The validation of a triangulated boundary representation in 3D

Literature

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

The validation of a triangulated boundary representation in 3D

Literature

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

Contents

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 could 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.

This report contains a literature overview on this topic, it is a kind of summary of books, papers and articles that are available and useable. Every chapter describes only one article, paper or book, and all chapters are ordered on the writers family name alphabetically.

As a conclusion, I would like to thank here those who contributed, in one way or another, to this report, and above all my supervisor Edward Verbree.

Keywords: mesh generation, triangulation, tetrahedralization, triangle, tetrahedron, relationship, intersection, TetGen.

2. Robust and efficient Cartesian mesh generation for component-based geometry

Aftosmis M.J., Berger M.J., Melton J.E. (1997) Robust and efficient Cartesian mesh generation for component-based geometry, Technical Report AIAA-97-0196, US Air Force Wright Laboratory, pp. 2-4.

2.1 Abstract

This article documents a new method for rapid and robust Cartesian mesh generation for component-based geometry. The new algorithm adopts a novel strategy, which first intersects the components to extract the wetted surface before proceeding with volume mesh generation in a second phase. The intersection scheme is based on a robust geometry engine that uses adaptive precision arithmetic and which automatically and consistently handles geometric degeneracies with an algorithmic tie-breaking routine. The intersection procedure has worse case computational complexity of O(N log$N$) and is demonstrated on test cases with up to 121 overlapping and intersecting components including a variety of geometric degeneracies.

The volume mesh generation takes the intersected surface triangulation as input and generates the mesh through cell division of an initially uniform coarse grid. In refining hexagonal cells to resolve the geometry, the new approach preserves the ability to directionally divide cells, which are well-aligned with local geometry. The mesh generation scheme has linear asymptotic complexity with memory requirements that total approximately 14 words per cell. The mesh generation speed is approximately 10^6cells per minute on a 195 MHz RISC R10000 workstation.

2.2 Intersection of Generally Positioned Triangles in R^3 (pp. 2-4)

With the task of intersecting a particular triangle reduced to an intersection test between that triangle and those on the list of candidates provided by the ADT, the intersection problem is re-cast as a series of tri-tri intersection computations. Figure 1 shows a view of two intersecting triangles as a model for discussion. Each intersecting tri-tri pair will contribute one segment to the final polyhedra that will comprise the wetted surface of the configuration. The assumption of data in *general* (as opposed to *arbitrary*) position implies that the intersection is always non-degenerate. Triangles may not share vertices, and edges of tri-tri pairs do not intersect exactly. Thus, all intersections will be proper. This restriction will be lifted in later sections with the introduction of an automatic tie-breaking algorithm.

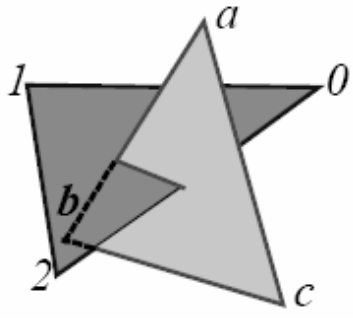


Figure 1: An intersecting pair of generally positioned triangles in three dimensions.

Several approaches exist to compute such intersections but a particularly attractive technique comes in the form of a Boolean test. This predicate can be performed robustly and quickly using only multiplication and addition, thus avoiding the inaccuracy and robustness pitfalls

associated with division using fixed width representations of floating point numbers. It is useful to present a rather comprehensive treatment of this intersection primitive because subsequent sections on robustness will return to these relations.

For two triangles to properly intersect in 3D space, the following conditions must exist:

1. Two edges of one triangle must cross the plane of the other.
2. If condition (1) exists, there must be a total of two edges (of the six available), which pierce within the boundaries of the triangles.

One approach to checking these conditions is to directly compute the pierce points of the edges of one triangle in the plane of the other. Pierce locations from one triangle's edges may then be tested for containment within the boundary of the other triangle. This approach, while conceptually simple, is error prone when implemented using finite precision mathematics. In addition to demanding special effort to trap out zeros, the floating-point division required by this approach may result in numbers not exactly representable by finite width words. This results in a loss of control over precision and may cause serious problems with robustness.

An alternative to this slope-pierce test is to consider a Boolean check based on computation of a triple product without division. A series of such logical checks have the attractive property that they permit one to establish the existence and connectivity of the segments without relying on the problematic computation of the pierce locations. The final step of computing the locations of these points may then be relegated to post-processing where they may be grouped together and, since the connectivity is already established, floating point errors will not have fatal consequences.

The Boolean primitive for the 3D intersection of an edge and a triangle is based on the concept of the signed volume of a tetrahedron in R^3. This signed volume is based on the well-established relationship for the computation of the volume of a simplex, T, in d dimensions in determinate form (see for ex. O'Rourke (1998)). The signed volume $V(T)$ of the simplex T with vertices $(v_0, v_1, v_3, \ldots, v_d)$ in d dimensions is:

$$d!V\left(T_{v_0 v_1 v_3 \ldots v_d}\right) = \det\begin{bmatrix} v_{0_0} & v_{0_1} & \cdots & v_{0_{d-1}} & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ v_{d_0} & v_{d_1} & \cdots & v_{d_{d-1}} & 1 \end{bmatrix} \tag{1}$$

where v_{k_j} denotes the j^{th} coordinate of the k^{th} vertex with $j,k \in \{0,1,2,\ldots,d\}$. In 3 dimensions, equation (2) gives six times the signed volume of the tetrahedron T_{abcd}.

$$6V\left(T_{abcd}\right) = \begin{vmatrix} a_0 & a_1 & a_2 & 1 \\ b_0 & b_1 & b_2 & 1 \\ c_0 & c_1 & c_2 & 1 \\ d_0 & d_1 & d_2 & 1 \end{vmatrix} = \begin{vmatrix} a_0 - d_0 & a_1 - d_1 & a_2 - d_2 \\ b_0 - d_0 & b_1 - d_1 & b_2 - d_2 \\ c_0 - d_0 & c_1 - d_1 & c_2 - d_2 \end{vmatrix} \tag{2}$$

This volume serves as the fundamental building block of the geometry routines. It is positive when *(a,b,c)* forms a counterclockwise circuit when viewed from an observation point located on the side of the plane defined by *(a,b,c)* which is opposite from *d*. Positive and negative volumes define the two states of the Boolean test while zero indicates that the four vertices are exactly coplanar. If the vertices are indeed coplanar, then the situation constitutes a "tie" which will be resolved with a general tie-breaking algorithm presented shortly. In applying this logical test to edge *ab* and triangle *(0,1,2)* in Figure 1, *ab* crosses the plane if and only if

(iff) the signed volumes T_{012a} and T_{012b} have opposite signs. Figure 2 presents a graphical look at the application of this test.

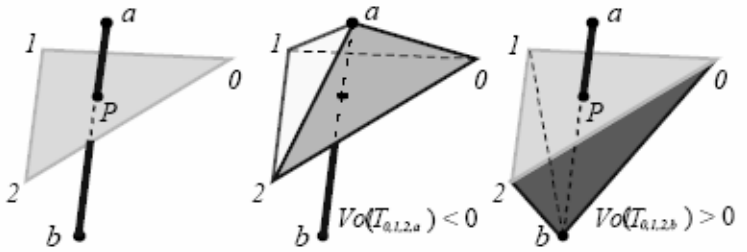


Figure 2: Boolean test to check if edge ab crosses the plane defined by triangle (0,1,2) through computation of two signed volumes.

With *a* and *b* established on opposite sides of the plane *(0,1,2)*, all that remains is to determine if *ab* pierces within the boundary of the triangle. This will be the case only if the three tetrahedra formed by connecting the end points of *ab* with the three vertices of the triangle *(0,1,2)* (taken two at a time) all have the same sign, that is:

$$\begin{aligned} &V(T_{a12b}) < 0 \wedge V(T_{a01b}) < 0 \wedge V(T_{a20b}) < 0 \quad \text{or} \\ &V(T_{a12b}) > 0 \wedge V(T_{a01b}) > 0 \wedge V(T_{a20b}) > 0 \end{aligned} \tag{3}$$

Figure 3 illustrates this test for the case where the three volumes are all positive.

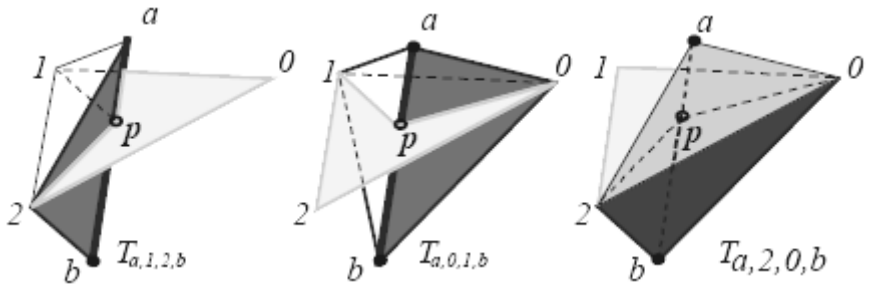


Figure 3: Boolean test for pierce of a line segment ab within the boundary of a triangle (0,1,2).

After determining the existence of all the segments which result from intersections between tri-tri pairs and connecting a linked list of all such segments to the triangles that intersect to produce them, all that remains is to actually compute the locations of the pierce points. This is accomplished by using a parametric representation of each intersected triangle and the edge, which pierces it. The technique is a straightforward three dimensional generalization of the 2D method presented in “Computational Geometry in C” (O'Rourke, 1998).

3. The Dimensional Model: A Framework to Distinguish Spatial Relationships

Billen R., Zlatanova S., Mathonet P., Boniver F. (2002) The Dimensional Model: A Framework to Distinguish Spatial Relationships, Advances in Spatial Data Handling, Springer-Verlag, Heidelberg, pp. 286-287, 290-293, 294-296.

3.1 Abstract

A number of frameworks use topology as a basic mechanism to define spatial relationships. In this research, topology is not suitable. In this paper, a new framework for representing spatial relationships – the *Dimensional Model* – is introduced. The model addresses a substantial group of spatial relationships and provides a flexible framework to consider either generalised or specialised types of associations. This framework will be adopted in this research to describe spatial relations between two triangles.

3.2 An Order Formula for Convex Bodies (pp. 286-287)

Affine Subspaces and Convex Sets

Geographers and mathematicians use co-ordinates to describe location in space. Whenever d is a positive integer, we denote by R^d the set of tuples $(\alpha_1,\ldots,\alpha_d)$ of real numbers $\alpha_1,\ldots,\alpha_d$. As Euclidean vector (or affine) space, R^d is the natural framework in which geometry can formally be studied. It is also the first example of Euclidean topological space.

A subset A of R^d is an *affine subspace* if, for any distinct points x,y belonging to A, the (infinite) straight line defined by x and y lies in A. Points, straight lines, planes, and R^3 itself are the only affine subspace of R^3. Their respective *dimensions* are 0, 1, 2 and 3. An affine subspace of dimension d-1 of R^d is named *hyperplane*. For instance, the hyperplanes are merely straight lines in R^2 and planes in R^3. A hyperplane divides the whole space in two regions, called *halfspaces*.

A subset A of R^d is *convex* if, for any two points x,y belonging to A, the segment $[x,y]$ lies in A. A *supporting hyperplane M* of a convex set C is a hyperplane such that

- C is included in one of the halfspaces defined by M,
- $M \cap C \neq \varnothing$.

Figure 4 shows examples of convex sets and supporting hyperplanes in R^2 and R^3.

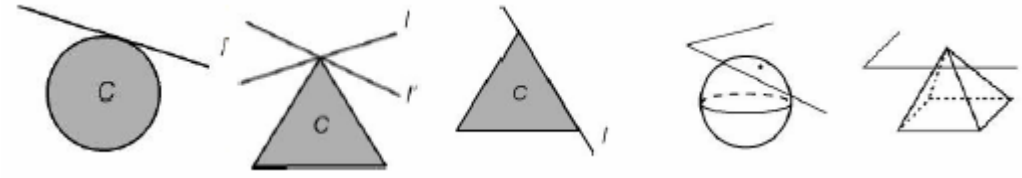


Figure 4: Examples of closed convex sets and hyperplanes

Order of Points in Closed Convex Set

Let C be a convex set, closed with respect to the Euclidean topology. Each point of C has an *order*, and can be stated as follow. Let C be a closed convex set in R^d and $x \in C$. The order of x in C, denoted by $o(x,C)$, is the dimension of the intersection of all supporting hyperplanes containing x.

In particular, if no supporting hyperplanes contains x, then x has order d. One can prove that those points with order d are exactly the interior points of C with respect to the Euclidean topology.

Figure 5 shows points with various orders in a triangle, a drop, and a segment. For instance, the top point of the triangle has order 0. Indeed, infinitely many supporting hyperplanes (only two are represented) go through it and intersect in that point itself. Figure 6 shows examples in R^3.

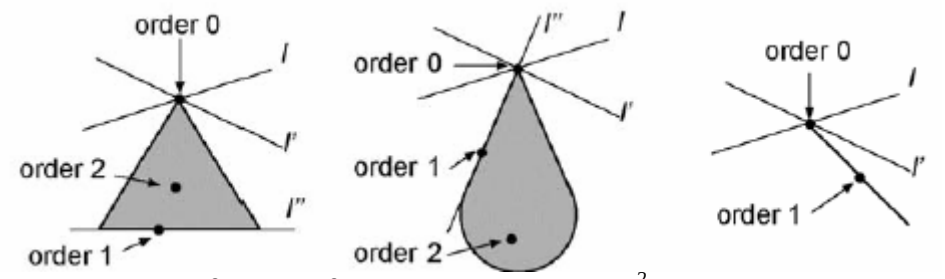


Figure 5: Order of points of closed convex sets in R^2 (with some hyperplanes)

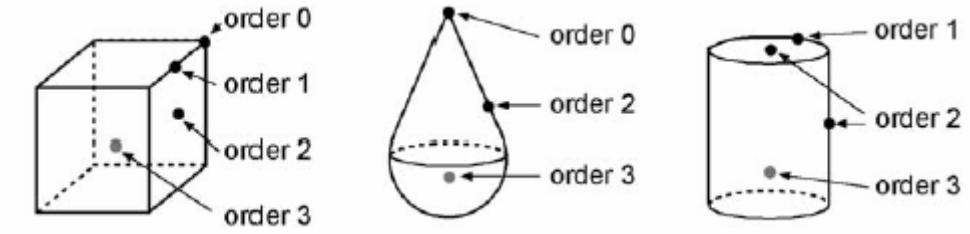


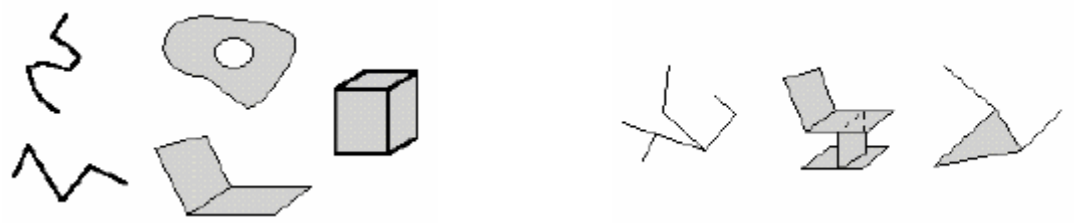
Figure 6: Order of points of closed convex sets in R^3 (without hyperplanes)

3.3 The Dimensional Model (pp. 290-293)

The Dimensional Model (DM) is a (conceptual) framework to describe both spatial objects and spatial relationships. The spatial objects are composed by *dimensional elements,* which are based on order's points of object. The spatial relationships between spatial objects are described in terms of *dimensional relationships*, i.e. relationships that exist between the dimensional elements of the objects.

Spatial objects in DM

In our model, *a simple spatial object of dimension d is equivalent to a topological d-manifold*. They are called simple because it is possible to apply directly the order formula to them, and therefore determine their dimensional elements. We also define a complex spatial object (as a combination of simple spatial object), which will not be discussed here (see Figure 7).



Simple spatial objects (manifold) Complex spatial objects (aggregation of simple objects)

Figure 7: Examples of spatial objects

Dimensional elements of DM

The *dimensional elements* are associated with different parts (or points) of a spatial object according to their order.

1. The α-dimensional element (denoted αD-element) of a spatial object C (which has at least dimension α), corresponds to the set of all the points (or parts) of C which have order 0 to α.
2. The αD-element of a spatial object C has an **extension** and may have a **limit**.
3. The extension is the subset of C formed by its points of order α, and the limit is the subset of C formed by its points of order 0 to order (α-1).

Thus, if the αD-element has a limit, this limit corresponds to a lower (α-1)D-element. The 0D-element does not have a limit by definition. Figure 8 illustrates the dimensional elements of a polygon. First, the order of all the points is determined.

This convex is composed of 2D, 1D and 0D-elements. The different extension and limits are also presented in Figure 8. In the case of an ellipse, the 1D-element does not have a limit. It should be noted that there is one and only one *x*D-element for this object.

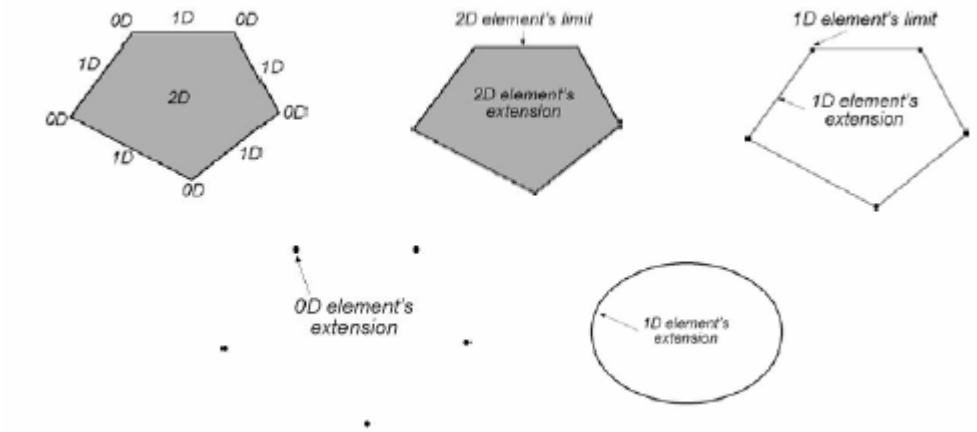


Figure 8: Order of points and dimensional elements of a polygon and of an ellipse

Dimensional Relationships

The *dimensional relationships* are defined as the relationships existing between dimensional elements. These relationships can either be total, partial or nonexistent, and are oriented (from one element to another one).

- A dimensional element is in **total relation** with another dimensional element if their intersection is equal to the first element, and if the intersection between their extensions is not empty.
- A dimensional element is in **partial relation** with another dimensional element if their intersection is not equal to the first element, and if the intersection between their extensions is not empty.
- A dimensional element is in **no relation (non-existent)** with another dimensional element if the intersection between their extensions is empty.

Figure 9 illustrates the three types of dimensional relationships for 2D-elements.

No relation (non-existent) Total relation Partial relation

Figure 9: The different types of dimensional relationships between two 2D-elements (from black element to grey element)

The Dimensional Model for Investigation of Spatial Relationships

The dimensional elements (with their limits and extensions) and the dimensional relationships (i.e. total, partial and non-existent) are the basic tools to decode spatial relationships. The spatial relationship between two objects can be expressed by the dimensional relationships that exist between the dimensional elements of both objects. For example, let us consider a polygon *A* (with 2D, 1D, 0D elements) and a line *B* (with 1D and 0D-elements). The dimensional relationships between the spatial object *A* and the spatial object *B* can be identified in the following sequence: first, check the dimensional relationship between the 2D-element of *A* and all the dimensional elements of spatial object *B*; then, check the dimensional relationship between the 1D-element of *A* and all the dimensional elements of spatial object *B*, etc. The dimensional relationships between *B* and *A* can be found following the same approach. Three groups of dimensional relationships can be distinguished following this approach, i.e. the *simplified*, the *basic* and the *extended* relationships.

A dimensional relationship is coded *RnDy*, using the notations *R* for relationships, *nD* for dimension of the element of the first object, and *y* dimension of the element of the second object. R2D1 represents the dimensional relationships between the 2D-element of the first object and the 1D-element of the second object. Furthermore, a numeric code for the three types of dimensional relationships, i.e. 0 for non-existent, 1 for total and 2 for partial, is specified.

The basic relationships. This group contains all the relationships between every possible combination of dimensional elements. For example, the spatial relationship between a polygon (considering 2D, 1D, 0D elements) and a line (considering 1D and 0D elements) can be expressed by basic dimensional relationships as follows:

R2D1	R2D0	R1D1	R1D0	R0D1	R0D0
{0,1,2}	{0,1,2}	{0,1,2}	{0,1,2}	{0,1,2}	{0,1,2}

where 0, 1, 2 correspond to the possible dimensional relationships, i.e. nonexistent, total and partial.

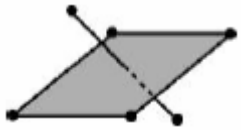
The extended relationships. The partial relation can be further investigated for the dimension of the intersection. For example, in R^3, a 2D-element and a 1Delement may have a 1D- or a 0D-intersection. If the intersection has the same dimension as the lowest dimensional element in the relation, it keeps the code 2 (e.g., if a 2D-element and a 1D-element have a 1D-intersection, it would be noted as R2D1 2). If the dimension of the intersection is just inferior, then it would have code 3 (a 0D-intersection in the example, R2D1 3). Our example of the extended dimensional relationships between a polygon and a line becomes:

R2D1	R2D0	R1D1	R1D0	R0D1	R0D0
{0,1,2,3}	{0,1,2}	{0,1,2,3}	{0,1,2}	{0,1,2}	{0,1,2}

The simplified dimensional relationships. In many cases the dimensional elements of the second object is not relevant. For example, it might be interesting to know if the 2D-element of a polygon has a relationship with another object independently of its dimensional elements. In such cases, some dimensional relationships can be aggregated. The complete aggregation rules will not be exposed here. Our example of the simplified dimensional relationship becomes:

R2D	R1D	R0D
{0,1,2}	{0,1,2}	{0,1,2}

Figure 10 illustrates how the spatial relationships between a polygon and a line are represented according to the different groups in the Dimensional Model.



Basic relationships

R2D1	R2D0	R1D1	R1D0	R0D1	R0D0
2	0	0	0	0	0

Extended relationships

R2D1	R2D0	R1D1	R1D0	R0D1	R0D0
3	0	0	0	0	0

Simplified relationships

R2D	R1D	R0D
2	0	0

Figure 10: The Dimensional Model applied to the relationship between a polygon and a line

3.4 Possible Spatial Relationships (pp. 294-296)

As mentioned, three groups of dimensional relationships can be used to express the spatial relationship between objects. Furthermore, one has the choice to take into consideration only relevant dimensional elements in a geographical perspective. For example, a particular geographical phenomena represented by a polygon may not need a distinction between the 1D-element and the 0D-element which form its border. In such a case, although the 0D-element exists in the object's definition, it would not be taken into account in the determination of the spatial relationship.

The number of potential relationships between two objects depends, with respect to the Dimensional Model, on: 1) the dimensional nature of the objects (given by dimensional elements), 2) the semantic dimension of the object (only the "relevant" dimensional elements from a semantic point of view) and 3) the group of dimensional relationships. Similarly to the 9-intersection model, only a small number of the theoretical relationships can be realised in reality. The same approach is adopted, i.e. elimination of impossible relationships by negative conditions. All the possible relationships between line-line, line-surface, line-body, surface-surface, surface-body and finally body-body have been established and studied for the different criterion mentioned above. Note that in this study, some elements have been simplified. For example the 0D element of a line corresponds only to its extremities (no broken lines). Table 1 portrays simplified, basic and extended relationships for all levels of dimensional relevancy.

Table 1: Possible relationships according to the Dimensional Model

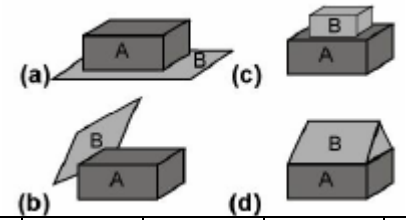
D element	*Dim. Rel.*	*Line-Line*	*Surface-line*	*Body-body*	*Surface-surface*	*Body-Line*	*Body-surface*
(n)D	S.	5	3	5	5	3	3
	B.	5	3	5	5	3	3
	E.	7	5	5	11	3	3
(n)D	S.	11	10	8	15	6	8
&(n-1)D	B.	**33**	**31**	**8**	**43**	**19**	**19**
	E.	61	?	15	?	43	48
(n)D	S.	-	?	?	?	19	?
&(n-1)D	B.	-	?	?	?	?	?
&(n-2)D	E.	-	?	?	?	?	?

(n)D &(n-1)D &(n-2)D &(n-3)D	S. B. E.	- - -	- - -	? ? ?	- - -	? ? ?	? ? ?

With Dim. Rel. = Dimensional relationship; S. = simplified; B. = basic; E. =extended

?	non determinate
-	impossible case
33	possible relationships according to the 9 intersection model

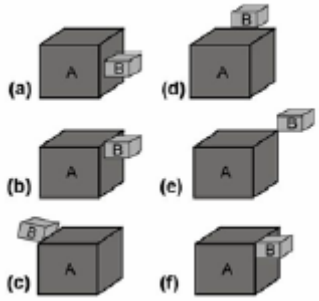
Considering the highest and the second highest dimensional element and using the basic relationship, the relationships reported by Zlatanova (2000) are found. They are shown in bold font in Table 1 (gray row). Most of the equivalent topological cases can be "refined" using some of the more complex criteria in the Dimensional Model (i.e. everything that is below the shaded line in Table 1). Further, more aggregated relationships can be found with simpler criterions (i.e. everything that is above this line). Figure 11 presents the extended dimensional solutions (0D elements are not taken into account) to the topological equivalence R095 and R287 (between a body *A* and a surface *B*).



	R3D2	R3D1	R2D2	R2D1	R1D2	R1D1
(a)	0	0	2	0	2	0
(b)	0	0	0	0	2	0
(c)	0	0	0	2	0	0
(d)	0	0	0	0	0	2

Figure 11: Dimensional relationships for, (a) and (b) topological equivalence R095, and (c) and (d) R287

Another interesting example concerns the "meet" relationship between two bodies. All the spatial situations in Figure 12 are equivalent to R287 according to the 9-intersection model, while all of them can be differentiated with the Dimensional model. Solutions are given only for cases *a, b, c* and *d*. The cases *e* and *f* need to consider the 0D-element.



	R3D3	R3D2	R3D1	R2D3	R2D2	R2D1	R1D3	R1D2	R1D1
(a)	0	0	0	0	2	2	0	0	0
(b)	0	0	0	0	2	2	0	0	2
(c)	0	0	0	0	0	0	0	2	2
(d)	0	0	0	0	0	0	0	0	2

Figure 12: Dimensional relationships for topological equivalence (R287)

4. Finite-element mesh generation from constructive-solid-geometry models

Boender E., Bronsvoort W.F., Post F.H. (1994) Finite-element mesh generation from constructive-solid-geometry models, Butterworth-Heinemann Ltd, Computer-Aided Design Volume 26 Number 5 May 1994.

4.1 Abstract

This paper discusses automatic finite element (FE) mesh generation from a constructive solid geometry (CSG) model of a 3D solid object with curved faces. The derivation of a mesh from a CSG model thus proceeds in two steps. The first is boundary evaluation of the CSG model, which entails its conversion into a B-rep by computation of the boundary elements of the object. The second step is the generation of a tetrahedral mesh from this B-rep. In the B-rep, faces are represented as trimmed, rational, Bezier patches. All the computations on edges and faces are reduced to 2D. The FE meshing of the B-rep is then performed by triangulation of the faces, followed by tetrahedralization of the interior of the solid. In this paper only some examples of triangulations are useful.

5. Geometry and Topology for Mesh Generation

Edelsbrunner H. (2001) Geometry and Topology for Mesh Generation, Cambridge University Press, Cambridge, pp. 44-46, 59-60, ISBN 0521793092.

5.1 Abstract

The book combines topics in mathematics (geometry and topology), computer science (algorithms), and engineering (mesh generation). Mesh generation is a topic in which a meaningful combination of these different approaches to problem solving is inevitable. The book develops methods from both areas that are amenable to combination and explains recent breakthrough solutions to meshing that fit into this category.

5.2 Simplicial complexes (pp. 44-46)

The book uses simplicial complexes as the fundamental tool to model geometric shapes and spaces. Simplicial complexes generalize and formalize the somewhat loose geometric notations of a triangulation. Because of their combinatorial nature, simplicial complexes are perfect data structures for geometric modelling algorithms.

Simplices

A finite collection of points is *affinely independent* if no affine space of dimension *i* contains more than *i* + 1 of the points, and this is true for every *i*. A k-simplex is the convex hull of a collection of k + 1 affinely independent points, $\sigma = \operatorname{conv} S$. The dimension of σ is $\dim \sigma = k$. In $\mathbb{R}^d$, the largest number of affinely independent points is *d* + 1, and there are simplices of dimension -1,0,…,*d*. The (-1)-simplex is the empty set. The convex hull of any subset $T \subseteq S$ is again a simplex.

Simplicial complexes

A *simplicial complex* is the collection of faces of a finite number of simplices, any two of which are either disjoint or meet in a common face. A *subcomplex* is a subset that is a simplicial complex itself.

5.3 Orientability (pp. 59-60)

Manifolds with or without boundary can be either orientable or non-orientable. The distinction is a global property that cannot be observed locally. Intuitively, we can imagine a *(k+1)*-dimensional ant walking on the *k*-manifold (or in this case on the *k*-simplex). At any moment, the ant is on one side of the local neighbourhood with which it is in contact. The manifold is *non-orientable* if there is a walk that brings the ant back to the same neighbourhood but now on the other side, ant it is *orientable* if no such path exists.

6. Delaunay Triangulation and Meshing

George, P. and Borouchaki, H. (1998) Delaunay Triangulation and Meshing, Application to Finite Elements, Hermes, Paris, pp. 5-11, 13-14.

6.1 Abstract

Automatic mesh generation is of the utmost importance in various engineering domains. The purpose of this book is to provide a comprehensive description of the meshing techniques based on the so-called Delauney triangulation. Both theoretical and computational aspects are discussed from a very practical point of view. The book first addresses triangulation construction methods, then discusses classical mesh generation techniques along with more advanced topics. Key issues for mesh adaptation are also highlighted.

6.2 Triangle and tetrahedron (pp. 5-11)

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

$$K = (P_1, P_2, P_3). \tag{4}$$

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

$$S_K = \frac{1}{2}\begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix} \tag{5}$$

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

$$K = (P_1, P_2, P_3, P_4). \tag{6}$$

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:

$$V_k = \frac{1}{6}\begin{vmatrix} x_1 & y_1 & z_1 & 1 \\ x_2 & y_2 & z_2 & 1 \\ x_3 & y_3 & z_3 & 1 \\ x_4 & y_4 & z_4 & 1 \end{vmatrix} \tag{7}$$

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.

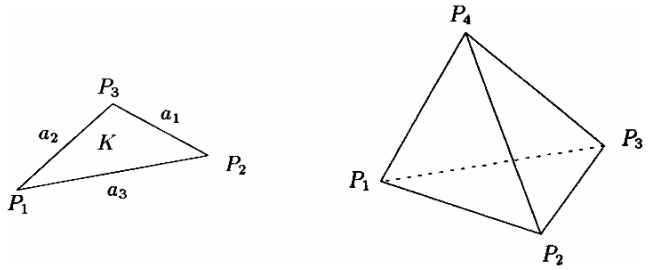


Figure 13: A triangle (left) and a tetrahedron (right)

6.3 Triangulation and tetrahedralization (pp. 13-14)

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

$$\sum_{i=1}^{n} \lambda_i P_i \tag{8}$$

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 simplicial 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.

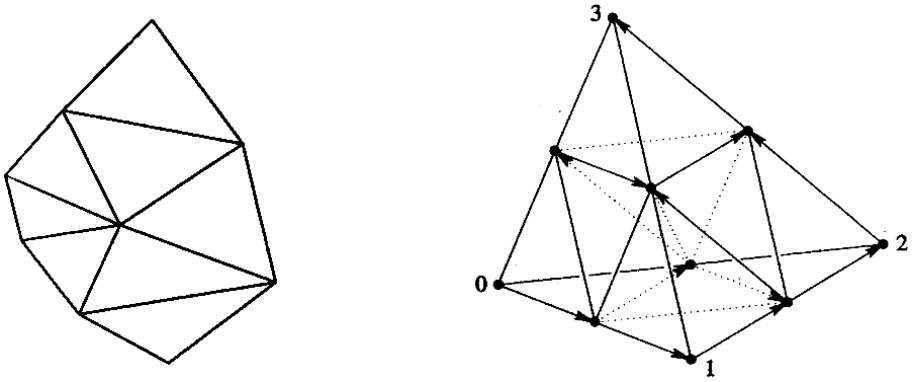


Figure 14: A triangulation (left) and a tetrahedralization (right)

6.4 Valid triangulation (pp. 14)

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.*

7. Handbook of Discrete and Computational Geometry

Goodman J. E. and O'Rourke J. (1997) Handbook of Discrete and Computational Geometry, Boca Raton, FL: CRC Press, pp. 377, 413, 475.

7.1 Abstract

The second edition of the Handbook of Discrete and Computational Geometry is a thoroughly revised version of the bestselling first edition. With the addition of 500 pages and 14 new chapters covering topics such as geometric graphs, collision detection, clustering, applications of computational geometry, and statistical applications, this is a significant update. This edition includes expanded coverage on the topics of mesh generation in two and three dimensions, aspect graphs, center points, and probabilistic roadmap algorithms. It also features new results on solutions of the Kepler conjecture, and honeycomb conjecture, new bounds on k-sets, and new results on face numbers of polytopes.

The book provides a one-stop reference both for researchers in geometry and geometric computing and for professionals who use geometric tools in their work. The book covers a broad range of topics in discrete and computational geometry as well as numerous applications. The book presents results in the forms of theorems, algorithms, and tables, and the book addresses many important new developments in the field, including solution of the Kepler conjecture, results on the 2-center problem, new bounds on k-sets and geometric permutations, and new art gallery theorems.

7.2 Definition of a triangulation (pp. 413)

A triangulation is a partition of a geometric domain, such as a point set, polygon, or polyhedron, into simplices that meet only at shared faces.

7.3 Definition of Delauney triangulation (pp. 377)

The Delauney triangulation is the unique triangulation of a set of sites such that the circumsphere of each full-dimensional simplex has no sites in its interior.

7.4 Hidden surface removal (pp. 475)

“Hidden surface removal” is one of the key problems in computer graphics, and has been the focus of intense research for two decades. The typical problem instance is a collection of (planar) polygons in space, from which the view from $z = \infty$ must be constructed.

8. VWO Wiskunde B Samengevat

Keemink N.C. (1993) VWO Wiskunde B Samengevat, schematisch overzicht van de examenstof, uitgeverij Onderwijspers BV Leiden, pp. 85-88.

8.1 Abstract

This book gives a summary of all the material of mathematics for the A level examination. In this research it will be used as a theoretical analytical geometry background.

8.2 Method to explore the relation of an edge and a face (pp. 85-86)

Given: edge $E(\boldsymbol{a},\boldsymbol{b})$ and face $F(\boldsymbol{c},\boldsymbol{d},\boldsymbol{e})$:

$$E(x,y,z)=(a_x,a_y,a_z)+\lambda(b_x-a_x,b_y-a_y,b_z-a_z) \tag{9}$$

$$F(x,y,z)=(c_x,c_y,c_z)+\upsilon(d_x-c_x,d_y-c_y,d_z-c_z)+\tau(e_x-c_x,e_y-c_y,e_z-c_z) \tag{10}$$

Determine normal of the face F:

$$\boldsymbol{n}\cdot(d_x-c_x,d_y-c_y,d_z-c_z)=0\wedge\boldsymbol{n}\cdot(e_x-c_x,e_y-c_y,e_z-c_z)=0\Rightarrow$$

$$(a,b,c)\cdot(d_x-c_x,d_y-c_y,d_z-c_z)=0\wedge(a,b,c)\cdot(e_x-c_x,e_y-c_y,e_z-c_z)\Rightarrow \tag{11}$$

$$\begin{cases}a(d_x-c_x)+b(d_y-c_y)+c(d_z-c_z)=0\\ a(e_x-c_x)+b(e_y-c_y)+c(e_z-c_z)=0\end{cases}\Rightarrow\boldsymbol{n}=\left[n_x(a,b,c),n_y(a,b,c),n_z(a,b,c)\right]$$

choose $a=0,b=0,c=0$ and fill in for $(c_x,c_y,c_z)\Rightarrow F:n_xc_x+n_yc_y+n_zc_z=f$ (12)

$$E:\begin{cases}x=a_x+\lambda(b_x-a_x)\\ y=a_y+\lambda(b_y-a_y)\\ z=a_z+\lambda(b_z-a_z)\end{cases}\text{ substitute in }F:n_xx+n_yy+n_zz=f\text{ gives} \tag{13}$$

$$F:n_x(a_x+\lambda(b_x-a_x))+n_y(a_y+\lambda(b_y-a_y))+n_z(a_z+\lambda(b_z-a_z))=f\Leftrightarrow$$

$$\begin{cases}\lambda\in\mathbb{R}\Rightarrow E\text{ is on }F\\ \lambda\in\varnothing\Rightarrow E\,//\,F\\ \text{one solution for }\lambda\Rightarrow E\text{ and }F\text{ intersect in theory}\end{cases} \tag{14}$$

$$\text{if and only if, }(x,y,z)\Rightarrow\begin{cases}a_x\le x\le b_x\\ a_y\le y\le b_y\\ a_z\le z\le b_z\end{cases}\text{ and }(x,y,z)\Rightarrow\begin{cases}\min(c_x,d_x,e_x)\le x\le\max(c_x,d_x,e_x)\\ \min(c_y,d_y,e_y)\le y\le\max(c_y,d_y,e_y)\\ \min(c_z,d_z,e_z)\le z\le\max(c_z,d_z,e_z)\end{cases} \tag{15}$$

8.3 Method to explore the relation of two faces (pp. 87-88)

Almost the same as above, but given two faces $F_1(\boldsymbol{a},\boldsymbol{b},\boldsymbol{c})$ and $F_2(\boldsymbol{d},\boldsymbol{e},\boldsymbol{f})$:

$$\begin{aligned}F_1(x_1,y_1,z_1)&=(a_x,a_y,a_z)+\lambda(b_x-a_x,b_y-a_y,b_z-a_z)+\upsilon(c_x-a_x,c_y-a_y,c_z-a_z)\\ F_2(x_2,y_2,z_2)&=(d_x,d_y,d_z)+\rho(e_x-d_x,e_y-d_y,e_z-d_z)+\tau(f_x-d_x,f_y-d_y,f_z-d_z)\end{aligned} \tag{16}$$

Determine normal of the faces:

$$\begin{cases} F_1 : n_{x1}x + n_{y1}y + n_{z1}z = g_1 \\ F_2 : n_{x2}x + n_{y2}y + n_{z2}z = g_2 \end{cases} \Leftrightarrow \begin{cases} x(y,z) \\ y(x,z) \\ z(x,y) \end{cases} \text{ substitute until } x \text{ or } y \text{ or } z \text{ is free.} \tag{17}$$

$$\text{Define free}(x,y,z) = \mu \Rightarrow \begin{cases} x(\mu) \\ y(\mu) \\ z(\mu) \end{cases} \Rightarrow \text{line of intersection } l : (x,y,z) = [x(\mu), y(\mu), z(\mu)] \tag{18}$$

$$\text{with}(x,y,z) \Rightarrow \begin{cases} \min(a_x, b_x, c_x, d_x, e_x, f_x) \le x \le \max(a_x, b_x, c_x, d_x, e_x, f_x) \\ \min(a_y, b_y, c_y, d_y, e_y, f_y) \le y \le \max(a_y, b_y, c_y, d_y, e_y, f_y) \\ \min(a_z, b_z, c_z, d_z, e_z, f_z) \le z \le \max(a_z, b_z, c_z, d_z, e_z, f_z) \end{cases} \tag{19}$$

9. Linear algebra and its applications (Second Edition)

Lay D.C. (1998) Linear algebra and its applications, second edition, University of Maryland, Addison-Wesley, pp. 133-137.

9.1 Abstract

This book contains the material for the examination of Linear Algebra at Delft University of Technology. In this research it will be used as a theoretical analytical geometry background, the function described is used in the predicates-file of TetGen.

9.2 Matrix Factorization (pp. 133-137)

A *factorization* of a matrix *A* is an equation that expresses *A* as a product of two or more matrices. Whereas matrix multiplication involves a synthesis of data (combining the effect of two or more linear transformations into a single matrix), matrix factorization is an analysis of data. In the language of computer science, the expression of *A* as a product amounts to a *pre-processing* of the data in *A*, organizing that data into two or more parts whose structures are more useful in some way, perhaps more accessible for computation.

The LU Factorization

The LU factorization, described below, is motivated by the fairly common industrial an business problem of solving a sequence of equations, all with the same coefficient matrix:

$$A\boldsymbol{x} = \boldsymbol{b}_1, A\boldsymbol{x} = \boldsymbol{b}_2, \ldots, A\boldsymbol{x} = \boldsymbol{b}_p \tag{20}$$

When *A* is invertible, one could compute A^{-1} and then compute $A^{-1}\boldsymbol{b}_1$, $A^{-1}\boldsymbol{b}_2$, and so on. However, in modern practice, the first equation in (20) is solved by row reduction, an LU factorization of *A* is obtained at the same time. Thereafter, the remaining equations in (20) are solved with the LU factorization.

At first, assume *A* is an $m \times n$ matrix that can be row reduced to echelon form, *without row interchanges*. Then A can be written in the form $A = LU$, where *L* is an $m \times m$ lower triangular matrix with 1's on the diagonal and *U* is an $m \times n$ echelon form of *A*. For instance see equation (21). Such a factorization is called an **LU factorization** of *A*. The matrix *L* is invertible and is called a *unit* lower triangular matrix.

$$A = \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 \\ * & 1 & 0 & 0 \\ * & * & 1 & 0 \\ * & * & * & 1 \end{bmatrix}}_{L} \underbrace{\begin{bmatrix} \blacksquare & * & * & * & * \\ 0 & \blacksquare & * & * & * \\ 0 & 0 & 0 & \blacksquare & * \\ 0 & 0 & 0 & 0 & \blacksquare \end{bmatrix}}_{U} \tag{21}$$

Before studying how to construct *L* and *U*, we should look at why they are so useful. When $A = LU$, the equation $A\boldsymbol{x} = \boldsymbol{b}$ can be written as $L(U\boldsymbol{x}) = \boldsymbol{b}$. Writing $\boldsymbol{y}$ for $U\boldsymbol{x}$, we can find $\boldsymbol{x}$ by solving the pair of equations.

$$\begin{aligned} L\boldsymbol{y} &= \boldsymbol{b} \\ U\boldsymbol{x} &= \boldsymbol{y} \end{aligned} \tag{22}$$

First solve $L\boldsymbol{y} = \boldsymbol{b}$ for $\boldsymbol{y}$ and then solve $U\boldsymbol{x} = \boldsymbol{y}$ for $\boldsymbol{x}$. Each equation is easy to solve because *L* and *U* are triangular.

Example 1 It can be verified that

$$A=\begin{bmatrix} 3 & -7 & -2 & 2 \\ -3 & 5 & 1 & 0 \\ 6 & -4 & 0 & -5 \\ -9 & 5 & -5 & 12 \end{bmatrix}=\begin{bmatrix} 1 & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 \\ 2 & -5 & 1 & 0 \\ -3 & 8 & 3 & 1 \end{bmatrix}\begin{bmatrix} 3 & -7 & -2 & 2 \\ 0 & -2 & -1 & 2 \\ 0 & 0 & -1 & 1 \\ 0 & 0 & 0 & -1 \end{bmatrix}=LU \tag{23}$$

Use this LU factorization of A to solve $A\boldsymbol{x}=\boldsymbol{b}$, where $\boldsymbol{b}=\begin{bmatrix} -9 \\ 5 \\ 7 \\ 11 \end{bmatrix}$.

Solution The solution of $L\boldsymbol{y}=\boldsymbol{b}$ needs only 6 multiplications and 6 additions, because the arithmetic takes place only in column 5. (The zeros below each pivot in L are created automatically by our choice of row operations.)

$$\begin{bmatrix} L & \boldsymbol{b} \end{bmatrix}=\begin{bmatrix} 1 & 0 & 0 & 0 & -9 \\ -1 & 1 & 0 & 0 & 5 \\ 2 & -5 & 1 & 0 & 7 \\ -3 & 8 & 3 & 1 & 11 \end{bmatrix}\sim\begin{bmatrix} 1 & 0 & 0 & 0 & -9 \\ 0 & 1 & 0 & 0 & -4 \\ 0 & 0 & 1 & 0 & 5 \\ 0 & 0 & 0 & 1 & 1 \end{bmatrix}=\begin{bmatrix} I & \boldsymbol{y} \end{bmatrix} \tag{24}$$

Then, for $U\boldsymbol{x}=\boldsymbol{y}$, the "backwards" phase of row reduction requires 4 divisions, 6 multiplications, and 6 additions. (For instance, creating zeros in column 4 of $\begin{bmatrix} U & \boldsymbol{y} \end{bmatrix}$ requires 1 division in row 4 and 3 multiplication-addition pairs to add multiples of row 4 to the rows above.)

$$\begin{bmatrix} U & \boldsymbol{y} \end{bmatrix}=\begin{bmatrix} 3 & -7 & -2 & 2 & -9 \\ 0 & -2 & -1 & 2 & -4 \\ 0 & 0 & -1 & 1 & 5 \\ 0 & 0 & 0 & -1 & 1 \end{bmatrix}\sim\begin{bmatrix} 1 & 0 & 0 & 0 & 3 \\ 0 & 1 & 0 & 0 & 4 \\ 0 & 0 & 1 & 0 & -6 \\ 0 & 0 & 0 & 1 & -1 \end{bmatrix},\ \boldsymbol{x}=\begin{bmatrix} 3 \\ 4 \\ -6 \\ -1 \end{bmatrix} \tag{25}$$

To find $\boldsymbol{x}$ requires 28 arithmetic operations, excluding the cost of finding L and U. In contrast, row reduction of $\begin{bmatrix} A & \boldsymbol{b} \end{bmatrix}$ to $\begin{bmatrix} I & \boldsymbol{x} \end{bmatrix}$ takes 62 operations. The computational efficiency of the LU factorization depends on knowing L and U. The next algorithm shows that the row reduction of A to an echelon form U amounts to an LU factorization because it produces L with essentially no extra work. After the first row reduction, L and U are available for solving additional equations whose coefficient matrix is A.

An LU Factorization Algorithm

Suppose A can be reduced to an echelon form U without row interchanges. Then, since row scaling is not essential, A can be reduced to U with only row replacements, adding a multiple of one row to another row *below* it. In this case, there exist unit lower triangular elementary matrices $E_1,...,E_p$ such that

$$E_p\cdots E_1A=U \tag{26}$$

Then

$$A = \left(E_p \cdots E_1\right)^{-1} U = LU \tag{27}$$

Where

$$L = \left(E_p \cdots E_1\right)^{-1} \tag{28}$$

It can be shown that products and inverses of unit lower triangular matrices are also unit lower triangular. Thus L is unit lower triangular. Note that the row operations in (26), which reduce A to U, also reduce the L in (27) to I, because $E_p \cdots E_1 L = \left(E_p \cdots E_1\right)\left(E_p \cdots E_1\right)^{-1} = I$. This observation is the key to *constructing L*.

Thus, the algorithm for an LU factorization is:

1. Reduce A to an echelon form U by a sequence of row replacement operations, if possible.
2. Place entries in L such that the *same sequence of row operations* reduces L to I.

Step 1 is not always possible, but when it is, the argument above shows that an LU factorization exists. Example 2 will show how to implement step 2. By construction, L will satisfy

$$\left(E_p \cdots E_1\right) L = I \tag{29}$$

using the same $E_1, \ldots, E_p$ as in (21). Thus L will be invertible, by the Invertible Matrix Theorem, with $\left(E_p \cdots E_1\right)^{-1} = L$. From (21), $L^{-1}A = U$, and $A = LU$. So step 2 will produce an acceptable L.

Example 2 Find an LU factorization of

$$A = \begin{bmatrix} 2 & 4 & -1 & 5 & -2 \\ -4 & -5 & 3 & -8 & 1 \\ 2 & -5 & -4 & 1 & 8 \\ -6 & 0 & 7 & -3 & 1 \end{bmatrix} \tag{30}$$

Solution Since A has four rows, L should be 4×4. The first column of L is the first column of A divided by the top pivot entry:

$$L = \begin{bmatrix} 1 & 0 & 0 & 0 \\ -2 & 1 & 0 & 0 \\ 1 & & 1 & 0 \\ -3 & & & 1 \end{bmatrix} \tag{31}$$

Compare the first columns of A and L. *The row operations that create zeros in the first column of A will also create zeros in the first column of L*. We want this same correspondence

of row operations to hold for the rest of L, so we watch a row reduction of A to an echelon form U:

$$A=\begin{bmatrix} \boldsymbol{2} & 4 & -1 & 5 & -2 \\ -4 & -5 & 3 & -8 & 1 \\ 2 & -5 & -4 & 1 & 8 \\ -6 & 0 & 7 & -3 & 1 \end{bmatrix} \sim \begin{bmatrix} \boldsymbol{2} & 4 & -1 & 5 & -2 \\ 0 & \boldsymbol{3} & 1 & 2 & -3 \\ 0 & -9 & -3 & -4 & 10 \\ 0 & 12 & 4 & 12 & -5 \end{bmatrix} = A_1$$
$$\sim A_2 = \begin{bmatrix} \boldsymbol{2} & 4 & -1 & 5 & -2 \\ 0 & \boldsymbol{3} & 1 & 2 & -3 \\ 0 & 0 & 0 & \boldsymbol{2} & 1 \\ 0 & 0 & 0 & 4 & 7 \end{bmatrix} \sim \begin{bmatrix} \boldsymbol{2} & 4 & -1 & 5 & -2 \\ 0 & \boldsymbol{3} & 1 & 2 & -3 \\ 0 & 0 & 0 & \boldsymbol{2} & 1 \\ 0 & 0 & 0 & 0 & \boldsymbol{5} \end{bmatrix} = U \tag{32}$$

The bold-italic entries above determine the row reduction of A to U. At each pivot column, divide the bold-italic entries by the pivot and place the result into L:

$$\begin{matrix} \begin{bmatrix} 2 \\ -4 \\ 2 \\ -6 \end{bmatrix} & \begin{bmatrix} 3 \\ -9 \\ 12 \end{bmatrix} & \begin{bmatrix} 2 \\ 4 \end{bmatrix} & [5] \\ \div 2 & \div 3 & \div 2 & \div 5 \\ \downarrow & \downarrow & \downarrow & \downarrow \end{matrix}$$
$$\begin{bmatrix} 1 & & & \\ -2 & 1 & & \\ 1 & -3 & 1 & \\ -3 & 4 & 2 & 1 \end{bmatrix}, \text{ and } L = \begin{bmatrix} 1 & 0 & 0 & 0 \\ -2 & 1 & 0 & 0 \\ 1 & -3 & 1 & 0 \\ -3 & 4 & 2 & 1 \end{bmatrix} \tag{33}$$

An easy calculation verifies that this L and U satisfy $A = LU$.

10. Computational Geometry in C (Second Edition)

O'Rourke J. (1998) Computational Geometry in C (Second Edition), Cambridge University Press, Cambridge, see http://cs.smith.edu/~orourke/, pp. 220-238.

10.1 Abstract

This is the newly revised and expanded edition of the popular introduction to the design and implementation of geometry algorithms arising in areas such as computer graphics, robotics, and engineering design. The second edition contains material on several new topics, such as randomized algorithms for polygon triangulation, planar point location, 3D convex hull construction, intersection algorithms for ray-segment and ray-triangle, and point-in-polyhedron. A new "Sources" chapter points to supplemental literature for readers needing more information on any topic. A novel aspect is the inclusion of working C code for many of the algorithms, with discussion of practical implementation issues. The self-contained treatment presumes only an elementary knowledge of mathematics, but reaches topics on the frontier of current research, making it a useful reference for practitioners at all levels. The code in this new edition is significantly improved from the first edition, and four new routines are included. Java versions for this new edition are also available.

The seventh chapter of the book (O'Rourke, 1998) examines several problems that can be loosely classified as involving search or intersection (or both). This is a vast, well-developed topic, and O'Rourke makes no attempt at systematic coverage. The chapter starts with to constant-time computations: intersecting two segments (lines) and intersecting a segment with a triangle. Implementations are present for both tasks. These two computations will be used in this research. In the code of Si (2004) only the second step is implemented, it is my job to implement the first step too.

10.2 Segment-segment intersection (pp. 220-226)

In the first chapter of the book (O'Rourke, 1998) there is spent some time developing a code that detects intersection between two segments for use in triangulation, there is never bothered to compute the point of intersection. It was not needed in the triangulation algorithm, and it would have forced them to leave the comfortable world of integer coordinates. For many applications, however, the floating-point coordinates of the point of intersection are needed. It is not too difficult to compute the intersection point (although there are potential pitfalls), and the necessary floating-point calculations are not as problematical here as they sometimes are. In this section the code for this task is developed (see O'Rourke, 1998), I will only report the resulting code.

Code 1: SegSegInt

```
/*-------------------------------------------------------------------
SegSegInt: Finds the point of intersection p between two closed
segments ab and cd.  Returns p and a char with the following meaning:
   'v': An endpoint (vertex) of one segment is on the other segment,
        but 'e' doesn't hold.
   '1': The segments intersect properly (i.e., they share a point and
        neither 'v' nor 'e' holds).
   '0': The segments do not intersect (i.e., they share no points).
Note that two collinear segments that share just one point, an endpoint
of each, returns 'e' rather than 'v' as one might expect.
-------------------------------------------------------------------*/
char    SegSegInt( tPointi a, tPointi b, tPointi c, tPointi d, tPointd p )
{
   double  s, t;       /* The two parameters of the parametric eqns. */
   double num, denom;  /* Numerator and denoninator of equations. */
```

```
   char code = '?';    /* Return char characterizing intersection. */

   denom = a[X] * (double)( d[Y] - c[Y] ) +
           b[X] * (double)( c[Y] - d[Y] ) +
           d[X] * (double)( b[Y] - a[Y] ) +
           c[X] * (double)( a[Y] - b[Y] );

   /* If denom is zero, then segments are parallel: handle separately. */
   if (denom == 0.0)
      return  ParallelInt(a, b, c, d, p);

   num =    a[X] * (double)( d[Y] - c[Y] ) +
            c[X] * (double)( a[Y] - d[Y] ) +
            d[X] * (double)( c[Y] - a[Y] );
   if ( (num == 0.0) || (num == denom) ) code = 'v';
   s = num / denom;
   printf("num=%lf, denom=%lf, s=%lf\n", num, denom, s);

   num = -( a[X] * (double)( c[Y] - b[Y] ) +
            b[X] * (double)( a[Y] - c[Y] ) +
            c[X] * (double)( b[Y] - a[Y] ) );
   if ( (num == 0.0) || (num == denom) ) code = 'v';
   t = num / denom;
   printf("num=%lf, denom=%lf, t=%lf\n", num, denom, t);

   if      ( (0.0 < s) && (s < 1.0) &&
             (0.0 < t) && (t < 1.0) )
     code = '1';
   else if ( (0.0 > s) || (s > 1.0) ||
             (0.0 > t) || (t > 1.0) )
     code = '0';

   p[X] = a[X] + s * ( b[X] - a[X] );
   p[Y] = a[Y] + s * ( b[Y] - a[Y] );

   return code;
}
```

With this code it is not possible to determine if segments are overlapping, therefore below code has to be added.

Code 2: ParallelInt

```
/*---------------------------------------------------------------------
ParallelInt: Finds if the segments ab and cd collinearly overlap. Returns a
char with the following meaning:
   'e': The segments collinearly overlap, sharing a point.
   '0': The segments do not intersect (i.e., they share no points).
Note that two collinear segments that share just one point, an endpoint
of each, returns 'e' rather than 'v' as one might expect.
---------------------------------------------------------------------*/
char    ParallelInt( tPointi a, tPointi b, tPointi c, tPointi d, tPointd p )
{
   if ( !Collinear( a, b, c) )
      return '0';

   if ( Between( a, b, c ) ) {
      Assigndi( p, c );
      return 'e';
   }
   if ( Between( a, b, d ) ) {
      Assigndi( p, d );
      return 'e';
   }
   if ( Between( c, d, a ) ) {
      Assigndi( p, a );
      return 'e';
   }
```

```
   if ( Between( c, d, b ) ) {
      Assigndi( p, b );
      return 'e';
   }
   return '0';
}
void   Assigndi( tPointd p, tPointi a )
{
   int i;
   for ( i = 0; i < DIM; i++ )
      p[i] = a[i];
}

/*---------------------------------------------------------------------
Returns TRUE iff point c lies on the closed segement ab.
Assumes it is already known that abc are collinear.
---------------------------------------------------------------------*/
bool   Between( tPointi a, tPointi b, tPointi c )
{
   tPointi      ba, ca;

   /* If ab not vertical, check betweenness on x; else on y. */
   if ( a[X] != b[X] )
      return ((a[X] <= c[X]) && (c[X] <= b[X])) ||
             ((a[X] >= c[X]) && (c[X] >= b[X]));
   else
      return ((a[Y] <= c[Y]) && (c[Y] <= b[Y])) ||
             ((a[Y] >= c[Y]) && (c[Y] >= b[Y]));
}

Int    Collinear( tPointi a, tPointi b, tPointi c )
{
   return AreaSign( a, b, c ) == 0;
}
int    AreaSign( tPointi a, tPointi b, tPointi c )
{
    double area2;

    area2 = ( b[0] - a[0] ) * (double)( c[1] - a[1] ) -
            ( c[0] - a[0] ) * (double)( b[1] - a[1] );

    /* The area should be an integer. */
    if      ( area2 >  0.5 ) return  1;
    else if ( area2 < -0.5 ) return -1;
    else                     return  0;
}
```

10.3 Segment-triangle intersection (pp. 226-238)

Let us turn to the more difficult, but still ultimately straightforward, computation of the point of intersection between a segment and a triangle in three dimensions. In fact this is one of the most prevalent geometric computations performed today, because it is a key step in “ray tracing” used in computer graphic: finding the intersection between a light ray and a collection of polygons in space. O’Rourke will again use a parametric representation to derive the equations. Throughout he will let $T = \Delta abc$ be the triangle and qr the segment, where q is viewed as the originating (“query”) endpoint in case qr represents a ray and r is the “ray” endpoint. He assumed throughout that $r \neq q$, so the input segment has nonzero length. The computation process is divided in steps. Before one can really start, the triangles in space have to be defined.

Code 3: Type definitions for triangles in space

```
#define X 0
#define Y 1
#define Z 2
#define DIM 3                 /* Dimension of points */
typedef int tPointi[DIM];     /* Type integer point */
typedef double tPointd[DIM];  /* Type double point */
#define PMAX 10000            /* Max # of pts */
tPointi Vertices[PMAX];       /* All the points */
tPointi Faces[PMAX];          /* Each triangle face is 3 indices */

main()
{
   int V, F;

   V = ReadVertices();
   F = ReadFaces();
   /* Processing */
}
```

Segment-plane intersection

The first step is to determine if qr intersects the plane π containing T. The work is partitioned in two procedures, the first, `PlaneCoeff`, computes N (the Normal of the plane π) and D (if N is a vector of unit length, then D is the distance from the origin to π)

Code 4: PlaneCoeff

```
/*-------------------------------------------------------------------
Computes N & D and returns index m of largest component.
-------------------------------------------------------------------*/
int    PlaneCoeff( tPointi T, tPointd N, double *D )
{
   int i;
   double t;                  /* Temp storage */
   double biggest = 0.0;      /* Largest component of normal vector.*/
   int m = 0;                 /* Index of largest component. */

   NormalVec( Vertices[T[0]] , Vertices[T[1]] , Vertices[T[2]], N );
   *D = Dot( Vertices[T[0]], N );

   /* Find the largest component of N. */
   for ( i = 0; i < DIM; i++ ) {
      t = fabs(N[i] );
      if ( t > biggest ) {
         biggest = t;
         m = i;
      }
   }
   return m;
}
```

Code 5: Vector utility functions.

```
/*-------------------------------------------------------------------
Compute the cross product of (b-a)x(c-a) and place into N.
-------------------------------------------------------------------*/
void   NormalVec( tPointi a, tPointi b, tPointi c, tPointd N )
{
   N[X]=( c[Z] - a[Z] ) * ( b[Y] - a[Y] ) –
        ( b[Z] - a[Z] ) * ( c[Y] – a[Y] );
   N[Y]=( b[Z] - a[Z] ) * ( c[X] - a[X] ) –
        ( b[X] - a[X] ) * ( c[Z] - a[Z] );
   N[Z]=( b[X] - a[X] ) * ( c[Y] - a[Y] ) –
```

```
        ( b[Y] - a[Y] ) * ( c[X] - a[X] );
}

/*---------------------------------------------------------------------
Returns the dot product of the two input vectors.
---------------------------------------------------------------------*/
double Dot( tPointi a, tPointd b )
{
   int i;
   double sum = 0.0;

   for( i = 0; i < DIM; i++ )
      sum += a[i] * b[i];
   return sum;
}

/*---------------------------------------------------------------------
a - b ==> c.
---------------------------------------------------------------------*/
void   SubVec( tPointi a, tPointi b, tPointi c )
{
   int i;

   /* a-b => c. */
   for( i = 0; i < DIM; i++ )
      c[i] = a[i] - b[i];
}
```

O'Rourke now follows the convention established in the segment-segment intersection. The intersection procedure returns a code to classify the intersection.

Code 6: SegPlaneInt

```
/*---------------------------------------------------------------------
    'p': The segment lies wholly within the plane.
    'q': The q endpoint is on the plane (but not 'p').
    'r': The r endpoint is on the plane (but not 'p').
    '0': The segment lies strictly to one side or the other of the plane.
    '1': The segment intersects the plane, and 'p' does not hold.
---------------------------------------------------------------------*/
char   SegPlaneInt( tPointi T, tPointi q, tPointi r, tPointd p, int *m)
{
   tPointd N; double D;
   tPointi rq;
   double num, denom, t;
   int i;

   *m = PlaneCoeff( T, N, &D );
   num = D - Dot( q, N );
   SubVec( r, q, rq );
   denom = Dot( rq, N );

   if ( denom == 0.0 ) {      /* Segment is parallel to plane. */
      if ( num == 0.0)        /* q is on plane. */
         return 'p';
      else
         return '0';
   }
   else
      t = num / denom;

   for( i = 0; i < DIM; i++ )
      p[i] - q[i] + t * ( r[i] - q[i] );

   if ( (0.0 < t) && (t < 1.0) )
      return '1';
   else if ( num == 0.0 )     /* t == 0 */
      return 'q';
```

```
   else if ( num == denom)    /* t == 1 */
      return 'r';
   else return '0';
}
```

Segment-triangle classification

Now that point *p* of the intersection between the segment qr and the plane π containing the triangle T is determined, it only remains to classify the relationship between p and T: Is it inside or out, on a boundary, at a vertex? Although this may be a simple task, there are some subtleties. Only the resulting code is reported.

Code 7: InTri3D

```
/*-------------------------------------------------------------------
Assumption: p lies in the plane containing T. Returns a char:
  'V': the query point p coincides with a Vertex of triangle T.
  'E': the query point p is in the relative interior of an Edge of T.
  'F': the query point p is in the relative interior of a Face of T.
  '0': the query point p does not intersect (misses) triangle T.
-------------------------------------------------------------------*/
char    InTri3D( tPointi T, int m, tPointi p )
{
   int i;              /* Index for X,Y,Z */
   int j;              /* Index for X,Y */
   int k;              /* Index for triangle vertex */
   tPointi pp;         /* projected p */
   tPointi Tp[3];      /* projected T: Three new vertices */

   /* Project out coordinate m in both p and the triangular face */
   j = 0;
   for( i = 0; i < DIM; i ++ ) {
      if ( i !=  m ) { /* skip largest coordinate */
         pp[j] = p[i];
         for ( k = 0; k < 3; k++ )
            Tp[k][j] = Vertices[T[k]][i];
         j++;
      }
   }
   return ( InTri2D( Tp, pp ) );
}
```

Code 8: InTri2D

```
/*-------------------------------------------------------------------
compute three AreaSign() values for pp w.r.t. each edge of the face in 2D
-------------------------------------------------------------------*/
char    InTri2D( tPointi Tp[3], tPointi pp )
{
   int area0, area1, area2;

   area0 = AreaSign( pp, Tp[0], Tp[1] );
   area1 = AreaSign( pp, Tp[1], Tp[2] );
   area2 = AreaSign( pp, Tp[2], Tp[0] );

   if ( ( area0 == 0 ) && ( area1 > 0 ) && ( area2 > 0 )||
        ( area1 == 0 ) && ( area0 > 0 ) && ( area2 > 0 )||
        ( area2 == 0 ) && ( area0 > 0 ) && ( area1 > 0 ))
      return 'E';

   if ( ( area0 == 0 ) && ( area1 < 0 ) && ( area2 < 0 )||
        ( area1 == 0 ) && ( area0 < 0 ) && ( area2 < 0 )||
        ( area2 == 0 ) && ( area0 < 0 ) && ( area1 < 0 ))
      return 'E';
```

```
   if ( ( area0 > 0 ) && ( area1 > 0 ) && ( area2 > 0 )||
        ( area1 < 0 ) && ( area1 < 0 ) && ( area2 < 0 ))
      return 'F';

   if ( ( area0 == 0 ) && ( area1 == 0 ) && ( area2 == 0 ) )
      fprintf( stderr, "Error in InTriD\n" ),
         exit (EXIT_FAILURE);

   if ( ( area0 == 0 ) && ( area1 == 0 )||
        ( area0 == 0 ) && ( area2 == 0 )||
        ( area1 == 0 ) && ( area2 == 0 ) )
      return 'V';

   else
      return '0';
}
```

Code 9: SegTriCross

```
/*-------------------------------------------------------------------
The signed volumes of three tetrahedra are computed, determined
by the segment qr, and each edge of the triangle.
Returns a char:
   'v': the open segment includes a vertex of triangle T.
   'e': the open segment includes a point in the relative interior of an
        edge of triangle T.
   'f': the open segment includes a point in the relative interior of a
        face of triangle T.
   '0': the open segment does not intersect triangle T.
-------------------------------------------------------------------*/
char   SegTriCross( tPointi T, tPointi q, tPointi r )
{
   int vol0, vol1, vol2;

   vol0 = VolumeSign( q, Vertices [ T[0] ], Vertices[ T[1] ], r );
   vol1 = VolumeSign( q, Vertices [ T[1] ], Vertices[ T[2] ], r );
   vol2 = VolumeSign( q, Vertices [ T[2] ], Vertices[ T[0] ], r );

   /* Same sign: segment intersects interior of triangle. */
   if ( ( ( vol0 > 0 ) && ( vol1 > 0 ) && ( vol2 > 0 ) )||
        ( ( vol0 < 0 ) && ( vol1 < 0 ) && ( vol2 < 0 ) ) )
      return 'f';

   /* Opposite sign: no intersection between segment and triangle. */
   if ( ( ( vol0 > 0) || ( vol1 > 0 ) || ( vol2 > 0 ) ) &&
        ( ( vol0 < 0) || ( vol1 < 0 ) || ( vol2 < 0 ) ) )
      return '0';

   else if ( ( vol0 == 0 ) && ( vol1 == 0 ) && ( vol2 == 0 ) )
      fprintf( stderr, "Error 1 in SegTriCross\n" ),
      exit (EXIT_FAILURE);

   /* Two zeros: segment intersects vertex. */
   else if ( ( ( vol0 == 0 ) && ( vol1 == 0 ) ) ||
             ( ( vol0 == 0 ) && ( vol2 == 0 ) ) ||
             ( ( vol1 == 0 ) && ( vol2 == 0 ) ) )
      return 'v';

   /* One zero: segment intersects edge. */
   else if ( ( vol0 == 0 ) || ( vol1 == 0) || ( vol2 == 0 ) )
      return 'e';

   else
      fprintf( stderr, "Error 2 in SegTriCross\n" ),
      exit (EXIT_FAILURE);
}
```

Code 10: SegTriInt

```
char    SegTriInt( tPointi T, tPointi q, tPointi r, tPointd p )
{
   int code;
   int m;

   code = SegPlaneInt( T, q, r, p, &m );

   if ( code == 'q')
      return InTri3D( T, m, q );
   else if ( code == 'r')
      return InTri3D( T, m, r );
   else if ( code == 'p')
      return InPlane( T, m, q, r, p );
   else
      return SegTriCross( T, q, r );
   else /* Error */
      return code;
}

char    InPlane( tPointi T, int m, tPointi q, tPointi r, tPointd p)
{
    /* NOT IMPLEMENTED */
    return 'p';
}
```

Also needed are below codes, these are treated earlier in the book.

Code 11: AreaSign

```
/*---------------------------------------------------------------------
compute three AreaSign() values for pp w.r.t. each edge of the face in 2D.
---------------------------------------------------------------------*/
int     AreaSign( tPointi a, tPointi b, tPointi c )
{
    double area2;

    area2 = ( b[0] - a[0] ) * (double)( c[1] - a[1] ) -
            ( c[0] - a[0] ) * (double)( b[1] - a[1] );

    /* The area should be an integer. */
    if      ( area2 >  0.5 ) return  1;
    else if ( area2 < -0.5 ) return -1;
    else                     return  0;
}
```

Code 12: VolumeSign

```
int     VolumeSign( tPointi a, tPointi b, tPointi c, tPointi d )
{
   double vol;
   double ax, ay, az, bx, by, bz, cx, cy, cz, dx, dy, dz;
   double bxdx, bydy, bzdz, cxdx, cydy, czdz;

   ax = a[X];
   ay = a[Y];
   az = a[Z];
   bx = b[X];
   by = b[Y];
   bz = b[Z];
   cx = c[X];
   cy = c[Y];
   cz = c[Z];
```

```
    dx = d[X];
    dy = d[Y];
    dz = d[Z];

    bxdx=bx-dx;
    bydy=by-dy;
    bzdz=bz-dz;
    cxdx=cx-dx;
    cydy=cy-dy;
    czdz=cz-dz;
    vol =   (az-dz) * (bxdx*cydy - bydy*cxdx)
          + (ay-dy) * (bzdz*cxdx - bxdx*czdz)
          + (ax-dx) * (bydy*czdz - bzdz*cydy);


    /* The volume should be an integer. */
    if      ( vol > 0.5 )   return  1;
    else if ( vol < -0.5 )  return -1;
    else                    return  0;
}
```

11. About Invalid, Valid and Clean Polygons

Oosterom P., Quak W., Thijssen T. () About Invalid, Valid and Clean Polygons.

11.1 Abstract

Spatial models are often based on polygons both in 2D and 3D. Many Geo-ICT products support spatial data types, such as the polygon, based on the OpenGIS "Simple Feature Specification". OpenGIS and ISO have agreed to harmonize their specifications and standards. In this paper the relevant aspects related to polygons in these standards are discussed and several implementations are compared. A quite exhaustive set of test polygons (with holes) has been developed. The test results reveal significant differences in the implementations, which causes interoperability problems. Part of these differences can be explained by different interpretations (definitions) of the OpenGIS and ISO standards (do not have an equal polygon definition). Another part of these differences is due to typical implementations issues, such as alternative methods for handling tolerances. Based on these experiences the authors propose an unambiguous definition for polygons, which makes polygons again the stable foundation it is supposed to be in spatial modelling and analysis. Valid polygons are well defined, but as they may still cause problems during data transfer, also the concept of (valid) clean polygons is defined. This paper is not of great interest, because triangles are by definition already valid (clean).

12. Topological models for 3D spatial information systems

Pigot S. (1991) Topological models for 3D spatial information systems, In: D. M. Mark and D. White (Eds.), Auto Carto 10, Baltimore, MD, pp. 374-377, 381-387.

12.1 Abstract

The need for complex modelling and analysis of 3D data within a spatial information system (SIS) has been established in many fields While much of the data that is currently being modelled seems to require "soft-edge" data structures such as grids or rasters, the need for certain types of complex topological modelling and analysis is clear. Current plane topology models such as winged edge, widely used in computer aided design (CAD), are limited in the types of analysis that can be performed but useful because of their basis in the field of algebraic topology. This paper firstly reviews the neighbourhood structure provided by current plane topological models. It then describes the derivation of a fundamental set of binary topological relationships between simple spatial primitives of like topological dimension in 3D. It is intended that these relationships provide both a measure of modelling sufficiency and analytical ability in a spatial information system based on three dimensional neighbourhoods.

12.2 Theory (pp. 374-377)

What topological relationships may exist between abstract geometric primitives in Euclidean 3-space? To answer a detailed question about the nature and type of all topological relationships is an attempt to classify the types and situations of manifolds. This is possible for R^1 (1-space) and R^2 (2-space) however, R^3 (3-space) has a number of quite difficult and unexpected situations which make general classification very difficult. Fortunately, it is not necessary to attempt this. A number of assumptions about the nature of the relationships and the geometry of the n-cells involved can be made without limiting the power and application of the derived relationships. Specifically, only binary topological relationships between closed, connected (genus 0 – no internal holes) n-simplexes will be considered. The use of simplexes rather that cells is intuitive; simplicial complex theory is the starting point for the more generalized and advanced cell complex theory. Cells can be decomposed into simplexes in what is termed a simplicial decomposition, thus the results derived using simplicial complex theory can be generalized to cell complex theory via the decomposition.

Point-set topology (classical topology) provides a much more intuitive view of topological relationships. In the paper of Pigot (1991), point-set binary topological relationships between 1-simplexes in R^3, 2-simplexes in R^3 and 3-simplexes in R^3 are based on the consideration of the fundamental boundary, interior and exterior point-sets of any n-simplex in R^n. Additional point-sets are formed generically by embedding the n-simplex and its fundamental point-sets for R^n, within R^{n+1}. Consideration of the possible intersections of these point-sets with the boundary point-set of a second n-simplex then gives the fundamental topological relationships. The relationships are point-set topological relationships because they are derived from the intersection of these fundamental point-sets only.

In all of the following discussion, a 1-simplex is called an interval, a 2-simplex is called a face and a 3-simplex is called a volume.

Theoretical Background

All results used and derived in this section are for metric topological spaces since metric topological spaces are most commonly used for modelling purposes. Metric topological spaces are a subset of general topological spaces. An n-simplex in R^n divides R^n into three useful and intuitive point-sets, well known in point-set topology.

- 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*: $\partial C = C - C^{\circ}$
- Exterior set of an n-simplex *C*: complement of *C*

A simple and complete method can be found for finding all topological relationships between closed, connected n-simplexes. In the paper of Pigot (1991), a powerful and fundamental method is used which is based on the set intersection of the boundary, interior and exterior point-sets of an n-simplex n_1 and the boundary, interior and exterior sets of another n-simplex n_2 in R^n. In the paper, the generic term set is used in place of point-set. The theory can now be summarized in five steps as follows:

1. Formulate the boundary, interior and exterior sets of an n-simplex n_1 in R^n.
2. Derive basic relationships based on all possible set intersections of the boundary set of a second n-simplex n_2 and the interior, boundary and exterior sets of the n-simplex n_1 from step 1.
3. Consider the union of the interior, exterior and boundary sets of any n-simplex in R^n as an n-manifold equivalent to R^n with the definition of the open/closed properties of these sets strictly relative to R^n.
4. Disconnect R^{n+1} into two new open sets by choosing an embedding of R^n (created in step 3) in R^{n+1} such that the *n* orthogonal basis vectors of R^n are coincident with *n* of the *n+1* orthogonal basis vectors of R^{n+1}.
5. Derive additional relationships based on the possible set intersections of the boundary set of an n-simplex n_2 with the boundary, interior and exterior sets of a second n-simplex n_1, with the boundary set of n_2 intersecting either or both of the two new sets predicted in step 4.

12.3 Faces (2-simplexes) (pp. 381-387)

The boundary, interior and exterior sets of a face (or 2-simplex) all in R^2 are shown in Figure 15.

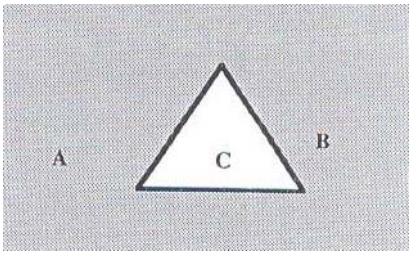


Figure 15: Exterior (A), Boundary (B) and Interior (C) point-sets of a face in R^2

Note that there is a single closed boundary set (B), a single open exterior set (A) and a single open interior set (C). The union of sets A, B and C is a 2-manifold equivalent to R^2. All possible binary topological relationships between faces in R^2 can then be derived from the possible set relationships between the boundary, interior and exterior sets A, B and C of a_1 and the boundary set *X* of a_2. e.g. if the boundary set *X* of a_2 is contained within the interior set *C*, then the face a_2 will he contained within a_1. The combinations matrix showing the possible relationships between the boundary of the face a_2 and the exterior, boundary and interior sets A, B and C of a_1 is shown in Table 2.

Table 2: Set intersection relationships between the boundary set of a_2 and the interior, exterior and boundary sets of a_1 in R^2.

Exterior A	X			X		X	X
Boundary A		X		X	X	X	
Interior A			X	X	X		X

Note that the seventh relationship in the last column of the table is not possible in R^2 because of the restriction to closed, connected faces. The six distinct relationships, their names and spatial interpretations are shown in Figure 16.

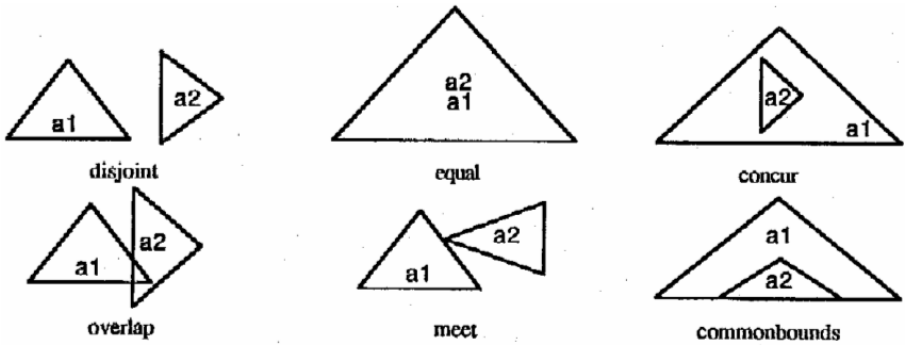


Figure 16: Six possible relationships between faces based on the intersection of the boundary set of face a_2 and the exterior, boundary and interior sets of face a_1 in R^2.

If we define the open/closed properties of these sets strictly relative to R^2 then these properties and the set relationships in R^2 are preserved when the 2-manifold (equivalent to R^2) formed by their union is embedded in R^3. If the embedding is chosen such that any two orthogonal basis vectors of R^2 are coincident to two of any three orthogonal basis vectors of R^3 then R^2 disconnects R^3 info two open sets with the third open set corresponding to R^2 itself. The situation is shown in Figure 17.

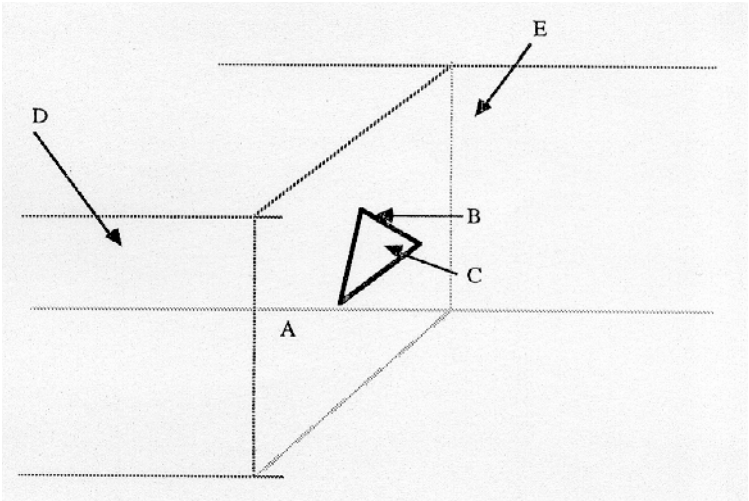


Figure 17: New point sets D & E obtained by embedding the union of the boundary (B), exterior (A) and interior (C) of a face a_1 in R^3 ($A \cup B \cup C = R^2$).

All possible binary topological relationships between faces in R^3 can be derived in the same way as for R^2, by considering the possible set relationships between boundary set of a face a_2

and the boundary, interior and exterior sets of the face a_1 plus the two new sets D and E which result from embedding R^2 in R^3. Since all set relationships derived for R^2 are preserved in R^3, only the combinations involving the new sets D and E will be considered.

By examination of Figure 17, the set relationships can be divided into two groups. The first group represents the situation where the boundary set X of a_2 is contained within the plane P formed from the union of the interior, exterior and boundary sets of a_1. This situation corresponds to faces in R^2 and was considered above. The second group corresponds to the situation where the boundary set X of a_2 intersects either D or E but not both. This corresponds to the spatial situation where a_2 is completely on one side of the plane P formed by the boundary, interior and exterior sets A, B and C of a_1. In this situation, the boundary set X of a_2 may intersect the plane P and hence the boundary, interior and exterior sets A, B and C or not at all. All combinations are shown in Table 3.

Table 3: Set intersections between the boundary set X of a2 and the interior (A), exterior (B) and boundary (C) sets of a1 in R3 when a1 intersects only one of the sets D or E

	1	2	3	4	5	6	7	8
Exterior A				X		X	X	X
Boundary A			X		X	X		X
Interior A		X			X		X	X
Above D	X	X	X	X	X	X	X	X
Below E								

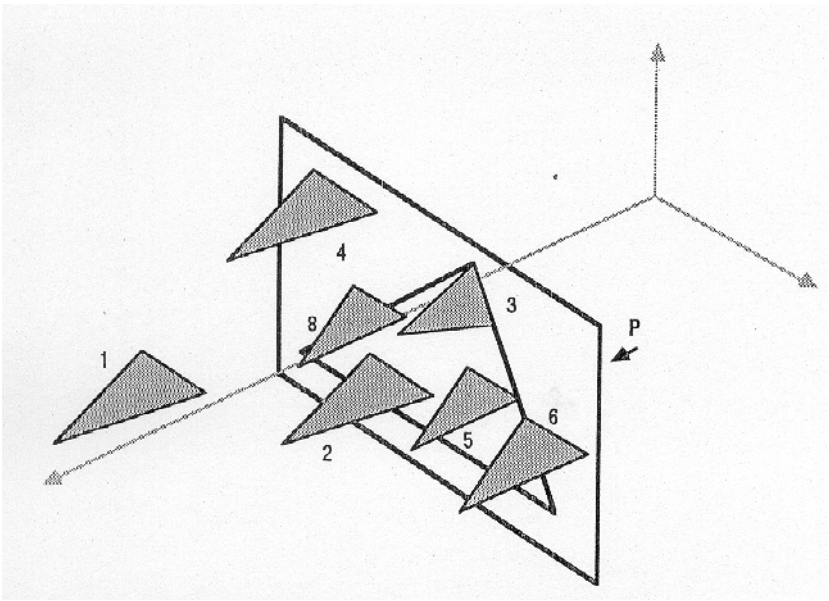


Figure 18: Relationships formed by the intersection of the boundary set X of a face a_2 with the boundary, interior and exterior sets (A, B and C) of a face a1 when the boundary set of the face intersects the point-set D (or E). a_2 is shown shaded, however only the black outline is the boundary set of a_2.

Since the topological relationships are the same no matter which set D or E on either side of the plane P the boundary set of a_2 intersects, the combinations are shown in the table with the

marker offset between D and E. Note that relationship 7 is not possible between two closed connected simplexes. The other seven relationships are shown spatially in Figure 18.

The third group of relationships occurs when the boundary set *X* of a_2 intersects both D and E and hence must intersect the sets A, B and C of a_1 at an interval whose boundaries correspond to two points from the boundary set *X* of a_2 and interior corresponds to the interior set Y of a_2. The possible combinations between the boundary set *X* of a_2 and the boundary, interior and exterior sets A, B and C of a_1 when the boundary set *X* intersects both D and E as well, are shown in Table 4.

Table 4: Set Intersections between the boundary of set X of a_2 and the exterior (A), boundary (B) and interior (C) sets of a_1 in R^3 when the boundary of a_2 intersects both of the sets D and E.

	9	10	11	12	13	14	15
Exterior A			X		X	X	X
Boundary A		X		X	X		X
Interior A	X			X		X	X
Above D	X	X	X	X	X	X	X
Below E	X	X	X	X	X	X	X

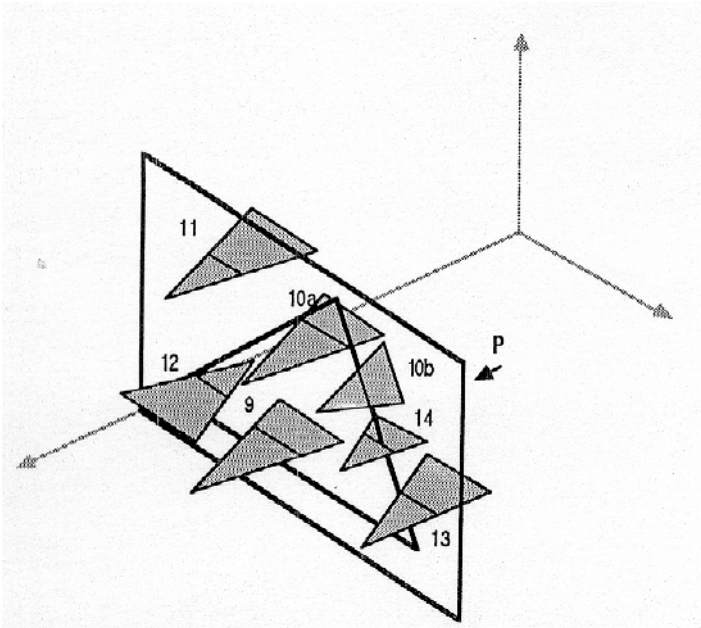


Figure 19: Relationships formed by the intersection of the boundary set X of a face a_2 with die boundary, interior and exterior sets (A, B and C) of a face a_1 when the boundary set of the face intersects both point-sets D and E (passes through the plane P formed from the union of the boundary, interior and exterior sets of a_1). Although a_2 is shown shaded, only the black outline is the boundary set.

The spatial interpretations are shown in Figure 19. Note that for relationship 10 in column two, the interior set of the face a_2 may be used to derive a second possibility. These relationships are marked 10a and 10b in the spatial interpretations of these relationships,

shown in Figure 19. In addition, relationship 14 is not possible between closed, connected faces.

By examination of all relationships in Figure 16, Figure 18 and Figure 19, the number of unique relationships between faces in R^3 is fourteen since relationships 1, 4 and 11 are particular types of the disjoint relationship shown in Figure 16 and relationships 3, 6 and 13 are particular types of the meet relationship shown in Figure 16.

To reduce these fourteen relationships in detail, the union of the boundary and interior points-sets of a_1 and a_2 in each relationship is considered. Relationships which are homeomorphic can then be reduced to their homeomorphs. Thus, the complete two layer hierarchy of binary topological relationships between faces (2-simplexes) in R^3 is shown in Figure 20.

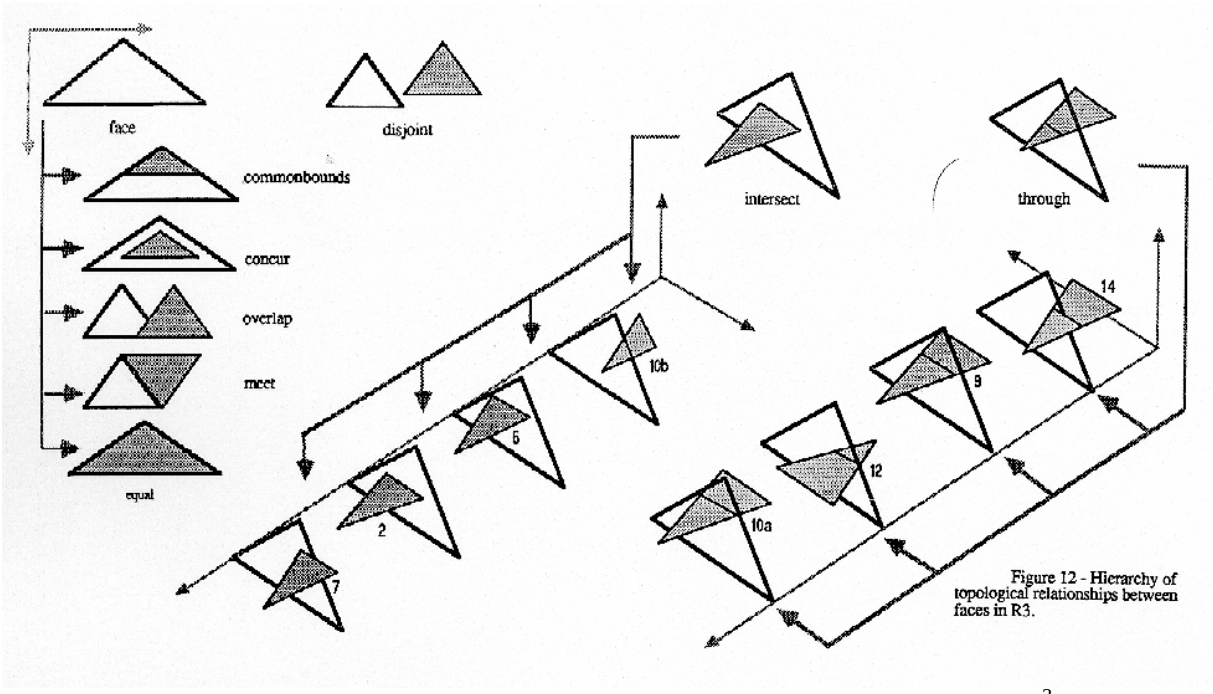


Figure 20: Hierarchy of topological relationships between faces in R^3.

13. A topological model for a 3D spatial information system

Pigot S. (1992) A topological model for a 3D spatial information system, in Proceedings of the 5th Int. Symp. on Spatial Data Handling, Charleston, USA, vol. 1.

13.1 Abstract

This paper focuses on the definition and development of a topological model and operators for a three-dimensional spatial information system (SIS). The domain of the model consists of general k-dimensional spatial objects ($0 \leq k \leq 3$) forming k-dimensional topological spaces and represented by finite k-dimensional cell complexes or subdivisions embedded in the open Euclidean 3-manifold R^3. The topological properties of 0-cell complexes are trivial whilst for l-cell complexes graph theoretic techniques with the addition of consistency chooks from knot theory are sufficient for the embedding in R^3. 2-cell complexes define open or closed surfaces and assume the topological properties of 2-manifolds and a plethora of results in algebraic topology for calculating these properties and maintaining consistency. If a 3-cell is defined as an open 3-manifold with 2-manifold boundary(s) then a 3-cell complex of the open Euclidean 3-manifold modelling space is formed by the solids, however the lack of topological results for characterising arbitrary subdivisions of any 3-manifold forces the introduction of practical methods for calculating their important topological properties and maintaining their consistency based on the 2-manifold boundaries which define the solids (and thus the subdivision). The individual 2-cells and 3-cells of these higher dimensional complexes may be singular, a general notion which includes the domain of non-manifold relationships mentioned in other work. Traversal operators may be defined based on the possible orderings of the neighbourhood structure of the cell complexes which will also be defined in this work. Local and global assembly and disassembly operators may be defined by considering combinations of local elements (i.e. cells) and global elements (i.e. complexes) with consistency and sufficiency as the major objectives. A review of the current approaches to n-dimensional topological modelling is then made on the basis of the domain and operators defined and suggested in this research.

14. TetGen, A Quality Tetrahedral Mesh Generator and Three-Dimensional Delaunay Triangulator

Si, H. (2004) TetGen, A Quality Tetrahedral Mesh Generator and Three-Dimensional Delaunay Triangulator, User's Manual (version 1.3), http://tetgen.berlios.de, pp. 17-18, 22.

14.1 Abstract

TetGen is a quality tetrahedral mesh generator and 3-dimensional Delaunay triangulator. Based on the the-state-of-the-art algorithms for Delaunay tetrahedralization and mesh generation, this program has been specifically designed to fulfil the task of automatically generating high quality tetrahedral meshes, which are suitable for scientific computing using numerical methods such as finite element and finite volume methods.

The purpose of the manual is to give a brief explanation of the problems solved by TetGen and to provide a detailed user's documentation. In the manual, the user will learn how to create tetrahedral meshes using TetGen's input files and command line switches. In the last chapter of the manual, the programming interface of TetGen for calling TetGen from another program is explained. Various examples are given to simplify the "getting started".

The main goal of TetGen is to create tetrahedral meshes that have special properties for solving partial differential equations (PDEs) by finite element methods (FEM) and finite volume methods (FVM). These methods have been widely applied in contemporary engineering applications for simulating physical phenomena, such as mechanical deformation, heat transfer, electromagnetic problems, etc.

TetGen is the starting point of this research. The aim is to add functions to TetGen which can solve the problems that are occurring when triangles are intersecting.

14.2 Using TetGen (pp. 17)

This section describes the use of TetGen as a stand alone program. It is invoked from the command line with a set of switches and an input file name. Switches are used to control the behaviour of TetGen and to specify the output files. In correspondence to the different switches, TetGen will generate the Delaunay tetrahedrization, the constrained Delaunay tetrahedrization or a quality conforming Delaunay mesh. The command syntax is:

```
tetgen [-pq__a__Ars__iMT__dzo_fengGOBNEFICQVvh] input_file
```

Underscores indicate that numbers may optionally follow certain switches. Do not leave any space between a switch and its numeric parameter. The "input file" must be a file with extension .node, or extension .poly or, .smesh, or .off, or .mesh if the -p switch is used. If -r is used, you must supply .node and .ele files, and possibly a .face file, and a .vol file as well.

14.3 Command line switches (pp. 18)

-p Tetrahedralizes a piecewise linear complex.
-q Quality mesh generation. A minimum radius-edge ratio may be specified (default 2.0).
-a Applies a maximum tetrahedron volume constraint.
-A Assigns attributes to identify tetrahedra in certain regions.
-r Reconstructs/Refines a previously generated mesh.
-s Attempts to remove slivers. A maximum dihedral angle may be specified (default 175 degree).
-i Inserts a list of additional points into mesh.
-M Does not merge coplanar facets.
-T Set a tolerance for coplanar test (default 1e-8).
-d Detect intersections of PLC facets.

-z Numbers all output items starting from zero.
-o2 Generates second-order subparametric elements.
-f Outputs faces (including non-boundary faces) to .face file.
-e Outputs subsegments to .edge file.
-n Outputs tetrahedra neighbours to .neigh file.
-g Outputs mesh to .mesh file for viewing by Medit.
-G Outputs mesh to .msh file for viewing by Gid.
-O Outputs mesh to .off file for viewing by Geomview.
-B Suppresses output of boundary information.
-N Suppresses output of .node file.
-E Suppresses output of .ele file.
-F Suppresses output of .face file.
-I Suppresses mesh iteration numbers.
-C Checks the consistency of the final mesh.
-Q Quiet: No terminal output except errors.
-V Verbose: Detailed information, more terminal output.
-v Prints the version information.
-h Help: A brief instruction for using TetGen.

14.4 TetGen's file formats (pp. 22)

.node input/output a list of nodes.
.poly input a PLC.
.smesh input/output a simple PLC.
.ele input/output a list of tetrahedra.
.face input/output a list of triangular faces.
.edge output a list of boundary edges.
.vol input a list of maximum volumes.
.neigh output a list of neighbours.

References

Aftosmis M.J., Berger M.J., Melton J.E. (1997) Robust and efficient Cartesian mesh generation for component-based geometry, Technical Report AIAA-97-0196, US Air Force Wright Laboratory, pp. 2-4.

Billen R., Zlatanova S., Mathonet P., Boniver F. (2002) The Dimensional Model: A Framework to Distinguish Spatial Relationships, Advances in Spatial Data Handling, Springer-Verlag, Heidelberg, pp. 286-287, 290-293, 294-296.

Boender E., Bronsvoort W.F., Post F.H. (1994) Finite-element mesh generation from constructive-solid-geometry models, Butterworth-Heinemann Ltd, Computer-Aided Design Volume 26 Number 5 May 1994.

Edelsbrunner H. (2001) Geometry and Topology for Mesh Generation, Cambridge University Press, Cambridge, pp. 44-46, 59-60, ISBN 0521793092

George, P. and Borouchaki, H. (1998) Delaunay Triangulation and Meshing, Application to Finite Elements, Hermes, Paris, pp. 5-11, 13-14.

Goodman J. E. and O'Rourke J. (1997) Handbook of Discrete and Computational Geometry, Boca Raton, FL: CRC Press, pp. 377, 413, 475.

Keemink N.C.(1993) VWO Wiskunde B Samengevat, schematisch overzicht van de examenstof, uitgeverij Onderwijspers BV Leiden, pp. 85-88.

Lay D.C. (1998) Linear algebra and its applications, second edition, University of Maryland, Addison-Wesley, pp. 133-137.

O'Rourke J. (1998) Computational Geometry in C (Second Edition), Cambridge University Press, Cambridge, pp. 220-238.

Oosterom P., Quak W., Thijssen T. () About Invalid, Valid and Clean Polygons.

Pigot S. (1991) Topological models for 3D spatial information systems, In: D. M. Mark and D. White (Eds.), Auto Carto 10, Baltimore, MD, pp. 374-377, 381-387.

Pigot S. (1992) A topological model for a 3D spatial information system, in Proceedings of the 5th Int. Symp. on Spatial Data Handling, Charleston, USA, vol. 1.

Si, H. (2004) TetGen, A Quality Tetrahedral Mesh Generator and Three-Dimensional Delaunay Triangulator, User's Manual (version 1.3), http://tetgen.berlios.de, last visited on 2004-11-09, pp. 17-18, 22.