Moving mesh methods in Fluidity and Firedrake

T. M. McManus[1,2], J. R. Percival[1], B. A. Yeager[1], N. Barral[1], G. J. Gorman[1], and M. D. Piggott[*1]

[1] *Department of Earth Science and Engineering, Imperial College London, SW7 2AZ, UK*
[2] *Current address: Lawrence Livermore National Laboratory, 7000 East Ave., Livermore, CA 94550, USA*
(This work was performed while TMM was under the auspices of Imperial College London)

April 3, 2017

Abstract

This report summarises the work conducted under the eCSE (embedded Computational Science & Engineering) project 'Integrating mesh movement (r-adaptive) technology within Fluidity and the PRAgMaTIc parallel anisotropic (h) adaptive mesh toolkit'. The overall objective was to integrate new mesh movement (or r-adaptive) methods within two existing PDE solver frameworks (Fluidity & Firedrake) which, to differing degrees, already possess h-adaptive capabilities.

Contents

[*]Corresponding author: m.d.piggott@imperial.ac.uk

1 Introduction

The numerical solution of large-scale, time-dependent, three-dimensional partial differential equations (PDEs) is inherently computationally expensive. Even with access to large high-performance computing (HPC) systems, efforts to improve computational efficiency are still required in order to solve ever and ever larger problems, and/or to provide faster run-times for use within iterative design [12] or uncertainty quantification studies. Adaptive mesh methods represent one category of approaches to achieve this, and indeed have been used extensively within the open source Fluidity code for the study of industrial and geophysical flow problems, e.g. [18, 20, 22, 19, 24, 21, 1].

Broadly speaking, adaptive methods update the computational mesh in some manner in response to error indicators/measures which are based upon known or computed solution characteristics. For the purposes of this work we characterise these methods in one of two ways:

1. h-adaptive methods which locally alter the mesh spacing via structural changes to the mesh, e.g. dividing/collapsing edges/elements or completely regenerating the mesh, and consequently change the degree of freedom count for the problem;

2. r-adaptive (as in relocation/redistribution), or *mesh movement*, methods which maintain the mesh and its structure (i.e. its connectivity and degree of freedom count) but redistribute the vertices of the mesh in order to increase or decrease mesh resolution locally.

Note also that these techniques can be used in combination, such as in hr-adaptive methods which both move the mesh as well as have the option for making structural changes to it where and when necessary, e.g. to avoid mesh tangling, which imparts significantly more power, flexibility and robustness to the overall adaptive mesh functionality.

Note that the terms adaptive mesh refinement (AMR), adaptive remeshing, mesh optimisation, etc are used by some as synonyms for h-adaptivity, while for others these each have clear distinctions. Here we take the latter view, making use of *mesh optimisation* methods as a class of h-adaptivity.

Finally note that we are excluding consideration of p-adaptivity based methods which do not alter the underlying mesh, but vary the degree of approximation over the cells making up the mesh.

This project started from a strong base of h-adaptive (mesh optimisation) functionality, with the overall objective of the work being to increase our ability to make use of r-adaptive methods in challenging real-world problems, and hence also deliver combined hr-adaptive capabilities.

1.1 Background

This work built upon a long track record in the development and application of mesh adaptivity technology at Imperial College London [23, 26, 27, 25, 11, 13, 4]. To date this has revolved almost entirely around so-called mesh optimisation, an example of an h-adaptive method, where the shape and size of the elements making up an unstructured mesh are periodically updated in response to an evolving numerical solution. The aim being to only use high mesh resolution where and when it is needed to optimise the accuracy and efficiency of a calculation. Significant effort has gone into associated parallel performance, dynamic load balancing [14] and accurate mesh to mesh interpolation [11], including a previous dCSE project which integrated the Zoltan data management library with our existing mesh adaptivity library [30].

Mesh movement, or r-adaptive, methods take a different approach where the number of elements in a mesh, and their connectivity, remains constant while the locations of the nodes of the mesh evolve [6]. Through this evolution mesh resolution can again simply be focused on particular spatial locations. However, major benefit result for problems with evolving solution features (such as shocks/fronts/interfaces or boundaries) which can be continuously tracked by higher resolution components of the mesh. If these evolving features are particularly important to a problem, or hard to solve, then mesh movement methods can provide a very powerful addition to an underlying PDE solver.

However, since the total degree of freedom count remains fixed, and there is the potential (and for some methods/applications an inevitability) for elements to become badly skewed or even inverted, mesh movement methods benefit greatly from the additional robustness afforded by combining with

some form of h-adaptivity (leading to a so-called hr-adaptive method). In particular, the ability of mesh optimisation techniques to only make local updates to the mesh, including simply 'flipping' edges and faces, makes this a particularly attractive approach for ensuring mesh quality and robustness. We note that r-adaptive methods can be made to be robust and avoid tangling when adapting to solution features within the domain. However, a major target application for this work include problems where internal or external boundaries move, pinch or rotate, and here it is inevitable even with the most sophisticated r-adaptive approach that problems with mesh quality will result.

1.2 Aims

The overall aim of this work was therefore to integrate new mesh movement (r-adaptive) functionality with our existing Fluidity model and mesh optimisation (h-adaptive) capabilities/libraries. The latter also includes the PRAgMaTIc parallel anisotropic adaptive mesh toolkit which has recently been integrated with the Firedrake PDE solver framework [4].

The principle objectives of this project were therefore to:

1. *Parallelise a pre-existing prototype mesh movement algorithm* – this took the form of an implementation of one approach to mesh movement, namely the solution of a PDE describing a mapping from a fixed, uniform computational domain/mesh into an adapted physical domain/mesh, e.g. as has emanated from so-called elliptic or Poisson mesh generation in the 1960s/70/80s. Our existing approach solved these PDEs in serial using FEniCS/Dolfin infrastructure.

2. *Expand available mesh movement algorithms* – our existing algorithms included simple Lagrangian-based mesh movement and approaches based on the solution of an elliptic PDE (namely Laplacian smoothing and the so-called Winslow method). We planned to use the flexibility afforded by code generation approaches (Dolfin/Firedrake) to include more sophisticated *"PDE-based"* methods. In addition, we planned to implement some examples of an alternative approach to mesh movement, namely so-called *"spring-based analogy"* iterative methods which may well be appropriate for certain applications.

3. *Integrate mesh movement with the Fluidity CFD & marine modelling code* – Fluidity is used across a very wide range of application areas, many of which currently make successful use of mesh optimisation, but which would benefit significantly from mesh movement. Here we planned to implement the ability to invoke mesh movement from within Fluidity and explore its value across several application areas. We also planned to consider the combined use of mesh optimisation with mesh movement (i.e. hr-adaptivity), largely to mitigate issues with mesh tangling as would be expected with sustained use of mesh movement alone.

4. *Integrate mesh movement capabilities with Firedrake and its initial coupling to PRAgMaTIc* – To allow for the further uptake of the developed r-based methods by wider applications codes we will integrate them with the PRAgMaTIc mesh optimisation library through the utilisation of PETSc's DMPlex unstructured mesh management library, with a demonstration using a Firedrake based PDE solver.

2 Underlying methods and libraries

2.1 Fluidity

Fluidity is a general purpose finite element problem solving environment, with the majority of its applications in industrial and geophysical fluid dynamics. It is being used across a wide range of funded projects, primarily in the areas of energy generation, pollution dispersal, and Earth system science. Performance analysis of Fluidity has been addressed in previous dCSE and eCSE projects[1][2]. More recent performance results are described in [16, 17]. In particular, a previous project demonstrated

[1] http://www.hector.ac.uk/cse/distributedcse/reports/fluidity-zoltan/fluidity-zoltan.pdf
[2] http://www.hector.ac.uk/cse/distributedcse/reports/fluidity-icom01/fluidity-icom01.pdf

scalability on up to 16,384 cores using hybrid MPI/OpenMP parallelism[3]. Scaling on ARCHER in the context of parallel I/O and the integration of PETSc's DMPLex mesh management library has been addressed as part of eCSE project 'Scalable and interoperable I/O for Fluidity' (eCSE01-009)[4], and also in the context of Firedrake[5].

2.2 Firedrake

While Fluidity is a very flexible and robust finite element based PDE solver framework which has been developed and re-coded in modern Fortran over the past decade it does have limitations. Firedrake on the other hand takes a far more modern approach to the finite element solution of PDEs where the weak form of the discretised PDE is defined in a high-level domain-specific language, and automatic code generation techniques are used to generate low level code which is robust and optimised for a target platform. Additional benefits of the code generation approach are that powerful adjoint models are far more easily generated [10], leading to the ability to perform parameter optimisation and sensitivity studies far more readily [12].

In addition to targeting Firedrake as the basis for application codes which will make use of mesh movemement, Firedrake also offers a solution for the implementation of r-adaptivity via the approach that requires the solution of a PDE. These PDEs can be complex, nonlinear and hard to solve, and the ease with which methods for this can be implemented using Firedrake will be exploited in this project.

Firedrake is also actively being run on ARCHER to perform a range of benchmarking problems[6], with preliminary results showing good strong and weak scaling of up to several hundred cores.

2.3 PRAgMaTIc

PRAgMaTIc is a standalone anisotropic mesh optimisation library, improving the mesh adaptation tool from Fluidity with recent software development concepts and advances in meshing. From an original mesh and a prescribed metric field on that mesh (a size map, generated from a solution field), it generates a new unstructured adapted mesh, *i.e.* a mesh whose elements respect the sizes and orientations prescribed by the metric field. If the metric field is chosen wisely, the underlying equations are resolved with a greater accuracy on the resulting adapted mesh

PRAgMaTIc handles meshes with complex geometries in both 2D and 3D, and is able to generate meshes with very high aspect ratio (>10,000) elements, thus taking advantage of the anisotropy of the solutions to reduce the number of degrees of freedom.

As opposed to the algorithm within Fluidity, PRAgMaTIc was devised from the beginning as a parallel library, and combines shared- and distributed-memory parallelism. The adapted mesh is obtained through a series of advanced local mesh operations: refinement, coarsening and edge/face swapping optimise the resolution and the quality of the mesh, and quality-constrained Laplacian smoothing fine-tunes the mesh quality. Advanced algorithms are used to perform mesh optimisation operations concurrently on a partition, while remeshing the halos coherently without repartitioning. Information on the integration of PRAgMaTIc with PETSc and Firedrake may be found in [4].

3 Existing *hr*-adaptivity framework

The parallel, load balanced, anistropic hr-mesh adaptivity algorithm used in Fluidity is described in [23, 25, 14]. The algorithm uses a series of mesh operations to minimize a mesh quality functional which locally evaluates element size and shape with respect to a metric tensor. At a particular time level, this tensor thus encodes the ideal (anisotropic) size of elements. In 3D a bespoke method implmented for Fluidity is used [23], while in 2D Fluidity iterfaces with the ani2d library[7].

[3] http://www.hector.ac.uk/cse/distributedcse/reports/fluidity-icom02/fluidity-icom02.pdf
[4] http://prism.ac.uk/wp-content/uploads/2014/05/mlange_dmplex-io.pdf
[5] http://www.mcs.anl.gov/petsc/petsc-20/conference/Lange_M.pdf
[6] https://github.com/firedrakeproject/firedrake-bench/wiki/Archer
[7] http://ani2d.sourceforge.net/

Prior to this project the pre-existing framework was already termed as being *hr*, since it did allow the user to specify grid velocities analytically, and hence to solve the physical PDEs in moving reference frames. It also allowed for restricted mesh movement in response to an evolving free surface field deforming the whole solution domain. But it must be made clear that this was by no means a 'fully functioning *hr*-adaptive capability'. We also note that very limited 'mesh movement', via discrete Laplacian smoothing, is often utilised as a step within mesh optimisation to improve overall mesh quality.

3.1 Coupling mesh movement with governing model equations

Fluidity implements a finite element solver for the Navier-Stokes equations, including a linear ALE (arbitrary Lagrangian-Eulerian) framework as described in [8], in which for each timestep the equations take the form

$$\left.\frac{\partial \boldsymbol{u}}{\partial t}\right|_{\boldsymbol{\xi}} + (\boldsymbol{u}-\boldsymbol{v})\cdot\nabla_{\boldsymbol{x}}\boldsymbol{u} = -\nabla_{\boldsymbol{x}} p + \frac{1}{\mathrm{Re}}\nabla^2_{\boldsymbol{x}}\boldsymbol{u}, \qquad \text{(momentum)}$$

$$\nabla_{\boldsymbol{x}}\cdot\boldsymbol{u} = 0. \qquad \text{(continuity)}$$

Here we have chosen to give the incompressible, constant density form, with $\boldsymbol{u}(\boldsymbol{x},t)$ and $p(\boldsymbol{x},t)$ the Eulerian fluid velocity and pressure. The mesh coordinate, $\boldsymbol{\xi}$, is the assumed to satisfy an implicit auxiliary equation,

$$\left.\frac{d\boldsymbol{x}}{dt}\right|_{\boldsymbol{\xi}} = \boldsymbol{v}\left(\boldsymbol{\xi},t\right),$$

in terms of the mapping between the mesh and the physical Eulerian spatial coordinate, $\boldsymbol{x}(\boldsymbol{\xi},t)$. In particular, if we choose to apply the fluid velocity in the flow equation for the mesh coordinate, so that we may identify it with the Lagrangian coordinates for the flow, then the advection term vanishes as expected.

We discretize this equation set using a $\mathrm{P_{1DG}P_2}$ element pair for velocity and pressure, and a $\mathrm{P_1}$ discretization for $\boldsymbol{x}$ and $\boldsymbol{v}$ over a simplicial partition of the domain, Ω. Closure demands that we describe the mesh velocity $\boldsymbol{v}$ in terms of the other quantities,

$$\boldsymbol{v} := \boldsymbol{v}\left(\boldsymbol{\xi}; \boldsymbol{x}\left(t_0\right), \boldsymbol{u}, \Omega\right).$$

Mesh movement algorithms differ in how they perform this closure. The methods implemented as part of this project are summarised in the next section, with mathematical details given for completeness in the appendix.

4 Mesh movement methods overview

The mesh movement algorithms considered in this project can be split into three families:

1. *"Prescribed" mesh velocity methods* – where a subset of the nodes of the mesh move in a prescribed manner defined either by the underlying fluid motion (e.g. as in Lagrangian-like mesh movement methods), or through a pre-defined or diagnostically-computed movement of a structure or boundary within the domain. With such a simple approach, issues can arise rapidly with mesh tangling although this can be addressed through periodic use of mesh optimisation where edge/face swapping can mitigate poor element quality.

2. *"Spring"–based methods* – where a subset of the nodes are computed as per approach 1 above (to track an evolving boundary), and the rest of the mesh is then evolved based on an assumption that element edges are a network of springs. While this can help with the issue of mesh quality, mesh optimisation still needs to be used periodically.

3. *"PDE"–based methods* – here a PDE is solved to yield a mapping from a fixed "computational" domain to an adapted mesh in the "physical" domain upon which the underlying problem of interest is discretised. The nature of the precise "moving mesh PDE" solved imparts different

mesh quality properties on the resulting mesh. While poor quality meshes can thus be avoided, the approach still generally needs to be combined with mesh optimisation, e.g. to allow for the number of degrees of freedom in the discretisation to change – e.g. to grow as a dynamic problem spins-up.

The current state-of-the-art in combined mesh optimisation & mesh movement can be found in [2] which follows approach 3 (with the PDE motivated by a linear-elasticity analogy, and hence is also related physically to approach 2). The state-of-the-art in PDE-based mesh movement involves the solution of the nonlinear Monge-Ampere equation [5].

We have been tackling the latter PDE-based approach through a novel approach which uses code generation technology (such as is implemented within the Firedrake project) in order to prescribe the PDE (and hence the precise mesh movement method) in a high-level domain-specific language (UFL), and to automate the generation of appropriate and optimal finite element code. This provides a solution to the mesh movement PDE and thus the transformation from computational to physical space yielding the adapted mesh. As the underlying solution evolves, and updated error metrics/indicators are available at every time step, the mesh movement PDE is solved repeatedly providing a new transformation and hence a "moving mesh". The use of a single Firedrake-based approach means that different mesh movement strategies can be used within the same framework, and that parallelisation and code optimisation strategies carry across rather than having to be done on a case by case basis.

5 Examples of target applications

The application areas for which mesh movement technology is required (or highly desirable), in isolation or combined with mesh optimisation, can be split into two categories within computational fluid dynamics (CFD), namely those with:

1. *Evolving internal structures* (with two-way coupling between the structures driving, or being driven by, fluid dynamics), e.g.
   (a) *Turbines* – for the highly accurate simulation of wind/tidal turbines and their turbulent wakes for the purpose of optimising the designs of turbines, and understanding interactions between devices and subsequent impacts on optimal array designs.
   (b) *Centrifugal pumps* – to model pump performance, optimise designs, and further to predict and minimise component wear as per area 1a above.
2. *Evolving external boundaries* (where two-way dynamic coupling between domain geometry and resulting fluid dynamics, which drives boundary evolution, needs to be modelled), e.g.
   (a) *Wear in industrial fluid dynamics* – to track the dynamic evolution of pipe/ pump walls in response to the motion of abrasive slurry in mineral processing.
   (b) *Scour in geophysical fluid dynamics* – as above but in response to sediment transport on the seabed around structures in the coastal ocean.
   (c) *Ice melting/freezing* – to better represent and hence understand processes by which ice shelves in Antarctica/Greenland evolve, thin, and lead to sea level rise.

5.1 Advection problem (evolving internal solution field – shear & rotation)

This is a simple example where r-adaptivity is used to track an internal solution feature – here the advection of a top hat function in a velocity field with shear and rotation. We use this opportunity to present a result calculated entirely within Firedrake. A PDE-based r-adaptive method was implemented with Firedrake, and coupled to an advection solver also implemented with Firedrake. This particular example used the method of Cenicerous & Hou described in section A.3 The monitor function is given by $\mathbf{M} = m\mathbf{I}$, where m is defined by

$$m = \sqrt{1 + (\nabla u)^2}\,, \tag{1}$$

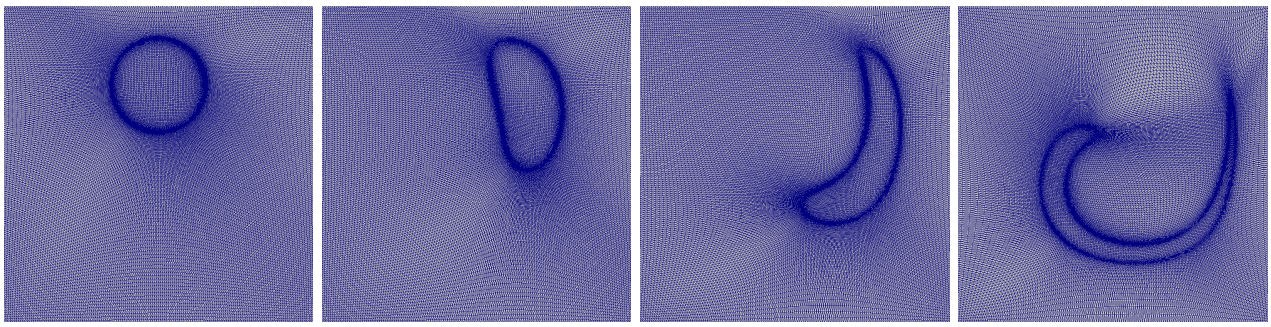

Figure 1: Meshes at a series of time levels computed using r-adaptivity only for the advection problem.

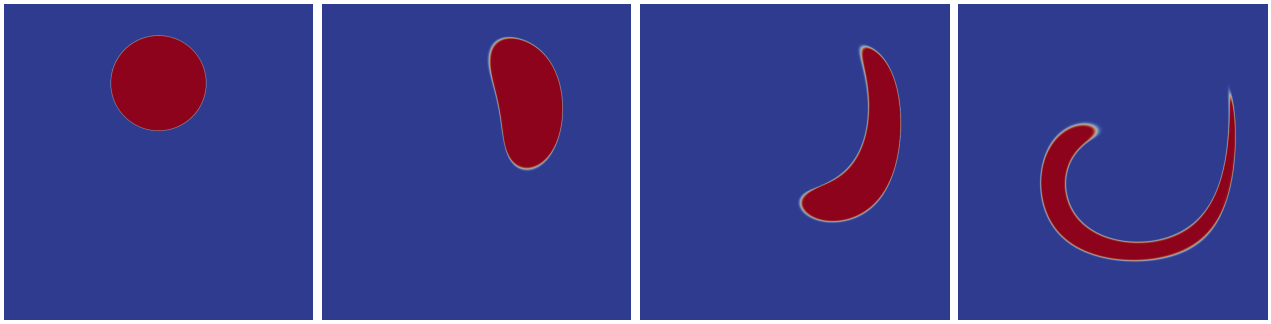

Figure 2: Solution fields at a series of time levels computed using r-adaptivity only for the advection problem.

where u is the field to which we want to adapt the mesh.

We consider the scalar advection equation for a quantity u on domain Ω:

$$\frac{\partial u}{\partial t} + \nabla \cdot (\boldsymbol{c}u) = 0\,, \tag{2}$$

where $\boldsymbol{c}$ is a prescribed velocity field defined on Ω. Cases in both 2D and 3D domains Ω were considered. For completeness we present the details for the 3D case, while only showing results for 2D. In 3D the computational domain is a $[0,1] \times [0,1] \times [0,1]$ cube. At $t = 0$, $u = 0$ everywhere except in a ball of radius 0.35 centered in $(0.35, 0.35, 0.35)$, which models a bubble and where $u = 1$. The following velocity field is considered, so the bubble quickly becomes distorted:

$$\boldsymbol{c}(x,y,z,t) = \begin{cases} 2\sin^2(\pi x)\sin(2\pi y)\sin(2\pi z)\cos(2\pi t/T) \\ -\sin(2\pi x)\sin^2(\pi y)\sin(2\pi z)\cos(2\pi t/T) \\ -\sin(2\pi x)\sin(2\pi y)\sin^2(\pi z)\cos(2\pi t/T) \end{cases}, \tag{3}$$

where T is the period. Here $T = 6$, and the simulation is run until $t = 1.5$.

The moving mesh equation is solved at each advection solver time-step, to ensure that the mesh is always adapted to the advected quantity. Both moving mesh and advection equations are solved using a Lagrange P_1 finite element method with SUPG stabilisation. This is easily achieved thanks to Firedrake: one just has to write the variational formulations in its high-level language, and the corresponding low-level (parallel) solver is automatically generated.

For the 2D case snapshots of the adapted meshes and solutions are shown in figs 1 and 2. The effect of the adaptation is clearly visible: the mesh vertices are moved from the regions far from the advected bubble to the vicinity of the interface of the bubble, and track it as it advects and deforms, so that the accuracy of the solution is improved. Note that as a relatively simple method for the solution of the advection problem is used here (i.e. SUPG), the lack of resolution leads to strong spurious smearing of the solution field which is not seen in the results from fig.2 which use r-adaptivity.

The simulation was run in serial on a 150×150 2D mesh (22,500 vertices and 45,000 triangles), and on 16 cores on a 125×125×125 mesh in 3D (2,500,000 vertices and 12,000,000 tetrahedra).

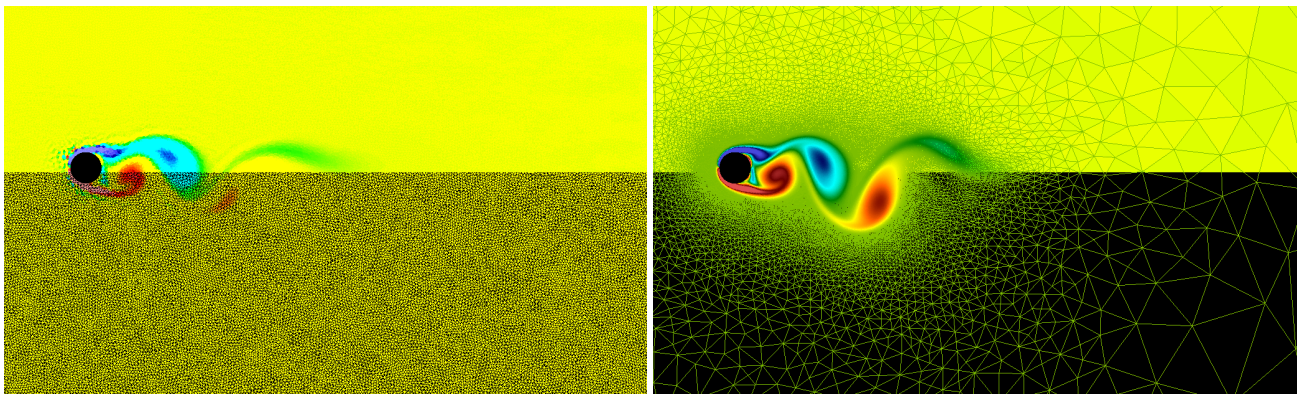

Figure 3: Two snapshots of the oscillating cylinder problem for different mesh movement methods. The left image shows the result with prescribed mesh movement, without *hr*-adaptivity, the right uses viscous drag mesh movement with an *hr*-adaptive mesh. The number of degrees of freedom in each is approximately equal. Here a case for a oscillation amplitude of 0.15 cylinder diameters (a small displacement case) and $f = 0.22$ is shown.

5.2 Oscillating cylinder (evolving internal structure – translation)

Here we visit the problem of the oscillating cylinder in a cross flow. While this is is close to a standard test case for CFD methods with deforming internal boundaries, there does also exist considerable experimental data. This problem was solved using Fluidity, with various choices of mesh movement method examined for their effect on computational accuracy and run-time.

We solve the non-dimensional incompressible constant density Navier-Stokes equations in a 2D domain of size 38.4×25.2, based upon [15]. The inflow condition is $\boldsymbol{u}(0, y, t) = 1$, with free slip conditions on the side walls and do-nothing conditions on the outflow. The domain contains a cylinder of unit diameter, with its centre 8 units from the inlet, at the origin of the local coordinate system. The viscosity value is chosen to give a cylinder Reynolds number of 200. The cylinder is forced to oscillate such that the y-coordinate of the centre of the cylinder satisfies sinusoidal harmonic motion. The use of mesh movement here within an ALE framework allows for an explicit boundary representation for the cylinder, in contrast to the immersed boundary method employed in the earlier paper.

Since the problem involves a moving boundary, there is no constant mesh against which the various mesh movement and adaptivity methods can be compared. Instead we introduce the concept of a *prescribed mesh movement* (as example of the "Lagrangian" like family described above), in which a user-specified grid velocity is applied to move the entire mesh. Since the motion of the cylinder is specified analytically a-priori, this global velocity field can also be specified analytically. Here we take a piecewise linear, continuous field, which represents a continuation, through linear interpolation in the y variable, of the boundary motion into the interior of the domain.

For small displacements of the cylinder, all of the implemented mesh movement methods were found to be able to generate valid meshes for multiple oscillation cycles (refer to benchmarking section). However, for large displacements with amplitudes greater than the cylinder size, the Laplacian and lineal-spring methods were found to give grid velocity fields which lead to tangling and invalid meshes, regardless of the simulation timestep. The viscous drag and lineal-torsional methods were found to be the most robust, with the latter particularly resistant to tangling.

5.3 Turbines and pumps (evolving internal boundary – rotation)

These are examples where structures internal to the fluid domain evolve, either driving or being driven by the underlying fluid dynamics. Of course if these internal structures are also wearing (as in the case of pumps) then this case overlaps with the boundary evolution case described next. Here we have two application areas in mind: (i) pumps & engines, in which an interior component rotates relative to a static exterior, generating unsteady turbulent flows; and (ii) Rotating tidal & wind turbines where we are interested in the accurate simulation of forces on the turbine as well as turbulent turbine wakes for

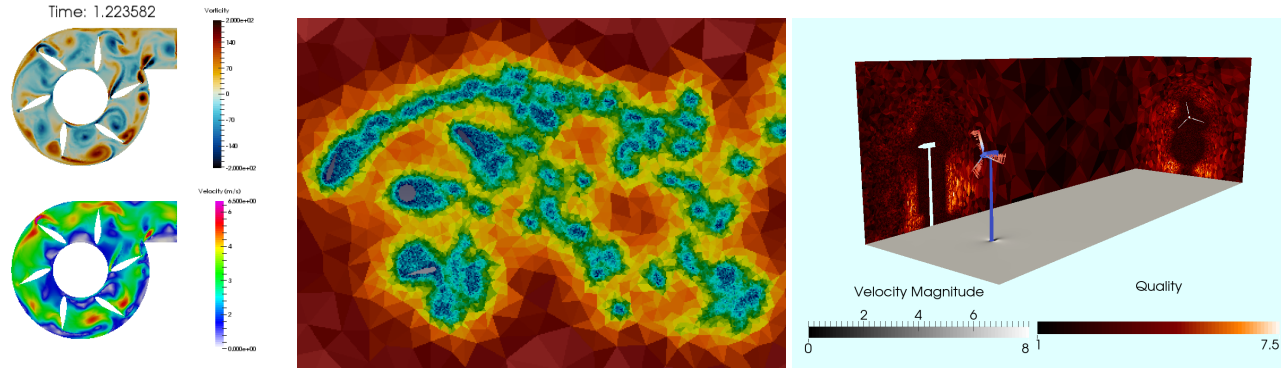

Figure 4: Images showing physical applications of the new functionality in Fluidity, from left to right: vorticity and velocity fields for an idealised 2D centrifugal blower; mesh size from an *hr*-adaptivity simulation of a 2D slice through a rotating Darrieus style vertical axis turbine; and a 3D simulation of mesh quality and flow past a horizontal axis wind turbine.

the purposes of optimising individual turbine designs, and in understanding the interactions between multiple devices for the purposes of optimal array designs and accurate energy resource predictions. The developments made under this project mean that problems previously intractable to us are now possible to consider – some examples are given in fig. 4.

5.4 Seabed scour (evolving external boundary – translation & deformation)

This is an example of a problem where two-way dynamic coupling between an evolving domain/geometry and the resulting fluid dynamics (which in turn drives the boundary evolution) needs to be modelled. The feedback makes this a complicated problem to solve numerically, but is needed for predictive accuracy and to fully understand the underlying processes at play. We have considered test cases of flow past horizontal (infinite) cylinders in 2D and vertical piles in 3D. For the 2D case we have built on the example presented in [28] which made use of an existing serial Laplacian smoothing implementation via Dolfin. As part of this work we are now making use of the lineal-torsional mesh movement method, fully functioning in combination with mesh optimisation. For this test case the latter is particularly needed as large deformation of the boundary close to the pipe leads to pinching as well as significant stretching of the mesh which *r*-adaptivity in isolation cannot deal with robustly.

6 Benchmarking & profiling

As part of the embedded CSE project, the various mesh movement methods were compared for their parallel scaling efficiency within Fluidity on ARCHER. Weak and strong scaling results in the case of the oscillating cylinder application (with the same parameters as given for fig. 3 – although scaling results were found to be fairly insensitive to these choices, assuming the method did not tangle) are shown in fig. 5, with run times normalized by those achieved by the simulation with prescribed grid velocity (as described in section 5.2).

As predicted, the Laplacian smoother scales particularly robustly, increasing runtime by less than 20% across 96 cores. However, since the method has been shown to be unstable with large displacements, the linear elasticity analogy, which has a slightly larger overhead, but similarly good scaling behaviour, is recommended. By comparison, running the code applying full *hr*-adaptivity on every simulation timestep produced a roughly 300% slow down, compared to a run with the same target resolution on a fixed topology mesh,

Profiling the individual runs (fig. 6) showed a similar pattern across all mesh movement algorithms, with the largest additional cost coming from generating the solution of the elliptic problem, rather than assembly of the discretized operators. Indeed even the most complex method, the lineal-torsional approach, had assembly costs which were a fraction of those involved in calculating the fluid pressure.

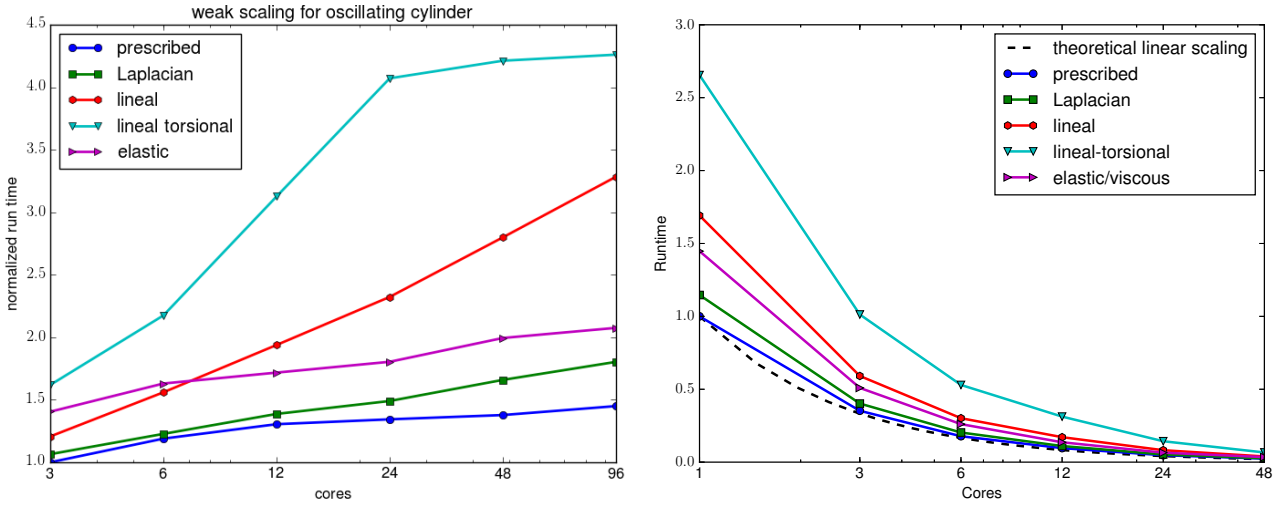

Figure 5: Weak (left) and strong (right) scaling results for the various r-adaptivity methods performed using Fluidity on the ARCHER HPC system. 'Prescribed' refers to the analytical method of defining the mesh velocity as described in section 5.2, while the other methods are as described in the appendix.

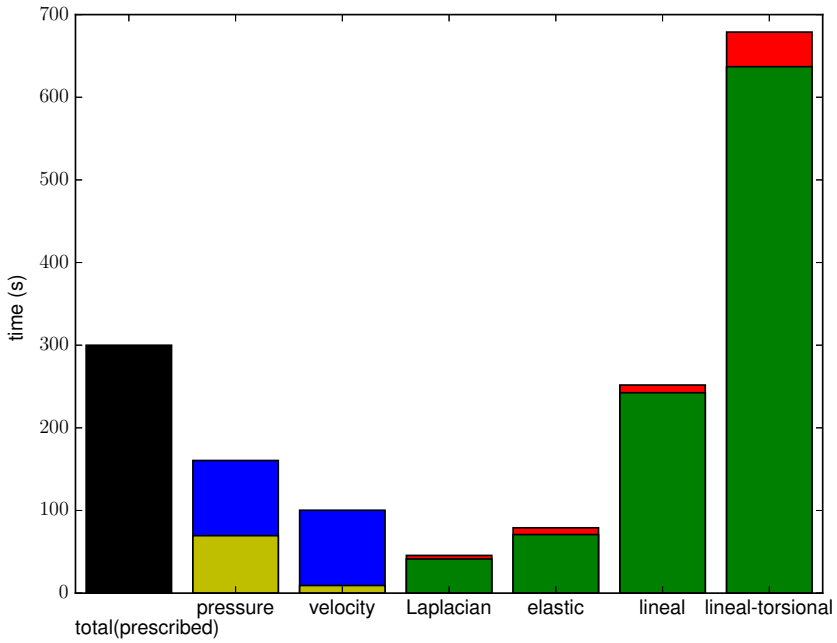

Figure 6: Breakdown of profiling data for the Fluidity implementation of the mesh movement methods for the oscillating cylinder problem on 24 cores of the ARCHER HPC system. The black bar shows the total runtime for the fluid solver, using prescribed mesh movement, with further splits for the pressure and velocity solves, coloured by matrix assembly (blue) and linear solve time (yellow). This is compared against the assembly (red) and solve time for four mesh movement algorithms.

7 Achievements

The main achievements of this project may be summarised as follows:

1. We have implemented, validated and benchmarked three families of highly parallelised mesh movement methods; this delivers on our aim of parallelising a pre-existing prototype method as well as significantly expanding on both the number as well as types of methods available to us.

2. These have been integrated with Fluidity, as well as its internal mesh optimisation library, to deliver fully-functional *r*- as well as *hr*-adaptivity capabilities to users; associated supporting manual material and test cases having been written.

3. The methods have been demonstrated on all the target applications discussed in the proposal. For cases with strong external boundary deformation, or internal boundary/structure rotation, we are now in a position to tackle, in parallel, problems which were not previously tractable with our existing *h*-adaptive methods alone.

4. Based upon associated testing and benchmarking the following initial observations can be made: *Laplacian smoothing* (available in 1D, 2D, and 3D) provides a simple and straightforward way to smooth a mesh that has undergone a small (relative to the domain size) internal/external boundary deformation. The *lineal-spring* method provides a more robust (less likely to tangle) smoother, primarily because it couples mesh movement in the $\hat{x}$- and $\hat{y}$-directions, whilst not adding too much in terms of computational cost. Important to keep in mind for this method, however, is that it only prevents inter-nodal collisions and does not ensure that the mesh will not tangle. For that, the *lineal-torsional spring* method was implemented which effectively guaranteed a mesh would not tangle (owing to the method accounting for elemental area, and not just edge-length), but at the expense of element quality, and overall computational cost. The *linear elastic* and *viscous drag* methods have proven to be a good compromise in terms of parallel efficiency and robustness. For rotational problems the *viscous drag* method appears more robust than the elastic method, but is less robust than the more costly *lineal-torsional* method.

   Worth noting are the following caveats: (i) The one-parameter elliptic method is currently limited to Laplacian (i.e. not Winslow smoothing) in Fluidity; and (ii) The current implementation of the *lineal-torsional* method is limited to 2D problems, since the complex spring terms needed to control a tetrahedron volume in 3D are not currently implemented (although this is a straightforward extension).

5. Examples of PDE-based mesh movement methods (namely Cenicerous & Hou) have also been demonstrated as functioning in combination with Firedrake-based application codes. An approach to allow a full *hr*-capability through the additional use of PRAgMaTIc is currently being tested. Implementation of the wider range of mesh movement algorithms as discussed above and in the appendix with Firedrake will be the subject of future work.

8 Conclusions & future work

This project has achieved its primary goal of making a range of *r*-adaptive, as well as combined *hr*-adaptive, methods available to users of Fluidity. These have been parallelised, benchmarked on ARCHER, and tested across a range of applications which form representative examples for all of the diverse real-world use-cases which motivated this work. For Firedrake-based models, a subset of these *r*-adaptive methods have also been demonstrated, while full *hr*-adaptivity, applications and benchmarking against similar capabilities now available in Fluidity are the subject of ongoing work. In addition, through further method benchmarking and real-world applications, future work will seek to identify the most appropriate *r*- and/or *hr*-strategy to employ for a given problem type.

9 Acknowledgements

This work was funded under the embedded CSE programme of the ARCHER UK National Supercomputing Service (http://www.archer.ac.uk). The authors would also like to thank the complementary support provided by the EPSRC projects EP/L000407/1 and EP/M011054/1, and industrial research funding from the Weir Group.

References

[1] M. Abolghasemi, M. Piggott, J. Spinneken, A. Vire, C. Cotter, and S. Crammond. Simulating tidal turbines with multi-scale mesh optimisation techniques. *Journal of Fluids and Structures*, 66:69–90, 2016.

[2] F. Alauzet. A changing-topology moving mesh technique for large displacements. *Engineering with Computers*, 30(2):175–200, 2014.

[3] S. Balay, W. D. Gropp, L. C. McInnes, and B. F. Smith. Efficient management of parallelism in object oriented numerical software libraries. In E. Arge, A. M. Bruaset, and H. P. Langtangen, editors, *Modern Software Tools in Scientific Computing*, pages 163–202. Birkhäuser Press, 1997.

[4] N. Barral, M. G. Knepley, M. Lange, M. D. Piggott, and G. J. Gorman. Anisotropic mesh adaptation in Firedrake with PETSc DMPlex. In *International Meshing Roundtable*, 2016.

[5] C. Budd, R. Russell, and E. Walsh. The geometry of r-adaptive meshes generated using optimal transport methods. *Journal of Computational Physics*, 282:113 – 137, 2015.

[6] C. J. Budd, W. Huang, and R. D. Russell. Adaptivity with moving grids. *Acta Numer.*, 18:111–241, 2009.

[7] H. D. Ceniceros and T. Y. Hou. An efficient dynamically adaptive mesh for potentially singular solutions. *Journal of Computational Physics*, 172(2):609 – 639, 2001.

[8] J. Donea, A. Huerta, J. Ponthot, and A. Rodrïguez-Ferran. Arbitrary Lagrangian-Eulerian Methods. In *Encyclopedia of Computational Mechanics*, pages 1–25. John Wiley & Sons, Ltd, Chichester, UK, nov 2004.

[9] C. Farhat, B. Degand, B. Koobus, and M. Lesoinne. Torsional springs for two-dimensional dynamic unstructured fluid meshes. *Computer Methods in Applied Mechanics and Engineering*, 163:231–245, 1998.

[10] P. Farrell, D. Ham, S. Funke, and M. Rognes. Automated Derivation of the Adjoint of High-Level Transient Finite Element Programs. *SIAM J. Sci. Comput.*, 35(4):C369–C393, 2013.

[11] P. E. Farrell, M. D. Piggott, C. C. Pain, G. J. Gorman, and C. R. Wilson. Conservative interpolation between unstructured meshes via supermesh construction. *Comput. Methods Appl. Mech. Eng.*, 198(33–36):2632–2642, 1 July 2009.

[12] S. Funke, P. Farell, and M. Piggott. Tidal turbine array optimisation using the adjoint approach. *Renewable Energy*, 63, 2014.

[13] G. Gorman, J. Southern, P. Farrell, M. Piggott, G. Rokos, and P. Kelly. Hybrid openmp/mpi anisotropic mesh smoothing. *Procedia Computer Science*, 9:1513 – 1522, 2012.

[14] G. J. Gorman, C. C. Pain, M. D. Piggott, A. P. Umpleby, P. E. Farrell, and J. R. Maddison. Interleaved parallel tetrahedral mesh optimisation and load-balancing. In P. Bouillard and P. Diez, editors, *Adaptive Modeling and Simulation (ADMOS) 2009*, pages 101–104, 2009.

[15] E. Guilmineau and P. Queutey. a Numerical Simulation of Vortex Shedding From an Oscillating Circular Cylinder. *Journal of Fluids and Structures*, 16(6):773–794, aug 2002.

[16] X. Guo, G. Gorman, M. Lange, L. Mitchell, and M. Weiland. Exploring the thread-level parallelisms for the next generation geophysical fluid modelling framework fluidity-icom. *Procedia Engineering*, 61:251 – 257, 2013.

[17] X. Guo, M. Lange, G. Gorman, L. Mitchell, and M. Weiland. Developing a scalable hybrid mpi/openmp unstructured finite element model. *Computers & Fluids*, 110:227 – 234, 2015. ParCFD 2013.

[18] H. R. Hiester, M. D. Piggott, and P. A. Allison. The impact of mesh adaptivity on the gravity current front speed in a two-dimensional lock-exchange. *Ocean Model.*, 38(1–2):1–21, 2011.

[19] H. R. Hiester, M. D. Piggott, P. E. Farrell, and P. A. Allison. Assessment of spurious mixing in adaptive mesh simulations of the two-dimensional lock-exchange. *Ocean Model.*, 73(0):30–44, Jan. 2014.

[20] J. Hill, M. D. Piggott, D. A. Ham, E. E. Popova, and M. A. Srokosz. On the performance of a generic length scale turbulence model within an adaptive finite element ocean model. *Ocean Model.*, 56(0):1–15, Oct. 2012.

[21] J. Hill, E. E. Popova, D. A. Ham, M. D. Piggott, and M. Srokosz. Adapting to life: ocean biogeochemical modelling and adaptive remeshing. *Ocean Sci.*, 10(3):323–343, 9 May 2014.

[22] C. T. Jacobs, G. S. Collins, M. D. Piggott, S. C. Kramer, and C. R. G. Wilson. Multiphase flow modelling of volcanic ash particle settling in water using adaptive unstructured meshes. *Geophys. J. Int.*, 192(2):647–665, 1 Feb. 2013.

[23] C. Pain, A. Umpleby, C. De Oliveira, and A. Goddard. Tetrahedral mesh optimisation and adaptivity for steady-state and transient finite element calculations. *Computer Methods in Applied Mechanics and Engineering*, 190(29):3771–3796, 2001.

[24] S. D. Parkinson, J. Hill, M. D. Piggott, and P. A. Allison. Direct numerical simulations of particle-laden density currents with adaptive, discontinuous finite elements. *Geoscientific Model Development*, 7(5):1945–1960, 2014.

[25] M. D. Piggott, P. E. Farrell, C. R. Wilson, G. J. Gorman, and C. C. Pain. Anisotropic mesh adaptivity for multi-scale ocean modelling. *Philos. Trans. A Math. Phys. Eng. Sci.*, 367(1907):4591–4611, 28 Nov. 2009.

[26] M. D. Piggott, G. J. Gorman, C. C. Pain, P. A. Allison, A. S. Candy, B. T. Martin, and M. R. Wells. A new computational framework for multi-scale ocean modelling based on adapting unstructured meshes. *Int. J. Numer. Methods Fluids*, 56(8):1003–1015, 2008.

[27] M. D. Piggott, C. C. Pain, G. J. Gorman, D. P. Marshall, and P. D. Killworth. Unstructured adaptive meshes for ocean modeling. In H. Hasumi and M. W. Hecht, editors, *Ocean Modeling in an Eddying Regime*, pages 383–408. AGU, 2008.

[28] J. M. N. Rattia, J. R. Percival, B. Y. B, S. Neethling, and M. D. Piggott. Numerical Simulation of Scour below Pipelines using Flexible Mesh Methods. In *8th International Conference on Scour and Erosion*, 2016.

[29] A. M. Winslow. Numerical solution of the quasilinear poisson equation in a nonuniform triangle mesh. *Journal of Computational Physics*, 1(2):149 – 172, 1967.

[30] P. Woodhams, J. Hill, and P. Farrell. Improving Load Balancing and Parallel Partitioning in Fluidity. HECToR dCSE technical report,, 2012.

A Appendix: r-adaptive mesh movement methods

A.1 Laplacian smoothing

Here the mesh velocity, $\boldsymbol{v}$, is assumed to satisfy a PDE,

$$\nabla^2_{\boldsymbol{\xi}} \boldsymbol{v} = \mathbf{0}, \quad \boldsymbol{\xi} \in \Omega, \qquad \boldsymbol{v} = \boldsymbol{v}^D, \quad \boldsymbol{\xi} \in \partial\Omega,$$

where the computational mesh coordinate $\boldsymbol{\xi} := \boldsymbol{x}(t_0)$ is the initial physical coordinate *at the beginning of the simulation*. Since the individual components of $\boldsymbol{v}$ are not coupled, this equation can be solved using standard numerical methods quickly and compactly.

The Fluidity implementation builds and solves a parallel finite element discretisation of the PDE using C code based on the data structures and linear solvers of the PETSc software suite [3], coupled to the preexisting mesh decompositions used within the Fluidity fluids solver. In Firedrake the PDE based nature of this simple method makes this trivial to implement (cf. the example in the case of the more complex methods presented in section 5.1).

A.2 Winslow smoothing

In [29], Winslow introduces a mesh smoothing technique similar to Laplacian smoothing, but instead based upon the mapping $\boldsymbol{\xi} \equiv \boldsymbol{\xi}(\mathbf{x})$. Following this through gives an implicit elliptic equation

$$\nabla^2_{\boldsymbol{x}} \boldsymbol{\xi} = 0, \quad \boldsymbol{\xi} \in \Omega.$$

This may be rewritten in terms of the dependent variable $\boldsymbol{x}$ as

$$\left[\boldsymbol{G}^{-1}\right]_{ij} \frac{\partial^2 \boldsymbol{x}}{\partial \xi_i \partial \xi_j} = \mathbf{0}, \quad \text{where} \quad \boldsymbol{G} = \nabla_{\boldsymbol{\xi}} \boldsymbol{x} \cdot (\nabla_{\boldsymbol{\xi}} \boldsymbol{x})^T.$$

Note that these equations are nonlinear, making them more difficult to solve than the equations for Laplacian smoothing.

A.3 The method of Ceniceros & Hou

The method of Ceniceros & Hou [7] is motivated by the desire to minimize a measure of the gradient of some PDE solution u in the computational domain; i.e., in one-dimension

$$\min_{x(\xi)} \int_{\Omega_C} \sqrt{1 + u_\xi^2(x(\xi))}\, d\xi = \min_{x(\xi)} \int_{\Omega_C} \sqrt{1 + u_x^2(x(\xi)) x_\xi^2}\, d\xi.$$

The Euler–Lagrange form of this equation is

$$\left(\frac{u_x^2}{\sqrt{1+u_x^2 x_\xi^2}} x_\xi \right)_\xi = \frac{u_x u_{xx} x_\xi^2}{\sqrt{1+u_x^2 u_\xi^2}}. \tag{4}$$

Through numerical experiments, Ceniceros & Hou determined that (4) still produces satisfactory spreading of the mesh, and is much easier to solve, when the RHS is set to zero. Setting the RHS to zero and modifying the LHS to avoid degeneracy, they obtain

$$\left(\frac{1+u_x^2}{\sqrt{1+u_x^2 x_\xi^2}} x_\xi \right)_\xi = 0.$$

Replacing u_x in the numerator with $u_\xi = u_x x_\xi$ for smoother solutions gives

$$(M x_\xi)_\xi = 0, \quad \text{where} \quad M = \sqrt{1+u_x^2 x_\xi^2}.$$

Ceniceros & Hou note that this equation resembles the 1D Winslow equation with derivatives in computational, rather than physical, space. As with all methods based upon the mapping $\mathbf{x} = \mathbf{x}(\boldsymbol{\xi})$, the mesh generation equations are relatively simple but also more likely to result in mesh folding than equations based upon the reverse mapping.

In 2D, the equations become

$$\nabla_{\boldsymbol{\xi}} \cdot (M \nabla_{\boldsymbol{\xi}} x) = 0, \quad \nabla_{\boldsymbol{\xi}} \cdot (M \nabla_{\boldsymbol{\xi}} y) = 0. \tag{5}$$

These are the Euler–Lagrange equations for the functional

$$I(x,y) = \frac{1}{2} \int_{\Omega_C} \left[\nabla x^T \mathbf{M} \nabla x + \nabla y^T \mathbf{M} \nabla y \right] d\boldsymbol{\xi}, \tag{6}$$

with $\mathbf{M} = m\mathbf{I}$. Ceniceros & Hou give a general monitor function for the 2D equations of the form $\mathbf{M} = m\mathbf{I}$, with

$$m = \sqrt{1 + \beta^2 \left| \nabla_{\boldsymbol{\xi}} u^2 \right| + g^2(u)},$$

where β is a scaling factor and $g(u)$ is chosen based upon the properties of the time-dependent physical problem being solved; see [7]. Ceniceros & Hou suggest solving (5) by setting the left-hand side equal to the derivative of the independent variable in pseudotime τ; i.e.

$$\nabla_{\boldsymbol{\xi}} \cdot (M \nabla_{\boldsymbol{\xi}} x) = x_\tau, \quad \nabla_{\boldsymbol{\xi}} \cdot (M \nabla_{\boldsymbol{\xi}} y) = y_\tau.$$

The solution can then be marched forward in time to steady state (within some tolerance).

A.4 Winslow's variable diffusion method

In [29], Winslow generalized the method presented in section A.2 to

$$\nabla_{\mathbf{x}} \cdot (w \nabla_{\mathbf{x}} \boldsymbol{\xi}) = \mathbf{0}, \tag{7}$$

where $w(\mathbf{x}) > 0$ is chosen to control the distribution of mesh resolution in the interior of Ω. Solving (7) is equivalent to minimizing the Winslow functional with $\mathbf{M} = (\frac{1}{w})\mathbf{I}$. Inverting (7), we obtain

$$\begin{aligned} \alpha x_{\xi\xi} - 2\beta x_{\xi\eta} + \gamma x_{\eta\eta} &= -\left(w_\xi y_\eta - w_\eta y_\xi \right) \frac{J}{w}, \\ \alpha y_{\xi\xi} - 2\beta y_{\xi\eta} + \gamma y_{\eta\eta} &= -\left(w_\xi x_\eta - w_\eta x_\xi \right) \frac{J}{w}, \end{aligned}$$

where the Jacobian of the forward mapping, $J := x_\xi y_\eta - x_\eta y_\xi$ and α, β, and γ are defined as above.

A.5 Linear elastic analogy

Here we solve an equation modelled on the stress term for linear elastic solid,

$$\nabla_{\boldsymbol{X}} \cdot \sigma\left(\boldsymbol{x}\right) = 0, \qquad \boldsymbol{x} = \boldsymbol{x}^D, \quad \boldsymbol{\xi} \in \delta\Omega,$$

and σ is a symmetric displacement -based "mesh stress" tensor

$$\sigma = \lambda \nabla_{\boldsymbol{X}} \cdot \boldsymbol{x} I + \mu \left(\nabla_{\boldsymbol{X}} \boldsymbol{x} + \nabla_{\boldsymbol{X}} \boldsymbol{x}^T\right),$$

with two user specified Lamé parameters, λ and μ, expressing the mesh's resistance to compression and shearing respectively. Since $\boldsymbol{X}$ is fixed in time, this is implemented detail, in the form

$$\nabla_{\boldsymbol{X}} \cdot \dot{\sigma}\left(\boldsymbol{v}\right) = 0, \qquad \dot{\sigma} = \lambda \nabla_{\boldsymbol{X}} \cdot \boldsymbol{v} I + \mu \left(\nabla_{\boldsymbol{X}} \boldsymbol{v} + \nabla_{\boldsymbol{X}} \boldsymbol{v}^T\right),$$

for the grid velocity directly. This coupled vector PDE is discretized and assembled using standard P1 finite element methods already coded in Fortran 2003, with the resulting linear equation solved in parallel using the solvers from PETSc.

A.6 Lineal spring analogy

The linear elastic model above solves a continuum force balance equation on a discrete finite element mesh. However, the equations themselves can also be generated discretely [9]. The lineal spring analogy approach models fictitious springs along each mesh edge, with spring constant inversely proportional to the edge length. Taking the finite dimensional vectors of original and solution simplex vertex coordinates, $\overline{\boldsymbol{X}}$ and $\overline{\boldsymbol{x}}$, this gives a matrix stiffness problem

$$\boldsymbol{K}(\overline{\boldsymbol{x}} - \overline{\boldsymbol{X}}) = 0,$$

with the boundary displacements known. The lineal spring stiffness, $\boldsymbol{K}^{ij}_{\text{lineal}}$ between nodes i and j is defined as

$$\boldsymbol{K}^{ij}_{\text{lineal}} = \frac{1}{l_{ij}} \begin{bmatrix} \cos\alpha^2 & \sin\alpha\cos\alpha & -\cos\alpha^2 & -\sin\alpha\cos\alpha \\ \sin\alpha\cos\alpha & \sin\alpha^2 & -\sin\alpha\cos\alpha & -\sin\alpha^2 \\ -\cos\alpha^2 & -\sin\alpha\cos\alpha & \cos\alpha^2 & \sin\alpha\cos\alpha \\ -\sin\alpha\cos\alpha & -\sin\alpha^2 & \sin\alpha\cos\alpha & \sin\alpha^2 \end{bmatrix}^{ij},$$

where l_{ij} is the length between nodes i and j, and α the angle formed between the $\hat{\boldsymbol{x}}$-axis and the vector connecting nodes i and j. The Fluidity implementation calls C code to assemble and solve (using PETSc) the matrix problem in parallel for the graph given by the mesh decomposition.

A.7 Viscous drag analogy

Here we solve a similar "force-balance" equation

$$\nabla_{\boldsymbol{x}} \cdot \sigma\left(\boldsymbol{v}\right) + \beta(\boldsymbol{u} - \boldsymbol{v}) = 0, \qquad \boldsymbol{v} = \boldsymbol{v}^D, \quad \boldsymbol{\xi} \in \delta\Omega$$

with σ the symmetric "mesh stress" tensor

$$\sigma = \lambda \nabla_{\boldsymbol{x}} \cdot \boldsymbol{v} I + \mu \left(\nabla_{\boldsymbol{x}} \boldsymbol{v} + \nabla_{\boldsymbol{x}} \boldsymbol{v}^T\right).$$

Whereas the previous method applies the analogy of the mesh as an elastic medium, here we treat it as an massless, inertia-free viscous inter-penetrating fluid. The solution method is identical to that used for a linear elastic mesh.

A.8 Lineal-torsional smoothing

This approach [9] is a direct extension of the lineal spring analogy. What sets the two methods apart is that $\boldsymbol{K}$ is no longer solely a function of elemental edge length, but also elemental area. Conceptually this means $\boldsymbol{K} = \boldsymbol{K}^{ij}_{\text{lineal}} + \boldsymbol{K}^{ijk}_{\text{torsional}}$. Elemental (triangle $\mathcal{T}_{ijk}$ in this case) area contributes to torsional spring stiffness at vertex i via

$$C_i^{ijk} = \frac{l_{ij}^2 l_{ik}^2}{4A_{ijk}^2}, \tag{8}$$

where l_{ij} is the distance between vertices i and j, l_{ik} is the distance between vertices i and k, and A_{ijk} is the area of the triangle $\mathcal{T}_{ijk}$. After taking into account the remaining vertices,

$$\boldsymbol{C}_{ijk} = \begin{bmatrix} C_i^{ijk} & 0 & 0 \\ 0 & C_j^{ijk} & 0 \\ 0 & 0 & C_k^{ijk} \end{bmatrix}, \tag{9}$$

and constructing with the rotation matrix

$$\boldsymbol{R}^{ijk} = \begin{bmatrix} b_{ik} - b_{ij} & a_{ij} - a_{ik} & b_{ij} & -a_{ij} & -b_{ik} & a_{ik} \\ -b_{ji} & a_{ji} & b_{ji} - b_{jk} & a_{jk} - a_{ji} & b_{jk} & -a_{jk} \\ b_{ki} & -a_{ki} & -b_{kj} & a_{kj} & b_{kj} - bki & a_{ki} - a_{kj} \end{bmatrix}, \tag{10}$$

where $a_{ij} = \frac{x_{ij}}{l_{ij}^2}$ and $b_{ij} = \frac{y_{ij}}{l_{ij}^2}$, the torsional stiffness is defined as

$$\boldsymbol{K}^{ijk}_{\text{torsional}} = \boldsymbol{R}^{ijk^T} \boldsymbol{C}^{ijk} \boldsymbol{R}^{ijk}. \tag{11}$$

Here it is important to keep in mind that $\boldsymbol{K}^{ijk}_{\text{toriosnal}}$ does not entirely safeguard against mesh tangling, and a further step must be taken to limit the rotation, $\boldsymbol{R}^{ijk}$, of each triangle within the mesh as well. The Fluidity implementation calls C code to assemble and solve (using PETSc) the matrix problem in parallel for the given mesh decomposition.