| | |
|---|---|
| **Titre:** Title: | Wall Distance Evaluation Via Eikonal Solver for RANS Applications |
| **Auteur:** Author: | Anthony Bouchard |
| **Date:** | 2017 |
| **Type:** | Mémoire ou thèse / Dissertation or Thesis |
| **Référence:** Citation: | Bouchard, A. (2017). Wall Distance Evaluation Via Eikonal Solver for RANS Applications [Mémoire de maîtrise, École Polytechnique de Montréal]. PolyPublie. https://publications.polymtl.ca/2825/ |

UNIVERSITÉ DE MONTRÉAL

# WALL DISTANCE EVALUATION VIA EIKONAL SOLVER FOR RANS APPLICATIONS

ANTHONY BOUCHARD
DÉPARTEMENT DE GÉNIE MÉCANIQUE
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

MÉMOIRE PRÉSENTÉ EN VUE DE L'OBTENTION
DU DIPLÔME DE MAÎTRISE ÈS SCIENCES APPLIQUÉES
(GÉNIE AÉROSPATIAL)
AOÛT 2017

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Ce mémoire intitulé :

WALL DISTANCE EVALUATION VIA EIKONAL SOLVER FOR RANS APPLICATIONS

présenté par : BOUCHARD Anthony
en vue de l'obtention du diplôme de: Maîtrise ès sciences appliquées
a été dûment accepté par le jury d'examen constitué de :

M. GUIBAULT François, Ph. D., président
M. LAURENDEAU Éric, Ph. D., membre et directeur de recherche
M. CASTONGUAY Patrice, Ph. D., membre

# ACKNOWLEDGMENTS

Sincere thanks to my research supervisor, Professor Éric Laurendeau, for his guidance throughout the whole project. The knowledge he shared allowed me to learn so much during my Master's studies.

This project would not have been possible without the financial support of Bombardier Aerospace, Cray Canada, the Natural Sciences and Engineering Research Council of Canada (NSERC) and the Consortium for Research and Innovation in Aerospace in Québec (CRIAQ)

I also owe a debt of gratitude to all my colleagues, especially Simon Bourgault-Côté, Pierre Lavoie and Minh Tuan Nguyen, for the support and knowledge they shared as well as discussions that helped throughout the project.

I also thank the jury members for their many comments that have helped increase its quality.

Finally, I am grateful to my family for their unconditional support and love.

# RÉSUMÉ

Les logiciels de mécanique des fluides assistée par ordinateur (CFD) sont de plus en plus utilisés pour la conception d'aéronefs. L'utilisation de grappes informatiques haute performance permet d'augmenter la puissance de calcul, aux prix de modifier la structure du code. Dans les codes CFD, les équations de Navier-Stokes moyennées (plus connues sous le nom des équations RANS) sont souvent résolues. Par conséquent, les modèles de turbulence sont utilisés pour approximer les effets de la turbulence. Dans l'industrie aéronautique, le modèle Spalart-Allmaras est bien accepté. La distance à la paroi dans ce modèle, par exemple, joue un rôle clé dans l'évaluation des forces aérodynamiques. L'évaluation de ce paramètre géométrique doit alors être précis et son calcul efficace.

Avec les nouveaux développement des hardwares, un besoin se crée dans la communauté afin d'adapter les codes CFD à ceux-ci. Les algorithmes de recherche comme les distances euclidienne et projetée sont des méthodes souvent utilisées pour le calcul de la distance à la paroi et ont tendance à présenter une mauvaise scalabilité. Pour cette raison, un nouveau solveur pour la distance à la paroi doit être développé. Pour utiliser les solveurs et techniques d'accélération déjà existantes au sein du code CFD, l'équation Eikonal, une équation aux différentielles partielles non-linéaires, a été choisie.

Dans la première partie du projet, le solveur d'équation Eikonal est développé en 2D et est résolue dans sa forme advective au centre de cellule. Les méthodes des différences finies et des volumes finis sont testées. L'équation est résolue à l'aide d'une discrétisation spatiale de premier ordre en amont. Les solveurs ont été vérifiés sur des cas canoniques, tels une plaque plane et un cylindre. Les deux méthodes de discrétisation réussissent à corriger les effets de maillages obliques et courbes. La méthode des différences finies possède un taux de convergence en maillage de deuxième ordre tandis que la méthode des volumes finis a un taux de convergence de premier ordre. L'addition d'une reconstruction linéaire de la solution à la face permet d'étendre la méthode des volumes finis à une méthode de deuxième ordre. De plus, les méthodes de différence finie et de volume fini de deuxième ordre permettent de bien représenter la distance à la paroi dans les zones de fort élargissement des cellules. L'équation Eikonal est ensuite vérifié sur plusieurs cas dont un profil NACA0012 en utilisant trois modèles de turbulence : Spalart-Allmaras, Menter SST et Menter-Langtry SST transitionnel. Pour le premier modèle, l'équation Eikonal est capable de corriger les effets de non-orthogonalité du maillage sur la viscosité turbulente ainsi que sur les coefficients aérodynamiques tandis que pour les deux autres méthodes, l'équation Eikonal donne des résultats similaires aux

distances euclidienne et projetée. Pour vérifier l'implémentation et la convergence du schéma multi-grilles, le nouveau solveur de distance à la paroi est également vérifié sur un profil en condition d'accumulation de glace. De plus, les capacités de maillages chimères du solveur de la distance à la paroi ont été vérifiées sur le profil McDonnell Douglas. Enfin, un cas 3D, le DLR-F6, est résolu pour montrer que l'équation Eikonal a été étendue pour les maillages 3D.

Dans la deuxième partie du projet, le nouveau solveur de distance à la paroi est parallélisé sur une architecture à mémoire partagée et une architecture à mémoire distribuée en utilisant OpenMP et MPI respectivement. Le code montre de bonnes performances parallèles en utilisant OpenMP. Cependant, lors de l'utilisation de MPI, le calcul devient plus lent pour deux processeurs par rapport à un. Par la suite, l'efficacité semble devenir linéaire lors de l'ajout de plus de processeurs. Cela peut être dû à une mauvaise répartition des blocs pour les processeurs.

Le nouveau solveur de distance à la paroi est limité par certains aspects. En effet, la performance de l'équation Eikonal dépend du code dans lequel elle est implémentée. Par exemple, le fractionnement et la distribution des blocs du pré-processeur ne semblent pas optimisée. De plus, le schéma d'avancement en temps est limité à ceux qui peuvent être parallélisés. Enfin, le solveur de distance à la paroi est également limité à la vraie distance pour l'utilisation sur des maillages chimère. L'utilisation d'un terme diffusif supplémentaire reste donc à étudier pour corriger les mauvaises évaluations de la turbulence près des zones convexes et concaves.

# ABSTRACT

Computational fluid dynamics (CFD) software is being used more often nowadays in aircraft design. The use of high performance computing clusters can increase computing power, but requires change in the structure of the software. In the aeronautical industry, CFD codes are often used to solve the Reynolds-Averaged Navier-Stokes (RANS) equations, and turbulence models are frequently used to approximate turbulent effects on flow. The Spalart-Allmaras turbulence model is widely accepted in the industry. In this model, wall distance plays a key role in the evaluation of aerodynamic forces. Therefore calculation of this geometric parameter needs to be accurate and efficient.

With new developments in computing hardware, there is a need to adapt CFD codes. Search algorithms such as Euclidean and projected distance are often the methods used for computation of wall distance but tend to exhibit poor scalability. For this reason, a new wall distance solver is developed here using the Eikonal equation, a non-linear partial differential equation, chosen to make use of existing solvers and acceleration techniques in RANS solvers.

In the first part of the project, the Eikonal equation solver was developed in 2D and solved in its advective form at the cell center. Both finite difference and finite volume methods were tested. The Eikonal equation was also solved using a first-order upwind spatial discretization. The solvers were verified through canonical cases like a flat plate and a cylinder. Both methods were able to correct the effects of skewed and curved meshes. The finite difference method converged at a second-order rate in space while the finite volume method converged at a first-order rate. The addition of a linear reconstruction of the solution at the face extended the finite volume method to a second-order method. Moreover, both finite difference and second-order finite volume methods were well represented by wall distance in zones of strong cell growth. The finite difference method was chosen, as it required less computing time. The Eikonal equation was then verified for several cases including a NACA0012 using three turbulence models: Spalart-Allmaras, Menter's SST and Menter-Langtry transitional SST. For the first model, the Eikonal equation was able to correct grid skewness on the turbulent viscosity as well as on the aerodynamic coefficients, while for the other two yielded results similar to Euclidean and projected distance. To verify the implementation and convergence of the multi-grid scheme, the new wall distance solver was tested on an ice-accreted airfoil. In addition, the overset grid capabilities of the wall distance solver were verified on the McDonnell Douglas airfoil. Finally, the DLR-F6, a 3D case, was solved to show that the Eikonal equation can be extended to 3D meshes.

In the second part of the project, the new wall distance solver was parallelized on shared memory and distributed memory architectures using OpenMP and MPI respectively. Good parallel performance was obtained using OpenMP. However, with MPI the computation was slower for two processors compared to one. However, when more central processing units (CPUs) were added the efficiency became linear, which may be due to a poor distribution of the CPU blocks.

The new wall distance solver was limited in some aspects. Indeed, the performance of the Eikonal equation was very dependent on the code in which it was implemented. For example, the block splitting and distribution of the pre-processor did not seem to be optimized. The time advancement scheme was limited to only those that were parallelized. Finally, the wall distance solver was limited to the real distance on overset grids, suggesting an additional diffusive term to correct turbulence defects near convex and concave zones may still need to be considered.

# TABLE OF CONTENTS

# LIST OF TABLES

# LIST OF FIGURES

# LIST OF ACRONYMS AND ABREVIATIONS

| | |
|---|---|
| 2D | Two-Dimensional |
| 3D | Three-Dimensional |
| CFD | Computational Fluid Dynamics |
| CFL | Courant-Friedrichs-Lewy number |
| CPU | Central Processing Unit |
| CUDA | Compute Unified Device Architecture |
| DPLUR | Data-Parallel Lower-Upper Relaxation |
| FANSC | Full-Aircraft-Navier-Stokes-Code |
| GPU | Graphics Processing Unit |
| GPGPU | General-Purpose Programming on Graphics Processing Unit |
| HPC | High Performance Computing |
| LU-Jacobi | Lower-Upper Jacobi |
| LU-SGS | Lower-Upper Symmetric Gauss Seidel |
| MDA | McDonnell Douglas research Airfoil |
| MPI | Message Passing Interface |
| NS | Navier-Stokes |
| NSCODE | Navier-Stokes-Code |
| OpenMP | Open Multi-Processing |
| PDE | Partial Differential Equation |
| RANS | Reynolds Averaged Navier-Stokes |
| RK | Runge-Kutta |
| SA | Spalart-Allmaras |
| SST | Shear Stress Transport |
| URANS | Unsteady Reynolds Averaged Navier-Stokes |

# LIST OF APPENDICES

# CHAPTER 1 INTRODUCTION

More than fifty years ago, computational fluid dynamics (CFD) began to be used in aircraft design for the evaluation of aerodynamic forces. Nowadays, CFD software has a large impact on early design choices, which influences the rest of the aircraft program development process. At the end of the aircraft design phase, expensive wind tunnel and ultimately flight tests are performed to validate the results provided by CFD software. Hence the aeronautical industry, turning to accurate CFD solutions as the basis of the design process, requires a tool that will match these tests while reducing costs. Furthermore, time is always a critical criterion in a project as it can have a large impact on costs and competitiveness. Therefore, CFD software must be as efficient as possible. The desire to always develop more efficient code is leading to the study of various acceleration techniques. From mathematics to computer science, the community is considering every possible technique. With the continuing advances and developments in computing hardware to deliver more computing power, a possible strategy is to use high performance computing (HPC) clusters. A change in the structure and computing algorithms of CFD software is required to allow the most efficient use of these resources.

## 1.1 Basic Concepts

This section describes basic concepts referred to in parts of this thesis.

### 1.1.1 Reynolds-Averaged Navier-Stokes Equations

The Navier-Stokes equations (Blazek, 2005) describe the dynamical behavior of a fluid based on three conservation laws: the conservation of mass, momentum and energy and are given as:

$$\begin{aligned} \frac{\partial}{\partial t}(\rho) + \frac{\partial}{\partial x_j}(\rho u_j) &= 0 \\ \frac{\partial}{\partial t}(\rho u_i) + \frac{\partial}{\partial x_j}(u_j \rho u_i) &= -\frac{\partial}{\partial x_j}(p) + \frac{\partial}{\partial x_j}(\sigma_{ij}) + S_{ij} \\ \frac{\partial}{\partial t}(\rho E) + \frac{\partial}{\partial x_j}(u_j \rho E) &= -\frac{\partial}{\partial x_j}(p) + \frac{\partial}{\partial x_j}(\sigma_{ij} u_i) - \frac{\partial}{\partial x_j}(q_j) + + S_E \end{aligned} \tag{1.1}$$

where $\rho$, $u$, $E$ and $p$ are the density, velocity, energy and pressure respectively. $S$ contains source terms such as external forces: $(x, y, z, t)$ are the Cartesian space-time coordinates; $q_{ij}$

contains the Temperature $T$:

$$q_j = -k\frac{\partial T}{\partial x_j} \tag{1.2}$$

where $\sigma_{ij}$ is the viscous stress tensor, which is defined by:

$$\sigma_{ij} = \nabla\left(\mu \nabla U\right) \tag{1.3}$$

and where $\mu$ is the dynamic viscosity. For typical aircraft applications, the flow has an unsteady behavior which means that the flow parameters (i.e. density, velocity, temperature, etc.) vary with respect to time. Direct numerical simulation (DNS) using the Navier-Stokes equations simulates these effects. However, DNS requires very large resources. To reduce the computation cost of such simulations, the turbulence is approximated. One method is to perform a time averaging of the variables. In this case, the density becomes:

$$\rho = \overline{\rho} + \rho' \tag{1.4}$$

where $\overline{\rho}$ is the mean density and $\rho'$ is the fluctuating part. The new variables are replaced in the Navier-Stokes equations to obtain the Reynolds-Averaged Navier-Stokes (RANS) equations as given below.

$$\begin{aligned}
\frac{\partial}{\partial t}\left(\overline{\rho}\right) + \frac{\partial}{\partial x_j}\left(\overline{\rho u}_j\right) &= 0 \\
\frac{\partial}{\partial t}\left(\overline{\rho u}_i\right) + \frac{\partial}{\partial x_j}\left(\overline{u}_j \overline{\rho u}_i\right) &= -\frac{\partial}{\partial x_j}\left(\overline{p}\right) + \frac{\partial}{\partial x_j}\left(\overline{\sigma}_{ij}\right) + \frac{\partial}{\partial x_j}\left(\overline{\tau}_{ij}\right) + S_{ij} \\
\frac{\partial}{\partial t}\left(\overline{\rho E}\right) + \frac{\partial}{\partial x_j}\left(\overline{u}_j \overline{\rho H}\right) &= -\frac{\partial}{\partial x_j}\left(\overline{\rho u_j' e'} + \overline{u}_i \tau_{ij} + \frac{1}{2}\overline{\rho u_i' u_i' u_j'}\right) \\
&\quad + \frac{\partial}{\partial x_j}\left(\overline{\sigma}_{ij}\overline{u}_i + \overline{\sigma_{ij}' u_i'}\right) - \frac{\partial}{\partial x_j}\left(\kappa_T \frac{\partial \overline{T}}{\partial x_j}\right) + S_E
\end{aligned} \tag{1.5}$$

where, $\tau_{ij}$ is the Reynolds stress term:

$$\tau_{ij} = -\overline{\rho u_i' u_j'} \tag{1.6}$$

### 1.1.2 Turbulence Modeling

In the RANS environment, turbulence is generally modeled using either linear eddy viscosity models, nonlinear eddy viscosity models or Reynolds stress models (Blazek, 2005). Because of

the numerical complexity of the Reynolds stress and non-linear eddy viscosity models, linear eddy viscosity models are often used. Linear models are based on the Boussinesq assumption (the eddy viscosity hypothesis), which states that:

$$\sigma_{ij} = 2\mu_t \left[ \frac{1}{2} \left( \frac{\partial u_i}{\partial x_i} + \frac{\partial u_j}{\partial x_j} \right) - \frac{1}{3} \frac{\partial u_k}{\partial x_k} \delta_{ij} \right] - \frac{2}{3} \rho k \delta_{ij} \tag{1.7}$$

where $\mu_t$ is the turbulent viscosity and $\delta$ is the Kronecker delta. In RANS equations, the viscous tensor then becomes:

$$\sigma_{ij} = \nabla \left( (\mu + \mu_t) \nabla U \right) \tag{1.8}$$

Linear eddy viscosity models can be divided into several categories: algebraic, one equation and multiple equation models. These models solve for $\mu_t$, the turbulent viscosity. Algebraic turbulence models are computed without need for additional partial differential equations (i.e. $\mu_t$ is calculated directly with the flow variables) and therefore history effects are not considered. One and multiple equation turbulence models utilize transport equations to calculate the turbulence properties of the flow and then retrieve $\mu_t$. Two widely used turbulence models, namely the Spalart-Allmaras and Menter's SST models, use the wall distance as a local parameter. The accuracy of the wall distance is important for good reproduction of the near-wall characteristics of the turbulence field.

### 1.1.3 High Performance Computing in CFD

High performance computers containing multiple central processing units (CPUs) or general-purpose graphics processing units (GPGPUs) deliver much more power than typical computers. Historically, efficient codes are parallelized so that they can run on multiple CPUs simultaneously, reducing job computation time. With advancing technology, GPUs and Intel Xeon Phi accelerators are now being used more frequently. In fact, these two highly parallel hardware architectures are optimized for parallel computing. A CPU consists of a small number of cores optimized for sequential tasks while a GPU can have thousands of cores for the computation of multiple tasks in parallel. Intel Xeon Phi accelerators are a hybrid of a CPU and GPU. The former consists of many cores (i.e. up to 72 cores) for parallel computing and has models that can also be used as a host CPU.

### 1.1.4 Overset Grids

The overset grid approach, also known as the Chimera approach, (Suhs *et al.*, 2002) is gaining popularity in the CFD community. The idea is to overlap meshes and interpolate variables in the cell overlaps, therefore removing constraints on generating individual meshes. Areas of application are wide: easily changed angle of attack of an airfoil in a wind tunnel, mesh multi-element airfoils, etc. To properly cut the meshes, multiple criteria are used for identifying the mesh hierarchy. Cell volume and wall distance are widely used for this purpose. The latter is often considered a good criterion (Levesque *et al.*, 2015). However, in order to be used for this purpose, the wall distance needs to be accurate across the entire computational domain, even distant from the walls.

## 1.2 Problematic Elements

### 1.2.1 Accuracy of the Solution

CFD analysis plays a key role in aircraft design, as previously mentioned. As time and costs are important constraints for industry, RANS equations are often used as the basis of flow solvers. Therefore, turbulence models are used to approximate the effects of turbulent phenomena in flow physics. An accurate assessment of the aerodynamic forces is crucial for design and thus the choice of turbulence model is important as it can greatly influence the flow solution. The Spalart-Allmaras one-equation model is often used in aeronautical applications (Blazek, 2005). This linear eddy viscosity model, like some others, uses wall distance as a global parameter. An inaccurate evaluation of this parameter can lead to errors in the turbulence model and can even inhibit convergence (Tucker *et al.*, 2003; Xu *et al.*, 2011). Close to walls this parameter becomes even more important as the distance term in the turbulence model gains amplitude. Turbulence modeling can also have a large impact in multi-physics problems. For example, in ice accretion conditions, the evolution of the shape of the iced airfoil is greatly affected by the presence of turbulent phenomena in the flow. The wall distance can influence other features as well. For example, overset grid problems can use the wall distance as the hole-cutting criterion, which means that this metric must be evaluated accurately over the entire domain. Incorrect hole cutting could lead to an inaccurate flow solution. Hence, the accuracy of the wall distance becomes an important element to keep in mind when choosing the computing method.

### 1.2.2 Efficiency of the Method

In aircraft design, thousands of geometries are analyzed. The wall distance is needed for every RANS analysis. In unsteady calculations, and even when using quasi-steady approaches as in aircraft icing, thousands of analyses are needed on the deforming geometries. In other words, the flow solver is run multiple times as the geometry evolves over time. Software is always being developed to provide ever more accurate solutions. Thus, knowing that a finer grid gives better results, the mesh can be modified in strategic places so that it helps to capture some of the flow physics such as shocks. These can be seen as deforming/adaptive grid problems. Usually, CFD software consists of three aspects: pre-processing, the flow solver and post-processing. During pre-processing, the wall distance, among other metrics, is evaluated once for the whole domain. However, in the case of deforming/adaptive grid problems, the wall distance must be reevaluated each time the grid is changed. Because a grid can have over a hundred million nodes to compute, as in the three dimensional case, the wall distance algorithm must be able to scale well when using large meshes, even in the case where grid metrics are only computed once. For these reasons, a fast and economical algorithm is desirable.

### 1.2.3 Parallelization of the Algorithm

From multiple CPUs to Xeon Phi accelerators to GPGPUs, HPC clusters are being used more frequently in the CFD community. To be able to make full use of these resources, algorithms within the CFD code must be developed in a way that can support and efficiently run on these types of architecture. In other words, the scalability of the algorithm itself is important when increasing the number of cores. Instinctively, the first step is to limit sequential operations in the algorithm. One may split the computation so that a greater number of CPU cores can be used at the same time. A simple parallelization strategy is to divide, as evenly as possible, the grid into multiple blocks. Each one of these blocks is then computed on a single CPU (or CPU core). Each CPU has its own local memory that can be quickly accessed. Depending on the parallelization technique and on the number of CPUs, the CPUs may need to communicate data between one another. The computing time is thus dependent on the speed of information exchange between the CPU cores and the CPUs themselves. Hence, when contemplating a method to compute wall distance, it is important to limit the exchange between blocks (i.e. CPUs) so the parallelization is the most efficient possible.

## 1.3 Objectives

The aim of this project is to adapt CFD codes to the advances in computing hardware. To do so, the data structure, preprocessing and flow solver must all be reprogrammed. In particular, the current wall distance algorithm must be entirely revised. In RANS code, two popular methods with similar algorithms, the Euclidean and projected distance, are often implemented, but which are not efficient on parallel computers. Therefore, this project has the global objective of developing a wall distance method that can be used on highly parallel hardware architectures. The 2D RANS solver is developed and tested prior to implementing the 3D RANS solver. The two specific objectives for this project are:

- Develop an Eikonal solver for calculation of wall distance and investigate the influence of the new distance field on RANS simulations.
- Evaluate different parallelization strategies for scalability.

## 1.4 Plan of Thesis

The rest of the thesis is divided into four chapters. Chapter 2 presents a literature review on the computation of wall distance for turbulence models. The most popular methods are studied. Chapter 3 details the development of the Eikonal solver and describes the ensuing implementation. The software is verified through multiple test cases and the results shown. Comparisons between the new computation method and standard methods are also presented. Chapter 4 describes the parallelization strategies used for the implementation of the new wall distance solver and presents the improvements in computation time and parallelization efficiency. Finally, chapter 5, presents conclusions, sums up the work, discusses limitations, and suggests possible future work.

# CHAPTER 2 LITERATURE REVIEW

The two objectives of this project, to develop an efficient wall distance solver and to parallelize it, are intertwined. Indeed, to find an efficient solver, it is first necessary to identify which parallelization technique to use. The literature review first presents the parallelization techniques considered and then discusses a number of wall distance algorithms.

## 2.1 Parallelization Techniques

The project originates from the need to adapt CFD codes to advances in computing hardware, such as Intel Xeon Phi accelerators. For multi-processing, two interfaces are often used as parallelization techniques: OpenMP through shared memory, and MPI through distributed memory architecture. An overview of these two different architectures is presented to provide an understanding of how these drive the choice for the solver.

### 2.1.1 Shared Memory Architecture

The use of a shared memory system is one of the simplest ways to parallelize code as it can be performed using simple OpenMP commands on a single CPU. A CPU may consist of multiple cores which can execute various tasks simultaneously while sharing the same physical memory. In this context, parallelizing the loops can accelerate the code. In CFD code, there are mainly two types of loops: loops on blocks and loops on the domain. First, grids can be split into multiple blocks. This technique, called multi-blocking, while useful for parallelization, has other applications like maintaining the quality of the mesh. Each block belongs to one CPU core so that every core shares the computation time. Information is shared through halo cells at connection boundaries between each iteration (Hathaway and Wood, 1996). Second, there are also loops on the domain, inside each block for multi-blocking, looping through every cell. However, a loop can only be parallelized if in each instance it is independent of the others.

### 2.1.2 Distributed Memory Architecture

Distributed memory architecture allows further acceleration of code. Indeed, Message Passage Interface (MPI) employs multiple nodes that allow access to more resources than shared memory architecture (Hathaway and Wood, 1996). The code is parallelized by splitting the computational domain and by giving each part to a single core or CPU. Each node, unlike

shared memory architecture, has its own local memory not accessible to others. In order to exchange data with another CPU, local algorithms are used to minimize data transfer (Jeong *et al.*, 2007). Global algorithms, which introduce distant dependencies, are not desirable. One well-proven technique is multi-blocking. Between iterations, data is transferred only through halo cells. However, this data transfer affects the speed that can be gained from MPI.

## 2.2 Wall Distance in Turbulence Models

As discussed earlier, wall distance is used in some turbulence models such as the Spalart-Allmaras and Menter SST models either for length scale or for the interface between the $k-\epsilon$ and the $k-\omega$ regions respectively. Despite the fact that these models are widely used, some argue that the distance should not be used because it breaks the locality of the Navier-Stokes equations and introduces an algebraic input to the model (Spalart, 2015). Spalart (2015) says that this argument is a "soft fallacy", arguing that wall distance can be used in turbulence models. In fact, the wall distance has no effect on free shear flows and has a negative power (i.e. $\nu = f(\frac{1}{d^2})$) so that it only affects flow close to the walls. Another concern with wall distance is that small objects have the same influence as large objects. Furthermore, for linear eddy viscosity models, distances need to be overestimated close to convex features and underestimated close to concave features (Fares and Schröder, 2002). Spalart (2015) states that a distance function that addresses these concerns would represent significant progress for the physics of turbulence. One solution could be distance based on partial differential equations (PDE). Indeed, PDEs can smooth the result and reduce the impact of small objects. PDE-based distances are discussed in the next section.

## 2.3 Wall Distance Evaluation Methods

Wall distance calculation methods can be divided into two groups: search algorithms and differential equation based approaches. In this section, we present an overview of these methods.

### 2.3.1 Search Algorithms

Search algorithms, which are global ways of solving a problem, are often seen in CFD codes and are the easiest methods to implement. As wall distance is the minimum of every distance to the discretized geometry, many distances can be calculated through search algorithms. Here two approaches are presented: Euclidean and projected distance. Even though other

search algorithms give the exact wall distance over the domain, these two methods will be referenced throughout the thesis, as they are the methods implemented in the two software programs. In addition to these two algorithms, a method that exactly calculates the distance field is presented for comparison purposes.

The Euclidean distance approach adopted in this thesis is the easiest way of determining wall distance. It is calculated by:

$$\sqrt{(x - x_{wall})^2 + (y - y_{wall})^2 + (z - z_{wall})^2} \tag{2.1}$$

where $x$, $y$, $z$ and $x_{wall}$, $y_{wall}$, $z_{wall}$ are the coordinates of a place in the domain and the closest wall element respectively. Each wall element is taken as the center point between two adjacent wall nodes. One problem arising with the Euclidean distance method is that it does not account for grid skewness, and thus may overestimate the distance field. Close to the walls, where distance is more important, this overestimation can be relatively high. Figure 2.1 highlights the problem with this method by showing two distances to the nearest wall element leading to inaccurate solutions and discontinuities in the error over the domain where the green arrow is the analytical solution and the red arrow is the Euclidean distance.

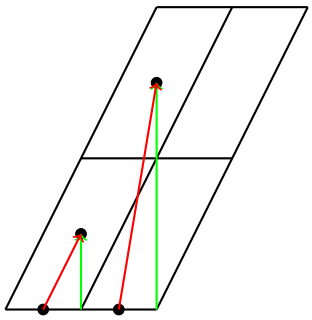


Figure 2.1 Euclidean distance discontinuity on a skewed grid

The other method is projected distance. In this thesis, projected distance is defined as the projection of the Euclidean distance on the normal direction of the closest surface element. In other words, it considers the skewness of the grid. Specifically, the method searches the nearest Euclidean distance to the wall and then computes the projected distance using:

$$(x - x_{wall})n_x + (y - y_{wall})n_y + (z - z_{wall})n_z \tag{2.2}$$

where $n_x$, $n_y$ and $n_z$ are the normals to the wall in each direction. Like the Euclidean distance it easy to implement. Figure 2.2 illustrates the correction to the Euclidean distance where the green arrow is the projected distance and the red arrow the Euclidean distance.

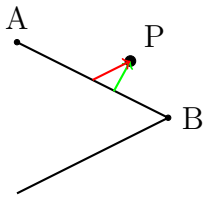



Figure 2.2 Projected distance on the trailing edge of an airfoil

Even so, projected distance introduces errors when evaluating wall distance past sharp angled edges, such as the trailing edge of an airfoil, as seen in Figure 2.3 where the red arrow is the analytical solution and the green arrow is the projected distance.

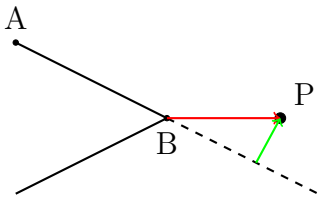



Figure 2.3 Projected distance past the trailing edge of an airfoil

A correction to the projected distance algorithm for cases as in Figure 2.3, is to limit the distance when the point is outside the segment $\overline{AB}$ to the minimum of the two Euclidean distances $\overrightarrow{AP}$ and $\overrightarrow{BP}$ (the red arrow). In this the thesis this distance is referred to as the real distance.

Search algorithms, in general, have one important disadvantage: they do not scale well when the mesh is enlarged. Indeed, they are computed in $\mathcal{O}(N_v N_s)$ operations (Roget and Sitaraman, 2013; Boger, 2001; Loehner *et al.*, 2001), where $N_v$ and $N_s$ are the number of cells and wall faces respectively. However, Wigton (1998) and Boger (2001) developed special search algorithms computed in $\mathcal{O}(N_v\sqrt{N_s})$ and $\mathcal{O}(N_v \log N_s)$ operations, respectively. Even though promising, they have been proven to be difficult to apply in complex geometric problems (Tucker, 2003). Another disadvantage of search procedures is that the algorithm is not efficient when running on parallel computers (Tucker, 2003). Indeed, the memory of each surface point must be accessible to every CPUs. In other words, all of the geometric parameters of each surface point must be allocated on each CPU.

However, seeking a method that gives the exact wall distance, that scales well and is suitable for parallel computers, Roget and Sitaraman (2013) proposed a search-based method using sphere voxelization for parallel computing. A voxel is a volume element that can be compared to a two-dimensional pixel. This method consists of limiting the number of surface elements

that are potentially nearest to the surface of a given point in the domain with a sphere voxelization algorithm. In other words, an approximated sphere radius is increased, from a given point, until intersection is detected. The minimum distance is then computed by searching through the candidate surface elements. For better efficiency, Roget and Sitaraman (2013) even rebalanced the CPU loads. This method gives promising results and is computed in $\mathcal{O}(N_v^{0.8}\sqrt{N_s})$ operations when used on a large number of CPU cores. On a DLR-F6 unstructured mesh of 1.2 million query points, the method ran 12 to 17 times faster than basic search algorithms for up to 256 cores and approximately seven times faster than their Eikonal equation solver. The latter is a PDE-based wall distance method, which is presented in the next section.

### 2.3.2 Differential Equation Based Approaches

Differential equations are another approach for determining wall distance. They present some advantages compared to other techniques. Usually, they scale well on larger meshes and are naturally efficient on parallel computers. There are three well-known differential equations used to compute wall distance: the Eikonal, Hamilton-Jacobi and Poisson equations.

The Eikonal equation computes the exact wall distance. Wall distance is defined as the distance from the wall normal direction. In other words, the distance can be seen as an advancing front with unit velocity in the direction of the wall normal. Another description is that the variation of the wall distance normal to the wall is proportional to the displacement. This is described by the equation:

$$\mid \nabla\phi \mid = 1 \tag{2.3}$$

where $\phi$ is the wall distance.

The Hamilton-Jacobi equation, a more general form of the Eikonal equation, is defined as:

$$H\left(\nabla\phi, x\right) = \epsilon\nabla^2\phi \tag{2.4}$$

where

$$H\left(\nabla\phi, x\right) = F(x) \mid \nabla\phi \mid -1 \tag{2.5}$$

In this equation, $\phi$ is the first arrival time of the front and $F(x)$ is the front propagating velocity, where $x$ is a position in the domain. To obtain the wall distance, $\phi$ is solved and $d = F(x)\phi$. The stationary Eikonal equation is obtained by setting $F(x)$ to 1 and $\epsilon$ to 0.

The Poisson equation can also be used for calculating wall distance as suggested by Tucker (2003, 1998). First, the equation below is solved.

$$\nabla^2 \phi = -1 \tag{2.6}$$

Then, the distance $d$ is recovered with the gradient of $\phi$ by:

$$d = \pm \sqrt{\sum_{j=1,3} \left( \frac{\partial \phi}{\partial x_j} \right)^2} + \sqrt{\sum_{j=1,3} \left( \frac{\partial \phi}{\partial x_j} \right)^2 + 2\phi} \tag{2.7}$$

The Poisson equation is usually computed in $\mathcal{O}(N_v \log N_v)$ operations (Tucker, 2003). Although one of the easiest differential equation to implement it only gives accurate distances close to the wall (Tucker *et al.*, 2005). The Laplacian is also often found in CFD codes, enabling re-use of some parts of the code (Tucker *et al.*, 2005).

Tucker *et al.* (2003) studied the Eikonal and Poisson equations. They note that the Eikonal equation requires upwind metrics for an accurate distance solution, even in cases of stretched and distorted grids. In addition, the Eikonal equation makes use of a Laplacian to control concave and convex features in order to correct turbulence model defects. The Poisson equation gives solutions similar to search procedures. Generally, the Poisson equation tends to overestimate distances. Furthermore, even though there is no way to control concave and convex features, the equation naturally takes these features into account.

## 2.4 Numerical Methods for Differential Equations

A variety of methods are used to solve differential equations. Instinctively, they can be transformed into hyperbolic problems and solved using a pseudo-time term in the same way steady state RANS equations are solved. This allows re-use of some parts of the flow solver and implemented acceleration techniques. Other methods have been developed especially for the resolution of the Eikonal equation and these have been extended to Hamilton-Jacobi types of equations. For the latter, there are three well-known algorithms: Fast Marching, Fast Sweeping and Fast Iterative Methods. Modified algorithms have also been developed based on these three approaches.

### 2.4.1 Time Evolving Methods

The two main differential equations that can be solved using time evolving methods are the Eikonal and Hamilton-Jacobi equations. One advantage of these differential equations is that

they can be transformed into hyperbolic-type equations by adding a pseudo-time step ($\tau$) term. Starting from the Eikonal equation,

$$| \nabla\phi | = 1 \tag{2.8}$$

and the Hamilton-Jacobi equation,

$$F(x) \mid \nabla\phi \mid = 1 + \epsilon\nabla^2\phi \tag{2.9}$$

the velocity $U = \nabla\phi$ is defined. The equations then become

$$U \cdot \nabla\phi = 1 \tag{2.10}$$

and

$$F(x) \cdot (U \cdot \nabla\phi) = 1 + \epsilon\nabla^2\phi \tag{2.11}$$

By adding the pseudo-time term, the equations are transformed into

$$\frac{\partial\phi}{\partial\tau} + U \cdot \nabla\phi = 1 \tag{2.12}$$

and

$$\frac{\partial\phi}{\partial\tau} + F(x) \cdot (U \cdot \nabla\phi) = 1 + \epsilon\nabla^2\phi \tag{2.13}$$

This allows the differential equation solver to inherit all the acceleration techniques implemented in the flow solver for the RANS equations (Tucker *et al.*, 2005). According to Tucker (2003) the solution for the Eikonal equation can be computed in $\mathcal{O}(N_v \log N_v)$ operations. However, Xia and Tucker (2010) were able to compute it almost in $\mathcal{O}(N_v)$ operations. The Eikonal equation was found to be accurate even in cases of highly stretched, non-orthogonal, and curvilinear grids when using upwind metrics in the direction of front propagation.

Fares and Schröder (2002) described a method to solve the wall distance based on the Eikonal equation by substituting $\phi$ with its inverse $\frac{1}{\phi}$. This method overcomes the following three drawbacks of Eikonal equation solvers:

- The need to have an initial solution that shows convergence in cases where there is no wall or the wall is far away.
- The need for the solution to be converged even far from the wall.

- The fact that there is no suitable values for distance other than $\phi = \infty$ when the domain is not bounded by walls.

For the Hamilton-Jacobi equation, the parameters $F(x)$ and $\epsilon$ can be changed, for example, to enhance convergence and smooth the solution. An example is the hybrid Hamilton-Jacobi-Poisson equation developed by Tucker (2011). First, the Poisson equation is solved for the distance. Then, beginning with the Poisson distances ($\phi_P$), the Hamilton-Jacobi equation is solved until convergence where the velocity is defined by:

$$U = \alpha \nabla \phi_{HJ} + (1 - \alpha) \frac{\nabla \phi_P}{\mid \nabla \phi_P \mid} \tag{2.14}$$

and $\alpha$ is a user-defined relaxation factor. The elliptic behavior of the Poisson equation has the effect of improving the robustness of convergence on poor quality grids. Furthermore, to overestimate and underestimate distances at convex and concave features respectively with the Laplacian, $\alpha$ can be space dependent.

Differential equations can be solved using finite difference methods (Tucker *et al.*, 2003; Xu *et al.*, 2011), finite volume methods (Xia and Tucker, 2010; Tucker, 2011) or finite element methods (Liu *et al.*, 2010).

### 2.4.2 Fast Marching Method

The Fast Marching Method (Zhao, 2005) was developed primarily to solve the Eikonal equation. This method consists of solving each grid point, one by one, starting from the wall using a finite difference approach. All grid points are marked either as accepted, narrow band or far away. To start, all points on the wall boundaries are marked as narrow band and all others as far away. The point with the minimum distance in the narrow band list is marked as accepted and the computed values of neighboring points are marked as narrow. If the computed value is less than the old value, it is replaced. The algorithm continues until every point is accepted. This method has a complexity of $\mathcal{O}(N_v \log N_v)$ but is not found to be efficient when run on multiple cores. The complexity is equivalent to the number of operations needed for the completion of the algorithm.

Methods have been developed to improve either the accuracy of the solution or the efficiency of the algorithm. For a more accurate solution of the Eikonal equation, Sethian *et al.* (2003) developed the Higher Accuracy Fast Marching Method which consists of approximating the gradients using a second-order difference instead of a first-order difference when the solution of neighboring points are known. Using a different approach, Danielsson and Lin (2003)

developed a modified Fast Marching Method that computes the gradients and the solution at grid cell centers in addition to grid points. By doing so, a grid point has eight neighbors instead of four on 2D structured grids. Hassouna and Farag (2007) developed the Multi-Stencils Fast Marching Method, which incorporates metric stencils for diagonal points in addition to nearest points.

The Group Marching Method was developed (Kim, 2001) to improve the efficiency of the Fast Marching Method. This algorithm computes gradients of a group of points in the narrow band simultaneously rather than sorting and computing points one by one. This reduced the complexity of the Fast Marching Method to $\mathcal{O}(N_v)$. Furthermore, Yatziv *et al.* (2006) used an untidy priority queue instead of a sorted heap, which also reduced the complexity of the algorithm to $\mathcal{O}(N_v)$.

There have also been attempts to parallelize the Fast Marching Method. Herrmann (2003) proposed a domain decomposition technique where blocks are solved simultaneously in different cores and where communication is through the halos. When a halo is accepted, all accepted points that have a value greater than the halo are rejected. Breuß *et al.* (2011) proposed a method that splits the boundaries among the threads. Tugurlan (2008) developed a distributed memory approach using a domain decomposition technique that computes the Fast Marching Method synchronously on each CPU and the halos are updated between iterations. Gillberg *et al.* (2014) also developed a parallel Fast Marching Method based on a domain decomposition technique where a sub-domain is added to the active list if its halo is changed. The active blocks are updated in parallel. A semi-ordered list can also be used. Finally, Yang and Stern (2016) proposed another parallel Fast Marching Method using a domain decomposition technique that advances the narrow-band at a specific stride in parallel depending on the front characteristic dependencies. The algorithm showed potential as it achieved a significant increase in speed, using up to 65,536 cores in some cases.

### 2.4.3 Fast Sweeping Method

The Fast Sweeping Method (Zhao, 2005) also uses a finite difference approach. It consists of performing Gauss-Seidel iterations by alternating sweeping directions. The motivation behind this approach comes from the upwind discretization of the equation. A given point's smaller neighbors influence the solution. Therefore, every sweeping direction will follow a particular upwind direction. In other words, the algorithm is completed after sweeping on $2^n$ diagonal directions (with $n$ being the number of dimensions). This method gives the exact same solution as the Fast Marching Method but has an optimal complexity of $\mathcal{O}(N_v)$. On the other hand, Gauss-Seidel iterations do not scale efficiently on parallel architectures (Jeong

and Whitaker, 2008).

Knowing that the Fast Sweeping Method updates points that cannot be calculated every iteration, Bak *et al.* (2010) adopted some improvements to decrease the computation time. The first improvement was to keep track of points that cannot be successfully computed in the iteration but have already been successfully calculated. The second improvement was to create two queues, which contain unlocked points, and define a queue cutoff. All the points in the first queue are updated in order. If the newly updated value is smaller than the old value, every greater neighbor is put in the first queue if the new value is smaller than the queue cutoff, or in the second queue if the new value is greater than the queue cutoff. When the first queue is empty, the second queue becomes the first queue and the queue cutoff changes.

Chacon and Vladimirsky (2012) developed three new methods based on the Fast Marching and Fast Sweeping Methods. The first, called the Fast Marching Sweeping Method, consists of decomposing the domain into sub-domains and running the Fast Marching Method on the coarser grid to sort the blocks from the smallest to the greatest distance. The Fast Sweeping Method is then run on each block in the predefined order. If some neighboring blocks have already been computed, the sweeping directions are changed accordingly, which produces acceleration over the simple Fast Sweeping Method. They also introduced the Heap-Cell Method that consists of a heap-sort data structure ordered by cell-values to determine the most influential cells (i.e. cells with minimum distance). The cells are updated using the Locking Sweeping Method. Finally, the accelerated version of the Heap-Cell Method, the Fast Heap-Cell Method, introduces multiple techniques to restrict sweeping directions. The authors concluded that usually, the Fast Marching Sweeping Method was the fastest of the three outperforming the Fast Marching, Fast Sweeping and Locking Sweeping Methods. Chacon and Vladimirsky (2015) further parallelized the Heap-Cell Method by decomposing the domain and computing them simultaneously. In other words, lists are created for each CPU core. For load balancing, lists are not filled according to blocks, but instead, are filled evenly.

### 2.4.4 Fast Iterative Method

The Fast Iterative Method, first developed by Jeong *et al.* (2007), is also based on a finite difference approach. This method was introduced because no other Eikonal algorithm was efficient with parallel computers. Jeong *et al.* (2007) developed an iterative algorithm that computed grid points in any order, which eliminated sorted heaps or inter-iteration dependencies (as in the case of Gauss-Seidel iterations). For this purpose, the author based his approach on a label-correcting method using Jacobi iterations. The algorithm has two steps:

initialization and updating. In the first step, the values of the boundaries are initialized and all other values set to infinity. All neighboring points to the boundaries are put in the active list. The second step consists of updating all points in the active list simultaneously. If a point converges based on its old solution (e.g. $10^{-6}$), the latter is removed from the active list. All neighbors are computed and if the new solution is smaller than the previous solution, the point is added to the active list. The updating sequence continues until the active list is empty. According to the author, this method works effectively using multi-threading programming. The method scales with a complexity of $\mathcal{O}(kN_v)$, where k depends on the input. For simple speed functions, as the Eikonal equation, $k$ is small (Hong and Jeong, 2016a).

Furthermore, to make use of the power of massively parallel hardware architectures, Jeong and Whitaker (2008) developed the Block Fast Iterative method, which is a modification of the Fast Iterative Method for a block-based algorithm. The algorithm works the same way with the only difference being that the grid is split into multiple blocks. Here, the active list contains active blocks and a block converges only if every node inside it converges. Otherwise, every node is recomputed at the next iteration. The method has also been extended for unstructured meshes (Fu *et al.*, 2013). Hong and Jeong (2016b) introduced an overlapped domain decomposition for a multi-GPU implementation. Connecting boundaries are exchanged between GPUs through halos and domain decomposition is adapted between iterations so that sub-domains have the same number of blocks. Using up to eight GPUs, the algorithm resulted in an improvement of computation time up to six times the standard GPU implementation.

Hong and Jeong (2016a) also introduced some modifications to the Fast Iterative Method in order to improve the scalability of the method for multi-core shared memory systems. They implemented a lock-free parallel algorithm using local active lists and introduced a load-balancing step for efficiency. They also developed the Group-Ordered Fast Iterative Method, which was able to better manage complex speed functions. Similar to the Block Fast Iteration Method, the domain is split into blocks. Each one is assigned a speed function, using the cell maximum, minimum or mean within the block. The Fast Iteration Method is executed on the coarser grid. Each block is then ordered by distance with blocks with values close to one another grouped together. The active list then updates the blocks simultaneously with respect to the order. In other words, at the first iteration, all the blocks that belong to order number one are in the active list, then at the second iteration, the blocks of orders number one and two will be in the list. The results show that the scalability of the first algorithm is better for simple speed functions (i.e. Eikonal equation) while the Group-Ordered Fast Iterative Method is better for complex speed functions.

## 2.5 Choice of the Wall Distance Evaluation Method

Now that a multitude of wall distance methods have been discussed, a choice has to be made. First, a summary of the various wall distance calculation methods is presented to bring into perspective their advantages and disadvantages. Based on this summary the rational for the choice of method is presented.

As the literature review has revealed, the various wall distance methods can be divided into two groups: search algorithms and differential equation based methods. For the purpose of this comparison, naive implementations (i.e. Euclidean and the projected distances) and other ones, like the sphere voxelization algorithm, will be differentiated. Among the differential equation based methods, the Poisson, Eikonal and Hamilton-Jacobi equations are the most well known for computation of wall distance. Table 2.1 presents a summary of these methods.

Table 2.1 Summary of the different wall distance calculation methods

| | Search | Sphere Voxelization | Poisson | Eikonal | Hamilton-Jacobi |
|---|---|---|---|---|---|
| True distance | Yes | Yes | No | Yes | No |
| Complexity | $\mathcal{O}(N_v N_s)$ | $\mathcal{O}(N_v^{0.8}\sqrt{N_s})$ | $\mathcal{O}(N_v \log N_v)$ | $\mathcal{O}(N_v \log N_v)$ | $\mathcal{O}(N_v \log N_v)$ |
| Parallel efficiency | No | Yes | Yes | Yes | Yes |
| Programming simplicity | Yes | No | No | No | No |

Due to the importance of wall distance in turbulence models and the use of overset grids, the wall distance computational method must be able to solve for the true wall distance over the entire domain. In other words, modified wall distance solutions, such as the Poisson and Hamilton-Jacobi equations are not good candidates. Since the algorithm will be used on highly parallel hardware architectures another concern is the parallel efficiency of the method. Differential equations are more naturally suited for this kind of task (Tucker, 2003). The sphere voxelization method can also be suitable for the computation on parallel architectures like Roget and Sitaraman (2013) showed. In addition, the wall distance solver will be used on large meshes. Differential equation approaches edge naive search algorithms in terms of complexity of the algorithm. Indeed, even though these algorithms are far simpler to program than differential equations, they are computed in $\mathcal{O}(N_v N_s)$ operations compared to $\mathcal{O}(N_v \log N_v)$ for differential equations. With the previous discussion, the sphere voxelization method and the Eikonal equation seem to be on par with one another. However, the presence of a differential equation solver in RANS solvers can make the Eikonal equation simpler to program. Given the above rationale we prefer the Eikonal equation.

Different numerical methods can be used for computation of the Eikonal equation. Four methods are studied: time evolving methods, the Fast Marching Method, the Fast Sweeping

Method and the Fast Iterative Method. Each of these methods has been developed in an efficient way on parallel computers. What drives the choice for the numerical method then becomes programming simplicity. Efficient fast marching, fast sweeping and fast iterative algorithms all need a data structure that differs from that in CFD codes. Moreover, the use of a time evolving method for the computation of the Eikonal equation allows for re-use of part of the code, including some additional numerical features that further accelerate the computation of the differential equations. For these reasons, in this project the Eikonal equation was solved using a time evolving method.

# CHAPTER 3 WALL DISTANCE SOLVER DEVELOPMENT

## 3.1 Software

The finite volume RANS solver was the framework adopted for this project, which impacted the choice of discretization strategies. In particular, the algorithms were first developed for 2D problems and then extended to 3D problems.

### 3.1.1 NSCODE

NSCODE, developed at Polytechnique Montreal (Pigeon *et al.*, 2014), is a multi-block 2D structured mesh-based RANS and URANS solver that uses the cell-centered finite volume method to discretize the spatial terms of the Navier-Stokes equations. NSCODE can utilize three different turbulence models, Baldwin-Lomax, Spalart-Allmaras and Menter's SST. NSCODE has one transitional model, the Langtry-Menter 4-equation transitional SST model, also known as the gamma-Retheta-SST model. In addition, a multi-layer ice accretion module, NSCODE-ICE (Bourgault-Cote and Laurendeau, 2015) has been added to NSCODE. Finally, NSCODE enables overset grids capability (Levesque *et al.*, 2015). Overset grids use wall distance over the entire domain to compute interpolation weights.

### 3.1.2 FANSC

FANSC is a multi-block 3D structured mesh-based RANS and URANS solver developed at Bombardier Aerospace (Cagnone *et al.*, 2011). It is also a cell-centered finite volume method for spatial terms discretization that uses the Spalart-Allmaras turbulence model, among others.

## 3.2 Eikonal Equation

As indicated in the literature review, the present study adopted the Eikonal method for computing wall distance. This methodology maximizes re-use of existing parts of the NSCODE and FANSC code. As each code is cell-centered, so too is the Eikonal equation solver.

The Eikonal equation is a non-linear partial differential equation that can be treated as a hyperbolic Hamilton-Jacobi type equation:

$$\beta \frac{\partial \phi}{\partial t} + H\left(\nabla\phi, x, \beta\right) = \epsilon \nabla^2 \phi \tag{3.1}$$

where

$$H\left(\nabla\phi, x, \beta\right) = F(x) \mid \nabla\phi \mid -(1-\beta) \tag{3.2}$$

In this equation, $\phi$ is the initial front arrival time and $F(x)$ is the front propagating velocity, where $x$ is a position in the domain. To obtain the wall distance, $\phi$ is solved and $d = F(x)\phi$. To obtain the stationary Eikonal equation, $F(x)$ was set to 1 and $\beta$ to 0, giving the following:

$$\mid \nabla\phi \mid = 1 + \epsilon\nabla^2\phi \tag{3.3}$$

In this case, the distance $d$ becomes $\phi$. Tucker (2003); Tucker *et al.* (2005) showed that the diffusion term ($\epsilon\nabla^2\phi$) is useful for controlling the front velocity. However, as we were searching for the exact wall distance we set the diffusion term to 0. The exact Eikonal equation then becomes:

$$\mid \nabla\phi \mid = 1 \tag{3.4}$$

In order to make use of the optimized and parallelizable numerical schemes already implemented in the RANS solver, the Eikonal equation was transformed into a Euler-like advection equation by squaring the equation and defining $U = \nabla\phi$, which led to:

$$U \cdot \nabla\phi = 1 \tag{3.5}$$

### 3.3 Spatial Discretization

The Eikonal equation uses two discretization methods: finite difference and finite volume.

#### 3.3.1 Finite Difference

In the finite difference method, the Eikonal equation was solved by changing the physical variables $(x, y, z)$ to a computational $(\xi, \eta, \zeta)$ domain. The advection equation was rewritten as:

$$\begin{aligned} U\frac{\partial\phi}{\partial x} + V\frac{\partial\phi}{\partial y} + W\frac{\partial\phi}{\partial z} &= 1 \\ \sum_i U_i \frac{\partial\phi}{\partial x_i} &= 1 \end{aligned} \tag{3.6}$$

where $U$, $V$ and $W$ are the $x$, $y$ and $z$-components of the gradient, respectively. Using the chain rule, the above equation was expressed in the computational domain as:

$$\begin{aligned}\sum_i U_i \frac{\partial \phi}{\partial x_i} &= \sum_i \left( U_i \xi_{x_i} \frac{\partial \phi}{\partial \xi} + U_i \eta_{x_i} \frac{\partial \phi}{\partial \eta} + U_i \eta_{x_i} \frac{\partial \phi}{\partial \zeta} \right) \\ &= \hat{U} \frac{\partial \phi}{\partial \xi} + \hat{V} \frac{\partial \phi}{\partial \eta} + \hat{W} \frac{\partial \phi}{\partial \zeta}\end{aligned} \tag{3.7}$$

where

$$\begin{aligned}\hat{U} &= U\xi_x + V\xi_y + W\xi_z \\ \hat{V} &= U\eta_x + V\eta_y + W\eta_z \\ \hat{W} &= U\zeta_x + V\zeta_y + W\zeta_z\end{aligned} \tag{3.8}$$

The gradient components $U$, $V$ and $W$ were derived from the chain rule as follows:

$$\begin{aligned}U &= \xi_x \frac{\partial \phi}{\partial \xi} + \eta_x \frac{\partial \phi}{\partial \eta} + \zeta_x \frac{\partial \phi}{\partial \zeta} \\ V &= \xi_y \frac{\partial \phi}{\partial \xi} + \eta_y \frac{\partial \phi}{\partial \eta} + \zeta_y \frac{\partial \phi}{\partial \zeta} \\ W &= \xi_z \frac{\partial \phi}{\partial \xi} + \eta_z \frac{\partial \phi}{\partial \eta} + \zeta_z \frac{\partial \phi}{\partial \zeta}\end{aligned} \tag{3.9}$$

Xu *et al.* (2011) mentioned that to obtain convergence the Eikonal equation must be discretized using an upwind method. Because the gradient must be one through the entire domain, except at local maxima, a first-order difference was accurate enough for the wall distance (Tucker, 2003; Tucker *et al.*, 2005). For precision in the evaluation of wall distance, the geometric derivatives $(\xi_x, \xi_y, \eta_x, \eta_y)$ must be upwind in order to prevent overestimation of the wall distance for stretched grids (Xu *et al.*, 2011). Using an upwind discretization with $(i, j, k)$ as the cell center they were computed as follows:

$$\begin{aligned}\Delta x_i &= x_i - x_{i-1} \quad \Delta y_i = y_i - y_{i-1} \quad \Delta z_i = z_i - z_{i-1} \\ \Delta x_j &= x_j - x_{j-1} \quad \Delta y_j = y_j - y_{j-1} \quad \Delta z_j = z_j - z_{j-1} \\ \Delta x_k &= x_k - x_{k-1} \quad \Delta y_k = y_k - y_{k-1} \quad \Delta z_k = z_k - z_{k-1}\end{aligned} \tag{3.10}$$

If the upwind direction is reversed, the difference is instead determined between $i+1$, $j+1$

or $k+1$ and $i$, $j$ or $k$ respectively. The geometric derivatives are then expressed as:

$$\begin{aligned}
\xi_x &= \frac{\Delta y_j \Delta z_k - \Delta y_k \Delta z_j}{\mid J \mid} \quad \eta_x = \frac{\Delta y_k \Delta z_i - \Delta y_i \Delta z_k}{\mid J \mid} \quad \zeta_x = \frac{\Delta y_i \Delta z_j - \Delta y_j \Delta z_i}{\mid J \mid} \\
\xi_y &= \frac{\Delta x_k \Delta z_j - \Delta x_j \Delta z_k}{\mid J \mid} \quad \eta_y = \frac{\Delta x_i \Delta z_k - \Delta x_k \Delta z_i}{\mid J \mid} \quad \zeta_y = \frac{\Delta x_j \Delta z_i - \Delta x_i \Delta z_j}{\mid J \mid} \\
\xi_z &= \frac{\Delta x_j \Delta y_k - \Delta x_k \Delta y_j}{\mid J \mid} \quad \eta_y = \frac{\Delta x_i \Delta y_k - \Delta x_i \Delta y_k}{\mid J \mid} \quad \zeta_y = \frac{\Delta x_i \Delta y_j - \Delta x_j \Delta y_i}{\mid J \mid}
\end{aligned} \tag{3.11}$$

where $\mid J \mid$ is the determinant of the Jacobian defined by:

$$\mid J \mid = \Delta x_i(\Delta y_j \Delta z_k - \Delta y_k \Delta z_j) - \Delta x_j(\Delta y_i \Delta z_k - \Delta y_k \Delta z_i) + \Delta x_k(\Delta y_i \Delta z_j - \Delta y_j \Delta z_i) \tag{3.12}$$

For the advective terms in eq. 3.7, the upwind difference in the $\xi$ direction is expressed as:

$$\hat{U}\frac{\partial \phi}{\partial \xi} = \hat{U}^+ u_{i-1} + \hat{U}^- u_{i+1} \tag{3.13}$$

where

$$\begin{aligned}
\hat{U}^{\pm} &= 0.5(\hat{U} \pm \mid \hat{U} \mid) \\
u_{i-1} &= \phi_i - \phi_{i-1} \\
u_{i+1} &= \phi_{i+1} - \phi_i
\end{aligned} \tag{3.14}$$

As suggested by Tucker *et al.* (2005), in order to match the discretization of the advective terms, a first-order upwind difference was used in eq. 3.9 for $\frac{\partial \phi}{\partial \xi}$ and $\frac{\partial \phi}{\partial \eta}$ inside of $\hat{U}$ and $\hat{V}$ respectively. The difference is shown below:

$$\frac{\partial \phi}{\partial \xi} = n_{i-1} u_{i-1} + n_{i+1} u_{i+1} \tag{3.15}$$

with

$$\begin{aligned}
n_{i-1} &= 0.25(1 + \text{sign}(u_{i-1} + u_{i+1}))(1 + \text{sign}(u_{i-1})) \\
n_{i+1} &= 0.25(1 - \text{sign}(u_{i-1} + u_{i+1}))(1 - \text{sign}(u_{i+1}))
\end{aligned} \tag{3.16}$$

where sign$(*)$ returns 1 or $-1$ if the input $(*)$ is positive or negative, respectively. At local maxima, a particular case occurs where $n_{i-1} = n_{i+1} = 0$. To facilitate convergence, it is possible to transform $n_{i-1}$ and $n_{i+1}$ into:

$$\begin{aligned} n_{i-1} &= 0.25(1 + sign(u_{i-1} + u_{i+1})) \\ n_{i+1} &= 0.25(1 - sign(u_{i-1} + u_{i+1})) \end{aligned} \tag{3.17}$$

Since updating the metrics each iteration slows the simulation, making them less stable, the metrics were initialized at the beginning and then recomputed only once when the Eikonal equation had sufficiently converged ($10^{-4}$ or $10^{-5}$). For further stability, a gradient normalization was carried out knowing that the exact gradient field must satisfy $\mid Vel \mid = \sqrt{U^2 + V^2 + W^2} = 1$:

$$\begin{aligned} U &= \frac{U}{\mid Vel \mid} \\ V &= \frac{V}{\mid Vel \mid} \\ W &= \frac{W}{\mid Vel \mid} \end{aligned} \tag{3.18}$$

Note that the entire finite difference method was developed from scratch and no part of the code was re-used.

### 3.3.2 Finite Volume

In the finite volume method, the divergence theorem was used to transform the Eikonal equation into a transport equation:

$$\nabla \cdot U\phi = 1 + \phi \nabla \cdot U \tag{3.19}$$

A control volume was defined and volume and surface integrals applied.

$$\oint_{\partial\Omega} \phi U \cdot n dA = \oint_{\Omega} [1 + \phi \nabla \cdot U] dV \tag{3.20}$$

Frolkovič *et al.* (2015) approximated the viscosity term in the volume integral as a surface integral:

$$\oint_{\Omega} \phi \nabla \cdot U dV \approx \phi \oint_{\partial\Omega} U \cdot n dA \tag{3.21}$$

yielding the following form of the Eikonal equation:

$$\oint_{\partial\Omega} \phi U \cdot n dA - \phi \oint_{\partial\Omega} U \cdot n dA = \oint_{\Omega} 1 dV \tag{3.22}$$

Using the finite volume method, a sum on every face was carried out for each cell:

$$\sum_{k}^{nface} (\phi U_n)_k \Delta A_k - \phi \sum_{k}^{nface} (U_n)_k \Delta A_k = \Delta V \tag{3.23}$$

where $\Delta A_k$ is the area of the faces $k$, $\Delta V$ is the control volume and $U_n$ is the normal velocity at the interface. As explained for the finite difference method, a first-order upwind discretization was used for the convective fluxes ($f = \phi U_n$).

$$f_{1/2} = \frac{1}{2}(f_l + f_r) - \frac{1}{2} \mid U_n \mid (\phi_r - \phi_l) \tag{3.24}$$

where $f_{1/2}$ is the flux at the face, and $f_l$ and $f_r$ are the fluxes of the left and right control volumes respectively. Also, $U_n$ is only an average of the two control volumes:

$$U_n = \frac{1}{2}(U_l + U_r) \cdot n \tag{3.25}$$

where the gradient $U$ was calculated using a Green-Gauss approach:

$$U = \frac{1}{\Delta V} \sum_{k}^{nface} \frac{1}{2}(\phi_l + \phi_r)_k \Delta A_k \tag{3.26}$$

The viscous flux was found by:

$$f_{1/2} = U_n = \frac{1}{2}(U_l + U_r) \cdot n \tag{3.27}$$

Furthermore, for stability purposes, as discussed in the finite difference approach, a gradient normalization was performed:

$$
\begin{aligned}
U &= \frac{U}{\mid Vel \mid} \\
V &= \frac{V}{\mid Vel \mid} \\
W &= \frac{W}{\mid Vel \mid}
\end{aligned}
\tag{3.28}
$$

A linear reconstruction of the cells was used to extend the method to a second-order method (Barth and Jespersen, 1989):

$$
\begin{aligned}
\phi_l &= \phi_I + \Psi_I(\nabla\phi_I \cdot \vec{r}_l) \\
\phi_r &= \phi_J + \Psi_J(\nabla\phi_J \cdot \vec{r}_r)
\end{aligned}
\tag{3.29}
$$

where $(*)_I$ and $(*)_J$ are the left and right cell-centered values respectively and $\vec{r}$ is the vector between the cell center and the face center. $\Psi$ is a limiter function as defined by Barth and Jespersen (1989), at a cell $i$, as follows:

$$
\Psi_i = \min_j \begin{cases} \min\left(1, \dfrac{\phi_{\max} - \phi_i}{\Delta_2}\right) & \text{if} \quad \Delta_2 > 0 \\ \min\left(1, \dfrac{\phi_{\min} - \phi_i}{\Delta_2}\right) & \text{if} \quad \Delta_2 < 0 \\ 1 & \text{if} \quad \Delta_2 = 0 \end{cases}
\tag{3.30}
$$

with

$$
\begin{aligned}
\Delta_2 &= \nabla\phi_i \cdot \vec{r} \\
\phi_{\max} &= \max(\phi_i, \max_j \phi_j) \\
\phi_{\min} &= \min(\phi_i, \min_j \phi_j)
\end{aligned}
\tag{3.31}
$$

where $\max_j$ and $\min_j$ are the maximum and minimum of every neighboring cell to the cell $i$. Note that the extension of the finite volume method to a second-order method was implemented with the help of M. Simon Bourgault-Côté.

While no part of the code was re-used for the finite difference method, multiple parts of the finite volume form of the Eikonal equation were slightly modified, as this form is similar to the equation in the CFD code. For example, the metrics used to compute the flow solver

were adapted for the Eikonal solver, while the convective fluxes of the flow solver were only slightly modified. Appendix A shows the two convective flux subroutines as an example of modifications that were adapted to the code.

### 3.4 Temporal Discretization

For both finite difference and finite volume methods, the system to be solved was expressed as:

$$\left(\frac{\Omega}{\Delta t} + \frac{\partial R}{\partial \phi}\right)\Delta\phi^n = -R^n \tag{3.32}$$

where $\Omega$ is the cell volume in finite volume and one in finite difference, and where $R = U \cdot \nabla\phi - 1$ computed at a pseudo-time $n$. Using an explicit solver, $\frac{\partial R}{\partial \phi}$ becomes 0:

$$\frac{\Omega}{\Delta t}\Delta\phi^n = -R^n \tag{3.33}$$

n this work, two pseudo-time advancement schemes, the explicit Runge-Kutta (RK) and Data-Parallel Lower-Upper Relaxation (DPLUR), also referred as block Jacobi were implemented. The former was used for both spatial discretization methods while the latter was developed only for the finite difference method.

Because the flow solver consists of a pseudo-time stepping scheme, the loop structure was adopted for re-use in the Eikonal solver.

#### 3.4.1 Explicit Runge-Kutta

The explicit Runge-Kutta scheme is a very simple multistage scheme. In fact, since most CFD codes use this scheme for computing the Navier-Stokes equations, the code for the RANS solvers was re-used for the Eikonal equation solver. The system to be solved is expressed as:

$$\frac{\Omega}{\Delta t}\Delta\phi^n = -R^n \tag{3.34}$$

where multiple stages are used to obtain the solution at the pseudo-time $n + 1$.

$$\begin{aligned}
\phi^{(0)} &= \phi^n \\
\phi^{(1)} &= \phi^{(0)} - \alpha_1 \frac{\Delta t}{\Omega} R^{(0)} \\
\phi^{(2)} &= \phi^{(0)} - \alpha_2 \frac{\Delta t}{\Omega} R^{(1)} \\
&\cdots \\
\phi^{n+1} &= \phi^{(m)} = \phi^{(0)} - \alpha_m \frac{\Delta t}{\Omega} R^{(m-1)}
\end{aligned} \tag{3.35}$$

where the $\alpha$'s are stage coefficients. For this solver, the same coefficient values shown in Table 3.1 were used, as suggested by Blazek (2005).

Table 3.1 Runge Kutta stage coefficients

| stages | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| $\alpha_1$ | 1.0000 | 0.5000 | 0.1481 | 0.0833 | 0.0533 |
| $\alpha_2$ | | 1.0000 | 0.4000 | 0.2069 | 0.1263 |
| $\alpha_3$ | | | 1.0000 | 0.4265 | 0.2375 |
| $\alpha_4$ | | | | 1.0000 | 0.4414 |
| $\alpha_5$ | | | | | 1.0000 |

### 3.4.2 Data-Parallel Lower-Upper Relaxation

Implicit Lower-Upper Symmetric Gauss Seidel (LU-SGS) and LU-Jacobi schemes for the Navier-Stokes solver exist in both software packages. Therefore, the codes were re-used for fast implementation of a DPLUR scheme for the Eikonal equation by changing only the lower, upper and diagonal matrices. The DPLUR scheme was used for parallelization purposes instead of the LU-SGS or the LU-Jacobi scheme. The DPLUR scheme arises from the Jacobi iteration and system was solved according to:

$$\left( \frac{\Omega}{\Delta t} + \frac{\partial R}{\partial \phi} \right) \Delta \phi^n = -R^n \tag{3.36}$$

The difference between the RK and DPLUR schemes derives from the $\frac{\partial R}{\partial \phi}$ term. This system can be seen as a simple $Ax = b$ problem where:

$$\begin{aligned} A &= \frac{\Omega}{\Delta t} + \frac{\partial R}{\partial \phi} \\ b &= -R^n \\ x &= \Delta \phi^n \end{aligned} \tag{3.37}$$

The Jacobi iteration was written as:

$$x_i^{k+1} = \frac{1}{a_{ii}} \left( b_i - \sum_{\substack{j=1 \\ j \neq i}}^{n} a_{ij} x_j^k \right) \quad i = 1, 2, ..., n \tag{3.38}$$

where $n$ is the number of cells, $i$ is the $i$-th cell of the domain (and so the $i$-th component of the vectors and matrix), $k$ the iteration and $a$ is a component of the $A$ matrix. The next step was to decompose $A$ into:

$$A = D + L + U \tag{3.39}$$

where $D$, $L$ and $U$ are the diagonal, lower and upper matrices respectively. The Jacobi iteration then becomes, where $d$, $l$ and $u$ are the components of the matrices $D$, $L$ and $U$ respectively:

$$x_i^{k+1} = \frac{1}{d_{ii}} \left( b_i - \sum_{j=1}^{n} \left( l_{ij} + u_{ij} \right) x_j^k \right) \quad i = 1, 2, ..., n \tag{3.40}$$

Knowing that the lower and upper matrices contain a number of components equivalent to the case dimension, a new Jacobi iteration of the $i$-th cell was derived as below:

$$x_i^{k+1} = \frac{1}{d_i} \left( b_i - l_i - u_i \right) \quad i = 1, 2, ..., n \tag{3.41}$$

where, in finite difference and at cell $(i, j, k)$:

$$
\begin{aligned}
x_i &= x_{ijk} = \Delta\phi_{ijk} \\
b_i &= b_{ijk} = -R_{ijk} \\
d_i &= d_{ijk} = 1 + \Delta t_{ijk}\left(\hat{U}^+_{ijk} - \hat{U}^-_{ijk} + \hat{V}^+_{ijk} - \hat{V}^-_{ijk} + \hat{W}^+_{ijk} - \hat{W}^-_{ijk}\right) \\
l_i &= l_{ijk} = -\Delta t_{ijk}\left(\hat{U}^+_{ijk}\Delta\phi_{i-1} + \hat{V}^+_{ijk}\Delta\phi_{j-1} + \hat{W}^+_{ijk}\Delta\phi_{k-1}\right) \\
u_i &= u_{ijk} = \Delta t_{ijk}\left(\hat{U}^-_{ijk}\Delta\phi_{i+1} + \hat{V}^-_{ijk}\Delta\phi_{j+1} + \hat{W}^-_{ijk}\Delta\phi_{k+1}\right)
\end{aligned}
\tag{3.42}
$$

Multiple Jacobi iterations were carried out to optimize the system before updating the solution. Four iterations were carried out with the present code:

$$
\begin{aligned}
D\Delta\phi^1 &= \Delta tR \\
D\Delta\phi^2 &= \Delta tR - (L+U)\Delta\phi^1 \\
D\Delta\phi^3 &= \Delta tR - (L+U)\Delta\phi^2 \\
D\Delta\phi^4 &= \Delta tR - (L+U)\Delta\phi^3
\end{aligned}
\tag{3.43}
$$

An under-relaxation factor $\omega$, set to 0.8 for the present algorithm, was added to help stabilize the scheme. Although no optimization studies were performed it nevertheless was found to be adequate:

$$\phi^{n+1} = \phi^n + \omega * \Delta\phi \tag{3.44}$$

## 3.5 Boundary Conditions

In CFD codes, the field is discretized and special treatments are applied at the boundaries. In NSCODE and FANSC, there are five different boundaries: solid wall, far field, symmetry, multi-block connection and overset boundaries.

### 3.5.1 Solid Wall

Wall distance was defined as the distance to the nearest point on the wall. On a solid wall, the wall distance becomes zero. The condition was expressed as a Dirichlet condition:

$$\phi = 0 \tag{3.45}$$

### 3.5.2 Far Field

At far field boundaries, to continue growth (i.e. keep the gradient constant) of the wall distance in the normal $n$ direction, the following condition was used:

$$\frac{\partial^2 \phi}{\partial n^2} = 0 \tag{3.46}$$

### 3.5.3 Symmetry

Symmetry conditions were simple to perform as the value of the halo ($\phi_h$) contained the value in the cell adjacent to the boundary ($\phi_c$).

$$\phi_h = \phi_c \tag{3.47}$$

### 3.5.4 Multi-Block Connection

To parallelize the code, multi-block capabilities were implemented in NSCODE and FANSC. To connect blocks with one another, the value in the cell adjacent to the boundary of the connecting ($\phi_{con}$) block was stored in the halo of the current block ($\phi_h$).

$$\phi_h = \phi_{con} \tag{3.48}$$

### 3.5.5 Overset Boundary

Overset grids are being used more frequently in CFD. This method simplifies the mesh generation process around a challenging geometry by cutting it into multiple pieces and overlapping them on to one other. As the overset grid method is implemented in NSCODE (Levesque *et al.*, 2015), the overset capability carries over to the Eikonal solver. The main objective of an overset grid pre-processor is to compute the weights for the interpolation of the solution at the overset boundaries. For this purpose, the code first needs to identify the overset cells according to five types (Guay, 2017):

- Computed
- Dominant
- Interpolated
- Blanked
- Buffer

Computed cells do not overlap other cells. A dominant cell overlapped by another cell is found to dominant the latter. In other words, a dominant cell will be computed. An interpolated cell is also overlapped by another cell, but is not computed because it loses to the dominant cell. A blanked cell is located inside the geometry. Finally, a buffer cell is at the interface between the computed (dominant) and the interpolated regions and is computed to ensure full coverage of the domain. Figure 3.1 depicts the overset identities for the main mesh of the McDonnell-Douglas airfoil (MDA).

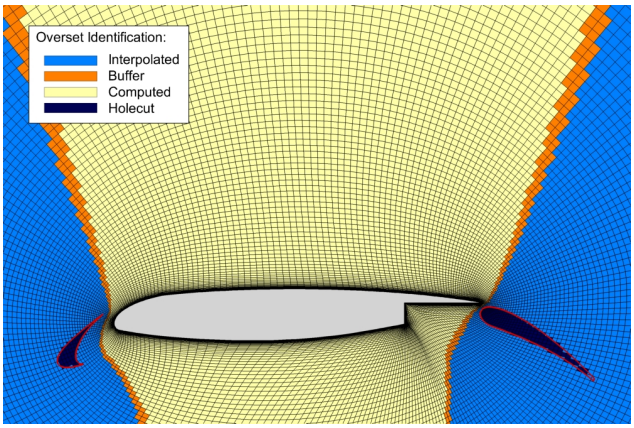



Figure 3.1 Identification of overset cells of the main element in a multi-element airfoil configuration © Guay, 2017. Reproduced with permission.

Figure 3.2 shows the search tree used to determine the overset identity of each cell. One of the criteria used to determine overset identity is the local wall distance. This value is computed independently of other meshes. In other words, the wall distance is initially solved without interacting with other meshes so as to compute the interpolation weights and complete the chimera pre-processing. In this step, the chimera boundaries were treated as far field boundaries. Subsequently, the wall distance was recomputed to obtain the global wall distance, this time taking into account the adjacent meshes (by treating the chimera boundaries as they are). This was the value used in turbulence models and represents the wall distance over the complete domain. In the second computation, the cells inside a wall were blanked, the overlapping (fringe) cells were interpolated using the weights, while the other cells were computed normally. Note that residuals of blanked and fringe cells were not taken into account in the computation of the convergence norm of the iterative solver.

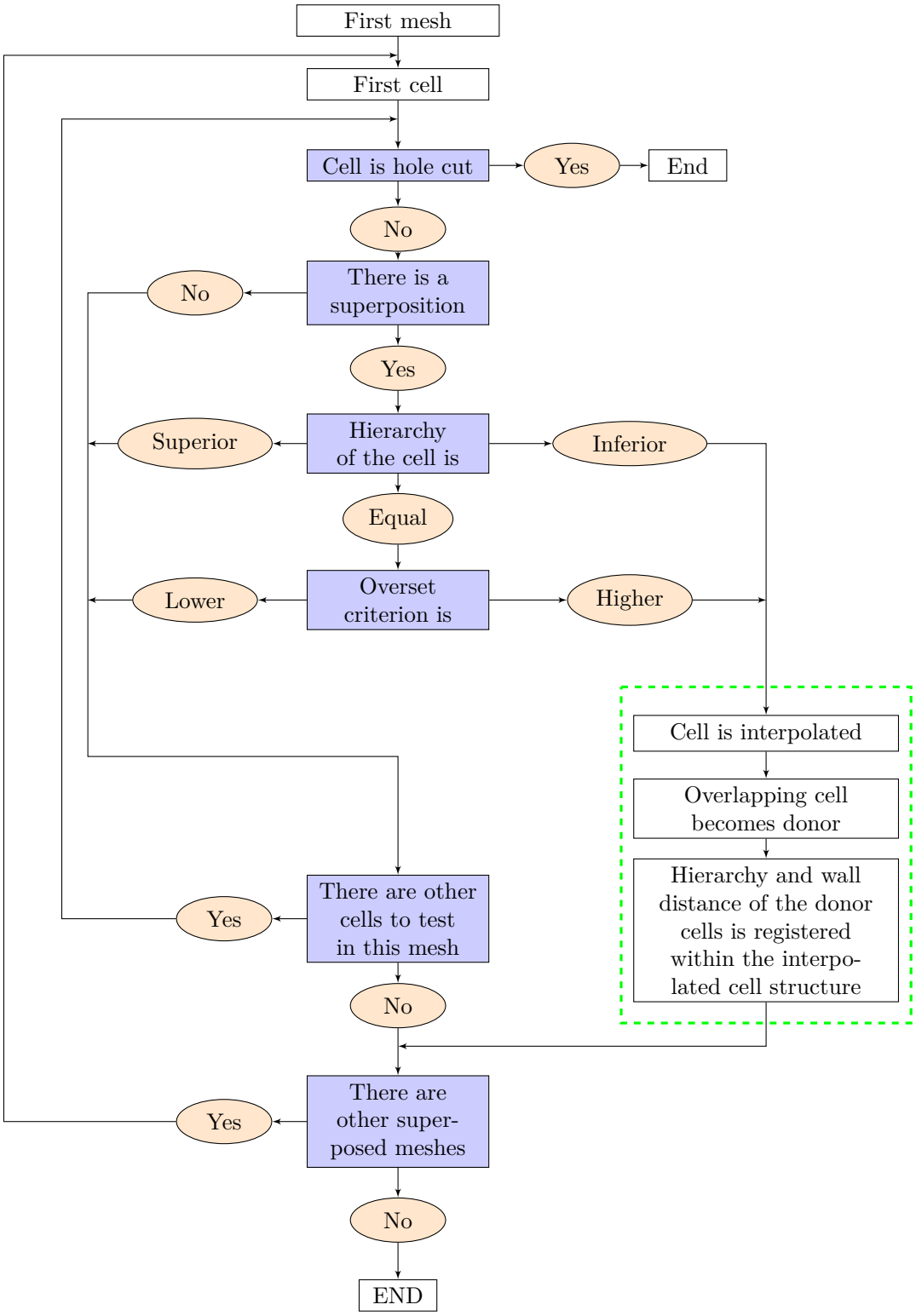



Figure 3.2 Overset identification search tree algorithm © Guay, 2017. Reproduced with permission.

## 3.6 Other Numerical Features

### 3.6.1 Initial Solution

One important but often less discussed issue for the solver is the initial solution. Indeed, the initial solution plays a large role in the convergence of the solution enhancing or slowing down convergence. Even though starting with null wall distances over the entire field can help debug the code, this is not a good initial solution as further cells take significant time to converge. Rather a good approximation of the wall distance should be found. For the Eikonal equation to converge, the initial solution must be within the range $]0, \infty[$. Xu *et al.* (2011) proposed a good starting point is the distance to the point $(0, 0, 0)$. This has proven to help converge the Eikonal equation faster than beginning with $\phi = 1$:

$$\phi_{ini} = \sqrt{x^2 + y^2 + z^2} \tag{3.49}$$

A problem arises with this initial solution when the solid geometry is far from this point. The solution was changed slightly to account for this exception. In most CFD codes for aeronautical engineering, the center of mass is given as input for the computation of the moment coefficients. Therefore we began by taking the distance to the center of mass as:

$$\phi_{ini} = \sqrt{(x - x_{cm})^2 + (y - y_{cm})^2 + (z - z_{cm})^2} \tag{3.50}$$

### 3.6.2 Convergence Criterion

As mentioned earlier, wall distance is an important parameter in some turbulence models. Knowing that values close to the walls have a greater impact on the turbulence model solution, the convergence of the Eikonal equation was altered to detect convergence of the equation sooner while maintaining a valid solution. In other words, residuals close to the wall need to have a greater impact than residuals far from it. To address this issue, multiple options were explored. Cutting the cells that have a higher distance than 10 was found to work adequately. This method is analogous to the work of Gariépy *et al.* (2011). To evaluate the convergence of the Eikonal equation, the root mean square, or the $l^2$-norm, of the residuals was performed.

$$R_{conv} = \|R\|_2 \tag{3.51}$$

### 3.6.3 Multi-Grid

Multi-grids are often used in industrial codes. Indeed, this technique is one of the most efficient to accelerate convergence. The main idea behind multi-grid techniques is to use the solution and residuals of coarser grids on the finest grid. For the Eikonal equation, the multi-grid scheme was taken from NSCODE itself, adapted from Blazek (2005). The solution to the Eikonal equation is transferred from the fine grid to the coarse grid by an averaging of the cells, resulting in 2D for readability in:

$$(\phi_{2h})_{i,j} = \frac{(\phi_h)_{i,j} + (\phi_h)_{i+1,j} + (\phi_h)_{i,j+1} + (\phi_h)_{i+1,j+1}}{4} \tag{3.52}$$

Because volume is already considered in the residuals for the Eikonal equation, the restriction operator was also defined as an averaging:

$$(R_{2h})_{i,j} = \frac{(R_h)_{i,j} + (R_h)_{i+1,j} + (R_h)_{i,j+1} + (R_h)_{i+1,j+1}}{4} \tag{3.53}$$

The prolongation of the coarse grid corrections is defined in 2D as follows:

$$(\delta\phi_h)_{i,j} = \frac{1}{16}\left[9(\delta\phi_{2h})_{i,j} + 3(\delta\phi_{2h})_{i-1,j} + 3(\delta\phi_{2h})_{i,j-1} + (\delta\phi_{2h})_{i-1,j-1}\right] \tag{3.54}$$

and in 3D as:

$$\begin{aligned}(\delta\phi_h)_{i,j,k} = \frac{1}{64}[&27(\delta\phi_{2h})_{i,j,k} + 9(\delta\phi_{2h})_{i-1,j,k} + 9(\delta\phi_{2h})_{i,j-1,k} + 9(\delta\phi_{2h})_{i,j,k-1} \\ &+ 3(\delta\phi_{2h})_{i-1,j-1,k} + 3(\delta\phi_{2h})_{i-1,j,k-1} + 3(\delta\phi_{2h})_{i,j-1,k-1} + (\delta\phi_{2h})_{i-1,j-1,k-1}]\end{aligned} \tag{3.55}$$

### 3.6.4 Local Time-Stepping

Local time-stepping was used to achieve a steady-state solution and fast convergence of the Eikonal equation. For upwind discretization, the formula used for computation of the time-step was:

$$\Delta t = \sigma \frac{\Omega}{\max(\Lambda_I, \Lambda_J, \Lambda_K)} \tag{3.56}$$

where $\sigma$ is the CFL number (usually 5.0 for the Eikonal equation), $\Omega$ the volume of the cell, and $\Lambda$ the spectral radii of the incoming convective flux Jacobians in the three grid directions:

$$\begin{aligned}\Lambda_I &= \mid \vec{U} \cdot \vec{n}^{\,I} \mid \Delta S^I \\ \Lambda_J &= \mid \vec{U} \cdot \vec{n}^{\,J} \mid \Delta S^J \\ \Lambda_K &= \mid \vec{U} \cdot \vec{n}^{\,K} \mid \Delta S^K\end{aligned} \tag{3.57}$$

Because the Eikonal equation has a unit gradient over the entire domain and a conservative time-step, Xu *et al.* (2011) proposed a modified form of the spectral radii:

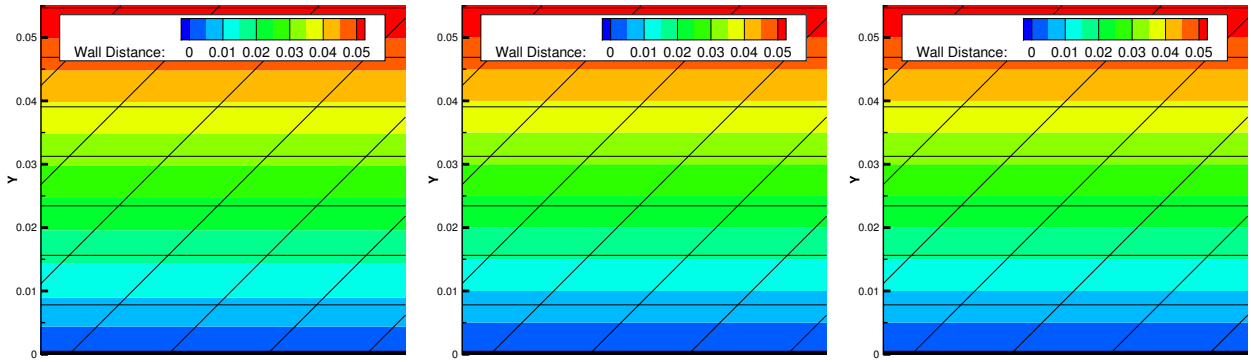
$$\begin{aligned}\Lambda_I &= \Delta S^I \\ \Lambda_J &= \Delta S^J \\ \Lambda_K &= \Delta S^K\end{aligned} \tag{3.58}$$

## 3.7 Numerical Experiments

In this section, we detail how the implementation of the Eikonal equation was verified through multiple test cases. To verify the precision of the Eikonal equation and compare the new wall distance with other traditional methods, a flat plate case with a non-orthogonal mesh to the wall was studied and compared to the analytical solution. A comparison of finite difference, and first and second-order finite volume algorithms was made on a cylinder test case. To verify that the Eikonal equation gives a better assessment of the aerodynamic coefficients than the Euclidean and projected distance method on poor quality meshes, an algebraic skewed NACA0012 grid was compared to an orthogonal grid. Then, an ice accreted geometry case with a challenging grid was explored to ensure the robustness of the code and show multi-grid capabilities. A multi-element airfoil with overset meshes was also tested to verify the algorithm with these types of cases. Finally, a DLR-F6 case was studied to ensure the validity of the extension of the algorithm to three-dimensional grids. Note that, for all the cases presented, the equations were converged to $10^{-6}$ unless stated otherwise.

### 3.7.1 Flat Plate

One of the main purposes of the Eikonal equation is to ensure precise values of wall distances close to solid walls for a poor quality mesh. To verify the precision of the algorithm, a flat plate with a mesh angle of 45 degrees at the wall was tested, using a 129x65 mesh with equidistant grid spacing. Figure 3.3 shows a close-up of the wall distance contours close to the wall.




(a) (b) (c)

Figure 3.3 Wall distance contours of a flat plate with (a) Euclidean distance, (b) projected distance and (c) the Eikonal equation

From these images, little difference was seen between the Euclidean and the other two wall distance methods. However, even though the difference was small, knowing that the analytical wall distance in this case is the height $Y$, the Eikonal and projected distances were closer to the analytical solution than the Euclidean, which overestimated wall distance close to the wall. Figure 3.4 shows the relative error of wall distance against the height $Y$ for the three methods, where the difference between the three distance methods is more obvious so as to better evaluate the magnitude of the error.

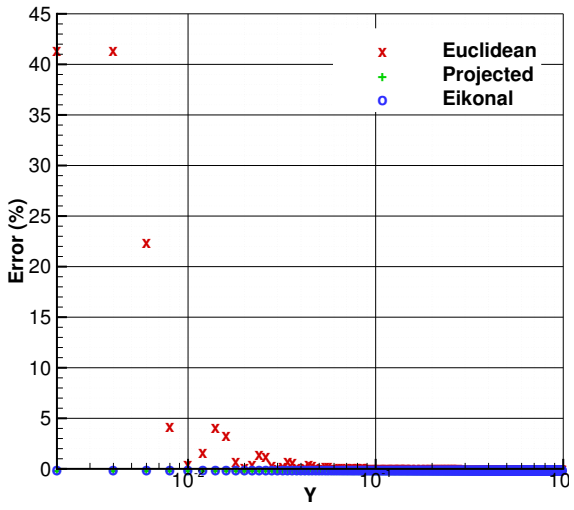



Figure 3.4 Comparison of the relative error of wall distance of Euclidean, projected and Eikonal computations on a skewed flat plate

As anticipated, the Euclidean distance yielded greater error close to the wall, which could be above 40%. The projected and Eikonal wall distance methods produced the exact solution,

in double precision, even in the case of a poor quality mesh.

### 3.7.2 Cylinder

The previous test showed that close to the wall the projected and Eikonal wall distance methods had fewer errors than the Euclidean distance. However, the mesh used did not present any curved features. Since the Euclidean and the projected distance gave the exact discretized solution on non-skewed meshes, only the Eikonal equation was run. Three discretization techniques were implemented for the Eikonal equation: finite difference, 1st-order finite volume and 2nd-order finite volume. First, a mesh convergence study was carried out on a 1.0 diameter cylinder with a stretching ratio of 1.0 to verify the accuracy of the methods. In this case, the Eikonal solutions were compared to the analytical solution. Note that error corresponds to the root mean square of the difference between the two solutions and that the equations were converged to $10^{-13}$.

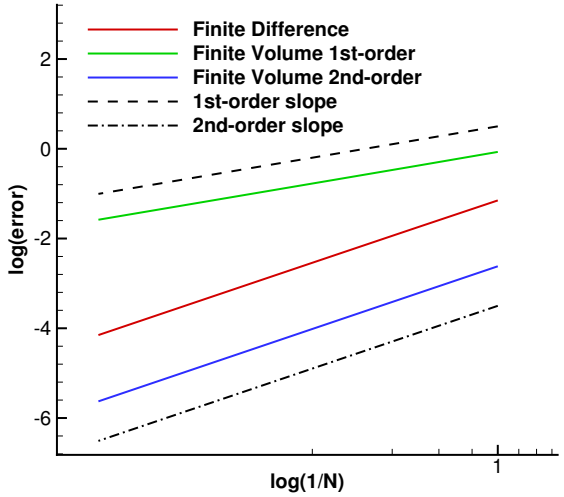



Figure 3.5 Mesh convergence of finite difference and finite volume Eikonal computations on a cylinder

As it can be seen in Figure 3.5, the 1st and 2nd-order finite volume methods had the expected accuracy. The finite difference method was shown to be 2nd-order in space. Moreover, a 129x65 1.0 diameter cylinder with a growth rate of 1.25 was used to ensure that the accuracy of the solution was not greatly affected by the grid expansion rate. Note that the solutions were compared to the Euclidean distance. Figure 3.6 shows the relative error of wall distance versus the height Y for the three cases.

The finite difference method was not greatly affected by the properties of the mesh, having only a 0.03% error compared to the Euclidean distance. On the other hand, these features,

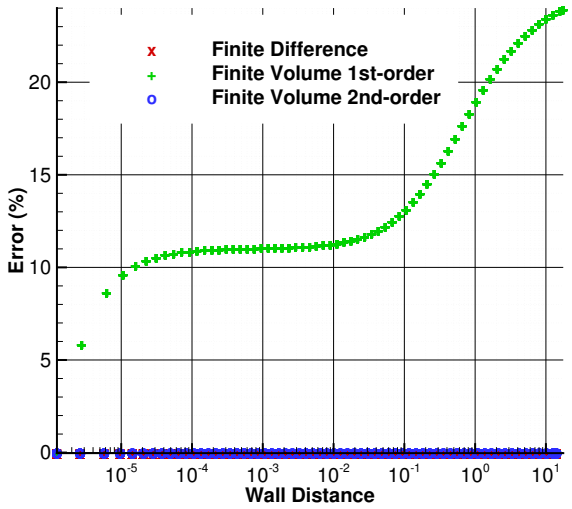



Figure 3.6 Comparison of relative error of wall distance for a cylinder with a growth rate of 1.25

showing relative errors as high as 20% for a 1.25 grid stretching case, significantly affected the first-order finite volume method. This difference may be due to the fact that the metrics were not upwind discretized in the finite volume method. To correct this behavior, a linear recomposition, with the help of a limiter function, was introduced which greatly improved the solution of the 1st-order finite volume method, yielding a solution as close to the Euclidean distance as the finite difference method.

### 3.7.3 NACA0012

Having verified that the solution to the Eikonal equation is not greatly affected by the curvature of the mesh, a NACA0012 case was explored. First, a mesh convergence study was carried out with the help of four algebraic O-grids, the largest being 2049x1025. These grids are shown in Figure 3.7.

Figure 3.8 presents the mesh convergence study on a NACA0012 airfoil. Since it was determined that the first-order finite volume method does not meet the accuracy requirements, only the finite difference and 2nd-order finite volume methods are shown in the figure. The error is, as in the case of the cylinder, the root mean square of the difference between the Eikonal solution and the real distance solution. Note that the equations were converged to $10^{-13}$.

As can be seen, unlike the results for the cylinder, none of the curves showed a defined order of convergence. Many reasons may explain this behavior, but the source could not be found. Therefore, unfortunately, this result cannot be explained. However, an accuracy as low as

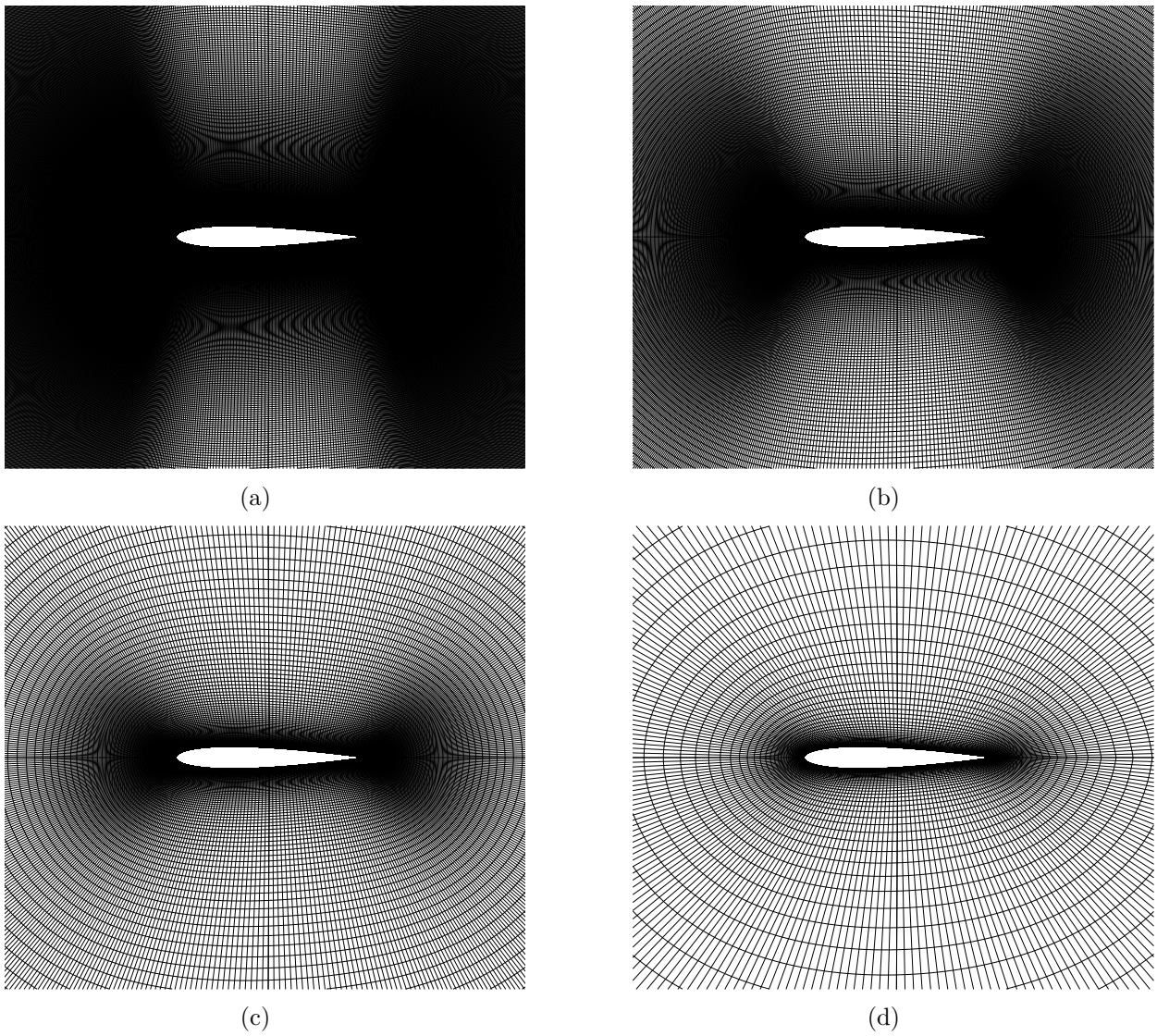


Figure 3.7 (a) 2049x1025, (b) 1025x513, (c) 513x257 and (d) 257x129 NACA0012 O-grids

$10^{-6}$ was observed on finer grids.

The computation times for the two methods were compared. On the 257x129 grid, the finite difference method required approximately three times less computing time than the finite volume method. Therefore the finite difference method was chosen for the remainder of the cases.

Simulations on orthogonal and skewed grids were run to determine how the aerodynamic coefficients were affected by the wall distance solution. To accomplish this an algebraic NACA0012 grid was constructed and smoothed for good orthogonality at the wall (Hasanzadeh *et al.*, 2015). The algebraic grid served as the skewed grid while the smoothed grid

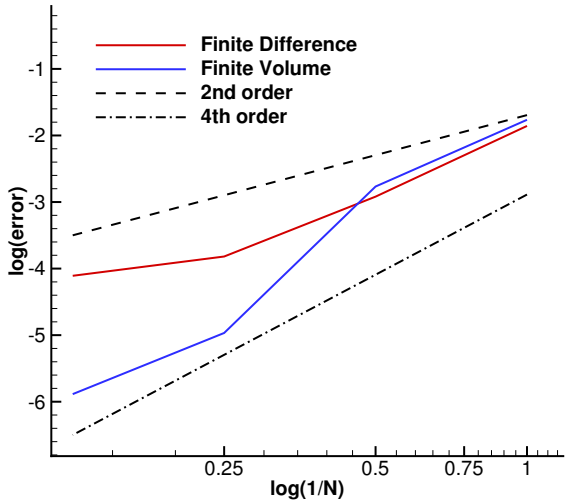
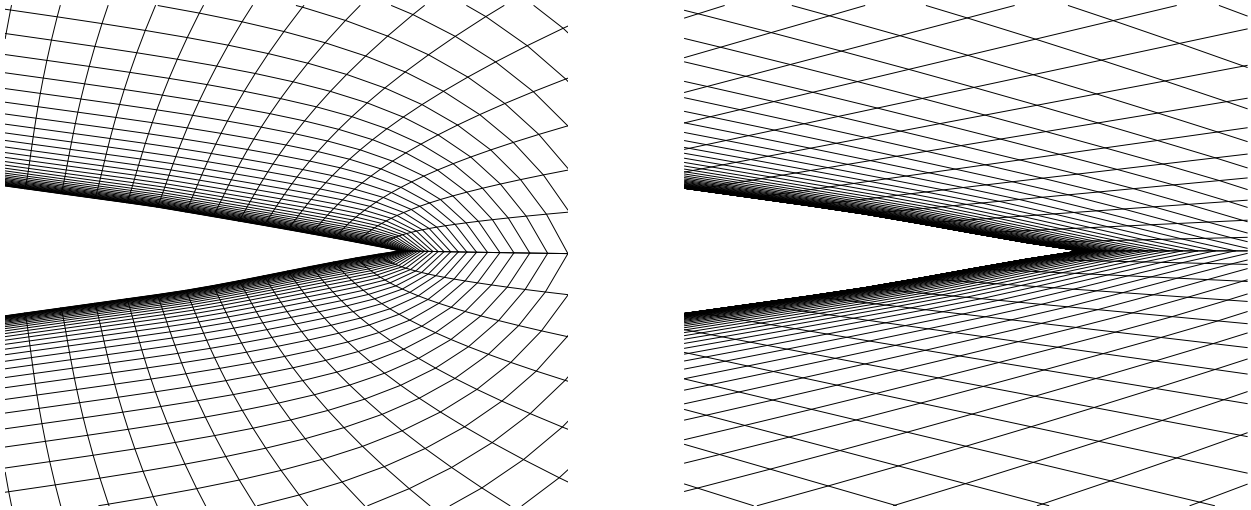



Figure 3.8 Mesh convergence of finite difference and finite volume Eikonal computations on a NACA0012 airfoil

served as the orthogonal grid. Figure 3.9 shows these two 257x129 O-grids.



(a) (b)

Figure 3.9 (a)Orthogonal and (b) skewed NACA0012 O-grids

Another grid feature that can affect the wall distance solution is the presence of a sharp element such as the wing trailing edge. Figure 3.10 shows the wall distance contours for the three methods as well as the real distance on a NACA0012 grid. As can be seen, the projected distance is affected by the sharp trailing edge, leading to a false representation of the distance beyond it.

Three turbulence models were used to determine how these different wall distance solutions

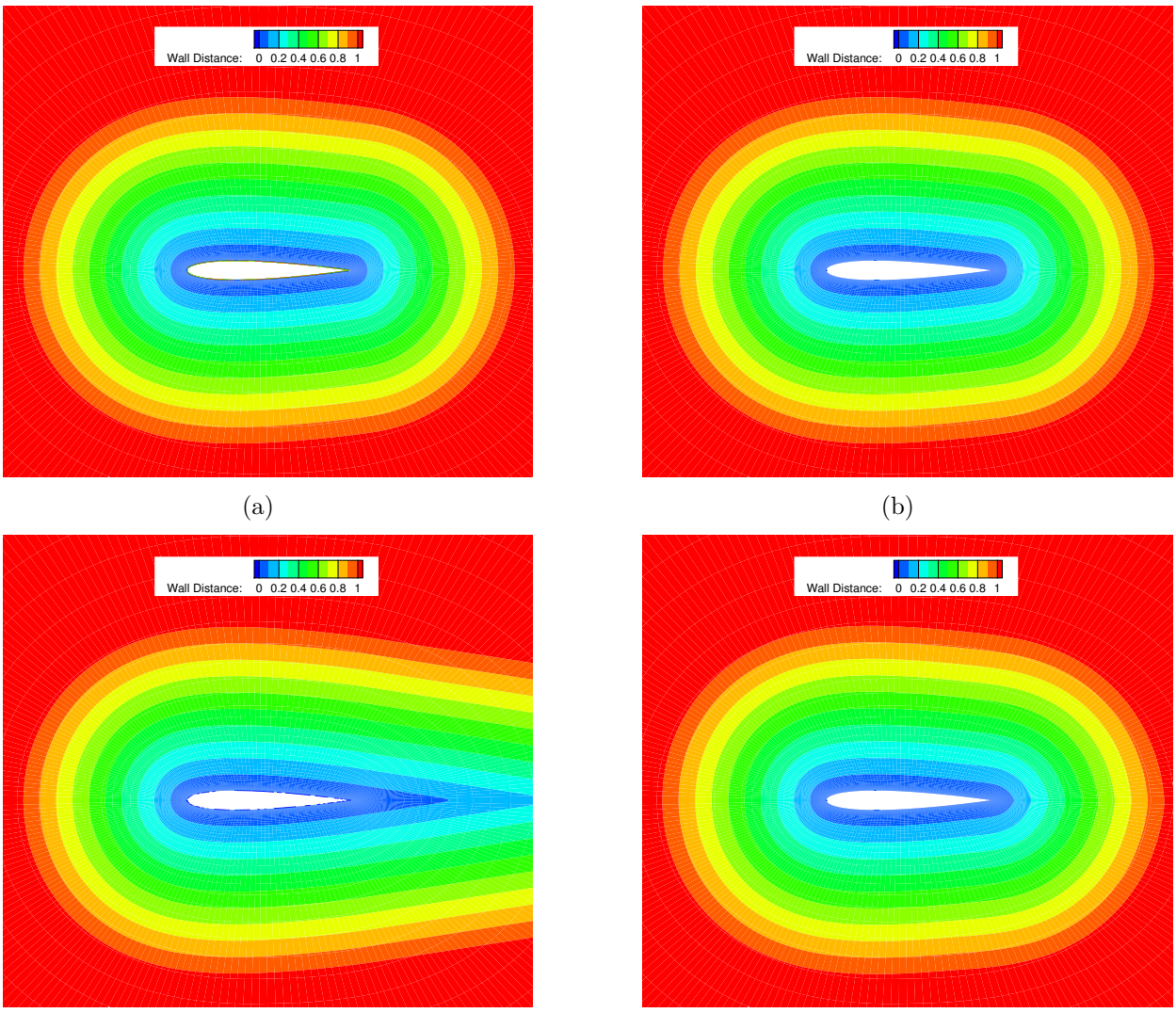
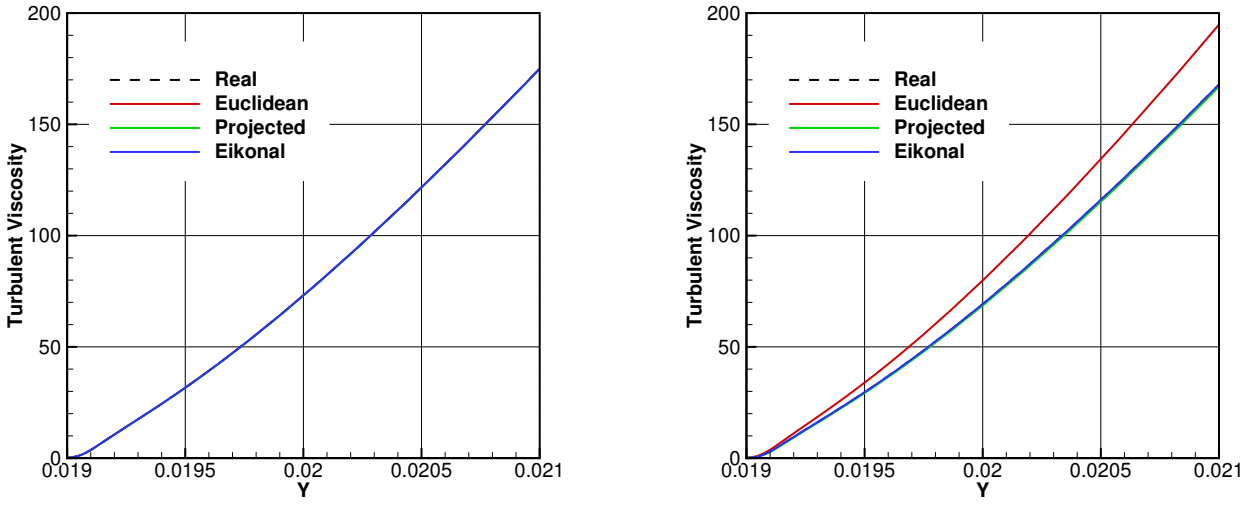



Figure 3.10 Wall distance contours of NACA0012 with (a) the real distance, (b) Euclidean distance, (c) projected distance and (d) the Eikonal equation

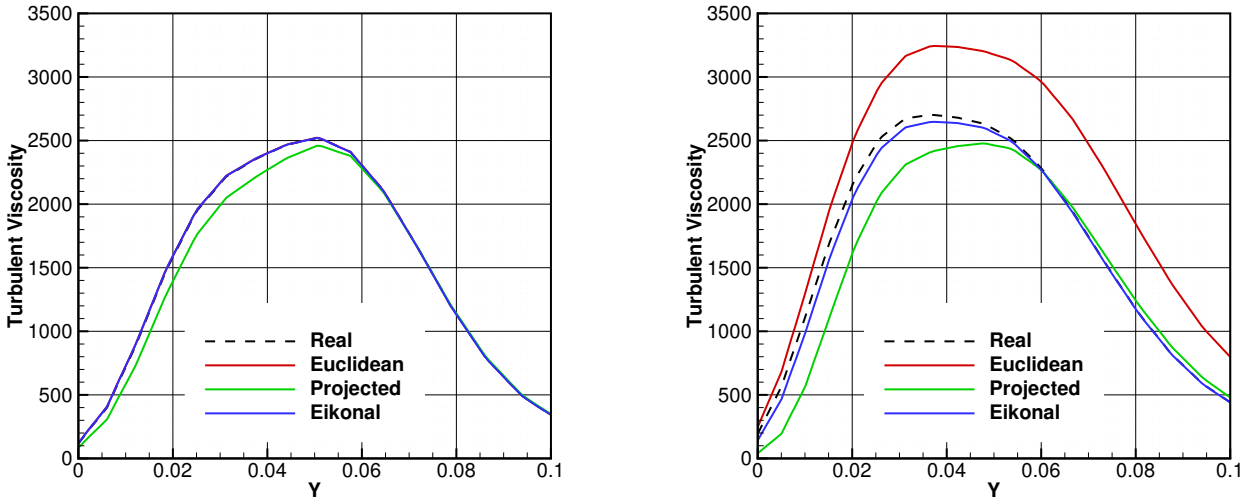
influence the flow solution: the Spalart-Allmaras, Menter's SST and Langtry-Menter transitional SST models. For the first turbulence model, the free stream conditions were set to $M = 0.15$ and $Re_c = 6 \times 10^6$. The simulations were run at $\alpha = 15°$. Figures 3.11 and 3.12 show the results of the simulations on the turbulent viscosity on the upper surface ($x = 0.856c$) and at the wake ($x = 1.2c$) respectively.

It can be seen that, on the upper surface, the Euclidean distance yielded different results than the other three distances on the algebraic mesh, confirming that Euclidean distance does not account for grid skewness. Moreover, at the wake the projected distance produced different turbulent viscosities than the three other distances on the smooth grid, while the




(a) (b)

Figure 3.11 Turbulent viscosity on the upper body ($x = 0.856c$) for (a) the smooth mesh and (b) the skewed mesh with Spalart-Allmaras




(a) (b)

Figure 3.12 Turbulent viscosity at the wake ($x = 1.2c$) for (a) the smooth mesh and (b) the skewed mesh with Spalart-Allmaras

Euclidean distance appeared to be greatly affected by grid skewness. The difference between the projected and Eikonal distance remained approximately constant between the two grids, even as the real solution changed between the two grids. Table 3.2 shows how these results affect the flow forces on the airfoil, by comparing the aerodynamic coefficients for the two grids and the three wall distance methods as well as the difference between the two meshes.

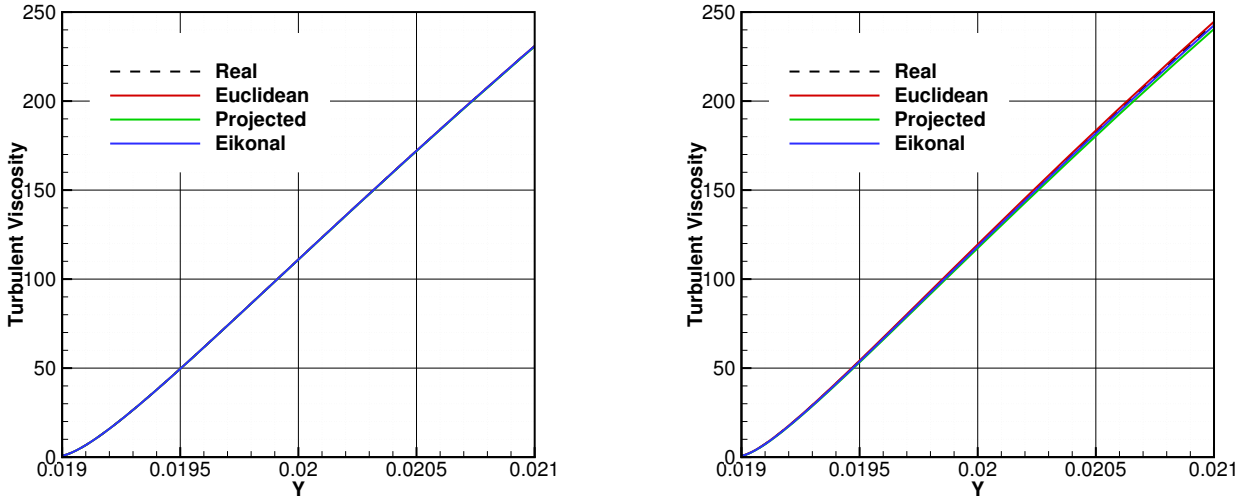
Table 3.2 Comparison of orthogonal and skewed NACA0012 aerodynamic coefficients at $\alpha = 15°$, $M = 0.15$, $Re_c = 6 \times 10^6$ with Spalart-Allmaras

| | | $C_L$ | $C_{Dp}$ | $C_{Dv}$ | $C_D$ | $L/D$ |
|---|---|---|---|---|---|---|
| Real | Orthogonal | 1.5746 | 202 | 52 | 254 | 62.0 |
| | Skewed | 1.5696 | 203 | 52 | 255 | 61.6 |
| | Difference (%) | 0.3 | 0.5 | 0.0 | 0.4 | 0.6 |
| Euclidean | Orthogonal | 1.5751 | 202 | 53 | 255 | 61.8 |
| | Skewed | 1.5414 | 228 | 60 | 288 | 53.5 |
| | Difference (%) | 2.1 | 12.9 | 13.2 | 12.9 | 13.4 |
| Projected | Orthogonal | 1.5727 | 202 | 53 | 254 | 61.9 |
| | Skewed | 1.5499 | 199 | 52 | 252 | 61.5 |
| | Difference (%) | 1.5 | 1.5 | 0.2 | 0.8 | 0.6 |
| Eikonal | Orthogonal | 1.5750 | 202 | 53 | 254 | 62.0 |
| | Skewed | 1.5656 | 201 | 53 | 253 | 61.9 |
| | Difference (%) | 0.6 | 0.5 | 0.0 | 0.4 | 0.2 |

From these results, it can be seen that, for the orthogonal grid, the Euclidean and Eikonal distances yielded forces as close to the coefficients of the real distance as might be expected based on previous results. The skewed grid gave better aerodynamic coefficients with the Eikonal distances as the lift ($C_L$) and total drag ($C_D$) coefficient only varied from 0.6% and 0.4% for this method compared to 2.1% and 12.9% for the Euclidean distance respectively. Note that even the real distance presented small differences between the two grids. The projected distance yielded values close to the other two methods for the non-skewed mesh but varied little with the skewed mesh. Therefore, the projected distance, like the Eikonal equation, produced accurate results in both cases as it was less affected by mesh orthogonality. It can also be seen that, for the projected and the Eikonal distances, the viscous drag ($C_{Dv}$) did not change significantly between the orthogonal and skewed meshes. Knowing that the viscous drag is computed from the shear stress on the airfoil, the first layer of cells at the wall was used. This result adds to the argument that these two methods give a good approximation of the distance close to the wall even for non-orthogonal grids. In other words, these two methods overcome problems related to the mesh as they yielded results for the skewed mesh that compared well with the orthogonal grid. However, as noted earlier, even if the distance field is accurate as for the two different grids, poor grid quality may alter the evaluation of the flow variables. This may lead to small differences in the aerodynamic coefficients between the two meshes as observed with the real distance. Therefore, it is a valid result that the assessment of the aerodynamic coefficients for the Eikonal equation are very close to the real distance when using both orthogonal and skewed meshes. Finally, it can be seen that the lift to drag ratio of the Eikonal and projected distances, as in the case of the real distance, did

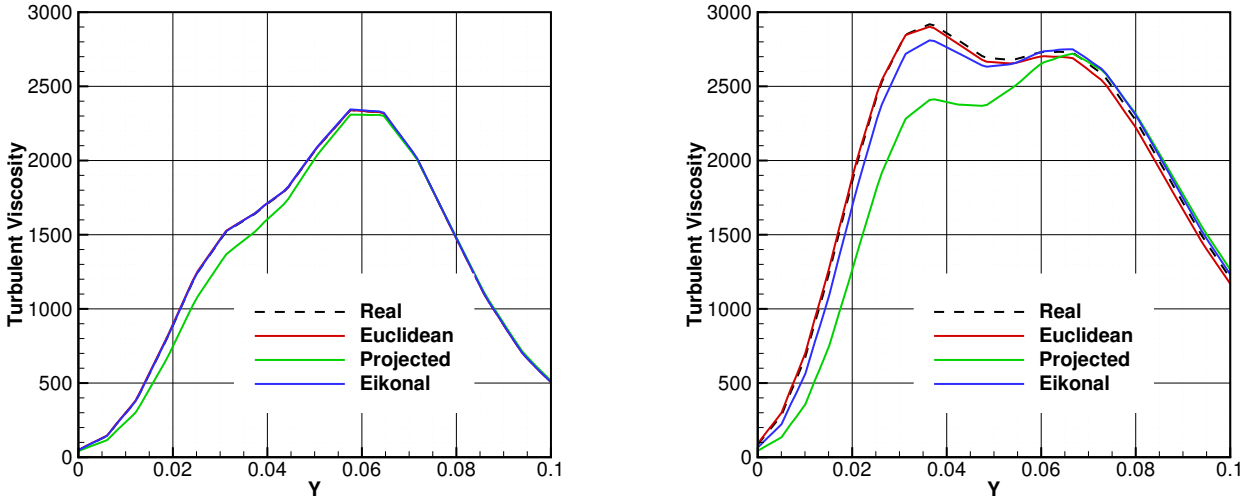
not change significantly between the two grids while, for the Euclidean distance, this ratio changed completely.

The free stream conditions were the same for Menter's SST turbulence model as the Spalart-Allmaras model. Figure 3.13 and 3.14 show the vertical variation of the turbulent viscosity on the upper surface ($x = 0.856c$) and at the wake ($x = 1.2c$) for the two grids.




(a) (b)

Figure 3.13 Turbulent viscosity on the upper surface ($x = 0.856c$) for (a) the smooth mesh and (b) the skewed mesh with Menter SST




(a) (b)

Figure 3.14 Turbulent viscosity at the wake ($x = 1.2c$) for (a) the smooth mesh and (b) the skewed mesh with Menter's SST

On the upper surface, the four distances gave results very close to one another for the two grids. At the wake, each wall distance was greatly affected by grid skewness, as indicated by the significant difference between results of the two grids. However, results of the Euclidean distance were near the real distance for values of $Y$ less than 0.06, while the Eikonal equation yielded the closest results for values greater than 0.06.

Table 3.3 shows how these results affect the flow forces on the airfoil by comparing the aerodynamic coefficients for the two grids and the three wall distance methods as well as the difference between the two meshes.

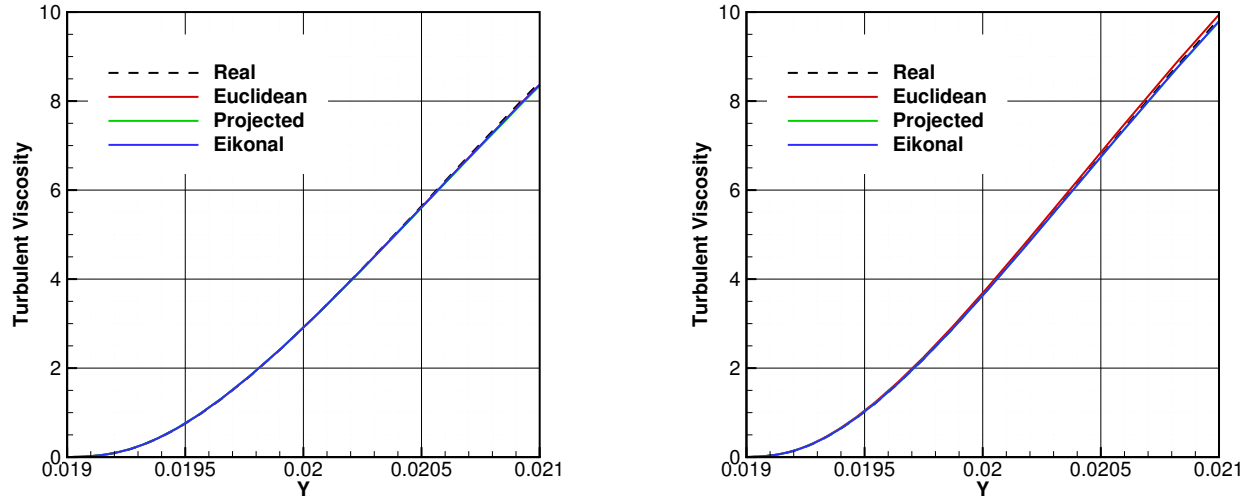
Table 3.3 Comparison of orthogonal and skewed NACA0012 aerodynamic coefficients at $\alpha = 15°$, $M = 0.15$, $Re_c = 6 \times 10^6$ with Menter's SST

| | | $C_L$ | $C_{Dp}$ | $C_{Dv}$ | $C_D$ | $L/D$ |
|---|---|---|---|---|---|---|
| Real | Orthogonal | 1.5067 | 213 | 52 | 265 | 56.9 |
| | Skewed | 1.4635 | 252 | 61 | 312 | 46.9 |
| | Difference (%) | 2.9 | 18.3 | 17.3 | 17.7 | 17.6 |
| Euclidean | Orthogonal | 1.5070 | 214 | 52 | 265 | 56.9 |
| | Skewed | 1.4668 | 250 | 61 | 311 | 47.2 |
| | Difference (%) | 2.7 | 16.8 | 17.3 | 17.4 | 17.0 |
| Projected | Orthogonal | 1.5040 | 213 | 52 | 265 | 56.8 |
| | Skewed | 1.4534 | 249 | 61 | 310 | 46.9 |
| | Difference (%) | 3.4 | 16.9 | 17.3 | 17.0 | 17.4 |
| Eikonal | Orthogonal | 1.5069 | 214 | 52 | 265 | 56.9 |
| | Skewed | 1.4598 | 251 | 61 | 312 | 46.8 |
| | Difference (%) | 3.1 | 17.3 | 17.3 | 17.7 | 17.8 |

Here, it can be seen that, as with the Spalart-Allmaras model, the Euclidean and Eikonal distances produced forces close to the real distance on the orthogonal mesh. Barring the lift coefficient, the projected distance also compared very well to the real distance. However, the four distances did not seem to exhibit any tendency between the two meshes. Indeed, each method can have differences as high as 17% for the drag coefficients and 3% for the lift coefficient. The role of wall distance in this turbulence model differs from the role in the Spalart-Allmaras model. Indeed, the metric is only used as a blending function between two turbulence models. In other words, very close to the walls, the model did not vary much from one distance to another. It is then expected that the use of different wall distance solutions does not help to correct the errors in the evaluation of the turbulence introduced by bad mesh quality. In brief, for the Menter's SST model, the wall distance appeared to have a smaller influence on the turbulent viscosity and flow solution close to the wall compared to the Spalart-Allmaras model and therefore, the effects of the mesh quality cannot be corrected

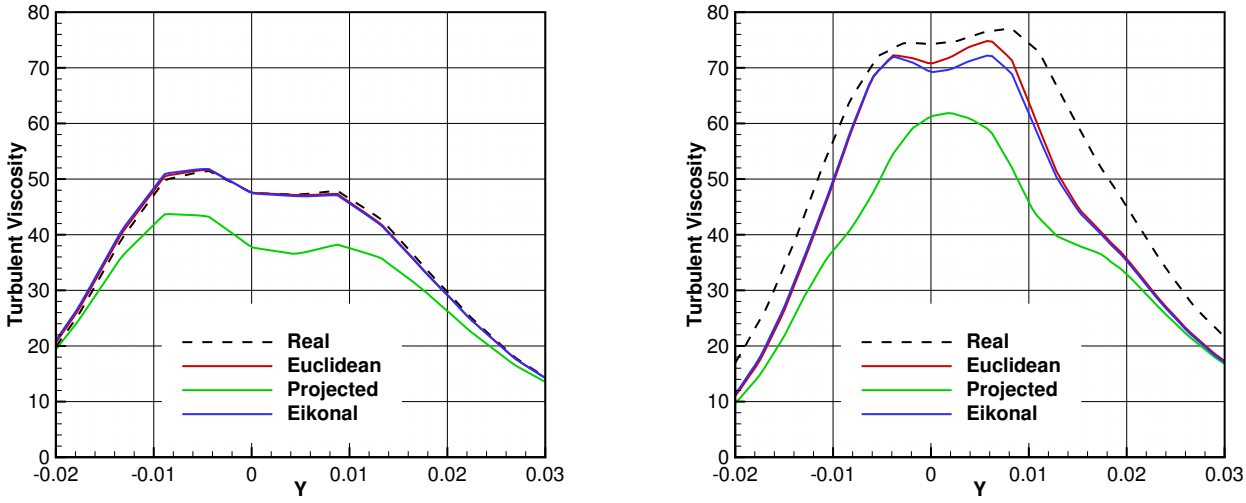
by a good evaluation of the wall distance.

Finally, for the Langtry-Menter transitional SST model, the free stream conditions were set to $M = 0.2$ and $Re_c = 0.283 \times 10^6$ and the angle of attack to $\alpha = 3°$. Figure 3.15 and 3.16 show the vertical variation of the turbulent viscosity on the upper surface ($x = 0.856c$) and at the wake ($x = 1.1c$) for the two grids.




(a) (b)

Figure 3.15 Turbulent viscosity on the upper surface ($x = 0.856c$) for (a) the smooth mesh and (b) the skewed mesh with Langtry-Menter transitional SST




(a) (b)

Figure 3.16 Turbulent viscosity at the wake ($x = 1.1c$) for (a) the smooth mesh and (b) the skewed mesh with Langtry-Menter transitional SST

For turbulent viscosity, the same conclusions can be drawn as for Menter's SST model, except that the solution for real distance was closer to the Euclidean distance everywhere on the graph. Table 3.4 shows how these results affect the flow forces on the airfoil by comparing the aerodynamic coefficients for the two grids and the three wall distance methods as well as the difference between the two meshes.

Table 3.4 Comparison of orthogonal and skewed NACA0012 aerodynamic coefficients at $\alpha = 3°$, $M = 0.2$, $Re_c = 2.83 \times 10^5$ with Langtry-Menter transitional SST

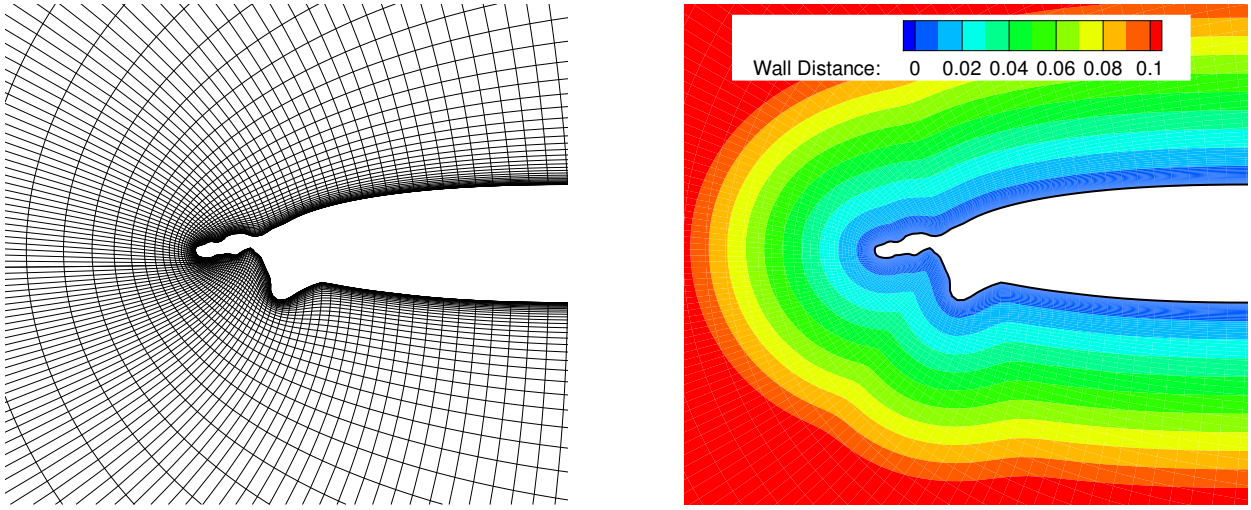
| | | $C_L$ | $C_{Dp}$ | $C_{Dv}$ | $C_D$ | $L/D$ |
|---|---|---|---|---|---|---|
| Real | Orthogonal | 0.3682 | 52 | 44 | 97 | 38.0 |
| | Skewed | 0.3464 | 52 | 46 | 98 | 35.3 |
| | Difference (%) | 5.9 | 0.0 | 4.5 | 1.0 | 7.1 |
| Euclidean | Orthogonal | 0.3713 | 53 | 45 | 97 | 38.3 |
| | Skewed | 0.3477 | 52 | 46 | 98 | 35.5 |
| | Difference (%) | 6.4 | 0.2 | 2.2 | 1.0 | 7.3 |
| Projected | Orthogonal | 0.3753 | 53 | 45 | 97 | 38.7 |
| | Skewed | 0.3495 | 53 | 46 | 98 | 35.7 |
| | Difference (%) | 6.9 | 0.0 | 2.2 | 1.0 | 7.8 |
| Eikonal | Orthogonal | 0.3724 | 53 | 45 | 97 | 38.4 |
| | Skewed | 0.3482 | 53 | 46 | 98 | 35.5 |
| | Difference (%) | 6.5 | 0.0 | 2.2 | 1.0 | 7.6 |

Here, it is observed that the three wall distance methods produced results close to the real distance for the two meshes. Moreover, when using the skewed mesh, the coefficients changed only slightly compared with the previous mesh. This can be explained by the fact that, to have a transition on the airfoil, the flow conditions changed and therefore turbulence effects were less significant. The transitional model is, thus, really less sensitive to wall distance than the other two turbulence models. Moreover, the wall distance was used in blending functions, as in the case of the Menter's SST model. The grid metric did not seem to significantly influence the flow solution or correct terrors introduced by the bad mesh.

Overall, what can be concluded from these results is that the wall distance only has a significant influence on turbulence models when this metric is explicitly used in the equations, as in the Spalart-Allmaras model. Models that use it in other way, such as in blending functions, are not significantly sensitive to wall distance and thus, an accurate solution is not mandatory for accuracy of the flow solution.

### 3.7.4 Ice Accreted Airfoil

To show the robust convergence of the Eikonal equation and the multi-grid acceleration, the algorithm was tested on a 257x129 O-grid ice accreted NACA0012 airfoil. For visualization purposes and to highlight wall distance contours Figure 3.17 depicts every other point of the original grid in the normal-wall direction.




(a) (b)

Figure 3.17 Ice accreted NACA0012 (a) O-grid and (b) wall distance contours

From these figures, it can be seen that the wall distance contours appear adequate and fit well with the shape of the airfoil. To ensure that the Eikonal equation converges well for this type of geometry, Figure 3.18 shows the iterative convergence graph for both usual and modified convergence criteria. Iterations consist of one pre-sweep and one post-sweep, where each sweep has four DPLUR relaxation steps.

In this case, we see that the convergence of the Eikonal equation was detected sooner by approximately ten iterations with the modified criterion. On larger and more complex meshes, this gain could be even higher. Moreover, it is clear that the modified criterion does not hide any convergence problems at the far field as both criteria converge to $10^{-16}$ smoothly. For the subsequent convergence history graphs, the modified criterion was used. To show the acceleration of the multi-grid scheme, Figure 3.19 plots the convergence of the Eikonal equation up to five levels of the multi-grid with respect to both multi-grid cycles and CPU time.

This case was computed on four threads of an Intel Core i7-3930K CPU. The figures show smooth convergences of the Eikonal equation up to five multi-grid levels. Also, the solver

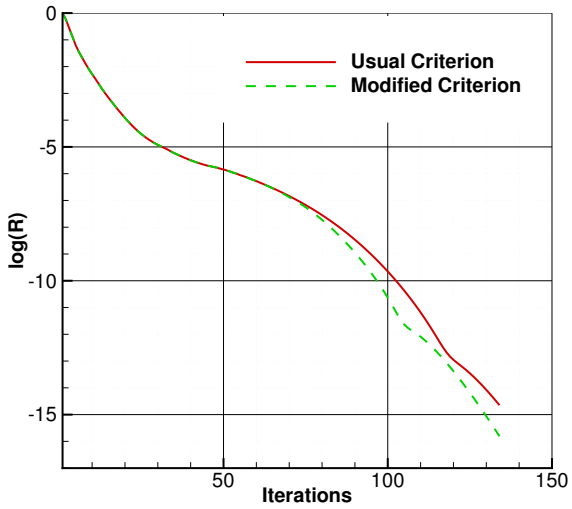
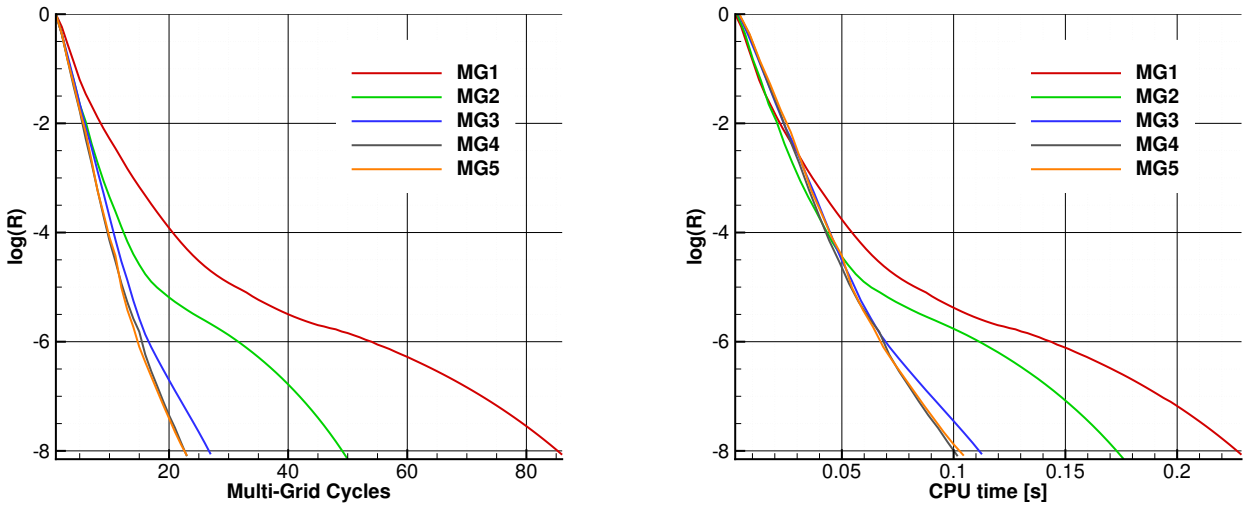



Figure 3.18 Iterative convergence of an ice accreted NACA0012 with usual and modified convergence criteria




(a) (b)

Figure 3.19 Convergence of an ice accreted NACA0012 with respect to (a) multi-grid cycles and (b) CPU time

was approximately two times faster converging in half the number of iterations when the number of multi-grid levels was increased to three, converging in the same time and in the same number of iterations. We conclude that, in this case, the gain is not significant beyond four levels of the multi-grid. However, for larger problems, more multi-grid levels might bring more gains.

In the last case, the wall distance was computed on the final icing layer. However, since ice

accretion causes deforming grid problems, wall distance needs to be computed more than once (i.e. at each layer). In addition, the Eikonal equation did not need to converge to machine accuracy since the engineering precision was approximately $10^{-6}$. Furthermore, the solution can be restarted from the last layer to give a better first approximation. To see how the Eikonal equation performed compared to the other two wall distance methods, all icing layers were run from the beginning of the multi-layer process. Iteration consisted of one pre-sweep and one post-sweep with three multi-grid levels where each sweep had four DPLUR relaxation steps. In this work, the same methodology for computation time was applied for each case. The solver was run five times with a ten second break between each computation. Appendix B shows the mean time of these simulations as well as the standard deviation for every run. Figure 3.20 and Table 3.5 show the convergence of the Eikonal equation on every layer and the computation times for each method respectively.

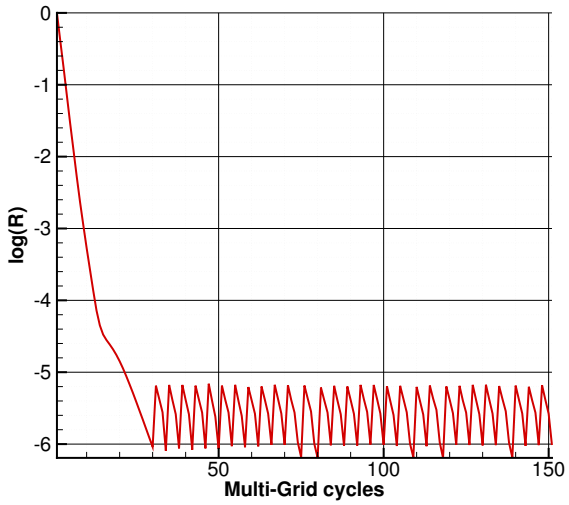



Figure 3.20 Iterative convergence of the Eikonal equation for a multi-layer icing case on an Intel Core i7-3930K CPU using 8 OpenMP threads

Table 3.5 Computation times for wall distance for a multi-layer icing case on an Intel Core i7-3930K CPU using 8 OpenMP threads

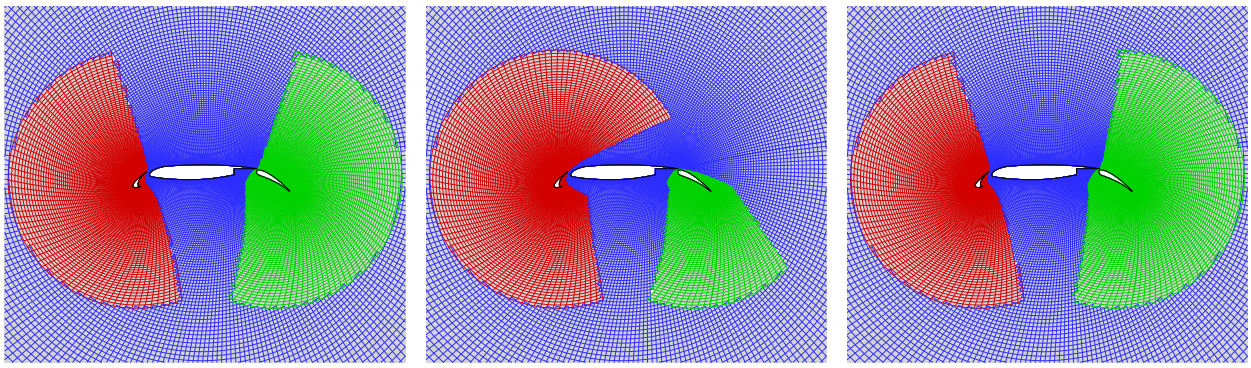
| | Euclidean | Projected | Eikonal |
|---|---|---|---|
| Mean time [s] | 0.9310 | 1.1884 | 0.4758 |
| Standard Deviation [s] | 0.0098 | 0.0032 | 0.0044 |

As can be seen, when the previous solution was used as the initial solution for the next layer the computation of the Eikonal equation accelerated. For this reason, it was expected that the re-initialized solution would show an edge for the Eikonal equation compared to the other

two methods. In fact the Eikonal equation required considerably less computing time than the other two wall distance methods.

### 3.7.5 McDonnell Douglas Airfoil (MDA)

A multi-element airfoil was tested with overset meshes to show the capability of the overset grid algorithm. The MDA airfoil consisted of the main body, the flap and the slat. Each element had its own mesh. The main mesh was a 257x257 O-grid while the slat and flap meshes were 129x129 O-grids. Figure 3.21 shows the hole cutting of the meshes for the three wall distance methods.

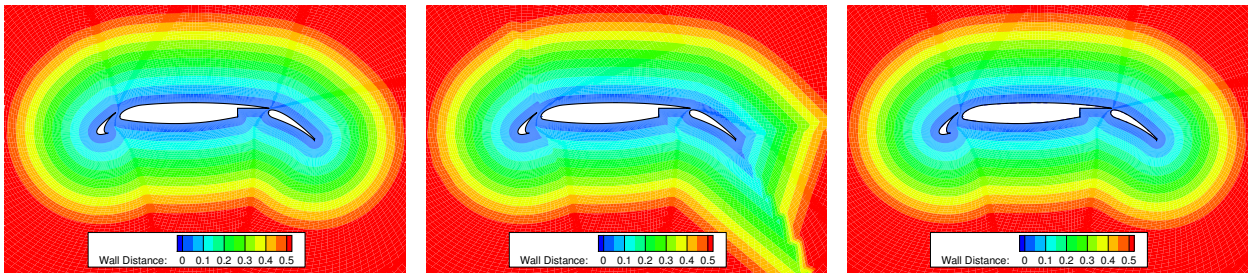


(a) (b) (c)

Figure 3.21 Mesh construction of the MDA with (a) Euclidean distance, (b) projected distance and (c) the Eikonal equation

From the above figures, it can be seen that hole cutting for the Euclidean and the Eikonal distances were similar and produced intuitive results. However, the hole cutting was inaccurate when the projected distance was used as the criterion. Figure 3.22 illustrates how these results affect the final wall distance evaluation, by showing the contours of the wall distance on the MDA airfoil for the three distance algorithms.

The contours were similar for Euclidean and Eikonal distances. Moreover, they were smooth and fit well the shape of the airfoil showing no discontinuity at the junctions. However, the contours presented odd features such as discontinuities for the projected distance. We concluded that the overset grid is well supported by the Eikonal algorithm developed in this work.

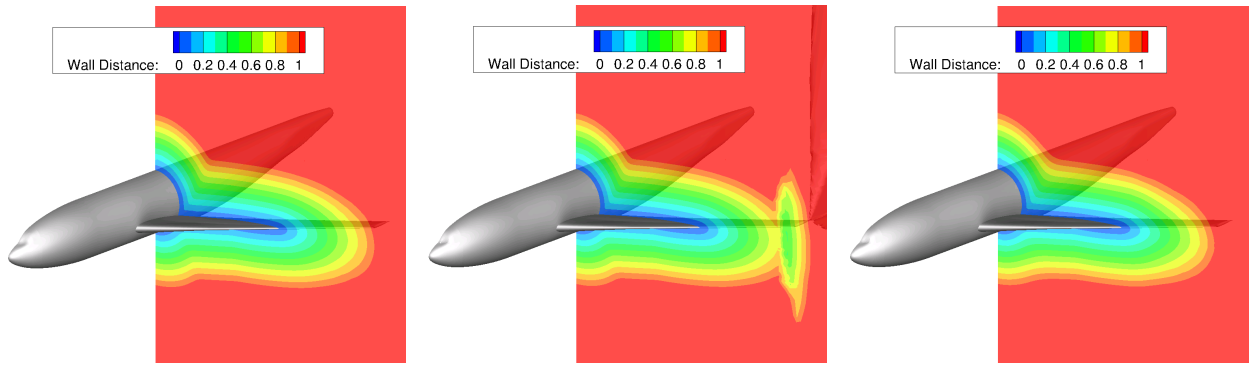



(a) (b) (c)

Figure 3.22 Wall distance contours of a MDA with (a) Euclidean distance, (b) projected distance and (c) the Eikonal equation

### 3.7.6 DLR-F6

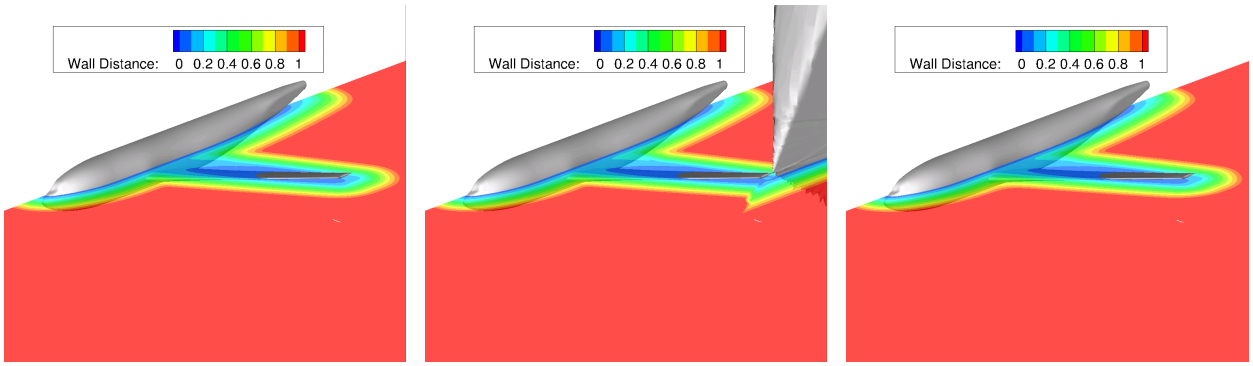
The Eikonal equation was then tested for a 3D case implemented on the 3D RANS solver FANSC. The case chosen was the DLR-F6, with a drag prediction workshop geometry. Figures 3.23, 3.24 and 3.25 show the wall distance contours on $X$, $Y$ and $Z$ slices respectively. Note that the real distance seen in earlier NSCODE results was not coded in FANSC. Results of the Eikonal equation were compared to Euclidean and projected distances.




(a) (b) (c)

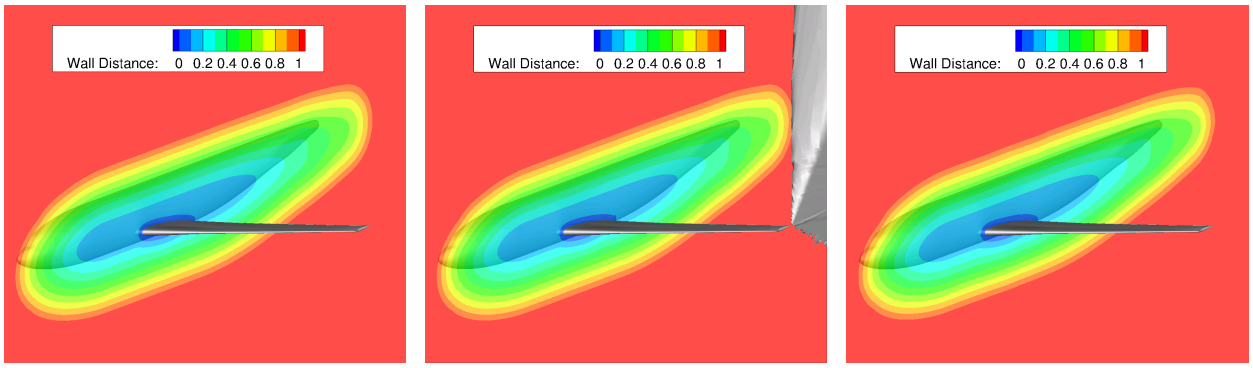
Figure 3.23 Wall distance contours of a DLR-F6 with (a) Euclidean distance, (b) projected distance and (c) the Eikonal equation

As expected from previous 2D results, the projected distance had problems at the tip of the wing while the two other methods fit well with the shape of the airplane. However, as is well known, it is difficult to obtain good quality meshes on 3D geometries, as grid skewness sometimes occurs at the wall. It has already been shown that grid quality may affect results for wall distance and thus air flow solutions. We expected that the Euclidean distance would produce inaccurate drag prediction due to these effects while the two other methods should be close to one another. To verify this hypothesis, Table 3.6 compares the aerodynamic




(a) (b) (c)

Figure 3.24 Wall distance contours of a DLR-F6 with (a) Euclidean distance, (b) projected distance and (c) the Eikonal equation




(a) (b) (c)

Figure 3.25 Wall distance contours of a DLR-F6 with (a) Euclidean distance, (b) projected distance and (c) the Eikonal equation

coefficients given by the three methods at flow conditions of $M = 0.75$ and $Re_c = 3 \times 10^6$ at an angle of attack of $\alpha = 0.5°$ using the Spalart-Allmaras turbulence model.

Table 3.6 Comparison of the DLR-F6 aerodynamic coefficients at $\alpha = 0.5°$, $M = 0.75$, $Re_c = 3 \times 10^6$ with Spalart-Allmaras

| | $C_L$ | $C_{Dp}$ | $C_{Dv}$ | $C_D$ | $L/D$ |
|---|---|---|---|---|---|
| Euclidean | 0.5051 | 177 | 148 | 324 | 15.6 |
| Projected | 0.5566 | 180 | 130 | 310 | 18.0 |
| Eikonal | 0.5653 | 182 | 130 | 312 | 18.1 |

As anticipated, the Euclidean distance produced a much higher drag than the two other methods. This difference was due to the viscous drag, previously determined to be more sensitive to grid skewness. Moreover, the lift coefficients of the projected and Eikonal distances

were closer to one another compared to the Euclidean distance, which yielded a smaller value. Furthermore, the Lift-to-Drag ratio ($L/D$) of the projected and the Eikonal distances were similar while the Euclidean distance was much smaller for the skewed NACA0012 grid results as seen in section 3.7.3 where the Eikonal and projected distances showed results closer to the real distance.

These simulations were performed using a HPC cluster from Calcul Québec and run on four nodes on two CPUs (Intel Xeon E5-2670 Sandy Bridge) each with the help of 64 MPI ranks. Figure 3.26 shows the smooth convergence of the Eikonal equation to $10^{-6}$ using one pre-sweep and one post-sweep for four DPLUR relaxations, each with three levels of multi-grid.

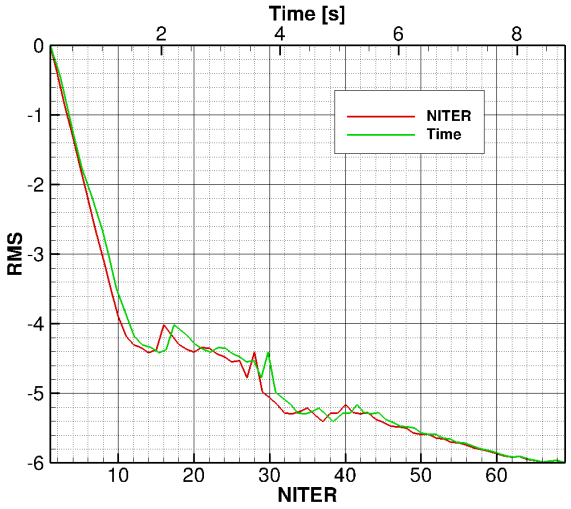



Figure 3.26 Iterative convergence of the Eikonal equation for a DLR-F6 case on 4 nodes using 64 MPI ranks

Table 3.7 shows the computation times for each wall distance method illustrating how the Eikonal equation compares to the Euclidean and projected distances. Appendix B shows the results of each run.

Table 3.7 Computational times of the wall distance for a DLR-F6 case on 4 nodes using 64 MPI ranks

| | Euclidean | Projected | Eikonal |
|---|---|---|---|
| Mean time [s] | 3.26 | 3.33 | 8.61 |
| Standard Deviation [s] | 0.11 | 0.14 | 0.23 |

Both the Euclidean and projected distance methods had shorter computation times than the Eikonal equation. Further analyses on parallelization are performed in section 4 and

providing an understanding as to why the Eikonal equation takes longer than expected to compute.

# CHAPTER 4 PARALLELIZATION OF THE SOLVER

Having verified the Eikonal equation solver for a variety of test cases, we discuss parallelization. Previously, some results were shown where the Eikonal equation was solved with multi-threading. In this chapter, we present studies carried out on the scalability of the Eikonal equation and results compared with the other two solvers. First, the parallel efficiency of the Eikonal equation was assessed using a shared memory architecture with the 2D RANS solver NSCODE. Then, with the 3D RANS solver FANSC, the Eikonal solver was parallelized on a distributed memory architecture.

## 4.1 Shared Memory Architecture

The Eikonal equation was parallelized using NSCODE, with multi-threading on a single CPU, using OpenMP. We first describe implementation and then present the results.

### 4.1.1 Implementation in 2D and 3D Solvers

As was discussed in the literature review, OpenMP is straightforward to implement. Indeed, only simple OpenMP commands need to be added to the code, and then recompiled. In NSCODE, multi-threading is applied on blocks. Algorithm 1 shows an example of a loop that was parallelized in NSCODE.

Algorithm 1 OpenMP Implementation in NSCODE

```
#pragma omp parallel shared(bag) num_threads(bag− >ncpus)
Declare Variables
#pragma omp for schedule(static)
for each block do
  for each j do
    for each i do
      Do some operations
    end for
  end for
end for
```

One consequence of this method is that blocks need to be evenly distributed to have optimal parallelization. The parallelization was done differently in FANSC. Because FANSC has MPI capabilities that are applied on blocks, the OpenMP parallelization was applied to *cell* loops

instead. Compared to NSCODE, this method distributes the cells more evenly to the threads regardless of the block distribution. However, halo cells were also computed to prevent a jump in the memory, and thus slow down the code. Algorithm 2 shows an example of a loop that was parallelized in FANSC.

Algorithm 2 OpenMP Implementation in FANSC

```
#pragma omp parallel firstprivate(block)
for each block do
  Declare Variables
  #pragma omp for schedule(static)
  for each cell do
    Do some operations
  end for
end for
```

As can be seen in both cases, only a few lines of code needed to be added for parallelization on shared memory architecture.

### 4.1.2 Results

The wall distance solvers in NSCODE were run on a NACA0012 airfoil O-grid with 1025x513 nodes split in four blocks as shown in Figure 4.1.

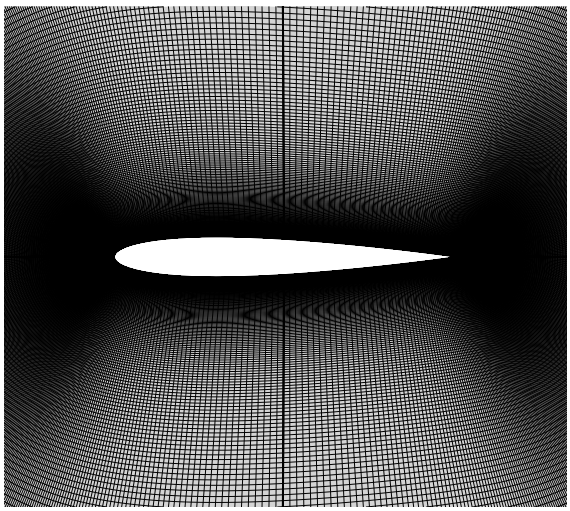


Figure 4.1 NACA0012 airfoil O-grid with 1025x513 nodes

The solvers were run using one, two and four threads on an Intel Core i7-3930K CPU. Figure 4.2 shows the CPU time for these three wall distance solvers. Results of the simulations can

be found in Appendix B.

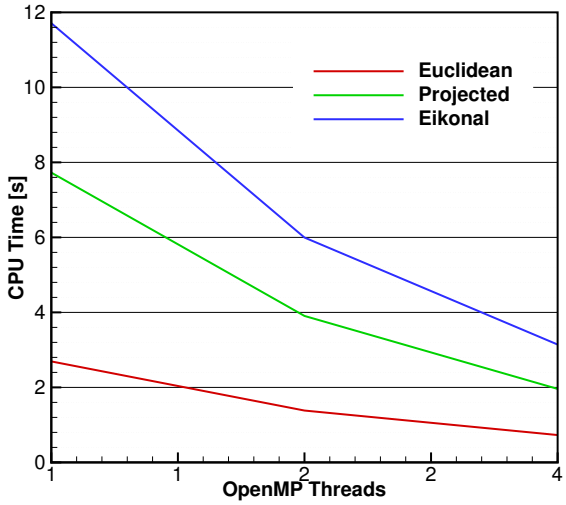



Figure 4.2 Computation time for wall distance solvers vs. OpenMP threads

The Eikonal equation required more time to compute than the other two methods. On this small mesh, the Euclidean and projected distances ran faster than the Eikonal equation. However, as stated in the literature review, as meshes grow in size, the Eikonal equation was expected to be faster than the two algebraic methods. Therefore, it was important to check the speedup of the solvers as shown in Figure 4.3.

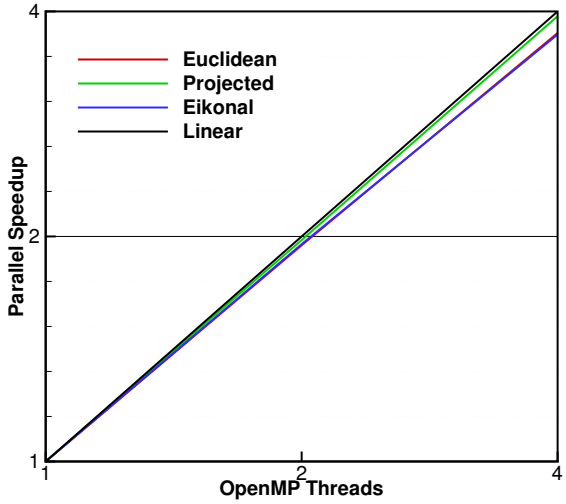



Figure 4.3 Parallel speedup of wall distance solvers with respect to OpenMP threads

It is apparent that the projected distance scales best. The Euclidean and projected solvers share the same code structure. Therefore, it is not clear why these two solvers have different

scalability. However, because the projected distance uses the result of the Euclidean distance for its computation, the former will always take more time to compute than the Euclidean distance. Also, the Eikonal equation should be computed in less time than the two other methods for larger meshes. In future work it would be interesting to assess the efficiency of parallelization more precisely. Figure 4.4 shows the parallel efficiency of the three methods.

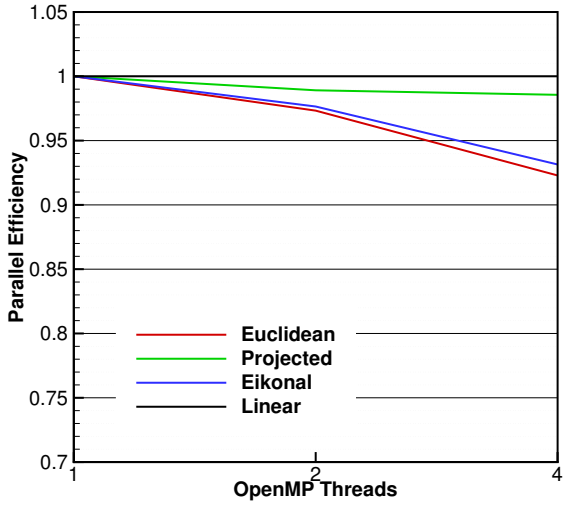



Figure 4.4 Parallel efficiency of wall distance solvers verses OpenMP threads

Similar conclusions can be drawn from parallel speedup and parallel efficiency. In brief, the Eikonal equation performed well in terms of scalability on shared memory architectures.

## 4.2 Distributed Memory Architecture

The Eikonal equation was parallelized in FANSC code for multiprocessor systems using MPI. We first describe implementation and then present the results.

### 4.2.1 Implementation

MPI was more complex to implement than OpenMP. Indeed, the code had to be thought through in its entity. In order to run on the different MPI ranks the blocks were first divided and then multiple instances of the code run on all ranks. On all block loops, the code checked out if the MPI rank corresponded to its block and if so, the code ran the block. After updating the new solution in the restricted domain, the halos were updated. For the connection boundaries, since the CPUs did not have access to each other's memory, information was shared between one another. CPUs send the updated solution at the boundaries to the CPUs that need them. Algorithm 3 shows an example of this process in FANSC.

Algorithm 3 MPI Implementation in FANSC

```
for each block do
  if MPI Rank == block rank then
    Do some operations
  end if
end for
for each connection boundary do
  MPI send
end for
for each connection boundary do
  MPI receive
end for
```

### 4.2.2 Results

The wall distance solvers in FANSC were run on the DLR-F6 aircraft using up to 64 cores on four Intel Xeon E5-2670 Sandy Bridge CPUs with the help of Calcul Québec clusters. Figure 4.2 shows CPU time for the three wall distance solvers. Results of theses simulations can be found in Appendix B.

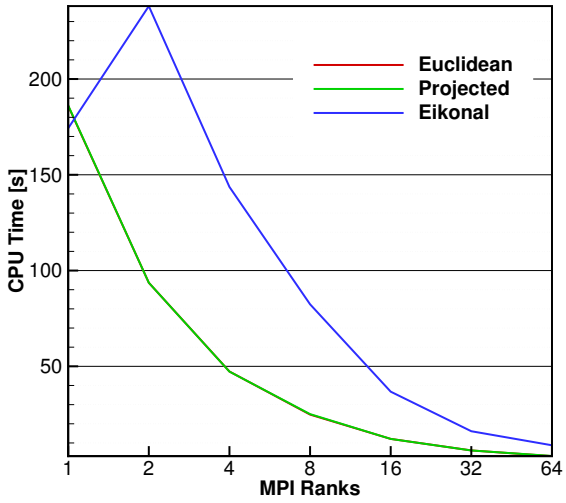



Figure 4.5 Computational time of wall distance solvers with respect to MPI ranks

We note first, with one core, the Eikonal equation was faster than the Euclidean and projected distances. This is expected as the Eikonal equation scales better when the mesh is enlarged compared to the Euclidean and projected distance as it has been stated in section 2.5. This is a direct effect of the better complexity of the Eikonal algorithm compared to search algorithms. However, when using two cores, the code slowed down. This may be due

to the MPI communication, which has the effect of shifting the curve. More work needs to be carried out in this area to determine the cause of the slow down. The computation time for the Eikonal equation appears to decrease at a similar rate to the two other wall distance methods. To better assess the improvement, Figure 4.6 presents speedup for the various wall solvers verses MPI ranks compared to simulations with one core.

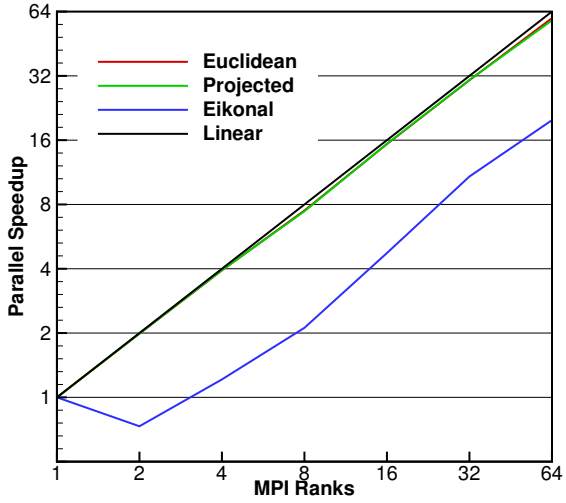



Figure 4.6 Parallel speedup of wall distance solver vs. MPI ranks

We note that the computation time for the Eikonal equation slowed down for two MPI ranks. Furthermore, with more than two MPI ranks, the Eikonal equation appeared to speed up linearly, similar to the two other wall distance methods, which suggests that if the problem with the load balancing or communication were solved, the Eikonal solver would have a similar performance to the Euclidean and projected distances.

# CHAPTER 5 CONCLUSION

In this chapter we present a synthesis of the work, discuss limitations to the proposed solution, and suggest possible future work.

## 5.1 Synthesis of Work

The overarching objective of the work was to take advantage of the development in massively parallel hardware architectures in CFD; more precisely, to develop a suitable wall distance solver for these architectures. The Eikonal equation was chosen for its scalability and accuracy and implemented in two CFD codes: NSCODE, a development 2D RANS code, and FANSC, an industrial 3D RANS code. The codes were parallelized for better performance on shared memory architectures and distributed memory architectures as well as hardware architectures like Intel Xeon Phi's and GPGPUs.

### 5.1.1 Development of the Eikonal Solver

The software for the solver dictated the choice of the algorithm. Since NSCODE and FANSC already had well optimized differential equation algorithms, we chose the Eikonal equation over other options. Moreover, we selected a cell-centered method rather than a vertex-centered method since NSCODE and FANSC are center-based cell methods. The Eikonal equation was solved using the advective form of the equation, using both finite difference and finite volume methods.

For the finite difference method, the non-conservative form of the equation was chosen and solved using a first-order upwind discretization. In addition, the metrics were discretized using a first-order upwind approximation. For the finite volume method, the conservative form of the Eikonal equation was solved. A first-order upwind discretization was used for the convective fluxes, while the diffusive fluxes were discretized using a second-order centered approximation. No special treatment was applied to the metric terms. The solution was advanced using an explicit multi-stage Runge-Kutta scheme. The finite difference method was shown to converge in space at a second-order rate, while the finite volume method converged at a first-order rate. To obtain a second-order convergence in space, a linear reconstruction of the solution at the faces was used. Both second-order finite volume and finite difference methods produced a precise solution on highly stretched grids. However, the finite difference algorithm was chosen for the simple reason it required less computing time.

A DPLUR scheme was implemented to accelerate the convergence of the Eikonal equation as it can be easily parallelized. Special features were also used to accelerate the solver such as multi-grid and local time stepping, as well as a special initial solution and a modified convergence criterion.

The results of the Eikonal equation were optimal for the cases chosen. Indeed, the Eikonal equation was shown to have a better wall distance solution than the Euclidean and projected distances with poor mesh quality. The new wall distance algorithm was also compatible with the overset grid implementation in NSCODE. On the other hand, in most cases the Eikonal equation was slower than the other two wall distance methods. However, the fact that the Eikonal equation can restart from any solution, helped make the method faster than the other two for deforming/adaptive mesh problems such as ice accretion.

### 5.1.2 Parallelization of the Eikonal Solver

To improve the computation time for the Eikonal equation, the solver was parallelized on shared memory and distributed memory architectures.

First, the code was parallelized on shared memory architectures using OpenMP, and showed good parallel efficiency compared to the other two methods. MPI was used to parallelize the code on distributed memory architectures, however, results were not as good as expected. Indeed, the Eikonal equation solver decelerated between the first and second cores instead of accelerating. The reason is thought to be poor splitting or distribution of the blocks to the CPUs, producing much more communication than it should. However, beyond two cores, speedup appeared to be linear, which suggested that if this problem were to be solved, the Eikonal equation would show good results for MPI parallelization.

## 5.2 Limitations of the Proposed Solution

The most limiting aspect of the new wall distance solver is that performance was dependent on the software in which the code was implemented. Indeed, as stated above, parallelization of the Eikonal equation on distributed memory architectures is affected by MPI communication in FANSC. The same behavior should be observed for multiple GPGPUs or Intel Xeon Phi accelerators.

Moreover, the fact that the Eikonal equation was developed for use on massively parallel hardware architectures limited the choices for the time advancement scheme. Indeed, a simple Jacobi iteration was chosen for the DPLUR scheme.

Finally, our proposal is a solution for the real wall distance. However, as discussed in the literature review wall distance may not be the best choice for turbulence models. Indeed, wall distance does not represent turbulence as well in convex and concave zones nor on small objects like wires.

## 5.3 Future Work

The Eikonal equation solver developed in this project could be improved in several ways. Methods to overcome the limitations of our solution are proposed here.

The problem related to MPI performance could be solved by changing the way blocks are split or distributed to the CPUs. In this work, blocks were distributed by considering only their size so as to more evenly balance the ranks. However, other parameters are also influential including the position of the block and number of boundary points. Indeed, neighboring blocks should be placed on the same rank, and blocks split in a way to limit the communication between ranks. The challenge is to optimize these two variables for best performance.

There do exist parallelizable faster time advancement schemes such as a red-black or lagged LUSGS scheme that could be used instead of the DPLUR.

Moreover, an additional diffusive term could be added to the Eikonal equation to account for convex and concave features as well as small objects. However, overset grid approaches require the true wall distance, which would prevent using a different solution. Further tests are needed to better assess the impact of a slightly different wall distance on the overset grid approach.

Finally, the Eikonal equation should also be tested on various GPGPUs as well as Intel Xeon Phi accelerators.

# REFERENCES

BAK, S., MCLAUGHLIN, J., and RENZI, D., "Some improvements for the fast sweeping method," *SIAM Journal on Scientific Computing*, vol. 32, no. 5, pp. 2853–2874, 2010.

BARTH, T. and JESPERSEN, D., "The design and application of upwind schemes on unstructured meshes," in *27th Aerospace Sciences Meeting*, 1989, p. 366. [Online]. Available: 10.2514/6.1989-366

BLAZEK, J., *Computational Fluid Dynamics: Principles and Applications:(Book with accompanying CD)*. Elsevier, 2005.

BOGER, D. A., "Efficient method for calculating wall proximity," *AIAA journal*, vol. 39, no. 12, pp. 2404–2406, 2001.

BOURGAULT-COTE, S. and LAURENDEAU, E., "Two-dimensional/infinite swept wing ice accretion model," *AIAA (SciTech 2015)*, pp. 5–9, 2015.

BREUß, M., CRISTIANI, E., GWOSDEK, P., and VOGEL, O., "An adaptive domain-decomposition technique for parallelization of the fast marching method," *Applied Mathematics and Computation*, vol. 218, no. 1, pp. 32–44, 2011.

CAGNONE, J., SERMEUS, K., NADARAJAH, S. K., and LAURENDEAU, E., "Implicit multigrid schemes for challenging aerodynamic simulations on block-structured grids," *Computers & Fluids*, vol. 44, no. 1, pp. 314–327, 2011.

CHACON, A. and VLADIMIRSKY, A., "Fast two-scale methods for eikonal equations," *SIAM Journal on Scientific Computing*, vol. 34, no. 2, pp. A547–A578, 2012.

CHACON, A. and VLADIMIRSKY, A., "A parallel two-scale method for eikonal equations," *SIAM Journal on Scientific Computing*, vol. 37, no. 1, pp. A156–A180, 2015.

DANIELSSON, P.-E. and LIN, Q., "A modified fast marching method," in *Scandinavian conference on image analysis*. Springer, 2003, pp. 1154–1161.

FARES, E. and SCHRÖDER, W., "A differential equation for approximate wall distance," *International journal for numerical methods in fluids*, vol. 39, no. 8, pp. 743–762, 2002.

FROLKOVIČ, P., MIKULA, K., and URBÁN, J., "Semi-implicit finite volume level set method for advective motion of interfaces in normal direction," *Applied Numerical Mathematics*, vol. 95, pp. 214–228, 2015.

FU, Z., KIRBY, R. M., and WHITAKER, R. T., "A fast iterative method for solving the eikonal equation on tetrahedral domains," *SIAM Journal on Scientific Computing*, vol. 35, no. 5, pp. C473–C494, 2013.

GARIÉPY, M., TRÉPANIER, J.-Y., and MASSON, C., "Convergence criterion for a far-field drag prediction and decomposition method," *AIAA journal*, vol. 49, no. 12, pp. 2814–2817, 2011.

GILLBERG, T., BRUASET, A. M., HJELLE, Ø., and SOUROURI, M., "Parallel solutions of static hamilton-jacobi equations for simulations of geological folds," *Journal of Mathematics in Industry*, vol. 4, no. 1, p. 10, 2014.

GUAY, J., "Extension of the overset grid preprocessor for surface conforming meshes," Ph.D. dissertation, École Polytechnique de Montréal, 2017.

HASANZADEH, K., LAURENDEAU, E., and PARASCHIVOIU, I., "Adaptive curvature control grid generation algorithms for complex glaze ice shapes rans simulations," *AIAA (SciTech 2015)*, pp. 5–9, 2015.

HASSOUNA, M. S. and FARAG, A. A., "Accurate tracking of monotonically advancing fronts," in *Deformable Models*. Springer, 2007, pp. 235–258.

HATHAWAY, M. D. and WOOD, J. R., *Application of a multi-block CFD code to investigate the impact of geometry modeling on centrifugal compressor flow field predictions*. National Aeronautics and Space Administration, 1996, vol. 107198.

HERRMANN, M., "A domain decomposition parallelization of the fast marching method," DTIC Document, Tech. Rep., 2003.

HONG, S. and JEONG, W.-K., "A group-ordered fast iterative method for eikonal equations," *IEEE Transactions on Parallel and Distributed Systems*, 2016.

HONG, S. and JEONG, W.-K., "A multi-gpu fast iterative method for eikonal equations using on-the-fly adaptive domain decomposition," *Procedia Computer Science*, vol. 80, pp. 190–200, 2016.

JEONG, W.-K. and WHITAKER, R. T., "A fast iterative method for eikonal equations," *SIAM Journal on Scientific Computing*, vol. 30, no. 5, pp. 2512–2534, 2008.

JEONG, W.-K., WHITAKER, R. T. *et al.*, "A fast eikonal equation solver for parallel systems," in *SIAM conference on Computational Science and Engineering*. Citeseer, 2007.

KIM, S., "An O(N) level set method for eikonal equations," *SIAM journal on scientific computing*, vol. 22, no. 6, pp. 2178–2193, 2001.

LEVESQUE, A. T., PIGEON, A., DELOZE, T., and LAURENDEAU, E., "An overset grid 2d/infinite swept wing urans solver using recursive cartesian virtual grid method," in *53RD AIAA AEROSPACE SCIENCES MEETING*, 2015, AIAA Paper 2015–0912.

LIU, C.-B., NITHIARASU, P., and TUCKER, P., "Wall distance calculation using the eikonal/hamilton-jacobi equations on unstructured meshes: A finite element approach," *Engineering Computations*, vol. 27, no. 5, pp. 645–657, 2010.

LOEHNER, R., SHAROV, D., LUO, H., and RAMAMURTI, R., "Overlapping unstructured grids," in *39th Aerospace Sciences Meeting and Exhibit*, 2001, p. 439.

PIGEON, A., LEVESQUE, A. T., and LAURENDEAU, E., "Two-dimensional navier-stokes flow solver developments at École Polytechnique de Montréal," in *CFD Society of Canada $22^{nd}$ Annual Conference*. Toronto, CA: CFDsc, 2014.

ROGET, B. and SITARAMAN, J., "Wall distance search algorithm using voxelized marching spheres," *Journal of Computational Physics*, vol. 241, pp. 76–94, 2013.

SETHIAN, J. A. *et al.*, "Level set methods and fast marching methods," *Journal of Computing and Information Technology*, vol. 11, no. 1, pp. 1–2, 2003.

SPALART, P. R., "Philosophies and fallacies in turbulence modeling," *Progress in Aerospace Sciences*, vol. 74, pp. 1–15, 2015.

SUHS, N., ROGERS, S., and DIETZ, W., "Pegasus 5: an automated pre-processor for overset-grid cfd," in *32nd AIAA Fluid Dynamics Conference and Exhibit*, 2002, p. 3186.

TUCKER, P. G., RUMSEY, C. L., BARTELS, R. E., and BIEDRON, R. T., "Transport equation based wall distance computations aimed at flows with time-dependent geometry," NASA Report, NASA-TM-2003-212680, 2003.

TUCKER, P. G., RUMSEY, C. L., SPALART, P. R., BARTELS, R. B., and BIEDRON, R. T., "Computations of wall distances based on differential equations," *AIAA Journal*, vol. 43, no. 3, pp. 539–549, 2005.

TUCKER, P., "Assessment of geometric multilevel convergence robustness and a wall distance method for flows with multiple internal boundaries," *Applied Mathematical Modelling*, vol. 22, no. 4, pp. 293–311, 1998.

TUCKER, P., "Differential equation-based wall distance computation for DES and RANS," *Journal of Computational Physics*, vol. 190, no. 1, pp. 229–248, 2003.

TUCKER, P., "Hybrid hamilton–jacobi–poisson wall distance function model," *Computers & Fluids*, vol. 44, no. 1, pp. 130–142, 2011.

TUGURLAN, M. C., "Fast marching methods-parallel implementation and analysis," Ph.D. dissertation, Citeseer, 2008.

WIGTON, L. B., "Optimizing CFD codes and algorithms for use on Cray computers," in *Frontiers Of Computational Fluid Dynamics 1998*. World Scientific, 1998, pp. 277–292.

XIA, H. and TUCKER, P. G., "Finite volume distance field and its application to medial axis transforms," *International journal for numerical methods in engineering*, vol. 82, no. 1, p. 114, 2010.

XU, J.-L., YAN, C., and FAN, J.-J., "Computations of wall distances by solving a transport equation," *Applied Mathematics and Mechanics*, vol. 32, pp. 141–150, 2011.

YANG, J. and STERN, F., "A highly scalable massively parallel fast marching method for the eikonal equation," *Journal of Computational Physics*, 2016.

YATZIV, L., BARTESAGHI, A., and SAPIRO, G., "O (n) implementation of the fast marching algorithm," *Journal of Computational Physics*, vol. 212, no. 2, pp. 393–399, 2006.

ZHAO, H., "A fast sweeping method for eikonal equations," *Mathematics of Computation*, vol. 74, no. 250, pp. 603–627, 2005.

# APPENDIX A EXAMPLE OF THE MODIFICATION BETWEEN THE FLOW SOLVER AND WALL DISTANCE SOLVER SUBROUTINES

```
#include "preproc.h"
#include "nscode.h"

/*************************************************/
/*                                               */
/* This function calculates the Euler fluxes     */
/*                                               */
/*************************************************/

void eflux(int step, S_block *block, int level)
{
    int i,j,ii,jj,ia,iap,ian,is,ie,js,je,rimax,rjmax;
    int BCf,inci,incj,nbpt,idir,inc,incn,ijmax;
    _FLOAT g,ro,uu,vv,pp,Vn,qq,ww,ss,Vn0;
    _FLOAT *W,*Ri_W,*FLUX;
    _FLOAT fl0,fl1,fl2,fl3,fl4,fl5,fr0,fr1,fr2,fr3,fr4,fr5;
    _FLOAT *sx,*sy,ssx,ssy;
    _FLOAT *M_uu,*M_vv,Muu,Mvv;
    S_mesh *mesh;
    S_subBCface *wksubBCface;

    g = bag->gamma;

    mesh = block->mesh[level];
    inci = mesh->inci;
    incj = mesh->incj;
    nbpt = mesh->nbpt;

    W    = &(mesh->W[step][0][0][0]);
    Ri_W = &(mesh->R[step][0][0][0]);
    FLUX = &(mesh->tmp_W[step][0][0][0]);
```

```
M_uu  = &(mesh->MUU[step][0][0][0]);
M_vv  = &(mesh->MUU[step][1][0][0]);

/*loop on the two directions*/
for(idir=0;idir<=1;idir++)
{
   switch (idir)
   {
   case 0:/* i direction */
      sx = &(mesh->SIIX[step][0][0][0]);
      sy = &(mesh->SIIX[step][1][0][0]);
      inc = inci;
      incn= incj;/*inc normal to direction*/
      rimax = mesh->rimax+1;
      rjmax = mesh->rjmax;
      break;
   case 1:/* j direction */
      sx = &(mesh->SIIX[step][3][0][0]);
      sy = &(mesh->SIIX[step][4][0][0]);
      inc = incj;
      incn= inci;/*inc normal to direction*/
      rimax = mesh->rimax;
      rjmax = mesh->rjmax+1;
      break;
   }

   /*face centered loop on the restricted domain that computes
      the fluxes through each faces*/
   for (j=2;j<=rjmax;j++)
   {
       for (i=2;i<=rimax;i++)
       {
         /*define matrix index for single pointer addressing in
             the domain*/
              ia   = i*inci + j*incj;
         int  ia1  = (i+1)*inci + (j  )*incj;
```

```
int   ia3   = (i   )*inci + (j+1)*incj;
int   ia6   = (i+1)*inci + (j+1)*incj;

iap = ia - inc;

/*initialize the four fluxes of the current face*/
FLUX[ia + 0*nbpt] = 0.;
FLUX[ia + 1*nbpt] = 0.;
FLUX[ia + 2*nbpt] = 0.;
FLUX[ia + 3*nbpt] = 0.;
FLUX[ia + 4*nbpt] = 0.;     // 2.5D
FLUX[ia + 5*nbpt] = 0.;     // spalart

/*fetch face dimensions*/
ssx = sx[ia];
ssy = sy[ia];

/*compute the fluxes on the "left" side of the face*/
Muu = 0.5*(M_uu[ia]+M_uu[ia+incn]);
Mvv = 0.5*(M_vv[ia]+M_vv[ia+incn]);

ro =W[iap + 0*nbpt];
uu =W[iap + 1*nbpt];
vv =W[iap + 2*nbpt];
pp =W[iap + 3*nbpt];
ww =W[iap + 4*nbpt];
ss =W[iap + 5*nbpt];

Vn = (uu-Muu)*ssx+(vv-Mvv)*ssy;
Vn0 = (uu)*ssx+(vv)*ssy;
qq  = uu*uu + vv*vv + ww*ww ;       // 2.5 D  + ww*ww;

/*with the flow variables, compute the fluxes*/
fl0 = ro*Vn;
fl1 = ro*Vn*uu+pp*ssx;
fl2 = ro*Vn*vv+pp*ssy;
```

```
fl3 = ro*Vn*(0.5*qq+1./(g-1)*pp/ro) + pp*Vn0;//ro*Vn
    *(0.5*qq+g/(g-1.)*pp/ro);
fl4 = ro*Vn*ww; //Muu*ssx+Mvv*ssy;//      //2.5D
fl5 = ro*Vn*ss;    //spalart

/*compute the fluxes on the "right" side of the face*/
ro = W[ia + 0*nbpt];
uu = W[ia + 1*nbpt];
vv = W[ia + 2*nbpt];
pp = W[ia + 3*nbpt];
ww = W[ia + 4*nbpt];
ss = W[ia + 5*nbpt];    //spalart

Muu = 0.5*(M_uu[ia]+M_uu[ia+incn]);
Mvv = 0.5*(M_vv[ia]+M_vv[ia+incn]);
Vn0 = (uu)*ssx+(vv)*ssy;
Vn = (uu-Muu)*ssx+(vv-Mvv)*ssy;
qq  = uu*uu + vv*vv  + ww*ww ;      // 2.5 D      + ww*ww
    ;

/*with the flow variables, compute the fluxes*/
fr0 = ro*Vn;
fr1 = ro*Vn*uu+pp*ssx;
fr2 = ro*Vn*vv+pp*ssy;
fr3 = ro*Vn*(0.5*qq+1./(g-1)*pp/ro) + pp*Vn0;//ro*Vn
    *(0.5*qq+g/(g-1.)*pp/ro);
fr4 = ro*Vn*ww; //Muu*ssx+Mvv*ssy;//ro*Vn*ww; // 2.5D
fr5 = ro*Vn*ss;    //spalart

/*average the two fluxes at the face*/
FLUX[ia + 0*nbpt] += 0.5*(fl0+fr0);
FLUX[ia + 1*nbpt] += 0.5*(fl1+fr1);
FLUX[ia + 2*nbpt] += 0.5*(fl2+fr2);
FLUX[ia + 3*nbpt] += 0.5*(fl3+fr3);
FLUX[ia + 4*nbpt] += 0.5*(fl4+fr4);    //2.5 D
FLUX[ia + 5*nbpt] += 0.5*(fl5+fr5);    //spalart
```

```
    }
}

/*loop on the two faces in the current direction*/
for (BCf=2*(idir);BCf<(2*(idir+1));BCf++) /* i direction BCf
    = 0,1    j direction BCf = 2,3 */
{
    for(wksubBCface=block->faces[level]->BCface[BCf].
        firstsubBCface;  /* Loop through all subBCfaces */
         wksubBCface!=NULL;wksubBCface=wksubBCface->
             nextsubBCface)
    {
        /*If Bc type is WAL of FAR on the idir face*/
        if(wksubBCface->type!=_CON)
        {
            /*Obtain current indexes*/
            is = wksubBCface->indexBC[0];
            ie = wksubBCface->indexBC[1];
            js = wksubBCface->indexBC[2];
            je = wksubBCface->indexBC[3];

            /*add increment if it is a "max" face to get the
                last face of the domain*/
            if (wksubBCface->face_id%2)/*face # 1, 3 or 5*/
            {
                ijmax = 1;
            }
            else/*face # 2, 4 or 6*/
            {
                ijmax = 0;
            }

            /*face centered loop on the WAL or FAR face*/
            for (j=min(js,je);j<=max(js,je);j++)
            {
                for (i=min(is,ie);i<=max(is,ie);i++)
```

```
{
    /*define matrix index for single pointer
       addressing in the domain*/
    ia  = i*inci + j*incj + ijmax*inc;
    iap = ia - inc;

    /*fetch face dimensions*/
    ssx = sx[ia];
    ssy = sy[ia];

    /*average variables at the face*/
    ro = 0.5*(W[ia + 0*nbpt]+W[iap + 0*nbpt]);
    uu = 0.5*(W[ia + 1*nbpt]+W[iap + 1*nbpt]);
    vv = 0.5*(W[ia + 2*nbpt]+W[iap + 2*nbpt]);
    pp = 0.5*(W[ia + 3*nbpt]+W[iap + 3*nbpt]);
    ww = 0.5*(W[ia + 4*nbpt]+W[iap + 4*nbpt]);
    ss = 0.5*(W[ia + 5*nbpt]+W[iap + 5*nbpt]);

    Muu = 0.5*(M_uu[ia] + M_uu[ia+incn]);
    Mvv = 0.5*(M_vv[ia] + M_vv[ia+incn]);

    Vn0 = (uu)*ssx+(vv)*ssy;
    Vn = (uu-Muu)*ssx+(vv-Mvv)*ssy;
    qq = uu*uu + vv*vv + ww*ww ;    //2.5 D  + ww*
       ww;

    fl0 = ro*Vn;
    fl1 = ro*Vn*uu+pp*ssx;
    fl2 = ro*Vn*vv+pp*ssy;
    fl3 = ro*Vn*(0.5*qq+1./(g-1)*pp/ro) + pp*Vn0;
       // ro*Vn*(0.5*qq+g/(g-1.)*pp/ro);
    fl4 = ro*Vn*ww; //Muu*ssx+Mvv*ssy; //ro*Vn*ww
       ;    //2.5d
    fl5 = ro*Vn*ss;    //spalart

    FLUX[ia + 0*nbpt] = fl0;
```

```
                FLUX[ia + 1*nbpt] = fl1;
                FLUX[ia + 2*nbpt] = fl2;
                FLUX[ia + 3*nbpt] = fl3;
                FLUX[ia + 4*nbpt] = fl4; // 2.5d
                FLUX[ia + 5*nbpt] = fl5; // spalart
                        }
            }
        }
    }
}

/*cell centered loop on the restricted domain*/
for (j=2;j<=mesh->rjmax;j++)
{
    for (i=2;i<=mesh->rimax;i++)
    {
        /*define matrix index for single pointer addressing in
            the domain*/
        ia  = i*inci+j*incj;
        ian = ia + inc;

        /*convention:
            flux coming in    control volume is negative
            flux going out of control volume is positive*/
        Ri_W[ia + 0*nbpt] += (FLUX[ian + 0*nbpt] - FLUX[ia +
            0*nbpt]);
        Ri_W[ia + 1*nbpt] += (FLUX[ian + 1*nbpt] - FLUX[ia +
            1*nbpt]);
        Ri_W[ia + 2*nbpt] += (FLUX[ian + 2*nbpt] - FLUX[ia +
            2*nbpt]);
        Ri_W[ia + 3*nbpt] += (FLUX[ian + 3*nbpt] - FLUX[ia +
            3*nbpt]);
        Ri_W[ia + 4*nbpt] += (FLUX[ian + 4*nbpt] - FLUX[ia +
            4*nbpt]);
        Ri_W[ia + 5*nbpt] += (FLUX[ian + 5*nbpt] - FLUX[ia +
            5*nbpt]);
```

```
                }
            }
        }

        return;
    }
```

```
#include "preproc.h"

/*********************************************************/
/*                                                       */
/* This function computes the advective fluxes and add   */
/* them to the residuals of the Eikonal equation         */
/* Author : Anthony Bouchard                             */
/* Date : March 21st, 2016                               */
/*                                                       */
/*********************************************************/

void d2wall_flux(int level)
{
   #pragma omp parallel shared(bag) num_threads(bag->ncpus)
   {
   int i,j,inci,incj,ia,iap,ian,rimax,rjmax,blk;
   int idir,inc,BCf,is,ie,js,je,ijmax,incv;
   _FLOAT *l,*R,*U1,*U2,*sx,*sy,*ss,*FLUX,*area,rx,ry,lf,lr;
   _FLOAT Vn,fl,fr,fd,U,V,signss,*Up,*Um,*limiter,*x,*y,*xc,*yc;
   S_mesh *mesh;
   S_subBCface *wksubBCface;

   #pragma omp for schedule(static)
   for(blk=1;blk<=bag->nblks;blk++)
   {
      mesh = find_block(blk)->mesh[level];
      inci = mesh->inci;
      incj = mesh->incj;

      l     = &(mesh->lcv[0][0]);
      U1    = &(mesh->U1[0][0]);
      U2    = &(mesh->U2[0][0]);
      R     = &(mesh->Rl[0][0]);
      FLUX = &(mesh->FFl[0][0]);
      area = &(mesh->area[0][0]);
      limiter = &(mesh->limiter[0][0]);
```

```
x        = &(mesh->X [0][0][0][0]);
y        = &(mesh->X [0][1][0][0]);
xc       = &(mesh->XC[0][0][0][0]);
yc       = &(mesh->XC[0][1][0][0]);

/** Flux **/
/*loop on the two directions*/
for(idir=0;idir<=1;idir++)
{
   switch (idir)
   {
   case 0:/* i direction */
      sx = &(mesh->SIIX[0][0][0][0]);
      sy = &(mesh->SIIX[0][1][0][0]);
      inc  = inci;
      incv = incj;
      rimax = mesh->rimax+1;
      rjmax = mesh->rjmax;
      Up    = &(mesh->Up[0][0]);
      Um    = &(mesh->Um[0][0]);
      break;
   case 1:/* j direction */
      sx = &(mesh->SIIX[0][3][0][0]);
      sy = &(mesh->SIIX[0][4][0][0]);
      inc = incj;
      incv = inci;
      rimax = mesh->rimax;
      rjmax = mesh->rjmax+1;
      Up    = &(mesh->Vp[0][0]);
      Um    = &(mesh->Vm[0][0]);
      break;
   }

   /*face centered loop on the restricted domain that
      computes the fluxes through each faces*/
   for (j=2;j<=rjmax;j++)
```

```
{
    for (i=2;i<=rimax;i++)
    {
        ia  = i*inci + j*incj;
        iap = ia - inc;

        /* Gradient */
        U = 0.5*(U1[ia]+U1[iap]);
        V = 0.5*(U2[ia]+U2[iap]);

        Vn = U*sx[ia] + V*sy[ia];

        Up[ia] = 0.5*(Vn + _FABS(Vn));
        Um[ia] = 0.5*(Vn - _FABS(Vn));

        /* Left flux */
        rx = 0.5*(x[ia]+x[ia+incv]) - xc[iap];
        ry = 0.5*(y[ia]+y[ia+incv]) - yc[iap];
        lf = l[iap] + limiter[iap]*(rx*U1[iap] + ry*U2[iap
            ]);
        //lf = l[iap];
        fl = Up[ia]*lf;

        /* Right flux */
        rx = 0.5*(x[ia]+x[ia+incv]) - xc[ia];
        ry = 0.5*(y[ia]+y[ia+incv]) - yc[ia];
        lr = l[ia] + limiter[ia]*(rx*U1[ia] + ry*U2[ia]);
        //lr = l[ia];
        fr = Um[ia]*lr;

        FLUX[ia] = fl+fr;
    }
}

/*cell centered loop on the restricted domain*/
for (j=2;j<=mesh->rjmax;j++)
```

```
            {
                for (i=2;i<=mesh->rimax;i++)
                {
                    ia  = i*inci+j*incj;
                    ian = ia + inc;

                    R[ia] += (FLUX[ia] - FLUX[ian])/area[ia];
                }
            }
        }
    }
    } /*omp parallel*/

    return;
}
```

# APPENDIX B COMPUTATIONAL TIMES OF WALL DISTANCE SIMULATIONS

Table B.1 Computational times of the Euclidean distance simulations

| RUN\CASES | Icing | DLR-F6 | OMP 1 | OMP 2 | OMP 4 | MPI 1 | MPI 2 | MPI 4 | MPI 8 | MPI 16 | MPI 32 | MPI 64 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **1** | 0,9231 | 3,19 | 2,7182 | 1,3832 | 0,7294 | 194,91 | 97,97 | 49,60 | 25,68 | 12,64 | 6,36 | 3,19 |
| **2** | 0,9277 | 3,23 | 2,6926 | 1,3949 | 0,7252 | 194,77 | 98,04 | 49,65 | 25,78 | 12,88 | 6,34 | 3,23 |
| **3** | 0,9401 | 3,18 | 2,6874 | 1,3869 | 0,7185 | 194,53 | 97,98 | 49,66 | 25,80 | 13,12 | 6,67 | 3,18 |
| **4** | 0,9447 | 3,22 | 2,6792 | 1,3795 | 0,7287 | 194,56 | 98,17 | 49,58 | 25,93 | 12,90 | 6,33 | 3,22 |
| **5** | 0,9198 | 3,48 | 2,7006 | 1,3851 | 0,7336 | 194,35 | 98,28 | 49,60 | 25,69 | 12,87 | 6,53 | 3,48 |
| **Mean** | **0,9311** | **3,26** | **2,6956** | **1,3859** | **0,7271** | **194,62** | **98,09** | **49,62** | **25,77** | **12,88** | **6,45** | **3,26** |
| **1** | 0,000063524 | 0,00444 | 0,000510760 | 0,000007398 | 0,000005382 | 0,08193 | 0,01456 | 0,00031 | 0,00957 | 0,05741 | 0,00774 | 0,00444 |
| **2** | 0,000011643 | 0,00100 | 0,000009000 | 0,000080640 | 0,000003534 | 0,02067 | 0,00271 | 0,00121 | 0,00001 | 0,00001 | 0,01097 | 0,00100 |
| **3** | 0,000081303 | 0,00564 | 0,000067240 | 0,000000960 | 0,000073616 | 0,00827 | 0,01099 | 0,00162 | 0,00072 | 0,05650 | 0,05151 | 0,00564 |
| **4** | 0,000186809 | 0,00176 | 0,000268960 | 0,000041216 | 0,000002624 | 0,00447 | 0,00694 | 0,00174 | 0,02393 | 0,00027 | 0,01419 | 0,00176 |
| **5** | 0,000127740 | 0,04633 | 0,000025000 | 0,000000672 | 0,000042510 | 0,07411 | 0,03774 | 0,00024 | 0,00765 | 0,00014 | 0,00720 | 0,04633 |
| **Variance** | **0,000094204** | **0,01183** | **0,000176192** | **0,000026178** | **0,000025534** | **0,03789** | **0,01459** | **0,00102** | **0,00838** | **0,02287** | **0,01832** | **0,01183** |
| **Standard Deviation** | **0,0097** | **0,11** | **0,0133** | **0,0051** | **0,0051** | **0,19** | **0,12** | **0,03** | **0,09** | **0,15** | **0,14** | **0,11** |

Table B.2 Computational times of the projected distance simulations

| RUN\CASES | Icing | DLR-F6 | OMP 1 | OMP 2 | OMP 4 | MPI 1 | MPI 2 | MPI 4 | MPI 8 | MPI 16 | MPI 32 | MPI 64 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **1** | 1,1883 | 3,24 | 7,7326 | 3,8968 | 1,9542 | 194,20 | 97,97 | 49,58 | 25,78 | 12,61 | 6,41 | 3,24 |
| **2** | 1,1827 | 3,50 | 7,7265 | 3,9017 | 1,9634 | 194,30 | 98,08 | 49,61 | 25,75 | 12,62 | 6,33 | 3,50 |
| **3** | 1,1891 | 3,23 | 7,7305 | 3,9058 | 1,9659 | 194,55 | 98,06 | 49,54 | 25,74 | 12,63 | 6,34 | 3,23 |
| **4** | 1,1855 | 3,49 | 7,7216 | 3,9076 | 1,9598 | 194,30 | 98,07 | 49,61 | 25,72 | 12,62 | 6,38 | 3,49 |
| **5** | 1,1955 | 3,19 | 7,7224 | 3,9091 | 1,9526 | 194,27 | 97,93 | 49,54 | 25,92 | 12,70 | 6,66 | 3,19 |
| **Mean** | **1,1882** | **3,33** | **7,7267** | **3,9042** | **1,9592** | **194,32** | **98,02** | **49,57** | **25,78** | **12,64** | **6,43** | **3,33** |
| **1** | 0,000000003 | 0,00895 | 0,000034574 | 0,000054760 | 0,000024800 | 0,01578 | 0,00248 | 0,00004 | 0,00003 | 0,00055 | 0,00014 | 0,00895 |
| **2** | 0,000030579 | 0,02906 | 0,000000048 | 0,000006250 | 0,000017808 | 0,00074 | 0,00357 | 0,00093 | 0,00071 | 0,00026 | 0,00983 | 0,02906 |
| **3** | 0,000000757 | 0,00948 | 0,000014288 | 0,000002560 | 0,000045158 | 0,05150 | 0,00111 | 0,00148 | 0,00188 | 0,00002 | 0,00657 | 0,00948 |
| **4** | 0,000007289 | 0,02536 | 0,000026214 | 0,000011560 | 0,000000384 | 0,00048 | 0,00251 | 0,00131 | 0,00417 | 0,00047 | 0,00196 | 0,02536 |
| **5** | 0,000053322 | 0,01897 | 0,000018662 | 0,000024010 | 0,000043296 | 0,00274 | 0,00872 | 0,00119 | 0,01955 | 0,00429 | 0,05577 | 0,01897 |
| **Variance** | **0,000018390** | **0,01836** | **0,000018758** | **0,000019828** | **0,000026290** | **0,01425** | **0,00368** | **0,00099** | **0,00527** | **0,00112** | **0,01485** | **0,01836** |
| **Standard Deviation** | **0,0043** | **0,14** | **0,0043** | **0,0045** | **0,0051** | **0,12** | **0,06** | **0,03** | **0,07** | **0,03** | **0,12** | **0,14** |

Table B.3 Computational times of the Eikonal equation simulations

| RUN\CASES | Icing | DLR-F6 | OMP 1 | OMP 2 | OMP 4 | MPI 1 | MPI 2 | MPI 4 | MPI 8 | MPI 16 | MPI 32 | MPI 64 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **1** | 0,4784 | 8,73 | 11,7158 | 6,0023 | 3,1430 | 159,14 | 224,79 | 138,74 | 80,40 | 36,98 | 15,75 | 8,73 |
| **2** | 0,4715 | 8,32 | 11,7049 | 5,9919 | 3,1396 | 158,50 | 223,30 | 136,72 | 80,59 | 36,04 | 15,48 | 8,32 |
| **3** | 0,4814 | 8,93 | 11,7100 | 5,9887 | 3,1457 | 157,91 | 223,70 | 137,68 | 80,06 | 36,14 | 15,62 | 8,93 |
| **4** | 0,4724 | 8,70 | 11,7195 | 5,9965 | 3,1484 | 158,26 | 223,85 | 138,76 | 80,28 | 36,91 | 16,18 | 8,70 |
| **5** | 0,4753 | 8,68 | 11,7084 | 5,9959 | 3,1412 | 158,25 | 223,67 | 139,83 | 79,81 | 36,32 | 17,01 | 8,68 |
| **Mean** | **0,4758** | **8,67** | **11,7117** | **5,9951** | **3,1436** | **158,41** | **223,86** | **138,35** | **80,23** | **36,48** | **16,01** | **8,67** |
| **1** | 0,000006682 | 0,00336 | 0,000016646 | 0,000052418 | 0,000015840 | 0,52998 | 0,86118 | 0,15524 | 0,02958 | 0,25200 | 0,06656 | 0,00336 |
| **2** | 0,000018619 | 0,12390 | 0,000046512 | 0,000009986 | 0,000000336 | 0,00774 | 0,31584 | 2,64388 | 0,13104 | 0,19184 | 0,27878 | 0,12390 |
| **3** | 0,000031562 | 0,06656 | 0,000002958 | 0,000040450 | 0,000004494 | 0,25200 | 0,02624 | 0,44356 | 0,02822 | 0,11424 | 0,15054 | 0,06656 |
| **4** | 0,000011323 | 0,00078 | 0,000060528 | 0,000002074 | 0,000023232 | 0,02310 | 0,00014 | 0,17140 | 0,00270 | 0,18662 | 0,02958 | 0,00078 |
| **5** | 0,000000274 | 0,00006 | 0,000011022 | 0,000000706 | 0,000005664 | 0,02624 | 0,03686 | 2,20226 | 0,17472 | 0,02496 | 1,00400 | 0,00006 |
| **Variance** | **0,000013692** | **0,03894** | **0,000027534** | **0,000021126** | **0,000009914** | **0,16782** | **0,24806** | **1,12326** | **0,07326** | **0,15394** | **0,30590** | **0,03894** |
| **Standard Deviation** | **0,0037** | **0,20** | **0,0052** | **0,0046** | **0,0031** | **0,41** | **0,50** | **1,06** | **0,27** | **0,39** | **0,55** | **0,20** |