6th Workshop on Numerical Methods in Applied
Science and Engineering (NMASE 07)
Vall de Núria, 10 a 12 de enero de 2007

AUTOMATIC GENERATION OF QUADRILATERAL STRUCTURED MESHES USING LINEAR PROGRAMMING AND TRANSFINITE INTERPOLATION

E. Ruiz-Gironés and J. Sarrate

Laboratori de Càlcul Numèric (LaCàN),
Departament de Matemàtica Aplicada III,
Universitat Politècnica de Catalunya,
Jordi Girona 1-3, E-08034 Barcelona, Spain
e-mail: {eloi.ruiz,jose.sarrate}@upc.edu,
web: www-lacan.upc.edu

Key words: Finite element method; mesh generation; submapping; structured quadrilaterals; linear programming; transfinite interpolation.

Abstract. *This paper presents an implementation of a structured quadrilateral algorithm called submapping. The meshing procedure splits the geometry into patches and then meshes each patch separately preserving the mesh compatibility between patches via an linear integer problem. The submapping algorithm can only be applied to geometries such that each angle between consecutive edges is, approximately, an integer multiple of $\pi/2$.*

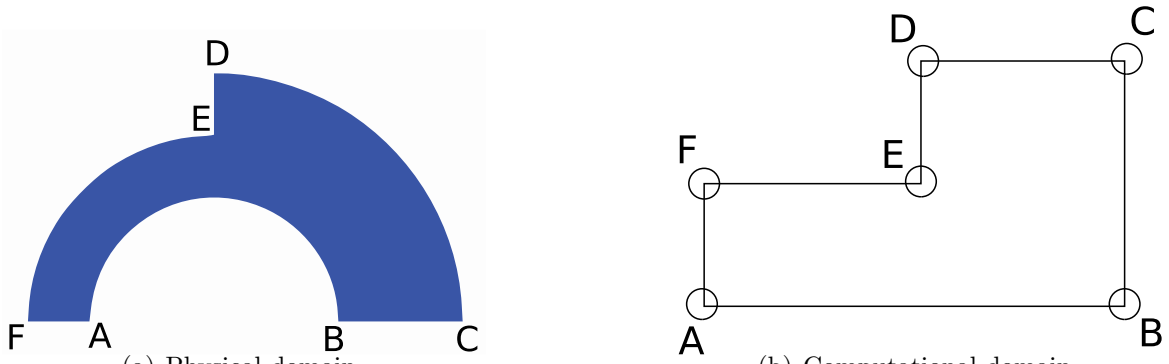
1 Introduction

A wide range of problems in applied science and engineering can be simulated by partial derivate equations (PDE). In the last decades, one of the most relevant techniques to solve a PDE is the Finite Element Method (FEM). It is well known that a good quality mesh is required in order to obtain an accurate solution. Hence, the construction of a mesh is one of the most important steps. In this article, we present an implementation of the submapping method [1]. Given a geometry, this method generates a mesh of structured quadrilaterals. The basic idea is to decompose the entire geometry into patches that are logically equivalent to a quadrilateral. Then, each patch is meshed using a structured quadrilateral mesh in such a way that all the compatibilities between patches are verified via a linear integer problem. In our implementation, each patch is meshed using the transfinite interpolation method (TFI) [2]. The submapping method can only be applied to surfaces such that the angle between two edges is, approximately, an integer multiple of $\pi/2$. Although it may appear that it can be a hard constrain, in fact, there is a wide number of surfaces that satisfies this condition.

2 Submapping

As previously mentioned, the submapping algorithm is a method to construct a structured mesh of quadrilaterals in a given geometry. This method relies on the condition that every angle between two consecutive edges is approximately integer multiple of $\pi/2$. Therefore, we will use this condition through all the algorithm. The basic idea of the submapping method is to divide the geometry into patches logically equivalent to a quadrilateral and then mesh each patch separately preserving the mesh compatibility via a linear integer problem.

In order to obtain a structured mesh, we will assume that there exists a representation of the geometry in which all the edges are horizontal or vertical. We define this geometry as the computational space, whereas the initial geometry is defined as the physical space. We will use the computational space to split the initial geometry into patches. Figure 1 shows the physical domain and the corresponding computational domain of a given geometry.



(a) Physical domain (b) Computational domain

Figure 1: Physical and computational domains of a geometry

The submapping algorithm is composed of the following five steps:

1. Characterization of nodes.
2. Characterization of edges.
3. Discretization of the boundary and construction of the computational space.
4. Subdivision of the geometry into patches that can be meshed by the TFI algorithm.
5. Discretization of each patch using the TFI method.

2.1 Characterization of nodes

Each vertex of the initial geometry is classified according to the angle between the two edges adjacent to the vertex. Recalling that each angle is approximately an integer multiple of $\pi/2$, the classification of a vertex is made as follows:

- **Side**: the angle between edges is near 0.
- **End**: the angle between edges is near $\pi/2$.
- **Reversal**: the angle between edges is near π.
- **Corner**: the angle between edges is near $3\pi/2$.

Figure 2 shows a geometry and the classification of each vertex.

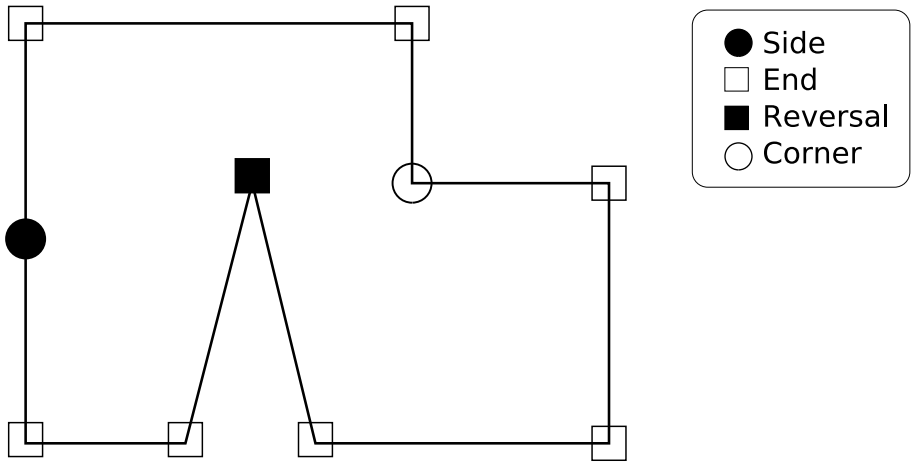


Figure 2: Simple geometry in the physical domain and it's node classification.

Once the vertices are classified, we can do a fast check to ensure that the surface is submappable. If the surface is submappable, the following two conditions are verified:

1. There are at least 4 vertices classified as *end*.
2. Let θ_i be the angle detected in vertex i. Then,

$$\sum_{i=1}^{N_{vertices}} \theta_i = k\frac{\pi}{2}, \tag{1}$$

where $N_{vertices}$ is the number of vertices and k has to be an integer multiple of 4 ($k \equiv 0 \mod 4$).

2.2 Characterization of edges

Each edge of the initial geometry is classified according to its direction in the computational space. As every edge is horizontal or vertical, an edge can be classified as:

- **+i**: the edge is horizontal and goes from left to right.
- **-i**: the edge is horizontal and goes from right to left.
- **+j**: the edge is vertical and goes from down to up.
- **-j**: the edge is vertical and goes from up to down.

Figure 3 shows the representation in the computational domain of the geometry presented in Figure 1 and the classification of each of its edges.

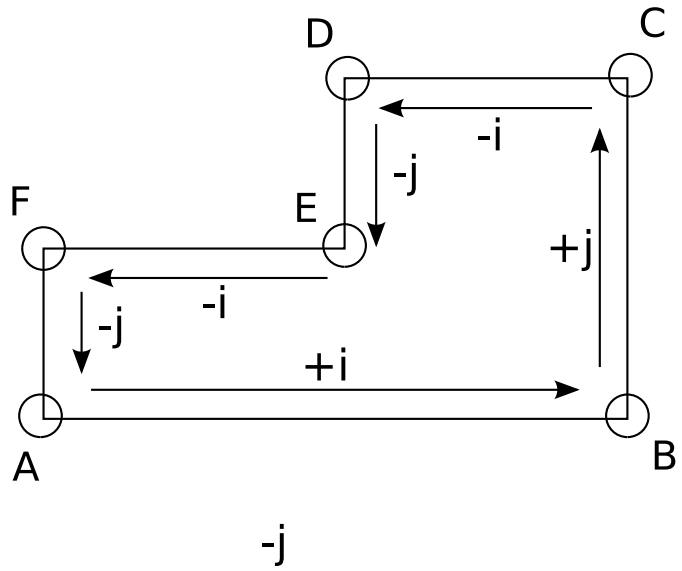


Figure 3: Classification of edges in the computational domain of the geometry presented in figure 1.

Note that the process of edge classification can be accomplished before constructing the computational space. In order to find the edge classification, we will use the classification of vertices in the following manner. We classify the first edge as $+i$, arbitrarily. The classification of the next edge will be induced by the classification of their common vertex. In that way, if the classification of their common vertex is *end*, it means that the next edge is classified as $+j$ because we turned $\pi/2$ on the vertex. For example, in the geometry of Figure 1, we take the first edge as the edge **AB** and we classify it as $+i$. Vertex **B** is classified as *end* and, therefore, the edge **BC** is classified as $+j$. The next vertex is vertex **C** and it is also classified as *end*. It means that we turn $pi/2$ in that vertex and the edge **CD** will be classified as $-i$. The following edges are classified in the same way.

It is worth to notice that when the process of edge classification ends, we should find that the first edge is classified as $+i$. In other case, the vertices have not been correctly classified or there was a mistake while classifying the edges.

Finally, we define the following four sets of edges:

$$\begin{aligned} I^+ &= \{\text{edges classified as } +i\}, \\ I^- &= \{\text{edges classified as } -i\}, \\ J^+ &= \{\text{edges classified as } +j\}, \\ J^- &= \{\text{edges classified as } -j\}. \end{aligned}$$

2.3 Discretization of the boundary and construction of the computational space

In order to construct a structured mesh of quadrilaterals, we must impose the following condition on the edges:

$$\begin{aligned} \sum_{e \in I^+} n_e &= \sum_{e \in I^-} n_e, \\ \sum_{e \in J^+} n_e &= \sum_{e \in J^-} n_e, \end{aligned} \tag{2}$$

where n_e is the number of subdivisions of edge e. If we want to keep the number of nodes to a reasonable number while preserving the compatibility equation (2), we will solve the next linear integer problem:

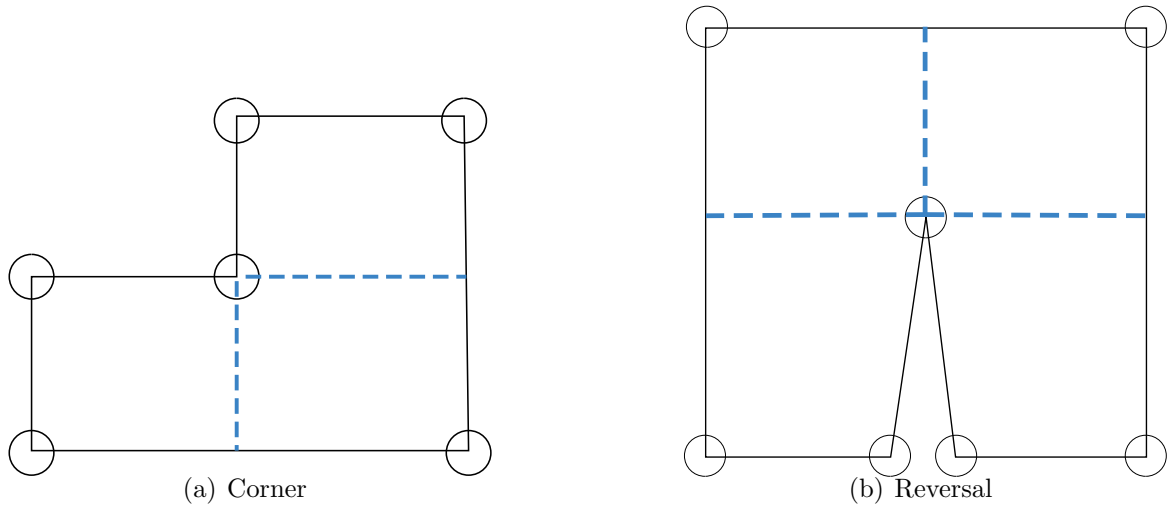
$$\begin{cases} \min \sum_e n_e, \\ \text{subject to:} \\ \sum_{e \in I^+} n_e = \sum_{e \in I^-} n_e, \\ \sum_{e \in J^+} n_e = \sum_{e \in J^-} n_e, \\ n_e \geq N_e, \end{cases} \tag{3}$$

where N_e is a lower bound for the number of elements in the edge e. Note that using these lower bounds, we may impose the desired element size. It is important to notice that the solution of this problem provides us the necessary information to construct a mesh in the boundary of the geometry that allows us to obtain a structured mesh in the interior. The solution of (3) can be computed using the Branch&Bound method [3].

Once the solution of the compatibility problem (3) has been found, we can proceed to build the computational space. Recall that the edges of the computational domain are either horizontal or vertical. In addition, we set the length of an edge equals to its number of intervals. In this way, the element size in the computational space will be the unit. Furthermore, the position of each node in the computational space will have integer coordinates. Finally, it is worth to notice that the compatibility equations (2) are required to create a closed boundary in the computational domain. Therefore, we can check the solution of the compatibility problem when constructing the computational space.

2.4 Subdivision of the geometry into patches

As we have previously noted, the computational space will help us to split the geometry into patches that are easier to mesh. In order to split the geometry, we will use splitting lines. We will impose that splitting lines are horizontal or vertical in the computational space. In addition, we will assume that the splitting lines start in a *corner* or *reversal* node. Therefore, there are two possible splitting lines in a *corner* node and three possible splitting lines in a *reversal* node. Figure 4 shows the possible splitting lines for a *corner* (Subfigure 4(a)) and *reversal* nodes (Subfigure 4(b)).



(a) Corner (b) Reversal

Figure 4: Possible splitting lines for *corner* and *reversal* nodes.

We will choose the shortest cutting edge in the computational domain to split the geometry. Once a splitting line is found, we have to mesh this line and proceed to subdivide the geometry into two pieces. Therefore, we have to know how many elements are in the splitting line. In the computational space, the length of the edges are equal to its number of intervals. Therefore, the number of elements of the splitting edge will be equal to its length. The algorithm is iterated recursively in every piece until there are no *corner* nor *reversal* nodes.

2.5 Discretization of patches

Once we obtain a patch of the geometry that does not contain *corner* or *reversal* nodes, we can mesh it. To this end, we use the transfinite interpolation method (TFI). Since the patches are logically equivalent to a quadrilateral, there exists a mapping between the unit square and the patch we want to mesh. Therefore, we can transfer a mesh in the unit square to the given patch. The TFI method builds that application.

3 Examples

In this section, we present four examples of submapping meshes. The first example presents geometry that does not contain any *corner* nor *reversal* nodes. Thus, the geometry is not subdivided and the mesh is created using TFI method. Figure 5 shows the generated mesh for a section of a tubular geometry.

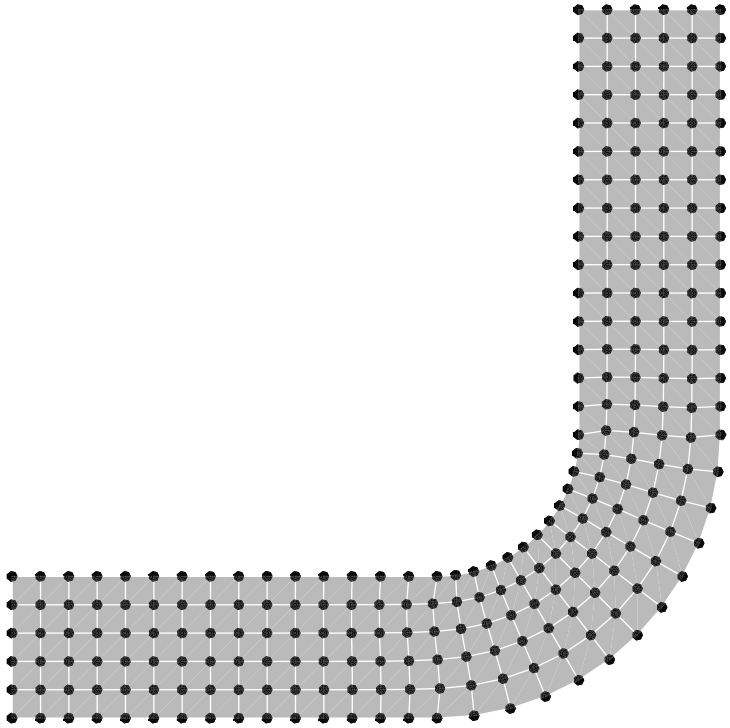


Figure 5: Meshed created in a tubular geometry.

In the second example, a geometry with anode classified as*corner* is considered. In this example, unlike the first one, it is necessary to split the geometry through this vertex in order to simplify the surface. Figure 6 shows the mesh obtained for this geometry.

In the third example, a geometry with a reversal node is presented. Therefore, it is necessary to generate a splitting line through the *reversal* node. Figure 7 shows the geometry and the mesh created using the submapping method. Note that on the *reversal* node, there are four elements attached, while on a *corner* node there are three elements attached.

In the last example, a mechanical piece is considered. This example shows the process automatization of the submapping algorithm. Many splitting lines are automatically created in order to simplify the geometry. Figure 8 shows the geometry and the created mesh by the submapping method.

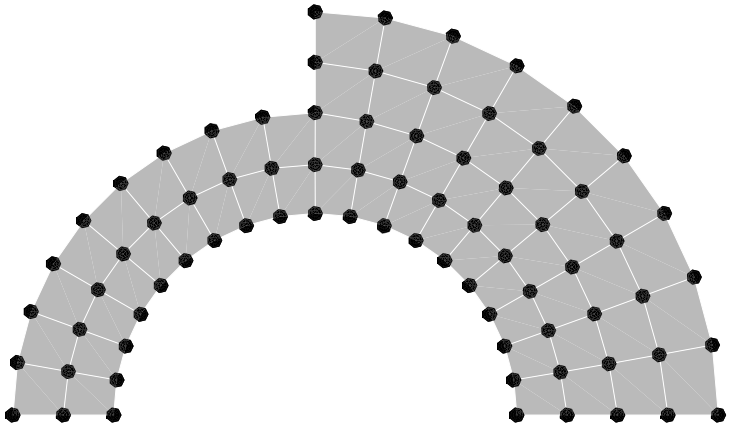


Figure 6: Mesh created in a geometry with a *corner* node.

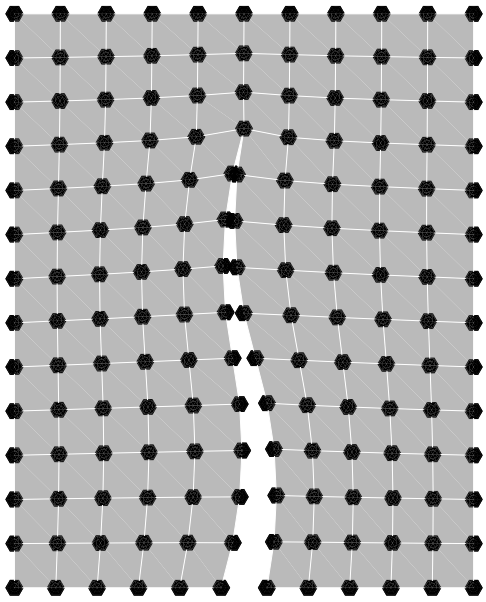


Figure 7: Mesh created in a geometry with a *reversal* node.

4 Conclusion

In this article we present an implementation of the submapping algorithm. This algorithm allows to mesh a given geometry using a structured mesh of quadrilaterals. The idea of the algorithm is to divide the geometry into patches, and then mesh each patch separately preserving the mesh compatibility between them. This compatibility is ensured by solving the linear integer problem (3). Although not every surface can be meshed, wide range of realistic surfaces can be discretized using this algorithm. The main restriction is

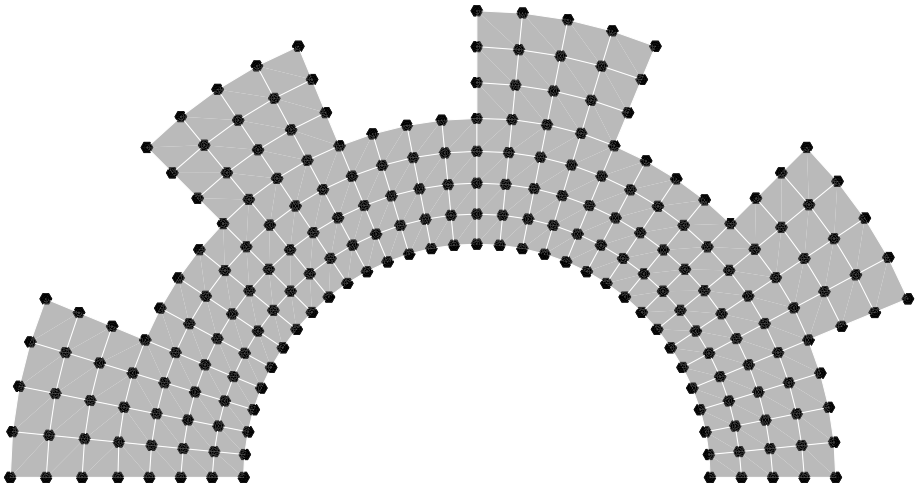


Figure 8: Mesh created in a mechanical piece by submapping method.

that the angles between two edges must be near an integer multiple of $\pi/2$. Finally, it is worth to notice that the submapping algorithm has been successfully implemented in the ez4u environment [4]. In our implementation, we have minimised the linear problem (3) using the GLPK library [5].

The future research will be focused on classifying *fuzzy* angles (angles that can not be correctly detected) and meshing multiply-connected surfaces. Detecting correctly *fuzzy* angles is crucial in the algorithm, because every step is based on this classification. However, if *fuzzy* angles are present, we can classify the in several ways. Among them, we have to select a classification such that equation (1) is verified and it is not *far* from the classification we have previously found. Figure 9 shows a simple geometry with two *fuzzy* angles. To obtain a valid classification of vertices, we can classify these *fuzzy* angles either as *end* and *corner* or as *side* and *side*. Note that for each classification, we will obtain different valid meshes.

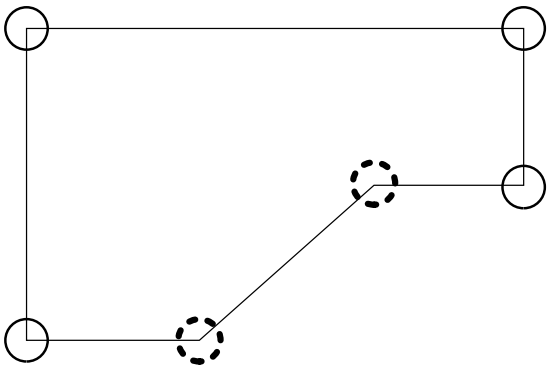


Figure 9: Geometry with two *fuzzy* angles.

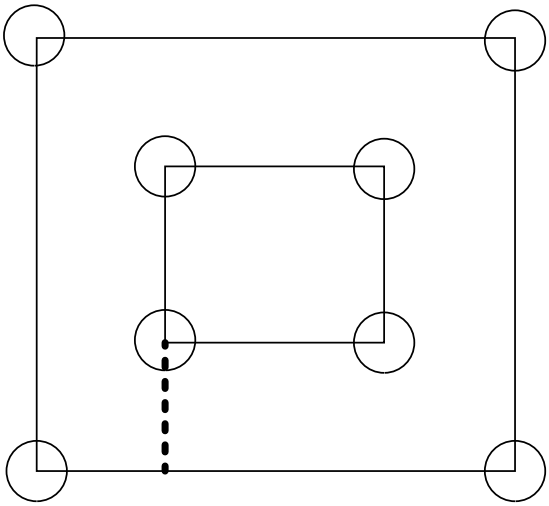


Figure 10: Multiply connected geometry converted to simply connected geometry using a virtual edge (dotted black line).

The algorithm explained in this article is valid for simply connected surfaces. Multiply connected surfaces are defined by more than one loop of edges, therefore, if we want to mesh multiply connected surfaces using the submapping method, first we have to convert the geometry into a simply connected surface. To this end, we will find virtual edges that connects interior boundaries with exterior boundaries. The main problem when looking for virtual edges is that we have to preserve the angles condition, that is, new angles created by virtual edges must be near an integer multiple of $\pi/2$. Figure 10 shows a multiply connected surface, defined by two loops of edges, converted to a simply connected surface using a virtual edge that connects both boundaries.

REFERENCES

[1] White, Roger D. Automatic Quadrilateral and hexaedral meshing of presudo-cartesian geomatries using virtual subdivision. *Department of Civil and Environmental Engineering, Brigham Young University*, august 1996.

[2] Thompson JF, Soni B, Weatherill N. *Handbook of Grid Generation*. CRC Press, 1999.

[3] Schrijver, A. *Theory of Linear and Integer Programming*, John Wiley and Sons, 1998.

[4] X. Roca and J. Sarrate, An interactive mesh generation environment for geometry-based simulations, *5th Workshop on Numerical Methods in Applied Science and Engineering (NMASE 06)*.

[5] GNU linear programming kit (GLPK), *http://www.gnu.org/software/glpk/*.