Multi-Resolution Triangulations with Adaptation to the Domain Based on Physical Compression

RICARDO MARROQUIM[1], PAULO ROMA CAVALCANTI[1], LUIZ VELHO[2], CLAUDIO ESPERANÇA[1]

[1] UFRJ - Universidade Federal do Rio de Janeiro
[2] IMPA - Instituto de Matemática Pura e Aplicada
{ricardo,roma,esperanc}@lcg.ufrj.br
lvelho@visgraf.impa.br

Abstract. This paper presents a method for generating multi-resolution triangulations of non-manifold objects composed of several regions with arbitrary geometry. The process of adapting the triangulation to the boundary of the object is based on physical compression, more specifically, a mass-spring system. The final triangulation usually has no degenerated triangles and provides an approximation of the boundary based on a chosen resolution.

1 Introduction

Numerical simulations have become an essential step in the development of engineering products, or in the prediction of the behavior of physical phenomena, such as weather conditions, oil generation, tide movement, earthquakes, etc.

In order to be able to carry out any simulation it is necessary to discretize the domain of interest, onto which a set of equations, describing physical laws, must be solved. This discretization process is generically referred to as the triangulation of the domain.

Although several triangulation algorithms have been proposed in the past three decades, these algorithms are meant mainly to deal with mechanical parts, produced by CAD systems. However, there are numerous important applications where the domain presents no symmetry at all, and boundaries are not well or clearly established.

Geological models, for instance, are often composed of several regions (forming the geological layers), and possess an irregular geometry, where surfaces can meet at very small angles, a situation known as a "pinch-out". The geometry of the surfaces (horizons and faults) are obtained from seismic data, by a process which is not automatic and demands the interpretation of a geologist. As a consequence, the whole process of creating the model is not exact, and tends to be error prone.

Traditional triangulation algorithms can be classified according to the way they work. Delaunay based algorithms recover the boundaries after triangulating the convex hull of the pointset [2]. Ruppert and Shewchuck [10, 11, 12] insert a vertex at the circumcenter of a bad element to improve the mesh quality. In practice, if the original model already contains small angles, their algorithms have problems in converging, specially in 3D. Advancing front algorithms [8], on the other hand, start at the boundaries and proceed toward the center of the model, inserting new vertices to create good shaped elements. The trick is to merge the different fronts without generating any inconsistency. The success of the algorithm is highly dependent on the simplicity of the geometry of the boundary. Packing sphere algorithms [1, 7, 6] generate a distribution of vertices, forcing edges and faces to be present in the final mesh, which can be then generated, by a Delaunay like algorithm.

These traditional triangulation algorithms are hard to implement, have difficulty to deal with models having complex and irregular boundaries, and do not support multi-resolution. Even when a triangulation algorithm does a good job meshing this kind of model, it may generate small angles in the final mesh, which usually cause problems when running numerical simulations. Furthermore, if the original model already contains small angles these methods cannot eliminate them.

This paper presents an adaptative algorithm to triangulate non-manifold, multi-region models, which approximate their irregular boundaries instead of trying to match them exactly. Our main concern is to generate a good mesh, although the boundary is just an approximation, as close as possible to the original one.

Physically based triangulation algorithms are not new. Molino [9] has used a similar approach to obtain numerical meshes. Nonetheless, despite his good results, he considered only single region manifold objects. In this paper we extend Molino's work to be able to deal with non-manifold multi-region objects, which are necessary to address a broader set of applications.

The algorithm has two main steps. In the first step, a grid is created over the object, adptatively, in order to subdivide the surrounding space. Some techniques to efficiently create the subdivision are also described in section 2. The representation of the input data is explained in section 3, and Section 4 describes the distance function used to speed up the geometrical search in the algorithm.

The second step, described in section 6, conforms the vertices of the grid to the boundary of the object through a compression scheme, while trying to maintain good shaped

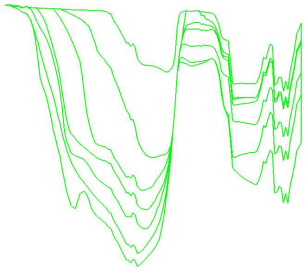

Figure 1: 2D slice taken from a 3D Gulf of Mexico model.

elements.

Although the main goal is to be able to triangulate 3D models, in this paper we focus only 2D models, sometimes obtained by slicing a 3D model (Figure 1). The input data is then a set of polygonal curves, which describe the boundaries (possibly disconnected) of 2D multi-region objects.

2 Spatial Subdivision

The first step of the algorithm is to subdivide the surrounding space of the model. An adaptive 4-8 mesh is used to hold the subdivision.

A 4-8 mesh is a *triangulated quadrangulation* [16, 13, 14, 15], which means it combines structural properties of triangular and quadrilateral meshes. The grid obtained from the subdivision is also the base simplicial complex for the triangulation.

The criterion used for subdividing the space is the distance from an edge to the boundary, that is, the grid will be more densely refined at the boundary neighborhood. 4-8 adaptive meshes support multi-resolution naturally, and also have the property of producing a gradual transition of resolution, therefore leaving no gap between levels in the subdivision process.

To subdivide a particular edge, the following conditions must hold:

- the maximum subdivision level has not been reached;
- the distance from its midpoint to the boundary is greater than a given tolerance, and
- its length is greater than the distance of its midpoint to the boundary.

The maximum subdivision level and the tolerance are set by the user, and have a great influence on the number of created triangles (Figure 2).

All triangles in a 4-8 mesh have angles equal to $\pi/4$ or $\pi/2$. This is an important property since there will be no small angle. The goal is to preserve these angles wherever possible.

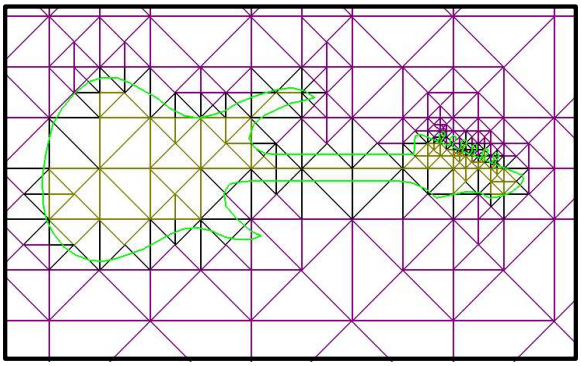

Figure 3: Subdivision grid over the guitar model.

2.1 Local Subdivision

The maximum level set by the user has a global influence in the final mesh, but usually some regions will be more refined than others. A low maximum subdivision level will approximate a complex feature on the boundary poorly. A high maximum level will generate too many triangles for simple regions. In order to improve the subdivision algorithm a local subdivision process is performed.

The local subdivision looks for triangles containing more than one point and subdivides them. If there is a triangle with many points inside it, the approximation in that area will be poor. Triangles crossed by more than one segment are also further subdivided.

There are then two maximum levels: one for the entire mesh and a higher one for the local subdivision process. A low global maximum level followed by a local subdivision produces the best results. In Figure 3 it is shown a guitar triangulated using the local subdivision process. Note that there are more triangles near the knobs than around the body.

3 Polygon Representation

The set of polygons creates a partition of the space. A two-dimensional cell is called a *region*. The algorithm requires a good distance function, to speed up the classification of points according to this partition.

A polygon is represented by two structures: a vertex list and a binary tree containing its segments (one tree for each region). Each node of the tree contains one or more adjacent segments of the curve, and holds a pointer to the first and last vertices. The root node, for example, covers the whole polygon. Hence, it has pointers to the first and last vertices of the curve. If the curve has no boundary, the root node has two pointers to the same vertex. Child nodes contain the two halves of the parent node.

For a polygon with eight vertices, for example, the root node has pointers to vertices 1 and 8, its left child to vertices

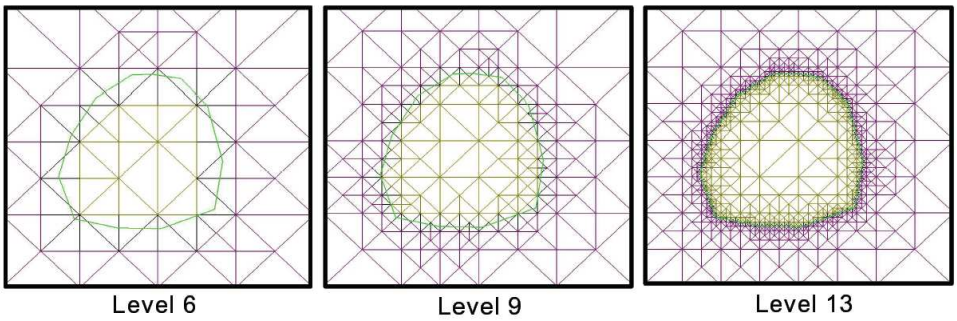

Figure 2: Polygonal curve subdivided with 3 different levels. The level defines the maximum number of times an edge can be subdivided.

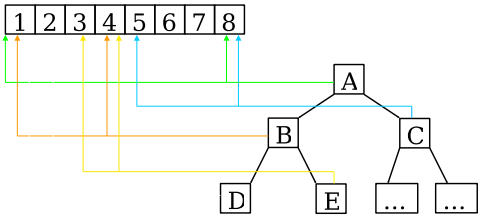

Figure 4: Binary tree representing a polygonal curve with eight vertices.

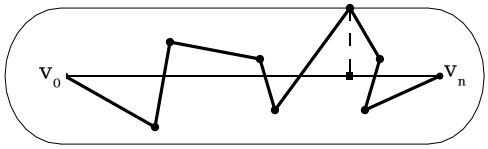

Figure 5: The bounding region of a node defined by the maximum internal distance (dashed black line). The line with endpoints v_0 and v_n is the central segment.

1 and 4, and its right child to vertices 5 and 8. Going down the tree, the segments are further split in two halves, until there is only one segment, as shown in Figure 4. The union of all nodes in a given level produces the whole polygonal curve.

To optimize the distance function, the *maximum distance* inside a node is also computed and stored. This value is the maximum distance from all vertices in a node to the segment defined by the first and last vertices of the node. This particular segment is called the *central segment* of a node.

4 Distance Function

A distance function is needed throughout the execution of the algorithm. The signed distance function we chose is based on [4, 5], and uses the binary tree of the polygon representation (section 3).

To compute the smallest distance from a point to a polygonal curve the binary tree is traversed. The *maximum distance* stored in each node defines a bounding region for it (Figure 5). An auxiliary heap is also created to keep track of visited nodes.

The sorting criterion for the heap is the difference between the node maximum internal distance, and the distance from the given point to the node's central segment, i.e., the distance from the point to the node's bounding region.

When a node is visited, it is removed from the heap. Then, the criterion distance for sorting is computed for its child nodes, which are inserted in the heap.

The algorithm will repeatedly visit the node with the smallest distance (the node at the front of the heap) until it finds a node with no child (a leaf node contains only one segment). So, when the stop criterion is met, it means that the exact distance to the closest segment has been found. The reason is that the distance from a point, p, to the node's polygonal line, L, will be not smaller than the distance from p, to the node's bounding region, B:

$$\delta(p, B) \leq \delta(p, L).$$

To perform a point in region testing, a signed distance function is necessary. Using the closest segment to the point, and a few cross products, it is possible to determine the sign of the distance based on the boundary circulation. The sign is positive when the point is inside the region, and negative otherwise.

When dealing with multi-region objects, the distance of a point is taken considering each region separately. Regions with holes are possible, since the signed distance can be used to determine the inclusion of a region into another. Points inside a hole are considered outside the model (just one level of holes).

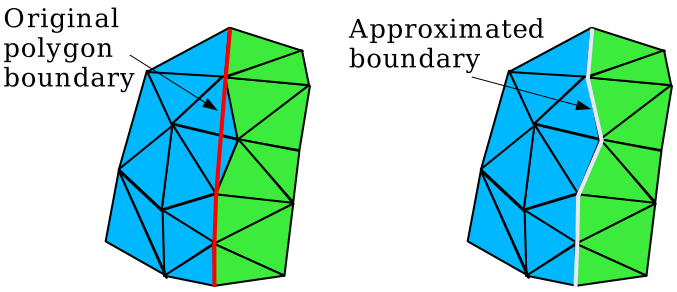

Figure 6: Final boundary after face classification. Edges between faces with different tags defines the approximated boundary.

5 Mesh classification

An essential step to obtain the final triangulation is to classify the initial mesh elements, using the distance function. A vertex may be classified in three different ways: (1) being inside the unlimited external region, (2) onto the boundary of a region, (3) or inside a region. In the last case, the vertex saves the region it is into as an attribute.

Edges are classified in a similar way. There are exterior edges, intersection edges and interior edges. The region the edge is into is also stored.

Faces are only classified after the completion of the compression stage, as described in section 6. Faces must belong to one polygon region or to none (exterior faces). Those crossing a boundary are classified as belonging to the region containing its barycenter. After all faces have been properly classified, the edges are reclassified. Edges whose two adjacent faces have different tags are classified as boundary edges, therefore defining the final approximated boundary, see Figure 6. All other edges are classified as either interior or exterior.

6 Mesh Compression

After classifying the grid elements, the simplicial complex has to adapt to the boundary. As a matter of fact, grids refined densely will produce better approximations.

The goal is to push some of the vertices near the boundary towards it, as shown in Figure 7. A simple projection would result in many collapsed triangles, and consequently small angles. To maintain good shaped triangles, there are springs associated to each triangle, forming a mass-spring physical system.

The mesh can be thought of as so many connected springs. When the outer springs are pushed inward, the interior ones distribute the compression forces thus rearranging themselves.

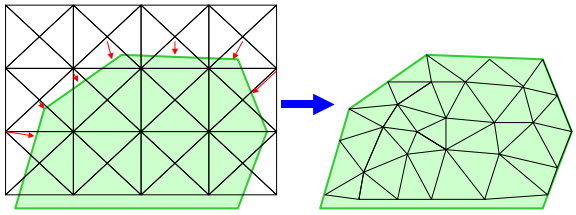

Figure 7: Compressing the grid inside the polygon boundarys.

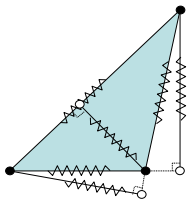

Figure 8: Projection springs: springs between black and white vertices. White represents auxiliary vertices, and black real vertices.

6.1 Spring Configurations

The springs may be placed in different configurations, e.g., one spring for each edge. Several configurations have been tested, and the code is implemented so as to allow new configurations to be added easily.

Triangles cannot collapse during the compression, as this would cause zero or negative areas. The configuration that worked best in our tests uses one spring per edge plus one spring per projection edge, as shown in Figure 8. This configuration proved to prevent collapses efficiently [3].

There are two global coefficients: spring elasticity and damping constant. The only information that needs to be stored per spring is its rest length, which is stored as an edge attribute. For some configurations there may be more than one spring per edge. The extra springs are called auxiliary springs. Some auxiliary vertices are also created by these springs, i.e., non-real vertices. In Figure 8 the auxiliary vertices are drawn in white.

To compute spring forces there are two different equations: one for edge springs and the other for repulsion springs (projection edges). The edge spring equation is the Hooke's Law:

$$F = k_s * (|\triangle x| - x_o) + k_d \left(\frac{\triangle v \cdot \triangle x}{|x|} \right) * \frac{\triangle x}{|\triangle x|}$$

Where ks and kd are the elasticity and damping constant, x_0 is the rest length and $\triangle x$ and $\triangle v$ are the vertices position and velocity differences.

For the repulsion spring there is a non-linear equation, as explained in [3]. These non-linear springs apply an in-

finity force when their lengths go to zero.

$$F = k_s * \left(|\triangle x| - \frac{x_o}{|\triangle x|} \right) * \frac{\triangle x}{|\triangle x|}$$

6.2 Particle System

To simulate the mass-spring oscillatory movement, the mesh is treated as a particle system [17]. Each vertex represents a particle in the system, and particles are associated to spring endpoints. Some temporary particles may be introduced to represent auxiliary vertices. These particles only exist when computing the force associated to auxiliary springs, having only local influence in the system.

Each particle has the following attributes: position, velocity, force, and mass. The system iterates in time steps, updating at each cycle, the particles' position and velocity. At each time step there is an iteration through all edges to compute the forces at their particles. The force on each vertex is the sum of the forces applied by all incident springs.

The duration of one step can be redefined by the user. Usually, small steps are better because they create less oscillatory movement on the springs. However, smaller time steps result in more iterations, and consequently longer simulation running times.

Each system iteration has three main phases:

1. computation of the force accumulators for each vertex;
2. calculation of derivative values;
3. computation of new positions and velocities.

6.2.1 Force Accumulators

The force accumulated in each vertex is the sum of all contributions from all incident springs. Auxiliary vertices are not considered, but they contribute to the force accumulated on the other endpoint of the spring, if the endpoint is a real vertex.

The first step is to reset all force accumulators, with a simple iteration through all particles. Next, the edges are examined in order to compute the forces applied on their vertices. If there is any auxiliary spring associated to the current edge, the forces on its endpoints are also computed.

6.2.2 Derivative Values

In this step, each particles' position and velocity derivatives are calculated, i.e., its velocity and acceleration, respectively. Here, an auxiliary array is created to hold all of the values. The vector size is $4n$, where n is the number of particles in the system. The derivative vector is scaled for the time of one iteration step.

Another array is used to hold the particles' current position and velocity. The vector size is the same of the derivative vector. To compute the new attribute values the two vectors are added, and the particles are updated with the resulting values.

6.3 Compression Criteria

The springs rest lengths are set to their initial length, meaning that initially the particle system is completely stable. Unless some disturbance is applied on the system, it will not change. To start the compression some vertices are marked to be pushed towards the boundary.

There are many different ways to compress the mesh, and the first question is which particles should be pushed. We implemented different rules for choosing these particles, form the candidate set of all endpoints of edges that intersect the boundary.

The first criterion is to push all vertices in the candidate set. This is actually not a good choice because pushing many vertices will cause many collapses. Actually, in all criteria, triangles may collapse and have to be treated at a later stage in the simulation. However, this rule collapses more triangles than the other rules.

The second criterion is to push all vertices lying at one side of the boundary. If the boundary is a frontier between a region and the exterior, all exterior vertices are pushed. If it is a frontier between two regions, all the vertices in one of the regions are pushed. This criterion collapses few triangles because all vertices are pushed in the same direction.

The third criterion is to push the vertices closest to the boundary. By choosing some vertices in each side of the boundary there is a great chance of squishing the triangles, but not as many as with the first criterion. The advantage of this method is that there is less particle movement, preventing too much oscillation.

Tests show that the second criterion usually behaves better than the others. In all of them, the vertices are pushed in an orthogonal direction to the boundary, with a constant velocity.

The particles associated to marked vertices, i.e., the ones being pushed, will behave differently from the others in the system. In particular, they will not suffer any influence from the other particles. After the initial push, they will not change course or speed until they reach the boundary, or until they cannot go further, without collapsing a triangle. If these vertices were allowed to interact freely with the system after the push, the system would eventually return to its initial state.

Since these vertices are not treated in a physically sound way, it cannot be guaranteed that the springs will prevent the triangles from collapsing. It is necessary to do some simple tests to ensure that this will not happen. A check is made to find out if a particle movement will cross one of its adjacent springs in each step. For each particle-spring

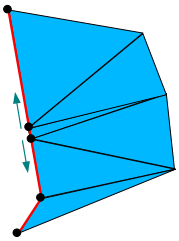

Figure 9: Scattering the particles that landed too close.

pair, two cross products are compared. If the test predicts a collapse, the particle movement is restrained until the next time step.

6.4 Scattering Boundary Vertices

After all particles have reached the boundary, or have gone as far as they can go without collapsing any triangle, they are allowed to interact with the rest of the system. Many particles will land close together generating small angles in the triangulation. To spread them apart their restraint is loosened, but not completely. Their movement must remain limited to the boundary. We might think of it as if particles were running on a rail while the edges between them generate the necessary repulsion forces to spread them apart.

In order to obtain a perfect match of the original boundary it is necessary to have one mesh vertex placed at each polygon point. For this reason, a vertex is restrained from further movement when it passes over a boundary point with high curvature. Therefore, boundary points almost collinear are not fixed.

6.5 Handling Degeneration

When pushing the vertices towards the boundary, and even while scattering, some triangles might collapse, or get stuck with a very small area. To solve these problems, the longest edge of these triangles are flipped. This check is done every few time steps so that after flipping the edges, the vertices may continue to move along the boundary. This procedure continues until no small angles are found, or the system stabilizes. However, flips must be only applied at the end of the process, for not losing the multi-resolution scheme.

6.6 Aliasing

Aliasing occurs around areas in which there are two or more very narrow adjacent regions. The mesh resolution might not be high enough to place a triangle inside just a single region. After compression, triangles may cross a region without having any vertex inside, or clipped against its boundary. In Figure 10, it is shown an example of aliasing. The triangles in the middle might belong to any of the three

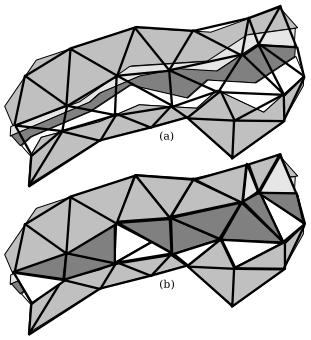

Figure 10: (a) Mesh resolution is not high enough to place triangles inside all thin regions causing aliasing. (b) The same mesh after face classification.

thin regions, by using the barycenter rule. Some regions are split in several smaller regions, and the topology cannot be always preserved.

As a solution, the mesh can be further refined around these regions until the triangles are small enough to fit inside them. However, near "pinch-outs" or very thin regions, this process may be unfeasible, because of the number of triangles required.

7 Results and Conclusions

The Lake Superior, Figure 11, is triangulated with a bound of 10^o, and has 1906 triangles. The smallest angle is 11.5^o, and 87.4% of the angles are in the range $[30^o, 60^o]$. The magnified detail of the model depicts that the boundary is slightly different, but well approximated, as can be seen in Figure 12.

Another example is the salt dome (Figure 13). It has 6 different regions with 6183 triangles. The smallest angle is 10.4^o, and 89.4% of the angles are in the range $[30^o, 60^o]$.

In Figure 14 it is shown the triangulation of a 2D slice of the Gulf of Mexico model. This slice is composed of 15 different regions, and possesses 11015 triangles. The smallest angle is 5.25^o, and 75% of the angles are in the range $[30^o, 60^o]$. Some regions of the mesh are shown in detail in Figure 15. Aliasing occurs near the "pinch out", and, as a consequence, the boundary is not well approximated, as can be seen in Figure 15(b).

We have presented an algorithm for triangulating multi-region, non-manifold 2D models. Although the boundaries of the triangulation may differ from the boundary of the original model, there are several applications where this is acceptable.

Our algorithm has many advantages over traditional triangulation techniques, such as: (1) being easy to implement, and offering, at the same time, multi-resolution; (2) being capable of dealing with models possessing complex

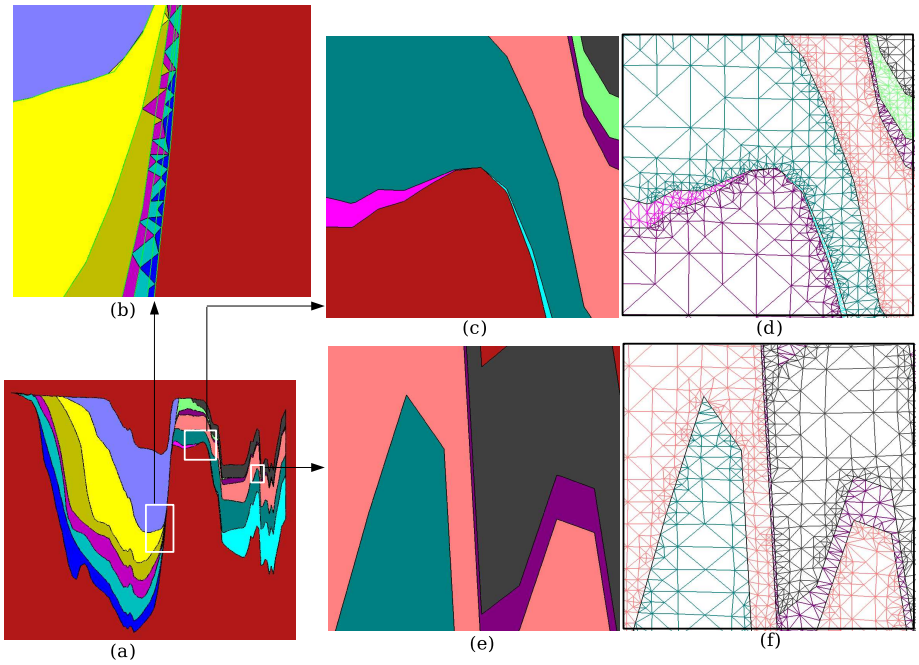

Figure 15: (a) mesh with painted regions. (b) occurence of aliasing where many regions meet. (c) and (e) detailed regions with an approximation near of a perfect match. (d) and (f) the same regions but triangulated.

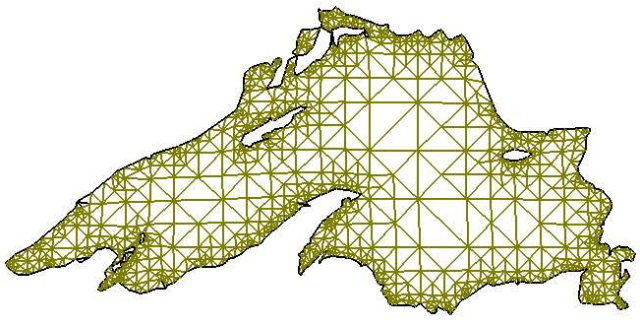

Figure 11: Lake Superior.

Figure 12: Detail of the Lake Superior. Approximated boundary of the small islands. Original boundaries are the green and blue outlines.

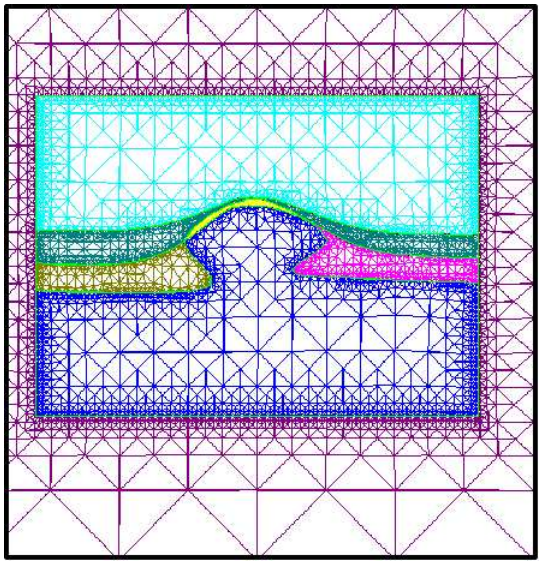

Figure 13: Salt dome model.

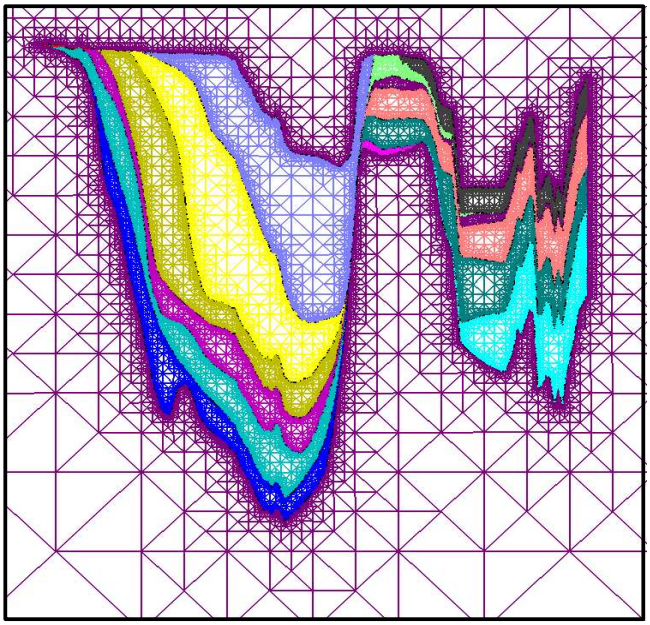

Figure 14: Cross section of the Gulf of Mexico.

and irregular boundaries; and (3) being not affected in a high degree by numerical issues.

The algorithm is also easily extendable to 3D, although new data structures are necessary in this case.

8 Acknowledgements

We would like to thank the CAPES and CNPq agencies and, specially, the financial support received from Cenpes/Petrobras, through the COPPETEC project Pesc 3678.

References

[1] Marshall Bern and David Eppstein. Quadrilateral meshing by circle packing. *International Journal of Computational Geometry and Applications*, 10(4), August 2000.

[2] Paulo Cavalcanti and Ullisses Mello. Three-dimensional constrained Delaunay triangulation: A minimalist approach. In *8th International Meshing Roundtable*, pages 119–129, Lake Tahoe, California, October 1999. Sandia National Laboratories.

[3] Lee Cooper and Steve Maddock. Preventing collapse within mass-spring-damper models of deformable objects. In *The Fifth Internation Conference in Central Europe on Computer Graphics and Visualization*, pages 70–78, Plzen, Czech Republic, February 1997.

[4] Luiz Henrique de Figueiredo, Jorge Stolfi, and Luiz Velho. Approximating parametric curves with strip trees using affine arithmetic. *Computer Graphics Forum*, 22(2):171–179, 2003.

[5] André Guéziec. Meshsweeper: Dynamic point-to-polygonal-mesh distance and applications. *IEEE Transactions on Visualization and Computer Graphics*, 7(1):47–61, 2001.

[6] Xiang-Yang Li, Shang-Hua Teng, and Alper Üngör. Biting: Advancing front meets sphere packing. In *2nd Symposium on Trends in Unstructured Mesh Generation*, University of Colorado, Boulder, August 1999.

[7] Xiang-Yang Li, Shang-Hua Teng, and Alper Üngör. Biting spheres in 3d. In *Proc. 8th Int. Meshing Roundtable*, pages 85–95, South Lake Tahoe, CA, October 1999.

[8] Dimitri J. Mavriplis. An advancing front Delaunay triangulation algorithm designed for robustness. 117(1):90–101, 1995.

[9] Neil Molino, Robert Bridson, Joseph Teran, and Ronald Fedkiw. A crystalline, red green strategy for meshing higly deformable objects with tetrahedra. In *12th Int. Meshing Roundtable*, pages 103–114, Santa Fe, New Mexico, September 2003.

[10] Jim Ruppert. *Results on Triangulation and High Quality Mesh Generation*. PhD thesis, University of California at Berkeley, 1992.

[11] Jim Ruppert. A Delaunay refinement algorithm for quality 2-dimensional mesh generation. *J. Algorithms*, 18(3):548–585, 1995.

[12] Jonathan Richard Shewchuk. *Delaunay Refinement Mesh Generation*. PhD thesis, Department of Computer Science, Carnegie Mellon University, 1997.

[13] Luiz Velho. Using semi-regular 4-8 meshes for subdivision surfaces. *Journal of Graphics Tools*, 5(3):35–47, 2001.

[14] Luiz Velho. Dynamic adaptive meshes. In *Dagstuhl Seminar on Hierarchical Methods in Computer Graphics.*, 2003.

[15] Luiz Velho. A dynamic adaptive mesh libraby based on stellar operators. *Journal of Graphics Tools*, to appear.

[16] Luiz Velho and Denis Zorin. 4-8 subdivision. *Computer-Aided Geometric Design*, 18(5):397–427, 2001. Special Issue on Subdivision Techniques.

[17] Andrew Witkin. An introduction to physically based modeling: Particle system dynamics. ACM Siggraph Course Notes, 1994.