the SPH method in terms of accuracy and performance [BT07; SP09; MM13; BK16; Hu+19; AHA12]. The particle-based approach has the advantage that it conserves the mass of the fluid by design and limits valuable computation time to the location of the fluid. Apart from that, there is usually less dissipation, as in the Eulerian viewpoint, since the advection does not require interpolated values. However, unlike Eulerian approaches, unconditional robustness is challenging to implement. This is caused by the explicit advection of particles, whereas for grids we can work with implicit advection schemes [Sta99; SC91]. In addition, grid representations are spatially regularly defined, while particle-based representations are sparse and irregular. For instance, interpolated values and derivatives at positions between the particles are more challenging to obtain than for implicit representations. Similar to SPH simulations, there has been great progress in Eulerian simulations [FM96]. Current methods rely, for example, on multi-grid methods for fast computation [BL11; MST10; CM11]. To counteract the dissipation problems, grid-based methods have also been combined with a Lagrangian scheme. These hybrid approaches were first mentioned in Harlow et al. [HW65] and are referred to as *Particle-in-Cell* (PIC) methods. They were improved later on by Brackbill and Ruppel [BR86], by a method called *Fluid-Implicit-Particle* (FLIP). PIC/FLIP-based approaches still dominate state-of-the-art results today [Jia+15; Cor+14; Fer+16; Sat+18], alongside advanced SPH simulations. Another special line of work is based on shallow water equations [Vre94], established in computer graphics [Woj+17] or particular areas like tsunami simulations [Gis08]. Unlike the previously discussed solvers, they are only approximate methods limited to surface simulation on a two-dimensional heightfield. The approach is usually limited to solving wave equations which are highly parallelizable, allowing the fast simulation of large-scale fluids. Shallow water solvers are related to our approaches in prioritizing surface behavior for higher efficiency. However, our methods can still catch fluid-internal behavior and be applied in three dimensions, while shallow water simulations are limited to heightfields.

Regardless of the different approaches and viewpoints, most numerical methods solve the Navier-Stokes equation with a simplification through operator splitting. The equation is divided into three parts, *advection*, *external forces*, and *projection*, which are solved separately for each time step of a sequence in the following order:

Advection

$$\frac{\partial q}{\partial t} + \vec{u} \cdot \nabla q = 0 \tag{2.4}$$

Here, the advection generally describes the transport of quantities q based on the fluid's velocity field. In the case of particle-based methods, this corresponds to a simple integration step of the particles based on the respective velocity. In grid-based approaches, on the other hand, a semi-Lagrangian advection [SC91; Sta99] is usually used. The quantity transported can be any feature of the fluid, but the advection of the velocity $q = \vec{u}$ itself is crucial for further steps.

External Forces

$$\frac{\partial \vec{u}}{\partial t} = \vec{g} \tag{2.5}$$

After the advection step, the effects of external forces are included, which usually corresponds to an integration step of the velocity with gravity.

Projection

$$\frac{\partial \vec{u}}{\partial t} + \frac{1}{\rho}\nabla p = 0 \quad s.t. \quad \nabla \cdot \vec{u} \tag{2.6}$$

Finally, the resulting velocity field is made divergence-free by adjusting the internal pressure forces to satisfy the incompressibility condition. This step is usually called a *projection* in the literature. It requires solving a global linear system of equations, which is generally very resource intensive. The result is a divergence-free velocity field that can be used for the advection step in the next time step, where the process restarts. Solving the pressure equation is usually the most expensive and complex step in solving the fluid equation. Many methods, therefore, start here when it comes to optimizing the speed of the solver.

2.1.1. Adaptation of Fluid Solvers

Modern numerical methods are very generically applicable and define a high standard regarding various properties in fluid simulations, whether for accuracy or for speed. However, numerical solvers quickly reach their limits in real-world computer graphics applications. Due to the curse of dimensionality, high-resolution simulations become very expensive in computation time, despite high-performance solvers. Another problem is the chaotic, strongly nonlinear behavior of fluids. A small change in the initial conditions can significantly affect later behavior. An example can be seen in Figure 2.1. The significant time requirement and prediction difficulty make flow simulation very hard to handle, especially when designing fluid scenes, such as special effects in movies. The goal is to attain a high-quality fluid simulation with behavior that matches the envisioned concept. The fluid complexity makes it hard to find the correct initial parameters right away, and the high-quality requirements result in a very long simulation time, making multiple iterations expensive. An obvious solution is to generate a low-resolution, quickly computable simulation, which can be iterated on faster until the result corresponds to the desired outcome. Subsequently, the desired high-resolution variant can be generated based on the determined parameters. However, the simulation's resolution can influence the outcome, complicating the approach in practice. Better solutions are post-processing methods, which subsequently enhance fluid simulations. Thereby, the resolution, and thus the necessary time, of the basic simulation can be reduced. The post-processing method is then applied to the base simulation to retain the original rough shape and approximate the behavior of a high-

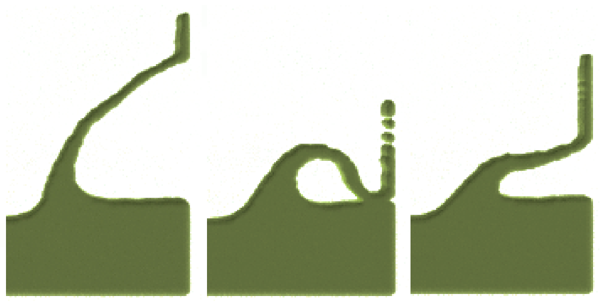


Figure 2.1.: Three liquid surfaces after 60 time steps differing only by $\pm\epsilon$ in initial conditions. Even this initially minimal difference can lead to large differences in surface position.

resolution variety. For example, in Kim et al. [Kim+08], adaptive turbulence is added based on wavelets. In Chu & Thuerey [CT17], on the other hand, machine learning is used, replacing low-resolution fluid patches with matching high-resolution variants. Another approach is to control the flow behavior by formulating an inverse problem. The fluid behavior is modified, e.g., by induced external forces, so that a set of specific constraints are respected. This approach also applies to reconstructing fluid behavior from sparse and noisy data. Besides linear solvers [EUT19], more and more methods based on differentiable physics are used for this purpose [HKT20; Um+20; SF18]. The approach in our first work is also centered around an inverse problem, while our second work can be seen as part of the post-processing methods.

Especially in this application-oriented area, deep learning methods clearly stand out. Deep learning-based solvers achieve comparable performance as state-of-the-art numerical solvers while at the same time outperforming traditional solvers by orders of magnitude in terms of runtime. However, deep learning and classical methods should not be considered complementary. As already shown in the field of differentiable physics, the combination of classical numerical methods and deep learning is very promising. The benefits of using domain knowledge were also shown in recent publications [Wan+20; Cai+22; Pra+22a].

2.2. Deep Learning for Fluids

The idea of using deep learning is to speed up fluid behavior computation by using the processed data's statistics. In contrast to many classic simulators that use sophisticated and handcrafted model equations to describe the motion of fluids [Pop00; Tez+96], deep learning-based methods belong to the class of data-driven solvers, which learn representations entirely from observations. Methods based on reduced representation like Dunteman [Dun89], Treuille et al. [TLP06], De Witt et al. [DLF12], Raveendran et al. [Rav+14], and Thuerey [Thu16] follow a similar objective. The same is true for machine learning methods like Chu & Thuerey [CT17] and Ladicky et al. [Lad+15]. However, the rise of deep learning methods has impacted application scenarios beyond the fluid simulation domain. The flexibility and power of deep learning methods have quickly surpassed conventional machine learning methods and dominate the current research landscape.

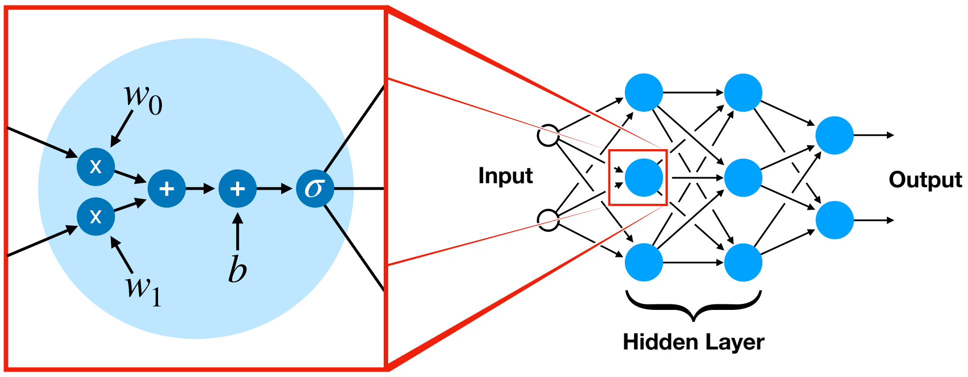


Figure 2.2.: Illustration of a deep neural network. The detail on the left shows the setup of a single neuron, with $w_{\{0,1\}}$ and b being the trainable weights multiplied or added to the inputs, respectively. σ is the non-linear activation function applied at the end. The right shows a neural network with multiple layers consisting of neurons (blue circles). The empty circles represent the input layer.

2.2.1. Deep Neural Networks

Deep learning [HS06; HOT06] has experienced a real boom in recent years due to the development of backpropagation [RHW86b] and access to an increasing amount of computational resources. The basis for neural learning is the so-called neuron (Figure 2.2). The neuron represents a simple linear function whose output can be transformed by a non-linear activation function σ. The parameters of the neuron, w_* and b, are trained with gradient descent such that the neuron's output corresponds to the desired output. Several neurons can be used in parallel, organized as a layer, to solve simple problems. For more complex, non-linear problems, multiple fully-connected layers can be stacked sequentially. However, the training of models with multiple layers is challenging. Backpropagation proposed an efficient way to update the parameters of several successive layers, the basis of deep learning. We briefly discuss fundamental advances in deep learning in the following, which are relevant to our methods. For a more general and detailed introduction to deep learning, we refer to Nielsen [Nie15].

2.2.2. Convolutional Layers

Neural networks based on fully-connected layers are very generic and robust. However, for spatially dependent problems, they are often replaced by convolution layers [Fuk79]. Convolution layers are characterized by their translation invariance and a spatial inductive bias in the form of a spatial kernel. That means, unlike fully-connected layers, the convolution is limited to the local reference frame of a data point and its surrounding neighbors. The efficiency of these layers has led to them quickly becoming a standard tool in deep learning. Especially in the field of fluids, this development plays an essential role since the data is primarily spatio-temporal. Fully-convolutional neural networks (CNNs) are used, for example, in the seminal works of Tompson et al. [Tom+17], Um et al. [Um+20], Kochkov et al. [Koc+21], and Yang et al. [YYX16]. Based on the success of CNNs, methods emerged that could apply the concept of convolution not only to structured data, such as images but also to unstructured data, such as point clouds or SPH simulations. Prominent work in that regard are PointNet [Qi+17a] and PointNet++ [Qi+17b]. Many following works applied PointNet++ to solve different tasks on point clouds [Gue+18], including PU-Net [Yu+18; Yin+18], on which our second method [Pra+20] is based. At a similar time, graph neural networks (GNNs) were introduced [Bat+16; Li+19; San+20] to handle graph-structured data. GNNs are used in many state-of-the-art fluid prediction methods [Li+19; San+20; Pfa+20; BWW22]. Even though PointNet++ and GNNs consider spatial neighborhood relations, they generally do not use kernel functions like convolutions. Using a kernel can be seen as a spatial-based bias, which allows for more efficient models since the neural network does not have to learn the spatial relationships first [SF18; Wan+18; Fey+18]. For example, the efficiency of convolution-based methods for unstructured data has been shown in Ummenhofer et al. [Umm+20], which implements a trainable continuous convolution operator resulting in SPH simulations with compact networks. Transformer models [Vas+17; Zha+21] can also be interpreted as a special kind of convolution. They adopt the self-attention mechanism by differentially weighting the significance of the input data. Similar to convolution, spatial or temporal dependencies are accumulated in a weighted manner, but the weighting is data dependent and normalized.

2.2.3. Temporal Learning

Convolutional layers cover the spatial aspect of fluid data well. For physical data, however, temporal evolution also plays an important role. A relatively obvious application are again convolutions, which are also used in the time dimension [Lea+16; Hew+20; Wan+19; BKK18]. We take inspiration from this principle in our first method, which uses 4D spatio-temporal convolution. Another line of work is based on recurrent neural networks (RNNs) [RHW86a; Sch93], from which long-short term memory (LSTM) modules [Gra12] have been developed. LSTMs use a particular structure to generate and store an internal representation. They have been successfully used for fluid simulation prediction [WBT19] and are widely used but have recently been surpassed by transformer models [Vas+17]. These sequence-based methods are compelling but are sometimes expensive and laborious to use and train. In physics-based simulations, temporal coherence through temporal loss functions has become more popular. A common approach is to process a data sequence and evaluate the results concerning temporal coherence without keeping an internal temporal state as in LSTMs. For example, our second method uses position, velocity, and acceleration deviations to the reference solution. Other methods like tempoGAN [Xie+18] or TecoGAN [Chu+18b] use a perceptual, adversarial loss to ensure temporal coherence.

2.3. Transformations and Deformations

Even though the physical laws are crucial for learning fluid dynamics, we also have to consider the geometric aspect of the data. In particular, our methods primarily address a geometric-based solution to the problem. From this perspective, fluid data are sequences of volumes represented as grids or particles, depending on the representation. Regardless of the data-generating process and the underlying physical rules, our work aims to generate liquid volumes that are part of the distribution of reference volumes based on ground-truth simulations. This geometric perspective opens up many other aspects and approaches that can be helpful for an efficient solution. Thus, the idea of our first method [PBT19] is based on considering fluid simulations as spatio-temporal volumes. With the help of spatio-temporal interpolation between given fluid volumes, new ones can be generated dynamically and quickly. We implement the interpolation by a partial deformation between fluid volumes. The deformation describes an advection with a vector field generated with an optical flow solver [Thu16]. The idea is based on methods for optical flow inference and image correspondences [BVS16; Dos+15; RB17; Ilg+17]. However, we learn deformations and their weighting in an unsupervised manner without detailed ground truth data. Thus, our method shares similarities with spatial transformer networks (STNs) [Jad+15], and unsupervised approaches for optical flow [MHR18].

For the second method [Pra+20], we are also guided by a geometric-based method, super-resolution. Super-resolution refers to a set of methods that attempt to generate high-resolution data from low-resolution data. This approach is ubiquitous in image or movie data but also plays an increasingly important role in geometric data processing. Single image super-resolution was targeted in several works [Led+16; SSH17], some of which employed stochastic formulations [GGY18]. Video upsampling with recurrent approaches was proposed in [SVB18], which was extended by follow-up works that considered CNN-based temporal coherence [Chu+18a; Pér+18]. Upsampling of 3D smoke volumes over time was likewise targeted in Chu & Thuerey [CT17] and in Xie et al. [Xie+18]. The work by Yu et al. [Yu+18] is one of the first to use deep learning for super-resolution for Lagrangian data similar to ours.

3 Generation of Liquids

This chapter overviews the proposed methods and puts them in a shared context. Their common goal is to learn liquid behavior based on given initial conditions. Let us assume a Navier-Stokes boundary value problem with a liquid-gas interface Ω (Figure 3.1), which depends on the initial conditions c. We treat the fluid dynamics as a single space-time volume $\Omega(c) \subseteq \mathbb{R}^4$. Our goal is to implement a learnable model $G(c)$ to capture and approximate the space of volumes for a given set of initial conditions $c \in C$, generated with an accurate numerical solver.

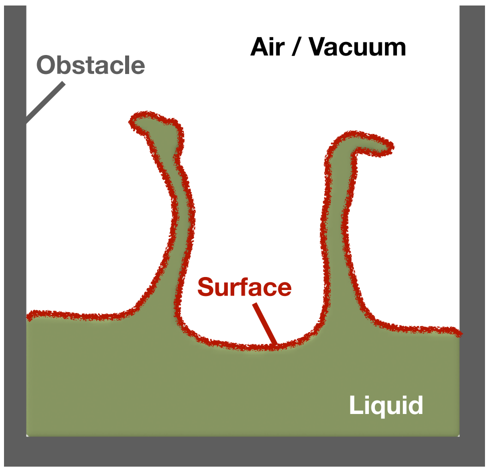
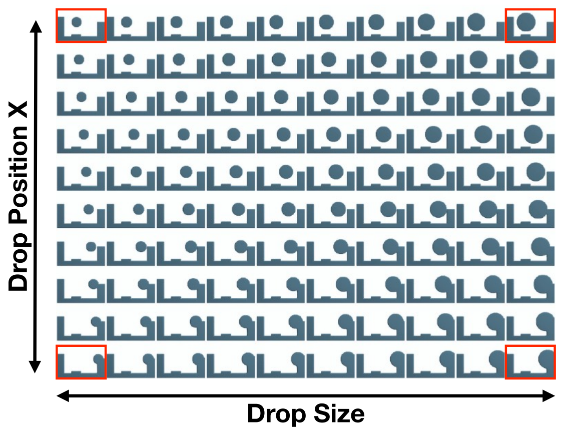


Figure 3.1.: Illustration of a classic liquid setup with a free surface.

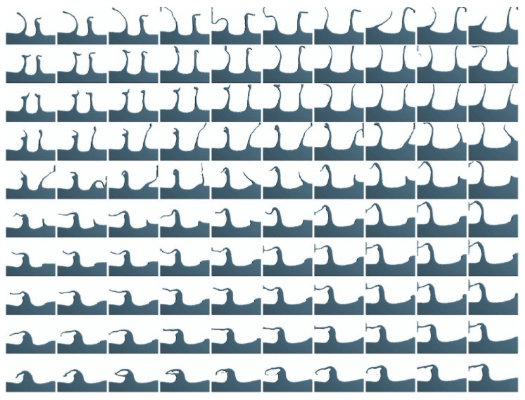
We deliberately want to focus on the surface behavior and pay less attention to the fluid-internal behavior as the volumetric dynamics have little effect on the visual perception of the fluid [UHT17]. The simplification positively impacts the temporal performance and generalizability of the problem without much loss of quality. Accordingly, we designed our methods to be adaptive so they can be explicitly applied to the surface. Since the surface is only a fraction of the domain, this can drastically reduce the runtime and memory requirements. Moreover, it simplifies the learning task since less critical dynamics do not need to be learned. We individually discretize all the functions and operators for our methods because we use different representations for both approaches. In the first method, we consider the problem from the Eulerian viewpoint, while a Lagrangian viewpoint was chosen in the second paper. Both representations have advantages and disadvantages and are widely used. Given the presented problem formulation, we addressed mainly the following points:

- Learning reduced representations of liquid spaces.
- Generation of fast, controllable approximations of liquid behavior.
- Adaptive, temporally-coherent enhancement of liquids with small-scale details.

3.1. Reduced Representations of Liquids



(a) Initial conditions.



(b) Input data at $t = 30$.

Figure 3.2.: The left image (a) illustrates an example of initial conditions for a two-dimensional parameter space setup. It consists of a set of two-dimensional liquid simulations, in which we vary the position of the liquid drop along x as α_1 and its size as α_2. We highlighted possible key simulations in red. The right half (b) shows the data used for training at $t = 30$. Note the significant variance in positions of small-scale features such as the thin sheets.

In our first work [PBT19], we focus on learning a reduced representation of a set of fluid volumes $\Omega(c)$. The goal is to use it to quickly and efficiently approximate the dynamics of $\Omega(c)$. The set is defined over a chosen region of input conditions C. In practice, C could contain any simulation parameters, e.g., initial positions or physical parameters such as viscosity. In the example of Figure 3.2, C consists of the varying position and scale of the drop falling into the basin. The full dynamics of the data are entirely dependent on C.

We discretize the data in the form of four-dimensional signed-distance function (SDF) grids:

$$\Phi_c \in \mathbb{R}^{N_x \times N_y \times N_z \times N_t},$$

with $N_{\{x,y,z,t\}}$ being the dimension of the grid. SDFs are used for an implicit representation of the data. The cells of an SDF grid contain signed scalars indicating the minimum distance to the surface. The value zero defines the surface, while negative values stand for the fluid volume and positive for the space around the fluid. In addition, we map the selected condition factors $c \in C$ onto a linear, normalized N_c-dimensional vector $\alpha \in [0,1]^{N_c}$, where N_c is the dimension of parameter space. The goal is to generate a reduced representation of the discretized simulation set Φ_α, with the normalized key factors α as conditions. Thus, the proposed generative network G has the task of learning a mapping from the linear parameter space α to an approximation of the nonlinear fluid behavior Φ_α:

$$G(\alpha) \approx \Phi_\alpha. \tag{3.1}$$

To achieve this, we work with a deformation-based interpolation between predefined key simulations. The key simulations are a subset of the reference simulations based on deliberately chosen parameters. Usually, the selection of parameters consists of the extreme cases, i.e., where alpha values contain either 0 or 1. For example, in Figure 3.2, the scenes where the droplet is either smallest/biggest, combined with where the position is leftmost/rightmost, could be considered key simulations. We can then generate new unseen surfaces by interpolating the surfaces of this pre-computed subset.

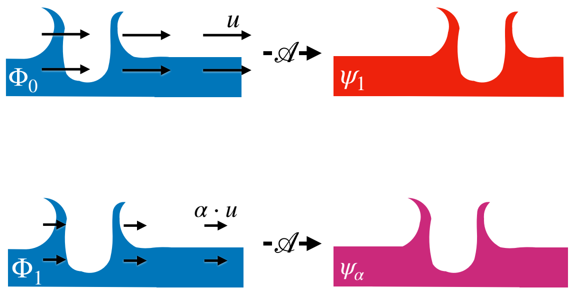


Figure 3.3.: Illustration of the deformation-based interpolation between two surfaces Φ_0 and Φ_1. In the top row, the advection u is applied without weighting, resulting in the approximation ψ_1. In the bottom row, the vector field is weighted by a factor $\alpha \approx 0.5$ generating an interpolation between both surfaces.

The interpolation is based on an improved version of the work by Thuerey et al. [Thu16]. The basic idea is to generate a vector field u between two predefined surfaces, Φ_0 and Φ_1, similar to optical flow. Using this vector field, we can deform the initial surface Φ_0 into an approximation ψ_1 of the other by advection $\mathcal{A} : \mathcal{A}(\Phi_0, u) = \psi_1 \approx \Phi_1$. By weighting the advection, intermediate states can be obtained $\mathcal{A}(\Phi_0, \alpha \cdot u) = \psi_\alpha \approx \Phi_\alpha$, corresponding to the desired interpolation (Figure 3.3). We directly use α as the interpolation factor, which is possible because it has been explicitly normalized for this purpose. Using this deformation-based method based on a simple advection, we can

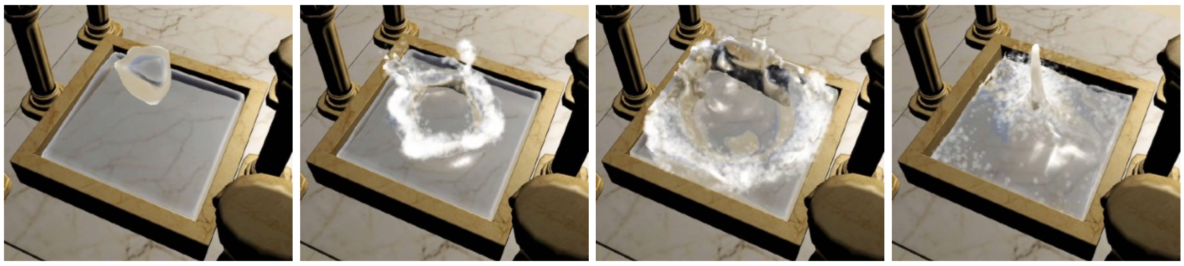


Figure 3.4.: Sample results of our method running on a smartphone in real-time.

interpolate between a few key simulations and thus approximate unseen intermediate states very quickly. Apart from that, the approach requires only one initial reference surface to reconstruct the entire parameter space because all the deformation vector fields can be pre-computed. As the initial surface for inference we typically use the zero point of our parameter space, i.e., $\psi_0 = \Phi_{\alpha_0}$, with $\alpha_0 = (0, 0, \ldots, 0)$. The used advection can take different forms, such as a stable semi-Lagrangian advection [SC91] used primarily on fluid simulations. In our case, we use a more advanced method that combines semi-Lagrangian advection with a forward advection. For more details, we refer to the paper [PBT19].

The deformation-based interpolation is a reasonable basis for a reduced representation. However, the approach is linear, and since fluid behavior can be strongly nonlinear, it cannot capture fluid behavior appropriately. Therefore, we extend the method with neural networks. Specifically, we use two fully-convolutional networks; the first network learns the mapping between initial conditions and the parameter space to $G_p(\alpha) = \beta$. The second network corrects the deformation field used for the final advection $G_d(u, \alpha) = v$, allowing for the required non-linearity. The final result is computed as $G(\alpha) = \mathcal{A}(\psi_0, \beta \cdot v)$.

Our method reduces the space of surfaces to one initial surface, some pre-computed deformations, and two small models. The compressed representation results in a small memory footprint and fast inference. We show that the method is executable in real-time, even on devices with reduced computational power, such as smartphones (Figure 3.4). In addition, the method allows for the possibility to approximate intermediate states in a controlled and fast fashion based on a few key simulations. This is especially interesting for rapid prototyping, where parameters must be tuned for the desired result, especially when the parameters have a non-linear influence. Our method maps the considered linear space of the parameters to the non-linear space of the liquid behavior, providing a simple way to interact with it. Finally, these advances are not limited to liquid data but could also be used for other volumetric data.

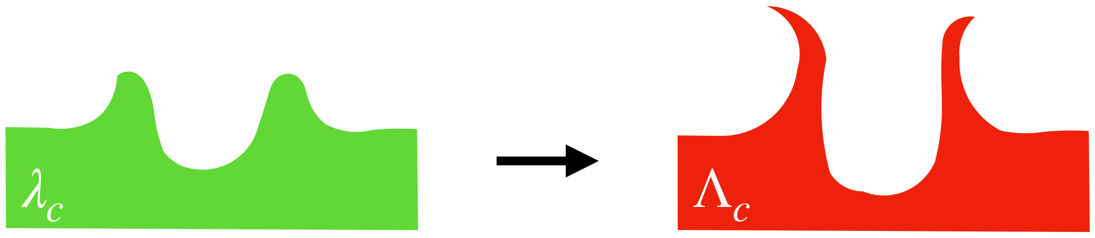


Figure 3.5.: Illustration of a low-res. fluid surface λ_c and the corresponding high-res. counterpart Λ_c.

3.2. Enhancing Fluid Simulations

Our second paper aims to enhance an existing simulation by generating appropriate spatio-temporal details. The approach achieves that by generating a super-resolution output, approximating a high-resolution simulation Λ_c. The super-resolution is conditioned on a low-resolution simulation λ_c generated with the same initial conditions c as the high-resolution reference. Both references are discretizations of Ω_c. An example of a corresponding low- and high-resolution surface can be found in Figure 3.5. Similar to the first method, we generate unseen fluid data by transforming a pre-calculated simulation to quickly generate a controllable water simulation. The difference, however, is that no constant, predefined simulation is used as input but rather a low-resolution simulation that can be generated quickly. This gives the advantage that the method generalizes and extrapolates much better and is not limited to a reduced parameter space. Apart from that, we discretize the data with particles, a standard representation for liquids.

An inherent difficulty of point-based data is their lack of order, which makes operations such as convolutions, which are easy to perform for Eulerian data, more complicated. This comes with the challenge of generating temporally coherent data. Several powerful approaches for point-based neural networks have been proposed [Qi+17a; Her+18; Qi+17b; HTY18], and we leverage a similar neural network architecture in conjunction with the permutation-invariant Earth Mover's Distance (EMD) to propose the formulation of a loss for temporal coherence. In addition, several works have recognized the importance of training point networks for localized patches. In this way, we avoid having the network rely on a full view of the whole data set for inherently local tasks, such as normal estimation [Gue+18], and super-resolution [Yu+18]. This patch-based approach also makes it possible to flexibly process inputs of any size. Due to this, our method allows adaptive super-sampling of the input, limited to the surface, to maximize efficiency. In conjunction, the network dynamically determines the degree of upsampling. As a result, the resolution of the generated data is controlled as needed, optimizing the cost-to-quality ratio. A general challenge here is to deal with varying input sizes and, for super-resolution tasks, also varying output sizes.

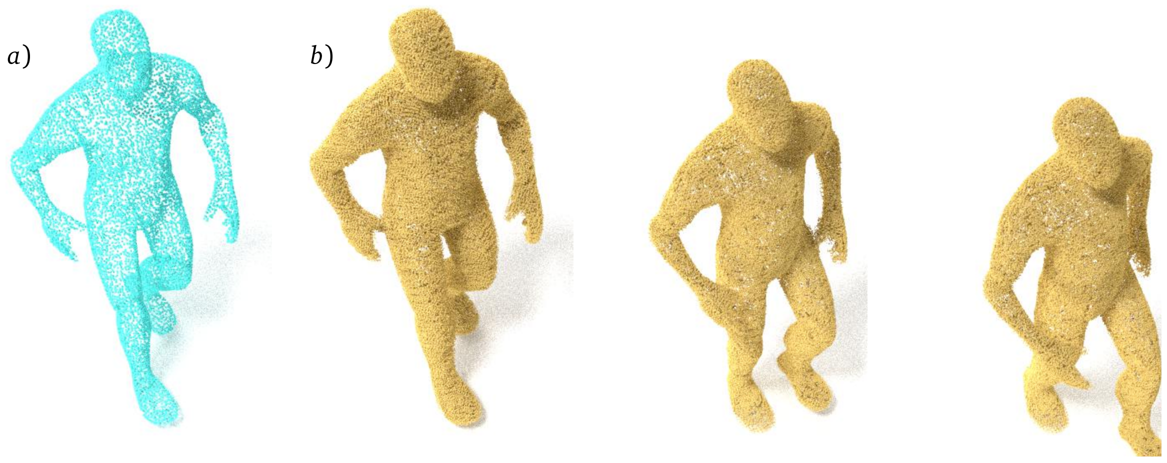


Figure 3.7.: Our algorithm upsamples an input point cloud (a) in a temporally coherent manner. Three exemplary outputs in (b).

For upsampling, we use a generative neural network based on PointNet++ [Qi+17b], a prevalent network architecture for unstructured data:

$$f_s(\lambda_c) \approx \Lambda_c.$$

λ_c is the low-resolution input consisting of particle positions X and velocities V. We train the model with an EMD loss, which calculates the difference between the generated output and the reference. Besides the spatial distance, we evaluate differences in velocity and acceleration to learn the temporal dynamics. Therefore, we run the model three times during one training step with time-shifted frames (Figure 3.6). We can then compute the required data from the three generated frames using finite differences. This parallel execution of a model is also called a *Siamese* setup in the literature [BC93]. Our final method works mainly with geometric data, i.e., the position and velocity of the individual particles. Thus, the method is very generic and can be easily applied to other problems. As an example, Figure 3.7 shows the upsampling of a human-based mesh, where the network used for this purpose was trained with fluid data.

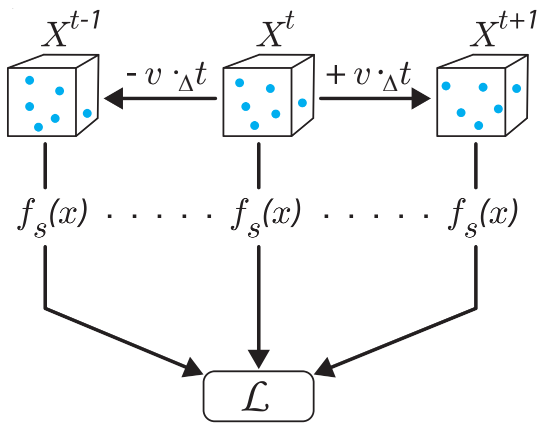


Figure 3.6.: Siamese network setup for temporal loss calculation. t denotes the time; $\mathcal{L}$ stands for the loss calculation used for the training.

4

Summary of Papers

A. Generating Liquid Simulations with Deformation-aware Neural Networks

Abstract of Paper We propose a novel approach for deformation-aware neural networks that learn the weighting and synthesis of dense volumetric deformation fields. Our method specifically targets the space-time representation of physical surfaces from liquid simulations. Liquids exhibit highly complex, non-linear behavior under changing simulation conditions such as different initial conditions. Our algorithm captures these complex phenomena in two stages: a first neural network computes a weighting function for a set of pre-computed deformations, while a second network directly generates a deformation field for refining the surface. Key for successful training runs in this setting is a suitable loss function that encodes the effect of the deformations, and a robust calculation of the corresponding gradients. To demonstrate the effectiveness of our approach, we showcase our method with several complex examples of flowing liquids with topology changes. Our representation makes it possible to rapidly generate the desired implicit surfaces. We have implemented a mobile application to demonstrate that real-time interactions with complex liquid effects are possible with our approach.

Author Contributions The first author contributed a great deal to the idea, implementation, and some studies presented in this work. **Boris Bonev** assisted in implementing the neural network. **Nils Thuerey** took over a part of the evaluations. The paper was written in equal parts by all authors.

B. Tranquil Clouds: Neural Networks for Learning Temporally Coherent Features in Point Clouds

Abstract of Paper Point clouds, as a form of Lagrangian representation, allow for powerful and flexible applications in a large number of computational disciplines. We propose a novel deep-learning method to learn stable and temporally coherent feature spaces for points clouds that change over time. We identify a set of inherent problems with these approaches: without knowledge of the time dimension, the inferred solutions can exhibit strong flickering, and easy solutions to suppress this flickering can result in undesirable local minima that manifest themselves as halo structures. We propose a novel temporal loss function that takes into account higher time derivatives of the point positions, and encourages mingling, i.e., to prevent the aforementioned halos. We combine these techniques in a super-resolution method with a truncation approach to flexibly adapt the size of the generated positions. We show that our method works for large, deforming point sets from different sources to demonstrate the flexibility of our approach.

Author Contributions The first author is responsible for the overall idea, implementation as well as all studies presented in this work. The paper was composed by the first author under consideration of feedback as well as minor revisions by **Nuttapong Chentanez**, **Stefan Jeschke**, and **Nils Thuerey**.

5 Conclusion

This thesis discussed new deep learning methods for liquids in computer graphics. We looked at the differences in liquids' properties compared to other fluids or physical phenomena. Assuming the context of how we perceive liquids in practice, we concluded that the surface plays a decisive role. Based on this finding, we looked at ways to enhance liquid behavior using deep learning methods. We demonstrated the power of neural networks and how they can be used efficiently to outperform conventional methods. This is reflected in the results, which allow real-time interactions with physical effects that are orders of magnitude slower to compute with traditional solvers [PBT19]. In addition, our second work showed how to maximize efficiency and generalizability using an adaptive, geometric-oriented approach. The presented temporal loss allows the learning of temporal dynamics and provides temporal coherence.

5.1. Outlook

During the development of our methods, we encountered several limitations that can be an inspiration for future work. We roughly divide the main findings into three parts:

Generalizability Generalizability plays a significant role in deep learning-based methods and machine learning in general. It refers to the performance of the method in the context of unseen test scenarios, i.e., the ability to extract properties inherent in a data generation process from a training data set. In the context of physical simulations, that is usually a method that can identify, extract, and learn the underlying physical laws. The goal is to ensure that the method is bias-free and does not overfit to specific scenarios. Usually, this is achieved by making the training data set independent and identically distributed (i.i.d.), with high variance for a better generalization. However, the complexity and diversity of big data sets, including fluid simulations, in the form of countless degrees of freedom usually make the data set's composition difficult. The similarity to the true distribution of the data is challenging to quantify, making it almost impossible to generate data sets that are uniformly distributed and without bias. To counteract the resulting distribution drift in our methods, we generate, select, and augment fluid data based on expert knowledge and heuristics, similar to those used in AutoAugment or ImageNet. For example, we ensure that the data's velocity distribution is relatively uniform. This way, we were able to reduce potential biases and to generate overall better results, but the approach is far from optimal. The manual data selection process is very time consuming and requires many iterations of regenerating whole data sets. This could be improved by more sophisticated methods like Goel et al. [Goe+20], which build on learned data augmentation, while methods such as JTT [Liu+21] and BT-Adapt [Sch+20] resample the data set dynamically. Other methods learn an unsupervised representation of the data approximating the real prior [Hig+16]. We refer to the paper by Wiles et al. [Wil+21] for a detailed assessment. Apart from a data-centered perspective, we can think of other options, like reducing the model's degrees of freedom using cleverly chosen inductive biases. This can reduce the problem's dimensionality and increase the model's efficiency by filtering unimportant parts. In our second work, we thus achieve increased generalization by applying the problem to local patches. We thereby eliminate information about the global context, which plays little role in our geometric-based approach, and significantly reduce the complexity and variance of the data. As a result, our method generalizes much better. Based on that, it would be interesting to test more advanced inductive biases in future work. For example, continuous convolution is used in Ummenhofer et al. [Umm+20], which includes an additional spatial bias, unlike our PointNet++ solution. In methods like Wang et al. [WWY20], Villar et al. [Vil+21], Prantl et al. [Pra+22a] and Sattoras et al. [SHW21], on the other hand, symmetry properties of the data are exploited.

Safety and Robustness In contrast to generalizability, robustness and safety represent the model's resilience to outliers and noisy data. The study of these properties addresses what happens when the method is pushed to its limits [Pas+18]. A significant drawback of current deep learning methods is the lack of understanding of what is being learned, and the uncertainty of the model is not easy to evaluate. This plays a significant role when a guarantee for the reliability of the results of the method is needed. This limitation is currently an obstacle to applying deep learning methods in engineering [SD20], for autonomous vehicle control [MDK19] or other safety-critical areas like

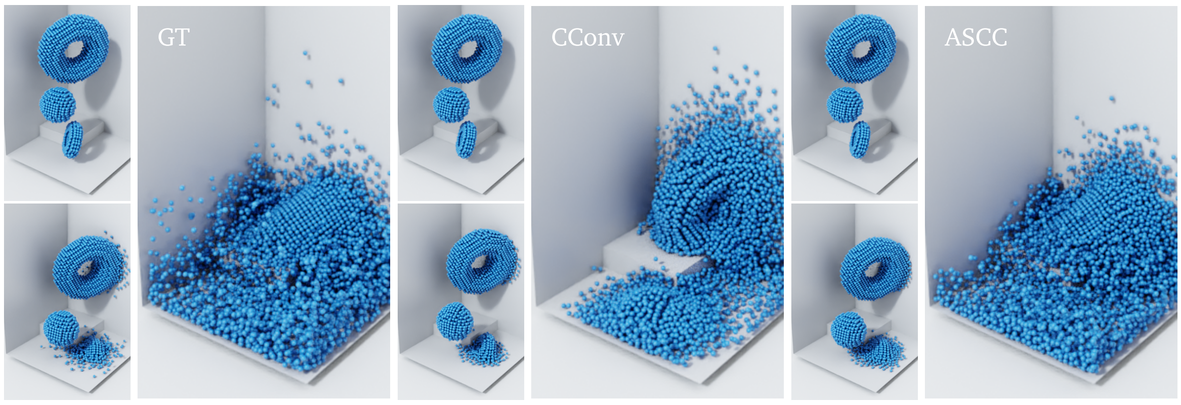


Figure 5.1.: The figure was taken from Prantl et al. [Pra+22a] and shows a comparison of two deep learning-based fluid solvers. The method based on *CConvs* visibly dampens gravitational acceleration, unlike the one with *ASCCs*, which is much closer to ground truth (*GT*) due to the correct conservation of momentum.

healthcare [AP21; CWG20]. The prerequisite is mostly safety and robustness with a provable guarantee. Current research shows that many state-of-the-art methods can be easily deceived [Pap+16; PMG16]. Therefore, the trend is currently moving more and more toward robust algorithms. In physics-based simulations, especially in engineering, the guaranteed adherence to error tolerances and unconditional stability plays a significant role. The general impression is that neural networks are difficult to control due to their complexity. This uncertainty makes deep learning methods unattractive for many engineering applications. In our methods, the focus is more on speed and visual appearance and less on the physical accuracy of the results. This prioritization is based on their relevance in computer graphics. Nevertheless, we use inductive biases to limit the scope of action of the neural networks, reducing the potential for error. Taking the idea further, we could construct the bias in such a way that it can guarantee a certain behavior. This was shown, for example, in the paper Prantl et al. [Pra+22a], which shows that we can preserve linear momentum with deep learning by exploiting the symmetrical properties of physics. The presented method is based on continuous convolutions (CConv) [Umm+20] with special anti-symmetric layers (ASCC). Figure 5.1 shows the positive effect of conservation of momentum on the prediction of SPH simulations. Bayesian Neural Networks (BNNs) [LV01; KG17] follow a different concept. Unlike conventional NNs, trained BNNs do not produce a parameter point estimate but instead learn a distribution over the parameter space. The mean value corresponds to the forecast, while the variance quantifies the certainty of the prediction. The provided certainty can deliver much better insight into how trustworthy the result from the network is. BNNs have recently been used for the regression of physical problems [Lin+22; LMR19]. However, the provided uncertainty is only an approximation and can be erroneous.

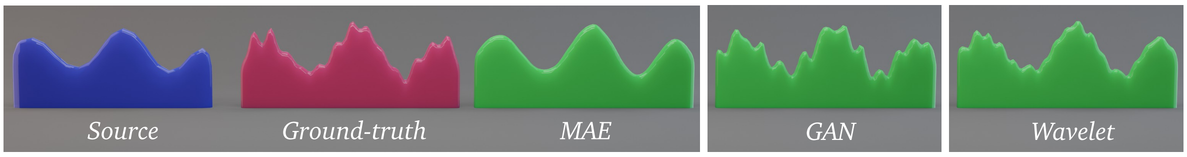


Figure 5.2.: The figure was taken from Prantl et al. [Pra+22b] and compares different loss functions for a multi-modal generative setup. The high-frequency part of the *ground truth* is randomly chosen and thus not reconstructible from the *source*. *MAE* generates very smooth results, while more advanced methods like *GAN* and a frequency-based *wavelet* loss successfully reconstruct the high-frequency features.

Similarity Metrics for Fluids Another challenge we encountered in our work was a proper quantitative evaluation of fluid simulations. It is challenging to validate spatio-temporal data based on its *fluid-likeness* and compare it to a reference distribution. For the most part, standard distance metrics, such as mean squared error (MSE) or similar, ignore critical physical properties. Physics-based loss values are difficult to interpret due to the nonlinearity and complexity of fluid behavior. Finally, there is also the problem that minor errors add up for long sequences, often making a direct comparison with a reference difficult due to the chaotic nature of fluids. Common practice is to use perceptual methods [Sal+16; Heu+17; KUT20]. One proposed tool is the generative adversarial network (GAN) [Goo+14]. GANs learn to generate new data with the same distribution as the training set. This means that the model is not trained to minimize the distance to a specific data point but rather to fit the generated results to the training distribution. The range of applicability of GANs is vast, especially for inverse generative tasks like super-resolution [Xie+18; Chu+18b]. In practice, however, they are challenging to use and interpret [Sri+17]. Mode collapse and the difficulty of reliably evaluating the performance of GANs make it difficult to properly exploit their potential. Another significant advancement regarding generative tasks was achieved by diffusion models [Soh+15; HJA20; DN21]. They are based on a Markov chain of diffusion steps which slowly add random noise to the input data and learn to revert the diffusion process. As a result, the final model can synthesize high-quality images based on input noise, even outperforming GANs. GANs and diffusion models are essential when solving multi-modal problems. However, our particular case, the generation of liquids, differs partly from perceptual tasks, such as the up-sampling of images. While in the case of images information is sometimes completely missing and has to be invented, liquids are subject to physical rules that strongly limit the solution space. The question is whether one can achieve similar performance as with GANs, with a more straightforward, deterministic loss function. The paper Prantl et al. [Pra+22b] tackles that problem. The proposed loss function is based on wavelets. The idea is to avoid the low-pass filter behavior of conventional distance functions like mean absolute error (MAE) or MSE by considering the different frequency ranges separately. The low-pass filter effect is illustrated in Figure 5.2. A MAE loss, as we have also used in this thesis, generates results that are not part of the ground-truth distribution. GANs and the presented wavelet-based loss, on the other hand, are able to reconstruct most of the high-frequency behavior. Further evaluations in the paper show that advanced loss functions can keep up with GANs. Another possibility for a more analytical approach is to formulate an optimal transport problem. We can quantify the difference between two fluid sequences by the strength of the correction parameters necessary to transform one sequence into the other. One possibility would be to use optical flow [Thu16], or the earth-movers distance (EMD), similar to what we have already done in our methods. A more advanced approach would be to use differentiable physics as in Um et al. [Um+20]. The advantage over conventional distance functions is the inclusion of volumetric and geometric properties, in addition to the underlying physics.

5.2. Conclusion

From the discussion in the outlook, we can see that combining deep learning with classical numerical methods is crucial for success. Using domain knowledge in the form of physically inductive biases improves generalization. Additionally, it gives better insight and control over the behavior of the neural network. Improving the transparency of the learning task counteracts the uncertainty inherent in the myth that associates neural networks with *black boxes*. Based on that, neural networks are a system that offers no insight and whose behavior cannot be understood. This is a controversial concern, as it precludes deep learning from being used in areas where deterministic, accurate behavior is prioritized. However, we believe deep learning as a technique is just not yet sufficiently explored. As shown in the outlook section, we can see that there are still many unanswered questions to be addressed, promising an exciting future for research in deep learning for liquids. In our opinion, future work will resolve the uncertainties, and deep learning methods for liquids will become more established accordingly.

Bibliography

[AHA12] S. Adami, X. Y. Hu, and N. A. Adams. “A generalized wall boundary condition for smoothed particle hydrodynamics”. In: *Journal of Computational Physics* 231.21 (Aug. 30, 2012), pp. 7057–7075. ISSN: 0021-9991. DOI: 10.1016/j.jcp.2012.05.005.

[AP21] Kyriakos D. Apostolidis and George A. Papakostas. “A Survey on Adversarial Deep Learning Robustness in Medical Image Analysis”. In: *Electronics* 10.17 (2021). ISSN: 2079-9292. DOI: 10.3390/electronics10172132.

[Bat+16] Peter Battaglia et al. “Interaction networks for learning about objects, relations and physics”. In: *Advances in Neural Information Processing Systems*. 2016, pp. 4502–4510.

[BC93] Pierre Baldi and Yves Chauvin. “Neural networks for fingerprint recognition”. In: *neural computation* 5.3 (1993), pp. 402–418.

[BK16] Jan Bender and Dan Koschier. “Divergence-free SPH for incompressible and viscous fluids”. In: *IEEE Transactions on Visualization and Computer Graphics* 23.3 (2016), pp. 1193–1206.

[BKK18] Shaojie Bai, J Zico Kolter, and Vladlen Koltun. “An empirical evaluation of generic convolutional and recurrent networks for sequence modeling”. In: *arXiv preprint arXiv:1803.01271* (2018).

[BL11] Achi Brandt and Oren E Livne. *Multigrid Techniques: 1984 Guide with Applications to Fluid Dynamics, Revised Edition*. SIAM, 2011.

[BR86] Jeremiah U Brackbill and Hans M Ruppel. "FLIP: A method for adaptively zoned, particle-in-cell calculations of fluid flows in two dimensions". In: *Journal of Computational physics* 65.2 (1986), pp. 314–343.

[Bri15] Robert Bridson. *Fluid simulation for computer graphics*. AK Peters/CRC Press, 2015.

[BT07] Markus Becker and Matthias Teschner. "Weakly compressible SPH for free surface flows". In: *Proc. of symposium on Computer animation*. SCA '07. Eurographics Association, 2007, pp. 209–217.

[BVS16] Christian Bailer, Kiran Varanasi, and Didier Stricker. "CNN-based Patch Matching for Optical Flow with Thresholded Hinge Loss". In: *arXiv: 1607.08064* (2016).

[BWW22] Johannes Brandstetter, Daniel E. Worrall, and Max Welling. "Message Passing Neural PDE Solvers". In: *International Conference on Learning Representations*. 2022.

[Cai+22] Shengze Cai et al. "Physics-informed neural networks (PINNs) for fluid mechanics: A review". In: *Acta Mechanica Sinica* (2022), pp. 1–12.

[Chu+18a] Mengyu Chu et al. "Temporally Coherent GANs for Video Super-Resolution (TecoGAN)". In: *CoRR* abs/1811.09393 (2018).

[Chu+18b] Mengyu Chu et al. "Temporally coherent gans for video super-resolution (tecogan)". In: *arXiv preprint arXiv:1811.09393* 1.2 (2018), p. 3.

[CM11] Nuttapong Chentanez and Matthias Müller. "A multigrid fluid pressure solver handling separating solid boundary conditions". In: *Proceedings of the 2011 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. 2011, pp. 83–90.

[Cor+14] Jens Cornelis et al. "IISPH-FLIP for incompressible fluids". In: *Computer Graphics Forum*. Vol. 33. 2. Wiley Online Library. 2014, pp. 255–262.

[CT17] Mengyu Chu and Nils Thuerey. "Data-driven synthesis of smoke flows with CNN-based feature descriptors". In: *ACM Transactions on Graphics* 36.4 (July 20, 2017), 69:1–69:14. ISSN: 0730-0301. DOI: `10.1145/3072959.3073643`.

[CWG20] Daniel C. Castro, Ian Walker, and Ben Glocker. "Causality matters in medical imaging". In: *Nature Communications* 11.1 (July 2020). DOI: `10.1038/s41467-020-17478-w`.

[DLF12] Tyler De Witt, Christian Lessig, and Eugene Fiume. "Fluid simulation using laplacian eigenfunctions". In: *ACM Transactions on Graphics (TOG)* 31.1 (2012), pp. 1–11.

[DN21] Prafulla Dhariwal and Alexander Nichol. "Diffusion models beat gans on image synthesis". In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 8780–8794.

[Dos+15] Alexey Dosovitskiy et al. “Flownet: Learning optical flow with convolutional networks”. In: *International Conference on Computer Vision (ICCV)*. IEEE. 2015, pp. 2758–2766.

[Dun89] George H Dunteman. *Principal components analysis*. 69. Sage, 1989.

[EUT19] Marie-Lena Eckert, Kiwon Um, and Nils Thuerey. “ScalarFlow: A Large-Scale Volumetric Data Set of Real-World Scalar Transport Flows for Computer Animation and Machine Learning”. In: *ACM Trans. Graph.* 38.6 (Nov. 2019). ISSN: 0730-0301. DOI: 10.1145/3355089.3356545.

[Fer+16] Florian Ferstl et al. “Narrow band FLIP for liquid simulations”. In: *Computer Graphics Forum*. Vol. 35. 2. Wiley Online Library. 2016, pp. 225–232.

[Fey+18] Matthias Fey et al. “Splinecnn: Fast geometric deep learning with continuous b-spline kernels”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 869–877.

[FM96] Nick Foster and Dimitri Metaxas. “Realistic Animation of Liquids”. In: *Graphical Models and Image Processing* 58.5 (1996), pp. 471–483. ISSN: 1077-3169. DOI: https://doi.org/10.1006/gmip.1996.0039.

[Fuk79] Kunihiko Fukushima. “Neural network model for a mechanism of pattern recognition unaffected by shift in position-Neocognitron”. In: *IEICE Technical Report, A* 62.10 (1979), pp. 658–665.

[GGY18] Weifeng Ge, Bingchen Gong, and Yizhou Yu. “Image Super-resolution via Deterministic-stochastic Synthesis and Local Statistical Rectification”. In: *ACM Trans. Graph.* 37.6 (Dec. 2018), 260:1–260:14. ISSN: 0730-0301. DOI: 10.1145/3272127.3275060.

[Gis08] Galen R Gisler. “Tsunami simulations”. In: *Annu. Rev. Fluid Mech.* 40 (2008), pp. 71–90.

[GM77] Robert A Gingold and Joseph J Monaghan. “Smoothed Particle Hydrodynamics: Theory and Application to Non-Spherical Stars”. In: *Monthly Notices of the Royal Astronomical Society* 181.3 (1977).

[Goe+20] Karan Goel et al. “Model patching: Closing the subgroup performance gap with data augmentation”. In: *arXiv preprint arXiv:2008.06775* (2020).

[Goo+14] Ian Goodfellow et al. “Generative Adversarial Nets”. In: *Advances in Neural Information Processing Systems*. Vol. 27. Curran Associates, Inc., 2014.

[Gra12] Alex Graves. “Long short-term memory”. In: *Supervised sequence labelling with recurrent neural networks* (2012), pp. 37–45.

[Gue+18] Paul Guerrero et al. “PCPNet Learning Local Shape Properties from Raw Point Clouds”. In: *Computer Graphics Forum* 37.2 (2018), pp. 75–85. ISSN: 1467-8659. DOI: 10.1111/cgf.13343.

[Her+18] Pedro Hermosilla et al. “Monte Carlo convolution for learning on non-uniformly sampled point clouds”. In: *ACM Transactions on Graphics* 37.6 (Dec. 4, 2018), 235:1–235:12. ISSN: 0730-0301. DOI: 10.1145/3272127.3275110.

[HER55] F.H. Harlow, M. Evans, and R.D. Richtmyer. *A Machine Calculation Method for Hydrodynamic Problems*. LAMS (Los Alamos Scientific Laboratory). Los Alamos Scientific Laboratory of the University of California, 1955.

[Heu+17] Martin Heusel et al. “Gans trained by a two time-scale update rule converge to a local nash equilibrium”. In: *Advances in neural information processing systems* 30 (2017).

[Hew+20] Pradeep Hewage et al. “Temporal convolutional neural (TCN) network for an effective weather forecasting using time-series data from the local weather station”. In: *Soft Computing* 24.21 (2020), pp. 16453–16482.

[Hig+16] Irina Higgins et al. “beta-vae: Learning basic visual concepts with a constrained variational framework”. In: (2016).

[HJA20] Jonathan Ho, Ajay Jain, and Pieter Abbeel. “Denoising diffusion probabilistic models”. In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 6840–6851.

[HKT20] Philipp Holl, Vladlen Koltun, and Nils Thuerey. “Learning to control pdes with differentiable physics”. In: *arXiv preprint arXiv:2001.07457* (2020).

[HOT06] Geoffrey E Hinton, Simon Osindero, and Yee-Whye Teh. “A fast learning algorithm for deep belief nets”. In: *Neural computation* 18.7 (2006), pp. 1527–1554.

[HS06] Geoffrey E Hinton and Ruslan R Salakhutdinov. “Reducing the dimensionality of data with neural networks”. In: *science* 313.5786 (2006), pp. 504–507.

[HTY18] Binh-Son Hua, Minh-Khoi Tran, and Sai-Kit Yeung. “Pointwise Convolutional Neural Networks”. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2018, pp. 984–993.

[Hu+19] Yuanming Hu et al. “Difftaichi: Differentiable programming for physical simulation”. In: *arXiv preprint arXiv:1910.00935* (2019).

[HW65] Francis Harlow and Eddie Welch. “Numerical Calculation of Time-dependent Viscous Incompressible Flow of Fluid with Free Surface”. In: *Physics of Fluids* 8.12 (1965), pp. 2182–2189.

[Ilg+17] Eddy Ilg et al. “FlowNet 2.0: Evolution of Optical Flow Estimation With Deep Networks”. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2017, pp. 2462–2470.

[Jad+15] Max Jaderberg et al. “Spatial Transformer Networks”. In: *Advances in Neural Information Processing Systems*. Vol. 28. Curran Associates, Inc., 2015.

[Jia+15] Chenfanfu Jiang et al. “The Affine Particle-In-Cell Method”. In: *TOG* 34.4 (July 2015), 51:1–51:10.

[KG17] Alex Kendall and Yarin Gal. “What uncertainties do we need in bayesian deep learning for computer vision?” In: *Advances in neural information processing systems* 30 (2017).

[Kim+08] Theodore Kim et al. “Wavelet Turbulence for Fluid Simulation”. In: *TOG* 27 (3) (2008), 50:1–6.

[Koc+21] Dmitrii Kochkov et al. “Machine learning–accelerated computational fluid dynamics”. In: *Proceedings of the National Academy of Sciences* 118.21 (2021).

[Kos+22] Dan Koschier et al. “A Survey on SPH Methods in Computer Graphics”. In: *Computer Graphics Forum* 41.2 (2022), pp. 737–760. ISSN: 1467-8659. DOI: 10.1111/cgf.14508.

[KUT20] Georg Kohl, Kiwon Um, and Nils Thuerey. “Learning Similarity Metrics for Numerical Simulations”. In: *Proceedings of the 37th International Conference on Machine Learning*. Ed. by Hal Daumé III and Aarti Singh. Vol. 119. Proceedings of Machine Learning Research. PMLR, July 2020, pp. 5349–5360.

[Lad+15] L’ubor Ladický et al. “Data-driven fluid simulations using regression forests”. In: *ACM Transactions on Graphics* 34.6 (Oct. 26, 2015), 199:1–199:9. ISSN: 0730-0301. DOI: 10.1145/2816795.2818129.

[Lea+16] Colin Lea et al. “Temporal convolutional networks: A unified approach to action segmentation”. In: *European conference on computer vision*. Springer. 2016, pp. 47–54.

[Led+16] Christian Ledig et al. “Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network.” In: *CoRR* abs/1609.04802 (2016).

[Li+19] Yunzhu Li et al. “Learning Particle Dynamics for Manipulating Rigid Bodies, Deformable Objects, and Fluids”. In: *ICLR*. 2019.

[Lin+22] Kevin Linka et al. “Bayesian Physics Informed Neural Networks for real-world nonlinear dynamical systems”. In: *Computer Methods in Applied Mechanics and Engineering* (2022), p. 115346. ISSN: 0045-7825. DOI: https://doi.org/10.1016/j.cma.2022.115346.

[Liu+21] Evan Z Liu et al. “Just train twice: Improving group robustness without training group information”. In: *International Conference on Machine Learning*. PMLR. 2021, pp. 6781–6792.

[LMR19] Tyler LaBonte, Carianne Martinez, and Scott A Roberts. “We know where we don’t know: 3d bayesian cnns for credible geometric uncertainty”. In: *arXiv preprint arXiv:1910.10793* (2019).

[LV01] Jouko Lampinen and Aki Vehtari. “Bayesian approach for neural networks—review and case studies”. In: *Neural networks* 14.3 (2001), pp. 257–274.

[MDK19] Xiaobai Ma, Katherine Rose Driggs-Campbell, and Mykel J. Kochenderfer. “Improved Robustness and Safety for Autonomous Vehicle Control with Adversarial Reinforcement Learning”. In: *CoRR* abs/1903.03642 (2019). arXiv: 1903.03642.

[MHR18] Simon Meister, Junhwa Hur, and Stefan Roth. “UnFlow: Unsupervised Learning of Optical Flow With a Bidirectional Census Loss”. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 32.1 (Apr. 27, 2018). Number: 1. ISSN: 2374-3468. DOI: 10.1609/aaai.v32i1.12276.

[MM13] Miles Macklin and Matthias Müller. “Position based fluids”. In: *TOG* 32.4 (2013), p. 104.

[MST10] Aleka McAdams, Eftychios Sifakis, and Joseph Teran. “A Parallel Multigrid Poisson Solver for Fluids Simulation on Large Grids.” In: *Symposium on Computer Animation*. 2010, pp. 65–73.

[NB16] Michael B Nielsen and Robert Bridson. “Spatially adaptive FLIP fluid simulations in bifrost”. In: *ACM SIGGRAPH 2016 Talks*. 2016, pp. 1–2.

[Nie15] Michael A Nielsen. *Neural networks and deep learning*. Vol. 25. Determination press San Francisco, CA, USA, 2015.

[Pap+16] Nicolas Papernot et al. “The limitations of deep learning in adversarial settings”. In: *2016 IEEE European symposium on security and privacy (EuroS&P)*. IEEE. 2016, pp. 372–387.

[Pas+18] Magdalini Paschali et al. “Generalizability vs. Robustness: Adversarial Examples for Medical Imaging”. In: *CoRR* abs/1804.00504 (2018). arXiv: 1804.00504.

[PBT19] Lukas Prantl, Boris Bonev, and Nils Thuerey. “Generating Liquid Simulations with Deformation-aware Neural Networks”. In: International Conference on Learning Representations. Feb. 20, 2019.

[Pér+18] E. Pérez-Pellitero et al. “Photorealistic Video Super Resolution”. In: *Workshop PIRM at ECCV* (2018).

[Pfa+20] Tobias Pfaff et al. “Learning mesh-based simulation with graph networks”. In: *arXiv preprint arXiv:2010.03409* (2020).

[PMG16] Nicolas Papernot, Patrick McDaniel, and Ian Goodfellow. “Transferability in machine learning: from phenomena to black-box attacks using adversarial samples”. In: *arXiv preprint arXiv:1605.07277* (2016).

[Pop00] S. B. Pope. *Turbulent Flows*. Cambridge University Press, 2000.

[Pra+20] Lukas Prantl et al. “Tranquil Clouds: Neural Networks for Learning Temporally Coherent Features in Point Clouds”. In: International Conference on Learning Representations. Mar. 11, 2020.

[Pra+22a] Lukas Prantl et al. “Guaranteed Conservation of Momentum for Learning Particle-based Fluid Dynamics”. In: (May 26, 2022).

[Pra+22b] Lukas Prantl et al. *Wavelet-based Loss for High-frequency Interface Dynamics*. 2022. DOI: 10.48550/ARXIV.2209.02316.

[Qi+17a] C.R. Qi et al. “PointNet: Deep learning on point sets for 3D classification and segmentation”. In: Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017. Vol. 2017-January. 2017, pp. 77–85. ISBN: 978-1-5386-0457-1. DOI: 10.1109/CVPR.2017.16.

[Qi+17b] Charles Ruizhongtai Qi et al. “Pointnet++: Deep hierarchical feature learning on point sets in a metric space”. In: *Advances in neural information processing systems* 30 (2017).

[Rav+14] Karthik Raveendran et al. “Blending liquids”. In: *ACM Transactions on Graphics* 33.4 (July 27, 2014), 137:1–137:10. ISSN: 0730-0301. DOI: 10.1145/2601097.2601126.

[RB17] Anurag Ranjan and Michael J. Black. “Optical Flow Estimation Using a Spatial Pyramid Network”. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2017, pp. 4161–4170.

[RHW86a] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. “Learning representations by back-propagating errors”. In: *nature* 323.6088 (1986), pp. 533–536.

[RHW86b] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. “Learning representations by back-propagating errors”. In: *Nature* 323.6088 (Oct. 1, 1986), pp. 533–536. ISSN: 1476-4687. DOI: 10.1038/323533a0.

[Sal+16] Tim Salimans et al. “Improved techniques for training gans”. In: *Advances in neural information processing systems* 29 (2016).

[San+20] Alvaro Sanchez-Gonzalez et al. “Learning to simulate complex physics with graph networks”. In: *ICML*. 2020.

[Sat+18] Takahiro Sato et al. “Extended narrow band FLIP for liquid simulations”. In: *Computer Graphics Forum*. Vol. 37. 2. Wiley Online Library. 2018, pp. 169–177.

[SC91] Andrew Staniforth and Jean Côté. “Semi-Lagrangian integration schemes for atmospheric models—A review”. In: *Monthly weather review* 119.9 (1991), pp. 2206–2223.

[Sch+20] Steffen Schneider et al. “Improving robustness against common corruptions by covariate shift adaptation”. In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 11539–11551.

[Sch93] Jürgen Schmidhuber. “Habilitation thesis: System modeling and optimization”. In: *Page 150 ff demonstrates credit assignment across the equivalent of 1,200 layers in an unfolded RNN* (1993).

[SD20] Matteo Sangiorgio and Fabio Dercole. “Robustness of LSTM neural networks for multi-step forecasting of chaotic time series”. In: *Chaos, Solitons & Fractals* 139 (2020), p. 110045.

[SF18] Connor Schenck and Dieter Fox. “SPNets: Differentiable Fluid Dynamics for Deep Neural Networks”. In: *Conference on Robot Learning*. 2018.

[SHW21] Victor Garcia Satorras, Emiel Hoogeboom, and Max Welling. “E(n) Equivariant Graph Neural Networks”. In: *Proceedings of the 38th International Conference on Machine Learning*. International Conference on Machine Learning. ISSN: 2640-3498. PMLR, July 1, 2021, pp. 9323–9332.

[Soh+15] Jascha Sohl-Dickstein et al. “Deep unsupervised learning using nonequilibrium thermodynamics”. In: *International Conference on Machine Learning*. PMLR. 2015, pp. 2256–2265.

[SP09] Barbara Solenthaler and Renato Pajarola. “Predictive-corrective incompressible SPH”. In: *TOG* 28.3 (2009), p. 40.

[Sri+17] Akash Srivastava et al. “Veegan: Reducing mode collapse in gans using implicit variational learning”. In: *Advances in neural information processing systems* 30 (2017).

[SSH17] M. S. M. Sajjadi, B. Schölkopf, and M. Hirsch. "EnhanceNet: Single Image Super-Resolution Through Automated Texture Synthesis". In: *IEEE International Conference on Computer Vision (ICCV)*. Oct. 2017, pp. 4501–4510. DOI: 10.1109/ICCV.2017.481.

[Sta99] Jos Stam. "Stable fluids". In: *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*. 1999, pp. 121–128.

[SVB18] Mehdi S. M. Sajjadi, Raviteja Vemulapalli, and Matthew Brown. "Frame-Recurrent Video Super-Resolution". In: *CVPR*. 2018.

[Tez+96] T Tezduyar et al. "Flow simulation and high performance computing". In: *Computational Mechanics* 18.6 (1996), pp. 397–412.

[Thu16] Nils Thuerey. "Interpolations of Smoke and Liquid Simulations". In: *ACM Transactions on Graphics* 36.1 (Sept. 9, 2016), 3:1–3:16. ISSN: 0730-0301. DOI: 10.1145/2956233.

[TLP06] Adrien Treuille, Andrew Lewis, and Zoran Popović. "Model reduction for real-time fluids". In: *ACM Transactions on Graphics (TOG)* 25.3 (2006), pp. 826–834.

[Tom+17] Jonathan Tompson et al. "Accelerating Eulerian Fluid Simulation With Convolutional Networks". In: *Proceedings of the 34th International Conference on Machine Learning*. International Conference on Machine Learning. ISSN: 2640-3498. PMLR, July 17, 2017, pp. 3424–3433.

[UHT17] Kiwon Um, Xiangyu Hu, and Nils Thuerey. "Perceptual Evaluation of Liquid Simulation Methods". In: *TOG* 36(4).143 (2017).

[Um+20] Kiwon Um et al. "Solver-in-the-Loop: Learning from Differentiable Physics to Interact with Iterative PDE-Solvers". In: *Advances in Neural Information Processing Systems* (2020).

[Umm+20] Benjamin Ummenhofer et al. "Lagrangian Fluid Simulation with Continuous Convolutions". In: International Conference on Learning Representations. Mar. 11, 2020.

[Vas+17] Ashish Vaswani et al. "Attention is all you need". In: *Advances in neural information processing systems* 30 (2017).

[Vil+21] Soledad Villar et al. "Scalars are universal: Equivariant machine learning, structured like classical physics". In: *Advances in Neural Information Processing Systems*. Vol. 34. Curran Associates, Inc., 2021, pp. 28848–28863.

[Vre94] Cornelis Boudewijn Vreugdenhil. *Numerical methods for shallow-water flow*. Vol. 13. Springer Science & Business Media, 1994.

[Wan+18] Shenlong Wang et al. “Deep parametric continuous convolutional neural networks”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 2589–2597.

[Wan+19] Renzhuo Wan et al. “Multivariate temporal convolutional network: A deep neural networks approach for multivariate time series forecasting”. In: *Electronics* 8.8 (2019), p. 876.

[Wan+20] Rui Wang et al. “Towards Physics-informed Deep Learning for Turbulent Flow Prediction”. In: *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. KDD ’20. New York, NY, USA: Association for Computing Machinery, Aug. 23, 2020, pp. 1457–1466. ISBN: 978-1-4503-7998-4. DOI: 10.1145/3394486.3403198.

[WBT19] Steffen Wiewel, Moritz Becher, and Nils Thuerey. “Latent space physics: Towards learning the temporal evolution of fluid flow”. In: *Computer graphics forum*. Vol. 38. 2. Wiley Online Library. 2019, pp. 71–82.

[Wil+21] Olivia Wiles et al. “A fine-grained analysis on distribution shift”. In: *arXiv preprint arXiv:2110.11328* (2021).

[Wil06] David C Wilcox. *Turbulence Modeling for CFD*. 3rd. DCW industries, 2006.

[Woj+17] Chris Wojtan et al. “Large-scale interactive water simulation with directional waves”. In: *ACM SIGGRAPH 2017 Real Time Live!* 2017, pp. 19–19.

[Wu+18] Kui Wu et al. “Fast Fluid Simulations with Sparse Volumes on the GPU”. In: *Computer Graphics Forum (Proceedings of EUROGRAPHICS 2018)* 37.2 (2018), pp. 157–167. DOI: 10.1111/cgf.13350.

[WWY20] Rui Wang, Robin Walters, and Rose Yu. “Incorporating symmetry into deep dynamics models for improved generalization”. In: *arXiv preprint arXiv:2002.03061* (2020).

[Xie+18] You Xie et al. “tempoGAN: a temporally coherent, volumetric GAN for super-resolution fluid flow”. In: *ACM Transactions on Graphics* 37.4 (July 30, 2018), 95:1–95:15. ISSN: 0730-0301. DOI: 10.1145/3197517.3201304.

[Yin+18] Kangxue Yin et al. “P2P-NET: bidirectional point displacement net for shape transform”. In: *ACM Transactions on Graphics* 37.4 (July 30, 2018), 152:1–152:13. ISSN: 0730-0301. DOI: 10.1145/3197517.3201288.

[Yu+18] Lequan Yu et al. “PU-Net: Point Cloud Upsampling Network”. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2018, pp. 2790–2799.

[YYX16] Cheng Yang, Xubo Yang, and Xiangyun Xiao. “Data-driven projection method in fluid simulation”. In: *Computer Animation and Virtual Worlds* 27.3-4 (2016), pp. 415–424.

[Zha+21] Hengshuang Zhao et al. “Point transformer”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2021, pp. 16259–16268.

GENERATING LIQUID SIMULATIONS WITH DEFORMATION-AWARE NEURAL NETWORKS

Lukas Prantl, Boris Bonev & Nils Thuerey
Department of Computer Science
Technical University of Munich
Boltzmannstr. 3, 85748 Garching, Germany
{lukas.prantl,boris.bonev,nils.thuerey}@tum.de

ABSTRACT

We propose a novel approach for deformation-aware neural networks that learn the weighting and synthesis of dense volumetric deformation fields. Our method specifically targets the space-time representation of physical surfaces from liquid simulations. Liquids exhibit highly complex, non-linear behavior under changing simulation conditions such as different initial conditions. Our algorithm captures these complex phenomena in two stages: a first neural network computes a weighting function for a set of pre-computed deformations, while a second network directly generates a deformation field for refining the surface. Key for successful training runs in this setting is a suitable loss function that encodes the effect of the deformations, and a robust calculation of the corresponding gradients. To demonstrate the effectiveness of our approach, we showcase our method with several complex examples of flowing liquids with topology changes. Our representation makes it possible to rapidly generate the desired implicit surfaces. We have implemented a mobile application to demonstrate that real-time interactions with complex liquid effects are possible with our approach.

1 INTRODUCTION

Learning physical functions is an area of growing interest within the research community, with applications ranging from physical priors for computer vision problems Kyriazis & Argyros (2013), over robotic control Schenck & Fox (2017), to fast approximations for numerical solvers Tompson et al. (2017). While the underlying model equations for many physics problems are known, finding solutions is often prohibitively expensive for phenomena on human scales. At the same time, the availability of model equations allows for the creation of reliable ground truth data for training, if enough computational resources can be allocated.

Water, and liquids in general, are ubiquitous in our world. At the same time, they represent an especially tough class of physics problems, as the constantly changing boundary conditions at the liquid-gas interface result in a complex space of surface motions and configurations. In this work we present a novel approach to capture parametrized spaces of liquid behavior that is based on space-time deformations. We represent a single 3D input surface over time as a four-dimensional signed-distance function (SDF), which we deform in both space and time with learned deformations to recover the desired physical behavior. To calculate and represent these deformations efficiently, we take a two-stage approach: First, we span the sides of the original parameter region with pre-computed deformations, and infer a suitable weighting function. In a second step, we synthesize a dense deformation field for refinement. As both the parameter weighting problem and the deformation synthesis are highly non-linear problems, we demonstrate that neural networks are a particularly suitable solver to robustly find solutions.

We will demonstrate that it is possible to incorporate the non-linear effects of weighted deformations into the loss functions of neural networks. In particular, we put emphasis on incorporating the influence of deformation alignment into the loss gradients. This alignment step is necessary to ensure the correct application of multiple consecutive deformations fields. The second stage of our algorithm is a generative model for deformation fields, for which we rely on a known parametrization of the inputs. Thus, in contrast to other generative models which learn to represent unknown parametrization of data sets Radford et al. (2016), our models are trained with a known range and dimensionality to parameter range, which serves as input.

Once trained, the models can be evaluated very efficiently to synthesize new implicit surface configurations. To demonstrate its performance, we have implemented a proof-of-concept version for mobile devices, and a demo app is available for Android devices in the *Google Play store*. Our approach generates liquid animations several orders of magnitude faster than a traditional simulator, and achieves effective speed up factors of more than 2000, as we will outline in Sec. 5. The central contributions of our work are:

- A novel *deformation-aware neural network approach* to very efficiently represent large collections of space-time surfaces with complex behavior.
- We show how to compute suitable loss gradient approximations for the sub-problems of parameter and deformation inference.
- In addition we showcase the high performance of our approach with a mobile device implementation that generates liquid simulations interactively.

2 Related Work

Capturing physical behavior with learning has a long history in the field of learning. Early examples targeted minimization problems to determine physical trajectories or contact forces Bhat et al. (2002); Brubaker et al. (2009), or plausible physics for interacting objects Kyriazis & Argyros (2013; 2014). Since initial experiments with physically-based animation and neural networks Grzeszczuk et al. (1998), a variety of new deep learning based works have been proposed to learn physical models from videos Battaglia et al. (2016); Chang et al. (2016); Watters et al. (2017). Others have targeted this goal in specific settings such as robotic interactions Finn et al. (2016), sliding and colliding objects Wu et al. (2015; 2016), billiard Fragkiadaki et al. (2015), or trajectories in height fields Ehrhardt et al. (2017). The prediction of forces to infer image-space motions has likewise been targeted Mottaghi et al. (2016a;b), while other researchers demonstrated that the stability of stacked objects can be determined from single images Lerer et al. (2016); Li et al. (2016). In addition, the unsupervised inference of physical functions poses interesting problems Stewart & Ermon (2017). While these methods have achieved impressive results, an important difference to our method is that we omit the projection to images. I.e., we directly work with three-dimensional data sets over time.

In the context of robotic control, physics play a particularly important role, and learning object properties from poking motions Agrawal et al. (2016), or interactions with liquids Schenck & Fox (2017) were targeted in previous work. Learning physical principles was also demonstrated for automated experiments with reinforcement learning Denil et al. (2016). Recently, first works have also addressed replacing numerical solvers with trained models for more generic PDEs Farimani et al. (2017); Long et al. (2017). In our work we target a more narrow case: that of surfaces deforming based on physical principles. However, thanks to this narrow scope and our specialized loss functions we can generate very complex physical effects in 3D plus time.

Our method can be seen as a generative approach representing samples from a chaotic process. In this context, the learned latent-space representation of regular generative models Masci et al. (2011); Rezende et al. (2014); Goodfellow et al. (2014); Radford et al. (2016); Isola et al. (2017) is replaced by the chosen parametrization of a numerical solver. Our model shares the goal to learn flow physics based on examples with other methods Ladicky et al. (2015); Tompson et al. (2017); Chu & Thuerey (2017), but in contrast to these we focus on 4D volumes of physics data, instead of localized windows at single instances of time. Alternative methods have been proposed to work with 4D simulation data Raveendran et al. (2014); Thuerey (2017), however, without being able to capture larger spaces of physical behavior. Due to its focus on deformations, our work also shares similarities with methods for optical flow inference and image correspondences Bailer et al. (2016); Dosovitskiy et al. (2015); Ranjan & Black (2016); Ilg et al. (2016). A difference to these approaches is that we learn deformations and their weighting in an unsupervised manner, without explicit ground truth data. Thus, our method shares similarities with spatial transformer networks (STNs) Jaderberg et al. (2015), and unsupervised approaches for optical flow Meister et al. (2017).

However, instead of aligning two data sets, our method aims for representing larger, parametrized spaces of deformations.

3 Learning Deformations

We first explain our formulation of the simulation parameter space, which replaces the latent space of other generative models such as autoencoders Masci et al. (2011), GANs Radford et al. (2016), or auto-regressive models Van Oord et al. (2016). Given a Navier-Stokes boundary value problem with a liquid-gas interface, we treat the interface over time as a single space-time surface. We work with a set of these space-time surfaces, defined over a chosen region of the N-dimensional simulation parameters $\boldsymbol{\alpha}$. We assume the parameters to be normalized, i.e. $\boldsymbol{\alpha} \in [0,1]^N$. In practice, $\boldsymbol{\alpha}$ could contain any set of parameters of the simulation, e.g. initial positions, or even physical parameters such as viscosity. We choose implicit functions $\phi(\boldsymbol{\alpha}) \in \mathrm{R}^4 \to \mathrm{R}$ to represent specific instances, such that $\Gamma(\boldsymbol{\alpha}) = \left\{\mathbf{x} \in \mathrm{R}^4; \phi(\boldsymbol{\alpha}, \mathbf{x}) = 0\right\}$ is the space of surfaces parametrized by $\boldsymbol{\alpha}$ that our generative model should capture. In the following, ϕ and ψ will denote four-dimensional signed distance functions. We will typically abbreviate $\phi(\boldsymbol{\alpha})$ with $\phi_{\boldsymbol{\alpha}}$ to indicate that $\phi_{\boldsymbol{\alpha}}$ represents a set of constant reference surfaces. While we will later on discretize all functions and operators on regular Cartesian grids, we will first show the continuous formulation in the following.

Deforming Implicit Surfaces Representing the whole set $\phi_{\boldsymbol{\alpha}}$ is very challenging. Due to bifurcations and discretization artifacts $\phi_{\boldsymbol{\alpha}}$ represents a process that exhibits noisy and chaotic behavior, an example for which can be found in Fig. 1. Despite the complex changes in the data, our goal is to find a manageable and reduced representation. Thus, we generate an approximation of $\phi_{\boldsymbol{\alpha}}$ by deforming a single initial surface in space and time. We apply space-time deformations $\mathbf{u} : \mathrm{R}^4 \to \mathrm{R}^4$ to the initial surface $\psi_0(\mathbf{x})$. As a single deformation is limited in terms of its representative power to produce different shapes, we make use of N sequential, pre-computed deformations, thus $\mathbf{u}_i$ with $i \in [1 \cdots N]$, each of which is scaled by a scalar weight parameter β_i, whose values are normally between 0 and 1. This gives us the freedom to choose how much of $\mathbf{u}_i$ to apply. The initial surface deformed by, e.g., $\mathbf{u}_1$ is given by $\psi_0(\mathbf{x} - \beta_1 \mathbf{u}_1)$.

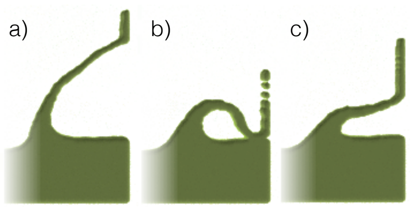


Figure 1: Three liquid surfaces after 60 time steps differing only by $\pm\epsilon$ in initial conditions. Even this initially very small difference can lead to large differences in surface position, e.g., the sheet in b) strongly curving downward.

The sequence of deformed surfaces by sequentially applying all pre-computed $\mathbf{u}_i$ is given by $\psi_i(\mathbf{x}, \boldsymbol{\beta}) = \psi_{i-1}(\mathbf{x} - \beta_i \mathbf{u}_i)$. It is crucial to align such sequences of Eulerian deformations with each other. Here we employ the alignment from previous work Thuerey (2017), which we briefly summarize in the following, as it influences the gradient calculation below. Each deformation i relies on a certain spatial configuration for the input surface from deformation $i-1$. Thus, when applying $\mathbf{u}_{i-1}$ with a weight $\beta_{i-1} < 1$, we have to align $\mathbf{u}_i$ correspondingly. Given the combined deformation $\mathbf{v}_{\text{sum}}(\mathbf{x}, \boldsymbol{\alpha}) = \sum_{i=1}^{N} \beta_i \mathbf{u}_i^*(\mathbf{x})$, with intermediate deformation fields $\mathbf{u}_{i-1}^*(\mathbf{x}) = \mathbf{u}_{i-1}(\mathbf{x} - \mathbf{u}_i^*(\mathbf{x}))$, we compute an inversely weighted offset field as $\mathbf{v}_{\text{inv}}(\mathbf{x}, \boldsymbol{\alpha}) = -\sum_{i=1}^{N}(1 - \beta_i)\mathbf{u}_i^*(\mathbf{x})$. This offset field is used to align the accumulated deformations to compute the final deformation as $\mathbf{v}_{\text{fin}}(\mathbf{x} + \mathbf{v}_{\text{inv}}(\mathbf{x}, \boldsymbol{\beta}), \boldsymbol{\beta}) = \mathbf{v}_{\text{sum}}(\mathbf{x}, \boldsymbol{\beta})$. $\mathbf{v}_{\text{fin}}$ now represents all weighted deformations $\beta_i \mathbf{u}_i$ merged into a single vector field. Intuitively, this process moves all deformation vectors to the right location for the initial surface ψ_0, such that they can be weighted and accumulated.

To achieve the goal of representing the full set of target implicit surfaces $\phi_{\boldsymbol{\alpha}}$ our goal is to compute two functions: the first one aims for an optimal weighting for each of the deformations in the sequence, i.e. $\boldsymbol{\beta}$, while the second function computes a final refinement deformation $\mathbf{w}$ after the weighted sequence has been applied. We will employ two neural networks to approximate the two functions, which we will denote as f_p, and f_d below. Both functions depend only on the simulation parameters space $\boldsymbol{\alpha}$, i.e., $f_p(\boldsymbol{\alpha}) = \boldsymbol{\beta}$, and $f_d(\boldsymbol{\alpha}) = \mathbf{w}$.

Splitting the problem into f_p and f_d is important, as each of the pre-computed deformations weighted by f_p only covers a single trajectory in the space of deformed surfaces. In the follow-

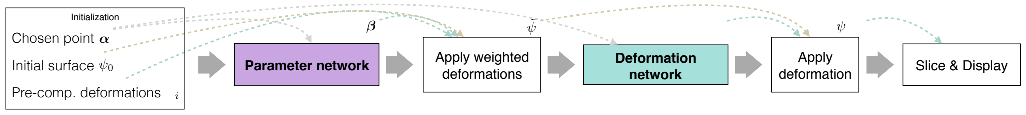


Figure 2: This illustration gives an overview of our algorithm. It works in two stages, a weighting and refinement stage, each of which employs a neural network to infer a weighting function and a dense deformation field, respectively.

ing, we employ an optical flow solve from previous work to pre-compute deformations between the inputs Thuerey (2017), which deform the input for the extrema of each original parameter dimension α_i. E.g., for a two-dimensional parameter space this yields two deformations along the sides of the unit cube. This first step robustly covers rough, large scale deformations. As a second step, we employ f_d, which is realized as a generative CNN, to infer a deformation for refining the solution. Below we will explain the resulting equations for training. The full equations for applying the deformations, and a full derivation of our loss functions can be found in the supplemental materials. To shorten the notation, we introduce the helper function $\mathcal{D}(\mathbf{x}_i, \boldsymbol{\alpha})$, which yields a deformed set of coordinates in R^4 depending on $\boldsymbol{\alpha}$ that incorporates the deformation sequence weighted by $f_p(\boldsymbol{\alpha})$, and a refinement deformation from $f_d(\boldsymbol{\alpha})$.

We express the overall goal in terms of minimizing the L_2 distance between the deformed and the target implicit surfaces for all possible values in the parameter space $\boldsymbol{\alpha}$, using $\boldsymbol{\beta}$ and $\mathbf{w}$ as degrees of freedom:

$$\underset{\boldsymbol{\beta},\mathbf{w}}{\operatorname{argmin}}\, L, L = \int \|\psi_0(\mathcal{D}(\mathbf{x}_i, \boldsymbol{\alpha})) - \phi_{\boldsymbol{\alpha}}\|_2^2 \,\mathrm{d}\alpha \,. \tag{1}$$

Our work addresses the problem of how to compute weighting of the deformations and on synthesizing the refinement field. The main difficulty lies in the non-linearity of the deformations, which is why we propose a novel method to robustly approximate both functions with NNs: f_p will be represented by the *parameter network* to compute $\boldsymbol{\beta}$, and we make use of a *deformation network* that to generate $\mathbf{w}$. We employ relatively simple neural networks for both functions. Key for training them is encoding the effect of deformations in the loss functions to make the training process aware of their influence. Hence, we will focus on describing the loss functions for both networks and the corresponding discretized gradients in the following.

Learning Deformation Weights For training the NNs we propose the following objective function, which measures the similarity of a known reference surface $\phi_{\boldsymbol{\alpha}}$ and the corresponding, approximated result $\psi_0(\mathbf{x}, \boldsymbol{\alpha})$ for a parameter value $\boldsymbol{\alpha}$. We introduce the numerical equivalent of the continuous L_2 loss from Eq. (1) as

$$L = \frac{1}{2} \sum_i \left(\psi_0(\mathcal{D}(\mathbf{x}_i, \boldsymbol{\alpha})) - \phi_{\boldsymbol{\alpha}}(\mathbf{x}_i)\right)^2 \Delta x_i \,, \tag{2}$$

which approximates the spatial integral via the sum over all sample points i with corresponding evaluation positions $\mathbf{x}_i \in \mathrm{R}^4$, where $\Delta x_i = \|\mathbf{x}_i - \mathbf{x}_{i-1}\|$ is constant in our case. This corresponds to a regular grid structure, where the loss is accumulated per cell. The central challenge here is to compute reliable gradients for $\mathcal{D}$, which encapsulates a series of highly non-linear deformation steps. We first focus on inferring $\boldsymbol{\beta}$, with $\mathbf{w} = 0$.

The gradient of Eq. (2) with respect to one component of the deformation parameters β_j is then given by

$$\frac{\mathrm{d}}{\mathrm{d}\beta_j} L = \sum_i \left(-\frac{\partial}{\partial \beta_j} \mathbf{v}_{\text{fin}}(\mathbf{x}_i + \mathbf{v}_{\text{inv}}(\mathbf{x}_i, \boldsymbol{\beta}), \boldsymbol{\beta}) \cdot \nabla \psi_0(\mathbf{x}_i - \mathbf{v}_{\text{fin}}(\mathbf{x}_i, \boldsymbol{\beta})) \right) \left(\psi_0(\mathbf{x}_i, \boldsymbol{\beta}) - \phi_{\boldsymbol{\alpha}}(\mathbf{x}_i)\right) \,, \tag{3}$$

where the first term on the sum over i in parentheses represents the gradient of the deformed initial surface. Here we compute the derivative of the full deformation as $\frac{\partial}{\partial \beta_j} \mathbf{v}_{\text{fin}}(\mathbf{x}_i + \mathbf{v}_{\text{inv}}(\mathbf{x}_i, \boldsymbol{\beta}), \boldsymbol{\beta}) = \mathbf{u}_i^*(\mathbf{x}_i)$. The offset by $\mathbf{v}_{\text{inv}}$ on the left hand side indicates that we perform a forward-advection step for this term. Details of this step, and for the full derivation of the gradient are given in Appendix B.1.

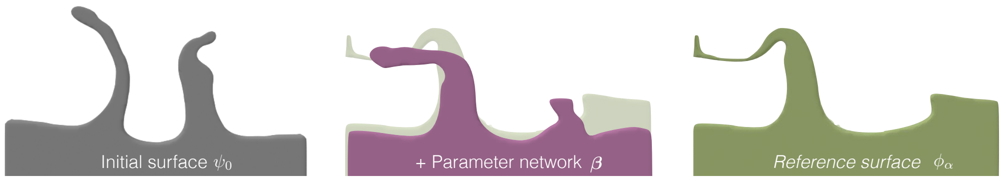


Figure 3: An example of our parameter learning approach. F.l.t.r.: the initial undeformed surface, the surface deformed by the weighting from the trained parameter network, and the reference surface only. The reference surface is shown again in the middle in light brown for comparison. The weighted deformations especially match the left liquid arm well, while there are not enough degrees of freedom in the pre-computed deformations to independently raise the surface on the right side.

A trained NN with this loss functions yields an instance of f_p, with which we can infer adjusted deformation weights $f_p(\boldsymbol{\alpha}) = \boldsymbol{\beta}$.

Learning to Generate Deformations Based on $\boldsymbol{\beta}$, we apply the deformation sequence $\mathbf{u}_i$. The goal of our second network, the deformation network f_d, is to compute the refinement deformation $\mathbf{w}$. In contrast to the pre-computed $\mathbf{u}_i$, $f_d(\boldsymbol{\alpha}) = \mathbf{w}$ now directly depends on $\boldsymbol{\alpha}$, and can thus capture the interior of the parameter space. Given the initial surface ψ_0 deformed by the set of $\beta_i \mathbf{u}_i$, which we will denote as $\tilde{\psi}$ below, the refinement deformation is applied with a final deformation step as $\psi(\mathbf{x}) = \tilde{\psi}\left(\mathbf{x} - \mathbf{w}(\mathbf{x}, \alpha)\right)$.

In order to compute the gradient of the deformation loss, we introduce the indicator function $\chi_j(\mathbf{x})$ for a single deformation vector $\mathbf{w}_j$ of $\mathbf{w}$. We found it useful to use a fine discretization for the implicit surfaces, such as ψ, and lower resolutions for $\mathbf{w}$. Hence, each discrete entry $\mathbf{w}_j$ can act on multiple cells of ψ, which we enumerate with the help of χ_j. Now the derivative of Eq. (1) for a fixed $\boldsymbol{\beta}$ with respect to a single deformation vector $\mathbf{w}_j$ of $\mathbf{w}$ is given by

$$\frac{\mathrm{d}}{\mathrm{d}\mathbf{w}_j} L = -\sum_i \chi_j(\mathbf{x}_i)\, \nabla\tilde{\psi}(\mathbf{x}_i - \mathbf{w}(\mathbf{x}_i, \boldsymbol{\alpha}))\, \left(\tilde{\psi}(\mathbf{x}_i, \boldsymbol{\alpha}) - \phi_{\boldsymbol{\alpha}}(\mathbf{x}_i)\right). \tag{4}$$

The full derivation of this gradient is given in Appendix B.2. Our approach for deformation learning can be regarded as an extension of STNs Jaderberg et al. (2015) for dense, weighted fields, and semi-Lagrangian advection methods. The parameter network corresponds to an STN which learns to combine and weight known deformation fields. The deformation network, on the other hand, resembles the thin plate spline STNs, where a network generates an offset for each cell center, which is then used to sample a deformed image or grid. Note that in our case, this sampling process corresponds to the semi-Lagrangian advection of a fluid simulation.

Training Details For f_p we use a simple structure with two fully connected layers, while f_d likewise contains two fully connected layers, followed by two or more four-dimensonal de-convolution layers. All layers use ReLU activation functions. Details can be found in App. B, Fig. 12.

In practice, we also found that a small amount of weight decay and L_2 regularization of the generated deformations can help to ensure smoothness. Thus, the loss function of the deformation network, with regularization parameters γ_1 and γ_2 is

$$L_t = L + \gamma_1 ||\theta||_2 + \gamma_2 ||\mathbf{w}||_2\ , \tag{5}$$

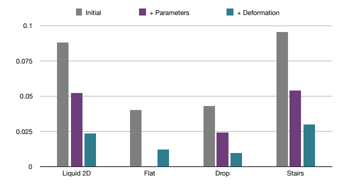
where θ denotes the network weights. In addition, regular SDFs can lead to overly large loss values far away from the surface due to linearly increasing distances. Thus, we apply the $\tanh()$ function to the SDF values, in order to put more emphasis on the surface region.

Special care is required for boundary conditions, i.e, the sides of our domain. Assuming constant values outside of the discretized domain, i.e. $\partial\psi(\mathbf{x})/\partial\mathbf{n} = 0$ for all $\mathbf{x}$ at the domain sides leads to vanishing gradients $\nabla\psi(\mathbf{x}) = \mathbf{0}$ in App. B, Eq. (4). We found this causes artificial minima and maxima in the loss function impeding the training process. Hence, we extrapolate the SDF values with $\partial\psi(\mathbf{x})/\partial\mathbf{n} = \pm 1$ in order to retrieve non zero gradients at the domain sides.

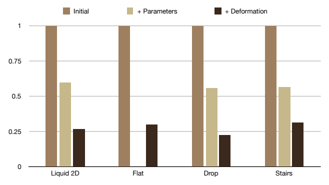
To train both networks we use stochastic gradient descent with an ADAM optimizer and a learning rate of 10^{-3}. Training is performed separately for both networks, with typically 1000 steps for f_d,

	Initial	+Parameters	+Deformation
Liquid 2D	0.0876	0.0521	0.0234
Flat	0.0403	-	0.0121
Drop	0.0431	0.024	0.0096
Stairs	0.0953	0.0537	0.0299

Loss (numeric)



Loss (absolute)



Loss (normalized)

Figure 4: Ablation study for our method. We evaluated the average loss for a test data set of the different data sets discussed in the text. Left: numeric values, again as a graph (center), and a graph of the loss values normalized w.r.t. initial surface loss on the right. Our method achieves very significant and consistent reductions across the very different data sets.

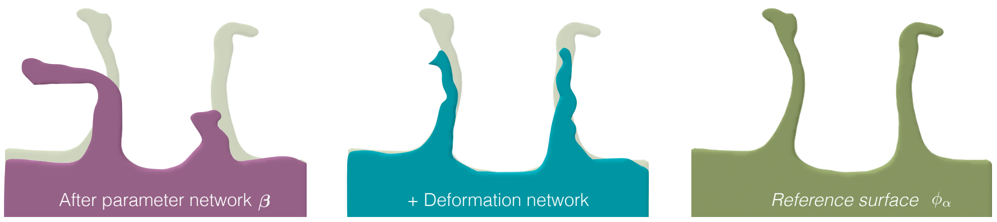

After parameter network β
+ Deformation network
Reference surface ϕ_α

Figure 5: An example of our deformation learning approach. F. l. t. r.: the result after applying weighted deformations, and with an additional deformation from a trained deformation network. Both show the reference surface in light brown in the background, which is shown again for comparison on the right. The inferred deformation manages to reconstruct large parts of the two central arms which can not be recovered by any weighting of the pre-computed deformations (left).

and another ca. 9000 steps for f_d. Full parameters can be found in App. B, Table 2. As training data we generate sets of implicit surfaces from liquid simulations with the FLIP method Bridson (2015). For our 2D inputs, we use single time steps, while our 4D data concatenates 3D surfaces over time to assemble a space-time surface. Working in conjunction, our two networks capture highly complex behavior of the fluid space-time surface ϕ_α over the whole parameter domain. We will evaluate the influence of the networks in isolation and combination in the following.

4 Evaluation

In order to evaluate our method, we first use a two-dimensional parameter space with two dimensional implicit surfaces from a liquid simulation. An overview of the space of 2156 training samples of size 100^2 can be found in the supplemental materials. For our training and synthesis runs, we typically downsample the SDF data, and use a correspondingly smaller resolution for the output of the deformation network, see Appendix C.2, Table 2. The effect of our trained networks in terms of loss reductions is shown on the left side of Fig. 4 under *Liquid 2D*. As baseline we show the loss for the undeformed surface w.r.t. the test data samples. For this 2D data set, employing the trained parameter network reduces the loss to 59.4% of the initial value. Fig. 3 shows the surface of an exemplary result. Although our result does not exactly match the target due to the constraints of the pre-computed deformations, the learned deformation weights lead to a clear improvement in terms of approximating the target.

The inferred deformation of our deformation network further reduces the surface loss to 26.6% of its initial value, as shown in Fig. 4. This is equivalent to a reduction to 44.8% compared the result after applying the weighted deformations. Note that the graphs in this figure correspond to an ablation study: starting with the loss for the undeformed surface, over using only the parameter network, to deformations for a flat surface, to our full algorithm. An example surface for these two-dimensional cases can be found in Fig. 5. This figure compares the surface after applying weighted and inferred deformations, i.e. our full method (right), with a surface deformed by only by deformations weighted by the parameter network (left). The NN deformation manages to reconstruct the two arm in the center of the surface, which the pre-computed deformations fail to capture. It is also apparent

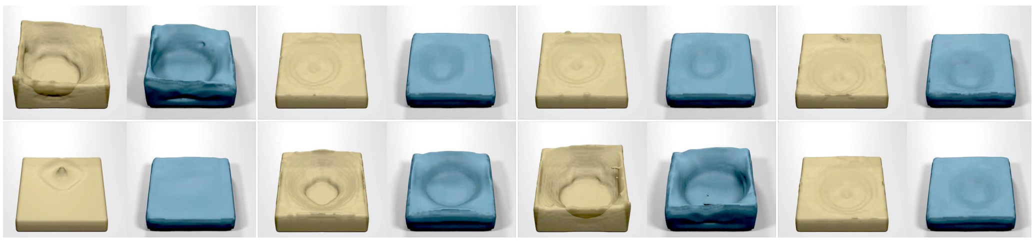


Figure 6: Eight examples of the learned deformations for a flat initial surface. For each pair the reference surfaces are depicted in yellow and the deformed results in blue. The trained model learns to recover a significant portion of the large-scale surface motion over the whole parameters space.

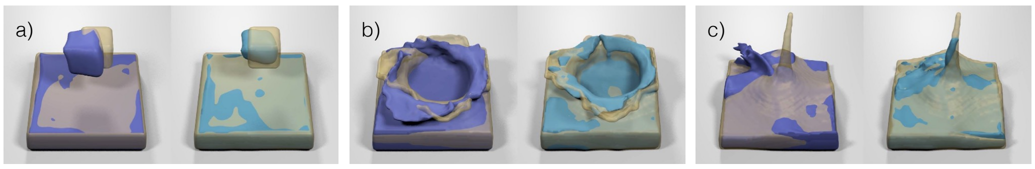


Figure 7: Each pair shows the reference surface in transparent brown, and in purple on the left the deformed surface after applying the precomputed deformations. These surfaces often significantly deviate from the brown target, i.e. the visible purple regions indicates misalignments. In cyan on the right, our final surfaces based on the inferred deformation field. These deformed surface match the target surface closely, and even recover thin features such as the central peak in (c).

that despite the improvement, this surface does not reach the tip of the arms. This is caused by regularization over the varying set of target surfaces, leading to an averaged solution for the deformation. Additional examples for this two dimensional setup can be found in the supplemental video.

4D Surface Data Next we consider complete space-time data sets in four dimensions. with a three dimensional parameter space $\boldsymbol{\alpha}$. The three parameter dimensions are x- and y-coordinates of the initial drop position, as well as its size. We use a total of 1764 reference SDFs with an initial resolution of 100^4, which are down-sampled to a resolution of 40^4. To illustrate the capabilities of the deformation network, we start with a completely flat initial surface as ψ_0, and train the deformation network to recover the targets. As no pre-computed deformations are used for this case, we do not train a parameter network. The flat initial surface represents an especially tough case, as the network can not rely on any small scale details in the reference to match with the features of the targets. Despite this difficulty, the surface loss is reduced to 30% of the initial loss purely based on the deformation network. A set of visual examples can be seen in Fig. 6. Due to the reduced resolution of the inferred deformation w.r.t. the SDF surface, not all small scale features of the targets are matched. However, the NN manages to reconstruct impact location and size very well across the full parameter space. Additional 2D and 4D results can be found in the supplemental materials.

Once we introduce a regular initial surface for ψ_0, in this case the zero parameter configuration with the drop in one corner of the domain with the largest size, our networks perform even better than for the 2D data discussed above. The weighted deformations lead to a loss reduction to 55.6% of the initial value, and the learned deformation reduce the loss further to 22.2% of the baseline loss (Fig. 4). An example is shown in Fig. 7. In contrast to the flat surface test, the network deformation can now shift and warp parts of ψ_0, such as the rim of the splash of Fig. 7 to match the targets.

5 Additional Results with Implicit Surfaces in 4D

Our method yields highly reduced representations which can be used to very efficiently synthesize simulation results. To demonstrate the representational capabilities and the performance of our method, we have integrated the evaluation pipeline for our trained networks into an Android application. As our method yields an implicit surface as 4D array, visualizing the resulting animation is very efficient. We render slices of 3D data as implicit surfaces, and the availability of a full

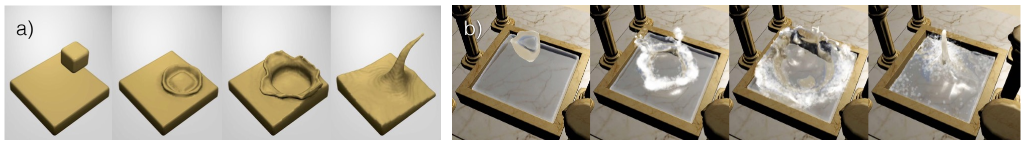


Figure 8: a) Liquid drop data set example: several 3D surfaces of a single simulation data point in ϕ_{α}. b) An example splash generated by our method, visualized interactively.

3D representations makes it possible to add curvature-based shading and secondary particle effects on the fly. In this context, please also consider the supplemental materials, which contain these sequences in motion. They are available at: `https://ge.in.tum.de/publications/2017-prantl-defonn/`.

Performance One of the setup available in this app is the liquid drop setup with 3D parameter space described above. With this setup a user can release drops of liquid at varying positions and of varying size. An example reference and generated result can be found in Fig. 8. For this liquid drop setup, evaluating the network takes 69ms on average, and assembling the final deformation field another 21.5ms. We use double buffering, and hence both tasks are evaluated in a background thread. Rendering the implicit surface takes an average of 21ms, with an average frame rate of 50 fps. The original simulation for the drop setup of Fig. 8 took 530 seconds on average with a parallel implementation to generate a single 4D surface data point.Assuming a best-case slowdown of only 4x for the mobile device, it would require more than 32 minutes to run the original simulation there. Our app generates and renders a full liquid animation in less than one second in total. Thus, our algorithm generates the result roughly 2000 times faster than the regular simulation. Our approach also represents the space of more than 1700 input simulations, i.e., more than 17GB, with less than 30MB of storage.

Stairs A next setup, shown in Fig. 9, captures a continuous flow around a set of obstacles. Liquid is generated in one corner of the simulation domain, and then flows in a U-shaped path around a wall, down several steps. In the interactive visualization, green arrows highlight in- and outflow regions. The three dimensional parametrization of this setup captures a range of positions for the wall and two steps leading to very different flow behaviors for the liquid. In this case the data set consists of 1331 SDFs, and our app uses an output resolution of 50^4. The corresponding loss measurements can be found in the right graphs of Fig. 4. As with the two previously discussed data sets, our approach leads to very significant reductions of the surface loss across the full parameter space, with a final residual loss of 31.3% after applying the learned deformation. Due to larger size of the implicit surfaces and the inferred deformation field, the performance reduces to a frame rate of 30 fps on average, which, however, still allows for responsive user interactions.

Discussion Our approach in its current form has several limitations that are worth mentioning. E.g., we assume that the space of target surfaces have a certain degree of similarity, such that a single surface can be selected as initial surface ψ_0. In addition, our method currently does not make use of the fact that the inputs are generated by a physical process. E.g., it would be highly interesting for future work to incorporate additional constraints such as conservation laws, as currently our results can deviate from an exact conservation of physical properties. E.g., due to its approximating nature our method can lead to parts of the volume disappearing and appearing over time. Additionally, the L2 based loss can lead to rather smooth results, here approaches such as GANs could potentially improve the results.

6 Conclusions

We have presented a novel method to generate space-time surfaces with deformation-aware neural networks. In particular, we have demonstrated the successful inference of weighting sequences of aligned deformations, and the generation of dense deformation fields across a range of varied inputs. Our method exhibits significant improvements in terms surface reconstruction accuracy across the full parameter range. In this way, our networks can capture spaces of complex surface behavior, and

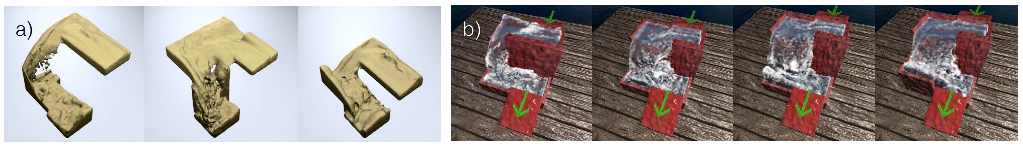


Figure 9: a) Three example configurations from our stairs data set. b) The interactive version of the stair setup shown in the demo app. Notice how the flow around the central wall obstacle changes. As the wall is shifted right, the flow increases corresonpondingly.

allow for real-time interactions with physics effects that are orders of magnitudes slower to compute with traditional solvers.

Beyond liquid surfaces, our deformation networks could also find application for other types of surface data, such as those from object collections or potentially also moving characters. Likewise, it could be interesting to extend our method in order to infer deformations for input sets without an existing parametrization.

REFERENCES

Pulkit Agrawal, Ashvin V Nair, Pieter Abbeel, Jitendra Malik, and Sergey Levine. Learning to poke by poking: Experiential learning of intuitive physics. In *Advances in Neural Information Processing Systems*, pp. 5074–5082, 2016.

Christian Bailer, Kiran Varanasi, and Didier Stricker. Cnn-based patch matching for optical flow with thresholded hinge loss. *arXiv preprint: 1607.08064*, 2016.

Peter Battaglia, Razvan Pascanu, Matthew Lai, Danilo Jimenez Rezende, et al. Interaction networks for learning about objects, relations and physics. In *Advances in Neural Information Processing Systems*, pp. 4502–4510, 2016.

Kiran S Bhat, Steven M Seitz, Jovan Popović, and Pradeep K Khosla. Computing the physical parameters of rigid-body motion from video. In *European Conference on Computer Vision*, pp. 551–565. Springer, 2002.

Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006. ISBN 0387310738.

Robert Bridson. *Fluid Simulation for Computer Graphics*. CRC Press, 2015.

Marcus A Brubaker, Leonid Sigal, and David J Fleet. Estimating contact dynamics. In *Computer Vision, 2009 IEEE 12th International Conference on*, pp. 2389–2396. IEEE, 2009.

Michael B Chang, Tomer Ullman, Antonio Torralba, and Joshua B Tenenbaum. A compositional object-based approach to learning physical dynamics. *arXiv:1612.00341*, 2016.

Mengyu Chu and Nils Thuerey. Data-driven synthesis of smoke flows with CNN-based feature descriptors. *ACM Trans. Graph.*, 36(4)(69), 2017.

Misha Denil, Pulkit Agrawal, Tejas D Kulkarni, Tom Erez, Peter Battaglia, and Nando de Freitas. Learning to perform physics experiments via deep reinforcement learning. *arXiv preprint arXiv:1611.01843*, 2016.

Alexey Dosovitskiy, Philipp Fischery, Eddy Ilg, Caner Hazirbas, Vladimir Golkov, Patrick van der Smagt, Daniel Cremers, Thomas Brox, et al. Flownet: Learning optical flow with convolutional networks. In *International Conference on Computer Vision (ICCV)*, pp. 2758–2766. IEEE, 2015.

Sebastien Ehrhardt, Aron Monszpart, Niloy J Mitra, and Andrea Vedaldi. Learning a physical long-term predictor. *arXiv:1703.00247*, 2017.

Amir Barati Farimani, Joseph Gomes, and Vijay S Pande. Deep learning the physics of transport phenomena. *arXiv:1709.02432*, 2017.

Chelsea Finn, Ian Goodfellow, and Sergey Levine. Unsupervised learning for physical interaction through video prediction. In *Advances in neural information processing systems*, pp. 64–72, 2016.

Katerina Fragkiadaki, Pulkit Agrawal, Sergey Levine, and Jitendra Malik. Learning visual predictive models of physics for playing billiards. *arXiv preprint arXiv:1511.07404*, 2015.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pp. 2672–2680, 2014.

Radek Grzeszczuk, Demetri Terzopoulos, and Geoffrey Hinton. Neuroanimator: Fast neural network emulation and control of physics-based models. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pp. 9–20. ACM, 1998.

Eddy Ilg, Nikolaus Mayer, Tonmoy Saikia, Margret Keuper, Alexey Dosovitskiy, and Thomas Brox. Flownet 2.0: Evolution of optical flow estimation with deep networks. *arXiv preprint: 1612.01925*, 2016.

Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. *CVPR*, 2017.

Max Jaderberg, Karen Simonyan, Andrew Zisserman, et al. Spatial transformer networks. In *Advances in neural information processing systems*, pp. 2017–2025, 2015.

Nikolaos Kyriazis and Antonis Argyros. Physically plausible 3d scene tracking: The single actor hypothesis. In *Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on*, pp. 9–16. IEEE, 2013.

Nikolaos Kyriazis and Antonis Argyros. Scalable 3d tracking of multiple interacting objects. In *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on*, pp. 3430–3437. IEEE, 2014.

Lubor Ladicky, SoHyeon Jeong, Barbara Solenthaler, Marc Pollefeys, and Markus Gross. Data-driven fluid simulations using regression forests. *ACM Trans. Graph.*, 34(6):199, 2015.

Michael Lentine, Mridul Aanjaneya, and Ronald Fedkiw. Mass and momentum conservation for fluid simulation. In *Symposium on Computer Animation*, pp. 91–100. ACM, 2011.

Adam Lerer, Sam Gross, and Rob Fergus. Learning physical intuition of block towers by example. In *International Conference on Machine Learning*, pp. 430–438, 2016.

Wenbin Li, Seyedmajid Azimi, Ales Leonardis, and Mario Fritz. To fall or not to fall: A visual approach to physical stability prediction. *arXiv preprint arXiv:1604.00066*, 2016.

Zichao Long, Yiping Lu, Xianzhong Ma, and Bin Dong. Pde-net: Learning pdes from data. *arXiv:1710.09668*, 2017.

Jonathan Masci, Ueli Meier, Dan Cireşan, and Jürgen Schmidhuber. Stacked convolutional auto-encoders for hierarchical feature extraction. *Artificial Neural Networks and Machine Learning–ICANN 2011*, pp. 52–59, 2011.

Simon Meister, Junhwa Hur, and Stefan Roth. Unflow: Unsupervised learning of optical flow with a bidirectional census loss. *arXiv preprint arXiv:1711.07837*, 2017.

Roozbeh Mottaghi, Hessam Bagherinezhad, Mohammad Rastegari, and Ali Farhadi. Newtonian scene understanding: Unfolding the dynamics of objects in static images. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3521–3529, 2016a.

Roozbeh Mottaghi, Mohammad Rastegari, Abhinav Gupta, and Ali Farhadi. "what happens if..." learning to predict the effect of forces in images. In *European Conference on Computer Vision*, pp. 269–285. Springer, 2016b.

Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *Proc. ICLR*, 2016.

Anurag Ranjan and Michael J. Black. Optical flow estimation using a spatial pyramid network. *CoRR*, abs/1611.00850, 2016. URL `http://arxiv.org/abs/1611.00850`.

Karthik Raveendran, Nils Thuerey, Chris Wojtan, and Greg Turk. Blending Liquids. *ACM Trans. Graph.*, 33 (4):10, August 2014.

Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. In *Proc. ICML, Vol. 32*, pp. II–1278–II–1286, 2014.

Connor Schenck and Dieter Fox. Reasoning about liquids via closed-loop simulation. *arXiv:1703.01656*, 2017.

Russell Stewart and Stefano Ermon. Label-free supervision of neural networks with physics and domain knowledge. In *AAAI*, pp. 2576–2582, 2017.

Nils Thuerey. Interpolations of Smoke and Liquid Simulations. *ACM Trans. Graph.*, 36(1):15, July 2017.

Jonathan Tompson, Kristofer Schlachter, Pablo Sprechmann, and Ken Perlin. Accelerating eulerian fluid simulation with convolutional networks. In *International Conference on Machine Learning*, pp. 3424–3433, 2017.

Aaron Van Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. Pixel recurrent neural networks. In *International Conference on Machine Learning*, pp. 1747–1756, 2016.

Nicholas Watters, Daniel Zoran, Theophane Weber, Peter Battaglia, Razvan Pascanu, and Andrea Tacchetti. Visual interaction networks: Learning a physics simulator from video. In *Advances in Neural Information Processing Systems*, pp. 4542–4550, 2017.

Jiajun Wu, Ilker Yildirim, Joseph J Lim, Bill Freeman, and Josh Tenenbaum. Galileo: Perceiving physical object properties by integrating a physics engine with deep learning. In *Advances in neural information processing systems*, pp. 127–135, 2015.

Jiajun Wu, Joseph J Lim, Hongyi Zhang, Joshua B Tenenbaum, and William T Freeman. Physics 101: Learning physical object properties from unlabeled videos. In *BMVC*, volume 2:6, pp. 7, 2016.

Appendix: Generating Liquid Simulations with Deformation-aware Neural Networks

This supplemental document will first detail the necessary steps to align multiple, weighted deformation fields. Afterwards, we will derive the gradients presented in the paper for the parameter and deformation networks, and then present additional results.

A Deformation Alignment

As before, $\phi_{\boldsymbol{\alpha}}$ denotes the reference signed distance functions (SDFs) of our input parameter space, while ψ denotes instances of a single input surface, typically deformed by our algorithm. We will denote the single initial surface without any deformations applied with ψ_0. Here, we typically use the zero point of our parameter space, i.e., $\psi_0 = \phi_{\alpha_0}$, with $\alpha_0 = \mathbf{0}$. Hence, we aim for deforming ψ_0 such that it matches all instances of $\phi_{\boldsymbol{\alpha}}$ as closely as possible.

For the pre-computed, end-point deformations, it is our goal to only use a single deformation for each dimension of the parameter space $\boldsymbol{\alpha}$. Thus $\mathbf{u}_1$ will correspond to α_1 and be weighted by β_1, and we can apply $\beta_1\mathbf{u}_1$ to compute a deformation for an intermediate point along this dimension. Given the sequence of pre-computed deformations $\{\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_N\}$ and a point in parameter space $\{\beta_1, \ldots, \beta_N\}$ a straight-forward approach is to apply each deformation sequentially

$$\begin{aligned} \psi_1(\mathbf{x}, \boldsymbol{\beta}) &= \psi_0(\mathbf{x} - \beta_1\mathbf{u}_1) \\ \psi_2(\mathbf{x}, \boldsymbol{\beta}) &= \psi_1(\mathbf{x} - \beta_2\mathbf{u}_2) \\ &\vdots \\ \psi_N(\mathbf{x}, \boldsymbol{\beta}) &= \psi_{N-1}(\mathbf{x} - \beta_N\mathbf{u}_N). \end{aligned} \tag{6}$$

However, there are two disadvantages to this approach. The main problem is that the deformations $\mathbf{u}_i$ are only meaningful if applied with $\beta_i = 1$.

Thus, if a previous deformation wasn't applied fully with a weight of 1, each subsequent deformation will lead to an accumulation of deformation errors. The second disadvantage of this simple method

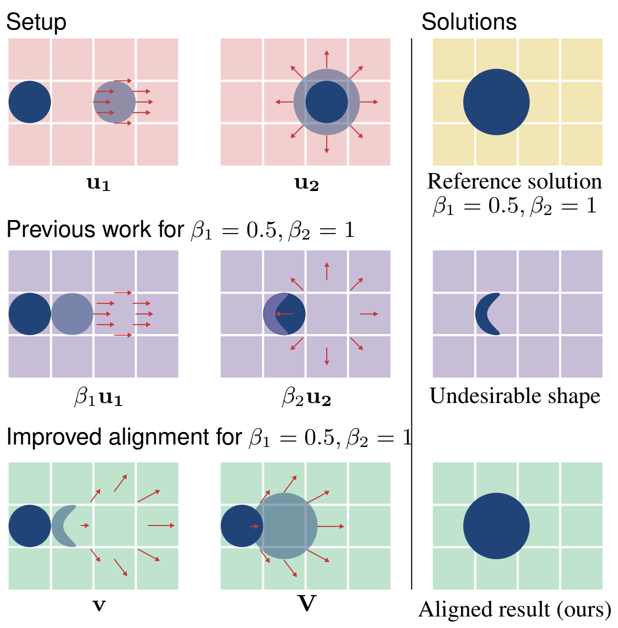


Figure 10: Illustration of our deformation alignment procedure.

is the fact that many advection steps have to be applied in order to arrive at the final deformed SDF. This also affects performance as each advection step introduces additional computations, and scattered memory accesses. This is best illustrated with the an example, shown in Fig. 10. In row 1) the first two figures in red show two pre-computed end-point deformations (red arrows). The first one ($\mathbf{u}_1$) moves a drop to the right, while $\mathbf{u}_2$ changes its size once it has reached its final position. Images with deformations show the source in dark blue, and the deformed surface in light blue. In this example, the two deformations should be combined such that the horizontal position and the drop size can be independently controlled by changing β_1 and β_2. E.g., on the top right, a correct solution for $\beta_1 = 0.5, \beta_2 = 1$ is shown. Row 2) of Fig. 10 shows how these deformations are applied in previous work: The second deformation acts on wrong parts of the surface, as the drop has not reached its left-most position for $\beta_1 = 0.5$. The undesirable result in this case is a partially deformed drop, shown again in the middle row on the right.

We present an alternative approach which aligns the deformation fields to the final position of the deformation sequence. Then, all aligned deformation fields can simply be accumulated by addition, and applied to the input in a single step. To do this, we introduce the intermediate deformation fields:

$$\begin{aligned}
\mathbf{u}_N^*(\mathbf{x}) &= \mathbf{u}_N(\mathbf{x}), \\
\mathbf{u}_{N-1}^*(\mathbf{x}) &= \mathbf{u}_{N-1}(\mathbf{x} - \mathbf{u}_N^*(\mathbf{x})), \\
\mathbf{u}_{N-2}^*(\mathbf{x}) &= \mathbf{u}_{N-2}(\mathbf{x} - \mathbf{u}_N^*(\mathbf{x}) - \mathbf{u}_{N-1}^*(\mathbf{x})), \\
&\vdots \\
\mathbf{u}_1^*(\mathbf{x}) &= \mathbf{u}_1(\mathbf{x} - \mathbf{u}_N^*(\mathbf{x}) - \mathbf{u}_{N-1}^*(\mathbf{x}) \ldots - \mathbf{u}_2^*(\mathbf{x})).
\end{aligned} \tag{7}$$

Each $\mathbf{u}_i^*$ is moved by all subsequent deformations $\mathbf{u}_j^*, j \in [i+1 \cdots N]$, such that it acts on the correct target position under the assumption that $\beta_i = 1$ (we will address the case of $\beta_i \neq 1$ below). The Eulerian representation we are using means that advection steps look backward to gather data, which in our context means that we start with the last deformation $\mathbf{u}_N$ to align previous deformations. Using the aligned deformation fields $\mathbf{u}^*$ we can include $\boldsymbol{\beta}$ and assemble the weighted intermediate fields

$$\mathbf{v}_{\text{sum}}(\mathbf{x}, \boldsymbol{\beta}) = \sum_{i=1}^{N} \beta_i \mathbf{u}_i^*(\mathbf{x}) \tag{8}$$

and an inversely weighted correction field

$$\mathbf{v}_{\text{inv}}(\mathbf{x}, \boldsymbol{\beta}) = -\sum_{i=1}^{N} (1 - \beta_i) \mathbf{u}_i^*(\mathbf{x}). \tag{9}$$

The first deformation field $\mathbf{v}_{\text{sum}}$ represents the weighted sum of all aligned deformations, weighted with the correct amount of deformation specified by the deformation weights β_i. The second deformation $\mathbf{v}_{\text{inv}}$ intuitively represents the offset of the deformation field $\mathbf{v}_{\text{sum}}$ from its destination caused by the $\boldsymbol{\beta}$ weights. Therefore, we correct the position of $\mathbf{v}_{\text{sum}}$ by this offset with the help of an additional forward-advection step calculated as:

$$\mathbf{v}_{\text{fin}}(\mathbf{x} + \mathbf{v}_{\text{inv}}(\mathbf{x}, \boldsymbol{\beta}), \boldsymbol{\beta}) = \mathbf{v}_{\text{sum}}(\mathbf{x}, \boldsymbol{\beta}), \tag{10}$$

This gives us the final deformation field $\mathbf{v}_{\text{fin}}(\mathbf{x}, \boldsymbol{\beta})$. It is important to note that the deformation $\mathbf{v}_{\text{sum}}$ for a position $\mathbf{x}$ is assembled at a location $\mathbf{x}'$ that is not known a-priori. It has to be transported to $\mathbf{x}$ with the help of $\mathbf{v}_{\text{inv}}$, as illustrated in Fig. 11.

This correction is not a regular advection step, as the deformation is being 'pushed' from $\mathbf{x} + \mathbf{v}_{\text{inv}}(\mathbf{x}, \boldsymbol{\beta})$ to $\mathbf{x}$. In order to solve this advection equation we use an inverse semi-Lagrangian step, inspired by algorithms such as the one by Lentine et al. Lentine et al. (2011), pushing values forward with linear interpolation. As multiple values can end up in a single location, we normalize their contribution. Afterwards, we perform several iterations of a "fill-in" step to make sure all cells in the target deformation grid receive a contribution (we simply extend and average deformation values from all initialized cells into uninitialized regions).

The deformed SDF is then calculated with a regular advection step applying the final, aligned deformation with

$$\psi(\mathbf{x}, \boldsymbol{\beta}) = \psi_0(\mathbf{x} - \mathbf{v}_{\text{fin}}(\mathbf{x}, \boldsymbol{\beta})). \tag{11}$$

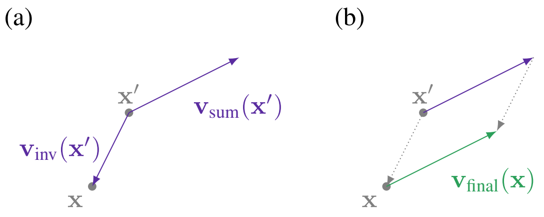

(a)
(b)
$\mathbf{x}'$
$\mathbf{v}_{\text{sum}}(\mathbf{x}')$
$\mathbf{v}_{\text{inv}}(\mathbf{x}')$
$\mathbf{x}$
$\mathbf{x}'$
$\mathbf{v}_{\text{final}}(\mathbf{x})$
$\mathbf{x}$

Figure 11: This figure illustrates the forward advection process: Both deformation $\mathbf{v}_{\text{sum}}$ and the correction $\mathbf{v}_{\text{inv}}$ are initially located at $\mathbf{x}'$ in (a). $\mathbf{v}_{\text{inv}}$ is applied to yield the correct deformation at location $\mathbf{x}$, as shown in (b).

Based on our correction step from Eq. (10) this method now respects the case of partially applied deformations. As the deformations $\mathbf{u}^*(\mathbf{x})$ are already aligned and don't depend on $\boldsymbol{\beta}$, we can pre-compute them. To retrieve the final result it is now sufficient to sum up all deformations in $\mathbf{v}_{\text{sum}}$ and $\mathbf{v}_{\text{inv}}$, then apply one forward-advection step to compute $\mathbf{v}_{\text{fin}}$, and finally deform the input SDF by applying semi-Lagrangian advection. While our method is identical with alignment from previous work Thuerey (2017) for $\beta_i = 1$, it is important for practical deformations with weights $\beta_i \neq 1$.

Our method is illustrated in row 3) of Fig. 10. In the bottom left we see the deformation field $\mathbf{v}_{\text{sum}}$ from previous work. It is also the starting point of our improved alignment, but never applied directly. Our method corrects $\mathbf{v}_{\text{sum}}$ by transforming it into $\mathbf{v}_{\text{fin}}$, bottom center, which acts on the correct spatial locations. In this example, it means that the expanding velocities from $\mathbf{u}_2$ are shifted left to correctly expand the drop based on its initial position. Our method successfully computes the intended result, as shown in the bottom right image.

This algorithm for aligning deformations will be our starting point for learning the weights $\boldsymbol{\beta}$. After applying the weighted deformations, we adjust the resulting surface we an additional deformation field generated by a trained model. In the following, we will derive gradients for learning the weighting as well as the refinement deformation.

B LEARNING DEFORMATIONS

As outlined in the main document, we aim for minimizing the L_2 loss between the final deformed surface and the set of reference surfaces $\phi_{\boldsymbol{\alpha}}$, i.e.:

$$L = \frac{1}{2} \sum_i \left(\psi_0(\mathcal{D}(\mathbf{x}_i, \boldsymbol{\alpha})) - \phi_{\boldsymbol{\alpha}}(\mathbf{x}_i) \right)^2 \ , \tag{12}$$

where $\mathcal{D}(\mathbf{x}_i, \boldsymbol{\alpha})$ denotes the joint application of all weighted and generated deformation fields.

B.1 LEARNING DEFORMATION WEIGHTING

We will first focus on the parameter network to infer the weighting of the pre-computed deformation fields based on the input parameters $\boldsymbol{\alpha}$. Thus, the NN has the goal to compute $\boldsymbol{\beta}(\boldsymbol{\alpha}) \in \mathrm{R}^N = (\beta_1(\boldsymbol{\alpha}), \dots, \beta_N(\boldsymbol{\alpha}))$ in order to minimize Eq. (12). The application of the deformations weighted by $\boldsymbol{\beta}$ includes our alignment step from Sec. A, and hence the neural networks needs to be aware of its influence. To train the parameter network, we need to specify gradients of Eq. (12) with respect to the network weights θ_i. With the chain rule we obtain $\frac{\mathrm{d}}{\mathrm{d}\theta_{ij}^l} L = \frac{\mathrm{d}\beta}{\mathrm{d}\theta_{ij}^l} \frac{\mathrm{d}L}{\mathrm{d}\beta}$. Since the derivative of the network output β_i with respect to a specific network weight θ_{ij}^l is easily calculated with backpropagation Bishop (2006), it is sufficient for us to specify the second term. The gradient of Eq. (12) with respect to the deformation parameter β_i is given by

$$\frac{\mathrm{d}}{\mathrm{d}\beta_i} L = \sum_j \frac{\mathrm{d}}{\mathrm{d}\beta_i} \psi(\mathbf{x}_j, \boldsymbol{\beta}) \left[\psi(\mathbf{x}_j, \boldsymbol{\beta}) - \phi_{\boldsymbol{\alpha}}(\mathbf{x}_j) \right], \tag{13}$$

where we have inserted Eq. (11). While the second term in the sum is easily computed, we need to calculate the first term by differentiating Eq. (11) with respect to β_i, which yields

$$\frac{\mathrm{d}}{\mathrm{d}\beta_i}\psi(\mathbf{x},\boldsymbol{\beta}) = -\frac{\mathrm{d}}{\mathrm{d}\beta_i}\mathbf{v}_{\text{fin}}(\mathbf{x},\boldsymbol{\beta}) \cdot \nabla\psi_0(\mathbf{x}-\mathbf{v}_{\text{fin}}(\mathbf{x},\boldsymbol{\beta})). \tag{14}$$

As the gradient of ψ_0 is straight forward to compute, $\frac{\mathrm{d}}{\mathrm{d}\beta_i}\mathbf{v}_{\text{fin}}(\mathbf{x},\boldsymbol{\beta})$ is crucial in order to compute a reliable derivative. It is important to note that even for the case of small corrections $\mathbf{v}_{\text{inv}}(\mathbf{x},\boldsymbol{\beta})$, Eq. (10) cannot be handled as another backward-advection step such as $\mathbf{v}_{\text{fin}}(\mathbf{x},\boldsymbol{\beta}) = \mathbf{v}_{\text{sum}}(\mathbf{x} - \mathbf{v}_{\text{inv}}(\mathbf{x},\boldsymbol{\beta}),\boldsymbol{\beta})$. While it might be tempting to assume that differentiating this advection equation will produce reasonable outcomes, it can lead to noticeable errors in the gradient. These in turn quickly lead to diverging results in the learning process, due to the non-linearity of the problem.

The correct way of deriving the change in $\mathbf{v}_{\text{fin}}(\mathbf{x},\boldsymbol{\beta})$ is by taking the total derivative of $\mathbf{v}_{\text{sum}}(\mathbf{x},\boldsymbol{\beta}) = \mathbf{v}_{\text{fin}}(\mathbf{x}+\mathbf{v}_{\text{inv}}(\mathbf{x},\boldsymbol{\beta}),\boldsymbol{\beta})$ with respect to β_i:

$$\begin{aligned}&\frac{\mathrm{d}}{\mathrm{d}\beta_i}\mathbf{v}_{\text{sum}}(\mathbf{x},\boldsymbol{\beta})\\ &= \frac{\partial}{\partial\beta_i}\mathbf{v}_{\text{fin}}(\mathbf{x}+\mathbf{v}_{\text{inv}}(\mathbf{x},\boldsymbol{\beta}),\boldsymbol{\beta}) + \mathbf{J}_V(\mathbf{x}+\mathbf{v}_{\text{inv}}(\mathbf{x},\boldsymbol{\beta}),\boldsymbol{\beta})\,\frac{\partial}{\partial\beta_i}\mathbf{v}_{\text{inv}}(\mathbf{x},\boldsymbol{\beta}),\end{aligned} \tag{15}$$

where, $\mathbf{J}_V(\mathbf{x}+\mathbf{v}_{\text{inv}}(\mathbf{x},\boldsymbol{\beta}),\boldsymbol{\beta})$ denotes the Jacobian of $\mathbf{v}_{\text{fin}}$ with respect to $\mathbf{x}$, evaluated at $\mathbf{x} + \mathbf{v}_{\text{inv}}(\mathbf{x},\boldsymbol{\beta})$. Rearranging Eq. (15) and inserting $\mathbf{v}_{\text{sum}}$ and $\mathbf{v}_{\text{inv}}$ yields

$$\frac{\partial}{\partial\beta_i}\mathbf{v}_{\text{fin}}(\mathbf{x}+\mathbf{v}_{\text{inv}}(\mathbf{x},\boldsymbol{\beta}),\boldsymbol{\beta}) \tag{16}$$

$$\begin{aligned}&= \frac{\mathrm{d}}{\mathrm{d}\beta_i}\mathbf{v}_{\text{sum}}(\mathbf{x},\boldsymbol{\beta}) - \mathbf{J}_V(\mathbf{x}+\mathbf{v}_{\text{inv}}(\mathbf{x},\boldsymbol{\beta}),\boldsymbol{\beta})\,\frac{\partial}{\partial\beta_i}\mathbf{v}_{\text{inv}}(\mathbf{x},\boldsymbol{\beta})\\ &= \frac{\mathrm{d}}{\mathrm{d}\beta_i}\sum_{i=1}^{N}\beta_i\mathbf{u}_i^*(\mathbf{x}) + \mathbf{J}_V(\mathbf{x}+\mathbf{v}_{\text{inv}}(\mathbf{x},\boldsymbol{\beta}),\boldsymbol{\beta})\,\frac{\partial}{\partial\beta_i}\sum_{i=1}^{N}(1-\beta_i)\,\mathbf{u}_i^*(\mathbf{x})\\ &= [\mathbf{1} - \mathbf{J}_V(\mathbf{x}+\mathbf{v}_{\text{inv}}(\mathbf{x},\boldsymbol{\beta}),\boldsymbol{\beta})]\,\mathbf{u}_i^*(\mathbf{x}).\end{aligned} \tag{17}$$

We note that the Jacobian in the equation above has small entries due to the smooth nature of the deformations $\mathbf{v}_{\text{fin}}$. Thus, compared to the unit matrix it is small in magnitude. Note that this relationship is not yet visible in Eq. (15). We have verified in experiments that $\mathbf{J}_V$ does not improve the gradient significantly, and we thus set this Jacobian to zero, arriving at

$$\frac{\partial}{\partial\beta_i}\mathbf{v}_{\text{fin}}(\mathbf{x}+\mathbf{v}_{\text{inv}}(\mathbf{x},\boldsymbol{\beta}),\boldsymbol{\beta}) \approx \mathbf{u}_i^*(\mathbf{x}), \tag{18}$$

where the $\mathbf{u}^*$ are the deformation fields aligned for the target configuration from Eq. (7). We use Eq. (18) to estimate the change in the final deformation fields for changes of the i-th deformation parameter. We see that this equation has the same structure as Eq. (10). On the left-hand side, we have $\frac{\partial}{\partial\beta_i}\mathbf{v}_{\text{fin}}$, evaluated at $\mathbf{x}+\mathbf{v}_{\text{inv}}(\mathbf{x},\boldsymbol{\beta})$, whereas $\mathbf{u}_i^*$ on the right-hand side is evaluated at $\mathbf{x}$. To calculate $\frac{\mathrm{d}}{\mathrm{d}\beta_i}\mathbf{v}_{\text{fin}}(\mathbf{x},\boldsymbol{\beta})$ then, we can use the same forward-advection algorithm, which is applied to the correction in Eq. (10). With this, we have all the necessary components to assemble the gradient from Eq. (13) for training the parameter network with back-propagation.

B.2 Learning to Generate Deformations

Our efforts so far have been centered around producing a good approximation of $\phi_{\boldsymbol{\alpha}}$, with a set of given end-point deformations $\{\mathbf{u}_0,\ldots,\mathbf{u}_n\}$. The performance of this method is therefore inherently constrained by the amount of variation we can produce with the deformation inputs. To allow for more variation, we propose to generate an additional space-time deformation field $\mathbf{w}(\boldsymbol{\alpha})$, that changes with the simulation parameters $\boldsymbol{\alpha}$. Once again, we model this function with a neural network, effectively giving the network more expressive capabilities to directly influence the final deformed surface.

For this network we choose a structure with a set of four-dimensional deconvolution layers that generate a dense space-time deformation field. We apply the trained deformation with an additional

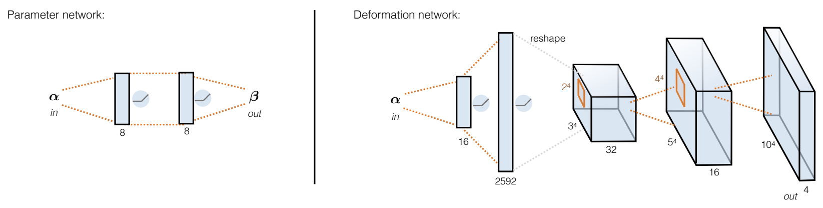


Figure 12: Overview of our two neural networks. While the parameter network (left) is simple, consisting of two fully connected layers, its cost functions allows it to learn how to apply multiple long-range, non-linear deformation fields. The deformation network (right), which makes use of several de-convolutional layers, instead learns to generate dense deformation fields to refine the final surface.

advection step after applying the deformations weighted by the parameter network:

$$\tilde{\psi}(\mathbf{x}) = \psi_0\left(\mathbf{x} - \mathbf{v}_{\text{fin}}(\mathbf{x}, \boldsymbol{\beta}(\boldsymbol{\alpha}))\right), \tag{19}$$

$$\psi(\mathbf{x}) = \tilde{\psi}\left(\mathbf{x} - \mathbf{w}(\mathbf{x}, \boldsymbol{\alpha})\right). \tag{20}$$

Thus, the deformation network only has to learn to refine the surface $\tilde{\psi}$ after applying the weighted deformations, in order to accommodate the nonlinear behavior of $\phi_{\boldsymbol{\alpha}}$.

As input, we supply the deformation network with the simulation parameters $\boldsymbol{\alpha} = (\alpha_i, \dots, \alpha_N)$ as a vector. The output of the network are four-component vectors, with the resolution $R_x \times R_y \times R_z \times R_t$. Note that in general the SDF resolution and the deformation resolution do not need to be identical. Given a fixed SDF resolution, we can use a smaller resolution for the deformation, which reduces the number of weights and computations required for training. Thus in practice, each four-dimensional vector of the deformation acts on a region of the SDF, for which we assume the deformation to be constant. Therefore, we write the deformation field as

$$\mathbf{w}(\mathbf{x}, \boldsymbol{\alpha}) = \sum_j \xi_j(\mathbf{x})\, \mathbf{w}_j(\boldsymbol{\alpha}), \tag{21}$$

where $\xi_j(\mathbf{x})$ is the indicator function of the j-th region on which the four-dimensional deformation vector $\mathbf{w}_j(\boldsymbol{\alpha})$ acts. This vector is the j-th output of the deformation network.

For training, we need to calculate the gradient of the loss-function Eq. (12) with respect to the network weights. Just like in the previous section, it is sufficient to specify the gradient with respect to the network outputs $\mathbf{w}_i(\boldsymbol{\alpha})$. Deriving Eq. (12) yields

$$\begin{aligned}
&\frac{\mathrm{d}}{\mathrm{d}\mathbf{w}_i} L \\
=&\sum_j \frac{\mathrm{d}}{\mathrm{d}\mathbf{w}_i} \psi(\mathbf{x})\left(\psi(\mathbf{x}) - \phi_{\boldsymbol{\alpha}}(\mathbf{x})\right) \\
=&\sum_j \frac{\mathrm{d}}{\mathrm{d}\mathbf{w}_i} \tilde{\psi}\left(\mathbf{x} - \mathbf{w}(\mathbf{x}, \boldsymbol{\alpha})\right)\ \left(\psi(\mathbf{x}) - \phi_{\boldsymbol{\alpha}}(\mathbf{x})\right) \\
=&-\sum_j X_i(\mathbf{x}_j)\, \nabla\tilde{\psi}(\mathbf{x}_j - \mathbf{w}(\mathbf{x}_j, \boldsymbol{\alpha}))\ \left(\psi(\mathbf{x}_j, \boldsymbol{\alpha}) - \phi_{\boldsymbol{\alpha}}(\mathbf{x}_j)\right).
\end{aligned} \tag{22}$$

Thus, we can calculate the derivative by summation over the region that is affected by the network output $\mathbf{w}_i$. The gradient term is first calculated by evaluating a finite difference stencil on $\tilde{\psi}(\mathbf{x}_j)$ and then advecting it with the corresponding deformation vector $\mathbf{w}(\mathbf{x}_j, \boldsymbol{\alpha})$. The other terms in Eq. (22) are readily available. Alg. 1 summarizes our algorithm for training the deformation network. In particular, it is important to deform the input SDF gradients with the inferred deformation field, in order to calculate the loss gradients in the correct spatial location for backpropagation.

ALGORITHM 1: Training the deformation network

```
Data: training samples from $\phi_{\boldsymbol{\alpha}}$
Result: trained deformation network weights $\Theta$
for each training sample $\{\tilde{\boldsymbol{\alpha}}, \tilde{\phi}\}$ do
    evaluate neural network to compute $\beta(\tilde{\boldsymbol{\alpha}})$
    load reference SDF $\tilde{\phi}$, initial SDF $\psi_0$
    calculate $\mathbf{v}_{\text{fin}}(\mathbf{x}_i, \beta(\tilde{\boldsymbol{\alpha}}))$
    $\tilde{\psi}$ = advect $\psi_0$ with $\mathbf{v}_{\text{fin}}$
    calculate $\nabla\tilde{\psi}$
    evaluate neural network to compute $\mathbf{w}_i(\tilde{\boldsymbol{\alpha}})$
    assemble $\mathbf{w}(\mathbf{x}_i)$ from $\mathbf{w}_i(\tilde{\boldsymbol{\alpha}}, \Theta)$ according to Eq. (21)
    advect $\tilde{\psi}$ with $\mathbf{w}$
    advect $\nabla\tilde{\psi}$ with $\mathbf{w}$
    for each $\mathbf{w}_i$ do
        calculate the gradient $\frac{\mathrm{d}}{\mathrm{d}\mathbf{w}_i}L$ according to Eq. (22)
    end
    backpropagate $\frac{\mathrm{d}}{\mathrm{d}\mathbf{w}_i}L_t$ from Eq. (5) to adjust $\Theta$
end
```

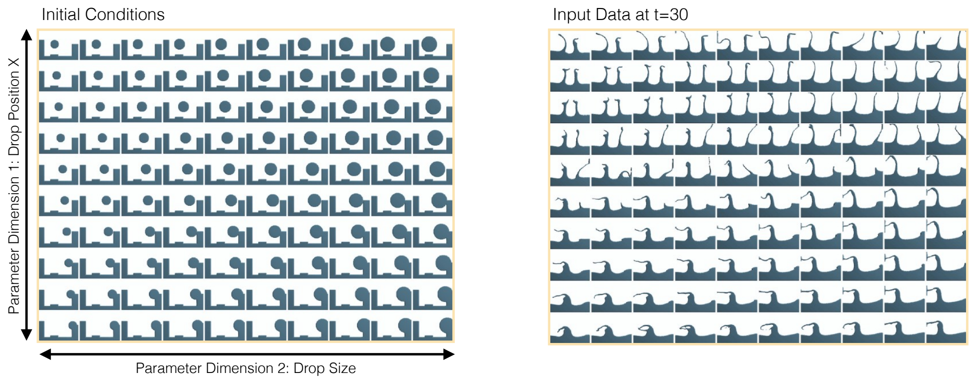

Initial Conditions
Input Data at t=30
Parameter Dimension 1: Drop Position X
Parameter Dimension 2: Drop Size

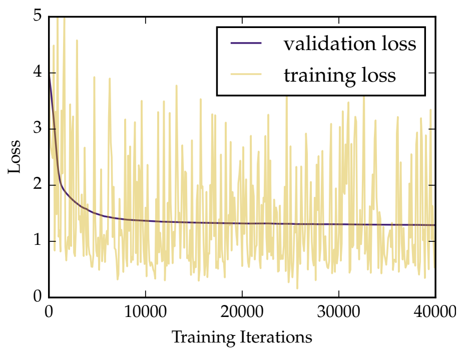
Figure 13: The left image illustrates the initial conditions of our two dimensional parameter space setup. It consists of a set of two-dimensional liquid simulations, which vary the position of the liquid drop along x as α_1, and its size as α_2. The right half shows the data used for training at $t = 30$. Note the significant amount of variance in positions of small scale features such as the thin sheets. Both images show only a subset of the whole data.

C Additional Evaluation

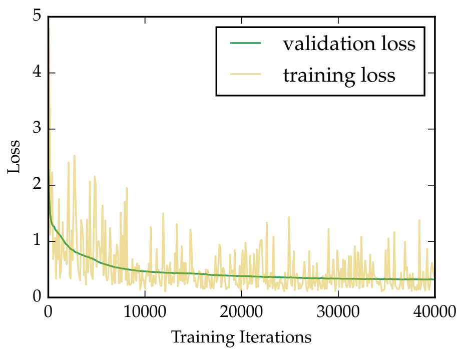
C.1 2D Data Set

In the following, we explain additional details of the evaluation examples. For the two dimensional data set, we use the SDFs extracted from 2D simulations of a drop falling into a basin. As simulation parameters we choose α_1 to be the size of the drop, and α_2 to be its initial x-position, as shown in Fig. 13. From this simulation we extract a single frame at $t = 30$, which gives us a two-dimensional parameter-space $\boldsymbol{\alpha} = (\alpha_1, \alpha_2)$, where each instance of $\boldsymbol{\alpha}$ has a corresponding two-dimensional SDF. In order to train the networks described in section 3, we sample the parameter domain with a regular 44×49 grid, which gives us 2156 training samples, of which we used 100 as a validation set.

Fig. 14 shows the validation loss and the training loss over the iterations both for parameter learning and for deformation learning. We observe that in both cases the learning process reduces the loss, and finally converges to a stable solution. This value is lower in the case of deformation training, which can be easily explained with the increased expressive capabilities of the deformation network.



(a) Parameter Learning



(b) Deformation learning

Figure 14: Loss during training both for parameter learning and deformation learning. In yellow we show the loss for the current sample, while the dark line displays the loss evaluated on the validation set.

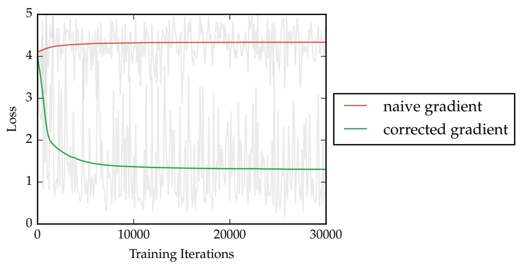


Figure 15: Training with different gradient approximations: validation loss with a simplified advection (red), and the correct gradient from forward advection (green). The simplified version does not converge.

We verified that the solution converged by continuing training for another 36000 steps, during which the change of the solution was negligible.

As mentioned above, it might seem attractive to use a simpler approximation for the forward advection in Eq. (13), i.e., using a simpler, regular advection step. However, due to the strong non-linearity of our setting, this prevents the network from converging, as shown in Fig. 15.

The effect of our deformation network approach is illustrated in Fig. 5. This figure compares our full method (on the right) with several other algorithms. A different, but popular approach for non-linear dimensionality reduction, which can be considered as an alternative to our method, is to construct a reduced basis with PCA. Using the mean surface with four eigenvectors yields a similar reduction to our method in terms of memory footprint. We additionally re-project the different reference surfaces into the reduced basis to improve the reconstruction quality of the PCA version. However, despite this the result is a very smooth surface that fails to capture any details of the behavior of the parameter space, as can be seen in the left column of Fig. 5.

The next column of this figure (in pink) shows the surfaces obtained with the learned deformation weights with our parameter network (Fig. 12 top), but without an additional deformation network. As this case is based on end-point deformations, it cannot adapt to larger changes of surface structure in the middle of the domain. In contrast, using our full pipeline with the deformation network yields surfaces that adapt to the varying behavior in the interior of the parameter space, as shown on the right side of Fig. 5. However, it is also apparent that the deformations generated by our approach do not capture every detail of the references. The solution we retrieve is regularized by the varying reference surfaces in small neighborhoods of α, and the networks learns an averaged behavior from the inputs.

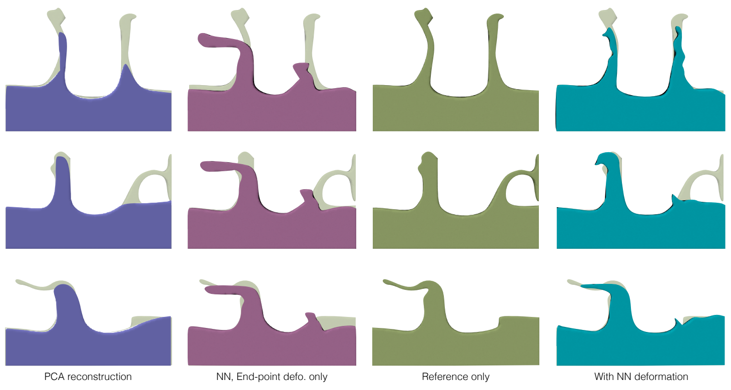


Figure 16: Different example surfaces from the 2D parameter space of Fig. 13. From left to right: surfaces reconstructed with PCA (purple), weighted deformations using a trained parameter network (pink), the reference surfaces (brown), and on the far right the output of our full method with a deformation network (teal). Note that none of the other methods is able to reconstruct both arms of liquid in the first row, as well as the left sheet in the bottom row. The reference surfaces are shown in light brown in the background for each version.

C.2 4D Data Sets

Below we give additional details for our results for the 4D data sets and experiments presented in the main document.

Liquid Drop As our first 4D test case, we chose a drop of liquid falling into a basin. As our simulation parameters we chose the x- and y-coordinates of the initial drop position, as well as the size of the drop. We typically assume that the z-axis points upwards. To generate the training data, we sample the parameter space on a regular grid, and run simulations, each with a spatial resolution of 100^3 to generate a total of 1764 reference SDFs. Here, ψ_0 contains a 4D SDF of a large drop falling into the upper right corner of the basin. In Fig. 17 we show additional examples how the introduction of the deformation network helps to represent the target surface across the full parameter range.

The advantages of our approach also become apparent when comparing our method with a direct interpolation of SDF data-sets, i.e., without any deformation. Our algorithms requires a single full-resolution SDF, three half resolution deformations, and the neural network weights (ca. 53.5k). While a single 40^4 SDF requires ca. 2.5m scalar values, all deformations and network weights require ca. 2m scalars in total. Thus our representation encodes the full behavior with less storage than two full SDFs. To illustrate this point, we show the result of a direct SDF interpolation in Fig. 18. Here we sample the parameter space with 8 SDFs in total (at all corners of the 3D parameter space). Hence, this version requires more than 4x the storage our approach requires. Despite the additional memory, the direct interpolations of SDFs lead to very obvious, and undesirable artifacts. The results shown on the right side of Fig. 18 neither represent the initial drop in (a), nor the resulting splash in (b). Rather, the SDF interpolation leads to strong ghosting artifacts, and an overall loss of detail. Instead of the single drop and splash that our method produces, it leads to four smoothed, and repeated copies. Both the PCA example above, and this direct SDF interpolation illustrate the usefulness of representing the target surface in terms of a learned deformation.

For the falling drop setup, our video also contains an additional example with a larger number of 14 pre-computed deformations. This illustrates the gains in quality that can be achieved via a larger number of deformation fields. However, in this case the parameter and deformation network only lead to negligible changes in the solution due to the closely matching surface from the pre-computed deformations.

Stairs Our second test setup illustrates a different parameter space that captures a variety of obstacle boundary conditions parametrized with $\boldsymbol{\alpha}$. Our first two simulation parameters are the heights of

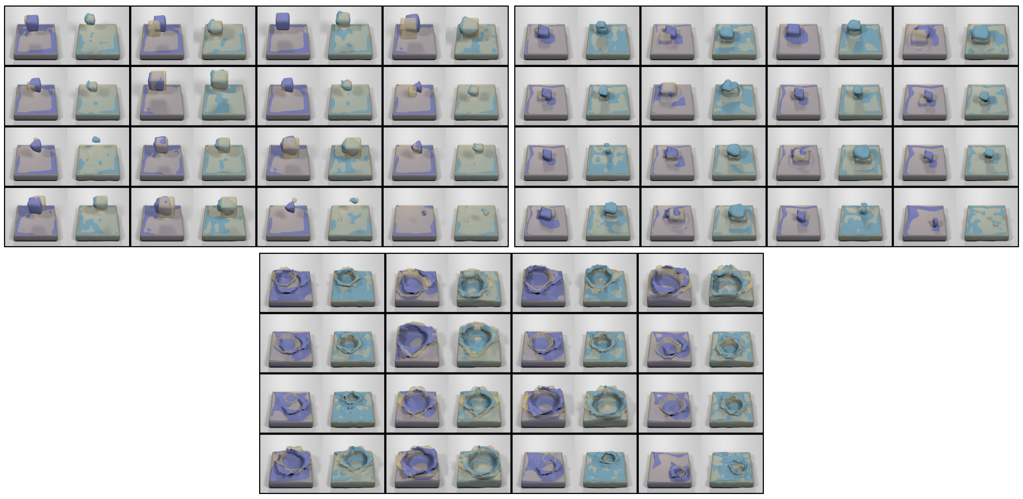


Figure 17: Additional examples of the influence of the deformation network for three different time steps ($t = 1, 4, 8$ from top to bottom). Each pair shows the reference surface in transparent brown, and in purple on the left the deformed surface after applying the precomputed deformations. These surfaces often significantly deviate from the brown target, i.e. the visible purple regions indicates misalignments. In cyan on the right, our final surfaces based on the inferred deformation field. These deformed surface often match the target surface much more closely.

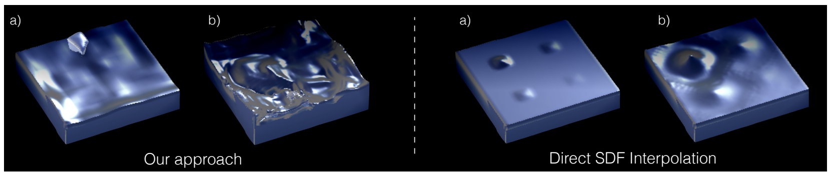


Figure 18: Two frames generated with our approach (left) and with a direct SDF interpolation using a similar amount of overall memory (right). The latter looses the inital drop shape (a), and removes all splash detail (b). In addition, the direct SDF interpolation leads to strong ghosting artifacts with four repeated patterns.

two stair-like steps, while the third parameter is controlling the position of a middle divider obstacle, as illustrated in Fig. 19. The liquid flows in a U-shaped manner around the divider, down the steps. For this setup, we use a higher overall resolution for both space-time SDFs, as well as for the output of the deformation network. Performance details can be found in Table 1.

Fig. 20 depicts still frames captured from our mobile application for this setup. With this setup the user can adjust stair heights and wall width dynamically, while deformations are computed in the background. While this setup has more temporal coherence in its motion than the drop setup, the changing obstacle boundary conditions lead to strongly differing streams over the steps of the obstacle geometry. E.g., changing the position of the divider changes the flow from a narrow, fast stream to a slow, broad front.

Table 1: Performance and setup details of our 4D data sets in the Android app measured on a Samsung S8 device. The *"defo. align"* step contains alignment and rescaling of the deformations.

	SDF res.	Defo. res.	NN eval.	Defo. align	Rendering
Drop	40^4	20^4	69ms	21.5ms	21ms
Staris	50^4	25^4	410ms	70ms	35ms

Table 2: Overview of our 2D and 4D simulation and machine learning setups. Timings were measured on a Xeon E5-1630 with 3.7GHz. *Res*, *SDF* and *Defo* denote resolutions for simulation, training, and the NN deformation, respectively; *Sim* and *Train* denote simulation and training runtimes. $s_p, s_d, \gamma_1, \gamma_2$ denote training steps for parameters, training steps for deformation, and regularization parameters, respectively.

Setup	Res.	SDF	Defo.	Sim.	Train	s_p	s_d
2D setup, Fig. 13	100^2	100^2	25^2	-	186s	40k	10k
Drop, Fig. 8	$100^3 \cdot 100$	40^4	10^4	8.8m	22m	12k	2k
Stairs, Fig. 20	$110^3 \cdot 110$	50^4	15^4	9.7m	186m	9k	1k

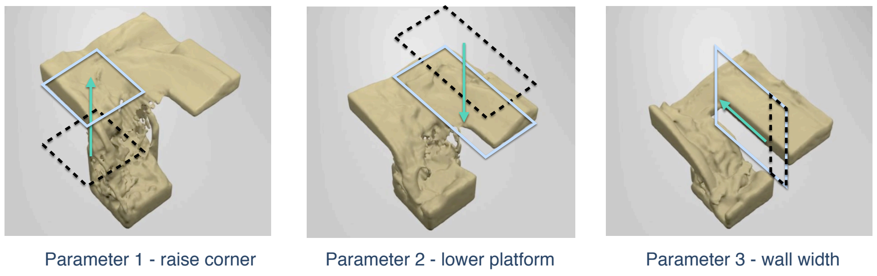


Figure 19: The geometric setup of the three deformations of our stairs setup from 20 are illustrated in this figure.

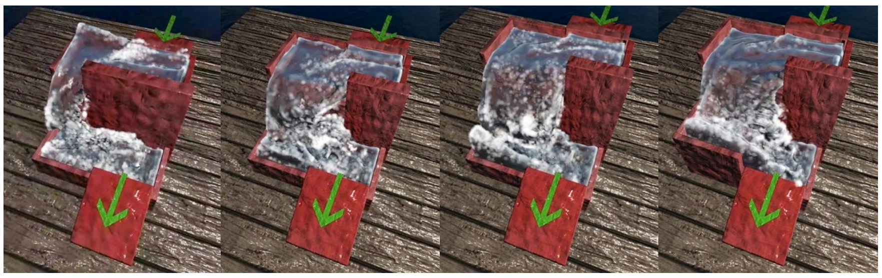


Figure 20: These screens illustrate our stairs setup running in our mobile application. From left to right, the middle divider is pulled back, leading to an increased flow over the step in the back. In the right-most image, the left corner starts to move up, leading to a new stream of liquid pouring down into the outflow region in the right corner of the simulation domain.

TRANQUIL CLOUDS: NEURAL NETWORKS FOR LEARNING TEMPORALLY COHERENT FEATURES IN POINT CLOUDS

Lukas Prantl
Department of Computer Science
Technical University of Munich
Munich, Germany

Nuttapong Chentanez
NVIDIA
Bangkok, Thailand

Stefan Jeschke
NVIDIA
Vienna, Austria

Nils Thuerey
Department of Computer Science
Technical University of Munich
Munich, Germany

ABSTRACT

Point clouds, as a form of Lagrangian representation, allow for powerful and flexible applications in a large number of computational disciplines. We propose a novel deep-learning method to learn stable and temporally coherent feature spaces for points clouds that change over time. We identify a set of inherent problems with these approaches: without knowledge of the time dimension, the inferred solutions can exhibit strong flickering, and easy solutions to suppress this flickering can result in undesirable local minima that manifest themselves as halo structures. We propose a novel temporal loss function that takes into account higher time derivatives of the point positions, and encourages mingling, i.e., to prevent the aforementioned halos. We combine these techniques in a super-resolution method with a truncation approach to flexibly adapt the size of the generated positions. We show that our method works for large, deforming point sets from different sources to demonstrate the flexibility of our approach.

1 INTRODUCTION

Deep learning methods have proven themselves as powerful computational tools in many disciplines, and within it a topic of strongly growing interest is deep learning for point-based data sets. These Lagrangian representations are challenging for learning methods due to their unordered nature, but are highly useful in a variety of settings from geometry processing and 3D scanning to physical simulations, and since the seminal work of Qi Charles et al. (2017), a range of powerful inference tasks can be achieved based on point sets. Despite their success, interestingly, no works so far have taken into account time. Our world, and the objects within it, naturally move and change over time, and as such it is crucial for flexible point-based inference to take the time dimension into account. In this context, we propose a method to learn temporally stable representations for point-based data sets, and demonstrate its usefulness in the context of super-resolution.

An inherent difficulty of point-based data is their lack of ordering, which makes operations such as convolutions, which are easy to perform for Eulerian data, unexpectedly difficult. Several powerful approaches for point-based convolutions have been proposed (Qi et al., 2017; Hermosilla et al., 2018; Hua et al., 2018), and we leverage similar neural network architectures in conjunction with the permutation-invariant *Earth Mover's Distance* (EMD) to propose a first formulation of a loss for temporal coherence.

In addition, several works have recognized the importance of training point networks for localized patches, in order to avoid having the network to rely on a full view of the whole data-set for tasks that are inherently local, such as normal estimation (Qi Charles et al., 2017), and super-resolution (Yu et al., 2018a). This also makes it possible to flexibly process inputs of any size without being limited by memory requirements. Later on we will demonstrate the importance of such a patch-based approach with sets of changing cardinality in our setting. A general challenge here is to deal with varying input sizes, and for super-resolution tasks, also varying output sizes. Thus, in summary we target an extremely challenging learning problem: we are facing permutation-invariant inputs and targets of varying size, that dynamically move and deform over time. In order to enable deep learning approaches in this context, we make the following key contributions: Permutation invariant loss terms for temporally coherent point set generation; A Siamese training setup and generator architecture for point-based super-resolution with neural networks; Enabling improved output variance by allowing for dynamic adjustments of the output size; The identification of a specialized form of mode collapse for temporal point networks, together with a loss term to remove them. We demonstrate that these contributions together make it possible to infer stable solutions for dynamically moving point clouds with millions of points.

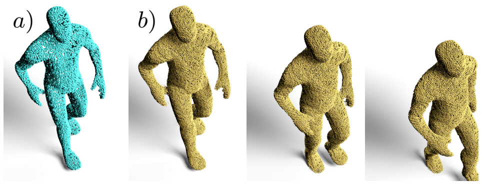


Figure 1: Our algorithm upsamples an input point cloud (a) in a temporally coherent manner. Three exemplary outputs are shown in yellow in (b).

More formally, we show that our learning approach can be used for generating a point set with an increased resolution from a given set of input points. The generated points should provide an improved discretization of the underlying ground truth shape represented by the initial set of points. For the increase, we will target a factor of two to three per spatial dimension. Thus, the network has the task to estimate the underlying shape, and to generate suitable sampling positions as output. This is generally difficult due to the lack of connectivity and ordering, and in our case, positions that move over time in combination with a changing number of input points. Hence it is crucial that the network is able to establish a temporally stable latent space representation. Although we assume that we know correspondences over time, i.e., we know which point at time t moved to a new location at time $t + \Delta t$, the points can arbitrarily change their relative position and density over the course of their movement, leading to a substantially more difficult inference problem than for the static case.

2 Related Work

Deep learning with static point sets was first targeted in PointNet (Qi Charles et al., 2017) via order-invariant networks, while PointNet++ (Qi et al., 2017) extended this concept to generate features for localized groups similar to a convolution operation for grid-based data. This concept can be hierarchically applied to the generated groups, in order to extract increasingly abstract and global features. Afterwards, the extracted features can be interpolated back to the original point cloud. The goal to define point convolutions has been explored and extended in several works. The MCNN approach (Hermosilla et al., 2018) phrased convolution in terms of a Monte Carlo integration. PointCNN (Hua et al., 2018) defined a pointwise convolution operator using nearest neighbors, while extension-restriction operators for mapping between a point cloud function and a volumetric function were used in Atzmon et al. (2018). The PU-Net (Yu et al., 2018a) proposed a network for upsampling point clouds, and proposed a similar hierarchical network structure of PointNets along the lines of PointNet++ to define convolutions. Being closer to our goals, we employ this approach for convolutional operations in our networks below. We do not employ the edge-aware variant of the PU-Net (Yu et al., 2018b) here, to keep it as simple and general as possible as we focus on temporal changes in our work.

Permutation invariance is a central topic for point data, and was likewise targeted in other works (Ravanbakhsh et al., 2016; Zaheer et al., 2017). The Deep Kd-network (Klokov and Lempitsky, 2017) defined a hierarchical convolution on point clouds via kd-trees. PointProNets (Roveri et al., 2018) employed deep learning to generate dense sets of points from sparse and noisy input points for 3D reconstruction applications. PCPNet (Guerrero et al., 2018), as another multi-scale variant of

PointNet, has demonstrated high accuracy for estimating local shape properties such as normal or curvature. P2PNet (Yin et al., 2018) used a bidirectional network and extends PointNet++ to learn a transformation between two point clouds with the same cardinality.

Recently, the area of point-based learning has seen a huge rise in interest. One focus here are 3D segmentation problems, where numerous improvements were proposed, e.g., by SPLATNet (Su et al., 2018), SGPN (Wang et al., 2018a), SpiderCNN (Xu et al., 2018), PointConv (Wu et al., 2018), SO-NEt(Li et al., 2018a) and 3DRNN (Ye et al., 2018). Other networks such as Flex Convolution (Groh et al., 2018), the SuperPoint Graph (Landrieu and Simonovsky, 2018), and the fully convolutional network (Rethage et al., 2018) focused on large scale segmentation. Additional areas of interest are shape classification (Wang et al., 2018b; Lei et al., 2018; Zhang and Rabbat, 2018; Skouson, 2018) and object detection (Simon et al., 2018; Zhou and Tuzel, 2018), and hand pose tracking (Ge et al., 2018). Other works have targeted rotation and translation invariant inference (Thomas et al., 2018), and point cloud autoencoders (Yang et al., 2018; Deng et al., 2018). A few works have also targeted generative models based on points, e.g., for point cloud generation (Sun et al., 2018), and with adversarial approaches (Li et al., 2018b). It is worth noting here that despite the huge interest, the works above do not take into account temporally changing data, which is the focus of our work. A notable exception is an approach for scene flow (Liu et al., 2018), in order to estimate 3D motion directly on the basis of point clouds. This work is largely orthogonal to ours, as it does not target generative point-based models.

3 Methodology

We assume an input point cloud $X = \{x_1, x_2, ..., x_k\}$ of size $k \in [1, k_{max}]$. It consists of points $x_i \in \mathbb{R}^d$, where d includes 3 spatial coordinates and optionally additional features. Our goal is to let the network $f_s(X)$ infer a function $\tilde{Y}$ which approximates a desired super-resolution output point cloud $Y = \{y_1, y_2, ..., y_n\}$ of size $n \in [1, n_{max}]$ with $y_i \in \mathbb{R}^3$, i.e. $f_s(X) = \tilde{Y} \approx Y$. For now we assume that the number of output points n is defined by multiplying k with a user-defined upsampling factor r, i.e. $n = rk$. Figure 2a) illustrates the data flow in our super-resolution network schematically. We treat the upsampling problem as a *local* one, i.e., we assume that the inference problem can be solved based on a spatially constrained neighborhood. This allows us to work with individual *patches* extracted from input point clouds. At the same time, it makes it possible to upsample adaptively, for example, by limiting the inference to relevant areas, such as complex surface structures. For the patch extraction we use a fixed spatial radius and normalize point coordinates within each patch to lie in the range of $[-1, 1]$.

Our first building block is a measure for how well two point clouds represent the same object or scene by formulating an adequate spatial loss function. Following Achlioptas et al. (2017), we base our spatial loss $\mathcal{L}_S$ on the *Earth Mover's Distance* (EMD), which solves an assignment problem to obtain a differentiable bijective mapping $\phi : \tilde{y} \to y$. With ϕ we can minimize differences in position for arbitrary orderings of the points clouds via:

$$\mathcal{L}_S = \min_{\phi:\tilde{y}\to y} \sum_{\tilde{y}_i \in \tilde{Y}} \|\tilde{y}_i - \phi(\tilde{y}_i)\|_2^2 \tag{1}$$

3.1 Temporal Coherence

When not taking temporal coherence explicitly into account, the highly nonlinear and ill-posed nature of the super-resolution problem can cause strong variations in the output even for very subtle changes in the input. This results in significant temporal artifacts that manifest themselves as flickering. In order to stabilize the output while at the same time keeping the network structure as small and simple as possible, we propose the following training setup. Given a sequence of high resolution point clouds Y^t, with t indicating time, we can compute a velocity $V^t = \{v_1^t, v_2^t, ..., v_k^t\}$, where $v_i^t \in \mathbb{R}^3$. For

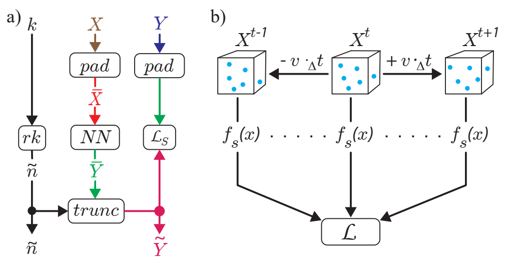


Figure 2: a) Schematic overview of $f_s(X)$. Black arrows represent scalar data. Point data is depicted as colored arrows with the color indicating data cardinality (brown=k, red = k_{max}, green = n_{max}, blue = n, and purple = $\tilde{n}$). b) Siamese network setup for temporal loss calculation.

this we use a finite difference $(y_i^{t+1} - y_i^t)$, where we assume, without loss of generality, $\Delta t = 1$, i.e. the time step is normalized to one. For training, the low resolution inputs X can now be generated from Y via down-sampling by a factor of r, which yields a subset of points with velocities. Details of our data generation process will be given below.

To train a temporally coherent network with the Y^t sequences, we employ a *Siamese* setup shown in Figure 2b. We evaluate the network several times (3 times in practice) with the same set of weights, and moving inputs, in order to enforce the output to behave consistently. In this way we avoid recurrent architectures that would have to process the high resolution outputs multiple times. In addition, we can compute temporal derivatives from the input points, and use them to control the behavior of the generated output.

Under the assumption of slowly moving inputs, which theoretically could be ensured for training, a straightforward way to enforce temporal coherence would be to minimize the movement of the generated positions over consecutive time steps in terms of an L_2 norm:

$$\mathcal{L}_{2V} = \sum_{i=1}^{n} \|\tilde{y}_i^{t+1} - \tilde{y}_i^t\|_2^2. \tag{2}$$

While this reduces flickering, it does not constrain the change of velocities, i.e., the acceleration. This results in a high frequency jittering of the generated point positions. The jitter can be reduced by also including the previous state at time step $t-1$ to constrain the acceleration in terms of its L_2 norm:

$$\mathcal{L}_{2A} = \sum_{i=1}^{n} \|\tilde{y}_i^{t+1} - 2\tilde{y}_i^t + \tilde{y}_i^{t-1}\|_2^2 \tag{3}$$

However, a central problem of a direct temporal constraint via Equations (2) and (3) is that it consistently leads to a highly undesirable clustering of generated points around the center point. This is caused by the fact that the training procedure as described so far is unbalanced, as it only encourages minimizing changes. The network cannot learn to reconstruct realistic, larger motions in this way, but rather can trivially minimize the loss by contracting all outputs to a single point. For this reason, we instead use the estimated velocity of the ground truth point cloud sequence with a forward difference in time, to provide the network with a reference. By using the EMD-based mapping ϕ established for the spatial loss in Equation (1), we can formulate the temporal constraint in a permutation invariant manner as

$$\mathcal{L}_{EV} = \sum_{i=1}^{n} \|\big(\tilde{y}_i^{t+1} - \tilde{y}_i^t\big) - \big(\phi(\tilde{y}_i^{t+1}) - \phi(\tilde{y}_i^t)\big)\|_2^2. \tag{4}$$

Intuitively, this means the generated outputs should mimic the motion of the closest ground truth points. As detailed for the L_2-based approaches above, it makes sense to also take the ground truth acceleration into account to minimize rapid changes of velocity over time. We can likewise formulate this in a permutation invariant way w.r.t. ground truth points via:

$$\mathcal{L}_{EA} = \sum_{i=1}^{n} \|\big(\tilde{y}_i^{t+1} - 2\tilde{y}_i^t + \tilde{y}_i^{t-1}\big) - \big(\phi(\tilde{y}_i^{t+1}) - 2\phi(\tilde{y}_i^t) + \phi(\tilde{y}_i^{t-1})\big)\|_2^2. \tag{5}$$

We found that a combination of $\mathcal{L}_{EV}$ and $\mathcal{L}_{EA}$ together with the spatial loss $\mathcal{L}_S$ from Eq. 1 provides the best results, as we will demonstrate below. First, we will introduce the additional loss terms of our algorithm.

3.2 Variable Point Cloud Sizes

Existing network architectures are typically designed for processing a fixed amount of input and output points. However, in many cases, and especially for a localized inference of super-resolution, the number of input and output points varies significantly. While we can safely assume that no patch exceeds the maximal number of inputs k_{max} (this can be ensured by working on a subset), it can easily happen that a certain spatial region has fewer points. Simply including more distant points could guarantee that we have a full set of samples, but this would mean the network has to be invariant to scaling, and to produce features at different spatial scales. Instead, we train our

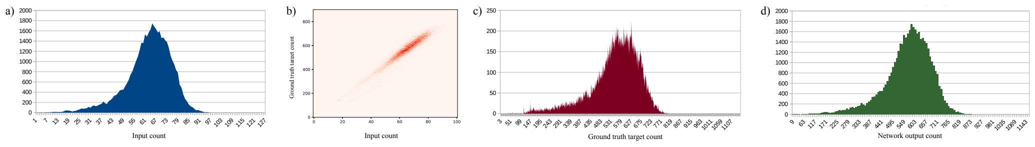


Figure 3: An illustration of the relationship between input and output size. (a,b,d) show histograms of point set sizes for: (a,b) the input set; (c) the ground truth target sets; and (d) the network output, i.e. r times larger than the input. The latter deviates from the ground truth in (c), but follows its overall structure. This is confirmed in (b), which shows a heat map visualization of input vs. ground truth output size. The diagonal structure of the peak confirms the approximately linear relationship.

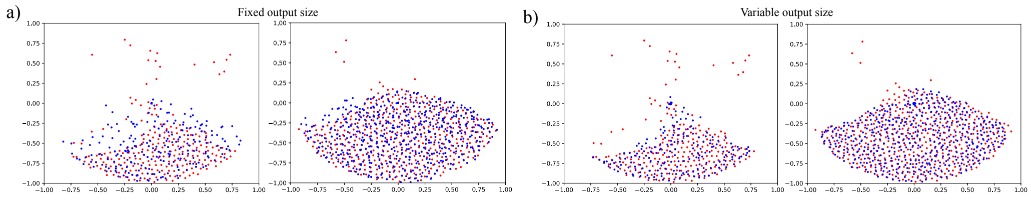


Figure 4: The effect of our variable output handling for exemplary patches. In red the ground truth target, in blue the inferred solution. Left (a) with fixed output size, and on the right (b) with the proposed support for variable output sizes. The latter approximates the shape of the red ground truth points significantly better. (a) leads to rather uniform shapes that, e.g., cover empty space above the ground truth in both examples.

network for a fixed spatial size, and ensure that it can process varying numbers of inputs. For inputs with fewer than k_{max} points, we pad the input vector to have a fixed size. Here, we ensure that the padding values are not misinterpreted by the network as being point data. Therefore, we pad X with $p \in \{-2\}^d$, which represents a value outside the regular patch coordinate range $[-1, 1]$: $\bar{X} = \{x_1, x_2, ..., x_k, \underbrace{p, p, ..., p}_{k_{max}-k}\}$. The first convolutional layer in our network now filters out the padded entries using the following mask: $M_{in} = \{m_{i\in[0,k]}\} = \{\underbrace{1, 1, ..., 1}_{k}, \underbrace{0, 0, .., 0}_{k_{max}-k}\}$. The entries of p allow us to compute the mask on the fly throughout the whole network, without having to pass through k. For an input of size k, our network has the task to generate $\tilde{n} = rk$ points. As the size of the network output is constant with rk_{max}, the outputs are likewise masked with M_{out} to truncate it to length $\tilde{n}$ for all loss calculations, e.g., the EMD mappings. Thus, as shown in Figure 2a, $\tilde{n}$ is used to truncate the point cloud $\bar{Y} = \{\bar{y}_1, \bar{y}_2, ..., \bar{y}_{n_{max}}\}$ via a mask M_{out} to form the final output $\tilde{Y} = \{\bar{y}_i | i \in [1, \tilde{n}]\}$.

Note that in Sec. 3.1, we have for simplicity assumed that $n = rk$, however, in practice the number of ground truth points n varies. As such, $\tilde{n}$ only provides an approximation of the true number of target points in the ground truth data. While the approximation is accurate for planar surfaces and volumes, it is less accurate in the presence of detailed surface structures that are smaller than the spatial frequency of the low-resolution data.

We have analyzed the effect of this approximation in Fig. 3. The histograms show that the strongly varying output counts are an important factor in practice, and Fig. 4 additionally shows the improvement in terms of target shape that results from incorporating variable output sizes. In general, $\tilde{n}$ provides a good approximation for our data sets. However, as there is a chance to infer an improved estimate of the correct output size based on the input points, we have experimented with training a second network to predict $\tilde{n}$ in conjunction with a differentiable output masking. While this could be an interesting feature for future applications, we have not found it to significantly improve results. As such, the evaluations and results below will use the analytic calculation, i.e., $\tilde{n} = rk$.

3.3 PREVENTING HALO ARTIFACTS

For each input point the network generates r output points, which can be seen as individual groups g: $\psi(g) = \{\tilde{Y}_i | i \in [rg + 1, (r + 1)g]\}$. These groups of size r in the output are strongly related to the input points they originate from. Networks that focus on maintaining temporal coherence for the dynamically changing output tend to slide into local minima where r output points are attached as a

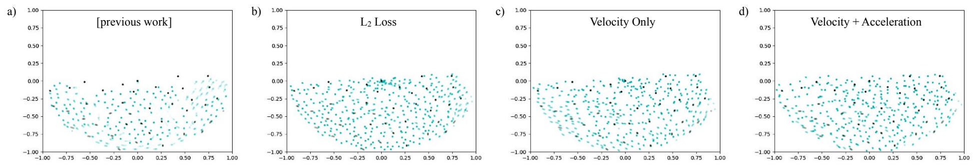


Figure 5: Ablation study for our temporal loss formulation. Black points indicate targets, while green points are generated (both shown as time average). a) Result from previous work; b) With L_{2V} loss; c) the proposed velocity loss L_{EV}; d) our full loss formulation with $L_{EV} + L_{EA}$. While (a) has difficulties approximating the target shape and the flickering output is visible as blurred positions, the additional loss terms (esp. in (c) and (d)) provide stable results that closely approximate the targets. Note that (b) leads to an undesirably static motion near the bottom of the patch. As the input points here are moving the output should mimic this motion, like (c,d).

fixed structure to the input point location. This manifests itself as visible static halo-like structures that move along with the input. Although temporal coherence is good in this case, these cluster-like structures lead to gaps and suboptimal point distributions in the output, particularly over time. These structures can be seen as a form of temporal mode collapse that can be observed in other areas of deep learning, such as GANs. To counteract this effect, we introduce an additional *mingling* loss term to prevent the formation of clusters by pushing the individual points of a group apart:

$$\mathcal{L}_M = \frac{1}{\lceil \frac{\tilde{n}}{r} \rceil} \sum_{i}^{\lceil \frac{\tilde{n}}{r} \rceil} \frac{|\psi(i)|}{\sum\limits_{\tilde{y}_g \in \psi(i)} \| \frac{\sum \psi(i)}{|\psi(i)|} - \tilde{y}_g \|_2} \tag{6}$$

Note that in contrast to previously used repulsion losses (Yu et al., 2018a), $\mathcal{L}_M$ encourages points to globally mix rather than just locally repelling each other. While a repulsion term can lead to a deterioration of the generated outputs, our formulation preserves spatial structure and temporal coherence while leading to well distributed points, as is illustrated in Fig. 6.

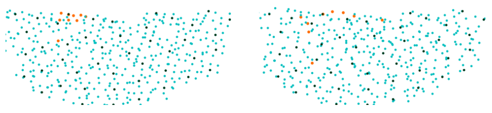


Figure 6: Left, a result without the mingling loss from Eq. 6, right with (a single point group highlighted in orange). The former has many repeated copies of a single pattern, which the mingling loss manages to distribute as can be seen in the right picture.

In combination with the spatial and temporal terms from above, this leads to our final loss function $\mathcal{L}_{final} = \mathcal{L}_S + \gamma\mathcal{L}_{EV} + \mu\mathcal{L}_{EA} + \nu\mathcal{L}_M$, with weighting terms γ, μ, ν.

4 Evaluation and Results

We train our network in a fully supervised manner with simulated data. To illustrate the effect of our temporal loss functions, we employ it in conjunction with established network architectures from previous work (Qi Charles et al., 2017; Yu et al., 2018a). Details of the data generation and network architectures are given in the appendix. We first discuss our data generation and training setup, then illustrate the effects of the different terms of our loss function, before showing results for more complex 3D data sets. As our results focus on temporal coherence, which is best seen in motion, we refer readers to the supplemental materials at `https://ge.in.tum.de/publications/2020-iclr-prantl/` in order to fully evaluate the resulting quality.

Ablation Study We evaluate the effectiveness of our loss formulation with a two dimensional ablation study. An exemplary patch of this study is shown in Fig. 5. In order to compare our method to previous work, we have trained a previously proposed method for point-cloud super-resolution, the PU-Net (Yu et al., 2018a) which internally uses a PointNet++ (Qi et al., 2017), with our data set, the only difference being that we use zero-padding here. This architecture will be used in the following comparisons with previous work. Fig. 5a) shows a result generated with this network. As this figure contains an average of multiple frames to indicate temporal stability, the blurred regions, esp. visible on the right side of Fig. 5a), indicate erroneous motions in the output. For this network the difficulties of temporally changing data and varying output sizes additionally lead to a suboptimal approximation

	$\mathcal{L}_S$	$\mathcal{L}_N$	$\mathcal{L}_M$	$\mathcal{L}_{2V}$	$\mathcal{L}_{2A}$	$\mathcal{L}_{EV}$	$\mathcal{L}_{EA}$
2D Previous work	0.0784	0.329	5.499	0.1	0.402	0.107	0.214
2D With $\mathcal{L}_{2V}$	0.044	0.00114	2.197	1.1e-05	4.2e-05	0.00197	0.00276
2D Only $\mathcal{L}_{EV}$	0.0453	0.00114	2.713	2.6e-05	6.0e-06	6.15e-04	5.27e-04
2D Full	0.0487	0.00116	3.0307	2.1e-05	1.0e-06	6.52e-04	1.46e-04
3D Previous work	0.0948	0.494	10.558	0.325	1.299	0.19	0.365
3D Full	0.0346	0.00406	3.848	8.04e-04	2.0e-06	0.00179	7.09e-04

Table 1: Quantitative results for the different terms of our loss functions, first for our 2D ablation study and then for our 3D versions. The first three columns contain spatial, the next four temporal metrics. $\mathcal{L}_N = \|\tilde{n} - n\|_2^2$ is given as a measure of accuracy in terms of the size of the generated outputs (it is not part of the training).

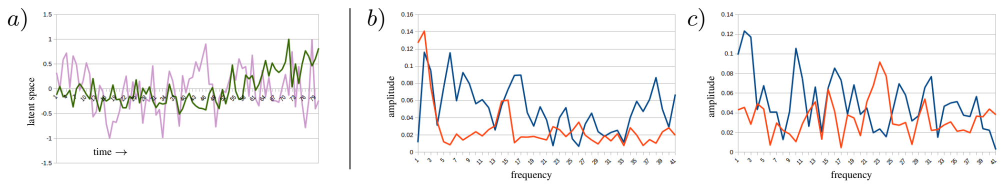


Figure 7: Illustrations of the latent spaces learned by our networks. (a) shows averaged latent space values for 100 random patch sequences of our 2D data set. The green curve shows our method with temporal coherence loss, while the pink curve was generated without it. The same data is shown in frequency space in (b), where the red curve represents the frequency of the data with temporal loss, and the blue curve the frequency of the data without. This graph highlights the reduced amount of high frequency changes in the latent space with temporal loss, esp. in frequency space, where the red curve almost entirely lies below the blue one. (c) contains frequency information for the latent space content of the same 100 patch sequences, but in a random order. In this case, the blue and red curve both contain significant amounts of high-frequencies. I.e., our method reliably identifies strongly changing inputs.

of the target points, that is also visible in terms of an increased $\mathcal{L}_S$ loss in Table 1. While Fig. 5b) significantly reduces motions, and leads to an improved shape as well as $\mathcal{L}_S$ loss, its motions are overly constrained. E.g., at the bottom of the shown patch, the generated points should follow the black input points, but in (b) the generated points stay in place. In addition, the lack of permutation invariance leads to an undesirable clustering of generated points in the patch center. Both problems are removed with $\mathcal{L}_{EV}$ in Fig. 5c), which still contains small scale jittering motions, unfortunately. These are removed by $\mathcal{L}_{EA}$ in Fig. 5d), which shows the result of our full algorithm. The success of our approach for dynamic output sizes is also shown in the $\mathcal{L}_N$ column of Table 1, which contains an L_2 error w.r.t. ground truth size of the outputs.

Temporally Coherent Features A central goal of our work is to enable the learning of features that remain stable over time. To shed light on how our approach influences the established latent space, we analyze its content for different inputs. The latent space in our case consists of a 256-dimensional vector that contains the features extracted by the first set of layers of our network. Fig. 7 contains a qualitative example for 100 randomly selected patch sequences from our test data set, where we collect input data by following the trajectory of each patch center for 50 time steps to extract coherent data sets. Fig. 7a) shows the averaged latent space content over time for these sequences. While the model trained with temporal coherence (green curve) is also visually smoother, the difference becomes clearer when considering temporal frequencies. We measure averaged frequencies of the latent space dimensions over time, as shown in Fig. 7b,c). We quantify the differences by calculating the integral of the frequency spectrum $\tilde{f}$, weighted by the frequency x to emphasize high frequencies, i.e, $\int_x x \cdot \tilde{f}(x)dx$. Hence, small values are preferred. As shown in Fig. 7b), the version trained without our loss formulations contains significantly more high frequency content. This is also reflected in the weighted integrals, which are 36.56 for the method without temporal loss, and 16.98 for the method with temporal loss. To verify that our temporal model actually establishes a stable temporal latent space instead of ignoring temporal information altogether, we evaluate the temporal frequencies for the same 100 inputs as above, but with a randomized order over time. In this case, our model correctly identifies the incoherent inputs, and yields similarly high frequencies as the regular model with 28.44 and 35.24, respectively. More details in Appendix C.

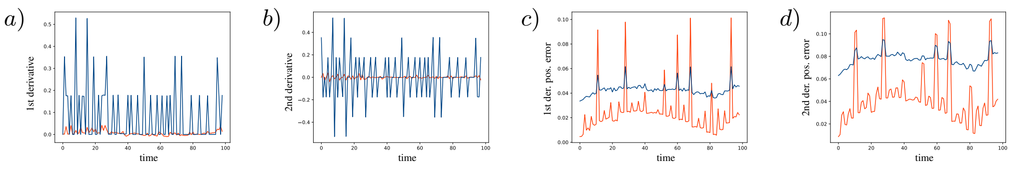


Figure 8: Evaluation of the temporal stability for generated point clouds, in red with our temporal loss formulation, in blue without. Graph (a) shows the temporal change of the point density (1st derivative), while (b) shows the 2nd derivative. In (c) and (d) the error of the 1st and 2nd derivatives of the positions w.r.t. ground-truth reference points is shown.

In addition, we evaluated the changes of generated outputs over time w.r.t. ground truth motion. For this we mapped the generated point clouds $\tilde{Y}^t = \{\tilde{y}^t_1, \tilde{y}^t_2, ..., \tilde{y}^t_{\tilde{n}}\}$ for 100 frames to evenly and dense sampled ground-truth points on the original mesh $Y^t = \{y^t_1, y^t_2, ..., y^t_n\}$ (the moving man shown in Fig. 1). This gives us a dense correlation between the data and the generated point clouds. For the mapping we used an assignment based on nearest neighbors: $\gamma : \tilde{y} \rightarrow y$. Using γ we divide $\tilde{Y}^t$ into n subsets $\hat{Y}_i = \{\tilde{y}_j | \gamma(\tilde{y}_j) = y_i\}$ which correlate with the corresponding ground-truth points. For each subset we can now compute the mean position $\frac{1}{|\hat{Y}_i|} \sum_{\hat{y} \in \hat{Y}_i} \hat{y}$ and the sample density $|\hat{Y}_i|$ measured by the number of points assigned to a ground-truth sample position. The temporal change of these values are of particular interest. The change of the mean positions should correspond to the ground-truth changes, while the change of the density should be one. We have evaluated the error of the first and second derivative of positions, as well as the first and second derivative of density (see Fig. 8 and Table 2). As can be seen from the plots, our method leads to clear improvements for all measured quantities. The individual spikes that are visible for both versions in the position errors (c,d) most likely correspond to sudden changes of the input motions for which our networks undershoots by producing a smooth version of the motion.

	w/o	with
Velocity	0.043	0.024
Acceleration	0.078	0.043

	w/o	with
Variance of 1st Derivative	0.016	0.00013
Variance of 2nd Derivative	0.038	0.00017

Table 2: Measurements averaged over 100 frames for a version of our network without temporal loss ("w/o") and with our full temporal loss formulation ("with"). The left table shows the results for the error evaluation of the velocity and the acceleration, whereas in the right table one can see the variance of the density derivatives.

3D Results Our patch-based approach currently relies on a decomposition of the input volumes into patches over time, as outlined in Appendix A. As all of the following results involve temporal data, full sequences are provided in the accompanying video. We apply our method to several complex 3D models to illustrate its performance. Fig. 9 shows the input as well as several frames generated with our method for an animation of a spider. Our method produces an even and temporally stable reconstruction of the object. In comparison, Fig. 9b) shows the output from the previous work architecture (Yu et al., 2018a). It exhibits uneven point distributions and outliers, e.g., above the legs of the spider, in addition to uneven motions.

A second example for a moving human figure is shown in Fig. 1. In both examples, our network covers the target shape much more evenly despite using much fewer points, as shown in Table 3. Thanks to the flexible output size of our network, it can adapt to sparsely covered regions by generating correspondingly fewer outputs. The previous work architecture, with its fixed output size, needs to concentrate the fixed number of output points within the target shape, leading to an unnecessarily large point count. In order to demonstrate the flexibility of our method, we also apply it to a volumetric moving point cloud obtained from a liquid simulation. Thanks to the patch-based evaluation of our network, it is agnostic to the overall size of the input volume. In this way, it can be used to generate coherent sets with millions of points. These examples also highlight our method's capabilities for generalization. While the 3D model was only trained on data from physics simulations, as outlined above, it learns stable features that can be flexibly applied to volumetric as well as to surface-based data. The metrics in Table 1 show that for both 2D and 3D cases, our method leads to significantly improved quality, visible in lower loss values for spatial as well as temporal terms.

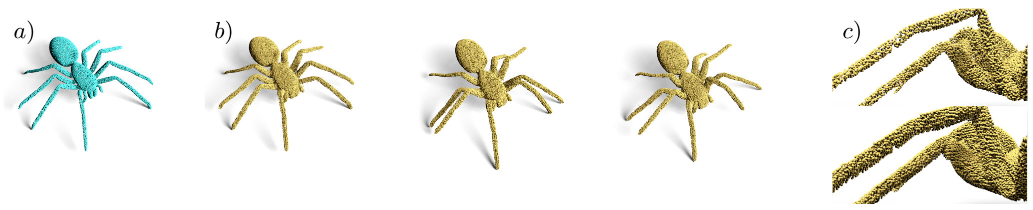


Figure 9: Our method applied to an animation of a moving spider. (a) Input point cloud, (b) three frames of our method, (c) a detail from previous work (top) and our method (bottom). Note that our method at the bottom preserves the shape with fewer outliers, and leads to a more even distribution of points, despite generating fewer points in total (see Table 3).

	Input points	P.W., output points	P.W., factor	Ours, output points	Ours, factor
Spider	7,900	3,063,704	387.81	251,146	31.79
Moving person	10,243	5,224,536	510.06	367,385	35.87
Liquid	513,247	-	-	6,430,984	12.53

Table 3: Point counts for the 3D examples of our video. Input counts together with output counts for previous work (P.W.) and our proposed network are shown. Factor columns contain increase in point set size from in- to output. As previous work cannot handle flexible output counts, a fixed number of points is generated per patch, leading to a huge number of redundant points. However, our network flexibly adapts the output size and leads to a significantly smaller number of generated points that cover the object or volume more evenly.

Another interesting field of application for our algorithm are physical simulations. Complex simulations such as fluids, often employ particle-based representations. On the one hand, the volume data is much larger than surface-based data, which additionally motivates our dynamic output. On the other hand, time stability plays a very important role for physical phenomena. Our method produces detailed outputs for liquids, as can be seen in our supplemental video.

Convergence graphs for the different versions are shown in Fig. 12 of the supplemental material. These graphs show that our method not only successfully leads to very low errors in terms of temporal coherence, but also improves spatial accuracy. The final values of $\mathcal{L}_S$ for the 2D case are below 0.05 for our algorithm, compared to almost 0.08 for previous work. For 3D, our approach yields 0.04 on average, in contrast to ca. 0.1 for previous work.

5 Conclusion

We have proposed a first method to infer temporally coherent features for point clouds. This is made possible by a novel loss function for temporal coherence in combination with enabling flexible truncation of the results. In addition we have shown that it is crucial to prevent static patterns as easy-to-reach local minima for the network, which we avoid with the proposed a mingling loss term. Our super-resolution results above demonstrate that our approach takes an important first step towards flexible deep learning methods for dynamic point clouds.

Looking ahead, our method could also be flexibly combined with other network architectures or could be adopted for other applications. Specifically, a combination with PSGN (Fan et al., 2016) could be used to generate point clouds from image sequences instead of single images. Other conceivable applications could employ methods like Dahnert et al. (2019) with our approach for generating animated meshes. Due to the growing popularity and ubiquity of scanning devices it will, e.g., be interesting to investigate classification tasks of 3D scans over time as future work. Apart from that, physical phenomena such as elastic bodies and fluids (Li et al., 2019) can likewise be represented in a Lagrangian manner, and pose interesting challenges and complex spatio-temporal changes.

Acknowledgments

This work is supported by grant TH 2034/1-1 of the Deutsche Forschungsgemeinschaft (DFG).

REFERENCES

R Qi Charles, Hao Su, Mo Kaichun, and Leonidas Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. pages 77–85, 07 2017. doi: 10.1109/CVPR.2017.16.

Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In *Advances in Neural Information Processing Systems*, pages 5099–5108. Curran Associates, Inc., 2017.

Pedro Hermosilla, Tobias Ritschel, Pere-Pau Vázquez, Àlvar Vinacua, and Timo Ropinski. Monte carlo convolution for learning on non-uniformly sampled point clouds. *ACM Trans. Graph.*, 37(6): 235:1–235:12, December 2018. doi: 10.1145/3272127.3275110.

Binh-Son Hua, Minh-Khoi Tran, and Sai-Kit Yeung. Pointwise convolutional neural networks. In *Computer Vision and Pattern Recognition (CVPR)*, 2018.

Lequan Yu, Xianzhi Li, Chi-Wing Fu, Daniel Cohen-Or, and Pheng-Ann Heng. Pu-net: Point cloud upsampling network. In *CVPR*, 2018a.

Matan Atzmon, Haggai Maron, and Yaron Lipman. Point convolutional neural networks by extension operators. *ACM Trans. Graph.*, 37(4):71:1–71:12, July 2018. doi: 10.1145/3197517.3201301.

Lequan Yu, Xianzhi Li, Chi-Wing Fu, Daniel Cohen-Or, and Pheng-Ann Heng. Ec-net: an edge-aware point set consolidation network. In *European Conference on Computer Vision*, pages 398–414. Springer, 2018b.

Siamak Ravanbakhsh, Jeff G. Schneider, and Barnabás Póczos. Deep learning with sets and point clouds. *CoRR*, abs/1611.04500, 2016.

Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabás Póczos, Ruslan R. Salakhutdinov, and Alexander J. Smola. Deep sets. In *Advances in Neural Information Processing Systems*, pages 3394–3404, 2017.

Roman Klokov and Victor S. Lempitsky. Escape from cells: Deep kd-networks for the recognition of 3d point cloud models. In *IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, October 22-29, 2017*, pages 863–872, 2017. doi: 10.1109/ICCV.2017.99.

Riccardo Roveri, A. Cengiz Öztireli, Ioana Pandele, and Markus Gross. Pointpronets: Consolidation of point clouds with convolutional neural networks. *Computer Graphics Forum*, 37(2):87–99, 2018. doi: 10.1111/cgf.13344.

Paul Guerrero, Yanir Kleiman, Maks Ovsjanikov, and Niloy J. Mitra. PCPNet: Learning local shape properties from raw point clouds. *Computer Graphics Forum*, 37(2):75–85, 2018. doi: 10.1111/cgf.13343.

Kangxue Yin, Hui Huang, Daniel Cohen-Or, and Hao Zhang. P2p-net: Bidirectional point displacement net for shape transform. *ACM Trans. Graph.*, 37(4):152:1–152:13, July 2018. doi: 10.1145/3197517.3201288.

Hang Su, Varun Jampani, Deqing Sun, Subhransu Maji, Evangelos Kalogerakis, Ming-Hsuan Yang, and Jan Kautz. Splatnet: Sparse lattice networks for point cloud processing. In *CVPR*, pages 2530–2539. IEEE Computer Society, 2018.

Weiyue Wang, Ronald Yu, Qiangui Huang, and Ulrich Neumann. Sgpn: Similarity group proposal network for 3d point cloud instance segmentation. pages 2569–2578, 06 2018a. doi: 10.1109/CVPR.2018.00272.

Yifan Xu, Tianqi Fan, Mingye Xu, Long Zeng, and Yu Qiao. Spidercnn: Deep learning on point sets with parameterized convolutional filters. In *ECCV (8)*, volume 11212 of *Lecture Notes in Computer Science*, pages 90–105. Springer, 2018.

Wenxuan Wu, Zhongang Qi, and Fuxin Li. Pointconv: Deep convolutional networks on 3d point clouds. *CoRR*, abs/1811.07246, 2018.

Jiaxin Li, Ben M. Chen, and Gim Hee Lee. So-net: Self-organizing network for point cloud analysis. In *CVPR*, 2018a.

Xiaoqing Ye, Jiamao Li, Hexiao Huang, Liang Du, and Xiaolin Zhang. 3d recurrent neural networks with context fusion for point cloud semantic segmentation. In *ECCV*, 2018.

F. Groh, P. Wieschollek, and H. P. A. Lensch. Flex-convolution (million-scale point-cloud learning beyond grid-worlds). In *Computer Vision - ACCV 2018 - 14th Asian Conference on Computer Vision*, December 2018.

Loïc Landrieu and Martin Simonovsky. Large-scale point cloud semantic segmentation with superpoint graphs. In *CVPR*, 2018.

Dario Rethage, Johanna Wald, Jürgen Sturm, Nassir Navab, and Federico Tombari. Fully-convolutional point networks for large-scale point clouds. In *Computer Vision – ECCV 2018*, pages 625–640. Springer International Publishing, 2018.

Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E. Sarma, Michael M. Bronstein, and Justin M. Solomon. Dynamic graph cnn for learning on point clouds. *CoRR*, abs/1801.07829, 2018b.

Huan Lei, Naveed Akhtar, and Ajmal Mian. Spherical convolutional neural network for 3d point clouds. *CoRR*, abs/1805.07872, 2018.

Yingxue Zhang and Michael G. Rabbat. A graph-cnn for 3d point cloud classification. *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6279–6283, 2018.

Mark B. Skouson. Ursa: A neural network for unordered point clouds using constellations. *CoRR*, abs/1808.04848, 2018.

Martin Simon, Stefan Milz, Karl Amende, and Horst-Michael Groß. Complex-yolo: Real-time 3d object detection on point clouds. *CoRR*, abs/1803.06199, 2018.

Yin Zhou and Oncel Tuzel. Voxelnet: End-to-end learning for point cloud based 3d object detection. In *CVPR*, 2018.

Liuhao Ge, Yujun Cai, Junwu Weng, and Junsong Yuan. Hand pointnet: 3d hand pose estimation using point sets. In *CVPR*, 2018.

Nathaniel Thomas, Tess Smidt, Steven M. Kearnes, Lusann Yang, Li Li, Kai Kohlhoff, and Patrick Riley. Tensor field networks: Rotation- and translation-equivariant neural networks for 3d point clouds. *CoRR*, abs/1802.08219, 2018.

Yaoqing Yang, Chen Feng, Yiru Shen, and Dong Tian. Foldingnet: Point cloud auto-encoder via deep grid deformation. In *CVPR*, 2018.

Haowen Deng, Tolga Birdal, and Slobodan Ilic. Ppfnet: Global context aware local features for robust 3d point matching. In *CVPR*, pages 195–205. IEEE Computer Society, 2018.

Yongbin Sun, Yue Wang, Ziwei Liu, Joshua E. Siegel, and Sanjay E. Sarma. Pointgrow: Autoregressively learned point cloud generation with self-attention. *CoRR*, abs/1810.05591, 2018.

Chun-Liang Li, Manzil Zaheer, Yonghui Zhang, Barnabás Póczos, and Ruslan Salakhutdinov. Point cloud gan. *CoRR*, abs/1810.05795, 2018b.

Xingyu Liu, Charles Ruizhongtai Qi, and Leonidas J. Guibas. Learning scene flow in 3d point clouds. *CoRR*, abs/1806.01411, 2018.

Panos Achlioptas, Olga Diamanti, Ioannis Mitliagkas, and Leonidas J. Guibas. Representation learning and adversarial generation of 3d point clouds. *CoRR*, abs/1707.02392, 2017.

Haoqiang Fan, Hao Su, and Leonidas J. Guibas. A point set generation network for 3d object reconstruction from a single image. *CoRR*, abs/1612.00603, 2016. URL `http://arxiv.org/abs/1612.00603`.

Manuel Dahnert, Angela Dai, Leonidas Guibas, and Matthias Nießner. Joint embedding of 3d scan and cad objects. In *ICCV 2019*, 2019.

Yunzhu Li, Jiajun Wu, Russ Tedrake, Joshua B Tenenbaum, and Antonio Torralba. Learning particle dynamics for manipulating rigid bodies, deformable objects, and fluids. In *ICLR*, 2019.

Markus Ihmsen, Jens Cornelis, Barbara Solenthaler, Christopher Horvath, and Matthias Teschner. Implicit incompressible sph. *IEEE Transactions on Visualization and Computer Graphics*, 20(3): 426–435, 2014.

Tranquil Clouds: Neural Networks for Learning Temporally Coherent Features in Point Clouds, *Supplemental Material*

A Training and Evaluation Modalities

Data Generation We employ a physical simulation to generate our input and output pairs for training. This has the advantage that it leads to a large variety of complex motions, and gives full control of the generation process. More specifically, we employ the IISPH (Ihmsen et al., 2014) algorithm, a form of Lagrangian fluid simulator that efficiently generates incompressible liquid volumes. These simulations also have the advantage that they inherently control the density of the point sampling thanks to their volume conserving properties. In order to generate input pairs for training, we randomly sample regions near the surface and extract points with a given radius around a central point. This represents the high-resolution target. To compute the low-resolution input, we downsample the points with a Poisson-disk sampling to compute a point set with the desired larger spacing. In order to prevent aliasing from features below the coarse resolution, we perform a pass of surface fairing and smoothing before downsampling. Due to the large number of patches that can be extracted from these simulations, we did not find it necessary to additionally augment the generated data sets. Examples of the low- and high-resolution pairs are shown in the supplemental material.

Below we will demonstrate that models trained with this data can be flexibly applied to moving surface data as well as new liquid configurations. The surface data is generated from animated triangle meshes that were resampled with bicubic interpolation in order to match a chosen average per-point area. This pattern was generated once and then propagated over time with the animation. When applying our method to new liquid simulations, we do not perform any downsampling, but rather use all points of a low-resolution simulation directly, as a volumetric re-sampling over time is typically error prone, and gives incoherent low resolution inputs.

Given a moving point cloud, we decompose it into temporally coherent patches in the following manner: We start by sampling points via a Poisson-disk sampling in a narrow band around the surface, e.g., based on a signed distance function computed from the input cloud. These points will persist as patch centers over time, unless they move too close to others, or too far away from the surface, which triggers their deletion. In addition, we perform several iterations for every new frame to sample new patches for points in the input cloud that are outside all existing patches. Note that this resampling of patches over time happens instantaneously in our implementation. While a temporal fading could easily be added, we have opted for employing transitions without fading, in order to show as much of the patch content as possible.

Network Architecture and Training Our architecture heavily relies on a form of hierarchical point-based convolutions. I.e., the network extracts features for a subset of the points and their nearest neighbors. For the point convolution, we first select a given number of group centers that are evenly distributed points from a given input cloud. For each group center, we then search for a certain number of points within a chosen radius (a fraction of the [-1,1] range). This motivates our choice for a coordinate far outside the regular range for the padded points from Sec. 3.2. They are too far away from all groups by construction, so they are filtered out without any additional overhead. In this way, both feature extraction and grouping operations work flexibly with the varying input sizes. Each group is then processed by a PointNet-like sub-structure (Qi Charles et al., 2017), yielding one feature vector per group.

The result is a set of feature vectors and the associated group position, which can be interpreted as a new point cloud to repeatedly apply a point convolution. In this way, the network extracts increasingly abstract and global features. The last set of features is then interpolated back to the original points of the input. Afterwards a sub-pixel convolution layer is used to scale up the point cloud extended with features and finally the final position vectors are generated with the help of two additional shared, fully-connected layers. While we keep the core network architecture unmodified to allow for comparisons with previous work, an important distinction of our approach is the input and output masking, as described in Sec. 3.2.

Our point data was generated with a mean point spacing, i.e., Poisson disk radius, of 0.5 units. For the 2D tests, an upscaling factor of $r = 9$ was used. For this purpose, patches with a diameter of 5 were extracted from the low-resolution data and patches with a diameter of 15 from the high-resolution data. We used the thresholds $k_{max} = 100$ and $n_{max} = 900$. For the loss, we used $\gamma = 10$, $\mu = 10$, and $\nu = 0.001$. The network was trained with 5 epochs for a data set with 185k pairs, and a batch size of 16, the learning rate was 0.001 with a decay of 0.003. For the 3D results below, the scaling factor r was set to 8. The diameter of the patches was 6 for the low-resolution data and 12 for the high-resolution data, with $k_{max} = 1280$ and $n_{max} = 10240$. The loss parameters were $\gamma = \mu = 5$, with $\nu = 0.001$. Learning rate and decay were the same for training, but instead we used 10 epochs with 54k patches in 3D, and a batch size of 4.

B NETWORK ARCHITECTURE DETAILS

The input feature vector is processed in the first part of our network, which consists of four point convolutions. We use $(n_g, r_g, [l_1, ..., l_d])$ to represent a level with n_g groups of radius r_g and $[l_1, ..., l_d]$ the d fully-connected layers with the width $l_i (i = 1, ..., d)$. The parameters we use are $(k_{max}, 0.25, [32, 32, 64])$, $(k_{max}/2, 0.5, [64, 64, 128])$, $(k_{max}/4, 0.6, [128, 128, 256])$ and $(k_{max}/8, 0.7, [256, 256, 512])$. We then use interpolation layers to distribute the features of each convolution level among the input points. In this step, we reduce the output of each convolution layer with one shared, fully-connected layer per level, to a size of 64 and then distribute the features to all points of the input point cloud depending on their position. This extends the points of our original point cloud with 256 features. Fig. 11 shows a visual overview of the data flow in our network.

Afterwards, we process the data in r separate branches consisting of two shared, fully interconnected layers with 256 and 128 nodes. The output is then processed with two shared fully-connected layers of 64 and 3 nodes. Finally, we add our resulting data to the input positions that have been repeated r times. This provides an additional skip connection which leads to slightly more stable results. All convolution layers and fully interconnected layers use a tanh() activation function.

For the input feature vector, we make use of additional data fields in conjunction with the point positions. Our network also accepts additional features such as velocity, density and pressure of the SPH simulations used for data generation. For inputs from other sources, those values could be easily computed with suitable SPH interpolation kernels. In practice, we use position, velocity and pressure fields. Whereas the first two are important (as mentioned in Sec. 3.1), the pressure fields turned out to have negligible influence.

C FREQUENCY EVALUATION OF LATENT SPACE

In this section we give details for the frequency evaluation of Sec. 4. In order to measure the stability of the latent space against temporal changes, we evaluated the latent space of our network with and without temporal loss, once for 100 ordered patch sequences and once for 100 un-ordered ones. The central latent space of our network consists of the features generated by the point-convolution layers in the first part of the network and is 256 dimensional (see Fig. 11). To obtain information about its general behavior, we average the latent space components over all 100 patch sequences, subtract the mean, and normalize the resulting vector w.r.t. maximum value for each data set. The result is a time sequence of scalar values representing the mean deviations of the latent space. The Fourier transform of these vectors $\tilde{f}$, are shown in Fig. 7, and were used to compute the weighted frequency content $\int_x x \cdot \tilde{f}(x)dx$. Here, large values indicate strong temporal changes of the latent space dimensions. The resulting values are given in the main document, and highlight the stability of the latent space learned by our method.

D TRAINING DATA AND GRAPHS

Two examples with ground truth points and down-sampled input versions are shown in Fig. 10.

Additionally, Fig. 12 shows loss graphs for the different versions shown in the main text: 2D previous work, our full algorithm in 2D, as well as both cases for 3D. The mingling loss $\mathcal{L}_M$ is only shown as reference for the previous work versions, but indicates the strong halo-like patterns forming for the architectures based on previous work.

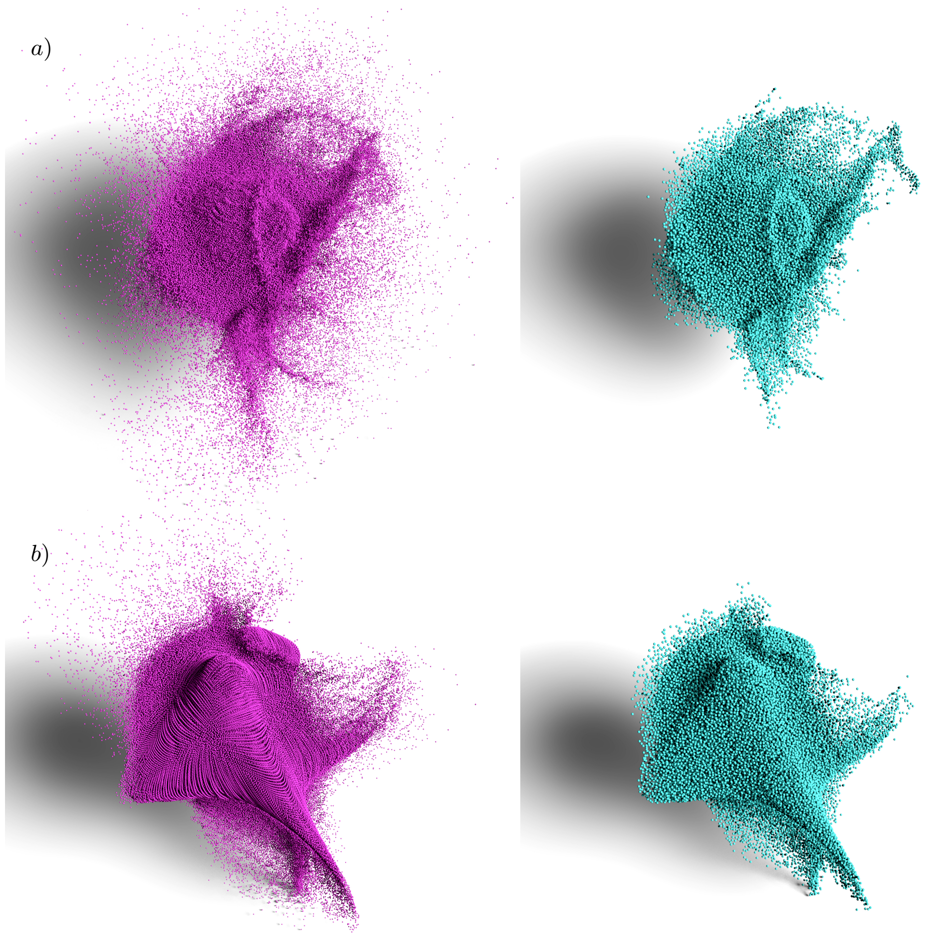


Figure 10: Examples from our synthetic data generation process. In both sections (a) and (b) a high resolution reference frame is shown in purple, and in green the down-sampled low resolution frames generated from it. The training data is generated by sampled patches from these volumes.

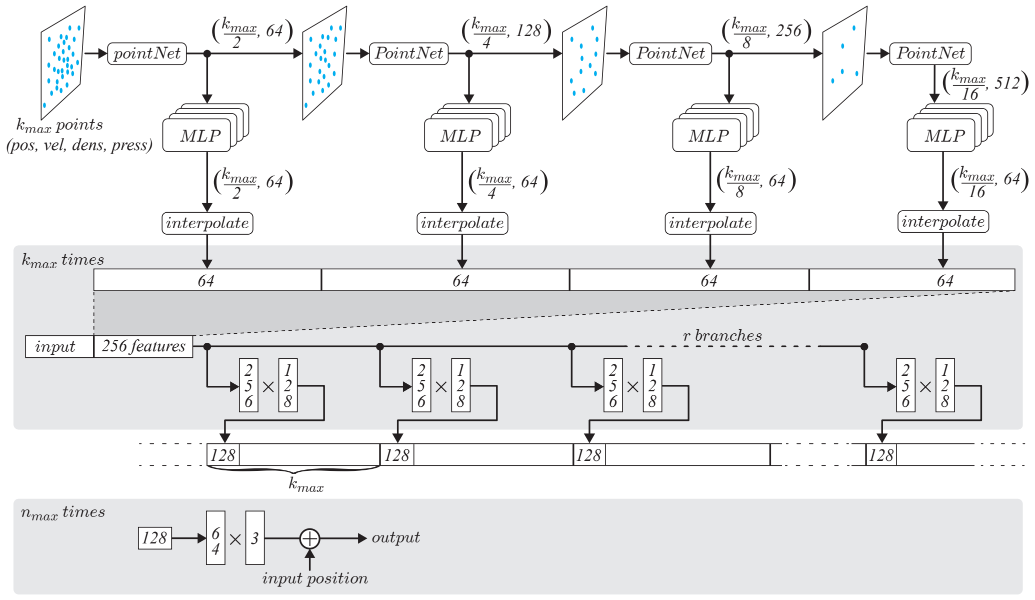

k_{max} points
(pos, vel, dens, press)
pointNet
PointNet
$(\frac{k_{max}}{2}, 64)$
$(\frac{k_{max}}{4}, 128)$
$(\frac{k_{max}}{8}, 256)$
$(\frac{k_{max}}{16}, 512)$
MLP
$(\frac{k_{max}}{4}, 64)$
$(\frac{k_{max}}{8}, 64)$
$(\frac{k_{max}}{16}, 64)$
interpolate
k_{max} times
64
input
256 features
r branches
256 × 128
128
k_{max}
n_{max} times
64 × 3
output
input position

Figure 11: An overview of our network architecture. The first row shows the hierarchical point convolutions, while the bottom rows illustrate the processing of extracted features until the final output point coordinates are generated.

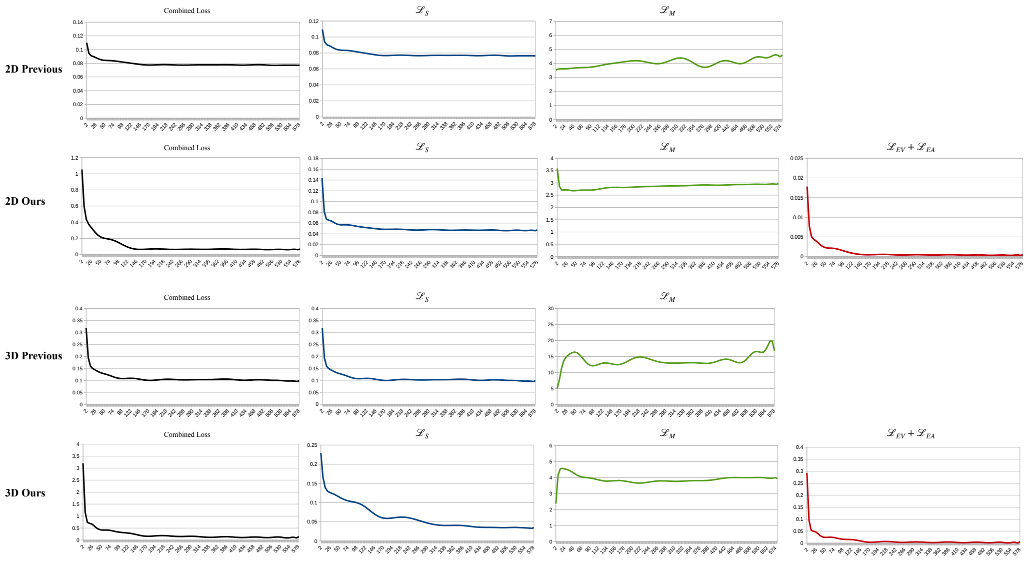


Figure 12: Convergence plots for the training runs of our different 2D and 3D versions. The combined loss only illustrates convergence behavior for each method separately, as weights and terms differ across the four variants. $\mathcal{L}_M$ for previous work is not minimized, and only given for reference.