TECHNICAL UNIVERSITY OF CRETE

SCHOOL OF PRODUCTION ENGINEERING & MANAGEMENT

Validation of a High-Order Numerical Discretization Scheme for the Solution of the 3-D Euler Equations

By

Dimitrios P. Angelopoulos

A dissertation submitted in partial fulfilment

of the requirements for the degree of

Master of science (MSc)

Supervisor: Prof. Dr. Ioannis K. Nikolos

Chania, May 2019

"Intentionally left blank"

TECHNICAL UNIVERSITY OF CRETE

SCHOOL OF PRODUCTION ENGINEERING & MANAGEMENT

Validation of a High-Order Numerical Discretization Scheme for the Solution of the 3-D Euler Equations

By

Dimitrios P. Angelopoulos

Approved by:

Dr. Ioannis K. Nikolos
Professor
Technical University of Crete
School of Production
Engineering & Management

_____

Dr. Anargiros I. Delis
Associate Professor
Technical University of Crete
School of Production
Engineering & Management

_____

Dr. Georgios Arampatzis
Assistant Professor
Technical University of Crete
School of Production
Engineering & Management

_____

"Intentionally left blank"

"Man knows at last that he is alone in the universe's unfeeling immensity, out of which he emerged only by chance. His destiny is nowhere spelled out, nor his duty. The kingdom above or the darkness below: it is for him to choose."

Jacques Monod

"Intentionally left blank"

To my family and Nasia

"Intentionally left blank"

ABSTRACT

In this study, the application and evaluation of a high-order spatial and time discretization method for the numerical solution of 2-dimensional Euler equations is reported. An alternative high-order approach [Yan14] enhances the in-house academic solver, named *EU2*, employing the dimensionless Euler equations, discretized with a node-centered finite volume method on triangular unstructured girds, to simulate inviscid compressible flows. Most methodologies that have been developed during the past years, e.g. the discontinuous Galerkin and K-exact scheme, necessitate a non-trivial increase of the DoFs (Degrees of Freedom) and consequently a considerable increase of computational resources. Moreover, major modifications to existing CFD codes are required for their implementation. The adopted high-order scheme is based on the incorporation of additional high order terms to the reconstructed nodal values, used for the computation of the inviscid fluxes. The required higher-order derivatives are computed with the corresponding lower-order ones on the existing DoFs via a successive differentiation technique. As a result, the connectivity requirements are restricted to the first neighbouring points, overcoming the inherent constraint of the unstructured solvers to retrieve information from a wider computational stencil. The aforementioned technique was incorporated with a variable extrapolation numerical scheme, named U-MUSCL, which closely resembles the traditional MUSCL one, and was coupled with a high-order time discretization that employs a Strong Stability Preserving Runge-Kutta method (SSPRK). To assess the effectiveness of the aforementioned numerical scheme, the *EU2* solver is used against a benchmark problem having analytic solution. A satisfactory agreement is obtained, demonstrating the proposed scheme's potential to increase the solution's accuracy for a given grid density. Furthermore, a corresponding high-order formulation is extended to a 3-dimensional numerical fluid model. An elaborate construction method of 3-d computational meshes for various grid types is presented in detail for future exploitation on the numerical evaluation of equivalent 3-d high order schemes.

"Intentionally left blank"

ACKNOWLEDGEMENTS

With the completion of this study I would like to express my gratitude to those who have supported me and have contributed to this work.

First and foremost, I would like to express my deepest appreciation to my advisor Professor Ioannis K. Nikolos, for giving me the opportunity to complete this thesis and for introducing me to the fascinating 'world' of Computational Fluid Dynamics. Beyond his solid and deep scientific skills, he is also an exceptional person and character. I am very thankful for all the psychological, educational and ethical support that he provided to me, throughout all my work.

I would also like to thank my thesis committee, Associate Professor Anargiros I. Delis and Assistant Professor Georgios Arampatzis for all their useful suggestions. They have both honored me by participating to this study.

My sincere thanks also go to Dr. Georgios Lygidakis and Stavros Leloudas, my fellow lab mates that were always eager to provide their help when needed, as well as to Ioannis Thomadakis for his assistance and useful advice on the editing of this thesis.

Last but not least, nobody has been more important to me in the pursuit of this thesis than the members of my family. I would like to thank my parents and my sister, whose love and guidance are with me in whatever I pursue. But most importantly, I wish to thank my companion Nasia for all her endless love and support. This thesis is dedicated to them.

"Intentionally left blank"

CONTENTS

NOMENCLATURE

$\underline{A}$	Jacobian matrices	E_P	area of 2-d control volume of a node P
$\tilde{a}_P$	sound speed of node P	V_P	volume of 3-d control volume of a node P
c_p	constant pressure specific heat	$\hat{n}$	unit normal vector
c_v	constant volume specific heat	$\vec{W}$	conservative variables' vector
CFL	Courant-Friedrichs-Lewy number	x, y, z	Cartesian coordinates
e	energy per unit mass	γ	ideal gas constant (γ=1.4)
E	total energy per unit mass	μ	laminar viscosity
$\vec{F}, \vec{G}, \vec{J}$	Euler PDE's vectors	ρ	density
h_t	specific total enthalpy	τ_{ij}	stress tensor
I	unit matrix	***Superscripts***	
M	Mach number	inv	inviscid
p	pressure	vis	viscous
Pr	laminar Prandtl number (Pr=0.72)	$\sim$	normalized variable
q_i	thermal tensor	***Subscripts***	
R_g	gas constant (R_g=287.04 $m^2sec^{-2}K^{-1}$)	in	inlet
$\vec{S}$	source term	out	outgoing
t	time	PQ	edge connecting P and Q nodes
T	temperature	P, p	present control volume
u, v, w	components of the velocity	Q, q	adjacent control volume
$\tilde{U}$	averaged Roe value of a primitive variable	ref	*reference*

"Intentionally left blank"

INTRODUCTION

Computational Fluid Dynamics (CFD) is an ever advancing multidisciplinary scientific field, emerging from the combination of physics, numerical analysis and computer science, and providing sufficient numerical results for various types of fluid models. It originated in the early 1970s and it has developed into a very powerful technique that has been routinely applied in a wide range of industrial and non-industrial application areas ever since [Bla01, Spa16]. Notwithstanding the considerable ongoing evolution, CFD still faces several challenges that need to be addressed. Therefore, although various academic and commercial compressible flow solvers have been developed in the past years, many issues concerning the methods of grid generation, discretization, flux computation, turbulence modeling, etc., are still subjects of continuous research [Tor97, Bla01].

A primary concern that engages research activity in the CFD community, while also being the subject of this study, is the efficiency of the numerical flow solvers in producing accurate numerical solutions over more complex configurations. It is well known that the majority of the commercial unstructured CFD codes do not provide much more than a second-order accuracy. During the last decades significant efforts have been exerted for the development of higher-order spatial discretization methods, as they allow for improved accuracy in a given grid density. Nevertheless, most popular methodologies, e.g. the k-exact scheme [Bar93], necessitate for extra information beyond the first neighbouring cells to compute high-order reconstructed values. Unlike structured solvers, where node connectivity between neighbouring grid points is implied, the calculation of the higher derivatives poses limitations for the unstructured ones, due to the lack of explicit connectivity beyond the first neighbors. On the other hand, in the Discontinuous Galerkin method [Per12] - a formulation different from the classical finite volume approach - this constraint is managed by introducing extra DoFs (Degrees of Freedom) in each cell to fit a high-order polynomial solution. As a result, extra memory requirements are needed, leading unavoidably to a significant increase of computational resources. The implementation of such methodologies into existing CFD codes requires substantial modifications, especially in parallelization strategies, where the interventions on the code structure might prove to be rather laborious. Furthermore, the increased turnaround time of the numerical solution, associated with most high-order schemes, is a limiting factor for a more wide spread use as, in many practical scenarios, the computational cost is prohibiting.

The high-order scheme applied in this work relies on the incorporation of the high-order correction terms to the reconstructed nodal values, used for the computation of the inviscid fluxes. The required higher-order derivatives are computed with the corresponding lower-order ones on the existing DoFs, via a successive differentiation technique and, consequently, the connectivity requirements are restricted to the first neighbouring points [Yan14, Yan15, Yan16]. This is made feasible by exploiting the fundamental properties of the Green-Gauss theorem, overcoming the inability of unstructured flow solvers to retrieve information on a wider computational stencil. In this way, not only an improvement of the solution accuracy is achieved but the computational effort and memory requirements are retained on a reasonable level. This approach, thus, seems to be particularly appealing for incorporation to an existing CFD code, with only minor adjustments compared to other methodologies.

In the present study, a 3rd order interpolation module is applied for the numerical solution of 2-dimensional Euler Equations. This module was integrated into an in-house compressible flow solver, named *EU2*. The discrete form of the governing equations is solved with a Node-Centered Finite-Volume scheme, while for the computation of the inviscid fluxes an upwind method, applying Roe's approximate Riemann solver is employed. High-order spatial accuracy is based on U-MUSCL scheme, which closely resembles the traditional MUSCLE one [Bur05]. The time advancement of the aforementioned equations is achieved with an explicit scheme, using a Strong Stability Preserving (SSR), five stage, and fourth order Runge-Kutta method (SSPRK (5, 4)).

To demonstrate the effectiveness of the developed methodology, the *EU2* solver is used against a benchmark test case with a well-known analytical solution. This problem concerns the transport of an isentropic vortex in inviscid compressible flow. An extensive evaluation of the numerical solution was conducted, using a controlled environment through a successive grid refinement procedure for different types of triangular grids. Satisfactory results were obtained, demonstrating the scheme's potential to increase the solution's accuracy for a given grid density.

Finally, the aforementioned high-order numerical scheme is extended to 3-dimensional problems. In the context of the finite volume approach for unstructured grids, the mathematical and numerical modeling of the 3-D Euler equations is offered, where the formulation of the corresponding high-order module is reserved for future work. Moreover, a detailed demonstration of the construction method for specific types of 3-D unstructured computational meshes is presented in detail. An extensive description of the data structures of the algorithms is carried out, providing essential information of the grid features for future exploitation.

The rest of this dissertation is organized as follows. In chapter 1, a thorough representation of the 2-dimensional fluid model is undertaken, including the mathematical and numerical modeling of the 2-dimensional Euler equations. Chapter 2 is devoted to the description of the adopted high-order scheme, where the methodology for the calculation of the high-order terms, the variable-extrapolation U-MUSCL-scheme and the application of Strong Stability Preserving Runge-Kutta Method (SSPRK) are demonstrated. Chapter 3 contains the numerical results of the convergence studies against the benchmark problem of travelling vortex, including quantitative and qualitative comparisons with the analytical solution. In Chapter 4, a 3-dimensional fluid model is introduced, containing the mathematical and numerical formulation, and incorporating the proposed high-order scheme. Finally, Chapter 5 provides an analytical description of the developed algorithms that produce 3-D computational meshes.

CHAPTER 1

MATHEMATICAL AND NUMERICAL MODELING OF 2-D EULER EQUATIONS

1.1 Mathematical Modeling in 2-D

1.1.1 Principles of the Governing Equations

Fluid dynamics is concerned with the study of fluids' behavior. This is exemplified in the investigation of the interactive motion of large individual particles, i.e. molecules or atoms, partitioning the fluid. Taking under account the continuum assumption, the density of the fluid is considered high enough to be approached as a continuum. In this sense, instead of examining the fluid molecules per se, the focus is on minuscule fluid elements containing a sufficient number of particles to be regarded as a continuum. For each element, mean velocity and mean kinetic energy can be determined. This implies that velocity, temperature, density, along with other fluid quantities, are defined for each segment of the fluid.

Three conservation laws are respected for the derivation of the principal equations describing the physical properties of the fluid [Bla01]:

- Conservation of mass
- Conservation of momentum
- Conservation of energy

Conservation requires that for the three fundamental quantities – mass, momentum, and energy – their total variation inside the volume of an element is defined, primarily, as the effect of the amount of the quantity being transported across the boundary, which is called flux, as the effect of any internal forces and sources, and, finally, of the external forces acting on the volume. Two are the different terms to which flux is decomposed, the convective and the diffusive. The former owes to the convective transport, while the latter to the molecular motion present in the fluid at rest [Bla01].

In what follows, a thorough presentation of the governing 2-D Navier-Stokes equations for a compressible viscous Newtonian fluid is implemented [Lyg15], while the corresponding Euler equations are then derived.

1.1.2 Navier-Stokes 2-D Equations

A compressible viscous flow is described by the Navier-Stokes equations. Arranged into convective (inviscid), diffusive (viscous), and source terms, the differential form of the equation is written as follows:

$$\frac{\partial \vec{W}}{\partial t} + \frac{\partial \vec{F}^{inv}}{\partial x} + \frac{\partial \vec{G}^{inv}}{\partial y} - \frac{\partial \vec{F}^{vis}}{\partial x} - \frac{\partial \vec{G}^{vis}}{\partial y} = \vec{S} \tag{1.1}$$

The conservative variables' vector $\vec{W} = (\rho, \rho u, \rho v, \rho E)^T$, the inviscid flux vectors $\vec{F}^{inv}, \vec{G}^{inv}$, the viscous flux vectors $\vec{F}^{vis}, \vec{G}^{vis}$ and the vector of the source term $\vec{S}$ are expressed in terms of the

primitive variables (ρ, u, v, p). Considering the source term as equal to zero for 2-D problems, the inviscid and viscous vectors are determined as shown in the following equations [Koo00, Lyg14b].

$$\vec{F}^{inv} = \begin{pmatrix} \rho u \\ \rho u^2 + p \\ \rho uv \\ (\rho E + p)u \end{pmatrix}, \qquad \vec{G}^{inv} = \begin{pmatrix} \rho v \\ \rho vu \\ \rho v^2 + p \\ (\rho E + p)v \end{pmatrix} \tag{1.2}$$

$$\vec{F}^{vis} = \begin{pmatrix} 0 \\ \tau_{xx} \\ \tau_{xy} \\ u\tau_{xx} + v\tau_{xy} + q_x \end{pmatrix}, \vec{G}^{vis} = \begin{pmatrix} 0 \\ \tau_{yx} \\ \tau_{yy} \\ u\tau_{yx} + v\tau_{yy} + q_y \end{pmatrix} \tag{1.3}$$

The viscous stresses originate from the friction between the fluid and the surface of an element and depend on the dynamical properties of the medium. For the Newtonian fluid (including compressible viscous fluids), the shear stresses are proportional to local strain rate, the rate of change of its deformation over time. The diffusive flux vectors $\vec{F}^{vis}, \vec{G}^{vis}$ are defined from the stress tensor and calculated according to the Equation 1.4 [Hir90]

$$\tau_{ij} = \mu \left[\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} \right) - \frac{2}{3} (\nabla \cdot \vec{V}) \delta_{ij} \right] \tag{1.4}$$

where μ is the dynamic viscosity coefficient; for a perfect gas, μ heavily relies on the temperature and to a smaller extend to the pressure [Bla01]. The dynamic viscosity can be computed based on the local temperature of the fluid (in *K*) via the Sutherland formula as in 1.5 [Luo05]

$$\mu = \frac{c_1 T^{3/2}}{T + c_2} \tag{1.5}$$

where the coefficients c_1 and c_2 are equal to *1.458E-6 kg m^{-1} s^{-1} K$^{-1/2}$* and *110.4 K* respectively, e.g., the obtained dynamic viscosity for air at *300 K* equals to *1.846E-5 kg m^{-1} s^{-1}*. Based on the reference dynamic viscosity μ_{ref} and the reference temperature T_{ref}, whose values are usually used in the far field, a different formulation is applied to express dynamic viscosity, as shown in 1.6 [Luo05].

$$\mu = \mu_{ref} \left(\frac{T}{T_{ref}} \right)^{3/2} \frac{T_{ref} + c_2}{T + c_2} \tag{1.6}$$

The four conservative variables $(\rho, \rho u, \rho v, \rho E)$ are expressed by the two-dimensional Navier-Stokes equations with a set of four equations containing, though, six unknown flow field variables (ρ, u, v, E, p, T).

Two more equations are, therefore, required to complete the full set of the equation system. Assuming that in pure aerodynamics the fluid works as a perfect gas, the state equation is represented in 1.7 [Lan98]

$$p = \rho R_g T \tag{1.7}$$

where the gas constant R_g equals to *287.04* $m^2s^{-2}K^{-1}$ and it is associated with the constant pressure and volume specific heat coefficients with the following equations

$$R_g = c_p - c_v \quad , \quad \gamma = c_p/c_v \tag{1.8}$$

while these coefficients are defined as follows

$$h = c_p T \quad , \quad e = c_v T \tag{1.9}$$

where h and e are the enthalpy and internal energy of the gas per unit mass. The particular heat coefficients are regarded as constants. However, different types of gases receive different values; for air, the constant pressure specific heat coefficient c_p equals to *1004.64* $m^2s^{-2}K^{-1}$, the constant volume specific heat coefficient c_v equals to *717.6* $m^2s^{-2}K^{-1}$ and the dimensionless coefficient γ equals to *1.4* [Lan98].

In order to complete the equation set, pressure p is associated with the total energy per unit volume ρE as in 1.10 [Bla01]

$$\rho E = \rho e + \frac{1}{2}\rho(u^2 + v^2) = \rho T c_v + \frac{1}{2}\rho(u^2 + v^2) = \\ \frac{p}{R_g}c_v + \frac{1}{2}\rho(u^2 + v^2) = \frac{p}{(\gamma - 1)} + \frac{1}{2}\rho(u^2 + v^2) \tag{1.10}$$

where ρe is the internal energy per unit volume. The corresponding specific total enthalpy h_t is then associated with the pressure p and the total energy per unit volume ρE as shown in 1.11.

$$h_t = \frac{\rho E + p}{\rho} = \frac{\gamma p}{\rho(\gamma - 1)} + \frac{1}{2}(u^2 + v^2) \tag{1.11}$$

The heat flux vector (q_x, q_y) in the energy equation is defined accordingly to the stress tensor as illustrated below, where the conductivity coefficient χ depends on the dimensionless Prandtl number *Pr* [Bla01].

$$q_i = \chi \nabla T \quad , \quad \chi = \frac{\mu c_p}{Pr} \tag{1.12}$$

1.1.3 Euler 2-D Equations

While Navier-Stokes equations describe the behavior of viscous fluids, for the cases of inviscid flows, like for example for high Reynolds-number flows where the boundary layer is very thin compared to the dimensions of the body, only the corresponding flux vectors $\vec{F}^{inv}, \vec{G}^{inv}$ are considered. This leads in the so-called Euler equations depicted in 1.13 and 1.14, while the remaining terms are given in 1.7-1.11 [Bla01].

$$\frac{\partial \vec{W}}{\partial t} + \frac{\partial \vec{F}^{inv}}{\partial x} + \frac{\partial \vec{G}^{inv}}{\partial y} = \vec{S} \tag{1.13}$$

$$\vec{F}^{inv} = \begin{pmatrix} \rho u \\ \rho u^2 + p \\ \rho uv \\ (\rho E + p)u \end{pmatrix}, \qquad \vec{G}^{inv} = \begin{pmatrix} \rho v \\ \rho vu \\ \rho v^2 + p \\ (\rho E + p)v \end{pmatrix} \tag{1.14}$$

Henceforth, for the purposes of this study merely Euler equations will be taken into account.

1.1.4 Non-Dimensionalization Procedure

The differential equations representing the conservation laws are rarely solved using dimensional variables. The common practice is to write these equations in a non-dimensional form, using dimensionless quantities, obtained through a proper characteristic scale. This allows for the number reduction of the appropriate parameters contributing thus to the revelation of the relative magnitude of the various terms in the conservation equation and, consequently, of those that can be neglected [Mou16].

A dimensional variable is transformed into a non-dimensional one by dividing the variable by a quantity that has the same dimension as the original variable. Therefore, the normalization of the variables is performed utilizing a characteristic length L_{ref}, the free-stream velocity V_{ref}, the free-stream density ρ_{ref}, the free-stream dynamic viscosity μ_{ref}, and the constant volume specific heat coefficient c_v as shown in 1.15.

$$\tilde{x_i} = \frac{x_i}{L_{ref}} \, , \; \tilde{u}_i = \frac{u_i}{V_{ref}} \, , \; \tilde{\rho} = \frac{\rho}{\rho_{ref}} \, , \; \tilde{\mu} = \frac{\mu}{\mu_{ref}} \, , \; \tilde{R}_g = \frac{R_g}{c_v} = \gamma - 1 \tag{1.15}$$

Considering the previous normalizations, the rest of the variables included in Equations (1.13)-(1.14) are expressed as follows [Mun98]:

$$\tilde{p} = \frac{p}{\rho_{ref} V_{ref}^2} \, , \; \widetilde{\rho E} = \frac{\rho E}{\rho_{ref} V_{ref}^2} \, , \; \tilde{h}_t = \frac{h_t}{V_{ref}^2} \, , \; \tilde{T} = \frac{T}{V_{ref}^2 / c_v} \, , \; \tilde{t} = \frac{t}{L_{ref} / V_{ref}} \tag{1.16}$$

Moreover, the constant pressure and the constant volume specific heat coefficients are normalized ($\tilde{c}_p = \gamma$ and $\tilde{c}_v = 1$), while the perfect gas equation is transformed as:

$$p = \rho R_g T \Rightarrow \tilde{p} \, \rho_{ref} V_{ref}^2 = \tilde{\rho} \rho_{ref} \tilde{R}_g c_v \tilde{T} \left(\frac{V_{ref}^2}{c_v} \right) \Rightarrow \tilde{p} = \tilde{\rho} \tilde{R}_g \tilde{T} \Rightarrow \tilde{p} = \tilde{\rho}(\gamma - 1)\tilde{T} \tag{1.17}$$

Lastly, two additional expressions are used, concerning the computation of the local speed of sound at a node P [Lan98]

$$\tilde{a}_P = \sqrt{\gamma \tilde{R}_g \tilde{T}_P} = \sqrt{\gamma(\gamma - 1)\tilde{T}_P} = \sqrt{\frac{\gamma \tilde{p}_P}{\tilde{\rho}_P}} \tag{1.18}$$

and the computation of corresponding Mach number [Mun98]:

$$M_P = \frac{\sqrt{\tilde{u}_{P+}^2 \tilde{v}_P^2}}{\tilde{c}_P} \tag{1.19}$$

For simplification reasons, the superscript "~" denoting the normalized variables will be neglected in the following sections.

1.2 Numerical Modeling of 2-D Equations

1.2.1 Spatial Discretization

A Node-Centered Finite-Volume (NCFV) scheme is employed for the discretization of the governing equations and, consequently, for the computation of the numerical fluxes. In this approach, the computational domain is divided into a finite number of cells, from which control volumes are formed surrounding each vertex in the mesh.

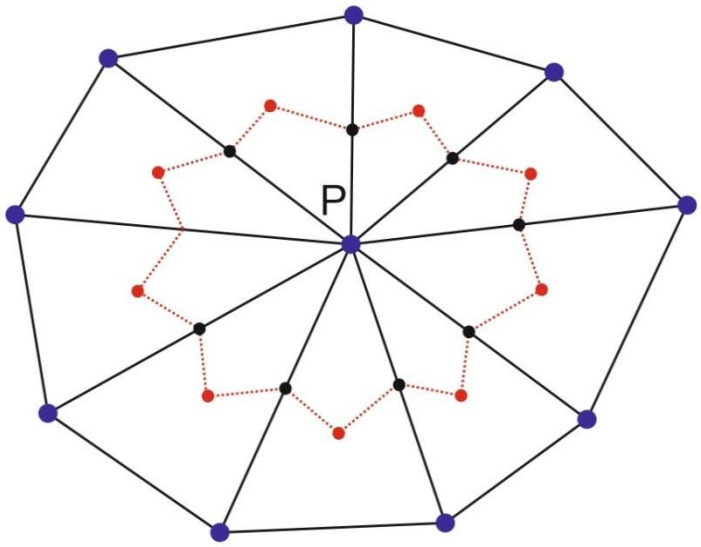


Figure 1.1: *Median control volume surrounding a node in a 2-D grid*

Consequently, these non-overlapping control volumes cover through a median dual partition the entire computational domain, which is dual to the primal mesh. The flow variables are stored at each mesh vertex. In a two-dimensional triangular mesh the median dual control volume for a node P is formed by connecting the barycenter of each neighboring triangular cell (sharing this node) to the midpoint of the corresponding cell edges, as illustrated in Figure 1.1 [Kal96, Kal05, Lyg12, Sar14]. Given this definition, the nodes of each element, which compose a control volume, divide the volume of this element to equal parts.

Taking into account the above described discretization scheme, Euler Equation 1.13 is integrated over the control volume CE_P of each node P as:

$$\iint_{CE_P} \frac{\partial \vec{W}}{\partial t} dxdy + \iint_{CE_P} \frac{\partial \vec{F}^{inv}}{\partial x} + \frac{\partial \vec{G}^{inv}}{\partial y} dxdy = \iint_{CE_P} \vec{S} dxdy \tag{1.20}$$

After the employment of the Green-Gauss divergence theorem the equation is transformed as follows

$$\iint_{CE_P} \frac{\partial \vec{W}}{\partial t} dxdy + \int_{\partial CE_P} \vec{\tilde{H}}^{inv} dl = \iint_{CE_P} \vec{S} dxdy \tag{1.21}$$

where ∂CE_P denotes the boundaries of the control volume of node P defined by the facets constructed around the edges connecting node P with each neighboring node Q. If ∂CE_{PQ} is the interfacing part of ∂CE_P and ∂CE_Q, $K_N(P)$ is the set of neighboring nodes to P, and Γ is the domain's external boundary, then ∂CE_P is defined as

$$\partial CE_P = \bigcup_{Q \in K_N(P)} \partial CE_{PQ} + (\partial CE_P \cap \Gamma) \tag{1.22}$$

where $\vec{\tilde{H}}^{inv}$ is the vector of the inviscid numerical fluxes and is evaluated at the midpoint of an edge that is connected to node P. This midpoint coincides with the interface between the adjacent control volumes of nodes P and Q connected with this edge. Utilizing the outward unit normal vector $\vec{\hat{n}}_{PQ}$ of the corresponding ∂CE_{PQ} face of the control volume, the aforementioned vectors are described as [Koo00, Kou03]

$$\vec{\tilde{H}}^{inv} = \hat{n}_{PQ,x} \vec{\tilde{F}}^{inv} + \hat{n}_{PQ,y} \vec{\tilde{G}}^{inv} \tag{1.23}$$

$$\vec{\hat{n}}_{PQ} = \frac{\vec{n}_{PQ}}{|\vec{n}_{PQ}|} = \left(\hat{n}_{PQ,x}, \hat{n}_{PQ,y}\right) \tag{1.24}$$

where $\vec{n}_{PQ}$ is defined as the vector sum of the outward normal vectors of the two facets forming ∂CE_{PQ}. Figure 1.2 presents the two normal vectors $\vec{n}_{PQ,1}$ and $\vec{n}_{PQ,2}$ that define the outward normal vector of a facet.

Thus, Equation 1.21 is transformed as follows:

$$\iint_{CE_P} \frac{\partial \vec{W}}{\partial t} dxdy + \sum_{Q \in K_N(P)} \int_{\partial CE_{PQ}} \vec{\tilde{H}}^{inv} dl + \int_{\partial CE_P \cap \Gamma} \vec{\tilde{H}}^{inv} dl = \iint_{CE_P} \vec{S} dxdy \tag{1.25}$$

Assuming that the conservative variables at node P are equal to their mean values over CE_P, the first term of 1.25 becomes:

$$\iint_{CE_P} \frac{\partial \vec{W}}{\partial t} dxdy = \left(\frac{d\vec{W}}{dt}\right)_P \iint_{CE_P} dxdy = \left(\frac{d\vec{W}}{dt}\right)_P E_P \tag{1.26}$$

Expressing the integrals of the numerical fluxes as summations of fluxes through the faces composing the control volume of node P, Equation 1.25 is transformed as

$$\left(\frac{d\vec{W}}{dt}\right)_P E_P + \sum_{Q \in K_N(P)} \vec{\Phi}_{PQ}^{inv} + \sum_{(K_{out} \in \partial CE_P \cap \Gamma)} \vec{\Phi}_{P,out}^{inv} = \iint_{CE_P} \vec{S} dxdy \tag{1.27}$$

$$\vec{\Phi}_{PQ}^{inv} = \int_{\partial CE_{PQ}} \vec{H}^{inv}\, dl = \vec{f}\left(\vec{W}_{PQ}^{L}, \vec{W}_{PQ}^{R}, \vec{n}_{PQ}\right)$$
$$\vec{\Phi}_{P,out}^{inv} = \int_{\partial CE_{P} \cap \Gamma} \vec{H}^{inv}\, dl = \vec{f}\left(\vec{W}_{P}, \vec{W}_{out}, \vec{n}_{out}\right) \tag{1.28}$$

where $\vec{W}_{PQ}^{L}$ and $\vec{W}_{PQ}^{R}$ are the vectors of the conservative variables on the left and right side of the edge PQ respectively, while $\vec{W}_{out}$ is the corresponding vector on the boundary of the flow domain.

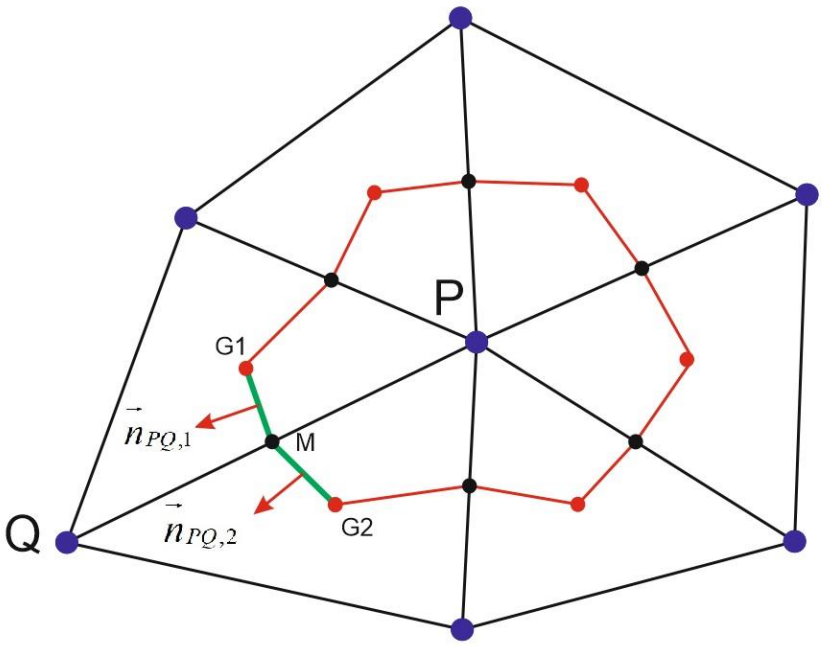


***Figure 1.2**: Outward normal vectors at an interface among nodes P and Q*

1.2.2 Numerical Fluxes

First Order Accurate Scheme

The convective numerical fluxes of the flow equations are computed by employing an upwind scheme, which distinguishes between upstream and downstream influences, i.e. the wave propagation directions, considering the physical properties of the Euler equations. A one-dimensional Riemann problem, which is based on the solution of the locally one-dimensional Euler equations for discontinuous (left and right) states at an interface is utilized and applied in the direction of the normal vector for each face of the control volume of a node P. Since the computational effort of the exact solution of the Riemann problem would require excessive numerical effort [Lan98], Roe's approximate Riemann solver [Roe81] is employed for the evaluation of the inviscid fluxes at the midpoint of edge PQ as in 1.29

$$\vec{\Phi}_{PQ}^{inv} = \frac{1}{2}\left(\vec{H}^{inv}\left(\vec{W}_{PQ}^{L}, \vec{n}_{PQ}\right) + \vec{H}^{inv}\left(\vec{W}_{PQ}^{R}, \vec{n}_{PQ}\right)\right) - \frac{1}{2}\left|\tilde{\underline{A}}_{PQ}\right|\left(\vec{W}_{PQ}^{R} - \vec{W}_{PQ}^{L}\right) \tag{1.29}$$

where $\underline{\tilde{A}}$ is the Jacobian matrix[1] of the convective flux vector $\vec{H}^{inv}$, which is evaluated at the midpoint of the corresponding edge PQ by utilizing Roe's averaged values of the primitive variables (denoted with tilde ˜) [Roe81, Ven95, Lan98, Koo00, Kou03] and is defined as in 1.30

$$\vec{\tilde{U}}_{PQ} = \frac{\sqrt{\rho_L}\,\vec{U}_L + \sqrt{\rho_R}\,\vec{U}_R}{\sqrt{\rho_L} + \sqrt{\rho_R}} \tag{1.30}$$

where $\vec{U}_L$ and $\vec{U}_R$ in first order accurate schemes are the values of primitive variables at the left and right side of edge PQ respectively.

Based on the following formula (1.31), Equation 1.29 is transformed in its equivalent in 1.32 [Roe81, Lan98]:

$$\vec{H}^{inv}(\vec{W}_{PQ}^{R}) - \vec{H}^{inv}(\vec{W}_{PQ}^{L}) = \underline{\tilde{A}}_{PQ}^{-}(\vec{W}_{PQ}^{R} - \vec{W}_{PQ}^{L}) \tag{1.31}$$

$$\vec{\Phi}_{PQ}^{inv} = \vec{H}^{inv}(\vec{W}_{PQ}^{L}, \vec{n}_{PQ}) + \underline{\tilde{A}}_{PQ}^{-}(\vec{W}_{PQ}^{R} - \vec{W}_{PQ}^{L}) \tag{1.32}$$

On account of computational effort and memory requirements on unstructured grids, edge-wise data structure of the algorithm is used, as a more sophisticated data structure. Within this approach the solver receives information from the examined mesh as sets of nodes connected by an edge. Along these lines, the evaluation of the convective fluxes for all the mesh nodes is achieved with a single edge-loop, since no information is needed about the cell topology [Lyg14a, Lyg15].

Second-Order Accurate Scheme

In a second-order accurate scheme, left and right states of an edge PQ are reconstructed with the Taylor series expansions which consider the corresponding values of more neighboring mesh nodes during the computation of the numerical fluxes. The incorporated second-order accurate scheme is based on the MUSCL (Monotonic Upstream Scheme for Conservation Laws) reconstruction of the primitives or conservative variables. In order to alleviate the generation of oscillations and spurious solutions in regions of high-order gradients such as shocks, slope limiters are utilized to achieve a monotonicity preserving scheme (Van Albada -Van Leer [VanA82], Min-mod [Swe84]). Thus, the left and right states for a primitive or a conservative variable U at the midpoint of an edge PQ are approximated as [Bar92, And94, Bla01, ANSYS06, Sar14]:

$$\begin{aligned} U_{PQ}^{L} &= U_P + \frac{1}{2}\cdot(\nabla U)^{L}\cdot\vec{r}_{PQ} \\ U_{PQ}^{R} &= U_Q - \frac{1}{2}\cdot(\nabla U)^{R}\cdot\vec{r}_{PQ} \end{aligned} \tag{1.33}$$

The first R/H side terms are the left and right nodes' values of variable U and $\vec{r}_{PQ}$ is the vector connecting these nodes. The extrapolation gradients $(\nabla U)^{L}$ and $(\nabla U)^{R}$ are computed using the

[1] Information on the computation of the Jacobian matrix is given in Appendix A.

gradients $(\nabla U)_P$ and $(\nabla U)_Q$ at the nodes P and Q respectively. The evaluation of these derivatives employs the element-by-element approach [Bar92]. In this case, the gradient for a node P (where P is the common vertex of the neighboring triangles T), is described as [Bar92]

$$(\nabla U)_P = \frac{1}{E_P} \sum_{T \in K_T(P)} \frac{E_T}{3} (\nabla U)_T \tag{1.34}$$

where E_P and E_T are the areas of the control volume of node P and adjacent element T. However, because of utilizing the edge-based data structure of the algorithm [Bar92, Bla01], derived by the Green-Gauss linear representation method, an equivalent expression as in 1.35 is more appropriate:

$$(\nabla U)_P = \frac{1}{E_P} \sum_{Q \in K_N(P)} \frac{1}{2} (U_P + U_Q) \cdot \vec{n}_{PQ} \tag{1.35}$$

In case of a boundary node (Figure 1.3) the previous equation is modified to include also the boundary interfaces as follows [Lyg13]:

$$(\nabla U)_P = \frac{1}{E_P} \left(\sum_{Q \in K_N(P)} \frac{1}{2} (U_P + U_Q) \cdot \vec{n}_{PQ} + \sum_{(K_{out} \in \partial C E_P \cap \Gamma)} U_P \cdot \vec{n}_{out} \right) \tag{1.36}$$

1.2.3 Boundary Conditions

Numerical flow simulations are always restricted to a specific part of the real physical domain. Thus, artificial boundaries are formed with the truncation of the computational domain and, correspondingly, physical quantity values have to be specified. Types of boundary conditions that encountered in the numerical solution are wall, inlet, outlet and symmetry boundaries. Consequently, the contribution of the boundary surfaces is also taken into account in the flux balance of the corresponding nodes.

With respect to the wall boundary nodes, a free-slip boundary condition is employed for the solution of the Euler equations, regarding inviscid flows. The free-slip condition is implemented implicitly, by adding a flux with zero normal to the boundary face velocity V_n described as [Mav94]

$$V_n = \vec{V} \cdot \vec{\hat{n}}_{out} = 0 \tag{1.37}$$

where $\vec{\hat{n}}_{out} = (\hat{n}_{out,x}, \hat{n}_{out,y})$ is the normal to the boundary face unitary vector (outward-positive). An example of such vectors is presented in Figure 1.3 for a boundary node.

Finally, the added free-slip convective flux is calculated as in the following equation:

$$\vec{H}_{freeslip} = \begin{pmatrix} \rho V_n \\ \rho u V_n + p\hat{n}_{out,x} \\ \rho v V_n + p\hat{n}_{out,y} \\ (\rho E + p) V_n \end{pmatrix} = \begin{pmatrix} 0 \\ p\hat{n}_{out,x} \\ p\hat{n}_{out,y} \\ 0 \end{pmatrix} \tag{1.38}$$

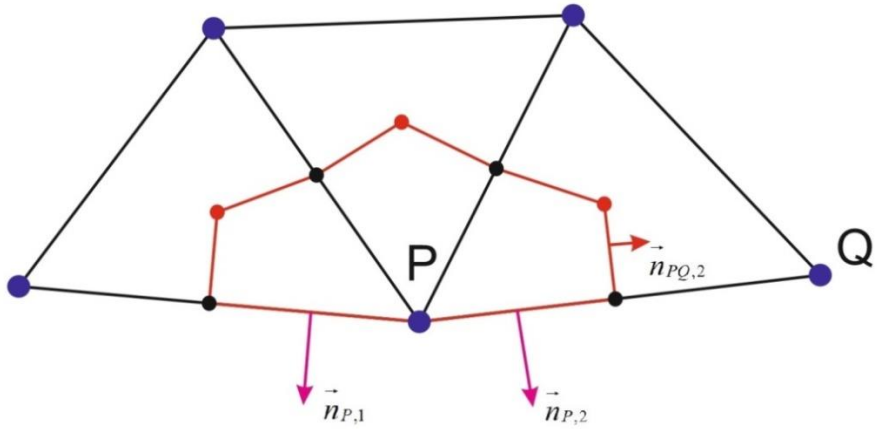


Figure 1.3: *Normal outward vectors for a boundary node.*

Regarding inlet boundary faces, a one one-dimensional Riemann problem is considered between the face's midpoint and the far-field to compute the convective fluxes and then to distribute them to the corresponding boundary nodes. Employing the Steger-Warming scheme [Ste81, Lan98], Equation 1.39 is obtained

$$\vec{H}_{K,out}^{inv} = \underline{\tilde{A}}_K^{+}\vec{W}_K + \underline{\tilde{A}}_K^{-}\vec{W}_{out} \tag{1.39}$$

where subscript K denotes the midpoint of the boundary face, while subscript *out* denotes the far field; the values of the variables of vector $\vec{W}_{out}$ are obtained either from the far field or the boundary midpoint depending on the type of the flow [Hir90, Bla01].

With reference to outlet boundary faces, the computation of the convective fluxes is performed on the inlet ones in a similar manner; depending on the type of the flow, the values of the variables of vector $\vec{W}_{out}$ are obtained by implementing a one-dimensional Riemann problem between the midpoint face and the far-filed. In the case of a symmetry surface, free-slip boundary conditions are imposed to the flow equations similarly to these for solid free-slip wall boundaries.

1.2.4 Time Integration

The governing equations require a separate discretization in space and time. For time integration an explicit scheme is incorporated for solving the Euler equations. A widely used method is the multistage time-stepping Runge-Kutta scheme, where the solution advances in several stages and the residual is evaluated at intermediate states [Kal96, Bla01, Lyg14a].

Applying time discretization leads to the transformation of Equation 1.27 into the following one

$$-E_P\left(\frac{d\vec{W}}{dt}\right)_P = -E_P\frac{\Delta\vec{W}_P^{n+1}}{\Delta t_P} =$$

$$\sum_{Q\in K_N(P)} \vec{\Phi}_{PQ}^{inv} + \sum_{(K_{out}\in\partial CE_P\cap\Gamma)} \vec{\Phi}_{P,out}^{inv} - \vec{S}_P E_P = \vec{R}_P^n \tag{1.40}$$

where Δt_P is the local time step at node P and is computed as [Kim03, Lyg11]

$$\Delta t_P = CFL \cdot \frac{0.5\alpha_{min\ l\ edge,P}}{\left|\vec{U}_P\right| + a_P} \tag{1.41}$$

where $\left|\vec{U}_P\right|$ is the value of velocity at node P, a_P is the speed of sound evaluated on the same node and $\alpha_{min\ l\ edge,P}$ is the length of the shortest edge connected to P.

The local time stepping constitutes a typical time convergence acceleration methodology to the steady state solution, which amounts to advancing the solution in each control volume with the maximum allowable time step [Bla01]; in case a global time step is required, it is defined as the smallest of the local time steps of all the nodes in the mesh.

When a second-order scheme is implemented a four Runge-Kutta (RK (4)) method is employed to solve Equation 1.40. It occurs iteratively as follows [Bla01, Lyg15, Lal88 and Sor03]

$$\begin{aligned} \vec{W}_P^{n+1,0} &= \vec{W}_P^{n} \\ \vec{W}_P^{n+1,k} &= \vec{W}_P^{n} - \alpha_k \frac{\Delta t_P}{E_P} \vec{R}\left(\vec{W}_P^{n+1,k-1}\right), \qquad k = 1, \dots, 4 \\ \vec{W}_P^{n+1} &= \vec{W}_P^{n+1,4} \end{aligned} \tag{1.42}$$

where k is the number of current internal stage of the scheme. Constants α_1, α_2, α_3 and α_4 of the method with values *0.11*, *0.26*, *0.5* and *1.0* respectively, are used attributing second-order temporal accuracy to the procedure [Bla01].

Given the relatively low convergence rate of explicit methods, an acceleration method aiming at increasing the maximum possible time step is required. This occurs by introducing a certain amount of implicitness in the explicit scheme allowing for the utilization of larger *CFL* numbers. This technique, termed *implicit residual smoothing,* modifies the residual for a node P and is defined as

$$R_P^{m+1} = \frac{R_P^0 + \varepsilon \sum_{j=1}^{l} R_{Q_j}^m}{1 + \varepsilon \sum_{j=1}^{l} 1} \tag{1.43}$$

where Q_j are the neighboring nodes of node P and ε is a coefficient with typical values *0.5-0.8*, defining the blending degree [Bla01].

"Intentionally left blank"

CHAPTER 2

HIGH-ORDER NUMERICAL SCHEME

2.1 Introduction to High-Order Formulation

In recent years, the focus of interest has been shifted to high-order scheme development, owing to the fact that such approaches offer greater accuracy at a permissible computational cost [Yan14, Yan15]. However, despite the rapid reduction of the truncation error in higher order methods compared to the lower order ones, the former scheme is utterly more cost effective. Another potential constraint against the spread of high order schemes relates to the substantial code modifications required for its implementation, especially with respect to the unstructured grid procedures [Yan15, Yan16].

Within the higher order scheme development, emphasis has been placed on variants of the Discontinuous Galerkin method (DG). According to this method additional degrees of freedom (DOFs) are introduced within a given cell to fit a high order polynomial to the solution. In this framework, structured connectivity is recovered within each cell, as shown in Figures 2.1 and 2.2 [Per12, Yan14].

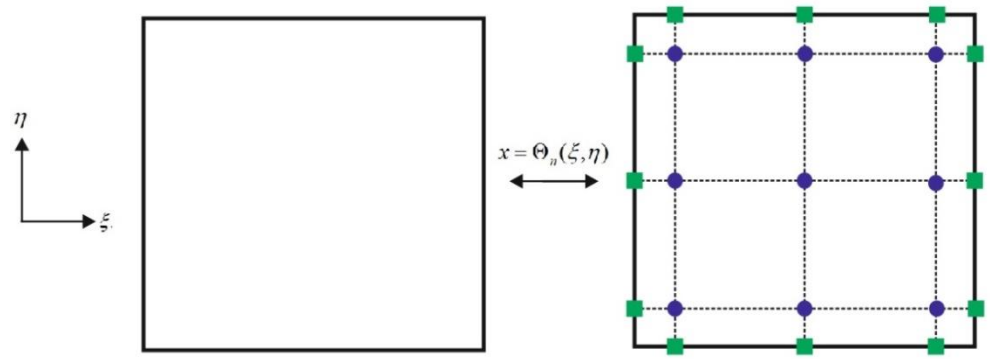


Figure 2.1: *Stencil for the DG method in a quadrilateral cell*

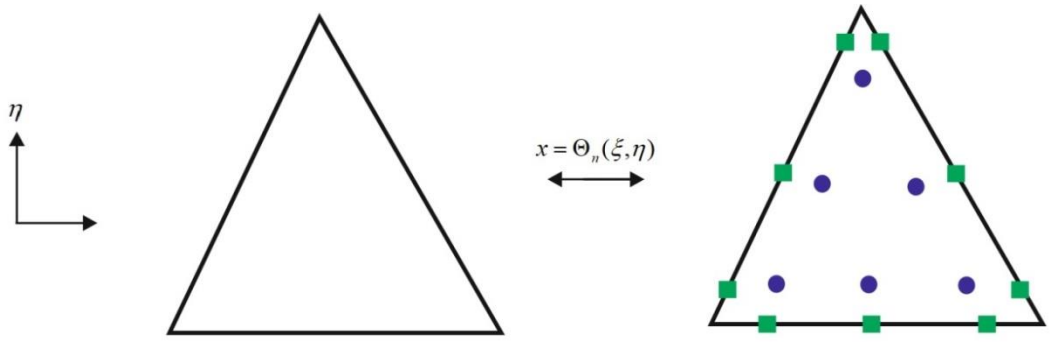


Figure 2.2: *Stencil for the DG method in a triangular cell*

A fundamental difference between the DG methods and the traditional finite volume one lies on the fact that, due to the tight linking of DOFs within a cell, the mass matrix is a full matrix rather than a diagonal one, and needs to be stored and inverted implicitly [Per12]. This means that the

application of DG methods into currently existing CFD codes would demand in depth code transformations. Additionally, for such a high-order scheme to advance and to be incorporated into new production codes, it would be at the expense of considerable verification and validation efforts.

In this chapter, a modular high-order scheme with low-dissipation flux difference-splitting is applied [Yan16]. According to this approach, no increase in DOFs within each cell occurs, unlike the k-exact finite volume method [Bar93], which leads into the need for great amounts of extra storage. The core idea is the achievement of high-order accuracy by adding high-order correction terms to the governing equations and by not introducing extra variables. The main advantage of this approach is its smooth application to any existing code, since only minor modifications are required.

As already mentioned in the Introduction, the development of the presented high-order scheme was founded on the academic *EU2* CFD code, which is an in-house unstructured-grid, Node-Centered Finite-Volume flow solver. The code then integrates a generic third-order interpolation module, which, despite the fact that it is not formally third-order accurate on arbitrary meshes, offers a significant improvement on the accuracy over the existing second-order scheme.

The presentation of this high-order method begins with the introduction of a scalar advection equation problem in 1-D, where it is demonstrated how the high-order accuracy is achieved by calculating the high-order terms. In what follows, the presentation gradually escalates into a 2-D formulation in order to be implemented into a variable-extrapolation formulation named U-MUSCL. Finally, a higher-order time discretization scheme is introduced with the employment of an explicitly Strong Stability-Preserving Runge-Kutta method (SSPRK).

To begin with, the one-dimensional scalar advection equation considered is the following

$$\frac{\partial u}{\partial t} + \frac{\partial (cu)}{\partial x} = 0 \tag{2.1}$$

where u is a scalar quantity and c represents the velocity. Figure 2.3 below depicts part of the computational domain for a node-centered scheme where the numerical figures 1-5 sign the data points where the values of the scalar are located, while A-F stand for the faces (grid points) among the control volumes.

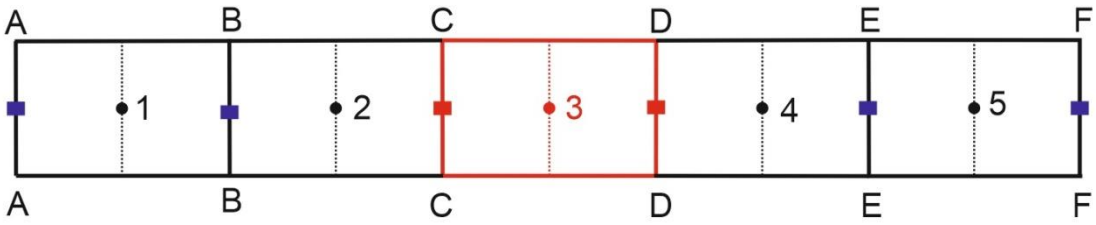


Figure 2.3: *Problem of 1-D scalar equation*

After integrating Equation 2.1 over a control volume (for instance, the marked control volume 3 in Figure 2.3) and applying Gauss theorem the result is shown in 2.2.

$$\int_V \frac{\partial u}{\partial t} dV + \int_V \frac{\partial (cu)}{\partial x} dV = 0 \rightarrow \int_V \frac{\partial u}{\partial t} dV + \oint_{\partial V} (cu)\, \hat{n} dS = 0 \tag{2.2}$$

Assuming the flow of the quantity follows a certain direction, the resulting equation is

$$\frac{\partial u}{\partial t} V + (cu)_D S_D - (cu)_C S_C = 0 \tag{2.3}$$

In order to compute the fluxes at faces C and D, the values of the scalar quantity at the aforementioned faces need to be calculated. Given the values at the data points, it is possible to manage this by applying the Taylor series expansions. The implementation of this procedure to the face C leads to Equations 2.4:

$$\begin{aligned} u_C^L &= u_2 + \frac{du_2}{dx}(x_C - x_2) + \frac{1}{2!}\frac{d^2u_2}{dx^2}(x_C - x_2)^2 + \frac{1}{3!}\frac{d^3u_2}{dx^3}(x_C - x_2)^3 + O((x_C - x_2)^4) \\ u_C^R &= u_3 + \frac{du_3}{dx}(x_C - x_3) + \frac{1}{2!}\frac{d^2u_3}{dx^2}(x_C - x_3)^2 + \frac{1}{3!}\frac{d^3u_3}{dx^3}(x_C - x_3)^3 + O((x_C - x_3)^4) \end{aligned} \tag{2.4}$$

The superscript "R" denotes the right state, while "L" denotes the left. What is more, for face D the result is shown below.

$$\begin{aligned} u_D^L &= u_3 + \frac{du_3}{dx}(x_D - x_3) + \frac{1}{2!}\frac{d^2u_3}{dx^2}(x_D - x_3)^2 + \frac{1}{3!}\frac{d^3u_3}{dx^3}(x_D - x_3)^3 + O((x_D - x_3)^4) \\ u_D^R &= u_4 + \frac{du_4}{dx}(x_D - x_4) + \frac{1}{2!}\frac{d^2u_4}{dx^2}(x_D - x_4)^2 + \frac{1}{3!}\frac{d^3u_4}{dx^3}(x_D - x_4)^3 + O((x_D - x_4)^4) \end{aligned} \tag{2.5}$$

Considering a uniform mesh for reasons of simplicity where $h = (x_C - x_2) = -(x_C - x_3)$, Equations 2.4 (for example for face C) are transformed as below.

$$\begin{aligned} u_C^L &= u_2 + \frac{du_2}{dx}h + \frac{1}{2!}\frac{d^2u_2}{dx^2}h^2 + \frac{1}{3!}\frac{d^3u_2}{dx^3}h^3 + O(h^4) \\ u_C^R &= u_3 - \frac{du_3}{dx}h + \frac{1}{2!}\frac{d^2u_3}{dx^2}h^2 - \frac{1}{3!}\frac{d^3u_3}{dx^3}h^3 + O(h^4) \end{aligned} \tag{2.6}$$

What can be noted from the formulation above is that a high-order accuracy of the solution is feasible provided that the derivatives at each cell center $\frac{du}{dx}$, $\frac{d^2u}{dx^2}$, $\frac{d^3u}{dx^3}$ can be computed. Nevertheless, for an unstructured grid only the first neighboring points are available, raising considerable difficulties in computing the higher-order derivatives. To achieve this, the method under discussion exploits the Green-Gauss theorem as demonstrated in what follows. The theorem states that

$$\begin{aligned} \int_V \nabla u\, dV &= \oint_{\partial V} u\, \hat{n}\, dS \\ \nabla u &= \frac{1}{V} \sum_{faces} u\, \hat{n}\, dS \end{aligned} \tag{2.7}$$

where ΔS is the surface area of each face and $\hat{n}$ is the corresponding surface unitary normal vector (outward-positive). After calculating the first derivatives, as it is performed in a standard second-order scheme, the above expression regarding to the gradient of the 3rd grid point, leads to the following expression.

$$\frac{du_3}{dx} = \frac{1}{2h}(u_D - u_C) = \frac{1}{2h}\left[\frac{1}{2}(u_4 + u_3) - \frac{1}{2}(u_3 + u_2)\right] = \frac{1}{4h}(u_4 - u_2) \tag{2.8}$$

The above formulation computes only the first derivatives on unstructured grids. Proceeding with the computation of the higher derivatives, the method heavily relies on the definition of the Green-Gauss theorem [Yang14]. According to it, the computed gradient is considered a volume-averaged value, rather than a local value. Given this, the same procedure is iterated to calculate the second derivatives, resulting to the following expression.

$$\begin{aligned}\frac{d^2u_3}{dx^2} &= \frac{d}{dx}\left(\frac{du_3}{dx}\right) = \frac{1}{2h}\left(\frac{du_D}{dx} - \frac{du_C}{dx}\right) = \frac{1}{2h}\left[\frac{1}{2}\left(\frac{du_4}{dx} + \frac{du_3}{dx}\right) - \frac{1}{2}\left(\frac{du_3}{dx} + \frac{du_2}{dx}\right)\right] \\ &= \frac{1}{4h}\left(\frac{du_4}{dx} - \frac{du_2}{dx}\right)\end{aligned} \tag{2.9}$$

Once the first derivative field is built, the computation of the second derivatives field is possible. Hence, given the derivatives of 2nd and 4th data point

$$\frac{du_2}{dx} = \frac{1}{4h}(u_3 - u_1) \quad , \quad \frac{du_4}{dx} = \frac{1}{4h}(u_5 - u_3) \tag{2.10}$$

the final outcome of Equation 2.9 is formulated as follows:

$$\frac{d^2u_3}{dx^2} = \frac{1}{4h}\left(\frac{1}{4h}(u_5 - u_3) - \frac{1}{4h}(u_3 - u_1)\right) = \frac{1}{16h^2}(u_5 - 2u_3 + u_1) \tag{2.11}$$

Relying on the second derivatives values, the same process is repeated in order for the third derivatives to be calculated. The result is:

$$\begin{aligned}\frac{d^3u_3}{dx^3} &= \frac{d}{dx}\left(\frac{d^2u_3}{dx^2}\right) = \frac{1}{2h}\left(\frac{d^2u_D}{dx^2} - \frac{d^2u_C}{dx^2}\right) = \frac{1}{2h}\left[\frac{1}{2}\left(\frac{d^2u_4}{dx^2} + \frac{d^2u_3}{dx^2}\right) - \frac{1}{2}\left(\frac{d^2u_3}{dx^2} + \frac{d^2u_2}{dx^2}\right)\right] \\ &= \frac{1}{4h}\left(\frac{d^2u_4}{dx^2} - \frac{d^2u_2}{dx^2}\right)\end{aligned} \tag{2.12}$$

As soon as the second derivative field is built, it is feasible again to calculate the third set of derivatives. Given the following second derivatives

$$\frac{d^2u_2}{dx^2} = \frac{1}{16h^2}(u_4 - 2u_2 + u_0) \quad , \frac{d^2u_4}{dx^2} = \frac{1}{16h^2}(u_6 - 2u_4 + u_2) \tag{2.13}$$

the combinatory process leads to Equation 2.14:

$$\frac{d^3u_3}{dx^3} = \frac{1}{4h}\left(\frac{1}{16h^2}(u_6 - 2u_4 + u_2) - \frac{1}{16h^2}(u_4 - 2u_2 + u_0)\right) = \frac{1}{64h^3}(u_6 - 3u_4 + 3u_2 - u_0) \tag{2.14}$$

All in all, within an upwind scheme the problem under discussion concerning the reconstructed values of control volume 3 receives the following representation for 3rd order accuracy.

$$
\begin{aligned}
u_C^L &= u_2 + \frac{du_2}{dx}h + \frac{1}{2!}\frac{d^2u_2}{dx^2}h^2 = u_2 + \frac{1}{4}(u_3 - u_1) + \frac{1}{32}(u_4 - 2u_2 + u_0) \\
u_C^R &= u_3 - \frac{du_3}{dx}h + \frac{1}{2!}\frac{d^2u_3}{dx^2}h^2 = u_3 - \frac{1}{4}(u_4 - u_2) + \frac{1}{32}(u_5 - 2u_3 + u_1) \\
u_D^L &= u_3 + \frac{du_3}{dx}h + \frac{1}{2!}\frac{d^2u_3}{dx^2}h^2 = u_3 + \frac{1}{4}(u_4 - u_2) + \frac{1}{32}(u_5 - 2u_3 + u_1) \\
u_D^R &= u_4 - \frac{du_4}{dx}h + \frac{1}{2!}\frac{d^2u_4}{dx^2}h^2 = u_4 - \frac{1}{4}(u_5 - u_3) + \frac{1}{32}(u_6 - 2u_4 + u_2)
\end{aligned}
\tag{2.15}
$$

As evident, a wider stencil of cells can be incorporated in the process of the derivatives' computation. The data points of the cells involved in the computation of each derivative order is demonstrated in Figures 2.4 -2.7, where the colored points indicate the first, second, third, and fourth neighboring cells respectively.

With the use of the method above, each higher accuracy order stems from the addition of the corresponding high order correction term. All the aforementioned derivative computations fall under the same recursive pattern. High-order accuracy can be obtained with the successive implementation of the Green-Gauss theorem. It is worth noting that a research by Diskin and Thomas [Dis07] shows that implementing the Green-Gauss formula results in accuracy deterioration by one order for every consecutive application of the formula. Therefore, the introduced procedure may not give high-order accuracy on general unstructured grids; nonetheless, it is easier to apply onto a standing CFD code than the DG method, so that the accuracy of the base second-order scheme is improved.

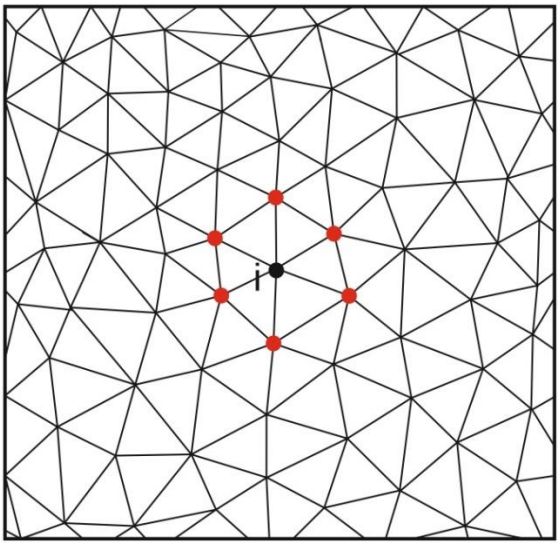


Figure 2.4: *Stencil for the first derivative*

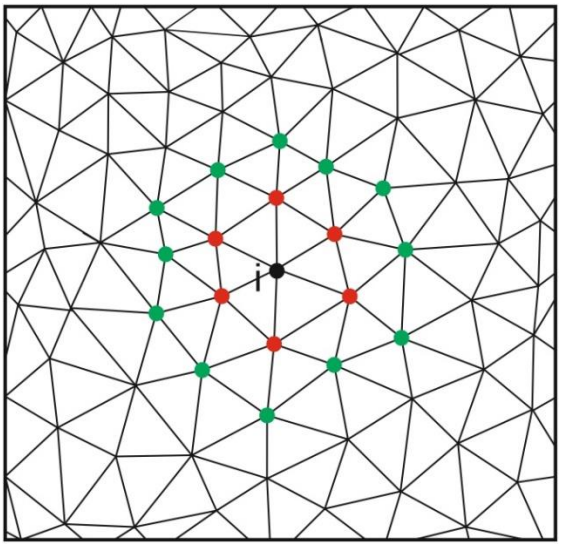


Figure 2.5: *Stencil for the second derivative*

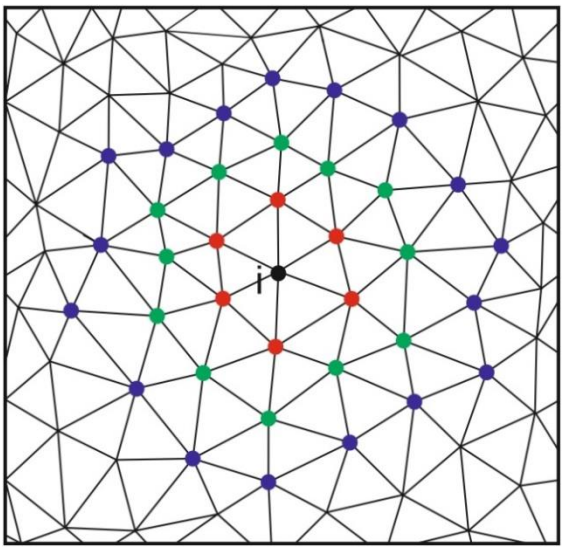


Figure 2.6: *Stencil for the third derivative*

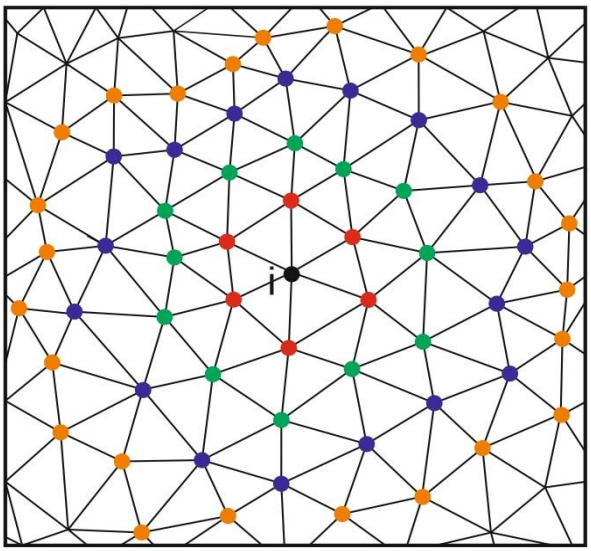


Figure 2.7: *Stencil for the fourth derivative*

2.2 Derivation of High-Order Accuracy for 2-D Problems

As a starting point the traditional second-order scheme with the functional form, as in 1.33, is presented below:

$$U^2 = U(x_0, y_0) + \frac{\partial U}{\partial x}(x_i - x_0) + \frac{\partial U}{\partial y}(y_i - y_0) \tag{2.16}$$

Based on the description outlined in the section 2.1 a Taylor series expansion is applied to achieve a higher order of accuracy. Therefore, for a third-order scheme the functional form is:

$$\begin{aligned} U^h = U(x_0, y_0) + \frac{\partial U}{\partial x}(x_i - x_0) + \frac{\partial U}{\partial y}(y_i - y_0) + \frac{1}{!2}\left[\frac{\partial^2 U}{\partial x^2}(x_i - x_0)^2\right. \\ \left. + \frac{\partial^2 U}{\partial y^2}(y_i - y_0)^2 + 2\frac{\partial^2 U}{\partial x \partial y}(x_i - x_0)(y_i - y_0)\right] \end{aligned} \tag{2.17}$$

It is noticeable that the first three terms on the right-hand side are the U value of the 2nd order scheme, as exemplified in Equation 2.18, where the high-order correction term is presented [Yang15]:

$$\Delta U^{h-2} = U^h - U^2 = \frac{1}{2}\left[\frac{\partial^2 U}{\partial x^2}(x_i - x_0)^2 + \frac{\partial^2 U}{\partial y^2}(y_i - y_0)^2 + 2\frac{\partial^2 U}{\partial x \partial y}(x_i - x_0)(y_i - y_0)\right] \tag{2.18}$$

The Green-Gauss theorem for a function f states:

$$\frac{\partial f}{\partial x} = \frac{1}{E}\oint_{\partial E} f\,\hat{n}_x dl \quad , \quad \frac{\partial f}{\partial y} = \frac{1}{E}\oint_{\partial E} f\,\hat{n}_y dl \tag{2.19}$$

Having computed the first derivatives, a successive application of the Green-Gauss theorem is performed, so as to calculate the terms in 2.18 as depicted in 2.20:

$$\frac{\partial^2 U}{\partial x^2} = \frac{1}{E}\oint_{\partial E} \frac{\partial U}{\partial x}\,\hat{n}_x dl\,, \qquad \frac{\partial^2 U}{\partial y^2} = \frac{1}{E}\oint_{\partial E} \frac{\partial U}{\partial y}\,\hat{n}_y dl\,, \qquad \frac{\partial^2 U}{\partial x \partial y} = \frac{1}{E}\oint_{\partial E} \frac{\partial U}{\partial y}\,\hat{n}_x dl \tag{2.20}$$

Consequently, based on the values of the first derivatives, the calculation of the second derivatives is possible with the use of the same procure, as indicated in 2.21:

$$\frac{\partial^2 U}{\partial x^2} = \frac{\partial}{\partial x}\left(\frac{\partial U}{\partial x}\right) \quad , \quad \frac{\partial^2 U}{\partial y^2} = \frac{\partial}{\partial y}\left(\frac{\partial U}{\partial y}\right) \quad , \quad \frac{\partial^2 U}{\partial x \partial y} = \frac{\partial}{\partial x}\left(\frac{\partial U}{\partial y}\right) \tag{2.21}$$

In case a higher level of accuracy is desirable, the higher order correction terms are applied to the third-order formulation, providing a fourth-order scheme. The correction term is presented in 2.22.

$$\begin{aligned} \Delta U^{h-3} = \frac{1}{6}\Bigg[&\frac{\partial^3 U}{\partial x^3}(x_i - x_0)^3 + \frac{\partial^3 U}{\partial y^3}(y_i - y_0)^3 + 3\frac{\partial^3 U}{\partial x^2 \partial y}(x_i - x_0)^2(y_i - y_0) \\ &+ 3\frac{\partial^3 U}{\partial y^2 \partial x}(y_i - y_0)^2(x_i - x_0)\Bigg] \end{aligned} \tag{2.22}$$

Finally, the application of the same procedure is iterated to compute the terms of 2.22 as shown in 2.23:

$$\begin{aligned} &\frac{\partial^3 U}{\partial x^3} = \frac{1}{E}\oint_{\partial E} \frac{\partial^2 U}{\partial x^2}\,\hat{n}_x dl \quad , \quad \frac{\partial^3 U}{\partial y^3} = \frac{1}{E}\oint_{\partial E} \frac{\partial^2 U}{\partial y^2}\,\hat{n}_y dl \\ &\frac{\partial^3 U}{\partial x^2 \partial y} = \frac{1}{E}\oint_{\partial E} \frac{\partial^2 U}{\partial x \partial y}\,\hat{n}_x dl \quad , \quad \frac{\partial^3 U}{\partial y^2 \partial x} = \frac{1}{E}\oint_{\partial E} \frac{\partial^2 U}{\partial y \partial x}\,\hat{n}_y dl \end{aligned} \tag{2.23}$$

In the present work a numerical scheme up to 3rd order of accuracy was applied.

2.3 U-MUSCL Scheme

The derivation of the presented high order formulation was combined with the implementation of a variable-extrapolation named U-MUSCL-scheme. This formulation, developed as in [Bur05], is based on information currently available to the unstructured flow solvers, namely the variable and gradient information. U-MUSCLE closely resembles the traditional MUSCLE scheme and it is trivial to implement within most finite flow solvers. According to it, the interpolation function in Equation 1.33 is replaced by the following formulation

$$\begin{aligned} U_{PQ}^{L}(\kappa) &= U_P + \frac{\kappa}{2}(U_Q - U_P) + \frac{1}{2}\cdot(1-\kappa)\nabla U_P \cdot \vec{r}_{PQ} \\ U_{PQ}^{R}(\kappa) &= U_Q + \frac{\kappa}{2}(U_P - U_Q) - \frac{1}{2}\cdot(1-\kappa)\nabla U_Q \cdot \vec{r}_{PQ} \end{aligned} \tag{2.24}$$

where κ is the *U-MUSCL* parameter, U_P and U_Q are the left and right nodes' values of variable U and $\vec{r}_{PQ}$ is the vector connecting these nodes from point P to Q.

A one-parameter family of equations is represented in this new variable extrapolation formulation, which, in specific conditions, totally equals the MUSCL-scheme, a one parameter family as well [Bur05]. In the case of setting κ to *0*, the original unstructured formulation for 2nd-order variable extrapolation is obtained. In the case of setting κ το *-1*, the 2nd-order fully upwind MUSCL-type variable extrapolation is obtained. In the case of setting κ το *1/2*, a 3rd-order variable extrapolation to the cell face is made possible, whereas when κ is set to *1/3*, a 3rd-order approximation to the derivative at the node is obtained. If κ is set to *1*, a central difference scheme is achieved. Provided that $\kappa < 1$, this formula is an upwind one, becoming stable for hyperbolic systems of equations not containing shocks, and for high-quality grids. The following table summarizes what described above.

Table 2.1: *U-MUSCL with different values of κ parameter*

PARAMETER (κ)	**DESCRIPTION**
−1	*Second-order MUSCL-type scheme*
0	*Second-order unstructured upwind scheme*
1/3	*Third-order MUSCL-type scheme*
1/2	*Third-order extrapolation to face*
1	*Central-difference formula*

In a similar fashion Equation 2.24 can be written as in Equation 2.25, where a 3rd order scheme is achieved [Yan15]. In the present work the parameters κ and κ_3 are defined as *-1/6* and *-4/3* respectively.

$$
\begin{aligned}
U_{PQ}^{L}(\kappa) &= U_P + \frac{\kappa}{2}\left(U_Q - U_P\right) + \frac{1}{2}\cdot(1-\kappa)\nabla U_P \cdot \vec{r}_{PQ} \\
&\quad + \frac{1}{2}\left[\frac{\kappa_3}{4}\left(\nabla U_Q \cdot \vec{r}_{PQ} - \nabla U_P \cdot \vec{r}_{PQ}\right) + \frac{1}{4}(1-\kappa_3)\nabla\left(\nabla U_P \cdot \vec{r}_{PQ}\right)\cdot \vec{r}_{PQ}\right] \\
&= U_P + \tfrac{\kappa}{2}\left(U_Q - U_P\right) + \tfrac{1}{2}\cdot(1-\kappa)\nabla U_P \cdot \vec{r}_{PQ} \\
&\quad + \frac{1}{2}\left[\frac{\kappa_3 \Delta x_{PQ}}{4}\left(\left(\frac{\partial U}{\partial x}\right)_Q - \left(\frac{\partial U}{\partial x}\right)_P\right) + \frac{1}{4}\cdot(1-\kappa_3)\,\Delta x_{PQ}\,\nabla\left(\left(\frac{\partial U}{\partial x}\right)_P\right)\cdot \vec{r}_{PQ}\right] \\
&\quad + \frac{1}{2}\left[\frac{\kappa_3 \Delta y_{PQ}}{4}\left(\left(\frac{\partial U}{\partial y}\right)_Q - \left(\frac{\partial U}{\partial y}\right)_P\right) + \frac{1}{4}\cdot(1-\kappa_3)\,\Delta y_{PQ}\,\nabla\left(\left(\frac{\partial U}{\partial y}\right)_P\right)\cdot \vec{r}_{PQ}\right]
\end{aligned}
\tag{2.25}
$$

According to the above formulation, what can be shown is that that an existing code structure for a second-order scheme is capable to compute the higher derivatives simply by calling the same routine used for the calculation of the first derivatives.

2.4 High-Order Time Integration

In the current methodology of the high-order accuracy, time integration was performed by utilizing a high order Strong Stability Runge-Kutta method (SSPK) [Ruu05, Got05]. The development of Strong Stability Preserving (SSP) time discretization arose from the need to manage nonlinear stability properties in time and spatial discretization of hyperbolic PDEs. The core idea lies in the assumption that, with a suitably restricted time step Δt, the first order forward Euler time discretization of the method of lines ODE is strongly stable under a certain norm. In view of this, a higher order time discretization (Runge–Kutta or multi step) maintaining strong stability for the same norm emerges, possibly under a different time step restriction.

A general m stage Runge-Kutta method is written in the form [Ruu05]

$$\begin{aligned} U_P^{(0)} &= U_P^n \\ U_P^{(i)} &= \sum_{k=0}^{i-1} \left(\alpha_{\iota,k} U_P^{(k)} \right) + \Delta t_P \beta_{\iota,k} R\left(U_P^{(k)} \right), \qquad a_{i,k} \geq 0, \;\; i = 1, \dots, m \\ U_P^{n+1} &= U_P^{(m)} \end{aligned} \tag{2.26}$$

where Δt_P is the local time step at node P. In the current work, the five stage fourth order Runge-Kutta SSPRK (5, 4) developed by Ruuth [Ruu05] has been employed. An analytic expression, along with the appropriate coefficients of the optimal SSPRK (5, 4), is presented as follows [Got05]

$$\begin{aligned} U_P^{(1)} &= U_P^n + 0.391752226571890 \cdot \Delta t_P R(U_P^n) \\ U_P^{(2)} &= 0.444370493651235 \cdot U_P^n + 0.555629506348765 \cdot U_P^{(1)} \\ &\quad + 0.368410593050371 \cdot \Delta t_P R\left(U_P^{(1)} \right) \\ U_P^{(3)} &= 0.620101851488403 \cdot U_P^n + 0.379898148511597 \cdot U_P^{(2)} \\ &\quad + 0.251891774271694 \cdot \Delta t_P R\left(U_P^{(2)} \right) \\ U_P^{(4)} &= 0.178079954393132 \cdot U_P^n + 0.821920045606868 \cdot U_P^{(3)} \\ &\quad + 0.544974750228521 \cdot \Delta t_P R\left(U_P^{(3)} \right) \\ U_P^{n+1} &= 0.517231671970585 \cdot U_P^{(2)} \\ &\quad + 0.096059710526147 \cdot U_P^{(3)} + 0.063692468666290 \cdot \Delta t_P R\left(U_P^{(3)} \right) \\ &\quad + 0.386708617503269 \cdot U_P^{(4)} + 0.226007483236906 \cdot \Delta t_P R\left(U_P^{(4)} \right) \end{aligned} \tag{2.27}$$

which is SSP with CFL coefficient $CFL = 1.508$, and effective $CFL_{eff} = 0.377$. In the numerical simulations of the convergence study that follows, CFL was set to 1.5. Table 2.2 depicts the optimal coefficients from the above equation in compact form.

Table 2.2: *Coefficients of optimal SSPRK (5, 4) scheme [Ruu05]*

STAGES	***1***	***2***	***3***	***4***	***5***
$a_{i,k}$	*1*				
	0.444370493651235	0.555629506348765			
	0.620101851488403	0	0.379898148511597		
	0.178079954393132	0	0	0.821920045606868	
	0	0	0.517231671970585	0.096059710526147	0.386708617503269
$\beta_{i,k}$	0.391752226571890				
	0	0.368410593050371			
	0	0	0.251891774271694		
	0	0	0	0.544974750228521	
	0	0	0	0.063692468666290	0.226007483236906
CFL			1.50818004918983		

CHAPTER 3

NUMERICAL TEST AND RESULTS

3.1 Test Case

An extensive evaluation of the proposed high-order scheme was performed through a benchmark problem where a well-known analytical solution exists. As a verification test for the high-order scheme the transport of an isotropic vortex problem is examined. The ability to conserve the vortex shape and strength is important to many practical scenarios, in which a shed vortex interacts well downstream of the vortex origin. The particular problem is characterized by its smoothness with the absence of contact discontinuities [Yan15, Yan16].

On a computational domain $\Omega = [-10,10]\,X\,[-10,10]$ a vortex with center $(x_c, y_c) = (0,0)$ is simulated and moving from left to wright in a diagonal direction. The initial solution is given by the following equations [Yan15]

$$u = u_\infty - \beta u_\infty \frac{y - y_c}{R} exp\left(\frac{1 - r^2}{2}\right)$$

$$v = v_\infty + \beta u_\infty \frac{x - x_c}{R} exp\left(\frac{1 - r^2}{2}\right)$$

$$\rho = \rho_\infty \left[1 - \frac{\gamma - 1}{2\gamma}(\beta u_\infty)^2 exp(1 - r^2)\right]^{\frac{1}{\gamma - 1}} \tag{3.1}$$

$$p = p_\infty \left[1 - \frac{\gamma - 1}{2\gamma}(\beta u_\infty)^2 exp(1 - r^2)\right]^{\frac{1}{\gamma - 1}}$$

$$r = \sqrt{(x - x_c)^2 + (y - y_c)^2}\,/R, \quad \beta = 1/2\pi$$

where r denotes the distance from the vortex core, R refers to the vortex radius and the subscript '∞' express the uniform mean flow. In this study, the non-dimensionalized variables are set as u_∞=1, v_∞=1, ρ_∞=1, p_∞=1 and periodic boundary conditions are imposed in the x- and y-direction. As the analytical solution is obtainable at any given time, the numerical error of the simulations is feasible to be determined. The computational model is shown in Figure 3.1 along with the initial pressure field.

3.2 Computational Meshes

As introduced in the first chapter, the Finite Volume approach requires partitioning the computational domain $\Omega \subset R_3$ into a set of non-overlapping control volumes and the numerical implementation over each control volume. In the Node Centered Finite Volume scheme (NCFV) used in this work, solution values are defined at the mesh nodes while their locations are called data points. An initial decomposition of the computational domain into grid elements, the primal mesh, is used by the median dual partition to generate non-overlapping control volumes for the

node-discretization. These control volumes cover the entire computational domain and compose a mesh that is dual to the primal mesh.

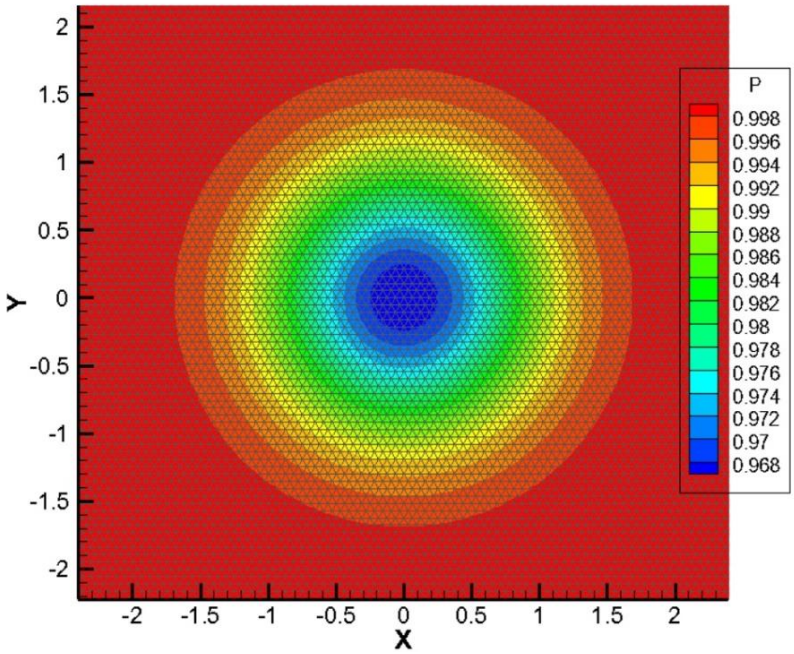


Figure 3.1: *Initial pressure field of the vortex*

The grids used in the present study can be categorized as either Regular or Irregular. Regular grids are derived by a smooth mapping from grids with periodic node connectivity, periodic cell distribution including, but not necessarily being limited to, grids derived from Cartesian ones [Del11, Dell13]. Four types of grids are considered in the present work:

1. Equilateral Triangular Grid (Type I)
2. Orthogonal Grid (Type II)
3. Orthogonal Grid (Type III)
4. Distorted Grid (Type IV)

The gird of Type I is composed of triangular elements with equal sides. Orthogonal gird of Type II refers to a regular triangular grid derived from a regular quadrilateral grid where squared cells are decomposed in four triangular cells by a diagonal splitting, while Orthogonal of Type II is derived, in a similar fashion, where two triangular cells are produced. As far as the Distorted Grid of Type IV is concerned, grid irregularities are introduced by perturbing the grid nodes of a Type-I Equilateral Triangular Grid from their original positions. The distortion of the nodes occurs with random shifts in each dimension and the perturbation is defined as $0.4r\Delta x$, where $r \in [-1/2, 1/2]$ is a random number and Δx is the local mesh size along the given dimension. These representative grid types are depicted in Figures 3.2-3.5.

As the main focus lies in the numerical accuracy and the performance of the proposed numerical scheme, a major prerequisite in order to perform convergence studies with a sequence of refined grids is the Consistency Refinement Property [Dis10, Dis11, Tho08]. This property requires the maximum distance across the grid cells to decrease consistently with increase of the total number of grid data points.

For a given computational domain with dimensions $L_x \, x \, L_y$ in the x- and y- dimension respectively, a subdivision of L_x by N_x line segments is defined, specifically $\Delta x = L_x/N_x$. Depending on the grid type, the subdivision $\Delta y = L_y/N_y$ can easily be determined. Accordingly, a characteristic length (effective mesh) is defined for each grid type as $h_N = \sqrt{(L_x \, x \, L_y)/N}$. A consistent grid refinement is performed when a reduction $\Delta x/2$, results $h'_N \simeq h_N/2$ and $N' \simeq 4N$. Having been defined as such, a series of increasingly fine grids from 20 x 20, 40 x 40, 80 x 80, 160 x 160 to 320 x 320 is employed for the previously stated types of grids. Table 3.1 depicts the results of the successive refinement procedure.

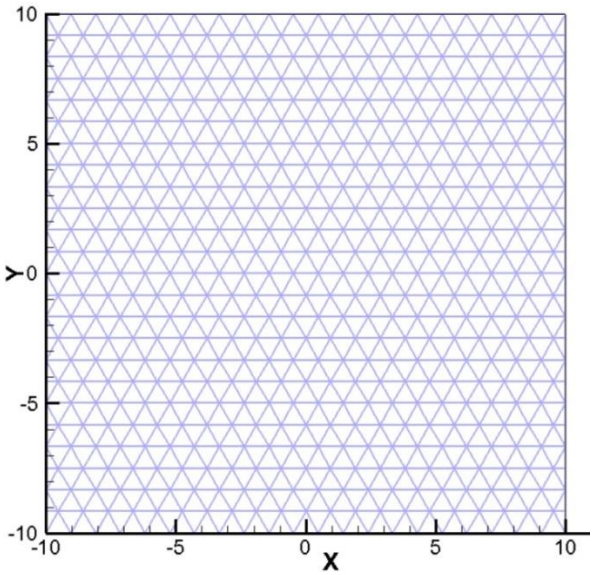


Figure 3.2: *Equilateral Grid (Type I)*

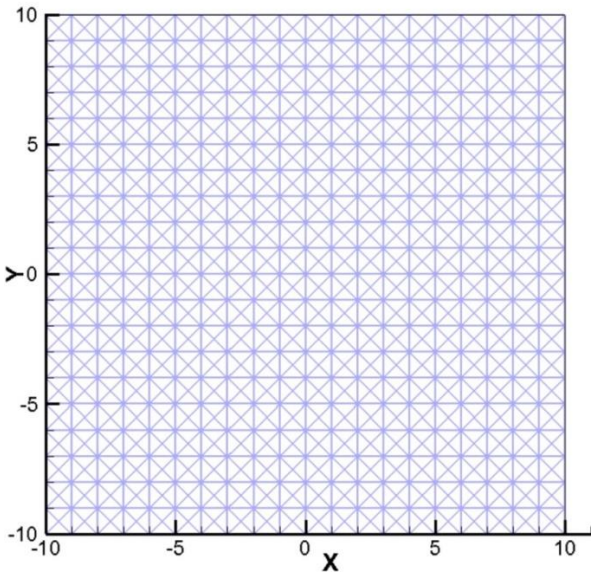


Figure 3.3: *Orthogonal Grid (Type II)*

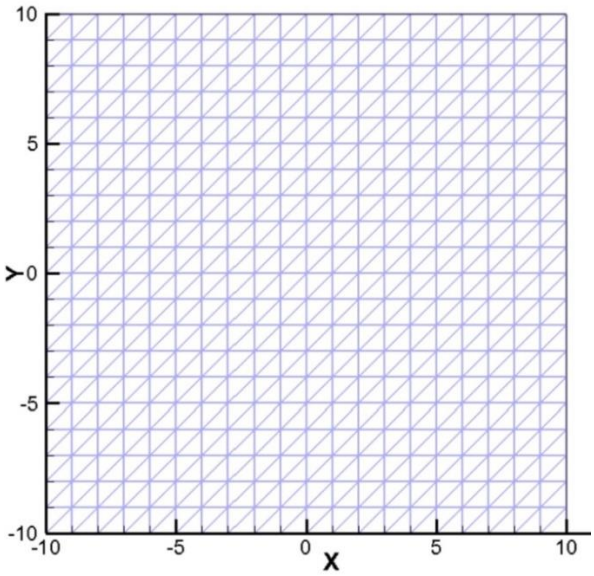


Figure 3.4: *Orthogonal Grid (Type III)*

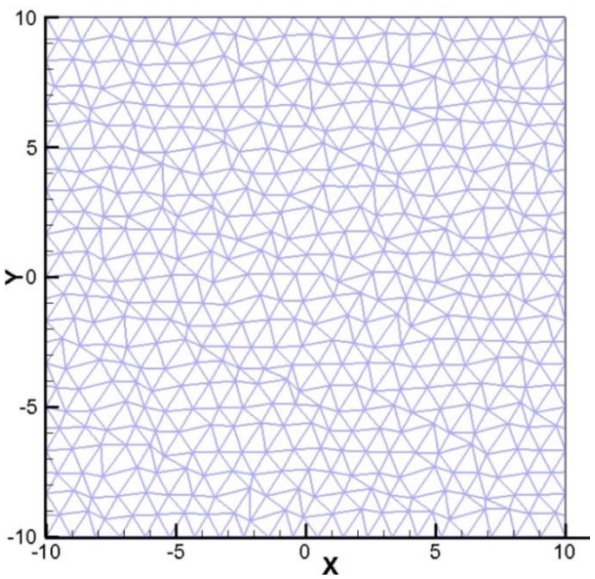


Figure 3.5: *Distorted Grid (Type IV)*

***Table 3.1**: Typical grid values for characteristic length and degrees of freedom*

GRID TYPES						
	TYPE I and IV		**TYPE II**		**TYPE III**	
N_x	N	h_N	N	h_N	N	h_N
20	562	*0.843649081*	*841*	*0.689655172*	*441*	*0.952380952*
40	*1950*	*0.452910813*	*3281*	*0.349161926*	*1681*	*0.487804878*
80	*7579*	*0.229733348*	*12961*	*0.175675314*	*6561*	*0.24691358*
160	*29877*	*0.115707497*	*51521*	*0.088112566*	*25921*	*0.124223602*
321	*119647*	*0.057820133*	*205441*	*0.044125174*	*103041*	*0.062305295*

3.3 Numerical Results

In this section, the numerical results of the conducted simulations are presented. In order to measure the solution error, the volume weighted norm L_K of the error was used, defined as [Del11, Del13]

$$\|U_i - U_i^{ex}\|_{L_{K(\Omega)}} = \left(\frac{\sum_{\iota=1}^{N}|\Omega_\iota|(U_i - U_i^{ex})^K}{\sum_{\iota=1}^{N}|\Omega_\iota|}\right)^{\frac{1}{K}} \tag{3.2}$$

where U_i^{ex} is the exact solution and U_i the numerical one, defined at node i of the conserved variables $(\rho, \rho u, \rho v, \rho E)$, while Ω_i is the corresponding volume and N is the number of the corresponding data points. The errors were measured in three different norms ($K = 1, K = 2, K = \infty$) between the numerical variables and their analytical counterparts at $t = 7\ s$ and $t = 60\ s$.

Figures 3.6-3.23 present the iterative convergence histories in all norms for the conservative variables $\rho, \rho u, \rho e$ on each grid used, and in two different time periods. More specifically, Figures 3.6-3.11 depict the corresponding convergence results in the *L2* norm, Figures 3.12-3.17 the ones in the *L1* norm and Figures 3.18-3.23 in the L_{max} norm. Additionally, contour plots for the conservative variable 'ρ' are given for comparison along with the analytical numerical solution. The contour plots exhibit two different grid refinements, 80 and 320, and on two different time periods $t = 7\ s$ and $t = 60\ s$, i.e. in Figures 3.24-3.27 the contour plots correspond to $t = 7\ s$ and Figures 3.28-3.31 correspond to $t = 60\ s$.

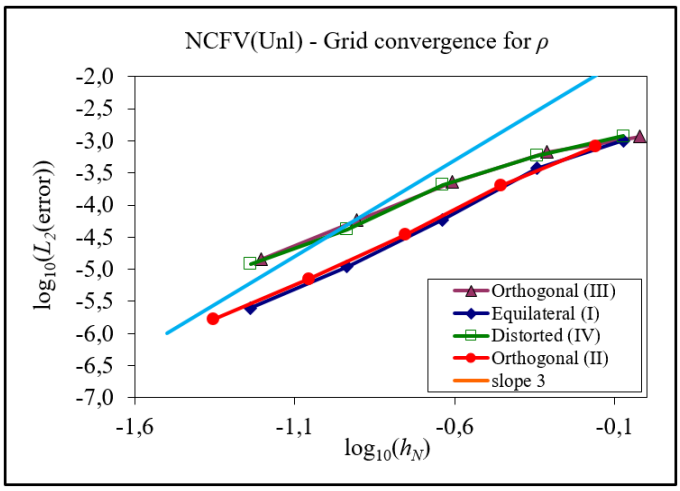


Figure 3.6: *Convergence results of L2 Norm for the conservative variable ρ at t=7 s*

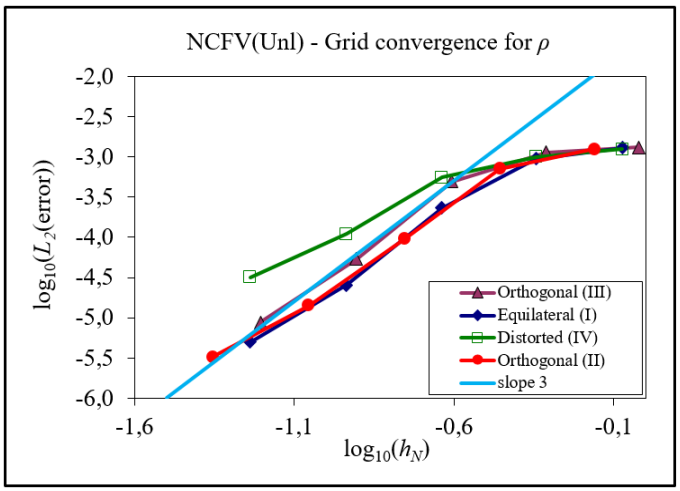


Figure 3.7: *Convergence results of L2 Norm for the conservative variable ρ at t=60 s*

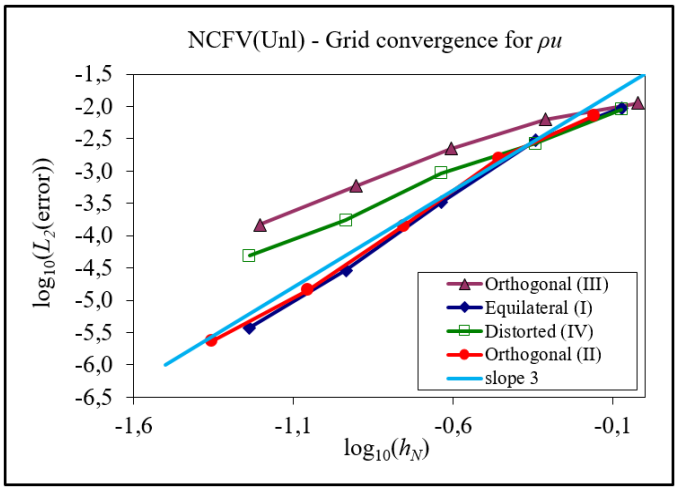


Figure 3.8: *Convergence results of L2 Norm for the conservative variable ρu at t=7 s*

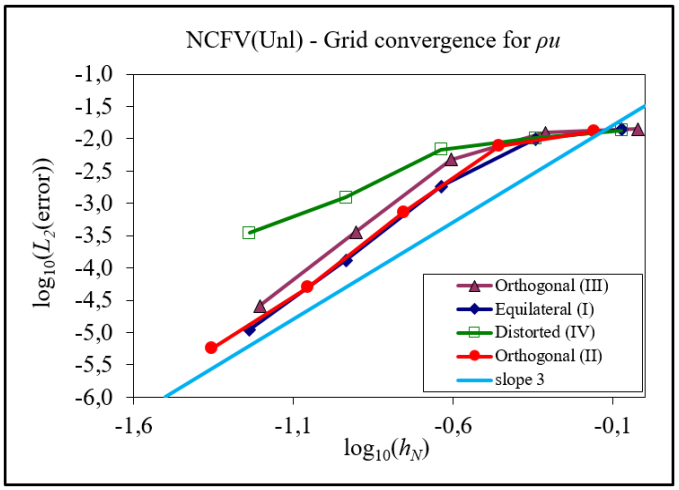


Figure 3.9: *Convergence results of L2 Norm for the conservative variable ρu at t=60 s*

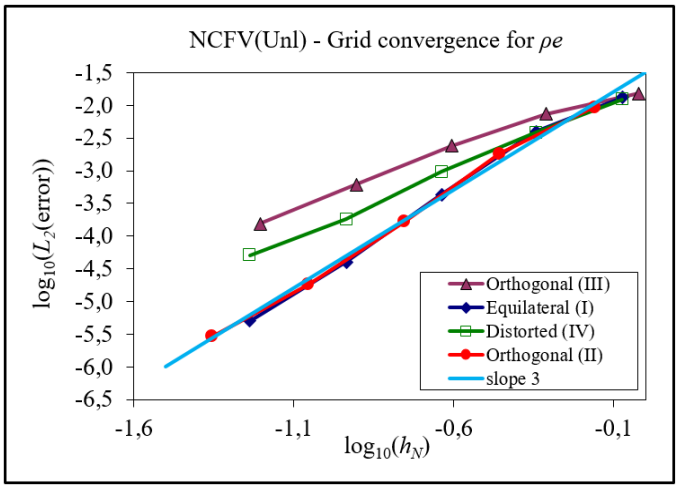


Figure 3.10: *Convergence results of L2 Norm for the conservative variable ρe at t=7 s*

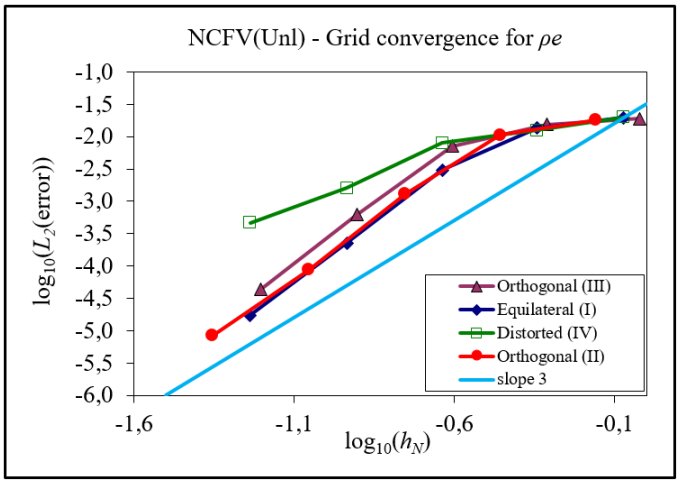


Figure 3.11: *Convergence results of L2 Norm for the conservative variable ρe at t=60 s*

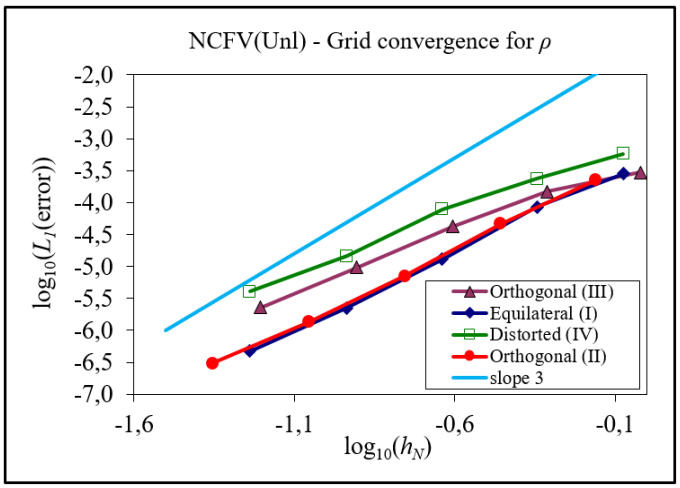


Figure 3.12: *Convergence results of L1 Norm for the conservative variable ρ at t=7 s*

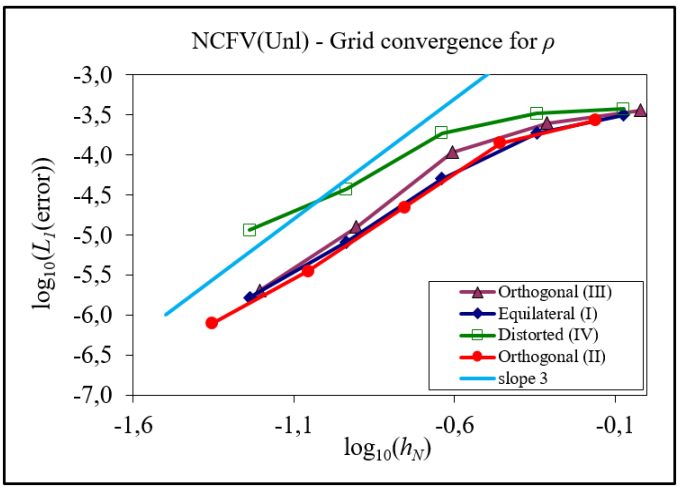


Figure 3.13: *Convergence results of L1 Norm for the conservative variable ρ at t=60 s*

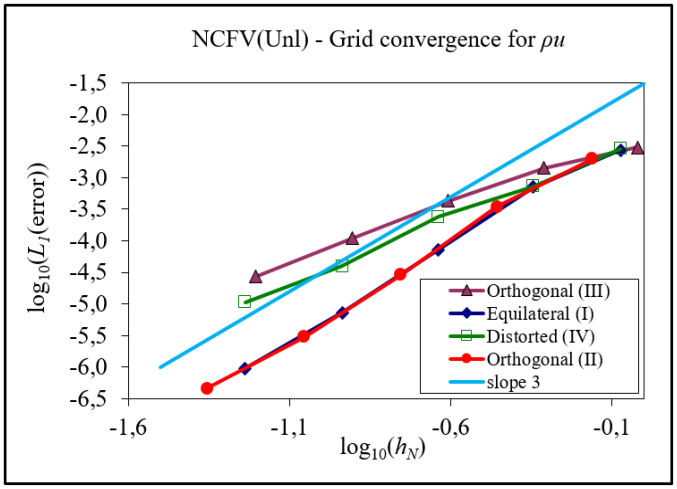


Figure 3.14: *Convergence results of L1 Norm for the conservative variable ρu at t=7 s*

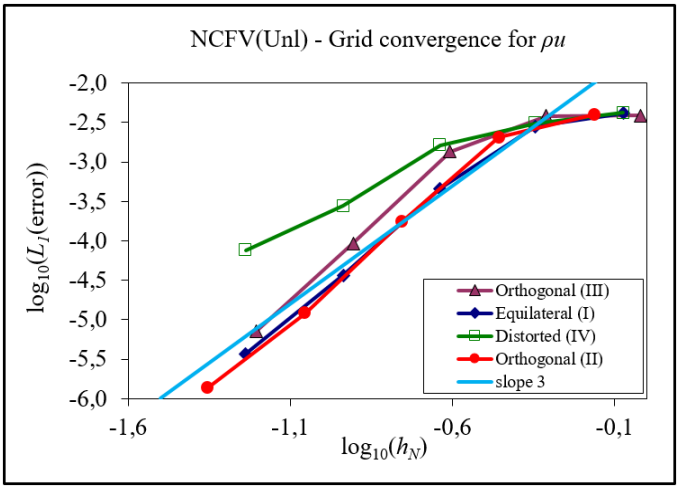


Figure 3.15: *Convergence results of L1 Norm for the conservative variable ρu at t=60 s*

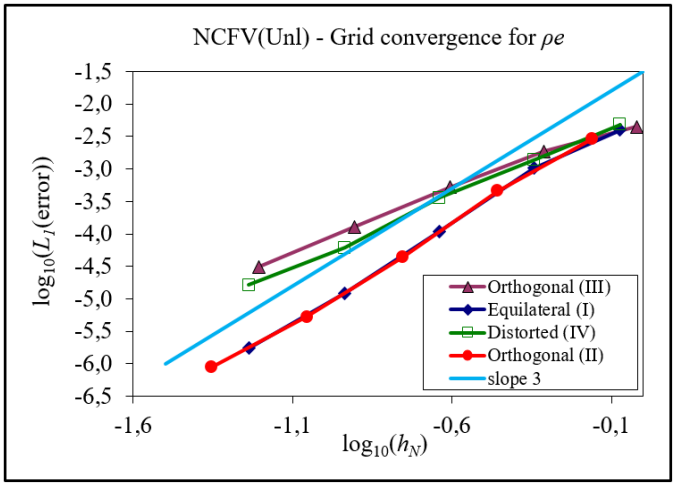


Figure 3.16: *Convergence results of L1 Norm for the conservative variable ρe at t=7 s*

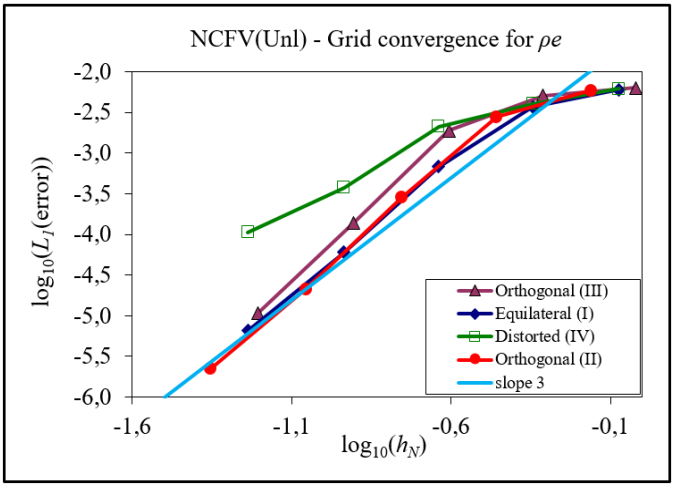


Figure 3.17: *Convergence results of L1 Norm for the conservative variable ρe at t=60 s*

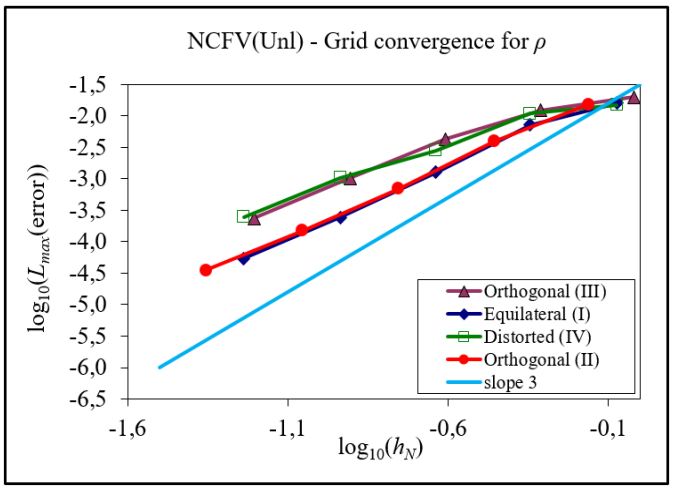


Figure 3.18: *Convergence results of Lmax Norm for the conservative variable ρ at t=7 s*

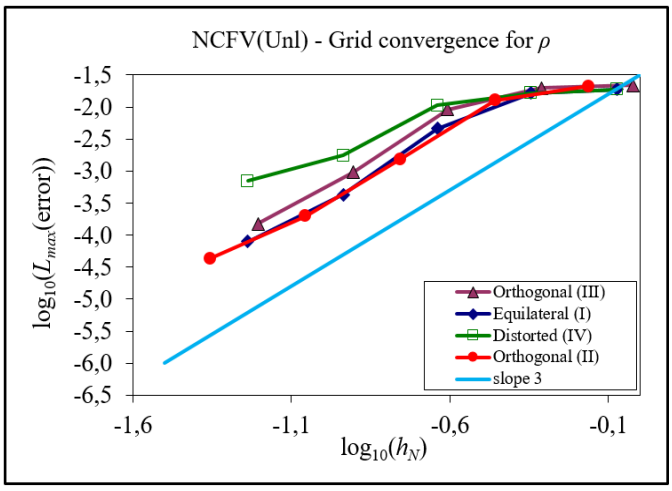


Figure 3.19: *Convergence results of Lmax Norm for the conservative variable ρ at t=60 s*

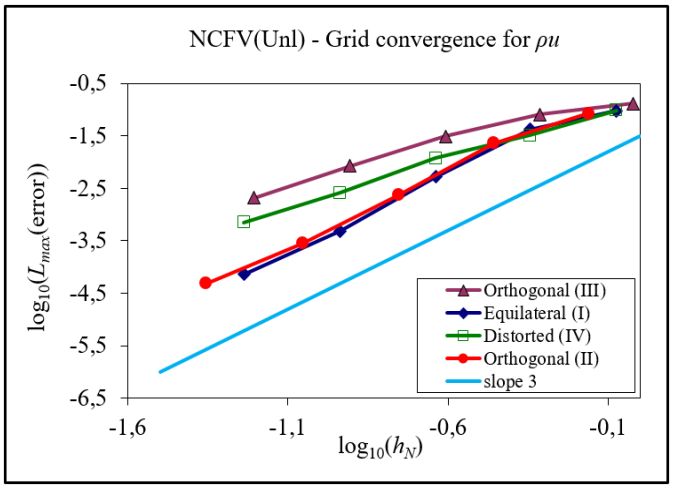


Figure 3.20: *Convergence results of Lmax Norm for the conservative variable ρu at t=7 s*

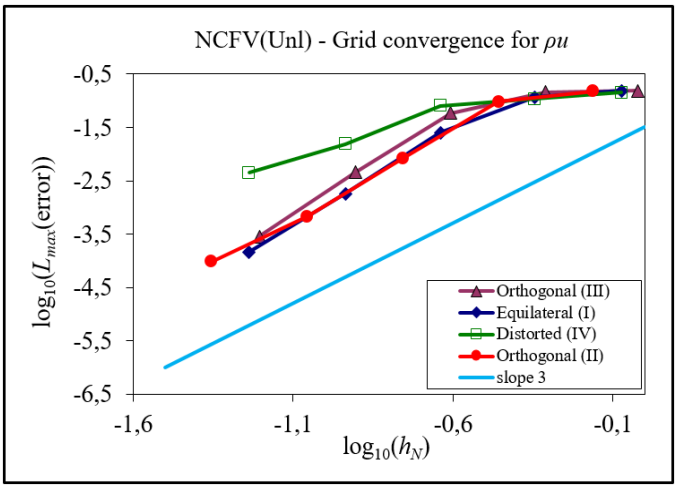


Figure 3.21: *Convergence results of Lmax Norm for the conservative variable ρu at t=60 s*

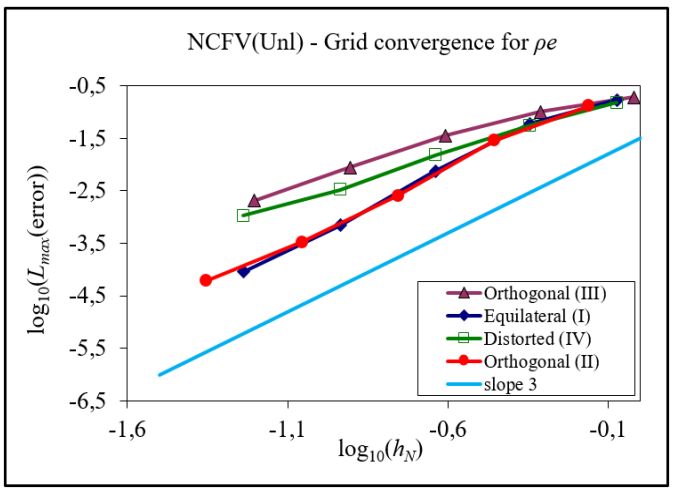


Figure 3.22: *Convergence results of Lmax Norm for the conservative variable ρe at t=7 s*

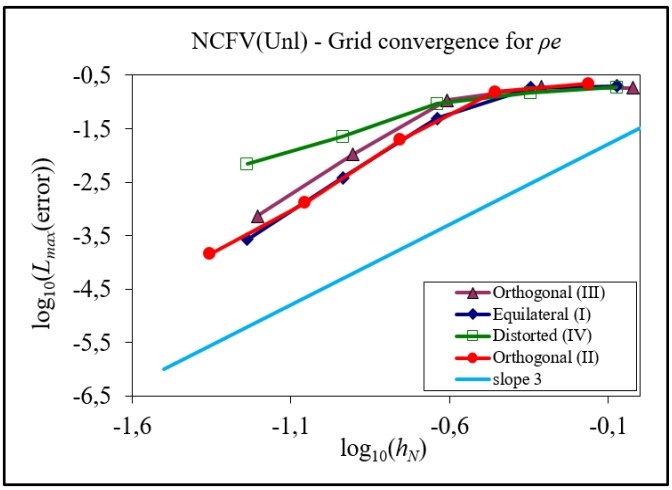


Figure 3.23: *Convergence results of Lmax Norm for the conservative variable ρe at t=60 s*

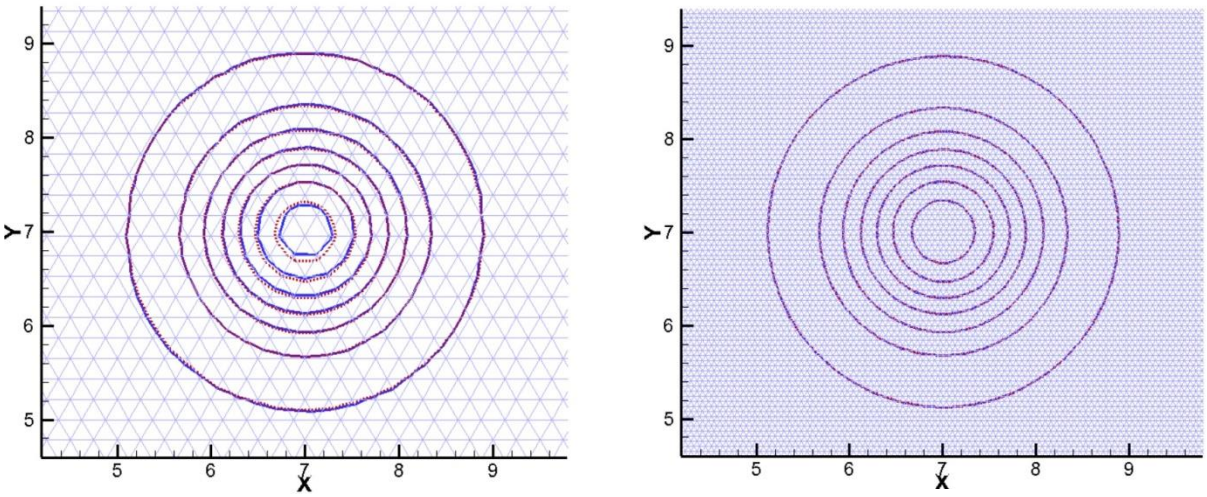


Figure 3.24: *Contour plots for ρ between the analytical (dashed line) and numerical solution for t=7 s. Grid refinement 80 (left) and 320 (right) for Equilateral Type I*

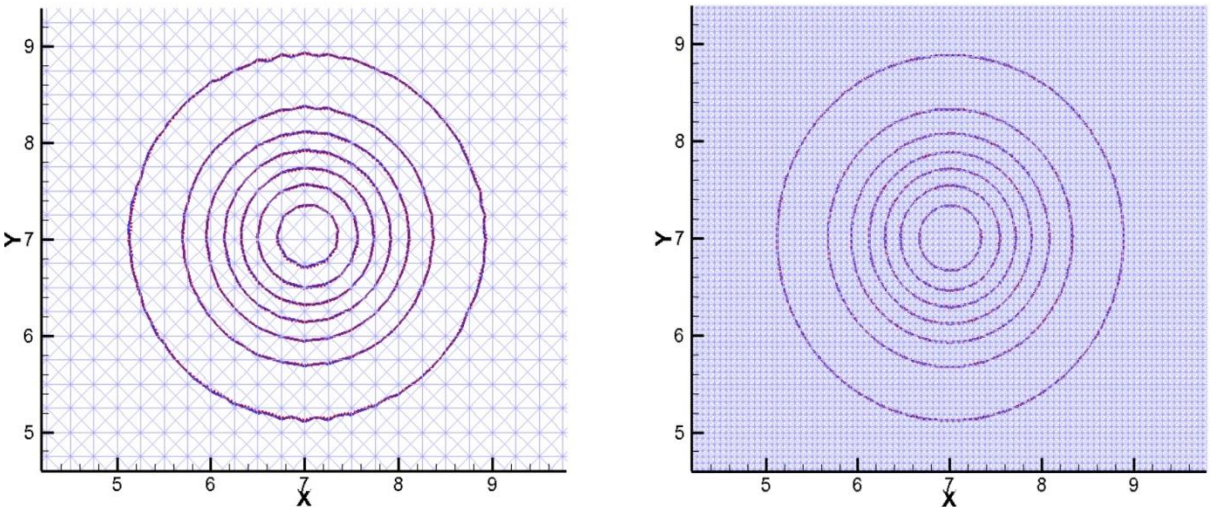


Figure 3.25: *Contour plots for ρ between the analytical (dashed line) and numerical solution for t=7 s. Grid refinement 80 (left) and 320 (right) for Orthogonal Type II*

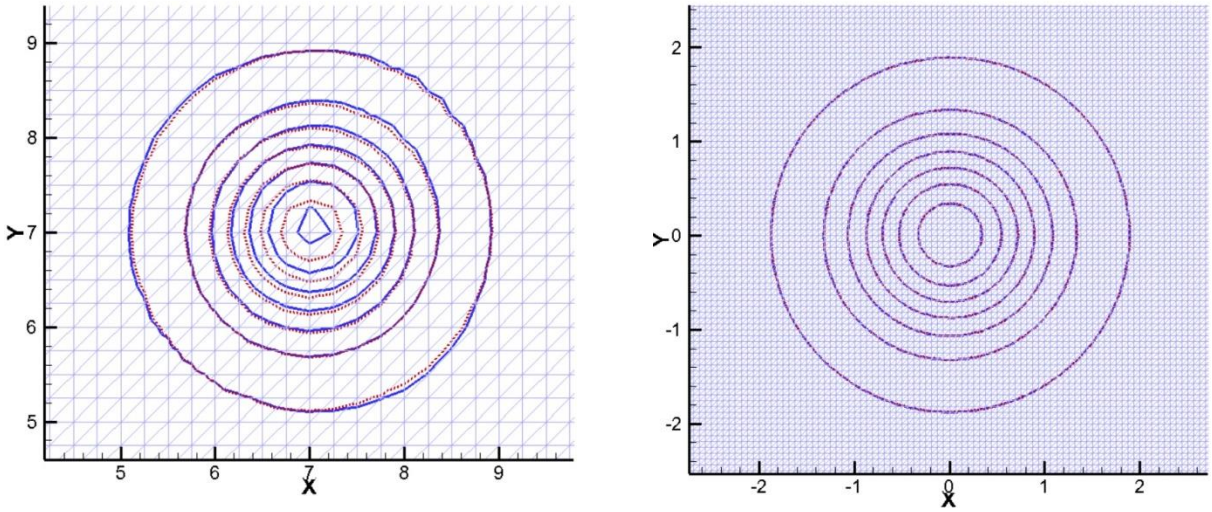


Figure 3.26: *Contour plots for ρ between the analytical (dashed line) and numerical solution for t=7 s. Grid refinement 80 (left) and 320 (right) for Orthogonal Type III*

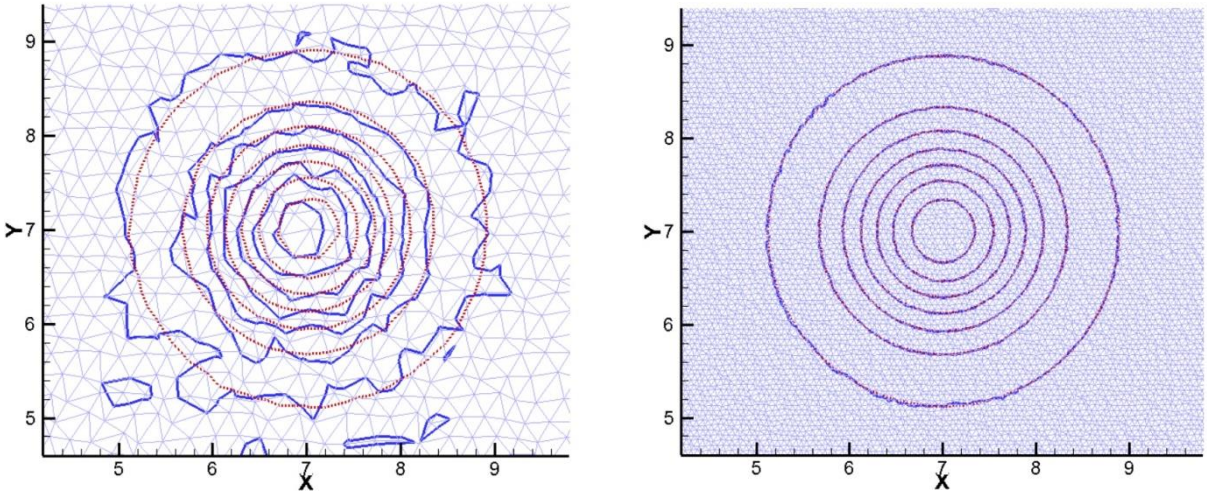


Figure 3.27: *Contour plots for ρ between the analytical (dashed line) and numerical solution for t=7 s. Grid refinement 80 (left) and 320 (right) for Distorted Type IV*

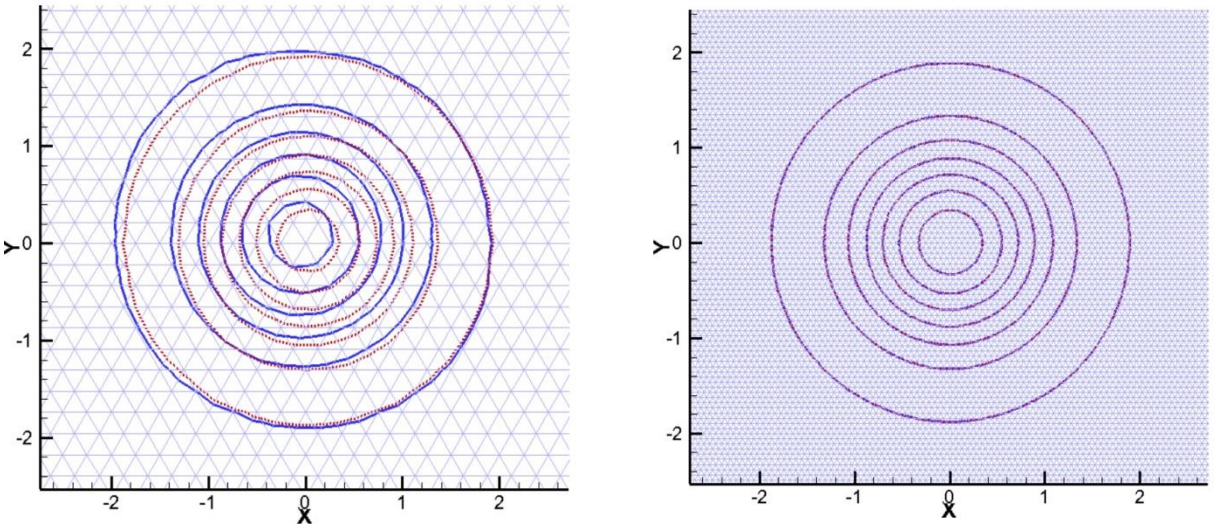


Figure 3.28: *Contour plots for ρ between the analytical (dashed line) and numerical solution for t=60 s. Grid refinement 80 (left) and 320 (right) for Equilateral Type I*

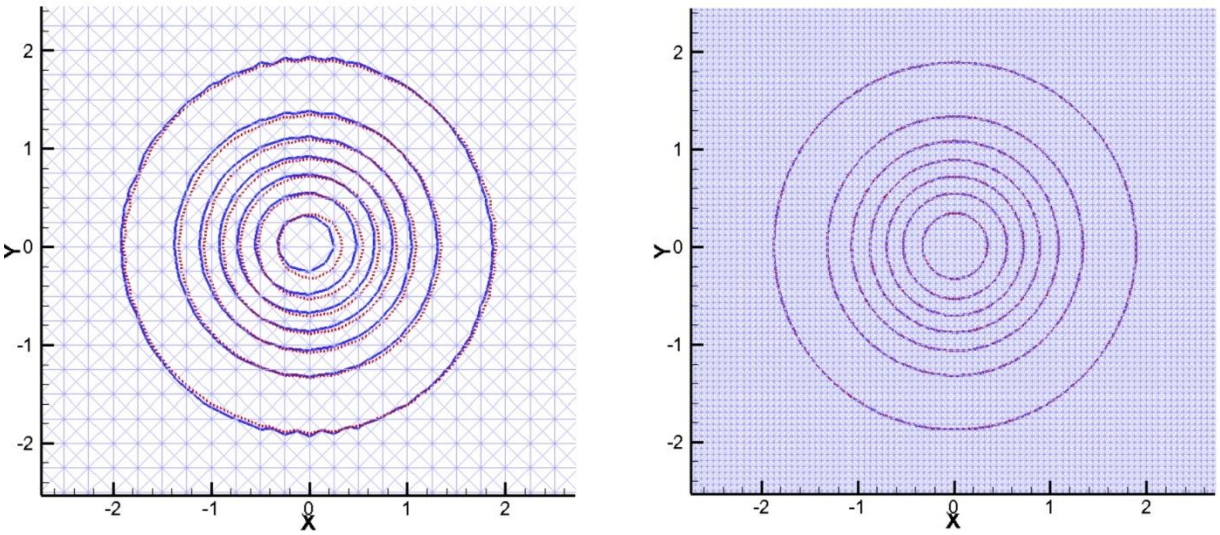


Figure 3.29: *Contour plots for ρ between the analytical (dashed line) and numerical solution for t=60 s. Grid refinement 80 (left) and 320 (right) for Orthogonal Type II*

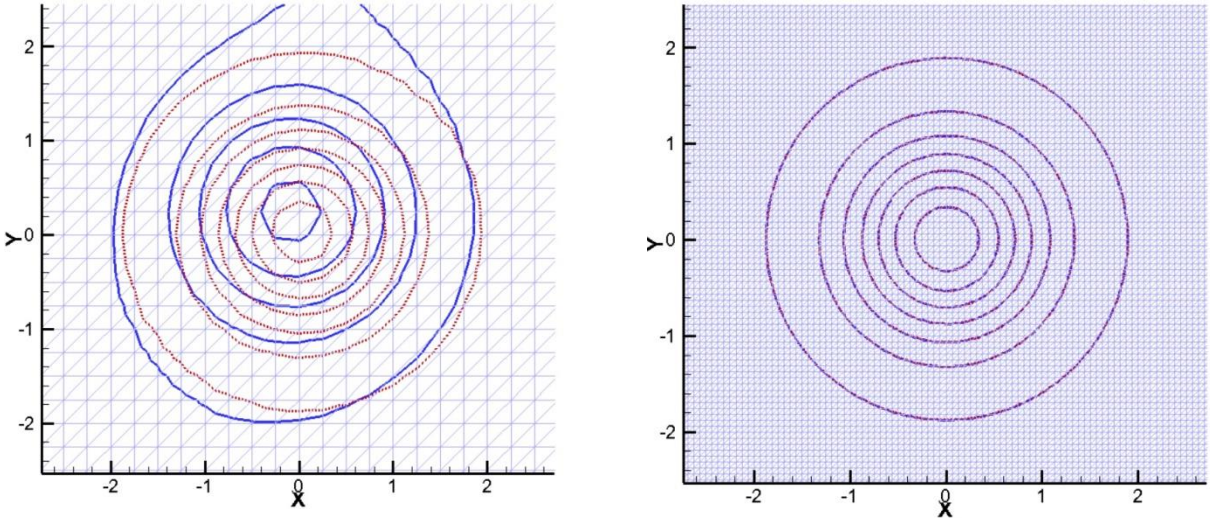


Figure 3.30: *Contour plots for ρ between the analytical (dashed line) and numerical solution for t=60 s. Grid refinement 80 (left) and 320 (right) for Orthogonal Type III*

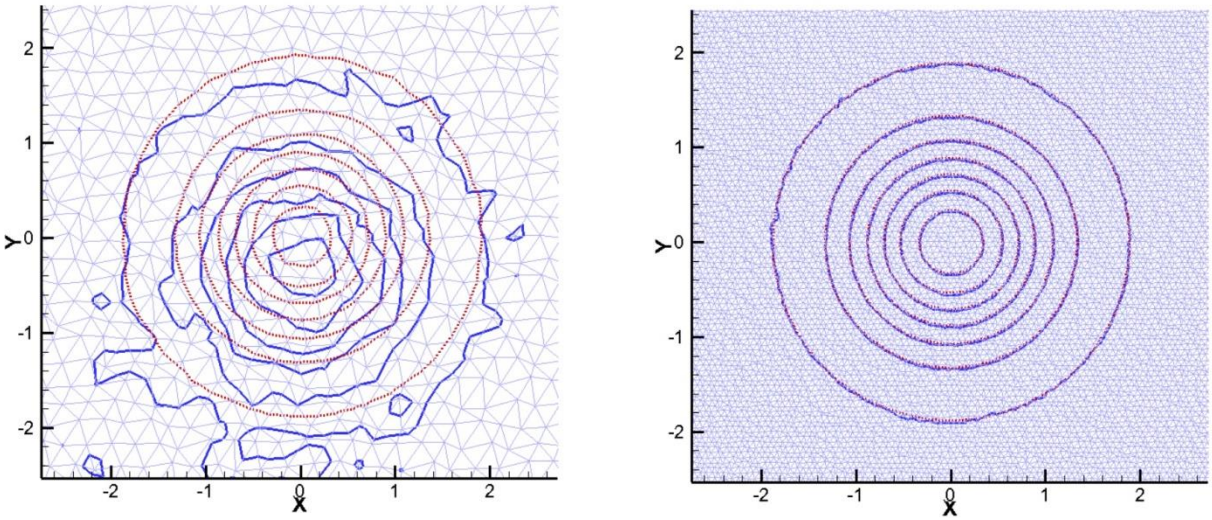


Figure 3.31: *Contour plots for ρ between the analytical (dashed line) and numerical solution for t=60 s. Grid refinement 80 (left) and 320 (right) for Distorted Type IV*

CHAPTER 4

INTRODUCTION OF HIGH-ORDER TO 3-D PROBLEMS

4.1 Introduction

The numerical tests have shown satisfactory results in the implementation of the current higher-order scheme, improving significantly the accuracy of the numerical solution. The proposed methodology could be extended to 3-dimensional problems and applied to a 3-D flow solver with slight alternations, as presented in the previous sections. The key aspects of the methodology concern, in summary, the calculations of the high-order correction terms up to the desirable order of accuracy, the incorporation of the U-MUSCL scheme and the employment of high-order time discretization with a multiple stage Runge-Kutta (SSPRK).

An academic in-house 3-D solver named *Galatea* [Lyg14a, Lyg14b, Lyg15] will be utilized for the application of the current high-order scheme. It employs the dimensionless Navier-Stokes equations, discretized with a Node-Centred Finite-Volume method on three-dimensional tetrahedral or hybrid unstructured grids, to simulate inviscid, viscous laminar and viscous turbulent compressible flows.

On the first chapter an extensive presentation of the fundamental properties of the fluid was undertaken and the governing equations for the 2-dimensional fluid flow were introduced in detail. Having set the above as a basis, the work then proceeds with the mathematical modeling of the governing equations in 3-dimensional space, and especially the Euler equations that are considered in the present thesis, followed by presenting the discretization of governing equations according to the numerical scheme implemented in *Galatea* solver and, finally, by discussing the high-order formulation in 3 dimensions.

4.2 Mathematical Modeling in 3-D

4.2.1 Navier-Stokes Equations

The motion of the fluid in three dimensions for a compressible viscous flow is described by the Navier-Stokes equations. The differential form arranged into convective (inviscid), diffusive (viscous), and source terms is expressed by the following equation:

$$\frac{\partial \vec{W}}{\partial t} + \frac{\partial \vec{F}^{inv}}{\partial x} + \frac{\partial \vec{G}^{inv}}{\partial y} + \frac{\partial \vec{J}^{inv}}{\partial z} - \frac{\partial \vec{F}^{vis}}{\partial x} - \frac{\partial \vec{G}^{vis}}{\partial y} - \frac{\partial \vec{J}^{vis}}{\partial z} = \vec{S} \tag{4.1}$$

According to the above equation, $\vec{W} = (\rho, \rho u, \rho v, \rho w, \rho E)$ refers to the convective variables' vector. $\vec{F}^{inv}, \vec{G}^{inv}$ and $\vec{J}^{inv}$ represent the inviscid flux vectors while $\vec{F}^{vis}, \vec{G}^{vis}, \vec{J}^{vis}$ refer to the viscous ones. The aforementioned are expressed in terms of the five primitive variables (ρ, u, v, w, p). The inviscid and viscous vectors are defined as shown in Equations 4.2 and 4.3, where the source term is considered to be zero [Koo00, Lyg14b].

$$\vec{F}^{inv} = \begin{pmatrix} \rho u \\ \rho u^2 + p \\ \rho uv \\ \rho uw \\ (\rho E + p)u \end{pmatrix}, \vec{G}^{inv} = \begin{pmatrix} \rho v \\ \rho vu \\ \rho v^2 + p \\ \rho vw \\ (\rho E + p)v \end{pmatrix}, \vec{J}^{inv} = \begin{pmatrix} \rho w \\ \rho wu \\ \rho wv \\ \rho w^2 + p \\ (\rho E + p)w \end{pmatrix} \tag{4.2}$$

$$\vec{F}^{vis} = \begin{pmatrix} 0 \\ \tau_{xx} \\ \tau_{xy} \\ \tau_{xz} \\ u\tau_{xx} + v\tau_{xy} + w\tau_{xz} + q_x \end{pmatrix} \vec{G}^{vis} = \begin{pmatrix} 0 \\ \tau_{yx} \\ \tau_{yy} \\ \tau_{yz} \\ u\tau_{yx} + v\tau_{yy} + w\tau_{yz} + q_y \end{pmatrix} \tag{4.3}$$

$$\vec{J}^{vis} = \begin{pmatrix} 0 \\ \tau_{zx} \\ \tau_{zy} \\ \tau_{zz} \\ u\tau_{zx} + v\tau_{zy} + w\tau_{zz} + q_z \end{pmatrix}$$

The diffusive flux vectors $\vec{F}^{inv}, \vec{G}^{inv}, \vec{J}^{inv}$ are defined from the stress tensor $(\tau_{xx}, \tau_{xy}, \tau_{xz}, \tau_{yy}, \tau_{yz}, \tau_{zz})$ and calculated according to Equation 4.4 [Hir90].

$$\tau_{ij} = \mu \left[\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} \right) - \frac{2}{3} (\nabla \cdot \vec{V}) \delta_{ij} \right] \tag{4.4}$$

where μ is the dynamic viscosity coefficient. In Equation 4.5 below, two alternative expressions are presented for the calculation of the dynamic viscosity. In the first one, it can be calculated based on the local temperature of the fluid (in K) i.e. the Sutherland law, whereas in the second, the reference dynamic viscosity μ_{ref} and the reference temperature T_{ref} -values usually used in the far field- are utilized [Luo05]. The coefficients c_1 and c_2 depend on the type of the fluid.

$$\mu = \frac{c_1 T^{3/2}}{T + c_2} \quad , \quad \mu = \mu_{ref} \left(\frac{T}{T_{ref}} \right)^{3/2} \frac{T_{ref} + c_2}{T + c_2} \tag{4.5}$$

The complete set of the equation system is obtained via the expression for the thermodynamic relations between the state variables. This stands with the assumption that in pure aerodynamics the fluid can quite behave like a perfect gas. Hence, the state equation refers to the following mathematical expression:

$$p = \rho R_g T \tag{4.6}$$

The term R_g refers to the gas constant equal to *287.04* $m^2 s^{-2} K^{-1}$ and it is related with the constant pressure and volume specific heat coefficients with the following equations

$$R_g = c_p - c_v \quad , \quad \gamma = c_p / c_v \tag{4.7}$$

where these coefficients are determined as follows

$$h = c_p T \quad , \quad e = c_v T \tag{4.8}$$

and h and e are the specific enthalpy and specific internal energy of the gas (per unit mass) and are regarded as constants [Lan98]. With respect to the above and after certain manipulations, the total energy per unit volume ρE and the corresponding specific total enthalpy h_t are computed via Equation 4.9.

$$\rho E = \frac{p}{(\gamma - 1)} + \frac{1}{2}\rho(u^2 + v^2 + w^2) \quad , \quad h_t = \frac{\gamma p}{\rho(\gamma - 1)} + \frac{1}{2}(u^2 + v^2 + w^2) \tag{4.9}$$

As illustrated below, the heat flux vector (q_x, q_y, q_z) in the energy equation is determined based on the stress tensor. Evidently, the conductivity coefficient χ relies on the dimensionless Prandtl number Pr [Bla01]:

$$q_i = \chi \nabla T \quad , \quad \chi = \frac{\mu c_p}{Pr} \tag{4.10}$$

4.2.2 Euler Equations

A simplified form of the Navier-Stokes equations where viscosity and thermal conductivity are assumed equal to zero, i.e. inviscid flows, is given by the Euler equations. In this case, the corresponding flux vectors $\vec{F}^{vis}, \vec{G}^{vis}$ and $\vec{J}^{vis}$ are neglected, leading to 4.11-4.12, while the remaining terms 4.6-4.9 complete the system of the equations [Bla01]:

$$\frac{\partial \vec{W}}{\partial t} + \frac{\partial \vec{F}^{inv}}{\partial x} + \frac{\partial \vec{G}^{inv}}{\partial y} + \frac{\partial \vec{J}^{inv}}{\partial z} = \vec{S} \tag{4.11}$$

$$\vec{F}^{inv} = \begin{pmatrix} \rho u \\ \rho u^2 + p \\ \rho uv \\ \rho uw \\ (\rho E + p)u \end{pmatrix}, \vec{G}^{inv} = \begin{pmatrix} \rho v \\ \rho vu \\ \rho v^2 + p \\ \rho vw \\ (\rho E + p)v \end{pmatrix}, \vec{J}^{inv} = \begin{pmatrix} \rho w \\ \rho wu \\ \rho wv \\ \rho w^2 + p \\ (\rho E + p)w \end{pmatrix} \tag{4.12}$$

Following the common practice of writing the conservation equations in a non-dimensional form using dimensionless quantities, the normalization of the variables is performed by utilizing a characteristic length L_{ref}, the free-stream velocity V_{ref}, the free-stream density ρ_{ref}, the free-stream dynamic viscosity μ_{ref}, and the constant volume specific heat coefficient c_v, as presented in 4.13. In this sense, each variable is divided by a quantity that has the same dimension as the original variable.

$$\tilde{x_i} = \frac{x_i}{L_{ref}} \, , \tilde{u}_i = \frac{u_i}{V_{ref}} \, , \; \tilde{\rho} = \frac{\rho}{\rho_{ref}} \, , \; \tilde{\mu} = \frac{\mu}{\mu_{ref}} \, , \; \tilde{R}_g = \frac{R_g}{c_v} = \gamma - 1 \tag{4.13}$$

With respect to the rest of the variables included in Equations 4.11-4.12, taking into account the aforementioned normalizations, they are expressed in the following manner [Mun98]

$$\tilde{p} = \frac{p}{\rho_{ref} V_{ref}^2} \ , \ \widetilde{\rho E} = \frac{\rho E}{\rho_{ref} V_{ref}^2} \ , \ \tilde{h}_t = \frac{h_t}{V_{ref}^2} \ , \ \tilde{T} = \frac{T}{V_{ref}^2 / c_v} \ , \ \tilde{t} = \frac{t}{L_{ref} / V_{ref}} \tag{4.14}$$

while the perfect gas equation, according to the normalized constant pressure ($\tilde{c}_P = \gamma$) and volume specific heat ($\tilde{c}_v = 1$) coefficients, is rewritten as in 4.15:

$$\tilde{p} = \tilde{\rho}(\gamma - 1)\tilde{T} \tag{4.15}$$

Finally, the calculation of the local speed of sound at node P ($\tilde{a}_P$) and the corresponding Mach number (M_P) is obtained with the last expressions below [Mun98].

$$\tilde{a}_P = \sqrt{\frac{\gamma \, \tilde{p}_P}{\tilde{\rho}_P}} \quad , \quad M_P = \frac{\sqrt{\tilde{u}_P^2 + \tilde{v}_P^2 + \tilde{w}_P^2}}{\tilde{a}_P} \tag{4.16}$$

It is worth reminding at this point that for simplification reasons, the superscript "~", denoting the normalized variables, will be neglected in the following sections.

4.3 Numerical Modeling of 3-D Equations

4.3.1 Spatial Discretization

The node-centered scheme has already been discussed in previous sections. Briefly, the main concept concerns a computational domain divided into a finite number of cells, from which non-overlapping control volumes are defined. The control volumes are formed around each vertex, where the variables are stored, covering the entire computational domain and composing a mesh that is dual to the primal mesh.

In the context of a three-dimensional space, the dual control volume of a node P is constructed by connecting lines defined by edge midpoints, the barycenter of faces, and the barycenter of elements sharing this node [Mav94, Kal96, Mav96, Koo00, Bla01, Kim03, Kou03, Kal05, Lyg12]. Figure 4.1 depicts part of control volume around a node P for a three-dimensional unstructured grid [Ada05]. Evidently, this part comprises the individual parts of the control volume, which three of the tetrahedrons contribute to, with node P (orange color) pertaining to them. In addition, the middle points of the edges (black color), the barycenter of the tetrahedron faces (red color) and the tetrahedrons' barycenter (green color) are also illustrated. More specifically, nodes $J, K, L, M,$ and Q are adjacent to node P and points G_1, G_2, G_3 are the barycenters of tetrahedrons $PQMJ, PQJK$ and $PQLK$, respectively. Furthermore, M stands as the middle point of edge PQ and C_1, C_2 as the barycenters of faces PQJ and PQK, correspondingly.

More details are given in Figure 4.2, where the contribution of different types of elements to the control volume of a node P is depicted [Lyg15]. Again, G stands for the barycenter of the element, G_1 and G_2 denote the barycenters of the respective faces, while $M_1, M_2,$ and M_3 the midpoints of the edges.

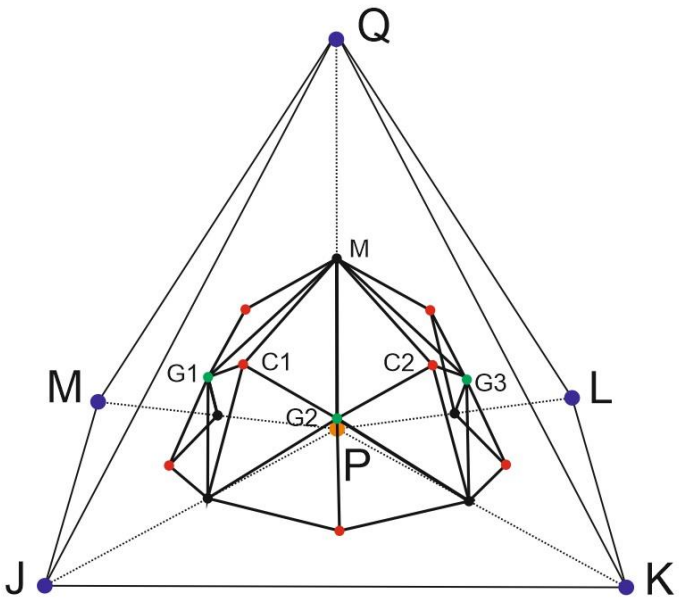


Figure 4.1: *Part of control volume around node P*

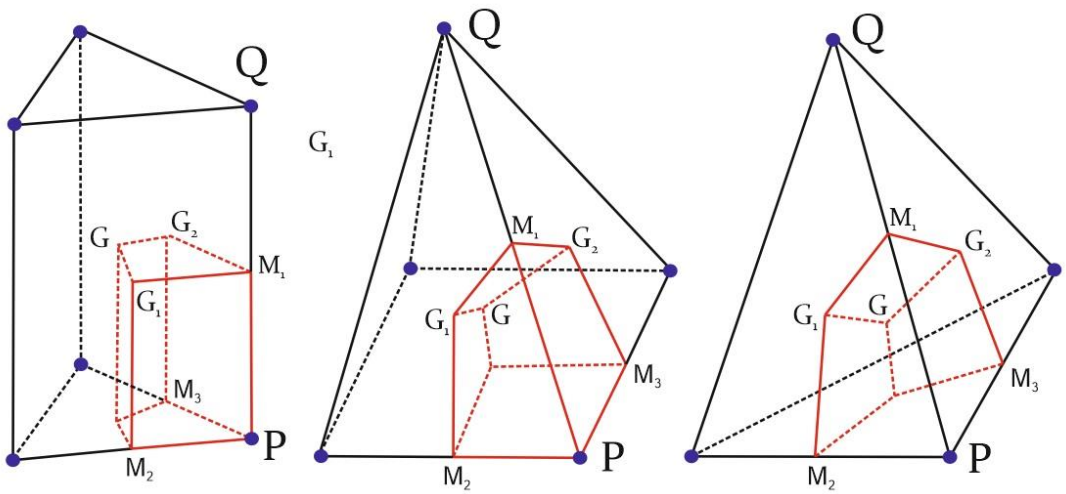


Figure 4.2: *Contribution of a prismatic, pyramidal and tetrahedral element to the control volume of a node P*

The integration of Euler equations over the control volume CV_P, along with the employment of the Green-Gauss divergence theorem, leads to

$$\iiint_{CV_P} \frac{\partial \vec{W}}{\partial t} dxdydz + \iint_{\partial CV_P} \vec{\tilde{H}}^{inv} ds = \iiint_{CV_P} \vec{S}\, dxdydz \tag{4.17}$$

where ∂CV_P demarcates the boundaries of the control volume of node P delineated by the facets constructed around the edges connecting node P with each neighboring node Q. Furthermore, $\vec{\tilde{H}}^{inv}$ is the vector of the inviscid numerical fluxes and is evaluated at the midpoint of an edge connected to node P. This midpoint coincides with the interface between the adjacent control volumes of nodes P and Q connected with this edge. The expression of $\vec{\tilde{H}}^{inv}$ is thus

$$\vec{H}^{inv} = \hat{n}_{PQ,x}\,\vec{F}^{inv} + \hat{n}_{PQ,y}\,\vec{G}^{inv} + \hat{n}_{PQ,z}\,\vec{J}^{inv} \tag{4.18}$$

$$\vec{\hat{n}}_{PQ} = \frac{\vec{n}_{PQ}}{|\vec{n}_{PQ}|} = \left(\hat{n}_{PQ,x}, \hat{n}_{PQ,y}, \hat{n}_{PQ,z}\right) \tag{4.19}$$

where $\vec{n}_{PQ}$ is determined as the vector sum of the outward normal vectors of all the facets forming ∂CV_{PQ}. What is presented in Figure 4.3 is part of such a vector $\vec{n}_{PQ}$ contributed by a tetrahedron. The computation of this vector is performed with the use of the tetrahedron G barycenter, of the faces G_1, G_2 medians and of edge M. Figure 4.5 represents the total face area and the mean unit normal vector associated with edge PQ.

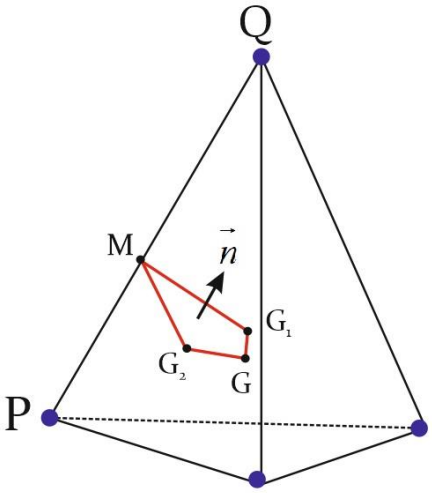


Figure 4.3: *Part of vector $\vec{n}_{PQ}$ contributed by a tetrahedron*

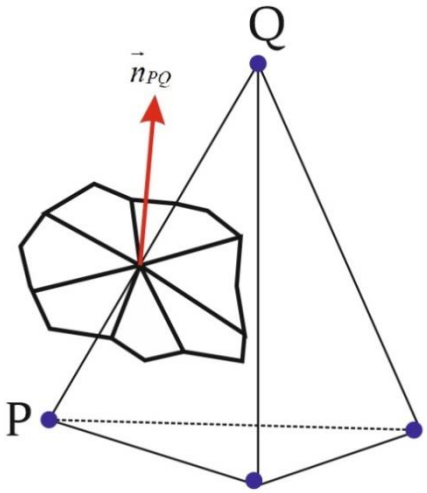


Figure 4.5: *Total face area and mean unit normal vector associated with the edge PQ*

If ∂CV_{PQ} is the interfacing part of ∂CV_P, $K_N(P)$ is the set of nodes adjacent to P, and Γ is the domain's external boundary, Equation 4.17 is transformed into 4.20 for ∂CV_P being expressed as in 4.21.

$$\iiint_{CV_P} \frac{\partial \vec{W}}{\partial t} dxdydz + \sum_{Q \in K_N(P)} \iint_{\partial CV_{PQ}} \vec{\hat{H}}^{inv}\, ds + \iint_{\partial CV_P \cap \Gamma} \vec{\hat{H}}^{inv}\, ds = \iiint_{CV_P} \vec{S}\, dxdydz \tag{4.20}$$

$$\partial CV_P = \bigcup_{Q \in K_N(P)} \partial CV_{PQ} + (\partial CV_P \cap \Gamma) \tag{4.21}$$

Presuming the conservative variables at node P are equal to their mean values over CV_P, the first term in Equation 4.17 becomes:

$$\iiint_{CV_P} \frac{\partial \vec{W}}{\partial t} dxdydz = \left(\frac{d\vec{W}}{dt}\right)_P \iiint_{CV_P} dxdydz = \left(\frac{d\vec{W}}{dt}\right)_P V_P \tag{4.22}$$

When the integrals of the numerical fluxes are expressed as summations of fluxes through the faces composing the control volume of node P, Equation 4.20 is expressed as

$$\left(\frac{d\vec{W}}{dt}\right)_P V_P + \sum_{Q\in K_N(P)} \vec{\Phi}_{PQ}^{inv} + \sum_{(K_{out}\in \partial CV_P\cap\Gamma)} \vec{\Phi}_{P,out}^{inv} = \iiint_{CV_P} \vec{S}\, dxdydz \tag{4.23}$$

$$\vec{\Phi}_{PQ}^{inv} = \iint_{\partial CV_{PQ}} \vec{\tilde{H}}^{inv} ds = \vec{f}\left(\vec{W}_{PQ}^{L}, \vec{W}_{PQ}^{R}, \vec{n}_{PQ}\right)$$
$$\vec{\Phi}_{P,out}^{inv} = \iint_{\partial CV_P\cap\Gamma} \vec{\tilde{H}}^{inv}\, ds = \vec{f}\left(\vec{W}_{P}, \vec{W}_{out}, \vec{n}_{out}\right) \tag{4.24}$$

where $\vec{W}_{PQ}^{L}$ and $\vec{W}_{PQ}^{R}$ are the vectors of the conservative variables on the left and right side of edge PQ respectively, while $\vec{W}_{out}$ is the corresponding vector on the boundary.

4.3.2 Numerical Fluxes

The computation of the numerical inviscid fluxes is achieved with the employment of a one-dimensional Riemann problem. This is performed in the direction of each normal vector that corresponds to every particular face forming the control volume of a node P. Moreover, an upwind scheme using Roe's approximate Riemann solver is implemented [Roe81], due to the expensive amount of calculations that the exact solution requires [Lan98]. Eventually, the inviscid fluxes are evaluated in the middle point of an edge PQ, as shown in 4.25.

$$\vec{\Phi}_{PQ}^{inv} = \frac{1}{2}\left(\vec{H}^{inv}\left(\vec{W}_{PQ}^{L}, \vec{n}_{PQ}\right) + \vec{H}^{inv}\left(\vec{W}_{PQ}^{R}, \vec{n}_{PQ}\right)\right) - \frac{1}{2}\left|\underline{\tilde{A}}_{PQ}\right|\left(\vec{W}_{PQ}^{R} - \vec{W}_{PQ}^{L}\right) \tag{4.25}$$

The Jacobian matrix $\underline{\tilde{A}}_{PQ}$ of the inviscid flux vector $\vec{H}^{inv}$ is calculated according to the Roe's averaged values of the primitive variables as in 4.26 at the midpoint of the corresponding edge PQ [Roe81, Ven95, Lan98]. Detailed information for the matrix $\underline{\tilde{A}}_{PQ}$ is provided in Appendix A.

$$\vec{\tilde{U}}_{PQ} = \frac{\sqrt{\rho_L}\,\vec{U}_L + \sqrt{\rho_R}\,\vec{U}_R}{\sqrt{\rho_L} + \sqrt{\rho_R}} \tag{4.26}$$

where $\vec{U}_L$ and $\vec{U}_R$ in first order accurate schemes are the values of primitive variables at the left and right side of edge PQ respectively.

An equivalent expression of Equation 4.25 is the following [Roe81, Lan98]

$$\vec{\Phi}_{PQ}^{inv} = \vec{H}^{inv}\left(\vec{W}_{PQ}^{L}, \vec{n}_{PQ}\right) + \underline{\tilde{A}}_{PQ}^{-}\left(\vec{W}_{PQ}^{R} - \vec{W}_{PQ}^{L}\right) \tag{4.27}$$

Whenever a second-order scheme is required, the traditional MUSCL reconstruction of the primitive or conservative variables is incorporated using appropriate limiters (Van Albada -Van Leer [VanA82], Min-mod [Swe84] or Barth-Jespersen [Bar89]) to control the total variation. Left and right states of an edge PQ are reconstructed using Taylor series expansion, taking into account the corresponding values of the neighboring nodes. Consequently, the primitive or conservative variables U of each state at the midpoint of an edge PQ are approximated as following [Bar92, And94, Bla01, ANSYS06, Lyg13, and Sar14]:

$$U_{PQ}^{L} = U_P + \frac{1}{2} \cdot (\nabla U)^L \cdot \vec{r}_{PQ}$$
$$U_{PQ}^{R} = U_Q - \frac{1}{2} \cdot (\nabla U)^R \cdot \vec{r}_{PQ} \quad (4.28)$$

The quantities marked by L and R subscripts denote the values of the variables taken at the left and right side of the boundary between the nodes P and Q, while $\vec{r}_{PQ}$ is the vector connecting these nodes and is directed from P to Q. The extrapolation gradients $(\nabla U)^L$ and $(\nabla U)^R$ are equal to the gradients $(\nabla U)_P$ and $(\nabla U)_Q$ at the nodes P and Q respectively and calculated with the employment of the Green-Gauss linear representation method as [Bar92, Bla01]:

$$(\nabla U)_P = \frac{1}{V_P} \sum_{Q \in K_N(P)} \frac{1}{2}\left(U_P + U_Q\right) \cdot \vec{n}_{PQ} \quad (4.29)$$

where V_P is the volume of the control volume of node P. In the case of a boundary node, the equivalent expression is the following [Lyg13] .

$$(\nabla U)_P = \frac{1}{V_P}\left(\sum_{Q \in K_N(P)} \frac{1}{2}\left(U_P + U_Q\right) \cdot \vec{n}_{PQ} + \sum_{(K_{out} \in \partial C_P \cap \Gamma)} U_P \cdot \vec{n}_{out} \right) \quad (4.30)$$

4.3.3 Boundary Conditions

In order to compute the flux balance of nodes that reside in the computational boundary domain, additional fluxes have to be encountered with the enforcement of the appropriate boundary conditions depending on the type of, i.e. wall, inlet, outlet and symmetry. Such fluxes are computed at the barycenter of each boundary face with the use of the arithmetic averages for the conservative variables of their nodes. These fluxes are assigned to the nodes weighted by the area of the face which corresponds to them.

In inlet boundary faces, a one-dimensional Riemann problem is employed between the face's barycenter and the far-field, while the obtained fluxes are distributed to the corresponding surrounding nodes. When the Steger-Warming scheme [Ste81, Lan98] is applied, is formulated as:

$$\vec{H}_{K,out}^{inv} = \underline{\tilde{A}}_K^{+} \vec{W}_K + \underline{\tilde{A}}_K^{-} \vec{W}_{out} \quad (4.31)$$

where subscript K represents the barycenter of the boundary face, while subscript *out* indicates the far field; the values of the variables of vector $\vec{W}_{out}$ are obtained either from the far field or the boundary barycenter, depending on the type of the flow (internal or external) [Hir90, Bla01]. The outlet boundary ones are treated in a similar manner.

As far as the wall boundary nodes are concerned, a free-slip boundary condition is implemented implicitly, by adding a flux with zero normal to the boundary face velocity V_n described in Equation 4.32 [Mav94]

$$V_n = \vec{V} \cdot \vec{\hat{n}}_{out} = 0 \tag{4.32}$$

where $\vec{\hat{n}}_{out} = (\hat{n}_{out,x}, \hat{n}_{out,y}, \hat{n}_{out,z})$ is the normal to the boundary face unitary vector (outward-positive). An example of such a vector is presented in Figure 4.4 for a tetrahedral element. In this figure, M stands for the median point of the boundary face and M_P, M_Q, M_R signify the median points of the corresponding edges. For the computation of the normal vector, all the aforementioned points are utilized. Eventually, the added free-slip convective flux is calculated as the following equation shows:

$$\vec{H}_{freeslip} = \begin{pmatrix} \rho V_n \\ \rho u V_n + p\,\hat{n}_{out,x} \\ \rho v V_n + p\,\hat{n}_{out,y} \\ \rho w V_n + p\,\hat{n}_{out,z} \\ (\rho E + p) V_n \end{pmatrix} = \begin{pmatrix} 0 \\ p\,\hat{n}_{out,x} \\ p\,\hat{n}_{out,y} \\ p\,\hat{n}_{out,z} \\ 0 \end{pmatrix} \tag{4.33}$$

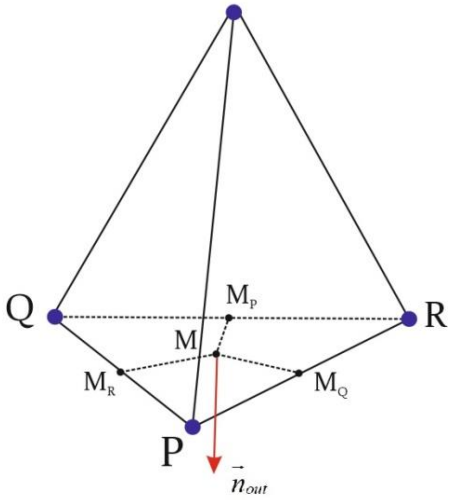


Figure 4.4: *Normal to the boundary face PQR vector* $\vec{n}_{out}$.

4.3.4 Time Integration

In an explicit scheme, the time integration of the discretized governing equation leads to the following equation [Bla01]

$$-V_P \left(\frac{d\vec{W}}{dt}\right)_P = -V_P \frac{\Delta \vec{W}_P^{n+1}}{\Delta t_P} = \sum_{Q \in K_N(P)} \vec{\Phi}_{PQ}^{inv} + \sum_{(K_{out} \in \partial CV_P \cap \Gamma')} \vec{\Phi}_{P,out}^{inv} - \vec{S}_P V_P = \vec{R}_P^n \tag{4.34}$$

where Δt_P is the local time step at node P and is calculated as [Kim03, Lyg11]

$$\Delta t_P^{inv} = CFL \cdot \frac{0.5 \alpha_{min\ l\ edge,P}}{|\vec{U}_P| + a_P} \tag{4.35}$$

where $|\vec{U}_P|$ is the value of velocity at node P, a_P is the speed of sound evaluated on the same node and $\alpha_{min\ l\ edge,P}$ is the length of the shortest edge connected to P.

In the second-order scheme, a four-step Runge-Kutta (RK (4)) method is employed as introduced in section 1.2.4. Furthermore, acceleration techniques, such as *implicit residual smoothing*, are utilized to decrease the time of the numerical simulation. In the concept of the high-order scheme, as described in the previous sections, the implementation of the high-order Strong Stability Runge-Kutta method is considered [Ruu05, Got05].

4.4 Derivation of the High-Order Scheme

4.4.1 Calculation of the High-order Terms

The functional form of the existing second-order scheme is shown below

$$U^2 = U(x_0, y_0, z_0) + \frac{\partial U}{\partial x}(x_i - x_0) + \frac{\partial U}{\partial y}(y_i - y_0) + \frac{\partial U}{\partial z}(z_i - z_0) \tag{4.36}$$

where a Taylor series expansion is applied, leading to the following functional form of a third order scheme [Yang16].

$$\begin{aligned} U^h = U(x_0, y_0, z_0) &+ \frac{\partial U}{\partial x}(x_i - x_0) + \frac{\partial U}{\partial y}(y_i - y_0) + \frac{\partial U}{\partial z}(z_i - z_0) + \frac{1}{2!}\left[\frac{\partial^2 U}{\partial x^2}(x_i - x_0)^2\right. \\ &+ \frac{\partial^2 U}{\partial y^2}(y_i - y_0)^2 + \frac{\partial^2 U}{\partial z^2}(z_i - z_0)^2 + 2\frac{\partial^2 U}{\partial x \partial y}(x_i - x_0)(y_i - y_0) \\ &\left. + 2\frac{\partial^2 U}{\partial x \partial z}(x_i - x_0)(z_i - z_0) + 2\frac{\partial^2 U}{\partial y \partial z}(y_i - y_0)(z_i - z_0)\right] \end{aligned} \tag{4.37}$$

What can be noted from the formulation above is that the first three terms in the right hand side are the U value of the existing scheme, but with the correction term as expressed in 4.38.

$$\begin{aligned} \Delta U^{h-2} = U^h - U^2 = \frac{1}{2}\left[\frac{\partial^2 U}{\partial x^2}(x_i - x_0)^2 + \frac{\partial^2 U}{\partial y^2}(y_i - y_0)^2 + \frac{\partial^2 U}{\partial z^2}(z_i - z_0)^2\right. \\ \left. + 2\frac{\partial^2 U}{\partial x \partial y}(x_i - x_0)(y_i - y_0) + 2\frac{\partial^2 U}{\partial x \partial z}(x_i - x_0)(z_i - z_0) + 2\frac{\partial^2 U}{\partial y \partial z}(y_i - y_0)(z_i - z_0)\right] \end{aligned} \tag{4.38}$$

Considering the methodology introduced in previous chapters, high-order accuracy is feasible, provided that the derivatives of the high-order terms can be computed. This is made possible by the consecutive implementation of the Green-Gauss theorem, which for a function f states that

$$\iiint_V \nabla f \, dV = \oiint_{\partial V} f \, \hat{n} \, dS \Rightarrow \nabla f = \frac{1}{V} \oiint_{\partial V} f \, \hat{n} \, dS \tag{4.39}$$

where $\hat{n}$ is the outward-pointing unitary normal vector to the boundary ∂V of V. Relying on the first derivatives field from the existing second-order scheme and using the Green-Gauss theorem, the third derivatives are determined in the following manner:

$$\frac{\partial^2 U}{\partial x^2}=\frac{1}{V}\oiint_{\partial V}\frac{\partial U}{\partial x}\hat{n}_x dS,\quad \frac{\partial^2 U}{\partial y^2}=\frac{1}{V}\oiint_{\partial V}\frac{\partial U}{\partial y}\hat{n}_y dS,\quad \frac{\partial^2 U}{\partial z^2}=\frac{1}{V}\oiint_{\partial V}\frac{\partial U}{\partial z}\hat{n}_z dS,$$

$$\frac{\partial^2 U}{\partial x\partial y}=\frac{1}{V}\oiint_{\partial V}\frac{\partial U}{\partial y}\hat{n}_x dS\quad,\quad \frac{\partial^2 U}{\partial x\partial z}=\frac{1}{V}\oiint_{\partial V}\frac{\partial U}{\partial z}\hat{n}_x dS\quad,\quad \frac{\partial^2 U}{\partial y\partial z}=\frac{1}{V}\oiint_{\partial V}\frac{\partial U}{\partial z}\hat{n}_y dS \tag{4.40}$$

The identical procedure could be used for the calculation of these derivatives based on the values of the first derivatives field, as shown in 4.41.

$$\frac{\partial^2 U}{\partial x^2}=\frac{\partial}{\partial x}\left(\frac{\partial U}{\partial x}\right)\quad,\quad \frac{\partial^2 U}{\partial y^2}=\frac{\partial}{\partial y}\left(\frac{\partial U}{\partial y}\right)\quad,\quad \frac{\partial^2 U}{\partial z^2}=\frac{\partial}{\partial z}\left(\frac{\partial U}{\partial z}\right)$$

$$\frac{\partial^2 U}{\partial x\partial y}=\frac{\partial}{\partial x}\left(\frac{\partial U}{\partial y}\right)\quad,\quad \frac{\partial^2 U}{\partial x\partial z}=\frac{\partial}{\partial x}\left(\frac{\partial U}{\partial z}\right)\quad,\quad \frac{\partial^2 U}{\partial y\partial z}=\frac{\partial}{\partial y}\left(\frac{\partial U}{\partial z}\right) \tag{4.41}$$

Additionally, the fourth order correction terms of Equation 4.42 are introduced to the third order interpolation function and, potentially, even higher order accuracy can be achieved.

$$\begin{aligned}\Delta U^{h-3}=\frac{1}{!3}\Bigg[&\frac{\partial^3 U}{\partial x^3}(x_i-x_0)^3+\frac{\partial^3 U}{\partial y^3}(y_i-y_0)^3+\frac{\partial^3 U}{\partial z^3}(z_i-z_0)^3\\&+3\frac{\partial^3 U}{\partial x^2\partial y}(x_i-x_0)^2(y_i-y_0)+3\frac{\partial^3 U}{\partial x^2\partial z}(x_i-x_0)^2(z_i-z_0)\\&+3\frac{\partial^3 U}{\partial y^2\partial x}(y_i-y_0)^2(x_i-x_0)+3\frac{\partial^3 U}{\partial y^2\partial z}(y_i-y_0)^2(z_i-z_0)\\&+3\frac{\partial^3 U}{\partial z^2\partial x}(z_i-z_0)^2(x_i-x_0)+3\frac{\partial^3 U}{\partial z^2\partial y}(z_i-z)^2(y_i-y_0)\\&+6\frac{\partial^3 U}{\partial x\partial y\partial z}(x_i-x_0)(y_i-y)(z_i-z_0)\end{aligned} \tag{4.42}$$

Again, taking into account the values of the third derivatives field, the terms of 4.42 are computed as follows:

$$\frac{\partial^3 U}{\partial x^3}=\frac{1}{V}\oiint_{\partial V}\frac{\partial^2 U}{\partial x^2}\hat{n}_x dS\quad,\quad \frac{\partial^3 U}{\partial y^3}=\frac{1}{V}\oiint_{\partial V}\frac{\partial^2 U}{\partial y^2}\hat{n}_y dS\quad,\quad \frac{\partial^3 U}{\partial z^3}=\frac{1}{V}\oiint_{\partial V}\frac{\partial^2 U}{\partial z^2}\hat{n}_z dS$$

$$\frac{\partial^3 U}{\partial x^2\partial y}=\frac{1}{V}\oiint_{\partial V}\frac{\partial^2 U}{\partial x\partial y}\hat{n}_x dS\quad,\quad \frac{\partial^3 U}{\partial x^2\partial z}=\frac{1}{V}\oiint_{\partial V}\frac{\partial^2 U}{\partial x\partial z}\hat{n}_x dS\quad,\quad \frac{\partial^3 U}{\partial y^2\partial x}=\frac{1}{V}\oiint_{\partial V}\frac{\partial^2 U}{\partial y\partial x}\hat{n}_y dS$$

$$\frac{\partial^3 U}{\partial y^2\partial z}=\frac{1}{V}\oiint_{\partial V}\frac{\partial^2 U}{\partial y\partial z}\hat{n}_y dS\quad,\quad \frac{\partial^3 U}{\partial z^2\partial x}=\frac{1}{V}\oiint_{\partial V}\frac{\partial^2 U}{\partial z\partial x}\hat{n}_z dS\quad,\quad \frac{\partial^3 U}{\partial z^2\partial y}=\frac{1}{V}\oiint_{\partial V}\frac{\partial^2 U}{\partial z\partial y}\hat{n}_z dS \tag{4.43}$$

$$\frac{\partial^3 U}{\partial x\partial y\partial z}=\frac{1}{V}\oiint_{\partial V}\frac{\partial^2 U}{\partial y\partial z}\hat{n}_x dS$$

4.4.2 U-MUSCL Scheme

The current methodology incorporates a variable extrapolation, the U-MUSCL numerical scheme, closely resembled to the existing MUSCL-scheme [Buro5], whose interpolation function is expressed in the following equation [Yan16].

$$U_{PQ}^{L}(\kappa) = U_P + \frac{\kappa}{2}\left(U_Q - U_P\right) + \frac{1}{2}\cdot(1-\kappa)\nabla U_P \cdot \vec{r}_{PQ}$$
$$U_{PQ}^{R}(\kappa) = U_Q + \frac{\kappa}{2}\left(U_P - U_Q\right) - \frac{1}{2}\cdot(1-\kappa)\nabla U_Q \cdot \vec{r}_{PQ} \tag{4.44}$$

Finally, the presented high-order scheme of 3rd order of accuracy is written in a similar manner as [Yan16]:

$$\begin{aligned}
U_{PQ}^{L}(\kappa) &= U_P + \frac{\kappa}{2}\left(U_Q - U_P\right) + \frac{1}{2}\cdot(1-\kappa)\nabla U_P \cdot \vec{r}_{PQ} \\
&\quad + \frac{1}{2}\left[\frac{\kappa_3}{4}\left(\nabla U_Q \cdot \vec{r}_{PQ} - \nabla U_P \cdot \vec{r}_{PQ}\right) + \frac{1}{4}(1-\kappa_3)\,\nabla\left(\nabla U_P \cdot \vec{r}_{PQ}\right)\cdot \vec{r}_{PQ}\right] \\
&= U_P + \tfrac{\kappa}{2}\left(U_Q - U_P\right) + \tfrac{1}{2}\cdot(1-\kappa)\nabla U_P \cdot \vec{r}_{PQ} \\
&\quad + \frac{1}{2}\left[\frac{\kappa_3 \Delta x_{PQ}}{4}\left(\left(\frac{\partial U}{\partial x}\right)_Q - \left(\frac{\partial U}{\partial x}\right)_P\right) + \frac{1}{4}\cdot(1-\kappa_3)\Delta x_{PQ}\nabla\left(\left(\frac{\partial U}{\partial x}\right)_P\right)\cdot \vec{r}_{PQ}\right] \\
&\quad + \frac{1}{2}\left[\frac{\kappa_3 \Delta y_{PQ}}{4}\left(\left(\frac{\partial U}{\partial y}\right)_Q - \left(\frac{\partial U}{\partial y}\right)_P\right) + \frac{1}{4}\cdot(1-\kappa_3)\Delta y_{PQ}\nabla\left(\left(\frac{\partial U}{\partial y}\right)_P\right)\cdot \vec{r}_{PQ}\right] \\
&\quad + \frac{1}{2}\left[\frac{\kappa_3 \Delta z_{PQ}}{4}\left(\left(\frac{\partial U}{\partial z}\right)_Q - \left(\frac{\partial U}{\partial z}\right)_P\right) + \frac{1}{4}\cdot(1-\kappa_3)\Delta z_{PQ}\nabla\left(\left(\frac{\partial U}{\partial z}\right)_P\right)\cdot \vec{r}_{PQ}\right]
\end{aligned} \tag{4.45}$$

CHAPTER 5

DEVELOPMENT OF 3-D GRID GENERATORS

5.1 Presentation of 3-D Grids

The implementation of the proposed high-order module, introduced in this study, into the current CFD *Galatea* solver requires the quality assessment of the numerical results. In the 3rd Chapter an extensive evaluation methodology was presented for the verification test of the 2-dimensional equations, using representative grid types. In order to employ this methodology to a 3-dimensional problem, computational meshes of the same dimensional order need to be constructed. Regarding the latter, it has to be generated so that it preserves the conservation properties of the governing equations; thus, the following conditions need to be satisfied [Bla01]:

- The physical domain has to be covered completely by the grid
- There must be no free space left between elements
- The grid cells should not overlap each other

Strictly in mathematical terms, this is expressed by the following: considering a conforming decomposition of the computational domain T^{h_N} of Ω with characteristic length h_N, as a set of finitely element subsets $T_p \subset \Omega,\ P = 1,2,3 \dots, N$, the following conditions are satisfied [Del11, Del13]:

- $\Omega = \bigcup_{p\in\{1,2,3\dots,N\}} T_p$
- every T_p is closed
- for two $T_p, T_q \in T^{h_N}$ with $p \neq q$ their interiors satisfy $\dot{T}_p \cap \dot{T}_q \neq \emptyset$
- every two dimensional face of any $T_p \in T^{h_N}$ is either a subset of $\partial\Omega$ or a side of another $T_q,\ q \neq p$

Various types of grids were developed for the scope of this work. The elements composing the different types of grids are tetrahedrons, pyramids and prisms (Figure 5.1).

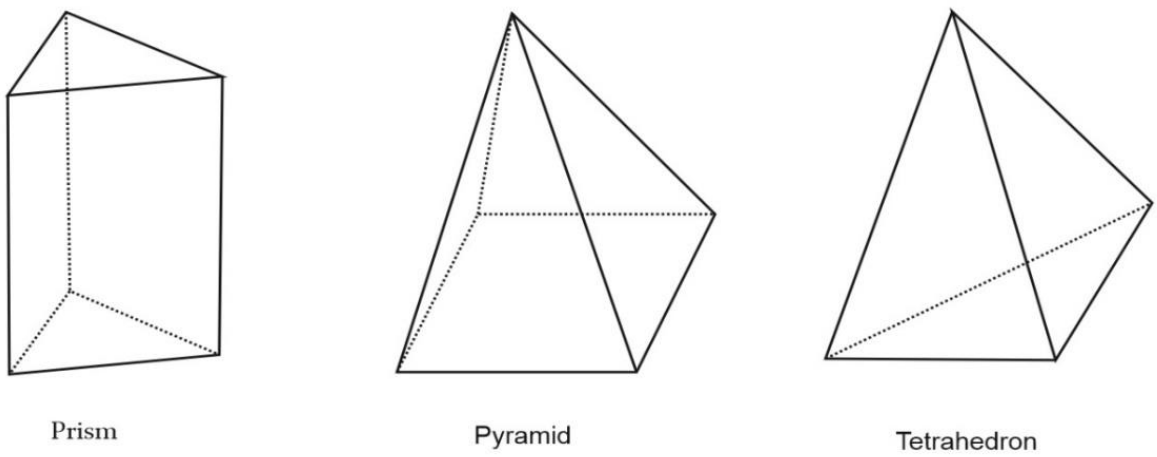


Figure 5.1: *Different types of 3-D elements*

Generally, as for the 2-dimensional grids introduced in the previous sections, the produced 3-D are Regular grids, derived by a smooth mapping from grids with periodic node connectivity, periodic cell distribution, and include (but are not limited to) grids derived from Cartesian ones. Additionally, a small perturbation of the initial node locations may derive Irregular grids from the Regular ones. Overall, 6 types of grids are studied in the present work, as shown in the following table:

Table 5.1: *Regular and Irregular grid types*

GRID TYPES	MODE I	MODE II
Prismatic Grid of Type I	*Regular*	*Irregular*
Prismatic Grid of Type II	*Regular*	*Irregular*
Prismatic Grid of Type III	*Regular*	*Irregular*
Pyramidal Grid	*Regular*	*Irregular*
Tetrahedral Grid	*Regular*	*Irregular*

A brief discussion of the grids precedes a comprehensive description of the construction method at the next sections. The Prismatic Grid of Type I consists of prisms derived from a regular Cartesian grid through the decomposition of its hexahedral elements with a diagonal splitting (Figures 5.2-5.3). Two prismatic elements are produced by this procedure. The Prismatic Grid of Type II is generated in a similar way, the only difference being that the outcome of the hexahedron elements' division includes 4 prisms (Figures 5.4–5.5). Grids of Type III may be regarded as an extrusion of a two-dimensional grid composed of triangular equilateral elements to a third dimension creating thus; prismatic elements (Figures 5.6-5.7). The Pyramidal Grid is created when the hexahedral cells of a regular Cartesian grid are decomposed. The insertion of a node at its barycenter and its connectivity to the corresponding vertices produces 6 pyramids (Figures 5.8 –5.9). Additionally, two of the abovementioned types of grid, the Pyramidal and the Prismatic with equilateral triangular base, are used to derive the Tetrahedral Grids by decomposing their elements into tetrahedrons (Figures 5.10-5.13). Finally, random shifts of the original grid node positions cause distortions that provide the Irregular Grids (Figures 5.14-5.15). This is possible for all grid types. These constructed grids are considered typical of those that are usually applied for the numerical solution of the Navies-Stokes equations.

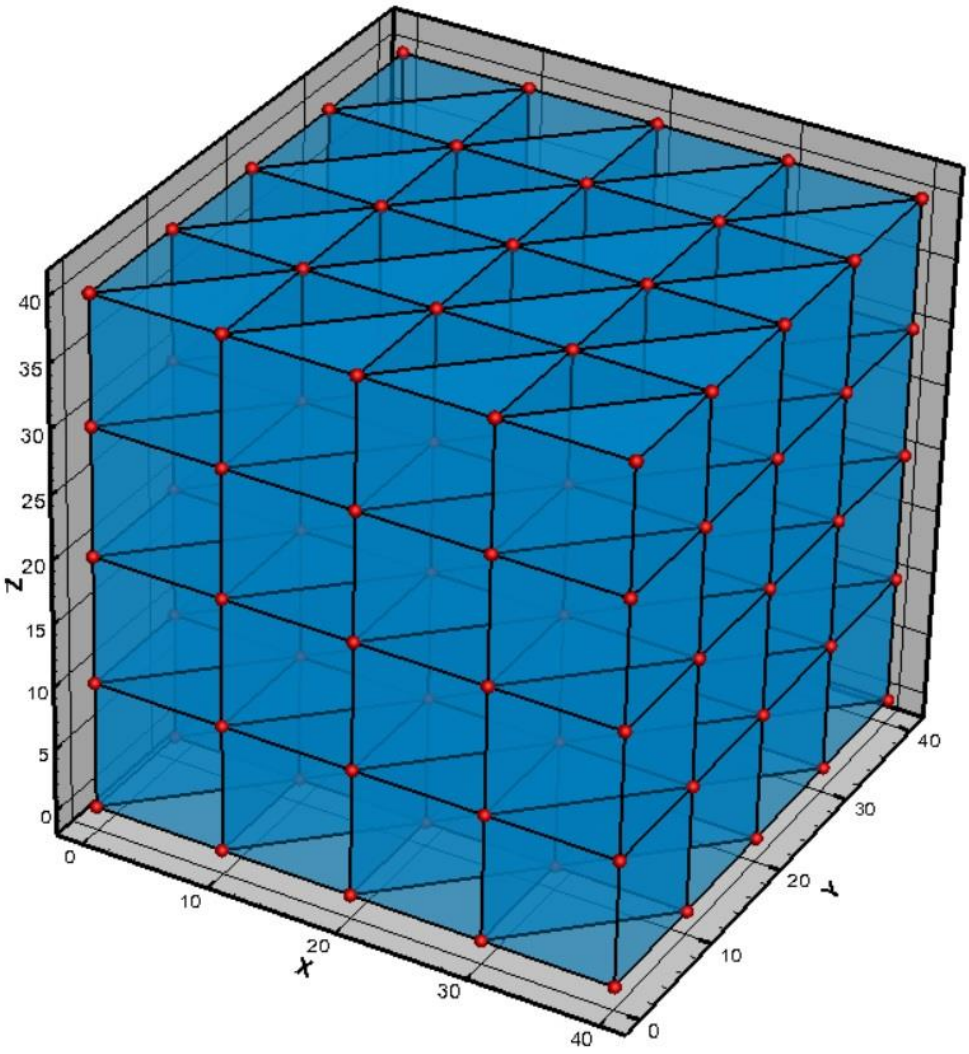


Figure 5.2: *Prismatic Grid of Type I*

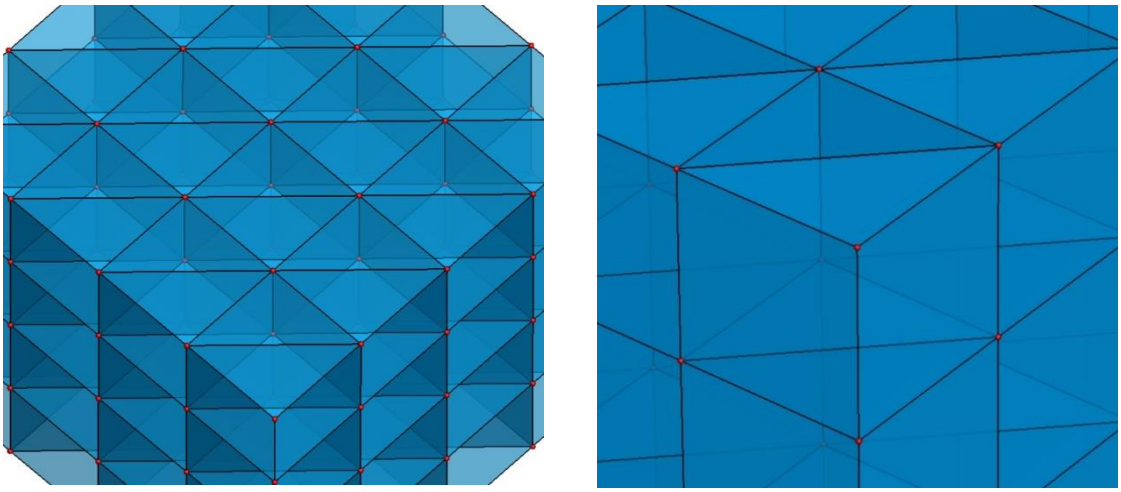


Figure 5.3: *Prismatic Grid of Type I - details*

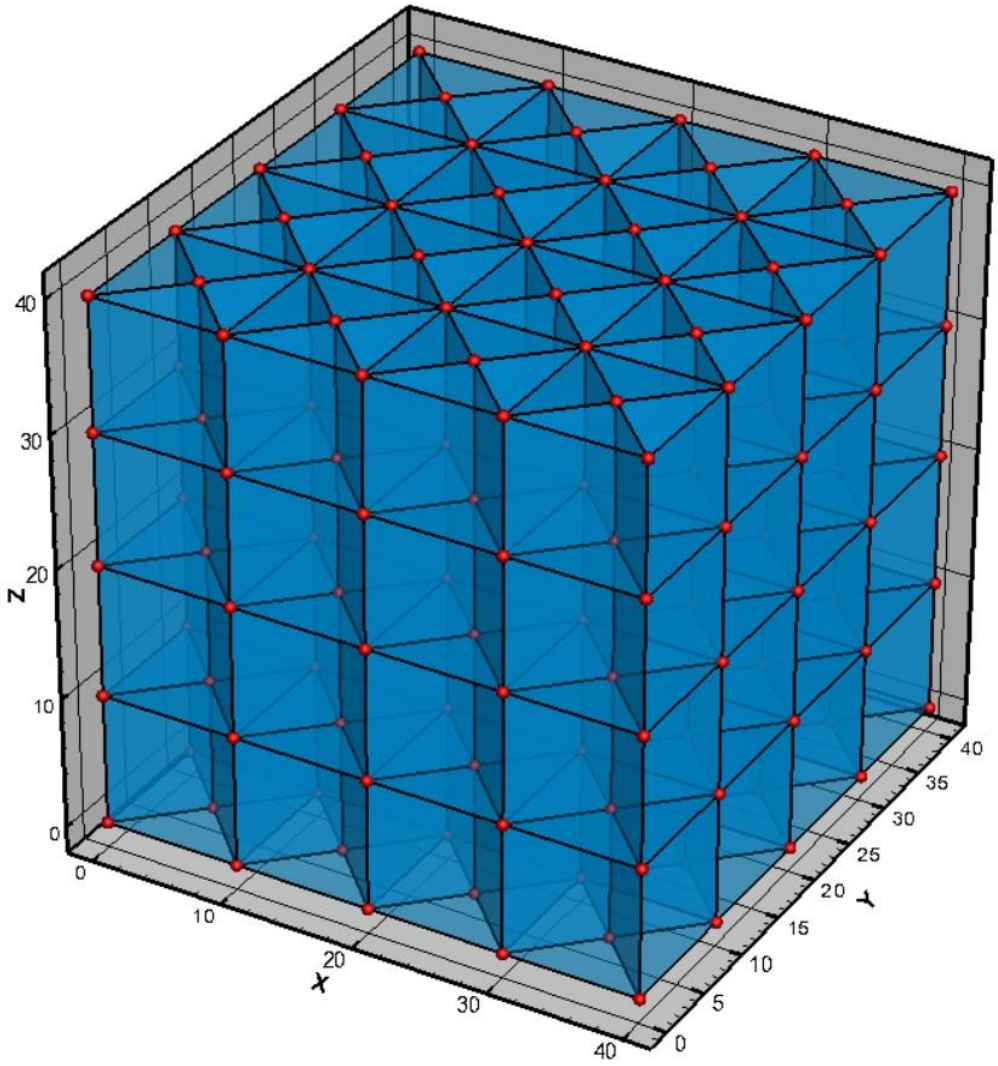


Figure 5.4: *Prismatic Grid of Type II*

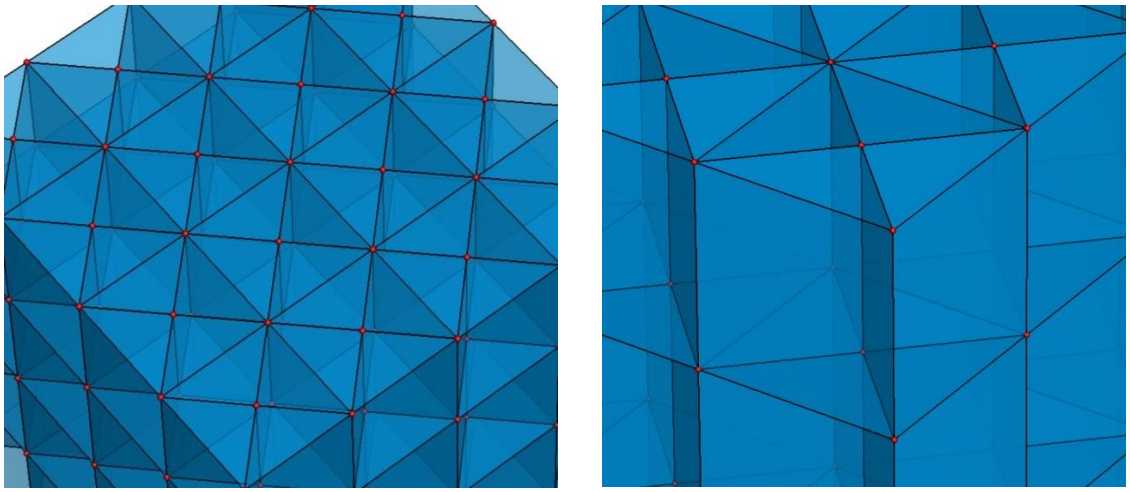


Figure 5.5: *Prismatic Grid of Type II - details*

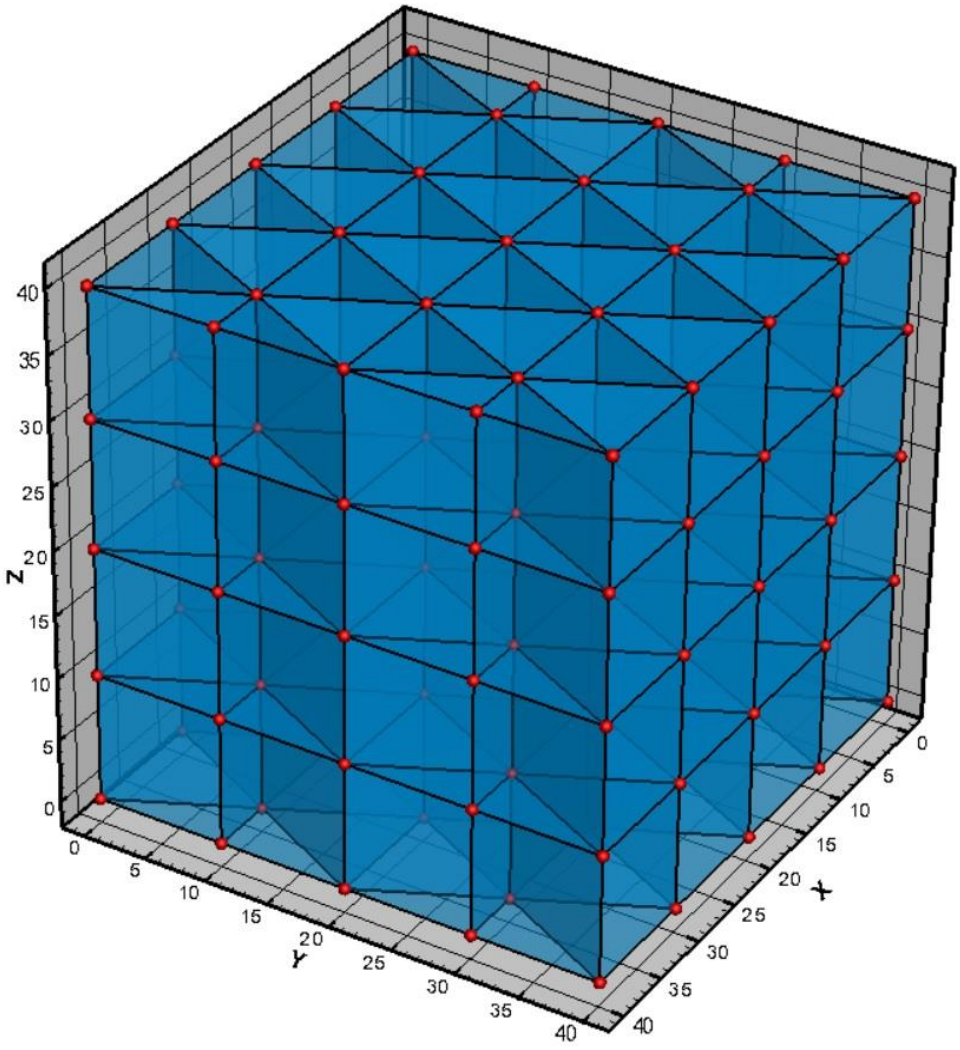


Figure 5.6: *Prismatic Grid of Type III*

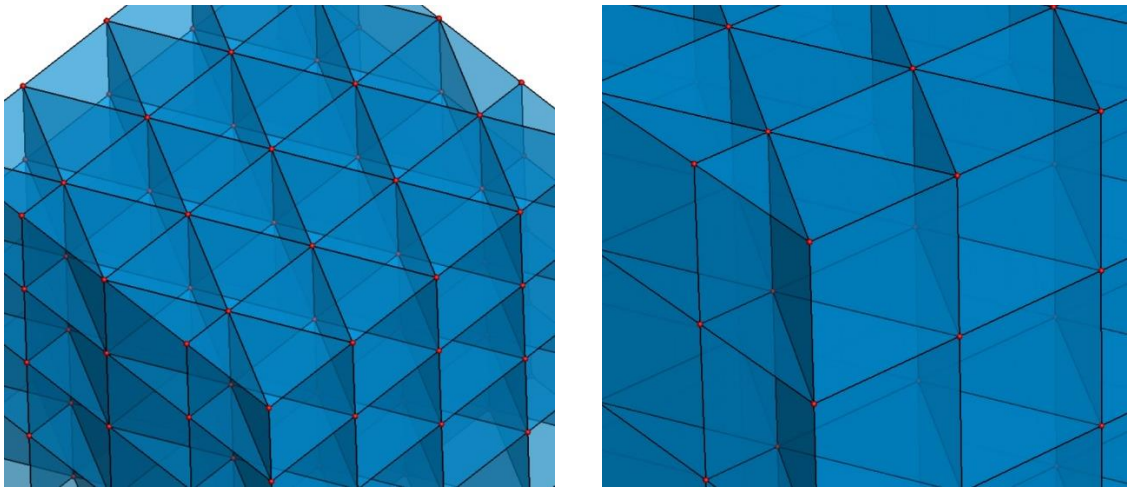


Figure 5.7: *Prismatic Grid of Type III - details*

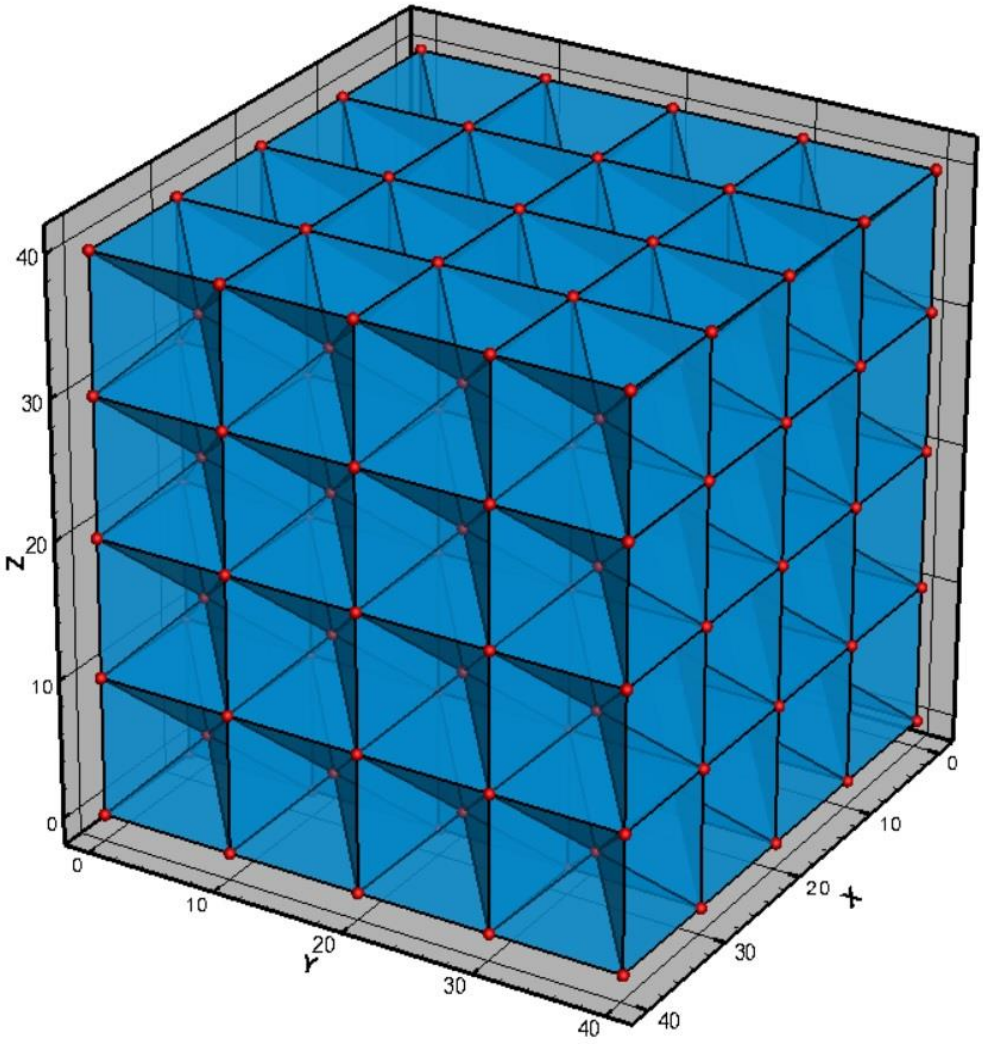


Figure 5.8: *Pyramidal Grid*

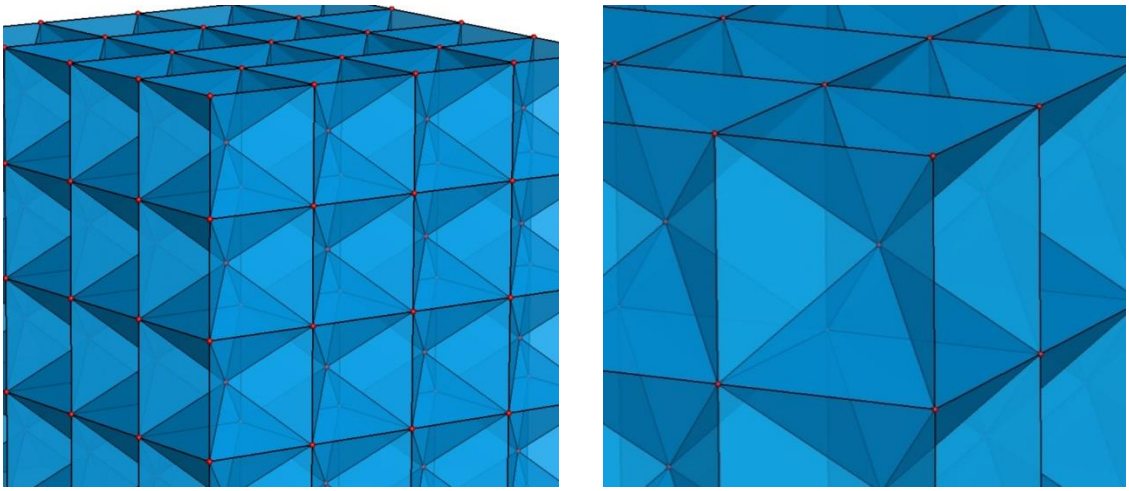


Figure 5.9: *Pyramidal Grid - details*

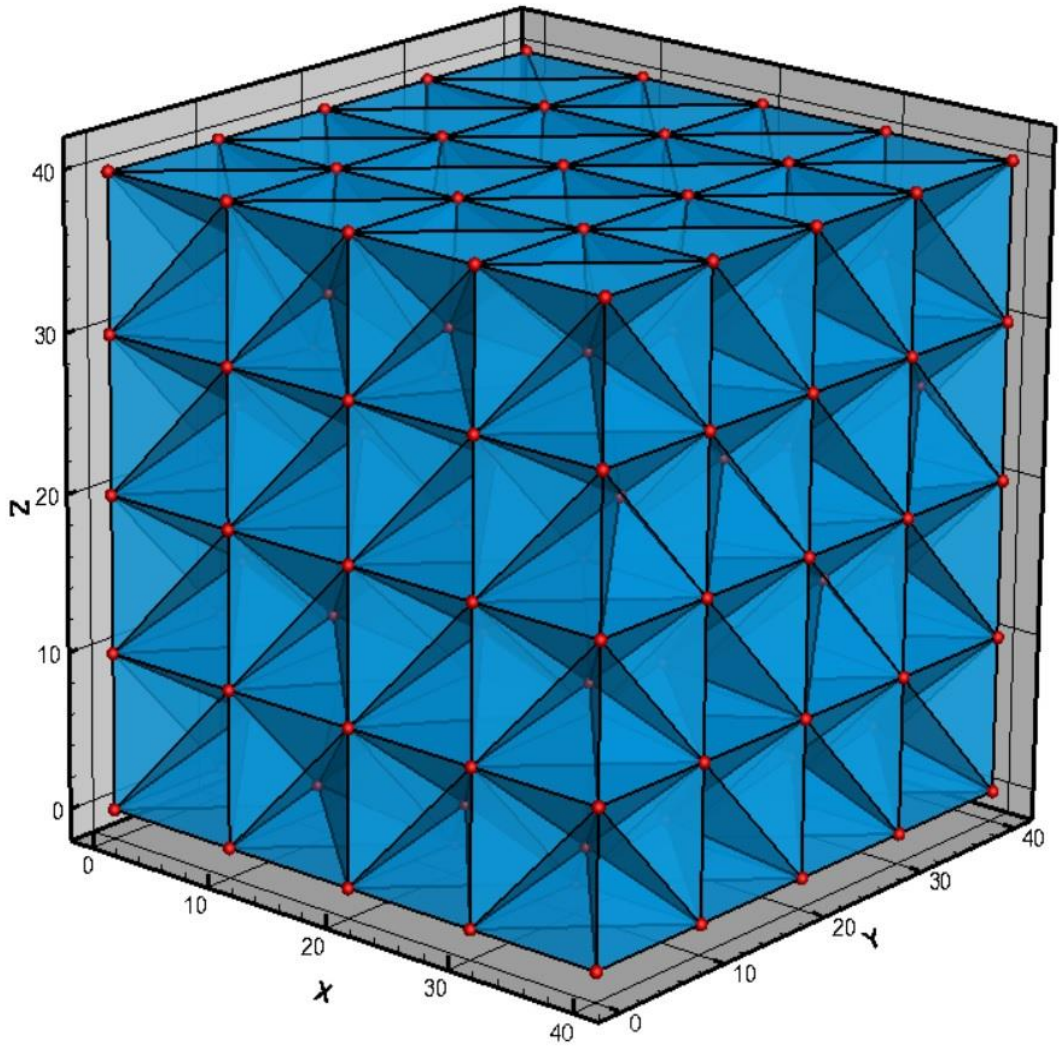


Figure 5.10: *Tetrahedral Grid produced from the Pyramidal Grid*

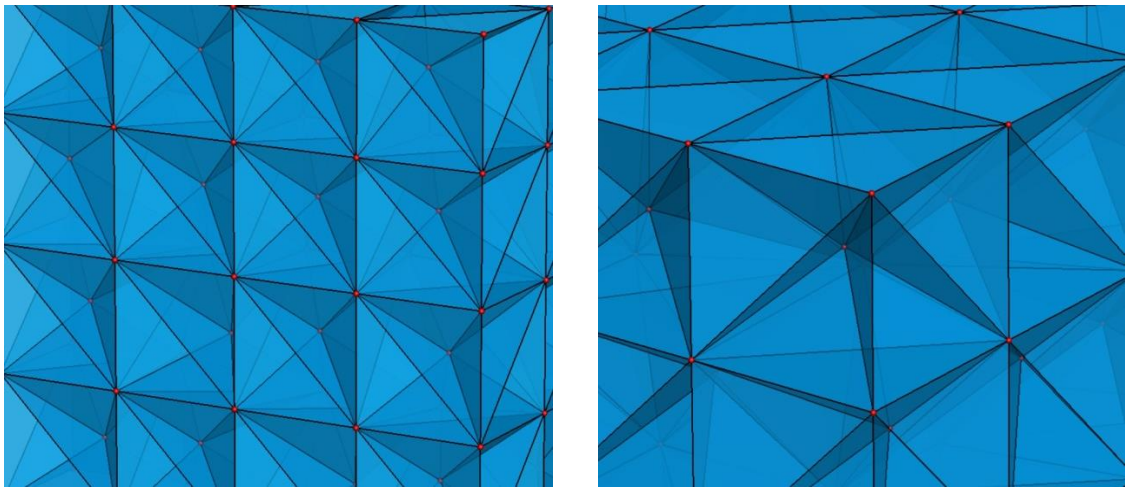


Figure 5.11: *Tetrahedral Grid produced from the Pyramidal Grid - details*

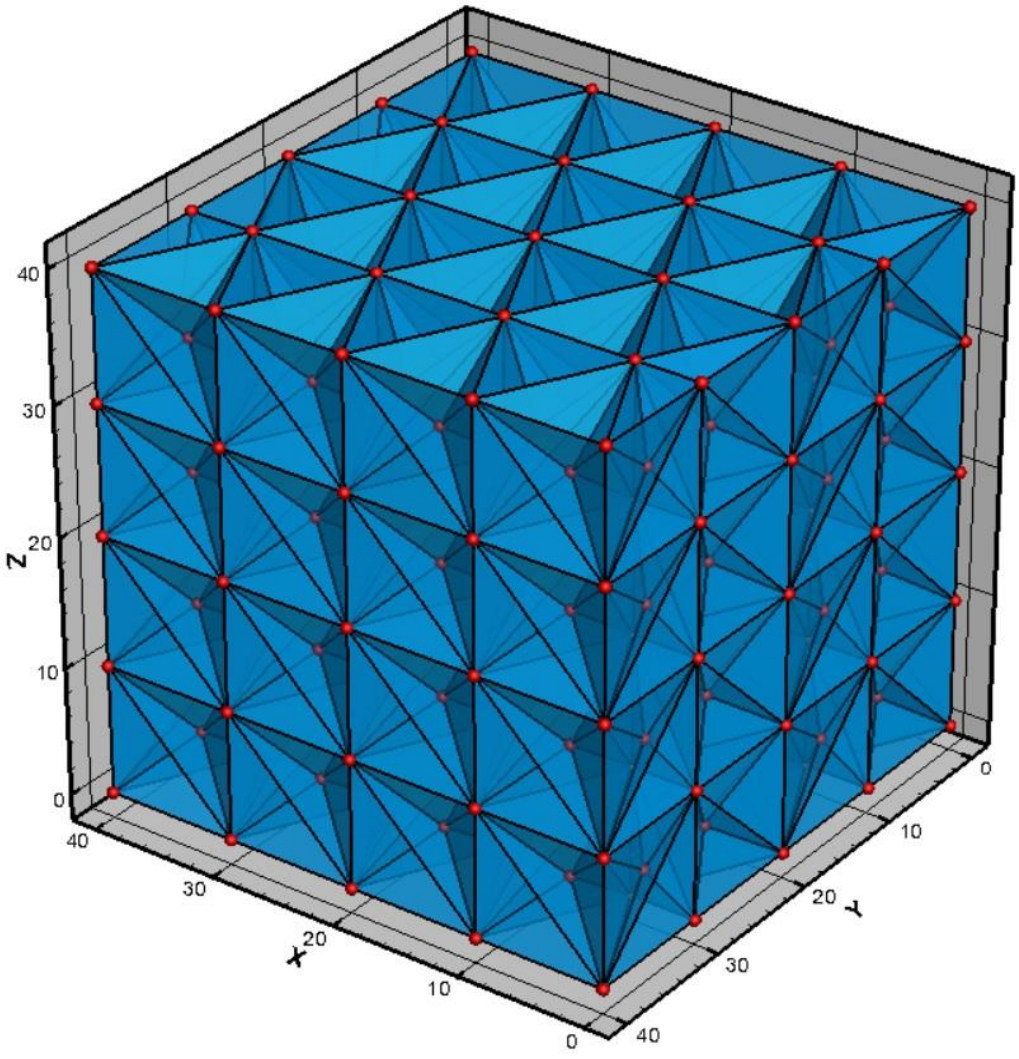


Figure 5.12: *Tetrahedral Grid produced from the Prismatic Grid of Type III*

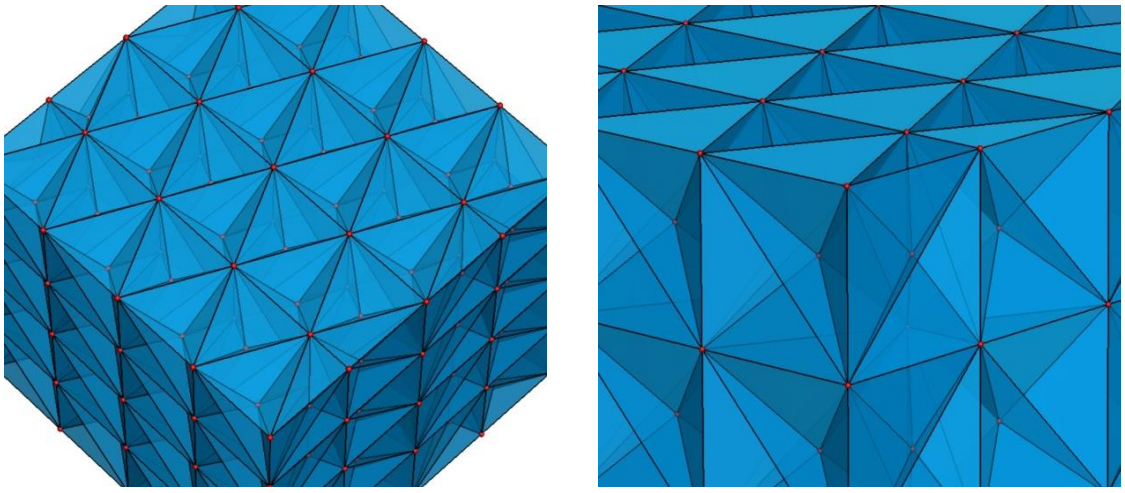


Figure 5.13: *Tetrahedral Grid produced from the Prismatic Grid of Type III - details*

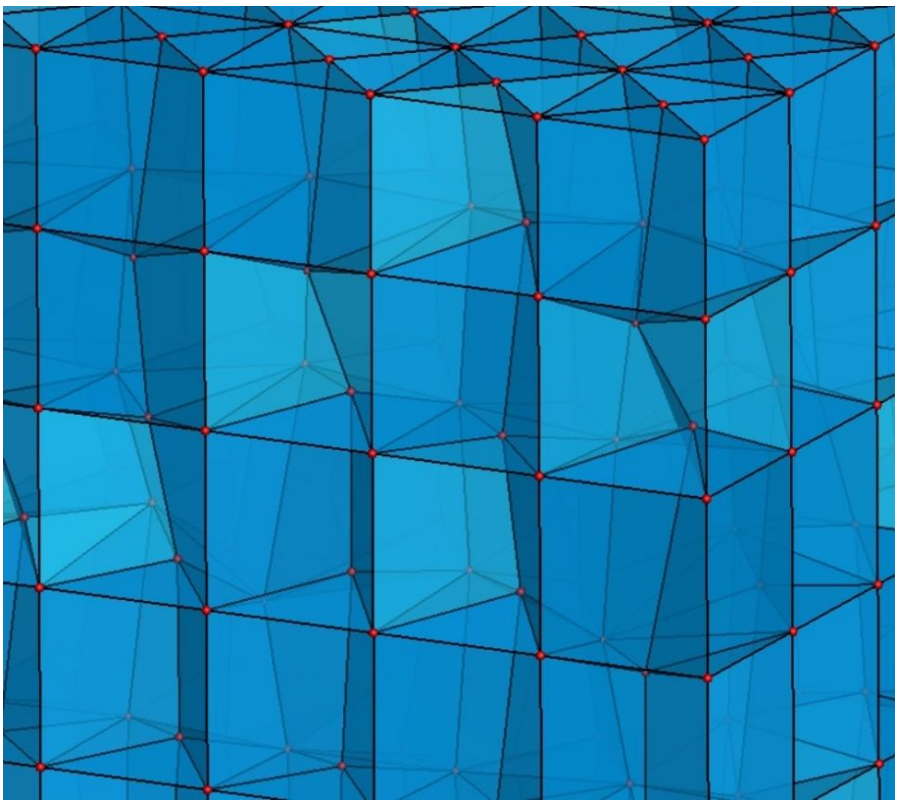


Figure 5.14: *Irregular mode of Prismatic Gird of Type II*

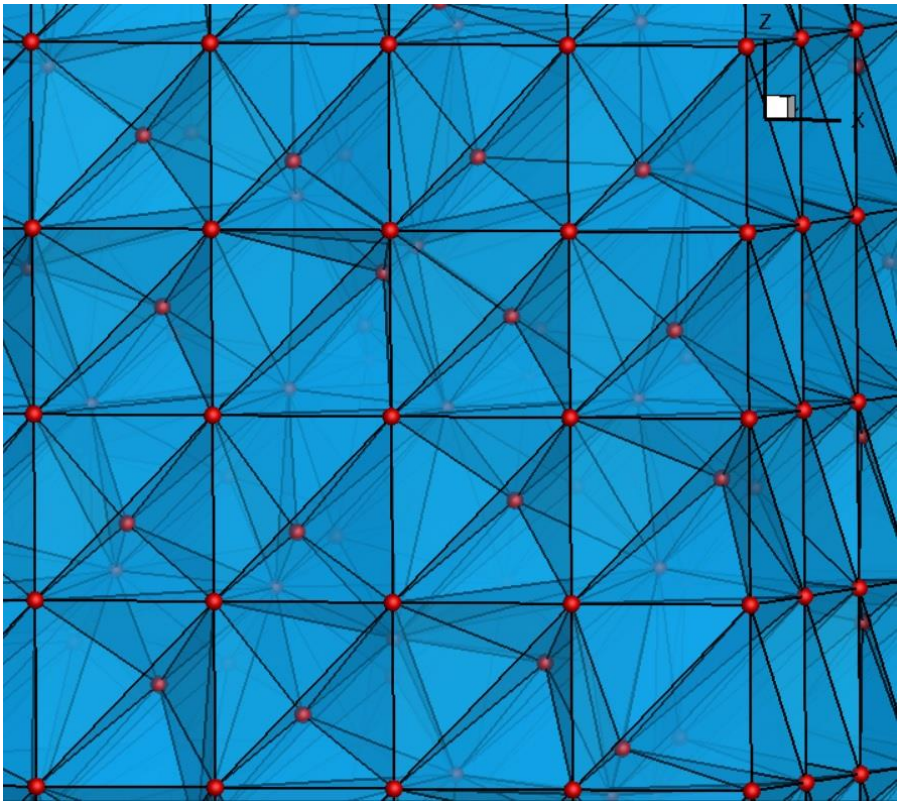


Figure 5.15: *Irregular mode of Tetrahedral Grid produced from the Pyramid Grid*

5.2 Introduction to the Algorithms

In this section a thorough representation of the developed algorithms is undertaken with a comprehensive description of the construction method for each type of grid. Representative figures throughout the consecutive stages of the development are given for deeper understanding. The source code of all grid generator types is developed in *FORTRAN 90*, which is a general-purpose, compiled imperative programming language, especially suited to numerical computation and scientific computing, prevailing in the Computational Fluid Dynamics field. In this introduction the focus will be on the key aspects of the algorithms identical for all grid types, i.e. the data structures, the boundaries of the computational domain and the output data.

As a primary step for each algorithm, an initialization procedure is executed in order to generate the nodes of the grid, where the assignment of the elements occurs. After an input file containing the variables for the specification of the grid is imported, the code proceeds with the calculation of all the nodes and cells composing the grid and stores them to the integer variables *NNODE* and *NCELL* respectively. With the determination of the nodes, the Cartesian coordinates (*X, Y, Z*) are allocated onto three one-dimensional arrays named *X (i), Y (i)* and *Z (i)* where the index defines the number of each node.

The following step includes defining the elements. This requires a set of information such as the nodes which compose the relevant element, its faces, and the neighboring cells. A two-dimensional array, named *NC (i, j)*, is declared in order to assign the nodes which compose a certain element. The former index, "*i*", stands for the node number of the element vertices, while the latter, "*j*" represents the number of the corresponding element. Depending on the element type, hexahedron, pyramid, or prism, the amount of nodes assigned is 4, 5 and 6. The orientation of the assignment is not arbitrary, but it is standardized by the *ANSYS* format and is depicted in Figure (5.16).

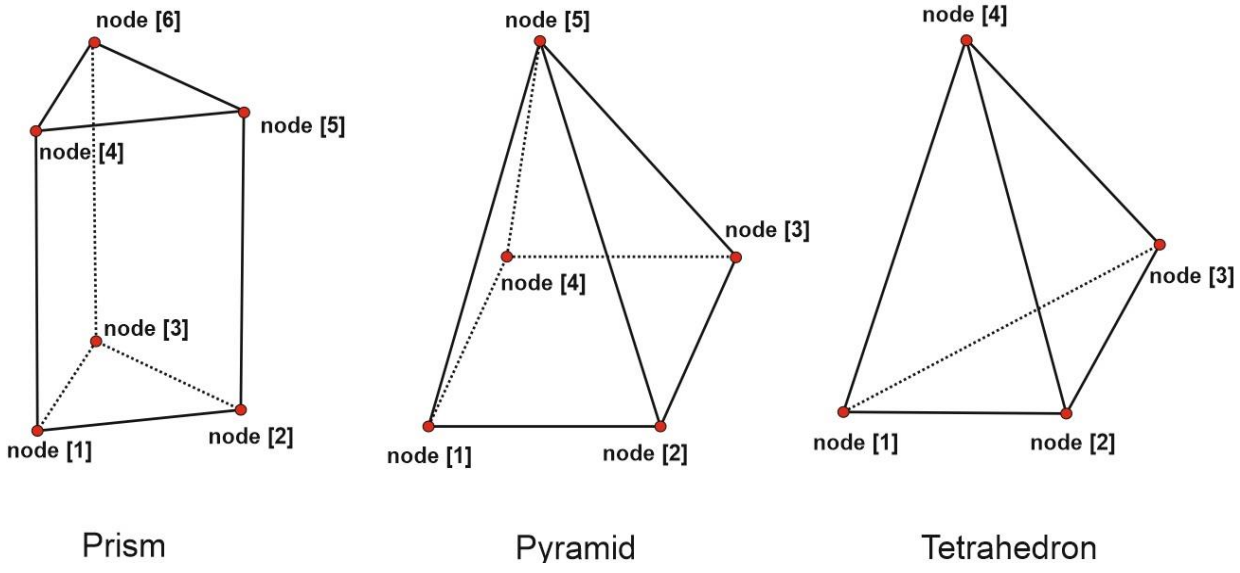


Figure 5.16: *Node orientation for each 3D element*

As far as the definition of the element faces is concerned, a three-dimensional array *NF (i, j, k)* is declared. In this matrix what is stored is the set of nodes, which determine each face of the element. The first index declares the number of the element face; the second states the number of each node composing the face; the third index refers to the number of the corresponding cell. Again, the number of faces depends on the element type, while the number of face nodes depends on the face shape.

Lastly, for each element the neighboring cells need to be determined. This requires a two-dimensional array *NE (i, j)* which stores the number of the neighboring cells for each element. The former index declares the successive neighbors and the latter the element itself. Five neighbors are declared for the pyramidal and prismatic elements and four neighbors for the tetrahedral.

The definition of the variables is not over unless the demarcation of the computational domain is specified. Because of its hexahedral shape, 6 boundary planes need to be determined as boundaries. By convention, each plane is attributed a specific name, as illustrated in Figure 5.17 and a particular index is assigned, as indicated in Table 5.3. The integer variable *NBOUND* refers to the total sum of the boundaries.

To define a boundary plane, it is necessary to determine the faces it consists of with a set of information; this includes the nodes and the cell to which the relevant face belongs. A given number is tied with each boundary face according to a specific orientation. Four two-dimensional arrays *NOD (k) (i, j)*, where *k=1, 2… 4* store the nodes. The first index corresponds to the number of the boundary face and the second to the number of the boundary plane. The cell that corresponds to each boundary face is allocated to a two-dimensional array *NBCELL (i, j)* where both indexes have the same connotation as before. Moreover, a third two-dimensional array *NBFACE (i, j)* states the number face of the cell that is identical to the boundary face. Finally, the total sum of the boundary faces for each plane is declared with one-dimensional array, the *NCB (NBOUND)*.

The index assignment on the faces of each boundary plane follows a specific perspective the algorithm retains on the computational domain. Marking *XZ_1* plane as default perspective (Figure 5.17), results in the rest of the planes' views as derivations from shifts on the axes of Coordinate System. In particular:

Table 5.2: *Perspective determination of the boundary planes*

DEFAULT PERSPECTIVE	SHIFT ON AXIS	OUTCOME
XZ_*1* Plane	*Z by 90°*	*YZ_2*
	Z by 180°	*XZ_2*
	Z by 270°	*YZ_1*
	X by 90°	*XY_1*
	X by 270°	*XY_2*

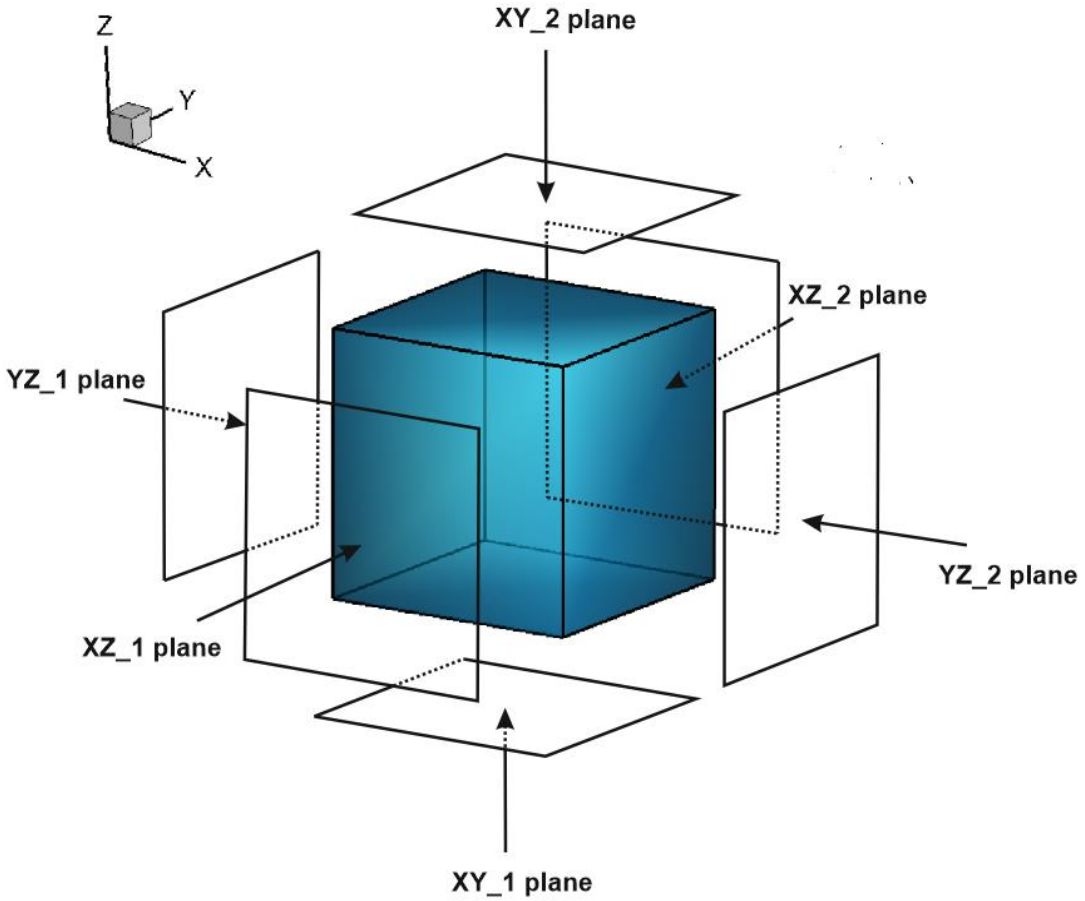


Figure 5.17: *Notation of boundary planes*

Table 5.3**:** *Boundary index assignment*

BOUNDARY INDEX ASSIGNING	PLANES
Boundary [1]	*XZ_1 Plane*
Boundary [2]	*YZ_2 Plane*
Boundary [3]	*XZ_2 Plane*
Boundary [4]	*YZ_1 Plane*
Boundary [5]	*XY_1 Plane*
Boundary [6]	*XY_2 Plane*

The following table summarizes the data structures that are described in this section.

Table 5.4: *Summarizing table of data structures*

NOTION	TYPE	DESCRIPTIOIN
X, Y, Z	*Array*	*Coordinates of the nodes*
NCELL	*Integer*	*Number of total cells*
NNODE	*Integer*	*Number of total nodes*
NC	*Array*	*Node number in each cell*
NF	*Array*	*Node number of each face in each cell*
NE	*Array*	*Cell number of each neighbor*
NBOUND	*Integer*	*Number of total boundary planes*
NCB	*Array*	*Number of cell faces on each boundary plane*
NOD(i)	*Array*	*Nodes of each boundary face*
NBCELL	*Array*	*Number of the corresponding boundary cell*
NBFACE	*Array*	*Number of corresponding face of each boundary cell*

The execution of the algorithms produces two types of output files for subsequent processing steps. The former is a text file with the extension .PLT which is a customized format for *TECPLOT*, a visualization and analysis software. The latter is in *ANSYS.CFX* format where the structure of the information is standardized and includes the following data:

Table 5.5: *Data output for ANSYS*

DATA	DESCRIPTION
Name of the Grid **Number of Nodes** **Number of Cells**	*Definition of primary information*
X,Y,Z Coordinates	*Definition of the node coordinates*
Nodes of each Element	*Definition of the nodes composing each element*
Boundary Cells and Faces	*Definitions of the boundary cell numbers along with the corresponding cell face numbers*

5.3 Regular Grids

5.3.1. Prismatic Grid of Type I

The algorithm produces a prismatic grid whose prisms have an orthogonal triangular basis (Figures 5.2 – 5.3) through the decomposition of a regular Cartesian grid. Initially, a set of input values is provided for the code execution. These values refer to the specifications of the generated grid determining the dimensions of the computational domain and, implicitly, the magnitude of the consisting elements. In the following table, the first column illustrates the input values the user defines and the second column captures the auxiliary variables the algorithm determines in order to store them.

Table 5.6: *Input values of Grid Type I and their corresponding variables*

INPUT VALUES	VARIABLES
Length of the Rectangle in the X – Direction	*X_L*
Length of the Rectangle in the Y – Direction	*Y_L*
Length of the Rectangle in the Z - Direction	*Z_L*
Number of Edges in the X – Direction	*NX*
Number of Edges in the Y – Direction	*NY*
Number of Edges in the Z – Direction	*NZ*

The length of the rectangle in three dimensions declares the magnitude of the hexahedral computational domain. The number of edges determines the segmentation in each direction, so that a regular Cartesian grid blueprint is constructed. Based on the input values, the total number of both nodes (*NNODE*) and elements (*NCELL*) is calculated.

At this stage the node initialization process occurs. On the Euclidean space the *X, Y, Z* Cartesian Coordinates of the nodes are determined along with their allocation onto the three corresponding arrays *X(i), Y(i)* and *Z(i)*, the index *i* signing the number of the node. The alliance of the nodes is set up in a fashion, such that the vertices of hexahedral elements, which define a regular Cartesian grid, are determined (Figure 5.18).

An iterative procedure consisting of three loops is implemented for the node initialization. Considering zero point of the Cartesian coordinates as the starting point, the first loop determines the nodes in the *Y*-direction, based on the input value length (*Y_L*). The distance between the nodes (*dy*) is defined by the division of the *Y*-direction length (*Y_L*) with the number of the edges (*NY*) in this direction. The same iterative process is repeated by a second loop in the *X*-direction defining the successive rows of nodes. Again, the distance (*dx*) between the rows is the division result of the (*X_L*) with the total amount of the edges (*NX*). Finally, a third loop completes the node determination in the *Z*-direction positioning the nodes on the following

successive levels. The indexing of the nodes follows the flow of the described iterative procedures and is depicted in Figure 5.18.

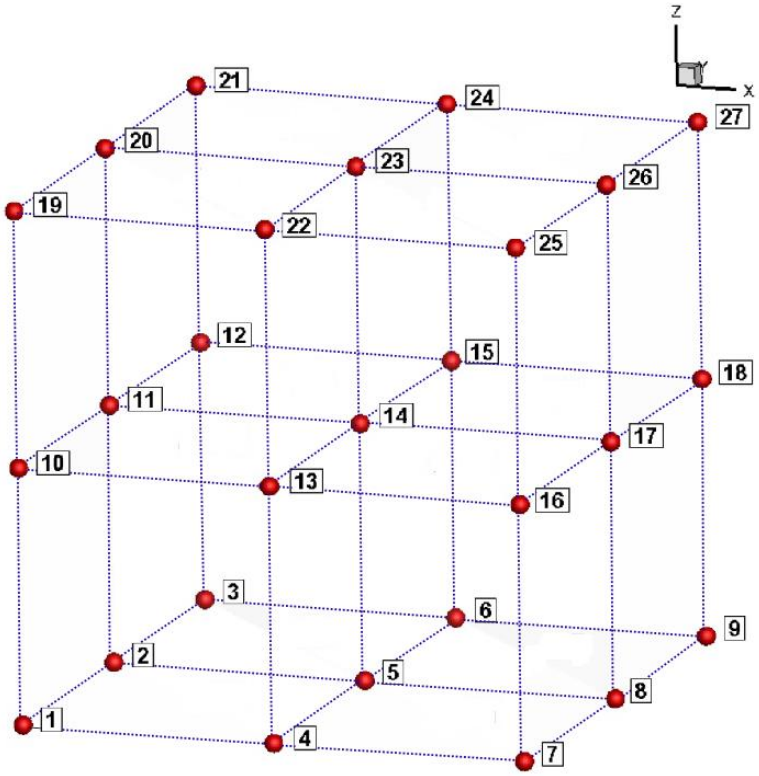


Figure 5.18: *Node orientation of the blueprint regular Cartesian grid for Prismatic Grid of Type I*

The next step is the definition of the elements. The main concept here is the construction of two prismatic cells with a diagonal decomposition of every hexahedral cell created by the node initialization (Figure 5.19).

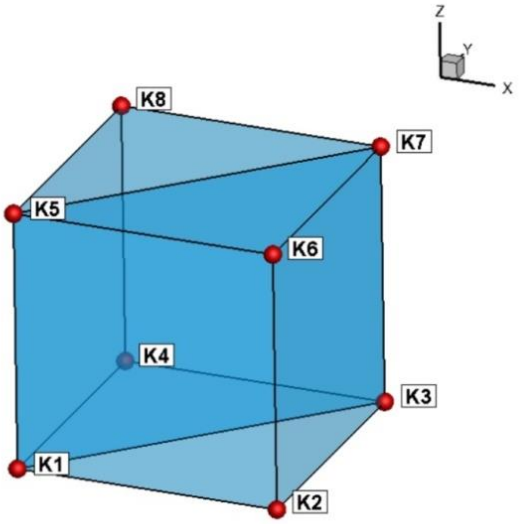


Figure 5.19: *Assignment of auxiliary variables on a hexahedral element*

Each hexahedral element is scanned by an iterative procedure and dummy variables are utilized to store temporarily the 8 node numbers of the vertices, in order to construct the prismatic cells. Subsequently, the nodes are distributed to the *NC (i, j)* array with a specific orientation, as mentioned in the previous section (Figure 5.16), defining thus two distinct elements. Figure 5.19 illustrates the location of the dummy variables, while Table 5.7 indicates the assigning of the nodes to the two prisms.

Table 5.7: *Node assignment to the two prisms*

DEFINITION	NODE (1)	NODE (2)	NODE (3)	NODE (4)	NODE (5)	NODE (6)
Prism [1]	*K1*	*K2*	*K3*	*K5*	*K6*	*K7*
Prism [2]	*K1*	*K3*	*K4*	*K5*	*K7*	*K8*

At the same stage, both the determination of the faces, which compose each cell and the neighboring cells takes place. With respect to the faces, the index numbering of each face depends on the nodes it consists of. The assignment of the nodes to the *NF (i, j, k)* occurs in accordance to a specific orientation. The final outcome is illustrated in Table 5.8. In connection with the neighboring cells, 5 neighbors are defined for the prismatic cells. The index number of each neighbor is dependent on the number face it is aligned with, as shown in Table 5.9.

Table 5.8: *Face indexing and node assignment on the NF array*

FACES	NODE ASSIGNMENT TO NF ARRAY			
	1st	**2nd**	**3rd**	**4th**
Face [1]	*Node (1)*	*Node (3)*	*Node (6)*	*Node (4)*
Face [2]	*Node (1)*	*Node (2)*	*Node (5)*	*Node (4)*
Face [3]	*Node (2)*	*Node (3)*	*Node (6)*	*Node (5)*
Face [4]	*Node (1)*	*Node (2)*	*Node (3)*	_
Face [5]	*Node (4)*	*Node (5)*	*Node (6)*	_

Table 5.9: *Neighboring cell indexing with reference to the corresponding face*

INDEXING NEIGHBORS	CORRESPONING FACE
Neighbor [1]	*Face [3]*
Neighbor [2]	*Face [1]*
Neighbor [3]	*Face [2]*
Neighbor [4]	*Face [4]*
Neighbor [5]	*Face [5]*

The abovementioned processes are executed for every hexahedral element, so that prisms are created across the domain. The direction of the flow procedure, which the algorithm follows in order to scan all hexahedrons, is similar to the node alliance. Ordering the hexahedral element that lies at the zero point of Cartesian Coordinates as the initial point, an iterative procedure of three loops is implemented. The first loop defines the elements in the *Y*-direction (Figure 5.20), while the second loop proceeds to the next rows in the *X*-direction (Figure 5.21). Lastly, the third loop is executed to the layers higher up in the *Z*-direction by determining the elements of the whole domain (Figure 5.22). Once again, the index cell numbering follows the flow of the iterative processes.

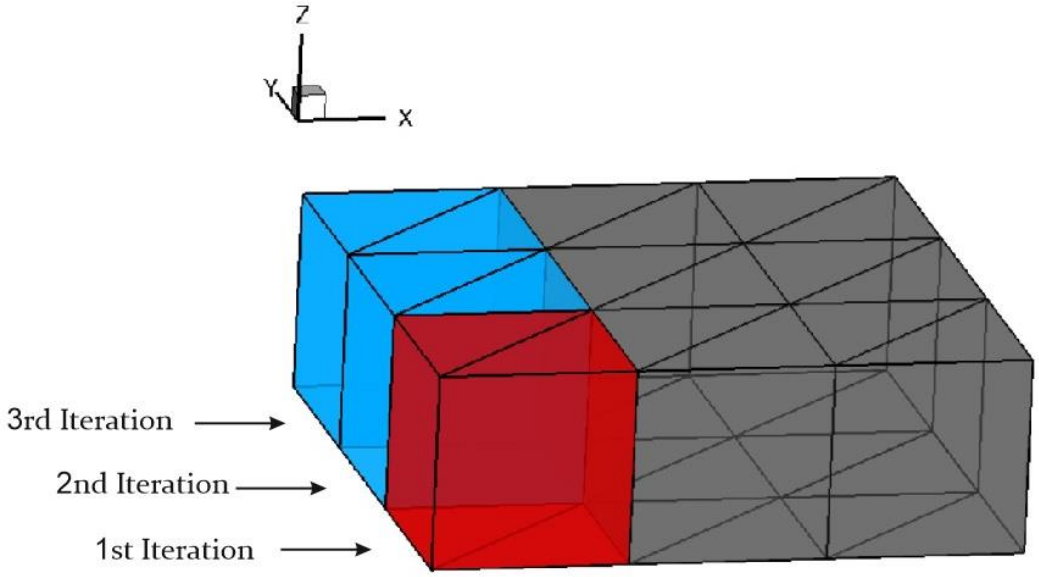


Figure 5.20: *Loop for cell definition in Y-direction*

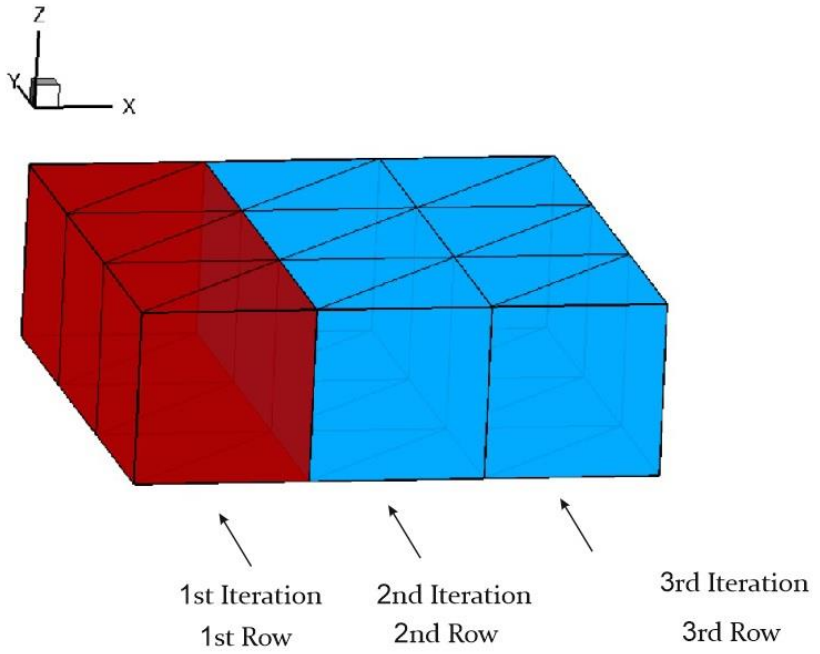


Figure 5.21: *Loop for cell definition in X-direction*

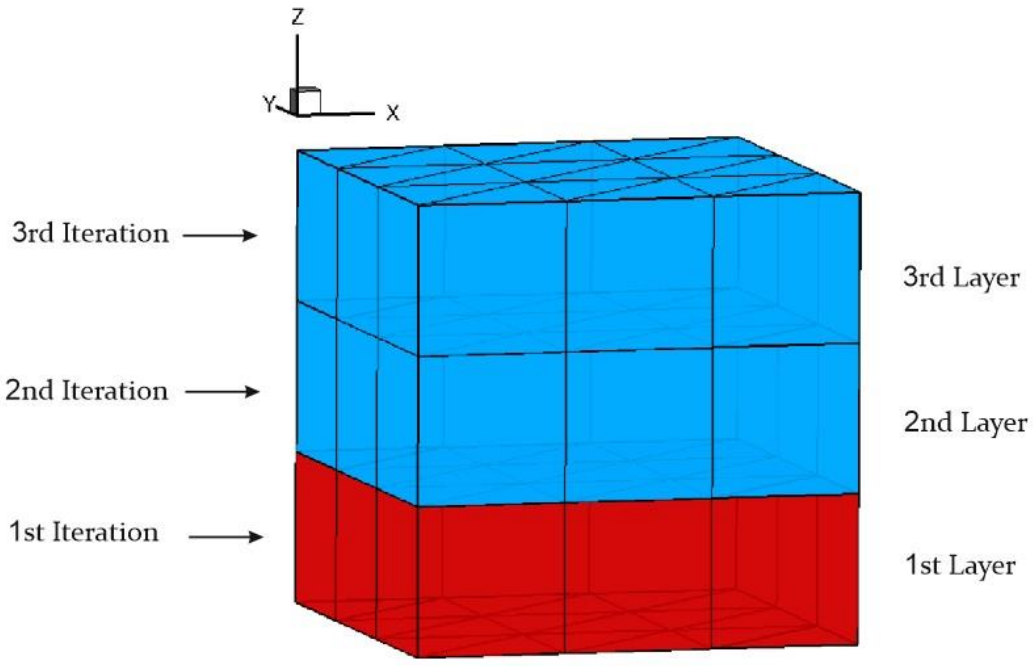


Figure 5.22: *Loop for cell definition in Z-direction*

The final stage of the algorithm execution contains the boundary determination of the grid domain. In the previous section, 6 boundary planes were defined because of the hexahedral shape of the domain. An iterative procedure is implemented for each boundary plane, in order to declare the boundary faces by assigning the number of nodes which compose each boundary face (*NOD*), the number of the corresponding cell (*NBCELL*), along with the corresponding element face (*NBFACE*). Two loops are used to define the two-dimensional planes: For the *XZ_1*, *XZ_2*, *YZ_1* and *YZ_2*, which are comprised of rectangular elements, the former loop defines the elements in the *Y*-direction, while the latter loop those in the *X*-direction. In Figures 5.23 – 5.24, the index numbering of these boundary faces and the orientation, which the nodes are assigned, are respectively shown. For the *XY_1* and *XY_2* planes, which are comprised of triangular elements, the iterative process follows the reverse flow: firstly, in the *X*-direction and then in the *Y*-direction, as illustrated in Figures 5.25-5.28. The difference in the orientation of the triangles on the two planes stems from the predetermined perspective view discussed in introduction (Figure 5.17).

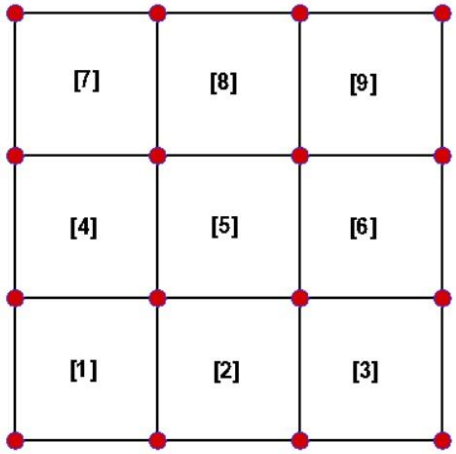


Figure 5.23: *Indexing of the boundary faces on XZ_1, XZ_2, YZ_1, and YZ_2 planes*

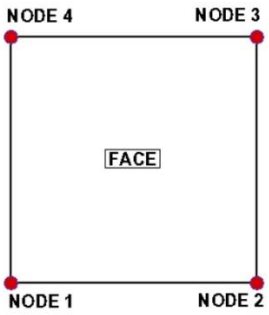


Figure 5.24: *Node assignment on each face on XZ_1, XZ_2, YZ_1, and YZ_2 planes*

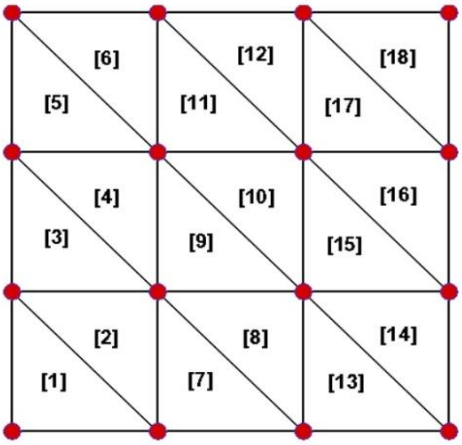


Figure 5.25: *Indexing of the boundary faces on XY_1 plane*

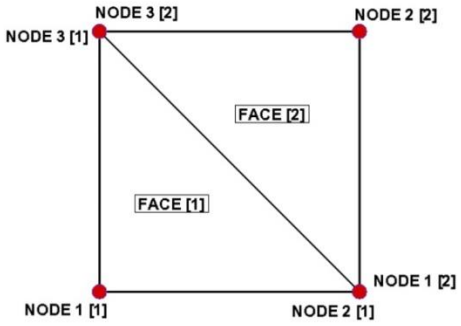


Figure 5.26: *Node assignment on each face on XY_1 plane*

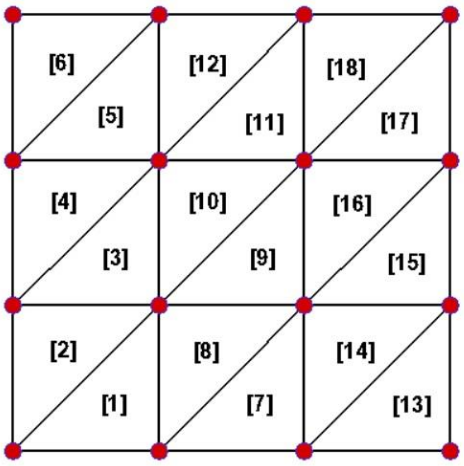


Figure 5.27: *Indexing of the boundary faces on XY_2 plane*

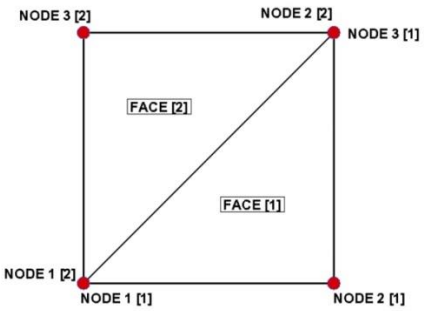


Figure 5.28: *Node assignment on each face on XY_2 plane*

5.3.2 Prismatic Grid of Type II

The aim of this section is the development of the algorithm that produces a grid of prismatic elements with a regular triangular base of Type II (Figures 5.4-5.5). A construction technique similar to the grid generation of Type I is employed, the main difference being the decomposition process of the Cartesian regular grid. More specifically, two diagonal planes divide each hexahedral element of the regular grid into 4 prismatic cells. The focus on this section will mainly be on those aspects that differentiate the proceeding steps of the algorithm development from the ones described for the development of the Grid of Type I.

An import file provides the data for the definition of the features about the grid (Table 5.6), i.e. the length of the rectangle in the three dimensions, in order to define the computational domain, and the number of edges, which are implicitly used for the segmentation of each direction. To begin with, the algorithm proceeds with the node initialization calculating the Cartesian Coordinates (*X,Y,Z*) and stores them in the corresponding arrays (*X(i),Y(i),Z(i)*). A blueprint of the nodes forming a regular Cartesian grid is created and extra nodes are set at the barycenter of each quadrilateral face formed by the nodes on each two-dimensional plane. These nodes are utilized for the division of the hexahedral elements.

The nodes of the regular grid are initialized with a procedure identical to the one described for Grid Type I. Three loops are executed in the relevant dimensions *X, Y, Z* and an index number is assigned for each node as shown in Figure 5.18. In addition, the auxiliary nodes are determined in accordance to the orientation of the nodes that define the Cartesian grid. The index numbering further continues the last arithmetic succession. Figure 5.29 illustrates the final outcome.

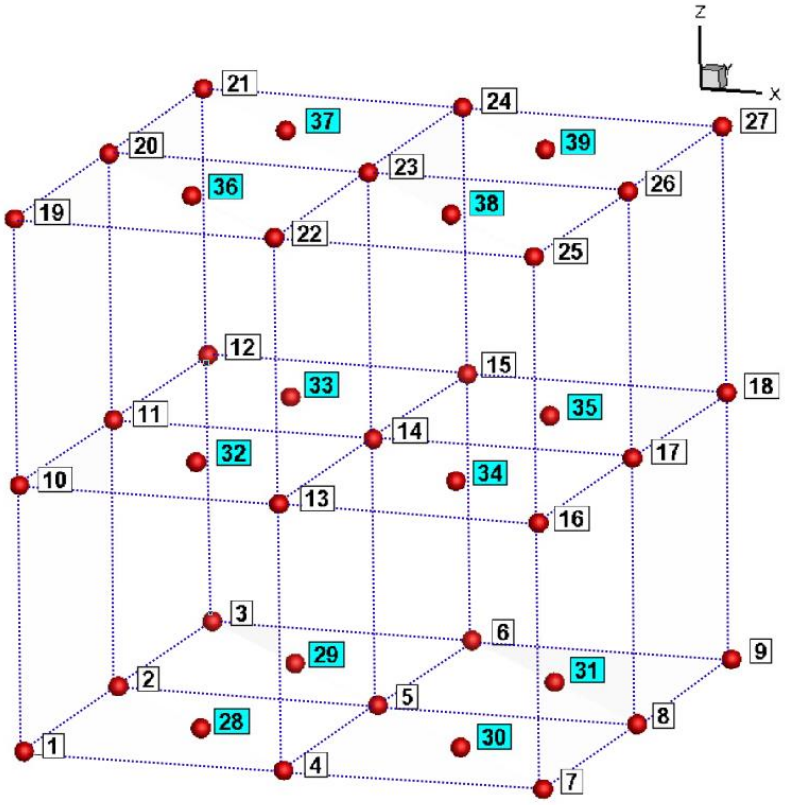


Figure 5.29: *Node orientation for the Prismatic Grid of Type II*

Proceeding to the next step, the algorithm determines the prismatic elements assigning the node set to the *NC* array for each element. The iterative process, which scans every hexahedral element in order to perform its decomposition into prisms, is similar to the one depicted in Figures 5.20-5.22. As far as the decomposition is concerned, 4 distinct prismatic elements are produced. The auxiliary variables, which include the vertex nodes of the hexahedron along with the middle nodes, are defined, as shown in Figure 5.30 and they are assigned to the prisms, as indicated in the Table 5.10. Finally, the assignment of the element faces (*NF*) as well as the determination of the neighboring cells (*NE*) occurs in the way that was described in Tables 5.8 and 5.9.

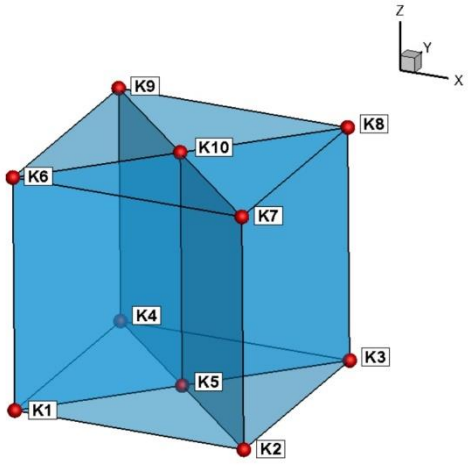


Figure 5.30: *Assignment of auxiliary variables of a hexahedral element*

Table 5.10: *Node assignment for the 4 prisms*

DEFINITION	NODE (1)	NODE (2)	NODE (3)	NODE (4)	NODE (5)	NODE (6)
Prism [1]	*K1*	*K2*	*K5*	*K6*	*K7*	*K10*
Prism [2]	*K2*	*K3*	*K5*	*K7*	*K8*	*K10*
Prism [3]	*K3*	*K4*	*K5*	*K8*	*K9*	*K10*
Prism [4]	*K4*	*K1*	*K5*	*K9*	*K6*	*K10*

The determination of the grid's boundary conditions completes the execution of the algorithm. An iterative procedure of two loops defines the two-dimensional boundary planes i.e. the nodes of the faces (*NOD*), the corresponding elements (*NBCELL*), and the respective element faces (*NBFACE*). The planes *XZ_1, XZ_2, YZ_1*, and *YZ_2* comprised of rectangular elements are assigned a consecutive numbering. The flow of the iterative process begins at the first row and proceeds on the layers above, as shown in Figures 5.31-5.32. On the other hand, with respect to the planes XY_1 and XY_2 composed of triangular elements, the iterative procedure assigns in a circular way a successive numbering to each triangular element found on each rectangle. This is performed, firstly, in the *Y*-direction and then in the *X*-direction. This is presented in Figures 5.33-5.34. Note that the perspective in the two-dimensional figures echoes the discussion in the introductory part.

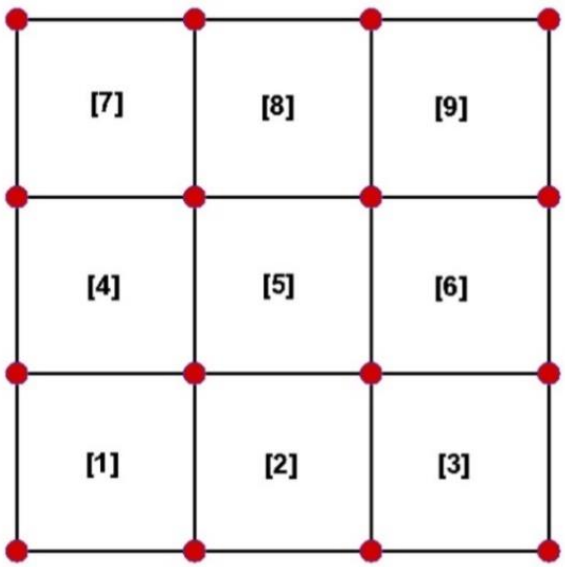


Figure 5.31: *Indexing of the boundary faces on XZ_1, XZ_2, YZ_1, and YZ_2 planes*

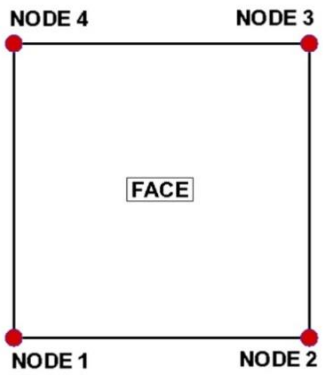


Figure 5.32: *Node assignment on each face on XZ_1, XZ_2, YZ_1, and YZ_2 planes*

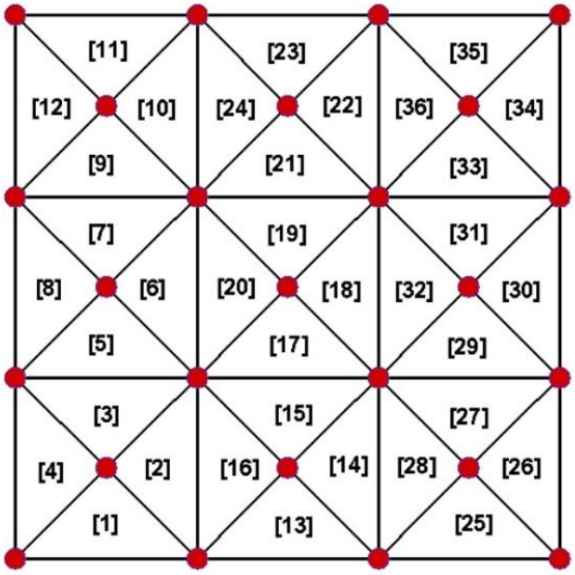


Figure 5.33: *Indexing of the boundary faces on XY_1 and XY_2 planes*

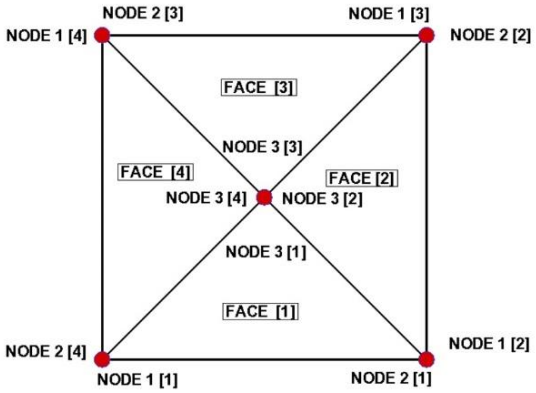


Figure 5.34: *Node assignment on each face on XY_1 and XY_2 planes*

5.3.3 Prismatic Grid of Type III

The purpose of this section is the study of the prismatic grids with an equilateral triangular base (Figures 5.6–5.7). The resulting grid is regarded as the outcome of a two-dimensional grid extrusion (composed from equilateral triangular elements) to the third dimension. A detailed presentation of the construction method is introduced.

The features needed for the construction of the grid are provided through an input file. Unlike the notions given in the previous algorithms, the content of this file lacks one value, concerning the number of edges in the *Y*-dimension for reasons which are going to be discussed below.

Table 5.11: *Input values of grid Type III and their corresponding variables*

INPUT VALUES	VARIABLES
Length of the Rectangle in the X – Direction	*X_L*
Length of the Rectangle in the Y – Direction	*Y_L*
Length of the Rectangle in the Z - Direction	*Z_L*
Number of Edges in the X – Direction	*NX*
Number of Edges in the Z – Direction	*NZ*

The initialization of the nodes is the primary step of the algorithm. An iterative procedure is implemented, where a two-dimensional grid of equilateral triangular elements is constructed in order to be replicated in the third dimension.

The construction of equilateral triangles requires the definition of the equilateral edge length; this is calculated as the value *dx* stemming from the division of *X_L* (the length of *X*-dimension), with the *NX* (the number of edges in *X*-dimension). Employing a trigonometric relation, the algorithm calculates the height of the equilateral triangle on the premises of the defined value *dx*, which corresponds to the length of *dy*. Consequently, the number of edges in the *Y*-dimension (*NY)* is inferred by the division of the defined *dy* with the length of *Y_L*. This justifies the missing notion of *NY* in the input file as mentioned earlier.

All of these calculated values determine the segmentation of the two-dimensional plane for the allocation of the nodes to take place, defining the vertices of the triangular elements which constitute the grid blueprint. An iterative procedure is implemented in order to initialize the coordinate nodes (*X(i), Y(i), Z(i)*) assigning the corresponding indexes, as depicted in Figure 5.35. Note that the edge triangles are not of equilateral shape, because of a constraint that prohibits the filling of quadrilateral space merely with equilateral triangles.

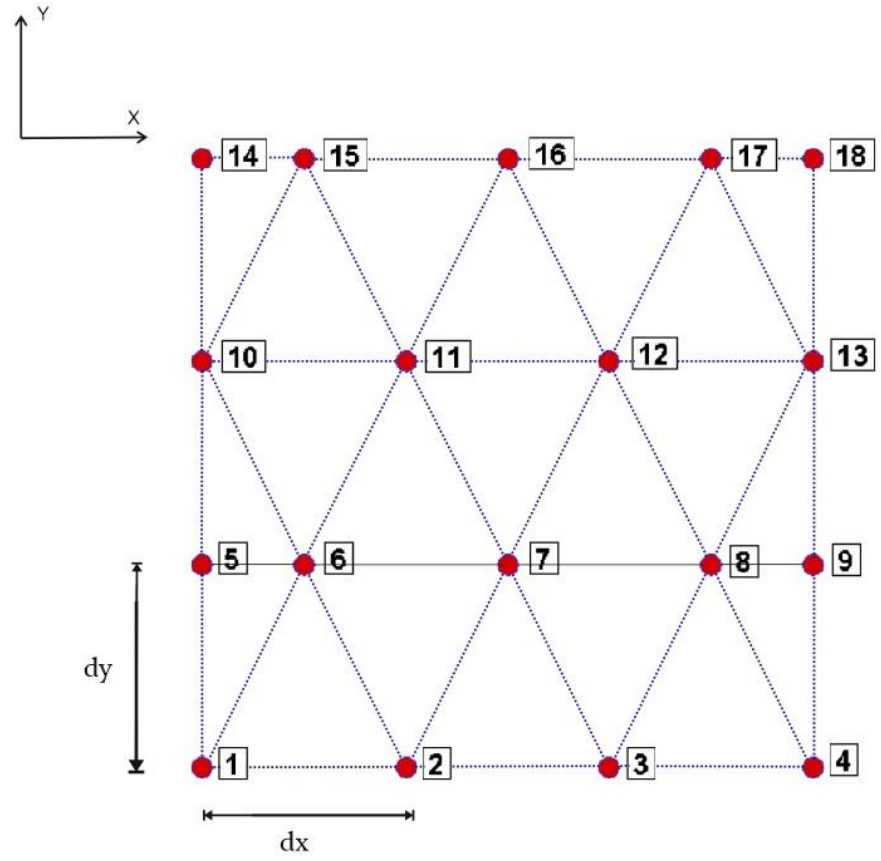


Figure 5.35: *Node initialization for the two-dimensional plane of the Prismatic Grid Type III*

Lastly, the constructed two-dimensional grid is reproduced towards the *Z*-dimension with an additional loop, according to the predefined values *NZ* and *Z_L*. The final outcome is illustrated in Figure 5.36.

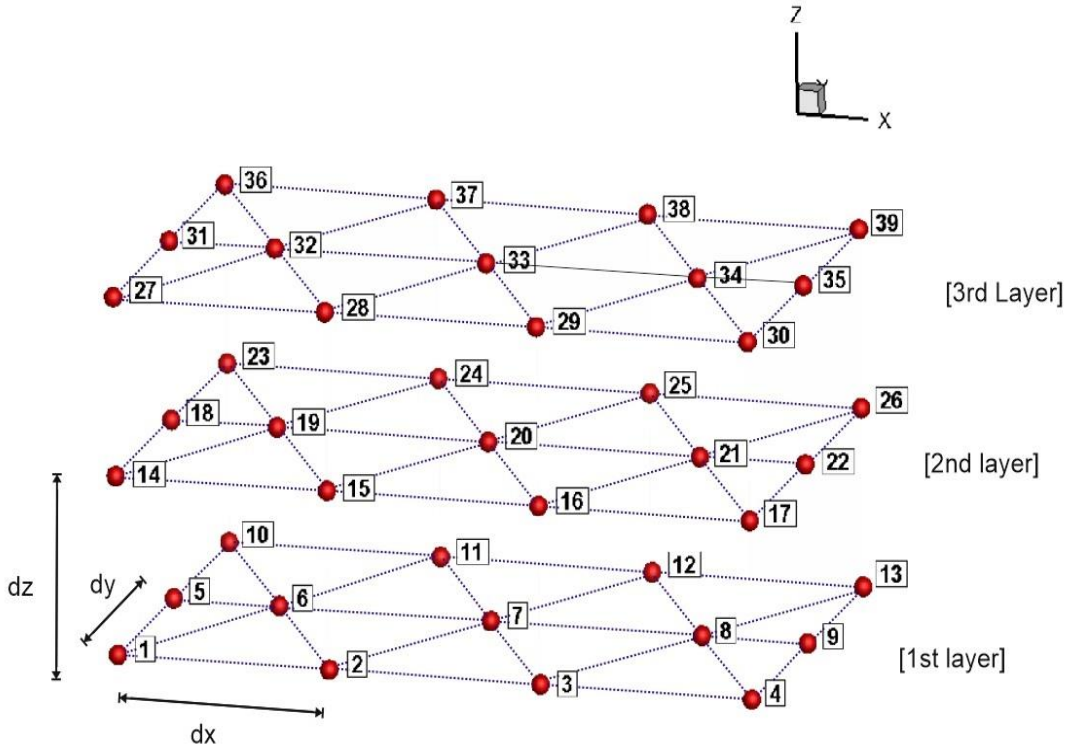


Figure 5.36: *Node orientation for the Prismatic Grid of Type III*

The algorithm proceeds to the definition of the prismatic elements. Prisms are derived by a direct node connectivity of the neighboring 2-D planes in the way exemplified in Figure 5.37. An iterative procedure is applied, scanning each triangular element on each 2-D plane and performing the appropriate connectivity of the nodes on the layer immediately above.

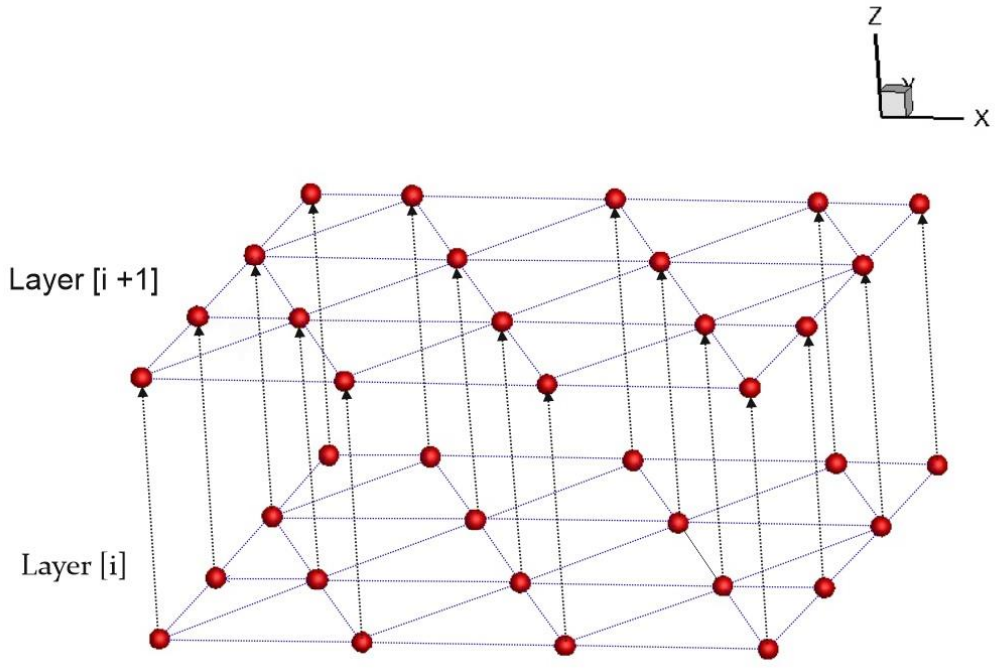


Figure 5.37: *Prism derivation*

The cell definition procedure follows a specific pattern: 4 rows of elements in the *X*-direction are determined by a set of 4 inner loops, each one of them represented with a different color in the scheme below.

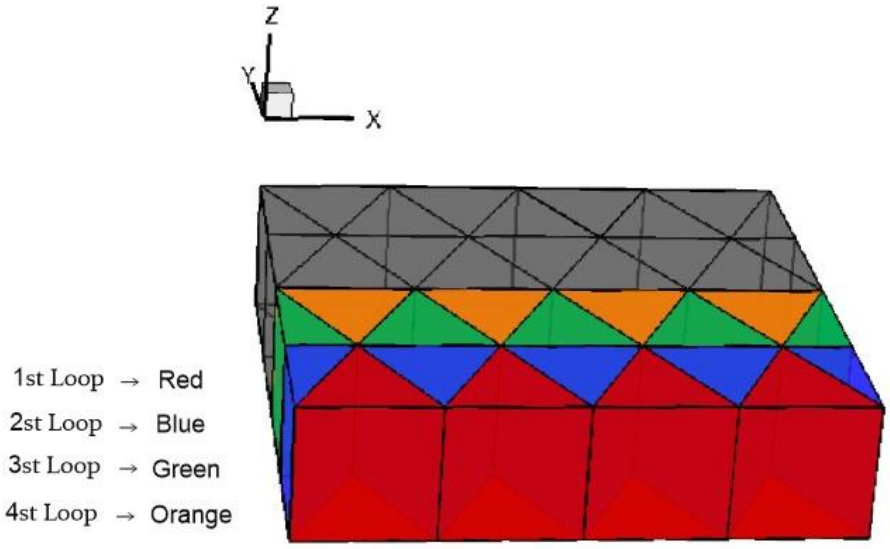


Figure 5.38: *Loops for cell definition in the X-direction*

An additional loop iterates the same pattern in the *Y*-direction determining all the prismatic elements of the first layer (Figure 5.39). The iterative procedure is executed as many times as dictated by the division of the *NY/2*. In case the *NY* equals with odd number, at the end of the iterative procedure under discussion an extra loop is performed, in order to determine the remaining two rows of elements.

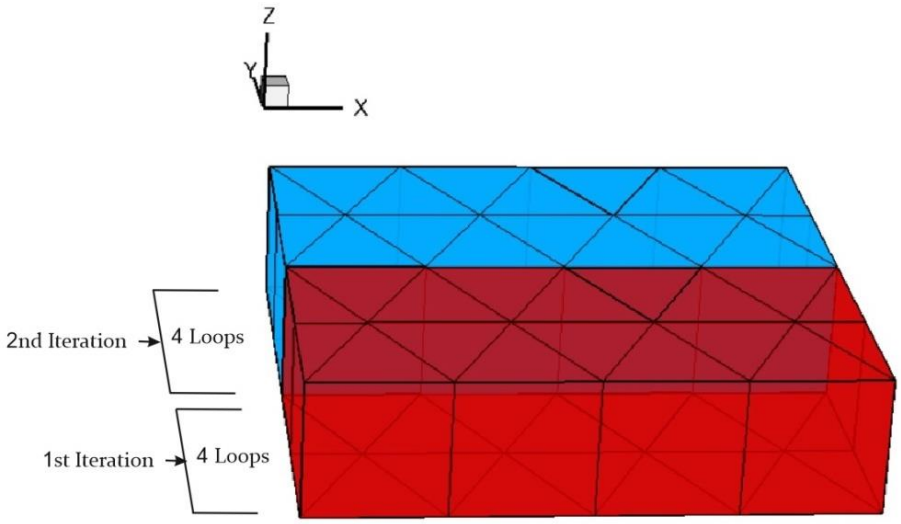


Figure 5.39: *Loop for cell definition in the Y-direction*

To complete the cell definition procedure, a loop in the *Z*-direction is implemented, which defines the elements on the succeeding layers above (Figure 5.40). The index cell numbering follows the flow of the entire iterative process.

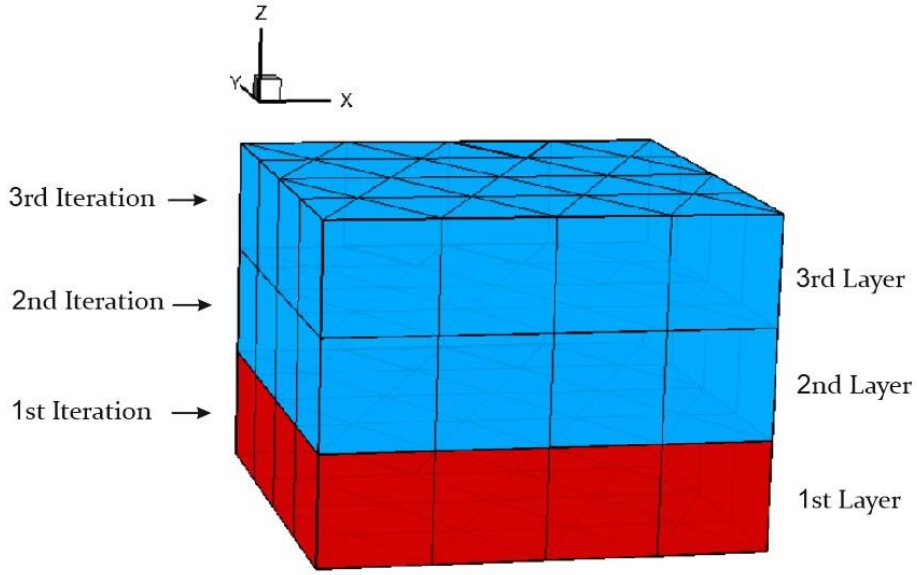


Figure 5.40: *Loop for cell definition in the Z-direction*

There are two distinct orientations which a prism might get in the grid. Figure 5.41 illustrates the auxiliary variables assignment of the two cases and Table 5.12 the node assignment to the prisms.

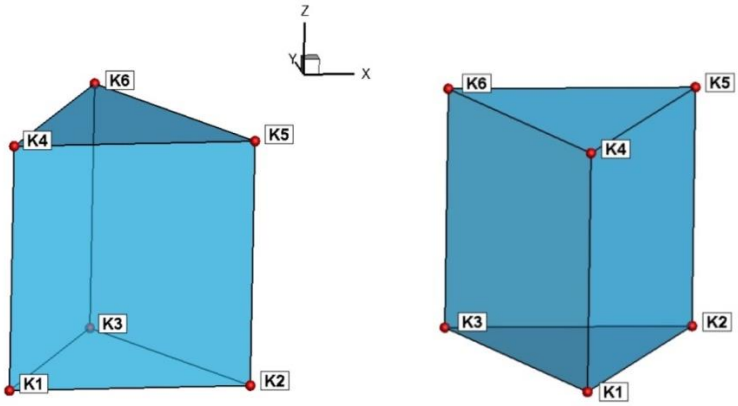


Figure 5.41: *Assignment of auxiliary variables based on the two alternative orientations on the grid*

Table 5.12: *Node assignment on the prisms of Prismatic Grid Type III*

DEFINITION	NODE (1)	NODE (2)	NODE (3)	NODE (4)	NODE (5)	NODE (6)
Prism [1]	*K1*	*K2*	*K3*	*K4*	*K5*	*K6*
Prism [2]	*K1*	*K2*	*K3*	*K4*	*K5*	*K6*

With reference to the determination of the faces and the neighboring cells for each prismatic element and the assignment to the *NF* and *NE* arrays respectively, they are both described in Tables 5.8 and 5.9.

The algorithm terminates with the definition of the 6 boundary planes (*NOD, NBFACE*, and *NBCELL*). An iterative procedure of two loops at the two dimensions define the YZ_1, YZ_2, XZ_1, XZ_2 planes having quadrilateral elements, starting from the first column and proceeding to the consecutive ones, as illustrated in Figures 5.42 – 5.43, while the triangular elements of the *XY_1* and the *XY_2* planes are determined by an iteration beginning from the first row and proceeding to the consecutive ones (Figures 5.44-5.45).

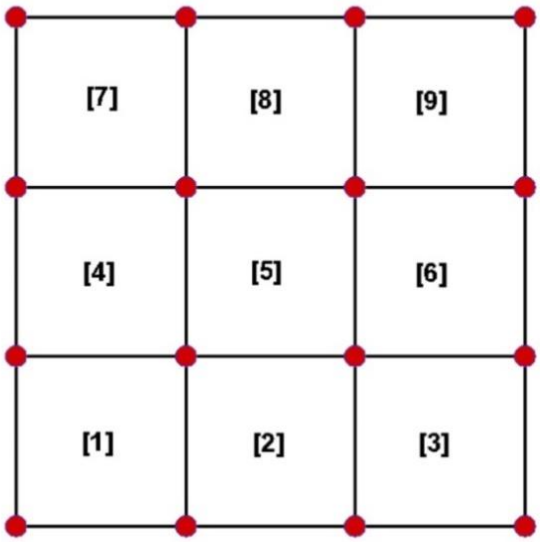


Figure 5.42: *Indexing of the boundary faces on XZ_1, XZ_2, YZ_1, and YZ_2 planes*

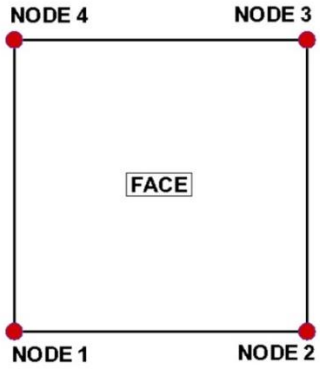


Figure 5.43: *Node assignment on each face on XZ_1, XZ_2, YZ_1, and YZ_2 planes*

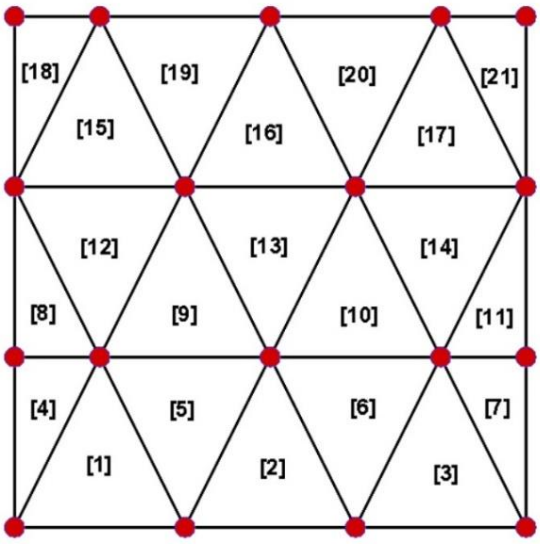


Figure 5.44: *Indexing of the boundary faces on XY_1 and XY_2 planes*

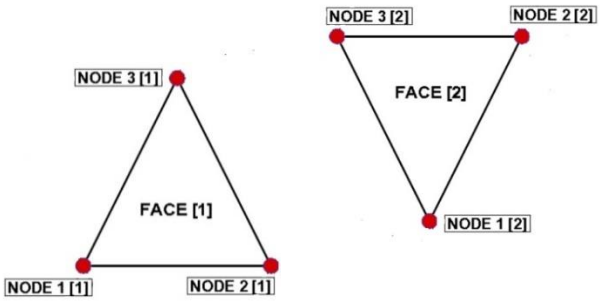


Figure 5.45: *Node assignment on each face on XY_1 and XY_2 planes*

5.3.4 Pyramidal Grid

In this section the development of the grid composed by pyramidal elements is presented (Figures 5.8-5.9). This type is derived from a regular Cartesian one, whose elements are divided into pyramids with the insertion of a middle node at its barycenter. The connectivity of this node with the 8 vertices decomposes the hexahedral element into 6 pyramids.

The initialization of the nodes, which constitute the blueprint of a Cartesian gird, has been already introduced in the previous subsections, where it was noted that the assignment of the node coordinates (*X (:), Y (:), Z (:)* array) occurs according to the import file (Figure 5.18, Table 5.6). A second iterative procedure determines the middle nodes at the barycenter of each hexahedral element, in an orientation identical with the one applied for the initialization of the rectangular nodes. The numbering of the middle nodes follows the arithmetic consecution of the nodes forming the regular grid. Figure 5.46 illustrates the final result for the nodes and the assigned indexes.

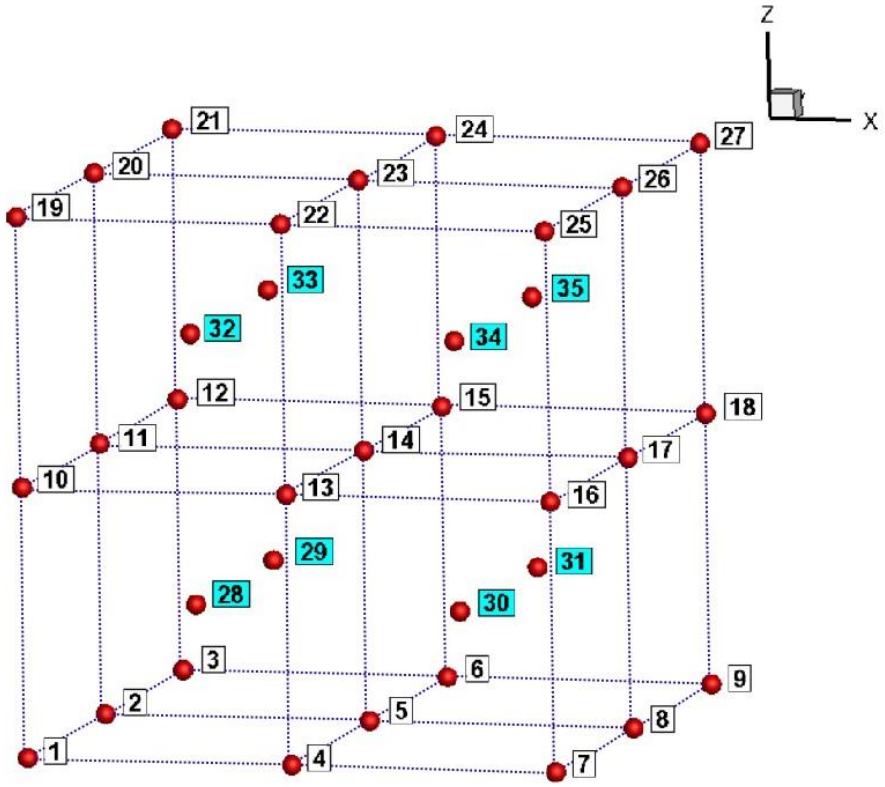


Figure 5.46: *Node orientation for the Pyramidal Grid*

Regarding the cell definition, an iterative procedure is implemented in order to decompose each hexahedral into pyramidal elements. The procedure is executed by a set of three loops, whose flow reflects the description in section 5.3.1, Figures 5.20-5.22.

The decomposition of the hexahedral elements requires that during the iterative process auxiliary variables be assigned to the 8 vertices of each hexahedron and to the additional middle node at the barycenter (Figure 5.47). Consequently, the nodes to be distributed to the *NC (i, j)* arrays form

6 distinct pyramidal elements. Table 5.13 shows the node distribution to the constructed pyramids.

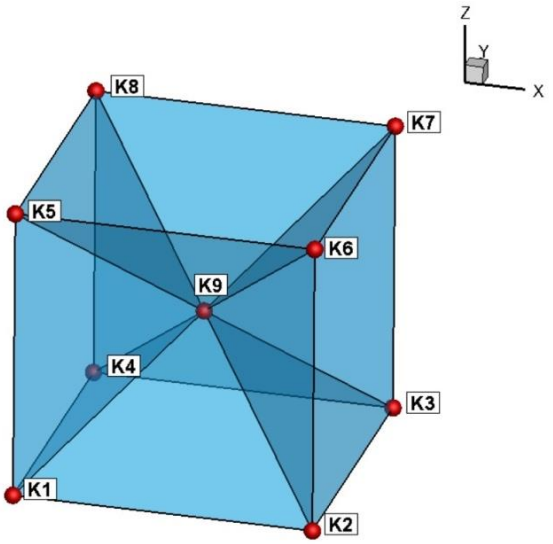


Figure 5.47: *Assignment of auxiliary variables on a hexahedral element*

Table 5.13: *Node assignment for the 6 pyramids*

CELL DEFITION	NODE (1)	NODE (2)	NODE (3)	NODE (4)	NODE (5)
Pyramid [1]	*K1*	*K2*	*K3*	*K4*	*K9*
Pyramid [2]	*K5*	*K6*	*K7*	*K8*	*K9*
Pyramid [3]	*K1*	*K2*	*K6*	*K5*	*K9*
Pyramid [4]	*K2*	*K3*	*K7*	*K6*	*K9*
Pyramid [5]	*K3*	*K4*	*K8*	*K7*	*K9*
Pyramid [6]	*K4*	*K1*	*K5*	*K8*	*K9*

Concerning the indexing of the faces stored in the *NF* array, it is directly determined by the node numbers it consists of (Table 5.14). Finally, 5 neighboring cells for each element need to be declared in an ascending numerical order and in relation to the number face adjacent to it, as indicated in Table 5.15.

Table 5.14: *Face indexing and node assignment on the NF array*

FACES	NODE ASSIGNMENT TO NF ARRAY			
	1st	2nd	3rd	4th
Face [1]	*Node (4)*	*Node (1)*	*Node (5)*	_
Face [2]	*Node (2)*	*Node (3)*	*Node (5)*	_
Face [3]	*Node (1)*	*Node (2)*	*Node (5)*	_
Face [4]	*Node (3)*	*Node (4)*	*Node (5)*	_
Face [5]	*Node (1)*	*Node (2)*	*Node (3)*	*Node (4)*

Table 5.15: *Neighboring cell indexing with reference to the corresponding face*

INDEXING NEIGHBORS	CORRESPONING FACE
Neighbor [1]	*Face [5]*
Neighbor [2]	*Face [3]*
Neighbor [3]	*Face [2]*
Neighbor [4]	*Face [4]*
Neighbor [5]	*Face [1]*

In order to determine the boundary conditions for the grid domain, two loops are required, so that the boundary faces on the 6 surfaces are defined, in other words the *NOD, NBCELL*, and *NBFACE* arrays. The orientation of the face indexing and the node assignment for the planes *XZ_1, XZ_2, YZ_1*, and *YZ_2* on the one hand and for the planes *XY_1* and *XY_2* on the other are presented in Figures 5.48-5.49 and Figures 5.50-5.51 respectively.

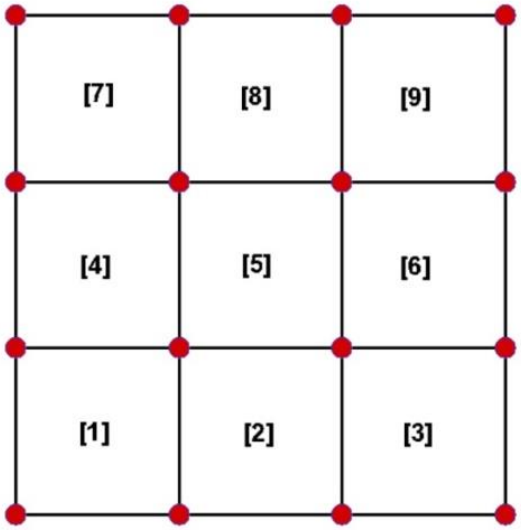


Figure 5.48: *Indexing of the boundary faces on XZ_1, XZ_2, YZ_1, and YZ_2 planes*

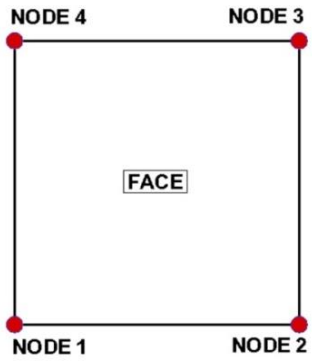


Figure 5.49: *Node assignment on each face on XZ_1, XZ_2, YZ_1, and YZ_2 planes*

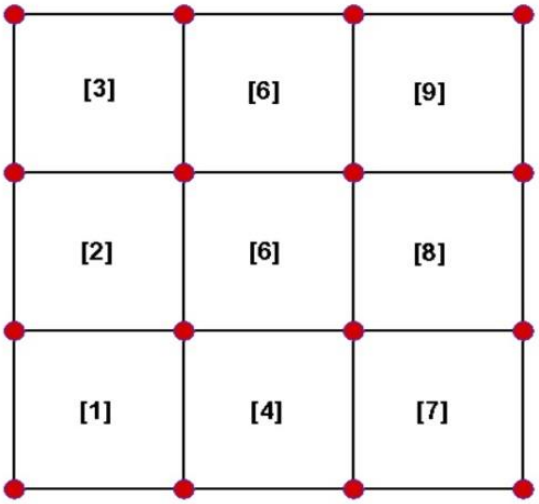


Figure 5.50: *Indexing of the boundary faces on XY_1 and XY_2 planes*

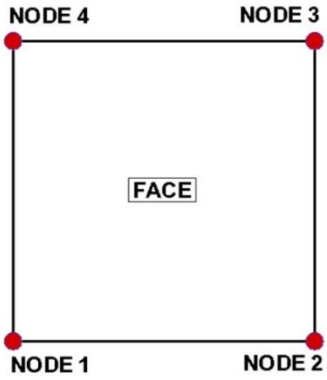


Figure 5.51: *Node assignment of each face on XY_1 and XY_2 planes*

5.3.5. Tetrahedral Grid

Two algorithms have been developed for the generation of tetrahedral elements (Figures 5.10-5.13). The former relies on the pyramidal grid algorithm, which further decomposes the generated elements into tetrahedrons. The latter is based on the algorithm generating equilateral prismatic grids; in this case, as well, the produced prisms are fragmented into tetrahedrons.

Type I

As it was mentioned above, the process heavily depends on the approach of the Pyramidal Grid; the node initialization is defined based on the input values (Table 5.6) creating the footprint of a regular Cartesian grid along with the middle nodes at the barycenter of the hexahedrons (Figure 5.46). The algorithm proceeds in scanning and decomposing the hexahedral elements with the same iterative procedure, which has already been discussed in section 5.3.1 (Figures 5.20-5.22).

Given the notion that tetrahedral elements are derived from the pyramidal ones with a diagonal splitting, which creates two tetrahedral cells for each pyramid (Figure 5.52), a further decomposition takes place at the cell definition procedure. In this case, the 6 pyramids of each hexahedron are decomposed into 12 tetrahedrons. The dummy variables (*K1, K2, K3… K9*) are assigned at the vertices of each hexahedral elements and their barycenter (Figure 5.53) and allocated to the corresponding *NC* array (Table 5.16).

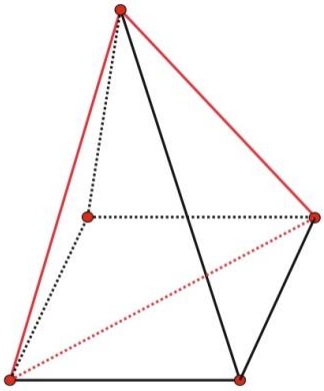


Figure 5.52: *Pyramid splitting into two tetrahedrons*

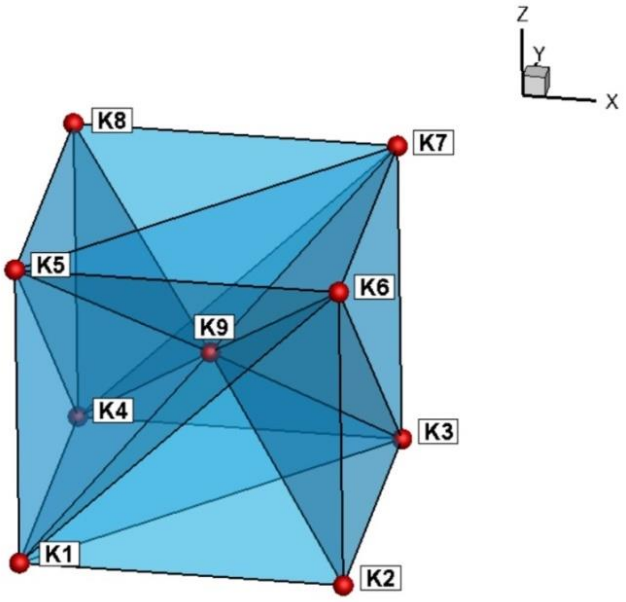


Figure 5.53: *Assignment of auxiliary variables on a hexahedral element*

Table 5.16: *Node assignment for the 12 tetrahedrons*

DEFINITION	NODE [1]	NODE [2]	NODE [3]	NODE [4]
Tetrahedron [1]	*K1*	*K2*	*K3*	*K9*
Tetrahedron [2]	*K1*	*K3*	*K4*	*K9*
Tetrahedron [3]	*K5*	*K6*	*K7*	*K9*
Tetrahedron [4]	*K5*	*K7*	*K8*	*K9*
Tetrahedron [5]	*K1*	*K2*	*K6*	*K9*
Tetrahedron [6]	*K1*	*K6*	*K5*	*K9*
Tetrahedron [7]	*K3*	*K6*	*K2*	*K9*
Tetrahedron [8]	*K3*	*K7*	*K6*	*K9*
Tetrahedron [9]	*K4*	*K7*	*K3*	*K9*
Tetrahedron [10]	*K4*	*K8*	*K7*	*K9*
Tetrahedron [11]	*K4*	*K1*	*K5*	*K9*
Tetrahedron [12]	*K4*	*K5*	*K8*	*K9*

Concerning the 4 faces of the tetrahedral element, they need to be stored in *NF* array; the index numbering of each face corresponds to the nodes indicated in Table 5.17. Moreover, the neighboring cells are defined accordingly, with each face corresponding to the indexing of its neighbor (Table 5.18).

Table 5.17: *Face indexing and node assignment in NF array*

FACES	NODE ASSIGNMENT TO NF ARRAY		
	1st	**2nd**	**3rd**
Face [1]	*Node (1)*	*Node (2)*	*Node (3)*
Face [2]	*Node (1)*	*Node (2)*	*Node (4)*
Face [3]	*Node (2)*	*Node (3)*	*Node (4)*
Face [4]	*Node (1)*	*Node (3)*	*Node (4)*

Table 5.18: *Neighboring cell indexing with reference to the corresponding face*

INDEXING NEIGHBORS	CORRESPONING FACE
Neighbor [1]	*Face [1]*
Neighbor [2]	*Face [2]*
Neighbor [3]	*Face [3]*
Neighbor [4]	*Face [4]*

All the boundary planes are composed from triangular elements and according to the perspective view taken to each plane. Figures 5.54–5.61 demonstrate the face indexing and the node orientation of each boundary face.

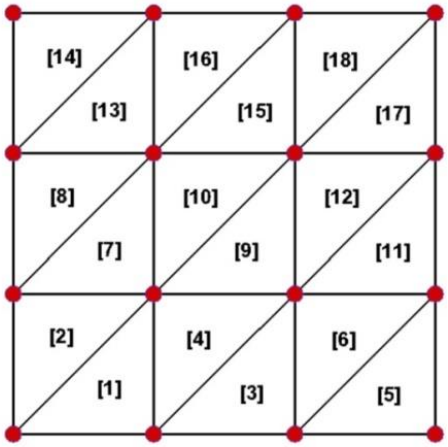


Figure 5.54: *Indexing of the boundary faces on XZ_1 and YZ_1 planes*

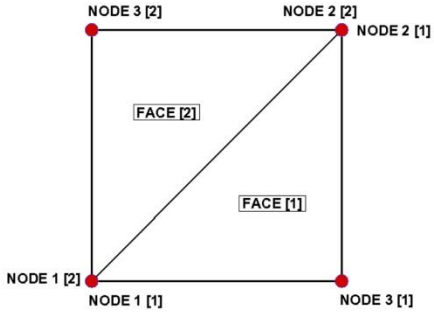


Figure 5.55: *Node assignment on each face on XZ_1 and YZ_1 planes*

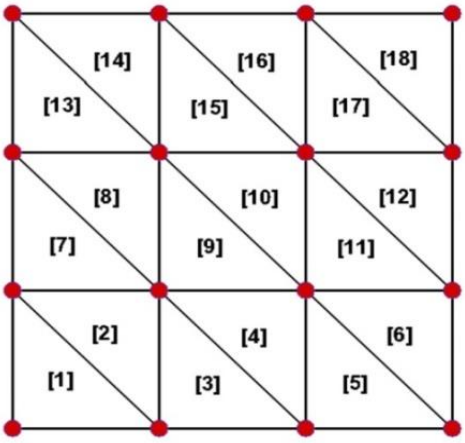


Figure 5.56: *Indexing of the boundary faces on XZ_2, YZ_2 planes*

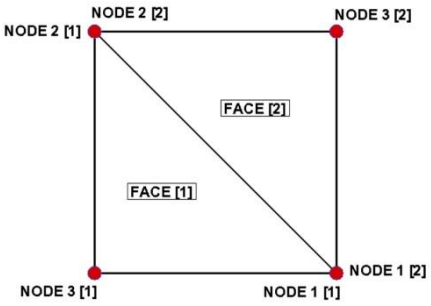


Figure 5.57: *Node assignment of each face on XZ_2, YZ_2 planes*

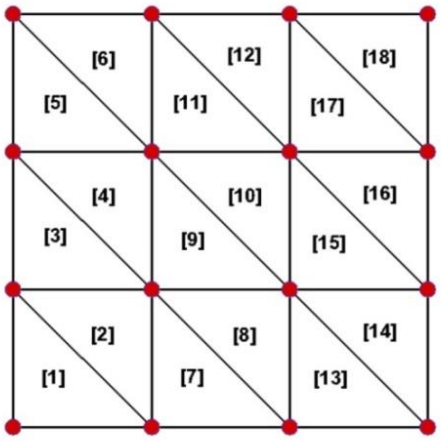


Figure 5.58: *Indexing of the boundary faces on XY_1 plane*

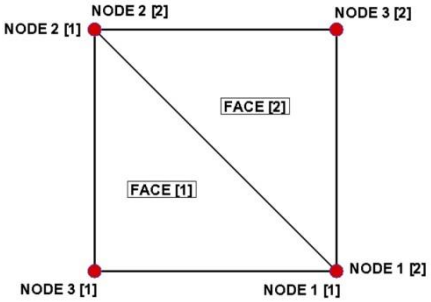


Figure 5.59: *Node assignment of each face on XY_1 plane*

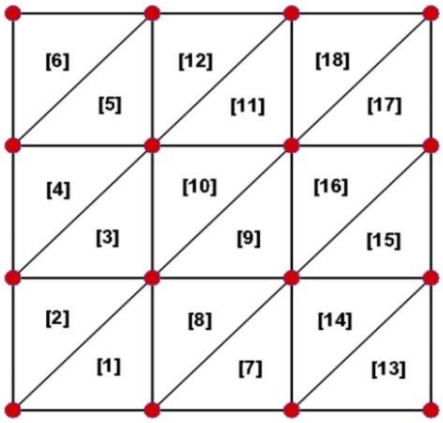


Figure 5.60: *Indexing of the boundary faces on XY_2 plane*

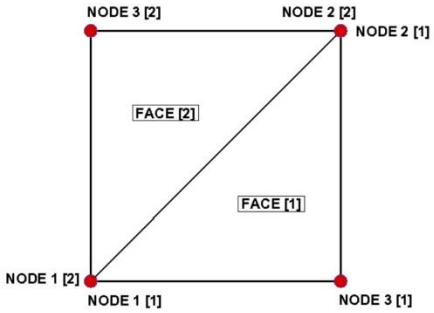


Figure 5.61: *Node assignment of each face on XY_2 plane*

Type II

With respect to the generation of tetrahedrons of type II, the algorithmic processing of the prismatic elements of Type III is extended on the basis of the prisms' decomposition into tetrahedrons. This is performed with a node insertion at the barycenter of each prism. Its connectivity with the vertices of the prism gives rise to two tetrahedrons and 4 pyramids. A further decomposition of the pyramids with a diagonal splitting completes the creation of 8 tetrahedral elements (Figure 5.62).

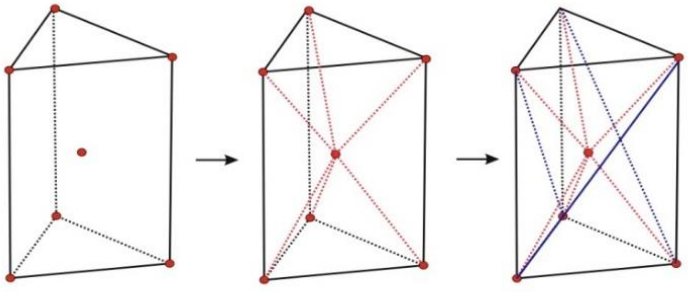
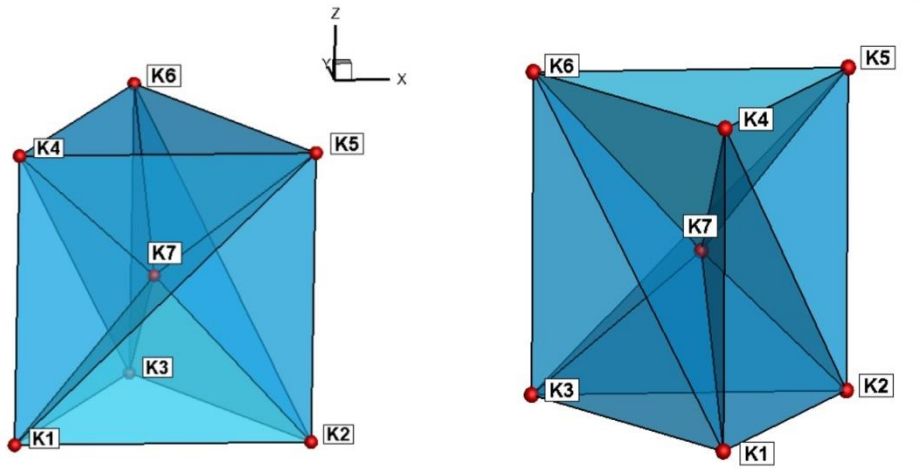


Figure 5.62: *Decomposition of a prism into 8 tetrahedrons*

Again, an input file defines the features of the grid (Table 5.11) and the node initialization occurs (Figure 5.36). At this point an additional loop defines the middle nodes for each prism. In what follows, the same iterative procedure defines the elements in the manner discussed in section 5.3.3, Figures 5.38-5.40. The dummy variables assignment for the two different orientations the grid might receive is shown in Figure 5.63. For each prism 8 distinct tetrahedrons are created through the allocation to the corresponding *NC* arrays. Tables 5.19 and 5.20 indicate these definitions. *NF* and *NE* arrays, referring to the faces and the neighboring cells, are determined according to Tables 5.17 and 5.18 respectively.



Prism [1] Prism [2]

Figure 5.63: *Assignment of auxiliary variables, based on the two alternative orientations of the grid*

Table 5.19: *Node assignment on the 8 tetrahedrons of each prism [1]*

DEFINITION	NODE [1]	NODE [2]	NODE [3]	NODE [4]
Tetrahedron [1]	*K7*	*K1*	*K2*	*K5*
Tetrahedron [2]	*K7*	*K1*	*K5*	*K4*
Tetrahedron [3]	*K7*	*K2*	*K3*	*K6*
Tetrahedron [4]	*K7*	*K2*	*K6*	*K5*
Tetrahedron [5]	*K7*	*K3*	*K1*	*K4*
Tetrahedron [6]	*K7*	*K3*	*K4*	*K6*
Tetrahedron [7]	*K7*	*K1*	*K2*	*K3*
Tetrahedron [8]	*K7*	*K4*	*K5*	*K6*

Table 5.20: *Node assignment on the 8 tetrahedrons of each prism [2]*

DEFINITION	NODE [1]	NODE [2]	NODE [3]	NODE [4]
Tetrahedron [1]	*K7*	*K1*	*K2*	*K4*
Tetrahedron [2]	*K7*	*K2*	*K5*	*K4*
Tetrahedron [3]	*K7*	*K2*	*K3*	*K5*
Tetrahedron [4]	*K7*	*K3*	*K6*	*K5*
Tetrahedron [5]	*K7*	*K3*	*K1*	*K6*
Tetrahedron [6]	*K7*	*K1*	*K4*	*K6*
Tetrahedron [7]	*K7*	*K1*	*K2*	*K3*
Tetrahedron [8]	*K7*	*K4*	*K5*	*K6*

Taking under consideration the different perspective views on the boundary faces, *XZ_1, XZ_2, YZ_1* and *YZ_2* planes are composed of triangular elements, as depicted in Figures 5.64-5.65, 5.66-5.67 and 5.68-5.69, respectively, while for the planes *XY_1* and *XY_2* the equilateral triangles, of which they consist, have the same orientation as illustrated in Figures 5.70-5.71.

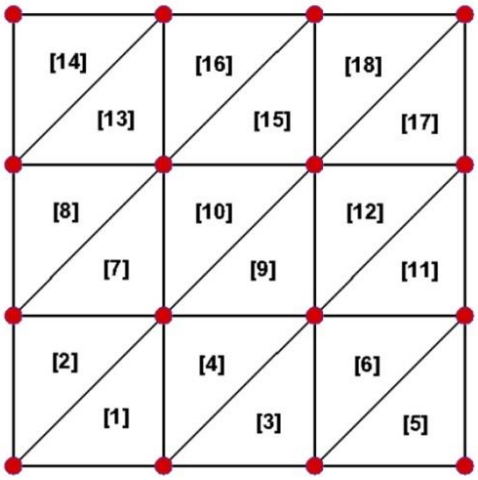


Figure 5.64: *Indexing of the boundary faces on the XZ_1 plane*

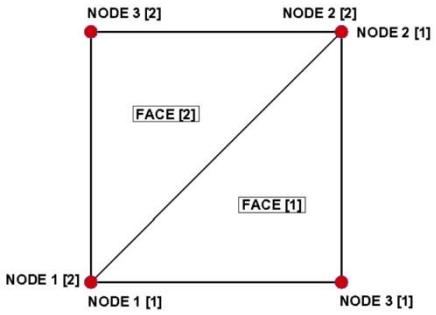


Figure 5.65: *Node assignment of each face on the XZ_1 plane*

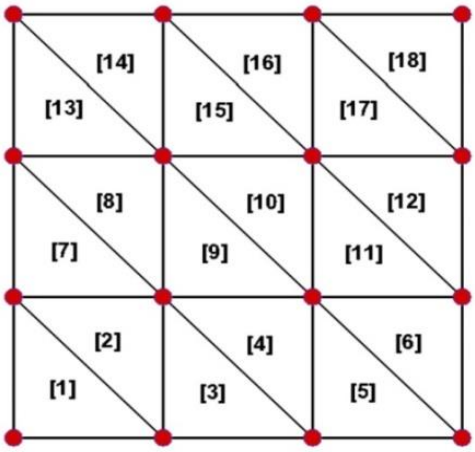


Figure 5.66: *Indexing of the boundary faces on XZ_2 plane*

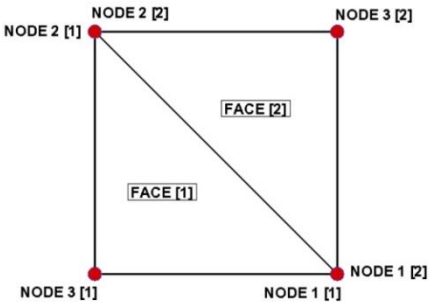


Figure 5.67: *Node assignment of each face on XZ_2 plane*

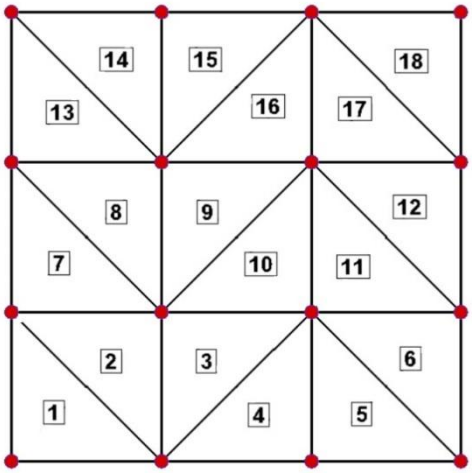


Figure 5.68: *Indexing of the boundary faces on YZ_1 and YZ_2 planes*

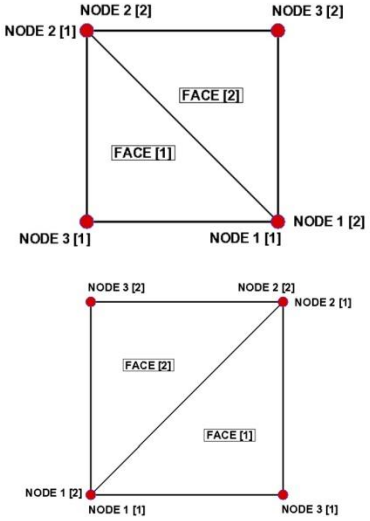


Figure 5.69: *Node assignment of each face on YZ_1 and YZ_2 planes*

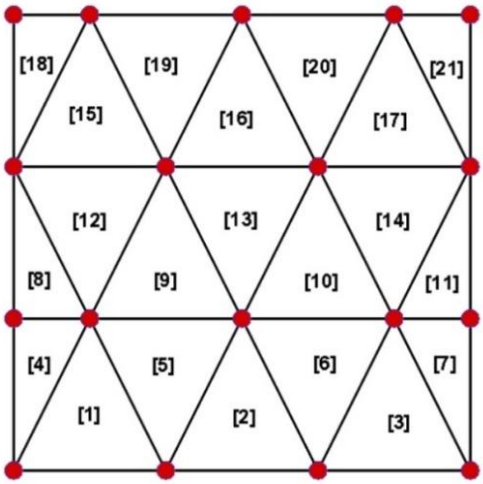


Figure 5.70: *Indexing of the boundary faces on XY_1 and XY_2 planes*

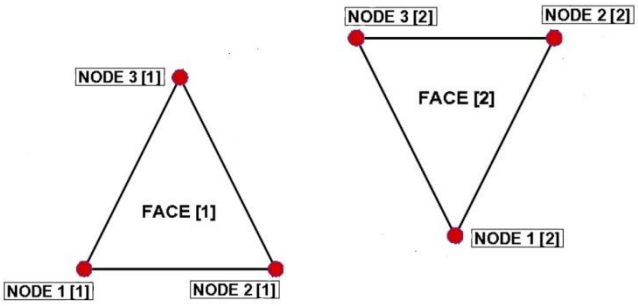


Figure 5.71: *Node assignment of each face on XY_1 and XY_2 planes*

5.4 Irregular Grids

Grid irregularities are introduced by perturbing the grid nodes from their original positions with random shifts. All the aforementioned grid types are developed so as to produce Irregular Girds with the addition of a simple subroutine where the distortion of the grid nodes occurs.

An iterative procedure assigns indexes on the nodes, differentiating the boundary nodes from the internal ones. This happens in order for the boundary nodes to remain intact. As a result, the computational domain retains its form. Subsequently, a recursive process disturbs the original coordinates of the internal nodes. This perturbation takes place randomly in each dimension and is defined as $0.4r\Delta x$, where $r \in [-1/2, 1/2]$ is a random number and Δx is the local mesh size along the given dimension.

The subroutine is outlined in the pseudo-code below.

Subroutine Distortion

```
//For the boundary nodes//
Do i =1, NNODE
  IF (node is a boundary) THEN
    Set index to the node
  END
ENDDO
//Node perturbation//
Do i =1, NNODE
  IF (node is not a boundary) THEN
    Distort the node
  END
ENDDO
```

For each of the regular grid types, its corresponding irregular version is shown in the following Figures (5.72-5.77).

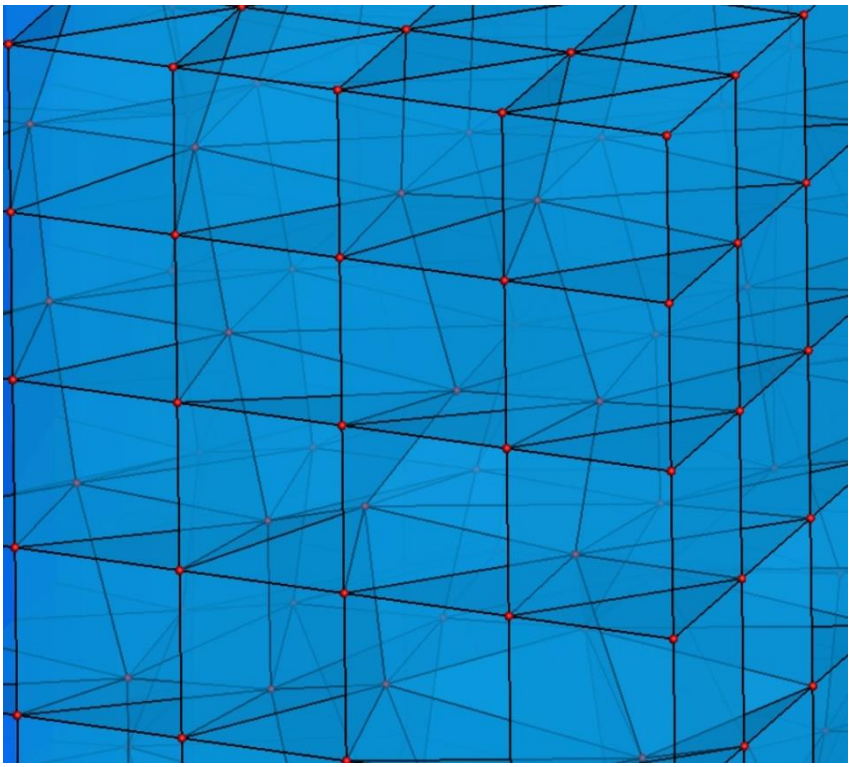


Figure 5.72: *Distorted Prismatic Grid of Type I*

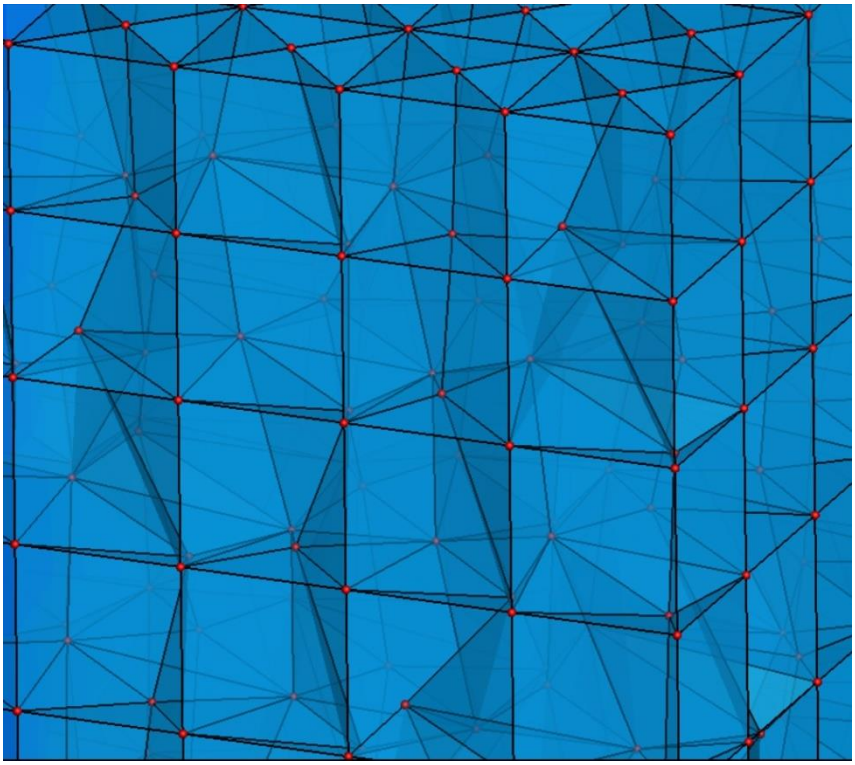


Figure 5.73: *Distorted Prismatic Grid of Type II*

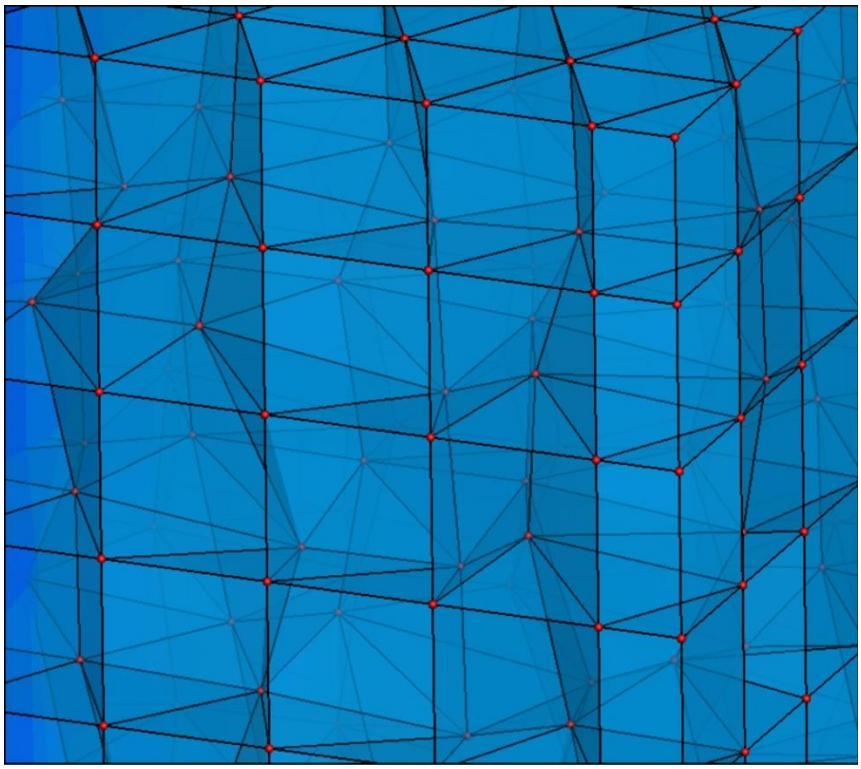


Figure 5.74: *Distorted Prismatic Grid of Type III*

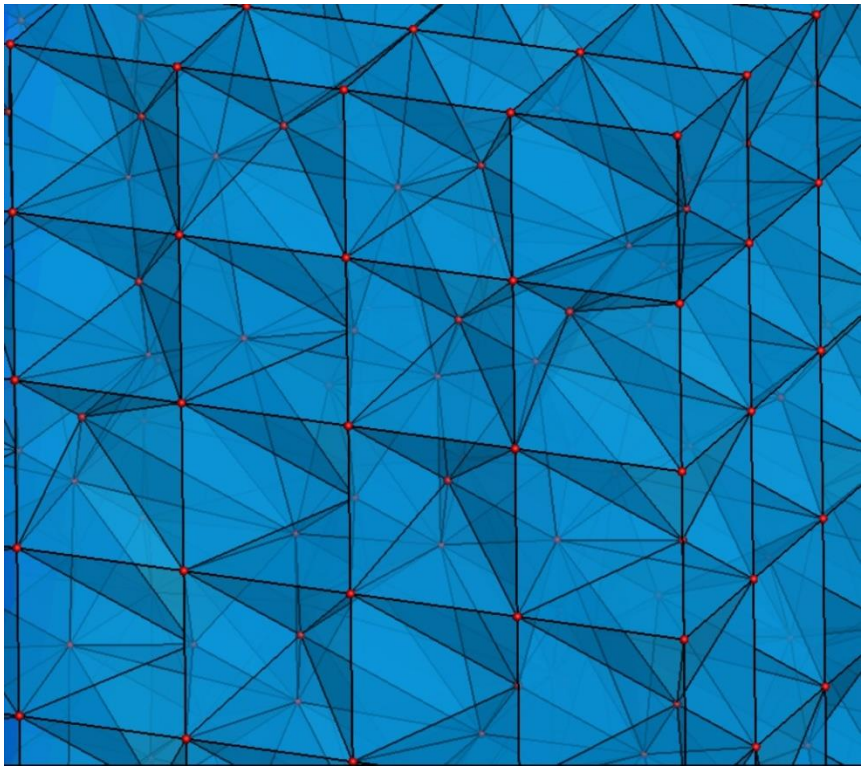


Figure 5.75: *Distorted Pyramidal Grid*

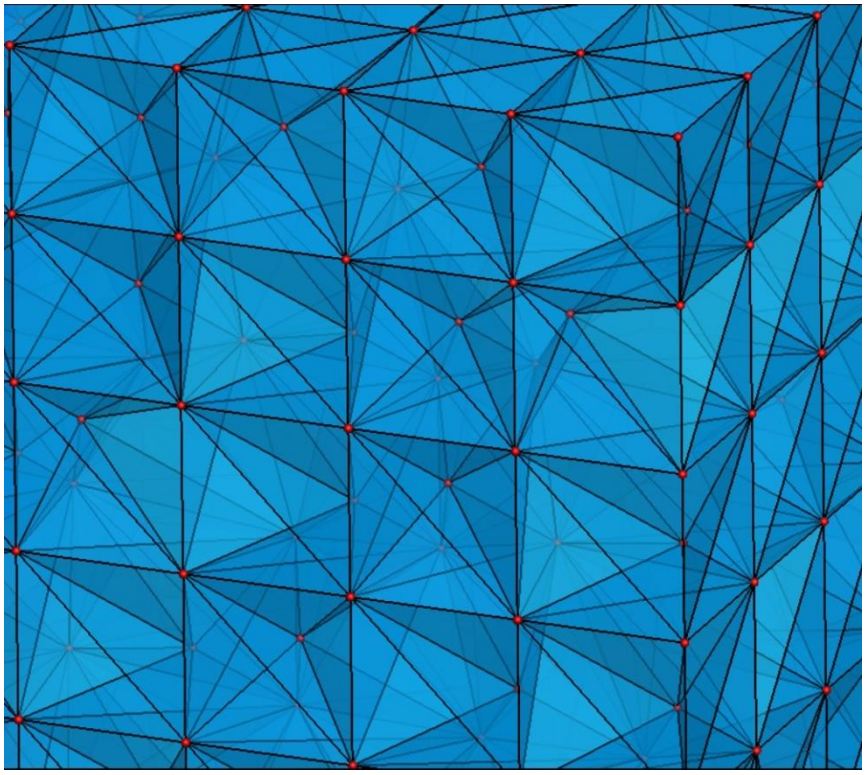


Figure 5.76: *Distorted Tetrahedral Grid of Type I*

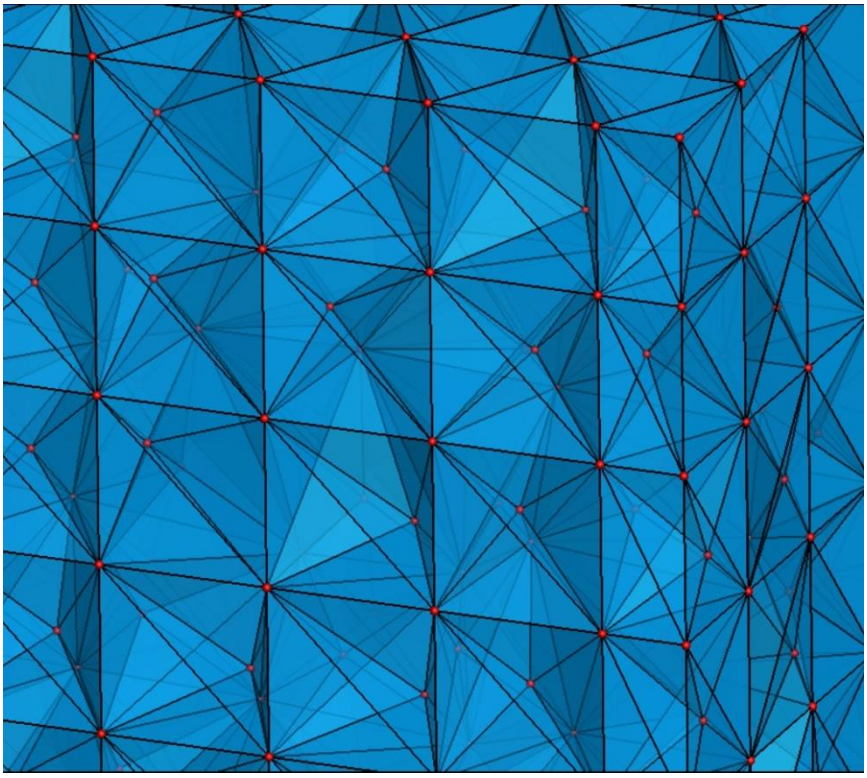


Figure 5.77: *Distorted Tetrahedral Grid of Type II*

"Intentionally left blank"

CONCLUSIONS AND FUTURE WORK

In the current study, a high-order numerical scheme was integrated into an existing academic solver (*EU2*) for the numerical solution of 2-dimensional Euler equations. The discretized governing equations are solved with a Finite Volume Node-centered scheme on unstructured triangular grids, while an upwind method is implemented for the computation of the inviscid fluxes, employing the Roe's approximate Riemann. A successive differentiation technique is utilized to achieve up to third order spatial accuracy incorporating the high-order correction terms to the reconstructed nodal values. The aforementioned formulation is based on a variable-extrapolation, the U-MUSCL type reconstruction, which closely resembles the traditional MUSCL one. Additionally, a Strong Stability Preserving (SSP) fourth order - five stage Runge-Kutta method is used for time discretization.

The benchmark problem of an isentropic travelling vortex with a well-known analytical solution is utilized for the evaluation of the numerical accuracy. It was chosen to examine the behavior of the numerical scheme under certain conditions and determine its effectiveness to the numerical solution. Emphasis has been placed on grid convergence study, to define the order of the numerical accuracy of the scheme. Using a controlled environment through a successive grid refinement procedure, numerical simulations have been carried out on different type of triangular grids and in two different time periods. The following major conclusions can be drawn from the present work:

-The presented high-order scheme enhances the numerical accuracy of the existing solver. In terms of the convergence behavior, the numerical results for regular grids obtain a satisfactory agreement. The convergences histories of Equilateral (Type I) and Orthogonal (Type II) grids exhibit an identical behavior for all conservatives' variables and for all different norms and achieve a third order accuracy, which, in some cases, was surpassed. As expected, orthogonal grid (Type III) and especially the distorted grid have shown an order reduction on the convergence rates, compared to the previous ones. The employment of Green-Gauss formula may not give the expected high-order accuracy on general unstructured grids but it certainly improves the accuracy of a base second-order scheme.

-The experience of integrating the current numerical scheme into the existing academic solver shows an easy implementation without excessive efforts. Through the successive differentiation process, the computation of the high-order terms is feasible by exploiting the existing structure of the code with only slight modifications. The same routine that computes the field of first derivatives is utilized to obtain the field of the higher derivatives. Since there is no need to introduce additional DoFs (Degreed of Freedom) to the reconstructed values, this advantage seems quite preferable for parallel unstructured flow solvers where the employment of the common high-order schemes is rather challenging.

- The memory requirements of the incorporated high-order module and the computation time of the numerical solution are kept low. The computation of the high order terms requires only the allocation of 3 additional arrays in the case of a 3rd order scheme, while the computational effort for the flux calculation in each iteration has only a low overhead, associated with the calculation of the above terms.

Following from this work, the derivation of the corresponding high-order formulation is already introduced into a 3-dimensional numerical flow model. On-going development of high-order accuracy to the 3-D CFD solver *Galatea* is being carried out. As already mentioned in Chapter 4, *Galatea* is a parallelized node-centered finite volume solver for the numerical solution of compressible fluid flows in hybrid unstructured grids. The developed 3-D computational meshes demonstrated in chapter 5 will be used for an extensive evaluation of the numerical results similar to the methodology that has been employed in this study.

Finally, the implementation of the high-order module was conducted without taking into account quadrature rules in the computation of the numerical fluxes. Hence, it was restricted to the existing midpoint rule for the calculation of the flux integrals, albeit providing a significant improvement to the numerical solutions. A future development using additional quadrature points could hypothetically increase the order of the numerical scheme, along, however, with the analogous computational cost. Further research could be undertaken to explore the potential improvements on the numerical performance of the aforementioned procedures.

REFERENCES

[Ada05] Λ.Δ. Αδαμούδης, Επίλυση των Εξισώσεων Euler σε Τρεις Διαστάσεις με Χρήση μη Δομημένου Πλέγματος και Εφαρμογή της Μεθόδου των Πεπερασμένων Ογκων, Διπλωματική Εργασία, *Πολυτεχνείο Κρήτης*, 2005.

[And94] W.K. Anderson and D.L. Bonhaus, An Implicit Algorithm for Computing Turbulent Flows on Unstructured Grids, *Computers & Fluids*, vol. 23, pp. 1-21, 1994.

[ANSYS06] ANSYS CFX-Solver Theory Guide, ANSYS CFX Release 11.0, December 2006.

[Bar89] T.J. Barth and D.C. Jespersen, The Design and Application of Upwind Schemes on Unstructured Meshes, *Proceedings of the 27th Aerospace Sciences Meeting and Exhibit, AIAA*, Reno, NV, Jan 9-12 1989, pp. 1-12, AIAA-89-0366.

[Bar92] T.J. Barth, Aspects of Unstructured Grids and Finite-Volume Solvers for the Euler and Navier-Stokes Equations, *Proceedings of the AGARD-FDP-VKI special course at VKI*, Rhode-Saint-Genese 2-6 March 1992, AGARD-R-787, pp. 6.1-6.61.

[Bar93] T.J. Barth, Recent Developments in High Order K-Exact Reconstruction on Unstructured Meshes, *31st AAIA Aerospace Sciences Meeting & Exhibit*, January 11-14, Reno NV, AAIA 93-0068.

[Bla01] J. Blazek, Computational Fluid Dynamics: Principles and Applications, *Kidlington: Elsevier Science*, 2001.

[Bur05] C.O.E. Burg, Higher Order Variable Extrapolation for Unstructured Finite Volume RANS Flow Solvers, *17th AIAA Computational Fluid Dynamics Conference*, Toronto, Ontario Canada, AIAA 2005-4999.

[Dis07] B. Diskin, J. Thomas, Accuracy Analysis for Mixed-Element Finite-Volume Discretization Schemes, *National Institute of Aerospace,* Rept. 2007-08.

[Dis10] B. Diskin, J.L. Thomas, E.J. Nielsen, H. Nishikawa and J.A. White, Comparison of Node-Centered and Cell-Centered Unstructured Finite-Volume Discretizations: Viscous Fluxes, *AIAA Journal*, vol. 48, pp. 1326-1338, 2010.

[Dis11] B. Diskin, T.J. Thomas, Comparison of Node-Centered and Cell-Centered Unstructured Finite-Volume Discretizations: Inviscid Fluxes, *American Institute of Aeronautics and Astronautics Journal*, AIAA vol. 49, pp. 836-854, 2011.

[Del11] A.I. Delis, I.K. Nikolos and M. Kazolea, Performance and Comparison of Cell-Centered and Node-Centered Unstructured Finite Volume Discretizations for Shallow Water Free Surface Flows, *Archives of Computational Methods Engineering,* vol. 58, pp. 57-118, 2011.

[Del13] A.I. Delis and I.K. Nikolos, A Novel Multidimensional Solution Reconstruction and Edge-Based Limiting Procedure for Unstructured Cell-Centered Finite Volumes with Application to

Shallow Water Dynamics, *International Journal for Numerical Methods in Fluids*, vol. 71, pp. 584-633, 2013.

[Got05] S. Gottlieb, On High Order Strong Stability Preserving Runge–Kutta and Multi Step Time Discretizations, *Journal of Scientific Computing,* vol. 25, pp. 105-128, 2005.

[Hir90] C. Hirsch, Numerical Computation of Internal and External Flows. Vol. 2: Computational Methods for Inviscid and Viscous Flows, *John Wiley and Sons*, New York, 1990.

[Kal96] Y. Kallinderis, A 3-D Finite Volume Method for the Navier Stokes Equations with Adaptive Hybrid Grids, *Applied Numerical Mathematics*, vol. 20, pp. 387-406, 1996.

[Kal05] Y. Kallinderis and H.T. Ahn, Incompressible Navier-Stokes Method with General Hybrid Meshes, *Journal of Computational Physics*, vol. 210, pp. 75-108, 2005.

[Kim03] K. Kim, Three-Dimensional Hybrid Grid Generator and Unstructured Flow Solver for Compressors and Turbines, PhD thesis, *Texas A&M University*, 2003.

[Koo00] B. Koobus, C. Farhat and H. Tran, Computation of Unsteady Viscous Flows around Moving Bodies Using the k-ε Turbulence Model on Unstructured Dynamic Grids, *Computer Methods in Applied Mechanics and Engineering*, vol. 190, pp. 1441-1466, 2000.

[Kou03] D.G. Koubogiannis, A.N. Athanasiadis and K.C. Giannakoglou, One- and Two-Equation Turbulence Models for the Prediction of Complex Cascade Flows Using Unstructured Grids, *Computer and Fluids*, vol. 32, pp. 403-430, 2003.

[Lal88] M.H. Lallemand, Etude de Schemas Runge-Kutta a 4 pas pour la Resolution Multigrille des Equations d' Euler 2D, *Raport de Recherche*, INRIA, 1988.

[Lan98] C.B. Laney, Computational Gasdynamics, *Cambridge University Press*, 1998.

[Luo05] H. Luo, J.D. Baum and R. Lohner, High-Reynolds Number Viscous Flow Computations Using an Unstructured-Grid Method, *Journal of Aircraft*, vol. 42, pp. 483-492, 2005.

[Lyg11] G.N. Lygidakis and I.K. Nikolos, An Unstructured Node-Centered Finite Volume Method for Computing 3D Viscous Compressible Flows on Hybrid Grids, *Proceedings of the 7th GRACM (Greek Association of Computational Mechanics) International Congress on Computational Mechanics,* Athens, 30 June - 2 July 2011.

[Lyg12] G.N. Lygidakis and I.K. Nikolos, Using the Finite-Volume Method and Hybrid Unstructured Meshes to Compute Radiative Heat Transfer in 3-D Geometries, *Numerical Heat Transfer Part B: Fundamentals*, vol. 62, pp. 289-314, 2012.

[Lyg13] G.N. Lygidakis and I.K. Nikolos, Using a High-Order Spatial/Temporal Scheme and Grid Adaptation with a Finite-Volume Method for Radiative Heat Transfer, *Numerical Heat Transfer Part B: Fundamentals*, vol. 64, pp. 89-117, 2013.

[Lyg14a] G.N. Lygidakis and I.K. Nikolos, Assessment of the Academic CFD Code "Galatea" Using the NASA Common Research Model (CRM), *Proceedings of the 12th Biennial Conference on Engineering Systems Design and Analysis, ASME-ESDA2014*, Copenhagen, Denmark, 25-27 June 2014, ESDA 2014-20265.

[Lyg14b] G.N. Lygidakis and I.K. Nikolos, Using the DLR-F6 Aircraft Model for the Evaluation of the Academic CFD Code "Galatea", *Proceedings of the International Mechanical Engineering Congress and Exposition, ASME-IMECE2014,* Montreal, Canada, 14-20 November 2014, IMECE 2014-39756.

[Lyg15] G.N. Lygidakis, On the Numerical Solution of Compressible Fluid Flow and Radiative Heat Transfer Problems, Ph.D. Thesis, *Technical University of Crete*, 2015.

[Mav94] D.J. Mavriplis and L. Martinelli, Multigrid Solution of Compressible Turbulent Flow on Unstructured Meshes Using a Two-Equation Model, *International Journal for Numerical Methods in Fluids*, vol. 18, pp. 887-914, 1994.

[Mav96] D.J. Mavriplis and V. Venkatakrishnan, A 3D Agglomeration Multigrid Solver for the Reynolds-Averaged Navier-Stokes Equations on Unstructured Meshes, *International Journal for Numerical Methods in Fluids*, vol. 23, pp. 527-544, 1996.

[Mou16] F. Moukalled, M. Darwish and L. Mangani, The Finite Volume Method in Computational Fluid Dynamics - An Advanced Introduction with OpenFOAM® and Matlab®, *Springer International Publishing Switzerland*, 2016.

[Mun98] B.R. Munson, D.F. Young and T.I. Okiishi, Fundamentals of Fluid Mechanics, *John Wiley & Sons, Inc.*, New York, 1998.

[Per12] P. Persson, High-Order Navier-Stokes Simulations Using a Sparse Line-based Discontinuous Galerkin method, *50th AIAA Aerospace Sciences Meeting including the New Horizons Forum and Aerospace Exposition,* AIAA 2012-456.

[Roe81] P. Roe, Approximate Riemann Solvers, Parameter Vectors and Difference Schemes, *Journal of Computational Physics*, vol. 43, pp. 357-371, 1981.

[Ruu05] S. Ruuth, Global Optimization of Strong-Stability Preserving Runge-Kutta Methods, *Mathematics of Computation*, vol. 75, pp. 183–207, 2006.

[Sar14] S.S. Sarakinos, G.N. Lygidakis and I.K. Nikolos, Evaluation of a Parallel Agglomeration Multigrid Finite-Volume Algorithm, named Galatea-I, for the Simulation of Incompressible Flows on 3D Hybrid Unstructured Grids, *Proceedings of the International Mechanical Engineering Congress and Exposition, ASME-IMECE2014,* Montreal, Canada, 14-20 November 2014, IMECE2014-39759.

[Spa16] P.R. Spalart and V. Venkatakrishnan, On the Role and Challenges of CFD in the Aerospace Industry, *The Aeronautical Journal,* vol. 120, pp. 209-232, 2016.

[Ste81] J.L. Steger and R.F. Warming, Flux Vector Splitting of the Inviscid Gasdynamic Equations with Application to Finite Difference Methods, *Journal of Computational Physics*, vol. 40, pp. 263-293, 1981.

[Sor03] K.A. Sorensen, O. Hassan, K. Morgan and N.P. Weatherill, A Multigrid Accelerated Hybrid Unstructured Mesh Method for 3D Compressible Turbulent Flow, *Computational Mechanics*, vol. 31, pp. 101-114, 2003.

[Swe84] P.K. Sweby, High Resolution Schemes Using Flux Limiters for Hyperbolic Conservation Laws, *SIAM Journal on Numerical Analysis*, vol. 21, pp. 995-1011, 1984.

[Tho08] J.L. Thomas, B. Diskin, C.L. Rumsey, Towards Verification of Unstructured-Grid Solvers, *AIAA Journal*, vol. 46, pp. 3070-3079, 2008.

[Tor97] E.F. Toro, Riemann Solvers and Numerical Methods for Fluid Dynamics. A Practical Application. 2nd Edition, *Springer*, 1997.

[VanA82] G.D. Van Albada, B. Van Leer and W.W. Roberts, A Comparative Study of Computational Methods in Cosmic Gas Dynamics, *Astronomy and Astrophysics*, vol. 108, pp.46-84, 1982.

[Ven95] V. Venkatakrishnan, Implicit Schemes and Parallel Computing in Unstructured Grid CFD, *Proceedings of the 26th Computational Fluid Dynamics Lecture Series Program*, Von Karman Institute for Fluid Dynamics, Rhode, Saint-Genese, Belgium,13-17 March 1995.

[Yan14] H.Q. Yang, Z.J. Chen, A. Przekwas and J. Dudley, A High-Order CFD Method Using Successive Differentiation, *Journal of Computational Physics,* vol. 281, pp. 690–707, 2015.

[Yan15] H. Q. Yang, R. E. Harris, Development of Vertex–Centered, High-Order Schemes and Implementation in FUN3D, *22nd AIAA Computational Fluid Dynamics Conference,* 22-26 June Dallas TX, 2015.

[Yan16] H. Q. Yang, R. E. Harris, Development of Vertex-Centered High-Order Schemes and Implementation in FUN3D, *AIAA Journal*, vol. 54, pp. 3742-3760, 2016.

APPENDIX A: Jacobian Matrix Decomposition

The Jacobian matrix of the convective flux vector $\vec{H}^{inv}$ is analyzed via the eigenvalue decomposition as follows [Hir90, Lan98]

$$\underline{A} = \underline{T}\,\underline{\Lambda}\,\underline{T}^{-1} \tag{A.1}$$

where $\underline{\Lambda}$ is a *5x5* diagonal matrix, whose entries are the eigenvalues of the Jacobian matrix $\underline{A}$, defined as [Hir90]

$$\underline{\Lambda} = diag\{\hat{V}_n|\vec{n}|, \hat{V}_n|\vec{n}|, \hat{V}_n|\vec{n}|, (\hat{V}_n + c)|\vec{n}|, (\hat{V}_n - c)|\vec{n}|\} \tag{A.2}$$

while $\underline{T}$ is a matrix, including the eigenvectors of the Jacobian matrix $\underline{A}$ [Hir90]

$$\underline{T} = \begin{bmatrix} \hat{n}_x & \hat{n}_y & \hat{n}_z & C & C \\ \hat{n}_x u & \hat{n}_y u - \hat{n}_z \rho & \hat{n}_z u + \hat{n}_y \rho & C(u + c\hat{n}_x) & C(u - c\hat{n}_x) \\ \hat{n}_x v + \hat{n}_z \rho & \hat{n}_y v & \hat{n}_z v - \hat{n}_x \rho & C(v + c\hat{n}_y) & C(v - c\hat{n}_y) \\ \hat{n}_x w - \hat{n}_y \rho & \hat{n}_y w + \hat{n}_x \rho & \hat{n}_z w & C(w + c\hat{n}_z) & C(w - c\hat{n}_z) \\ \frac{\vec{V}^2}{2}\hat{n}_x + \rho X & \frac{\vec{V}^2}{2}\hat{n}_y + \rho Y & \frac{\vec{V}^2}{2}\hat{n}_z + \rho Z & C(H + c\hat{V}_n) & C(H - c\hat{V}_n) \end{bmatrix} \tag{A.3}$$

and $\underline{T}^{-1}$ is its inverse matrix [Hir90]

$$\underline{T}^{-1} = \begin{bmatrix} B\hat{n}_x - \frac{X}{\rho} & \frac{(\gamma-1)\hat{n}_x u}{c^2} & \frac{(\gamma-1)\hat{n}_x v}{c^2} + \frac{\hat{n}_z}{\rho} & \frac{(\gamma-1)\hat{n}_x w}{c^2} - \frac{\hat{n}_y}{\rho} & -\frac{(\gamma-1)\hat{n}_x}{c^2} \\ B\hat{n}_y - \frac{Y}{\rho} & \frac{(\gamma-1)\hat{n}_y u}{c^2} - \frac{\hat{n}_z}{\rho} & \frac{(\gamma-1)\hat{n}_y v}{c^2} & \frac{(\gamma-1)\hat{n}_y w}{c^2} + \frac{\hat{n}_x}{\rho} & -\frac{(\gamma-1)\hat{n}_y}{c^2} \\ B\hat{n}_z - \frac{Z}{\rho} & \frac{(\gamma-1)\hat{n}_z u}{c^2} + \frac{\hat{n}_y}{\rho} & \frac{(\gamma-1)\hat{n}_z v}{c^2} - \frac{\hat{n}_x}{\rho} & \frac{(\gamma-1)\hat{n}_z w}{c^2} & -\frac{(\gamma-1)\hat{n}_z}{c^2} \\ \frac{c}{\rho}\left(\frac{(\gamma-1)\vec{V}^2}{2c^2} - \frac{\hat{V}_n}{c}\right) & \left(\hat{n}_x - \frac{(\gamma-1)u}{c}\right)\frac{1}{\rho} & \left(\hat{n}_y - \frac{(\gamma-1)v}{c}\right)\frac{1}{\rho} & \left(\hat{n}_z - \frac{(\gamma-1)w}{c}\right)\frac{1}{\rho} & \frac{(\gamma-1)}{\rho c} \\ \frac{c}{\rho}\left(\frac{(\gamma-1)\vec{V}^2}{2c^2} + \frac{\hat{V}_n}{c}\right) & \left(-\hat{n}_x - \frac{(\gamma-1)u}{c}\right)\frac{1}{\rho} & \left(-\hat{n}_y - \frac{(\gamma-1)v}{c}\right)\frac{1}{\rho} & \left(-\hat{n}_z - \frac{(\gamma-1)w}{c}\right)\frac{1}{\rho} & \frac{(\gamma-1)}{\rho c} \end{bmatrix} \tag{A.4}$$

where u, v and w are the components of velocity $\vec{V}$, $\vec{\hat{n}} = (\hat{n}_x, \hat{n}_y, n_z)$ is the unit normal vector and $\hat{V}_n = \vec{V} \cdot \vec{\hat{n}}$ is the value of corresponding velocity. The terms C, H, X, Y, Z and B are auxiliary values defined as:

$$\begin{aligned} C &= \frac{\rho}{2c} \\ H &= \frac{\vec{V}^2}{2} + \frac{c^2}{(\gamma-1)} \\ X &= \hat{n}_z v - \hat{n}_y w \\ Y &= \hat{n}_x w - \hat{n}_z u \\ Z &= \hat{n}_y u - \hat{n}_x v \\ B &= 1 - \frac{(\gamma-1)\vec{V}^2}{2c^2} \end{aligned} \tag{A.5}$$

Depending on the eigenvalue λ the Jacobian matrix for the Roe's approximate Riemann solver is calculated as [Hir90]

$$\underline{A}^{\pm} = \underline{T}\,\underline{\Lambda}^{\pm}\,\underline{T}^{-1},\ \ \underline{\Lambda}^{\pm} = diag\{\lambda_i^{\pm}\} \qquad \text{(A.6)}$$
$$\left|\underline{A}\right| = \underline{T}\left|\underline{\Lambda}\right|\underline{T}^{-1},\ \ \left|\underline{\Lambda}\right| = diag\{|\lambda_i|\}$$

while the eigenvalues λ as:

$$\lambda_i^{+} = max(\lambda_i, 0), \qquad i = 1, \dots 5 \qquad \text{(A.7)}$$
$$\lambda_i^{-} = min(\lambda_i, 0), \qquad i = 1, \dots 5$$

APPENDIX B: Convergence Results (Isentropic Vortex)

Table B.1: *Errors and convergence rates for Equilateral Grid (Type I) at t= 7 s*

EQUILATERAL I		T=7	
L	**N**	**h^N**	**log10(h^N)**
1	*20*	***0.8436***	*-0.0738*
1	*40*	***0.4529***	*-0.3440*
1	*80*	***0.2297***	*-0.6388*
1	*160*	***0.1157***	*-0.9366*
1	*320*	***0.0578***	*-1.2379*

L2	L2	L2	L2	L2	L2	L2	L2	L2
Error(ρ)	**Error(ρu)**	**Error(ρe)**	**log10(error(ρ))**	**log10(error(ρu))**	**log10(error(ρe))**	**(ρ)**	**(ρu)**	**(ρe)**
1.011E-03	*9.356E-03*	*1.307E-02*	*-2.995E+00*	*-2.029E+00*	*-1.884E+00*	**Slope**	**Slope**	**Slope**
3.712E-04	*2.976E-03*	*3.946E-03*	*-3.430E+00*	*-2.526E+00*	*-2.404E+00*	*1.61*	*1.84*	*1.93*
5.929E-05	*3.259E-04*	*4.315E-04*	*-4.227E+00*	*-3.487E+00*	*-3.365E+00*	*2.70*	*3.26*	*3.26*
1.106E-05	*2.909E-05*	*4.017E-05*	*-4.956E+00*	*-4.536E+00*	*-4.396E+00*	*2.45*	*3.52*	*3.46*
2.488E-06	*3.773E-06*	*5.025E-06*	*-5.604E+00*	*-5.423E+00*	*-5.299E+00*	*2.15*	*2.94*	*3.00*
					Average slope	***2.23***	***2.89***	***2.91***

L1	L1	L1	L1	L1	L1	L1	L1	L1
Error(ρ)	**Error(ρu)**	**Error(ρe)**	**log10(error(ρ))**	**log10(error(ρu))**	**log10(error(ρe))**	**(ρ)**	**(ρu)**	**(ρe)**
2.837E-04	*2.754E-03*	*4.004E-03*	*-3.547E+00*	*-2.560E+00*	*-2.397E+00*	**Slope**	**Slope**	**Slope**
8.484E-05	*7.107E-04*	*1.022E-03*	*-4.071E+00*	*-3.148E+00*	*-2.991E+00*	*1.94*	*2.18*	*2.20*
1.304E-05	*7.250E-05*	*1.093E-04*	*-4.885E+00*	*-4.140E+00*	*-3.962E+00*	*2.76*	*3.36*	*3.29*
2.241E-06	*7.270E-06*	*1.231E-05*	*-5.650E+00*	*-5.138E+00*	*-4.910E+00*	*2.57*	*3.35*	*3.18*
4.757E-07	*9.579E-07*	*1.734E-06*	*-6.323E+00*	*-6.019E+00*	*-5.761E+00*	*2.23*	*2.92*	*2.82*
					Average slope	***2.38***	***2.95***	***2.87***

Lmax	Lmax	Lmax	Lmax	Lmax	Lmax	Lmax	Lmax	Lmax
Error(ρ)	**Error(ρu)**	**Error(ρe)**	**log10(error(ρ))**	**log10(error(ρu))**	**log10(error(ρe))**	**(ρ)**	**(ρu)**	**(ρe)**
1.5842E-02	*9.4260E-02*	*1.6983E-01*	*-1.8002E+00*	*1.0257E+00*	*-7.6999E-01*	**Slope**	**Slope**	**Slope**
7.1290E-03	*4.1755E-02*	*5.9835E-02*	*-2.1470E+00*	*-1.3793E+00*	*-1.2230E+00*	*1.28*	*1.31*	*1.68*
1.2758E-03	*5.2295E-03*	*7.5776E-03*	*-2.8942E+00*	*-2.2815E+00*	*-2.1205E+00*	*2.53*	*3.06*	*3.04*
2.4505E-04	*4.8608E-04*	*6.9753E-04*	*-3.6107E+00*	*-3.3133E+00*	*-3.1564E+00*	*2.41*	*3.46*	*3.48*
5.3963E-05	*7.2616E-05*	*9.1160E-05*	*-4.2679E+00*	*-4.1390E+00*	*-4.0402E+00*	*2.18*	*2.74*	*2.93*
					Average slope	***2.10***	***2.64***	***2.78***

Table B.2: *Errors and convergence rates for Orthogonal Grid (Type II) at t= 7 s*

ORTHOGONAL II		**T=7**	
L	**N**	**h^N**	**log10(h^N)**
1	*20*	***0.6897***	*-0.1614*
1	*40*	***0.3492***	*-0.4570*
1	*80*	***0.1757***	*-0.7553*
1	*160*	***0.0881***	*-1.0550*
1	*320*	***0.0441***	*-1.3553*

L2	**L2**	**L2**	**L2**	**L2**	**L2**	**L2**	**L2**	**L2**
Error(ρ)	**Error(ρu)**	**Error(ρe)**	**log10(error(ρ))**	**log10(error(ρu))**	**log10(error(ρe))**	**(ρ)**	**(ρu)**	**(ρe)**
8.217E-04	*7.206E-03*	*9.395E-03*	*-3.085E+00*	*-2.142E+00*	*-2.027E+00*	**Slope**	**Slope**	**Slope**
2.018E-04	*1.579E-03*	*1.831E-03*	*-3.695E+00*	*-2.802E+00*	*-2.737E+00*	*2.06*	*2.23*	*2.40*
3.438E-05	*1.435E-04*	*1.685E-04*	*-4.464E+00*	*-3.843E+00*	*-3.773E+00*	*2.58*	*3.49*	*3.47*
7.052E-06	*1.442E-05*	*1.828E-05*	*-5.152E+00*	*-4.841E+00*	*-4.738E+00*	*2.30*	*3.33*	*3.22*
1.646E-06	*2.345E-06*	*2.929E-06*	*5.784E+00*	*-5.630E+00*	*-5.533E+00*	*2.10*	*2.63*	*2.65*
					Average slope	***2.26***	***2.92***	***2.94***

L1	**L1**	**L1**	**L1**	**L1**	**L1**	**L1**	**L1**	**L1**
Error(ρ)	**Error(ρu)**	**Error(ρe)**	**log10(error(ρ))**	**log10(error(ρu))**	**log10(error(ρe))**	**(ρ)**	**(ρu)**	**(ρe)**
2.235E-04	*1.981E-03*	*2.901E-03*	*-3.651E+00*	*-2.703E+00*	*-2.537E+00*	**Slope**	**Slope**	**Slope**
4.690E-05	*3.440E-04*	*4.618E-04*	*-4.329E+00*	*-3.463E+00*	*-3.336E+00*	*2.29*	*2.57*	*2.70*
7.084E-06	*2.917E-05*	*4.379E-05*	*-5.150E+00*	*-4.535E+00*	*-4.359E+00*	*2.75*	*3.59*	*3.43*
1.345E-06	*3.020E-06*	*5.246E-06*	*-5.871E+00*	*-5.520E+00*	*-5.280E+00*	*2.41*	*3.29*	*3.08*
3.068E-07	*4.570E-07*	*8.803E-07*	*-6.513E+00*	*-6.340E+00*	*-6.055E+00*	*2.14*	*2.73*	*2.58*
					Average slope	***2.40***	***3.05***	***2.95***

Lmax	**Lmax**	**Lmax**	**Lmax**	**Lmax**	**Lmax**	**Lmax**	**Lmax**	**Lmax**
Error(ρ)	**Error(ρu)**	**Error(ρe)**	**log10(error(ρ))**	**log10(error(ρu))**	**log10(error(ρe))**	**(ρ)**	**(ρu)**	**(ρe)**
1.5173E-02	*8.3985E-02*	*1.3061E-01*	*-1.8189E+00*	*-1.0758E+00*	*-8.8401E-01*	**Slope**	**Slope**	**Slope**
3.9076E-03	*2.2827E-02*	*2.8686E-02*	*-2.4081E+00*	*-1.6415E+00*	*-1.5423E+00*	*1.99*	*1.91*	*2.23*
6.9066E-04	*2.3590E-03*	*2.5652E-03*	*-3.1607E+00*	*-2.6273E+00*	*-2.5909E+00*	*2.52*	*3.30*	*3.51*
1.5020E-04	*2.8361E-04*	*3.3301E-04*	*-3.8233E+00*	*-3.5473E+00*	*-3.4775E+00*	*2.21*	*3.07*	*2.96*
3.5056E-05	*4.8871E-05*	*6.1079E-05*	*-4.4552E+00*	*-4.3109E+00*	*-4.2141E+00*	*2.10*	*2.54*	*2.45*
					Average slope	***2.21***	***2.71***	***2.79***

Table B.3: *Errors and convergence rates for Orthogonal Grid (Type III) at t= 7 s*

ORTHOGONAL III		**T=7**	
L	**N**	**h^N**	**log10(h^N)**
1	*20*	***0.9524***	*-0.0212*
1	*40*	***0.4878***	*-0.3118*
1	*80*	***0.2469***	*-0.6075*
1	*160*	***0.1242***	*-0.9058*
1	*320*	***0.0623***	*-1.2055*

L2	**L2**	**L2**	**L2**	**L2**	**L2**	**L2**	**L2**	**L2**
Error(ρ)	**Error(ρu)**	**Error(ρe)**	**log10(error(ρ))**	**log10(error(ρu))**	**log10(error(ρe))**	**(ρ)**	**(ρu)**	**(ρe)**
1.171E-03	*1.134E-02*	*1.518E-02*	*-2.931E+00*	*-1.945E+00*	*-1.819E+00*	**Slope**	**Slope**	**Slope**
6.729E-04	*6.234E-03*	*7.421E-03*	*-3.172E+00*	*-2.205E+00*	*-2.130E+00*	*0.83*	*0.89*	*1.07*
2.273E-04	*2.210E-03*	*2.403E-03*	*-3.643E+00*	*-2.656E+00*	*-2.619E+00*	*1.59*	*1.52*	*1.66*
5.795E-05	*5.861E-04*	*6.170E-04*	*-4.237E+00*	*-3.232E+00*	*-3.210E+00*	*1.99*	*1.93*	*1.98*
1.436E-05	*1.471E-04*	*1.535E-04*	*-4.843E+00*	*-3.832E+00*	*-3.814E+00*	*2.02*	*2.00*	*2.02*
					Average slope	***1.61***	***1.59***	***1.68***

L1	**L1**	**L1**	**L1**	**L1**	**L1**	**L1**	**L1**	**L1**
Error(ρ)	**Error(ρu)**	**Error(ρe)**	**log10(error(ρ))**	**log10(error(ρu))**	**log10(error(ρe))**	**(ρ)**	**(ρu)**	**(ρe)**
2.980E-04	*3.011E-03*	*4.425E-03*	*-3.526E+00*	*-2.521E+00*	*-2.354E+00*	**Slope**	**Slope**	**Slope**
1.488E-04	*1.418E-03*	*1.856E-03*	*-3.827E+00*	*-2.848E+00*	*-2.731E+00*	*1.04*	*1.13*	*1.30*
4.220E-05	*4.310E-04*	*5.264E-04*	*-4.375E+00*	*-3.366E+00*	*-3.279E+00*	*1.85*	*1.75*	*1.85*
9.680E-06	*1.089E-04*	*1.276E-04*	*-5.014E+00*	*-3.963E+00*	*-3.894E+00*	*2.14*	*2.00*	*2.06*
2.272E-06	*2.712E-05*	*3.127E-05*	*-5.644E+00*	*-4.567E+00*	*-4.505E+00*	*2.10*	*2.01*	*2.04*
					Average slope	***1.78***	***1.72***	***1.81***

Lmax	**Lmax**	**Lmax**	**Lmax**	**Lmax**	**Lmax**	**Lmax**	**Lmax**	**Lmax**
Error(ρ)	**Error(ρu)**	**Error(ρe)**	**log10(error(ρ))**	**log10(error(ρu))**	**log10(error(ρe))**	**(ρ)**	**(ρu)**	**(ρe)**
1.9962E-02	*1.3211E-01*	*1.9078E-01*	*-1.69979743*	*-0.879064533*	*-0.719470948*	**Slope**	**Slope**	**Slope**
1.2303E-02	*8.0891E-02*	*1.0049E-01*	*-1.90998199*	*-1.092101564*	*-0.997897158*	*0.72*	*0.73*	*0.96*
4.2449E-03	*3.1025E-02*	*3.5464E-02*	*-2.37212952*	*-1.508294302*	*-1.450206172*	*1.56*	*1.41*	*1.53*
1.0226E-03	*8.4404E-03*	*8.7714E-03*	*-2.99029354*	*-2.073636013*	*-2.056931751*	*2.07*	*1.89*	*2.03*
2.3584E-04	*2.1077E-03*	*2.0631E-03*	*-3.62738065*	*-2.676184498*	*-2.685484332*	*2.13*	*2.01*	*2.10*
					Average slope	***1.62***	***1.51***	***1.65***

Table B.4: *Errors and convergence rates for Distorted Grid (Type IV) at t= 7 s*

DISTORTED IV		T=7	
L	**N**	**h^N**	**log10(h^N)**
1	*20*	***0.8436***	*-0.0738*
1	*40*	***0.4529***	*-0.3440*
1	*80*	***0.2297***	*-0.6388*
1	*160*	***0.1157***	*-0.9366*
1	*320*	***0.0578***	*-1.2379*

L2	L2	L2	L2	L2	L2	L2	L2	L2
Error(ρ)	**Error(ρu)**	**Error(ρe)**	**log10(error(ρ))**	**log10(error(ρu))**	**log10(error(ρe))**	**(ρ)**	**(ρu)**	**(ρe)**
1.182E-03	*9.030E-03*	*1.243E-02*	*-2.927E+00*	*-2.044E+00*	*-1.905E+00*	**Slope**	**Slope**	**Slope**
5.884E-04	*2.671E-03*	*3.844E-03*	*-3.230E+00*	*-2.573E+00*	*-2.415E+00*	*1.12*	*1.96*	*1.89*
2.058E-04	*9.293E-04*	*9.607E-04*	*-3.687E+00*	*-3.032E+00*	*-3.017E+00*	*1.55*	*1.56*	*2.04*
4.187E-05	*1.747E-04*	*1.817E-04*	*-4.378E+00*	*-3.758E+00*	*-3.741E+00*	*2.32*	*2.44*	*2.43*
1.207E-05	*4.889E-05*	*5.132E-05*	*-4.918E+00*	*-4.311E+00*	*-4.290E+00*	*1.79*	*1.84*	*1.82*
					Average slope	***1.70***	***1.95***	***2.05***

L1	L1	L1	L1	L1	L1	L1	L1	L1
Error(ρ)	**Error(ρu)**	**Error(ρe)**	**log10(error(ρ))**	**log10(error(ρu))**	**log10(error(ρe))**	**(ρ)**	**(ρu)**	**(ρe)**
5.826E-04	*2.886E-03*	*4.845E-03*	*-3.235E+00*	*-2.540E+00*	*-2.315E+00*	**Slope**	**Slope**	**Slope**
2.377E-04	*7.347E-04*	*1.372E-03*	*-3.624E+00*	*-3.134E+00*	*-2.862E+00*	*1.44*	*2.20*	*2.03*
7.868E-05	*2.385E-04*	*3.619E-04*	*-4.104E+00*	*-3.623E+00*	*-3.441E+00*	*1.63*	*1.66*	*1.96*
1.449E-05	*3.992E-05*	*6.097E-05*	*-4.839E+00*	*4.399E+00*	*-4.215E+00*	*2.47*	*2.61*	*2.60*
4.063E-06	*1.072E-05*	*1.628E-05*	*-5.391E+00*	*-4.970E+00*	*-4.788E+00*	*1.83*	*1.90*	*1.90*
					Average slope	***1.84***	***2.09***	***2.12***

Lmax	Lmax	Lmax	Lmax	Lmax	Lmax	Lmax	Lmax	Lmax
Error(ρ)	**Error(ρu)**	**Error(ρe)**	**log10(error(ρ))**	**log10(error(ρu))**	**log10(error(ρe))**	**(ρ)**	**(ρu)**	**(ρe)**
1.4981E-02	*9.6456E-02*	*1.5181E-01*	*-1.8245E+00*	*-1.0157E+00*	*-8.1871E-01*	**Slope**	**Slope**	**Slope**
1.0878E-02	*3.2335E-02*	*5.5092E-02*	*-1.9635E+00*	*-1.4903E+00*	*-1.2589E+00*	*0.51*	*1.76*	*1.63*
2.8372E-03	*1.1804E-02*	*1.5020E-02*	*-2.5471E+00*	*-1.9280E+00*	*-1.8233E+00*	*1.98*	*1.48*	*1.91*
1.0414E-03	*2.5916E-03*	*3.3408E-03*	*-2.9824E+00*	*-2.5864E+00*	*-2.4762E+00*	*1.46*	*2.21*	*2.19*
2.4678E-04	*7.1541E-04*	*1.0691E-03*	*-3.6077E+00*	*-3.1454E+00*	*-2.9710E+00*	*2.08*	*1.86*	*1.64*
					Average slope	***1.51***	***1.83***	***1.84***

Table B.5: *Errors and convergence rates for Equilateral Grid (Type I) at t= 60 s*

EQUILATERAL I		T=60	
L	**N**	**h^N**	**log10(h^N)**
1	*20*	***0.8436***	*-0.0738*
1	*40*	***0.4529***	*-0.3440*
1	*80*	***0.2297***	*-0.6388*
1	*160*	***0.1157***	*-0.9366*
1	*320*	***0.0578***	*-1.2379*

L2	L2	L2	L2	L2	L2	L2	L2	L2
Error(ρ)	**Error(ρu)**	**Error(ρe)**	**log10(error(ρ))**	**log10(error(ρu))**	**log10(error(ρe))**	**(ρ)**	**(ρu)**	**(ρe)**
1.298E-03	*1.396E-02*	*1.971E-02*	*-2.887E+00*	*-1.855E+00*	*-1.705E+00*	**Slope**	**Slope**	**Slope**
9.424E-04	*9.899E-03*	*1.362E-02*	*-3.026E+00*	*-2.004E+00*	*-1.866E+00*	*0.51*	*0.55*	*0.59*
2.311E-04	*1.822E-03*	*3.034E-03*	*-3.636E+00*	*-2.739E+00*	*-2.518E+00*	*2.07*	*2.49*	*2.21*
2.557E-05	*1.319E-04*	*2.265E-04*	*-4.592E+00*	*-3.880E+00*	*-3.645E+00*	*3.21*	*3.83*	*3.78*
5.014E-06	*1.112E-05*	*1.741E-05*	*-5.300E+00*	*-4.954E+00*	*-4.759E+00*	*2.35*	*3.57*	*3.70*
					Average slope	***2.04***	***2.61***	***2.57***

L1	L1	L1	L1	L1	L1	L1	L1	L1
Error(ρ)	**Error(ρu)**	**Error(ρe)**	**log10(error(ρ))**	**log10(error(ρu))**	**log10(error(ρe))**	**(ρ)**	**(ρu)**	**(ρe)**
3.164E-04	*4.170E-03*	*6.040E-03*	*-3.500E+00*	*-2.380E+00*	*-2.219E+00*	**Slope**	**Slope**	**Slope**
1.870E-04	*2.826E-03*	*3.674E-03*	*-3.728E+00*	*-2.549E+00*	*-2.435E+00*	*0.85*	*0.63*	*0.80*
5.034E-05	*4.594E-04*	*6.823E-04*	*-4.298E+00*	*-3.338E+00*	*-3.166E+00*	*1.93*	*2.68*	*2.48*
8.127E-06	*3.655E-05*	*5.985E-05*	*-5.090E+00*	*-4.437E+00*	*-4.223E+00*	*2.66*	*3.69*	*3.55*
1.646E-06	*3.709E-06*	*6.513E-06*	*-5.784E+00*	*-5.431E+00*	*-5.186E+00*	*2.30*	*3.30*	*3.20*
					Average slope	***1.93***	***2.57***	***2.51***

Lmax	Lmax	Lmax	Lmax	Lmax	Lmax	Lmax	Lmax	Lmax
Error(ρ)	**Error(ρu)**	**Error(ρe)**	**log10(error(ρ))**	**log10(error(ρu))**	**log10(error(ρe))**	**(ρ)**	**(ρu)**	**(ρe)**
1.8958E-02	*1.5256E-01*	*1.9849E-01*	*-1.7222E+00*	*-8.1656E-01*	*-7.0227E-01*	**Slope**	**Slope**	**Slope**
1.6462E-02	*1.1609E-01*	*1.8520E-01*	*-1.7835E+00*	*-9.3519E-01*	*-7.3237E-01*	*0.23*	*0.44*	*0.11*
4.5654E-03	*2.5371E-02*	*4.9192E-02*	*-2.3405E+00*	*-1.5957E+00*	*-1.3081E+00*	*1.89*	*2.24*	*1.95*
4.2968E-04	*1.8246E-03*	*3.8222E-03*	*-3.3669E+00*	*-2.7388E+00*	*-2.4177E+00*	*3.45*	*3.84*	*3.73*
7.8762E-05	*1.4601E-04*	*2.6831E-04*	*-4.1037E+00*	*-3.8356E+00*	*-3.5714E+00*	*2.45*	*3.64*	*3.83*
					Average slope	***2.00***	***2.54***	***2.40***

Table B.6: *Errors and convergence rates for Orthogonal Grid (Type II) at t= 60 s*

ORTHOGONAL II	T=60		
L	**N**	**h^N**	**log10(h^N)**
1	*20*	***0.6897***	*-0.1614*
1	*40*	***0.3492***	*-0.4570*
1	*80*	***0.1757***	*-0.7553*
1	*160*	***0.0881***	*-1.0550*
1	*320*	***0.0441***	*-1.3553*

L2	L2	L2	L2	L2	L2	L2	L2	L2
Error(ρ)	**Error(ρu)**	**Error(ρe)**	**log10(error(ρ))**	**log10(error(ρu))**	**log10(error(ρe))**	**(ρ)**	**(ρu)**	**(ρe)**
1.244E-03	*1.313E-02*	*1.796E-02*	*-2.905E+00*	*-1.882E+00*	*-1.746E+00*	**Slope**	**Slope**	**Slope**
7.121E-04	*7.684E-03*	*1.058E-02*	*-3.147E+00*	*-2.114E+00*	*-1.976E+00*	*0.82*	*0.79*	*0.78*
9.500E-05	*7.389E-04*	*1.290E-03*	*-4.022E+00*	*-3.131E+00*	*-2.889E+00*	*2.93*	*3.41*	*3.06*
1.413E-05	*5.053E-05*	*8.781E-05*	*-4.850E+00*	*-4.296E+00*	*-4.056E+00*	*2.76*	*3.89*	*3.89*
3.252E-06	*5.752E-06*	*8.474E-06*	*-5.488E+00*	*-5.240E+00*	*-5.072E+00*	*2.12*	*3.14*	*3.38*
					Average slope	***2.16***	***2.81***	***2.78***

L1	L1	L1	L1	L1	L1	L1	L1	L1
Error(ρ)	**Error(ρu)**	**Error(ρe)**	**log10(error(ρ))**	**log10(error(ρu))**	**log10(error(ρe))**	**(ρ)**	**(ρu)**	**(ρe)**
2.707E-04	*4.002E-03*	*5.836E-03*	*-3.568E+00*	*-2.398E+00*	*-2.234E+00*	**Slope**	**Slope**	**Slope**
1.399E-04	*2.056E-03*	*2.758E-03*	*-3.854E+00*	*-2.687E+00*	*-2.559E+00*	*0.97*	*0.98*	*1.10*
2.220E-05	*1.755E-04*	*2.841E-04*	*-4.654E+00*	*-3.756E+00*	*-3.547E+00*	*2.68*	*3.58*	*3.31*
3.513E-06	*1.207E-05*	*2.111E-05*	*-5.454E+00*	*-4.918E+00*	*-4.675E+00*	*2.67*	*3.88*	*3.77*
7.828E-07	*1.363E-06*	*2.227E-06*	*-6.106E+00*	*-5.865E+00*	*-5.652E+00*	*2.17*	*3.15*	*3.25*
					Average slope	***2.12***	***2.90***	***2.86***

Lmax	Lmax	Lmax	Lmax	Lmax	Lmax	Lmax	Lmax	Lmax
Error(ρ)	**Error(ρu)**	**Error(ρe)**	**log10(error(ρ))**	**log10(error(ρu))**	**log10(error(ρe))**	**(ρ)**	**(ρu)**	**(ρe)**
2.1213E-02	*1.5128E-01*	*2.1736E-01*	*-1.6734E+00*	*-8.2022E-01*	*-6.6283E-01*	**Slope**	**Slope**	**Slope**
1.2803E-02	*9.6764E-02*	*1.5300E-01*	*-1.8927E+00*	*-1.0143E+00*	*-8.1530E-01*	*0.74*	*0.66*	*0.52*
1.5294E-03	*8.3746E-03*	*1.9430E-02*	*-2.8155E+00*	*-2.0770E+00*	*-1.7115E+00*	*3.09*	*3.56*	*3.00*
1.9688E-04	*6.7222E-04*	*1.2921E-03*	*-3.7058E+00*	*-3.1725E+00*	*-2.8887E+00*	*2.97*	*3.66*	*3.93*
4.3373E-05	*9.6684E-05*	*1.4189E-04*	*-4.3628E+00*	*-4.0146E+00*	*-3.8481E+00*	*2.19*	*2.80*	*3.19*
					Average slope	***2.25***	***2.67***	***2.66***

Table B.7: *Errors and convergence rates for Orthogonal Grid (Type III) at t= 60 s*

ORTHOGONAL III	T=60		
L	**N**	**h^N**	**log10(h^N)**
1	*20*	***0.9524***	*-0.0212*
1	*40*	***0.4878***	*-0.3118*
1	*80*	***0.2469***	*-0.6075*
1	*160*	***0.1242***	*-0.9058*
1	*320*	***0.0623***	*-1.2055*

L2	L2	L2	L2	L2	L2	L2	L2	L2
Error(ρ)	**Error(ρu)**	**Error(ρe)**	**log10(error(ρ))**	**log10(error(ρu))**	**log10(error(ρe))**	**(ρ)**	**(ρu)**	**(ρe)**
1.316E-03	*1.406E-02*	*1.903E-02*	*-2.881E+00*	*-1.852E+00*	*-1.721E+00*	**Slope**	**Slope**	**Slope**
1.134E-03	*1.238E-02*	*1.550E-02*	*-2.946E+00*	*-1.907E+00*	*-1.810E+00*	*0.22*	*0.19*	*0.31*
4.958E-04	*4.742E-03*	*7.122E-03*	*-3.305E+00*	*-2.324E+00*	*-2.147E+00*	*1.21*	*1.41*	*1.14*
5.387E-05	*3.603E-04*	*6.254E-04*	*-4.269E+00*	*-3.443E+00*	*-3.204E+00*	*3.23*	*3.75*	*3.54*
8.754E-06	*2.590E-05*	*4.359E-05*	*-5.058E+00*	*-4.587E+00*	*-4.361E+00*	*2.63*	*3.82*	*3.86*
					Average slope	***1.83***	***2.29***	***2.21***

L1	L1	L1	L1	L1	L1	L1	L1	L1
Error(ρ)	**Error(ρu)**	**Error(ρe)**	**log10(error(ρ))**	**log10(error(ρu))**	**log10(error(ρe))**	**(ρ)**	**(ρu)**	**(ρe)**
3.639E-04	*3.908E-03*	*6.316E-03*	*-3.439E+00*	*-2.408E+00*	*-2.200E+00*	**Slope**	**Slope**	**Slope**
2.519E-04	*3.855E-03*	*5.090E-03*	*-3.599E+00*	*-2.414E+00*	*-2.293E+00*	*0.55*	*0.02*	*0.32*
1.077E-04	*1.359E-03*	*1.870E-03*	*-3.968E+00*	*-2.867E+00*	*-2.728E+00*	*1.25*	*1.53*	*1.47*
1.255E-05	*9.318E-05*	*1.398E-04*	*-4.901E+00*	*-4.031E+00*	*-3.854E+00*	*3.13*	*3.90*	*3.78*
2.029E-06	*7.140E-06*	*1.079E-05*	*-5.693E+00*	*-5.146E+00*	*-4.967E+00*	*2.64*	*3.72*	*3.71*
					Average slope	***1.89***	***2.29***	***2.32***

Lmax	Lmax	Lmax	Lmax	Lmax	Lmax	Lmax	Lmax	Lmax
Error(ρ)	**Error(ρu)**	**Error(ρe)**	**log10(error(ρ))**	**log10(error(ρu))**	**log10(error(ρe))**	**(ρ)**	**(ρu)**	**(ρe)**
2.1699E-02	*1.5410E-01*	*1.8479E-01*	*-1.66355077*	*-0.812183647*	*-0.733320225*	**Slope**	**Slope**	**Slope**
1.9855E-02	*1.4384E-01*	*1.9023E-01*	*-1.70213748*	*-0.842107197*	*-0.720716165*	*0.13*	*0.10*	*-0.04*
9.1691E-03	*5.8831E-02*	*1.0874E-01*	*-2.03767133*	*-1.230391033*	*-0.963603872*	*1.13*	*1.31*	*0.82*
9.5535E-04	*4.5257E-03*	*1.0471E-02*	*-3.01983833*	*-2.344317134*	*-1.979996544*	*3.29*	*3.73*	*3.41*
1.4947E-04	*2.8554E-04*	*7.3371E-04*	*-3.82545021*	*-3.544326865*	*-3.134475421*	*2.69*	*4.00*	*3.85*
					Average slope	***1.81***	***2.29***	***2.01***

Table B.8: *Errors and convergence rates for Distorted Grid (Type IV) at t= 60 s*

DISTORTED IV		T=60	
L	**N**	**h^N**	**log10(h^N)**
1	*20*	***0.8436***	*-0.0738*
1	*40*	***0.4529***	*-0.3440*
1	*80*	***0.2297***	*-0.6388*
1	*160*	***0.1157***	*-0.9366*
1	*320*	***0.0578***	*-1.2379*

L2	L2	L2	L2	L2	L2	L2	L2	L2
Error(ρ)	**Error(ρu)**	**Error(ρe)**	**log10(error(ρ))**	**log10(error(ρu))**	**log10(error(ρe))**	**(ρ)**	**(ρu)**	**(ρe)**
1.250E-03	*1.348E-02*	*1.987E-02*	*-2.903E+00*	*-1.870E+00*	*-1.702E+00*	**Slope**	**Slope**	**Slope**
1.001E-03	*1.028E-02*	*1.248E-02*	*-3.000E+00*	*-1.988E+00*	*-1.904E+00*	*0.36*	*0.44*	*0.75*
5.569E-04	*6.749E-03*	*7.935E-03*	*-3.254E+00*	*-2.171E+00*	*-2.100E+00*	*0.86*	*0.62*	*0.67*
1.097E-04	*1.243E-03*	*1.608E-03*	*-3.960E+00*	*-2.906E+00*	*-2.794E+00*	*2.37*	*2.47*	*2.33*
3.171E-05	*3.476E-04*	*4.664E-04*	*-4.499E+00*	*-3.459E+00*	*-3.331E+00*	*1.79*	*1.84*	*1.78*
					Average slope	***1.34***	***1.34***	***1.38***

L1	L1	L1	L1	L1	L1	L1	L1	L1
Error(ρ)	**Error(ρu)**	**Error(ρe)**	**log10(error(ρ))**	**log10(error(ρu))**	**log10(error(ρe))**	**(ρ)**	**(ρu)**	**(ρe)**
3.740E-04	*4.203E-03*	*6.160E-03*	*-3.427E+00*	*-2.376E+00*	*-2.210E+00*	**Slope**	**Slope**	**Slope**
3.306E-04	*3.110E-03*	*4.004E-03*	*-3.481E+00*	*-2.507E+00*	*-2.398E+00*	*0.20*	*0.48*	*0.69*
1.865E-04	*1.635E-03*	*2.115E-03*	*-3.729E+00*	*-2.787E+00*	*-2.675E+00*	*0.84*	*0.95*	*0.94*
3.729E-05	*2.765E-04*	*3.759E-04*	*-4.428E+00*	*-3.558E+00*	*-3.425E+00*	*2.35*	*2.59*	*2.52*
1.143E-05	*7.575E-05*	*1.059E-04*	*-4.942E+00*	*-4.121E+00*	*-3.975E+00*	*1.70*	*1.87*	*1.83*
					Average slope	***1.27***	***1.47***	***1.49***

Lmax	Lmax	Lmax	Lmax	Lmax	Lmax	Lmax	Lmax	Lmax
Error(ρ)	**Error(ρu)**	**Error(ρe)**	**log10(error(ρ))**	**log10(error(ρu))**	**log10(error(ρe))**	**(ρ)**	**(ρu)**	**(ρe)**
1.8823E-02	*1.4147E-01*	*1.8707E-01*	*-1.7253E+00*	*-8.4933E-01*	*-7.2799E-01*	**Slope**	**Slope**	**Slope**
1.6407E-02	*1.0832E-01*	*1.4722E-01*	*-1.7850E+00*	*-9.6531E-01*	*-8.3204E-01*	*0.22*	*0.43*	*0.39*
1.0575E-02	*7.9710E-02*	*9.3162E-02*	*-1.9757E+00*	*-1.0985E+00*	*-1.0308E+00*	*0.65*	*0.45*	*0.67*
1.7460E-03	*1.5482E-02*	*2.2263E-02*	*-2.7579E+00*	*-1.8102E+00*	*-1.6524E+00*	*2.63*	*2.39*	*2.09*
7.0474E-04	*4.4452E-03*	*6.9439E-03*	*-3.1520E+00*	*-2.3521E+00*	*-2.1584E+00*	*1.31*	*1.80*	*1.68*
					Average slope	***1.20***	***1.27***	***1.21***