Compactly representing massive terrain models as TINs in CityGML

Kavisha Kumar, Hugo Ledoux, and Jantien Stoter

3D Geoinformation, Delft University of Technology, The Netherlands

Kumar K, Ledoux H, Stoter J. Compactly representing massive terrain models as TINs in CityGML. *Transaction in GIS*. 2018;00:1-28, In Press.
DOI: `https://doi.org/10.1111/tgis.12456`

The code of the project is available at
`https://github.com/tudelft3d/CityGML_iTINs_ADE`

Terrains form an important part of 3D city models. GIS practitioners often model terrains with 2D grids. However, TINs (Triangulated Irregular Networks) are also increasingly used in practice. One such example is the 3D city model of the Netherlands (3DTOP10NL) which covers the whole country as one massive triangulation with more than one billion triangles. Due to massive size of terrain datasets, the main issue is how to efficiently store and maintain them. The international 3D GIS standard CityGML allows to store TINs using Simple Feature representation. However, we argue that it is not appropriate for storing massive TINs and has limitations. We focus in this paper on an improved storage representation for massive terrain models as TINs. We review different data structures for compactly representing TINs, and explore how they can be implemented in CityGML as an ADE (Application Domain Extension) to efficiently store massive terrains. We model our extension using UML, and XML schemas for the extension are automatically derived from these UML models. Experiments with massive real-world terrains show that, with this approach, we can compress CityGML files up to a factor of ∼20 with one billion+ triangles, and our method has the added benefit of explicitly storing the topological relationships of a TIN model.

1 Introduction

The use of 3D city models for urban planning and management has increased in recent years. Several cities like Rotterdam, Brussels, and Berlin have already created 3D city models for use in different applications such as noise mapping, estimating energy demand of buildings, and calculating building rooftop solar irradiation (Biljecki et al., 2015). However, in practice these applications are mostly centred around buildings; other terrain features like

vegetation, roads, and water bodies are often ignored. Formal specifications for modelling buildings in 3D space are often more prominently defined than other urban features. For example, in the international 3D GIS standard CityGML, the concept of LODs (Levels Of Detail) is very well established for buildings and bridges, but is vague in case of terrains and landuse (Open Geospatial Consortium, 2012).

Over the last few decades, grids and TINs (Triangulated Irregular Networks) have become the two most popular models for representing terrains. GIS practitioners often model terrains as grids. However, grids have several shortcomings. First, they cannot be used to represent terrains with vertical walls and overhangs which are quite common in cities (we give precise definitions of these in Section 2). Secondly, grids, being restricted to 2.5D, do not conform well to the variability in terrain complexity. This might result in loss of sample points which could be important for spatial analysis such as points representing balconies, dormers, chimneys, vertical walls, and banks of the canals (Fisher, 1997). Another disadvantage is that grids can be very large for fine resolution terrains (Fisher, 1997). For instance, in case of 3D grids, the size of voxels (3D pixels) increases as the resolution of data increases, which requires more storage space (Stoter and Zlatanova, 2003). On the other hand, TINs have numerous benefits. In a TIN, the local density of points can be altered based on the variations in the height of original terrain (Kumler, 1994). For example, areas of detailed relief can be represented in a TIN with a denser triangulation than areas with a smooth relief (Kumler, 1994). Another advantage is that the points describing balconies, dormers, chimneys, and vertical walls can be well represented as constraints in a TIN (Kumler, 1994). However, storing TINs is more complicated than storing grids, as it requires not only storing the TIN geometry but also efficiently storing and querying the topological relationships between the triangles. A terrain can be stored either as one massive TIN with continuous elevation values or as a constrained TIN with 3D objects like buildings, roads, and vegetation as constraints in the triangulation.

CityGML supports the storage of DTMs (Digital Terrain Models) as TINs but it is not efficient for storing massive TINs. Generally, the number of triangles in a TIN is roughly twice the number of vertices used in triangulation (De Berg et al., 2000). The CityGML datasets can become very large for massive TINs because of the redundancy in the underlying data structure, which greatly hinders web-based rendering and exchange of data. Moreover, there is very little topological information stored, which prevents us from efficiently using the datasets for analysis. For instance, 3DTOP10NL (Kadaster, 2015), the 3D city model of the Netherlands, covers the whole country (including buildings, roads, water bodies, and bridges) as one massive triangulation with more than one billion triangles (Figure 1). CityGML requires a file size of $\sim$700 GBs just to store the geometry of 3DTOP10NL terrain dataset (without any topological information).

Therefore, the main focus of this paper is to develop an improved representation for storing massive terrains as TINs in the context of 3D city models. This paper is an actual implementation and extension to the ideas that we proposed in the initial phase of the research (Kumar et al., 2016b,a). In this paper, we review different data structures for compactly representing TINs, and explore how they can be implemented in GML/CityGML to efficiently store massive TINs. The research is not limited to model terrains as 2.5D TINs. It also includes vertical walls, overhangs and constraints in the terrain model (see Section 2). Three existing compact TIN data structures, namely *Indexed triangles* (Ravada et al., 2009), *TriStrips* (Speckmann and Snoeyink, 2001), and *Stars* (Ledoux, 2015) are introduced as new geometry types in GML geometry model for representing TINs. These new geometry types are extended to CityGML as an ADE (Application Domain Extension) for compactly representing massive TIN terrains (see Section 3). We model the extension using UML (Unified Modelling Language). XML schemas for the extension are automatically derived

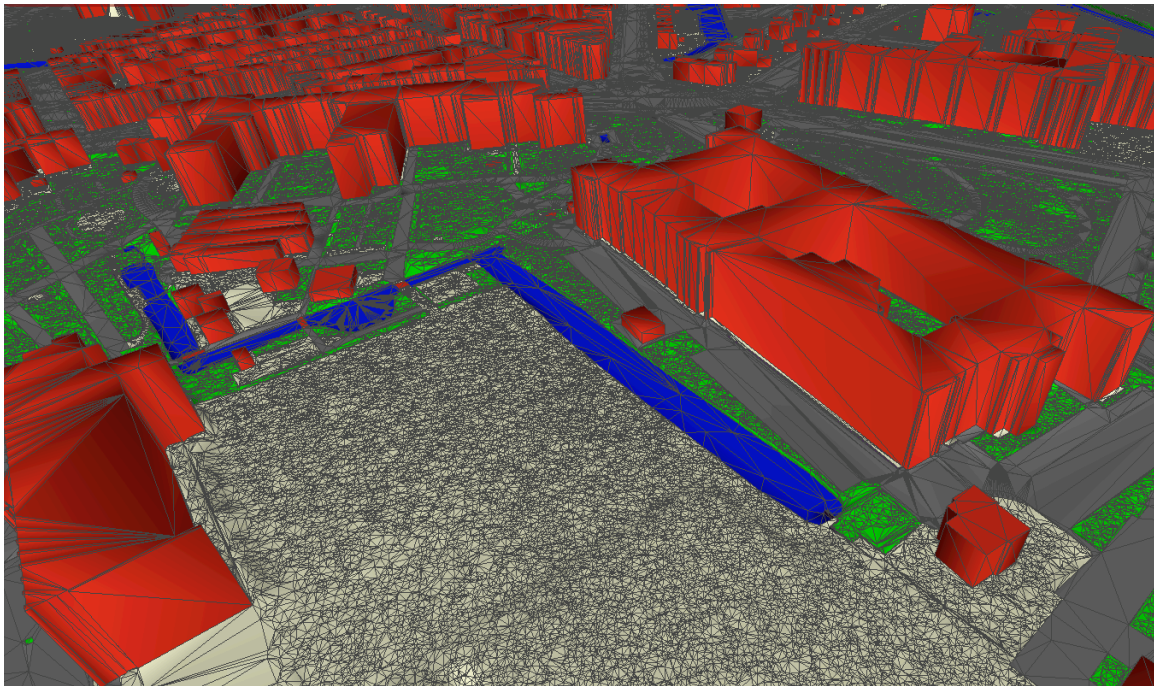


Figure 1: Snapshot of 3DTOP10NL dataset of a part of Delft, the Netherlands. Note that the terrain is one massive TIN with buildings, roads, water bodies and other features. CityGML requires ∼700 GBs of storage space just for storing the 3DTOP10NL terrain geometry

from these UML models. We made a prototype to implement these TIN data structures in CityGML datasets. We tested our proposed CityGML extension with several real-world datasets and we report on the compression factors achieved in Section 4. Our approach allows to compress up to a factor of ∼20 with massive real-world terrain datasets. For example, the storage space required for the 3DTOP10NL terrain in a CityGML file is reduced from ∼700 GBs to nearly ∼40 GBs. Moreover, our method has the added advantage of explicitly storing the topological relationships of a TIN model. We close the paper with conclusions and future work in Section 5.

2 State of art in modelling terrains with TINs

Terrain (Latin *Terra* meaning Earth) in simple terms refers to the lay of land described in terms of elevation, slope, or other attributes of the landscape (Wikipedia, 2017). Modelling the terrain surface with precision has always been a challenge for the geo-researchers. The irregular nature of the surface makes it difficult to depict the true model of a terrain. In this section, we provide an overview of different TIN representations used for modelling terrains. Several data structures have been proposed in different domains to represent and store TINs; they exhibit data redundancy and also store a lot of information for maintaining the adjacency relationships. We review different TIN data structures that can be integrated efficiently in the GML3 geometry model and extended to CityGML for representing massive terrains.

2.1 Representation of Terrains

A terrain is usually modelled as a grid of elevation values or as a TIN. These are also referred to as field representations in GIS (Kumler, 1994; Cova and Goodchild, 2002). A *field* is a

model of spatial variation of an attribute over a spatial domain (Ledoux, 2017). Fields are generally used to represent continuous geographical phenomena such as elevation of a terrain, surface temperature, etc. (Ledoux, 2017; Cova and Goodchild, 2002). A terrain can be modelled as a field, by a function $f(x, y)$ mapping each (x, y) location in spatial domain to an elevation value (z) i.e. $z = f(x, y)$ (Figure 3(a)).

Modelling terrains by storing only one elevation value (z) for any (x, y) location is referred to as *"2.5D"* (Figure 3(a)). Topologically, the surface depicted by a TIN is a *2-manifold* i.e. each edge of the TIN is incident only to one or two triangles and the triangles incident to a vertex form either a closed or an open fan (Gotsman et al., 2002) (Figure 2).

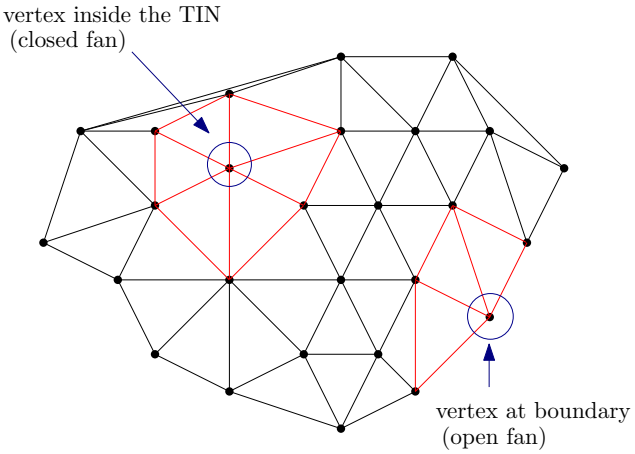


Figure 2: 2-manifold TIN. Each edge of the TIN is incident only to one or two triangles of the TIN

However, it is not possible to represent features like vertical walls, roof overhangs, caves/-tunnels, and overfolds like balconies and dormers with 2.5D field models. For instance, 3DTOP10NL terrain data has vertical walls. Modelling it in 2.5D will result in loss of information points representing the vertical walls. Therefore, we focus on geometrical representations which extend the field based 2.5D model to handle such features. In Figure 3(b), an example is shown where a location (x, y) has more than one elevation value (z) to model the vertical walls of natural or man-made objects like buildings. It is a so-called *2.5D+* model which is topologically equivalent to 2.5D model as it is still a 2-manifold (Penninga, 2008).

The ISO 19107:2003 Spatial Schema (ISO, 2003) standard defines `GM_TIN` geometry type for representing TIN models, which in theory should allow vertical triangles in a TIN and therefore can be referred to as a 2.5D+ data structure. Features like balconies, and overhangs of rocks and roof surfaces are not covered by these models and are described using *2.75D* models (Gröger and Plümer, 2005; Tse and Gold, 2004). A *2.75D* model is a 2.5D+ model extended to model any 2-manifold surface with features like balconies and overhangs (Figure 3(c)). These models are described in the context of TINs and not grids. They are sufficient for applications like visualization and watershed modelling (Lyon, 2003).

However, for some applications, even 2.5D+ and 2.75D models have limitations. For instance, applications estimating population and building energy demand using 3D city models require to compute the volume of buildings (Biljecki et al., 2015) which is not possible to calculate using these terrain models. To compute the volume of a building, it should be closed at the base i.e. modelled as solids. Based on the above argument, we refer to *3D* model of a terrain as a 2.5D+/2.75D model with buildings modelled as solids

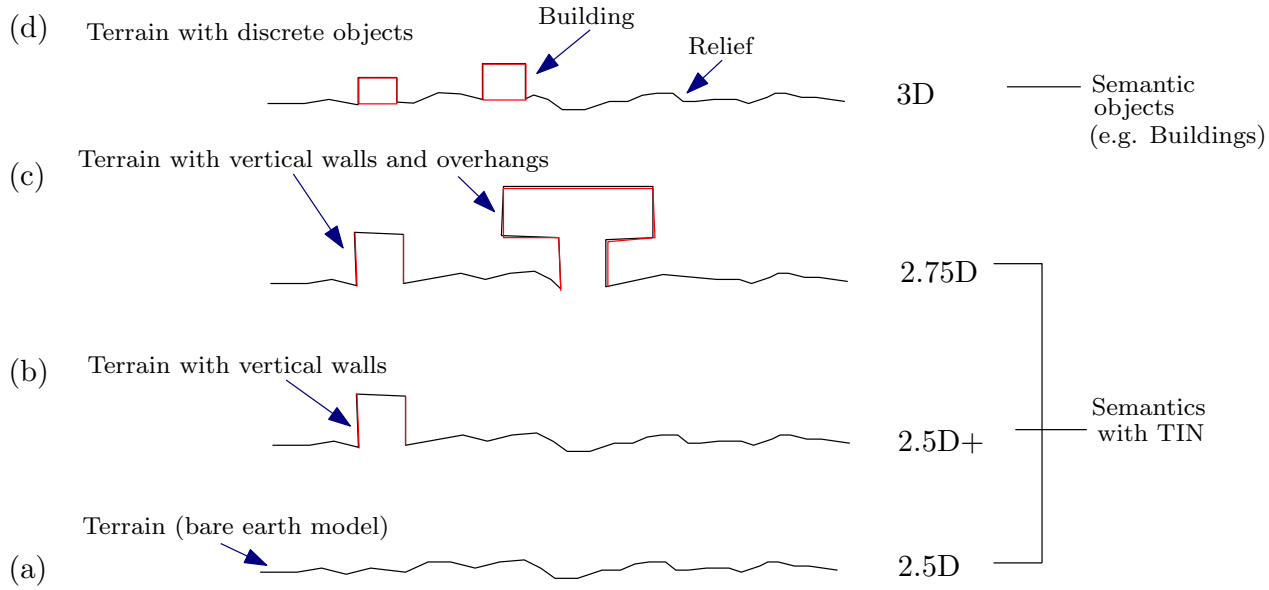


Figure 3: Different TIN representations for modelling terrains considered in this research. Semantics are attached to the entire TIN in 2.5D/2.5D+/2.75D and to the discrete objects (e.g. Buildings) embedded in the TIN in 3D

(Figure 3(d)). The boundary surfaces of the solid can be modelled using TINs (triangles) or polygons.

The above discussed surface representations provide the geometrical model of a terrain and do not include explicit representation of individual terrain features (can be natural or man made) such as landuse, buildings, roads, and water bodies. A representation of terrain features is required to support semantic queries about these features. To identify these individual terrain features one must define them as discrete objects and provide their characteristics and relations to other features explicitly through semantics. In an object perspective, a terrain can be viewed as a container populated by these objects, each with identity, spatial embedding, and attributes (Cova and Goodchild, 2002). We see here that conceptually, field and object based models are not mutually exclusive in case of terrains. Therefore, we describe a terrain as a: *continuous surface with elevation value(s) (can be more than one in case of 2.75D) for every location within its spatial domain and these locations are mapped to individual terrain objects, each with its own semantic model of information.*

2.2 TIN representations

The simplest way of representing a TIN is to store each of its triangles as a list of vertex coordinates. *Simple Feature* (Open Geospatial Consortium, 2011) is an example of such data structure. It stores each triangle as a closed linear ring of its vertex coordinates (Figure 4) (Kumar et al., 2016b). It is simple to store and represent and is supported by CityGML (GML) and almost all other spatial databases. The ISO 19136:2007 implementation standard GML uses Simple Feature structure for storing object geometry (ISO, 2007). However, it has certain limitations; first, the structure exhibits data redundancy. In Simple Feature structure, the first vertex of every ring is repeated as the last vertex of the linear ring (Figure 4). Given the vertices follow the poisson distribution, the average degree of a vertex in a 2D Delaunay triangulation is exactly 6 (Okabe et al., 2009). This suggests that on average each vertex is stored $6+(6/3) = 8$ times in the Simple Feature structure (Kumar et al., 2016b). The size of the dataset increases considerably with this repeated storage of

vertex information for every triangle. Secondly, it has very limited topology and does not explicitly store the adjacency relationships between the triangles which are necessary for traversing the TIN.

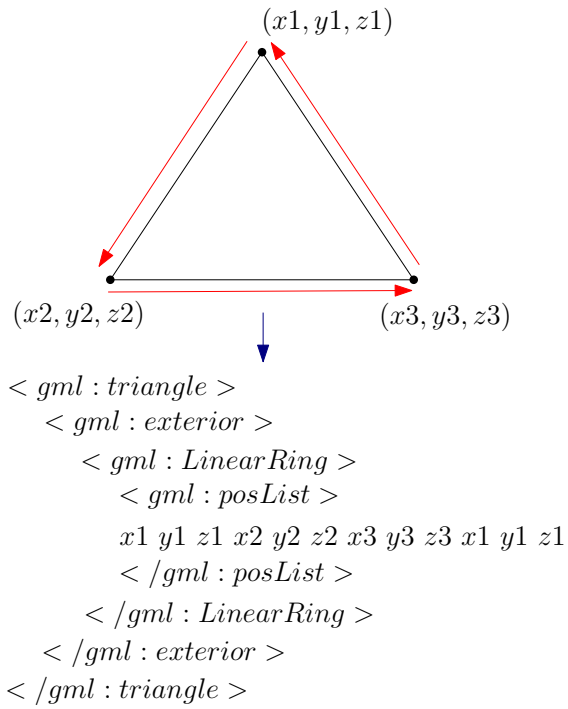


Figure 4: Simple Feature representation for a triangle in ISO 19136:2007 implementation standard GML (Kumar et al., 2016a). The first vertex $(x1, y1, z1)$ of every triangle is repeated as the last vertex $(x1, y1, z1)$ to close the linear ring

The need for storage efficient representations for triangular meshes has contributed to the development of a number of compact data structures which have different goals such as compression and/or explicit storage of topological relationships, e.g. *Indexed Triangles* (similar to *OBJ*), *Triangles with adjacency information(we refer it here as Triangle+)* (Boissonnat et al., 2002; Shewchuk, 1996), *Stars* (Blandford et al., 2005; Ledoux, 2015) *TriStrips*(Speckmann and Snoeyink, 2001), *Half-edge or DCEL* (Mäntylä, 1987; Muller and Preparata, 1978), *SQuad* (Gurung et al., 2011a), *Grouper* (Luffel et al., 2014), *Laced Ring* (Gurung et al., 2011b), *Zipper* (Gurung et al., 2013), and *Tripod* (Snoeyink and Speckmann, 1999).

The TIN data structures that we consider in this research are *Indexed Triangles*, *Stars*, and *TriStrips*. The other data structures are also capable of reducing the storage requirements for a TIN and ensuring an efficient implementation with respect to run-time and mesh operations. They can be useful for streaming and visualization of large TINs. CityGML, on the other hand, is a XML based data model for storing and representing 3D city objects. Visualization of data is not the main task of CityGML. Storing data in XML format with highly compressed data structures would require more preprocessing and later on extensive decoding for comprehensibility. Therefore, we only consider simple solutions that fit in the CityGML (GML) model and still assure interoperability. We present in the following text the details of the data structures, and we use them for tests in Section 4.

Indexed Triangle It stores every triangle of the TIN as references to the IDs of the three vertices forming the triangle (Kumar et al., 2016b). The vertices are stored in a separate

list with IDs and are not repeated for every triangle like in Simple Feature. For instance, in Figure 5, a triangle T has three vertices with IDs $\{v_1, v_2, v_3\}$ each with a tuple of location coordinates (x, y, z). With Indexed Triangle, the triangle T and its vertices are represented as shown below:

```
# list of vertices
v1 = (x1,y1,z1)
v2 = (x2,y2,z2)
v3 = (x3,y3,z3)
# list of triangles
T = {v1,v2,v3}
```

3D data formats like OBJ, and ITF (Intermediate TIN Format) (VTP, 2012) use this data structure for storing triangles. The information about the adjacency and incidence relationships between the triangles of a TIN can be easily derived using this data structure.

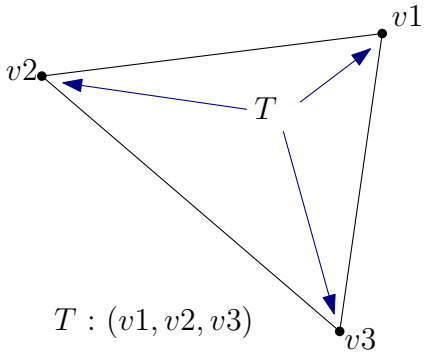


Figure 5: Indexed Triangle (Kumar et al., 2016b). Every triangle T is represented by the IDs of the three vertices (v_1, v_2, v_3) forming the triangle

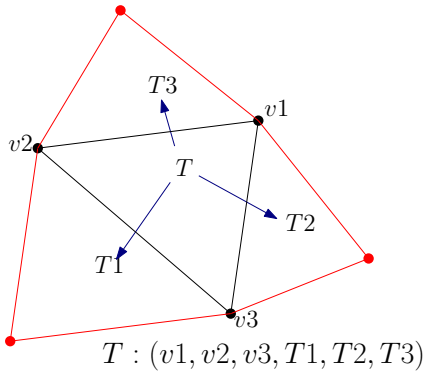
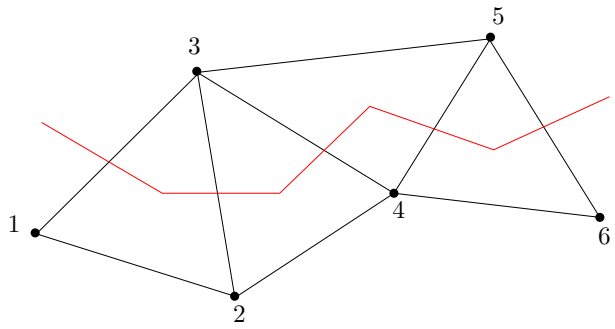


Figure 6: Triangle+ (Kumar et al., 2016b). Every triangle T is represented by the IDs of the three vertices (v_1, v_2, v_3) forming the triangle and its three adjacent triangles $\{T1, T2, T3\}$

Another variation of Indexed Triangle structure is *Triangle+*, which stores triangles along with their adjacency information. CGAL (Computational Geometry Algorithms Library) 2D triangulations (Boissonnat et al., 2002), and Shewchuk's Triangle (Shewchuk, 1996) use this data structure. The vertex coordinates (x, y, z) are stored in a separate list with their IDs. Apart from storing references to the three bounding vertices $\{v_1, v_2, v_3\}$, it also stores references to the three adjacent triangles $\{T1, T2, T3\}$ for storing the topology (Figure 6). However, the storage requirements are increased with the presence of adjacency relationships.

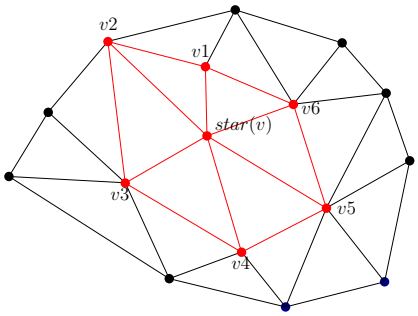
TriStrip A TriStrip or a triangle strip is a sequence of $n+2$ vertices that represents n triangles of a triangulation (Figure 7) (Speckmann and Snoeyink, 2001). TriStrips are based on the same concept like Indexed Triangles but are potentially capable of reducing the storage by a factor of 3 (Speckmann and Snoeyink, 2001). The vertex coordinates (x, y, z) are stored in a separate list with their IDs. To generate a TriStrip, we start with the three vertices of a triangle, then add a new vertex, and drop the oldest vertex to form the next triangle in sequence (Speckmann and Snoeyink, 2001). For instance, in Figure 7, the TriStrip

$(1, 2, 3, 4, 5, 6)$ represents 4 triangles: $\Delta 123$ (formed by the first three vertices, $\Delta 234$ (formed by dropping the first vertex and taking up the next vertex in sequence), $\Delta 345$, and $\Delta 456$. OpenGL, and 3D data standards like COLLADA support TriStrips for representing the geometry of objects.



$TriStrip : (1, 2, 3, 4, 5, 6)$

Figure 7: TriStrip (Speckmann and Snoeyink, 2001). The first triangle ($\Delta 123$) is formed by the first three vertices and the next triangle ($\Delta 234$) is formed by dropping the first vertex and taking up the next vertex in sequence



$Star(v) : (v1, v2, v3, v4, v5, v6)$

Figure 8: Star (Kumar et al., 2016b). Every triangle incident to the vertex v is represented by v and the two consecutive vertices in the list v_i e.g. Δvv_1v_2 is given by $\{v, v_1, v_2\}$

Star It is a vertex based, compressed, and pointerless data structure for compactly representing the triangular meshes (Blandford et al., 2005). The star of a vertex is represented as an ordered list (counter-clockwise) of IDs of the vertices incident on it (Ledoux, 2015) e.g. in Figure 8, star of vertex v, $star(v)$, is represented by the vertex list $\{v_1, v_2, v_3, v_4, v_5, v_6\}$. The vertex coordinates (x, y, z) are stored in a separate list with their IDs. The triangles are not stored explicitly but computed on-the-fly. Every triangle incident to the vertex v is represented by v and the two consecutive vertices in the list v_i e.g. Δvv_1v_2 is given by $\{v, v_1, v_2\}$.

2.3 Storing terrains in CityGML and associated problems

The data model of CityGML consists of a core module and several thematic modules for describing urban features such as `Building, Relief, LandUse, Transportation, Vegetation, and WaterBody` (Open Geospatial Consortium, 2012). In CityGML, terrains are defined within the thematic module `Relief` and represented by the class `ReliefFeature` in LOD 0-4 (Open Geospatial Consortium, 2012). With `ReliefFeature`, a terrain can be represented either as a TIN `(TINReflief)`, or as a grid `(RasterRelief)`, or as mass points `(MasspointRelief)`, or as break lines `(BreaklineRelief)`. It is also possible to represent a terrain as a combination of different terrain types in one CityGML dataset e.g. as a TIN with break lines or as a coarse grid with some areas as TINs.

The CityGML class that we are interested in is `TINReflief`. It represents terrains as TINs using either `gml:TriangulatedSurface` or `gml:Tin` (GML3 geometry types). With `gml:TriangulatedSurface`, the triangles of a TIN are explicitly specified with Simple Feature geometry (`gml:Triangle`), whereas in `gml:Tin` a list of 3D control points is specified along with the triangles. The support for triangles (`gml:Triangle`) in GML3 as Linear

Rings is in accordance with the ISO 19107:2003 Spatial schema and OGC Simple Feature Common Architecture (Open Geospatial Consortium, 2011). However, the shortcomings of Simple Feature structure (described in Section 2.2) are clearly visible in the CityGML implementation when working with massive terrain datasets.

With advancements in 3D data acquisition and processing technologies, it is now possible to generate billions of 3D points for even an area of few square kilometres and therefore, the TIN generated from these points is also massive in size. Based on literature review and experiments conducted, we found that there are several problems in storing these massive TINs with CityGML (Kumar et al., 2016b). First, CityGML datasets becomes very large with the repeated storage of vertex information in Simple Feature data structure. Secondly, there is very little topological information stored with Simple Feature. Each triangle is stored individually regardless of its neighbours, which hinders spatial analysis greatly. Third, there is no referencing scheme for the vertices of a triangle in Simple Feature structure. Each of the triangles is specified with repetition of full vertex coordinate values which takes a lot of storage space (Figure 4) (Kumar et al., 2016b). This is one of the main reasons for increased size of CityGML datasets.

Another problem concerns the representation of vertical triangles in a TIN model. CityGML is implemented as an application schema of GML3 (Open Geospatial Consortium, 2012). The `gml:Tin` is based on ISO 19107:2003 specification of `GM_TIN` which in theory is a *2.5D+* structure and can have vertical triangles. However, there is no procedure in CityGML/GML to explicitly handle these vertical triangles.

3 Modelling a CityGML extension for massive TINs

3.1 CityGML extension modelling

Depending upon the application requirements, users may want to model objects and attributes of 3D city models which are not covered in the data model of CityGML. For instance, CityGML does not contain explicit thematic models for embankments, excavations and city walls (Open Geospatial Consortium, 2012). One solution can be to model these objects using the CityGML module *Generics*. Generics is a semi-structured extension mechanism where the city objects are extended with additional objects and attributes without making any changes in the CityGML schema. But using Generics has certain limitations. CityGML datasets with generic objects and/or attributes cannot be validated against the schema because their names and data types are not formally defined in the schema. Moreover, name conflicts of the generic attributes and objects may occur. Consequently, using Generics has very limited semantic and syntactic interoperability.

The second approach that CityGML uses to specify extensions to the model is ADE (Application Domain Extension). While Generics are created at runtime without introducing any changes in the CityGML schema, an ADE is formally specified in a separate XSD (XML Schema Definition) file and has its own namespace (Open Geospatial Consortium, 2012). ADEs are actively used by information communities to create application specific extensions such as the Energy ADE for energy modelling (Nouvel et al., 2015), the GeoBIM ADE for BIM-IFC integration with CityGML (de Laat and Van Berlo, 2011), the IMGeo ADE for modelling Dutch topographic data in CityGML (Brink et al., 2013), and the Noise ADE for noise modelling (Open Geospatial Consortium, 2012). The advantage of using ADEs is that the extensions are formally specified which ensures semantic and syntactic interoperability for the exchange of application specific information. The extended CityGML instances can be validated. Additionally, it is possible to use more than one ADE in the same dataset.

After comparing the the two alternatives, we adopted the ADE approach for modelling an extension to CityGML. ADEs can be modelled in two ways; first, directly in the XSD

schema file. Secondly, it can be modelled by extending the UML (Unified Modelling Language) model of CityGML with application specific attributes/objects and later generating the XML schema from the UML model. Brink et al. (2013) describes six alternatives for modelling ADEs in CityGML. One approach is to add new application specific attributes directly in the existing CityGML classes. However, this implies editing the standard CityGML schema, which is controlled by a different authority: OGC (Open Geospatial Consortium). Alternatively, we can use ADE hooks; every CityGML feature type has a "hook" `_GenericApplicationPropertyOf<Featuretypename>` in its in its XML schema definition which allows to attach an arbitrary number of additional attributes to it in the ADEs. Another approach is to add new attributes or objects in subclasses in an ADE package. Since we are modelling an extension to CityGML, defining the new classes as subclasses of existing CityGML classes and adding the new attributes to these subclasses seems appropriate. Therefore, we prefer to adopt this approach for modelling the ADE. The method of inheritance with classes and subclasses is easy to understand with some basic knowledge of UML. This approach was also accepted as best practice by OGC (Open Geospatial Consortium, 2014).

3.2 Modelling choices for new TIN geometry types in GML

CityGML features are spatially represented by the GML3 geometry model. The geometry model of GML3 is based on the ISO 19107:2003 'Spatial Schema' (ISO, 2003). It consists of geometric primitives such as points, lines and polygons, which are be combined to form complexes, aggregates, or composite geometries. Therefore, we introduce the new geometry types in GML3 geometry model (see Figure 9) and extend them to CityGML feature types as an ADE.

To avoid any name conflict with the existing GML elements, the new schema elements are defined in a separate XSD file `"iTIN_GML.xsd"` with a different namespace `"http://godzilla.bk.tudelft.nl/schemas/iTIN_GML"` and the `igml` identifier. We introduce new geometry types (primitives, aggregates and composites) in this model for compactly representing TINs (see Table 1). New abstract classes for representing these geometry types are added so as not to disturb the original hierarchy of the GML3 model.

`igml:_iPointPrimitive`. An *_iPointPrimitive* is an abstract class for modelling the point geometries. It is modelled as a type of `gml:_GeometricPrimitive`.

`igml:iPoint`. An *iPoint* (or an *indexed Point*) represents the geometry of an individual point (or vertex). It is modelled as a type of `igml:_iPointPrimitive`. Each iPoint has an integer ID and a list of its coordinates (x, y, z) given by `<igml:id>` and `<igml:coordinates>`, respectively. An `igml:iPoint` representation for a point is given below:

```
<igml:iPoint>
  <igml:id>1234</igml:id>
  <igml:coordinates>
  85027.492 447446.125 1.51
  </igml:coordinates>
</igml:iPoint>
```

`igml:iPointList`. An *iPointList* (or an *indexed Point List*) is a list of all the points (or vertices) of a surface defined by space separated values of all the coordinates. It is modelled as a type of `igml:_iPointPrimitive`.

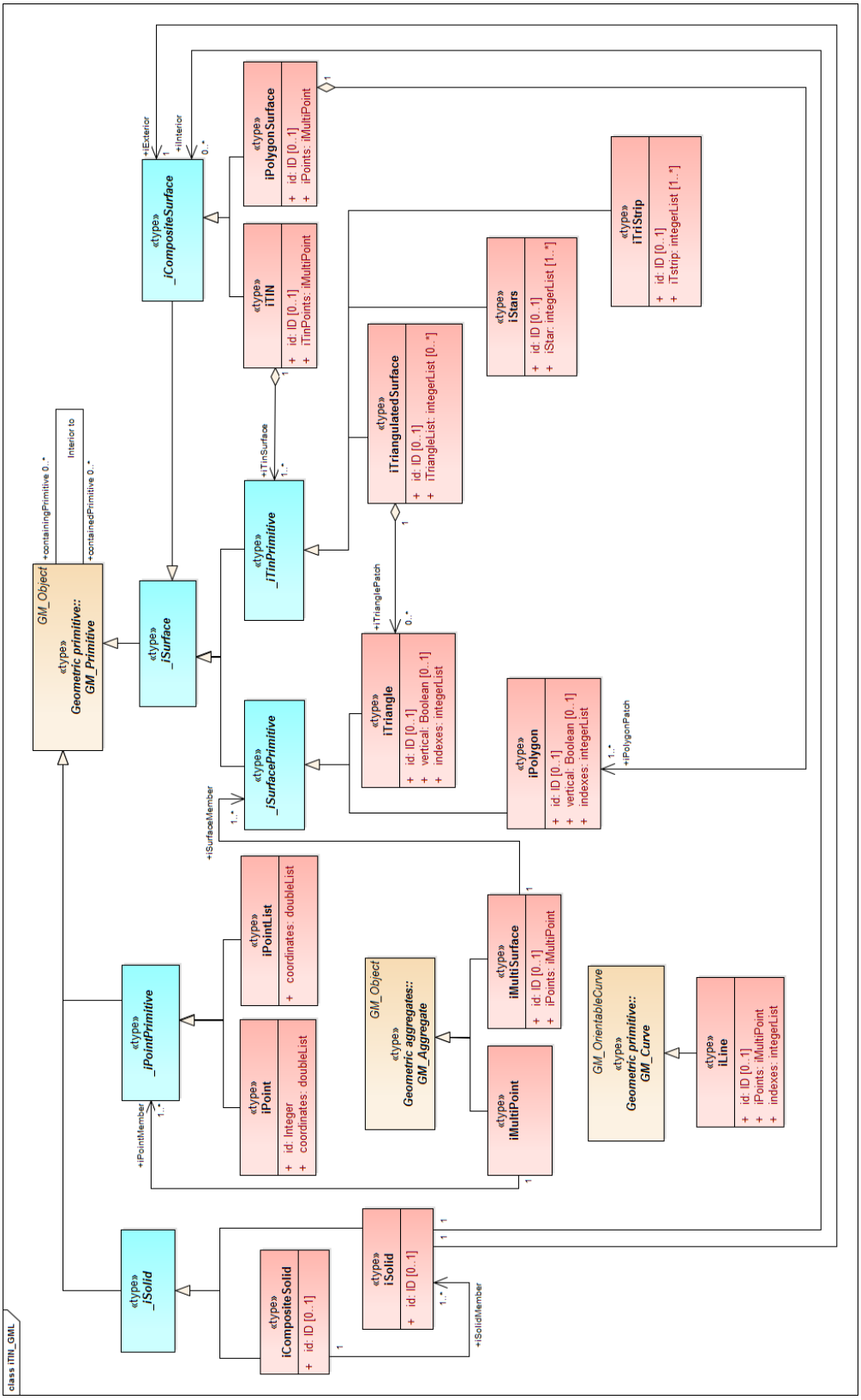


Figure 9: Our proposed geometry types in GML3 geometric model for massive TINs (proposed abstract classes are shown in blue and implementation classes are shown in red)

#	iTIN_GML	Base class	
		GML	iTIN_GML
1.	_*iPointProimitive*	_*GeometricPrimitive*	
2.	iPoint		_*iPointProimitive*
3.	iPointList		_*iPointProimitive*
4.	iMultiPoint	_*GeometricAggregate*	
5.	iMultiSurface	_*GeometricAggregate*	
6.	iLine	_*Curve*	
7.	_*iSurface*	_*Surface*	
8.	_*iSurfacePrimitive*		_*iSurface*
11.	iTriangle		_*iSurfacePrimitive*
12.	iPolygon		_*iSurfacePrimitive*
9.	_*iTinPrimitive*		_*iSurface*
13.	iTriangulatedSurface		_*iTinPrimitive*
14.	iTriStrip		_*iTinPrimitive*
15.	iStars		_*iTinPrimitive*
10.	_*iCompositeSurface*		_*iSurface*
16.	iTIN		_*iCompositeSurface*
17.	iPolygonSurface		_*iCompositeSurface*
18.	_*iSolid*	_*GeometricPrimitive*	
19.	iSolid		_*iSolid*
19.	iCompositeSolid		_*iSolid*

Table 1: Proposed iTIN_GML geometry elements. Prefix *"i"* signifies that everything is indexed and refers to the extension we proposed to the model. Prefix "_" indicates an abstract class in the model.

```
<igml:iPointList>
  <igml:coordinates>
    85027.492 447446.125 1.51
    85027.289 447446.156 1.31
    85049.219 447448.312 1.37
    85068.219 447447.332 1.64
    ....
  <igml:coordinates>
</igml:iPointList>
```

`igml:iMultiPoint`. To represent all the points of a surface, we added a new class `igml:iMultiPoint`. An *iMultiPoint* is a collection of all the points (i.e. vertices) of a surface and is a type of `gml:_GeometricAggregate`. With `igml:iMultiPoint`, it is possible to store points either as a collection of individual `igml:iPoint(s)` referenced through `igml:iPointMember` elements or as a `igml:iPointList` (see snippet below).

```
<igml:iMultiPoint>
  <igml:iPointMember>
    <igml:iPoint>
      ....
    </igml:iPoint>
  <igml:iPointMember>
```

```
<igml:iMultiPoint>
  <igml:iPointMember>
    <igml:iPointList>
      ....
    </igml:iPointList>
  </igml:iPointMember>
```

```
  ....                                    </igml:iMultiPoint>
</igml:iMultiPoint>
```

`igml:iLine`. An *iLine* (or an *indexed Line*) represents the geometry of an individual line segment (or curve). It is modelled as a type of `gml:_Curve` which is subtype of `gml:_GeometricPrimitive`. We did not introduce any separate abstract base class (such as *_iLine*) because it is a complete geometry (with points and indexes) and hence, can be reused with `gml:MultiCurve`. The existing hierarchy of elements in the GML model is followed for defining new classes in the model. Each iLine has an ID given by `<igml:id>` and a list of IDs of the points forming the line given by `<igml:indexes>`. The `<igml:indexes>` lists the IDs of the points comprising the geometry instead of repeating the coordinate values of the points again. An `igml:iLine` representation for a line connecting two point (with given point IDs) is defined as:

```
<igml:iLine>
  <igml:id>D23</igml:id>
  <igml:iPoints>
  ....
  </igml:iPoints>
  <igml:indexes>1 2</igml:indexes>
</igml:iLine>
```

`igml:_iSurface`. For modelling the surfaces, we introduced another abstract class `igml:_iSurface` as a type of `gml:_GeometricPrimitive`. It has 3 subclasses: `igml:_iSurfacePrimitive` for modelling individual surface elements (polygon and triangle), `igml:_iTinPrimitive` for modelling TIN representations, and `igml:_iCompositeSurface` for modelling TINs and composite polygonal surfaces.

`igml:iTriangle`. An *iTriangle* (or an *indexed Triangle*) represents the geometry of an individual triangle. It is modelled as a type of `igml:_iSurfacePrimitive`. An `igml:iTriangle` is specified by the references to IDs of the 3 vertices of the triangle given by `gml:iPoint`. It has an optional element `igml:vertical` to specify if the triangle is a vertical triangle. For some applications such as flow modelling, adjacency and network analysis, it is sufficient to use a city model and its buildings as a single triangulated surface containing vertical triangles instead of using a volumetric model (Gorte and Lesparre, 2012). The `<igml:vertical>` element helps us to identify these vertical surfaces modelled in the terrain without relying on the geometry and on-the-fly computation (which are prone to precision errors). This means that the model is more than 2.5D, but is less than 3D; the geometry is 3D, but the underlying topology remains 2D.

```
<igml:iTriangle>
  <igml:id>34</igml:id>
  <igml:vertical>false</igml:vertical>
  <igml:indexes>1 2 3</igml:indexes>
</igml:iTriangle>
```

`igml:iPolygon`. An *iPolygon* (or an *indexed Polygon*) represents the geometry of an individual polygon. It is also modelled as a type of `igml:_iSurfacePrimitive` and has the same geometrical representation as `igml:iTriangle`. An `igml:iPolygon` is specified by

the references to IDs of the vertices (>3) of the polygon. It also has an optional element `igml:vertical` to specify if the polygon is a vertical surface.

```
<igml:iPolygon>
  <igml:id>14</igml:id>
  <igml:vertical>true</igml:vertical>
  <igml:indexes>3 4 5 6</igml:indexes>
</igml:iPolygon>
```

`igml:iMultiSurface`. An *iMultiSurface* is a collection of surfaces (triangles/polygons) which can be disjoint, overlapping, touching, or even disconnected. It is modelled as a type of `gml:_GeometricAggregate`. We did not introduce any separate abstract base class (such as *_iGeometricAggregate*) because it is a complete geometry (with points and indexes) and hence, can be reused in other geometry types. With `igml:iMultiSurface`, it is possible to store surface either as a collection of triangles (`igml:iTriangle`) or as a collection of polygons (`igml:iPolygon`) referenced through `igml:iSurfaceMember` elements(see snippet below).

```
<igml:iMultiSurface>
  <igml:id>f24</igml:id>
  <igml:iSurfaceMember>
   <igml:iTriangle>
    ....
   </igml:iTriangle>
  </igml:iSurfaceMember>
  ....
</igml:iMultiSurface>
```

```
<igml:iMultiSurface>
   <igml:id>f24</igml:id>
  <igml:iSurfaceMember>
    <igml:iPolygon>
     ....
    </igml:iPolygon>
   </igml:iSurfaceMember>
  ....
</igml:iMultiSurface>
```

`igml:_iCompositeSurface`. To model disjoint, non-overlapping, topologically connected surfaces, we introduced an abstract base class `igml:_iCompositeSurface`. It has two subclasses `igml:iTIN` and `igml:iPolygonSurface`.

`igml:iTIN`. For representing TINs, we added a new class `igml:iTIN` as a subclass of `igml:_iCompositeSurface` (and not aggregates) because TINs represent surfaces with disjoint, non-overlapping and topologically connected triangles. Apart from the above described geometric primitives and aggregates, we added three new TIN representation types: `igml:iTriangulatedSurface`, `igml:iStars` and `igml:iTriStrips` as subclasses of `igml:_TinPrimitives`. In `igml:iTIN`, the TIN vertices are represented using `igml:iMultiPoint` and the TIN surface can be represented using any of these three new surface types.

```
<igml:iTIN>
  <igml:id>A24</igml:id>
  <igml:iTinPoints>
    <igml:iMultiPoint>
     ....
    </igml:iMultiPoint>
  </igml:iTinPoints>
  <igml:iTinSurface>
   .....
  <igml:iTinSurface>
</gml:iTIN>
```

`igml:iPolygonSurface`. For representing topologically connected polygon surfaces, we added a new class `igml:iPolygonSurface` as a subclass of `igml:_iCompositeSurface`. Points are represented using `igml:iMultiPoint` and the polygons are represented using `igml:iPolygon` geometry referenced through `igml:iPolygonPatch` elements.

```
<igml:iPolygonSurface>
  <igml:id>A22</igml:id>
  <igml:iPoints>
     ....
  </igml:iPoints>
  <igml:iPolygonPatch>
    <igml:iPolygon>
      .....
    </igml:Polygon>
  <igml:iPolygonPatch>
</gml:iPolygonSurface>
```

`igml:iTriangulatedSurface`. An *iTriangulatedSurface* stores triangles either as a collection of individual `igml:iTriangle` referenced through `igml:iTrianglePatch` elements or as `igml:iTriangleList` (see snippet below). An `igml:iTriangleList` is a space separated list of IDs of the vertices of all the triangles.

```
<igml:iTriangulatedSurface>
  <igml:id>A24</igml:id>
  <igml:iTrianglePatch>
   <igml:iTriangle>
    ....
   </igml:iTriangle>
  </igml:iTrianglePatch>
  ....
</igml:iTriangulatedSurface>
```

```
<igml:iTriangulatedSurface>
  <igml:id>A24</igml:id>
  <igml:iTriangleList>
    1 2 3 2 3 4 3 4 5 ..
    ....
  </igml:iTriangleList>
  ....
</igml:iTriangulatedSurface>
```

`igml:iTriStrip`. An *iTriStrip* is collection of triangles represented by `igml:iTStrip` elements. In each *iTstrip*, the first triangle is formed from first, second, and third vertex. Each subsequent triangle is formed from the next vertex in sequence, reusing the previous two vertices. Each `igml:iTriStrip` can have any number of `igml:iTstrip` elements to depict local connectivity.

```
<igml:iTriStrip>
  <igml:id>B54</igml:id>
  <igml:iTstrip id = "1"> 1 2 3 4 5 </igml:iTstrip>
  <igml:iTstrip id = "2"> 11 12 13 14 </igml:iTstrip>
  ....
</igml:iTriStrip>
```

`igml:iStars`. An *iStars* is a collection of `igml:iStar` elements defined for every vertex of a triangulated surface. For every vertex, an *iStar* stores an ordered list of IDs of the

vertices incident to it (see snippet below). Every triangle incident to a vertex is represented by the ID of that vertex and the IDs of two consecutive vertices in the list.

```
<igml:iStars>
  <igml:id>A34</igml:id>
  <igml:iStar id = "1">2 3 4 5 6 7</igml:iStar>
  <igml:iStar id = "2">3 4 7 8 9 11</igml:iStar>
  ....
</igml:iStars>
```

`igml:_iSolid`. For representing solids, we introduced another abstract class `igml:_iSolid`. It is modelled as a type of `gml:_GeometricPrimitive`.

`igml:iSolid`. It is modelled as a type of `igml:_iSolid` with exterior and interior of the solid modelled as a composite surface `igml:_iCompositeSurface`. The exterior shell and interior of the solid can be modelled either as a TIN (`igml:iTIN`) or as polygonal surface (`igml:iPolygonSurface`) referenced through `igml:iExterior` and `igml:iInterior` elements.

```
<igml:iSolid>
  <igml:id>A234</igml:id>
  <igml:iExterior>
    <igml:iTIN>
      .....
    </igml:iTIN>
  </igml:iExterior>
  <igml:iInterior>
    <igml:gml:iPolygonSurface>
      .....
    </igml:gml:iPolygonSurface>
  </igml:iInterior>
</igml:iSolid>
```

`igml:iCompositeSolid`. It is modelled as a type of `igml:_iSolid`. It is a collection of solids (igml:iSolid) referenced through `igml:iSolidMmeber` elements.

```
<igml:iCompositeSolid>
  <igml:id>A1234</igml:id>
  <igml:iSolidMember>
    <igml:iSolid>
      .....
    </igml:iSolid>
  </igml:iSolidMember>
  .....
</igml:iCompositeSolid>
```

3.3 Extending CityGML for massive terrains

For modelling terrains as TINs, the `iTIN_GML` elements are added to CityGML using an ADE. The initial idea was to integrate these TIN representations directly in the GML model

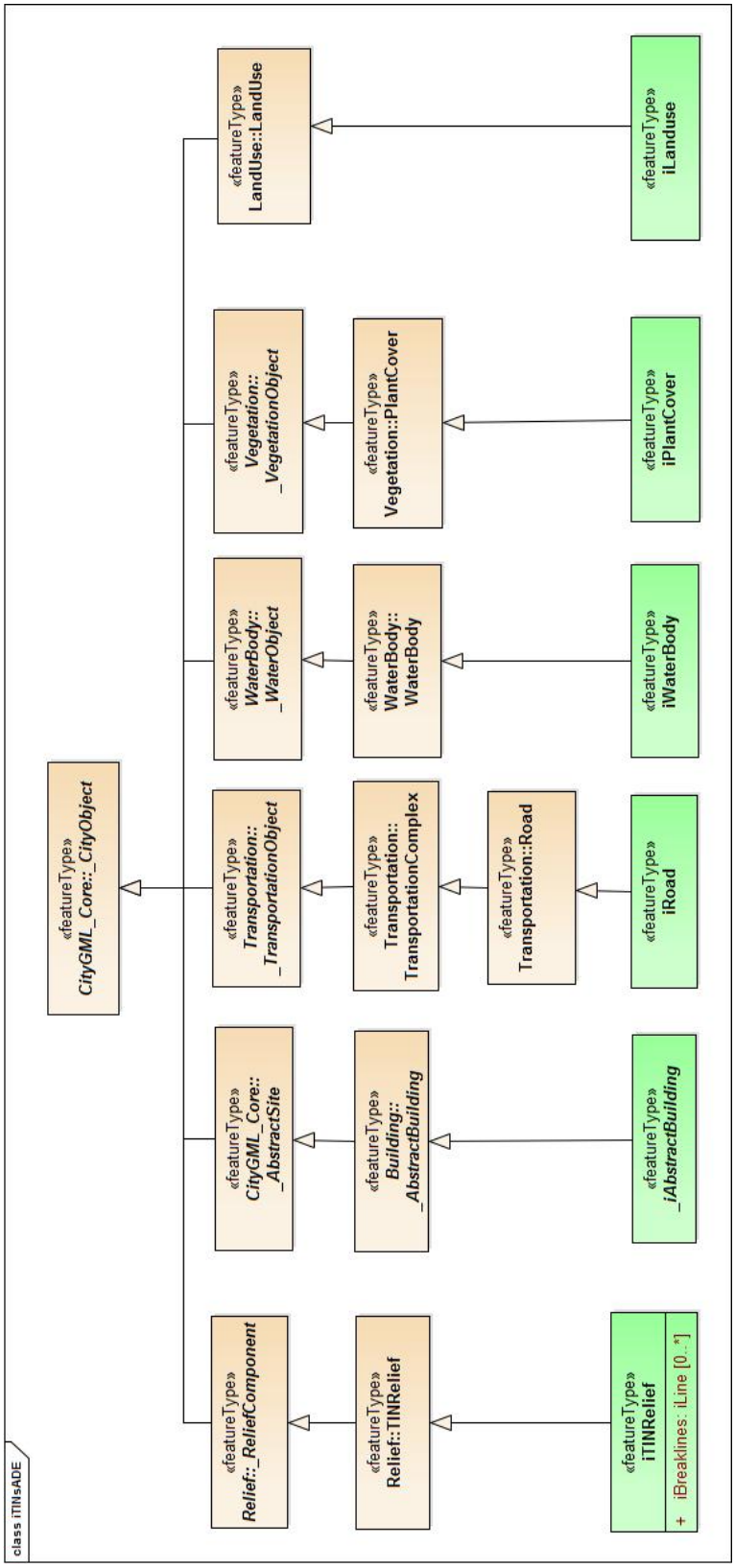


Figure 10: Proposed classes in CityGML iTINs ADE for massive terrains (ADE classes shown in green color)

so as to use the same namspace and identifier of GML. CityGML would then inherit these geometry types automatically from the enhanced GML model. This would have eliminated the need to extend the existing CityGML classes with these new geometrical representations. However, both GML and CityGML are controlled by a formal authority: OGC. It would have been unwise to change the original GML and CityGML model without the approval of OGC.

Therefore, to show the benefits of this approach, we developed it as an ADE. We created a separate package to model the new TIN geometry types and added them to CityGML by extending the existing CityGML classes in an ADE package. Moreover, these geometry types can be easily added to the original GML/CityGML model, if approved by OGC.

The new classes are modelled as subclasses of the existing CityGML classes (marked with stereotype «featureType») and can have their own properties (Table 2). The CityGML Relief module is extended to include the iTIN_GML elements for modelling terrains (see Figure 10). Similarly, we extended other CityGML modules Relief, Building, Vegetation, Transportation(Road), WaterBody, and LandUse to include the iTIN_GML elements for representing TINs. These elements can be used independently for compact geometrical representation of terrain and its features such as buildings, roads, vegetation. The ADE classes are defined in a separate file "CityGML_iTINs_ADE.xsd" with a different namespace "http://godzilla.bk.tudelft.nl/schemas/iTINs_ADE" and the itin identifier.

#	CityGML iTINs ADE	CityGML module	iTIN_GML geometry			
			iTIN	iPS	iMS	iSolid
1	iTINRelief	Relief	✓			
2	iWaterBody	WaterBody	✓	✓	✓	
3	iRoad	Transportation	✓	✓	✓	
4	iPlantCover	Vegetation	✓	✓	✓	
5	iLandUse	LandUse	✓	✓	✓	
6	_iAbstractBuilding	Building	✓	✓		✓

Table 2: New classes in the CityGML ADE (iPS = iPolygonSurface, iMS = iMultiSurface)

iTINRelief. In the CityGML Relief module, a new relief component called iTINRelief is introduced as a subclass dem:TINRelief. iTINRelief extends all the properties of the base class like name, description and LOD, etc. and has igml:iTIN geometrical representation (Figure 11). In the original dem:TINRelief class, the level of detail (LOD) is specified using dem:lod element. Here, we introduced separate geometrical representations for the relief LODs (0-4) using lod0iTIN, lod1iTIN, lod2iTIN, lod3iTIN and lod4iTIN elements. Another element called iExtent is also introduced to mark the extent of the TIN using igml:iPolygonSurface geometry. To represent the break lines in a TIN, we introduced an element called iBreaklines with geometry igml:iLine.

```
<cityObjectMember>
  <dem:ReliefFeature>
    <gml:name> Example iTINRelief </gml:name>
    <dem:lod> 1 </dem:lod>
    <dem:reliefComponent>
      <itin:iTINRelief>
        <dem:lod> 1 </dem:lod>
        <itin:iTINobject>
```

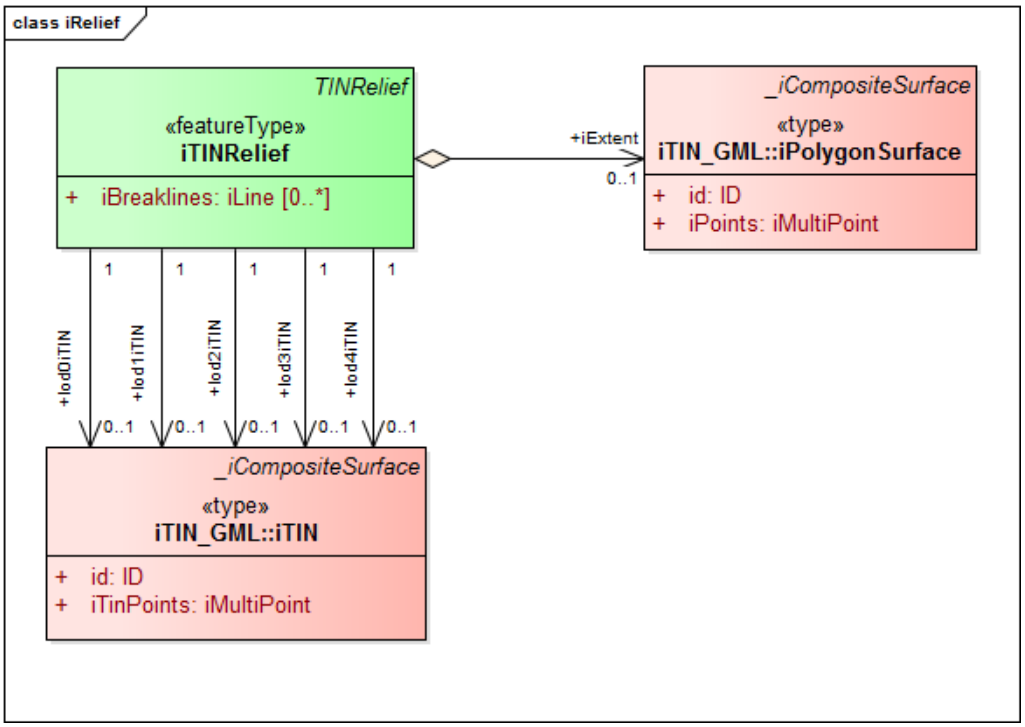


Figure 11: `iTINRelief` modelled in CityGML iTINs ADE using iTIN_GML

```
          <itin:lod1iTIN>
            <igml:iTIN>
              ...
            </igml:iTIN>
          <itin:lod1iTIN>
        </itin:iTINobject>
      </itin:iTINRelief>
    </dem:reliefComponent>
  </dem:ReliefFeature>
</cityObjectMember>
```

`iLandUse`. In the CityGML `LandUse` module, a new component called `iLandUse` is introduced as a subclass `luse:LandUse`. `iLandUse` extends all the properties of the base class like name, description and LOD, etc. It can be represented either with `igml:iTIN` or `igml:iPolygonSurface` or `igml:iMultiSurface` geometrical representations at different LODs (0-4) (Figure 12).

`iPlantCover`. In the CityGML `Vegetation` module, a new component called `iPlantCover` is introduced as a subclass `veg:PlantCover`. The `Vegetation` module has class `veg:SolitaryVegetationObject` to model single vegetation objects, and class `veg:PlantCover` to model areas filled with a specific vegetation type. Solitary vegetation objects are usually modelled with implicit geometries, therefore we added iTIN_GML representations only to `veg:PlantCover`. Vegetation can be represented with `iPlantCover` using either `igml:iTIN` or `igml:iPolygonSurface` or `igml:iMultiSurface` geometrical representations at different LODs (0-4) (Figure 13).

`iRoad`. In the CityGML `Transportation` module, a new component called `iRoads` is

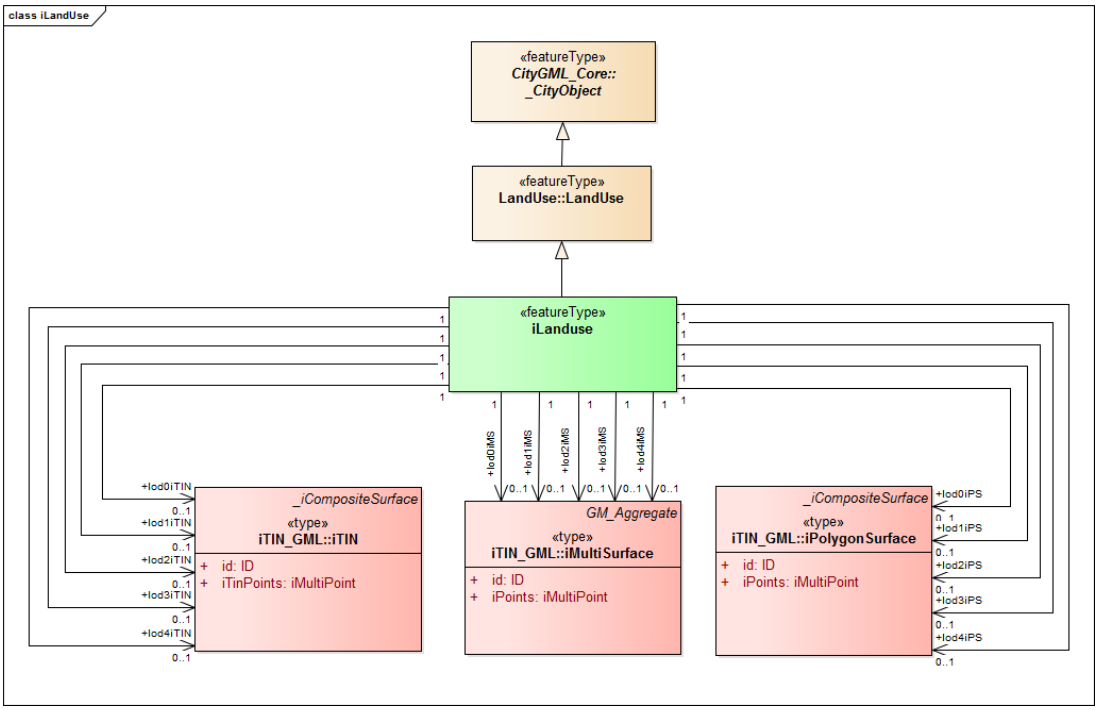


Figure 12: `iLandUse` modelled in CityGML iTINs ADE using iTIN_GML

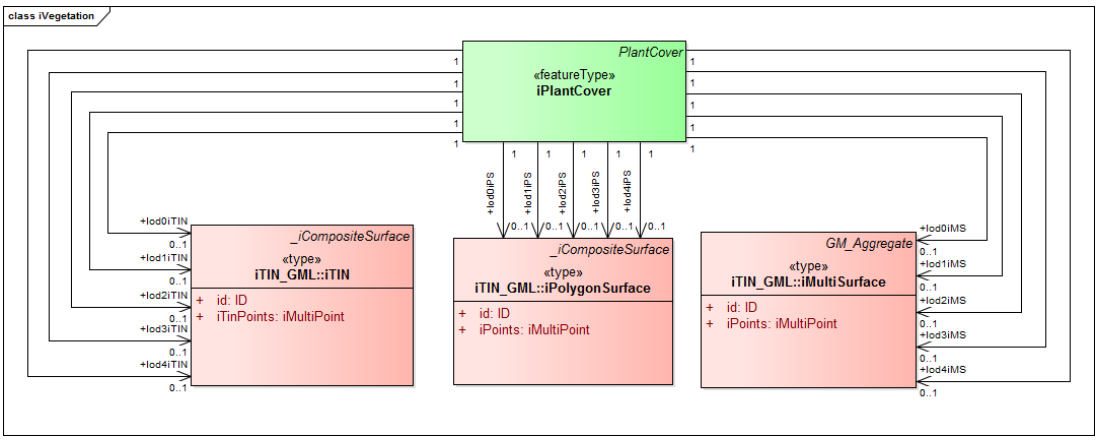


Figure 13: `iPlantCover` modelled in CityGML iTINs ADE using iTIN_GML

introduced as a subclass `tran:Roads`. The road is represented as a `tran:TransportationComplex` in CityGML with different geometrical representation at different levels of detail. At LOD-0, `iRoads` use `igml:iLine` geometry for representing roads. From LOD 2-4, `iRoads` can be represented using either `igml:iTIN` or `igml:iPolygonSurface` or `igml:iMultiSurface` geometrical representations (Figure 14). In CityGML, objects such as *Track, Road, Railway, Square* are also modelled as a type of `tran:TransportationComplex`. These objects are beyond the scope of this study and therefore, are not included in the ADE. However, we assure that these objects can be extended in a similar manner for representation.

`iWaterBody`. In the CityGML `WaterBody` module, a new component called `iWaterBody` is

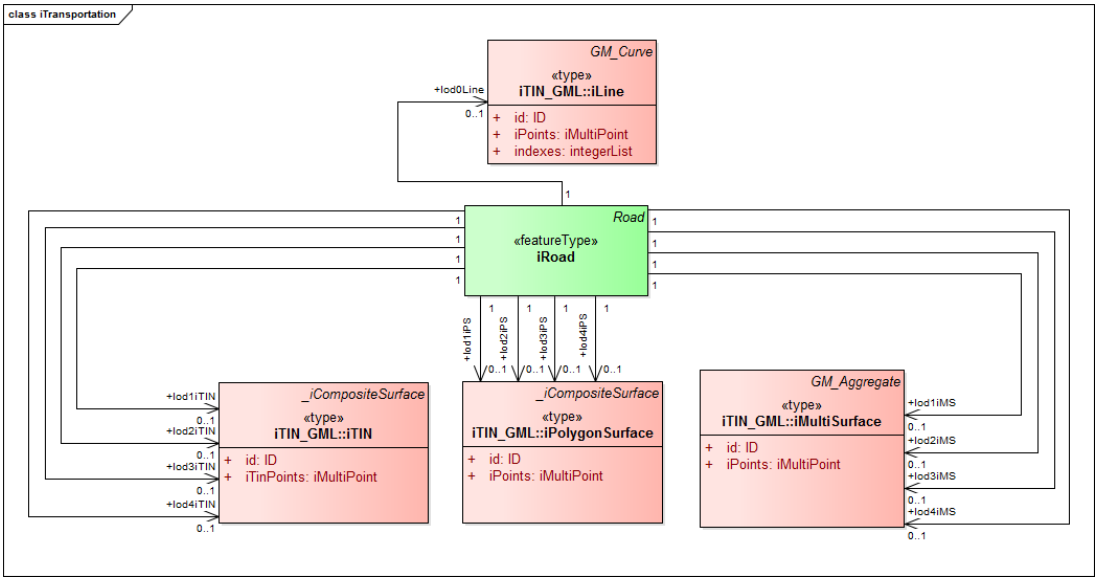


Figure 14: `iRoad` modelled in CityGML iTINs ADE using iTIN_GML

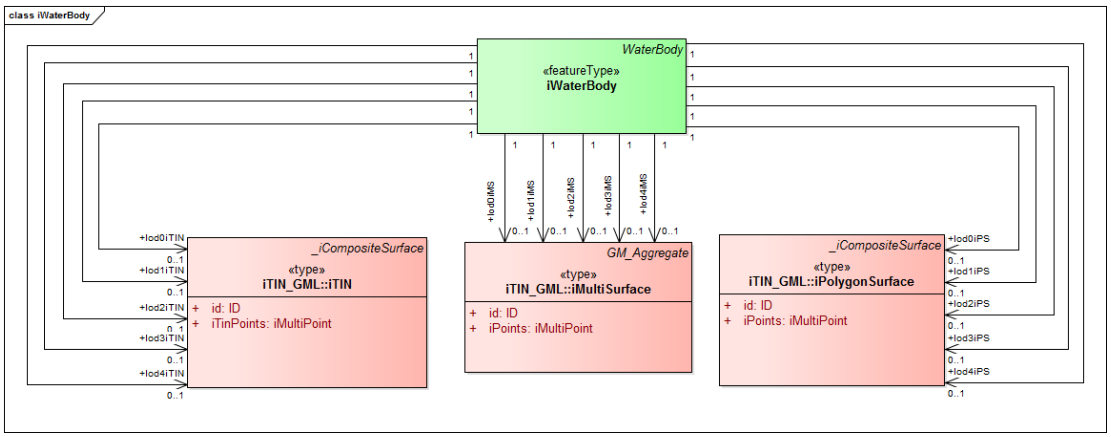


Figure 15: `iWaterBody` modelled in CityGML iTINs ADE using iTIN_GML

introduced as a subclass `wtr:WaterBody`. Water can be represented with `iWaterBody` using either `igml:iTIN` or `igml:iPolygonSurface` or `igml:iMultiSurface` geometrical representations at different LODs (0-4) (Figure 15). Theoretically, in CityGML, any WaterBody can also be represented by a solid, bounded by thematic surfaces, at LOD 2-4 (Open Geospatial Consortium, 2012). But in real world scenarios, it is usually modelled as a surface and therefore we do not take solid representation into account. However, it can be added to the ADE in the same way as surface representation.

`_iAbstractBuilding`. In the CityGML `Building` module, a new abstract class `_iAbstractBuilding` is added as a subclass of `bldg:_AbstractBuilding`. `_iAsbtractBuilding` has two subclasses: `iBuilding` and `iBuildingPart`. Buildings and building parts can be represented either with `igml:iSolid` or `igml:iTIN` or `igml:iPolygonSurface` geometric representation (Figure 16). `_iAsbtractBuilding` is modelled for LOD 0-3. Openings and boundary surfaces are also represented for modelling LOD-3. LOD-4 with building interior can be modelled in the same manner.

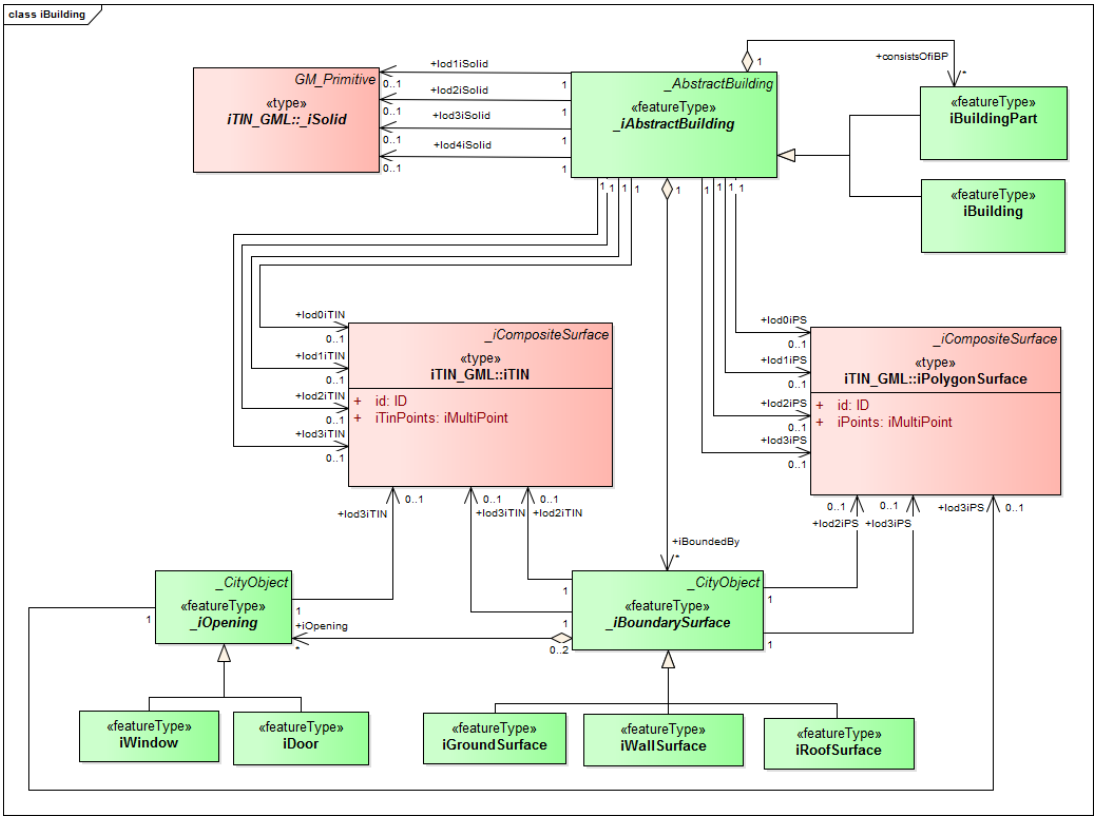


Figure 16: `_iAbstractBuilding` modelled in CityGML iTINs ADE using iTIN_GML

4 Implementation and experiments with real world datasets

4.1 iTIN_GML and CityGML iTINs ADE schema generation

We used the ShapeChange[1] tool to derive the XML Schemas of the iTIN_GML and CityGML iTINs ADE from the UML model. ShapeChange is a JAVA based tool which implements UML to GML encoding rules described in ISO 19136, ISO 10118, and ISO 19109. We only generated the XML Schema for the GML and CityGML extensions and not for the whole data models as they are already publicly available. The generated CityGML ADE schema only requires to import the existing CityGML schema and the generated iTIN_GML schema containing new geometry types for representing TINs. These dependencies are resolved by the ShapeChange during the transformation from UML packages to XML schema. The UML models and XML schemas for iTIN_GML and CityGML iTINs ADE are available here: `https://github.com/tudelft3d/CityGML_iTINs_ADE`

4.2 Prototype testing

The terrain datasets used for testing the implementation are:

1. *AHN3 TIN. AHN3(Actueel Hoogtebestand Nederland version 3)* (AHN3, 2015) is the national height model of the Netherlands, and contains billions of 3D points (more than 10 points/m^2). AHN3 Tile# 37EN/1 (size 5 km X 6.25 km) of the AHN3 point cloud is used as input for generating the TIN using Lastools (Hug et al., 2004). The dataset is generated in streaming mesh format (*.sma) (Isenburg et al., 2005a).

[1] http://shapechange.net

2. *3DBGT. 3DBGT (3D Basisregistratie Grootschalige Topografie)* is the 3D city model of the Netherlands created using the open source software *3dfier*[2]. 3DBGT is a constrained triangulation generated from AHN3 point cloud and 2D BGT (large-scale 2D topographic dataset of the Netherlands) footprints (BGT, 2016). 3dfier takes 2D topographic datasets and lifts every 2D polygon to the required height to make them 3D. This height information is obtained from the point cloud data. We used 3DBGT TIN of the Amsterdam area for testing. The dataset is available in OBJ format (*.obj).

3. *3DTOP10NL.* 3DTOP10NL is the 3D city model of Netherlands, which covers the whole country, including buildings, terrain, roads, canals, etc. in 1368 tiles. It is generated by adding the height information from AHN2 point cloud to the 2D topographic objects in TOP10NL (Elberink et al., 2013). The layer that we are interested in 3DTOP10NL dataset is the **''terreinVlak_3D_LOD0''** which contains the terrain model with more than 1 billion triangles. The dataset is available in ESRI GeoDatabase format (*.gdb).

The details of the input terrain datasets along with their size in CityGML format are given in Table 3. A prototype was created to introduce new TIN representations in CityGML datasets. The prototype reads the input datasets and maps the Simple Feature representation of triangles to index based structure of `igml:iTIN`. The resulting storage size of the prototype testing are given in Table 3 along with the achieved compression factors.

We also compared the time taken to generate data in CityGML and CityGML_iTINs-_ADE formats from original test datasets (Table 4) to observe the performance of the system in handling massive terrain data. These tests were performed on Linux Godzilla server with 40 Intel Xeon E5-2650 v3 CPUs, 128 GB RAM, 3.3 GHz base clock speed and 3.6 GHz turbo boost speed. The three test datasets are available in three different formats (OB, SMA & GDB) and the time taken to generate output data from these datasets differs significantly. From Table 4, we can see that it takes less time to generate CityGML data from 3DTOP10NL GeoDatabase. This can be attributed to the fact that both CityGML and ESRI GeoDatabase follow Simple Feature structure for representing geometry. While generating iTIN_GML geometry types from this Simple Feature structure, most of the time is consumed in cleaning the vertices (removing duplicate), generating integer IDs for the vertices and assigning these indexes to the triangles for representing geometry. However, in case of other formats like OBJ & SMA which already follow simple indexing scheme, the `igml:iTriangulatedSurface` structure is generated very quickly. For `igml:iTriStrip` and `igml:iStars`, the data generation time is a bit high as it also includes the time taken to compute the neighbouring triangles/vertices (required for TIN traversal).

We also tested for the storage size of quantized vertices (Isenburg et al., 2005b). A vertex is called quantized when we store only the difference of its coordinates from the centroid vertex (or any other vertex) and not the full vertex coordinates. The centroid vertex is the centroid of the vertices of the TIN or can also be selected randomly. We also tried storing the difference of the coordinates from the first vertex of the TIN. However, storing quantized vertices did not change the compression factors significantly. As this was not the main objective of our study, we did not test it further.

[2]https://3d.bk.tudelft.nl/opendata/3dfier/

Terrain dataset	Number of triangles	CityGML file size	iTS		iTriStrip		iStars	
			size	CF	size	CF	size	CF
3DBGT	13,688,402	4.65 GB	337.92 MB	14.32	236.54 MB	20.1	816.13 MB	5.83
Tile #37EN/1	40,788,573	13.98 GB	952.32 MB	15.13	747.52 MB	19.22	2.28 GB	6.13
3DTOP10NL	1,156,641,666	698.74 GB	46.64 GB	14.38	37.43 GB	18.67	108.33 GB	6.45

Table 3: Details of the input datasets showing the number of triangles in each terrain dataset, and the storage space required for storing each dataset in CityGML and CityGML iTINs ADE format (CF = Compression Factor, iTS = iTriangulatedSurface)

Terrain Dataset	CityGML	iTS	iTriStrip	iStar
3DBGT (obj)	25.63 min	15.68 min	63.83 min	27.21 min
Tile #37EN/1 (sma)	52.19 min	38.87 min	93.77 min	54.32 min
3DTOP10NL (gdb)	38.54 min	121.63 min	194.31 min	166.91 min

Table 4: Time taken to generate CityGML & CityGML_iTINs_ADE data from test datasets

As it can be observed from the results, the highest compression factor of 20.1 is achieved by using iTriStrip referencing scheme for storing TINs in place of Simple Feature structure. The data structures in decreasing order of storage requirements are:

$$\text{iStars} > \text{iTriangulatedsurface} > \text{iTriStrip}$$

Although, inclusion of triangle strips (iTriStrips) provides maximum reduction in storage size, it has certain topological restrictions. We used the *TriangleStripifier* module of the *PyFFI* python package to generate triangle strips for our datasets (PyFFI, 2011). *TriangleStripifier* is a python adaptation of the *NvTriStrip* library (NVIDIA, 2004) and converts triangles into a list of strips. A triangle strip enters each triangle at one edge (known as *entry-edge*) and exits that triangle on the left or the right remaining edges (known as *exit-edges*) (Speckmann and Snoeyink, 2001). The triangle strip alternates between left and right exit-edges with each successive triangle until it reaches a triangle with no forwards connections (Speckmann and Snoeyink, 2001). For the remaining triangles, the same process is repeated until all the triangles are placed in triangle strips. The process of generating triangle strips from the test datasets is depicted in Figure 18. Therefore, for a single TIN, we can have a number of disconnected triangle strips storing the mesh triangles (Figure 17). This means there is local topological connectivity within the individual triangle strips but no overall connectivity for the entire TIN. Certain operations are thus not possible in constant time such as finding the adjacent triangles of a given triangle.

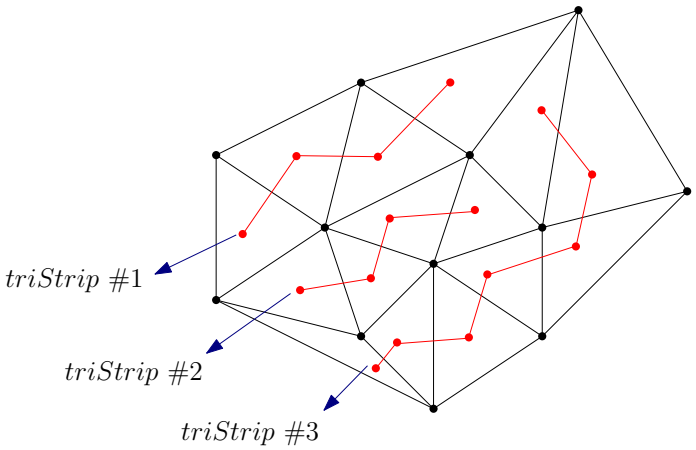


Figure 17: A single TIN can have a number of disconnected triangle strips. There is local connectivity within each strip (shown in red color) but no overall connectivity for the entire TIN

This is not the case with Stars. When all the Stars in a TIN are represented, each triangle is present in exactly 3 stars (its 3 vertices) and each edge is present in exactly 2 stars (its 2 vertices) (Ledoux, 2015). There is a significant overlap in the stars from which we can

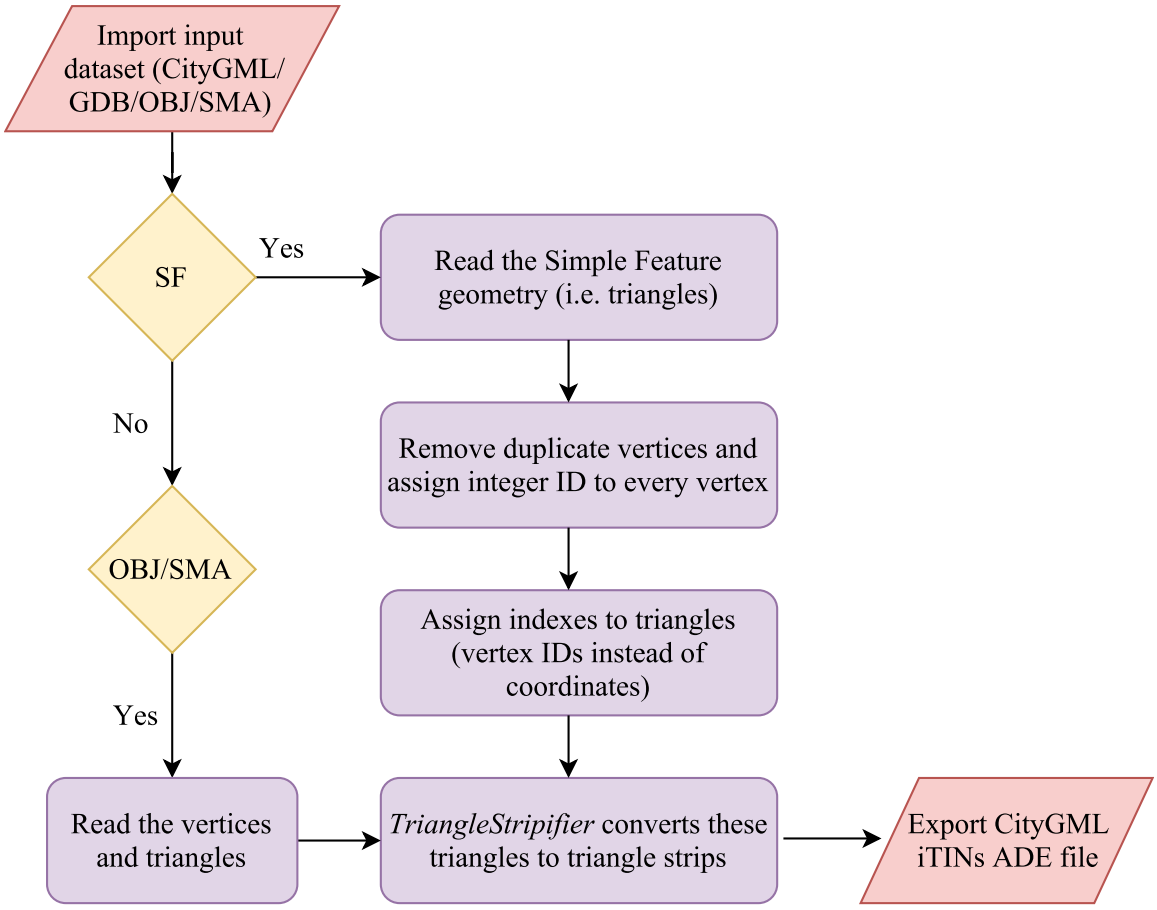


Figure 18: Flow diagram for generating triangle strips from the CityGML test datasets

derive the adjacency and incidence relationships of the triangles of a TIN (Ledoux, 2015). For a given vertex we can easily find the incident edges or triangles using stars. Therefore, these data structures in increasing order of topology can be represented as:

$$\text{iTriStrip} < \text{iTriangulatedsurface} < \text{iStars}$$

5 Conclusions & further research

The article presents a new CityGML extension for efficiently storing massive TIN terrains in CityGML. We investigated several TIN data structures for their storage requirements and topology storage, and explored how they can be implemented in CityGML for storing massive TINs. We introduced three new indexed-based geometry types (*Indexed Triangles, TriStrips, and Stars*) for representing TINs in the GML schema and extended them to CityGML as an ADE. Our approach allows us to store TIN terrains in CityGML with nearly 20x less storage than the Simple Feature structure in CityGML. This CityGML ADE addresses the issues of massive size of TIN terrain datasets, and explicit handling of vertical triangles in these datasets. It is a stepping stone in the direction of reducing the large size of CityGML datasets while still maintaining usability for different applications.

CityGML differentiates five consecutive LODs (LOD-0 to LOD-4), wherein features become much more detailed in their geometry and semantic differentiation with each increasing LOD (Open Geospatial Consortium, 2012). This LOD concept is very well established in case of buildings, bridges, roads however, this is not the case with other CityGML modules like relief (terrain), landuse and vegetation (Löwner et al., 2016; Biljecki et al., 2016). For

instance, the LOD of a relief object is expressed as integer attribute `gml:lod` with values between 0 and 4. We added new elements *lod1iTIN,lod2iTIN...lod4iTIN* in the CityGML `Relief` (and other modules) to model different LODs of the terrain. However, the proper specification to model the geometry and semantics of terrains at each LOD is still missing in the CityGML specifications. The CityGML specifications do not distinguish between different terrain LODs at geometrical and semantic level although its possible to model different levels of terrain (Luebke, 2003). Since a terrain is a depiction of location-elevation values, it cannot always be an otherwise flat LOD-0 model with one elevation value per triangle in a TIN. A terrain model can also have vertical walls and overhangs. Our future plan is to extend the concept of LODs for terrains and include it in CityGML semantic model of the ADE.

Next step is integrate this ADE in the database to see its overall performance in handling terrain data. We plan to use 3DCityDB[3] (PostgreSQL) for the database implementation of the ADE. Our previous tests have shown that it takes significantly large amount of time to populate and index the TIN datasets with Simple Feature structure than the indexed-based data structures in the database (Kumar et al., 2016b). In case of the ADE, we expect that the loading time from CityGML ADE file to the database will improved most likely. The spatial index will be smaller as it does not require creating complex spatial index like giST (in case of Simple Feature). The indexing can be done at vertex level with a simple B-tree (Ledoux, 2015; Kumar et al., 2016b).

CityGML is designed for the storage and exchange of 3D city models and not for visualizing them. To visualize CityGML models over web, they are usually converted to commonly used 3D graphics formats. We expect the CityGML_iTINs_ADE datasets to load faster over web owing to their small file sizes and indexed-based geometry representations. The CityGML_iTINs_ADE datasets can also be used for other applications utilizing CityGML models such as noise modelling, flood modelling, visibility analysis, visualization for navigation purposes, etc.

Acknowledgements

This work is a part of the research project 3D4EM (3D for Environmental Modelling) in the Maps4Society programme (grant No. 13740) which is funded by the NWO (Netherlands Organisation for Scientific Research), and partly funded by the Ministry of Economic Affairs. For her contribution to this research, the third author (Jantien Stoter) was funded by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (grant No. 677312 UMnD).

References

References

AHN3 (2015). Actueel Hoogtebestand Nederland version 3. Retrieved from `https://www.pdok.nl/nl/ahn3-downloads`.

BGT (2016). Basisregistratie Grootschalige Topografie. Retrieved from `https://www.pdok.nl/nl/producten/pdok-downloads/download-basisregistratie-grootschalige-topografie`.

Biljecki, F., Ledoux, H., and Stoter, J. (2016). An improved LOD specification for 3D building models. *Computers, Environment and Urban Systems*, 59:25–37.

[3]https://www.3dcitydb.org/

Biljecki, F., Stoter, J., Ledoux, H., Zlatanova, S., and Çöltekin, A. (2015). Applications of 3D city models: State of the art review. *ISPRS International Journal of Geo-Information*, 4(4):2842–2889.

Blandford, D. K., Blelloch, G. E., Cardoze, D. E., and Kadow, C. (2005). Compact representations of simplicial meshes in two and three dimensions. *International Journal of Computational Geometry and Applications*, 15(1):3–24.

Boissonnat, J.-D., Devillers, O., Pion, S., Teillaud, M., and Yvinec, M. (2002). Triangulations in CGAL. *Computational Geometry—Theory and Applications*, 22:5–19.

Brink, L., Stoter, J., and Zlatanova, S. (2013). UML-based approach to developing a CityGML Application Domain Extension. *Transactions in GIS*, 17(6):920–942.

Cova, T. J. and Goodchild, M. F. (2002). Extending geographical representation to include fields of spatial objects. *International Journal of geographical information science*, 16(6):509–532.

De Berg, M., Van Kreveld, M., Overmars, M., and Schwarzkopf, O. C. (2000). *Computational geometry*. Springer.

de Laat, R. and Van Berlo, L. (2011). Integration of BIM and GIS: The development of the CityGML GeoBIM extension. In *Advances in 3D geo-information sciences*, pages 211–225. Springer.

Elberink, S. O., Stoter, J., Ledoux, H., and Commandeur, T. (2013). Generation and dissemination of a national virtual 3D city and landscape model for the Netherlands. *Photogrammetric engineering & remote sensing*, 79(2):147–158.

Fisher, P. (1997). The pixel: a snare and a delusion. *International Journal of Remote Sensing*, 18(3):679–685.

Gorte, B. and Lesparre, J. (2012). Representation and reconconstruction of triangular irregular networks with vertical walls. *ISPRS-International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 3826:15–19.

Gotsman, C., Gumhold, S., and Kobbelt, L. (2002). Simplification and compression of 3D meshes. In *Tutorials on Multiresolution in Geometric Modelling*, pages 319–361. Springer.

Gröger, G. and Plümer, L. (2005). How to get 3-D for the price of 2-D? topology and consistency of 3-D urban GIS. *Geoinformatica*, 9(2):139–158.

Gurung, T., Laney, D., Lindstrom, P., and Rossignac, J. (2011a). SQuad: Compact representation for triangle meshes. In *Computer Graphics Forum*, volume 30, pages 355–364. Wiley Online Library.

Gurung, T., Luffel, M., Lindstrom, P., and Rossignac, J. (2011b). *LR: compact connectivity representation for triangle meshes*, volume 30. ACM.

Gurung, T., Luffel, M., Lindstrom, P., and Rossignac, J. (2013). Zipper: A compact connectivity data structure for triangle meshes. *Computer-Aided Design*, 45(2):262–269.

Hug, C., Krzystek, P., and Fuchs, W. (2004). Advanced lidar data processing with LasTools. In *XXth ISPRS Congress*, pages 12–23.

Isenburg, M., Lindstrom, P., Gumhold, S., and Snoeyink, J. (2005a). Streaming Formats for Geometric Data Sets. Retrieved from `http://www.cs.unc.edu/~isenburg/research/sm/download/streaming_formats.pdf`.

Isenburg, M., Lindstrom, P., and Snoeyink, J. (2005b). Lossless compression of predicted floating-point geometry. *Computer-Aided Design*, 37(8):869–877.

ISO (2003). 19107: 2003(E) Geographic information: Spatial schema.

ISO (2007). 19136: 2007(E) Geographic information: Geography Markup Language (GML).

Kadaster (2015). 3DTOP10NL. `http://arcg.is/1GKYy7E`. (Last accessed: April 15, 2016).

Kumar, K., Ledoux, H., and Stoter, J. (2016a). A CityGML extension for handling very large TINs. *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 4:137.

Kumar, K., Ledoux, H., and Stoter, J. (2016b). Comparative analysis of data structures for storing massive TINs in a DBMS. *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, XLI-B2:123–130.

Kumler, M. (1994). An intensive comparison of triangulated irregular networks (TINs) and digital elevation models (DEMs). *Cartographica*, 31(2):1–99.

Ledoux, H. (2015). Storing and analysing massive tins in a dbms with a star-based data structure. Technical Report 2015.01, 3D geoinformation, Delft University of Technology, Delft, the Netherlands.

Ledoux, H. (2017). Representation: Fields. In *The International Encyclopedia of Geography*, pages 1–15. John Wiley & Sons, Ltd.

Löwner, M. O., Gröger, G., Benner, J., Biljecki, F., and Nagel, C. (2016). Proposal for a new LOD and multi-representation concept for CityGML. *ISPRS Ann. Photogramm. Remote Sens. Spatial Inf. Sci.*, IV-2/W1:3–12.

Luebke, D. P. (2003). *Level of detail for 3D graphics*. Morgan Kaufmann.

Luffel, M., Gurung, T., Lindstrom, P., and Rossignac, J. (2014). Grouper: A compact, streamable triangle mesh data structure. *Visualization and Computer Graphics, IEEE Transactions on*, 20(1):84–98.

Lyon, J. G. (2003). *GIS for Water Resource and Watershed Management*. CRC Press.

Mäntylä, M. (1987). *An Introduction to Solid Modeling*. Computer Science Press, Inc., New York, NY, USA.

Muller, D. E. and Preparata, F. P. (1978). Finding the intersection of two convex polyhedra. *Theoretical Computer Science*, 7:217–236.

Nouvel, R., Bahu, J.-M., Kaden, R., Kaempf, J., Cipriano, P., Lauster, M., and Casper, E. (2015). Development of the CityGML Application Domain Extension energy for urban energy simulation. *Proceedings of Building Simulation 2015*.

NVIDIA (2004). NvTriStrip Library. Retrieved from `http://www.nvidia.com/object/nvtristrip_library.html`.

Okabe, A., Boots, B., Sugihara, K., and Chiu, S. N. (2009). *Spatial tessellations: concepts and applications of Voronoi diagrams*, volume 501. John Wiley & Sons.

Open Geospatial Consortium (2011). Opengis® implementation specification for geographic information-simple feature access-part 1: common architecture. *Document reference 06-103r4*.

Open Geospatial Consortium (2012). OGC City Geography Markup Language (CityGML) Encoding Standard 2.0.0.

Open Geospatial Consortium (2014). Modeling an application domain extension of CityGML in UML OGC Best Practice. *OGC Best Practice. Document reference 12-066.*

Penninga, F. (2008). *3D topography: a simplicial complex-based solution in a spatial DBMS.* PhD thesis, TU Delft, Delft University of Technology.

PyFFI (2011). Package PyFFI: Class TriangleStripifier. Retrieved from `http://pyffi.sourceforge.net/apidocs/`.

Ravada, S., Kazar, B. M., and Kothuri, R. (2009). Query processing in 3D spatial databases: Experiences with oracle spatial 11g. In *3D Geo-Information Sciences*, pages 153–173. Springer.

Shewchuk, J. R. (1996). Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator. *Applied Computational Geometry: Towards Geometric Engineering*, 1148:203–222.

Snoeyink, J. and Speckmann, B. (1999). Tripod: a minimalist data structure for embedded triangulations. *Computational Graph Theory and Combinatorics.*

Speckmann, B. and Snoeyink, J. (2001). Easy triangle strips for TIN terrain models. *International journal of geographical information science*, 15(4):379–386.

Stoter, J. and Zlatanova, S. (2003). 3D GIS, where are we standing? In *ISPRS Joint Workshop on 'Spatial, Temporal and multi-dimensional data modelling and analysis', Québec.*

Tse, R. and Gold, C. (2004). TIN meets CAD-extending the TIN concept in GIS. *Future Generation Computer Systems*, 20(7):1171–1184.

VTP (2012). ITF Format - Virtual Terrain Project. `http://vterrain.org/Implementation/Formats/ITF.html`.

Wikipedia (2017). Terrain - Wikipedia, The Free Encyclopedia. Retrieved from `https://en.wikipedia.org/w/index.php?title=Terrain&oldid=805193455`.