INTERNATIONAL CENTER FOR NUMERICAL METHODS IN ENGINEERING

PUMI: An Explicit 3D Unstructured Finite Element Solver for the Euler Equations

R. Flores
E. Ortega

Publication CIMNE Nº-326, November 2008

PUMI: An Explicit 3D Unstructured Finite Element Solver for the Euler Equations

R. Flores
E. Ortega

Publication CIMNE Nº-326, November 2008

International Center for Numerical Methods in Engineering
Gran Capitán s/n, 08034 Barcelona, Spain

PUMI: AN EXPLICIT 3D UNSTRUCTURED FINITE ELEMENT SOLVER FOR THE EULER EQUATIONS

R. Flores*, E. Ortega

* CIMNE, Edificio C-1, Campus Norte Universidad Politécnica de Cataluña
C/ Gran Capitán s/n
Barcelona 28040, España
e-mail: rflores@cimne.upc.edu

Keywords: CFD, Finite Element, Euler, Explicit Analysis, 3D.

Abstract. *The PUMI flow solver has been developed at CIMNE in to address the need for fast solutions of the flow field around complex geometries. Nowadays calculations involving a number of cells on the order of ten millions are performed routinely. PUMI was created to deal with this kind of large-scale problem using modest hardware, therefore special emphasis was placed on the computational efficiency of the code. Design guidelines where minimum memory requirement, very fast single-threaded performance as well as satisfactory parallel scaling up to a moderate number of threads (as found on current desktop hardware using a small number of multi-core CPUs . In order to speedup the mesh generation activities an unstructured finite element formulation was selected. This papers describes the theoretical basis of the algorithms as well as detail of the implementation that increase the robustness and efficiency of the code.*

1. INTRODUCTION

The quest of the aerospace industry for ever increasing levels of economy, comfort, safety and environmental friendliness has fuelled the development of computational fluid dynamics. Both the large costs and time overheads associated with wind tunnel testing and the tremendous yearly increase in the available computing power have driven the push for extensive numerical simulation. Nowadays complex 3D configurations (e.g. complete aircrafts) are analyzed in a daily basis. Therefore there is a need for fast and efficient flow solvers that yield detailed results with a minimum turnaround time. The most economic method for obtaining realistic flow fields in the transonic range (the main area of interest for the aeronautical industry) is the solution of the Euler equations (simpler strategies like panel methods do not yield accurate results when compressibility effects become important). While today there are more accurate methods available (RANS mainly, as LES is still too complex to be used for production tasks) the inviscid solution is often suitable for preliminary design purposes. Besides, the solution can be improved through coupling with a boundary layer solver thus capturing viscous effects while keeping the solution time at a minimum. Furthermore, an Euler solver lies at the core of every NS solver so it is extremely important to have such a tool available as a basis for later developments. CIMNE being a research institution focusing on software and numerical methods development does not have huge computational resources available. Therefore, in order to cooperate with the aerospace industry it needs a flow solver suitable for use on limited hardware resources while still delivering a solution in a reduced time. The PUMI code was developed to address these requirements. Also, in order to better integrate the solver with the rest of CIMNE's in-house developed software a finite element approach had to be chosen (due to the ongoing effort to integrate all of CIMNE's codes into a common development framework [1] which is specially suited for finite element applications). To achieve optimum performance and reduce the memory requirements an edge based data structure was selected. As it shall be shown, this choice results in a numerical scheme which bear a remarkable resemblance to a finite volume formulation. This has the added advantage to enable use of many are well tested techniques originating from the finite volume community. To take advantage of the recent crop of multi-core CPUs available in desktop computers the code has been extensively parallelized using the Open-MP compiling directives.

The first section of this paper explains the basic equations and their weak formulation suitable for use with the finite element method as well as the changes in the traditional formulation brought about by the edge-based data structure. Next the implementation of the algorithm shall be described, including details abut the stabilization techniques, convergence acceleration and code optimization. Finally, some real life application examples will be shown to illustrate the capabilities of the software.

2. BASIC EQUATIONS

The starting point is the Euler equation set formulated in an Eulerian (space fixed) reference frame and written using conservative variables (in order to obtain well placed shocks)

$$\frac{\partial \mathbf{\Phi}}{\partial t} + \frac{\partial \mathbf{F}_k}{\partial x_k} = 0 \quad for \quad k = 1...3 \tag{2.1}$$

with $\mathbf{\Phi}$ being the vector of conservative variables and $\mathbf{F_k}$ the convective flux along the k^{th} direction:

$$\mathbf{\Phi} = \begin{bmatrix} \rho \\ U_1 \\ U_2 \\ U_3 \\ e \end{bmatrix} \quad \mathbf{F}_i = \begin{bmatrix} U_i \\ u_i U_1 + p\delta_{i1} \\ u_i U_2 + p\delta_{i2} \\ u_i U_3 + p\delta_{i3} \\ u_i h \end{bmatrix} \tag{2.2}$$

with δ_{ij} being the Kronecker delta. The state vector contains the density, momentum ($U_i = \rho u_i$) and total energy (internal plus kinetic) per unit volume of the fluid. Assuming the fluid behaves like an ideal gas the expression for the total energy, enthalpy and equation of state are

$$e = \rho\left(c_v T + \frac{u^2}{2}\right) \qquad h = e + p = \rho\left(c_p T + \frac{u^2}{2}\right) \qquad p = \rho RT \qquad R = c_p - c_v \tag{2.3}$$

In order solve the equations a suitable set of boundary conditions is needed. For solid walls the slip condition (velocity parallel to the wall) is enforced

$$\mathbf{u}(\mathbf{x}) \cdot \mathbf{n} = 0 \quad \forall \quad \mathbf{x} \in \Sigma_S \tag{2.4}$$

where $\mathbf{\Sigma_S}$ denotes the solid boundary. Some far field conditions are also needed, which depend on whether or not the flow is supersonic

$$\begin{aligned} &\mathbf{\Phi} \to \mathbf{\Phi}_\infty \text{ for } \mathbf{x} \to \infty \text{ if } M_\infty < 1 \\ &\mathbf{\Phi} \to \mathbf{\Phi}_\infty \text{ for } \mathbf{x} \to \infty^- \text{ if } M_\infty \geq 1 \end{aligned} \tag{2.5}$$

where the symbol ∞^- indicates the upstream part of the far field. In order to achieve a well conditioned behaviour and obtain more general results it is convenient to write the equations in non-dimensional form. By choosing the following variables

$$\begin{aligned} &\tilde{t} = \frac{t\, c_\infty}{L} \qquad \tilde{x}_i = \frac{x_i}{L} \qquad \tilde{u}_i = \frac{u_i}{c_\infty} \qquad \tilde{U}_i = \frac{u_i}{\rho_\infty c_\infty} \\ &\tilde{p} = \frac{p}{\rho_\infty c_\infty^2} \quad \tilde{e} = \frac{e}{\rho_\infty c_\infty^2} \qquad \tilde{h} = \frac{h}{\rho_\infty c_\infty^2} \qquad \tilde{T} = \frac{TR}{c_\infty^2} \end{aligned} \tag{2.6}$$

the equations may be recast as

$$\frac{\partial \tilde{\mathbf{\Phi}}}{\partial \tilde{t}} + \frac{\partial \tilde{\mathbf{F}}_k}{\partial \tilde{x}_k} = 0 \quad for \quad k = 1...3 \tag{2.7}$$

In (2.6) L represents the characteristic length of the problem and ρ_∞ and a_∞ denote the density and speed of sound ($a^2 = \gamma RT$) of the undisturbed fluid (far away upstream of the solid body). The non-dimensional vectors of unknowns and fluxes become

$$\tilde{\mathbf{\Phi}} = \begin{bmatrix} \tilde{\rho} \\ \tilde{U}_1 \\ \tilde{U}_2 \\ \tilde{U}_3 \\ \tilde{e} \end{bmatrix} \quad \tilde{\mathbf{F}}_i = \begin{bmatrix} \tilde{U}_i \\ \tilde{u}_i\tilde{U}_1 + \tilde{p}\delta_{i1} \\ \tilde{u}_i\tilde{U}_2 + \tilde{p}\delta_{i2} \\ \tilde{u}_i\tilde{U}_3 + \tilde{p}\delta_{i3} \\ \tilde{u}_i\tilde{h} \end{bmatrix} \tag{2.8}$$

With this change of variables the thermodynamic relationships for the ideal gas now read:

$$\tilde{e} = \tilde{\rho}\left(\frac{\tilde{T}}{\gamma - 1} + \frac{\tilde{u}^2}{2}\right) \qquad \tilde{h} = \tilde{\rho}\left(\frac{\gamma}{\gamma - 1}\tilde{T} + \frac{\tilde{u}^2}{2}\right) \qquad \tilde{p} = \tilde{\rho}\tilde{T} \qquad \gamma = \frac{c_p}{c_v} \tag{2.9}$$

The far field conditions now become much simpler

$$\tilde{\mathbf{\Phi}}_\infty = \begin{bmatrix} 1 \\ M_\infty^X \\ M_\infty^Y \\ M_\infty^Z \\ \dfrac{1}{\gamma(\gamma - 1)} + \dfrac{M_\infty^2}{2} \end{bmatrix} \tag{2.10}$$

It is easy to show that the far field pressure and temperature are both equal to $\frac{1}{\gamma}$. The non-dimensional solution is therefore a function of the Mach number, the specific heat ratio and the shape of the solid body only. The non-dimensional parameters have the additional advantage of being usually close to unity, thus yielding a better conditioned system. From this point on, for the sake of clarity, the tilde will be dropped from the non-dimensional variables. Unless otherwise stated, the non-dimensional form of the expressions shall be assumed.

To develop the finite element approximation to the equation set (2.7) we start with the weak form of the differential equations [2]. Let $\boldsymbol{W}$ be a generic test function defined inside the flow domain. The weak form of the flux balance statement reads:

$$\int_\Omega W(\mathbf{x})\left(\frac{\partial \mathbf{\Phi}}{\partial t} + \frac{\partial \mathbf{F}_k}{\partial x_k}\right) d\Omega = 0 \ \ \forall \ W \tag{2.11}$$

as long as (2.11) holds for any $\boldsymbol{W}$, both forms are equivalent. Next we use the Galerkin method to build an approximate solution $\tilde{\mathbf{\Phi}}$ which is a linear combination of the nodal values and the interpolating functions (N_j).

$$\begin{aligned} \tilde{\mathbf{\Phi}}(\mathbf{x}) &= N_j(\mathbf{x})\tilde{\mathbf{\Phi}}(\mathbf{x}^j) = N_j\,\tilde{\mathbf{\Phi}}^j \\ W(\mathbf{x}) &= N_i(\mathbf{x}) \end{aligned} \tag{2.12}$$

In (2.12) summation is assumed over the repeated index and the supra-index denotes nodal values. For the particular case of the finite element method the interpolation functions are the shape functions which are defined in an element-by-element basis. They

have the property

$$N_i(\mathbf{x}^j) = \delta_{ij} \tag{2.13}$$

The semi-discrete form of (2.11) becomes

$$\int_\Omega N_i \left(N_j \dot{\tilde{\mathbf{\Phi}}}^j + \frac{\partial \tilde{\mathbf{F}}_k}{\partial x_k} \right) d\Omega = 0 \;\; for \;\; i = 1...n_{node} \tag{2.14}$$

where the dot over the state vector indicates the time derivative. In (2.14) there are as many equations as unknowns (nodes) therefore the system can be solved for the nodal values of the approximate solution (provided the boundary conditions are correctly enforced). The integrals in (2.14) are evaluated using Gauss quadrature; in order to evaluate the fluxes at the interpolation points the consistent approach would be:

$$\tilde{\mathbf{F}}_k^{IP} = \tilde{\mathbf{F}}_k \left(\tilde{\mathbf{\Phi}}^{IP} \right) = \tilde{\mathbf{F}}_k \left(N_j(\mathbf{x}^{IP}) \tilde{\mathbf{\Phi}}^j \right) \tag{2.15}$$

in order to increase the efficiency of the algorithm we shall assume that it is acceptable to interpolate the fluxes inside the elements from their nodal values [3]. This is equivalent to using the Lobato quadrature for the fluxes and does not affect the final result in any significant way

$$\tilde{\mathbf{F}}_k \cong N_j \tilde{\mathbf{F}}_k(\mathbf{x}^j) = N_j \tilde{\mathbf{F}}_k^j \tag{2.16}$$

With this change (2.14) is transformed into

$$\int_\Omega N_i \left(N_j \dot{\tilde{\mathbf{\Phi}}}^j + \frac{\partial N_j}{\partial x_k} \left(\tilde{\mathbf{F}}_k^j \right) \right) d\Omega = 0 \;\; for \;\; i = 1...n_{node} \tag{2.17}$$

Moving the flux term to the RHS and using matrix notation, the time derivative of the state vector can be written as

$$\begin{aligned} \dot{\tilde{\mathbf{\Phi}}}^j &= \mathbf{M}^{-1} \mathbf{r} \\ \mathbf{M} &= \int_\Omega N_i \, N_j \, d\Omega \\ \mathbf{r} &= -\int_\Omega N_i \frac{\partial N_j}{\partial x_k} d\Omega \, \tilde{\mathbf{F}}_k^j \end{aligned} \tag{2.18}$$

where **M** is the consistent finite element mass matrix. To avoid solving a linear system of equations at each time step the consistent mass matrix is usually replaced by its diagonal (lumped) counterpart such that

$$\mathbf{M}_{ij}^d = \delta_{ij} \sum_j \mathbf{M}_{ij} \tag{2.19}$$

thus rendering the process of matrix inversion trivial. To achieve better computational efficiency the residual vector (**r**) must be rearranged. Let us split the residual of the ***i***-th equation in two parts:

$$\mathbf{r}^{\mathbf{i}} = -\sum_{j\neq i}\int_{\Omega} N_i\, N_{j,k}\, d\Omega\, \tilde{\mathbf{F}}_k^j - \int_{\Omega} N_i\, N_{i,k}\, d\Omega\, \tilde{\mathbf{F}}_k^i \quad where \quad N_{j,k} = \frac{\partial N_j}{\partial x_k} \tag{2.20}$$

in the expression above there is no sum over index ***i***, and the sum over index ***j*** extends to all the possible values except for ***i***. From this point on, to keep the notation compact, the summation sign will be omitted, but the remark still holds (i.e., ***i*** does not enter the sum over ***j***). Integrating (2.20) by parts yields

$$\mathbf{r}^{\mathbf{i}} = \int_{\Omega} N_{i,k}\, N_j\, d\Omega\, \tilde{\mathbf{F}}_k^{ij} - \int_{\Omega} N_{i,k}\, N_j\, d\Omega\, \tilde{\mathbf{F}}_k^i - \int_{\Gamma} N_i\, N_j\, n_k d\Gamma\, \tilde{\mathbf{F}}_k^j - \frac{1}{2}\int_{\Gamma} N_i\, N_i\, n_k d\Gamma\, \tilde{\mathbf{F}}_k^i \tag{2.21}$$

where we have defined the numerical interface flux as

$$\tilde{\mathbf{F}}_k^{ij} = \tilde{\mathbf{F}}_k^i + \tilde{\mathbf{F}}_k^j \tag{2.22}$$

Remark that (2.22) is twice the average of the nodal values. The expression (2.21) must now be symmetrised to achieve the full benefit of the edge data structure

$$\begin{aligned} \mathbf{r}^{\mathbf{i}} = {} & \frac{1}{2}\int_{\Omega}\left(N_{i,k}\, N_j - N_i\, N_{j,k}\right) d\Omega\, \tilde{\mathbf{F}}_k^{ij} + \int_{\Gamma} N_i\, N_j\, n_k d\Gamma\, \tilde{\mathbf{F}}_k^{ij} - \\ & - \int_{\Omega} N_{i,k}\, N_j\, d\Omega\, \tilde{\mathbf{F}}_k^i - \int_{\Gamma} N_i\, N_j\, n_k d\Gamma\, \tilde{\mathbf{F}}_k^j - \frac{1}{2}\int_{\Gamma} N_i\, N_i\, n_k d\Gamma\, \tilde{\mathbf{F}}_k^i \end{aligned} \tag{2.23}$$

Making use of the shape function property $N_i = 1 - \sum_{j\neq i} N_j$ and after some algebraic manipulation the residual can be written as

$$\begin{aligned} \mathbf{r}^{\mathbf{i}} &= d_k^{ij}\tilde{\mathbf{F}}_k^{ij} + b_k^{ij}\tilde{\mathbf{F}}_k^{ij} + c_k^i\tilde{\mathbf{F}}_k^i \\ d_k^{ij} &= \frac{1}{2}\int_{\Omega}\left(N_{i,k}\, N_j - N_i\, N_{j,k}\right) d\Omega \\ b_k^{ij} &= -\frac{1}{2}\int_{\Gamma} N_i\, N_j\, n_k d\Gamma \\ c_k^i &= -\int_{\Gamma} N_i\, N_i\, n_k d\Gamma \end{aligned} \tag{2.24}$$

Note that the shape functions of node ***i*** are zero over any element not containing ***i***; therefore the integrals in (2.24) need only computed for pairs ***ij*** of nodes sharing the same elements. Any such pair is called an edge. When the element is a simplex (i.e. triangle in 2D or tetrahedron in 3D) the computational edges coincide with the geometric edges; otherwise aditional internal computational edges shall appear. Also, remark the ***d*** coefficients are antisymmetric, meaning that only half of them need be stored. Moreover, the ***b*** and ***c*** terms are zero for any interior edge so the storage requirements are greatly reduced [4].

The ***d*** coefficients being antisymmetric implies that the scheme (2.18) is conservative. Indeed, the net contribution to the residual is zero for any internal (i.e. not containing boundary nodes) edge ***e***

$$\mathbf{r}_e^{\mathbf{i}} + \mathbf{r}_e^{\mathbf{j}} = d_k^{ij}\tilde{\mathbf{F}}_k^{ij} + d_k^{ji}\tilde{\mathbf{F}}_k^{ji} = d_k^{ij}\tilde{\mathbf{F}}_k^{ij} - d_k^{ij}\tilde{\mathbf{F}}_k^{ij} = \mathbf{0} \tag{2.25}$$

Using the form (2.24) of the residual entails important savings with regards to the number of operations and memory accesses [5]. Also note that for interior edges where only the ***d*** terms remain the expression for the residual is quite similar to what would be obtained from a cell-centered finite volume scheme whose cell centroids were located at the nodes of the FE mesh. Therefore, most techniques developed for use with unstructured finite volume solvers can be also applied to the edge-based finite element formulation.

3. CONVECTIVE STABILIZATION

The basic Galerkin formulation, being equivalent to a second order finite difference scheme suffers from the same shortcomings. In particular, it is susceptible to the odd-even decoupling phenomenon which allows spurious solutions to contaminate the results [6]. To prevent the occurrence of non-physical solutions some for of stabilization must be added to the basic Galerkin scheme. As the Euler equation set is hyperbolic, it represents the propagation of waves. Thus, the upwnding techniques developed for the wave equation can be successfully applied to the Euler equations. This is achieved using a change of variables to uncouple the set (2.7). Let us define the flux Jacobians ($\mathbf{A_i}$) such that

$$\frac{\partial \mathbf{F}_i}{\partial x_i} = \mathbf{A}_i \frac{\partial \mathbf{\Phi}}{\partial x_i} \tag{3.1}$$

The Jacobians are therefore the derivatives of the flux vectors with respect to the state variables. Incidentally, the flux vectors are homogeneous functions of the conservative variables so it is also possible to write

$$\mathbf{F}_i = \mathbf{A}_i \mathbf{\Phi} \tag{3.2}$$

Substituting (3.1) into (2.7) yields the quasi-linear form of the Euler equations

$$\frac{\partial \mathbf{\Phi}}{\partial t} + \mathbf{A}_k \frac{\partial \mathbf{\Phi}}{\partial x_k} = 0 \tag{3.3}$$

Note that the equations (3.3) are not linear as the Jacobians are functions of $\mathbf{\Phi}$. However, if we consider only small fluctuations around an equilibrium state it is acceptable to linearize the behaviour and assume the Jacobians are constant. For the sake of simplicity, the following discussion will assume a 1D flow along the ***x*** direction, the results still hold for any arbitrary direction. For the 1D case (3.3) reduces to

$$\frac{\partial \mathbf{\Phi}}{\partial t} + \mathbf{A} \frac{\partial \mathbf{\Phi}}{\partial x} = 0 \tag{3.4}$$

the index is dropped as there is only one space direction. The equation system being hyperbolic means the eigenvalues of the Jacobian are always real. It is possible to build a complete set of right eigenvectors [9]

$$\mathbf{A}_{il}\mathbf{R}_{lj} = \lambda_j \mathbf{R}_{ij} \tag{3.5}$$

so the Jacobian can be written as

$$\mathbf{A} = \mathbf{R}\mathbf{\Lambda}\mathbf{R}^{-1} \quad \textit{where} \quad \mathbf{\Lambda}_{ij} = \delta_{ij}\lambda_i \tag{3.6}$$

the matrix $\mathbf{\Lambda}$ is diagonal and contains the eigenvalues of the Jacobian. If we multiply (3.4) by $\mathbf{R^{-1}}$ we have

$$\mathbf{R}^{-1}\frac{\partial \mathbf{\Phi}}{\partial t}+\mathbf{R}^{-1}\mathbf{A}(\mathbf{R}\mathbf{R}^{-1})\frac{\partial \mathbf{\Phi}}{\partial x}=0 \tag{3.7}$$

Using the eigenvector decomposition of $\mathbf{A}$ and with the change of variable $\boldsymbol{\varphi}=\mathbf{R}^{-1}\mathbf{\Phi}$ the system becomes

$$\frac{\partial \boldsymbol{\varphi}}{\partial t}+\mathbf{\Lambda}\frac{\partial \boldsymbol{\varphi}}{\partial x}=0 \tag{3.8}$$

as the matrix $\mathbf{\Lambda}$ is diagonal, all the equations in (3.8) are uncoupled. The evolution of the component of $\boldsymbol{\varphi}$ (called characteristic variables) is described by the wave equation. The wave speeds are the eigenvalues of the Jacobian. There are three distinct eigenvalues:

$$\begin{aligned}\lambda_1 &= u+c\\ \lambda_2 &= u-c\\ \lambda_3 &= u\end{aligned} \tag{3.9}$$

The first two eigenvalues represent acoustic perturbations and the third one entropy waves. In multi-dimensional cases the third eigenvalue is multiple due to the existence of vorticity waves which also travel with the same speed as the fluid.

A well known technique to stabilize the numerical behaviour of the wave equation is upwinding. Switching from central to backward differencing for the convective term solves the decoupling problem [7]. This solution can be applied to the characteristic variables in (3.8), however, it is important to stress that the correct upwinding direction is not the same for all components of the solution. Indeed, unless the flow is supersonic (***u*>*c*)** not all the eigenvalues in (3.9) have the same sign (i.e. not all the waves travel in the same direction). Therefore, the upwinding direction must be chosen in a component by component basis. In the context of an edge-based finite element solver, upwinding is done by replacing the interface fluxes with their value at the upwind side of the edge. If we define a characteristic flux in (3.8) as $\mathbf{f}_k=\lambda_k\boldsymbol{\varphi}_k$ (where the supra-index k indicates the specific characteristic component) we can recast the system as

$$\frac{\partial \boldsymbol{\varphi}_k}{\partial t}+\frac{\partial \mathbf{f}_k}{\partial x}=0 \tag{3.10}$$

which has the same form as (2.7) enabling the use of the discretization (2.24). For an internal edge we have

$$\mathbf{r}_k^i=d^{ij}\mathbf{f}_k^{ij}=d^{ij}(\mathbf{f}_k^i+\mathbf{f}_k^j) \tag{3.11}$$

Now, let us assume that the x axis points from node **i** to node **j**. Depending on the sign of the wave propagation the upwind approximation of (3.11) is

$$\begin{aligned}\mathbf{r}_k^i\Big|_{Upwind} &= d^{ij}(2\mathbf{f}_k^i)=d^{ij}\left(\mathbf{f}_k^{ij}-(\mathbf{f}_k^j-\mathbf{f}_k^i)\right) \quad if \quad \lambda_k>0\\ \mathbf{r}_k^i\Big|_{Upwind} &= d^{ij}(2\mathbf{f}_k^j)=d^{ij}\left(\mathbf{f}_k^{ij}+(\mathbf{f}_k^j-\mathbf{f}_k^i)\right) \quad if \quad \lambda_k<0\end{aligned} \tag{3.12}$$

This can be rewritten as

$$\begin{aligned} \mathbf{r}_k^i\big|_{Upwind} &= d^{ij}\left(\mathbf{f}_k^{ij} - \lambda_k(\boldsymbol{\varphi}_k^j - \boldsymbol{\varphi}_k^i)\right) \quad if \quad \lambda_k > 0 \\ \mathbf{r}_k^i\big|_{Upwind} &= d^{ij}\left(\mathbf{f}_k^{ij} + \lambda_k(\boldsymbol{\varphi}_k^j - \boldsymbol{\varphi}_k^i)\right) \quad if \quad \lambda_k < 0 \end{aligned} \tag{3.13}$$

Note that both expressions in (3.13) can be merged [6] into

$$\mathbf{r}_k^i\big|_{Upwind} = d^{ij}\left(\mathbf{f}_k^{ij} - \left|\lambda_k\right|(\boldsymbol{\varphi}_k^j - \boldsymbol{\varphi}_k^i)\right) \tag{3.14}$$

If we define the matrix $\left|\boldsymbol{\Lambda}\right|$ as a matrix containing in its diagonal the absolute value of the eigenvalues, the discrete upwind approximation to (3.10) becomes

$$\mathbf{r}^i\big|_{Upwind} = d^{ij}\left(\mathbf{f}^{ij} - \left|\boldsymbol{\Lambda}\right|(\boldsymbol{\varphi}^j - \boldsymbol{\varphi}^i)\right) \tag{3.15}$$

Where the index ***k*** has been dropped, as (3.15) applies to the complete state vector (all characteristic variables). Now the change from conservative to characteristic variables can be reverted yielding

$$\mathbf{r^i}\big|_{Upwind} = d^{ij}\left(\mathbf{F^{ij}} - \mathbf{R}\left|\boldsymbol{\Lambda}\right|\mathbf{R^{-1}}(\boldsymbol{\Phi^j} - \boldsymbol{\Phi^i})\right) \tag{3.16}$$

the matrix $\mathbf{R}\left|\boldsymbol{\Lambda}\right|\mathbf{R^{-1}} = \left|\mathbf{A}\right|$ is called the positive Jacobian.. It is obtained by making all the eigenvalues of the flux Jacobian positive.

At this point, it is important to remember the discussion above assumed linearized behaviour thus enabling the use of a constant flux Jacobian. In real-life, however, obtaining a matrix **A** such that $\mathbf{F^j} - \mathbf{F^i} = \mathbf{A}(\boldsymbol{\Phi^j} - \boldsymbol{\Phi^i})$ where states ***i*** and ***j*** can be vastly different involves solving a Riemann problem. While this approach is feasible, a heavy computational cost penalty is involved. As an alternative, the PUMI solver uses Roe's approximate Riemann solver [10]

$$\mathbf{F^j} - \mathbf{F^i} \approx \tilde{\mathbf{A}}_{Roe}(\boldsymbol{\Phi^j} - \boldsymbol{\Phi^i}) \quad where \quad \tilde{\mathbf{A}}_{Roe} = \mathbf{A}(\tilde{\boldsymbol{\Phi}}_{Roe}^{ij}) \tag{3.17}$$

where $\tilde{\boldsymbol{\Phi}}_{Roe}^{ij}$ denotes the approximate Roe intermediate state which can be easily computed from states ***i*** and ***j***. Thus, (3.16) is replaced by

$$\mathbf{r^i}\big|_{Upwind} = d^{ij}\left(\mathbf{F^{ij}} - \left|\tilde{\mathbf{A}}_{Roe}\right|(\boldsymbol{\Phi^j} - \boldsymbol{\Phi^i})\right) \tag{3.18}$$

It is possible to compute the positive Jacobian in (3.18) using the eigenvalues (3.9) and the corresponding eigenvector set (for which an explicit expression is available, see [10]). However the cost associated with this operation is very high. A more efficient alternative is to directly compute the product $\left|\tilde{\mathbf{A}}_{Roe}\right|(\boldsymbol{\Phi^j} - \boldsymbol{\Phi^i})$ without evaluating $\left|\tilde{\mathbf{A}}_{Roe}\right|$. There are very efficient algorithms to achieve this goal which can be found in [11] and [12].

It is worth mentioning that at stagnation or sonic points some eigenvalues of the Jacobian matrix vanish [13]. This causes in turn a loss of stabilization for some characteristic components of the solution, with the potential for non-physical spurious solutions. This problem is addressed in PUMI by setting a lower bound on the absolute value of the eigenvalues according to

$$\left|\mathbf{\Lambda}\right|_{ii} = \max(\left|\lambda_i\right|, \alpha c) \tag{3.19}$$

with the α parameter in (3.19) being user-selectable. Setting α to 0,2 is usually enough to achieve god results. Up to this point, the discussion has dealt with the 1D version of the Euler equations. However, PUMI solves the 3D equation set, so additional care must be taken. The flux Jacobian for transport along an arbitrary direction (given by a unit vector **n)** can be obtained by linear combination of the Jacobians for the ***x***, ***y*** and ***z*** directions

$$\mathbf{A}^{\mathbf{n}} = \sum_k \mathbf{n}_k \mathbf{A}_k \tag{3.20}$$

because the equations are hyperbolic, the eigenvalues of (3.20) shall always be real, irrespective of the choice of **n** and the positive Jacobian can thus be found. To stabilize the 3D version of the equations a transport direction for each edge has to be defined so that the positive Jacobian can be calculated from (3.20). In PUMI the direction of choice is the edge itself, the unit vector for the edge connecting nodes ***i*** and ***j*** is

$$\mathbf{n}^{ij} = \frac{\mathbf{x}^j - \mathbf{x}^i}{\left\|\mathbf{x}^j - \mathbf{x}^i\right\|} \tag{3.21}$$

Once this direction has been defined, the stabilized contribution to the residual vector can be written as

$$\mathbf{r}^i\Big|_{Upwind} = d_k^{ij}\mathbf{F}_k^{ij} - d_k^{ij}\mathbf{n}_k^{ij}\left|\tilde{\mathbf{A}}_{Roe}^{\mathbf{n}^{ij}}\right|\Delta_{ij}\mathbf{\Phi} \tag{3.22}$$

where we introduced the edge difference $\Delta_{ij}\mathbf{\Phi} = \mathbf{\Phi}^j - \mathbf{\Phi}^i$.This is by no means the only possibility, other choices of the transport direction are equally suitable. A common choice is to make the **n** vector parallel to the d_k^{ij} coefficient so that the product $d_k^{ij}\mathbf{n}_k^{ij}$ is maximum [3].

The discretization (3.22) is stable, but being no longer centered it is of lower accuracy than the original form (2.24) (first order versus second order) [9]. To develop an efficient solver, at least second order space accuracy should be achieved over most of the flow domain. In PUMI this is accomplished by limiting the stabilizing term in (3.22) so that it tends to vanish in areas were the solution behaves smoothly. To keep the notation compact, let us define the artificial flux in (3.22) as

$$\mathbf{Fa}_k^{ij} = -\mathbf{n}_k^{ij}\left|\tilde{\mathbf{A}}_{Roe}^{\mathbf{n}^{ij}}\right|\Delta_{ij}\mathbf{\Phi} \tag{3.23}$$

To approach second order accuracy we cut back the artificial flux (3.23) by reducing the difference term

$$\mathbf{Fa}_k^{ij}\Big|_{\lim ited} = -\mathbf{n}_k^{ij}\left|\tilde{\mathbf{A}}_{Roe}^{\mathbf{n}^{ij}}\right|\left(\mathbf{\Phi}^{j-1/2} - \mathbf{\Phi}^{i+1/2}\right) \tag{3.24}$$

where $\mathbf{\Phi}^{j-1/2}$ and $\mathbf{\Phi}^{i+1/2}$ are two high order approximations of the solution at the interface. The limited value is expected to be smaller than (3.23) thus reducing the amount of artificial diffusion. When working with uniformly spaced grids obtaining the high order approximation of the interface solution is relatively easy [7]. On the other hand, when using an unstructured grid as is the case with finite element solvers the process of

reconstruction of the interface values is slightly more involved [8]. To take advantage of the techniques developed for structured meshes, two additional extrapolated states are defined for each edge. These correspond to virtual points ***i-1*** and ***j+1*** such that

$$\left.\begin{aligned}\mathbf{x}^{i-1} &= \mathbf{x}^{i} - \mathbf{u}^{ij} \\ \mathbf{x}^{j+1} &= \mathbf{x}^{j} + \mathbf{u}^{ij}\end{aligned}\right\} \quad where \quad \mathbf{u}^{ij} = \mathbf{x}^{j} - \mathbf{x}^{i} \tag{3.25}$$

these points need not coincide with any node of the mesh, in fact they may even lay outside the fluid domain when the edge is close to the boundaries. There are many different approaches to estimate the value of the solution at the virtual points [8]:

- By standard finite element interpolation in the element they lay in
- Using the value of the nearest node
- Using the nodal gradients of the solution

The last method is the choice in PUMI. When solving the NS equations (which the code is also capable of treating) the nodal gradients are used to calculate the diffusive fluxes. Therefore the information is readily available. This also simplifies the treatment of virtual nodes which lie outside the fluid domain. The process of derivative recovery is described in the implementation details section of this paper. From the second order centered approximation of the nodal derivative, the forward and backward extrapolated states can be calculated

$$\begin{aligned}\mathbf{\Phi}^{j+1} &\simeq \mathbf{\Phi}^{i} + 2\mathbf{u}^{ij} \cdot \nabla\mathbf{\Phi}^{j} \\ \mathbf{\Phi}^{i-1} &\simeq \mathbf{\Phi}^{j} - 2\mathbf{u}^{ij} \cdot \nabla\mathbf{\Phi}^{i}\end{aligned} \tag{3.26}$$

From this data two additional differences (backward and forward) can be defined

$$\Delta_i^- = \mathbf{\Phi}^{i} - \mathbf{\Phi}^{i-1} \qquad \Delta_j^+ = \mathbf{\Phi}^{j+1} - \mathbf{\Phi}^{j} \tag{3.27}$$

The high-order interface approximations are built using a variable order reconstruction scheme

$$\begin{aligned}\mathbf{\Phi}^{i+1/2} &= \mathbf{\Phi}^{i} + \frac{1}{4}\left[(1-k)\Delta_i^- + (1+k)\Delta_{ij}\right] \\ \mathbf{\Phi}^{j-1/2} &= \mathbf{\Phi}^{j} - \frac{1}{4}\left[(1-k)\Delta_j^+ + (1+k)\Delta_{ij}\right]\end{aligned} \tag{3.28}$$

The ***k*** parameter in (3.28) controls the order of extrapolation [3], some typical values are:

- ***k*** = -1 second-order full upwind scheme
- ***k*** = 0 From's scheme
- ***k*** = 1/3 third-order upwind scheme
- ***k*** = 1 three-point central difference scheme

By using the high order artificial flux (3.24) the accuracy of the scheme is improved. However, the high order solution is susceptible to non-physical oscillations in the neighbourhood of shocks or sharp gradients. To overcome this limitation the scheme should revert to (3.23) (i.e. $\mathbf{\Phi}^{i+1/2} \to \mathbf{\Phi}^{i}$ and $\mathbf{\Phi}^{j-1/2} \to \mathbf{\Phi}^{j}$) whenever discontinuities appear on the solution (in areas where the solution is not smooth only the first order scheme is free form oscillations, as stated by Godunov's theorem [7]). Sharp changes in the solution

are tracked in PUMI by measuring the local curvature of the solution. At each node a limiter is calculated based on the forward and backward differences. The limiter should ideally approach unity when both differences are similar and vanish when they are vastly different

$$\mathbf{l}^i = f(\Delta_i^-, \Delta_{ij}) \quad \mathbf{l}^j = f(\Delta_{ij}, \Delta_j^+) \tag{3.29}$$

Note the limiters are written as vectors, as they may be different for each component of the state vector. See the next section for details on the limiters available in PUMI. The expressions in (3.28) are modified to control the amount of extrapolation according to:

$$\begin{aligned} \mathbf{\Phi}^{i+1/2} &= \mathbf{\Phi}^i + \frac{\mathbf{l}^i}{4}\left[(\mathbf{1} - k\mathbf{l}^i)\Delta_i^- + (\mathbf{1} + k\mathbf{l}^i)\Delta_{ij}\right] \\ \mathbf{\Phi}^{j-1/2} &= \mathbf{\Phi}^j - \frac{\mathbf{l}^j}{4}\left[(\mathbf{1} - k\mathbf{l}^j)\Delta_j^+ + (\mathbf{1} + k\mathbf{l}^j)\Delta_{ij}\right] \end{aligned} \tag{3.30}$$

it is clear that the high order interface values revert to $\mathbf{\Phi}^i$ and $\mathbf{\Phi}^{j-1/2}$ when the limiters approach zero thus helping preserve the monotonicity of the solution.

4. TIME INTEGRATION

PUMI was developed to tackle steady problems, therefore the main goal of the time marching algorithm is to improve the robustness of the solver and to speed up convergence as much as possible, without much concern about the time accuracy. An explicit multi-stage Runge-Kutta scheme is chosen in order to increase the allowable time step an increase robustness of the solution process [9]. To advance (2.18), for each time step some intermediate states $\mathbf{\Psi_j}$ are created such that:

$$\begin{aligned} &\mathbf{\Psi}_0 = \tilde{\mathbf{\Phi}}\Big|_{t_i} \\ &\mathbf{\Psi}_j = \mathbf{\Psi}_0 + \theta_j \left(t_{i+1} - t_i\right)\mathbf{M}^{-1}\mathbf{r}(\mathbf{\Psi}_{j-1}) \\ &\tilde{\mathbf{\Phi}}\Big|_{t_{i+1}} = \mathbf{\Psi}_n \\ &\theta_n = 1 \end{aligned} \tag{4.1}$$

with n being the number of stages of the scheme. Each particular scheme is characterized by a choice of n and θ_i. Two common choices which provide reasonable robustness are:

- 3-stage scheme with $\theta_1 = \frac{3}{5} \quad \theta_2 = \frac{3}{5} \quad \theta_3 = 1$
- 4-stage scheme with $\theta_1 = \frac{1}{4} \quad \theta_2 = \frac{1}{3} \quad \theta_3 = \frac{1}{2} \quad \theta_4 = 1$

Under many circumstances the 4-stage scheme provides a better balance between allowable step size and cost per step. When computing the residual vector in (4.1) it is not necessary to update the artificial fluxes at each stage. Doing so would increase substantially the cost per stage while yielding only a marginal increase in robustness. Therefore the stabilization terms are only computed at the first stage under most circumstances. The scheme presented in (4.1) is explicit and therefore only conditionally stable. There is an upper limit on the allowable time step. For a purely convective

problem, the stability limit for a given element could be calculated as:

$$\Delta t_{el} = C\frac{h}{\lambda_{\max}} \tag{4.2}$$

where C denotes the allowable Courant number, h is some measure of the element size and λ_{max} is the maximum eigenvalue of the Jacobian matrix. If a time accurate solution is sought, a global time step equal to the minimum allowable step for all the elements of the mesh. However, when solving steady problems a local time stepping approach is acceptable [15]. This has the advantage of propagating the solution much faster, thus achieving convergence in a smaller number of steps. The local time stepping is implemented at the nodal level. For each node in the mesh a nodal size is calculated as

$$h_{nod}^{i} = \min(h_{el}^{j}) \;/\; i \in el^{j} \tag{4.3}$$

that is, the size of node i is the minimum of the sizes of all the elements to which i belongs. The element size is taken as the minimum height of the element. For example, consider a tetrahedral element:

$$h_{el}^{i} = \frac{3\Omega^{i}}{\max(A_{j}^{i})} \qquad j = 1,...,4 \tag{4.4}$$

with Ω^{i} being the element volume and A^{i}_{j} the areas of its faces. This choice of size estimate increases robustness when high aspect ratio elements are present in the mesh. The allowable step at each node is then calculated as

$$\Delta t_{nod}^{i} = C\frac{h_{nod}^{i}}{\left\|\mathbf{v}^{i}\right\| + c^{i}} \tag{4.5}$$

The value of C in (4.5) is, in principle, global. It is then used for all the nodes of the mesh. However, in areas of low speed flow due to the increased stiffness of the equations problems with oscillations can arise requiring a small value of C. To avoid degrading the overall convergence of the mesh the user is able to specify two different values of the Courant number, where the smaller will be used in areas where the Mach number is below some adjustable threshold. To increase the allowable time step an implicit residual smoothing is used. A Laplacian smoothing is included to extend the support of the interpolation functions, yielding an increase in the allowable Courant number.

$$\bar{\mathbf{r}}^{i} = \mathbf{r}^{i} + \varepsilon\sum_{j}\left(\bar{\mathbf{r}}^{j} - \bar{\mathbf{r}}^{i}\right) \quad \text{for all } j \text{ connected to } i \tag{4.6}$$

solving exactly for the smoothed residual $\bar{\mathbf{r}}^{i}$ would require solving a linear system of equations negating the advantages of the explicit scheme. However, the smoothing is only used as a means to achieve faster convergence and has no effect on the steady state solution. Therefore, a rough approximation to the value of $\bar{\mathbf{r}}^{i}$ is enough to obtain the desired result. The approximate value of the smoothed residual is computed by means of a Jacobi iteration scheme:

$$\overline{\mathbf{r}}_n^{\,i} = \frac{\mathbf{r}^i + \varepsilon \sum_j \overline{\mathbf{r}}_{n-1}^{\,j}}{1 + \varepsilon \sum_j 1} \tag{4.7}$$

in common applications a good result can be obtained by running two passes of the scheme (4.7) and setting the smoothing coefficient $\boldsymbol{\varepsilon}$ to 0,1. The residual smoothing need not be applied at each stage of the Runge-Kutta scheme. For example, in the case of the 4 stage scheme smoothing just the first and third stages is usually enough to produce a twofold increase in the allowable Courant number.

6. ADDITIONAL DETAILS OF IMPLEMENTATION

Choice of limiters

Most simulations in PUMI are run using the Van Albada limiter [3] which yields good results under normal circumstances

$$\mathbf{l}^i = \max\left[\frac{2\Delta_i^- \Delta_{ij} + \varepsilon}{\left(\Delta_i^-\right)^2 + \left(\Delta_{ij}\right)^2 + \varepsilon},\ 0 \right] \tag{5.1}$$

where $\boldsymbol{\varepsilon}$ is a small number which prevents division by zero when the flow is uniform ($\boldsymbol{\varepsilon} \sim 10^{-5}$). For cases where achieving convergence is difficult there is the option to switch to a limiter based on the minimum modulus difference to increase robustness (albeit at the expense of a more diffusive solution) [5]

$$\mathbf{l}^i = \begin{cases} \operatorname{sgn}(\Delta_i^-) \dfrac{\min\left[\left|\Delta_i^-\right|, \left|\Delta_{ij}\right|\right]}{\max\left[\left|\Delta_i^-\right|, \left|\Delta_{ij}\right|\right]} & \textit{if}\ \ \operatorname{sgn}(\Delta_i^-) = \operatorname{sgn}(\Delta_{ij}) \\ 0 & \textit{otherwise} \end{cases} \tag{5.2}$$

when this approach is chosen there is the possibility to speed-up the computation using a simpler form of the MINMOD limiter by replacing (3.24) with

$$\mathbf{Fa}_k^{ij}\Big|_{\lim ited} = -\mathbf{n}_k^{ij} \left| \tilde{\mathbf{A}}_{Roe}^{\mathbf{n}^{ij}} \right| \Delta_{ij}^{\lim ited}$$

$$\Delta_{ij}^{\lim ited} = \begin{cases} \Delta_{ij} - \max\left[\min\left(\Delta_i^-, \Delta_j^+\right), 0 \right] & \textit{if}\ \ \Delta_{ij} > 0 \\ \Delta_{ij} - \min\left[\max\left(\Delta_i^-, \Delta_j^+\right), 0 \right] & \textit{otherwise} \end{cases} \tag{5.3}$$

Use of (5.3) bypasses the calculation of (3.30) thus improving performance slightly.

Sometimes a situation is found where the computation tends to diverge during the first steps of the analysis. This is often the case when the geometry contains non streamlined features for which the initial conditions are very far from the steady state. To improve robustness the user is offered to choice to run a certain number of time steps using the first order scheme (i.e. setting all the limiters to zero) at the beginning of the simulation. This is usually enough to reach conditions close enough to the steady state so that the high order scheme converges smoothly.

Towards the end of the simulation, when the system is very close to the steady state convergence might me hampered by the continuous change of the limiters. The

nonlinearity they introduce slows down the rate of convergence while no longer providing a tangible gain in accuracy [8]. To overcome this limitation there is the option to freeze the limiters once the solution is close enough to the steady state. This point is usually chosen as the step where the residual has decreased three orders of magnitude with respect to its initial value. The user also has the option to freeze the limiters in a progressive way, decreasing the update frequency from once per step to completely fixed along the simulation. Decreasing the limiter update frequency also has the added benefit of reducing the computational cost of each step.

For edges having a node on the solid boundary the process of extrapolation is further complicated by the fact that one virtual node usually lies outside the fluid domain. As the information used to recover the solution at the virtual node is one-sided a loss in accuracy is expected. This may eventually cause the limiters on the boundary nodes to have lower values than desirable thus increasing the amount of artificial diffusion. The effect can be detected on the solution as an spurious curvature of the Mach lines close to the solid wall, specially when the mesh is coarse. In order to mitigate the problem there is the choice of setting the limiters of all the affected nodes to unity. This way, the artificial flux is calculated using information from the interior node of the edge, where the loss of accuracy is much smaller.

Derivative Recovery

To recover the nodal values of the solution gradient it is assumed that the gradient field can be approximated using the shape functions and the nodal values

$$\frac{\partial \tilde{\mathbf{\Phi}}}{\partial x_k} = \nabla_k \tilde{\mathbf{\Phi}} = N_j(\mathbf{x}) \left. \frac{\partial \tilde{\mathbf{\Phi}}}{\partial x_k} \right|_{\mathbf{x}^j} = N_j \nabla_k \tilde{\mathbf{\Phi}}^j \qquad k = 1,2,3 \tag{5.4}$$

On the other hand, the elemental values of the gradient can also be obtained trough derivation of the shape functions

$$\left. \frac{\partial \tilde{\mathbf{\Phi}}}{\partial x_k} \right|_{el} = \frac{\partial N_j}{\partial x_k} \tilde{\mathbf{\Phi}}^j = \nabla_k N_j \, \tilde{\mathbf{\Phi}}^j \tag{5.5}$$

by equating the fields (5.4) and (5.5) in the weak sense we have

$$\int_\Omega W N_j \nabla_k \tilde{\mathbf{\Phi}}^j \, d\Omega = \int_\Omega W \nabla_k N_j \, \tilde{\mathbf{\Phi}}^j \, d\Omega \;\; \forall \;\; W \tag{5.6}$$

with $\boldsymbol{W}$ being a generic test function. Using the Galerkin method (i.e. setting $\boldsymbol{W=N_i}$) yields

$$\int_\Omega N_i N_j \nabla_k \tilde{\mathbf{\Phi}}^j \, d\Omega = \int_\Omega N_i \nabla_k N_j \, \tilde{\mathbf{\Phi}}^j \, d\Omega \;\; \forall \;\; W \tag{5.7}$$

which is a linear system for the unknowns $\nabla_k \tilde{\mathbf{\Phi}}^j$.

$$\begin{aligned} &\mathbf{M} \nabla_k \tilde{\mathbf{\Phi}} = \mathbf{D} \tilde{\mathbf{\Phi}}^j \\ &\mathbf{D} = \int_\Omega N_i \nabla_k N_j \, d\Omega \end{aligned} \tag{5.8}$$

note that RHS of (5.8) can also assembled by edges as it is similar to the RHS of (2.18).

The expression above, which uses the consistent mass matrix, calls for the solution of a linear system. Two choices are available in PUMI:

- Use of the lumped mass matrix to solve (5.8)
- Solution of the system trough an iterative algorithm which uses the lumped mass matrix as an approximation of the consistent one:

When the second method is chosen the solution scheme proceeds as follows:

$$\begin{aligned} &\textit{Given} \quad \mathbf{M}\mathbf{x} = \mathbf{b} \\ &\mathbf{x}^0 = \mathbf{M}_\mathbf{d}^{-1}\mathbf{b} \\ &\mathbf{x}^\mathbf{n} = \mathbf{M}_\mathbf{d}^{-1}\left(\mathbf{b} - \mathbf{M}\mathbf{x}^{\mathbf{n-1}}\right) \end{aligned} \tag{5.9}$$

the scheme (5.9) converges usually in just two or three iterations [3] so the increase in runtime is small. Once enough steps of the simulation have been run and the solution is close to the steady state, it is also acceptable to use the gradients at the previous step as initial guess for the iterative algorithm.

Boundary conditions

The boundary conditions that can be enforced fall into two broad categories, walls (slip) and far field (freestream values).

Walls are assumed airtight, so the normal component of the velocity must vanish on them. This condition can be enforced in a weak form by setting the flux across the boundary in (2.24) to:

$$\mathbf{u}\cdot\mathbf{n} = 0 \;\Rightarrow\; \mathbf{F}_\mathbf{n} = \mathbf{F}\cdot\mathbf{n} = \begin{bmatrix} 0 \\ p\,n_1 \\ p\,n_2 \\ p\,n_3 \\ 0 \end{bmatrix} \tag{5.10}$$

that is, only the pressure term of the momentum flux remains. Unfortunately, enforcing (5.10) alone does not guarantee zero mass flow everywhere (as the condition is only fulfilled in an average sense). This kind of problem arises mostly at the forward stagnation point. To achieve good results, zero normal velocity is enforced for points over the solid boundary in addition to (5.10). The velocity obtained after each stage of the time integration scheme is corrected according to:

$$\mathbf{u}_{\mathbf{Corr}} = \mathbf{u} - \alpha\left(\mathbf{u}\cdot\mathbf{n}\right)\mathbf{n} \qquad \alpha \in [0,1] \tag{5.11}$$

The parameter a in (5.11) is ramped slowly from zero at the beginning of the analysis to unity over a number of time steps [8]. This way the slip boundary condition is enforced in a progressive way, avoiding a sudden change at the first step which could lead to convergence problems if the initial condition is vastly different from the steady state solution (e.g. if the uniform freestream conditions are chosen as initial conditions). The normal vector at a node is taken as the weighted average of the normals of all the elements sharing the node

$$\mathbf{n}_{nod}^{i} = \frac{\sum_{k} A_k \mathbf{n}_{el}^{k}}{\sum_{k} A_k} \quad \text{where element } k \text{ contains node } i \tag{5.12}$$

It must be noted that were there is sharp corner in the solid surface the normal in not properly defined and (5.10) cannot be always enforced. Two cases must be distinguished. If the solid surface is concave at the edge, the condition (5.11) is replaced by

$$\mathbf{u}_{\mathbf{Corr}} = (\mathbf{u} \cdot \mathbf{t})\mathbf{t} \tag{5.13}$$

where t is the unit vector tangent to the edge (i.e. velocity parallel to the edge is enforced). The condition (5.10) is set on the faces of elements sharing the edge. On the other hand, if the surface is convex the direction of the velocity is not known a priori so it cannot be enforced. The velocity at the edge, in general, won't be parallel to both faces so (5.11) cannot hold. In this case the value of the normal flux is calculated the usual way and no further restrictions are set on the velocity. This is the case, for example, of the trailing edge of a wing.

The far-field boundary conditions require a certain amount of extra care. At the outer boundary where the fluid enters or leaves the computational domain only the components of the solution which enter the domain can be enforced, whereas those moving outward have to be taken from the interior solution. It is however impractical to change form conservative to characteristic variables, make the proper choice based on the propagation direction and finally revert to conservative variables. The same result can be obtained by using Roe's approximate Riemann solver to calculate the fluxes at the outer boundary [8]. Let $\tilde{\mathbf{A}}^{\mathbf{n}}_{Roe}$ be the flux Jacobian normal to the outer boundary for the intermediate state between the outer boundary and the freestream conditions. We have:

$$\mathbf{F}_{\mathbf{n}}^{\infty} - \mathbf{F}_{\mathbf{n}}^{O} \simeq \tilde{\mathbf{A}}^{\mathbf{n}}_{Roe} \left(\mathbf{\Phi}^{\infty} - \mathbf{\Phi}^{O} \right) \tag{5.14}$$

with the supra-indexes ∞ and O indication freestream and outer boundary conditions respectively. The value of the outer flux that takes into account the wave propagation characteristics of the solution can be approximated using the positive Jacobian

$$\mathbf{F}_{\mathbf{n}}^{\mathbf{Wave}} = \frac{\mathbf{F}_{\mathbf{n}}^{\infty} + \mathbf{F}_{\mathbf{n}}^{O}}{2} - \left| \tilde{\mathbf{A}}^{\mathbf{n}}_{Roe} \right| \frac{\mathbf{\Phi}^{\infty} - \mathbf{\Phi}^{O}}{2} \tag{5.15}$$

By setting the boundary flux to the value calculated through (5.15) the correct behaviour is obtained without having to enforce the value of the solution at the boundary. This method ensures that perturbations from the internal region of the domain are not reflected back at the outer boundary. Perturbations of the solution are thus minimized and no distinction has to be made between inflow/outflow or subsonic/supersonic boundaries when generating the input files.

Performance enhancement

To reduce solution time on modern desktop computers and workstations a node reordering mechanism is implemented to reduce the likehood of cache misses during execution. Starting with the first point (the choice of which is arbitrary) the nodes connected to the first one are assigned consecutive numbers. Once all of them have been

renumbered, the procedure is repeated starting with the nodes just renamed and renumbering their neighbours. The cycle is repeated until all the nodes in the mesh have been assigned a new numbering. When performing operations on the edges (which are processed in the same order as during the renumbering) the chances of a cache miss are reduced because neighbouring nodes are stored close to each order on the computer memory. The result of the renumbering algorithm depends obviously on the choice of the first node. This dependency can be somewhat mitigated by running several passes of the resequencer taking as first node at each iteration the last one from the previous pass [3].

To decrease solution time in multi-core and multi-CPU computers the code has been parallelized using OPEN-MP directives. To ensure no conflict (race condition) takes place during writes each of the active threads keeps a private copy of the residual vector. Thus, all the stages of the system assembly can be run in parallel without interaction from the different threads. Once all the private copies of the residual vector have been assembled, a reduction stage is performed in which all the private copies are merged (once again in parallel) to obtain the complete RHS of the equation. Once this has been done the solution using the lumped mass matrix can also be performed in parallel. This method of isolating the threads creates a slight memory overhead (as several partial copies of the vector are stored, one per thread) but it does not affect performance severely as long as the number of threads is not large. As PUMI is routinely run in systems with a limited number of cores (around four) there is no noticeable performance hit.

7. EXAMPLES OF APPLICATION

The code has been successfully tested with cases of industrial relevance by running simulations of a complete subsonic transport airplane. The study has been completed as part of the EU's 6^{th} Framework project REMFI (Rear Empennage and Fuselage Flow Investigation). A very detailed study of the flow field around the tail surfaces of a A380-like geometry has been carried out analyzing the aerodynamic interference and aeroelastic effects caused by the twin-sting support mechanism. This device is used in wind tunnel testing for direct force measurements at the tail section. Due to the increased torsional loads applied on the wing, an optimal boom spacing must be sought which while minimizing direct aerodynamic interference at the tail prevents excessive wing twist which could introduce additional errors through it effect on downwash. To gain further understanding of the phenomena involved the complete aircraft model including the suspension devices had to be simulated. The typical mesh for an inviscid computation contains approximately ten million tetrahedral cells. The surface mesh for such a case is shown in Fig. 1 with a close-up of the interface between wing and support arm in Fig. 2. Different boom spacings have been tested for a wide range of flight conditions, covering from low subsonic speeds (M=0,35) all the way to high transonic regime (M=0,95 which is well above typical cruise conditions for such and aircraft). Fig. 3 shows a detail of the pressure distribution around the adaptor area evidencing the large aerodynamic interference effect. The numerical results have been checked against wind tunnel measurements in order to validate the code. In Fig. 4 the computed pressure distribution over the HTP at 70% span is plotted together with experimental data gathered during wind tunnel testing. A good agreement between both sets of results is observed.

8. CONCLUSIONS

The mathematical foundations and details of implementation of a 3D unstructured finite

element Euler solver have been presented. The code is designed to provide fast results while keeping the memory requirements at a low level. In addition to the optimized memory footprint, the code is parallelized to take advantage of the growing number of multi-core or multi-processor platforms available today. The software has been tailored to run on modest hardware (desktop computers) while still being able to deal with configurations of industrial relevance (models containing tens of millions of cells). The cost of the computations is kept low while achieving good quality results (checked against other codes and wind tunnel measurements).

REFERENCES

[1] Dadvand P., 'A Framework for Developing Finite Element Codes for Multidisciplinary Applications', Ph.D. Thesis, (2007)

[2] Donea J., Huerta A., 'Finite Element Methods for Flow Problems', John Wiley & Sons Ltd., (2003)

[3] Löhner R., 'Applied CFD Techniques', John Wiley & Sons Ltd., (2001)

[4] Morgan K., Peraire J., Peiró J., 'Unstructured Grid Methods for Compressible Flows', In Report 787 – Special Course on Unstructured Grid Methods for Advection Dominated Flows. AGARD, (1992)

[5] Morgan K., Peraire J., 'Unstructured Grid Finite-Element Methods for Fluid Mechanics', Rep. Prog. Phys. 61: 569-638 (1998)

[6] Lyra P. R. M., Morgan K., 'A Review and Comparative Study of Upwind Biased Schemes for Compressible Flow Computation. Part I: 1-D First-Order Schemes, Arch. Comput. Methods Eng.,Vol. 7: 19-55 (2000)

[7] Lyra P. R. M., Morgan K., 'A Review and Comparative Study of Upwind Biased Schemes for Compressible Flow Computation. Part II: 1-D Higher-order schemes, Arch. Comput. Methods Eng.,Vol. 7: 333-377 (2000)

[8] Lyra P. R. M., Morgan K., 'A Review and Comparative Study of Upwind Biased Schemes for Compressible Flow Computation. Part III: Multidimensional Extension on Unstructured Grids', Arch. Comput. Meth. Engrg Vol 9: 207-256 (2002)

[9] Lomax H., 'Fundamentals of Computational Fluid Dynamics', Springer-Verlag, (2001)

[10] Hirsch C., 'Numerical Computation of Internal and External Flows', Volume 2, John Wiley & Sons. (1990)

[11] Turkel E., 'Improving the Accuracy of Central Difference Schemes', ICASE Report 8853, September (1988)

[12] Hu G., 'The Development and Applications of a Numerical Method for Compressible Vorticity Confinement in Vortex Dominant Flows', PhD Thesis, Virginia Polytechnic, (2001)

[13] Swanson R.C., Turkel E., 'Multistage Schemes with Multigrid for Euler and Navier-Stokes Equations. Components and Analysis', NASA Technical Paper 3631, August (1997)

[14] Hirsch C., 'Numerical Computation of Internal and External Flows', Volume 1, John Wiley & Sons. (1990)

[15] Zienkiewicz O., Taylor R., 'The Finite Element Method: Volume 3, Fluid Dynamics.' 5th edition, Butterworth-Heinemann; (2000)

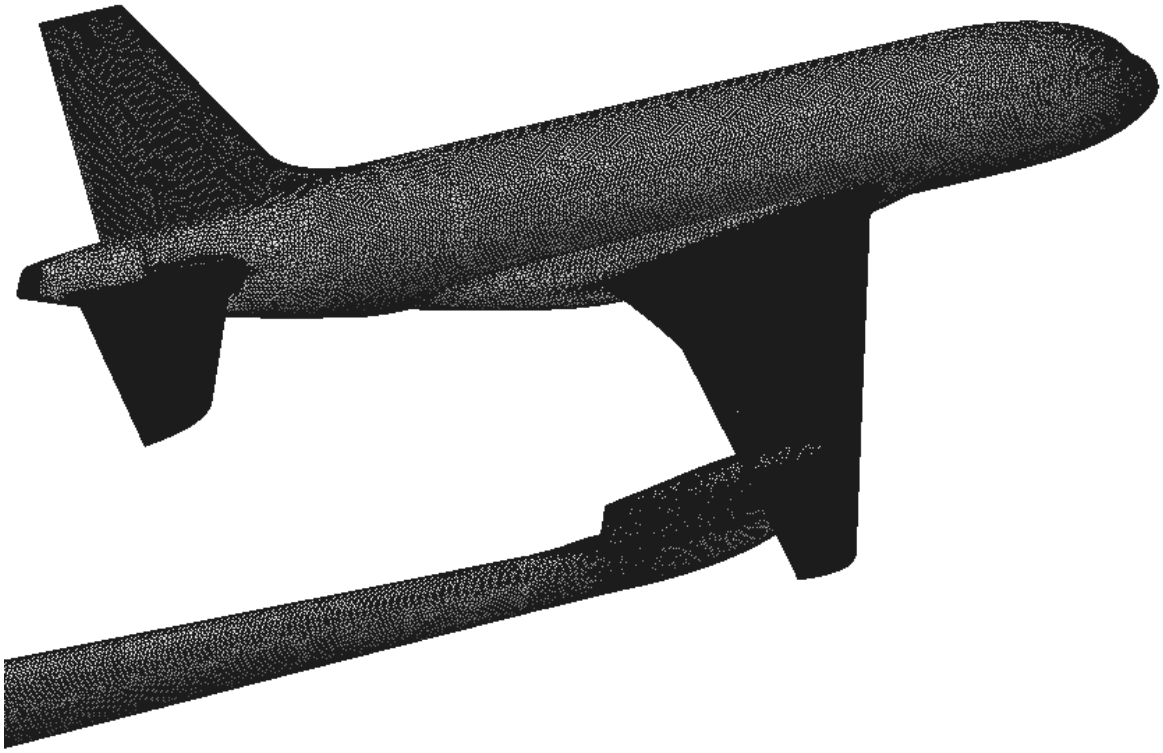

Fig. 1 Surface mesh of test model including booms and adaptors.

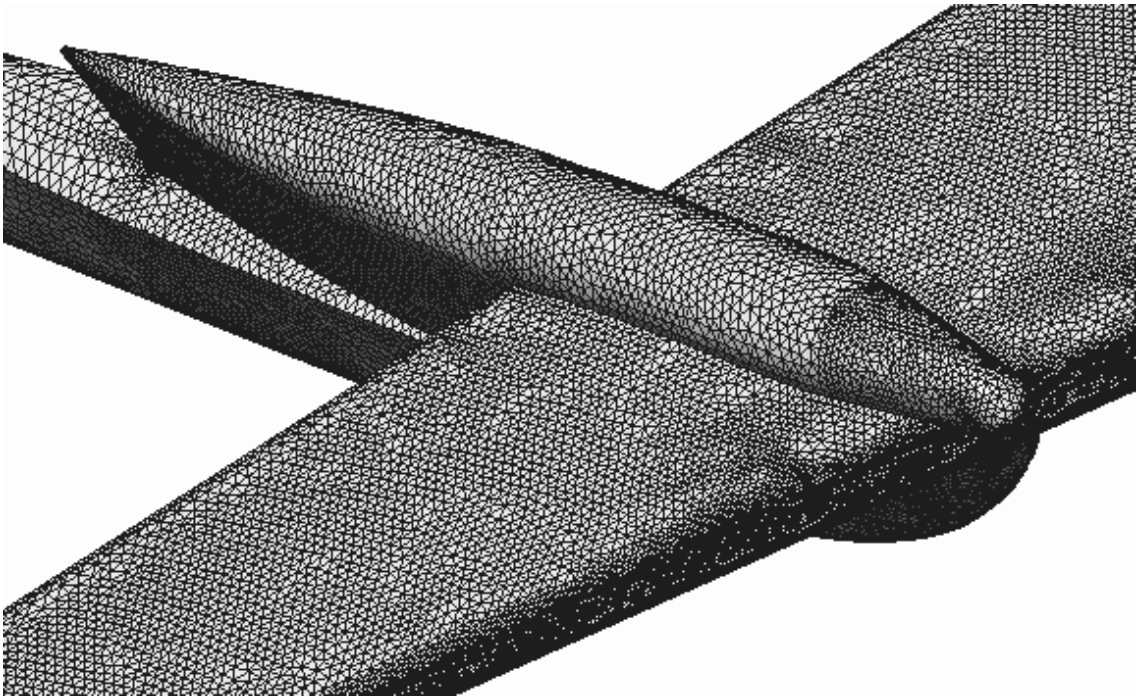

Fig. 2 Surface mesh detail around the wing-boom adaptor area.

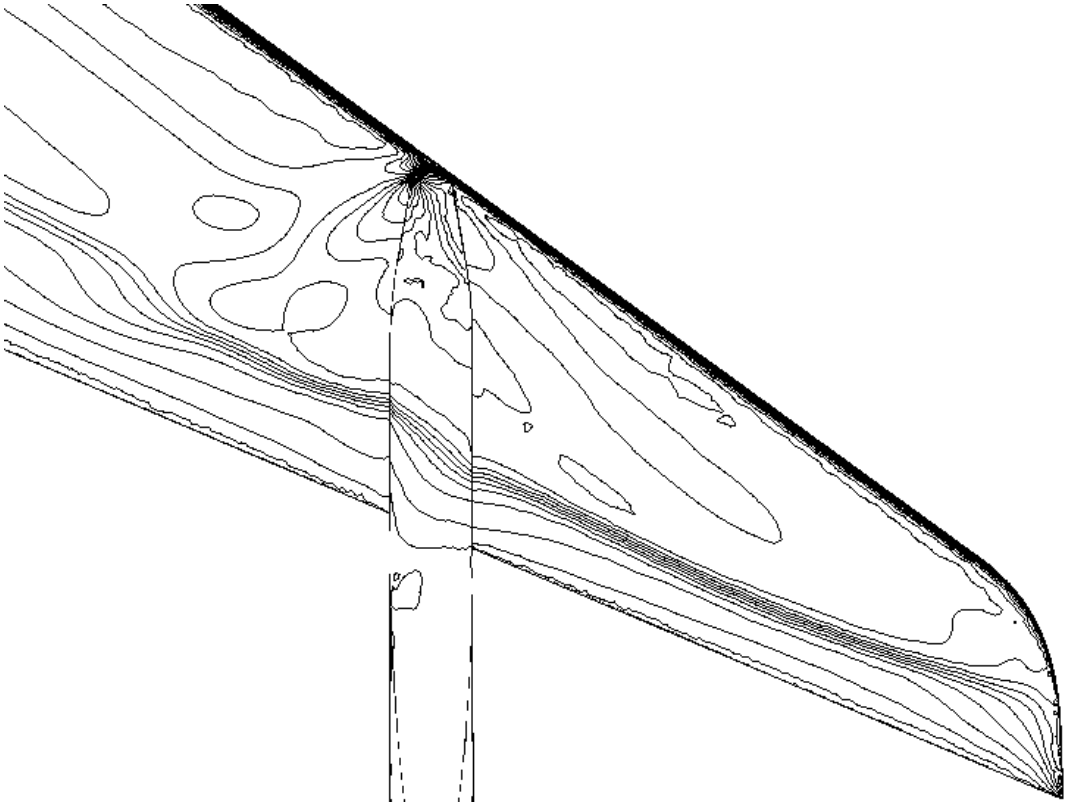

Fig. 3 Wing pressure contours showing the disruptive effect of the wing-boom adaptor.

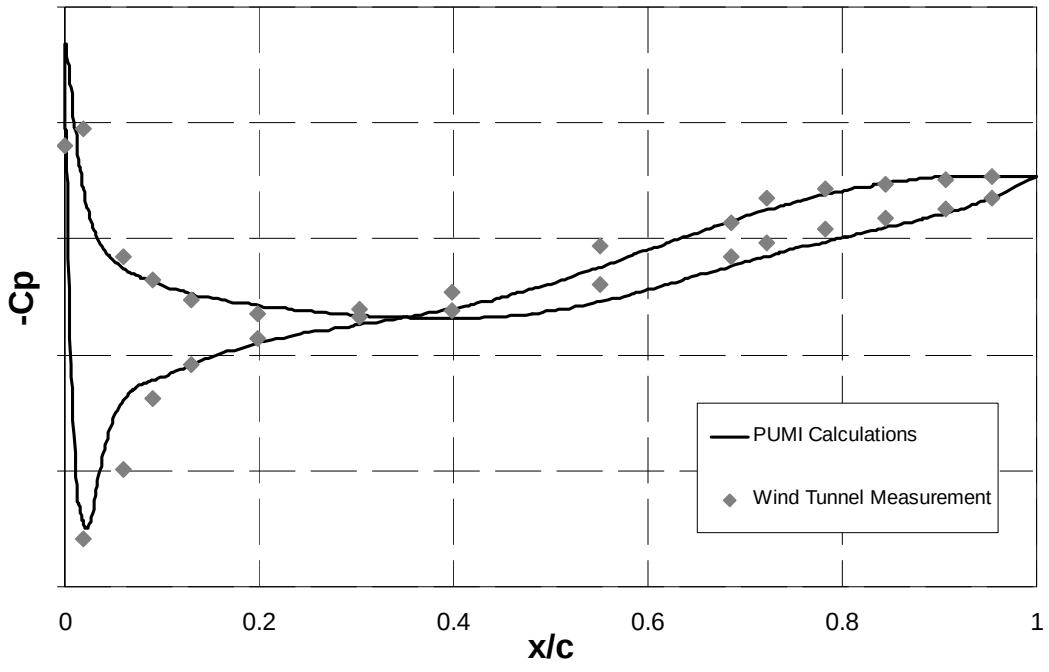

Fig. 4 Pressure distribution at the horizontal stabilizer. Comparison of simulation and wind tunnel experimental data.