International Journal of Advanced Engineering Research and Science (IJAERS)
Peer-Reviewed Journal
ISSN: 2349-6495(P) | 2456-1908(O)
Vol-8, Issue-7; Jul, 2021
Journal Home Page Available: https://ijaers.com/
Article DOI: https://dx.doi.org/10.22161/ijaers.87.44

A Simplified Mesh Generation Scheme for 3D Geometries Composed by Planar Faces to be used with BEM

Geraldo Creci

Instituto Federal de Educação, Ciência e Tecnologia de São Paulo (IFSP-BRA)
Av. Major Fernando Valle, 2013 - 12903-000, Bragança Paulista, SP, Brasil.
Email: gcreci@ifsp.edu.br – ORCID: 0000-0003-1578-6520

Received:11 Jun 2021;

Received in revised form: 12 Jul 2021;

Accepted: 19 Jul 2021;

Available online: 30 Jul 2021

***Keywords*— boundary element method (BEM), computational geometry, Delaunay triangulation, geometrical transformations, unstructured mesh generation.**

***Abstract*—** *This paper presents the development of a simplified mesh generation scheme for three-dimensional boundary element method (BEM). The developed program presented in this work takes into account three-dimensional geometries composed by planar faces. This is a simplification regarding the form of the acceptable geometries, but a great variety of geometries and problems can be modelled with sufficient accuracy to perform numerical analysis. The main program's idea consists of moving each face belonging to the three-dimensional geometry to bi-dimensional space using geometrical transformation matrices. In bi-dimensional space, then, it is applied a bi-dimensional mesh generation algorithm for element generation. After element generation is done, new geometrical transformations are applied in order to send the meshed face back to its original position in three-dimensional space. The continuity of the final mesh is assured by promoting the discretization of the edges of the geometry only once. The proposed scheme has several positive aspects regarding simplicity, efficiency and robustness. A practical and immediate use of the proposed algorithm can be found for those who already have a bi-dimensional mesh generator code implemented and intent to expand its functionalities to treat simple three-dimensional geometries composed by planar faces. Several examples of meshes in arbitrary three-dimensional geometries composed by planar faces are presented in order to illustrate the capabilities of the developed computational program and some computational simulations have been performed to show the quality of the meshes in problems with specific boundary conditions.*

I. INTRODUCTION

The mesh generation process is crucial for computational simulations since it strongly affects the results obtained by the numerical solution of partial differential equations that govern the analyzed problem [1-3]. The generated mesh should be able to provide sufficiently accurate results according to the complexity of the geometry and physical phenomena involved. Several complex applications can be analyzed with sufficient accuracy level [4,5]. In computational numerical simulations, the pre-processing phase is one of the "bottle-neck" with respect to the computer processing time. This is due to the fact that in the pre-processing phase the mesh is generated and the problem is modelled with specific boundary conditions. It is very difficult to generate suitable meshes for problems involving complex three-dimensional geometries and/or boundary conditions. The algorithms for three-dimensional mesh generation are usually very difficult to be implemented for arbitrary cases and the computational costs are generally expensive [6]. That is why, still nowadays, great efforts and investments are done to develop new techniques and algorithms for

improving the mesh generation process on three-dimensional geometries. In this paper it is presented a simplified scheme to implement a three-dimensional mesh generator to be used in numerical analyses by the boundary element method. The boundary element method has some particular features that make it more favorable in certain types of applications, when compared to other numerical methods. Among those features, there is the fact that, in boundary element method, only a discretization of the geometry boundaries is necessary [7,8]. In other words, in three-dimensional cases only surface elements must be generated. This is a huge benefit from the mesh generation point of view. The algorithm proposed in this work only takes into account three-dimensional geometries composed by planar faces. This is a simplification regarding the form of the acceptable geometries, but a great variety of geometries and problems can be modelled with sufficient accuracy to perform numerical analysis. The main algorithm's idea consists of moving each face belonging to the three-dimensional geometry to bi-dimensional space using geometrical transformation matrices. In bi-dimensional space, then, it is applied the Delaunay triangulation method over a grid of points to cover the face domain with triangular-linear elements, but other algorithms and elements could also be used taking into account the general idea. The Delaunay triangulation method is one of the best methods to generate meshes with triangular elements devoted to numerical analysis. It has the property of maximizing the triangles' minimum internal angles which is favorable and suitable for numerical methods in general [9,10]. After element generation is done, new geometrical transformations are applied in order to send the meshed face back to its original position in three-dimensional space. The continuity of the final mesh is assured by promoting the discretization of the edges of the geometry only once. This is essential because each edge belongs to two planar faces of the three-dimensional geometry. Therefore, no duplicate nodes should be generated over an edge and the final mesh is supposed to contain only conformal elements. The proposed scheme has several positive aspects regarding simplicity, efficiency and robustness. Firstly, the code is extremely simplified since the problem is transformed from three-dimensional space to bi-dimensional space by using geometrical transformations matrices. The algorithms for element generation are bi-dimensional, that is to say, they are much simpler, reliable and easier to be implemented [11-13]. Regarding efficiency and robustness, as each face of the three-dimensional geometry is moved to bi-dimensional space and the mesh is generated over one face per time, the occurrence of a bad element generation is minimized. As the whole mesh is performed face by face, the problem is subdivided in several small problems and the main algorithm assumes the divide-to-conquer paradigm, which assures high efficiency and computer processing velocity [14,15]. A practical and immediate use of the proposed algorithm can be found for those who already have a bi-dimensional mesh generator code implemented and intent to expand its functionalities to treat simple three-dimensional geometries composed by planar faces. The generated surface meshes over these geometries can be used in three-dimensional boundary element analysis or, even though, in numerical analysis using the shell finite element formulation. All the code has been written using the object-oriented paradigm in C++ combined with UML notation. Using this approach the program can be easier modified, the maintenance costs are reduced and new implementations can be carried out as user's and/or programmer's needs [16].

II. THREE-DIMENSIONAL BOUNDARY ELEMENT METHOD

The boundary element method is based upon boundary singular integral equations. The analytical formulation involves the transformation of the governing differential equation applicable to the whole domain into an integral over the boundary [7]. To illustrate the technique it is presented the elasticity problem in the form of the partial differential equation known as the Navier equation of elasticity:

$$\mu u_{i,jj} + \frac{\mu}{1-2v} u_{j,ji} + b_i = 0, \quad (1)$$

where b_i are body force components, $\mu = E/2(1+v)$ is the shear modulus, E is the Young's modulus, v is the Poisson's ratio and u_i are displacements components. Equation 1 can be transformed into an integral equation over the boundary. The displacement boundary integral formulation of elasticity can be derived using the Somigliana's Identity and Betti's reciprocity theorem [8]. The fundamental solutions for displacements and tractions are given respectively by:

$$U_{ij}(X', X) = \frac{1}{16\pi(1-v)\mu R}\{(3-4v)\delta_{ij} + R_{,i}R_{,j}\} \quad (2)$$

$$T_{ij}(X', X) = \frac{-1}{8\pi(1-v)R^2}\{\frac{\partial R}{\partial n}\left[(1-2v)\delta_{ij} + 3R_{,i}R_{,j}\right] - (1-2v)\left(n_j R_{,i} - n_i R_{,j}\right)\} \quad (3)$$

where $R = \sqrt{R_i R_i}$, $R_i = X_i - X'_i$, $R_{,i} = \frac{\partial R_i}{\partial X_i} = \frac{R_i}{R}$ and $\frac{\partial R}{\partial n} = R_{,i} n_i$. The load point or source point is represented by X' and the field point by X. Therefore, the displacement boundary integral equations for elasticity can be written, neglecting the body forces b_i, by considering the limiting process of an internal point that goes to the boundary [7,8], i.e., $X' \to x'$, as:

$$c_{ij}(x')u_j(x') + \int_\Gamma T_{ij}(x',x)t_j(x)d\Gamma(x) = \int_\Gamma U_{ij}(x',x)t_j(x)d\Gamma(x) \quad (4)$$

where $c_{ij}(x')$ is a function of the contour shape at the boundary point x'. From Equation 4 one can note that there are only boundary dependent terms. Thus, it is not necessary to generate domain elements in the discretization procedure. That is why the three-dimensional mesh generator implemented in this work only generates surface elements. For simplicity, the mesh generator was implemented by using a bi-dimensional Delaunay mesh generation algorithm, which gives origin to triangular-linear elements, as shown by Figure 1. Therefore, a boundary element solver with three-dimensional elasticity isotropic-linear homogeneous formulation, for example, can be used to perform numerical analyses considering solid mechanics engineering problems. Using the shape functions N^1, N^2, N^3, Equation 4 can be written in a discretized form as [7]:

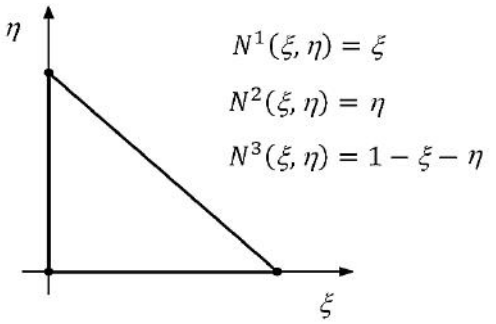

Fig.1: Triangular-linear continuous element and its shape functions.

$$c_{ij}(x^\kappa)u_j(x^\kappa) + \sum_{n=1}^{N}\sum_{m=1}^{M} u_j^{nm} \int_{-1}^{1}\int_{-1}^{1} T_{ij}[x^\kappa, x(\xi,\eta)]N^m(\xi,\eta)J^n(\xi,\eta)d\xi d\eta = \sum_{n=1}^{N}\sum_{m=1}^{M} t_j^{nm} \int_{-1}^{1}\int_{-1}^{1} U[x^\kappa, x(\xi,\eta)]N^m(\xi,\eta)J^n(\xi,\eta)d\xi d\eta \quad (5)$$

where M is the number of nodes per element, N is the total number of elements on the mesh, κ represents the number of the node on the mesh that is being evaluated as source point and J^n is the Jacobian of the transformation. The Jacobian can be calculated by $J_n = |x_\xi \times x_\eta|$, where the subscripts ξ and η denote the derivatives with respect to ξ and η, respectively. Equation 5 can be presented in the matrix form as:

$$Hu = Gt, \quad (6)$$

which can be rearranged in $Ax = f$ and readily solved.

III. MESH GENERATION

This work uses bi-dimensional Delaunay triangulation over a grid of points to generate elements in an arbitrary planar faces of the three-dimensional geometry. There are several ways to perform bi-dimensional mesh generation over an arbitrary planar straight line graph (PSLG). As bi-dimensional algorithms for mesh generation are extensively know in literature, the main focus of this paper is devoted to present the structure of the program that allows three-dimensional surface mesh generation. To perform this task, geometrical transformation matrices are used to move each planar face from the three-dimensional geometry to bi-dimensional space and vice-versa.

Data Structure

The process to obtain the input file for the mesh generator presented in this work is illustrated by Figure 2. First, the three-dimensional geometry is drawn using CAD software. The three-dimensional geometry must only be constituted by planar faces. Then, the geometry is saved on IGES format and an IGES translator developed during this research project is used to generate the input file accepted by the mesh generator program. This input file contains the geometry information in the format presented by Figure 3(a). A similar data structure for the input file can be found in [11]. The MeshPar parameter is a numerical value provided by the user. It is given in length unit. This parameter affects the level of the mesh refinement. The key-words Vertex, Edge and Facet contain information about the vertexes, edges and planar faces of the 3D geometry, respectively. The format of the output file containing the mesh data is presented by Figure 3(b).

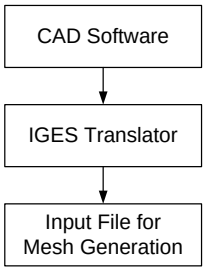

Fig.2: Process to obtain the input file for the mesh generator.

```
MeshPar = [ (meshpar_value) ]

Vertex=[
    (number) (coordinate_X  coordinate_Y  coordinate_Z)
    …
]

Edge=[
    (number) (vertex_A  vertex_B)
    …
]

Facet=[
    (quantity_of_edges) (number) (edge_A)  (edge_B)
(edge_C) (edge_D)  …
    …
]
```

(a)

```
NodeNumber = number_of_nodes

ElementNumber = number_of_elements

Nodes=[
    (number) (coordinate_X  coordinate_Y  coordinate_Z)
    …
]

Elements=[
    (number) (node_A  node_B  node_C)
    …
]
```

(b)

Fig.3: Data structures: (a) input file and (b) output file.

Computational Aspects

In this subsection the main classes of the developed program are presented. The classes were modeled by using the UML notation. A special focus is given to the Node, Segment, Facet, Element, Triag, Mesh and GeoTrafo classes, which are the main classes of the program. In Figure 4 are presented the Node, Element, Triag and Centroid classes. It is also possible to see the Dictionary class, with its hidden attributes and methods. The Dictionary class is very important for the developed program because it makes possible to read the input data file easily and efficiently. It can be seen from the diagram presented in Figure 4 that both the Element and Node classes depend on the Dictionary class. In fact, the Segment and Facet classes also depend on it, but they were not shown in this diagram for simplification purpose.

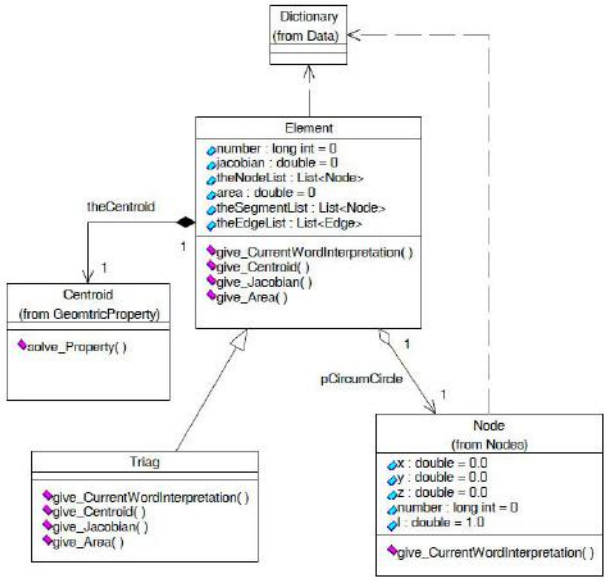

Fig.4: Class diagram exhibiting the relationship between Node and Element Classes.

The Element class has a number of attributes and methods, as can be seen. The signature of the methods, i.e., the type of method and the arguments that are passed to it were not shown at the diagram for visual simplicity. The give_CurrentWordInterpretation() method is a static method and, therefore, it can be called without the need to instantiate an object. It is only necessary to use the class name and the "::" operator followed by the name of the method. The Triag class is the only class derived from the Element class, because generated meshes consist only of triangular elements. The give_CurrentWordInterpretation() method is an specialist that has the ability to interpret input file information related to a word previously added to the dictionary. The Node, Segment, and Facet classes also have the give_CurrentWordInterpretation() method to read the data structure shown by Figure 3. The other methods of the Element class, give_Centroid(), give_Jacobian(), and give_Area() are virtual methods, which allow the assignment of Element-type pointers to Triag-type objects.

Thus, the code for calculating the centroid, the Jacobian and the area of a triangular element are implemented in the methods of the Triag class. It is possible to note two associations existing in Figure 4: an association between the Element class and the Node class; and, an association between the Element class and the Centroid class. In the first, a Node-type object is aggregated by reference to the Element class, which implies that every Element-type object will have a Node-type attribute, called pCircumCircle, aggregated by reference. This pCircumCircle attribute will have the coordinates of the center of the circle that circumscribes a Triag element. In the second, an object of the Centroid-type, theCentroid, is added by value to the Element class. Figure 5 shows the Segment, Facet, and Edge classes. Also, the List and Container classes are presented. They are Template classes, that is, they are parameterized. Note that the List class depends on the Node, Segment, Facet, Edge and Element classes, because it is necessary to create lists of Node-type objects, Segment-type objects and so on.

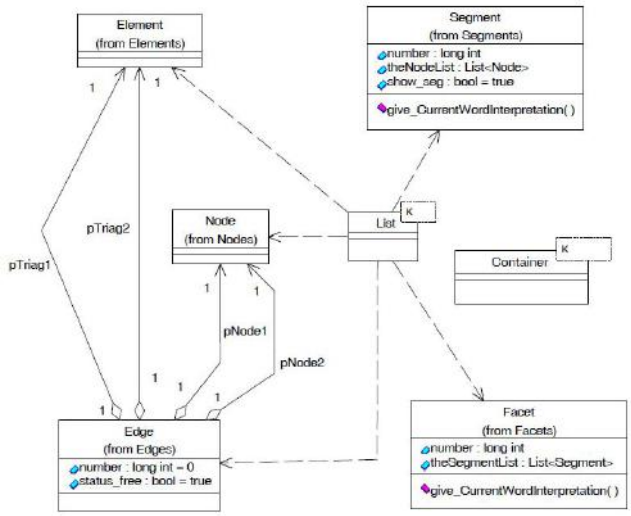

Fig.5: Class diagram showing the modeling of Segment, Facet and Edge classes.

The Edge class was abstracted aiming at the improvement of the element generation algorithm. With the Edge class it is possible to know the geometric entities in a certain region of interest more quickly and efficiently. For example, as the Edge class relates to the Element class and the Node class, it is possible to see that each Edge-type object has two Element-type objects (pTriag1 and pTriag2) and two Node-type objects (pNode1 and pNode2). Thus, for a particular Edge-type object, it is immediately known which are the two triangles of the mesh that are part of it, and which are the two nodes that define it. This is extremely beneficial from the processing time saving point of view, because, it is not necessary to search for a whole set of geometric entities, but only in a particular region of interest. Figure 6 shows the GeoTrafo and Mesh classes that are the most important classes of the developed program. Again, to facilitate viewing of the classes, the signatures of the methods were not displayed.

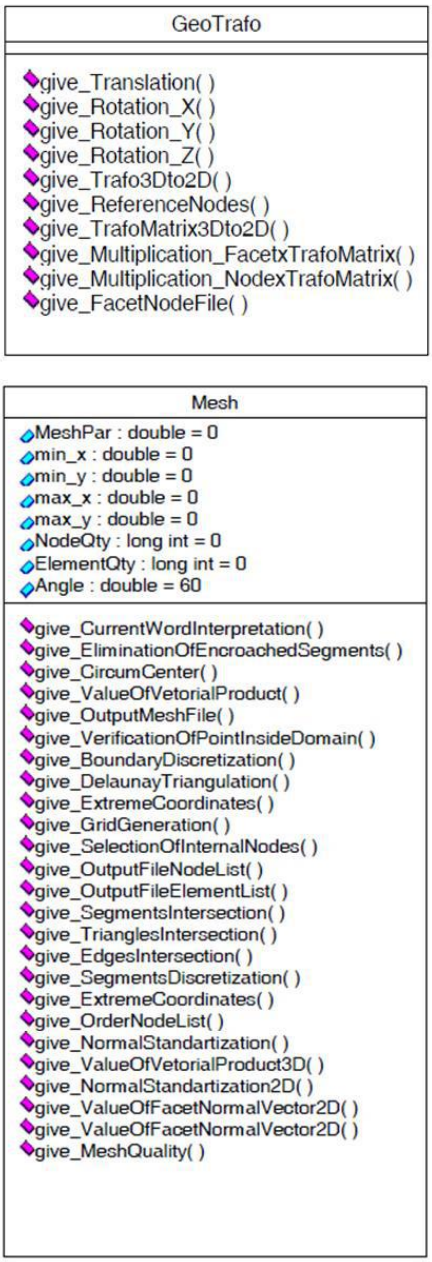

Fig.6: GeoTrafo and Mesh classes modeled using UML notation.

Mesh Generator Flowchart

The execution flowchart of the implemented mesh generation program is presented in Figure 7. The mesh

generation process begins with the input file reading. As soon as this file is read, the computer has already stored in memory all lists needed to generate the mesh on the geometry. Three lists are created: a VertexList which contains Vertex-type objects with information about the geometry vertexes; an EdgeList which contains Edge-type objects with information about the geometry edges; and a FacetList which contains Face-type objects with information about the planar faces of the geometry. The MeshPar value is stored in a double type variable. The mesh generation process starts from a loop in the FacetList. For each face belonging to the FacetList, it is applied a geometrical transformation matrix which will move it to bi-dimensional space. After the face is in bi-dimensional space, a loop in the EdgeList is performed, which defines the current face. Each face owns as attribute an EdgeList which contains all the segments of the face. Each segment of the face owns as attribute a VertexList which contains the Vertex-type objects that define the segment. Initially, it is checked if the VertexList of the current segment has only two Vertex. This means that the segment has not been discretized. If the VertexList of the segment contains more than two Vertex, it means that the segment has already been discretized and should not be discretized again. This verification is crucial because each segment belongs to two faces. So, in order to respect the continuity of the final mesh, the nodes on the edges of the geometry are defined a priori and only once. Afterwards the loop in the EdgeList of the Face object is finished, it is called the method that generates the mesh over the face using a grid of points. Good references to implement a bi-dimensional mesh generator using the Delaunay triangulation method, can be found in [12,17-19]. However, there are several approaches to generate meshes with different types of elements [13]. An advancing front approach, instead of generating a grid of points over a region, is also very efficient for generating triangular meshes [14]. This technique is particularly suitable for the boundary element method because it starts the element generation from the boundary data. An execution flowchart using the advance front technique for mesh generation, whose structure is similar in some aspects to the proposed flowchart, can be found in [15]. As soon as the mesh on the face is generated in a conformal procedure, the inverse of the transformation matrix, used to move the face to bidimensional space, is applied to send the meshed face back to its original position in 3D space. Then, another face of the geometry is selected and the process goes on until all the faces have been analyzed. At the end of the process, the program will have stored a NodeList containing the nodes of the final mesh and an ElementList containing the nodal connectivity. An output file containing this information is written and used as input file by the boundary element solver.

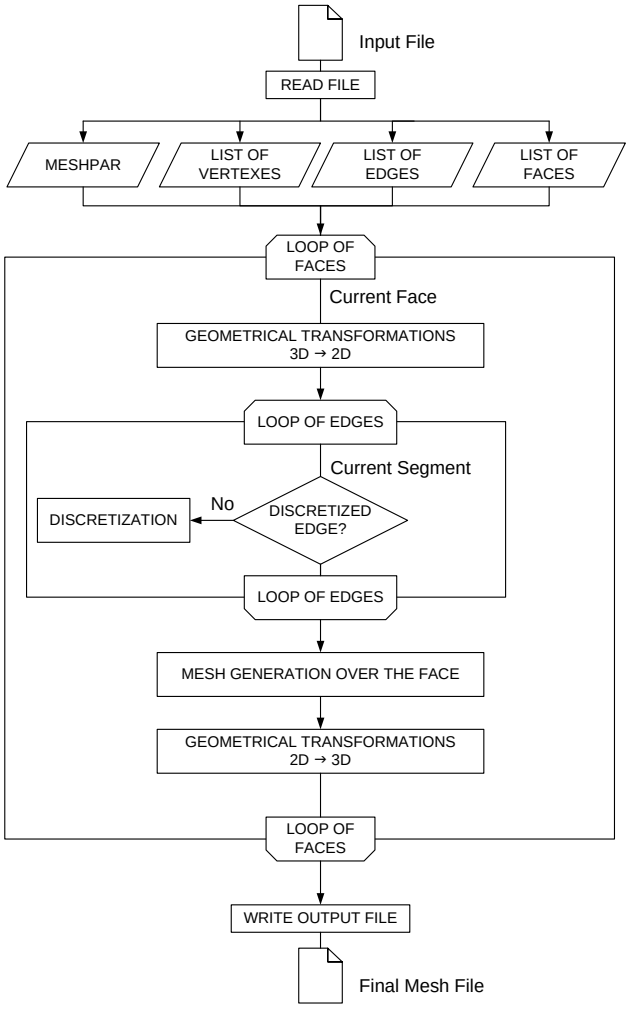

Fig.7: Execution flowchart of the developed mesh generation program.

Geometrical Transformations

The process of moving each planar face of the geometry from three-dimensional space to bi-dimensional xy plane is obtained by using geometrical transformation matrices [20]. In Figure 8 this process is illustrated step by step. In Figure 8(a) it is presented an arbitrary planar face in three-dimensional space constituted by linear segments. This face can be a non-convex polygon and have holes in its interior. First, it is detected the point of the face which has the minimum y-coordinate, in this case P_{ij}. Then it is applied a translation matrix in order to move all the face entities to the origin of the coordinate system, as presented in Figure 8(b). The translation matrix $\mathbf{A}$ is given by:

$$A_{ij} = \delta_{ij} + \delta_{i4}(-P_{ki}) - \delta_{i4}(-P_{ki})\delta_{j4} \quad (7)$$

where P_{ki} is the point used for the face translation. The subscript k is the number of the point, i represents the cartesian components of the point, and, δ_{ij} is the Kronecker delta function. The vector $\boldsymbol{V_n}$ is the face normal vector calculated by the cross product between $\boldsymbol{V_1}$ and $\boldsymbol{V_2}$. As soon as the face is moved to the origin of the coordinate reference system, it is necessary to apply a geometrical transformation matrix $\mathbf{B}$ in order to align $\boldsymbol{V_n}$ with z-axis and $\boldsymbol{V_2}$ with y-axis. This is shown by Figure 8(c). The matrix $\mathbf{B}$ is given by:

$$B_{ij} = R_{ij} - \left(\delta_{4j}R_{ij} + \delta_{i4}R_{ij}\right) + \delta_{i4}\delta_{j4}R_{ij} + \delta_{i4}\delta_{j4} \quad for \quad i,j = 1,4 \quad (8)$$

where R_{ij} are the cartesian components of the $\boldsymbol{R_i}$ vectors. The $\boldsymbol{R_i}$ vectors can be calculated by $\boldsymbol{R_3} = \frac{\boldsymbol{V_n}}{|\boldsymbol{V_n}|}$, $\boldsymbol{R_1} = \frac{\boldsymbol{V_2}\times\boldsymbol{R_3}}{|\boldsymbol{V_2}\times\boldsymbol{R_3}|}$ and $\boldsymbol{R_2} = \boldsymbol{R_3} \times \boldsymbol{R_1}$. So, the transformation matrix $\mathbf{T}$ to be implemented is given by the product of matrices $\mathbf{B}$ with $\mathbf{A}$, as:

$$T_{ij} = B_{ik}A_{kj}. \quad (9)$$

Applying on a set of points P, the face F_i is moved to xy plane as shown in Fig. 8(d) by the transformation:

$$P_{ij}^{T} = T_{ik}P_{kj} \quad (10)$$

The inverse of the transformation matrix $\mathbf{T^{-1}}$ is used to send the planar face back to its original position in three-dimensional space.

IV. RESULTS AND DISCUSSIONS

Generated Meshes

In order to show the capabilities of the developed mesh generator, it is presented four meshes in arbitrary three-dimensional geometries composed by planar faces, see Figures 9 to 12. Since the developed mesh generator uses

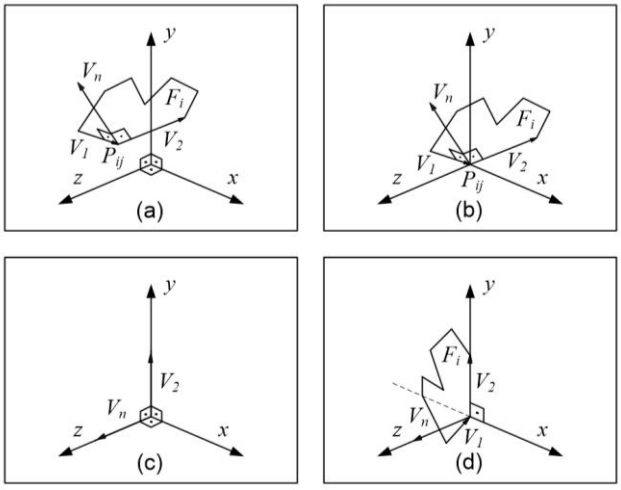

Figure 8: Process of moving an arbitrary planar face F_i from 3D space to 2D xy plane.

the Delaunay triangulation method, the triangles' minimum internal angles found in every mesh are significantly high, being limited only by the minimum angle of the drawing geometry. Therefore, it can be noticed that all presented meshes shown good quality and are suitable for numerical analysis using the three-dimensional boundary element method or shell finite element formulation. The meshes presented in this paper are for illustration purposes and, thus, they have a very small number of nodes and elements. However, the MeshPar parameter can be easily adjusted as user's needs in order to generate more refined meshes, with much more nodes and elements, according to computational hardware availability.

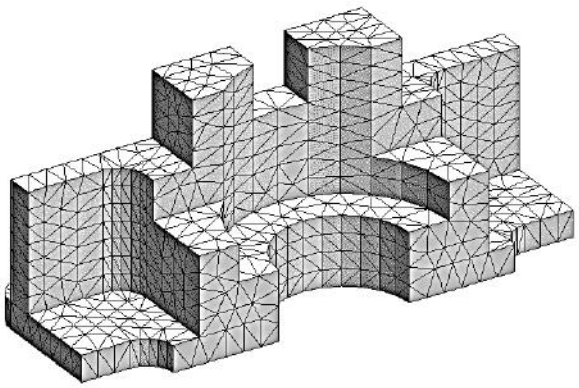

Fig.9: Three-dimensional surface mesh with 1148 nodes and 2292 elements.

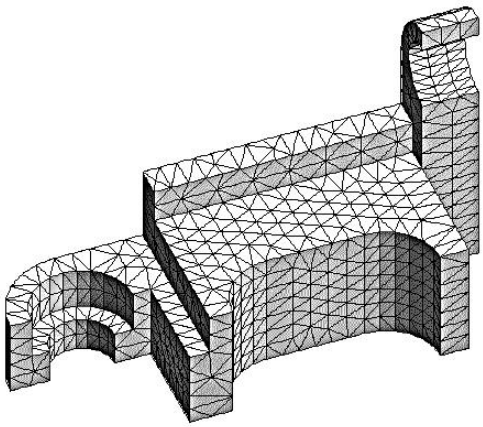

Fig.10: Three-dimensional surface mesh with 957 nodes and 1910 elements.

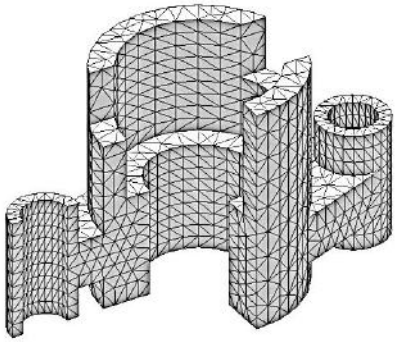

Fig.11: Three-dimensional surface mesh with 1737 nodes and 3474 elements.

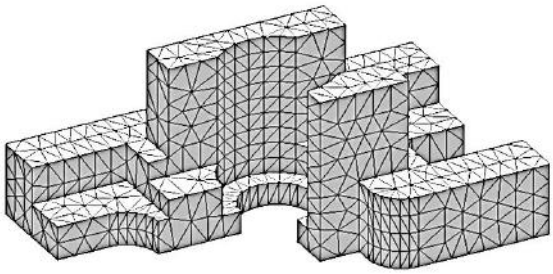

Fig.12: Three-dimensional surface mesh with 1058 nodes and 2116 elements.

Computational Simulations

Some computational simulations were performed in order to demonstrate practical applications of the generated meshes. The first problem analyzed is a tub with internal pressure. The Poisson ratio used is equal to 0.3 and the Young´s Modulus is 2600 units. The boundary conditions applied are:

- x-direction displacement restriction on the nodes of the face parallel to yz plane;
- y-direction displacement restriction on the nodes of the face parallel to xz plane;
- z-direction displacement restriction on the nodes of the faces parallel to xy plane;
- 50 units of internal pressure applied on the elements of the internal wall tub.

Figure 13 shows the displacement map for the studied problem considering that the modeled mesh is generated by the mesh generator presented by this work having 200 nodes and 396 triangular-linear elements; and, the solution is calculated by a boundary element solver, ECon-3D, which was developed prior to this research project. In a similar way, the same problem was analyzed again considering a modeled mesh generated by the Ansys software having 212 nodes and 420 Shell63 triangular-linear elements; the solution shown by Figure 14 was also calculated by ECon-3D solver. It can be seen that the mesh modeling strongly influences the results. The analytical solution for this problem can be found in [7] by means of comparison. A good agreement can be observed among all the achieved results.

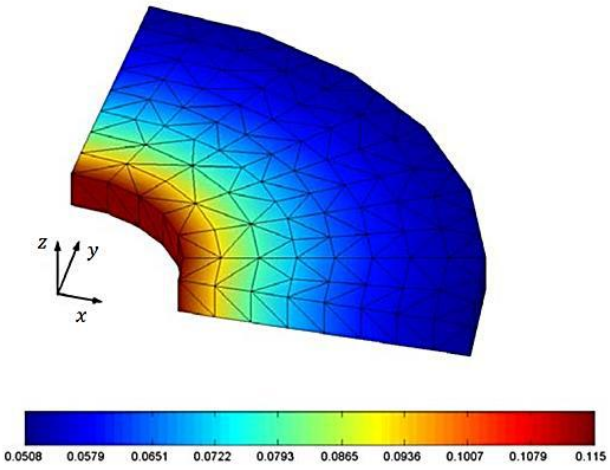

Fig.13: Displacement map for a mesh generated by the mesh generator developed by this work; the solution was carried out by Econ-3D boundary element solver.

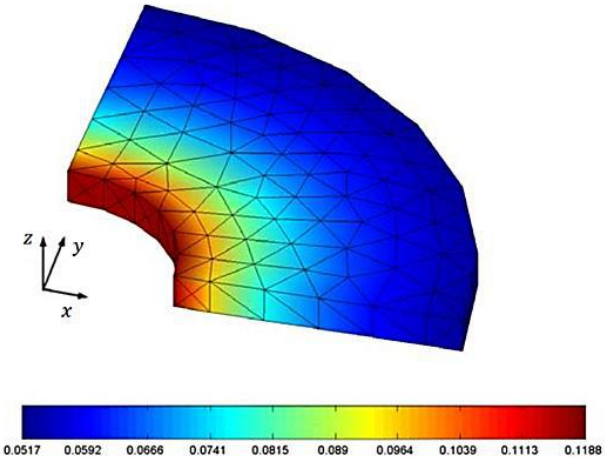

Fig.14: Displacement map for a mesh generated by the Ansys software; the solution was carried out by Econ-3D boundary element solver.

A second case study was performed considering the structural support whose geometry is presented by Figure 15. To analyze this problem, a Poisson ratio of 0.21 and a Young´s Modulus of 2900 units were adopted. The applied boundary conditions applied are:

- x- and z- direction displacement restriction on the nodes of the two holes;
- y- direction displacement restriction on the nodes of the two outer side faces containing the holes;
- 500 units of pressure applied to the elements at the rod end.

Figure 15 shows the displacement map of the solution considering the problem modeled with a mesh generated by the Ansys software having 934 nodes and 1832 Shell63 elements. The respective solution of this problem was calculated out by ECon-3D solver. Figure 16 shows another displacement map of the solution considering the problem modeled with a mesh generated by the mesh generator developed by this work having 936 nodes and 1876 elements. The respective solution of this problem was also performed by ECon-3D solver. A final analysis was performed using the finite element method in the Ansys software by using Solid45 elements. Figure 17 shows the mesh generated by the Ansys software with 1618 nodes and 6013 elements. The Solid45 element was chosen because it has linear interpolation functions, thus allowing a fairer comparison among the results. All three analyzes showed very close results. This demonstrates that the meshes generated by the mesh generator developed by this work are of good quality and suitable for using in numerical analysis.

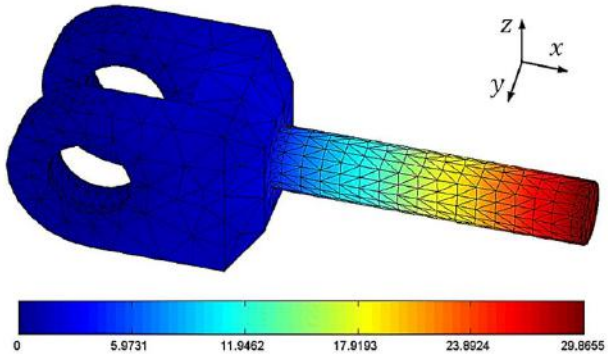

Fig.15: Displacement map for a mesh generated by the Ansys software; the solution was carried out by Econ-3D boundary element solver.

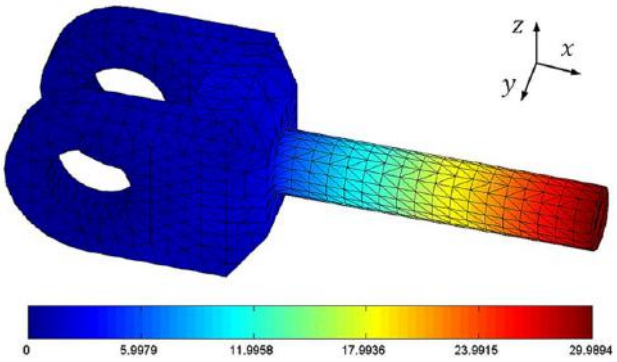

Fig.16: Displacement map for a mesh generated by the mesh generator developed by this work; the solution was carried out by Econ-3D boundary element solver.

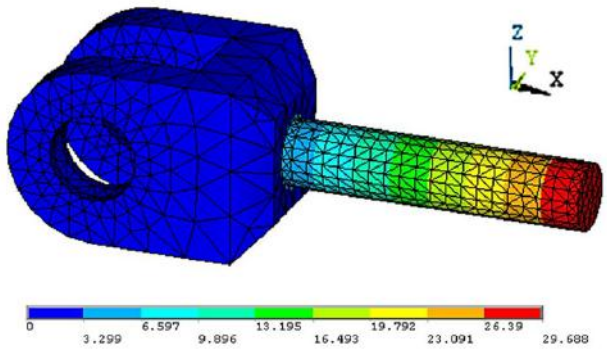

Fig.17: Displacement map for a mesh generated by the Ansys software with Solid45 elements; the solution was carried out by the Ansys solver.

V. CONCLUSIONS

It was developed a triangular-linear surface mesh generator for arbitrary three-dimensional geometries composed by planar faces. As programming was done using the object-oriented paradigm, it was possible to construct a very clear and efficient program structure. In addition, this structure is extremely flexible and allows for faster and easier code expansion and enhancement. Although the mesh generator developed by this work only accepts geometries composed by planar faces, it can be seen from the analysis of the presented results that a wide variety of three-dimensional problems can be modeled with considerable complexity. All the geometries analyzed by the program lead to the generation of satisfactory meshes which can be progressively refined according to the MeshPar parameter and the processing capacity of the computer hardware. The proposed algorithm makes use of geometrical transformation matrices which allow the development of a fast, robust and efficient program. The program flowchart can be particularly useful for those who already have a bi-dimensional mesh generator code implemented and intend to extend its functionalities to treat simple three-dimensional geometries composed by planar faces. Some computational simulations have been performed to show the quality of the meshes in problems with specific boundary conditions. It could be observed that the developed mesh generator provides triangular-linear elements with good technical features for numerical analysis considering the boundary element method or the shell finite element formulation. Besides, the developed program is an open source engineering program that can be used for research purposes. Future improvements can be done according to user's needs.

ACKNOWLEDGEMENTS

The author gratefully acknowledges the financial support from CAPES, FAEP (1301/03) and FAPESP (03/10832-1).

REFERENCES

[1] Sarti Leme A. D. et al. Finite Element Analysis to Verify the Structural Integrity of an Aeronautical Gas Turbine Disc Made from Inconel 713LC Superalloy. Advanced Engineering Forum, v. 32, p. 15-26, 2019. Available in: <https://doi.org/10.4028/www.scientific.net/AEF.32.15>

[2] Gao, Q. and Zhang, S. Moving mesh method for simulating high-dimensional time dependent PDEs with fast propagating shock waves. Engineering Analysis with Boundary Elements, v. 103, p. 116-125, 2019. <Available in: https://doi.org/10.1016/j.enganabound.2019.03.001>

[3] Creci Filho, G. Influência da dinâmica dos mancais na resposta vibratória de uma turbina aeronáutica de 5-KN de empuxo. São José dos Campos - SP: ITA, 2012. 307p. Tese (Doutorado) - Available in: <http://www.bdita.bibl.ita.br/tesesdigitais/lista_resumo.php?num_tese=62130>

[4] Creci, G. et al. Rotordynamic Analysis of a 5-kN Thrust Gas Turbine by Considering Bearing Dynamics. Journal of Propulsion and Power, v. 27, n. 2, p. 330-336, 2011. Available in: <https://doi.org/10.2514/1.B34104>

[5] Creci, G. et al. Influence of the Radial Clearance of a Squeeze Film Damper on the Vibratory Behavior of a Single Spool Gas Turbine Designed for Unmanned Aerial Vehicle Applications. Shock and Vibration, v. 2017, p. 1-13, 2017. Available in: <https://doi.org/10.1155/2017/4312943>

[6] Bastian M. and Li B. Q. An efficient automatic mesh generator for quadrilateral elements implemented using C++. Finite Elements in Analysis and Design, v. 39(9), p. 905-930, 2003. Available in: <https://doi.org/10.1016/S0168-874X(02)00138-5>

[7] Kane J. H. Boundary Element Analysis in Engineering Continuum Mechanics. New Jersey - Clarkson University: Prentice-Hall; p. 676, 1994. Available in: <https://books.google.com.br/books?isbn=0130869279>

[8] Brebia C. A. and Dominguez J. Boundary Elements: An Introductory Course. Southampton - Boston: Computational Mechanics Publications, p. 322, 1994. Available in: <https://books.google.com.br/books?isbn=1853123498>

[9] Tsuboi, H. et. al. Adaptive triangular mesh generation for boundary element method in three-dimensional electrostatic problems. IEEE Transactions On Magnetics, v. 34(5), p. 3379-3382, 1998. Available in: <https://doi.org/10.1109/20.717795>

[10] Phongthanapanich, S. and Dechaumphai, P. Adaptive triangulation with object oriented programming for crack propagation analysis. Finite Elements in Analysis and Design, v. 40(13-14), p. 1753-1771, 2004. Available in: <https://doi.org/10.1016/j.finel.2004.01.002>

[11] Tsuboi H. and Shimotsukasa T. Triangular mesh generation using knowledge base for three-dimensional boundary element method. IEEE Transactions on Magnetics, v. 26(2), p. 799-802, 1990. Available in: <https://doi.org/10.1109/20.106438>

[12] Du C. An algorithm for automatic Delaunay triangulation of arbitrary planar domains. Advances in Engineering Software, v. 27(1-2), p. 21-26, 1996. Available in: <https://doi.org/10.1016/0965-9978(96)00004-X>

[13] Lee K.-Y., Kim I.-I., Cho D.-Y. and Kim T.-w. An algorithm for automatic 2D quadrilateral mesh generation with line constraints. Computer-Aided Design, v. 35(12), p. 1055-1068, 2003. Available in: <https://doi.org/10.1016/S0010-4485(02)00145-8>

[14] Mavriplis D. J. An advancing front Delaunay triangulation algorithm designed for robustness. Journal of Computational Physics, v. 117(1), p. 90-101, 1995. Available in: <https://doi.org/10.1006/jcph.1995.1047>

[15] El-Hamalawi A. A 2D combined advancing front-Delaunay mesh generation scheme. Finite Elements in Analysis and

Design, v. 40(9-10), p. 967-989, 2004. Available in: <https://doi.org/10.1016/j.finel.2003.04.001>

[16] Bastian M. and Li B. Q. An efficient automatic mesh generator for quadrilateral elements implemented using C++. Finite Elements in Analysis and Design, v. 39(9), p. 905-930, 2003. Available in: <https://doi.org/10.1016/S0168-874X(02)00138-5>

[17] Ruppert J. A Delaunay refinement algorithm for quality 2-dimensional mesh generation. Journal of Algorithms, v. 18(3), p. 548-585, 1995. Available in: <https://doi.org/10.1006/jagm.1995.1021>

[18] Shewchuk J. R. Delaunay refinement algorithms for triangular mesh generation. Computational Geometry-Theory and Applications, v. 22(1-3), p. 21-74, 2002. Available in: <https://doi.org/10.1016/S0925-7721(01)00047-5>

[19] Secchi S. and Simoni L. An improved procedure for 2D unstructured Delaunay mesh generation. Advances in Engineering Software, v. 34(4), p. 217-234, 2003. Available in: <https://doi.org/10.1016/S0965-9978(02)00131-X>

[20] Foley J. D. et al. Computer Graphics: Principles and Practice in C. Addison-Wesley Publishing Company, 2nd. ed., p. 1179, 1996. Available in: https://books.google.com.br/books?isbn=0201848406