# Evaluation of Flux Correction on Three-dimensional Strand Grids with an Overset Cartesian Grid

Dalon G. Work
*Utah State University*

# EVALUATION OF FLUX CORRECTION ON THREE-DIMENSIONAL STRAND GRIDS WITH AN OVERSET CARTESIAN GRID

by

Dalon G. Work

A dissertation submitted in partial fulfillment
of the requirements for the degree

of

DOCTOR OF PHILOSOPHY

in

Mechanical Engineering

Approved:

UTAH STATE UNIVERSITY
Logan, Utah

2017

## ABSTRACT

Evaluation of Flux Correction on Three-dimensional Strand Grids with an Overset Cartesian Grid

by

Dalon G. Work, Doctor of Philosophy
Utah State University, 2017

Major Professor: Doug Hunsaker, Ph.D.
Department: Mechanical and Aerospace Engineering

The high-order flux correction method on strand grids is evaluated in an overset context, using a Cartesian grid to accurately resolve the wake of the bodies of interest. The flow in the strand grids is solved using the high-order flux correction method in the surface direction and high-order summation-by-parts operators in the strand direction. The off-body flow is solved using high-order summation-by-parts operators. The two meshes are coupled using an implicit hole-cutting method and interpolation at the interface. Fundamental studies are considered which demonstrate the effectiveness of high-order methods in solving practical flows of interest.

(105 pages)

## PUBLIC ABSTRACT

Evaluation of Flux Correction on Three-dimensional Strand Grids with an Overset Cartesian Grid

Dalon G. Work

Simulations of fluid flows over complex geometries are typically solved using a solution technique known as the overset meshing method. The geometry is meshed using grid types appropriate to the local geometry in a patchwork fashion, rather than meshing the entire geometry with one type of mesh. The strand-Cartesian approach is a simplification of this process. While high-order accurate solvers on Cartesian grids are simple to implement, strand grids are usually restricted to second-order accuracy, resulting in poor quality solutions. Flux correction is a high-order accurate solution method, specifically designed for use on strand grids. The flux correction method on strand grids is evaluated in conjunction with an overset Cartesian grid. Fundamental studies are considered which demonstrate the effectiveness of high-order methods in solving practical flows of interest.

For my wife.
Let’s never do this again. :-)
For my God, whose grace has given me a good life.

## ACKNOWLEDGMENTS

This work is supported by the Computational Research and Engineering for Acquisition Tools and Environments (CREATE) Program, which is sponsored by the U.S. Department of Defense HPC Modernization Program Office, by the Army Research Office Fluid Dynamics Program (grant no. W911NF-12-1-0008), and by the Office of Naval Research Sea-Based Aviation Program (grant no. N000141310827).

A huge thank you to Dr. Aaron Katz, for teaching me so much about Computational Fluid Dynamics and giving me the freedom to explore. Another huge thank you to Dr. Doug Hunsaker, who took over as my advisor with the departure of Dr. Katz. His support and positive outlook has kept me going to the finish.

Another huge thank you to my co-workers Ted Giblette and Yushi Yanagita, who helped me with so much to get to this point. I would not have been able to finish this without their support. I should probably take them out to lunch sometime. The support of my wife, parents, family, and friends has been invaluable. There are so many people that have helped me in some way during this time, that there is not space enough to list them all. I'll do my best to personally thank everyone.

Last of all, the Lord Jesus Christ has been the foundation of my life through this journey. His continued guidance and grace reminds me to focus on that which is most important: the people around me.

Dalon G. Work

CONTENTS

LIST OF FIGURES

## ACRONYMS

| | |
|---|---|
| AMR | automated mesh refinement |
| BDF | backward difference |
| CFD | computational fluid dynamics |
| CFL | Courant-Friedrichs-Lewy condition |
| FAS | full approximate storage |
| FC | flux-correction |
| FD | finite difference |
| FV | finite volume |
| LUSGS | lower-upper symmetric Gauss-Seidel |
| MMS | method of manufactured solutions |
| MPI | Message Passing Interface |
| PDE | partial differential equation |
| RANS | Reynolds-averaged Navier-Stokes |
| RMS | root-mean-squared |
| SA | Spalart-Allmaras |
| SAT | simultaneous-approximation terms |
| SBP | summation-by-parts |
| SLIP | symmetric limited positive |
| SPMD | single process multiple data |
| SSDC | self-satisfying domain connectivity |
| SST | shear stress transport |

# CHAPTER 1
# INTRODUCTION

Computational fluid dynamics (CFD) is a mature and useful tool for solving fluid dynamics problems in industry. The multitude of commercial and open source products that are available is a testament to the development of this field. Many product suites offer a wide-array of solver strategies, turbulence models, automated meshing capabilities, and multi-physics capabilities. These offerings are generally second-order accurate in space and time, which is adequate for many situations. Some of the most complicated flows are those involving multiple bodies in relative motion with each other, such as a full simulation of a helicopter landing on a ship at sea. This situation is considered to be a vortex-dominated flow, since most of the flow is determined by the tip vortices produced by the rotor blades. This situation combines the most complicated parts of CFD into a single simulation, including: turbulence modeling, shock capturing, vortex resolution, low- and high- Mach number regions, boundary layer resolution, multiple bodies, and relative motion. The simulation can be further complicated by the coupling of a structural code to determine the bending moments and motion of the rotor blades.

In solving vortex-dominated flows, established CFD practices produce too much numerical dissipation, which causes the strength of the tip vortices to weaken in an unphysical manner. Recent advances in turbulence modeling, including large-eddy simulation and direct numerical simulation, have also been shown to require reduced numerical dissipation. Accuracy could be maintained by refinement of the mesh, but eventually the mesh becomes too large for current hardware to handle. Methods with a higher order of accuracy (third or greater) are required to obtain realistic answers without excessive grid refinement. On Cartesian grids, high-order accuracy can be easily achieved through the application of higher-order Finite Difference (FD) methods or quadratic interpolation for Finite Volume (FV) methods.

Cartesian grids, however, are limited in the geometries they can be used to solve, because most geometries are not axis-aligned. This necessitates the use of unstructured grids which can handle complex geometries. Unfortunately, in general, high-order accurate methods on unstructured grids are not yet at a production level, despite much research at the academic level [1]. These methods have not yet reached production-scale levels due to three major barriers:

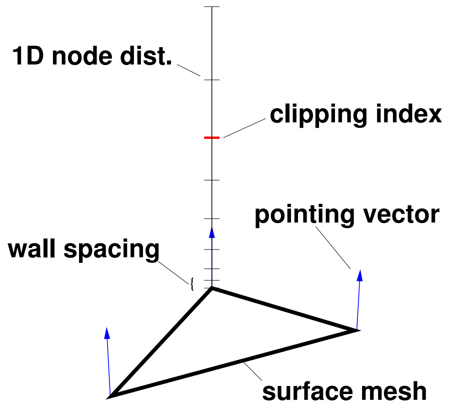
1. High-order methods are fundamentally different from the traditional low-order methods currently in use today. Implementation would require a huge investment in time and resources to write large portions of the software from scratch.
2. High-order methods tend to be less stable, requiring restrictive time steps and more computation time, not to mention user frustration when the software fails to work.
3. High-order methods typically require complicated and tailored schemes for handling shocks or high-gradient regions, or special cases which require preconditioning. Determining the best way for high-order methods to handle this is an active area of research.

Another important aspect of CFD is generating a quality mesh. While automated unstructured meshing is available, this feature typically requires much input from the user, especially in unsteady multibody situations. Creating a high-quality grid for complex flows can take meshing experts days or even weeks. Without increasing mesh automation, the percentage of time spent on meshing relative to solving will continue to increase.

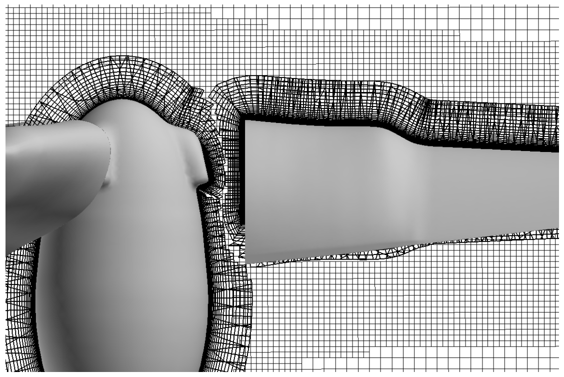
Typically, a rotorcraft simulation is solved using an overset methodology, which combines the meshing capabilities of unstructured grids with the efficient solutions of a Cartesian grid. The region close to a solid body (the near-body) is meshed using an unstructured or body-fitted grid. The near-body grid is completely enclosed by a Cartesian grid, which covers the whole domain. The unstructured grid is used to efficiently resolve the viscous boundary layer, while the Cartesian grid propagates important wake features. Thus, the entire domain is accurately resolved without excessive degrees of freedom, and the bulk of the flow is solved using efficient Cartesian grids. Meshing in this way is especially attractive when problems involve multiple bodies with relative motion to each other. Rather than create a single mesh that may not be able to accommodate all the bodies in an accurate way, each body can be meshed separately, without regard to the location of other bodies in the flow. An example of this can be seen in figure 1.1b. The price to pay for these advantages is in interfacing the multiple grids so all the grids form a single cohesive solution. This problem is known as the "domain connectivity" problem. With extremely large meshes, solving the domain connectivity problem can be difficult to implement in a scalable fashion on parallel hardware.

The strand-Cartesian approach has shown great potential to alleviate many of these difficulties [2–5]. Strand and Cartesian grids allow the possibility of automatic volume grid generation while enhancing scalability of the domain connectivity problem and the potential for high-order accuracy in the near-body mesh. Near solid bodies, the strand approach automatically generates a prismatic mesh along "strands." Strands emanate from pointing vectors determined from the surface tessellation, as shown in figure 1.1a. This near-body mesh is used to resolve viscous boundary layers and other physical effects occurring close to the body. Away from solid bodies, adaptive Cartesian grids resolve wake features with efficient high-order algorithms, as shown in figure 1.1b. The mesh generation process is robust and automatic, making the technique easily extensible to moving-body problems, for which the grid can be automatically regenerated at each time step. The two grids communicate through implicit overset interpolation [6–8]. This is simplified by the compact nature of the strand-Cartesian system. A typical three-dimensional strand-Cartesian grid system may be stored on all processors in a parallel computation, creating self-satisfying domain connectivity and reducing the time required for inter-grid communication [3, 9, 10].

Previous work has shown the viability of the strand-Cartesian approach, using second-order solvers in the strand grid, and high-order solvers in the Cartesian grid [3, 4, 10]. Originally, an unstructured solver was used in the strand grid, but this did not take advantage of the structure inherent in the strand grid. A custom second-order solver was then presented



(a) A strand projecting from a triangular surface element



(b) A strand-Cartesian overset grid on a TRAM rotor

Fig. 1.1: Strand grid definitions and an example of strand grids in an overset simulation

by Katz [11], with improved results. Even with the strand-specific solver, the strand mesh was unable to sustain vortex structures created at the solid bodies due to the high numerical dissipation of the second-order solution methods. This is shown in figure 1.2, which shows the vorticity of a rotortip vortex as it travels away from the tip into the freestream. Inside the strand grid, the vorticity decreases sharply, while the high-order Cartesian grid maintains the strength of the vortex.

This issue led to development of a strand-grid-specific high-order solution method, termed flux correction. Flux correction is a novel method of obtaining third- to fourth-order accuracy on strand grids. It was originally proposed by Katz and Sankaran [13] for use on two-dimensional unstructured grids. A second-order node-centered Galerkin finite-volume method is used as a starting point, to which truncation error canceling terms are added, increasing the order of accuracy. Theoretically, this means that an established second-order finite-volume code could be easily modified and "upgraded" to be higher-order. The method requires no additional flux quadrature or second derivatives, and requires minimal computational overhead beyond second-order schemes. Following this, flux correction was extended to include viscous terms by Pincock and Katz [14], which was then combined with summation-by-parts (SBP) operators by Katz and Work for use on strand grids [15].

Previous work by Tong [16–19], Thorne [20], and Yanagita [21] have demonstrated the

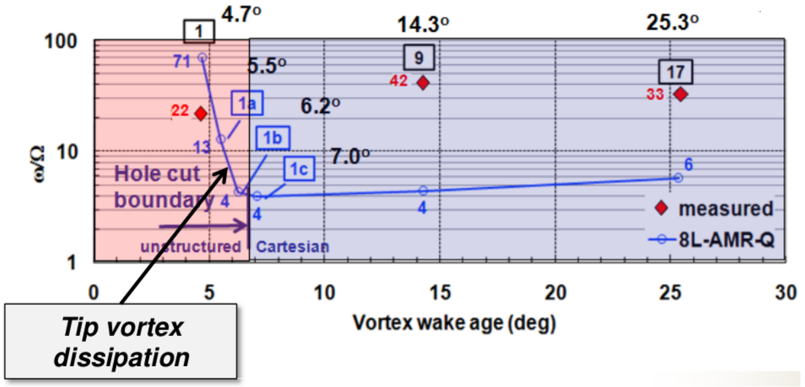


Fig. 1.2: Plot of the vorticity inside a trailing vortex of a rotor blade. The plot starts at the rotor tip, and travels through the strand grid into the Cartesian grid. (Courtesy of Andrew Wissink [12])

viability of the high-order flux correction method on strand grids. These references show high-order accuracy with multiple turbulence models, transsonic flow, and incompressible flow. Each of these showed that flux correction provides better accuracy with minimal computational overhead, and little modification to established second-order finite-volume practices. However, inaccuracies in the results revealed that strand grids need to be used in an overset context, as they do not resolve wake regions well [16, 22]. This work details the development of a third-order Cartesian solver and accompanying domain connectivity. The Cartesian solver is developed separately from the strand solver, while the domain connectivity is written in a third software package. Thus, both the Cartesian and strand solvers can be used separately if necessary. For overset applications, the domain connectivity library is used to tie the two solvers together. The solvers and the library are written using a Single Process Multiple Data (SPMD) parallel paradigm, implemented with the Message Passing Interface (MPI) in C++. The flux correction method on strand grids is then demonstrated on simple geometries, and results are used to determine how flux correction responds to the presence of the overset Cartesian grid. Flux correction is evaluated on accuracy, convergence, and computational time.

# CHAPTER 2
# LITERATURE REVIEW

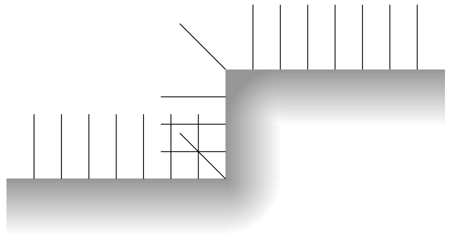
## 2.1 Strand Grids

The strand-grid methodology, first introduced by Meakin [3], is a novel approach to overset grid methods. Instead of using body-fitted or unstructured grids, a prismatic volumetric near-body grid is formed from an unstructured surface tessellation by extending straight lines (strands) in the normal direction away from solid bodies. This is demonstrated in figure 1.1a. Each strand contains the same nodal spacing along its length, with adjacent strands connected to form the prismatic grid around the body. The base node on the surface is termed the "root" of the strand, while the other end is the "tip". A strand consists of three distinct regions: the root node (node 0), the field nodes, and the fringe nodes. The root and field nodes are used to solve the flow. The fringe nodes are used for the domain connectivity, and receive interpolated values from the Cartesian grid. The division between field and fringe nodes is demarcated by the "clipping index," which indicates the last node to be used for solution. Generally, a progressive nodal spacing is used along the strand, in order to effectively capture the boundary layer around the solid body.

The advantages to this particular mesh formulation are many, especially when viewed in a parallel context. By using a progressive spacing along the strands, the tips of the strands provide a reasonable "interface" spacing for a block-structured AMR Cartesian grid to refine to. The refinement of the Cartesian grid around the strand grid can be made fully automatic. Since strand grids are based solely on the surface geometry, the entire volumetric meshing process can be made fully automatic. Due to the structure in the strand direction, it is possible to create efficient solution algorithms [13]. Finally, because of the one-dimensional spacing along the strands, the entire in-memory description of the strand grid is essentially collapsed from a three-dimensional to a two-dimensional description. Any necessary volumetric geometry can be easily computed on the fly. This is significant, because this allows all processors in a parallel computation to retain in memory a view of the global mesh, thus simplifying the domain connectivity problem by reducing the amount of needed communication between processors. Meakin [3] terms this ideal situation "self-satisfying domain connectivity" (SSDC) (See Section 2.3.2 for more information). Efficient and automatic connectivity methods can also be developed to connect strand-to-strand and strand-to-Cartesian grids, reducing the time spent in domain connectivity [10]. This becomes even more important in unsteady moving body problems, where the connectivity must be reestablished at every timestep.

If the surface is smooth, strand grids provide adequate coverage of the immediate volume surrounding a surface. However, strand grids are extremely sensitive to discontinuities in the surface mesh from which they are generated. In concave regions, the strands can potentially overlap, while in convex regions, the volume coverage is extremely poor. Both of these situations are illustrated in figure 2.1a. To address this issue, an iterative smoothing procedure is used to "bend" the strands, forming a more uniform mesh. This smoothing procedure works well for convex corners, but does not always prevent self-intersections in concave geometries. In this case, the clipping index is used to clip the tip of the strands, preventing the intersections. In these clipped regions, the Cartesian grid is refined to fully cover the domain.

The initial tests of strand-grid capabilities were performed using NSU3D [23, 24], a second-order unstructured solver, which was coupled with a fifth-order inviscid-only Carte-


(a) Unsmoothed strand grid

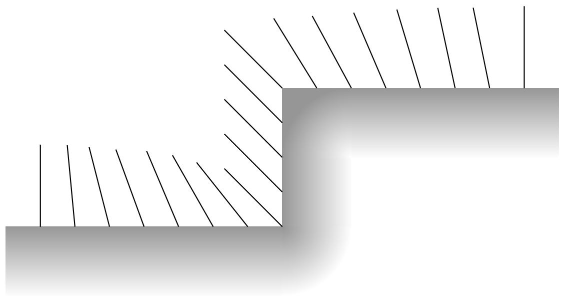

(b) Smoothed strand grid

Fig. 2.1: A strand grid before and after pointing vector smoothing

sian solver in the HELIOS [25, 26] framework. The strand-Cartesian hybrid was able to produce accurate results for various test cases, including laminar flow over a sphere, flow over circular and square cylinders, turbulent flow over a NACA 0015 finite wing, a V-22 TRAM rotor in hover, and a DLR-F6 Wing-body Transport [2, 3]. These test cases showed accurate results compared to experimental values and validated unstructured solvers.

However, these tests showed a severe sensitivity of NSU3D to the amount of pointing vector smoothing used in the strand grid. The smoothing introduced skewness into the grid, which affected the accuracy of the solution. The shedding frequency of a square cylinder at $Re = 250$ was used by Katz [2] to quantify the effect of strand smoothing on solution accuracy. With no smoothing, the coverage of the corner regions was poor, yet unexpectedly provided decent results. Extreme convergence of the smoothing operation resulted in worse results, incorrectly predicting a chaotic shedding behavior. The best results were found when the procedure was only converged to a small degree, striking a balance between orthogonality of the strands to the surface and volume grid coverage of the domain.

By utilizing the structure inherent in the strands, a robust and efficient second-order cell-centered strand solver was developed by Katz [2, 13]. This solver was developed specifically to handle the high-aspect-ratio cells generated by the strand grid, and exhibited improved convergence over NSU3D. The solution itself is based on implicit line Gauss-Seidel sweeps, with a fully non-linear agglomeration multigrid. The strand solver was found by Work [27] to be much more tolerant to the grid skewness caused by the smoothing procedure. Work also tested an alternative meshing procedure, which consisted of emanating multiple strands from a single location to improve the grid resolution. This multi-strand approach was found to suffer from accuracy and convergence issues. Research into a new meshing option, entitled strand shortening, is ongoing [28].

## 2.2 Flux Correction

Results from the rotorcraft simulations showed that the second-order accuracy of the near-body unstructured mesh severely limited the accuracy of the tip vortices [12]. (See figure 1.2.) This drove the effort to develop a high-order strand-specific solver. Katz and

Sankaran [13, 29] developed a third-order unstructured method for the Euler equations called flux correction. The method is based on a second-order node-centered finite-volume Galerkin method. The method required minimum changes to an already established code base, and does not require flux quadratures, second derivatives, or quadratic reconstruction techniques. The only requirement is gradients of the fluxes that are at a minimum, second-order accurate. Tests showed that the method converged at the same rate as the second-order method, and increased the required walltime by only 50%. The accuracy of flux correction was tested with a grid refinement study, which evaluated the drag coefficient as it approached the mathematical limit of zero, with flux correction showing an order of accuracy improvement over the second-order Galerkin method. Flux correction was successfully used to compute a shocked flow over an airfoil with no modification to the parent codebase. The flux correction method was soon extended to include viscous fluxes by Pincock and Katz [30]. In order for the same procedure to work with viscous terms, it was found that third-order accurate gradients had to be used. Through the method of manufactured solutions (MMS) and grid refinement studies, the viscous terms were found to converge in a fourth-order manner.

Katz and Work [31] modified the flux correction method to work on three-dimensional strand grids, creating an efficient and high-order solver which took advantage of the available structure in the strand. They accomplished this by viewing the strand grid as layers of prismatic cells. Each layer is solved individually as a two-dimensional unstructured problem using flux correction. These solutions along the strand are then coupled together using summation-by-parts (SBP) operators. A zero pressure gradient flat plate boundary layer test showed improved accuracy over the traditional second-order finite-volume method. As seen in previous tests, the flux correction convergence rate was similar to that of the finite-volume solver's convergence, with only a 50% increase in walltime. The flux correction method on strands also showed improvement in predicting the shedding behavior of a circular cylinder at low Reynolds numbers.

The negative Spalart-Allmaras (SA) [32,33] turbulence model was studied by Tong [16] in the context of flux correction. Results from two-dimensional grid refinement studies, including a zero pressure gradient flat plate and a bump-in-a-channel, showed that flux correction gave extremely accurate results on much coarser meshes than those employed by other established software packages. A grid refinement study using MMS demonstrated fourth-order accuracy with the fully coupled RANS-SA equations. Tong [17] next extended the SA turbulence model to three dimensions, followed by an implementation of the Menter-SST k-$\omega$ turbulence model [19]. This turbulence model is typically not used with high-order methods, due to oscillation of the turbulence variable $\omega$ and convergence issues. Tong was able to modify several limiting techniques which allowed for stable high-order solutions using flux correction and the Menter-SST $k$-$\omega$ turbulence model. A following paper [34] demonstrated the ability of flux correction to stably and accurately capture shocks through the implementation of the common SLIP limiter. This limiter was originally developed for unstructured finite-volume methods, and was easily added to the flux-correction method, providing stable high-order accuracy in the presence of shocks. Thorne [20] subsequently added preconditioning to flux correction, allowing the solution of incompressible and low-Mach number flows for arbitrary equations of state. Tong [18] followed up on this work, demonstrating high-order solutions of submersibles in water using both the SA and SST turbulence models. These results are significant because each of the schemes used were traditional finite-volume schemes which required little modification to implement with flux correction on strand grids.

All of the results presented in the above references were extremely consistent in their findings. First, MMS grid refinement studies showed the flux correction method to be third- to fourth-order accurate for all equations of state used. Second, convergence rates were identical or similar to the second-order Galerkin FV method that flux correction is based on, while flux correction on strand grids in three dimensions took approximately 30% longer in walltime. Third, the results clearly show that flux correction gives more accurate results than the Galerkin FV method on which it is based. In some cases, the difference in accuracy was substantial. Fourth, all results showed an obvious need for the Cartesian mesh to assist the strand mesh in resolving wake features. The purpose of this work is to couple the high-order near-body strand mesh with a high-order off-body Cartesian mesh, and demonstrate the viability of the strand-Cartesian methodology.

## 2.3 Overset Methods

When a flow involves multiple bodies which are moving relative to each other, meshing the domain becomes an extremely difficult and complicated problem. If an unstructured mesh is used, then it must be regenerated and possibly modified by hand for each timestep. In lieu of a single mesh type, a composite mesh formed from multiple overlapping grids can be formed. This type of mesh is shown in Figure 2.2. Unstructured or curvilinear grids are extended a short distance from the bodies, while a structured grid is placed in the background, filling the rest of the domain. This approach was first introduced by Steger [6], and simplifies the meshing process by using small, simple grid components. The flexibility of this method has led to the development of several open-source and commercial software packages which employ overset methods [23–26, 35–40].

The difficulty introduced by this method is in coordinating the individual grids to work together as a single cohesive mesh, and is termed the domain connectivity problem. Each node in a grid is given one of three designations: a normal node, a fringe node, or a hole node. Normal nodes have no overlapping grids, and are used to compute the flow solution normally. Fringe nodes exist wherever the boundary of one grid component overlaps another grid, and receive their value from an overlapping grid. Any node which is contained inside a solid body or is located deep inside an overlapping grid (and therefore is not a fringe node) becomes a hole node. The solution at hole nodes is not updated, and therefore has no effect on the flow.

The first step in the domain connectivity process is determining fringe and hole nodes, which is called the “hole cutting” problem. Many approaches have been developed to solve the hole cutting problem. These include brute force search methods, Cartesian bucketing methods (and the quadtree/octree/alternating digital tree derivatives), inverse hole maps,

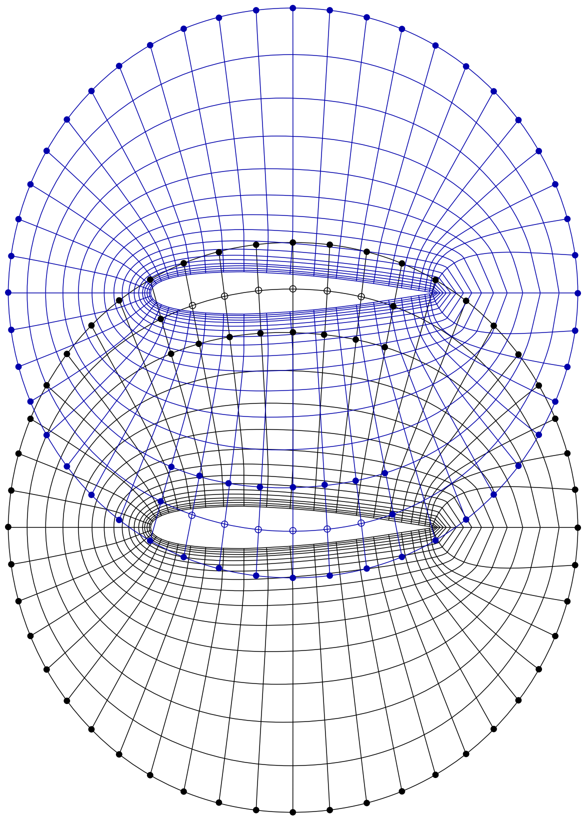


Fig. 2.2: Example of an overset mesh with a pair of airfoils and their O-grids. Hole nodes are indicated with open circles, while solid circles indicate fringe nodes. The Cartesian background grid is not depicted for clarity.

numerous different inside/outside tests, analytic shape cutting, X-Ray methods, and direct cut methods [41–44]. For more information on the advantages and disadvantages of these hole cutting techniques see the chapter by Meakin [45]. Each one typically requires a great deal of user interaction and special logic to deal with various extreme cases. Once hole points are correctly identified, the fringe nodes are identified by their proximity to the hole.

The second step is to determine, for each fringe node, a viable donor cell in another grid from which to receive interpolated solution values. When multiple grids overlap in a single region and multiple viable donor cells could be used, the question becomes more vague: Which donor cell will be best? This question is often left to the user, who must define which grids take priority.

Locating a viable donor cell uses various approximate methods to generate an initial guess for the cell which contains a given point. These approximate methods include alternating digital tree methods or stencil walking, where the results from the last node found are used as an initial guess for the current node. Using this initial guess, a gradient search method is used to locate that point within the cell. If the gradient search moves out of the cell, the gradient search indicates a direction to move in the grid, and thus indicates the next cell to try. Once the gradient search converges, a suitable donor cell has been identified. The cells/nodes used to interpolate can then be marked as such.

The third step is to define the interpolation stencils. The interpolation stencils used are typically second-order interpolations (bi- or tri-linear), defined using Lagrange or B-spline interpolations. Recent research, however, has shown that when using high-order solution methods, high-order interpolations are necessary to maintain the global order of accuracy [46–48]. Research has also shown that standard interpolations are not conservative, meaning they do not satisfy the conservation equations being solved [49]. This is especially important with high-gradient solutions and solutions with propagating shock waves. Meakin [50], however, argues that no amount of conservation will produce more accurate solutions, and that the most important factor in interpolation of values from one grid to another is that the average grid spacings of the two grids be approximately the same at the point of interpolation.

For completeness, it is mentioned that care must be taken to handle two pathological situations: orphan nodes, in which there are not enough nodes in a donating grid to interpolate to a fringe node on a receiving grid (making the fringe node an orphan with no solution), and coupled donor relationships, in which two fringe nodes which live on separate grids would be used to interpolate values to each other. Solving the orphan node problem can be difficult, and typically requires user intervention. The coupled donor node problem is easily fixed through the use of implicit interpolation, which forms a linear system of equations to determine the correct values of the fringe nodes.

### 2.3.1 Implicit Hole Cutting

In 2003, Lee and Baeder introduced a new hole cutting method, termed "implicit hole cutting" [8, 51]. They argue that explicit identification of holes in grids is unnecessary. If instead, the fringe nodes are identified, and that fringe is thick enough, it produces a wall of interpolated values which separate the interior flow from the exterior flow.

Implicit hole cutting redefines the first two steps of the traditional domain connectivity algorithm. Instead of identifying holes in grids, followed by fringe nodes, the method identifies possible donor cells for all nodes in a grid. If no viable donors are found, the node is assumed to be in a region of no overlap, and is marked as a normal node. If viable donors are found, the best quality donor is selected, and the node is marked as a fringe node. The cell quality is a user selected metric, and is typically the cell volume. In this case, a smaller cell volume is regarded as "better", because it is assumed the smaller cell has a more accurate answer. However, arguments for other metrics can be made.

Of course, testing every point in every grid against every other grid would be a time-consuming task. This is simplified by generating an "overlap" boolean matrix, which is a symmetric $n \times n$ matrix, where $n$ is the number of grids in the overset mesh. For any pair of grids $(i, j)$, the corresponding entry in the matrix will be 0 if they do not overlap, and 1 if they do. Thus, a grid $j$ need only be searched for donor cells if the $(i, j)$ entry equals 1, thus keeping the donor search cost from ballooning. For details on the generation of this matrix, see Lee [8].

Redefining the domain connectivity in this way avoids a whole host of problems related to hole cutting, especially with very complex geometry. This is because, with implicit hole cutting, the grids take precedence over the geometry. The grids, to some extent, smooth out the complicated features of the geometry, thus easing the domain connectivity problem. An example of the implicit hole cutting method is shown in figure 2.3 and figure 2.4.

Note that with this methodology, no nodes are marked as hole nodes. This means that nodes which are actually contained inside solid bodies (and would have been marked as hole nodes with a traditional domain connectivity method) are still used in the computations.

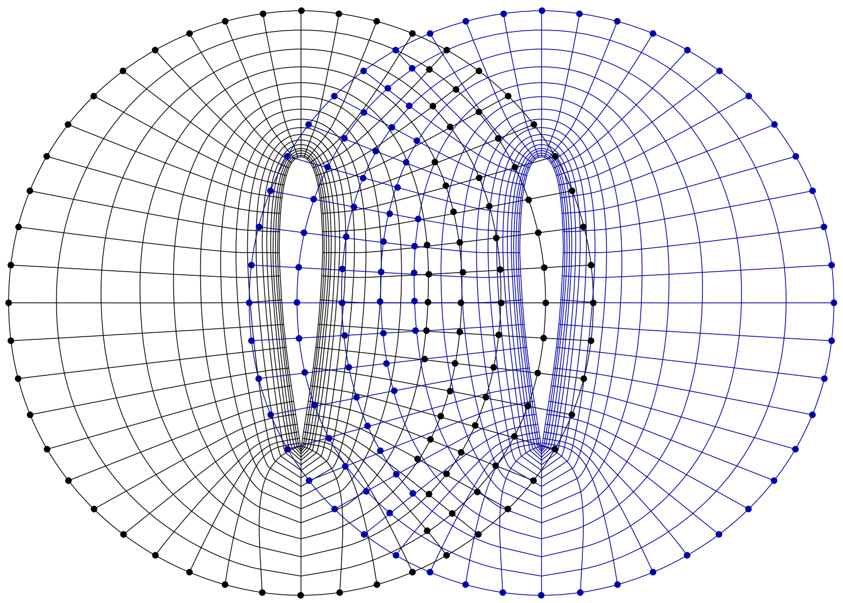


Fig. 2.3: Example of an overset mesh with a pair of airfoils and their O-grids. Solid circles indicate fringe nodes, chosen through the implicit hole cutting method. The Cartesian background grid is not depicted for clarity

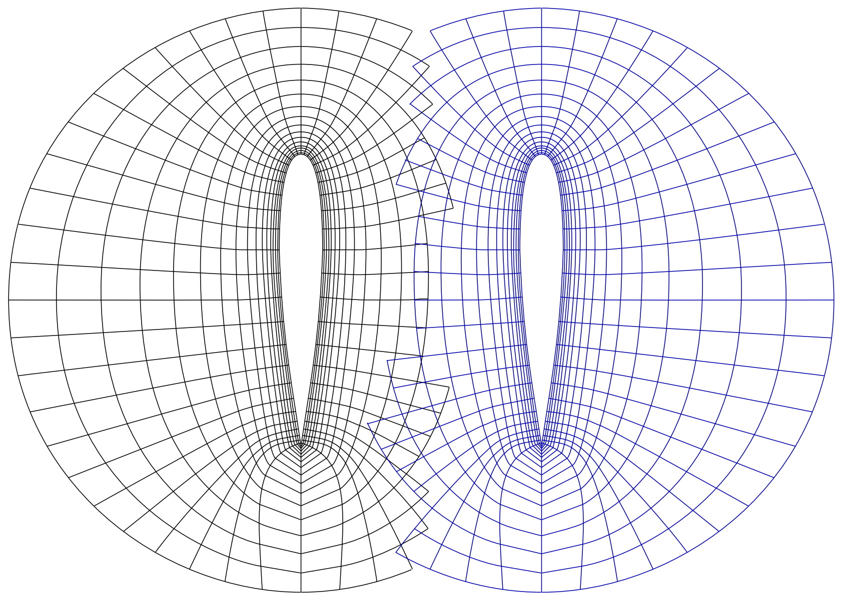


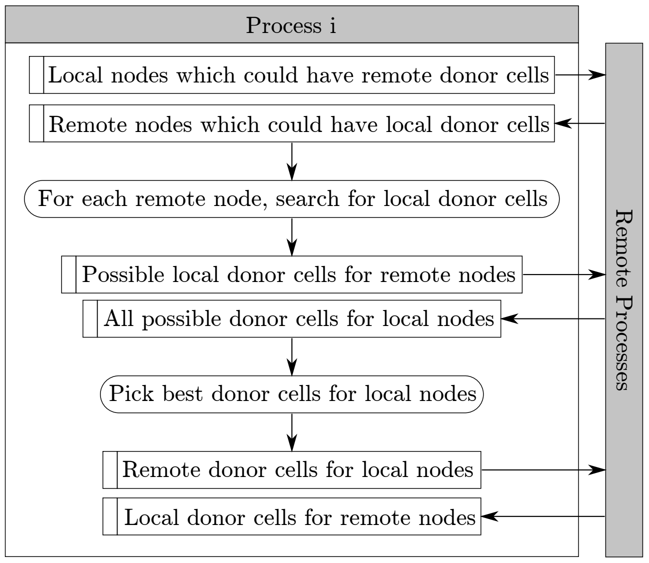
Fig. 2.4: Example of an overset mesh with a pair of airfoils and their O-grids. The "implicit" mesh resulting from the implicit hole cutting mesh. The Cartesian background grid is not depicted for clarity

Any computed value at these nodes is obviously non-physical. This may appear problematic, but as long as these “implicit hole” nodes are surrounded by a sufficiently thick band of fringe nodes, they will not impact the physical solution on the exterior of the solid body.

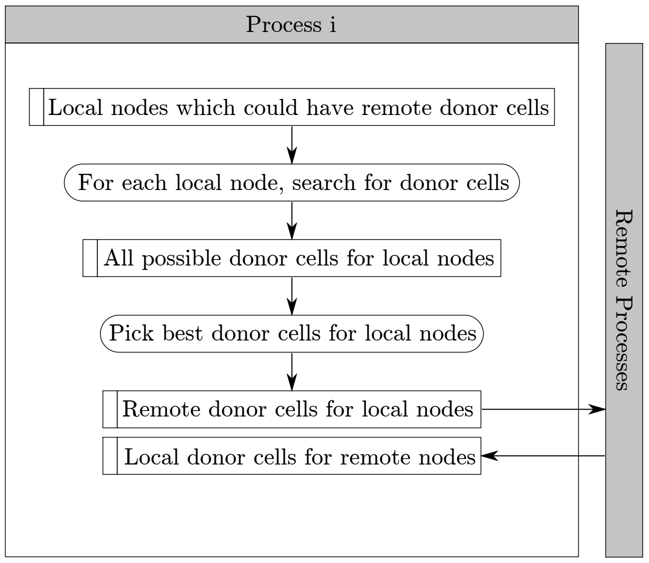
### 2.3.2 Parallel Considerations

The overset algorithms, as detailed above, have been described in a sequential manner, without regard to implementation in a parallel manner. When solving large, complicated geometries, an overset mesh can consist of hundreds of grids, each of which could contain thousands to millions of cells. Such a large mesh would not fit in the memory of a single processor, which complicates the domain connectivity problem. Landmann and Montagnac [52] published a general algorithm for performing implicit hole cutting in a distributed environment, which is summarized in figure 2.5a. However, if the global mesh is known to each process, the algorithm becomes much simpler, with far less inter-process communication, resulting in a more scalable algorithm, as shown in figure 2.5b.

This is one of the main advantages of using strand grids. By collapsing the memory requirements from volume grids to surface grids, the global mesh description, even for very large problems, can fit in the memory of a single processor, resulting in self-satisfying domain connectivity.


Process i
Local nodes which could have remote donor cells
Remote nodes which could have local donor cells
For each remote node, search for local donor cells
Possible local donor cells for remote nodes
All possible donor cells for local nodes
Pick best donor cells for local nodes
Remote donor cells for local nodes
Local donor cells for remote nodes
Remote Processes

(a) Global mesh is not known to each process


Process i
Local nodes which could have remote donor cells
For each local node, search for donor cells
All possible donor cells for local nodes
Pick best donor cells for local nodes
Remote donor cells for local nodes
Local donor cells for remote nodes
Remote Processes

(b) Global mesh is known to each process

Fig. 2.5: Data flow charts for performing basic implicit hole cutting in a distributed computing environment. Sender/Receiver pairs are determined via the block overlap matrix. When the global mesh is known to each process, complexity of the algorithm and the amount of communication is significantly less, increasing the scalability of the domain connectivity

# CHAPTER 3
# FLUX CORRECTION ON STRAND GRIDS

Flux Correction is a novel high-order method for solving conservation equations on general unstructured two-dimensional grids. At its core, flux correction is derived by taking an existing numerical method, performing a truncation error analysis of the numerical method, and identifying where the limiting order of error comes from. Following this, a higher-order version of the method can be derived by adding truncation error-canceling terms, thus improving the overall order of accuracy. In this chapter, flux correction, as it has been developed, is described. The chapter starts with a brief description of the conservation equations which are solved in this work. The explanation of flux correction starts with a brief foray into the difference between solution error and truncation error, and the relationship between them. This is followed by a simple one-dimensional description of flux correction, and then moves on to detail the implementation of flux correction on a two-dimensional triangular unstructured grid. From this foundation, the extension of flux correction to three-dimensional strand grids is made, with details of solution techniques used for solving the Navier-Stokes equations.

## 3.1 Equations of Motion

The Navier-Stokes equations can be written in a conservative vector format (using Einstein notation) as

$$\frac{\partial Q}{\partial t} + \frac{\partial F_j}{\partial x_j} - \frac{\partial F_j^v}{\partial x_j} = S \tag{3.1}$$

where $Q$ is the vector of conserved variables, $F_j = (F, G, H)$ is the inviscid fluxes, and $F_j^v = (F^v, G^v, H^v)$ is the viscous fluxes. The variables $Q$, $F_j$, and $F_j^v$ are defined as:

$$Q = \begin{pmatrix} \rho \\ \\ \rho u_i \\ \\ \rho e \end{pmatrix}, \quad F_j = \begin{pmatrix} \rho u_j \\ \\ \rho u_i u_j + p\delta_{ij} \\ \\ \rho h u_j \end{pmatrix}, \quad F_j^v = \begin{pmatrix} 0 \\ \\ \sigma_{ij} \\ \\ \sigma_{ij} u_i - q_j \end{pmatrix} \tag{3.2}$$

Here, $\rho$ is the density, $u_j$ is the $j$th component of the fluid velocity, $p$ is the pressure, $e$ is the total energy per unit mass, $h \equiv e + \frac{p}{\rho}$ is the total enthalpy per unit mass, $\sigma_{ij}$ is the deviatoric stress tensor, and $q_j$ is the $j$th component of the heat flux vector coefficient. The deviatoric stress tensor is defined as

$$\sigma_{ij} = 2\mu \left( S_{ij} - \frac{1}{3} \frac{\partial u_k}{\partial x_k} \delta_{ij} \right) \tag{3.3}$$

where $\mu$ is the dynamic viscosity, and $S_{ij}$ is the strain rate tensor, defined as

$$S_{ij} = \frac{1}{2} \left( \frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} \right) \tag{3.4}$$

The heat flux vector is obtained with Fourier's Law,

$$q_j = -c_p \left( \frac{\mu}{Pr} \right) \frac{\partial T}{\partial x_j} \tag{3.5}$$

where $T$ is the temperature, $c_p$ is the specific heat, and $Pr$ is the Prandtl number. In addition, Sutherland's Law is used to related dynamic viscosity and temperature, and the ideal gas equation of state is used. In this work, equation 3.1 is solved on both the near-body strand grid and the off-body Cartesian grid.

## 3.2 Truncation Error and Solution Error

Truncation error arises from the discretization of a continuous differential equation, and is distinct from the solution error, which is defined as the difference between the true solution and the discretized solution. The relationship between the two types of error is difficult to determine, but it can be shown that a relationship does exist.

Consider a general conservation law:

$$\frac{\partial Q}{\partial t} + \nabla \cdot F = 0 \tag{3.6}$$

where $Q$ represents the vector of conserved variables and $F$ is the flux. For a linear flux, the discretization of equation 3.6 becomes

$$\mathcal{D}\left\{Q^h\right\} = B \tag{3.7}$$

where $\mathcal{D}$ is the discretization operator, and $B$ incorporates the boundary conditions. The discrete solution $Q^h$ exactly satisfies this algebraic system of equations. If the exact solution $Q^e$ is substituted into the left-hand side of equation 3.7, an error term must be added to the right-hand side:

$$\mathcal{D}\left\{Q^e\right\} = B + E_t \tag{3.8}$$

where $E_t$ is the truncation error. The solution error, $E_s$ is, by definition, the exact solution minus the discrete solution, or $E_s = Q^e - Q^h$. Rearranging equations 3.7 and 3.8 and substituting into the solution error definition yields

$$\mathcal{D}\left\{E_s\right\} = E_t \tag{3.9}$$

From this we find that the truncation error and the solution error are related through the discretization operator. This can be viewed as the truncation error driving the solution error. The order of accuracy of the truncation and solution error are distinct. In general, there does not appear to be any way to prove the order of the solution error from the truncation error. Numerical observations have shown that the order of the solution error is never lower than the order of the truncation error.

## 3.3 Flux Correction in One Dimension

The node-centered grid in figure 3.1 is used to discretize the following hyperbolic differential equation:

$$\frac{\partial Q}{\partial t} + \frac{\partial F}{\partial x} = S(x) \tag{3.10}$$

where $S(x)$ is an arbitrary source term. Using the following definitions,

$$\Delta x_i = \frac{1}{2}\left(\Delta x_{i-1/2} + \Delta x_{i+1/2}\right) \tag{3.11a}$$

$$\Delta x_{i+1/2} = x_{i+1} - x_i \tag{3.11b}$$

$$\Delta x_{i-1/2} = x_i - x_{i-1} \tag{3.11c}$$

the discretization of this equation in a Galerkin fashion leads to a discretized flux derivative at node $i$:

$$\left.\frac{\partial F}{\partial x}\right|_i \approx \frac{1}{\Delta x_i}\left(F^h_{i+1/2} - F^h_{i-1/2}\right) \tag{3.12}$$

$F^h_{i\pm1/2}$ are numerical fluxes computed at the respective cell faces. The source term discretization is given by:

$$S^h_i = \frac{2}{3}S_i + \frac{1}{6\Delta x_i}\left(S_{i-1}\Delta x_{i-1/2} + S_{i+1}\Delta x_{i+1/2}\right) \tag{3.13}$$

### 3.3.1 Traditional Galerkin

In order to proceed, it is necessary to determine the numerical fluxes at the faces $i \pm 1/2$. In the traditional method, this is performed using

$$F^h_{i+1/2} = \frac{1}{2}\left(F^h_{i+1} + F^h_i\right) - D^h_{i+1/2} \tag{3.14a}$$

$$D^h_{i+1/2} = \frac{1}{2}\left|A^h(Q^R_{i+1/2}, Q^L_{i+1/2})\right|\left[Q^R_{i+1/2} - Q^L_{i+1/2}\right] \tag{3.14b}$$

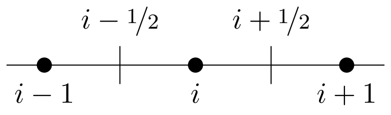


Fig. 3.1: Sample one-dimensional node-centered finite-volume grid

The dissipation term, shown in equation 3.14b, is added to enforce upwinding, making the method numerically stable. The flux jacobian is defined as $A = \frac{\partial F}{\partial Q}$ and is a function of the reconstructed state $Q^R$ and $Q^L$ at the face. The solution states at the face $Q^R$ and $Q^L$ are extrapolated using a truncated Taylor series

$$Q^L_{i+1/2} = Q^h_i + \frac{1}{2}\Delta x_{i+1/2}\left.\frac{\partial Q^h}{\partial x}\right|_i^h \tag{3.15a}$$

$$Q^R_{i+1/2} = Q^h_{i+1} - \frac{1}{2}\Delta x_{i+1/2}\left.\frac{\partial Q^h}{\partial x}\right|_{i+1}^h \tag{3.15b}$$

Computation of the gradient is assumed to approximate the true gradient to some order of accuracy $O(h^p)$. Substituting equation 3.14 into equation 3.12, the flux derivative at node $i$ becomes:

$$\left.\frac{\partial F}{\partial x}\right|_i \approx \frac{1}{2\Delta x_i}\left(F^h_{i+1} - F^h_{i-1}\right) - \frac{1}{2\Delta x_i}\left(D^h_{i+1/2} - D^h_{i-1/2}\right) \tag{3.16}$$

The method is now in a form that is convenient to determining the truncation error, the details of which can be found in other work, and are summarized here [11]. By representing the exact solution for $Q^R$, $Q^L$, $F_{i\pm1/2}$, and $S_{i\pm1}$ with Taylor series expansions centered at node $i$, it can be shown that the truncation error of the traditional Galerkin method is

$$\begin{aligned}\varepsilon = \left(\Delta x_{i+1/2} - \Delta x_{i-1/2}\right)\left(\frac{1}{2}\left.\frac{\partial^2 F}{\partial x^2}\right|_i - \frac{1}{3}\left.\frac{\partial S}{\partial x}\right|_i\right) \\ -\frac{1}{48\Delta x_i}\left(\Delta x^3_{i+1/2}\left|A_{i+1/2}\right| - \Delta x^3_{i-1/2}\left|A_{i-1/2}\right|\right)\left.\frac{\partial^3 Q}{\partial x^3}\right|_i \\ +O\left(h^3\right) + O\left(h^p\right)\end{aligned} \tag{3.17}$$

The limiting order comes from the first term, which is $O(h^1)$, and the last term, which is $O(h^p)$. The former originates in the central difference approximation of the flux and source terms, while the latter comes from the $p$-order approximation of the gradient of the solution variable. It can be shown that the solution order of accuracy for the Galerkin method is second-order on arbitrary grids [11]

### 3.3.2 Flux-Corrected Galerkin

Having identified the location of the limiting truncation error terms, a new definition of the fluxes is formulated to raise the order of the truncation error, and therefore, the solution error. Instead of using the traditional definition given in equation 3.14a, the following definition is used:

$$F^h_{i+1/2} = \frac{1}{2}\left(F^L_{i+1/2} + F^R_{i+1/2}\right) - D^h_{i+1/2} \tag{3.18a}$$

$$F^h_{i-1/2} = \frac{1}{2}\left(F^L_{i-1/2} + F^R_{i-1/2}\right) - D^h_{i+1/2} \tag{3.18b}$$

where $D^h_{i+1/2}$ is the same as equation 3.14b. Notice here that the *flux* is now being reconstructed to the midway point, and is done in a similar fashion as the solution variable $Q$:

$$F^L_{i+1/2} = F^h_i + \frac{1}{2}\Delta x_{i+1/2} \left.\frac{\partial F^h}{\partial x}\right|_i^h \tag{3.19a}$$

$$F^R_{i+1/2} = F^h_{i+1} - \frac{1}{2}\Delta x_{i+1/2} \left.\frac{\partial F^h}{\partial x}\right|_{i+1}^h \tag{3.19b}$$

The truncation error of equation 3.19 is dependent on the order of the gradient approximation, and not on the Taylor series truncation. The only step remaining is to determine an appropriate method of estimating the gradient at $x_{i+1/2}$ and $x_{i-1/2}$. In one dimension a compact second-order method can be derived from Taylor series as:

$$\left.\frac{\partial Q^h}{\partial x}\right|_i^h = \frac{\Delta x^2_{i-1/2} Q^h_{i+1} - \Delta x^2_{i+1/2} Q^h_{i-1} + \left(\Delta x^2_{i+1/2} - \Delta x^2_{i-1/2}\right) Q^h_i}{\Delta x_{i+1/2}\Delta x_{i-1/2}\left(\Delta x_{i+1/2} + \Delta x_{i-1/2}\right)} \tag{3.20}$$

The truncation error of the flux-corrected method is found to be

$$\varepsilon = -\frac{1}{24\Delta x_i}\left(\Delta x^3_{i+1/2} + \Delta x^3_{i-1/2}\right) \left.\frac{\partial^3 F}{\partial x^3}\right|_i + O(h^3) + O(h^p) \tag{3.21}$$

The order of each term is $O(h^2)$, $O(h^3)$ and $O(h^p)$. The only limiting factor is the order of the gradient computations. As long as the gradient is second-order accurate, then the truncation error will also be second-order accurate. The solution error is found through experimentation to be third-order on arbitrary grids [11].

A major advantage to the flux correction method is that it can be rewritten in terms of the traditional flux definition from equation 3.14a, and defined as a correction to the traditional flux, shown here:

$$F^h_{i+1/2} = \underbrace{\frac{1}{2}\left(F^h_i + F^h_{i+1}\right) - D^h_{i+1/2}}_{\text{Traditional flux}} \underbrace{- \frac{1}{4}\Delta x_{i+1/2}\left(\left.\frac{\partial F^h}{\partial x}\right|^h_{i+1} - \left.\frac{\partial F^h}{\partial x}\right|^h_i\right)}_{\text{Correction}} \tag{3.22}$$

This is important because it means that an already established code could be "upgraded" simply by adding in a single subroutine which computes the correction to the flux definition.

## 3.4 Flux Correction on an Unstructured, Triangular Mesh

The two-dimensional formulation is determined in a similar fashion to the 1D case. The hyperbolic equation in 2D is given as:

$$\frac{\partial Q}{\partial t} + \nabla \cdot \boldsymbol{F} = 0 \tag{3.23}$$

A triangulation around node 0 is shown in figure 3.2. Also depicted is the median-dual volume surrounding node 0, which is formed by connecting the centroids of the triangular elements with the midpoint of the lines bounding the triangle. An approximation of the median-dual volume can be made by only connecting the centroids of the triangular elements.

The divergence of a vector-valued function $\boldsymbol{\theta}$ at node 0 is found with

$$\nabla^h \cdot \boldsymbol{\theta}^h = \frac{1}{V_0}\sum_i \frac{1}{2}\left(\boldsymbol{\theta}^h_i + \boldsymbol{\theta}^h_0\right) \cdot \boldsymbol{n}_{0i} s_{0i} \tag{3.24}$$

where $\boldsymbol{n}_{0i} s_{0i}$ is the area-weighted normal of the median-dual face located between nodes 0 and $i$, with $s_{0i}$ being the area of the face [53]. Applying the vector divergence from

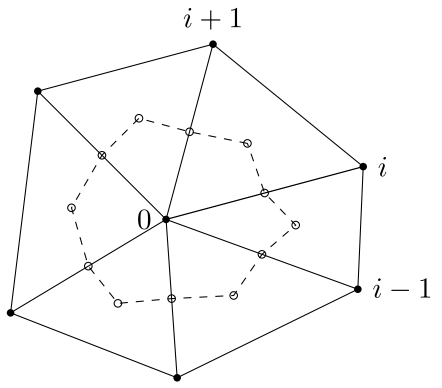


Fig. 3.2: Node-centered stencil on a two-dimensional unstructured triangular grid. Also shown is the median-dual volume surrounding node 0 (dashed lines)

equation 3.24 to the flux term of (3.23) results in

$$\nabla^h \cdot \boldsymbol{F}^h = \frac{1}{V_0}\sum_i \boldsymbol{F}_{0i}^h \cdot \boldsymbol{n}_{0i} s_{0i} = \frac{1}{V_0}\sum_i \mathcal{F}_{0i}^h s_{0i} \tag{3.25}$$

This is the starting point for both the traditional and flux correction methods. In the following, the scalar directed flux $\mathcal{F} = \boldsymbol{F}\cdot\boldsymbol{n}$ will be used, as it simplifies the notation. The traditional method approximates $\mathcal{F}_{0i}^h$ as:

$$\mathcal{F}_{0i}^h = \frac{1}{2}\left(\mathcal{F}_0^h + \mathcal{F}_i^h\right) - D_{0i}^h \tag{3.26a}$$

$$D_{0i}^h = \frac{1}{2}\left|A_{0i}^h\right|\left(Q_{0i}^R - Q_{0i}^L\right) \tag{3.26b}$$

where $A = \frac{\partial \mathcal{F}}{\partial Q}$ is the directed flux Jacobian, and $Q_{0i}^R$ and $Q_{0i}^L$ are the solution variables reconstructed to the midpoint of the edges connecting nodes 0 and $i$, which is

$$Q_{0i}^L = Q_0^h + \frac{1}{2}\Delta\boldsymbol{r}_{0i}^T \nabla^h Q_0^h \tag{3.27a}$$

$$Q_{0i}^R = Q_i^h - \frac{1}{2}\Delta\boldsymbol{r}_{0i}^T \nabla^h Q_i^h \tag{3.27b}$$

and $\Delta\boldsymbol{r}_{0i}$ is the position vector from node 0 to node $i$. There are many methods to compute the gradient of $Q$, and it can be shown that the truncation error of the complete method depends on the accuracy of the gradient approximation. The truncation error also depends on the form of the approximation for $F^h_{0i}$. Flux correction changes the definition of the flux to

$$\mathcal{F}^h_{0i} = \frac{1}{2}\left(\mathcal{F}^L_{0i} + \mathcal{F}^R_{0i}\right) - D^h_{0i} \tag{3.28}$$

which uses a reconstructed flux instead of an average flux:

$$\mathcal{F}^L_{0i} = \mathcal{F}^h_0 + \frac{1}{2}\Delta\boldsymbol{r}^T_{0i}\nabla^h\mathcal{F}^h_0 \mathcal{F}^R_{0i} = \mathcal{F}^h_i - \frac{1}{2}\Delta\boldsymbol{r}^T_{0i}\nabla^h\mathcal{F}^h_i \tag{3.29a}$$

This method can be cast into a "correction" of the linear flux, similar to the one-dimensional case, and used in equation 3.25. The correction is given as:

$$\mathcal{F}^h_{i+1/2} = \underbrace{\frac{1}{2}\left(\mathcal{F}^h_i + \mathcal{F}^h_{i+1}\right) - D^h_{i+1/2}}_{\text{Traditional Flux}} \underbrace{-\frac{1}{4}\Delta x_{i+1/2}\left(\left.\frac{\partial\mathcal{F}^h}{\partial x}\right|^h_{i+1} - \left.\frac{\partial\mathcal{F}^h}{\partial x}\right|^h_i\right)}_{\text{Correction}} \tag{3.30}$$

Experiments have shown that this method is third-order accurate on arbitrary unstructured triangular grids [31].

### 3.4.1 Gradient Approximation

The truncation error analysis above has shown that the accuracy of the gradients has a direct impact on the accuracy of the numerical method. Specifically, the gradient calculation must be a minimum of second-order accurate in order to increase the order of the truncation error.

Katz and Sankaran [11] employed a quadratic least-squares methodology in their original paper. However, least-squares methods have been shown to be sensitive to high aspect ratios and curvature of a mesh. This suggests that they will give erroneous gradients in a practical viscous mesh needed to resolve boundary layers.

Katz and Pincock [14] developed a new gradient method using element mappings. The reference triangle used is depicted in figure 3.3a, and is defined in an $(r, s)$ computational domain. The parent mesh triangles are subdivided into "sub-triangles," which are formed by placing nodes at equally spaced intervals inside the parent triangle. Examples of the subdivision are shown in figure 3.3. A Lagrange interpolation polynomial, as a function of $r$ and $s$, is applied to the reference triangle.

By estimating gradients in this manner, gradient stencils can be kept compact, promoting stability and solution speed. The subdivided mesh is used for the solution, while the parent triangles provide gradients at the nodes through the Lagrange polynomials. Since the Galerkin method is a continuous method, neighboring triangles will have multiple estimates for the gradients at edge and corner nodes. To make the method consistent, the multiple values at nodes are Jacobian-averaged:

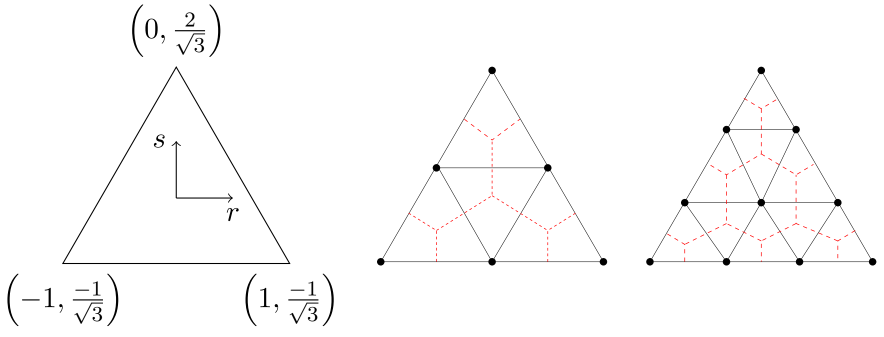
$$\left.\frac{\partial Q^h}{\partial x}\right|_i^h = \frac{\sum_{k\in i} J_k \left.\frac{\partial Q^h}{\partial x}\right|_k^h}{\sum_{k\in i} J_k} \tag{3.31a}$$

$$\left.\frac{\partial Q^h}{\partial y}\right|_i^h = \frac{\sum_{k\in i} J_k \left.\frac{\partial Q^h}{\partial y}\right|_k^h}{\sum_{k\in i} J_k} \tag{3.31b}$$

where $k$ are the various approximations to the gradient at node $i$. For linear elements, this reduces to a Green-Gauss procedure.

### 3.4.2 Viscous Terms

Quadratic gradients lead the inviscid terms to be globally third-order, but viscous terms stay second-order. Pincock [14] discovered that if cubic gradients are used, then the viscous


$\left(0, \frac{2}{\sqrt{3}}\right)$
$s$
$r$
$\left(-1, \frac{-1}{\sqrt{3}}\right)$
$\left(1, \frac{-1}{\sqrt{3}}\right)$

(a) Reference triangle (b) Quadratic (c) Cubic

Fig. 3.3: Reference triangle used in this work. Also depicted are second- and third-order triangles. Red dashed lines indicate the median-dual control volumes

terms jump to fourth-order. Cubic gradients caused the inviscid terms to become unstable on the boundary. This was resolved by using quadratic gradients on the boundary nodes for the inviscid terms, while using cubic gradients for the inviscid terms on the interior nodes and for the viscous terms throughout the domain. To form a quadratic gradient on a cubic triangle, overlapping quadratic triangles are extracted from the cubic triangle. These are shown in figure 3.4.

In general, viscous terms require special treatment for them to be stable and accurate. Positivity and stencil compactness have been shown to be necessary in any viscous discretization [54]. Pincock [14] investigated the stability of the viscous terms in the method and found that stability could be achieved by using the same element mappings as the inviscid terms, *without any Jacobian averaging*. With no Jacobian-averaging, stencil compactness is preserved, and the viscous terms remain stable. Similar to the inviscid flux, the viscous flux is defined as:

$$\mathcal{F}_{0i}^{v,h} = \frac{1}{2}\left(\mathcal{F}_{0i}^{v,L} + \mathcal{F}_{0i}^{v,R}\right) \tag{3.32}$$

with left and right corrected fluxes,

$$\mathcal{F}_{0i}^{v,L} = \mathcal{F}_{0}^{v,h} + \frac{1}{2}\Delta \boldsymbol{r}_{0i}^{T}\nabla^{h}\mathcal{F}_{0}^{v,h} \tag{3.33a}$$

$$\mathcal{F}_{0i}^{v,R} = \mathcal{F}_{0}^{v,h} - \frac{1}{2}\Delta \boldsymbol{r}_{0i}^{T}\nabla^{h}\mathcal{F}_{i}^{v,h} \tag{3.33b}$$

### 3.4.3 Source Terms

When present, source terms must also be discretized in a correct manner [55]. This can include unsteady time terms, turbulent production/destruction terms, or source terms from the method of manufactured solutions. The traditional Galerkin discretization for source terms in given as:

$$S_0^h = \sum_i \frac{1}{2}\left(S_0 + S_i\right) V_{0i} \tag{3.34a}$$

$$V_{0i} = \frac{1}{4}\Delta \boldsymbol{r}_{0i} \cdot \boldsymbol{n}_{0i} s_{0i} \tag{3.34b}$$

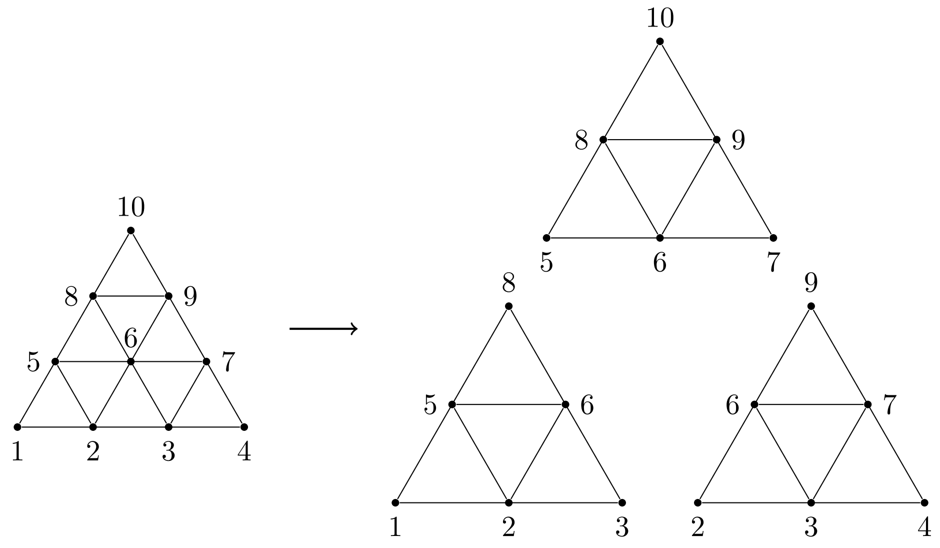


Fig. 3.4: Decomposition of a cubic element into three quadratic elements

This discretization can be shown to be second-order for irregular grids and third-order for regular grids. The traditional source terms, (equation 3.34), are replaced with a flux correction approximation,

$$S_0^h = \sum_i \frac{1}{2}\left(S^L + S^R\right)_{0i} V_{0i} \tag{3.35a}$$

$$S_{0i}^L = S_0 - \frac{1}{2}\Delta \boldsymbol{r}_{0i}^T \nabla^h S_0 - \frac{1}{8}\Delta \boldsymbol{r}_{0i}^T \boldsymbol{H}^h(S_0)\Delta \boldsymbol{r}_{0i} \tag{3.35b}$$

$$S_{0i}^R = S_i - \frac{1}{2}\Delta \boldsymbol{r}_{0i}^T \nabla^h S_i - \frac{1}{8}\Delta \boldsymbol{r}_{0i}^T \boldsymbol{H}^h(S_i)\Delta \boldsymbol{r}_{0i} \tag{3.35c}$$

where $\boldsymbol{H}^h(S_0)$ is the Hessian matrix of the source term. The gradient $\nabla S_0$ must be computed to the same $p$ order of accuracy as the fluxes, while the Hessian must be computed to $O(p-1)$ accuracy. To compute the Hessian, the second derivative is taken of the first derivative locally in each element. Multiple approximations on edge and corner nodes are Jacobian-averaged using equation 3.31b.

## 3.5 Flux Correction on Three-Dimensional Strand Grids

In three-dimensions, the Navier-Stokes equations are given by equation 3.1, which will be solved on a strand grid. Each stack of prismatic cells emanating from the surface is mapped to a standard computational space as shown in figure 3.5. In order to create high-order derivatives, the triangular base of a stack is divided into equally spaced sub-triangles in the $r$-$s$ plane. Quadratic elements are shown in figure 3.5. The distribution of nodes along the strands is mapped to an equally spaced distribution in the $\eta$-direction in the computational space, where $\eta \in [0, 1]$, with 0 at the root of the strand stack. The equations of motion are solved on all sub-triangles, with high-order derivatives of $r$ and $s$ computed using Lagrange mappings. Derivatives of $\eta$ are computed using high-order summation-by-parts (SBP) operators.

Equation 3.1 is transformed to the computational space, resulting in

$$\begin{gathered}
\frac{\partial \hat{Q}}{\partial t} + \frac{\partial \hat{F}}{\partial r} + \frac{\partial \hat{G}}{\partial s} + \frac{\partial \hat{H}}{\partial \eta} - \frac{\partial \hat{F}^v}{\partial r} - \frac{\partial \hat{G}^v}{\partial s} - \frac{\partial \hat{H}^v}{\partial \eta} = \hat{S} \\
\hat{Q} \equiv JQ, \quad \hat{S} \equiv JS, \\
\hat{F} \equiv J\left(r_x F + r_y G + r_z H\right), \quad \hat{F}^v \equiv J\left(r_x F^v + r_y G^v + r_z H^v\right) \\
\hat{G} \equiv J\left(s_x F + s_y G + s_z H\right), \quad \hat{F}^v \equiv J\left(s_x F^v + s_y G^v + s_z H^v\right) \\
\hat{H} \equiv J\left(\eta_x F + \eta_y G + \eta_z H\right), \quad \hat{F}^v \equiv J\left(\eta_x F^v + \eta_y G^v + \eta_z H^v\right) \\
\begin{pmatrix} r_x & s_x & \eta_x \\ r_y & s_y & \eta_y \\ r_z & s_z & \eta_z \end{pmatrix} = \frac{1}{J} \begin{pmatrix} y_s z_\eta - z_s y_\eta & z_r y_\eta - y_r z_\eta & y_r z_s - z_r y_s \\ z_s x_\eta - x_s z_\eta & x_r z_\eta - z_r x_\eta & z_r x_s - x_r z_s \\ x_s y_\eta - t_s x_\eta & y_r x_\eta - x_r y_\eta & x_r y_s - y_r x_s \end{pmatrix} \\
J = x_\eta(y_r z_s - z_r y_s) + y_\eta(z_r x_s - x_r z_s) + z_\eta(x_r y_s - y_r x_s)
\end{gathered} \tag{3.36}$$

Here, $J$ is the Jacobian of the transformation, $\hat{F}_j$ and $\hat{F}_j^v$ are the transformed inviscid and viscous fluxes, and partial differentiation is denoted with a subscript.

The high-order discretization method treats each layer in the prism stack individually. Two-dimensional median-dual volumes are constructed around each node in the $r$-$s$ plane. The sub-triangles and median-dual control volumes in a single surface element are shown as the black solid lines and red dashed lines in figure 3.3 for quadratic and cubic surface elements, respectively. To retain high-order accuracy, it is necessary to treat the $\eta$-derivatives correctly [16]. This is accomplished by moving them to the right-hand side and treating them as source terms. The physical time derivative is also moved, and a pseudo-time derivative is added on the left-hand side to employ a semi-implicit time-marching solution [16].

$$\frac{\partial \hat{Q}}{\partial \tau} + \frac{\partial \hat{F}}{\partial r} + \frac{\partial \hat{G}}{\partial s} - \frac{\partial \hat{F}^v}{\partial r} - \frac{\partial \hat{G}^v}{\partial s} = \tilde{S} \tag{3.37a}$$

$$\tilde{S} = \hat{S} - \frac{\partial \hat{Q}}{\partial t} - \frac{\partial \hat{H}}{\partial \eta} + \frac{\partial \hat{H}^v}{\partial \eta} \tag{3.37b}$$

This reduces the three-dimensional equations to a two-dimensional problem in the $r$-$s$ plane, which is discretized using the two-dimensional flux correction (now in the $(r, s)$ plane, instead of the $(x, y)$ plane) as described previously in Section 3.4. Each plane is coupled through the source term. As long as all terms in $\tilde{S}$ are computed at least to second-order accuracy, the $(r, s)$ solution will retain its high-order properties. SBP finite-difference operators are used to compute the required $\eta$-derivatives, which allow the scheme to retain high-order accuracy, stability, and discrete conservation in three dimensions [56–61]. Relevant details of the SBP operators are given in Appendix A.

## 3.6 Solution Techniques

Because flux correction is based on finite-volume methodology, a plethora of mature solution techniques already exist, including multigrid methods, adaptive pseudo-time step-

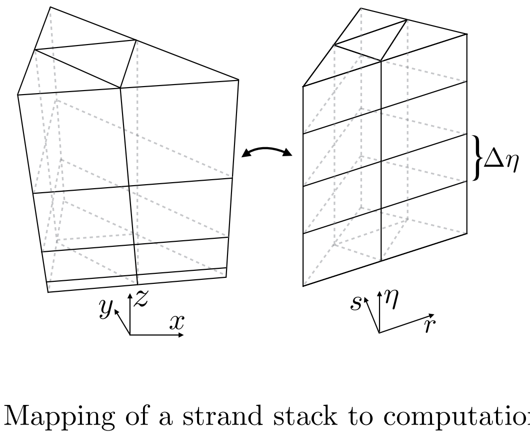


Fig. 3.5: Mapping of a strand stack to computational space

ping, and line-implicit methods. Each node in the strand grid is denoted by an index pair $(n, j)$: $n$, which is the surface node index, and $j$, which is the node along the strand which emanates from surface node $n$. The application of flux correction to equation 3.37 at a strand node $(0, j)$ results in a residual function $\hat{R}_{(0,j)}$, which is defined as

$$\hat{V}_0 J_{(0,j)} \frac{\partial Q_{(0,j)}}{\partial \tau} + \hat{R}_{(0,j)} = 0 \tag{3.38a}$$

$$\hat{R}_{(0,j)} \equiv \sum_{i \in 0} \left( \hat{\mathcal{F}}^h_{0i,j} - \hat{\mathcal{F}}^{v,h}_{0i,j} \right) - \tilde{S}^h_{(0.j)} \tag{3.38b}$$

where $\hat{V}_0$ is the median-dual volume of node 0 in the $(r, s)$ plane, $\hat{\mathcal{F}}^h_{0i,j}$ and $\hat{\mathcal{F}}^{v,h}_{0i,j}$ are numerical inviscid and viscous fluxes, computed at the median-dual edges between nodes 0 and $i$, and $\hat{\mathcal{F}} = \hat{A}_r \hat{F} + \hat{A}_s \hat{G}$ is the area-weighted directed flux in the $(r, s)$ plane. Nodes which lie on corners or edges will have multiple residual approximations from the neighboring elements. The residual approximations are simply added together to form the complete residual approximation for node $(0, j)$.

When running in a distributed computing environment, the mesh is partitioned element-wise. By computing nodal residuals on an element basis, a simple parallel algorithm can be used. After computing residuals on local elements, a parallel reduce function is used to sum all residual contributions on process boundaries. Residuals interior to process boundaries are implicitly summed as elements are processed.

### 3.6.1 Strand-Implicit Solution Method

The structure inherent in a strand grid allows for specialized solution techniques. First, the spacing along the strands allows for the use of finite-difference methods, and second, the strands themselves are obvious lines in the grid which can be used for a line-implicit scheme. With the line-implicit scheme, stiffness due to the high-aspect-ratio cells needed for turbulent boundary layers is relieved, while maintaining simplicity and scalability in the more isotropic unstructured layers of the grid. The line-implicit method used is a nonlinear Lower-Upper Symmetric Gauss-Seidel (LUSGS) implicit scheme [1,62–67]. This is combined with an explicit Runge-Kutta method in each unstructured layer of the strand grid [68]. The residual at node $(0, j)$ contains contributions from the other nodes in the unstructured plane $(i, j)$, and contributions from the nodes along the strand $(0, j \pm k)$. Following the notation of Tong [16], these contributions will be denoted as $Q_{\in 0}$ and $Q_{\in j}$, respectively.

$$R_{(0,j)} \equiv R_{(0,j)}(Q_{\in 0}, Q_{\in j}) \tag{3.39}$$

Following the nonlinear LUSGS procedure along strands, contributions along the strand are treated implicitly in a Gauss-Seidel procedure. Starting at the surface $(j = 0)$, each unstructured layer is solved with an explicit Runge-Kutta procedure. The next higher layer is then solved, utilizing the new data from the lower layer in the implicit procedure. This continues until the end of the strand are reached, at which point the sweep proceeds back down the strands, During the sweeps, the right-hand side always utilizes the latest available data, while maintaining only the left-hand side contributions from the current layer. The result is a block diagonal equation, each line of which reads

$$\hat{V}_0 J_{(0,j)} \frac{\partial Q_{(0,j)}}{\partial \tau} + \boldsymbol{D}_{(0,j)} \Delta Q^{\tau+1} = -R_{(0,j)}(Q^{\tau}_{\in 0}, Q^{\star}_{\in j}) \tag{3.40}$$

The $Q_{\in 0}$ terms are treated at the current pseudo-time station, while the $Q_{\in j}$ terms are treated at the latest available station, $\star$, and $\Delta Q^{\tau+1} = Q^{\tau+1} - Q^{\tau}$. The block diagonal is given by

$$\mathbf{D}_{(0,j)} = \frac{1}{2}\left(|\mathbf{B}_{(0,j-1/2)}| + |\mathbf{B}_{(0,j+1/2)}|\right) + \frac{\partial}{\partial Q_{(0,j)}} \left(\mathsf{D}_{2,\eta}(\mathsf{B}_{\eta}) Q^{p}\right) \tag{3.41}$$

where $\mathbf{B} = \frac{\partial \hat{H}}{\partial Q}$, $\mathsf{D}_{2,\eta}(\mathsf{B}_{\eta}$ is a SBP operator of the pure $\eta$ second derivative, and $Q^p$ is the vector of primitive variables. The second term can also be written as

$$\frac{\partial}{\partial Q_{(0,j)}} \left(\mathsf{D}_{2,\eta}(\mathsf{B}_{\eta}) Q^{p}\right) = \frac{\partial Q^{p,h}_{,\eta\eta}}{\partial Q_{(0,j)}} \tag{3.42}$$

where the comma in the subscript indicates differentiation.

### 3.6.2 Explicit Runge-Kutta Pseudo-time Stepping

The update equation is treated with an explicit $m$-stage Runge-Kutta scheme of Jameson [68]

$$
\begin{aligned}
Q^0 &= Q^\tau \\
Q^k &= Q^\tau + \Delta Q^k, k = 1, \ldots, m \\
Q^{\tau+1} &= Q^m
\end{aligned}
\tag{3.43}
$$

where $\tau$ is the pseudo-time counter, $k$ is the Runge-Kutta stage counter, and $\Delta Q^k$ is the $k$th stage update. Applying the Runge-Kutta algorithm to the implicit update equation yields

$$
\left[\frac{V_{(0,j)}}{\alpha_k \Delta\tau}\boldsymbol{I} + \boldsymbol{D}_{(0,j)}\right] \Delta Q^k = -R_{(0,j)}(Q^{k-1}_{\in 0}, Q^\star_{\in j})
\tag{3.44}
$$

at each node, where $\alpha_k$ is the Runge-Kutta coefficient for stage $k$. The left-hand side has been mass-lumped for convenience in the pseudo-time update. Before the updates are applied, they are smoothed in the $(r, s)$ plane with an implicit residual smoothing operation [69]. This is performed with two Jacobi iterations.

### 3.6.3 Implicit Physical Time Stepping

Unsteady time terms are treated using a $k$-step backward difference formula (BDF), which in general assumes the following form:

$$
\frac{\partial Q^h}{\partial t} = \frac{1}{\Delta t}\left(\gamma_1 Q^{n+1} + \sum_{i=0}^{1-k} \gamma_i Q^{n+i}\right)
\tag{3.45}
$$

where $\gamma_i$ depend on the order of the time derivative, and $\Delta t$ is the time step. The iteration in physical time is defined as $n$.

Pincock [14] studied two BDF formulations, BDF2 and BDF3. BDF2 is a second-order method and BDF3 is third-order. While third-order temporal accuracy would be more desirable, it quickly became unstable, and thus will not be used for this work. The coefficients for BDF2 are: $\gamma_1 = {}^1\!/_2, \gamma_0 = -2, \gamma_{-1} = -{}^1\!/_2$.

### 3.6.4 Multigrid

The mesh refining procedure gives a convenient agglomeration to be used in a multigrid solver. Each coarser level is derived by using a lower-order approximation of the parent mesh elements. For example, the finest mesh would use fourth-order elements, while the next two multigrid levels are coarsened to second- and first-order elements, respectively. Differing solver orders can also be used on the different levels, providing a combination $h$ and $p$ multigrid. The multigrid used in this work is the standard Full Approximation Storage (FAS) algorithm of Brandt [70, 71]. Restriction and prolongation operations are performed by interpolating solutions, residuals, and corrections using the available Lagrange interpolating polynomials over each element. Using these available interpolations allows for more accurate transfers then conventional averaging or injection procedures. Multigrid forcing terms are added on coarse levels in the standard fashion. This methodology was observed to provide good convergence acceleration for all test cases.

## 3.7 Numerical Approximation of Element Mappings

The transformation to computational space in equation 3.36 involves the computation of the appropriate mapping terms for each strand stack. Visbal [72] showed that if the computation of the mapping terms is not consistent with the numerical method, then the high-order accuracy will be lost. Tong [19] developed a method for use with flux correction on strand grids, which is summarized here.

Each of the mapping terms can be placed into "conservative form" by multiplying by the Jacobian: $\hat{r}_x \equiv J r_x$. The transformation leads to three constraints that must be discretely satisfied:

$$(\hat{r}_x)_r + (\hat{s}_x)_s + (J\eta_x)_\eta = 0 \tag{3.46a}$$

$$(\hat{r}_y)_r + (\hat{s}_y)_s + (J\eta_y)_\eta = 0 \tag{3.46b}$$

$$(\hat{r}_z)_r + (\hat{s}_z)_s + (J\eta_z)_\eta = 0 \tag{3.46c}$$

Only the first is discussed in detail, as the other two follow a similar method. equation 3.46

is discretized in a manner consistent with the flux correction method, including artificial dissipation and penalty boundary conditions. Each conservative mapping term is computed locally within each element

$$\hat{r}_x = (y_s z)_\eta - (y_\eta z)_s \tag{3.47a}$$

$$\hat{s}_x = (y_\eta z)_r - (y_r z)\eta \tag{3.47b}$$

The following discretization is then used to find $\eta_x$ at each node:

$$\begin{aligned}\sum_{i\in 0}\left[\frac{1}{2}\left(\hat{r}_x^L + \hat{r}_x^R\right)\hat{A}_r + \frac{1}{2}\left(\hat{s}_x^L + \hat{s}_x^R\right)\hat{A}_s\right]& \\ + \mathsf{D}_{1,\eta}\left(\boldsymbol{M}J\eta_x\right) - \frac{1}{2}D_{j+1/2}\left(\boldsymbol{M}J\eta_x\right) + \frac{1}{2}D_{j-1/2}\left(\boldsymbol{M}J\eta_x\right) - \text{penalty} = 0&\end{aligned} \tag{3.48}$$

$D_{j+1/2}\left(\boldsymbol{M}J\eta_x\right)$ represents artificial dissipation acting on $\boldsymbol{M}J\eta_x$, $\boldsymbol{M}$ is a source discretization operator, and "penalty" is a boundary penalty term. Solution for $\eta_x$ is straightforward.

## 3.8 Computation of $(r, s)$ Gradient

Moving the $\eta$-derivatives into the source term effectively created a two-dimensional problem in each layer. However, $\nabla_{(r,s,\eta)}$ is a three-dimensional operator, and naively applying the operator would include contributions from nodes along the strand when computing $r$ and $s$ derivatives. Mavriplis showed that including out-of-plane nodes potentially corrupts the gradient estimates for high aspect ratio grids with curvature [73]. Fortunately, strand contributions to $\nabla_{(r,s)}$ are small enough to be neglected, without incurring errors greater than the order of the mesh elements. The proof of this can be found in the work of Katz [15].

# CHAPTER 4
# CARTESIAN OFF-BODY AND ADAPTIVE MESH REFINEMENT

The Cartesian grid covers the rest of the domain in order to propagate any wake effects away from the body. The Cartesian mesh is implemented using the SAMRAI [74–76] framework, which uses a Berger-Collela [77] style multilevel AMR grid hierarchy. SAMRAI manages grid construction and data communication on the Cartesian grid hierarchy. figure 4.1 demonstrates a typical patch-based Cartesian hierarchy. Patches are defined by the global indices of the cells in the bottom-left and top-right corners of the patch. The global indices for each level start from the bottom-left corner of the domain. Each patch runs a single instance of the Cartesian solver. Cartesian grid generation starts with the coarsest level, which is defined over the whole domain. Finer patch levels are added around the strand grid and in wake regions, where important flow features will exist. Refinement around the strand grid is continued until the local Cartesian resolution is approximately the strand resolution, which is defined as the strand spacing at the local clipping index.

The inviscid flow in the off-body is solved using second and third-order node-centered SBP finite difference operators that are used in the near-body, while the viscous terms are solved using a second-order FV method, which is detailed in the dissertation by Nadarjah [78]. The equations of motion are solved with an explicit Runge-Kutta method in pseudo-time, with an implicit second-order Backwards Difference Formula used to integrate the physical time derivative. Due to the hierarchical nature of the Cartesian grid, a multigrid approach based on the Full Approximate Storage (FAS) algorithm of Brandt [70] is used to enhance convergence of the Cartesian off-body. In this method, solution variables and residuals on finer levels are restricted to the coarser levels after each pseudo-time step. On coarser levels where a finer level exists, a multigrid forcing term is computed and added to the residual. An iteration on the coarse level results in a corrected solution, which is prolonged back to the fine grid. This procedure is invoked recursively on all levels [27].

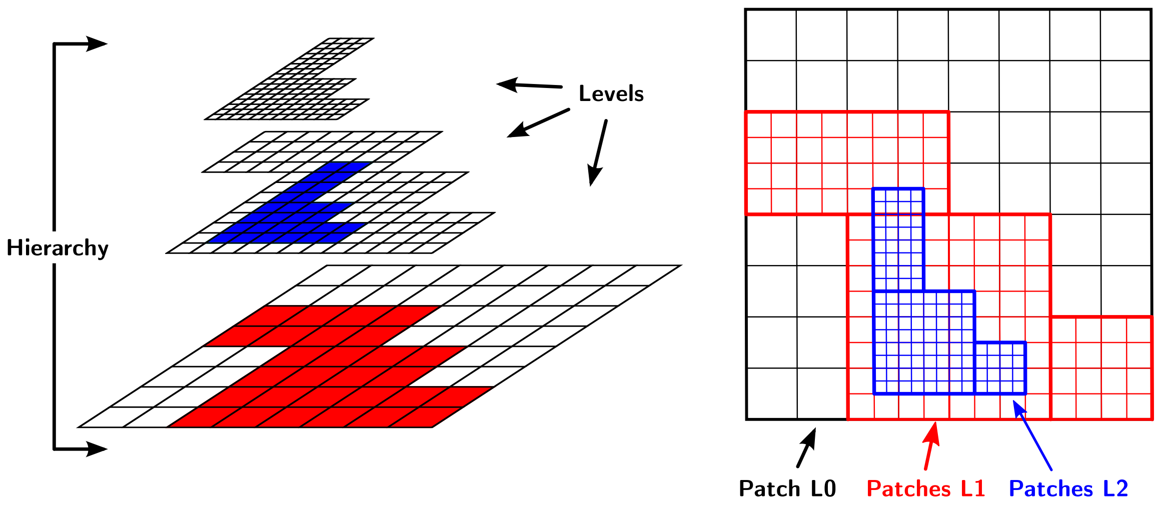


Fig. 4.1: Adaptive Cartesian Grid Hierarchy

These methods are standard, but must be modified because of the presence of the near-body mesh. The implicit hole-cutting strategy uses an `iblank` integer array to determine which Cartesian nodes are to be solved. This array follows this convention:

- Normal `iblank = 1`; no special treatment.
- Receiver `iblank = 0`; receives its solution values from the strand grid.
- Hole `iblank = 0`; Cartesian node located inside the solid body; no computation performed.
- Multigrid `iblank = -1` Cartesian node with an overlying fine grid; used for multigrid.

Prior to performing the domain connectivity, the iblank array is initialized to Normal. The nodes which have finer grids on top are then marked as Multigrid. During the domain connectivity, Multigrid nodes are skipped. If a Normal node is found to be a Receiver node or a Hole node, it is marked with `iblank = 0`. The Hole nodes are propagated down through the Cartesian levels. Following this, a stencil check for all Multigrid nodes is performed. These nodes must have complete stencils with no Receiver or Hole nodes in the stencil. If an incomplete stencil is found, these nodes are changed to `iblank = 0` because it cannot serve to compute flow. This procedure does not affect the finest levels, but it does cause the implicit holes on the coarser levels to grow. Consequently, this reduces the multigrid convergence around the strand-Cartesian interface, but the degradation has not been found to be significant.

## 4.1 Adaptive Mesh Refinement

The adaptive mesh refinement in the Cartesian solver is managed with the SAMRAI library. SAMRAI provides a "tagging" interface, which allows the user to write custom routines for marking certain cells for refinement. After the tagging is complete, SAMRAI uses that information to determine where to refine the grid and how to partition the grid among processes. SAMRAI also provides an interface for defining interpolations between coarse-fine grids, along with a few concrete implementations of commonly used interpolations. While SAMRAI allows for arbitrary fine-coarse refinement ratios, This work only used a 2:1 refinement ratio between Cartesian levels.

To minimize user input while still providing a suitable interface spacing, the Cartesian grid automatically refines around the strand grid. This is accomplished by computing the $\Delta s$ spacing at the clipping index of each strand, and then refining the Cartesian grid around this strand until $\Delta x$ of the finest Cartesian level is smaller than $\Delta s$. This ensures that the interface spacings between the grids are roughly the same. The tagging is extremely efficient, as determining the Cartesian cell a strand node is contained in is a simple algebraic equation.

Finding and refining around critical flow features is not a trivial task. What appears obvious to the human eye can be difficult to quantify, especially in determining the location of vortical structures in the flow. Several different methods for locating vortices can be used, including the $\Delta$ method, the $\mathcal{Q}$-criterion, the $\lambda_2$ method, and the $S$-$\Omega$ method [79]. The method used in this work is the $\mathcal{Q}$-Criterion:

$$\mathcal{Q} = \frac{1}{2}\left(\|\Omega\|^2 - \|S\|^2\right) \tag{4.1}$$

where $\Omega$ is the rotation rate matrix, $S$ is the strain rate matrix, and $\|A\|^2$ is the square of the Frobenius norm, which for an arbitrary real matrix $M$ is

$$\|M\| = \sqrt{\text{trace}\left(MM^T\right)} \tag{4.2}$$

Positive values of $\mathcal{Q}$ indicate that the rotation of the fluid is larger than the strain, while negative values are the opposite. By setting a positive threshold value, regions of vorticity can be found. Computing $\mathcal{Q}$ on the Cartesian grid is trivial, and cells where $\mathcal{Q}$ is larger than the designated threshold ($\mathcal{Q} > 30$ for all results presented) are tagged for refinement. The maximum level of refinement is controlled by a `maxLevels` parameter. More robust solutions using Richardson extrapolation to determine the maximum level of refinement are also possible [80].

# CHAPTER 5
# STRAND–CARTESIAN DOMAIN CONNECTIVITY

The domain connectivity between the strand and Cartesian meshes is based on a donor-recipient relationship and is accomplished with implicit hole cutting. The donor search is facilitated by the structure inherent in the strand and Cartesian grids and the ability to locally search the global mesh.

On the strand grid, identification of recipient nodes is controlled by the local clipping index. The clipping index for this work is set to 2. For surfaces with concave geometries though, the clipping index can be set lower to prevent intersections. All strand nodes above the local clipping index are located in the Cartesian grid index system via

$$I_s = \text{floor}\left(\frac{x_s - x_0}{\Delta x_l}\right) \tag{5.1}$$

where $\Delta x_l$ is the spacing for a given level. This gives the global Cartesian cell that the strand node falls in for that level. A search is then made through the global Cartesian mesh to determine if that cell exists in the currently defined patches. If the cell does not exist, then the next lower Cartesian level is used, until a viable Cartesian donor cell is found.

The Cartesian nodes needed to interpolate solution values to this strand node are added to a global Cartesian donor node list. Because SAMRAI decomposes the domain based on cells, the nodes on partition boundaries are duplicated across processors. It is possible, then, for a shared node to be marked as a donor node on one processor and not on another. When all strand nodes have been matched to a donor Cartesian cell, the global donor node list is synchronized across processes. Since donor nodes give their values to the strand grid, they cannot receive interpolated values from the strand grid.

## 5.1 Strand Donor Cells for Cartesian Nodes

Determining strand donor cells for the Cartesian nodes is a much more difficult proposition. Note that the techniques described here are not new. Wissink [10] has presented efficient techniques for finding donor cells in strand grids, which take advantage of the structure in the strand grid. This work is an investigation of flux correction on strand grids in an overset environment, and not an investigation into domain connectivity techniques. Hence, the methods presented here were chosen for ease of implementation, and are not the most efficient or scalable methods possible using strand grids.

On each process, an octree is built around the global strand grid. A description of the octree and the search algorithm can be found in appendix C. The octree is filled with the strand node's $(j, n)$ IDs and spatial coordinates. Next, a bounding box in the Cartesian index space is built around the global strand grid. Each process loops through local Cartesian nodes which are contained inside the bounding box, and, for each node, runs the algorithm shown in figure 5.1.

First, the octree is queried for a list of the $N$ closest strand nodes, which are sorted by distance, from closest to farthest. This list is then looped through, testing each strand node one at a time. If the strand node is a root node ($n = 0$), then it is assumed that the Cartesian node is contained inside a solid body, and it is set to be a Hole node. In practice, it is helpful to check if $n < k$, where $k$ is a small fraction of the total number of strand nodes along a strand. This is because the donor search occasionally has convergence issues near a surface, where the cells have extremely high aspect ratios. If the strand node is above the local clipping index, it is assumed that the Cartesian node is not contained in the strand mesh, and it is rejected as a Receiver node. Otherwise, a list of all the cells surrounding strand $j$ is pulled, and each one is tested in turn to determine if it contains the Cartesian node. Due to the use of high-order surface elements and the non-orthogonality of the strands with the surface geometry, simple cross products or basic inside/outside tests are not able to accurately determine whether or not a point lies inside of a strand stack. Instead, an optimization routine based on the strand cell Lagrange mappings is used to

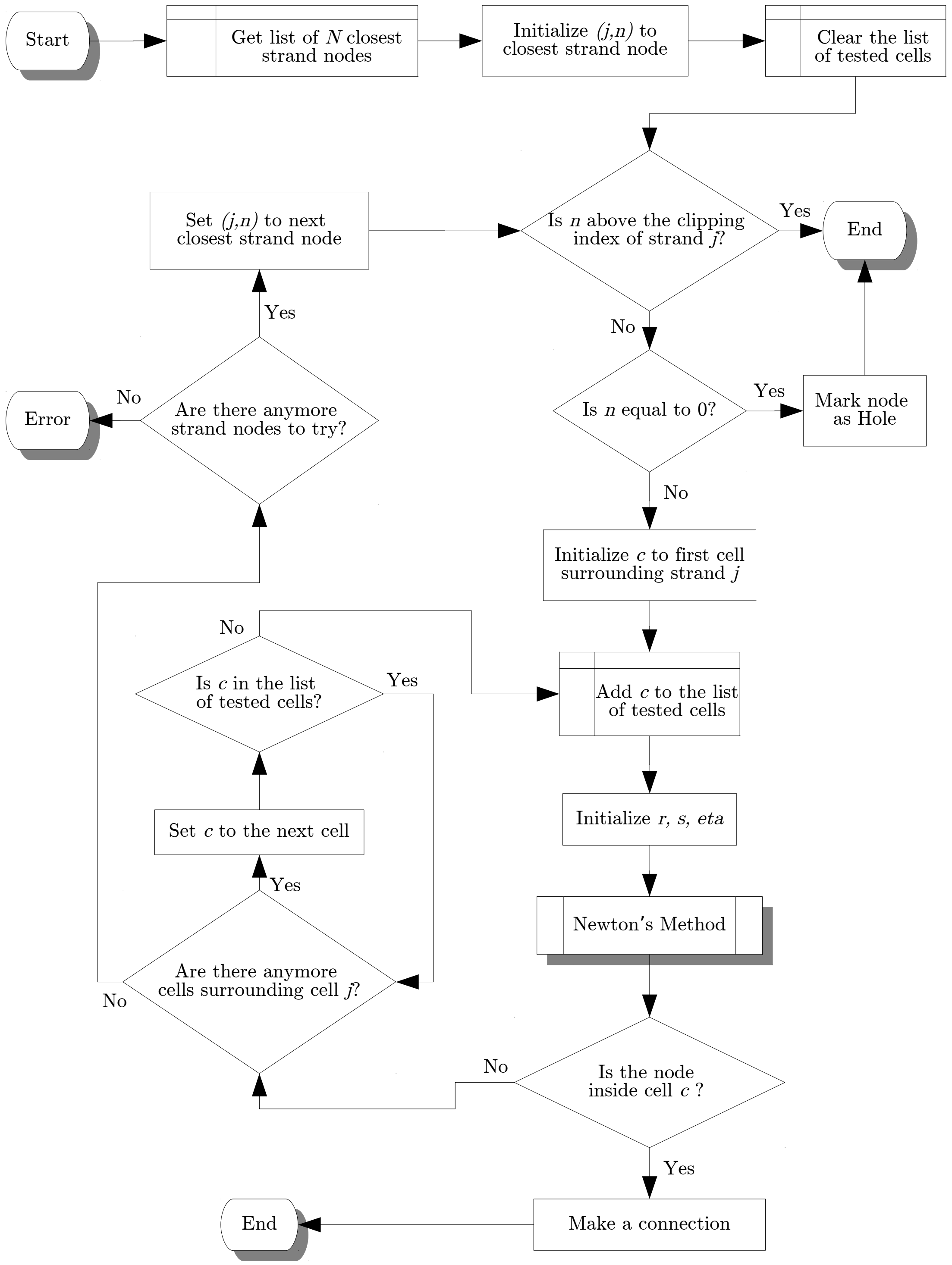

Start
Get list of $N$ closest strand nodes
Initialize *(j,n)* to closest strand node
Clear the list of tested cells
Is $n$ above the clipping index of strand $j$?
Yes
End
No
Set *(j,n)* to next closest strand node
Yes
Is $n$ equal to 0?
Yes
Mark node as Hole
No
Error
No
Are there anymore strand nodes to try?
Initialize $c$ to first cell surrounding strand $j$
No
Is $c$ in the list of tested cells?
Yes
Add $c$ to the list of tested cells
Initialize $r$, $s$, $eta$
Set $c$ to the next cell
Yes
Newton's Method
Are there anymore cells surrounding cell $j$?
No
No
Is the node inside cell $c$ ?
Yes
End
Make a connection

Fig. 5.1: Flowchart for the locating the strand donor cell for a Cartesian node

determine whether or not the Cartesian node is contained in a strand stack.

For each strand stack tested, the optimization routine returns true if the Cartesian node is located inside, or false otherwise. If true, then a connection between the strand cell and the Cartesian node is made, and the appropriate process is notified of the connection. If false, the next cell is tested. If all the cells surrounding strand $j$ have been tested, then the next strand node in the octree-returned list is tested. If all nodes have been tested, then an error is thrown, as the Cartesian node has not been located in the strand grid. In general, getting a list of 20 strand nodes from the octree is more than enough to find the donor cell. During the entire process, a list of tested cells is kept to prevent duplicate tests.

Interpolation from the Cartesian to strand grid is a simple tri-linear interpolation within the Cartesian cell containing the strand node. Interpolation from the strand to Cartesian grid is more involved. The $(r, s)$ element Lagrange polynomials form the base of the interpolation. Since no analytic mapping for $\eta$ exists, it is approximated at node $j$ with a one-dimensional Lagrange polynomial:

$$\phi(\eta) = \sum_{j=0}^{N-1} \phi_j \ell_j(\eta) \tag{5.2}$$

To interpolate within a strand cell, an outer product of $L_i(r, s)$ and $\ell_j(\eta)$ is taken, resulting in

$$\phi(r, s, \eta) = \sum_{i=0}^{N-1} \sum_{j=0}^{p} \phi_{ij} L_i(r, s) \ell_j(\eta) \tag{5.3}$$

Once the $r$-$s$-$\eta$ location of a Cartesian node is known, equation 5.3 is used to interpolate solution values to the Cartesian node.

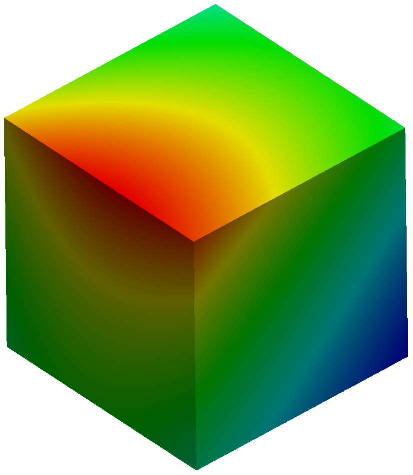
# CHAPTER 6
# COMPUTATIONAL RESULTS

In this chapter, results showcasing a working strand-Cartesian solver are presented. First, the Method of Manufactured Solutions (MMS) is used to verify the correctness of the Cartesian code, as well as to explore the accuracy of flux correction when used in an overset environment. Secondary orders of accuracy are then explored by evaluating the drag coefficient and pressure error on an inviscid sphere at a low Mach number. Following this, viscous flow over a sphere at a low Reynolds number using the combined strand-Cartesian solver is compared against experiment and against previous results with the strand grid alone. The Reynolds number is then increased into an unsteady shedding regime, where flux correction is validated against experiment, and against the industry code Star-CCM+. Finally, the ability of flux correction to maintain the strength of a vortex tube is investigated.

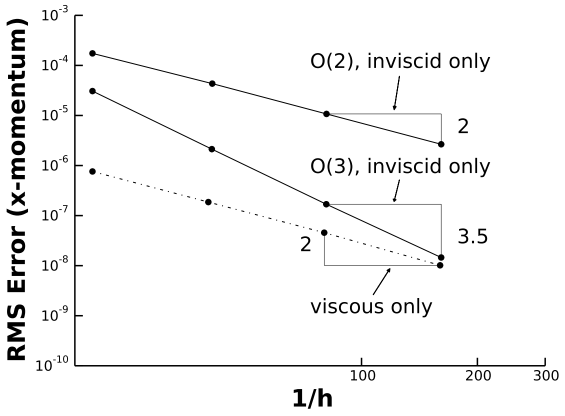
The results presented in this chapter lead to four main conclusions. First, that flux correction on strand grids does lead to an overall increase of solution accuracy. Second, that flux correction can provide that accuracy at a reasonable cost. Third, the accuracy of flux correction is diminished slightly by the overset method. Fourth, the convergence and stability of flux correction is lowered by the overset method.

## 6.1 Method of Manufactured Solutions

MMS is a method of verifying that the implementation of a numerical method is free of coding errors and gives the expected order of accuracy. It was originally described by Roache [81], and consists of choosing an analytic solution and substituting it into the differential equation to determine an appropriate source term. This source term is then used in the numerical method to force the method to converge toward the chosen analytic solution. A grid refinement study is then used to determine the order of accuracy of the numerical method. A plot of the exact density function on a Cartesian grid is shown in figure 6.1a.



(a) Cartesian density solution



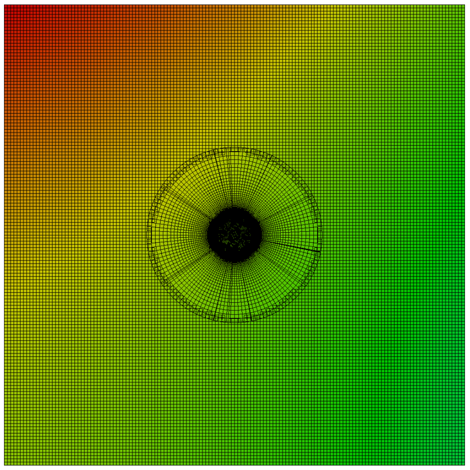
(b) Cartesian grid refinement study

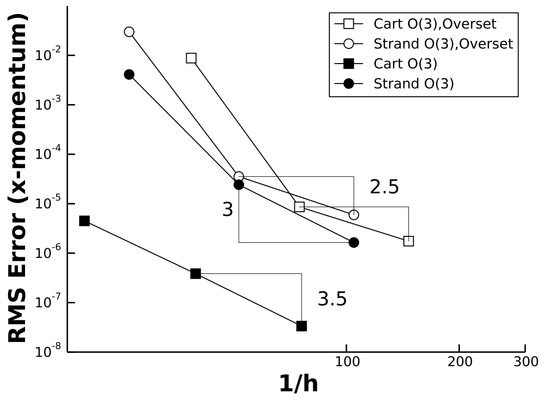
Fig. 6.1: Verification of Cartesian solver using MMS

The Cartesian code was independently verified using this method. The strand code has already been subjected to this test in previous work, with the flux correction method showing third-order accuracy for the inviscid terms and fourth-order accuracy for the viscous terms [16]. The Cartesian domain was set to be $x, y, z \in [0, 1]$, and the mesh was solved until the residuals fell below $10^{-13}$ on successively refined grids. Refinement was achieved by splitting the cells in half in each direction. The error was computed using an $L^2$ norm for each grid. Results for the Cartesian solver are shown in figure 6.1b, showing total error in the $x$-momentum equation, plotted against $\frac{1}{h}$, where $h$ is the characteristic cell size. Second-order and third-order accurate SBP operators were examined for the inviscid terms, while the second-order FV discretization was tested for the viscous terms. Both the viscous terms and the second-order SBP operators are second-order accurate, while the nominally third-order accurate SBP operators show an order of 3.5. This is most likely due to the dissipation operators, which are one order more accurate than the inviscid operators.

In order to better understand the effect of the overset meshing technique, this same test is run with a combined strand-Cartesian mesh, and solved using the Euler equations. Both the strand and Cartesian grids were refined systematically, and all cases were run until all residuals converged to $10^{-13}$. The strand grid was solved using flux correction, while the Cartesian grid was solved with the third-order operators. The mesh and resulting orders of accuracy are shown in figure 6.2. The order for the strand and Cartesian grids are shown when solved using an overset method, and when solved alone, without the presence of the other grid. The total RMS error for the $x$-momentum is shown.

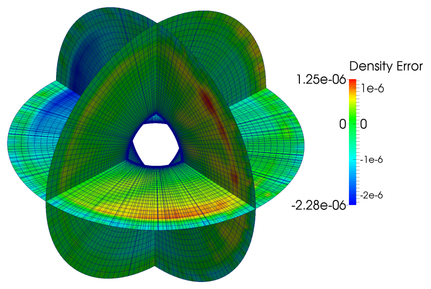
On their own, each grid shows typical asymptotic behavior. The strand grid asymptotes to third-order, and the Cartesian grid asymptotes to an order of 3.5. Together, however, more complex behavior appears. Overall, the total solution error increases by orders of magnitude, while the order of accuracy for both the strand and Cartesian solvers drops to 2.5. This shows that the second-order interpolations do not globally preserve the order of


(a) Strand-Cartesian density solution

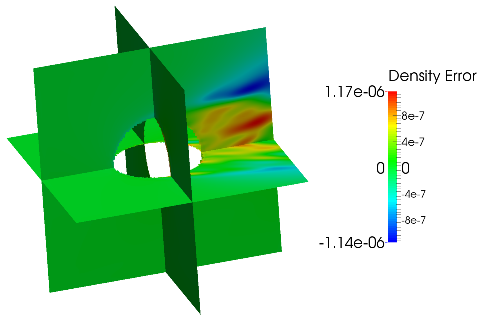


(b) Overset strand-Cartesian solver

Fig. 6.2: X-momentum error and order of accuracy studies using the Method of Manufactured Solutions



(a) Density errors in the finest strand grid



(b) Density errors in the finest Cartesian grid

Fig. 6.3: Slices of the strand and Cartesian MMS solution, showing solution error in the density

accuracy for either the strand or Cartesian solvers. Since the exact solution is known, it is also possible to view the local error in the final solution. The local error in the density solution is shown in figure 6.3. The errors in the strand grid are localized toward the tips of the strand, thus showing the impact of the interpolations there. The errors in the Cartesian grid are located downstream of the strand grid. This clearly demonstrates the need for high-order solution methods in any near-body grid. Off-body Cartesian solvers are typically extremely high-order, yet all that accuracy is for naught if it is given a poor starting point from an overset mesh.

## 6.2 Inviscid Sphere

Inviscid, incompressible flow, also known as potential flow, over a body of arbitrary shape produces zero drag on the body. For this test, the convergence of the drag coefficient of a sphere is evaluated with a grid refinement study. In particular, three grid refinement studies are performed. The first two are with the strand grid alone; one solved second-order, and the other with flux correction. These provide a baseline comparison for the third grid refinement study, which is combined with the Cartesian grid and run high-order. The Mach number was set to M=0.15, and the solution initialized to the exact potential solution, with the freestream flow in the $z$-direction. The exact solution, defined in spherical coordinates is

$$\begin{aligned} u_r &= w\left[1-\left(\frac{a}{r}\right)^3\right]\cos\theta \\ u_\theta &= -w\left[1+\frac{1}{2}\left(\frac{a}{r}\right)^3\right]\sin\theta \end{aligned} \tag{6.1}$$

where $a$ is the radius of the sphere. Using this as a starting point, the solution was run until the force on the sphere converged. For the grid refinement studies, a coarse, medium, and fine mesh were used. For the stand-alone strand grid cases, these were 64x32, 256x64, and 1024x128, respectively, where the first number indicates the number of fourth-order elements on the sphere's surface, and the second number is the number of strand nodes used. The strands were extended to a length of 10 diameters, and a progressive spacing along the strand was utilized. The wall spacing was also cut in half with each grid refinement.

For the overset case, the three meshes used were 64x18, 256x32, and 1024x64. The strands were extended to a length of 1.5 diameters. The Cartesian mesh was extended from $(-5,-5,-5)$ to $(5,5,5)$, with a $\Delta x$ spacing to match the spacing at the strand tips. An image of the coarse mesh with pressure contours is shown in figure 6.4. The drag coefficient and the pressure error on the surface of the sphere are used to evaluate the effect of the Cartesian grid on the surface of the sphere. The exact solution for the pressure on the surface of the sphere is given by

$$C_P = 1 - \frac{9}{4}\sin^2\theta \tag{6.2}$$

Using this equation as the exact solution, the root-mean-squared error in the converged solution on the surface of the sphere can be computed, the results of which are shown in figure 6.5a. The second-order solver (Strand) starts out strong at 1.5, but slows down to an order of 0.5. Flux correction (Strand FC) is slightly more accurate, and converges at a slightly better order of accuracy (1.8). The drag force is shown in figure 6.5b. Both the second-order and flux correction solvers demonstrate a clear third-order accurate convergence rate on the drag force, with flux correction being slightly more accurate.

The results from the overset meshes show that in this situation, the Cartesian grid has almost no effect on the solution at the surface of the sphere. This highlights that the overset approach, in which near-body and off-body regions are separated in differing meshes and solvers, is valid. This also highlights the importance of using high-order solvers in the

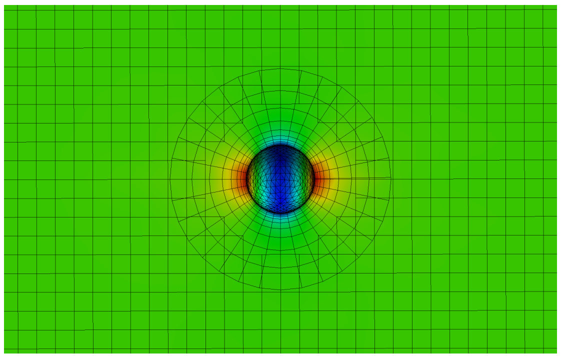
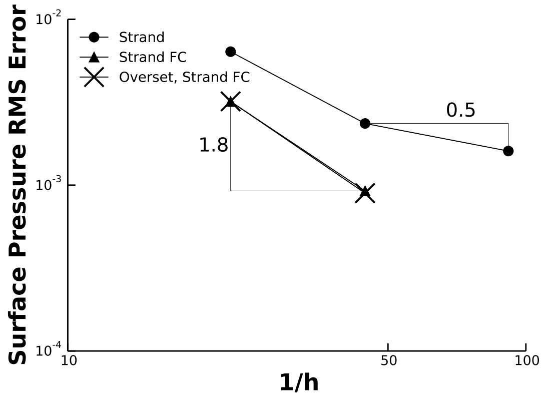
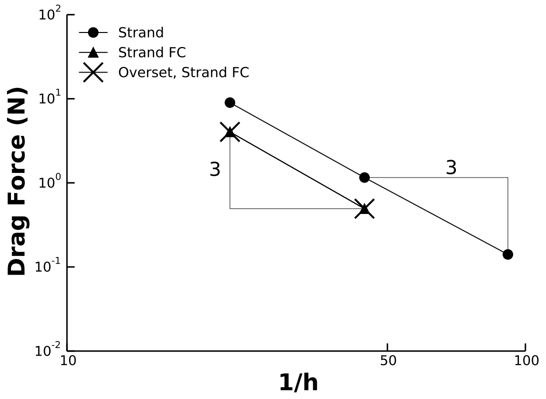


Fig. 6.4: Coarsest overset mesh with pressure contours of the inviscid flow solution



(a) Pressure RMS Error



(b) Drag Force Error

Fig. 6.5: Results from the grid refinement study using an incompressible, inviscid solution

near-body, as they will have the most influence on the accuracy on the surface, which is generally the region of interest in almost any flow simulation.

It was necessary to lower the CFL on both the strand and the Cartesian grid to achieve a stable solution. It was even necessary to lower the CFL when running flux correction on the 1024x128 stand-alone strand grid. Even when it was not necessary to lower the CFL, the flux correction solver took much longer to converge than the second-order solver. Some of this is attributed to the low Mach number, which causes a decoupling of the velocity and energy equations in the compressible Navier-Stokes equations. Preconditioning of the equations would speed convergence, but was not available in the Cartesian solver. As the solution converged, the drag coefficient converged in an oscillatory manner, with an exponential decay. The decay rate was much slower with flux correction, and might be due to the smaller numerical dissipation in the solution, which leads to less damping in the solution. Presumably, alternative solution techniques are available which could increase the convergence rate. The poor convergence of the finest flux correction case and the finest overset case prevented their inclusion in the results.

## 6.3 Steady Laminar Sphere

Next, we turn to validation cases using steady laminar flow over a sphere at Mach number of 0.2 over a range of Reynolds numbers. The surface mesh used by the strand grid consisted of 1024 fourth-order triangular elements. The strands were extended to a length of two diameters, with 32 nodes in the strand direction, the last two of which were clipped to interface with the Cartesian grid. The Cartesian grid covered the rest of the domain, which was defined as a rectangular region extending 8 diameters upstream, 14 diameters downstream, and 8 diameters on either side of the sphere. The grid was manually refined around the strand grid and in the wake region, using three levels of refinement to approximately match the strand spacing at the clipped index. The full mesh is depicted in figure 6.6, and consists of approximately 260,000 non-clipped strand nodes and 400,000 non-multigrid and non-hole Cartesian nodes, for a total of approximately 660,000 degrees of freedom. For each Reynolds number, the solution was run until all residuals fell below $10^{-6}$. The Cartesian grid was solved using third-order finite differences, while the strand solver was tested with the second-order solver (Strand) and the high-order flux correction solver (Strand FC).

This mesh was used to perform a low Reynolds number sweep at Re=40, 80, 120, 160, and 200. At each Reynolds number, the separation angle, recirculation length, and upper center coordinates of the standing ring vortex were compared against computational data from Magnaudet [82] and Tomboulides [83], and experimental data from Pruppacher [84] and Tenada [85]. This same experiment was run by Tong [19] using just the strand grid, which was extended to 20 diameters and used 128 nodes along the strands, which resulted

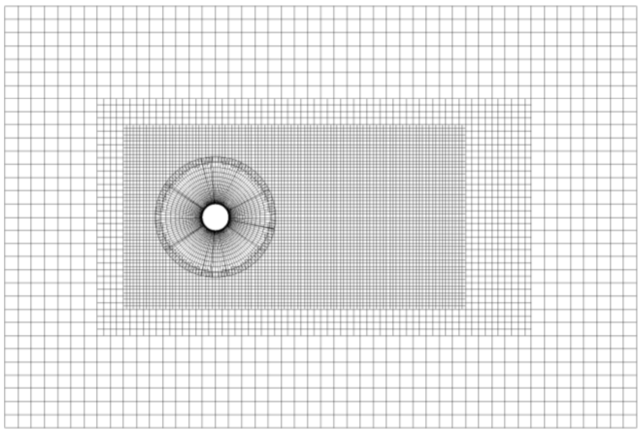


Fig. 6.6: Overset mesh used for laminar sphere test cases

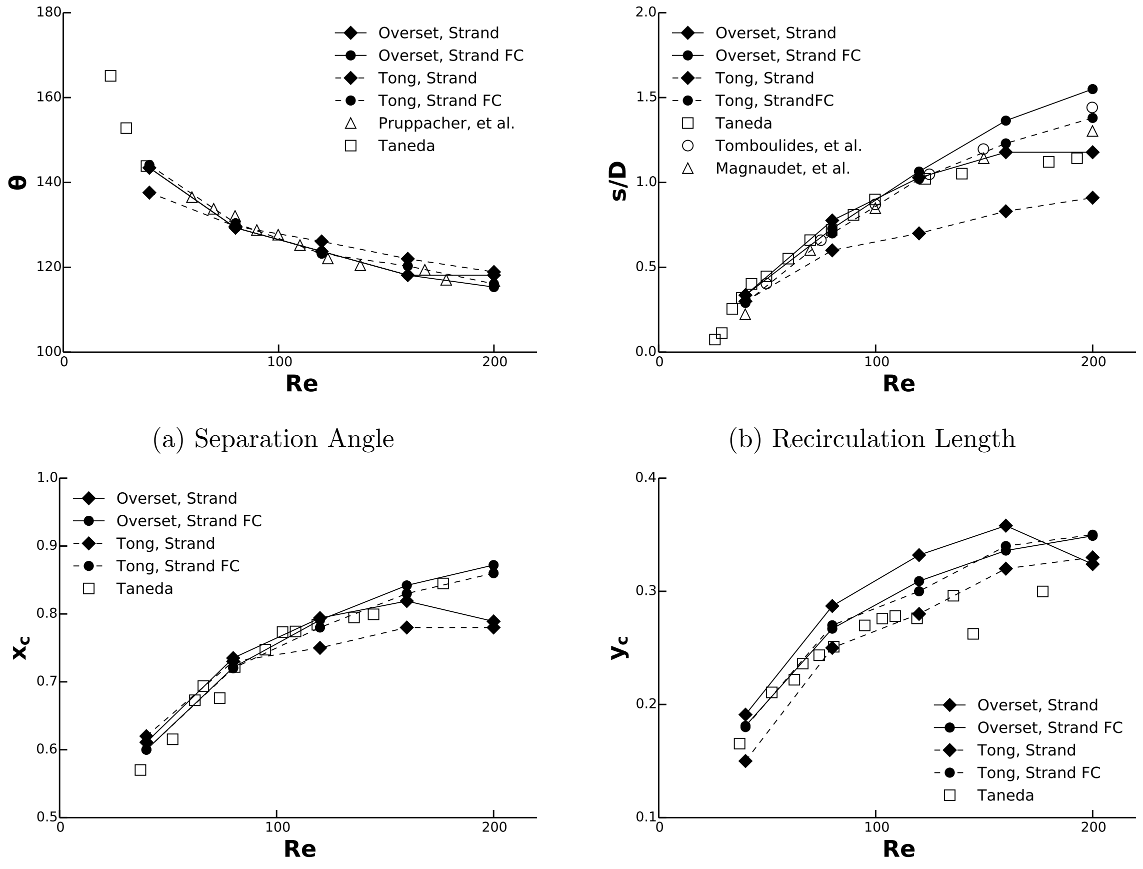
in roughly 1 million nodes total. It is interesting to note that using the overset method, the total number of nodes was reduced by 34%, without sacrificing accuracy.

The results of this experiment are shown in figure 6.7. Both the overset results from this work and the results of Tong [19] show good agreement with the computational and experimental results. Use of the overset Cartesian mesh improved estimates of the separation angle slightly, while the second-order estimates of recirculation length were greatly improved by the presence of the Cartesian mesh. The flux correction method appears to overpredict the recirculation length slightly at the higher Reynolds numbers. For the $x$-location of the vortex center, the second-order strand solver again shows an improvement in prediction. Flux correction has little change from the results of Tong [19]. The second-order method appears to significantly overpredict the $y$-locations, while flux correction shows little difference from using the strand solver by itself. It should be noted that the second-order solver predicted an unphysical asymmetric solution at Re=200, which accounts for the drastic change between Re=160 and Re=200 which is seen in figure 6.7d. As the standing vortex grew with the increased Reynolds number, it moved into the interpolation region between the two meshes. The interface between the strand and Cartesian grids most likely produced numerical errors which induced the asymmetric solution. The problem was not noted with the flux correction method.

The flux correction method for these cases took approximately 15% more iterations to converge, with only a 23% increase in computation time per iteration (solved on 60 processes). An iteration is defined as an iteration in pseudotime on the strand and Cartesian solvers, along with the necessary interpolations and transfers between the two grids.

## 6.4 Unsteady Laminar Flow over a Sphere

The next case evaluates the overset methodology for unsteady flow over a sphere at M=0.2, with a Reynolds number sweep from 400 to 2000. For each Reynolds number, four cases were run. The first case was run using the industry software Star-CCM+. Star-CMM+ is a well-established second-order cell-centered finite-volume software, and was available to use for a baseline comparison. The next two cases both used the strand-Cartesian method-



(a) Separation Angle

(b) Recirculation Length

(c) Vortex center $x$-coordinates

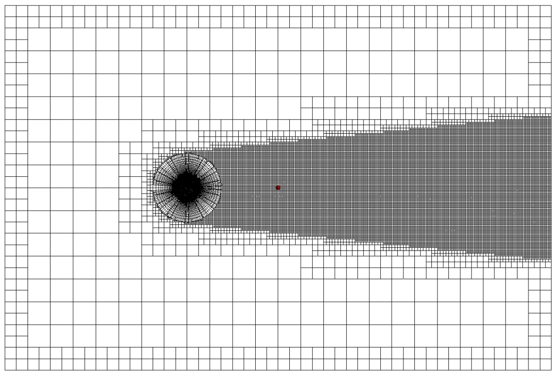
(d) Vortex center $y$-coordinates

Fig. 6.7: Low Reynolds number laminar sphere results from the overset mesh compared against results from Tong [19] using only a strand grid. The strand grid was tested using second-order FV (Strand) and high-order flux correction (Strand FC). Results are also compared against experimental and computational data

ology. In both cases, the Cartesian grid automatically refined to the strand grid, and automatically refined to flow features using the Q-criterion. The Cartesian solver was run third-order accurate for both cases. The only difference between the two cases was in the solver used on the strand grid, which were the second-order solver and the third-order flux correction solver. The fourth case uses just a strand grid with the flux correction solver, without the Cartesian overset grid. For all four cases, the solution was computed for a total physical time of 10 seconds, with a physical time step of 0.002 seconds, for a total of 5000 timesteps.

This particular case was used by Tong [19] to evaluate the flux correction method on strand grids. It is well known that when the flow over a sphere exceeds a Reynolds number of 480, an irregular mode is reached in which the shedding of hairpin vortical structures becomes chaotic. Tong [19] found that the second-order strand solver added enough numerical dissipation that the effective Reynolds number was lowered into a regular shedding mode, but the flux correction method was able to resolve the irregular shedding.

The Star-CMM+ and strand-Cartesian meshes were constructed to have a similar number of degrees of freedom. The near-body meshes had a radial distance of one diameter, with 48 prism layers in Star-CCM+ and 48 strand nodes on the strand grid. The Star-CCM+ cases did not use any automated mesh refinement, but the mesh was constructed with a "wake-refinement" grid, to the same $\Delta x$ spacing as the Cartesian grid. Images of the Star-


(a) Star-CCM+ mesh, with wake-refined grid

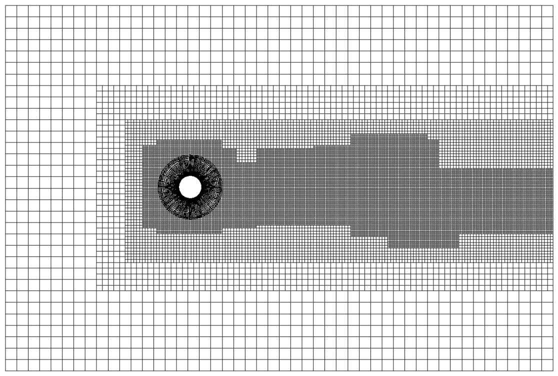

(b) Overset strand-Cartesian mesh, shown at $t = 10s$ with AMR

Fig. 6.8: Meshes used for the unsteady laminar sphere cases

CCM+ and the strand-Cartesian meshes are shown in figure 6.8. Each physical timestep was run with 100 pseudosteps or until the residuals fell below $10^{-6}$.

For all four cases, the Strouhal number was measured and compared to the extensive experimental study of Sakamoto and Haniu [86]. The Strouhal number is a nondimensional frequency, and is defined as

$$St = \frac{fL}{v_\infty} \tag{6.3}$$

where $f$ is the dominate frequency, $L$ the characteristic length (diameter in this setup), and $v_\infty$ is the freestream velocity. In their work, the Strouhal number was measured with a hot-wire probe placed at a distance of 3-4 diameters behind the sphere. Mimicking this setup, the velocity was recorded for all three cases at $\boldsymbol{x}_p = (4, 0, 0)$ in the wake of the sphere. This location falls in the Cartesian grid, and so provides a test as to how flux correction in the near-body affects the global flow in the off-body. A power spectrum from a Fourier transform of the velocity over time reveals the dominant shedding frequency. Occasionally, a high-pass filter is used to discard lower frequencies. The power spectrums can be found in appendix E, with the frequencies used for the computation of the Strouhal number shown with a circle. The first two seconds of data were not used in the computation of the Strouhal number in order to avoid pollution by the initial transients. The resulting Strouhal computations will be examined first, followed by a qualitative examination of the results.

The computed Strouhal numbers are shown in figure 6.9. The open circles indicate the experimental data of Sakamoto. It is obvious that none of the overset methods accurately predict the Strouhal number. The overset method with flux correction in the strand grid performed even worse at predicting the Strouhal number than the second-order methods. To provide some insight to this baffling result, the fourth case, using flux correction with just a strand grid and no overset Cartesian grid, was run. This results in much better accuracy than any of the overset methods. From this, it appears that the overset interface is affecting the temporal accuracy of the solution. It is possible that the overset cases were not converged enough, as the convergence of the overset cases was worse than the standalone strand grid case. The overset cases hit the 100 pseudostep limit, while the standalone strand grid cases would hit the residual convergence limit of $10^{-6}$. This was seen as minor at the time, as the overset residuals always ended at approximately $3 \times 10^{-6}$. A test case was run at Re=2000 which allowed for full convergence of the residuls below $10^{-6}$. This showed a slight improvement in the computed Strouhal number, but it was not considered significant enough to rerun all Reynolds numbers.

The poor showing from flux correction is unfortunate, but all is not lost. A closer look at the power spectrums at Re=400 and Re=600, shown in figure 6.10 and figure 6.11, reveals that flux correction is the only method that predicts the correct shedding mode. When Re=400, the second-order methods predict an unsteady shedding pattern, while flux correction obtains a clear periodic shedding frequency. When Re=600, the situation is reversed, with flux correction correctly predicting an aperiodic shedding, and the second-order methods predicting a periodic shedding. The correct prediction of the shedding modes is verified by consulting the work of Sakamoto [86].

Lastly, examination of the wake at Re = 600 for the overset cases and the standalone strand grid in figure 6.12 shows clearly resolved vortices (using the Q-Criterion at Q=20) in the Cartesian grid. This is a significant improvement over using the strand grid by itself.

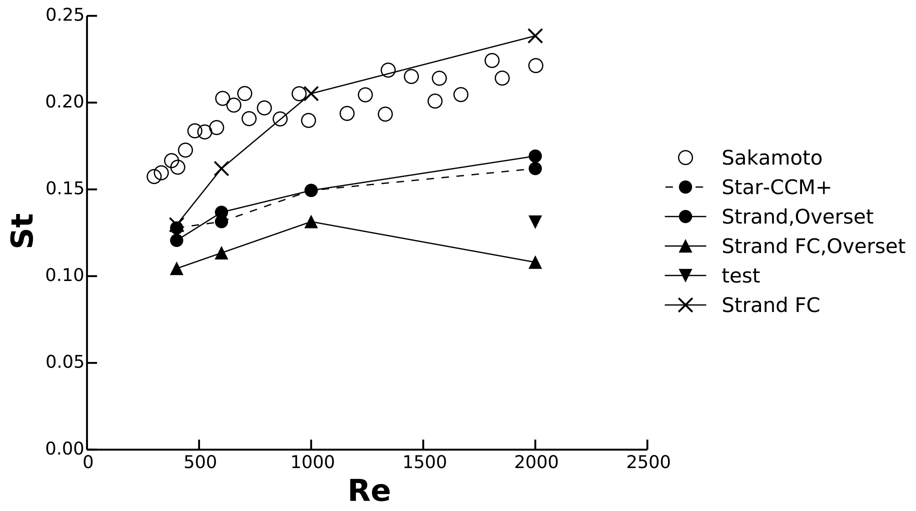
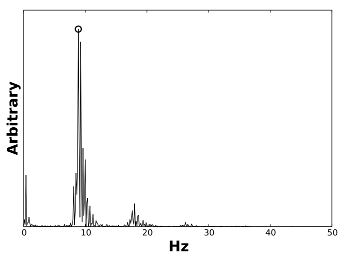
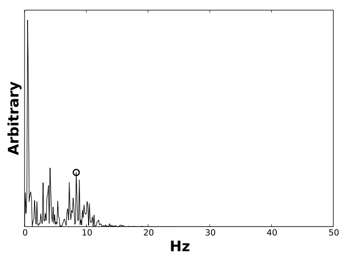


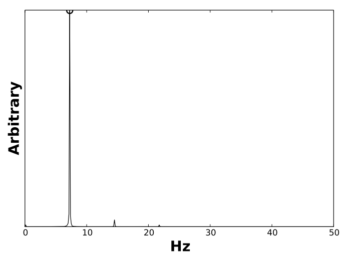
Fig. 6.9: Unsteady laminar sphere results from the overset mesh compared against the industry software Star-CCM+ and the experimental data of Sakamoto [86]. The strand grid was tested using a second-order solver (Strand) and flux correction (Strand FC)



(a) Star-CCM+

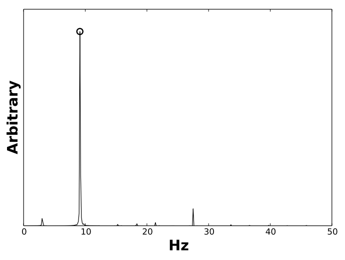


(b) Overset, second-order

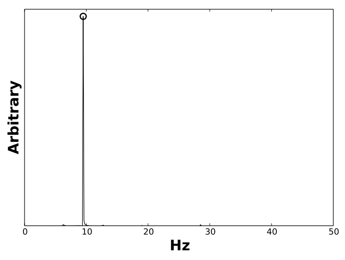


(c) Overset, flux correction

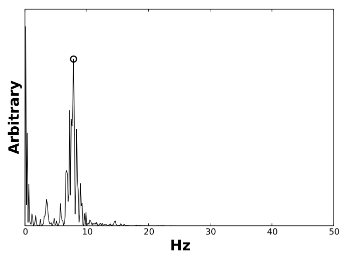
Fig. 6.10: Power spectrums of shedding sphere at Re = 400



(a) Star-CCM+



(b) Overset, second-order

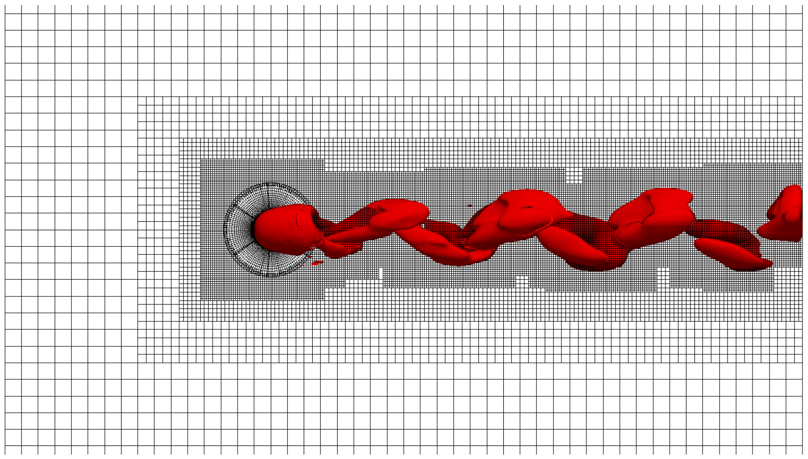


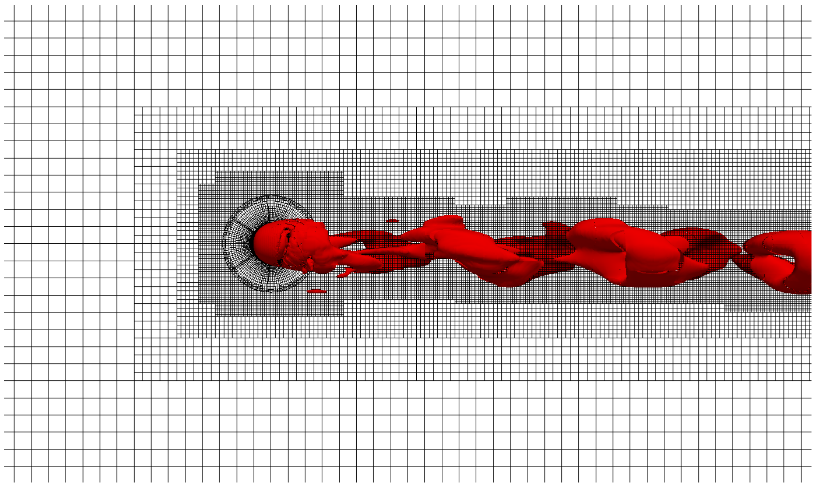
(c) Overset, flux correction

Fig. 6.11: Power spectrums of shedding sphere at Re = 600

## 6.5 Isentropic Vortex Propagation

The development of flux correction on strand grids was driven by the inability of a traditional second-order solver to retain the strength of a vortex, which was being generated by a rotor blade. (See figure 1.2.) In this test case, the ability of flux correction to propagate a vortex in an overset context is tested. The setup is demonstrated in figure 6.13a. To generate the strand mesh, a $2 \times 2$ square in the $x$-$y$ plane was meshed with right triangles. Each point was randomly perturbed in the $x$-$y$ plane to prevent any superconvergence of flux correction. The square was then rotated $45^\circ$ about the $z$-axis. A cube is formed by extending the strands to a length of 2. The plane was translated so that the center of the volume mesh is located at the origin. The plane was meshed with 15 triangles on a side, for a total of 50 linear triangles. After the fourth-order triangle subdivision procedure, this amounts to 60 triangles per side, with characteristic mesh length $h \approx 0.0\overline{33}$. Sixty strand nodes were placed along the strands for a strand spacing of $\Delta s = 0.0\overline{33}$. The Cartesian grid is defined from $(-10, -10, -4)$ to $(10, 10, 4)$, with a base level spacing of $\Delta x_0 = 0.2\overline{66}$.


(a) Overset, second-order


(b) Overset, third-order

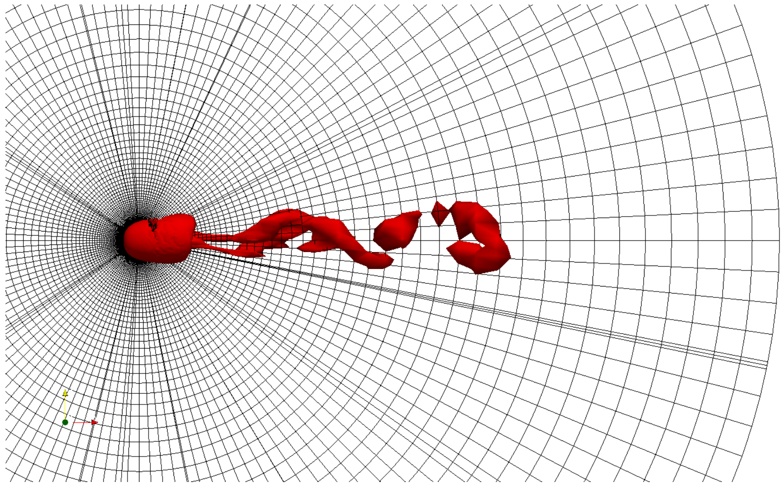

(c) Flux correction, strand grid

Fig. 6.12: Q=20 contour of wake behind sphere at Re=600

The automatic refinement was used to refine around the strand grid, creating three more refinement levels with a spacing of $\Delta x_3 = 0.0\overline{33}$, to match the strand grid spacing. Both the strand and Cartesian grids were initialized to the exact solution, and the Cartesian grid was automatically refined around the vortex. Note that a large buffer value was used in the refinement process to ensure the entire vortex was contained in the finest Cartesian level.

The exact solution is a two-dimensional isentropic vortex in the $x$-$y$ plane, the definition of which can be found in the work by Shu [87]. This is superimposed on a uniform flow in the $x$-direction. The equation is reproduced here.

$$\begin{gathered}
u = 1 + \Delta u, \quad v = 0 + \Delta v, \quad w = 0 \\
R_{gas} = P = 1, \quad T = 1 + \Delta T, \quad \gamma = 1.5 \\
a = 3, \quad \epsilon = 5, \quad r = \sqrt{x^2 + y^2} \\
\Delta u = -ay\frac{\epsilon}{2\pi}\exp\left(\frac{1}{2}\left(1 - a^2 r^2\right)\right) \\
\Delta v = ax\frac{\epsilon}{2\pi}\exp\left(\frac{1}{2}\left(1 - a^2 r^2\right)\right) \\
\Delta T = -\frac{(\gamma - 1)\,\epsilon^2}{8\gamma\pi^2}\exp\left(1 - a^2 r^2\right)
\end{gathered} \tag{6.4}$$

Here, $u$, $v$, and $w$, are the $x$, $y$, and $z$ velocities, respectively. The universal gas constant is denoted with $R_{gas}$, $\gamma$ is the ratio of specific heats, $P$ is the pressure, and $T$ is the temperature. The value $\frac{1}{a}$ defines the radius of maximum velocity in the vortex, and $\epsilon$ is the strength of the vortex. The Cartesian solver boundaries used farfield boundary conditions on the $x$ and $y$ boundaries, and periodic boundaries on the $z$ boundaries.

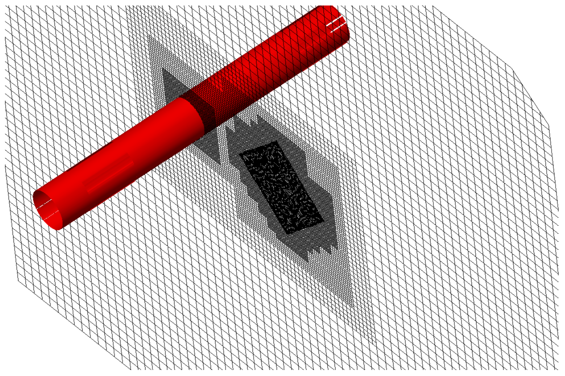
A few modifications to the strand solver and the domain connectivity were required to implement this test case. The clipping index was set to clip only the last node of each strand. An additional clipping array was added to the strand solver to clip the root nodes, and the two clipping indices were set to clip the entire strand for strands on the boundary of the plate. This essentially removed the plate from the computation of the strand grid.

The exact solution for the Euler equations is the propagation of the isentropic vortex in the $x$-direction. Two test cases were run with the previously described setup. The first used the second-order solver on the strand grid, while the second used the high-order flux correction. The Cartesian grid was solved third-order. Each case was initialized to the exact solution, and then solved in time. A baseline case using just the Cartesian solver alone was also run. The mesh for this case is shown in figure 6.14a. In order to mimic the strand setup, an additional level of Cartesian refinement was manually placed at the origin. This is to determine if refinement in the Cartesian mesh affects the vortex. Each timestep was solved in pseudotime until the residuals dropped below $10^{-6}$. The vortex starts at $x = -3$ and stepped until $t = 6$ with $\Delta t = 0.002$, for a total of 3000 timesteps.

The final vortex, after it has passed mostly through the strand grid (or the refined Cartesian region), is shown in figure 6.13b and figure 6.14b. It is clear that the vortex is not maintained in the interface region of the strand grid, while the Cartesian-only mesh easily maintains the structure of the vortex.

This leaves the question as to who is at fault: Flux correction, or the interface? To answer this question, the vorticity along the $x$-axis was examined at every timestep. A few timesteps are shown in figure 6.15, specifically, the initial solution, as the vortex passes through the front interface, with the vortex inside the strand grid, and as the vortex passes through the back interface. Also shown in figure 6.16 is the maximum vorticity along the $x$-axis, plotted as a function of time. The vertical dashed lines indicate when the center of the vortex should move from one grid to the other. For a yet unknown reason, the vortex travels slower than expected, and the vorticity drops significantly at first, only to eventually level off for all cases. The discrepancies only start to appear as the vortex passes through the interface.

Consider each figure in turn. As the vortex passes through the first interface, the vorticity drops, and the overall width of the vortex gets larger. The interaction with the interface also causes a few anomalies which grow as the vortex passes through the strand grid, causing the negative vorticity regions to grow. These anomalies are seen in both the second-order and flux correction solvers, so the growth of these are not due to flux correction. Then as the vortex passes through the back interface, the vorticity grows and the vortex is


(a) Initial condition and mesh setup.

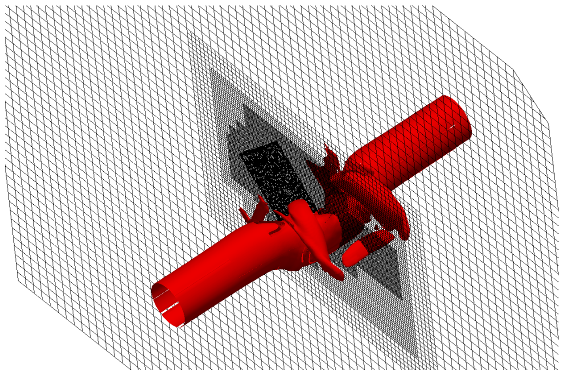
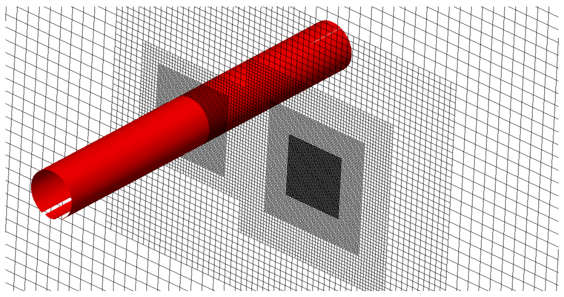

(b) Time $t = 6$ seconds using an overset mesh

Fig. 6.13: Isentropic vortex solution and mesh for the overset case. The Cartesian grid was solved third-order, and flux correction was used on the strand grid. The contour shown is $Q = 0.1$


(a) Initial condition and mesh setup

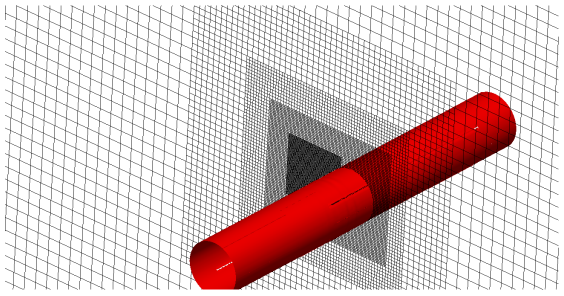
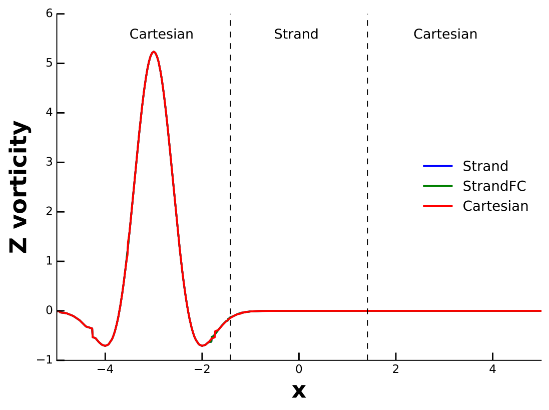

(b) Time $t = 6$ seconds using a refined Cartesian grid

Fig. 6.14: Isentropic vortex solution and mesh for the Cartesian-only case, which was solved using a third-order solver. The contour shown is $Q = 0.1$

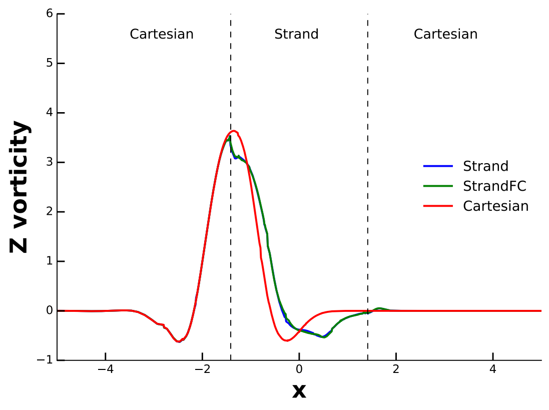
squeezed tighter. Looking at figure 6.16, it can be seen that after the vortex passes through the first interface, the maximum vorticity stays constant until the vortex passes through the back interface. This indicates that the numerical errors shown are issues with the overset interface, and not issues with flux correction itself. The results using just the Cartesian grid shows the vorticity remaining almost constant after the initial dropoff, showing that a refinement region with a proper interface does not affect the solution at all.

## 6.6 Summary of Computational Results

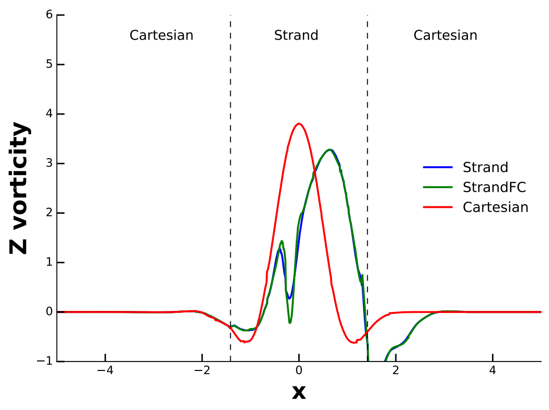
These computational results demonstrate a working strand-Cartesian solver, with the high-order flux correction solver employed on the strand grid. The MMS cases demonstrated a decreased global order of accuracy for both flux correction and the Cartesian solver, due to the interpolations used in the overset grid. The inviscid sphere showed that the accuracy of near-body steady-state quantities were not significantly affected by the overset paradigm, but that the convergence to steady-state was. Steady-state low Reynolds number viscous flow over a sphere shows improved accuracy using the overset mesh, while only using half the degrees of freedom, demonstrating the effectiveness of applying different grid types and solvers to different regions. However, when moving to unsteady shedding from a sphere, that accuracy was lost in the downstream regions of the flow. Further clarification is found in the advection of an isentropic vortex, which loses integrity upon contact with the overset interface. These last two cases demonstrate the need for future research on time-accurate interpolations on the overset interface.



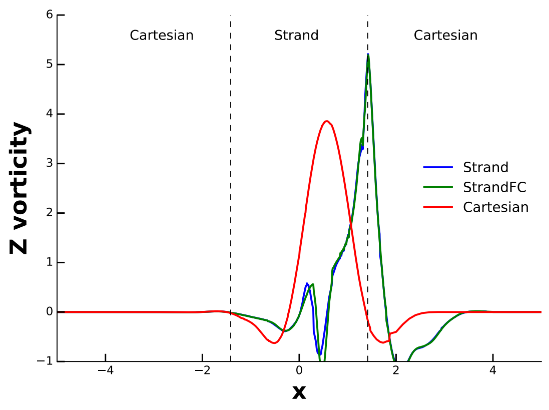
(a) 1. $t = 0$ seconds



(b) 2. $t = 2.43$ seconds



(c) 3. $t = 4.508$ seconds



(d) 4. $t = 5.378$ seconds

Fig. 6.15: Vorticity magnitude, plotted along the $x$-axis at various times

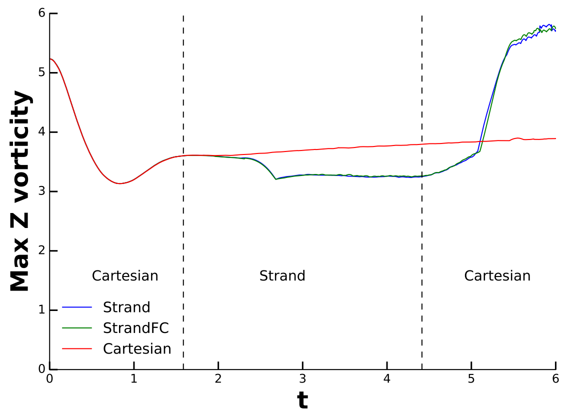


Fig. 6.16: Maximum vorticity on the $x$-axis, plotted as a function of time. The dashed lines indicate when the vortex should be entering/leaving the strand grid based on the $x$-velocity

# CHAPTER 7
# CONCLUSIONS AND FUTURE WORK

The development of strand grids is driven by the need to accurately and efficiently compute solutions to problems involving multiple moving bodies and problems involving highly vortical flows (such as those produced by rotorcraft). Strand grids are designed to capture the formation of boundary layers and vortices close to the surface of the bodies, while overset Cartesian grids then carry body-generated flow features into the rest of the domain. Previous work used second-order solvers on the strand grids, with high-order solvers in the Cartesian grids. The second-order solvers were unable to accurately resolve the strong wingtip vortices, leading to a global degradation of accuracy. This spurred the development of the high-order flux correction method, which achieves third-order accuracy for the inviscid terms, and fourth-order accuracy for the viscous terms on unstructured triangular grids. Through the use of high-order summation-by-parts operators and a semi-implicit method, the flux correction method retains accuracy on strand grids, while remaining stable and convergent, with minimal computational overhead. Previous work with flux correction used only stand alone strand grids, with no overset Cartesian grid.

This work demonstrates a coupled high-order flux correction strand-grid solver with a high-order Cartesian solver in an overset methodology. The inherent structure in the strand grid and the Cartesian grid allows the use of unique and simple methods for performing the necessary domain decomposition and interpolations. Verification of the Cartesian solver was performed using MMS, showing third-order accuracy for the inviscid terms of the Navier-Stokes equations. The viscous terms were treated using a second-order finite-volume scheme, which is also verified using MMS.

The flux correction solver was tested with the overset methodology using several test cases. The first was a grid refinement study using MMS, which showed a decrease in the order of accuracy of flux correction from $O(3)$ to $O(2.5)$. The order of accuracy was diminished by the second-order interpolations. The effect of more robust interpolation methods could be explored in later work.

The effects of the overset method on the stability and convergence of flux correction were clearly seen in the second test case, which solved low Mach number inviscid flow over a sphere. Using the overset method with flux correction adversely affected the convergence to steady-state, and required a lower CFL to remain stable. The accuracy of the drag coefficient on the sphere, was not affected by the overset mesh, however.

The next test case was low-Reynolds-number flow over a sphere. These tests showed that, in general, solutions on the strand grid benefit greatly from the use of the Cartesian grid in an overset manner. The high-order flux correction method more accurately predicts characteristics of the recirculation bubble behind the sphere, especially at higher Reynolds numbers. Unsteady flow over a sphere, over a range of Reynolds numbers, is also tested using second-order and high-order methods. Comparisons against experiment and the industry software Star-CCM+ were made. None of the overset methods correctly predicted the Strouhal number, but flux correction on a stand-alone strand grid performed much better. This indicates that the overset method can affect the accuracy of temporal statistics in the flow. Despite this, flux correction in an overset situation still predicted the correct shedding mode. The Cartesian mesh also allowed for the resolution of the wake, which has not been possible up to this point with strand grids alone.

Flux correction is more accurate locally in the strand grids than the traditional second-order methods on which it is based. This is shown in the MMS test cases, the inviscid sphere cases, and the low-Reynolds-number steady-sphere cases. However, that improvement in the order of accuracy is not maintained in the presence of an overset Cartesian grid. Also, as seen in the Strouhal number of the unsteady sphere cases, that improvement in accuracy does not propagate well into the rest of the domain. The convergence of flux correction was seen to be affected dramatically by the behavior of the solver to which is was coupled. Intuitively, this makes sense, but it does call into question the traditional overset mindset. This mindset is that different grid types and solvers can be used in conjunction, without regard for the other solvers being used. Additionally, the stability of the flux correction method was occasionally adversely affected. The CFL had to be lowered in order to force a stable computation.

The two previous points indicate that in order to assure a stable, globally high-order accurate, and convergent solution, a rigorous framework is needed to develop high-order overset methods. High-order methods in general have been the subject of much research for the past decade, and have made great strides in becoming industry-ready. However, the preceding results indicate that there is still work to be done before high-order methods can be used in a traditional overset simulation. A good starting point could be the Simultaneous Approximation Terms used in a multiblock mesh for SBP operators [88]. These allow the *joining* of two different meshes at an interface, and are provably stable and globally accurate. Another important aspect to consider is correctly designing high-order methods for use on moving meshes, which if done incorrectly, can degrade the method to a low-order method [72].

Computationally, flux correction still provides high-order accuracy with minimal computational effort. Currently, the flux correction implementation only increases the walltime per pseudostep by about 25%. It is believed that reworking the solution algorithm would allow for an even smaller time increase. Using the overset method also allowed for a significant decrease in the number of nodes, without sacrificing accuracy. This is because the most complex portion of the flow (the flow near the body) is accurately resolved with flux correction, and the rest is efficiently resolved with the Cartesian grid.

Flux correction on strand grids has now reached full maturity with its integration into a complete overset strand-Cartesian framework. The flux correction strand solver is now ready for integration into a production-level code, such as that fielded by the CREATE program. To make the combined solver fully high-order, research into conservative and high-order interpolation schemes must be performed. Modification of the flux correction method to allow for moving grids would allow the method to finally realize the goal of quickly and accurately predicting flow solutions over full ship-aircraft and aircraft-aircraft interactions.

## REFERENCES

[1] Wang, Z. J., “High-order Methods for the Euler and Navier–Stokes Equations on Unstructured Grids,” *Progress in Aerospace Sciences*, Vol. 43, 2007, pp. 1–41.

[2] Katz, A., Wissink, A., Sankaran, V., Meakin, R., and Sitaraman, J., “Application of Strand Meshes to Complex Aerodynamic Flow Fields,” *Journal of Computational Physics*, Vol. 230, 2011, pp. 6512–6530.

[3] Meakin, R., Wissink, A., Chan, W., Pandya, S., and Sitaraman, J., “On Strand Grids for Complex Flows,” *AIAA 18th Computational Fluid Dynamics Conference*, No. AIAA 2007-3834, 2007.

[4] Wissink, A., Katz, A., Chan, W., and Meakin, R., “Validation of the Strand Grid Approach,” *AIAA 19th Computational Fluid Dynamics Conference*, No. AIAA 2009-3792, 2009.

[5] Wissink, A., Potsdam, M., Sankaran, V., Sitaraman, J., Yang, Z., and Mavriplis, D., “A Coupled Unstructured Adaptive Cartesian CFD approach for Hover Prediction,” Tech. rep., American Helicopter Society 66th Annual Forum, Phoenix, AZ, 2010.

[6] Steger, J., Dougherty, F., and Benek, J., “A Chimera Grid Scheme,” *Advances in Grid Generation*, Houston, TX, June 20-22 1983.

[7] Benek, J. A., Steger, J. L., and Dougherty, F. C., “A Flexible Grid Embedding Technique with Application to the Euler Equations,” *AIAA 6th Computational Fluid Dynamics Conference*, No. AIAA Paper 1983-1944, Danvers, MA, 1983.

[8] Lee, Y. L. and Baeder, J., “Implicit Hole Cutting: A New Approach to Overset Grid Connectivity,” *AIAA 16th Computational Fluid Dynamics Conference*, No. AIAA Paper 2003–4128, Orlando, FL, 2003.

[9] Sitaraman, J., Floros, M., Wissink, A., and Potsdam, M., “Parallel Domain Connectivity Algorithm for Unsteady Flow Computational using Overlapping and Adaptive Grids,” *Journal of Computational Physics*, Vol. 229, 2008, pp. 4703–4723.

[10] Wissink, A., Katz, A., and Sitaraman, J., “PICASSO: A Meshing Infrastructure for Strand-Cartesian CFD Solvers,” *30th AIAA Applied Aerodynamics Conference*, No. 2012-2916, 2012.

[11] Katz, A. and Sankaran, V., “An Efficient Correction Method to Obtain a Formally Third-order Accurate Flow Solver for Node-Centered Unstructured Grids,” *Journal of Scientific Computing*, Vol. 51, No. 2, 2012, pp. 375–393.

[12] Wissink, A., Sitaraman, J., and Katz, A., “Development of a High-Order Strand Solver for Helios,” *12th Symposium on Overset Grids and Solution Technology*, Georgia Institute of Technology, Oct 2014.

[13] Katz, A. and Wissink, A., "Efficient Solution Methods for Strand Grid Applications," *30th AIAA Applied Aerodynamics Conference*, No. AIAA 2012-2779, 2012.

[14] Pincock, B. and Katz, A., "High-Order Flux Correction for Viscous Flows on Arbitrary Unstructured Grids," *Journal of Scientific Computing*, Vol. 61, No. 2, 2014, pp. 454–476.

[15] Katz, A. and Work, D., "High-order flux correction/finite difference schemes for strand grids," *Journal of Computational Physics*, Vol. 282, 2015, pp. 360 – 380.

[16] Tong, O., Katz, A., and Yanagita, Y., "Verification and Validation of a High-Order Strand Grid Method for Two-Dimensional Turbulent Flows," *Computers & Fluids*, 2017.

[17] Tong, O., Blakely, C., Schaap, R., and Katz, A., "Assessment of a Two-Equation Turbulence Model in the High-Order Flux Correction Scheme," *54th AIAA Aerospace Sciences Meeting*, 2016.

[18] Tong, O. and Katz, A., "An Assessment of Various Turbulence Models for Preconditioned High-Order Solutions of Flow Around Submersibles," *46th AIAA Fluid Dynamics Conference*, 2016.

[19] Tong, O., Katz, A., Yanagita, Y., Casey, A., and Schaap, R., "High-Order Methods for Turbulent Flows on Three-Dimensional Strand Grids," *Journal of Scientific Computing*, Vol. 67, No. 1, April 2016, pp. 84–102.

[20] Thorne, J., Katz, A., Tong, O., Yanagita, Y., Tamaki, Y., and Delaney, K., "High-order Strand Grid Methods for Low-speed and Incompressible Flows," *International Journal for Numerical Methods in Fluids*, 2016.

[21] Yanagita, Y., Tong, O., and Katz, A., "Critical Evaluation of Turbulence Model with the Flux Correction Method on Strand Grids," *54th AIAA Aerospace Sciences Meeting*, No. AIAA 2016-1360, 2016.

[22] Work, D., Tong, O., Workman, R., and Katz, A., "Strand Grid Solution Procedures for Sharp Corners," *51st AIAA Aerospace Sciences Meeting*, No. AIAA 2013-800, 2013.

[23] Mavriplis, D. and Venkatakrishnan, V., "A Unified Multigrid Solver for the Navier-Stokes Equations on Mixed Element Meshes," *International Journal of Computational Fluid Dynamics*, Vol. 8, No. 4, 1997, pp. 247–263.

[24] Mavriplis, D. J., Aftosmis, M. J., and Berger, M., "High Resolution Aerospace Applications Using the NASA Columbia Supercomputer," *Proceedings of the 2005 ACM/IEEE Conference on Supercomputing*, SC '05, IEEE Computer Society, Washington, DC, USA, 2005.

[25] Wissink, A., Sitaraman, J., Sankaran, V., Mavriplis, D., and Pulliam, T., "A Multi-Code Python-Based Infrastructure for Overset CFD with Adaptive Cartesian Grids," *Aerospace Sciences Meetings*, American Institute of Aeronautics and Astronautics, Jan. 2008.

[26] Sitaraman, J., Katz, A., Jayaraman, B., Wissink, A., and Sankaran, V., "Evaluation of a Multi-Solver Paradigm for CFD Using Unstructured and Structured Adaptive Cartesian Grids," *Aerospace Sciences Meetings*, American Institute of Aeronautics and Astronautics, Jan. 2008, pp. –.

[27] Work, D., Tong, O., Workman, R., Katz, A., and Wissink, A., "Strand-Grid-Solution Procedures for Sharp Corners," *AIAA Journal*, Vol. 52, No. 7, 2014, pp. 1528–1541.

[28] Tong, O., Yanagita, Y., and Katz, A. J., "Asymptotic Geometry Representation for Complex Configurations," *54th AIAA Aerospace Sciences Meeting*, American Institute of Aeronautics and Astronautics, Jan. 2016.

[29] Katz, A. and Sankaran, V., "Mesh Quality Effects on the Accuracy of CFD Solutions on Unstructured Meshes," *Journal of Computational Physics*, Vol. 230, No. 20, 2011, pp. 7670 – 7686.

[30] Pincock, B. and Katz, A., "High-order Flux Correction for Viscous Flows on Arbitrary Unstructured Grids," *21st AIAA Computational Fluid Dynamics Conference*, The American Institute of Aeronautics and Astronautics, June 2013.

[31] Katz, A. and Work, D., "High-order Flux Correction/Finite Difference Schemes for Strand Grids," *AIAA 52$^{nd}$ Aerospace Sciences Meeting*, No. AIAA 2014-0937, AIAA, Jan. 2014.

[32] Spalart, P. and Allmaras, S., "A One-Equation Turbulence Model for Aerodynamic Flows," *Recherche Aerospatiale*, Vol. 1, 1994, pp. 5–21.

[33] Allmaras, S., Johnson, F., and Spalart, P., "Modifications and Clarifications for the Implementation of the Spalart–Allmaras Turbulence Model," *7th International COnference on Computational Fluid Dynamics*, 2012.

[34] Tong, O., Yanagita, Y., Schaap, R., Harris, S., and Katz, A., "High-Order Strand Grid Methods for Shock Turbulence Interaction," *22nd AIAA Computational Fluid Dynamics Conference*, 2015.

[35] Buning, P. G., Jespersen, D., Pulliam, T. H., Chan, W., Slotnick, J. P., Krist, S., and Renze, K. J., "Overflow user's manual," *NASA Langley Research Center*, Vol. 1, 1998.

[36] Rogers, S. E., Suhs, N. E., and Dietz, W. E., "PEGASUS 5: An Automated Preprocessor for Overset-Grid Computational Fluid Dynamics," *AIAA Journal*, Vol. 41, No. 6, June 2003, pp. 1037–1045.

[37] Noack, R., Boger, D., Kunz, R. e., and Carrica, P., "Suggar++: An Improved General Overset Grid Assembly Capability," *19th AIAA Computational Fluid Dynamics*, AIAA, June 2009.

[38] Noack, R., "DiRTlib: A Library to Add an Overset Capability to Your Flow Solver," *Fluid Dynamics and Co-located Conferences*, American Institute of Aeronautics and Astronautics, June 2005.

[39] Noack, R., "SUGGAR: A General Capability for Moving Body Overset Grid Assembly," *Fluid Dynamics and Co-located Conferences*, American Institute of Aeronautics and Astronautics, June 2005.

[40] Chan, W., "Developments in Strategies and Software Tools for Overset Structured Grid Generation and Connectivity," *Fluid Dynamics and Co-located Conferences*, American Institute of Aeronautics and Astronautics, June 2011.

[41] Meakin, R., "A New Method for Establishing Intergrid Communication Among Systems of Overset Grids," *10th Computational Fluid Dynamics Conference*, 1991.

[42] Meakin, R., "Object X-Rays for Cutting Holes in Composite Overset Structured Grids," *15th AIAA Computational Fluid Dynamics Conference*, 2001.

[43] Kim, N. and Chan, W., "Automation of Hole-Cutting for Overset Grids Using the X-rays Approach," *20th AIAA Computational Fluid Dynamics Conference*, 2011.

[44] Noack, R. W., "A Direct Cut Approach for Overset Hole Cutting," *18th AIAA Computational Fluid Dynamics Conference*, 2007.

[45] Meakin, R. L., *Handbook of Grid Generation*, chap. 11, Composite Overset Structured Grids, CRC Press, 1999.

[46] Tam, C. and Hu, F., "An Optimized Multi-Dimensional Interpolation Scheme for Computational Aeroacoustics Applications Using Overset Grid," *Aeroacoustics Conferences*, American Institute of Aeronautics and Astronautics, May 2004, pp. –.

[47] Sherer, S. E. and Scott, J. N., "High-Order Compact Finite-Difference Methods on General Overset Grids," *Journal of Computational Physics*, Vol. 210, No. 2, 2005, pp. 459 – 496.

[48] Lee, K. R., Park, J. H., and Kim, K. H., "High-Order Interpolation Method for Overset Grid Based on Finite Volume Method," *AIAA Journal*, Vol. 49, No. 7, July 2011, pp. 1387–1398.

[49] Chesshire, G. and Henshaw, W. D., "A Scheme for Conservative Interpolation on Overlapping Grids," *SIAM Journal on Scientific Computing*, Vol. 15, No. 4, July 1994, pp. 819–845.

[50] Meakin, R., "On the Spatial and Temporal Accuracy of Overset Grid Methods for Moving Body Problems," *12th Applied Aerodynamics Conference*, 1994.

[51] Lee, Y. and Baeder, J. D., "High-Order Overset Method for Blade Vortex Interaction," *40th AIAA Aerospace Sciences Meeting & Exhibit*, 2002.

[52] Landmann, B. and Montagnac, M., "A Highly Automated Parallel Chimera Method for Overset Grids Based on the Implicit Hole Cutting Technique," *International Journal for Numerical Methods in Fluids*, Vol. 66, No. 6, 6 2011, pp. 778–804.

[53] Barth, T., "Numerical Aspects of Computing High Reynolds Number Flows on Unstructured Meshes," *29th Aerospace Sciences Meeting*, AIAA, 1991.

[54] Haselbacher, A., *Grid-transparent Numerical Method for Compressible Viscous Flows on Mixed Unstructured Grids*, Ph.D. thesis, Loughborough University, 1999.

[55] Thorne, J. and Katz, A., "Source Term Discretization Effects on the Accuracy of Finite Volume Schemes," *53rd AIAA Aerospace Sciences Meeting*, 2015.

[56] Carpenter, M., Gottlieb, D., and Abarbenel, S., "The Stability of Numerical Boundary Treatments for Compact High-Order Finite-Difference Schemes," *Journal of Computational Physics*, Vol. 108, No. 2, 1993, pp. 272–295.

[57] Fernández, D. C. D. R. and Zingg, D., "High-Order Compact-Stencil Summation-by-Parts Operators for the Second Derivative with Variable Coefficients," *7th International Conference on Computational Fluid Dynamics (ICCFD7)*, No. Technical Report ICCFD7-2803, Big Island, HI, 2012.

[58] Kreiss, H. and Scherer, G., "Finite Element and Finite Difference Methods for Hyperbolic Partial Differential Equations," *Mathematical Aspects of Finite Elements in Partial Differential Equations*, edited by C. D. Boor, Academic Press, MA, 1974.

[59] Mattsson, K., "Summation by Parts Operators for Finite Difference Approximations of Second-Derivatives with Variable Coefficients," *Journal of Scientific Computing*, Vol. 51, 2012, pp. 650–682.

[60] Strand, B., "Summation by Parts for Finite Difference Approximation for d/dx," *Journal of Computational Physics*, Vol. 110, 1994, pp. 47–67.

[61] Diener, P., Dorband, E., Schnetter, E., and Tiglio, M., "Optimized High-Order Derivative and Dissipation Operators Satisfying Summation by Parts, and Applications in Three-Dimensional Multi-Block Evolutions," *Journal of Scientific Computing*, Vol. 32, 2007, pp. 109–145.

[62] Yoon, S. and Jameson, A., "Lower-Upper Symmetric-Gauss-Seidel Method for the Euler and Navier-Stokes Equations," *AIAA Journal*, Vol. 26, No. 9, September 1988, pp. 1025–1026.

[63] Rieger, H. and Jameson, A., "Solution of Steady Three-Dimensional Compressible Euler and Navier-Stokes Equations by An Implicit LU Scheme," *AIAA 26th Aerospace Sciences Meeting*, 1988.

[64] Chen, R. F. and Wang, Z. J., "Fast, Block Lower-Upper Symmetric Gauss-Seidel Scheme for Arbitrary Grids," *AIAA Journal*, Vol. 38, No. 12, December 2000, pp. 2238–2245.

[65] Jameson, A. and Caughey, D. A., "How Many Steps are Required to Solve the Euler Equations of Steady, Compressible Flow: In Search of a Fast Solution Algorithm," *AIAA 15th Computational Fluid Dynamics Conference*, 2001.

[66] Caughey, D. A. and Jameson, A., "Fast Preconditioned Multigrid Solution of the Euler and Navier-Stokes Equations for Steady, Compressible Flows," *International Journal for Numerical Methods in Fluids*, 2003.

[67] Pletcher, R. H., Tannehill, J. C., and Anderson, D. A., *Computational Fluid Mechanics and Heat Transfer*, chap. 9, CRC Press, 3rd ed., 2013, pp. 610–613.

[68] Mavriplis, D. J. and Jameson, A., "Multigrid Solution of the Two-Dimensional Euler Equations on Unstructured Triangular Meshes," *AIAA 25th Aerospace Sciences Meeting*, 1987.

[69] Jameson, A. and Mavriplis, D., "Finite Volume Solution of the Two-Dimensional Euler Equations on a Regular Triangular Mesh," *AIAA Journal*, Vol. 24, No. 4, April 1986, pp. 611–618.

[70] Brandt, A., "Multi-Level Adaptive Solutions to Boundary-Value Problems," *Mathematics of Computation*, Vol. 31, No. 138, 1977, pp. 333–390.

[71] Pletcher, R. H., Tannehill, J. C., and Anderson, D. A., *Computational Fluid Mechanics and Heat Transfer*, chap. 4, CRC Press, 3rd ed., 2013, pp. 173–174.

[72] Visbal, M. R. and Gaitonde, D. V., "On the Use of Higher-Order Finite-Difference Schemes on Curvilinear and Deforming Meshes," *Journal of Computational Physics*, Vol. 181, No. 1, 2002, pp. 155–185.

[73] Mavriplis, D., "Revisiting the Least-Squares Procedure for Gradient Reconstruction on Unstructured Meshes," *16th AIAA Computational Fluid Dynamics Conference*, 2003.

[74] Hornung, R. D. and Kohn, S. R., "Managing Complex Data and Geometry in Parallel Structured AMR Applications," *Engineering with Computers*, Vol. 22, No. 3, 2006, pp. 181–195.

[75] Hornung, R. D. and Kohn, S. R., "Managing Application Complexity in the SAMRAI Object-Oriented Framework," *Concurrency and Computation: Practice and Experience*, Vol. 14, 2002, pp. 347–368.

[76] Wissink, A. M., Hornung, R. D., Kohn, S. R., Smith, S., and Elliott, N., "Large Scale Parallel Structured AME Calculations Using the SAMRAI Framework," *Proceedings of SC01*, November 2001.

[77] Berger, M. J. and Colella, P., "Local Adaptive Mesh Refinement for Shock Hydrodynamics," *Journal of Computational Physics*, Vol. 82, 1989.

[78] Nadarajah, S. K., *The Discrete Adjoint Approach to Aerodynamic Shape Optimization*, Ph.D. thesis, Stanford University, 2003.

[79] Kamkar, S. J., Jameson, A., and Wissink, A. M., "Automated Grid Refinement Using Feature Detection," *47th AIAA Aerospace Sciences Meeting*, 2009.

[80] Kamkar, S. J. and Wissink, A. M., "An Automated Adaptive Mesh Refinement Scheme for Unsteady Aerodynamic Wakes," *50th AIAA Aerospace Sciences Meeting*, 2012.

[81] Roache, P. J., "Code Verification by the Method of Manufactured Solutions," *Journal of Fluids Engineering*, Vol. 124, 2001, pp. 4–10.

[82] Magnaudet, J., Riviero, M., and Fabre, J., "Accelerated Flows Past a Sphere Or Spherical Bubble, Part 1: Steady Streaming Flow," *Journal of Fluid Mechanics*, Vol. 284, 1995, pp. 97–135.

[83] Tomboulides, A., Orszag, S., and Karniadakis, G., "Direct and Large Eddy Simulations of Axisymmetric Wakes," *31st Aerospace Sciences Meeting*, No. AIAA Paper 93-0546, Reno, NV, 1993.

[84] Pruppacher, H., Clair, B. L., and Hamielec, A., "Some Relations Between Drag and Flow Pattern of Viscous Flow Past a Sphere and Cylinder at Low and Intermediate Reynolds Numbers," *Journal of Fluid Mechanics*, Vol. 44, 1970, pp. 781–791.

[85] Tenada, S., "Experimental Investigation of Wake Behind a Sphere at Low Reynolds Numbers," *Journal of the Physical Society of Japan*, Vol. 11, 1956, pp. 1104–1108.

[86] Sakamoto, H. and Haniu, H., "A Study on Vortex Shedding From Spheres in a Unifrom Flow," *Journal of Fluids Engineering*, Vol. 112, No. 4, 1990, pp. 386–392.

[87] Shu, C.-W., "Essentially Non-Oscillatory and Weighted Essentially Non-Oscillatory Schemes for hyperbolic Conservation Laws," NASA ICASE Report NASA/CR-97-206253, Institute for Computer Applications in Science and Engineering, November 1997.

[88] Hicken, J. E. and Zingg, D. W., "Parallel Newton-Krylov Solver for the Euler Equations Discretized Using Simultaneous-Approximation Terms," *AIAA Journal*, Vol. 46, No. 11, November 2008.

[89] Carpenter, M. H., Gottlieb, D., and Abarbanel, S., "Time-Stable Boundary Conditions for Finite-Difference Schemes Solving Hyperbolic Systems: Methodology and Application to High-Order Compact Schemes," *Journal of Computational Physics*, Vol. 111, No. 2, 1994, pp. 220 – 236.

[90] Fernández, D. C. D. R., Hicken, J. E., and Zingg, D. W., "Review of Summation-by-Parts Operators with Simultaneous Approximation Terms for the Numerical Solution of Partial Differential Equations," *Computers & Fluids*, Vol. 95, May 2014, pp. 171–196.

[91] Nordström, J. and Svärd, M., "Well-Posed Boundary Conditions for the Navier–Stokes Equations," *SIAM Journal on Numerical Analysis*, Vol. 43, No. 3, 2005, pp. 1231–1255.

[92] Svärd, M., "On Coordinate Transformations for Summation-by-Parts Operators," *Journal of Scientific Computing*, Vol. 20, No. 1, 2004, pp. 29–42.

[93] Svärd, M., Carpenter, M. H., and Nordström, J., "A Stable High-Order Finite Difference Scheme for the Compressible Navier–Stokes Equations, Far-Field Boundary Conditions," *Journal of Computational Physics*, Vol. 225, No. 1, 2007, pp. 1020–1038.

[94] Svärd, M. and Nordström, J., "A Stable High-Order Finite Difference Scheme for the Compressible Navier–Stokes Equations: No-Slip Wall Boundary Conditions," *Journal of Computational Physics*, Vol. 227, No. 10, 2008, pp. 4805–4824.

[95] Guennebaud, G., Jacob, B., et al., "Eigen v3," http://eigen.tuxfamily.org, 2010.

[96] WhiteTimberwolf, "Wikimedia Image," http://commons.wikimedia.org/wiki/File:Octree2.svg, 2010.

[97] Hjaltason, G. R. and Samet, H., "Ranking in Spatial Databases," *Proceedings of the 4th Symposium on Spatial Databases*, 1995.

# APPENDICES

# APPENDIX A

## Summation-by-Parts Operators

Summation-by-Parts operators are specialized finite difference stencils, which have been derived to ensure conservation and stability while maintaining high-order accuracy. They also provide a framework for the implementation of penalty boundary conditions, through the use of Simultaneous Approximation Terms [89]. Together, the full method is termed SBP:SAT.

In this brief discussion, the notation employed by Fernandez [90] will be followed. Vectors are denoted with lowercase bold fonts, $\mathbf{x}$, and matrices use uppercase sans-serif fonts, $\mathsf{M}$. Uppercase script letters $\mathcal{U}$ denote continuous functions on a specified domain $\mathcal{X} \in [\alpha, \beta]$. Lowercase bold fonts $\mathbf{u}$ denote the restriction of those functions onto a grid. For example, the restriction of $\mathcal{U}$ onto the grid $\mathbf{x}$ is given by:

$$\mathbf{u} = [\mathcal{U}(\mathrm{x}_0), \ldots, \mathcal{U}(\mathrm{x}_N)]^T \tag{A.1}$$

In general, the operators used have three different order of accuracy associated with them: the accuracy on the interior of the domain $(a)$, the accuracy on the boundary of the domain $(b)$, and the global accuracy $(c)$. A given discrete operator is then denoted by $\mathsf{D}_i^{(a,b,c)}$, where $i$ denotes that this is an operator for the $i$-th derivative. For the second derivative with variable coefficients, the dependence of the operator on those coefficients is made explicit with $\mathsf{D}_2^{(a,b,c)}(\mathsf{B})$, where $\mathsf{B}$ is a diagonal matrix with the variable coefficients along its diagonal. For a one-dimensional grid with $N$ nodes, $\mathsf{D}$ is a coefficient matrix of size $N \times N$.

SBP operators are constructed to satisfy a discrete version of the integration-by-parts equation,

$$\int_\alpha^\beta \mathcal{V} \frac{d\mathcal{U}}{d\mathcal{X}} d\mathcal{X} = \mathcal{U}\mathcal{V}|_\alpha^\beta - \int_\alpha^\beta \mathcal{U} \frac{d\mathcal{V}}{d\mathcal{X}} d\mathcal{X} \tag{A.2}$$

Well-posed boundary conditions can be formulated for the solution of PDEs through the use of this equation. SBP operators are constructed on a grid of $N$ nodes. The summation-by-parts equation is written as:

$$\mathbf{v}^T \mathsf{H} \mathsf{D}_1 \mathbf{u} = \mathbf{v}^T \mathsf{E} \mathbf{u} - \mathbf{u}^T \mathsf{H} \mathsf{D}_1 \mathbf{v} \tag{A.3}$$

where $\mathsf{E} = \text{diag}\,[-1, 0, \cdots, 0, 1]$, and $\mathsf{H}$ is a diagonal or block diagonal norm. The derivation of a given SBP operator starts by specifying an interior stencil (generally a high-order centered-difference stencil), and a norm, and then solving a system of equations to determine appropriate boundary stencils, such that conservation, accuracy, and stability properties are satisfied.

In this work, only SBP operators derived from a diagonal norm are considered because they are provably stable in a multidimensional context [91, 92]. Katz and Work [15] tested multiple operators, and showed that using $\mathsf{D}_1^{(6,3,4)}$ and $\mathsf{D}_2^{(6,3,5)}$ operators gave the best results for flux correction on strand grids. A second-order method can be recovered through the use of $\mathsf{D}_1^{2,1,2}$ and $\mathsf{D}_2^{2,1,2}$ operators. These operators are based on those of Mattsson [59] and Fernandez and Zingg [57]. Artificial dissipation is added with operators based on those by Diener [61]. The dissipation operators are chosen to be have order one greater than the order of the inviscid operators.

Applying the $\mathsf{D}_1$ operator to the inviscid fluxes $\frac{\partial \hat{H}}{\partial \eta}$ is straightforward. Second derivatives with variable coefficients and mixed derivatives, as is the case with the Navier-Stokes equations, is more complicated. The viscous flux can be decomposed as

$$\hat{H}^v = \mathsf{B}^r \frac{\partial Q^p}{\partial r} + \mathsf{B}^s \frac{\partial Q^p}{\partial s} + \mathsf{B}^\eta \frac{\partial Q^p}{\partial \eta} \tag{A.4}$$

where $Q^p$ is the vector of primitive variables

$$Q^p = \begin{bmatrix} u & v & w & T \end{bmatrix}^T \tag{A.5}$$

The $\mathsf{B}$ matrices can be found in previous work by Katz [15]. The derivative of $\hat{H}^v$ with respect to $\eta$ can then be found by

$$\frac{\partial \hat{H}^v}{\partial \eta} \approx \mathsf{D}_1 \left[ \mathsf{B}^r \left( \frac{\partial Q^p}{\partial r} \right)^h + \mathsf{B}^s \left( \frac{\partial Q^p}{\partial s} \right)^h \right] + \mathsf{D}_2 \left[ \mathsf{B}^\eta \left( Q^p \right) \right] \tag{A.6}$$

The partial derivatives with respect to $r$ and $s$ are computed using the Lagrange polynomial mappings, with no averaging on element boundaries.

Boundary conditions are implemented via penalty terms, which are based on the well-posedness of the Navier-Stokes equations [91]. By definition, the root of a strand lies on a solid surface, while the strand tips coincide with far field boundary conditions. Derivations for both of these can be found in the works of Svärd [93,94]. The penalty terms lead to strict time stability of the method, as well as improved convergence of the numerical method over traditional boundary condition implementations.

# APPENDIX B
## Strand Stack Mappings

A generic two-dimensional $p$-order polynomial in $r$ and $s$ with coefficients $\alpha_{j,k}$ can be written as:

$$P(r,s) = \sum_{j=0}^{p} \sum_{k=0}^{p-j} \alpha_{jk} r^j s^k \tag{B.1}$$

A generic one-dimensional $n$-order polynomial in $\eta$ with coefficients $\beta_q$ can be written as:

$$P(\eta) = \sum_{q=0}^{n} \beta_q \eta^q \tag{B.2}$$

The mapping between $(x, y, z)$ and $(r, s, \eta)$ is performed using Lagrange polynomials. Also known as shape functions, a set of $N$ polynomials is defined over a set of $N$ discrete points with known values. Interpolations can then be performed within the set. The complete interpolation function is given by

$$\phi(\eta) = \sum_{n=0}^{N-1} \phi_n \ell_n(\eta) \tag{B.3a}$$

$$\phi(r,s) = \sum_{i=0}^{N-1} \phi_i L_i(r,s) \tag{B.3b}$$

for one and two dimensions, respectively. The $\phi_n$, $\phi_i$ values are known values of $\phi$ at the points $n$, $i$. The Lagrange coefficients can be found by enforcing the property

$$\begin{aligned} \ell_n(x_m) &= 0 \\ \ell_n(x_n) &= 1 \end{aligned} \tag{B.4}$$

For example, for a one-dimensional second-order interpolation, three points are required. Each polynomial has three coefficients, which are represented as $\beta_{nq}$. Applying the above property results in the linear system

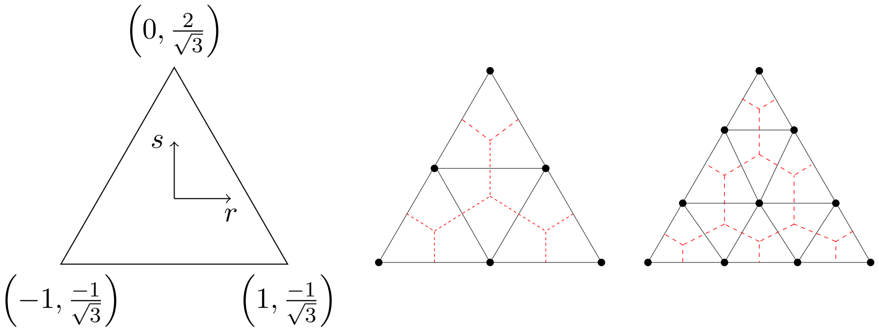
$$\begin{bmatrix} 1 & \eta_0 & \eta_0^2 \\ 1 & \eta_1 & \eta_1^2 \\ 1 & \eta_2 & \eta_2^2 \end{bmatrix} \begin{bmatrix} \beta_{00} & \beta_{10} & \beta_{20} \\ \beta_{01} & \beta_{11} & \beta_{21} \\ \beta_{02} & \beta_{12} & \beta_{22} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{B.5}$$

Solution of this system gives the Lagrange coefficients. The two-dimensional case is similar. Final interpolation of $\phi(\eta)$, in matrix form, is

$$\phi(\eta) = \begin{bmatrix} 1 & \eta & \eta^2 \end{bmatrix} \begin{bmatrix} \beta_{00} & \beta_{10} & \beta_{20} \\ \beta_{01} & \beta_{11} & \beta_{21} \\ \beta_{02} & \beta_{12} & \beta_{22} \end{bmatrix} \begin{bmatrix} \phi_0 \\ \phi_1 \\ \phi_2 \end{bmatrix} \tag{B.6}$$

## B.1 Interpolation and Derivatives in a Strand Stack

Application of equation B.3b to the reference triangle shown in figure B.1a allows us to map an arbitrary triangle in $\mathbb{R}^3$ to the reference triangle in the computational space $(r, s)$. This mapping is the same for all triangles in the strand grid. Derivatives of $\phi$ in the triangle


$\left(0, \frac{2}{\sqrt{3}}\right)$
$s$
$r$
$\left(-1, \frac{-1}{\sqrt{3}}\right)$
$\left(1, \frac{-1}{\sqrt{3}}\right)$

(a) Reference triangle (b) Quadratic (c) Cubic

Fig. B.1: Reference triangle used in this work. Also depicted are second- and third-order triangles. Red dashed lines indicate the median-dual control volumes

are determined by

$$\frac{\partial \phi}{\partial r}(r,s) = \sum_{i=0}^{N-1} \phi_i \frac{\partial}{\partial r}(L_i) \tag{B.7a}$$

$$\frac{\partial L_i}{\partial r} = \sum_{j=0}^{p} \sum_{k=0}^{p-j} j\alpha_{jk} r^{j-1} s^k \tag{B.7b}$$

The derivatives in $(r, s)$ can be changed to derivatives in $x$ using the inverse Jacobian mapping terms

$$\frac{\partial \phi}{\partial x} = \frac{\partial \phi}{\partial r}\frac{\partial r}{\partial x} + \frac{\partial \phi}{\partial s}\frac{\partial s}{\partial x} \tag{B.8}$$

Derivatives in $y$ and $z$ are similar. Along the strands, no analytic mapping exists to change the nodal strand spacing to a uniform spacing in $\eta$. Therefore it is approximated using the one-dimensional Lagrange polynomials in the region of interest. It should be noted that polynomials constructed in this manner are only valid in a small region of $eta$.

A full three-dimensional interpolation function inside a strand cell with $N$ points and using $M$ strand nodes can be formed by using an outer product of the two Lagrange polynomials

$$\phi(r,s,\eta) = \sum_{i=0}^{N-1} \sum_{n=0}^{M-1} \phi_{in} L_i(r,s) \ell_n(\eta) \tag{B.9}$$

A matrix formulation with a second-order (6 node) triangle and using 3 strand layers looks like

$$A = \begin{bmatrix} 1 & r_0 & s_0 & r_0^2 & rs_0 & s_0^2 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & r_5 & s_5 & r_5^2 & rs_5 & s_5^2 \end{bmatrix} \tag{B.10a}$$

$$[I] = \begin{bmatrix} [A] & [A]\eta_0 & [A]\eta_0^2 \\ [A] & [A]\eta_1 & [A]\eta_1^2 \\ [A] & [A]\eta_2 & [A]\eta_2^2 \end{bmatrix} \begin{bmatrix} \\ \gamma \\ \\ \end{bmatrix} \tag{B.10b}$$

where the $\alpha$ and $\beta$ coefficients have been combined into $\gamma$ coefficients.

For the domain connectivity, the first and second-derivatives of equation B.9 are needed. A simple templated C++ function can be written to compute the $(r, s, \eta)$ vector for any order derivatives, mixed or pure. This function is shown in figure B.2.

```
// This helper template computes a pseudo-factorial
// n is the number of derivatives
// i is the loop index
template<int n> int sf(int i);
template<> int sf<0>(int i) { return 1; }
template<int n> int sf(int i) { return i*sf<n-1>(i-1); }

// dr, ds, de are the number of derivatives
// in each dimension. 0 is no derivatives,
// 1 is the first derivative, etc.
template<int dr,int ds,int de>
Eigen::RowVectorXd compute_rse_vector(double r,
                                      double s,
                                      double e)
{
  int ip,jp,kp,ic,jc,kc;

   // N is the number of nodes in the triangle
   // M is the number of nodes in the strand direction
   // p is the order of the triangle
  Eigen::RowVectorXd rse(N*M);

  for (int k=0; k<M; k++) {
    for (int i=0; i<=p; i++) {
      for (int j=0; j<=p-i; j++) {
        ip = max(0,i-dr);
        jp = max(0,j-ds);
        kp = max(0,k-de);
        ic = sf<dr>(i);
        jc = sf<ds>(j);
        kc = sf<de>(k);
        rse(q) = ic*jc*kc*pow(r,ip)*pow(s,jp)*pow(e,kp);
        q++;
      }
    }
  }
  return rse;
}
```

Fig. B.2: C++ code for computing the tensor product and any order derivatives in a strand stack. This function uses the Eigen linear algebra library [95]

# APPENDIX C

## Locating the Closest Strand Node with an Octree

An octree is the simplest kind of three-dimensional spatial partitioning tree, allowing for fast nearest-neighbor searches and other spatial queries. The two-dimensional variant is known as a quadtree. They are used extensively in games (such as Minecraft) for efficiently storing terrain data in huge virtual environments. In this work, a basic octree is utilized to efficiently determine the closest strand node to a given Cartesian node. The recursive tree structure is visualized in figure C.1.

Each node in an octree is classified as either a root node or a leaf node. A root node has exactly eight children, one for each octant of the root node. These children can be leaf or root nodes. A leaf node does not have any children, but does contain the data of interest. Each leaf node can hold $N$ points of data before it is split into eight children and is converted into a root node. Each node of the octree is defined by an origin, $o_i$, and a half-spacing around the origin, $\Delta x_i$, where subscript $i$ is used in tensor notation.

The first step is to build the octree. The initial root node is specified to be large enough to cover the entire strand grid. The location of each strand node is computed in turn. The location, along with the $(j, n)$ index of the strand node, is inserted into the octree. The algorithm is recursive in nature, and is depicted in algorithm 1.

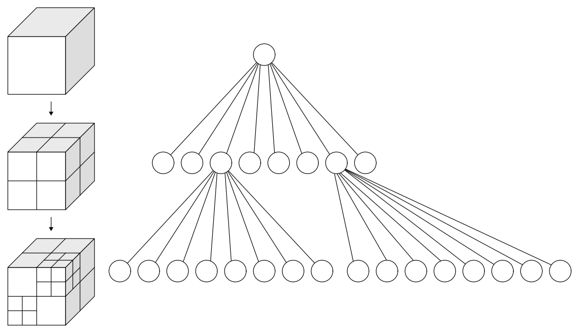


Fig. C.1: Depiction of octree subdivision procedure and resulting tree [96]

The octant can be easily found with bitwise operations. An example code is shown in figure C.2. Once the octree is built, it can then be used to identify the nearest strand node to any point in space. For this work, the algorithm developed by Hjaltason [97] is used, as it provides not just the closest node, but a sorted list of the closest $N$ strand nodes. The method uses a prioirity queue to maintain a list of objects, sorted by distance, with the closest elements on the top of the queue. The algorithm is iterative in nature, and is detailed in algorithm 2. The elements of the queue are not just the closest strand nodes, but also the closest octree nodes. As the search progresses down the tree, the distance to each octant is computed and then added to the priority queue. The octant which contains the search point has its distance set to 0. This will place it on the top of the queue, moving the search further down the tree. Once a leaf node is reached, the distance to each strand node in that leaf node is computed and added to the priority queue. As strand nodes are encountered in the queue, they are popped off and added to a list of return values.

```cpp
int getOctant(const vec3& loc)
{
  int oct = 0;
  if (loc[0] >= origin[0]) oct |= 4;
  if (loc[1] >= origin[1]) oct |= 2;
  if (loc[2] >= origin[2]) oct |= 1;
  return oct;
}
```

Fig. C.2: C++ code for computing the octant a point is located in inside of an octree node. The `vec3` type is an array of size 3

**Algorithm 1** Algorithm for building the octree from strand node data

```
function InsertStrandNode(OctreeNode,StrandNode)
    if OctreeNode is not a Leaf Node then
        Determine octant containing StrandNode
        InsertStrandNode(children[octant],StrandNode)
    else
        if $n = N$ then
            Allocate children[8]
            for all old ← old strand nodes do
                Determine octant containing old
                InsertStrandNode(children[octant],old)
            end for
            Determine octant $oct$ containing $s$
            InsertStrandNode(children[octant],StrandNode)
        else
            Append strand node to data
            $n \leftarrow n + 1$
        end if
    end if
end function
```

**Algorithm 2** Algorithm for locating the closest N strand nodes

```
function GetNearestStrandNodes(location,N)
    Values ← Empty List
    Add root of octree to priority queue
    while Priority queue is not empty do
        Q ← Pop top element from priority queue
        if Q is Strand Node then
            Add Q to Values
            if Size of Values == N then
                return Values
            end if
        else
            if Q is a leaf node then
                Compute distance to all strand nodes and add to queue
            else
                Compute distance to all children octants and add to queue
            end if
        end if
    end while
    Error: Empty queue
end function
```

# APPENDIX D

## Testing a Strand Cell using Newton's Method

A Newton's method is employed for determining if a Cartesian node is inside of a given strand stack. If it is, then as a result, the $(r, s, \eta)$ location in the strand cell is returned. figure D.1 is a flowchart for the algorithm used in this work.

The Newton's method is used to minimize the distance squared function $d^2(r, s, \eta)$, where (using Einstein notation) $d_i = x_i^C - x_i(r, s, \eta)$, $x_i^C$ is the location of the Cartesian node, and $x_i(r, s, \eta)$ is the location in space using strand cell interpolations. The squared distance function is used as it is a smooth function in space, with the same minimum as the distance function. The Newton minimization equation for a function of a vector of variables $\boldsymbol{r}$ is

$$\boldsymbol{r}^{n+1} = \boldsymbol{r}^n - \boldsymbol{H}(f(\boldsymbol{r}^n))^{-1}\nabla f(\boldsymbol{r^n}), \tag{D.1}$$

where $k$ is the iteration counter. Thus, it is necessary to compute the gradient and Hessian of the squared distance function. Both are derived here.

$$\frac{\partial d^2}{\partial r_j} = \frac{\partial d_i d_i}{\partial r_j} = 2d_i\frac{\partial d_i}{\partial r_j}, \tag{D.2a}$$

$$\frac{\partial d_i}{\partial r_j} = -\frac{\partial x_i}{\partial r_j} = -\frac{\partial}{\partial r_j}\left(\sum\sum x_i L(r, s)\ell(\eta)\right), \tag{D.2b}$$

where the derivatives of the Lagrange polynomials are computed as described in Appendix B. The Hessian is

$$\frac{\partial}{\partial r_k}\left(\frac{\partial d^2}{\partial r_j}\right) = \frac{\partial}{\partial r_k}\left(2d_i\frac{\partial d_i}{\partial r_j}\right) = 2\left[d_i\frac{\partial^2 d_i}{\partial r_k \partial r_j} + \frac{\partial d_i}{\partial r_k}\frac{\partial d_i}{\partial r_j}\right], \tag{D.3a}$$

$$\frac{\partial^2 d_i}{\partial r_k \partial r_j} = -\frac{\partial^2 x_i}{\partial r_k \partial r_j} = -\frac{\partial^2}{\partial r_k \partial r_j}\left(\sum\sum x_i L(r, s)\ell(\eta)\right). \tag{D.3b}$$

While the basic procedure is simple, important corner cases greatly increased the complexity. These corner cases were, no pun intended, when the Cartesian node landed directly

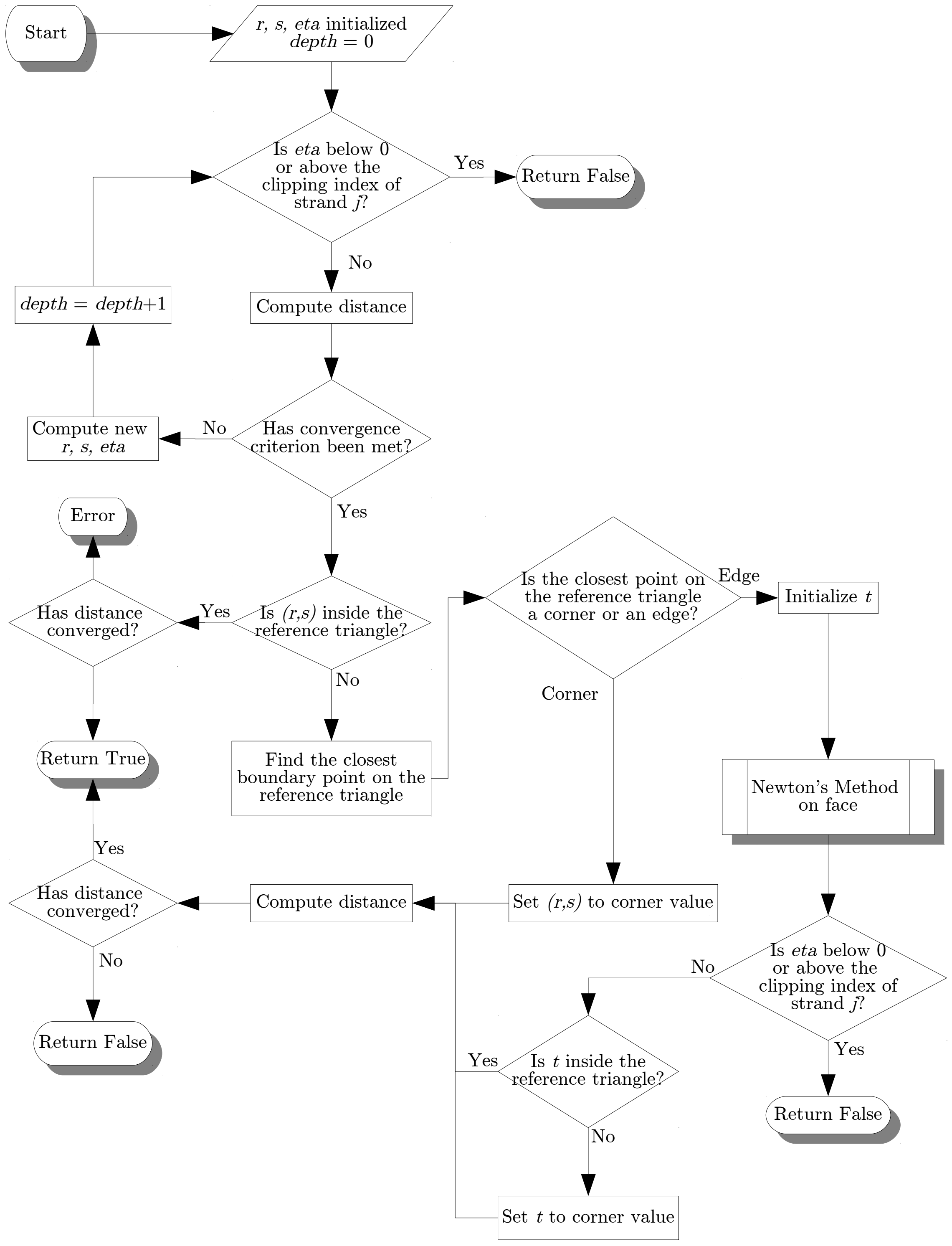

Start
r, s, eta initialized
depth = 0
Is eta below 0 or above the clipping index of strand j?
Yes
Return False
No
depth = depth+1
Compute distance
Compute new r, s, eta
No
Has convergence criterion been met?
Yes
Error
Has distance converged?
Yes
Is (r,s) inside the reference triangle?
No
Is the closest point on the reference triangle a corner or an edge?
Edge
Initialize t
Corner
Return True
Find the closest boundary point on the reference triangle
Newton's Method on face
Yes
Has distance converged?
Compute distance
Set (r,s) to corner value
No
Is eta below 0 or above the clipping index of strand j?
No
Yes
Return False
Return False
Yes
Is t inside the reference triangle?
No
Set t to corner value

Fig. D.1: Flowchart for the Newton's method

on, or extremely close to, a corner or face of the strand stack. In these cases, the Newton's method could incorrectly claim that the node did not land inside *any* strand stack, due to roundoff error. To address this issue, the Newton's method was allowed to converge completely, no matter whether the node was inside or outside of the strand stack. Technically, the Lagrange polynomials are not valid outside the strand stack, but since it is outside, accuracy of the method does not matter. One exception to this was if $\eta$ ever went below 0 or above the clipping index of strand $j$, it was immediately rejected. $r$ and $s$, however, were not tested until convergence.

Once convergence is reached, the location is tested to see if it lies inside or outside the reference triangle. Each edge of the reference triangle can be represented by an equation. These equations, starting at the bottom edge and moving counterclockwise, are

$$s_1 = -\frac{1}{\sqrt{3}} \tag{D.4a}$$

$$s_2 = \frac{2}{\sqrt{3}} - \frac{3}{\sqrt{3}}r \tag{D.4b}$$

$$s_3 = \frac{2}{\sqrt{3}} + \frac{3}{\sqrt{3}}r \tag{D.4c}$$

Determining whether the converged solution $(r^\star, s^\star)$ lies outside these equations is a matter of plugging $r^\star$ into each edge equation and comparing $s$ and $s^\star$. If any of those tests are false, then the node lies outside the reference triangle. However, to account for the corner cases, another step must be used. Extending the edges to infinity separates the $(r, s)$ plane into seven differing regions: The inside, three corner regions, and three edge regions. If two of the tests are false, then the node lies in a corner region. If only one is false, then it lies in an edge region. The next step is to locate the closest boundary point on the triangle. If the node lies in a corner region, then the closest point is the corner. $r^\star$ and $s^\star$ are set to the corner values, and the distance is evaluated to determine if the Cartesian node lies on the corner. If the closest point is an edge, then an initial guess is determined and a two-dimensional Newton's method in $t$ and $\eta$ is used to find the closest point on the face.

The edge is parameterized so that $r$ and $s$ are functions of $t$, with $0 \le t \le 2$. The functions are

$$r = r_0 + t\delta r \tag{D.5a}$$

$$s = s_0 + t\delta s \tag{D.5b}$$

where $r_0$, $s_0$, $\delta r$, and $\delta s$ are dependent on the edge being tested. The gradient of $d^2$ on the face is

$$\frac{\partial d^2}{\partial t} = \frac{\partial d_i d_i}{\partial t} = 2d_i \frac{\partial d_i}{\partial t} = 2d_i \left( \frac{\partial d_i}{\partial r}\delta r + \frac{\partial d_i}{\partial s}\delta s \right) = \frac{\partial d^2}{\partial r}\delta r + \frac{\partial d^2}{\partial s}\delta s \tag{D.6a}$$

$$\frac{\partial d^2}{\partial \eta} = \frac{\partial d^2}{\partial \eta} \tag{D.6b}$$

and the $2 \times 2$ Hessian is

$$H_{tt} = H_{rr}\delta r^2 + 2H_{rs}\delta r \delta s + H_{ss}\delta s^2 + 2\left(\frac{\partial d_i}{\partial t}\right)^2 \tag{D.7a}$$

$$H_{t\eta} = H_{\eta t} = H_{r\eta}\delta r + H_{s\eta}\delta s + 2\frac{\partial d_i}{\partial \eta}\frac{\partial d_i}{\partial t} \tag{D.7b}$$

$$H_{\eta\eta} = H_{\eta\eta}, \tag{D.7c}$$

where the Hessian subscripts indicate elements in the original $3 \times 3$ Hessian, and

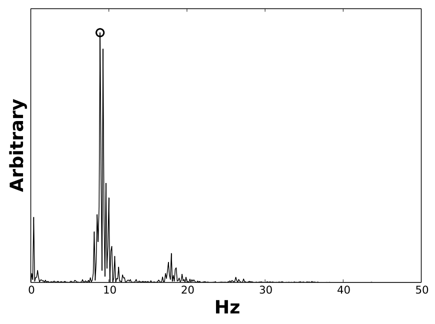
$$\frac{\partial d_i}{\partial t} = \frac{\partial d_i}{\partial r}\delta r + \frac{\partial d_i}{\partial s}\delta s \tag{D.8}$$

Once the Newton's method converges to the minimum $t$ is tested to see if it lies between 0 and 2, and the minimum distance is tested to determine if the Cartesian node lies directly on the face. The corner cases of Cartesian nodes lying on strand or cell boundaries are handled in this manner, with good results.

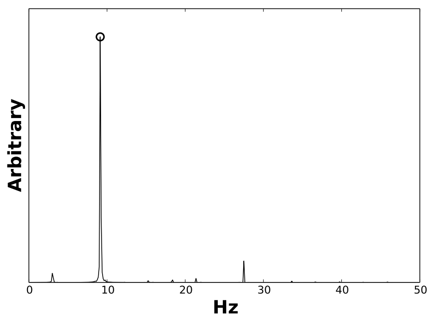
# APPENDIX E

## Unsteady Laminar Sphere Fourier Transforms

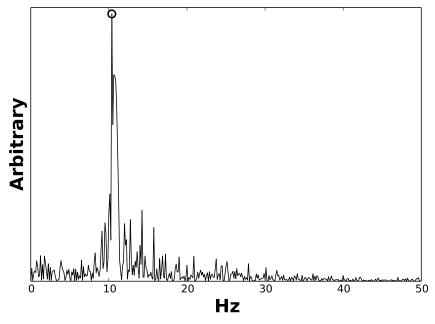
This appendix contains the power spectrums of the velocity probes for the unsteady laminar sphere cases. The probe was placed at $(4, 0, 0)$, 3.5 diameters behind the edge of the sphere. The largest peak was used as the dominant shedding frequency. Occasionally, a high-pass filter was used to discard extraneous lower frequencies. The frequencies used to compute the Strouhal numbers are marked with a circle. The power spectrums for Star-CCM+ are shown in figure E.2, for the overset mesh in figure E.3, and the strand grid alone in figure E.4.



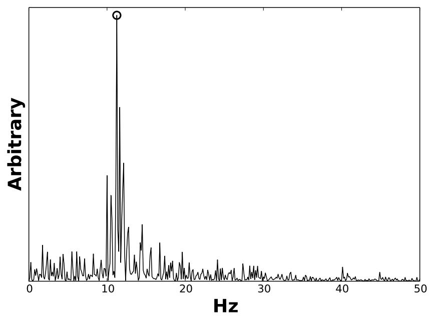
(a) Re = 400



(b) Re = 600

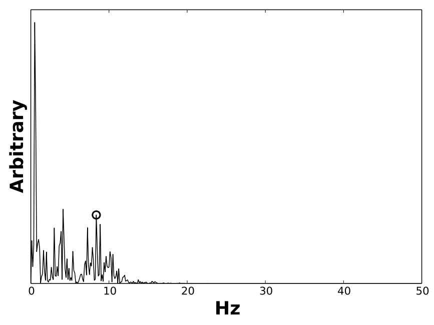


(c) Re = 1000

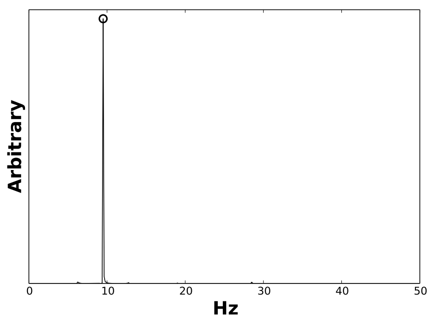


(d) Re = 2000

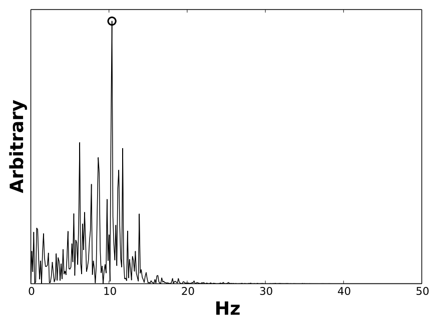
Fig. E.1: Star-CCM+



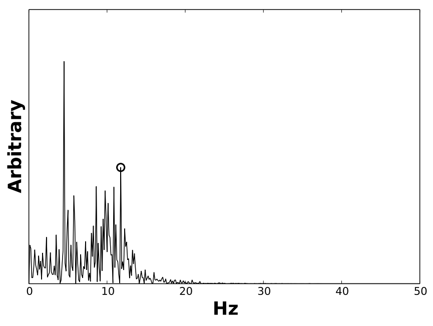
(a) Re = 400



(b) Re = 600

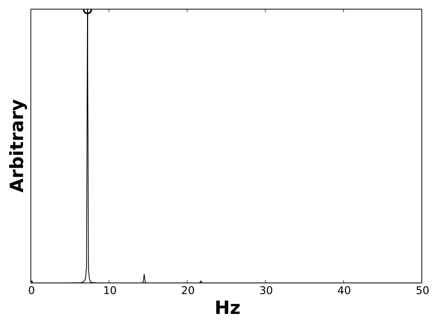


(c) Re = 1000

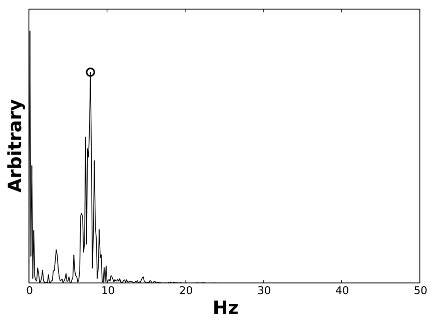


(d) Re = 2000

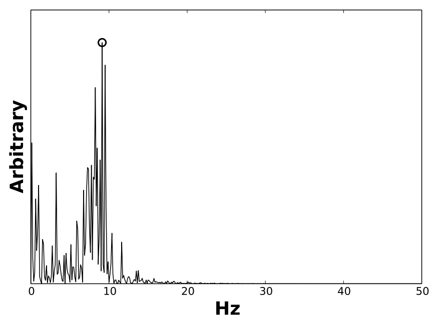
Fig. E.2: Overset mesh with second-order solver in the strand grid



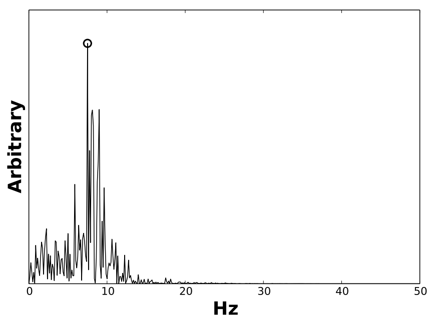
(a) Re = 400



(b) Re = 600

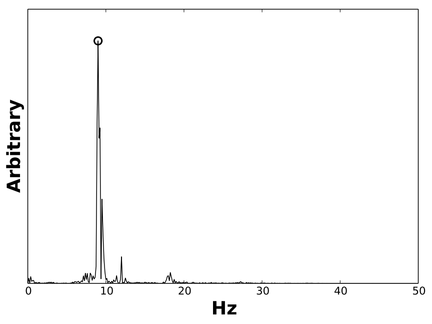


(c) Re = 1000

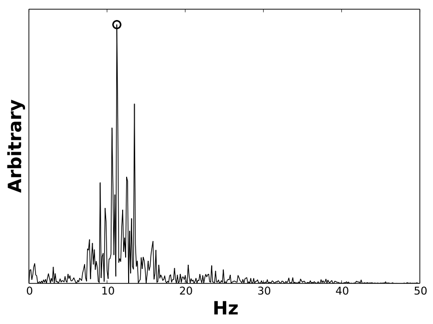


(d) Re = 2000

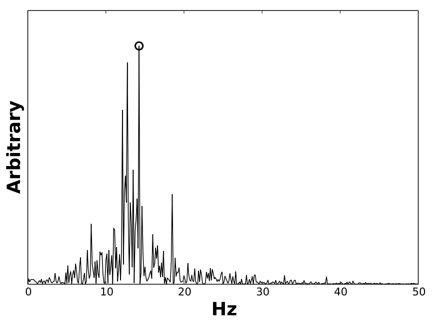
Fig. E.3: Overset mesh with flux correction solver in the strand grid



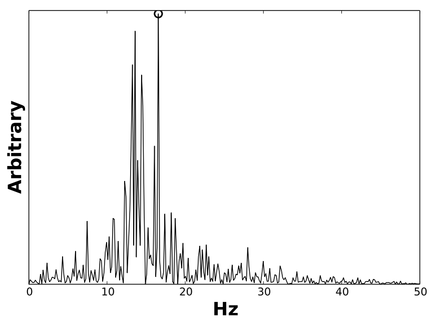
(a) Re = 400



(b) Re = 600



(c) Re = 1000



(d) Re = 2000

Fig. E.4: Strand grid only with flux correction solver