A Graph Neural Network for Guiding Adaptive Finite Element Mesh Refinement in Transient Flow Simulation

Z Zhang[1,2], P K Jimack*[1], H Wang[3]

[1] School of Computer Science, University of Leeds, Leeds, UK, LS2 9JT
[2] Luoyang Ship Material Research Institute, Luoyang 471023, China
[3] AI Centre, Department of Computer Science, University College London, London, UK, WC1E 6BT

Keywords: Transient flow simulation, Mesh adaptivity, Graph neural networks, Finite element method

Abstract: This paper proposes the use of a graph convolutional network (GCN) to accelerate the mesh adaptivity steps when undertaking finite element (FE) simulations of two-dimensional transient flow problems. A traditional work flow uses an FE spatial discretization combined with a suitable, stable, time-stepping scheme. At the end of each time step an error estimator is used to assess the current solution quality and guide adaptation of the FE mesh for the next time step. Once the mesh is adapted the current solution is interpolated onto it, and the next time step is taken. In this paper we propose replacing the error estimation calculation, based upon the solution at the end of the time step, with a machine learning component that predicts the error distribution at the end of the step prior to the completion of the step. The motivation is to allow the mesh for the next time step to be generated in parallel with computing the FE solution at the current time step, in order to accelerate the overall execution. Results are presented that evaluate the performance of the GCN-based adaptive refinement against mesh refinement using the true error estimator after each time step (the ground truth), and against mesh refinement using the true error estimator applied to the solution at the end of the previous time step (a baseline for assessing the added value of the GCN).

1 Introduction

Throughout this paper we consider the solution of time-dependent flow problems using a finite element (FE) spatial discretization combined with a stable, one-step, time stepping scheme. All of our implementations use the open source software FreeFEM++ ([1]), which provides full support for adaptive mesh refinement within the solution process. Specifically, this includes mesh generation, implicit time-stepping, finite element solution, error estimation, mesh adaptivity and solution interpolation.

In the case of incompressible Newtonian flows, the governing partial differential equations that we consider are the two-dimensional Navier-Stokes equations in Cartesian coordinates:

$$\begin{aligned} \rho\left(\frac{\partial u}{\partial t}+u\frac{\partial u}{\partial x}+v\frac{\partial u}{\partial y}\right) &= \rho f_x-\frac{\partial p}{\partial x}+\mu\left(\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 u}{\partial y^2}\right) \\ \rho\left(\frac{\partial v}{\partial t}+u\frac{\partial v}{\partial x}+v\frac{\partial v}{\partial y}\right) &= \rho f_y-\frac{\partial p}{\partial y}+\mu\left(\frac{\partial^2 v}{\partial x^2}+\frac{\partial^2 v}{\partial y^2}\right) \\ \frac{\partial u}{\partial x}+\frac{\partial v}{\partial y} &= 0 \end{aligned} \tag{1}$$

where u and v are the x and y velocity components, p is the pressure, ρ is the density, μ is the dynamic viscosity, and f_x and f_y are external force components. The FE spaces that are considered in this work use piecewise linear pressures and piecewise linear plus quadratic bubbles for velocities (see [1]), based upon triangulations of the spatial domain: with the size and orientation of the triangular elements chosen to provide a good approximation to the current solution. As the solution evolves in time, it is therefore necessary to update the triangulation in an adaptive manner. For a given mesh of triangles, T_h^t at time t, we denote the corresponding finite element space for representing (u, v, p) as V_h^t

1.1 Adaptive mesh refinement

A typical approach to coupling mesh adaptivity with the FE solution of a transient problem consists of four high level steps that are undertaken in sequence at each time step:

1. Computation to obtain the solution $(u, v, p)^t$ on FE space V_h^t

*Corresponding Author, P K Jimack P.K.Jimack@leeds.ac.uk

2. Error estimation, $(u, v, p)^t \rightarrow E^t$
3. Mesh generation, $E^t \rightarrow T_h^{t+1}$
4. Read mesh and interpolate $(u, v, p)^t$ 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, $(u, v, p)^{t-1}$ is known and the velocity and pressure are updated to $(u, v, p)^t$ by solving the Galerkin approximation to (1) following temporal discretization.

For the second step, there are multiple options for undertaking the error estimation calculation. The most reliable approach, so-called *a posteriori* error estimation, involves completing additional computations using a richer finite element space in order to directly approximate the error in each region of the spatial domain. In this work however we make use of a less rigorous approach based upon computation of the Hessian of the current velocity components at each point in the spatial domain. This is an option available in FreeFem++ ([1]), designed specifically for generating anisotropic meshes, where the triangles can be far from equilateral but must align with the local anisotropy in the solution field ([2], [3]). To illustrate this, consider a single solution component $\eta(x)$. We assume that this has been computed on the current mesh and is a piecewise continuous quadratic function on this mesh. Then the so-called *interpolation error* depends on the Hessian matrix of η ([4]):

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

The second equality above is based upon the eigen-decomposition of the real symmetric matrix $H(x)$, so λ_1 and λ_2 are the local eigenvalues of the Hessian. We may then define a metric tensor which will be used in the following step to generate a new anisotropic mesh:

$$M(x) = R(x) \begin{matrix} |\lambda_1| & 0 \\ 0 & |\lambda_2| \end{matrix} R(x)^{-1} \tag{3}$$

For elements of order p the mesh generation (see next step) aims to produce a non-uniform and anisotropic triangulation such that

$$\frac{h_i}{h_j} = (|\lambda_j|/|\lambda_i|)^{1/(p+1)} \tag{4}$$

where h_i and h_j are the local mesh sizes in two perpendicular directions.

In the third step, the mesh is globally regenerated. The anisotropic mesh size control is based on an error distribution $E^t(x)$ which, in this paper, is the metric $M(x)$ given by (3). In many 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. Such a background mesh is used in this work, where it must be sufficiently fine that it can represent the solution and mesh size metric accurately throughout the whole spatial domain at all times in the evolution of the solution. The FreeFem++ embedded mesh generator, BAMG ([5]), uses this $M(x)$ to provide the next anisotropic mesh (with some smoothing parameters to ensure gradual changes in the mesh size and orientation across the domain).

In the fourth step, the nonuniform mesh T_h^{t+1} (generated by BAMG) is utilized and a new FE space V_h^{t+1} is built upon this mesh. The solution $(u, v, p)^t$ is then interpolated from FE space V_h^t to V_h^{t+1} using a barycentric solution interpolation provided by FreeFem++. We note that this interpolation algorithm, whilst very efficient, is not guaranteed to be mass conserving so could be switched out for a mass-conserving alternative (e.g. [6]) – however this is not the main focus of the presented research so we do not consider this any further here.

1.2 Contributions of this work

The objective of this research is to replace step 2 in the above workflow with a machine learning model to approximate the error distribution $E^t(x)$ (given by $M(x)$ in equation (3)). This model will *not* require the latest computed solution, $(u, v, p)^t$, as its input – instead its prediction will be based upon the solution at the two previous times steps. The benefit of being able to predict the error metric in this way is that step 3 of the workflow can be computed in parallel with step 1, ensuring that the new mesh is ready

*Corresponding Author, P K Jimack P.K.Jimack@leeds.ac.uk

for interpolation to take place as soon as step 1 has completed. This therefore allows the possibility to significantly accelerate the adaptive solver by hiding the costs of steps 2 and 3 at each time step (by executing them in parallel with step 1).

In the remainder of this paper we introduce our proposed machine learning model, based upon the application of a graph convolutional network (GCN) and then we evaluate its performance in comparison to the conventional error estimation algorithm that is used to train it (the "ground truth"). We further observe that it is also possible to complete step 3 (mesh generation) in parallel with step 1 (solving for the current time step) if we replace step 2 with an error estimate that is based upon an *old* solution value (at time level $t-1$ for example). In order to comprehensively evaluate the benefit of our GCN we also compare it against results obtained using the old solution to guide the mesh generation (the "baseline").

2 Prediction of Error Estimates using a Graph Neural Network

Our goal is to estimate the solution metric $M(x)$, which has three components due to the symmetry of the Hessian, at each node x in the background grid T_b. This provides input to the mesh generator, allowing it to create an appropriate non-uniform, anisotropic, mesh.

2.1 Neural network architecture

We use a graph neural network made up of n convolutional layers (typically $n = 5$), where each layer is based upon the graph of T_b (whose adjacency matrix we denote as $A \in R^{N \times N}$). The input to the network is the error metrics at each node from the previous two time steps (M^{t-2} and M^{t-1}) and the output is the predicted error metric at each node of T_b at the end of the next time step. Following [7], each convolutional layer is represented by a truncated expansion of Chebyshev polynomials,

$$\sum_{k=0}^{K} \theta_k T_k(\tilde{L})(M^{t-2},\ M^{t-1}) \tag{5}$$

where the θ's are trainable weights and $\tilde{L} = \frac{2}{\lambda_{max}} L - I_N$. Here L is the normalized Laplacian (i.e. $L = I_N - D^{-1/2} A D^{-1/2}$ with $D_{ii} = \sum_{j=1}^{N} A_{ij}$ and I_N the $N \times N$ identity) and λ_{max} is its largest eigenvalue. Further details may be found in [8].

2.2 Data preparation and training

The first step in data preparation is to interpolate the latest solution onto the background mesh. The error metric is then constructed from the Hessian of the solution on this background mesh. Note that our framework is agnostic to the specific choice of error estimator, however we only consider the Hessian in this paper. Once constructed, the error metric is then normalised in order to facilitate training. We use *Keras* based on *TensorFlow* as the deep learning platform. Our convolution layers are typically constructed with first-order Chebyshev polynomials; however, the use of multiple layers enables information to be passed across multiple graph edges. We set 8-16-32-16-8 channels in the hidden layers, since it is advantageous to first increase then decrease the number of channels. The activation functions selected in the model are rectified linear functions. The training uses mean squared error loss and the stochastic gradient descent optimizer, Adam ([9]), with batch sizes of 8-32 for different problems.

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. When we prescribe the target number of elements for the non-uniform mesh, we adjust the element size proportionally to reach this target number.

3 Computational Results

We assess the performance of the proposed GCN through its application to two test problems. In each case the training data is generated using FreeFEM++, following the adaptive approach outlined in subsection 1.1. Once trained, the GCN is used to evaluate the error metric $M(x)$ at each point on a uniform background grid, which then guides the embedded mesh generator.

*Corresponding Author, P K Jimack P.K.Jimack@leeds.ac.uk

3.1 Flow with two immiscible fluids

Consider two immiscible fluids k = 1,2 with respective density and viscosity ρ_k and ν_k ($\rho_1 = 1$, $\rho_2 = 8$, $\nu_1 = 0.05$ and $\nu_2 = 0.01$). We neglect surface tension forces at the interface and use a level set function for front tracking. The spatial domain is a 2×1 rectangle and the initial interface location is problem dependent. We constructed five training problems with different initial conditions and 100 steps each. Training used a batch size of 32 and 2000 epochs.

Two tests are reported here, with unseen initial conditions: $y = 0.25+0.3x$ and $y = 0.1+0.375x$. Figure 1 illustrates snapshots of the interface in the second case (with the denser fluid below), whilst Figure 2 compares the GCN and ground truth meshes at step 24. In order to asses the performance of the GCN approach we compare the solutions obtained using the ground truth error estimator, the GCN estimator and a baseline approach which computes the Hessian error indicator using the solution at the previous time step. Figure 3 shows the corresponding error, based upon use of an ultra-fine mesh to compute a reference solution. It is apparent that, in both tests, our neural network approach closely follows the ground truth solution for a significant number of time steps (in one case for the whole simulation and in the other for the first 25 steps at least). Furthermore, the GCN outperforms the baseline, in which the error estimate is based upon an old solution.

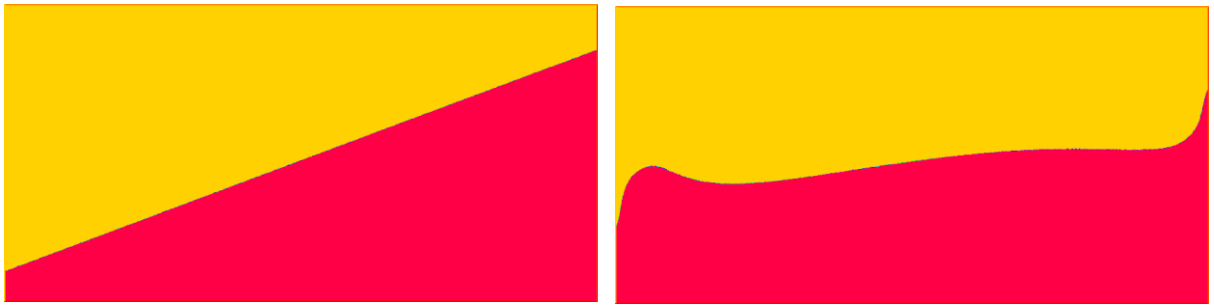

Figure 1: Snapshots of the two phase fluid interface at time steps 2 and 30, with initial interface at $y = 0.1 + 0.375x$.

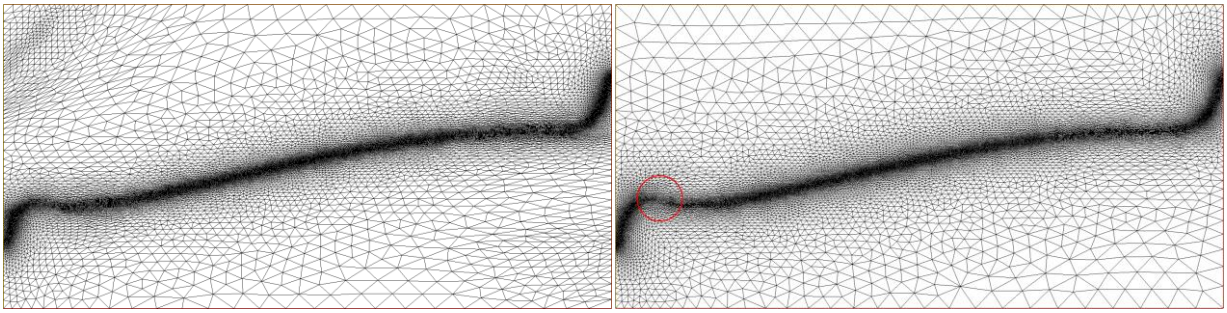

Figure 2: Snapshots of the GCN mesh (left) and the ground truth mesh (right) at time step 24, with initial interface at $y = 0.1 + 0.375x$.

3.2 Vortex shedding in flow past obstacles

For this example we consider flows in a confined channel with either a single or a pair of cylinders as obstacles. Figure 4 illustrates one such example, with a parabolic inflow, such that the parameters in (1) correspond to a Reynolds number (Re) of 200. In order to assess the generalizability of the GCN six different networks were trained, each with the same architecture but with different training sequences (corresponding to two geometries and three Reynolds numbers). We then assessed the performance of each GCN on different test cases to those used to train it.

Figure 5 illustrates one example, where the GCN was trained on a problem with a single obstacle and a Re=400 but is tested on a problem with a double obstacle and Re=200. As with the two-phase flow example, it is clear that the resulting mesh is very similar to the corresponding ground truth mesh. It

*Corresponding Author, P K Jimack P.K.Jimack@leeds.ac.uk

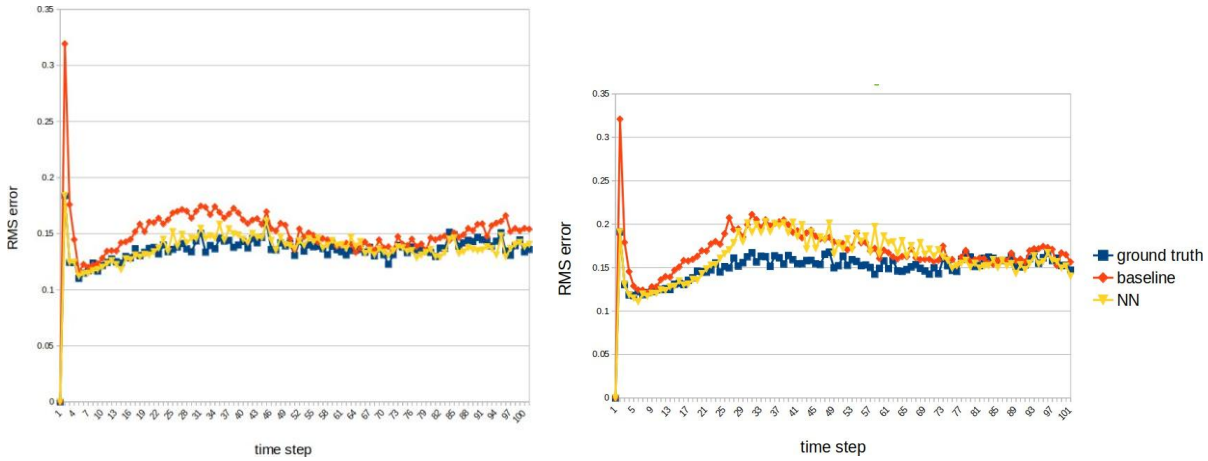

Figure 3: Evolution of the solution error (compared to ultra-fine mesh solutions) for the three different refinement strategies on the two test cases: with initial interfaces at $y = 0.25 + 0.3x$ (left) and $y = 0.1 + 0.375x$ (right).

is possible to quantify this similarity by plotting the difference between the solution computed on the GCN meshes to the solution computed on the ground truth meshes as the flow evolves in time. This is shown in Figure 6, which also shows the corresponding difference between the ground truth solution and that computed using mesh adaptivity based upon the solution at previous time steps (the baseline). It is apparent that the GCN approach is superior over at least the first 100 time steps.

4 Conclusions

In this paper we have used a GCN to predict an error metric that drives the anisotropic generation of triangular finite element meshes. This is designed to accelerate the conventional workflow for adaptive time-dependent flow simulations by allowing the mesh for the next time step to be generated concurrently with computing the FE solution at the current time step. Our initial results confirm that this is a reliable approach, at least up to roll-outs of many tens of steps, and often significantly more. Furthermore, in the cases presented in this paper the GCN approach is shown to have improvements relative to a simple baseline in which an old solution (from the previous time step) is used to guide the mesh refinement. Although not show here, there are some examples where this baseline is found to perform almost identically to the ground truth. In such cases, there is of course no advantage to using the GCN approach unless an improved ground truth error estimator can be defined: see [8] for further details.

References

[1] F. Hecht, “New development in freefem++,” *Journal of numerical mathematics*, vol. 20, no. 3-4, pp. 251–266, 2012.

[2] W. Huang, “Metric tensors for anisotropic mesh generation,” *Journal of computational physics*, vol. 204, no. 2, pp. 633–665, 2005.

[3] W. Wessner, H. Ceric, C. Heitzinger, A. Ho¨ssinger, and S. Selberherr, *Anisotropic mesh adaption governed by a Hessian matrix metric*. na, 2003.

[4] M. Castro-D´ıaz, F. Hecht, B. Mohammadi, and O. Pironneau, “Anisotropic unstructured mesh adaption for flow simulations,” *International Journal for Numerical Methods in Fluids*, vol. 25, no. 4, pp. 475–491, 1997.

[5] F. Hecht, “Bamg: bidimensional anisotropic mesh generator,” *User Guide. INRIA, Rocquencourt*, vol. 17, 1998.

[6] P. E. Farrell and J. R. Madison, “Conservative interpolation between volume meshes by local galerkin projection,” *Computer Methods in Applied Mechanics and Engineering*, vol. 200, pp. 89–100, 2011.

[7] D. K. Hammond, P. Vandergheynst, and R. Gribonval, “Wavelets on graphs via spectral graph theory,” *Applied and Computational Harmonic Analysis*, vol. 30, no. 2, pp. 129–150, 2011.

[8] Z. Zhang, *Machine Learning Aided Finite Element Non-uniform Mesh Generation*. PhD thesis, University of Leeds, 2024.

[9] D. P. Kingma and J. Ba, “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*, 2015.

*Corresponding Author, P K Jimack P.K.Jimack@leeds.ac.uk

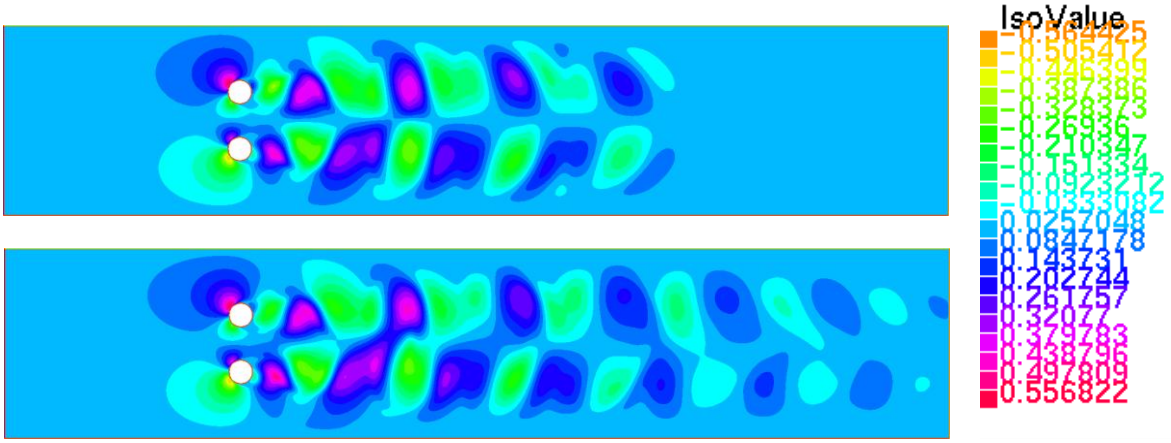

Figure 4: Evolution of the vertical velocity component in the flow past two cylinders at Re=200: time steps 150 and 300.

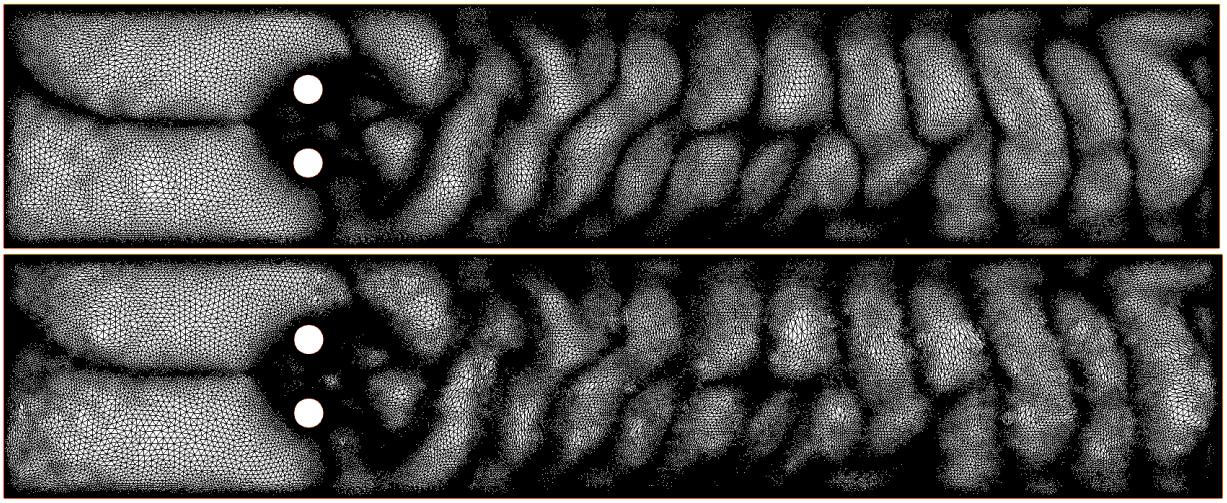

Figure 5: Snapshot meshes at the same point in time: ground truth mesh (top) and GCN mesh (bottom).

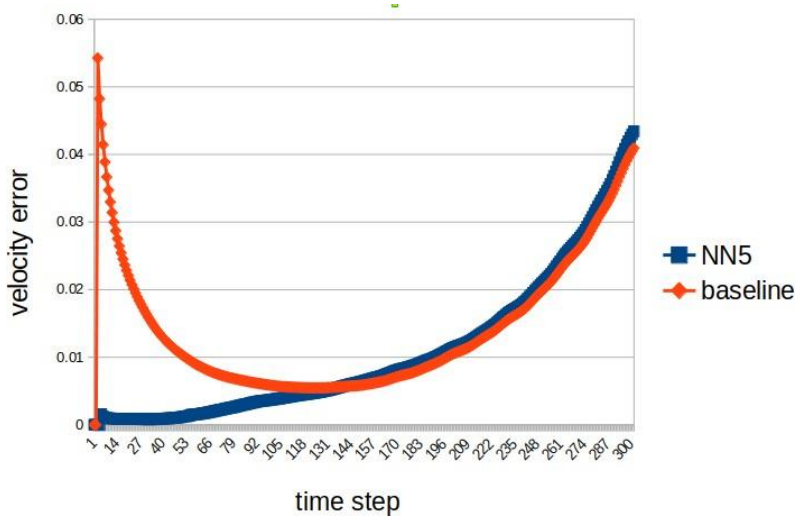

Figure 6: The L1 velocity error (computed as the difference with the ground simulation) across 300 time steps of simulation: the GCN model was trained on a different geometry and a different Reynolds number; the baseline model uses the solution at the previous time step to adapt the mesh.

*Corresponding Author, P K Jimack P.K.Jimack@leeds.ac.uk