OPTIMISING A ONE MILLION BLOCK GRID FOR A TVZ FLOW MODEL

M. Letourneur[1], D. Dempsey[1], J. O'Sullivan[1], A. Croucher[1], M. O'Sullivan[1]

[1]Department of Engineering Science, University of Auckland, Auckland, New-Zealand

martin.letourn@gmail.com

Keywords: *Optimisation, unstructured grids, finite volume, orthogonality, flow models, minimisation, objective function, Waiwera.*

ABSTRACT

The recently developed multiphase geothermal flow simulator, Waiwera, is now able to handle unstructured grids. However, to reduce errors related to the grid, certain quality factors should be optimised. In applying the finite volume method, Waiwera assumes that for each pair of neighbouring tetrahedral cells the line connecting their centroids is perfectly orthogonal to their shared interface. Any deviation from orthogonality will introduce an error in the flow simulation. Our work aims to optimize a given grid so as to minimise departures from orthogonality as much as possible. Our algorithm is written in Python. It uses the MeshIO library to read and write various formats of mesh files, and the SciPy library for optimisation algorithms. Our approach is to minimise the sum of all deviations from orthogonality within the mesh. The inputs are the node coordinates and connectivity and any surface constraints (e.g., model boundaries, faults). Then, our algorithm moves the nodes around in space in an attempt to minimise deviations from orthogonality.

Several challenges were addressed in developing the optimisation algorithm. First, we require that some nodes should be constrained to move only along boundaries (model edges, topography, faults) so that geometric features honoured by the original mesh are not altered. This reduces a node's degrees of freedom from 3 – the number of coordinate directions - to (3 – n), where n is the number of boundaries the node lies on. As a result, the algorithm can handle plane and complex boundaries (including faults and topography), intersections between planes and between a plane and a complex surface.

Second, we aimed to minimise the running time of the algorithm so that the approach is practical for full-sized meshes (up to one million blocks). This included implementing and passing analytical Jacobians to the optimiser, and the use of multiprocessing to subdivide and parallelise optimisation of mesh subsections. The mesh is divided into m sets of disjoint (non-overlapping, non-adjacent) node clusters, where $\boldsymbol{m}$ is the number of parallel processors, and each cluster contains $\boldsymbol{k}$ nodes. The coordinates of the $\boldsymbol{k}$ nodes are optimised and updated before a new set of $\boldsymbol{m}$ disjoint clusters are chosen. Calibrating tolerance parameters carefully, we are able to achieve convergence of an optimal mesh in less than 10 iterations (for a 10^4 node mesh we have achieved optimisation in just over 90 seconds).

In this paper, we present the algorithm details, several test cases, and a challenge mesh: a one million block grid of the TVZ, with realistic topography, and approximate faults and basement contact.

1. INTRODUCTION

Multiphase geothermal flow simulators such as TOUGH2 and Waiwera allow for simulations to be run on unstructured grids. This allows the user to import tetrahedral grids, which are able to capture with more precision complex geological features, such as topographic surfaces, faults and lithological contacts, than a structured hexahedral grid. To ensure accuracy in the calculations, it is useful to optimize for some quality factors of the grid. When Waiwera applies the finite volume method it assumes that for any pair of neighbouring tetrahedra the line connecting their centroids is orthogonal to the shared face (*Croucher and O'Sullivan*, 2013). Therefore, if the grid is not optimized in this respect, a divergence between the calculated and theoretical flow directions will occur (Figure 1).

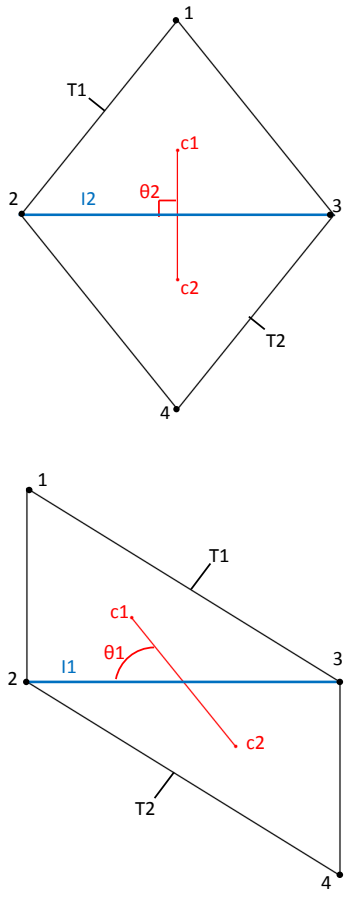

Figure 1: Two cases of connection-interface orthogonality in 2D. (top) A pair of blocks in which the connector between centroids, line c1-c2, is orthogonal to the shared interface, 2-3. (bottom) An instance of non-orthogonality for the same interface. Optimisation could improve the mesh by moving nodes 1 and 4 in the bottom configuration to the positions in the top.

Unstructured meshing software such as GMSH already apply an optimisation after triangulating the mesh. These software optimize the grid with respect to the aspect ratio (angles, sides, length ratio) but not with respect to connection-interface orthogonality. This is because grids built by this software are intended for use within a simulator that applies the finite element method (*Geuzaine and Remacle*, 2009). Requirements are different for the finite volume method. Therefore, it was necessary to develop our own optimisation algorithm. Similar problems have been solved in the past for finite element grid optimisation (*Pain et al.,* 2001) and tetrahedral grid smoothing (*Escobar et al.,* 2003). The main strategy used is the minimisation of an objective function. Although our quality factors are different, we can apply a similar optimisation approach in our algorithm.

2. METHODS

In this section, we describe the main features of our optimisation algorithm. We will first present the general components found in most optimisation problems. Then, we will focus on the profiling methods specific to our problem.

2.1 Formulating the optimisation problem

2.1.1 Objective function

Our approach to optimisation is to minimise the deviation from orthogonality for all connections in the grid. To achieve this, we investigated a class of quality metrics call p-norms:

$$f = \left(\sum_i |deviation_i|^p \right)^{\frac{1}{p}}.$$

These allow us to vary the relative weight of the poorest connections, directing the optimisation algorithm to focus on these, e.g.:

$$\lim_{p \to \infty} f = \max(|deviation|).$$

Our findings suggested that $p = 1$ was optimal in terms of overall grid quality and algorithm running time. The same conclusions were drawn in the work of *Escobar et al.* (2003). Therefore the objective function was reduced to the simple sum of all deviations.

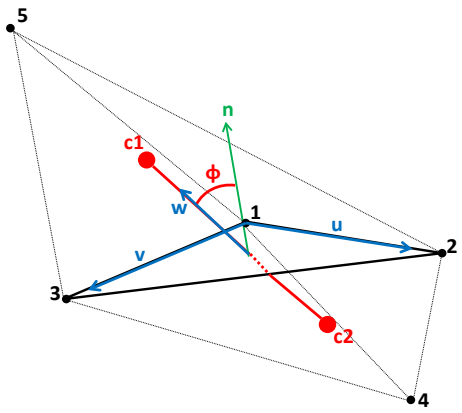

Figure 2: System of two neighbouring tetrahedra. The centroids of the tetrahedron (1235) and (1234) are respectively c1 and c2.

To compute the deviation, we take the cross product of two vectors, **u** and **v**, defining edges of a triangular interface, which yields the interface normal, **n** (Figure 2). We then take the dot product of the normal and a vector, **w,** connecting the two block centroids (Figure 2), which gives the cosine of the angular deviation, ϕ, from orthogonality $\varepsilon = \mathbf{w} \cdot (\mathbf{u} \times \mathbf{v})$. Finally, the deviation is computed $1 - |\varepsilon|$, which is zero (a minimum) for an orthogonal connection.

2.1.2 Degrees of freedom

In Figure 3, we can see how the degrees of freedom in displacing a node varies with its position in the mesh:

- Node 6, lies within the volume of our model, and is not on an internal surface. It can move freely in x and y directions (and z if this was a 3D mesh). It has a degree of freedom of two which means that both its coordinates can be modified during optimisation
- Node 5, lies on an external boundary. In this case, the node can only displace in the y direction, otherwise the model domain would change. It has a degree of freedom of one.
- Nodes 1-4 are fixed during the optimisation, they have 0 degrees of freedom.

In 3D, a node on a boundary surface will have one of its coordinates fixed depending on the value of the other two. Thus, one task of our algorithm is to determine mathematical expressions linking the dependent coordinates to the free ones for arbitrary surfaces. For nodes on plane surfaces, we use an equation of the form: $ax + by + cz + d = 0$, where

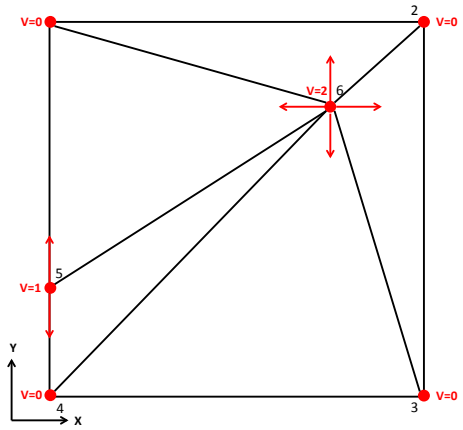

Figure 3: 2D triangular mesh showing in red the degree of freedom of each node as well as its possible direction of displacement.

a, b, c and d are determined by points on the plane.

The intersection of two planes gives a line whose equation can also be easily accessed by solving the equation system formed by the two plane equations.

Although many models are exclusively bounded by plane surfaces, others can include complex surfaces such as a topographic upper layer. Degree of freedom reduction on such surfaces is more complex to describe. Our approach is to describe the complex surface using 2D splines, as these ensure both a continuous surface and continuous derivatives

at spline boundaries. A 2D example of curve fitting using splines is showed in Figure 4.

Our algorithm is also able to handle the case of a plane intersecting a complex surface, such as might occur at the surface expression of a fault. When a complex interface intersects a plane it yields a curve that lies on the plane. This curve can be described using splines in the coordinate system (w, v) which is specific to the plane. The simplest case is if the plane is either vertical or horizontal, in which case the coordinate system (w, v) becomes respectively (x, z)or (y, z) and (x, y), in that case no projections are needed to transform from one coordinate basis to another and the curve resulting from the intersection can be described in our (x, y, z) space. In other cases, a more complex projection is required to transform from (w, v) to (x, y, z) coordinates.

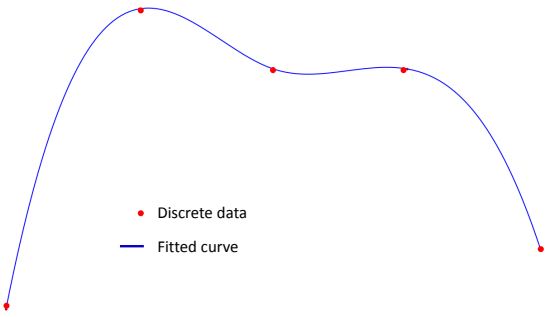

Figure 4: 2D dataset of 5 nodes fitted using splines.

Figure 5 shows a more complex example, considering an inclined plane, $P(x, y, z): y - z = 0$, and a complex surface $CS(x, y, z)$. The intersection, represented by the red nodes, is the curve I which is expressed with respect to the space (w, v) specific to the plane P (Figure 5). In this case, the axes w and v are coincident. The axis w, on the other hand, has a more complex relationship with the (x, y, z) space. Using the dip of the plane θ, we can express w as:

$$w = y\cos(\theta) + z\sin(\theta).$$

In that case any nodes on $I(w, x)$ will have a degree of freedom of 1. Any displacement along x will give a new value of w. This value of w can then be translated into the corresponding values of y and z using both the equation of P and the relation of w with respect to y and z.

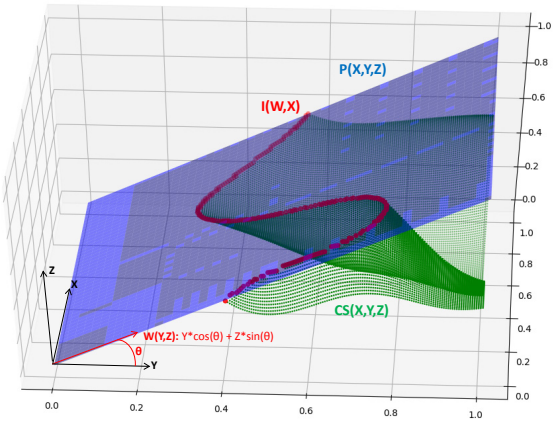

Figure 5: 3D plot of a plane, P, in blue, a complex surface, CS, in green and their intersection, I, as red dots.

2.1.3 Optimisation algorithm

As our goal in optimising the mesh was to minimise an objective function of orthogonality deviations we elected to use the function `minimize` from the SciPy python library. This function offers various minimisation methods. We decided to use the Sequential Least Square Programming algorithm because it enables the user to provide the Jacobian of the objective function as well as constraints on the variables. The algorithm is also capable of approximating the Jacobian using the finite difference method to obtain the partial derivatives of the function.

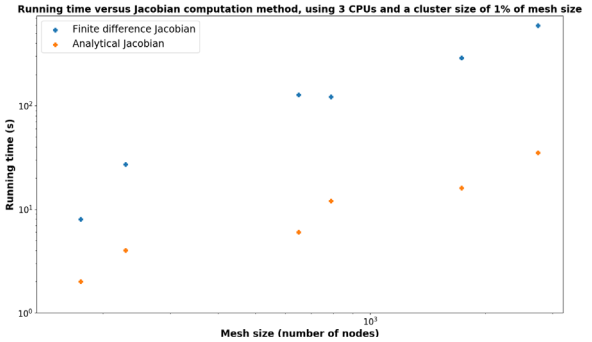

Figure 6: Running times vs. mesh size when computing the Jacobian either analytically (orange) or using the finite difference method (blue markers).

$$\frac{\partial f}{\partial x} \approx \lim_{h \to 0} \frac{f(x+h) - f(x-h)}{2h}$$

However, using this method greatly increases the number of calls to the objective function. For small problems this is not an issue but when the number of variable gets higher (i.e. when working with larger meshes) then the running time increases exponentially (Figure 6).

One way to address this problem is to provide the analytical expression of the Jacobian to the minimisation algorithm.

Let's take a variable, v_i of our system which is a free coordinate of the node n_i. The objective function f is the sum of the deviations, $1 - |\varepsilon|$. Note that all deviations are independent to each other. Then, the partial derivative of $f$with respect to $v_i$is:

$$\frac{\partial f}{\partial v_i} = \sum_j \frac{\partial f}{\partial 1 - |\varepsilon|_j} \frac{\partial 1 - |\varepsilon_j|}{\partial v_i}.$$

Where $\frac{\partial f}{\partial 1 - |\varepsilon|_j} = 1$, since all deviations have the same weight.

Then, we can write that:

$$\frac{\partial 1 - |\varepsilon_j|}{\partial v_i} = \frac{\partial 1 - |\varepsilon_j|}{\partial \varepsilon_j} \frac{\partial \varepsilon_j}{\partial v_i}.$$

Where, $\frac{\partial 1-|\varepsilon_j|}{\partial \varepsilon_j} = \begin{cases} -1, \varepsilon_j \geq 0 \\ 1, \varepsilon_j < 0 \end{cases}$.

Finally using the expression for ε we defined in Section 2.1.1, we can write that:

$$\frac{\partial f}{\partial v_i} = \sum_j -sign(\varepsilon_j) \frac{\partial (\mathbf{w} \cdot (\mathbf{u} \times \mathbf{v}))_j}{\partial v_i}.$$

The partial derivative of ε with respect to v_i depends on the degree of freedom, v, of the node n_i. We can highlight this dependency using the 3 following cases:

$$\begin{cases} case\ a.\ v = 3: n_i \begin{vmatrix} x_i = v_i \\ y_i \\ z_i \end{vmatrix} \xrightarrow{yields} \frac{\partial \varepsilon}{\partial v_i} = \frac{\partial \varepsilon}{\partial x_i} \\ case\ b.\ v = 2: n_i \begin{vmatrix} x_i = v_i \\ y_i(x_i, z_i) \\ z_i \end{vmatrix} \xrightarrow{yields} \frac{\partial \varepsilon}{\partial v_i} = \frac{\partial \varepsilon}{\partial x_i} + \frac{\partial y_i}{\partial x_i}\frac{\partial \varepsilon}{\partial y_i} \\ case\ c.\ v = 1: n_i \begin{vmatrix} x_i = v_i \\ y_i(x_i) \\ z_i(x_i) \end{vmatrix} \xrightarrow{yields} \frac{\partial \varepsilon}{\partial v_i} = \frac{\partial \varepsilon}{\partial x_i} + \frac{\partial y_i}{\partial x_i}\frac{\partial \varepsilon}{\partial y_i} + \frac{\partial z_i}{\partial x_i}\frac{\partial \varepsilon}{\partial z_i} \end{cases}$$

The vectors **w**,**u** and **v** depends on the coordinates of the 5 nodes forming the system of two neighbouring tetrahedra. Their partial derivatives with respect to each coordinates can then be computed. It is possible to calculate the partial derivatives of a coordinate with respect to another using the relations between the coordinates we determined in Section 2.2.2.

2.1.4 Constraints

The next issue we tackled was determining if constraints needed to be applied to the system to maintain a valid mesh during optimisation. Indeed if a node crosses an interface during the optimisation process then the blocks related to that node will either be flipped or crossed. In that case the mesh is corrupted and can no longer be used for flow simulations. One way to avoid corruption is to control the sign of the oriented volume of a block (*He et al.*, 1997).

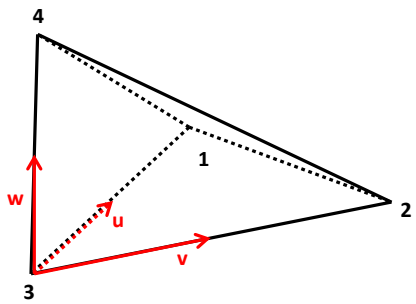

Figure 7: Tetrahedron (1234), in red are the vectors used to compute its oriented volume.

In the Figure 8, we show two theoretical solutions originating from the optimisation process of an initial mesh. In the first instance, optimisation results in node 6 being repositioned within the boundary of the model and, thus, there is no change to any signs of the triangles' oriented volumes. In the second instance, node 6 is moved through the interface [2,3]. In that case, we can see that the triangles (126),(156),(456),(463) are now overlapping the triangle (236) rendering the mesh unusable for simulation. At the same time, the sign of the oriented volume of those triangles changed. Therefore, it would have been possible to avoid

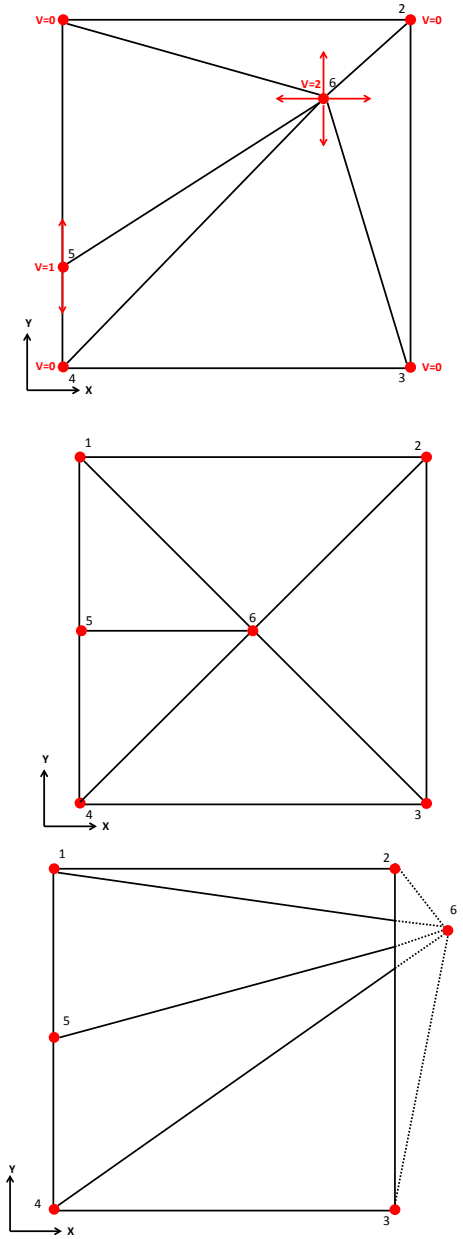

Figure 8: (top) Initial 2D mesh. (middle) Optimisation resulting in a mesh with improved orthogonality. (bottom) Optimisation resulting in a corrupted mesh.

this situation by checking at each iteration if the sign of the oriented volumes were still equal to the initial sign. This can easily be implemented in our algorithm since the minimising function we use allows for constraint inputs on the variables. Since the oriented volume V of a given tetrahedron (1234) (Figure 7) is given by the formula:

$$V = \frac{\mathbf{w} \cdot (\mathbf{u} \times \mathbf{v})}{6}$$

Then the constraints for a block (1234) would have the following form, V_0 being the initial oriented volume:

$$\begin{cases} \mathbf{w} \cdot (\mathbf{u} \times \mathbf{v}) < 0, & if\ V_0 < 0 \\ \mathbf{w} \cdot (\mathbf{u} \times \mathbf{v}) > 0, & if\ V_0 > 0 \end{cases}$$

However, in our investigations so far, node displacements have generally been small enough (compared to the size of the blocks) that volume sign changes are rare. Moreover, the implementation of those constraints is extremely expensive in terms of optimisation running time. Therefore it was decided to run the optimisation without applying any constraints on our system.

2.2 Optimisation strategies

The main issue that was encountered in the early stages of the algorithm's development was the dramatic increase of the running time with the increase of mesh size. Several methods were tested to address this issue and make the algorithm usable for reasonable grid sizes (up to 1 million blocks).

2.2.1 Mesh clustering

The first strategy was division of the mesh into clusters of nodes, which were optimised independently, holding the other nodes fixed. Using this strategy, we found that it is faster to optimize, say, 10 clusters of 100 nodes each than directly optimizing an entire 1000 node mesh. In our algorithm, different clustering strategies were tested.

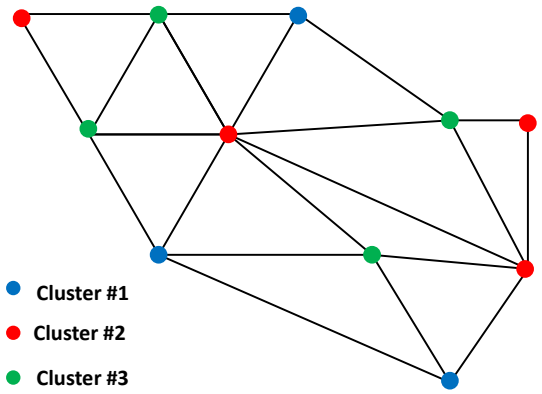

Figure 9: Example of random clustering of a 2D triangular mesh. The desired cluster size is 4.

The easiest, but least effective, strategy is random clustering (Figure 9). The user gives the algorithm a desired cluster size, s, and then the algorithm distributes the nodes randomly into n clusters of maximal size s.

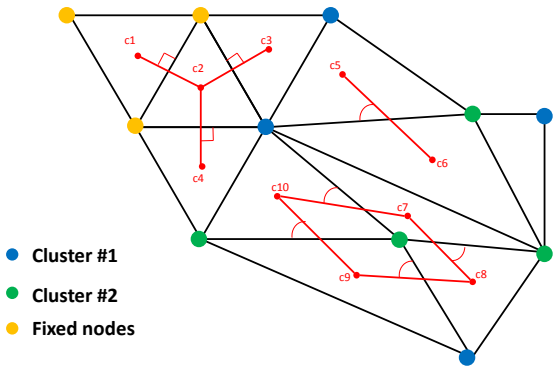

Figure 10: Example of clustering while targeting large deviations. The red dots are the triangles' centroids. The lowest and highest deviations are represented.

The second approach was to target nodes with large deviations from orthogonality. For example, in the system shown in Figure 10, some nodes (yellow) are associated with connections with small or no deviation. Thus, these nodes would be ignored (fixed) during optimisation. The remaining nodes are randomly distributed into n clusters of maximal size s. This algorithm does require a specified limit at which a deviation is considered sufficiently bad for optimisation to proceed. The advantage is that nodes that would be unlikely to be displaced during optimisation can be ignored, and computational effort can be concentrated on the bad regions of the mesh.

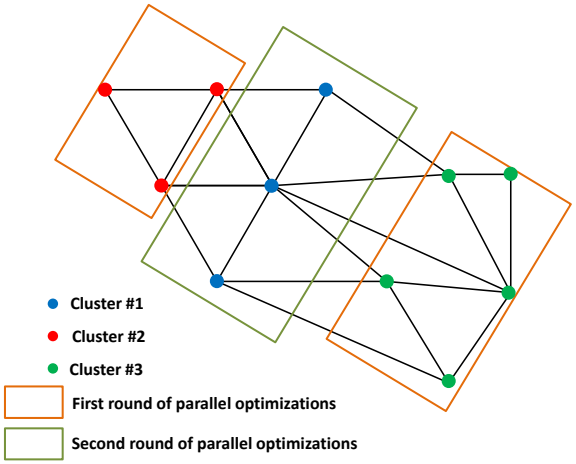

Figure 11: Example of a disjoint clustering of a 2D triangular mesh. The clusters 2 and 3 are disjoints.

2.2.2 Optimisation in parallel

We also investigated reductions in running time of the algorithm through implementation of multiprocessing. The basic approach is to optimize multiple clusters of the mesh in parallel. For this to work, clusters could no longer be randomly formed. For instance, if two clusters contain nodes that are connected to each other, then their optimisations are no longer independent. The key was to create groups of disjoint clusters (the size of the groups depending on the number of clusters the user wishes to optimize simultaneously). Two clusters are considered disjoint if they contain no neighbouring nodes.

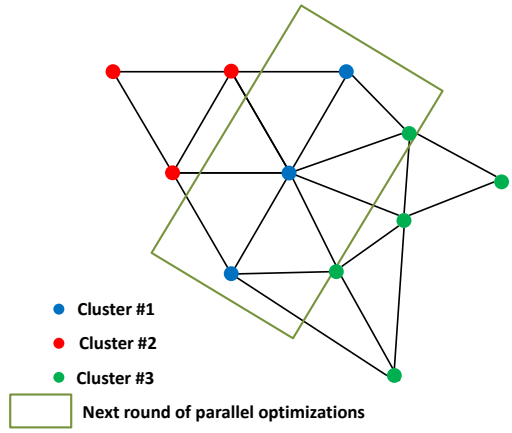

Figure 12: Result of the first round of optimisation. Clusters 2 and 3 are optimised without displacing the nodes from cluster 1.

Figure 11 shows the creation of three clusters such that clusters 2 and 3 are disjoints and can be optimized in parallel. Cluster 1 is not so it has to be optimized during a subsequent round of optimisation.

The result of the first round of optimisation would give the result shown in Figure 12. The nodes from cluster 1 are left unchanged. Clusters 2 and 3 were optimized in parallel successfully.

In the algorithm we build the clusters in two steps. First, we construct n clusters of size s using the `KMeans` function of the SciKit library. The K-means method aims at building clusters while minimising their inertia. In our case the inertia of a cluster is defined as the sum of the distance from a cluster's node to the cluster's centroid. Therefore the K-means method will try to divide our mesh into clusters of neighbouring nodes.

The second step of our clustering method is to distribute those clusters among m subgroups of size k where k is the number of CPUs the user whishes to use. All clusters in one subgroup must abide to our definition of disjoint.

In practice we use a combination of targeting high deviations nodes and disjoint clustering to be able to apply multiprocessing while reducing the number of variables in our system. That way we obtain the best possible running times.

3. RESULTS AND PERFORMANCE

In this section, we demonstrate and benchmark the optimisation algorithm on a range of small test problems. We then apply the method to optimise a production-scale problem: a million block grid of the Taupō Volcanic Zone (TVZ).

3.1 Test problems

3.1.1 Simple 3D problem

The first mesh is a simple cube with 15 nodes, one on each vertex, one on each faces, and one in the volume. In the initial state, all face nodes are placed away from the centre and the volume node is placed away from the centroid of the cube. After optimisation, we should expect the nodes on each face and in the volume to be located at their respective centres.

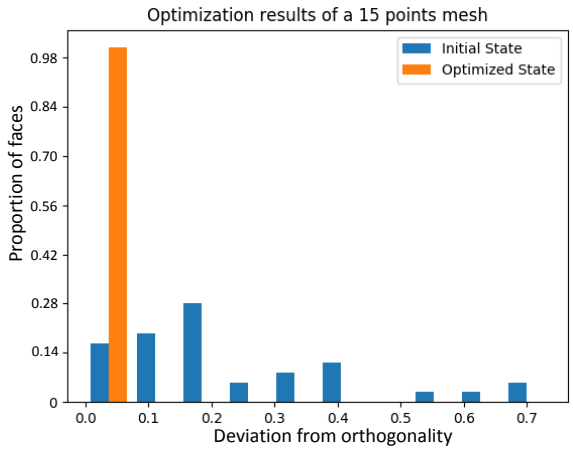

Figure 13: Histogram showing the distribution of deviations in the mesh. The initial state is in blue and the optimized configuration is in orange.

Figure 13 shows that the optimised mesh has minimal deviation from orthogonality. Furthermore, the boundaries have not been deformed, indicating that the degree of freedom reduction has been computed successfully (Figure 14).

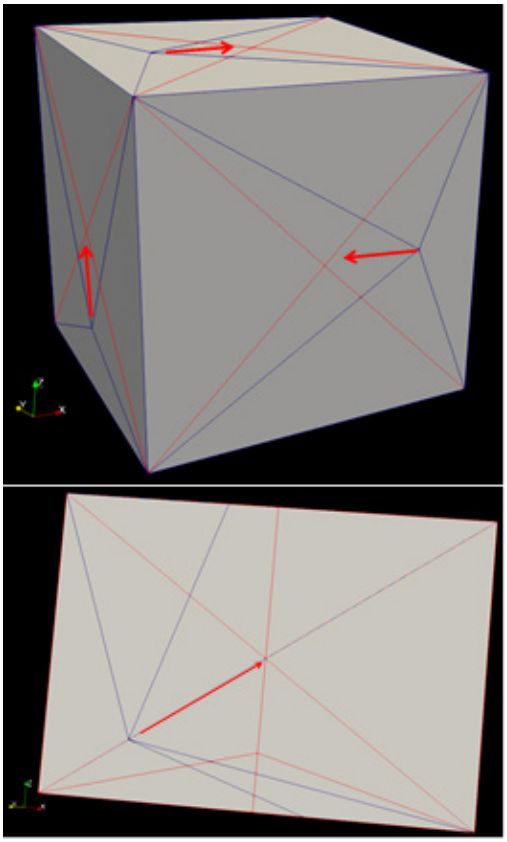

Figure 14: Two screenshots from Paraview. The blue and red wireframes represent respectively the initial and optimised states. The top one is a general view of the cube; the bottom one is a section by the plane –x-y=0.

3.1.2 Internal boundaries and complex topography

The second mesh is more complex (Figure 15). It has two internal horizontal boundaries that act as lithological contacts located respectively at a depth of 600m and 1800m. The model also has a complex surface that represents topography. In total the mesh comprises 650 nodes.

The mesh is optimized using the "disjoint/bad nodes" clustering technique. Three optimisation runs are performed for deviation thresholds of 0.4, 0.3 and 0.1. We can see in the Figure 16 that the number of faces showing perfect orthogonality has been increased by 20% after applying our optimisation process.

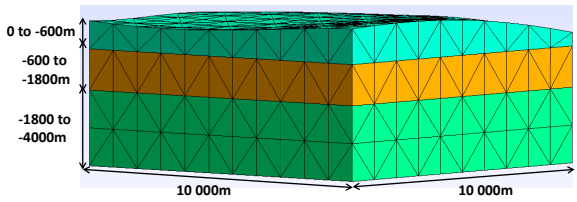

Figure 15: Overview of the mesh in GMSH.

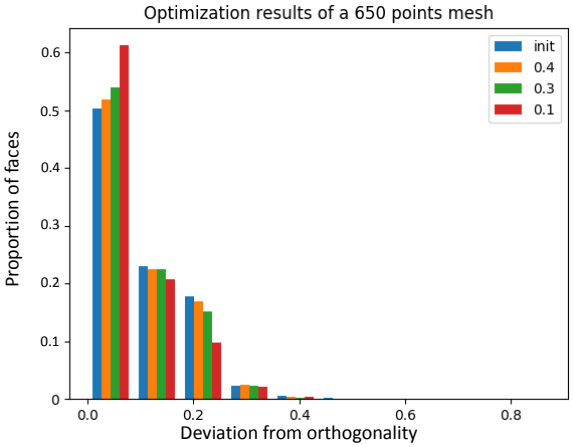

Figure 16: Histogram showing the evolution of the distribution of deviations in the mesh through successive optimization processes. The deviation limits used are, in order, 0.4, 0.3 and 0.1. In red is the most optimized state;

In Figure 17, we plot the nodes lying on the topographic surface, coloured blue or red representing respectively the initial and final state of the system. All nodes remained on the surface during optimisation process. Furthermore, nodes along the edge of the surface remain on the curve defining this edge. These results demonstrate optimisation constrained to complex surfaces as well as the intersection between planes and complex surfaces can be handled correctly by the algorithm.

3.1.3 Sequential optimisation

Using the clustering strategy, it is possible to perform successive optimisations on a mesh, targeting populations of nodes with progressively smaller deviations each. We have found that three successive runs with decreasing deviation threshold yields the best results.

It is notable that different results can be obtained when optimizing the same mesh using different clustering settings. This highlights the existence of local minima for our objective function.

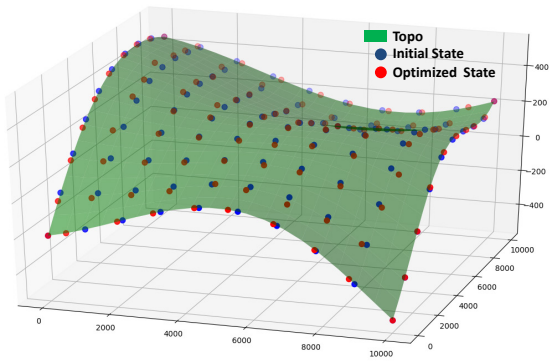

Figure 17: 3D plot showing in green the topographic surface of the mesh (approximated using 2D splines). The blue and red dots are the nodes of the mesh lying on this surface, respectively at the initial and final state.

After optimisation, there still remain a significant proportion of non-optimal connections. This may indicate an inherent limitation of tetrahedral elements to both capture geometric complexity while achieving element orthogonality. In future work, optimisation of unstructured hexahedral elements may be necessary to ensure sufficient degrees of freedom.

To improve our chances at finding a global minimum, we have explored the use of genetic algorithms. This works by generating N child meshes whose nodes are randomly perturbed from a common parent. Then optimisation proceeds for all children and only the best result is kept, to generate further children. However, in our tests, this strategy did not sufficiently improve the objective function to justify the additional computational expense. A more efficient way to improve quality of the minimum was the sequential optimisation discussed earlier.

3.2 Performance tests

A series of meshes with the same geometry but different numbers of nodes were created to assess performance and scaling. The geometry of the meshes is similar to the one described in Section 3.1.2 with the exception that the topographic surface was replaced by a horizontal plane at $z = 0$. The original meshes were constructed using GMSH.

3.2.1 Multiprocessing

In this test we ran some optimisation on a 6871 nodes mesh, using a cluster size of 1% of the mesh size and a number of processors ranging from 1 to 6.

In Figure 18, we plot the speed up between running on 1 processor and n processors. We can observe that doubling the number of processors does not lead to doubling the speed up, the multiprocessing seems less effective in our case. Because of the necessity of having disjoint clusters the number of iteration is not exactly divided by two when the number of processors is doubled. Indeed it is harder for the algorithm to produce groups of 4 disjoints s-sized clusters (needed to optimise on 4 CPUs) than groups of 2 (needed to optimise on 2 CPUs). This could be a reason behind the loss in speed up when using multiprocessing.

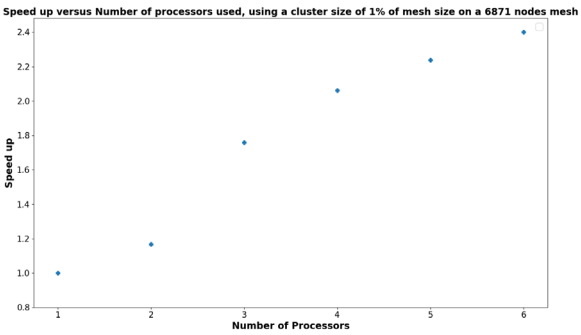

Figure 18: Plot showing the speed up with respect to the number of processors used. The speed up is defined as the ratio between the initial running time (with 1 CPU) and the 'new' running time (with n CPUs).

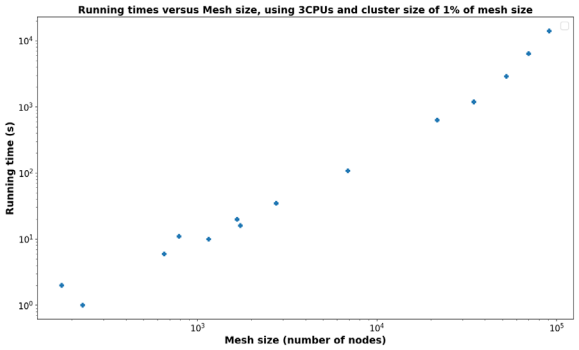

Figure 19: Running time vs. mesh size.

3.2.2 Mesh size

In this test, we have used multiprocessing on 3 CPU for each runs and the same clustering logic we used in 3.2.1. The mesh size ranges from a 100 to 100000 nodes. We can observe in Figure 19, that the running time is not linearly correlated to the mesh size; In fact the running time seems to be increasing quasi-exponentially with respect to the number of nodes in our system. We still obtain an optimisation of a 100000 nodes mesh in just under four hours which is encouraging since we wanted this algorithm to be applicable to full size grids.

3.2.3 Cluster size

Finally, we ran some optimisations on three models (respectively 650, 1733 and 6871 nodes) while modifying the cluster size used (Figure 20). We can observe that all three meshes present the same behaviour. Decreasing cluster size does not always result in a faster optimisation. Each grid present an optimal run time for a cluster size between 0.5 and 1.5% of the mesh size. Extremely small cluster sizes not yielding the best results arises because the amount of time spent launching all the parallel processes is much larger than the time solving the minimisation problem.

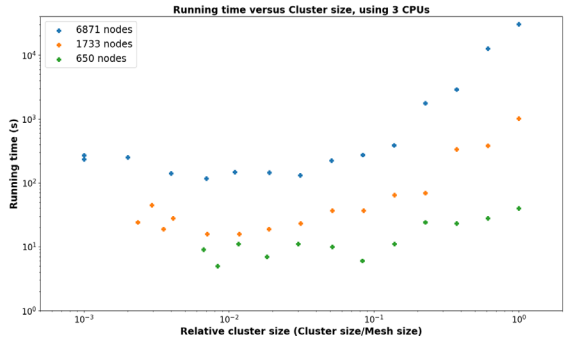

Figure 20: Running time vs. cluster size, for several mesh sizes of meshes (650, 1733, 6871 nodes).

3.3 Optimisation of a full size TVZ like mesh

This final section demonstrates application of the optimisation algorithm on an extremely large grid. We use a grid containing more than one million blocks (Figure 21). The 160km x 80km x 8km mesh covers the entire TVZ region from the Lake Taupō to the Bay of Plenty coast. We use Google Maps to build the real topographic surface of the region for our mesh. The basement contact is also approximated by 4 NE normal faults that delimit five blocks.

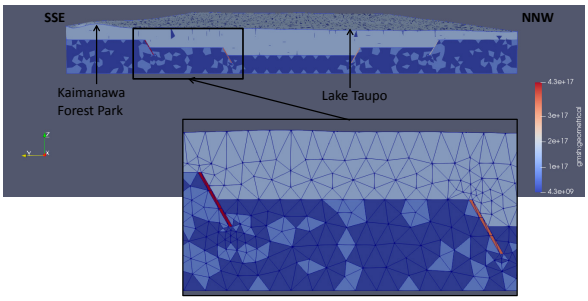

Figure 21: Overview of the TVZ mesh in Paraview. The coloring is used only to distinguish the different region of the mesh. The close up shows a better view of the structure around the faults.

In Figure 22, we can see that the proportion of faces presenting less than a 0.1 deviation from orthogonality has been increased by 45%. In that case we can see that the first two runs of optimisation did not lead to much improvement within our grid. However, if the optimisation was to be run directly for a deviation limit of 0.1 the quality gain would be diminished.

In terms of running time, the first two rounds of optimisation took 2.5 hours, while the final took 16h. Thus, it is reasonable to expect optimisation of production-sized meshes in less than 24 hours. Furthermore, mesh optimisation needs only be performed once, after which any number of flow simulations can be performed.

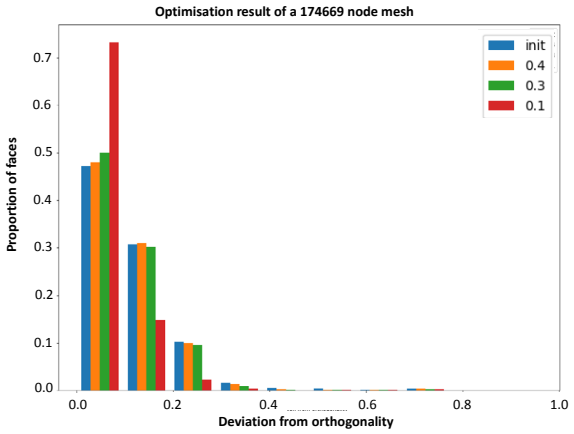

Figure 22: Histogram showing mesh deviations change after successive optimization passes. The deviation limits used are, in order, 0.4, 0.3 and 0.1. In red is the most optimized state; in blue the initial state.

4. CONCLUSIONS

We present a parallelised optimisation algorithm that can improve grid block orthogonality in a one million block mesh in less than 24 hours. Our algorithm respects and preserves various complex geometries, including plane boundaries, complex surfaces and boundary intersections. We have investigated how algorithm performance is improved for several optimisation parameters. In the future, we will improve the algorithm by introducing new types of geometric constraints, including intersection between two complex surfaces and arbitrarily oriented planes. We will also perform flow simulations to quantify the improved accuracy due to mesh optimisation.

REFERENCES

Croucher, A. E., and M. J. O'Sullivan. *Approaches to local grid refinement in TOUGH2 models*. Proceedings of the 35th New Zealand Geothermal Workshop, Rotorua, (2013).

Escobar, J. M., E. Rodrıguez, R. Montenegro, G. Montero, and J. M. González-Yuste. *Simultaneous Untangling and Smoothing of Tetrahedral Meshes*. Computer Methods in Applied Mechanics and Engineering 192, 2775–2787 (2003).

Geuzaine, C., and J.-F. Remacle. *Gmsh: A 3-D Finite Element Mesh Generator with Built-in Pre- and Post-Processing Facilities.* International Journal for Numerical Methods in Engineering 79, 1309–1331 (2009).

He, B., O. Ghattas, and J. F. Antaki. *Computational Strategies for Shape Optimization of Time–dependent Navier–Stokes Flows*. Engineering Design Research Center, TR-CMU-CML-97-102, Carnegie Mellon Univ, (1997).

Pain, C.C., A. P. Umpleby, C. R. E. de Oliveira, and A. J. H. Goddard. *Tetrahedral mesh optimisation and adaptivity for steady-state and transient finite element calculations*. Computer Methods in Applied Mechanics and Engineering 190, 3771-3796 (2001).