©Saxe-Coburg Publications, 2008.
Trends in Engineering Computational Technology
B.H.V. Topping and M. Papadrakakis, (Editors)
Saxe-Coburg Publications, Stirlingshire, Scotland, 888-888.

Chapter 0123456789

XXXXXXXXXXXX PROOF XXXXXXXXXXXXXX

# Advances in the Meccano Technique for Adaptive Tetrahedral Mesh Generation

**R. Montenegro[1], J.M. Cascón[2], E. Rodríguez[1], G. Cascón[2] and J.M. Escobar[1]**
**[1] University Institute for Intelligent Systems and Numerical Applications in Engineering**
**University of Las Palmas de Gran Canaria, Spain**
**[2] Department of Mathematics, Faculty of Sciences**
**University of Salamanca, Spain**

## Abstract

This paper introduces a new automatic strategy for adaptive tetrahedral mesh generation. A local refinement/derefinement algorithm for nested triangulations and a simultaneous untangling and smoothing procedure are the main techniques involved. The mesh generator is applied to 3-D complex domains whose boundaries are projectable on external faces of a *meccano* approximation composed of cuboids. The domain surfaces must be given by a mapping between *meccano* surfaces and object boundary.

**Keywords:** tetrahedral mesh generation, adaptive refinement/derefinement, nested meshes, mesh smoothing, mesh untangling.

## 1 Introduction

In finite element simulation in engineering problems, it is crucial to automatically adapt the three-dimensional discretization to geometry and to solution. In the past, many authors have devoted great effort to solving this problem in different ways [1–4], but automatic 3-D mesh generation is still an open problem. Generally, as the complexity of the problem increases (domain geometry and model), the methods for approximating the solution become more complicated. At present, it is well known that most mesh generators are based on Delaunay triangulation and the advancing front technique. On the other hand, local adaptive refinement strategies are employed to adapt the mesh to singularities of numerical solution. These adaptive methods usually involve remeshing or nested refinement [5–8]. Another interesting idea is to adapt simultaneously the model and the discretization in different regions of the domain. A perspective on adaptive modeling and meshing is studied in [9]. The main objective of all these adaptive techniques is to achieve a good approximation of the *real* solution

with minimal user intervention and low computational cost. For this purpose, the mesh element quality is also an essential aspect for the efficiency and numerical behavior of finite element method. The element quality measure should be understood depending on the isotropic or anisotropic character of the numerical solution.

In this paper we present new ideas and applications of an innovative tetrahedral mesh generator which was introduced in [10–12]. This automatic mesh generation strategy uses no Delaunay triangulation, nor advancing front technique, and it simplifies the geometric discretization problem for 3-D complex domains, whose surfaces can be mapped from a *meccano* face to object boundary. The main idea of the new mesh generator is to combine a local refinement/derefinement algorithm for 3-D nested triangulations [6] and a simultaneous untangling and smoothing procedure [13]. The resulting adaptive meshes have an appropriate quality for finite element applications.

The mesh generator starts building a *meccano* approximation formed by cuboids. Then, a coarse and valid hexahedral mesh of the *meccano* approximation is generated. The automatic subdivision of each hexahedron into six tetrahedra produces an initial tetrahedral mesh of the *meccano* approximation. The main idea is to construct a sequence of nested meshes by refining only those tetrahedra with a face on the *meccano* boundary. The virtual projection of the *meccano* external faces defines a valid triangulation on the domain boundary. Then a 3-D local refinement/derefinement is carried out so that the approximation of domain surfaces verifies a given precision. Once this objective is reached, those nodes placed on the *meccano* boundary are really projected on their corresponding true boundary, and inner nodes are relocated using a suitable mapping. As the mesh topology is kept during node movement, poor quality or even inverted elements could appear in the resulting mesh; therefore, we finally apply a mesh optimization procedure.

At present, the refinement/derefinement module is implemented in ALBERTA code [14, 15]. This software can be used for solving several types of 1-D, 2-D or 3-D problems with adaptive finite elements, but we only use the local refinement/derefinement algorithm. It is based on local bisection. Actually, ALBERTA has implemented an efficient data structure and adaption for 3-D domains which can be decomposed into hexahedral elements as regular as possible. Each hexahedron is subdivided into six tetrahedra by constructing a main diagonal and its projections on its faces, see Figure 2(a). The local bisection of the resulting tetrahedra is recursively carried out by using general ideas of the longest edge [17] and the newest vertex bisection methods. The refinement of a given triangulation is performed by a recursive algorithm. In order to guarantee that this procedure terminates in a finite number of iterations, the algorithm requires that the refinement edge of an element in the initial mesh is the same for all elements that share this edge. Details about the local refinement technique implemented in ALBERTA and restrictions on initial mesh are analyzed in [6, 14, 16]. This strategy works very efficiently for initial meshes obtained by subdivision of regular quadrilateral or hexahedral elements. In these cases, the degeneration of the resulting 2-D or 3-D triangulations after successive refinements is avoided. The restriction

on the initial element shapes and mesh connectivities makes it necessary to develop a particular mesh generator for ALBERTA. The presented algorithm produces compatible meshes with ALBERTA. Obviously, our mesh generation technique could be developed for other types of local refinement/derefinement algorithms for tetrahedral meshes [5, 7, 8].

The *meccano* technique presents several advantages with respect to more traditional approaches, such as Delaunay triangulation or the advancing front technique [1–4]. Delaunay triangulation requires a control in order to avoid *slivers*. Furthermore, the mesh conformity with the object boundary is not a trivial problem for complex geometry. On the other hand, advancing front technique requires a suitable surface triangulation. In addition, an appropriate definition of element sizes is demanded for obtaining good quality tetrahedra. Other technical difficulties appear when these two methods are applied to objects comprising different materials.

Our approach is based on the combination of several former procedures (refinement, derefinement, projection, untangling and smoothing) which are not in themselves new, but the overall integration is an original contribution. Authors have used them in different ways. Triangulations for convex domains can be constructed from a coarse mesh by using refinement/projection [15]. Adaptive nested meshes have been constructed with refinement and derefinement algorithms for evolution problems [18]. Large domain deformations can lead to severe mesh distortions, especially in 3-D. Mesh optimization is thus key for keeping mesh shape regularity and for avoiding a costly remeshing [19,20]. In traditional mesh optimization, mesh moving is guided by the minimization of certain overall functions, but it is usually done in a local fashion. In general, this procedure involves two steps [21, 22]: the first is for mesh untangling and the second one for mesh smoothing. Each step leads to a different objective function. In this paper, we use the improvement proposed by [13], where a simultaneous untangling and smoothing guided by the same objective function is introduced.

Some advantages of our technique are that: surface triangulation is automatically constructed, the final 3-D triangulation is conforming with the object boundary, inner surfaces are automatically preserved (for example, interface between several materials), node distribution is adapted in accordance with the object geometry, and parallel computations can easily be developed for meshing the *meccano* pieces. Nevertheless an admissible mapping between the *meccano* boundary and the object surface must be defined. Some effort should be made in that respect in the future.

In the following section we present a description of the main stages of the new mesh generation procedure. In Section 3 we show a test problem which illustrates the efficiency of this strategy. Finally, conclusions and future research are presented in Section 4.

# 2 Description of the Mesh Generator

In this section, we present the main ideas that have been introduced in the mesh generation procedure. The following algorithm describes the whole *mesh generation approach*

Mesh generation

1. Construct a *meccano* approximation formed by cuboids.
2. Define an admissible mapping between the *meccano* approximation and the object boundaries.
3. Construct a valid hexahedral mesh of the *meccano* approximation.
4. Construct a coarse tetrahedral mesh from the previous hexahedral mesh.
5. Generate a local refined tetrahedral mesh of the *meccano* for a given precision.
6. Move the boundary nodes of the *meccano* to the object surface according to the mapping defined in 2.
7. Relocate the inner nodes of the *meccano*.
8. Optimize the actual tetrahedral mesh applying the simultaneous untangling and smoothing procedure.

In Sections 2.1 and 2.2, we start with the definition of the domain and its subdivision in an initial 3-D triangulation that verifies the restrictions imposed by ALBERTA. In Section 2.3, we continue with the presentation of different strategies to obtain an adapted mesh which can approximate the boundaries of the domain within a given precision. We construct a mesh of the domain by projecting the boundary nodes from a *meccano* plane face to the true boundary surface and by relocating the inner nodes. These two steps are summarized in Sections 2.4 and 2.5, respectively. Finally, in Section 2.6 we present a procedure to optimize the resulting mesh.

## 2.1 Object *Meccano*

The first step of the procedure is to construct a *meccano* approximation. We have developed a simple CAD application that allows the user to generate a meccano approximation by connecting simple cuboids. This toolkit, called *MECCANO*, has the most common options of a graphic design application. The user can move, resize, rotate, delete, undelete, copy or paste cuboids, and edit the properties of each piece of the meccano approximation. Our application also verifies if the restrictions imposed on the topology and structure are correct each time the user makes some changes in the scene. MECCANO is based on library Coin3D [23]. Coin3D is a high level toolkit for developing 3D simulation and visualization applications. It is built on OpenGL and

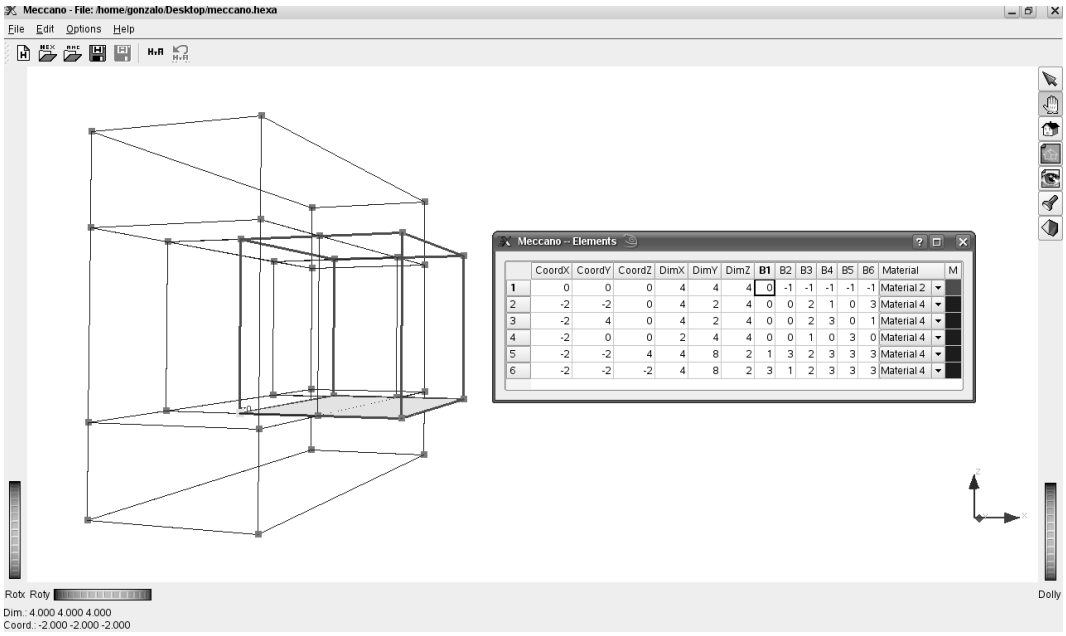



Figure 1: View of the main window of toolkit *MECCANO*. The *meccano* approximation in picture corresponds to the example of Section 3

uses scene graph data structures to render 3D graphics in real-time. In Figure 1 a view of the main window of *MECCANO* is shown.

Note that, in general the union of cuboids is non-valid hexahedral mesh. The general idea of the *meccano* technique could be understood as the connection of different polyhedral pieces. The use of cuboid pieces is an initial particular case.

Once the *meccano* approximation is fixed, we have to define an *admissible* mapping between the boundary faces of the *meccano* and the boundary of the object. We now introduce this concept. Let $\Sigma_0$ be the boundary of the *meccano* and $\Sigma$ the boundary of the object. We denote $\Sigma_0^i$ the $i$*-th* face of the *meccano* boundary, such that $\Sigma_0 = \bigcup_{i=1}^{n} \Sigma_0^i$ where $n$ is the number of *meccano* boundary faces. We define $\Pi : \Sigma_0 \to \Sigma$ as a piecewise function, such that $\Pi_{|\Sigma_0^i} = \Pi^i$ where $\Pi^i : \Sigma_0^i \to \Pi^i(\Sigma_0^i) \subset \Sigma$. Then, $\Pi$ is called an *admissible* mapping if it satisfies:

1. Functions $\{\Pi^i\}_{i=1}^n$ are compatible on $\Sigma_0$. That is $\Pi^i_{|\Sigma_0^i \cap \Sigma_0^j} = \Pi^j_{|\Sigma_0^j \cap \Sigma_0^i}$, $\forall i, j = 1, \ldots, n$, with $i \neq j$ and $\Sigma_0^i \cap \Sigma_0^j \neq \emptyset$.

2. Global mapping $\Pi$ is continuous and biyective between $\Sigma_0$ and $\Sigma$.

3. Functions $\Pi^i$ are differentiable on $\Sigma_0^i$.

We note that, if the mesh size of the *meccano* boundary is not small enough, the resulting surface mesh could be non-valid. The appropriate element size depends on gradient of $\Pi^i$. We also note that admissible mapping is not unique. Obviously, the quality of the resultant surface mesh depends on the chosen mapping.

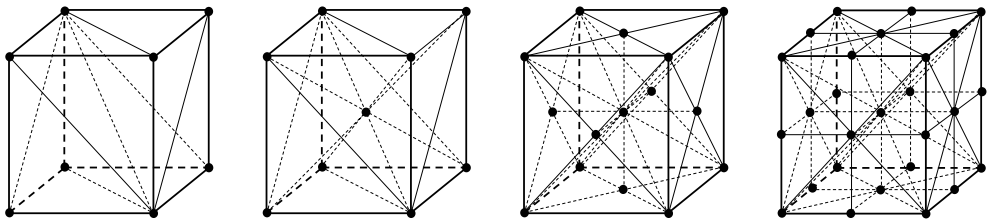


Figure 2: Refinement of a cube by using Kossaczky's algorithm: (a) cube subdivision into six tetrahedra, (b) bisection of all tetrahedra by inserting a new node in the cube main diagonal, (c) new nodes in diagonals of cube faces and (d) global refinement with new nodes in cube edges

## 2.2 Coarse Tetrahedral Mesh of the *Meccano*

The *meccano* is now decomposed into a coarse and valid hexahedral mesh by an appropriate subdivision of initial cuboids. Then, we build a coarse tetrahedral mesh by splitting each hexahedron into six tetrahedra [6]. For this purpose, it is necessary to define a main diagonal on each hexahedron and corresponding diagonal on its faces. For an example of the subdivision of a cube, see Figure 2(a). In order to get a conforming tetrahedral mesh, all hexahedra are subdivided in the same way, maintaining compatibility between the diagonal of their faces. The resulting initial mesh $\tau_1$ can be introduced in ALBERTA since it verifies the imposed restrictions about topology and structure. The user can introduce in the code the necessary number of recursive global bisections [6] for fixing a quasi-uniform element size in the whole initial mesh. Three consecutive global bisections for a cube are presented in Figures 2 (b), (c) and (d). The resulting mesh of Figure 2(d) contains $8$ cubes similar to the one shown in Figure 2(a). Obviously, the quality of the resulting tetrahedral mesh is directly related to the quality of the previous hexahedral mesh. Therefore, although the ideal case is the subdivision of the cuboids into cubes, it is not always possible.

## 2.3 Local Refined Mesh of the *Meccano*

The next step in the mesh generator includes a recursive adaptive local refinement strategy of those tetrahedra with a face placed on a boundary face of the initial coarse mesh. The refinement process is done in such a way that the true surfaces are approximated by a linear piecewise interpolation within a given precision. That is, we seek an adaptive triangulation on the *meccano* boundary faces, so that the resulting triangulation after node mapping on the object true boundary is a good approximation of this boundary. The user has to introduce as input data a parameter $\varepsilon$, which is a tolerance to measure the separation allowed between the linear piecewise interpolation and the true surface. At present, we have considered two criteria: the first related to the Euclidean distance between both surfaces and the second attending to the difference in terms of volume.

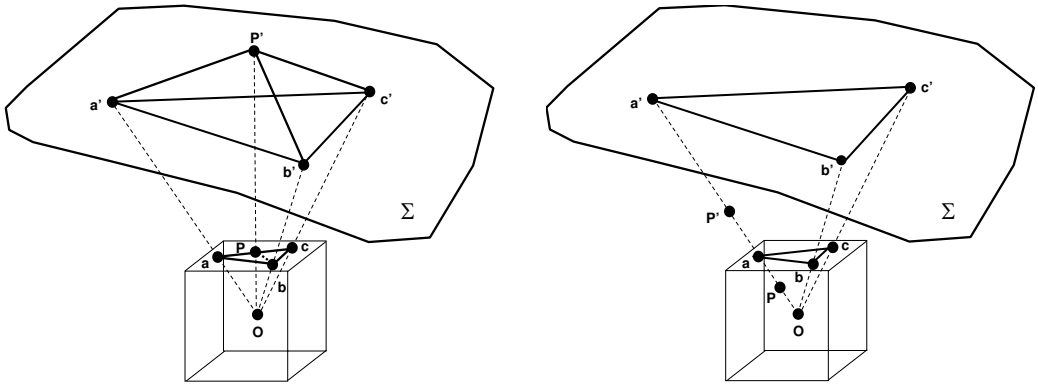



Figure 3: Node mapping from *meccano* to real domain: (a) transformation of an external node P and (b) of an inner node P

To illustrate these criteria, let $abc$ be a triangle placed on the *meccano* boundary, and $a'b'c'$ the resulting triangle after projecting the nodes $a$, $b$ and $c$ on surface $\Sigma$, see Figure 3. We define two different criteria to decide whether it is necessary to refine the triangle (and consequently the tetrahedra containing it) in order to improve the approximation.

For any point $Q$ in the triangle $abc$ we define $d_1^Q$ as the euclidean distance between the mapping of $Q$ on $\Sigma$, $Q'$, and the plane defined by $a'b'c'$. This definition is an estimate of the distance between the surface of the object and the current piecewise approximation.

We also introduce a measure in terms of volume, then, for any $Q$ in the triangle $abc$ we define $d_2^Q$ as the volume of the *virtual* tetrahedron $a'b'c'Q'$. In this case, $d_2^Q$ is an estimate of the lost volume in the linear approximation by the face $a'b'c'$ of the true surface.

The threshold of whether to refine the triangle or not is given by a tolerance $\varepsilon_i$ fixed by the user. With the previous definition, $d_1^Q < \varepsilon_1$ for all $Q$ in the boundary on the *meccano* implies that the distance between the surface of the object and its piecewise linear approximation is less than $\varepsilon_1$. On the other hand, $d_2^Q < \varepsilon_2$ for all $Q$ in the boundary on the *meccano* would mean that the lost volume per boundary face is bounded by $\varepsilon_2$. Alternatively, $\varepsilon_2$ could be defined as the allowed difference of volumes and we could use an equidistribution strategy as is usual in *a-posteriori* error estimates. Nevertheless, here we prefer to use a local version of $\varepsilon_2$, so the difference of volumes is estimated by multiplying $\varepsilon_2$ by the number of boundary faces of the final approximation.

Obviously, other measures could be introduced in line with the desired approximation type (curvature, points properties, *etc.*). What is more, the user could consider the combination of several measures simultaneously.

Once we have defined separation measures $d_i$ and tolerances $\varepsilon_i$, we propose two different strategies for reaching our objective in the following subsections.

### 2.3.1 Sequence of Refinements and Derefinement

The first strategy consists of a simple method. It combines a sequence of refinement steps with one derefinement step.

Simple refinement step. We construct a sequence of tetrahedral nested meshes by recursive bisection of all tetrahedra that contain a face located on the *meccano* boundary faces; see Figure 2. The number of bisections $n_b$ is determined by the user as a function of the desired resolution. At this point, we identify the true surface with the linear approximation obtained with this resolution. So, we have a uniform distribution of nodes on these *meccano* faces and we can consider their *virtual* mapping on the object boundary.

Derefinement step. We apply a derefinement procedure, which is a generalization of the strategy developed in [18]. This criterion fixes which tetrahedra introduced in the refinement sequence can be eliminated without damaging the approximation for the prescribed tolerance $\varepsilon_i$. The derefinement is applied iteratively to the current mesh and concludes either when there are no elements to remove or when the coarse mesh is reached.

Note that each tetrahedron $T$ (generated by bisection of its *father*) has a so-called *newest* node $P$. This node was introduced at the middle point of a prescribed edge $ac$ of the father of $T$ to generate its two sons, see Figure 3(a). The derefinement criterion is efficient because it only computes the distance $d_i^P$ relative to this *newest* node to decide if the tetrahedron $T$ could be removed. This distance is related to the face $abc$ (or its virtual mapping $a'b'c'$) of the father of $T$.

Then, the *derefinement criterion*, associated to a tetrahedron $T$ of the sequence of nested meshes, can be introduced as:

> Derefinement criterion. Tetrahedron $T$ is marked to be derefined, if it satisfies **one** of the following conditions:
>
> 1. The *newest* node $P$ of $T$ is interior.
> 2. The *newest* node $P$ of $T$ is placed on the boundary of the *meccano* and $d_i^P < \varepsilon_i$.

A marked tetrahedron $T$ will be removed only if all the elements generated by the bisection of the edge $ac$ of its father are also marked. So, the *refinement/derefinement procedure* to construct a local refined tetrahedral mesh of the *meccano* is summarized in the following algorithm:

> Refinement/derefinement procedure
>
> 1. Set $n_b$ and $\varepsilon_i$.
> 2. Construct the coarse tetrahedral mesh of the *meccano*.
> 3. Refine $n_b$ times all tetrahedra with at least one face placed on the *meccano* boundary.

4. Mark for derefinement all tetrahedra that satisfy the *derefinement criterion* for a distance $d_i$ and a tolerance $\varepsilon_i$.
5. Derefine the mesh.
6. If the mesh was modified, go to step 4.

### 2.3.2 Sequence of Local Refinements

The second strategy also starts with the coarse mesh of the *meccano*, but it only applies local refinement to obtain the fine one. In this case the *refinement criterion* for tetrahedron $T$ is:

Refinement criterion. Tetrahedron $T$ is marked to be refined, if it satisfies the following **two** conditions:

1. $T$ has a face $F$ on the boundary of the *meccano*.
2. $d_i^Q \geq \varepsilon_i$ for some point $Q$ located on face $F$ of $T$.

From a numerical point of view, the number of points $Q$ (analyzed in this strategy) is reduced to a set of points on a uniform mesh of a given resolution, or a set of points of quadrature. Finally, the *refinement strategy* for constructing a local refined tetrahedral mesh of the *meccano* is summarized in the following algorithm:

Refinement procedure

1. Set $n_b$ and $\varepsilon_i$.
2. Construct the coarse tetrahedral mesh of the *meccano*.
3. Mark for refinement all tetrahedra which satisfy the *refinement criterion* for a distance $d_i$ and a tolerance $\varepsilon_i$.
4. Refine the mesh.
5. If the number of refinement steps is less than $n_b$ and the mesh was modified, go to step 3.

While the first strategy is simpler, it could lead to problems with memory requirements if the number of tetrahedra is very high before applying the derefinement algorithm. For example, this situation can occur when there are surfaces defined by very high resolution functions. Nevertheless, the user could control the number of recursive bisections $n_b$ and the tolerance $\varepsilon_i$.

On the other hand, the problem of the second strategy is to determine whether a face placed on *meccano* boundary must be subdivided to achieve the desired approximation of the true surface. This analysis must be done every time that a boundary face is subdivided into its two *son* faces. Suppose, for example, that the true surface is given by a discrete function. Then, the subdivision criterion should stop for

a particular face when all the surface discretization points, defined on this face, have been analyzed and all of them verify the approximation criterion. So, this second strategy has the inconvenience that each surface discretization point could be studied many times and, therefore, it generally involves a higher computational cost than the first strategy. Nevertheless, both of those strategies could be faster depending on the geometry of the object surface and the parameters fixed by the user.

## 2.4 External Node Mapping on Object Boundary

Although ALBERTA has already implemented a node projection on a given boundary surface during the bisection process, it has two important restrictions: nodes belonging to the initial mesh are not projected and inverted elements could appear in the case of projecting new nodes on complex surfaces (*i.e.* non-convex object). In the latter case, the code does not work properly since it is only prepared to manage *valid* meshes.

Therefore, a new strategy must be developed in the mesh generator. The projection (or mapping) is really done once we have defined the local refined mesh by using one of the two methods proposed in the previous section. Then, the nodes placed on the *meccano* faces are projected (or mapped) on their corresponding true surfaces, maintaining the position of the inner nodes of the *meccano* triangulation.

After this process, we obtain a valid triangulation of the domain boundary, but a tangled tetrahedral mesh could appear. Inner nodes of the *meccano* could now be located even outside the domain. Thus, an optimization of the mesh is necessary. Although the final optimized mesh does not depend on the initial position of the inner nodes, it is better for the optimization algorithm to start from a mesh with as good a quality as possible. Therefore, we propose to relocate the inner nodes of the *meccano* in a reasonable position before the mesh optimization.

## 2.5 Relocation of Inner Nodes

There would be several strategies for defining an appropriate position for each inner node of the domain. An acceptable procedure is to modify their relative position as a function of the distance between boundary surfaces before and after their projections. This relocation is done relative to proportional criteria along the corresponding projection line. For example, relocation of inner node $P$ in its new position $P'$, such that $OP' = OP \times Oa' \,/\, Oa$, is represented in Figure 3(b).

Although this node movement does not solve the tangle mesh problem, it normally lessens it. In other words, the resulting number of inverted elements is lower and the mean quality of valid elements is greater.

## 2.6 Object Mesh Optimization: Untangling and Smoothing

An efficient procedure is necessary to optimize the current mesh. This process must be able to smooth and untangle the mesh and is crucial in the proposed mesh generator.

The most usual techniques to improve the quality of a *valid* mesh, that is, a mesh with no inverted elements, are based upon local smoothing. In short, these techniques consist of finding the new positions that the mesh nodes must hold, in such a way that they optimize an objective function. Such a function is based on a certain measurement of the quality of the *local submesh,* $N(v)$, formed by the set of elements connected to the *free node* $v$ whose coordinates are given by $\mathbf{x}$. We have considered the following objective function (1) derived from an *algebraic mesh quality metric* studied in [20], but it would also be possible to use other objective functions that have barriers like those presented in [19]:

$$K(\mathbf{x}) = \left[ \sum_{m=1}^{M} \left( \frac{1}{q_{\eta_m}} \right)^p (\mathbf{x}) \right]^{\frac{1}{p}}, \tag{1}$$

where $M$ is the number of elements in $N(v)$, $q_{\eta_m}$ is an algebraic quality measure of the $m$-th element of $N(v)$ and p is usually chosen as 1 or 2. Specifically, we have considered the mean ratio quality measure, which for a tetrahedron is $q_\eta = \frac{3\sigma^{\frac{2}{3}}}{|S|^2}$ and for a triangle is $q_\eta = \frac{2\sigma}{|S|^2}$, being $|S|$ the Frobenius norm of matrix $S$ associated with the affine map from the *ideal* element (usually equilateral tetrahedron or triangle) to the physical one, and $\sigma = \det(S)$. Other algebraic quality measures can be used as, for example, the metrics based on the condition number of matrix $S$, $q_\kappa = \frac{\rho}{|S||S^{-1}|}$, where $\rho = 2$ for triangles and $\rho = 3$ for tetrahedra.

As it is a local optimization process, we cannot guarantee that the final mesh is globally optimum. Nevertheless, after repeating this process several times for all the nodes of the current mesh, quite satisfactory results can be achieved. Objective functions are usually appropriate to improve the quality of a valid mesh, but they do not work properly when there are inverted elements. This is because they present singularities (barriers) when any tetrahedron of $N(v)$ changes the sign of its Jacobian determinant. To avoid this problem it is possible to proceed as Freitag *et al.* in [21, 22], where an optimization method consisting of two stages is proposed. In the first, the possible inverted elements are untangled by an algorithm that maximizes their negative Jacobian determinants [21] while, in the second, the resulting mesh from the first stage is smoothed using another objective function based on a quality metric of the tetrahedra of $N(v)$ [22]. After the untangling procedure, the mesh has a very poor quality because the technique has no motivation to create good-quality elements. As remarked in [22], it is not possible to apply a gradient-based algorithm to optimize the objective function because it is not continuous all over $\mathbb{R}^3$, making it necessary to use other non-standard approaches.

We have proposed an alternative to this procedure [13], so the untangling and smoothing are carried out at the same stage. For this purpose, we use a suitable mod-

ification of the objective function such that it is regular all over $\mathbb{R}^3$. It consists of substituting the term $\sigma$ in the quality metrics with the positive and increasing function $h(\sigma) = \frac{1}{2}(\sigma + \sqrt{\sigma^2 + 4\delta^2})$. When a feasible region (subset of $\mathbb{R}^3$ where $v$ could be placed, $N(v)$ being a valid submesh) exists, the minima of the original and modified objective functions are very close and, when this region does not exist, the minimum of the modified objective function is located in such a way that it tends to untangle $N(v)$. The latter occurs, for example, when the fixed boundary of $N(v)$ is tangled. With this approach, we can use any standard and efficient unconstrained optimization method [24] to find the minimum of the modified objective function.

In addition, a smoothing of the boundary surface triangulation could be applied before the movement of inner nodes of the domain by using the new procedure presented in [25] and [26]. This surface triangulation smoothing technique is also based on a vertex repositioning defined by the minimization of a suitable objective function. The original problem on the surface is transformed into a two-dimensional one on the *parametric space*. In our case, the parametric space is a plane, chosen in terms of the local mesh, in such a way that this mesh can be optimally projected performing a *valid* mesh, that is, without *inverted* elements.

# 3 Test Example

The performance of our new mesh generator is shown in the following application. We use our strategy to obtain a 3D mesh of the $25^{\text{th}}$ Civil-Comp Conference logo. A few more examples can be found at [10–12], where we analyze different issues of our algorithm: refinement strategies, derefinement parameter, surface deviation of the resulting mesh, applications to complex geometries, etc.

We are going to stamp the $25^{\text{th}}$ Civil-Comp Conferences logo on a *button*. This object is the union of a sphere with ratio 2, and semi-sphere with ratio 4, both with the same center point. We stamp the number "25" on the uncovered surface of the sphere, and the legend "- Civil-Comp Conferences - 1983 - 2008" on the uncovered flat place of the semi-sphere.

The input data can be easily generated with toolkit MECCANO. It consists of five cuboids for the semi-sphere and one cuboid for the small sphere, see Figure 1 or Figure 4(a). This *meccano* approximation is included in a parallelepiped whose dimensions are $8 \times 6 \times 8$. A one-to-one projection between *meccano* and the object is defined by a sphere projection.

The *meccano* is first split into 288 cubes, see Figure 4(b), and then into 3-D triangulation of 1728 tetrahedra and 455 nodes, see Figure 4(c). We apply $9$ recursive bisections on all tetrahedra which have a face placed on the *meccano* boundary or on the sphere interface, and an additional $9$ recursive bisections on the area where the legend will be stamped. This mesh contains 469672 nodes and 2161960 tetrahedra. The derefinement criterion is applied to capture the sphere curvature and the legend, according to Section 2.3. We use here $\varepsilon_2 = 0.0001$ for the curvature, and decide

to keep in the surface triangulation all triangles in the interface of the legend. The resulting mesh contains 187750 nodes and 43710, see Figure 4(d).

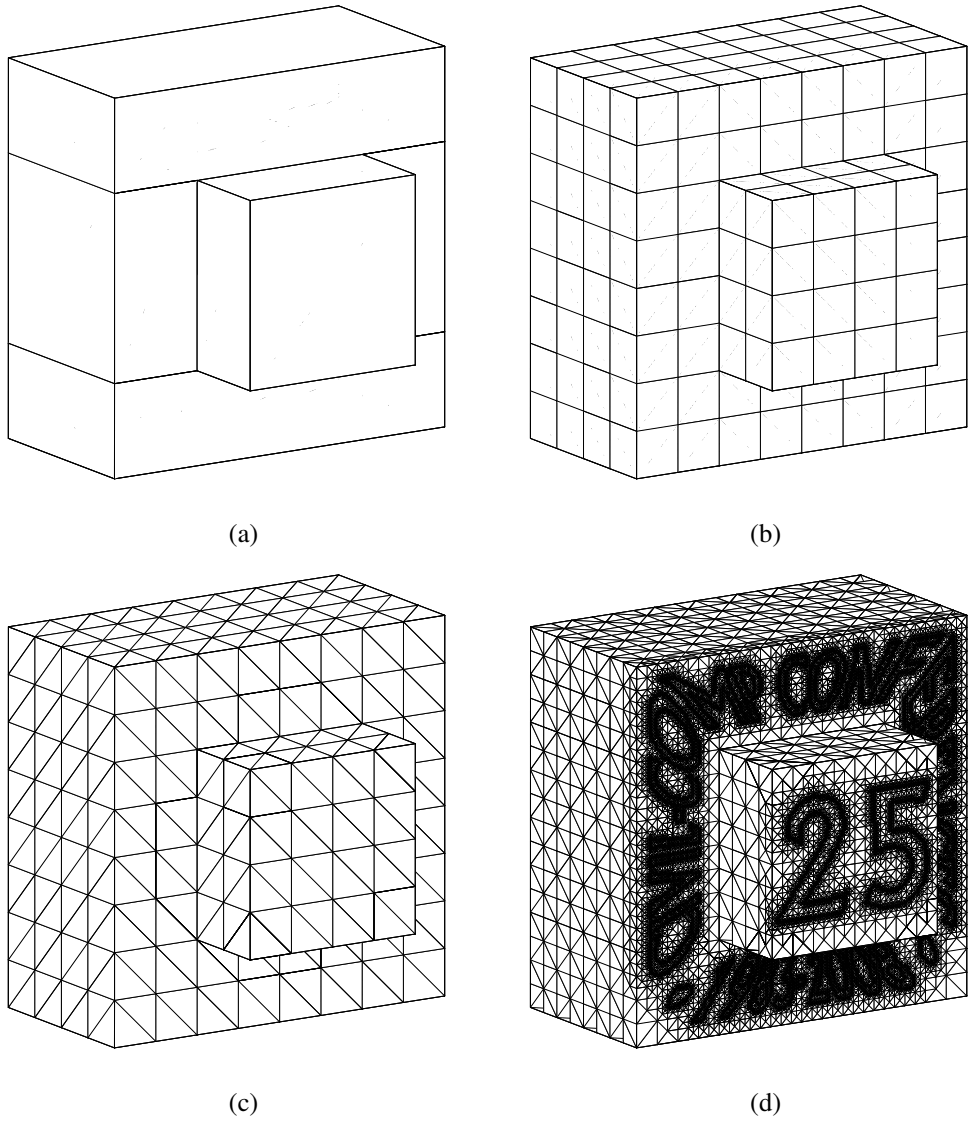
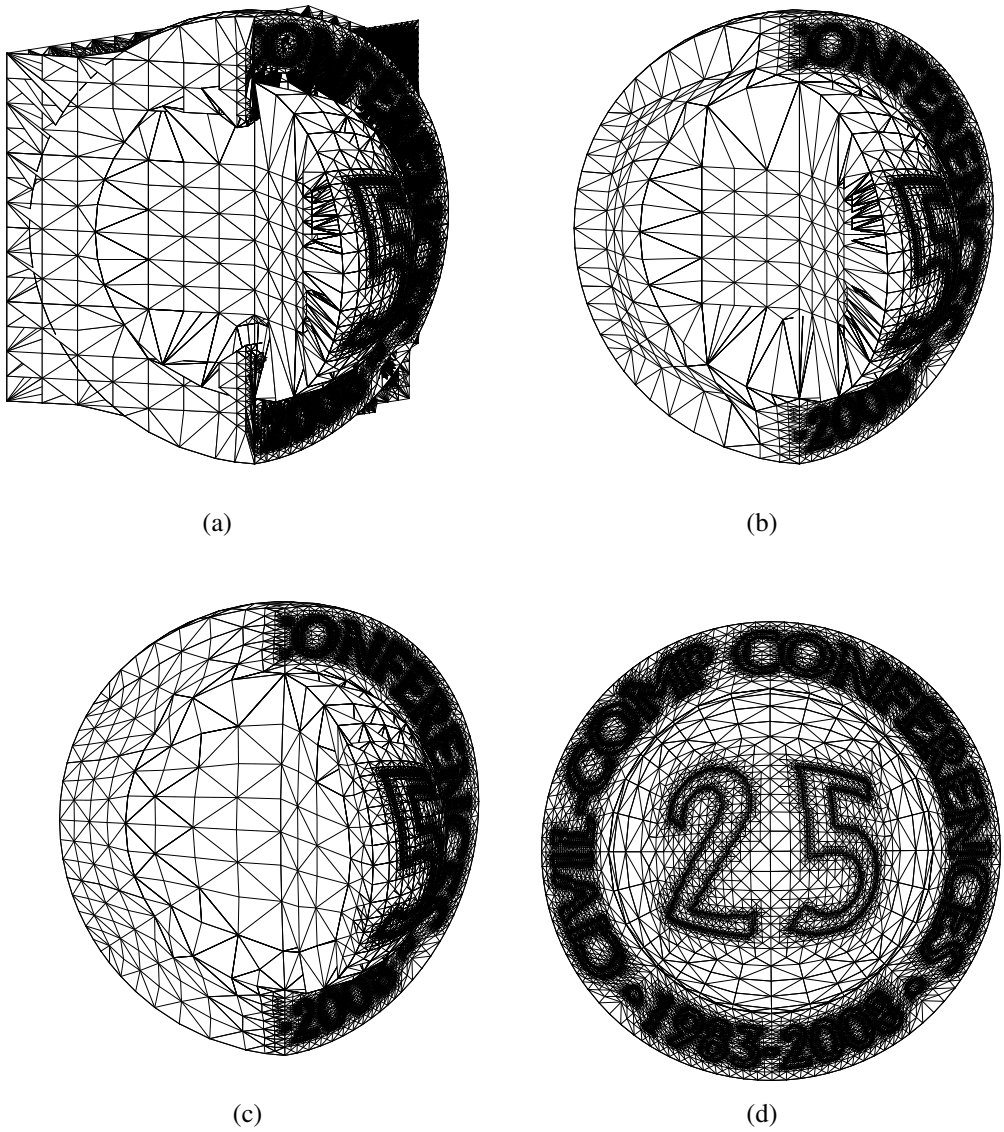



Figure 4: Main stages of the mesh generator: (a) *meccano*, (b) valid mesh of cubes, (c) coarse tetrahedral mesh, (d) mesh adaption after applying the refinement/derefinement procedure

The projection of this *meccano* surface triangulation on the true surface produces a 3-D tangled mesh with 19682 inverted elements, see Figure 5(a). The relocation of inner nodes by using a proportional criterion reduces the number of inverted tetrahedra to 88, Figure 5(b). After 9 iterations, the mesh optimization algorithm of Section 2.6 converts the tangled mesh into the one presented in Figure 5(c).



(a) (b)

(c) (d)

Figure 5: Main stages of the mesh generator (continued): (a) cross section of tangled mesh after the projection, (b) cross section of tangled mesh after relocation, (c) cross section of resulting mesh after mesh optimization process. (d) fontral view of the final mesh

The mesh quality is improved to a minimum value of $0.32$ and an average $\overline{q}_\kappa = 0.69$. The quality curves for the initial, intermediate and final triangulations are shown in Figure 6.

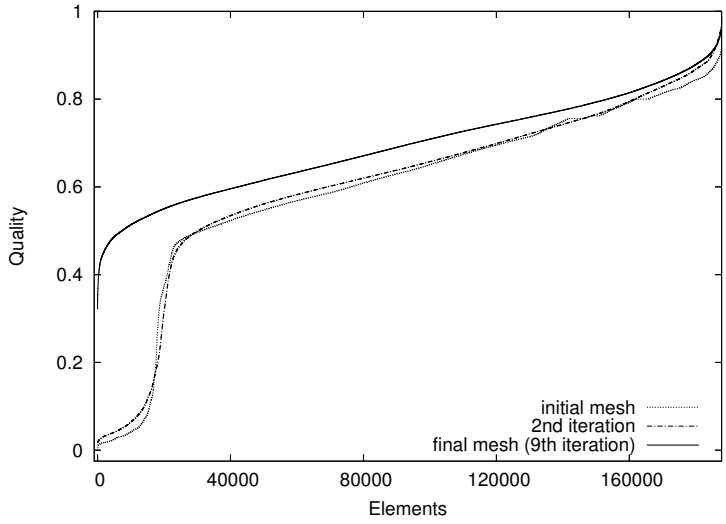



Figure 6: Quality curves, using $q_\kappa = \frac{3}{|S||S^{-1}|}$, for the initial and optimized meshes after two and nine (final mesh) iterations

The CPU time for constructing the initial mesh (refinement/derefinement, projection and inner relocation) is approximately $37$ seconds and for its optimization (untangling and smoothing) is $120$ seconds on a Intel Xeon processor, $3.06\ GHz$ and $4\ Gb$ RAM memory.

# 4 Conclusions and Future Research

The proposed mesh generator is an efficient method for creating tetrahedral meshes on domains with boundary faces projectable on a *meccano* boundary. We remark that it requires minimum user intervention and has a low computational cost.

Although this procedure is at present limited in applicability for highly complex geometries, it results in a very efficient approach to the problems that fall within the mentioned class. At present, the user has to define the *meccano* associated to the object and mapping between *meccano* and object surfaces. Once these aspects are fixed, the mesh generation procedure is fully automatic. A simple CAD application has been developed to design the *meccano* approximation. In future works, we will increase the functionality of our CAD application, including new types of pieces in the construction of the *meccano*, and adding an automatic mapping between the boundary faces of the *meccano* and the boundary of the object. Specifically, object surface patches should be defined using *meccano* surfaces as parametric spaces.

The mesh generation technique is based on sub-processes (subdivision, projection, optimization) which are not in themselves new, but the overall integration using a simple shape as a starting point is an original contribution of this paper and it has some obvious performance advantages. We have also introduced a generalized derefinement condition for a simple approximation of surfaces. Finally, another interesting property of the new mesh generation strategy is that it automatically fixes the boundary between materials and achieves a good mesh adaption to the geometrical characteristics of the domain.

# Acknowledgments

This work has been supported by the Spanish Government, "Secretaría de Estado de Universidades e Investigación", "Ministerio de Educación y Ciencia", and FEDER, grant contracts: CGL2004-06171-C03-02-03 and CGL2007-65680-C03-01-03.

# References

[1] G.F. Carey, "Computational Grids: Generation, Adaptation, and Solution Strategies", Taylor & Francis, Washington, 1997.

[2] P.J. Frey, P.L. George, "Mesh Generation", Hermes Science Publishing, Oxford, 2000.

[3] P.L. George, H. Borouchaki, "Delaunay Triangulation and Meshing: Application to Finite Elements", Editions Hermes, Paris, 1998.

[4] J.F. Thompson, B. Soni, N. Weatherill, "Handbook of Grid Generation", CRC Press, London, 1999.

[5] J.M. González-Yuste, R. Montenegro, J.M. Escobar, G. Montero, E. Rodríguez, "Local Refinement of 3-D Triangulations Using Object-oriented Methods", Adv. Eng. Soft., 35, 693-702, 2004.

[6] I. Kossaczky, "A Recursive Approach to Local Mesh Refinement in Two and Three Dimensions", J. Comput. Appl. Math., 55, 275-288, 1994.

[7] R. Löhner, J.D. Baum, "Adaptive H-Refinement on 3-D Unstructured Grids for Transient Problems", Int. J. Num. Meth. Fluids, 14, 1407-1419, 1992.

[8] M.C. Rivara, C. Levin, "A 3-D Refinement Algorithm Suitable for Adaptive Multigrid Techniques", J. Comm. Appl. Numer. Meth., 8, 281-290, 1992.

[9] G.F. Carey, "A Perspective on Adaptive Modeling and Meshing (AM&M)", Comput. Meth. Appl. Mech. Eng., 195, 214-235, 2006.

[10] R. Montenegro, J.M. Cascón, J.M. Escobar, E. Rodríguez, G. Montero, "Implementation in ALBERTA of an automatic tetrahedral mesh generator", in: "Proceeding 15th International Meshing Roundtable", Springer, Berlin, 325-338, 2006.

[11] J.M. Cascón, R. Montenegro, J.M. Escobar, E. Rodríguez, G. Montero, "A new *meccano* technique for adaptive 3-D triangulation" in: "Proceedings of 16th International Meshing Roundtable", Springer, Berlin, 103-120, 2007.

[12] R. Montenegro, J.M. Cascón, J.M. Escobar, E. Rodríguez, G. Montero, "An automatic strategy for adaptive tetrahedral mesh generation", Applied Numerical Mathematics (to appear).

[13] J.M. Escobar, E. Rodríguez, R. Montenegro, G. Montero, J.M. González-Yuste, "Simultaneous Untangling and Smoothing of Tetrahedral Meshes", Comput. Meth. Appl. Mech. Eng., 192, 2775-2787, 2003.

[14] "ALBERTA - An Adaptive Hierarchical Finite Element Toolbox", http://www.alberta-fem.de/.

[15] A. Schmidt, K.G. Siebert, Design of Adaptive Finite Element Software: The Finite Element Toolbox ALBERTA, "Lecture Notes in Computer Science and Engineering", Vol. 42, Springer, Berlin, 2005.

[16] W.F. Mitchell, "A Comparison of Adaptive Refinement Techniques for Elliptic Problems", ACM Trans. Math. Soft., 15, 326-347, 1989.

[17] M.C. Rivara, "A Grid Generator Based on 4-Triangles Conforming. Mesh-refinement Algorithms", Int. J. Num. Meth. Eng., 24, 1343-1354, 1987.

[18] L. Ferragut, R. Montenegro, A. Plaza, "Efficient Refinement/Derefinement Algorithm of Nested Meshes to Solve Evolution Problems", Comm. Num. Meth. Eng., 10, 403-412, 1994.

[19] P.M. Knupp, "Achieving Finite Element Mesh Quality Via Optimization of the Jacobian Matrix Norm and Associated Quantities. Part II-A Frame Work for Volume Mesh Optimization and the Condition Number of the Jacobian Matrix", Int. J. Num. Meth. Eng., 48, 1165-1185, 2000.

[20] P.M. Knupp, "Algebraic Mesh Quality Metrics", SIAM J. Sci. Comput., 23, 193-218, 2001.

[21] L.A. Freitag, P. Plassmann, "Local Optimization-based Simplicial Mesh Untangling and Improvement", Int. J. Num. Meth. Eng. 49, 109-125, 2000.

[22] L.A. Freitag, P.M. Knupp, "Tetrahedral Mesh Improvement Via Optimization of the Element Condition Number", Int. J. Num. Meth. Eng., 53, 1377-1391, 2002.

[23] "Coin 3D-3D Graphics Development Tools", http://www.coin3d.org/.

[24] M.S. Bazaraa, H.D. Sherali, C.M. Shetty, "Nonlinear Programing: Theory and Algorithms", John Wiley and Sons Inc., New York, 1993.

[25] J.M. Escobar, G. Montero, R. Montenegro, E. Rodríguez, "An Algebraic Method for Smoothing Surface Triangulations on a Local Parametric Space", Int. J. Num. Meth. Eng., 66, 740-760, 2006.

[26] R. Montenegro, J.M. Escobar, G. Montero, E. Rodríguez, "Quality improvement of surface triangulations", in: "Proceeding 14th International Meshing Roundtable", Springer, Berlin, 469-484, 2005.