

Chapter 11

First implementation results and open issues on the Poincaré-TEN data structure

Friso Penninga and Peter van Oosterom

Abstract

Modeling 3D geo-information has often been based on either simple extensions of 2D geo-information modeling principles without considering the additional 3D aspects related to correctness of representations or on 3D CAD based solutions applied to geo-information. Our approach is based from the scratch on modeling 3D geo-information based on the mathematically well-defined Poincaré-TEN data structure. The feasibility of this approach still has to be verified in practice. In this paper, the first experiences of loading a reasonable sized data set, comprised of about 1,800 buildings represented by nearly 170,000 tetrahedrons (including the 'air' and 'earth'), are discussed. Though the Poincaré-TEN data structure is feasible, the experience gained during the implementation raises new research topics: physical storage in one (tetrahedron only) or two tables (tetrahedron and node), effective clustering and indexing improvements, more compact representations without losing too much performance, etc.

11.1 Introduction

11.1.1 Motivation

This paper presents the first implementation results of the Poincaré-TEN data structure, as presented earlier in [1]. This structure is developed within a research project 3D Topography and a prototype is being developed within

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

Oracle Spatial. The theoretical strengths of this concept (a compact topological DBMS approach based on a solid mathematical foundation) were demonstrated in previous papers [1, 2, 3]. Despite these strengths, the applicability of the new approach depends heavily on whether the approach is feasible in terms of storage requirements and performance. Therefore, implementing and testing these new ideas is essential. The first implementation results will provide insight to the number of TEN elements and provide some preliminary ideas on storage requirements (as future optimization steps will affect these requirements). At the same time implementing the approach raises new design questions and these open problems will be presented.

11.1.2 Related research

Research in the field of 3D GIS has been performed over the last two decades. Zlatanova et al. [4] gave an overview of the most relevant developments during this period. Related to the topics discussed in this paper, Carlson [5] 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 [6]) is mentioned as a possibility. The explicit use of 3D manifolds to model 3D features is explored by Pigot [7, 8] and Pilouk [9] introduces the TETrahedral irregular Network (TEN), in which the 3-simplex is used as a building block. However, in their work, a rigid mathematical foundation is missing. As far as can be deduced from their descriptions, the 3D simplices are explicitly represented by 2D simplices, specifically, triangles (which are in turn represented by edges and nodes). A topological data model based on 2D simplicial complexes (in 2D space) is introduced [10] and implemented in the PANDA system [11], an early object-oriented database. In applications polyhedrons are often used as 3D primitive [12, 13].

11.1.3 Overview of paper

Before describing the first implementation results and open issues, we will first describe the core characteristics of the previously introduced Poincaré-TEN approach in Section 11.2. After, the approach will be applied to modeling 3D Topography in Section 11.3, while Section 11.4 summarizes the implementation details. The preliminary implementation results with the 1,800 building data set are described in Section 11.5. This paper ends with discussing the current implementation and related open issues in Section 11.6.

11.2 The Poincaré-TEN approach

In this section, first three aspects of our Poincaré-TEN approach are further explained, before the full concept of the approach is used as the foundation for 3D topography modeling:

- It models the world as a full decomposition of 3D space
- The world is modelled in a Tetrahedronized Irregular Network (TEN)
- The TEN is modelled based on Poincaré simplicial homology

11.2.1 *Characteristic 1: Full Decomposition of Space*

As the Poincaré-TEN approach is developed with 3D topographic data in mind, two fundamental observations are of great importance [14]:

- 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 abstraction/generalization). The ISO 19101 Geographic information - Reference model [15] defines features as 'abstractions of real world phenomena'. In most current modeling approaches, the abstraction (read 'simplification') is in the choice of 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 (i.e. no gaps within the domain) modelled space. As a consequence, objects like 'earth' or 'air' are explicitly part of the real world and thus have to be modelled.

Although volume features are the basic elements in the model, planar features might still be very useful, as they mark the boundary (or transition) between two volume features. This approach allows for the existence of planar features, but only as 'derived features'. In terms of UML class diagrams, these planar features are modelled as association classes. For instance, the 'earth surface' is the result of the association between 'earth' and 'non-earth'. Such features might be labeled (for instance as 'grassland' or 'road surface' with additional attributes), but they do not represent or describe the volume object. For example, a road is represented by a volume (despite the appearance of planar features like the road surface), with neighboring volumes that might represent air, earth or other adjacent features.

The explicit inclusion of earth and air features is not very common, since these features are usually considered empty space in between topographic features. Based on following two arguments, we decided to deviate from common practice. First, air and earth features are often also the subject of analyses.

One can think of applications like modeling noise propagation or air pollution. Second, by introducing earth and air features future extensions of the model will be enabled (beyond Topography). Space that is currently labeled as air can be subdivided into air traffic or telecommunication corridors, while earth might be subclassified into geographic layers or polluted regions.

11.2.2 Characteristic 2: using a TEN

Despite initial ideas on a hybrid data model (an integrated TIN/TEN model, based on a pragmatic approach to model in 2,5D as much as possible and to switch to a full 3D model in exceptional cases only), the decision was made [14] to model all topographic features in a TEN. The preference for these simplex-based data structures is based on certain qualities of simplexes (a simplex can be defined as the simplest geometry in a dimension, regarded as the number of points required to describe the geometry):

- 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 (which simplifies amongst others point-in-polygon tests)

Due to the use of simplexes, a 1:n relationship between features and their representations is introduced. The actual usability of the Poincaré-TEN approach depends on the actual size of this n and will be discussed later in this paper.

11.2.3 Characteristic 3: applying Poincaré simplicial homology

The new volumetric approach uses tetrahedrons to model real world features. Tetrahedrons consist of nodes, edges and triangles. All four data types are simplexes: the simplest geometry in each dimension, in which simple refers to minimizing the number of points required to define the shape. A more formal definition [16] of a n -simplex S_n is: a n -simplex S_n is the smallest convex set in Euclidian space \mathbb{R}^m containing $n + 1$ points v_0, \dots, 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: $S_n = \langle v_0, \dots, v_n \rangle$. The boundary of a n -simplex is defined by the following sum of $n - 1$ dimensional simplexes [17] (the *hat* symbol indicates omitting the specific node):

$$\partial S_n = \sum_{i=0}^n (-1)^i \langle v_0, \dots, \hat{v}_i, \dots, v_n \rangle$$

This results in the following boundaries (also see Figure 11.1):

$$\begin{aligned} S_1 = \langle v_0, v_1 \rangle & \quad \partial S_1 = \langle v_1 \rangle - \langle v_0 \rangle \\ S_2 = \langle v_0, v_1, v_2 \rangle & \quad \partial S_2 = \langle v_1, v_2 \rangle - \langle v_0, v_2 \rangle + \langle v_0, v_1 \rangle \\ S_3 = \langle v_0, v_1, v_2, v_3 \rangle & \quad \partial S_3 = \langle v_1, v_2, v_3 \rangle - \langle v_0, v_2, v_3 \rangle \\ & \quad + \langle v_0, v_1, v_3 \rangle - \langle v_0, v_1, v_2 \rangle \end{aligned}$$

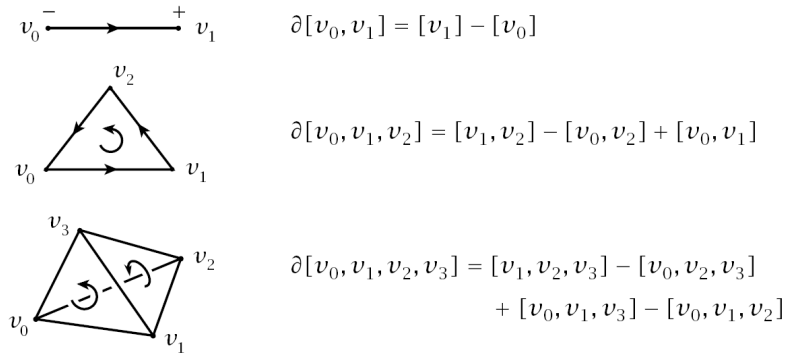


Fig. 11.1 Simplexes and their boundaries (From [16])

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 $S_n = \langle v_0, \dots, v_n \rangle$ have the same orientation, all odd permutations have opposite orientation. So edge $S_1 = \langle v_0, v_1 \rangle$ has boundary $\partial S_1 = \langle v_1 \rangle - \langle v_0 \rangle$. The other permutation $S_1 = - \langle v_0, v_1 \rangle = \langle v_1, v_0 \rangle$ has boundary $\partial S_1 = \langle v_0 \rangle - \langle v_1 \rangle$, which is the opposite direction. As a consequence operators like the dual of a simplex, that is the simplex with the opposite orientation, become very simple: it only requires a single permutation.

The direction of all oriented boundaries of a given simplex obtained with the above boundary operator formula is the same. In 3D this results in the favorable characteristic that with S_3 either all normal vectors of the boundary triangles point inwards or all normal vectors point outwards. This is a direct result of the boundary operator definition, as it is defined in such a manner that $\partial^2 S_n$ is the zero homomorphism, i.e. the boundary of the boundary equals zero (summing-up the positive and negative parts). For example, consider $\partial^2 S_3$, a tetrahedron. The boundary of this tetrahedron consists of four triangles, and the boundaries of these triangles consist of edges. Each of the six edges of S_3 appears twice, as each edge bounds two triangles. Since 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 neighboring triangles have opposite signs if and only if the triangles have similar 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 11.2 shows an example in which all boundaries of the tetrahedrons are added to obtain the boundary triangulation of the building.

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

$$\begin{array}{l}
 C_3 = \\
 \quad - \langle v_0, v_3, v_4 \rangle + \langle v_0, v_1, v_4 \rangle - \langle v_0, v_1, v_3 \rangle + \langle v_2, v_3, v_6 \rangle + \\
 \quad \langle v_1, v_2, v_6 \rangle - \langle v_1, v_2, v_3 \rangle - \langle v_1, v_5, v_6 \rangle - \langle v_1, v_4, v_5 \rangle - \langle v_3, v_6, v_7 \rangle \\
 \quad + \langle v_3, v_4, v_7 \rangle + \langle v_6, v_7, v_8 \rangle - \langle v_4, v_7, v_8 \rangle + \langle v_4, v_5, v_8 \rangle + \\
 \quad - \langle v_6, v_8, v_9 \rangle - \langle v_5, v_8, v_9 \rangle + \langle v_5, v_6, v_9 \rangle
 \end{array}$$

Fig. 11.2 Deriving the boundary triangulation from the TEN

11.3 Poincaré-TEN approach to modeling 3D Topography

11.3.1 Conceptual model

Usually [8, 9], tetrahedrons are defined by four triangles, triangles by three edges and edges by two nodes. Geometry is stored at node level. As a result, reconstructing geometry, for instance a tetrahedron, becomes a relatively laborious operation. In simplicial homology, simplexes of all dimensions are defined by their vertices only, while relationships between other simplexes can be derived by applying the boundary operator. Due to the availability of this operator, there is no need for explicit storage of these relationships. This concept is illustrated in the UML class diagram in Figure 11.3. Tetrahedrons, triangles and edges are defined by an ordered list of nodes. The mutual relationships between tetrahedrons, triangles and nodes (the boundary/coboundary relationships) are derived and signed (i.e. oriented).

Figure 11.3 shows the concept of full space decomposition. 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 on the relationship between two volume features.

11.3.2 Extending simplex notation: vertex encoding

In the Poincaré-TEN approach to 3D topographic data modeling, simplexes are defined by their vertices. Identical to the simplex notation from simplicial homology, where for instance a tetrahedron is noted as $S_3 = \langle v_0, v_1, v_2, v_3 \rangle$, simplex identifiers are constructed by concatenating the vertex ID's. In doing so, unique identifiers exist that contain orientation information as well, since the order of vertices determines the orientation. In an earlier paper [1], we suggested the use of x, y and z coordinate concatenation as node ID. Since 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. The geometry itself will be a unique identifier. Concatenating the coordinate pairs into one long identifier code and sorting the resulting 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.

Figure 11.4 [1] illustrates this idea of vertex encoding in a simplicial complex-based approach. A house is tetrahedronized and the resulting tetrahedrons are coded as the 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 ele-

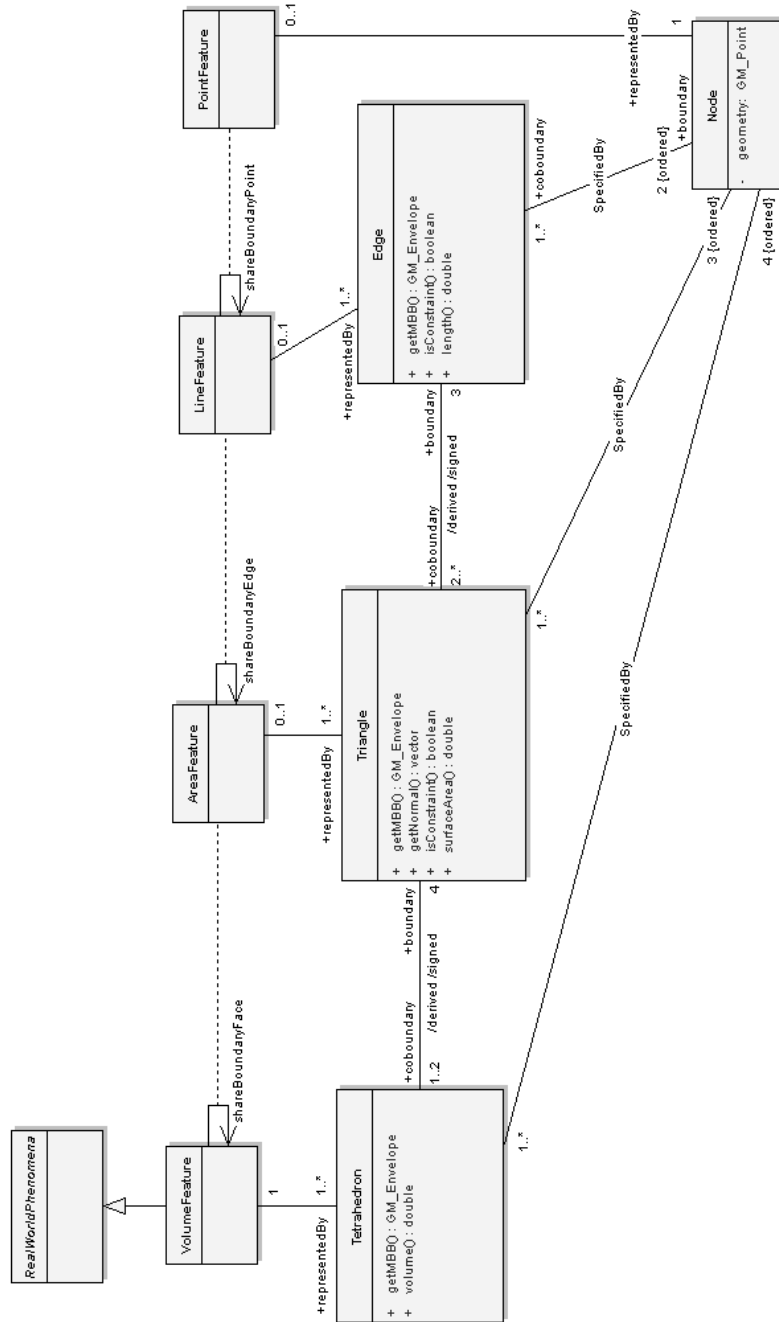


Fig. 11.3 UML class diagram of the simplicial complex-based approach

ment. Therefore, the last row (100000000600100600100608) should be interpreted 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.

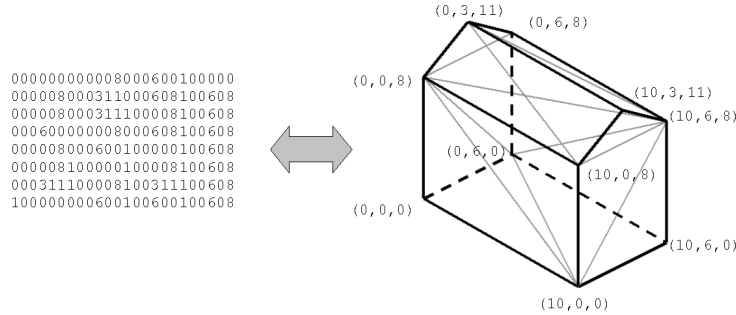


Fig. 11.4 Describing tetrahedrons by their encoded vertices

11.4 Current implementation

To provide greater insight into the proposed new approach, the basic structure is implemented within the Oracle DBMS. This section will summarize the current status of the implementation, but one has to realize that this is still a work in progress. At this moment, the required tetrahedronization algorithms are not implemented within the DBMS, so TetGen [18] is used to perform an external batch tetrahedronisation. The input is a Piecewise Linear Complex (PLC), see Figure 11.5. A PLC [19] is a set of vertices, segments and facets, where a facet is a polygonal region. Each facet may be non-convex and contain holes, segments and vertices, but it should not be a curved surface. A facet can represent any planar straight line graph (PSLG), which is a popular input model used by many two-dimensional mesh algorithms. A PSLG is [20] a graph embedding of a planar graph (i.e. a graph without graph edge crossings), in which only straight line segments are used to connect the graph vertices.

Compared to a polyhedron, a PLC is a more flexible format. If one looks at the shaded facet in Figure 11.5, one can see that this facet cannot be described by a polygon because there are loose and dangling line segments. However, in our application, situations like these will be rare or not appear at all. Based on an input PLC, TetGen creates a constrained Delaunay tetrahedronisation. This tetrahedronization is loaded into the database and then converted into the Poincaré-TEN format. Figure 11.6 shows this concept with a small test dataset, from the input PLC (top), via the tetrahedronization (mid) to the

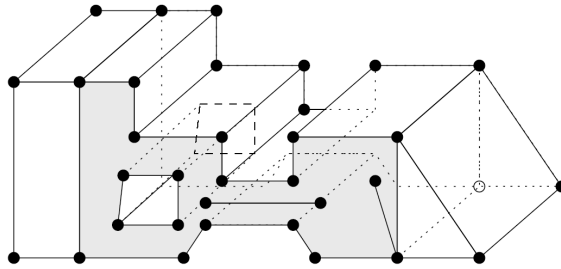


Fig. 11.5 A Piecewise Linear Complex (PLC), input for the tetrahedronization algorithm (From [21])

output in which only the constrained triangles (the feature boundary faces) are drawn (bottom).

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 $x_1y_1z_1x_2y_2z_2x_3y_3z_3x_4y_4z_4id$ (based on fixed length character strings). Note that besides the geometry, a unique identifier is added, which refers to a volume feature that is (partly) represented by the tetrahedron. The tetrahedrons are not signed, but are assumed to be a positive permutation, meaning that all normal vectors on boundary triangles are oriented outwards. This is checked and ensured during the initial loading proces. A consistent orientation is required to ensure that each boundary triangle appears twice: once with positive and once with negative orientation. The orientation simplifies determination of left/right and inside/outside relations. 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

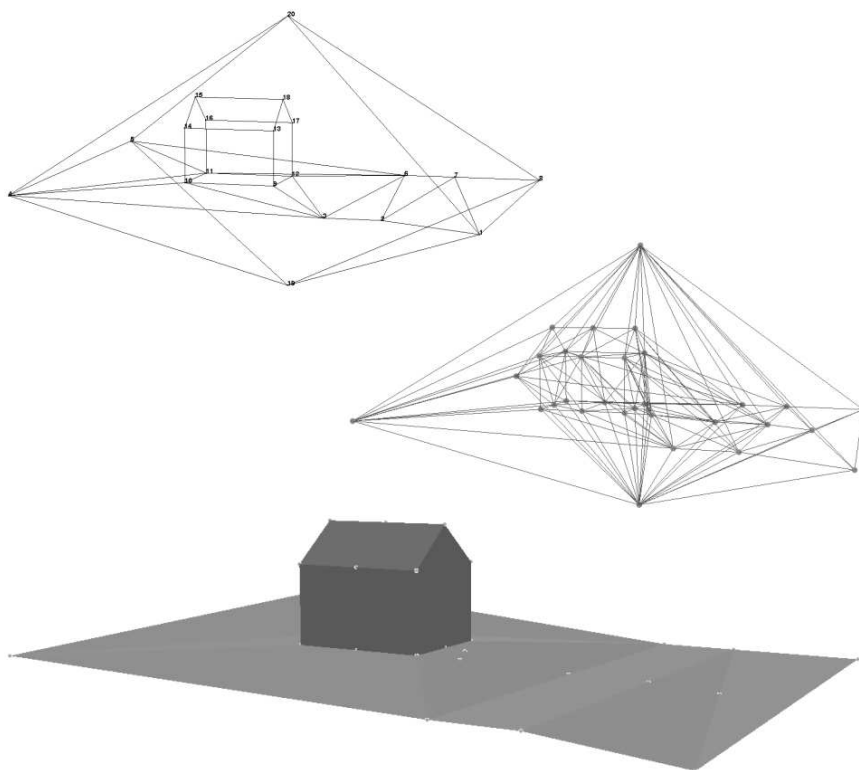


Fig. 11.6 Input PLC (top), the resulting tetrahedronization (mid) and as output the constrained triangles (i.e. the feature boundaries)(bottom)

of which the triangle is part of the (internal) boundary. The reason for this will be introduced in the next section. Also, it can 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 the same permutation is used both in the positive and negative orientations, as they will not cancel out in pairs otherwise. The `ordertriangle` procedure always rewrites a triangle $\langle a, b, c \rangle$ such that $a < b < c$ holds, which is an arbitrary criterion.

Based on a slightly altered version of the `deriveboundarytriangles` operator the triangle view is created. The resulting view contains all triangles (coded by their geometry and inherited object id's) 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 a tetrahedron, of which the triangle is part of the boundary. This coboundary will prove useful in deriving topo-

logical relationships later in this section. The resulting view will contain four times the number of tetrahedrons and every triangle appears twice: once with a positive and once with a negative sign (and not in a permuted form, due to the `ordertriangle` procedure). However, it must be realized that this is just a view and no actual storage takes place:

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

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 update proces). To achieve this, the incremental tetrahedronization 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. This derivation is based on 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 twice. As stated before, the triangle code inherits the 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. A view with constrained triangles can be derived:

```
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);
```

Other views might be defined to simplify operations, for instance, a view with triangles stored by their geometry alone or a view without duals.

Similar to deriving the triangle views, views with edges, constrained edges and nodes can be constructed. Note that the views with edges contain no duals, i.e. edges are described only by their geometry:

```
create or replace view edge as
```

```

select distinct deriveabsboundaryedge1(tricode) edcode
from triangle
UNION
select distinct deriveabsboundaryedge2(tricode) edcode
from triangle
UNION
select distinct deriveabsboundaryedge3(tricode) edcode
from triangle;

create or replace view constrainededge as
select distinct deriveabsboundaryedge1(tricode) edcode
from constrainedtriangle
UNION
select distinct deriveabsboundaryedge2(tricode) edcode
from constrainedtriangle
UNION
select distinct deriveabsboundaryedge3(tricode) edcode
from constrainedtriangle;

create or replace view node as
select distinct deriveboundarynode1(edcode) nodecode
from edge
UNION
select distinct deriveboundarynode2(edcode) nodecode
from edge;

```

In the current implementation edges are undirected and do not inherit object id's, as no application for this has been identified. However, strict application of the boundary operator results in directed triangles. With the tetrahedron table and triangle, edge and node views, the data structure is accessible at different levels. Due to encoding of the vertices, both geometry and topology are present at every level, thus enabling switching to the most appropriate approach for every operation.

11.5 Preliminary implementation results

The very small dataset from Figure 11.6 is now replaced by a larger dataset. It consists of 1796 buildings in the northern part of Rotterdam (see Figure 11.7) and covers an area of about seven square kilometres. At this moment other topographic features like the earth surface, roads, tunnels etc. are still missing due to lacking appropriate 3D data. Complex situations like multiple land use and highway interchanges are lacking as well. However, this dataset will provide more insight into the number of elements of a TEN.

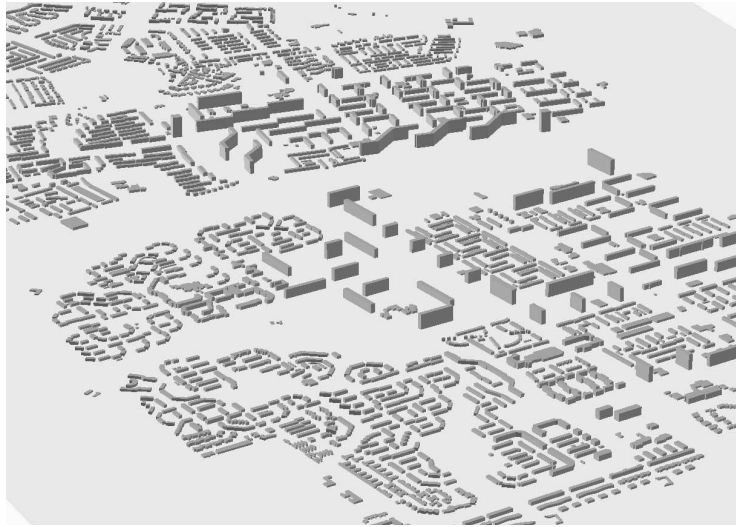


Fig. 11.7 Rotterdam test data set: 1796 buildings

11.5.1 An alternative model

Until now, simplex codes are obtained by concatenating the node coordinates, like $x_1y_1z_1x_2y_2z_2x_3y_3z_3x_4y_4z_4$. This approach is based on the idea that a node table only contains geometry and that adding an identifier would be a bit redundant, since the geometry is already a unique identifier. Nevertheless, one can question whether this approach actually reduces storage requirements, since each node is part of multiple tetrahedrons (the Rotterdam tetrahedronization shows an average of about fifteen tetrahedrons per node; see next subsection). Due to this result, the concatenated coordinate pair is used multiple times. As long as a node identifier requires considerably less storage space compared to this concatenated geometry, switching to a tetrahedron-node approach might be feasible. An additional node table containing a node identifier is required to create shorter simplex codes like $nid_1nid_2nid_3nid_4$.

With this idea in mind tetrahedronization of the Rotterdam data set was performed. Since TetGen output consists of a comparable structure with both a tetrahedron and a node table, incorporating TetGen results in the Poincaré-TEN structure was easier in the tetrahedron-node version. Since obtaining a working implementation was strongly favored over minimizing storage requirements or optimizing performance at this point, the tetrahedron-node implementation was used. As a result, two tables with tetrahedrons and nodes are stored. All simplexes are identified by a concatenation/permutation of node id's instead of concatenated coordinate pairs. All pro's and con's re-

garding the choice between a tetrahedron-only and a tetrahedron-node implementation will be discussed into more detail in Section 11.6.2.

11.5.2 Preliminary results on storage requirements

The Rotterdam data set is first converted into the input format for TetGen, the tetrahedronization software. This input format requires a list of nodes with geometry, a list of faces (described by their nodes) and a list of points inside each object to identify the object. Real volumetric 3D data is rare, so one has to convert data or integrate multiple sources. Modifying this into the topological format in which faces are described by their nodes is usually a very time-consuming task. The input dataset consists of 26,656 nodes, 16,928 faces and 1,796 points to identify the 1,796 buildings. Tetrahedronizing this input set with TetGen results in a TEN, consisting of 30,877 nodes, 54,566 constrained triangles and 167,598 tetrahedrons. One should note that the tetrahedronization results in one network, i.e. the space in between the buildings is tetrahedronized as well! The increase in the number of nodes is caused by the addition of Steiner points; additional points required to either enable tetrahedronization or improve tetrahedronization quality (in terms of avoiding ill-shaped triangles and tetrahedrons to avoid numerical instability).

The tetrahedronization results are loaded into the Poincaré-TEN structure. The tetrahedron table consists of 167.598 tetrahedrons. Based on these tetrahedron table, views are created with triangles, constrained triangles, edges and nodes by repeatedly applying the boundary operator. Since the Poincaré-TEN structure contains duals of all triangles as well, the numbers differ from the initial tetrahedronization. From the 167,598 tetrahedrons 670,392 (4 x number of tetrahedrons) triangles are derived, of which 109,120 are constrained triangles. This number slightly differs from multiplying the original 54.566 constrained triangles by two (because of inclusion of the dual), since the outer boundary of the TEN consists of twelve triangles without a dual. The edge view provides information for the 198,480 edges. Note that these edges are described by their nodes alone, so without inherited object id, dual or sign, i.e. each geometry is unique.

As stated before, the current implementation is very straightforward. Obtaining a working implementation was strongly favored over minimizing storage requirements or optimizing performance. However, improving these aspects is one of the most important tasks for the upcoming period. Nevertheless, storage requirements of the current approach are compared to requirements of a polyhedron approach. The polyhedrons are described in Oracle as a solid, defined by a set of polygonal faces, each described by their vertices. The TEN approach slightly differs from previously described implementations, as it consists of both a tetrahedron and a node table.

The Poincaré-TEN approach requires 1.44 and 19.65 MB, respectively, for the node and tetrahedron table, while the polyhedron tables requires 4.39 MB. This means that the current (absolutely not optimized!) implementation requires about 4.8 times more storage space. However, as will be discussed in Section 11.6.4, the feasibility of our approach should be assessed both based on storage requirements as well as performance. A simple storage reduction can be obtained by using bit string instead of character string representation of the coordinates. Not only will this save storage space (estimated between a factor 2 to 3), but it would also increase performance and no ascii to binary conversions are necessary when using the coordinates.

11.6 Discussion of open issues

The prototype implementations show that the Poincaré-TEN approach is indeed feasible and can be used for well defined representation, but is still usable in basic GIS functions: selection of relevant objects and their visualization. Further, basic analysis is very well possible: both using topology (e.g. find the neighbors of a given feature) and geometry (e.g. compute volume of a given feature by summing tetrahedron volumes). Finding neighbors of a given feature can be implemented by querying the constrained triangle view to find all boundary triangles with a specific feature identifier. Through a view with triangle duals, the neighboring features can be identified quickly. An alternative approach would be to traverse the TEN tetrahedron by tetrahedron and test for feature identifier changes. A function to find neighboring tetrahedrons can be defined:

```
create or replace function getneighbourtet1(
(...))
select fromtetcode into neighbourtet from triangle
where removeobjectid(tricode)= -1 *removeobjectid(tricode);
(...)
```

or the volume of a tetrahedron can be calculated using the Cayley-Menger determinant [22] (with d_{ij} as length of edge $\langle v_i, v_j \rangle$):

$$288V^2 = \begin{vmatrix} 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & d_{01}^2 & d_{02}^2 & d_{03}^2 \\ 1 & d_{10}^2 & 0 & d_{12}^2 & d_{13}^2 \\ 1 & d_{20}^2 & d_{21}^2 & 0 & d_{23}^2 \\ 1 & d_{30}^2 & d_{31}^2 & d_{32}^2 & 0 \end{vmatrix}$$

Although these capabilities have been established, ongoing research is attempting to provide answers to a number of some open issues. These issues will be described in the final subsections of this paper.

11.6.1 Open issue 0. Spatial clustering and indexing

The large real world data set will require spatial organization of the data to enable the efficient implementation of spatial queries such as the rectangle (or box) selections. Spatial organization includes spatial clustering (things close in reality are also close in computer memory, which is tricky given the one dimensional nature of computer memory) and spatial indexing (given the spatial selection predicate, the addresses of the relevant objects can be found efficiently). If the current coding of the tetrahedrons (first x, then y, then z) is replaced by bitwise interleaving, the tetrahedron code itself may be used for spatial clustering (similar to the Morton code) and used for spatial indexing without using additional structures (such as quad-tree or r-tree, also requiring significant storage space and maintenance during updates) [23]. Only the coboundary references of the triangles might need functional indexes to improve performance.

11.6.2 Open issue 1. Minimizing redundancy: tetrahedron only vs. tetrahedron-node

In this paper, two variants of the implementation have been described. If a separate node table is used, with compact node id's, then the issue of realizing spatial clustering and indexing is relevant for both tables. As there is no direct geometry in the tetrahedron table, the bitwise interleaving approach of the coordinates cannot be used (and probably a more explicit technique has to be applied). At this time, no comparative results are available, but one can expect the tetrahedron-node variant to be cheaper in terms of storage than the tetrahedron-only approach. However, reducing data storage might deteriorate performance, as additional operations are necessary to perform geometrical operations on top of simplexes. If one thinks, for instance, of the operation that checks whether a tetrahedron is oriented positively or negatively, one needs the node coordinates to calculate a normal vector on one of the triangles and calculate the angle between this normal vector and a vector from a triangle opposite to the fourth node to determine whether the normal points inwards or outwards. To perform this operation in the tetrahedron-node implementation, one has to search the node table first to obtain the node geometries.

11.6.3 Open issue 2. Dealing with storage requirements: storing all coordinates vs. storing differences

Assuming that one opts for the tetrahedron only approach, storage requirements can be reduced by avoiding storage of the full coordinates. Since the four nodes are relatively close to each other, one might choose to store the coordinates of one node and only give difference vectors to the other three nodes: $x_1y_1z_1x_2y_2z_2x_3y_3z_3x_4y_4z_4$ would change into $xyz\delta x_1\delta y_1\delta z_1\delta x_2\delta y_2\delta z_2\delta x_3\delta y_3\delta z_3$. Similar to the choice between the tetrahedron only and the tetrahedron-node implementation, reducing data storage will come at a price. Again additional operators are required to reconstruct the four node geometries when necessary. However, if these can be implemented efficiently (and there is no reason why this can not be done), they could be used in a view translating the compact physical storage representation in a more verbose full representation (but as this is only a view, it is not stored so it does not matter that this size is larger). Also, the bitwise interleaving approach to provide spatial clustering and indexing may still work well with this approach (as it is sufficient to do only bitwise interleaving of the first coordinate).

11.6.4 Open issue 3. How to assess feasibility of the Poincaré-TEN approach

In the implementation of the theory, as indicated in this paper (first prototype and also the open issues described above for further improvement), care has to be taken so that the storage requirements are not excessive (compared to other approaches) as this would make the approach less feasible (storage requirements should be linear in the number of features represented). In general, bulky storage requires more time to retrieve data from the disk, as compared to compact storage. However, if very expensive computations are needed (e.g. joins which are not well supported), then bad response times could occur. It is important to implement the typical basic functionality effectively (both w.r.t. storage and time performance). At this moment, there seems to be no basic functions that cannot be implemented time efficiently (when proper clustering/indexing is applied). However, this assumption still has to be proven.

11.6.5 Open issue 4. Correct insertion of 3D objects: snapping to the earth surface

3D data sets are required to load into the current implementation. Although research efforts are made to increase availability of such datasets [24], dependence of the availability of such data sets seriously limits applicability of the data structure at this time. Therefore, additional functionality is required to switch from importing 3D data sets into importing 3D data from different sources. One can imagine that creation of a 3D topographic model starts with the creation of the earth surface, followed by inclusion of 3D buildings. In general, buildings are built on top of the earth surface. As the earth surface and building data originates from different sources, these objects are not likely to fit together perfectly. To cope with such situation, one needs a snap-to-earth-surface operator. Such an operator will project the buildings footprint onto the terrain and determine the distance between terrain and buildings underside. If this distance is smaller than a certain pre-set tolerance, the building will be placed on the terrain by applying a vertical displacement, thus ensuring a tight fit. Two options exist for this, as one can either adjust the buildings underside to fit the terrain or adjust the terrain to fit the (usually flat) underside of the building. The snapping operator can also be utilized for inclusion of infrastructural objects and land coverage objects.

11.6.6 Open issue 5. Incremental updating of existing structure in DBMS

The current implementation lacks any tetrahedronization algorithms. At this time, TetGen software is used and the resulting output is loaded into the database and subsequently converted into the Poincaré-TEN structure. With the intended application of 3D Topography in mind, bulk loading is useful for the initial model build, but updates should be handled incrementally. A theoretical framework of incremental updates in a TEN structure is presented in [2, 25]. However, these ideas still need further implementation and development. It will be most effective to devise incremental update procedures that act as local as possible, with the risk that quality parameters like the Delaunay criterion or shortest-to-longest edge ratios are temporarily not met. This could be compensated by a cleaning function that performs a local rebuild or even a full retetrahedronization. Obviously, such an operation needs to be performed every now and then, but not after every update, thus speeding up the update process.

References

- [1] Penninga, F., van Oosterom, P.: A Compact Topological DBMS Data Structure For 3D Topography. In Fabrikant, S., Wachowicz, M., eds.: *Geographic Information Science and Systems in Europe, Agile Conference 2007*. Lecture Notes in Geoinformation and Cartography, Springer (2007)
- [2] Penninga, F., van Oosterom, P.: Updating Features in a TEN-based DBMS approach for 3D Topographic Data modeling. 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
- [3] Penninga, F., van Oosterom, P., Kazar, B.M.: A TEN-based DBMS approach for 3D Topographic Data modeling. In Riedl, A., Kainz, W., Elmes, G., eds.: *Progress in Spatial Data Handling, 12th International Symposium on spatial Data Handling*, Springer (2006) 581–598
- [4] 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)
- [5] Carlson, E.: Three-dimensional conceptual modeling of subsurface structures. In: *Auto-Carto 8*. (1987) 336–345
- [6] 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)
- [7] 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
- [8] Pigot, S.: A topological model for a 3-dimensional Spatial Information System. PhD thesis, University of Tasmania, Australia (1995)
- [9] Pilouk, M.: Integrated modeling for 3D GIS. PhD thesis, ITC Enschede, Netherlands (1996)
- [10] Egenhofer, M., Frank, A., Jackson, J.: A Topological Data Model for Spatial Databases. In: *Proceedings of First Symposium SSD'89*. (1989) 271–286
- [11] 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
- [12] Zlatanova, S.: 3D GIS for urban development. PhD thesis, Graz University of Technology (2000)
- [13] Stoter, J.: 3D Cadastre. PhD thesis, Delft University of Technology (2004)
- [14] Penninga, F.: 3D Topographic Data modeling: Why Rigidity Is Preferable to Pragmatism. In Cohn, A.G., Mark, D.M., eds.: *Spatial Infor-*

- mation Theory, Cosit'05. Volume 3693 of Lecture Notes on Computer Science., Springer (2005) 409–425
- [15] ISO/TC211: Geographic information - reference model. Technical Report ISO 19101, International Organization for Standardization (2005)
 - [16] Hatcher, A.: Algebraic Topology. Cambridge University Press (2002) Available at <http://www.math.cornell.edu/hatcher>.
 - [17] Poincaré, H.: Complément à l'Analysis Situs. Rendiconti del Circolo Matematico di Palermo **13** (1899) 285–343
 - [18] <http://tetgen.berlios.de/>: (2007)
 - [19] Miller, G.L., Talmor, D., Teng, S.H., Walkington, N., Wang, H.: Control Volume Meshes using Sphere Packing: Generation, Refinement and Coarsening. In: 5th International Meshing Roundtable, Sandia National Laboratories (1996) 47–62
 - [20] <http://mathworld.wolfram.com/PlanarStraightLineGraph.html>: (2007)
 - [21] Si, H.: TetGen, A Quality Tetrahedral Mesh Generator and Three-Dimensional Delaunay Triangulator. User's Manual. Technical report, Weierstrass Institute for Applied Analysis and Stochastics, Berlin, Germany (2006) Available at <http://tetgen.berlios.de/files/tetgen-manual.pdf>.
 - [22] Collins, K.D.: Cayley-Menger Determinant. From Mathworld – A Wolfram Web Resource. <http://mathworld.wolfram.com/Cayley-MengerDeterminant.html> (2003)
 - [23] van Oosterom, P., Vijlbrief, T.: The Spatial Location Code. In Kraak, M.J., Molenaar, M., eds.: Advances in GIS research II; proceedings of the seventh International Symposium on Spatial Data Handling - SDH'96, Taylor and Francis (1996)
 - [24] Oude Elberink, S., Vosselman, G.: Adding the Third Dimension to a Topographic Database Using Airborne Laser Scanner Data. In: Photogrammetric Computer Vision 2006. IAPRS, Bonn, Germany. (2006)
 - [25] Penninga, F., van Oosterom, P.: Editing Features in a TEN-based DBMS approach for 3D Topographic Data modeling. Technical Report GIS Report No. 43, Delft University of Technology (2006) Available at <http://www.gdmc.nl/publications/reports/GIS43.pdf>.