A Framework for the Automatic Geometric Repair of CityGML Models

Junqiao Zhao* **, Jantien Stoter*, Hugo Ledoux*

* Department of GIS Technology, OTB Research Institute, Delft University of Technology, Delft, the Netherlands
** College of Surveying and Geo-Information, Tongji University, Shanghai, P.R. China

Abstract. Three-dimensional (3D) city models based on the OGC CityGML standard are becoming increasingly available in the past few years. Although GIS applications call for standardized and geometric-topological rigorous 3D models, many existing visually satisfied 3D city datasets show weak or invalid geometry, as will be shown in this paper, and sometimes they are even standard contradicting. These defects prohibit the processing and analysis of such models. As a result, intensive work of model repair has to be conducted which is complex and labour-intensive. Although model repair is already a popular research topic for CAD models and is becoming important in GIS, existing researches either focus on certain defects or on a particular geometric primitive. Therefore, a framework that investigates the full set of validation requirements and provides ways to repair a CityGML model according to these requirements is needed and proposed in this paper. First, this paper demonstrates and classifies the errors found in existing CityGML models. Then, the repair question is defined based on the formal definition of both valid and invalid models. Finally, a recursive framework aiming at obtaining a valid, strong aggregated and standard conforming CityGML models is finally presented. The geometric terms adopted in this paper are compliant with the ISO19107 standard. Future work will further implement the framework.

Keywords: 3D models, CityGML, Validity, Repair

1. Introduction

The three-dimensional (3D) Geo-information has been treated as one of the essential sources of the latest and future GIS applications (Gruen 2008, Gröger et al. 2012, Stoter et al. 2013). However, current practice of digital city construction still focuses mainly on the visual appearance of the model rather than on the correctness of geometric-topological structure (Gröger & Plümer 2009). As a result, most of the analysis applications cannot be conducted with these ill-posed models, such as physically based structural and environmental analysis, indoor navigation and emergency management (Hughes et al. 2005, Isikdag et al. 2008, Haala et al. 2011). This is seen as a significant bottleneck of the 3D GIS industry and a waste of the expensive modeling efforts and expenses.

One of the error sources of 3D city models starts in the early stage of modeling. To produce visually satisfied 3D city models with the least effort, producers employ interactive modeling tools, i.e. 3D studio MAX, Maya, SketchUp and AutoCAD, to shape the appearance of the city objects with polygonal meshes. However, the freedom provided by these tools leads to error-prone mesh models (Botsch et al. 2007). In many projects, model library is widely used which dramatically decreases the modeling costs for similar features among 3D city models (Badler & Glassner 1997). However, mismatches between model parts create various types of errors such as intersection (*Figure 1 left*), overlap and gaps. Another error source is the model optimization to reduce the data size of the model. The optimization is accomplished by vertex welding and simplification in which close vertices and trivial triangles are merged. In this process, complex vertex and edges as well as degeneracies are often produced (*Figure 1 right*). Moreover, modelers usually delete the concealed surfaces to further decrease the data size, which leads to the incompleteness of the model (Zhao et al. 2012).

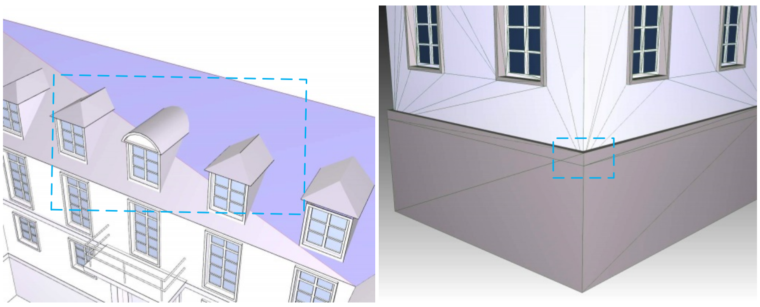


Figure 1. Errors found in visually satified 3D models (left, intersection between components; right, complex edges produced by merging).

Besides the errors that originated from the modelling process, errors can be created from conversion and semantics editing (Nagel et al. 2009). Many of the currently available CityGML building models are produced from conversion of CAD models. The different definitions of building structures in the GIS and the CAD domain lead to inappropriate interpretation of building components. For example, walls in the Industry Foundation Class standard (IFC) in CAD domain are often modeled as *solids* but should be represented by *surfaces* in CityGML (Liebich et al. 2010, Gröger et al. 2012). For the sake of simplicity, the solid wall is converted into a set of *MultiSurfaces*, which omits not only the volume information of the component itself but also the topological relationships between walls. Additionally, errors may occur in the manual semantics editing process of CityGML models, if mistakes are made when matching geometries with CityGML semantics (Benner et al. 2005, Wagner et al. 2012).

Repair of 3D models has become an important topic in the field of CAD and computer graphics. A number of methods have been proposed for the repair of polygonal meshes (Ju 2009, Campen et al. 2012). These approaches are not sufficient for city models, since 3D city models are aggregated models, made up by heterogeneous *geometric primitive*s in multiple dimensions. In addition, for 3D city models based on CityGML, semantics plays an essential role for interpreting the model. Thus attentions should also be paid on the semantics attached with geometry such as geometric types and rules for representing certain models, other than only geometry.

In the field of GIS, previous researches studied the validation of geometrical models. Van Oosterom et al. (2004) proposed an extended definition and a set of validation rules for *polygons*. Tolerance issue was introduced for practical consideration. Kazar et al. (2008) introduced the validation rules for geometry defined in Oracle, especially the *surface* and *solid*, but no repair method is provided. Verbree & Si (2008) and Ledoux et al. (2009) proposed methods to validate the GML *solids* based on tetrahedralizations. Gröger & Plümer (2012a, b) proposed rules and axioms for consistent representation and updating of 3D city models. However, their definition of geometry is not fully conformed to the ISO19107 standard (Herring 2005).

Recently, Ledoux et al. (2012) developed a triangulation based repair method for planar *polygons* with *holes*. Repair pipeline for CityGML models have been proposed in (Bogdahn & Coors 2010, Wagner et al. 2012). Although both the geometric and semantic aspects are included, they only repair errors that are similar to mesh repair such as *holes* and *incorrect orientations* of mesh surface, while the overview of the different types of geometric errors of CityGML model and their repair is not provided.

Started with the introduction of the geometric model of CityGML in *Section* 2, this paper demonstrates and classifies the errors found in existing CityGML models in *Section* 3. Then, the repair question is defined based on the valid and invalid models defined from an application perspective (*Sec-*

tion 4). All these definitions are based on the ISO 19107 standard (Herring 2005). According to the repair question, a framework for repair of geometric errors of CityGML is proposed in *Section* 5, which is a recursive framework and deals with the hierarchical models of the CityGML model. Finally, we conclude with a discussion on the implementation issues and on future realizations.

2. The Geometry Model of CityGML

To understand the repair task of a CityGML model, which is partly using the most restrictive geometry as possible, a brief introduction to the geometry types used in CityGML is provided.

The geometry model of CityGML is based on the ISO19107 standard (Herring 2005). An object can either be represented by a single *geometric primitive* (*GM_Primitive*) or by a collection of *geometric primitive*s. The 0-dimensional (0D) *geometric primitive* is a *point* (*GM_Point*) that represents a position by a 3-tuple of real numbers when embedded in a 3D Euclidean space. The 1-dimensional (1D) *geometric primitive* is a *curve* (*GM_Curve*), which represents the continuous image of a line. In CityGML, it is restricted to the straight *line string* (*GM_LineString*). The 2-dimensional (2D) *geometric primitive* is a *surface* (*GM_Surface*), which represents the continuous image of a region of a plane. In CityGML, *surfaces* are restricted to planar *polygons* (*GM_Polygon*). Finally, the 3D *geometric primitive* is a *solid* (*GM_Solid*), which represents the continuous image of a region of Euclidean 3 space.

A collection of *geometric primitive*s is represented by *aggregate* (*GM_Aggregate*) or *complex* (*GM_Complex*). The *aggregate* allows all kinds of arrangement of *geometric primitive*s without strict restrictions. They can intersect, overlap or isolate with each other. It is the loosest collection of *geometric primitive*s. On the contrary, the *complex* describes a more strict aggregation in which each *geometric primitive* should be *simple*[1], and should not intersect with each other. They touch only at the shared *boundaries*.

If a collection only contains *geometric primitive*s of the same dimension, the *aggregate* and *complex* can further be specialized in *MultiPrimitive*

---

[1] *Simple* means that every point in the *interior* of the object must have a metric neighborhood whose intersection with the object is isomorphic to an *n-sphere*, where *n* is the dimension of this *GM_Object* (Herring 2005).

(*GM_MultiPrimitive*) and *composite*[2] (*GM_Composite*) respectively. An example of *MultiSurface* and *CompositeSurface* is shown in *Figure 2*.

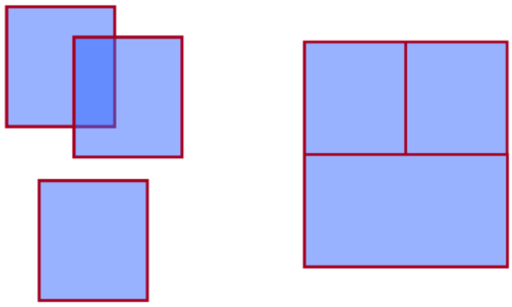


Figure 2. Geometric types for a collection of *surface* primitives (Gröger et al. 2012) (left, *MultiSurface*; right, *CompositeSurface*)

The inheritance diagram of geometric types discussed within this section is shown in *Figure 3*. Their comparison is shown in *Table 1*. After introducing the geometry model, errors existing in available CityGML models can be studied.

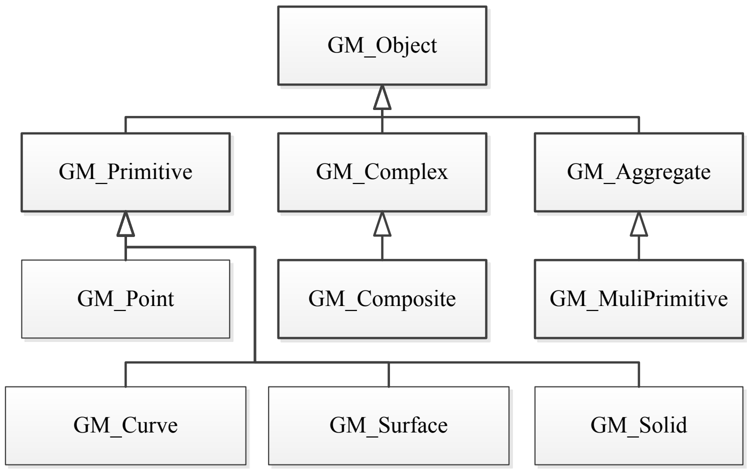


Figure 3. Inheritance of geometric representation types defined in ISO19107 (Herring 2005)

[2] The *composite* should be isomorphic to a primitive.

	Contain boundary	Same dimension	Allow isolation	Aggregation
GM_Aggregate	Likely	No	Yes	Weak
GM_Complex	Yes	No	Yes	Strong
GM_MultiPrimitive	Likely	Yes	Yes	Weak
GM_Composite	Yes	Yes	No	Strong
GM_Primitive	No	Yes	No	Strong

Table 1. The comparision of abstract geometric representations

3. Errors in CityGML Models

According to the error sources of CityGML modeling described in *Section* 1, geometric errors in CityGML datasets can be classified into three types: invalid geometry, weak aggregation and standard contradiction.

3.1. Invalid geometry

The geometry of a CityGML model may contain geometric-topological errors of any kinds. An inventory of recognised general errors in *geometric objects* can be summarized from existing researches (*Table 2*). Errors of a *composite* are listed along with the corresponding *primitive* because they share the most semantics (Herring 2005).

	Errors	Examples
Point/ composite point	Duplicate;	*Point*s shared the same coordinates (Defined with some *tolarance*)
Curve/ composite curve	Degeneration; Self-intersection and folding; Crack/Open (if it's a *ring*);	*Curve* represented by a *point*; *Curve* intersects or overlaps with itself; Gaps within a *curve or a ring is not closed*;
Surface/ composite surface	Degeneration; Self-intersection and folding;	*Surface* represented by a *curve* or a *point*; *Surface* intersects or overlaps with itself; A *hole* incorrectly touches or intersects with *boundaries*;

	Invalid *holes*; Orientations; Non-2-manifold; Non-planar (if it's a *polygon*); Open (if it's a *shell*);	*Surface*s with inconsistent orientations; Complex edge and singular vertex; *Bended polygon*; *Holes* on a *shell*
Solid/ composite solid	Degeneration; Self-intersection; Invalid *voids*;	*Solid* represented by any of the lower dimensional *geometric primitives*; *Solid* intersectes or overlaps with itself; The *void* degenerates, incorrectly touches or intersetes with *voids* (including itself) or the exterior *shell*;
Aggregate	Any or all of the above;	Polygonal soup

Table 2. The inventory of possible errors in *geometric objects* (examples of corresponding errors are also listed in sequence)

3.2. Weak aggregation

Even though each *geometric object* is valid, a collection of *geometric objects* may be weakly aggregated. For example, a *complex* is represented by an *aggregate*, or a *geometric primitive* is represented by the aggregation of its lower dimensional *boundaries*. These defects limit the use of the geometry in applications which prefer tight input, such as solid models. Thus, this can be considered as an error, termed *weak aggregation* (Herring 2005).

In a CityGML model, *weak aggregation*, i.e. *MultiPrimitive*, is frequently applied, because it offers the compatibility of the standard by accepting most 3D models represented by polygonal mesh. However, freedom granted by the *weak aggregation* would cause inconsistencies when part of the geometry is not modified correspondingly with its neighbours (*Figure 4*). Thus the *weak aggregation* fits the visualization purpose but is not friendly to geometric processing.

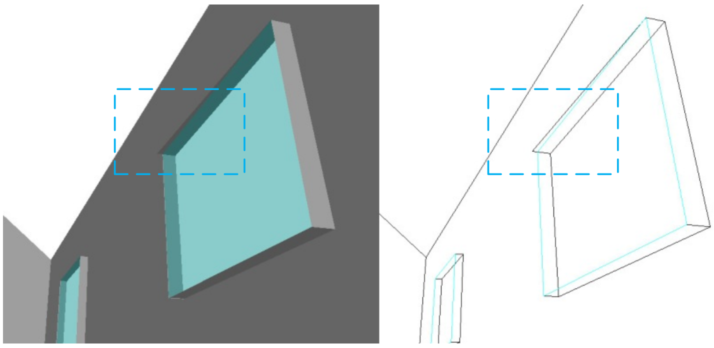


Figure 4. Inconsistency at the *boundary* of *surfaces* in a *MultiSurface* (found in the official example of LoD4 building of CityGML 2.0 dataset)

3.3. Standard contradiction

In CityGML, the geometric form used to represent a feature has been specified in a certain thematic model. A representation contradicting the standard will cause problems in the downstream applications, even the representation is a stronger aggregation than that defined in the standard. For example, a CityGML conforming parser would not be able to extract the right *surface* geometry from a wall defined by *solid*.

In some application domains, more specified implementation rules have been proposed based on the CityGML standard. *Figure 5* gives an example of the valid and invalid representation of walls and roofs in (SIG 3D/AG Quality 2012). Regarding the multiple choices of modeling, these rule sets should also be exploited as constraints that confine the geometric forms of a feature.

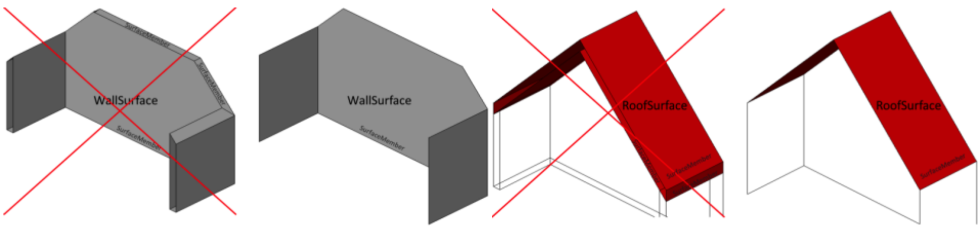


Figure 5. Rules for the modeling of 3D objects (left, invalid and valid WallSurfaces; right, invalid and valid RoofSurface)

4. Validity of CityGML Models

Before repairing, we should first agree on an exact definition of valid and invalid 3D city models. The ISO19107 standard has provided the criteria of valid geometry, such as *simple* and *orientable* (Herring 2005). The question is whether these criteria are sufficient for the downstream applications.

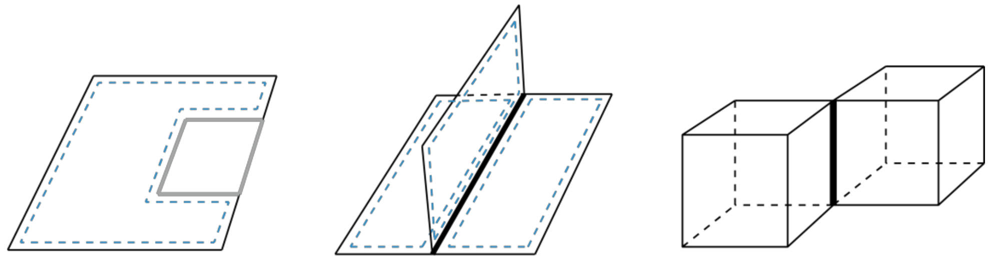
In visualization applications, such as urban planning and virtual reality, only the geometric primitives, i.e. point, curve and surface are required, thus there is no strict requirements for the input geometry (Hearn & Baker 1996). In more sophisticated applications such as 3D cadastre, accurate representations of parcels are mandatory, thus the n-D *geometric object* should represents the continuous image of an n-D *space* (Herring 2005), and should not intersect with itself or with other *geometric objects* (Thompson & van Oosterom 2011). Thus *simple* geometry is appropriate for this application.

However, analysis and processing intensive applications such as structural analysis and simulation demand more strict local and global properties for the input model. If are restricted to the boundary-based representation, these algorithms usually rely on orientable 2-manifold *surfaces*[3] (2-manifold for short) (Botsch et al. 2007). However, the *surface* of a *simple geometric object* might not be a 2-manifold. *Figure 6* a) shows a *surface* with its interior *ring* (shown by the grey line) tangent to its exterior *ring* (shown by the dark line) at an shared boundary edge. Because its *interior* (shown by the blue dash line) is isotropic, this *surface* is *simple* according to the definition (Herring 2005), but the shared *boundary* forms a dangling edge which is non-2-manifold. In *Figure 6* b), a *simple complex* composed of three *surface* primitives is illustrated in which the *surfaces* touch at a shared *boundary*. This *boundary* forms a *complex edge*, thus is non-2-manifold. For *simple* 3D *geometric objects*, its *surface* can also be non-2-manifold because of tangency (*Figure 6* c)).

In conclusion, *simple* is not sufficient to describe the validity of the input model for advanced applications. The least requirement to support the geometric processing intensive applications is that all the geometry that to be processed should be 2-manifold. However enforcing 2-manifold globally for a CityGML model would be too restrictive because 2-manifold based representation of a collection of geographic features is intricate and difficult to

[3] a manifold of dimension n is a topological space that near each point resembles n-dimensional Euclidean space. More precisely, each point of an n-dimensional manifold has a neighbourhood that is homeomorphic to the Euclidean space of dimension n.

model (Gröger & Plümer 2009, Thompson & van Oosterom 2011). Therefore, the validity of CityGML model should be defined in different hierarchies according to the application requirements.



a) b) c)

Figure 6. Non-2-manifold *simple geometric objects* (a) a *simple surface* with its interior *ring* tangent to its exterior *ring*; b) a *simple complex* of surface (Note it's not a *composite surface* which should be isophomic to a *surface*); c) a *simple composite solid*)

Because the CityGML model is composed of a hierarchical structure of features represented by instances of subclasses of *CityObject* (Gröger et al. 2012), we define the basic units of this hierarchy that to be used for the geometric processing purpose as *component models*. Their parents, collections of *component models*, are termed as *aggregate models*. Based on this definition, more exact definitions of valid geometry of *component model* as well as of *aggregate model* are proposed in the next sections.

4.1. Valid geometry of component models

Based on the previous definition, the *component model* should be treated as the unit of geometric processing applications, thus 2-manifold should be the criteria for the valid geometry.

We define the *dimension* of a component model as the highest dimension of its *geometric objects*. If a *component model* is represented by any of the 0D and 1D *geometric objects*, *simple* defined in the ISO19107 standard is sufficient because they are the lower dimensional primitives of a 2-manifold (Herring 2005). However, if 0D or 1D *geometric primitives* are presented in a higher dimensional *component model* (2D or 3D), they should be excluded because they form dangling cases which are non-2-manifold.

For 2D *component models*, they should only contain *surface* primitives with consistent orientations. For a surface without *hole*, a *simple surface* is also a 2-manifold with boundaries[4], thus in this case the *simple surface* is valid. The *closed simple surface*, a *shell*, represents a closed 2-manifold which is also valid. For a *valid composite surface*, each of the *surface*s included should all be *simple*, and self-intersection or isolation between *surface*s is not allowed. *Surface*s should only touch in an appropriate manner in which the touched part between *surface*s should be the existing shared *boundaries*, i.e. edges or vertices. Therefore "free touching" is not allowed, because the combinatorial structure does not represent the shared primitives, resulting in a gap. These valid and invalid cases of a *composite surface* are shown in *Figure 7*. For a *MultiSurface*, the valid geometry should contain only *simple surface*s with no intersections. However, the *surface*s can be isolated from each other or touch at an existing shared *boundary*. Both are valid cases that form a 2-manifold[5].

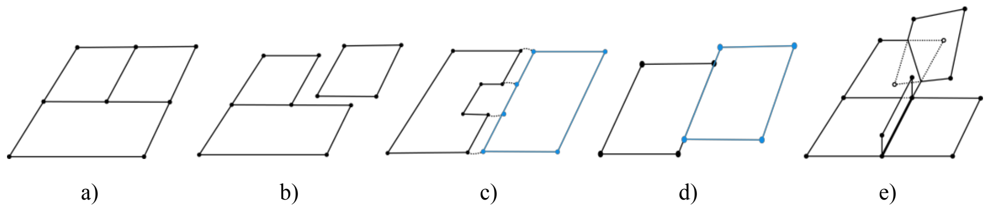


Figure 7. Valid and invalid cases of a *composite surface* (a) a valid *composite surface* which is also simple; b) an invalid *composite surface* with one of its *surface* isolated; c) an valid *composite surface* where the dashed lines indicate the two vertices are identical; d) an invalid *composite surface* contain a "free touching" between two *surfaces*; e) an invalid *composite surface* with its *surface*s intersected and form a complex edge)

If *holes* are present in a *surface*, the validity should be defined more carefully because of the possible tangencies. If all the interior *ring*s that form the *surface* are isolated and are located within the only exterior *ring*, the *simple surface* is a valid 2-manifold with boundaries. However, if *ring*s touch, they should only touch at an existing vertex (*Figure 8* a)) and do not break the interior of the *surface* into more than one *connected components*

[4] This boundary indicates the edges with only one neighbour face

[5] The disjoint union of a family of *n*-manifolds is a *n*-manifold (Lee 2010)

(*Figure 8* d)). If *ring*s touch at a shared edge, the edge will become an invalid dangling edge (*Figure 8* c) and e)). The "free touching" as shown in *Figure 8* b) will cause the error of T-junction, thus it is also not valid. If *holes* are present in a *shell*, the *shell* is invalid according to its definition. For a valid *composite surface* that contains *holes*, the *surface*s included should all be valid, and they should not intersect or isolate, but touch only at shared existing *boundaries* in an appropriate manner. Thus the complex edges and "free touching" which causes T-junction are also not allowed (*Figure 9*). For a valid *MultiSurface* that contains *holes*, the only difference with the valid *composite surface* is that isolation of *surface*s is allowed.

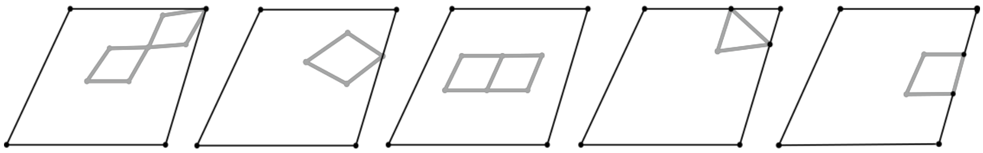


Figure 8. Valid and invalid *surface* with *holes* where vertices and edges of the extierior *ring* are colored in black while vertices and edges of the interior *ring* are colored in grey

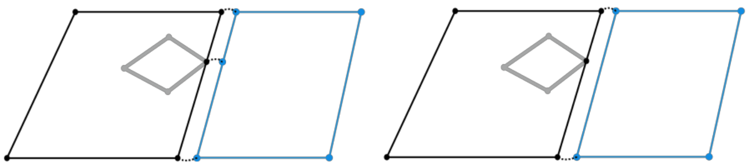


Figure 9. Valid and invalid *composite surface* with *holes* (left, a valid case of *composite surface* with one of its *surface* contains a *hole*, where the dashed lines indicate the two vertices are identical; right, the invalid case of "free touching")

Besides 2D *surfaces*, a *component model* may also be represented by 3D *geometric objects*, i.e. *solid, composite solid* and *MultiSolid*. The validity requirements of 2-manifold should be defined based on *shells* of the *solid* models. For a valid 3D *solid* without *voids* (*holes* in 3D), its exterior *shell* should be a closed valid *surface*, i.e. a compact, oriented 2-manifold without boundary. A 3D *solid* with a *handle* as shown in *Figure 10 left* can be treated as valid when referring to the connectivity of its *interior* and used for representation purpose ((Kazar et al. 2008). However, as a component model, non-2-manifold situations are formed in the touched edges (shown by bold lines), which is invalid for many geometric processing applications.

Figure 10 right shows another case of *simple solid* with non-2-manifod situation.

For a *composite solid*, even though all the contained *solids* are valid, the touch between *solids* leads to complex edges, which is not 2-manifold (as shown in *Figure 11* by bold lines). As a result, the *composite solid* is invalid to represent a *component model* for processing purpose. However, a *MultiSolid* with all its *solids* valid and isolated from each other consists of a set of 2-manifold objects, thus is valid.

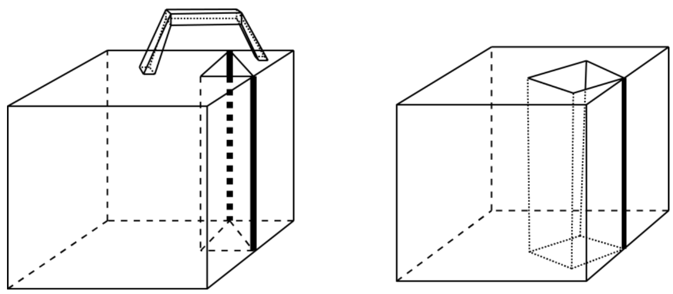


Figure 10. Non-2-manifold *solids* with their *interior* connected and *simple*

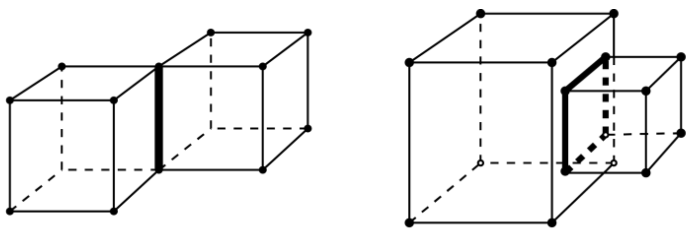


Figure 11. Non-2-manifold cases formed in *composite solid*

For *voids* in 3D, interior connected *solids* shown in *Figure 12* b) and c) present *complex edges* when two *shells* touch with each other. As a result, a valid 2-manifold *solid* with *voids* should exclude any case of tangency between *shells*. Likewise, a *composite solid* with *voids* is also not a 2-manifold. And a valid *MultiSolid* with *voids* should contain only the isolated valid *solids*, as shown in *Figure 12* a).

A valid 3D component model can also contain 2D surfaces. However, the *surfaces* should be isolated from *solids* in order to prevent non-2-manifold situations via tangency.

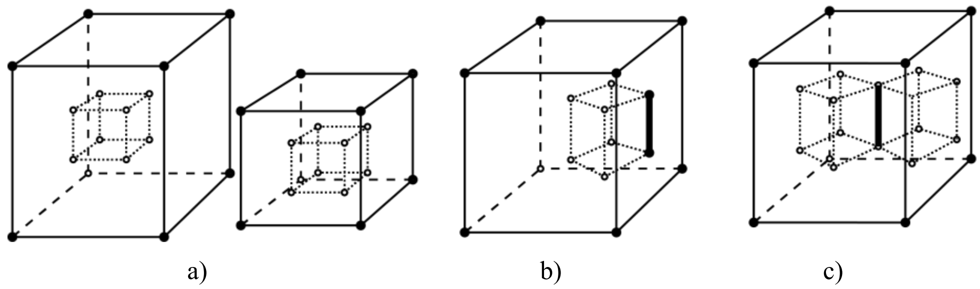
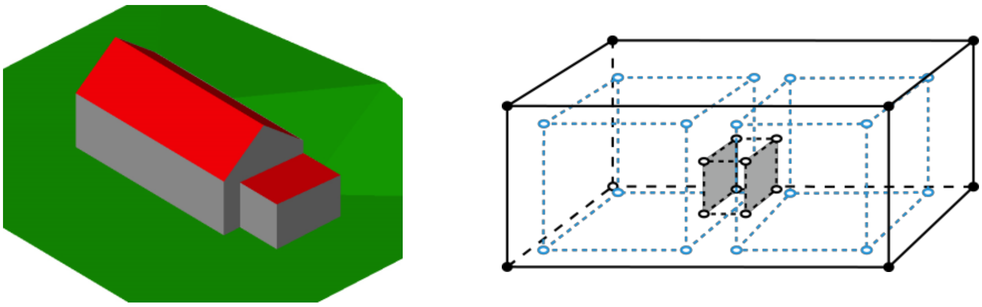


Figure 12. Valid and invalid *solid* with *voids* (a) two valid *solid* with *void* (form a valid *MultiSolid*); b) invalid *solid* with its interior *shell* touch the exterior *shell*; c) invalid *solid* with its two interior *shells* touch with each other)

4.2. Valid geometry of *aggregate models*

The *aggregate model* is represented by assembling *component models*. It is used to model more complex features in a CityGML model. Because *component models* may be represented by *geometric objects* of different dimensions, the geometry of an *aggregate model* can be modelled as a heterogeneous set, represented by *aggregate* or *complex* (Herring 2005). However, *aggregate* allows the intersection of primitives which is invalid for most applications. Thus, *complex* should be defined as the valid representation for *aggregate models*, where *component models* can disjoin, or touch only at shared *boundaries*.

Since the valid *component models* are defined as 2-manifold, they can already support most of the downstream applications. Thus the non-2-manifod geometry can be allowed for an *aggregate model*. In addition, 2-manifold based representation is difficult to represent relationships between two solids (Cavalcanti et al. 1997). Based on the Xlinks, *complex* can use the shared *boundary* to indicate the connectivity between two *component models* (Gröger et al. 2012), For example, a building and an attached garage can be concisely represented by a *composite solid* as shown in *Figure 13* a), even though it is not a 2-manifold as a w*hole*. Otherwise, each of the buildings should be represented by a *solid* and certain relationships should be defined to describe their connection via walls. In another case, when representing two connected rooms inside a building (*Figure 13* b)), it's proper to represent the model by a *complex*, which employs a *composite solid* to represent the collection of two connected rooms, and represent the building as a *shell*. It should be noted that the orientations of the *composite solid* are inverted (Gröger et al. 2012). This example also shows that the hierarchy of *aggregate model* and *component model* can provide the capability for advanced analysis applications and can also maintain the power of representation.



a) b)

Figure 13. Building represented by *complex* (a) A building with a garage attached on one of its wall surface (Gröger et al. 2012); b) Demonstration of two connected rooms inside a building)

4.3. Validity, aggregation enhancement and standard conforming

Besides geometry, the *weak aggregation* and standard contradiction described in *Section 3.2* and *Section 3.3* should also be prohibited in a valid model. Concerning the *weak aggregation*, for example, a *solid* model represented by a valid *MultiSurface* needs further check for closure in order to be used as a volume, which is often ignored in algorithms. As a result, if various representations are allowed, the strongest aggregated geometric type should always be chosen for the valid model. The standard contradicting valid geometric model will cause more serious problems. Therefore, if this occurs, it should be converted to standard conforming representations, even though a weaker aggregated geometry may be the result.

5. Repair Framework of CityGML Models

5.1. The definition of the repair question

After the discussion of the validity criteria for CityGML models, the definition of the repair question can be made, which clarifies the goal and scope of the repair.

The input model for repair, *M*, is a (not yet validated) CityGML model which may contain non-2-manifold or *non-simple* geometry. The geometry may also be weakly aggregated and contradicting the CityGML standard and certain additional implementation rules (MOHURD 2010, Stoter et al. 2011, SIG 3D/AG Quality 2012, van den Brink et al. 2012, Stoter et al. 2013).

The output model from repair, *M'* should be a CityGML model, in which each *component model* must be an orientable 2-manifold and each *aggregate model* must be a *complex* which is *simple*. Based on the valid geometry, the *component model* as well as the *aggregate model* should be structured as tight as possible. And they must obey the restrictions defined in the CityGML standard and the related implementation specifications.

The *component model* and the *aggregate model* are defined and extracted based on the semantics and the geometric information contained in the model. Thus the correctness of the semantics is assumed. For generic 3D city models without semantics, the *component model* and the *aggregate model* should be defined purely based on geometry, such as the largest connected component, or depending on certain application requirements. The repair process discussed in this paper will only focus on the geometric errors. Other issues, such as positional accuracy (Akca et al. 2010) and thematic quality defined in the ISO19157 standard (ISO 2012) are outside the scope of this paper and will therefore not be addressed.

5.2. The recursive paradigm

According to the hierarchical structure of CityGML model, the repair of CityGML model should be carried out in different levels in a sequential order. It is straightforward to employ a recursive approach to fulfil the repair goals. After repai*ring* each *component model*, the higher level *aggregate models* are repaired in a next step (*Figure 14*), until the w*hole* geometric model is valid as defined in *Section 4*.

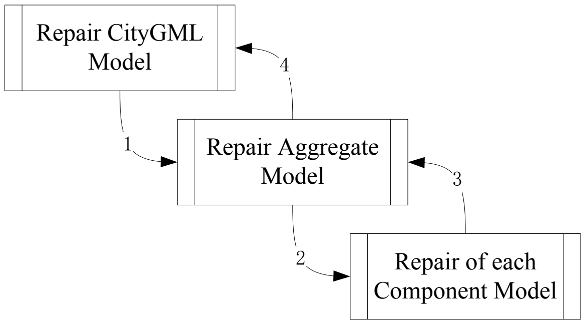


Figure 14. The recursive framework for repair of CityGML models (The numbers represent the order)

5.3. Repair of the *component model*

The repair of *the component model* consists of three sequential steps: geometric repair, enhancement of the weak aggregation and standard conformation. This order is made because geometry is the basis of the other two repair steps. The standard conforming requirement is an obligation for the CityGML model. In addition, a strong aggregated geometry, even though in some cases standard contradicting can be easily modified in a standard conforming geometry. Conversely, if conformation is conducted before the aggregation enhancement, the aggregation enhancement may result again in a not standard-conforming geometry. The three steps of repair are described in the next three sections.

5.2.1. Step 1: Geometry repair

The goal of this repair step is to convert invalid geometry to valid geometry. Since the invalid 0D and 1D *geometric objects* have to be repaired to *simple* versions. This process includes the removal of duplicate *point*s by introducing a *tolerance* value, the conversion of the degenerated *line segments* into *point*s and the decomposition of the intersected *line segments*. If the 1D *geometric object* is a *ring*, the closeness of the *line segments* should also be checked and ensured.

For invalid 2D *surface* primitive, Ledoux et al. (2012) have proposed a triangulation based repair method for planar *polygons*. For a more general *surface*, repair methods for each of the error types illustrated in *Section 3.1* should be employed, as shown in *Table 3*.

Firstly, *ring*s of each *surface* patch should be validated and repaired using the repair methods for *curve*s. Besides, the planarity of each *ring* should also be ensured for *polygons* by introducing a *tolerance* value (van Oosterom et al. 2004). Then, the possible degeneration of *surface patches* should be checked. If degenerated, a *surface* should be converted to the proper primitive types, e.g. a *line string* or a *point*. If an edge or a vertex has no neighbour faces, it is a dangling primitive and therefore should be removed. For a *shell*, there should be no *holes* in any of its *surface patches*. If *holes* are present, *hole*-filling algorithms should be employed (Liepa 2003). If a *surface patch* intersects with itself or with others in a set of *surface patches*, the intersection should be detected and the intersected *surface patches* should be decomposed by inserting vertices and edges. If the *surface* contains several isolated parts, it should be separated into different *surface*s or be converted to a *MultiSurface*, which has to be repaired afterwards. Because of the decomposition of intersected *surface patches*, complex edges and vertices may appear. Algorithms cutting along these complex primitives and stitching them in an appropriate way should be used (Guéziec et al. 2001). Finally, the orientations of all the *surface patches* should be checked for consistency. If incorrect, the orientations should be restored by propagating from the orientation of a valid *surface patch*.

It is important to identify the optimal order of applying these repair algorithms. Therefore, a pairwise comparison matrix of these steps is built in *Table 4*. The matrix shows that the step of separating the isolated *surface* parts should be conducted at the beginning. Then each part of the original *surface* is repaired separately following the order of F-A-B-C-D-E-G-B-C-H. Because decomposing *surface patches* may also lead to degeneracy and dangling primitives, the repair steps B and C should be conducted again. In the implementation, a *surface* can be triangulated after the repair of its *ring*s, because the 2-simplex based representation can help the repair process (Campen et al. 2012, Ledoux et al. 2012).

The repair of *composite surface* is similar to the repair of a *surface* where *surface*s inside the *composite surface* can be treated as *surface patches*. If there are isolated *surface*s, the *composite surface* should be separated. For a *MultiSurface*, the repair pipeline of *composite surface* can be employed but the isolation of *surface*s is allowed.

ID	Errors	Repair
A	Errors of *ring*s, non-planar (if it's a planar *polygon*)	Repair of the *ring*s
B	Degeneration	Remove degeneracies and convert them to proper representation

C	Dangling primitives	Remove dangling edges and isolated vertices
D	*Open* (if it's a *shell*)	Hole filling for a *shell*
E	Self-intersection and folding	Decompose the intersected *surfaces*
F	Isolation	Separate isolated *surface* parts
G	Non-2-manifold	Cutting and stitching complex primitives
H	Inconsistent orientations	Restore orientations

Table 3. Errors and their repair methods for 2D *surface*

	B	C	D	E	F	G	H
A	AB	AC	AD	AE	FA	AG	AH
B		BC	BD	BEB	FB	BGB	BH
C			CD	CEC	FC	CG	CH
D				DE	FD	DG	DH
E					FE	EG	EH
F						FG	FH
G							GH

Table 4. Pairwise comparison matrix of repair methods for 2D *surface* (shows the orders of each pair of repair methods)

For a 3D *solid,* all the *shells* that compose the *solid* should first be repaired based on the previous repair method for *surfaces*. Some *solids* may degenerate to a lower dimensional non-solid primitive, such as a *surface* or a *line string*. The representation types should be corrected and the 0D and 1D *primitives* should be removed. If there are *solids* that intersect with each other, the *solids* should either be decomposed and be converted to a *MultiSolid* or they should be merged into one *solid* if they share the same properties. Similar to the previous repair approach for *a surface*, if a *solid* contains several disconnected parts, it should be separated into several *solids* or be converted to a *MultiSolid*. As discussed in *Section 4.1*, *voids* inside a *solid* should not tangent to each other or to the exterior *shell*. If this occurs, it should be healed by offsetting the tangent primitive inward the exterior *shell*. In this process, only the tangent edges should be treated, because the tangent *surface*s results in dangling *surface*s, which should be removed.

These repair processes are listed in *Table 5* with corresponding errors. A pairwise comparison matrix is constructed again (*Table 6*). The proposed sequence of repair is then derived as D-A-B-E-F-C-B-F. Because degeneracies and dangling primitives may be generated in the step of decomposition, they should be removed again at the end.

According to the definition of validity, *composite solid* should be converted to a *MultiSolid* or be merged to a *solid*. Then the remaining separated *solid* can be repaired using the above steps.

ID	Errors	Repair
A	Errors of *shells*	Repair of the *shells*
B	Degeneration	Remove degeneracies
C	Intersection	Decompose or merge the intersected *solids*
D	Isolation	Separate isolated *solids*
E	Invalid *voids*	Offset tangent edges of *void*
F	Dangling primitives	Remove dangling *surface*, edges and isolated vertices

Table 5. Errors and their repair methods for 3D *solid*

	B	C	D	E	F
A	AB	AC	DA	AE	AF
B		BCB	DB	BE	BF
C			DC	EC	FCF
D				DE	DF
E					EF

Table 6. Pairwise comparison matrix of repair methods for 3D *solid* (shows the orders of each pair of repair methods)

5.2.2. Step 2: Aggregation enhancement

Based on the valid geometry, the next step is to enhance the aggregation of the geometry of a *component model*. In this step, the form of the representation of a *component model* is upgraded from a weaker aggregation to a stronger aggregation when possible. Meanwhile, the proper geometry-type is applied.

According to the degree of aggregation, different representations of objects can be sorted as following:

$$aggregate < complex < composite \leq primitive$$

where the *aggregate* defines the least constraints on the geometry which is not favourable by many geometric processing applications. Thus, the configuration of an *aggregate* should be examined and traversed. Those with primitives neatly connected with each other should be enhanced to the type of *complex* or even *composite* when it contains the *geometric primitives* of the same type. According to the ISO19107 standard, *primitive* and *composite* share most of the semantics, except the *boundaries* (Herring 2005). Therefore, if the *boundary* information is not significant for representing the object, a *composite* can be further converted to a corresponding *geometric primitive*. *Primitives* inside the *composite* are then converted to 2D *surface patches* or 1D *line segments*. Because the *solid* is already the highest degree of aggregation in a 3D space, thus the previous upgrading process is not applied on models of this type.

Besides enhancing the aggregation, also the primitive can be converted to more rigorous types. For example, a closed *surface* or a closed *composite surface* can be converted to a *shell*, and a closed *curve* or a closed *composite curve* can be converted to a *ring*. In some circumstances, the 2D *shell* can even be converted to a *solid* and the 1D *ring* can be converted to a 2D *polygon*, when their interiors are homogeneous. Both processes are show in *Figure 15*.

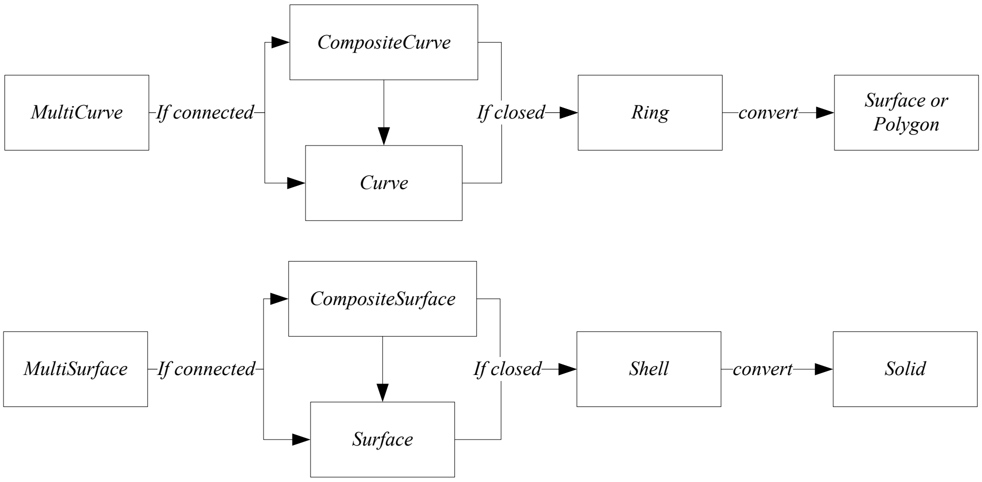


Figure 15. Aggregation enhancement upgrade flow (top, aggregation enhancement in 1D geometry and from 1D geometry to 2D geometry; bottom, aggregation enhancement in 2D geometry and from 2D geometry to 3D geometry)

5.2.3. Step 3: Standard conformation

After the geometric repair and aggregation enhancement, valid and strong aggregated geometry will be produced. However, the aggregation may not be conforming to the standard. *Figure 16* a) demonstrates a strong aggregated *solid* wall components upgraded from original standard conforming *MultSurface* based representation. However the CityGML standard defines the wall in LoD2 as the exterior *shell* that bounds the interior of the building. Therefore, the geometry should be revised to be standard conforming. For this case, the exterior *surface* of the *solid* wall should be extracted as shown in *Figure 16* a) right. Another similar example of roof defined in LoD2 is also shown in *Figure 16* b).

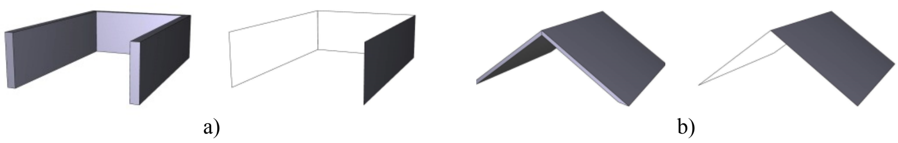


Figure 16. The solid based component and surface based component defined in LoD2 (a) left, walls are represented by *solid* model; a) right, walls are represented by *surface*s extracted from solid based representation; b) left, a roof is represented

by *solid* models; b) right, the roof is represented by *surface*s extracted from solid based representation)

Figure 17 left shows four walls in a LoD4 building model are also enhanced into solid based representation. However, to be standard conforming, they have to be converted into *surface patches* of the exterior *shell* and the interior *shell* (*Figure 17 right*). Because *openings* like window and door are present in LoD3 and LoD4 model, they have to be referenced twice in both the exterior and the interior wall *surface* to ensure the closeness of *shells* representing rooms and the building (*Figure 18*).

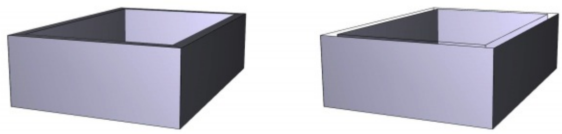


Figure 17. The solid based walls and surface based walls defined in LoD4 (left, walls are represented by *solid* model; right, walls are represented by *surface*s extracted from solid based representation)

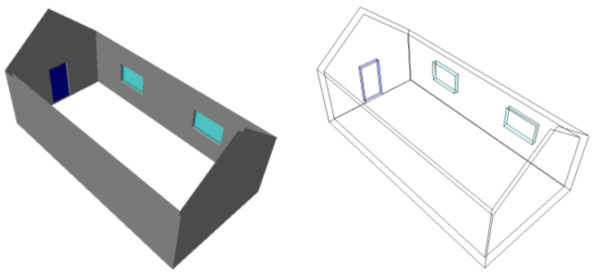


Figure 18. A valid, strong aggregated model of walls with openings defined in LoD4 (Gröger et al. 2012) (left, shown in shaded mode; right, shown in wireframe mode)

Besides the CityGML standard, more specific implementation rules can also be integrated in this step. Since all these rules may be set up from different applications, conversions between one type of geometric representation to another may also be needed, such as the conversion from a *surface* based railing to a *solid* model (Zhao et al. 2012) and the conversion from a *line* based pipe to a *solid* based pipe and its inverse (Thakura et al. 2009). Conversion routines show in *Table 7* may be needed to meet these requirements.

From\To	*Curve*	*Surface*	*Solid*
Curve	N/A	Sweep (Hearn & Baker 1996)	Sweep (Hearn & Baker 1996)
Surface	Median axis (Lee 1982)	N/A	Extrusion (Ledoux & Meijers 2011)
Solid	Abstraction (Mehra et al. 2009); Dimensional reduction (Thakura et al. 2009)	Decomposition; Dimensional reduction (Thakura et al. 2009)	N/A

Table 7. An incomplete list of conversions between different types of geometric representations

To implement the above conformation process, different strategies exist as shown in the *Figure 19*. For the first strategy, the error of standard contradiction is repaired right after the previous two repair steps. For the second strategy, *component models* that are contradicting with the standard will first be marked, and then repaired du*ring* the repair of the *aggregate model*. This is because the conversion or extraction of different geometric representations is easier when conside*ring* their context information. Taking the wall as an example, after marking all the wall components as standard contradicting, they can be first merged to a single *solid* with a *void* (together with *ceilings* and *floors*). In a next step, the exterior *shell* and interior *shell* will be easier to be extracted.

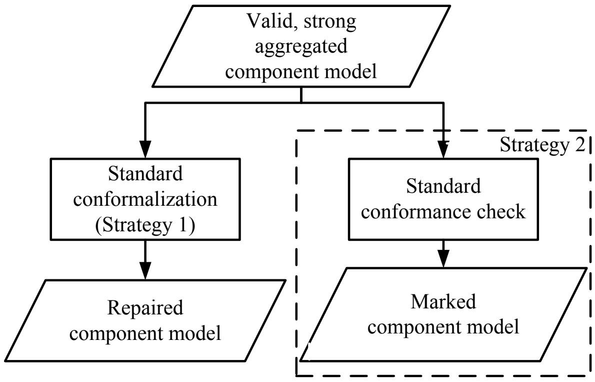


Figure 19. Different strategy to realize the standard conformation process

5.4. Repair of the *aggregate model*

As discussed in *Section 4.2*, the valid geometry of the *aggregate model* should be the type of *complex*. The way to construct a *complex* from a collection of valid *component models* is straightforward. The ISO19107 standard had already provided a working flow (Herring 2005):

a) If two primitives overlap, then subdivide them; eliminate repetitions until there is no overlap.
b) Similarly, if a primitive is not simple, subdivide it where it intersects itself; eliminate repetitions until there is no overlap.
c) If a primitive is not a *point*, calculate its *boundary* as a collection of other primitives, using those already in the generating set if possible, and insert them into the *complex*.
d) Repeat step "a" through "c" until no new primitive is required.

During this repair process, the decomposition of the overlapping *component models* without merging them into a single combinatorial structure will not produce invalid geometry. Then the *aggregate model* should be enhanced to a strong aggregation as discussed in *Section* 5.2.2 and components that are marked as standard contradicting should be further repaired as discussed in *Section 5.2.3*.

6. Conclusion and Discussions

This paper provides an overview of existing errors in CityGML models and proposes a repair framework for these errors. The goal of the repair is defined as validity from an application perspective. Hence, the result of the repair process will be valid, strong aggregated and standard conforming geometry of CityGML models. Based on the repair framework, the implementation should regard some crucial aspects. Firstly, the data structure should be able to faithfully record the input model without missing important information. In the geometry aspect, generic boundary based representation based data structure should be adopted which supports all the geometric types defined in CityGML and is also able to represent invalid non-2-manifold edges and vertices. Tolerance values are important to define the condition of duplication and coplanar, and should therefore be defined carefully. Due to the uncertainty caused by possible rounding-off error, exact predict method or even exact arithmetic should be used (Richard Shewchuk 1997). Then, basic repair routines should be developed, such as *triangulation*, which convert the input model into *simplical* 2-*complex* based representation; *regularization*, which removes dangling 0D or 1D elements (Granados et al. 2003, Worboys & Duckham 2004), and *intersection* and *decomposition*, which extracts the intersected *geometric primitives* and decomposes the model based on the extracted intersection. More mesh repair methods like *hole*-filling, non-2-manifold curing should also be incorporated (Campen et al. 2012). Besides, extraction and conversion from one geometric type to another geometric type as described in *Section 5.2.3* should be implemented in a parameterized way, so that it could be easily employed based on certain requirements derived from rule sets. Finally, the developed repair method should be modularized and made customizable so that it can be adapted to different applications.

Future works comprise the design and implementation of the data structure for the repair of the CityGML models. The automatic repair routines discussed before should then be developed based on the proper data structure. This paper mainly focus on the geometric aspect of repair, thus the semantics of the CityGML model should be further exploited to strengthen the ability of repair on specified models, such as a building. Finally, this paper assumes the correctness of the semantic information, and excludes other repair issues listed in the ISO19157 standard (ISO 2012). These topics should be studied in the follow-up researches.

Acknowledgements

This work is supported by the National Natural Science Foundation of China (41201379 and 41171311), the Dutch Technology Foundation STW, which is part of the Netherlands Organization for Scientific Research (NWO) and partly funded by the Ministry of Economic Affairs, Agriculture and Innovation. (Project code: 11300).

References

Akca, D, Freeman, M, Sargent, I and Gruen, A (2010) Quality assessment of 3D building data. The Photogrammetric Record, 25(132): 339-355

Badler, NI and Glassner, AS (1997) 3D object modeling. SIGGRAPH 97 Introduction to Computer Graphics Course Notes

Benner, J, Geiger, A and Leinemann, K (2005) Flexible Generation of Semantic 3D Building Models, International ISPRS / EuroSDR / DGPF-Workshop on Next Generation 3D City Models. EuroSDR, Bonn, Germany

Bogdahn, J and Coors, V (2010) Towards an automated Healing of 3D urban models. In: T. Kolbe, G. König and N. Claus (Eds), Proceedings of International Conference on 3D Geoinformation. Int. Archives of Photogrammetry, Remote Sensing and Spatial Information Science, Vol XXXVIII-4/W15. Aachen: Shaker Verlag GmbH, pp. 13-17

Botsch, M et al. (2007) Geometric Modeling based on Polygonal Meshes, ACM SIGGRAPH 2007 courses. ACM New York, NY, USA

Campen, M, Attene, M and Kobbelt, L (2012) A Practical Guide to Polygon Mesh Repairing. The Eurographics Association, pp. t4-undefined

Cavalcanti, PR, Carvalho, PCP and Martha, LF (1997) Non-manifold modelling: an approach based on spatial subdivision. Computer-Aided Design, 29(3): 209-220

Granados, M et al. (2003) Boolean operations on 3D selective Nef complexes: Data structure, algorithms, and implementation. In: G. Di Battista and U. Zwick (Eds), Algorithms - ESA 2003: 11th Annual European Symposium. Lecture Notes in Computer Science. Springer, Budapest, Hugary, September 2003, pp. 174‑186

Gröger, G and Plümer, L (2009) How to achieve consistency for 3D city models. GeoInformatica, 15:137-165

Gröger, G and Plümer, L (2012a) Provably correct and complete transaction rules for updating 3D city models. Geoinformatica: 1-34

Gröger, G and Plümer, L (2012b) Transaction rules for updating surfaces in 3D GIS. ISPRS Journal of Photogrammetry and Remote Sensing, 69: 134-145

Gröger, G, Kolbe, TH, Nagel, C and Häfele, K (2012) OpenGIS City Geography Markup Language (CityGML) Encoding Standard. Version 2.0.0. Open Geospatial Consortium. OGC 12-019

Gruen, A (2008) Reality-based generation of virtual environments for digital earth. International Journal of Digital Earth, 1(1): 88-106

Guéziec, A, Taubin, G, Lazarus, F and Hom, B (2001) Cutting and stitching: Converting sets of polygons to manifold surfaces. IEEE Transactions on Visualization and Computer Graphics, 7(2): 136-151

Haala, N, Fritsch, D, Peter, M and Khosravani, A (2011) Pedestrian Navigation and Modeling for Indoor Environments, Proceeding of 7th International Symposium on Mobile Mapping Technology, Cracow, Poland

Hearn, D and Baker, MP (1996) Computer Graphics, C version. Prentice Hall, 652 pp

Herring, JR (2005) ISO 19107:2005: Geographic information-Spatial schema. International Organization for Standardization

Hughes, T, Cottrell, JA and Bazilevs, Y (2005) Isogeometric analysis: CAD, finite elements, NURBS, exact geometry and mesh refinement. Computer Methods in Applied Mechanics and Engineering, 194(39-41): 4135-4195

Isikdag, U, Underwood, J and Aouad, G (2008) An investigation into the applicability of building information models in geospatial environment in support of site selection and fire response management processes. Advanced Engineering Informatics, 22(4): 504-519

ISO (2012) ISO 19157: Geographic information - Data quality. International Organization for Standardization

Ju, T (2009) Fixing geometric errors on polygonal models: a survey. Journal of Computer Science and Technology, 24(1): 19-29

Kazar, BM, Kothuri, R, Oosterom, P and Ravada, S (2008) On valid and invalid three-dimensional geometries, Advances in 3D Geoinformation Systems. Lecture Notes in Geoinformation and Cartography. Springer Berlin Heidelberg, pp. 19-46

Ledoux, H and Meijers, M (2011) Topologically consistent 3D city models obtained by extrusion. International Journal of Geographical Information Science, 25(4): 557-574

Ledoux, H, Ohori, KA and Meijers, M (2012) Automatically repairing invalid polygons with a constrained triangulation, Proceedings of Agile 2012, Avignon, France

Ledoux, H, Verbree, E and Si, H (2009) Geometric Validation of GML Solids with the Constrained Delaunay Tetrahedralization, Proceedings of the 4th International Workshop on 3D Geo-Information, Ghent, Belgium

Lee, DT (1982) Medial axis transformation of a planar shape. IEEE Transactions onPattern Analysis and Machine Intelligence(4): 363-369

Lee, JM (2010) Introduction to topological manifolds. Graduate Texts in Mathematics, 202. Springer New York, 433 pp

Liebich, T et al. (2010) Industry Foundation Classes 2x4 (IFC2x4) Release Candidate 2

Liepa, P (2003) Filling holes in meshes. Eurographics Association, pp. 200-205

Mehra, R, Zhou, Q, Long, J, Sheffer, A and Mitra, NJ (2009) Abstraction of manmade shapes. ACM Trancactions on Graphics (TOG), 28(5): 1-10

MOHURD (2010) Technical Specification for Three Dimensional City Modeling Of China (In Chinese)

Nagel, C, Stadler, A and Kolbe, TH (2009) Conceptual requirements for the automatic reconstruction of building information models from uninterpreted 3D models. In: T.H. Kolbe, H. Zhang and S. Zlatanova (Eds), GeoWeb 2009 Academic Track - Cityscapes, Vancouver, BC, Canada

Richard Shewchuk, J (1997) Adaptive precision floating-point arithmetic and fast robust geometric predicates. Discrete & Computational Geometry, 18(3): 305-363

SIG 3D/AG Quality (2012) Manual for the modeling of 3D objects. Special Interest Group 3D (SIG3D)

Stoter, J et al. (2011) Towards a national 3D Spatial Data Infrastructure: case of the Netherlands. Photogrammetrie-Fernerkundung-Geoinformation, 2011(6): 405-420

Stoter, J et al. (2013) Implementation of a National 3D Standard: Case of the Netherlands. In: J. Pouliot, S. Daniel, F. Hubert and A. Zamyadi (Eds), Progress and New Trends in 3D Geoinformation Sciences. Lecture Notes in Geoinformation and Cartography. Springer Berlin Heidelberg, pp. 277-298

Thakura, A, Banerjeea, AG and Gupta, SK (2009) A survey of CAD model simplification techniques for physics-based simulation applications. Computer-Aided Design, 41(2): 65-80

Thompson, R and van Oosterom, P (2011) Axiomatic Definition of Valid 3D Parcels, potentially in a Space Partition, Proceedings 2nd International Workshop on 3D Cadastres, November, Delft, pp. 397-416

Thompson, R and van Oosterom, P (2011) Connectivity in the regular polytope representation. GeoInformatica, 2(15): 223-246

van den Brink, L, Stoter, J and Zlatanova, S (2012) Establishing a national standard for 3D topographic data compliant to CityGML. International Journal of Geographical Information Science (In press)

van Oosterom, P, Quak, W and Tijssen, T (2004) About invalid, valid and clean polygons. In: P.F. Fisher (Eds), Developments in Spatial Data Handling–11th International Symposium on Spatial Data Handling, pp. 1-16

Verbree, E and Si, H (2008) Validation and Storage of Polyhedra through Constrained Delaunay Tetrahedralization, Geographic Information Science. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, pp. 354-369

Wagner, D et al. (2012) Geometric-semantical consistency validation of CityGML Models, 3D GeoInfo Conference 2012, Quebec City, Canada, May 16-17

Worboys, M and Duckham, M (2004) GIS: A computing perspective. CRC

Zhao, J, Stoter, J, Ledoux, H and Zhu, Q (2012) Repair and generalization of handmade 3D building models, Proceedings of the 15th Workshop of the ICA Commission on Generalisation and Multiple Representation jointly organised with EuroSDR Commission 4 - Data Specifications, Istanbul, Turkey, pp. 10