The validation of a triangulated boundary representation in 3D

Research plan

M.E. Hoefsloot - 1007157
Faculty of Civil Engineering and Geosciences
Department of Geodesy, Department of OTB
Delft University of Technology
Jaffalaan 9, 2628 BX Delft, the Netherlands
E-mail: M.E.Hoefsloot@student.tudelft.nl

1. Introduction

For many reasons we want to have a boundary representation of an object in 3D, which is valid and closed. One reason is that we want to construct a tetrahedralization; therefore, the boundary representation should be a valid and closed triangulation. For constructing this tetrahedralization, the program TetGen will be used. The input of TetGen should be a valid and closed representation, if not TetGen detects the invalid polygons. The aim of this research is to conduct a solution to detect invalid triangles, to make them valid and to repair unclosed triangulated 3D boundary representations.

2. Theoretical framework

The theory in this chapter is an extract of "Delaunay Triangulation and Meshing" (1998).

2.1 Triangle and tetrahedron

Triangle: While the triangle is a well-known object, a clear definition will be given. A triangle is a 3-sided polygon, it is defined by the ordered list of its three vertices, denoted as P_i, which are given counter clockwise

$$K = (P_1, P_2, P_3). \tag{1}$$

There are six ways (or permutations) for expressing the vertices defining a triangle. In the case where an orientation is defined, only three permutations are relevant. Thus for a triangle in a plane, the orientation is implicitly defined using the normal of the plane, and the three possible definitions imply that its surface is signed. Thus, the triangle considered will have a strictly positive surface. Therefore, the surface area S_K (in 2D) is positive and given by

$$S_K = \frac{1}{2}\begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix} \tag{2}$$

where x_i, y_i are the coordinates of vertex P_i $(i = 1,3)$ and $|.|$ stands for the determinant. This definition enables to explicitly define the sides (or edges) of a given triangle. Edge $i, (i = 1,3)$, denoted as a_i, is the edge joining vertex P_{i+1} to vertex P_{i+2} (while $P_i = P_{i-3}$ if $i > 3$ is assumed).

Tetrahedron: A tetrahedron is a polyhedron with four triangular faces. It is well defined by the ordered list of its four vertices P_i

$$K = (P_1, P_2, P_3, P_4). \tag{3}$$

There exits twelve permutations for expressing the vertices defining an oriented tetrahedron. This text assumes that the faces are oriented, thus their normals are also oriented. In addition, the volume is signed. Let V_K be the volume of element K, then V_K is defined as:

$$V_k = \frac{1}{6}\begin{vmatrix} x_1 & y_1 & z_1 & 1 \\ x_2 & y_2 & z_2 & 1 \\ x_3 & y_3 & z_3 & 1 \\ x_4 & y_4 & z_4 & 1 \end{vmatrix} \tag{4}$$

where x_i, y_i, z_i are the coordinates of vertex P_i of the element. This format enables to implicitly define the four faces of the element. A face of *K* is an ordered list of three vertices:

- face 1: $P_4 \quad P_3 \quad P_2$,
- face 2: $P_1 \quad P_3 \quad P_4$,
- face 3: $P_4 \quad P_2 \quad P_1$,
- face 4: $P_1 \quad P_2 \quad P_3$.

Similarly, the edges of K are implicitly defined as the following ordered pairs:

- edge 1: $P_1 \quad P_2$,
- edge 2: $P_1 \quad P_3$,
- edge 3: $P_1 \quad P_4$,
- edge 4: $P_2 \quad P_3$,
- edge 5: $P_2 \quad P_4$,
- edge 6: $P_3 \quad P_4$,

Each edge is defined from its first endpoint to its second endpoint.

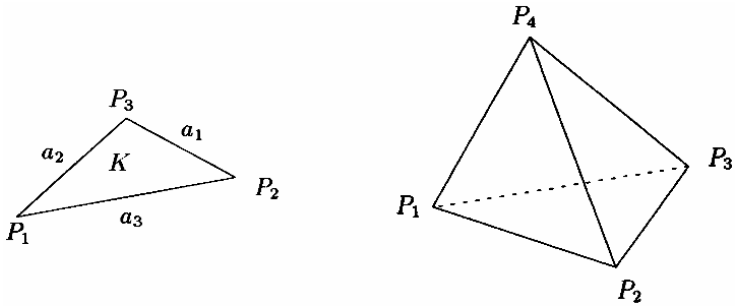

Figure 1: A triangle (left) and a tetrahedron (right)

2.2 Triangulation and tetrahedralization

To define a triangulation, the concept of convex hulls is needed.

Convex hull: Let *S* be a set of points in R^3. If the P_i's are these points, then

$$\sum_{i=1}^{n} \lambda_i P_i \tag{5}$$

represents a linear combination of points in *S*. These combinations of *n* members of *S*, for $\sum_{i=1}^{n} \lambda_i = 1$, define a subspace of R^3, which is referred to as the affine hull of the P_i's. If, for all $i, \lambda_i \geq 0$, such combinations are said to be convex. The convex hull of *S*, denoted as $Conv(S)$, is the subset of R^3, generated by all the convex linear combinations of the members of *S*. This hull is the smallest convex set including *S*.

Triangulation: Let *S* be a set of points in R^d $(d = 2 \text{ or } d = 3)$, the convex hull of *S* defines a domain Ω in R^d. If *K* is a simplex (triangle or tetrahedron according to *d*), then

Definition 1: *T_r is a simplical covering of Ω if the following conditions hold:*

- (*H*0) *The vertices of the elements in T_r is exactly S.*
- (*H*1) $\Omega = \bigcup_{K \in T_r} K$.
- (*H*2) *Every element K in T_r is non-empty.*

Tetrahedralization: In triangulations, the primitives with the highest dimension are faces. In a three dimensional triangulation, i.e. a tetrahedral network, the primitives with the highest dimension are tetrahedrons. A tetrahedral network is called a tetrahedralization. Using tetrahedrons is the easiest way in computing volumes and overlays.

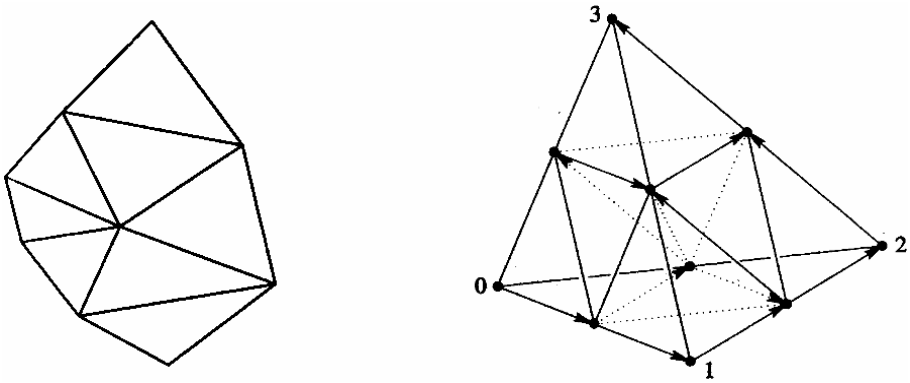

Figure 2: A triangulation (left) and a tetrahedralization (right)

2.3 Valid triangulation

In most cases a valid and closed covering (triangulation) is assumed. The aim of this research is to conduct a solution to repair unclosed and invalid triangulations. Therefore, the following definition is given:

Definition 2: *T_r is a valid triangulation of Ω, if T_r is a covering following Definition 1 and if the following conditions hold:*

- (*H*3) *The intersection of any two elements in T_r is either*
  - *an empty set,*
  - *a vertex,*
  - *an edge.*

3. First exploration

3.1 Topological relations

To come to a solution of the problem, the first step is to make an inventory of all topological relations exiting in 3D. A topological relation is defined as the set of properties, which are

invariant under homeomorphisms. When there is a kind of elastic transformation, metric properties are changing, topological properties not. In this research, three kinds of objects will be used: vertices, edges and faces, but only triangles). In this paragraph, all possible relations between these objects will be discussed. To be able to distinguish topologic relations especially those between two edges, two faces or an edge and a face, some spaces will be defined according to Pigot (1990, pp. 375).

Definition 3: *An n-simplex C can be divided into three parts:*

- *Interior set ° of an n-simplex C: a point x is an interior point of C provided there exist an open subset U such that x is an element of U and U is strictly contained within C. The union of all such points is the interior set.*
- *Boundary set ∂ of an n-simplex C: ∂C = C - C°*
- *Exterior set of an n-simplex C: complement of C*

Above definition can be applied for vertices, edges and faces. The scheme in Table 1 can be used to find topological relations between two objects. Looking to definition 3 and the definition of a face, the boundary of the face consists of the edges and the vertices defining these edges. As a help to find the relationships between two faces and between an edge and a face, it seems nice to add the double-boundary set ∂∂ of *C*, which separates edges and the vertices defining the boundary of a face.

Table 1: Application of definition 3

C	*C°*	*∂C*	*∂∂C*
vertex	-	vertex	-
edge	edge	vertices	-
face	face	edges	vertices

All relations, which have something to do with vertices, are easy, because the interior of a vertex is empty. The easiest relation is the point-point-relation. Two points can be disjoint or equal, this is easy to determine. Then, the vertex-edge relation. A vertex and an edge can be disjoint, meet or intersect. The relations between a vertex and a face are almost the same.

According to Pigot (1991), eight unique topological relations between two edges can be distinguished (see Figure 3): disjoint, meet, common boundaries, concur, overlap, equal, cross and intersect. Only three of them are problem free, the others will be used in this research.

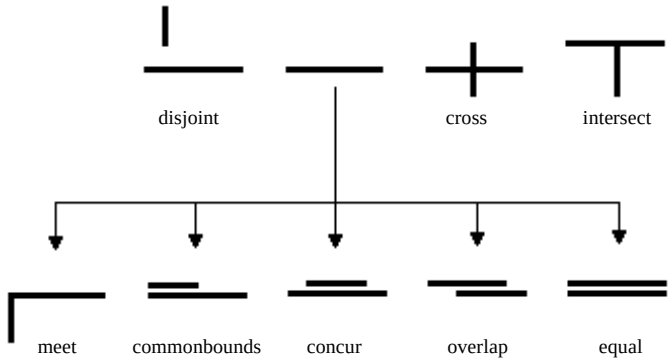

Figure 3: Eight unique relationships between two edges

Topological relations can also be put in a scheme. Let *V* be a vertex, let, *E* be an edge and let *F* be a face. A starting-point of it will be given in Table 2 until Table 5; the other topological relations (edge-face and face-face) will be worked out and implemented in the final report.

Table 2: vertex-vertex

		disjoint	equal
vertex V	∂V	-	X
vertex V	∂V	-	X

Table 3: vertex-edge

		disjoint	meet	intersect
vertex V	∂V	X	X	X
edge E	∂E	-	X	-
	E°	-	-	X

Table 4: vertex-face

		disjoint	meet	intersect	concur
vertex V	∂V	X	X	X	X
face F	$\partial\partial F$	-	X		
	∂F	-		X	
	F°	-			X

Table 5: edge-edge

		disjoint	concur	meet	intersect	intersect	cross	equal *
edge E	∂E	X	X	X	X			X
	E°	X	X			X	X	X
edge E	∂E	-		X		X		X
	E°	-	X		X		X	X

* or overlap or commonbounds

Properties of most of the relations will be used in computing intersections. Therefore, it is needed to make a plan on what to do in which situation.

3.2 Computing intersections

From "Linear Algebra" we know, there is only an intersection of a line (vector) and a face (spanned by vectors) if the vectors are linear independent. So, if T is a triangle spanned by two vectors ($\left[x_2^F - x_1^F\right]$ and $\left[x_3^F - x_1^F\right]$) and a base-vector ($\left[x_1^F\right]$), and if L is a line ($\left[x_2^L - x_1^L\right]$). There is an intersection if below equation has a solution. In addition, if that intersection is inside the triangle.

$$\alpha_1\left(\begin{bmatrix}x_2^F\\y_2^F\\z_2^F\end{bmatrix}-\begin{bmatrix}x_1^F\\y_1^F\\z_1^F\end{bmatrix}\right)+\alpha_2\left(\begin{bmatrix}x_3^F\\y_3^F\\z_3^F\end{bmatrix}-\begin{bmatrix}x_1^F\\y_1^F\\z_1^F\end{bmatrix}\right)+\begin{bmatrix}x_1^F\\y_1^F\\z_1^F\end{bmatrix}=\lambda_1\left(\begin{bmatrix}x_2^L\\y_2^L\\z_2^L\end{bmatrix}-\begin{bmatrix}x_1^L\\y_1^L\\z_1^L\end{bmatrix}\right) \tag{6}$$

Computing a point is inside a triangle is somewhat easier, and then equation (7) has to be derived. There is a big difference between calculating this in a software program like MATLAB, a registered trademark of the Mathworks, or in C++, although equations in MATLAB are written in C. Therefore, solving equations is just the beginning.

$$\alpha_1\left(\begin{bmatrix}x_2^F\\y_2^F\\z_2^F\end{bmatrix}-\begin{bmatrix}x_1^F\\y_1^F\\z_1^F\end{bmatrix}\right)+\alpha_2\left(\begin{bmatrix}x_3^F\\y_3^F\\z_3^F\end{bmatrix}-\begin{bmatrix}x_1^F\\y_1^F\\z_1^F\end{bmatrix}\right)+\begin{bmatrix}x_1^F\\y_1^F\\z_1^F\end{bmatrix}=\begin{bmatrix}x_1^P\\y_1^P\\z_1^P\end{bmatrix} \tag{7}$$

3.3 TetGen in general

TetGen is a program, written by Hang Si, for generating quality tetrahedral meshes and three-dimensional Delauney triangulations. It currently computes exact Delauney tetrahedralizations, constrained Delauney tetrahedralizations, and quality tetrahedral meshes. TetGen incorporates a suit of geometrical and mesh generation algorithms. A brief description of these algorithms used in TetGen can be found in the first section of the user's manual (Si, 2004) and a description of all relevant algorithms can be found in next paragraph.
The distributed version of TetGen includes the following files:

- README (3 kB)
- LICENSE (3 kB) Copyright notices.
- tetgen.h (90 kB) Header file of the TetGen library.
- tetgen.cxx (719 kB) C++ source code of the TetGen library.
- predicates.cxx (189 kB) C++ source code of the geometric predicates.
- makefile (2 kB) file for compiling TetGen.
- manual.pdf (342 kB) User's manual.
- example.poly (5 kB) A sample data file.

TetGen should run on all 32-bit and 64-bit computers. Use an ANSI C++ compiler to compile the program. The easiest way to compile it is to edit and use the included makefile. Before compiling, read the makefile, which describes the options, and edit it accordingly. You should specify the C++ compiler, and the level of optimization. After compiling, type "tetgen -h" for a brief introduction into TetGen. The included user's manual contains documentation and examples. TetGen may be freely copied, modified, and redistributed under the copyright notices given in the LICENSE file.

3.4 Relevant algorithms in TetGen

In this paragraph, a short description of some relevant algorithms is given. Although the code of Hang Si is well readable, it is comprehensive, so it will be hard to understand and use it.

- *Fast and Robust Triangle-Triangle Overlap Test*
  Several geometric predicates are defined. Their parameters are all points. Each point is an array of two or three double precision floating-point numbers. The geometric predicates are:
  int tri_tri_overlap_test_3d(p1,q1,r1,p2,q2,r2)
  int tri_tri_overlap_test_2d(p1,q1,r1,p2,q2,r2)
  int tri_tri_intersection_test_3d(p1,q1,r1,p2,q2,r2,coplanar,source,target)
  The last is a version that computes the segment of intersection when the triangles overlap. Each function returns 1 if the triangles (including their boundary) intersect, otherwise 0.
  (line 4176 of predicates.cxx)

- *tritritest() Test if two triangles are intersecting in their interior.*
  One triangle is represented by 'checktet', the other is given by three corners 'p1', 'p2' and 'p3'. This routine calls tri_tri_overlap_test_3d(). In the case that two triangles share exactly one vertex, we shrink one triangle a little bit inside before calling this routine. If no, four points are coplanar; the shrink does not change the test result. 'eps' is the relative epsilon value used to determine coplanar points. It is given here explicitly which means it may be different with the global 'b->epsilon'.
  (line 13248 of tetgen.cxx)

In tritritest () also the number of sharing points will be tested from which some things can be concluded.

- No common vertex, two triangles are completely separated.
- One vertex is common.
  - Get the common vertex of the first triangle in 'sp1', and other two vertices are 'vp1' and 'vp2'.
  - Do the triangle-triangle intersection test between two separated triangles.
- Two vertices are common.
  - Get the two common vertices in 'sp1' and 'sp2', two other vertices in 'vp1' and 'vp2'.
  - Test if the four vertices are coplanar.
  - They can be intersecting if one triangle includes other one.
  - The four points are degenerate, e.g. three of them are collinear.
- Three vertices are common, the trivial case, two faces are the same.

Above algorithms test whether two triangles intersect or not. Two problems, the first conclusion is not right, and it is still not clear if the exact intersection is derived. Solving these problems will be part of the research; it is probable much code has to be written.

3.5 Phased validation

It is clear the validation has to be done in small steps, because of the programming and because of the different topological relations. Therefore, a framework has to be designed and every step or possibility has to be added to it.

4. Research plan

4.1 Research Aim

As said in the first chapter, the outcome of this research should be a programming code, which can be used to change an invalid triangulated boundary representation into a valid one in 3D. This will be done in six steps.

1. Make an inventory of all topological relations that exist between two points, a point and a line, a point and a face, two lines, a line and a face and two faces.
2. Find a way to test whether two triangles intersect and to determine their topological relation.
3. Find the way to derive the intersection points.
4. Look to the solution of Hang Si (TetGen), and use this program to implement the new solution in C++.
5. Look to the solutions or parts of solutions made by other programmers and compare them.
6. Find possibilities, like indexing methods, exist to make the algorithm more efficient.

The steps are needed to give an answer to the research question:

What is an efficient way to change an invalid 3D triangulated boundary representation into a valid and closed one?

The use of the code of Hang Si and the use of C++ are the only preconditions that are defined in the assignment. This is because the code of Hang Si is good, and it is extensible by others because it is written in C++ and therefore well readable.

4.2 Time-schedule

This report is part of the course "Individual Assignment" and this course is part of the master study "Geodetic Engineering - Geo-information and Land Management". In the professional practice of an engineer, problem solving is core business. A central element in this problem solving is designing. In the "Individual Assignment", this designing will be trained in an integrated way, using a real life problem. This assignment should be done individually and it has to be completed within 5 months. The course can be done in Dutch as well as English; this depends on the topic. The course contains three central elements:

- making a time-schedule for the assignment;
- making the specific design;
- presenting the results of the research project by writing a report and an optional oral presentation.

This report is the first element.

The calendar time of the course is 5 months and the clock time is 8 Ects (224 hours). Because the starting date of this research is October 1 2004, it should be finished before March 1 2005. I have planned to work 2 or 3 days a week on this course, 2 days a week from October till December and three days a week from January, every day encompasses 6 hours of work; therefore I hope this planning will be obtained:

Research plan	**Main research**			**Report**
October	November	December	January	February
36 hour	48 hour	36 hour	48 hour	24 hour / 36 hour

This schema the total period is divided in 3 parts, the research plan, the main research and the writing of the report, the dates given below are the start and end dates of these parts:

- 2004-10-01: Official start of the course
- 2004-10-31: Research plan is finished
- 2004-11-01: Start of the main research
- 2004-01-15: Main research is finished
- 2005-01-16: Start of the writing of the report
- 2005-02-11: Report is finished
- 2005-03-01: Course should be finished

November will be used for literature study, to obtain answers to the three first research steps. In December and January the central part of the research will happen, step four the implementation in C++. When time is left, the last two steps will be carried out.

References

George, P. and Borouchaki, H. (1998) Delaunay Triangulation and Meshing, Application to Finite Elements, pp. 5, 6, 9, 10, 13, 14.

Pigot, S. (1991) Topological Models for 3D Spatial Information Systems, *Auto-Carto 10: Technical Papers of the 1991 ACSM-ASPRS Annual Convention*, Baltimore: ACSM-ASPRS, 1991. 6: 368-392.

Si, H. (2004) TetGen, A Quality Tetrahedral Mesh Generator and Three-Dimensional Delaunay Triangulator, User's Manual (version 1.3), pp. 5-8, http://tetgen.berlios.de.

Literature proposed for research

George, P. and Borouchaki, H. (1998) Delaunay Triangulation and Meshing, Application to Finite Elements, pp. 5, 6, 9, 10, 13, 14.

Lay, D.C. (1998) Linear algebra and its applications, second edition, Addison-Wesley, pp. 34, 35, 42, 62, 166, 216, 381.

Pigot, S. (1991) Topological Models for 3D Spatial Information Systems, *Auto-Carto 10: Technical Papers of the 1991 ACSM-ASPRS Annual Convention*, Baltimore: ACSM-ASPRS, 1991. 6: 368-392.

Schneider, P. and Eberly, D. (2003) Geometric tools for computer graphics, San Francisco: Morgan Kaufmann.

Si, H. (2004) TetGen, A Quality Tetrahedral Mesh Generator and Three-Dimensional Delaunay Triangulator, User's Manual (version 1.3), pp. 5-8, http://tetgen.berlios.de.