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

Friso Penninga & Peter van Oosterom

F.Penninga@tudelft.nl, oosterom@tudelft.nl

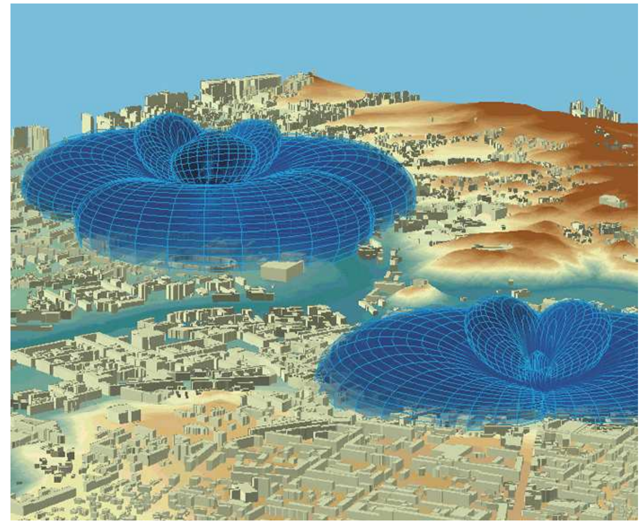
Delft University of Technology, OTB

section GIS Technology

Presentation outline

- Introduction
- Previous research
    - Characteristics Poincaré-TEN approach
    - Poincaré-TEN applied to 3D Topography
    - Implementation details
- Results Rotterdam data set
- Discussion of open issues
- Conclusions

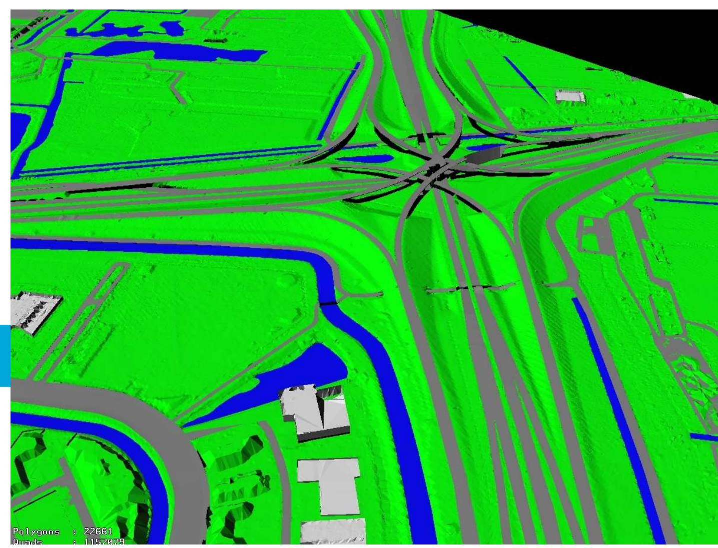
Introduction

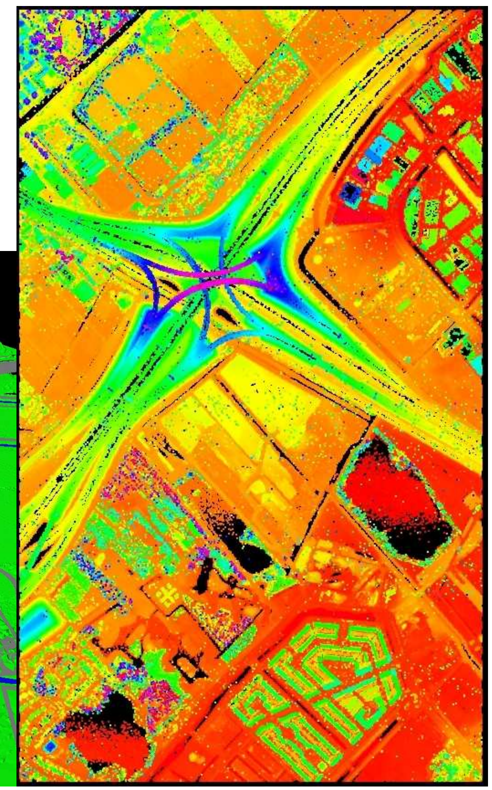


Poincaré-TEN structure:

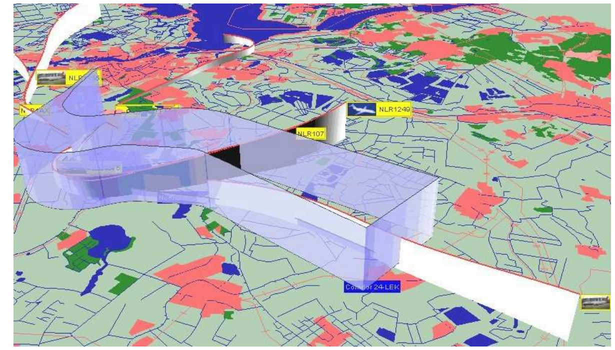
- DBMS data structure
- Supports query, analysis and validation

Developed within research project 3D Topography:
focus on 3D acquisition as well as 3D modelling





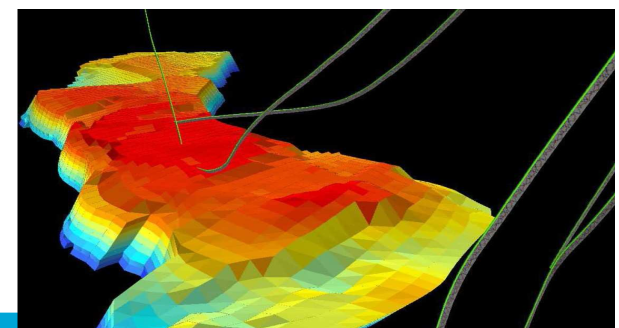
Previous research (1/3)
Poincaré-TEN characteristics



Characteristic 1: Full decomposition of space

Two fundamental observations (Cosit'05 paper):

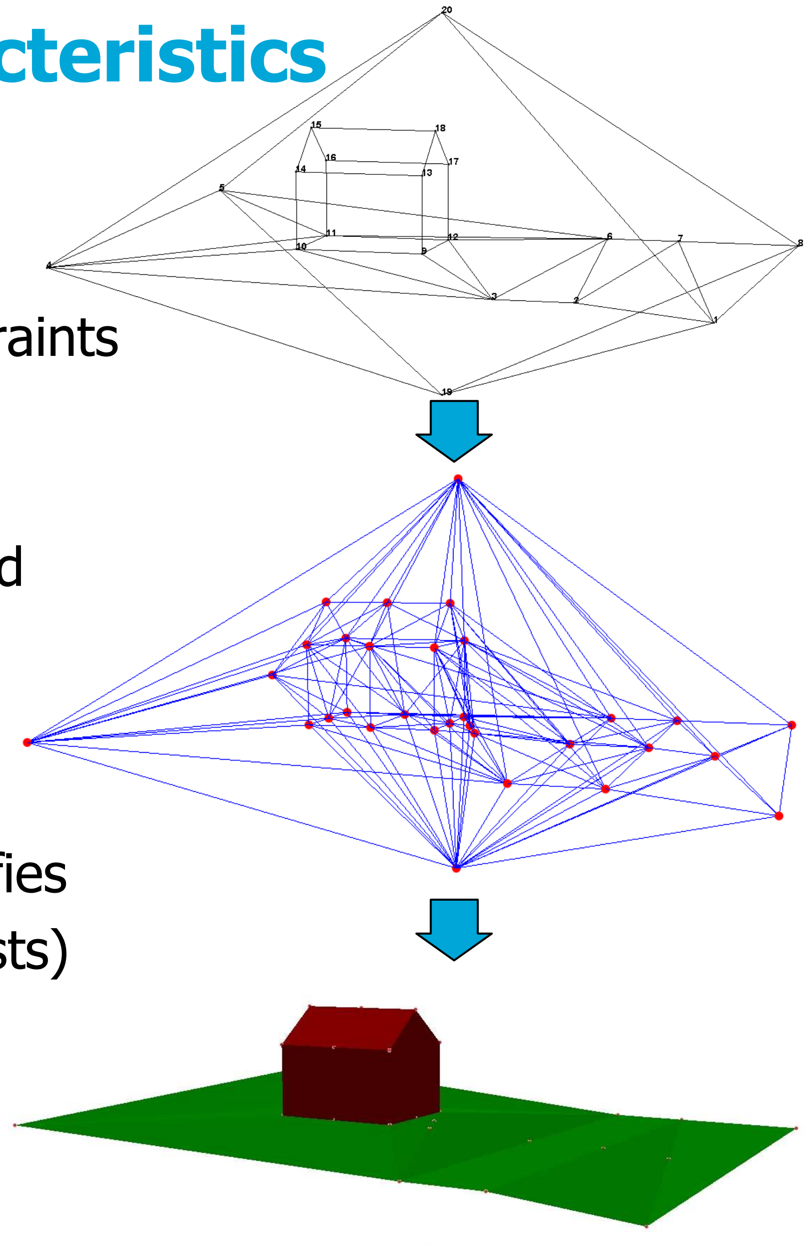
- ISO19101: a feature is an 'abstraction of real world phenomena'. These real world phenomena have by definition a volume
- Real world can be considered to be a volume partition (analogous to a planar partition: a set of non-overlapping volumes that form a closed modelled space)



Result: explicit inclusion of earth and air

Previous research (2/3)
Poincaré-TEN characteristics

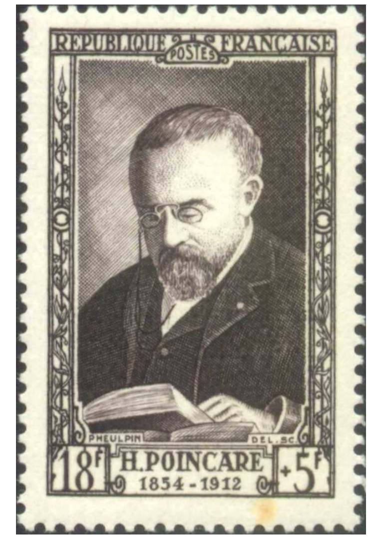
Characteristic 2: constrained TEN
object boundaries represented by constraints



Advantages of TEN:

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

Previous research (3/3)
Poincaré-TEN characteristics



Characteristic 3: based on Poincaré simplicial homology solid mathematical foundation (SDH'06 paper):

Simplex S_n defined by $(n+1)$ vertices: $S_n = < v_0, \ldots v_n >$

The boundary ∂ of simplex S_n is defined as sum of *(n-1)* dimensional simplexes (note that 'hat' means skip the node):

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

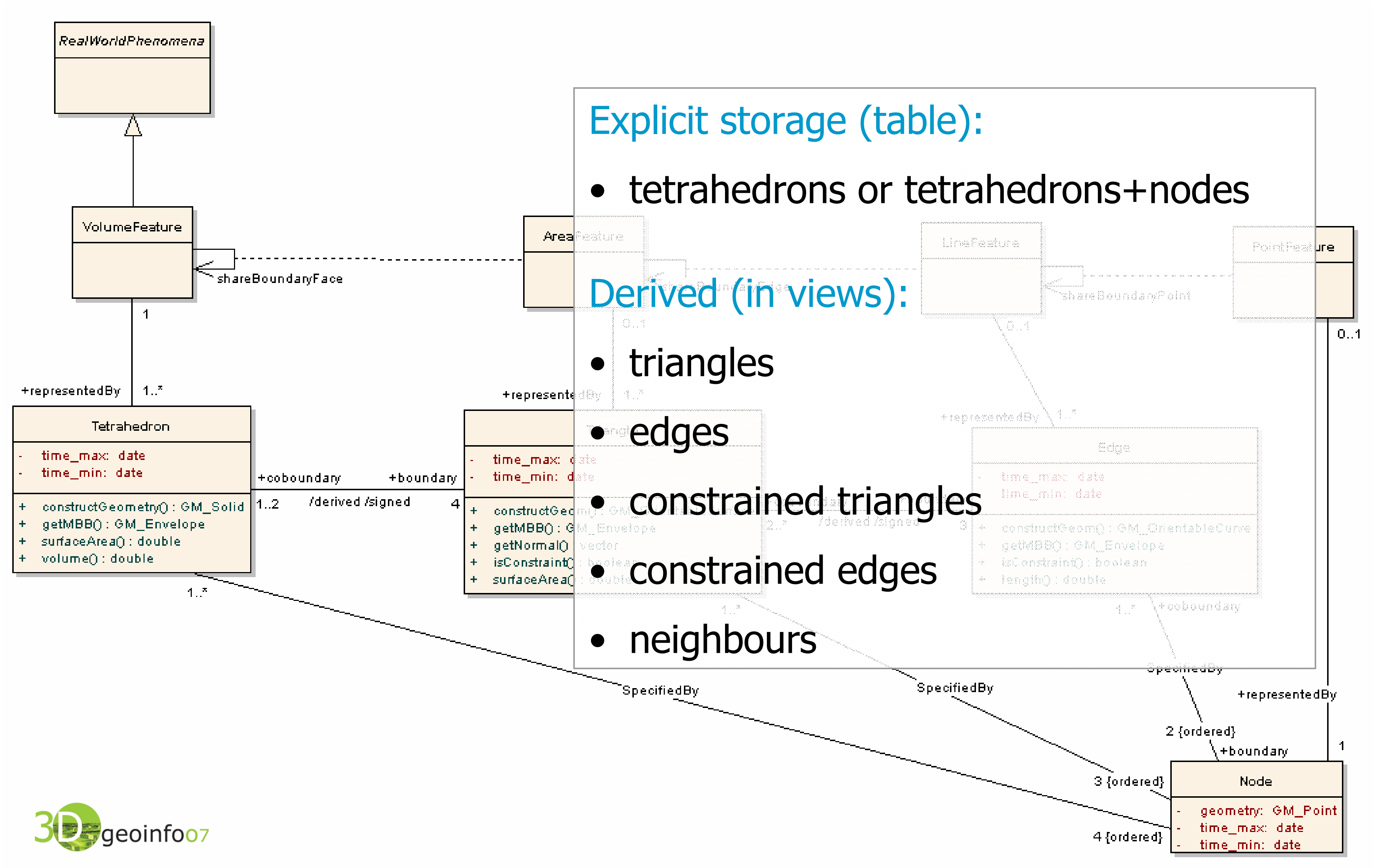
remark: sum has n+1 terms

$$S_1 = <v_0, v_1> \qquad \partial S_1 = <v_1> - <v_0>$$

$$S_2 = <v_0, v_1, v_2> \qquad \partial S_2 = <v_1, v_2> - <v_0, v_2> + <v_0, v_1>$$

$$S_3 = <v_0, v_1, v_2, v_3> \qquad \partial S_3 = <v_1, v_2, v_3> - <v_0, v_2, v_3> + <v_0, v_1, v_3> - <v_0, v_1, v_2>$$

Poincaré-TEN applied to 3D topography



Explicit storage (table):

- tetrahedrons or tetrahedrons+nodes

Derived (in views):

- triangles
- edges
- constrained triangles
- constrained edges
- neighbours

3Dgeoinfo07

Implementation details
DBMS

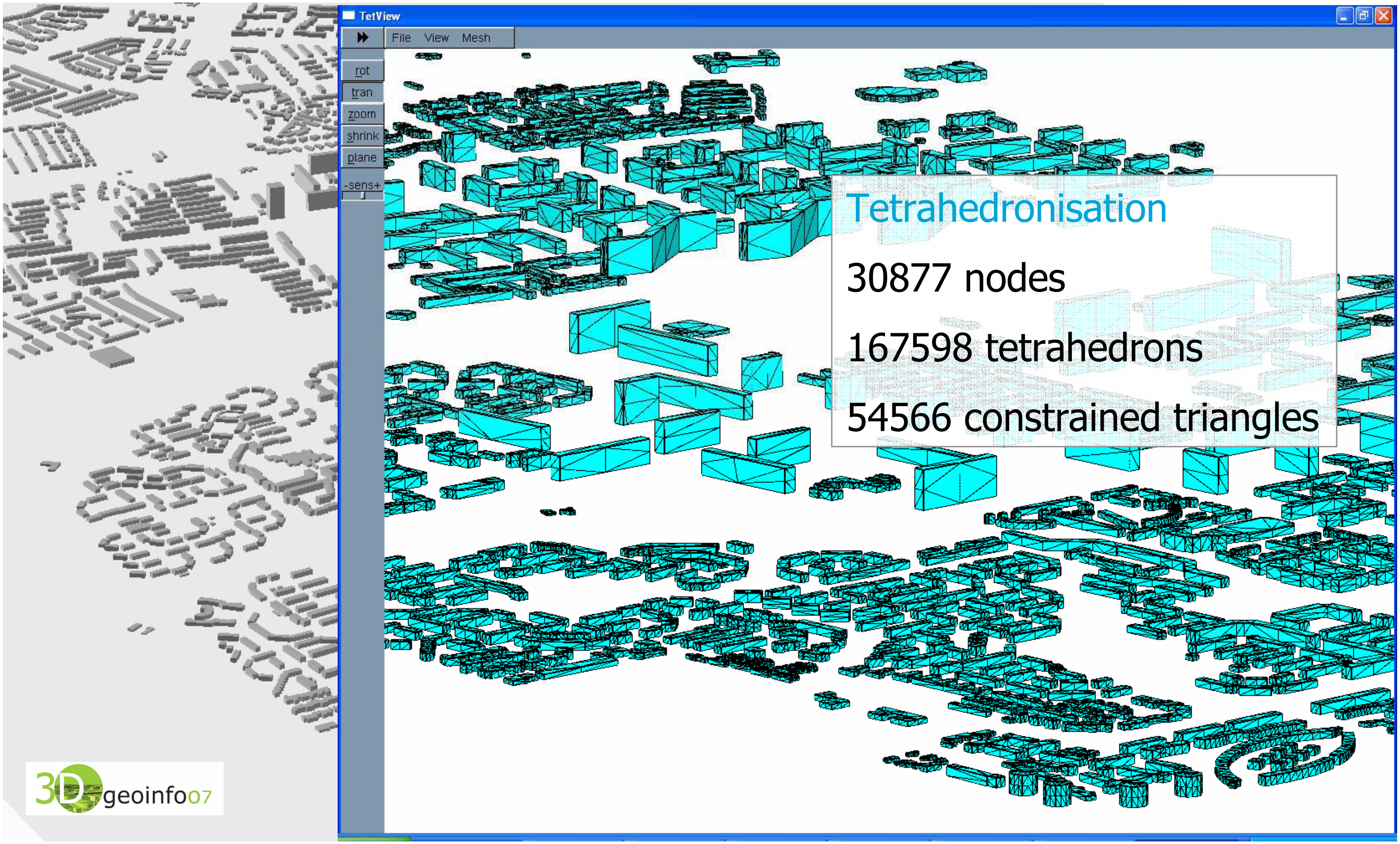
$$\partial S_n = \sum_{i=0}^{n} (-1)^i < v_0, \ldots, \hat{v}_i, \ldots, v_n >$$

Boundary operator implemented in PL/SQL procedure

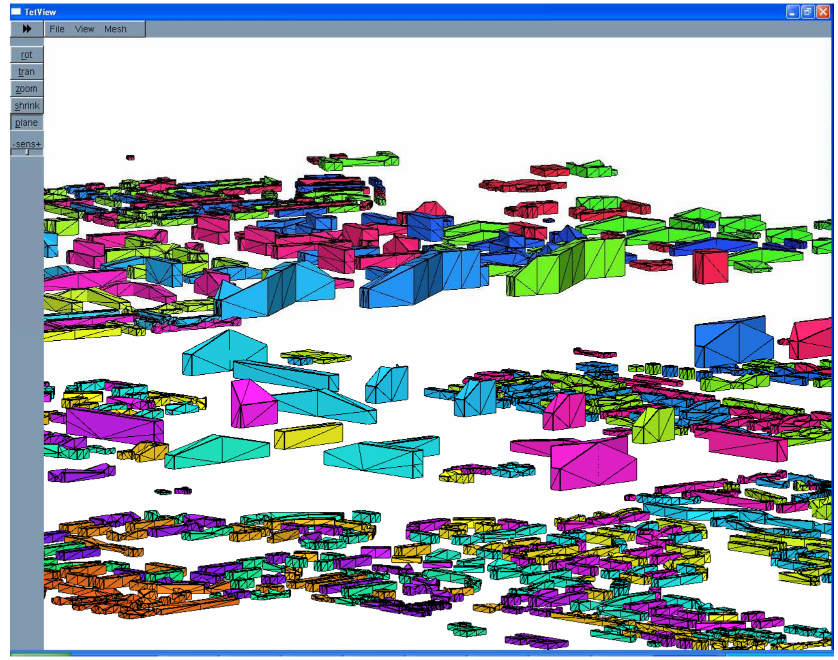
Procedure used to define views with triangles, edges, constrained triangles (object boundaries!), constrained edges, e.g.:

```
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
    ...
```

Results (1/2)
Rotterdam data set



Results (2/2)
Rotterdam data set



Data storage requirements

Poincaré-TEN	Polyhedron
21.09 MB	4.39 MB
(node 1.44 MB)	
(tetrahedron 19.65 MB)	

PT-approach costs about 4.8 times more storage…

new *(but over 77.7% of tetrahedrons represent either air or earth, so buildings require about 5.76 MB. So factor 4.8 → 1.3)*

Open issues
0. Spatial clustering and indexing

Basic idea:

Why add a meaningless unique id to a node, when its geometry is already unique?

0.1 Bitwise interleaving coordinates → Morton-like code → sorting these codes → spatial clustering

0.2 Use as spatial index → no addtional indexes (R-tree/quad tree)

Objective: reducing storage requirements

Open issues

1. Minimizing storage requirements: tetrahedron only vs. tetrahedron-node

Tetrahedron only: describe tetrahedrons by node geometries:

$x_1y_1z_1x_2y_2z_2x_3y_3z_3x_4y_4z_4$

Tetrahedron-node: describe tetrahedrons by node id's:

$id_1id_2id_3id_4$ with $id_1{:}x_1y_1z_1$, $id_2{:}x_2y_2z_2$, etc.

A node is part of multiple tetrahedrons (Rotterdam data set: av.20), so either repeating geometries or repeating identifiers in tetrahedron table.

Tetrahedron-node will require less storage space (as long as id takes less storage than coordinate triplet)

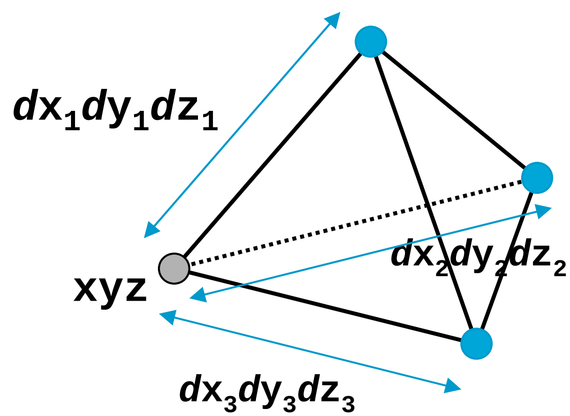
Open issues
2. Coordinates vs. coord. differences

Four nodes of a tetrahedron will be relatively close:
only small differences in coordinates

Alternative tetrahedron description:

$xyz\,dx_1 dy_1 dz_1 dx_2 dy_2 dz_2 dx_3 dy_3 dz_3$

Description is based on geometry
(so still unique) but smaller



Open issues
3. Feasibility assesment

Delicate balance between storage and performance

Open issues
4. Object snapping

Focus on snapping to earth surface:
buildings, roads, etc.

↓

Ensuring correctness of the model

Open issues
5. Incremental updates

Topography changes continuously

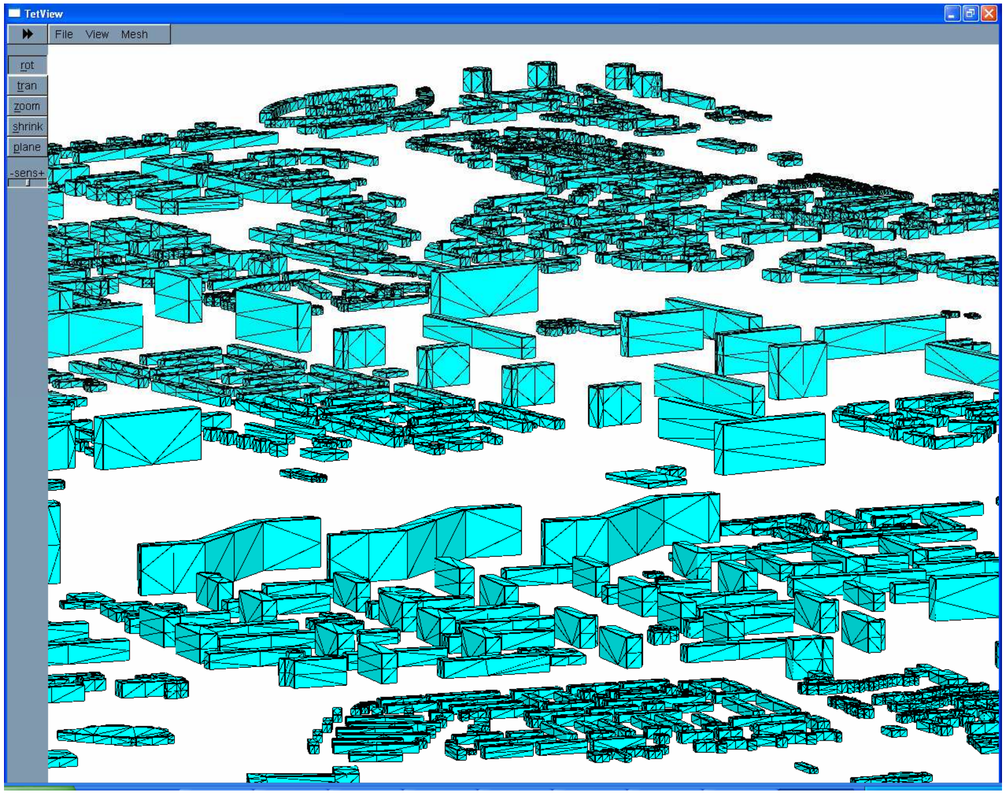
Need for incremental updates

act as locally as possible ⟷ ensuring tetrahedronisation quality

3Dgeoinfo07

TU Delft

Discussion



Friso Penninga & Peter van Oosterom
F.Penninga@tudelft.nl, oosterom@tudelft.nl

3D geoinfo07

TU Delft