A three-dimensional unstructured mesh generation method for environmental reservoir simulation analysis

Hyun Ho Shin, **Norberto Mangiavacchi**,
Departamento de Engenharia Mecânica, UERJ,
20550-013, Rio de Janeiro, RJ, Brazil
E-mail: hhs82528@yahoo.com, norberto.mangiavacchi@gmail.com,

Christian E. Schaerer,
Facultad de Politécnica, UNA
Campus Universitario, SL, Paraguay
E-mail: cschaer@pol.una.py,

Cássio Botelho Pereira Soares
Departamento de Engenharia Ambiental, Furnas Centrais Elétricas, S.A.
22281-900, Rio de Janeiro, RJ, Brazil.
E-mail: cassiobp@furnas.com.br.

Abstract: *This work presents a three-dimensional unstructured mesh generator for the analysis of hydroelectric power plants reservoirs using finite element methods. In order to obtain an accurate simulation of the physical flow of interest, the discrete mesh needs to consider adequately the geophysical data employed for the definition of the domain, which usually comes from sources with different precision, type and structure. The proposed algorithm is practical, stable and able to deal with different types of geophysical input data producing well conditioned three-dimensional meshes.*

1 Introduction

The finite element method (FEM) has emerged as one of the most powerful numerical methods so far devised [2]. To use the FEM for analyzing and evaluating environmental impacts of exploitation of hydroelectrical reservoirs, it is necessary to obtain a discrete mesh of the reservoir.

In [6], it is showed that the Delaunay triangulation is the most practical algorithm for a surface representation, providing a continuous surface of unique triangles. In addition, the resulting mesh is independent of the ordering and the dispersion of the points. In the Delaunay triangulation, the results are repeatable and predictable, while other methods depend on the starting point [6]. However, when a three-dimensional mesh with a huge relationship between the average of horizontal width and depth of the reservoir contained points is considered, the Delaunay incremental algorithm is unstable.

To take advantage of the 2D Delaunay triangulations properties, we consider the generation of a triangular surface mesh of the points data using a randomized incremental algorithm with a quickly search for a point location. Thereafter, a 3D mesh is obtained dividing each prism in several specified set of tetrahedra. This yields a stable and unstructured generator which provides a practical conforming tetrahedral mesh with low computational cost.

2 Delaunay triangulation

Consider a given set of points in the plane, a Delaunay construction yields a unique triangulation of these points [6] since it maximizes the smallest angle over all triangulations [1, 5]. This property is equivalent to that the circumscribed circle (denoted as circumcircle) of any triangle of the Delaunay triangulation does not

contain any other point in its interior [1].

Given an initial triangulation, a new point may be inserted into the triangulation by first locating and deleting all existing triangles whose circumcircles contain the newly inserted point (Figure 1). A new triangulation is then formed by joining the new point to all boundary edges of the cavity created by the previous removal of intersected triangles (see Figure 2). This process is known as the incremental algorithm of node insertion [3].

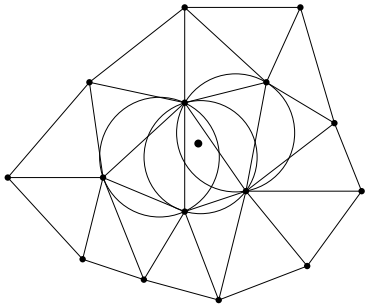

Figure 1: Delaunay Triangulation. Given an initial triangulation, the circumcircles containing a new point are showed.

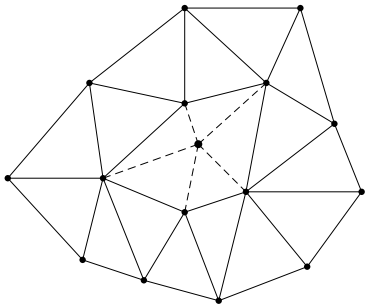

Figure 2: Delaunay Triangulation. A new triangulation is showed.

The pseudocode 1 shows the node insertion algorithm. Notice that the function $Incircle(\triangle_i, p)$ in line 2 is a function that verifies if the circumcircle of the triangle $\triangle_i$ contains the point p. Actually, to verify if the point p lies in the circumcircle of the triangle $\triangle_i$ is not trivial, since that round-off error induces instabilities. In order to verify this condition in a stable way, we used the algorithm represented in the Pseudocode 2. If the function $Incircle(\triangle_i, p)$ returns *true* then the edges of triangle $\triangle_i$ are saved in the list of edges $\mathcal{E}$ (line 3) and therefore the triangle is deleted from the list of triangles $\mathcal{T}$ (line 4).

The process to join the new point to all

Pseudocode 1: Node insertion algorithm.

Algorithm: NODEINSERTION($\mathcal{T}, p$)
Input: A Delaunay Triangulation mesh $\mathcal{T}$ and a point to insert p.
Output: A new Delaunay Triangulation mesh $\mathcal{T}$.
1. **for** each triangle $\triangle_i$ of $\mathcal{T}$
2. **if** $Incircle(\triangle_i, p)$ is true
3. save the edges of $\triangle_i$ in the list of edges $\mathcal{E}$.
4. delete $\triangle_i$ from $\mathcal{T}$.
5. identify the nonrepeated edges in $\mathcal{E}$.
6. **for** each edge identified
7. save the triangle created by the edge and the point p in $\mathcal{T}$.
8. **return** $\mathcal{T}$.

boundary edges of the cavity is represented in lines 5 to 7. Notice that (see Figure 1) the edges that form the boundary of the created cavity are saved just once because they belong to only one deleted triangle, while those which do not form this boundary are saved twice because they belong to two deleted triangles. Hence, the edges that form the boundary of the cavity are not duplicated in the list of edges $\mathcal{E}$. Once identified the non-repeated edges (line 5), the points that belong to the ends of the edges are joined forming new triangles. These new triangles are added to the list of triangle $\mathcal{T}$ (line 6 and 7).

2.1 The function $Incircle(\triangle_i, p)$

Next, we discuss the details of the function $Incircle(\triangle_i, p)$. The algorithm is shown at the Pseudocode 2. We consider the triangle $\triangle rst$ with vertices r, s and t. The points r, s, t and p have cartesian coordinates denoted by $r(r_x, r_y)$, $s(s_x, s_y)$, $t(t_x, t_y)$ and $p(p_x, p_y)$, respectively.

The projection of the points r, s, t and p onto the paraboloid defined by

$$z = x^2 + y^2 \tag{1}$$

are denoted by r', s', t' and p', respectively. Therefore, the cartesian coordinates of these projections are:

$$r'(r_x, r_y, r_x^2 + r_y^2), \tag{2}$$

Pseudocode 2: Algorithm to verify if a point lies in the circumcircle of any triangle.

Algorithm: INCIRCLE($\triangle rst, p$)
Input: The triangle $\triangle rst$ and the point t.
Output: *true* if s lies within the circumcircle of $\triangle rst$, else *false*.
1. project the points r, s, t and p onto the paraboloid $z = x^2 + y^2$
2. find the plane (α) passing through r', s' and t'
3. **if** p' lies on the lower side of the plane(α)
4. **return** *true*
5. **else**
6. **return** *false*

Pseudocode 3: Delaunay Triangulation.

Algorithm:DELAUNAYTRIANGULATION($\mathcal{P}$)
Input: A set $\mathcal{P}$ of n points in the plane.
Output: A Delaunay Triangulation $\mathcal{T}$ of $\mathcal{P}$.
1. Let p_{-1}, p_{-2} and p_{-3} be the points such that $\mathcal{P}$ is contained in the triangle $\triangle p_{-1}p_{-2}p_{-3}$.
2. Initialize $\mathcal{T}$ with the single triangle $\triangle p_{-1}p_{-2}p_{-3}$.
3. **for** $i \leftarrow 1$ **to** n
4. $NodeInsertion(\mathcal{T}, p_i)$
5. Discard p_{-1}, p_{-2} and p_{-3} with all their incident edges from $\mathcal{T}$.
6. **return** $\mathcal{T}$.

$$s'(s_x, s_y, s_x^2 + s_y^2), \quad (3)$$

$$t'(t_x, s_y, t_x^2 + t_y^2), \quad (4)$$

$$p'(p_x, p_y, p_x^2 + p_y^2). \quad (5)$$

The algorithm is based on the following statement demonstrated in [4]:

Proposition. *Let consider four distinct points r, s, t, p in the plane, and let r', s', t', p' be their respective projections onto the paraboloid defined by* (1)*. The point p lies within the circumcircle of r, s, t if and only if p' lies on the lower side of the plane passing through r', s' and t'.*

2.2 The Delaunay triangulation

We describe the Delaunay triangulation of a set $\mathcal{P} = \{p_1, p_2, \ldots, p_n\}$ of n points in the plane. The incremental point-insertion algorithm is used as a mesh generation strategy [3] (see Pseudocode 3).

A drawback for the Delaunay triangulation is its high computational cost as a new point is inserted. Notice that for each inserted point p, it is necessary to evaluate the function $Incircle(\triangle_i, p)$ in all triangles of the triangulation $\mathcal{T}$. Hence, the searching algorithm is a bottleneck in this process.

To reduce the computational cost, the function *QuicklySearch* is used into the function *NodeInsertion*. The advantage of the *QuicklySearch* consists in using an auxiliary structured quadrangular mesh associated to a list matrix of integers. This matrix stores all the information necessary to identify the triangles which are contained in each element of the quadrangular mesh.

In this way, when a new point is inserted, the function *QuicklySearch* identifies in which quadrangular element the new point lies. Then it evaluates the function *Incircle* only for the triangles contained into the quadrangular element. This strategy saves computational time. Once, the function *QuicklySearch* identifies the triangle which contains the point to be inserted, the function *NodeInsertion* evaluates all the possible circumcicle that contains the point. A disadvantage of the function *QuicklySearch* is the necessity to save the information of the neighbored triangles. However, this information can be used to easily identify the nodes at the mesh boundary. This is important in order to impose the partial differential equation boundary conditions in the FEM routine.

3 Reservoir 3D Mesh Generation Routine

The mesh generator is used in the context of an object oriented simulation software. There are two features that are desirable and usually found in cartesian unstructured reservoir meshes:

Layers. Node points are connected to other points at the same z coordinate. This feature is desirable, in special, in strongly

stratified flows since the velocity is preferably horizontal, and a mesh constructed with horizontal layers will introduce less artificial diffusion.

Sticks. Node points are connected vertically to other node points at the same x, y coordinate, but at a different z coordinate. This feature is desirable because it propitiates a simplification of the pressure calculation, which varies almost hydrostatically in the vertical direction. In the limit of very low vertical accelerations, the pressure distribution can be assumed hydrostatic and pressure can be computed from the surface pressure, integrating vertically the z-momentum equation.

Even if the pressure distribution is not assumed to be hydrostatic, the vertical alignment in the sticks produces a better coupling between the vertical velocity and the pressure fields.

The Pseudocode 4 outlines the proposed mesh generation method. The routine employs as input data the coordinate points of the contour lines corresponding to the bottom of the reservoir. However, these points are not sufficient to generate a tetrahedra mesh. Therefore additional points need to be defined inside the reservoir. Next, we discuss the mechanism of inclusion of these additional points (Pseudocode 4, step 2).

Pseudocode 4: Three-Dimensional Mesh Generation Routine.

Algorithm: 3D Mesh Generation
Input: A set $\mathcal{P}$ of n points of the contour lines.
Output: A Mesh of Tetrahedra (List of Tetrahedron $\mathcal{T}$).
1. Delaunay triangulation of the points data.
2. Generation of the points inside the reservoir.
3. Dividing the prisms producing the tetrahedral elements.
4. **return** $\mathcal{T}$.

3.1 Points adding process

Throughout each point that does not belong to the upper contour lines, a vertical line (stick) is drawn. This vertical lines intersect the horizontal planes (layers). The intersections lying inside the reservoir are the points to be added to the set of points $\mathcal{P}$. Notice that now the points are classified in layers. The Figure 3 shows an example of the contour lines points, while the Figure 4 shows the points after the point adding process.

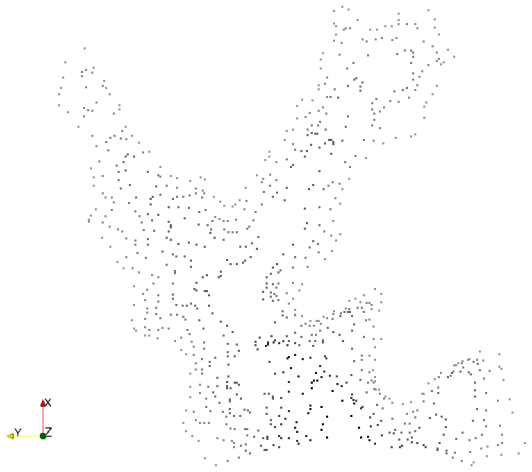

Figure 3: Before adding points (Points of the contour lines).

Figure 4: After adding points.

3.2 Prism partition process

After the point adding process, the code generates a three-dimensional mesh using the Delaunay triangulation with the information of the sticks and layers. Once the Delaunay triangulation was performed, the triangular prisms are constructed and partitioned in order to produce the tetrahedral elements.

The process of dividing the prisms is not trivial since it is possible to obtain a non-conforming tetrahedron (see Figure 5). Therefore we impose restrictions during the prism partitioning process.

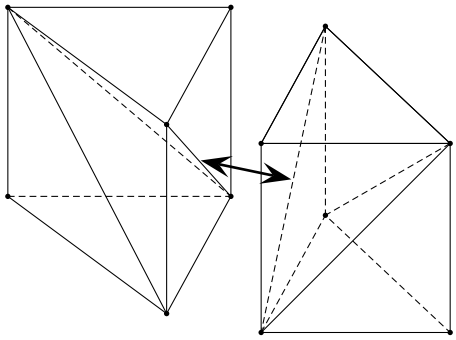

Figure 5: Partitioning prisms: Non-conforming tetrahedron.

To solve this problem, we orientate each edges (using arrows) considering the crescent x-coordinate. Using the origin vertex of the arrow as a starting point, we draw diagonals to each lateral face (see Figure 6). Notice that, since the orientation of the edges are arbitrate, the partition of the prism is not unique.

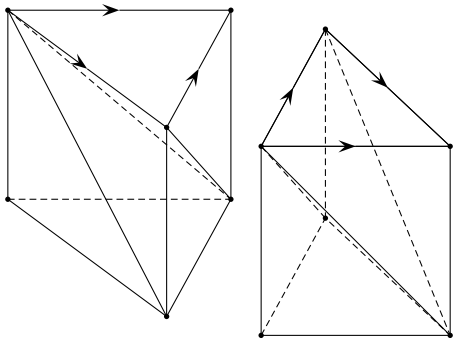

Figure 6: Partitioning prisms: Conforming tetrahedron.

The complete process of mesh generation presented at the Pseudocode 4 is shown at Figure 7. Notice the result in wireframe representation when four layers are considered. It can be observed that the final mesh has the desired feature: the points lie on the intersection of planes (layers) with vertical lines (sticks).

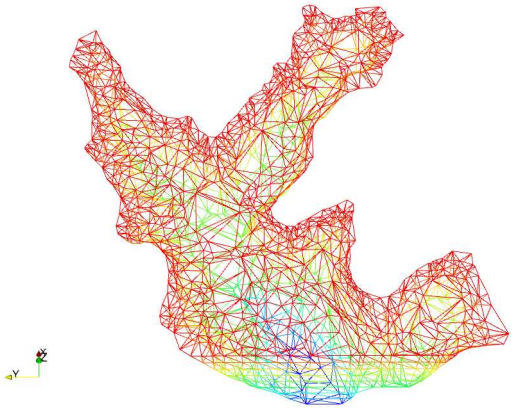

Figure 7: Three-dimensional Mesh (Wireframe representation).

4 CONCLUSION

We implemented the proposed algorithm in C++ in the context of a project for the analysis and evaluation of environmental impacts of hydroelectrical reservoirs. The algorithm is able to deal with several kinds of geophysical data, as well as, data with a huge relationship between the average of horizontal width and depth, which is a common situation in reservoir engineering. The mechanism for constructing the three dimensional mesh warranties to obtain conforming tetrahedra. In addition, the algorithm allows to have several criteria for the partitioning of the prism.

An advantage of the algorithm is the point insertion routine. This allows to easily implement a refinement of the mesh in order to increase the approximation of the simulation. In this sense, an a priori error estimation criterion can be used for the refinement of the grid.

References

[1] M. Berg, “Computational Geometry: Algorithm and Applications”, Springer-Verlag Berlin Heidelberg New York, Second Revised Edition, 2000.

[2] J. Donea and A. Huerta, “Finite Element Methods for Flow Problems”, J. Wiley, West Sussex, 2003.

[3] D. J. Mavriplis, “Unstructured Grid Techniques, *Annual Review Fluid Mechanics*, 29 (1997) 473-514

[4] D. M. Mount, “Computational Geometry, Lecture Notes, Department of Computer Science, University of Maryland, 2002.

[5] J. O’Rourke, “Commputational Geometry in C”, Cambridge University Press, Massachusetts, Second Edition, 1997.

[6] D. Shojaee, H. Helali and AA. Alesheikh, Triangulation for Surface Modelling, in “Ninth International Symposium on the 3D Analysis of Human Movement”, France, 2006.