Building a 3D Meshing Framework Using Good Software Engineering Practices[*]

N. Hitschfeld, C. Lillo, A. Cáceres, M. C. Bastarrica, and M. C. Rivara

Computer Science Department, FCFM, Universidad de Chile
{nancy|clillo|acaceres|cecilia|mcrivara}@dcc.uchile.cl

Abstract. 3D meshing tools are complex pieces of software involving varied algorithms generally with high computing demands. New requirements and techniques appear continuously and being able to incorporate them into existing tools helps keep them up to date. Modifying complex software is generally a complex task and software engineering strategies such as object-orientation and design patterns promote modifiability and flexibility. We present the design of a 3D meshing framework based on these concepts that yields a software that is both flexible at runtime and easy to modify, while not sacrificing performance severely. We also present an evaluation of the framework design quality and performance.

1 Introduction

A mesh is a discretization of a domain geometry. It may be composed of triangles or quadrilaterals in 2D, or tetrahedra or hexahedra in 3D. Building 3D meshing tools is a challenging task involving diverse issues: (a) depending on the application field where the tools are used, different algorithms are more appropriate than others, so there is the option of having either a multiplicity of different tools or a flexible software that adapts to different contexts; (b) 3D meshing is a very active research area, where new approaches, criteria, and algorithms are proposed continuously; if a tool is to have a long life, it should be able to incorporate these changes without much effort; and (c) tools should be able to manage big meshes, so performance issues such as efficient processing and storage usage are relevant and should be taken into account.

Mesh generation tools have usually been developed by their final users, i.e. mathematicians, physicists or engineers. This caused that not always the best methods for software development have been applied. We believe that there is an opportunity to improve the quality of meshing tools by applying the best software engineering practices known.

[*]The work of N. Hitschfeld, A. Cáceres and C. Lillo was supported by Fondecyt N°1030672. The work of M. C. Rivara was supported by Fondecyt N°1040713.

1.1 Good Practices in Software Engineering

The main goal of software engineering is to develop good practices so that to obtain good software. The are qualities related to software execution such as correctness and performance, that are well understood. However, there is another set of qualities that have been gaining relevance lately: flexibility, reusability or modifiability. These qualities are relevant because the cost of modifying software is high. Algorithms and data structures have a determinant influence over performance. Similarly, software design techniques such as object-orientation, design patterns or software architecture have more influence over the attributes not related to execution. Reaching the desired software quality depends on the requirements at hand. Generally optimizing some attributes can only be done at the expense of other qualities. Sophisticated meshing tools implementing high performing algorithms and data structures are usually less reusable, and certainly less maintainable. So a compromise among the required attributes is generally the best solution.

Software reuse promotes productivity and high quality. Software already developed can be incorporated in new systems saving development time and costs, and also counting on the properties of the reused parts. One of the known efforts to make available robust, efficient, flexible and easy to use implementations of geometric algorithms and data structures is the reusable library CGAL [5]. Software families is a modern approach based on planned massive reuse. A product family is a set of products that are built from a collection of reused assets in a planned manner. There have been some attempts in using software product family concepts for building meshing tools [2, 4].

1.2 3D Tetrahedral Meshing Tools

Meshing tools allow us to solve partial differential equations numerically or to visualize objects. In 3D, different meshing tools vary in the type of the elements they manage; the most widely used are tetrahedral and hexahedral meshes. There are several 3D tetrahedral meshing tools currently available but not all of them provide the same functionality [9] varying depending on the application for which they were designed.

Three examples of known meshing tools are TetGen, TetMesh and QMG. TetGen [13] is a very efficient and robust open source tool for the generation of quality Delaunay meshes for solving partial differential equations using finite element and finite volume methods. TetGen has been developed using C++, but not necessarily object-oriented concepts, since it is implemented using a few classes and without using inheritance, polymorphism, information hiding or encapsulation. TetMesh [7] is a commercial product for the generation of quality tetrahedral meshes for finite element methods. It was originally developed in FORTRAN 77 and afterwards migrated to C. QMG [8] is an open source octree based mesh generator for automatic unstructured finite element mesh generation. It was developed in C++ and Tcl/tk using object-orientation concepts, but since it uses octrees as the main data structure, all algorithms should conform to this structure, yielding an efficient yet highly coupled tool. In general, all the mesh generation tools are focused on reaching efficiency and robustness and not extensibility and modifiability.

1.3 Our Meshing Framework

The motivation of our work is to design and develop a framework that allows us the construction of new 3D meshing tools with little effort. We would like to have the flexibility of easily interchanging or adding new input/output data formats, mesh generation algorithms for each step, quality criteria and refinement/improvement region shapes. We have already designed the architecture of a family of 2D meshing tools [2] and now we have extended it for the generation of 3D mesh generators. The framework is implemented in C++ and currently includes Delaunay and Lepp-based algorithms, among others.

In this paper we propose a 3D tetrahedral meshing framework whose design is based on object-orientation and design patterns in order to achieve the flexibility and evolvability required, without sensibly sacrificing performance.

2 Framework Analysis, Design and Implementation

The framework has been developed using object-orientation and design patterns. Functional requirements were specified using UML use-case diagrams and described with sequence diagrams. Software structure was specified using class diagrams [2].

2.1 Requirements and Analysis

A flexible and complete 3D mesh generation framework should implement each one of the following processes:

– input geometry in different formats;
– generation of an initial volume mesh that fits the domain geometry;
– refinement/improvement of a mesh in order to satisfy the quality criteria;
– smoothing of the mesh according to a certain smoothing parameter;
– derefinement of a mesh according to density requirements;
– quality evaluation of the generated mesh;
– visualization of the mesh.

The specification of the input geometry and physical values can be generated by CAD programs or by other mesh generation tools. We have already

[2]Part of the framework design documentation can be found in `http://www.dcc.uchile.cl/~nancy/framework/diagrams.html`.

implemented the Off and Mesh formats. The algorithms that generate the initial volume mesh can receive as input the domain geometry described as a triangulated surface mesh or as a general polyhedron. We have implemented an initial volume mesh that fulfills the Delaunay condition and an initial volume tetrahedralization that may not satisfy it.

The initial volume mesh is the input of the refinement step that divides coarse tetrahedra into smaller ones until the refinement criteria are fulfilled in the indicated region. Either the initial volume mesh or the refined mesh can be the input of the improvement process. The user must specify an improvement criterion and a region where the improvement is to be applied. At the moment, we have implemented the refinement and improvement strategies based on the Lepp-concept [10] but it is possible to add other strategies, such as the Delaunay refinement [11], without much effort. The smoothing and derefinement processes are also applied according to a criterion and over a region of the domain.

Once a mesh has been processed, the user has the possibility of evaluating its quality according to different criteria. This is useful if the user wants to see the distribution and percentage of good and bad elements in the mesh. The visualization process is currently done using Geomview [1]. Each mesh generation process can also be skiped by representing it with a dummy algorithm.

2.2 Design and Implementation

Figure 1 shows the most important part of the meshing framework class diagram. We represent each mesh generation process as an abstract class and each different strategy implementing each process as a concrete subclass. For example, the *Refine* abstract class is realized by subclasses `LeppAlgorithms` and `VoronoiRefinement`, as shown in Fig. 2. We also represent all the criteria with the *Criterion* abstract class and all the region shapes with the *Region* abstract class in Fig. 1. This allows a programmer to add a new criterion, region shape or strategy by adding just a concrete class that inherits from the respective abstract class and without modifying the source code. The code of a particular mesh generator uses the abstract classes code, and the user must select which concrete algorithms he/she wants to use for each mesh generation process, criteria and region shapes. For example, *GenerateVolumeMesh* can be realized with `GMVDelaunay` to generate a Delaunay volume mesh. Similarly, the abstract class *Refine* can be realized with `LeppAlgorithms` receiving a *Region* and a *Criterion* as parameters realized as `WholeGeometry` and `LongestEdge`, respectively (see Fig. 2).

The mesh is modeled as a container object. The `Mesh` class provides methods for accessing and modifying its constituent elements (tetrahedra, faces, edges and points). `Tetrahedron`, `Face`, `Edge` and `Vertex` are also classes, each of them providing concrete functionality and also providing access to the neighborhood information. The mesh quality evaluation is modeled using the `Evaluate` class. This class uses a criterion and, according to some user parameters, it classifies the elements and generates a file with the evaluation results as output.

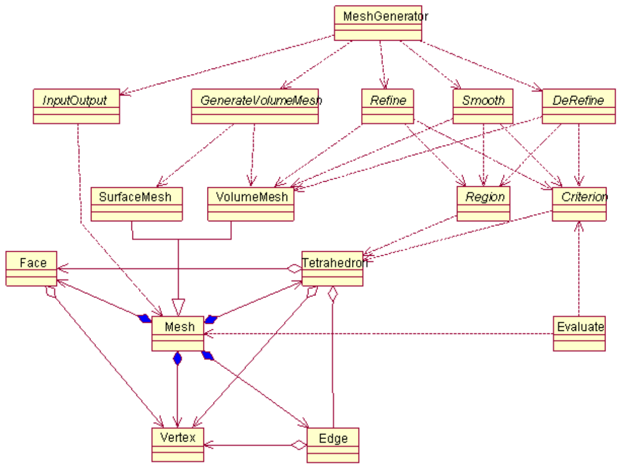

Fig. 1. Framework general class diagram

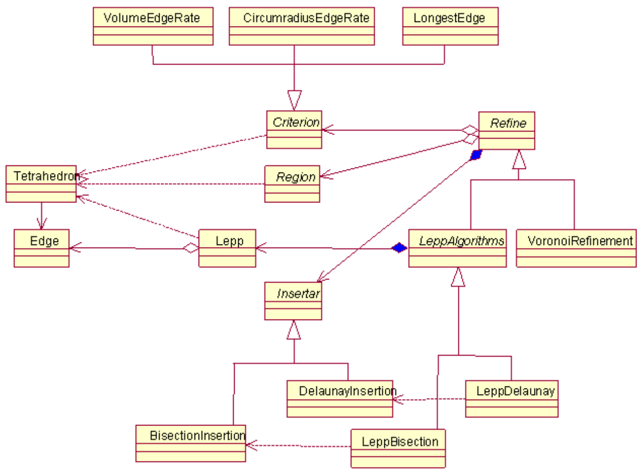

Fig. 2. Partial detailed class diagram

In the framework implementation, we used several design patterns [6]. Each different mesh generation process and each criterion follows the Strategy pattern. The region shape follows the Composite pattern. The mesh evaluation class follows the Observer pattern where the observed object is the Mesh. The interface is organized using the Command pattern. The mesh is a Singleton.

3 3D Framework Evaluation

Our goals was to achieve flexibility, modifiability and performance. While the first two depend on a good design, the last can only be evaluated at runtime.

3.1 Design Evaluation

Metrics for object-oriented design provide quantitative mechanisms for estimating design quality. Good metrics evaluation shows a good design but it does not guarantee good software. However, bad metrics evaluation almost guarantees bad software results. In this work, we use the metrics proposed in [3] because they are widely used for measuring flexibility and extensibility. A brief description of each metric is included in Table 1 and Table 2 shows the results of applying the metrics to the framework class diagram.

Name	**Description**
Weighted Methods per Class (WMC)	Sum of all method's complexity within a class. The number of methods and their complexity indicate the effort required for implementing a class. The larger the number of methods the more complex the inheritance tree will be, and also the more specific a class becomes, limiting its reusability.
Depth of Inheritance Tree (DIT)	Maximum length between the node and the root in the inheritance tree. The deeper the class, the more probable the class inherits a lot of methods. A deep class hierarchy may imply a complex design.
Number of children (NOC)	As the number of children grows, the abstraction represented by a class becomes vague, and its reusability decreases.
Coupling Between Objects (CBO)	It is the number of collaborations between a class and the rest of the system. As this number grows, the class reusability decreases. High values also make modifications and testing harder.
Response for a Class (RFC)	It is the number of methods that may be potentially executed as a response to a message received by a class object. As this metric grows, testing the class becomes harder, and the class complexity also grows.
Lack of Cohesion in Methods (LCOM)	A high LCOM indicates that methods can be grouped in disjoin sets with respect to attributes, and form two or more classes with them.

Table 1. Design metrics

	WMC	**DIT**	**NOC**	**CBO**	**RFC**	**LCOM**
Minimum	1	0	0	0	1	0
Maximum	36	2	8	22	36	100
Medium	7.60	0.60	0.50	3.87	12.67	30.98
St. Deviation	7.11	0.66	1.43	4.18	7.87	36.73

Table 2. Tool design evaluation

The WMC metric shows a value within the normal scope for this kind of system. There are only two classes out of this scope: `Predicates` and `Tetrahedron`. The former reuses a library described in [12]. The latter class contains several methods required for the Delaunay algorithm, such as the sphere test; thus it can be divided into two different classes: one that includes basic concepts about tetrahedron, and another one extending the first one that contains specific methods for Delaunay implementation. The DIT metric is always small, showing a low design complexity. The same occurs with the NOC metric. Both metrics can grow when extending the design. The CBO metric value is normal for an application with this size (52 classes). The maximum value is achieved in the `MeshGenerator` class that references the classes implementing the main processes and classes holding the main parameters, such as criteria and regions; this class is only used when the system is operated using the command line, so it can be excluded from the analysis. For the RFC metric, the values are within the normal scope for all classes except for `Predicates` and `Tetrahedron` for the same reasons explained for WMC. Finally, the LCOM metric has high values; however, the highest values are only found in abstract classes: their methods have no code, so they do not access instance variables; thus, the metric has no effect.

3.2 Performance Evaluation

Performance evaluation in 3D meshing tools is mainly related to the time it takes to execute typical mesh processes. Figure 3 shows an example of a volume before and after applying the refinement process and Fig. 4 shows the time as a function of the number of refined tetrahedra.

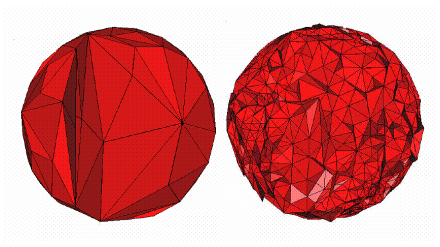

Fig. 3. Refinement process example: 170 points and 441 tetrahedra (left), and 8,823 points and 45,518 tetrahedra (right)

In general terms, a generated meshing tool with the same functionality as TetGen is around two times slower with respect to refinement and improvement. This difference may be due to the fact that in TetGen all data structures are accessed directly, not using information hiding or encapsulation, and there is no dynamic binding. On the other hand, the mesh generated mesh tool uses all these concepts.

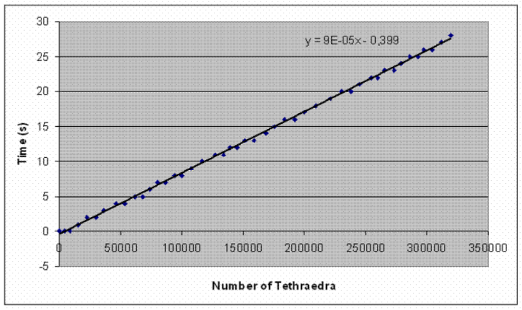

Fig. 4. Refinement framework time performance (executed in a Pentium IV processor with 2.6 GHZ and 1 GB RAM)

4 Conclusion

3D meshing tools are extremely complex software that apply resource consuming algorithms to big meshes. This is why performance has been the main focus of research around implementing this kind of software. However, since computers tend to have more and cheaper memory and CPU capacity, some of the burden has shifted towards the development process of the tools. In this context, we proposed an object-oriented design based on design patterns that has proved to yield a flexible and modifiable framework, without severely sacrificing performance.

References

1. Geometry Center at the University of Minnesota. Geomview, 1996. `http://www.geomview.org`.
2. M. C. Bastarrica and N. Hitschfeld-Kahler. Designing a Product Family of Meshing Tools. *Advances in Engineering Software*, 37(1):1–10, Jan 2006.
3. Shyan R. Chidamber and Chris F. Kemerer. A Metrics Suite for Object-Oriented Design. *IEEE Transactions on Software Engineering*, 20(6):476–493, June 1994.
4. A. H. ElSheikh, W. S. Smith, and S. E. Chidiac. Semi-formal design of reliable mesh generation systems. *Advances in Engineering Software*, 35(12):827–841, 2004.
5. Andrea Fabri. CGAL- the computational geometry algorithm library. In *Proceedings of the 10th Annual International Meshing Roundtable*, 2001.
6. Erich Gamma, Richard Helm, Ralph Hohnson, and Hohn Vlissides. *Design Patterns: Elements of Reusable Object Oriented Software*. Addison-Wesley, 1995.
7. Paul-Louis George, Frédéric Hecht, and Éric Saltel. TetMesh-GHS3D V3.1, the fast, reliable, high quality tetrahedral mesh generator and optimiser, 1986. White paper, `http://www.simulog.fr/mesh/gener2.htm`.

8. Scott A. Mitchell and Stephen A. Vavasis. Quality mesh generation in three dimensions. In *Proceedings of the Eighth Annual Symposium on Computational Geometry*, pages 212–221, Berlin, Germany, 1992. ACM.
9. Steve Owen. Meshing software survey, 1998. `http://www.andrew.cmu.edu/-user/sowen/softsurv.html`.
10. María Cecilia Rivara. New Longest-Edge Algorithms for the Refinement and/or Improvement of Unstructured Triangulations. *International Journal for Numerical Methods in Engineering*, 40:3313–3324, 1997.
11. Jim Ruppert. A Delaunay Refinement Algorithm for Quality 2-Dimensional Mesh Generation. *Journal of Algorithms*, 18(3):548–585, May 1995.
12. J. Shewchuk. Adaptive Precision Floating-Point Arithmetic and Fast Robust Geometric Predicates. *Discrete & Comp. Geometry*, 18(3):305–363, 1997.
13. H. Si and K. Gärtner. Meshing Piecewise Linear Complexes by Constrained Delaunay Tetrahedralizations. In *Proc of the 14^{th} International Meshing Roundtable*, 2005.