POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

Extension of a 2D Multiblock Structured Overset Suite to a Fully 3D Multiblock Unstructured Flow Solver

ALEXANDRE DESMARAIS
Département de génie mécanique

Mémoire présenté en vue de l'obtention du diplôme de *Maîtrise ès sciences appliquées*
Génie mécanique

Décembre 2020

POLYTECHNIQUE MONTRÉAL

affiliée à l'Université de Montréal

Ce mémoire intitulé :

Extension of a 2D Multiblock Structured Overset Suite to a Fully 3D Multiblock Unstructured Flow Solver

présenté par **Alexandre DESMARAIS**

en vue de l'obtention du diplôme de *Maîtrise ès sciences appliquées*

a été dûment accepté par le jury d'examen constitué de :

ACKNOWLEDGEMENTS

I would like to thank my professor and research director Éric for his advice and support throughout my master's program. Without you, I would not be where I am today, and I sincerely thank you for that.

Of course, all the work done in this thesis was not possible without the help of the extraordinary team in our research laboratory. Thank you very much Simon for your many tips and indispensable help in improving overset algorithms. I appreciate it very much. Thank you also to Frédéric, Matthieu, Miguel, Pierre, Réda, Anthony, Aviral, Alexandros, Benjamin, Maxime, Charles, Vincent P-C and Vincent L., Michael and Hélène for your support and amazing attitude. I will miss our lively discussions in the lab!

Thank you to my friends Bénédicte, Danika, Rémi, Mikaël P., Alexandre, Mathieu, Félix, Mikaël F., Gabriel and Vincent for supporting me during the last 6 years of my bachelor and master's degrees. You are the best!

I would also like to pay special attention to my family who has supported me throughout my academic career. Their constant support has allowed me to get through many personal hardships and I am eternally grateful to them. A big thank you to my mother and father, my sister Catherine and Daniel, my grandmother, Uncle François, Uncle Jean and Diane, Aunt Anne Marie and Uncle Chuck for accompanying me throughout my academic career. I adore you all.

RÉSUMÉ

Ce présent mémoire illustre les récents développements apportés aux logiciels chimères du laboratoire de recherche du professeur Éric Laurendeau à Polytechnique Montréal. Dans la dynamique des fluides numériques mieux connue en anglais sous le nom de « Computational Fluid Dynamics » (CFD), les méthodes chimères aussi connues en anglais sous le nom d'« overset » permettent de faire des simulations multigéométries complexes sans lesquelles le processus de génération de maillages pourrait être très compliqué, soit même impossible. Essentiellement, ces méthodes permettent de lier et d'assembler des maillages individuels contenant chacun différentes géométries en un seul par le biais d'interpolations et d'interfaces. Ceci permet de grandement réduire le temps à générer des maillages, car ces derniers peuvent être générés individuellement. Conséquemment, cela permet de passer plus de temps à exécuter des simulations ou même implémenter de nouvelles fonctionnalités au sein de solveurs de CFD, ce qui est très utile dans un environnement de travail industriel et de recherche. Pour que les méthodes chimères fonctionnent, il est nécessaire d'effectuer quelques étapes au niveau du préprocesseur d'un solveur de CFD. Cela normalement inclut un processus de découpage, un algorithme de recherche de paires de cellules donneuses et interpolées, et un algorithme d'interpolation. Une fois que ces étapes sont complétées, la simulation peut ensuite être effectuée.

Dans le cadre de ce laboratoire de recherche, la présente implémentation 2D a certains problèmes connus non résolus. Également, ces méthodes 2D sont difficilement transférables à des applications 3D lors des transferts technologiques avec l'industrie qui se servent de solveurs de CFD 3D à topologies structurée et non structurée.

La partie 2D de ce mémoire concerne les solutions aux problèmes au sein du préprocesseur chimère de NSCODE qui est un solveur de CFD 2D structuré. Plusieurs problèmes avec les méthodes implémentées dans le passé ont été identifiés et résolus dans le cadre de ce mémoire. Le premier problème concerne les erreurs de décalage avec les poids d'interpolation qui sont calculés aux nœuds des cellules au lieu d'être calculés au centre des cellules. Un nouveau schéma d'interpolation est mis en place dans lequel la précision et la fidélité du schéma précédent sont retenues tout en calculant les poids d'interpolation au centre des cellules. Le deuxième problème concernait la difficulté à générer des maillages de type «collar» qui sont un type particulier de maillages qui présentent une discrétisation plus fine de la géométrie pour un endroit en particulier du maillage d'arrière-plan. Des améliorations à NSGRID qui est le mailleur 2D structuré du laboratoire de recherche ont été effectuées. Ce dernier permet maintenant la génération de «collar grids» avec une projection normale à l'aide d'arguments d'entrée simplifiés. Le troisième problème identifié concerne les discontinuités solide-solide qui se présentent lorsque 2 maillages chimères partagent la même géométrie solide comme par exemple: une grille de type «collar» avec un maillage d'arrière-plan. Les résultats numériques et visuels démontrent la correction géométrique employée pour recréer convenablement les cellules de type «halo» qui sont un type de cellule présentes sur les frontières d'un maillage. Dans le cadre de la méthode chimère, ces cellules sont importantes pour avoir de résultats numériques précis pour des valeurs dérivées comme le coefficient de pression et de friction. Des résultats numériques sont présentés pour 3 cas tests distincts: un cylindre laminaire, un profil d'aile NACA0012 en laminaire et en turbulent. La conclusion de ces différents cas tests permet de démontrer une amélioration lorsque la correction employée est utilisée pour les cas en laminaire qui présentent une géométrie possédant une courbure prononcée comme dans le cas du cylindre. Cependant, comme il est possible d'apercevoir avec les résultats numériques, des problèmes de discontinuités sont encore présents avec l'utilisation d'un modèle de turbulence. Des suggestions pour des travaux futurs consistent à regarder si la non-interpolation des dérivées de vitesse a un effet direct sur le calcul des variables de turbulence. Le quatrième problème identifié concerne le mauvais calcul des poids d'interpolation lors de l'utilisation d'un «collar grid» avec un maillage d'arrière-plan. La dernière partie de ce chapitre concerne les améliorations faites à l'algorithme de découpage où qu'un nouveau schéma à 5 points est proposé. Des résultats visuels où une comparaison à l'ancien schéma à 1 point est présentée sur le profil d'aile MDA.

Le chapitre 3D de ce mémoire concerne l'implémentation en 3D de techniques chimères programmées dans le cadre de ce mémoire à titre de préprocesseur au sein de CHAMPS qui est le solveur de CFD 3D non structuré du laboratoire de recherche. Ce chapitre met en valeur les complexités et subtilités des différents algorithmes qui forment le préprocesseur chimère. En premier lieu, l'assemblage de maillages individuels en une seule grille chimère est présenté. Ensuite, l'utilisation de la distance à la paroi comme critère pour déterminer les paires de cellules interpolées et donneuses est décrite. Un port de l'algorithme récursif de type «quad-tree» de NSCODE en un algorithme récursif 3D de type «oct-tree» est exposé. Également, puisque l'algorithme de découpage 2D de NSCODE est inapproprié pour une utilisation en 3D, l'implémentation d'une nouvelle méthode de découpage basée sur la méthode des rayons X provenant de la littérature est présentée. Une fois que chacun de ces algorithmes a été couvert, la présentation de l'algorithme pour déterminer le chevauchement à l'aide de l'algorithme récursif «oct-tree» est montrée. Finalement, des améliorations aux algorithmes et méthodes présentement implémentées sont proposées.

En somme, le travail accompli dans le cadre de ce mémoire permet à de futurs étudiants du

laboratoire de recherche d'avoir une fondation solide pour pousser les algorithmes 3D chimères à de nouveaux sommets par l'implémentation des plus récents développements présentés dans la littérature. Le travail effectué en 2D permettra aux étudiants d'avoir une meilleure compréhension des racines de la méthode chimère. Le nouveau préprocesseur non structuré 3D permettra de faciliter les transferts technologiques avec l'industrie puisqu'ils se servent également de solveur de CFD 3D structurés et non structurés.

ABSTRACT

The present thesis addresses the recent developments made to the overset framework of Professor Éric Laurendeau's research lab at Polytechnique Montreal. In Computational Fluid Dynamics (CFD), overset methods allow complex multi-geometry simulations for which without its use the meshing process can be tedious, if not impossible. Essentially, these methods link individual meshes containing different geometries together by the means of interpolation and interfaces. This reduces the time spent on meshing since elements of the geometry can be meshed out individually, and thus more time can be spent on running simulations or implementing new features in a CFD flow solver which is a highly valuable resource in a research/industrial environment. In order for overset methods to work, a couple of steps are taken at the preprocessor level of the CFD flow solver in question. They usually include a hole cutting process, a donor search algorithm, and an interpolation algorithm. Once these steps are done, the simulation can then be performed.

In the case of this research lab, the current framework has some pitfalls with its 2D implementation and the ability of its methods to be used for 3D applications in the case of doing technological transfers with the industry who use both 3D CFD flow solvers with structured and unstructured topology.

The 2D part of this thesis concerns the solutions to the problems inside the overset preprocessor of NSCODE which is a 2D structured CFD flow solver. Several problems with the methods implemented in the past have been identified and are solved in the case of this thesis. The first problem concerns offset errors with the interpolation weights calculated at the cell vertices instead of the cell centers. A new interpolation scheme is devised in which the precision and accuracy of the first one are retained while having the weights properly calculated at the cell centers. The second issue was with the rigidity of generating collar grids which is a type of mesh that possesses a finer discretization of the geometry for a specified region of the background mesh. Improvements to NSGRID which is the 2D structured mesher of the research lab have been made in that regard. Users can now generate normally projected collar grids with easier inputs. The third issue concerns the solid-solid discontinuity problem that occurs when 2 overset meshes share the same solid geometry (i.e. a collar grid with a background mesh). The results demonstrate the geometric correction that is employed to recreate halo cells which are cells found on the fringes of the mesh. In the case of the overset method, these cells are necessary for proper computation of derived values such as the pressure coefficient and coefficient of friction. Numerical results are demonstrated in 3 distinct test cases: a laminar cylinder, a laminar NACA0012 and a turbulent NACA0012 airfoil. The conclusion from these test cases shows improvement when used in laminar cases with geometries with high curvature such as the cylinder. However, as highlighted with the numerical results, some problems are still present with the use of a turbulence model. Future work suggestions are to investigate if the effect of the non-interpolation of velocity derivatives have a direct impact on the calculation of turbulence variables. The fourth issue is that improper interpolation weights are calculated when using a collar grid with a background grid. In this case, the implementation of a correction of the wall distance is made to these interpolated cells to help out with the convergence of difficult test cases. The final part of this chapter presents improvements made to the hole cutting algorithm where a 5-point testing scheme for each cell is devised instead of the previous 1-point testing scheme who would fail with geometries with tight gaps. The results are presented on the MDA airfoil.

The 3D part of this thesis concerns the implementation that was programmed during the course of this thesis of overset methods as a new preprocessor inside CHAMPS which is a 3D unstructured CFD flow solver. This chapter highlights the complexities and intricacies of the different algorithms implemented that form the preprocessor. For starters, the algorithm showing the assembly of individual meshes into a single overset grid is explained. Afterwards, the use of the wall distance as a criterion for determining interpolated and donor cell pairs is described. A port of the quad-tree recursive boxing algorithm of NSCODE into an oct-tree recursive algorithm for CHAMPS is thoroughly examined. Since the 2D hole cutting method in NSCODE is not appropriate for 3D applications, the implementation of a new hole cutting algorithm for CHAMPS based on the X-ray method from the literature is portrayed. Once these algorithms are covered, the overlap detection algorithm in which the boxing solution is used in conjunction with the wall distance calculation for determining interpolated/donor cell pairs is shown. Finally, improvements to the current coded implementation are proposed.

In sum, the work done during this master enables future students of the research lab to have a solid foundation to push the 3D overset framework to new heights by the means of implementing the most recent developments found in the literature. The work done in 2D will allow students a better understanding of the fundamentals of the overset method. The newly created 3D unstructured framework will facilitate technology transfers with the industry since they also use both structured and unstructured 3D CFD flow solvers.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS AND ACRONYMS

AABB	Axis Aligned Bounding Box
ADT	Alternating Digital Tree
API	Application Programming Interface
CAD	Computer Aided Design
CDT	Constrained Delaunay Triangulation
CGNS	CFD General Notation System
CGT	Chimera Grid Tools
CHAMPS	CHApel Multi-Physics Software
$\boldsymbol{C_P}$	Pressure Coefficient
$\boldsymbol{C_f}$	Coefficient of Friction
DNS	Direct Numerical Simulation
$\boldsymbol{dS}$	Surface Element
$\boldsymbol{\partial\Omega}$	Boundary of a Control Volume
EGADS	Engineering Geometry Aircraft Design System
FVM	Finite volume method
$\boldsymbol{\vec{f}_e}$	External Volume Forces Vector
HPC	High-Performance Computing
IHC	Implicit Hole Cutting
$\boldsymbol{J}$	Jacobian
$\boldsymbol{1/J, J^{-1}}$	Inverse Jacobian
JST	Jameson, Schmidt, and Turkel
k	Coefficient of Thermal Conductivity
$\boldsymbol{\Lambda}$	Sweep Angle
$\boldsymbol{\lambda}$	Second Viscosity Coefficient
LES	Large Eddy Simulation
MPI	Message Passing Interface
$\boldsymbol{\mu}$	Dynamic Viscosity
MUSCL	Monotone Upstream-Centered Schemes for Conservation Laws
NS	Navier-Stokes
$\boldsymbol{\Omega}$	Control Volume
$\boldsymbol{p}$	Order of Convergence
PDE	Partial Differential Equation
$\boldsymbol{R^2}$	Coefficient of Determination

RANS	Reynolds Averaged Navier-Stokes
$\boldsymbol{\rho}$	Density
RVCG	Recursive Virtual Cartesian Grid
SA	Spalart-Allmaras
$\boldsymbol{t}$	Time
T	Static Temperature
$\bar{\bar{\boldsymbol{\tau}}}$	Viscous Stress Tensor
$\boldsymbol{\tau_{ij}}$	Stress Coefficient
TIOGA	Topology Independent Overset Grid Assembler
$\vec{\boldsymbol{v}}$	Velocity Vector with its Components: u, v, w
u,v,w	Cartesian Velocity Components
$\boldsymbol{X_\xi, Y_\xi}$	X and Y Coordinates Transformed in the ξ Referential
$\boldsymbol{X_\eta, Y_\eta}$	X and Y Coordinates Transformed in the η Referential

LIST OF APPENDICES

CHAPTER 1 INTRODUCTION

Since the early 1980s, overset methods have been in development and were precursors to many novel advancements in aerodynamics such as the space shuttle [1]. These methods are primarily used in Computational Fluid Dynamics (CFD) and are crucial for simulating fluids around complex geometries in CFD flow solvers.

1.1 Basic Concepts

In the case of this thesis, two different software are presented with two different approaches for overset algorithms. Without going into further details, the first one is a 2D structured approach and the second one is a 3D unstructured approach.

1.1.1 The RANS Equations

In aeronautics and aerospace, CFD allows engineers to design and simulate numerical models depicting accurate flight conditions on a variety of geometrical shapes. Coupled with the rising available computational power of modern computers, CFD managed in part to replace costly wind tunnel testing by doing it entirely numerically.

All of this is made possible by solving the Navier-Stokes (NS) equations. They consist of the following [2]:

$$\frac{\partial}{\partial t}\int_{\Omega}\rho\, d\Omega + \oint_{\partial\Omega}\rho(\vec{v}\cdot\vec{n})\, dS = 0 \tag{1.1}$$

$$\frac{\partial}{\partial t}\int_{\Omega}\rho\vec{v}\, d\Omega + \oint_{\partial\Omega}\rho\vec{v}(\vec{v}\cdot\vec{n})\, dS = \int_{\Omega}\rho\vec{f}_e\, d\Omega - \oint_{\partial\Omega}p\vec{n}\, dS + \oint_{\partial\Omega}(\bar{\bar{\tau}}\cdot\vec{n})\, dS \tag{1.2}$$

$$\frac{\partial}{\partial t}\int_{\Omega}\rho E\, d\Omega + \oint_{\partial\Omega}\rho H(\vec{v}\cdot\vec{n})\, dS = \oint_{\partial\Omega}k(\nabla T\cdot\vec{n})\, dS + \int \Omega(\rho\vec{f}_e\cdot\vec{v} + \dot{q_h})\, d\Omega + \oint_{\partial\Omega}(\bar{\bar{\tau}}\cdot\vec{v})\cdot\vec{n}\, dS \tag{1.3}$$

The first equation (1.1) consists of the integral form of the continuity equation. The second equation (1.2) is the integral form of the momentum conservation. The third equation (1.3) is the energy equation. Combined together, these equations represent the NS equations.

Due to the current cost of computing these equations, a modification of these equations is done. The resulting modified equations are the Reynolds Averaged Navier-Stokes (RANS) equations. They are made by adding two extra terms to the NS equations which are the Reynolds stress term and the turbulent heat flux vector [2]. This approach to solving the NS equations allows the use of coarser meshes for simulating flows and thus reduces the computing cost as opposed to more demanding methods such as Large Eddy Simulation (LES) and Direct Numerical Simulation (DNS) [2]. In the case of the present 2D and 3D framework, a finite volume approach is used.

1.1.2 Finite Volume Flow Solver

A finite volume approach consists of using several control volume cells and calculating fluxes passing through the boundary of each of the control volumes [2]. An example of this is shown in Figure 1.1. A mesh consists of a collection of all these control volumes in order to make a discretization of the entire fluid domain. A mesh can have structured or unstructured topology depending on the application on which the flow solver is based. In a structured mesh such as in Figure 1.2, connectivity between the different cells is implicit. In an unstructured mesh such as in Figure 1.3, connectivity must be explicited for each cell of the mesh. Both of these approaches to meshing present advantages and inconveniences.

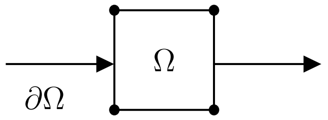


Figure 1.1 A cell and its control volume

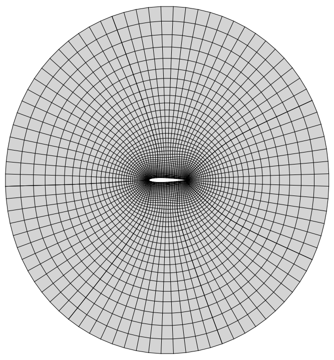


Figure 1.2 A structured mesh around a NACA0012 airfoil

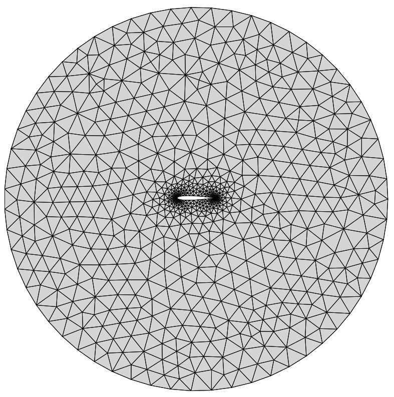


Figure 1.3 An unstructured mesh around a NACA0012 airfoil

Generally speaking, CFD flow simulations are a 3-step process. The first step consists of mesh generation in which the necessary grids are generated to accurately model the geometry on which the flow simulations are performed. Once the mesh generation is done, the actual flow simulations are then performed on the generated grid. The final step consists of the post-processing in which analyses of the produced results are done.

1.1.3 Overset Methods

Overset methods consist of using two or more independent meshes and linking them together for simulating complex geometries such as an airfoil with a flap and a slat as shown in Figure 1.5. These methods are enabled by using several algorithms to link the different meshes together. An early use of overset methods was in the design of the Integrated Space Shuttle Launch Vehicle (SSLV) as seen in Figure 1.6 in 1988 [3]. A problem encountered in the design of the SSLV was the mesh generation process that was difficult because of the complex geometries involved [3]. New tools were then developed to ease in the mesh generation process which included overset methods. Instead of meshing out the entire geometry which is fairly complicated in a structured mesh, individual components can be meshed out on different grids and then with an overset preprocessor linked together to carry out flow simulations. Usually, this means less time spent on the meshing process and/or also making possible simulating geometries impossible to mesh in a conventional way. Presently, common applications of the method are for again simulating complex geometries, but also for applications such as grids in relative motion.

In order for overset methods to work, several changes are implemented at the preprocessor level of the flow solver in question. They usually include a hole cutting process, a donor search algorithm, and an interpolation algorithm. Once these three steps are done and the interpolation weights for the donor cells are calculated, the simulation can then be carried out. An example of an assembled 2D structured overset mesh ready for flow simulation is shown in Figure 1.5. The individual structured meshes highlighted in green, red and blue were generated using as a basis the MDA geometry shown in Figure 1.4. They were then assembled using an overset preprocessor with the end result being Figure 1.5.

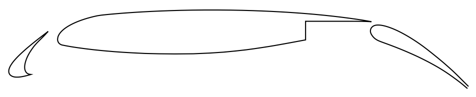


Figure 1.4 MDA airfoil geometry with its slat and flap

It is important to note that even though overset methods were originally developed for use with structured meshes and flow solvers, it is not uncommon to see nowadays overset applications with unstructured meshes and flow solvers as it is the case in this thesis with the 3D framework that is presented. They notably present an advantage for unstructured

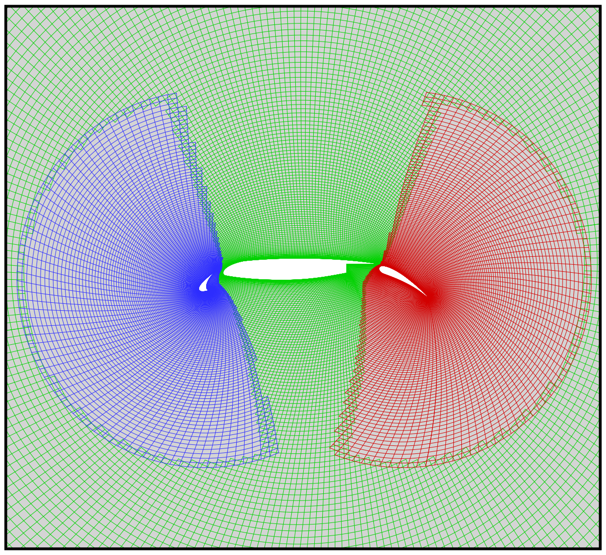


Figure 1.5 MDA airfoil with flap and slat assembled using the overset algorithm from NSCODE

Figure 1.6 Space Shuttle Launch Vehicle (SSLV). Figure from NASA (2007) [4]

applications by having an easier mesh generation process and easy local refinement of sections in a mesh.

1.2 Research Objectives

A common problem with overset methods is the treatment used to resolve solid-solid intersections between meshes such as an aircraft fuselage and a wing. This notably poses a problem when calculating forces and moments on the geometry as highlighted in Guay (2017) [5]. Multiple approaches to tackling this problem are highlighted in the scientific literature and will be further explored in the literature review.

The main objective of this master's project is to **solve solid-solid intersections in 3D**.

However, before being able to solve solid-solid intersections in 3D, it is necessary to familiarize ourselves with the research lab framework for using overset methods.

1.3 Problematic Elements

Two distinct problems are presented in this thesis. The first one pertains to the use of the existing overset module inside the 2D framework. The second one is concerning the existing framework for testing the overset methods in 3D.

1.3.1 2D - Problematic Elements

The 2D overset implementation is functional but presents certain issues in particular cases. The current problems encountered with the overset module are listed below :

- A discontinuity problem arising when two meshes sharing the same geometry are intersecting each other.
- Incorrect interpolation due to an offset error with the interpolation weights stored at the cell vertices instead of the cell centers.
- Incorrect weights calculated when using a collar grid.
- Failure of the hole cutting process when gaps around two geometries are too tight.

These 4 items are the main grievances of the 2D overset module and will be the object of the first chapter of this thesis.

1.3.2 3D - Problematic Elements

The second main problem inside the research group is that no framework is available to test methods and solutions developed in 2D when doing technological transfers with the industry in 3D. Since the 2D overset methods developed have reached a certain degree of maturity, it was only natural to pursue the 3D avenue for developing in-house an overset framework. Hence, a 3D overset pre-processor was coded as part of this thesis.

Most of the 2D methods and ideas for the oct-tree boxing algorithm and the overset cell classification using a wall distance criterion were inspired from the existing 2D solver overset module. However, not all the methods are portable to a 3D solver as for example the hole cutting process which in 2D is done using a Delaunay triangulation is very hard to implement in 3D. Solutions to this problem exist and will be detailed further in the development of this thesis. Also, in order for the overset algorithm to work in distributed memory as opposed to the shared memory implementation inside the 2D code, the code architecture and structure were thought out differently because of the supplementary communication interfaces needed between the computer nodes.

The primary focus of this chapter concerning the newly implemented 3D overset pre-processor will be to illustrate the various methods taken from the 2D overset suite and adapted for the unstructured 3D overset suite as well as the limitations posed by certain 2D structured overset algorithms when ported to a 3D unstructured flow solver.

1.4 Thesis Outline

Before going into detail on how the particular problems with the overset module are solved, a short but comprehensive literature review of the various existing overset methods used in 2D and 3D is presented. Afterwards, for the first part of this thesis pertaining to the 2D problems of the overset framework in NSCODE, an analysis of each of the problems is done. Solutions are then presented with pertinent numerical results. For the 3D chapter pertaining to the 3D overset module in CHAMPS, a description of the various implementations of the algorithms for each section of the 3D overset preprocessor is done. Numerical and visual results stemming from these algorithms are also shown. Finally, a synthesis of the work done is presented with a future works section recommending future improvements to the implemented 3D overset algorithms as well as possible solutions for the solid-solid discontinuities arising from the use of a turbulence model in 2D.

CHAPTER 2 LITERATURE REVIEW

This chapter covers the literature review that has been done on the various aspects of overset methods currently implemented inside NSCODE and also the modifications that have been made to support solid-solid overset intersections, as well as general improvements to the overset framework. It also covers 3D techniques that have been implemented inside CHAMPS for its overset suite.

2.1 2D - Flow Solver Architecture

One of the software used in the scope of this thesis is called NSCODE. It is a proprietary structured shared memory CFD flow solver solving the Reynolds Averaged Navier- Stokes (RANS) equations developed by the research group of Professor Éric Laurendeau at Polytechnique Montreal. It uses a finite volume method with a multiblock approach to simulate airflows. The 2D overset methods presented in this memoir are a part of a preprocessor module inside NSCODE which allows code modularity for usage with other methods such as the icing/droplet solver. It is written as a C shared library with OpenMP for local thread parallelization and complemented with a Python wrapper for ease of use when creating input files for simulations.

It uses the cell centered computational stencil as shown in Figure 2.1 based on the Jameson, Schmidt and Turkel (JST) artificial dissipation scheme [6]. In order for this stencil to work, each cell needs two neighbours. This poses a problem when encountering the edges of the mesh which is the reason why a double layer of halo cells is implemented. Halo cells are virtual cells that are on the fringes of the mesh and participate in the solving of the flow. They are also used for communication between the different mesh blocks.

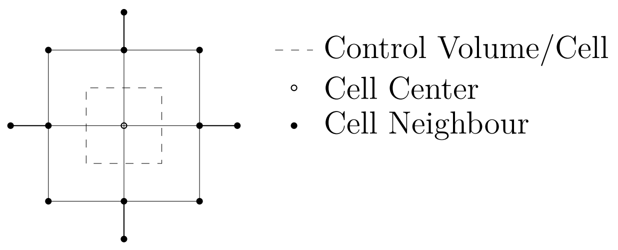


Figure 2.1 Computational stencil of NSCODE based on the JST scheme

2.1.1 2D - Boundary Conditions

Different types of boundary conditions are available to the user with NSCODE. Boundary conditions consist of a prescribed condition to define a particular physics problem. They are necessary to accurately model a simulation. Many boundary conditions are available to the user inside NSCODE. Here's a list of the most commonly used ones and of importance for the overset module:

- Connectivity (CON)
- Chimera (CHI)
- Farfield (FAR)
- Adiabatic wall (WAL)

A connectivity (CON) boundary condition is a necessary condition for information exchanges between mesh blocks. On a single mesh, this consists of placing the halo cells directly on the connecting block and exchanging data directly using these cells.

A chimera (CHI) boundary condition is essential in the overset module for use in solid-solid intersection. This boundary condition ensures that the halo cells on the fringes of the mesh are properly interpolated.

A farfield (FAR) boundary condition is usually implemented to ensure that the flow conditions at the ends of the mesh would be the same as an infinite domain. The current implementation of the far-field boundary condition inside NSCODE takes into account a vortex correction which assumes a vortex is generated at the center of the airfoil [2].

An adiabatic wall (WAL) boundary condition is where the velocity is tangent to the wall ($\vec{v} \cdot \vec{n} = 0$) and that the resulting heat transfer at the boundary is null ($Q = 0$).

An example of the use of the overset boundary condition (CHI) is seen in Figure 2.2 on a NACA0012 with a collar grid applied to the upper surface of the airfoil. It is possible to see the overset boundary conditions being applied to the fringes of the collar grid except for the share solid surface (WAL) with the mesh of the main airfoil.

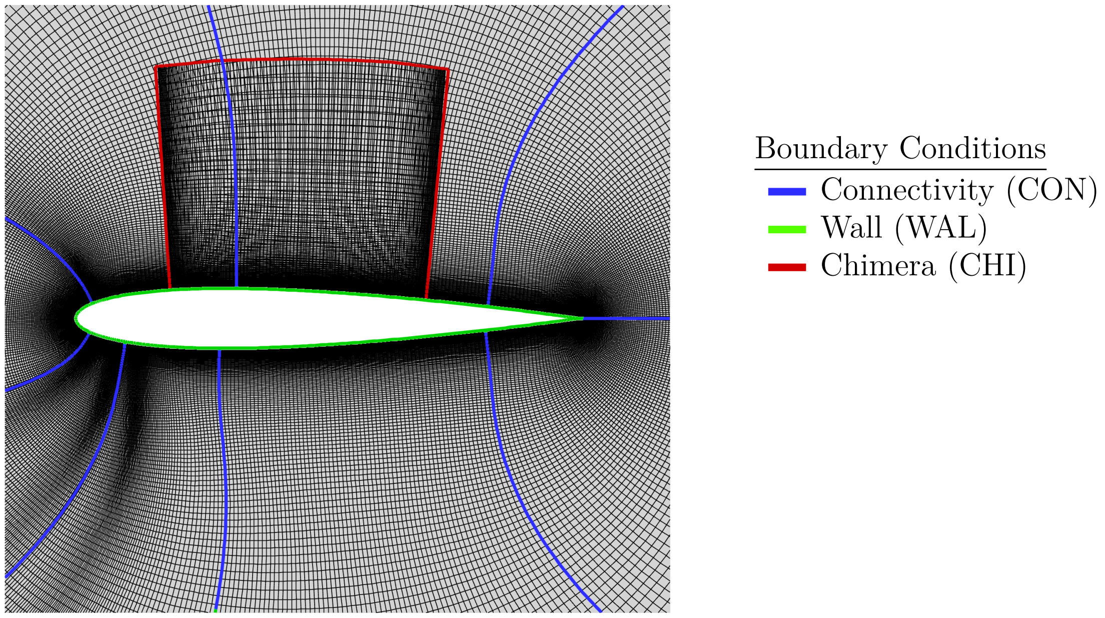


Figure 2.2 Example of boundary conditions on a NACA0012 with a collar grid applied to the upper section of the airfoil

2.1.2 2D - Overset Preprocessor

The overset preprocessor of NSCODE is launched in the first steps of the flow solver. It is comprised of the following steps:

1. Calculation of two different metrics: the wall distance of cells and the wall extrema of the geometry or geometries.

2. Creation of spatial boxing to group cells of overlapping meshes together and facilitate afterwards the donor search.

3. Creation of an internal mesh of the geometry using a Constrained Delaunay Triangulation (CDT) for further use with the hole cutting algorithm.

4. Hole cutting of the cells overlapping the geometries of the other meshes using the previously created CDT internal mesh.

5. Finding the cells overlapping each other between the various meshes. Then, depending on which cell is closer to the wall (wall distance criterion), defining which cell will be interpolated and which cells will act as donor cells to that interpolated cell.

6. Once donor cells have been identified, donor weight calculations for each of the interpolated cell with its respective donors take place.

7. Adding a buffer layer of cells to the fringe of the meshes to ensure that the mesh overlap is not containing any blanked cells.

2.2 3D - Flow Solver Architecture and Programming Language

The second software used for the 3D part of this memoir is called CHAMPS which stands for CHApel Multi-Physics Software [7]. It is a new code also developed by the research group of Professor Éric Laurendeau. It is written in the Chapel programming language and is a proprietary unstructured distributed memory 3D CFD flow solver. The overset module currently resides as a pre-processing tool inside the code. It uses a multiblock finite volume approach for solving the RANS equations.

Because of the nature of unstructured grids, a different approach to resolving fluxes must be taken to numerically solve the RANS equations when using an unstructured flow solver. The numerical scheme taken for CHAMPS is the Roe flux splitting scheme with facet reconstruction using the monotone upstream-centered schemes for conservation laws (MUSCL) approach with a Venkatakrishnan limiter taken from Blazek (2015) [2]. This approach only requires one layer of halo cells.

Advantages of Using the Chapel Programming Language

The Chapel programming language has several built-in features that allow developers to easily write a scalable and parallel code from the start without the use of external libraries such as OpenMP or Message Passing Interface (MPI). Another strong point of using this programming language is that information exchanges are much simpler to implement than a more traditional approach of a C/C++ code with an MPI library for use with distributed memory systems. With easier code maintainability and faster implementation, this allows the developer to spend less time on parallelizing the code and more time on running and testing out new features. This makes it a perfect candidate as a programming language in a research environment for writing a 3D CFD flow solver as it requires the use of distributed systems like HPC's in order to run large-scale simulations. As mentioned in Parenteau et al. (2020), the performance achieved by CHAMPS can rival the performance of other CFD codes such as SU2 which is written in C++ with MPI interfaces [7].

2.3 Overset Cell Classification

For the overset preprocessors of NSCODE and CHAMPS, the following terminology is used for the treatment of the various cells found within the meshes :

- Blanked/hole cut cells
- Buffer cells
- Computed cells
- Donor cells
- Interpolated cells

A blanked or hole cut cell is a cell that has been blanked out in the hole cutting process and was found to be inside the geometry of an overlapping mesh. This cell does not participate in the solving of the flow.

A buffer cell is a cell that is on the fringe of the overlap of two meshes. It is a computed cell and ensures that there are no holes when merging two meshes together.

A computed cell consists of a cell that is used directly for the solving of the flow.

A donor cell is also at the same time a computed cell and is used in conjunction with other donor cells with their respective interpolation weights for the interpolation of an interpolated cell.

An interpolated cell is a cell that its values are interpolated from donor cells. It also serves in the solving of the flow.

An example of the overset cell classification algorithm can be seen in Figure 2.3. This example is seen from the standpoint of the mesh of the main airfoil. It is possible to see the cells in a deep blue that are hole cut from its mesh because of the overlapping geometries of the flap and slat. It is also possible to see in yellow the double layer of buffer cells. The cells in green are the interpolated cells of the MDA main airfoil mesh and in turquoise its computed cells.

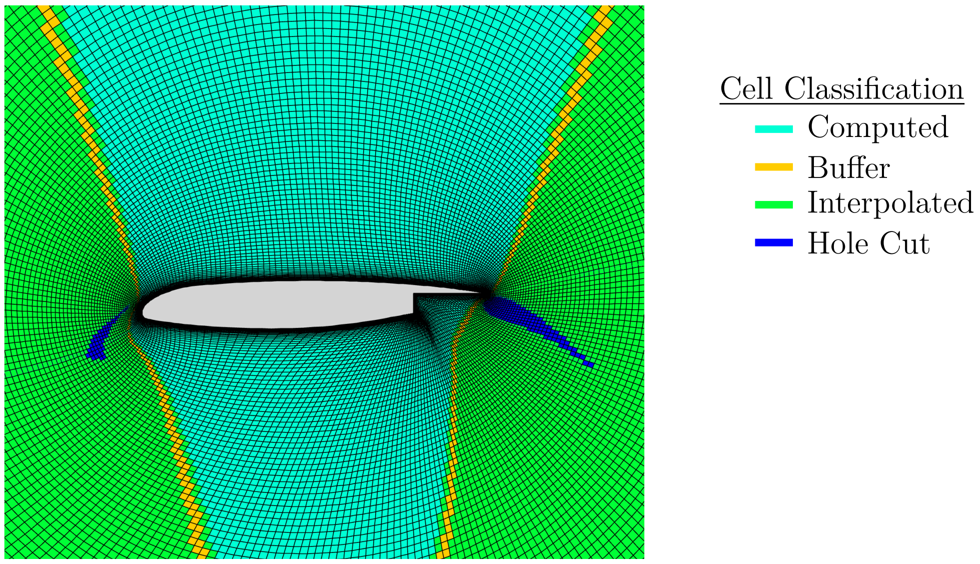


Figure 2.3 Overset cell classification example on the MDA airfoil

2D - Overset Cell Classification Algorithm

In Figure 2.4, an overview of the overset cell classification algorithm and its current implementation inside NSCODE is illustrated. This algorithm is done in step 5 of the overset preprocessor and is used for classifying interpolated and donor cells. A hierarchy criterion is present and is in place to differentiate meshes and their respective cells. This is done for explicitly telling the flow solver to classify cells with the lower hierarchy as interpolated cells. For example, when using a collar grid which is a grid with a much more refined resolution around a particular geometry, the cells of this grid have a higher hierarchy than cells from a background mesh, and thus will be the computed cells.

Finally, there is also the overset criterion which is based on either the cell volume or the wall distance depending on the user input. In the case of the wall distance, a cell with a smaller wall distance with the same hierarchy as the cell it is being compared against will be a donor cell and the other cell with the bigger wall distance will be interpolated. The same logic applies to the cell volume criterion. If a cell with a smaller volume with the same hierarchy as the cell it is being compared to will be the donor cell and the other cell with the larger volume will be the interpolated cell.

In most cases, the wall distance overset criterion has proven more robust than the cell volume criterion, and therefore it is the default method used in NSCODE unless explicitly specified

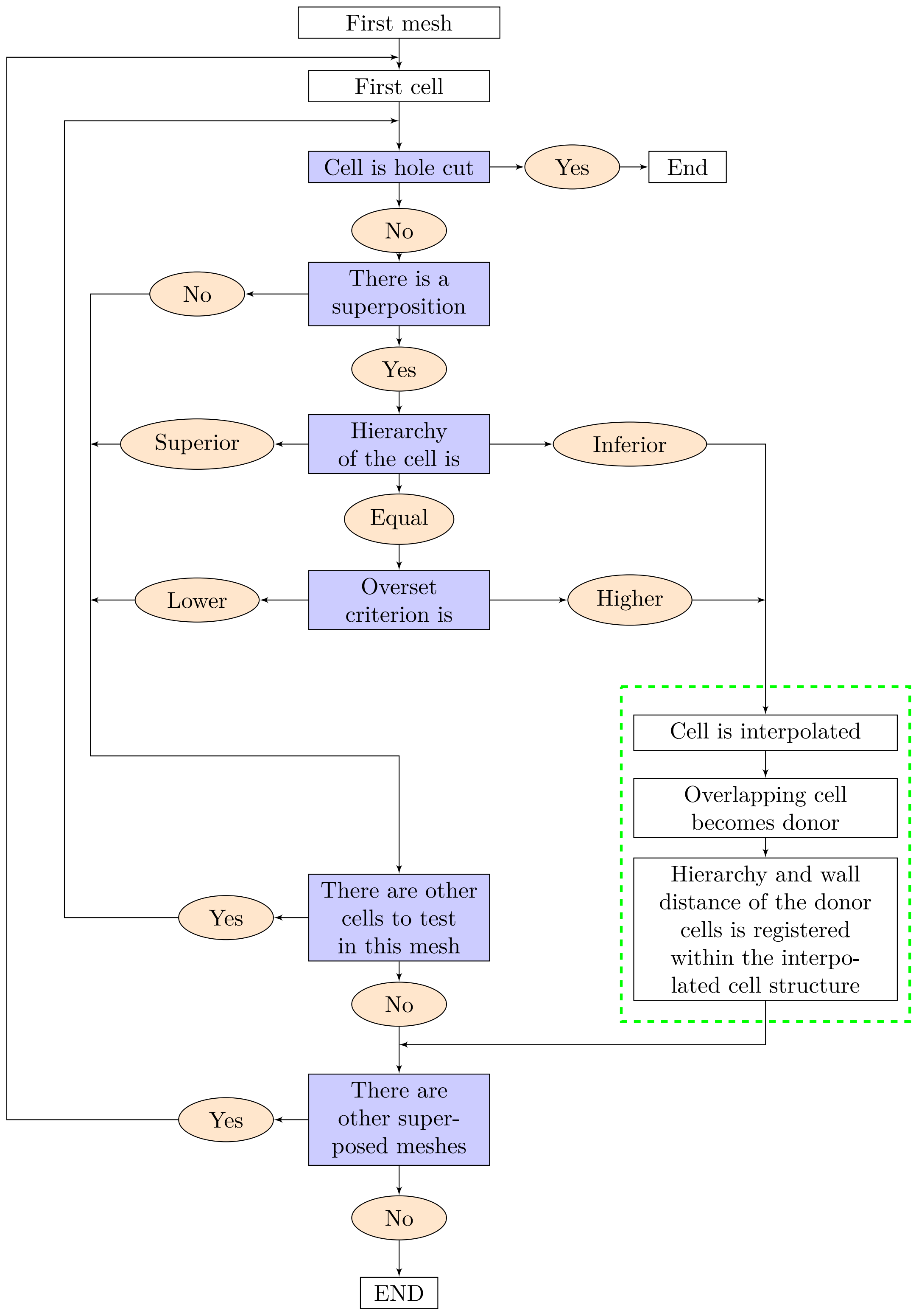

First mesh
First cell
Cell is hole cut
Yes
End
No
There is a superposition
No
Yes
Hierarchy of the cell is
Superior
Inferior
Equal
Overset criterion is
Lower
Higher
Cell is interpolated
Overlapping cell becomes donor
Hierarchy and wall distance of the donor cells is registered within the interpolated cell structure
There are other cells to test in this mesh
Yes
No
There are other superposed meshes
Yes
No
END

Figure 2.4 Overset cell classification algorithm implementation inside NSCODE from Guay (2017) [5]. Reproduced with permission

by the user [5].

Once all the cells have been classified, the weight calculations for each pair of interpolated cells and donor cells takes place.

2.4 Overlap and Donor Search

A critical part of any overset algorithm is finding the correct donor cells for an interpolated cell. This is sped up by using a spatial partitioning algorithm. This can be done in a variety of ways such as an: Alternating Digital Tree (ADT) or oct-tree algorithms in 3D and quad-trees in 2D. The main objective of the donor search is to find in which cell the cell from an overlapping mesh falls into and determining which cell is going to be interpolated and which cell will be the donor.

2.4.1 Alternating Digital Tree

One of the first methods that come to mind for spatially partitioning a mesh is the Alternating Digital Tree (ADT) method [8]. This method can be used in a variety of ways. For example, it could consist of taking the cells from each mesh and spatially partitioning them into a binary tree data structure defined by their bounding box [9]. In 2D, the variables used would be $x_{min}, x_{max}, y_{min}$ and y_{max}. Once the cells are stored in the binary tree structure, a point to test from the overlapping mesh (i.e. their cell center) can be taken and passed down through the tree.

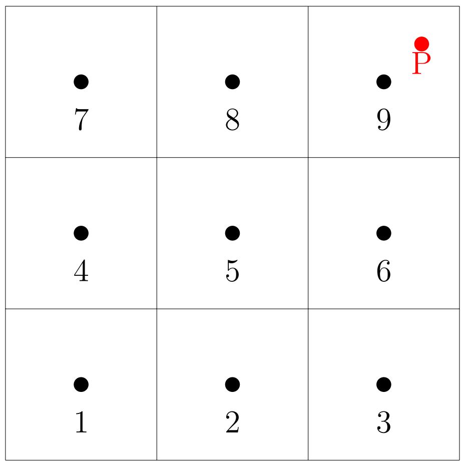


Figure 2.5 A Cartesian mesh with its cells numbered

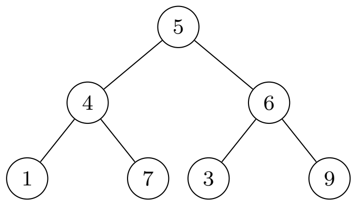


Figure 2.6 The corresponding binary tree for the Cartesian mesh

For example in Figure 2.5, a Cartesian mesh with its cells numbered from 1 to 9 is shown.

The point P highlighted in red is the cell center from an overlapping mesh that is used as a query point for the binary tree. In Figure 2.6, the corresponding binary search tree for the Cartesian mesh is schematized.

The method does have some drawbacks as noted by Roget & Sitaraman (2014) because of the computational cost of always starting from the root node for each query point and then descending through the tree [9]. Also, since it is a binary tree, to be effective the root node needs to be taken as one of the median cells in the mesh (i.e. located in the center of the mesh block) [9].

2.4.2 Quad-Tree and Oct-Tree Boxing

A common method to help find valid donor cells for interpolated cells and speed up the process is by the means of a boxing algorithm. In this type of algorithm, groups of cells from each mesh block are linked together as a way to ease in the donor search. In the case of NSCODE, a Recursive Virtual Cartesian Grid (RVCG) algorithm which resembles a quad-tree algorithm that was developed by Pigeon & Guay (2017) is used [5]. This algorithm imposes a virtual Cartesian grid on top of each mesh block. The cells from each block are then associated to their corresponding bounding boxes from the virtual Cartesian grid. A user input defines the number of cells that can be found within each of the boxes. If a certain box contains more cells than the allocated number from the user input, that box is then subdivided into 4 more boxes to separate the cells. This is done recursively until each box satisfies the user input. Once the boxing is done, a calculation of the boxes' extrema is made which defines the x_{min}, y_{min} and x_{max}, y_{max} of each box.

In Figure 2.7, the RVCG boxing done around the MDA airfoil and its slat and flap is shown highlighted in green. An interesting note about the recursiveness nature of the algorithm is that the finer boxing can be seen around the main airfoil and its flap and slat.

The same technique can be applied in 3D for CHAMPS using an oct-tree boxing technique which is essentially the same method as in 2D but extruded for the z dimension. An example of this is shown in Figure 2.8. In this particular case, an extruded NACA0012 airfoil is meshed around a spherical farfield in Pointwise. The mesh is then passed in CHAMPS with the overset preprocessor where the oct-tree boxing is applied. The end result of this process with the generated boxing is highlighted in green in Figure 2.8. It is possible to see the finer boxing around the airfoil at the center of the unstructured mesh.

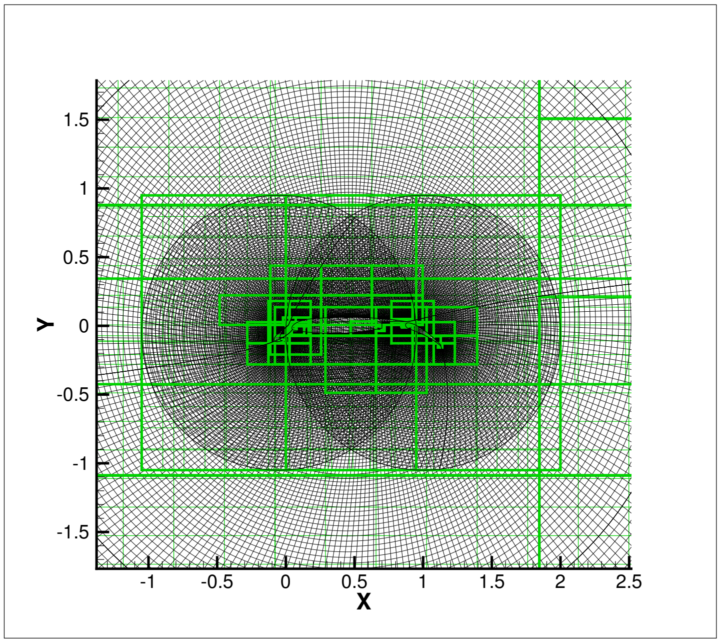


Figure 2.7 Example of the RVCG done with the MDA airfoil inside NSCODE

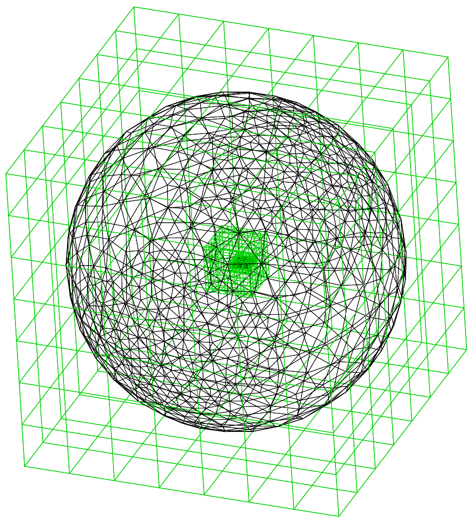


Figure 2.8 Example of the Oct-Tree RVCG boxing in CHAMPS on a NACA0012 unstructured mesh

Linking Donor Cells and Interpolated Cells in NSCODE in 2D

Once the boxing algorithm is done, the linking process between valid donor cells and interpolated cells is initiated. This consists of looping through each cell that needs to be interpolated and inside that loop, loop through each of the bounding boxes and find if the cell is inside any of the boxes. The cell center of an interpolated cell is tested against the extrema of the bounding boxes to see if it is found inside. This is done recursively until the finest boxing subdivision is reached. Then using the correctly found box, a loop through each of the cells contained inside the box is done. A triangle test is then performed. Cells found inside the box are each subdivided into two triangles and the cell center is then tested if it is found in any of the two triangles.

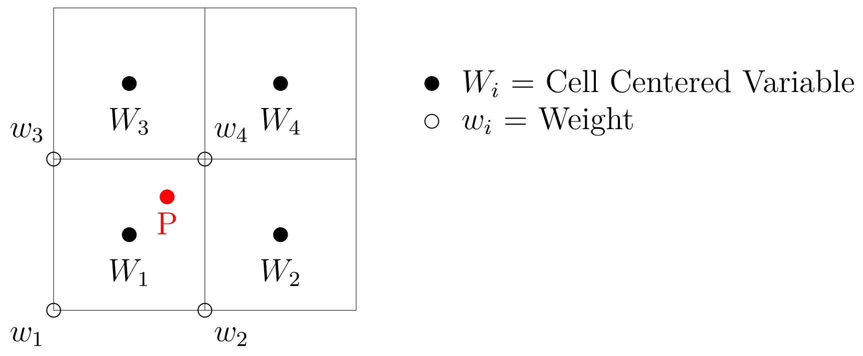


Figure 2.9 NSCODE donor cell stencil

Once a valid donor cell has been found for the interpolated cell, its donor list is defined by using the stencil found in Figure 2.9. The point in red "P" is the cell center of the interpolated cell that needs to be interpolated. The W_1, W_2, W_3, W_4 are the cell centers of the donor cells and also the location in which its flow variables are stored. The w_1, w_2, w_3, w_4 are the interpolation weights which are calculated using a bilinear interpolation algorithm. Other interpolation algorithms are also available to the user such as inverse distance interpolation, tetrahedral interpolation, and quadrilateral interpolation.

Linking Donor Cells and Interpolated Cells in CHAMPS in 3D

A similar approach for linking the donor cell to the interpolated cell is done in CHAMPS. The current element cell center is taken as a query point which is then used to loop through each of the 3D bounding boxes made from the oct-tree and find in which one it is contained.

Once the correct bounding box has been identified, finding the correct donor cell in 3D for CHAMPS is a bit trickier since it is more difficult to determine if a point is contained within a 3D element. A loop through each of the elements contained within the corresponding bounding box is necessary. For each cell, the scalar product between the vector comprised of the tested overlapping cell center and the query point is done against the vector of each of the element face surface normals. If the scalar product is negative for each of the element faces normal, this means the point is contained within the element.

Once the correct overlapping element has been found, a comparison of the current element wall distance against the overlapping element wall distance is made. The one with the smallest wall distance will be the donor cell and the one with the biggest will be the interpolated cell.

Having established the donor cell and interpolated cell pair, the interpolation stencil is based on a trilinear interpolation stencil that is built in from the CGNS standard [10].

2.5 Hole Cutting Methods

As mentioned in the introduction, one of the first steps to any overset algorithm is the hole cutting process. A hole cutting process essentially consists of blanking out cells from a mesh if they are part of the overlapping mesh geometry. Proper hole cutting of the geometries is essential because if the cells overlapping the geometry of another mesh are not blanked, the representation of the geometry will then be false and thus the simulation. Hole cutting methods can be classified into two subcategories: explicit and implicit.

Explicit hole cutting directly uses the geometry for determining if the cells overlapping the geometry need to be blanked. On the other hand, implicit hole cutting methods consist of blanking out the cells without relying directly on the geometry definition.

2.5.1 Explicit Hole Cutting Algorithms

CAD Based Hole Cutting

In Computer Aided Design (CAD) based hole cutting, it is possible to utilize the CAD surface definition as a basis for determining whether a cell is inside or outside the geometry.

This can be done in a variety of ways and is highly dependent on how the geometry was created. For example, at NASA, an extensive overset library named Chimera Grid Tools (CGT) was created for preprocessing and post-processing structured overset grids [3]. In the preprocessor module, this library is coupled directly with the CAD definition of the geometry to construct 3D structured meshes. It also directly perform hole cutting with X-rays using the CAD surface definition as a basis for the discretization of the geometry.

In the case of NSCODE, this is possible using the Engineering Geometry Aircraft Design System (EGADS) library [11]. This library utilizes the OpenCASCADE geometry kernel and interfaces with it as an API with several built-in functions [11]. Using the EGADS API, it allows testing if a coordinate is contained within the domain of the geometry or not. If the test fails, it means that the point is inside the geometry and the cell needs to be blanked.

2.5.2 Implicit Hole Cutting (IHC) Algorithms

Implicit hole cutting (IHC) is a form of hole cutting that doesn't rely directly on the geometry of the shape being hole cut. This can be done in a variety of methods with the following highlighted in the paragraphs below.

X-ray Hole Cutting

In X-ray hole cutting, it is possible to detect the geometry by passing through it a series of rays. It is a method commonly used in 3D simulations. It was first demonstrated by Meakin [12]. His method is comprised of projecting on a 2D Cartesian plane away from the object that needs to be hole cut a series of rays normal to the 2D plane. Afterwards, using the geometry discretization, it is possible to determine entry and exit points for each of the rays. Once the entry and exit points are known, testing for arbitrary coordinates such as cell centers of an overlapping grid can be done to see if they're found inside or outside the geometry by checking the distance from the entry and exit points of the rays. The computational cost associated with this method is directly linked to the number of rays used. Refinements over the original algorithm have been made in recent years to automate the method as its original algorithm is prone to user errors [13]. It is interesting to note that X-ray hole cutting can also be used in conjunction with other methods such as Cartesian hole maps to provide a more robust hole cutting [14].

Inside/Outside Tests Hole Cutting

Inside/outside tests as a form of hole cutting was first demonstrated by Baeder & Lee (2002) [15]. This method is implemented by performing along the wall surface boundary of the geometry a cross-product test between an arbitrary point such as a cell center and the nearest edge of the geometry. If the resulting sign of the cross-product is negative, it means that the point tested is found inside the geometry and then needs to be blanked. On the contrary, if the cross-product result is positive then the point is found outside of the geometry.

Cartesian Hole Maps

Cartesian hole maps are a hole cutting method in which a Cartesian grid is superposed on top of mesh [16]. If valid donor cells are not found for a given interpolation point within a specified distance/spectrum, that interpolation point is categorized as blanked.

NSCODE Hole Cutting Algorithm

The current hole cutting algorithm that is present inside NSCODE mainly consists of a test to see if the particular cell center of a cell is contained within a Constrained Delaunay Triangulation (CDT) of the geometry. First, the wall bounding boxes of an overlapping mesh are calculated. Then a simple test is made to see if the tested cell center is contained within the bounding box extrema. If it is, the cell center is then tested to see if it is contained within the triangles which make up the particular overlapping geometry. If the cell center is then also contained within the particular geometry, it is then blanked.

This method was successfully tested in the past inside NSCODE [5]. In Figure 2.10, an example of the internal mesh triangulation done via CDT on the MDA airfoil is shown.

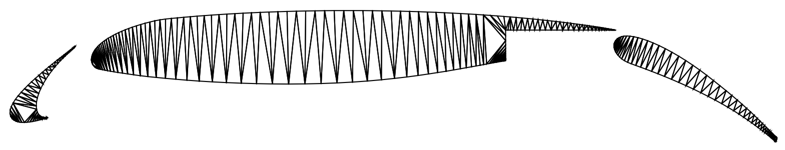


Figure 2.10 Example of the MDA airfoil internal mesh done via CDT

2.6 Mesh Generation

The mesh generation process is a very important step before any overset preprocessor operations occur. Usually, mesh generation entails the solving of elliptic, parabolic or hyperbolic partial differential equations (PDE) systems which results in the creation of computational grids. Within NSCODE, there is an in-house developed structured mesher called NSGRID [17]. NSGRID supports both algebraic and hyperbolic mesh generation.

2.6.1 Hyperbolic Mesh Generation

One of the methods available in NSGRID is the generation of meshes with the hyperbolic method. This method of meshing was first described in Steger & Chaussey (1980) [18]. This method in 2D consists of solving two PDE's which represent a hyperbolic system. The two equations that are solved are the following :

$$X_\xi X_\eta + Y_\xi Y_\eta = 0 \tag{2.1}$$

$$X_\xi Y_\eta + X_\eta Y_\xi = 1/J \tag{2.2}$$

Equation (2.1) consists of the orthogonality condition and equation (2.2) consists of the inverse Jacobian condition for transforming the x, y coordinate system into a adimensionalised ξ, η referential [19]. Coupled together these equations represent a hyperbolic system of PDE [19]. This system can be resolved by marching in the η direction [19].

The grids produced by this method present high quality metrics for general CFD use [18].

2.6.2 Collar Grids

Collar grids are refined meshes with a higher degree of discretization that share a solid surface of a geometry. These grids can be generated in a variety of ways using algebraic and hyperbolic meshers. In the case of NSGRID, collar grid generation is directly linked to the hyperbolic mesh generator. By inputting a particular geometry with a set of coordinates, a collar grid can be generated easily using the EGADS geometry kernel in conjunction with NSGRID. This is done by solving the PDE equations described in Section 2.6.1.

However, to ensure a good quality collar grid, it is imperative to make certain adjustments to the inputs given to the mesher and enforce certain boundary conditions.

In terms of inputs, we need to ensure that sufficient discretization between the points is achieved. This is done by specifying an adequate amount of faces in the i direction. Another input that is necessary for creating good quality collar grids is that the surface normal present at both ends of the collar grid needs to be preserved. This can be done by using a constant Cartesian plane boundary condition [20]. This boundary condition is fed implicitly to the hyperbolic mesh generator PDE system.

Figure 2.11 illustrates a collar grid with a forced vertical direction. This is the default method in NSGRID for generating collar grids.

In Figure 2.12, this is a collar grid with a preserved surface normal direction. This method presents better grid quality especially with cells at each of the i direction boundaries (i_{min} and i_{max}).

As mentioned previously, this method is possible by implementing a constant Cartesian plane

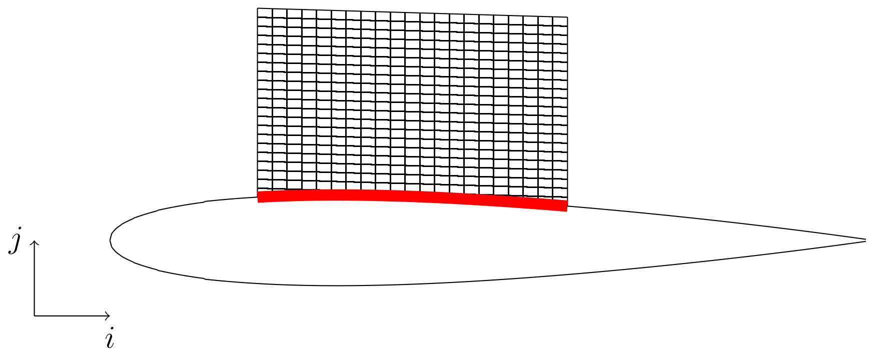


Figure 2.11 Schematization of a NACA0012 collar grid with a forced vertical direction

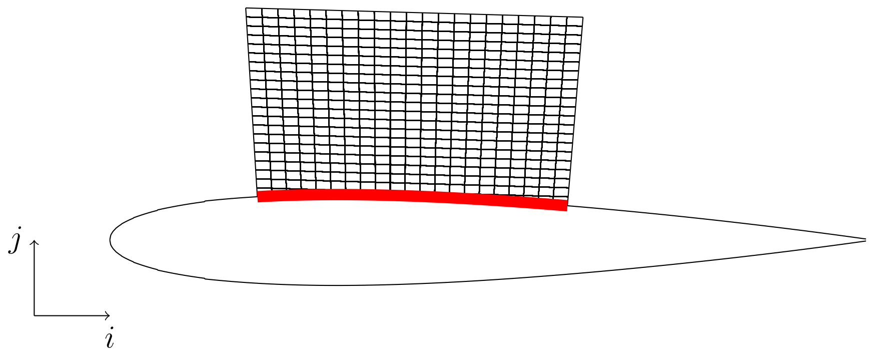


Figure 2.12 Schematization of a NACA0012 collar grid with surface normal direction imposed

boundary condition. In a 2D case, this means that ξ and η are restricted to constant planes in x and y defined by the surface normal at the i_{min} and i_{max} boundaries of the collar grid. This condition is described by Equation (2.3) [20].

$$\begin{pmatrix} \Delta x \\ \Delta y \end{pmatrix}_{j=1} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}_{j=2} \tag{2.3}$$

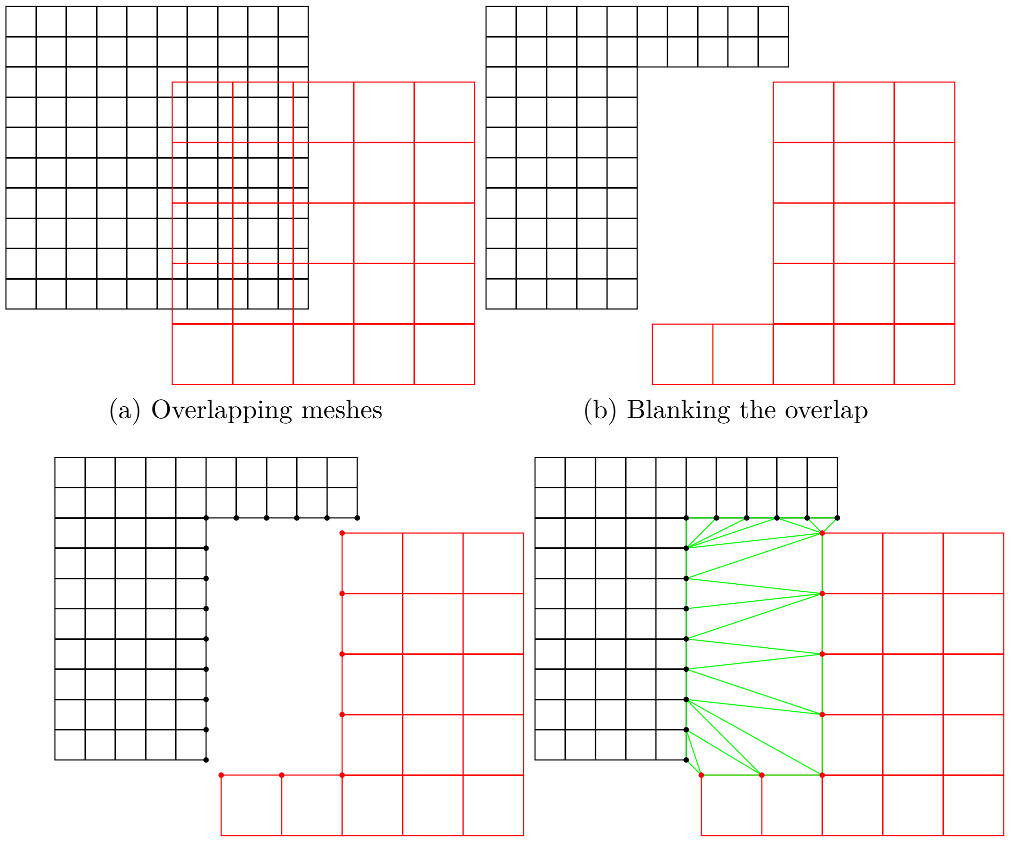
2.6.3 Zipper Grids

Zipper grids are a type of hybrid grid that is created by removing and filling the overlap with triangular elements between two meshes that are overlapping each other. The reason behind this method is to ease the use of merging several overlapping meshes in an automatic fashion. This method is particularly interesting in cases of solid-solid intersections such as the intersection of an aircraft wing and the aircraft fuselage. It could also be used in instances where a collar grid is applied to a background mesh and then is automatically merged by the zipper grid method with the said background mesh.

The roots of the method were instigated at NASA-Ames by Chan & Buning (1995) [21]. The original method consisted of 4 main steps enumerated below [21]:

1. Identify cells in the overlap regions and blank them out as seen in Figure 2.13a.
2. Cells on the fringe of the blanked overlap regions are identified and then ordered as seen in Figure 2.13b.
3. Using a triangulation algorithm and the identified cells in the fringe regions of the previous step, triangular cells are filled in the overlap region as seen in Figure 2.13c.
4. A final check is performed to ensure that the created elements from the triangulation are valid (no negative volume elements) as seen in Figure 2.13d.

The finalized result from this method is a structured mesh with quadrilateral and triangular elements. In Figure 2.13, the 4 main steps of the algorithm in action are schematized.



(a) Overlapping meshes

(b) Blanking the overlap

(c) Identifying and tagging the fringes

(d) Triangulation

Figure 2.13 Zipper grid algorithm illustrated steps

This method poses some challenges when using a structured flow solver. This difficulty is highlighted by the use of the newly created triangular elements. Since a structured flow solver exclusively uses quad elements, new procedures and methods need to be implemented for the treatment of this new triangular element region in the mesh domain. In the literature, this can be done in 2 distinct ways. The first one involves different algorithms to be used when treating the triangular region such as different algorithms for the integration of the surface forces and fluxes calculations [21]. The second one consists of writing out an entire unstructured flow solver routine inside the structured flow solver [21]. Another difficulty highlighted later on by Chan (2009) is that the fidelity of the algorithm greatly diminishes when using grids with different resolutions [1].

2.6.4 Schwarz Correction

When using collar grids with a solid-solid intersection, a correction to the interpolation weights must be made because of the finer discretization of the geometry surface seen by the collar grid. This correction was first demonstrated by Schwarz (2005) in his article on rotorcraft flows [22]. This implicates creating a virtual interpolation point in which the physical coordinates of the point in question are temporarily modified for their use in the calculation of the donor cells weights [22].

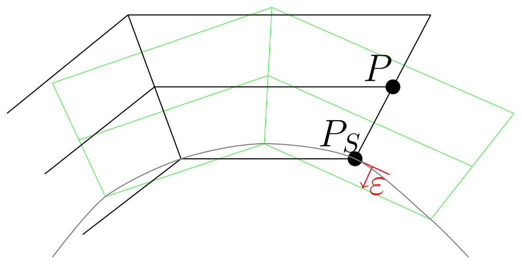


Figure 2.14 Schwarz correction illustrated for cell vertices. Reproduced from Schwarz (2005) [22]

In Figure 2.14, a collar grid with a finer discretization of the geometry is highlighted in green. The geometry is illustrated by the gray curve. The grid in black is the grid overlapping the collar grid which needs to be interpolated. This grid presents a coarser discretization of the geometry. The points P and P_S belong to the interpolated grid. For illustration purposes, the point P is the point that needs to be interpolated and the point P_S is the closest point to the surface from the point P. Finally, ε is the projection vector from the point P_S onto the closest collar grid surface.

$$\vec{x}_{P^*} = w \cdot \vec{\varepsilon} + \vec{x}_P \tag{2.4}$$

$$w = \begin{cases} 1 & 0 \le |\vec{x}_P - \vec{x}_{P*}| < d_1 \\ \frac{d_2 - |\vec{x}_P - \vec{x}_{P*}|}{d_2 - d_1} & d_1 \le |\vec{x}_P - \vec{x}_{P*}| < d_2 \\ 0 & d_2 \le |\vec{x}_P - \vec{x}_{P*}| \end{cases} \tag{2.5}$$

$$d_1 = 10 \cdot |\vec{\varepsilon}|,\ d_2 = 30 \cdot |\vec{\varepsilon}|$$

Equation (2.4) and Equation (2.5) are taken directly from Schwarz (2005) [22].

The virtual interpolation point is calculated via Equation (2.4) where $\vec{x}_{P^*}$ are the virtual interpolation point coordinates. The interpolation weight w is calculated using Equation (2.5). Finally, $\vec{\varepsilon}$ is the projection vector which is calculated by taking the nearest point on the surface of the coarse mesh and calculating the distance between it and the nearest corresponding surface on the collar grid. In Equation (2.5), d_1 and d_2 are used as weighting coefficients to diminish the correction the further the interpolation point is from the surface.

In Equation (2.5), the weight w value is calculated depending on whether the intermediate result $|\vec{x}_P - \vec{x}_{P*}|$ falls between which region of the domain is determined by the function by parts. d_1 and d_2 are weighting coefficients that determine the domain in which $|\vec{x}_P - \vec{x}_{P*}|$ falls into.

Once the virtual interpolation point coordinates have been calculated, it is then used in the interpolation stencil for calculating the donor weights in relation to that new virtual interpolation point.

2.7 Open Source Overset Libraries

It is good to know that open source libraries for overset exist. However, since overset algorithm implementations are highly dependent on the CFD flow solver architecture, it is very difficult to find one that matches a particular CFD software architecture without extensively modifying the interfaces between the CFD code and the external overset library.

Topology Independent Overset Grid Assembler (TIOGA) is an overset assembler library written in C++ developed by Jay Sitaraman [23]. It is used as an external assembler for merging both structured or unstructured meshes. It has implicit hole cutting as well as an alternating digital tree (ADT) for donor search.

The OpenFOAM overset module is another example of an open-source overset project. It can create and assemble structured and unstructured meshes for overset simulations [24]. It is also written in C++. As far as its capabilities go, it uses a quad-tree with bounding boxes to tag cells in the overlap for donor search, inverse distance interpolation, and hole maps for hole cutting.

There is also the Cassiopee open source python module library developed by a research group at ONERA [25]. Cassiopee has a variety of features as a preprocessor and post-processing tool for mesh manipulation. It does feature an overset suite, but as of this writing, it is proprietary to the research group.

CHAPTER 3 2D OVERSET FRAMEWORK IMPROVEMENTS

This chapter covers the development that has been done inside NSCODE concerning solid-solid intersections when using the overset module. Several improvements to the overset methods have been implemented.

3.1 Interpolation Changes

3.1.1 Previous Interpolation Stencil

In terms of interpolation methods, several exist. The current one that is used is comprised of the following stencil as shown in Figure 3.1.

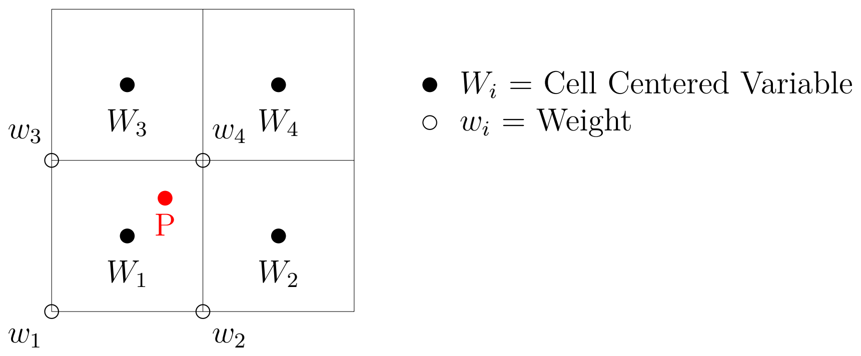


Figure 3.1 Previous 4-point bilinear interpolation stencil

The weights (w_1, w_2, w_3, w_4) are calculated using bilinear interpolation and then stored at the vertices of the cell in question. The variables (W_1, W_2, W_3, W_4) used in correlation to these weights are stored at the cell centers. The point P in red is the cell center of an underlying mesh that needs to be interpolated. The interpolation stencil shown in Figure 3.1 has a main pitfall. There is an offset error between where the interpolation weights (w_1, w_2, w_3, w_4) are calculated in the vertices as opposed to where the primitive values (W_1, W_2, W_3, W_4) for interpolation are located. Normally, the calculated weights should be calculated at the same position where the calculated variables are used for the interpolation. In this case, since the calculated variables are at the cell centers, the calculated weights need to be calculated using the cell centers coordinates for the bilinear interpolation. The bilinear interpolation algorithm of NSCODE is provided in Annex A.

3.1.2 New Interpolation Stencils

To avoid the errors induced by the use of the previous interpolation stencil, three new stencils were devised.

- A 9-point stencil with inverse distance interpolation
- A 9-point stencil with bilinear interpolation
- A 4-point stencil with bilinear interpolation

The first one is shown in Figure 3.2. This 9-point stencil stores the weights directly at their appropriate locations in the cell centers in order to no longer have an offset error. The nine weights are calculated using an inverse distance interpolation in which the weights are inversely proportional to the square of their distance to the interpolation point. For further reference on how the weights for the inverse distance interpolation stencil are calculated, please refer to Annex A. The second one is a variant of the first one presented, but instead

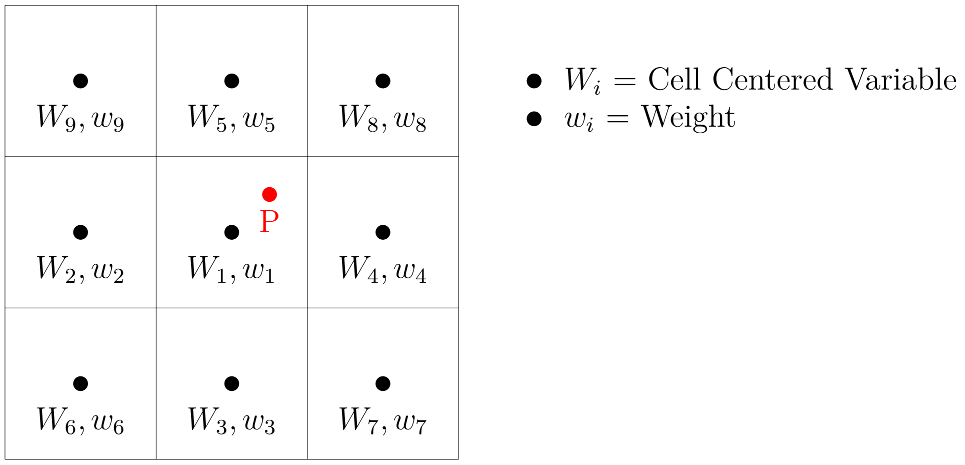


Figure 3.2 9-point inverse distance interpolation stencil

of using inverse distance interpolation, it uses a bilinear interpolation algorithm as shown in Figure 3.3. The values at the nodes W_i^* are calculated from the surrounding cells as described in Equation (3.1) to (3.4).

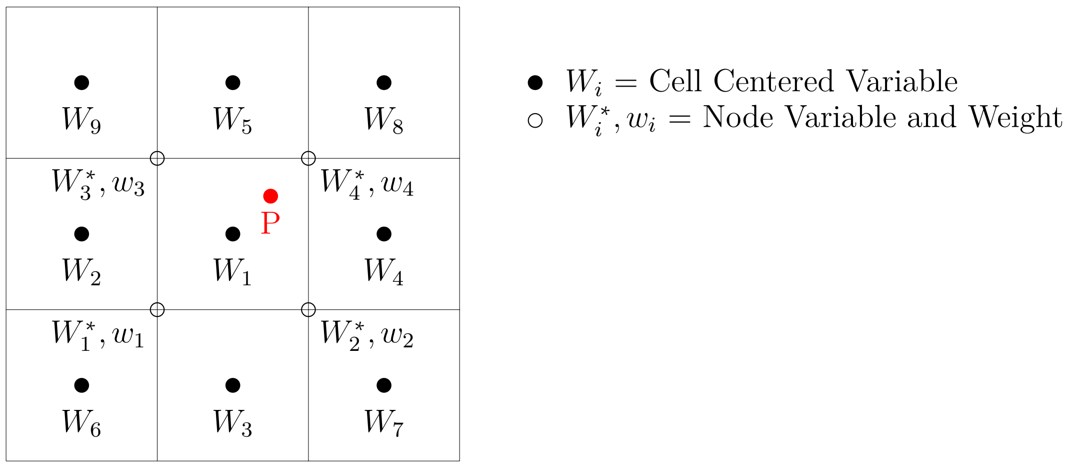


Figure 3.3 9-point bilinear interpolation stencil

$$W_1^* = 0.25(W_1 + W_2 + W_3 + W_6) \tag{3.1}$$
$$W_2^* = 0.25(W_1 + W_4 + W_3 + W_7) \tag{3.2}$$
$$W_3^* = 0.25(W_1 + W_2 + W_5 + W_9) \tag{3.3}$$
$$W_4^* = 0.25(W_1 + W_4 + W_5 + W_6) \tag{3.4}$$

The final variant of the interpolation scheme tested is the bilinear interpolation with a 4-point stencil with its weights calculated from the cell centered positions as shown in Figure 3.4.

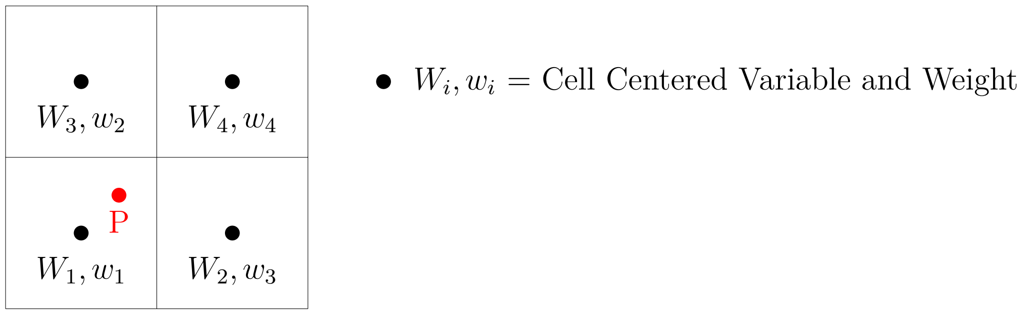


Figure 3.4 New 4-point bilinear interpolation stencil

3.1.3 Numerical Test Case

To test out the various stencils, a test case inspired by the methodology and work done by Pigeon (2015) to test out the interpolation stencil of the overset suite in NSCODE was devised [26]. This test case consists of a background Cartesian mesh with a patch mesh overlaid on top of it at the center as seen in Figure 3.5. In this case, the patch mesh is highlighted in green. This patch mesh computes data in its cells based on the chosen analytical function described in Equation (3.5). This analytical function was chosen because it presents a high rate of variability of computed values for an x and y combination for the chosen interpolation domain. To ensure that the same order of convergence of $p = 2$ as the previous 4-point bilinear interpolation stencil is maintained, for each of the tested stencils, their convergence order is computed and compared to the previous stencil. The interpolation error for each of the stencils can be calculated easily when comparing the value given by the interpolation at each of the interpolated cells of the background mesh against the analytical value calculated from Equation (3.5). This error is calculated by summing the offset error between the analytical value and the interpolation result for each of the cells as seen in Equation (3.6).

$$f(x,y) = \cos(\frac{\pi x}{10})\sin(\frac{\pi y}{10}) \tag{3.5}$$

$$\text{Interpolation Error} = \sum^{\#\text{ of cells}} |\text{Interpolation Result} - \text{Calculated Analytical Result}| \tag{3.6}$$

In order to test this, the patch mesh is progressively refined 3 times by a factor of 2 each time from its previous state. The characteristics of each of the refined meshes are described in Table 3.1.

Table 3.1 Patch mesh characteristics for each of its refinement levels

Refinement level #	**Number of faces**	**Number of cells**
1	51x51	2500
2	101x101	10 000
3	201x201	40 000
4	401x401	160 000

In Figure 3.6, the results of the 2D interpolation test case are presented. The slope from each of the linear regressions having the form of Equation (3.7) corresponds to the order of convergence.

$$\log(\frac{\text{Error}}{\#\text{ of cells}}) = p\log(\frac{1}{\#\text{ of cells}}) + b \quad \text{where } p \text{ is the order of convergence} \tag{3.7}$$

In Table 3.2, the order of convergence p and the coefficient of determination R^2 are presented for each of the stencils. The best one is found to be the new bilinear 4-point interpolation

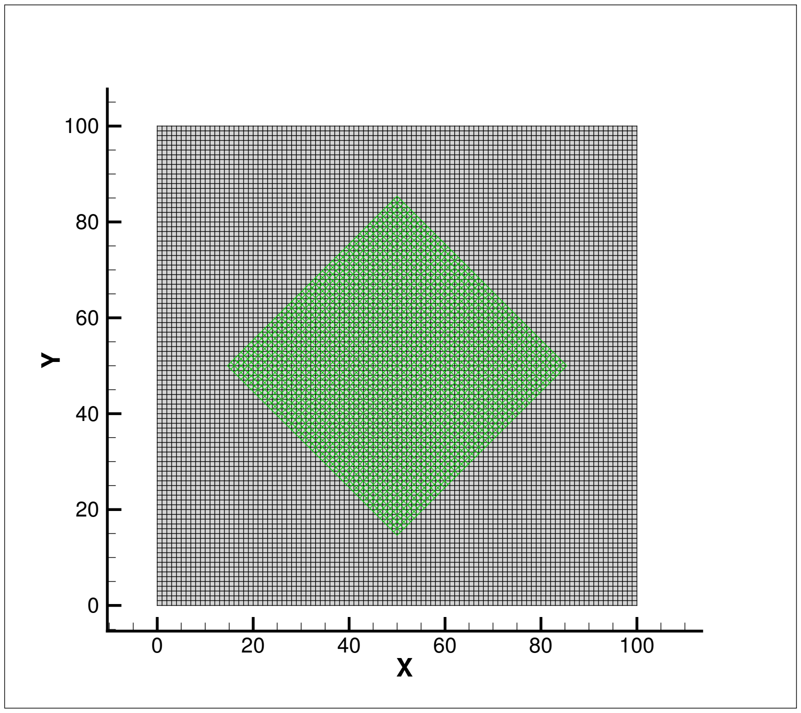


Figure 3.5 2D interpolation test case meshes with the 51x51 patch mesh highlighted in green

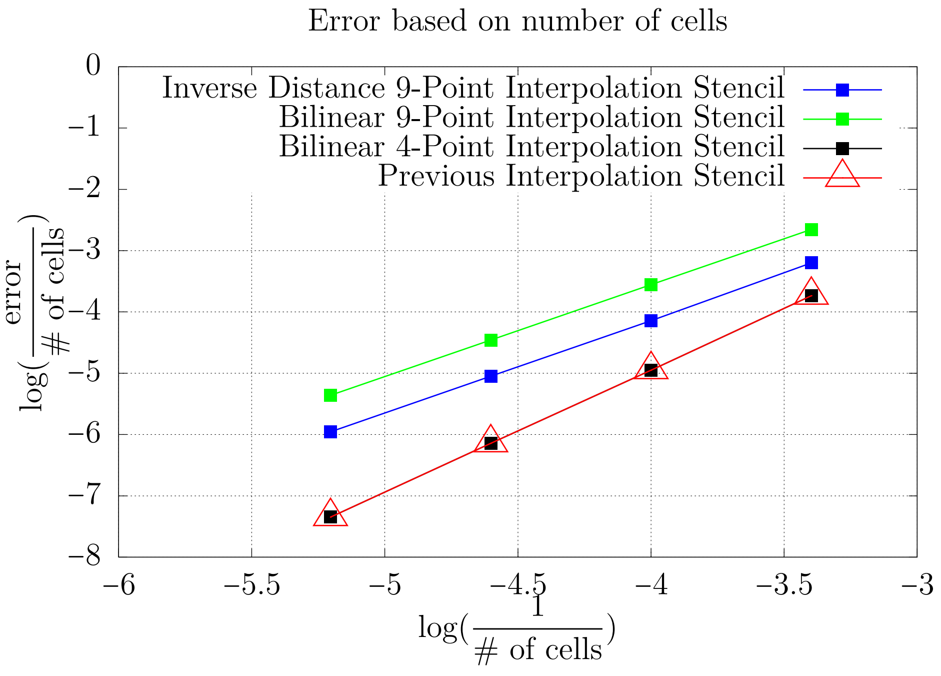


Figure 3.6 Interpolation results based on the interpolation stencil chosen

stencil. As seen in Figure 3.6, the two 9-point stencils present stronger interpolation errors for each level of the mesh refinements when compared to the old and new 4-point bilinear interpolation algorithms as well as a lower order of convergence. The reason why the data for the new 4-point bilinear stencil is identical to the previous one is because of the nature of the Cartesian grid used. Since it is perfectly orthogonal with exactly the same spacing between each of the cells, the interpolation weights calculated for each of the cells is identical to the previous interpolation stencil. However, this is not a problem since the test case was designed to evaluate the order of convergence and the accuracy of the interpolation methods by the means of evaluating the interpolation error from the chosen stencil. By testing different stencils, it ensures that the new chosen method isn't worse than the previous method implemented. Now, by having the weights stored at the same position as the values used for interpolation, it is ensured that no interpolation error will come from an offset of the weights.

Table 3.2 2D interpolation stencils results

Numerical Stencil	p	R^2
Inverse distance 9-point	1.5226	0.99
Bilinear 9-point	1.4991	1
New bilinear 4-point	1.9966	1
Previous bilinear 4-point	1.9966	1

3.2 Collar Grid Generation

As mentioned by Guay (2017), collar grid generation was added to NSGRID [5]. However, a common issue with the method was the forced vertical plane direction that was present at both extremities at i_{min} and i_{max} locations on the mesh. This notably produced skewed cells of poor quality on surfaces with high curvature. Also, another issue with the method was the ease of use of the algorithm which was difficult for inexperienced users of NSGRID.

3.2.1 Upgrades Made to the Collar Grid Generation

To rectify these two issues, upgrades were made to NSGRID to allow the generation of better quality meshes and also to decrease the complexity of the user input for the generation of these meshes.

Constant Cartesian Plane Boundary Condition

To generate better quality collar grids, it is necessary to preserve the boundary surface normals at the ends of the mesh. This can be done by implementing a constant cartesian plane boundary condition by implicitly modifying the equations that are solved in the hyperbolic mesh algorithm inside NSCODE [20]. This is done by calculating the surface normals of the geometry at the i_{min}, j_{min} and i_{max}, j_{min} boundaries of the mesh. In this case, the j_{min} boundary corresponds to the wall boundary. In order to accomplish this, it is possible to get from the EGADS geometry kernel the local surface derivatives dx, dy and from these derivatives calculate the unit normal vector of the surface.

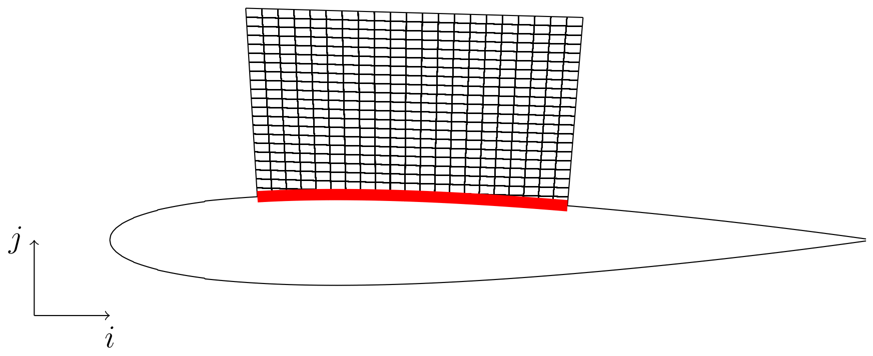


Figure 3.7 Example of a NACA0012 collar grid with surface normals

Once the unit normal vector is calculated, it is fed implicitly as a boundary condition to the matrix system being solved to generate the grid. An example of this algorithm in action is in Figure 3.8 where a collar grid for a NACA0012 airfoil upper surface has been generated. This collar grid will be used in the second numerical test case with the laminar NACA0012.

3.3 Solid-Solid Discontinuities

Another problem that was noted in Guay (2017) was that a discontinuity problem arises when a patch/collar grid mesh shares the same solid boundary [5]. This produces discontinuities exactly at the junction of the patch mesh and the mesh containing the geometry in question. The discontinuities are seen in the coefficient of friction magnitude and components (C_f, C_{f_x}, C_{f_y}, C_{f_z}) and the pressure coefficient magnitude and components (C_p, C_{P_x}, C_{P_y}, C_{P_z}).

An example of the discontinuities is shown in Figure 3.9 for a laminar cylinder. This test case is referred to later on in the numerical test cases in section 3.3.5.

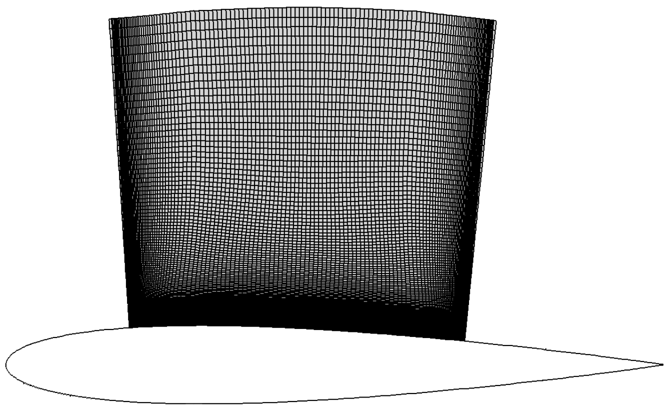


Figure 3.8 A NACA0012 collar grid generated on the upper surface of the airfoil with the new normal collar grid generation technique from NSGRID

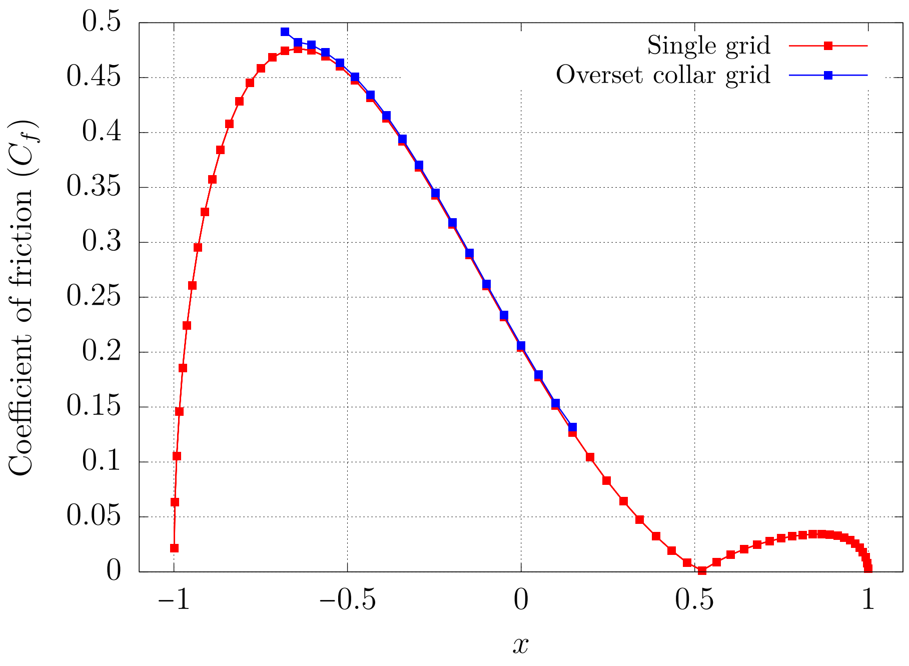


Figure 3.9 An example of discontinuities happening on the laminar cylinder test case for the coefficient of friction C_f

3.3.1 Root of the Problem

As mentioned previously, the problem only arises at the junctions of the collar grid mesh with the background mesh of the lower hierarchy whose cells are interpolated. A possible explanation highlighted in Guay (2017) is that the mesh halos are to blame [5]. As a matter of fact, halo cells are not only used for the interpolation of corner mesh cells, but are also used for calculating local dx, dy derivatives. Before going into the details on how these derivatives are used to calculate gradients and other derived values such as the C_f and C_P, let's take a look at how these halo cells are generated inside NSCODE.

Original Halo Cell Generation

Originally, the halo cells inside NSCODE were generated physically on top of each other by a linear extrapolation of the last two cells of the i_{min}, j_{min} and i_{max}, j_{max} boundaries. Since NSCODE uses a double layer of halo cells for its computational stencil, but only needs physically the location of the first layer, the memory allocation is done in such a way that there is only memory space allocated for one physical layer of halo cells to save on memory costs. However, without using the overset module and only using a single mesh with connectivity (CON) boundary conditions, the discontinuity problem never arose at the interconnection of blocks because the CON boundary condition ensures that the halo cells of a mesh are at the right physical location on the receiving side of a connecting block. In that case, the physical locations of the halos cells that were previously calculated using a linear extrapolation are overwritten with their correct physical locations by the means of the CON BC.

However, in the case of the overset module, when using a collar grid mesh with a background mesh, there is no connectivity boundary condition. This poses a problem because, without the correct physical location of the halo cells, the derivatives dx, dy dependent on the physical location of a cell are false. In turn, this falsifies the calculation of the gradient which then falsifies derived values from the said gradient.

3.3.2 Solution to the Discontinuities

A temporary fix to the problem was set up by Guay (2017) in which halos cells were directly specified by the users [5]. Typically, cells from the mesh extremities were taken to create the halo cells at their appropriate locations.

There are several downsides from using this method. The first one is that when generating the collar grid it needs to be made bigger with in mind that 2 layers of cells on all sides of the mesh will be taken as halo cells. The second downside is that this has to be done entirely manually by the user and thus is prone to user errors.

In order to fix this problem permanently in an automated fashion, two fixes to NSCODE must be implemented:

- Recreation of the mesh boundaries at the halo cells locations
- Memory upgrade to store physically the double layer of halo cells

The first item pertains to getting the halo cells to their actual physical locations. This is done by recreating how the mesh was originally generated.

The second item on the list is the memory enlargement that is necessary for NSCODE to store physically the second layer of halo cells.

For purposes of clarity, the fixes are highlighted on a cylinder mesh with a collar grid that is identical to its background mesh. The reason for this is that the discontinuity problem is highly apparent on surfaces with high curvature such as a cylinder especially when linearly extrapolating the halos. Also, the results can be easily compared to the case without the collar grid and overset preprocessor treatment applied. It should present the same results if the overset treatment is done properly, since the cells are exactly the same as the background grid. The cylinder with its collar grid highlighted in green is seen in Figure 3.10. It is possible to see the original method in which halo cells were linearly interpolated in Figure 3.11a. In Figure 3.11b, it is possible to see the normal projection fix that is applied to the halo cells.

Mesh Recreation

The first step to fixing the problem is to recreate the boundaries of the collar grid mesh. The general idea behind this is to position the halo cells at their correct location in such a fashion that it resembles how they would have been generated using the hyperbolic mesh generator. This is made possible by using the EGADS geometry kernel.

First, a linear extrapolation of the first cell is done. Using the EGADS API, the point closest to the newly created temporary cell vertices is found on the B-Spline curve representing the geometry of the collar grid. From this point on the curve, the surface normal is calculated using EGADS and the calculated derivatives (dx, dy).

The derivatives dx and dy can be taken as the tangent vector to our point on the curve of the geometry. By performing a simple geometric transformation as shown in Equation (3.8), the unit normal vector can be calculated.

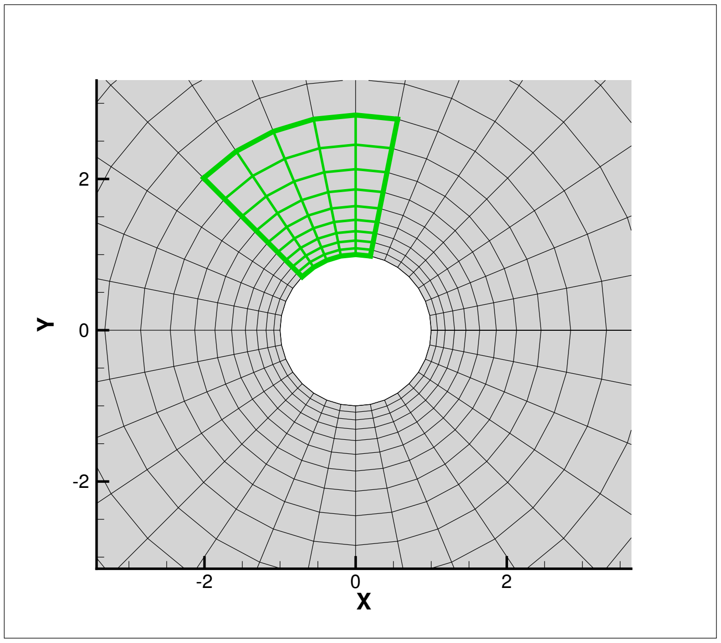
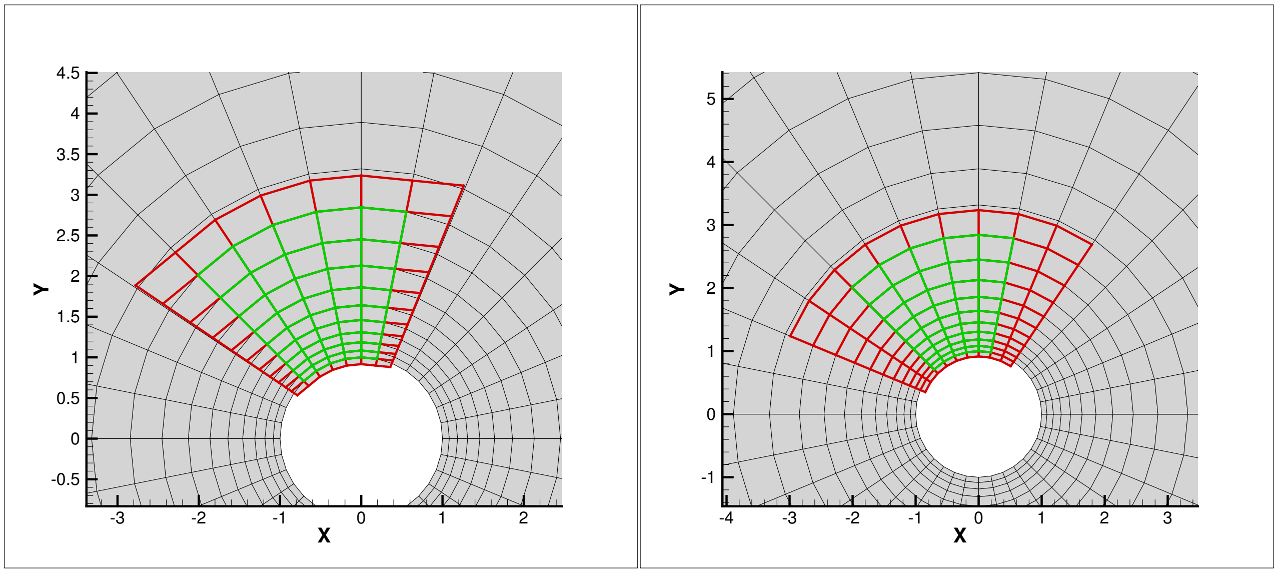


Figure 3.10 Cylinder mesh with its collar grid identical to its background mesh highlighted in green



(a) Linear extrapolation of the halo cells

(b) Normal projection of the halo cells

Figure 3.11 Laminar cylinder showing the linear extrapolation and normal projection correction applied to the halo cells

$$\frac{1}{\sqrt{dx^2 + dy^2}} \begin{bmatrix} -dy \\ dx \end{bmatrix} = \begin{bmatrix} n_x \\ n_y \end{bmatrix} \tag{3.8}$$

From the computed unit normal vector components, the physical position of the next halo cell vertices (x_{j+1} and y_{j+1}) can be calculated as shown in Figure 3.12. The first step is to calculate the $\vec{x}_{j,j-1}$ and $\vec{y}_{j,j-1}$ vectors as shown in Equation (3.9) and (3.10). Afterwards, the norm of these 2 vectors combined is calculated in Equation (3.11). Once the norm is calculated, this distance can be multiplied by the unit normal vectors to get the new position of the vertices x_{j+1} and y_{j+1} as shown in Equation (3.12) and (3.13). The points being referred to in Equation (3.9) to (3.13) can be seen illustrated in Figure 3.12.

$$\vec{x}_{j,j-1} = x_j - x_{j-1} \tag{3.9}$$

$$\vec{y}_{j,j-1} = y_j - y_{j-1} \tag{3.10}$$

$$\delta = \sqrt{\vec{x}^2_{j,j-1} + \vec{y}^2_{j,j-1}} \tag{3.11}$$

$$x_{j+1} = x_j + n_{x_j}\delta \tag{3.12}$$

$$y_{j+1} = y_j + n_{y_j}\delta \tag{3.13}$$

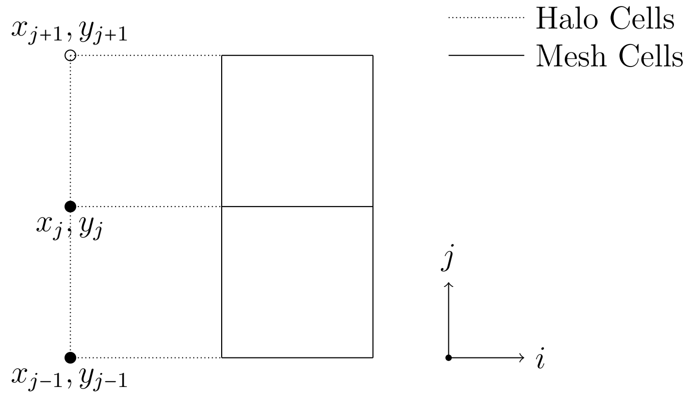


Figure 3.12 Normal Projection

NSCODE Memory Upgrade

Another issue arising from the previous implementation of the halo cells inside NSCODE is that the memory allocation is done in such a way that only the physical location of the first layer of halo cells is stored. The reason for this is that the physical location of the second layer of halo cells is not necessary in the computational stencil of NSCODE. In order to store the second layer of halo cells in memory, a modification of the memory allocation needs to be done. Since NSCODE is a C code and the way that the physical location of cells is managed inside arrays, modifications have to be made at the original memory allocation by storing 1 additional float inside the arrays for the additional layer of physical cells on the right side of the mesh block. Afterwards, all the starting and end points of loops inside the code need to be modified because they were explicitly defined by the previous boundaries of the array. In total, modifications inside 70 different files were necessary to implement these changes.

0	1	2	$\cdots$	i_{max}	i_{max}+1	i_{max}+2	empty

Figure 3.13 NSCODE original array memory allocation in the i direction for physical cells

In Figure 3.13, the original memory allocation is shown in which there is one empty array element to store a float. This array can be used to store the additional right side of the second layer of halo cells in the i direction. In Figure 3.14, the new memory allocation is done in which an additional empty array float container is added. The same operation is also done on the array in the j direction.

0	1	2	$\cdots$	i_{max}	i_{max}+1	i_{max}+2	empty	empty

Figure 3.14 NSCODE new array memory allocation in the i direction of physical cells

Calculation of Derived Values

Before going into the test cases and numerical results, it is important to know how the coefficient of friction and pressure coefficient are calculated. For simplicity purposes, the calculations are only shown in the x direction, but the process is similar in the y and z direction and the equations are provided in Annex A. Equation (3.14) to (3.17) show how the various $\tau_{i,j}$ are calculated. In each of these equations, velocity derivatives are involved in the process. These velocity derivatives (e.g. $\frac{dU}{dx}$, $\frac{dU}{dy}$, etc.) are calculated using the physical position of the cells within NSCODE. Of course, if any of these positions are falsified by the means of an improper method such as the use of linear interpolation, discontinuities will arise in the C_f results. Also, the normal vectors are calculated using the dx and dy derivatives as shown in Equation (3.18) and (3.19) which in turn are also used in the calculation of the C_{f_x} by the means of calculating the F_{f_x}, F_{f_y} and F_{f_z} friction forces components as shown in Equation (3.20) to (3.22). Once the friction forces components $F_{f_{xyz}}$ are calculated, the C_f

can be computed.

$$\tau_{xx} = \frac{1}{2}\lambda(\frac{dU_i}{dx_i} + \frac{dV_i}{dy_i} + \frac{dU_{i+1}}{dx_{i+1}} + \frac{dV_{i+1}}{dy_{i+1}}) + \mu_{total}(\frac{dU_i}{dx_i} + \frac{dU_{i+1}}{dx_{i+1}}) \tag{3.14}$$

$$\tau_{xy} = \frac{1}{2}\mu_{total}(\frac{dU_i}{dy_i} + \frac{dV_i}{dx_i} + \frac{dU_{i+1}}{dy_{i+1}} + \frac{dV_{i+1}}{dx_{i+1}}) \tag{3.15}$$

$$\tau_{xz} = \frac{1}{2}\mu_{total}(\frac{dW_i}{dx_i} + \frac{dW_{i+1}}{dx_{i+1}}) \tag{3.16}$$

$$\tau_{yz} = \frac{1}{2}\mu_{total}(\frac{dW_i}{dy_i} + \frac{dW_{i+1}}{dy_{i+1}}) \tag{3.17}$$

$$n_x = \frac{-dy}{\sqrt{dx^2 + dy^2}} \tag{3.18}$$

$$n_y = \frac{dx}{\sqrt{dx^2 + dy^2}} \tag{3.19}$$

$$F_{f_x} = \mu(\tau_{xx}n_x + \tau_{xy}n_y)\, ds \tag{3.20}$$

$$F_{f_y} = \mu(\tau_{xy}n_x + \tau_{yy}n_y)\, ds \tag{3.21}$$

$$F_{f_z} = \mu(\tau_{xz}n_x + \tau_{yz}n_y)\, ds \tag{3.22}$$

$$C_{f_x} = F_{f_x}\cos(\Lambda) + F_{f_z}\sin(\Lambda) \tag{3.23}$$

$$C_f = \frac{\sqrt{F_{f_x}^2 + F_{f_y}^2 + F_{f_z}^2}}{ds} \tag{3.24}$$

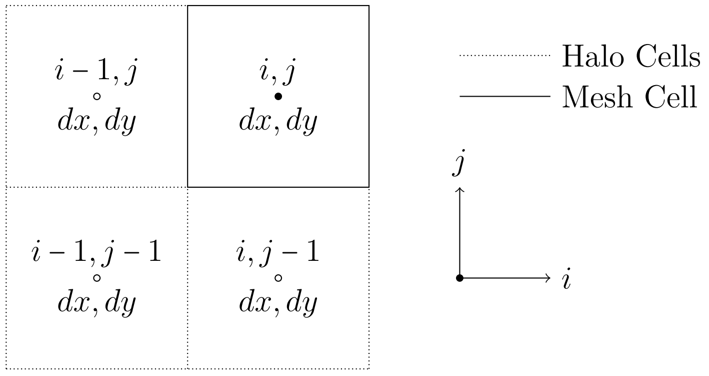


Figure 3.15 Schematization of corner halo cells at $i = 1, j = 1$

The corner halo cells are schematized in Figure 3.15 for $i = 1, j = 1$. This schematization is to understand how the speed derivatives are calculated in Equation (3.14) to (3.17).

The pressure coefficient is a similar story to the coefficient of friction. The error from the linear interpolated halo cells is found to be from the derivatives used for calculating the surface normals as shown in Equation (3.18) and (3.19), but also from the pressure calculated at the wall as shown in Equation (3.25). Of course, this error propagates to the derived component values $C_{p_{x,y,z}}$ calculated in Equation (3.29) to (3.31) and thus falsifies the results.

$$p_{wall} = 0.5(p_i + p_{i+1}) \tag{3.25}$$

$$F_{p_x} = -\frac{2(p_{wall} - p_\infty)}{\gamma M^2}\, ds\, n_x \tag{3.26}$$

$$F_{p_y} = -\frac{2(p_{wall} - p_\infty)}{\gamma M^2}\, ds\, n_y \tag{3.27}$$

$$F_{p_z} = 0 \text{ in 2D} \tag{3.28}$$

$$C_{p_x} = F_{p_x}\cos(\Lambda) + F_{p_z}\sin(\Lambda) \tag{3.29}$$

$$C_{p_y} = F_{p_y} \tag{3.30}$$

$$C_{p_z} = F_{p_x}\sin(\Lambda) - F_{p_z}\cos(\Lambda) \tag{3.31}$$

$$C_p = \frac{2(p_{wall} - p_\infty)}{\gamma M^2} \tag{3.32}$$

Test Cases

To test out the changes implemented and make sure the problem arising from solid solid discontinuities is correctly solved, three distinct test cases were planned out. The first one is a laminar cylinder clearly highlighting the curvature problem when linearly interpolating halo cells physical position. The second test case is a laminar NACA0012 with a collar grid patch applied on top of the airfoil. The final test case is a turbulent case with a NACA0012 using the Spalart-Almaras turbulence model (SA).

The first 2 test cases are run in laminar because it presents certain advantages as opposed to using a turbulent test case. Notably, it is easier to highlight issues stemming from a laminar solver as opposed to using a turbulence model, which could induce other errors or phenomena not in the scope of interest of solving the discontinuity problem which is already present in laminar.

3.3.3 Laminar Cylinder Test Case

The cylinder test case is a perfect example of what happens when the halos are linearly interpolated and the discontinuity problem that arises from it. The test case is not taken from any part of the literature since it is purely for numerical applications. The test case conditions are described in Table 3.3.

Table 3.3 Laminar cylinder test case simulation conditions

Parameter	**Value**
Angle of attack (AOA)	0°
Mach	0.2
Reynolds (Re)	30
Turbulence model	Laminar

For the mesh used, it was generated using NSGRID V7 containing the geometry definition with EGADS. The collar grid is directly taken from the mesh. It is an exact copy of the background mesh. It is highlighted in green in Figure 3.16. As mentioned previously, since the collar grid patch mesh is identical to its background grid, if the overset preprocessor is perfect there shouldn't be any solid solid discontinuities arising from its use when comparing against the use of the single grid. It is important to note that the simplicity of this mesh and test case also allows an easier understanding of what happens behind the code when debugging and printing information about the simulation.

Numerical Results

In terms of numerical results stemming from the changes made with the memory allocation of NSCODE and also the mesh recreation process, the discontinuities have greatly diminished as opposed to the original method using the linear extrapolation of the halos. For example, in the C_f graph in Figure 3.17, the offset from the curve of the solo simulation is greatly reduced. In Figure 3.18, it is possible to see the offset error at the starting point of the collar grid in $x \approx= -0.68$. The curve in green is the overset process done with the linear extrapolation of the halos. The curve in blue is the overset process done with the new method with the normal projection of the halos from the geometry definition. In theory, if the overset process is perfect the red curve should be completely aligned with our new method with the blue curve. However, since the derivatives are highly sensitive to the position of our halo cells which are necessary for the interpolation of the corner cells, it is expected to not have a perfect process. In Figure 3.18, it is highly apparent that the offset error is greatly diminished when using the

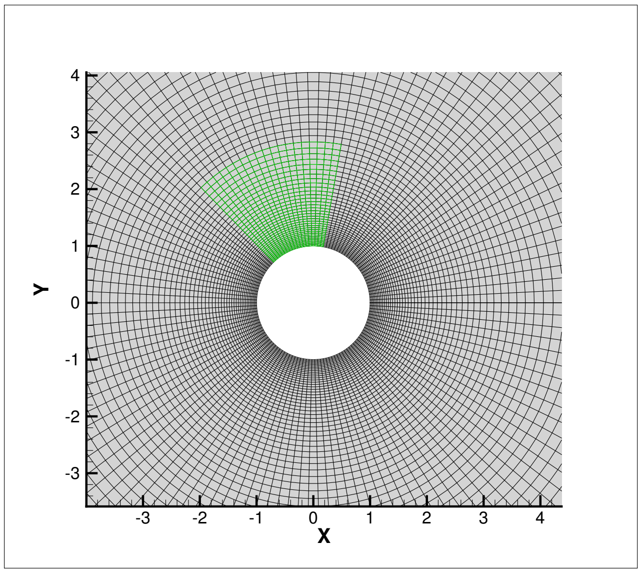


Figure 3.16 Mesh used for the laminar cylinder test case with its conforming collar grid highlighted in green

normal projection as opposed to the linear extrapolation of the halos when calculating the coefficient of friction. The offset from the C_f curve is less apparent in Figure 3.19 because the tangent is less pronounced at the end of the mesh (in $i = i_{max}$) and thus the offset error stemming from the linear interpolation is lesser.

For the C_p, it is possible to see a slight offset with the linear projection method which is corrected by using the normal projection of the halo cells as seen in Figure 3.20. C_{p_x} and C_{p_y} graphs are also provided in Annex A respectively in Figure A.2 and A.3. They present similar findings to the C_p graph.

Since the collar grid for this test case is identical to the background grid, it is possible to directly calculate the error stemming from the offset of the curves. In Table 3.4 and 3.5, it is possible to see the reduction in the error for the C_f and C_p components. As mentioned previously, the normal projection of the halos cells is not a perfect process in the sense that the physical location of the halo cell is not as good as for example directly taking cells from the mesh as user defined halo cells, but allows a more automated process with a significant error reduction as seen in this test case.

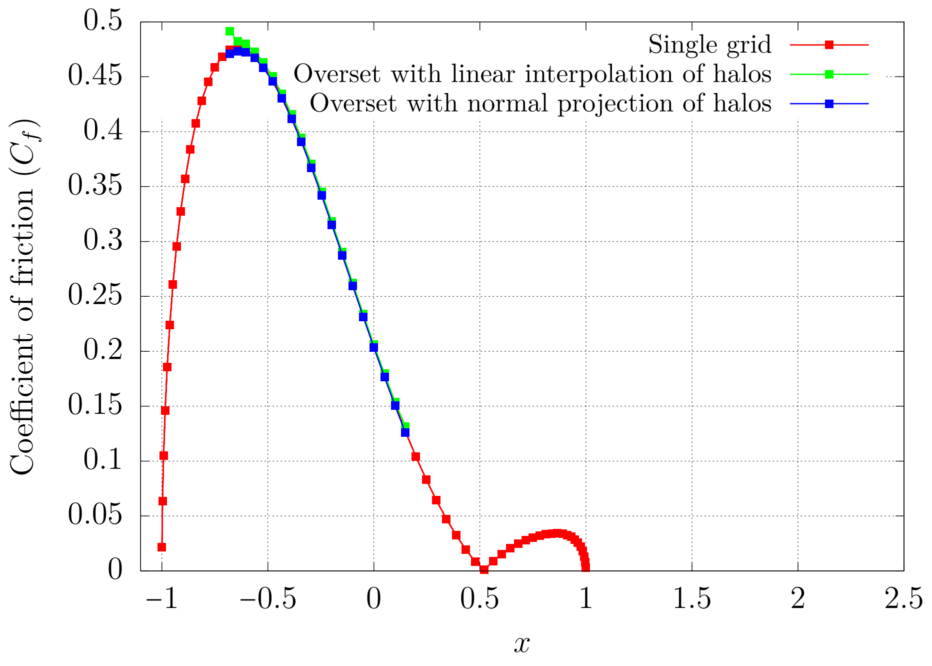


Figure 3.17 Laminar cylinder coefficient of friction (C_f)

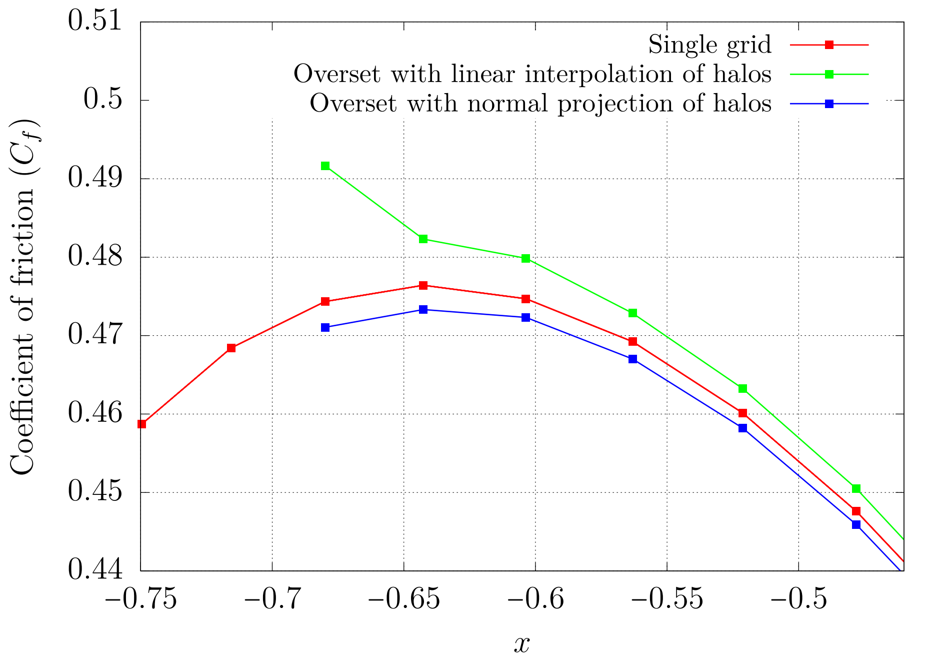


Figure 3.18 Zoom on the start of the collar grid for the C_f results of the laminar cylinder test case

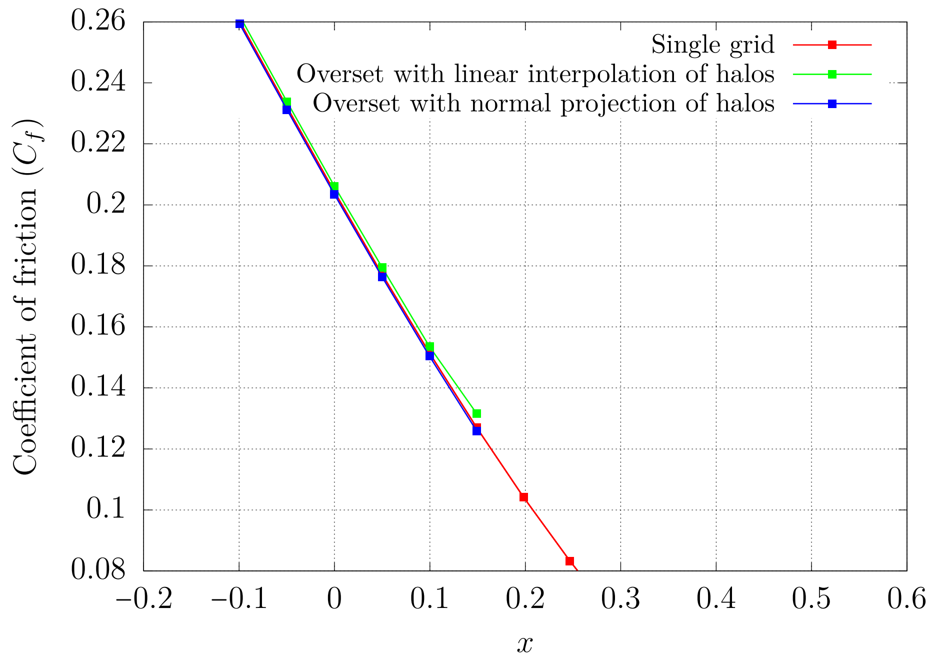


Figure 3.19 Zoom on the end of the collar grid for the C_f results of the laminar cylinder test case

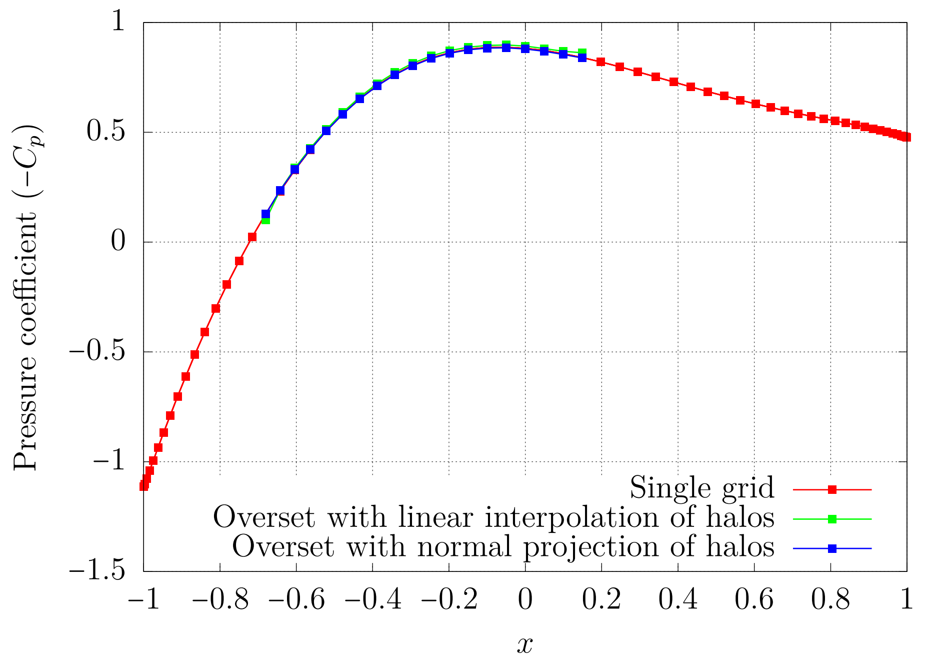


Figure 3.20 Laminar cylinder pressure coefficient (C_p)

Table 3.4 Offset error for the laminar cylinder test case for the C_f and its components

Technique	**$\sum$ of the offset error**		
	C_f	C_{f_x}	C_{f_y}
Linear extrapolation of the halo cells	0.06824	0.002909	0.001539
Normal projection of the halo cells	0.02872	0.001137	0.000797
Error reduction (%)	57.9	60.9	48.2

Table 3.5 Offset error for the laminar cylinder test case for the C_p and its components

Technique	**$\sum$ of the offset error**		
	C_p	C_{p_x}	C_{p_y}
Linear extrapolation of the halo cells	0.1923	0.003045	0.008771
Normal projection of the halo cells	0.0424	0.000547	0.001985
Error reduction (%)	78.0	82.0	77.4

3.3.4 NACA0012 Airfoil Laminar Test Case

The second test case used to validate the results of the geometric correction is a laminar test case taken from Swanson & Langer (2016) [27]. The simulation conditions of the test case are shown in Table 3.6.

Table 3.6 Laminar NACA0012 test case simulation conditions

Parameter	**Value**
Angle of attack (AOA)	0°
Mach	0.5
Reynolds (Re)	5000
Turbulence model	Laminar

The structured mesh used in this test case is an O-mesh generated with NSGRID V7 with 139 264 elements generated around a NACA0012 with a sharp trailing edge. In Figure 3.21, the collar grid is highlighted in green and is located on the upper surface of the airfoil which starts at $x \approx= 0.2$ and ends at $x \approx= 0.7$. As opposed to the previous cylinder test case, this collar grid is not an exact match of the background mesh, but rather a finer discretization of the geometry as expected when using collar grids. Also important to note is that the curvature at the start and end of this collar grid is less pronounced than in the case with the cylinder and has a direct impact on the numerical results obtained.

Numerical Results

For the numerical results for this test case, the benefits of the method are not as clearly obvious as for the laminar cylinder. A possible explanation for this is that in cases where curvature is less blatant, the offset error coming from the linear extrapolation of the halos is very similar to the performance achieved by using the normal projection of the halo cells. In Figure 3.22, it is possible to see that the blue curve representing the normal projection of the halos is superposed on top of the green curve representing the linear extrapolation method. Zooming in on the start of the collar grid in Figure 3.23, an offset error for both the linear extrapolation and normal projection of the halos methods is seen when comparing against the single grid results. However, at the end of the collar grid in Figure 3.24, the normal projection matches more accurately the single grid performance as opposed to the linear extrapolation. The reason for this is the higher curvature of the geometry present at the end of the collar grid where the weakness of the linear extrapolation method is clearly shown.

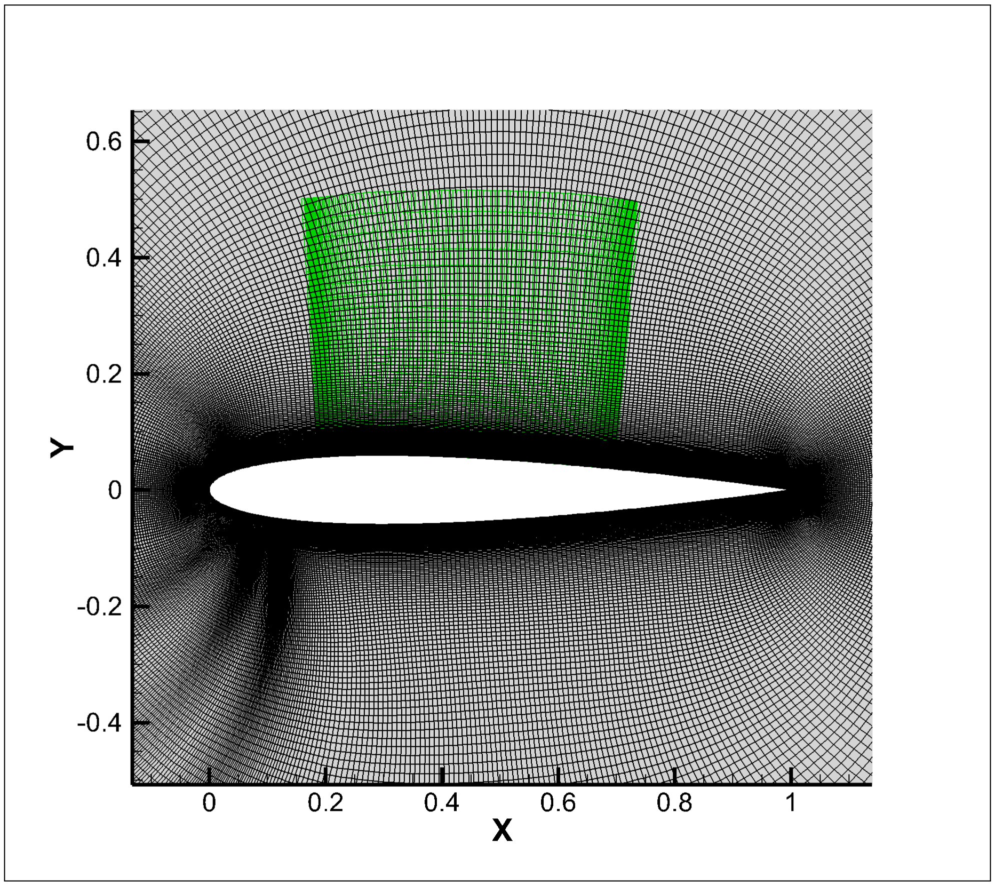


Figure 3.21 NACA0012 laminar mesh with its collar grid highlighted in green

For the C_p results, the benefits of projecting normally the halos are not obvious for this test case. If looking only at the C_p curve in Figure 3.25, it is possible to see the linear extrapolation method results are very similar to the performance achieved by projecting normally the halos. However, there is an offset error, but less striking than in the C_f results. The reason can be because the C_p equations as mentioned previously are less sensitive to the physical position of the halos cells, because they only take in physical derivatives as opposed to the C_f equations that also take in velocity derivatives dependent on the physical position of the halo cells. Graphs for the C_p components (C_{p_x}, C_{p_y}) are also provided in Annex A in Figure A.4 and A.5 respectively. In both cases, there is a mismatch on the single grid curves for both the linear extrapolation and normal projection methods. Again, in this case, the linear extrapolation (in green) is aligned with the normal projection curve (in blue).

The only apparent benefit for using the new method in this test case is illustrated in the C_f results as shown in Table 3.7 where we have a 6 % in error reduction. This can be explained by the C_f equations as mentioned previously who also take into account velocity

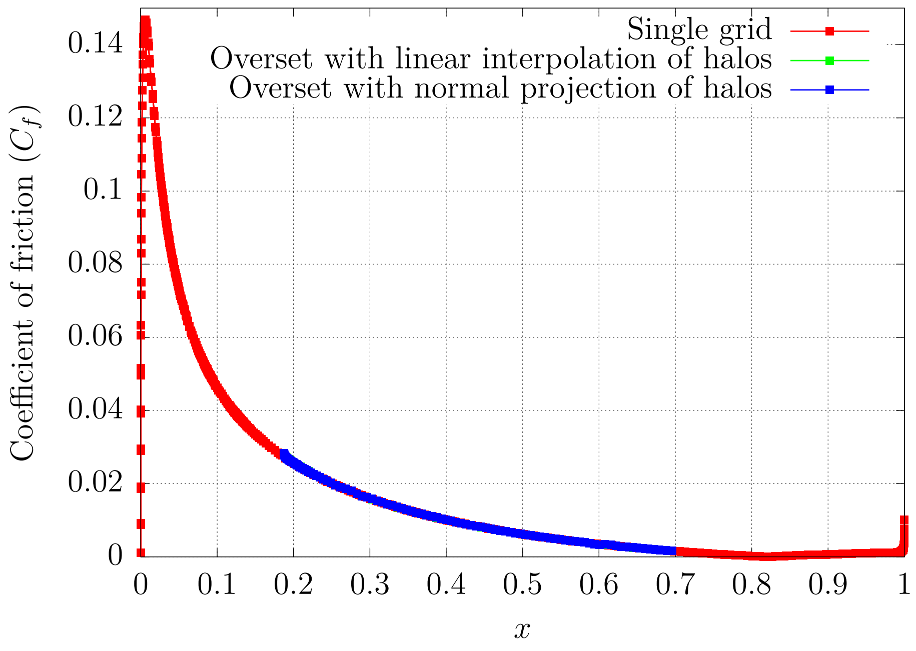


Figure 3.22 Laminar NACA0012 coefficient of friction (C_f)

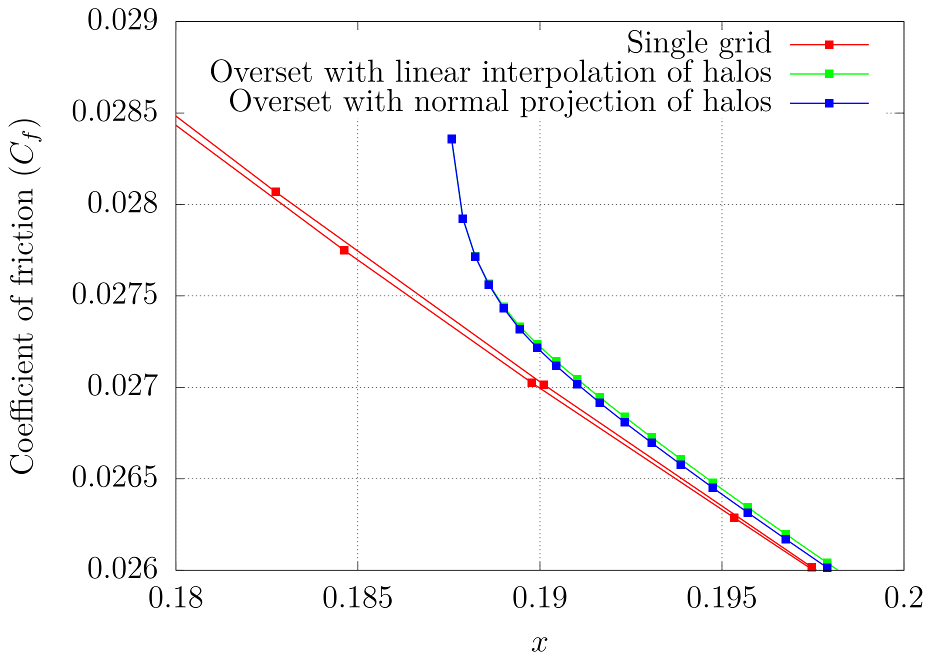


Figure 3.23 Zoom on the start of the collar grid for the laminar NACA0012 test case for the C_f

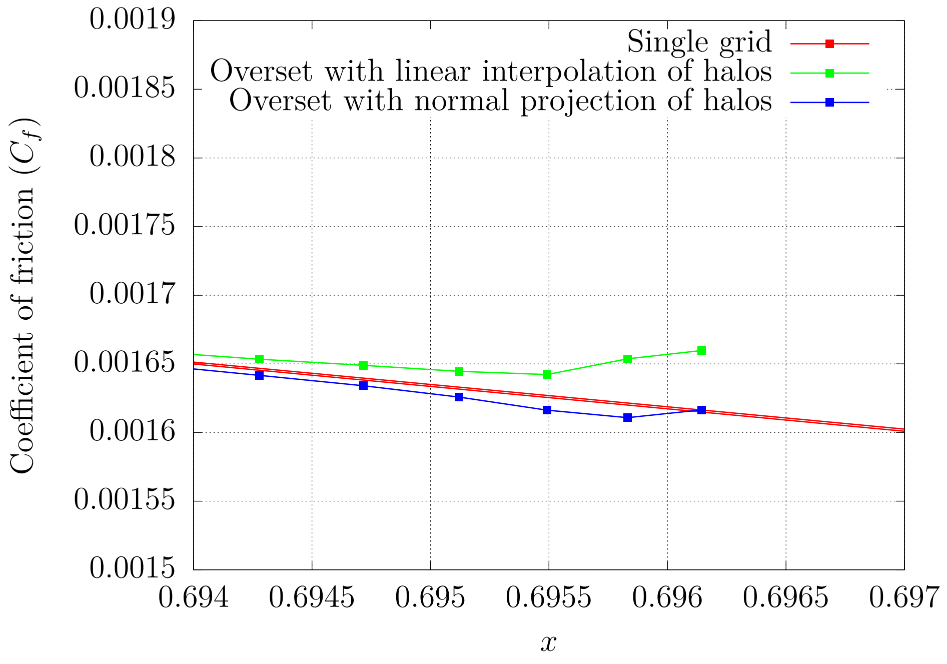


Figure 3.24 Zoom on the end of the collar grid for the laminar NACA0012 test case for the C_f

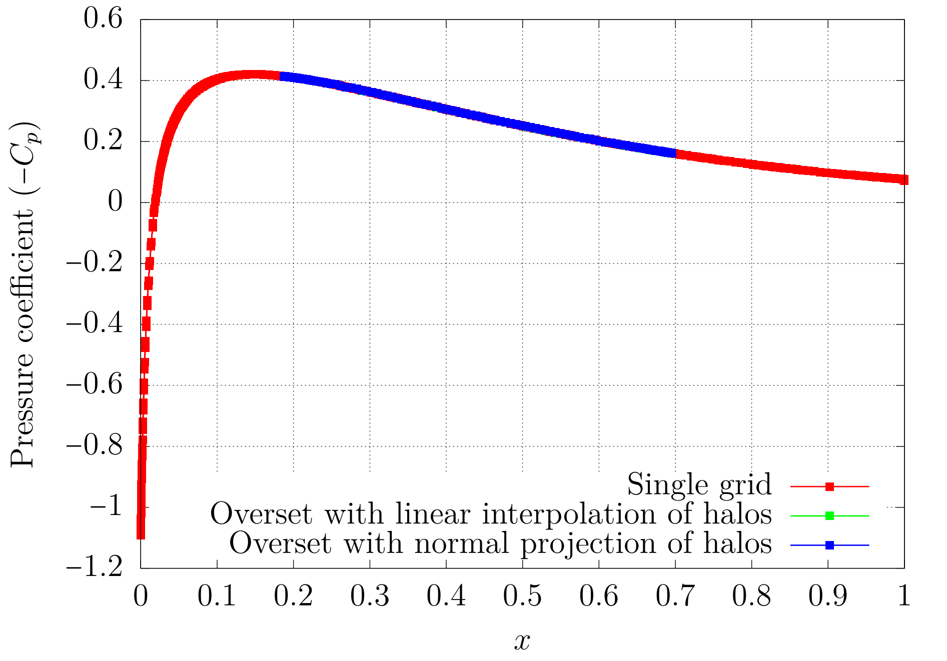


Figure 3.25 Laminar NACA0012 pressure coefficient (C_p)

derivatives. However, there is a problem with the C_p results as shown in Table 3.8 because the method performs worse as illustrated by the -42.5% which is an error augmentation. This is not alarming because these offset errors are stemming from the physical derivatives (dx, dy) when calculating the C_p. It shows that the normal projection fix is not a perfect solution when used with a low curvature geometry. The main conclusion from this test case is that the linear extrapolation method has similar performance to the normal projection method when using collar grids with low curvature at the start and end of the grid.

Table 3.7 Offset error for the laminar NACA0012 test case for the C_f and its components

Technique	**$\sum$ of the offset error**		
	C_f	C_{f_x}	C_{f_y}
Linear extrapolation of the halo cells	0.007123	0.005769	0.0003322
Normal projection of the halo cells	0.006644	0.005768	0.0003321
Error reduction (%)	6.72	0.02	0.03

Table 3.8 Offset error for the laminar NACA0012 test case for the C_p and its components

Technique	**$\sum$ of the offset error**		
	C_p	C_{p_x}	C_{p_y}
Linear extrapolation of the halo cells	0.02860	0.008125	0.1254
Normal projection of the halo cells	0.04076	0.008120	0.1253
Error reduction (%)	-42.50	0.06	0.08

3.3.5 NACA0012 Airfoil Turbulent Test Case

For the final test case to the solid discontinuities, a turbulent test case was chosen. This test case unfortunately highlights some of the problems that are still present even with the geometric fixes implemented for the collar grids. The chosen test case is from the NASA turbulence model validation cases for the Spalart-Allmaras turbulence model [28]. The simulation conditions are found in Table 3.9.

Table 3.9 Turbulent (SA) NACA0012 test case simulation conditions

Parameter	**Value**
Angle of attack (AOA)	$0°$
Mach	0.15
Reynolds (Re)	6 000 000
Turbulence model	Spalart-Allmaras

For the mesh chosen it is an NACA0012 O-mesh with a sharp trailing edge. A collar grid is applied to the lower surface of the airfoil. Both meshes are generated using NSGRID V7 and can be seen merged together in Figure 3.26.

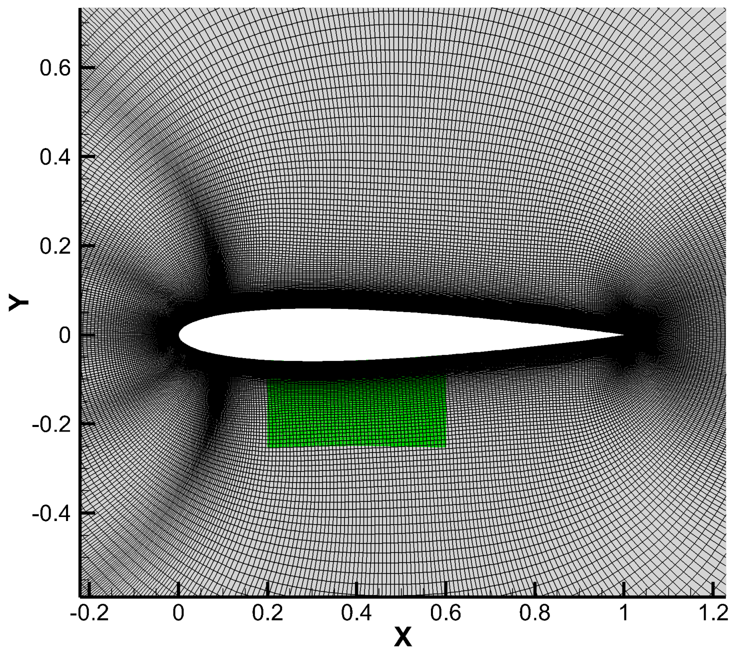


Figure 3.26 Turbulent (SA) NACA0012 mesh with collar grid applied on the lower surface of the airfoil

Numerical Results

For the simulations results, it is straightaway apparent that numerical problems are present for the C_f result as seen in Figure 3.27. However, this is not all bad news because the only difference between this test case and the two previous test cases presented before is the addition of the turbulence model. It is important to note that the nature of these discontinuities is different than the ones present in laminar because of the addition of the turbulence model. This all points to a different issue within the code when using a turbulence model.

Looking at the C_p results there is not offset present meaning that geometric fixes seem to have worked out with our normal projection curve aligned with single grid curve in Figure3.28. Also, since in this case once again, the curvature is not pronounced the C_p results for the linear extrapolation are aligned with the normal projection results.

For the C_f results and looking how the interpolation is done within NSCODE, there seems to be missing an interpolation of the velocity derivatives when updating the numerical solution at each iteration which might explain why the sudden discontinuities when switching from a laminar simulation to a turbulent one. However, this is only speculation for the source of the probable culprit because this avenue has not yet been fully explored in the code.

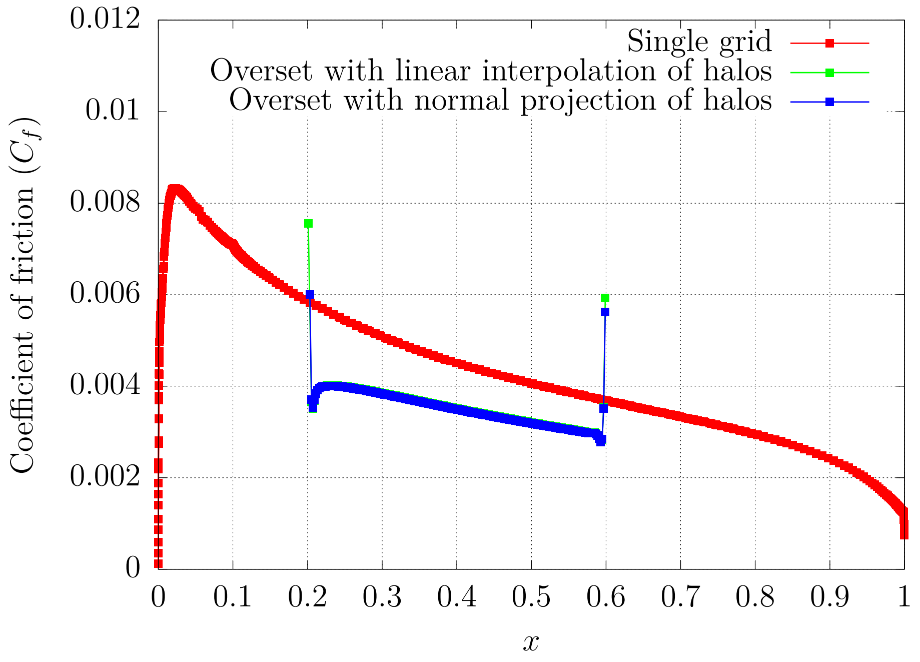


Figure 3.27 Coefficient of friction (C_f) results for the turbulent (SA) NACA0012 test case

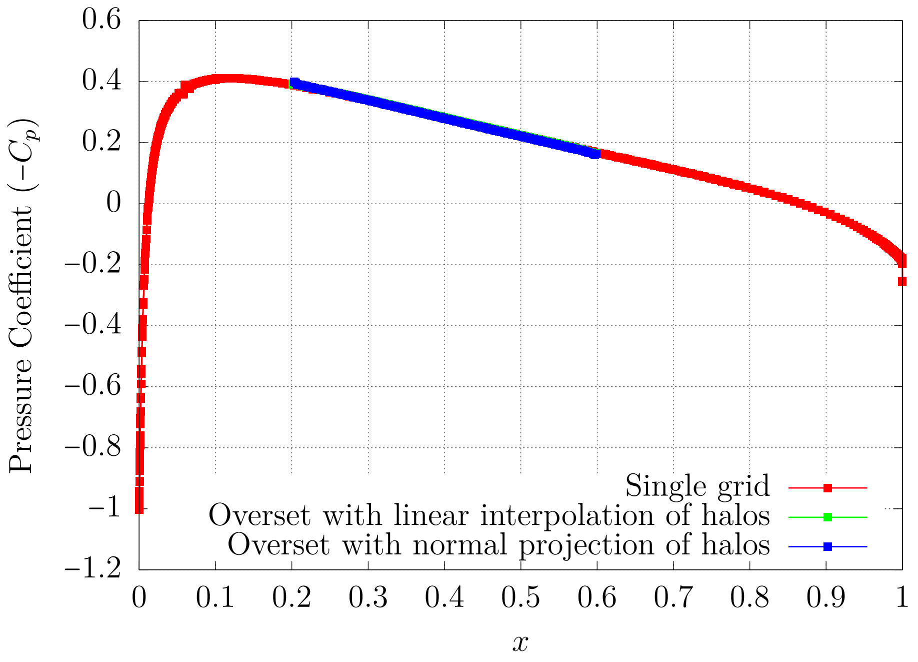


Figure 3.28 Pressure coefficient (C_p) results for the turbulent (SA) NACA0012 test case

3.3.6 Limitations of the Present Method

As mentioned with the laminar NACA0012 and turbulent NACA0012 test cases, there is some pitfalls with the present implementation of the overset framework in NSCODE for solid solid discontinuities. Firstly, there is an apparent problem when using a turbulence model as highlighted in the SA test case. In the case of this thesis, the problem surrounding discontinuities in laminar was fixed because of the geometry implications. However, some problems are still there and point to the interpolation of the velocities done in the flow solver which is not an avenue that was explored during the course of this thesis. It is worth mentioning that the geometry fix was not the only approach taken to solve the discontinuities. A fix involving changing the calculation method of the gradients with a Green-Gauss method was explored, but not retained since after the implementation no notable differences/ameliorations in the numerical results were noted. Future work will have to be done by looking how the C_f is impacted by the turbulence model and the steps taken by the overset framework to make the interpolation at each iteration.

3.4 Schwarz Correction

Another key part of the overset literature concerning collar grids is the implementation of the Schwarz correction to take into account the finer discretization that is encountered when using collar grids. This method inside NSCODE was implemented in a similar fashion that it is explained in Schwarz (2006) [22]. In the literature review, it was illustrated for cell vertices in Figure 2.14. In the case of NSCODE, since cell centers are used for weight calculcation, this operation is then made for cell centers.

For each cell of the interpolated mesh underlapping the collar grid, their position needs to be corrected to have the correct wall distance. This is done by looping through each j column of the mesh and find in which of these row the cell center of the interpolated cell is located. Afterwards, the cell center from the collar grid closest to the interpolated cell center is found. Once a pair of interpolated cell center and the closest collar grid cell center is found, a projection vector from the face of the cell at the wall closest to the interpolated point is calculated. With this projection vector, it is possible to calculate a virtual interpolation point as noted in Equation (2.4). This process is repeated for every background cell that needs to be interpolated from the collar grid. Once all the virtual interpolation points have been found, they are taken to the interpolation weights calculation algorithm instead of using the cell centers.

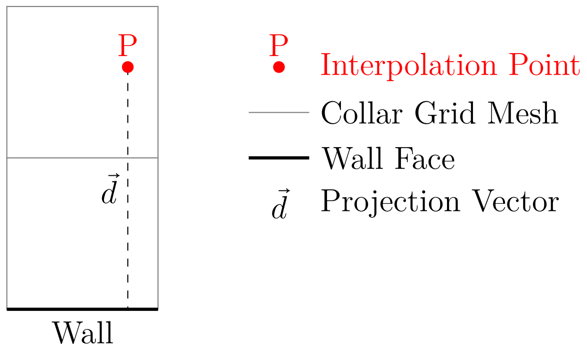
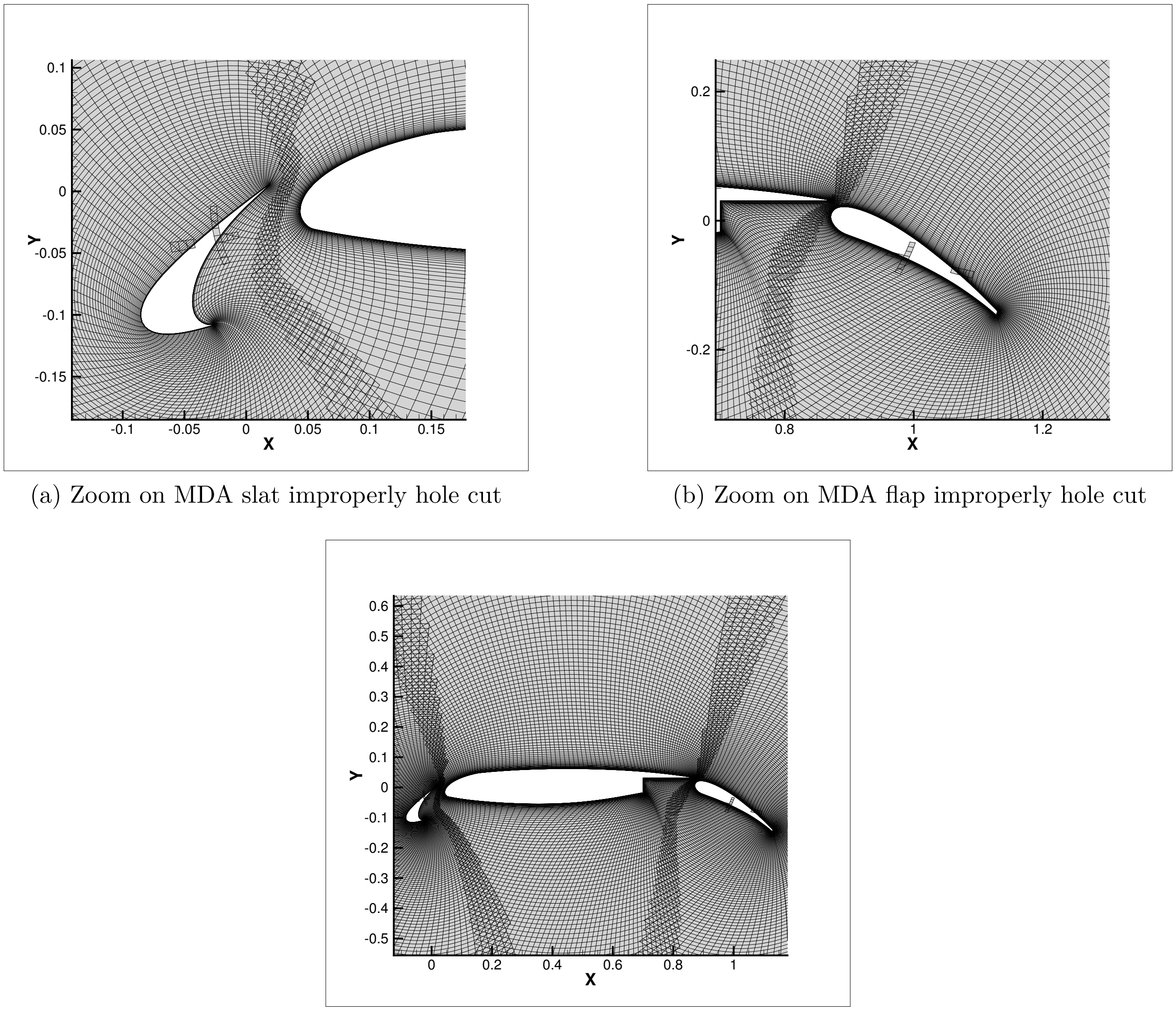


Figure 3.29 Schematization of the Schwarz correction applied within NSCODE

Even after implementing this algorithm, it was hard to find a test case that the normal overset preprocessor could not handle and produce converged results. Thus, this is the reason why no numerical results will be presented in this section.

3.5 Improvements Made to the Hole Cutting Process

Some improvements have been made to the hole cutting process to make it more robust because in certain geometries it would fail. Failing the hole cutting process means that some overlapping cells were not blanked inside the geometry of another mesh. A good example of this is with the MDA airfoil shown in Figure 3.30. Note that in that particular case, some cells have their cell center in the triangulated geometry and should have been blanked. The reason this happens is because of the added buffer cell layers (2 in total).



(a) Zoom on MDA slat improperly hole cut

(b) Zoom on MDA flap improperly hole cut

(c) MDA full geometry improperly hole cut

Figure 3.30 NSCODE old hole cut method showing improperly hole cut flap and slat

This problem is due to the method used for hole cutting the geometry. As mentioned in the literature review, the interior of the geometry is triangulated as shown in Figure 2.10.

The cell center of an overlapping cell is tested to see if it is contained in any of the interior triangles of the triangulated geometry. Situations like the one illustrated in Figure 3.31 is the primary reason why in some geometries the hole cutting fails.

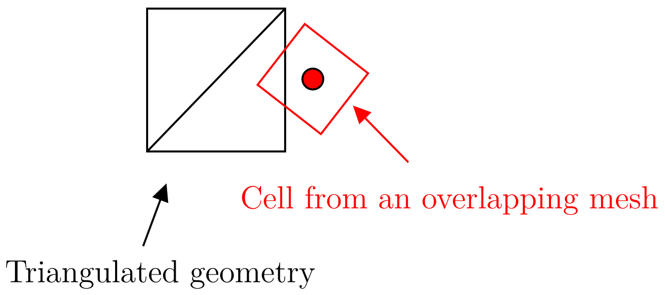


Figure 3.31 Example of the old hole cutting method

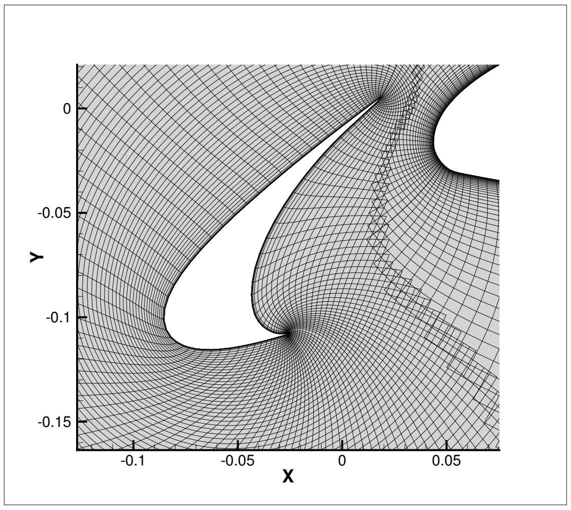
3.5.1 Default Method Improvements

A better way to ensure that cells are not left unblanked is to also test the nodes of the tested overlapping cell. This is illustrated in Figure 3.33. This ensures that at least one of the nodes or the cell center falls inside the geometry of the mesh. Of course, this test is not exhaustive in the sense that some scenarios could end up in a situation where all the nodes and the cell center are not found in the triangulated geometry. However, these situations are rare. When generating meshes with NSGRID, users normally try to have overlapping meshes cell size similar to the other meshes. The result of the successful implementation of the method is shown in Figure 3.32 which is the same geometry and meshes that the previous hole cutting method failed on illustrated in Figure 3.30.

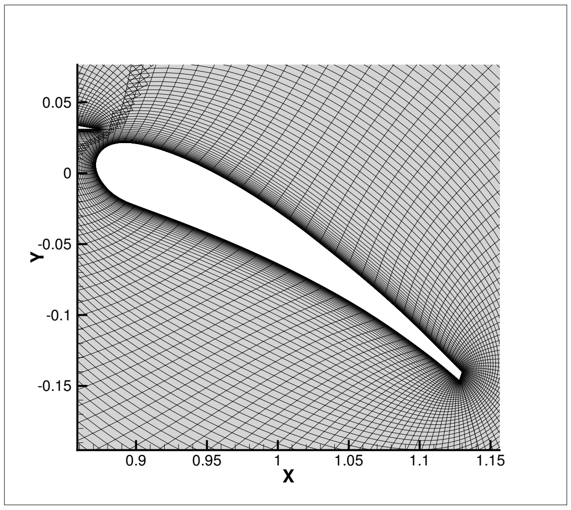
Table 3.10 Timing of the hole cutting method

Hole cut Method	**Time (seconds)**
Previous Method (Cell center only)	0.21037
New Method (Cell center and cell nodes)	0.671875

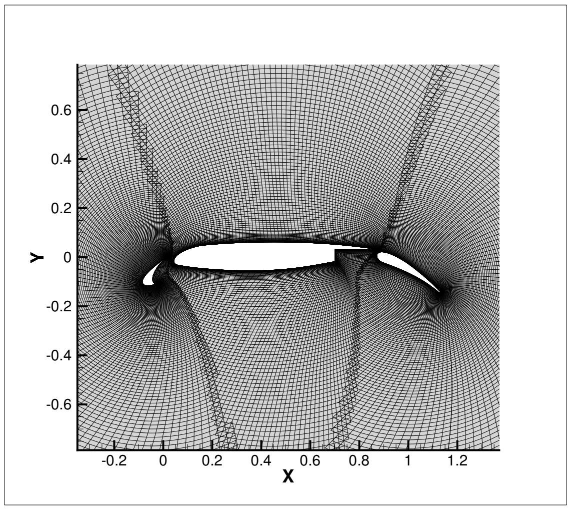
Since more points inside the triangulated geometry are being tested, this is more computationally costly and is directly reflected by the time taken for the calculations. In this example, with the MDA mesh that has 98 304 elements, a 3x time increase is observed in Table 3.10.



(a) Zoom on MDA slat properly hole cut



(b) Zoom on MDA flat poperly hole cut



(c) MDA full geometry propely hole cut

Figure 3.32 NSCODE new hole cut method showing properly hole cut flap and slat

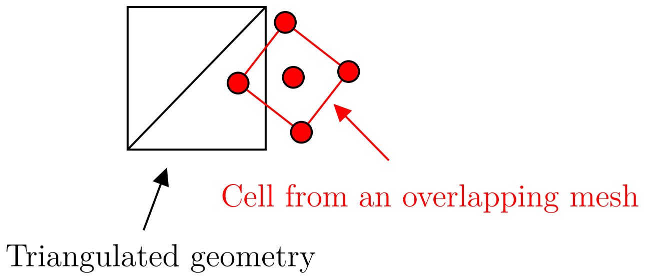


Figure 3.33 Example of the new robust hole cutting method

3.5.2 EGADS CAD Hole Cutting

Another method that was envisioned for the hole cutting process is by directly using the CAD definition of the geometry using the EGADS geometry kernel library. All meshes generated with the most recent version of NSGRID have their CAD geometry stored inside them. This allows leveraging the power of the EGADS API which has some built-in function for determining geometry intersections. However, a current pitfall of the implementation of the library is that it only supports one loaded geometry at a time. This makes it useless currently for overset purposes as we often need to hole cut 2 or more geometries at a time. Upgrades for support of multiple geometries will have to be made in order for it to work.

CHAPTER 4 3D OVERSET METHODS

In order to have an appropriate framework to test out overset methods in 3D and to take advantage of the modular nature of the newest CFD code of Professor Laurendeau's lab, a 3D unstructured overset module needed to be implemented inside CHAMPS. This new module created during the course of this thesis takes advantage of the knowledge developed by the previous generations and current generation of students working on the overset methods inside the lab. As mentioned previously, the foundations of this overset module are taken in a large part by what has been done in 2D in NSCODE, but taken to the next level in 3D inside CHAMPS. Some methods and approaches taken in 2D are simply not feasible to implement such as the CDT method used for hole cutting. This chapter touches upon the entire software implementation that has been coded during this thesis and the current state of pre-processor as of this writing.

4.1 CHAMPS - Overset Pre-Processor Overview

The newly implemented overset preprocessor inside CHAMPS follows a similar procedure in terms of the steps taken to perform the overset process as in NSCODE.

The principal steps consist of the following order:

1. Merge of the CGNS grids used for the overset simulation by adding them sequentially in a newly created CGNS file
2. Wall distance calculation of each of the grids using the Eikonal method [29]
3. Creation of the boxing using the RCVG method in 3D to speed up the donor search
4. Hole cutting using X-ray method [12]
5. Overlap definition and donor search using wall distance/grid hierarchy criterion
6. Calculation of donor weights based on interpolated cell center position using a trilinear interpolation method
7. Add overset interfaces to the overset CGNS file containing information pertaining to the weights, blanked cells, and interpolated/donor cell pairs

Once the above steps have been completed, the overset pre-processing is done and the mesh file contains all the necessary information to be used for an overset flow simulation.

4.2 Merge of the CGNS files

The first step of the overset preprocessor is to merge the various CGNS files used for the overset simulation. This is done via calling the CGNS API by creating a new empty CGNS file to receive sequentially each of the zones contained within the other CGNS files. Once the empty CGNS file has been created, a copy of each of the zones from each mesh is made inside of it. Each of these zones is able to be linked to the original mesh by assigning them a unique set id. For example, if the zone comes from the first mesh file, its set id is set to 1.

4.3 Wall Distance Calculation

For determining in a donor cell / interpolated cell pair which will be the donor and which will be the interpolated, a criterion needs to be put in place. This is done in 2 ways. The first is the hierarchy criterion as used in the 2D overset suite of NSCODE where if a zone has a higher hierarchy than the zone it is being compared to, then the cell from the higher hierarchy zone is the donor cell and the one from the lower hierarchy zone is the interpolated cell. The second criterion that is checked after the hierarchy of the mesh is the wall distance. In CHAMPS, wall distance can be calculated by two different methods. The first one is a 3D projected Euclidean wall distance. The second one is calculated using the Eikonal method which was originally developed inside NSCODE by Bouchard (2017) [29]. This method was ported over to CHAMPS by the research team and is currently used by the overset preprocessor as the wall distance calculation method.

4.4 Oct-Tree Boxing Implementation

In order to speed up the donor search, a spatial partitioning algorithm needs to be put in place to quickly find which cells are close matches to a possible interpolated/donor cell. The way it is done in the overset suite of CHAMPS is very similar to the way it was done in NSCODE with the RVCG algorithm, but this time in 3D. The way the algorithm works in an unstructured way in 3D is done via several steps for each of the zones contained in the overset mesh. The general form of this algorithm is called an oct-tree and has been successfully used in the past in the literature to speed up donor searches in overset algorithms [30].

The first step is to get the zone extrema in the x, y, z directions (x_{min}, y_{min}, z_{min} and x_{max}, y_{max}, z_{max}). Once the extrema have been found, the mesh can be subdivided in what is called a boxing level. Each boxing level is comprised of sub boxes. The dimensions of each sub box are defined by a dx, dy, dz calculated based on Equation (4.1) to (4.4).

$$n_x, n_y, n_z = \sqrt{\frac{\text{Number of Elements in the Zone}}{\text{Critical Number}}} \tag{4.1}$$

$$dx = \frac{x_{max} - x_{min}}{n_x} \tag{4.2}$$

$$dy = \frac{y_{max} - y_{min}}{n_y} \tag{4.3}$$

$$dz = \frac{z_{max} - z_{min}}{n_z} \tag{4.4}$$

The critical number in Equation (4.1) is usually set to 10 000 as good measure as usually meshes generated for CHAMPS rarely have more than 10 000 elements per zone for load balancing purposes. Once the dx, dy, dz have been determined, each of the sub boxes dimensions and positions can be computed. They are also defined by their extrema since they are 3D axis-aligned bounding boxes (AABB) as illustrated in Figure 4.1.

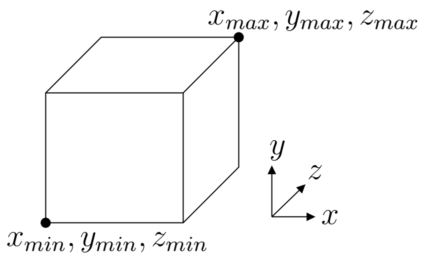


Figure 4.1 Example of a sub box (AABB)

Once the sub boxes have been created for the first boxing level, the list of all the elements contained within the current zone is passed through. To make sure that elements are not missed by only for example testing if their cell center is contained within a sub box, a bounding sphere that contains the whole element is created. This bounding sphere is characterized by two properties namely its center coordinates and its radius. The center coordinates are the same of the cell center of the element being considered. The radius is calculated based on the largest distance between the cell center and its nodes multiplied by a safety margin of 1.5. This is illustrated with a tetrahedral element in Figure 4.2, but is the same procedure with different type of elements.

Once the bounding sphere has been created, a collision test with each of the bounding spheres is executed to see in which sub boxes they fall into. This collision test consists of clamping the bounding sphere on the sub box and checking if the distance between the clamped sphere and the clamped point is smaller than the radius of the sphere [31]. If it is then the sphere

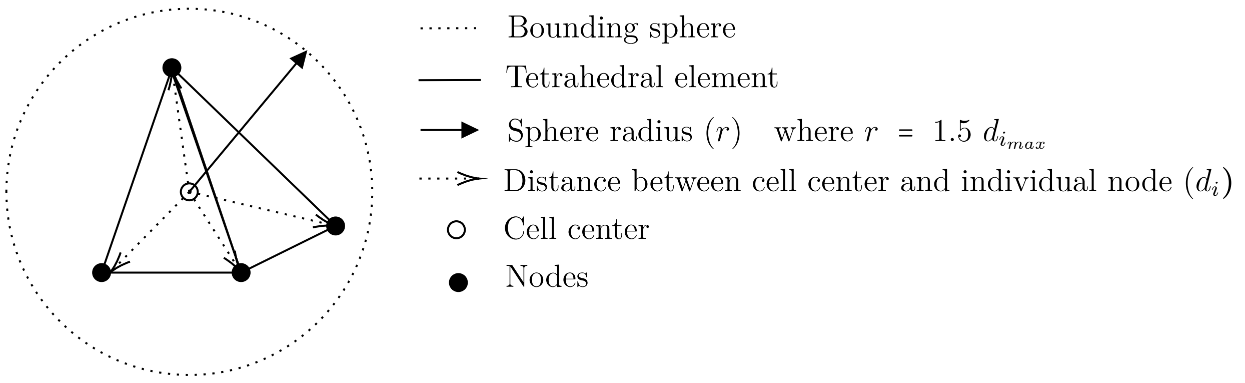

Bounding sphere
Tetrahedral element
Sphere radius (r) where $r = 1.5\ d_{i_{max}}$
Distance between cell center and individual node (d_i)
Cell center
Nodes

Figure 4.2 Example of a bounding sphere created around a tetrahedral element

is considered to be inside the sub box and thus the element also.

Once all the elements have been assigned to one or multiple sub boxes, a check on the number of elements contained per sub box is performed. If the number is greater than an assigned number which is referred to as the critical number than the box is subdivided into a new boxing level. This is done recursively until either a maximum specified boxing level is reached or the number of elements contained within each sub box is under the specified critical number.

An example of the oct-tree algorithm on a grid of an extruded NACA0012 profile is shown in Figure 4.3 and 4.4.

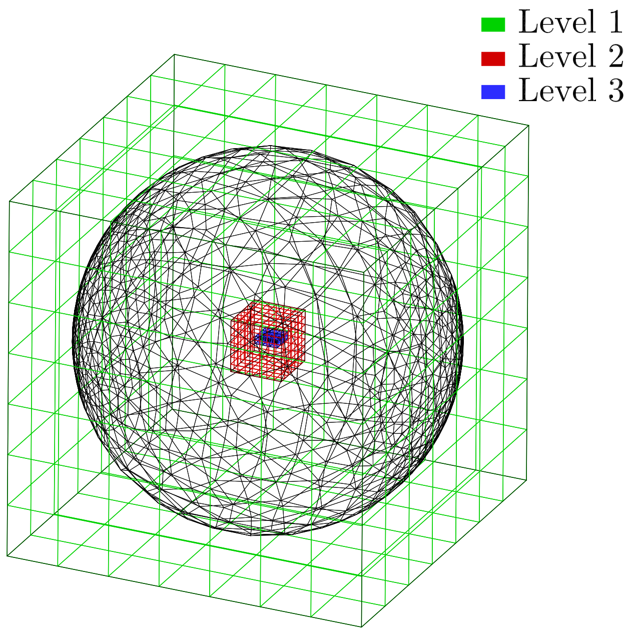


Figure 4.3 Example of the boxing made around an extruded NACA0012 airfoil mesh with each of the boxing levels highlighted with a different color

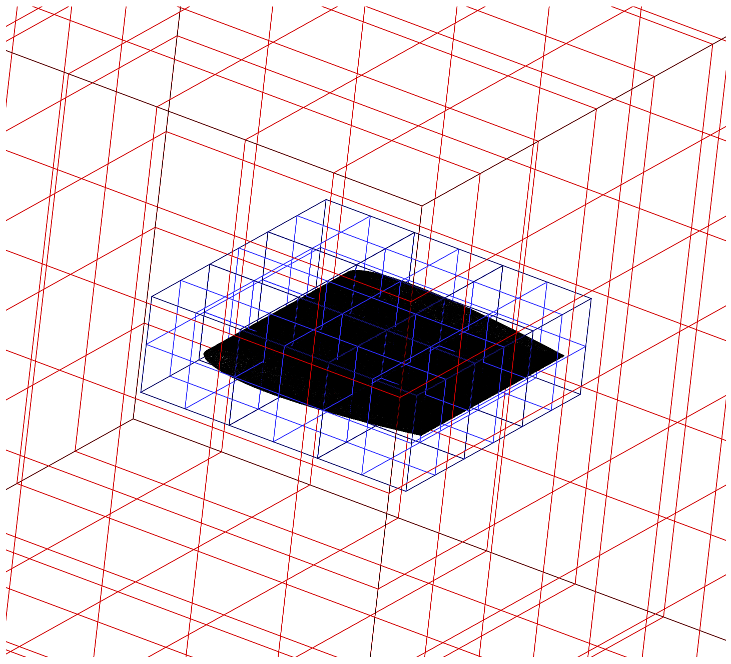


Figure 4.4 Zoom on the boxing made around an extruded NACA0012 airfoil mesh with each of the boxing levels highlighted with a different color

4.5 X-ray Hole Cutting

The X-ray hole cutting algorithm used for CHAMPS is largely inspired by the work done by Meakin in his original article describing his approach to using X-rays to hole cut a geometry from an overlapping mesh [12].

The way it is implemented in CHAMPS is done through several steps. First, an AABB is created around the solid wall geometry of an object that is specified by using a family name which is a name property that is defined in the CGNS standard which allows to clearly specify which geometries to be hole cut. Specifying a family name is done in the meshing process in Pointwise when implementing the boundary conditions on the geometry that is being meshed. Once the AABB has been created, a series of rays are created that start from the floor of the box and end on the ceiling of the box. For purposes of illustration, the floor and ceiling of the box is situated in the xz plane, but there is also provisions in the code to select other planes such as xy or yz as floors and ceilings.

The properties of the rays are the following :

- Their start and end points in xyz coordinates
- The number of piercing points and the xyz coordinates of the pierce points

In Figure 4.5, the X-rays are discretized in the xz plane. The spacing of the X-rays can be specified by manually inputting a dx and dz between the rays. Although, an automatic method to discretize the rays is also implemented by using the smallest edge on a face of the solid geometry that is being hole cut. In the literature, it is recommended to have 3 to 10 times smaller the discretization than the smallest edge of the face of the geometry [12]. However, this method is not perfect because sometimes a degenerated edge appears in which dimensions approaches 0 and cause an extremely fine discretization of the rays in the x and z directions. Finer discretization results in an imposing number of rays which thus impedes calculation times because for each of these rays piercing points need to be determined.

Speaking of piercing points, this is done using a collision detection algorithm between a line segment which is what our X-ray is and a triangle. The algorithm was taken from Ericson (2004) [31]. In Figure 4.6, an example of what the triangle and line segment collision test consists of is illustrated. The procedure is essentially the same when applied to a quad face element, the only difference is that the quad element needs to be split up into two triangles which are then used to perform the line segment and triangle collision test. This can be seen in Figure 4.7 with the imaginary delimitation separating the quad face element into

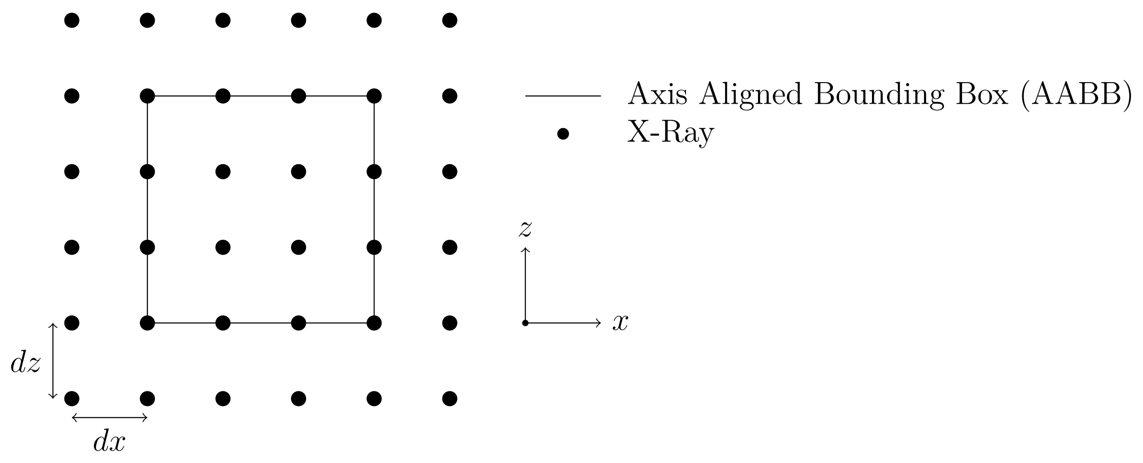


Figure 4.5 X-ray layout on an xz plane

two distinct triangular face elements that are used in the collision detection algorithm. Code snippets written in Chapel for the algorithms of the triangular and quad face elements are provided in Annex B and B respectively.

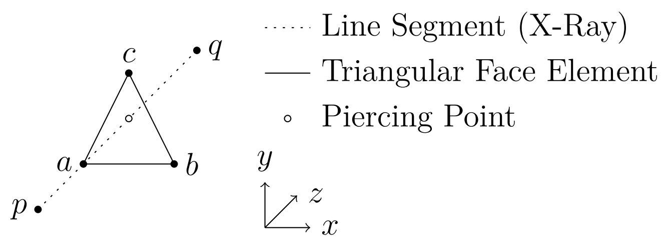


Figure 4.6 Example of the line segment and triangular face element collision test

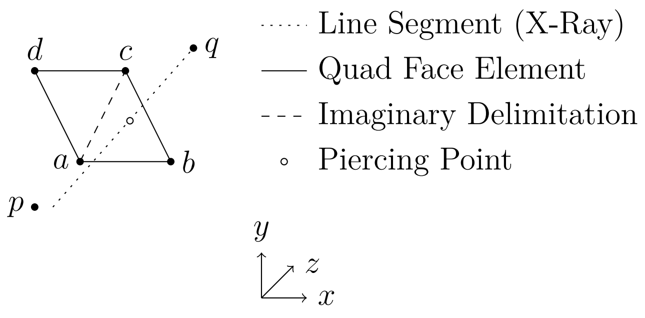


Figure 4.7 Example of the line segment and quad face element collision test

The collision detection algorithm needs to be run on each X-ray for every face of the solid body geometry that is being considered for hole cut. This can take a considerable time depending on the number of faces of the geometry but also because of the number of rays. Once the piercing points have been determined, it is now possible to test the cell center and nodes of an overlapping mesh against the X-rays. The first test consists of checking if the point being tested is found to be inside the bounding box of the solid body geometry. If it is the case, it is possible to easily locate in which 4 rays the point tested fall into because the rays are positioned like a Cartesian grid. Once the 4 rays are found, an elevation check is made to see if the point falls in between two piercing points. If it is the case, the point is considered inside the geometry and its b_i value is set to 0. For example, in Figure 4.8, it is possible to see that the red point P is found between the two piercing points of the X-ray where the ray pierces the bottom of the airfoil at the "In" point and exits at the "Out" point of the upper surface of the airfoil. If the point would fall outside of piercing points as it is the case of the green point F in Figure 4.9, its b_i value will be set to 1. A top-down view in the xz of the same example is seen in Figure 4.10. A 3D example of this is seen in Figure 4.12 where the MDA airfoil is pierced at regular intervals of 0.1 in x and z by the X-rays. The same 3D example is seen Figure 4.13 but from a side-view perspective.

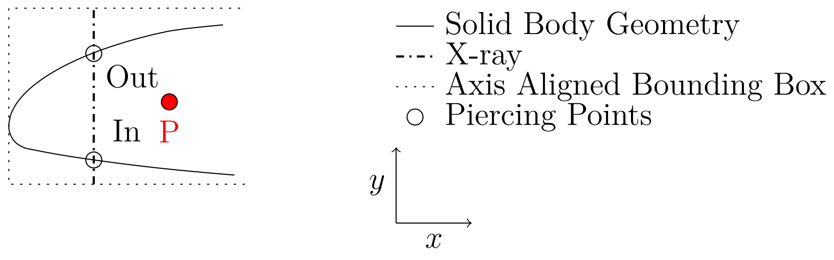


Figure 4.8 Inside and outside test on the MDA main airfoil zoomed in on the leading edge

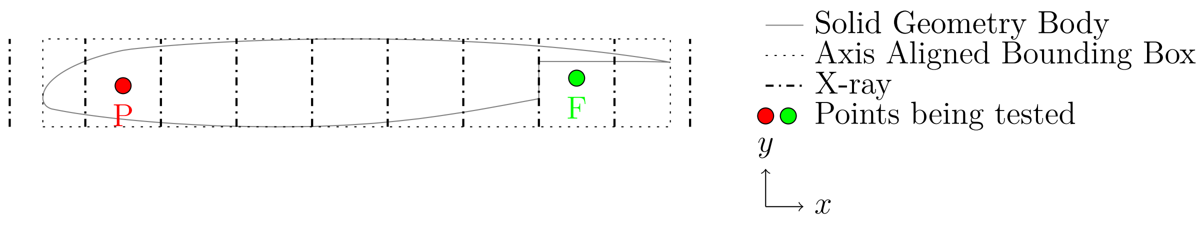


Figure 4.9 Example of the MDA main airfoil with X-rays in the xy plane

To determine if the cell is blanked or not, bilinear interpolation weights are calculated based on the position of the point tested as illustrated in Figure 4.11. The (x, z) coordinates of the tested point can be transformed easily into adimensionalized (η, ξ) coordinates which range from 0 to 1. The weights are calculated using Equation (4.5) to (4.8). Once the weights have been calculated and determining the b_i status for each of the rays against the tested point, the blanking status of the cell can be determined using Equation (4.9). If the calculated B is lower than 0.5 then the cell is considered blanked [12].

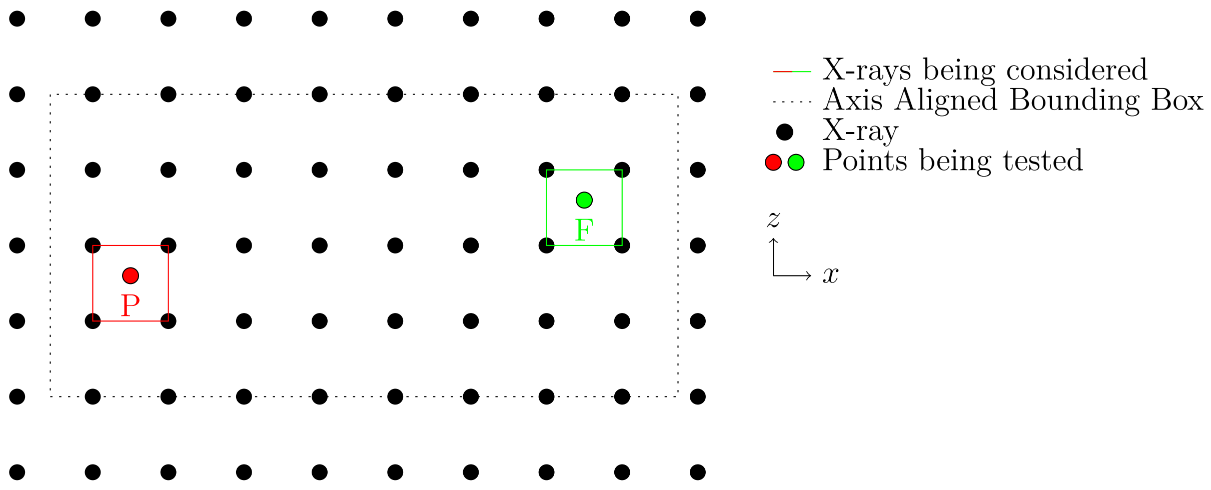


Figure 4.10 Example of the MDA main airfoil with X-rays in the xz plane

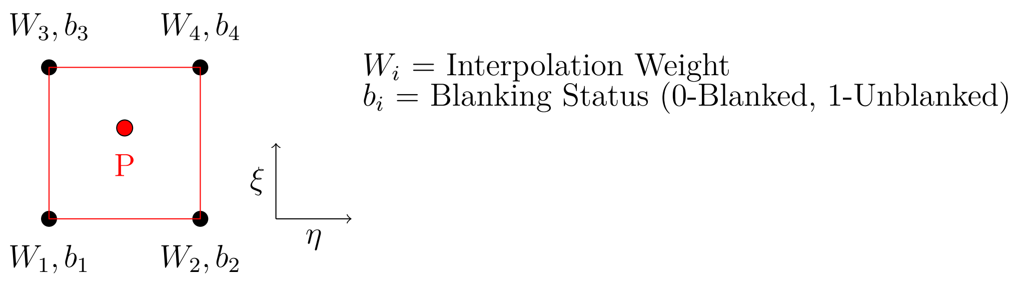

W_3, b_3
W_4, b_4
W_1, b_1
W_2, b_2
P
ξ
η
W_i = Interpolation Weight
b_i = Blanking Status (0-Blanked, 1-Unblanked)

Figure 4.11 Bilinear interpolation stencil for X-ray in the adimensionalized $\eta\xi$ plane

$$W_1 = (1-\eta)(1-\xi) \tag{4.5}$$

$$W_2 = (1-\eta)\xi \tag{4.6}$$

$$W_3 = \eta(1-\xi) \tag{4.7}$$

$$W_4 = \eta\xi \tag{4.8}$$

$$B = \sum_{i=1}^{4} W_i b_i \quad \text{if } B < 0.5 \text{ then cell is blanked} \tag{4.9}$$

This process is ran for every cell of an overlapping mesh that falls into the bounding box of the solid geometry that needs to be hole cut. An example of a geometry hole cut using the CHAMPS X-ray algorithm is seen in Figure 4.14. The 3D unstructured generated mesh is based on the existing structured 2D meshes extruded in the z direction. The resulting blanked cells from the algorithm are shown in Figure 4.15. However, a pitfall of the present implementation of the X-ray algorithm can be seen on the slat in Figure 4.16. Cells highlighted in red have been improperly blanked when they shouldn't have. The reason behind this is because of the elevation detection algorithm checks if a point falls between an "In" and "Out" piercing point to determine if the cell is blanked or not. In this case, with the sharp concave leading edge of the slat, a possible scenario is that only one piercing point is found on the surface of the slat around the leading edge instead of 2 which causes the misidentified red cells to be tagged as blanked. A possible fix to this is to rerun the algorithm with the starting orientation of the rays changed from a xz plane to a zy plane in order to minimize rays passing through the sharp concave trailing edge.

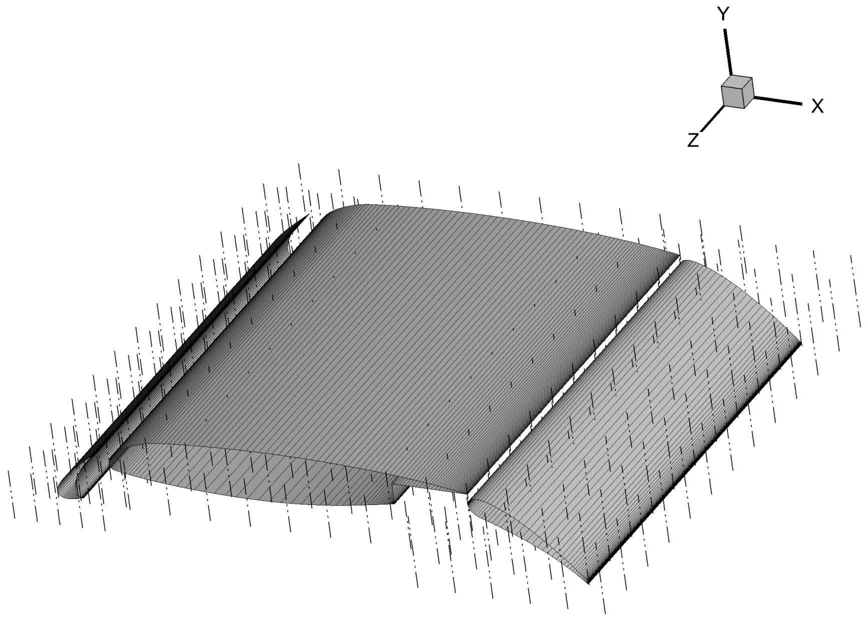


Figure 4.12 Example of the MDA airfoil with its flap and slat being pierced by the X-rays

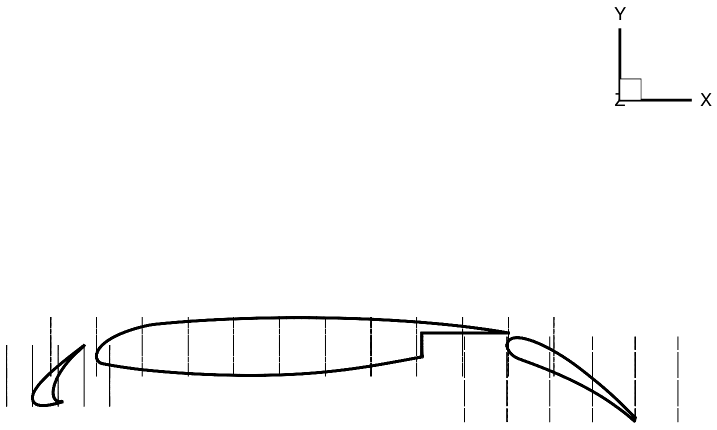


Figure 4.13 Side-view of the MDA example being pierced by the X-rays

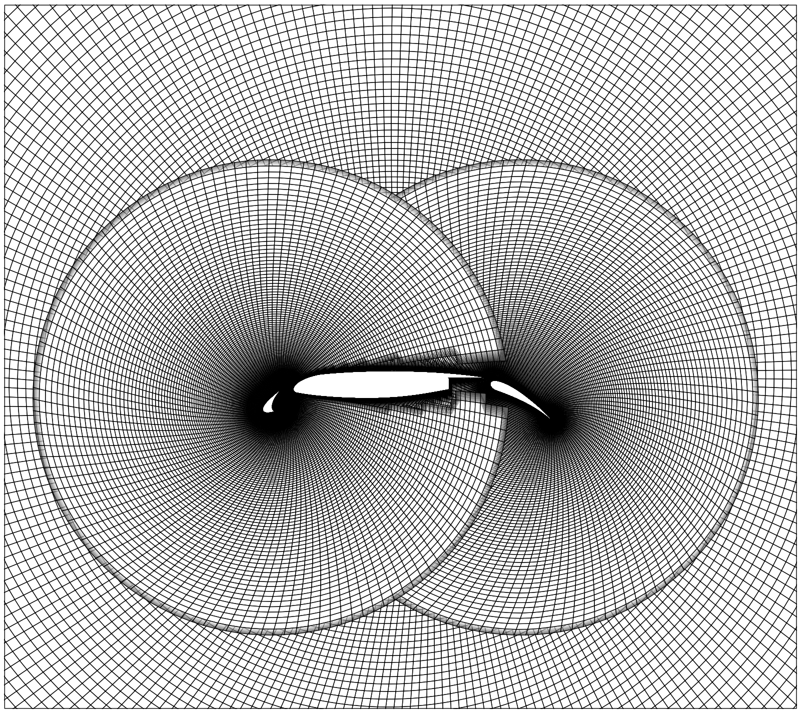


Figure 4.14 Example of the MDA airfoil 3D extruded mesh hole cut with the CHAMPS X-ray algorithm

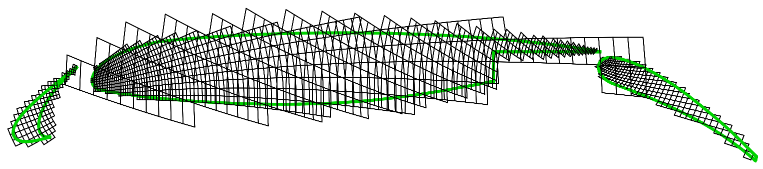


Figure 4.15 Blanked cells of the MDA airfoil 3D extruded mesh with the CHAMPS X-ray algorithm

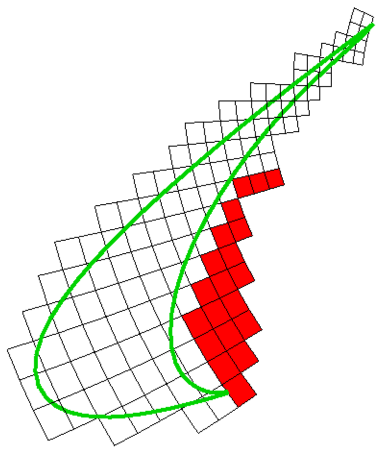


Figure 4.16 Zoom on the blanked cells of the MDA slat using CHAMPS X-ray algorithm with improper blanked cells highlighted in red

4.6 Overlap Definition and Donor Search

Once the cells from the overlapping mesh have been properly hole cut, it is time to determine which cells are going to be donor cells and which are going to be interpolated. This is done by using the previously created oct-tree boxing to quickly determine which cells come from an overlapping mesh when comparing a cell of a current mesh. This is done by finding in which sub box the cell center of the element of the current mesh being tested is in.

After the sub box is correctly identified, the element list of all the elements stored in the sub box is gone through. To find a suitable donor/interpolated cell pair, the cell center of the current element being tested against elements of an overlapping mesh must be inside one of these overlapping elements. To test this, a scalar product is done with the vectors comprised of the cell center of the current element ($\vec{f}_i$) and each of the wall facets and the normal vectors of the wall facets ($\vec{n}_i$) as described in Equation (4.10). This is also illustrated in Figure 4.17 for a 2D triangular element.

$$\begin{array}{rll} \text{if} & \vec{f}_i \cdot \vec{n}_i > 0 & \text{cell center is considered outside of the element} \\ \text{else} & \vec{f}_i \cdot \vec{n}_i =< 0 & \text{cell center is considered inside the element} \end{array} \tag{4.10}$$

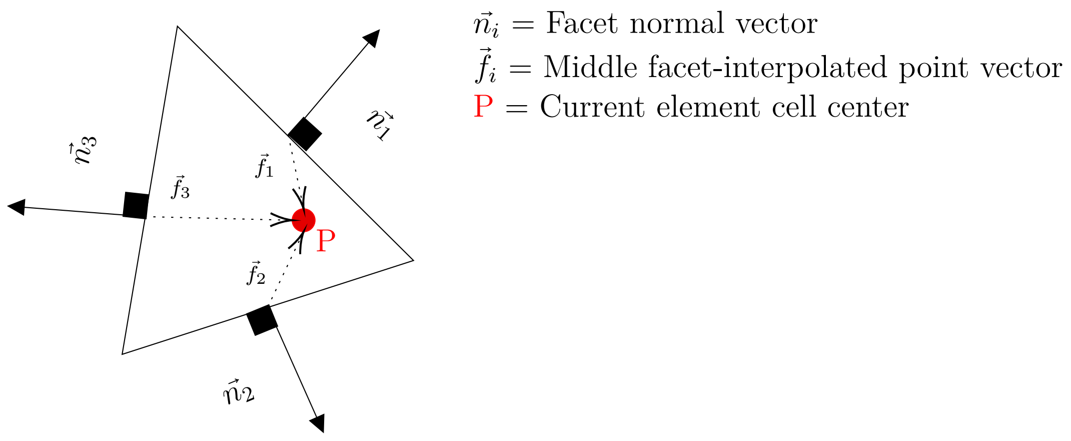

$\vec{n}_i$ = Facet normal vector
$\vec{f}_i$ = Middle facet-interpolated point vector
P = Current element cell center
$\vec{n}_1$
$\vec{n}_2$
$\vec{n}_3$
$\vec{f}_1$
$\vec{f}_2$
$\vec{f}_3$
P

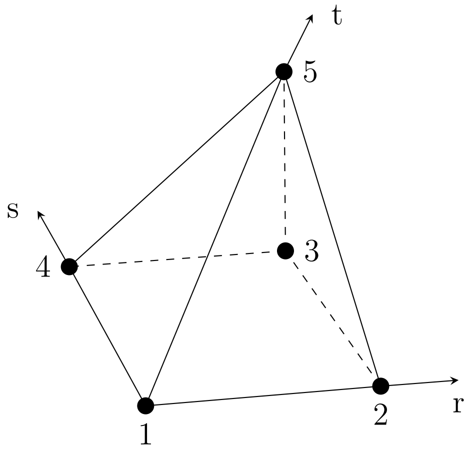
Figure 4.17 Example of the overlap scalar product around a 2D triangular element

Once the corresponding overlapping element has been found for the current element, first the hierarchy criterion is checked. If the current element has a higher hierarchy, it is considered the donor cell and the overlapping element will be the interpolated cell and vice-versa. If both cells have the same hierarchy then the wall distance criterion is used. The cell with the smallest wall distance meaning it is closer to its geometry will be the donor cell and the other cell will be the interpolated one.

After all the elements from each zone have been compared to their overlapping elements, the interpolated/donor cells are written to the CGNS file as overset interfaces with the interpolated and donor cell element indexes.

4.7 Interpolation Stencil

Once the overlap search has been done and lists of donor/element have been assembled, it is time to calculate the weights of the donor cells for the interpolated cell. For the interpolation stencil, it is taken from the CGNS standard and is a trilinear interpolation stencil [10]. The reason for this choice is the ease of use for storing the weight information in the CGNS file. For every element shape, a combination of r, s, t variables that vary from 0 to 1 representing the position of the interpolated point adimensionalized is stored in the CGNS file. With these variables as exemplified in Figure 4.18, it is possible to calculate every weight W_i for each of the element nodes.



$$W_1 = (1-r)(1-s)(1-t)$$
$$W_2 = r(1-s)(1-t)$$
$$W_3 = rs(1-t)$$
$$W_4 = (1-r)s(1-t)$$
$$W_5 = t$$

Figure 4.18 CGNS standard for storing interpolation weights for the 3D pyra element. Reproduced from CGNS [10]

As for the interpolation stencil, since weights are stored at the nodes and the computational stencil is cell centered, an interpolation from the cell centered information/variables must be done to have the proper information at the nodes. In order to do so, information from the cell centers of neighbour elements sharing the nodes are interpolated to the node by inverse interpolation. Once the interpolated values have been transferred to the nodes, the interpolated value can be calculated as shown in Equation (4.11).

$$W_{interpolated} = \sum_{j=1}^{\# \, of \, nodes} W_{node_j} w_j \tag{4.11}$$

4.8 Storing Overset Data in the CGNS Format

As for storing data generated using the overset pre-processor in CHAMPS, it boils down to three main points:

- Overset interfaces between interpolated element id and its corresponding donor cell id from another zone
- Corresponding r, s, t information to calculate weights from the donor cell
- List of blanked (hole cut) elements for each zone

This is done by calling the CGNS API from inside CHAMPS and writing the above information as data arrays. A merged overset CGNS file with its overset interfaces and blanking information stored will be similar to the one presented in the CGNS tree-like structure in Figure 4.19. The overset information is present in the "Zone Grid Connectivity" node for each of the zones.

4.9 Possible Improvements to the Current Implementation

Several improvement areas have been noted for the current state of the preprocessor.

The first area of improvement is with the hole cutting process with the X-ray. As seen with the MDA example, some cells are left blanked when they shouldn't be. A reason for this is because of the problems of the algorithm with concave geometries. A possible fix highlighted is by using X-rays in different orientations to more accurately map out the geometry. In the literature, other improvements to the algorithm have been made by for example using in conjunction an oct-tree algorithm for limiting the number of tested cell facets and thus reducing the computational time of the algorithm [14].

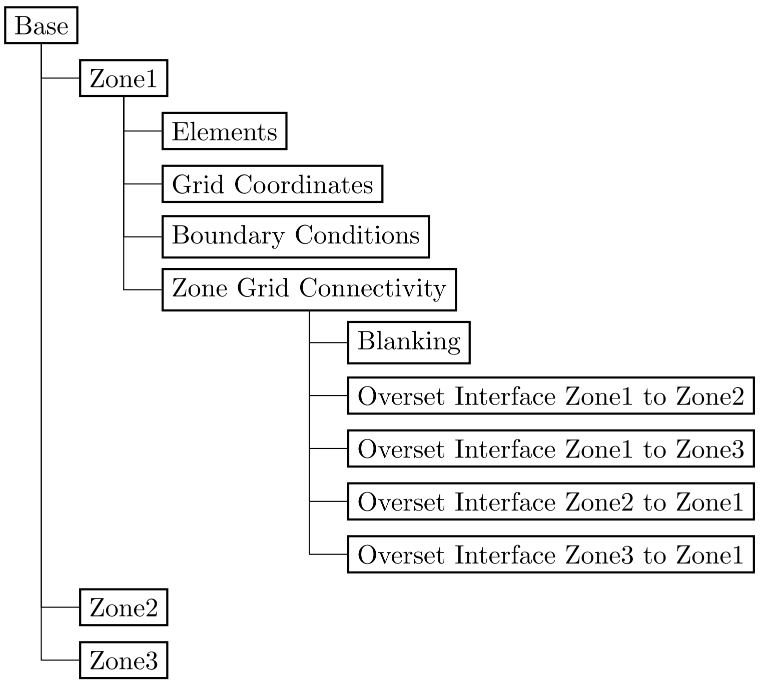

Base
Zone1
Elements
Grid Coordinates
Boundary Conditions
Zone Grid Connectivity
Blanking
Overset Interface Zone1 to Zone2
Overset Interface Zone1 to Zone3
Overset Interface Zone2 to Zone1
Overset Interface Zone3 to Zone1
Zone2
Zone3

Figure 4.19 Example of the corresponding merged CGNS mesh of the MDA airfoil schematized in the CGNS tree node structure

CHAPTER 5 CONCLUSION

As seen in the various chapters of this thesis, work was done on multiple aspects of the overset framework of Professor's Laurendeau research lab.

5.1 Summary of Works

In 2D some avenues taken have generated good results for certain cases however as it is the case with overset methods no solution is entirely perfect as it is highly dependent on the software implementation of the flow solver on which it is applied to. For the interpolation methods, it was seen that the best choice despite new methods implemented with 9-point stencils was the 4-point bilinear interpolation method with the weights calculated and stored at the cell centers. For the solid-solid discontinuity fix, it was shown that the geometric fix of recreating the halo cells via a normal projection on the surface of the geometry was appropriate for laminar cases with high curvature such as the laminar cylinder. With lower curvature geometry in laminar cases such as the laminar NACA0012, offset errors are still present but in a smaller proportion than in the cylinder case. However, as seen in the turbulent NACA0012 test case, it was shown that the solution obtained present discontinuities not seen in the laminar cases. A possible explanation for this is how the velocities derivatives are interpolated in the code. For the Schwarz correction, an implementation of the algorithm was done inside the overset preprocessor of NSCODE. However, as mentioned previously, no numerical results were presented in that section as no cases ran exhibited problems with convergence or presented numerical differences with the solution without the correction applied. In hindsight after testing numerous cases with 2D collar grids, the Schwarz correction seems more applicable for 3D uses rather than 2D uses. Hole cutting improvements were also made by adding the nodes to test on the CDT internal mesh of the geometry.

For the 3D chapter of the thesis, a presentation of the implementation coded during the course of this master's of CHAMPS 3D unstructured overset pre-processor was presented. The nature of an unstructured 3D flow solver posed some challenges in the implementation of certain 2D methods such as the hole cutting. Since NSCODE's method of hole cutting relied on Constrained Delaunay Triangulation which is hardly applicable in 3D, an implementation of the X-ray method from the literature was applied. However, this was not the case with all methods such as the recursive quad-tree boxing that was applied in 3D as a recursive oct-tree algorithm. Other methods had to be developed from scratch such as the overlap search to determine if a cell center is contained within another element. Also, since CHAMPS works with the CGNS format, methods such as the trilinear interpolation were taken from the CGNS standard for ease of use. The same applies to merging individual CGNS meshes into a single overset grid using the CGNS API.

5.2 Limitations and Future Research

As mentioned in the 2D section of the thesis, some limitations to the current approach taken to solve solid solid discontinuities were noted. The first one obviously concerns the results presented for the turbulent NACA0012 test case. The results presented highlight some of the shortcomings of the geometric fix proposed because of some unexplored problems with the algorithm. Possible preliminary culprits have been identified such as the velocity derivatives not being interpolated which could cause problems with the use of the SA turbulence model. Future work will have to be done to explore the behavior of the overset algorithm with turbulent test cases. Other areas of improvement for the 2D overset suite is in the area of hole cutting where EGADS can be used to perform more accurate hole cutting if proper support of multiple geometries is implemented. The memory management and allocation inside NSCODE could be changed since it is an incredibly rigid framework to work with as demonstrated by the number of files needed to be modified to implement a simple change of dynamically allocating instead of a hardcoded allocation of extra elements inside a physical coordinates array.

For the 3D section, current limitations are expressed by the hole cutting algorithm with concave geometries. Fixes to the method proposed are by changing how the starting plane in which the X-rays are allocated and using different orientations. Also, combining methods such as a spatial partitioning algorithm like an oct-tree boxing to limit the number of facets tested for piercing points are a worthy endeavor to pursue in order to reduce computational times.

Future work to support solid solid intersections in 3D will have to be done. Since it is an unstructured CFD code and as mentioned in the literature review, zipper grids with triangulation of overlapping meshes could be a possible avenue to pursue.

REFERENCES

[1] W. M. Chan, “Overset grid technology development at NASA Ames Research Center,” *Computers & Fluids*, vol. 38, no. 3, pp. 496–503, Mar. 2009.

[2] J. Blazek, *Computational Fluid Dynamics: Principles and Applications.* Oxford, United Kingdom: Elsevier Science & Technology, 2015.

[3] W. Chan, R. Gomez, S. Rogers, and P. Buning, “Best Practices in Overset Grid Generation,” in *32nd AIAA Fluid Dynamics Conference and Exhibit.* St. Louis, Missouri, U.S.A.: American Institute of Aeronautics and Astronautics, Jun. 2002.

[4] NASA, “Space Shuttle Discovery,” Oct. 2007.

[5] J. Guay, “Extension of the Overset Grid Preprocessor for Surface Conforming Meshes,” M.Sc.A. Thesis, École Polytechnique de Montréal, Apr. 2017.

[6] A. Jameson, W. Schmidt, and E. Turkel, “Numerical solution of the Euler equations by finite volume methods using Runge Kutta time stepping schemes,” in *14th Fluid and Plasma Dynamics Conference.* Palo Alto, California, U.S.A.: American Institute of Aeronautics and Astronautics, Jun. 1981, p. 14.

[7] M. Parenteau, S. Bourgault-Cote, F. Plante, and E. Laurendeau, “Development of Parallel CFD Applications on Distributed Memory with Chapel,” in *2020 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, May 2020, pp. 651–658.

[8] J. Bonet and J. Peraire, “An alternating digital tree (ADT) algorithm for 3D geometric searching and intersection problems,” *International Journal for Numerical Methods in Engineering*, vol. 31, no. 1, pp. 1–17, 1991.

[9] B. Roget and J. Sitaraman, “Robust and efficient overset grid assembly for partitioned unstructured meshes,” *Journal of Computational Physics*, vol. 260, pp. 1–24, Mar. 2014.

[10] CGNS Steering Comittee, “CGNS Standard Interface Data Structures,” https://cgns.github.io/CGNS_docs_current/sids/index.html.

[11] R. Haimes and M. Drela, “On The Construction of Aircraft Conceptual Geometry for High-Fidelity Analysis and Design,” in *50th AIAA Aerospace Sciences Meeting Includ-*

ing the New Horizons Forum and Aerospace Exposition. Nashville, Tennesse, U.S.A.: American Institute of Aeronautics and Astronautics, Jan. 2012, p. 21.

[12] R. Meakin, "Object X-rays for cutting holes in composite overset structured grids," in *15th AIAA Computational Fluid Dynamics Conference.* Anaheim, California, U.S.A.: American Institute of Aeronautics and Astronautics, Jun. 2001, p. 17.

[13] N. Kim and W. Chan, "Automation of Hole-Cutting for Overset Grids Using the X-rays Approach," in *20th AIAA Computational Fluid Dynamics Conference.* Honolulu, Hawaii, U.S.A.: American Institute of Aeronautics and Astronautics, Jun. 2011.

[14] R. Noack, "A Direct Cut Approach for Overset Hole Cutting," in *18th AIAA Computational Fluid Dynamics Conference.* Miami, Florida: American Institute of Aeronautics and Astronautics, Jun. 2007.

[15] Y. Lee and J. Baeder, "High-order overset method for blade vortex interaction," in *40th AIAA Aerospace Sciences Meeting & Exhibit*, ser. Aerospace Sciences Meetings. American Institute of Aeronautics and Astronautics, Jan. 2002.

[16] W. M. Chan and S. A. Pandya, "Advances in Distance-Based Hole Cuts on Overset Grids," in *22nd AIAA Computational Fluid Dynamics Conference.* Dallas, Texas, U.S.A.: American Institute of Aeronautics and Astronautics, Jun. 2015.

[17] K. Hasanzadeh Lashkajani, "Reynolds-Averaged Navier-Stokes Based Ice Accretion For Aircraft Wings," PhD Thesis, École Polytechnique de Montréal, Dec. 2015.

[18] J. L. Steger and D. S. Chaussee, "Generation of Body-Fitted Coordinates Using Hyperbolic Partial Differential Equations," *SIAM Journal on Scientific and Statistical Computing*, vol. 1, no. 4, pp. 431–437, Dec. 1980.

[19] D. W. Kinsey and T. Barth, "Description of a Hyperbolic Grid Generating Procedure for Arbitrary Two-Dimensional Bodies," Air Force Wright Aeronautical Laboratories, Wright-Patterson Air Force base, Ohio, U.S.A., Tech. Rep. AFWAL-TM-84-191, 1984.

[20] W. M. Chan and J. L. Steger, "Enhancements of a three-dimensional hyperbolic grid generation scheme," *Applied Mathematics and Computation*, vol. 51, no. 2, pp. 181–205, Oct. 1992.

[21] W. Chan and P. Buning, "Zipper grids for force and moment computation on overset grids," in *12th Computational Fluid Dynamics Conference.* San Diego, California, U.S.A.: American Institute of Aeronautics and Astronautics, Jun. 1995, pp. p.391–401.

[22] T. Schwarz, "The Overlapping Grid Technique for the Time Accurate Simulation of Rotorcraft Flows," in *31st European Rotorcraft Forum*, Florence, Italy, Sep. 2005.

[23] J. Sitaraman, "TIOGA," Aug. 2020.

[24] "OpenFOAM," Sep. 2020.

[25] C. Benoit, S. Péron, and S. Landier, "Cassiopee: A CFD pre-and post-processing tool," Tech. Rep. hal-01141585, Apr. 2015.

[26] A. Pigeon, "Développement d'une méthode d'accélération par grilles virtuelles récursives pour l'assemblage de maillages chimères," Ph.D. dissertation, École Polytechnique de Montréal, Mar. 2015.

[27] R. C. Swanson and S. Langer, "Comparison of NACA 0012 Laminar Flow Solutions: Structured and Unstructured Grid Methods," NASA Langley Research Center, Hampton, Virginia, U.S.A., Technical Memorandum NASA/TM-2016-219003, Jan. 2016.

[28] NASA, "2D NACA 0012 Airfoil Validation - SA Model Results," https://turbmodels.larc.nasa.gov/naca0012_val_sa.html.

[29] A. Bouchard, "Wall Distance evaluation via Eikonal solver for RANS applications," PhD Thesis, École Polytechnique de Montréal, Aug. 2017.

[30] D. Belk and R. Maple, "Automated assembly of structured grids for moving body problems," in *12th Computational Fluid Dynamics Conference*. San Diego, California, U.S.A.: American Institute of Aeronautics and Astronautics, Jun. 1995, pp. p.381–390.

[31] C. Ericson, *Real-Time Collision Detection*, 1st ed. CRC Press, Dec. 2004.

[32] G. Abbruzzese, "Unstructured Grid Generation Using Overset-Mesh Cutting and Single-Mesh Reconstruction," PhD Thesis, Universidad Politécnica de Madrid, 2018.

[33] J. Baeder and Y. Lee, "Implicit Hole Cutting - A New Approach to Overset Grid Connectivity," in *16th AIAA Computational Fluid Dynamics Conference*, ser. Fluid Dynamics and Co-Located Conferences. Orlando, Florida, U.S.A.: American Institute of Aeronautics and Astronautics, Jun. 2003, p. 14.

[34] J. A. Benek, J. L. Steger, F. C. Dougherty, and P. G. Buning, *Chimera: A Grid-Embedding Technique*, Apr. 1986.

[35] F. Blanc, "Méthodes numériques pour l'aéroélasticité des surfaces de contrôle des avions," PhD Thesis, Institut Supérieur de l'Aéronautique et de l'Espace, Dec. 2009.

[36] "*Cassiopée*: A CFD pre and post-processing python package," http://elsa.onera.fr/Cassiopee/.

[37] W. Chan, "Developments in Strategies and Software Tools for Overset Structured Grid Generation and Connectivity," in *20th AIAA Computational Fluid Dynamics Conference*. Honolulu, Hawaii, U.S.A.: American Institute of Aeronautics and Astronautics, Jun. 2011.

[38] W. Chan and J. Steger, "A generalized scheme for three-dimensional hyperbolic grid generation," in *10th Computational Fluid Dynamics Conference*. Honolulu, Hawaii, U.S.A.: American Institute of Aeronautics and Astronautics, Jun. 1991.

[39] I.-T. Chiu and R. Meakin, "On automating domain connectivity for overset grids," in *33rd Aerospace Sciences Meeting and Exhibit*, ser. Aerospace Sciences Meetings. American Institute of Aeronautics and Astronautics, Jan. 1995.

[40] J. G. Coder, T. H. Pulliam, and J. C. Jensen, "Contributions to HiLiftPW-3 Using Structured, Overset Grid Methods," in *2018 AIAA Aerospace Sciences Meeting*, ser. AIAA SciTech Forum. Kissimmee, Florida, U.S.A.: American Institute of Aeronautics and Astronautics, Jan. 2018.

[41] J. Q. Cordova, "Towards a theory of automated elliptic mesh generation." NASA. Langley Research Center, Software Surface Modeling and Grid Generation Steering Committee, Apr. 1992, p. 12.

[42] M. Khoshniat, G. R. Stuhne, and D. A. Steinman, "Relative Performance of Geometric Search Algorithms for Interpolating Unstructured Mesh Data," in *Medical Image Computing and Computer-Assisted Intervention - MICCAI 2003*, G. Goos, J. Hartmanis, J. van Leeuwen, R. E. Ellis, and T. M. Peters, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, vol. 2879, pp. 391–398.

[43] B. Landmann and M. Montagnac, "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, pp. 778–804, 2011.

[44] A. T. Lévesque, "Development of an Overset Structured 2D RANS/URANS Navier-Stokes Solver Using an Implicit Space and Non-Linear Frequency Domain Time Operators," M.Sc.A. Thesis, École Polytechnique de Montréal, Apr. 2015.

[45] R. Noack, "SUGGAR: A General Capability for Moving Body Overset Grid Assembly," in *17th AIAA Computational Fluid Dynamics Conference*. Toronto, Ontario, Canada: American Institute of Aeronautics and Astronautics, Jun. 2005.

[46] P. L. Roe, "Approximate Riemann solvers, parameter vectors, and difference schemes," *Journal of Computational Physics*, vol. 43, no. 2, pp. 357–372, Oct. 1981.

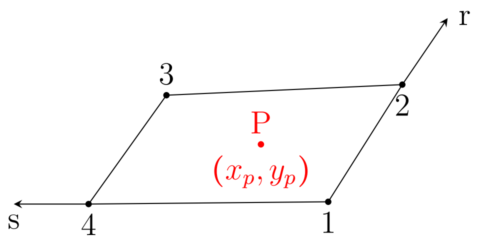
[47] S. E. Rogers, N. E. Suhs, and W. E. Dietz, "PEGASUS 5: An Automated Preprocessor for Overset-Grid Computational Fluid Dynamics," *AIAA Journal*, vol. 41, no. 6, pp. 1037–1045, Jun. 2003.

[48] F. Spiering, "Development of a Fully Automatic Chimera Hole Cutting Procedure in the DLR TAU Code," in *New Results in Numerical and Experimental Fluid Mechanics X*, ser. Notes on Numerical Fluid Mechanics and Multidisciplinary Design, A. Dillmann, G. Heller, E. Krämer, C. Wagner, and C. Breitsamter, Eds. Springer International Publishing, 2016, pp. 585–595.

APPENDIX A NSCODE - ALGORITHMS AND NUMERICAL RESULTS

NSCODE - 2D Bilinear Interpolation Method

For the 2D bilinear interpolation in NSCODE, it is implemented in the following fashion.



$$W_1 = (1-r)(1-s)$$
$$W_2 = r(1-s)$$
$$W_3 = rs$$
$$W_4 = (1-r)s$$

P = Interpolated Point

Figure A.1 Example of a quadrilateral element with its interpolation weights W_i and interpolation point "P". Reproduced from CGNS [10]

Using the schematization of a quadrilateral found in Figure A.1, the values of r & s which corresponds to the natural coordinates of the point P must be found. In order to do this, a transformation of the quadrilateral from Cartesian coordinates to natural coordinates must be done via a bilinear mapping of the quadrilateral.

The following equation system must be resolved for r & s.

$$\begin{bmatrix} x_1 & x_2 & x_3 & x_4 \\ y_1 & y_2 & y_3 & y_4 \end{bmatrix} \begin{bmatrix} W_1 \\ W_2 \\ W_3 \\ W_4 \end{bmatrix} = \begin{bmatrix} x_p \\ y_p \end{bmatrix}$$

Once developed, the following form is obtained:

$$f_1 = x_1 + r(x_2 - x_1) + s(x_4 - x_1) + rs(x_1 + x_3 - x_2 - x_4) = x_p \tag{A.1}$$

$$f_2 = y_1 + r(y_2 - y_1) + s(y_4 - y_1) + rs(y_1 + y_3 - y_2 - y_4) = y_p \tag{A.2}$$

f_1 and f_2 in Equation (A.1) and (A.2) respectively are an equation system that can be resolved iteratively using a multi-variable Newton's method which corresponds to the following as shown in Equation (A.3):

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \tag{A.3}$$

n consists of the current iteration and $n+1$ of the updated value of the next iteration. It is important to note for the Newton method to work initial guesses for the first iteration must be provided (when $n = 1$). In this case, the value of 0.5 was arbitrarily chosen ($r_{n=1} = s_{n=1} = 0.5$). In our case, the system solved consists of the following as shown in Equation (A.4).

$$\begin{bmatrix} r_{n+1} \\ s_{n+1} \end{bmatrix} = \begin{bmatrix} r_n \\ s_n \end{bmatrix} - J^{-1} \begin{bmatrix} f_1 \\ f_2 \end{bmatrix} \tag{A.4}$$

The Jacobian (J) is comprised of four terms:

$$J = \begin{bmatrix} J_{11} & J_{12} \\ J_{21} & J_{22} \end{bmatrix}$$

These terms are the partial derivatives of f_1 & f_2 by r & s:

$$\frac{\partial f_1}{\partial r} = J_{11} = (x_2 - x_1) + (x_1 + x_3 - x_2 - x_4)s \tag{A.5}$$

$$\frac{\partial f_1}{\partial s} = J_{12} = (x_4 - x_1) + (x_1 + x_3 - x_2 - x_4)r \tag{A.6}$$

$$\frac{\partial f_2}{\partial r} = J_{21} = (y_2 - y_1) + (y_1 + y_3 - y_2 - y_4)s \tag{A.7}$$

$$\frac{\partial f_2}{\partial s} = J_{22} = (y_4 - y_1) + (y_1 + y_3 - y_2 - y_4)r \tag{A.8}$$

The system described in Equation (A.4) is resolved iteratively until the difference between $|r_{n+1} - r_n| <$ 1E-12 & $|s_{n+1} - s_n| <$ 1E-12 or the max number of iterations is reached (user defined to 20).

Once the values of r & s are known for the point P, the interpolation weights can be calculated ($W_{1..4}$). It is then possible to calculate the value of the interpolated point using the values ($V_{1..4}$) stored where the weights are located as shown in Equation (A.9).

$$P_{\text{value}} = W_1V_1 + W_2V_2 + W_3V_3 + W_4V_4 \tag{A.9}$$

NSCODE - Inverse Distance Interpolation

$$dx = x_{donor} - x_{interpolated} \tag{A.10}$$

$$dy = y_{donor} - y_{interpolated} \tag{A.11}$$

$$\text{Distance} = \sqrt{dx^2 + dy^2} \tag{A.12}$$

$$\text{Weight} = \frac{1}{(\text{Distance} + \epsilon)^p} \text{ where p=2 \& } \epsilon\text{=1E-12} \tag{A.13}$$

NSCODE - Coefficient of Friction Computation in the y and z Directions

$$\tau_{yy} = \frac{1}{2}\lambda(\frac{dU_i}{dx_i} + \frac{dV_i}{dy_i} + \frac{dU_{i+1}}{dx_{i+1}} + \frac{dV_{i+1}}{dy_{i+1}}) + \mu_{total}(\frac{dV_i}{dy_i} + \frac{dV_{i+1}}{dy_{i+1}}) \tag{A.14}$$

$$\tau_{xz} = \frac{1}{2}\mu_{total}(\frac{dW_i}{dx_i} + \frac{dW_{i+1}}{dx_{i+1}}) \tag{A.15}$$

$$\tau_{yz} = \frac{1}{2}\mu_{total}(\frac{dW_i}{dy_i} + \frac{dW_{i+1}}{dy_{i+1}}) \tag{A.16}$$

$$\tau_{xy} = \frac{1}{2}\mu_{total}(\frac{dU_i}{dy_i} + \frac{dV_i}{dx_i} + \frac{dU_{i+1}}{dy_{i+1}} + \frac{dV_{i+1}}{dx_{i+1}}) \tag{A.17}$$

$$F_{f_y} = \mu(\tau_{xy}n_x + \tau_{yy}n_y)\, ds \tag{A.18}$$

$$F_{f_z} = \mu(\tau_{xz}n_x + \tau_{yz}n_y)\, ds \tag{A.19}$$

$$C_{f_y} = F_{f_y} \tag{A.20}$$

$$C_{f_z} = F_{f_x}\sin(\Lambda) - F_{f_z}\cos(\Lambda) \tag{A.21}$$

NSCODE - Solid Solid Discontinuites - Laminar Cylinder Results

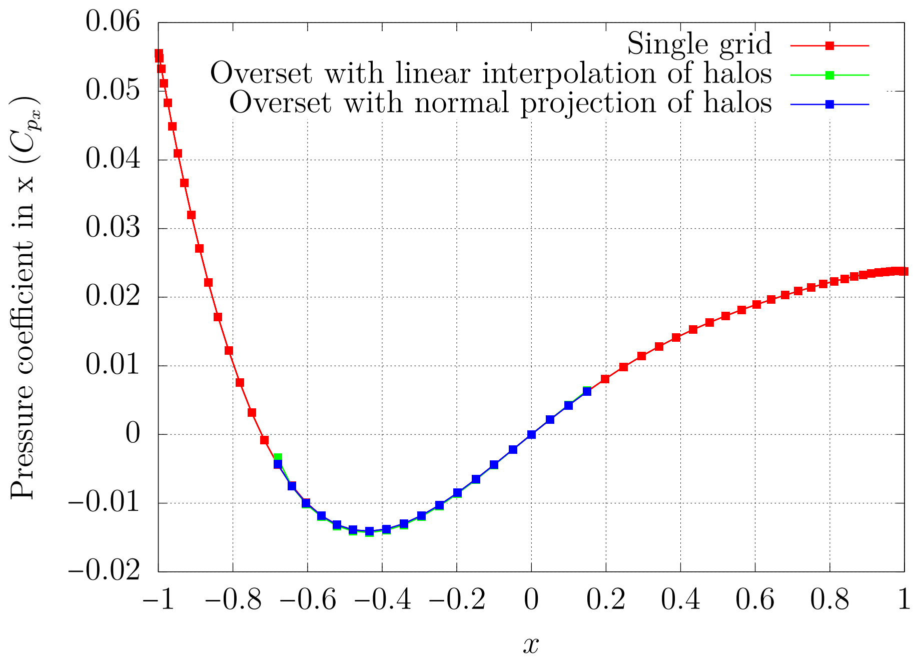


Figure A.2 Laminar cylinder pressure coefficient in x (C_{p_x})

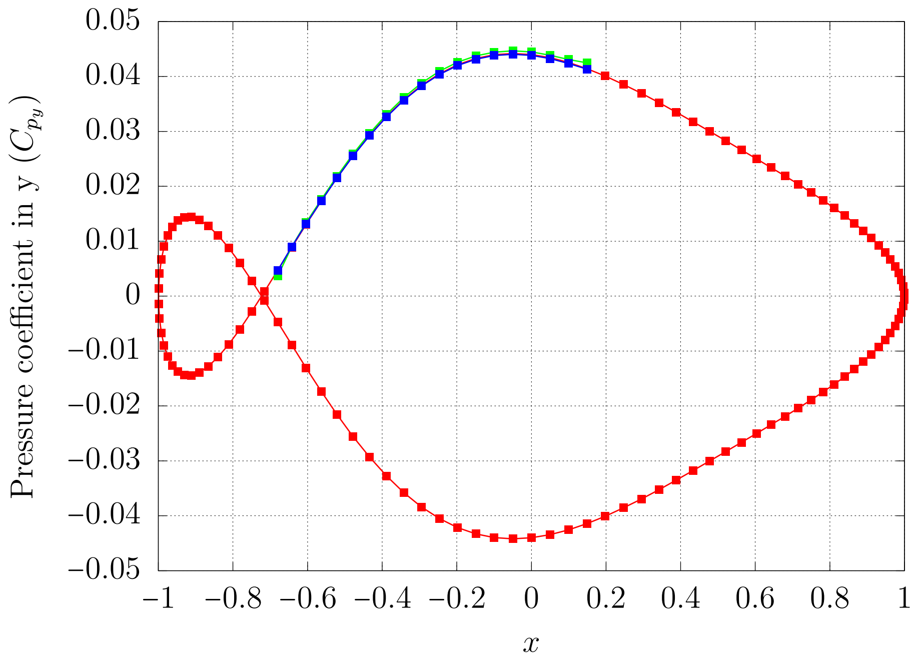


Figure A.3 Laminar cylinder pressure coefficient in y (C_{p_y})

NSCODE - Solid Solid Discontinuites - Laminar NAC0012 Results

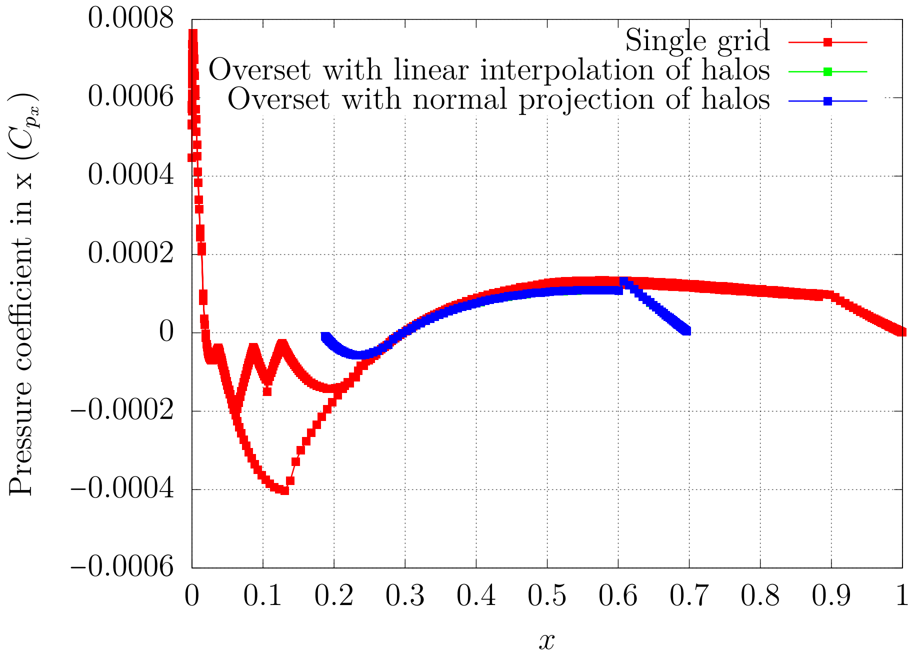


Figure A.4 Laminar NACA0012 pressure coefficient in the x direction (C_{p_x})

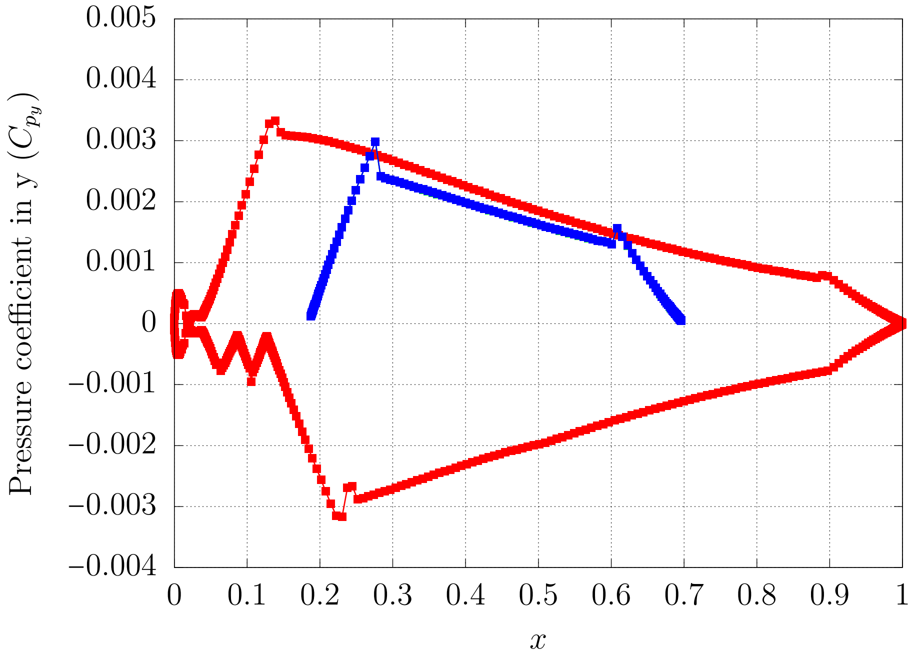


Figure A.5 Laminar NACA0012 pressure coefficient in the y direction (C_{p_y})

APPENDIX B CHAMPS - ALGORITHMS

CHAMPS - Sphere and AABB Collision Detection Algorithm

This is the the collision detection test mentioned in Section 4.4 between a bounding sphere of an element and a sub box which is an axis aligned bounding box (AABB). It was taken from Ericson (2004) [31]. The pseudo-code is written in Chapel.

```
for subBoxIndex in
currentBoxingLevel.cellCenterSizeDomain_
{
ref currentSubBox =
currentBoxingLevel.subBoxList_[subBoxIndex];

var currentSubBoxExtremums : Extremums_r =
currentBoxingLevel.getSubBoxExtremums(subBoxIndex);

var x : real_t = max(currentSubBoxExtremums.xMin_,
min(sphereXCenter,currentSubBoxExtremums.xMax_));
var y : real_t = max(currentSubBoxExtremums.yMin_,
 min(sphereYCenter,currentSubBoxExtremums.yMax_));
var z : real_t = max(currentSubBoxExtremums.zMin_,
min(sphereZCenter,currentSubBoxExtremums.zMax_));

var distanceSquare = (x - sphereXCenter)**2 +
(y - sphereYCenter)**2 + (z - sphereZCenter)**2;

if distanceSquare < sphereRadiusSquare
{
    writeln("Collision␣detected")
}
}
```

Listing B.1 Sphere and AABB collision detection algorithm

CHAMPS - Line Segment and Triangle Collision Detection Algorithm

This is the line segment and triangle collision test that is mentioned in section 4.5. It was also taken from Ericson (2004) [31]. The pseudo-code is written in Chapel.

```
proc intersectLineTriangle(q : Vec3_r, p : Vec3_r,
currentXRay : XRay_r , a : Vec3_r, b : Vec3_r, c : Vec3_r)
{
var foundFacet : bool = false;
var foundCoordinates : Vec3_r = new Vec3_r(0.0,0.0,0.0);
var pq : Vec3_r;

pq = p-q;

var pa : Vec3_r = a - p;
var pb : Vec3_r = b - p;
var pc : Vec3_r = c - p;

// Double Sided test for both
// counter clockwise and clockwise directions
var m : Vec3_r = pq^pc;
var u : real_t = pb*m;
var v : real_t = -pa*m;

if (checkSign(u) != checkSign(v))
then return (foundFacet, foundCoordinates);
var w : real_t = (pq^pb)*pa;
if (checkSign(u) != checkSign(w))
then return (foundFacet, foundCoordinates);
if (u == 0.0 && v == 0.0 && w == 0.0)
then return (foundFacet, foundCoordinates);

var denom : real_t = 1.0/(u + v + w);
u *= denom;
v *= denom;
w *= denom;
foundCoordinates = u*a + v*b + w*c;
foundFacet = true;
return (foundFacet, foundCoordinates);
}
```

Listing B.2 Line segment and triangle collision detection algorithm

CHAMPS - Line Segment and Quad Collision Detection Algorithm

This is the line segment and quad collision test that is mentioned in section 4.5. It consists of seperating the quad into two triangles and then using the triangle and line segment collision test described previously in Annex B. The pseudo-code is also written in Chapel.

```
proc quadRayIntersect(q : Vec3_r, p : Vec3_r, currentXRay : XRay_r ,
 a : Vec3_r, b : Vec3_r, c : Vec3_r, d: Vec3_r)
{

var foundFacet : bool = false;
var foundCoordinates : Vec3_r = new Vec3_r(0.0,0.0,0.0);
ref startEndPoints = currentXRay.startEndPoints_;

var pq : Vec3_r;
var pa : Vec3_r = a - p;
var pb : Vec3_r = b - p;
var pc : Vec3_r = c - p;

var m : Vec3_r = pc^pq;

var v : real_t = pa*m;
var results : (bool, Vec3_r);
if (v >= 0.0)
then results = intersectLineTriangle(q, p, currentXRay, a, b, c);
else
then results = intersectLineTriangle(q, p, currentXRay, d, a, c);
return results;

}
```

Listing B.3 Line segment and quad collision detection algorithm