Towards a 3D Feature Overlay through a Tetrahedral Mesh Data Structure

Edward Verbree, Arno van der Most, Wilko Quak, and Peter van Oosterom

ABSTRACT: The use of 3D features within GIS has been increasing due to the need to represent, query, manipulate, and analyze man-made objects in relationship to other 3D features related to the surface of the earth. This will yield an increased use of 3D boundary representations of the features. The spatial relationship between two or more features is often evaluated using a geometrical overlay of these features, which reveals whether these features overlap and—if they do—to which extent. We present the design of a 3D overlay algorithm which overlays 3D triangulated boundary representations through a constrained tetrahedral mesh. The intersections between the constrained facets of the 3D features are calculated on the fly and within a restricted neighborhood. We can identify and reconstruct the overlaid parts of the 3D boundary representation within the tetrahedral mesh. The implementation is based on the Computational Geometry Algorithms Library, which proved to have the functionality needed but also has its limitations.

KEYWORDS: Triangular mesh, tetrahedral mesh, overlay algorithms

Introduction

Current developments in three-dimensional (3D) sensors and measuring devices, such as terrestrial lasers canners, make it possible to model and represent real-world features so that they more closely resemble their actual shape. Until recently, geographic information science and cartography have been utilizing 3D models primarily for geo-visualization. Although visualization is useful for obtaining visual insight and performing qualitative analysis, for most applications a more quantitative analysis is needed.

We can distinguish between geometric queries performed for each 3D model separately (compute area, volume) and relational queries. The latter are more or less topological in nature, used to inquire whether or not geometrical representations overlap, and if they do, where and to which extent. These overlay calculations are well known and implemented in two dimensions (2D) between two or more planar partitions with the geometrical intersection and the propagation of the identifiers.

The overlay operation between 3D features, or between 3D volumetric partitions, is more complex. The geometric intersection has its challenges, and it should be performed within reasonable computational time. One way to manage that task is by decomposing the feature into a set of cells, such that a feature is defined as the union of all cells. In most applications this cell is a voxel, a 3D cubic primitive. Feature decomposition has the disadvantage that the representation of the feature is directly related to the measure of the voxel, giving it a rough appearance; but it has the benefit of requiring no intersection calculation when the spatial resolution of the objects to be overlaid is the same.

Another approach is to use a tetrahedral network (TEN) as the modeling environment in which the overlay is performed. The features are defined by their boundary representation as a 3D triangular irregular network (3D TIN). By inserting the features one by one into the overlay TEN, the geometrical overlay is performed on the fly, and the process is

Edward Verbree, Assistant Professor, OTB Research Institute for Housing, Urban and Mobility Studies, Delft University of Technology, Jaffalaan 9, 2628BX Delft, the Netherlands. E-mail:<e.verbree@otb.tudelft.nl>. **Arno van der Most**, M.Sc. Geodesy, GIS-Technology, Delft University of Technology. E-mail:<avdmost@planet.nl>. **Wilko Quak**, Assistant Researcher, OTB Research Institute for Housing, Urban and Mobility Studies of Delft University of Technology. E-mail: <w.quak@otb.tudelft.nl>. **Peter van Oosterom**, Professor, Technology, Policy and Management, Delft University of Technology. E-mail:<oosterom@otb.tudelft.nl>.

supported by the internal neighborhood search possibilities of the TEN. The characteristics, given by the identifier of the features, are propagated to the tetrahedron primitives within the overlay TEN. The overlay TEN then acts as a container in which all the features are stored. One can retrieve the features as a volume representation given by a set of tetrahedra with the same identifier, or as a surface representation given by the set of boundary facets of the tetrahedra. An overlap is detected by finding tetrahedra that belong to more than one feature, querying the tetrahedra and creating the overlay using these identifiers.

This paper first addresses the use of TIN/TEN data models with respect to boundary representations. Then we give some definitions and provide a background to triangulations in general, focusing on Cavalcanti and Mello's (1999) approach to decomposing a polygon boundary representation by a conformal triangulation. An overview of existing map overlay methods is given in the following section to explain the need for the overlay of 2D and 3D features, which are described in the next two sections. A short description of the implementation of the 2D counterpart of this method is followed by a summary of the 3D results. We conclude by highlighting some other possibilities of the proposed method.

TIN/TEN Data Models for Boundary Representations

In topographical modeling, real-world objects are characterized by a certain representation and stored in a geo-database. The representation depends on the purpose of the geo-database but also on the identification and data capture process. In this section we will introduce 2D TIN and 3D TEN data models, as they will be used in the subsequent overlay computations.

Two-dimensional (2D) Boundary Representation

We can describe a 2D boundary explicitly by its polygon geometry through an ordered list of points connected to one another by straight-line segments, with the last point connecting to the first. We could also use topology, where points are identified as numbered nodes and the nodes define a polygon. The geometric description of polygons and other simple features within 2D space is defined by the Simple Feature Specification (SFS) of the Open GeoSpatial Consortium (OGC 2005). Validation functions are available to determine whether or not a given feature is valid, i.e., a polygon is not self-intersecting. Although the validation process is not complicated, many implementation and definition problems exist (see Van Oosterom et al 2004; Van Oosterom et al 2005).

If the boundary of the object is defined, then the interior is also, in some way, given. This interior could also be made explicit or materialized, i.e., when the boundary polygon is triangulated in a set of triangles and stored within a 2D Triangular Irregular Network (2D TIN) data structure. Because boundary edges can be derived from a set of triangles, it is no longer necessary to store the boundary polygon per sai. The TIN acts as the base data structure for the feature representation. The embedded space between two or more objects could also be defined by a triangulation of the covering polygon, where all objects are enclosed by sets of interior triangulations. The space is thus fully partitioned by triangular meshes.

3D Boundary Representation

The identification and capture of real-world objects is far more complicated when we move to full 3D applications. In 3D, opposed to 2.5D, it is not possible to assume that objects can be flattened and defined as polygon footprints on a surface. Instead, a proper 3D boundary representation of the object is needed, meaning more than one Z-value is attached to a 2D polygon, and also more information is needed than one Z-value attached to the vertices of the 2D polygon.

In 3D, the definition of the geometric and/or topological description and validation of the correctness is far more complicated than in 2D. The polyhedral approach, as described in, among others, Teunissen and Van Oosterom (1988) and Stoter (2004) defines the boundary as a set of polygons, where each polygon is "flat" and valid according to the SFS of the OGC, i.e., non-self intersecting. The set of faces should enclose a single volume, meaning it should be "watertight" or "closed."

In fact this kind of explicit boundary representation is quite common in engineering disciplines, where 3D objects are actually measured, i.e., by lasers canning tools. Constructive solid geometry (CSG), on the other hand, deals with combining natural geometric primitives, such as the sphere, cylinder, cone, and torus. These primitives are commonly treated as solids. Binary set operations (union, intersection, and difference) are used to combine more complex shapes. The boundary shape description of this complex shape is called the implicit surface representation. The boundaries of such representations are not measured but derived from the CSG by particles connected to a triangulation (Hart 1997).

The validation of explicitly defined boundary representations (either derived from the implicit surface of a CSG or given by a polyhedral representation) consists of validating the following assumptions: the polyhedron is non-valid when Eulers Law does not hold: the number of vertices (nodes) plus the number of faces minus the number of edges should be equal to 2. However, when the equation holds, it is still not certain that the polyhedron is valid, since the faces can intersect with each other. To determine whether or not faces do intersect is easier said than done.

One way to simplify this problem is to use a pre-processing step where each face is subdivided into triangular patches. Not only does this avoid the need to check on "flat" polygons, as a triangle is flat by definition, but also the intersection test can now be performed without the actual calculation of the intersecting points (Möller 1997). And, triangular faces are easier to manage within a topological data model.

When an object is described by a valid, watertight set of triangles, its interior is also given. As in the 2D scenario, this interior can be made explicit by a set of tetrahedrons, which fill the interior completely. A tetrahedronization algorithm performs this task. Once this TEN is available, the boundary can be derived. The boundary is represented by a set of triangles, and it could be explicitly made available within a 3D TIN.

Triangulations

In this section we discuss how to store our input data within a TIN/TEN representation, as needed by the overlay computation.

Triangulations: General Issues

In n-dimensions, a triangulation of a set of points *V* is a set of non-intersecting simplices *T(V)*, whose vertices form *V*, and whose union completely fills the convex hull of *V*. Tetrahedra are dimension 3 simplices whose faces are simplices of dimension 2 (triangles), whose edges are simplices of dimension 1 (segment lines), whose vertices are simplices of dimension 0 (points).

A Delaunay triangulation is one of the many ways to triangulate a set *V*. In a Delaunay triangulation, for any simplex of *T(V)* there is an empty n-dimensional sphere that passes through all the vertices of that simplex. A sphere is said to be empty if there is no vertex in its interior. The incremental cavity algorithm (Watson 1981) deletes all n-dimensional simplices that are no longer empty, according the Delaunay circumsphere criterion, after the insertion of the new point. The algorithm then connects each n-1 facet at the cavity boundary to the newly inserted point to create new n-dimensional simplices. Another approach is the flip algorithm created by (Joe 1989), which generalizes to 3D the well known 2D flip algorithm of (Lawson 1972). The flip algorithm is described in detail below.

In some GIS-related literature (i.e., Pilouk 1996; Zlatanova 2000), a 3D tetrahedral mesh is called a TEN and a 2D triangular mesh is a TIN. We will follow this convention to emphasize the fundamental difference in handling 3D data and 2D data, the problems still existing in constructing TINs and TENs, and to emphasize that TENs are mostly used to represent (volumetric) 3D objects and TINs are typically used in 2D computational geometry applications and to represent 2.5D and 3D surfaces.

Conformal / Constrained (Delaunay) Triangulations

As reported by Cavalcanti and Mello (1999), it is still a problem to create a tetrahedral mesh that represents 3D objects bounded by polygonal faces or interior-constraining faces and edges. Traditional unconstrained Delaunay triangulations (Watson 1981; Joe 1989) use only point datasets.

However, a point mesh generator can be used as the foundation for the implementation of a conformal/constrained mesh generator. This is achieved by either adding extra (Steiner) points to the triangulation on missing faces, or by flipping faces on the tetrahedra of the starting mesh. Although in some literature references (also in Cavalcanti and Mello 1999), constrained Delaunay triangulations are utilized, we will follow the more strict definition of conformal triangulations where Steiner points are inserted to represent the polygonal faces by a set of triangular faces within the triangulation. If this can be implemented in such way that the Delaunay criterion is obeyed, it is a conformal Delaunay triangulation.

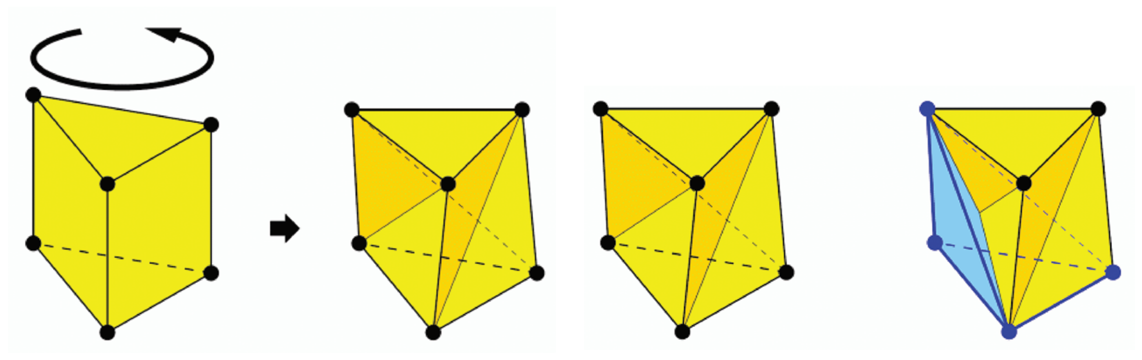
Figure 1. Schönharts polyhedron.

If the polygonal faces can be recovered without adding extra Steiner points, the triangulation is constrained. As this task generally cannot be performed without obeying the Delaunay criterion, strictly constrained Delaunay triangulations are very rare. One good example of a 3D polyhedron which has to be treated as a conformal triangulation is the Schönharts polyhedron (Aichholzer et al. 2002; Shewchuk 2002), as there is no possibility to create a constrained triangulation for this object (see Figure 1).

Conformal TEN Data Models for Storing Polygonal Boundary Representations

Cavalcanti and Mello (1999) describe an approach to generating a conformal tetrahedronization using the following steps:

- **Step 1:** Generate an unconstrained Delaunay tetrahedral mesh (TEN) using the object point set. This is performed by the flipping algorithms as given by Joe (1989).
- **Step 2:** Recover missing constrained edges. Steiner points are inserted in the tetrahedronization at the midpoint of the missing edge, and also on the corresponding bounding edge of the polygon, followed by a re-tetrahedronization, and check for missing constrained edges. This process will iterate until all constrained subdivided edges are part of the TEN. This method is followed by Kraak and Verbree (1992). Till now the TEN is still Delaunay, but not constrained. The missing edges could also be recovered by edge flipping, but the TEN is than not longer Delaunay.
- **Step 3:** Recover missing polygon faces. Now all (subdivided) edges are part of the TEN, but this does not automatically mean that all constrained facets are also part of the TEN. Therefore, the polygon transformations are also triangulated in 2D to obtain a triangular representation. All triangles that do not have a counterpart within the TEN will yield new Steiner points at the midpoint of the missing edge of the triangle. The updated TEN is still Delaunay. As this process does not converge always, a forth step is necessary.
- **Step 4:** Perform local triangulation of tetrahedra if there are some faces still missing. When polygon faces are not recovered within the TEN, the global Delaunay criteria are abandoned and a local re-triangulation of the tetrahedra is performed, whose interiors are intersected by a missing polygon.

Existing Map Overlay Methods

Overlays in the Raster Domain

Cartographic modeling has successfully been applied in the analysis and synthesis of geographical data (Tomlin 1990; DeMers 2001). This kind of map algebra works by decomposing data sets, data processing capabilities, and data processing control specifications into elementary components that can be recombined with relative ease and great flexibility. The complete body of data for a given geographical study area exists as a cartographic model consisting of map layers. Each layer is a two-dimensional image on which every location is associated with exactly one characteristic. The data-processing capabilities as given by Tomlin (1990) are to be performed between map layers and are intended to facilitate the interpretation of cartographic data. The LocalProduct operator, for example, multiplies each location's value on one specified layer to its value(s) on one or more additional layers.

Although, in our opinion, the term "cartographic" modeling is not accurate, since the data processing is performed on the data layers that are behind the map visualization, it shows clearly the advantages of this kind of "map combinations." Each layer models a certain aspect of the real world; it is a natural to organize data from different. These single-valued, general-context data sources are combined in a more specific context without the need to model the relationships between the input maps beforehand. The values express geographical characteristics, and the mathematical functions (summation, multiplication, etc.) express relations as more information is revealed during map overlay process.

Overlay in the Vector Domain

Although some of the implementations of cartographic modeling, such ArcGIS Spatial Analyst (ESRI, 2005), describe the spatial relationships only on raster data, there is nothing that prevents the same method from being applied to vector (polygon) data set. To do this, the local, zonal, and focal operations need to be defined in the vector domain, and, importantly, a vector map overlay has to be performed.

The map overlay of two (or more) polygon-based feature maps is usually computed in three logical phases (Frank 1987; Franklin 1990). The first step is performed at the metric level and computes all intersections between the edges (line segments) of the different layers. The second step is the topological reconstruction of the resulting new polygons, and the third step is the assignment of the identifiers. The computational drawback of most algorithms is that the line segments should be sorted based on their X-coordinate to be intersected in an efficient way, i.e., plane-sweep (Van Roessel 1990; Kriegel et al. 1991). The map overlay algorithm by Van Oosterom (1994) attempts to overcome that limitation by using the R-tree spatial index structure to test whether two line segments are candidates for intersecting. If two line segments overlap, then an intersect function has to determine whether the lines really intersect, after which four new line segments with the propagated identifiers are returned. When all intersections are computed, the resulting line segments have to be assembled to form new polygons. Finally, the identifiers of the original maps have to be set or combined into a new value and attached to the new area features.

Limitations of Existing Overlay Methods

The map overlay as performed by cartographic modeling is limited to the propagation and functional calculation of cell values obtained from the original data sets. No intersection calculation is needed, as the input raster cells are similar for all data sets.

This said, cartographic modeling can be performed with ease in three-dimensional space, as voxel cells are the three-dimensional equivalents of the two-dimensional raster cells. But polygon feature map layers should first be rasterized, and three-dimensional polyhedral features need to be transformed into a voxel representation. Depending on the cell size of the pixels or voxels selected, the results could be less accurate, and large data storage or high calculation times may entail.

The map overlay of polygon-based feature maps requires an intersection of all line segments. To speed up this calculation, either the line segments need to be sorted or an efficient spatial index and clustering must be made available. Both options will take some time to perform, to build, and to tune. And one can assume the complexity of implementing an efficient and correct map overlay algorithm of three-dimensional polyhedral features.

Examples of True 3D Overlays

For most applications, the real world is modeled through representations that fit their purposes. As our activities are in general related at the earth surface, we can use 2D or 2.5D (with height added) data structures, and thus also 2D overlays algorithms. But as soon as the space above or below the ground is occupied, the third dimension has to be taken into account. That's why in geology, 3D modeling is needed.

Another good example of 3D, or even 4D, modeling is given by in flight navigation systems made possible by new on board systems based on the 'tunnel in the sky' concept. Restricted areas, like military sites are shown at a display in the cockpit of the aircraft. But it is also possible to show the location of nearby airplanes with their intended routes. If a route overlays with the one of the own airplane the system will warn the pilot and ground control.

More examples and applications on 3D GIS for Urban Development and 3D Cadastre can be found in within the PhD work of (Zlatanova 2000) and (Stoter 2004).

2D Feature Overlay in TIN

Background

To overcome the limitations of the existing overlay methods the datasets themselves could be manipulated for the overlay operator. This can be accomplished with a data structure that is capable of storing the polygon features and has an internal spatial search structure that can be used within the geometrical intersection process of the map overlay. In addition, efficient propagation of the identifiers attached to the polygon features to the new overlaid map is required. We believe that the TIN of two-dimensional polygon feature maps and the TEN of three-dimensional polyhedral feature maps are data structures that can attain that goal.

The idea behind feature overlay in TIN/TEN data models is to perform a map overlay by inserting all features, expressed as a polygonal boundary representation, in one triangulation. We will describe this approach by first explaining the 2D feature overlay in a TIN, and then the 3D feature overlay in a TEN. The main advantage of this method is that the insertion of the features, the geometrical intersecting of the line segments (in 2D) and the polygon facets (in 3D), is performed in a local neighborhood. After the insertion and intersection are completed, the overlaid features are re-assembled and the identifiers of the original features are propagated and attached to the overlaid features.

Feature Overlay Algorithm Project Strategy

We use the following strategy to deal with the highly complex algorithms:

1. Formulate a general feature overlay algorithm;
2. Elaborate this algorithm with two-dimensional sub algorithms; and
3. Translate the two-dimensional sub-algorithms to their three-dimensional counterparts.

2D Feature Overlay Algorithm

First an empty overlay triangulation is performed. Then the polygon features are inserted into it one by one. Their edges are processed iteratively. When a new feature edge intersects with a constrained TIN edge previously inserted into the overlay TIN, this feature edge is split into sections. The same applies to the intersected constrained TIN edge. When all sections of the feature edges of all polygon features are inserted into the overlay TIN, the entire polygon feature set is overlaid, and the post-processing phase can begin. The post processing involves re-assembling the overlaid features and propagation of the original identifiers. This procedure is illustrated in Figure 2.

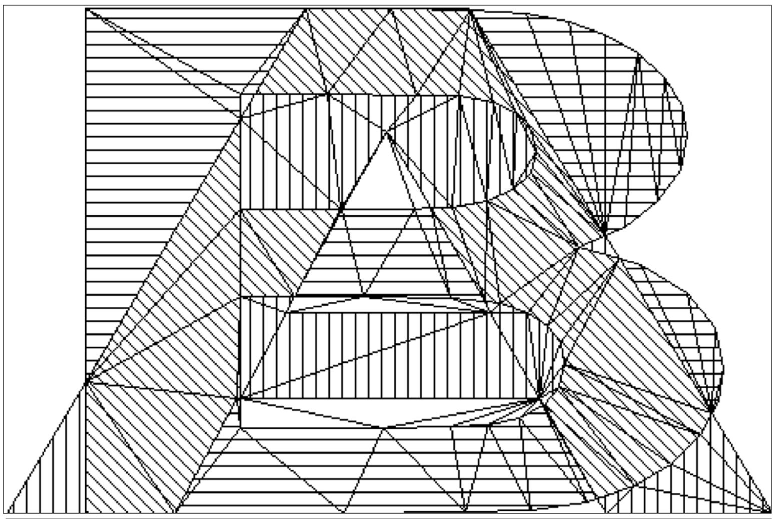
Figure 2. Result of overlaying polygon features of 'A' and 'B'.

In Detail: Inserting 2D Feature Edges by Flipping

The algorithm inserts feature edges into the overlay TIN in three steps. The feature edge is inserted into the overlay TIN by inserting its start and its end node. This was implemented in the Computational Geometry Algorithm Library (CGAL 2005) environment, using a point location algorithm from the family of triangle line walk. The line walk involves inserting the start and end nodes and then "walking" from the start to the end node until all intersected TIN edges in the overlay TIN are detected. What happens with the inserted feature edges depends on the type of the intersected TIN edges. When the intersected TIN edges are constrained, the intersecting points are determined and inserted into the overlay TIN. The intersecting feature edge is then split into segments, as is the intersected feature edge. These segments start a flipping process, such that the intersecting segments are represented by constrained TIN edges in the overlay TIN.

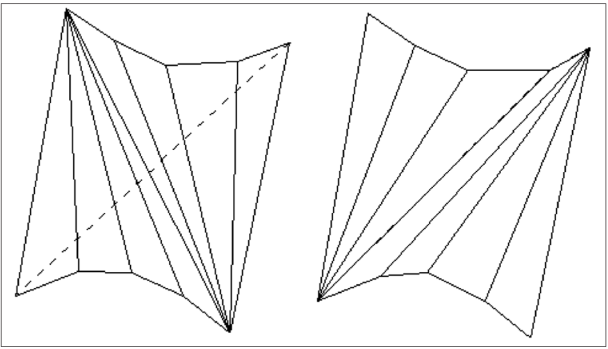

Figure 3. Worst-case scenario (n^2 flips) for non-flippable edge.

A section is constructed in the overlay TIN by flipping all non-constrained edges that intersect with this section until none of them intersects anymore. Flipping is not a trivial matter, as some configurations could pose problems that should be treated carefully. In Figure 3, a worst-case flipping scenario is given.

Re-assembling Overlaid 2D Features

The edges of the polygon features are processed in counter clockwise order. The counterparts of these edges are the constrained TIN edges in the overlay TIN. During the insertion of these constrained edges, the left-right information is propagated from the feature edges. A flood fill algorithm is initialized by providing it with a face that lies in the interior of a feature and an identifier of that feature. The algorithm searches for corresponding TIN faces in the overlay TIN by recursively checking its three neighboring faces.

3D Feature Overlay in a TEN Data Model

When developing the 3D feature overlay algorithm, it was assumed that the basic 2D algorithm could be extended to the third dimension, by using input data that are one dimension higher and performing the overlay in a 3D triangulation or TEN. Consequently, all subalgorithms were expected to work in 3D. Below we address the main difficulties encountered during the implementation of this approach.

3D Feature Overlay Algorithm

Note that in this algorithm, the input polyhedral features are described by their 3D boundary representations. These features should be "valid" and "simple." No intersections or gaps are allowed with respect to these features, thus they are also "watertight." To be "simple," all feature faces of the polyhedron are split into triangular, i.e., flat facets. The edges of these triangular facets are expected to be constrained, i.e., to be maintained within the overlay triangulation.

The algorithm itself is similar to the 2D algorithm. First an empty 3D overlay triangulation is initialized. All polyhedral features are inserted into this TEN one by one. The feature edges of the polyhedral triangulation are processed iteratively. As in the 2D scenario, the feature edges that intersect with a constrained TEN edge previously inserted into the overlay TEN causes this feature edge and the intersected feature edge to split into sections.

If all sections of the edges of a polyhedral feature are inserted into the overlay triangulation, a special check has to be performed, as the presence of all (sections of the) feature edges does not necessarily guarantee that all features facets are present in the triangulation. Calvacanti and Mello (1999) encountered this problem while computing a TEN from one polyhedron (see previous discussion). In our case, the polyhedral features were preprocessed to simple triangular facets, so that the test of missing feature facets was not difficult to perform. This notwithstanding, the problem of the missing feature facets is still to be tackled. Below are some thoughts on how to handle this problem.

In Detail: Inserting 3D Feature Edges by Flipping

Flipping in 3D is far more complex than in 2D. The main reason is that the so-called bistellar flipping of an edge can cause the destruction of two tetrahedra and the creation of three new tetrahedra, and vice versa. Besides this 2-3 flip, a 4-4 flip exists, where four tetrahedra are destroyed and four new ones are created. The 4-4 flip is, however, only possible when two TEN facets in adjacent tetrahedra lie in the same plane (Figure 4).

Here the algorithm inserts feature edges into the overlay TEN by inserting the start and end nodes of the feature edge (which causes some flips) into the overlay TEN and by walking from the start to the end node to detect all intersected TEN edges and TEN facets. The feature edges are split into segments at the intersecting edges by inserting the intersecting vertices in the overlay TEN (which again causes some flips). When the start and end vertex of the section are present in the overlay TEN, all TEN facets that intersect with this section need to be flipped out of the way. Figure 5 shows a segment that needs to be inserted with one last intersecting TEN facet. After flipping the facets with a 2-3 flip, the segment is propagated in the overlay TEN. But, contrary to the 2D feature edge case, no proof exists that under all conditions all feature edges are preserved in the overlay TEN (Anwei and Baida 2000).

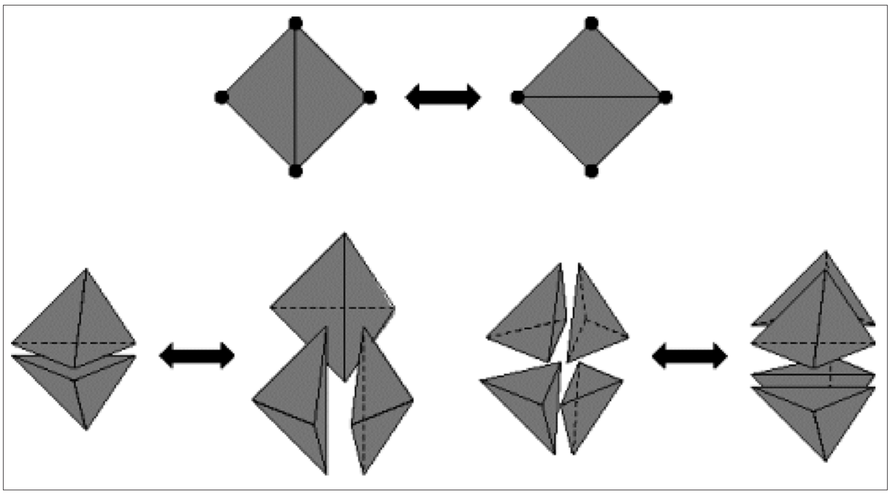
Figure 4. Above: 2D flip; below: 2-3 bistellar 3D-flip and 4-4 bistellar 3D-flip.

In Detail: Confirming 3D Feature Facets by Flipping

Compared to the conforming method described by Calvacanti and Mello (1999), this algorithm is very economical with regard to inserting extra points into the triangulation. It is only when feature edges (from different feature polyhedra) intersect with TEN edges that an intersecting point is inserted into the overlay TEN. No unnecessary Steiner points are inserted. However, inserting (sections of) feature edges is not sufficient to make sure that the feature facets are presents in the overlay TEN, since they still intersect with the facets of the inserted feature edges.

This problem can be solved in three steps. First all constrained TEN facets are marked. A TEN facet is constrained when its three TEN edges are constrained by the same feature (and thus have a feature segment as a counterpart). The second step is to find a tetrahedron that intersects with a constrained TEN facet, and then use the neighborhood relationships embedded in the TEN to detect all other intersecting tetrahedra. Once their constrained TEN facets are detected, the intersecting point is inserted into the overlay TEN. All other non-constrained facets are flipped out of the way.

As with the insertion of constrained edges, this algorithm has only partially been proven Shewchuk 2002; 2003), but experiments show that it works most of the time (Anwei and Baida, 2000).

Re-assembling Overlaid 3D Features

When the sections of the edges of the feature facets are inserted into the overlay TEN and all tetrahedra that intersect with the facets are processed, the entire polyhedral feature is present in the tetrahedral network. After all feature polyhedra have been inserted, a flood fill algorithm similar to the one used for the 2D feature overlay algorithm can be applied to assign a feature identifier to all of the tetrahedra in the overlay TEN that are inside the boundary of the feature.

The overlay TEN now acts as a container where all features are stored. One can retrieve the features as a volume representation given by a collection of tetrahedra with the same identifier, and as the surface representation given by the set of boundary facets of these tetrahedra. An overlay is detected by those tetrahedra that have more than one identifier, and, by querying the tetrahedra according to these identifiers, this overlay can be materialized.

2D Feature Overlay Algorithm Implementation

We have implemented the basic ideas of the feature overlay algorithm using CGAL (2005). CGAL is a collaborative effort of several sites in Europe and Israel. Its goal is to make the most important solutions and methods developed in computational geometry available to users in industry and academia in a C++ library. The intention is to provide easy access to useful, reliable geometric algorithms.

The CGAL library comprises a *Kernel* of geometric primitives such as points, vectors, lines, and predicates which is used to test, for instance, the relative positions of points, intersections, and distance calculation. Its *Basic Library* is a collection

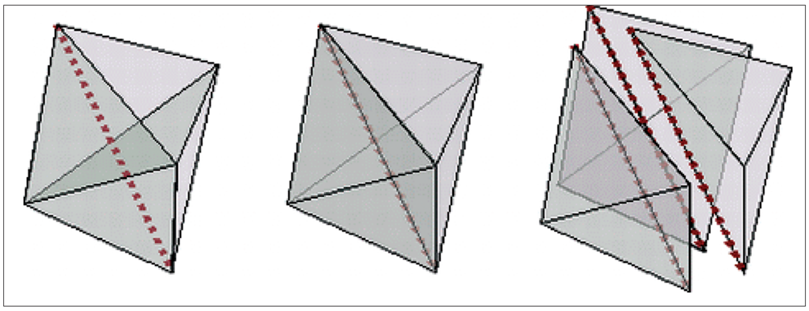
Figure 5. Last flip produces section as an edge.

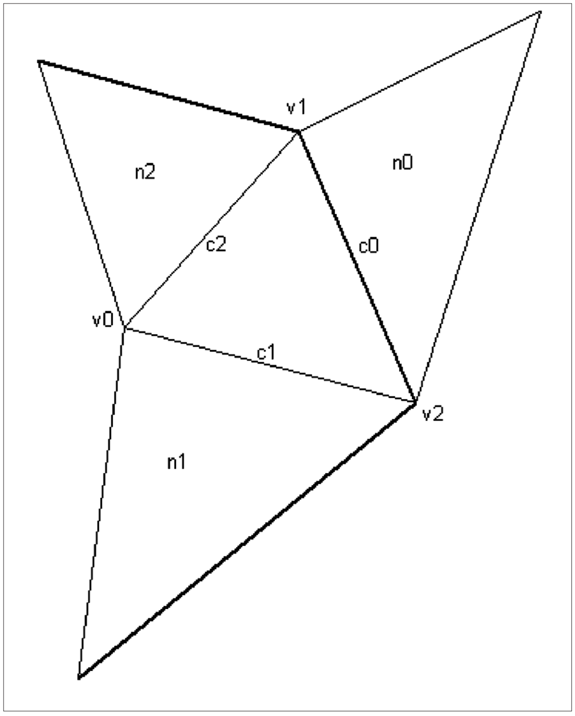

Figure 6. Face definition.

of standard data structures and geometric algorithms, such as convex hull in 2D/3D, (Delaunay) triangulation in 2D/3D, planar map, polyhedron, smallest enclosing circle, and multidimensional query structures. The *Support Library* offers interfaces to other packages, e.g., for visualization, and I/O and other support facilities.

Our goal was to investigate the available CGAL data structures for triangulation in both 2D and 3D and to identify the possibilities and limitations of CGAL. Besides, the concept of performing an overlay within a triangulation has to be proved to work. Currently, the algorithm itself is implemented only in 2D. Implementation in 3D is possible, but can only be done when the computational and geometrical difficulties as described in the previous paragraph are fully understood and tackled.

Class Definition of Constrained Faces

The implementation of the algorithm depends heavily on the explicit availability of edges in the data structure. However, the triangulation data structure used by CGAL does not contain explicit edges. These edges are only implicitly accessible by the vertices opposite to them in the same face. Our constructor of the constrained face class reads:

```
Constrained_face_2 (void* v0, void* v1, void* v2,
                    void* n0, void* n1, void* n2,
                    bool c0, bool c1, bool c2,
                    long face_id = 0)
        : Fab(v0,v1,v2,n0,n1,n2)
        {
        set_constraints(c0,c1,c2);
        set_id(face_id);
        }
```

In overlay triangulation, v0, v1 and v2 are the handles of constrained vertices while n0, n1 and n2 are the handles of the neighboring faces. These neighboring faces are chosen in such a way that neighbor n0 lies opposite to vertex v0. c0, c1 and c2 define constraints on the edges of the face (Figure 6).

For the reconstruction of the overlaid features each face will hold two feature_ids to identify in the features in which the face lies, this limits the implementation to overlay two features on top of each other.

Inserting features

The implementation follows the steps given above for 2D feature overlay in TIN by the following custom-code, non-standard CGAL functions:

```
// Framework functions:
void insert_obj(long object_id);
Vertex_handle insert_edge(const Vertex_handle
vt1, Point pt2, long object_id);
Vertex_handle insert_section(Vertex_handle
start_vertex, Vertex_handle end_vertex, long
object_id);
// Support functions:
Vertex_handle insert_vertex(const Point pt, long
object_id);
```

```
Vertex_handle insert_vertex_in_edge(const
Point pt, Face_handle f, int i,
long object_id);
void c_flip(Edge flip_edge);
Face_handle get_start_face(const Vertex_handle
vstart, Vertex_handle vend);
```

The insert_obj function is called each time an object has been loaded into the overlay_triangulation. The insert_edge function first uses the insert_vertex function to insert the start and end nodes of the edge into the overlay_triangulation. It then inserts all sections of the edge with the insert_section function. Because of the earlier mentioned shortcoming of the CGAL triangulation data structure (it does not contain edges explicitly), the sections cannot be determined before hand. The insert_section function determines each section on the fly, by processing the edge without the previously processed sections.

The insert_section function does the bulk of the processing needed to insert features into the overlay triangulation. First, it uses the get_start_face function to initialize the triangular walk algorithm. It then performs the triangular walk process until it reaches a constraint. When a constrained edge is found, it uses the insert_vertex_in_edge function to insert the end point of the section. The section's start vertex and end vertex are known, and the section can reconstructed using the c_flip function.

Results

Figure 7 shows a detail of a feature triangulation, which was created by an overlay of two datasets. The first (shown in green) consists of 200 features, the second (shown in orange) consists of 100 features. The overlay parts, enclosed by constrained (red) edges, are shown in brown. The unconstrained edges of the triangulation are shown in blue.

Conclusions and Recommendations

We have presented a method of storing triangular boundary representation objects in a tetrahedral mesh (TEN), where the intersection between the different features is calculated on the fly, and the overlay TEN is a constrained triangulation, as only the intersecting points are inserted in the TEN on top of the feature nodes, edges, and facets.

The overlay TEN acts as a container structure where each feature can be reconstructed as its tetrahedral volume or triangular surface representation. As each tetrahedron has an identifier(s) inherited from its originating feature, the inserted features can be materialized by querying the overlay TEN for these identifiers.

This method could also be used to validate "non-watertight" and self-intersecting boundary representations. If the overlay TEN is used to process these features, then the facets of this feature intersecting other facets will be found and corrected while inserting.

We have tested and implemented this method for the 2D scenario. As far as we know, in itself, this is a completely new approach. More research is needed to compare the performance with other 2D overlay methods. A fully proven and operational implementation within 3D still needs to be done.

ACKNOWLEDGMENTS
This paper is the result of the research programme of the Delft Centre for Sustainable Urban Development, part of the Delft University of Technology. The remarks of the reviewer on the first version of this paper as presented at Auto-Carto 2005 are greatly appreciated, as is the proofreading by Elfriede Fendel. The final editorial review has improved the paper considerably.

REFERENCES

Aichholzer, O, L. Alboul, and F. Hurtado. 2002. On flips in polyhedral surfaces. *International Journal of Foundations of Computer Science (IJFCS)* (Special issue on Volume and Surface Triangulations) 13(2): 303-11.

Anwei, L., and M. Baida. 2000. How far flipping can go towards 3D conforming/constrained triangulation. In: *Proceedings 9th International Meshing Roundtable*, Sandia National Laboratories, New Orleans, Lousiana, pp: 307-315.

Calvacanti, P.R., and U.T. Mello. 1999. Three-dimensional constrained Delaunay triangulation: A minimalist approach. In: *Proceedings, 8th International Meshing Roundtable*, South Lake Tahoe, California, USA. pp: 119-29.

CGAL (Computational Geometry Algorithm Library). 2005. www.cgal.org [accessed January 2005].

DeMers, M.N. 2001. *GIS modeling in raster*. [city, state?]: Wiley, New York

Frank, A.U. 1987. Overlay processing in spatial information systems. In: *Auto-Carto 8*, ACSM. pp: 12-31.

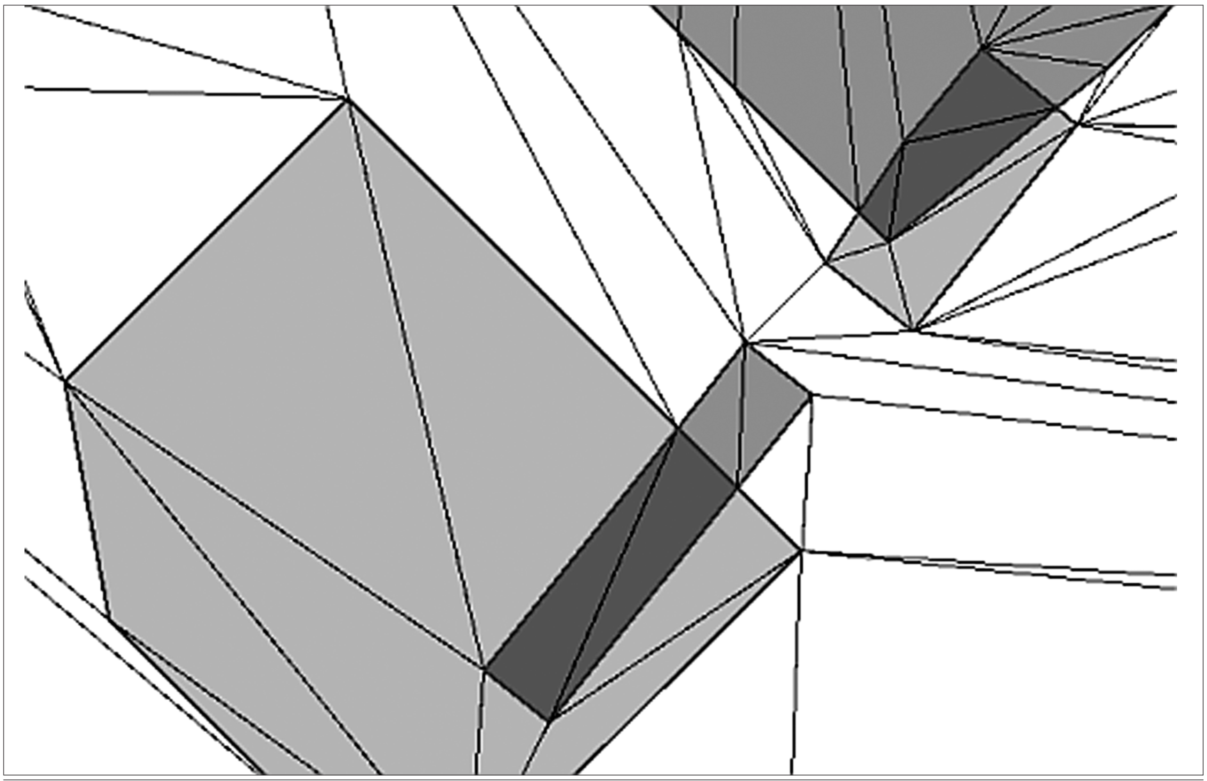

Figure 7. Detail of result of 2D feature overlay.

Franklin, Wm. R. 1990. Calculating map overlay polygons' areas without explicitly calculating the polygons: Implementation. In: Proceedings, *4th International Symposium on Spatial Data Handling*, Zürich, Switzerland. pp: 151-60.

Hart, J. C. 1997. Morse theory for implicit surface modeling. In: *Hege, H.-C. and Polthier, K., eds., Mathematical Visualization*, pp. 257--268. Springer-Verlag, Heidelberg, 1998.

Joe, B. 1989. Constructing of three-dimensional triangulations using local transformations. *SIAM Journal of Scientific and Statistical Computing* 10(4): 718-41.

Kraak, M.-J., and E. Verbree. 1992 . Tetrahedrons and animated maps in 2D and 3D space. In: E. P. Bresnahan, E. Corwin, and D. Cowen (eds), *Proceedings of the 5th International Symposium on Spatial Data Handling*, IGU Commission of GIS, Charleston, 1992. pp: 63-71.

Kriegel, H.-P., T. Brinkhoff, and R. Schneider. 1991. The combination of spatial access methods and computational geometry in geographic database systems. In: *Advances in Spatial Databases*, 2nd Symposium, SSD'91, Zürich, Switzerland. Springer-Verlag. pp: 3-21.

Lawson, C.L. 1972. Transforming triangulations. *Discrete math*ematics 3 pp: 365-372.

Möller, T. 1997. A fast triangle-triangle intersection test. *Journal of Graphics Tools* 2(2): 25-30

OGC (Open GeoSpatial Consortium) 2005. Simple feature specification. [http://www.opengeospatial.org/specs/. Accessed January 2005].

Pilouk, M. 1996. *Integrated modeling for 3D GIS*. PhD thesis, ITC, the Netherlands.

Shewchuk, J.R. 2002. Constrained Delaunay tetrahedralizations and provably good boundary recovery. In: Proceedings, *Eleventh International Meshing Roundtable*, Ithaca, New York. pp: 193-204

Shewchuk, J.R. 2003. Updating and constructing constrained Delaunay and constrained regular triangulations by flips. In: *Proceedings of the Nineteenth Annual Symposium on Computational Geometry*, San Diego, California. pp: 181-90.

Stoter, J.E. 2004. 3D cadastre. *PhD thesis*, Delft University of Technology, Nederlandse Commissie voor Geodesie (NCG).

Teunissen Wim J.M., and P. J.M. van Oosterom. 1988. The creation and display of arbitrary polyhedra in HIRASP. *Technical report*, University of Leiden, Department of Computer Science, Report 88-20.

Tomlin, C. D. 1990. *Geographic information systems and cartographic modeling*. Englewood Cliffs, NJ: Prentice Hall.

Van Oosterom, P.J.M. 1994. An R-tree based map overlay algorithm. In: *EGIS/MARI'94*, Paris,
March 29 - April 1, pages 318-327

Van Oosterom, P.J.M., C.W. Quak and T.P.M. Tijssen. 2003. Polygons: the unstable foundation of spatial modeling, *ISPRS Joint Workshop on "Spatial, Temporal* *and Multi-Dimensional Data Modeling and Analysis"*, Québec, Canada.

Van Oosterom, P.J.M., W. Quak, and T. Tijssen. 2004. About invalid, valid and clean polygons. In: Fisher, P. F. (ed.), *Developments in Spatial Data Handling*. Springer Verlag, Berlin. pp: 1-16.

Van Roessel, J. W. 1990. Attribute propagation and line segment classification in planesweep overlay. In: Proceedings, *4th International Symposium on Spatial Data Handling*, Zürich, Switzerland. pp: 127-40.

Watson, D.F. 1981. Computing the n-dimensional Delaunay tesselation with applications to Voronoi polytypes. *The Computer Journal* 24(2): 167-72.

Zlatanova, S. 2000. 3D GIS for urban development. *PhD thesis*, Graz, Austria.

■