# Neural Kinematic Bases for Fluids

ANONYMOUS AUTHOR(S)
SUBMISSION ID: 399

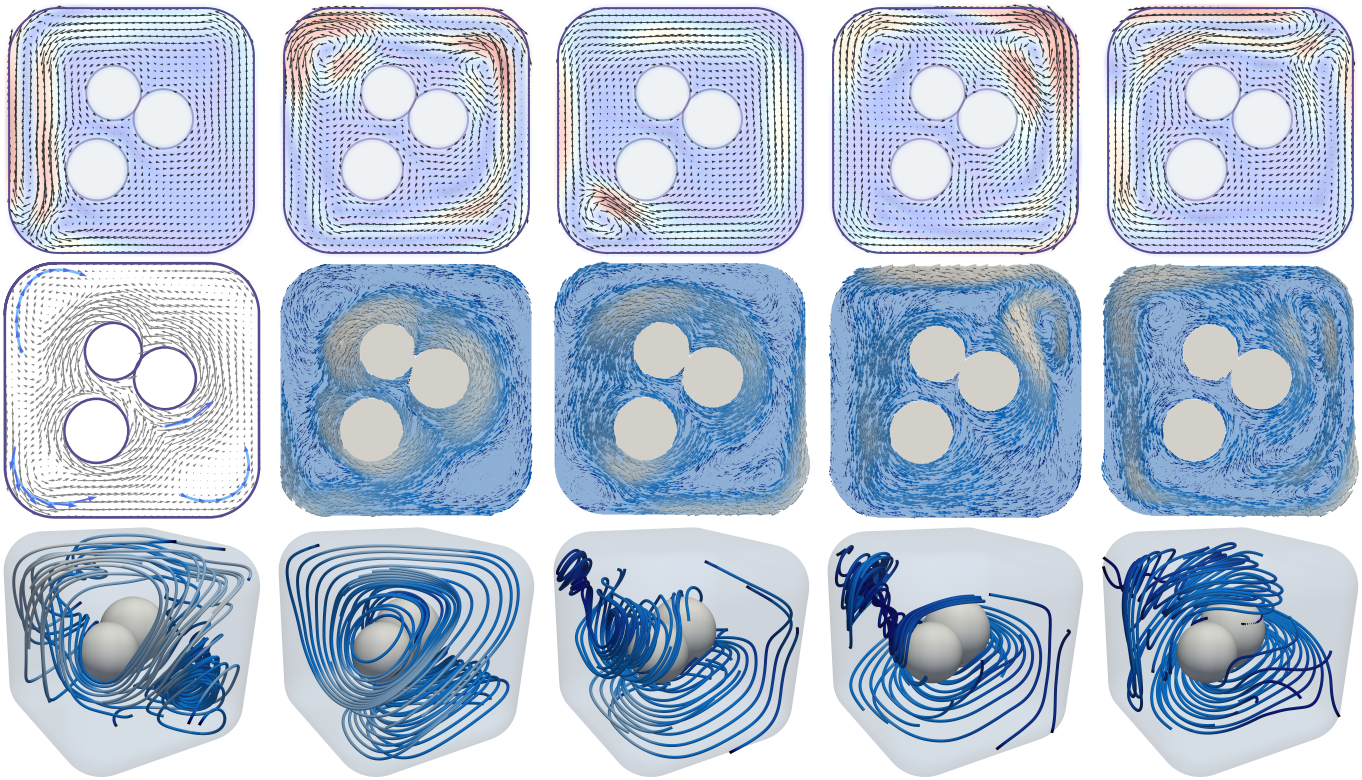

Fig. 1. We use neural kinematic bases (top) to fit an input sketch of a flow (middle, left) which we then simulate in real-time with standard semi-implicit advection. We extend the same construction to three dimensions where we can simulate fluids in real time (bottom).

We propose mesh-free fluid simulations that exploit a kinematic neural basis for velocity fields represented by an MLP. We design a set of losses that ensures that these neural bases satisfy fundamental physical properties such as orthogonality, divergence-free, boundary alignment, and smoothness. Our neural bases can then be used to fit an input sketch of a flow, which will inherit the same fundamental properties from the bases. We then can animate such flow in real-time using standard time integrators. Our neural bases can accommodate different domains and naturally extend to three dimensions.

**ACM Reference Format:**
Anonymous Author(s). 2025. Neural Kinematic Bases for Fluids. *ACM Trans. Graph.* 1, 1 (January 2025), 8 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn


ACM 0730-0301/2025/1-ART
https://doi.org/10.1145/nnnnnnn.nnnnnnn

## 1 INTRODUCTION

Animating 2D or 3D shapes from *sparse* data is a complex problem at the core of computer graphics which started with pioneering work such as that of Badler and Morris [1982]. The main motivation is character animation, where the complex animation and deformation are driven by a skeleton (sparse data) [Lewis et al. 2000]. The general problem can be abstracted as computing a sparse representation (or basis) that allows interactive animation and shape deformation. The main challenge is to ensure that any generated deformation looks physically realistic, which can be enforced by carefully designing the bases. With the advent of machine learning in character animation, the effort moved from creating bases to designing networks for such bases [Bertiche et al. 2021; Kavan et al. 2024; Li et al. 2021].

In parallel, machine learning enabled replacing complicated and costly accurate physical simulations by cheap networks [Du 2023], in particular for fluid simulations (Section 2). However, these methods share the same “problems” as physical simulations: the input is the state of the system (initial and boundary condition), and the output is an animation. This setup lacks the fine-grained control present in animation pipelines and the ability to control it from sparse data.

We propose a novel neural kinematic basis that allows animators to quickly and interactively design and build fluid animation in two and three dimensions. Our idea is to borrow the interactivity and ease of traditional animation pipelines and combine them with the power of neural representation used in physical simulations. Similar to skeletal animation, we aim to design a basis that encodes the complex dynamics of the fluid. We represent our neural bases with an MLP, which we train purely using fundamental physical properties such as incompressibility, smoothness, boundary alignment, and orthogonality; therefore, we do not require any ground truth. Our training data is small and is only used to allow our neural bases to generalize to the new unseen domains. Since we force the MLP network to encode fundamental laws, any animation generated from our bases will also inherit the same properties. Once we fit the initial flow, we can use standard integration techniques to develop a plausible fluid animation (Figure 1).

## 2 RELATED WORK

Reduced-order models in fluid dynamics trace back to Lumley [1967]. In computer graphics, a similar concept using principal component analysis (PCA) was introduced by Pentland and Williams [1989] as a method for reducing degrees of freedom in the deformation of solids. This foundational idea spurred extensive follow-up work, extending deformable models. For instance, Barbič and James [2005] proposed a fast subspace integration method for reduced-coordinate nonlinear deformable models, leveraging mass-scaled PCA to generate low-dimensional bases. They demonstrated that model reduction allows internal forces to be precomputed as cubic polynomials, enabling efficient simulations with costs independent of geometry. Building on PCA, Treuille et al. [2006] extended the approach to fluid dynamics, constructing a reduced-dimensional velocity basis by applying PCA to velocity fields from full-dimensional simulations.

To address computational challenges in PCA-based methods, An et al. [2008] introduced optimized cubature schemes for efficient integration of force densities associated with specific subspace deformations. Similarly, Kim and James [2009] developed an online, incremental reduced-order nonlinear model. Later, Kim and Delaney [2013] extended cubature schemes to efficiently perform consistent semi-Lagrangian advection within a fluid subspace, enabling re-simulation of fluid systems with modified parameters. In comparison, our work employs sparse integration techniques and semi-Lagrangian advection.

Von Tycowicz et al. [2013] accelerated the construction of reduced dynamical systems by approximating reduced forces. Our work also incorporates approximations, using smoothed boundary indicators and domain masks. Additionally, De Witt et al. [2012] represented fluids as linear combinations of eigenfunctions of the vector Laplacian, performing time integration via Galerkin discretization of the Navier-Stokes equations. While this method supported immersed rigid bodies by projecting out velocity components corresponding to boundary flux, Cui et al. [2018] extended the approach to handle Dirichlet and Neumann boundary conditions. They further improved scalability, following Jones et al. [2016], by utilizing sine and cosine transforms to reduce storage demands for eigenfunctions. Inspired by Laplacian eigenfunction methods, Mercier and Nowrouzezahrai [2020] and Wicke et al. [2009] developed reduced basis functions on regular tiles, enforcing consistency constraints between adjacent tiles. Our approach adopts a basis-function perspective but employs neural representations instead of eigenfunctions.

Yang et al. [2015] identified modal matrix construction, cubature training, and dataset generation as key bottlenecks in traditional approaches. In contrast, our method trains basis functions to preserve geometric invariants in fluid simulations. Similarly, Xu and Barbič [2016] precomputed separate reduced models for different object poses and combined them at runtime into a unified dynamic system. We build on this idea by utilizing local geometric invariants to address complex nonlinear spaces.

Romero et al. [2021] augmented linear handle-based subspace formulations with nonlinear, learning-based corrections to decouple internal and external contact-driven effects. In our work, contact handling is achieved by conditioning neural basis functions on contextual information. Likewise, Aigerman et al. [2022] used neural networks to predict piecewise linear mappings of arbitrary meshes, incorporating smoothing techniques to handle discontinuities. We adopt a similar gradient-smoothing strategy to address these challenges.

In the domain of Eulerian fluid configurations, Wu et al. [2023] learned latent space embeddings and applied linear time integration operators based on the sines and cosines of latent variables. Further, Chen et al. [2023b] and Chang et al. [2023] introduced discretization-independent reduced-order modeling, representing displacement fields as continuous maps encoded by implicit neural fields. Extending these ideas, Chen et al. [2023a] applied neural fields to the material point method, while Tao et al. [2024] proposed neural implicit reduced fluid simulations, using non-linear latent embeddings to capture fluid-fluid interactions.

Our work follows the neural field paradigm to represent learned basis functions, enabling robust and efficient reduced-order fluid simulations while preserving geometric and physical invariants.

## 3 METHOD

The core idea of our approach is to design a set of nonlinear neural basis functions (Figure 2) that respect common invariants in fluid flow problems. Observe that our neural bases, $\varphi_i$, are vector fields. This is a little different from finite element approaches where the shape functions would be scalar functions and the unknown coefficient would be vectors living at nodal points. Let $\Omega$ be a unit rounded square domain with $m$ potentially overlapping circular holes (Figure 2). On this domain, we define $b$ neural basis functions $\varphi_k, k = 1 \dots b$ that satisfy the following common invariants.

*Divergence.* Crucially, for incompressible fluid simulation (inviscid and otherwise), we require that our basis can reconstruct divergence-free velocity fields:

$$\int_\Omega \operatorname{div}(\varphi_k) = 0, \qquad \forall k = 1 \dots b.$$

*Boundary.* To ensure that the fluid remains inside the domain, we restrict our neural bases to satisfy the slip boundary condition

$$\int_{\partial\Omega} \langle n, \varphi_k \rangle = 0, \qquad \forall k = 1 \dots b,$$

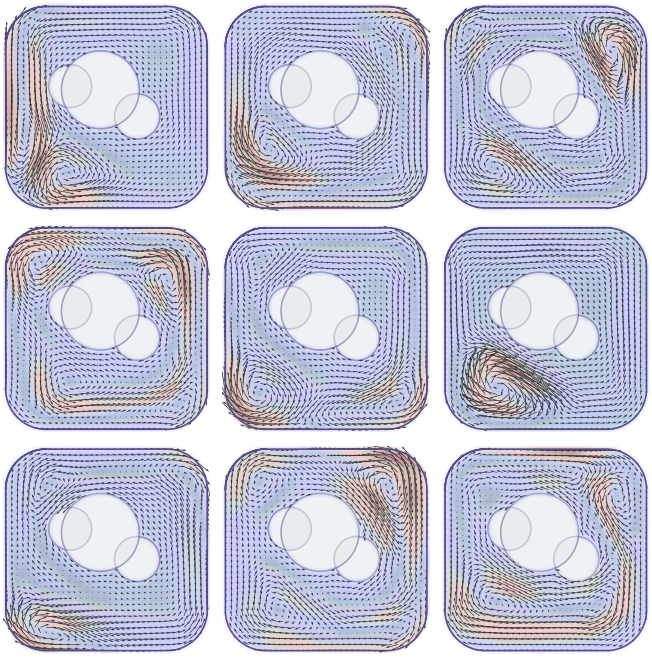

Fig. 2. Example of neural basis functions generated by our MLP network. We clearly see that they are parallel to the boundary, non-zero, and smooth.

with $n$ the normal on the boundary.

*Orthonormality.* Bases need to be linearly independent, additionally, we require them to be orthogonal to each other,

$$\int_\Omega \langle \varphi_k, \varphi_l \rangle = \delta_{k,l}, \qquad \forall k, l = 1 \dots b.$$

Using these bases, we can approximate any field velocity

$$v(p) = \sum_{k=1}^{b} \varphi_k(p)\alpha_k \tag{1}$$

which will obviously satisfy the above invariants and lead to a plausible fluid simulation that can be integrated with standard semi-implicit time integrators. Here, $\alpha_i$ are the unknown coefficients we solve for during simulation. Their role will be similar to the weights of eigenmodes or coefficients of a finite element method. Note that, differently from traditional finite element bases, the velocity $v$ will satisfy common invariants independently from the choice of $\alpha_i$. Our $\alpha_i$ does not live at a nodal position in space but exists in an abstract global setting. Our choice aggressively reduces the degrees of freedom.

Our goal is to learn a set of neural basis functions $\varphi_i$ dependent on both the position of the circles (which define the domain) and the evaluation point, such that we can use (1) as the *kinematic neural basis* for fluid simulation. By doing so, we can quickly evaluate specific bases (as the evaluation is done at inference time) for a given domain and thus generate fluid animation in real-time. Our design pipeline (Figure 3) starts with an input sketch of the domain $\Omega$ with a set of curves to guide the flow which we fit to our neural bases (Section 3.3). To obtain the neural bases, we define a set of

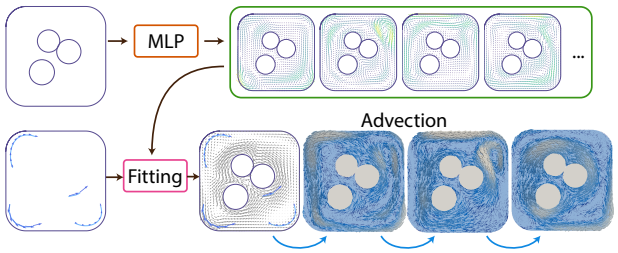


Fig. 3. Overview of the pipeline of our method. We start with a sketch and our pre-trained MLP that generates fluid bases. We fit the bases to the input sketch which we then advect to generate an animation.

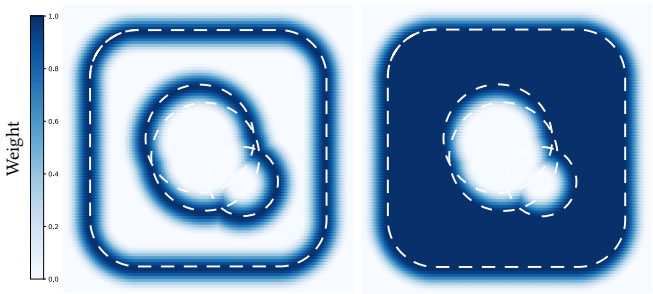


Fig. 4. Boundary indicator function $w_b$ (left) and domain mask $w$ (right).

losses (Section 3.1) that produces bases that satisfy fluid invariants, which we train on different domains (Section 3.2). Finally, we advect the initial fluid (Section 3.4) to generate an animation of the fluid.

## 3.1 Losses

We will proceed by defining a set of losses that measure the physical appropriateness of our kinematic neural basis. Our basis neural fields $\varphi_i$ will be computed by minimizing these losses in aggregate over a collection of $n$ randomly sampled points $p_i$. This sampling can be seen as using Monte Carlo integration. The main advantage of our approach is that we purely rely on fundamental physical properties (e.g., divergence-free or slip boundary conditions) and do not require any ground-truth data; we only require the bases to generalize over the position and size of the circles.

The input of our MLP network $F$ is an evaluation point $p \in \mathbb{R}^2$ and the center $c$ and radius $r$ of $m$ circles; it produces a set of $b$ basis functions $\varphi_k(p)$. That is,

$$F(p, \rho, \theta) = \{\varphi_k(p)\}_{k=1,\dots,b},$$

where $\rho = \{c_i, r_i\}_{i=1}^m$ is the circle set, $\theta$ the MLP parameters, and $\varphi_k(p)$ our neural bases evaluated at $p$.

Since we sample the plane independently from the domain, we apply a soft mask to filter out the out-of-domain points. We first define a boundary indicator function (Figure 4, left)

$$d_b(p) = \left( \frac{2|d(p)|^3}{\varepsilon^3} - \frac{3|d(p)|^2}{\varepsilon^2} + 1 \right)$$

$$w_b(p) = \begin{cases} 0, & |d(p)| > \epsilon \text{ or } d_b(p) < 0 \\ d_b(p)^4 & \text{otherwise} \end{cases}$$

where $d(p)$ is the distance between $p$ and the boundary of $\Omega$. Then, we define the mask $w$ (Figure 4, right) by setting all values of $w_b$ out of the domain to zero and the values in the domain to 1. This function is one inside the domain and drops to zero for any point farther than $\varepsilon$ to the boundary. Note that the function is non-zero on a small region outside the domain, and therefore, we approximate our integral on the $[-\varepsilon, 1+\varepsilon]^2$ domain. To ensure that our losses are independent from the number of points and their position, we normalize them by

$$S = Wb, \qquad \text{with} \qquad W = \sum_{i=1}^{m} w(p_i).$$

We can now encode the divergence invariant as the loss

$$\mathcal{L}_{\text{div}}(\theta) = \sum_{k=1}^{b}\sum_{i=1}^{n} \text{div}(\varphi_k(p_i))^2 w(p_i)/S. \tag{2}$$

Interestingly, we do not require the individual $\varphi_i$ to be divergence-free but wish for an expressive basis that can produce divergence-free fields (e.g., under different boundary conditions).

Next, we formulate a loss for the slip boundary conditions

$$\mathcal{L}_{\text{bc}}(\theta) = \sum_{k=1}^{b}\sum_{i=1}^{n} \text{cossim}(\varphi_k(p_i), n(p_i))^2 w_b(p_i)/S_b \tag{3}$$

where

$$\text{cossim}(a,b) = \frac{\langle a, b\rangle}{\|a\|\|b\|}$$

is the cosine similarly, and $n(p_i)$ is the normal of the closest point on the boundary of the domain. Since this is a discretization of a boundary integral (with a different weighting function), we normalize this loss by $S_b = b\sum_i w_p(p_i)$.

Finally, to prevent all bases from being the same, we enforce an average orthogonality using

$$\mathcal{L}_{\text{orth}}(\theta) = \sum_{i=1}^{n}\sum_{k=1}^{b}\sum_{l=k+1}^{b} \langle \varphi_k(p_i), \varphi_l(p_i)\rangle w(p_i)/S_o. \tag{4}$$

Instead of normalizing by the number of bases, we normalize by the number of pairs $p$ ($S_o = Wp$). We note that this loss does not require that the bases have unit length $\langle \varphi_k(p_i), \varphi_k(p_i)\rangle = 1$ as the third sum starts at $k+1$. Instead, we explicitly require the average length of the vectors by requiring that the length is close to a target value $c$

$$\mathcal{L}_{\text{len}}(\theta) = \sum_{k=1}^{b}\left(\sum_{i=1}^{n} \|\varphi_k(\theta, p_i)\| w(p_i)/W - c\right)^2 /b, \tag{5}$$

and penalize small bases

$$\mathcal{L}_{\text{small}}(\theta) = \sum_{k=1}^{b}\sum_{i=1}^{n} \text{ReLU}(\delta - \|\varphi_k(p_i)\|) w(p_i)/S. \tag{6}$$

Finally, to facilitate the learning process, we also encourage smoothness of the basis by adding

$$\mathcal{L}_{\text{smooth}}(\theta) = \sum_{k=1}^{b}\sum_{i=1}^{n} \|J_{\varphi_k}(p_i)\|_F^2 w(p_i)/S, \tag{7}$$

where $J_{\varphi_k}(p_i)$ is the Jacobian matrix.

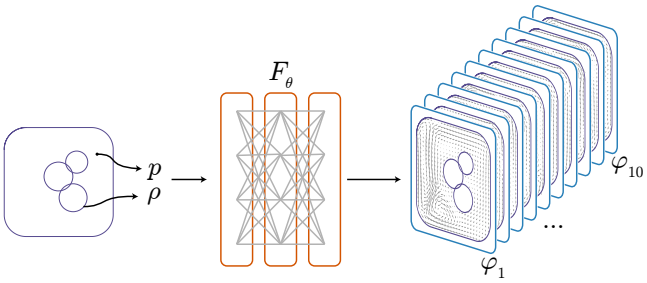


Fig. 5. Overview of our MLP architecture: it takes as input a point $p$ and the set of circles' parameters $\rho$ and produces our neural bases $\varphi_i$, $i = 1, \ldots, b$.

We sum the aforementioned 6 losses with their own respective weight to formulate the fluid loss $\mathcal{L}_{\text{fluid}}$.

### 3.2 Training

We limit the training to a domain represented by three circles and compute the distance $d(p)$ using a soft minimum to the 3 circles. Our neural fields are parameterized by MLPs that have 8 fully connected layers, which use the leaky ReLU activation function (except ELU for the last layer as we want to produce negative numbers) and have 256 channels per layer (Figure 5). The input vector is the concatenation of the position of one sample point $p_i$ and 3 circle parameters. The output vector consists of the vector for the evaluation of the $b$ neural bases.

The weights for terms in the fluid loss are $w_{\text{drch}} : w_{\text{div}} : w_{\text{orth}} : w_{\text{bc}} : w_{\text{len}} : w_{\text{small}} = 0.01 : 5 : 100 : 50 : 100 : 100$. We choose the threshold for small base penalty $\delta$ as 0.05 and the average length for length penalty as $c = 0.37$.

The MLP uses Kaiming initialization for its parameters. We train the MLP using the Adam optimizer with a learning rate initialized at 0.001. To dynamically adjust the learning rate, we employed an ExponentialLR scheduler, applying a decay factor of 0.96, resulting in a learning rate scaled by $LR \times 0.96^t$ per epoch, where $t$ is the epoch.

We train on 1000 different geometric samples, each of which is a set of randomly generated 3 circles with centers in the $[0.25, 0.75]^2$ square and radius in the $[0.1, 0.3]$ interval. They can be (partially) overlapping or separate. For each sample, we randomly sample $n = 10^6$ points over the domain. The training was conducted for a total of 10 epochs with a batch size of $10^4$.

We train the boundary loss in two stages: first, we use a power of 4 in the cosine similarity, then lower it to 2. This approach leads the MLP to first remove outliers and have a correct alignment close to the boundary, Figure 6 left (higher power leads to a narrower band). By then lowering it, the MLP can relax the condition and propagate the aligned to a larger (and smoother region), Figure 6, right.

### 3.3 Fitting to a Sketch

We now aim to use our neural bases to animate an input image; the image contains the position of the circles and several streamlines represented by parametric curves $\gamma(t)$. We start by drawing the circles and a few curves, and this quickly generates a velocity field;

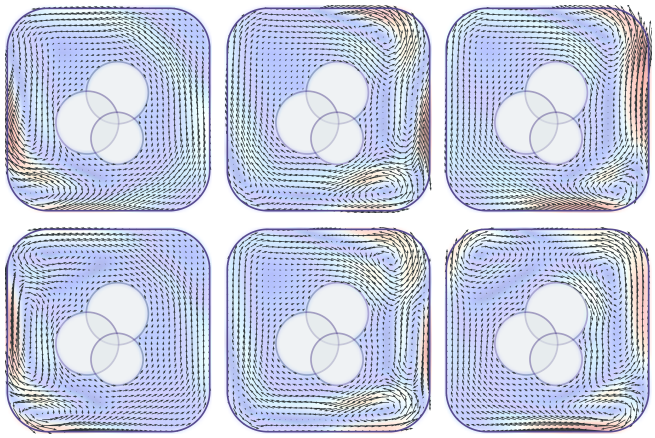

Fig. 6. Visualization of three bases at the end of the first stage when the power is four, top. After lowering the power to two, the bases align well with the boundary, bottom.

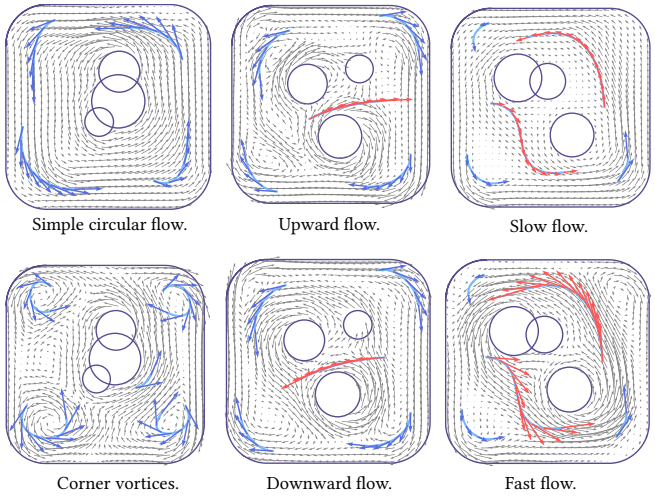


Fig. 7. Example of the different interactions possible to design a fluid flow. From the initial design to different types of editing.

by adding additional curves, we can interactively refine the flow by, for instance, adding vortices or changing the flow direction. Figure 7 demonstrates how the interactive sketches control the initial flow. Our bases can fit a variety of curves in a smooth manner; for instance, they fit a simple circular flow, or they allow the creation of many vortices and can control the direction or intensity of the flow.

To generate the input flow, we sample every curve $\gamma$ at $c_i$ points uniformly spaced in parametric space and compute the target velocity $t_i = \nabla\gamma(c_i)/\|\nabla\gamma(c_i)\|$ as the normalized tangent at $c_i$. We use this set of points and velocities to least-square fit an initial set of parameter $\alpha$;

$$\alpha^0 = \arg\min_{\alpha} \sum_i \left\| \sum_{k=1}^{b} \varphi_k(c_i)\alpha_k - t_i \right\|^2 .$$

### 3.4 Advection

We then use $\alpha^0$ to compute the initial velocity $v^0$, which we advect using a semi-implicit integrator [Stam 1999, 2023]. For every time step $t$, for every point, we compute the origin position

$$p_o = p - v^t(p)dt,$$

where $dt$ is the time-step, note that if $p_o$ lands outside the domain, we project it to the closest point. Following the integration scheme, the new velocity is copied from the velocity at the origin position

$$\bar{v}^{t+1}(p) = \sum_{k=1}^{b} \varphi_k(p_o)\alpha_k^t.$$

This new velocity might not be representable by our bases; therefore we compute the new $\alpha$ by least-square fit $\bar{v}^{t+1}(p)$ into our bases

$$\alpha^{t+1} = \arg\min_{\alpha} \sum_i \left\| \sum_{k=1}^{b} \varphi_k(p_i)\alpha_k - \bar{v}^{t+1}(p_i) \right\|^2 ,$$

and use it to compute

$$v^{t+1} = \sum_{k=1}^{b} \varphi_k(p_i)\alpha_k^{t+1}.$$

## 4 RESULTS

We will show how our neural bases generalize to different domains (Section 4.1), how the different losses converge during the training process (Section 4.2), and how the different losses contribute to how the bases behave (Section 4.3). Finally, we present two- and three-dimensional animations by fitting our kinematic neural bases and integrating them with a semi-implicit integrator (Section 4.4).

### 4.1 Generalization

We evaluate our neural bases on a test dataset comprised of 100 random unseen circles with centers and radii sampled from the same distribution (figures 8-10, purple, show the value of the different losses during training). At the end of the training, our neural bases have similar losses: the length losses and orthogonality are practically the same, while the divergence and boundary loss are just around 15% larger.

### 4.2 Convergence

Overall, our fluid loss steadily decreased over the training process. Note that we use blue vertical lines to indicate when the second stage of our training starts. After an initial phase, both length losses and orthogonality become stationary, indicating that the bases reached the desired length, are individually not zero, and are different (Figure 8). It is interesting to note that subsequent epochs do not change their value, indicating that the training mostly rotates the vectors. The orthogonality loss and the smoothness loss (which depend on the gradient of the MLP and impose first-order conditions) require more steps (400 thousand) to reach a stationary value, suggesting that our bases maintain the fluid properties and are smooth in most of our training process (Figure 9). Both losses are affected by the two-stage training, but they quickly reach a new stable flat state. We note that, as expected, the smoothness loss increases in the beginning as the bases become larger. Additionally, the smoothness loss

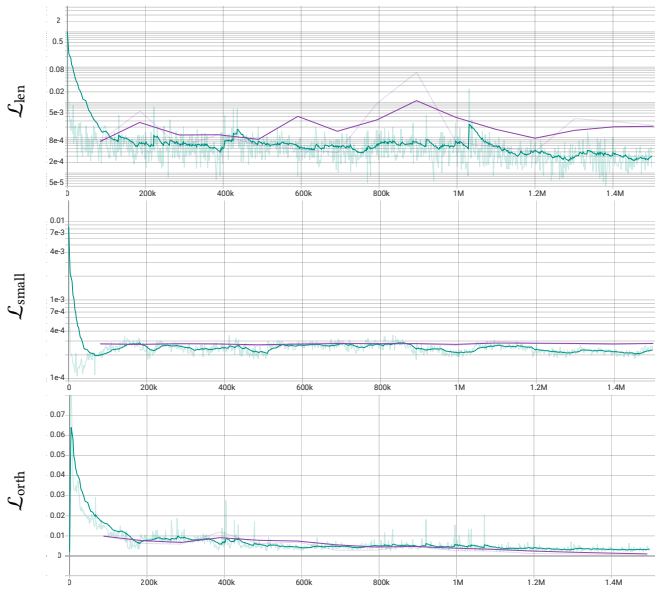


Fig. 8. Convergence of the length-based losses. After the first 100 thousand steps, both losses are converged.

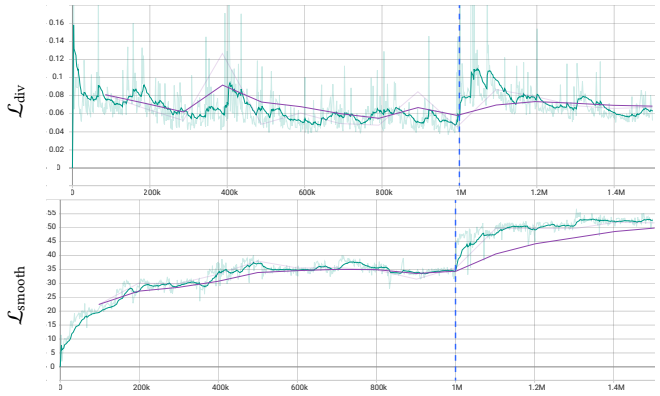


Fig. 9. Convergence of the "gradient" based losses. These losses impose conditions on the gradient and thus require longer to converge.

is the only loss that can not reach a minimum of zero, as it would lead to constant bases. The boundary condition loss is the only one affected by the two-stage training: after the first 10 epochs, we decrease the power of the cosine similarity (Figure 10). We clearly see the "jump" in the loss when we switch. After the initial increase, the loss quickly decreases and reaches similar values.

## 4.3 Ablation study

We showcase the importance of three main features of our pipeline (Figure 11). To ease the comparison, we limit the dataset to 3 different circles as we do not test for generalization.

*Smoothness loss.* This is the only loss that is not motivated by fundamental physical law. We included it to bias the MLP towards smooth solutions (which our bases must be). Adding it improves

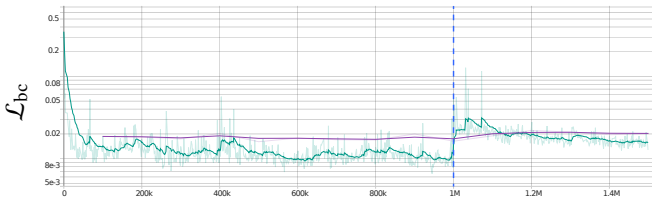


Fig. 10. Convergence of the boundary loss. When decreasing the power (blue line), the loss increases to then quickly drops again.

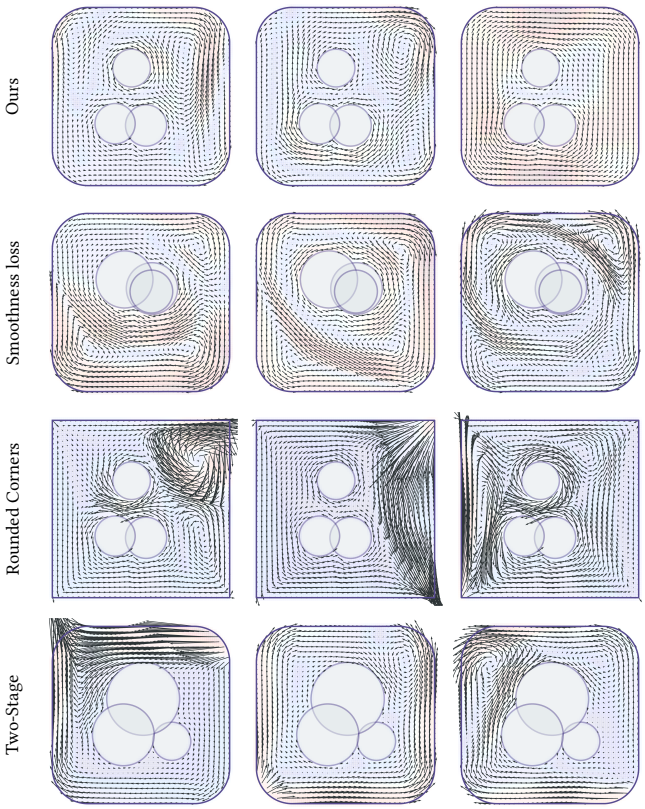


Fig. 11. Bases generated by omitting different parts of our pipeline. Each omission leads to different artifacts.

the reliability of the training process: in our experiments, without it, we obtain unreasonable bases more frequently (around 30% more).

*Rounded Corners.* We decided to smooth the corners of the domain to ensure that we can generate a smooth solution. Without it, the training fails to transition around the corners (where there should be a discontinuity) and generate bases that push the flow outside the domain.

*Two-stage training for boundary.* As previously mentioned, we train our MLP network in two phases. Without this approach, the bases do not align with the boundary (for power of 2) or lack smoothness (for power of 4).

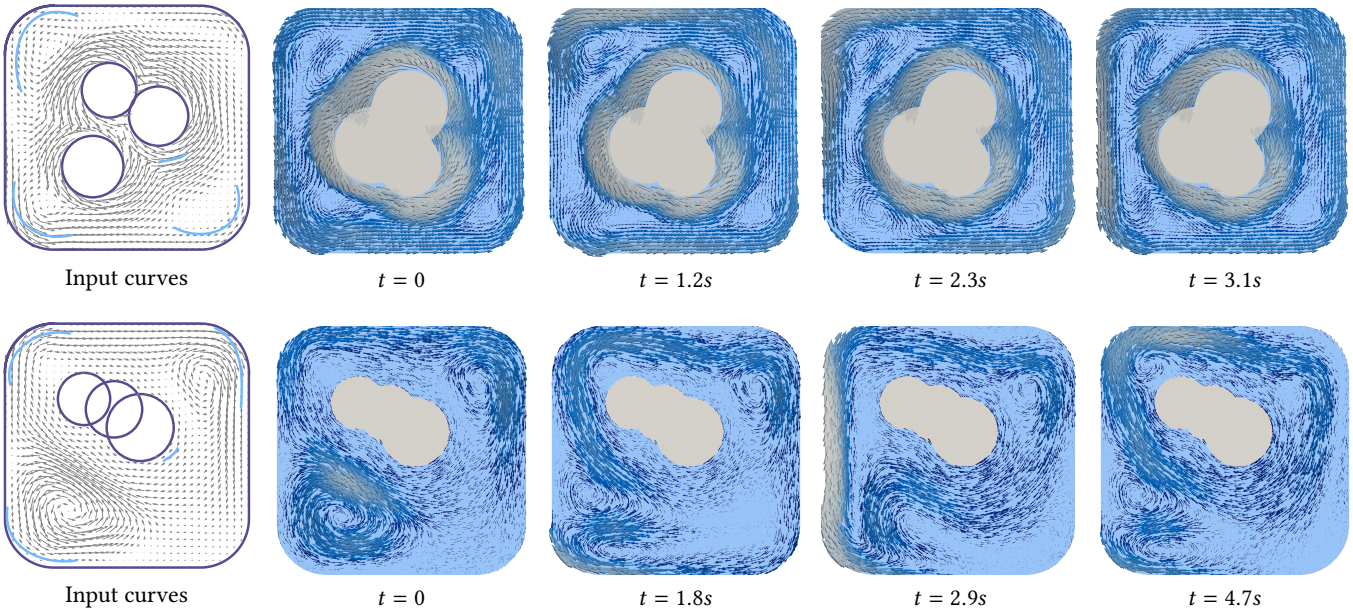


Fig. 12. Example of different fluid animation frames for different input sketches. The flow adapts to the position of the circles and the curves and produces natural animation.

### 4.4 Animations

We run all our experiments on a GeForce RTX 3090. Generating the initial fluid is interactive, while the simulation runs at 40 frames per second for 250 thousand points.

*Two-dimensional.* Figure 12 shows how the input sketch leads to a realistic fluid simulation. If the input flow is tamed, the simulation simulation converges to a calm stationary velocity. By starting with more vortices, the flow remains turbulent even in later frames.

*Three-dimensional.* All our methods can be generalized to three dimensions. The domain is now represented by three spheres and their radii (12 parameters instead of 9); the MLP network accepts 3D points and produces a volumetric vector field. Since all losses are physically motivated, they naturally extend to arbitrary dimensions. Similarly, the advection of the fluid is dimension agnostic. Since sketching volumetric curves is more complicated, we initialize the flow using random $\alpha$. Figure 13 shows the results for three overlapping spheres, while Figure 1 shows how the flow passes between the spherical holes. Both animations are run in real-time as the MLP is only evaluated. Both figures show the velocity field both as arrows and streamlines colored by their velocity.

## 5 CONCLUSIONS

We introduce a novel neural kinematic base represented by an MLP that captures fundamental physical properties and generalizes to different domains. We show how using our bases and a standard advection integrator, we can generate two- and three-dimensional animations.

To keep the training times reasonable, we decided to use only three circles and ten bases, but our method should be able to generate more bases for more complex domains. Additionally, we could represent the domain as a collection of capsules, thus allowing for a more detailed domain geometry. We note that if we have more than three primitives, the domain might become disconnected, and it would be interesting to see how our bases could capture such scenarios.

Our current setup restricts the bases to align with the boundary, which is a typical computational fluid dynamics boundary condition. It would be interesting to extend our bases to support, for instance, no slip or an inflow/outflow boundary. This would require a user interface design to control the boundary condition and a mechanism to blend bases based on the input.

Finally, since our bases are encoded with an MLP, they are fully differentiable. Inverse design, for instance, could be an interesting avenue for future work. For instance, it could involve optimizing the position of the circle to match a given animation.

## REFERENCES

Noam Aigerman, Kunal Gupta, Vladimir G. Kim, Siddhartha Chaudhuri, Jun Saito, and Thibault Groueix. 2022. Neural jacobian fields: learning intrinsic mappings of arbitrary meshes. *ACM Trans. Graph.* 41, 4, Article 109 (July 2022), 17 pages. https://doi.org/10.1145/3528223.3530141

Steven S. An, Theodore Kim, and Doug L. James. 2008. Optimizing cubature for efficient integration of subspace deformations. *ACM Trans. Graph.* 27, 5, Article 165 (Dec. 2008), 10 pages. https://doi.org/10.1145/1409060.1409118

Norman I Badler and Mary Ann Morris. 1982. Modelling flexible articulated objects. In *Proc. Computer Graphics' 82, Online Conf.* 305–314.

Jernej Barbič and Doug L. James. 2005. Real-Time subspace integration for St. Venant-Kirchhoff deformable models. *ACM Trans. Graph.* 24, 3 (July 2005), 982–990. https://doi.org/10.1145/1073204.1073300

Hugo Bertiche, Meysam Madadi, Emilio Tylson, and Sergio Escalera. 2021. DeePSD: Automatic Deep Skinning and Pose Space Deformation for 3D Garment Animation.

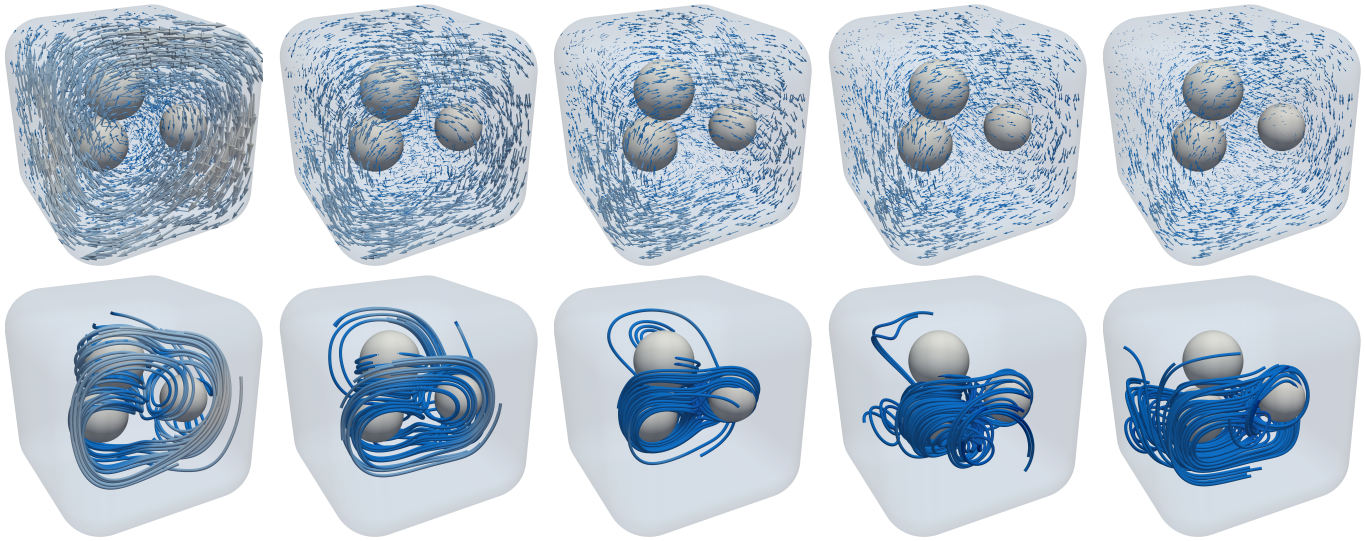


Fig. 13. Three-dimensional fluid animation using our neural kinematic bases. We show both the velocity and streamlines colored by their velocity.

In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. 5471–5480.

Yue Chang, Peter Yichen Chen, Zhecheng Wang, Maurizio M. Chiaramonte, Kevin Carlberg, and Eitan Grinspun. 2023. LiCROM: Linear-Subspace Continuous Reduced Order Modeling with Neural Fields. In *SIGGRAPH Asia 2023 Conference Papers* (Sydney, NSW, Australia) *(SA '23)*. Association for Computing Machinery, New York, NY, USA, Article 111, 12 pages. https://doi.org/10.1145/3610548.3618158

Peter Yichen Chen, Maurizio M. Chiaramonte, Eitan Grinspun, and Kevin Carlberg. 2023a. Model reduction for the material point method via an implicit neural representation of the deformation map. *J. Comput. Phys.* 478, C (April 2023), 30 pages. https://doi.org/10.1016/j.jcp.2023.111908

Peter Yichen Chen, Jinxu Xiang, Dong Heon Cho, Yue Chang, G A Pershing, Henrique Teles Maia, Maurizio M Chiaramonte, Kevin Thomas Carlberg, and Eitan Grinspun. 2023b. CROM: Continuous Reduced-Order Modeling of PDEs Using Implicit Neural Representations. In *The Eleventh International Conference on Learning Representations*. https://openreview.net/forum?id=FUORz1tG8Og

Qiaodong Cui, Pradeep Sen, and Theodore Kim. 2018. Scalable laplacian eigenfluids. *ACM Trans. Graph.* 37, 4, Article 87 (jul 2018), 12 pages. https://doi.org/10.1145/3197517.3201352

Tyler De Witt, Christian Lessig, and Eugene Fiume. 2012. Fluid simulation using Laplacian eigenfunctions. *ACM Trans. Graph.* 31, 1, Article 10 (feb 2012), 11 pages. https://doi.org/10.1145/2077341.2077351

Tao Du. 2023. Deep Learning for Physics Simulation. In *ACM SIGGRAPH 2023 Courses* (Los Angeles, California) *(SIGGRAPH '23)*. Association for Computing Machinery, New York, NY, USA, Article 6, 25 pages.

Aaron Demby Jones, Pradeep Sen, and Theodore Kim. 2016. Compressing fluid subspaces. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (Zurich, Switzerland) *(SCA '16)*. Eurographics Association, Goslar, DEU, 77–84.

Ladislav Kavan, John Doublestein, Martin Prazak, Matthew Cioffi, and Doug Roble. 2024. Compressed Skinning for Facial Blendshapes. In *ACM SIGGRAPH 2024 Conference Papers* (Denver, CO, USA) *(SIGGRAPH '24)*. Association for Computing Machinery, New York, NY, USA, Article 47, 9 pages.

Theodore Kim and John Delaney. 2013. Subspace fluid re-simulation. *ACM Trans. Graph.* 32, 4, Article 62 (jul 2013), 9 pages. https://doi.org/10.1145/2461912.2461987

Theodore Kim and Doug L. James. 2009. Skipping steps in deformable simulation with online model reduction. *ACM Trans. Graph.* 28, 5 (Dec. 2009), 1–9. https://doi.org/10.1145/1618452.1618469

J. P. Lewis, Matt Cordner, and Nickson Fong. 2000. Pose space deformation: a unified approach to shape interpolation and skeleton-driven deformation. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '00)*. ACM Press/Addison-Wesley Publishing Co., USA, 165–172. https://doi.org/10.1145/344779.344862

Peizhuo Li, Kfir Aberman, Rana Hanocka, Libin Liu, Olga Sorkine-Hornung, and Baoquan Chen. 2021. Learning skeletal articulations with neural blend shapes. *ACM Trans. Graph.* 40, 4, Article 130 (July 2021), 15 pages.

John Leask Lumley. 1967. The structure of inhomogeneous turbulent flows. *Atmospheric turbulence and radio wave propagation* (1967), 166–178.

Olivier Mercier and Derek Nowrouzezahrai. 2020. Local Bases for Model-reduced Smoke Simulations. *Computer Graphics Forum* 39, 2 (2020), 9–22. https://doi.org/10.1111/cgf.13908

A. Pentland and J. Williams. 1989. Good vibrations: modal dynamics for graphics and animation. In *Proceedings of the 16th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '89)*. Association for Computing Machinery, New York, NY, USA, 215–222. https://doi.org/10.1145/74333.74355

Cristian Romero, Dan Casas, Jesús Pérez, and Miguel Otaduy. 2021. Learning contact corrections for handle-based subspace dynamics. *ACM Trans. Graph.* 40, 4, Article 131 (July 2021), 12 pages. https://doi.org/10.1145/3450626.3459875

Jos Stam. 1999. Stable fluids *(SIGGRAPH '99)*. ACM Press/Addison-Wesley Publishing Co., USA, 121–128.

Jos Stam. 2023. *Stable Fluids* (1 ed.). Association for Computing Machinery, New York, NY, USA.

Yuanyuan Tao, Ivan Puhachov, Derek Nowrouzezahrai, and Paul Kry. 2024. Neural Implicit Reduced Fluid Simulation. In *SIGGRAPH Asia 2024 Conference Papers (SA '24)*. Association for Computing Machinery, New York, NY, USA, Article 121, 11 pages. https://doi.org/10.1145/3680528.3687628

Adrien Treuille, Andrew Lewis, and Zoran Popović. 2006. Model reduction for real-time fluids. *ACM Trans. Graph.* 25, 3 (July 2006), 826–834. https://doi.org/10.1145/1141911.1141962

Christoph von Tycowicz, Christian Schulz, Hans-Peter Seidel, and Klaus Hildebrandt. 2013. An efficient construction of reduced deformable objects. *ACM Trans. Graph.* 32, 6, Article 213 (Nov. 2013), 10 pages. https://doi.org/10.1145/2508363.2508392

Martin Wicke, Matt Stanton, and Adrien Treuille. 2009. Modular bases for fluid dynamics. *ACM Trans. Graph.* 28, 3, Article 39 (jul 2009), 8 pages. https://doi.org/10.1145/1531326.1531345

Haixu Wu, Tengge Hu, Huakun Luo, Jianmin Wang, and Mingsheng Long. 2023. Solving High-Dimensional PDEs with Latent Spectral Models. In *International Conference on Machine Learning*.

Hongyi Xu and Jernej Barbič. 2016. Pose-space subspace dynamics. *ACM Trans. Graph.* 35, 4, Article 35 (July 2016), 14 pages. https://doi.org/10.1145/2897824.2925916

Yin Yang, Dingzeyu Li, Weiwei Xu, Yuan Tian, and Changxi Zheng. 2015. Expediting precomputation for reduced deformable simulation. *ACM Trans. Graph.* 34, 6, Article 243 (Nov. 2015), 13 pages. https://doi.org/10.1145/2816795.2818089