Physics-Informed Neural Networks for Mesh Deformation with Exact Boundary Enforcement

A. Aygun* [1], R. Maulik[2] and A. Karakus[1]

[1]*Department of Mechanical Engineering, Middle East Technical University, Ankara, Turkey 06800*
[2]*Mathematics and Computer Science Division, Argonne National Laboratory, Lemont, IL, 60439, USA*

In this work, we have applied physics-informed neural networks (PINN) for solving mesh deformation problems. We used the collocation PINN method to capture the new positions of the vertex nodes while preserving the connectivity information. We use linear elasticity equations for mesh deformation. To prevent vertex collisions or edge overlap, the mesh movement in this work is conducted in steps with relatively small movements. For moving boundary problems, the exact position of the boundary is essential for having an accurate solution. However, PINNs are frequently unable to satisfy Dirichlet boundary conditions exactly. To overcome this issue, we have used hard boundary condition enforcement to automatically satisfy Dirichlet boundary conditions. Specifically, we first trained a PINN with soft boundary conditions to obtain a particular solution. Then, this solution was tuned with exact boundary positions and a proper distance function by using a new PINN considering only the equation residual. To assess the accuracy of our approach, we used the classical translation and rotation tests and compared them with a proper mesh quality metric considering the change in the element area and shape. The results show the accuracy of this approach is comparable with that of finite element solutions. We also solved different moving boundary problems, resembling commonly used fluid-structure interaction problems. This work provides insight into using PINN for mesh-deformation problems without needing a discretization scheme with reasonable accuracy.

Keywords: physics-informed neural networks, mesh deformation, exact boundary enforcement, linear elasticity

1 Introduction

Dynamic grids in numerical fluid flow simulations generally arise in many applications, such as airfoil movement [1, 2], blood flow [3], parachute mechanics [4, 5], and free surface flow problems [6]. These and other fluid-structure interaction (FSI) problems need to move the computational grid with moving boundaries. The naive choice is to regenerate the mesh every time the boundary moves. Regenerating the mesh for a complex geometry results in a need for an automatic mesh generator [7]. This approach alters the grid connectivity and, therefore, brings up a need to project the solution to the new mesh. This introduces new projection errors each time the mesh is updated. Moreover, the cost of calling a new mesh generation algorithm can be overwhelming, especially for 3D problems [8].

Specific mesh moving techniques can overcome the drawbacks of remeshing for moving boundary problems. These methods try to update the position of the nodes of the original mesh under some prescribed laws without changing the grid connectivity. Farhat et al. introduced a spring analogy, where they fictitiously attach a torsional spring to the nodes of the mesh [9]. The system has fictitious mass, damping, and stiffness matrices, and the forcing is the displacement of the moving boundaries. This approach prevents vertex collisions as well as penetrating grid edges. In [8], the authors used a

---

*Corresponding author. E-mail address: atakana@metu.edu.tr

linear elastic equation to represent the fluid domain as an elastically deformable body and introduced a parallel finite element strategy. Using the same elasticity formulation, Stein et al. [10] solved the equation using a Jacobian-based stiffening. They introduced an additional stiffening power as a function of transformation Jacobian in the finite element formulation. This addition allowed them to stiffen the smaller elements more than the larger ones, resulting in improved mesh quality near the moving surfaces. Takizawa et al. [11], introduced a method based on the fiber-reinforced hyperelasticity model. They introduced fibers in different directions according to the motion, which allows the model to reduce the distortion of a mesh element. The moving mesh problem can be solved using the Laplacian or biharmonic equations [12–14]. Although using the biharmonic operator introduces extra computational complexity compared to the Laplacian equation systems, it can give the extra ability to control the normal mesh spacing [14].

Apart from conventional numerical methods, machine learning methods are also used to solve partial differential equations. Deep neural networks were first used by Lee and Kang [15], and Lagaris et al [16] to predict the solution of a partial differential equation (PDE). Raissi et al. [17] introduced the concept of physics-informed neural networks (PINN) to solve PDEs without any given data. This approach gives information about the physical laws to the neural network. Using the information on the boundary and initial conditions, neural networks can predict the solution of a PDE. The PINN formulation has received great attention and has been studied in wide content. There are numerous extensions of PINN to improve the methodology. Several domain decomposition models are designed to improve the accuracy and allow parallelization [18–20]. Bayesian PINNs are proposed to tackle the problems involving solving PDEs where noisy data is available [21] and where uncertainty quantification is important. Applications of PINN cover the solutions of conservation laws [22], fractional and stochastic differential equations [23, 24], solution of Navier-Stokes equations [25, 26], Euler equations [27], heat transfer problems [28, 29], Boltzmann equation with Bhatnagar-Gross-Krook collision model [30], Allen-Cahn and Cahn-Hilliard equations [31, 32], free boundary and Stefan problems [33] and many more.

Despite the success of PINN across a range of different problems, it can face difficulties when solving multiscale and multiphysics problems [34], especially for dynamical systems with chaotic or turbulent behavior [35]. The fully connected networks face difficulties in learning high-frequency functions. This phenomenon is named spectral bias [36, 37]. The high-frequency behavior in the objective function results in sharp gradients. Therefore, PINN models can have difficulties while penalizing the residual loss. Although there are several approaches to tackle these problems and improve the training capabilities of PINN, the classical PINN method shows better performance to accurately solve the PDEs that govern the mesh deformation. Therefore our research focuses on using PINNs in the application of these problems.

The main objective of this paper is to show the applicability of physics-informed neural networks for moving mesh problems. The PINN approach can produce satisfactory solutions for the movement of boundaries without needing a discretization scheme. However, using the original PINN formulation for mesh moving problems can have difficulties with the static and moving boundaries. PINN minimizes the loss at the boundaries, without imposing boundary conditions exactly. To overcome this problem, we used exact boundary enforcement. After obtaining a particular solution that weakly satisfies the boundary conditions, the prediction is corrected by training another PINN. To the best of our knowledge, using PINNs on mesh movement problems with exact boundary enforcement is not studied in detail in the literature.

The remainder of this paper is organized as follows. First, basic information is given about physics-informed neural networks. This chapter is enhanced with the methodology of automatically satisfying boundary conditions using exact boundary enforcement. Most common mesh movement techniques are presented in the next chapter alongside the mesh quality metric used for comparing different methods. The results are presented with classical translation and rotation tests followed by examples resembling

commonly used moving boundary problems.

2 Physics-Informed Neural Networks

A basic, fully connected deep neural network architecture can be used to solve differential equations [38]. Given an input vector $\mathbf{x} \in \mathbb{R}^d$, a single layer neural network gives an output $\hat{\mathbf{u}}$ by the following form:

$$\hat{\mathbf{u}} = \sigma(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1)\mathbf{W}_2 + \mathbf{b}_2, \tag{1}$$

where $\mathbf{W}$ are the weight matrices and $\mathbf{b}$ are the bias vectors. $\sigma(\cdot)$ is a nonlinear function known as the activation function. In general, Sigmoid, hyperbolic tangent, and rectified linear unit (ReLU) are popular choices for the activation function. The hyperparameters $\theta = [\mathbf{W}, \mathbf{b}]$ are estimated by the following optimization problem

$$\theta^* = \arg\min_{\theta} J(\theta; \mathbf{x}). \tag{2}$$

Here, J is the objective function to be minimized. In this work, this function is defined as the mean squared error of the prediction. This minimization problem in Equation 2 can be solved by using first-order stochastic gradient descent (SGD) algorithms [39]. In each iteration, the hyperparameters are updated in such a way,

$$\theta^{i+1} = \theta^i - \eta^i \nabla_\theta J(\theta; \mathbf{x}), \tag{3}$$

where i being the current iteration and η is the learning rate. The gradient of the loss function, $\nabla_\theta J(\theta; \mathbf{x})$, is calculated by backpropagation [40].

For the physics-informed a neural network, we consider the general form of partial differential equations:

$$\mathbf{u}_t + \mathcal{N}[u] = 0, \qquad\qquad \mathbf{x} \in \Omega,\ t \in [0, T] \tag{4a}$$
$$\mathbf{u}(\mathbf{x}, 0) = f(\mathbf{x}), \qquad\qquad \mathbf{x} \in \Omega \tag{4b}$$
$$\mathbf{u}(\mathbf{x}, t) = g(\mathbf{x}, t), \qquad\qquad \mathbf{x} \in \partial\Omega,\ t \in [0, T] \tag{4c}$$

where $\mathcal{N}$ is a generalized differential operator that can be linear or nonlinear, $\mathbf{x} \in \mathbb{R}^d$ and $t \in [0, T]$ are the spatial and temporal coordinates. Ω and $\partial\Omega$ represent the computational domain and the boundary respectively. $\mathbf{u}(\mathbf{x}, t)$ is the general solution of the PDE with $f(\mathbf{x})$ is the initial condition and $g(\mathbf{x}, t)$ is the boundary condition. The hidden solution, $\mathbf{u}(\mathbf{x}, t)$, can be approximated under the PINN framework proposed by Raissi et al. [17], by a feedforward neural network $\hat{\mathbf{u}}(\mathbf{x}, t; \theta)$ with parameters θ. For the supervised training the only labeled data comes from the boundary/initial points. Inside the domain, the loss is determined by the PDE residual. By utilizing automatic differentiation (AD) [41], PINNs can differentiate the network output w.r.t the input layer. AD applies the chain rule repeatedly to the elementary functions and arithmetic operations to achieve the derivative of the overall composition. AD is well implemented in popular deep learning frameworks such as TensorFlow [42] and PyTorch [43].

In classical PINN implementations, the loss term is a composite term including supervised data loss on the boundaries and initial points and the PDE loss. The total loss term can be written such that,

$$\mathcal{L} = w_R\mathcal{L}_R + w_{BC}\mathcal{L}_{BC} + w_{IC}\mathcal{L}_{IC}. \tag{5}$$

Here the terms represent the boundary loss $\mathcal{L}_{BC}$, the initial condition loss $\mathcal{L}_{IC}$, and the PDE residual loss $\mathcal{L}_R$. The w terms are specific weights of each loss term that can be user-specified or tuned manually

or automatically [37, 44]. Each loss term can be written as,

$$\mathcal{L}_R = \frac{1}{N_R} \sum_{i=1}^{N_R} |\mathbf{u}_t + \mathcal{N}[\mathbf{u}(\mathbf{x}^i, t^i)]|^2 \tag{6a}$$

$$\mathcal{L}_{BC} = \frac{1}{N_{BC}} \sum_{i=1}^{N_{BC}} |\mathbf{u}(\mathbf{x}^i, t^i) - g(\mathbf{x}^i, t^i)|^2 \tag{6b}$$

$$\mathcal{L}_{IC} = \frac{1}{N_{IC}} \sum_{i=1}^{I_{BC}} |\mathbf{u}(\mathbf{x}^i, 0) - f(\mathbf{x}^i)|^2. \tag{6c}$$

Here N_R, N_{BC}, and N_{IC} are the total number of points used for calculating the mean squared error used here as the loss function. The schematic of a classical PINN can be seen in the left part of Figure 1.
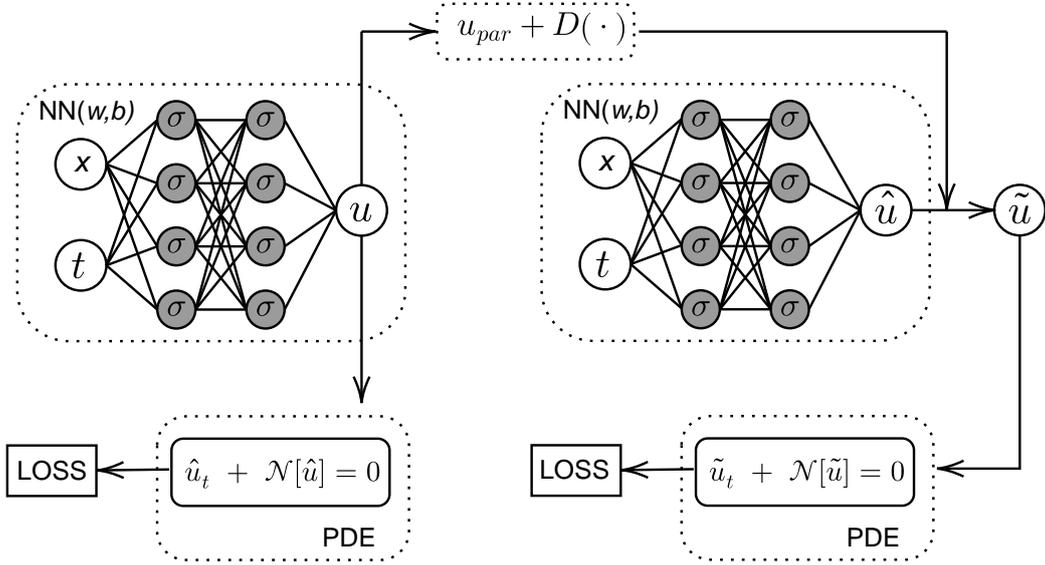

Figure 1: Schematic of PINN approach with exact boundary enforcement. The first PINN on the left shows the original formulation with weakly enforced Dirichlet boundary conditions. The second network uses the particular solution with exact boundary enforcement to satisfy Dirichlet boundaries exactly

The hyperparameters $\theta = [\mathbf{W}, \mathbf{b}]$ can be optimized by a chosen optimization algorithm to find the minimum total loss defined in Equation 5. As mentioned above, stochastic gradient descent algorithms are commonly used in neural network implementations [39]. This method aims to find new parameters θ in the opposite direction of the gradient of the objective function. The gradient of the loss function w.r.t. hyperparameters is calculated by backpropagation. In this work, we used the ADAM algorithm [45] as the SGD optimizer.

2.1 Exact Boundary Enforcement

The optimization algorithm used in PINN tries to minimize the physics-based loss, $\mathcal{L}_R$. Using proper boundary and initial conditions can regularize the physics loss in deep neural networks. This classical PINN boundary condition implementation in Equation 6 is named soft boundary enforcement [46]. In this approach, the boundary prediction is minimized in the composite loss function. Although the SGD

algorithms can minimize these loss functions, they do not satisfy the boundary values exactly. However, some PDE applications, such as mesh movement, need exact boundary values. For this purpose, we apply exact boundary enforcement. Sun et al. [46] used this boundary condition enforcement to exactly satisfy the velocity and pressure values on the boundaries of internal flow cases with Navier-Stokes equations. Sukumar and Srivastava [47] introduced geometry-aware trial functions. They multiply the neural network output with these functions and use its generalization to exactly satisfy boundary conditions on complex geometries. In this work, we use this idea with multiple physics-informed neural networks to exactly satisfy Dirichlet boundary conditions. First, we trained a PINN with soft boundaries. For the mesh movement problem, the displacement vector $\mathbf{u} = [X, Y]^T$ will give the new coordinates of the nodes from the first neural network prediction. This solution is then changed on the boundaries with the exact values. This new solution is the particular solution of our approach. Then, a new PINN is trained with an output $\hat{\mathbf{u}}(\mathbf{x}; \theta)$. This output is modified with the following equation.

$$\tilde{\mathbf{u}}(\mathbf{x}; \theta) = \mathbf{u}_{par}(\mathbf{x}) + D(\mathbf{x})\hat{\mathbf{u}}(\mathbf{x}; \theta). \tag{7}$$

Here, $\mathbf{u}_{par}$ is a particular solution that is a globally defined smooth function that only satisfies the boundary conditions. Any smooth function can be used for the particular solution such as radial basis functions (RBF) or linear functions [46]. In this work, we use the classical PINN predictions with the soft boundary condition implementation as the particular solution. D is a specified distance function from the boundary. Equation 7 states that on the boundaries $D(\mathbf{x}) = 0$, the particular solution satisfies the exact boundary values, $\mathbf{u} = g$ on $\partial\Omega$. For a general approach, we used the shortest distance between the residual points and the boundaries. Since the geometric domains used in this paper are not too complex, this approach is not very time consuming. For complex geometries, approximate distance functions using R-functions [47] or pre-trained deep neural networks [48] can be used. This modified output contributes to the physics loss of the new PINN. In this network, the objective function is only consisting of the PDE residual $\mathcal{L}_R$ and trained with the same PDE. This approach allows us to exactly satisfy the Dirichlet boundary conditions using PINN.

3 Mesh Movement

Mesh movement strategies to deform the mesh with a moving boundary generally can be performed by solving a PDE or using an interpolation scheme [49]. All of these techniques have the goal to provide a displacement of the moving boundary and propagate this movement into the domain. Methods with a PDE solution, generally model the mesh movement as a physical process which can be solved using numerical methods. One of the popular versions includes modeling the domain with torsional springs that prevent the vertices to collide [9]. In a similar manner, this movement can be modeled with an elastic [8, 10] and hyperelastic [11] analogy, where the computational domain is simulated as an elastic body. Nonlinear elasticity equations with neo-Hookean models can be used in the same way as the elastic equations [50]. Other techniques include mesh deformation as a diffusive system modeled with the Laplacian or biharmonic equations [14]. All these PDEs can be solved using traditional numerical methods such as FEM.

Interpolation schemes consider the mesh movement as a problem of interpolation from the boundaries to the domain. These schemes use interpolation on scattered data and generally do not need connectivity information. Using radial basis functions (RBF) is one of the common methods. In [51], de Boer et al. use RBF interpolation on unstructured grids to estimate the movement. The equation system only involves the boundary nodes and displacement of the whole mesh is modeled. Extending this method, in [52], the authors use data reduction algorithms using a coarse subset of the surface mesh. With greedy algorithms, this approach is effective, especially for mesh motion problems with smooth surface deformations.

In this work, we used one of the common PDEs for mesh movement. The mesh motion is calculated by using the linear elasticity equation from structural mechanics. The coordinates of the nodes will be defined as $\mathbf{u}$, the computational domain is referred to as Ω, and the boundaries are $\partial\Omega$. Boundaries also include the moving objects inside the meshes. The new coordinates of the moving and stationary boundaries are given as the Dirichlet boundary condition. The movement of an object inside the mesh deforms the computational domain which is modeled as an elastic body. The new coordinates can be found by the following linear elasticity equation:

$$\nabla \cdot \boldsymbol{\sigma}(\mathbf{u}) = 0 \quad \text{in } \Omega \tag{8a}$$

$$\mathbf{u} = \mathbf{u}_b \quad \text{on } \partial\Omega. \tag{8b}$$

Here $\boldsymbol{\sigma}$ is the Cauchy stress tensor. It is related to the strain tensor $\boldsymbol{\epsilon} = (\nabla\mathbf{u} + \nabla\mathbf{u}^T)/2$. The stress tensor can be written in a way by Hooke's law:

$$\boldsymbol{\sigma} = \lambda \text{tr}(\boldsymbol{\epsilon})\mathbf{I} + 2\mu\boldsymbol{\epsilon}. \tag{9}$$

The Lamé parameters λ and μ are structural parameters coming from the elastic modulus E and Poisson's ratio ν. Since the mesh domain is not a real elastic body, the exact values for these parameters are not known. A value between 0.3 and 0.45 is recommended for Poisson's ratio since a high value can lead to distorted elements, and a lower value can reduce the resistance [50].

To be able to compare the effectiveness of different mesh movement techniques after a deformation, we use a mesh quality metric based on [10]. In these metrics, the area and shape changes are considered by checking the element area and the aspect ratio. Both metric uses the initial mesh elements as reference elements and measures the change according to them. The element area change f_A^e and shape change f_{AR}^e is defined as :

$$f_A^e = \left| \log\left(\frac{A^e}{A_o^e}\right) / \log(2.0) \right|, \tag{10a}$$

$$f_{AR}^e = \left| \log\left(\frac{AR^e}{AR_o^e}\right) / \log(2.0) \right|. \tag{10b}$$

Here, the superscript e represents the specific element, and the subscript o is the initial mesh element before the deformation occurs. AR^e is the element aspect ratio defined in [10] as:

$$AR^e = \frac{(l_{max}^e)^2}{A^e}. \tag{11}$$

Here, l_{max}^e is the maximum edge length for the specific element. For comparison of different techniques, we use the global area and shape changes by considering the maximum values of element area and shape changes, respectively.

4 Results

The movement of dynamic meshes with PINN is presented with several different test cases. First, a deformed square is presented where we squeeze the domain from the top and bottom. Then, the basic translation and rotation tests are performed and the solutions of the PINN approach are compared with the finite element solutions. Lastly, the movement of a flexible beam is presented where one end of the beam is fixed. For all the problems, initial meshes are generated using the Gmsh mesh generator [53]. We used TensorFlow to construct our PINN framework with Adam optimizer as the gradient descent algorithm. We initialized all the neural networks using the Glorot scheme and used 7 hidden layers with 50 units. The classical neural networks are trained for 40000 iterations, and the networks with exact boundary enforcement are trained for 5000 iterations. The learning rate is 10^{-3} with a decay rate of 0.9. The Lamé parameters are selected as $\mu = 0.35$ and $\lambda = 1$ as recommended [50].

4.1 Deformed Square

In this test case, a square domain is deformed from its boundaries. The square domain is x, $y \in [0,1] \times [0,1]$ and the unstructured mesh consists of 2744 triangular elements. The initial mesh can be seen in Figure 2. We want to find a deformed mesh where the position of the top boundary becomes $\hat{y} = y - 0.25\sin(\pi x)$. On the top surface, we implement this condition as a Dirichlet boundary condition as well as $\hat{x} = x$. All the other boundaries have the same Dirichlet boundary condition as $\hat{y} = y$, and $\hat{x} = x$. The deformed mesh can be seen in Figure 2. The figure in the middle shows the results obtained by only using classical PINN. This shows the boundaries, especially the corners, are not in the exact position and are deformed in an undesired way. The figure on the right shows the solution after exact boundary enforcement. The boundary values are corrected with the exact positions with the proposed approach. The L_2 error on the boundary nodes is calculated as 0.031. For this test case, we increased the specific weight of the boundary loss of the composite loss function in Equation 5. Since the deformation of the boundary is higher than the deformation of the computational domain, the boundary weight is increased. The weight ratio of the boundary loss and the residual loss is set to 25 to capture the boundary values more precisely.
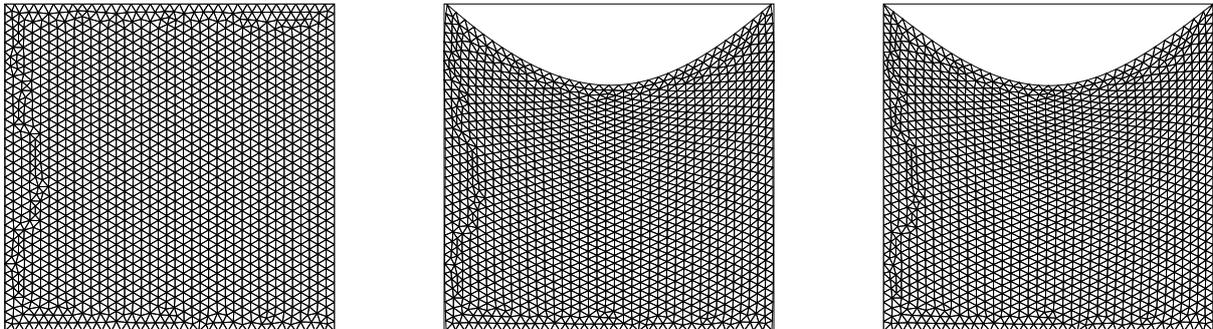

Figure 2: Initial and deformed meshes of the deformed square case with its deformed top boundary. The unstructured mesh consists of 2744 triangular elements The first deformed figure shows the solution with classical PINN. The last figure represents the solution with exact boundary enforcement.

The mesh quality measure of the deformed mesh based on the element area and shape changes can be seen in 3. The top surface is deformed according to a sinusoidal function. The elements near the deformed boundary have the most change in size and shape as expected. Especially in the middle where the deformation is the largest, the elements are squeezed and get smaller. In the corners where the element vertices have two boundary conditions in each direction, the element area change is not significantly large. However, the shape of the corner elements changes more than the other elements on the boundary. These elements are bounded by the two boundaries and therefore the aspect ratios get larger. The deformation of the inner elements is relatively low, especially near the bottom boundary. The mesh deformation metrics get lower as the elements' position moves away from the deformed boundary. The global area and shape change metrics are calculated as $|f_A^\infty| = 0.744$, $|f_{AR}^\infty| = 1.264$, respectively.

To see the capabilities of our approach we further deform the bottom boundary with its coordinates $\hat{y} = y + 0.25\sin(\pi x)$. The Dirichlet boundary conditions on the stationary boundaries are the same as the other, $\hat{x} = x$, $\hat{y} = y$. The same specific weight ratio for the loss function of the PINN formulation is used. The deformed configuration can be seen in Figure 4. The figure in the middle is the solution with the classical PINN approach. The vertices on the boundaries are not in exact positions. Especially
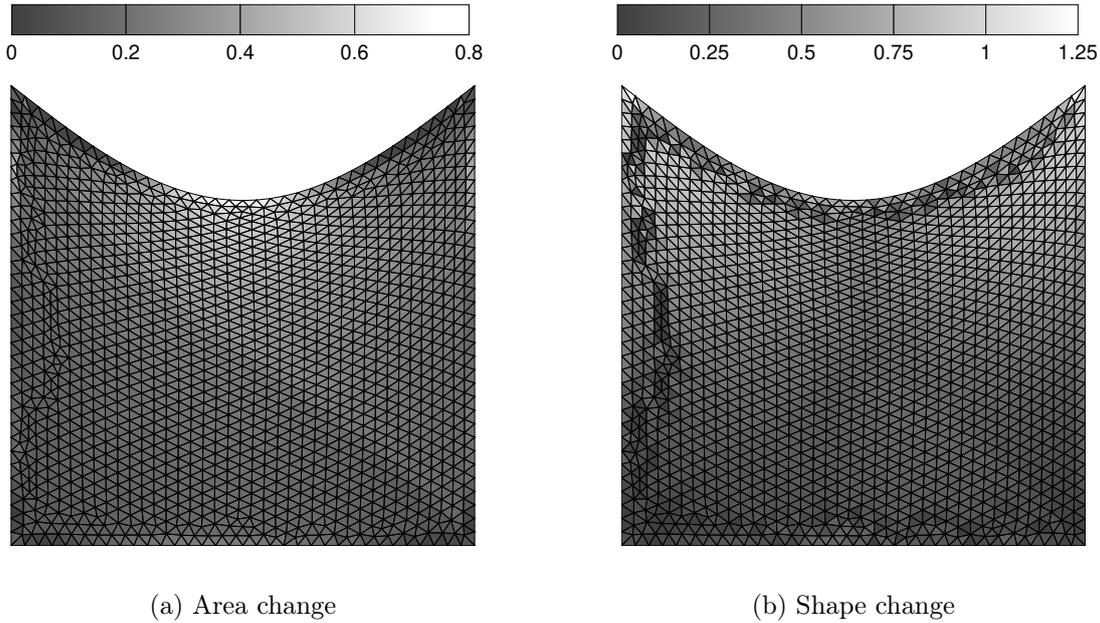
(a) Area change

(b) Shape change

Figure 3: Element quality metrics of the square with deformed top boundary. The figure on the left shows the element area change and the figure on the right shows the element shape change with respect to the initial mesh elements.

on the corners, the classical PINN solution has difficulty satisfying the positions. The L_2 error of the boundary positions is calculated as 0.076 for this case.

The elementwise quality measures of this case can be seen in Figure 5. the elements on the top and the bottom boundaries are deformed the most, the same as in the previous case. The elements in the middle collapsed more than the case before. The global area and shape change values are $|f_A^\infty| = 1.701$, $|f_{AR}^\infty| = 1.845$, respectively. The element shape and size change significantly as the deformation is increased.

4.2 Translation and Rotation tests

To test the accuracy of our approach, translation and rotation tests in [10] are performed. The original mesh can be seen in Figures 6 and 7. There is a line object located in $(-L, 0) \times (L, 0)$ in a $(-2L, -2L) \times (2L, 2L)$ domain. A total of 2182 triangles are generated for the mesh.

For the translation tests, the object is moved $0.5L$ upwards. The movement is performed in 10 steps with $0.05L$ and in 5 steps with $0.1L$ movement upwards in two different training settings. The last step of the movement can be seen in Figure 6. In Figure 8, the PINN method is compared with the approach in [10]. The area and shape change metrics of two PINN solutions are presented alongside the classical finite element solutions and solutions with Jacobian-based stiffening. The authors applied a stiffening power to prevent the deformation of the smaller elements. The stiffened approach represents the best value obtained in [10] with different applied stiffening power. The two PINN solutions are representing the overall motion in 5 and 10 steps. The total number of steps is represented in parentheses in the figure. As seen in the first row of Figure 8, the PINN solutions are comparable with the FEM solutions with Jacobian-stiffening. As mentioned before, the PINN approach does not have any criteria to prevent mesh overlapping and sudden movements move the vertex nodes in an undesired way. Therefore, the quality of the deformed mesh improves as the number of steps increases.

For the rotation tests, the object is rotated 0.25π counterclockwise. Again, to prevent overlapping
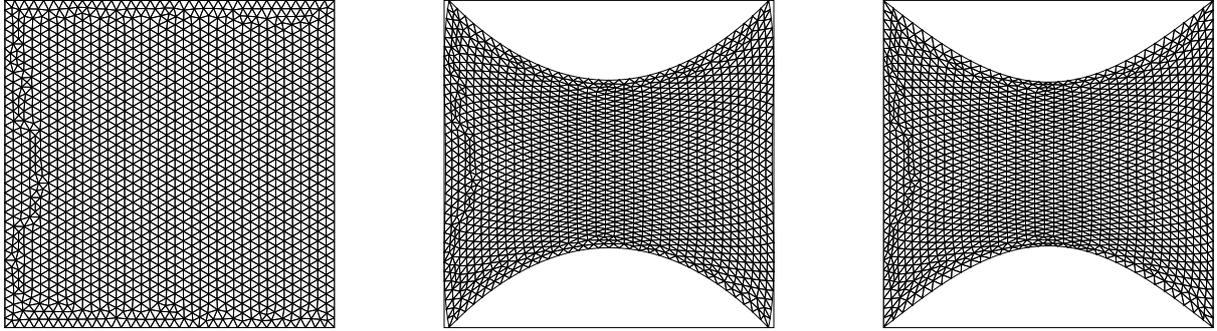
Figure 4: Initial and deformed meshes of the deformed square case. The square is squeezed from its top and bottom boundary. The first deformed figure shows the solution with classical PINN. The last figure represents the solution with exact boundary enforcement.
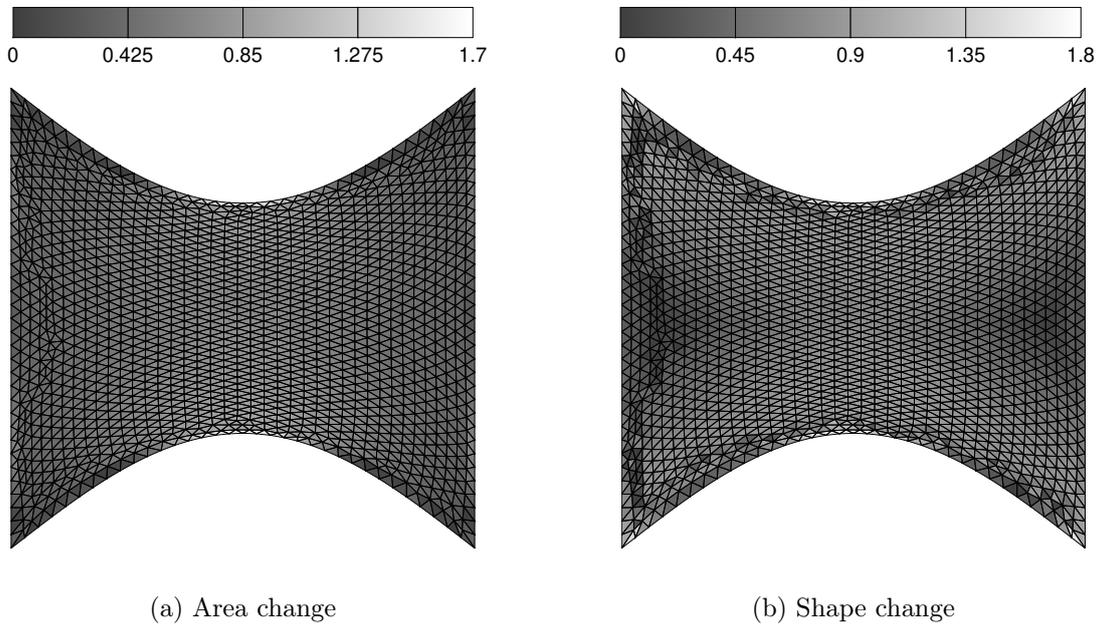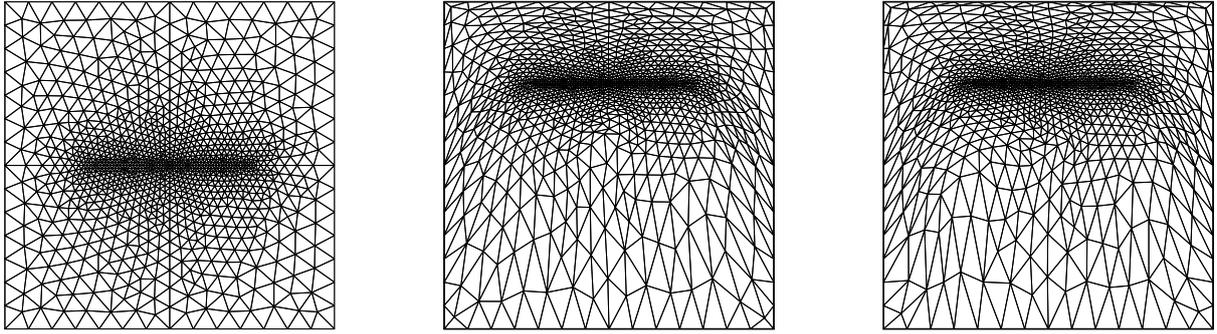

(a) Area change

(b) Shape change

Figure 5: Element quality metrics of the square deformed from the top and bottom boundaries. The figure on the left shows the element area change and the figure on the right shows the element shape change with respect to the initial mesh elements.

Figure 6: Initial mesh and deformed mesh after a total translation of 5 units. The solution in the middle is performed in 10 steps while the solution on the right is performed in 5 steps.

of edges and collision of vertices, the movement is performed in steps with 0.025π and 0.05π counterclockwise movement in each step in two different training. The last step of the rotation can be seen in Figure 7. The deformed mesh differs especially on the boundaries between different PINN solutions. The small elements near the moving boundary start to collapse in the PINN solution with 5 steps. As the number of steps increases, the mesh quality increases. The comparison of the rotation tests with the same finite element solution of the translation tests is presented in Figure 8. The PINN approach again lies between the classical solution and the solution with Jacobian-based stiffening.
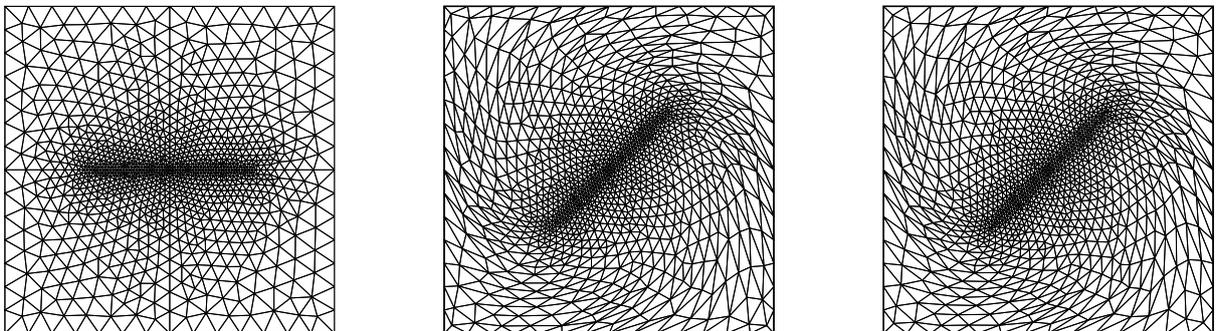

Figure 7: Initial mesh and deformed mesh after a total rotation of 0.25π. The solution in the middle is performed in 10 steps while the solution on the right is performed in 5 steps.

Table 1: Global area and shape changes of translation tests. The solution is performed in 10 steps. The values are given in every step.

Δy	0.05	0.1	0.15	0.2	0.25	0.3	0.35	0.4	0.45	0.5
$|f_A|_\infty$	0.667	1.196	1.189	1.291	1.518	1.627	1.697	1.715	1.791	1.998
$|f_{AR}|_\infty$	0.596	1.125	1.118	1.220	1.447	1.556	1.626	1.663	1.921	2.258

In both tests, the global mesh quality metric presented in section 3 is used. The $|f_A|_\infty$ and $|f_{AR}|_\infty$ are calculated as the maximum area and shape change of the values in Equation 10 in every step. The
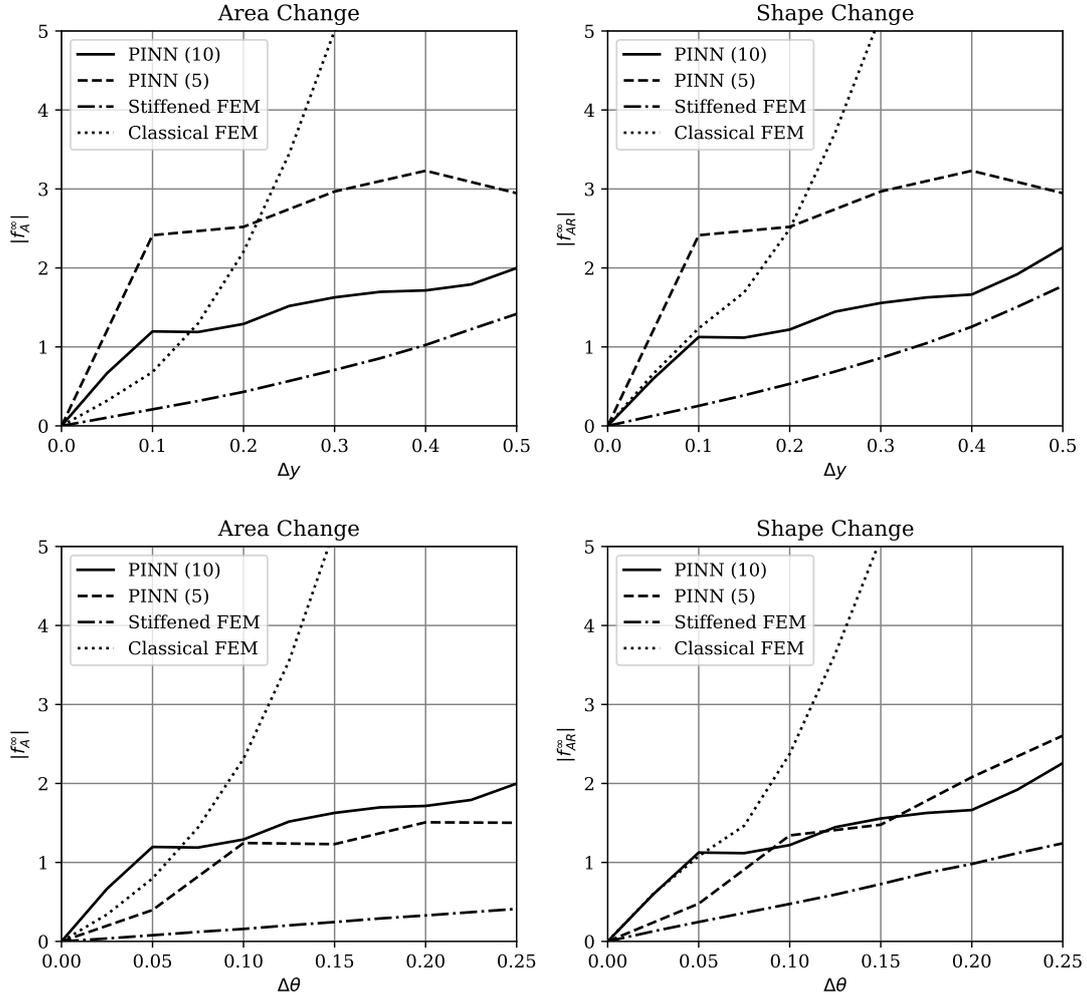
10

Figure 8: Global area and shape change metrics of the translation and rotation tests compared with the FEM solution in [10]. The first row shows the comparison of the translation test, while the second row shows the comparison of the rotation tests.

Table 2: Global area and shape changes of rotation tests. The solution is performed in 10 steps. The values are given in every step.

$\Delta\theta(\pi)$	0.025	0.05	0.075	0.1	0.125	0.15	0.175	0.2	0.225	0.25
$|f_A|_\infty$	0.274	0.432	0.663	0.881	0.882	1.152	1.093	1.256	1.295	1.324
$|f_{AR}|_\infty$	0.355	0.508	0.753	1.034	1.266	1.554	1.914	2.137	2.517	2.573

area change and shape change values are presented in Tables 1 and 2, for the translation and rotation tests, respectively.

4.3 Flexible Beam

This test case consists of a mesh movement due to a motion of a flexible beam adapted from the problem in [50]. The beam is fixed on its left end and sits in the center of the domain. Domain dimensions are $(-10, 10) \times (-10, 10)$ and the structure's position is $(-5, 5) \times (-0.5, 0.5)$ The deformation is based on a sinusoidal function $\sin(\frac{\pi}{2}\frac{x}{L})$ with varying amplitude. The initial mesh can be seen in Figure 9. This unstructured mesh consists of 2098 triangular elements. The right end of the structure first moves to 4 units upwards, then 8 units downwards, following a 4-unit upward motion to return to its initial state. The movements are performed in steps with 2-unit motions, upwards or downwards.
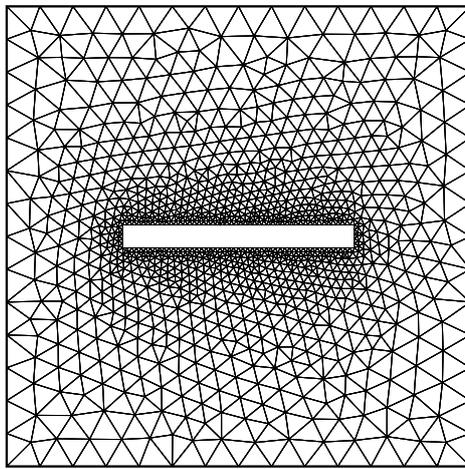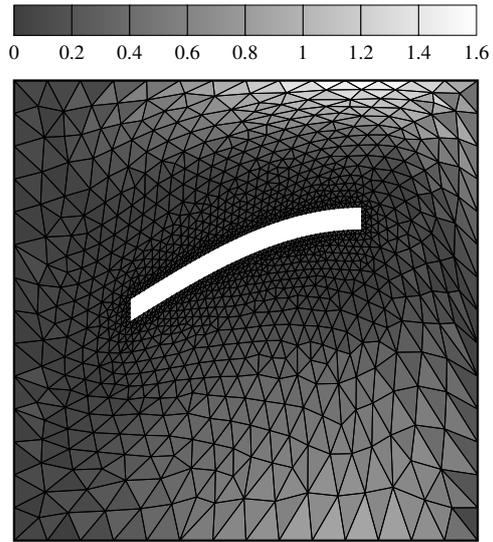

Figure 9: Initial mesh of the flexible beam test case with 2098 triangular elements. The elements are concentrated on the moving boundary to track the deformation in a precise way.

In Figure 10, the deformed mesh after two steps of movement is presented with the mesh quality presented with the global area and shape change metric. Using exact boundary enforcement gives the true boundary position and therefore fixes the vertices on the boundaries. Therefore, on the outer boundaries, elements are stretched and squeezed more than the inner elements. Especially elements near the tip of the moving boundary have the most area and shape changes.
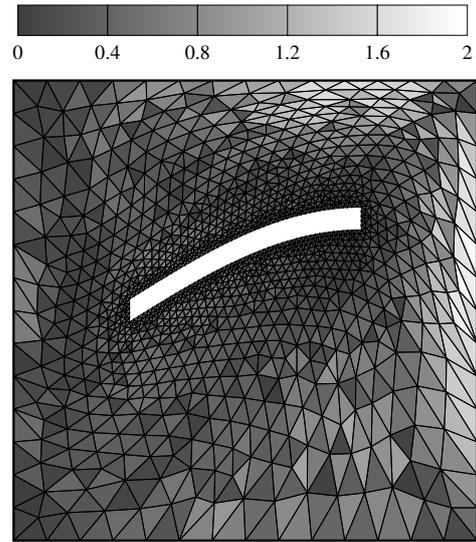
In figure 11, the mesh after one cycle of motion is presented. The structure returns to its original place after eight iterations. By looking at the area change, the sinusoidal motion of the structure can be observed. The most deformed elements are located at the top and bottom boundaries and near the moving tip of the structure. These elements are squeezed first and cannot recover themselves after the relative stretching.

5 Conclusion and Future Work

In this work, we solved mesh deformation problems with physics-informed neural networks. The selected method uses the linear elastic model since PINN can give accurate results for solving this type of PDE. We note that vertex nodes are moved according to boundary movement. Moreover,
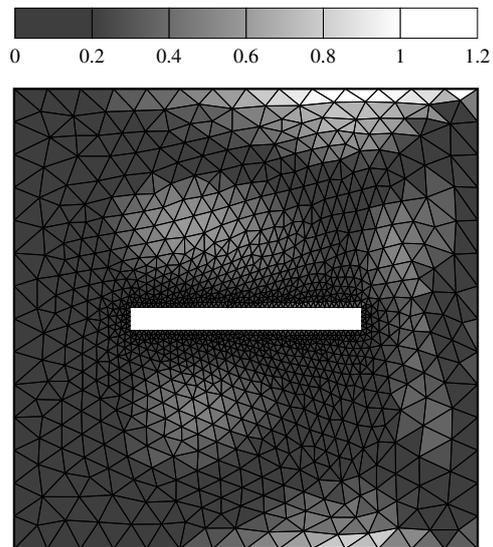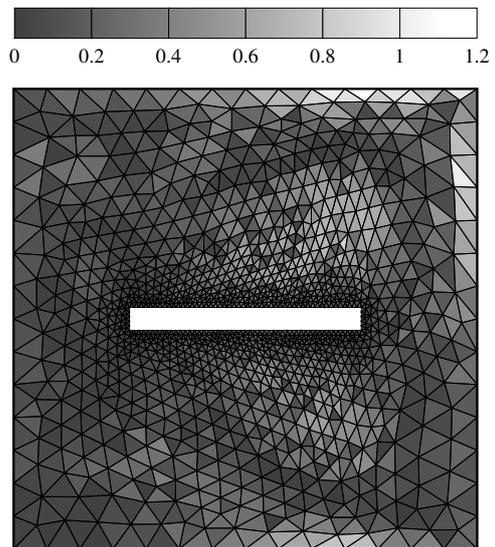
(a) Area change

(b) Shape change

Figure 10: Element quality metrics when the structure tip moves to $y = 4$.

(a) Area change

(b) Shape change

Figure 11: Element quality metrics when the structure returns its original position.

exact boundary values are enforced to satisfy the Dirichlet boundary conditions exactly. We test this approach with translation and rotation tests and compared it with finite element solutions. We showed that the PINN solution is comparable with the FEM solutions. The deformation is performed in numerous steps instead of a sudden movement. This prevents vertex collision and edge overlapping. We showed that as the number of steps is increased, the deformed mesh quality gets higher. For a greater mesh quality, the number of steps can be increased.

The mesh movement method in this paper only includes linear elastic equations, although it can be extended to other techniques. Other commonly used methods such as the Laplacian or biharmonic equations are also applicable to PINN formulation. Our future work aims to use other methods that prevent mesh overlapping in the training of the PINN. The network parameters and formulation can be extended in a way that vertex collisions and edge overlapping is prevented.

References

[1] J. T. Batina, "Unsteady Euler airfoil solutions using unstructured dynamic meshes," *AIAA journal*, vol. 28, no. 8, pp. 1381–1388, 1990.

[2] B. A. Robinson, J. T. Batina, and H. T. Yang, "Aeroelastic analysis of wings using the Euler equations with a deforming mesh," *Journal of Aircraft*, vol. 28, no. 11, pp. 781–788, 1991.

[3] Y. Bazilevs, V. M. Calo, Y. Zhang, and T. Hughes, "Isogeometric fluid–structure interaction analysis with applications to arterial blood flow," *Computational Mechanics*, vol. 38, no. 4, pp. 310–322, 2006.

[4] K. Stein, R. Benney, V. Kalro, T. E. Tezduyar, J. Leonard, and M. Accorsi, "Parachute fluid–structure interactions: 3-d computation," *Computer Methods in Applied Mechanics and Engineering*, vol. 190, no. 3-4, pp. 373–386, 2000.

[5] T. E. Tezduyar, S. Sathe, J. Pausewang, M. Schwaab, J. Christopher, and J. Crabtree, "Interface projection techniques for fluid–structure interaction modeling with moving-mesh methods," *Computational Mechanics*, vol. 43, no. 1, pp. 39–49, 2008.

[6] T. E. Tezduyar, M. Behr, S. Mittal, and J. Liou, "A new strategy for finite element computations involving moving boundaries and interfaces—the deforming-spatial-domain/space-time procedure: Ii. computation of free-surface flows, two-liquid flows, and flows with drifting cylinders," *Computer methods in applied mechanics and engineering*, vol. 94, no. 3, pp. 353–371, 1992.

[7] T. E. Tezduyar, "Finite element methods for flow problems with moving boundaries and interfaces," *Archives of Computational Methods in Engineering*, vol. 8, no. 2, pp. 83–130, 2001.

[8] A. A. Johnson and T. E. Tezduyar, "Mesh update strategies in parallel finite element computations of flow problems with moving boundaries and interfaces," *Computer methods in applied mechanics and engineering*, vol. 119, no. 1-2, pp. 73–94, 1994.

[9] C. Farhat, C. Degand, B. Koobus, and M. Lesoinne, "Torsional springs for two-dimensional dynamic unstructured fluid meshes," *Computer Methods in Applied Mechanics and Engineering*, vol. 163, no. 1, pp. 231–245, 1998.

[10] K. Stein, T. Tezduyar, and R. Benney, "Mesh moving techniques for fluid-structure interactions with large displacements," *J. Appl. Mech.*, vol. 70, no. 1, pp. 58–63, 2003.

[11] K. Takizawa, T. E. Tezduyar, and R. Avsar, "A low-distortion mesh moving method based on fiber-reinforced hyperelasticity and optimized zero-stress state," *Computational Mechanics*, vol. 65, no. 6, pp. 1567–1591, 2020.

[12] R. Löhner and C. Yang, "Improved ALE mesh velocities for moving bodies," *Communications in numerical methods in engineering*, vol. 12, no. 10, pp. 599–608, 1996.

[13] I. Robertson and S. Sherwin, "Free-surface flow simulation using hp/spectral elements," *Journal of Computational Physics*, vol. 155, no. 1, pp. 26–53, 1999.

[14] B. T. Helenbrook, "Mesh deformation using the biharmonic operator," *International journal for numerical methods in engineering*, vol. 56, no. 7, pp. 1007–1021, 2003.

[15] H. Lee and I. S. Kang, "Neural algorithm for solving differential equations," *Journal of Computational Physics*, vol. 91, no. 1, pp. 110–131, 1990.

[16] I. E. Lagaris, A. Likas, and D. I. Fotiadis, "Artificial neural networks for solving ordinary and partial differential equations," *IEEE transactions on neural networks*, vol. 9, no. 5, pp. 987–1000, 1998.

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

[18] A. D. J. . G. E. Karniadakis, "Extended physics-informed neural networks (XPINNs): A generalized space-time domain decomposition based deep learning framework for nonlinear partial differential equations," *Communications in Computational Physics*, vol. 28, no. 5, pp. 2002–2041, 2020.

[19] E. Kharazmi, Z. Zhang, and G. E. M. Karniadakis, "hp-VPINNs: Variational physics-informed neural networks with domain decomposition," *Computer Methods in Applied Mechanics and Engineering*, vol. 374, p. 113547, 2021.

[20] K. Shukla, A. D. Jagtap, and G. E. Karniadakis, "Parallel physics-informed neural networks via domain decomposition," *Journal of Computational Physics*, vol. 447, p. 110683, 2021.

[21] L. Yang, X. Meng, and G. E. Karniadakis, "B-PINNs: Bayesian physics-informed neural networks for forward and inverse PDE problems with noisy data," *Journal of Computational Physics*, vol. 425, p. 109913, 2021.

[22] A. D. Jagtap, E. Kharazmi, and G. E. Karniadakis, "Conservative physics-informed neural networks on discrete domains for conservation laws: Applications to forward and inverse problems," *Computer Methods in Applied Mechanics and Engineering*, vol. 365, p. 113028, 2020.

[23] G. Pang, L. Lu, and G. E. Karniadakis, "fPINNs: Fractional physics-informed neural networks," *SIAM Journal on Scientific Computing*, vol. 41, no. 4, pp. A2603–A2626, 2019.

[24] L. Yang, D. Zhang, and G. E. Karniadakis, "Physics-informed generative adversarial networks for stochastic differential equations," *SIAM Journal on Scientific Computing*, vol. 42, no. 1, pp. A292–A317, 2020.

[25] M. Raissi, A. Yazdani, and G. E. Karniadakis, "Hidden fluid mechanics: Learning velocity and pressure fields from flow visualizations," *Science*, vol. 367, no. 6481, pp. 1026–1030, 2020.

[26] 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*, vol. 426, p. 109951, 2021.

[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*, vol. 360, p. 112789, 2020.

[28] S. Cai, Z. Wang, S. Wang, P. Perdikaris, and G. E. Karniadakis, "Physics-informed neural networks for heat transfer problems," *Journal of Heat Transfer*, vol. 143, no. 6, 2021.

[29] A. Aygun and A. Karakus, "Physics informed neural networks for two dimensional incompressible thermal convection problems," *Isı Bilimi ve Tekniği Dergisi*, vol. 42, no. 2, pp. 221–232, 2022.

[30] Q. Lou, X. Meng, and G. E. Karniadakis, "Physics-informed neural networks for solving forward and inverse flow problems via the Boltzmann-BGK formulation," *Journal of Computational Physics*, vol. 447, p. 110676, 2021.

[31] C. L. Wight and J. Zhao, "Solving Allen-Cahn and Cahn-Hilliard equations using the adaptive physics informed neural networks," *arXiv preprint arXiv:2007.04542*, 2020.

[32] R. Mattey and S. Ghosh, "A novel sequential method to train physics informed neural networks for Allen-Cahn and Cahn-Hilliard equations," *Computer Methods in Applied Mechanics and Engineering*, vol. 390, p. 114474, 2022.

[33] S. Wang and P. Perdikaris, "Deep learning of free boundary and stefan problems," *Journal of Computational Physics*, vol. 428, p. 109914, 2021.

[34] G. E. Karniadakis, I. G. Kevrekidis, L. Lu, P. Perdikaris, S. Wang, and L. Yang, "Physics-informed machine learning," *Nature Reviews Physics*, vol. 3, no. 6, pp. 422–440, 2021.

[35] S. Wang, S. Sankaran, and P. Perdikaris, "Respecting causality is all you need for training physics-informed neural networks," *arXiv preprint arXiv:2203.07404*, 2022.

[36] N. Rahaman, A. Baratin, D. Arpit, F. Draxler, M. Lin, F. Hamprecht, Y. Bengio, and A. Courville, "On the spectral bias of neural networks," in *Proceedings of the 36th International Conference on Machine Learning*, pp. 5301–5310, PMLR, 2019. ISSN: 2640-3498.

[37] S. Wang, X. Yu, and P. Perdikaris, "When and why PINNs fail to train: A neural tangent kernel perspective," *Journal of Computational Physics*, vol. 449, p. 110768, 2022.

[38] I. E. Lagaris, A. Likas, and D. I. Fotiadis, "Artificial neural networks for solving ordinary and partial differential equations," *IEEE transactions on neural networks*, vol. 9, no. 5, pp. 987–1000, 1998.

[39] S. Ruder, "An overview of gradient descent optimization algorithms," *arXiv preprint arXiv:1609.04747*, 2016.

[40] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[41] A. G. Baydin, B. A. Pearlmutter, A. A. Radul, and J. M. Siskind, "Automatic differentiation in machine learning: a survey," *The Journal of Machine Learning Research*, vol. 18, no. 1, pp. 5595–5637, 2017.

[42] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, *et al.*, "TensorFlow: a system for large-scale machine learning," in *12th USENIX symposium on operating systems design and implementation (OSDI 16)*, pp. 265–283, 2016.

[43] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *Advances in neural information processing systems*, vol. 32, 2019.

[44] S. Wang, Y. Teng, and P. Perdikaris, "Understanding and mitigating gradient flow pathologies in physics-informed neural networks," *SIAM Journal on Scientific Computing*, vol. 43, no. 5, pp. A3055–A3081, 2021.

[45] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[46] L. Sun, H. Gao, S. Pan, and J.-X. Wang, "Surrogate modeling for fluid flows based on physics-constrained deep learning without simulation data," *Computer Methods in Applied Mechanics and Engineering*, vol. 361, p. 112732, 2020.

[47] N. Sukumar and A. Srivastava, "Exact imposition of boundary conditions with distance functions in physics-informed deep neural networks," *Computer Methods in Applied Mechanics and Engineering*, vol. 389, p. 114333, 2022.

[48] J. Berg and K. Nyström, "A unified deep artificial neural network approach to partial differential equations in complex geometries," *Neurocomputing*, vol. 317, pp. 28–41, 2018.

[49] E. Luke, E. Collins, and E. Blades, "A fast mesh deformation method using explicit interpolation," *Journal of Computational Physics*, vol. 231, no. 2, pp. 586–601, 2012.

[50] A. Shamanskiy and B. Simeon, "Mesh moving techniques in fluid-structure interaction: robustness, accumulated distortion and computational efficiency," *Computational Mechanics*, vol. 67, no. 2, pp. 583–600, 2021.

[51] A. de Boer, M. S. van der Schoot, and H. Bijl, "Mesh deformation based on radial basis function interpolation," *Computers & Structures*, vol. 85, no. 11, pp. 784–795, 2007.

[52] T. C. S. Rendall and C. B. Allen, "Efficient mesh motion using radial basis functions with data reduction algorithms," *Journal of Computational Physics*, vol. 228, no. 17, pp. 6231–6249, 2009.

[53] C. Geuzaine and J.-F. Remacle, "Gmsh: A 3-d finite element mesh generator with built-in pre- and post-processing facilities," *International journal for numerical methods in engineering*, vol. 79, no. 11, pp. 1309–1331, 2009.