Engineering with Computers manuscript No.
(will be inserted by the editor)

Improving the quality of finite volume meshes through genetic optimisation

B. Fabritius · G. Tabor

Received: date / Accepted: date

Abstract Mesh quality issues can have a substantial impact on the solution process in Computational Fluid Dynamics (CFD), leading to poor quality solutions, hindering convergence and in some cases, causing the solution to diverge. In many areas of application, there is an interest in automated generation of finite volume meshes, where a meshing algorithm controlled by pre-specified parameters is applied to a pre-existing CAD geometry. In such cases the user is typically confronted with a large number of controllable parameters, and adjusting these takes time and perserverence. The process can however be regarded as a multi-input and possibly multi-objective optimisation process which can be optimised by application of Genetic Algorithm techniques. We have developed a GA optimisation code in the language Python, including an implementation of the NGSA-II multi-objective optimisation algorithm, and applied to control the mesh generation process using the `snappyHexMesh` automated mesher in OpenFOAM. We demonstrate the results on three selected cases, demonstrating significant improvement in mesh quality in all cases.

B. Fabritius
CFD+engineering
IB Fischer CFD+engineering GmbH
Lipowskystr. 12
81373 Munich
Germany
Tel.: +49 89 74 11 807 - 0
E-mail: bfa@gumby.de

G. Tabor
CEMPS, University of Exeter
Harrison Building, North Park Road
Exeter EX4 4QF. UK
E-mail: g.r.tabor@ex.ac.uk

Keywords Mesh generation · Mesh quality improvement · Mesh optimisation

1 Introduction

Mesh generation is commonly recognised as one of the main challenges in Computational Fluid Dynamics (CFD). Mesh quality issues can impact substantially on the accuracy of the eventual solution, even to the point where the solver diverges and no solution is generated; they can also significantly affect the level of computational work (e.g. number of iterations) necessary to reach the solution. Modern Finite Volume (FV) CFD codes tend to use arbitrary unstructured or polyhedral meshes, allowing for a wide variety of cell shapes to accommodate complex geometries. This also allows for a wide variety of mesh problems; non-orthogonality, face skewness etc, and whilst modern solution algorithms can typically correct for mild levels of mesh problems, this is at the cost of additional numerical error. Pathological levels of mesh problems can lead to algorithm divergence. The acceptable level of mesh quality also varies according to the details of modelling being used, for example the turbulence modelling in Large Eddy Simulation (LES) ties in very closely with aspects of the mesh such as cell size, thus requiring much higher levels of mesh quality than for RANS methods.[28] Note that our discussion revolves around issues relating to mesh generation for FV CFD, which is our area of familiarity. Similar issues undoubtably arise for Finite Element methods and other applications of these techniques.

Given its importance, significant effort has been put into developing metrics to quantify mesh quality, as well as into methodologies to improve mesh quality. At the

most basic level, a mesh metric represents an *a priori* assessment of the mesh, which can be used as a target for mesh development or as a test to assess suitability for progress to the stage of simulation. Numerous individual metrics have been proposed, particularly with reference to FE meshes (eg. [35,34,5], see for example [9] for a review). Meanwhile, Knupp [20–22] demonstrated an algebraic framework to derive quality metrics from the Jacobian matrix for the elements, which contains information on basic element properties such as size, orientation and shape. His work also identifies the occurrence of different types of metric, and recognises that there may be several possible and interchangable metrics for a metric of a particular type, such as element shape[23]. In the FV method, commercial and open source practice has tended to utilise specific metrics such as non-orthogonality and cell skewness [38]. Although practitioners utilise and rely on mesh metrics as a key indicator of the suitability of a mesh for computation, a direct link between mesh metrics and numerical aspects of the subsequent calculation such as truncation error is difficult to establish [17,19,7]. *A posteriori* evaluation of mesh quality can also be important – in CFD, most notably the checking of near wall y^+ values to assess the validity of wall modelling – however this is obviously not possible until after a simulation has actually been run.

Mesh quality metrics also provide input to various techniques for mesh quality improvement methodologies. Two main approaches have been investigated to improve mesh quality; global approaches involving smoothing, and local approaches involving reworking groups of cells [11]. The simplest smoothing algorithms are based on Laplacian smoothing [8]; however this heuristic approach can be unstable and sometimes inverts or otherwise degrades local elements. Optimisation-based smoothing is based on local gradients of element quality using algebraic minimisation approaches such as Conjugate Gradient methods [12], although these can be computationally expensive. Local mesh improvement methods are topological in nature, involving deleting elements and replacing with alternative arrangements of elements, edge and face removal [33].

Over the years, numerous methods for mesh generation have been developed and are available either open-source or in commercial packages, either associated with particular CFD codes (e.g. Gambit, ANSYS Mesher) or independently (such as Pointwise, CENTAUR or Harpoon). Practically we can distinguish between meshers which provide CAD capabilities integrating geometric construction with meshing, and those intended as pure meshers, where the geometry of interest is provided in the form of a CAD file. The degree of control provided to the user also varies, with some meshers providing total control down to individual sub-blocks of the mesh, whilst others try to provide an automated pipeline for the process. Exactly what meshing strategy to adopt will depend on the exact problem being investigated, and it is probable that no one universal meshing solution is possible. However in many areas of CFD there is an interest in automated meshing of pre-existing CAD geometries, for instance in the automotive industry, where CAD files of new vehicle designs are available from the design process. Meshers can generate tetrahedral, hexahedral or mixed meshes; or recently there has been an increased interest in polyhedral meshes. Whilst tetrahedral meshes are technically easier to generate automatically from Delaunay methods, hexahedral cells provide some important advantages in solution accuracy [36]. Most hexahedral mesh generators can be classified as either geometry-first or mesh-first methods [32]; in the first, the CAD representation of the boundary surfaces is used to grow elements or cells into the domain, whilst in the second a space-filling mesh is constructed and then modified to capture the geometric features of the CAD model. However the meshing is achieved however, it is a complex process controlled by significant numbers of user-controllable parameters.

A typical example of an automated mesher used for such problems is `snappyHexMesh`, which is part of the open source OpenFOAM CFD package [39]. To use `snappyHexMesh` the user provides an STL file of the geometry and a base mesh (typically a simple hexahedral block mesh). `snappyHexMesh` then operates a 3 stage meshing process of castillation, snapping and boundary layer refinement. In the first step (castillation), cells are identified which are intersected by edges of the surface geometry; these cells are then refined by repeated cell splitting, with maximum and minimum levels of refinement being a definable parameter, and further surface refinement also being controllable. After this refinement process, all cells which lie "outside" the desired geometric domain are deleted from the mesh (for a car this would be cells on the interior of the STL geometry, of course). In the second, snapping step, vertices on the edge of the domain are "snapped" to the STL surface, using an iterative process of mesh movement, cell refinement and face merging, again controlled by user defined parameters such as number of iterations and specific mesh quality constraints. In a final and optional step, cell layers can be added to the surface to move the mesh away from the boundary to specifically refine a boundary layer. The whole process is robust and automated, but is controlled by a large number of

user specified parameters provided in advance as an input file. As with any meshing process, the user typically has to experiment with the different settings in order to optimise the mesh. Mesh quality may ultimately be judged by the success of the resulting CFD run, but as a proxy various mesh quality indicators such as skewness and non-orthogonality can more easilly be evaluated.

The significant point here is that this process may be regarded as a multi-parameter and probably multi-objective optimisation problem. Such optimisation problems abound in Engineering, and numerous techniques to solve them have been developed. Optimisation problems involving multiple input parameters and a complex response surface (which is plausibly the case here) have often been approached using Genetic or Evolutionary Algorithm techniques (the two terms are almost interchangeable). In these approaches, the set of individual parameters necessary to define the solution are regarded as an individual within a randomly chosen population of N individuals. The "fitness" of each individual is then evaluated algorithmically, and a new generation created from the "most fit" individuals through a combination of genetic recombination and mutation. Over M generations this process will explore the parameter space and find the optimum solution to the problem. Although the process of automated meshing may be regarded as a suitable multi-parameter optimisation process, to our knowledge the process has not actually been approached in this way, and this paper represents a first attempt at doing this.

The structure of the paper is as follows. In the next section we provide a more detailed description of the GA process and our implementation of this in the `pyFoam` code, which is a Python wrapper around OpenFOAM providing run-time control of the parametric input into the code (essentially a scripting facility for the OpenFOAM code itself). In section 4 we present the results of applying this to a number of simple meshing test cases, and in section 5 we analyse our results and experiences with this novel approach.

2 GA Optimisation

Genetic Algorithms are based on the principles of natural selection and descent with modification [14] which operate on biological organisms and which have generated the diversity of species seen in nature. A set of parameters in a GA will generally be coded as a string of finite length, most commonly a binary string. Each of these strings (also *chromosome* or *genotype*) represents one possible solution to the optimisation problem. At the outset a population of these individuals is initialised at random, representing a diverse set of possible solutions. The population then undergoes simulated evolution. Individuals are selected for reproduction depending on their fitness value. This selection process is stochastically controlled, assigning fitter individuals a higher probability to get chosen. From those individuals (*parents*) selected in this manner, offspring (*children*) are generated by applying crossover and mutation operators. The crossover operator uses two parents and combines elements from one parent with elements from the other, creating a new individual that now contains information from both its ancestors. An example of single point crossover between two chromosomes (binary strings) a and b of length n+1:

$$a = \langle a_n\, a_{n-1}\, \ldots\, a_1\, a_0 \rangle$$
$$b = \langle b_n\, b_{n-1}\, \ldots\, b_1\, b_0 \rangle$$

with a randomly selected crossover point $X\,\epsilon\,[0, n-2]$, creating children:

$$a' = \langle a_n\, a_{n-1}\, \ldots\, a_{X+1}\, b_X\, b_{X-1} \ldots\, b_1\, b_0 \rangle$$
$$b' = \langle b_n\, b_{n-1}\, \ldots\, b_{X+1}\, a_X\, a_{X-1} \ldots\, a_1\, a_0 \rangle$$

Mutation is in most cases implemented as bitwise mutation where the value of a single bit in a chromosome is inverted. The probability of mutation or crossover occuring is controlled by external variables P_M and P_C respectively. Other parameters that influence the performance of the GA are the population size S and the number of generations G. In the optimisation problem at hand, the multiple real values are bit-string encoded and the fitness objectives are measurable properties of the flow.

Since the problem variables are real values and their chromosomal representation is a binary string, a mapping between the two has to be defined. For a single coefficient $c\,\epsilon\,[c_{lo}, c_{hi}]$ the length of the bitfield has to be determined by taking into account the desired resolution Δ_c of the interval. The number of bits required is now

$$n = \left\lceil \log_2 \left(\frac{c_{hi} - c_{lo}}{\Delta_c} + 1 \right) - 1 \right\rceil \tag{1}$$

Translation from binary to decimal values can now easily be done as follows:

$$\langle b_n\, b_{n-1}\, \ldots\, b_1\, b_0 \rangle_2 = \left(\sum_{i=0}^{n} b_i \cdot 2^i \right)_{10} = c' \tag{2}$$

$$c = c_{lo} + c' \cdot \frac{c_{hi} - c_{lo}}{2^{n+1} - 1} \tag{3}$$

Compared to conventional optimisation methods, GA's exhibit several important benefits when used to optimise multi-parameter systems. In particular, GA's are very thorough in exploring the parameter space of the problem, and will climb many peaks simultaneously

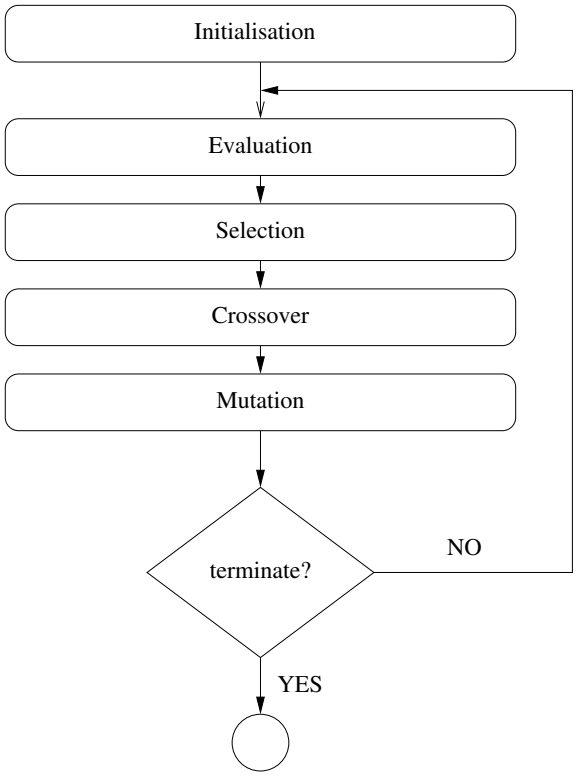

Fig. 1 Schematic of the workflow of a typical GA

during the evolution process. This reduces the probability to concentrate on the wrong peak representing a local optimum, as common gradient based methods would do. Figure 1 depicts the sequence of operations in a typical GA. Two opposed strategies are at work here: Exploitation of a single solution versus exploration of the solution space. Classical gradient based methods concentrate on the former, while sole usage of the latter would correspond to a random search. GAs manage to reach a surprisingly good balance between those two extremes [30].

2.1 Multi-objective Optimisation

Complex optimisation problems often seek to find optimal solutions with respect to multiple, often concurrent, objectives. Many multi-objective evolutionary algorithms (MOEAs) have been developed in the last few decades [37, 41, 10]. Since it is generally the case that a problem has no single solution that is optimal with respect to all objectives simultaneously, normally a number of equally optimal solutions are generated each of which is optimal for a specific set of weightings between the objectives. The set of these non-dominating solutions is described as the Pareto-optimal front. The algorithm that is used in this study is a fast elitist non-dominated sorting genetic algorithm (NSGA), that was originally introduced by Srinivas and Deb [37] and improved by Deb et. al. [6]. The second generation version NSGA-II removed some of the criticised flaws in the original algorithm and is able to capture high order Pareto surfaces. Elitism speeds up the convergence of the GA and prevents the loss of the best solutions. The sorting procedure orders solutions by the level of dominance over concurring solutions. That way the most dominant individuals are considered to be the fitter ones and therefore have a higher chance to contribute to the next generation. The algorithm has been successfully used in engineering optimisation problems [4, 18].

2.2 Implementation and code design

Available for OpenFOAM is a toolset called pyFoam [1] written in the object-oriented language Python. It offers applications to read, modify and run OpenFOAM cases as well as analyse the results. Inspired by this, the framework for the evolutionary computation capabilities was developed in Python. That way the invoking and manipulation functions provided by pyFoam can be used and execution of the program can easily be controlled by using scripts. The overall design of the GA software was based on the guidelines by Gagné and Parizeau on how to write generic EC (Evolutionary Computation) software tools [13]. The aim is that operators, such as the crossover or selection operator, should be interchangeable regardless of the objects they are applied to. In addition the underlying representation of a solution should not affect the way the GA works. The user can choose at run-time between a given set of predefined operators or can add new operators to meet specific needs. This is usually the case for the fitness evaluation which is a problem dependent function. Reusability and independence of the optimisation problem on top are key features of the selection and crossover mechanisms. Commonly used realisations of these are therefore included in the developed framework, but can be altered or new ones implemented. Equally flexible is the selection of the coding algorithm that encodes and decodes the chromosome as described in section 2. In the developed software package control parameters can be set using external configuration files. For every variable that is subject to the evolution process the user can define lower and upper bounds as well as the desired precision. This allows running different test cases with different initial setups without altering the code. The only element that has to be adapted for

[1] http://openfoamwiki.net/index.php/Contrib_PyFoam

each case is the fitness evaluation function since it is problem dependent.

One of the most important requirements in the development of a GA software in the context of CFD is the capability to parallelise the code to allow for faster computation spread over several processing units. This parallelisation is implemented at the population level, with individual cases in the population being assigned to individual processing units in a master-slave arrangement. A commonly used library to realise the inter-processor communication is the MPI (Message Passing Interface) standard [29]. The Python implementation `mpi4py` is used in the current project. While it is not a full realisation of the MPI standard, it provides all the required functions for the purpose of this research.

3 Optimisation Objectives

When generating a mesh with `snappyHexMesh` (or indeed any other mesh generator) the user is trying to balance particular constraints, making this a classic multi-objective optimisation problem. In this case we have three main objectives for the mesh optimisation :

1. Total number of mesh cells
2. Mesh quality
3. Accuracy of snapping to STL surface

Objectives 1 and 2 are probably significant for any meshing algorithm. As one can imagine, with an unlimited number of cells even the most complex surfaces could be captured accurately. However we are unlikely ever to be in a position in which unlimited cell counts are a realistic possibility, so we might well be interested in the tradeoff between cell count and other aspects of the mesh. In fact the structure of `snappyHexMesh` is such that the cell count is heavily dependent on the resolution of the base hexahedral mesh, and so the mesh sizes varied little if at all between the different individuals for certain settings (such as for the snapping stage), so for this work we have chosen to concentrate on the tradeoff between 2 and 3.

Measurement of objective 2 can be accomplished using a variety of individual metrics. For this preliminary work we chose to construct a simple fixed measure of mesh quality based on a fixed combination of skewness, non-orthogonality and minimum cell size, as described below. Objective 3 is a feature of `snappyHexMesh` and other CAD-based automatic meshers which do not necessarily produce perfect geometric accuracy, but which trade a degree of geometric inaccuracy for greater robustness.

Because of the strucure of the algorithm, the fitness evaluation operator had to be defined such that it would try to minimise the value of each objective function. Multi-objective optimisation with mixed objective value interpretation, where for example one objective value has to minimised while another one has to be maximised, is not possible in the current implementation of the NSGA-II algorithm. However this is not a significant restriction as algebraic manipulation of the form of the fitness function can always be used to ensure that all the fitness functions are being minimised.

3.1 Mesh Quality and Geometric Accuracy

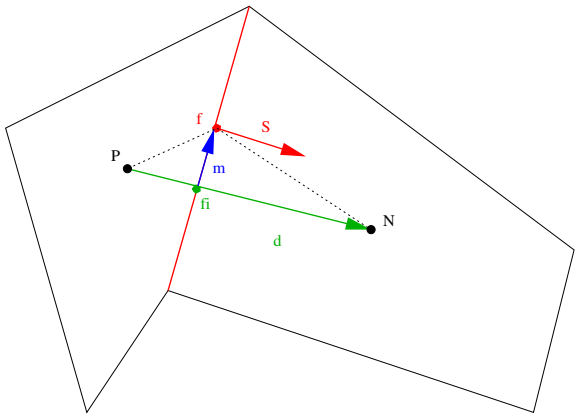

Fig. 2 Determining skewness on a face

Two very specific measures of mesh quality are commonly used in CFD : skewness and non-orthogonality, and these deserve further discussion. The skewness error is a numerical diffusion-type error emerging from the finite volume discretisation [16]. The standard form of the transport equation for a scalar property ϕ is

$$\frac{\partial \rho\phi}{\partial t} + \nabla \bullet (\rho \mathbf{U}\phi) - \nabla \bullet (\rho \Gamma_\phi \nabla\phi) = S_\phi(\phi) \tag{4}$$

where ρ is the density of the fluid (which is constant for incompressible flow), $\mathbf{U}$ is the velocity vector, Γ is the diffusivity, and S_ϕ an arbitrary source term. In the finite volume method this equation is integrated over the volume of each cell and Gauss' theorem used to convert integrals of the spatial derivatives (particularly the convection term) into surface integrals over the boundary of the control volume, i.e. flux terms. The convection term for example is then handled as:

$$\int_{V_P} \nabla \bullet (\rho \mathbf{U}\phi) dV = \sum_f F \cdot \phi_f \tag{5}$$

where F represents the mass flux through the face f:

$$F = \mathbf{S} \cdot (\rho \mathbf{U})_f \tag{6}$$

Figure 2 shows a typical situation causing the skewness error in two adjacent cells P and N connected by a face with centre f, and face area vector **S**. The value of the face integral requires the variable value at point f.

$$\int_f d\mathbf{S}\phi = \mathbf{S}\phi_f \tag{7}$$

In the finite volume implementation the value ϕ_f is often calculated from a linear interpolation between points P and N. This yields the value of ϕ at the point f_i, which is not necessarily equal to f. The error E_S of the convection term in Eqn. 5 is estimated as:

$$E_S = \sum_f \mathbf{S} \cdot [(\rho\mathbf{U})_f \mathbf{m} \cdot (\nabla\phi)_f] \,. \tag{8}$$

On meshes of reasonable quality, $|\mathbf{m}|$ should be much smaller than $|\mathbf{d}|$, but when this condition is no longer met, as in very skewed meshes, the influence of **m** in Eqn. (8) becomes more significant. The accuracy will suffer when the mesh is highly skewed. This results mainly from the way in which the face–centered pressure gradients are computed using cell–centered pressure values. Usually a second order central-difference approximation is used and the accuracy might drop to first order for very high skewness [40]. In other words, skewness is a measure of how far off the face center between two adjacent cells does the connecting vector d of the two cell centers intersect the face.

A similar measure is non–orthogonality, which describes the angle between the vector d and the face normal S. In a good quality mesh, these two vectors should be parallel, i.e. d is orthogonal to the face. Since the diffusive terms in the finite volume discretisation of the Navier-Stokes-Equations in OpenFOAM use the face normal vector to calculate fluxes between cells, it is desirable to minimise non–orthogonality. For both measures there is is the question of which is likely to be more significant; the average value or the maximum value. The average non-orthogonality (i.e. the average of the non-orthogonality values for all faces in the mesh) is a significant index of overall mesh quality, whilst the max non-orthogonality (the value of the most non-orthogonal face in the mesh) is also significant as just one awkward face is sometimes sufficient to destroy convergence.

The last objective considered is the accuracy of the snapping algorithm, which refers to how close the resulting mesh coincides with the desired surface. To quantify this criterion the distance between external mesh faces and any of the STL surfaces is measured and the sum of all these distances represents the fitness value. This is of course limited to the cells that are near an STL surface in their normal direction. To this end an application was developed within the OpenFOAM framework that loops over all exterior faces and calculates the distance to the nearest STL patch. Exterior faces in this sense are those that lie on the surface of the domain.

3.2 Fitness evaluation

To get a measure of the achieved mesh quality, two sources of information were used. The tool `snappyHexMesh` outputs mesh quality information as it proceeds, information which can be logged and scanned to extract the necessary data; additionally the OpenFOAM library includes a utility `checkMesh` which can be run to provide additional data.

Table 1 Value constraints for the objective variables in mesh generation optimisation. Accuracy value of 1 signifies an integer variable.

Parameter	min value	max value	accuracy
resolveFeatureAngle	30	80	1
nSmoothPatch	5	50	1
nRelaxIter	3	15	1
nFeatureSnapIter	10	30	1
maxNonOrtho	30	80	1
maxSkewness	0.5	1	0.01
minVolRatio	0.01	0.1	0.01

The evaluation of a solution's fitness now depends on how a quality value needs to be interpreted. In case of cell volume, for example, good fitness might mean that the minimum volume is not lower than a given value, while the average cell volume lies within a certain range of values. In our case we opted to track three distinct indices of mesh quality; maximum skewness, maximum non-orthogonality and minimum cell size. All these individual fitness measures then have to be accumulated into one number that represents the mesh quality, i.e. the second objective in the multi–objective optimisation. Agreement with the quality constraints of each parameter calculated by the `checkMesh` utility was not realised as a different objective function for each value. Instead the grades of agreement (or disagreement) were combined into a single fitness value. To account for different orders of magnitude in the actual calculated numbers, the fractional biased error was used to limit the fitness value for each entry to a certain range. Equation 9 shows how such a value is computed per quality constraint. The symbol ξ_O represents the observed value obtained by running `checkMesh` and ξ_P is the prescribed value set in an optimisation objective.

$$FB(I) = 2 \times \frac{\xi_O - \xi_P}{\xi_O + \xi_P} \tag{9}$$

The advantage of the fractional bias is that it limits the values to the interval $[-2, 2]$. The sign just represents the direction of disagreement and a value of zero means a total agreement of prescription and observation. If the direction is not of interest, the bias can be squared to assure positive numbers only. The fractional bias is a useful method to compare real data with predicted data, because it equally weights positive and negative bias estimates.

3.3 Input parameters

`snappyHexMesh` requires a substantial number of input parameters. Experimentation reduced this to a set of 7 whose definition is provided below :

- resolveFeatureAngle Maximum level of refinement applied to cells that intersect with edges at angles exceeding this value.
- nSmoothPatch Number of patch smoothing operations before a corresponding point is searched on the target surface. Smooth patches are more likely to be parallel to the target surface, making it more probable to find a matching point.
- nRelaxIter Number of iterations to relax the mesh after moving points. When points are snapped to the target, the displacement propagates through the underlying layers of points that are not on the surface. By relaxing this propagation, a smoother displacement can be achieved.
- nFeatureSnapIter The total number of iterations tried to snap points to the target. If insufficient quality is reached after `nFeatureSnapIter` iterations, the snapping is cancelled and the last state is recovered.
- maxNonOrtho Non-orthogonality measures the angle between two faces of the same cell. In a grid with only rectangular cells the value would be zero. Any deviation from this counts as non-orthogonal. High values mean there are very low angles that usually occur in a prism layer.
- maxSkewness Skewness is the ratio between the largest and the smallest face angles in a cell. A value of 0 is the perfect cell and 1 is the worst. For tetrahedral cells the value should not be greater than 0.95 to ensure accuracy of the calculation. Within the dictionary different quality constraints can be assigned to boundary cells and internal cells. Because in a simple geometry the cells on the boundaries are more likely to be affected by skewness problems, only this value was part of the optimisation.
- minVolRatio The ratio in cell volume between adjacent cells should not be too large. A large aspect ratio leads to interpolation errors of unacceptable magnitude.

All these parameters were used as decision variables in the optimisation and Table 1 lists these variables and their value constraints used in the optimisation.

The parameters that were subject to the optimisation can be split into two groups: cell quality and snapping accuracy. For the first group of cell quality the `snappyHexMesh` sub-dictionary `meshQualityControls` contains the values that were of interest here. From experience using snappyHexMesh and because the bearing test case was a rather simple geometry without any sharp angles, the constraints listed in Table 2 were considered.

Table 2 Mesh quality settings in snappyHexMesh

Parameter	Min	Max
maxBoundarySkewness	1.1	2.4
maxNonOrtho	40	80
minVolRatio	0.01	0.1

As mentioned above, the mesh creation optimises towards multiple objectives. Running `snappyHexMesh` on a case with a target size of about 250,000 cells is computationally very expensive in terms of time and memory. To save disk space the workflow was slightly modified so that only the Pareto optimal individuals of each generation are physically stored, while the others are deleted after their evaluation and before the evolution proceeds to the next generation. Since the coefficients of each individual in every generation are logged anyway, this could be further improved by not storing any meshes, but reconstructing solutions on demand using the values stored in the log file.

4 Meshing test cases

Three test cases were investigated selected to illustrate specific problems in meshing. The first two were a Bearing problem (section 4.1) and a simple Packed Bed, which illustrate specific issues such as the handling of contact points between spheres (for the Packed Bed). The third case under analysis was a fairly simple real-world example of meshing, that of mesh construction for the Ahmed body, a commonly-used case in vehicle aerodynamics, and one where the solution is known to be sensitive to the mesh details.

4.1 Bearing

This simple test case is comprised of two pipes of different diameter that are connected by a planar disk. The inside of this assembly is to be meshed using snappyHexMesh. Figure 3 shows the three parts and how they are arranged in the structure. A detailed view of the connector disk (Figure 4) reveals a chamfer at the inlet to the smaller pipe. From a meshing standpoint this geometry is relatively easy to describe, but contains a few difficulties that can have severe impacts on the mesh quality. For example where the base of the bigger pipe meets the connector disk, a combination of straight and curved edges in one cell is required. The curvature should be captured by all cells along the joint and should be reasonably smooth to represent good cell quality. On the other hand around the chamfer different angles between faces have to be created to fully capture the geometry change in this area. While being a rather simple geometry, it offers enough difficulties for an automatic mesh generator to be of interest here.

The initial rectangular mesh outlined on the left of Figure 3 was created using OpenFOAM's blockMesh utility. It consists of 1372 cells, or 28 by 28 by 14 in three dimensions. The axial direction of the tubes is the z-axis. The target mesh size was limited to 200,000 cells in the snappyHexDict with refinement along the tube walls and around the diameter change at the position of the connector. The optimisation was run using 30 individuals per generation and terminated after 20 generations.

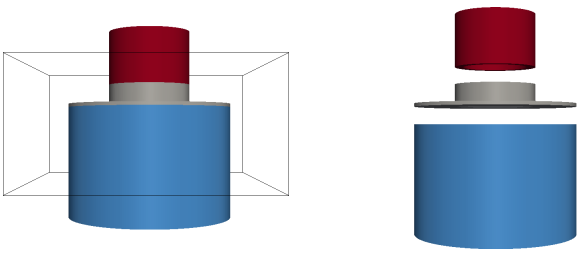

Fig. 3 Geometry of the snappyHexMesh bearing test case. The black box on the left is the outline of the original mesh that will be snapped to the inside of the geometry. The right image shows the three parts that make up the bearing.

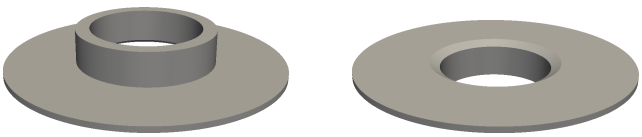

Fig. 4 Detailed view of the connector disk's top and bottom side showing the chamfered edges.

The mesh quality objective index as stated above is a composite of several parameters, and as such is difficult to interpret. Instead it is informative to look at how individual quality indicators have changed as part of the optimisation process. Figure 5 shows graphs of various mesh and geometric quality indicators evaluated directly from `checkMesh`. In each case, the squares represent the first generation values, the triangles the final generation values, and the filled triangles the Pareto set for the final generation values. Note that the quantities *max surface displacement* and *max non-orthogonality* are outputs which are parts of the overall mesh quality metric (and are raw, rather than being scaled through the fractional biassed error method) and are different from the parameters`maxNonOrtho` and `maxSkewness`, which are input parameters to `snappyHexMesh`. Graphs 5.a. and b. examine the relationship between the maximum values of surface displacement and non-orthogonality and the average values, showing a reasonable level of correlation for the displacement but very little correlation for the non-orthogonality. The other two figures examine the correlation between the average surface displacement, which is one of the optimisation objectives, and max non-orthogonality and max skewness, which factor into the mesh quality metric. As might be expected, a few individuals in the initial population are geometrically faithful, but the majority deviate quite significantly, as indicated by the surface displacement parameter. For this case this has significantly improved by the last generation, although there seems to have been a compensating deterioration in the max non-orthogonality value.

Table 3 shows the final parameter settings in the `snappyHexMeshDict`. The bad quality example was randomly selected from the dominated population of the last generation and the good example was taken from the Pareto front. The results of the mesh optimisation are visualised in Figure 6. These images were chosen to highlight those parts of the mesh that are clearly of different quality. The total number of cells was almost identical in both meshes, with 60,452 in the bad example versus 62,195 in the optimal case. Comparing the parameter settings in all individuals of the Pareto front showed that for the `minVolRatio` the value was always 0.01 or very close to it. It can be assumed that this is actually the optimal setting for this parameter. Table 3 lists the meshing parameters for these two example meshes as well as the value ranges found in the Pareto front of the final generation. A significant improvement in the average mesh non-orthogonality is evident between the bad and good examples.

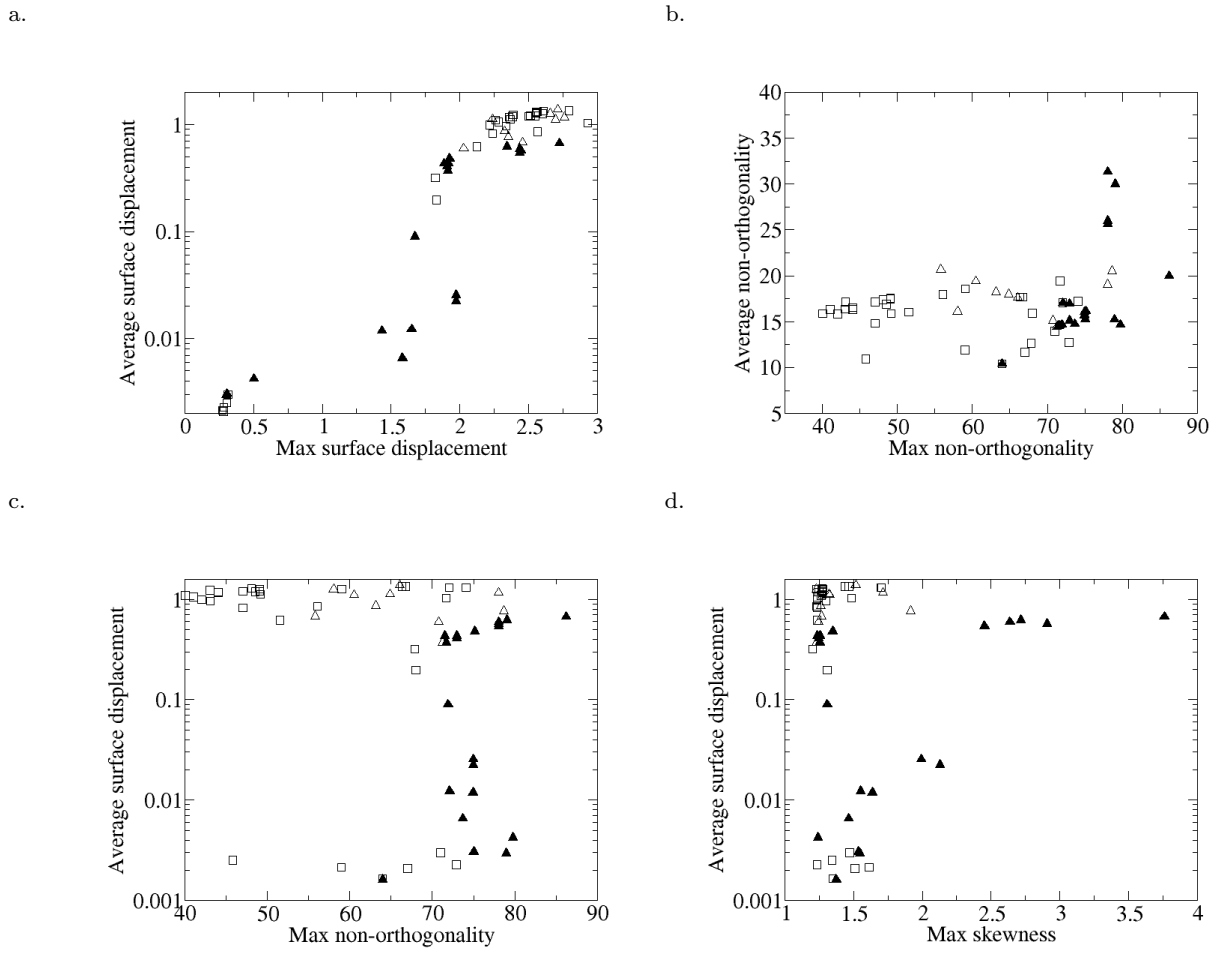

Fig. 5 Graphs of a. average surface displacement vs. max surface displacement, b. average mesh non-orthogonality vs. max non-orthogonality, c. average surface displacement vs. max non-orthogonality, d. av. displacement vs. max skewness for the bearing case. In each case; squares represent the initial population, triangles the final population and filled triangles the Pareto set.

Table 3 Parameter settings for `snappyHexMesh` for the bearing test case reffering to the two examples depicted above and value ranges in the Pareto front.

Parameter	bad example	good example	Pareto range
maxNonOrtho	70	72	60–79
maxSkewness	6.0	10.7	8.0–12.3
minVolRatio	0.07	0.01	0.01–0.03
Average non-orthogonality	19.24	10.40	

4.2 Packed Bed

In simulations of granular media on a macroscopic scale, material particles are often modelled in an idealised manner as spheres. These spheres are then stacked or packed together as a regular or irregular lattice leaving small spaces between individual particles as the flow domain. In the idealised case the spheres will touch tangentially at a single point and the cells around this connection need to be wedge shaped, resulting in high skewness and non-orthogonality [3]. Finding a good compromise between cell shape and mesh quality is vital for a reliable numerical treatment of the flow through a packed bed, and in fact it can prove necessary to relax geometric accuracy in order to produce a usable mesh [2]. Thus, automatically generating a mesh that meets the quality requirements is a difficult task. Using a genetic algorithm to improve the mesh generation could therefore be a useful tool.

The case setup for this problem consisted of eight spheres enclosed by a rectangular box. Each of the spheres touches its three neighbouring spheres in a

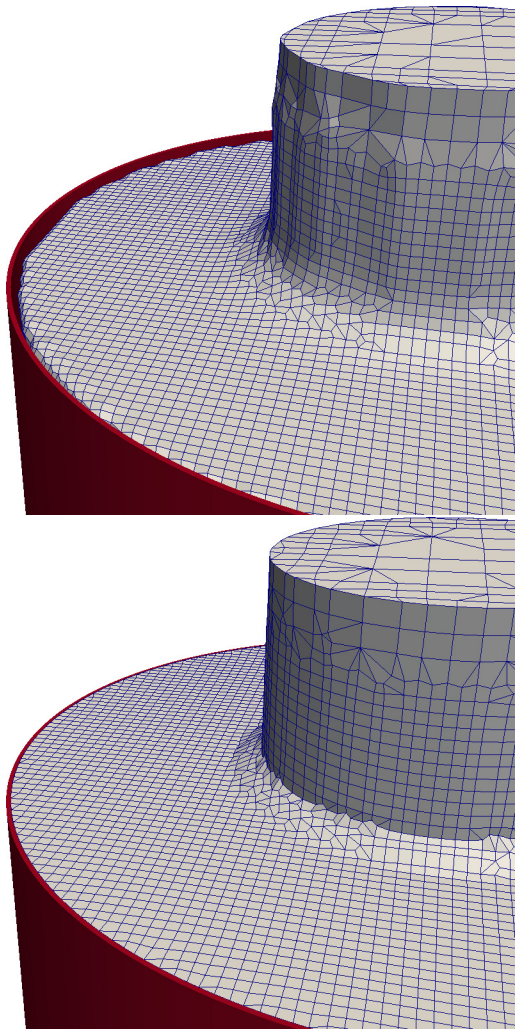

Fig. 6 Examples for bad (top) and good (bottom) snapping quality at the intersection of the large tube (red) and the connector disk in the bearing test case.

very small area. Figure 7 shows an axial and an isometric view of the geometry as well as the background mesh created with blockMesh, used in `snappyHexMesh` to confine the computational domain.

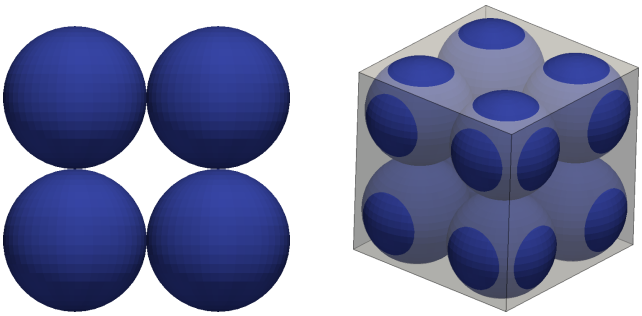

Fig. 7 Geometrical setup for the packed bed. Axial view (left) and isometric view with background mesh (right).

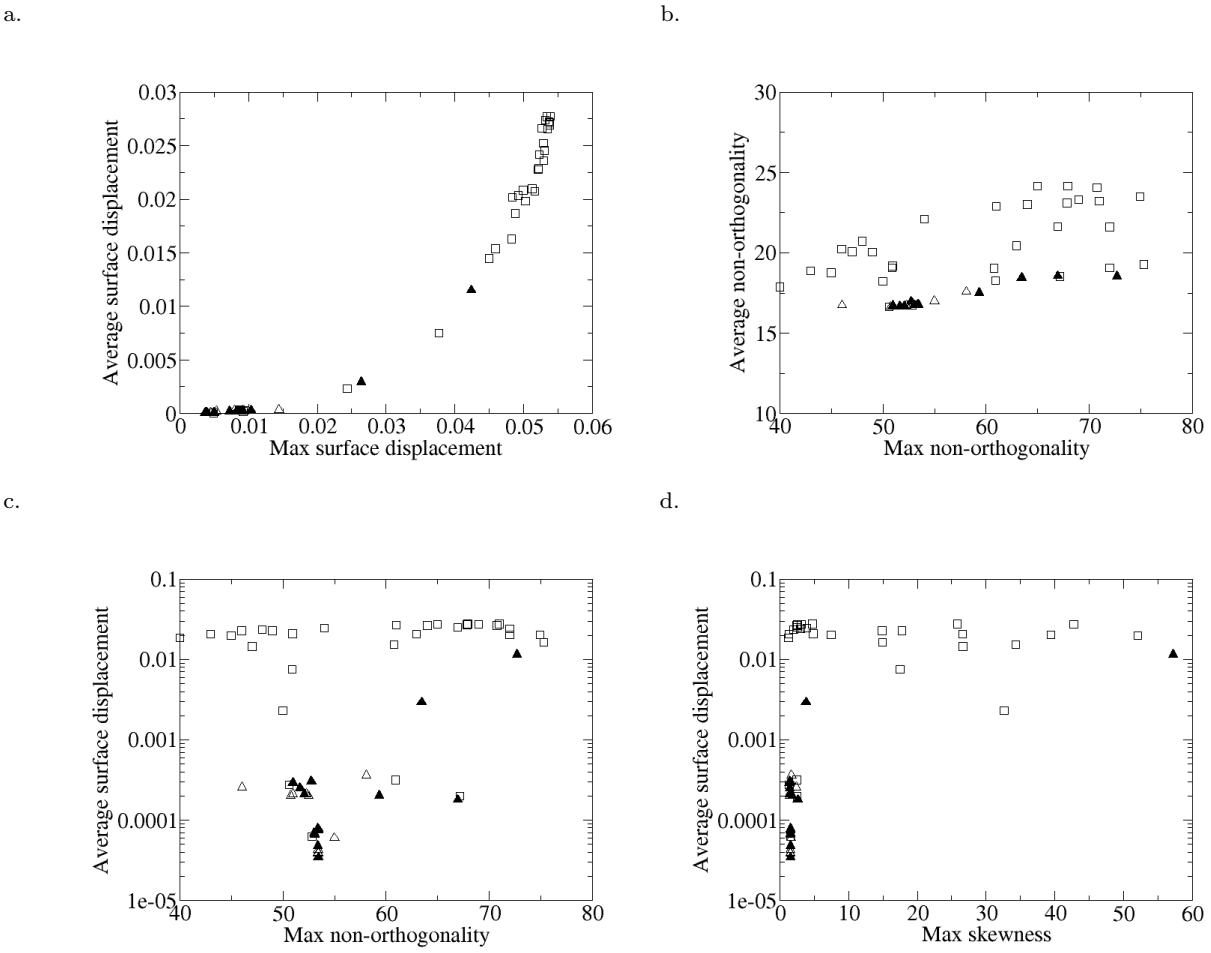

Fig. 8 Graphs of a. average surface displacement vs. max surface displacement, b. average mesh non-orthogonality vs. max non-orthogonality, c. average surface displacement vs. max non-orthogonality, d. av. displacement vs. max skewness for the packed bed case. In each case; squares represent the initial population, triangles the final population and filled triangles the Pareto set.

The initial mesh created with OpenFOAM's blockMesh utility consisted of 80,000 cells, or 20 by 20 by 20 in three dimensions, forming a cube with edge length $L = 1.8R$ not quite enclosing eight spheres of radius R. The snappyHexMesh parameters that were subject to optimisation and their allowed value ranges are listed in Table 4. The size of the solution space can be calculated from this table as $\approx 1.5 \times 10^{10}$. Although the same three optimisation targets were prescribed in this case: total number of cells, overall cell quality and accuracy of capturing the geometric features, just changing the quality restrictions in the snappyHexMeshDict had no influence on the resulting mesh size. Hence all individuals produced equally sized meshes, rendering the first optimisation objective obsolete.

As with the previous example, figure 8 shows relationships between a number of specific parameters of the mesh in detail. A very strong correlation is shown between average and max surface displacement in this case, and average non-orthogonality, surface displacement and max skewness are all seen unambiguously to improve by the final generation.

Table 4 Optimisation parameter value ranges for the packed bed test case as defined in the `gaDict`.

Parameter	min value	max value	accuracy
castellated mesh controls			
resolveFeatureAngle	30	60	1.
mesh quality controls			
maxNonOrtho 40	80	1.	
maxSkewness	2.0	10.0	0.1
snap controls			
nSmoothPatch	5	50	1.
tolerance 1.	2.5	0.1	
nRelaxIter	3	15	1.
nFeatureSnapIter	10	30	1.

When visualising the resulting meshes, it is possible to discern good from bad quality meshes in terms of capturing the geometric features. When looking at the thin volume in between two neighbouring spheres, the optimal shape would be a perfectly round circle with a small radius. Comparing a Pareto optimal mesh and a non-optimal mesh, as shown in Figure 9, one can see the higher roundness in the good mesh. Unfortunately this characteristic is not easily measurable automatically, otherwise it could be used as an additional optimisation objective.

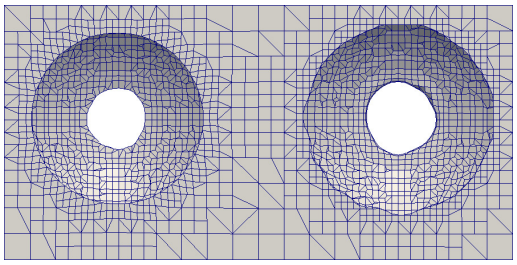

Fig. 9 Comparison of a Pareto front individual (left) versus a non-optimal solution (right). Notable is the difference in roundness and radius of the connecting area.

4.3 Ahmed Body

The characteristics of the *Ahmed* body were first described by Ahmed [1] in an experimental paper. It has become a well documented benchmark test case for car aerodynamics and is widely used to test turbulence models or other modelling techniques. Also many experimental data sets are available (e.g. [27,26]). To accurately predict lift and drag coefficients, as these are important quantities in automobile aerodynamics, good grid quality has to be assured especially in the area of eddy detachment at the back of the car and also on the underside of the body. This is even more the case for Large-Eddy Simulations as performed on this test geometry by various researchers [15,25,24,31]. The geometry pictured in Figure 10 was used here as a third, more realistic test case.

The initial rectangular mesh created with OpenFOAM's blockMesh utility consisted of 12,000 cells, or 40 by 30 by 10 in three dimensions. Figure 11 shows the results of the `snappyHexMesh` optimisation around the body's wheels while Figure 12 highlights the curved edge of the rear end of the body. In this test case a larger number of parameters was subject to the optimisation. A total of six values were modified, this time not only taken from the mesh quality sub-dictionary, but also from some controlling the castellation and the snapping procedure. The respective sub-dictionaries and the prescribed values are listed in Table 5. Again, a population size of 30 individuals was used and the optimisation was stopped after 20 generations.

As was the case for the bearing discussed previously, the Pareto set after the end of the optimisation procedure was rather large. In this case it still contained up to 50 % of the total population which were identified as being mutually non–dominant. This could mean, that the parameters modified in the `snappyHexMeshDict` had little or no influence on the outcome of the meshing process. Or it could be that creating a really 'bad' mesh for this geometry was actually difficult. One explanation for the latter could be that the fitness measurements were insufficient to identify discrepancies between target and result. In comparison to the bearing case, bad mesh quality could be very localised, mainly around the 'wheels' at the bottom of the body. If the quality restrictions were met on the majority of the surface, maybe small local errors do not influence the fitness very much.

Figure 13 shows again the correlations between the same specific mesh quality and accuracy parameters. Once again there is a strong correlation between average and max surface displacement, although not quite as strong as in the previous case; and in this case the max non-orthogonality has improved quite significantly. Max skewness has not been affected so strongly by the optimisation process.

Table 5 Parameter settings for `snappyHexMesh` for the two examples of the Ahmed body test case depicted above.

Parameter	bad example	good example
castellated mesh controls		
resolveFeatureAngle	45	32
mesh quality controls		
maxNonOrtho	65	80
maxSkewness	20	22
snap controls		
nSmoothPatch	3	7
nRelaxIter	3	6
nFeatureSnapIter	10	10

5 Analysis

Figures 5, 8 and 13 plot the same mesh and geometry parameters for each of the cases. Of these parameters, some are separate targets (e.g. max surface displacement), some are combined as part of the overall mesh quality index, such as max skewness, and some are alternative parameters which could have be targetted but were not. In this last category; we could have

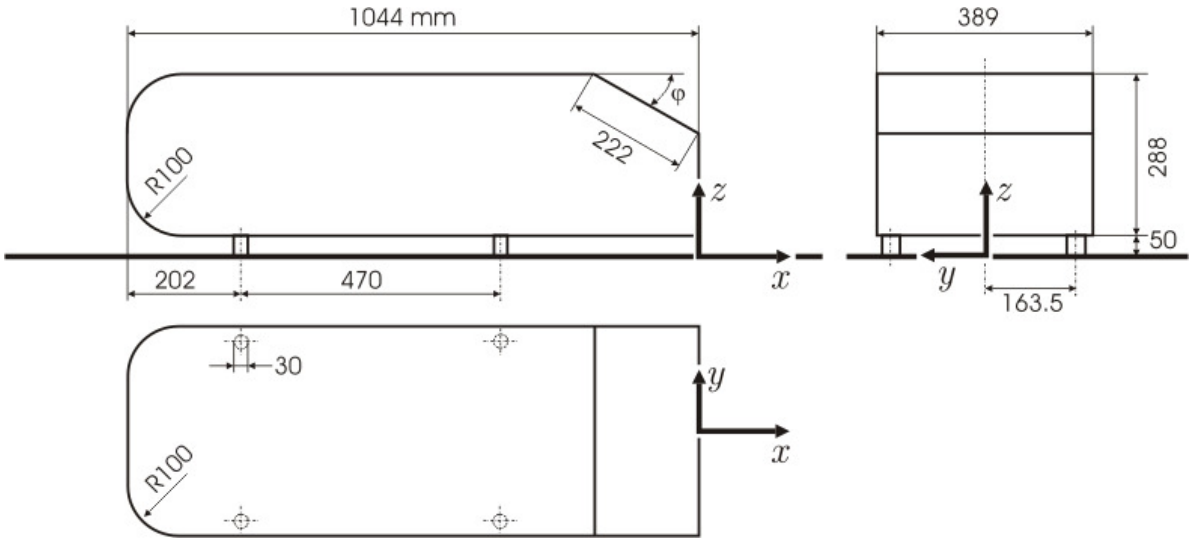

Fig. 10 Geometry of the Ahmed body as a simplified car model for aerodynamic investigations.

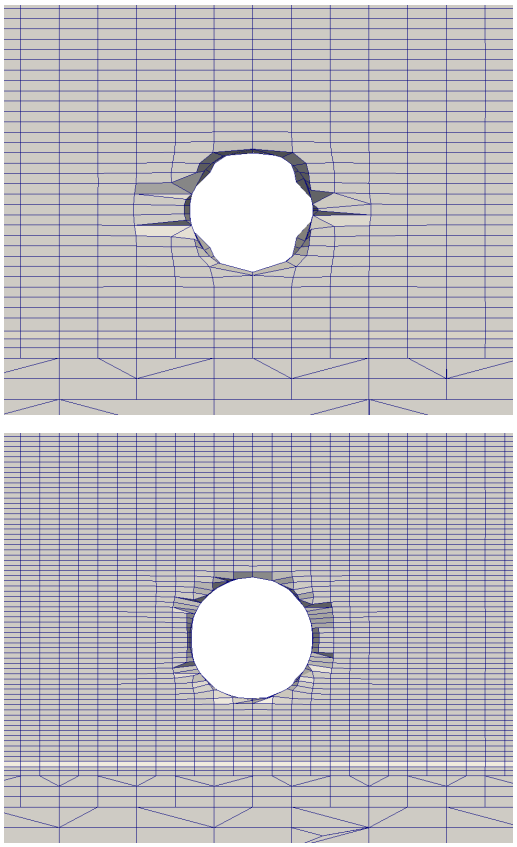

Fig. 11 Examples for bad (top) and good (bottom) snapping quality in the wheel region of the Ahmed body.

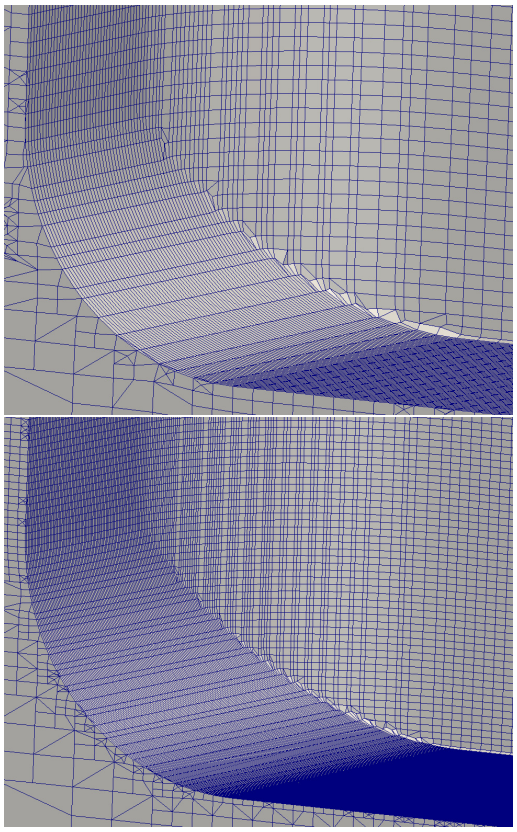

Fig. 12 Examples for bad (top) and good (bottom) snapping quality in the rear region of the Ahmed body.

targetted average values of surface displacement and non-orthogonality rather than max values, and the correlation between the average and maximum values of these parameters are explored in the graph series a. and b. For all the cases there is a correlation between average and maximum surface displacement (figures 5.a, 8.a and 13.a), suggesting that only one of these quantities need be examined, but this is particularly marked for the case of the spheres, less so for the other cases. However there is much less correlation between average and maximum non-orthogonality. Both of these are im-

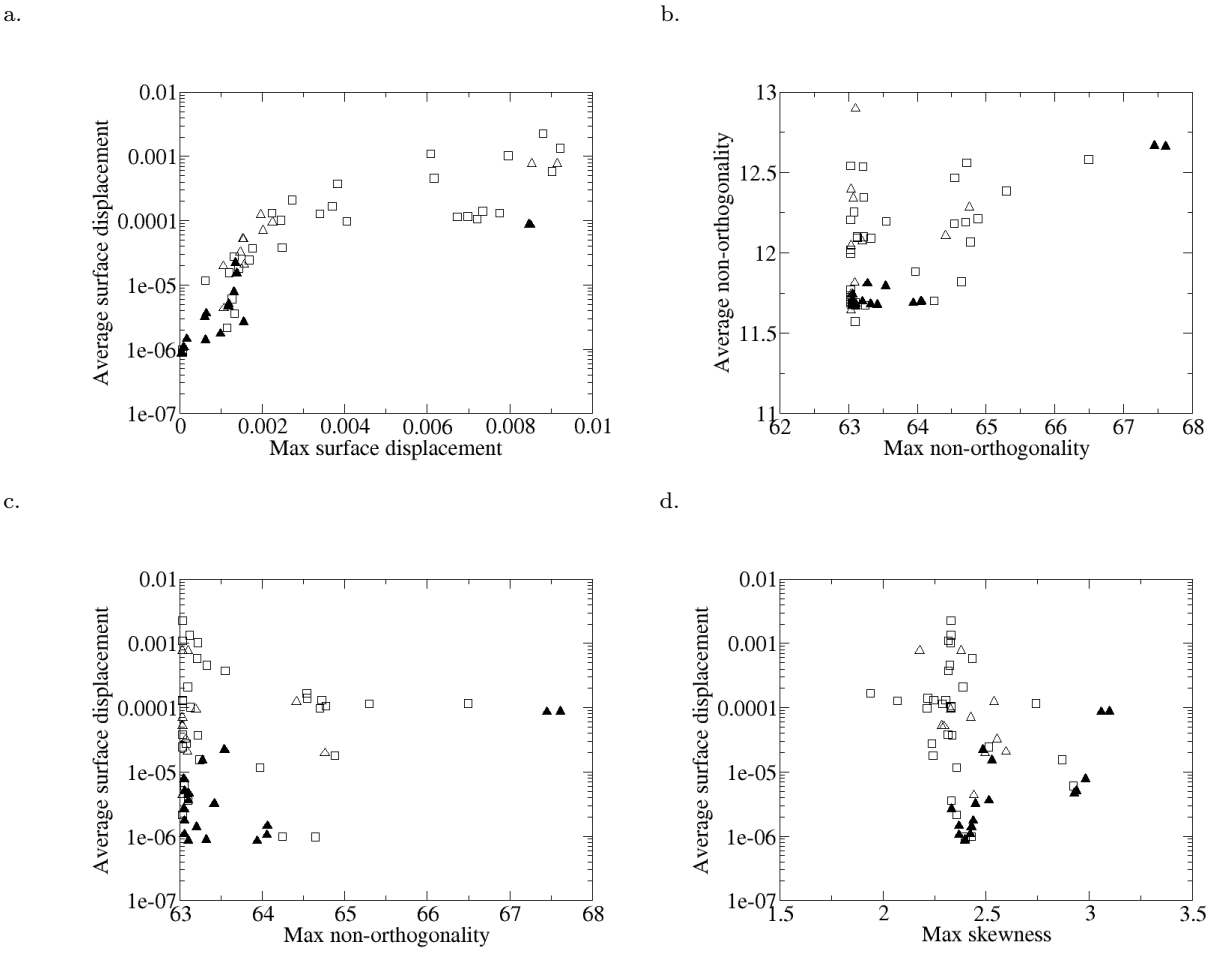

Fig. 13 Graphs of a. average surface displacement vs. max surface displacement, b. average mesh non-orthogonality vs. max non-orthogonality, c. average surface displacement vs. max non-orthogonality, d. average displacement vs. max skewness for the Ahmed Body case. In each case; squares represent the initial population, triangles the final population and filled triangles the Pareto set.

portant parameters in mesh quality, and so both should probably be targetted separately.

Indices such as max non-orthogonality, max skewess are part of the composite mesh quality index; it is instructive to break this down into the constituent components and look for correlation between these parameters and the average surface displacement (series c. and d. in the figures). Again, the results vary according to the case under consideration, and not in an obvious sequence. For the bearing case the surface displacement does not seem to easily correlate with either of these quantities (figure 5.c and d) although most of the Pareto set seem to cluster around a max non-orthogonality value of 70-80. As is accepted with GA processes, the initial population in each case explores the full parameter space, generating some individuals in the initial population which are quite good (low values of non-othogonality and skewness). By the final generation more individuals are optimal, although the population still contains less optimal solutions as well, as can be seen in figure 8.d. (one value of skewness in the Pareto set with a value of max skewness of nearly 60). In several cases there seem to be a definite limit to the values which can be achieved; for example the lowest value of max non-orthogonality for the Ahmed case seems to be around 63 (figures 13.b.and c.) There are evidently certain trade-offs between different parameters being made in these cases, although not perfectly as they are part of a composite fitness. This could be improved of course by making each parameter a separate optimisation objective. The geometric fidelity (as indexed by the average surface displacement) has unambiguously improved in all cases.

In addition the meshes have been examined visually in all cases; figures 6, 9 and 11 illustrate particular areas of the mesh for the different cases. In all cases the

mesh has been observed visually to improve in quality. Humans of course are very good at pattern recognition; computers considerably less so, and thus it would not always be possible to quantify aspects of the improvement. It should also be noted that OpenFOAM is a polyhedral code allowing mesh refinement through edge and face splitting. `paraview`, which has been used to display the meshes, deals with this by further triangulating some of the faces, and thus some of the diagonal triangulated lines in the images are ficticious.

Also of interest is the runtime for the process. All simulations were carried out on a twin 6-core AMD Opteron processor running at 800MHz, which at the beginning of the project was a fairly high spec desktop machine, but no more than is typical for CFD simulations in general. Typical runtimes of `snappyHexMesh` for final generation individuals on this machine were 4m45s for the bearing case and 16m40s for the more complex Ahmed case. Based on the evaluation of 600 individuals (30 individuals $\times$ 20 generations) gives estimates of 47.5 CPU hours processing for the bearing case and 166.8 CPU hours for the Ahmed case. Running the optimisation in parallel using 10 individual cores (i.e. 10 individual fitness evaluations performed simultaneously) actually took 6.28 hours for the bearing case and 13.5 hours for the Ahmed case, quite close to these estimates. Discrepancies will be accounted for by other operations in the fitness evaluation such as running `checkMesh`, and whether the 'typical' runtimes are in fact typical of the time taken (the examples evaluated were selected purely at random). To investigate the scaling of the calculation with mesh size, a repeat of the bearing case was undertaken with a $2\times$ base mesh size; $35 \times 35 \times 18 = 22050$ cells as against 10976 cells for the original base mesh (i.e. created by `blockMesh` before `snappyHexMesh` was run). Running `snappyHexMesh` straight off on a typical case with this new base mesh generated a mesh of $444,340$ cells, a considerable increase in cell count over the $60,000$ cells typical of the earlier calculation, and took 16m23s to execute. We note that this is shorter than a simple cell count scaling would indicate; `snappyHexMesh` has taken 3.5 times as long to generate a mesh with 7 times as many cells. Running 240 individuals (30 indiviuals $\times$ 8 generations) as a test on 8 cores (to reduce memory usage) took 20.1 CPU hours to execute; again an increase over the expected time due to variations in mesh and time to evaluate fitness. This would produce an expected run time of 50 hours to complete the full calculation (20 generations). Such a run time would not be at all unreasonable in the context of a full CFD calculation, particularly as these are automated processes which do not require human intervention beyond setting up.

6 Conclusions

Meshing is a highly complicated process which is recognised as having a significant impact on the quality (and sometimes the existance) of results from CFD. It is also highly labour-intensive; when combined with problems of CAD repair and cleanup, the meshing process can be the single most time-consuming part of the CFD process. Automated meshers such as `snappyHexMesh` were developed to provide robust if geometrically imprecise meshing solutions, but rely on significant numbers of input parameters whose values need to be determined typically by trial and error. We have shown here the potential of using Genetic Optimisation-based approaches to improve upon this. Runtimes of the optimisation took between 6 and 14 hours on a machine of a spec that might well be used for the CFD simulation, and whilst a full optimisation analysis might require substantially more evaluations, even an increase in evaluations of a factor of 10 would not be an impractical proposition. Given the importance in CFD of a high quality mesh, spending say 60 hours computing time to generate a good mesh should be seen as a worthwhile investment.

The work presented here we believe demonstrates the potential of this process but there are obviously significant further improvements to be made. In this work 3 specific metrics were targetted; the total number of cells in the mesh, the mesh quality and the geometric fidelity, as measured by the distance to the STL surface. Unfortunately due to aspects of the behaviour of `snappyHexMesh` the input parameters chosen for the meshing did not fully control the number of cells being generated. Full control of cell numbers would enable the investigation in detail of the tradeoff between mesh size and quality, a very significant aspect of mesh generation. Similarly, for simplicity we decided to use a single index of mesh quality, as an even-weighted combination of skewness, non-orthogonality and geometric fidelity. Other mesh quality indices could be included in this, and of course the weightings could be changed to reflect their importance in mesh generation process for particular cases. In a further analysis these indices could be regarded as independent objectives, allowing us to investigate tradeoffs, for example between geometric fidelity, maximum non-orthogonality and mean non-orthogonality. A final issue to be included would be constraints on the mesh generation process, for example requiring the creation of a boundary layer mesh which is often of importance in the final CFD analysis.

References

1. Abe, K., Ohtsuka, T.: Some salient features of the time-averaged ground vehicle wake. Tech. Rep. SAE-Paper 840300, SAE (1984)
2. Baker, M., Daniels, S., Young, P., Tabor, G.: Investigation of ow through a computationally generated packed column using CFD and Additive Layer Manufacturing. Computers and Chemical Engineering **67**, 159 – 165 (2014)
3. Baker, M.J., Tabor, G.R.: Computational analysis of transitional airflow through packed columns of spheres using the finite volume technique. Computers and Chemical Engineering **34**, 878 – 885 (2010)
4. Behzadian, K., Kapelan, Z., Savic, D., Ardeshir, A.: Stochastic sampling design using a multi-objective genetic algorithm and adaptive neural networks. Environmental Modelling and Software **24**, 530 – 541 (2009)
5. Berzins, M.: Mesh quality: a function of geometry, error estimates, or both? Eng.Comput. **15**(3), 236 – 247 (1999)
6. Deb, K., Anand, A., Joshi, D.: A computationally efficient evolutionary algorithm for real-parameter optimization. Evolutionary Computation **10**(4), 371–395 (2002)
7. Diskin, B., Thomas, J.: Effects of mesh regularity on accuracy of Finite-Volume schemes. In: 50th AIAA Aerospace Sciences Meeting including the New Horizons Forum and Aerospace Exposition, 2012-0609. AIAA (2012)
8. Field, D.: Laplacian smoothing and Delaunay triangulations. Commun.Numerical Methods **4**, 709 – 712 (1988)
9. Field, D.: Qualitative measures for initial meshes. International Journal for Numerical Methods in Engineering **47**, 887–906 (2000).
10. Fonseca, C.M., Fleming, P.J.: An overview of evolutionary algorithms in multiobjective optimization. Evolutionary Computation **3**, 1–16 (1995)
11. Freitag, L.: Local optimization-based untangling algorithms for quadrilateral meshes. In: Proceedings of the 10th International Meshing Roundtable, Newport Beach, California, pp. 397 – 406. Sandia National Laboratories (2001)
12. Freitag, L., Jones, M., Plassmann, P.: A parallel algorithm for mesh smoothing. SIAM J.Sci.Comput **20**(6), 2023 – 2040 (1999)
13. Gagné, C., Parizeau, M.: Genericity in evolutionary computation software tools; principles and case-study. Int.J. on Artificial Intelligence Tools **15**(2), 173 – 194 (2006)
14. Goldberg, D.E.: Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley (1989)
15. Hinterberger, C., Garca-Villalba, M., Rodi, W.: Large eddy simulation of flow around the Ahmed body. In: J.R. R. McCallen F. Browand (ed.) Lecture Notes in Applied and Computational Mechanics / The Aerodynamics of Heavy Vehicles: Trucks, Buses, and Trains. Springer Verlag (2004)
16. Jasak, H.: Error analysis and estimation for the Finite Volume method with applications to fluid flows. Ph.D. thesis, Imperial College (1996)
17. Jeng, Y., Chen, J.: Truncation error analysis of the finite volume method for a model steady convective equation. J.Comp.Phys **100**, 64 – 76 (1992)
18. Jourdan, L., Corne, D., Savic, D., Walters, G.: Preliminary investigation of the learnable evolution model for faster/better multiobjective water systems design. In: Proceedings of The Third International Conference on Evolutionary Multi-Criterion Optimization, EMO?05 (2005)
19. Kallinderis, Y., Kontzialis, C.: A priori mesh quality estimation via direct relation between truncation error and mesh distortion. J.Comp.Phys **228**, 881 – 902 (2009)
20. Knupp, P.: Achieving finite element mesh quality via optimization of the Jacobian matric norm and associated quantities. Part 1 – a framework for surface mesh optimization. Int.J.Num.Meth.Engng **48**, 401 – 420 (2000)
21. Knupp, P.: Achieving finite element mesh quality via optimization of the Jacobian matric norm and associated quantities. Part 2 – a framework for volume mesh optimization. Int.J.Num.Meth.Engng **48**, 1165 – 1185 (2000)
22. Knupp, P.: Algebraic mesh quality metrics. SIAM J.Sci.Comput. **23**(1), 192 – 218 (2001)
23. Knupp, P.: Algebraic mesh quality metrics for unstructured initial meshes. Finite Elements in Analysis and Design **39**, 217 – 241 (2003)
24. Krajnovic, S., Davidson, L.: Flow around a simplified car. Part 2: Understanding the flow. J. Fluids Engng. **127**(5), 919 – 928 (2005)
25. Krajnovic, S., Davidson, L.: Flow around a simplified car. Part 1: Large Eddy Simulations. J. Fluids Engng. **127**(5), 907 – 918 (2005)
26. Lienhart, H., Becker, S.: Flow and turbulence structure in the wake of a simplified car model. In: SAE 2003 World Congress, Detroit, USA (2003)
27. Lienhart, H., Stoots, C., Becker, S.: Flow and turbulence structures in the wake of a simplified car model (Ahmed model). In: DGLR Fach Symp. der AG STAB, Stuttgart University (2000)
28. McRae, D., Laflin, K.: Dynamic grid adaption and grid quality. In: J. Thompson, B. Soni, N. Weatherhill (eds.) Handbook of Grid Generation, chap. 34. CRC Press (1999)
29. Message Passing Interface Forum: MPI: A Message-Passing Interface Standard, Version 2.2. High-Performance Computing Center Stuttgart (2009)
30. Michalewicz, Z.: Genetic Algorithms + Data Structures = Evolution Programs. Springer, Berlin, Heidelberg (1996)
31. Minguez, M., Pasquetti, R., Serre, E.: High-order Large-Eddy Simulation of flow over the 'Ahmed body' car model. Physics of Fluids **20**(9), 095,101 (2008)
32. Owen, S.J., Shelton, T.R.: Evaluation of grid-based hex meshes for solid mechanics. Engineering with Computers pp. 1–15 (2014). DOI 10.1007/s00366-014-0368-8
33. Pagnutti, D., Ollivier-Gooch, C.: Two-dimensional Delaunay-based anisotropic mesh adaption. Engineering with Computers **26**, 407 – 418 (2010)
34. Parthasarathy, V., Graichen, C., Hathaway, A.: A comparison of tetrahedron quality measures. Finite Elements in Analysis and Design **15**, 255 – 261 (1993)
35. Robinson, J.: Quadrilateral and hexahedron shape parameters. Finite Elements in Analysis and Design **16**, 43 – 52 (1994)
36. Shepherd, J.F., Johnson, C.R.: Hexahedral mesh generation constraints. Engineering with Computers **24**(3), 195 – 213 (2008)
37. Srinivas, N., Deb, K.: Multiobjective optimization using nondominated sorting in genetic algorithms. Evolutionary Computation **2**(3), 221–248 (1995)
38. Versteeg, H.K., Malalasekera, W.: An Introduction to Computational Fluid Dynamics : The Finite Volume Method. Longman Scientific & Technical (1995)
39. Weller, H.G., Tabor, G., Jasak, H., Fureby, C.: A tensorial approach to computational continuum mechanics using object orientated techniques. Computers in Physics **12**(6), 620 – 631 (1998)

40. Zang, Y., Street, R.L., Koseff, J.R.: A non-staggered grid, fractional step method for time-dependent incompressible navier-stokes equations in curvilinear coordinates. Journal of Computational Physics **114**(1), 18–33 (1994)
41. Zitzler, E., Laumanns, M., Bleuler, S.: Comparison of multiobjective evolutionary algorithms: Empirical results. Evolutionary Computation **8**, 173–195 (2000)