ECAI 2024
U. Endriss et al. (Eds.)

doi:10.3233/FAIA240513

# Solving PDEs on Point Clouds by Physics-Informed Learning with Graph Neural Networks

**Rakhoon Hwang**[a,1], **Junseung Ryu**[b,1], **Seungtae Park**[c] **and Hyung Ju Hwang**[b,d,e,*]

[a]Advanced Vehicle Platform Division, Hyundai Motor Company
[b]Department of Mathematics, Pohang University of Science and Technology
[c]Department of Mechanical Engineering, Pohang University of Science and Technology
[d]Graduate School of Artificial Intelligence, Pohang University of Science and Technology
[e]AMSquare Corp.
ORCID (Rakhoon Hwang): https://orcid.org/0000-0001-9220-5027, ORCID (Junseung Ryu): https://orcid.org/0009-0002-3185-3738, ORCID (Seungtae Park): https://orcid.org/0000-0002-8776-8375, ORCID (Hyung Ju Hwang): https://orcid.org/0000-0002-3678-2687

**Abstract.** Scientific machine learning (SciML) explores the development of neural network models to approximate solutions to Partial Differential Equations (PDEs). However, there exists a significant research gap when computational domains are arbitrary manifolds, which are common in real-world scientific and engineering applications. The inherent challenge arises when calculating differential operators defined on curved surfaces, particularly in scenarios where surface parameterization is unavailable. In this paper, we present a neural network-based method for solving PDEs on surfaces described only by point clouds, without any other geometrical priors. Our method comprises two steps—local surface approximation based on graph neural networks and solving PDEs on point clouds. For surface reconstruction, our graph neural networks can be generalized based on the predictions of simple geometries during training to significantly more complicated surfaces for evaluation. The proposed approach demonstrates its capacity to learn geometric features from point cloud data without requiring external datasets, offers superior performance compared to benchmark models across various PDE types, and exhibits robustness in handling complex surfaces, non-uniform point distributions, and noise.

## 1 Introduction

The field of scientific machine learning (SciML) encompasses the development of various techniques, including physics-informed neural networks (PINN). These models softly regularize neural networks to follow target Partial Differential Equations (PDEs), resulting in accurate approximate solutions. In several bechmark experiments, PINNs show promising results [21, 22, 32]. These experiments show that PINN frameworks have several advantages over traditional numerical solvers such as finite element methods and pseudo spectral methods: 1) it converges faster than the numerical solvers, 2) it does not require meticulous mesh design and 3) it shows marginal deviation from analytical solutions.

* Corresponding Author. Email: hjhwang@postech.ac.kr.
[1] Equal contribution.

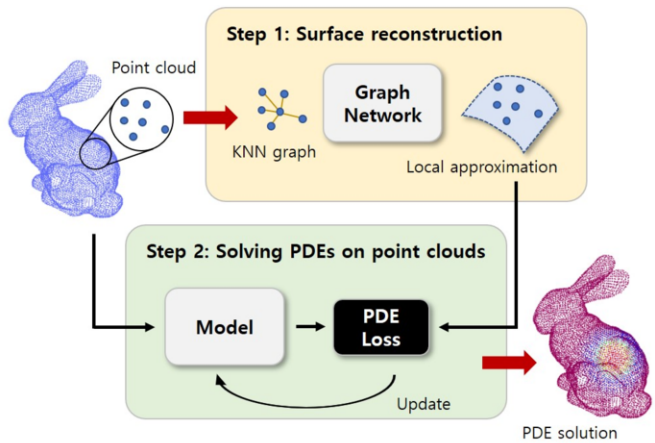


**Figure 1.** Overview of the proposed method: Graph Network (GN) approximates the underlying surface locally for the given point cloud data. Here, the GN is trained on simple geometries, such as the torus. Based on the geometric information obtained using the GN, the model is optimized to minimize the residual loss associated with the PDE of interest.

However, training PINNs on an arbitrary manifold with curvature is relatively less studied while most of real-world scientific and engieering problems need solving PDEs on an curved surface [7, 10, 12]. The challenge arises when calculating differential operators defined on curved surfaces. For instance, when modeling phenomenon over surfaces, we need to evaluate the rate of change of physical quantities which vary along the surfaces. The surface gradient and the Laplace-Beltrami operators arise as generalizations of the conventional gradients, and provide effective tools when describing many problems on surfaces. The calculation of these differential operators involve approximations of surface normal vectors. Thus, it is not straightforward to apply conventional PINN frameworks, especially for PDEs on complex surfaces represented as meshes or point clouds, which lack analytical expressions. Thus, understanding the effective estimation of surface normals and their integration into PINNs is a subject worth exploring.

Some learning-based methodologies have been introduced recently to solve PDEs on curved surfaces. Most of them focus on the approximation of differential operators on surfaces, e.g., the Laplace-Beltrami operator. When the surface parameterization is known, PINNs can be employed to solve PDEs by introducing extra loss terms to enforce normal constraints [12, 18]. Surface derivatives can be directly computed using global coordinates and incorporated into the PINN loss function. However, it is important to note that these approaches are most effective when surface parameterization is explicitly provided, limiting their utility for complex, challenging-to-parameterize surfaces. In our study, we concentrate on solving PDEs when surface information is available in the form of a point cloud, which is the simplest and most general representation of geometric objects.

In this paper, we propose a two-step method for solving PDEs on point clouds. The first step involves local surface approximation using graph neural networks (GNNs) for the given point cloud data, while the second step focuses on minimizing residual losses to approximate the solutions of the PDEs. The key contributions of our study are as follows:

- We introduce a comprehensive framework for solving PDEs on point clouds, which is the simplest and most general representation of geometric objects. This framework learns geometric features from synthetic data and then approximates PDE solutions based on given point clouds. Importantly, our approach needs no external data sources except point cloud data for geometric objects, as both the geometric feature estimator and PINNs do not rely on precomputed datasets.
- We provide quantitative and qualitative results across various types of PDEs to demonstrate the superior performance of our proposed method compared to benchmark models.
- We experimentally validate the robustness of our method regarding surface complexity, non-uniform point distribution, and noise, affirming its applicability to real-world problems.

## 2 Related Works

**Surface reconstruction** The initial step of the proposed framework is surface reconstruction based on 3D point clouds, which is widely used in various fields, e.g., computer graphics and medical imaging. A classical method for surface reconstruction involves the use of Moving Least Squares (MLS) [1, 13]. For example, the surface can be locally approximated using multivariate polynomials. Such an algorithm works well when points in the domain are distributed uniformly, but not in non-uniform cases. This is main difficulty in applying it to real-world problems since point clouds in real world, usually obtained by 3D scanning, are often nonuniform [24, 25].

On the other hand, some deep learning-based approaches have been used for surface reconstruction recently [16, 40, 15, 2, 5, 42]. For example, [40] approximated a local chart by a deep neural network for surface reconstruction. Although their model exhibits good performance, the calculation of differential operators with respect to the surface, e.g., the surface gradient and Laplace-Beltrami operator, is impractical.

In contrast, the method proposed in this paper approximates the surface using a simple polynomial, as in [26]. The main difference between the two approaches is that our method uses Graph Networks (GNs) [4] to construct a local coordinate system. GN, a type of GNNs [35], is widely used in various fields involving interactions between physical objects [3, 33, 41, 38]. Recently, [34] developed a novel framework called Graph Network-based Simulators, which learn to simulate diverse physical domains using GNs. Inspired by the previous work, we propose a GN-based framework to estimate normal vectors on point clouds. We introduce GNs briefly and apply them to construct the local coordinate system in Section 3.1.

**Solving PDEs on surfaces** Many numerical methods have been proposed for solving PDEs on point clouds over the years. [29, 23] are based on the radial basis function finite difference (RBF-FD) scheme. Such methods are highly sensitive to choice of RBFs, easily to be unstable for the case of noisy and complex domains. [26] utilizes the principal component analysis (PCA) with Finite Difference Methods (FDM). This method performs well when points are distributed uniformly and dense enough on simple domains, but it is not robust on noise and complexivity of domains.

Artificial neural networks have been used as function approximators to solve PDEs in several studies [30, 39, 17, 37, 27, 36]. Typically, the model parameters are optimized to minimize the residuals of the governing equations or physical constraints by computing the spatial and temporal derivatives via auto-differentiation. Some studies [20, 28] propose a domain-decomposition-based frameworks which enables solving PDEs on complex geometrical domains in Euclidean spaces. Although these approaches are successful in many benchmarking experiments, most are not applicable to curved surface domains. In particular, differential operators defined on curved surfaces cannot be in the same way as the previous approaches.

Recently, solving PDEs on surfaces using neural networks has been considered. The strategy depends on the representation of the surface. When a precise parameterization of the surface is known, the PINN framework can be employed using the full geometric information of the surface. For example, in [12], the authors introduced an additional loss term to ensure that the solution exhibits zero normal derivatives on the surface. Under this restriction, the Laplace-Beltrami operator was replaced with the conventional Laplace operator, as in classical numerical schemes [31, 29]. On the other hand, in [18], the Laplace-Beltrami operator was expressed in terms of the Cartesian differential operator, which was used directly in the loss function. When triangulation or mesh information of the surface is given, PDEs can be discretized on each triangular mesh and used to train models [32, 8, 14]. However, in general, obtaining good parameterization or triangulation of surfaces, particularly of complicated ones, is difficult. In this study, we focus on solving PDEs on surfaces described by point clouds, which is the most practical but least informative way to represent surfaces.

## 3 Methods

We consider PDEs on surfaces in the general form:

$$\begin{aligned} \mathcal{L}u(\mathbf{x}) &= f(\mathbf{x}) \text{ on } \mathcal{M}, \\ \mathcal{B}u(\mathbf{x}) &= g(\mathbf{x}) \text{ on } \partial\mathcal{M} \end{aligned} \tag{1}$$

where $u$ denotes the quantity of interest, $\mathbf{x} = (x, y, z)$ denotes the spatial coordinate on the surface $\mathcal{M} \subset \mathbb{R}^3$ with boundary $\partial\mathcal{M}$, $\mathcal{L}$ denotes a general differential operator defined on $\mathcal{M}$, $\mathcal{B}$ denotes an operator on $\partial\mathcal{M}$ that defines the boundary conditions, with prescribed functions $f$ and $g$. We consider the following form for time-dependent PDEs:

$$\begin{aligned} \frac{\partial u(t,\mathbf{x})}{\partial t} + \mathcal{L}u(t,\mathbf{x}) &= f(t,\mathbf{x}) \text{ in } (0,T)\times\mathcal{M}, \\ \mathcal{B}u(t,\mathbf{x}) &= g(t,\mathbf{x}) \text{ on } (0,T)\times\partial\mathcal{M} \\ u(0,\mathbf{x}) &= h(\mathbf{x}) \text{ on } \mathcal{M} \end{aligned} \tag{2}$$

for some $T > 0$ and initial condition, $h$. In this study, we assume that $\mathcal{M}$ is represented by a point cloud $X = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N\}$. For convenience, it is assumed to be a regular orientable surface in $\mathbb{R}^3$, which means that the surface has differentiable normal vectors. It is worthy mentioning that, as $u$ is a function defined on $\mathcal{M}$, the derivatives on surfaces need to be treated in a different way than in simple planar domains. For instance, the surface gradient $\nabla_{\mathcal{M}}$, and the Laplace-Beltrami operator $\Delta_{\mathcal{M}}$, included in Equations (1) and (2). These differential operators are quite different from the standard gradient and Laplace operator in that they can measure the rate of change of quantities along a curved surface.

More precisely, let $\mathcal{M}$ be parameterized by $\Gamma = \Gamma(x_1, x_2)$. We then obtain the metric tensors $g_{ij}$'s defined by $g_{ij} := \langle \Gamma_{x_1}, \Gamma_{x_2} \rangle$. Let $G := [g_{ij}]$, $g := \det(G)$, and $g^{ij}$ denote the elements of the inverse matrix $G^{-1}$. The surface gradient of u is defined by

$$\nabla_{\mathcal{M}} u = \left( g^{11} \frac{\partial u}{\partial x_1} + g^{12} \frac{\partial u}{\partial x_2} \right) \Gamma_{x_1} + \left( g^{21} \frac{\partial u}{\partial x_1} + g^{22} \frac{\partial u}{\partial x_2} \right) \Gamma_{x_2}, \tag{3}$$

and the Laplace-Beltrami operator of $u$ is

$$\begin{aligned} \Delta_{\mathcal{M}} u = & \frac{1}{\sqrt{g}} \frac{\partial}{\partial x_1} \left( \sqrt{g} g^{11} \frac{\partial u}{\partial x_1} \right) + \frac{1}{\sqrt{g}} \frac{\partial}{\partial x_1} \left( \sqrt{g} g^{12} \frac{\partial u}{\partial x_2} \right) \\ & + \frac{1}{\sqrt{g}} \frac{\partial}{\partial x_2} \left( \sqrt{g} g^{21} \frac{\partial u}{\partial x_1} \right) + \frac{1}{\sqrt{g}} \frac{\partial}{\partial x_2} \left( \sqrt{g} g^{22} \frac{\partial u}{\partial x_2} \right). \end{aligned} \tag{4}$$

We refer [26] and [11] for more details about operators defined on manifolds.

In this section, we present a novel framework for solving PDEs (Equations (1) and (2)) on surfaces represented by point clouds. The proposed method comprises two main steps. First, the surface is locally approximated with respect to a local coordinate system at each point. In the second step, we solve PDEs on the point clouds by adopting a physics-informed learning framework utilizing the geometric quantities obtained in the first step. Each step is described in detail in Section 3.1 and Section 3.2, respectively.

## *3.1 Surface reconstruction from point cloud*

In order to solve PDEs on point clouds, it is crucial to extract geometric information of the points, which means that the underlying surface should be reconstructed effectively and robustly from the input point cloud data. Here we obtain local parameterization of the surface, which are directly used when computing function values and their derivatives along the surface in the PDE-solving step. Our surface reconstruction step consists of the following substeps: (1) surface normal estimation, (2) local approximation of surface.

### *3.1.1 Surface normal estimation*

We first introduce a graph learning method to estimate the normal vector on the surface. The estimated normal vector and two tangent vectors define the local coordinate system for each point. Since the tangent vectors can be selected as two independent vectors perpendicular to the normal vector, it is enough to obtain the normal vectors in the first step.

Here we use a variant of GN [4]. The GNs can extract geometric features efficiently from unstructured position information of points by learning the transmission of information between the points in a neighbor. Such an approach has recently shown good generalization performance for physical problems [33, 34]. Our experimental results corroborate this, demonstrating that the proposed model trained on relatively simple geometries performs well on highly complicated surfaces.

**Model architecture** The proposed model consists of three main components: an encoder, a processor, and a decoder (Figure 9 in Appendix A [19]).

First, the encoder converts the point cloud, $X$, to a simple directed graph structure $G^0 = (V^0, E^0)$. To be specific, it converts $X$ into a $K$-nearest neighbor ($K$NN) graph $G = (X, E)$ for some $K$ and then embeds into the latent graph $G^0$. Here, $\mathbf{r}_{ji} \in E$ if and only if $j \in N(i)$ where $N(i)$ is the set of indices of $K$NN of the node $x_i$. Initially, every node feature, $v_i^0 \in V^0$, is set to zero. For each edge, $e_{ji}^0 := \epsilon^e(\mathbf{r}_{ji})$ for some learnable multi-layer perceptron (MLP), $\epsilon^e$, where $\mathbf{r}_{ji}$ can be any property representing information between senders and receivers, e.g., displacement. In practice, we use a pointwisely normalized displacement for $\mathbf{r}_{ji}$, i.e. $\mathbf{r}_{ji} = \frac{1}{d_i}(\mathbf{x}_i - \mathbf{x}_j)$, where $d_i = \max_{j \in N(i)} \|x_i - x_j\|_2$. This allows the edge feature to keep geometric information while normalizing the size of the vectors.

Subsequently, the process updates the latent graph $M$ times by the Message Passing algorithm. The Message Passing propagates information between node and edges in the latent space. We use identical models for each update—so all parameters are shared.

Following the feature update, the graph is returned to the original point cloud and normal vectors are estimated using updated features by the decoder. The normal vector corresponding to each node is predicted to be $\tilde{\mathbf{n}}_i := \delta^v(v_i^M)$ for some learnable MLP, $\delta^v$.

**Loss function and regularization for continuity** The model is trained in a supervised manner. To measure an error between the estimated normal vector $\tilde{\mathbf{n}}_i$ and the ground truth $\mathbf{n}_i$, we use a modified cosine distance defined by

$$d(\tilde{\mathbf{n}}_i, \mathbf{n}_i) = 1 - |\tilde{\mathbf{n}}_i \cdot \mathbf{n}_i| / (\|\tilde{\mathbf{n}}_i\|_2 \|\mathbf{n}_i\|_2).$$

This means that the error becomes zero when two vectors are equal up to sign.

In addition, we introduce a regularizing term to impose continuity of the output along the surface. As every regular orientable surface has a differentiable normal vector field, we expect our predictions to be continuous. In other words, we want the normal vectors to vary consistently and gradually over the surface, so that close points correspond to similar normal vectors. This can be achieved by penalizing rapid changes in neighboring normal vectors. Given $K_{reg}$, let $N_{reg}(i)$ denote the set of indices of $K_{reg}$-NN for each $\mathbf{x}_i$. The proposed regularizer is written as

$$L_{reg} = \frac{1}{K_{reg}} \sum_{i=1}^{N} \sum_{k \in N_{reg}(i)} d(\tilde{\mathbf{n}}_k, \tilde{\mathbf{n}}_i).$$

By adding this regularization term, the model is expected to be continuous, especially near boundaries, and robust to complex surfaces. In our implementation, we simply choose $K_{reg} = K$.

**Training dataset** For training, we only use elliptic tori. An elliptic torus is a very simple geometrical object (Figure 2(a)), but by changing its parameters, one can generate various surfaces with different curvatures. In addition, there are some additional processes to augment the dataset: random rotation, random sampling, and truncation.

First, every data is randomly rotated for data augmentation. Next, each point cloud data is randomly sampled at a rate of 20%, to enhance our model's robustness on a non-uniform point cloud data. Finally, we segment half of the dataset and cut them by a plane (Figure 2(b)). By adding truncated data, our model learns not only closed

surface data, but also open surfaces which contains boundaries. More details for the training data are presented in Appendix D.1.1[19]. Although the training dataset is consist of simple geometrical domains, it is experimentally confirmed that our model shows excellent generalization performance for various domains.

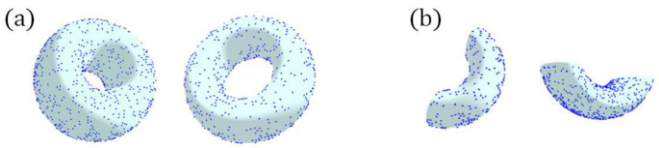


**Figure 2.** Examples of training data for GNs. The dataset consists of **(a)** whole tori and **(b)** cut tori.

### 3.1.2 *Local approximation of surface*

Using the estimated normal vector, a local coordinate system at each point can be constructed. We consider a tangent plane at point $\mathbf{x}_i$ which is normal to $\tilde{\mathbf{n}}_i$, and select two orthonormal basis vectors in the plane, namely $\mathbf{t}_i^{(1)}, \mathbf{t}_i^{(2)}$. These three vectors form a local coordinate system $\langle \mathbf{x}_i; \mathbf{t}_i^{(1)}, \mathbf{t}_i^{(2)}, \tilde{\mathbf{n}}_i \rangle$ around $\mathbf{x}_i$. We now reconstruct the surface from the point cloud by approximating local parameterization. To obtain a local approximation around $\mathbf{x}_i$, we fit a bivariate polynomial of degree two with respect to $\langle \mathbf{x}_i; \mathbf{t}_i^{(1)}, \mathbf{t}_i^{(2)}, \tilde{\mathbf{n}}_i \rangle$ using the MLS. Specifically, we determine a local parameterization, $\Gamma_i = (x, y, z_i(x, y))$, where $z_i(x, y) = a_1 + a_2 x + a_3 y + a_4 x^2 + a_5 xy + a_6 y^2$. We fit the undetermined coefficients, $a_1$, ..., $a_6$, by minimizing the weighted least-squares sum

$$\sum_{j \in N(i)} w\left(\|\mathbf{x}_j - \mathbf{x}_i\|_2\right) \left(z_i(x_j, y_j) - z_j\right)^2,$$

where $(x_j, y_j, z_j)$ is the local coordinates of $K$NN of $\mathbf{x}_i$. The resulting parameterization of the underlying surface provides an analytic expression to evaluate function values and its derivatives on the surface. Also, we can leverage the ability of graph learning method in the surface reconstruction step, such as the generalization capability and the robustness to input data quality. The evaluation of the proposed method will be presented with several experiments.

## 3.2 *Solving PDEs on point clouds*

In this section, we describe the incorporation of the surface reconstruction with the PINN frameworks to solve PDEs on surfaces represented by point clouds. Based on the local parameterization of the surface, we first derive the approximation of functions and their derivative on surfaces. Theses results will be directly used in PDE residual losses during training the PDE solution approximator in the PINN framework.

### 3.2.1 *Functions and derivatives on surfaces*

First, we turn our attention to approximations of functions and their derivatives on surfaces. A key component of solving PDEs on surfaces is the approximation of differential operators on surfaces—this is the primary difference compared to the solving of standard PDEs on flat domains. We again use MLS to approximate a given function, $u$, locally near a point $\mathbf{x}_i$ in the local coordinate system $\langle \mathbf{x}_i; \mathbf{t}_i^{(1)}, \mathbf{t}_i^{(2)}, \tilde{\mathbf{n}}_i \rangle$. In our proposal, $u_i$ is assumed to be a bivariate polynomial of degree two, $u_i(x, y) = b_1 + b_2 x + b_3 y + b_4 x^2 + b_5 xy + b_6 y^2$, where the coefficients $b_1$, ..., $b_6$ are obtained by minimizing the following weighted least-squares sum:

$$\sum_{j \in N(i)} w\left(\|\mathbf{x}_j - \mathbf{x}_i\|_2\right) \left(u_i(x_j, y_j) - u(\mathbf{x}_i)\right)^2.$$

The resulting coefficients provides an explicit expression $u_i(x, y)$ of $u$ near $\mathbf{x}_i$. Now we can derive the surface derivatives such as $\nabla_{\mathcal{M}}$ and $\Delta_{\mathcal{M}}$ using the coefficients of $z_i$ and $u_i$.

For instance, we rewrite each term in the Laplace-Beltrami operator $\Delta_{\mathcal{M}}$ as some function of coefficients of $z_i$ and $u_i$. That is, if we rewrite the Laplace-Beltrami operator at the node $\mathbf{x}_i$ as

$$\begin{aligned}\Delta_{\mathcal{M}} u_i(\mathbf{x}_i) = A_1 \frac{\partial u_i}{\partial x}(\mathbf{x}_i) + A_2 \frac{\partial u_i}{\partial y}(\mathbf{x}_i) \\ + A_3 \frac{\partial^2 u_i}{\partial x^2}(\mathbf{x}_i) + A_4 \frac{\partial^2 u_i}{\partial x \partial y}(\mathbf{x}_i) + A_5 \frac{\partial^2 u_i}{\partial y^2}(\mathbf{x}_i),\end{aligned}$$

then each coefficient $A_i$ and derivative term can be computed using the coefficients $a_i$ and $b_i$. For instance, we can derive

$$A_4 = \frac{-2a_2 a_3}{1 + a_2^2 + a_3^2}, \quad A_5 = \frac{1 + a_2^2}{1 + a_2^2 + a_3^2},$$

and $\frac{\partial^2 u_i}{\partial x \partial y}(\mathbf{x}_i) = b_5$, $\frac{\partial^2 u_i}{\partial y^2}(\mathbf{x}_i) = 2b_6$. Other terms can be calculated in the same way, and the surface gradient $\nabla_{\mathcal{M}}$ can be obtained similarly. The detailed derivation is presented in the Appendix C [19].

### 3.2.2 *Physics-informed learning on surfaces*

In this section, we introduce a method to solve PDEs on surfaces by combining all the aforementioned ingredients. The main strategy is to use neural network models as approximators of the solutions to the problems, Equations (1) and (2), by minimizing the residuals of the equations via gradient-based optimizers. The neural networks use spatial (and temporal) coordinates as the input and the solution at the same point as the output, respectively. Then, the PDE residuals are computed as a loss function to embed physical information into the network. In conventional physics-informed learning, each derivative is computed using automatic differentiation. Instead, here we use the results obtained in the previous section to compute the derivatives along surfaces. These significantly reduce the computational cost when solving PDEs on surfaces, since they do not involve any automatic differentiation procedure. We note that, in our experiments the auto-differentiation is only used to compute time derivatives $\partial/\partial t$.

We denote the neural network approximation of the solution by $u^{NN}$. For a time-independent problem (Equation (1)), the parameters of the network $u^{NN}$ are updated by minimizing the mean-squared error of the residuals of the PDEs:

$$\begin{aligned}Loss\left(u^{NN}\right) = \frac{1}{N} \sum_{i=1}^{N} \left[\mathcal{L} u^{NN}(\mathbf{x}_i) - f(\mathbf{x}_i)\right]^2 \\ + \frac{\lambda_B}{N_B} \sum_{j=1}^{N_B} \left[\mathcal{B} u^{NN}(\mathbf{x}_j) - g(\mathbf{x}_j)\right]^2\end{aligned} \tag{5}$$

for the given training points, $\{\mathbf{x}_i \in \mathcal{M}\}_{i=1}^{N}$ in the interior and $\{\mathbf{x}_j \in \partial\mathcal{M}\}_{j=1}^{N_B}$ at the boundary, with a loss weight of $\lambda_B$. For

time-dependent problems (Equation (2)), the following loss function can be considered:

$$\begin{aligned} &Loss\left(u^{NN}\right) \\ &= \frac{1}{N}\sum_{i=1}^{N}\left[\frac{\partial u^{NN}(t_i,\mathbf{x}_i)}{\partial t} + \mathcal{L}u^{NN}(t_i,\mathbf{x}_i) - f(t_i,\mathbf{x}_i)\right]^2 \\ &+ \frac{\lambda_B}{N_B}\sum_{j=1}^{N_B}\left[\mathcal{B}u^{NN}(t_j,\mathbf{x}_j) - g(t_j,\mathbf{x}_j)\right]^2 \\ &+ \frac{\lambda_I}{N_I}\sum_{k=1}^{N_I}\left[u^{NN}(0,\mathbf{x}_k) - h(\mathbf{x}_k)\right]^2 \end{aligned} \tag{6}$$

for training points $\{(t_i,\mathbf{x}_i)\in(0,T)\times\mathcal{M}\}_{i=1}^{N}$, $\{(t_j,\mathbf{x}_j)\in(0,T)\times\partial\mathcal{M}\}_{j=1}^{N_B}$, and $\{\mathbf{x}_k\in\mathcal{M}\}_{k=1}^{N_I}$, with loss weights of $\lambda_B$ and $\lambda_I$. We note that the model can be modified to satisfy the initial or boundary conditions exactly. In our experiments on time-dependent problems, the output of the model is set to be $u^{NN}(t,\mathbf{x}) = h(\mathbf{x}) + t\cdot\tilde{u}^{NN}(t,\mathbf{x})$, such that the model always satisfies the prescribed initial condition, $u^{NN}(0,\mathbf{x}) = h(\mathbf{x})$, for any $\mathbf{x}\in\mathcal{M}$. This enables the omission of the loss term for the initial condition in Equation (6).

## 4 Experiments

In this section, we compare the performance of the proposed method with those of existing methods. We first evaluate our surface reconstruction method in Section 4.1. Next we demonstrate the application of our method with representative examples for PDEs on point clouds. From Section 4.2 to Section 4.5, we conduct error analysis on synthetic datasets with known analytic solutions. We adopted the relative error in $L_2$ and max norm as the error criteria. In addition, we discuss examples that are closer to real-world problems in Section 4.6.

As a baseline for the PDE-solving step, we consider three baseline models: (1) the numerical method using PCA with FDM [26], and the learning-based methods (2) DeepFit [5] and (3) AdaFit [42]. Note that both DeepFit and AdaFit are utilized only for the first part, surface reconstruction, and each of them is combined with PINNs for the second part, solving PDEs on point clouds. More details for implementation and baseliens are described in Appendix C and D [19]. Our code and data are publicly available at *https://github.com/JSRYU1998/surface_pde*.

### 4.1 Surface Normal Estimation

First, the normal estimation performance of the proposed GN-based model is evaluated and compared with those of baselines on various surfaces. We verify the predictive robustness of the proposed model as the underlying surface becomes increasingly more complex for a given point cloud.

We consider point cloud data sampled from a *curvature flow* generated by the methods presented in [9]. Curvature flow is a sequence of smooth approximations for a given geometry. In our experiment, sequences starting from the *Stanford bunny* data and converging to a sphere are considered by reducing the fairing energy (Figure 3). The errors between the estimated normal vectors and the ground truth data are measured in terms of the modified cosine distance. As with Section 4.6, the dataset is initially given as triangular meshes, so we sample point clouds uniformly with $N = 4800$.

Figure 4 indicates that the proposed graph-based method computes normal vectors most accurately in earlier time steps compared to other methods. When domain is relatively simple, all methods show similar performance. As the complexity increased, however, only our model remains robust against such complexity. In particular, the performance degradation in DeepFit was the most pronounced, which, as described in [42], is due to an increasing inconsistency in the fitting polynomial order of each point as the domain becomes more complex. This implies that our model can capture more detailed features in more complex geometries than the baselines. This also implies that the proposed model is more applicable to realistic domains.

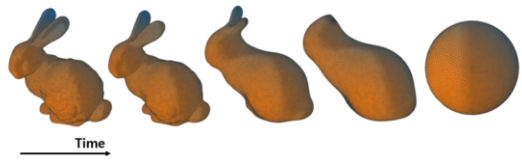


**Figure 3.** Visualization of the curvature flow from the bunny to a sphere.

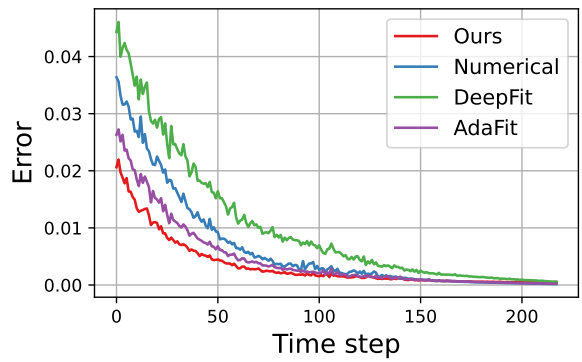


**Figure 4.** Error over time when using our model and the baselines.

### 4.2 Elliptic equation on an open surface with Dirichlet boundary condition

We consider the following elliptic equation on a surface $\mathcal{M}$ with a boundary $\partial\mathcal{M}$

$$\begin{aligned} \Delta_{\mathcal{M}} u &= f \text{ on } \mathcal{M}, \\ u &= 0 \text{ on } \partial\mathcal{M}, \end{aligned} \tag{7}$$

where the surface $\mathcal{M}$ is defined by the level set

$$y^2 + z^2 = \left(\sqrt{1-0.25x^2}\left(0.75x^2+0.5\right)\right)^2$$

with an inequality constraint $x \le 0.75$ and the boundary region at the equality constraint (Figure 6(a)). This is one of the fundamental problems associated with mechanical and physical phenomena on surfaces. We set the analytic solution to be $u(x,y,z) = (0.75 - x)\sin(y - z)$ and compute the term $f$ accordingly, such that Equation (7) holds. Point clouds are generated using different mesh sizes, $h\in[0.08, 0.22]$. The corresponding $N$ vary from 687 to 5204 (Figure 6(b)).

As reported in Figure 5 and Table 3 in Appendix F [19], ours achieve better accuracy compared to baselines for all cases, confirming that our model can successfully solve boundary value problems.

### 4.3 Diffusion equation on closed surfaces

Next, we consider a time-dependent diffusion equations on closed surfaces. We set up three synthetic surfaces with varying complexity,

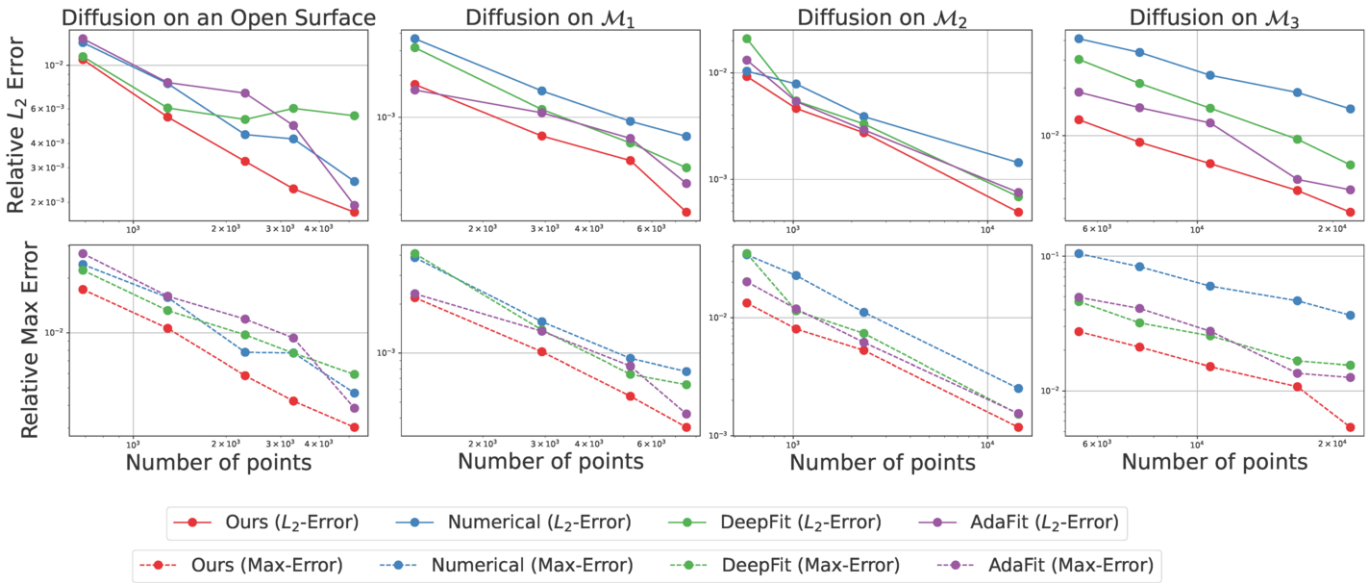


**Figure 5.** Comparison of the results for Section 4.2 and Section 4.3. The relative errors are plotted in the log-log scale.

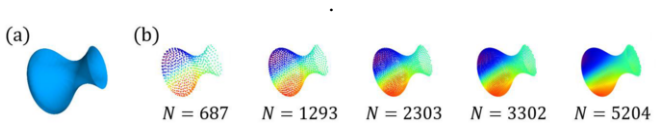


**Figure 6.** Illustration of **(a)** the underlying surface and **(b)** the sampled point cloud data, ranging from sparse to dense. The color in (b) represents the values of the approximate solution obtained from our model.

formulated as follows: $r = 1$, $r = 1 + 0.4\sin(3\theta)\cos\phi$, $r = 1 + 0.4\sin(5\theta)\cos\phi$, for $\mathcal{M}_1, \mathcal{M}_2, \mathcal{M}_3$, respectively, where $(r, \theta, \phi)$ is the spherical coordinate. As depicted in Figure 7(a), (b), and (c), the complexity of surfaces is controlled by manipulating the the value of polar and azimuthal angles.

The equations are given by:

$$\begin{aligned} u_t - \Delta_{\mathcal{M}_i} u &= f \text{ in } (0,1) \times \mathcal{M}_i, \\ u(0, \mathbf{x}) &= h_i(\mathbf{x}) \text{ on } \mathcal{M}_i \end{aligned} \tag{8}$$

for $i$= 1, 2 and 3. We choose the following exact solutions $u(t, x, y, z) = \sin(x + \sin(t)) \exp(\cos(y - z))$ for all $i$'s, and then $h_i$ and $f_i$ are determined accordingly.

We compare the performance of our method with those of the other methods when domain complexity changes. In this experiment, points are sampled from each domain uniformly. As described in Figure 5 and Table 3 in Appendix F [19], the proposed model achieves better performace than baselines on almost cases. In particular, on $\mathcal{M}_3$, which is the most complex domain, our model significantly outperforms the baseline models, showcasing its superior robustness with respect to domain complexity. Finally, we provide qualitative results by visualizing the diffusion of heat on the domains over time in Figure 7(d).

### 4.4 *Diffusion equation on non-uniformly distributed point clouds*

We solve Equation (8) with the same exact solution above on a cheese-like domain with the level set

$$\begin{aligned} (4x^2-1)^2 + (4y^2-1)^2 + (4z^2-1)^2 + 16(x^2+y^2-1)^2 \\ +16(y^2+z^2-1)^2 + 16(z^2+x^2-1)^2 - 16 = 0. \end{aligned}$$

The main difference compared to Section 4.3 lies in the use of non-uniformly distributed point clouds in this section (Figure 7(e)). The experiment begins by uniformly and densely sampling points ($N = 13824$) from the surface. Subsequently, we randomly select each point with probabilities of 0.5, 0.2, and 0.1, as illustrated in Figure 7(e). To evaluate the performance, we conduct 10 runs for each probability setting, varying the random seeds. It should be noted that the domains are no longer uniform after random sampling.

The results presented in Table 1 demonstrate that our method consistently maintains a high level of performance even when confronted with non-uniform point distributions. Both DeepFit and AdaFit (with PINN) method perform worse overall compared to our model. Additionally, AdaFit with PINN exhibited overall instability, resulting in significant errors, especially in the 50% sampling case. Furthermore, the numerical method [26] fails to converge in any of the cases, with the error exponentially increase as the iterative updates progress. It assumes that the point clouds are well-distributed with a sufficiently fine ratio; therefore, their model fails to converge in this experiment.

However, in contrast, our model is most robust on non-uniform point distribution as it is trained on a non-uniform dataset for normal vector estimation, enabling surface reconstruction even when the points are distributed non-uniformly.

### 4.5 *Advection Equation on a noisy surface*

In this section, we investigate the robustness of our approach in the presence of varying noise intensities in comparison to the baselines.

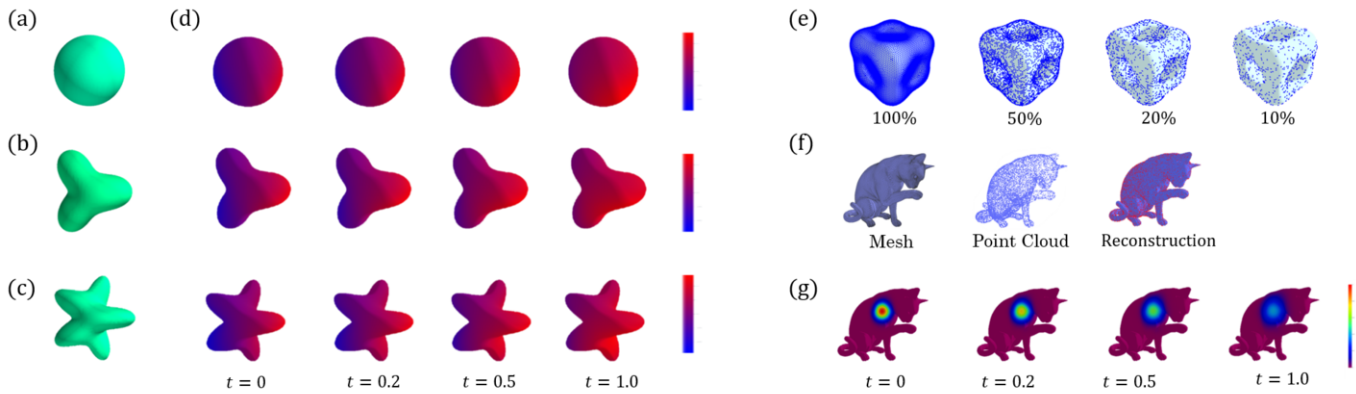


**Figure 7.** **(a), (b), (c)** Illustration of closed surfaces $\mathcal{M}_1$, $\mathcal{M}_2$, and $\mathcal{M}_3$. **(d)** We visualize the solutions approximated by our model, which correspond to the heat diffusion over time. **(e)** Point cloud data with non-uniform distribution in Section 4.4 **(f)** Visualizations of some examples from the TOSCA dataset in Section 4.6. **(g)** The visualization of diffusion equation as time goes, which indicates that heat spreads as predicted. For **(d)** and **(g)**, triangular meshes are used for better visualization. We note that the mesh information is only used for visualization, not as part of the main algorithm.

**Table 1.** Relative $L_2$ errors corresponding to each sampling ratio on a cheese-like surface. NaN means that the predicted values explode, resulting in a failure of convergence. (Averaged over 10 repetitions)

| Sampling Ratio | Ours | DeepFit +PINN | AdaFit +PINN | Numerical method |
|---|---|---|---|---|
| 50% | **1.29e-2** | 1.80e-2 | 2.44e-1 | NaN |
| 20% | **3.69e-2** | 3.98e-2 | 2.80e-2 | NaN |
| 10% | **5.56e-2** | 9.07e-2 | 5.94e-2 | NaN |

We consider the advection equation

$$\begin{aligned} \frac{\partial u}{\partial t} + \mathbf{v} \cdot \nabla_{\mathcal{M}} u = 0 \text{ on } (0,T) \times \mathcal{M}, \\ u(0,\theta,\phi) = u_0(\theta,\phi) \text{ on } \mathcal{M} \end{aligned} \tag{9}$$

on a torus $\mathcal{M}$, parametrized by $((R + r\cos\phi)\cos\theta, (R + r\cos\phi)\sin\theta, r\sin\phi)$ for $0 \le \theta, \phi < 2\pi$ with $R = 1$ and $r = 0.5$. Here, $\mathbf{v}$ denotes the velocity vector field and $u_0(\theta,\phi)$ denotes the initial condition. We set the analytic solution to be $u(t,\theta,\phi) = \cos(\theta - t) + \sin(\phi - 2t)$, and then compute $\mathbf{v}$ and $u_0$ accordingly. This equation governs the transport of a substance on a fluid surface. To that end, we first sample point cloud of $N = 4560$ from the surface and then inject i.i.d. Gaussian noise $\mathcal{N}(0, \sigma^2 I)$ to each point. The intensity of the noise, denoted as $\sigma$, is adjusted in proportion to the spread of the points. As with Section 4.4, we iterates 10 times for generation of noise. We vary $\sigma$ from 0.25% to 1.0% of the maximum distance from the centroid of each point cloud. We evaluate the approximate solutions at the terminal time $T = \pi$.

As in Table 2, our model achieved superior performance compared to baselines. Specifically, when $\sigma$ increases, the numerical method [26] diverges exponentially, resulting in a failure to converge. On the contrary, our method demonstrates only a minor decline in performance when the intensity increases, and performs better than baselines.

## 4.6 *Diffusion equation on real-world dataset*

The final experiment concerns Equation (8) on more complex domains from real world. We consider the TOSCA dataset [6], which

**Table 2.** Relative $L_2$ errors corresponding to each noise intensity on a torus. NaN means that the predicted values explode, resulting in a failure of convergence. (Averaged over 10 repetitions)

| Noise Intensity | Ours | DeepFit +PINN | AdaFit +PINN | Numerical method |
|---|---|---|---|---|
| 0.25% | **9.96e-3** | 1.13e-2 | 1.07e-2 | 2.90e-2 |
| 0.50% | **1.80e-2** | 2.73e-2 | 2.90e-2 | 4.89e-1 |
| 0.75% | **3.30e-2** | 5.06e-2 | 6.25e-2 | NaN |
| 1.00% | **9.43e-2** | 1.25e-1 | 1.49e-1 | NaN |

consists of 80 objects, including various poses of humans and animals (see Figure 7(f) for example). Since the TOSCA dataset is initially given as triangular meshes, represented by vertices and triangular faces, we sample point clouds uniformly in a manner proportional to the area of each triangular face. Consequently, each data consists of $N = 12000$ points (Figure 7(f)).

For the initial condition, we randomly select a single source point $\mathbf{s}$ from the point cloud and introduce sharp and bell-shaped Gaussian pulse in its vicinity:

$$u(\mathbf{x}, 0) = \exp(-75 \cdot \|\mathbf{x} - \mathbf{s}\|_2^2)$$

Thus a solution to this problem describes a heat diffusion from a single point along the surfaces.

Since the exact expressions for these domains are not available, our model only presents qualitative results through visualizations of surface reconstruction and heat diffusion in Figure 7(f) and (g), respectively. In Figure 7(f), the reconstructed surface resulting from the first step of our method are described, and Figure 7(g) shows the heat diffusion predictions for each example. The successful spreading of heat on the surfaces, as expected, confirms the strong performance of our model, even in more complex domains. More examples for other data are demonstrated in Appendix G [19]. This experiment demonstrates the potential of applying our proposed method to real-world problems.

## 4.7 *Further Experiments*

We lastly demonstrate supplemental experiments for in-depth analysis of our framework. We first evaluate the normal estimation performance on various surfaces of our model. We next investigate the

time required of each step of our model. We conduct further ablation studies for a deeper analysis of several properties of our model. These experimental results support the effectiveness of the proposed method in different cases. More details are described in Appendix H and I [19].

## 5 Conclusion

In this paper, we propose a GN-based method to solve PDEs on surfaces, particularly those described using point clouds. The method is trained on a simple geometry to reconstruct the surface locally; however, it performs well on several complicated domains as well. The model can be used to solve several types of PDEs on curved surfaces. Based on quantitative and qualitative experimental results, we conclude that it is robust even when the surface becomes highly complex and points are distributed non-uniformly.

Several types of future research may be based on this framework. For example, the proposed method can be applied to moving surface problems. We have only considered stationary domains and utilized only the spatial features. By infusing both spatial and temporal features into our model, various problems on evolving surfaces can be solved.

## Acknowledgements

This work was supported by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MSIT) (RS-2023-00219980 and RS-2022-00165268) and by the Ministry of Education (2023RIS-009) and supported by Institute for Information & Communications Technology Promotion (IITP) grant funded by the Korea government(MSIP) (No. 2019-0-01906, Artificial Intelligence Graduate School Program (POSTECH)).

## References

[1] M. Alexa, J. Behr, D. Cohen-Or, S. Fleishman, D. Levin, and C. T. Silva. Computing and rendering point set surfaces. *IEEE Transactions on visualization and computer graphics*, 9(1):3–15, 2003.

[2] M. Atzmon and Y. Lipman. Sal: Sign agnostic learning of shapes from raw data. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2565–2574, 2020.

[3] P. Battaglia, R. Pascanu, M. Lai, D. Jimenez Rezende, et al. Interaction networks for learning about objects, relations and physics. *Advances in neural information processing systems*, 29, 2016.

[4] P. W. Battaglia, J. B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner, et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.

[5] Y. Ben-Shabat and S. Gould. Deepfit: 3d surface fitting via neural network weighted least squares. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part I 16*, pages 20–34. Springer, 2020.

[6] A. M. Bronstein, M. M. Bronstein, and R. Kimmel. *Numerical geometry of non-rigid shapes*. Springer Science & Business Media, 2008.

[7] M. K. Chung and J. Taylor. Diffusion smoothing on brain surface via finite element method. In *2004 2nd IEEE International Symposium on Biomedical Imaging: Nano to Macro (IEEE Cat No. 04EX821)*, pages 432–435. IEEE, 2004.

[8] F. S. Costabal, S. Pezzuto, and P. Perdikaris. $\Delta$-pinns: physics-informed neural networks on complex geometries. *arXiv preprint arXiv:2209.03984*, 2022.

[9] K. Crane, U. Pinkall, and P. Schröder. Robust fairing via conformal curvature flow. *ACM Transactions on Graphics (TOG)*, 32(4):1–10, 2013.

[10] K. Crane, C. Weischedel, and M. Wardetzky. Geodesics in heat: A new approach to computing distance based on heat flow. *ACM Transactions on Graphics (TOG)*, 32(5):1–11, 2013.

[11] G. Dziuk and C. M. Elliott. Finite element methods for surface pdes. *Acta Numerica*, 22:289–396, 2013.

[12] Z. Fang and J. Zhan. A physics-informed neural network framework for pdes on 3d surfaces: Time independent problems. *IEEE Access*, 8: 26328–26335, 2019.

[13] S. Fleishman, D. Cohen-Or, and C. T. Silva. Robust moving least-squares fitting with sharp features. *ACM transactions on graphics (TOG)*, 24(3):544–552, 2005.

[14] H. Gao, M. J. Zahr, and J.-X. Wang. Physics-informed graph neural galerkin networks: A unified framework for solving pde-governed forward and inverse problems. *Computer Methods in Applied Mechanics and Engineering*, 390:114502, 2022.

[15] A. Gropp, L. Yariv, N. Haim, M. Atzmon, and Y. Lipman. Implicit geometric regularization for learning shapes. *arXiv preprint arXiv:2002.10099*, 2020.

[16] T. Groueix, M. Fisher, V. G. Kim, B. C. Russell, and M. Aubry. A papier-mâché approach to learning 3d surface generation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 216–224, 2018.

[17] J. Han, A. Jentzen, and E. Weinan. Solving high-dimensional partial differential equations using deep learning. *Proceedings of the National Academy of Sciences*, 115(34):8505–8510, 2018.

[18] W.-F. Hu, Y.-J. Shih, T.-S. Lin, and M.-C. Lai. A shallow physics-informed neural network for solving partial differential equations on surfaces. *arXiv preprint arXiv:2203.01581*, 2022.

[19] R. Hwang, J. Ryu, S. Park, and H. J. Hwang. Solving pdes on point clouds by physics-informed learning with graph neural networks. *Available at SSRN 4634151*.

[20] A. D. Jagtap and G. E. Karniadakis. Extended physics-informed neural networks (xpinns): A generalized space-time domain decomposition based deep learning framework for nonlinear partial differential equations. In *AAAI spring symposium: MLPS*, volume 10, 2021.

[21] W. Ji, W. Qiu, Z. Shi, S. Pan, and S. Deng. Stiff-pinn: Physics-informed neural network for stiff chemical kinetics. *The Journal of Physical Chemistry A*, 125(36):8098–8106, 2021.

[22] X. Jin, S. Cai, H. Li, and G. E. Karniadakis. Nsfnets (navier-stokes flow nets): Physics-informed neural networks for the incompressible navier-stokes equations. *Journal of Computational Physics*, 426:109951, 2021.

[23] A. M. Jones, P. A. Bosler, P. A. Kuberry, and G. B. Wright. Generalized moving least squares vs. radial basis function finite difference methods for approximating surface derivatives. *Computers Mathematics with Applications*, 147:1–13, 2023. ISSN 0898-1221. doi: https://doi.org/10.1016/j.camwa.2023.07.015. URL https://www.sciencedirect.com/science/article/pii/S0898122123003097.

[24] R. Li, X. Li, C.-W. Fu, D. Cohen-Or, and P.-A. Heng. Pu-gan: a point cloud upsampling adversarial network. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 7203–7212, 2019.

[25] R. Li, X. Li, P.-A. Heng, and C.-W. Fu. Point cloud upsampling via disentangled refinement. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 344–353, 2021.

[26] J. Liang and H. Zhao. Solving partial differential equations on point clouds. *SIAM Journal on Scientific Computing*, 35(3):A1461–A1486, 2013.

[27] Z. Mao, A. D. Jagtap, and G. E. Karniadakis. Physics-informed neural networks for high-speed flows. *Computer Methods in Applied Mechanics and Engineering*, 360:112789, 2020.

[28] M. Penwarden, A. D. Jagtap, S. Zhe, G. E. Karniadakis, and R. M. Kirby. A unified scalable framework for causal sweeping strategies for physics-informed neural networks (pinns) and their temporal decompositions. *J. Comput. Phys.*, 493(C), jan 2024. ISSN 0021-9991. doi: 10.1016/j.jcp.2023.112464. URL https://doi.org/10.1016/j.jcp.2023.112464.

[29] A. Petras, L. Ling, and S. J. Ruuth. An rbf-fd closest point method for solving pdes on surfaces. *Journal of Computational Physics*, 370: 43–57, 2018.

[30] M. Raissi, P. Perdikaris, and G. E. Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707, 2019.

[31] S. J. Ruuth and B. Merriman. A simple embedding method for solving partial differential equations on surfaces. *Journal of Computational Physics*, 227(3):1943–1961, 2008.

[32] F. Sahli Costabal, Y. Yang, P. Perdikaris, D. E. Hurtado, and E. Kuhl. Physics-informed neural networks for cardiac activation mapping. *Frontiers in Physics*, 8:42, 2020.

[33] A. Sanchez-Gonzalez, N. Heess, J. T. Springenberg, J. Merel, M. Riedmiller, R. Hadsell, and P. Battaglia. Graph networks as learnable physics engines for inference and control. In *International Conference on Machine Learning*, pages 4470–4479. PMLR, 2018.

[34] A. Sanchez-Gonzalez, J. Godwin, T. Pfaff, R. Ying, J. Leskovec, and

P. Battaglia. Learning to simulate complex physics with graph networks. In *International Conference on Machine Learning*, pages 8459–8468. PMLR, 2020.

[35] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini. The graph neural network model. *IEEE transactions on neural networks*, 20(1):61–80, 2008.

[36] K. Shukla, A. D. Jagtap, J. L. Blackshire, D. Sparkman, and G. E. Karniadakis. A physics-informed neural network for quantifying the microstructural properties of polycrystalline nickel using ultrasound data: A promising approach for solving inverse problems. *IEEE Signal Processing Magazine*, 39(1):68–77, 2021.

[37] J. Sirignano and K. Spiliopoulos. Dgm: A deep learning algorithm for solving partial differential equations. *Journal of computational physics*, 375:1339–1364, 2018.

[38] Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein, and J. M. Solomon. Dynamic graph cnn for learning on point clouds. *ACM Transactions on Graphics (tog)*, 38(5):1–12, 2019.

[39] E. Weinan and B. Yu. The deep ritz method: a deep learning-based numerical algorithm for solving variational problems. *Communications in Mathematics and Statistics*, 6(1):1–12, 2018.

[40] F. Williams, T. Schneider, C. Silva, D. Zorin, J. Bruna, and D. Panozzo. Deep geometric prior for surface reconstruction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10130–10139, 2019.

[41] Q. Zhao, D. B. Lindell, and G. Wetzstein. Learning to solve pde-constrained inverse problems with graph networks. In *International Conference on Machine Learning*, pages 26895–26910. PMLR, 2022.

[42] R. Zhu, Y. Liu, Z. Dong, Y. Wang, T. Jiang, W. Wang, and B. Yang. Adafit: Rethinking learning-based normal estimation on point clouds. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 6118–6127, 2021.