Journal of Computational Physics 228 (2009) 6231–6249

Contents lists available at ScienceDirect

Journal of Computational Physics

journal homepage: www.elsevier.com/locate/jcp

Efficient mesh motion using radial basis functions with data reduction algorithms

T.C.S. Rendall *, C.B. Allen

Aerospace Engineering Department, University of Bristol, Bristol BS8 1TR, UK

ARTICLE INFO

Article history:
Received 5 June 2008
Received in revised form 6 May 2009
Accepted 11 May 2009
Available online 18 May 2009

PACS:
01.30.–y

Keywords:
CFD mesh deformation
Radial basis functions
Greedy algorithms
Aeroelastics

ABSTRACT

Mesh motion using radial basis functions has been demonstrated previously by the authors to produce high quality meshes suitable for use within unsteady and aeroelastic CFD codes. In the aeroelastic case the structural mesh may be used as the set of control points governing the deformation, which is efficient since the structural mesh is usually small. However, as a stand alone mesh motion tool, where the surface mesh points control the motion, radial basis functions may be restricted by the size of the surface mesh, as an update of a single volume point depends on all surface points. In this paper a method is presented that allows an arbitrary deformation to be represented to within a desired tolerance by using a significantly reduced set of surface points intelligently identified in a fashion that minimises the error in the interpolated surface. This method may be used on much larger cases and is successfully demonstrated here for a 10^6 cell mesh, where the initial solve phase cost reduces by a factor of eight with the new scheme and the mesh update by a factor of 55. It has also been shown that the number of surface points required to represent the surface is only geometry dependent (i.e. grid size independent), and so this reduction factor actually increases for larger meshes.

1. Introduction

During aeroelastic simulation it is usually necessary to update a fluid volume mesh based on the deformation of one or more surfaces. Similarly, it may also be convenient to deform a fluid mesh to accommodate a design modification, for example, during an optimisation exercise, rather than regenerate an entire mesh from scratch based on a new surface. A new mesh will have a new discretisation error and it may be preferable to assess the effect of design modifications on a mesh that is identical to the original in terms of connectivity, but with small changes in the geometry. In either case a convenient and efficient way of deforming a volume mesh around multiple moving bodies while preserving quality is required.

The authors have recently developed a unified meshless approach to fluid–structure coupling and mesh motion [1,2]. This method is based on multivariate interpolation using radial basis functions and has several significant advantages. Primarily, all volume mesh and flow-solver type dependence is removed, and all operations are performed on totally arbitrary point clouds of any form. Hence, all connectivity and user-input requirements are removed from the motion scheme, as only point clouds are required to determine the dependence. The method may equally well be applied to structured and unstructured grids, again because no connectivity information is required. Furthermore, no elaborate computations are required during an unsteady simulation, just matrix–vector multiplications, since the required dependence relations are computed only once prior to the start of the simulation and then remain constant. This property means the method is both perfectly parallel, since

* Corresponding author. Tel.: +44 0117 331 7641; fax: +44 0117 927 2771.
E-mail addresses: thomas.rendall@bristol.ac.uk (T.C.S. Rendall), c.b.allen@bristol.ac.uk (C.B. Allen).

0021-9991/$ - see front matter © 2009 Elsevier Inc. All rights reserved.
doi:10.1016/j.jcp.2009.05.013

only the data relevant to each structured block or unstructured partition are required to move those points, and totally independent from the flow-solver. This allows a completely generic 'black box' tool to be developed which is ideal for use in an optimisation [3,4] approach or unsteady simulation providing the surface deformations remain continuous (no topological changes, such as surface splitting or merging, are possible). However, while the approach is particularly attractive for high quality mesh motion it is quite expensive in pure form. Hence, this paper presents results of recent work aimed at developing a very efficient implementation. Smart data reduction schemes have been developed, which choose a significantly reduced data set based on minimising an error function, and are shown to reduce the cost of the mesh motion by over $50\times$ in terms of both time and memory requirements for a 10^6 cell mesh.

2. Mesh motion

There are many ways to deform a volume mesh during a time-accurate simulation, but the method adopted often depends on both the mesh type used and the application. For example, for a structured single block mesh, a simple interpolation scheme can be used as the grid connectivity is known explicitly, and this means algebraic type methods can be employed, see for example, Allen [5]. However, for multiblock meshes the situation is more complex, as the motion needs to cross block boundaries, where connectivity may be discontinuous, i.e. for example, an i,j face on one block may not connect to an i,j face on its neighbour. Each block can be parameterised, but a global parameterisation is not normally possible. In this case, motion can be achieved by solving a global set of dependence equations for the block corners, then solving for block faces, and finally interpolating the motions through each block, see for example, Jones et al. [6].

For an unstructured (or hybrid) mesh there are two common methods to solve the global problem: the spring analogy, first presented by Batina [7], or solution of a set of partial differential equations. In the spring analogy approach each node can be considered as connected to its neighbours by springs, the stiffness of which is proportional to the reciprocal of the length [7,8]. This approach has been applied to optimisation and unsteady flow problems, but is expensive and can lead to grid quality problems for larger motions. This was improved by Farhat et al. [9] to include a torsional spring to avoid the grid crossover problem, and in Blom [10] this torsional spring is shown to be essential for moving viscous meshes. Sheta et al. [11] have also reconsidered the formulation to use solid structural elements in an effort to prevent cell inversion.

The PDE solution approach usually involves an elliptic problem solution, see for example, Loehner and Yang [12,13] or Jasak and Tukovic [14]. Grid quality can be improved by solving a bi-harmonic set of equations that also preserve orthogonality [15,16]. However, for large meshes, or for an unsteady approach where the mesh may need to be moved several times per time-step, a spring analogy or elliptic solution approach is likely to be too expensive. Unfortunately, cheaper methods, for example, interpolation schemes, do not normally produce meshes of sufficient quality. Hence, cheaper but high quality alternative mesh motion schemes have been sought. For example, Melville [17,18] developed a method based on connectivity to moving surfaces, and applied it to overset meshes for aeroelastic simulations, Allen [19] presented a universal grid type-independent approach based on connectivity, which maintains orthogonality by accounting for surface rotations, and Cizmas and Gargoloff [20] also presented a scheme accounting for surface rotations. Liu et al. [21] have recently presented a cheap and very effective method, based on Delaunay mapping, although this does not account for surface rotations.

Mesh motion methods are normally driven by the motion of the solid surface and, as this is part of the fluid–structure coupling method, it can make sense to develop a unified approach to CFD–CSD coupling and volume mesh motion [1]. However, the goal here is to develop RBFs purely for mesh motion. The simplistic, exact, cost of using radial basis functions (RBFs) for mesh motion is quite large (in terms of data required), as the dependence matrix is strictly $N_{sp} \times N_{vp}$, where *sp* refers to surface points and *vp* to volume points, but the quality is comparable to the best that may be achieved with either a spring analogy or PDE. However, a significant point is that there are no elaborate computations required during the simulation, just matrix multiplications. As stated earlier it is the goal of this paper to present a framework in which the data requirements are significantly lowered to achieve an efficient mesh motion scheme applicable to meshes of any size.

3. Radial basis function mesh motion

3.1. Formulation

Examples of the application of RBFs to fluid–structure interpolation are given by Wendland et al. [22,23] where the method is applied to find static wing deflections in transonic flow. The general theory of RBFs is presented by Buhmann [24] and Wendland [25]. The work presented here moves the volume using the surface deformations; hence, for an aeroelastic simulation the deformed surface needs to be obtained first from the fluid–structure interpolation. A wide range of methods are available for this, also including radial basis functions, and the reader is referred to Wendland et al. [22,23] for details.

Relevant work in area of mesh motion is by Sprekreijse et al. [26], who uses a volume spline (RBF interpolation with $\phi(\xi) = \xi$) to calculate the block vertices on a structured mesh and then updates the entire mesh using transfinite interpolation (TFI) within each block. This work, that has its origins in a work by Hounjet and Meijer [27], was combined with another volume spline for fluid–structure interpolation and used to analyse a full F-16 configuration under static conditions by Prananta and Meijer [28]. More recent applications to mesh motion are by de Boer and Bijl [29,30] and Jakobsson and Amoignon [31]. de Boer and Bijl [29,30] also used compact RBFs for mesh motion, but included a polynomial term and did not explore

any methods to increase the speed of the method; in comparison, for the work presented here polynomial terms are excluded and a data reduction method is used to minimise the cost and make the scheme suitable for large meshes. de Boer and Bijl [29,30] also moved the volume mesh directly from the structural mesh, whereas in the work presented here the volume is deformed from the surface, making the method applicable outside aeroelasticity. Jakobsson and Amoignon [31] used a wide range of basis functions to compute mesh motion for an adjoint fixed-wing optimisation process. For this the RBF motion scheme was advantageous as it provided the derivative of the mesh node positions with respect to the control (surface) points. The size of the problem was tackled by reducing the control points to be a small, evenly distributed subset of the surface points, while showing that this strategy had only a small influence on the mesh node positions. An assessment of the efficiency of the RBF method compared to a Laplacian approach was given, together with consideration of the resulting mesh quality. The RBF method outperformed that based on Laplace's equation in terms of speed and preserved the quality of the initial mesh in nearly all cases.

The form of an RBF interpolation is [1,2]

$$s(\mathbf{r}) = \sum_{i=1}^{i=N} \alpha_i \phi(\|\mathbf{r} - \mathbf{r}_i\|) \tag{1}$$

Here, $s(\mathbf{r})$ is the function to be evaluated at location $\mathbf{r}$ and will define the motion of the volume points, ϕ is the form of function adopted (see Table 1 for options, where $\mathbf{v}$ is any vector and c and r are tuning parameters [25,24] for the other functions), the index i identifies the centres for the RBFs, while $\mathbf{r}_i$ is the location of that centre. In this case the centres correspond to mesh points located on the moving surfaces. The coefficients α_i are found by requiring exact recovery of the original function at these points $\mathbf{r}_i$. It is common to append polynomial terms to the interpolation (which allows rotations to be recovered exactly) but for mesh motion this is undesirable as it results in motion of the far-field boundary (Table 2).

Functions may be divided in to three different categories: global, local and compact. Global functions are always non-zero and grow moving away from the origin; examples are the thin plate spline, multiquadric and volume spline. Local functions are always non-zero but decay moving away from the origin, these include the Gaussian and the inverse multiquadric. Compact functions share the decaying property of local functions, but in addition reach zero at some finite distance, termed the support radius. Global functions can produce large deformations of the far-field if used for mesh motion, so a local character is preferred. Compact functions bring the additional benefit that deformation is strictly limited to remain within the support radius.

Wendland's C2 function is selected as the basis function since this provides a satisfactory compromise between the quality of the mesh motion, owing to its improved smoothness over the C0 function, and the conditioning of the linear system solved to find the set of coefficients α_i. In addition the compact support strictly limits the deformation to be within the region bounded by the support, which is a convenient trait. Further comparison of different functions may be found in de Boer and Bijl [29].

An important point is that compact functions have a value of zero at large distances away from their centres. This means it is necessary to work with displacements rather than actual position, i.e. $\Delta\mathbf{x}$ is the function to be interpolated rather than $\mathbf{x}$.

Table 1
Basis functions.

Name	Definition
Gaussian (G)	$\phi(\xi) = e^{-\alpha\xi}$
Thin plate spline (TPS)	$\phi(\xi) = \xi^2 ln(\xi)$
Volume spline (VS)	$\phi(\xi) = \xi$
Hardy's multiquadric (HMQ)	$\phi(\mathbf{v}) = (c^2 + \|\mathbf{v}\|^2)^{\frac{1}{2}}$
Hardy's inverse multiquadric (HIMQ)	$\phi(\mathbf{v}) = \frac{1}{(c^2+\|\mathbf{v}\|^2)^{\frac{1}{2}}}$
Wendland's C0 (C0)	$\phi(\xi) = (1-\xi)^2$
Wendland's C2 (C2)	$\phi(\xi) = (1-\xi)^4(4\xi+1)$
Wendland's C4 (C4)	$\phi(\xi) = (1-\xi)^6(35\xi^2+18\xi+3)$
Wendland's C6 (C6)	$\phi(\xi) = (1-\xi)^8(32\xi^3+25\xi^2+8\xi+1)$
Euclid's Hat (EH)	$\phi(\xi) = \pi\left(\left(\frac{1}{12}\xi^3\right) - r^2\xi + \left(\frac{4}{3}r^3\right)\right)$

Table 2
Algorithm costs.

Algorithm	Solve	Update
Full method	N_{sp}^3	$N_{sp}N_{vp}$
1	$nN_{sp} + N_{sel}^3$	$N_{sel}N_{vp}$
2	$\sum_{j=1}^{n} j^3 + jN_{sp}$	$N_{sel}N_{vp}$
3	$N_{sp}\left(n - \lfloor\frac{n}{m}\rfloor\right) + \sum_{j=1}^{j=\lfloor n/m \rfloor}[(mj)^3 + mjN_{sp}]$	$N_{sel}N_{vp}$

For easy computational treatment that maximises the use of matrix multiplication the problem is written in the following fashion.

Using s to denote a surface mesh point

$$\Delta \mathbf{x}_s = \mathbf{M}\mathbf{a}_x \tag{2}$$

$$\Delta \mathbf{y}_s = \mathbf{M}\mathbf{a}_y \tag{3}$$

$$\Delta \mathbf{z}_s = \mathbf{M}\mathbf{a}_z \tag{4}$$

where

$$\Delta \mathbf{x}_s = \begin{pmatrix} \Delta x_{s_1} \\ \vdots \\ \Delta x_{s_N} \end{pmatrix} \quad \mathbf{a}_x = \begin{pmatrix} \alpha^x_{s_1} \\ \vdots \\ \alpha^x_{s_N} \end{pmatrix} \tag{5}$$

(analogous definitions hold for $\Delta \mathbf{y}_s$ and $\Delta \mathbf{z}_s$ and their $\mathbf{a}$ vectors)

$$\mathbf{M} = \begin{pmatrix} \phi_{s_1 s_1} & \phi_{s_1 s_2} & \cdots & \phi_{s_1 s_N} \\ \vdots & \vdots & \ddots & \vdots \\ \phi_{s_N s_1} & \phi_{s_N s_2} & \cdots & \phi_{s_N s_N} \end{pmatrix} \tag{6}$$

with

$$\phi_{s_1 s_2} = \phi(\xi_{s_1 s_2}) = \phi(\|\mathbf{r}_{s_1} - \mathbf{r}_{s_2}\|/R) \tag{7}$$

indicating the basis function evaluated on the distance between points s_1 and s_2 with a support radius of R.

To determine the dependence of the volume points on the surface points the following matrix must be formed, where v indicates a volume node:

$$\mathbf{A} = \begin{pmatrix} \phi_{v_1 s_1} & \phi_{v_1 s_2} & \cdots & \phi_{v_1 s_N} \\ \vdots & \vdots & \ddots & \vdots \\ \phi_{v_N s_1} & \phi_{v_N s_2} & \cdots & \phi_{v_N s_N} \end{pmatrix} \tag{8}$$

so that the positions of the volume points, given by the vectors $\Delta \mathbf{x}_v, \Delta \mathbf{y}_v$ and $\Delta \mathbf{z}_v$ are

$$\Delta \mathbf{x}_v = \mathbf{A}\mathbf{a}_x = \mathbf{A}\mathbf{M}^{-1}\Delta \mathbf{x}_s = \mathbf{H}\Delta \mathbf{x}_s \tag{9}$$

$$\Delta \mathbf{y}_v = \mathbf{A}\mathbf{a}_y = \mathbf{A}\mathbf{M}^{-1}\Delta \mathbf{y}_s = \mathbf{H}\Delta \mathbf{y}_s \tag{10}$$

$$\Delta \mathbf{z}_v = \mathbf{A}\mathbf{a}_z = \mathbf{A}\mathbf{M}^{-1}\Delta \mathbf{z}_s = \mathbf{H}\Delta \mathbf{z}_s \tag{11}$$

Or, more concisely

$$\begin{pmatrix} \Delta \mathbf{x}_v \\ \Delta \mathbf{y}_v \\ \Delta \mathbf{z}_v \end{pmatrix} = \begin{pmatrix} \mathbf{H} & 0 & 0 \\ 0 & \mathbf{H} & 0 \\ 0 & 0 & \mathbf{H} \end{pmatrix} \begin{pmatrix} \Delta \mathbf{x}_s \\ \Delta \mathbf{y}_s \\ \Delta \mathbf{z}_s \end{pmatrix} \tag{12}$$

It is clear that there are two stages for accomplishing the mesh motion. First comes the 'solve', which is defined as either finding $\mathbf{M}^{-1}$ or solving for $\mathbf{a}_{x/y/z}$. This is followed by the 'update', which is equivalent to either multiplying $\mathbf{a}_{x/y/z}$ by $\mathbf{A}$ or multiplying through by $\mathbf{H}$, depending on which option was used for the solve.

3.2. Implementation and efficiency

In the form presented the matrix $\mathbf{H}$ gives the fixed weightings used to move the volume points. These can be calculated once at the beginning of the computation (external to the time loop), written to a file and then used within the time loop, as indicated in Figs. 1 and 2. This makes the method very simple for small to medium sized meshes, but for larger cases $\mathbf{H}$ is large and difficult to store, as it contains $N_{sp} \times N_{vp}$ entries. An alternative, also listed in Fig. 2, is to solve the linear system during each timestep to obtain only $\mathbf{a}_{x/y/z}$ so that the matrix $\mathbf{H}$ is never explicitly formed. This could be done directly with a decomposition but since the matrix $\mathbf{M}$ is symmetric positive definite (providing the basis function is defined in the correct fashion [25,24]) then a simple iterative method, such as the popular conjugate gradient method [32], may also be applied. If the function is also compact then $\mathbf{M}$ will usually have a degree of sparsity, making the application of an iterative method even more appealing. However, choosing between a direct and iterative solver is ultimately hardware and problem size

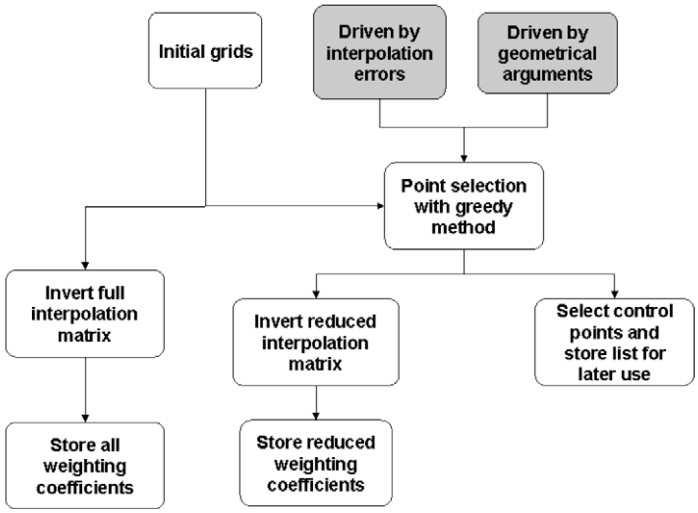


Fig. 1. Flowchart of methods for achieving mesh motion: before time loop begins.

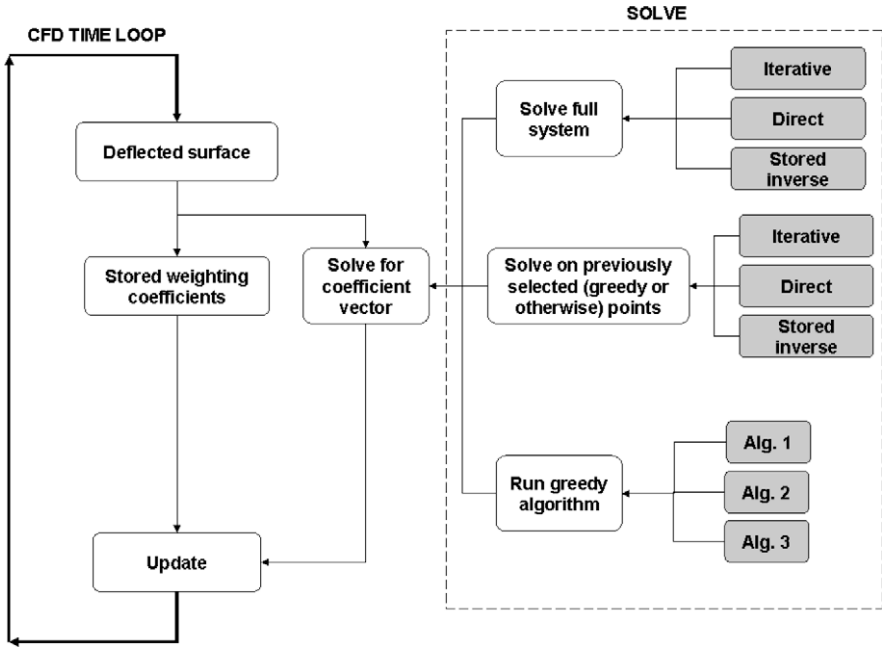


Fig. 2. Flowchart of methods for achieving mesh motion: during time loop.

dependent. In the current work the conjugate gradient method is applied as it leaves open the option to deal with much larger problems if needed. It should be noted for the small system sizes used later in this work a direct solution could be preferable, although for small systems the question of cost is inherently of less interest.

Examples of fluid–structure coupling and mesh motion (with quality analysis) using RBFs are given by Allen and Rendall [1,2]. These results are illustrated in Fig. 4 for a 2 m aerodynamic surface tip deflection in the 18th mode of the Brite-Euram MDO (multidisciplinary optimisation) wing [33] of semispan 35 m, for a mesh with 250×10^3 cells and 3881 surface points, where Wendland's C2 function was used for the full list of surface points with a support radius of 1.0 and 2.0 mean aerodynamic chords (MAC). These values were selected for demonstration purposes; the practical restriction on the support radius is that it should be larger than the biggest motion of any surface point, usually by a factor of several times (in this case 5). Deformation of the wing surface in the 18th mode is shown in Fig. 3. Although the mesh motion produced is of high quality the cost of the full method would be too high to contemplate for use on a larger mesh; the time and memory associated with

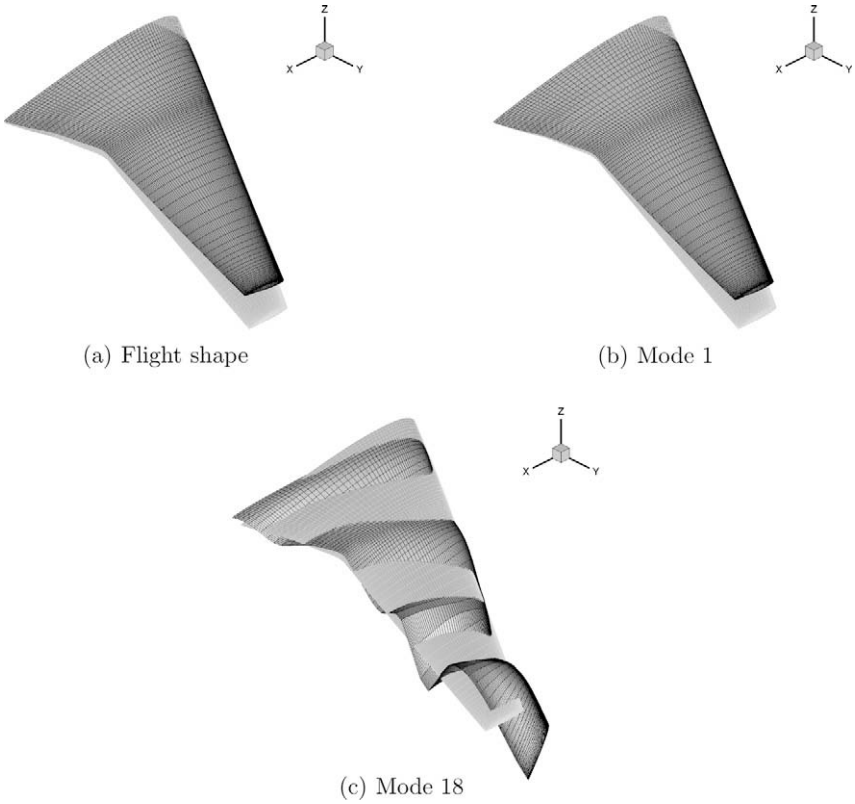


Fig. 3. MDO wing mesh motion deflected shapes. Grey – original shape, black – deflected shape.

solving the interpolation scale with N_{sp}^3 and N_{sp}^2 respectively while a mesh update scales with $N_{sp} \times N_{vp}$. The objective of this work is therefore to preserve the good quality of the motion scheme whilst reducing the memory and time requirements to manageable levels.

Considering how to achieve this, it may be observed that the dominant factor in the cost of the method is N_{sp} as this influences the cost of both solving and updating the interpolation. Two philosophies exist for compressing the method: one seeks an exact solution to an approximate problem, while the other an approximate solution to the exact problem. Greedy methods [34–36] fall into the former category, while most RBF techniques, such as the partition of unity [25,23,37], multilevel [38] and alternating projection methods [25] reside in the latter, although there can be overlap as these methods sometimes begin to merge [35]. For mesh motion the partition of unity method, which forms local interpolations and then assembles these through a weighting function, is less desirable because discontinuities or a loss of smoothness can exist in regions where interpolations overlap. Multilevel and alternating projection methods are most effective at speeding the solve stage, but may only influence update costs to a lesser extent.

An appealing option is not to use the full set of surface points to define the motion, but rather a suitable reduced set that represents the deformed geometry to a good degree of accuracy. Jakobsson and Amoignon [31] made this choice by simply thinning the cloud of surface points without attempting to preserve geometrical accuracy. Although this is a sound approach, it is possible to adopt a method that both reduces the cost and simultaneously attempts to maximise the accuracy. Such a method has recently been considered for RBF interpolation problems with greedy algorithms and shown to be of promise [39–41], so that approach is adopted for this work.

If only a reduced set of surface points (the 'control points') define the motion and no intervening action is taken, then any surface point which is not a control point will be moved by the interpolation. This means sacrificing the manner in which the surface of the volume mesh conforms to the desired surface displacements, and is only suitable providing the error at these points remains below an acceptable level, which in turn must be achieved by making a 'good' choice for the set of control points. Alternatively, the remaining incorrect surface points could be corrected exactly, providing the mesh quality did not deteriorate in the first layer of cells away from the surface. The correction could also be propagated a few cell layers away from the surface using a decaying function, thereby preserving the quality in this region more effectively.

Exactly how a greedy method would be implemented within a CFD code depends on whether it is to be placed inside or outside the time loop. If placed inside then it could be run every timestep, or every nth timestep, using the real surface deformations to keep the errors low. Placing the greedy method outside the loop would be suitable if the selected centres were to

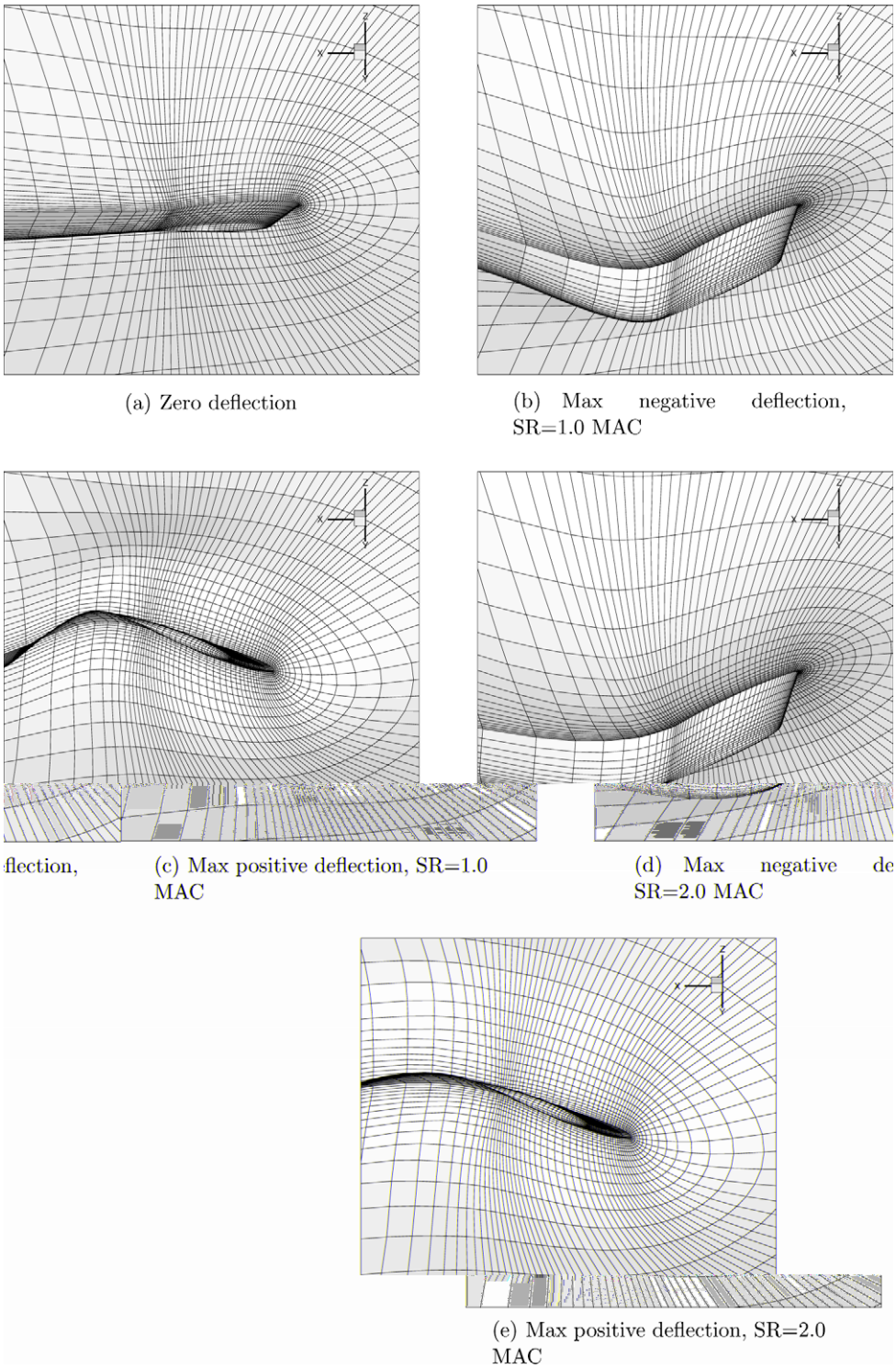


Fig. 4. Mesh Motion for MDO 18th mode, mesh at three positions. Surface, wake plane, and a chordwise plane.

be kept constant during the simulation, though this would imply choosing the points before the simulation started using a benchmark deformation (see Section 4). If points were selected before the simulation then it would also be possible to store a factorisation to speed solution. The various options for how to handle the implementation outside and within the time loop are illustrated in Figs. 1 and 2.

4. Greedy algorithms

A greedy algorithm [34–36] is one that always makes the optimal choice based on only a local assessment of the situation. The method is 'greedy' because at any point the algorithm targets only the largest error for correction and makes no account for the consequences of this choice. In the context of RBFs greedy algorithms have been considered because they simultaneously help to tackle both the computational challenges of solving and updating the interpolation. Upon completion they return a reduced set of centres, together with their associated coefficients, that reproduce the function to within a certain tolerance. Since RBF interpolation is a collocation type problem if all points were used the error would be zero everywhere.

Greedy algorithms for RBF interpolation can be considered in a number of different forms. The difference between these forms is only a matter of how they derive the information used to decide which point, or points, to add next to the list of points being used ('active list') and the error signal used to make the choices. The behaviour of this approach can be controlled, for example, by changing the number of points added to the active list on each execution of the loop and the initial points used to begin the method. Descriptions of the candidate error signals shall be given first, followed by a discussion of the algorithms developed here.

4.1. Error signal description

The first question to consider is what type of signal to use to control the greedy method's choices. It is possible to use either geometrical arguments [39], interpolation errors [41], the power function [39] or any other prescribed function for this purpose (as shown in Fig. 1). When using the interpolation errors it is possible to choose a form that is independent of the data; this raises the interesting question of whether or not the selected points should be dependent on the original function.

The errors $\mathbf{E}^x, \mathbf{E}^y, \mathbf{E}^z$ in either x, y or z come from the actual surface deflection and are one good error signal. That error may be expressed at each point

$$\mathbf{E}^x = (\Delta \mathbf{x}_s - \mathbf{A}\mathbf{M}^{-1}\Delta \mathbf{x}_{red}) \tag{13}$$

$$\mathbf{E}^y = (\Delta \mathbf{y}_s - \mathbf{A}\mathbf{M}^{-1}\Delta \mathbf{y}_{red}) \tag{14}$$

$$\mathbf{E}^z = (\Delta \mathbf{z}_s - \mathbf{A}\mathbf{M}^{-1}\Delta \mathbf{z}_{red}) \tag{15}$$

These errors may be squared and added to give a single error at point i

$$E_i = \sqrt{\left(E_i^x\right)^2 + \left(E_i^y\right)^2 + \left(E_i^z\right)^2} \tag{16}$$

where $\mathbf{A}$ and $\mathbf{M}$ are the conventional evaluation and interpolation matrices for the centres on the active list. $\Delta \mathbf{x}_{red}, \Delta \mathbf{y}_{red}, \Delta \mathbf{z}_{red}$ consist of the entries of $\Delta \mathbf{x}_s, \Delta \mathbf{y}_s, \Delta \mathbf{z}_s$ for those points on the active list (which is a reduced list of all points).

4.2. Algorithm descriptions

Algorithm 1 (*Greedy one point algorithm*). The first form of the greedy method appends the data site where $\mathbf{E}$ attains its largest value and then corrects only that point. Only a simple scan over all points is needed to identify the single worst error, so this method has a cost proportional to N_{sp}.

Where i_{arb} may be any index

$$f_{max} = f_{i_{arb}} \tag{17}$$

$$i_{max} = i_{arb} \tag{18}$$

Find the required constant to correct the selected point

$$\beta = \frac{f_{max}}{\phi(0)} \tag{19}$$

Add correction to coefficient vector

$$\alpha(i_{max}) \leftarrow \alpha(i_{max}) + \beta \tag{20}$$

Correct the interpolation at all points, where f_{ev} is the evaluation of the interpolation

$$f_{ev}(\mathbf{r}) \leftarrow f_{ev}(\mathbf{r}) + \beta\phi(\|\mathbf{r} - \mathbf{r}_{i_{max}}\|) \tag{21}$$

Find new errors, where f_{ex} is the exact value of the function

$$e(\mathbf{r}) = F(f_{ex}(\mathbf{r}) - f_{ev}(\mathbf{r})) \tag{22}$$

Identify maximum error

$$i_{max} = i(max(e)) \tag{23}$$

$$f_{max} = f_{ex}(\mathbf{r}_{i_{max}}) - f_{ev}(\mathbf{r}_{i_{max}}) \tag{24}$$

Return to Eq. (19), or if finished take selected points and solve exactly on these control points

Either a one point correction (see Wendland [25,34]) or a full solution (see Carr et al. [41]) may be used in each cycle of the greedy loop. Even if an exact solution is used at each stage (as in Algorithm 2, described below) then the set of points could still not be expected to be the actual optimal set of points because a greedy algorithm will always need some arbitrary starting point, which will continue to influence how the point set evolves indefinitely unless points are systematically removed. Although adding points where the error is large is a simple idea it is more complicated to decide how to remove 'unnecessary' points, to the extent that this is almost never done. This is because although there is a list of points where the error is large, all points that have been used have a small or zero error, and this makes it difficult and costly to distinguish between them. Algorithm 1 does tend to show a degree of self limiting behaviour in terms of how many points it uses (see Section 5), often returning to correct a previously identified point rather than introducing a new one. This is not an advantage in practice as it is related to the inefficiency of the method at solving the linear system, i.e. although the number of points increases slowly, the error also goes down comparatively slowly.

The function F corresponds to any suitable function for converting the error in all three coordinates into a single number at any point. In this case, the Euclidean distance has been used. A possible variant of the algorithm exists where a different set of points is used to interpolate each coordinate direction but it is not thought to offer any additional advantages.

Algorithm 1 is very fast, probably the fastest of all possible RBF interpolation algorithms, running in a time proportional to the number of centres N_{sp}. Unfortunately, as a pure method for solving the interpolation problem there are severe drawbacks. Convergence is very fast initially, but rapidly drops off as points are added with a relationship similar to the form $1/k$, where k is the iteration number. It might be tempting to think that the result would be the exact interpolation when all points are used, but this is not the case either, in fact it would be necessary to continue iterating well beyond the stage at which all points are being used for the method to approach the true solution. This would still be unlikely to occur in a reasonable amount of time because of the way points interact; whenever a correction is made a further error is introduced at another point. The method continues to 'iron out' the largest error at every stage, but when using a global function or a large support radius this could take a long time.

For this reason it is beneficial to follow the execution of Algorithm 1 with a full solve, where all of the points selected by the greedy method are taken and the interpolation solved exactly on those points. The effect of doing this is shown in Fig. 5 for the 250×10^3 cell mesh and reveals that the surface points around the tip are improved, particularly in the region of the trailing edge.

The second greedy algorithm, rather than solving and correcting a single point at a time, solves the complete system before choosing which new point to include.

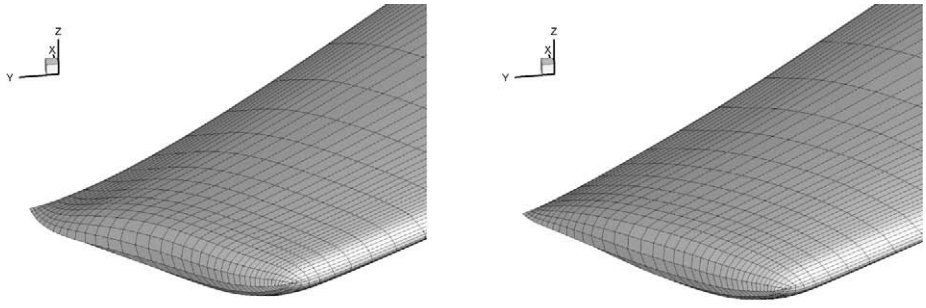
Algorithm 2 (*Greedy full point algorithm*). Guiding the greedy method with a full solution to the interpolation problem at each step means that the selection of points is not influenced by using only a basic local correction at each point, as presented in Algorithm 1. It would also be possible to run this method adding more than one point at a time to the active list; in this case it would be necessary to implement a sorting system to select all the largest errors.

Select initial set of points

$$\mathbf{a}_{red} = \mathbf{a}_{int} \tag{25}$$

Solve interpolation

$$\mathbf{a}_{red} = \mathbf{M}^{-1}\mathbf{f}_{red} \tag{26}$$



(a) Algorithm 1 only

(b) Algorithm 1 followed by a full solve

Fig. 5. Wing tip with and without a full final solve for Algorithm 1.

Evaluate interpolation at all points

$$\mathbf{f}_{ev} = \mathbf{A}\mathbf{a}_{red} \tag{27}$$

Find errors

$$e(\mathbf{r}) = F(f_{ex}(\mathbf{r}) - f_{ev}(\mathbf{r})) \tag{28}$$

Include new point(s) with largest e and return to Eq. (26)

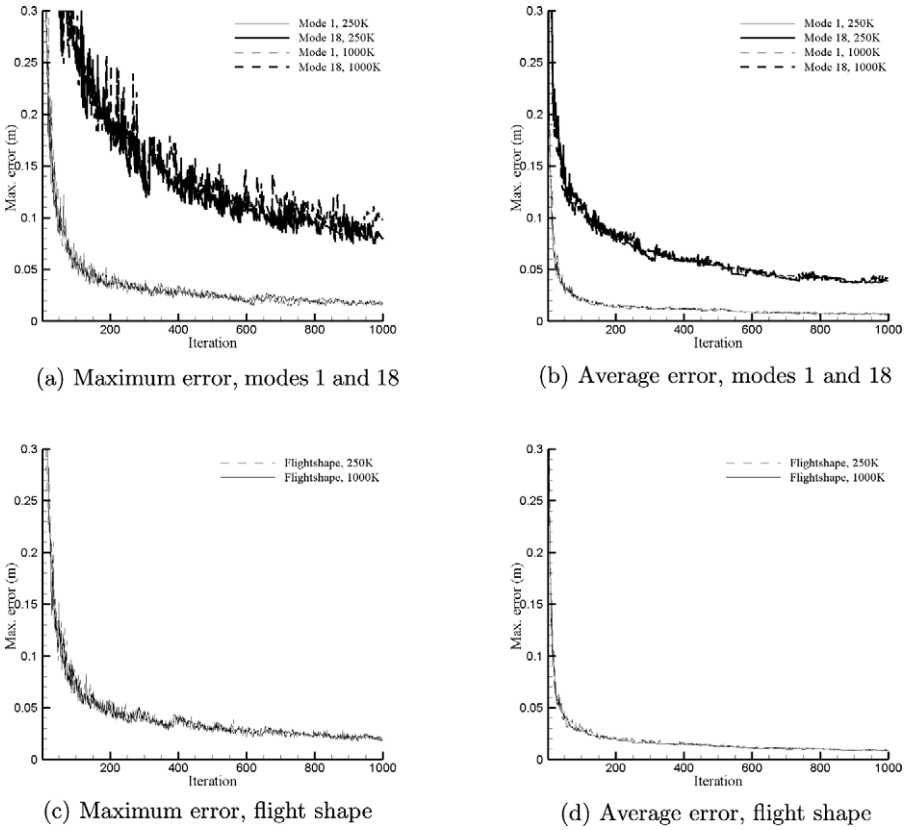
Another option is to run these methods together to create a hybrid method. The fast point adding of Algorithm 1 can be combined with the superior point efficiency of Algorithm 2 by alternating the cycles; Algorithm 1 can be executed most of the time for speed, but, say, on every 10th execution Algorithm 2 is used.

Algorithm 3 (*Hybrid algorithm*). This method has been developed to use all of the different settings for Algorithms 1 and 2; however, the main factor determining the performance is m. Once every m cycles Algorithm 2 runs instead of Algorithm 1. Since Algorithm 2 can only deal with distinct points a list must be kept to prevent any point being visited more than once.

do $n = 1, n_{stop}$
if n is a multiple of m then
→ run one step of Algorithm 2
else
→ run one step of Algorithm 1
endif
enddo

5. Greedy results

The Brite-Euram MDO aircraft [42] closely resembles the A380 and has a semispan of 35 m. Large static deformations in the range of 3 m are observed for the highest cruise loads. For assessing the mesh motion two meshes are used: a 250×10^3



(a) Maximum error, modes 1 and 18

(b) Average error, modes 1 and 18

(c) Maximum error, flight shape

(d) Average error, flight shape

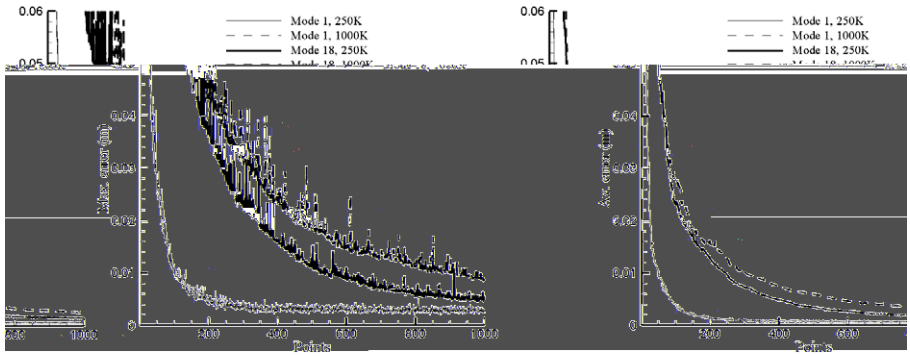
Fig. 6. Convergence of Algorithm 1.

cell mesh with 3881 points on the surface and a 10^6 cell mesh with 11,593 surface points. Moving the 10^6 cell mesh with the basic method of inverting the interpolation matrix on the surface points would be too expensive and demands the improved greedy strategy. The meshes are deformed using three different displacements: a typical flight shape for cruise case three [33] and deflections of both modes 1 and 18 scaled to produce a vertical tip displacement of 2 m. All of these displacements are illustrated in Fig. 3.

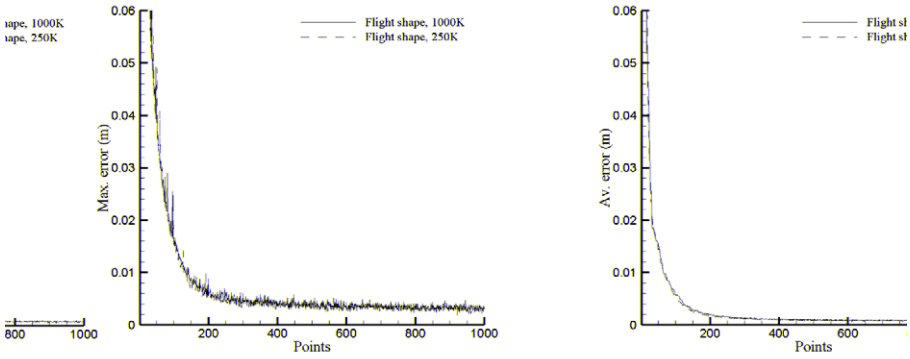
In all cases, linear systems were solved using an unpreconditioned conjugate gradient method implemented in compressed sparse row (CSR) form, started from a zero vector and iterated until the norm of the residual fell to 0.001 of its initial value. The real displacements of the surface were used to drive the greedy point selection.

Convergence of Algorithm 1 is shown against iteration count in Fig. 6 for the three test cases and two mesh densities. The error varies approximately with the reciprocal of the iteration count. An interesting feature is the noise in the decaying signal; this is likely to be due to mutual interference of points within each other's support radius. Convergence of Algorithms 2 and 3, similar in general form, is shown in Figs. 7 and 8. Algorithm 2 always achieves a lower error than Algorithm 1 for the same number of points and both show larger errors for mode 18 as a result of the more complicated shape and higher curvature. A very important feature is that the convergence on both the 250×10^3 and 10^6 cell meshes for both algorithms is almost identical for the mode 1 and flight shape displacements, but appears slower on the 10^6 cell mesh than the 250×10^3 cell mesh for mode 18. The parallel convergence on two different mesh sizes is of importance because it implies that the number of points required to represent a certain deformed geometry may be independent of the number of surface mesh points. A result of this type is crucial as it leads to a mesh motion cost that is only proportional to the number of volume points and independent of the number of surface points. For the more complicated mode 18 shape there is a slower convergence on the 10^6 cell mesh, but this is unlikely to be a serious problem because mode 18 would never participate in any realistic motion with an amplitude as large as 2 m. Since the interpolation is a linear function of the data the error will scale down proportionally.

Convergence of Algorithms 1–3 is compared directly for the mode 18 case on the 10^6 cell mesh in Fig. 9. The behaviour of Algorithm 3, the hybrid method, lies between that of 1 and 2. How closely the performance resembles that of either of the other methods depends, as might be expected, on how many cycles of either Algorithm 1 or 2 are used. Even using only a small number of executions of Algorithm 2 improves the convergence considerably over that of Algorithm 1 alone but represents a fairly small increase in cost. Figs. 11 and 12 illustrate with contour plots the manner in which the surface errors decay as more points are added for both Algorithms 2 and 3; slightly higher errors may be noticed for Algorithm 3 but these



1 and 18 (a) Maximum error, modes 1 and 18 (b) Average error, modes



shape (c) Maximum error, flight shape (d) Average error, flight

Fig. 7. Convergence of Algorithm 2.

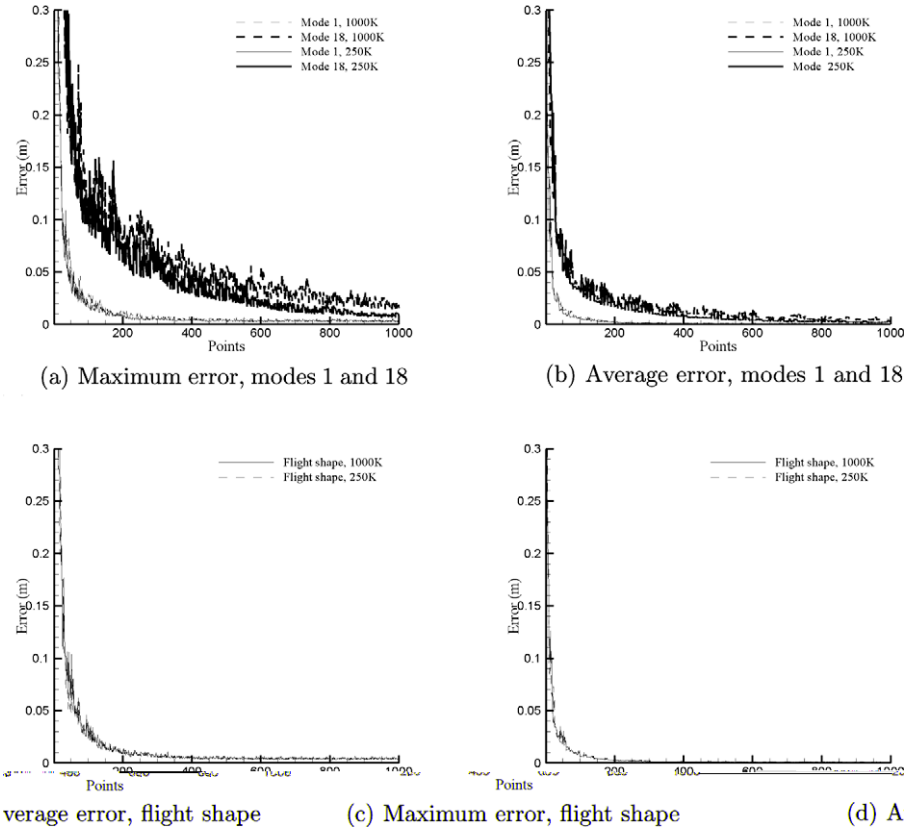


Fig. 8. Convergence of Algorithm 3, using a full update every 10th step.

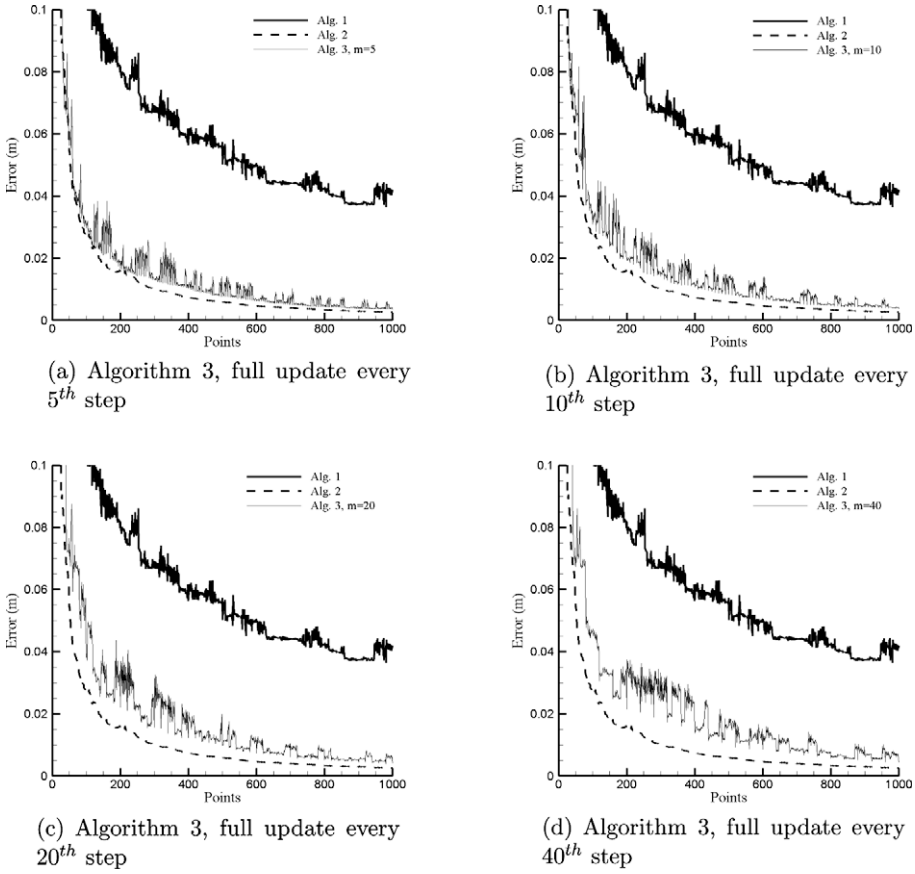
come with the advantage of a faster execution time. Hence, it is possible on the 10^6 cell mesh to achieve maximum errors of less than 0.02% of the semispan (or 0.2% of motion amplitude) with as few as 200 surface points when using the flight shape.

Comparing Figs. 6 and 7 it is clear that using a full solution at each stage improves the compression significantly. Unfortunately while the cost of Algorithm 1 is about $nN_{sp} + N_{sel}^3$ that of Algorithm 2 is near to $\sum_{j=1}^{n} j^3 + jN_{sp}$, where N_{sel} is the number of points selected and n the total number of times to run the loop. These costs are shown in Tables 3 and 4 for the 250×10^3 and 10^6 cell meshes, normalised by the time taken to update the 10^6 cell mesh using the full method with the C0 function and the mode 18 deflection shape. However, every centre that is saved removes $3N_{vp}$ operations from the update every CFD timestep which is potentially a huge saving, so that extra effort reducing the number of centres is normally worthwhile. Table 4 shows that the cost of Algorithm 2 is reasonable up to about 400 points on the 10^6 cell mesh but thereafter becomes large, but it is important to note that the greedy method cost need not be incurred for every update of the mesh if the point selection is frozen before the CFD time stepping begins as shown in Fig. 1.

The cost of the hybrid scheme is $N_{sp}(n - \lfloor \frac{n}{m} \rfloor) + \sum_{j=1}^{j=\lfloor n/m \rfloor} [(mj)^3 + mjN_{sp}]$, where m represents the number of cycles of the loop needed to give one execution of Algorithm 2 (the other executions being of Algorithm 1). This recovers nN_{sp} for $m > n$ (given that $\sum_a^b = 0$ for $b < a$) and $\sum_{j=1}^{n} j^3 + jN_{sp}$ when $m = 1$. The hybrid method therefore allows a good cost compromise whilst preserving reasonable convergence, as illustrated in Fig. 8 and Tables 3 and 4. Tables 3 and 4 also show a comparison to the time taken to solve the same problem fully, revealing that if the full set of 11,593 surface points for the 10^6 cell mesh is used then the update becomes the dominating cost. Also, with so many centres the matrix is poorly conditioned and a less smooth function (Wendland's C0) must be used, whereas the greedy methods can still use the superior Wendland's C2 function with impunity owing to the smaller system size.

The distribution of points selected by Algorithms 1–3 are shown in Fig. 13. Generally Algorithm 2 selects a more even distribution than Algorithm 1, while Algorithm 3 selects an intermediate set of points.

It should be noted that for Algorithm 1 a centre may be visited more than once, therefore the total number of executions of the algorithm is not equal to the total number of points used. The number of points used is a function of the basis function, interpolated function and the data points available. For a run interpolating the 18th mode of the MDO wing the number of points used versus the number of iterations is shown in Fig. 10. The rate of increase of points used drops off with increasing iterations, although if run forever the method would still eventually pick all possible centres, so although the graph appears asymptotic towards a value smaller than the total number of centres this is not the case. Algorithms 2 and 3 add exactly one point every time the loop runs, so that the number of points used always equals the number of executions.



(a) Algorithm 3, full update every 5^{th} step

(b) Algorithm 3, full update every 10^{th} step

(c) Algorithm 3, full update every 20^{th} step

(d) Algorithm 3, full update every 40^{th} step

Fig. 9. Comparison of average error for Algorithms 1–3, using mode 18 shape and 10^6 cell mesh.

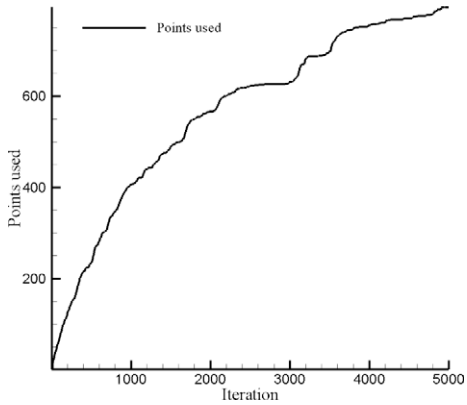


Fig. 10. Number of points used by Algorithm 1, mode 18, 250×10^3 cell mesh.

6. Mesh motion results

In order to assess the efficacy of RBF mesh motion a deflection of the 18th mode shape of the MDO wing has been considered using a 10^6 cell structured multiblock mesh. The mesh was generated with the multiblock generator of Allen [43]. Scaled to have a 2 m deflection at the tip of the structural nodes (double that of the case in Fig. 4), which gives a maximum of 4.5 m at the aerodynamic tip nodes, this represents a difficult case and certainly a harsher one than would normally be encountered in aeroelastic simulation. The surface to mesh interpolation used Wendland's C2 function, with a support radius of twice the mean aerodynamic chord (MAC) and used 200 surface points selected with the hybrid greedy method.

Fig. 14 shows three views of selected grid planes, with the mesh at undeformed, maximum negative tip deflection, and maximum positive tip deflection. Orthogonality is maintained very well, which is attributable to the function used resulting

in greater influence of nearby points. If these mostly rotate (or translate) this motion is naturally propagated gently into the mesh.

6.1. Grid quality

The quality of the above mesh during this deformation has been analysed using a measure of cell orthogonality change. For a constant k plane, for example, the orthogonality at any point in that plane can be defined, following Siebert and Dulikravich [44], using the four neighbour points, as:

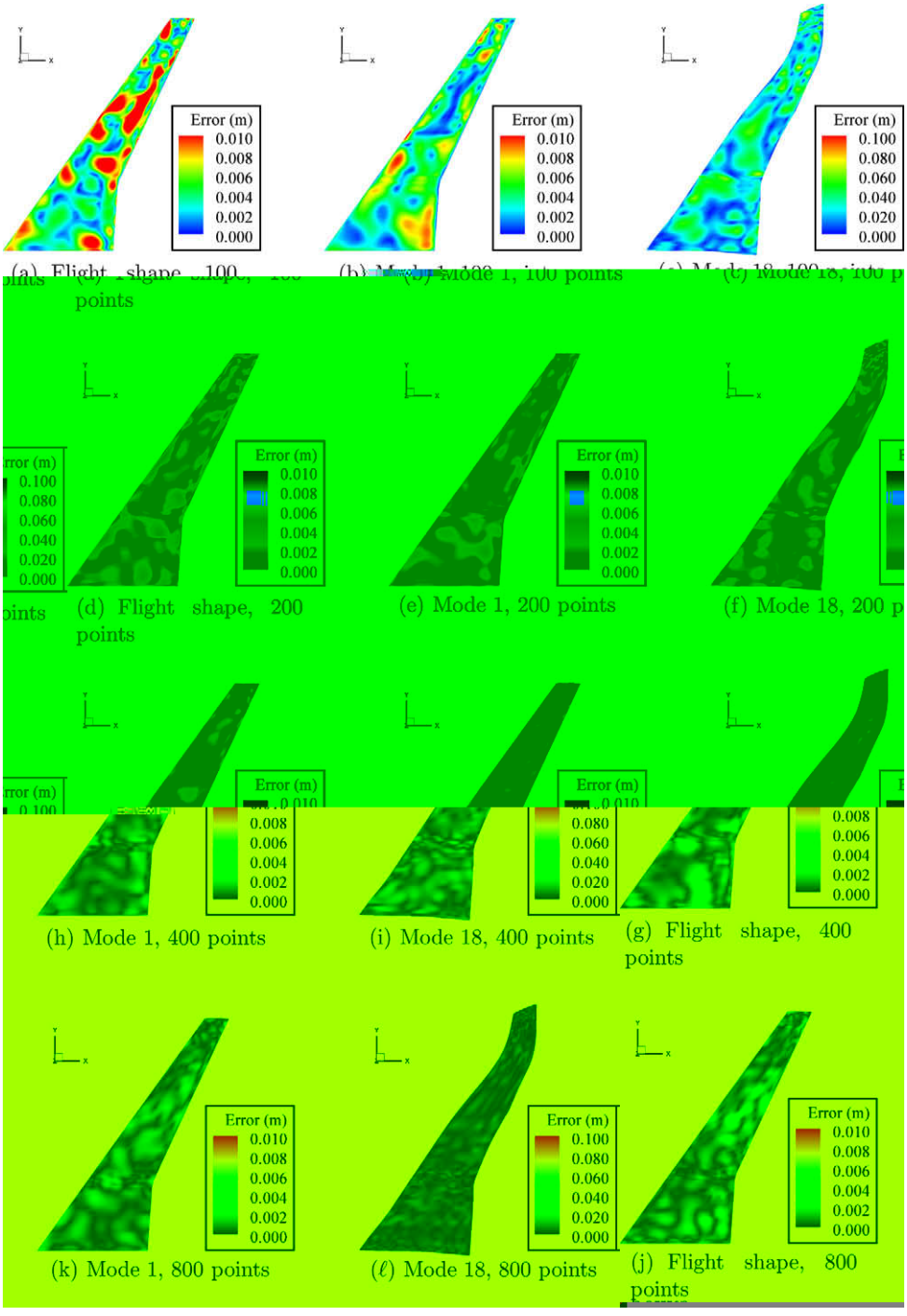
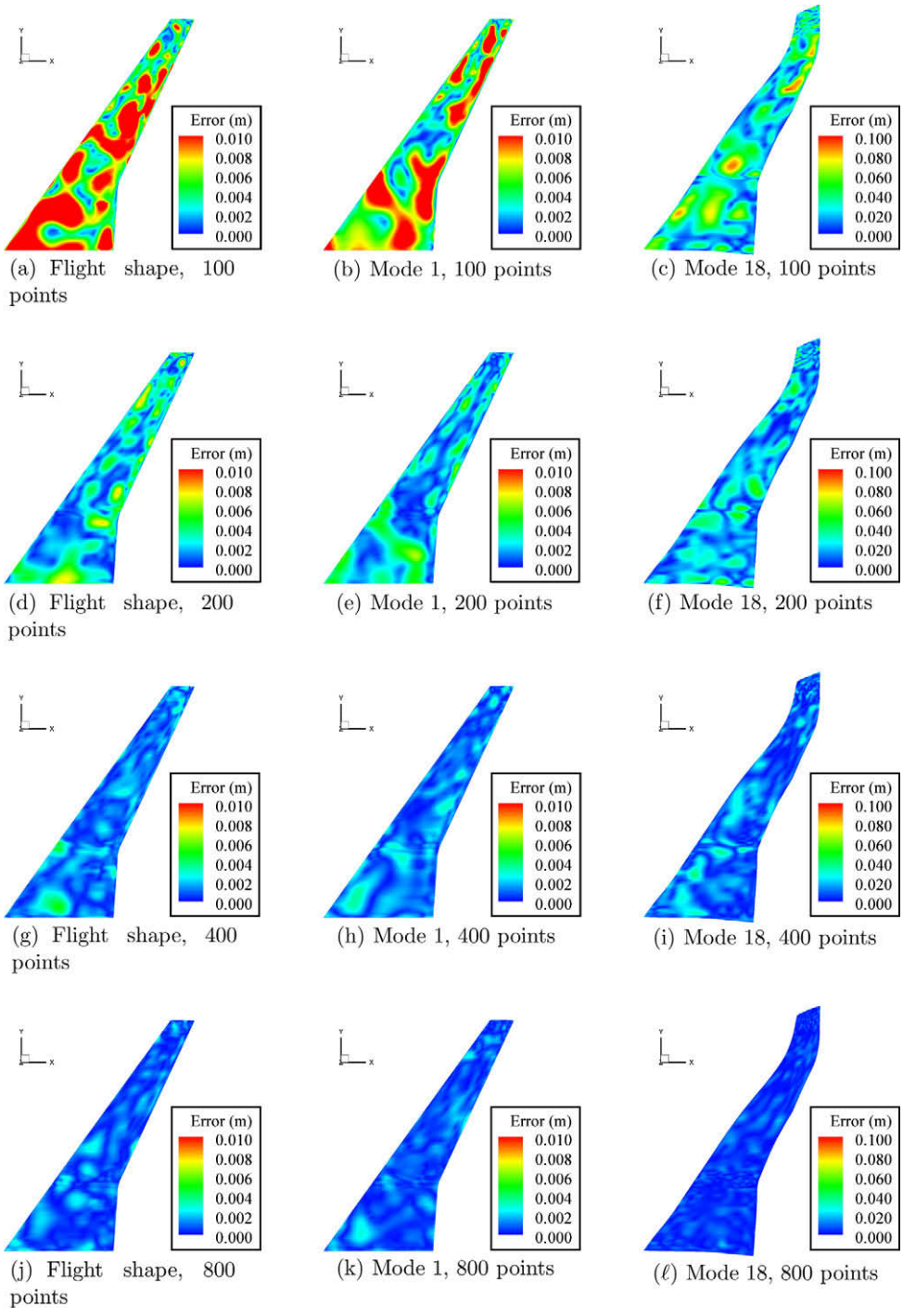


Fig. 11. MDO Wing surface errors using Algorithm 2, 10^6 cell mesh.



(a) Flight shape, 100 points (b) Mode 1, 100 points (c) Mode 18, 100 points

(d) Flight shape, 200 points (e) Mode 1, 200 points (f) Mode 18, 200 points

(g) Flight shape, 400 points (h) Mode 1, 400 points (i) Mode 18, 400 points

(j) Flight shape, 800 points (k) Mode 1, 800 points (ℓ) Mode 18, 800 points

Fig. 12. MDO Wing surface errors using Algorithm 3, 10^6 cell mesh, $m = 10$.

$$\mathbf{v}_1 = \mathbf{r}_{i+1,j,k} - \mathbf{r}_{i,j,k} \tag{29}$$

$$\mathbf{v}_2 = \mathbf{r}_{i,j+1,k} - \mathbf{r}_{i,j,k} \tag{30}$$

$$\mathbf{v}_3 = \mathbf{r}_{i-1,j,k} - \mathbf{r}_{i,j,k} \tag{31}$$

$$\mathbf{v}_4 = \mathbf{r}_{i,j-1,k} - \mathbf{r}_{i,j,k} \tag{32}$$

$$q_k = \frac{1}{4}\left\{\frac{(\mathbf{v}_1\cdot\mathbf{v}_2)^2}{\mathbf{v}_1^2\mathbf{v}_2^2}+\frac{(\mathbf{v}_2\cdot\mathbf{v}_3)^2}{\mathbf{v}_2^2\mathbf{v}_3^2}+\frac{(\mathbf{v}_3\cdot\mathbf{v}_4)^2}{\mathbf{v}_3^2\mathbf{v}_4^2}+\frac{(\mathbf{v}_4\cdot\mathbf{v}_1)^2}{\mathbf{v}_4^2\mathbf{v}_1^2}\right\} \tag{33}$$

Table 3
Computation times, mode 18, 250×10^3 cell mesh, C2 function unless stated, with all, 200, and 400 surface points, $m = 10$. All times normalised by the time required to update the 10^6 cell mesh using full method.

Algorithm	All surface points		200 points		400 points	
	Solve	Update	Solve	Update	Solve	Update
Full (C0 function)	0.060	0.099	–	–	–	–
1	–	–	0.003	0.005	0.018	0.009
2	–	–	0.026	0.005	0.117	0.009
3	–	–	0.005	0.005	0.020	0.009

Table 4
Computation times, mode 18, 10^6 cell mesh, C2 function unless stated, with all, 200, and 400 surface points, $m = 10$. All times normalised by the time required to update the 10^6 cell mesh using full method.

Algorithm	All surface points		200 points		400 points	
	Solve	Update	Solve	Update	Solve	Update
Full (C0 function)	0.201	1.0	–	–	–	–
1	–	–	0.027	0.018	0.043	0.038
2	–	–	0.101	0.018	0.305	0.038
3	–	–	0.025	0.018	0.050	0.038

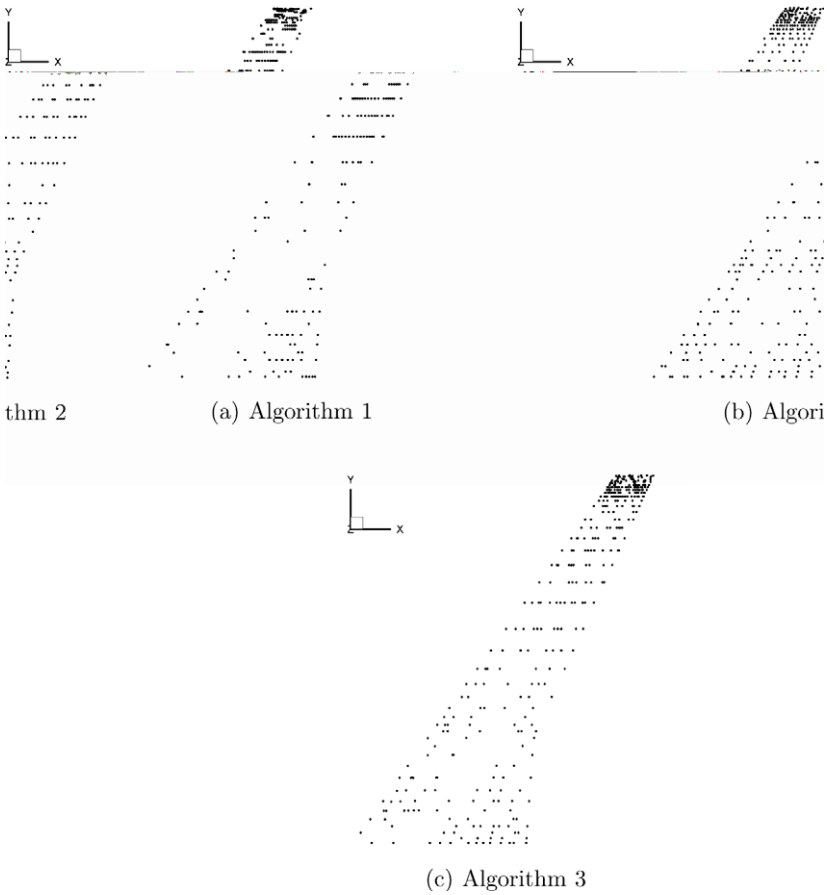


Fig. 13. Point selection for Algorithms 1–3 (with $m = 10$) using 400 points for 18th mode, 250×10^3 cell mesh.

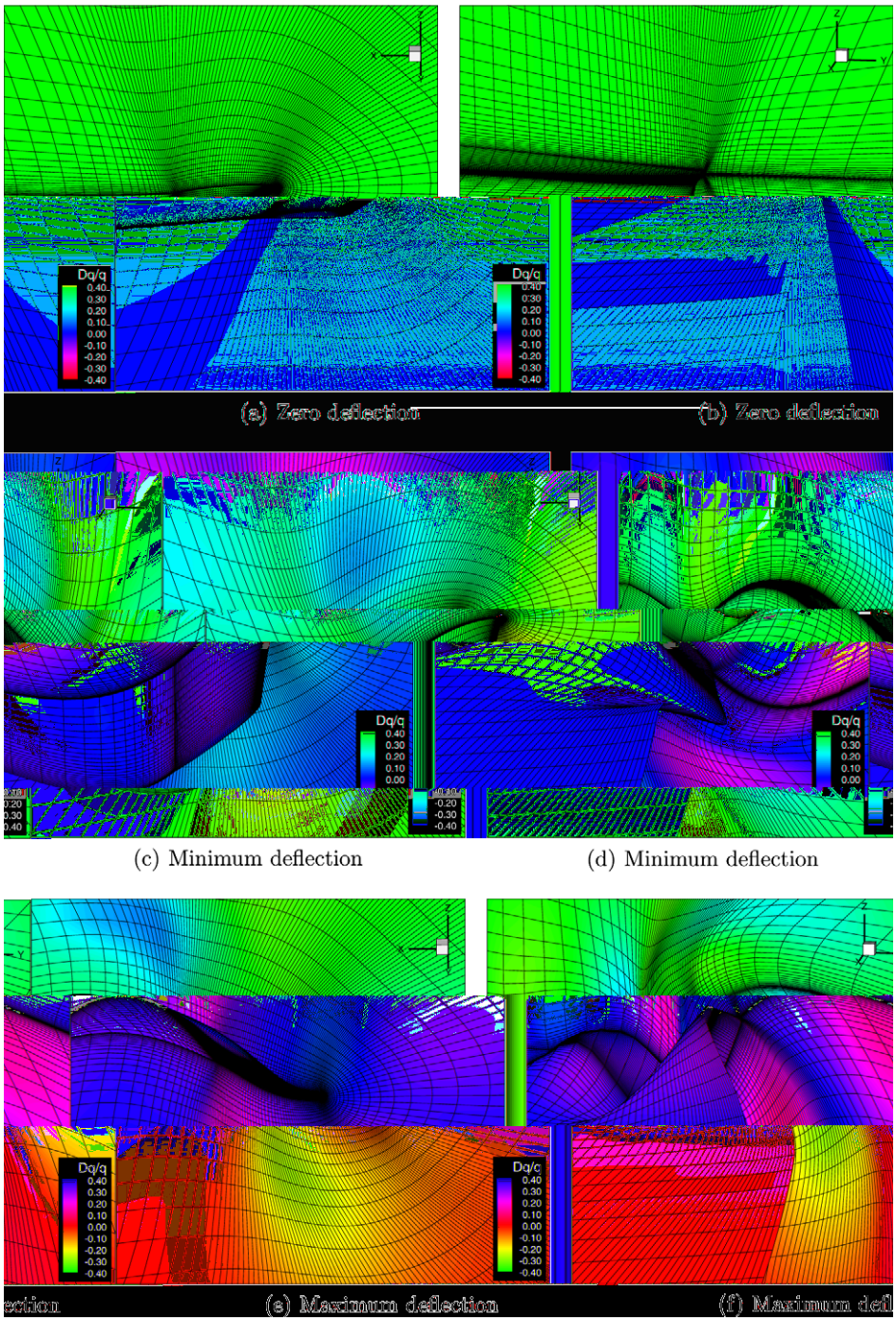


Fig. 14. Mesh quality for 10^6 cell mesh, deformed in 18th mode for MDO wing with tip amplitude of 4.5 m.

Similar arguments can be used for the other two directions, to give a local orthogonality value for each grid point. Orthogonality is often taken as zero to be perfectly orthogonal, i.e. all values of cos (θ) are zero, but it is the relative change in orthogonality that is important for a mesh motion scheme, not the absolute value, and so the value here is reversed to mean 1.0 is perfectly orthogonal.

Since the orthogonality will vary throughout the mesh, it is desirable to isolate the influence of the mesh motion by finding the change in this value, which can then be normalised by the undisturbed value, as shown in Fig. 14. A value of 0 now means that no change in orthogonality has occurred.

The most significant point is, even though there are large changes in orthogonality and volume propagated into the mesh, the orthogonality at and near the surface remains completely unchanged, which is an essential property of an effective mesh motion scheme.

The implications of a motion scheme of this kind for other types of mesh, especially those containing viscous cells, are important. The deformation is entirely independent of the connectivity and shapes of the cells to be deformed; therefore, problems regarding inversion of high aspect ratio viscous cells do not arise because the interpolation does not recognize them as cells, only groups of points. This is a significant difference to pseudo-structural methods such as the spring analogy, which use connectivity to determine the forces on mesh vertices. These methods can struggle on viscous grids because of the severe differences in edge lengths, which are used to produce the spring forces, possibly leading to cell inversion. Edge lengths are never used in this approach and therefore the same problem cannot arise.

7. Conclusions

A very efficient high quality mesh motion method has been presented using radial basis functions. This method is suitable for mesh motion where the surface deformations are smooth (surfaces do not separate or join), as encountered in aeroelastic simulation. The maximum allowable motion is determined by the support radius, which is a numerical parameter that may be chosen by the user. A larger support radius allows a larger motion.

Results have shown that greedy algorithms offer an extremely effective method of cost reduction for this approach. Compared to solving the full system, the most effective greedy method was found to provide a $4\times$ speedup for the solve stage and a $27\times$ speedup for the update (based on a 10^6 cell mesh with 11,593 surface points, 400 of which were used to move the mesh), while in terms of memory the saving is close to $\frac{11,593}{400} = 29\times$. Using only 200 surface points these improvements become $8\times$ for the solve, $55\times$ for the update and $\frac{11,593}{200} = 58\times$ in terms of memory. For the MDO flight shape case, using 200 of the 11,593 surface points resulted in less than a 0.02% of semispan (0.2% of motion amplitude) maximum error in surface displacement over the whole wing, which is particularly impressive. Mesh orthogonality was tested for a large amplitude deformation of the 18th mode for the 10^6 cell mesh and found to be of high quality, particularly close to the surface.

Compression (reduction of the number of surface points) may be most efficiently carried out by Algorithm 2, but this is also the most time consuming method. Algorithm 3 is favoured here, as this combines the cycles of both Algorithms 1 and 2 in an efficient alternating fashion. However, Algorithm 2 may still be preferable if reduction of the surface position error is an absolute priority. Since Algorithm 2 is a simplification of Algorithm 3 both methods may easily be combined and used as desired.

Significantly, for simple deformations such as in mode 1 or a typical flight shape, the number of surface points required to represent the surface to within a certain tolerance at all other points has been shown to be almost independent of the number of surface points. This is extremely significant as it means the cost of the mesh motion may be reduced to be proportional to N_{vp}, plus a constant that depends on the number of surface points required for that geometry but which does not vary with mesh size.

References

[1] T.C.S. Rendall, C.B. Allen, Fluid–structure interpolation and mesh motion using radial basis functions, International Journal for Numerical Methods in Engineering 74 (10) (2008) 1519–1559.
[2] C.B. Allen, T.C.S. Rendall, A unified approach to CFD–CSD interpolation and mesh motion using radial basis functions, in: 25th Applied Aerodynamics Conference, AIAA Paper No. AIAA-2007-3804, Miami, FL, 2007.
[3] A.M. Morris, C.B. Allen, T.C.S. Rendall, Development of generic CFD-based aerodynamic optimisation tools for helicopter rotor blades, in: 25th Applied Aerodynamics Conference, AIAA Paper No. AIAA-2007-3809, Miami, FL, 2007.
[4] A.M. Morris, C.B. Allen, T.C.S. Rendall, Domain element paramterisation for CFD-based optimisation of aerofoils using deformation by radial basis functions, International Journal for Numerical Methods in Fluids 58 (8) (2008) 827–860.
[5] C.B. Allen, Aeroelastic computations using algebraic grid motion, The Aeronautical Journal 106 (1064) (2002) 559–570.
[6] D.P. Jones, A.L. Gaitonde, S.P. Fiddes, Moving mesh generation for unsteady flows about deforming complex configurations using multiblock meshes, CFD Journal, Japanese Society of CFD (2001) 430–439.
[7] J.T. Batina, Unsteady Euler algorithm with unstructured dynamic mesh for complex-aircraft aerodynamic analysis, AIAA Journal 29 (3) (1991) 327–333.
[8] K.P. Singh, J.C. Newman III, O. Baysal, Dynamic unstructured method for flows past multiple objects in relative motion, in: 32nd Aerospace Sciences Meeting and Exhibit, AIAA Paper No. AIAA-94-0058, Reno, NV, 1994.
[9] C. Farhat, C. Degand, B. Koobus, M. Lesoinne, Torsional springs for two-dimensional dynamic unstructured fluid meshes, Computational Methods in Applied Mechanics and Engineering 163 (1998) 231–245.
[10] F. Blom, Considerations on the spring analogy, International Journal for Numerical Methods in Fluids 32 (2000) 647–668.
[11] E.F. Sheta, H.Q. Yang, S.D. Habchi, Solid brick analogy for automatic grid deformation for fluid–structure interaction, in: 36th AIAA Fluid Dynamics Conference and Exhibit, AIAA Paper No. AIAA-2006-3219, San Francisco, CA, 2006.
[12] R. Loehner, C. Yang, Improved ALE mesh velocities for moving bodies, Communications in Numerical Methods in Engineering 12 (1996) 599–608.
[13] J.D. Bau, H. Luo, R. Loehner, E. Goldberg, A. Feldhun, Application of unstructured moving body methodology to the simulation of fuel tank separation from an F-16 fighter, in: 35th Aerospace Sciences Meeting and Exhibit, AIAA Paper No. AIAA-1997-0166, Reno, NV, 1997.
[14] H. Jasak, Z. Tukovic, Mesh motion for the unstructured finite volume method, Transactions of FAMENA 30 (2) (2007).
[15] L. Tysell, Grid deformation of 3D hybrid grids, in: Proceedings of the eighth International Conference on Numerical Grid Generation in Computational Field Simulations, 2002.
[16] B. Helenbrook, Mesh deformation using the biharmonic operator, International Journal for Numerical Methods in Engineering 56 (2003) 1007–1021.
[17] R. Melville, Nonlinear simulation of F-16 aeroelastic instability, in: 39th Aerospace Sciences Meeting and Exhibit, AIAA Paper No. AIAA-2001-0570, Reno, NV, 2001.
[18] R. Melville, Dynamic Aeroelastic Simulation of Complex Configurations using Overset Grid Systems, AIAA Paper No. AIAA-2000-2341, 2000.

[19] C.B. Allen, Parallel universal approach to mesh motion and application to rotors in forward flight, International Journal for Numerical Methods in Engineering 69 (10) (2007) 2126–2149.
[20] P. Cizmas, J. Gargoloff, Mesh generation and deformation algorithm for aeroelasticity simulations, in: 45th Aerospace Sciences Meeting, AIAA Paper No. AIAA-2007-556, Reno, NV, 2007.
[21] X. Liu, N. Qin, H. Xia, Fast dynamic grid deformation based on delaunay graph mapping, Journal of Computational Physics 211 (2006) 405–423.
[22] A. Beckert, H. Wendland, Multivariate interpolation for fluid–structure-interaction problems using radial basis functions, Aerospace Science and Technology 5 (May–June) (2001) 125–134.
[23] R. Ahrem, A. Beckert, H. Wendland, A meshless spatial coupling scheme for large-scale fluid–structure interaction problems, Computer Modeling in Engineering and Sciences 12 (2006) 121–136.
[24] M. Buhmann, Radial Basis Functions, first ed., Cambridge University Press, 2005.
[25] H. Wendland, Scattered Data Approximation, first ed., Cambridge University Press, 2005.
[26] S.P. Sprekreijse, B.B. Prananta, J.C. Kok, A Simple, Robust and Fast Algorithm to Compute Deformations of Multiblock Structured Grids, Technical Report NLR-TP-2002-105, NLR, 2002.
[27] M.H.L. Hounjet, J.J. Meijer, Evaluation of Elastomechanical and Aerodynamic Data Transfer Methods for Non-planar Configurations in Computational Aeroelastic Analysis, Technical Report NLR-TP-95690U, NLR, 1994.
[28] B.B. Prananta, J.J. Meijer, Transonic Static Aeroelastic Simulations of a Fighter Aircraft, Technical Report NLR-TP-2003-187, NLR, 2003.
[29] A. de Boer, M.S. dan der Shoot, H. Bijl, Mesh deformation based on radial basis function interpolation, Computers and Structures 85 (2007) 784–795.
[30] A.H. van Zuijlen, A. de Boer, H. Bijl, Higher order time integration through smooth mesh deformation for 3D fluid–structure interaction simulations, Journal of Computational Physics 224 (2007) 414–430.
[31] S. Jakobsson, O. Amoignon, Mesh Deformation Using Radial Basis Functions for Gradient Based Aerodynamic Shape Optimization, Technical Report FOI-R-1784-SE, FOI, December 2005.
[32] M.R. Hestenes, E. Stiefel, Methods of conjugate gradients for solving linear systems, Journal of Research of the National Bureau of Standards 49 (1952) 409–436.
[33] D. Haase, V. Selmin, B. Winzell, Notes on Numerical Fluid Mechanics and Multidisciplinary Design, Progress in Computational Flow–Structure Interaction, first ed., vol. 81, Springer, 2002.
[34] R. Schaback, H. Wendland, Adaptive greedy techniques for approximate solution of large RBF systems, Numerical Algorithms 24 (2000) 239–254.
[35] Y. Ohtake, A. Belyaev, H. Seidel, Multi-scale and adaptive CS-RBFs for shape reconstruction from cloud of points, in: MINGLE workshop on Multiresolution in Geometric Modelling, Cambridge, UK, September 2003, pp. 337–348. Available at: <http://citeseer.ist.psu.edu/ohtake03multiscale.html>.
[36] R. Schaback, H. Wendland, Numerical techniques based on radial basis functions, in: Albert Cohen, Christophe Rabut, Larry Schumaker (Eds.), Curve and Surface Fitting, Vanderbilt University Press, Nashville, TN, 2000. Available at: <http://citeseer.ist.psu.edu/schaback00numerical.html>.
[37] H. Wendland, Fast evaluation of radial basis functions: methods based on partition of unity, Approximation Theory X: Wavelets Splines, and Applications, Vanderbilt University Press, Nashville, Texas, USA, 2002. pp. 473–483.
[38] A. Iske, J. Levesley, Multilevel scattered data approximation by adaptive domain decomposition, Numerical Algorithms 39 (July) (2005) 187–198.
[39] S. De Marchi, On optimal locations for radial basis function interpolation: computational aspects, Rendiconti Del Seminario Matematico 63 (3) (2003) 343–357.
[40] S. De Marchi, R. Schaback, H. Wendland, Near-optimal data-independent point locations for radial basis function interpolation, Advances in Computational Mathematics 23 (2005) 317–330.
[41] J.C. Carr, R.K. Beatson, J.B. Cherrie, T.J. Mitchell, W.R. Fright, B.C. McCallum, T.R. Evans, Reconstruction and representation of 3D objects with radial basis functions, Proceedings of the ACM SIGGRAPH Conference on Computer Graphics (2001) 67–76.
[42] S. Allwright, Multi-discipline optimisation in preliminary design of commercial transport aircraft, in: J.-A. Desideri, C. Hirsch, P. Le Tallec, E. Onate, M. Pandolfi, J. Periaux, E. Stein (Eds.), Computational Methods in Applied Sciences, ECCOMAS, Wiley, 1996, pp. 523–526.
[43] C.B. Allen, Towards automatic structured multiblock mesh generation using improved transfinite interpolation, International Journal for Numerical Methods in Engineering 75 (4) (2007) 697–733.
[44] B.W. Siebert, G.S. Dulikravich, Grid generation using a posteriori optimization with geometrically normalised functions, in: Eighth Applied Aerodynamics Conference, AIAA Paper No. AIAA-1990-3048, Portland, OR, 1990.