UNIVERSITY OF LEEDS

Machine Learning Aided Finite Element Non-uniform Mesh Generation

Zheyan Zhang

Submitted in accordance with the requirements for the degree of PhD. Computer Science

The University of Leeds

Faculty of Engineering and Physical Sciences

School of Computing

December 2023

Intellectual Property

Some parts of the work in this thesis has been published in the following articles:

Zhang, Zheyan, Jimack, Peter K, and Wang, He (2021). “MeshingNet3D: Efficient generation of adapted tetrahedral meshes for computational mechanics”. In: Advances in Engineering Software 157, p. 103021.

Zhang, Zheyan, Wang, Yongxing, Jimack, Peter K, and Wang, He (2020). “MeshingNet: A new mesh generation method based on deep learning”. In: Computational Science–ICCS 2020: 20th International Conference, Amsterdam, The Netherlands, June 3–5, 2020, Proceedings, Part III 20. Springer, pp. 186–198

The above publication is primarily the work of the candidate.

Signed Zheyan Zhang

Acknowledgements

I would like to thank my supervisors Dr. He Wang and Prof. Peter Jimack for their support during my PhD life. I thank Dr. Wang for give me direct methodology support on my project; thanks also for his encouragement, suggestions and guidance, so that I can continuously do my research; I thank Prof. Jimack for attendance my weekly meeting and the relevant conferences; I appreciate their continuous support - the thesis is finished under their help.

I would also thank my parents for the financial and moral support. Without their support, I could not have the courage to start and finish my PhD.

Abstract

We present novel deep learning methods for guiding mesh generation during finite element simulation. The finite element (FE) method is a general method for numerically solving partial differential equations. Mesh generation is an essential and critical pre-process for the finite element method. A high quality mesh ensures low time and memory cost while preserving high accuracy. However, traditional methods for guiding non-uniform mesh generation are time consuming. The state of the art requires multiple iterations of mesh generation, FE solution, and *a posteriori* error estimation. We propose to use deep neural networks to replace *a posteriori* error estimation to guide mesh refinement with reduced iterations. The training data is generated using the conventional mesh refinement methods. After supervised learning, the neural network predicts non-uniform element size distribution instantly. What we achieve is generating a high quality non-uniform mesh with little time cost.

In this thesis, we present an introduction of finite element mesh refinement and deep learning, then a review of existing deep learning methods for solving partial differential equations and generating meshes. Then we introduce our research for finite element mesh generation in several categories of PDEs, including steady 2D, 3D and transient differential equations. The proposed methods have been implemented for Poisson's equation, linear elasticity problems and the Navier-Stokes equations of fluid dynamics. In these test problems, the use of deep learning significantly reduces the time cost and produces results with accuracy close to the traditional methods. The potential and risks of the proposed methods have been discussed. The thesis ends with a conclusion and a presentation of topics that will inspire further investigation.

Contents

1 Introduction **1**

2 Background **3**

2.1 Finite element analysis 3

2.1.1 Three steps of FEM 3

2.1.2 Formulation of FE equations by Galerkin method 5

2.1.3 Formulation of FE equations by variational method 8

2.1.4 FEM error estimation 9

2.1.5 Mesh generation 12

2.2 Deep learning 13

2.2.1 Artificial neural networks 15

2.2.2 Convolutional neural networks 17

2.2.3 Graph convolutional networks 18

2.3 Machine learning for PDEs 20

2.3.1 Machine learning based PDE solver 20

2.3.2 Machine learning aided PDE solver 22

2.4 Machine learning for mesh processing 22

2.4.1 Machine learning for mesh generation 22

2.4.2 Machine learning for mesh adaptation 23

2.5 Mean value coordinates 24

2.6 Summary 26

3 Mesh adaptation for 2D static PDEs **27**

3.1 Introduction 27

3.2 Proposed methods . . . 30
3.2.1 Overview . . . 30
3.2.2 Component Details . . . 32
3.3 Validation results . . . 34
3.3.1 2D Poisson's equation . . . 34
3.3.2 2D Linear elasticity . . . 35
3.4 Discussion . . . 38

4 Mesh for 3D elasticity 41
4.1 Introduction . . . 41
4.1.1 Non-uniform mesh generation . . . 42
4.1.2 Mean value coordinates . . . 44
4.2 Methodology . . . 44
4.2.1 Theory . . . 44
4.2.2 Software and evaluation . . . 45
4.2.3 Data generation . . . 46
4.2.4 Training and using the neural network . . . 48
4.3 Computational experiments . . . 49
4.3.1 Clamped beam . . . 49
4.3.2 Laminar material . . . 52
4.3.3 Hex-bolt with a hole . . . 54
4.3.4 Irregular polyhedron . . . 56
4.3.5 Discussion . . . 59

5 Mesh for transient PDEs 63
5.1 Introduction . . . 63
5.2 Methodology . . . 64
5.2.1 Theory . . . 65
5.2.2 Overview . . . 67
5.2.3 Architecture of the software . . . 69
5.2.4 Data generation . . . 70
5.2.5 Normalisation . . . 71
5.2.6 Training and applying the neural networks . . . 72

5.2.7 Evaluation 73
5.3 Experiments 73
5.3.1 Flow past by cylinders 74
5.3.2 Two phase flow 77
5.3.3 Compressible flow 79
5.4 Discussion 81

6 Conclusion and future work 94
6.1 Conclusion 94
6.2 Future work 95

References 96

List of Figures

2.1 An example of convolution, top left is the input, top right is the kernel and bottom is the output 17

2.2 An undirected graph with its adjacency matrix, if a vertex connects with another vertex, the corresponding entry of the adjacency is 1. When a vertex connects with itself, the corresponding entry is 2. The adjacency matrix of an undirected graph is symmetric. 18

2.3 Overview of graph convolution networks (Zhou et al. 2020), Graph neural networks (GNNs) are neural models that capture the dependence of graphs via message passing between the nodes of graphs. Convolution is the mostly used operator for GNN models. The main idea of convolution operators is to generalize convolutions from other domain to the graph domain. The two categories of graph convolution are spectral and spatial approaches. 19

2.4 Star-shaped polygon, we care the MVC of point v_0. 25

3.1 A high-level diagram of MeshingNet workflow showing the ANN alongside the training regime (left) and testing regime (right). Grey, orange and blue arrows represent data generation, training and testing processes. For each problem we only use one fine mesh and one coarse mesh. 31

3.2 Our two residual networks, illustrated with 27 input parameters. ResNet1 is a modification of the FCN with the output of the first hidden layer added to the output of the last hidden layer. ResNet2 has all hidden layers of the same dimension and the output of the first hidden layer is added to outputs of the three front hidden layers. 34

3.3 For Poisson's equation: the L1 error distribution for the LAS (left); the 4000 elements mesh generated by *Triangle* under the guidance of MeshingNet (middle); and the uniform mesh with 4000 elements generated by *Triangle* (right). 35

3.4 For Poission's problem, L1 error distribution on uniform meshes (4000 elements) and MeshingNet meshes (4000 elements). Each bar shows the proportion of test data meshes whose FE solution error is in the range of the bar: the uniform meshes give FE errors between 0.0015 to 0.0025, whilst the MeshingNet meshes give FE errors between 0.0007 and 0.0015. 36

3.5 L2 training loss on elasticity training dataset during 60 training epochs. Three curves representing FCN (blue), ResNet1 (red) and ResNet2 (green) converge individually. 37

3.6 Potential energy comparison of solving 2D elasticity test problems on 4000 element meshes: uniform element size (left); MeshingNet (using FCN) meshes (centre); and ZZ refined meshes (right). We use the HAS energy as our baseline: the MeshingNet mesh solutions have energies that are significantly closer to the HAS energies than the ZZ mesh solutions and uniform mesh solutions (since a greater proportion of results are distributed near zero). The rightmost bar represents the proportion of all tests where the energy difference is no smaller than 0.6. 38

3.7 FE error (top) relative to HAS on coarse uniform mesh, non-uniform mesh guided by MeshingNet (middle) and non-uniform mesh refined by ZZ (bottom). The left geometry is an octagon and other two are heptagons (defined by placing their final vertex at the centre of edge 7). 39

4.1 Illustration of the training data for *MeshingNet3D*: each individual problem is defined by the geometry G, the PDE parameters M and the boundary condition parameters B (not shown here). However for each such problem there are multiple sample points, X, in the domain, with the corresponding local mesh size S specified. 47

4.2 The geometry and boundary conditions for the *Clamped beam*, with constant cross section along the z-axis. The gravity is uniformly distributed over the volume. The surfaces bounded by four vertices with blue triangles are clamped. . 50

4.3 Training and validation loss of the clamped beam experiment 51

4.4 For the *Clamped beam*, FE energies of neural network (NN) generated meshes versus uniform mesh FE energies and ground truth (GT) energies. The height of each bar represents the proportion of experiment results in the energy range shown on the x-axis (as a percentage of the ground truth energy). 51

4.5 For the *Clamped beam*, ground truth meshes (top) and NN-guided meshes (bottom) for two test cases. 52

4.6 The boundary conditions and loads of the *laminar material* where the height of the interface is random 53

4.7 Training and validation loss of the laminar material experiment 53

4.8 For the *Laminar material*, FE energies of neural network (NN) generated meshes versus uniform mesh FE energies and ground truth (GT) energies. The height of each bar represents the proportion of experiment results in the energy range shown on the x-axis (as a percentage of the ground truth energy). 54

4.9 (a)(c) and (b)(d) are two problems in the *laminar material* experiments. (a) and (b) are ground truth meshes and (c) and (d) are non-uniform meshes guided by the neural network. A comparison of (b) and (d) shows NN does not refines well at the interface between two laminars. 55

4.10 The boundary conditions and loads of the $hex-bolt$ (left) and *irregular polyhedron* (right). On $hex - bolt$, eight surfaces of the hole are clamped, linear distributed pressure is applied on top and bottom surfaces. 56

4.11 Training and validation loss of the hex-bolt with a hole experiment 56

4.12 For $hex-bolt$ *with a hole*, FE energies of neural network (NN) generated meshes versus uniform mesh FE energies and ground truth (GT) energies. The height of each bar represents the proportion of experiment results in the energy range shown on the x-axis (as a percentage of the ground truth energy). 57

4.13 Hex-bolt experiment, ground truth meshes (top) and NN meshes (bottom) , the left and right are two problems that only have different geometries 58

4.14 Training and validation loss of the irregular experiment 59

4.15 For *irregular polyhedron*, FE energies of neural network (NN) generated meshes versus uniform mesh FE energies and ground truth (GT) energies. The height of each bar represents the proportion of experiment results in the energy range shown on the x-axis (as a percentage of the ground truth energy). 59

4.16 A ground truth mesh (a, c and e) and corresponding NN mesh (b, d and f) selected from 500 testing problems, they are in front (a and b), right (c and d) and bottom (e and f) views. 62

5.1 A comparison of conventional FE with mesh adaptation process and our proposed parallel framework. In one these fictitious fluid simulations, the conventional method takes 4.1 seconds for a time step while the proposed parallel framework only takes 2.1 seconds. GPU is not good at all the tasks (especially when working with unstructured meshes) thus is only used for running the NN (inference). . . . 67

5.2 The original set up of the simulation with adaptive mesh 69

5.3 The architecture of our proposed model 69

5.4 The geometry G1 (top) and G2 (bottom) 75

5.5 The evolution vertical velocity of the flow past cylinders in time steps 2, 100, 200 and 300. 83

5.6 The L1 error of velocity per area during 300 time steps of simulation, all the meshes are non-uniform and time dependent. The reference solution is from "ground truth" mesh simulation 84

5.7 The L1 error of velocity per area during 300 time steps of simulation, all the meshes are non-uniform and time dependent. The reference solution is from ultra fine "reference" mesh simulation. 84

5.8 From top to bottom are the background mesh, ground truth mesh, baseline mesh, NN3 mesh, NN5 mesh and reference mesh. The reference mesh has around 440000 elements and other meshes have around 130000 elements. 85

5.9 The evolution of the sloping two phase flow in time steps 2, 10, 20, 30, 40, 50, 60, 70, 80, 90 and 100. The yellow region is filled with low density flow and the red region is filled with high density flow. 86

5.10 The L1 error of density per area during 100 time steps of simulation, all the meshes are non-uniform and time dependent. Initial fluid interface is y=0.25+0.3x. 87

5.11 The NN mesh of time step 10, 20, 40, 60, 80, 100,initial interface is y=0.25+0.6x. 88

5.12 The L1 error of density per area during 100 time steps of simulation, all the meshes are non-uniform and time dependent. Initial fluid interface is y=0.1+0.375x. 88

5.13 From top to bottom are the NN mesh at step 15, NN mesh at step 25 and ground truth mesh at step 24. 89

5.14 Top is the NN mesh at step 23 and bottom is the NN mesh at step 24. The mesh near the interface is not very anisotropic. This is perhaps because at the interface the mesh is isotropic and other region in refined by smoothness. 90

5.15 The geometry of the compressible flow experiment 91

5.16 The evolution of density of the flow in time steps 0, 25, 50, 100, 150 and 200. . . 91

5.17 The L1 error of density per area during 200 time steps of simulation, all the meshes are non-uniform and time dependent. 92

5.18 The L1 error of density per area during 200 time steps of simulation, all the meshes are non-uniform and time dependent. The reference solution is from "ground truth" mesh simulation 92

5.19 From left top to bottom are the background mesh, ground truth mesh, baseline mesh, NN mesh and reference mesh. We refine the mesh according to the Hessian of the density. 93

List of Tables

4.1 Comparison of 5 different fully connected NNs based upon normalised average energies of the finite element solutions. NN3 gives the lowest average energy and therefore provides the best mean performance. 60

5.1 A set of 6 training problems with different geometry and inlet velocity. They are used to train 6 NNs from NN1 to NN6. 76

5.2 A set of 5 training problems with different initial condition. 78

Chapter 1

Introduction

Deep learning has made great successes in the last decade. The widely known applications of deep learning include image and natural language processing. Recently, people have begun to investigate ways of using supervised and unsupervised deep learning to solve partial differential equations (PDEs). Before giving the general introduction to the rest of the thesis, we need to do an introduction of the general problem. We describe an investigation on machine learning of PDE solvers in chapter 2 and list the progress and limitations of previous work. We observe researchers have difficulties in producing techniques that are as reliably accurate as the conventional methods such as the finite element (FE) method. Hence we decided to look at ways of using machine learning in combination with FE so as to maintain the reliability but improve the efficiency. We turn our interest into using deep learning to accelerate finite element mesh generation. Non-uniform mesh generation is critical where good mesh size distribution promotes computing efficiency. However it is tolerant of some errors where mesh size distribution does not have to be very precise, which fits the current deep learning techniques where deep learning usually suffers output accuracy problem. We reviewed previous attempts of using machine learning techniques for mesh adaptation. We aim to use deep learning to guide mesh adaptation without prior knowledge about where the mesh should be refined.

In the work reported in chapter 3 and chapter 4 we propose deep neural networks models, MeshingNet and MeshingNet3D, to learn from the *a posteriori* error estimate on an initial coarse uniform mesh and predict non-uniform mesh density for refinement, without the need for solving an FE system and then computing the *a posteriori* error estimate. MeshingNet is trained using an accurate error estimation strategy which can be expensive but is only computed offline.

Hence, the mesh generation process itself is extremely fast since it is able to make immediate use of standard, highly-tuned, software to produce a high quality mesh at the first attempt (at similiar cost to generating a uniform mesh with that number of elements). Rather than using deep learning to predict PDE solutions, we only aim to use deep learning to predict mesh size distribution, because we can provide more reliable predictions in this way, based upon the observation that a greater variation in the quality of predictions can be tolerated for the mesh size than for the solution itself. We further include research on the transient PDEs, specifically fluid dynamic problems in chapter 5. We run accurate simulation to simulate fluids flow and store the data on a relatively coarse background mesh. Using the sequences data to train a graph convolution neural network by supervised learning, the network is able to predict a new mesh density distribution and guide adaptive mesh refinement in parallel with the FE solution and error estimation at the current time step. In this way we accelerate the overall simulation time.

The remainder of this thesis is structured as follows. In Chapter 2, we introduce the background knowledge of the techniques used in this thesis, such as finite element method. In Chapter 3 and Chapter 4 we introduce MeshingNet and MeshingNet3d to demonstrate how to use neural networks to guide mesh generation with mean value coordinates for static problems. Finally, in Chapter 5 we introduce the work that investigates generating the mesh for the next time step in parallel with the FE solution and error estimation at the current time step in transient fluid flow problems. In chapter 6 we conclude the above work and list several future works.

Chapter 2

Background

This chapter introduces the techniques that closely link this project such as finite element analysis and deep learning. Then we review the literature on machine learning based mesh processing.

2.1 Finite element analysis

Finite element (FE) analysis is a numerical approach to answer scientific questions and guide engineering practice with partial differential equation solving or functional minimisation backgrounds. The ideas of FE dates back to 1940s (Liu et al. 2022). FE basically uses assembly of simple geometries such as triangles to approximate field equations. The simple geometries are called elements where each element uses a low degree polynomial to approximate the solution. When the size of elements becomes infinitesimally small, the approximated solution converges to the analytical (exact) solution. On the other hand when finite elements are applied, the error of numerical solution is tolerable by refining the meshes, which divide the domain to finite elements. In this section, we introduce how FEM works, classical FEM applications and the challenges remaining.

2.1.1 Three steps of FEM

Typical FEM works in 3 steps, pre-processing, solving the equation system and post-processing. The first step of pre-processing is mesh generation, which divides the region into finite elements. A mesh file usually contains coordinates of nodal points and connectivity of nodes. Because finite element models contain very large number of degrees of freedom, it is impossible to create meshes fully manually. Structured meshes are typically composed of quadrilateral elements in 2D and hexahedral elements in 3D. The advantage of such a mesh is that the points of an elemental cell can be easily addressed by a double of indices (i, j) in two dimensions or a triple of indices (i, j, k) in three dimensions. Generation of such meshes generally requires gross partitioning and mapping blocks to simple geometries. Structured meshes require significantly less memory and are more efficient (Bern and Plassmann 2000). Generating structured meshes on complicated geometry is very difficult while unstructured meshes can be successfully applied. Triangle elements in 2D and tetrahedral elements in 3D offers flexibility over complex domain geometry. One of the important methods is Delaunay triangulation (Cheng et al. 2012) which allows users to define mesh density distribution in arbitrary domains. After the mesh is generated, interpolation functions should be selected. Usually polynomial functions are selected. Then the matrix equation for the elements should be established. The nodal values and other parameters are linked by the variational approach or weighted residual approach. The last step of pre-processing is to assemble the element equations to global matrix equations. Boundary conditions are imposed at this step.

The second step of FEM is solving the global equation. The equations are linear when the PDE is linear. If the PDE is nonlinear the finite element system is a set of nonlinear algebraic equations. Normally this is solved by a Newton's method or some variant of it. They transfer the non-linear problem to a sequence of linear problems. Usually the FE system is sparse where the diagonals are generally nonzero and most elements of the matrices are zero.

One way to solve the sparse algebraic equations are the iterative methods. For large sparse problems iterative methods are preferable because of less computing time and memory requirements. The conjugate gradient method (Golub and Van Loan 2013) is a successful iterative method. When use conjugate gradient method to solve a linear system:

$$\mathbf{Ax} = \mathbf{b} \tag{2.1}$$

An improved convergence rate can often be achieved by replacing the system by the preconditioned system (Kaasschieter 1988):

$$\mathbf{M}^{-1}\mathbf{A}\mathbf{x} = \mathbf{M}^{-1}\mathbf{b} \tag{2.2}$$

$\mathbf{M}$ is a symmetric positive definite matrix and the matrix $\mathbf{M}^{-1}\mathbf{A}$ has a more ‘favourable’ spectrum of eigenvalues than $\mathbf{A}$. The conjugate gradient method is only suitable for symmetric positive-definite and semi-definite systems. Most of the thesis (except for transient problems in Chapter 5) we consider linear self-adjoint PDE systems and therefore the finite element equations are symmetric positive-definite. For non-symmetric problems the iterative solvers generally require more iterations to finish. Such solvers include GMRES (Saad and Schultz 1986) and BiCGstab (Sleijpen and Fokkema 1993).

Another way to solve the algebraic equations are the direct methods which are usually used for problems with moderate size. Lower Diagonal Upper (LDU) methods factor a square matrix into a product of a lower triangular matrix (L), a diagonal matrix (D) and a upper triangular matrix (U). Solution of LDU method consists three stages: factorisation, farward solution and back substitution (Nikishkov 2004). Other direct solvers include Cholesky (Higham 2009) factorisation and MUMPS (Amestoy et al. 2000) parallel solver. The computational complexity of most solvers is worse than linear. In some special cases, the computational work grows linearly with the degrees of freedom (Notay 2010).

The last step of FEM is post-processing. The values of certain functions are obtained from the solution. In many cases additional processing may be required. In elasticity problems for example, strain and stress are critical for the safety of mechanical structures which can be computed from the displacement field (which may be the actual solution). Usually, visualisation of solutions are extremely helpful for the users of FEM. The visualisation are provided either by FE software (Hecht 2012) or independent post-processing software (Ahrens et al. 2005).

2.1.2 Formulation of FE equations by Galerkin method

There are two ways of formulation of FE equations, direct solving variational method such as Ritz method and weighted residual method such as Galerkin method. This subsection uses 1D Poisson equation to introduce the formulation with linear elements by Galerkin method and next subsection variational method (Nikishkov 2004). Suppose we solve equation below with two linear elements by Galerkin method:

$$a\frac{d^2u}{dx^2} + b = 0, \quad 0 \leq x \leq 2L \tag{2.3}$$

with boundary conditions

$$\begin{aligned} &u|_{x=0} = 0, \\ &a\frac{du}{dx}|_{x=2L} = R. \end{aligned} \tag{2.4}$$

On the left Dirichlet boundary condition is applied and on the right Neumann boundary condition is applied. Each element has two nodes and we can approximate the function with nodal values and shape function:

$$u = N_1u_1 + N_2u_2 \tag{2.5}$$

Where u_1 is the left nodal value and u_2 is the right nodal value. When x_1 and x_2 are the left and right nodal coordinates, the shape function is constructed:

$$\begin{aligned} N_1 &= 1 - \frac{x - x_1}{x_2 - x_1}, \\ N_2 &= \frac{x - x_1}{x_2 - x_1}. \end{aligned} \tag{2.6}$$

When u is approximated by the linear combination of the nodal values, the equation can be expressed as:

$$a\frac{d^2}{dx^2}[N]\{u\} + b = \psi \tag{2.7}$$

$$[N] = [N1 \quad N2] \tag{2.8}$$

$$\{u\} = \{u1 \quad u2\} \tag{2.9}$$

where ψ is the residual representing the difference of the exact and approximated value in the element. Galerkin method ensures the integration of the nonzero weighted residual over the element to be zero. In the integration, the residual is weighted by the shape function (we still work on a single element).

$$\int_{x_1}^{x_2} [N]^T a \frac{d^2}{dx^2}[N]\{u\}dx + \int_{x_1}^{x_2} [N]^T b dx = 0 \tag{2.10}$$

Integrate by parts the discrete form becomes:

$$\int_{x_1}^{x_2} [\frac{dN}{dx}]^T a [\frac{dN}{dx}] dx\{u\} - \int_{x_1}^{x_2} [N]^T b dx - [N]^T a \frac{du}{dx}|_{x=x2} + [N]^T a \frac{du}{dx}|_{x=x1} = 0 \tag{2.11}$$

If we work on the right element $a\frac{du}{dx}|_{x=x2}$ is obtained from the Neumann boundary condition.

In solid mechanics the relation is presented as:

$$\begin{aligned}
&[k]\{u\} = \{f\} \\
&[k] = \int_{x_1}^{x_2} [\frac{dN}{dx}]^T a [\frac{dN}{dx}] dx \\
&\{f\} = \int_{x_1}^{x_2} [N]^T b dx + [N]^T a \frac{du}{dx}|_{x=x2} - [N]^T a \frac{du}{dx}|_{x=x1}
\end{aligned} \tag{2.12}$$

Where $[k]$ is stiffness matrix and $\{f\}$ is load vector. For two elements:

$$\begin{aligned}
&[k1] = [k2] = \frac{a}{L}\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \\
&\{f_1\} = \frac{bL}{2}\begin{Bmatrix} 1 \\ 1 \end{Bmatrix} \quad \{f_2\} = \frac{bL}{2}\begin{Bmatrix} 1 \\ 1 \end{Bmatrix} + \begin{Bmatrix} 0 \\ R \end{Bmatrix}
\end{aligned} \tag{2.13}$$

The assembled global equation system is obtained by above two separate elements and their equations:

$$\frac{a}{L}\begin{bmatrix} 1 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 1 \end{bmatrix}\begin{Bmatrix} u_1 \\ u_2 \\ u_3 \end{Bmatrix} = \frac{bL}{2}\begin{Bmatrix} 1 \\ 2 \\ 1 \end{Bmatrix} + \begin{Bmatrix} 0 \\ 0 \\ R \end{Bmatrix} \tag{2.14}$$

We apply the Dirichlet boundary condition and change the first row of the system:

$$\frac{a}{L}\begin{bmatrix} 1 & 0 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 1 \end{bmatrix}\begin{Bmatrix} u_1 \\ u_2 \\ u_3 \end{Bmatrix} = \frac{bL}{2}\begin{Bmatrix} 0 \\ 2 \\ 1 \end{Bmatrix} + \begin{Bmatrix} 0 \\ 0 \\ R \end{Bmatrix} \tag{2.15}$$

An alternative is to eliminate $u1$ from the system entirely.

2.1.3 Formulation of FE equations by variational method

In this subsection we derive the FE formulation of variational method (Zienkiewicz, Taylor, et al. 2005). We consider the physical meaning of this problem is tension of one dimensional bar with distributed load b and concentrated load R at x_2. The Young's Modulus is E and cross section area is A. $a = EA$ is the resistance to deformation. This linear elasticity problem can be regarded as minimisation of the potential energy Π:

$$\Pi = \int_0^{2L} \frac{1}{2}a(\frac{du}{dx})^2 dx - \int_0^{2L} budx - Ru|_{x=2L} \tag{2.16}$$

The condition for the minimisation of the potential energy is:

$$\frac{\partial \Pi}{\partial u_i} = 0, i = 1, 2, 3 \tag{2.17}$$

Where u_i is the nodal displacement. Using shape function and two elements, the potential energy of the right element is:

$$\Pi = \int_{x_1}^{x_2} \frac{1}{2}a\{u\}^T[\frac{dN}{dx}]^T[\frac{dN}{dx}]\{u\}dx - \int_{x_1}^{x_2} \{u\}^T[N]^T bdx - \{u\}^T \begin{Bmatrix} 0 \\ R \end{Bmatrix} \tag{2.18}$$

Where $[\frac{dN}{dx}]$ and $[N]$ is the same as the ones in the Galerkin method. The finite element equation of this element is:

$$\int_{x_1}^{x_2} \frac{1}{2}a[\frac{dN}{dx}]^T[\frac{dN}{dx}]\{u\}dx - \int_{x_1}^{x_2} -[N]^T bdx - \begin{Bmatrix} 0 \\ R \end{Bmatrix} = 0 \tag{2.19}$$

This is the same as equation 2.11, and the assembled global system is the same as the system of Galerkin method.

2.1.4 FEM error estimation

Finite element method as a numerical method involves several sources of numerical errors. This thesis focuses on the discretisation error which is because of the limitation of the finite element space to approximate the solution. The analysis of such error either gives the asymptotic information or can be used to steer mesh refinement(spaces h-version adaptivity) or increasing the order of the element polynomial degree chosen(spaces p-version adaptivity) (De SR Gago et al. 1983). The former error is called *a priori* error and the latter is called *a posteriori* error where the solution of FE analysis is employed. In this subsection we introduce *a priori* error and the *a posteriori* error analysis which is used for the adaptive mesh generation in this thesis.

Consider a boundary value problem with two dimensional Poisson's equation as model problem:

$$\begin{aligned} &-\Delta u = f \quad \text{on} \quad \Omega; \\ &u = 0 \quad \text{on} \quad \Gamma_D; \\ &\mathbf{n} \cdot \nabla u = g \quad \text{on} \quad \Gamma_N; \end{aligned} \tag{2.20}$$

where $\Gamma = \Gamma_D \cup \Gamma_N$ and $\mathbf{n}$ is the unit vector outward to Γ. The variation form of such problem could be:

$$\begin{aligned} a(u,v) &= l(v) \quad \forall v \in V \\ a(u,v) &= \int_\Omega \nabla u \cdot \nabla v d\Omega \\ l(v) &= \int_\Omega f v d\Omega + \int_{\Gamma_N} g v ds \end{aligned} \tag{2.21}$$

Where V is the space of all functions that are zero on the Dirchlet boundary and whose first derivatives are square integrable over Ω, The form $a(u,v)$ is assumed to be a bilinear form on $V(\Omega) \times V(\Omega)$ and the linear functional $l(v)$ is an element of the dual space $V'(\Omega)$. The energy norm is defined:

$$||v||_E = \sqrt{a(v,v)} \tag{2.22}$$

The existence and uniqueness of the solution is provided by the Lax-Milgram theorem (Ramaswamy 1980):

$$|a(v,w)| \leq M||v||_V||w||_V \quad \forall v,w \in V \times V$$
$$a(v,v) \geq \alpha||v||_V^2 \tag{2.23}$$

Above are the conditions that guarantee existence of a unique solution. M and α are positive constants independent of v and w. Solving the finite element find the solution $u_h \in V_h$ where V_h is a subspace of V, the FE equation is:

$$a(u_h, v_h) = l(v_h) \quad \forall v_h \in V_h \tag{2.24}$$

The finite element error is $e_h = u - u_h$, then define the residual:

$$a(e_h, v) = l(v) - a(u_h, v) \quad \forall v \in V \tag{2.25}$$

Because of Galerkin orthogonality (Angermann 2002), the residual is 0 when the test function is in the FE space.

$$a(e_h, v_h) = 0 \tag{2.26}$$

A priori error does not offer the exact error of a given mesh at engineering practice but instead as already outlined, it gives the general information of asymptotic behaviour (Grätsch and Bathe 2005).

$$||u - u_h||_E = min||u - v_h||_E \quad \forall v \in V_h \tag{2.27}$$

It states that for error in the energy norm, the FE solution u_h is the closest to the exact solution u than any other solution v_h in the FE space V_h.

Céa's lemma (Chen and Křížek 2006) asserts:

$$||u - u_h||_V \leq \frac{m}{\alpha} \inf_{v_h \in V_h} ||u - v_h||_V \tag{2.28}$$

V is a real Hilbert space with the norm $||\cdot||$. The error of FE solution in V-norm is of the same order of the interpolation error. Further, errors measured in the H_1 and L_2 norm:

$$\begin{aligned} ||u - u_h||_{H1}(\Omega) &\leq ch^p ||u||_{H^{p+1}(\Omega)} \\ ||u - u_h||_{L2}(\Omega) &\leq ch^{p+1} ||u||_{H^{p+1}(\Omega)} \end{aligned} \tag{2.29}$$

$||u||_{L2} = (\int u^2 dx)^{1/2}$ and $||u||_{Hp} == \sum_{i=0}^{p} ||D^i u||_{L2}$. The finite element solution converges in these norms in $O(h^p)$ and $O(h^{p+1})$, where h denotes the element size and p denotes the polynomial order.

Even though *a priori* error estimate provides the information of convergence, it does not point out where in the domain the error is higher. It also applies only for uniform mesh refinement (so that the mesh size is decreased at the same rate throughout the domain). To help the decision of adaptive mesh refinement, we consider the *a posteriori* error. The early *a posteriori* errors are for the energy norm (Babuška and Rheinboldt 1978) and later goal oriented errors (Oden and Prudhomme 2001) are developed. Goal oriented methods are used when the energy norms are not useful or when specific quantities are of interest.

In the energy norm methods, explicit error estimators directly use the data from finite element estimation whereas implicit error estimators solve auxiliary problems. It is possible to quantify almost the true error if we have the solution from an extremely fine mesh. However such solution is too expensive in both memory and time costs. Solution difference estimators compare the difference of two solutions but without requiring the fine mesh to be superfine. The difference of solution from N elements and 2N elements mesh can be useful error indicator (Kita and Kamiya 2001). Another explicit error estimator is based on the interior element residuals and jumps (discontinuity of normal flux) at the element boundaries (Babuška and Rheinboldt 1978, Lazarov et al. 2009). Where for the model problem, the interior element residual is

$$R = f + \Delta u_h \tag{2.30}$$

This does not work for piecewise linear finite elements as the residual is always equal to f in this case.

Implicit error estimators (Bank and Weiser 1985, Prudhomme et al. 2004) approximate the actual error by solving auxiliary problems. The problem is solved either on a single element (element residual method) or on a small subdomain of elements (subdomain residual method)

(Ainsworth and Oden 1997). Another type of energy norm error estimator other than explicit and implicit estimators is recovery error estimator (Zhang and Naga 2005, Carstensen and Funken 2001). It uses the fact that gradient of FE solution is generally not continuous on inter-element edges. Then a post-processed (smoothed) gradient should be closer to the real gradient than the original gradient from FE solution. The local error e_h is estimated as the difference of the recovered ($M[u_h]$) and the original gradient (∇u_h):

$$(e_h)^2 = (M[u_h] - \nabla u_h)^2 \tag{2.31}$$

A widely used recovery estimator in the elasticity problems is called ZZ error estimator which is based on super-convergence property (Zienkiewicz and Zhu 1992). For triangle elements, the smoothed nodal gradients are calculated as the average of the surrounding elements', which contributes to the recovered elemental gradients. In this thesis we mainly use recovery-based error estimates however the methods that we present are not restricted to this family of methods.

2.1.5 Mesh generation

Mesh generation here means the process of creating a mesh to be used for FEM. This process is time consuming and error prone (Zienkiewicz, Taylor, et al. 2005). It is not surprising that current engineering practices can use meshes with up to tens of billions nodes (Kato et al. 2020). Hence, automatic mesh generation has been an active topic for decades (Ho-Le 1988, Baker 2005, Owen 1998). Structured meshes usually present as a combination of quadrilateral in 2D and hexahedral in 3D. Generating such meshes generally takes two steps: divide the geometry into simple subregions such as blocks, map the subregions onto regular grids. The difficulty lies on the first step where dividing the geometry is the manual work. Unstructured meshes usually use triangle elements in 2D and tetrahedral elements in 3D. Generating such meshes most commonly uses Delaunay triangulation (Caendish et al. 1985) or the advancing front method (Lo 1985). A special type of unstructured mesh generator is hierarchical tree(Yerry and Shephard 1984) in which quadrilateral and hexahedral elements (Ito et al. 2009) can be used.

The general concept of mesh generation also involves adaptive mesh refinement and mesh quality (Parthasarathy et al. 1994) enhancement. Adaptive mesh refinement is the technique that modify the mesh according to the previous result. The refined mesh aims for either permissible error with the least cost or best accuracy under prescribed cost. Note that even if mesh generation is automatic, the cost of memory and time is significant. The uniform mesh is generally far from optimal when the error is non-uniformly distributed. An example is stress concentration in elasticity problems, where the highest stress and error are usually on the location where load or constraints are applied. Even though a non-uniform mesh can be designed manually by expert knowledge, for a complex problem (such as multi-physics) and time dependent problems (such as fluid dynamics), automatic refinement techniques are more useful. There are three types of refinement:

1. h refinement: Keep the element type, adjust the element size.
2. p refinement: keep the original element size, increase the order of polynomial in elements.
3. r refinement: Keep the original nodes number, change the position of nodes (W. Huang et al. 1994).

There are several ways to implement h-refinement such as subdivision and re-meshing. Complete mesh regeneration can also be efficient. The key is to know where the mesh should be coarse and where it should be fine. An ideal mesh for efficiency should have error equally distributed (Zienkiewicz, Taylor, et al. 2005). One strategy is using mesh density or mesh size function. Knowing the solution, mesh size distribution can be defined on either uniform background mesh or previous non-uniform mesh. The state of art mesh size distribution is quantified by estimating *a posteriori* error. For a prescribled target error, error estimating and mesh refinement could act iteratively for several times. Likewise, p-refinement could require a sequence of steps. The joint of h and p methods (hp-refinement) can be efficient. One of the method (Zienkiewicz, J. Zhu, et al. 1989) is pursuing non-uniform h-refinement first to get error almost equal distribution, then do uniform p refinement to increase accuracy globally. Moving mesh method (Baines et al. 2005) is r-refinement which suitable for moving boundary problems. r-refinement generally can not guarantee the prescribed accuracy. It is difficult to use in practice. In this thesis we use the approach of h-refinement.

2.2 Deep learning

Building an intelligent machine that can think is an ancient idea. Since the birth of electrical computers, hopes of artificial intelligence fall on these programmable machines. In 20th century, computers have shown strong ability and potential in problems that are formally describable.

A good example is solving partial differential equations by finite element method which is extremely difficult for human beings. However, tasks without mathematical forms such as image or speech recognition were challenging for early days of electrical computers. These mathematically vague tasks are easy for human where we can instantly recognise faces or animals when we see and understand words when we hear. Early artificial intelligence is good at computational expensive and application limited problems. For example, the chess player Deep Blue (Hsu 2002) is a successful rule based AI system whose operation is clearly defined by program. The expert systems like Deep Blue obtain hard-coded knowledge from human experts. The rigid knowledge limits the intelligence of machines when many tasks being undertaken are flexible. The expert system can hardly deal with tasks such as objects detection.

For human beings our knowledge is learned from experiences rather than DNA. For example, a child learns what is fruit by tasting fruits such as apples and oranges. To overcome the above shortage of expert system, machines acquire knowledge by extracting patterns from raw data (Goodfellow et al. 2016). We call this capability machine learning. Naive Bayes (Rish et al. 2001) is a kind of machine learning algorithm that is good at classification tasks, such as spam email filtering. Conventional machine learning techniques depend on the representation of the data. Representation learning (Bengio et al. 2013) allows the machine to learn the way to transforming raw data to representative features. Such features could be non-describable to humans. Deep learning (LeCun et al. 2015) is a kind of tool that uses deep hierarchical structural layers to extract features and solve AI problems. Despite the hierarchical structures are designed by human, the parameters in each layer is generally adjustable. The process to adapt the parameters to tasks is called training. Supervised training is very common where data is labeled. For example photos for training are labeled ”cat” or ”dog” in animal classification. Unsupervised and semi-supervised training requires less labelling work however the system is harder to train. Deep learning has made great success in image and language understanding and contributed to applications such as:

1. medical image analysing (Shen et al. 2017)
2. facial recognition (Guo and Zhang 2019)
3. natural language processing (Young et al. 2018, Otter et al. 2020)
4. self-driving cars (Rao and Frtunikj 2018)

5. products recommendation (Da'u and Salim 2020, Naumov et al. 2019)

In this section, we will introduce several models of deep learning such as convolutional neural networks and briefly review the application of deep learning in scientific computing, especially in mesh generation.

2.2.1 Artificial neural networks

Artificial neural networks (introduced in section 2.2.1) are the mathematical models that use a network of "neurons" with activation functions to mimic biological neural networks. An ANN is composed of a large number of free parameters which define the network that connects the "neurons". These trainable parameters are generally not explainable though there are machine learning interpretability methods (Linardatos et al. 2020). However, with them ANNs can approximate the mapping between inputs and outputs. A training loss function reflects how well the predicted output an ANN performs for a given input. Furthermore, an ANN can be trained by gradient decent methods because this loss function is generally differentiable with respect to the network parameters. In recent years, with the developments of parallel hardware, larger/deeper neural networks have been proven to supersede existing methods on various high-level tasks such as object recognition (Krizhevsky et al. 2012). Within computational science, DNNs have also been explored to solve ordinary differential equations and PDEs under both supervised (Chen, Rubanova, et al. 2018; Long et al. 2018) and unsupervised (Han et al. 2018) settings.

We mainly introduce neural networks that are applied in this thesis. Feedforward neural network has no feedback connection which is used in the recurrent neural networks. The deep feedforward neural network is composed of many layers of perceptrons. The network has strong capability to approximate mappings. In the mapping $\boldsymbol{y} = f(\boldsymbol{x}, \boldsymbol{\theta})$, $\boldsymbol{x}$ and $\boldsymbol{y}$ are the input and output tensors of the neural network. $\boldsymbol{\theta}$ is the trainable parameters which adjust the neural network to best approximate the desired mappings. The universal approximation theorem (Cybenko 1989) proves neural networks can approximate arbitrary function to any accuracy provided it has sufficient parameters.

The feedforward neural network can be regarded as a chain of networks and the mapping $f(\boldsymbol{x}) = f^n(\cdots f^2(f^1(\boldsymbol{x})))$ represents the n layers of the network where n is the depth of the model. The first layer of the network is called input layer and the last (nth) is called output layer. The training data only gives the desired input and output therefore other layers are called hidden layers. In each layer the number of neurons is the width of the layer. The neurons in the same layer work in parallel as a function that inputs tensor and outputs a scalar. The neurons are also called perceptrons and feed forward neural networks are also called multilayer perceptrons. In the linear function $f(\boldsymbol{x}) = \boldsymbol{\omega x} + b$, $\boldsymbol{\omega}$ is the weight acts on inputs and b is the scalar bias. To enable neural networks to approximate nonlinear mappings, activition functions are applied on neurons and the function becomes $f(\boldsymbol{x}) = a(\boldsymbol{\omega x} + b)$, where a is the nonlinear activation function. Though a single layer network is sufficient to represent any function, using deep models reduces the total neuron numbers and the generalisation error (Goodfellow et al. 2016). When designing an ideal architecture, experiments are necessary where validation error is monitored.

When a neural network is designed, it is initialised far from representing the desired mappings. Training is the process to fit the network to the mappings or desired tasks. Training can be regarded as an optimization procedure where a chosen cost function should be minimised. There are several ways to define the distance between the model outputs and the desired outputs. The choice of the cost function depends on the models. When the model defines a distribution $p(\boldsymbol{y}|\boldsymbol{x};\theta)$, we use maximum likelihood, for example, negative log likelihood (Bosman and Thierens 2000) (equivalent to cross-entropy likelihood) as cost function. When the model simply outputs statistic $\boldsymbol{y}$ given $\boldsymbol{x}$, we can use mean square error or mean absolute error as cost function. To find the global minimum of the cost function, gradient descend methods (Ruder 2016) are used. The efficient way to find the gradient of the model output $\boldsymbol{y}$ respecting to the trainable parameters $\boldsymbol{\theta}$ is back propagation (Werbos 1990). By the chain rule, information flows from cost function backwards through the network for gradients. During training, rather than feeding the network the entire training data, usually a subset of the data are used and the subset is called a batch. A single pass of the entire training data is called an epoch. The entire data is usually split into training, validation and test dataset. After training, if the model fail to fit the training data, we call it under-fitting. If the model fits the training data well however performs poorly on the validation data, we call it over-fitting. Over-fitting can be controlled by regularisation techniques (Goodfellow et al. 2016).

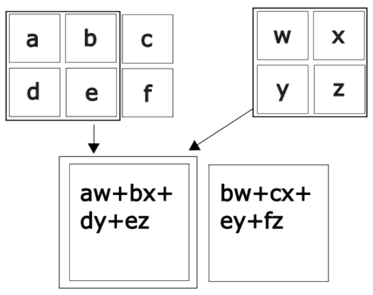


Figure 2.1: An example of convolution, top left is the input, top right is the kernel and bottom is the output

2.2.2 Convolutional neural networks

Convolutional neural networks (CNNs) (Gu et al. 2018) have made great success when applied to problems involving grid topology data, such as those involving images. The concept of convolution originates from the mathematical operation:

$$s(t) = \int x(a)w(t-a)da \tag{2.32}$$

which is denoted as

$$s(t) = (x * w)(t) \tag{2.33}$$

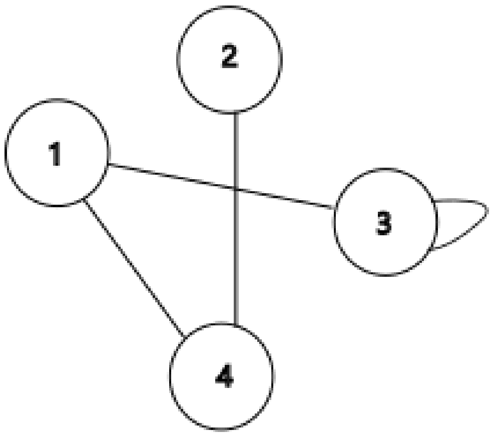
When we use image I as input and undertaking a convolution on 2 axis by 2 dimensional kernel K,

$$s(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m)(j-n) \tag{2.34}$$

Where i, j are the horizontal and vertical index and variable of summation m and n depends on the kernel size. Many deep learning libraries implement cross-correlation

$$s(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i+m)(j+n) \tag{2.35}$$

but call it convolution where the difference is the flipped kernel. Figure 2.1 is a demonstration of a 2x2 convolution. In each layer of fully connected networks, every output unit interacts with every input unit. However in a single layer of convolutional networks, because the kernel is generally much smaller than the input, every output only interacts with a small number of input units. We call this sparse interactions which saves memory and is statistical efficient. Another



0	0	1	1
0	0	0	1
1	0	2	0
1	1	0	0

Figure 2.2: An undirected graph with its adjacency matrix, if a vertex connects with another vertex, the corresponding entry of the adjacency is 1. When a vertex connects with itself, the corresponding entry is 2. The adjacency matrix of an undirected graph is symmetric.

property of convolutional networks is parameter sharing where the kernel is shared at different position of inputs. Convolutional networks usually contains not only convolutional layers but also pooling layers (Gholamalinezhad and Khosravi 2020). Pooling usually reduce the size by outputting a summary stastic of nearby inputs.

2.2.3 Graph convolutional networks

Classic convolution are based on grid data. However, there are many applications for which data is non-Euclidean structured. For example, social network and traffic networks have data defined on graphs. A graph is defined as a set of vertices and a set of edges. An adjacency matrix indicates the connections between vertices. Figure 2.2 demonstrates an undirected graph and its adjacency matrix. The matrix is symmetric because the graph is undirected. There is an entry of 2 in the (3,3) position because a edge connect node 3 at two sides. Graph neural networks (Zhou et al. 2020) are deep learning based methods operating on graph domain, such as graph convolutional networks (S. Zhang et al. 2019), graph attention networks (Lee et al. 2019, Velickovic et al. 2017) and graph recurrent networks (Scarselli et al. 2008). Graph convolution generalise the convolution operation to the graphs, with two categories: spectral (Bruna et al. 2013) and spatial (Duvenaud et al. 2015) approaches.

Figure 2.3 from (Zhou et al. 2020) demonstrates the rich development of graph neural networks using convolution operations. In each categories models are ordered by published time. General framework are proposed aiming to integrate different models into one single framework. In this

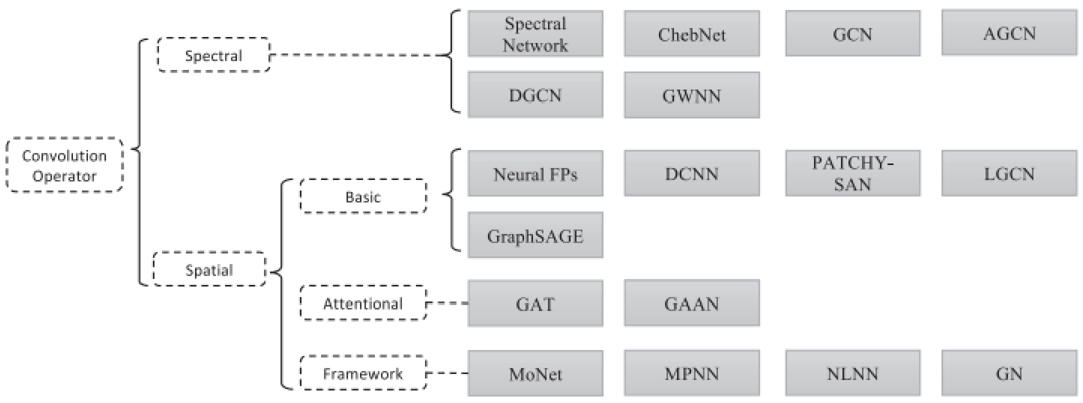


Figure 2.3: Overview of graph convolution networks (Zhou et al. 2020), Graph neural networks (GNNs) are neural models that capture the dependence of graphs via message passing between the nodes of graphs. Convolution is the mostly used operator for GNN models. The main idea of convolution operators is to generalize convolutions from other domain to the graph domain. The two categories of graph convolution are spectral and spatial approaches.

section we only introduce a spectral method call graph convolutional networks (GCN) (Kipf and Welling 2017). Consider spectral convolution defined as the multiplication of a signal x with a filter $g_\theta = diag(\theta)$ where trainable parameter θ is in the Fourier domain.

$$g_\theta * x = U g_\theta U^T x \tag{2.36}$$

Where U is the matrix of eigenvectors of the normalised graph Laplacian $L = I_N - D^{-\frac{1}{2}} A D^{-\frac{1}{2}} = U \Lambda U^T$. A is the adjacency matrix and for diagonal matrix D $D_{ii} = \sum_j A_{ij}$. Λ is diagonal of the eigenvalues of L. Eigendecomposition of L for large graph is too expensive. Hammond et al. 2011 suggests using truncated expansion in terms of Chebyshev polynomials $T_k(x)$ up to kth order to approximate g_θ:

$$g_{\theta'} * x \approx \sum_k \theta'_k T_k(\tilde{L}) x \tag{2.37}$$

θ'_k is the trainable weight. The Chebyshev polynomials are defined as $T_k(x) = 2xT_{k-1}(x) - T_{k-2}(x)$, with $T_0(x) = 1$,$T_1(x) = x$. $\tilde{L} = \frac{2}{\lambda_m ax} L - I_N$, λ is the largest eigenvalue of L. The expression is k localised i.e. it depends only on the nodes that are maximum kth steps away from the central node.

2.3 Machine learning for PDEs

Machine learning has been applied with great success in tasks in computer vision and natural language processing which used to be challenging for computers. These tasks are data rich however computational models are frequently too expensive to provide more than a limited amount of labeled data. On the contrary, many fundamental research disciplines are model rich and classical computational methods are successful, such as computational physics. Recently, we have seen exciting joint research of these fields with machine learning such as machine learning with fluid mechanics (Brunton et al. 2020), chemistry, material science (Choudhary and al. 2022) and biology. Mathematics models used to play the important role in these fundamental disciplines. In this thesis we are particularly interested in PDEs and look forward to seeing machine learning help PDEs solving. The key motivation of this study is improving the efficiency of PDE solving where deep learning acts as either an accelerator or replacer to the conventional solver such as the finite element method. There is vast work on machine learning based PDE solver and below we introduce some representable examples in categories of supervised learning and unsupervised learning. Then we introduce the function of machine learning in aiding solving PDEs especially in mesh generation.

2.3.1 Machine learning based PDE solver

In supervised learning, the machine learning model requires large volume of data. For PDE driven fields such as computational physics, the data comes from sensors or simulation via the conventional methods such as finite element, finite volume, finite difference and spectral methods. Still, obtraining big data is expensive and difficult, which constrained the development of machine learning in scientific problems. Physics informed machine learning (Karniadakis et al. 2021) tackle this by training from additional information obtained by enforcing physical laws. Physics informed neural networks (PINN) (Raissi et al. 2019,S. Cai et al. 2021) solve forward and inverse problem in the same framework. The parameters of the networks are trained by minimising a loss function that depends on the differential equation, boundary condition and eventually some known data. Physics reinforced neural network (W. Chen et al. 2021) as an efficient reduced order model, is more accurate than PINN for complex non-linear problem. The full-order model is replaced with a low-dimensional reduced-order model to efficiently resolve the main dynamics of the problem. Bar-Sinai et al. 2019 generate a training set of high-resolution data. The neural network estimates spatial derivatives to allow the use of resolution 4x to 8x coarser than standard finite difference method when accuracy is ensured. PDENet (Long et al. 2018) approximates differential operations by convolutions with properly constrained filters. It can uncover the hidden equation of the observed dynamics, and predict the dynamical behavior for a relatively long time.

Unsupervised learning does not require labeled training data. Machine learning takes place of the PDE solver. It is even possible to tackle problems that conventional methods cannot solve, such as high-dimensional problems (Han et al. 2018). Different from (Lagaris et al. 1998) using equidistant points on grid, Deep Galerkin Method (DGM)(Sirignano and Spiliopoulos 2018) is meshfree, which is claimed the key to solving high-dimensional PDEs. The solution is approximated by the neural networks rather than a combination of basis function. The neural networks are trained on random sampled time and points in space. The cost function measures how well the approximated solution satisfies the differential operator, boundary condition and initial condition. The risk of this method is the solution may converge to local minimum, which is generally true for non-convex optimisation. Geneva and Zabaras 2020 use a deep auto-regressive CNN to learn and surrogate model the dynamics of transient PDEs. Training used physics constrained deep learning where equation of the system are used to formulate a loss function. The model is trained without training data.

Machine learning based PDE solver has made great progress in the last few years. However, we rarely find machine learning PDE solver other than reduced order method implemented in engineering practice. The reason might be:

1. lack of data
2. lack of guarantee of accuracy
3. difficult in generalization
4. low and unpredicted efficiency

Item 2 is especially important in many cases where the accuracy of the solution is critical. Item 1 constrains supervised learning methods where the performance of models usually depends on the amount of training data. Item 2 is common for both supervised and unsupervised learning. The methodology such as the choice of activation function in PINN relates to the accuracy. In unsupervised learning such as Deep Ritz method Yu et al. 2018 the solution sometimes converges to local minima. Item 3 is a problem for supervised learning and item 4 is a problem for unsupervised learning where Deep Ritz method is much slower than FEM.

2.3.2 Machine learning aided PDE solver

Despite the success in high-dimensional PDEs, ML based PDE solvers currently are difficult to replace the conventional solvers in fields such as computational physics. We therefore turn our interest into a more conservative approach, using ML to aid rather than to replace the traditional PDE solvers. For example, Kochkov et al. 2021 use ML for interpolation and correction in finite volume method. It achieve the same accuracy when using much coarser resolution and perform the computation 40 to 80X faster.

Now that we are aware of the lack of guaranteed accuracy of machine learning PDE solver, we give machine learning the jobs where error is tolerable. Preconditioning for the solver (Götz and Anzt 2018) and mesh generation give ML great opportunities. We consider ML related mesh processing in detail.

2.4 Machine learning for mesh processing

Despite more than one hundred papers on Intelligent Mesh Generation (Lei et al. 2023), the vast majority of them are not for PDE solving. Such as generating surface mesh for animation and visualisation (Wang and Zhang 2022, Song et al. 2021), and for clinical applications (Kong et al. 2021). In this section we only introduce mesh generation for PDE solving purpose. The concept of mesh generation involves mesh processing techniques such as mesh generation, mesh adaptation and mesh quality enhancement. The majority of ML aided mesh processing are in category of mesh generation and adaptation. We have also seen the implementation of ML in mesh quality assessing (Iqbal and Carey 2002, Chen, Liu, et al. 2020) and mesh partitioning for parallel computing (Bahreininejad et al. 1996).

2.4.1 Machine learning for mesh generation

The application of machine learning to the mesh generation starts from the use of self-organizing neural networks (Chang-Hoi et al. 1991, Manevitz, Yousef, et al. 1997). At each iteration, the neurons compete against each other and the only the weights (point coordinates) linked with the winner are updated. The meshing process starts from a fix grid topology and during unsupervised learning, the locations of vertices change. To overcome the fixed grid size, Let-it-grow neural networks (Alfonzetti et al. 1996) are proposed, where not only moving nodes but also node insertion are applied. The neural network in self-organizing and let it grow both differs from modern feed forward neural networks in aspects such as learning is not done by minimising cost function. For self-organizing learning it is done by a selection of a node by a competition and moving the node.

More recent work follows the latest developments of neural networks. Papagiannopoulos et al. 2021 uses three feed forward neural networks to predict the number of inner vertices of the mesh, their location and the entries of the connection table. MGNet (X. Chen et al. 2022) is a novel differential structured mesh generation method based on unsupervised neural network. Mehendale and Soman 2021 use LSTM to generate coordinates of mesh grid points then employ Delauny triangulation. FreeMesh-RL (Pan et al. 2023) formulates the mesh generation as a Markov decision problem, using a soft actor-critic approach to automatically generate mesh by reinforcement learning. These works generally produce good quality meshes however lack evidence to demonstrate neural networks methods are superior than classical mesh generators. In this thesis we do not develop mesh generator but use conventional mesh generators guided by deep learning.

2.4.2 Machine learning for mesh adaptation

In the context of using ML to guide high-quality non-uniform mesh generation there are several prior research. In Dolšak et al. 1994, for example, early knowledge-based approaches were considered, though with limited success. Time-dependent remeshing is studied by Manevitz, Bitar, et al. 2005, where an NN is used to undertake time-series predictions that identify areas of greater (and less) refinement at different times, though on a domain with simple geometry. In (Chedid and Najjar 1996, Dyck et al. 1992) NNs were applied successfully to generate high quality finite element meshes for elliptic PDEs, however the input vectors are highly problem-dependent: requiring specific *a priori* knowledge of the geometry being considered. They require *a priori* knowledge such as the mesh density depends on the distance to certain position of the magnetic device. The challenge of using DNNs to generate pseudo-optimal FE meshes on quite general geometries was first considered in MeshingNet (Z. Zhang, Y. Wang, et al. 2020) for selected two-dimensional PDEs. MeshingNet3D (Z. Zhang, Jimack, et al. 2021) extends these ideas to problems in three dimensions, to consider PDE systems with rich variation in geometry, boundary conditions and material properties. These works are described in detail in Chapter 3 and 4 respectively.

Since the publication of MeshingNet we have seen renewed interest in apply machine learning for mesh adaptation. Some work uses machine learning as adaptive mesh refinement strategies. Gillette et al. 2022 and Yang et al. 2023 regard adaptive mesh refinement as a Markov decision process and use reinforcement learning for mesh adaptation. It is claimed reinforcement learning policies can outperform widely used ZZ error estimation. Bohn and Feischl 2021 use recurrent neural network as optimal mesh refinement strategies in element mark method, which can be applied on problems with no optimal adaptive strategy known yet.

For applications, we have seen the use of machine learning based mesh refinement on problems such as elasticity (Chan et al. 2022) and advection-diffusion (Falini and Sampoli 2021) problems. For adaptive mesh generation, computation fluid dynamics (CFD) is an important problem. Huang, Krügener, et al. 2021 used unsupervised trained U-net to predict static optimal CFD mesh. Ojha 2022 use flow parameters as inputs to predict optimal static mesh for aeroelastic simulation. Tingfan et al. 2022 use a regression model to predict flow fluid and improve and efficency and accuracy of the solution in the process of moving mesh. Fidkowski and Chen 2021 use NN to predict the desired element aspect ratio from readily accessible features of the primal and adjoint solutions. The challenge now is how to use machine learning to learn the dynamics of the mesh since optimal meshes of CFD are generally time dependent.

2.5 Mean value coordinates

Mean value coordinates (MVC) (Floater 2003) is a generalisation of barycentric coordinates which allows a vertex in a planar triangulation to be expressed as a convex combination of its neighbouring vertices. The coordinates are motivated by the Mean Value Theorem for harmonic functions and can be used to simplify and improve methods for parameterization and morphing.

Here we illustrate MVC in 2D. Let $v_0, v_1, ..., v_k$ be points in the plane with $v_1, ..., v_k$ arranged in an anticlockwise ordering around v_0 as in Fig 2.4. The points $v_1, ..., v_k$ form a star-shaped polygon with v_0 in its kernel. Our aim is to study sets of weights $\lambda_1, ..., \lambda_k \geq 0$ such that

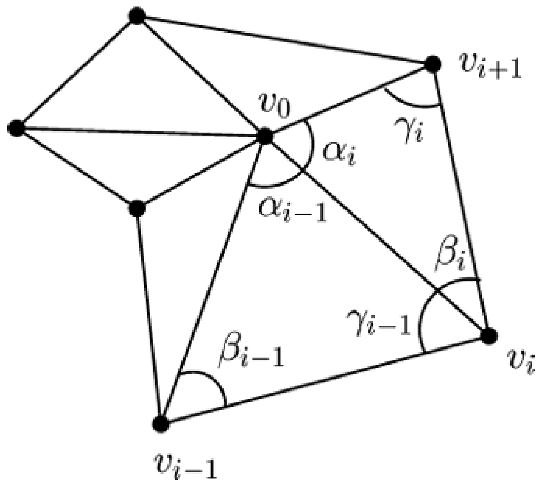


Figure 2.4: Star-shaped polygon, we care the MVC of point v_0.

$$\sum_k \lambda_i v_i = v_0 \tag{2.38}$$

$$\sum_k \lambda_i = 1 \tag{2.39}$$

In the simplest case $k = 3$, the weights $\lambda_1, \lambda_2, \lambda_3$ are uniquely determined by Equation 2.38 and 2.39 alone; they are barycentric coordinates, and they are positive. This motivates calling any set of non-negative weights satisfy Equation 2.38 and 2.39 for general k. For MVC, the weights

$$\lambda_i = \frac{w_i}{\sum_{j=1}^{k} w_j}, w_i = \frac{tan(\alpha_{i-1}/2) + tan(\alpha_i/2)}{||v_i - v_0||} \tag{2.40}$$

These weights can be derived from an application of the mean value theorem for harmonic functions, which suggests calling them mean value coordinates. They obviously depend smoothly on the points $v_0, v_1, ..., v_k$.

2.6 Summary

The finite element method is a successful computational method for solving PDEs which is the base for physical simulations. With the fast development of machine learning and intelligent systems we see new opportunities and challenges for solving PDEs. Supervised and unsupervised trained artificial neural networks have shown strong capability in replacing the conventional PDE solvers and solving high dimensional PDEs. However we have limited confidence in applying these novel methods in engineering practice, since they generally lack guarantee in accuracy. The solution may has significant error or converge to local minima. We therefore turn to seek using machine learning to aid the conventional solvers. Mesh generation is a complex however error tolerable process where the mesh size distribution does not have to be highly precise. We have seen renewed interest in mesh generation and adaptation by neural networks. And now we face the new challenges for machine learning such as:

1. dynamic adaptive mesh for CFD
2. geometry partitioning and mapping for structured mesh generation
3. mesh generation for fluid structure interaction and multi-physics simulation
4. error estimation and 3D anisotropic mesh generation

Chapter 3

Mesh adaptation for 2D static PDEs

We introduce a novel approach to automatic unstructured mesh generation by using machine learning to predict an optimal mesh element size distribution for a previously unseen problem. The framework that we have developed is based around training an ANN to guide standard mesh generation software, based upon a prediction of the required local mesh density throughout the domain. We describe the training regime that is proposed, based upon the use of *a posteriori* error estimation, and discuss the topologies of the ANNs (the way the neurons are connected) that we have considered. We then illustrate performance using two standard test problems, a single elliptic PDE and a system of PDEs associated with linear elasticity. We demonstrate the effective generation of high quality meshes for arbitrary polygonal geometries and a range of material parameters, using a variety of user-selected error norms.

3.1 Introduction

Mesh generation is a critical step in the numerical solution of a wide range of problems arising in computational science. The use of unstructured meshes is especially common in domains such as computational fluid dynamics (CFD) and computational mechanics, but also arises in the application of finite element (FE) and finite volume (FV) methods for estimating the solutions of general partial differential equations (PDEs): especially on domains with complex geometries. The quality of the FE/FV solution depends critically on the nature of the underlying mesh. For an ideal mesh the error in the FE solution (as discussed in chapter 2 we focuson the FE method (FEM) in this thesis) will be distributed equally across the elements of the mesh, implying the need for larger elements where the local error is small and smaller elements where the local error is large. This equidistribution property tends to ensure that a prescribed global error tolerance (i.e. an acceptable error between the (unknown) true solution and the computed FE solution) can be obtained with the fewest number of elements in the mesh (Dörfler 1996; Stevenson 2007). This is a desirable feature since as discussed in Chapter 2 the computational work generally grows at least linearly with the number of elements (in some special cases, this can be linear (Notay 2010)).

The conventional approach to obtain high quality unstructured meshes involves multiple passes, where a solution is initially computed on a relatively coarse uniform mesh then a post-processing step, known as *a posteriori* error estimation (as introduced in section 2.1.4), is undertaken. This typically involves solving many auxiliary problems (e.g. one per element or per patch of elements) in order to estimate the local error in the initial solution. These local errors can be combined to form an overall (global) error estimate but they can also be used to determine where the local mesh density most needs to be increased (mesh refinement), and by how much, and where the local mesh density may be safely decreased (mesh coarsening), and by how much. A new mesh is then generated based upon this *a posteriori* error estimate and a new FE solution is computed on this mesh. A further *a posteriori* error estimate may be computed on this new mesh to determine whether the solution is satisfactory (i.e. has an error less than the prescribed tolerance) or if further mesh refinement is required.

A necessary requirement for efficient *a posteriori* error estimators is that they should be relatively cheap to compute. For example, the approaches of Ainsworth and Oden 1997; Apel, Grosman, et al. 2004; Bank and Weiser 1985 each solve a supplementary local problem on each finite element in order to estimate the 2-norm of the local error from the local residual. Alternatively, recovery-based error estimators use local "superconvergence" properties of FE to estimate the energy norm of the local error based purely on a locally recovered gradient: for example the so called ZZ estimation (Zienkiewicz and Zhu 1992). Here the difference between the original gradient and a patch-wise recovered gradient indicates the local error. In the context of linear elasticity problems, the elasticity energy density of a computed solution is evaluated at each element and the recovered energy density value at each vertex is defined to be the average of its adjacent elements. The local error is then proportional to the difference between the recovered piece-wise linear energy density and the original piece-wise constant values.

In this chapter we exploit data-driven methods to improve the efficiency of non-uniform mesh generation compared with existing approaches. The core of non-uniform mesh generation is to find an appropriate mesh density distribution in space and perhaps also time. If a model can learn from the data, it no longer needs an FE solution and error estimator to predict a good mesh density distribution, but instead can rely on learning from a set of similar problems for prediction.

In the work reported here we propose a DNN model, MeshingNet, to learn from the *a posteriori* error estimate on an initial(coarse) uniform mesh and predict non-uniform mesh density for refinement, without the need for (or computational expense of) solving an FE system or computing an *a posteriori* error estimate. MeshingNet is trained using an accurate error estimation strategy which can be expensive but is only computed offline. Hence, the mesh generation process itself is extremely fast since it is able to make immediate use of standard, highly-tuned, software (in our case Shewchuk 2002) to produce a high quality mesh at the first attempt (at similar cost to generating a uniform mesh with that number of elements). As already outlined, it is not our intention in this work to use deep learning to solve the PDEs directly: instead we simply aim to provide a standard FE solver with a high quality mesh, because we can provide more reliable predictions in this way, based upon the observation that a greater variation in the quality of predictions can be tolerated for the mesh than for the solution itself. For example, in an extreme case where the DNN predicts a constant output for all inputs, the result would be a uniform mesh (which is tolerable) however such a poor output would be completely unacceptable in the case where the network is used to predict the solution itself. Both our work and PINN (physics informed neural network) use supervised learning however we do not use NN to solve the PDE directly and our NN ignores physics laws.

Formally, we propose, what is to the best of our knowledge, the first DNN-based predictor of *a posteriori* error, that can:

1. efficiently generate non-uniform meshes with a desired speed
2. seamlessly work with existing mesh generators
3. generalize to different geometric domains with various governing PDEs, boundary conditions (BCs) and parameters.

The remainder of this chapter is structured as follows. In the next section we describle our proposed deep learning algorithm. This is not the first time that researchers have attempted to apply ANNs to mesh adaptation. However, previous attempts have been for quite specific problems : Chedid and Najjar 1996; Lowther and Dyck 1993 and have therefore been able to assume substantially more *a priori* knowledge than our approach. Consequently the novelty in Sec. 3.2 comes through both the generality of the approach used in formulating the problem (i.e. generality of the inputs and outputs of the DNN) as well as the network itself. In Sec. 3.3, we demonstrate and assess the performance of our approach on two standard elliptic PDE problems. These tests allow us to account for variations in the PDE system, the domain geometry, the BCs, the physical problem parameters and the desired error norm when considering the efficacy of our approach. Finally, in Sec. 3.4, we discuss our plans to further develop and apply this deep learning approach.

3.2 Proposed methods

3.2.1 Overview

We consider a standard setting where the FEM is employed. Given a geometry and a mesh generator, a low density uniform mesh (LDUM) can be easily computed, then refined non-uniformly based on the *a posteriori* error distribution, for better accuracy. We propose a DNN-based supervised learning method to replace the time consuming *a posteriori* error estimation.

Given a family of governing PDEs and material parameters (for elasticity), we assume that there is a mapping

$$F : \Gamma, B, M, X \to A(X) \tag{3.1}$$

that can be learned, where Γ is a collection of domain geometries (x,y coordinates of the vertices of the polygons), B is a set of BCs, M is a set of PDE parameters (e.g. material properties), $x \in X$ is a location in the domain and $A(X)$ is the target area upper bound (area of triangle elements should be smaller than this value at the location X) distribution over the whole domain. To represent an interior location, we use Mean Value Coordinates (Floater 2003) because they provide translational and rotational invariance with respect to the boundary. Given Γ, B, M and X, we aim to predict $A(X)$ quickly. The mapping F is highly non-linear and is therefore learned by our model, MeshingNet. Under the supervised learning scheme, we first build up our training data set by computing high-accuracy solutions (HASs) on high-density uniform

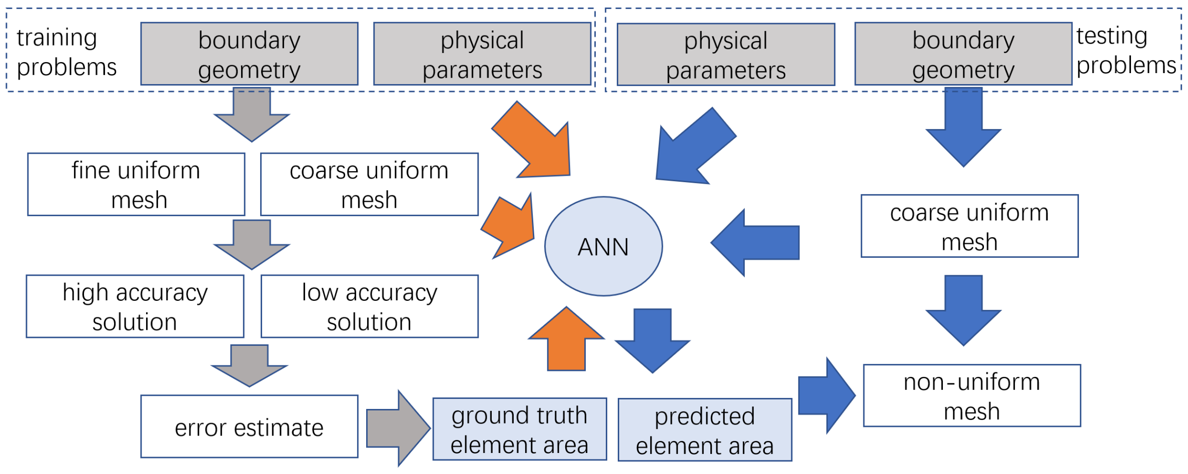


Figure 3.1: A high-level diagram of MeshingNet workflow showing the ANN alongside the training regime (left) and testing regime (right). Grey, orange and blue arrows represent data generation, training and testing processes. For each problem we only use one fine mesh and one coarse mesh.

meshes (HDUMs) using a standard FE solver. The same computation is also done on LDUMs to obtain lower accuracy solutions (LAS). Then an *a posteriori* error distribution $E(X)$ is computed based upon interpolation between these solutions. According to $E(X)$, we compute $A(X)$ for refinement. The training data is enriched by combining different geometries with different parameters and BCs. Next, MeshingNet is trained, taking as input the geometry, BCs and material properties, with the predicted local area upper bound $A(X)$ as output. After training, MeshingNet is used to predict $A(X)$ over a new geometry with a LDUM. The final mesh is generated either by refining the LDUM non-uniformly or using it to generate a completely new mesh guided by the predicted target local area upper bound. Figure 3.1 illustrates the whole workflow of our mesh generation system.

The approach that we propose has a number of components that are not fixed and may be selected by the user. MeshingNet is agnostic about both the mesh generator and the particular FE/FV solver that are used. It is designed to work with existing methods. Furthermore, the *a posteriori* error can be computed using any user-defined norm (in this chapter we consider L1 and energy norms respectively in our two validation tests). Some specific examples of governing equations, geometries, boundary conditions and material parameters are illustrated in the evaluation section. Prior to this however we provide some additional details of the components used in this chapter.

3.2.2 Component Details

Mesh generation

All meshes (LDUM, HDUM and the refined mesh) are created using the software *triangle* which conducts Delaunay triangulation. *Triangle* (Shewchuk 2002) reads a planar graph, representing the boundary of a two-dimensional polygonal domain as its input and the user may define a single maximum element area across the entire polygon when a uniform mesh is desired. To refine a mesh non-uniformly the user specifies, within each coarse element, what the element area upper bound should be after refinement. *Triangle* ensures a smooth spatial variation of the element size. The refinement is not nested since, although it does not eliminate the pre-existing vertices of the original mesh, it does break the original edges.

Mesh refinement via error estimation

Broadly speaking, the finer the mesh is, the FE result is expected closer to the ground truth. We regard the HAS (High resolution results) as the ground truth by ensuring that HDUMs are always significantly finer than LDUMs. Linear interpolation is used to project the LAS to the fine mesh to obtain LAS*, where LAS* has the same dimension as HAS. An error estimate approximates the error E by comparing LAS* and HAS in the selected norm, on the HDUM. After that the error is projected back to the original LDUM. Rather than defining the average mesh area, the target area upper bound for the refined mesh within each LDUM element is then computed to be inversely correlated with E. We are able to but do not predict the *a posteriori* error estimate because it is the element area that directly controls mesh adaptivity. In this chapter, we select $K/E(x_i)^\alpha$ as the area upper bound for element number i of the LDUM (to be refined), where x_i is the center of the i th element, K and α determines the overall target element number and, in the examples given here, we always choose $\alpha = 1$. If choose $\alpha = 2$ the nonuniform pattern would be more aggresive. By varying K appropriately it is possible to adjust the refined mesh to reach a target total number of elements.

MeshingNet model and training

As outlined in the overview, for a given PDE system, MeshingNet approximates the target local element area upper bound A(x) at a given point within a polygonal domain based upon inputs which include: the coordinates of the polygon's vertices, key parameters of the PDE and the mean value coordinates of the specified point (mean value coordinates parameterizes a location within a 2D polygon as a convex combination of polygon vertices). Two types of DNNs are considered: a fully connected network (FCN) and two residual networks (ResNets). The dimensions of our FCN layers are X-32-64-128-64-32-8-1 (where X represents the dimension of the input and is problem-specific) and each hidden layer uses rectified linear units (ReLU) as the activation function. To further improve and accelerate training, two ResNets are also experimented with to enhance FCN (see Fig. 3.2) ResNet1 enhances FCN by adding multiple residual connections from the first hidden layer to the output of the last one. Note that residual connections can help to resolve the vanishing gradient problem in deep networks and improve training (Veit et al. 2016). ResNet2 enhances FCN by adding multiple residual connections. This is inspired by recent densely connected convolutional networks (Huang and al. 2017) which shows superior data-fitting capacity with a relatively small number of parameters. The training data set samples over geometries, BCs and parameter values. Each geometry with fixed BCs and parameters uniquely defines a problem. In a problem, each LDUM element centroid (represented by its mean value coordinates) and its target $A(x)$ forms an input-output training pair. We randomly generated 3800 problems of which 3000 are used for training and 800 for testing (because each LDUM contains approximately 1000 elements, there are over 3 million training pairs). We then use stochastic gradient descent, with a batch size 128, to optimize the network. We use mean square error as the loss function and *Adam* (Kingma and Ba 2015) as the optimizer. The implementation is done using *Keras* (Chollet et al. 2015) on *Tensorflow* (Martın and al. 2015) and the training is conducted on a single *NVIDIA Tesla* K40c graphics card.

Guiding mesh generation via MeshingNet

After training, the network is able to predict the target distribution $A(x)$ on a previously unseen polygonal domin. Given a problem, a LDUM is first generated; next, MeshingNet predicts the local target area upper bound, $A(x)_i$ at the centre, x, of the i th element; *Triangle* then refines the LDUM to generate the non-uniform mesh based upon $KA(x)_i$ within each element of the LDUM (to be refined). Optionally, this last step may be repeated with an adjusted value of K to ensure that the refined mesh has a desired total number of elements (this allows an automated approximation to “the best possible mesh with X elements”).

Error norms

Error estimation provides means of quantifying both the local and the global error in an initial solution. However the precise magnitude and distribution of the error depends on the choice

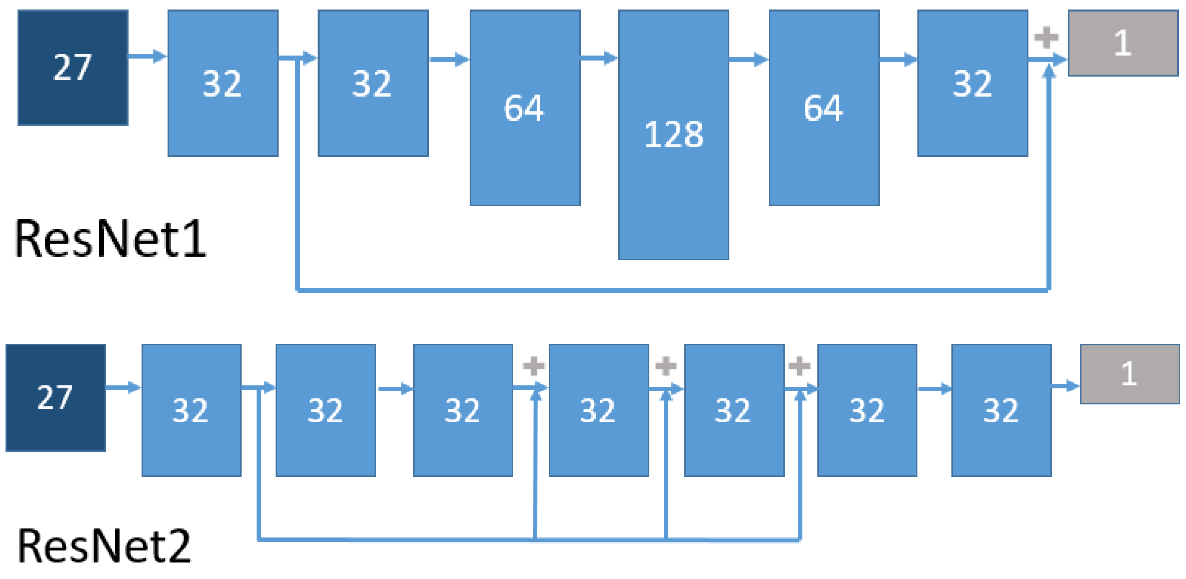


Figure 3.2: Our two residual networks, illustrated with 27 input parameters. ResNet1 is a modification of the FCN with the output of the first hidden layer added to the output of the last hidden layer. ResNet2 has all hidden layers of the same dimension and the output of the first hidden layer is added to outputs of the three front hidden layers.

of the norm used to compute the difference between the LAS and the HAS. Different norms lead to different non-uniform meshes. Consequently, for any given PDE system, a single norm should be selected in order to determine A from Γ, B and M. The appropriate choice of norm is a matter for the user to decide, similar to the choice of specific *a posteriori* error estimate in the conventional adaptive approach. In the following section two different norms are considered for the purposes of illustration.

3.3 Validation results

We now assess the performance of the 2D static mesh generation system through two computational example: a single PDE, for which we consider only the effect of the domain geometry on the optimal mesh; and a system of PDEs, for which we consider the influence of geometry, boundary conditions and material parameters on the optimal mesh.

3.3.1 2D Poisson's equation

We solve Poisson's equation ($\nabla^2 u + 1 = 0$) on a simply connected polygon Ω with boundary $\partial\Omega$ (on which $u = 0$). The polygons in our data set are all octagons, generated randomly, subject to constraints on the polar angle between consecutive vertices and on the radius being between

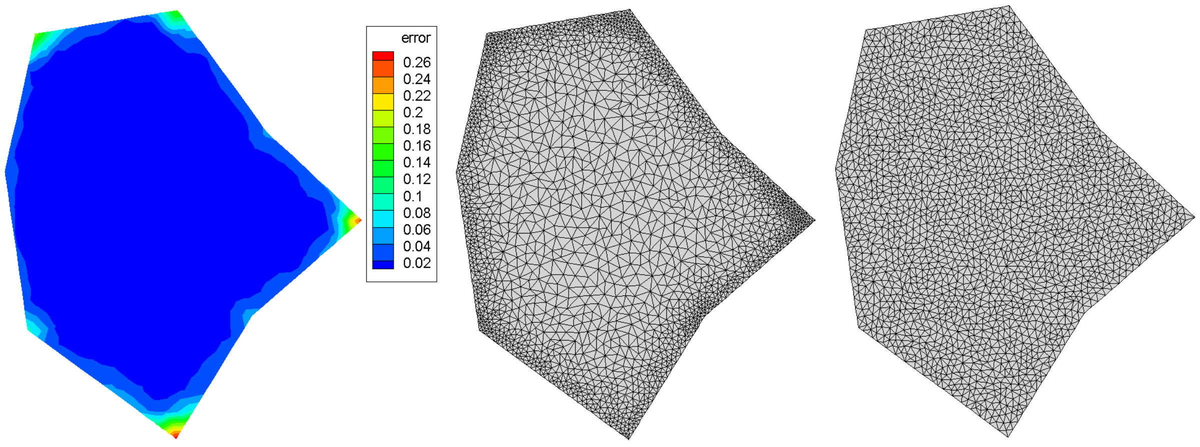


Figure 3.3: For Poisson's equation: the L1 error distribution for the LAS (left); the 4000 elements mesh generated by *Triangle* under the guidance of MeshingNet (middle); and the uniform mesh with 4000 elements generated by *Triangle* (right).

100 to 200 (so the polygons are size bounded). The L1 norm relative error estimate is

$$E = \left| \frac{u^{LAS} - u^{HAS}}{u^{HAS}} \right| . \tag{3.2}$$

Where u^{LAS} and u^{HAS} come from the coarse and fine mesh in Fig 3.1. As expected, in Figure 3.3 (which shows a typical test geometry), the mesh generated by MeshingNet is dense where the error for the LAS is high and coarse where it is low. Figure 3.4 quantifies the improvement of MeshingNet relative to an uniform mesh (Figure 3.3 right) by showing, for the entire test data set, the error distributions of the computed FE solutions (relative to the ground truth solution) in each case. We only apply a fully connected neural network with 6 hidden layers and each layer has 32, 32, 64, 128, 64, 32 neurons.

3.3.2 2D Linear elasticity

We solve 2D plane stress problems on a set of *different* polygons (6-8 edges). Each polygon edge is associated with one of three possible BCs. BC1: zero displacement; BC2: uniformly distributed pressure or traction (with random amplitude up to 1000); and BC3: unconstrained. For different geometries, we number the vertexes and edges anti-clockwise with the first vertex always on the positive x-axis, without loss of generality. To get a combinations of BCs, we always apply BC1 on the first edge, BC2 on the fourth and fifth edges and BC3 on the rest. We also allow different (homogeneous) material properties: density up to a value of 1 and Poisson's

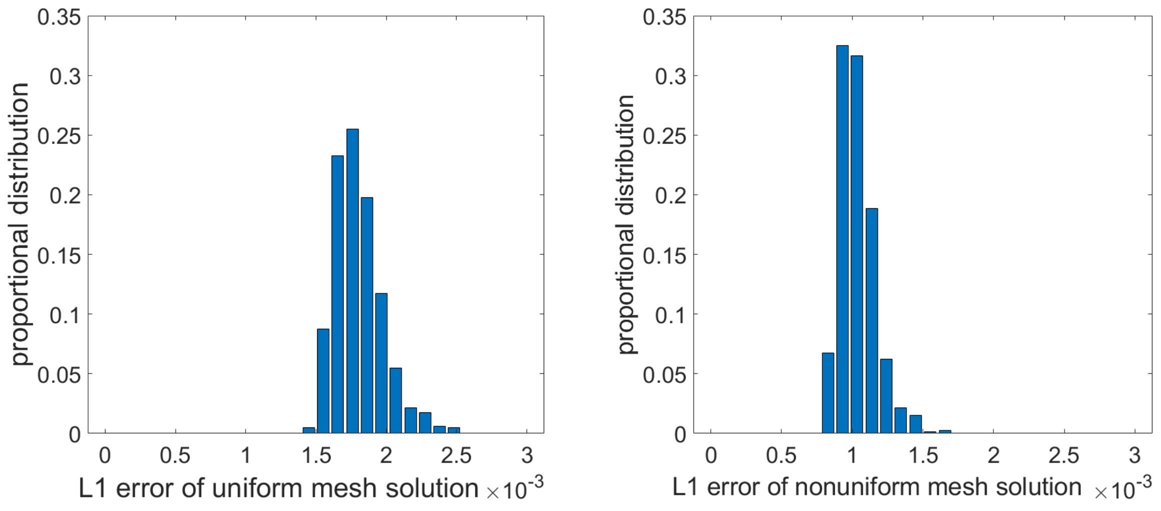


Figure 3.4: For Poission's problem, L1 error distribution on uniform meshes (4000 elements) and MeshingNet meshes (4000 elements). Each bar shows the proportion of test data meshes whose FE solution error is in the range of the bar: the uniform meshes give FE errors between 0.0015 to 0.0025, whilst the MeshingNet meshes give FE errors between 0.0007 and 0.0015.

ratio between 0 to 0.48. The error approximation uses energy norm

$$E = (\boldsymbol{\epsilon}^L - \boldsymbol{\epsilon}^H) : (\boldsymbol{\sigma}^L - \boldsymbol{\sigma}^H) \tag{3.3}$$

where $\boldsymbol{\epsilon}^L$ and $\boldsymbol{\epsilon}^H$ are strains of LAS* and HAS, $\boldsymbol{\sigma}^L$ and $\boldsymbol{\sigma}^H$ are stresses of LAS* and HAS. This is the "natural norm" for this problem since the PDEs are the Euler-Lagrange equations for the minimization of the following energy functional:

$$Ep = \int \frac{1}{2}\boldsymbol{\epsilon} : \boldsymbol{\sigma} - \boldsymbol{F} \cdot \boldsymbol{u} d\Omega - \int \boldsymbol{\sigma} \cdot \boldsymbol{u} d\Gamma \tag{3.4}$$

where $\boldsymbol{F}$ is the body force and $\boldsymbol{u}$ is the displacement. Due to the linearity of the problem, the relative accuracy of two FE solutions may be determined equivalently by which has the lower error in the energy norm or which has the lower total potential energy (we exploit this in our validation below).

There are 27 dimensions in MeshingNet's input: 16 for the polygon vertices, 8 for the mean value coordinates of the target point, and 1 each for the traction BC magnitude, density and Poisson's ratio. If the polygon has less than 8 faces, we allocate the additional vertices onto the last face. We train FCN, Resnet1 and Resnet2 for 50 epochs, each taking 142, 134 and 141 minutes respectively. Figure 3.5 shows that the training processes all converge. After

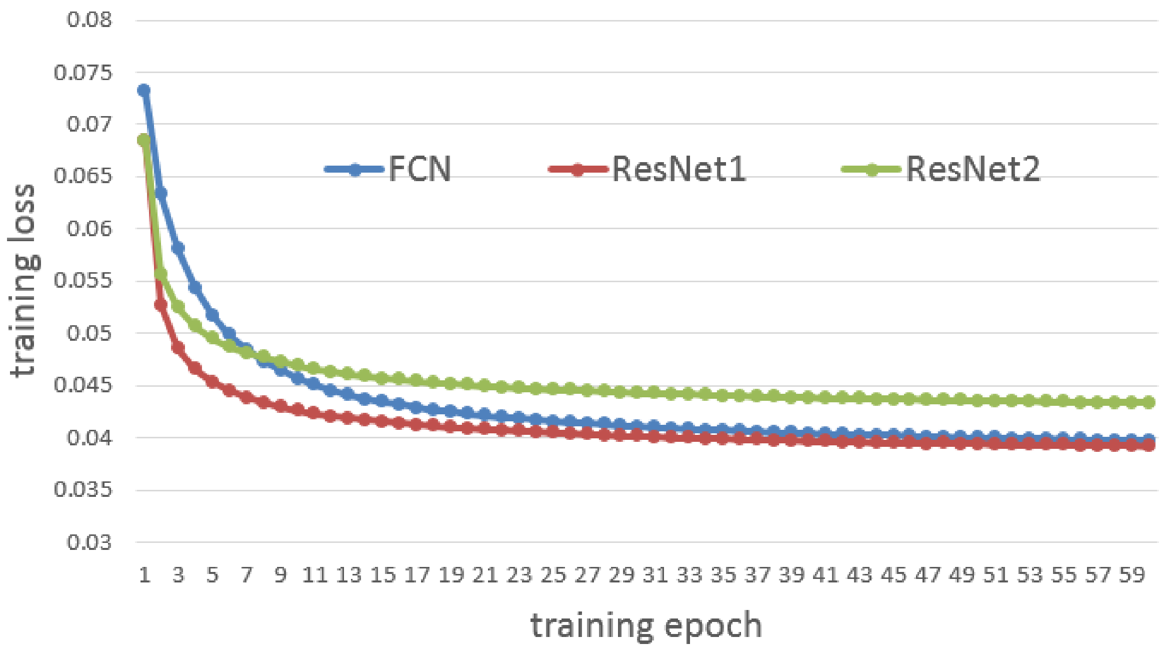


Figure 3.5: L2 training loss on elasticity training dataset during 60 training epochs. Three curves representing FCN (blue), ResNet1 (red) and ResNet2 (green) converge individually.

training, predicting the target A (on all LDUM elements) for one problem takes 0.046 seconds on average, which is over 300 times faster than using the *a posteriori* error method that generates the training data set.

Figure 3.6 shows a comparison of FE results computed on MeshingNet meshes, uniform meshes of the same number of elements (4000 elements) and non-uniform meshes (also of the same number of elements) computed based upon local refinement following ZZ error estimation. The former meshes have FE solutions with potential energy significantly lower than the uniform mesh and ZZ refined mesh (and much closer to the HAS potential energy). The ZZ refined meshes are not the absolutely best meshes. MeshingNet learns from a large volume of ZZ meshes therefore is possible to outperform a single ZZ error estimation. Figure 3.7 illustrates some typical meshes obtained using MeshingNet: the non-uniform meshes correspond to the error distributions in the LAS. We also find that the traction-to-density ratio impacts the non-uniform mesh most significantly. Overall, this example shows that MeshingNet can generate high quality meshes that not only account for geometry but also the given material properties and BCs.

Finally we compare different DNN models, using the average potential energy on the testing data set. The baseline is from the HDUMs, whose FE solutions have a mean energy of -7.7293, followed by the meshes from ResNet2 (mean energy -7.6816), ResNet1 (-7.6815), and FCN (-

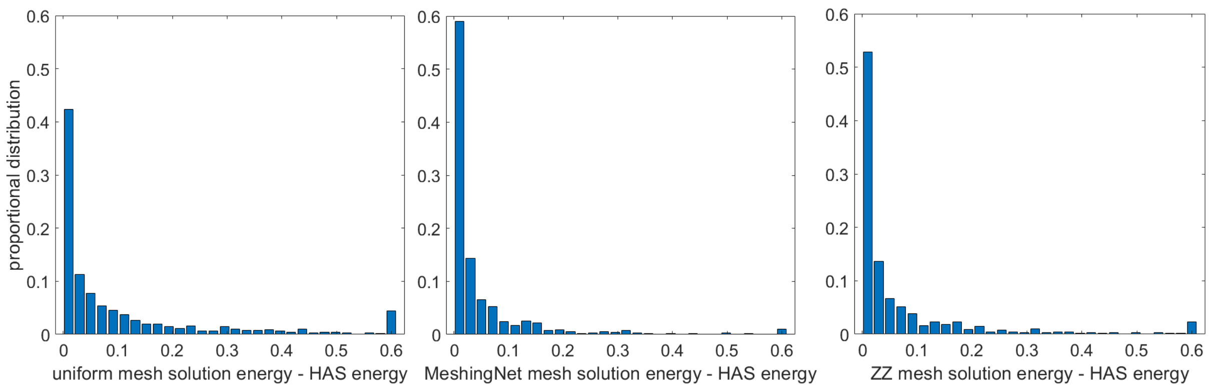


Figure 3.6: Potential energy comparison of solving 2D elasticity test problems on 4000 element meshes: uniform element size (left); MeshingNet (using FCN) meshes (centre); and ZZ refined meshes (right). We use the HAS energy as our baseline: the MeshingNet mesh solutions have energies that are significantly closer to the HAS energies than the ZZ mesh solutions and uniform mesh solutions (since a greater proportion of results are distributed near zero). The rightmost bar represents the proportion of all tests where the energy difference is no smaller than 0.6.

7.6813). The lower the better. These are all far superior to the uniform meshes of the same size (4000 elements), which yield FE solutions with a mean energy of -7.6030. Note that ResNet not only shortens the training time over FCN but, on average, produces better solutions.

3.4 Discussion

In this chapter, we have proposed a new non-uniform mesh generation method based on DNN. The approach is designed for general PDEs with a range of geometries, BCs and problem parameters. We have implemented a two-dimensional prototype and validated it on two test problems: Poisson's equation and linear elasticity. These tests have shown the potential of the technique to successfully learn the impact of domain geometries, BCs and material properties on the optimal finite element mesh. Quantitatively, meshes generated by MeshingNet are shown to be more accurate than uniform meshes and non-uniform ZZ meshes of the same number of elements. Most significantly, MeshingNet avoids the expense of *a posteriori* error estimation whilst still predicting these errors efficiently. Even though generating the training data set is expensive, it is offline and is thus acceptable in practice.

The meshes generated via MeshingNet may be used in a variety of ways. If our goal is to obtain a high quality mesh with a desired number of elements then the approach described in this paper provides a cost-effective means of achieving this. If however the goal is to produce a solution with an estimated *a posteriori* error that is smaller than a desired tolerance, then the generated mesh may not meet this criterion. This may be addressed either by regenerating the

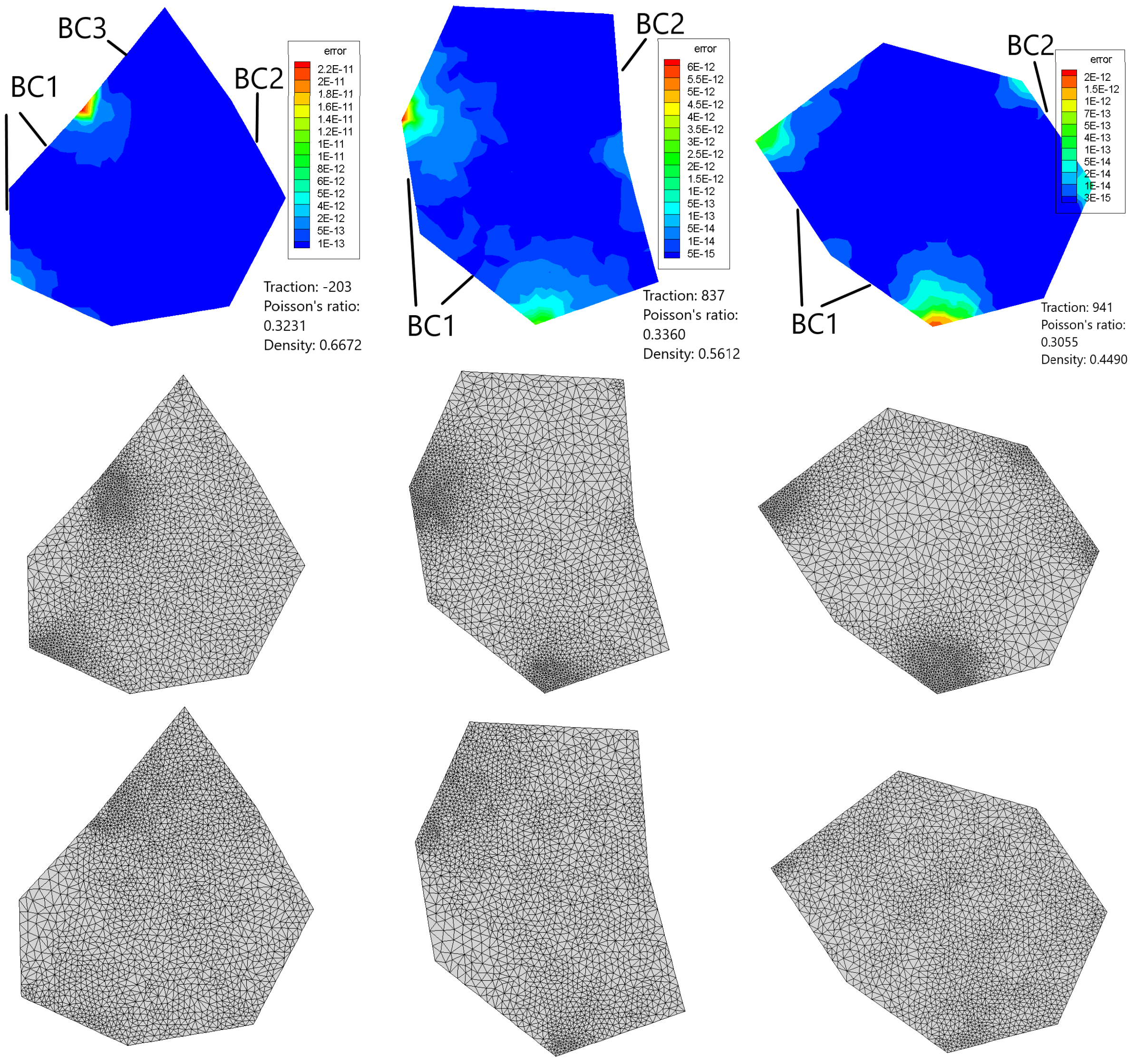


Figure 3.7: FE error (top) relative to HAS on coarse uniform mesh, non-uniform mesh guided by MeshingNet (middle) and non-uniform mesh refined by ZZ (bottom). The left geometry is an octagon and other two are heptagons (defined by placing their final vertex at the centre of edge 7).

mesh based upon a higher target number of elements in the final mesh, or through the use of a traditional *a posteriori* error estimate on the computed solution in order to guide further mesh refinement. In the latter case we can view MeshingNet as a means to obtaining an improved initial mesh within a traditional mesh adaptivity workflow.

In the next chapters, we generalize MeshingNet onto more general problems: 3D geometries, more complex PDE systems and BCs and time-dependent cases. In the future, it would be desirable to develop an interface to enable the mesh generator to read geometries from standard computer aided design software.

Finally, we note that the ANNs used in this initial investigation are relatively simple in their structure. In the future and later chapters the use of new DNN models, such as Convolutional/Graph Neural Networks, should be considered. These are appropriate for time dependent problems or problems with larger data sets, such as arising from more general geometries and BCs.

Chapter 4

Mesh for 3D elasticity

In this chapter we build upon, and extend, the two-dimensional work described in Chapter 3 to develop a novel algorithm for the generation of high quality tetrahedral meshes using artificial neural networks. The goal is to generate close-to-optimal meshes in the sense that the error in the computed FE solution is as small as it could be for a prescribed number of nodes or elements in the mesh. In this chapter we illustrate and investigate our proposed approach by considering the equations of linear elasticity, solved on a variety of three-dimensional geometries. This class of PDE is selected due to its equivalence to an energy minimization problem, which therefore allows a quantitative measure of the relative accuracy of different meshes (by comparing the energy associated with the respective FE solutions on these meshes). Once the algorithm has been introduced it is evaluated on a variety of test problems, each with its own distinctive features and geometric constraints, in order to demonstrate its effectiveness and computational efficiency.

4.1 Introduction

As discussed in the previous chapter, the finite element method (FEM) is one of the most widely used approaches for solving PDEs, which arise across multiple applications in computational mechanics. The key feature in determining the efficiency of the FEM on any given problem is the quality of the mesh: in general terms, the finer the mesh the better the solution but the greater the computational cost of obtaining it. This trade-off has led to vast body of research into the generation of high-quality FE meshes over decades. In this chapter, the objective is to predict the mesh size distribution in order to generate the best possible non-uniform tetrahedral mesh for a predetermined number of degrees of freedom.

Interest in the use of data driven methods to obtain solutions has grown significantly in recent years. In this work however, we do not aim to apply NNs to estimate PDE solutions directly: instead we continue to consider their use to estimate optimal meshes with which to compute traditional FE approximations. We present a universal deep-learning-based mesh generation system, MeshingNet3D, that extends our initial 2D ideas (last chapter), by building upon classical *a posteriori* error estimation techniques and adopting a new local coordinate system. Consequently, MeshingNet3D is able to guide non-uniform mesh generation for a wide range of PDE systems with rich variations of geometries, BCs and PDE parameters.

4.1.1 Non-uniform mesh generation

As outlined in Chapter 2, when applying FEM to approximate the solution of a computational mechanics problem, it is necessary to define both the type of elements and the computational mesh upon which the approximation is sought. In this subsection we expand this discussion, and that provided in Section 3.1, by describing in more detail the typical work flow that is used for adaptive mesh generation for elliptic problems in three dimensions.

The simplest elements are piecewise linear functions on simplexes (triangle in 2D and tetrahedra in 3D) however other choices are widely used. In 3D, these include higher order Lagrange elements (also defined on tetrahedra), trilinear and triquadratic elements (defined on octahedra) and more general elements associated with discontinuous Galerkin methods, which may be applied on hybrid meshes (J. Chan et al. 2016). In this chapter we restrict our consideration to unstructural tetrahedral meshes (Si 2015; Geuzaine and Remacle 2009). Structured meshes of octahedra do have some advantages, such as requiring less memory, however they are less flexible when considering complex geometries or when targeting highly non-uniform meshes, with optimal approximation properties, which is the goal of this work. When the volumes of the elements in a given mesh are approximately equal, and the aspect ratio of each tetrahedron is bounded by a small constant, we refer to the mesh as being *uniform*. Theoretical results about the asymptotic convergence of the FEM typically hold for sequences of finer and finer uniform meshes (Strang and Fix 1973). For many problems such meshes are not the best choice however: since the error in the corresponding FE solution may be much greater on some elements than on others. In such cases it would usually be far better to have more elements in the "high error" regions and fewer elements in the "low error" regions. The resulting mesh may have the same number of elements in total (with a non-uniform size distribution) but permit a much more accurate FE representation of the true solution. Ideally, we would like to identify an element size distribution to ensure that a prescribed global error tolerance can be obtained with the fewest possible number of elements. In practice this is attempted through the use of prior knowledge to control the mesh size distribution (e.g. geometrical information or *a priori* analysis Apel, Benedix, et al. 2011) or through an iterative process based around *a posteriori* error estimates for intermediate solutions (Stevenson 2007).

This iterative approach to mesh generation consists of three steps: (i) compute an FE solution on a coarse mesh (for some problems this method might miss features if the mesh is excessively coarse but for linear elasiticity it works fine); (ii) estimate the error locally throughout this solution; (iii) adapt the mesh based upon this estimate. At the next iteration these steps are repeated, beginning with the mesh produced in (iii).

There is a large body of work on the development of cheap and reliable *a posteriori* error estimates. Popular approaches include those which involve solving a set of local problems on each element, or on small patches of elements, to directly estimate the error function (Ainsworth and Oden 2011; Bank and Weiser 1985), and those based upon the recovery of derivatives of the solution field by sampling at particular points and then interpolating with a higher degree polynomial (Zienkiewicz and Zhu 1987; Zienkiewicz and Zhu 1992). For example, in the context of linear elasticity problems, the elasticity energy density of a computed solution is evaluated at each element and the recovered energy density value at each vertex is defined to be the average of its adjacent elements. The local stress error is then proportional to the difference between the recovered piece-wise linear energy density and the original piece-wise constant values, (Zienkiewicz and Zhu 1987). Considering its wide application in engineering practice, this "ZZ error estimate" has been used as the baseline in our work, to generate comparative data (and meshes) from which *MeshingNet3D* will be trained, and against which it will be evaluated.

On the third step no matter the type of refinement (h or r), the iterative process will generally require multiple passes to obtain a high quality final mesh. This therefore becomes a time-consuming pre-processing step - which we seek to avoid in this work.

4.1.2 Mean value coordinates

An important feature of our algorithm is the use of mean value coordinates (MVCs). These are a generalization of the barycentric coordinate system for simplexes (Hormann and Floater 2006; Floater 2003), to polygons in 2D and polyhedra with triangular faces in 3D (Floater et al. 2005), whereby the coordinates of any point within the polygon/polyhedron may be expressed as a convex combination of the positions of the boundary vertices. Consequently, all interior points in the neighbourhood of an arbitrary boundary vertex have a high value of the corresponding MVC component. MVCs also have a number of properties that make them attractive choices as input parameters to a DNN, for example their local smoothness with respect to spatial variations, as well as being both scale and and rotationally invariant.

4.2 Methodology

The goal of this research is to develop a robust, and widely applicable mesh generation procedure for the efficient FE solution of systems of elliptic PDEs in three space dimensions. Our particular emphasis here is on the equations of linear elasticity, however the approach described in this section may equally be applied to any family of problems for which a reliable *a posteriori* local error estimator is available to support the training phase for our neural network. In the first subsection we provide an overview of our methodology, with further details on the software design, training data generation and the training of the deep network given in the following subsections. Wherever possible we have maintained notations introduced for the 2D case in Chapter 3.

4.2.1 Theory

We seek to automatically generate high quality FE meshes for arbitrary instances within a given family of PDE problems, where each instance is defined by the domain geometry G (from a predefined family of possible geometries), the PDE parameters M, and the applied boundary conditions B. For any given mesh, the corresponding FE solution is assumed to be a unique solution, for which we have available a means of determining the local error. This computed *a posteriori* error is also assumed to be unique, and provides a mechanism for determining a desired FE element size for each location within the domain. Consequently, in order to generate a pseudo-optimal FE mesh we seek to estimate a mapping F that represents an ideal spatial ditribution of the FE element sizes:

$$F : X \to S \tag{4.1}$$

Here, X is the specified location in the domain and S is the target element isotropic size (for example average edge length) at X. This is precisely the information required by a 3D mesh generator in order to generate a non-uniform, unstructured finite element mesh. Noting that we define each instance by its specific geometry, parameter values and boundary conditions, we may express this mapping more precisely as:

$$F : X \to S(G, B, M; X) \tag{4.2}$$

Our goal is to make use of offline training to create a neural network that is able to learn the mapping

$$F : G, B, M, X \to S \tag{4.3}$$

After training, the NN is able to predict a pseudo-optimal mesh size distribution for unseen problems. Specifically, given G, B, M for any problem, and an arbitrary sample point X, the NN outputs a target element size at the sample point. This is demonstrated in Figure 4.1 that G, B, M, X are inputs and s is output of the NN.

4.2.2 Software and evaluation

In this chapter we use *Tetgen* (Si 2015) for tetrahedral mesh generation, and *FreeFem++* (Hecht 2012) to assemble and solve the corresponding global FE systems. The input to *Tetgen* includes a *.poly* file containing vertices and edges of polygons that define the boundary of the computational domain. From this file *Tetgen* is able to generate a uniform mesh based upon a single parameter, indicating a constant target element size. To generate a non-uniform mesh, *Tetgen* reads a background mesh from *.b.ele*, *.b.node* and *.b.face* files, and an element size list file *.b.mtr*, that defines target element sizes corresponding to the vertices of the background mesh. Having defined a valid mesh, *FreeFem++* is able to solve variational problems that are user defined. To do this it executes a *.edp* script file containing information such as: how to import the mesh; what type of finite element to use, what the specific variational form is; and which solver to apply. In this chapter all examples are based upon the use of linear tetrahedral elements (as generated by *Tetgen*) and the Lamé solver (user customised solver for the equations of linear elasticity). For the mass production of training problems a simple script has been produced that allows an appropriate *.poly* file to be generated for a given geometry G. Then, for each geometry this calls *FreeFem++* to obtain linear elasticity solutions for a range of material parameters (M) and boundary conditions (B). Note that *FreeFem++* not only solves the elasticity equations but also computes the total stored energy (see Equation 3.6), which may be used to evaluate the quality of a given FE solution. This is because the underlying PDE system corresponds to an energy minimization problem, so the analytic solution minimizes the energy function over all functions from the appropriate Soblov space ($(H_E^1)^3$ in this case). On the other hand, the FE solution minimizes the energy over all functions in the space of piecewise linear functions on the given tetrahedral mesh. Since this is a subspace of $(H_E^1)^3$, the energy corresponding to the FE solution is always greater than the energy of the analytic solution. Therefore, the lower the energy associated with the computed FE solution the better the solution. Consequently, the quality of any given set of tetrahedral meshes, for a particular problem, may be ranked based upon the computed energy corresponding to the finite element solution on each mesh: the lower the energy the better the mesh. We will again use this observation as part of the evaluation of our approach.

4.2.3 Data generation

Training data is required in order to sample the mapping of equation 4.2. Each training problem is defined by parameters that uniquely define the geometry (G), PDE parameters (M) and boundary conditions (B). For each such problem, multiple training data are generated by specifying numerous points, X, at which the target mesh size is given. This is illustrated in Figure 4.1: for which there are 3000 test problems, each of which generates multiple inputs, corresponding to different points X in the domain. The precise number of points X is problem-dependent which should be sufficient to represent the spatial mesh size variation throughout the domain (too many points will not decrease the training performance but will slow down the data generation). For each input the generated output, used to train the NN, is the target mesh size, S, for that point and that problem.

The value of S is computed using a variation of the iterative approach to mesh generation, based upon *a posteriori* error estimation (see Subsection 4.1.1). For each problem we generate a relatively coarse uniform mesh and compute the corresponding FE solution and error estimate. In this work we use the “ZZ” energy estimate. However, for different problems or different

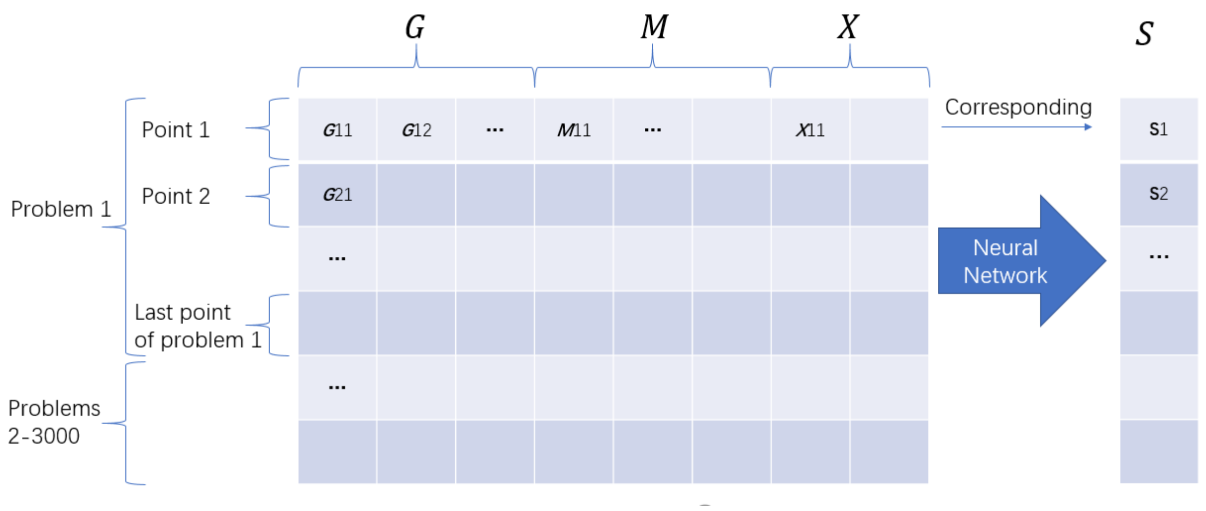


Figure 4.1: Illustration of the training data for *MeshingNet3D*: each individual problem is defined by the geometry G, the PDE parameters M and the boundary condition parameters B (not shown here). However for each such problem there are multiple sample points, X, in the domain, with the corresponding local mesh size S specified.

quantities of interest, other choices are possible. For each sample point, X, the estimated local error, $E(X)$, can be converted to a target element size using an inverse relationship such as

$$S(X) = \frac{K}{E(X)} , \tag{4.4}$$

for some scaling coefficient K. This is the value of $S(X)$ used to define Equation 4.2, as illustrated in Figure 4.1. The effect of the scaling coefficient is to control the total number of elements in the non-uniform mesh that is generated based upon the target local size distribution $S(X)$. Hence, for each test problem, K may be adjusted iteratively in order to obtain a target number of elements in the non-uniform mesh (or a target total error in the FE solution). There is mesh optimisation for the training data.

Note that the precise definition of the input vector in Figure 4.1 has to be problem dependent: a parameterization of the family of domains is required to define G; the number of free parameters in the PDE systems has to be predetermined; and the possible boundary conditions must also be parameterized. For each example shown in the experiments section of this chapter a different input vector has therefore been prescribed. Nevertheless, the methodology described here is shown to work robustly in all settings of different inputs.

The final component required for the data generation is the algorithm to select the sample points X for each of the training problems. This is achieved via two steps: first an initial non-uniform mesh with predefined target element number is generated (e.g. by *Tetgen*) based upon the *a posteriori* error computed on the coarse uniform mesh; Then we sample (each element of the non-uniform mesh has the same possibility been chosen) a fixed percentage of the elements of this mesh (we find that 10% is adequate), choosing each X to be the MVCs of the centroids of the sampled elements. Note that the advantage of sampling from the non-uniform, rather than the uniform, mesh is that the training data is weighted based upon the error distribution: our experiments show this to be advantageous. Though we sample elements for training, we have to predict target element size for all coarse mesh element when tested on unseen problems.

4.2.4 Training and using the neural network

The deep learning platform that we use in this work is *Keras* based on *Tensorflow*. Our networks are fully connected, typically with six hidden layers, though we find that our results are not especially sensitive to the number of layers or the precise number of neurons per layer. We do observe however that it is advantageous to first increase and then decrease the number of neurons per layer as we pass forward through the network. There is always only one neuron in the output layer. The activation functions selected in this model are rectified linear function (ReLU) for hidden units, with linear activation in the output layer. It is known from the study of PINNs that ReLU can lead to models that train very poorly (Zhao et al. 2023, Wang, Peng, et al. 2022) as the model is biased towards expressing the low-frequency spatial modes of the solutions and neglects the higher-order modes. However, we find ReLU performs good enough for out task. Before training, the input data is linearly normalised and 10% is selected for validation (monitoring the validation loss during training can help to identify and prevent over-fitting). The training itself uses mean square error loss and the stochastic gradient descent optimiser, *Adam*, with batch sizes of 128: for each of the examples considered in this paper this takes no longer than 3 hours on a Nvidia RTX 2070 graphic card.

Once trained, the NN can be used to guide mesh generation for unseen problems in real time. Given a new problem, defined by G, M and B, a uniform background mesh is generated based upon G alone. For each element in this background mesh we compute the MVC of its centre and concatenate this with the problem parameters to form an input vector for the NN. The corresponding output is the target element size at the centre of that element. The background mesh, with its associated target element size distribution, is then used to allow *Tetgen* to generate the desired non-uniform mesh. If the total number of elements in this mesh is outside the required range then each $S(X)$ may be scaled linearly before generating an updated non-uniform mesh. In this way, an adapted tetrahedral mesh of a specified size is generated directly, without the need to compute a sequence of FE solutions and *a posteriori* error estimates, as would otherwise be the case. Our framework can generalise over different numbers of input nodes becuase each time NN only predicts element size at one location.

4.3 Computational experiments

We present four computational tests which allow us to analyse the performing of MeshingNet3D across a range of different problems, geometries, boundary conditions and PDE parameters. The first and the third case involve prismatic geometries, which permit the description of spatial locations based upon "2.5D MVCs". These are composed of regular 2D MVCs in the x-y planes plus an additional z-coordinate, whilst the final example uses general polyhedral geometries and fully 3D MVCs.

For each of the examples we provide a brief description of the problem, followed by a discussion of the network topology used (including the specific input vector) and the training undertaken. We then present results based upon 500 unseen test problems. These results compare the FE solutions computed on the NN-guided mesh with those computed on a "ground truth" mesh of similar size, generated using the same ZZ *a posteriori* error estimator that was applied to train the network. We also compare against the FE solution computed on a uniform mesh with a similar number of elments. To facilitate these comparisons, for each of the 500 test problems, we compute the difference between the total energy of the FE solution on the NN-guided mesh with that of the FE solution on the comparison mesh. We then provide a histogram to illustrate the proportion of the test cases in different binned error ranges. A negative value of the difference indicates that the solution on the NN-based mesh has a lower energy and is therefore superior.

4.3.1 Clamped beam

We consider the problem of an over-hanging beam (under gravity), with different cross sections (G) and variable boundary conditions (B). In this case the material parameters (M) are not varied (the specific inputs to the Lamé solver in *FreeFEM++* being: density = 8000, Young's modulus = 210×10^9 and Poisson's ratio = 0.27).

Problem specification

The beam is a right prism with a convex quadrilateral cross section as illustrated in Figure 4.2. This cross section has vertices at $(x_0, y_0) = (0, 0)$ and $(x_1, y_1) = (0, 2)$, and also at (x_2, y_2) and (x_3, y_3) which are randomly sampled within $x_2 \in (1.5, 2.5)$, $y_2 \in (1.5, 2.5)$, $x_3 \in (-0.5, 0.5)$ and $y_3 \in (1.5, 2.5)$ for each problem. The length of the beam is fixed $(0 \leq z \leq 6)$ and a boundary shear, with components $(f_x, f_y, 0)$, is applied at the face $z = 6$. The face $z = 0$ is clamped and the bottom face is clamped between $z = 0$ and $z = \zeta$, where $2 < \zeta < 4$ (randomly sampled for each problem). All other boundaries are free, subject to zero normal stress. Hence the input vector for this problem requires values for x_2, y_2, x_3, y_3, ζ, f_x and f_y, along with the MVCs of the point at which the mesh spacing is required. In these examples, the parameters f_x and f_y are constrained to lie in the range $(-10^6, 10^6)$.

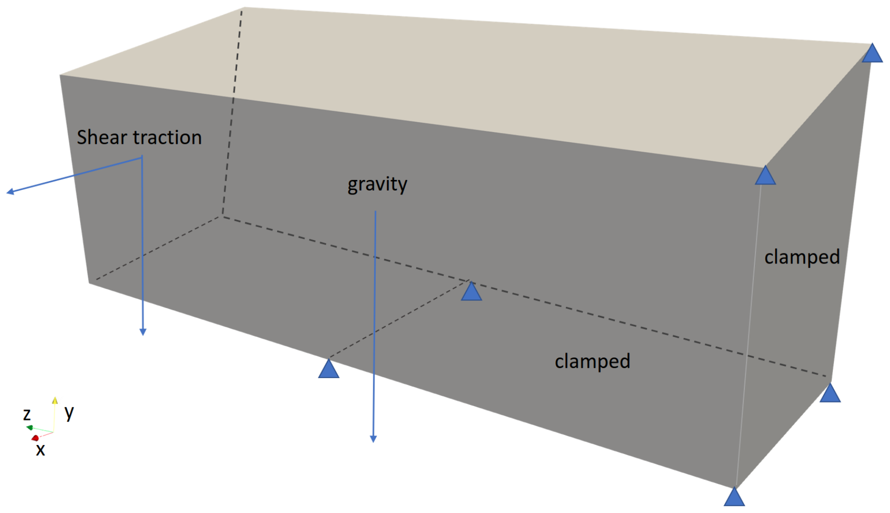


Figure 4.2: The geometry and boundary conditions for the *Clamped beam*, with constant cross section along the z-axis. The gravity is uniformly distributed over the volume. The surfaces bounded by four vertices with blue triangles are clamped.

Network information

In this example our fully-connected network has six hidden layers with 32, 64, 128, 64, 32 and 8 neurons respectively. Training data is generated based upon solving 3000 individual problems, each of which is obtained using a random choice for each input parameter (selected uniformly from its range), leading to 10, 740, 746 individual input-output pairs. Of these, 10% are selected for validation and the remainder are used for training using a batch size of 128. The training takes 10 epochs, meaning that each item of data has been used an average of 10 times. Figure 4.3

shows the rates of convergence for the training, along with the corresponding validation curve.

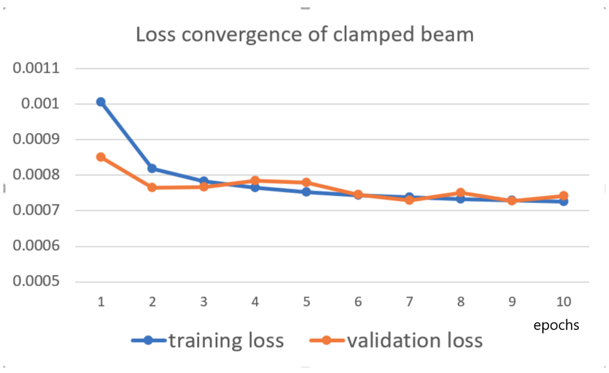


Figure 4.3: Training and validation loss of the clamped beam experiment

Results

Figure 4.4 demonstrates that the NN-guided meshes generally perform at least as well as the ground truth meshes (generated from explicitly-computed *a posteriori* error estimates) and, as expected, much better than uniform meshes. Two typical examples are shown in Figure 4.5, which compares NN-guided meshes (bottom) with their ground-truth counterparts (top). In each case the high mesh density near $y = 0$ and $z = \zeta$ is easily captured. More significantly however, high and low mesh density regions are captured well throughout the domain, with a smooth variation between these regions.

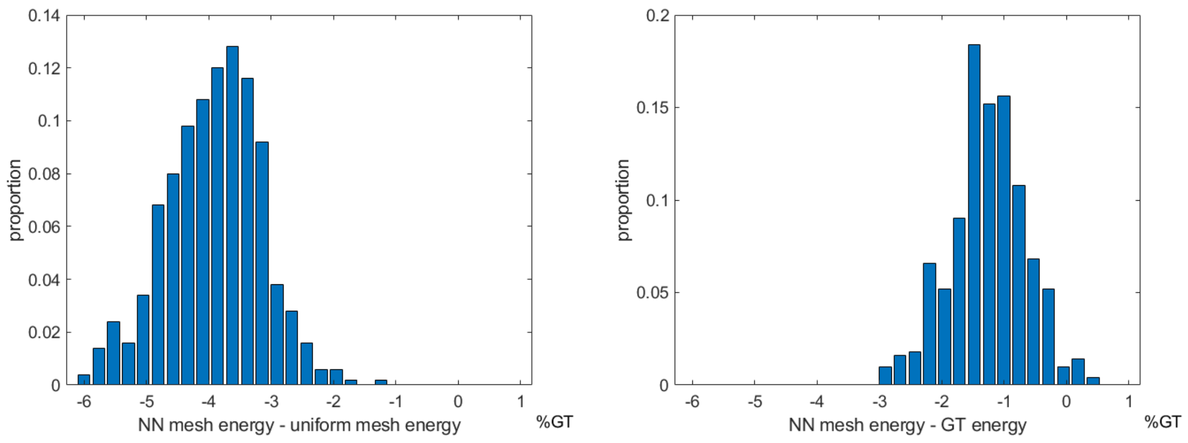


Figure 4.4: For the *Clamped beam*, FE energies of neural network (NN) generated meshes versus uniform mesh FE energies and ground truth (GT) energies. The height of each bar represents the proportion of experiment results in the energy range shown on the x-axis (as a percentage of the ground truth energy).

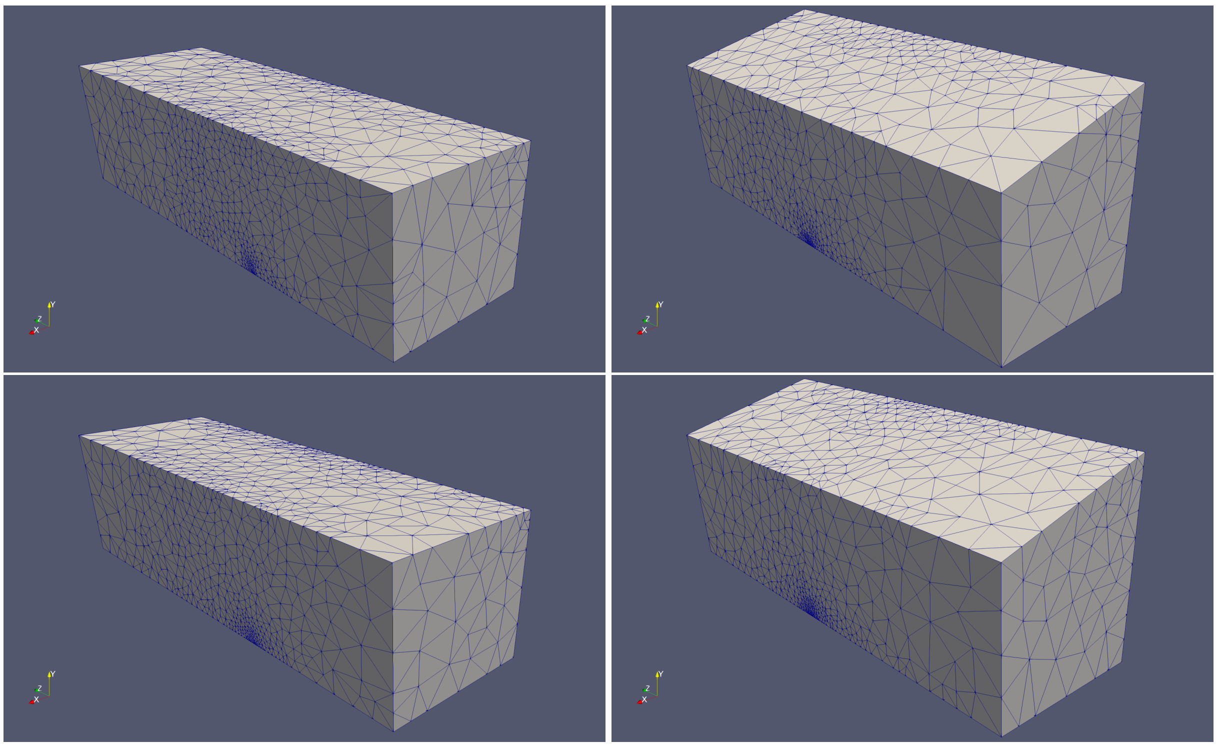


Figure 4.5: For the *Clamped beam*, ground truth meshes (top) and NN-guided meshes (bottom) for two test cases.

4.3.2 Laminar material

In this example we consider a variation of the previous problem for which the material parameters (M) are now permitted to vary but the geometry (G) and the boundary conditions (B) are kept fixed.

Problem specification

A beam of dimensions $1 \times 1 \times 5$ is composed of two horizontal layers, as illustrated in Figure 4.6. Each layer has a Young's modulus (E_{top} and E_{bot}) between 10^9 and 10^{11}, and a Poisson's ratio (ν_{top} and ν_{bot}) between 0.05 and 0.45. The densities of the two materials are both 8000 and the interface between the layers is at a height $y = h \in (0.2, 0.8)$. Half of the bottom surface ($y = 0$, $0 < z < 2.5$) is clamped, as is the surface $z = 0$. On the surface $z = 5$ a traction of amplitude 10000 is applied in the x direction, with all other boundaries free to displace under zero normal-stress conditions. Hence the input vector for this problem requires values for E_{top}, E_{bot}, ν_{top}, ν_{bot} and h, along with the coordinates of the point at which the mesh spacing is required. We actually use $\log_{10}(E_{\text{top}})$ and $\log_{10}(E_{\text{bot}})$ as the first two input parameters. This is because a large variation in Young's modulus results in a relatively small variation in the mesh size and using logarithm eases the learning of the neural networks.

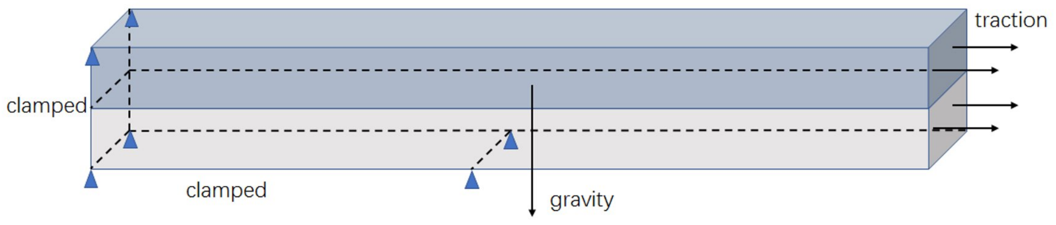


Figure 4.6: The boundary conditions and loads of the *laminar material* where the height of the interface is random

Network information

In this example our fully-connected network has five hidden layers with 32, 64, 32, 16 and 8 neurons respectively. Training data is generated based upon solving 3000 individual problems, each of which is obtained using a random choice for each input parameter (selected uniformly from its range), leading to $19,719,750$ individual input-output pairs. Of these, 10% are selected for validation and the remainder are used for training using a batch size of 128. The training takes 15 epochs, and Figure 4.7 shows the rates of convergence for this training, along with the corresponding validation curve.

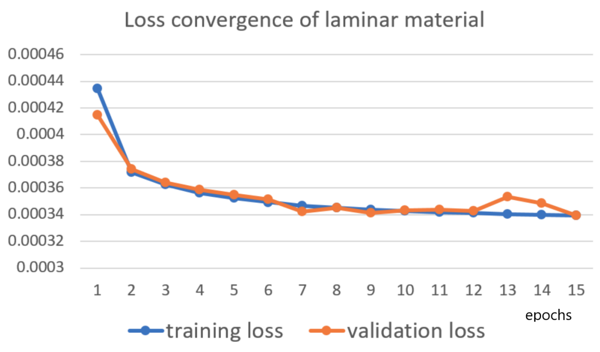


Figure 4.7: Training and validation loss of the laminar material experiment

Results

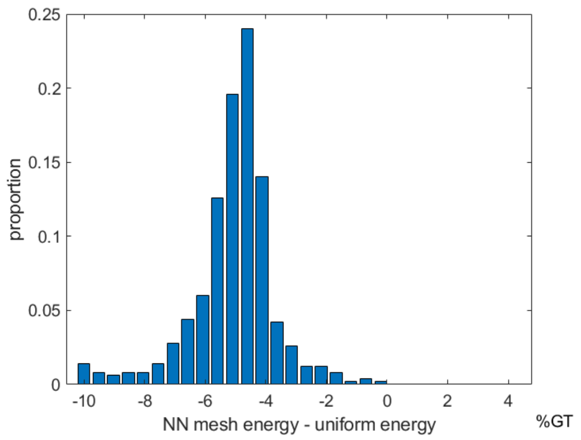
Figure 4.8 demonstrates that, as in the previous example, the NN-guided meshes typically perform on a par with the ground truth meshes, and much better than uniform meshes. Two typical examples are shown in Figure 4.9: in the case (a) and (c)

$$(\log_{10}(E_{\text{top}}), \log_{10}(E_{\text{bot}}), \nu_{\text{top}}, \nu_{\text{bot}}, h) = (10.82, 9.17, 0.34, 0.20, 0.34) \ ,$$

and for (b) and (d)

$$(\log_{10}(E_{\text{top}}), \log_{10}(E_{\text{bot}}), \nu_{\text{top}}, \nu_{\text{bot}}, h) = (9.17, 10.33, 0.44, 0.21, 0.41)\ .$$

In the first example the top layer has the higher Young's modulus, which leads to a higher mesh density in this layer (for both the NN-guided and the ground-truth meshes). Conversely, in the second example the bottom material is stiffer than the top and we see a very different distribution of the element size. In each case there is a strong correlation between the NN-guided mesh and the ground-truth case.



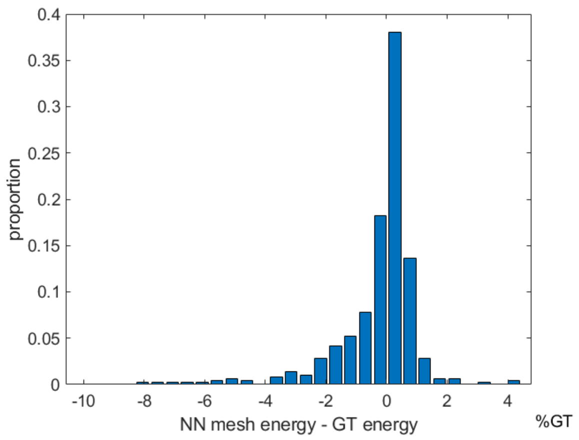


Figure 4.8: For the *Laminar material*, FE energies of neural network (NN) generated meshes versus uniform mesh FE energies and ground truth (GT) energies. The height of each bar represents the proportion of experiment results in the energy range shown on the x-axis (as a percentage of the ground truth energy).

4.3.3 Hex-bolt with a hole

We consider the problem of a hex-bolt (under torque), with different cross sections (G). In this case the material parameters (M) are not varied (the specific inputs to the Lamé solver in *FreeFEM++* being: density = 8000, Young's modulus = 210×10^9 and Poisson's ratio = 0.27).

Problem specification

A regular hexagonal prism has an octagonal prism hole inside it where the height of the prism is $h = 4$ (Figure 4.10 left). On the cross section, the edge length of the regular hexagon is 4 and the octagon is coaxial with the hexagon. The eight vertices of the octagon lie on the same circle, whose radius varies $r \in (0.2, 1.0)$. The arc angles between vertices are random. Linear distributed pressures are applied to create a torque on the top ($p = -10000x + 10000$) and

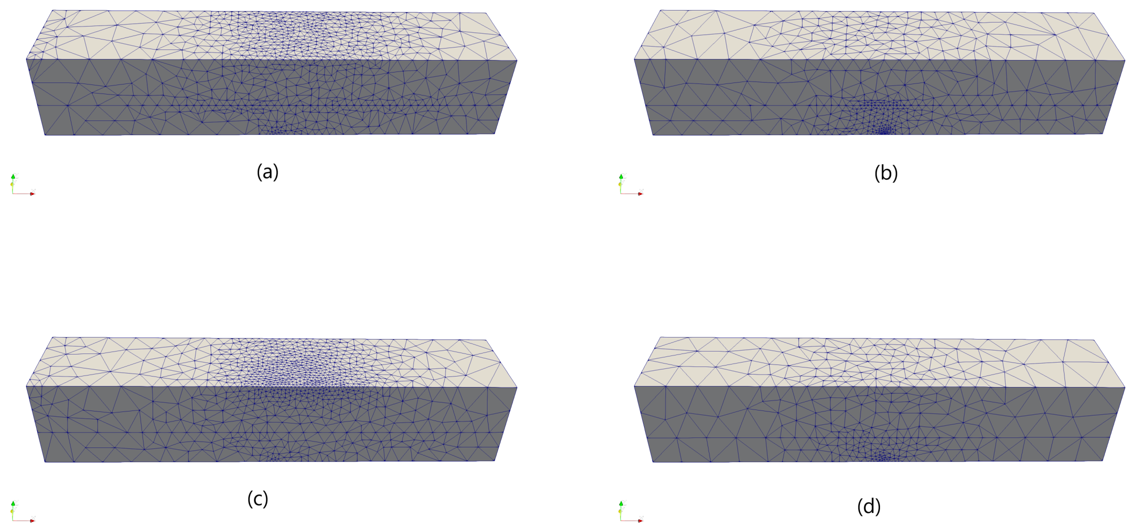


Figure 4.9: (a)(c) and (b)(d) are two problems in the *laminar material* experiments. (a) and (b) are ground truth meshes and (c) and (d) are non-uniform meshes guided by the neural network. A comparison of (b) and (d) shows NN does not refines well at the interface between two laminars.

bottom ($p = -10000x - 10000$) surfaces. The eight surfaces of the hole are clamped. The input vectors for this problem include the position of the octagon's eight vertices and the MVCs of the target point expressed with respect to both the vertices of the outer hexagon and the inner octagon (combined with its z coordinate, $z \in (-1.0, 0.0)$).

Network information

In this example our fully-connected network has four hidden layers with 32, 64, 16 and 8 neurons respectively. Training data is generated based upon solving 3000 individual problems, each of which is obtained using a random choice for each input parameter (selected uniformly from its range), leading to $10,748,618$ individual input-output pairs. Of these, 10% are selected for validation and the remainder are used for training using a batch size of 128. The training takes 10 epochs and Figure 4.11 shows the convergence for this training, along with the corresponding validation curve.

Results

Figure 4.12 shows that the *MeshingNet3D* meshes are again better than uniform meshes and that the NN mesh energies are very close to those of the ground truth. As illustrated in Figure 4.13,

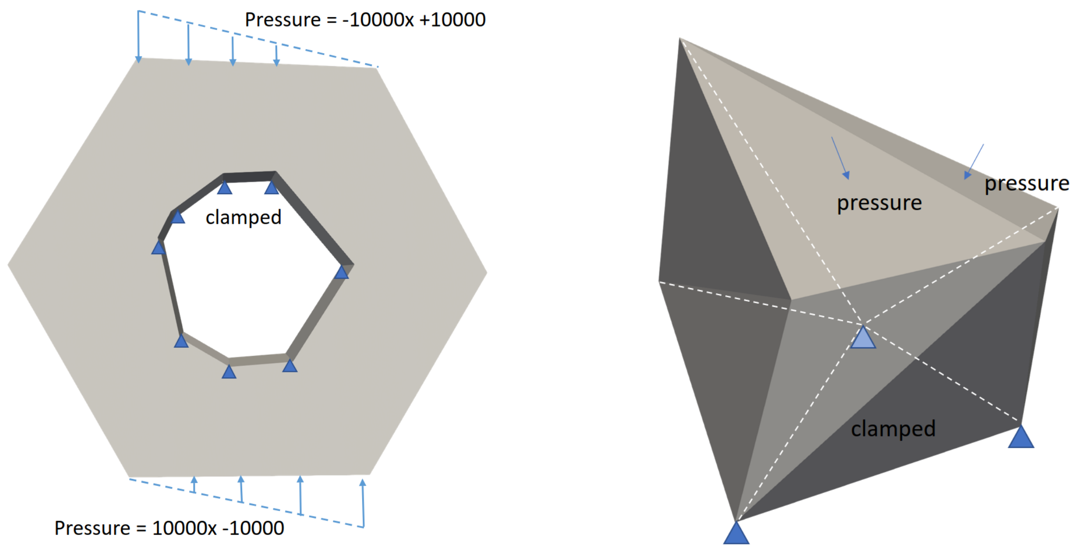


Figure 4.10: The boundary conditions and loads of the *hex−bolt* (left) and *irregular polyhedron* (right). On *hex − bolt*, eight surfaces of the hole are clamped, linear distributed pressure is applied on top and bottom surfaces.

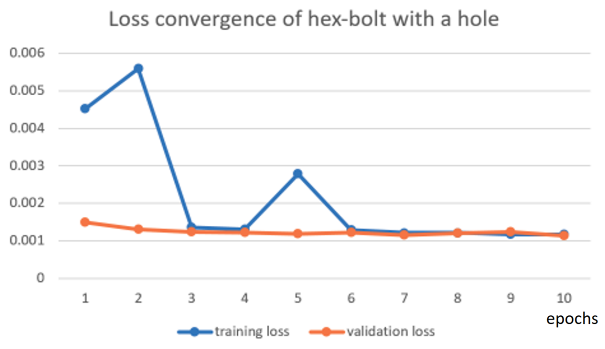
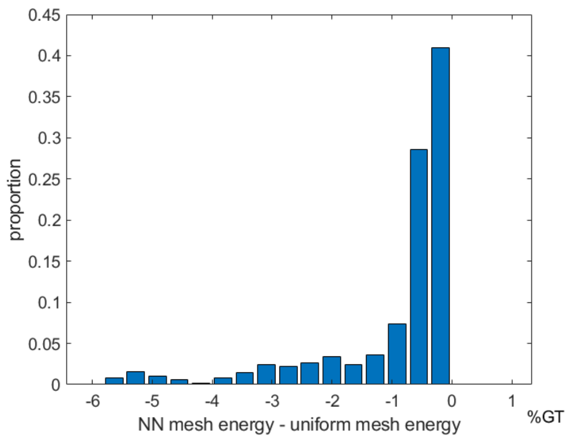


Figure 4.11: Training and validation loss of the hex-bolt with a hole experiment

the NN can successfully guide non-uniform mesh generation on very different geometries. This example also illustrates the success of the proposed approach on non-simply-connected domains. Note that the second problem (on the right) in Figure 4.13 illustrates one of the worst performing cases for the NN mesh relative to the ground truth: here, the NN mesh is more uniform than the ground truth (though still a vast improvement on a standard uniform mesh).

4.3.4 Irregular polyhedron

We now consider the problem of mesh generation on arbitrary twelve-faced polyhedra, with a range of geometries (G) and variable boundary conditions (B). In this case the material parameters (M) are not varied (the specific inputs to the Lamé solver in *FreeFEM++* being: density = 8000, Young's modulus = 210×10^9 and Poisson's ratio = 0.27).



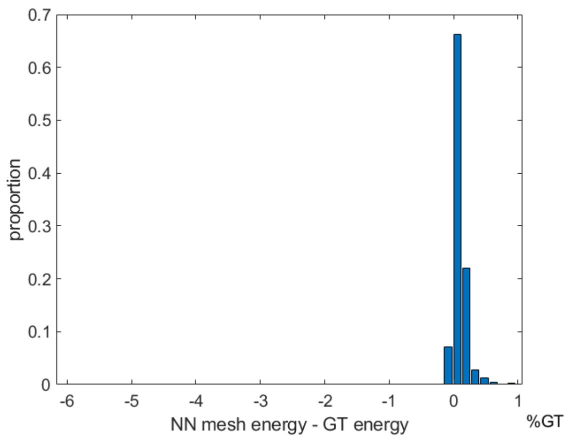


Figure 4.12: For *hex − bolt with a hole*, FE energies of neural network (NN) generated meshes versus uniform mesh FE energies and ground truth (GT) energies. The height of each bar represents the proportion of experiment results in the energy range shown on the x-axis (as a percentage of the ground truth energy).

Problem specification

An irregular polyhedron with twelve triangular faces and eight vertices is illustrated in Fig 4.10 (right). The four "bottom" vertices are constrained to be co-planar and one of the two bottom triangular surfaces (i.e. the two triangles whose union is bounded by the four co-planar vertices) is clamped. In all training and testing problems the geometries are subject to the restriction that the four bottom vertices always lie in the same plane. A normal pressure of amplitude 10000 is applied on the two "top" surfaces (i.e. the triangular faces whose union is bounded by the other four vertices) and zero normal stress is applied on the other nine triangular faces. The input vectors for this problem define the Cartesian coordinates of the eight vertices and the corresponding MVCs of the point at which the mesh spacing is required.

Network information

In this example our fully-connected network has four hidden layers with 32, 64, 32, 16, and 8 neurons respectively. Training data is generated based upon solving 3000 individual problems, each of which is obtained using a random choice for each input parameter, leading to $7,383,999$ individual input-output pairs. Of these, 10% are selected for validation and the remainder are used for training using a batch size of 128. We use the network after training 10 epochs and Figure 4.14 shows the convergence for this training, along with the corresponding validation curve.

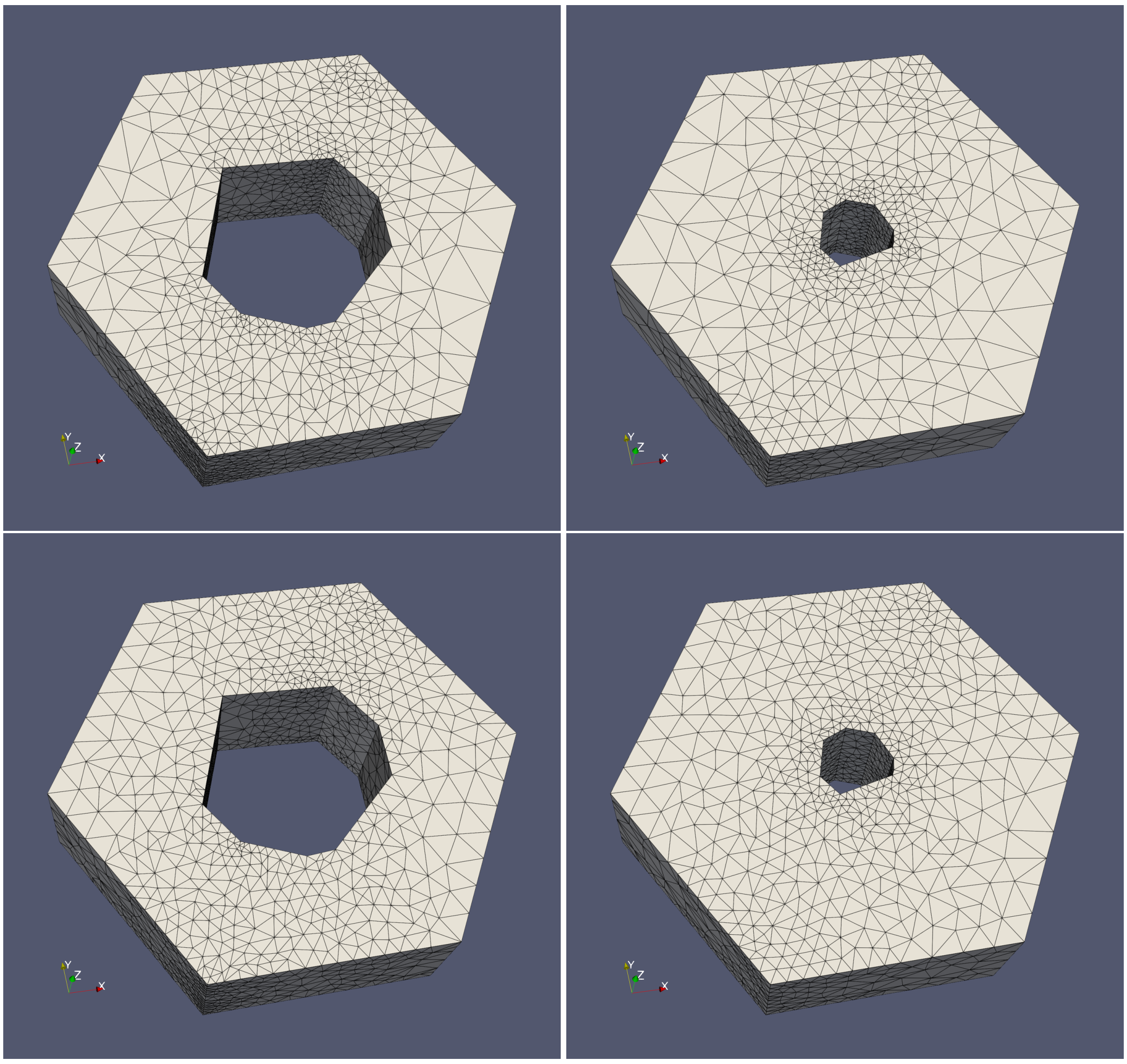


Figure 4.13: Hex-bolt experiment, ground truth meshes (top) and NN meshes (bottom) , the left and right are two problems that only have different geometries

Results

From Figure 4.15 it is clear that the *MeshingNet3D* meshes are significantly better than uniform meshes and that the solution energies are relatively close to those of the ground truth: though in a very small number of cases the ground truth mesh is slightly superior. One such example is shown in Figure 4.16 (three views of the same problem), where we see that the NN mesh appears to be more conservative in some aspects of its local refinement. Nevertheless, even in this worst-case scenario, the *MeshingNet3D* mesh generally has the same regions of refinement as the ground truth mesh.

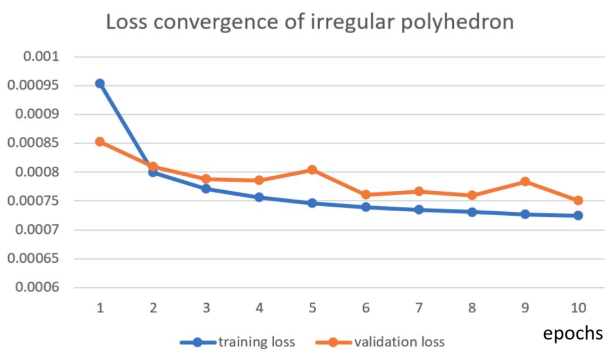


Figure 4.14: Training and validation loss of the irregular experiment

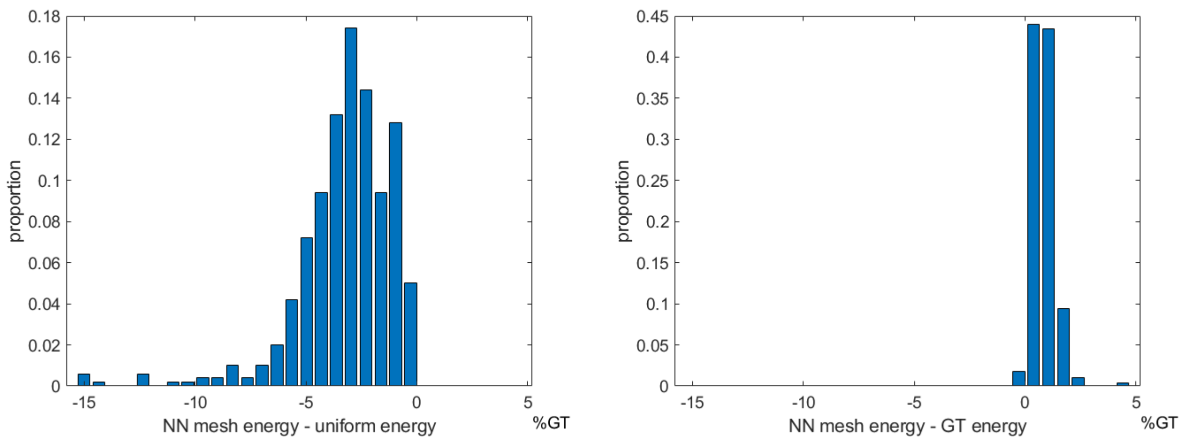


Figure 4.15: For *irregular polyhedron*, FE energies of neural network (NN) generated meshes versus uniform mesh FE energies and ground truth (GT) energies. The height of each bar represents the proportion of experiment results in the energy range shown on the x-axis (as a percentage of the ground truth energy).

4.3.5 Discussion

Across the four experiments described in this section we have shown results over a range of geometries, boundary conditions and material parameters. For each different family of geometries, we need to train a NN separately. Further work could be done to make the framework more generalisable than this. For each problem the input layer of the NN is necessarily of a different dimension, which is dependent on the problem specification (along with the MVCs of the target point), whereas the output is always a single value representing the predicted mesh spacing at the target point. The number and size of the hidden layers is not a critical choice, but does naturally have some impact on the performance of the network.

As an example, to illustrate this, Table 4.1 shows the performance of five different networks when applied to the fourth of the test problems above. In each case the networks have been trained on the same data set, with validation losses having converged after 10 epochs. The

NN	NN structure	training epochs	normalised average energy
NN1	32-16-8	10	9.0×10^{-3}
NN2	32-64-16-8	10	8.1×10^{-3}
NN3	32-64-32-16-8	10	7.9×10^{-3}
NN4	32-64-128-32-16-8	10	8.1×10^{-3}
NN5	32-64-128-64-32-16-8	10	8.6×10^{-3}

Table 4.1: Comparison of 5 different fully connected NNs based upon normalised average energies of the finite element solutions. NN3 gives the lowest average energy and therefore provides the best mean performance.

networks are then used to compute meshes on the same testing set of 500 unseen problems and the finite element solutions computed on all meshes. The energy of each solution is normalised against the energy of the finite element solution computed on the "ground truth" mesh so as to allow a meaningful average to be taken across all 500 cases. This is the value shown in the "normalised average energy" column of Table 4.1: so, the lower this energy the better the meshes are on average. The results shown in Subsection 4.3.4 are generated using NN3 from the table but NN2 and NN4 produced meshes of very similar quality. The network denoted by NN1 appears to have too few degrees of freedom to be able to model the non-uniform mesh patterns satisfactorily, whereas the network denoted by NN5 likely has too many degrees of freedom for the size of our training data set. Table 4.1 shows that NN3 has the lowest energy and best results. With the number of layers increases or decreases the energies both increase. If the degree of freedom is too low the NN can hardly model the complexity mapping. If the degree of freedom is too high, very large volume of training data is needed otherwise the trained NN performs poorly.

Note that our NNs are always "spindly", with the greatest number of neurons in the inner layers. We find from experiments that this kind of network appears to have the best performance for the set of tasks considered in this work. Given that our problems have a relatively small number of inputs and a very small number of outputs (typically one) this is perhaps not surprising: to capture the highly nonlinear relationships between the inputs and the mesh spacing across the domain, significant complexity must be introduced into the network between the input and output layers.

Finally, we note that MeshingNet3D has the potential to make simulations more efficient for designers who use pre-built 3D models provided within Computer Aidd Design (CAD) software to accelerate design. From screws and bolts, to washers and bearings, CAD can not only define geometries but also materials. Embedding pre-trained *MeshingNet3D* in these CAD libraries could save meshing cost and provide high-quality non-uniform meshes. Similarly, *MeshingNet3D* can help parametric design where the NN is pre-trained for each geometry topology: under the guidance of the NN an appropriate mesh is generated in response to each iteration of the design. To implement this efficiently the challenge will be in defining suitable family of boundary conditions as NN inputs, where forces due to interacting objects are unknown *a priori*. However, for components in a specific assembly, if contacts are defined, the load may be inferred by data-driven methods.

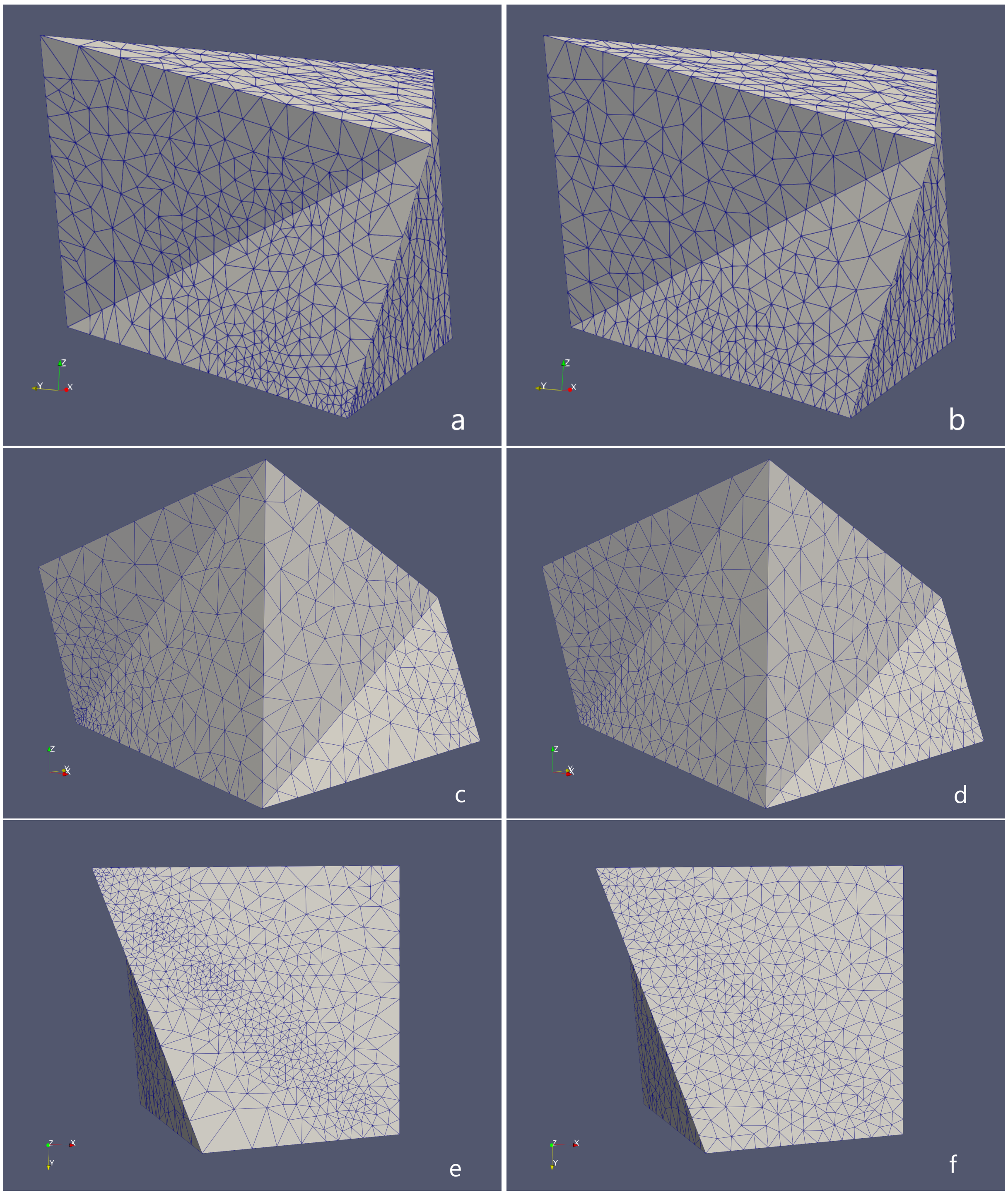


Figure 4.16: A ground truth mesh (a, c and e) and corresponding NN mesh (b, d and f) selected from 500 testing problems, they are in front (a and b), right (c and d) and bottom (e and f) views.

Chapter 5

Mesh for transient PDEs

In Chapter 3 and Chapter 4 we have used supervised trained neural networks to guide non-uniform mesh generation for steady PDEs. The success of instant high quality non-uniform mesh generation encourages us to explore deep learning guided mesh adaptation for transient PDEs, which is the topic of this chapter.

5.1 Introduction

For transient PDEs, the choice of the time step, initial conditions are critical. Compared with steady PDEs, transient PDEs are generally solved step by step through time incrementally and incur significant computational time cost. There are solutions for multiple time steps therefore rich data can be collected when solving transient PDEs.

Transient PDEs include linear cases such as the heat equation and the wave equation and non-linear cases such as the Navier-Stokes equations and the Hamilton-Jacobi equations. Transient PDEs are usually solved by discretizing separately in space and time. The time domain is discretized into uniform or nonuniform intervals then there are finite time steps. The finite element approximation only needs to cover the spatial part of the domain (but with time dependent coefficients). A single uniformly spaced spatial mesh can be used for solving transient PDEs however it may be far from optimal depending on the nature of the solution. The solution in space changes with time and therefore the nature of the solution can alter significantly throughout a simulation. A good mesh at one time step may not be a good mesh at a later step. For a new time step, if the solution varies significantly, mesh adapted to the latest solution is required.

In this chapter we use deep learning to predict a mesh that is adapted to the latest solution without seeing the latest solution but knowing the previous solutions. We first introduce the methodology of the framework then present three experiments.

In this chapter we focus on the CFD applications. As introduced in subsection 2.4.2, machine learning for CFD mesh adaptation has been actively studied in recent years. Most of these research aims to generate static non-uniform mesh even through the optimal meshes for many CFD applications are generally transient. Neural networks can map a boundary geometry to a non-uniform mesh by supervised learning. However, to capture the transient flow structure, such as vortex, shock and fluid interface is very difficult without seeing the solution. To tackle this challenge, we use the data of previous time steps and apply graph convolution networks for spatio-temporal prediction. Because we can predict an adaptive mesh before the FE solution, mesh adaptation and the solver of FE are possible to run in parallel. This novel framework has the potential to significantly accelerate the simulation. The goal of this chapter is to develop a new framework for the efficient generation of adaptive unstructured FE meshes for the systems of transient PDEs.

5.2 Methodology

Our particular emphasis here is on fluid dynamics, however the approach described in this section may equally be applied to other transient problems for which a reliable *a posteriori* local estimator is available to support the training phase for our neural network. The most well known PDEs for fluid dynamics are the Navier-Stokes (N-S) equations:

$$\begin{aligned}
\rho(\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y}) &= \rho f_x - \frac{\partial p}{\partial x} + \mu(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}) \\
\rho(\frac{\partial v}{\partial t} + u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y}) &= \rho f_y - \frac{\partial p}{\partial y} + \mu(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2})
\end{aligned} \tag{5.1}$$

u and v are the horizontal and vertical velocity. ρ is the density and μ is the viscosity. f_x and f_y are the external force. p is the pressure. The density is not time-dependent since the fluid is incompressible.

In the first subsection we provide an overview of our methodology, with further details on the software design, training data generation and the training of the deep network given in the following subsections.

5.2.1 Theory

In the conventional mesh adaptive FE-CFD process, FE and adaptive mesh generation work in serial. Generally there are 4 steps:

1. Computing to obtain solution $(u, v, \rho)^t$ on FE space V_h^t
2. Error estimation, $(u, v, \rho)^t \rightarrow E^t$
3. Mesh generation, $E^t \rightarrow T_h^{t+1}$
4. Read mesh and interpolate $(u, v, \rho)^t$ onto new FE space V_h^{t+1}

In the first step, the FE space V_h^t is defined by the spatial mesh T_h^t and the selected finite elements for each variable. For the time step t, $(u, v, \rho)^{t-1}$ is known and the velocity and density is updated to $(u, v, \rho)^t$ by solving the weak form of the N-S equation (the weak formulation does not absolutely hold the equations but instead with respects to certain test functions).

In the second step, the state of the art error estimation is based on the FE solution $(u, v, \rho)^t$, so called *a posteriori* error. There are many possible choices of error estimator and we describe one specfic example as used in FreeFem++ (Hecht 2012). Hessian (Huang 2005, Wessner et al. 2003) is the default anisotropic mesh matrix builder in FreeFem++ and we use it for error estimation in this chapter. The error is used to compute the metric for automatic non-uniform mesh generation. As we are interested in information on element shapes as well as sizes, we need to convert the solution to at each point $x \in \Omega$ three parameters. A metric tensor M is given at every point $x \in \Omega$ by

$$M(x) = R(x) \begin{pmatrix} \gamma_1 & 0 \\ 0 & \gamma_2 \end{pmatrix} R(x)^{-1} \tag{5.2}$$

where $\gamma_1 > 0$ and $\gamma_2 > 0$ are the eigenvalues of $M(x)$ and $R(x)$ is a rotation matrix. We need to choose a metric tensor that has the property that its eigenvalues are similar when the solution field is isotropic and very different when it is anisotropic. We can then use this to construct a mesh that aligns with any local anisotropy in the solution. Suppose we only work with one variable denoted η. Assume that a first solution has been computed over a given mesh and let

η be continuous piecewise quadratic interpolated. Then the interpolation error depends on the Hessian matrix of η (Castro-Dıaz et al. 1997).

$$H(x) = \begin{pmatrix} \partial^2\eta/\partial x^2 & \partial^2\eta/\partial x\partial y \\ \partial^2\eta/\partial x\partial y & \partial^2\eta/\partial y^2 \end{pmatrix} = R(x) \begin{pmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{pmatrix} R(x)^{-1} \tag{5.3}$$

and the matrix tensor M is defined by

$$M(x) = R(x) \begin{pmatrix} |\lambda_1| & 0 \\ 0 & |\lambda_2| \end{pmatrix} R(x)^{-1} \tag{5.4}$$

The interpolation error is characterized by the p+1st derivates. Error equidistribution yields the mesh stretching as

$$\frac{h_i}{h_j} = (|\lambda_j|/|\lambda_i|)^{1/(p+1)} \tag{5.5}$$

h_i and h_j are mesh sizes at two perpendicular directions and p is the order of the element.

In the third step, the mesh is globally regenerated. The anisotropic mesh size control is based on the error distribution $E^t(x)$ which in this chapter is the metric $M(x)$. In most cases the metric is defined on the non-uniform mesh T_h^t however it can also be defined on a uniform background mesh T_b. The background mesh should be moderate fine so that can hold the solution and mesh size metric. The FreeFem++ embeded mesh generator BAMG (Hecht 1998) applys the adjustable smoothing procedure and limits the maximum subdivision of a background edge. BAMG uses $M(x)$ which we introduced above, to generate 2D anisotropic adaptive mesh.

In the fourth step, the nonuniform mesh T_h^{t+1} (generated by BAMG) is read and a new FE space V_h^{t+1} is built upon this new mesh. The solution $(u, v, \rho)^t$ is interpolated from FE space V_h^t to V_h^{t+1}. FreeFem++ provides automatic solution interpolation. The Barycentric interpolation (using Barycentric coordinates as interpolation weights which is very efficient) does not conform mass conservation therefore frequent interpolation could be a source of error. We always adapt the mesh for every time step and there is no big change in the solution between time steps.

Figure 5.1 top demonstrates a breakdown of a single time step for a typical actual fluids simulation case where error estimation and mesh generation takes significant time (note that the timings given are ficticious but intended to be representative). This motivates us to find a new framework (Figure 5.1 bottom) letting FE and adaptive mesh generation work in parallel (they are not coupled). A problem arise when we do mesh generation for t+1 before FE solution for step t comes out: we can only estimate the error for t-1 where the optimal mesh for t+1 requires error estimate for t. Our proposed solution is to approximate the error for t using a sequence of history of error estimates. In this chapter we use a pretrained DNN to predict the error at time t based upon the error estimates at time t-2 and t-1. This will allow us to generate the mesh for the next time step in parallel with computing the solution at the current time step.

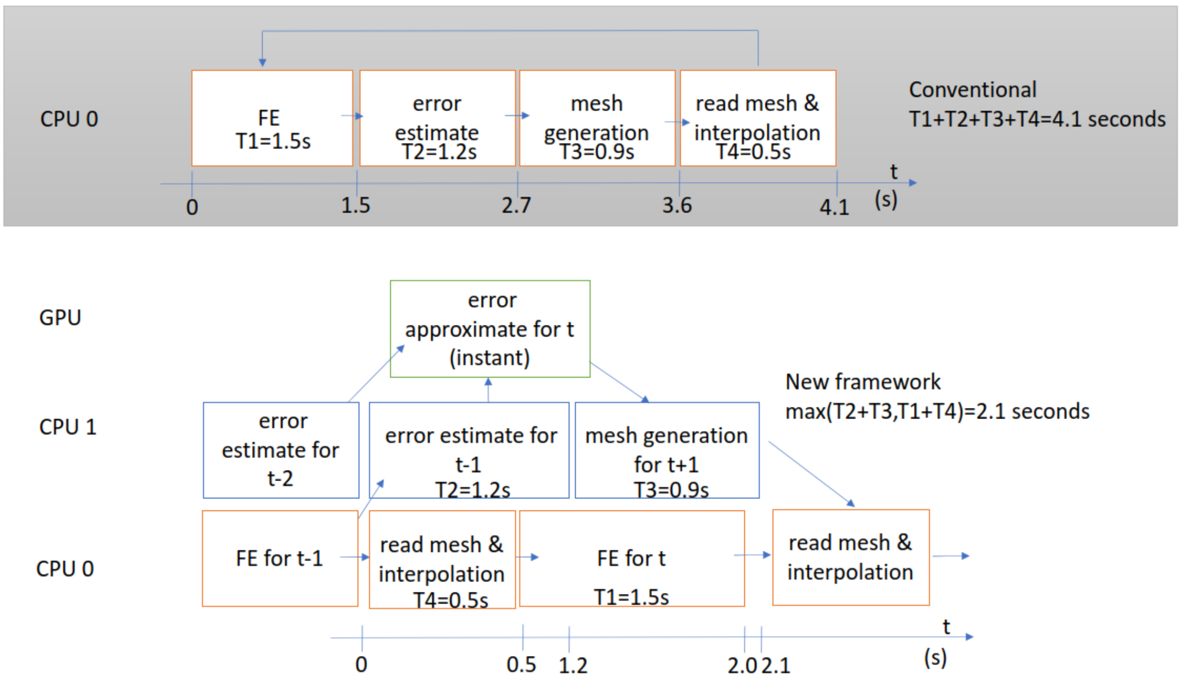

CPU 0
FE T1=1.5s
error estimate T2=1.2s
mesh generation T3=0.9s
read mesh & interpolation T4=0.5s
Conventional
T1+T2+T3+T4=4.1 seconds
0
1.5
2.7
3.6
4.1
t
(s)
GPU
CPU 1
CPU 0
error approximate for t (instant)
error estimate for t-2
error estimate for t-1 T2=1.2s
mesh generation for t+1 T3=0.9s
New framework
max(T2+T3,T1+T4)=2.1 seconds
FE for t-1
read mesh & interpolation T4=0.5s
FE for t T1=1.5s
read mesh & interpolation
0
0.5
1.2
2.0
2.1
t
(s)

Figure 5.1: A comparison of conventional FE with mesh adaptation process and our proposed parallel framework. In one these fictitious fluid simulations, the conventional method takes 4.1 seconds for a time step while the proposed parallel framework only takes 2.1 seconds. GPU is not good at all the tasks (especially when working with unstructured meshes) thus is only used for running the NN (inference).

5.2.2 Overview

As outlined above, the requirement is to predict what the error estimator distribution, $M(x)$, will be at the end of the current time step prior to actually computing the N-S solution for that time step. We proposed to do this using a trained DNN whose input is the actual $M(x)$ at the previous two time steps. Since this investigation is intended as a proof of concept, we have not actually implemented it in parallel as shown in Figure 5.1.

We apply a data driven model, based upon a graph convolutional network to make predictions. The supervised trained network reads previously estimated error and predicts new error distribution instantly. In the new framework, parallel computing starts at the finish of FE (0s at Figure 5.1) and ends (synchronise) at the completion of new mesh generation and FE (2.1s at Figure 5.1). The mesh reading, solution interpolation and FE work on CPU 0. The error estimation and mesh generation work on CPU 1 and the neural network works on the GPU. The parallel speedup gets highest when the two CPUs or CPU threads require similar execution times at each time step.

We use graph neural network because the data structure is non-Euclidean. Graph is a structure made of vertices and edges. An FE mesh can be regarded as an undirected graph. The nodes of the FE mesh corresponds to the vertices of the graph and the edges of FE mesh elements corresponds to the edges of the graph. We use graph convolutional networks (GCN) (use the background mesh as the graph) to make spatio-temporal predictions for mesh metric distribution M:

$$M^t = GCN(M^{t-2}, M^{t-1}) \tag{5.6}$$

which is applied at every node all coupled. M^t is not always positive definite. Different from autoregressive model, the output variable depends on its previous values non-linearly. We assume that the variation of the metric distribution is continuous across time steps. This is because the evolution of fluids structures are generally continuous. For example, if we know where a shock is at more than one previous time step, we know how fast it moves and can predict where it is likely to be at the new time step. It is possible to access more than two previous time steps however we think using only two time steps is adequate for this proof of concept.

We use first order truncated expansion in terms of Chebyshev polynomials to approximate a graph convolution layer (introduced in Subsection 2.3):

$$g_{\theta'} \star (M^{t-2}, M^{t-1}) \approx \theta'_k T_k(\tilde{L})(M^{t-2}, M^{t-1}) \tag{5.7}$$

Where $\tilde{L} = \frac{2}{\lambda_{max}} L - I_N$. λ_{max} denotes the largest eigenvalue of the graph Laplacian L (M^{t-2} and M^{t-1} must have a same graph). The Chebyshev polynomials are recursively defined as $T_k(x) = 2xT_{k-1} - T_{k-2}(x)$, with $T_0(x) = 1$ and $T_1(x) = x$. Note that the expression is k localized (the output nodal value depends on inputs value of the nodes within k distance) since it is a k^{th}-order polynomial in the Laplacian. We use $k = 1$, for the first GCN layer the output depends on $M^{t-2}(x)$, $M^{t-2}(x')$, $M^{t-1}(x)$ and $M^{t-1}(x')$ where when x is the location of the target nodes, x' is the location of nodes connected the target nodes with only one edge (graph convolution is done discretely in space). During graph convolution, there is information exchange on the edges and for each layer information only move 0 or 1 edge distance. When the GCN is composed of N layers, $M^t(x)$ depends M^{t-1} and M^{t-2} nodes within N edges distance of x. The closer the nodes to the target node, the stronger the dependency.

5.2.3 Architecture of the software

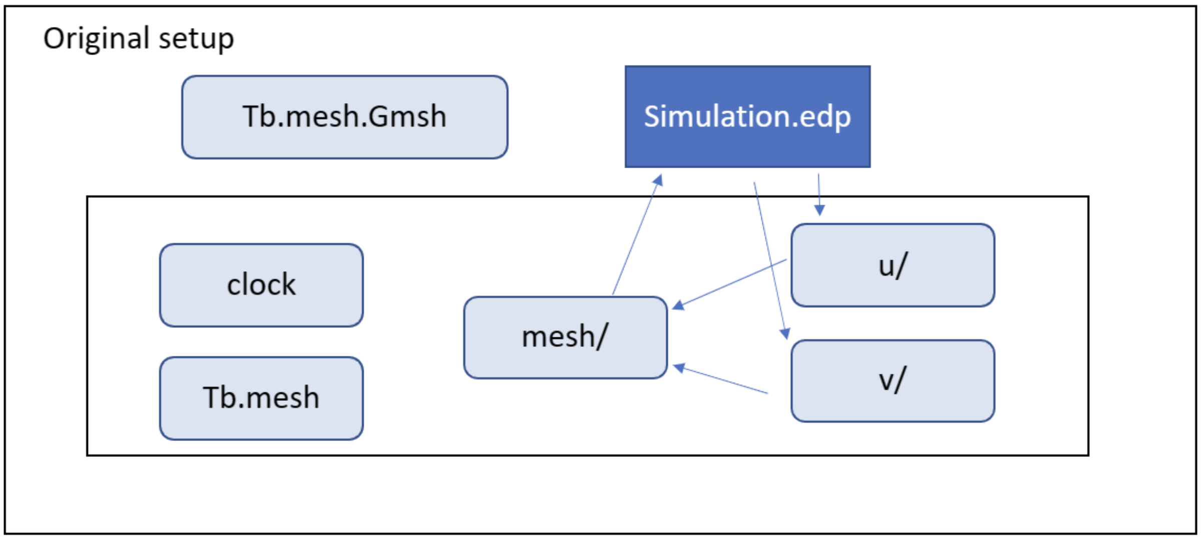


Figure 5.2: The original set up of the simulation with adaptive mesh

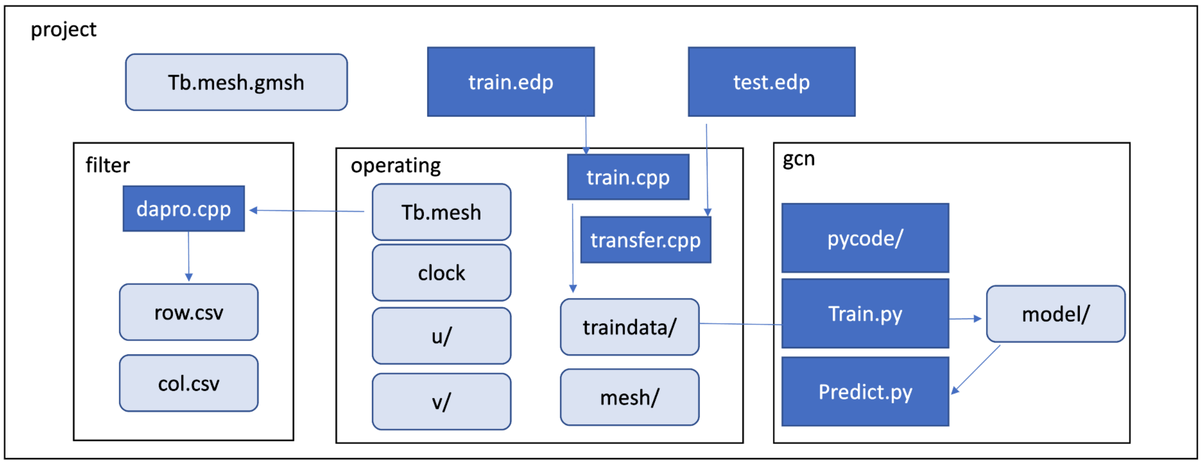


Figure 5.3: The architecture of our proposed model

In this chapter we use BAMG (the mesh generation module of FreeFem++) for 2D anisotropy mesh generation and FreeFem++ to assemble and solve the corresponding global FE systems.

BAMG is guided by the Hessian metric at each point on the background mesh.

Figure 5.2 demonstrates the setup of simulation with adaptive mesh. “original” means this is based upon the approach implemented in FreeFem++. *simulation.edp* is the executable script of FreeFem++. *clock* stores the time step. Simulation of each time step generates solutions in *u/* and *v/*. The solutions are interpolated onto the background mesh *Tb.mesh*. BAMG read the solution on the background mesh and generates adaptive mesh in *mesh/*. Finally *simulation.edp* reads the adaptive mesh.

Figure 5.3 shows the software architecture of the project. The project is published at Z. Zhang 2023. FreeFem++ scripts *train.edp* and *test.edp* generates training data and test the trained NN. The two *.edp* scripts are the core of the model. First, a geometry is defined and a background mesh *Tb.mesh* is established on the geometry. Then *dapro.cpp* converts *Tb.mesh* to a sparse Laplacian matrix of the corresponding graph. An iterative solver is defined for the specific fluid dynamic problem. At the finish of FE computing of each time step, the solution is stored in the *operation* folder. *clock* stores the time step. In *train.edp*, *operation/train.cpp* is called where error estimation, mesh adaption and data generation take place. In *test.edp*, *transfer.cpp* is called which use *gcn/predict.py* to predict a new mesh.

Our data-driven model is allocated in the folder *gcn*. In *gcn/pycode/module.py*, the neural network is defined by composition of layers. *gcn/pycode/multi_graph_cnn_layer.py* and *gcn/pycode/graph_ops.py* defines the graph convolution layer. *gcn/pycode/filters.py* and *gcn/pycode/norm.py* are used for computing filters based on graph Laplacian and do normalisation (introduced in Subsection 5.2.5). *gcn/train.py* reads data from *operation/data* to train the model and save at *gcn/model*.

5.2.4 Data generation

Data is required to train the model that predicts the error distribution. Each training problem is uniquely defined by geometry, PDE parameters and boundary conditions. For each problem, multiple training data are generated at different time steps t, at which error distributions (decide mesh size metric which is the key to mesh refinement) of t-2, t-1 (as input data) and t (as output data) are given. The weights of the NN are constant for different time steps. In a single sample, errors on all vertices of the background mesh are included. We may use data from different problems (but the same geometry) to train a network. For such case, all the problem are based on the same background mesh because they share the same filters in the graph convolution layers.

The *a posteriori* error estimation is used for both input and output data. First, the latest solution is interpolated onto the background mesh. Then, the error metric is constructed from the Hessian of the solution on the background mesh. The Hessian for each node of the background mesh is composed of three independent values to control the anisotropic mesh size. Note that our framework is agnostic to the specific choice of error estimator, other choices may work for different problems where we only have tested Hessian. When we prescribe the range of total element number of meshes, we use a global scaling coefficient K which multiplies the metric. For each time step, K may be adjusted iteratively in order to obtain a target number of elements in the non-uniform mesh. It is possible to predict K directly considering the ratio of current and expected elements number.

5.2.5 Normalisation

Normalisation is the process of translating data into the range [0,1] and is necessary for our deep learning model. The way to translate mesh metric M to $M^{normalised}$ is linear:

$$M_{i,j}^{normalised} = (M_{i,j} - min_{i,j})/(max_{i,j} - min_{i,j}) \tag{5.8}$$

where the key is to find the upper bound $max_{i,j}$ and lower bound $min_{i,j}$ for M. Because when testing the entire solution is unknown before simulation, the bounds are discovered during the simulation. max^{global} and min^{global} are the global maximum and minimum metric find during the simulation (initialise max by a very negative value, update max if find a value is greater than the current max). $max(x)$ and $min(x)$ are the local maximum and minimum metric find at the location x. Therefore we have global and local normalisation strategy. Local normalisation is better at enhancing the performance of our deep learning model, where maximum and minimum are spatio-temporal dependent distributions. In local normalisation, the range between maximum and minimum at each location is smaller than global normalisation. Learning and predictions in a smaller range is more precise.

5.2.6 Training and applying the neural networks

We use Keras based on *TensorFlow* as the deep learning platform. Our networks are typically constructed with five hidden graph convolution layers. We use first order Chebyshev polynomial (the basis of graph convolution algorithm) for the filter wherein each layer information passing distance is one (the output value on a vertex depends on the input value of the same vertex and its connected neighbours). Using multiple graph convolution layers enables longer distance information passing. We observe that using an adequate number of layers is important and using higher order Chebyshev polynomial does not help when only using a small number of layers. This means there is little direct dependency of long distance neighbours in spatio-temporal prediction. It is advantageous to first increase then decrease the number of channels (in each channel we have data on the graph) per layer as we pass forward through the network. We set 8-16-32-16-8 channels in the hidden layers. The activation functions selected in the model are rectified linear functions. The training uses mean square error loss and the stochastic gradient descent optimiser, Adam (Kingma and Ba 2015), with batch sizes of 8-32 for different problems. For each of examples considered in this chapter training takes no longer than 10 hours on a Nvidia RTX 2070 graphic card. The bottleneck of training is the memory size of the graphic card which limits the total size of the training data that the graphic card can load in.

Once trained, the NN can be used to predict mesh metrics and guide non-uniform mesh generation for unseen problems. Given a new problem, defined by geometry, PDE parameter, boundary condition and initial condition, a uniform background mesh is generated based on the geometry alone. The background mesh is generally coarser than the actual FE mesh and is the basis for graph convolution. The graph convolution filter is computed according to the graph Laplacian of the background mesh. The neural network reads the previous metrics defined on the background mesh and uses the convolution filter to predict the new metric to guide mesh adaptation. If we prescribe the target element number of the non-uniform mesh, we adjust the element size proportionally to reach the target element number. Even though we proposed the parallel computing framework, the main focus of this chapter is to show the supervised learning performance. We use serial computing to simulate the parallel computing and emphasis on demonstrating the application of the concept rather than rate of accelerating.

5.2.7 Evaluation

We evaluate the performance of the target mesh by constructing reference meshes. The reference meshes are the very highly refined nonuniform adaptive meshes. In each time step, we compute the target solution and reference solution based on the two meshes. The target solution is interpolated onto the reference mesh. Then we compute the root mean square (RMS) difference of the target solution and the reference solution as the error of the target solution. The RMS error is sensitive to outliers.

For best evaluating the accuracy of the NN predicted mesh, we also set up the ground truth mesh, something we refer to as the "baseline" mesh and their corresponding solutions. At time step t, the NN mesh for t+1 is predicted based on the error estimation of t-1 and t-2. The ground truth mesh is based on the error estimation of t, which requires the solution of t and mesh generation cannot work in parallel with FE. The baseline mesh (nonuniform) for t+1 is based on the error estimation of t-1. In the baseline method mesh generation and FE can work in parallel without extra techniques. Our goal is that the NN mesh outperforms the baseline mesh and closely reproduces the ground truth mesh.

5.3 Experiments

We present three computational tests which allow us to analyse the performance of the new framework across different problems, geometries, boundary conditions and initial conditions. We choose test problems with significant flow structure where mesh adaptation is needed. The first problem is incompressible flow past cylinders with the famous Von Karman vortices in the wake of cylinders. The second test considers the interacting flow of two immiscible incompressible fluids. The final case is for the supersonic flow of a compressible fluid around an obstacle.

For each of the examples we provide a description of the problem, followed by a discussion of neural network used. We then present the results based on unseen test problems. These results compare the solutions computed on the NN-guided mesh with those computed on the "ground truth" and "baseline" mesh of similar size.

5.3.1 Flow past by cylinders

The following experiment considers incompressible flow with parabolic inflow velocity past cylinders in a channel. The Reynolds numbers of the experiment sets are 100, 200 and 400. In this range, the flow develops a Karman vortex street, which results from periodic vortex shedding. This demonstrates the performance of our model on incompressible unsteady laminar flow.

An Newtonian, incompressible viscous fluid satisfies:

$$\begin{aligned} \partial_t \mathbf{u} + \mathbf{u} \cdot \nabla \mathbf{u} + \nabla p - \nu \Delta \mathbf{u} &= 0 \quad \text{in} \quad \Omega \\ \nabla \cdot \mathbf{u} &= 0 \quad \text{in} \quad \Omega \end{aligned} \tag{5.9}$$

Where $\mathbf{u}$ is the velocity, p is the pressure and ν is the viscosity coefficient. We use a projection algorithm solver provided by *FreeFem++* official document to solve N-S equation. The algorithm proposed by Chorin (Chorin 1967, Chorin 1968) is:

$$\begin{aligned} \frac{1}{\delta t}[\mathbf{u}^{t+1} - \mathbf{u}^t o \mathbf{X}^t] + \nabla p^t - \nu \Delta \mathbf{u}^t &= 0 \\ \nu \partial_n \mathbf{u} | \Gamma_{out} &= 0 \\ -\Delta p^{t+1} &= -\nabla \cdot \mathbf{u}^t o \mathbf{X}^t \end{aligned} \tag{5.10}$$

where δt is the time step interval and n is the normal direction of the boundary. $\mathbf{u}$o$\mathbf{X}$=$\mathbf{u}$($\mathbf{x}$-$\mathbf{u}(\mathbf{x})\delta t$) since $\partial_t \mathbf{u} + \mathbf{u} \cdot \nabla \mathbf{u}$ is approximated by method of characteristics. t denotes the current time step.

We use Chorin's algorithm with free boundary condition at outlet, to compute a correction q, to the pressure.

$$\begin{aligned} -\Delta q &= \nabla \cdot u \\ q &= 0 \quad \text{on} \quad \Gamma_{out} \end{aligned} \tag{5.11}$$

and define

$$\begin{aligned} \mathbf{u}^{t+1} &= \tilde{\mathbf{u}} + P \nabla q \delta t \\ p^{t+1} &= p^t - q \end{aligned} \tag{5.12}$$

where P is the L^2 projection with mass lumping. $\tilde{\mathbf{u}}$ is the $(\mathbf{u}^{t+1}, v^{t+1})$ of Chorin's algorithm. P1 elements are used. Above equations is provided by *FreeFem++* official document case "A Projection Algorithm for the Navier-Stokes equations".

To generate a non-uniform mesh, we first interpolate the solution onto the background mesh then do error estimation on the background mesh. This is suitable when there is no abrupt discontinuity in mesh size.

Problem specification

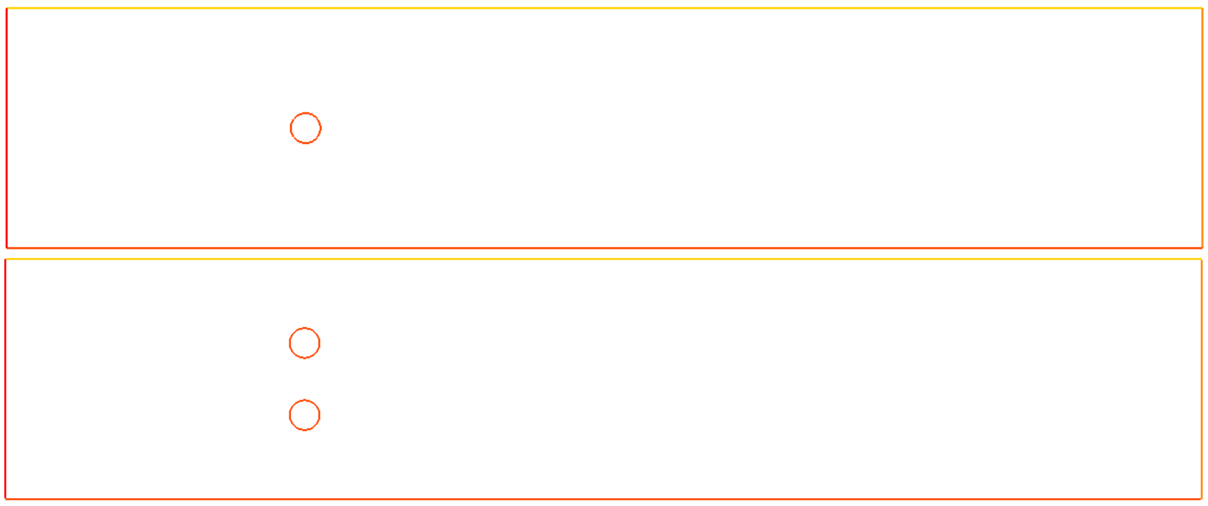


Figure 5.4: The geometry G1 (top) and G2 (bottom)

We do experiments on two geometries $G1$ (single cylinder) and $G2$ (double cylinders) (Figure 5.4) who share the same outer boundary but have different inner boundaries and topology. The outer boundary crops a rectangle $\Gamma 1 : x \in [-2, 8],\ y \in [-1, 1]$. $G1$ has inner boundary $\Gamma 2$: $x = 0.5 + cos(s)/8, \quad y = sin(s)/8$. $G2$ has inner boundary $\Gamma 3$: $x = 0.5 + cos(s)/8, \quad y = -0.3 + sin(s)/8$ and $\Gamma 4$: $x = 0.5 + cos(s)/8, \quad y = 0.3 + sin(s)/8$.

The boundary conditions are $u = 0,\ v = 0$ at $y = 1 \cup y = -1 \cup \Gamma 2 \cup \Gamma 3 \cup \Gamma 4$, $v = 0$ at $x = 8$, $u = -ky^2 + k,\ v = 0$ at $x = -2$. k is the amplitude of the parabolic inlet velocity which we choose 0.5, 1 and 2 in our experiments. The viscosity $\nu = 0.00125$ and the time step $\delta t = 0.05$. The Reynolds number $Re = \frac{uL}{\nu} = 200k$.

Network information

In this example we choose the following six problems in Table 5.1 to train six NNs. In each problem 500 time steps are used to train the NN. Training takes 3000 epochs and the batch

problem	NN	geometry	inlet velocity
1	NN1	G2	-y^2 + 1
2	NN2	G2	-2y^2 + 2
3	NN3	G2	-0.5y^2 + 0.5
4	NN4	G1	-y^2 + 1
5	NN5	G1	-2y^2 + 2
6	NN6	G1	-0.5y^2 + 0.5

Table 5.1: A set of 6 training problems with different geometry and inlet velocity. They are used to train 6 NNs from NN1 to NN6.

sizes are 32. The input dimension is (498,9320,6) and output dimension is (498,9320,3). The first entry is 498 because the last data contains time steps 498, 499 and 500. The second 9320 is the number of nodes in the background mesh. The third entries are 6 and 3 because input has two time steps of 3 anisotropic metrics and the output has the single time step of three metrics. Even though the input and output dimension seems massive, NN runs instantly, thus the framework is highly practical. We use local normalisation in this test problem. Training use Adam optimizer and mean square error loss. The output channel dimensions of each layer are 8, 16, 32, 16, 8 and 3.

Results

We only use problem 1 in Table 5.1 as the test problem and test NN1-NN6 with a comparison against baseline mesh and ground truth mesh, using ultra fine mesh as reference. The ultra fine mesh is nonuniform based upon Hessian of the solution and the error tolerance is very low so that the mesh is globally highly refined. There are about 400000 vertices in the ultra fine mesh. The details of evaluation methods are in Subsection 5.2.7. Figure 5.5 demonstrates the flow evolution of the test problem. The Reynolds number of the test problem is 200. The vortices generates at the cylinders. The clock wise and anticlockwise vortices alternately shedding and shape the vortex street. A high quality adaptive mesh should be refined at the vortices. We find the Hessian error estimator refines the mesh at the boundaries of the vortices rather than the center of the vortices (See Figure 5.8 ground truth mesh).

A visual comparison of meshes are in Figure 5.8 for time step 300. The reference mesh is almost uniformly using the finest elements and is globally finer than the other meshes. The ground truth mesh is fine near the cylinders and at the boundary of vortices. Compared with the ground truth mesh, the NN meshes have some region at the centre of the vortices that the elements are larger. The overall quality of the NN mesh is good and there is no case that fails completely.

Figure 5.7 shows the error of ground truth, baseline and NN meshes solutions. Each of the NNs performs well where NN1 and NN5 even outperforms the ground truth. The difference of 6 NNs is that they are trained on different data. This demonstrates the NN has strong generalization capability regarding different geometries and boundary conditions. When work on a different geometry, we only need to change the convolution filter corresponds to the graph of the new geometry. Note that for this specific test the baseline solution is better than the ground truth. The reason is perhaps the Hessian is not the best metric for mesh adaptation and perhaps refining at the boundaries of vortices is not the best strategy. A better strategy is refine at the centre of the vortices according to vorticity and reinforcement learning may provide insight about what is good refinement. Figure 5.6 shows the velocity error of NN5 mesh and baseline mesh using this ground truth solution as reference. It shows NN mesh is closer to the ground truth mesh than the baseline mesh which means the NN learns the ground truth very well. In conclusion, this test case shows NN can learn the mesh adaptation strategy very well though the ground truth meshes may not always be based upon an optimal strategy themselves in this particular case.

5.3.2 Two phase flow

Multiphase flows are of high interest in fluid engineering where they arise in many applications. The non-uniform mesh can be very efficient where the fluid boundary should be refined. We consider a simple test case which two immiscible incompressible fluids are sloshing in a closed box. This test case and the solving method comes from De Vuyst 2013.

To generate a nonuniform mesh, we first do error estimation then interpolate the error onto the background mesh. This is suitable because on the fluid interface there is abrupt discontinuity in mesh size and the refined region is very “sharp”. This has the risk that the mesh size jumps across time steps at some regions.

Problem specification

Let us consider two Newtonian viscous incompressible immiscible fluids k = 1,2 with respective density ρ_k, $k = 1, 2$ and constant viscosity ν_k. A spatial bounded domain Ω is filled up by

problem	initial ψ
1	0.2+0.3x-y
2	0.2+0.35x-y
3	0.1+0.3x-y
4	0.1+0.35x-y
5	0.1+0.4x-y

Table 5.2: A set of 5 training problems with different initial condition.

the two fluids. The surface tension forces at fluid interface is neglected in this study. In many applications the surface tension is a critical driving force but we are not considering it here for simplicity due to the focus on mesh adaptivity. A level set function is used for front tracking. We consider a levelset variable ψ such that $\psi < 0$ in Ω_1^t and $\psi > 0$ in Ω_2^t. The level $\psi = 0$ exactly tracks the fluid interface σ.

The geometry of the closed box is a 2×1 rectangle, the density of the two fluids are $\rho_1 = 1$, $\rho_2 = 8$ and the viscosity are $\nu_1 = 0.05$, $\nu_2 = 0.01$. The fluids are under gravity and the gravity is 9.81. The initial condition is problem dependent by setting the function of psi. The mesh is adapted according to the Hessian of the density.

Network information

We constructed five training problems with different initial condition (Table 5.2). Because each training problem has only 100 steps, we combine the data from 5 training problems to train a neural network. Training takes 2000 epochs and the batch size is 32. We use local normalisation in this test problem. Training uses Adam optimizer and mean square error as loss. The output channel dimensions of each layer are 8, 16, 32, 16, 8 and 3.

Results

We have done two experiments, one with initial fluid interface y=0.25+0.3x, one with interface y=0.1+0.375x. Figure 5.9 demonstrates the evolution of the flow with initial boundary y=0.1 +0.375x. The interface is straight at the beginning. At time steps 10-30 the both ends of the interface is curve, the middle of the interface is almost straight and the slope is getting lower and lower. At time steps 50-90 the interface is high at the left and low at the right. At time step 100 the whole interface is almost flat. For experiment with initial interface y=0.25+0.3x, Figure 5.10 shows the L1 per area density error of the ground truth, baseline and NN meshes solution. The error is computed using ultra fine mesh result as reference. At first 50 steps, NN solutions are close to ground truth solutions and better than the baseline. At time step 50-100 all three meshes solution are very close. Figure 5.11 demonstrates NN meshes at this experiment. We created a single NN in this case based upon training from multiple initial configurations. At the first 50 steps the fluid interface moves very fast therefore NN and ground truth solutions are significantly better than the baseline. At 50-100, the fluid interface is flater and moves slower therefore all three solutions are close.

For experiment with initial interface y=0.1+0.375x, Figure 5.12 shows the density error of ground truth, baseline and NN meshes solution. In steps 1-30, NN error is lower than baseline and close to the ground truth. In steps 31 -70, NN error is close to the baseline and higher than the ground truth. In steps 71-100, error of all three meshes are very close. In this experiment NN does not work as well as last experiment and we will analyse in the next paragraph.

From Figure 5.13 we can find at step 15 the mesh is refined at the fluid interface very well. However in step 25 the mesh is not fine enough inside the red circle. This is the reason that from step 25 the NN solution error arises compared with the ground truth. We try to find reasons that NN fail to make good spatio-temporal predictions here. Firstly, the ground truth mesh is relatively coarse in the red circle. Secondly, because the fluid interface is very sharp, the error estimation is done on the non-uniform mesh then error is interpolate onto the background mesh. Figure 5.14 shows in step 23 in red circle the mesh is relatively coarse (this is reasonable because the ground truth mesh is also relatively coarse here). Because the local mesh is coarse the error is high. In step 24 the mesh is relatively fine and thus the error is relatively low. By spatio-temporal prediction NN reads the error of step 23 and 24 to predict error of step 25. From step 23 to 24 there is a strong tendency that local error decrease therefore the predicted error of 25 is low. The local mesh size of step 25 is large since the predicted error is low. This mechanism results in the jump of mesh size and error.

5.3.3 Compressible flow

The following experiment tests compressible supersonic flow past an obstacle. The Mach numbers of the experiment sets are 1.5-2. In this range, the flow develops shock, which is sharp variation of density and velocity. This demonstrates the performance of our model on compressible flow. The solver used is from FreeFem++ example "a flow with shocks". Compressible Euler equations can be discretized with FEM with flux up-winding scheme but it is not implemented in

FreeFem++. Nevertheless acceptable results can be obtained with the method of characteristics provided that the mean values $\bar{f} = \frac{1}{2}(f^+ + f^-)$ are used at shocks in the scheme.

$$
\begin{aligned}
\partial_t \rho + \bar{u}\nabla\rho + \bar{\rho}\nabla \cdot u = 0 \\
\bar{\rho}\partial_t u + \frac{\bar{\rho u}}{\bar{\rho}}\nabla u + \nabla p = 0 \\
\partial_t \rho + \bar{u}\nabla p + (\gamma - 1)\bar{p}\nabla \cdot u = 0
\end{aligned}
\tag{5.13}
$$

Where γ is the ratio of the heat capacity. $\bar{p} = \frac{1}{2}(p^+ + p^-)$. One solving possibility (used by FreeFem++) is to couple u, p and then update ρ.

To generate a non-uniform mesh, we first interpolate the solution onto the background mesh then do error estimation onto the background mesh.

Problem specification

We do experiments on a fixed geometry (Figure 5.15). The geometry is a 2x1 channel with a radius 0.2 half circle obstacle whose center is at (0.9,0). The inlet is $x = 0$ and the outlet is $x = 2$. The angle of attack is 0. The time step for training problem is $\delta t = 0.005$ and for testing problem is $\delta t = 0.004$. This test the generalizability of the NN to work on a different time step to that with which is was trained. For the training problem the Mach number is 2. For the testing problem, the Mach number is 1.5 at the beginning and increases to 2.0 at step 25 and keeps to step 200. Figure 5.16 demonstrates the density evolution of the test problem. There is a shock (an abrupt discontinuity in the flow field) at the left of the obstacle. With the time increases, the shock moves towards the obstacle and gets straight. A good mesh should be fine and highly anisotropic (the element size along the shock should be significantly larger) at the shock.

Network information

In this example we use only one problem to generate training data. Training takes 3000 epochs and the batch sizes are 32. The input dimension is (298, 16386, 6) and output dimension is (298, 16386, 3). We use local normalisation in this test problem. Training uses mean square error as loss function and Adam as optimizer. The output channel dimensions of each layer are

8, 16, 32, 16, 8 and 3.

Results

A visual comparison of meshes are in Figure 5.19 for time step 200. The background mesh is isotropic and finer near the obstacle. The ground truth, baseline mesh and NN mesh are very similar where the meshes are globally coarse but refined at the shock on the left of the obstacle and at the area behind the obstacle. The NN mesh is relatively coarser on the left of the shock. The reference mesh is globally much finer than the other meshes.

Figure 5.17 shows the root mean square density error of NN mesh, ground mesh and baseline mesh solutions using ultra fine mesh solution as reference. The NN and baseline mesh both performs very close to the ground truth. The baseline is good because the optimal meshes of adjacent time steps are very close. This is because the solver requires the time step to be very small and the shock hardly moves between time steps. Figure 5.18 is a comparison of NN and baseline against the ground truth. At time steps 1-150 the baseline is closer to the ground truth and at 150-200 NN is closer to the ground truth. In conclusion, NN performs good in this test case but the baseline is also good enough.

5.4 Discussion

In this chapter we have introduced a novel parallel computing framework for FE with adaptive mesh refinement. In the conventional method only CPUs are applied for parallel computing of mesh generation and FE computing. We find an opportunity to apply GPU and machine learning to accelerate FE simulation. Our framework allows non-uniform mesh generation and FE computing run in parallel which is potentially highly efficient especially when we have abundant computing resources. In our experiments we have not compared the computational costs of the methods.

Our framework shows different performance at different test cases. In the flow past cylinder case, the NN learns the ground truth very well. The baseline method (the baseline method use last time step solution for mesh adaptation) outperforms the ground truth method. The reason might be the Hessian is not the optimal method for mesh refinement. The NN learns the ground truth better than the baseline reproduces the ground truth. In the two phase flow case, NN guided meshes are close to the ground truth mesh and are better than the baseline meshes.

In the compressible flow case, the movement of the shock is very slow. In this condition, the ground truth solution, baseline solution and NN solution all have close performance. The NN learns the ground truth very well and the baseline also does a good job. The efficient way to do such simulation is to use the baseline method which uses the previous solution to guide new mesh generation. Our approach shows advantage when the optimal mesh varies significantly across time steps.

Some settings are flexible yet critical in this chapter. The choice of background mesh is problem dependent. We recommend to try different background mesh and perhaps more than one mesh work well. We only use uniform background mesh though non-uniform background mesh could also work. The element size should be comparable with the movement of fluid structure in one time step. If the element size of the background mesh is too large, it is hard to guide mesh adaptation. If the element size is too small, it is hard for NN to make spatio-temporal prediction and the computational expense is high. The choice of mesh generator parameters such as smoothness factor is also problem dependent. We set the parameters to ensure the mesh size varies smoothly on the computational domain. The smoothness factor is not too high so that non-uniform mesh is efficient, with the ground truth mesh outperforms the baseline mesh.

The work in this chapter also explores the potential of deep learning. Supervised learning usually requires large volume of training data which creates a problem space. After training, the neural network is expected to solve problems in the problem space. However, in our first experiment, only a sequence of 300 time steps simulation data are used to train a NN. The trained NN is successfully applied on an unseen problem. NNs do not need large volume of training data to converge, a single sequence data with a hundred time steps seems adequate. We only use NN to extrapolate from t-2, t-1 to t in time steps however more options are possible for example from t-4, t-2 to t. Note that even the geometry of the training problem and test problem can be different. When apply on a new problem with new geometry, we can use the trained NN with fixed weights. However the filter should be changed. The reason of the high generalisation ability is the NN learns the relation between features of adjacent time steps rather than directly learning the features. And the relation to learn is consistent so that only a small number of training data supports the good learning results.

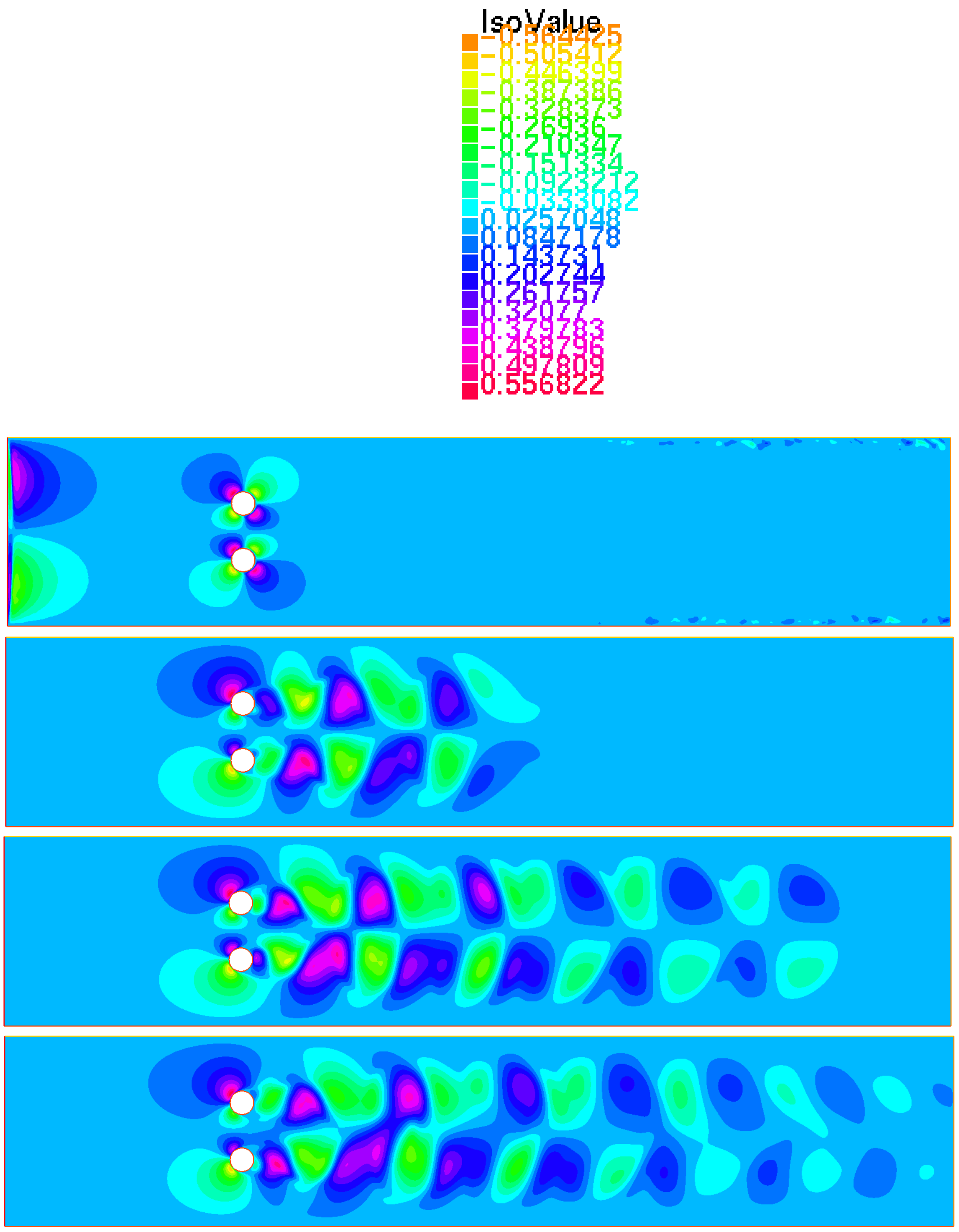


Figure 5.5: The evolution vertical velocity of the flow past cylinders in time steps 2, 100, 200 and 300.

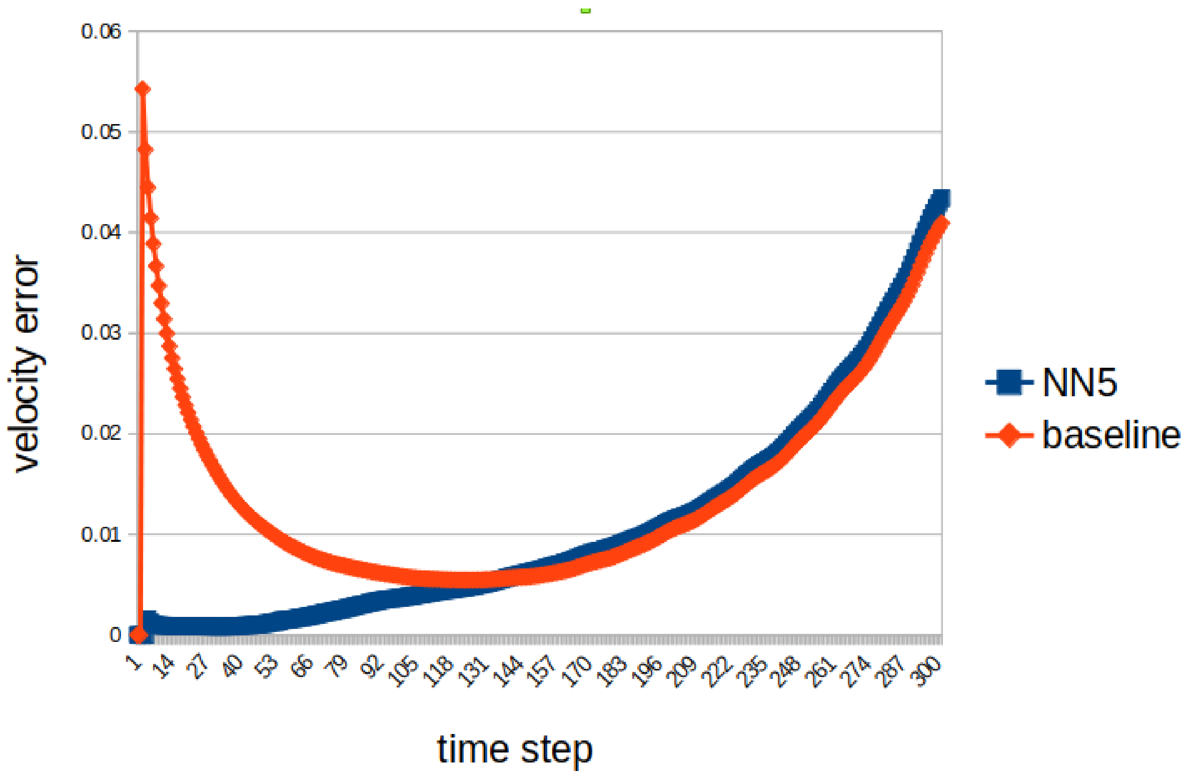


Figure 5.6: The L1 error of velocity per area during 300 time steps of simulation, all the meshes are non-uniform and time dependent. The reference solution is from "ground truth" mesh simulation

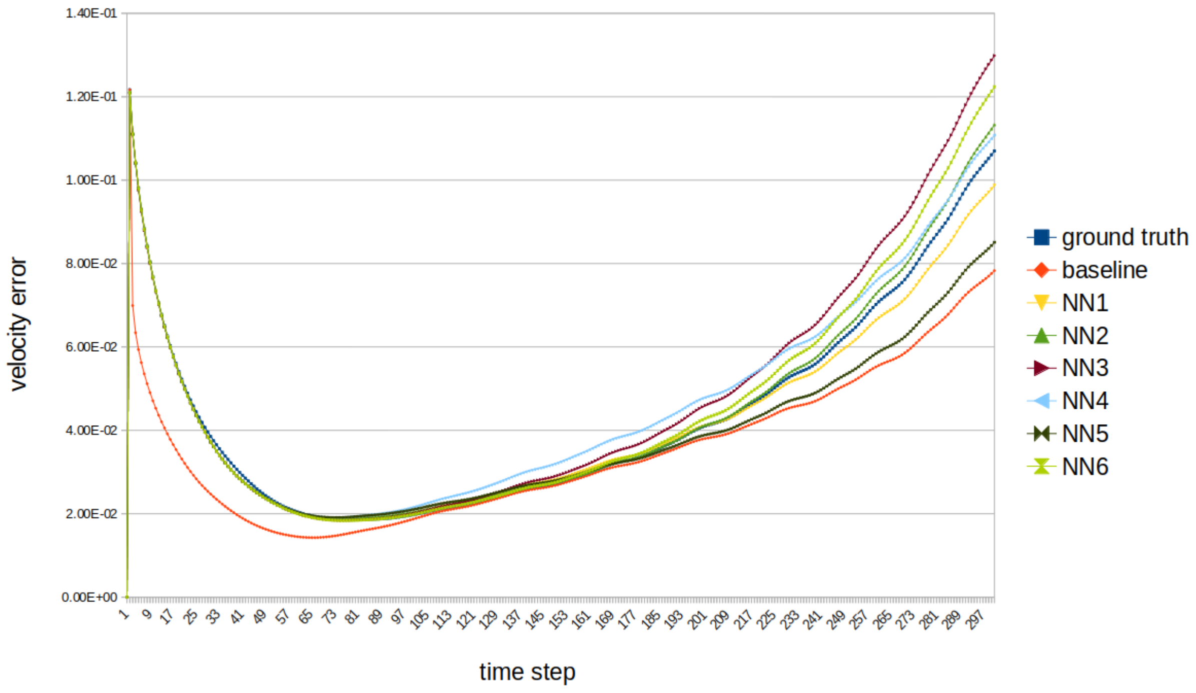


Figure 5.7: The L1 error of velocity per area during 300 time steps of simulation, all the meshes are non-uniform and time dependent. The reference solution is from ultra fine "reference" mesh simulation.

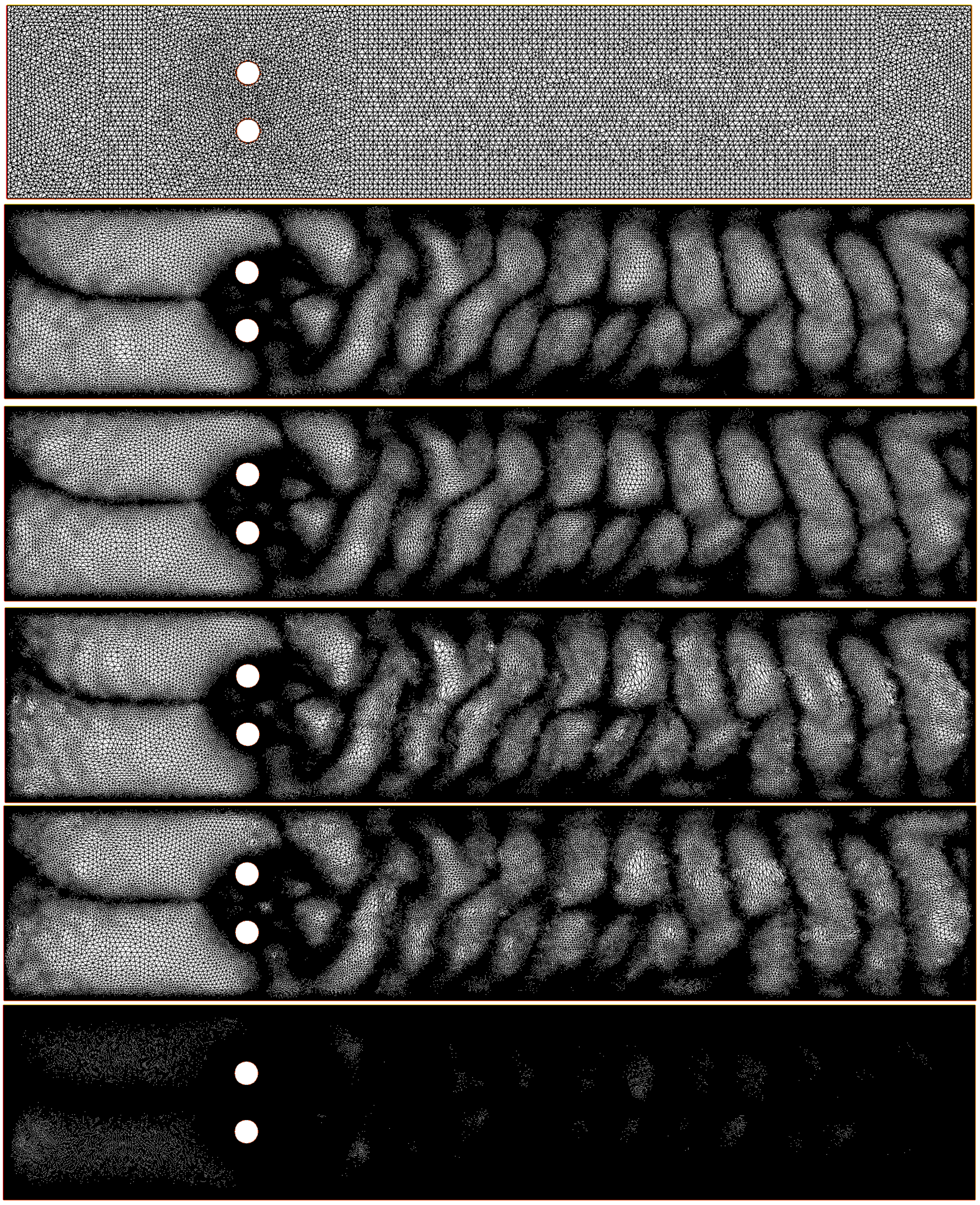


Figure 5.8: From top to bottom are the background mesh, ground truth mesh, baseline mesh, NN3 mesh, NN5 mesh and reference mesh. The reference mesh has around 440000 elements and other meshes have around 130000 elements.

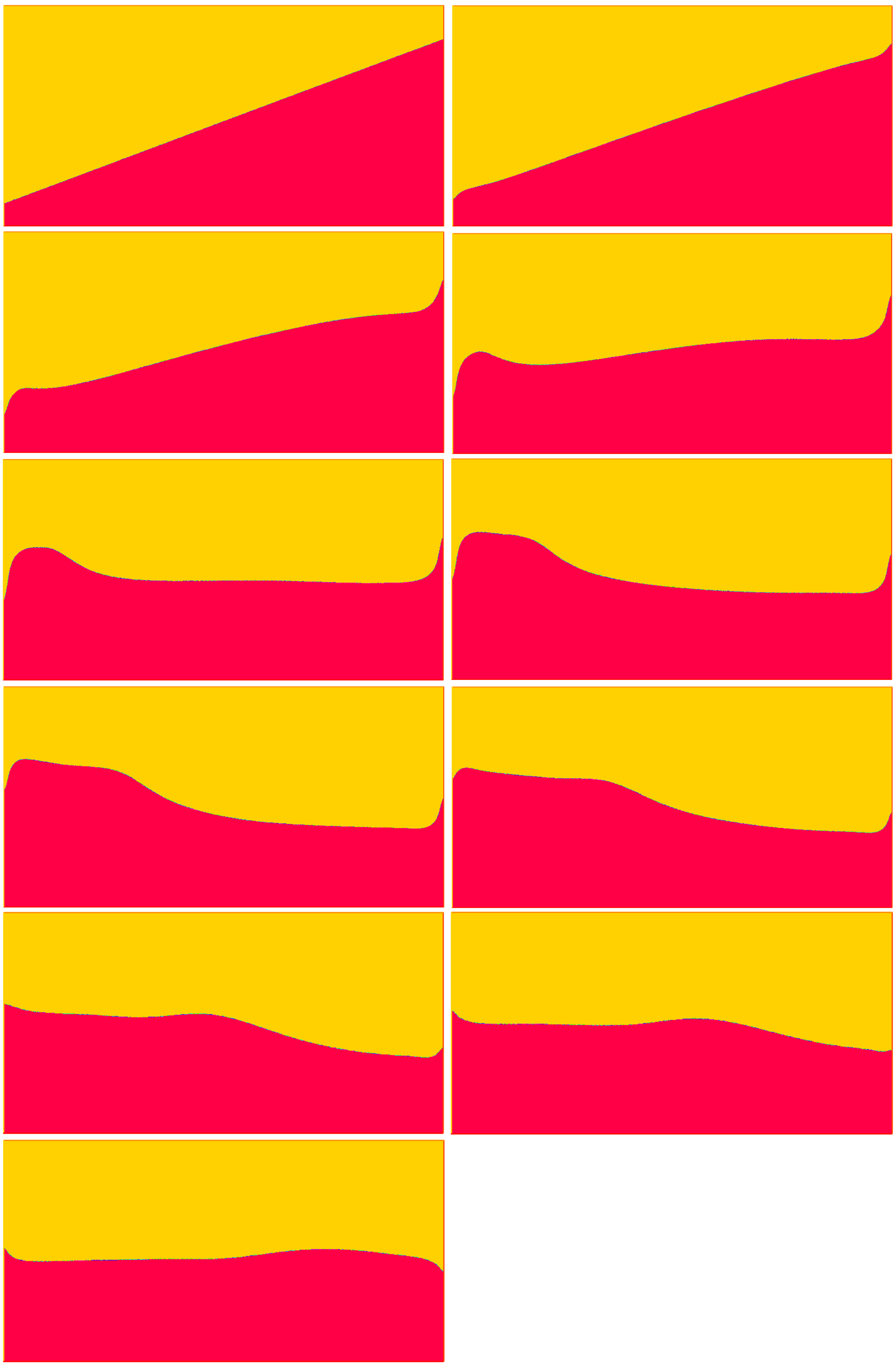


Figure 5.9: The evolution of the sloping two phase flow in time steps 2, 10, 20, 30, 40, 50, 60, 70, 80, 90 and 100. The yellow region is filled with low density flow and the red region is filled with high density flow.

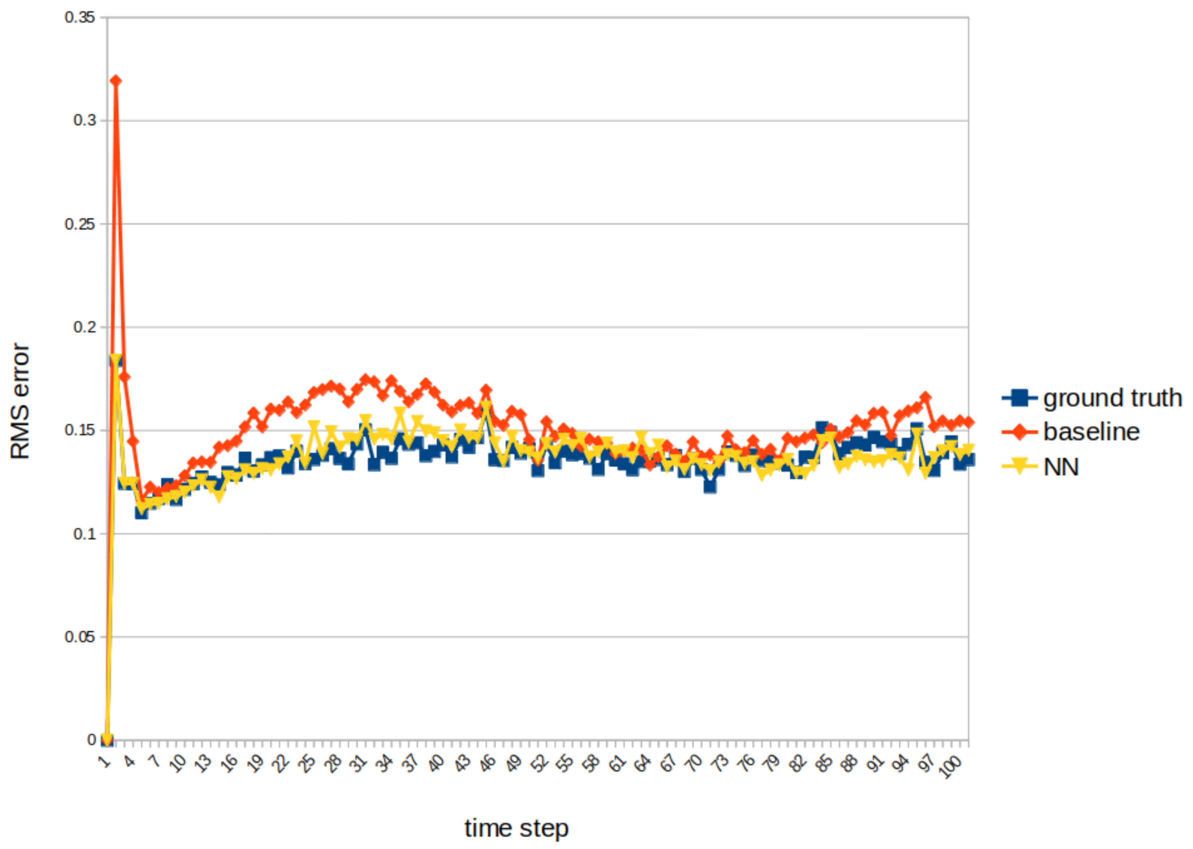


Figure 5.10: The L1 error of density per area during 100 time steps of simulation, all the meshes are non-uniform and time dependent. Initial fluid interface is y=0.25+0.3x.

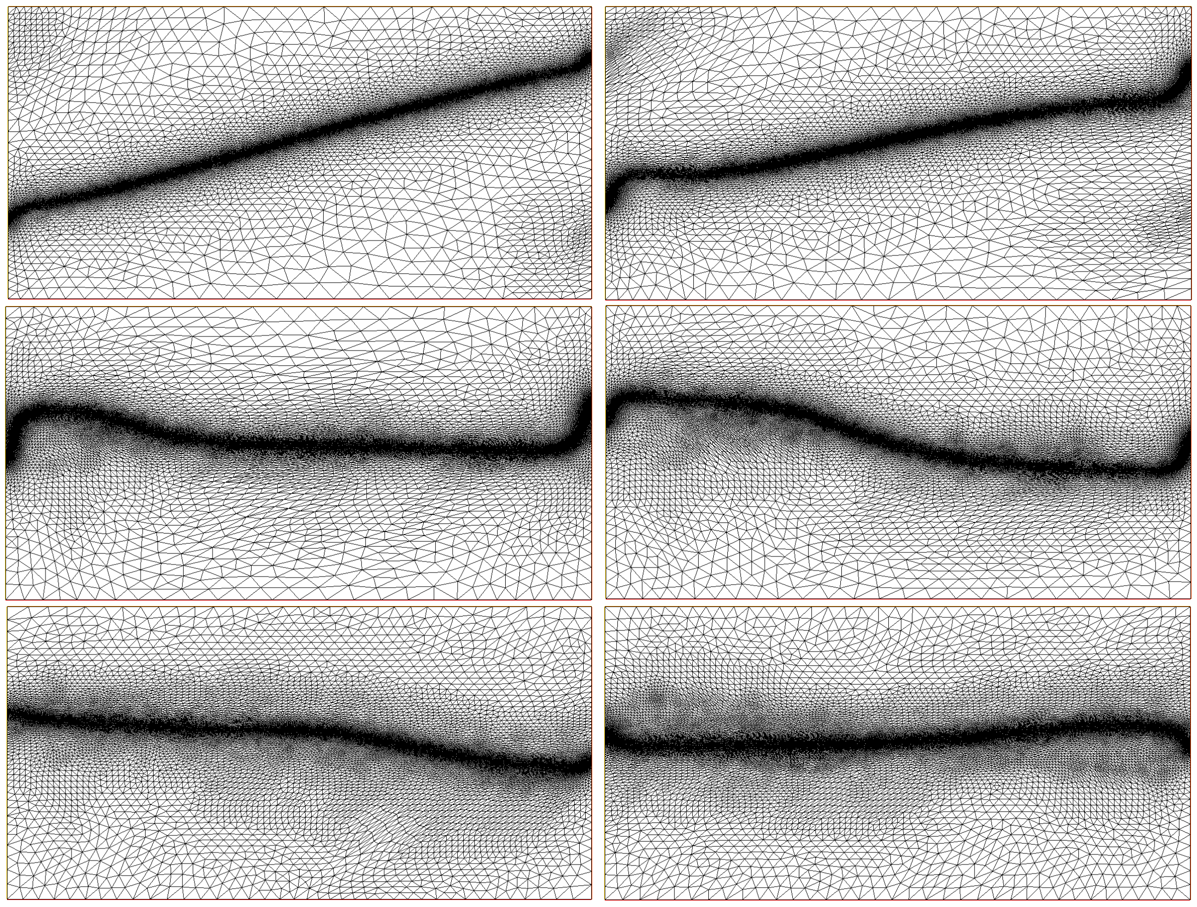


Figure 5.11: The NN mesh of time step 10, 20, 40, 60, 80, 100,initial interface is y=0.25+0.6x.

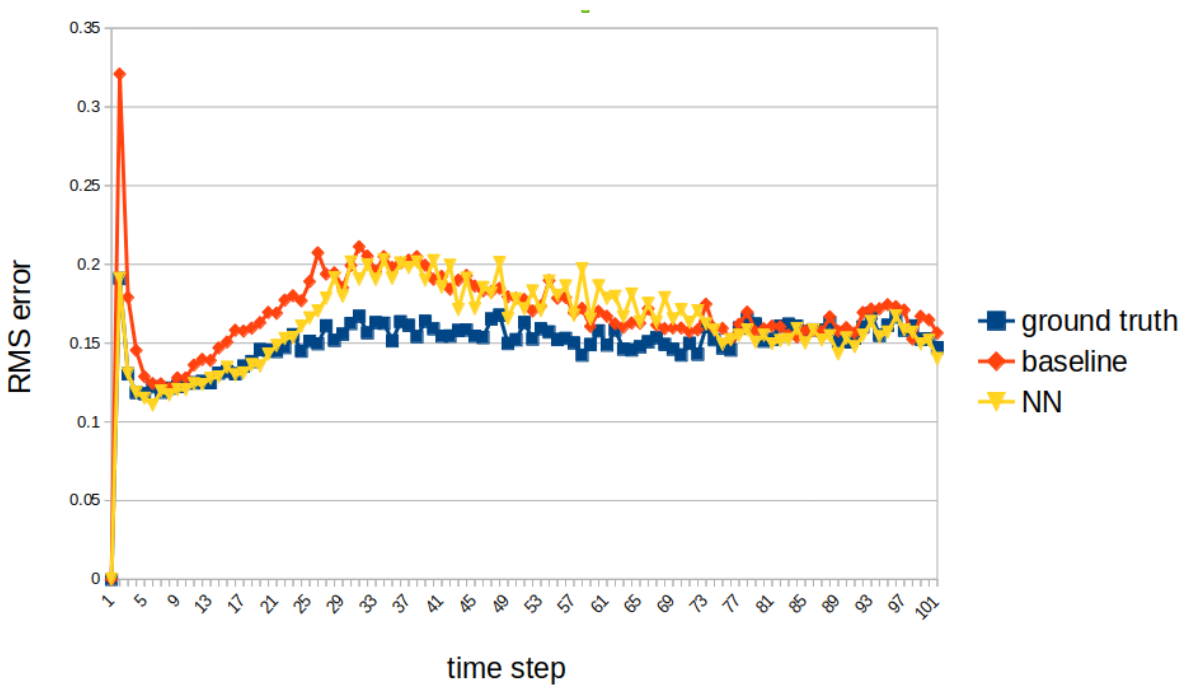


Figure 5.12: The L1 error of density per area during 100 time steps of simulation, all the meshes are non-uniform and time dependent. Initial fluid interface is y=0.1+0.375x.

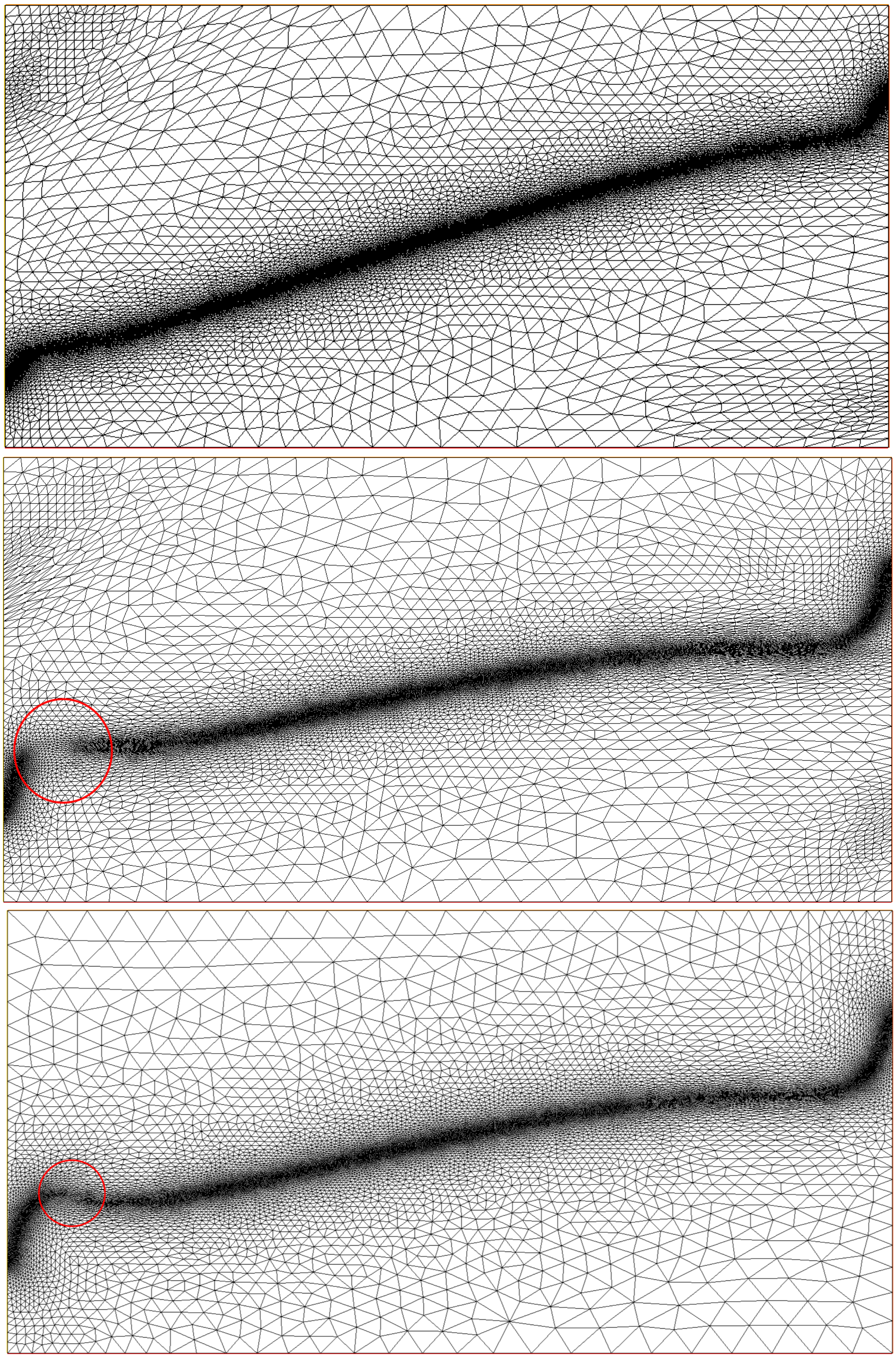


Figure 5.13: From top to bottom are the NN mesh at step 15, NN mesh at step 25 and ground truth mesh at step 24.

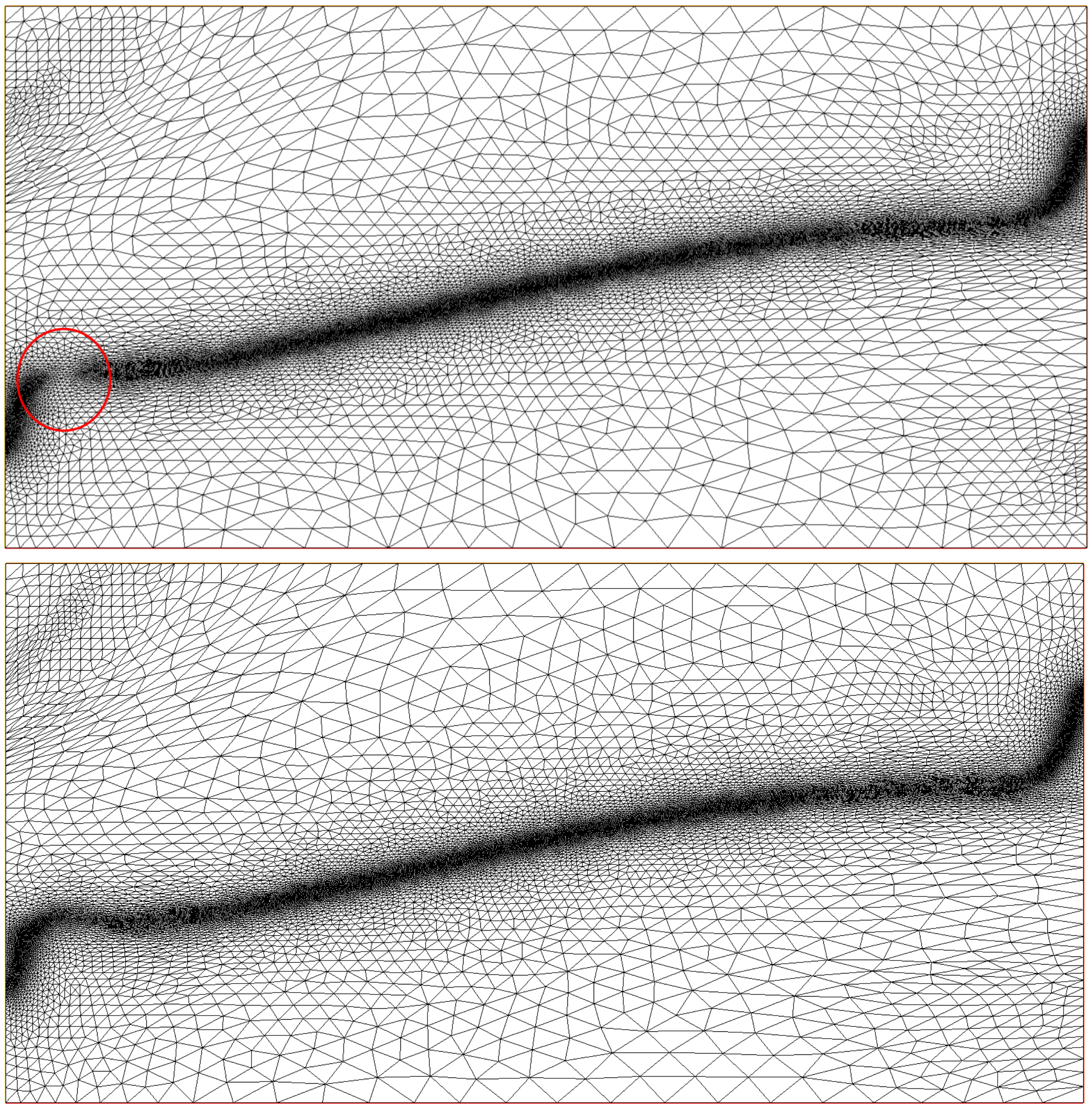


Figure 5.14: Top is the NN mesh at step 23 and bottom is the NN mesh at step 24. The mesh near the interface is not very anisotropic. This is perhaps because at the interface the mesh is isotropic and other region in refined by smoothness.

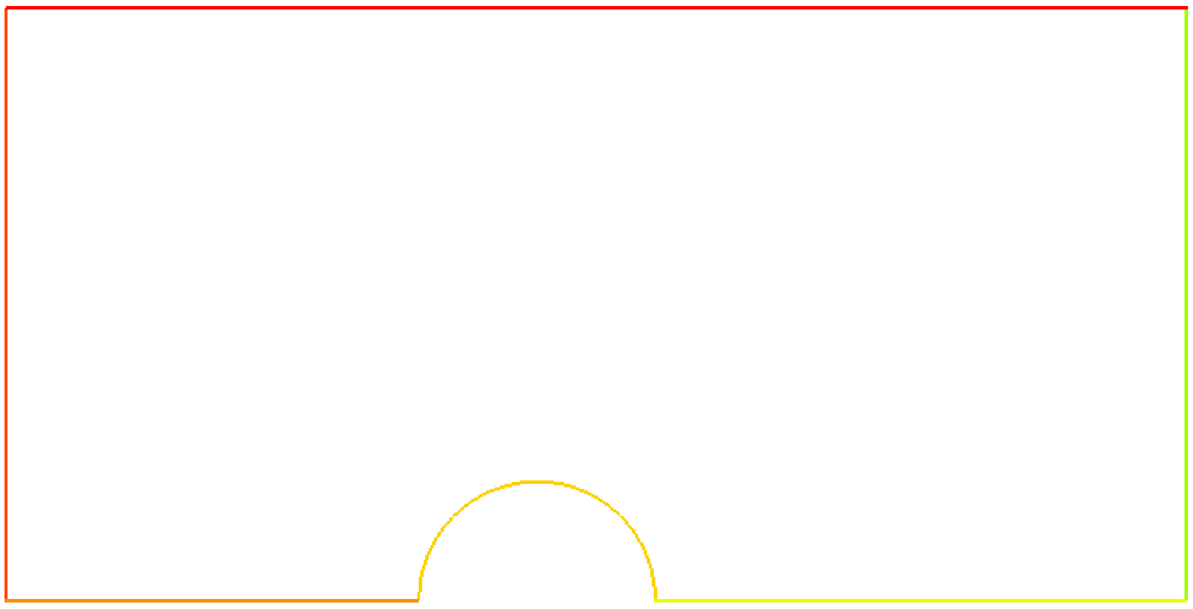


Figure 5.15: The geometry of the compressible flow experiment

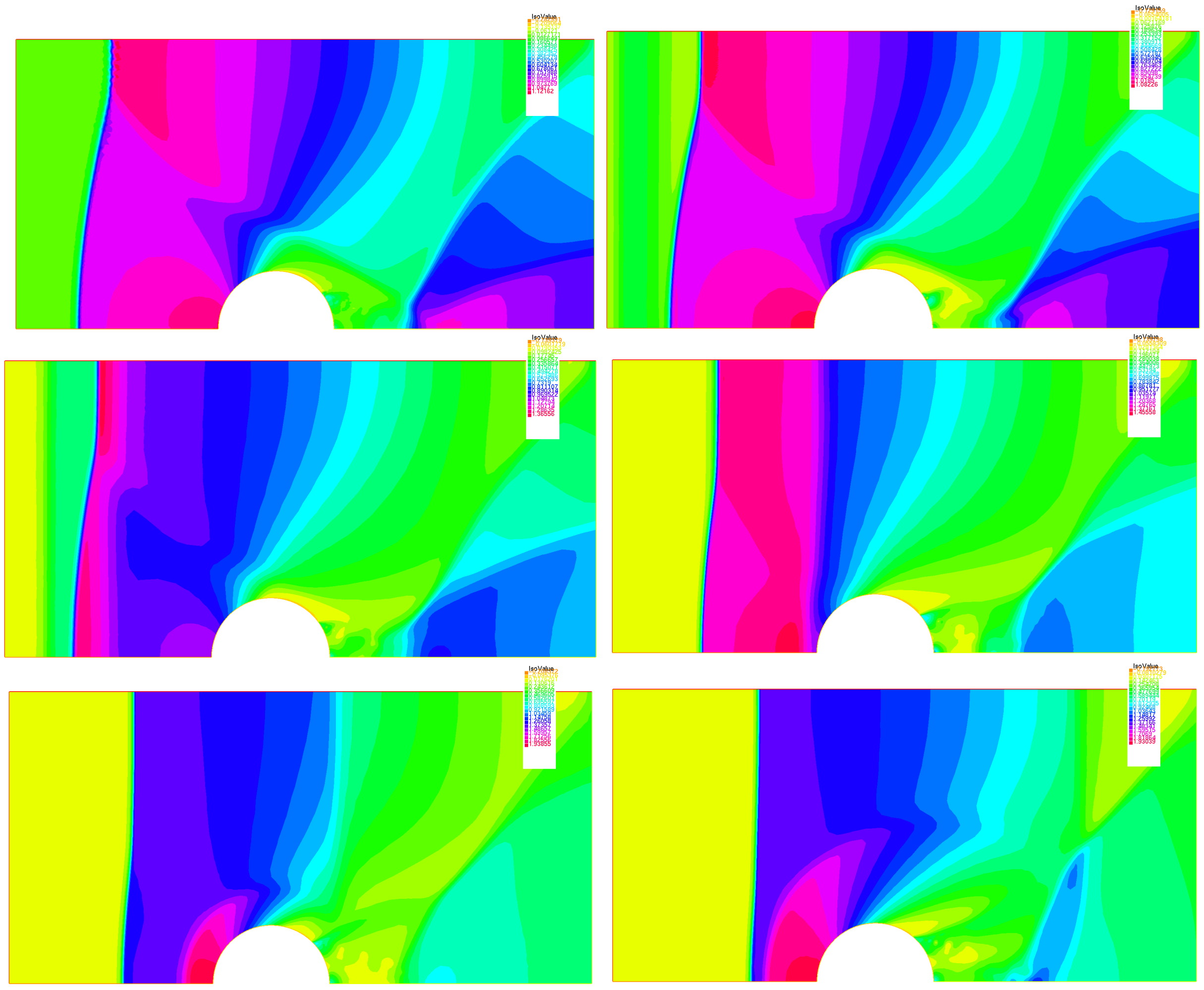


Figure 5.16: The evolution of density of the flow in time steps 0, 25, 50, 100, 150 and 200.

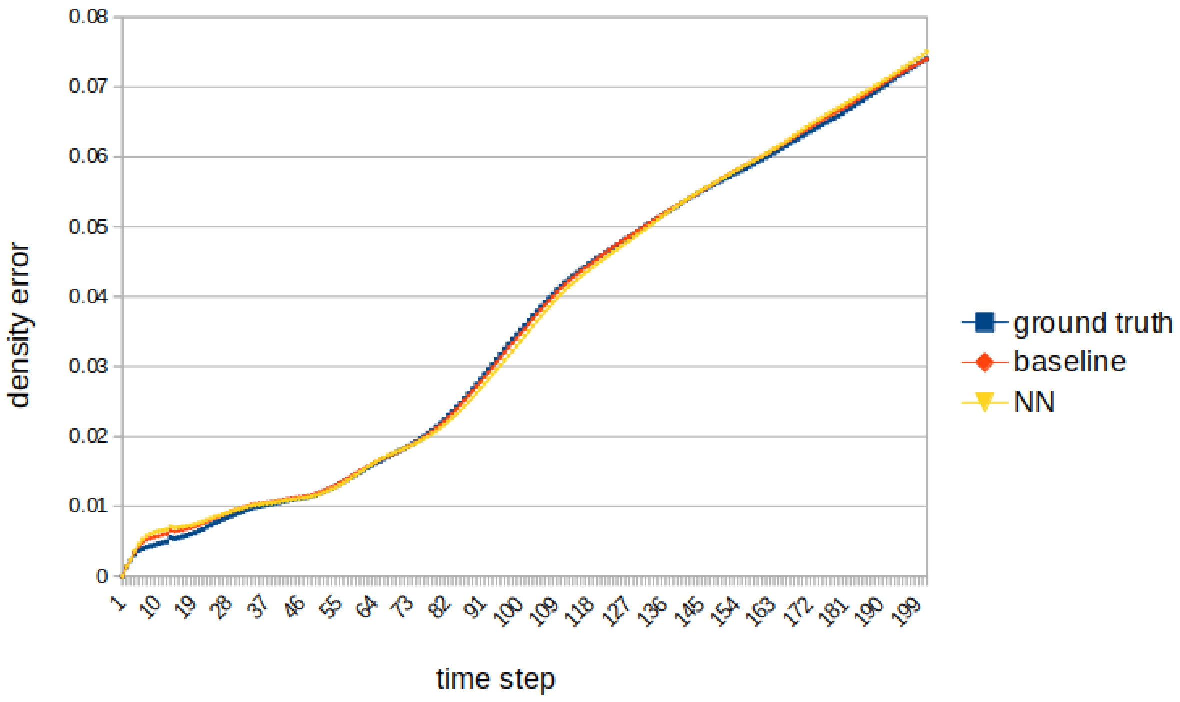


Figure 5.17: The L1 error of density per area during 200 time steps of simulation, all the meshes are non-uniform and time dependent.

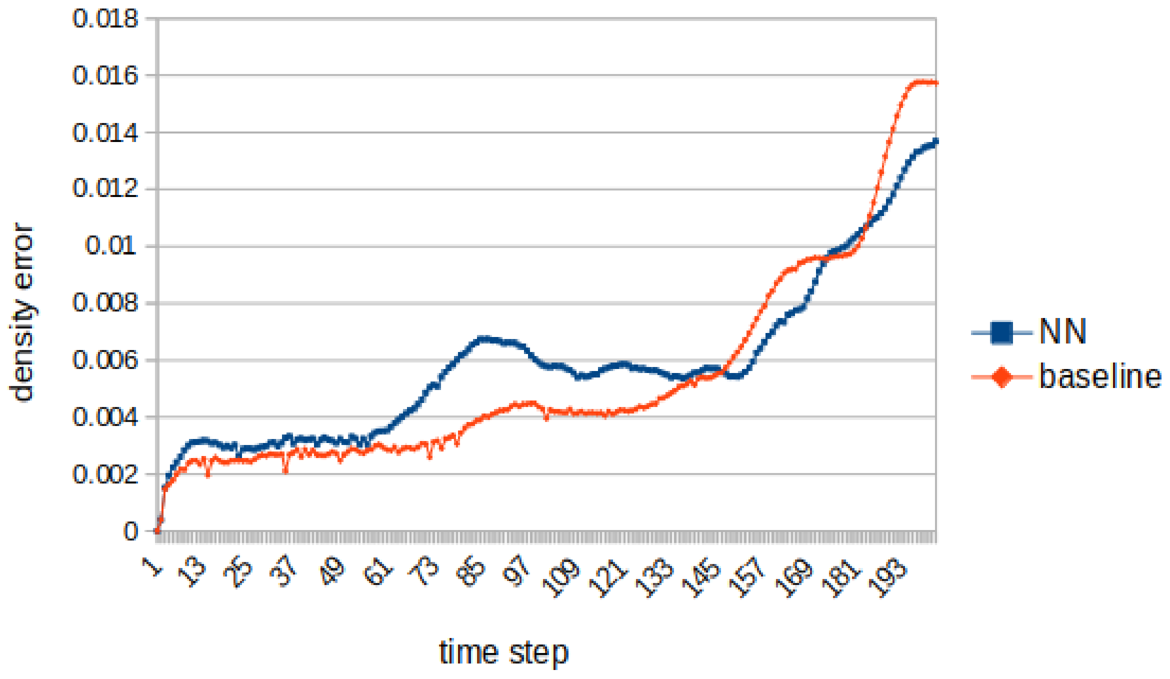


Figure 5.18: The L1 error of density per area during 200 time steps of simulation, all the meshes are non-uniform and time dependent. The reference solution is from "ground truth" mesh simulation

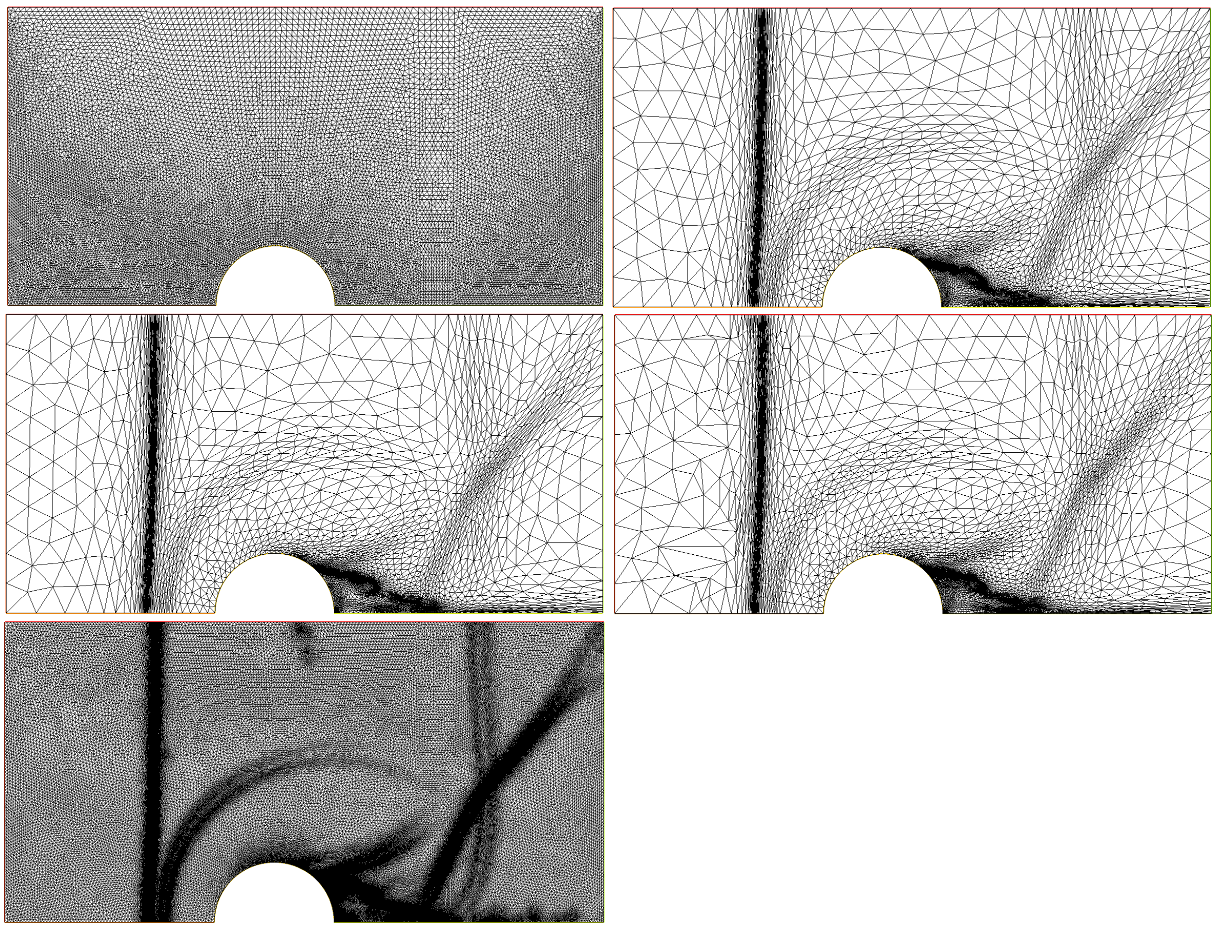


Figure 5.19: From left top to bottom are the background mesh, ground truth mesh, baseline mesh, NN mesh and reference mesh. We refine the mesh according to the Hessian of the density.

Chapter 6

Conclusion and future work

Machine learning for solving PDEs is an open question that is attracting significant research interest. There are opportunities in supervised and unsupervised PDE solvers. In this thesis we choose instead to use traditional finite element solvers but to make use of machine learning to support high-quality non-uniform mesh generation. Early research along these lines usually requires expert knowledge and can be only used for very specific problems. Recently, ML aided CFD mesh generation (Huang, Krügener, et al. 2021, Tingfan et al. 2022) has been studied where transient mesh adaptation is very difficult.

6.1 Conclusion

In this thesis we have studied deep learning guided non-uniform FE mesh generation. For 2D steady PDEs, we used the idea of previous studies that use neural networks to predict element size distribution. With the implementation of mean value coordinates, we use NN for instant non-uniform mesh generation, with respecting to domain geometry and PDE parameters. By analysing the FE solution of different meshes, the NN mesh is close to the ground truth (meshes refined by *a posteriori* error estimate) and much better than the uniform mesh. Our framework does not require prior knowledge such as the mesh should be fine at the corner and can be applied on a wide range of steady PDEs. We have extended our framework to 3D and tested it on 4 different 3D linear elasticity problems. NNs work well when predicting meshes regarding different geometries, boundary conditions and material propeerties. Althrough we only applied above framework to linear PDE systems, the approach should be equally applicable to nonlinear

PDE systems too.

For transient PDE, we use GCN (graph convolutional networks) to do spatio-temporal prediction and predict mesh metric from previous mesh metrics. Our approach enables mesh generation for next time step in parallel with computing the solution in the current step. The meshes guided by NN have potential to be better than the baseline non-uniform mesh and close to the ground truth. Our method first utilises the data produced from previous time steps and therefore succeeds on transient mesh generation.

6.2 Future work

There are many potential ways in which the research undertake in this thesis could be extended in the future. Some specific examples include the following suggestions.

1. For steady PDEs, only polygons in 2D and polyhedra in 3D have been meshed. A wider range of geometries such as geometries with curves and curved surface should be considered and more complex NNs should be used to realise this target.

2. For steady PDEs, embedding pre-trained NNs in computer aided design (CAD) software could save meshing costs. It is a question how to use CAD format as the input to the NNs.

3. For transient PDEs, we use serial computing to simulate parallel computing as a proof of concept. We run all the processes on a single CPU thread. In the future, real parallel computing should be applied and accelerating performace should be tested.

4. For transient PDEs, we only solve problems with fixed domain geometries, it is a challenge problem to solve problems with variant domain geometries. The way to use the background mesh should be changed since the background mesh is no longer fixed.

References

Ahrens, James, Geveci, Berk, Law, Charles, Hansen, C, and Johnson, C (2005). “36-paraview: An end-user tool for large-data visualization”. In: *The visualization handbook* 717, pp. 50038–1.

Ainsworth, Mark and Oden (1997). “A posteriori error estimation in finite element analysis”. In: *Computer methods in applied mechanics and engineering* 142.1-2, pp. 1–88.

– (2011). *A posteriori error estimation in finite element analysis.* Vol. 37. John Wiley & Sons.

Alfonzetti, S, Coco, S, Cavalieri, S, and Malgeri, M (1996). “Automatic mesh generation by the let-it-grow neural network”. In: *IEEE transactions on magnetics* 32.3, pp. 1349–1352.

Amestoy, Patrick R, Duff, Iain S, L’Excellent, Jean-Yves, and Koster, Jacko (2000). “MUMPS: a general purpose distributed memory sparse solver”. In: *International Workshop on Applied Parallel Computing.* Springer, pp. 121–130.

Angermann, Lutz (2002). “A posteriori error estimates for FEM with violated Galerkin orthogonality”. In: *Numerical Methods for Partial Differential Equations: An International Journal* 18.2, pp. 241–259.

Apel, Thomas, Benedix, Olaf, Sirch, Dieter, and Vexler, Boris (2011). “A priori mesh grading for an elliptic problem with Dirac right-hand side”. In: *SIAM journal on numerical analysis* 49.3, pp. 992–1005.

Apel, Thomas, Grosman, Sergei, Jimack, Peter K, and Meyer, Arnd (2004). “A new methodology for anisotropic mesh refinement based upon error gradients”. In: *Applied Numerical Mathematics* 50.3-4, pp. 329–341.

Babuška, Ivo and Rheinboldt, Werner C (1978). “A-posteriori error estimates for the finite element method”. In: *International journal for numerical methods in engineering* 12.10, pp. 1597–1615.

Bahreininejad, A, Topping, BHV, and Khan, AI (1996). “Finite element mesh partitioning using neural networks”. In: *Advances in Engineering Software* 27.1-2, pp. 103–115.

Baines, Mike J, Hubbard, ME, and Jimack, PK (2005). “A moving mesh finite element algorithm for the adaptive solution of time-dependent partial differential equations with moving boundaries”. In: *Applied Numerical Mathematics* 54.3-4, pp. 450–469.

Baker, Timothy J (2005). “Mesh generation: Art or science?” In: *Progress in Aerospace Sciences* 41.1, pp. 29–63.

Bank, Randolph E and Weiser, Alan (1985). “Some a posteriori error estimators for elliptic partial differential equations”. In: *Mathematics of computation* 44.170, pp. 283–301.

Bar-Sinai, Yohai, Hoyer, Stephan, Hickey, Jason, and Brenner, Michael P (2019). “Learning data-driven discretizations for partial differential equations”. In: *Proceedings of the National Academy of Sciences* 116.31, pp. 15344–15349.

Bengio, Yoshua, Courville, Aaron, and Vincent, Pascal (2013). “Representation learning: A review and new perspectives”. In: *IEEE transactions on pattern analysis and machine intelligence* 35.8, pp. 1798–1828.

Bern, Marshall W and Plassmann, Paul E (2000). “Mesh Generation.” In: *Handbook of computational geometry* 38.

Bohn, Jan and Feischl, Michael (2021). “Recurrent neural networks as optimal mesh refinement strategies”. In: *Computers & Mathematics with Applications* 97, pp. 61–76.

Bosman, Peter AN and Thierens, Dirk (2000). “Negative log-likelihood and statistical hypothesis testing as the basis of model selection in IDEAs”. In.

Bruna, Joan, Zaremba, Wojciech, Szlam, Arthur, and LeCun, Yann (2013). “Spectral networks and locally connected networks on graphs”. In: *arXiv preprint arXiv:1312.6203*.

Brunton, Steven L, Noack, Bernd R, and Koumoutsakos, Petros (2020). “Machine learning for fluid mechanics”. In: *Annual review of fluid mechanics* 52, pp. 477–508.

Caendish, James C, Field, David A, and Frey, William H (1985). “An apporach to automatic three-dimensional finite element mesh generation”. In: *International journal for numerical methods in engineering* 21.2, pp. 329–347.

Cai, Shengze, Mao, Zhiping, Wang, Zhicheng, Yin, Minglang, and Karniadakis, George Em (2021). “Physics-informed neural networks (PINNs) for fluid mechanics: A review”. In: *Acta Mechanica Sinica* 37.12, pp. 1727–1738.

Carstensen, Carsten and Funken, Stefan A (2001). “Averaging technique for FE–a posteriori error control in elasticity. Part I: Conforming FEM”. In: *Computer Methods in Applied Mechanics and Engineering* 190.18-19, pp. 2483–2498.

Castro-Dıaz, MJ, Hecht, Frédéric, Mohammadi, Bijan, and Pironneau, O (1997). “Anisotropic unstructured mesh adaption for flow simulations”. In: *International Journal for Numerical Methods in Fluids* 25.4, pp. 475–491.

Chan, Jesse, Wang, Zheng, Modave, Axel, Remacle, Jean-Francois, and Warburton, Tim (2016). “GPU-accelerated discontinuous Galerkin methods on hybrid meshes”. In: *Journal of Computational Physics* 318, pp. 142–168.

Chan, Scholz, Felix, and Takacs, Thomas (2022). “Locally refined quad meshing for linear elasticity problems based on convolutional neural networks”. In: *Engineering with Computers* 38.5, pp. 4631–4652.

Chang-Hoi, Ahn, Sang-Soo, Lee, Hyuek-Jae, Lee, and Soo-Young, Lee (1991). “A self-organizing neural network approach for automatic mesh generation”. In: *IEEE transactions on magnetics* 27.5, pp. 4201–4204.

Chedid, Riad and Najjar, Nayeff (1996). “Automatic finite-element mesh generation using artificial neural networks-Part I: Prediction of mesh density”. In: *IEEE Transactions on Magnetics* 32.5, pp. 5173–5178.

Chen and Křižek, Michal (2006). "What is the smallest possible constant in Céa's lemma?" In: *Applications of Mathematics* 51.2, pp. 129–144.

Chen, Liu, Pang, Yufei, Chen, Jie, Chi, Lihua, and Gong, Chunye (2020). "Developing a new mesh quality evaluation method based on convolutional neural network". In: *Engineering Applications of Computational Fluid Mechanics* 14.1, pp. 391–400.

Chen, Rubanova, Yulia, Bettencourt, Jesse, and Duvenaud, David K (2018). "Neural ordinary differential equations". In: *Advances in neural information processing systems*, pp. 6571–6583.

Chen, Wenqian, Wang, Qian, Hesthaven, Jan S, and Zhang, Chuhua (2021). "Physics-informed machine learning for reduced-order modeling of nonlinear problems". In: *Journal of computational physics* 446, p. 110666.

Chen, Xinhai, Li, Tiejun, Wan, Qian, He, Xiaoyu, Gong, Chunye, Pang, Yufei, and Liu, Jie (2022). "MGNet: a novel differential mesh generation method based on unsupervised neural networks". In: *Engineering with Computers* 38.5, pp. 4409–4421.

Cheng, Siu-Wing, Dey, Tamal K, and Shewchuk, Jonathan (2012). *Delaunay mesh generation.* CRC Press.

Chollet, François et al. (2015). *Keras.* `https://keras.io`.

Chorin, Alexandre Joel (1967). "The numerical solution of the Navier-Stokes equations for an incompressible fluid". In: *Bulletin of the American Mathematical Society* 73.6, pp. 928–931.

– (1968). "Numerical solution of the Navier-Stokes equations". In: *Mathematics of computation* 22.104, pp. 745–762.

Choudhary and al., et (2022). "Recent advances and applications of deep learning methods in materials science". In: *npj Computational Materials* 8.1, p. 59.

Cybenko, George (1989). "Approximation by superpositions of a sigmoidal function". In: *Mathematics of control, signals and systems* 2.4, pp. 303–314.

Da'u, Aminu and Salim, Naomie (2020). "Recommendation system based on deep learning methods: a systematic review and new directions". In: *Artificial Intelligence Review* 53.4, pp. 2709–2748.

De SR Gago, JP, Kelly, Donald W, Zienkiewicz, Olgierd C, and Babuska, I (1983). "A posteriori error analysis and adaptive processes in the finite element method: Part II—Adaptive mesh refinement". In: *International journal for numerical methods in engineering* 19.11, pp. 1621–1656.

De Vuyst, Florian (2013). "Numerical modeling of transport problems using freefem++ software–with examples in biology, CFD, traffic flow and energy transfer". In.

Dolšak, Bojan, Jezernik, Anton, and Bratko, Ivan (1994). "A knowledge base for finite element mesh design". In: *Artificial intelligence in engineering* 9.1, pp. 19–27.

Dörfler, Willy (1996). "A convergent adaptive algorithm for Poisson's equation". In: *SIAM Journal on Numerical Analysis* 33.3, pp. 1106–1124.

Duvenaud, David K, Maclaurin, Dougal, Iparraguirre, Jorge, Bombarell, Rafael, Hirzel, Timothy, Aspuru-Guzik, Alán, and Adams, Ryan P (2015). "Convolutional networks on graphs for learning molecular fingerprints". In: *Advances in neural information processing systems* 28.

Dyck, DN, Lowther, DA, and McFee, S (1992). "Determining an approximate finite element mesh density using neural network techniques". In: *IEEE transactions on magnetics* 28.2, pp. 1767–1770.

Falini, Antonella and Sampoli, Maria Lucia (2021). "Adaptive Refinement in Advection–Diffusion Problems by Anomaly Detection: A Numerical Study". In: *Algorithms* 14.11, p. 328.

Fidkowski, Krzysztof J and Chen (2021). "Metric-based, goal-oriented mesh adaptation using machine learning". In: *Journal of Computational Physics* 426, p. 109957.

Floater, Michael S (2003). "Mean value coordinates". In: *Computer aided geometric design* 20.1, pp. 19–27.

Floater, Michael S, Kós, Géza, and Reimers, Martin (2005). "Mean value coordinates in 3D". In: *Computer Aided Geometric Design* 22.7, pp. 623–631.

Geneva, Nicholas and Zabaras, Nicholas (2020). “Modeling the dynamics of PDE systems with physics-constrained deep auto-regressive networks”. In: *Journal of Computational Physics* 403, p. 109056.

Geuzaine, Christophe and Remacle, Jean-François (2009). “Gmsh: A 3-D finite element mesh generator with built-in pre-and post-processing facilities”. In: *International journal for numerical methods in engineering* 79.11, pp. 1309–1331.

Gholamalinezhad, Hossein and Khosravi, Hossein (2020). “Pooling methods in deep neural networks, a review”. In: *arXiv preprint arXiv:2009.07485*.

Gillette, Andrew, Keith, Brendan, and Petrides, Socratis (2022). “Learning robust marking policies for adaptive mesh refinement”. In: *arXiv preprint arXiv:2207.06339*.

Golub, Gene H and Van Loan, Charles F (2013). *Matrix computations*. JHU press.

Goodfellow, Ian, Bengio, Yoshua, and Courville, Aaron (2016). *Deep learning*. MIT press.

Götz, Markus and Anzt, Hartwig (2018). “Machine learning-aided numerical linear algebra: Convolutional neural networks for the efficient preconditioner generation”. In: *2018 IEEE/ACM 9th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems (scalA)*. IEEE, pp. 49–56.

Grätsch, Thomas and Bathe, Klaus-Jürgen (2005). “A posteriori error estimation techniques in practical finite element analysis”. In: *Computers & structures* 83.4-5, pp. 235–265.

Gu, Jiuxiang, Wang, Zhenhua, Kuen, Jason, Ma, Lianyang, Shahroudy, Amir, Shuai, Bing, Liu, Ting, Wang, Xingxing, Wang, Gang, Cai, Jianfei, et al. (2018). “Recent advances in convolutional neural networks”. In: *Pattern recognition* 77, pp. 354–377.

Guo, Guodong and Zhang (2019). “A survey on deep learning based face recognition”. In: *Computer vision and image understanding* 189, p. 102805.

Hammond, David K, Vandergheynst, Pierre, and Gribonval, Rémi (2011). “Wavelets on graphs via spectral graph theory”. In: *Applied and Computational Harmonic Analysis* 30.2, pp. 129–150.

Han, Jiequn, Jentzen, Arnulf, and E, Weinan (2018). “Solving high-dimensional partial differential equations using deep learning”. In: *Proceedings of the National Academy of Sciences* 115.34, pp. 8505–8510.

Hecht, Frédéric (1998). “BAMG: bidimensional anisotropic mesh generator”. In: *User Guide. INRIA, Rocquencourt* 17.

– (2012). “New development in FreeFem++”. In: *Journal of numerical mathematics* 20.3-4, pp. 251–266.

Higham, Nicholas J (2009). “Cholesky factorization”. In: *Wiley interdisciplinary reviews: computational statistics* 1.2, pp. 251–254.

Hormann, Kai and Floater, Michael S (2006). “Mean value coordinates for arbitrary planar polygons”. In: *ACM Transactions on Graphics (TOG)* 25.4, pp. 1424–1441.

Hsu, Feng-Hsiung (2002). *Behind Deep Blue: Building the computer that defeated the world chess champion.* Princeton University Press.

Huang (2005). “Metric tensors for anisotropic mesh generation”. In: *Journal of computational physics* 204.2, pp. 633–665.

Huang and al., et (July 2017). “Densely Connected Convolutional Networks”. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2261–2269.

Huang, Krügener, Moritz, Brown, Alistair, Menhorn, Friedrich, Bungartz, Hans-Joachim, and Hartmann, Dirk (2021). “Machine learning-based optimal mesh generation in computational fluid dynamics”. In: *arXiv preprint arXiv:2102.12923.*

Huang, Weizhang, Ren, Yuhe, and Russell, Robert D (1994). “Moving mesh methods based on moving mesh partial differential equations”. In: *Journal of Computational Physics* 113.2, pp. 279–290.

Iqbal, Saeed and Carey, Graham F (2002). “Neural nets for mesh assessment”. In: *TICAM Report*, pp. 02–02.

Ito, Yasushi, Shih, Alan M, and Soni, Bharat K (2009). “Octree-based reasonable-quality hexahedral mesh generation using a new set of refinement templates”. In: *International Journal for Numerical Methods in Engineering* 77.13, pp. 1809–1833.

Kaasschieter, Erik F (1988). “Preconditioned conjugate gradients for solving singular systems”. In: *Journal of Computational and Applied mathematics* 24.1-2, pp. 265–275.

Karniadakis, George Em, Kevrekidis, Ioannis G, Lu, Lu, Perdikaris, Paris, Wang, Sifan, and Yang, Liu (2021). “Physics-informed machine learning”. In: *Nature Reviews Physics* 3.6, pp. 422–440.

Kato, Chisachi, Yamade, Yoshinobu, Nagano, Katsuhiro, Kumahata, Kiyoshi, Minami, Kazuo, and Nishikawa, Tatsuo (2020). “Toward realization of numerical towing-tank tests by wall-resolved large eddy simulation based on 32 billion grid finite-element computation”. In: *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, pp. 1–13.

Kingma, Diederik P and Ba, Jimmy (2015). “Adam: A method for stochastic optimization”. In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings.*

Kipf, Thomas N and Welling, Max (2017). “Semi-supervised classification with graph convolutional networks”. In: *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings.* OpenReview.net.

Kita, Eisuke and Kamiya, Norio (2001). “Error estimation and adaptive mesh refinement in boundary element method, an overview”. In: *Engineering Analysis with Boundary Elements* 25.7, pp. 479–495.

Kochkov, Dmitrii, Smith, Jamie A, Alieva, Ayya, Wang, Qing, Brenner, Michael P, and Hoyer, Stephan (2021). “Machine learning–accelerated computational fluid dynamics”. In: *Proceedings of the National Academy of Sciences* 118.21, e2101784118.

Kong, Fanwei, Wilson, Nathan, and Shadden, Shawn (2021). “A deep-learning approach for direct whole-heart mesh reconstruction”. In: *Medical image analysis* 74, p. 102222.

Krizhevsky, Alex, Sutskever, Ilya, and Hinton, Geoffrey E (2012). “Imagenet classification with deep convolutional neural networks”. In: *Advances in neural information processing systems*, pp. 1097–1105.

Lagaris, Isaac E, Likas, Aristidis, and Fotiadis, Dimitrios I (1998). “Artificial neural networks for solving ordinary and partial differential equations”. In: *IEEE transactions on neural networks* 9.5, pp. 987–1000.

Lazarov, Raytcho, Repin, Sergey, and Tomar, Satyendra K (2009). “Functional a posteriori error estimates for discontinuous Galerkin approximations of elliptic problems”. In: *Numerical Methods for Partial Differential Equations* 25.4, pp. 952–971.

Ho-Le, K (1988). “Finite element mesh generation methods: a review and classification”. In: *Computer-aided design* 20.1, pp. 27–38.

LeCun, Yann, Bengio, Yoshua, and Hinton, Geoffrey (2015). “Deep learning”. In: *nature* 521.7553, pp. 436–444.

Lee, John Boaz, Rossi, Ryan A, Kim, Sungchul, Ahmed, Nesreen K, and Koh, Eunyee (2019). “Attention models in graphs: A survey”. In: *ACM Transactions on Knowledge Discovery from Data (TKDD)* 13.6, pp. 1–25.

Lei, Na, Li, Zezeng, Xu, Zebin, Li, Ying, and Gu, Xianfeng (2023). “What’s the Situation With Intelligent Mesh Generation: A Survey and Perspectives”. In: *IEEE Transactions on Visualization and Computer Graphics.*

Linardatos, Pantelis, Papastefanopoulos, Vasilis, and Kotsiantis, Sotiris (2020). “Explainable ai: A review of machine learning interpretability methods”. In: *Entropy* 23.1, p. 18.

Liu, Li, Shaofan, and Park, Harold S (2022). “Eighty years of the finite element method: Birth, evolution, and future”. In: *Archives of Computational Methods in Engineering* 29.6, pp. 4431–4453.

Lo, SH0587 (1985). “A new mesh generation scheme for arbitrary planar domains”. In: *International journal for numerical methods in engineering* 21.8, pp. 1403–1426.

Long, Zichao, Lu, Yiping, Ma, Xianzhong, and Dong, Bin (2018). “Pde-net: Learning pdes from data”. In: *International conference on machine learning*. PMLR, pp. 3208–3216.

Lowther, DA and Dyck, DN (1993). “A density driven mesh generator guided by a neural network”. In: *IEEE transactions on magnetics* 29.2, pp. 1927–1930.

Manevitz, Larry, Bitar, Akram, and Givoli, Dan (2005). “Neural network time series forecasting of finite-element mesh adaptation”. In: *Neurocomputing* 63, pp. 447–463.

Manevitz, Larry, Yousef, Malik, and Givoli, Dan (1997). “Finite–element mesh generation using self–organizing neural networks”. In: *Computer-Aided Civil and Infrastructure Engineering* 12.4, pp. 233–250.

Martın, Abadi and al., et (2015). *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. URL: `http://tensorflow.org/`.

Mehendale, Ninad and Soman, Sumedh (2021). “Finite Element (FE) Mesh Generation for 2D Shapes using Multiple Long Short-Term Memory Networks”. In: *SSRN 3968782 (2021)*.

Naumov, Maxim, Mudigere, Dheevatsa, Shi, Hao-Jun Michael, Huang, Jianyu, Sundaraman, Narayanan, Park, Jongsoo, Wang, Xiaodong, Gupta, Udit, Wu, Carole-Jean, Azzolini, Alisson G, et al. (2019). “Deep learning recommendation model for personalization and recommendation systems”. In: *arXiv preprint arXiv:1906.00091*.

Nikishkov, GP (2004). “Introduction to the finite element method”. In: *2009 Lecture Notes, University of Aizu*, pp. 1–70.

Notay, Yvan (2010). “An aggregation-based algebraic multigrid method”. In: *Electronic transactions on numerical analysis* 37.6, pp. 123–146.

Oden and Prudhomme, Serge (2001). “Goal-oriented error estimation and adaptivity for the finite element method”. In: *Computers & mathematics with applications* 41.5-6, pp. 735–756.

Ojha, Vivek (2022). “Goal-Oriented Mesh Adaptation for High-Fidelity Aeroelastic Simulations”. PhD thesis. University of Michigan.

Otter, Daniel W, Medina, Julian R, and Kalita, Jugal K (2020). “A survey of the usages of deep learning for natural language processing”. In: *IEEE transactions on neural networks and learning systems* 32.2, pp. 604–624.

Owen, Steven J (1998). “A survey of unstructured mesh generation technology.” In: *IMR* 239.267, p. 15.

Pan, Jie, Huang, Jingwei, Cheng, Gengdong, and Zeng, Yong (2023). “Reinforcement learning for automatic quadrilateral mesh generation: A soft actor–critic approach”. In: *Neural Networks* 157, pp. 288–304.

Papagiannopoulos, Alexis, Clausen, Pascal, and Avellan, Francois (2021). “How to teach neural networks to mesh: Application on 2-D simplicial contours”. In: *Neural Networks* 136, pp. 152–179.

Parthasarathy, VN, Graichen, CM, and Hathaway, AF (1994). “A comparison of tetrahedron quality measures”. In: *Finite Elements in Analysis and Design* 15.3, pp. 255–261.

Prudhomme, Serge, Nobile, Fabio, Chamoin, Ludovic, and Oden, J Tinsley (2004). “Analysis of a subdomain-based error estimator for finite element approximations of elliptic problems”. In: *Numerical Methods for Partial Differential Equations: An International Journal* 20.2, pp. 165–192.

Raissi, Maziar, Perdikaris, Paris, and Karniadakis, George E (2019). “Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations”. In: *Journal of Computational physics* 378, pp. 686–707.

Ramaswamy, S (1980). “The Lax-Milgram theorem for Banach spaces, I”. In.

Rao, Qing and Frtunikj, Jelena (2018). “Deep learning for self-driving cars: Chances and challenges”. In: *Proceedings of the 1st international workshop on software engineering for AI in autonomous systems*, pp. 35–38.

Rish, Irina et al. (2001). “An empirical study of the naive Bayes classifier”. In: *IJCAI 2001 workshop on empirical methods in artificial intelligence*. Vol. 3. 22, pp. 41–46.

Ruder, Sebastian (2016). “An overview of gradient descent optimization algorithms”. In: *arXiv preprint arXiv:1609.04747*.

Saad, Youcef and Schultz, Martin H (1986). “GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems”. In: *SIAM Journal on scientific and statistical computing* 7.3, pp. 856–869.

Scarselli, Franco, Gori, Marco, Tsoi, Ah Chung, Hagenbuchner, Markus, and Monfardini, Gabriele (2008). “The graph neural network model”. In: *IEEE transactions on neural networks* 20.1, pp. 61–80.

Shen, Dinggang, Wu, Guorong, and Suk, Heung-Il (2017). “Deep learning in medical image analysis”. In: *Annual review of biomedical engineering* 19, pp. 221–248.

Shewchuk, Jonathan Richard (2002). “Delaunay refinement algorithms for triangular mesh generation”. In: *Computational geometry* 22.1-3, pp. 21–74.

Si, Hang (2015). “TetGen, a Delaunay-based quality tetrahedral mesh generator”. In: *ACM Transactions on Mathematical Software (TOMS)* 41.2, p. 11.

Sirignano, Justin and Spiliopoulos, Konstantinos (2018). “DGM: A deep learning algorithm for solving partial differential equations”. In: *Journal of computational physics* 375, pp. 1339–1364.

Sleijpen, Gerard LG and Fokkema, Diederik R (1993). “BiCGstab (ell) for linear equations involving unsymmetric matrices with complex spectrum”. In: *Electronic Transactions on Numerical Analysis.* 1, pp. 11–32.

Song, Chaoyue, Wei, Jiacheng, Li, Ruibo, Liu, Fayao, and Lin, Guosheng (2021). “3D pose transfer with correspondence learning and mesh refinement”. In: *Advances in Neural Information Processing Systems* 34, pp. 3108–3120.

Stevenson, Rob (2007). “Optimality of a standard adaptive finite element method”. In: *Foundations of Computational Mathematics* 7.2, pp. 245–269.

Strang, Gilbert and Fix, George J (1973). “An analysis of the finite element method(Book- An analysis of the finite element method.)” In: *Englewood Cliffs, N. J., Prentice-Hall, Inc., 1973. 318 p.*

Tingfan, WU, Xuejun, LIU, Wei, AN, Huang, Zenghui, and Hongqiang, LYU (2022). “A mesh optimization method using machine learning technique and variational mesh adaptation”. In: *Chinese Journal of Aeronautics* 35.3, pp. 27–41.

Veit, Andreas, Wilber, Michael J, and Belongie, Serge (2016). “Residual networks behave like ensembles of relatively shallow networks”. In: *Advances in neural information processing systems*, pp. 550–558.

Velickovic, Petar, Cucurull, Guillem, Casanova, Arantxa, Romero, Adriana, Lio, Pietro, Bengio, Yoshua, et al. (2017). “Graph attention networks”. In: *stat* 1050.20, pp. 10–48550.

Wang, Peng, Xingjie, Chen, Zhang, Zhou, Bingyan, Zhou, Yajin, and Zhou, Nan (2022). “Surrogate modeling for neutron diffusion problems based on conservative physics-informed neural networks with boundary conditions enforcement”. In: *Annals of Nuclear Energy* 176, p. 109234.

Wang and Zhang (2022). “A Survey of Deep Learning-Based Mesh Processing”. In: *Communications in Mathematics and Statistics* 10.1, pp. 163–194.

Werbos, Paul J (1990). “Backpropagation through time: what it does and how to do it”. In: *Proceedings of the IEEE* 78.10, pp. 1550–1560.

Wessner, Wilfried, Ceric, Hajdin, Heitzinger, Clemens, Hössinger, Andreas, and Selberherr, Siegfried (2003). *Anisotropic mesh adaption governed by a Hessian matrix metric.* na.

Yang, Jiachen, Dzanic, Tarik, Petersen, Brenden, Kudo, Jun, Mittal, Ketan, Tomov, Vladimir, Camier, Jean-Sylvain, Zhao, Tuo, Zha, Hongyuan, Kolev, Tzanio, et al. (2023). “Reinforcement learning for adaptive mesh refinement”. In: *International Conference on Artificial Intelligence and Statistics.* PMLR, pp. 5997–6014.

Yerry, Mark A and Shephard, Mark S (1984). "Automatic three-dimensional mesh generation by the modified-octree technique". In: *International journal for numerical methods in engineering* 20.11, pp. 1965–1990.

Young, Tom, Hazarika, Devamanyu, Poria, Soujanya, and Cambria, Erik (2018). "Recent trends in deep learning based natural language processing". In: *ieee Computational intelligenCe magazine* 13.3, pp. 55–75.

Yu, Bing et al. (2018). "The deep Ritz method: a deep learning-based numerical algorithm for solving variational problems". In: *Communications in Mathematics and Statistics* 6.1, pp. 1–12.

Zhang and Naga, Ahmed (2005). "A new finite element gradient recovery method: superconvergence property". In: *SIAM Journal on Scientific Computing* 26.4, pp. 1192–1213.

Zhang, Si, Tong, Hanghang, Xu, Jiejun, and Maciejewski, Ross (2019). "Graph convolutional networks: a comprehensive review". In: *Computational Social Networks* 6.1, pp. 1–23.

Zhang, Zheyan (2023). *meshingnet*. URL: `https://gitlab.com/meshingnet`.

Zhang, Zheyan, Jimack, Peter K, and Wang, He (2021). "MeshingNet3D: Efficient generation of adapted tetrahedral meshes for computational mechanics". In: *Advances in Engineering Software* 157, p. 103021.

Zhang, Zheyan, Wang, Yongxing, Jimack, Peter K, and Wang, He (2020). "MeshingNet: A new mesh generation method based on deep learning". In: *Computational Science–ICCS 2020: 20th International Conference, Amsterdam, The Netherlands, June 3–5, 2020, Proceedings, Part III 20*. Springer, pp. 186–198.

Zhao, Ding, Xueying, and Prakash, B Aditya (2023). "PINNsFormer: A Transformer-Based Framework For Physics-Informed Neural Networks". In: *arXiv preprint arXiv:2307.11833*.

Zhou, Jie, Cui, Ganqu, Hu, Shengding, Zhang, Zhengyan, Yang, Cheng, Liu, Zhiyuan, Wang, Lifeng, Li, Changcheng, and Sun, Maosong (2020). "Graph neural networks: A review of methods and applications". In: *AI open* 1, pp. 57–81.

Zienkiewicz, Taylor, Robert Leroy, and Zhu, Jian Z (2005). *The finite element method: its basis and fundamentals*. Elsevier.

Zienkiewicz and Zhu (1987). “A simple error estimator and adaptive procedure for practical engineerng analysis”. In: *International journal for numerical methods in engineering* 24.2, pp. 337–357.

– (1992). “The superconvergent patch recovery and a posteriori error estimates. Part 1: The recovery technique”. In: *International Journal for Numerical Methods in Engineering* 33.7, pp. 1331–1364.

Zienkiewicz, Zhu, JZ, and Gong, NG (1989). “Effective and practical h–p-version adaptive analysis procedures for the finite element method”. In: *International Journal for Numerical Methods in Engineering* 28.4, pp. 879–891.