ABSTRACT

KASHI, ADITYA. Techniques for Mesh Movement and Curved Mesh Generation. (Under the direction of Hong Luo.)

This report describes methods of mesh movement for computational fluid dynamics (CFD) problems. The methods described can be classified into two families - elasticity-based methods and interpolation methods. The former class includes spring analogy methods, and elasticity methods. Interpolation methods described here include the 'Delaunay graph' mapping, interpolation by radial basis functions (RBF), and a combination of Delaunay graph mapping and radial basis function interpolation. The merits and disadvantages of these methods have been discussed, and some two-dimensional test cases have been presented. Further, a method for generating curved meshes from linear meshes is presented in this work. The method is designed to generate a curved mesh for any type of grid in a computationally inexpensive and numerically robust manner. In this method, high-order nodes are first placed in the linear mesh, the high-order boundary nodes are then relocated to the true boundary, and finally, the high-order interior nodes are moved using one of the mesh-movement methods, based on the motion of the high-order boundary nodes. The developed method is used to generate curved meshes for a number of geometries. The numerical experiments indicate that the RBF-interpolation method is much better than the prevalent linear-elasticity methods for generating high-order curved meshes judging by curved-mesh quality, computing cost, and ease of implementation.

Techniques for Mesh Movement and Curved Mesh Generation

by
Aditya Kashi

A thesis submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the Degree of
Master of Science

Mechanical Engineering

Raleigh, North Carolina

2016

APPROVED BY:

Jack R. Edwards Jr.	Pierre Gremaud

Hong Luo
Chair of Advisory Committee

DEDICATION

To my parents.

BIOGRAPHY

The author was born in Kasauli, HP, India and lived throughout his childhood in Delhi, India. He completed his Bachelor degree in Mechancial Engineering and integrated Masters degree in Biological sciences from Birla Institute of Technology and Science, Pilani, India. During this time, he completed internships related to computational engineering in Indian Institute of Science, Bangalore, and Centre for Fuel Cell Technology, International Advanced Research Centre for Powder Metallurgy and New Materials, Chennai, India. In August 2014, he began the MS Mechanical Engineering program at NC State University, and went on to start work at the Computational Fluid Dynamics Laboratory under the guidance of Professor Hong Luo.

ACKNOWLEDGEMENTS

First and foremost, I would like express my heartfelt thanks to my advisor Professor Hong Luo for guiding and advising me on various issues throughout my time at NC State. Next, I thank my committee members Professor Jack Edwards Jr and Professor Pierre Gremaud, for teaching me important concepts and agreeing to be on my MS defense committee. I am also thankful to my colleagues at the CFD lab - Aditya Pandare, Xiaodong Liu, Chuanjin Wang, Chad Rollins, Jialin Lou, Dr Xiaoquan Yang and Dr Jian Cheng for always entertaining my questions and discussing things when required. I am especially thankful to Xiaodong for running RDGFLO on some of my meshes, many interesting discussions and also the rides home. Finally, I am very thankful to my parents for being very supportive and for encouraging me to do my best.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

Moving meshes are required in various problems in computational fluid dynamics (CFD), such as aeroelasticity, aerodynamic shape optimization [14] and some kinds of multi-phase/multi-material flow simulations. Mesh movement may also be used in generating curved meshes for spatially high-order computational methods [23]. In general, good properties of a mesh movement scheme are as follows.

1. It should be robust. In many kinds of meshes, such as meshes required for viscous flow computations, even small boundary movements can invalidate the mesh. The scheme should be able to preserve mesh validity, at the very least.

2. Mesh quality should be preserved to a great extent. Elements should not become highly skewed or mis-shaped after movement, as this can impact flow computations on the deformed mesh. We discuss skew and shape of elements later in the report.

3. It should be computationally inexpensive. Many applications need very fast mesh movement schemes as they require the mesh to be moved many times during the simulation, possibly every time step.

Some mesh movement schemes that we present here satisfy only one or two of the above criteria. Our aim is to find a scheme that satisfies all three criteria well. We give a review of various mesh-movement methods in chapter 2.

With the rising popularity of high-order computational methods in the aerospace community, robust techniques to obtain a high-order representation of the geometry, ie., the boundary of the domain, have become important. High-order approximation of the boundary is required to attain high-order accuracy [22], and in some cases it may be crucial to even preserve qualitative correctness of the flow [2]. One technique of high-order boundary approximation is to produce high-order meshes, otherwise called curved meshes. Another technique in this regard

is isogeometric analysis [5] where CAD data is directly used in analysis. In this work, we deal with curved unstructured mesh generation, using either CAD data if available, or only the linear mesh data. If only linear mesh data is available, some kind of high-order boundary reconstruction is used to compute a higher order approximation, and the mesh is then curved according to this reconstructed boundary. In chapter 3, we describe the method of curved mesh generation adopted for this work.

Results for both mesh movement and curved mesh generation are presented in chapter 4. We conclude in chapter 5 with a comparison of the methods and give some more details of our implementation in the appendices.

Chapter 2

Review of mesh movement methods

In our knowledge, there are, broadly speaking, two types of methods to achieve movement of the interior nodes knowing the movement of the boundary nodes. The first type uses models based on mechanics to try to achieve valid mesh movement. Examples of this type are lineal spring analogy [3], torsion spring analogy [7], linear-elasticity (and variants thereof)[11] and non-linear elasticity [23]. Each of these requires solution of a system of equations of size proportional to the number of mesh points to be moved. The linear-elasticity based models require solution of Poisson-type partial differential equations.

The second type of technique to move the interior nodes involves *interpolation* of the boundary displacement to interior nodes. Several methods exist; some examples are radial basis function (RBF) interpolation [6], 'explicit' interpolation [21], and 'Delaunay graph mapping' [19]. These methods do not involve any physics-based modeling of the mesh.

2.1 Elasticity methods

Elasticity-based methods are among the oldest mesh-movement methods in use. These include lineal spring analogy, torsional spring analogy, and various kinds of elastic solid mechanics analogies. These methods model the mesh as some kind of solid entity, impose a boundary displacement and solve equilibrium equations to propagate the motion to the interior nodes.

2.1.1 Spring analogies

Lineal spring analogy is the simplest method that can be used to move internal mesh points in response to movement of the boundary, invented by J.T. Batina. Each edge of the mesh is assumed to represent a lineal spring connecting the two nodes which make up the edge. The idea is that after imposing boundary movement, the mesh is allowed to "go to equilibrium". At

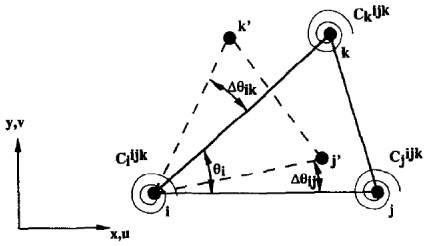

Figure 2.1: Movement of an element in torsion spring analogy; from [7]

every mesh point, we have

$$\sum_j k_{ij}(\Delta\mathbf{r}_i - \Delta\mathbf{r}_j) = \mathbf{0} \quad \forall i \tag{2.1}$$

where i ranges over all nodes, j ranges over points surrounding node i and $\Delta\mathbf{r}_i$ is the displacement of node i. k_{ij} is the stiffness of the spring between nodes i and j, which can be taken as

$$k_{ij} = \frac{1}{||\mathbf{r}_i - \mathbf{r}_j||}. \tag{2.2}$$

This method is not reliable as it generates invalid elements (having zero or negative value of the Jacobian determinant of the reference-to-physical element mapping) for even relatively small displacements, depending on the mesh. However, for the application it was developed, it sometimes gives acceptable results. It is used, for example, by Mavriplis *et. al.* [27] in an optimization problem involving inviscid flow only. It is also less computationally expensive compared to other elasticity-based methods.

Note that separate problems are solved in each coordinate direction, which are uncoupled. It is the author's opinion that this is the main reason for its unreliability for many kinds of mesh and movement combinations. However, because of this, extension to 3D is trivial.

The torsional spring analogy is an extension to the previous method, which adds torsional springs at each node in each element (see Farhat *et.al.* [7]). These torsional springs resist the relative angular motion between edges of an element. This leads to a substantially more robust movement. In the scheme of Farhat *et. al.*, the problem is divided into two parts, one related to lineal spring analogy and one to the torsional spring analogy. The lineal spring model is not the same as that used by Batina; in this case the lineal spring model also leads to a coupled system which is more robust than the uncoupled scheme. The torsional part adds even more resistance to invalid elements. As shown in figure 2.1, the torsional spring analogy adds torsion springs at each node of each element. The stiffness of a torsion spring associated with a node of a certain element is a function of the angle formed by the edges of that element meeting at the node.

While this method is significantly more robust than the lineal spring analogy of Batina, it is also much more computationally expensive. A coupled system of $n_{dim}N_n$ equations must be solved, where N_n is the number of nodes that are to be moved in the mesh and n_{dim} is the physical spatial dimension of the problem.

2.1.2 Elastic solid analogy ('Elasticity')

The deformable elastic solid analogy, employing the equations of (linear or non-linear) elasticity, is one of the widely-used mesh movement techniques. This class of schemes often lead to very robust mesh movement. The mesh is assumed to model a deformable solid body, which is then deformed according to the equations of solid mechanics, that is, linear or non-linear elasticity.

Linear elasticity

The simplest approach is linear elasticity.

$$\nabla \cdot \sigma = \mathbf{0} \quad \text{in}\,\Omega \tag{2.3}$$

Assuming an isotropic material, the constitutive relation can be taken as

$$\boldsymbol{\sigma} = 2\mu\boldsymbol{\epsilon} + \lambda(\text{tr}\boldsymbol{\epsilon})\boldsymbol{I} \tag{2.4}$$

where $\boldsymbol{\sigma}$ is the stress and $\boldsymbol{\epsilon}$ is the strain. Finally, the linear strain-displacement relation is given by

$$\boldsymbol{\epsilon} = \frac{1}{2}(\nabla\boldsymbol{u} + \nabla\boldsymbol{u}^T) \tag{2.5}$$

where $\boldsymbol{u}$ is the displacement field, with Dirichlet boundary conditions (prescribed boundary displacement $\boldsymbol{u}_b$)

$$\boldsymbol{u} = \boldsymbol{u}_b \quad \text{on}\,\partial\Omega \tag{2.6}$$

The weak form of this set of equations, solved by Galerkin finite element method (FEM) leads to a linear system of size n_dN_n, where n_d is the dimension of the problem and N_n is the number of nodes to be moved.

'Sitffened' linear elasticity

While the basic linear elasticity scheme is adequate for some applications, it is not adequate for stretched, boundary-layer meshes for viscous flow, among others. The scheme is often modified by 'stiffening' the mesh appropriately, so as to attain some control over the propagation of deformation into the interior of the mesh, as done, for instance, by Hartmann and Leicht [11].

In their work, the material is stiffened based on the determinant of the local Jacobian matrix of the reference-to-physical mapping, that is to say smaller elements are stiffer than larger ones. If $\hat{\kappa}$ is the reference element corresponding to the physical element κ, the local bilinear form on κ giving rise to the local stiffness matrix

$$(\dots)_\Omega = \sum_{\kappa \in T_h} \int_\kappa \dots d\mathbf{x} = \sum_{\kappa \in T_h} \int_{\hat{\kappa}} \dots J_\kappa d\mathbf{x} \tag{2.7}$$

is modified according to

$$\int_{\hat{\kappa}} \dots J_\kappa d\mathbf{x} \quad \text{becomes} \int_{\hat{\kappa}} \dots J_\kappa \left(\frac{J_0}{J_\kappa}\right)^\chi d\mathbf{x} \tag{2.8}$$

where J_κ is the Jacobian of the reference-to-physical mapping of the element κ, J_0 is a reference Jacobian for the mesh and χ is a constant to be chosen.

We could also use other stiffening criteria. In our work we have tried stiffening based on the shape quality metric (section 2.3), but we find that the shape metric generally leads to unacceptable results, while the size-shape metric is worse than Jacobian-based stiffening. Perhaps some other metric could be formulated that works better. Further, we could stiffen cells based on distance from boundaries. This would require a fast, and not necessarily accurate, boundary-distance estimation.

Nonlinear elasticity

Nonlinear elasticity is claimed to be a highly robust method for mesh movement by Persson and Peraire [23]. As long as the mesh is fine enough to resolve the displacements, they claim that element validity is guaranteed. In their work, for non-linear elasticity, the constitutive equation (2.4) and strain-displacement relation (2.5) are replaced by the 'neo-Hookean' constitutive model

$$\boldsymbol{P} = \mu((\boldsymbol{F}^T\boldsymbol{F})\boldsymbol{F}^{-T} - \boldsymbol{F}^{-T}) + \lambda(\ln\det\boldsymbol{F})\boldsymbol{F}^{-T} \tag{2.9}$$

where the deformation gradient $\boldsymbol{F}$ is given by

$$\boldsymbol{F} = \frac{\partial \boldsymbol{x}}{\partial \boldsymbol{X}}.$$

Here, $\boldsymbol{x}$ is the physical position vector of a point with coordinate $\boldsymbol{X}$ in the reference configuration. The system is solved using Newton-GMRES iterations.

In our opinion, this method probably cannot be used in any unsteady moving-geometry simulations, as the cost will be prohibitive. However, it leads to good results for curved mesh generation, as can be seen in Persson and Peraire's work.

The advantage of elasticity-based methods is that they can be made highly robust. Their disadvantage is that generally, the more robust the scheme, the more expensive it is. The cost of the system of equations that needs to be solved usually scales with the total number of mesh nodes to be moved; this can get very expensive for 3D viscous meshes. Further, the implementation is usually not very easy; different cell types require different treatments (different basis functions, for instance, in case of elastic-solid approaches) and extension from 2D to 3D may not be trivial (as in case of torsional springs).

2.2 Interpolation Methods

Also called algebraic methods, these methods do not involve any physics-based modeling of the mesh. They are purely mathematical manipulations of the mesh. Unlike elasticity-based methods, schemes of this type tend to be fast and independent of mesh topology.

2.2.1 Delaunay graph mapping (DGM)

Developed by Liu, Qin and Xia [19], DGM is a fast method of mesh movement. It consists of the following steps.

- A Delaunay triangulation is constructed from the boundary points of the mesh. This triangulation is referred to as the Delaunay graph.
- Each internal mesh point is located in the Delaunay graph, and its barycentric coordinates are calculated with respect to the Delaunay element it lies in.
- Using the prescribed boundary displacements, the Delaunay graph is moved.
- Using the calculated barycentric coordinates, the interior mesh points are mapped back to the deformed Delaunay elements keeping the barycentric coordinates constant, thus moving the mesh.

Thus, no linear system needs to be solved. The bulk of the work is in computing the Delaunay triangulation and traversing it, which can be done efficiently. We use the Bowyer-Watson algorithm to compute the Delaunay tessellation of the boundary points in both 2D and 3D [4]. Details of the exact algorithm used are provided in appendix ******A*******.

This method results in a valid mesh as long as the boundary displacement does not invalidate the Delaunay graph itself. If the Delaunay graph becomes invalid, the displacement must be carried out in several steps. The Delaunay graph is re-computed each step, and thus much more boundary movement is possible without invalidating the sequence of Delaunay graphs.

In our experiments, Delaunay graph mapping is not very robust when it comes to rotational motion, or otherwise very large deformation (see the results section).

2.2.2 Radial basis functions

Radial basis functions (RBFs) are radially-symmetric functions. They are used as a basis for the displacement field in the moving mesh [6]. Considering n_b boundary points in the mesh, we express the displacement as

$$\mathbf{s}(\mathbf{x}) = \sum_{j=1}^{n_b} \mathbf{a}_j \phi(\|\mathbf{x} - \mathbf{x}_{bj}\|) \tag{2.10}$$

where $\mathbf{s}$ represents the displacement field, $\mathbf{x}$ is the position vector of any point in the domain, $\mathbf{x}_{bj}$ is the position of the ith boundary point, ϕ is a radial basis function, $\phi(\|\mathbf{x} - \mathbf{x}_{bj}\|)$ is the jth basis function for the displacement field, and $\mathbf{a}_j$ is the jth coordinate or coefficient in that basis.

Since we know the displacements of the boundary nodes, we can solve for the coefficients $\mathbf{a}_j$ using

$$\mathbf{s}(\mathbf{x}_{ib}) = \sum_{j=1}^{n_b} \mathbf{a}_j \phi(\|\mathbf{x}_{bi} - \mathbf{x}_{bj}\|). \tag{2.11}$$

In each coordinate direction, this leads to a system of n_b equations in n_b unknowns:

$$\mathbf{A}\mathbf{a}_k = \mathbf{s}_k \tag{2.12}$$

where the (i, j) component of $\mathbf{A}$ is $\phi(\|\mathbf{x}_{bi} - \mathbf{x}_{bj}\|)$, $\boldsymbol{a}_k \in \mathbb{R}^{n_b}$ is the coefficient vector in the kth direction and $\boldsymbol{s}_k \in \mathbb{R}^{n_b}$ is the vector of boundary displacements in the kth coordinate direction.

In our implementation, we assemble $\mathbf{A}$ as a sparse matrix by evaluating the radial basis function between every pair of boundary points. This process can be parallelized easily.

Next, we need to evaluate equation (2.10) for each interior point, which requires one evaluation of the RBF for each interior point (in each direction). This too, can be parallelized easily.

The quality of the final mesh depends on which RBF is used. Many kinds of RBFs have been used in literature [6, 24]. In this work, we use Wendland's C2 function

$$\phi(x) = (1 - x)^4(4x + 1), \tag{2.13}$$

or rather, a modification,

$$\phi(x) = \begin{cases} \left(1 - \frac{x}{r_s}\right)^4 \left(4\frac{x}{r_s} + 1\right) & x < r_s \\ 0 & x \geq r_s \end{cases} \tag{2.14}$$

where r_s is a real number called the 'support radius'. This modified function has a compact

support. In light of this, we see the matrix $\mathbf{A}$ is sparse if the support radius is less than the characteristic dimension of the domain. For curved mesh generation, this support radius can be kept quite small relative to the whole domain, giving us a very sparse linear system which is solved quite fast.

The scaling of the argument by the support radius serves to make the value of the RBF 1.0 at the boundary. This means that points very close to the boundary will deform just like the boundary itself. This results in an essentially rigid motion of points very close to the boundary, which causes the near-boundary elements to retain their quality after the mesh movement. The support radius is chosen big enough to accommodate the expected deformation of the boundary, but small enough that the linear system is sparse. An algorithm to determine a good support radius will be very useful in this regard.

The RBF method can be carried out in multiple steps; this is found to increase the quality of generated meshes in some cases involving large rotational deformations (4.8), but not so much in others, such as curved mesh generation.

2.2.3 Interpolation using radial basis function on the Delaunay graph (DGRBF)

This method proposed by Wang, Qin and Zhao [28] combines both the Delaunay graph mapping and interpolation by RBF. The general scheme of the mesh movement is the same as in case of DGM, but with the important difference that interpolation is not done using barycentric coordinates of the nodes with respect to the Delaunay graph elements. Here, the interpolation is done via radial basis functions in each Delaunay graph element. The displacement of a node with initial position $\boldsymbol{x}$ is given by

$$\mathbf{s}(\mathbf{x}) = \sum_{j=1}^{n_t} \mathbf{a}_j \phi(\|\mathbf{x} - \mathbf{x}_{tj}\|) \tag{2.15}$$

where $\boldsymbol{x}_{tj}$ are positions of nodes of the Delaunay simplex that contains the node at $\boldsymbol{x}$, and n_t is the number of nodes in that simplex (3 in 2D and 4 in 3D). We need to solve for the $\boldsymbol{a}_j$, the RBF coefficients, by solving a 3×3 system in 2D or a 4×4 system in 3D, in each Delaunay element.

$$\mathbf{s}(\mathbf{x}_{ti}) = \sum_{j=1}^{n_t} \mathbf{a}_j \phi(\|\mathbf{x}_{ti} - \mathbf{x}_{tj}\|). \tag{2.16}$$

DGRBF method provides flexibility in choosing the RBF and the support radius, and with a good choice of these, robust mesh movement can be achieved for translational motion and possibly some other kinds of deformation. It has been observed in [28] and in our tests, that interpolation of displacements by DGRBF often does not give very good results for large rotational deformations. The remedy for this is to interpolate the rotational angles instead of the

displacements directly, and then calculate the displacements at each interior node using these interpolated rotation angles using the rotation matrix. For example, in 2D,

$$x_{new} = (x - x_0)\cos a_z - (y - y_0)\sin a_z + x_0 \tag{2.17}$$

$$y_{new} = (x - x_0)\sin a_z + (y - y_0)\cos a_z + y_0 \tag{2.18}$$

where (x_0, y_0) is the center of rotation, and a_z is the interpolated rotation angle at the node in question. For interpolation of the angles, the same formulae (2.15) and (2.16) are used, with the rotation angles replacing the displacements. Angle interpolation can be combined with displacement interpolation for problems involving both translation and rotation. This angle interpolation scheme, and its combination with the displacement interpolation scheme, is referred to as 'DGRBF2' in [28].

2.2.4 'Improved' Delaunay graph mapping

If the Delaunay graph (DG) lines are not approximately normal to the boundary, there can sometimes be issues with mesh movement. Mesh lines tend to be normal to the boundary for physical and numerical reasons. If Delaunay graph lines make very small or very large angles (compared to 90 degrees) with the boundary, successive interior nodes on the same mesh line can lie in different DG triangles. This would cause non-smooth displacement of interior mesh nodes; an example is presented in detail in section 3.3.1.

One way of improving this method, that was considered by the author and Dr Hong Luo, was to include a few interior mesh points in the Delaunay graph such that the Delaunay graph lines stayed approximately normal to the boundary near it. For example, a layer of nodes could be chosen around each component of a 3-component airfoil such that each airfoil boundary gets approximately normal Delaunay graph lines. This method would require movement of the interior Delaunay graph nodes by a different method, such as RBF or linear elasticity. Since the number of these interior nodes in the Delaunay graph is very small, solution of the RBF or linear elasticity problems could be done quickly.

Xiao et. al. present a very similar method [30]. Instead of choosing interior Delaunay graph points from among the interior mesh points, they generate a 'background' coarse mesh to serve as the Delaunay graph (DG). The motion of the DG is obtained using Batina's lineal spring analogy method [3].

2.3 Linear Mesh quality

In order to judge the effectiveness of mesh-movement methods, we need to measure the quality of the deformed mesh. Some mesh quality measures have been derived for linear 2D and 3D

elements by Knupp [17]. These include the relative size, shape and size-shape metrics for triangles and tetrahedra, and size, shape, skew, size-shape and size-skew metrics for quadrangles and hexahedra.

The size metric distinguishes elements having very small or very large volume with respect to some reference volume. The shape metric identifies elements that have unequal edges or unequal angles between edges; it is independent of size of the element. Finally, the skew metric is a measure of unequal angles between edges of the element only, irrespective of lengths of edges and volume of element. The skew metric is different from shape in case of non-simplicial (like quadrangular and hexahedral) elements. While dealing with quadrilaterals, we use the skew metric. This is because for boundary layer meshes, cells with high aspect-ratios are desired, but they have poor shape metric. For triangular meshes, cells with high aspect-ratio must have very unequal interior angles too, leaving no difference between shape and skew.

All metrics are normalized so that they are 1.0 for ideal elements with regard to the characteristic they measure, and they are 0.0 for degenerate elements.

Chapter 3

Curved mesh generation

3.1 Introduction

The problem of curved mesh generation involves at least one issue - the placement of the boundary "high-order" nodes. A description of a high-order boundary is required, and the optimal placement of high-order boundary nodes must be decided.

Additionally, we usually need to deal with one more issue - maintaining the quality of elements near the boundary after placing the high-order boundary nodes. The deformation of the initially polygonal/polyhedral boundary to get a curved boundary usually causes the quality of boundary elements to get reduced. In cases with a viscous boundary layer mesh, these elements even become invalid [23, 26]. Thus, the deformation of the boundary must be propagated into the domain, to displace interior nodes upto at least some distance from the boundary.

In our knowledge, there are three types of methods to achieve regularization of the interior mesh. The first type uses models based on solid mechanics to try to achieve valid mesh movement, as explained in the previous chapter. The second type of technique to move the interior nodes involves interpolation of the boundary displacement to interior nodes, again, as previously explained. An interesting development in curved mesh generation by interpolation is due to Ims at. al. [13], in which explicit interpolation ([21]) is used to curve the interior of the mesh. Finally, researchers have "untangled" the near-boundary elements, and otherwise improved the quality of the mesh, by optimization processes. The cost function is usually some kind of curved-mesh quality measure. One prominent example of this is in the Gmsh meshing software [26].

We present an alternative approach based on interpolation by radial basis functions. This approach is compared to the more prevalent methods of linear elasticity [11].

In the rest of the sections, we discuss the methodology used to generate 2D unstructured

quadratic meshes from linear meshes, and then present examples to demonstrate the effectiveness of our method. We compare the quality of meshes given by the RBF interpolation method and the linear-elasticity method.

We initially preprocess the linear mesh to generate a straight-faceted high-order mesh. To do this, we introduce extra nodes along edges, in faces and inside cells as required. Next, we either use CAD data or use a high-order boundary reconstruction method to obtain the actual positions of boundary high-order nodes. Finally, interior nodes are moved according to the boundary displacement imposed because of the previous step.

3.2 Spline reconstruction for boundaries of 2D meshes

Since only the linear mesh is taken as input, a high-order boundary representation first needs to be reconstructed from the piecewise linear C^0 boundary. For 2D meshes, we use a cubic spline reconstruction to get a smooth (C^2) parametric curve describing the boundary. All boundary nodes are used as spline control points, and we get a cubic function between every two consecutive boundary nodes. At the control points, we require the two spline curves meeting there to share a common tangent and curvature, thus enforcing C^2 continuity. Since it is common for the true boundary to be specified in terms of cubic spline curves, this is expected to give us an accurate reconstruction. Corners are detected by comparing the normal vectors of the two facets sharing a point. If the dot product between the two normal vectors is below a particular user-specified threshold, the point is considered as a corner and is not smoothed over by the reconstruction procedure. Alternatively, the user can list all the corner points as input. Thus the reconstructed boundary is piecewise C^2, which is found to work well for the test cases we considered. The spline reconstruction requires the solution of a symmetric positive definite linear system, of size equal to the number of boundary nodes in the curved part of the boundary, in order to calculate the cubic spline coefficients. The system(s) can be solved quickly using a conjugate gradient (CG) solver. More details are given in appendix B.

Instead of a global reconstruction by cubic splines, a local reconstruction at each boundary node could also be used. This is described briefly for 3D meshes in the last chapter, conclusions and future work.

Once we have the smooth reconstructed boundary, we calculate the final positions of the boundary nodes in the curved mesh. This is currently done by simply moving the high-order boundary nodes, originally at regular intervals on the boundary facet, to corresponding intervals in parameter space on the cubic spline curve associated with that facet. This method is found to be quite robust for the meshes tested. Thus, the boundary displacements are obtained. Alternatively, if the CAD geometry is available, the displacements can be computed using it.

3.3 Interior mesh movement

To propagate the boundary motion to the interior of the mesh, we favor interpolation by radial basis functions (RBFs) [6]. We have also used linear elasticity to propagate the mesh movement to the interior. The isotropic linear elasticity variational formulation, as given by Gockenbach [9], is implemented in a P2 Galerkin FEM code for generation of quadratic meshes. In this case however, the size of the linear system to be solved is proportional to the total number of nodes in the mesh.

Though the RBF method requires the solution of a linear system of size proportional to only the number of boundary points, extra computation is required to evaluate equation (2.10) at each of the interior nodes to compute their displacements. In comparison, elasticity-based methods directly give us the displacements.

While solving certain problems with the RBF method, we find that the linear system to be solved is sometimes quite ill-conditioned, depending upon the mesh. However, in the cases that we worked on (such as curved mesh generation of hybrid meshes for viscous simulations of airfoils), sparse direct methods are capable of solving the system efficiently and effectively. This is likely to hold true in general as the systems to be solved will usually not be very large (having size of the number of boundary nodes).

We have also tried using the stiffened linear elasticity method described in subsection 2.1.2. In this work, this is implemented by scaling each component of the element stiffness matrix of each element by the scaling term $\left(\frac{J_0}{J_\kappa}\right)^\chi$. This is valid as we only need to run the linear elasticity code on straight-sided meshes, and this means that the Jacobian J_κ is constant over the element.

3.3.1 Unsuitability of Delaunay-graph-based methods for curved mesh generation

Several variants of the Delaunay graph mapping method were also attempted for generating curved meshes. While Delaunay graph mapping (DGM) is a very efficient scheme for accomplishing many types of mesh movement, it has a characteristic because of which it is unsuitable for certain meshes and motions, especially curved mesh generation. This is the fact that the movement of a node depends only on the motion of the Delaunay element containing it, not necessarily on the motion of the nearest boundary facets.

In case of curved mesh generation for highly anisotropic viscous meshes, it is possible for some Delaunay triangulation facets make angles with the boundary that are very different from 90 degrees (see figure 3.2). Then we may have at least two issues.

1. Interior points close to a boundary facet can be influenced by another boundary facet far

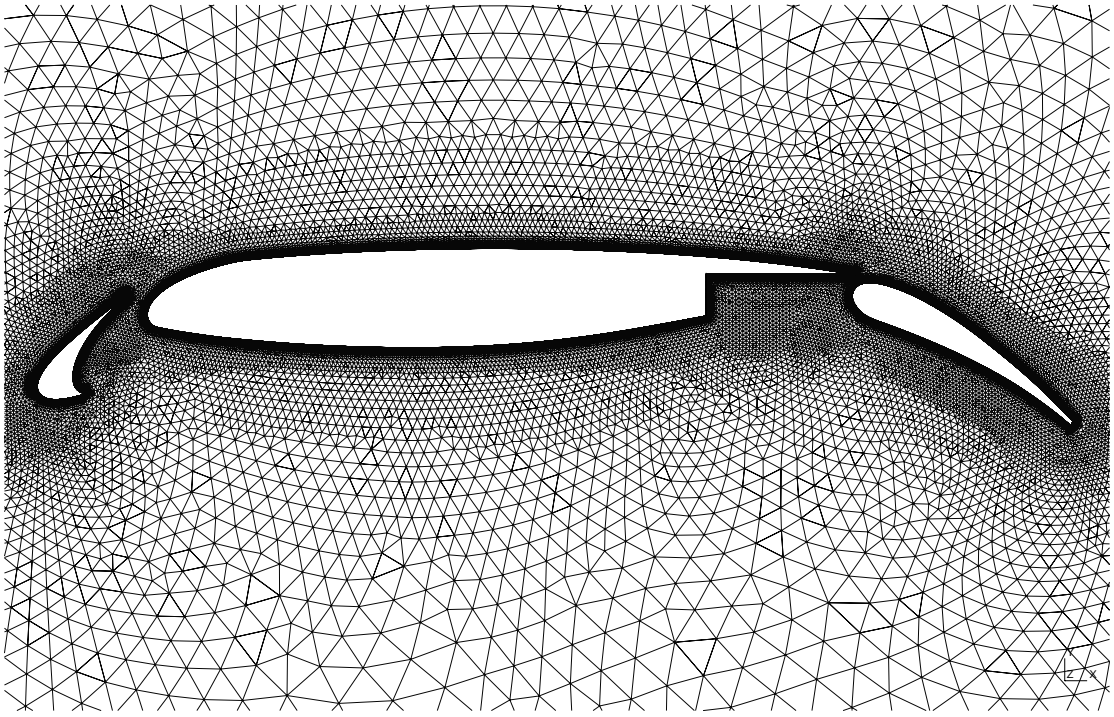

Figure 3.1: Mesh of a 3-element airfoil

away having a very different curvature.

2. It may be the case that an interior point is not at all influenced by the closest boundary facet.

These issues lead to low curved mesh quality in certain regions, and even invalid elements. We could try using DG-RBF (interpolation by radial basis functions in a Delaunay graph) to help alleviate issue (1), but issue (2) would still persist.

Figure 3.1 shows a viscous mesh of a 3-element airfoil. Figures 3.2 and 3.3 show a portion of its slat. Figure 3.2 shows the Delaunay graph and figure 3.3 shows the deformed mesh. We can see that the DG lines make an angle very different from 90 degrees with the boundary. It is clear how points originally in a straight line deform in an oscillatory manner because successive points depend on different Delaunay elements, and different Delaunay elements have different movements depending on which boundary nodes they are made up of.

One way of improving this situation has been described in section 2.2.4. However, the method of including a layer of interior points still does not work for curved mesh generation. This is simply to due to the high-frequency oscillatory nature of boundary displacements required for curved mesh generation and the fact that it is difficult to control how exactly the Delaunay graph (DG) elements are formed. It was found that some DG elements are formed by *two high-*

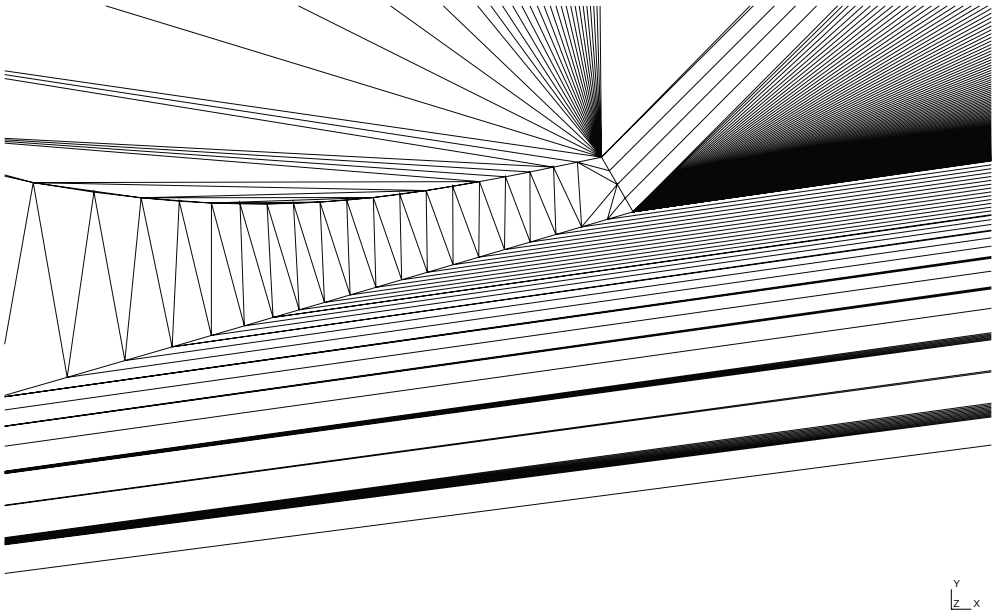

Figure 3.2: Portion of the Delaunay graph near the lower part of the slat

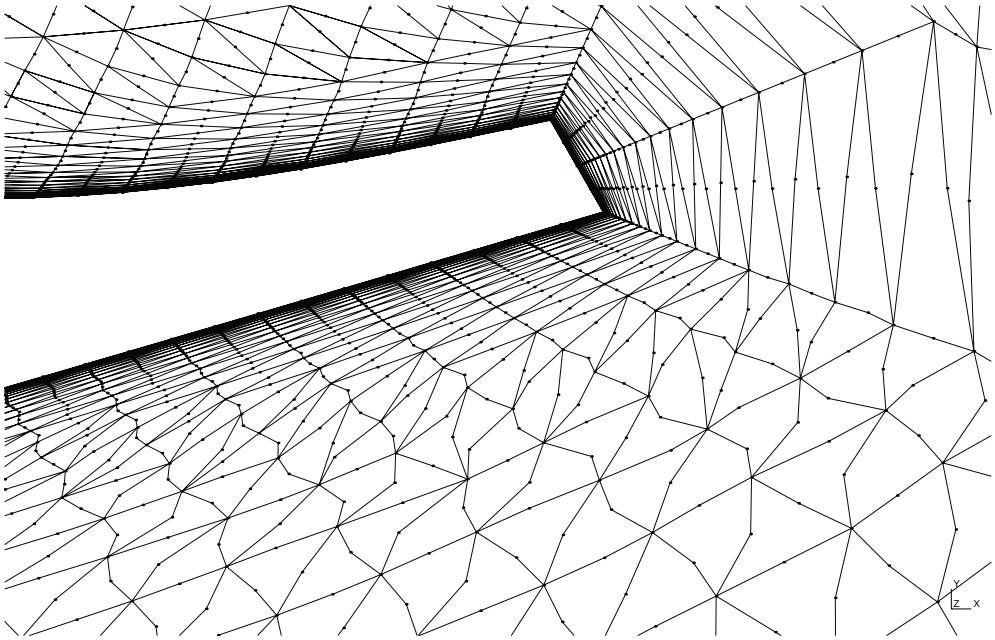

Figure 3.3: Portion of the generated mesh near the lower part of the slat

order boundary nodes and one interior node, as shown in 3.4. In the case shown, some interior nodes near the red circles (linear mesh nodes, called vertices) lie within a DG element that has nothing to do with the nearest boundary node, which is a boundary vertex in this case. A few interior nodes nearest to the boundary vertices are not inside the DG element, so they do not move because of the motion of the boundary. But beyond a small distance, the mesh nodes are in the DG element formed from two nearby high-order boundary nodes. This means that these nodes move with the DG element, and this creates a discontinuity in the interior node movement, leading to a bad boundary layer mesh.

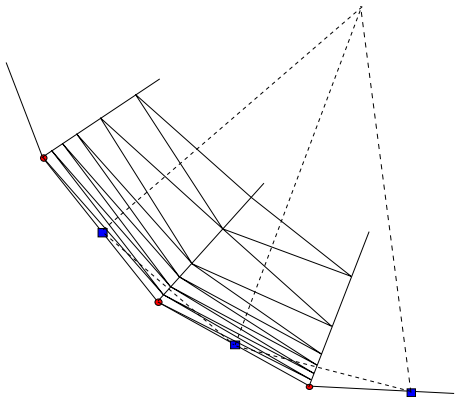

Figure 3.4: Schematic diagram to show a possible Delaunay graph (dashed lines) and mesh (solid lines) near a boundary; circles represent regular boundary nodes; squares represent high-order boundary nodes

Chapter 4

Results

4.1 Mesh movement

First, we show qualitatively the results for large deformation of a 2D unstructured linear mesh. In figure 4.1, we show the undeformed mesh. In each of the results that follow, the flap of the wing has been rotated anti-clockwise by 60 degrees. We take note of the quality of triangular cells in these results. By 'quality', we mean deviation of the angles of the triangle from 60°, that is, the shape metric (sec. 2.3), for this test case.

In figure 4.2, we show the result of the torsion spring method of Farhat et. al. [7]. We see that the result for such a large deformation is not acceptable; it is evident that there are invalid cells. It is interesting to note that cells near the trailing edge of the middle wing do not suffer much degradation, but cells somewhat farther away become invalid. Cells near the trailing edge of the flap also suffer large distortions.

Next, results of mesh movement by linear elasticity method are shown in figures 4.3, 4.4 and 4.5. The mesh is valid; however, it suffers from nearly-degenerate cells near the trailing edge of the flap 4.5. This could seriously impact the performance, accuracy and even stability of flow solvers using this mesh. In figure 4.4, we see that the quality of cells is better than near the trailing edge, but not very good.

In figures 4.6 and 4.7, we present results with the Delaunay graph mapping (DGM) method. We see that cell quality is severly degraded near the interface between trailing edge of the wing and the flap. The quality of the cells near the trailing edge of the flap is better than the linear elasticity case, however. We remark here that DGM is computationally the most frugal method out of all the ones presented, as we need to solve only 3x3 linear systems after generating a background Delaunay graph, both of which are very fast (sec. 2.2.1).

Finally, we take note of the good qualities of the mesh moved by radial basis function (RBF) interpolation in figures 4.8, 4.9 and 4.10. We see that mesh quality is not lost to a great

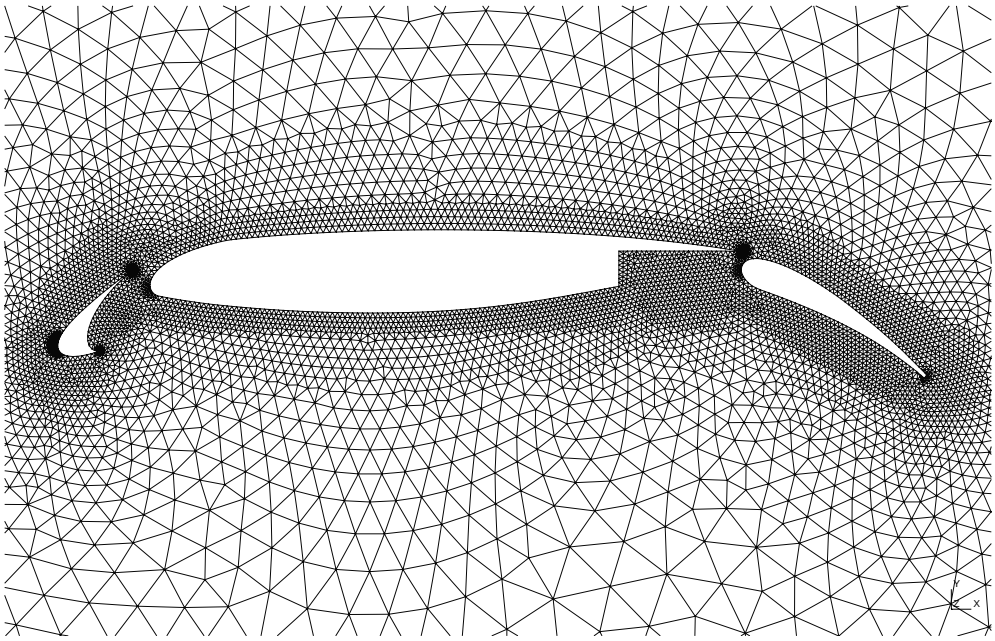

Figure 4.1: Original mesh for inviscid flow around 3-component airfoil

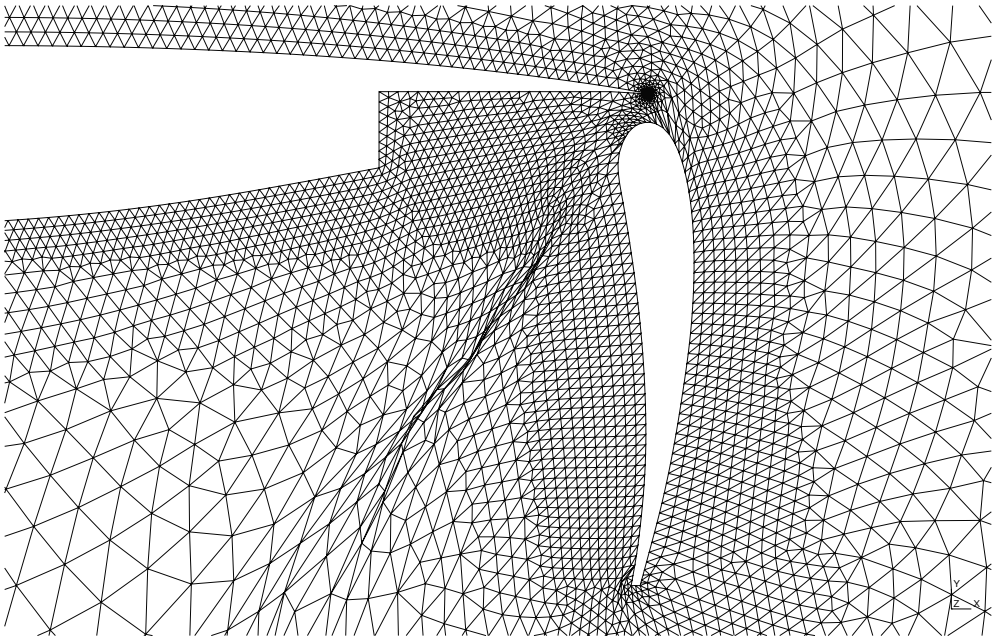

Figure 4.2: Inviscid 3-component airfoil mesh with 60° rotation of flap by torsion spring method

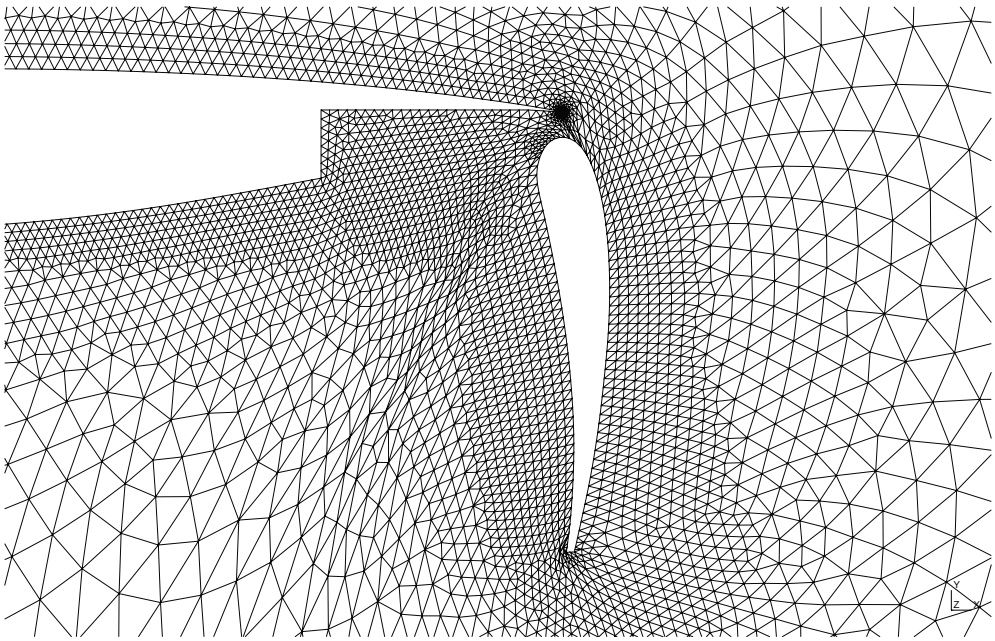

Figure 4.3: Inviscid 3-component airfoil mesh with 60° rotation of flap by linear elasticity method

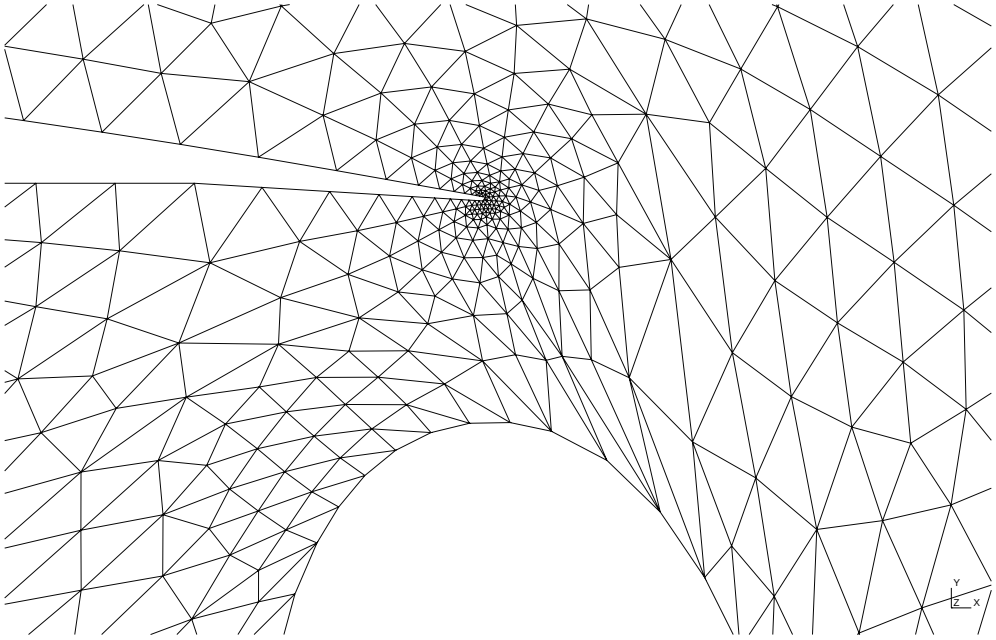

Figure 4.4: Inviscid 3-component airfoil mesh with 60° rotation of flap by linear elasticity method; zoomed to where the flap meets the wing

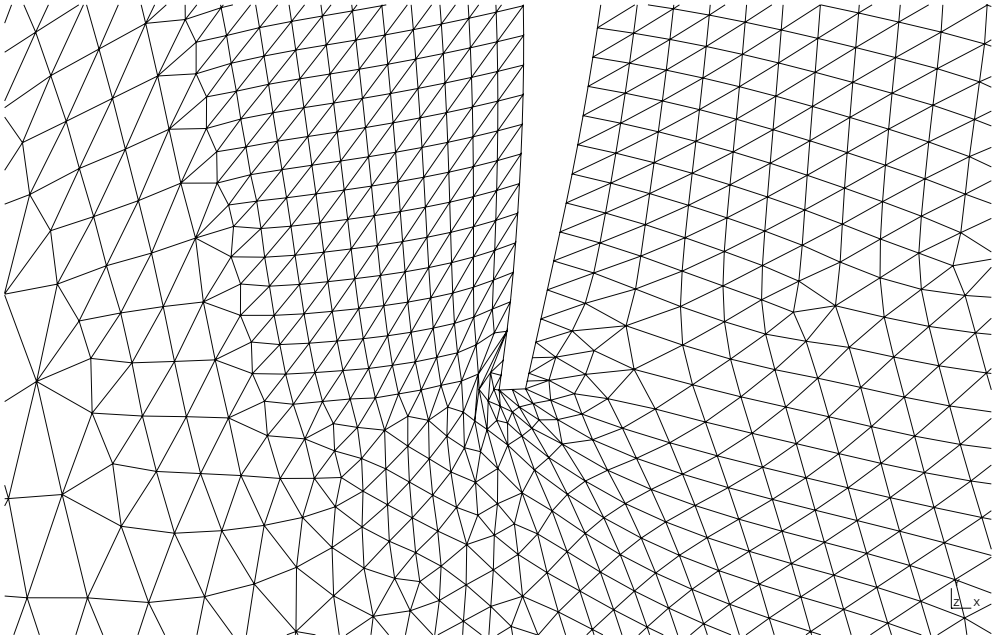

Figure 4.5: Inviscid 3-component airfoil mesh with 60° rotation of flap by linear elasticity method, zoomed to the trailing edge of the flap

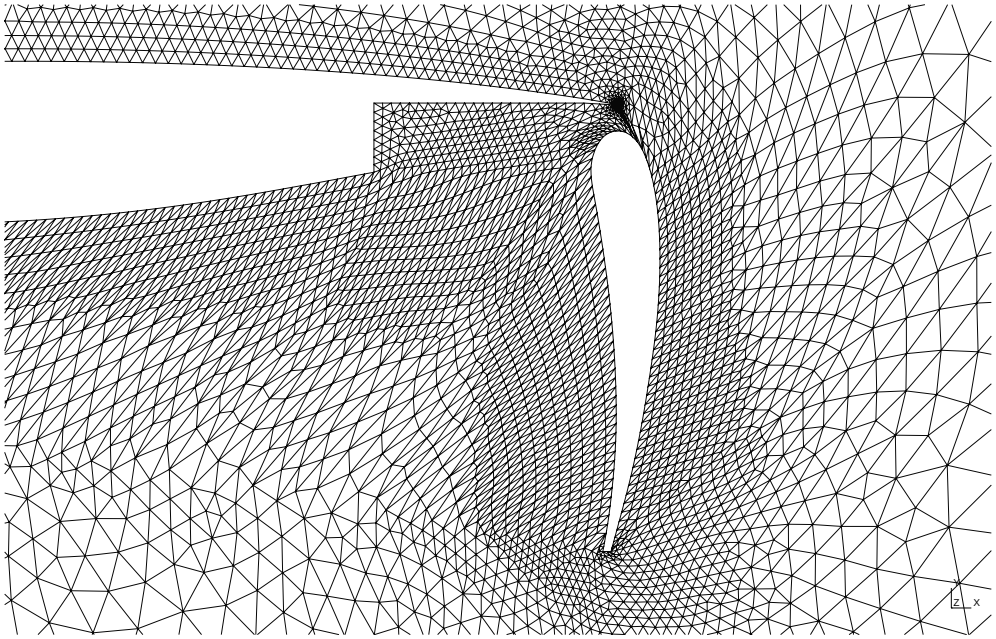

Figure 4.6: Inviscid 3-component airfoil mesh with 60° rotation of flap by DGM method

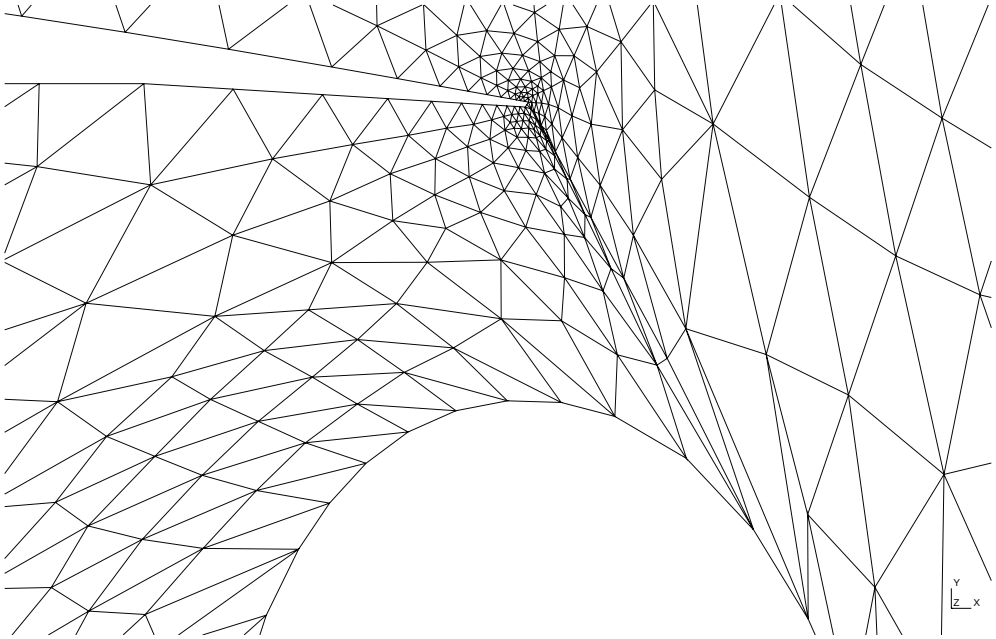

Figure 4.7: Inviscid 3-component airfoil mesh with 60° rotation of flap by DGM method; zoomed to where the flap meets the wing

extent anywhere. The support radius is 15.0 units (for reference, the chord length of the wing is approximately 18.3) and the entire movement is carried out in 3 steps.

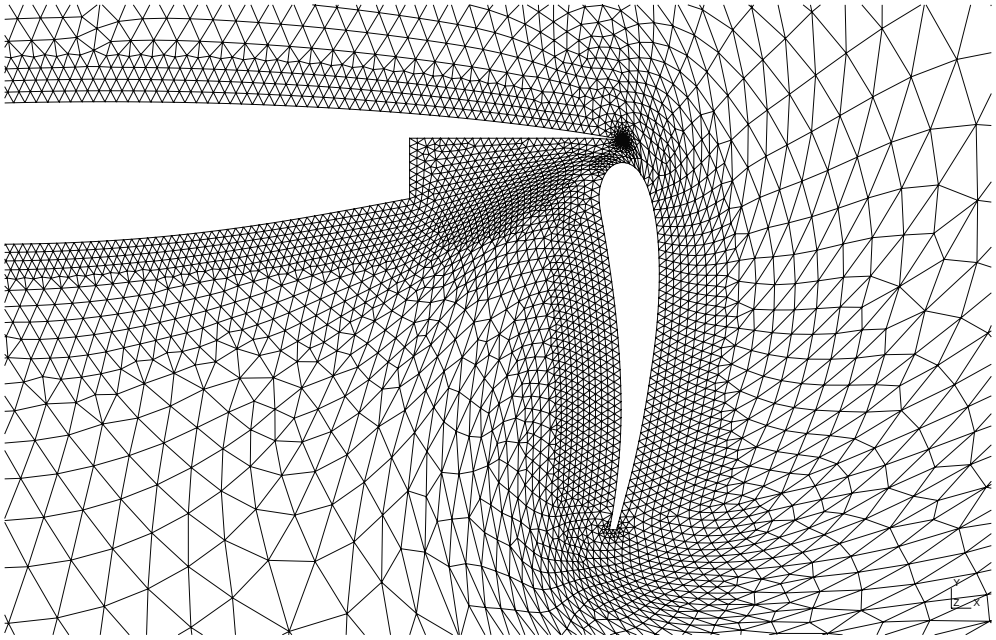

Figure 4.8: Inviscid 3-component airfoil mesh with 60° rotation of flap by RBF method

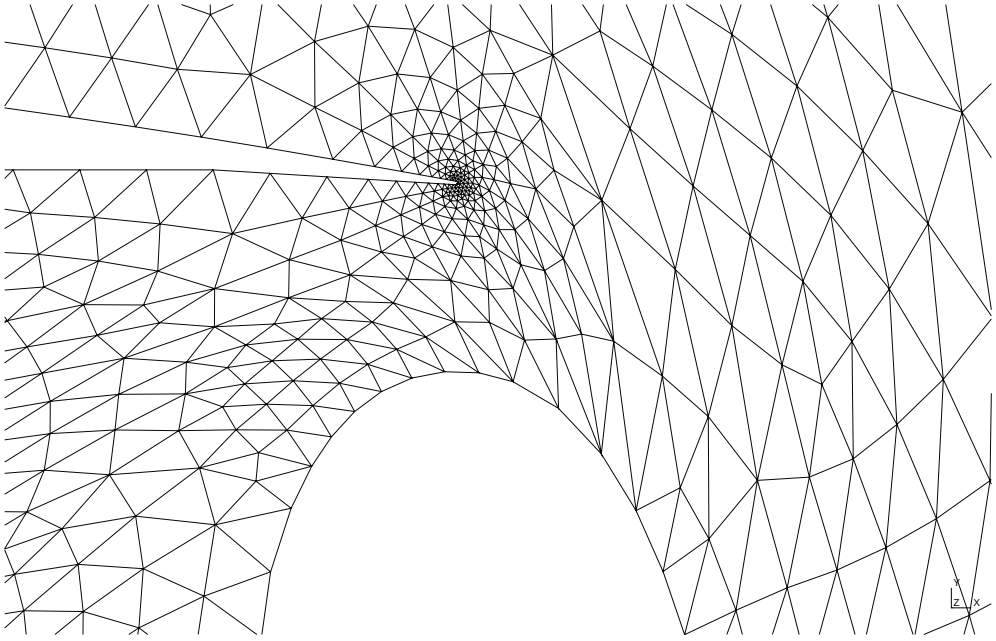

Figure 4.9: Inviscid 3-component airfoil mesh with 60° rotation of flap by RBF method; zoomed to where the flap meets the wing

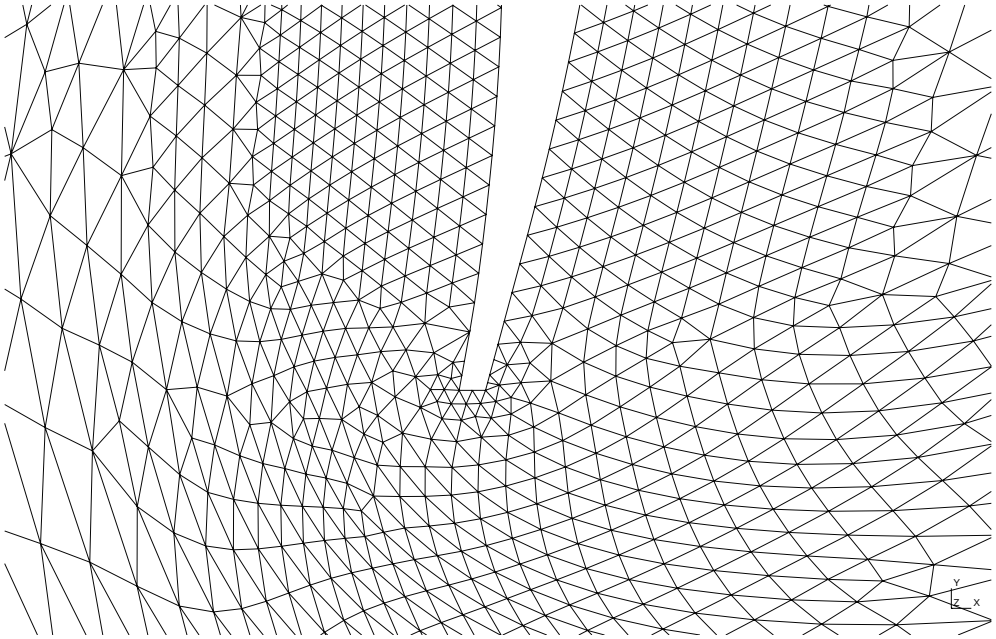

Figure 4.10: Inviscid 3-component airfoil mesh with 60° rotation of flap by RBF method, zoomed to the trailing edge of the flap

We present a case of rotation of an interior object inside a far-field boundary using interpolation methods on a quadrilateral mesh. The skew metric has been computed for this case. We show results for a 60-degree rotation in case of DGM (fig. 4.12), RBF (fig. 4.13) and DGRBF2 (fig. 4.14) methods. Note that DGRBF2 refers to the interpolation of rotation angles by DGRBF. The case has been referred from Wang *et. al.* [28]. Figure 4.11 shows the un-deformed mesh.

We note that the RBF and DGRBF2 methods can handle the rotation case quite well. The high-aspect-ratio elements near the inner boundaries are well-preserved, while the elements further in the interior, which are larger and can thus take more deformation, are distorted somewhat. Though RBF gives slightly better mesh quality than DGRBF2 for this amount of rotation, RBF takes 1.926 seconds of total CPU time (using conjugate gradient solver), while DGRBF2 takes only 0.08 seconds of total CPU time. Further, shown in figure 4.15 is a case of extreme rotation of 120 degrees, which DGRBF2 solves while maintaining mesh validity. None of the other methods are able to do this. RBF has been found to give valid meshes upto about 110 degrees of rotation. However, since DGRBF2 requires specification of rotation angles at boundary nodes, the method can be difficult to apply to general complex motions.

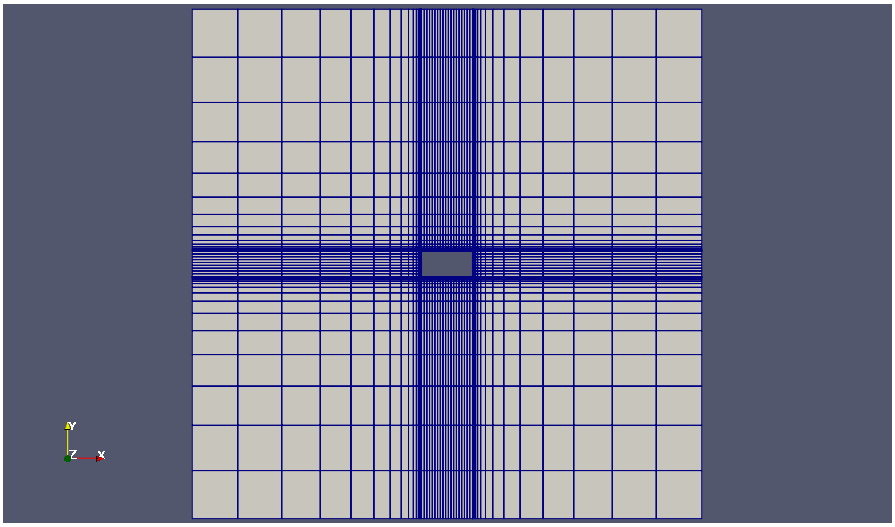

Figure 4.11: Original mesh

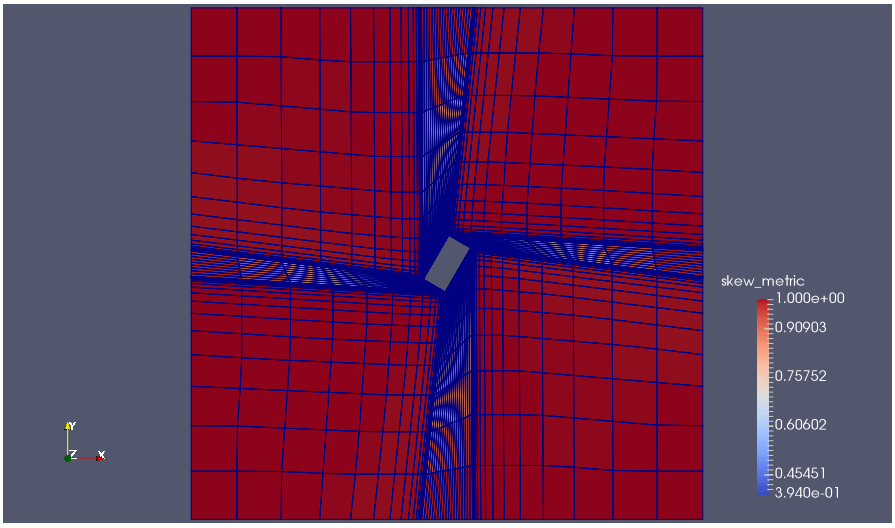

Figure 4.12: 60 degrees rotation by DGM

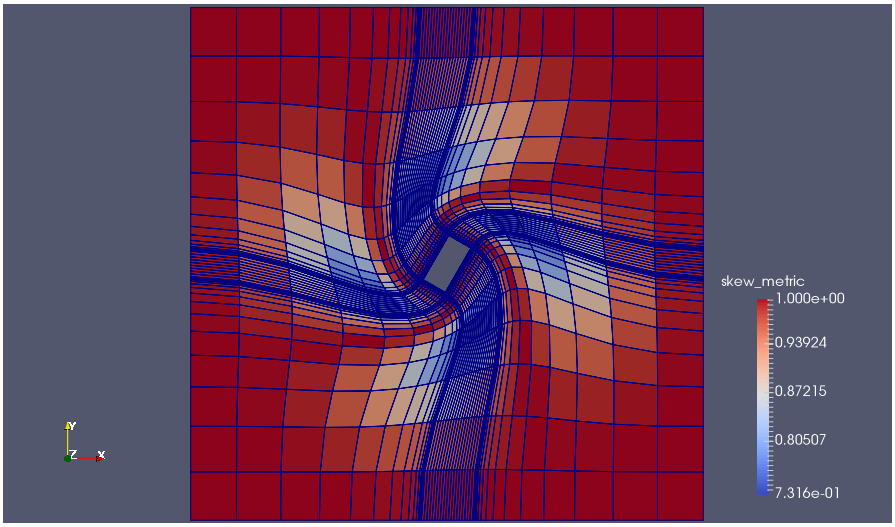

Figure 4.13: 60 degrees rotation by RBF

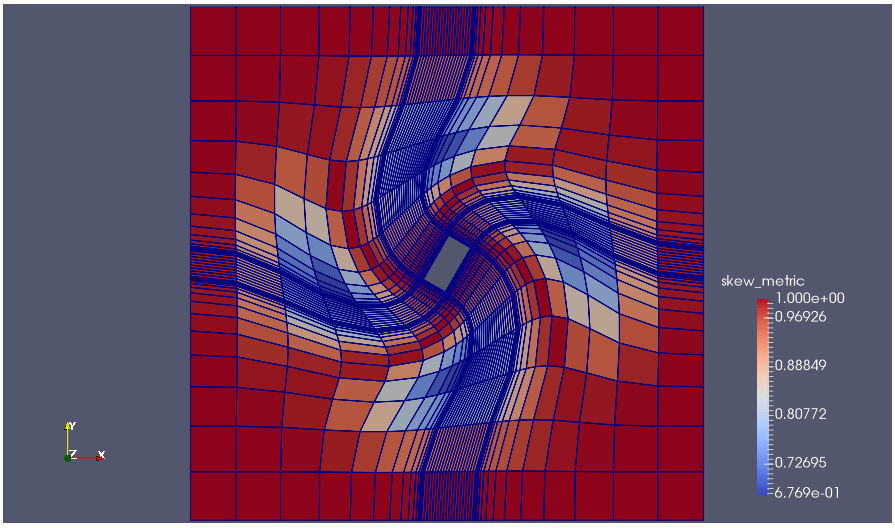

Figure 4.14: 60 degrees rotation by DGRBF2

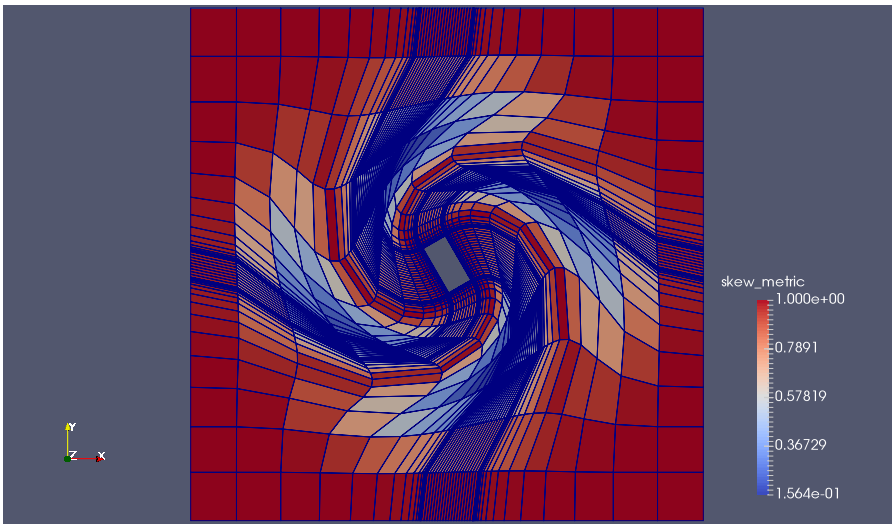

Figure 4.15: Large rotational motion carried out by DGRBF2

4.2 Generation of curved meshes

Curved meshes for various standard viscous flow test cases have been generated with the method described in the previous section. These include flow past NACA0012 airfoil, flow through a bump channel and flow past a multi-element airfoil. Out of these, the mesh for the multi-element airfoil case is a truly unstructured mesh; the others are structured meshes which we first convert to unstructured format.

The validity and quality of the generated mesh is established by computing the minimum scaled Jacobian determinant for each element. If the minimum value of the Jacobian determinant is zero or negative, the element is considered invalid. This is done as a post-processing step using the plugin 'AnalyseCurvedMesh' available in Gmsh [8]. The quantity computed by this plugin is given as

$$m_i = \frac{\inf_{\mathbf{x}\in\Omega_i} \det \mathbf{J}(\mathbf{x})}{\det \mathbf{J}_{\mathbf{l}i}}, \tag{4.1}$$

where $\mathbf{J}$ is the Jacobian matrix of the transformation of a reference element to the curved physical element, and $\mathbf{J_l}$ is the Jacobian of the transformation of the reference element to the linear (straight) physical element. Ω_i represents the ith element.

4.2.1 2D Viscous Flow Mesh of Multi-element airfoil

Here, we present results of 2D curved mesh generation in case of a multi-element airfoil. The mesh has 66530 triangular elements, 133768 total nodes and 1420 boundary nodes. It is meant for viscous flow computations and thus has highly stretched, thin elements near the wing bound-

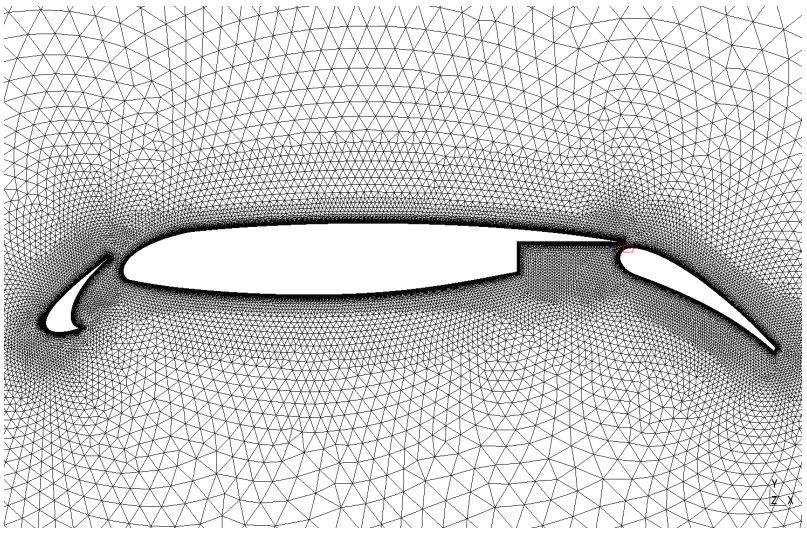

Figure 4.16: Mesh of multi-element airfoil. The little box shows approximately the region which is magnified in the figures that follow.

aries. In this case, if the interior nodes are not moved properly, invalid elements result. In the figures, we compare curved meshes generated by isotropic linear elasticity and RBF interpolation. Both problems are solved using a point Jacobi preconditioned conjugate gradient solver. Linear elasticity method uses a Young's modulus E of 1×10^6 Pa and a Poisson's ratio of 0.4, though it is observed that different values do not influence the outcome; a number of values over the range 1.0 to 1.0×10^{10} were tried for E. RBF interpolation uses a support radius of 0.04 (for reference, the chord length of the wing is approximately 18.3).

In figure 4.17, the first mesh is tangled; even though the nodes are being moved, they are not being moved far enough. It is clear that simple linear elasticity is not enough for curved mesh generation - we get negative Jacobians in several near-boundary elements for many curved boundary faces. However, the RBF method is satisfactory, with a scaled minimum Jacobian range of 0.64 to 1.0 throughout the mesh (figure 4.18). As explained earlier, we see that the elements very close to the boundary deform very little - they have scaled Jacobian nearly 1.0. As we go further from the boundary, the Jacobian drops to a minimum and then rises again to 1.0 at a distance that approximately equals the support radius.

Next, we present results of the same case, but using Jacobian-based stiffening in the linear elasticity formulation. In table 4.1 we compare RBF and stiffened linear elasticity; all cases are solved using point-Jacobi preconditioned conjugate gradient solver with a tolerance of 1×10^{-6}.

It is immediately clear that RBF vastly outperforms SLE for curved mesh generation while results of both methods for this case are largely equally good. One interesting characteristic of SLE is that the CG solver takes more and more iterations to converge with increase in

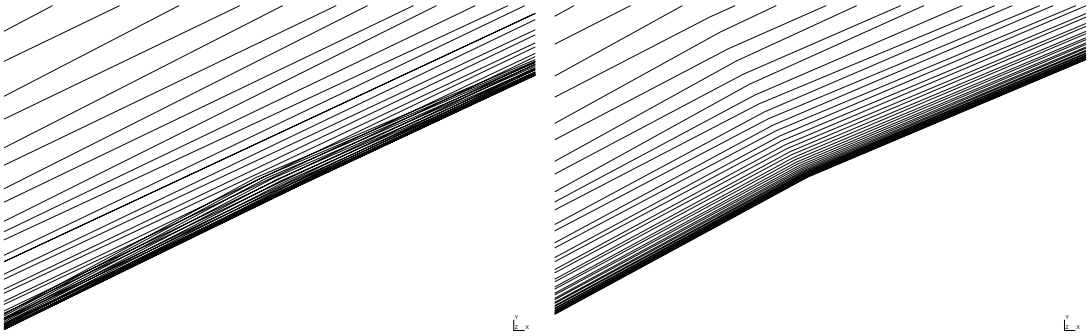

Figure 4.17: Portion of quadratic viscous mesh for multi-element airfoil, showing a boundary face in the flap, generated by linear elasticity (left) and RBF (right) methods. Both are zoomed to for a clearer view.

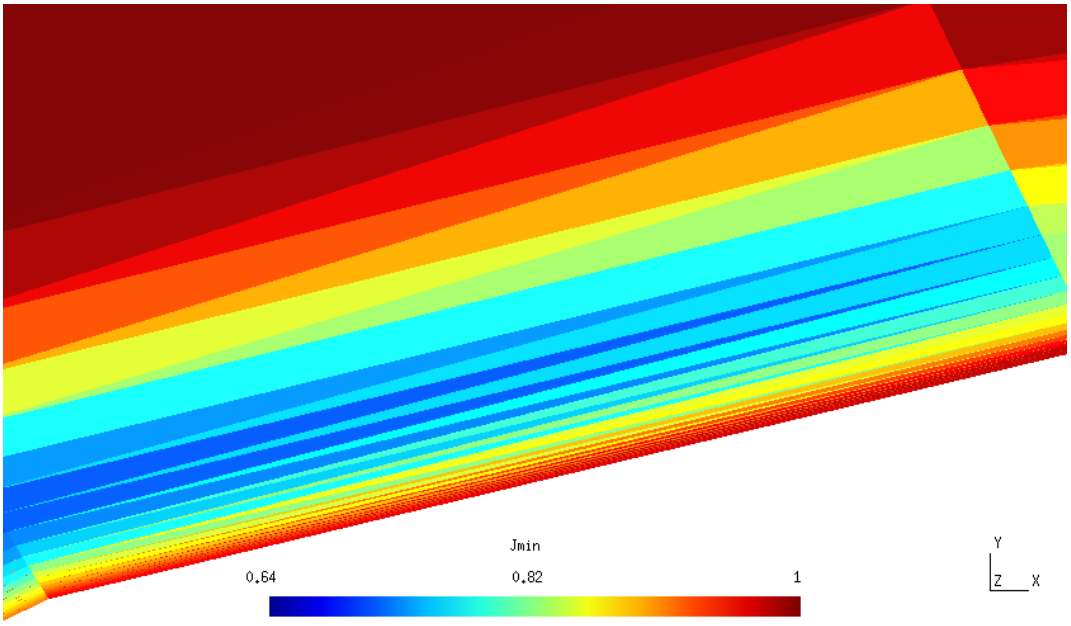

Figure 4.18: Minimum scaled Jacobian over each element for mesh generated by RBF interpolation. (The mesh is actually curved, though the graphics of Gmsh ignore that.)

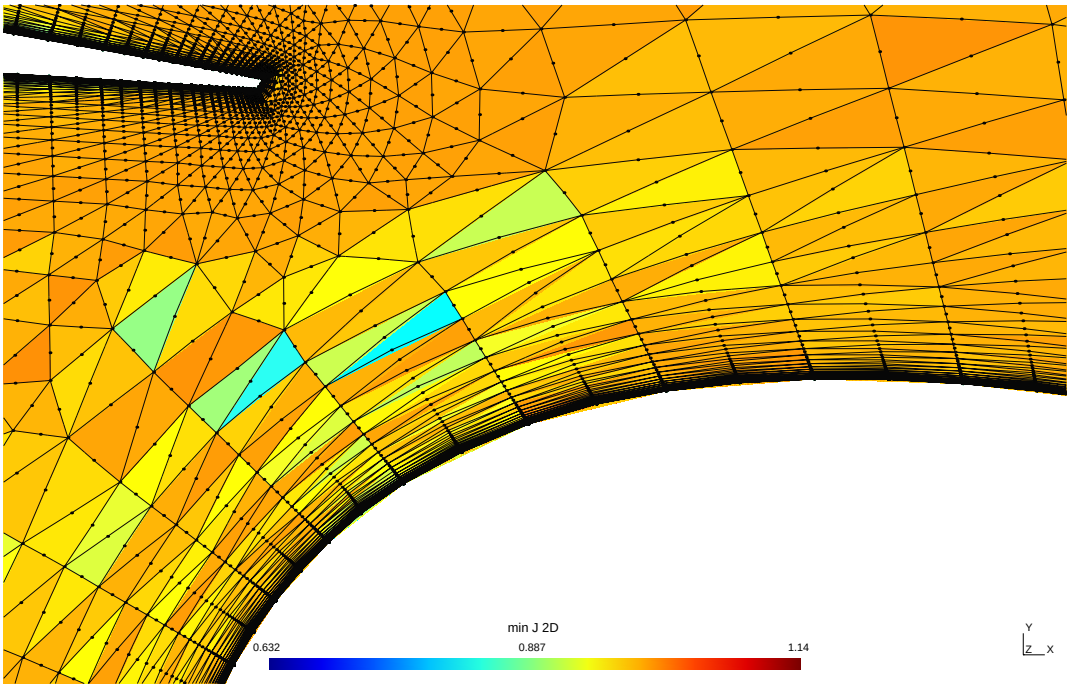

Figure 4.19: An illustrative example of a curved mesh generated by stiffened elasticity method; minimum scaled Jacobian over each element is shown

the stiffening exponent χ. Therefore, we conclude that the stiffness matrix seems to get more and more ill-conditioned with increase in the stiffening exponent χ, while not providing much improvement in mesh quality. As far as RBF is concerned, we see that increasing the number of steps is not worth the additional time consumed by the solver. Further, it is interesting to note that an increase in support radius beyond a value degrades the mesh quality; there is/are optimum value(s) of the support radius, less than and more than which the mesh quality drops.

Method	Parameters	Minimum scaled Jacobian	Wall clock time	Solver iterations
SLE	$\chi = 2.75$	0.632	19.3s	324
	$\chi = 2.9$	0.632	24.4s	423
RBF	1-step $r_s = 0.04$	0.640	1.86s	365 x 2
	1-step $r_s = 0.08$	0.639	1.9s	1150 x 2
	2-step $r_s = 0.04$	0.641	2.58s	365 x 4

Table 4.1: Comparison of RBF (radial basis function) and SLE (stiffened linear elasticity) methods

4.2.2 3D Viscous Flow Mesh for 3D Bump

This is a Reynolds-averaged Navier-Stokes (RANS) flow simulation test case from the NASA Turbulence models website [25]. The simulation parameters include a reference domain length scale of 1.0, Mach number 0.2, Reynolds number of 3×10^6, and a maximum bump height of 0.05.

For this case, boundary displacements were first obtained from the analytical expression for the bump given on the website. Then, RBF method was used to regularize the interior mesh using Wendland's C2 function. It is notable that this one of the cases where a conjugate gradient solver cannot solve the RBF system. We use a sparse LU decomposition from the Eigen3 C++ library [10].

	Coarse	Medium	Fine
No. of points	32841	243729	1875489
No. of boundary points	2304	8704	33792
Support radius	0.06	0.06	0.04
Minimum scaled Jacobian	0.714	0.852	0.926
Wall-clock time	0.68s	12.6s	338s

Table 4.2: Summary of 3D bump curved mesh generation

Further, we present results from NC State University's reconstructed discontinuous Galerkin turbulent flow code RDGFLO run on the curved meshes. A DG P1 scheme with Taylor basis, HLLC inviscid flux, Bassi-Rebay 2 (BR2) viscous flux, and first-order implicit time integration using Newton linearization and LU-SGS preconditioned GMRES solver is used for obtaining the solution; we refer to Liu et. al. for details [20]. For comparison, the result obtained from a curved mesh formed by agglomeration of a finer mesh is also shown. This is to show that the generated curved meshes actually work well when used for running simulations. The convergence of the mass-flux residual with time steps is shown in figure 4.23.

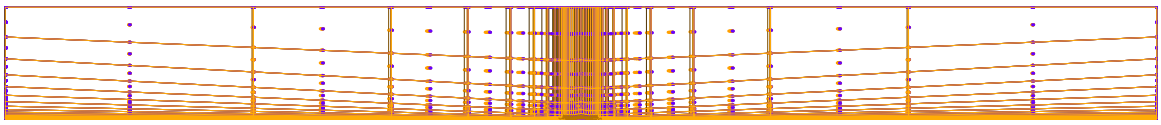

Figure 4.20: Coarsest curved mesh of 3D bump

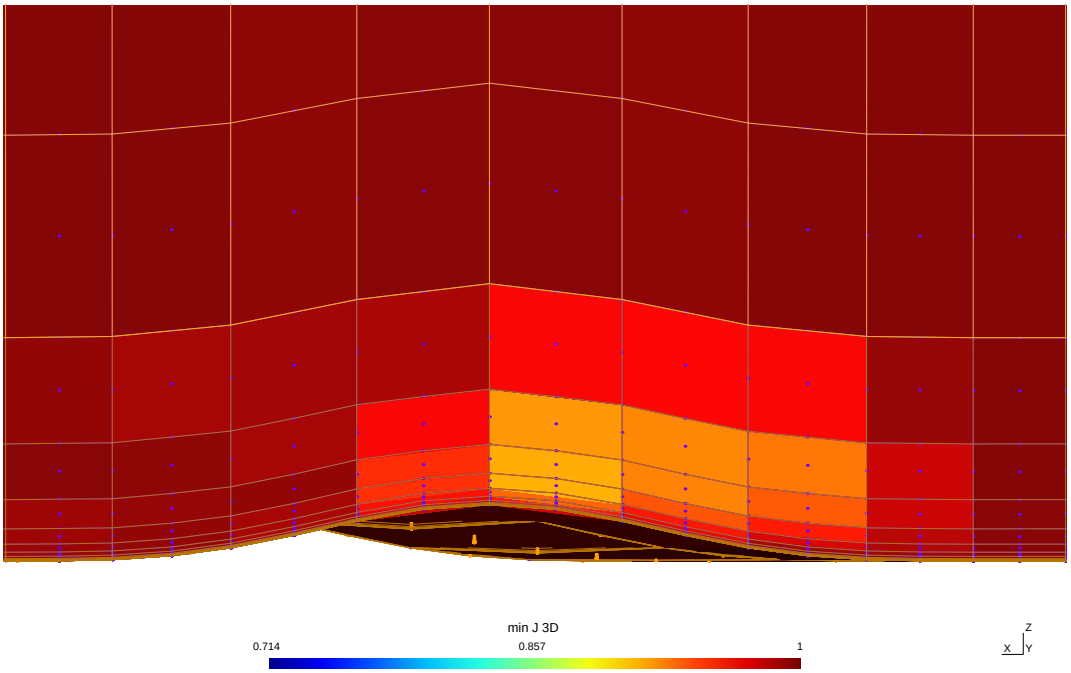

Figure 4.21: Minimum scaled Jacobian of coarse curved mesh of 3D bump

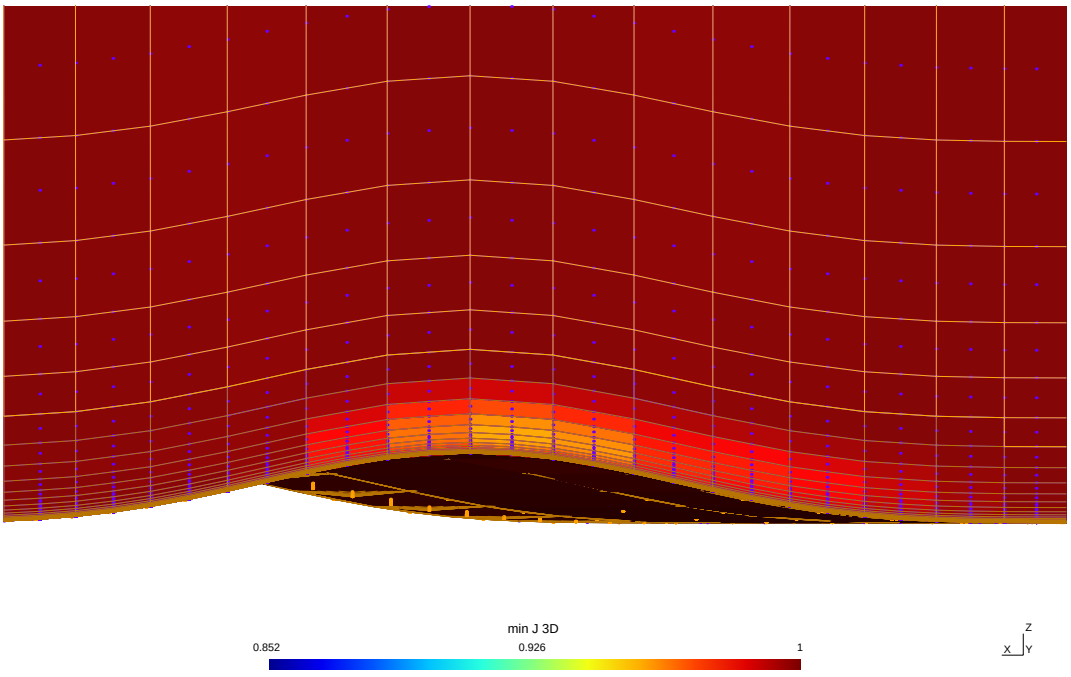

Figure 4.22: Minimum scaled Jacobian of medium curved mesh of 3D bump

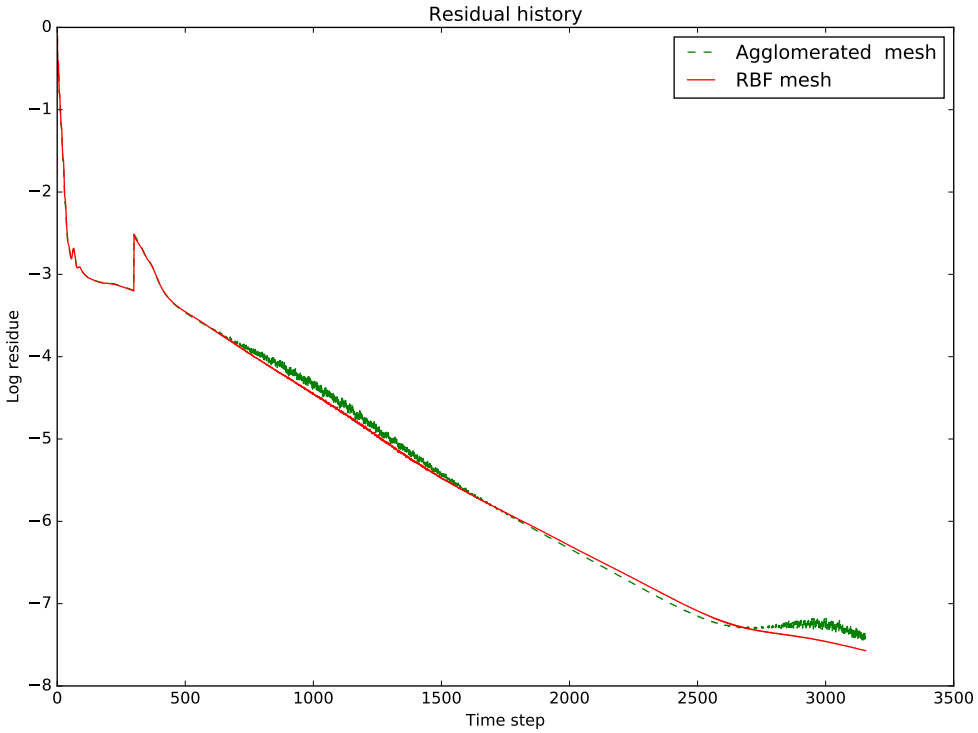

Figure 4.23: Comparison of mass-flux residual convergence history with time steps for implicit DG P1 solution

Chapter 5

Conclusions and Future Work

A lot of choice is available for mesh-movement techniques, but only a few will be good for a given application. We conclude that interpolation methods are generally well-suited for unsteady simulations that require mesh movement, as they are very fast. When deformations are relatively small and not highly rotational, Delaunay graph mapping (DGM) is a good choice; when larger and more general deformations are needed, the pure radial basis function (RBF) and to some extent the DGM with RBF interpolation (DGRBF) methods are good. The pure RBF method, while usually giving good robust results, is more expensive than the other interpolation methods considered here. The DGRBF methods, while being inexpensive, are only robust when implemented with angle interpolation, which may be difficult to do for general mesh movements. As for the linear elasticity methods in the context of general mesh movement, they perform reasonably well, and can be better than some interpolation methods such as DGM, but are usually much more expensive than interpolation methods.

RBF, and to some extent stiffened linear elasticity methods, are found to be effective for curved mesh generation, where computational cost is less of an issue. We have used RBF for curved mesh generation with good results for certain turbulent (RANS) flow cases. We find that RBF method is much more cost-effective than Jacobian-stiffened linear elasticity method for comparable results. Of note is that both methods provide 'knobs' to tune, such as the basis function and support radius in case of RBF and the stiffening criterion and stiffening exponent for the linear elasticity method. We also conclude that Delaunay graph mapping methods are pretty much unusable for curved mesh generation.

Even though the RBF linear system sometimes cannot be solved by iterative solvers like conjugate gradient and BiCGSTAB, sparse direct solvers have always worked in our experience. Since the number of boundary points is usually smaller by several orders of magnitude than the total number of points, this should usually not be an issue. Good, parallel sparse direct solvers are available, such as SuperLU [18], MUMPS [1], PaStiX [12] and many more. However, for

very large meshes with boundary nodes in the millions, it would be worthwhile to investigate iterative solvers for RBF.

One direction for future work is to complete a surface reconstruction procedure in 3D. For some meshes, the requirement of global C^2 continuity is too restrictive. Also, a procedure similar to that described in section 3.2 would be quite expensive for surfaces in $\mathbb{R}^3$. We therefore consider local fittings of 2D Taylor polynomials at every boundary vertex, described in [16] as "Weighted Averaging of Local Fittings" (WALF). In their paper, Jiao and Wang fit local 2D Taylor polynomials to each vertex of the surface mesh. The coefficients, that is, the derivatives of a local height function in a local coordinate system centered at the vertex, are solved for using vertex position data from a neighborhood of that vertex. It is ensured that there are more neighboring points being considered for data than the number of unknowns to solve for, thereby obtaining an over-determined system of equations. This is solved by a weighted least-squares approach. This is claimed to work well for both smooth surfaces and surfaces with C^1 discontinuities (ridges, corners etc.). For the latter case, additional preprocessing of the linear surface mesh is required to detect discontinuities [15].

REFERENCES

[1] P. R. Amestoy, I. S. Duff, J. Koster, and J.-Y. L'Excellent. A fully asynchronous multifrontal solver using distributed dynamic scheduling. *SIAM Journal on Matrix Analysis and Applications*, 23(1):15–41, 2001.

[2] F. Bassi and S. Rebay. High-order accurate discontinuous finite element solution of the 2d Euler equations. *Journal of Computational Physics*, 128:251–285, 1997.

[3] J.T. Batina. Unsteady Euler algorithm with unstructured dynamic mesh for complex-aircraft aerodynamic analysis. *AIAA Journal*, 29(3):327–333, 1991.

[4] A. Bowyer. Computing Dirichlet tessellations. *The Computer Journal*, 24(2):162–166, 1981.

[5] J.A. Cottrell, T.J.R. Hughes, and Y. Bazilevs. *Isogeometric Analysis*. John Wiley & Sons, Ltd, 2009.

[6] A. de Boer, M.S. van der Schoot, and M. Bijl. Mesh deformation based on radial basis function interpolation. *Computers & Structures*, 85:784–795, 2007.

[7] C. Farhat, C. Degand, B. Koobus, and M. Lesoinne. Torsional springs for two-dimensional dynamic unstructured fluid meshes. *Computer methods in applied mechanics and engineering*, 163:231–245, 1998.

[8] C. Geuzaine and J.-F. Remacle. Gmsh: a three-dimensional finite element mesh generator with built-in pre- and post-processing facilities. *International Journal for Numerical Methods in Engineering*, 79(11):1309–1331, 2009.

[9] M.S. Gockenback. *Understanding and Implementing the Finite Element Method*, chapter 9, pages 213–218. Society for Industrial and Applied Mathematics, 2006.

[10] Gaël Guennebaud, Benoît Jacob, et al. Eigen v3. http://eigen.tuxfamily.org, 2010.

[11] Ralf Hartmann and Tobias Leicht. High-order unstructured grid generation and discontinuous Galerkin discretization applied to a 3D high-lift configuration. 53rd AIAA Aerospace Sciences Meeting (SciTech 2015), AIAA 2015-0819, Kissimmee, Florida, 2015.

[12] P. Hénon, P. Ramet, and J. Roman. PaStiX: A High-Performance Parallel Direct Solver for Sparse Symmetric Definite Systems. *Parallel Computing*, 28(2):301–321, January 2002.

[13] J. Ims, Z. Duan, and Z.J. Wang. meshcurve : An automated low-order to high-order mesh generator. In *22nd AIAA Computational Fluid Dynamics conference*, 2015.

[14] S. Jakobsson and O. Amoignon. Mesh deformation using radial basis functions for gradient-based aerodynamic shape opitimization. *Computers & Fluids*, 36:1119–1136, 2007.

[15] X. Jiao and N.R. Bayyana. Identification of c^1 and c^2 discontinuities for surface meshes in CAD. *Computer-aided Design*, 40:160–175, 2008.

[16] X. Jiao and D. Wang. Reconstructing high-order surfaces for meshing. *Engineering with computers*, 28:361–373, 2012.

[17] P.M. Knupp. Algebraic mesh quality measures for unstructured initial meshes. *Finite Elements in Analysis and Design*, 39:217–241, 2003.

[18] Xiaoye S. Li. An overview of SuperLU: Algorithms, implementation, and user interface. *ACM Transactions on Mathematical Software*, 31(3):302–325, September 2005.

[19] X. Liu, N. Qin, and H. Xia. Fast dynamic grid deformation based on Delaunay graph mapping. *Journal of Computational Physics*, 211:405–423, 2006.

[20] Xiaodong Liu, Yidong Xia, Jian Cheng, and Hong Luo. Development and assessment of a reconstructed discontinuous galerkin method for the compressible turbulent flows on hybrid grids. In *54th AIAA Aerospace Sciences Meeting*. American Institute of Aeronautics and Astronautics, 2016.

[21] E. Luke, E. Collins, and E. Blades. A fast mesh deformation method using explicit interpolation. *Journal of Computational Physics*, 231:586–601, 2012.

[22] X. Luo, M.S. Shephard, and J.-F. Remacle. The influence of geometric approximation on the accuracy of high order methods. *Rensselaer SCOREC report*, 1, 2001.

[23] P.-O. Persson and J. Peraire. Curved mesh generation and mesh refinement using lagrangian solid mechanics. In *47th AIAA Aerospace Sciences Meeting*, 2009.

[24] T.C.S. Rendall and C.B. Allen. Efficient mesh motion using radial basis functions with data reduction algorithms. *Journal of Computational Physics*, 228:6231–6249, 2009.

[25] Christopher Rumsey. Verif/3db: 3d bump-in-channel verification case. NASA Langley Research Center Turbulence Models website, `http://turbmodels.larc.nasa.gov/bump3d.html`.

[26] T. Toulorge, C. Geuzaine, J.-F. Remacle, and J. Lambrechts. Robust untangling of curvilinear meshes. *Journal of Computational Physics*, 254:8–26, 2013.

[27] L. Wang, D.J. Mavriplis, and W.K. Anderson. Adjoint sensitivity formulation for high-order discontinuous Galerkin discretizations in unsteady inviscid flow problems. *AIAA journal*, 48(12), 2010.

[28] Y. Wang, N. Qin, and N. Zhao. Delaunay graph and radial basis function for fast quality mesh deformation. *Journal of Computational Physics*, 294:149–172, 2015.

[29] Eric W. Weisstein. Cubic spline. From *MathWorld* – A Wolfram Web Resource `http://mathworld.wolfram.com/CubicSpline.html`.

[30] T. Xiao, N. Qin, D. Luo, and S. Deng. Deformable overset grid for unsteady aerodynamic simulation. In *54th Aerosapce Sciences Meeting, AIAA Scitech*, 2016.

APPENDICES

Appendix A

Delaunay tessellation algorithm

Here, the exact algorithm used for the Delaunay tessellation (triangulation/tetrahedralization) in the context of Delaunay graph mapping (DGM) methods is explained. The method broadly follows the one used by Bowyer [4]. Since we are interested in a fast tessellation, and we do not need boundary-conforming facets, we do not try to ensure a boundary-conforming tessellation.

The Delaunay process maximizes the minimum angle in a simplex. Implementation of the "Delaunay kernel" is done such that orientation of elements and faces is preserved.

```
Bowyer_watson(pointList,npoin):
{
  Initialize dynamic array of nodes, elements and facets (1).
  Form a super simplex, containing all points in pointList.
  Add super simplex to list of elements, its facets to the list of
     facets and its nodes to the list of nodes.
  //c! The content of the following loop is referred to as the
     Delaunay kernel.
  For point in pointList do:
  {
    Add point to list of nodes.
    Find the element hostElement containing point (2).
    Declare a list of badElements and imsert hostElement in it.
    Declare a list of candidateElements and insert the neighbors of
       hostElement into it.
    For elem in candidateElement do:
    {
      Check the Delaunay criterion for elem (3).
      If the Delaunay criterion is violated:
```

```
      {
        Remove elem from candidateElements.
        Insert elem into badElements.
        Insert neighbors of elem, which have not already been
           checked for the Delaunay criterion, into
           candidateElements.
      }
    }
    Declare a list voidPolygon.
    For each facet in facets:
    {
      If facet is not shared by only one element in badElements:
        Insert facet into voidPolygon.
      Else if facet is shared by two elements in badElements:
        Delete facet from list of facets.
    }
    For elem in badElements:
      Delete elem from list of elements.
    For facet in voidPolygon:
    {
      Form a new simplex joining the facet with the (new) point,
          while ensuring that the ordering of nodes of the simplex is
           such that it is positively oriented. (4)
      Form new facets corresponding to the facets of newly-created
          simplices; ensure the orientation of these facets as well
          as that each required facet is created only once. (4)
    }
  }
  For elem in elements:
    If any node of elem is a super node, ie., if any node of elem is
        one of the first three nodes in the list of nodes, delete
       elem from elements.
  Return the list of elements and nodes.
}
```

We mention a few notes about some of the steps stated above.

1. The array of nodes is a simple array of size npoin+3 or npoin+4 containing all input points

and the additional 3 or 4 points from the 'super simplex'. The elements array holds, for each element, the node-indices of its nodes and the element-indices of its neighboring elements. The face array holds, for each facet, the node-indices of its nodes and the element-indices of its left- and right-elements (left and right is determined by the right-hand rule and the ordering of nodes).

2. We have implemented a walk through elements starting at the last element created. The three barycentric (area or volume ratio) coordinates are computed for the new point with respect to each element in the walk path. If all three values are positive, the current element contains the point. If not, we find the minimum value among the three barycentric coordinates; suppose the minimum value corresponds to the local node i among 1,2,3 or 1,2,3,4. The next element to check is the one neighboring the face opposite to node i.

3. The Delaunay criterion consists of comparing the distance of the new point from element center and the element's circumcircle radius. If the former is greater, the point does not violate the Delaunay criterion for this element; otherwise, the Delaunay criterion for the element will be violated upon addition of the new point.

4. In this author's opinion, preserving the orientation (local node-ordering) of each new element and each new facet is the most challenging task in writing a Delaunay kernel. It is accomplished in this work by using boolean helper arrays to keep account of associations between nodes and new facets created. We take advantage of the fact that in case of a simplex, each facet can be uniquely associated with the node opposite to it.

Appendix B

Spline reconstruction of a piecewise-linear boundary

The exact process used to reconstruct a twice-differentiable (C^2) boundary from the piecewise-linear boundary of a linear 2D mesh is described. Each one-dimensional boundary facet of the two-dimensional mesh is reconstructed into a cubic spline. The basic framework of the method is taken from [29]. We assume there are N boundary facets and thus $N+1$ boundary nodes.

We want to reconstruct the ith boundary facet into a curve in $\mathbb{R}^2$ as

$$\boldsymbol{r}_i(t) \;=\; \boldsymbol{a}_i + \boldsymbol{b}_i t + \boldsymbol{c}_i t^2 + \boldsymbol{d}_i t^3, \quad t \in [0,1] \tag{B.1}$$

$t = 0$ corresponds to the starting point of the boundary facet while $t = 1$ corresponds to the ending point. We impose C^0, C^1 and C^2 continuity of the curve to form a square linear system. That is, for all $i \in \{0, 1, ..., N\}$

$$\begin{aligned}
\boldsymbol{r}_i(0) &= \boldsymbol{R}_i && \text{(B.2)}\\
\boldsymbol{r}_i(1) &= \boldsymbol{R}_{i+1} && \text{(B.3)}\\
\boldsymbol{r}_i'(0) &= \boldsymbol{r}_{i-1}'(1) && \text{(B.4)}\\
\boldsymbol{r}_i'(1) &= \boldsymbol{r}_{i+1}'(0) && \text{(B.5)}\\
\boldsymbol{r}_i''(0) &= \boldsymbol{r}_{i-1}''(1) && \text{(B.6)}\\
\boldsymbol{r}_i''(1) &= \boldsymbol{r}_{i+1}''(0) && \text{(B.7)}
\end{aligned}$$

where $\boldsymbol{R}_i$ denotes the coordinates of the ith boundary point. The "boundary conditions" are,

in case of a closed curve,

$$\boldsymbol{R}_0 = \boldsymbol{R}_{N+1} \tag{B.8}$$

$$\boldsymbol{r}'_0(0) = \boldsymbol{r}'_N(1) \tag{B.9}$$

$$\boldsymbol{r}''_0(0) = \boldsymbol{r}''_N(1) \tag{B.10}$$

and in case of an open curve

$$\boldsymbol{r}''_0(0) = 0 \tag{B.11}$$

$$\boldsymbol{r}''_N(1) = 0. \tag{B.12}$$

For assembling the 3 systems of equations for the x-, y- and z-components of the spline curve, we define $D_i := r'_i(0) \, \forall \, i \in \{1, 2, ..., N\}$ and $D_{N+1} := r'_N(1)$ where r is x, y and z in turn. We express the coefficients a_i, b_i, c_i and d_i in terms of the D_i and R_i (the coordinates of boundary nodes). Finally, we obtain, for an open curve

$$\begin{bmatrix} 2 & 1 & & & & & \\ & 1 & 4 & 1 & & & \\ & & \ddots & \ddots & \ddots & & \\ & & & & 1 & 4 & 1 \\ & & & & & 1 & 2 \end{bmatrix} \begin{bmatrix} D_1 \\ D_2 \\ \vdots \\ D_N \\ D_{N+1} \end{bmatrix} = \begin{bmatrix} 3(r_2 - r_1) \\ 3(r_3 - r_1) \\ \vdots \\ 3(r_{N+1} - r_{N-1}) \\ 3(r_{N+1} - r_N) \end{bmatrix}. \tag{B.13}$$

After a similar derivation, for a closed curve we obtain

$$\begin{bmatrix} 4 & 1 & & & & 1 & 0 \\ & 1 & 4 & 1 & & & \\ & & \ddots & \ddots & \ddots & & \\ & & & & 1 & 4 & 1 \\ 0 & 1 & & & & 1 & 4 \end{bmatrix} \begin{bmatrix} D_1 \\ D_2 \\ \vdots \\ D_N \\ D_{N+1} \end{bmatrix} = \begin{bmatrix} 3(r_2 - r_N) \\ 3(r_3 - r_1) \\ \vdots \\ 3(r_{N+1} - r_{N-1}) \\ 3(r_2 - r_N) \end{bmatrix}. \tag{B.14}$$