COMMUNICATIONS IN NUMERICAL METHODS IN ENGINEERING
Commun. Numer. Meth. Engng 2004; **00**:1–8 *Prepared using* cnmauth.cls *[Version: 2002/09/18 v1.02]*

Adaptive Multi-Resolution Triangulations Based on Physical Compression

Ricardo Marroquim[1], Paulo Roma Cavalcanti[1], Claudio Esperança[1], Luiz Velho[2]

[1] *COPPE-Sistemas, Universidade Federal do Rio de Janeiro*
Cidade Universitária - Ilha do Fundão
Caixa Postal 68530, CEP 21945-970
Rio de Janeiro, RJ, Brasil
[2] *IMPA, Instituto de Matemática Pura e Aplicada*
Estrada Dona Castorina 110
CEP 22460-320
Rio de Janeiro, RJ, Brasil

SUMMARY

This paper presents a method for generating multi-resolution adaptive triangulations of non-manifold 3D objects composed of several regions with arbitrary geometry. The process first immerses the input boundary elements inside a semi-regular adaptive tetrahedral mesh known as a BMT Triangulation. The mesh elements are then pushed towards the boundary by means of a physically based compression scheme, more specifically, a mass-spring system. The final triangulation has no degenerated tetrahedron and provides an approximation of the boundary based on a chosen resolution. Copyright © 2004 John Wiley & Sons, Ltd.

KEY WORDS: Multi-resolution triangulations, physically based triangulations, mass-spring systems.

1. INTRODUCTION

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

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.

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 point set [4]. Ruppert and Shewchuck [18, 19, 20] 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 [13, 12, 16, 11, 8, 7], on the other hand, start at the boundaries and proceed toward the center of the model, inserting new vertices to create well-shaped elements. The trick is to

Received 12/12/04
Revised 12/12/04
Accepted 12/12/04

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 [21, 2, 9, 10] generate a distribution of vertices, forcing edges and faces to be present in the final mesh, which can then be generated by a Delaunay-like algorithm.

These triangulation algorithms 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 such models, it may generate tetrahedra with small dihedral 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 adaptive algorithm to triangulate non-manifold, multi-region models, approximating their irregular boundaries instead of trying to match them exactly. Our main concern is to generate a good mesh whose boundary approximates the original one within a given tolerance.

Physically-based triangulation algorithms are not new. Molino [15] 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, thus addressing a broader set of applications.

2. SPATIAL SUBDIVISION

The algorithm has two main steps: subdivision and compression. The first step of the algorithm is to subdivide the space surrounding the model. An adaptive Binary Multi-triangulation (BMT) is used to hold the subdivision [14]. The grid obtained from the subdivision is also the base simplicial complex for the 3D triangulation. It should be noted that a BMT mesh generates elements with good angles.

The adaptive property of this mesh allows a refinement based on a given criterion. The one chosen here is the distance from a tetrahedron to a boundary surface. BMT 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 tetrahedron, the distance from its centroid to the boundary must be smaller than the average length of its edges. Also, each tetrahedron may only be subdivided up to a certain number of times specified by the user.

3. INPUT MODEL AND DISTANCE FUNCTION

Three-dimensional multi-region models are in general non-manifold. Therefore, to be properly represented, they require quite complex data-structures, such as the radial-edge data structure (RED) [22]. The RED data structure allows one to obtain any adjacency relationship in constant time. Our models are created by using the *CGC* system [5].

Throughout the algorithm, the distance from a point in space to a boundary surface is computed several times. Therefore, a fast algorithm for computing distances is required. We make use of an octree to speed up the search.

The octree root node is defined by the model's bounding box coordinates. To make a compact structure, internal and leaf node coordinates are not stored, but computed on-the-fly while traversing the tree. Thus, the only attributes maintained in an internal node are pointers to its eight children, while a leaf node stores only the list of items (faces) intersecting its box. If the list has more elements than a specified limit, the node is subdivided. However, in order to prevent the refinement criterion from triggering an infinite subdivision loop, the height of the tree is also limited.

To find the distance from a given point to the boundary of the model, its octree is traversed starting at the root node. The estimated result distance is initialized to a suitable huge value. When a leaf node is visited, the distance to the closest face is returned. When an internal node (octant) is visited, the children octants are sorted in ascending order of distance from the query point. Each of these octants is then visited recursively, but only if the corresponding distance is smaller than the best estimate so far. If the traversal returns a distance estimate that is smaller than the overall best estimate, this is updated.

4. MESH COMPRESSION

To conform the refined mesh to the boundary of the model a physical mass-spring system is used. The idea is to select a subset of the mesh (called a *d*-mesh) and compress it to fit the boundary of the model. The set of vertices on the boundary of the *d*-mesh are marked to be projected towards the boundary of the model. When these vertices start to move, the forces created are propagated to the other vertices, rearranging all of the interior tetrahedra. Springs are placed on each tetrahedron to hold their shape during the compression.

A set of vertices must be chosen to be pushed towards the boundary. One solution is to classify all tetrahedra as being *inside, outside* or *intersecting* the boundary of the model, and choose all exterior vertices of the intersecting tetrahedra. Alternatively, all interior vertices could also have been chosen. However, merely choosing a set of vertices is not sufficient, since there can be inconsistencies during the compression. For example, some tetrahedra might have all four vertices on the boundary of the model, and chances are they would be crushed against its surface. Also, some selected vertices may be far away from the boundary, creating a great disturbance when displacing them towards the boundary.

Ideally, the *d*-mesh should be close enough to the boundary of the model. Following [15], the *d*-mesh is composed of all tetrahedra incident to an enveloped set. The enveloped set contains vertices with all incident edges at least 25% inside the model. This guarantees that all tetrahedra of the *d*-mesh have a minimum of one vertex inside the model.

All vertices of the *d*-mesh that do not belong to the enveloped set are marked, and form a set of vertices to be pushed towards the boundary. This procedure insures that no tetrahedron has its four vertices marked. This set of vertices is a rough approximation of the boundary of the model. Tetrahedra that do not belong to the *d*-mesh are discarded.

The set of marked vertices contains some exterior and interior vertices, but they are all close to the boundary of the model, as can be seen in Figure 1. The more a vertex has to move to reach the boundary, the greater the force it creates on the mesh. Very high compression forces cause disturbances in the mesh that may lead to badly shaped elements.

There are two cases that have to be addressed to provide a smooth compression. First, an internal edge of the *d*-mesh, with both vertices marked, may be crushed during compression. An internal edge is identified when all tetrahedra of its *star*, i.e., tetrahedra incident to the

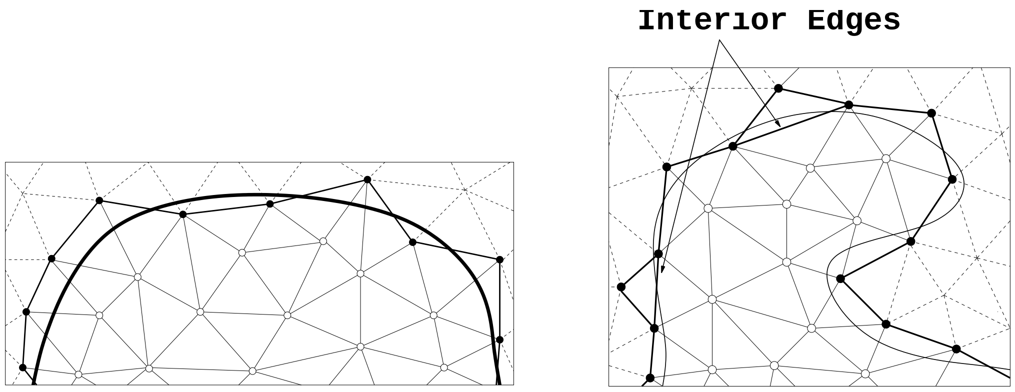

Figure 1. A 2D representation of a 3D *d*-mesh for the sake of simplicity. White vertices represent the enveloped set. Vertices marked to be projected towards boundary of the model are drawn in black. The dark black line represents the boundary of *d*-mesh. The model's boundary is the curved line and dashed lines mark discarded edges. Pointed edges (on the right) represent two interior edges connecting marked vertices.

edge, belong to the *d*-mesh. Second, edges on the boundary of the *d*-mesh with more than two incident boundary faces are identified as being *non-manifold*. These internal and non-manifold edges are subdivided in a single pass, and the *d*-mesh is computed once again. When subdividing an internal edge, the new vertex is automatically inserted in the enveloped set. Since the edge is in the interior of the mesh, this new vertex is also guaranteed to be in its interior.

In subsequent passes the mesh is checked again for remaining inadequate edges. For each such edge, the vertex with the smaller distance to the boundary is inserted into the enveloped set. Because the distance is negative inside the model, if both vertices are in the interior, the deeper one is chosen. After each pass the *d*-mesh is recomputed, and the process is repeated until no more inadequate edges are found.

5. MULTI-REGION MODELS

When working with multi-region models, Figure 2, different enveloped sets are created for each region. The enveloped set contains only vertices inside each region, therefore guaranteeing that different sets do not intersect. However, when computing the *d*-mesh, a tetrahedron might have vertices in different enveloped sets. Furthermore, this may lead to tetrahedra having all four vertices marked to be projected towards the boundary of the model, causing them to be crushed when compressed.

To solve this problem, it is necessary to have a delimiting surface between adjacent regions. This surface contains the marked vertices, and, after the compression stage, shall approximate the boundary between the regions.

When computing an enveloped set, the same rules are applied. A vertex must have all incident edges at least 25% inside the region to belong to the set. However, to find the delimiting surfaces, the *d*-mesh is only computed for some regions. A simple criterion based on

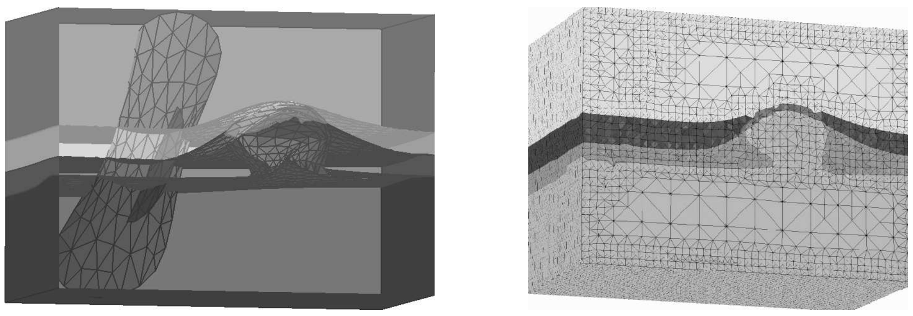

Figure 2. Internal view of a Salt Dome, and a cut of its mesh, with 5 regions.

the identification of the regions is used. When selecting the marked vertices on the boundary between two regions, only the region with the smallest identifier is considered. Solving the boundary of the d-mesh for a region automatically solves it for the adjacent region.

6. SPRING CONFIGURATION

A very important issue is how to place the springs on the physical system. Badly placed springs may not hold the shape of elements, or worse, may not prevent the tetrahedra from collapsing.

Two types of springs are used: edge and repulsion springs. An edge spring is placed on each d-mesh edge. Three repulsion springs are placed in each tetrahedron, connecting a vertex to its projection onto the plane of the opposite face (see Figure 3). This configuration proved to prevent collapses efficiently [6].

Each edge has a reference to its corresponding spring, while each face has references to two repulsion springs. An edge spring is connected by two real vertices. However, repulsion springs connect a real to a virtual vertex, thus requiring some auxiliary vertices to represent the projection of a real vertex.

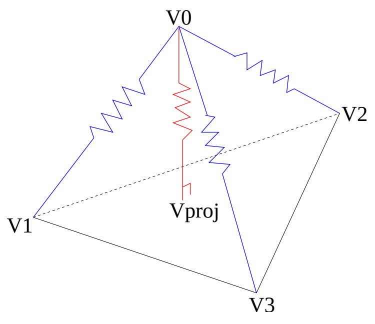

Figure 3. Incident springs to vertex V_0, where $V_0 - V_{proj}$ is a repulsion spring. The other springs are edge springs. The remaining three edge and repulsion springs for this tetrahedron are not shown here.

To compute spring forces two different equations are used: one for edge springs and the other for repulsion springs (projection edges). The edge spring equation is the Hooke's Law, as given by Eq. 1 below:

$$F = \left[k_s\left(|\triangle x| - x_o\right) + k_d\left(\frac{\triangle v \cdot \triangle x}{|\triangle x|}\right)\right]\frac{\triangle x}{|\triangle x|}, \tag{1}$$

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

For the repulsion spring there is a non-linear equation (see Eq. 2), as explained in [6]. These non-linear springs apply an infinite force as their lengths tend to zero. The values for k_d and k_s must be carefully adjusted to make the system converge. We have been using the range [0.35,1.3] and [0.85,2.3], respectively, for k_d and k_s in our implementation. The actual values may influence the quality of the final mesh.

$$F = k_s\left(|\triangle x| - \frac{x_o^2}{|\triangle x|}\right)\frac{\triangle x}{|\triangle x|}. \tag{2}$$

7. PARTICLE SYSTEM

To simulate the mass-spring oscillatory movement, the mesh is treated as a particle system [23]. 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 repulsion 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.

Also, the particles are given different mass values. Those near the boundary are made heavier than those far from it. The reason is to give more robustness to the small tetrahedra near the boundary, while giving the bigger interior ones more freedom to rearrange. This avoids crushing the small mesh elements when the interior of the mesh offers too much resistance. The mass of a particle is computed as the average of all incident tetrahedra masses. The mass of a tetrahedron is computed as the ratio between its level and the maximum subdivision level.

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.

8. 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 repulsion spring.

The first step is to reset all force accumulators, with a simple iteration through all particles. Next, the edges and faces are examined in order to compute the forces applied on their corresponding springs. 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 the values. The vector size is $6n$, where n is the number of particles in the system. The derivative vector is scaled for the time of an 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.

9. COMPRESSION CRITERIA

Spring 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, marked vertices of the d-mesh need to be pushed towards the boundary.

The initial velocity direction of a marked vertex is computed as the average of all incident boundary face normals. If the vertex is in the exterior of the mesh surface, the normals facing the interior of the incident tetrahedra are used. Otherwise, in case of an interior vertex, the normals facing outwards are used. This direction precludes a vertex from landing near edges or other vertices, spreading them apart fairly well.

Marked vertices also behave differently from the others in the system. Specifically, a marked vertex should travel towards the boundary at a constant rate. Thus, we only consider the force component which is perpendicular to the velocity vector. After each time step the velocity direction is recomputed according to the incident face normals. Therefore, marked vertices travel towards the mesh's surface while rearranging themselves along the plane perpendicular to their trajectory. If these vertices were allowed to interact freely with the system, the internal forces of the mesh would eventually return it to the initial state.

10. LIMITING THE DEFORMATION RATE

To better control the mesh during the simulation, some bounds are imposed onto the movement of the particles. These bounds are motivated by the techniques used in cloth simulation, and limit the strain and strain rate [3, 1, 17]. First, no spring may compress a tetrahedron more than 60% of its initial height. Second, in a time step, no particle may move more than 10% of the current height of any of its incident tetrahedra.

When a vertex surpasses the strain bound, it is halted during the current time step. The other vertex new positions are iteratively recomputed until all tetrahedra abide the restriction. These artificial bounds create a higher degree of robustness, while compressing the mesh. They also help to obtain better quality meshes.

By this criterion some vertices being pushed towards the boundary may be continuously halted, and never reach it. This happens more often with low refined meshes, where the vertices have to travel large distances, and the size variation, of the tetrahedra near the boundary and those in the interior, is small. This procedure is valid in cases where a perfect match of the boundary is not necessary. The simulation finishes when all marked vertices have either reached the boundary of the model, or have been halted.

11. CONCLUSIONS

We have presented an adaptive algorithm for triangulating multi-region, non-manifold 3D models. Although the boundaries of the triangulation may differ from the boundary of the original model, there are several applications where this is acceptable. For instance, in some Geoscience applications based on noisy seismic data, boundaries are only rough estimates created by specialists or software. Also, simulations may deform boundaries over time, which makes it important for triangulation algorithms to change the mesh topology as little as possible while, at the same time, maintaining a good boundary conformance.

Our algorithm allows one to specify how closely the boundary will be recovered, and has some advantages over traditional triangulation techniques, such as: (1) offering multi-resolution; (2) being capable of dealing with models possessing complex and irregular boundaries; and (3) not being affected in a high degree by numerical issues.

The algorithm has been implemented to run on ordinary personal computers with 1Gb of RAM, and generates good meshes with several million tetrahedra. It is not intended to compete in speed, say, with Delaunay-like algorithms, since it has to update the force accumulator vector at each time step. Therefore, the execution time may take from a few minutes to several hours, depending on the degree of precision set for recovering the boundaries, and the desired quality of the final mesh. Its main advantage, however, is that, at the end, there is no need to remove a large number of slivers, or bad shaped elements.

As an example, the triangulation of a tiger, shown in Figure 4, possesses 145K tetrahedra. The minimum dihedral angle is 11 degrees, and the processing time was 30 min on a Pentium 4 - 2.8C HT. The triangulation of the Salt Dome model in Figure 2 possesses 903K tetrahedra, minimum dihedral angle of 4.86 degrees, and was generated in 3 hours. Only 35 tetrahetra had minimum dihedral angle below 7 degrees (99.7% above 17.7 degrees).

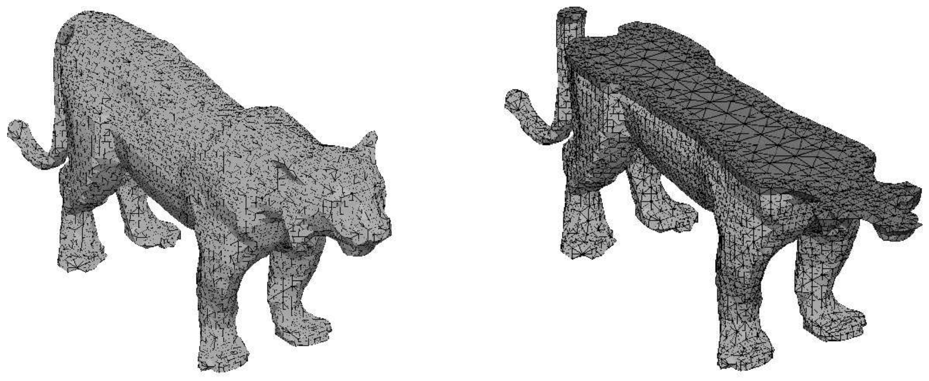

Figure 4. Triangulation of a tiger with approximately 145K tetrahedra.

Acknowledgments

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. David Baraff and Andrew Witkin. Large steps in cloth simulation. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 43–54. ACM Press, 1998.
2. Marshall Bern and David Eppstein. Quadrilateral meshing by circle packing. *International Journal of Computational Geometry and Applications*, 10(4):347–360, August 2000.
3. Robert Bridson, Ronald Fedkiw, and John Anderson. Robust treatment of collisions, contact and friction for cloth animation. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 594–603. ACM Press, 2002.
4. 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.
5. Paulo Roma Cavalcanti, Paulo C.P. Carvalho, and Luiz F. Martha. Nonmanifold modeling: An approach based on spatial subdivisions. *Computer-Aided Design*, 29(3):209–220, March 1997.
6. 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.
7. Peter Fleischmann and Siegfried Selberherr. Three-dimensional delaunay mesh generation using a modified advancing front approach. In *6th International Meshing Roundtable*, pages 267–278, Sandia National Laboratories, October 1997.
8. Pascal J. Frey, Houman Borouchaki, and Paul-Louis George. Delaunay tetrahedralization using an advancing-front approach. In *5th International Meshing Roundtable*, pages 21–46, Sandia National Laboratories, October 1996.
9. 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.
10. Xiang-Yang Li, Shang-Hua Teng, and Alper Üngör. Biting: Advancing front meets sphere packing. *International Journal for Numerical Methods in Engineering*, 49(1):61–91, September 2000.
11. Rainald Löhner. Progress in grid generation via the advancing front technique. *Engineering with Computers, Springer-Verlag*, 12:186–210, December 1996.
12. Rainald Löhner and Paresh Parikh. Three-dimensional grid generation by the advancing front method. *International Journal of Numerical Methods in Fluids*, 8:1135–1149, 1988.
13. Dimitri J. Mavriplis. An advancing front Delaunay triangulation algorithm designed for robustness. *Journal of Computational Physics*, 117(1):90–101, 1995.
14. Vinicius Mello, Luiz Velho, Paulo Roma Cavalcanti, and Claudio Silva. Bmt: A generic programming approach to multiresolution spatial decompositions. In H. Hege and K. Polthier, editors, *Visualization and Mathematics III*, pages 337–360. Springer-Verlag, Berlin, 2003.
15. 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.
16. Jaime Peraire, Joaquim Peiró, and Ken Morgan. Advancing front grid generation. In Joe F. Thompson, Bharat K. Soni, and Nigel P. Weatherill, editors, *Handbook of Grid Generation*, chapter 17. CRC Presss, 1999.
17. Xavier Provot. Deformation constraints in a mass-spring model to describe rigid cloth behavior. In Wayne A. Davis and Przemyslaw Prusinkiewicz, editors, *Graphics Interface '95*, pages 147–154. Canadian Human-Computer Communications Society, 1995.
18. Jim Ruppert. *Results on Triangulation and High Quality Mesh Generation.* PhD thesis, University of California at Berkeley, Berkeley, CA, 1992.
19. Jim Ruppert. A Delaunay refinement algorithm for quality 2-dimensional mesh generation. *J. Algorithms*, 18(3):548–585, 1995.
20. Jonathan Richard Shewchuk. *Delaunay Refinement Mesh Generation.* PhD thesis, Department of Computer Science, Carnegie Mellon University, Pittsburgh, PA, 1997.
21. Kenji Shimada and David C. Gossard. Bubble mesh: Automated triangular meshing of non-manifold

geometry by packing spheres. In *Proceedings of the 3rd ACM Symposium on Solid Modeling and Applications*, pages 409–419, Salt Lake City, Utah, May 1995.

22. Kevin Weiler. The radial-edge structure: A topological representation for non-manifold geometric boundary representations. In *Geometric Modeling for CAD Applications*, pages 3–36. North-Holland, 1988.
23. Andrew Witkin. An introduction to physically based modeling: Particle system dynamics. ACM Siggraph Course Notes, 1994.