

A PRELIMINARIES FOR THE MATHEMATICAL TOOLS

A.1 THE DIFFERENTIAL GEOMETRY IN \mathcal{R}^n

We provide an in-depth discussion of our field construction which follows the introduction in Appendix A of Richter-Powell et al. (2022). Please refer to the work for more details. We first discuss the basic concept in the differential geometry for the introduction of differential form that supports us to derive and prove our theorem in the paper.

We take a local coordinate chart for $x \in \mathcal{R}^n$ as $x = (x_1, \dots, x_n)$ and dx_1, \dots, dx_n denotes the coordinate differentials, i.e. $dx_i(x) = x_i$, $i \in [n] = \{1, \dots, n\}$, which is also the co-vector field of the local coordinates. Note that we discuss \mathcal{R}^n as an example and all the definition can be extended for smooth manifold and well-defined but needs the construction of the local chart or other mathematical manipulations. For more extensive introduction see Do Carmo (1998); Morita (2001).

Define the linear vector space $*^k(\mathcal{R}^n)$ for \mathcal{R}^n as the space of the k -linear alternating map:

$$\phi : \overbrace{\mathcal{R}^n \times \dots \mathcal{R}^n}^{k \text{ times}} \rightarrow \mathcal{R}. \quad (17)$$

A k -linear alternating map ϕ is linear in each coordinate and satisfies the alternating property:

$$\phi(v_1, \dots, v_i, \dots, v_j, \dots, v_n) = -\phi(v_1, \dots, v_j, \dots, v_i, \dots, v_n). \quad (18)$$

The basis of the space $\Lambda^k(\mathcal{R}^n)$ can be denoted with the differentials by $dx_{i_1} \wedge \dots \wedge dx_{i_k}$. The way of the basis act on k -vectors, $v_1, \dots, v_k \in \mathcal{R}^n$ as:

$$dx_{i_1} \wedge \dots \wedge dx_{i_k}(v_1, \dots, v_k) = \frac{1}{k!} \sum_{\Upsilon \in S_k} \text{sgn}(\Upsilon) dx_{i_1} \wedge \dots \wedge dx_{i_k}(v_{\Upsilon(1)}, \dots, v_{\Upsilon(k)}) = \det[dx_{i_r}(v_s)]_{r,s \in [k]}, \quad (19)$$

where S_k is a permutation for i_1, \dots, i_k .

More specifically, for $\chi \in \Lambda^k(\mathcal{R}^n)$ can be represented by the basis as:

$$\chi = \sum_{i_1 < i_2 < \dots < i_k} a_{i_1, \dots, i_k} dx_{i_1} \wedge \dots \wedge dx_{i_k} = \sum_I a_I dx^I, \quad (20)$$

where $i_1, \dots, i_k \in [n]$ and $I = (i_1, \dots, i_k)$ determining scalars a_I and $dx^I = dx_{i_1} \wedge \dots \wedge dx_{i_k}$.

The space of differential k -form $\mathcal{A}^k(\mathcal{R}^n)$ (k -forms in the main content) is defined by the smooth function $w_I : \mathcal{R}^n \rightarrow \mathcal{R}$ as $w \in \mathcal{A}^k(\mathcal{R}^n) : \mathcal{R}^n \rightarrow \Lambda^k(\mathcal{R}^n)$:

$$w = \sum_I w_I dx^I. \quad (21)$$

Then the differential operator can be viewed as $d : \mathcal{A}^0(\mathcal{R}^n) \rightarrow \mathcal{A}^1(\mathcal{R}^n)$ as:

$$df(x) = \sum_{i=1}^n \frac{\partial f}{\partial x_i}(x) dx_i. \quad (22)$$

The exterior derivative $d : \mathcal{A}^k(\mathcal{R}^n) \rightarrow \mathcal{A}^{k+1}(\mathcal{R}^n)$ is defined as a linear operator and can be calculated by:

$$dw(x) = \sum_I dw_I \wedge dx^I, \quad (23)$$

where the calculating rules for the exterior product \wedge for two forms $w = \sum_I w_I dx^I$ and $\iota = \sum_J \iota_J dx^J$ can be derived by (with Eq. 19):

$$w \wedge \iota = \sum_{I,J} w_I \iota_J dx^I \wedge dx^J. \quad (24)$$

For the exterior derivative, an important property is $dd = 0$. It can be checked with the definition Eq. 23 and $\frac{\partial f^2}{\partial x_i \partial x_j} = \frac{\partial f^2}{\partial x_j \partial x_i}$. It can be also extended to the general manifold \mathcal{M} with the local chart and the similar proof.

The hodge operator $*$: $\mathcal{A}^k(\mathcal{R}^n) \rightarrow \mathcal{A}^{n-k}(\mathcal{R}^n)$ maps each k -form to $(n - k)$ -form by functioning as:

$$*w_I(dx^I) = (-1)^{sgn(\Upsilon)} w_I(dx^J), \quad (25)$$

where Υ denotes the permutation $(I, J) = (i_1, \dots, i_k, j_1, \dots, j_{n-k})$ on $[n]$. We can also derive the

$$** = (-1)^{k(n-k)} \quad (26)$$

in \mathcal{R}^n (the similar sign function result also holds in the general \mathcal{M}). We can also define the codifferential operator $\delta : \mathcal{A}^k(\mathcal{R}^n) \rightarrow \mathcal{A}^{n-k}(\mathcal{R}^n)$ as:

$$\delta = (-1)^{n(k+1)+1} * d *. \quad (27)$$

Then for a vector field $v = (v_1, \dots, v_n)$, we can represent it as 1-form $\mathbf{v} = \sum_{i=1}^n v_i dx_i$. Then the divergence can be calculated as follows:

$$*d*\mathbf{v} = *d\sum_{i=1}^n v_i *dx_i = *\sum_{i=1}^n \frac{\partial v_i}{\partial x_i} dx_1 \wedge \dots \wedge dx_n = \sum_{i=1}^n \frac{\partial v_i}{\partial x_i}, \quad (28)$$

where the equation holds with the fact $dx_i \wedge dx_i = 0$.

Following the discussion above, we can also simply prove that our Eq. 3 is divergence free:

$$*d*\mathbf{v} = *d**d\mu = (-1)^r *dd\mu = 0, \quad (29)$$

where $r = (n - 1)$ for $(n - 1)$ -form $d\mu$ in \mathcal{R}^n .

There are various approaches to parameterize μ . One alternative involves directly parameterizing μ using an antisymmetric matrix-valued function (Richter-Powell et al., 2022) to characterize the anti-symmetric property in hodge star and wedge product computation. Also, the codifferential operator $\mu = \delta\nu$ can also be employed to keep the properties required for $\nu \in \mathcal{A}^{n-1}(\mathcal{R}^n)$, and

$$\mathbf{v} = *d\delta\nu. \quad (30)$$

This form can be written as a vector representation with ν and simplify the parameterization.

We also propose a specific example to explore the relationship between the divergence-free field construction through the differential operator and the classic approach involving the curl operator (vector potential in fluid simulation) to enforce divergence-free.

We take \mathcal{R}^3 , $\mu = \nu_x dx + \nu_y dy + \nu_z dz$ for $\nu = (\nu_x, \nu_y, \nu_z)$ and the differentials $dX = (dx, dy, dz)$. Then we can derive

$$\begin{aligned} \mathbf{v} &= \left(\frac{\partial \nu_x}{\partial y} - \frac{\partial \nu_y}{\partial x} \right) dz + \left(\frac{\partial \nu_y}{\partial z} - \frac{\partial \nu_z}{\partial y} \right) dx + \left(\frac{\partial \nu_x}{\partial z} - \frac{\partial \nu_z}{\partial x} \right) dy = *d\mu = (\nabla \times \nu) \cdot dX \\ &= (\text{curl } \nu) \cdot dX. \end{aligned} \quad (31)$$

With Eq. 3, we can deduce that $\mathbf{v} = *d\mu$ is divergence-free. Additionally, it coincides with the curl operator of the vector ν , which happens to be a vector field in \mathcal{R}^3 . Consequently, the curl of a vector field possesses zero divergence ($\text{div} \circ \text{curl}$) in \mathcal{R}^3 . This conclusion is frequently utilized in simulations, known as the vector potential design (Elcott et al., 2007a; Chang et al., 2022), where the divergence-free field construction leverages curl operator. However, for the general manifold, it might be tricky to define each “curl” operator and the classical vector potential can not even be simply designed as a vector, while the differential operator computation for the divergence-free field construction can still work. We will next focus on the calculation of $*d$ operator for the divergence-free parameterization.

A.2 INTRODUCTION TO THE CLOSEST POINT METHOD

We also discuss some preliminaries for the Closest Point Method. We follow the introduction in King et al. (2023). For more details, please refer to King et al. (2023). Consider \mathcal{S} embedded in \mathcal{R}^3 . The Closest Point Method utilizes a closest point surface representation, which is a mapping from $x \in \mathcal{R}^3$ to the point $cp(x) \in \mathcal{S}$. The point cp is defined as the closest point on \mathcal{S} to x in Euclidean distance, i.e.

$$cp(x) = \operatorname{argmin}_{y \in \mathcal{S}} \|x - y\|. \quad (32)$$

For the smooth surfaces, cp is unique and well-defined in a sufficiently narrow tubular neighborhood $\mathcal{N}(\mathcal{S}) \subset \mathcal{R}^3$ surrounding \mathcal{S} (Marz & Macdonald, 2012) as an embedding in \mathcal{R}^3 , which can be formally described by:

$$\mathcal{N}(\mathcal{S}) = \{x \in \mathcal{R}^3 \mid \|x - cp(x)\| \leq r\}, \quad (33)$$

where r is called the tube-radius. The method is designed for solving PDE on the surface. The former definition enables us to formulate an embedding PDE on $\mathcal{N}(\mathcal{S})$, whose solution agrees with the solution of the surface PDE at the points $y \in \mathcal{S}$. More specifically, let $\tilde{u}(y)$ for $y \in \mathcal{S}$ and $u(x)$ for $x \in \mathcal{N}(\mathcal{S})$ denote the solution to the surface PDE and the embedding PDE, respectively. Fundamentally, the Closest Point Method (CPM) is based on extending surface from \mathcal{S} onto $\mathcal{N}(\mathcal{S})$ such that the data is constant in the normal direction of \mathcal{S} . The task can be accomplished by the closest point extension, i.e. we take $u(x) = \tilde{u}(cp(x))$ for all $x \in \mathcal{N}(\mathcal{S})$. We can observe that the CP extension assigns surface data at the closest point of x to x itself. Then we solve the embedding PDE $u(x)$ and constrain the point on the surface to derive the solution. This extension also allows the differential forms on the surface to be replaced with the differential forms on the Cartesian differential forms (Ruuth & Merriman, 2008).

Hence, we can utilize the transform of differential forms on the surface to the differential forms on the neighborhood, parameterizing with it to preserve the certain property (divergence-free). Then we use the parametric functionals to solve u of PDEs on the Cartesian neighborhood and pull it back to the surface with $u(x) = \tilde{u}(cp(x))$. Hence, our objective is to calculate the transformation of differential form that is required in PDE. The basic differential operator transformations such as exterior derivative, hodge star, wedge product $d, *, \wedge$ are showed in Li et al. (2023a) and Tab. 2. We next employ them to calculate the differential operator that we need to construct the corresponding divergence free field.

B PROOFS AND DERIVATIONS

B.1 PROOF FOR THEOREM 3.1

We first propose the exterior calculus rules on 3D in Tab. 2, where $f_{(k)}$ means the k -form of f . f and g represent the scalar function and $\mathbf{u}, \mathbf{v}, \mathbf{w}$ represent vector functions.

Output type	Wedge product \wedge	Interior product $i_{\mathbf{u}}$	Exterior derivative d	Hodge star $*$
0-form	$f_{(0)} \wedge g_{(0)} = (fg)_{(0)}$	$i_{\mathbf{u}} \mathbf{w}_{(1)} = (\mathbf{u} \cdot \mathbf{w})_{(0)}$	N/A	$*f_{(3)} = f_{(0)}$
1-form	$f_{(0)} \wedge \mathbf{u}_{(1)} = f\mathbf{u}_{(1)}$	$i_{\mathbf{u}} \mathbf{w}_{(2)} = (\mathbf{w} \times \mathbf{u})_{(1)}$	$df_{(0)} = (\nabla f)_{(1)}$	$*\mathbf{u}_{(2)} = \mathbf{u}_{(1)}$
2-form	$\mathbf{u}_{(1)} \wedge \mathbf{v}_{(1)} = (\mathbf{u} \times \mathbf{v})_{(2)}$	$i_{\mathbf{u}} f_{(3)} = f\mathbf{u}_{(2)}$	$d\mathbf{u}_{(1)} = (\nabla \times \mathbf{u})_{(2)}$	$*\mathbf{u}_{(1)} = \mathbf{u}_{(2)}$
3-form	$\mathbf{u}_{(1)} \wedge \mathbf{v}_{(2)} = (\mathbf{u} \cdot \mathbf{v})_{(3)}$	N/A	$d\mathbf{u}_{(2)} = (\nabla \cdot \mathbf{u})_{(3)}$	$*f_{(0)} = f_{(3)}$

Table 2: Exterior calculus operators in 3D.

Based on Li et al. (2023a), we can further write down the exterior calculus with the Closest Point Method via cp^* . It allows us to emulate the operators on \mathcal{S} (denoted with a superscript \mathcal{S}) using the operators on \mathcal{R}^3 (denoted with a superscript \mathcal{R}^3):

$$\text{CP-wedge product: } cp^*(\alpha \wedge^{\mathcal{S}} \beta) = (cp^* \alpha) \wedge^{\mathcal{R}^3} (cp^* \beta), \quad (34a)$$

$$\text{CP-interior product: } cp^*(i_{\mathbf{F}\mathbf{u}}^{\mathcal{S}} \alpha) = i_{\mathbf{u}}^{\mathcal{R}^3} cp^* \alpha, \quad (34b)$$

$$\text{CP-exterior derivative: } cp^*(d^{\mathcal{S}} \alpha) = d^{\mathcal{R}^3} cp^* \alpha, \quad (34c)$$

$$\text{CP-hodge star: } cp^*(\ast^{\mathcal{S}} \alpha)|_{j(\mathcal{S})} = i_{\mathbf{n}} \ast^{\mathcal{R}^3} (cp^* \alpha)|_{j(\mathcal{S})}, \quad (34d)$$

where $\mathbf{F} = d(j \circ cp)$ denotes the Jacobian and CP-hodge star is only applicable directly at the surface.

Proof for Theorem 3.1: We prove our theorem 3.1 by mathematical manipulations. As we state in Sec. 3.1, we need to construct $v = *^S d^S \sigma$ via σ on the surface. Then we apply cp^* on both side, we derive

$$\begin{aligned} cp^* v &= cp^* (*^S d^S \sigma) \\ &= i_n *^{\mathcal{R}^3} (cp^* d^S \sigma) \\ &= i_n *^{\mathcal{R}^3} d^{\mathcal{R}^3} cp^* \sigma \\ &= i_n *^{\mathcal{R}^3} \nabla (cp^* \sigma)_{(1)} \\ &= i_n \nabla (cp^* \sigma)_{(2)} \\ &= \nabla (cp^* \sigma) \times n. \end{aligned} \tag{35}$$

Then on the surface, we have $(cp^* v)(j(x)) = v(cp \circ j(x)) = v(x)$ (the closest point for the surface point is itself). Hence, we can derive

$$v = (\nabla (cp^* \sigma) \circ j(x)) \times n. \tag{36}$$

Then for the vorticity function in 2D, it is defined by $w = \text{curl } v$ as a scalar. For the surface, we derive the vorticity function via the hodge decomposition. We need to derive the divergence-free component of v (as “curl” on surfaces), which actually is $*dv$. Then we can derive

$$\begin{aligned} cp^* \omega &= cp^* (*^S d^S v) \\ &= i_n *^{\mathcal{R}^3} (cp^* d^S v) \\ &= i_n *^{\mathcal{R}^3} d^{\mathcal{R}^3} (cp^* v)_{(1)} \\ &= i_n *^{\mathcal{R}^3} (\nabla \times (cp^* v))_{(2)} \\ &= i_n *^{\mathcal{R}^3} (\nabla \times (cp^* v))_{(1)} \\ &= \nabla (cp^* v) \cdot n. \end{aligned} \tag{37}$$

Similar with Eq. 35, we achieve

$$\omega = (\nabla \times v) \cdot n. \tag{38}$$

□

Note that on the surface, the rules work like ones in 2D: the stream and the vorticity functions are both scalars (compared with the vector potential functions in \mathcal{R}^3 (Elcott et al., 2007a; Chang et al., 2022)). It can also be verified with the exterior calculus. If σ is not 0-form but 1-form instead, we can not derive v as 1-form but 0-form, which brings trouble for our neural representation.

Remark: We can next parameterize the σ (with NN) and compute v and ω by sampling points and do calculus with Eqs. 36 and 38. The theorem provides a continuous and analytic formulation on the vorticity function on the surfaces with the help of the Closest Point Method in Tab. 2 while the classic method (Azencot et al., 2014) needs to compute the discretized differential form with the triangle meshes. CPM enables us to parameterize the surface field on the ambient space and facilitate the neural representation.

C CONDITIONING PROPERTY OF OUR FRAMEWORK

For the generation problems, to verify the capability, we mirror the architecture of variation auto-encoder. Let’s consider images as an example input, and our objective is to provide a model that receive the coordinate x and image conditions as an input and generate a divergence-free field of which vorticity ω resembles the corresponding conditioned images silhouette in visualization. Images q are encoded by the parametric encoder with θ_q and the features are provided with the reparameterization trick $z_q(\theta_q, q) \sim \mathcal{N}(\xi(\theta_q, q), \tau(\theta_q, q))$ (Kingma & Welling, 2013), where $\xi(\theta_q, q), \tau(\theta_q, q)$ is the mean and variance output by the encoder. The encoded feature $z = (z_q, z_{aux})$, where z_{aux} is auxiliary information such as the image class, is input together with positional embedding (like

siren) from x . Then the decoder translates the feature to σ function and further derive ω with Theorem 3.1. The loss function consists of two parts: One is $\|\omega(\sigma(\theta, z(\theta_q))) - G(q)\|_2$, where G is mapping from the image color space to vorticity space. This term supervises the vorticity field to shape as the silhouette of the corresponding images. The other term is Kullback-Leibler divergence to constrain the distribution of z_q to be normal, as given by $\|\xi^2 + \tau^2 - \log \tau^2 + 1\|_2$ with $\xi, \tau \in \mathcal{R}^r$, where r is the dimension of z_q and log operation is computed element-wise. With such design, we could derive a simple conditioning framework, which enables more data prior into the simulation via a more end-to-end manner. More details are also provided in Appendix G. Our neural parameterization provides more direct and effective approach to involve the semantic information and show more potential to combine with the advanced generation methods in 3D or language model.

Remark: The idea of the conditioning resonates with the eigen-decomposition of the fluid fields, as discussed in Cui et al. (2021). This method decomposes the vector field v into several divergence-free basis u_i , i.e. $v = \sum_{i=1}^r w_i u_i$ and u_i is the eigenfunction of the Laplacian operator. It's noteworthy that w_i can serve as the encoding z in our formulation. If we strengthen our neural represented velocity field to become eigenfunctions of the Laplacian operator on surfaces, we can derive similar decomposition weights through the conditioning process described above.

D PSEUDOCODE FOR ADVECTION

Algorithm 1 Advection for Neural Flow on Surfaces.

Input: Initial velocity field v_0 , vorticity field ω_0 , timestep size h , number of timesteps N , surface \mathcal{S} , training steps E , sample size k , learning rate α
Fitting the initial network weight θ_0 and non-zero harmonic term η with v_0 and ω_0 .
for $n = 1$ **to** N **do**
 $\theta_{n+1} \leftarrow \theta_n$
 for $i = 1$ **to** E **do**
 Sample k point on \mathcal{S} as sample set \mathcal{M} .
 Compute the stream function $\sigma(\theta_{n+1})$ on \mathcal{M} .
 Compute the velocity $v(\theta_n)$, $v(\theta_{n+1})$ and vorticity $\omega(\theta_n)$, $\omega(\theta_{n+1})$ with Eqs. 4 and 5 for time n and $n + 1$ with $\sigma(\theta_n)$, $\sigma(\theta_{n+1})$ and η .
 Construct loss function $\mathcal{L}_{\theta_{n+1}}$ with Eq. 15 for \mathcal{M} .
 $\theta_{n+1} \leftarrow \theta_{n+1} - \alpha \nabla \mathcal{L}_{\theta_{n+1}}$
 end for
end for

E ADDITIONAL EXPERIMENTAL RESULTS

We include additional experiments to verify the performances of our method as a supplementary. We provide the quantitative study for the flow on the inclined plane, the ablation and comparison on the sphere jet flow and the convergence study for our flow on the implicit neural representation.

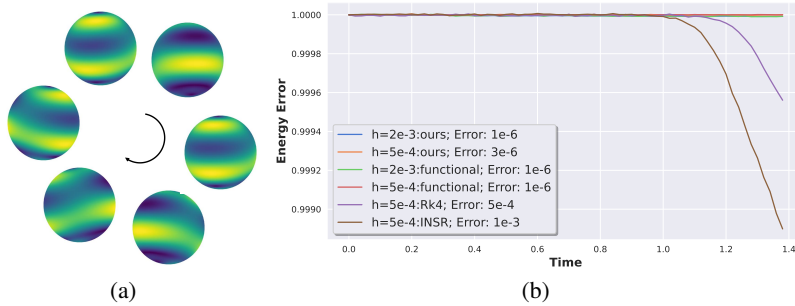


Figure 7: Results for rotating sphere flow. (a) Qualitative results for rotating sphere. (b) Quantitative results for energy preservation.

E.1 RESULTS FOR ROTATING SPHERE FLOW

We verify the energy conservation property of our method by examining the rotating sphere flow, using an analytic solution as validated by Azencot et al. (2014). This approach allows us to rigorously test our simulator’s reliability in preserving energy over time. The initial flow conditions combine a Killing vector field with a rotated gradient of an eigenfunction of the Laplace-Beltrami operator on the surface. Killing vector field is a vector field whose Lie derivative of the metric vanishes, meaning that the flow generated by the vector field remains constant in advection (Jost & Jost, 2008). A classic example on the sphere is the vector field $(-y, x, 0)$ around z -axis. The eigenfunctions of the Laplace-Beltrami operator on the sphere are well-known as Spherical Harmonics functions. We also apply the rotation on them for better identification. Actually, this rotating sphere flow appears as if the sphere with the vector field is rotating over time when observed from a fixed point. It can be demonstrated that the energy of inviscid flow with these initial conditions remains constant, making this configuration an excellent test case for energy conservation. As the results showed in Fig. 7 (a), the results exhibit periodic patterns resembling the sphere rotating.

We also plot the energy change across the entire sphere, comparing it with Functional Fluid on Surfaces (Azencot et al., 2014), the classic method with Runge-Kutta (RK) time integrator and Implicit Neural Spatial Representations (INSR) (Chen et al., 2023) with semi-Lagrangian advection as the loss function. The results indicate that our method achieves a relative change in energy on the order of 10^{-5} , which is comparable to the results from the Functional Fluid on Surfaces method for both larger and smaller time steps. Conversely, the RK method and INSR suffers from larger energy losses, even for smaller time steps.

E.2 MORE RESULTS FOR THE TAYLOR VORTICES ON THE INCLINED PLANE

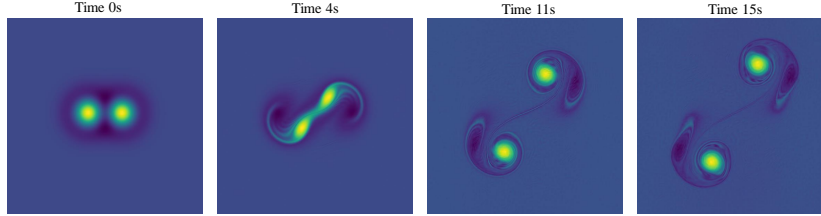


Figure 8: Dynamic Results for Taylor vortices on inclined plane.

We include the qualitative results of our simulation dynamics for Taylor vortices in Fig. 8 and more comparison results are provided in the supplementary video. We can observe a clear vortex pair and the phenomenon of separation.

Methods	Error	Time	Storage
PINN	4.16e5	9.6 h	568.1KB
INSR	3.45e3	16.8 h	516.3KB
Small-F.S.	3.21e2	0.2h	535.6KB
HomoLBM	8.92e1	3.8 h	4.3MB
Small-F.S.	3.21e2	0.2h	535.6KB
Ours	1.71e1	12.7 h	521.3KB
GT	N/A	2.3 h	3964.0KB

Table 3: Quantitative results for the Taylor vortices on inclined plane. Error: mean square error (MSE) averaged by 120 time steps with resolution of 400. The storage of our high resolution ground truth is 7.6 times than ours.

We also include the quantitative results for the Taylor vortices in Tab. 3, together with more comparison with classic methods on the 2D plane for benchmark studies, including Stable Fluid (with a resolution of 1024x1024) (Stam, 1999) and high-order Lattice Boltzmann methods (not parallel version with a resolution of 512x512) (Li et al., 2023b). The corresponding qualitative results are

presented in Fig. 9. The results demonstrate that our method enables high accuracy while showing high memory-efficiency, compared with both classical and recent advanced methods under the similar memory cost.

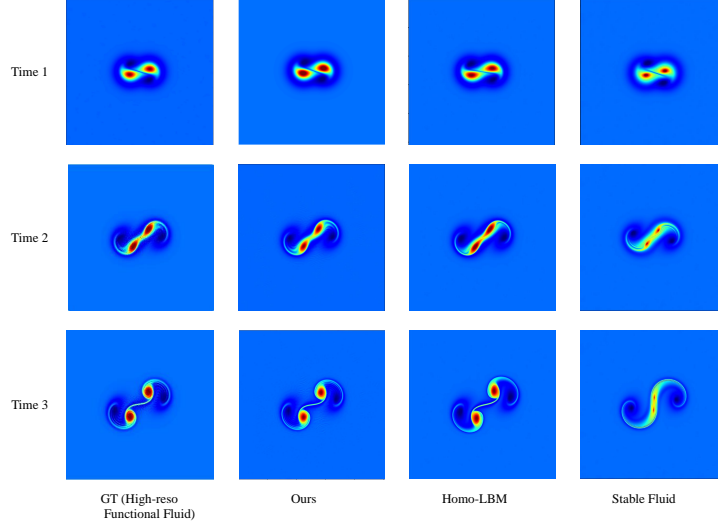


Figure 9: Qualitative results for the Taylor vortices compared with 2D classical methods.

E.3 MORE COMPARISON RESULTS ON THE SPHERE JET FLOW

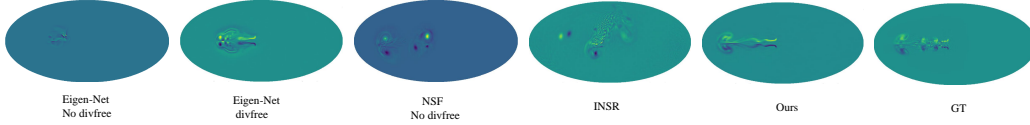


Figure 10: Qualitative results for the sphere jet flow compared with eigen-net and NSF.

We compare with other surface field representation methods to verify the effectiveness of our proposed framework. We try to adapt two methods into our framework. One is Koestler et al. (Koestler et al., 2022), which proposes a surface field representation via the eigenfunction of the Laplacian-beltrami operator (eigen-net) on the surface and the other is Xue et al. (Xue et al., 2023) that utilizes the MLP and projection operator (NSF). We implement them on the sphere jet flow case for comparison since the Laplacian-beltrami operator can be analytically computed by Spherical Harmonics. To further verify the effectiveness of our field function design, for eigen-net, we implemented two variations as ablations: one that incorporates our divergence-free design along with the covariant derivative advection, and another that avoids our divergence-free approach in favor of the traditional advection and divergence projection; for NSF we adopted the method without our divergence-free design, as incorporating it would result in a configuration similar to our own framework, with the primary distinction being the hyper-parameters of the MLP or siren. We show the quantitative results in the Tab. 4 and the corresponding qualitative results in Fig. 10.

Both quantitative and qualitative results demonstrate that our proposed framework achieves higher accuracy and memory-efficiency in the incompressible Euler flow simulation, compared with pure eigen-net or NSF surface field representations. The ablation results also indicate that that our proposed divergence-free field design framework significantly improves the accuracy of the eigen-net. Ours avoid the divergence-free projection, which reduces the extra fitting error and cascading error effects. The results also illustrate that our proposed divergence-free design is robust with the ways of parameterization (MLP, siren or eigen-net).

Methods	Error	Time	Storage
Eigen-net without divergence-free design	6.82e6	13.5 h	632.5KB
NSF without divergence-free design	9.76e4	12.8 h	501.3 KB
Eigen-net with divergence-free	1.13e3	16.7h	622.4KB
Ours	2.89e2	16.5 h	532.8KB

Table 4: Quantitative results for the sphere jet flow compared with eigen-net and NSF. Error: mean square error (MSE) averaged by 100 time steps on 81924 mesh vertices.

Methods	Error	Time	Storage
Elcott et al. 2007	1.27e4	6.8h	2643.0KB
Stable Fluid	8.62e5	0.2h	972.8KB
Small-F.S.	5.34e3	0.8h	583.8KB
Ours	2.89e2	16.5 h	532.8KB
GT	N/A	8.3 h	2643.0 KB

Table 5: More Quantitative results for the sphere jet flow with classic methods.

We also include more results of classical solvers for reference, including Stable Fluid (with a resolution of 256x256) (Stam, 1999) and Elcott et al. (Elcott et al., 2007a) (using the same mesh as GT). The quantitative results are shown in Tab. 5, and the qualitative results are presented in Fig. 11.

The results show that our method achieves high performance and low energy dissipation compared to classic methods. However, this improvement comes with the trade-off of higher time consumption as discuss in Appendix F.3.

E.4 CONVERGENCE VERIFICATION FOR THE FLOW ON THE IMPLICIT REPRESENTED SURFACES

Marching Cubes resolution	Average steps of Crashing	Storage
64	36.5	117.18 KB
128	53.4	768.32 KB
240	68.7	1123.15 KB
GT Mesh	88.6	1419.48 KB
Ours	N/A	523.4 KB

Table 6: Time steps that the classic method (Azencot et al., 2014) with Marching Cubes on implicit neural represented surfaces crashes. We repeat the process with 10 times and derive the averaged results. However, ours can work well with the implicit neural representation.

For the case of implicit represented surfaces, as previously stated in the main context (Sec. 5.2), for the classical method it is not convergent, leading to non-referable results. To substantiate our claim regarding non-convergence, we include a new Tab. 6 that lists the simulation crashing time steps across different marching cube resolutions for the traditional methods. This data will offer a quantitative perspective on the limitations of the classic methods in robustness. We also include the reference qualitative results in the supplementary video to exhibit the simulation process and describe the non convergence on different resolutions. The results demonstrate that our meshless and end-to-end method can achieve high adaptability and robustness for the simulation on the implicit neural representation, while the classical method fails in marching cube meshes and needs further complex geometry processing schemes to improve the mesh quality.

E.5 ABLATION STUDIES FOR THE NETWORK AND TRAINING DESIGN

We also include the ablation studies on the network size and sample count on our sphere jet case. The results are exhibited in Tab. 7 and Tab. 8, which indicate that our method is relatively insensitive to these settings unless the width is extremely small.

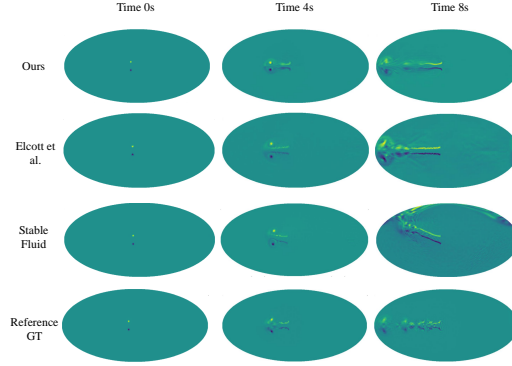


Figure 11: Qualitative results for the sphere jet flow compared with more classical methods.

Network Size	Error
4 layers, 128	2.89e2
4 layers, 256	2.21e2
4 layers, 64	4.57e2

Table 7: Ablation studies for network size for the sphere jet flow.

F DISCUSSIONS AND LIMITATIONS

F.1 DISCUSSIONS AND LIMITATIONS ON THE TOPOLOGY PROBLEMS

The first limitation is about the singularity problem. We don’t focus heavily on the singularity issues related to topology, such as those highlighted by the Poincaré-Hopf Theorem, which is a complex challenge. Our main interest lies in using neural approaches to simulate fluid dynamics and model vortex dynamics on surfaces for visual effects (as described in Eq. 6), rather than dealing with arbitrary vector fields. The scenes chosen in both Azencot et al. (2014); Ando et al. (2015) (which do not mention the issue) and ours typically maintain non-zero measure zero velocity and avoid poles to simplify the analysis. We initialize the simulation with finite vorticity, which makes it hard for advection to generate infinite vorticity (or singularities) within our settings. Furthermore, we employ the stream and curl regularization to avoid extreme values in the singularity points and preserve system stability. According to Sard’s theorem, the number of poles has zero measure in our simple cases, we maintain a very small probability of sampling at the poles, which improves the stability of the simulation. Though the velocity field near singularities will be underfitted and approximated by the network, potentially leading to inaccuracies and numerical dissipation near the “cyclone”, the overall results provide empirically reasonable visual effects, which is the primary goal of our application.

To further validate the effectiveness of our regularization and sampling, we applied our framework to fit the velocity $(-\sin(\theta), \cos(\theta))$ on a sphere with spherical coordinates $(1, \theta, \phi)$, which keeps a singularity at the northern/southern pole $((1, 0, 0), (1, \pi, 0))$. The errors and velocity magnitude are shown in Fig. 12, where we achieve the desired velocity pattern. Although some errors appear in the initial epochs, they become negligible over time, which is sufficient for graphics and visual effects.

Training Samples (millions)	Error
60	2.89e2
40	3.12e2
80	2.79e2

Table 8: Ablation studies for sample number for the sphere jet flow.

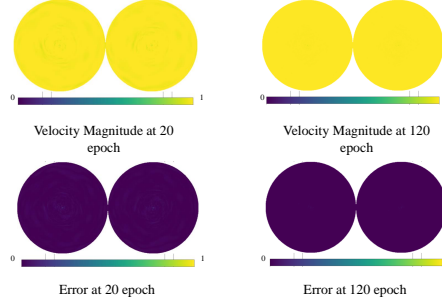


Figure 12: Qualitative results of the numerical studies (Velocity magnitude and error) for the constant sphere flow. (Views from the northern and southern poles)

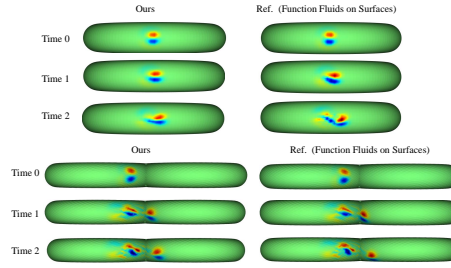


Figure 13: Qualitative results for the sphere jet flow on the explicit torus and double torus.

Another problem related with the topology is the cohomology term. Our method is mainly adopted for the topology for genus zero, where the velocity can be estimated by the stream function without more consideration of the time-variant cohomology component (as stated in Yin et al. (2023) Proposition 5 for the surfaces removing several poles). For the surface with high genus, we find it insufficient for the time-invariant cohomology term to handle highly turbulent flow and generate complex visual effects. The problem can be further addressed by incorporating the solver in Yin et al. (2023) with the neural network, which will be our future work. However, only for the visual effects, our method still performs well for the high genus surfaces and shows the correct jet flow behavior compared to the reference classic method. We plot the results in Fig. 13 with simple torus and double torus cases with explicit torus and double torus meshes. They do demonstrate reasonable visual effects on high-genus surfaces even without considering the time-variant cohomology.

F.2 DISCUSSIONS AND LIMITATIONS ON THE GEOMETRY PROBLEMS

As stated in Sec. 3.2, our method needs SDF for the flow on the implicit surfaces. This can pose challenges when calculating normals for open surfaces. To overcome this, one can use an unsigned distance field (UDF) to extend our framework to open surfaces, which is well applied in Chibane et al. (2020); Yang et al. (2023); Long et al. (2023).

Moreover, in the practical scenarios, the non-smooth surface, imperfect SDF (common problems in implicit neural representations such as Yifan et al. (2021)) and the complex surfaces (like unoriented surfaces) can also limit our performances. Noisy surfaces and normals introduce numerical viscosity, slowing down the simulation. But fortunately, we show that they do not drive our simulation to crash, as the neural representation provides smoother and more robust results (as demonstrated in Appendix E.4 with GT mesh data) compared to traditional methods. To further solve these issues, we can construct smooth approximator and utilize a large number of samples from a smaller tube radius (as described in King et al. (2023)) for improved performance. Additionally, the orientation problem is addressed in King et al. (2023) and the extension for our method can be similarly constructed.

F.3 DISCUSSIONS AND LIMITATIONS ON THE TIME AND MEMORY CONSUMPTION

Our method actually keeps large advantages at the memory consumption albeit with the increased time requirements. The low memory usage is crucial for handling and analyzing high-resolution data. For example, a 2D simulation at a 4096x4096 resolution requires about 200MB per frame, resulting in considerable memory demands for extended simulations. Similarly, high-resolution 3D grid simulations consume even more memory. Although mesh-based simulations can be more efficient, they encounter mesh quality issues, as shown in the robustness tests in Appendix E.4. The time cost remains high, as methods like siren or simple MLPs are too global and inefficient for optimization, and inference sampling (particularly for implicit neural representations) is time-consuming. To address the time consumption issue, a hybrid simulator to achieve both high speed and performances is necessary. Fortunately, recent advances in hybrid representations (Müller et al., 2022; Huang et al., 2023) have shown promising results in reducing training time from hours to seconds while maintaining the expressiveness. Employing these more efficient representations hold great promise for improvements of our method. In the inference time, a sampler (Sharp & Jacobson, 2022) with high efficiency can be adopted, which will provide huge saving for the inference time with KD hierarchies.

G MORE IMPLEMENTATION DETAILS

We provide our implementation details for our numerical studies. Our experiments are all implemented with Jax library (Bradbury et al., 2018) on an NVIDIA GeForce RTX 3090 GPU.

Sphere Jet: We adopt the 4-layers MLP (for shaper simulation results compared with siren) with 128 units for our implementation. The learning rate is set with the exponential decay from $1e - 5$ to $1e - 7$ with 60000 steps and batch size 1000 for each time step. The time step is chosen as $5e - 2$. The initialization is the same as Azencot et al. (2014) and the initial vorticity is kept for the whole simulation time. For the comparison, INSR uses the siren function with 4-layers 128 units for advection, projection and correction respectively. The learning rate is set as $1e - 6$. For each process, the siren iterates for 40000 steps with batch size 5000. For the original PINN, we adopt the MLP with 4-layers 128 units. The loss function is to enforce the incompressible Euler equation directly, as showed in Appendix A.3 of Chen et al. (2023). The learning rate is set by $1e - 5$ with 60000 steps and batch size 2000.

Taylor vortices on inclined plane: We adopt 4-layer siren with 128 units representation to conduct the positional encoding with the first layer frequency as 30. The time step is chosen as $5e - 3$. The initialization is set the same as McKenzie (2007), and we rotate the plane and make the normal be $(0.3, -0.5, 0.8)$. The domain size is set $[-\pi, \pi]$ with the periodic boundary condition. The time step is set as 0.05. The learning rate is set with the piece-wise constant from $1e - 5$ with a decay factor 0.1 on 40000 and 60000 steps for total 80000 steps and batch size 1000. For comparison, INSR adopts the siren MLP with 4 layers and 128 units per layer for advection, projection and correction respectively. The learning rate is set as $1e - 5$. For each process, the siren iterates for 20000 steps with batch size 1000. For the original PINN, we adopt MLP with 4 layers and 128 units. The loss function consists of the governing equation part which is the same as the one in sphere jet case and the periodic boundary condition part. The learning rate is set by $1e - 5$ with 60000 iterations and batch size 1000.

Rotating sphere flow: For the construction of σ , we adopt 4-layer siren with 128 units representation (Sitzmann et al., 2020) with the first layer frequency as 30. The learning rate is set with the exponential decay from $1e - 6$ to $1e - 8$ with 40000 steps and batch size 1000 for each time step. The time step is chosen as $5e - 4$ and $2e - 3$. The initialization of the velocity is set as the Killing field $(-y, x, 0)$ with the rotated 4-degree Spherical Harmonics functions with order 4 and 5. For INSR for our comparison, we adopt the siren function with 4-layers 64 units for advection, projection and correction respectively. The learning rate is set as $5e - 4$. For each process, the siren iterates for 10000 steps with batch size 1000.

Flow on the explicit meshes. We adopt a 4-layer siren with 128 units representation for both models with the first layer frequency as 30. The learning rate is set with the exponential decay from $1e - 6$ to $1e - 8$ with 40000 steps and batch size 2400 for each time step. The time step is chosen as $5e - 2$

and $8e - 2$ for the hand and spot respectively. The hand model is initialized by two vortices with the geodesic 0.41 by Crane et al. (2013b) and the spot model is set with 0.3.

Flow on the implicit mesh. First, we adopt 4-layer siren with 256 units to reconstruct the implicit neural representation of the SDF function based on Sitzmann et al. (2020) with the uniform sampling. The in the simulation, we take the rejection rules in Yang et al. (2021) to complete the uniformly sampling for the simulation function and avoid the samples concentrating near the high curvature area. The jet vortices are initialized by given two points on the mesh as an opposite pair. Our simulation neural field is also implemented as 4-layer siren with 128 units representation with the first layer frequency as 30. The learning rate is set with the exponential decay from $1e - 5$ to $1e - 7$ with 40000 steps and batch size 1000 for each time step. The time step is set by $5e - 2$ and $2e - 2$ for Armadillo and Lucy respectively.

Flow with conditioning: We adopt the encoder consists of 2-layer feature extraction MLP to reduce image data to a 32-dimension feature space; siren network to generate 128-dimensional feature with the input position x as neural field representation and an one-hot class encoder for image categorization. Note that the first layer frequency for the siren network above is also 30. The decoder is a 2-layer MLP that transforms the concatenated features from the encoders to the stream function σ . Then following our Theorem 3.1, we can also derive the corresponding v and ω taking the derivative with respect to x . In the inference time 32-dimension random Gaussian vector and the corresponding image class are input to concatenate with the positional encoding to generate the field value at the spatial point. The learning rate for the training process is set with the exponential decay from $1e - 4$ to $1e - 5$ with 400000 steps and batch size 1000. We map the color of the input images to the vorticity function and try to make the velocity field preserving the vortices shape as alphabets via Mean Least Square loss and KL loss.

Flow for Helmholtz decomposition: We adopt the atmosphere data (100-metre wind velocity) on Jan, 2024 (Raoult et al., 2017). We adopt 4-layer siren with 128 units representation for both models with the first layer frequency as 100. The learning rate is set with the exponential decay from $1e - 4$ to $1e - 5$ with 200000 steps and batch size 1000. We try to make our velocity results closer to the given data and derive the divergence-free component, similar as Richter-Powell et al. (2022).