Cartesian Grid Generation

Santiago Alagon Carrillo

October 28, 2013

Advisor : Prof.dr.ir. C. Vuik

Abstract

In this report an analysis of the advantages and drawbacks of *Cartesian grids* and *body-fitted grids* is presented. The basic ideas behind *cartesian grid* generation are studied aiming for the comprehension of the problem in hand, the *Point-in-Polyhedron* problem, which results from the need to determine the flow region in the Cartesian meshing process.

Contents

1 Introduction: Cartesian Mesh

One of the most critical steps in the numerical solutions of the equations involved in fluid dynamics is the division of the domain into a discrete grid. The discretization method for the domain depends on the numerical technique applied to solve the equations.

Nowadays the most used grids are *body fitted*, both structured and unstructured, but these techniques cannot be easily implemented in an automatic way and are cumbersome when dealing with complex geometries. Recently another grid generating technique has received more attention, the *Cartesian grids*. These are faster to generate, and have a straightforward implementation for moving boundaries and automatic grid generation [12].

1.1 Body-fitted grids

Body-fitted grids conform to the surface of the body, they are generated taking into account the geometry of the body inside the flow. In some cases the geometry of the problem allows the use of structured grids but for more complicated geometries the use of unstructured meshes, or mixed type meshes, is required, see Figure 1.

The major advantages of this meshing technique are, *c.f.*[9]:

a. It allows to place the nodes in an optimal way to resolve the geometrical features of the problem.

b. It allows to increase, in a flexible way, the mesh density towards the walls to ensure satisfactory near-wall resolution to capture the viscous layer adequately.

on the other hand, body-fitted grids entails a serious difficulties for complex geometries, *c.f.*[1],[9] :

a. Surface meshing is subject to conflicting requirements between local geometry and flow variations due to the link between the geometry, and the topology and connectivity of the cells. This implies that triangulation must have a good node distribution and all triangles must posses an adequate length scale.

b. Complicated geometries require the use of a block-structure grid formed by several connected curvilinear regions, see Figure 2,

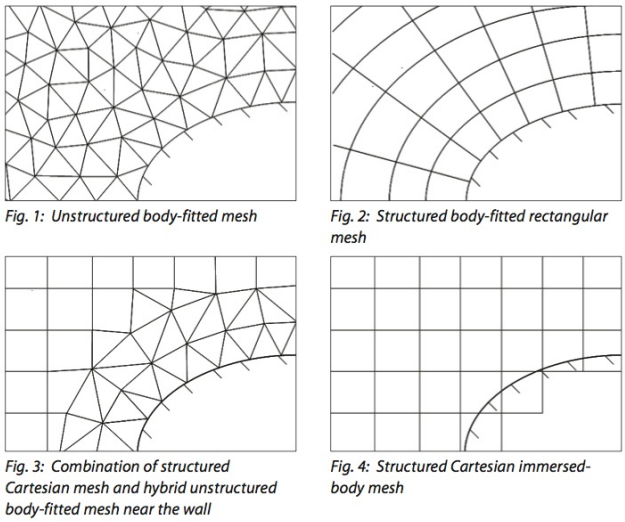

Figure 1: Mesh examples, reprint from [9]

c. The process requires user intervention, it cannot be turned into an automatic procedure.

d. Computations based on this grids are not very robust due to secondary fluxes through the diagonal grid faces.

e. Very sensitive to CAD geometry, requires clean representation of the surfaces.

f. Requires the transformation of the governing equations to curvilinear coordinates resulting in complex systems of equations which adversely impacts the stability, convergence, and number of operations for the solution.

1.2 Cartesian grid

Cartesian grid methods differ from body-fitted methods in that they are non-body fitted. The whole domain is divided by a hexahedral grid system, a set of right parallelepipeds, extending through solid walls within the computational domain. The cells are then flagged as *solid cells*, if they belong inside

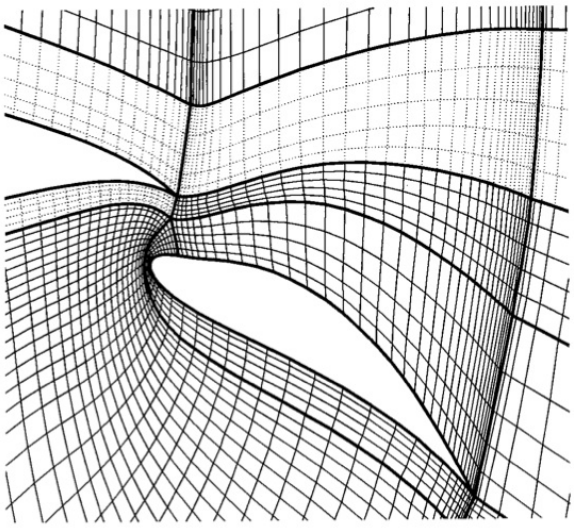

Figure 2: Block-structure for body-fitted mesh, reprint from [4]

the solid walls of the immersed body where no flow computations will take place, *flow cells*, if they belong to the region where the flow computations take place, and *boundary/cut cells* for the cells intersected by the solid walls, see Figure 3.

This transforms the problem from conforming the meshing to the surface into a characterizing and computing the intersection between the Cartesian grid and the surface geometry.

What distinguishes one type of cartesian grid method form the other is the way boundary conditions are treated, [7] [3]. One approach is to choose a staircase description of the boundary, see Figure 4(c), and impose the boundary conditions using a forcing function and extrapolation of the variables. With this approach the solution on the boundary is smeared out to the width of the local cell and no sharp fluid-to-boundary interphase can be guaranteed. The second approach is defining the so called *cut-cells* by discarding the part of each boundary cell interior to the solid boundary, the solid cell region, and using only the exterior portion, the fluid cell region, for computations; see Figure 4(b).

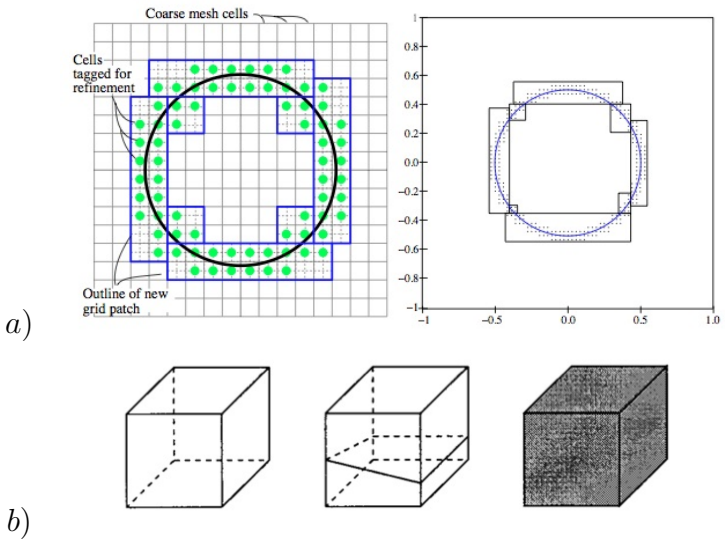

Figure 3: (a) Cell flagging; reprint from [1]; (b) Cell classification: flow cell, boundary cell, solid cell; reprint from [2]

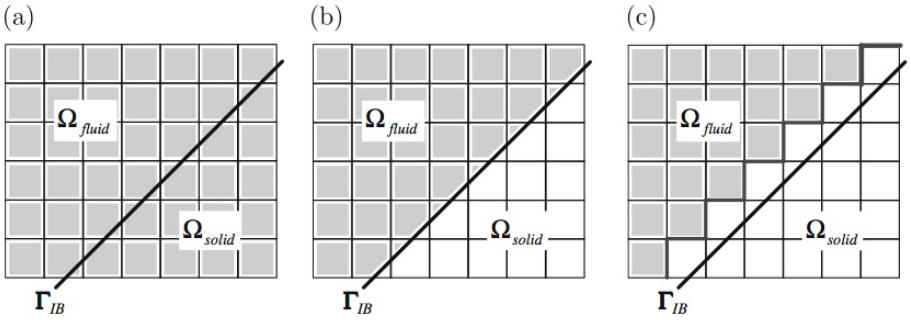

Figure 4: (a) Standard scheme, (b) Cut-cell scheme; (c) Staircase implementation; reprint from [8]

1.2.1 Cartesian mesh with cut cells

The remainder of this work is based on a cut-cell formulation of the boundary conditions for a cartesian mesh where the surface of the immersed body consists of triangular elements obtained from a previous triangulation process.

In this section we briefly review the procedure for this approach and list its advantages and disadvantages.

As it was already mentioned, the first step in generating a Cartesian grid is performing a cell division of the domain using a uniform hexahedral grid. Once the hexahedral cells have been defined they are marked, or *flagged*, as *solid cells*, *fluid cells* and *cut-cells*. Usually the solid cells are discarded unless other calculations will take place inside the immersed body.

The cut cells are then treated by dividing them into the region intersecting the immersed body and the region out side the body; usually the region inside the body is discarded as well.

The geometry of the immersed boundary is analyzed for each cell and refined based on criterions to capture the geometry of the body in a more precise manner.

Boundary-cells are intersected by the boundary in an arbitrary way which leads to complex intersections between the surface triangulation and the cut-cells. Both Cartesian cells and the triangles are convex so their intersections results in a convex polygon referred as *triangle-polygon*, *tp*. The edges of the *triangle-polygons* are obtained by clipping the edges of the triangles along the faces of the cartesian cell resulting in the *face-segments*, *fs*. The division of the faces of the cartesian cell by along their intersection with the surface triangulation result in the *face-polygons*, *fp*. This can be seen for an arbitrary Cut-cell in Figure 5

Because the surface can intersect the Cartesian mesh in an arbitrary way, the above mentioned process may produce small cut-cells which adversely affect the stability of the numerical method. To deal with this problems three procedures can be commonly found in literature,*c.f.* [10], [3],[6]

a. *cell merging*, Figure 6

b. *cell linking*, Figure 7

c. *mixed approach*

The three approaches have advantages and disadvantages and will be treated and studied if required during the development of the project. In general *cell-linking* does not perform very well for moving boundaries, [3].

After obtaining the cut-cells the grid may be refined if it is required. Assessment for the need of refinement is based on the curvature of the boundary

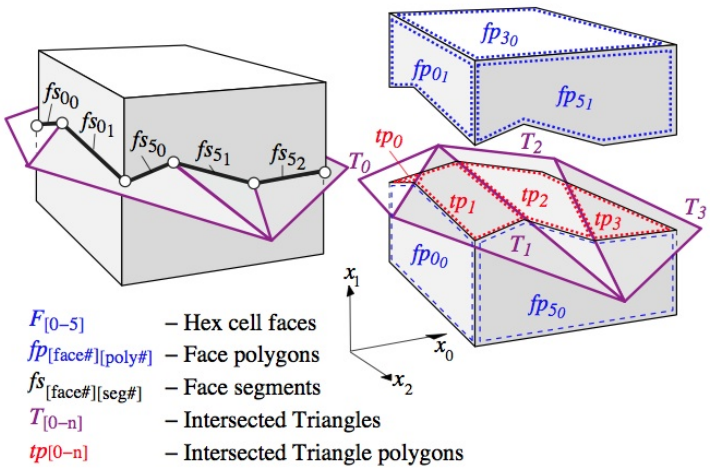

Figure 5: Anatomy of a cut cell; reprint from [1]

inside each cut-cell, the curvature difference between neighboring cells, Figure 8, and geometrical features of the problem such as small solids and narrow channels, Figures 9 and 10.

Refinement is based on thresholds for the acceptable curvature and the capacity of the mesh to capture enough detail of the geometry. The refinement process is performed by splitting each cell into two equal cells, cell splitting can take into account the flow direction for anisotropic refinement. Each time a cell is divided for refinement it is said that its refinement level has increased by one unit. The goal is to obtain a mesh not containing any of the following features, *c.f.* [13], see figure(11):

- Cell refinement level differences greater than 1 between two neighboring cells.
- Cell refinement level differences greater than 0 normal to body cut-cells.
- Cell refinement level differences greater than 0 through outer flow boundaries.
- Cell refinement level differences greater than 0 between three-sided cells and their neighbors.

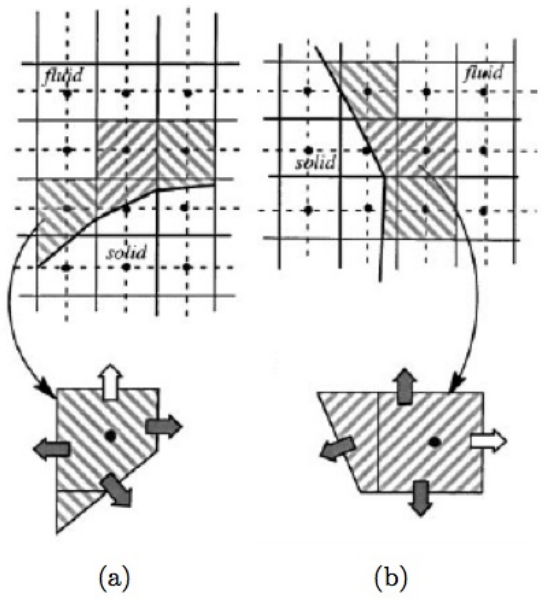

Figure 6: Cell merging for small cut-cells; reprint from [3]

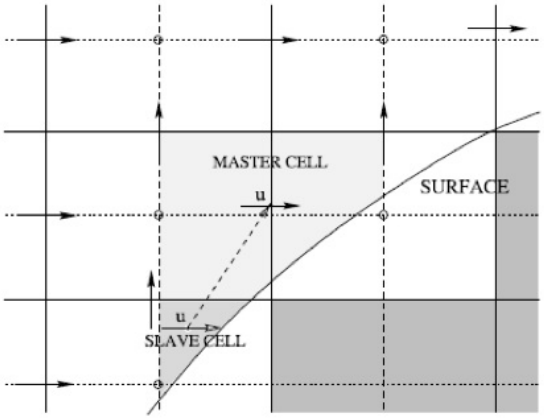

Figure 7: Cell linking for small cut-cells; reprint from [3]

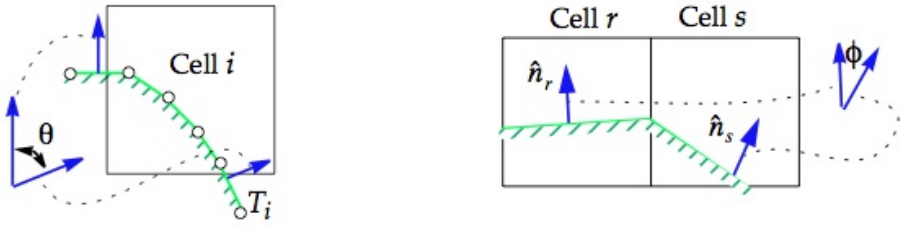

Figure 8: Measurement for the curvature inside each cut cell and between neighboring cells; reprint from [1]

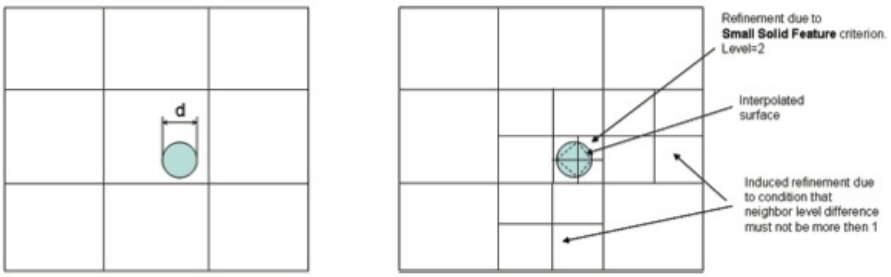

Figure 9: small geometrical features; reprint from [9]

- "Holes" in the mesh
- More than two cuts on a cell
- Cell refinement level differences greater than 0 on trailing edge of body.
- Bodies too close, only two cells apart.

With all the above process each *Cut-cell* is obtained, and thus the final grid.

Cartesian Meshes possess some disadvantages, *c.f.*[3],[1], [9]:

- Some geometrical features such as trailing edges and leading edges, require many levels of refinement

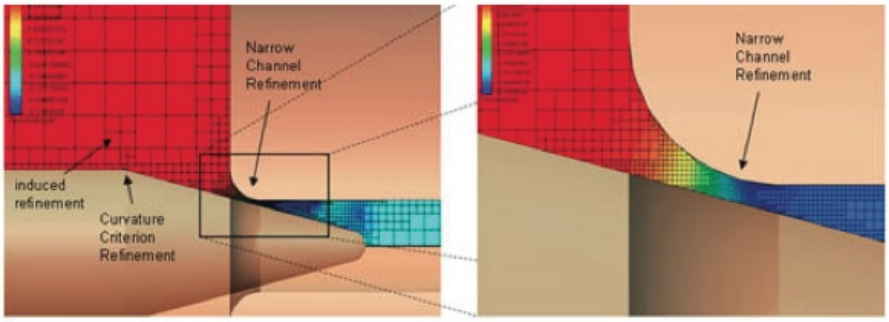

Figure 10: Narrow channels; reprint from [9]

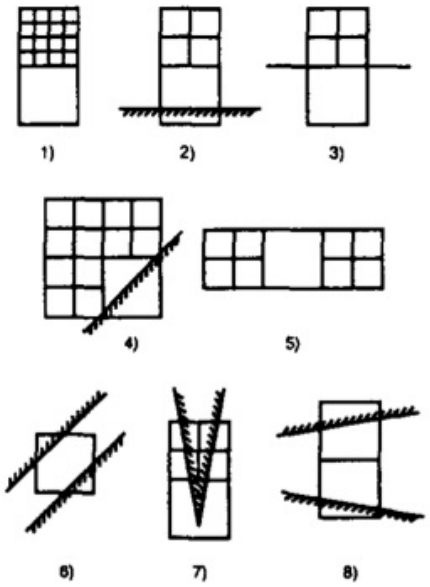

Figure 11: Undesired mesh features; reprint from [13]

- Treatment of boundary conditions on irregular cut-cells result in complex coding.
- Cut-cells should not become too small in order to maintain goos sta-

bility and convergence of the solution.

- Bodies too close, only two cells apart.

But in general are well balanced by the advantages the possess, *c.f.*[3],[1], [9]:

- Easy to convert into an automatic procedure.
- The meshing difficulties are restricted to lower order manifolds.
- It is possible to accurately impose the boundary conditions in the cut-cells.
- Cut-cell treatment of boundaries result in conservative schemes.
- Cut-cells are decoupled from the surface description.
- Easy implementation with adaptively refined grids.
- Away from the surfaces the good quality of the grid, uniform and orthogonal, imply better accuracy, lower discretization error and more efficiency.
- The meshing process is not linked to a particular representation of the boundaries, (NURB, CAD, triangulation, etcetera). The meshing process may be done using "dirty geometries", Figure 12
- Motion of the boundaries may be pre-programed and the implementation for moving geometries y quite straight forward.
- Permit the use of high resolution methods.
- Good for iterative methods.
- Good for multiphase/multimaterial flows.

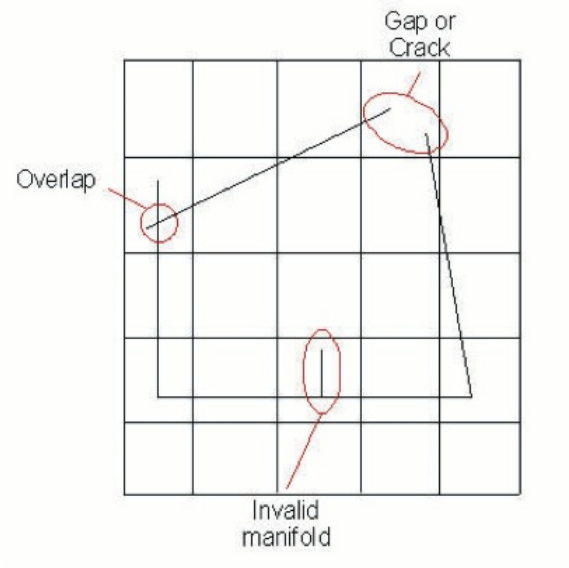

Figure 12: Schematics of dirty geometry; reprint from [9]

2 Introduction: Point-in-Polyhedron, Point-in-Polygon Problem

The *Point-in-Polygon Problem*, *PIP*, is one of the most elementary problems in computational geometry, it refers to the problem of determining wether a point R lies inside a polygon P, [16]. The *Point-in-Polyhedron Problem* is the extension to three dimensional space.

This problem arises naturally when constructing cartesian meshes due to the need to determine the sections of the cut cells inside and outside the flow, Figure(13). Accurate determination of the inner and outer regions of each cut-cell is important in order to preserve the conservative properties of the numerical method.

In this section the most common solutions to this problem will be reviewed.

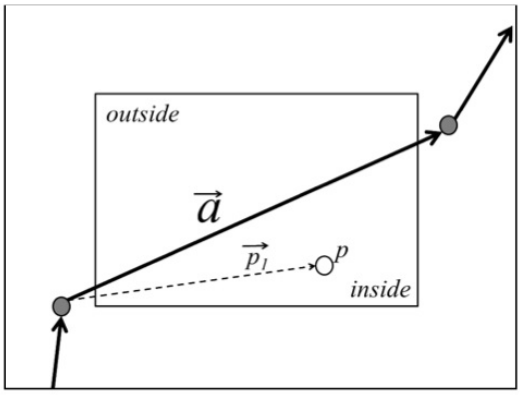

Figure 13: Point-in-Polyhedron Problem in meshing; reprint from [11]

2.1 Definitions

A *polygon* P is a set of n-points p_0, p_1, ..., p_n in the plane, called *vertices*, and n-line segments $e_0 = p_0p_1$, $e_1 = p_1p_2$, ..., $e_n = p_np_0$ called *edges*, where the union of all edges divides the plane into two regions, one bounded and one unbounded, see Figure(14), [21].

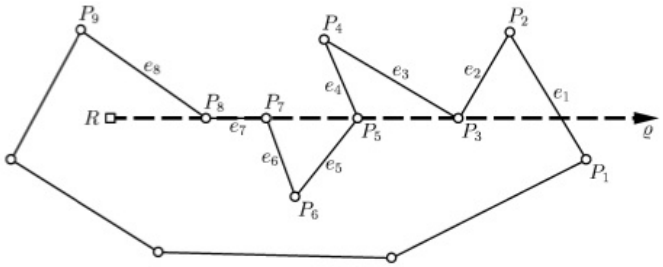

Figure 14: Point-in-Polyhedron Problem; reprint from [22]

A polygon is called *simple* if it satisfies two conditions, [16]:

- neighboring line segments meet at only one point
- non-neighboring line segments do not have any point in common.

these conditions imply that in a simple polygon non of its edges intersect or touch.

A vertex is called *convex* if less than half a small circe, centered at the vertex, is occupied by the polygon. Equivalently a polygon is *convex* if a straight line intersects the polygon's boundary at most twice, [16], [15], see Figure(15) .

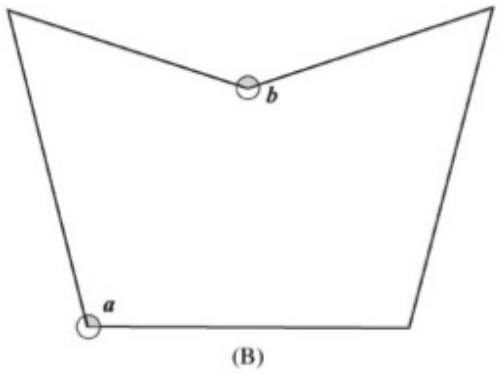

Figure 15: Convex vertex a, non-convex vertex b, ; reprint from [16]

A *polyhedron* is a set of piecewise planar polygons that bound a solid in $\mathbb{R}^3$, its faces are the planar polygons, see Figure(16), [21]. This definition is vague, but it is a difficult task to define properly this class of objects properly and it is beyond the scope of this work. A polyhedron is called *simple* if its faces do not intersect and it is called *convex* if any straight line intersect its faces at most two times, [15].

2.2 PIP algorithms

Many solution to the *PIP* problem can be found in the literature. The principles upon which these algorithms are based are, [15]:

- line crossing
- angle-sum
- area-sum

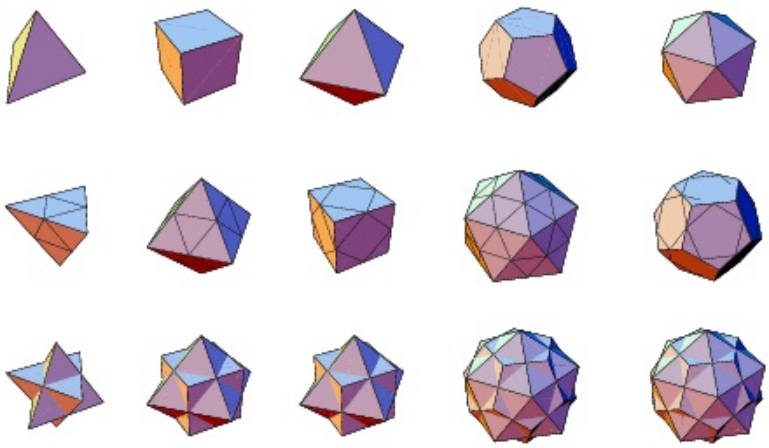

Figure 16: Examples of polyhedrons; reprint from [23]

- orientation method

these principles will be explained latter when the way they are implemented in the *PIP* strategies is described.

The algorithms based on the *line crossing* principles have been shown to be the most time and memory efficient, [15]. To better understand this principle we need to know the following theorem

Jordan Curve Theorem

Any simple closed curve divides the plane into exactly two regions, one bounded and one unbounded.[21]

Jordan's curve theorem implies that a point R is inside a simple polygon G if the parity of the number of intersections between G and a line extending from R to infinity is odd, see Figure(17).

Based on these principles the following algorithms have been suggested to solve the *PIP* problem for general polygons, [16]:

- *Ray crossing method.* a ray is shot from the test point to infinity and the number of intersections of the ray and the polygonal boundary is computed. In practice the *point to infinity* is determined to be a point far away in the domain, most of the times belongoing to the bounding box of the polygon. The line crossing principle is then used to determine

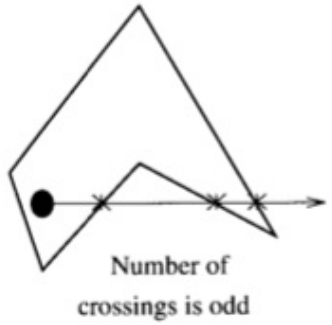

Figure 17: Line crossing principle; reprint from [14]

if the point is interior or exterior to the polygon, see Figures(17),(14). Works in $\mathcal{O}(n)$ time and solves for any polygon.

- *Sum of angles* The polygon is treated as a fan of triangles emanating from the test point where one of the edges of each triangle is an edge of the original polygon and the other two edges are the segments joining the test point with the edge of the polygon. The angle at the vertex of each triangle at the test point is summed and if it is equal to 360° the test point is determined to be inside, see Figure(18). Works in $\mathcal{O}(n)$ time and solves for any polygon but is sensitive to the problems of finite arithmetic, see also [1] p.60.

- *Swath method* This method requires pre-processing. The Polygon is firstly divided into swaths (trapezoids) in time $\mathcal{O}(n \log n)$. The swath where the point is located is determined and the *Ray crossing* test is applied, see Figure(19)

- *Grid method* A look up grid is placed over the polygon, each of the grid cells' is categorized as either internal, external or boundary using the line crossing principle.

- *Triangle-based method* The polygon is decomposed into a set of triangles in linear time. Each triangle is defined by a fixed vertex arbitrarily determined to be the origin, whereas the other two vertices are determined by a polygon edge. The triangles obtained are classified as

positively of negatively oriented. A line starting at the origin and containing the test point is matched with all triangles. The test point is determined to be interior or exterior based on the intersections of this line with the positive and negative triangles, see Figure(20)

Other methods are mentioned in the literature but they work only for convex polygons and will not be treated in this work.

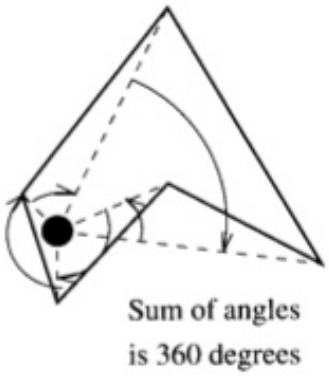

Figure 18: Sum of angles test; reprint from [14]

Due to the nature of the *Cartesin Meshing process* it is natural to have a deeper look into the *Grid Method* as a first option to determine the inner and outer regions, relative to the flow, of the object immersed in the flow.

The *Cartesian mesh* can be used as the look up grid where the line crossing criterion is applied, see Figure(21).

In the *Grid Method* the boundary cells must contain a list of edges of the polygon that overlap their boundaries. In the cell one corner or more are determined to be either inside or outside the polygon. A line segment is then traced between the test point and the corner of the cell for which the state is known, and the state of the test point is obtained applying the line crossing principle, see Figure(22)

Singular conditions can occur when applying the *Ray crossing method*, and any other of the other method which apply the ray crossing principle. This singular conditions happen when one or more vertices of the polygon lie on the test line or when an edge of the polygon is co-linear with the test

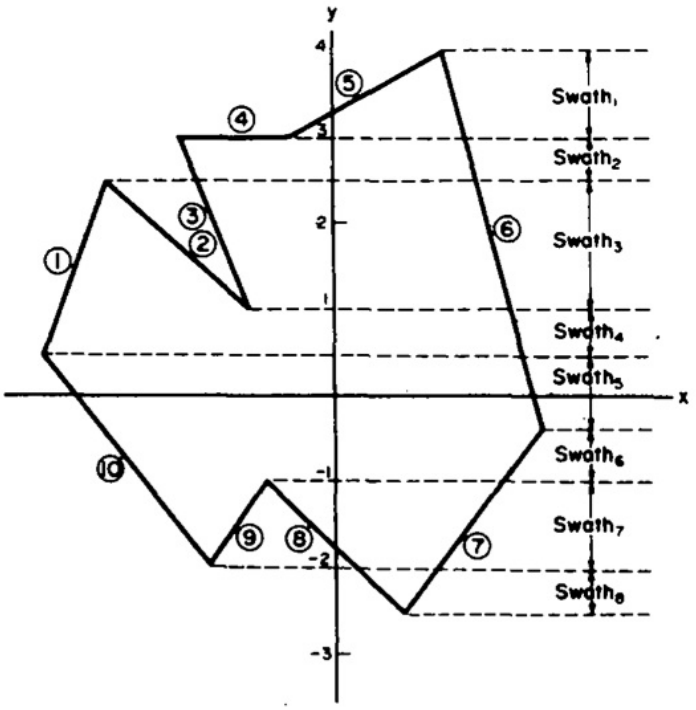

Figure 19: Swath method; reprint from [19]

line. The correctness of the *Grid Method* can be impaired by the singular conditions which also affect the *Ray crossing method*,

Usually for computational geometry a uniform grid is placed as a look up grid, but refinement can be used to obtain better accuracy and to treat singularities.

When refinement is used it is performed in each cell until one of the following occur, [17]:

- There are no more edges of the polygonal interface inside the cell.
- There is at most one vertex of the polygonal interface inside the cell with at most two edges of the polygonal interface crossing the boundaries of the cell.

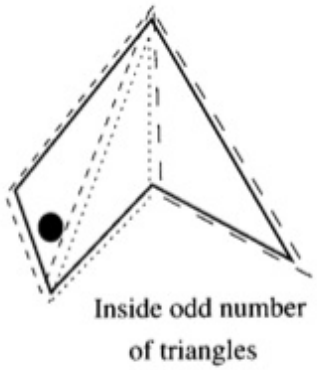

Figure 20: Triangle-based method; reprint from [14]

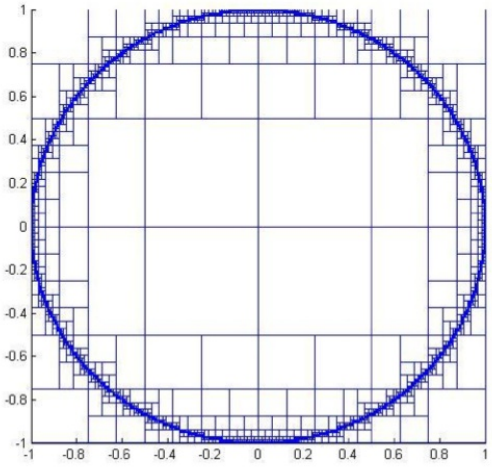

Figure 21: Example of refined Cartesian mesh used as look up grid; reprint from [17]

- There are no vertex of the polygonal interface inside the cell and there is at most one edge of the polygonal interface crossing the boundaries.

Refinement criterions for the look up grid might not be compatible with the refinement criterions for the *Cartesian Mesh*, this will be part of the

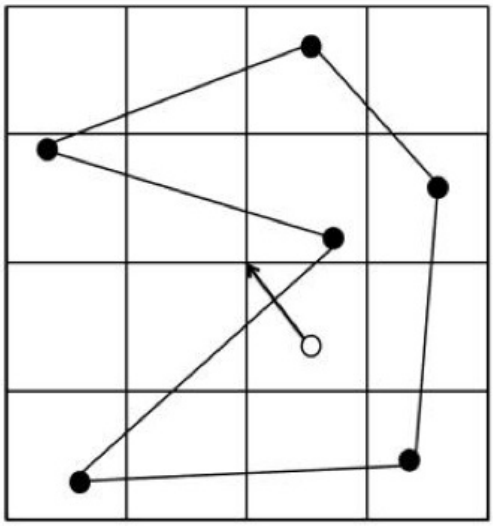

Figure 22: Grid method representation; reprint from [17]

research focus of this work.

Additional difficulties arise when a polygonal edge crosses exactly or close to a grid corner, in this case the state of corner is not classifiable and cannot be used to perform the *Ray crossing method in the local cell*. The accuracy of the flow solver might get impaired if the unclassifiable cell was the one used to determine the inner and outer parts of the polyhedron, the solution to this problem is the goal of this work.

3 Tracking the Propagating interface: Level set method

Once the numerical methods to solve the flow and the mechanics of the airbag have been defined, a way to obtain the evolution of the moving interface, airbag, needs to be developed. Accurate tracking of the interface is essential to keep the conservative properties of the *Finite Volume* solver.

In this section a review of the *Level-set method* is reviewed. This approach to the airbag deployment tracking has been shown in [5] to be both efficient and robust. The advantage of this method is that the inside and outside of the moving interface can be determined by the use of a signed distance function, which avoids the *Point-in-polyhedron* problem.

3.1 Level Set Method: Despription

The *Level Set Method* is a technique to track the movement of an interface and shapes when the velocity for each point of the interface is known. The *Level Set* approach allows to perform computations involving curves and surfaces on a fixed Cartesian grid without having to parametrize the objects. This makes it suitable to study the evolution of objects that change in topology, that is, when the interface splits or merges as well as when holes form in it.

The core idea of the *Level Set* method is to embed the surface that needs to be tracked, in a one higher-dimensional geometric manifold given by the so-called *Level Set Function* φ. The interface is embedded into φ in such a way that it is obtained as the *zero* level set, $\varphi = 0$, see Figure(23).

The goal of the *Level Set method* is to determine the initial *level set function* φ at time $t = 0$ and to appropriately evolve this funtion in time to match the evolution of the interface. Once this is achieved, at each time step the interface is given always by the *zero* level set $\varphi(\bar{x}, t) = 0$.

In such way, the *Level Set Method* exchanges a geometric problem in moving coordinate representation for a fixed coordinate representation.

3.2 Level Set Method: Technical details

Suppose that we are given an initial interface separating the space into two regions. Together with the interface we are given also the speed F of each

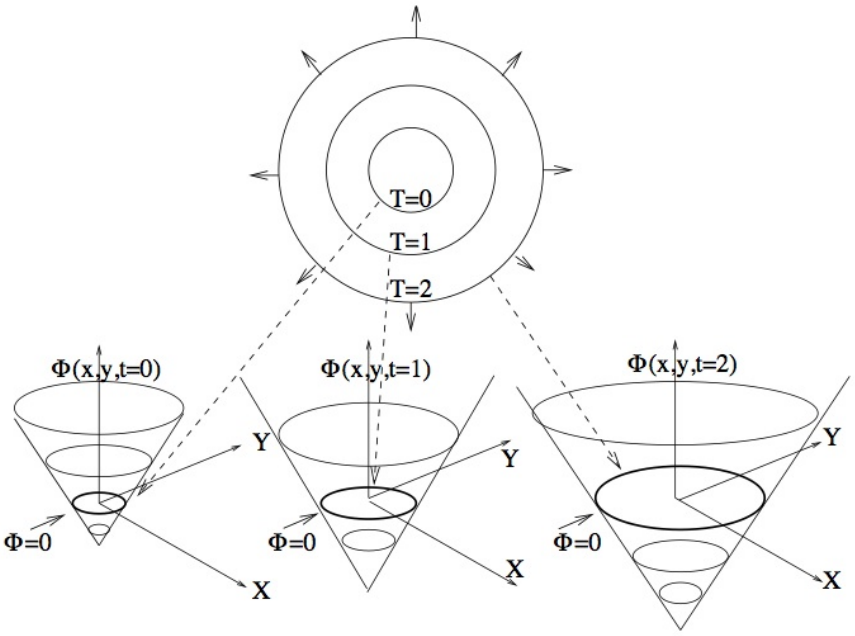

Figure 23: Representation of Level Set method; reprint from [24]

point in the interface in the normal direction to it.

The initial *Level set function*, $\varphi(\bar{x}, t = 0)$, is built using the signed distance function d from each point $\bar{x}$ in the Cartesian Grid to the interface. This way the points interior to the interface are such that $\varphi < 0$ while the points exterior to the interface are such that $\varphi > 0$, see Figure(24)

The only thing that remains is to determine how to adjust the value of φ in time to match the evolving interface. We want the *Level Set Function* to contain the interface, for any time, as the *zero* level set. So the interface is given always by

$$\varphi(\bar{x}(t), t) = 0$$

To obtain the evolution of the interface in time we differentiate using the chain rule

$$\varphi_t + \nabla\varphi(\bar{x}(t), t) \cdot \bar{x}' = 0$$

where the similarity with the so-called *material derivative*, used in elasticity

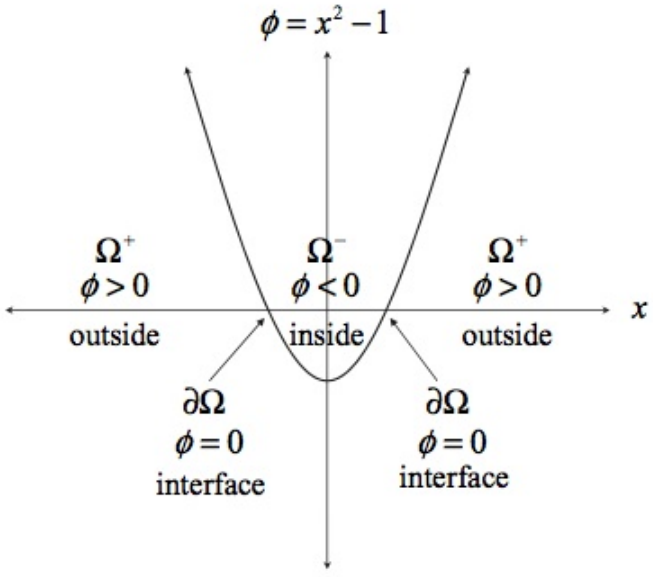

Figure 24: Decomposition of space into interior and exterior according to sign of φ; reprint from [26]

and fluid mechanics to describe the temporal change of a property due to both its change in position and time.

Only the normal speed $F = x' \cdot \hat{n}$ to the interface results in variations on its shape, so the last equation can be written as

$$\varphi_t + F|\nabla\varphi| = 0 \tag{1}$$

where we used $\hat{n} = \nabla\varphi/|\nabla\varphi|$. Equation(1) is know as the level set equation. Thus, an initial value problem is obtained.

$$\begin{aligned} \varphi_t &+ F|\nabla\varphi| = 0 \\ \Gamma(t) &= \{(x,y)|\varphi(\bar{x},t) = 0\} \end{aligned}$$

Stated as it is, and thinking of the solution in purely geometric terms, problems with uniqueness of the solution are encountered. The purely geometric approach to the solution is to move all the points with speed F in the

direction normal to the initial front. A second approach is to think of the front moving as if it were a wave front moving. The solution is given by the envelope of the waves emanating form each point, which satisfies uniqueness requirements. Both approaches are represented in Figure(25).

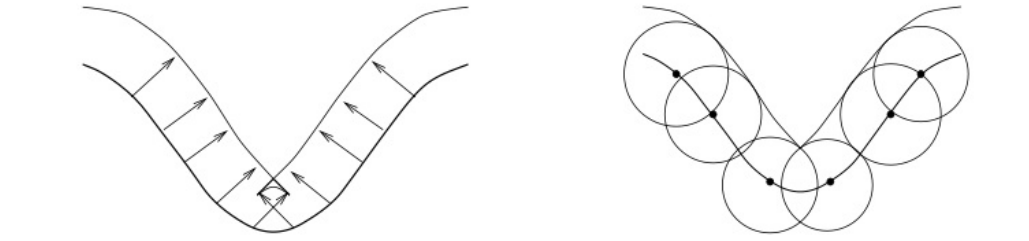

Figure 25: Moving interface. a) moving each point normal to the curve; b) using the envelope of the wave fronts emanating from each point; reprint from [24]

It can be shown that the solution obtained from the envelope of the wave front envelope is the same as the solution of the associate viscous problem $\varphi_t + F|\nabla\varphi| = \epsilon\nabla^2\varphi$ in the limit as $\epsilon \to 0$, see Figure(26)

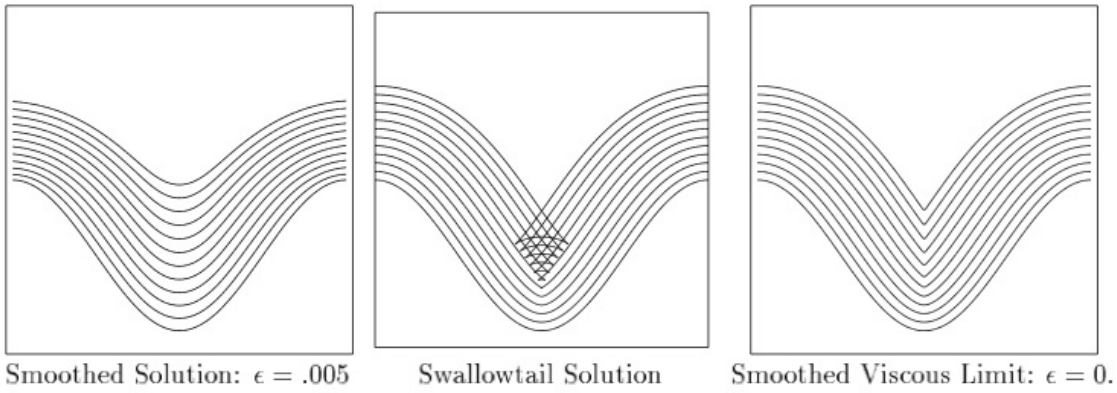

Figure 26: Solution to viscous problem; reprint from [24]

3.3 Level Set Method: Advantages and Disadvantages

For the interface tracking problem, the *Level Set Method* possesses certain advantages over other approaches, [26], [24], [25]:

- The general procedure is unchanged for any number of dimensions.
- Topological changes in the front are handled naturally.
- Rely on the viscosity solution of the associate partial differential equation.
- Can be converted into a computational scheme by using the already known schemes from hyperbolic conservation laws.
- There exist computational strategies to make it a more efficient procedure.

The only point of precaution when turning it into a computational scheme is the dependence on the CFL condition.

3.4 Level Set Method: Application to airbag deployment

The *Level Set Method* can be applied as a coupling algorithm for Eulerian-Lagrangian shell-fluid interaction as shown in [5] and [27], see Figure(27).

The arbitrary boundary of the airbag is immersed in the Cartesian Mesh creating cut-cells. By filling this cut-cells and a small layer of neighboring regular cells with an appropriate *ghost fluid*, cell updates is performed in the same way as the bulk cells in the computational domain. The discontinuities in the flow field resulting form the interface are directly embedded in the solution through appropriately populated ghost cells, [5], [27]. This approach is known as the *ghost fluid* method.

At each time step the signed distance from the deformed shell to the grid points in the Cartesian Mesh is computed resulting in an implicit representation of the fluid-shell boundary.

Because this approach avoids the generation of small cells, derived form the conformity requirement of the mesh at the interface and thus the use of cut cells, it has also the advantage of having less restrictive time step constrains.

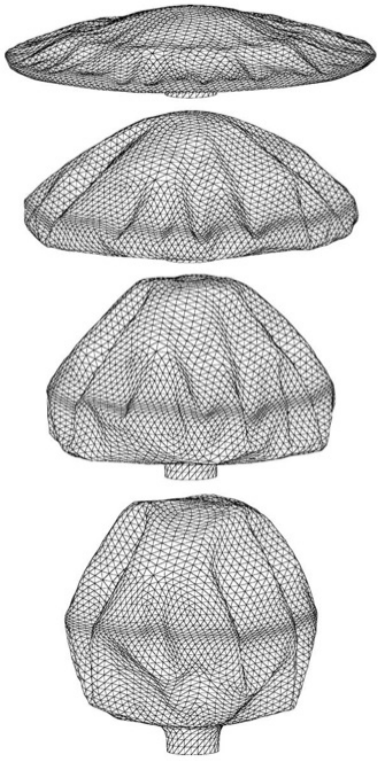

Figure 27: Airbag deployment simulation using level set method; reprint from [5]

4 MADYMO Gasflow2 Solver

Gasflow2 is a numerical solver for fluid-structure interaction, developed to solve the flow during the deployment phase of airbags. *Gasflow2* implements an Arbitrary Lagrangian-Eulerian solver based on a Cartesian cut-cell method.

A Cartesian cutcell method program is comprised of two mayor components, the flow solver and the geometry calculations. This section contains a short review of the actual implementation of these two components in *Gasflow 2*.

The information on these sections is based on the information in the technical reports provided by *TASS*, [28], [29], [30], [31], [32], [33], [34], [35],

[36], [37]

4.1 Flow solver in *Gasflow2*

Gasflow2 approximates the flow solution based on *Euler's equations* in 3-dimensional space. The flow is assumed to be a single component calorically perfect gas, *i.e.* the specific heat for both constant pressure and volume, denoted by c_p and c_v, are assumed to be constant.

For a bounded region Ω in $\mathbb{R}^3$, with boundary S, *Euler's equations* read

$$\frac{d}{dt}\int_{\Omega}\mathbf{q}d\Omega + \int_{S}\Phi(\mathbf{q},\hat{n})dS = \mathbf{S}(\mathbf{q}) \tag{2}$$

where

$$\mathbf{q} = \begin{pmatrix}\rho\\ \rho u\\ \rho v\\ \rho w\\ \rho e\end{pmatrix}, \text{ and } \Phi = \begin{pmatrix}\rho v_n\\ \rho v_n u + p\cdot n_x\\ \rho v_n v + p\cdot n_y\\ \rho v_n w + p\cdot n_z\\ \rho v_n H\end{pmatrix} \tag{3}$$

where $\hat{n} = (n_x, n_y, n_z)$ is the outward normal of S, $\mathbf{v} = (u, v, w)$ is the velocity Cartesian vector and $\mathbf{S}(\mathbf{q})$ denotes a general source term.

The total energy per unit volume is $\rho e = \rho u_{int} + \frac{1}{2}\rho(\mathbf{v}\cdot\mathbf{v})$ but during the simulation it is computed from $p = ((c_p/c_v) - 1)\left(\rho e - \frac{1}{2}\rho(\mathbf{v}\cdot\mathbf{v})\right)$.

A correction needs to be done at the interface with the airbag due to the movement of it produced by the flow. To apply the interface conditions an Arbitrary Lagrangian-Eulerian (ALE) formulation is used.

The airbag edge is described though a set of triangular segments whose position is approximated using a Finite Element solver.

For the fluid-structure interface the the two interface conditions that needs to be satisfied are the *kinematic condition*

$$v_f = v_s$$

and the *dynamic condition*

$$\sigma_f = \sigma_s$$

where σ is the stress tensor $\sigma = p\mathbf{I} + \tau$.

For an inviscid fluid the structure-fluid coupling, with structure velocity v_s and flow velocity v_f, conditions result in

$$\mathbf{v}_f\cdot\mathbf{n} = \mathbf{v}_s\cdot\mathbf{n} \text{ and } p_f = p_s \tag{4}$$

In the ALE formulation *Euler equation's* for the fluid domain are given by

$$\frac{d}{dt}\int_{\Omega(t)} \mathbf{q} d\Omega + \int_{S(t)} \Phi(\mathbf{q}, \mathbf{v}_s) \cdot \mathbf{n} dS = \mathbf{S}(\mathbf{q}) \tag{5}$$

where the flux function through the control volume boundary and the boundary state vectors are defined as

$$\mathbf{q_B} = \begin{pmatrix} \rho \\ \rho(\mathbf{v}_f - \mathbf{v}_s) \\ \rho e_B \\ \rho_1 \\ \vdots \\ \rho_{N-1} \end{pmatrix}, \text{ and } \Phi(\mathbf{q}, \mathbf{v}_s) = \begin{pmatrix} \rho(\mathbf{v}_f - \mathbf{v}_s) \\ \rho u(\mathbf{v}_f - \mathbf{v}_s) + p\mathbf{I} \\ \rho v(\mathbf{v}_f - \mathbf{v}_s) + p\mathbf{I} \\ \rho w(\mathbf{v}_f - \mathbf{v}_s) + p\mathbf{I} \\ \rho H(\mathbf{v}_f - \mathbf{v}_s) + p\mathbf{v}_s \end{pmatrix} \tag{6}$$

where $e_B = h - \frac{p}{\rho} + \frac{1}{2}|\mathbf{v}|^2$

4.1.1 Discretization

The discretization of the *Euler equations* is performed through the Method of Lines. The spatial derivatives are replaced by algebraic approximations, using Finite Volumes approach, to obtain a system of Ordinary Differential Equations where only time remains as a variable. The next step is to apply an integration algorithm for initial value Ordinary Differential Equations to compute the approximate solution of the original Partial Differentia Equation. In other words, the Method of Lines discretizes the spatial and temporal derivatives separately.

The spatial discretization is performed by a Finite Volumes method over hexahedral cells. The finite Volumes used in *Gasflow2* is based on a Roe scheme where the flux derivatives are approximated using a flow-differencing splitting scheme. This formulation does not allow discontinuous expansion fans.

To obtain a second order space discretization, the state of the cell face is computed from the state at the cell center assuming a linear variation of the form

$$\mathbf{q}_k(\mathbf{x}) = \mathbf{q}_k(\mathbf{x}_c) + (\Psi_k(\nabla \mathbf{q}_k))^T \cdot (\mathbf{x} - \mathbf{x}_c).$$

The gradient reconstruction for a component $\mathbf{q}_k$ in cell Ω_j follows form the minimization of the functional

$$\mathcal{J} = \frac{1}{2} \sum_{j \in N} ||\nabla q_k \cdot \Delta \mathbf{x}_j - \Delta q_j||^2$$

where N is the set of all faces of cell m and j is the index of the face which separates cell m and the neighbor n. Here $\Delta \mathbf{x}_j = \mathbf{x}_m - \mathbf{x}_n$ is the distance between the cell centers and $\Delta q_j = q_m - q_n$. This leads to a non-square linear problem which is equivalent to the solution of the system of normal equation. The system of normal equations is SPD and can be solved by direct methods.

The task of the limiter Ψ is to enforce the monotonicity principle, which requires that the reconstructed values does not exceed teh maximum and minimum of the neighboring centroid values and its own cell center value. The limiter acts in each component of $\mathbf{q}$. For a uniform mesh with cubic cells of length h, where $\Psi = 1$ the reconstruction has truncation error $\mathcal{O}(h^2)$ and in regions where $\Psi = 0$ the reconstruction has truncation error $\mathcal{O}(h^1)$.

The two limiters implemented are the *Venkatakrishnan-limiter* and the *Barth-Jespersen limiter*. Both are applied through *directional limiting*.

The temporal integration is performed via a Runge-Kutta method. The volumes are not assumed to be constant so the Runge-Kutta as applied in the discretization is also valid for changing cells, as is the case with *Adaptive Mesh Refinement*.

In general for a Cartesian Mesh without cutcells the time step restriction for the Runge-Kutta method follows from

$$\tau = min_{j \in N_c} \text{ , with } \tau_j = \frac{\sigma \Delta_j}{\lambda_{max}}$$

where N_c is the set of cells, Δ_j is the cell size of cell j, σ is the Courant number, and λ_{max} is the maximum wave speed of the system, [31].

As mentioned before, the volume of the cut-cells can be smaller than the smallest Cartesian cell which leads to stability problems. To avoid stability problems, cut-cells whose volume is smaller than that of the smallest Cartesian cells, are merged into clusters used to update the state vectors in the next time step. The stability criterion for the time step for the clustered cells follows from

$$\Delta t = min_{j \in N_c} \frac{|\Omega|_{min} \sigma}{\sum_{k=1}^{N_f} \Psi_k}$$

where $\Psi_k = (a + v_n)_k S_k$. The merging partner is selected based on the direction most normal to the boundary face.

4.1.2 Boundary conditions

In order to have a well posed problem, initial conditions and boundary conditions must be specified.

Initial conditions should satisfy (2) inside the domain and the boundary conditions should specify the state vector completely and conditions (4) have to be satisfied in the boundary and implemented on the discretization.

Boundary conditions for the cells are enforced through the specification of the fluxes at the boundary. For each cell face there are seven possible conditions

- wall
- subsonic inflow
- sonic inflow
- supersonic inflow
- subsonic outflow
- sonic outflow
- supersonic outflow

The specification of each of these boundary conditions depends on the eigenvalues of the *flux Jacobi matrix* Φ.

For the non-stationary boundary created by the airbag the velocity of the flow prescribes the normal velocity component of the airbag

$$v_n = V_n$$

and the boundary flux density Φ_B is computed using the relative velocities

$$\Phi_B(\mathbf{q}_B, \mathbf{n}_B) = \begin{pmatrix} \rho(v_n - V_n) \\ \rho(v_n - V_n)u + pn_x \\ \rho(v_n - V_n)v + pn_y \\ \rho(v_n - V_n)w + pn_z \\ \rho(v_n - V_n)H + pV_n \end{pmatrix}$$

Conditions (4) are implemented through the definitions of the fluxes in Φ as defined in (6). The structural velocity at each cell center $\mathbf{x}_c$ is obtained through means of interpolation between the Finite Element nodes according to

$$\mathbf{v}_s(\mathbf{x}_c) = \sum_{i=1}^{3} N_i(\xi)\mathbf{v_{N,i}}$$

where $\mathbf{v}_{N,i}$ is the nodal velocity and $N_i(\xi)$ is the local shape function evaluated in the FE-segment transformed coordinate system.

The pressure in the cutcells contributes to each of the nodal forces of the FE partition inside the cutcell. The FE segments inside each cutcell are partitioned into N_{sp}-number of segment polygons, which each exert a force

$$\mathbf{f}_{p,j} = p_j(\mathbf{x}_c)\mathbf{n}_j S_j$$

on the FE segment. The total contribution to node i is computed according to

$$\mathbf{f}_{N,i} = \sum_{j=1}^{N_{sp}} N_i(\xi)\mathbf{f}_{p,j}.$$

The coupling is obtained by a standard *loose-coupling* procedure, *i.e.*, at each time step one evaluation of FE and flow solver are performed.

For known locations of the FE elements, the fluid solution is advanced over one time step returning new contributions to the nodal forces $\mathbf{f}_n$ to the FE solver. The coupling procedure is represented in Figure(28)

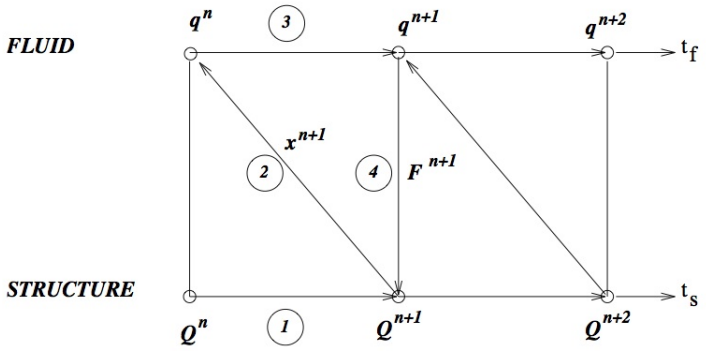

Figure 28: Schematic fluid-structure coupling algorithm; reprint from [28]

4.2 Geometrical Calculations in *Gasflow 2*

As mentioned before, the second mayor component of a Cartesian cut-cell program is the geometry calculations module.

To capture good detail of the geometry, *Gasflow2* implements Hybrid Adaptive Mesh Refinement (HAMR). An introduction to the Cartesian Mesh generation was already given in Section(1), along with some of the often used refinement criterions. In this section a brief overview of the actual implementation of the meshing process used in *Gasflow2* is presented.

The final result of the HAMR is the total division of the computational domain into cells. The cells in general are not of the same size due to the adaptive refinement process. The cells are stored in a linear-octree, a tree structure with eight children per node, which makes hierarchical data processing efficient.

During the whole refinement process a curve that goes through all the cells is constructed, this curve is called the *Space Filling Curve* (SFC), see Figure(29).

Based on the SFC each cell can be identified by a unique binary code associated to it called the *Morton key*. Vice versa, given the *Morton key* of a certain cell, its location in the computational domain can be known.

The total number of cells is constantly changing due to the HAMR process. This leads to an implementation that needs to be capable of dynamically providing or releasing memory according to the creation or removal of cells

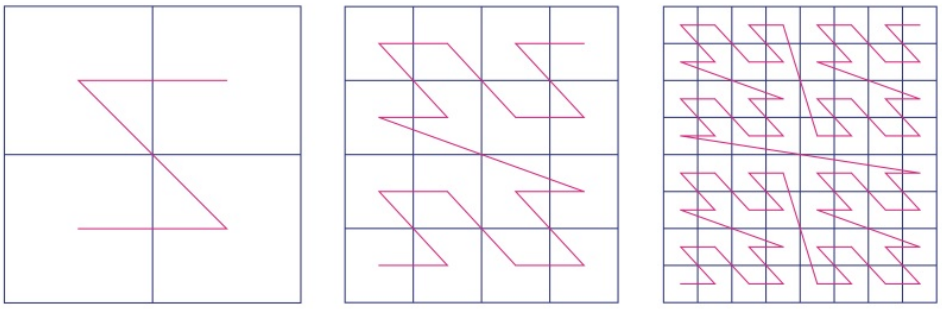

Figure 29: Space Filling Curve; reprint from [37]

during the process.

To satisfy such memory needs the whole octree structure of the cells is stored as a linked list forming a linear-octree. The one dimensional list of cells follows the order of the SFC, see Figure(30) for a bi-dimensional example.

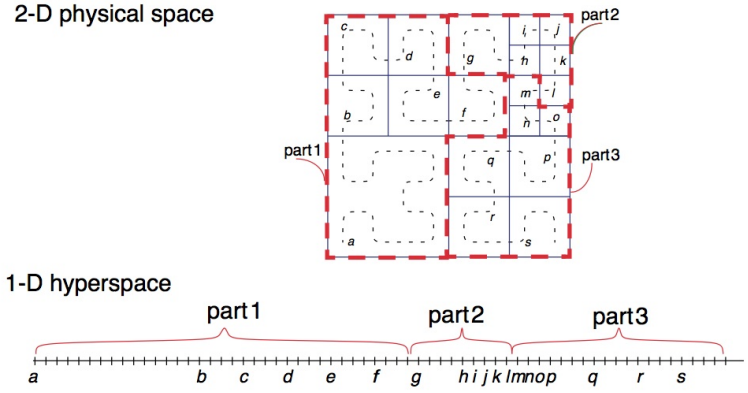

Figure 30: Space Filling Curve and octree representation; reprint from [37]

Sets of new cells can be added or removed at arbitrarily positions to the linked list to allow for dynamic memory management.

As it can be seen from Figure(30), the whole domain can be divided into subdomains, following the order of the SFC, containing the same number of cells, these subdomains are called *blocks*. The purpose of the block is to

define regions with equal distribution of computation workloads which can be later used for parallel computing and to treat regions of the whole domain separately according to their refinement needs. To manage efficient computation, each block contains relevant information about the connectivity of the cells in it, and each cell contains relevant informations about its neighbors as well as access to information regarding the flow solution in its centroid, and relevant geometric information about itself.

The HAMR process is performed in two main steps. The first step of the HAMR is build an initial *Block Frame Work* (BFW) based on the initial geometry and solution at each time step.

Building the BFW is achieved in five steps

1- Generation of a uniform mesh of empty blocks for the whole domain. Blocks are said to be empty because at this point they contain no geometrical or flow information.

2- Use of a search algorithm to identify and tag the blocks that intersect the flow domain.

3- Create an internal cell mesh in each tagged block. At these point the mesh is uniform inside all blocks.

4- Set initial flow solution in the obtained mesh.

5- Set boundary conditions for the cell faces.

The generation of the BFW is a simplified version of the whole HAMR process where refinement occurs only once in the tagged blocks and the final mesh, internal to each block, is uniform in size.

Once the BFW is defined the second step of the HAMR takes place. Using a search algorithm blocks are tagged for further refinement based on two criterions

1- A block is tagged as a result of intersecting with at least one segment of the FE partition.

2- A block is tagged as a result of having a more than one refinement level less than any of its neighboring blocks.

Because each cell is identified by a Morton key, the availablility width of the key determines the achievable resolution for refinement.

After each refinement, the second step of the HAMR process starts. This is performed until no further refinement can be performed based on the achievable resolution defined by the length of the Morton key.

After the refinement process is completed, the cells that intersect any segment of the FE triangulation of the airbag are tagged by the search algorithm and the exact geometry of the cut-cells is computed, obtaining a representation of each cut-cell as shown in Figure(5)

In practice the search algorithm is performed in three stages, each with different levels of accuracy. The accuracy of the stages of the search algorithm vary from finding list of possible FE segment intersecting each block, to finding the actual FE segment intersection with each cell.

To avoid making the HAMR a too intense computational process, not all three stages of the search algorithm are performed at each time step and refinement loop. For instance to build the initial BFW only the first level of accuracy of the search algorithm is used.

5 Cut cell construction, problem description

In this section an in-depth description of the search algorithm is presented with emphasis in the last stage where the exact geometry of the cut-cells is obtained.

The process to obtain the final cut-cells, as shown in Figure(5), is drafted in [29] and its implementations is presented in [34].

This section is based on the before mentioned [29] and [34], as well as in communications with Ir. E.H. Tazammourti.

The purpose of the search algorithm is to classify the cells into, active cells, inactive cells and cut cells, as well as to build the exact geometry of the cut-cells for the FV solver. The search algorithm is divided into three stages

- *Global search level.* This search determines a list of candidate segments that may intersect each bock, see Section(4.2) .
- *Euler search.* This search determines a set of segments that may intersect every cell.
- *Exact Geometry.* This search determines all intersections between Euler cells and FE-segments using the intersection candidate list. After that the face polygons, boundary faces and polyhedrons of each cut cell are obtained.

5.1 Global search level

The Cartesian cut-cell methodology used in *MADYMO*, along with the Arbitrary Lagragian-Eulerian formulation for the coupling between the airbag interface and the flux, results in the use of two different meshes, the Euler mesh, used for the flow calculations and the FE-mesh, used for tracking the structure of the airbag. The Euler mesh is formed by the hexahedral cells that divide the whole domain and the FE mesh is formed by the triangular segments which represent the airbag surface.

During the solution process these two meshes share information. The Euler mesh sends forces resulting from the flow to the FE-segments, and the FE-mesh sends geometric information to the Euler mesh to build the fluxes required by the FV solver.

During the Global search level, the FE-mesh uses the bounding box of the FE-segments to analyze the intersection of the bonding box with the Block

Frame Work (BFW), see Section(4.2), of the Euler mesh. During this step the bounding box of the FE-segments if defined with an off-set. As a result, the final list of segments intersecting the BFW is constituted by candidate segments, not the final ones. The off-set is used so that this stage of the search algorithm does not have to be repeated at every time step, it is only performed every time the displacement of the FE-segments is larger than the off-set.

The main topological tool used at this stage is verifying the intersection of two boxes. As a result of this Global search a list of possible segments intersecting each block is creates and stored in a list called *sblock*. For each block on the Euler mesh an *sblock* is created. Each block receives its own *sblock* to perform the next level of search. The process is represented in Figure(31a)

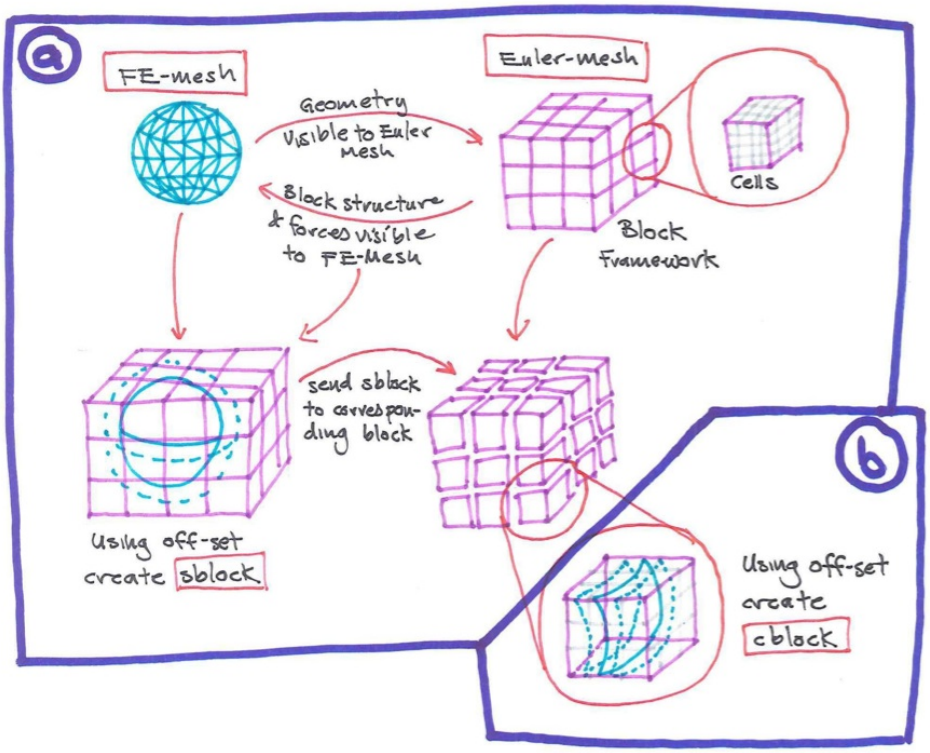

Figure 31: global search level

At the end of the global search each block which intersects the flow region, active block, is tagged and each block receives a list of possible segments intersecting it, which is called *sblock*

5.2 Euler search level

The Euler search level is performed in the same way as the Global search but at block level.

For each block in the Euler mesh, the bounding box of each segment in its corresponding list *sblock* is verified for intersection with each one of the cells in the block.

To reduce computational cost, this search level is performed using an off set in the bounding box of the segments. This result in a list called *cblock* of possible segments intersecting each cell in the block. The process is represented in Figure(31b).

At the end of the Euler search, each cell contains a list *cblock* of possible segments that might intersect it. At this stage the active blocks and cut-block are known as well as the active cells and cut-cells. For the cells also the totally immersed boundary faces are known, but the cell faces neighboring cut-cells still need to be classified. The *cblock* for each cut-cell contains the necessary information to perform the *Exact Geometry* search level.

5.3 Exact Geometry

As a result of the the two previous levels of search, each cut-cell has a list, *cblock*, containing a set of candidate segments intersecting it. The purpose of the *Exact Geometry* search level is to detect if a certain segment of the list of candidates really intersect the cell and, if so, to calculate the intersection points to construct the geometry of the cut-cell.

To test wether a certain segment intersects a cell, it is quite straight forward to think of a verification process in terms of geometry. For example, to calculate the intersection point of a certain segment's edge with one of the cell's faces one can think of solving the system for the intersection of the plane containing the cell's face and the line containing the segment's edge. These geometrical analysis also builds the intersection point in the process. The problem with geometrical methods is that they require the construction of "new geometry", called constructors, which have different precision than

the given data, and are subject to round off errors which at the end result in an unknown accuracy for the intersection points. [1].

Considering the problems that arise from solving the intersection problem in terms of geometrical descriptions, it is desired to first build topological tools to verify the actual occurrence of an intersection and if such intersection really happens then to build the intersection point using geometrical tools. This way the use of intersection constructors is kept to a minimum and so their effect on accuracy and use of computational resources. After all the intersection problem is a question of topology, not of geometry.

To achieve this, three topological tests are used

- *Point test.* For each FE node find in which cell it resides, see Figure(32a).
- *Edge test.* For each FE edge test wether it intersected by a certain cell's face, see Figure(32b).
- *Slice test.* For each FE segment test wether it is intersected by a certain cell's edge, see Figure(32c).

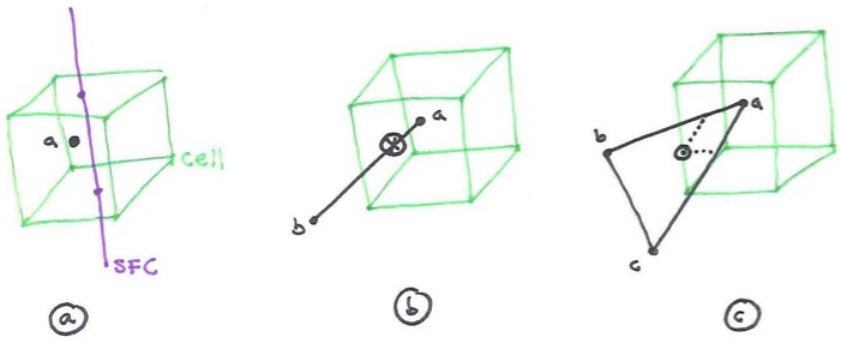

Figure 32: Exact Tests

In order to understand how these tests work, the concept of the *signed volume* for a tetrahedron needs to be known.

5.3.1 Signed Volume of a Simplex

A simplex is the generalization of a triangle to n-dimensions. The simplex in 3 dimensions is the tetrahedron.

A known property for the simplex is the computation of its volume in determinant form. This property states that the volume $V(T)$ of the simplex T with vertices $(v_0, v_1, \dots v_d)$ in d-dimensions is:

$$V(T_{v_0 v_1 \dots v_d}) = \frac{1}{d!} \begin{pmatrix} v_{0_0} & v_{0_1} & \dots & v_{0_{d-1}} & 1 \\ \dots & \dots & \dots & \dots & \dots \\ v_{d_0} & v_{d_1} & \dots & v_{d_{d-1}} & 1 \end{pmatrix} \tag{7}$$

which, for a tetrahedron $T_{a,b,c,d}$ reads:

$$V(T_{v_0 v_1 \dots v_d}) = \frac{1}{3!} \begin{pmatrix} v_{a_0} & v_{a_1} & v_{a_2} & 1 \\ v_{b_0} & v_{b_1} & v_{b_2} & 1 \\ v_{c_0} & v_{c_1} & v_{c_2} & 1 \\ v_{d_0} & v_{d_1} & v_{d_2} & 1 \end{pmatrix} \tag{8}$$

This volume is positive if the triangle $\Delta_{a,b,c}$ forms a counterclockwise circuit when viewed form a point located on the side of the plane defined by $\Delta_{a,b,c}$ which is opposite from d, *c.f* [1], see Figure(33)

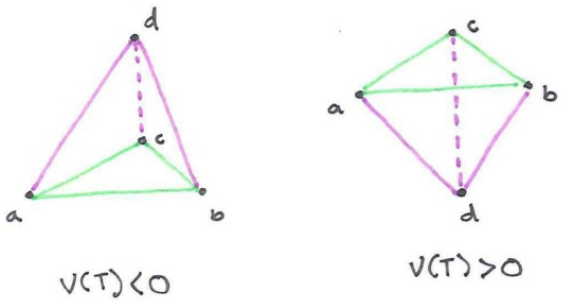

Figure 33: Signed Volume Property

The side of the point d with respect to the plane defined by $\Delta_{a,b,c}$ can be determined by the sign of the volume obtained vis the determinant. If this volume is zero the point d is coplanar with a, b, c.

Using the signed volume property, two tests can be defined, the *pierce test* and the *Inside test*

5.3.2 Pierce test

This test determines weather a line segment between two points a and b pierces trough the plane defined by a triangle $\Delta_{0,1,2}$.

Applying the signed volume test to the triangle $\Delta_{0,1,2}$ and the segment $\overline{ab}$, see Figure(34), $\overline{ab}$ crosses the pane if and only if the signed volumes $V(T_{012a})$ and $V(T_{012b})$ have opposite signs

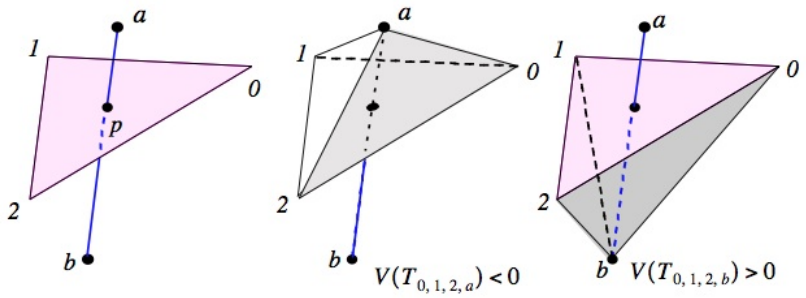

Figure 34: Pierce test, reprint form [1]

5.3.3 Inside test

Once it has been verified that the segment $\overline{ab}$ indeed pierces through the plane where the triangle $\Delta_{0,1,2}$ resides, what need to be determined is if the segment pierces inside the triangle.

According to the signed volume property, piercing occurs inside the triangle if the the volume of the tetrahedrons connecting the end points of segment $\overline{ab}$ with the end points of the edges of the triangle $\Delta_{0,1,2}$ have all the same sign, that is if:

$$[V(T_{a,1,2,b}) < 0 \wedge V(T_{a,0,1,b}) < 0 \wedge V(T_{a,2,0,b}) < 0]$$

or

$$[V(T_{a,1,2,b}) > 0 \wedge V(T_{a,0,1,b}) > 0 \wedge V(T_{a,2,0,b}) > 0]$$

see Figure(35) for a graphic representation of this test.

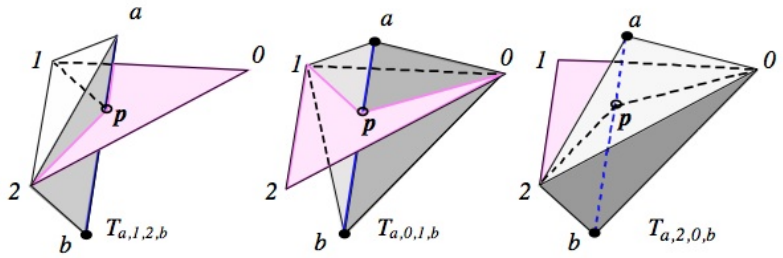

Figure 35: Pierce test, reprint form [1]

5.3.4 Topological Characteristics

To obtain the topological characteristics of a cut-cell the first step is to apply the *Point tests* to determine which of the candidate nodes resides inside it.

The *Point test* transfers the point coordinates in terms of the SPF index, using this information the cell where the edge resides can be known, see Figure(32a).

Once it is known the cell inside which a node resides, the *Edge test* is applied to determine which cell face is pierced by the edges containing such node as an end point.

For each of the edges containing the node and for each of the cell's faces, the *Edge Test* is performed in two stages. First the *pierce test* is used to determine which of the cell face's planes the edge pierces, see Figure(36a). Second, the *(*inside test) is performed to determine if the segment pierces the plane inside the cell's face, see Figure(36b).

The last thing that needs to be determined is which of the triangular segments of the FE-mesh cut each of the Euler cell's edges. This is determined using the *Slice test*.

The *Slice test* is performed in two stages. First the *pierce test* is performed to determine if a certain Euler cell's edge pierces trough any of the segment's planes, see Figure(37a). Second, the *inside test* is performed to determine if the cell's edge pierces inside the segment, see Figure(37b).

Degenerate cases There exist six cases where the *Point test*, *Pierce Test* or *Slice test* may fail, [29]

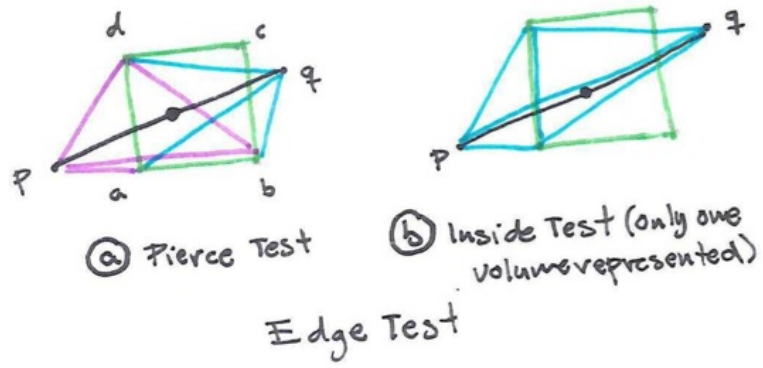

Figure 36: Edge test,

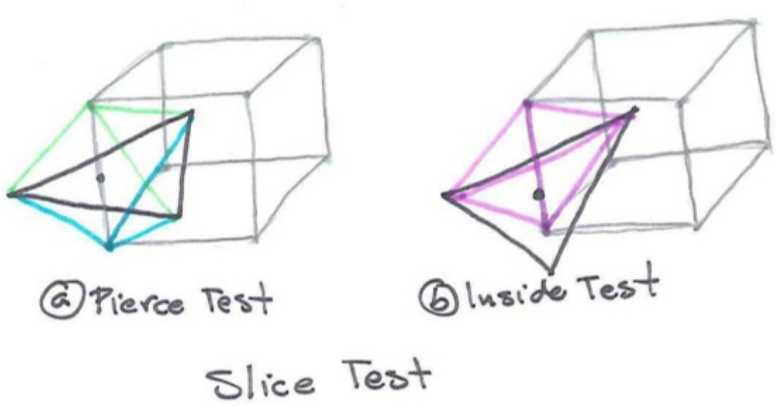

Figure 37: Slice test,

1- FE-node coincides with a face, see Figure(38a).

2- FE-node coincides with a cell edge, see Figure(38b).

3- FE-node coincides with cell vertex, see Figure(38c).

4- FE-edge intersects with a cell edge, see Figure(38d).

5- FE-edge intersects with a cell vertex, see Figure(38e).

6- FE-segment intersects with a cell vertex, see Figure(38f).

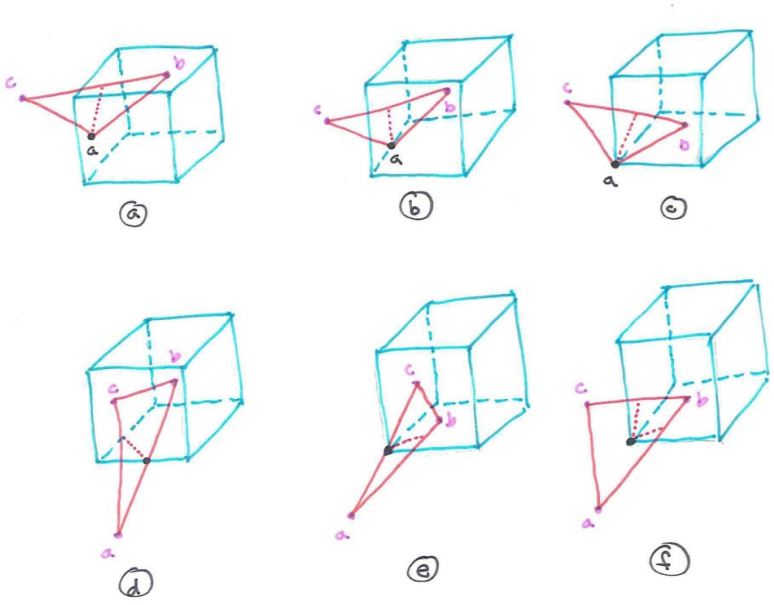

Figure 38: Degenerate cases

The first three degenerate cases cause trouble with the *point test*. The test is unable to determine the cell to which the problematic point belongs. To solve this problem a *Tie breaking algorithm* can be used, [1], [29].

Additionally, when cases 1, 2 or 3 occur the *Edge test* concludes no intersection because one of the signed volumes is zero. When cases 2 and 3 are found also the *Slice test* fails because one of its components gives a zero volume.

When cases 4 and 5 occur the *Edge test* fails because one of the components of the *Inside test* has a zero volume.

When case 6 occurs the *Edge test* fails because one of the components of the *pierce test* has zero volume, [29].

5.3.5 Geometrical Characteristics

After applying the above mentioned tests, for each cell face the FE-edges intersecting it are known, as well as the FE-segments cutting each of the cells' edges. The points where an FE-edge pierces a cell's face and the points where a cell's edge intersects a segments are known as *pierce points*. To build the exact geometry of the cut-cells the *pierce points* need to calculated.

Geometrical constructors were originally rejected because the problem of wether or not an edge pierces a plane inside a certain region is a problem of topology and not of geometry. Once the existence of *pierce points* have been established geometrical constructors should be used to generate the actual geometry of the pierce *pierce points*, [1].

6 Problem definition and research questions

As a result of the *search algorithm*, described in detail in the last section, all blocks of the Euler-mesh are classified as active, if they belong completely to the flow region, inactive if they are outside the flow region, or cut-block if the interface of the airbag intersects any of the cells belonging to the block.

In the same way, each cell is classified as active, inactive or cut-cell. The cell faces are also classified, the cell faces connecting two active cells are classified as flow faces. The faces connecting an active cell and a cut cell are classified as cut faces but this classification takes place when the cut-cell's exact geometry is obtained. As a result of the unresolved geometric degeneracies the faces connecting active cells to cut-cells can not be classified entirely.

In the actual cut-cells implementation used in *MADYMO* two issues need to be addressed

1- How to solve the degenerate cases? The aim of this question is to define an algorithm capable of determining consistently the existence of a pierce point for the degenerate geometries.

2- How to determine which region in a cut-cell belongs to the flow and which out-side? The only known fact about the cut-cells, in general, is their classification as such, but not which regions in them belong to the flow and which outside.

Thinking of a test to define the inner and outer regions of a cut-cell seems easy in terms of any of the already mentioned tests for the *Point in Polyhedron Problem*. For example the line crossing test could be used taking any test point in any of the two regions of a cut-cell and the centroid of any of the active cells, tracing a line segment between those two points and counting the number of intersections would reveal the answer. But soon problems arise, for example:

- How to chose the test point in the cut-cell?
- How does the choice of the active cell affects the accuracy and what role does connectivity plays?
- What happen if the test point in the cut-cell is too close to the airbag's boundary?

This problems become more complicated taking into account the fact that the boundary in the problem at hand is a moving, flexible boundary.

The two above mentioned questions and the derived subquestions are the research topic that this work will address.

7 Looking for solutions

In this section a description of various methods to solve the problem is presented. In the first part a brief explanation on the actual solution method implemented in MADYMO is presented along with an example of a degenerate geometry for which it fails.

The notation in this section follows the notation used in [1], see Figure(39).

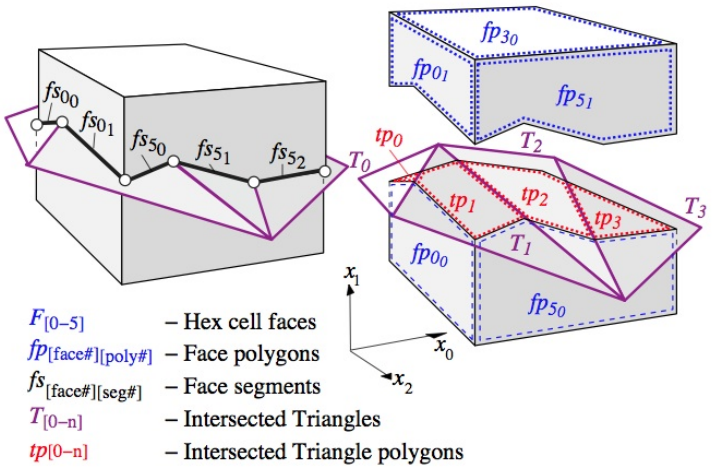

Figure 39: Anatomy of a cut cell; reprint from [1]

The implementation in MADYMO is already capable of determining each of the polyhedrons obtained after the cutting process for each cut-cell. The aim of the process to obtain the exact geometry is to determine accurately the area of the face polygons inside the airbag to make accurate calculations of the flows.

For the cut-cells, the face polygons are known as a list of their vertices in a counter-clockwise direction. An extreme geometrical case, known as collapsing FE-segments, occurs when the airbag folds causing two neighboring triangular FE-segments to become almost co-planar. For now the problem of collapsing FE-segments will be ignored, see Figure(40).

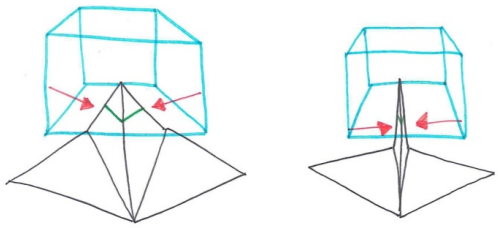

Figure 40: Collapsing triangles

7.1 Implemented solution method

The airbag is a simple polyhedron, which means that it is topologically equivalent to a sphere. The normal to each triangular FE-segment is known, thus, if a transversal cut of the airbag is performed by a plane, a simple polygon, topologically equivalent to a circle, is obtained. The normal to each of the polygon's edges is known, by projection of the triangular segments normal to the cutting plane, and also the counter-clock wise tangent direction to traverse the polygon.

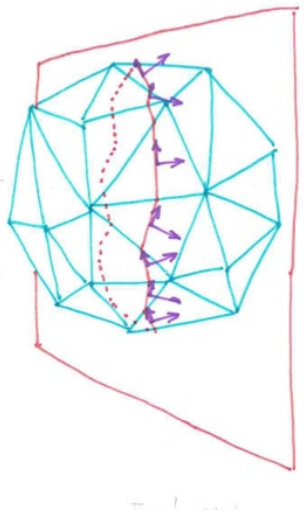

Figure 41: Equivalence to two-dimensional problem

The implemented method in MADYMO is based on the *signed volume test* for the simplex, described in Section(5.3.1). The assumption made for this method is that a point q is inside a simple polyhedron if the polygon, obtained by cutting the polyhedron by any plane containing the point q, is entirely traversed in counter-clockwise direction viewed as standing in q, see Figure(42).

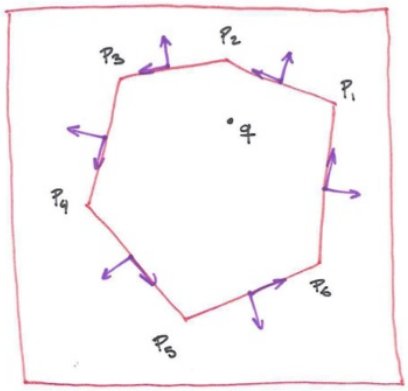

Figure 42: Point q inside polygon

For each cut-cell the problem can be stated equivalently: the normal to each triangular FE-segment is known and thus, by projection, the normal to the face segments in each cell's face is know. The problem of determining the inside/outside polyhedron for the cut-cells can be equivalently solved by determining the inside/outside polygons to each cell face, this turns the three-dimensional problem into a two-dimensional problem.

The traversing direction of the *face segments* is important for consistency in the *signed volume test*. For each cut-cell face, a vertex v is determined to be inside or outside based on the sign of the simplex formed by the vertex being tested and two of the *face segments* edges, p_1 and p_2, always preserving the direction of the *face segment* $\overline{p_1 p_2}$ according to the counterclockwise tangent defined by the normal.

If the signed area of the triangle is positive then the vertex v is inside because the triangle is defined counter-clock wise direction, and if the signed area is negative, the vertex v is outside. An example is presented in Figure(43)

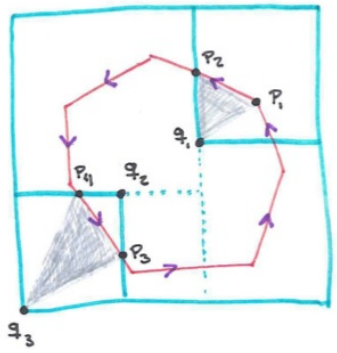

Figure 43: Point q_1 is inside with respect of the triangle $q_1P_1P_2$, point q_3 is out with respect to triangle $q_3P_4P_3$

.

In order to determine the inside/outside status of each *face polygon*, MADYMO needs to determine the inside/outside status of at least one test point belonging to such polygon. The test point is chosen systematically to be a cell-face vertex belonging to the *face polygon* that does not coincide with a *face segment* edge, see Figure(43)

The *signed volume* test fails for a *face polygon* when no vertices of the cut-cell face belong to such polygon. In such cases no test point can be chosen or the status of the coinciding cell vertex and *face segment* edge is undetermined, see Figure(44)

7.2 Solution Idea 1: Consistency in traversing direction.

The key point behind this method is that if two simple polygons sharing an edge are both traversed in a counter-clock wise direction from the point of view of a point inside each polygon, then the direction at which the shared edge will be traversed will be contrary for each polygon, see Figure(45).

Based on this, and because for now the problem of collapsing FE-segments is being ignored, the status of a *face polygon* can be determined by consistency of the traversing direction. Each *face polygon* is traversed in counter-clock wise direction and if the direction is contrary to the traversing direction

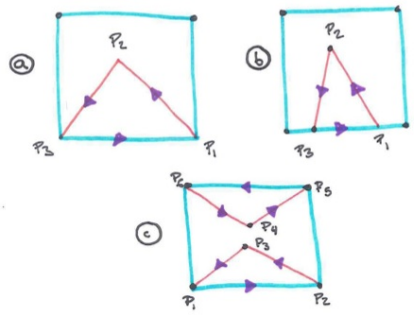

Figure 44: No test point can be chosen sistematically

.

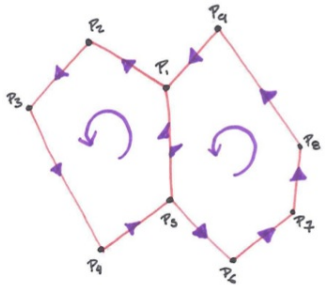

Figure 45: Simple polygons with coinciding edge.

.

of the *face segments*, dictated by the counter-clock wise orthogonal vector to the normal, then the polygon is determined to belong outside the flow region. Figure(46) presents an example.

In Figure(46 a) the cell face is cut into the polygons $P_1 = p_1p_2p_3$, $P_2 = p_1p_3p_2p_5p_4$ and $P_3 = p_6p_4p_5$, for this case P_1 and P_3 are consistent with the directions at which the *face segments* are traversed, while P_2 is not. It is concluded that the interior of P_1 and P_3 belong to the flow region while the interior of P_2 belongs outside.

In Figure(46 b) the cell face is cut into the polygons $P_1 = p_1p_2p_3$, $P_2 =$

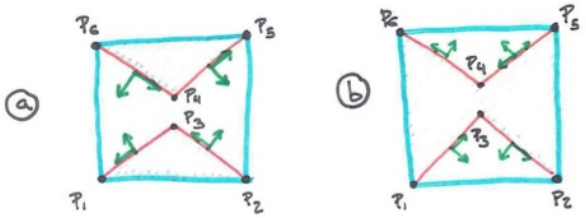

Figure 46: Traversing direction consistency.

.

$p_1p_3p_2p_5p_4$ and $P_3 = p_6p_4p_5$, for this case P_2 is consistent with the directions at which the *face segments* are traversed, while P_1 and P_3 are not. It is concluded that the interior of P_2 belongs to the flow region while the interior of P_1 and P_3 belong outside.

A case which cannot be treated directly by this method is the case of a cell face where the *face segments* divide the face into non-simple polygons, see Figure(47 a). To solve this case an auxiliary "double"-edge can be defined between one of the interior-polygon's edges and a cell face vertex, this transforms the non-simple polygon into a simple one allowing to use the proposed method, see Figure(47 b).

This case is not only problematic for this method, it is problematic also for the actual implementation used in MADYMO and is treated separately.

The *Consistency in traversing direction* method requires that the polygons into which each cut-cell face is divided are known in counter-clock wise direction, the traverse direction for each *face segment* in the counter-clock wise orthogonal direction to the normal, and a way to verify the equality between the directions in which polygon is traversed and the direction at which the *face segments* are traversed.

7.3 Solution Idea 2: Consistency in test point selection.

Other methods more geometric in nature can be though where the problem lies on achieving a systematic and consistent way of choosing a test point.

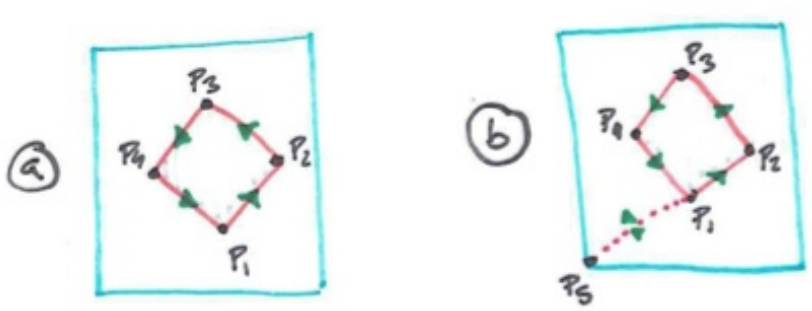

Figure 47: Non-simple polygon degeneracy.

.

Those methods will be discussed in this subsection.

References

[1] M. J. Aftosmis, *Solution Adaptive Cartesian Grid Methods for Aerodynamic Flows with Complex Geometries*. von Karman Institute for Fluid Dynamics Lecture Series 1997-02 March, 1997

[2] G. Yang, D. M. Causon, D. M. Ingram, *Calculation of compressible flows about complex moving geometries using a three-dimensional Cartesian cut cell method*. Int. J. Numer. Meth. Fluids 2000; 33: 11211151

[3] H. Bandringa, *Immersed boundary method*. Master Thesis in applied mathematics, University of Groningen, August 2010

[4] L. Dubuc, F. Cantariti, M. Woodgate, B. Gribben, K. J. Badcock and B. E. Richards, *A grid deformation technique for unsteady flow computations*. Int. J. Numer. Meth. Fluids 2000; 32: 285311

[5] F. Cirak, R. Radovitzky, *A Lagrangian-Eulerian shell-fluid coupling algorithm based on level set methods*. Computers and structures, Elesevier 2005, 83

[6] D. Hartmann, M. Meinke, W. Schrder *A strictly conservative Cartesian cut-cell method for compressible viscous flows on adaptive grids*. Comput. Methods Appl. Mech. Engrg. 200 (2011) 10381052

[7] R. Mittal, G. Iccarino, *Immersed boundary method*. Annu. Rev. Fluid Mech. 2005. 37:23961

[8] S. Kang, G. Iaccarino, F. Ham, P. Moin, *Prediction of wall-pressure fluctuation in turbulent flows with an immersed boundary method*. Journal of Computational Physics 228 (2009) 67536772

[9] *Advanced immersed boundary cartesian meshing technology in FloEFD*. Mentor Graphics Whitepaper 2011, www.mentor.com

[10] M. Meyer, A. Devesa, S. Hickel, X.Y. Hu, N.A. Adams *A conservative immersed interface method for Large-Eddy Simulation of incompressible flows*. Journal of Computational Physics, 229 (2010) 63006317

[11] J. Peteker *Point-in-Polygon Detection*. Bachelor Thesis University of California, Merced, April 2010

[12] P.G. Tucker, Z. Pan, *A Cartesian cut cell method for incompressible viscous flow*. Applied mathematics modeling, 24, 2000, 591-606

[13] D. De Zeeuw, K. Powell *An adptive refined cartesian mesh solver for Euler equations*. J. of Comp. Physics, 104, 56-69, 1993

[14] E. Haines *Poin-in-Polygon strategies*. Graphics Gems IV, ed. Paul Heckbert, Academic Press, 1994, p. 24-46.

[15] D. De Zeeuw, K. Powell *Efficient and consitent algorithms for determining the containment of point in polygons and polyhedra*. Eurographics Association, Elsevier Science Publications, 1987, 423-437

[16] B. Zalik, I. Kolingerova *A cell-based point-in-polygon algorithm suitable for large sets of points*. Computers & Geosciences 27 2001, 11351145

[17] J. Petker, *Point-in-Polygon Detection*. Batchelor of science Thesis, The University of California, Merced, April 2010

[18] M. Galetzka, P. O. Glauner *A correct even-odd algorithm for the point-in-polygon (PIP) problem for complex polygons*.

[19] K. B. Salomon *An efficient point-in-polygon algorithm*. Computers & Geosciences 4 1978, 173-178

[20] G. Taylor *Point-in-polygon test*. survey review 32, Department of Surveying, University of Newcastle upon Tyne October 1994, 479-484

[21] J. O'Rourke *Computational geometry in C*. Cambridge university press second edition

[22] K. Hornmann, A. Agathos, *The point in polygon problem for arbitrary polygons*. Computational Geometry 20 (2001), 131144

[23] Cundy, H. and Rollett, *Mathematical Models*. Tarquin Pub. 3rd ed., Stradbroke, England, 1989

[24] J.A. Sethian, *Fast marching methods and level set methods for propagating interfaces*. von Karman Institute Lecture Series, Computational Fluid Dynamics (1998)

[25] J.A. Sethian, *Level Set Methods: An Act of Violence*.

[26] S. Osher, R. Fedkiw *Level Set Methods and Dynamic Implicit Surfaces*. Springer, (2003)

[27] M. Arienti, P. Hung, E. Morano, J. E. Sheperd *A level set approach to Eulerian-Lagrangian coupling*. Journal of Computational physics, (2003), Number 185, pages 213-251

[28] BMN, *System Requirements, Global search*. TASS technical reports May 15, 2008

[29] M.R. Lewis, B.M. Neelis, *System Requirements - Exact Geometry*. TASS technical reports May 15, 2008

[30] H. Tazammourti, *System Requirements - MPP*. TASS technical reports September 4, 2006

[31] M.R. Lewis, *System Requirements - Euler flow solver*. TASS technical reports October 7, 2008

[32] H. Tazammourti, *System Requirements - cell merging*. TASS technical reports September 4, 2004

[33] M.R. Lewis, *System Requirements - Fluid Structure Coupling.* TASS technical reports November 25, 2008

[34] M.R. Lewis, *Design-Exact Geometry.* TASS technical reports July 3, 2007

[35] H. Tazammourti, *Design Document- Gasflow2-AMR and DataStructure.* TASS technical reports November 9, 2006

[36] M.R. Lewis, *System Requirements - Multiple species.* TASS technical reports April 23, 2008

[37] R. Schmehl, *Theory- AMR concepts and data structure.* TASS technical reports December 19, 2006