SURFSUP: Learning Fluid Simulation for Novel Surfaces

Arjun Mani*[1], Ishaan Preetam Chandratreya*[1], Elliot Creager[2 3], Carl Vondrick[1], Richard Zemel[1]
[1] Columbia University [2] University of Toronto [3] Vector Institute
surfsup.cs.columbia.edu

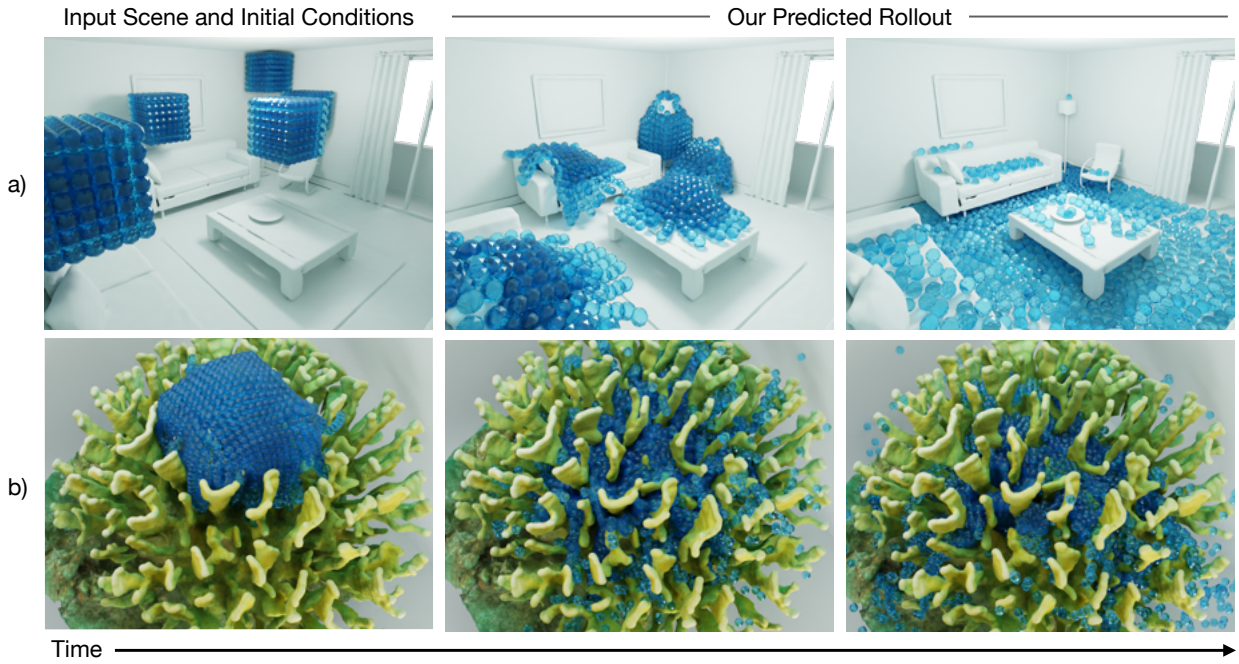


Figure 1: We introduce SURFSUP, which learns to predict how fluid particles interact with novel, complex 3D surfaces.

Abstract

Modeling the mechanics of fluid in complex scenes is vital to applications in design, graphics, and robotics. Learning-based methods provide fast and differentiable fluid simulators, however most prior work is unable to accurately model how fluids interact with genuinely novel surfaces not seen during training. We introduce SURFSUP*, a framework that represents objects* implicitly *using signed distance functions (SDFs), rather than an explicit representation of meshes or particles. This continuous representation of geometry enables more accurate simulation of fluid-object interactions over long time periods while simultaneously making computation more efficient. Moreover,* SURFSUP *trained on simple shape primitives generalizes considerably out-of-distribution, even to complex real-world scenes and objects. Finally, we show we can invert our model to design simple objects to manipulate fluid flow.*

*Authors contributed equally.

1. Introduction

Across engineering and science, the simulation of fluid dynamics has become an invaluable tool, and it will be a crucial component for building visual systems that are capable of understanding and interacting with real-world environments. Recently, a new class of simulators has emerged that *learn* the dynamics of physical systems from data [13, 31, 25]. These simulators offer several advantages over classical simulators by increasing speed, reducing reliance on hand-crafted dynamics models, and providing differentiable rollouts for solving inverse problems. Recent approaches based on graph neural networks (GNNs) have in particular shown impressive accuracy and generalization on a wide range of fluids represented as particle or mesh-based systems [25, 23].

In a wide variety of applications areas, fluid simulation must properly handle *interaction with solid structures* to attain practical relevance. Many problems involving

fluid simulation are fundamentally about object design; for example, designing aerodynamic shapes for cars and airplanes. In robotics too, an intelligent robot would have to understand fluid-surface interactions; for example, to carry a mug of hot coffee to a person without spilling the liquid from the container.

In both cases, the need to accurately model how fluids interact with *novel* solid surfaces[1]— with new shapes, configurations, orientations, and so on—motivates an challenging out-of-distribution (OOD) generalization subproblem for fluid simulation.

In this paper, we tackle this problem by introducing **Surf**ace implicit **s**U**b**stitution for **p**articles (SURFSUP), a framework for simulating the interactions of fluids and surfaces by representing surfaces using *implicit representations* [19]. Current GNN-based approaches represent surfaces explicitly, discretizing them into particles and augmenting the graph accordingly [25]. Implicit representations, in particular signed distance functions which represent surfaces as zero-level sets of functions, offer several compelling advantages. They can smoothly represent continuous surfaces, provide rich geometric information (e.g. whether a point is in the interior/exterior of the object), and can scale easily to large objects and scenes. Due to these properties, many state-of-the-art generative models for 3D shapes learn shapes with signed distance functions. We argue that these properties are useful not just for statics but also *dynamics*: in order to accurately model fluid-surface interactions for potentially complex surfaces, smooth and informative representations of *local* surface geometry are needed. In turn, we posit that these locally informative representations will enable fluid simulators to appropriately model interaction with never-before-seen surfaces.

Concretely, we propose an approach that integrates signed distance representations of surfaces [22, 19] with graph neural network models [26, 25]. Our architecture follows the encode-process-decode paradigm [25], where the particles in the physical system at time T are encoded into a graph, followed by several message-passing steps, and finally a decoder predicts the dynamics (accelerations). We represent the surface using its signed distance function, and exploit geometric properties of SDFs to model the distance and orientation of fluid particles with respect to the surface.

Experiments show that our approach can model fluid-surface interactions over long rollouts with a high degree of accuracy and visual realism. SURFSUP can model surfaces far beyond its training domain; when trained on simple primitive shapes (e.g. spheres, cones, toruses), we generalize *zero-shot* to complex real-world objects and scenes represented using neural implicit generative models (Fig. 1). We compare to a method which represents the surface as particles inside the graph network simulator, and show that our model achieves better generalization and efficiency, and can better handle complex surface geometry. Finally, we demonstrate the potential of SURFSUP to solve inverse problems across two object design tasks. Overall, our paper bridges learned physical models for dynamics and 3D implicit representations, and takes a step towards AI models which can understand and help design the physical world.

[1]For brevity we refer to surfaces of solid objects simply as "surfaces".

2. Related Work

Learning Differentiable Simulators. Using machine learning for predicting and modeling complex physical systems is a rapidly growing area of research. Learned physics simulators have been shown to accurately simulate colliding rigid objects, deformable objects, fluids, and other complex systems [13, 3, 12, 31, 25, 21].

Recent progress in learned fluid simulation has been driven by the Lagrangian viewpoint, which tracks the motion of fluid particles over time. Graph neural networks (GNNs) have in particular emerged as effective forward models for fluid simulation, and have demonstrated more stability and generalization capability than previous methods due to useful inductive biases [13, 14, 27, 25]. Sanchez-Gonzalez et al. [25] introduced a GNN architecture which can accurately and generalizably model fluid behavior over long rollouts; most subsequent work builds off their approach. However, most of these works generally consider simple surfaces; *e.g.* only cube-shaped containers for fluids. There has been little work focused on fluid-surface interactions; Mayr et al. [16] introduce an approach specifically for triangularized meshes. All the above approaches use particle-based representations of surfaces; we will argue that implicit representations allow learning of more accurate, generalizable physical dynamics.

Some works have explored the possibility of using learned fluid simulators for solving inverse problems [27, 12, 2]. Allen et al. [2] showed that graph network simulators can be used to solve design tasks by rolling out fluid trajectories and using gradient-based optimization. In 3D, they show that a 2D plane with a learned height field can be used to split a stream of fluid to fall into several locations on the ground. SDFs have the advantage of being able to learn geometry in a more smooth and unconstrained way, which could help solve inverse problems.

3D Implicit Representations. Implicit representations have become state-of-the-art for generative models of 3D objects [17, 22, 6, 29, 30]. These models represent the surface as a function $F(x)$. There are several choices for F. Park et al. [22] show that signed distance functions $F(x)$ are particularly effective and informative representations. Compared to point clouds [1, 28] or voxels [7, 33], models based on SDF representations can smoothly represent complex geometry, are compact and efficient, and can be learned effectively [29, 6]. Follow-up work has also stud-

ied learning "dual" generative models that jointly learn an uncontrained representation and one composed of primitive shapes [10, 32].

Implicit Representations for Physical Dynamics. Some classical particle-based fluid simulators based on the Smoothed Particle Hydrodynamics (SPH) formalism [20] have transitioned from particle-based to implicit surface representations. [11, 9, 4]. In these simulators implicit surfaces allow for smoother fluid motion with less artifacts [11]. For general-purpose learned simulators (GNNs), there is little prior work using implicit rather than particle-based representations of surfaces.

Another line of work combines implicit generative models with physical simulators [18, 24]. Mezghanni et al. [18] generate more physically stable SDF shapes by backpropagating gradients through a physical simulation layer. Their simulation mainly considers Newtonian dynamics. Our contribution is an effective, generalizable forward model for complex systems like fluids that represents surfaces accurately and efficiently, and can also be used for solving inverse problems.

3. Method

3.1. Setup: Particle-Based Fluid Simulation

We address the problem of learning the physical dynamics of fluids interacting with surfaces. Given the state of a physical system X^0 and some initial conditions (e.g. gravity, other external forces), rolling out a fluid simulation over T steps produces a sequence of states $X^{(1)}, ..., X^{(T)}$. In this paper, we focus on particle-based simulation; the state of the system at time t is represented by a set of particles $X^{(t)} = (p_1^{(t)}, ...p_n^{(t)})$, where p_i is the position of particle i in 3D space. A rollout over T steps is produced by repeatedly applying a forward model $S_\theta : X^{(t)} \rightarrow X^{(t+1)}$, which takes in the particle positions at time t (and optionally a short history $t-1, ...$) and produces the next-step particle positions. While the forward model can be used to directly predict $X^{(t+1)}$, most methods instead predict intermediate values Y, typically next-state accelerations for each particle, which can then be numerically integrated (e.g. Euler integration) to produce $X^{(t+1)}$; concretely, our task is to learn the parameters θ of this forward model $f_\theta : X^{(t)} \rightarrow Y$.

3.2. Graph Networks for Particle-Based Simulation

Graph neural networks lend themselves naturally to particle-based simulation, since they compute pairwise interactions between nodes and aggregate these interactions. Graph Network-based Simulators (GNS), a model architecture introduced by Sanchez-Gonzalez et al. [25], have in particular shown state-of-the-art performance on simulating fluids over long rollouts. These methods follow an "encode-process-decode" paradigm. First, the state $X^{(t)}$ is encoded into a graph, where nodes v_i are instantiated using a set of features (e.g. previous velocities, particle type, etc.); nodes v_i, v_j are then connected with an edge e_{ij} if their distance is less than a connectivity radius ϵ. Each node and edge is associated with an embedding. The "processor" stage then passes the encoded graph through M message-passing steps. Each message-passing step takes the current graph $\{(v_i, v_j, e_{ij})\}^{(m)}$. An edge module first computes the updated edge embeddings $e_{ij}^{(m+1)} = f_e(v_i^{(m)}, v_j^{(m)}, e_{ij}^{(m)})$; then the edge embeddings are aggregated to compute updated node embeddings $v_i^{(m+1)} = f_n(v_i^{(m)}, \sum_{j \in N(i)} e_{ij}^{(m)})$, where $N(i)$ are the set of neighbors of node i. f_e and f_n are generally MLPs with 2-3 hidden layers, and the weights are shared across edges/nodes (although differing across message passing steps), Finally, the "decoder" step applies an MLP f_D to the final node embeddings to predict the accelerations $\{y_i\}$.

In our problem, we further assume that there exists some rigid surface S that interacts with the fluid throughout the simulation. The GNS approach handles surfaces by discretizing the surface into particles (i.e. a point cloud) and adding these particles $\{s_i\}$ to the graph (denoting them by a different particle type, and masking during position updates) [25]. Follow-up work for mesh-based simulation handles surfaces similarly by representing the surface as an irregular mesh and adding nodes for mesh vertices [23].

3.3. Modeling Fluid-Surface Interactions with SDFs

We propose to model the interactions of fluids and surfaces with signed distance function (SDF) representations of surfaces. An SDF represents an object as a function

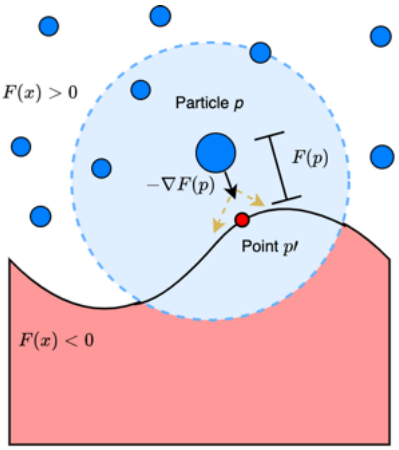


Figure 2: Rigid surfaces can be represented compactly and implicitly as the level set of a SDF: $\{x|F(x) = 0\}$. Evaluating F and its derivatives at particle position p provides useful information for simulating the interaction of that particle with the surface.

.

$F(x) = s$, which takes a spatial point $x \in \mathbb{R}^3$ and outputs its distance s to the *closest* corresponding point on the surface. The function is "signed", such that it takes on positive values for points outside the surface and negative values in the surface interior. The surface of the object is implicitly represented as the zero-level set $F(x) = 0$. Explicit representations of the surface can be constructed by applying meshing algorithms such as Marching Cubes [15].

3.3.1 Locality and Geometric Information for Fluid-Surface Interactions

A key insight of our method is that fluid-surface interactions are inherently about local rather than global surface geometry. The interaction of a fluid with a surface depends on a small local region of the surface, over which the geometry can have limited variation. This suggests that an effective representation of local surface geometry, trained on a variety of object shapes, can generalize OOD to shapes that may look different globally. By contrast, particles are *not* an effective representation of surface geometry; for any moderately complex surface, discretization leads to artifacts such as non-smooth fluid motion and even penetration of the surface boundary (we confirm this empirically).

By contrast, we argue that SDFs are a smooth, richly informative representation of local surface geometry for learning generalizable fluid-surface interactions. Our key insight is shown in Figure 2. Given a solid surface represented as an SDF $F(x)$, we can use properties of the SDF to inform the behavior of a fluid particle as it reaches the surface. The value of the SDF $F(x)$ gives the distance from the particle at point p to the closest point on the surface p'. Moreover, the SDF gradient $\nabla F(x)$ provides the unit-norm displacement vector from the particle to its closest surface point $\frac{(p-p')}{||p-p'||}$. This can be shown by observing that the SDF decreases most quickly on the line from p to p'. Thus, the SDF and its gradient provide the distance and orientation of the fluid particle w.r.t. to the closest point on the surface; as the fluid particle approaches the surface, its acceleration will more closely approach $\nabla F(x)$ and it will rebound from the surface. Moreover, since the gradient varies smoothly because of the continuity of F, these features do not suffer from artifacts.

Higher order derivatives of the SDF can also be used to gain geometric information. For example, we could march to the surface by computing $p - F(x)\nabla F(x)$. At the surface, the Hessian $\nabla^2 F(x)$ is the shape operator; its eigenvalues provide the principal directions of surface curvature κ_1 and κ_2, and the mean curvature is $H = \frac{1}{2}\text{Tr}(\nabla^2 F)$. We can also estimate the surface curvature via finite differences by slightly perturbing the gradient vector on each axis and computing the SDF (Fig. 2). In this paper, we only use the SDF and its gradient as features without curvature information; this can be seen as an *effective theory* where the relevant "local region" of the surface is a single closest point on the surface. As we show, modeling surfaces using SDF representations helps our fluid simulator generalize beyond the available training data by modeling fluid-surface interactions for unseen surfaces.

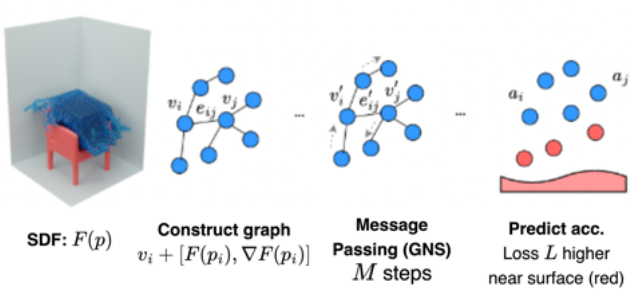


Figure 3: Given fluid particles and the SDF, our method augments the constructed graph with features from the SDF. We then apply message-passing and predict dynamics, weighting near-surface errors more highly in our loss.

3.4. Model

Having developed our ideas about SDFs and modeling fluid-surface interactions, we now describe our approach, which we call **Surf**ace implicit **s**Ubstitution for **p**articles (SURFSUP). We learn a forward predictor $f_\theta : X^{(t)}, F \to Y$, which takes as input a set of particle positions and a surface implicitly parametrized by an SDF F. We adopt the encode-process-decode paradigm for our model, but make two key changes. In the encoder step, for each fluid particle p_i we add the value of the SDF $F(p_i)$ and its gradient $\nabla F(p_i)$ to the initial node representations, such that the new representation is $v_i'^{(0)} = [v_i^{(0)}, F(p_i), \nabla F(p_i)]$. This is the way that the surface is encoded into the model; subsequently no particles representing the surface are added to the graph. The updated node embeddings are then passed through the "processor" with several message-passing steps and then decoded to predict the accelerations as standard. The other modification we make is to weight particles close to the surface higher during training when predicting dynamics. This has the effect of teaching the model to prioritize modeling fluid-surface interactions accurately. Since the model is predicting next-step accelerations, our loss becomes:

$$L = \sum_{F(p_i)>\alpha} ||a_i^{(t)} - \hat{a}_i^{(t)}||^2 + \sum_{F(p_i)\leq\alpha} \lambda ||a_i^{(t)} - \hat{a}_i^{(t)}||^2 \quad (1)$$

In our experiments, we set α based on the "neighborhood radius" of the classical simulator we train against, and set $\lambda = 5$. We found that this weighting helped the model learn

surface interactions more quickly without sacrificing overall accuracy on fluid prediction. Note that the SDF allows loss weighting to be implemented in a very natural way, by simple thresholding. Our approach is shown in Fig. 3.

3.5. Other Advantages of SDF Representations

We have argued for SDFs from the perspective of achieving smooth, information-rich representations of local surface geometry. Here we mention other advantages of SDFs, each of which we demonstrate in our experimental results.

Efficient message passing in the GNN. By modeling surfaces implicitly, we reduce the number of unique particles needed to compose the GNN's state. This allows us to perform more lightweight and computationally efficient simulation, as well as run the model for more message-passing steps. This advantage becomes especially clear for large objects/scenes or objects with complex surface geometry.

Shape Design with SDFs. SDFs are a useful paradigm for shape design and assembly. This is because compositions of shapes can be generated by union, intersection, and difference operations; e.g. the union of two SDFs F_1 and F_2 is $\min(F_1, F_2)$. The construction of shapes using primitives and these operations is known as Constructive Solid Geometry (CSG), and is implemented in many CAD packages for object design [8]. Moreover the recent emergence of neural SDF models as state-of-the-art 3D shape representations, offers another compelling avenue for shape design with end-to-end learning. These suggest that good forward models with SDFs could be leveraged for design problems.

4. Experiments

4.1. Fluid Domains

We evaluate our approach on complex prediction tasks involving fluids interacting with a variety of surfaces. Our main contribution here is a 3D fluid dataset consisting of water interacting with a range of primitive shapes, across shape parameters, positions, and orientations. For all our datasets, we use SPlisHSPlasH [5], a 3D-particle-based fluid simulator based on the SPH technique to generate the ground-truth. We emphasize that SURFSUP is trained only on simulations containing these primitive shapes and generalizes zero-shot to complex objects and scenes; we return to this point shortly.

PrimShapes Dataset. We train our models primarily on a 3D water dataset, where the fluid interacts with various shapes. We initialize a block of fluid directly above the shape, then let it fall due to gravity. The entire scene takes place in a $1 \times 1 \times 2$ m container. A random rotation is applied to the base shape which sits on the bottom of the container, and the initial position of the shape is also randomized. We extract 800 time-steps from the fluid rollout (with $\Delta t = 0.005$); this corresponds to 4000 steps for the classical simulator, which requires small time-steps for stability.

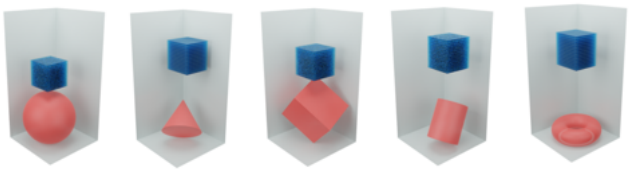


Figure 4: Training dataset: Examples showing the initial state of the simulation for different primitive shapes. Our training dataset consists only of simulations involving these five primitive shapes.

We use five "primitive" shapes in our dataset: spheres, boxes, cylinders, cones, and toruses. These shapes have analytical SDFs; for example the origin-centered sphere with radius r has SDF $F(p) = ||p||_2 - r$. Several of the other primitives have exact (if more complex) SDFs. These primitive shapes and their rigid-body transformations cover a range of geometric features such as curved surfaces, holes, edges, differently sloped/curved faces, and corners. Our key insight is that accurate fluid-surface simulation depends on understanding fluid interactions with local surface geometry; these primitives encompass a wide range of local geometric variation, which allows our model to generalize to interactions with surfaces of entirely novel *global* geometry. In Fig. 4 we show the initial states for several simulations in our training dataset. We again emphasize this **important point**: SURFSUP is trained on simulations containing only these primitive shapes, and is evaluated on complex shapes and real-world scenes without further training, for which it produces accurate, realistic fluid simulations (see Fig. 1, Fig. 5). Our training set consists of 1000 simulations and our testing set 100 simulations from the same distribution of shapes. See Appendix D.1 for further details on the creation of the dataset.

4.1.1 Generalization Test-Sets

We create a range of qualitative and quantitative evaluation scenarios to test our model's generalization ability (see Appendix D for details on these test sets):

Prim-OOD (60 simulations). The primitives-OOD dataset contains the same shapes as above, with OOD shape parameters from the training shapes (e.g. higher or lower radius r and height h for the cylinder).

Prim-Unions (40). This test set includes all possible pairwise unions of five shapes in our dataset. See Appendix D.2 for details and example shapes in this test set.

Funnels (12). We examine the model's ability to han-

Testing Set	Chamfer		Chamfer Surface		Number Inside		Mean SDF Inside	
	SURFSUP	GNS [25]	SURFSUP	GNS	SURFSUP	GNS	SURFSUP	GNS
Primitives	0.0297	0.0285	4.704	5.451	0	2000	-1.04e-11	-2.48e-05
Primitives-OOD	0.0321	0.0343	10.951	18.445	324	33006	-1.65e-06	-0.0006
Primitives-Unions	0.0380	0.0348	24.553	26.307	10000	63000	-4.46e-05	-0.001
Funnels	0.0362	0.0521	9.990	13.663	2	7204	-2.51e-09	-0.001
Complex-Scenes	0.0638	0.2658	50.521	276.176	40513	257722	-0.0011	-0.026

Table 1: Results on in-distribution (Primitives) and OOD test sets, comparing SURFSUP to GNS [25] which represents the surface as particles.

dle "differences" of shapes by creating funnels, which are formed by carving an inverted cone out of a cylinder.

'Complex' Scenes (7). As the most challenging test of our model's generalization, we examine predicted rollouts on several real world objects (Details of these objects and their sources are in Appendix D.3). To do so, we leverage neural SDF representations of shapes, which can represent complex shapes with high accuracy and fidelity [30, 22]. Instead of an analytical SDF, we use a trained neural SDF $f_\theta(p)$ to generate our SDF features during our forward pass. To test our model's ability to generalize to complex objects/scenes, we train a single SIREN model [30] for 7 fine-grained objects (including the coral, lion, and room scenes shown in Fig. 1, Fig. 5). For each surface we create a fluid scene using the ground-truth simulator, and evaluate our method on this small **Complex-Scenes** dataset. To represent the geometry at an appropriate scale, these objects are scaled 2x in the fluid scene compared to previous datasets. To test our model on *families* of shapes, which gives us the capacity to solve inverse problems, we train a variational DeepSDF model [22] for the ShapeNet chair and bowl categories.

4.2. Training Details

For all experiments, we train our model on the PrimShapes dataset. The training data is pairs $X^{(t)}, X^{(t+1)}$ from the ground-truth simulator. We use a graph network (GN) architecture with 10 message-passing steps and train with Adam for 2M steps (see Appendix E.1 for details).

4.3. Metrics and Baseline

Although we train on next-step prediction, we measure performance on full rollouts. We use mean Chamfer distance between the predicted particles P and ground-truth G to measure overall rollout agreement (averaging across timesteps). Compared to MSE, Chamfer distance is permutation-invariant and therefore is a better measure of similarity between fluid flows [25]. This metric, however, can be noisy since it averages over long rollouts on chaotic systems with many particles. Since we focus on the quality of fluid-surface interactions, we also compute a "Chamfer surface" metric that focuses on prediction errors near the surface. For each time-step, we consider the predicted point cloud only near the surface (where $F(p) < \alpha$ for some threshold α) and measure the Chamfer distance of these particles to G, summing over the particles; we do this symmetrically for G (see Appendix E.2 for details). The intuition is that discrepancies between P and G near the surface are particularly important for evaluating fluid-surface interactions.

Since surface penetration is a serious failure mode, we also measure the number of particles inside the surface through the rollout (summing over the number of timesteps). Additionally, we compute the average SDF value for particles *inside* the surface, indicating the extent of surface penetration (since SDFs become increasingly negative deeper into the surface interior).

Baseline. Our main comparison is to the graph network simulator (GNS) proposed by Sanchez Gonzalez et al. [25]. Their approach proposes to represent surfaces explicitly by discretizing the surface into particles, which are added to the graph and participate in the message-passing. To implement this baseline, we convert all the SDFs in our datasets to meshes using Marching Cubes [15]. From this mesh we uniformly sample 2000 surface particles (comparable to the number of fluid particles in our simulations) and add them to the graph. In all our experiments we compare our approach which represents the surface implicitly, to this baseline with explicit surface particles.

While the baseline method could in principle be extended to include local surface information (e.g. using surface normal estimates for each particle as additional features, or mesh connectivity), we omit such an extension here. This is because any explicit representation of large surfaces quickly becomes computationally infeasible, especially for large objects/scenes or complex geometry. Whereas the simulation quality of SURFSUP scales with the complexity of the SDF network, the baseline quality scales with the number of particles. We show in Sec 5 that SURFSUP still outperforms (with lower computational cost and inference time) a baseline with more particles.

5. Results

SURFSUP can accurately predict fluid behavior over long rollouts, interacting with a range of complex surfaces (Fig. 5). Across the different shapes in our dataset (e.g. cylinders, cones, tori) and different position and orientations, we can

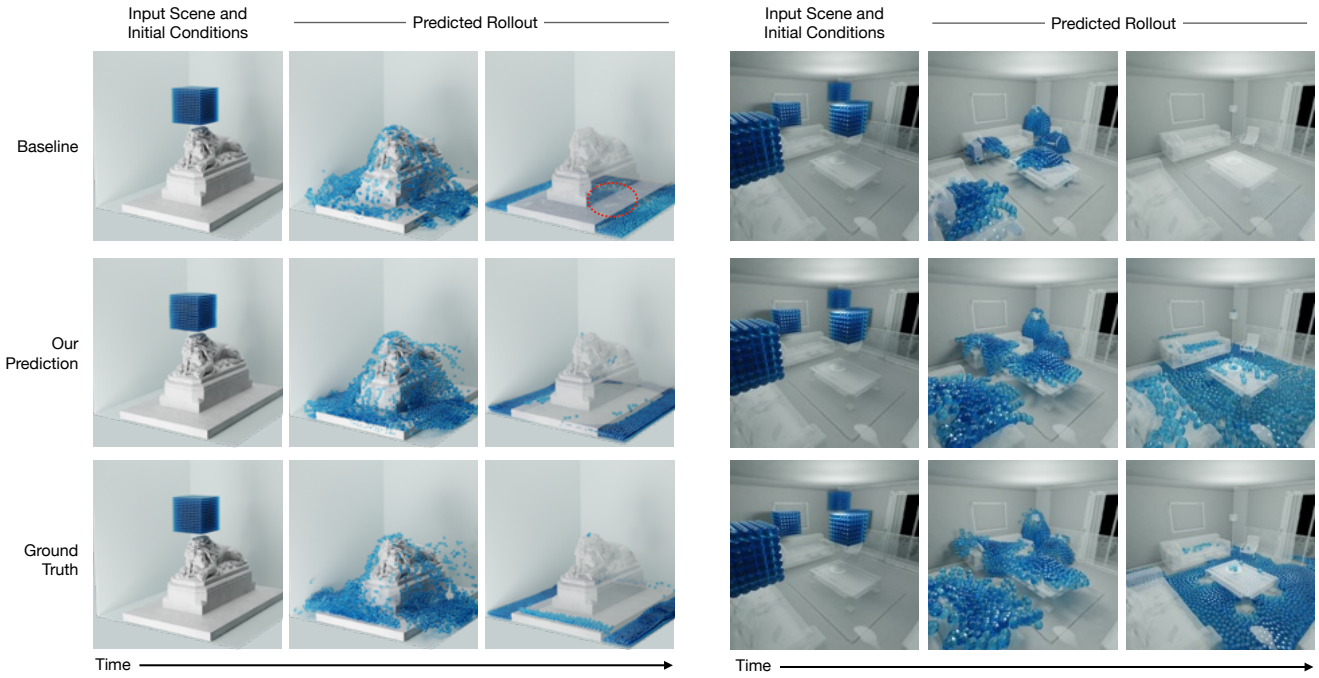


Figure 5: Although SURFSUP is only trained on primitive shapes, our model generalizes to complex and real-world shapes without any additional training. We visualize our predictions (middle) for two scenes. In comparison, the baseline (top) often generates unrealistic predictions, such as having water penetrate solid objects. In many cases, the predictions from the baseline cause the water to disappear because it falls through the floor (top right).

generate accurate and visually realistic predictions of fluid behavior (see Appendix A for rollouts on primitive shapes and simpler OOD test sets). Our *key result* is the following: we find that SURFSUP can generalize effectively to shapes that are significantly OOD for the learned model, and even generalize to complex real-world shapes and scenes. This demonstrates the ability of our approach to leverage the rich geometric information provided by the SDF to handle a wide range of surface geometry; and suggests that our approach can lead to a robust and generally accurate learned fluid simulator.

Quantitative Results and Comparison to Baseline. We evaluate SURFSUP on rollouts from an in-distribution test set and several OOD test sets, and compare to the GNS baseline. Results are shown in Table 1. Across all test sets, our model's fluid predictions display significantly lower surface penetration than the baseline (e.g. 324 vs. 33006 particles on average for the primitives+OOD test set). Moreover, for those few particles which *have* penetrated the surface, the average SDF value for our method is nearly zero; this implies that any particle that penetrates the boundary ($F(p) < 0$) is quickly projected outward to the region $F(p) > 0$. This indicates that the SDF provides a strong signal for fluid motion near the boundary; as $F(p)$ approaches 0 the particle tends to move away from or parallel to the surface. Chamfer distance near the surface, which measures *overall* accuracy of fluid-surface interactions beyond surface penetration, is consistently lower for our method than the baseline. This indicates that our model accurately predicts fluid dynamics near the surface, and that SURFSUP produces more accurate fluid simulations for interactions with solid objects compared to GNS [25]. Overall Chamfer distance between the two models is generally comparable. This is unsurprising given that small differences in particle positions along long time-horizons, potentially far away from the surface, can accumulate error (see Sec. 4.3).

Overall, the quantitative results indicate that SURFSUP generalizes well to unseen shapes. For example, the funnel is an inverted cone carved out of a cylinder; our model has seen standard cones and cylinders in isolation, and can reason successfully about dynamics inside the funnel (see Fig. 14). In the next section, we discuss our central results on complex shapes and scenes.

Generalizing to Novel Shapes The key result of our method is its ability to generalize zero-shot to new, complex shapes and objects unseen during training. We evaluate SURFSUP on several real-world scenes, including chairs and bowls from Shapenet, and fine-grained objects and scenes in the real-world (See Fig. 1, Fig. 5, and Fig. 6). Remarkably SURFSUP trained only with primitive shapes can

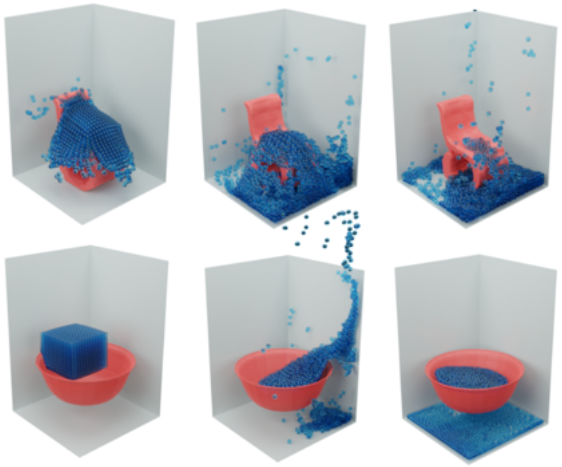


Figure 6: SURFSUP predictions on ShapeNet objects.

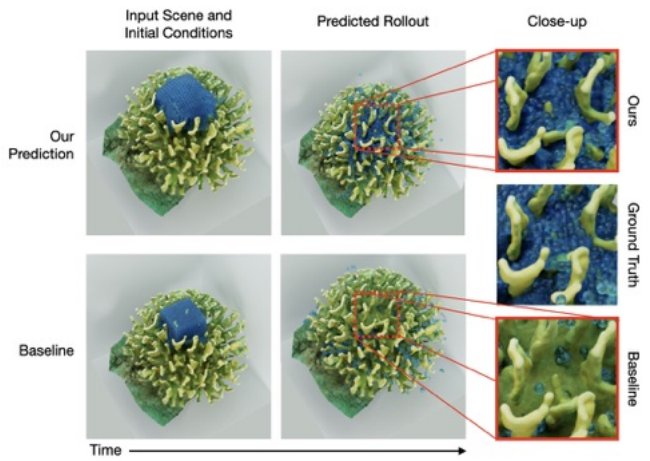


Figure 7: Comparing our method and the GNS baseline on the coral object. The particle-based baseline cannot capture the fine structure of the coral.

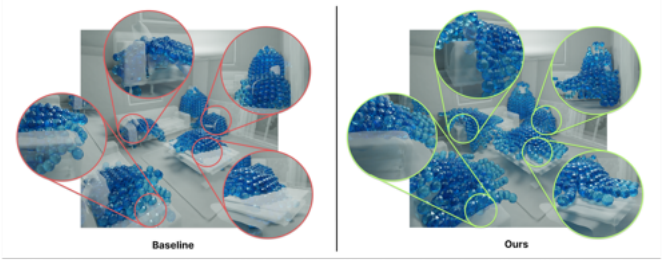


Figure 8: Comparing SURFSUP and the GNS baseline on the room scene, highlighting different areas of the scene. Notice that significant penetration of the surface occurs in the baseline, while our method's predictions correctly respect the shape of the objects in the scene.

generalize zero-shot to neural SDF representations of complex shapes and scenes. This is notable for two reasons: first, our chosen real-world objects differ *widely* in global geometry (see the variation in Fig. 5 and Fig. 6). indicating that our approach can robustly handle real-world variations in geometry to predict fluid behavior. Moreover, the SDFs generated by neural SDFs are imperfect compared to analytical SDFs. Despite these challenges, SURFSUP can predict fluid behavior with a high degree of accuracy and visual realism, while avoiding unintuitive artifacts (such as surface penetration) seen in the GNS baseline. Quantitative results (Table 1, row 'Complex-Scenes') show that SURFSUP significantly outperforms GNS across *all* metrics on a set of real-world objects and scenes; SURFSUP's advantages are elucidated by the larger object sizes and more complex geometry that characterizes these scenes.

Two key generalization capabilities of our method, compared to the particle-based baseline, are the ability to handle fine geometric structure, and to scale to large and complex scenes. Fig. 7 compares SURFSUP with GNS on fluid interacting with a coral object. GNS is unable to accurately model fluid interacting with the fine-grained maze of coral tentacles. In contrast, our method can accurately capture the fluid interacting with and filtering through the coral object (see the website for a video comparison). For generalizing to complex geometric objects, our implicit representation of surface geometry provides a better foundation for accurate fluid prediction.

Another key ability of SURFSUP is its ability to scale effectively to large, complex scenes. The scaling law of our method is the capacity of the network needed to represent the scene. In contrast GNS scales up with the number of particles, which quickly becomes computationally infeasible for fluid simulation with large scenes. In Fig. 8 we show our prediction on the room scene vs. the baseline and highlight different regions of the simulation. In the baseline the fluid consistently penetrates and passes through the surface of objects, while in SURFSUP the fluid interacts in accurate and realistic ways with the furniture in the scene.

Increasing Number of Surface Particles for GNS. In our experiments we use a baseline with 2000 surface particles; here we study increasing the number of particles for the GNS baseline on two different scenes. We examine a scene in the Primitives-Unions test set (the intersection of a sphere and box, see Fig. 14c), for which the GNS baseline's fluid predictions display significant penetration of the surface. We increase the number of surface particles for the baseline from 2000 to 5000 (Fig. 9). As the number of particles increases for the baseline, thus improving the resolution of the surface, surface penetration decreases and GNS's fluid predictions improve (see left plot and qualitative results in Fig. 9). However, even at 5000

surface particles, the number of fluid particles penetrating the surface for the baseline is an order of magnitude higher than for SURFSUP (12285 vs. 490 particles for ours over the entire rollout). Moreover, the computational cost of the baseline, indicated by the maximum number of edges in the fluid graph during a rollout, increases significantly with the number of surface particles (more than 2x of our method at 5000 surface particles). Our method displays near-zero surface penetration with constant, smaller-sized graphs. We conduct a similar analysis of the room scene (Fig. 5), which is situated in and occupies a high volume of a $[-1, 1]^3$ region. Remarkably, we find that our method outperforms a baseline even with 10000 surface particles representing the room scene (measured by overall Chamfer surface distance, see Appendix B for details). This indicates that while adding particles can improve the accuracy of GNS, in contrast to SURFSUP there is a fundamental difficulty in scaling up the baseline to large, complex scenes.

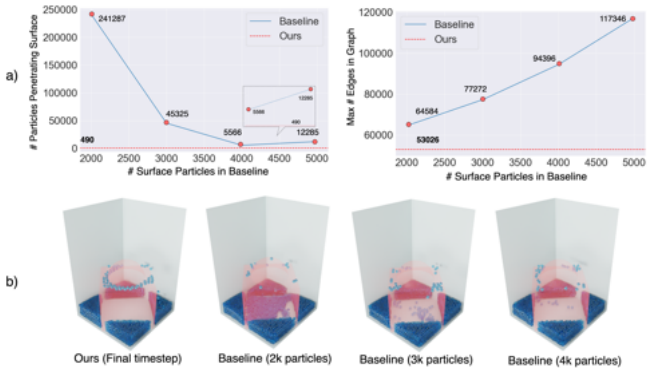


Figure 9: Comparing our method vs. the baseline with more surface particles. (a) Left shows how surface penetration is affected; even as surface penetration decreases with the number of surface particles, our model consistently displays lower surface penetration (red line, 490 particles). The speech bubble zooms in on 4000-5000 surface particles for the baseline, where our approach still has an order of magnitude improvement. The right plot shows how the size of the graphs increase with the number of surface particles. We measure the maximum number of edges during the rollout, which indicates memory cost. (b) Final timesteps of the fluid rollouts for our method and the baseline at 2000, 3000, and 4000 particles. Surface penetration for the baseline expectedly reduces with more particles; however, our method displays improved, near-zero penetration with no increase in computational cost.

Scaling up SURFSUP. As a final test of our method's ability to scale up in size and complexity, we set up a natural scene with complex and fast-varying topography, based on the Puncak Jaya mountains in Indonesia. This scene was

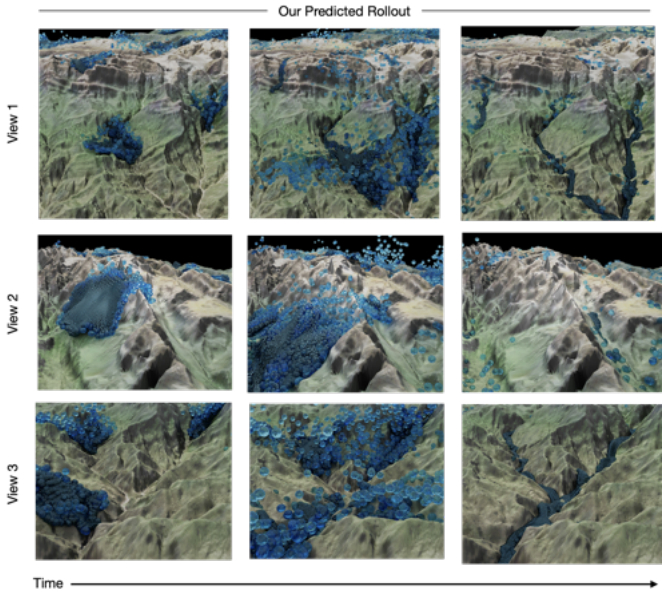


Figure 10: Predicted fluid rollouts from SURFSUP on different parts of a $40 \times 27 \times 6\ m^3$ mountain scene. Our method produces realistic fluid rollouts that accurately respect features of the scene such as steep mountain slopes, narrow valleys, and confluences (merging of two streams). This indicates the ability of SURFSUP to scale fluid simulation to large real-world scenes.

$40 \times 27 \times 6\ m^3$ in size ($6840\ m^3$ compared to roughly $10\ m^3$ for previous scenes), and contained 68,600 fluid particles (an order-of-magnitude more than previous scenes). See Appendix D.5 for setup details; notably we use the fact that SDFs, unlike particles, can be easily scaled to any size by applying a scaling transform with factor s). Fig. 10 shows the predictions of SURFSUP; our method generalizes effectively to this large scene and the resulting rollout captures complex features such as steep slopes and narrow valleys between the mountains. Note that GNS would require $O(1M)$ particles for this scene; by contrast we leave the network size for the SIREN model unchanged for this scene compared to previous scenes/objects e.g. in Fig. 5. This demonstrates the potential of implicit representations for large-scale, realistic dynamics. Please see the website for full rollouts of the entire mountain scene and the views displayed in Fig. 10.

Inverse Design. Finally, we evaluate SURFSUP on its potential to solve object design tasks. We show the ability of our model to design both parametrized shapes, by optimizing the shape parameters with gradients, as well as shapes represented by neural SDF models.

Parametrized Design Tasks. We consider the funnel, which is an inverted capped cone carved out of a cylinder (Fig. 14d), for which we have derived an analytical SDF.

This shape is parametrized by the cylinder radius R, height H, cone height $h = H$; and the radii r_1 and r_2 of the bottom and top circles that "cap" the cone, with $r_1 > r_2$.

In our setup, we drop a block of fluid directly onto the object. We consider two tasks: designing a *bowl* where the task is to contain the fluid inside the object, and a *funnel* where the task is to concentrate the water onto a location onto the ground. We optimize r_1 (which we call the "capture radius" that captures fluid into the object) and r_2 (the "filter radius" which decides how much water to filter onto the ground). For the funnel, the desired solution is large r_1 and small r_2, and for the bowl we want $r_2 \to 0$. If our learned model is performing well, the model should be able to discover these solutions.

We roll out our model for 50 (bowl) and 75 (funnel) timesteps. Our reward measures the log probability of a Gaussian centered at the desired location of the fluid particles. For the bowl task, we use the bottom of the object, i.e. $\mathcal{N}((0., 0., h), \Sigma)$, where h is the object height. For the funnel we use a 2D Gaussian $\mathcal{N}((0., 0.), \Sigma)$ and measure the reward on particles that reach the ground during training. For a given design, we roll out our model and obtain gradients, which we use to iteratively refine the design.

Qualitative results are shown in Fig. 11. Starting from a sub-optimal design with low r_1 and $r_2 > 0$, our model successfully converges for the bowl; similarly the funnel converges from a suboptimal to an optimal solution. This shows that over long rollouts and hundreds of message-passing steps with dense fluid-surface interactions, our model's gradients are informative and useful for solving inverse design problems. See Appendix F for details; we also show that performance is consistent across different initializations of r_1 and r_2.

Latent Space Design. We further examined whether SURFSUP could directly optimize in the latent space of a DeepSDF model, trained on ShapeNet chairs. For this investigation, as a proof-of-concept we aimed to discover a novel chair that traps and contains falling water (i.e. maximizes the "bowl reward" described above). Given a DeepSDF model $f_\theta(p, z)$, where z is the latent code, we iteratively refine the design by rolling out our simulation for 100 time-steps, computing the task reward, and updating z via gradient-based optimization. Across multiple trials with different initializations, our model is able to converge from an initially sub-optimal chair to a final design that can legitimately capture and contain falling water, while retaining the semantic look and feel of a chair (Fig. 12).

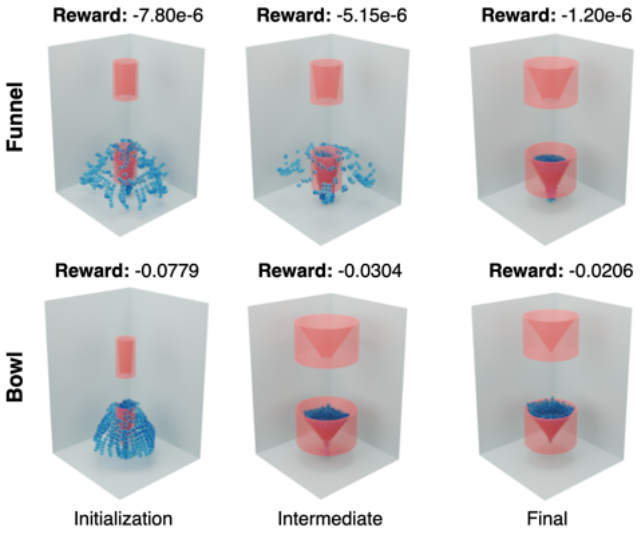


Figure 11: Design optimization; top is the funnel task, bottom the bowl task. For each frame, the design is shown near the top of the container and a timestep from the fluid rollout is shown below. Initial, intermediate, and final designs are shown; note that the reward increases through optimization and final designs approximate a funnel and bowl well.

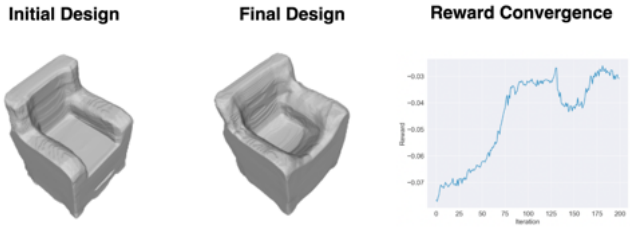


Figure 12: We differentiate through SURFSUP to directly optimize (in the latent space of a DeepSDF) a chair that holds water dropped onto it.

6. Conclusion

In this paper, we have taken a first step towards integrating implicit 3D representations into the dynamical simulation of physical models. SURFSUP realizes this approach by incorporating the implicit representation of solid surfaces (via SDFs) into a graph neural network fluid simulator. As such, SURFSUP is able to accurately model how fluids interact with surfaces, including highly complex surfaces unseen in training. We find that promoting this type of OOD generalization is helpful in both forward and inverse contexts.

Acknowledgements: We would like to thank Huy Ha for his help with creating visualizations of fluid rollouts. We also acknowledge Jan Bender for helpful feedback on finer points of the the SPlisHSPlasH simulator, as well as Kelsey Allen, Tobias Pfaff, and Alvaro Sanchez-Gonzalez for advice on training the graph network-based simulator. We thank Max Helman for early work on setup of the SPlisHSPlasH simulator. This research is based on work partially supported by the NSF STC for Learning the Earth with Artificial Intelligence and Physics, and the NSF NRI Award #1925157. AM is supported by the NSF fellowship.

References

[1] Panos Achlioptas, Olga Diamanti, Ioannis Mitliagkas, and Leonidas J Guibas. Learning representations and generative models for 3d point clouds. *arXiv preprint arXiv:1707.02392*, 2017.

[2] Kelsey R Allen, Tatiana Lopez-Guevara, Kimberly Stachenfeld, Alvaro Sanchez-Gonzalez, Peter Battaglia, Jessica Hamrick, and Tobias Pfaff. Physical design using differentiable learned simulators. *arXiv preprint arXiv:2202.00728*, 2022.

[3] Peter Battaglia, Razvan Pascanu, Matthew Lai, Danilo Jimenez Rezende, et al. Interaction networks for learning about objects, relations and physics. *Advances in neural information processing systems*, 29, 2016.

[4] Jan Bender and Dan Koschier. Divergence-free smoothed particle hydrodynamics. In *Proceedings of the 14th ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, SCA '15, page 147–155, New York, NY, USA, 2015. Association for Computing Machinery.

[5] Jan Bender and Dan Koschier. Divergence-free smoothed particle hydrodynamics. In *Proceedings of the 14th ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, SCA '15, page 147–155, New York, NY, USA, 2015. Association for Computing Machinery.

[6] Siddhartha Chaudhuri, Daniel Ritchie, Jiajun Wu, Kai Xu, and Hao Zhang. Learning generative models of 3d structures. In *Computer Graphics Forum*, volume 39, pages 643–666. Wiley Online Library, 2020.

[7] Christopher B Choy, Danfei Xu, JunYoung Gwak, Kevin Chen, and Silvio Savarese. 3d-r2n2: A unified approach for single and multi-view 3d object reconstruction. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2016.

[8] James D Foley, Foley Dan Van, Andries Van Dam, Steven K Feiner, and John F Hughes. *Computer graphics: principles and practice*, volume 12110. Addison-Wesley Professional, 1996.

[9] Christoph Gissler, Andreas Peer, Stefan Band, Jan Bender, and Matthias Teschner. Interlinked sph pressure solvers for strong fluid-rigid coupling. *ACM Transactions on Graphics (TOG)*, 38(1):1–13, 2019.

[10] Zekun Hao, Hadar Averbuch-Elor, Noah Snavely, and Serge Belongie. Dualsdf: Semantic shape manipulation using a two-level representation. pages 7631–7641, 2020.

[11] Dan Koschier and Jan Bender. Density maps for improved sph boundary handling. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 1–10, 2017.

[12] 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.

[13] Yunzhu Li, Jiajun Wu, Jun-Yan Zhu, Joshua B Tenenbaum, Antonio Torralba, and Russ Tedrake. Propagation networks for model-based control under partial observation. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 1205–1211. IEEE, 2019.

[14] Zhijian Liu, William T Freeman, Joshua B Tenenbaum, and Jiajun Wu. Physical primitive decomposition. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 3–19, 2018.

[15] William E Lorensen and Harvey E Cline. Marching cubes: A high resolution 3d surface construction algorithm. *ACM siggraph computer graphics*, 21(4):163–169, 1987.

[16] Andreas Mayr, Sebastian Lehner, Arno Mayrhofer, Christoph Kloss, Sepp Hochreiter, and Johannes Brandstetter. Boundary graph neural networks for 3d simulations. *arXiv preprint arXiv:2106.11299*, 2021.

[17] Lars Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, and Andreas Geiger. Occupancy networks: Learning 3d reconstruction in function space. In *Proceedings IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[18] Mariem Mezghanni, Théo Bodrito, Malika Boulkenafed, and Maks Ovsjanikov. Physical simulation layer for accurate 3d modeling. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13514–13523, 2022.

[19] Mateusz Michalkiewicz, Jhony K Pontes, Dominic Jack, Mahsa Baktashmotlagh, and Anders Eriksson. Implicit surface representations as layers in neural networks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 4743–4752, 2019.

[20] J. J. Monaghan. Smoothed particle hydrodynamics. *Annual Review of Astronomy and Astrophysics*, 30:543–574, Jan. 1992.

[21] Damian Mrowca, Chengxu Zhuang, Elias Wang, Nick Haber, Li F Fei-Fei, Josh Tenenbaum, and Daniel L Yamins. Flexible neural representation for physics prediction. *Advances in neural information processing systems*, 31, 2018.

[22] Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. Deepsdf: Learning continuous signed distance functions for shape representation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 165–174, 2019.

[23] Tobias Pfaff, Meire Fortunato, Alvaro Sanchez-Gonzalez, and Peter Battaglia. Learning mesh-based simulation with graph networks. In *International Conference on Learning Representations*, 2020.

[24] Edoardo Remelli, Artem Lukoianov, Stephan Richter, Benoit Guillard, Timur Bagautdinov, Pierre Baque, and Pascal Fua. Meshsdf: Differentiable iso-surface extraction. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 22468–22478. Curran Associates, Inc., 2020.

[25] Alvaro Sanchez-Gonzalez, Jonathan Godwin, Tobias Pfaff, Rex Ying, Jure Leskovec, and Peter Battaglia. Learning to simulate complex physics with graph networks. In *International Conference on Machine Learning*, pages 8459–8468. PMLR, 2020.

[26] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE transactions on neural networks*, 20(1):61–80, 2008.

[27] C. Schenck and D. Fox. Spnets: Differentiable fluid dynamics for deep neural networks. In *Proceedings of the Second Conference on Robot Learning (CoRL)*, Zurich, Switzerland, 2018.

[28] Dong Wook Shu, Sung Woo Park, and Junseok Kwon. 3d point cloud generative adversarial network based on tree structured graph convolutions. 2019.

[29] Vincent Sitzmann, Eric R. Chan, Richard Tucker, Noah Snavely, and Gordon Wetzstein. Metasdf: Meta-learning signed distance functions. In *Proc. Neurips*, 2020.

[30] Vincent Sitzmann, Julien N.P. Martel, Alexander W. Bergman, David B. Lindell, and Gordon Wetzstein. Implicit neural representations with periodic activation functions. In *Proc. NeurIPS*, 2020.

[31] Benjamin Ummenhofer, Lukas Prantl, Nils Thuerey, and Vladlen Koltun. Lagrangian fluid simulation with continuous convolutions. In *International Conference on Learning Representations*, 2019.

[32] Subeesh Vasu, Nicolas Talabot, Artem Lukoianov, Pierre Baqué, Jonathan Donier, and Pascal Fua. Hybridsdf: Combining deep implicit shapes and geometric primitives for 3d shape representation and manipulation. In *International Conference on 3D Vision*, 2022.

[33] Zhirong Wu, S. Song, A. Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and J. Xiao. 3d shapenets: A deep representation for volumetric shapes. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1912–1920, Los Alamitos, CA, USA, jun 2015. IEEE Computer Society.

A. Further Rollouts and Videos

In Figure 14 we show rollouts on several more examples in our test sets, including examples in the PrimShapes test set, the union and funnel test sets, and Shapenet objects. For each rollout we show the corresponding frames from the ground-truth simulator, demonstrating that our model has high fidelity to the ground-truth dynamics. See the website for full videos of several rollouts in the paper.

B. Increasing Particles for the GNS Baseline (Room Scene)

In the main paper we analyze a scene in the primitives+unions test set, and show how SURFSUP outperforms a GNS baseline even with more surface particles. We conduct a similar analysis on the room scene in the main paper. The room is situated within a $[-1, 1]^3$ box, and contains many objects that overall occupy a high volume of the space. As shown in the main paper (Fig. 8), the baseline's predictions cause fluid to fall through the surface and penetrate the objects in the scene. We investigate increasing the number of particles for the baseline for room scene (Fig. 13), and measure the overall accuracy of fluid-surface dynamics through the 'Chamfer surface' distance. Notably, our method still outperforms the GNS baseline with 10,000 surface particles (we observe similar results with surface penetration). At this number the inference times and graph sizes incurred by the baseline are significantly higher than those of our method. This makes it clear the advantages of our method; while we can scale effectively to arbitrarily large scenes, the baseline is fundamentally limited.

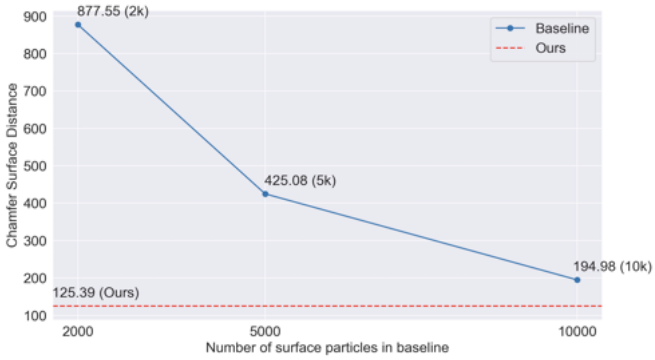


Figure 13: Comparing our method vs. the baseline on the room scene (Chamfer surface distance), as the number of surface particles increases from 2000 to 10000.

C. Ablation Study and Timing

We performed an ablation study on the SDF features provided to the model during training (Table 2). We found that providing only the SDF value and not the gradient (which conveys the particle's orientation w.r.t. the surface), degraded performance near the surface significantly; for example penetration went from 0 to > 30000 particles. Providing no SDF information at all and only fluid particles performs very poorly as expected.

	Chamfer	Cham-Surface	Num Inside
SURFSUP	0.0297	4.704	0
Value-only	0.0319	10.782	30433
No SDF	0.0815	45.463	193071

Table 2: Ablating SDF features on the PrimShapes test set.

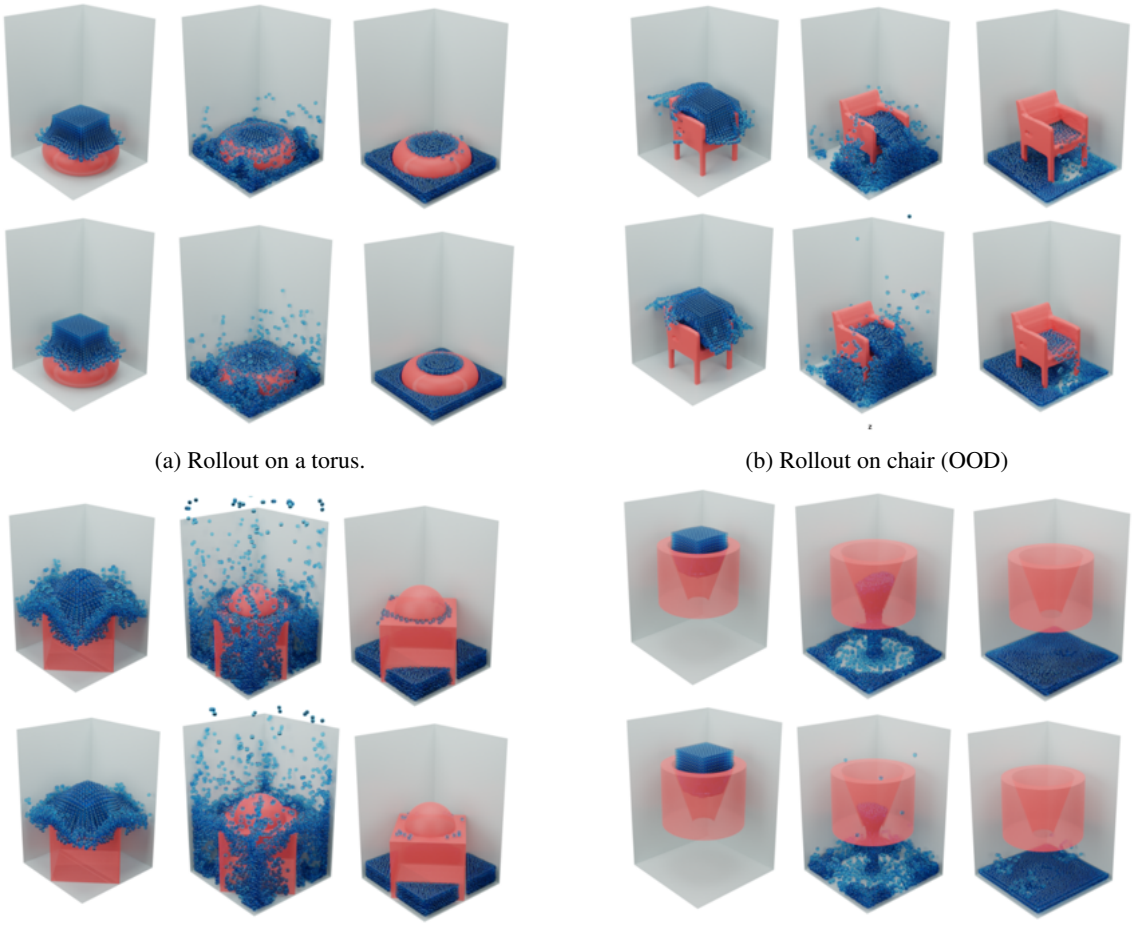
A brief note on inference times: Inference times can be highly variable, and we have chosen to report graph sizes in the main paper as a more reliable measure of computational cost. To provide a brief and rough comparison, we broadly find that SURFSUP's speed for fluid rollouts is comparable to if not slightly faster than the GNS baseline; experiments (benchmarked on a single A6000 GPU) suggest that we are 37% faster then the baseline on the PrimShapes test set and 13% faster on the Chairs test set (where SURFSUP involves evaluating neural SDFs). This can be attributed to the fewer particles and smaller graph sizes entailed by our method compared to the baseline. We find both methods 2-3x faster than the classical simulator (evaluated on a 40-core CPU).

D. Dataset Details

We provide further details about dataset creation and visualizations of shapes in our dataset. To sample trajectories for all datasets we use a classical (non-neural) particle-based fluid simulator. In particular, we use the SPlisHSPlasH simulator (github.com/SPlisHSPlasH), a 3D particle-based fluid simulator based on the SPH technique. We extract 800 time-steps from each simulation (with $\Delta t = 0.005$); this corresponds to 4000 time-steps of the classical simulator ($\Delta t = 0.001$), which requires smaller time-steps for physical stability. This also allows us to run our simulations faster than the classical simulator. All rollouts in training and test are of this length (total of $t = 4$ seconds). In all our simulations we simulate water.

D.1. PrimShapes Dataset

The Prim-Shapes Dataset consists of five primitive shapes. For the training set, we generate 200 simulations for each shape (total of 1000 simulations), where the simulation takes place in a $1 \times 1 \times 2$ m container. The test set consists of 100 simulations. We sample each primitive shape from the following:



(a) Rollout on a torus.

(b) Rollout on chair (OOD)

(c) Rollout on a union of primitives (OOD)

(d) Rollout on funnels (OOD)

Figure 14: Rollouts for several examples in our test sets. For each rollout, the top row shows our *prediction* and the bottom row shows the ground-truth simulation.

- **Sphere**: The sphere is parametrized by a radius r. We choose a random radius in the range $[0.25, 0.5]$.
- **Box**: The box is parametrized by the side lengths s_x, s_y, s_z. We choose each in the range $[0.4, 0.7]$.
- **Cylinder**: The cylinder is parametrized by its radius r and height h. We choose the radius and height in the ranges $[0.15, 0.35]$ and $[0.4, 0.75]$ respectively.
- **Cone**: The 'capped cone' is parametrized by its height h, its bottom radius r_1, and the top radius r_2 (with $r_1 > r_2$). We choose the height and bottom radius in the ranges $[0.4, 0.7]$ and $[0.2, 0.4]$ respectively; the top radius is fixed at 0.01 to approximate a true cone.
- **Torus**: The torus is constructed by revolving a circle with 'inner' radius' r_2 around an axis coplanar with the circle; the 'outer radius' r_1 is the distance from this circle center to the center of the torus. We choose r_1 between $[0.2, 0.4]$ and r_2 between $[r_1/4, 3r_1/4]$; this ensures there is always a hole in the center of the torus.

After selecting the shape, each shape is rotated randomly and translated to the bottom of the box and randomly on the $x - y$ plane. When computing the SDF, these are implemented as inverse transformations on the query point. For the classical simulator which takes in meshes, we run Marching Cubes on the SDFs. The block of fluid is initialized directly above the shape, with the height above the object randomized. The size of the fluid block is randomized between $[0.4, 0.5]^3$, each side length chosen independently; since $r = 0.015$ is the particle radius, this initializes the number of particles roughly in the range $[1900, 3800]$ for each simulation.

D.2. OOD Test Sets

The Prim-OOD test set is constructed by choosing shape parameters lower or higher than the ranges described above.

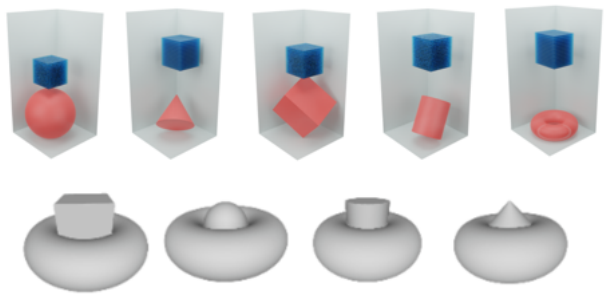


Figure 15: (Top) Examples of different initializations in the PrimShapes Dataset, reproduction of Fig. 4. (Bottom) Examples of shapes in the Prim-Unions Dataset with the torus as the base shape.

For the primitives-unions test set, we generate unions for all possible pairwise permutations of shapes. If (S_1, S_2) is a pair of shapes, we choose S_2 to be a larger, base shape and S_1 a smaller shape that 'sits' on top of S_2, intersecting with S_2 near the origin. Since we have 5 shapes, this yields a total of 20 unique shapes. Once the shape is generated, we translate it to the bottom of the box and initialize a block of fluid above the shape. For each unique shape we generate two simulations with the height and size of the fluid block over the shape randomly chosen. The SDF for these shapes is computed as $\min(S_1, S_2)$. Example shapes are shown in Fig. 15. Finally for the funnels test set, we derive an analytical SDF for the funnel; we generate 12 simulations with varying values of r_1 and r_2.

D.3. Complex shapes/scenes, SIREN training

We evaluate our model on several complex objects and scenes in the paper (Fig. 1, Fig. 5). For each of these objects we train a SIREN model [30], which uses sinusoidal activations to learn accurate SDFs of fine-grained shapes. We follow the training details in the original SIREN paper and in the github. During our forward pass the trained SIREN model is used to generate SDF features. The small 'Complex-Scenes' dataset, for which SIRENs were trained, consists of 7 shapes; two dragon statues and an armadillo statue (from the Stanford 3D Scanning Repository), two lion statues, a coral object, (all three obtained freely online, links: coral, lion1, lion2), and the room scene (from the SIREN github). SIREN rescales each object to within a $[-1, 1]^3$ box. For each object, we place this object inside a $2 \times 2 \times 3$ container and initialize a block of fluid above the object. For the room scene, we initialize fluid directly inside the room at several select locations above the furniture objects.

D.4. Shapenet objects, DeepSDF training

We also evaluated our model on chairs and bowls from Shapenet. For this purpose we trained a DeepSDF model [22]. DeepSDF is a variational 'auto-decoder' model, which associates objects in the training set with latent vectors. For the chairs dataset, our training set consisted of 5827 chairs with fine meshes, which were preprocessed into SDFs for training. The model was trained for 1000 epochs; see [22] for more details. For our simulations, we initialized a fluid block above the chair; we then use the DeepSDF model $f_\theta(p, z)$ (where p is the query point and z the shape latent code) to generate SDF features for our model.

D.5. Mountain Scene

The mountain scene was obtained from the freely available mesh here, and captures the Puncak Jaya mountains in Indonesia. We train a SIREN model on the mountain scene, which scales the scene to a $[-1, 1]^3$ box. To this SDF we apply a uniform scaling factor of $s = 20$, which results in a scene of dimension $40 \times 27 \times 6\ m^3$. We position 25 fluid blocks above the scene, each consisting of 2744 fluid particles, for a total of 68,600 fluid particles in the scene.

E. Experimental Details

E.1. Model Details

Input features. The input to the model is the current particle positions, as well as the SDF function and transformation parameters (rotation and translation). We also provide a short history of $T = 4$ previous time-steps, which is used in Sanchez-Gonzalez et al. [25]. In the 'encoder' step, we add several features to the node embeddings v_i including the previous velocities and distances to the container faces (see [25] for details). Additionally, we compute the SDF and its gradient on the current particle positions. As noted above, we can obtain these features from either neural or analytical SDFs. When computing the SDF, we first apply the inverse transformation to the query points. The edge features e_{ij} are initialized to contain the distance and displacement between v_i and v_j.

Model Architecture. We use a graph network architecture with ten message-passing steps. Please see Sanchez-Gonzalez et al. [25] for more details, which we leave largely unchanged.

Training Details. We train all our models for 2M steps, using Adam with a decaying learning rate schedule from 1e-4 to 1e-6. Noise is added to the input positions as is standard to improve stability. Details can be found in [25]. For the weighting of particles near the surface, we use $\lambda = 5$. We use $\alpha = 0.09$ for the threshold in Eq. (1).

E.2. Metrics: 'Chamfer Surface' Distance

Chamfer distance is computed *per-timestep* and averaged across time-steps. For the Chamfer Surface metric, at each time-step we consider $P_{\alpha'}$, the set of particles in P that have SDF value $F(p) < \alpha'$ in the predicted point cloud. We then measure the Chamfer distance of these close-to-surface particles to all of G. Similarly, we compute the Chamfer distance of $G_{\alpha'}$ to all of P. The metric becomes:

$$CD(P, G) = \sum_{x \in P_{\alpha'}} \min_{y \in G} ||x - y||_2^2 + \sum_{y \in G_{\alpha'}} \min_{x \in P} ||x - y||_2^2$$

Note that distances are summed across particles, not averaged as in Chamfer distance. The intuition is that if P diverges significantly from G near the surface, then the average distance between any particle in $P_{\alpha'}$ and any particle in G should be higher. By 'zooming in' only on particles near or inside the surface, we can understand errors near the surface more effectively.

F. Inverse Design: Further Details

Parametrized Design Tasks. For inverse design with parametrized shapes, we consider the tasks of designing 'bowls' and 'funnels', optimizing the capture radius r_1 and the filter radius r_2 of a parametrized SDF (see main paper for details). For both tasks, we obtain gradients of the task-specific objective function with respect to r_1 and r_2, and use Adam with a learning rate of 0.01 for optimization. For both tasks, we run the optimization for 100 steps.

Bowl task. The reward is measured as the log probability of a Gaussian centered at the bottom of the object, i.e. $N((0., 0., h), \Sigma)$, where h is the height of the object. This incentivizes the design to collect fluid inside the object. We use a spherical covariance with $\sigma = 0.8$. For a given design, the forward model is rolled out for 50 steps and the reward is measured on the final particle positions. If the filter radius r_2 reaches 0 during the optimization, it is then fixed at 0 and then only the capture radius r_1 is optimized.

Funnel task. The reward is measured as the log probability of a 2D Gaussian $N((0., 0.), \Sigma)$ at the bottom of the container, centered at the origin. This incentivizes the design to concentrate fluid onto the ground. We use a spherical covariance with $\sigma = 0.75$. For a given design, the forward model is rolled out for 75 steps. We measure the reward on any particle that reaches the bottom of the container, and subsequently remove this particle from the simulation.

In the results shown in the main paper, we initialize $r_1 = 0.07$ and $r_2 = 0.05$ for the bowl task, which neither captures nor contains the fluid; the final solution is $r_1 = 0.232$ and $r_2 = 0$, which both captures and contains the fluid. Similarly for the funnel, we initialize $r_1 = 0.10$ (smaller than desired) and $r_2 = 0.09$ (larger than desired); the final solution is $r_1 = 0.221$ and $r_2 = 0.0714$, which captures and more narrowly filters the water. Fig. 16 shows that our design optimization is robust to initialization. See the website for videos of the initial and final designs and the associated fluid rollouts.

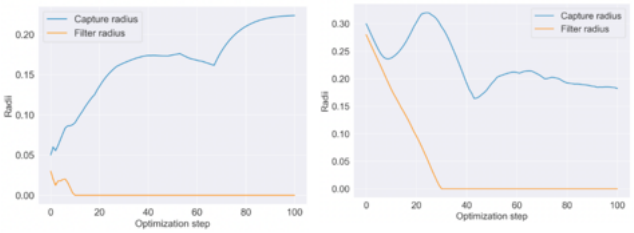


Figure 16: Different initializations for the bowl task; for both initializations the optimization converges to a sufficiently high capture radius r_1 and zero filter radius r_2.

Latent Space Design. We also perform latent space optimization with a DeepSDF model trained on ShapeNet chairs, to discover a chair that can maximize the 'bowl reward' described above (with $\sigma^2 = 0.6$). We initialize the latent code from $\mathcal{N}(0, 0.01^2)$, the prior for the DeepSDF autodecoder. Empirically we find that the optimization is robust to initialization. We run the optimization for $\sim$ 300 steps; in each step we roll out the simulator for 100 timesteps, compute the reward, and backpropagate gradients to the latent code. We use Adam with a learning rate of 0.001.