PROVABLY ROBUST VOLUME MESHING[‡]

Chee Yap* and Sylvain Pion
Courant Institute of Mathematical Sciences
New York University, New York, NY 10012
Email: `yap,pion@cs.nyu.edu`

July 24, 2002

ABSTRACT Numerical nonrobustness of programs is well-known and widespread in all of computational science. This phenomenon has major negative impact on productivity and automation in industry. The recent development of an approach called **Exact Geometric Computation** (EGC) promises to wipe out nonrobustness problems for a large class of computational problems. One of these problems is mesh generation. Meshing is a key process in automated design, manufacturing and simulation, all vital areas in the Army mission. We believe that EGC should become a key support technology in the Army of the 21st century. In this paper, we describe the design and implementation of a provably robust algorithm to produce a Cartesian volume mesh around a triangulated 2-manifold. Two features of our solution are (1) use of EGC techniques and (2) simplicity of the algorithm. The problem of numerical nonrobustness is solved by (1). With (2), it becomes possible to exhaustively analyze all degenerate situations, thereby eliminating the other main cause of non-robustness.

1. INTRODUCTION

Program crashes attributable to numerical errors are well-known. Dramatic recent examples include the Patriot Missile errors in the Gulf War and the French Ariane Rocket disaster. Although numerical errors can often be tolerated, serious problems arise when they lead to inconsistent combinatorial structures or invalid states in a program execution. Such errors (usually benign) have become "qualitative" or "catastrophic". Catastrophic errors are not mere inconveniences, but have major impact on scientific/programming productivity, and manufacturing costs and quality. Recent development of an approach called **Exact Geometric Computation** (EGC) in Computational Geometry promises to wipe out nonrobustness problems for a large class of computational problems. Roughly speaking, the problems which current EGC techniques can practically solve is the class of bounded-depth problems with low algebraic degree. The problems of surface and volume mesh generation fall under this class. Many robust algorithms have now been implemented using EGC principles, especially in the well-known `LEDA` and `CGAL` projects. There are two libraries that support general EGC capabalities for programmers: our `Core Library` and `LEDA` reals. See Yap (1997,2002) for a general survey and the references.

The goal of this paper is to design and implement a robust algorithm for Cartesian volume mesh. Our main application is in software systems for CFD simulations. Quoting[1] industry experts in computational fluid dynamics (CFD) on a typical scenario: *in a CFD aircraft analysis with 50 million elements, we spend 10-20 minutes for surface mesh generation, 3-4 hours for volume meshing, 1 hour for actual flow analysis, and finally 2-4 weeks for geometry repair.* Using a fully robust algorithms, we can eliminate the 2-4 weeks of repair effort, at the cost of more time in surface and volume meshing (and probably the original surface modeling step). This tradeoff is expected to be favorable as the realtime human intervention is eliminated. More generally, mesh generation is a key algorithmic component in automated design, manufacturing and simulation, all vital to the Army mission. In moderm manufacturing, precision and reliability are two key goals. Numerical robustness is an important part of reliability. Hence we believe that EGC ought to be part of Army's support technology of the future.

2. DESIGN ISSUES

There are several reasons why implementations are nonrobust. We note two main ones. (A) The first reason stems from the use of approximate machine arithmetic for numerical computation. Many solutions have

[‡]This research is supported by NSF/ITR Grant #CCR-0082056. Sylvain's work is conducted under a postdoc fellowship of this grant.

[1]Tom Peters (University of Connecticutt) and Dave Ferguson (The Boeing Company), talk at the DARPA/NSF CARGO Kickoff Workshop, Newport RI, May 20-23, 2002.

been proposed here but in practice, implementors use some form of the "epsilon-tweaking trick": knowing that comparison with 0 is too fragile, they replace 0 by some small empirical (epsilon) value. (B) The second reason is related to the first. Many implemented algorithms are simply incomplete (do not cover all possibilities). When fixed precision arithmetic is assumed, the issue of completeness becomes too complicated to state and degenerate cases are typically ignored by the program logic.

In this paper, we describe and implement a fully robust volume mesh generator. We avoid the numerical problems of (A) by using the principles of EGC. Indeed, this feature is mostly automatic because we implement our algorithm using the `Core Library`. We avoid the incompleteness of (B) by making our algorithm as simple as possible. Thus we use Cartesian meshes and design the algorithm around a simple algorithmic criteria for splitting cells. Cartesian meshes have demonstrated their usefulness in the past by succeeding on arbitrarily complex geometry, when other meshing algorithms break down. To see why simplicity is important, consider the intersection of a triangulated surface Σ with a volume cell B: the topology of $\Sigma \cap \partial B$ can be a very complex planar graph. Implementations often do not handle this correctly in the sense of (A) or (B). Indeed maintaining a full-blown planar graph structure is a major piece of software on its own right. Our approach avoids this.

3. ALGORITHM

Our algorithm constructs a volume mesh around a model that is a bounded triangulated 2-manifold Σ, represented by the usual triple (V, E, T) of vertices, edges and triangles. The mesh is basically a collection of polyhedral cells. For CFD applications, each cell stores its centroid, its volume and a list of all its faces. Each face stores its centroid and area and the two incident cells. "Cartesian" means the cells are obtained as the intersection of boxes with the interior or exterior of Σ. A **box** (or hexahedra) is a set of the form $[x_1, x_2) \times [y_1, y_2) \times [z_1, z_2)$. We have three kinds of boxes: **flow, solid, cut**, indicating (respectively) boxes that are exterior to, interior to, and intersecting the manifold. The initial mesh M_0 is a uniform grid of boxes that covers the manifold Σ. This mesh is successively refined into more refined meshes, $M_0, M_1, M_2, \ldots$. At the ith stage, we have a **mesh tree** T_i whose nodes are cubes and whose leaves form M_i. We obtain T_{i+1} from T_i by **splitting** a leaf u of T_i, producing subboxes as children of u.

Our **base algorithm** is extremely simple. Extension of the base algorithm will be discussed below. The **base goal** is this: *Each box B in the mesh intersects Σ in at most one vertex and has at most one connected component.* If this condition is violated at B, we split B. We assume that boxes are in fact cubes, and splitting a cube produces 8 identical subcubes. The base goal implies that cut cells can be classified as a V-cell (if it intersects a vertex), an E-cell (if it intersects an edge but is not a V-cell) or an F-cell (otherwise).

We outline a method to achieving the base goal. Assume that each leaf in the current merge tree is associated with a V-list, an E-list and an F-list that stores (respectively) all vertices, edges and faces that intersect the cube. The algorithm to decide to split or not, using the following sequence of decisions:
(1) If the V-list has more than one vertex, split.
(2) If the V-list has one vertex, but an edge in the E-list or face in the F-list is not incident on V, split.
(3) If the V-list has only one vertex, output a V-cell.
(4) If there is more than one edge in the E-list, split.
(5) If the E-list has only one edge, but a face in the F-list is not incident to this edge, split.
(6) If the E-list has only one edge, output an E-cell.
(7) If the F-list has more than one face, split.
(8) If the F-list has one face, output an F-cell.
(9) Output a FLOW or SOLID cell.
The simple criteria used here allows us to readily prove termination and correctness. The final decision (FLOW or SOLID) can be correctly determined if we propagate suitable information during splits. The predicates for these criteria are easy to implement using the Core Library. Degeneracy is explicitly handled: the fact that the boxes of the mesh partition the initial space helps this analysis.

We consider various extensions of the base algorithm. We can introduce criteria for smoothness or buffering to ensure that the change in levels between adjacent cells is gradual. Another is to allow splits to be uneven and boxes that are not cubes. The base goal can be relaxed by requiring that σ intersects the boundary of a box B only in a single cycle.

ACKNOWLEDGEMENT We are grateful for discussions with Marsha Berger on Cartesian meshes and CFD applications; with Tom Peters and Chris Hoffmann on robust meshing in practice.

REFERENCES

Yap, C. K., "Robust geometric computation." In J. E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, chapter 35, pages 653–668. CRC Press LLC, Boca Raton, FL, 1997. Updated 2002 for a new edition. (http://cs.nyu.edu/yap/)