

On the Use of Finite Elements in gOcad

Luiz Fernando Martha, João Luiz Campos, and Joaquim Cavalcante Neto
Tecgraf/PUC-Rio – Brazil

Abstract

This work describes the implementation of a plug-in that adds finite element pre- and post-processing capabilities to gOcad. The gFEM plug-in implements an algorithm for generating unstructured tetrahedral finite element meshes in arbitrarily shaped three-dimensional regions. This meshing algorithm incorporates aspects of well-known meshing procedures, but includes some original steps. It uses an advancing front technique that generates good shape quality elements, along with an octree to develop local guidelines for the size of generated elements. The advancing front technique is based on a standard procedure found in the literature with two additional steps to ensure valid volume mesh generation for virtually any domain. The first additional step is related to the generation of elements only considering the topology of the current front and the second additional step is a backtracking procedure with face deletion, to ensure that a mesh can be generated even when problems happen during the advance of the front. To improve mesh quality (as far as element shape is concerned), an *a posteriori* local mesh improvement procedure is used. A description of the new object classes introduced in gOcad data structure for the implementation of the finite element meshing and result visualization capabilities is presented.

1 Introduction

It is well known that gOcad can be used in many applications. This may be attributed to its modeling and visualization capabilities. One natural application of gOcad is pre- and post-processing of three-dimensional finite element simulation. This work describes the implementation of a gOcad plug-in, called gFEM, that adds capabilities to gOcad to act as a finite element mesh generator and as a finite element post-processor.

Finite element simulation imposes severe shape restrictions to triangular and tetrahedral objects. Shape quality is necessary to avoid numerical problems in a simulation. This implies that element shape quality measures are necessary. This topic is well represented in the literature [Joe 1991, Parthasarathy et al 1993, Liu & Joe 1994, Lewis et al 1996], and many unstructured mesh generation algorithms incorporate features to avoid generation of poor shape elements [Moller & Hansbo 1995, Chan & Anastasiou 1996, Rassineux 1998, Cavalcante Neto et al 2001].

The proposed gFEM plug-in uses a meshing algorithm devised by Cavalcante Neto and co-authors that has been recently published [2001]. This algorithm was designed to meet specific shape requirements. First, the algorithm should produce well-shaped elements, avoiding elements with poor aspect ratios. While the algorithm does not guarantee bounds on element aspect ratios, care is taken at each step to generate the best-shaped elements possible. Empirical observations, described in [Cavalcante Neto et al 2001], show that the algorithm is largely successful in meeting this requirement. The second requirement is that the algorithm generates a mesh that conforms to an existing triangular mesh on the boundary of a solid region. This is important in gOcad context because usually it is necessary to honor existing surface triangulations. Finally, a third requirement of the algorithm is that it has the ability to transition well between regions with elements of highly varying size. Finite element simulations require high degree of element density in regions of high response gradients. Therefore, it is not uncommon to have in a same region different orders of magnitude of element characteristic sizes. Some other algorithms work best when all generated elements have similar characteristic size, but the current algorithm has been designed to have good size transition capabilities.

The body of this paper is divided into six sections. Sections 2 and 3 summarize the new gOcad objects implemented in gFEM plug-in. MSolid class, which is similar to existing TSolid class, is used to represent a set of finite element mesh objects. Each solid region has a distinct mesh, which is represented by an MMesh object. A new class, called Mesh3dTessellator, encapsulates the mesh generation algorithm. Section 3 also describes the communication of gOcad with a finite element program, which is done through a neutral format file. In addition, this section gives some insight on the representation of finite element simulation attributes and results. Section 4, which is the major part of this paper, describes the steps of the mesh generation algorithm in some detail. Section 5 shows an application example of the new pre- and post-processing capabilities of gFEM plug-in. Finally, Section 6 gives some concluding remarks.

2 The gFEM plug-in

gFEM is a gOcad plug-in that provides new capabilities to the program, which may be used as a pre- and post-processor for finite element analysis. The plug-in is composed by a set of libraries with specific functions:

- mesh:** Implements a new gOcad object that provides a 3D tessellation algorithm to generate unstructured tetrahedral finite element meshes, and an interface to link gOcad with finite element programs.
- gapi:** Implements an API interface for the new object.
- gui:** Modifies the standard gOcad interface to allow the communication with finite element programs through a neutral file.

3 The MSolid object

A new gOcad object was implemented to support finite element mesh generation using gOcad. The MSolid object was designed to have similar properties than current TSolid object, which is used to represent a solid inside gOcad. The main differences are that MSolid implements a new mesh generation algorithm [Cavalcante Neto et al 2001] and holds a set of MMesh objects. Each MMesh object represents a finite element mesh inside a single solid region. The MSolid class derives from the AtomicGroup class, as can be seen in Figure 1. Alternatively, MSolid class could have been derived from TSolid class.

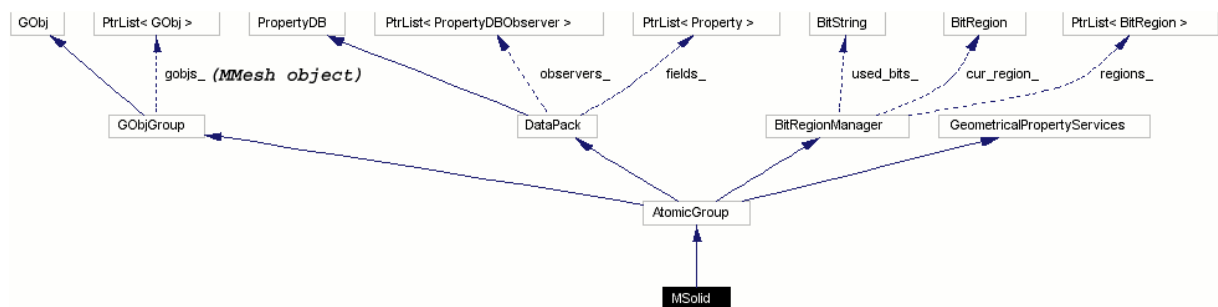


Figure 1 – MSolid class inheritance diagram.

The MMesh class derives from the Atomic class, as shown in Figure 2, and holds a tetrahedral mesh generated by the 3D tessellation algorithm. The mesh generation algorithm is implemented in C and has its own data structure. Therefore, a new object, called Mesh3dTessellator, was created to encapsulate the C code. As described in Section 4, this algorithm generates a finite element mesh for a single solid region.

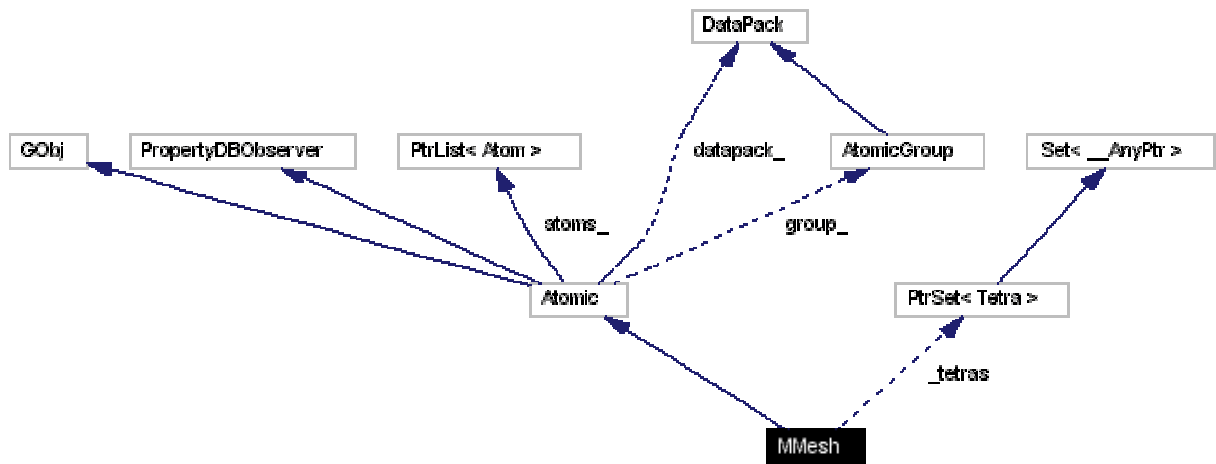


Figure 2 – MMesh class inheritance diagram.

When a Mesh3dTessellator object is created, its constructor method builds up the input data needed by the mesh generation algorithm. Basically, this consists of a list of triangles on the boundary of the target solid region. As for the creation of a TSolid object, an MSolid object requires a set of closed TSurf objects for its creation. Each closed TSurf object provides the list of triangles for the meshing algorithm. A method of the TSurf object checks whether it forms a closed region, which is required by this algorithm. From this input data, a Mesh3dTessellator object is created for each closed region. Each Mesh3dTessellator object creates an MMesh object to hold the generated mesh, which in turn is attached to the list of MMesh objects of the corresponding MSolid object. The generated mesh consists of a list of node coordinates and a list of tetrahedra defined by their node connectivity. Finally, a conversion method is called to incorporate the generated mesh into gOcad data structure. Each generated tetrahedron is a gOcad Tetra object.

To make the new object accessible from the gOcad program interface, an implementation of CLI commands was made for the MSolid object. In addition, all objects needed for visualization and input/output functions were implemented.

3.1 Neutral File communication

The communication between the gOcad application and a finite element program is made through an ASCII file in a neutral format, which is called Neutral File. This file format is described in site <http://suporte.tecgraf.puc-rio.br/manuais/neutralfile> and is used to represent both input data (mesh and simulation attributes) and output data (results) of a finite element simulation.

Two new functions were implemented to provide interface between gOcad and the finite element program: the Neutral File export function and the Neutral File import function (Figure 3).

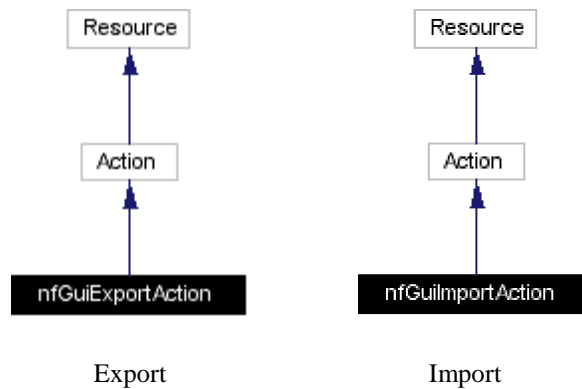


Figure 3 – Interface functions to export and import Neutral Files.

These two new interface functions were inserted in the File sub-menu on the gOcad's main window. When an option is chosen, the corresponding action is executed (Figure 4).

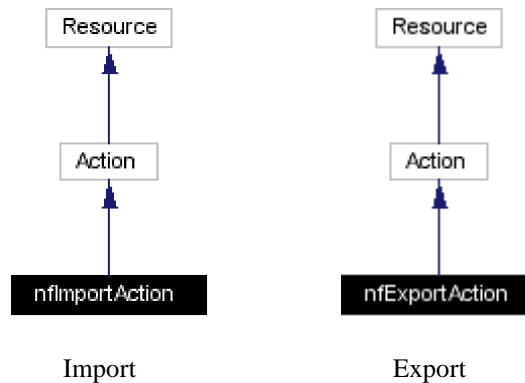


Figure 4 – Action functions to export and import Neutral Files.

For either export or import actions, the Action object must deal with finite element simulation attributes. These attributes must be translated to or from the Neutral File format. The translation of nodal attributes is straightforward, since in gOcad attributes are attached to vertices. This translation can be done by creating properties, in the gOcad model, that have direct correspondence with nodal attributes in the finite element model, such as:

- supp_x** – nodal displacement is restricted in the global X direction.
- supp_y** – nodal displacement is restricted in the global Y direction.
- supp_z** – nodal displacement is restricted in the global Z direction.
- load_x** – nodal force in the global X direction.
- load_y** – nodal force in the global Y direction.
- load_z** – nodal force in the global Z direction.

These properties can be combined to result in finite element boundary conditions for a structural stress analysis.

In order to set finite element material properties or any other property attached to elements, since the gOcad has only vertex properties, an extension of gOcad Tetra class was made having a new field to hold these properties (currently only material property was implemented). Material properties are saved both in gOcad object ASCII file format and in Neutral File format.

The results of a finite element analysis may be defined at nodes or elements. Nodal results are directly associated with vertex properties in the gOcad model. Element results are usually specified at integration points. In this case, an extrapolation/smoothing process is performed to get the results at node points, and all finite element results are imported as vertex properties in gOcad. Therefore, gOcad visualization tools may be used to show these results, and the program naturally works as a finite element post-processor.

4 Mesh generation algorithm

The current algorithm incorporates aspects of well-known meshing procedures and includes some original steps. It uses an advancing front technique (AFT), along with an octree to develop local guidelines for the size of generated elements. The AFT is based on a standard procedure found in the literature [Peraire et al 1988, Lohner & Parikh 1988, Jin & Tanner 1993, Moller & Hansbo 1995, Chan & Anastasiou 1996, Rassineux 1998] with additional steps to ensure valid volume mesh generation for virtually any domain. Special care is taken during the advancing front procedure to generate elements with the best shape possible.

The input to the present mesh generation algorithm is a faceted description of the boundary of a region to be meshed. This is given by a list of nodes defined by their coordinates, and a list of triangular faces defined by their node connectivity. This type of input can represent geometries of any shape, including holes, and it can be easily incorporated in any modeler or finite element system. The algorithm is organized in the phases listed in Figure 5.

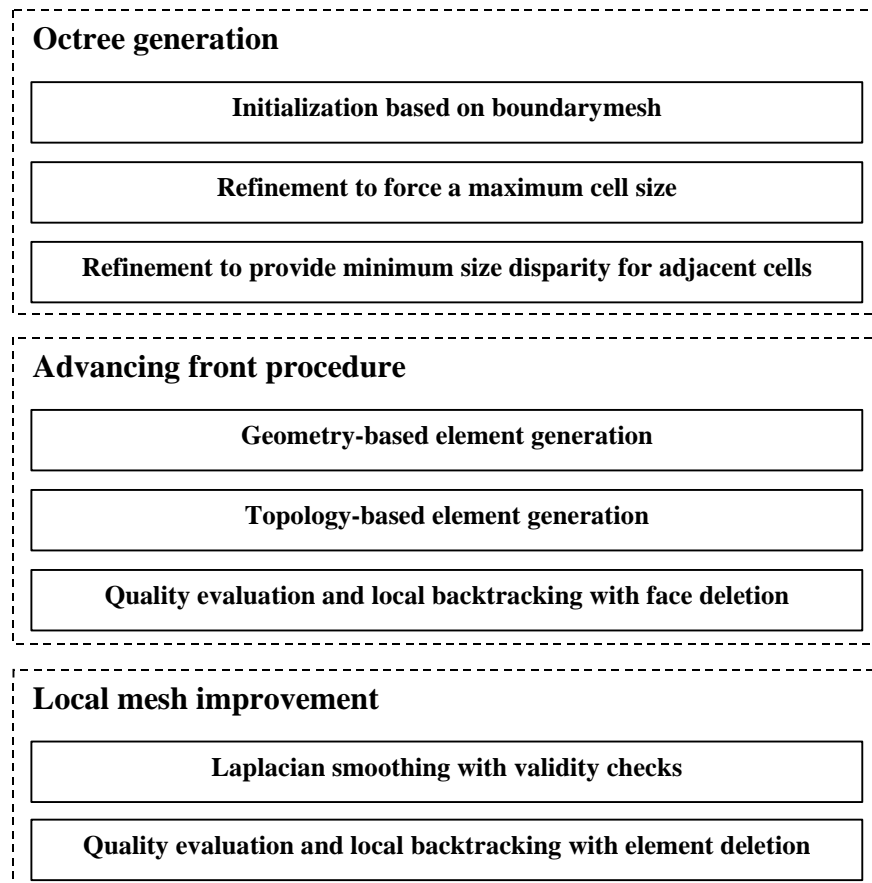


Figure 5 – Outline of mesh generation algorithm.

The primary purpose for the octree is to generate guidelines for the size of the elements generated during the advancing front procedure. The element size distribution through the region is inferred by the size distribution in the input boundary mesh. The octree generation involves three steps. In the first step, the octree is initialized based on the input data. In the other two steps, the octree is further refined. Figure 6 is used to illustrate the process of generating the octree, which, for clarity, depicts a two-dimensional example using a quadtree.

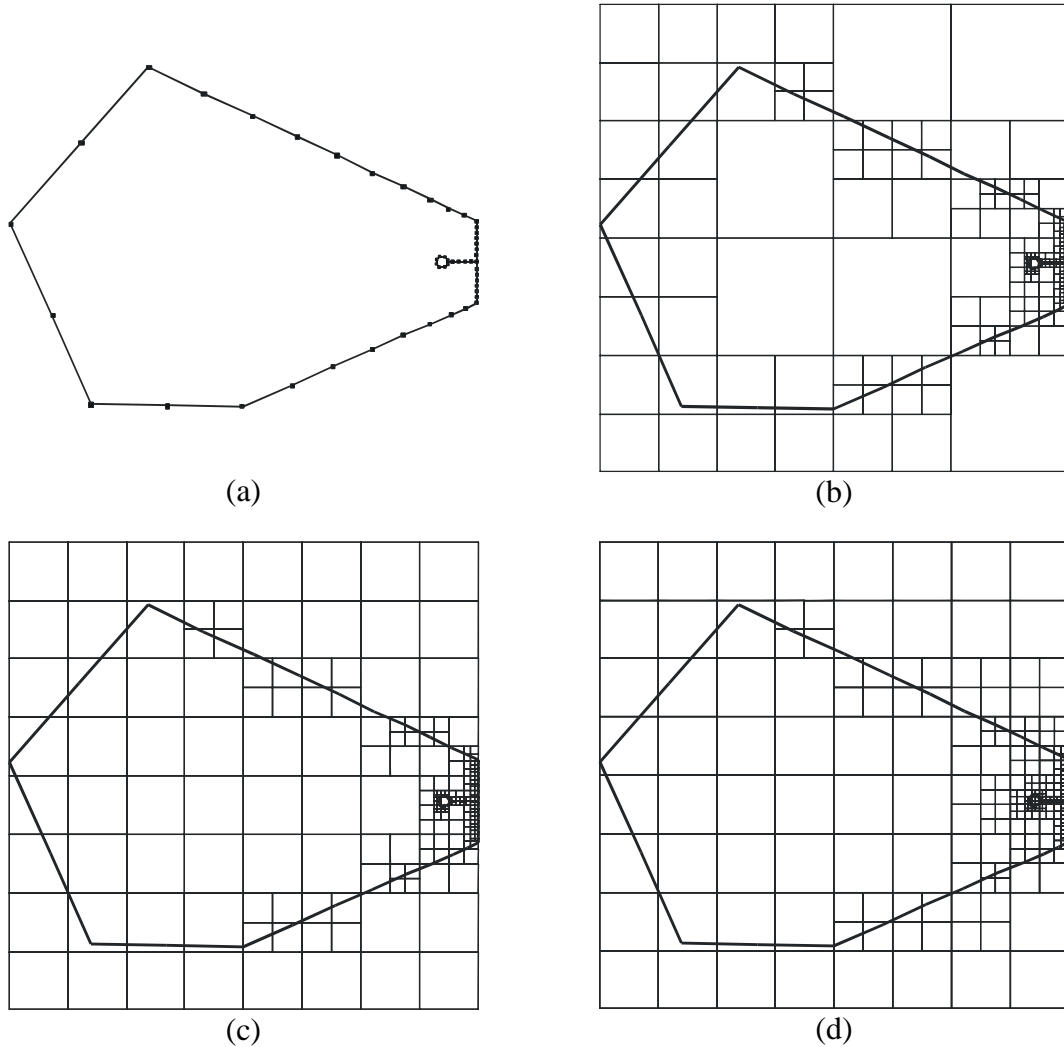


Figure 6 – Generation of background octree structure from given boundary mesh (2D example).

Initially, a bounding cube is created based on the maximum range of any of the three Cartesian coordinates of the nodes in the input data. This is the root cell of the octree. Figure 6-a illustrates a hypothetical two-dimensional input data represented by the model and its boundary refinement. The boundary model presents an increasing degree of refinement from the left side to the right side. In the first step of the octree generation, represented by the initialization of the octree, each face of the input boundary mesh is used to determine the local depth of subdivision. The octree cell containing the centroid of each input face is determined. If the area of this cell's face is larger than the area of the boundary face, then this cell is subdivided into eight smaller cells. This process is repeated recursively and finishes when the area of the cell's face is smaller than a constant times the area of the boundary face. In this implementation a factor of 0.4 was used. The use of this factor is recommended by other work found in the literature [Shephard & Georges 1991] to avoid excessive refinement in the octree generation. This process is repeated for all faces of the input data. The results are illustrated for the

two-dimensional example in Figure 6-b. The previous step can leave large octree cells in the interior of the region. In a second step, the octree is refined to guarantee that no cell in the interior is bigger than the largest cell at the boundary. This will avoid excessively large elements in the domain interior. Figure 6-c shows the resulting quadtree after this operation for the two-dimensional example. The octree is further processed in a third step, to force only one level of refinement between neighboring cells. This enforces a natural transition between regions of different degrees of refinement. Traversing the octree and examining the level of refinement between adjacent cells perform this operation. If the difference is more than one level, the appropriate cells are refined until the criterion is satisfied. Figure 6-d shows the quadtree generated for the two-dimensional example after this procedure.

The advancing front technique starts with a surface that bounds a region. Volume elements are “extracted” or “pared” from the region one at a time. As each element is extracted, the bounding surface is updated and the process is repeated. The procedure terminates when the entire region is meshed, or when one or more internal unmeshed cavities remain, from which valid elements cannot be extracted. In the present algorithm, the advancing front process is divided into three phases to ensure generation of valid volume meshes. In the first phase, a geometry-based element generation is pursued to generate elements of optimal shape. After this ideal phase is exhausted, and no more optimal elements can be generated, a topology-based element generation takes place, trying to create valid, but not necessarily well shaped, elements in the remaining region. In the last phase, a backtracking procedure is used to delete element faces that are preventing the algorithm from completing a mesh.

Ideally, the entire mesh would be generated in the geometry-based phase. However, this depends on the geometry and topology of the given boundary model and is strongly related to the shape quality of the given boundary mesh. The process starts with the creation of the initial advancing front, which is formed by the given boundary mesh. The current boundary mesh is stored in two separated lists. The first is a list of active faces, which includes all boundary mesh faces that have not been used in an attempt to generate valid tetrahedra. The other is a list of rejected faces, that is, with the faces that failed in the generation of elements for the current phase. Initially, all faces of the given boundary mesh are stored in the first list, which is the list used in the geometry-based generation phase.

In the geometry-based element generation phase, the current boundary mesh advances by trying to form tetrahedra based mainly on geometrical considerations. At each step, a triangular boundary face, referred to as base face, is chosen from the list of active faces. All existing faces represent the current front at a certain time of the algorithm. The procedure for generating a tetrahedron in this phase is explained by means of Figure 7. The optimal location N1 for the vertex of the tetrahedron to be formed is determined with the help of the octree. The octree cell containing the centroid M of the base face is determined. The optimal point N1 lies on a line perpendicular to the base face passing through this centroid. The distance from the optimal point to the base face centroid is equal to the octree cell size. The optimal point defines a search region where the vertex of the new tetrahedron may be located. This region is a sector of a sphere whose center is the optimal point and whose radius is proportional to the octree cell size. In the current implementation proportionality constant of 1.0 was adopted. This sphere defines an upper bound for the distance between the target vertex of the tetrahedron and the centroid of the base face. A lower bound is also defined to ensure that the generated tetrahedron will have volume greater than the smallest acceptable volume. In the current implementation, this lower bound is defined by a tetrahedron with height equal to 1/10 of the distance between N1 and M. The optimal region is used for two reasons. First, to ensure shape quality of the elements to be generated and, second, to ensure that new internal nodes will be created only when it is strictly necessary and always in good positions. Figure 7 shows, based on the bounds described, that points Q1 and Q2 are acceptable for forming a new tetrahedron, while points Q3, Q4, and Q5 are not. If no existing node is inside the optimal region, a new node is inserted at the optimal location N1 and an element is generated using this node. If only one node exists in the region, this node is used to generate the element. If more than one node is found in the region, they are ranked according to the solid angle that they will create with the base face. The node that will create the largest solid angle is used to generate the element. The solid angle is evaluated by projecting the base face onto a unit sphere of center at the candidate node and computing the area of the spherical polygon thus determined [Carvalho & Cavalcanti 1995]. Additional geometric checks are performed to ensure that the faces of the new element do not intersect any

existing face of the advancing front. If this is the case, the element is rejected. Once a valid tetrahedron is generated for the current base face, the list of active faces is updated. When there are no more faces in the list of active faces, the algorithm tries to generate elements using the faces that were rejected previously. Some base faces that failed previously might now work because the front has changed with the addition of elements. A similar procedure is mentioned in the literature [Moller & Hansbo 1995]. The geometry-based element generation phase ends when either there are no faces left in the boundary contraction lists (in which case an optimal mesh was generated), or when a rejected face fails a second time.

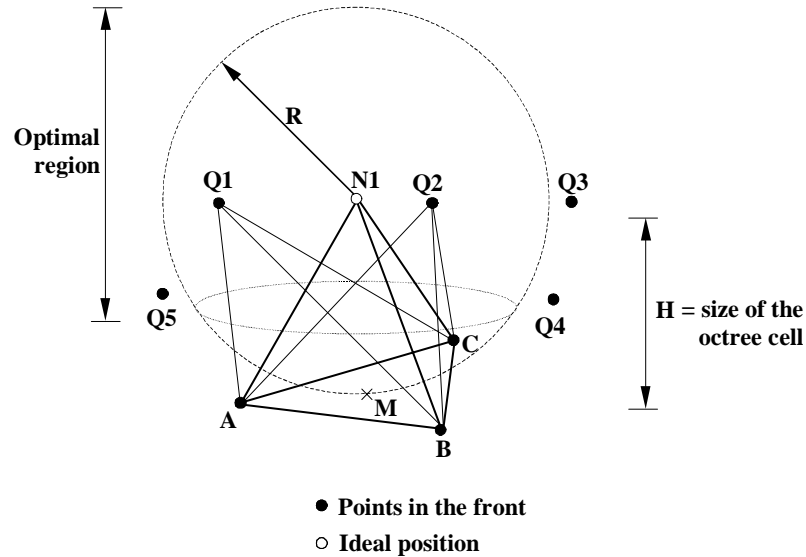


Figure 7 – The determination of a tetrahedron in the advancing front procedure.

The objective of topology-based element generation phase of the algorithm is to force the generation of valid tetrahedra if possible, even if the new elements do not satisfy the bounds used in the previous phase for element shapes. This phase starts when a boundary face fails twice in trying to generate an optimal element. The list of rejected faces of the previous phase is now considered as a list of active faces and, similarly to the geometry-based phase, a list of rejected faces is created for faces that eventually fail in generating valid tetrahedra. In the topology-based element generation phase, any node close to the current base face is selected and stored in a list of candidate nodes. The node that forms the best solid angle with the base face is chosen for the generation of the new tetrahedron. If the faces of this tetrahedron do not intercept any other face of the current advancing front, the element is created and the boundary is accordingly contracted. As in the geometry-based phase, the topology-based phase ends either when the list of active faces is empty or when a face of the advancing front is rejected twice due to intersection problems.

Sometimes the procedures performed in the previous phases are not sufficient to generate a valid mesh. The contracted boundary might end up in one or more polyhedra that cannot be meshed (cannot be subdivided into tetrahedra). One possible solution to this problem is to insert a node in the interior of a contracted region. Connecting this node to the region triangular boundary faces can then form valid elements. There are cases, however, in which such a construction is not possible. For example the resulting polyhedron might not have a “kernel region” in which any point is visible through a straight line from all its vertices. It is interesting to observe that this problem has no counterpart in two dimensions: any non-self-intersecting polygon can be triangulated with no need to insert additional vertices.

In the present algorithm, the solution to this problem is to locally modify the advancing front, deleting already generated adjacent tetrahedra until a “near” convex non-meshed polyhedron is formed. The procedure used to transform an ill-shaped polyhedron into one with a visible kernel is as follows. The boundary of the ill-shaped polyhedron related to a current base face is identified. A visibility test is performed. This consists of computing

the coordinates of the polyhedron centroid and counting the number of intersections that would occur, for each of the polyhedron's faces, if lines were drawn from the centroid to all of the polyhedron's vertices. If there is at least one intersection for any of the polyhedron's faces, the polyhedron must be modified. Removing the element attached to the face that has the highest number of intersections does this. This process is repeated until the centroid is visible from all nodes of the polyhedron. It is possible that the process of finding a "near" convex polyhedron may fail if faces to be removed are part of the original boundary mesh. When this occurs, the elements attached to internal faces with non-zero intersection counts are deleted and the element extraction procedure is restarted. If this polyhedron is still not meshable, the algorithm fails and terminates. In principle, it is possible to create a boundary input mesh that forces failure of the algorithm. Such a failure, however, has not yet been observed for several realistic input boundary meshes tested so far.

In the last two phases of the advancing front technique, poorly shaped tetrahedral elements might be generated. Two *a posteriori* local mesh modification procedures were implemented to improve mesh quality. The first is a conventional nodal relocation smoothing technique, which is based on node coordinates averaging, with validity checks. The second is a backtracking procedure, similar to the last phase of the advancing front technique, that deletes faces of poorly shaped elements to create a region where elements with better shape can be generated. The local mesh improvement procedures imply that element shape quality measures are necessary. Basically, any shape quality metric may be used. In the present case, the metric adopted is a normalized ratio between the root mean square of the lengths of the edges of a tetrahedron, and the volume of the tetrahedron [Weatherill & Hassan 1994]. This metric generates a good quality measure and is computationally efficient. The range of valid values varies from one to infinity ($[1, \infty]$) and the optimal value for the regular tetrahedron is approximately 8.5.

The objective of the backtracking procedure is to delete element faces surrounding a "bad" element to create a local polyhedron that can be remeshed with better-shaped elements. A local polyhedron to be meshed is created by deleting all elements adjacent to the "bad" element. This is illustrated by means of Figure 8, which shows a two-dimensional analogous case. After the creation of the local polyhedron, an attempt is made to generate elements by inserting a new internal node in the polyhedron's centroid, as shown in Figure 8. If this does not work, the backtracking procedure described before is employed.

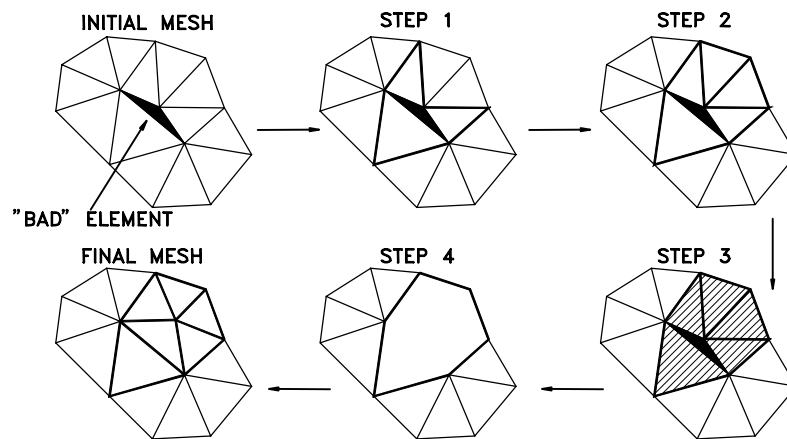


Figure 8 – Backtracking procedure to remesh around a "bad" element (2D example).

5 Example

An example is presented to illustrate the gFEM plug-in for finite element analysis. The example is a cube with 5.0 m of side and is composed by a material with the following properties:

Elasticity modulus: $E = 100.0 \text{ MPa}$

Poisson ratio: $\nu = 0.3$

Specific weight: $\gamma = 0.027 \text{ MN/m}^3$

To complete the finite element model, it is necessary to define boundary conditions and loading. A gravitational load is considered and the following displacement restrictions (supports) are used:

Side $X = 0$ and $X = 5$ is not allowed to move in the X direction;

Side $Y = 0$ and $Y = 5$ is not allowed to move in the Y direction;

Side $Z = 0$ is not allowed to move in the Z direction.

Figure 9 shows the original model geometry and its deformed configuration, in which the displacements values are scaled by a factor of 50.

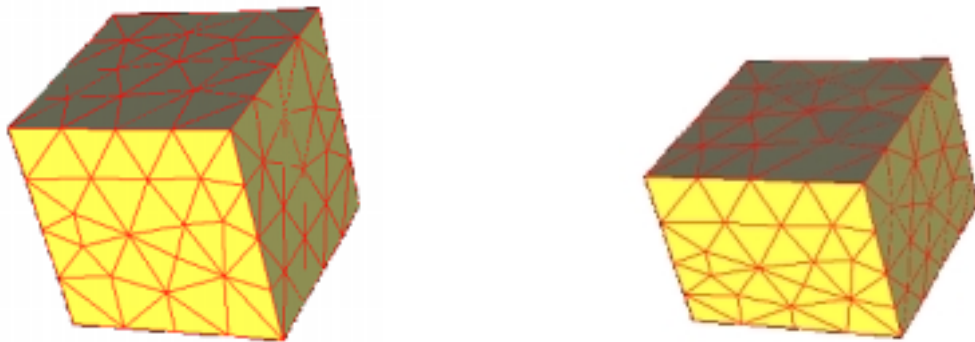


Figure 9 – Cube example with original geometry and deformed configuration.

Stress results of this analysis are shown in Figure 10. The principal stresses on the model boundary are shown in this figure.

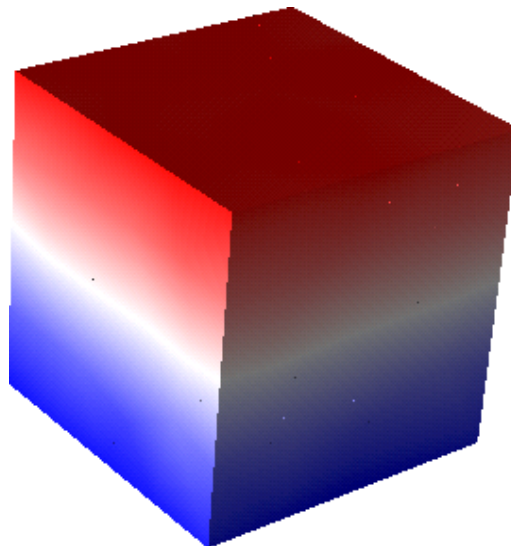


Figure 10 – Principal stress contour on model boundary.

6 Conclusion

A gOcad plug-in, called gFEM, was described in this paper. This plug-in adds capabilities to gOcad to act as a finite element mesh generator and as a finite element post-processor. Basically, the plug-in creates a class called MSolid that implements a new tetrahedral mesh generation algorithm, which may be used as an alternative to existing TSolid class. This algorithm has been recently published [Cavalcante Neto et al 2001] and has the following characteristics:

- It avoids producing elements with poor aspect ratios.
- It generates meshes that conform to existing triangular meshes on the boundary of a domain.
- It generates meshes that exhibit good transitions between regions of different element sizes.

The algorithm uses an advancing front technique (AFT), along with an octree to develop local guidelines for the size of generated elements. Although there are many algorithms that have been proposed in the literature in recent years, few of them seem to present the above characteristics in a complete satisfactory way. One of the reasons for this is the consideration of two additional steps. These steps, namely backtracking procedures, are heuristic attempts to avoid the problem of missing closure of the advancing front algorithm. The necessity of these procedures arises from the fact that, unlike triangulation in two dimensions, the discretization of any given volume into tetrahedra is not formally ensured, unless some additional steps are performed.

The advancing front method is divided into two phases. One, called the geometry-based phase, is based solely on the shape of the elements. In the other phase, called topology-based phase, the optimal element shape criteria are raised and the algorithm tries to create valid tetrahedra based only on topology, as in any advancing front method. This two-phase AFT differentiates this algorithm from other AFT-based meshing algorithms. To improve mesh quality (as far as element shape is concerned), an *a posteriori* local mesh improvement procedure is used.

Several works in the literature have addressed the solution to this problem recently. For example, the work of Chan and Anastasiou [1996] uses local mesh regeneration based on the deletion of sliver tetrahedra in a post-processing step. The *a posteriori* backtracking procedure of the present algorithm has the same objective of that step, but uses a different algorithm. In another recent work, Rassineux [1998] also optimizes the mesh by reconstruction of sub-volumes that are obtained by the deletion of a group of tetrahedra. It is worth mention that, in the present algorithm, the backtracking procedure is not only used in a post-processing stage to improve mesh quality but also during the advancing front phase, when a sub-volume that cannot be subdivided into tetrahedra is encountered. In this case, not only the validity of the mesh is enforced but also mesh quality is improved.

Another important difference between the current algorithm and the ones referred to above is that those algorithms generate internal nodes inside the domain in a prior step, while in the present algorithm internal nodes are generated simultaneously with element generation. Rassineux uses an octree procedure to generate internal nodes prior to the element generation. The current algorithm also uses an octree, but only as a node-spacing function. This approach tends to have a better control over the quality of the generated mesh and, apparently, decreases the amount of heuristic cleaning-up procedures.

The gFEM plug-in also extends gOcad to communicate with a finite element analysis program. This communication is done through ASCII files in a neutral format. This neutral file is used for both sending input data to finite element analysis and receiving simulation data from the analysis. Since in gOcad properties are attached to vertices, finite element results, usually specified at integration points, are extrapolated to nodal points and locally smoothed so that they can be visualized using gOcad graphics tools.

A simple example was presented to illustrate the capabilities of the gFEM plug-in.

References

- Anastasiou, K.; Chan, C. T. (1996), "Automatic Triangular Mesh Generation Scheme for Curved Surfaces." *Communications in Numerical Methods in Engineering*, **12**, 197-208.
- Carvalho, P. C. P.; Cavalcanti, P. R. (1995), "Point in Polyhedron Testing Using Spherical Polygons." In: Paeth, A. W. (Editor), *Graphics Gems V*, Academic Press, 709-749.
- Cavalcante Neto, J. B.; Wawrzynek, P. A.; Carvalho, M. T. M.; Martha, L. F.; Ingraffea, A. R. (2001), "An Algorithm for Three-dimensional Mesh Generation for Arbitrary Regions with Cracks." *Engineering with Computers*, **17**, 75-91.
- Chan, C. T.; Anastasiou, K. (1997), "An Automatic Tetrahedral Mesh Generation Scheme by the Advancing Front Method." *Communications in Numerical Methods in Engineering*, **13**, 33-46.
- Jin, H.; Tanner, R. I. (1993), "Generation of Unstructured Tetrahedral Meshes by Advancing Front Technique." *International Journal for Numerical Methods in Engineering*, **36**, 1805-1823.
- Joe, B. (1991), "Delaunay Versus Max-Min Solid Angle Triangulations for Three-dimensional Mesh Generation." *International Journal for Numerical Methods in Engineering*, **31**, 987-997.
- Lewis, R. H.; Zheng, Y.; Gethin, D. T. (1996), "Three-dimensional Unstructured Mesh Generation: Part 3. Volume meshes." *Computer Methods in Applied Mechanics*, **134**, 285-310.
- Liu, A.; Joe, B. (1994), "Relationship Between Tetrahedron Shape Measures." *BIT*, **34**, 268-287.
- Lohner, R.; Parikh, P. (1988), "Generation of Three-Dimensional Unstructured Grids by the Advancing-Front Method." *International Journal for Numerical Methods in Fluids*, **8**, 1135-1149.
- Moller, P.; Hansbo, P. (1995), "On Advancing Front Mesh Generation in Three Dimensions." *International Journal for Numerical Methods in Engineering*, **38**, 3551-3569.
- Parthasarathy, V. N.; Graichen, C. M.; Hathaway, A. F. (1993), "A Comparison of Tetrahedron Quality Measures." *Finite Element Analysis and Design*, **15**, 255-261.
- Peraire, J.; Peiro, J.; Formaggia, L.; Morgan, K.; Zienkiewicz, O. C. (1988), "Finite Euler Computation in Three-Dimensions." *International Journal for Numerical Methods in Engineering*, **26**, 2135-2159.
- Rassineux, A. (1998), "Generation and Optimization of Tetrahedral Meshes by Advancing Front Technique." *International Journal for Numerical Methods in Engineering*, **41**, 651-674.
- Shephard, M. S.; Georges, M. K. (1991), "Automatic Three-Dimensional Mesh Generation by the Finite Octree Technique." *International Journal for Numerical Methods in Engineering*, **32**, 709-749.
- Weatherill, N. P.; Hassan, O. (1994), "Efficient Three-Dimensional Delaunay Triangulation with Automatic Point Creation and Imposed Boundary Constraints." *International Journal for Numerical Methods in Engineering*, **37**, 2005-2039.