Concepts of the simplicial complex-based data model for 3D Topography

Report RGI – 011 - 05

Draft version 0.8

Friso Penninga TU Delft
Han Wammes Oracle

December 2006

Bsik project 3D Topografie

Introduction

In this report the new data model and its implementation will be described. Part I of the report consists of a paper describing the model and its implementation. Part II of the report consists of three appendices describing the initial creation and loading of the tetrahedron table, the creation of the required functions and procedures and the preperation of the TEN structure for query and analysis.

Delft, December 2006

Part I

A Compact Topological DBMS Data Structure For 3D Topography

Friso Penninga and Peter van Oosterom

Delft University of Technology, OTB, section GIS Technology,
Jaffalaan 9, 2628 BX the Netherlands
F.Penninga@tudelft.nl, oosterom@tudelft.nl

Abstract. The objective is to develop a data structure that is capable of handling large data volumes and offers support for querying, analysis and validation. Based on earlier results (i.e. the full decomposition of space, the use of a TEN structure and applying Poincaré simplicial homology as mathematical foundation) a simplicial complex-based TEN structure is developed. Applying simplicial homology offers full control over orientation of simplexes and enables one to derive substantial parts of the TEN structure, instead of explicitly store the entire network. The described data structure is developed as a DBMS data structure and the usage of views, function based indexes and 3D R-trees result in a compact topological 3D data structure.

1 Introduction

1.1 Motivation

Most current topographic data sets are limited to representing the real world in two dimensions. Nevertheless, due to developments in multiple land use, environmental modelling and data acquisition techniques, attention is shifting towards 3D topographic data modelling. Early 3D developments focused mostly on 3D visualisation, while with the maturing of 3D GIS research the scope has broadened and now also includes 3D query and analysis. As a result the need arised for a data structure capable of storing 3D data and supporting 3D query and analysis (see Figure 1). As data volume increases substantially with the step from 2D to 3D, which is partly due to new high point density data acquisition techniques (see Figure 2), maintaining data consistency becomes an important objective of the new data structure.

The objective of the research is to develop a data structure that is capable of handling large data volumes and offers support for querying, analysis and validation. An obvious step in dealing with large data volumes is to use spatial databases. Even though 3D coordinates can be used in some spatial databases, 3D data types are still missing [1]. Defining a new 3D data type is part of the research and in this research tetrahedrons will be used as building blocks for the 3D models.

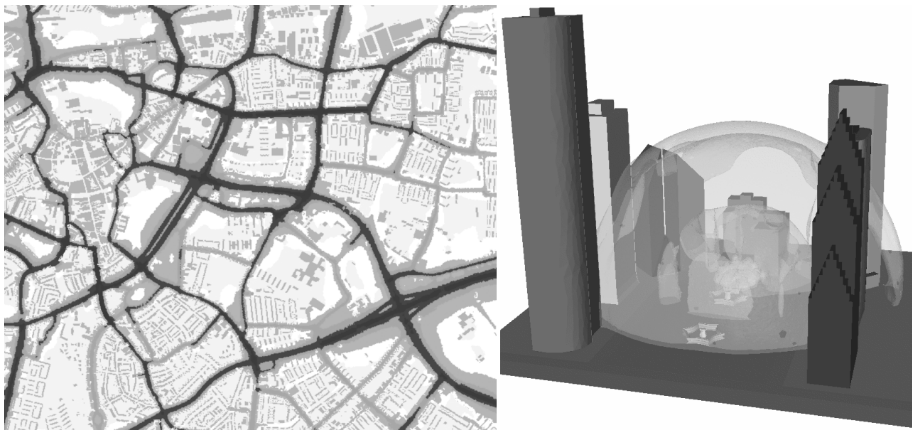


Fig. 1. Increasing reality: shift from 2D analysis to 3D

Fig. 2. Terrestrial laser scanning provides insight in complex objects

1.2 Related research

Research in the field of 3D GIS has been performed for about the last two decades. Zlatanova et al. [2] give an overview of the most relevant developments in this period. Related to the topics introduced in this paper, Carlson [3] can be seen as the starting point as he introduced a simplicial complex-based approach of 3D subsurface structures. However, this approach was limited to the use of 0-, 1- and 2-simplexes in 3D space. Extending this into higher dimensions (as indicated by Frank and Kuhn [4]) is mentioned as a possibility. The explicit use of 3D manifolds to model 3D features is explored by Pigot [5, 6] and Pilouk [7] introduces the TEtrahedral irregular Network (TEN), in which the 3-simplex is used as building block. A topological data model based on 2D simplicial complexes (in 2D space) is introduced [8] and implemented in the PANDA system [9], an early object-oriented database. In applications polyhedrons are often used as 3D primitive [10, 11].

2 Previous research

2.1 Modelling 3D Topography: full decomposition of 3D space

With respect to modelling 3D topographic data two fundamental observations are of great importance [12]:

- Physical objects have by definition a volume. In reality, there are no point, line or polygon objects, only point, line or polygon representations exist (at a certain level of generalisation). The ISO 19101 Geographic information - Reference model [13] defines features as 'abstractions of real world phenomena'. In most current modelling approaches the abstraction (read 'simplification') is in the choice for a representation of lower dimension. However, as the proposed method uses a tetrahedral network (or mesh), the simplification is already in the subdivision into easy-to-handle parts (i.e. it is a finite element method!).
- The real world can be considered a volume partition: a set of nonoverlapping volumes that form a closed modelled space. As a consequence, objects like 'earth' or 'air' are explicitly part of the real world and thus have to be modelled.

Inclusion of air and earth objects is often considered unnecessary, thus more serving the abstract goal of 'clean modelling' than an actual useful goal. This is however not the case. These air and earth objects do not just fill up the space between features of the other types, but are often also subject of analyses, such as noise and odour modelling.

Although the model consists of volume features, some planar features might still be very useful, as they mark the boundary (or transition) between two volume features. The new approach supports these planar features, but only as 'derived features', meaning that these features are lifetime dependent from their parents (the two neighbouring volume features). For instance, a 'wall' might be the result of the association between a 'house' and the 'air'. These planar features may even have attributes, but semantically they do not bound the building. In other words: the house is represented by a volume, with neighbouring volumes that may represent air, earth or perhaps another adjacent house.

2.2 Using a Tetrahedral irregular network (TEN)

After initial ideas [14] on a hybrid data model (an integrated TIN/TEN model, based on the pragmatic point of view: model in 2,5D where possible and only in exceptional cases switch to a full 3D model) the decision was made [12] to model all features in a single TEN. The preference for these simplex-based data structures is based on certain qualities of simplexes:

- Well defined: a n-simplex is bounded by $n+1$ $(n-1)$-simplexes. E.g. a 2-simplex (triangle) is bounded by 3 1-simplexes (edges)
- Flatness of faces: every face can be described by three points
- A n-simplex is convex (simplifies amongst others point-in-polygon tests)

A disadvantage of simplexes is the introduction of a 1:n relationship between features and their representations. Increasing complexity and data volume are usually the result. However, the problem of complexity might be tackled by a well-designed user interface. An average user should handle features (as polyhedrons) and internally these polyhedrons can be tetrahedronised and inserted into the topographic data model. The data volume problem is tackled in our approach by utilising the mathematical foundations of simplicial complexes.

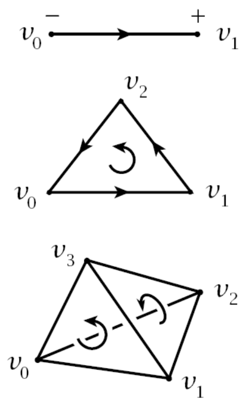
2.3 Mathematical foundation: Poincaré simplicial homology

The new volumetric approach uses tetrahedrons to model the real world. These tetrahedrons in the TEN structure consist of nodes, edges and triangles. All four data types are simplexes: the simplest geometry in each dimension. A more formal definition [15] of a n-simplex S_n can be given: a n-simplex S_n is the smallest convex set in Euclidian space $\mathbb{R}^m$ containing $n+1$ points $v_0, \ldots, v_n$ that do not lie in a hyperplane of dimension less than n. As the n-dimensional simplex is defined by $n+1$ nodes, it has the following notation: $S_n =< v_0, \ldots, v_n >$. The boundary of a n-simplex is defined by the following sum of $n-1$ dimensional simplexes [16] (the *hat* indicates omitting the specific node):

$$\partial S_n = \sum_{i=0}^{n} (-1)^i < v_0, \ldots, \hat{v}_i, \ldots, v_n >$$

This results in (see Figure 3):

$$\begin{array}{ll}
S_1 =< v_0, v_1 > & \partial S_1 =< v_1 > - < v_0 > \\
S_2 =< v_0, v_1, v_2 > & \partial S_2 =< v_1, v_2 > - < v_0, v_2 > + < v_0, v_1 > \\
S_3 =< v_0, v_1, v_2, v_3 > & \partial S_3 =< v_1, v_2, v_3 > - < v_0, v_2, v_3 > \\
 & \quad + < v_0, v_1, v_3 > - < v_0, v_1, v_2 >
\end{array}$$



$$\partial[v_0, v_1] = [v_1] - [v_0]$$

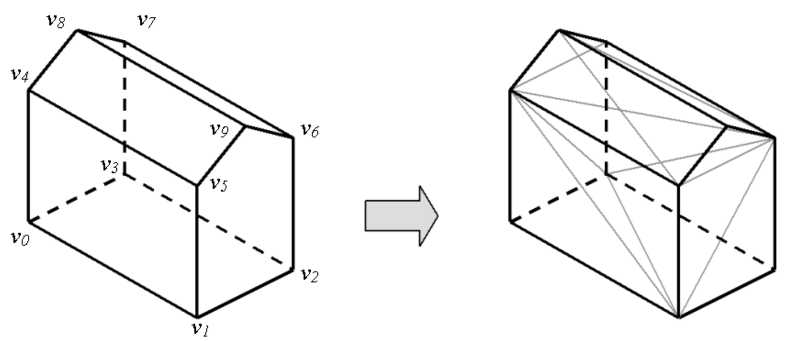
$$\partial[v_0, v_1, v_2] = [v_1, v_2] - [v_0, v_2] + [v_0, v_1]$$

$$\partial[v_0, v_1, v_2, v_3] = [v_1, v_2, v_3] - [v_0, v_2, v_3] + [v_0, v_1, v_3] - [v_0, v_1, v_2]$$

Fig. 3. Simplexes and their boundaries (From [15])

It is assumed that all simplexes are ordered. As a simplex S_n is defined by $n+1$ vertices, $(n+1)!$ permutations exist. All even permutations of an ordered simplex $S_n =< v_0, \ldots, v_n >$ have the same orientation, all odd permutations have opposite orientation. So edge $S_1 =< v_0, v_1 >$ has boundary $\partial S_1 =< v_1 > - < v_0 >$. The other permutation $S_1 = - < v_0, v_1 >=< v_1, v_0 >$ has boundary $\partial S_1 =< v_0 > - < v_1 >$, which is the opposite direction. In a similar way the boundaries of the other five combinations of S_2 and the other 23 combinations of S_3 can be given. As a consequence operators like the dual of a simplex become very simple: it only requires a single permutation.

Another favourable characteristic is that with S_3 either all normal vectors of the boundary triangles point inwards or all normal vectors point outwards. It is a direct result from the definition of the boundary operator, as it is defined in such a way that $\partial^2 S_n$ is the zero homomorphism, i.e. the boundary of the boundary equals zero. Now consider $\partial^2 S_3$. The boundary of a tetrahedron consist of four triangles, and the boundaries of these triangles consist of edges. Each of the six edges of S_3 appears two times, as each edge bounds two triangles. As the zero homomorphism states that the sum of these edges equals zero, this is the case if and only if the edges in these six pairs have opposite signs. The edges of two neighbouring triangles have opposite signs if and only if the triangles have the same orientation, i.e. either both are oriented outwards or both are oriented inwards. This characteristic is important in deriving the boundary of a simplicial complex (construction of multiple simplexes). If this identical orientation is assured for all boundary triangles of tetrahedrons (which can be achieved by a single permutation when necessary), deriving the boundary triangulation of a feature will reduce to adding up boundary triangles of all related tetrahedrons, as internal triangles will cancel out in pairs due to opposite orientation. Figure 4 shows an example in which all boundaries of the tetrahedrons are added up in order to obtain the boundary triangulation of the building.



$$
\begin{array}{ll}
S_{31} = \langle v_0, v_1, v_3, v_4 \rangle & \partial S_{31} = \langle v_1, v_3, v_4 \rangle - \langle v_0, v_3, v_4 \rangle + \langle v_0, v_1, v_4 \rangle - \langle v_0, v_1, v_3 \rangle \\
S_{32} = \langle v_1, v_2, v_3, v_6 \rangle & \partial S_{32} = \langle v_2, v_3, v_6 \rangle - \langle v_1, v_3, v_6 \rangle + \langle v_1, v_2, v_6 \rangle - \langle v_1, v_2, v_3 \rangle \\
S_{33} = \langle v_1, v_3, v_4, v_6 \rangle & \partial S_{33} = \langle v_3, v_4, v_6 \rangle - \langle v_1, v_4, v_6 \rangle + \langle v_1, v_3, v_6 \rangle - \langle v_1, v_3, v_4 \rangle \\
S_{34} = \langle v_1, v_4, v_5, v_6 \rangle & \partial S_{34} = \langle v_4, v_5, v_6 \rangle - \langle v_1, v_5, v_6 \rangle + \langle v_1, v_4, v_6 \rangle - \langle v_1, v_4, v_5 \rangle \\
S_{35} = \langle v_3, v_4, v_6, v_7 \rangle & \partial S_{35} = \langle v_4, v_6, v_7 \rangle - \langle v_3, v_6, v_7 \rangle + \langle v_3, v_4, v_7 \rangle - \langle v_3, v_4, v_6 \rangle \\
S_{36} = \langle v_4, v_6, v_7, v_8 \rangle & \partial S_{36} = \langle v_6, v_7, v_8 \rangle - \langle v_4, v_7, v_8 \rangle + \langle v_4, v_6, v_8 \rangle - \langle v_4, v_6, v_7 \rangle \\
S_{37} = \langle v_4, v_5, v_6, v_8 \rangle & \partial S_{37} = \langle v_5, v_6, v_8 \rangle - \langle v_4, v_6, v_8 \rangle + \langle v_4, v_5, v_8 \rangle - \langle v_4, v_5, v_6 \rangle \\
S_{38} = \langle v_5, v_6, v_8, v_9 \rangle & \underline{\partial S_{38} = \langle v_6, v_8, v_9 \rangle - \langle v_5, v_8, v_9 \rangle + \langle v_5, v_6, v_9 \rangle - \langle v_5, v_6, v_8 \rangle} \quad + \\
C_3 = & - \langle v_0, v_3, v_4 \rangle + \langle v_0, v_1, v_4 \rangle - \langle v_0, v_1, v_3 \rangle + \langle v_2, v_3, v_6 \rangle + \\
 & \langle v_1, v_2, v_6 \rangle - \langle v_1, v_2, v_3 \rangle - \langle v_1, v_5, v_6 \rangle - \langle v_1, v_4, v_5 \rangle - \langle v_3, v_6, v_7 \rangle \\
 & + \langle v_3, v_4, v_7 \rangle + \langle v_6, v_7, v_8 \rangle - \langle v_4, v_7, v_8 \rangle + \langle v_4, v_5, v_8 \rangle + \\
 & \langle v_6, v_8, v_9 \rangle - \langle v_5, v_8, v_9 \rangle + \langle v_5, v_6, v_9 \rangle
\end{array}
$$

Fig. 4. Deriving the boundary triangulation from the TEN

3 A simplicial complex-based TEN structure

3.1 Conceptual model

In a TEN structure tetrahedrons are usually defined by four triangles, triangles by three edges and edges by two nodes. Geometry is stored at node level. As a result reconstructing geometry of for instance a tetrahedron becomes a relatively laborious operation. In simplicial homology simplexes of all dimensions are defined by their vertices. Relationships between other simplexes, for instance between tetrahedrons and triangles, can be derived by applying the boundary operator. As a result [17], there is no need for explicit storage of these relationships. This concept is illustrated in the UML class diagram in Figure 5. The associations between the tetrahedron, triangle and edge class and the node class show that these simplexes are specified by an ordered list of nodes. The interrelationships between tetrahedrons, triangles and nodes (the boundary/coboundary relationships) are derived and signed (i.e. oriented).

Figure 5 shows also the concept of the full decomposition of space. The real world consists of volume features and features of lower dimension are modelled as association classes. As a result, instances of these classes are lifetime dependent from a relationship between two volume features.

3.2 Vertex encoding

In the simplicial complex-based approach simplexes will be defined by their vertices, resulting in a lot of references to these vertices. Since the geometry is the only attribute of a vertex, adding a unique identifier to each point and building an index on top of this table will cause a substantial increase in data storage. To deal with this an alternative approach is used. It is based on the observation that adding a unique identifier is a bit redundant, as the geometry in itself will be a unique identifier as well. To achieve this the coordinate pair is concatenated into one long identifier code. Sorting this list will result in a very basic spatial index. In a way this approach can be seen as building and storing an index, while the original table is deleted. The possibilities of applying techniques like bitwise interleaving, 3D Morton or Peano-Hilbert coding are recognised, but for reasons of insightfulness the concatenated version will be used in this paper.

Figure 6 illustrates this idea of vertex encoding in a simplicial complex-based approach. A house is tetrahedronised and the resulting tetrahedrons are coded as concatenation of their four vertices' coordinates. Each row in the tetrahedron encoding can be interpreted as $x_1y_1z_1x_2y_2z_2x_3y_3z_3x_4y_4z_4$. For reasons of simplicity only two positions are used for each coordinate element. Therefore the last row (100000000600100600100608) should be interpret as the tetrahedron defined by the vertices $(10, 00, 00), (00, 06, 00), (10, 06, 00)$ and $(10, 06, 08)$, which is the tetrahedron at the bottom right of the house.

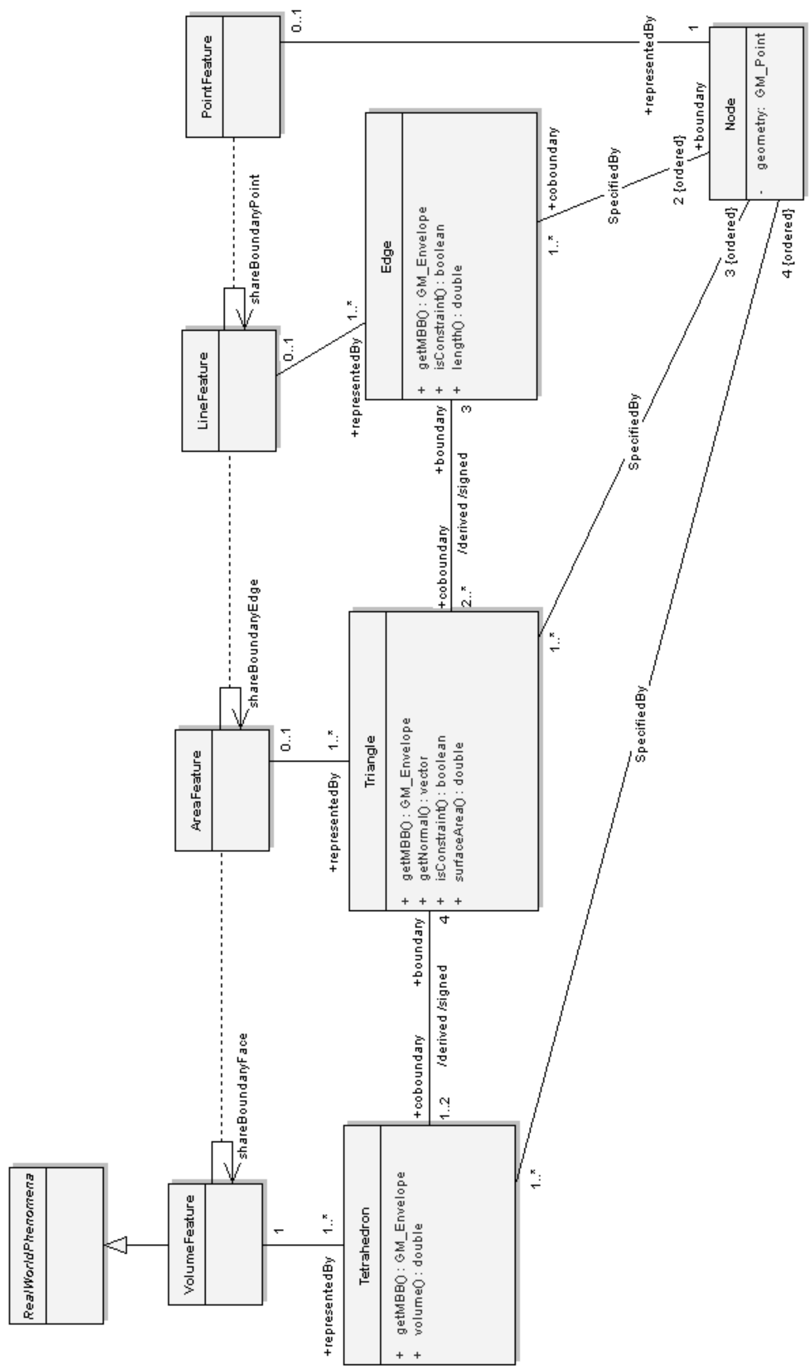


Fig. 5. UML class diagram of the simplicial complex-based approach

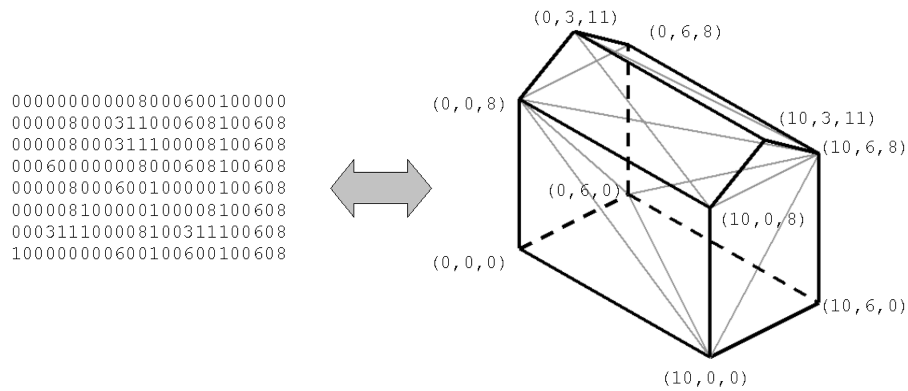


Fig. 6. Describing tetrahedrons by their encoded vertices

4 Implementation: proof of concept

In order to provide more insight in the proposed new approach this section will outline the current DBMS implementation. It is developed and tested with a small toy dataset, consisting of 56 tetrahedrons, 120 triangles, 83 edges and 20 nodes. At this moment the required tetrahedronisation algorithms are not implemented yet, although previous research did focus on this topic [17, 18]. As a temporal workaround the toy dataset was tetrahedronised by hand and the same dataset was used in a previous implementation (a classical TEN approach). Based on this implementation a 2D viewer (Oracle MapViewer) was adapted for 3D data by the use of a function `rotateGeom`. Both the implementation and the viewer are described in [17]. In Figure 7 the small dataset can be seen in the MapViewer. The dataset basically represents a small piece of the earth surface with a house and a road on top of it.

This section will start with creating the data structure, i.e. to define the table and views to store tetrahedrons, triangles, edges and nodes. After that it will be shown that also the constraints can be derived, so no additional explicit storage is required. The next topic is deriving topological relationships. The section continues with some remarks on validation, followed by some examples on querying and analysis and ends with initial remarks on performance.

4.1 Building the data structure

The tetrahedron table is the only table in the implementation. It consists of a single column (`NVARCHAR2`)in which the encoded tetrahedrons are described in the form $x_1y_1z_1x_2y_2z_2x_3y_3z_3x_4y_4z_4id$. Note that besides the geometry also an unique identifier is added, which refers to a volume feature that is (partly) represented by the tetrahedron. Each tetrahedron has positive orientation, meaning

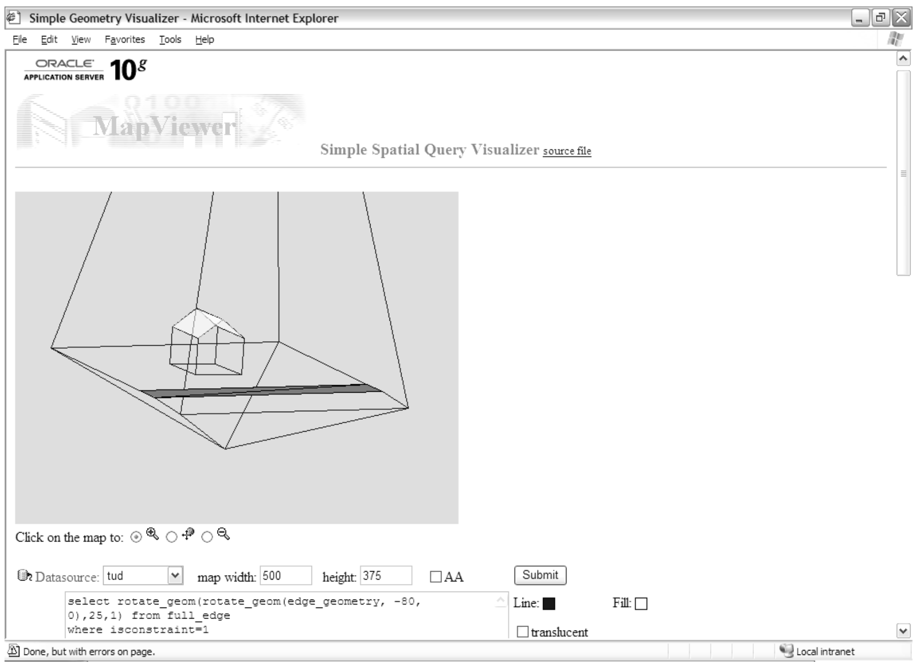


Fig. 7. Adapting the 2D MapViewer for 3D data by a function `rotateGeom`

that all normal vectors on boundary triangles are oriented outwards. This consistent orientation is required to ensure that each boundary triangle appears two times: once with positive and once with negative orientation. To achieve this, each tetrahedron's orientation is checked. All tetrahedrons with inward orientation are replaced by tetrahedrons with outward orientation:

```
create or replace procedure tettableoutwards
   (...)
      checkorientation(codelength,currenttetcode,bool);
      if (bool = 0) then
         permutation12(codelength,currenttetcode,newtetcode);
         update tetrahedron
         set tetcode=newtetcode where current of tetcur;
      (...)
```

The `checkorientation` procedure compares the direction of the normal vector of one of the boundary triangles with a vector from this triangle to the fourth (opposite) point of the tetrahedron. In case of an inward orientation a single permutation is carried out by the procedure `permutation12`, which permutes the first and second vertex: `permutation12`($< v_0, v_1, v_2, v_3 >$) results in $< v_1, v_0, v_2, v_3 >$.

Based on the encoded tetrahedrons the boundary triangles can be derived by applying the boundary operator:

```
create or replace procedure deriveboundarytriangles(
   (...)
   a := (SUBSTR(tetcode,1,3*codelength));
   b := (SUBSTR(tetcode,1+3*codelength,3*codelength));
   c := (SUBSTR(tetcode,1+6*codelength,3*codelength));
   d := (SUBSTR(tetcode,1+9*codelength,3*codelength));
   id := (SUBSTR(tetcode,1+12*codelength));
   ordertriangle(codelength,'+'||b||c||d||id, tricode1);
   ordertriangle(codelength,'-'||a||c||d||id, tricode2);
   ordertriangle(codelength,'+'||a||b||d||id, tricode3);
   ordertriangle(codelength,'-'||a||b||c||id, tricode4);
   (...)
```

Note that the triangles inherit the object id from the tetrahedron, i.e. each triangle has a reference to the volume feature represented by the tetrahedron of which the triangle is part of the boundary. The reason for this will be introduced later in this section. It can also be seen that each boundary triangle is ordered by the `ordertriangle` procedure. The objective of this procedure is to gain control over which permutation is used. A triangle has six (= 3!) permutations, but it is important that both in positive and negative orientation the same permutation is used, as they will not cancel out in pairs otherwise. The `ordertriangle` procedure always rewrites a triangle $< a, b, c >$ such that $a < b < c$ holds, which is an arbitrary criterion.

Slightly altered versions of the `deriveboundarytriangles` procedure are used to create the triangle view. The modified procedures derive respectively the first, second, third and fourth boundary triangle of a tetrahedron. The resulting view contains all triangles and their coboundaries (the coboundary of a n-dimensional simplex S_n is the set of all $(n+1)$-dimensional simplexes S_{n+1} of which the simplex S_n is part of their boundaries ∂S_{n+1}). In this case the coboundary is the tetrahedron of which the triangle is part of the boundary. This coboundary will prove useful in deriving topological relationships later in this section. The view is created as:

```
create or replace view triangle as
   select deriveboundarytriangle1(3,tetcode) tricode,
   tetcode fromtetcode from tetrahedron
   UNION ALL
   select deriveboundarytriangle2(3,tetcode) tricode,
   tetcode fromtetcode from tetrahedron
   UNION ALL
   select deriveboundarytriangle3(3,tetcode) tricode,
   tetcode fromtetcode from tetrahedron
   UNION ALL
   select deriveboundarytriangle4(3,tetcode) tricode,
   tetcode fromtetcode from tetrahedron;
```

The resulting view will contain four times the number of tetrahedrons, and every triangle appears two times: once with positive and once with sign negative sign (and not in a permutated form, due to the `ordertriangle` procedure).

In a similar way the views with edges and nodes can be constructed. In current implementation edges are undirected en do not inherit object ids, as no application for this is identified at the moment. However, strict application of the boundary operator would result in directed triangles. With the tetrahedron table and triangle, edge and node view the data structure is accessible at different levels. Due to the encoding of the vertices, both geometry and topology are present at every level, thus enabling switching to the most appropriate approach for every operation.

4.2 Creating views with derived constraints

Features in the model are represented by a set of tetrahedrons. To ensure that these tetrahedrons represent the correct geometry, the outer boundary is triangulated and these triangles are used as constraints. This implies that these triangles will remain present as long as the feature is part of the model (i.e. they are not deleted in a flipping proces). To achieve this, the incremental tetrahedronisation algorithm needs to keep track of these constrained triangles. In contrast with what one might expect, it is not necessary to store these constraints explicitly, as they can be derived as well:

```
create or replace view constrainedtriangle as
```

```
select t1.tricode tricode from triangle t1
where not exists (select t2.tricode from triangle t2
                  where t1.tricode = t2.tricode*-1);
```

This statement uses the fact that although every triangle (in a geometric sense) appears two times (with opposite orientation) in the triangle view, not every triangle code appears two times. As stated before the triangle code is a concatenation of the encoded vertices and the inherited object id from the tetrahedron (its coboundary). This implies that for internal triangles (i.e. within an object) the triangle and its dual will have (apart from the sign) the exact same triangle code (geometry + object id), but in case of boundary triangles (i.e. constrained triangles) this code will differ due to the different inherited object id's. So in simplified form, consider triangle codes -1,7,2,-7,-3 and 1. In this case triangles 2 and -3 will be constrained triangles. Deriving constrained edges from constrained triangles is straightforward, as all boundary edges from constrained triangles are constrained edges.

4.3 Creating views with derived topological relationships

In a TEN the number of possible topological relationships is limited. As the TEN can be considered as a decomposition of space, relationships like overlap, cover or inside do not occur. Only relationships based on the interaction between tetrahedron boundaries occur. Tetrahedrons (and their boundaries) are either disjoint or touch. The case in which two boundary triangles touch (i.e. the faces touch each other) is the neighbour relation. Two related relationships are derived in views in the implementation. The first is the relationship between a triangle and its dual. This relationship is important in the proces of finding neighbours from tetrahedrons. The view is created by a select statement that uses the identical geometric part of the triangle codes:

```
create or replace view dualtriangle as
   select t1.tricode tricode, t2.tricode dualtricode
   from triangle t1, triangle t2
   where removeobjectid(3,t2.tricode) =
    -1 *removeobjectid(3,t1.tricode);
```

By combining the triangle view and the dualtriangle view, neighbouring tetrahedrons can be found:

```
create or replace function getneighbourtet1(
   (...)
   select fromtetcode into neighbourtet from triangle
   where tricode = (select dt.dualtricode from dualtriangle dt
                    where dt.tricode =
                    deriveboundarytriangle1(codelength,tetcode));
   (...)
```

and based on functions like this one the view with tetrahedrons and their neighbours can be created.

4.4 Validating the data structure

The data structure can be validated by applying the Euler-Poincaré formula: $N - E + F - V = 0$ with N the number of nodes, E the number of edges, F the number of faces and V the number of volumes (including the exterior). As can be seen in Figure 8, the Euler-Poincaré formula holds for all simplicial complexes, including simplicial complexes that consist of simplexes of different dimensions. Due to this characteristic dangling edges and faces cannot be detected, but for instance holes (i.e. missing faces) can be detected.

Within the simplicial complex-based approach the validation strategy is to start with a valid tetrahedronisation and to check every update for correctness before committing it to the database. As a result one will migrate from one valid state into another valid state. This strategy will also include the application of for instance flipping algorithms for the deletion of vertices [19], as such algorithms are designed to maintain a valid TEN during each step of the proces.

Other correctness checks can be implemented, like for instance a check on the triangle view to ensure that every triangle appears two times (with opposite sign, ignoring the inherited object id's). Also validation on feature level can be considered, for instance one can check whether all constrained triangles form a valid polyhedron. For more details on the validation of polyhedrons, see [20].

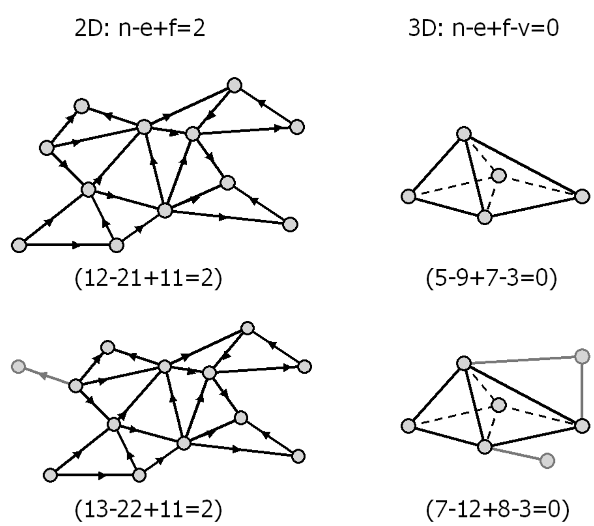


Fig. 8. Using Euler-Poincaré in 2D and 3D for validation: dangling edges and faces remain undetected

4.5 Query and analysis

The presence of views helps to simplify a lot of queries as the functions on which the views are based can be omitted from the queries. The most frequently used elements and relationships are made available through these views. If one is interested in for instance a boundary representation of a feature, one could query the constrained triangle view with a specific object id. The resulting set of constrained triangles will form a valid polyhedron, see Figure 9 for an example. One might consider to simplify this polyhedron further by merging triangles with identical (given a specific tolerance) normal vectors into polygons. However, a polyhedron may consist of triangular faces and these triangulation might be useful for visualisation purposes.

The number of analyses that can be performed on the TEN structure is virtually unlimited. One can think of basic operations like distance, line-of-sight or volume calculations, or more complex operations like tetrahedron-based buffer and overlay [21]. Also a wide variety of simulations can be performed on the tetrahedral mesh, like flooding or air flow simulations. Tetrahedronal meshes can be used and optimized for simulation purposes [22, 23].

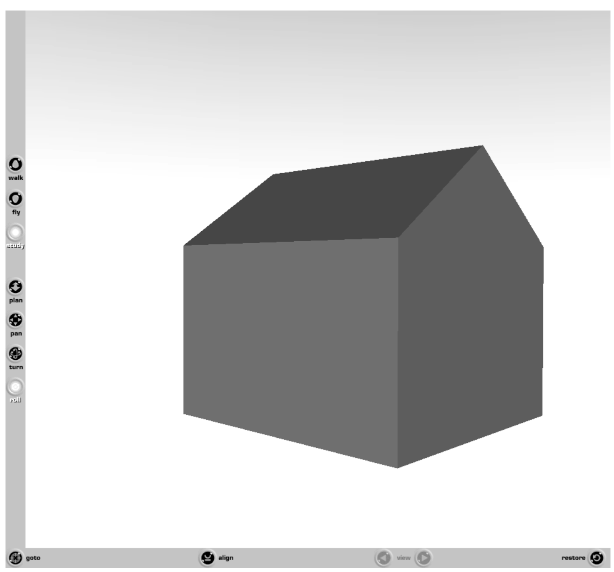


Fig. 9. Output in VRML: result of `select tricode from constrainedtriangle where getobjectid(3,tricode)=3;`. This is the same object as in Figure 7

4.6 Performance

The tetrahedron table is potentially very large, so indexing becomes an important aspect of the data structure. Sorting the table on the tetrahedron code will function as an index, as tetrahedrons in a particular area will be stored closed to each other in the table as well. However, a secondary index might still be needed. As the tetrahedron code contains all geometry, constructing the minimal bounding boxes and building a R-tree will be a logical step. To ensure performance for queries on the views, function based indexes are created for all functions that are used to create views.

5 Conclusions and Discussion

5.1 Conclusion

As stated in the introduction, the objective of this research is to develop a data structure that is capable of handling large data volumes and offers support for querying, analysis and validation. Based on earlier results (i.e. the full decomposition of space, the use of a TEN structure and applying Poincaré simplicial homology as mathematical foundation) a simplicial complex-based TEN structure is developed. Applying simplicial homology offers full control over orientation of simplexes and enables one to derive substantial parts of the TEN structure, instead of explicitly store the entire network. As a result only the single column tetrahedron table has to be stored explicitly. Due to the encoded vertices and inheritance of object id's all constrained edges and faces can be derived, thus avoiding redundant data storage. Since the topological relationships are also derived, updating the structure turns out to be limited to updating the tetrahedron tables. All implicit updates in less dimensional simplexes or topological relationships propagate from this single update action. The described data structure is developed as a DBMS data structure. Spatial DBMS characteristics as the usage of views, function based indexes and 3D R-trees are extensively used and contribute to the compactness and versatility of the data structure. Furthermore a database is capable of coping with large data volumes, which is an essential characteristic in handling large scale 3D data.

5.2 Discussion

An important question is whether the proposed method is innovative. As mentioned in section 1.2 both the idea to use a TEN data structure for 3D data and using simplexes (in terms of simplicial homology) in a DBMS implementation are described by other. However, the proposed approach reduces data storage and eliminates the need for explicit updates of both topology and less dimensional simplexes. By doing so, the approach tackles common drawbacks as TEN extensiveness and laboriousness of maintaining topology. Furthermore, applying simplicial homology offers full control over orientation of simplexes, which is a huge advantage especially in 3D. Integrating these concepts with database functionality results in a new innovative approach to 3D data modelling.

References

1. Kothuri, R., Godfrind, A., Beinat, E.: Pro Oracle Spatial: The essential guide to developing spatially enabled business applications. Apress (2004)
2. Zlatanova, S., Abdul Rahman, A., Pilouk, M.: 3D GIS: Current Status and Perspectives. In: Proceedings of Joint Conference on Geo-Spatial Theory, Processing and Applications, Ottawa, Canada. (2002)
3. Carlson, E.: Three-dimensional conceptual modeling of subsurface structures. In: Auto-Carto 8. (1987) 336–345
4. Frank, A.U., Kuhn, W.: Cell Graphs: A provable Correct Method for the Storage of Geometry. In: Proceedings of the 2nd International Symposium on Spatial Data Handling, Seattle, Washington. (1986)
5. Pigot, S.: A Topological Model for a 3D Spatial Information System. In: Proceedings of the 5th International Symposium on Spatial Data Handling. (1992) 344–360
6. Pigot, S.: A topological model for a 3-dimensional Spatial Information System. PhD thesis, University of Tasmania, Australia (1995)
7. Pilouk, M.: Integrated Modelling for 3D GIS. PhD thesis, ITC Enschede, Netherlands (1996)
8. Egenhofer, M., Frank, A., Jackson, J.: A Topological Data Model for Spatial Databases. In: Proceedings of First Symposium SSD'89. (1989) 271–286
9. Egenhofer, M., Frank, A.: PANDA: An Extensible Dbms Supporting Object-Oriented Software Techniques. In: Datenbanksysteme in Büro, Technik und Wissenschaft. Proceedings of GI/SI Fachtagung, Zürich, 1989. Informatik Fachberichten, Springer-Verlag (1989) 74–79
10. Zlatanova, S.: 3D GIS for urban development. PhD thesis, Graz University of Technology (2000)
11. Stoter, J.: 3D Cadastre. PhD thesis, Delft University of Technology (2004)
12. Penninga, F.: 3D Topographic Data Modelling: Why Rigidity Is Preferable to Pragmatism. In Cohn, A.G., Mark, D.M., eds.: Spatial Information Theory, Cosit'05. Volume 3693 of Lecture Notes on Computer Science., Springer (2005) 409–425
13. ISO/TC211: Geographic information - reference model. Technical Report ISO 19101, International Organization for Standardization (2005)
14. Penninga, F.: Towards 3D Topography using a Feature-based Integrated TIN/TEN Model. In Toppen, F., Painho, M., eds.: AGILE 2005, 8th Conference on Geographic Information Science. Conference Proceedings. Estoril, Portugal, May 26-28. (2005) 373–381
15. Hatcher, A.: Algebraic Topology. Cambridge University Press (2002) Available at http://www.math.cornell.edu/ hatcher.
16. Poincaré, H.: Complément a l'Analysis Situs. Rendiconti del Circolo Matematico di Palermo **13** (1899) 285–343
17. Penninga, F., van Oosterom, P., Kazar, B.M.: A TEN-based DBMS approach for 3D Topographic Data Modelling. In Riedl, A., Kainz, W., Elmes, G., eds.: Progress in Spatial Data Handling, 12th International Symposium on spatial Data Handling, Springer (2006) 581–598
18. Penninga, F., van Oosterom, P.: Updating Features in a TEN-based DBMS approach for 3D Topographic Data Modelling. In Raubal, M., Miller, H.J., Frank, A.U., Goodchild, M.F., eds.: Geographic Information Science, Fourth International Conference, GIScience 2006, Münster, Germany, September 2006, Extended Abstracts. Volume 28 of IfGI prints. (2006) 147–152

19. Ledoux, H., Gold, C.M., Baciu, G.: Flipping to robustly delete a vertex in a Delaunay tetrahedralization. In: Proceedings International Conference on Computational Science and its Applications-ICCSA 2005. Volume 3480 of Lecture Notes on Computer Science., Springer (2005) 737–747
20. Arens, C., Stoter, J., van Oosterom, P.: Modelling 3D spatial objects in a geo-DBMS using a 3D primitive. Computers & Geosciences **31**(2) (2005) 165–177
21. Verbree, E., van der Most, A., Quak, W., van Oosterom, P.: Towards a 3D Feature Overlay through a Tetrahedral Mesh Data Structure. Cartography and Geographic Information Science **32**(4) (2005) 303–314
22. Joe, B.: Construction of three-dimensional improved-quality triangulations using local transformations. SIAM Journal on Scientific Computing (6) (1995) 1292–1307
23. Cutler, B., Dorsey, J., McMillan, L.: Simplification and Improvement of Tetrahedral Models for Simulation. In Scopigno, R., Zorin, D., eds.: Proceedings of Eurographics Symposium on Geometry Processing. (2004) 93–102

Part II

Appendix A Create and fill the tetrahedron table

This appendix contains all scripts as developed in the project. It reflects the current state of implementation at December 20, 2006.

First an empty table is defined:

```
CREATE TABLE tetrahedron
(
      tetcode NVARCHAR2(100)
);
```

Then the table is filled with the results from the tetrahedronisation:

```
LOAD DATA
INFILE 'data/miniset.data'
APPEND
INTO TABLE tetrahedron
fields terminated by ' '
( tetcode )
```

Appendix B Creating functions and procedures

Note the following general assumptions:

- All coordinates are assumed to be positive integers
- Coded tetrahedrons are not signed (orientation is changed (if necessary) by an odd permutation)
- Coded triangles and edges are signed (and ordered (result of procedures) to have the same permutation as dual, only sign differences)
- An equal number of digits is used for all coordinates (e.g. 3 for x1, 3 for y1, 3 for z1, 3 for x2,...)
  (this number is refered to as codelength (in example above codelength=3))
- Vectors are similarly coded and terms are signed (sign for each term is NOT part of codelength)

```
==================================================================
                        CREATION OF FUNCTIONS

==================================================================

CREATE OR REPLACE FUNCTION cayleymengerdeterminant5x5(input1
NUMBER, input2 NUMBER, input3 NUMBER, input4 NUMBER, input5
NUMBER, input6 NUMBER)
   RETURN NUMBER
   AS LANGUAGE JAVA
      NAME 'Friso.cayleymengerdeterminant5x5(double, double,
             double, double, double, double) return double';
/

CREATE OR REPLACE FUNCTION cayleymengerdeterminant4x4(input1
NUMBER, input2 NUMBER, input3 NUMBER)
   RETURN NUMBER
   AS LANGUAGE JAVA
      NAME 'Friso.cayleymengerdeterminant4x4(double, double,
              double) return double';
/
```

```
===================================================================
                        CREATION OF PROCEDURES

===================================================================


======
Acts on TETRAHEDRON

Procedure name    : getcenterpoint
Parameters  :
integer   - codelength (number of digits used for a coordinate (so
for xxxyyyzzzxxxyyyz... codelenght=3, not 9!)
nvarchar2 - tetcode (the coded tetrahedron in form
xxyyzzxxyyzzxxyyzzxxyyzzididid)
nvarchar2 - centerpoint (of tetrahedron, coded as xxyyzz. output)

Description : calculates centrepoint (rounding coordinates to in-
tegers again) of tetrahedron
======


CREATE OR REPLACE PROCEDURE getcenterpoint(codelength IN INTEGER,
tetcode IN NVARCHAR2, x OUT NUMBER, y OUT NUMBER, z OUT NUMBER)
IS
   coordsum NUMBER;
BEGIN
   FOR i IN 0..2
   LOOP
      coordsum := 0;

      FOR j IN 0..3
      LOOP
         coordsum := coordsum +
(SUBSTR(tetcode,1+i*codelength+j*3*codelength,codelength));
      END LOOP;

      IF (i=0) THEN
         x := coordsum/4;
      ELSIF (i=1) THEN
         y := coordsum/4;
      ELSIF (i=2) THEN
         z:= coordsum/4;
      END IF;
   END LOOP;

END;
/
```

```
======
Acts on TRIANGLE

Procedure name    : calculatenormal
Parameters  :
integer   - codelength (number of digits used for a coordinate (so
for xxxyyyzzzxxxyyyz... codelenght=3, not 9!)
nvarchar2 - tricode (the coded triangle in form
xxyyzzxxyyzzxxyyzzxxyyzzididid. postive sign required!)
nvarchar2 - normalvec (of triangle, coded as xxyyzz. output)

Description : calculates normal vector of signed triangle
======

CREATE OR REPLACE PROCEDURE calculatenormal(codelength IN INTEGER,
tricode IN NVARCHAR2, normalvec OUT NVARCHAR2)
IS
   sign NVARCHAR2(1);
   x1 NVARCHAR2(100);
   x2 NVARCHAR2(100);
   x3 NVARCHAR2(100);
   y1 NVARCHAR2(100);
   y2 NVARCHAR2(100);
   y3 NVARCHAR2(100);
   z1 NVARCHAR2(100);
   z2 NVARCHAR2(100);
   z3 NVARCHAR2(100);
   normx NVARCHAR2(100);
   normy NVARCHAR2(100);
   normz NVARCHAR2(100);
   absnormx NVARCHAR2(100);
   absnormy NVARCHAR2(100);
   absnormz NVARCHAR2(100);
BEGIN
   sign := (SUBSTR(tricode,1,1));
   x1 := (SUBSTR(tricode,2,codelength));
   x2 := (SUBSTR(tricode,2+3*codelength,codelength));
   x3 := (SUBSTR(tricode,2+6*codelength,codelength));
   y1 := (SUBSTR(tricode,2+codelength,codelength));
   y2 := (SUBSTR(tricode,2+4*codelength,codelength));
   y3 := (SUBSTR(tricode,2+7*codelength,codelength));
   z1 := (SUBSTR(tricode,2+2*codelength,codelength));
   z2 := (SUBSTR(tricode,2+5*codelength,codelength));
   z3 := (SUBSTR(tricode,2+8*codelength,codelength));
   normx := (y2-y1)*(z3-z1)-(z2-z1)*(y3-y1);
   normy := (z2-z1)*(x3-x1)-(x2-x1)*(z3-z1);
   normz := (x2-x1)*(y3-y1)-(y2-y1)*(x3-x1);

   IF (normx<0) THEN
      absnormx := (SUBSTR(normx,2));
      WHILE (length(absnormx)<codelength)
      LOOP
         absnormx := '0'||absnormx;
      END LOOP;
      IF (sign='+') THEN
         normx := '-'||absnormx;
      ELSE
         normx := '+'||absnormx;
      END IF;
   ELSE
      WHILE (length(normx)<codelength)
      LOOP
```

```
            normx := '0'||normx;
        END LOOP;
        normx := sign||normx;
    END IF;

    IF (normy<0) THEN
        absnormy := (SUBSTR(normy,2));
        WHILE (length(absnormy)<codelength)
        LOOP
            absnormy := '0'||absnormy;
        END LOOP;
        IF (sign='+') THEN
            normy := '-'||absnormy;
        ELSE
            normy := '+'||absnormy;
        END IF;
    ELSE
        WHILE (length(normy)<codelength)
        LOOP
            normy := '0'||normy;
        END LOOP;
        normy := sign||normy;
    END IF;

    IF (normz<0) THEN
        absnormz := (SUBSTR(normz,2));
        WHILE (length(absnormz)<codelength)
        LOOP
            absnormz := '0'||absnormz;
        END LOOP;
        IF (sign='+') THEN
            normz := '-'||absnormz;
        ELSE
            normz := '+'||absnormz;
        END IF;
    ELSE
        WHILE (length(normz)<codelength)
        LOOP
            normz := '0'||normz;
        END LOOP;
        normz := sign||normz;
    END IF;

    normalvec := normx||normy||normz;
END;
/
```

```
======
Acts on 3D VECTORS

Procedure name    : anglebetweenvectors
Parameters  :
integer   - codelength (number of digits used for a coordinate (so
for xxxyyyzzzxxxyyyz... codelenght=3, not 9!)
nvarchar2 - normvec (normal vector, coded vector in format +xxx-
yyy-zzz) (3D vectors only!)
nvarchar2 - veccode1 (difference vector between one of the triang-
le points and opposite point
angle     - number (angle between the two vectors in radians!)

Description : calculates the angle between two 3D vectors. The re-
sulting angle is in radians
======


CREATE OR REPLACE PROCEDURE anglebetweenvectors(codelength IN
INTEGER, normcode IN NVARCHAR2, veccode1 IN NVARCHAR2,
                                                   angle OUT NUMBER)
IS
   dotproduct INTEGER;
   length1 INTEGER;
   length2 INTEGER;

BEGIN
   dotproduct := 0;
   length1 := 0;
   length2 := 0;

   FOR i IN 0..2
   LOOP
      dotproduct := dotproduct + (sub-
str(normcode,1+i*(codelength+1),codelength+1))*(substr(veccode1,1+
i*(codelength+1),codelength+1));
      length1 := length1 + po-
wer(substr(normcode,1+i*(codelength+1),codelength+1),2);
      length2 := length2 + po-
wer(substr(veccode1,1+i*(codelength+1),codelength+1),2);
   END LOOP;

   angle := ACOS(dotproduct/((SQRT(length1))*(SQRT(length2))));
END;
/
```

```
======
Acts on TETRAHEDRON

Procedure name    : checkorientation
Parameters  :
integer   - codelength (number of digits used for a coordinate (so
for xxxyyyzzzxxxyyyz... codelenght=3, not 9!)
nvarchar2 - tetcode (the coded tetrahedron in form
xxyyzzxxyyzzxxyyzzxxyyzzidididid)
number    - isoutwards (indicates whether the normals of the boun-
dary faces of a tetrahedron point outwards (1) or inwards (0)

Description : supposes poincare tetrahedron, so orientation of all
four boundary triangles is identical (either all inwards or all
outwards) of tetrahedron v0v1v2v3 it takes boundary v1v2v3
orientation is positive). the normal vector of this triangle is
calculated. a second vector is constructed, from v1 to v4. The an-
gle between these two vectors is calculated. if it is smaller than
90 degrees (0.5*PI) the orientation is inwards, otherwise outwards
======


CREATE OR REPLACE PROCEDURE checkorientation(codelength IN
INTEGER, tetcode IN NVARCHAR2, isoutwards IN OUT NUMBER)
IS
   angle NUMBER;
   tricode NVARCHAR2(100);
   veccode1 NVARCHAR2(100);
   normalvec NVARCHAR2(100);
   diffchar NVARCHAR2(100);
   absdiffchar NVARCHAR2(100);
   oppositepoint NVARCHAR2(100);
   diff NUMBER;
BEGIN
   oppositepoint := SUBSTR(tetcode,1,3*codelength);
   tricode := '+'||SUBSTR(tetcode,1+3*codelength);
   calculatenormal(codelength,tricode,normalvec);
   veccode1 := '';

   FOR i IN 0..2
   LOOP
      diff := (SUBSTR(oppositepoint,1+i*codelength,codelength))-
(SUBSTR(tricode,2+i*codelength,codelength));
      IF (diff<0) THEN
         diffchar := TO_CHAR(diff);
         absdiffchar := SUBSTR(diffchar,2);

         WHILE (LENGTH(absdiffchar)<codelength)
         LOOP
            absdiffchar := '0'||absdiffchar;
         END LOOP;
         veccode1 := veccode1||'-'||absdiffchar;

      ELSE
         absdiffchar := TO_CHAR(diff);

         WHILE (LENGTH(absdiffchar)<codelength)
         LOOP
            absdiffchar := '0'||absdiffchar;
         END LOOP;
         veccode1 := veccode1||'+'||absdiffchar;
```

```
      END IF;
   END LOOP;

   anglebetweenvectors(codelength,normalvec,veccode1,angle);

   IF (angle>(3.14159265358979323846264338327950/2)) THEN
      isoutwards := 1;
   ELSE
      isoutwards := 0;
   END IF;
END;
/
```

```
======
Acts on TETRAHEDRON

Procedure name    : permutation34
Parameters  :
integer   - codelength (input. number of digits used for a coordi-
nate (so for xxxyyyzzzxxxyyyz... codelenght=3, not 9!)
nvarchar2 - tetcode (coded tetrahedron)
nvarchar2 - tetcodeperm (coded permutated tetrahedron)

Description : Performs a permutation of the last two vertices in
the tetrahedron code. This single permutation causes a chance of
orientation (i.e. inwards instead of outwards or vice versa) of
the tetrahedron. For simplexes of dimension < 3 usage of signed
ordered encodings is preferred above permutations (as in that case
one cannot be sure which of the equivalent permutations is used)
======


CREATE OR REPLACE PROCEDURE permutation34(codelength IN INTEGER,
tetcode IN NVARCHAR2, tetcodeperm OUT NVARCHAR2)
IS
BEGIN
   tetcodeperm :=
(SUBSTR(tetcode,1,6*codelength))||(SUBSTR(tetcode,1+9*codelength,3
*codelength))||

(SUBSTR(tetcode,1+6*codelength,3*codelength))||(SUBSTR(tetcode,1+1
2*codelength));
END;
/
```

```
======
Acts on TRIANGLE

Procedure name    : ordertriangle
Parameters  : \
integer   - codelength (input. number of digits used for a coordi-
nate (so for xxxyyyzzzxxxyyyz... codelenght=3, not 9!)
nvarchar2 - tricode (code triangle: signxxyyzzxxyy...ID)
nvarchar2 - ordered tricode (coded trianglke signxxyyzzxxyy..ID)
with codevertex1<codevertex2<codevertex3

Description : Orders a triangle based on the coordinate code of
each vertex, from small to large, in order to ensure that each
triangle and its dual have the same code (apart from the sign) and
not one of thheir equivalent permutations
======


CREATE OR REPLACE PROCEDURE ordertriangle(codelength IN INTEGER,
tricode IN NVARCHAR2, orderedtricode OUT NVARCHAR2)
IS
   sign NVARCHAR2(1);
   a NVARCHAR2(100);
   b NVARCHAR2(100);
   c NVARCHAR2(100);
   id NVARCHAR2(100);
BEGIN
   sign := (SUBSTR(tricode,1,1));
   a := (SUBSTR(tricode,2,3*codelength));
   b := (SUBSTR(tricode,2+3*codelength,3*codelength));
   c := (SUBSTR(tricode,2+6*codelength,3*codelength));
   id := (SUBSTR(tricode,2+9*codelength));

   IF (a<b) THEN
      IF (b<c) THEN
         orderedtricode := tricode;
      ELSE
         IF (a<c) THEN
            IF (sign = '+') THEN
               orderedtricode := '-'||a||c||b||id;
            ELSE
               orderedtricode := '+'||a||c||b||id;
            END IF;
         ELSE
            orderedtricode := sign||c||a||b||id;
         END IF;
      END IF;
   ELSE
      IF (b<c) THEN
         IF (a<c) THEN
            IF (sign = '+') THEN
               orderedtricode := '-'||b||a||c||id;
            ELSE
               orderedtricode := '+'||b||a||c||id;
            END IF;
         ELSE
            orderedtricode := sign||b||c||a||id;
         END IF;
      ELSE
         IF (sign = '+') THEN
            orderedtricode := '-'||c||b||a||id;
```

```
         ELSE
            orderedtricode := '+'||c||b||a||id;
         END IF;
      END IF;
   END IF;

END;
/
```

```
======
Acts on TETRAHEDRON

Procedure name    : sorttetrahedron
Parameters  :
integer   - codelength (input. number of digits used for a coordi-
nate (so for xxxyyyzzzxxxyyyz... codelenght=3, not 9!)
nvarchar2 - tetcode (coded tetrahedron: nxxyyzzxxyy...ID)
nvarchar2 - sorted tetcode (coded tetrahedron xxyyzzxxyy..ID) with
codevertex1<codevertex2<codevertex3

Description : Orders a tetrahedron based on the coordinate code of
each vertex, from small to large, in order to ensure that each
triangle and its dual have the same code (apart from the sign) and
not one of thheir equivalent permutations
======




CREATE OR REPLACE PROCEDURE sorttetrahedron(codelength IN INTEGER,
tetcode IN NVARCHAR2, sortedtetcode OUT NVARCHAR2)
IS
   a NVARCHAR2(100);
   b NVARCHAR2(100);
   c NVARCHAR2(100);
   d NVARCHAR2(100);
   id NVARCHAR2(100);
BEGIN
   a := (SUBSTR(tetcode,1,3*codelength));
   b := (SUBSTR(tetcode,1+3*codelength,3*codelength));
   c := (SUBSTR(tetcode,1+6*codelength,3*codelength));
   d := (SUBSTR(tetcode,1+9*codelength,3*codelength));
   id := (SUBSTR(tetcode,1+12*codelength));

   IF (a<b) THEN
      IF (b<c) THEN
         IF (b<d) THEN
            IF (c<d) THEN
               sortedtetcode := a||b||c||d||id;
            ELSE
               sortedtetcode := a||b||d||c||id;
            END IF;
         ELSE
            IF (a<d) THEN
               sortedtetcode := a||d||b||c||id;
            ELSE
               sortedtetcode := d||a||b||c||id;
            END IF;
         END IF;
      ELSE
         IF (a<c) THEN
            IF (c<d) THEN
               IF (b<d) THEN
                  sortedtetcode := a||c||b||d||id;
               ELSE
                  sortedtetcode := a||c||d||b||id;
               END IF;
            ELSE
               IF (a<d) THEN
                  sortedtetcode := a||d||c||b||id;
               ELSE
```

```
                  sortedtetcode := d||a||c||b||id;
               END IF;
            END IF;
         ELSE
            IF (a<d) THEN
               IF (b<d) THEN
                  sortedtetcode := c||a||b||d||id;
               ELSE
                  sortedtetcode := c||a||d||b||id;
               END IF;
            ELSE
               IF (c<d) THEN
                  sortedtetcode := c||d||a||b||id;
               ELSE
                  sortedtetcode := d||c||a||b||id;
               END IF;
            END IF;
         END IF;
      END IF;
   ELSE
      IF (b<c) THEN
         IF (a<c) THEN
            IF (a<d) THEN
               IF (c<d) THEN
                  sortedtetcode := b||a||c||d||id;
               ELSE
                  sortedtetcode := b||a||d||c||id;
               END IF;
            ELSE
               IF (b<d) THEN
                  sortedtetcode := b||d||a||c||id;
               ELSE
                  sortedtetcode := d||b||a||c||id;
               END IF;
            END IF;
         ELSE
            IF (c<d) THEN
               IF (a<d) THEN
                  sortedtetcode := b||c||a||d||id;
               ELSE
                  sortedtetcode := b||c||d||a||id;
               END IF;
            ELSE
               IF (b<d) THEN
                  sortedtetcode := b||d||c||a||id;
               ELSE
                  sortedtetcode := d||b||c||a||id;
               END IF;
            END IF;
         END IF;
      ELSE
         IF (b<d) THEN
            IF (a<d) THEN
               sortedtetcode := c||b||a||d||id;
            ELSE
               sortedtetcode := c||b||d||a||id;
            END IF;
         ELSE
            IF (c<d) THEN
               sortedtetcode := c||d||b||a||id;
            ELSE
```

```
                sortedtetcode := d||c||b||a||id;
            END IF;
         END IF;
      END IF;
   END IF;

END;
/
```

```
======
Acts on TETRAHEDRON

Procedure name    : deriveboundarytriangles
Parameters  :
integer   - codelength (input. number of digits used for a coordi-
nate (so for xxxyyyzzzxxxyyyz... codelenght=3, not 9!)
nvarchar2 - tetcode (coded tetrahedron: xxyyzzxxyy...ID)
nvarchar2 - tricode1 (coded triangle: signxxyyzzxxyy..ID) with co-
devertex1<codevertex2<codevertex3
nvarchar2 - tricode2 (coded triangle: signxxyyzzxxyy..ID) with co-
devertex1<codevertex2<codevertex3
nvarchar2 - tricode3 (coded triangle: signxxyyzzxxyy..ID) with co-
devertex1<codevertex2<codevertex3
nvarchar2 - tricode4 (coded triangle: signxxyyzzxxyy..ID) with co-
devertex1<codevertex2<codevertex3

Description : Derives the four boundary triangles of a tetrahedron
by applying the boundary operator of the simplical complex homolo-
gy. All resulting triangles are ordered (v0<v1<v2) and signed(+/-)
======

CREATE OR REPLACE PROCEDURE deriveboundarytriangles(codelength IN
INTEGER, tetcode IN NVARCHAR2, tricode1 OUT NVARCHAR2,
                                            tricode2 OUT NVARCHAR2,
tricode3 OUT NVARCHAR2, tricode4 OUT NVARCHAR2)
IS
   a NVARCHAR2(100);
   b NVARCHAR2(100);
   c NVARCHAR2(100);
   d NVARCHAR2(100);
   id NVARCHAR2(100);
BEGIN
   a := (SUBSTR(tetcode,1,3*codelength));
   b := (SUBSTR(tetcode,1+3*codelength,3*codelength));
   c := (SUBSTR(tetcode,1+6*codelength,3*codelength));
   d := (SUBSTR(tetcode,1+9*codelength,3*codelength));
   id := (SUBSTR(tetcode,1+12*codelength));
   ordertriangle(codelength,'+'||b||c||d||id, tricode1);
   ordertriangle(codelength,'-'||a||c||d||id, tricode2);
   ordertriangle(codelength,'+'||a||b||d||id, tricode3);
   ordertriangle(codelength,'-'||a||b||c||id, tricode4);
END;
/


CREATE OR REPLACE FUNCTION deriveboundarytriangle1(codelength
INTEGER, tetcode NVARCHAR2)
RETURN NVARCHAR2 DETERMINISTIC
IS
   a NVARCHAR2(100);
   b NVARCHAR2(100);
   c NVARCHAR2(100);
   d NVARCHAR2(100);
   id NVARCHAR2(100);
   tricode NVARCHAR2(100);
BEGIN
   a := (SUBSTR(tetcode,1,3*codelength));
   b := (SUBSTR(tetcode,1+3*codelength,3*codelength));
   c := (SUBSTR(tetcode,1+6*codelength,3*codelength));
   d := (SUBSTR(tetcode,1+9*codelength,3*codelength));
```

```
   id := (SUBSTR(tetcode,1+12*codelength));
   ordertriangle(codelength,'+'||b||c||d||id, tricode);
   RETURN tricode;
END;
/



CREATE OR REPLACE FUNCTION deriveboundarytriangle2(codelength
INTEGER, tetcode NVARCHAR2)
RETURN NVARCHAR2 DETERMINISTIC
IS
   a NVARCHAR2(100);
   b NVARCHAR2(100);
   c NVARCHAR2(100);
   d NVARCHAR2(100);
   id NVARCHAR2(100);
   tricode NVARCHAR2(100);
BEGIN
   a := (SUBSTR(tetcode,1,3*codelength));
   b := (SUBSTR(tetcode,1+3*codelength,3*codelength));
   c := (SUBSTR(tetcode,1+6*codelength,3*codelength));
   d := (SUBSTR(tetcode,1+9*codelength,3*codelength));
   id := (SUBSTR(tetcode,1+12*codelength));
   ordertriangle(codelength,'-'||a||c||d||id, tricode);
   RETURN tricode;
END;
/

CREATE OR REPLACE FUNCTION deriveboundarytriangle3(codelength
INTEGER, tetcode NVARCHAR2)
RETURN NVARCHAR2 DETERMINISTIC
IS
   a NVARCHAR2(100);
   b NVARCHAR2(100);
   c NVARCHAR2(100);
   d NVARCHAR2(100);
   id NVARCHAR2(100);
   tricode NVARCHAR2(100);
BEGIN
   a := (SUBSTR(tetcode,1,3*codelength));
   b := (SUBSTR(tetcode,1+3*codelength,3*codelength));
   c := (SUBSTR(tetcode,1+6*codelength,3*codelength));
   d := (SUBSTR(tetcode,1+9*codelength,3*codelength));
   id := (SUBSTR(tetcode,1+12*codelength));
   ordertriangle(codelength,'+'||a||b||d||id, tricode);
   RETURN tricode;
END;
/

CREATE OR REPLACE FUNCTION deriveboundarytriangle4(codelength
INTEGER, tetcode NVARCHAR2)
RETURN NVARCHAR2 DETERMINISTIC
IS
   a NVARCHAR2(100);
   b NVARCHAR2(100);
   c NVARCHAR2(100);
   d NVARCHAR2(100);
   id NVARCHAR2(100);
   tricode NVARCHAR2(100);
BEGIN
   a := (SUBSTR(tetcode,1,3*codelength));
```

```
   b := (SUBSTR(tetcode,1+3*codelength,3*codelength));
   c := (SUBSTR(tetcode,1+6*codelength,3*codelength));
   d := (SUBSTR(tetcode,1+9*codelength,3*codelength));
   id := (SUBSTR(tetcode,1+12*codelength));
   ordertriangle(codelength,'-'||a||b||c||id, tricode);
   RETURN tricode;
END;
/
```

```
======
Acts on TRIANGLE

Procedure name    : deriveboundaryedges
Parameters  :
integer   - codelength (input. number of digits used for a coordi-
nate (so for xxxyyyzzzxxxyyyz... codelenght=3, not 9!)
nvarchar2 - tricode (coded triangle: signxxyyzzxxyy...ID)
nvarchar2 - edcode1 (coded edge: signxxyyzzxxyyzzID) with codever-
tex1<codevertex2
nvarchar2 - edcode2 (coded edge: signxxyyzzxxyyzzID) with codever-
tex1<codevertex2
nvarchar2 - edcode3 (coded edge: signxxyyzzxxyyzzID) with codever-
tex1<codevertex2

Description : Derives the three boundary edges of a signed triang-
le by applying the boundary operator of the simplical complex
homology. All resulting edges are ordered (v0<v1, as the triangles
are ordered) and signed(+/-)
======

CREATE OR REPLACE PROCEDURE deriveboundaryedges(codelength IN
INTEGER, tricode IN NVARCHAR2, edcode1 OUT NVARCHAR2,
                                          edcode2 OUT NVARCHAR2,
edcode3 OUT NVARCHAR2)
IS
   sign NVARCHAR2(1);
   a NVARCHAR2(100);
   b NVARCHAR2(100);
   c NVARCHAR2(100);
   id NVARCHAR2(100);
BEGIN
   sign := (SUBSTR(tricode,1,1));
   a := (SUBSTR(tricode,2,3*codelength));
   b := (SUBSTR(tricode,2+3*codelength,3*codelength));
   c := (SUBSTR(tricode,2+6*codelength,3*codelength));
   id := (SUBSTR(tricode,2+9*codelength));
   IF (sign='+') THEN
      edcode1 := '+'||b||c;
      edcode2 := '-'||a||c;
      edcode3 := '+'||a||b;
   ELSE
      edcode1 := '-'||b||c;
      edcode2 := '+'||a||c;
      edcode3 := '-'||a||b;
   END IF;
END;
/


CREATE OR REPLACE FUNCTION deriveabsboundaryedge1(codelength
INTEGER, tricode NVARCHAR2)
RETURN NVARCHAR2 DETERMINISTIC
IS
   b NVARCHAR2(100);
   c NVARCHAR2(100);
   edcode1 NVARCHAR2(100);
BEGIN
   b := (SUBSTR(tricode,2+3*codelength,3*codelength));
   c := (SUBSTR(tricode,2+6*codelength,3*codelength));
   edcode1 := b||c;
   RETURN edcode1;
```

```
END;
/


CREATE OR REPLACE FUNCTION deriveabsboundaryedge2(codelength
INTEGER, tricode NVARCHAR2)
RETURN NVARCHAR2 DETERMINISTIC
IS
   a NVARCHAR2(100);
   c NVARCHAR2(100);
   edcode2 NVARCHAR2(100);
BEGIN
   a := (SUBSTR(tricode,2,3*codelength));
   c := (SUBSTR(tricode,2+6*codelength,3*codelength));
   edcode2 := a||c;
   RETURN edcode2;
END;
/


CREATE OR REPLACE FUNCTION deriveabsboundaryedge3(codelength
INTEGER, tricode NVARCHAR2)
RETURN NVARCHAR2 DETERMINISTIC
IS
   a NVARCHAR2(100);
   b NVARCHAR2(100);
   edcode3 NVARCHAR2(100);
BEGIN
   a := (SUBSTR(tricode,2,3*codelength));
   b := (SUBSTR(tricode,2+3*codelength,3*codelength));
   edcode3 := a||b;
   RETURN edcode3;
END;
/
```

```
======
Acts on EDGE

Procedure name    : deriveboundarynodes
Parameters  :
integer   - codelength (input. number of digits used for a coordi-
nate (so for xxxyyyzzzxxxyyyz... codelenght=3, not 9!)
nvarchar2 - edcode (coded edge: xxyyzzxxyy..z or coded signed eg-
de: signxxyy..)
nvarchar2 - nodecode1 (coded node: xxyyzz)
nvarchar2 - nodecode2 (coded node: xxyyzz)

Description : Derives the two boundary nodes of a signed (or not)
edge by applying the boundary operator of the simplical complex
homology.
======


CREATE OR REPLACE PROCEDURE deriveboundarynodes(codelength IN
INTEGER, edcode IN NVARCHAR2, nodecode1 OUT NVARCHAR2,
                                          nodecode2 OUT NVARCHAR2)
IS
BEGIN
   IF (SUBSTR(edcode,1,1)='+') OR (SUBSTR(edcode,1,1)='-') THEN
      nodecode1 := (SUBSTR(edcode,2,3*codelength));
      nodecode2 := (SUBSTR(edcode,2+3*codelength,3*codelength));
   ELSE
      nodecode1 := (SUBSTR(edcode,1,3*codelength));
      nodecode2 := (SUBSTR(edcode,1+3*codelength,3*codelength));
   END IF;
END;
/


CREATE OR REPLACE FUNCTION deriveboundarynode1(codelength INTEGER,
edcode NVARCHAR2)
RETURN NVARCHAR2 DETERMINISTIC
IS
BEGIN
   IF (SUBSTR(edcode,1,1)='+') OR (SUBSTR(edcode,1,1)='-') THEN
      RETURN (SUBSTR(edcode,2,3*codelength));
   ELSE
      RETURN (SUBSTR(edcode,1,3*codelength));
   END IF;
END;
/


CREATE OR REPLACE FUNCTION deriveboundarynode2(codelength INTEGER,
edcode NVARCHAR2)
RETURN NVARCHAR2 DETERMINISTIC
IS
BEGIN
   IF (SUBSTR(edcode,1,1)='+') OR (SUBSTR(edcode,1,1)='-') THEN
      RETURN (SUBSTR(edcode,2+3*codelength,3*codelength));
   ELSE
      RETURN (SUBSTR(edcode,1+3*codelength,3*codelength));
   END IF;
END;
/
```

```
======
Acts on TETRAHEDRON

Procedure name    : gettetrahedronmbb
Parameters  :
integer   - codelength (input. number of digits used for a coordi-
nate (so for xxxyyyzzzxxxyyyz... codelenght=3, not 9!)
nvarchar2 - tetcode (coded tetrahedron: xxyyzzxxyy...ID)
nvarchar2 - mbbcode (coded minimum bounding box: minxminyminzmaxx-
maxymaxz)

Description : Derives the minimum bounding box of a tetrahedron by
selecting the min x, y, z and max x, y, z.
======


CREATE OR REPLACE PROCEDURE gettetrahedronmbb(codelength IN
INTEGER, tetcode IN NVARCHAR2, mbbcode OUT NVARCHAR2)
IS
   a NVARCHAR2(100);
   tempmin NVARCHAR2(100);
   tempmax NVARCHAR2(100);
   defmin NVARCHAR2(100);
   defmax NVARCHAR2(100);
BEGIN
   FOR j IN 0..2
   LOOP
      tempmin := (SUBSTR(tetcode,1+j*codelength,codelength));
      tempmax := (SUBSTR(tetcode,1+j*codelength,codelength));
      FOR i IN 1..3
      LOOP
        a :=
(SUBSTR(tetcode,1+i*3*codelength+j*codelength,codelength));
        IF (a<tempmin) THEN
           tempmin := a;
        END IF;
        IF (a>tempmax) THEN
           tempmax := a;
        END IF;
      END LOOP;
      defmin := defmin||tempmin;
      defmax := defmax||tempmax;
   END LOOP;
   mbbcode := defmin||defmax;
END;
/
```

```
======
Acts on TRIANGLE

Procedure name    : gettrianglembb
Parameters  :
integer   - codelength (input. number of digits used for a coordi-
nate (so for xxxyyyzzzxxxyyyz... codelenght=3, not 9!)
nvarchar2 - tricode (coded triangle: signxxyyzzxxyy...ID)
nvarchar2 - mbbcode (coded minimum bounding box: minxminyminzmaxx-
maxymaxz)

Description : Derives the minimum bounding box of a tetrahedron by
selecting the min x, y, z and max x, y, z.
======


CREATE OR REPLACE PROCEDURE gettrianglembb(codelength IN INTEGER,
tricode IN NVARCHAR2, mbbcode OUT NVARCHAR2)
IS
   a NVARCHAR2(100);
   tempmin NVARCHAR2(100);
   tempmax NVARCHAR2(100);
   defmin NVARCHAR2(100);
   defmax NVARCHAR2(100);
BEGIN
   FOR j IN 0..2
   LOOP
      tempmin := (SUBSTR(tricode,2+j*codelength,codelength));
      tempmax := (SUBSTR(tricode,2+j*codelength,codelength));
      FOR i IN 1..2
      LOOP
        a :=
(SUBSTR(tricode,2+i*3*codelength+j*codelength,codelength));
        IF (a<tempmin) THEN
           tempmin := a;
        END IF;
        IF (a>tempmax) THEN
           tempmax := a;
        END IF;
      END LOOP;
      defmin := defmin||tempmin;
      defmax := defmax||tempmax;
   END LOOP;
   mbbcode := defmin||defmax;
END;
/
```

```
======
Acts on TETRAHEDRON

Procedure name    : tetedgelengthsquare
Parameters  :
integer   - codelength (input. number of digits used for a coordi-
nate (so for xxxyyyzzzxxxyyyz... codelenght=3, not 9!)
nvarchar2 - tetcode (coded tetrahedron: xxyyzzxxyy...ID)
number    - a (square of length of edge v1v2)
number    - b (square of length of edge v1v3)
number    - c (square of length of edge v1v4)
number    - d (square of length of edge v2v3)
number    - e (square of length of edge v2v4)
number    - f (square of length of edge v3v4)

Description : Calculates the square of the lengths of the six ed-
ges of a tetrahedron. Used as input for Cayley-Menger determinant
======


CREATE OR REPLACE PROCEDURE tetedgelengthsquare(codelength IN
INTEGER, tetcode IN NVARCHAR2, a OUT NUMBER, b OUT NUMBER, c OUT
NUMBER,
                                                      d OUT NUMBER,
e OUT NUMBER, f OUT NUMBER)
IS
   x1 NVARCHAR2(100);
   y1 NVARCHAR2(100);
   z1 NVARCHAR2(100);
   x2 NVARCHAR2(100);
   y2 NVARCHAR2(100);
   z2 NVARCHAR2(100);
   x3 NVARCHAR2(100);
   y3 NVARCHAR2(100);
   z3 NVARCHAR2(100);
   x4 NVARCHAR2(100);
   y4 NVARCHAR2(100);
   z4 NVARCHAR2(100);
BEGIN
   x1 := (SUBSTR(tetcode,1,codelength));
   y1 := (SUBSTR(tetcode,1+codelength,codelength));
   z1 := (SUBSTR(tetcode,1+2*codelength,codelength));
   x2 := (SUBSTR(tetcode,1+3*codelength,codelength));
   y2 := (SUBSTR(tetcode,1+4*codelength,codelength));
   z2 := (SUBSTR(tetcode,1+5*codelength,codelength));
   x3 := (SUBSTR(tetcode,1+6*codelength,codelength));
   y3 := (SUBSTR(tetcode,1+7*codelength,codelength));
   z3 := (SUBSTR(tetcode,1+8*codelength,codelength));
   x4 := (SUBSTR(tetcode,1+9*codelength,codelength));
   y4 := (SUBSTR(tetcode,1+10*codelength,codelength));
   z4 := (SUBSTR(tetcode,1+11*codelength,codelength));
   a := power((x2-x1),2)+power((y2-y1),2)+power((z2-z1),2);
   b := power((x3-x1),2)+power((y3-y1),2)+power((z3-z1),2);
   c := power((x4-x1),2)+power((y4-y1),2)+power((z4-z1),2);
   d := power((x3-x2),2)+power((y3-y2),2)+power((z3-z2),2);
   e := power((x4-x2),2)+power((y4-y2),2)+power((z4-z2),2);
   f := power((x4-x3),2)+power((y4-y3),2)+power((z4-z3),2);
END;
/
```

```
======
Acts on TRIANGLE

Procedure name    : triedgelengthsquare
Parameters  :
integer   - codelength (input. number of digits used for a coordi-
nate (so for xxxyyyzzzxxxyyyz... codelenght=3, not 9!)
nvarchar2 - tricode (coded triangle: signxxyyzzxxyy...ID)
number    - a (square of length of edge v1v2)
number    - b (square of length of edge v1v3)
number    - c (square of length of edge v2v3)

Description : Calculates the square of the lengths of the six ed-
ges of a tetrahedron. Used as input for Cayley-Menger determinant
======


CREATE OR REPLACE PROCEDURE triedgelengthsquare(codelength IN
INTEGER, tricode IN NVARCHAR2, a OUT NUMBER, b OUT NUMBER, c OUT
NUMBER)
IS
   x1 NVARCHAR2(100);
   y1 NVARCHAR2(100);
   z1 NVARCHAR2(100);
   x2 NVARCHAR2(100);
   y2 NVARCHAR2(100);
   z2 NVARCHAR2(100);
   x3 NVARCHAR2(100);
   y3 NVARCHAR2(100);
   z3 NVARCHAR2(100);
BEGIN
   x1 := (SUBSTR(tricode,2,codelength));
   y1 := (SUBSTR(tricode,2+codelength,codelength));
   z1 := (SUBSTR(tricode,2+2*codelength,codelength));
   x2 := (SUBSTR(tricode,2+3*codelength,codelength));
   y2 := (SUBSTR(tricode,2+4*codelength,codelength));
   z2 := (SUBSTR(tricode,2+5*codelength,codelength));
   x3 := (SUBSTR(tricode,2+6*codelength,codelength));
   y3 := (SUBSTR(tricode,2+7*codelength,codelength));
   z3 := (SUBSTR(tricode,2+8*codelength,codelength));
   a := power((x2-x1),2)+power((y2-y1),2)+power((z2-z1),2);
   b := power((x3-x1),2)+power((y3-y1),2)+power((z3-z1),2);
   c := power((x3-x2),2)+power((y3-y2),2)+power((z3-z2),2);
END;
/
```

```
======
Acts on TRIANGLE AND TETRAHEDRON

Procedure name    : simplexvolume
Parameters  :
integer   - codelength (input. number of digits used for a coordi-
nate (so for xxxyyyzzzxxxyyyz... codelenght=3, not 9!)
nvarchar2 - simplexcode (coded triangle: signxxyyzzxxyy...ID or
coded tetrahedronxxyyzzxx...ID)
number    - simplexvolumea (i.e. volume of tetrahedron or area of
triangle)

Description : Calculates the volume of a 3- or 2-simplex, using
the Cayley-Menger determinant
======


CREATE OR REPLACE PROCEDURE simplexvolume(codelength IN INTEGER,
simplexcode IN NVARCHAR2, simplexvolume OUT NUMBER)
IS
   a NUMBER;
   b NUMBER;
   c NUMBER;
   d NUMBER;
   e NUMBER;
   f NUMBER;
   det NUMBER;
BEGIN
   IF (((SUBSTR(simplexcode,1,1)) = '+') OR
((SUBSTR(simplexcode,1,1)) = '-')) THEN
      triedgelengthsquare(codelength,simplexcode,a,b,c);
      det := cayleymengerdeterminant4x4(a,b,c);
      simplexvolume := SQRT(det/-16);
   ELSE
      tetedgelengthsquare(codelength,simplexcode,a,b,c,d,e,f);
      det := cayleymengerdeterminant5x5(a,b,c,d,e,f);
      simplexvolume := SQRT(det/288);
   END IF;
END;
/
```

```
======
Acts on TABLE called TETRAHEDRON

Procedure name    : tettableoutwards
Parameters  :
integer   - codelength (input. number of digits used for a coordi-
nate (so for xxxyyyzzzxxxyyyz... codelenght=3, not 9!)
number    - number of changed tetrahedrons (i.e. number of formar-
ly inwards oriented tetrahderons)
number    - number of unchanges tetrahedrons (i.e. number of te-
trahedrons already outwards oriented

Description : ckecks for all tetrahedrons in the tetrahedron table
whether they are oriented outwards. if not, the tetrahedrons are
replaced by an outwards oriented permutation (permutation of v0
and v1)
======

CREATE OR REPLACE PROCEDURE tettableoutwards(codelength IN
INTEGER, changes OUT NUMBER, nochanges OUT NUMBER)
IS
   CURSOR tetcur IS
      SELECT tetcode FROM tetrahedron FOR UPDATE;
   tetcode NVARCHAR2(100);
   currenttetcode NVARCHAR2(100);
   newtetcode NVARCHAR2(100);
   bool NUMBER;
   a NUMBER;
BEGIN
   dbms_output.put_line('ja');

   a := 0;
   changes := 0;
   nochanges := 0;
   OPEN tetcur;
   LOOP
      FETCH tetcur INTO tetcode;
      EXIT WHEN tetcur%notfound;
         dbms_output.put_line(tetcode);
      checkorientation(codelength,tetcode,bool);

      a:= a+1;

      dbms_output.put_line(bool);
      dbms_output.put_line(a);
      IF (bool = 0) THEN
         permutation34(codelength,tetcode,newtetcode);
         UPDATE tetrahedron SET tetcode=newtetcode WHERE CURRENT
OF tetcur;
         changes := changes+1;
      ELSE
         nochanges := nochanges+1;
      END IF;
   END LOOP;
   CLOSE tetcur;
END;
/
```

```
======
Acts on TABLE called TETRAHEDRON

Procedure name    : sorttettable
Parameters  :
integer   - codelength (input. number of digits used for a coordi-
nate (so for xxxyyyzzzxxxyyyz... codelenght=3, not 9!)
number    - number of changed tetrahedrons (i.e. number of unsor-
ted tetrahderons)
number    - number of unchanges tetrahedrons (i.e. number of te-
trahedrons already sorted correctly

Description : ckecks for all tetrahedrons in the tetrahedron table
whether the tetrahedron code is sorted (based on the coordinate
code of each vertex, from small to large, in order to ensure that
each  triangle and its dual have the same code (apart from the
sign) and not one of their equivalent permutations
======

CREATE OR REPLACE PROCEDURE sorttettable(codelength IN INTEGER,
changes OUT NUMBER, nochanges OUT NUMBER)
IS
   CURSOR tetcur1 IS
      SELECT tetcode FROM tetrahedron FOR UPDATE;
   tetcode NVARCHAR2(100);
   newtetcode NVARCHAR2(100);
   bool NUMBER;
   a NUMBER;
BEGIN
   a := 0;
   changes := 0;
   nochanges := 0;
   OPEN tetcur1;
   LOOP
      FETCH tetcur1 INTO tetcode;
      EXIT WHEN tetcur1%notfound;
      sorttetrahedron(codelength,tetcode,newtetcode);
      UPDATE tetrahedron SET tetcode=newtetcode WHERE CURRENT OF
tetcur1;
         changes := changes+1;
   END LOOP;
   CLOSE tetcur1;

END;
/
```

```
======
Acts on SIMPLEX with ID

Function name     : getobjectid
Parameters  :
integer   - codelength (input. number of digits used for a coordi-
nate (so for xxxyyyzzzxxxyyyz... codelenght=3, not 9!)
nvarchar2 - simplexcode (coded simplex: xxyyzzxxyy...ID)

Description : Returns object ID, to be used in SQL select state-
ments, for instance for finding all tetrahedrons or triangles of a
specific object.
======


CREATE OR REPLACE FUNCTION getobjectid(codelength INTEGER, sim-
plexcode NVARCHAR2)
RETURN NVARCHAR2 DETERMINISTIC
IS
BEGIN
   RETURN (SUBSTR(simplexcode, LENGTH(simplexcode)-codelength+1));

END;
/
```

```
======
Acts on SIMPLEX with ID

Function name     : removeobjectid
Parameters  :
integer   - codelength (input. number of digits used for a coordi-
nate (so for xxxyyyzzzxxxyyyz... codelenght=3, not 9!)
nvarchar2 - simplexcode (coded simplex: xxyyzzxxyy...ID)

Description : Returns simplexcode without object ID
======

CREATE OR REPLACE FUNCTION removeobjectid(codelength INTEGER, sim-
plexcode NVARCHAR2)
RETURN NVARCHAR2 DETERMINISTIC
IS
BEGIN
   RETURN (SUBSTR(simplexcode, 1, LENGTH(simplexcode)-
codelength));

END;
/
```

```
======
Acts on TABLE TETRAHEDRON and VIEWS TRIANGLE, EDGE and NODE

Function name     : validatestructure
Parameters  : nvarchar2 - result (result of validation)

Description : Performs several checks:
check1: Euler count (num of nodes - num of edges + num of triang-
les - (num of tetrahedrons + external volume)
check2: number of triangles equals number of unique triangles
check3: number of triangles equals four times number of tetrahe-
drons
======


CREATE OR REPLACE PROCEDURE validatestructure(result OUT
NVARCHAR2)
IS
   numnode NUMBER;
   numedge NUMBER;
   numtri NUMBER;
   numtri1 NUMBER;
   numtri2 NUMBER;
   numtet NUMBER;
   check1 NUMBER;
   check2 NUMBER;
   check3 NUMBER;

BEGIN
   SELECT COUNT(*) INTO numnode FROM node;
   SELECT COUNT(*) INTO numedge FROM edge;
   SELECT COUNT(DISTINCT ABS(removeobjectid(3,tricode))) INTO num-
tri FROM triangle;
   SELECT COUNT(*) INTO numtet FROM tetrahedron;
   check1 := numnode - numedge + numtri - (numtet+1);

   SELECT COUNT(*) INTO numtri1 FROM triangle;
   SELECT COUNT(DISTINCT tricode) INTO numtri2 FROM triangle;
   check2 := numtri1 - numtri2;

   check3 := numtri1 - 4*numtet;

   IF (check1+check2+check3 = 0) THEN
     result := 'Validation result: OK';
   ELSE
     IF (check1 <> 0) THEN
        result := 'Euler condition not satisfied';
     END IF;
     IF (check2 <> 0) THEN
        result := result||'-'||'Triangles not unique';
     END IF;
     IF (check3 <> 0) THEN
        result := result||'-'||'Error in deriving boundary triang-
les';
     END IF;
  END IF;

END;
/
```

```
======
Acts on TETRAHEDRON

Function name     : getneighnourtet1-4
Parameters  :
integer   - codelength (input. number of digits used for a coordi-
nate (so for xxxyyyzzzxxxyyyz... codelenght=3, not 9!)
nvarchar2 - tetcode (coded tetrahedron: xxyyzzxxyy...ID)

Description : Returns neighbouring tetrahderon by using the tri-
angle and dualtriangle view
======

CREATE OR REPLACE FUNCTION getneighbourtet1(codelength INTEGER,
tetcode NVARCHAR2)
RETURN NVARCHAR2 DETERMINISTIC
IS
   neighbourtet NVARCHAR2(100);
BEGIN
  select fromtetcode into neighbourtet from triangle
  where tricode=(select dt.dualtricode from dualtriangle dt where
dt.tricode=deriveboundarytriangle1(codelength,tetcode));
  RETURN neighbourtet;
END;
/

CREATE OR REPLACE FUNCTION getneighbourtet2(codelength INTEGER,
tetcode NVARCHAR2)
RETURN NVARCHAR2 DETERMINISTIC
IS
   neighbourtet NVARCHAR2(100);
BEGIN
  select fromtetcode into neighbourtet from triangle
  where tricode=(select dt.dualtricode from dualtriangle dt where
dt.tricode=deriveboundarytriangle2(codelength,tetcode));
  RETURN neighbourtet;
END;
/

CREATE OR REPLACE FUNCTION getneighbourtet3(codelength INTEGER,
tetcode NVARCHAR2)
RETURN NVARCHAR2 DETERMINISTIC
IS
   neighbourtet NVARCHAR2(100);
BEGIN
  select fromtetcode into neighbourtet from triangle
  where tricode=(select dt.dualtricode from dualtriangle dt where
dt.tricode=deriveboundarytriangle3(codelength,tetcode));
  RETURN neighbourtet;
END;
/

CREATE OR REPLACE FUNCTION getneighbourtet4(codelength INTEGER,
tetcode NVARCHAR2)
RETURN NVARCHAR2 DETERMINISTIC
IS
   neighbourtet NVARCHAR2(100);
BEGIN
  select fromtetcode into neighbourtet from triangle
  where tricode=(select dt.dualtricode from dualtriangle dt where
dt.tricode=deriveboundarytriangle4(codelength,tetcode));
  RETURN neighbourtet;  END; /
```

Appendix C Preparing the data structure

```
-- Rewrite tetrahedron table to achieve consistent outwards
orientation
var a number;
var b number;
exec sorttettable(3,:a,:b);


var c number;
var d number;
exec tettableoutwards(3,:c,:d);



-- Creating function based indexes to support view triangle

CREATE INDEX deriveboundarytriangle1_idx ON tetrahe-
dron(deriveboundarytriangle1(3,tetcode));
CREATE INDEX deriveboundarytriangle2_idx ON tetrahe-
dron(deriveboundarytriangle2(3,tetcode));
CREATE INDEX deriveboundarytriangle3_idx ON tetrahe-
dron(deriveboundarytriangle3(3,tetcode));
CREATE INDEX deriveboundarytriangle4_idx ON tetrahe-
dron(deriveboundarytriangle4(3,tetcode));



-- Creating view triangle with fields tricode and tetcode (tetra-
hedron the triangle is boundary of)

CREATE OR REPLACE VIEW triangle AS
   SELECT deriveboundarytriangle1(3,tetcode) tricode, tetcode
fromtetcode FROM tetrahedron
   UNION ALL
   SELECT deriveboundarytriangle2(3,tetcode) tricode, tetcode
fromtetcode FROM tetrahedron
   UNION ALL
   SELECT deriveboundarytriangle3(3,tetcode) tricode, tetcode
fromtetcode FROM tetrahedron
   UNION ALL
   SELECT deriveboundarytriangle4(3,tetcode) tricode, tetcode
fromtetcode FROM tetrahedron
;



-- Creating view dualtriangle (two columns: triangle and its dual,
both encoded including an inherited objectid

CREATE OR REPLACE VIEW dualtriangle AS
   SELECT t1.tricode tricode, t2.tricode dualtricode
   FROM triangle t1, triangle t2
   WHERE removeobjectid(3,t2.tricode) = -1
*removeobjectid(3,t1.tricode)
;
```

```
-- Creating function based indexes to support view tetrahedron-
neighbours

CREATE INDEX getneighbourtet1_idx ON tetrahe-
dron(getneighbourtet1(3,tetcode));
CREATE INDEX getneighbourtet2_idx ON tetrahe-
dron(getneighbourtet2(3,tetcode));
CREATE INDEX getneighbourtet3_idx ON tetrahe-
dron(getneighbourtet3(3,tetcode));
CREATE INDEX getneighbourtet4_idx ON tetrahe-
dron(getneighbourtet4(3,tetcode));



-- Creating view tetrahedronneighbours

CREATE OR REPLACE VIEW tetrahedronneighbours AS
   SELECT tetcode tetcode, getneighbourtet1(3,tetcode) ntet1, get-
neighbourtet2(3,tetcode)  ntet2,
                           getneighbourtet3(3,tetcode) ntet3, get-
neighbourtet4(3,tetcode)  ntet4
    FROM tetrahedron
;

-- Creating view constrainedtriangle (with inherited object id's)

CREATE OR REPLACE VIEW constrainedtriangle AS
   SELECT t1.tricode tricode FROM triangle t1
   WHERE NOT EXISTS (SELECT t2.tricode FROM triangle t2 WHERE
t1.tricode = t2.tricode*-1)
;




--  Creating view edge (without inherited object id's and orienta-
tion)

CREATE OR REPLACE VIEW edge AS
   SELECT DISTINCT deriveabsboundaryedge1(3,tricode) edcode FROM
triangle
   UNION
   SELECT DISTINCT deriveabsboundaryedge2(3,tricode) edcode FROM
triangle
   UNION
   SELECT DISTINCT deriveabsboundaryedge3(3,tricode) edcode FROM
triangle
;



-- Creating view constrainededge (without inherited object id's
and orientation)

CREATE OR REPLACE VIEW constrainededge AS
   SELECT DISTINCT deriveabsboundaryedge1(3,tricode) edcode FROM
constrainedtriangle
   UNION
```

```
   SELECT DISTINCT deriveabsboundaryedge2(3,tricode) edcode FROM
constrainedtriangle
   UNION
   SELECT DISTINCT deriveabsboundaryedge3(3,tricode) edcode FROM
constrainedtriangle
;


-- Creating view node

CREATE OR REPLACE VIEW node AS
   SELECT DISTINCT deriveboundarynode1(3,edcode) nodecode FROM ed-
ge
   UNION
   SELECT DISTINCT deriveboundarynode2(3,edcode) nodecode FROM ed-
ge
;
```

Corresponding address

Onderzoeksinstituut OTB
Technische Universiteit Delft
Jaffalaan 9, 2628 BX Delft
Postbus 5030, 2600 GA Delft
Telefoon (015) 278 30 05
Fax (015) 278 44 22
E-mail mailbox@otb.tudelft.nl
www.otb.tudelft.nl