Efficient Continuous Spatio-Temporal Simulation with Graph Spline Networks

Chuanbo Hua * 1  Federico Berto * 1  Stefano Massaroli 2  Michael Poli 3  Jinkyoo Park 1

Abstract

Complex simulation of physical systems is an invaluable tool for a large number of fields, including engineering and scientific computing. To overcome the computational requirements of high-accuracy solvers, learned graph neural network simulators have recently been introduced. However, these methods often require a large number of nodes and edges, which can hinder their performance. Moreover, they cannot evaluate continuous solutions in space and time due to their inherently discretized structure. In this paper, we propose GRAPHSPLINETNETS, a method based on graph neural networks and orthogonal spline collocation (OSC) to accelerate learned simulations of physical systems by interpolating solutions of graph neural networks. First, we employ an encoder-decoder message passing graph neural network to map the location and value of nodes from the physical domain to hidden space and learn to predict future values. Then, to realize fully continuous simulations over the domain without dense sampling of nodes, we post-process predictions with OSC. This strategy allows us to produce a solution at any location in space and time without explicit prior knowledge of underlying differential equations and with a lower computational burden compared to learned graph neural networks evaluating more space–time locations. We evaluate the performance of our approach in heat equation, dam breaking, and flag simulations with different graph neural network baselines. Our method shows is consistently Pareto efficient in terms of simulation accuracy and inference time, i.e. 3× speedup with 10% less error on flag simulation.

1. Introduction

Simulations of partial differential equations (PDEs) describing physical processes are an invaluable tool for an increasing number of disciplines. As a result, the scientific machine learning community has been focused on crafting computationally inexpensive yet accurate simulation methods to expand the range of applicability of dynamical system simulators. Traditional simulation methods (Houska et al., 2012), such as the first principle model solver and the generalized Gauss-Newton methods, can be costly in calculations: in particular, complex physical simulations need substantial computational resources to be performed. In recent years, PDEs simulators have been widely used in a variety of applied problems such as game physics engines (Lewin, 2021), Virtual Reality (VR), (Höll et al., 2018) and the metaverse (Taheri et al., 2021) where accurate simulators can help users immerse themselves in the virtual environment and make it more appealing. Thus, the need for a high-precision and low-calculation simulator becomes even more urgent and worthwhile.

Previous research has shown successes in applying deep learning for simulating a variety of PDEs (Raissi et al., 2019). Graph-based simulation methods (Sanchez-Gonzalez et al., 2020; Pfaff et al., 2021) have used graph neural networks as a natural representation of the simulation underlying discretized dynamics; their discretization is often a fundamental part of making simulations viable. These paradigms have proven successful in learning generalizable particle and mesh-based simulators thanks to the ability of graphs in capturing local and global phenomena while retaining properties such as spatial equivariance and translational invariance (Bronstein et al., 2021).

Despite their advantages, a significant downside of mesh-based graph models is their inherent structural discretization, making it hard to achieve physical space or time-space continuous simulations. The encoder–decoder strategy has hidden layers representing physical states out of mesh points, enabling a space-continuous representation (Alet et al., 2019). In addition, message passing neural network (MPNN) of derivatives, combined with numerical accumulation methods, work for time-continuous prediction (Iakovlev et al., 2020). However, these continuous
approaches request more mesh points for higher-accuracy simulations, which brings a heavy calculation workload.

On the other hand, collocation methods have considerable benefits in terms of the computational complexity of the number of collocation points and the ability to yield continuous results, such as smoothest spline collocation, modified spline collocation (Fairweather & Meade, 2020), and the Cubic Spline Orthogonal Spline Collocation (OSC) methods for PDEs (Bialecki & Fairweather, 2001).

In this paper, we introduce GraphSplineNets: by leveraging the OSC method, we can readily obtain space-time continuous, free form simulations by starting from an intrinsically discrete GNN without the need for explicit prior knowledge of the problem to be solved. We use the method as an efficient post-processing scheme which can be applied to several different GNN simulators. GraphSplineNets allow us to reduce the number of space and time sample points of the ad-hoc trained underlying graph module that leads to Pareto-efficient simulations in terms of solution accuracy and inference speed. We demonstrate the method on the heat equation PDE benchmark, on a dam-breaking particle-based meshless simulation, and on a mesh-based cloth simulation.

2. Related Work

We identify related works for this paper in the area of numerical methods and deep learning for PDE simulation and categorize them into three main areas, namely: deep learning for physical simulations, graph neural network simulators, and the relationship between collocation methods and deep learning.

Deep Learning for Physical Simulations Solving differential equations with deep neural networks has been an active research area to solve the issues of traditional PDEs solvers, which often suffer from unsustainable computational requirements and scalability issues that can hinder real-time applications (see Appendix C.1 for further insights). Deep neural networks have been shown to be a viable alternative to numerical methods to solve the issues of scalability and inference time requirements. Physics Informed Neural Networks (PINNs) (Raissi et al., 2019), which aid in both the solution and discovery of PDEs by using ad-hoc deep architectures and loss functions enforcing boundary conditions, have received considerable attention due to their flexibility in tackling a wide range of data-driven solutions and discovery of PDE, even though they have been shown suffer from unstable training and convergence issues (Wang et al., 2022). The problem of solving PDEs, which we deal with in this paper, has also been explored with convolutional neural networks (Guo et al., 2016; Bhatnagar et al., 2019) which naturally incorporate the inductive bias of spatial invariance. Another active area of research concerns the use of neural operators (Lu et al., 2019; Li et al., 2020a; Kovachki et al., 2021) which map between infinite-dimensional function spaces. Several software libraries that have been developed to deal with numerical methods for deep learning efficiently include Poli et al. (2020); Chen et al. (2020); Lu et al. (2021).

Graph Neural Network Simulators The use of graph neural networks (GNNs) to address the simulation of a system with a finite number of sample points has been investigated to address issues from other deep learning paradigms for simulation. GNNs extend other models as convolutional neural networks to irregular grids and also capture physical principles deriving from geometric deep learning such as spatial equivariance and permutation (Bronstein et al., 2021) while constraining interactions to local neighborhoods (see Appendix C.2 for further intuitions on the success of GNNs for simulation). Alet et al. (2019) introduces the Graph Element Networks architecture to model continuous underlying physical processes with no a-priori graph structure by modeling adaptively sampled points in a graph. Sanchez-Gonzalez et al. (2020) develops with the Graph-based Neural Simulator (GNS) paradigm a model which learns a system dynamic update by creating graph edges on neighbor finite particles and performing message passing: this is shown to faithfully contain rollout errors and generalize well to unseen conditions. Pfaff et al. (2021) extends the mesh-free GNS to mesh-based simulations: the resulting model can capture mesh-space interactions by having edges corresponding to the ones of the mesh and obtains cheaper simulations than the baseline numerical solvers. Other related works include applications to control (Sanchez-Gonzalez et al., 2018), the extension of neural operators to graphs (Li et al., 2020a,b), and hybrid approaches with graph networks with traditional fluid simulation solvers (de Avila Belbute-Peres et al., 2020).

Collocation Methods and Deep Learning While continuous-time graph models have been previously explored (Poli et al., 2019; Xhonneux et al., 2020) that can theoretically capture the system time evolution, space-continuous graph models still suffer from the problem of the inherent graph discretization. For this reason, representing the space and time-continuous nature of simulations have been mainly dealt by using interpolation methods such as linear interpolation in Alet et al. (2019). However, these methods may not be suitable for simulation due to their lack of differentiability that may be necessary for instance in control problems (Liang et al., 2019) as well as falling short of realistic simulation which veer often continuously differentiable (i.e. $C^1$ class). Unlike previous approaches, we employ the Orthogonal Spline Collocation (OSC) (Bialecki & Fairweather, 2001) method to efficiently
obtain $C^1$ class solutions to differential equations given few partition points and obtain both space and time continuous simulation based on an underlying GNN simulator. Other deep collocations methods include Guo et al. (2019), which shows an ad–hoc collocation method for the bending analysis of the Kirchoff plate which cannot be easily tackled with mesh–based methods since it requires $C^1$ continuity. Brink et al. (2021) introduces a deep–learning model based on feed–forward networks and collocation method to approximate a variety of strong–form PDEs. Unlike the paradigms mentioned above that rely on deep learning to obtain collocation weights, we employ the OSC method to obtain the weights which have theoretical guarantees on convergence. Moreover, the synergy of OSC with GNNs enables our module to tackle diverse problems without the need of crafting over–engineered schemes while efficiently balancing between solution accuracy and inference time.

such

3. Methodology

3.1. Problem Set

We consider a continuous dynamic PDE system with state $u(x, t) \in \mathbb{R}$ that evolves over time $t \in \mathbb{R}_+$ and bounded domain $x \in \Omega \subset \mathbb{R}^D$

\[
\begin{align*}
\mathcal{L}(u) &= f(x, t), (x, t) \in \Omega \times \mathbb{R}_+ \\
\mathcal{B}(u) &= g(x, t), (x, t) \in \partial \Omega \times \mathbb{R}_+ \\
u(x, 0) &= u_0(x), x \in \Omega
\end{align*}
\]

where $\mathcal{B}(\cdot)$ is the boundary condition and $u_0(\cdot)$ is the initial condition. Mark $\mathbf{X} = \{x_0, x_1, \cdots, x_N\} \in \mathcal{X}$ as a set of physical space locations of sample point. $\mathcal{Y}^t = \{y_0^t, y_1^t, \cdots, y_N^t\} \in \mathcal{Y}$ is the state of sample points at time $t$, i.e. $y_i^t = u(x_i, t)$.

A simulator $S : \mathcal{Y} \rightarrow \mathcal{Y}$ maps the current state of sample points to a future state with fixed timestep. Mark the prediction trajectory from $t_0$ to $t_K$ as $\{\mathbf{Y}^{t_0}, \mathbf{Y}^{t_1}, \cdots, \mathbf{Y}^{t_K}\}$. A simulation model $\mathcal{M}(\cdot; \theta) : \mathcal{Y} \rightarrow \mathcal{Y}$ with learnable parameter $\theta$ takes input $\mathbf{Y}^{t_k}$ and predicts next timestep state $\hat{\mathbf{Y}}^{t_{k+1}} = \mathcal{M}(\mathbf{Y}^{t_k}; \theta)$. The gap between prediction states and ground truth can be used as the loss function $\mathcal{L} = \|\hat{\mathbf{Y}}^{t_{k+1}} - \mathbf{Y}^{t_{k+1}}\|^2$.

In the rest of this paper, we will follow the format of notation in which the superscript denotes the time–space index and the subscript denotes the space index.

3.2. Graph OSC Network Architecture

The overall architecture of GRAPHSPLINE-NETS is shown in Figure 2. Given the initial state of the domain, we firstly employ graph neural networks to obtain discrete predictions. Then, we apply time–oriented collocation method and space–oriented collocation method on these discrete predictions to get simulation functions, generating time and space–continuous simulations. We describe our model in three parts: graph neural network, time–oriented orthogonal collocation, and space–oriented orthogonal collocation.

Graph neural network structure. We employed an Encoder–Processor–Decoder structure to predict sample point values at the next timestep.

The encoder will represent sample points in the input space as a node in latent graph space, where an adjacency matrix will be created to describe the connection. Symmetrically, the decoder will map the updated graph to output space, representing hidden features to physical space’s values for each sample point. A message passing neural network is applied to update node features in hidden graph layers dynamically.
The overall process of applying the OSC method. We will firstly base on rules to generate partition and collocation points, where the rules can be isometric distribution, Fundamental Solution Method (Katsurada & Okamoto, 1996), Gaussian–Legendre quadrature rule (De Boor & Swartz, 1973). Then, we define the simulator to be a series of $C^1$ continuous polynomials. To define the simulator’s parameters, we will generate an algebra problem by substituting values of collocation points to get the equations. Finally, we solve the algebra problem to get the parameters and then use the simulator to obtain values over the whole domain.

**Time-oriented OSC.** One sample point’s value changes from 0 to $T$ over time, following an Ordinary Differential Equation (ODE)

\[
\begin{cases}
  u(t) = f(t), t \in [0, T] \\
  u(0) = u_0(x_0) \\
  u(T) = u_T(x_0)
\end{cases}
\]

The primary technique for using the OSC approach is...
Figure 4. Running time and MSE error of interpolation methods and OSC method on [Left] 1-D non-linear PDEs and [Right] 2-D non-linear PDEs, by changing the number of collocation points from 10 to 20.

given in Figure 3. For time-oriented OSC we consider an isometric split of the temporal domain with $N$ partitions $\pi : 0 = t_0 < t_1 < \cdots < t_N = T$. We aim to find one polynomial under order $r$ on each partition and make these $N$ polynomials $C^1$ continuous. These polynomials have the degree of freedom $N(r - 1)$. To decide those parameters, we will select $r - 1$ collocation points in each partition to decide those parameters. Note that these collocation points can be isometric or non-isometric: we will leave them isometric for the sake of graph neural network prediction.

Then given an initial value of $t_0 = 0$, the graph neural network will rollout predict a series of values at these collocation points. By substituting locations and values of collocation points to polynomials, we can transfer this ODE problem to algebraic equations. Notice that the coefficient matrix of this algebraic equation is also ABD (De Boor & Swartz, 1973). Then we have in total $N^D(r - 1)^D$ collocation points in the space.

We can get prediction values at collocation points by graph neural network and time-oriented OSC. Then, we substitute locations and values to polynomials to transfer the original problem to an algebraic equation. Note that the coefficient matrix of this algebraic equation is also ABD. By solving this algebra equation, we can get the simulation result. Mode details about applying space-oriented OSC are shown in Appendix A.2.

More details about deriving the degree of freedom, visualization of the ABD coefficient matrix, and detailed technique of applying time-oriented OSC are shown in Appendix A.1.

**Space-oriented OSC** At one time frame, the state of the domain $\Omega$ can be described by

$$
\begin{align*}
B(u_x(x)) &= g_x(x), x \in \partial \Omega \\
(u_x(x)) &= f_x(x), x \in \Omega
\end{align*}
$$

For simplicity and without loss of generality, we consider the unit domain $[0, 1]^D = \Omega$. Similar with time-oriented OSC split strategy, for each dimension, we split the domain to $N$ partitions $\pi_i : 0 = p^0_i < p^1_i < \cdots < p^N_i = 1, i = 1, \cdots, D$. Here the partitions can be isometric or non-isometric. Our target is to find one polynomial under order $r$ on each partition for every dimension and make these $N \times D$ polynomials $C^1$ continuous in the domain. For example, we can choose piecewise Hermite cubics as the base. The simulation result is the linear combination of each dimension’s basis, with the degree of freedom $N^D(r - 1)^D$. For each dimension, we select $r - 1$ collocation points in each partition by Gauss—Legendre quadrature rule (De Boor & De Boor, 1978). Then we have in total $N^D(r - 1)^D$ collocation points in the space.

We evaluate GRAPHSPLINE-NETS on three dynamic systems: heat equation, mesh-free compressible fluids, and deformable mesh–based cloth simulation.

**4. Experiments**

We evaluate GRAPHSPLINE-NETS on three dynamic systems: heat equation, mesh–free compressible fluids, and deformable mesh–based cloth simulation.

**4.1. Experimental Domains**

**Heat Equation** These datasets are generated by FEniCS (Logg et al. 2012). We set the space domain to $\Omega = [0, 1] \times [0, 1]$ and time domain to $[0, 1]$ with a total of 500 time steps. Each time step corresponds to $\Delta t = 0.1$ s. At each time step, the dataset has a fixed mesh with 49 nodes and 248 edges. This dataset is split into batches of 500 : 100 : 100 of train, validation, and test set. We initialize these datasets by fixing four boundaries to 0 and setting one or multiple locations within the domain with the initial temperature.

\(1\) Widely used quadrature rule that can keep simulation results A-stable (Iserles, 2009)
Efficient Continuous Spatio-Temporal Simulation with Graph Spline Networks

4.2. Model Training

Heat Equation In the heat equation simulation, the number of nodes and connections do not change along the process and we hence keep the graph structure fixed. The input of the graph model is the state value, i.e. the temperature, of each mesh node at the initial time frame and its output is the mesh state at the subsequent. We train a message passing neural network model by minimizing the squared differences between the target next state and the model prediction (further details are available in Appendix B.2).

4.3. Results

OSC and Interpolation Methods We show the effectiveness of orthogonal by testing spline collocation methods with different basis (orthogonal and non-orthogonal) on 2D heat equation simulation and the results is shown in Figure 7 (Left). With orthogonal basis, the collocation method can get more accurate results with 40% running time. Moreover, since creating the OSC coefficient matrix involves independent operations, we can use multiprocessing to efficiently create it. Multiprocessing also helps for solving the generated algebraic equations with almost block diagonal matrices. We also further our contributions by implementing the algorithm on GPU to increase its efficiency. We show the results of parallelization and GPU implementation in Figure 7 (Right).

We also compare the OSC with several widely used interpolation methods: linear, bilinear, 1-D cubic, and 2-D cubic interpolation methods. Note that OSC methods can have customized orders, e.g. the same with cubic interpolation (order 3) or higher (order > 4). Higher-order polynomials can better describe non-linear problems. These methods are applied to four linear and non-linear problems. Results of running time and accuracy comparisons are shown in Figure 4. OSC methods have the lowest error while incurring low computational requirements among the compared methods. The number of sample points affects the performance: however, it is (Toraichi et al., 1987) demonstrated that OSC has lower calculation complexity $O(n^3 \log n)$ than the cubic interpolation method $O(n^3/4)$, where $n$ is the number of sample points. Further insights are shown in Appendix B.1.

Heat Equation Visualization of simulation results for central initialized heat equation is shown in Figure 6. We can see the GEN has discrete prediction results, while our approach can produce continuous simulations with errors lower than $10^{-7}$ at any point within the domain. Finally, we discuss the accuracy and running time with rollout steps for two approaches on heat equation by time region and space region.

With the help of time-oriented OSC, GRAPHSPLINENETS have a nearly 90% running time reduction while keeping similar or lower MSELoss compared with GEN as shown in Table 1. Moreover, Figure 9 shows the comparison of two models’ performance with the increase of collocation point and rollout step. With the increase of collocation point number, our model can capture more information than GEN. Meanwhile, since the GRAPHSPLINENETS has a
Efficient Continuous Spatio-Temporal Simulation with Graph Spline Networks

Figure 5. Comparing dam breaking simulation results (down) with the ground truth (up) at \( t = 1, 2, \cdots, 5 \) time frames. The colors changing from blue to red show the increase of particle position moving gradient. \textsc{GraphSplineNets} takes the initial state as the input and predicts 40 rollouts, where one rollout maps to 15 frames of ground truth (\( \Delta t = 0.15 \text{s} \)). Then, the time-oriented OSC is applied to help simulate values between rollout steps, including 5 frames shown here.

Figure 6. Heat Equation Simulation with central initialization (up) and four corners dynamic changing (down). GNN has discrete prediction result at each step, while with space-oriented OSC, our \textsc{GraphSplineNets} can have a continuous prediction in the domain.

5. Conclusion

We introduce \textsc{GraphSplineNets}, a novel method that can be integrated as a post-processing scheme into several learnable GNN modules for improving the solution of a variety of physical processes in terms of both accuracy and inference time. Our approach integrates the theory of Orthogonal Spline Collocation methods to achieve space and time continuous simulations without heavy burdens on the computational side. We demonstrate how \textsc{GraphSplineNets} are robust in predicting complex, high-dimensional processes characterized by several different PDE processes, such as the ones arising directly from differential equations.
Figure 8. Flag simulation results. [Left]: root mean squared error (RMSE) propagation: our modules predict state updates every 5 (Δt = 0.1 s - Ours−5) and 10 (Δt = 0.2 s, Ours−10) steps with time–oriented collocation can produce stable rollouts and perform competitively with the baseline model in the long run. [Middle] 1–second predictions: around this rollout time, decoherence takes effect due to the chaotic nature of the flag simulation: our method collects lower error compared to the baseline GNN. [Right]: CPU and GPU rollout time comparisons accounting for graph inference and the OSC method demonstrate noticeable speedups in terms of solution inference time.

Table 1. Inference timings and rollout errors of our method and GEN.

<table>
<thead>
<tr>
<th>Model</th>
<th>Running Time (10 × 10)</th>
<th>MSE (10 × 10)</th>
</tr>
</thead>
<tbody>
<tr>
<td>GEN</td>
<td>1.70 × 10⁻² 5.81 × 10⁻² 6.55 × 10⁻² 6.71 × 10⁻² 7.00 × 10⁻² 7.35 × 10⁻² 2.01 × 10⁻⁴ 1.28 × 10⁻⁴ 8.29 × 10⁻⁵ 7.14 × 10⁻⁵</td>
<td>2.14 × 10⁻⁶ 1.32 × 10⁻⁶</td>
</tr>
<tr>
<td>Ours</td>
<td>6.50 × 10⁻⁴ 1.99 × 10⁻³ 2.04 × 10⁻³ 5.88 × 10⁻³ 6.77 × 10⁻³ 8.52 × 10⁻³ 5.02 × 10⁻⁴ 2.78 × 10⁻⁴ 1.02 × 10⁻⁵ 2.13 × 10⁻⁵ 1.22 × 10⁻⁶ 9.88 × 10⁻⁷</td>
<td></td>
</tr>
</tbody>
</table>

Figure 9. Comparison between our GRAphSPLINE:NETS and GEN on time region. [Left] figure compares error and time of two models with the number of time–oriented OSC collocation point changes from 4 × 4 to 12 × 12. [Right] figure shows the error increasing with rollout steps of two models with 10 (time) and 10 × 10 (space) collocation points.

References


Efficient Continuous Spatio-Temporal Simulation with Graph Spline Networks


Efficient Continuous Spatio-Temporal Simulation with Graph Spline Networks


Supplementary Material: Efficient Continuous Spatio-Temporal Simulation with Graph Spline Networks

A. Supplementary Material on Orthogonal Spline Collocation

We further illustrate the OSC method by providing numerical examples in this section.

A.1. 1-D OSC Example

For simplicity and without loss of generality, we consider the function domain as unit domain [0, 1] and we set $N = 3, r = 2$, which means we will use a three-order three-piece function to simulate the 1-D ODE problem as shown in Equation 2. We firstly choose the partition points as $x_i, i = 0, \cdots, 3, x_0 = 0, x_3 = 1$. The number of partition points is $N + 1 = 4$. Distance between partition points can be fixed or not fixed. Then, based on Gauss-Legendre quadrature rule, we choose collocation points. The number of collocation point within one partition is $r = 3$. For simplicity and without loss of generality, we consider the function domain as unit domain [0, 1].

After getting partition points and collocation points, we will construct the simulator. Here we have three partitions; in each partition, we assign a third-order polynomial

$$a_{0,0} + a_{0,1}x + a_{0,2}x^2, x \in [x_0, x_1] \quad \text{(S1a)}$$

$$a_{1,0} + a_{1,1}x + a_{1,2}x^2, x \in [x_1, x_2] \quad \text{(S1b)}$$

$$a_{2,0} + a_{2,1}x + a_{2,2}x^2, x \in [x_2, x_3] \quad \text{(S1c)}$$

Notice that these three polynomials should be $C^1$ continuous at the connecting points, i.e. partition points within the domain. For example, Equation S1a and Equation S1b should be continuous at $x_1$, then we can get two equations

$$\begin{align*}
    a_{0,0} + a_{0,1}x_1 + a_{0,2}x_1^2 &= a_{1,0} + a_{1,1}x_1 + a_{1,2}x_1^2 \\
    0 + a_{0,1} + 2a_{0,2}x_1 &= 0 + a_{1,1} + 2a_{1,2}x_1
\end{align*} \quad \text{(S2)}$$

For boundary condition

$$\ddot{u}(x) = \begin{cases} b_1, & x = x_0 \\ b_2, & x = x_3 \end{cases} \quad \text{(S3)}$$

we can also get two equations

$$\begin{align*}
    a_{0,0} + 0 + 0 &= b_1 \\
    a_{1,0} + a_{1,1} + a_{1,2} &= b_2
\end{align*} \quad \text{(S4)}$$

And sum up the equations we got so far. Firstly, our undefined polynomials have $N \times (r + 1) = 9$ parameters. The $C^1$ continuous condition will create $(N - 1) \times 2 = 4$ equations and the boundary condition will create 2 equations. Then we have $N \times (r - 1)$ collocation points. For each collocation point, we substitute it to polynomials to get an equation. For example, if the ODE in Equation 2 is

$$\ddot{u}(x) + \dot{u}'(x) = f(x), x \in [0, 1] \quad \text{(S5)}$$

By substituting collocation point $\xi_0$ into the equation, we can get

$$\ddot{u}(\xi_0) + \dot{u}'(\xi_0) = f(\xi_0) \quad \implies a_{0,0} + a_{0,1}\xi_0 + a_{0,2}\xi_0^2 + a_{0,1} + 2a_{0,2}\xi_0 = f(\xi_0) \quad \text{(S6)}$$

$$\implies a_{0,0} + a_{0,1}(\xi_0 + 1) + a_{0,2}(\xi_0^2 + 2\xi_0) = f(\xi_0)$$
Now we can know that the number of equations can meet with the degree of freedom of polynomials

\[
(r + 1) \times N = \frac{2}{2} + (N - 1) \times 2 + N \times (r - 1)
\]

(S7)

In this example, generated equations will be constructed to an algebra problem \( \mathbf{A} \mathbf{a} = \mathbf{f} \) where the weight matrix is an almost g matrix as shown in Figure S1.

**Figure S1.** Visualization of an almost block diagonal matrix. This kind of matrix can be cheaply solved by exploiting its properties.

\[
\mathbf{A} = \begin{bmatrix}
1 & 0 & \xi_0 & \xi_0^2 & 0 & 0 & 0 & 0 \\
1 + \xi_0 & x_1 & x_1^2 & -1 & -x_1 & -x_1^2 \\
0 & 1 & 2x_1 & 0 & -1 & -2x_1 \\
0 & 0 & 0 & 1 & \xi_1 & \xi_1^2 \\
0 & 0 & 0 & 1 & 1 & 1 \\
\end{bmatrix},
\]

(S8a)

\[
\mathbf{a} = \begin{bmatrix}
a_{0,0} \\
a_{0,1} \\
a_{0,2} \\
a_{1,0} \\
a_{1,1} \\
a_{1,2} \\
\end{bmatrix}, \quad \mathbf{f} = \begin{bmatrix}
b_1 \\
f(\xi_0) \\
0 \\
f(\xi_1) \\
0 \\
\end{bmatrix}.
\]

(S8b)

To solve this problem, we can get the simulation results.

### A.2. 2-D OSC Example

For simplicity and without loss of generality, we consider the function domain as unit domain \([0, 1] \times [0, 1]\) and we set \(N_x = N_y = 2, r = 3\). Partition points and collocation points selection are similar with 1-D OSC method, we have \(N^2 \times (r - 1)^2 = 16\) collocation points in total. For simplicity, we note the partition points at two dimensions to be the same, i.e., \(x_i, i = 0, 1, 2\). Unlike the 1-D OSC method, we choose Hermit bases to describe as the simulator, which keeps \(C^1\) continuous. As a case, the base function at point \(x_1\) would be

\[
H_1(x) = f_1(x) + g_1(x)
\]

\[
f_1(x) = \begin{cases}
\frac{(x-x_0)(x_1-x)}{(x_0-x_1)^2}, & x \in (x_0, x_1] \\
\frac{(x-x_2)(x-x_1)}{(x_2-x_1)^2}, & x \in (x_1, x_2)
\end{cases}
\]

(S9)

\[
g_1(x) = \begin{cases}
\frac{(x_1-x_0)^2+2(x_1-x_0)(x-x_0)}{(x_0-x_1)^2}, & x \in (x_0, x_1] \\
\frac{(x_2-x_1)^2+2(x_2-x_1)(x-x_2)}{(x_2-x_1)^2}, & x \in (x_1, x_2)
\end{cases}
\]

We separately assign parameters to basis functions, i.e. \(H_1(x) = a_{1,i}f_1(x) + b_{1,i}g_1(x)\) for \(x\) variable in \([x_0, x_1] \times [y_{i-1}, y_i]\) partition. Then the polynomial in a partition is the multiple combinations of base functions of two dimensions. For example,
the polynomial in the partition \([x_0, x_1] \times [y_0, y_1]\) is

\[
\left[ a_{0,1} f_0(x) + b_{0,1}^x g_0(x) + a_{1,1} f_1(x) + b_{1,1}^x g_1(x) \right] \\
\times \left[ a_{0,1} f_0(y) + b_{0,1}^y g_0(y) + a_{1,1} f_1(y) + b_{1,1}^y g_1(y) \right]
\] (S10)

Now we consider the freedom degree of these polynomials. From definition, we have\(2n(r - 1)(n + 1) = 24\) parameter. Consider boundary conditions, we have \(24 - 4 \times N = 16\) parameters. The number is equal with collocation points \(N^2 \times (r - 1)^2\), which means we can get an algebra equation by substituting collocation points. Solving this equation, we can get the simulator parameters.

We can similarly multiple basis functions and set parameters to the simulation result for the higher dimension OSC method. And then select partition points and collocation points by the same strategy with 2–D OSC method. The rest algebra equation generating and solving equations parts will not be different.

### A.3. Simple Numerical Example

We set \(N = 3, r = 3\) to simulate the problem

\[
\begin{align*}
&\begin{cases}
  u + u' = \sin(2\pi x) + 2\pi \cos(2\pi x) \\
  u(0) = 0 \\
  u(1) = 0
\end{cases} \\
&u(x) = 6.2x - 0.4x^2 - 31.4x^3, x \in [0, 1/3) \\
&1.5 + 1.6x - 13.8x^2 + 9x^3, x \in [1/3, 2/3) \\
&28.5 - 100x + 108.5x^2 - 37x^3, x \in [2/3, 1]
\end{align*}
\] (S12)

Visualization of this simulation results is shown in Figure S2.

![Simulation Ground Truth Partition Points Collocation Points](image.png)

Figure S2. 1-D OSC example problem simulation result visualization, with the mean square loss less around \(5 \times 10^{-4}\) comparing with the real solution \(u(x) = \sin(2\pi x)\).

### B. Supplementary Experimental Results

#### B.1. OSC

We compared the OSC with linear, bilinear, 1–D cubic, and 2–D cubic interpolation methods on four types of problems: 1–D linear, 1–D non-linear, 2–D linear, and 2–D non-linear problems. In these experiments, we tested different simulator orders of the OSC method. For example, we set the order of the simulator to 4 for 1–D linear problem and 2 for 2–D linear problem. When the order of the simulator matches the polynomial order of the real solution, OSC can directly find the real solution. For non-linear problems, increasing the order of the simulator would be an ideal way to get lower loss. For example, we set the order of the simulator to 4 for 1–D non-linear problem and 5 for 2–D non-linear problem. Thanks to the efficient calculation of OSC, even though we use higher–order polynomials to simulate, we use less running time to get results.
We discussed the influence of collocation point numbers. By increasing the number of space–oriented OSC collocation points, we test the performance of GNN and ETS on one heat equation which has centrally initialized, and four interpolation methods on different PDEs problems. 1–D cubic interpolation method works on less dynamic boundary system. We choose 16 sample points within the domain and get the series of values along time, which is shown in Figure S5. In this experiment, we fixed the order of time–oriented OSC to 4. During training, we load the training trajectories (single frame) randomly to generate target frames nodes value (x, y, t = 0). After we get the prediction values of collocation points, we apply the space–oriented OSC to get simulation polynomials in the domain. Then we can get the same resolution predictions and the ground-truth one. We use the Adam optimizer (Kingma & Ba, 2014) to optimize the loss with a batch size of 8. GNN trained on FEniCS dataset generating, which follows the rule of unit mesh connecting. These connections would not change during training. To construct the neural network, we use 3 layers of message passing neural network; each layer contains 10 layers of message passing neural network; each layer contains 10 layers of message passing neural network; each layer contains 10 layers of message passing neural network; each layer contains 256 MPLs by simple layers. Before training, we apply independent noise N(0, 10−4) normalization to each node’s value. During training, we load the training trajectories (single frame) randomly to generate target frames nodes value (Δt = 0.01 for GNN and Δt = 0.05 for GRAPHSPLINENETS). The loss is the mean square error between the target frames’ node value and the ground-truth one. We use the Adam optimizer (Kingma & Ba, 2014) to optimize the loss with a batch size of 2.

We test the performance of GRAPHSPLINENETS and GNN on one heat equation which has centrally initialized, and four boundaries are fixed to 0 (Figure S3). The ground truth is generated with the resolution 256 × 256. GRAPHSPLINENETS is trained on 8 × 8 collocation points with a time step Δt = 0.5 s. After we get the prediction values of collocation points, we apply the space–oriented OSC to get simulation polynomials in the domain. Then we can get the same resolution predictions with ground truth. Meanwhile, we train a GNN model on all the ground truth nodes with a time step Δt = 0.1 s. We can see from the table that GRAPHSPLINENETS uses less than 10% of GNN’s running time while the loss gap is lower than 5 × 10−4. In some cases, this error gap even becomes closer.

We also test GRAPHSPLINENETS on different heat equations. Figure S4 shows one heat equation with zero initialization and dynamic boundary system. We choose 16 sample points within the domain and get the series of values along time, which is shown in Figure S5. In this experiment, we fixed the order of time–oriented OSC to 4. We can see that our model can always have an excellent simulation for different locations’ sample points. 1–D cubic interpolation method works on less dynamic points. However, for more complex points, the cubic interpolation method collects a considerable amount of error (Figure S5 (0.9, 0.9), while our approach can still stay convergent.

We discussed the influence of collocation point numbers. By increasing the number of space–oriented OSC collocation points.

### B.2. Heat Equation

We construct the graph by encoding sample points’ location and value to three features. The connection of nodes is provided by FEniCS dataset generating, which follows the rule of unit mesh connecting. These connections would not change during the training. To construct the neural network, we use 3 layers of message passing neural network; each layer contains 3 layers of message passing neural network; each layer contains 3 layers of message passing neural network; each layer contains 3 layers of message passing neural network; each layer contains 8 MPLs by simple layers. Before training, we apply independent noise N(0, 10−4) normalization to each node’s value. During training, we load the training trajectories (single frame) randomly to generate target frames nodes value (Δt = 0.01 for GNN and Δt = 0.05 for GRAPHSPLINENETS). The loss is the mean square error between the target frames’ node value and the ground-truth one. We use the Adam optimizer (Kingma & Ba, 2014) to optimize the loss with a batch size of 2.

Table 1. Error of OSC and four interpolation methods on different PDEs problems: \(u(x) = x^4 - 2x^3 + 1.16x^2 - 0.16x\) (1-D linear), \(u(x) = \sin(3\pi x)\) (1-D non-linear), \(u(x, y) = x^2y^2 - x^2y + xy\) (2-D linear), \(u(x, y) = \sin(3\pi x)\sin(3\pi y)\) (2-D non-linear).

<table>
<thead>
<tr>
<th>Model</th>
<th>1-D Linear</th>
<th>1-D Non-linear</th>
<th>2-D Linear</th>
<th>2-D Non-linear</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nearest Interpolation</td>
<td>2.3670 × 10^{-6}</td>
<td>1.7558 × 10^{-2}</td>
<td>1.9882 × 10^{-3}</td>
<td>3.8695 × 10^{-2}</td>
</tr>
<tr>
<td>Linear Interpolation</td>
<td>1.8928 × 10^{-7}</td>
<td>8.7731 × 10^{-4}</td>
<td>3.4317 × 10^{-4}</td>
<td>1.1934 × 10^{-2}</td>
</tr>
<tr>
<td>Quadratic Interpolation</td>
<td>2.6748 × 10^{-10}</td>
<td>2.8827 × 10^{-6}</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Cubic Interpolation</td>
<td>3.5232 × 10^{-12}</td>
<td>2.2654 × 10^{-7}</td>
<td>2.9117 × 10^{-4}</td>
<td>4.5441 × 10^{-3}</td>
</tr>
<tr>
<td>OSC</td>
<td>3.4153 × 10^{-31}</td>
<td>4.1948 × 10^{-8}</td>
<td>1.7239 × 10^{-32}</td>
<td>3.4462 × 10^{-5}</td>
</tr>
</tbody>
</table>

Figure S3. Heat equation with central initialized and zero boundary condition. [Left] ground truth solution of this problem. [Right] GRAPHSPLINENETS simulation results, with 16 partitions and (8 × 8) collocation points.
Table 2. Inference timings and rollout errors of our method and GNS on the Dam Breaking dataset with different collocation points for 50 rollout steps. GRAPHSPLINENETS perform competitively or even outperform the baseline model while cutting down considerably the running time.

<table>
<thead>
<tr>
<th>Model</th>
<th>Running Time</th>
<th>MSELoss</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>15</td>
<td>21</td>
</tr>
<tr>
<td>GNS</td>
<td>$3.44 \times 10^{-1}$</td>
<td>$4.71 \times 10^{-1}$</td>
</tr>
<tr>
<td>Ours</td>
<td>$4.02 \times 10^{-2}$</td>
<td>$6.55 \times 10^{-2}$</td>
</tr>
</tbody>
</table>

points, we can see the error of GRAPHSPLINENETS has a fast speed to get close to GNN (Figure S6 left) while keeping the running time short. In the case of $(14 \times 14)$ collocation points, GRAPHSPLINENETS has a loss gap lower than $3 \times 10^{-7}$ with GNN but takes only $43\%$ time of GNN. We also illustrate the error curve of our strategy based on the number of collocation sites to demonstrate its strong convergence capabilities.

B.3. Dam Breaking

We define the input ‘velocity’ as average velocity between the current and previous timesteps, calculated for three dimensions $p^k = (v^k_x - v^{k-1}_x)/\Delta t$. This property is encoded to the node feature. For each node, we find all neighbors within a connectivity radius to connect. We decide the connectivity radius by setting a maximum connectivity $k$. For each node, we calculate the average distance of its $k$ nearest neighbors, then keep this distance during training. We create the graph structure by using 3 message passing neural network layers, and each one contains 3 MLPs.

Before training, we apply independent noise $\mathcal{N}(0, 10^{-3})$ to every node’s vector. We only do one normalization before training. During the training, the graph takes several previous trajectories as input, and the output would be the velocity at this time step. Then we accumulate this output with time to get the position of this node at the next time frame. We calculate the loss by comparing the position and velocity of each node with the ground truth. Then apply Adam strategy to optimize the loss. Some results are shown in Table 2.

B.4. Flag Simulation

We retain most of the experimental settings of the FLAGSIMPLE experiment in (Pfaff et al., 2021). More specifically, we utilize the same latent vector representations of size 128 at each node and edge and the same number of message passing steps (15). We utilize a batch size of 2 instead of 1 as in (Pfaff et al., 2021), which we found to be less brittle to training for fewer epochs. We trained the model for 1M training steps with the Adam optimizer subject to an exponential learning rate
Efficient Continuous Spatio-Temporal Simulation with Graph Spline Networks

Figure S5. Value changing for 16 sample points along time of the heat equation in Figure S4, including ground truth, 1-D cubic interpolation results and our simulations. Location of sample point is marked in the left bottom corner of figures.

decay from $10^{-4}$ to $10^{-6}$. We empirically found the choice of training noise to be essential for successful model training. In particular, we used the same training noise of 1e-3 with the noise correction parameter $\gamma$ set to 0.1 in the one–step baseline GNN. For models predicting multiple steps, we experimentally found multiplying the step size ratio by the initial training noise to be successful. For instance, supposing that the baseline model predicts with a step update of 1, the model with step size 5 (i.e. $\Delta t = 0.1$ s) will be trained with noise of scale $1e^{-3} \times 5 = 5e^{-3}$. In Figure S7 we show differences between linear interpolation and OSC for a sample point in the flag simulation.

B.5. Hardware and Software

Experiments were carried out on a machine equipped with an Intel Core i9 7900X CPU with 20 threads and a NVIDIA RTX 2080 Ti graphic card with 11 GB of VRAM. Software–wise, we used FEniCS (Logg et al., 2012) for Finite Element simulations for the heat equation experiments; Taichi (Hu et al., 2019) for the dam breaking simulation, while the flag dataset was obtained with ArcSim (Narain et al., 2012). We implemented a parallelizable routine for the OSC method using the multiprocess libraries in Python and PyTorch for GPU parallelization (Paszke et al., 2019). The GRAPHSPLINE NETS code was written in Python, and PyTorch was used for deep learning while the Deep Graph Library (DGL) (Wang et al., 2020) for graph neural networks.

C. Supplementary Material on Physical Simulations and Deep Learning

C.1. Classical Simulators

Classical numerical methods for solving PDEs can be broadly divided into mesh–based and mesh–free approaches. Mesh-based methods notably include the Finite Element Method (FEM) (Zienkiewicz et al., 1977), Finite Volume Method, (Eymard et al., 2000) and deformable materials simulators (e.g., cloth) (Baraff & Witkin, 1998; Narain et al., 2012). Their mesh–free counterparts include Molecular Dynamics, (MD) (Rapaport & Rapaport, 2004), the Material Point Method (Bardenhagen & Kober, 2004) for finite particles, and the Smoothed Particle Hydrodynamics (Monaghan, 1992) which
Figure S6. Comparison between \textsc{GraphSplinesNets} and GNN in space region. [\textbf{Left}] figure compares the error and time of two models with the number of space–oriented OSC collocation points of \textsc{GraphSplinesNets} setting from $2 \times 2$ to $26 \times 26$, while the GNN model has the number of space collocation point $32 \times 32$. Note that the error of \textsc{GraphSplinesNets} is calculated by firstly applying space–oriented OSC to collocate values at GNN’s collocation points and then comparing them with the ground truth. [\textbf{Right}] shows the error decreasing with the number of space–oriented OSC collocation points increasing.

Figure S7. Trajectory in time and space of a single node from the flag simulation experiment. While linear interpolation fails to capture complex behaviors, the OSC method can model intermediate steps with guarantees of solution convergence and lower computational overheads compared to other methods such as cubic interpolation.

specializes in fluid simulation. The choice between mesh–free and mesh–based methods is usually dictated by computational efficiency. For example, while turbulent flow simulations may be represented in a particle-by-particle fashion, describing flows at sample points through a mesh may be desirable, thus noticeably reducing inference times. Similarly, domains with dynamically changing volumes, such as fluid simulation, can benefit more by mesh–free particle simulators rather than forcing an unnatural mesh description. Major downsides of classical numerical simulators are that they often require extensive knowledge about the simulated domains as well as taking considerable computational resources and time to infer solutions: deep learning has proven to be a valid and faster alternative to existing numerical solvers (Thuerey et al., 2021).

\textbf{C.2. Further Intuition on Graph Networks Simulators and OSC}

Graph neural networks have intrinsically desirable properties for physical simulation. Since they can be used on unstructured grids, they are able to represent complex systems with changing connectivity. Moreover, the graph structurally attains equivariance, while translation equivariance can also be easily achieved by considering relativity in nodes’ positions rather than their absolute values. Like Molecular Dynamics, our module can construct edges based on each particle neighborhood to capture message passing locality (Anonymous, 2022). Moreover, similarly to how FEM works, we can consider fixed neighbor connections in mesh space, thus constraining our optimization to a specific geometry. However, unlike the classical FEM and MeshGraphNets, \textsc{GraphSplinesNets} can infer the space continuously without the need to refine the mesh structure. Moreover, given collocation points in time, our model can also obtain time–continuous simulation without the need to constrain the rollouts to an iterative solver.