TOWARDS AUTOMATIC VALIDATION AND HEALING OF CITYGML MODELS FOR GEOMETRIC AND SEMANTIC CONSISTENCY

N. Alam [a, *], D. Wagner [a], M. Wewetzer [b], M. Pries [b], V. Coors [a]

[a] HFT Stuttgart – University of Applied Sciences, Faculty C, Schellingstraße 24, 70174 Stuttgart, Germany
(nazmul.alam, detlev.wagner, volker.coors)@hft-stuttgart.de

[b] Beuth Hochschule für Technik Berlin – University of Applied Sciences, Department II, Luxemburger Straße 10, 13353 Berlin, Germany
(mark.wewetzer | margitta.pries)@bht-berlin.de

KEY WORDS: Healing, Validation, 3D City models, CityGML, Geosemantics

ABSTRACT:

A steadily growing number of application fields for large 3D city models have emerged in recent years. Like in many other domains, data quality is recognized as a key factor for successful business. Quality management is mandatory in the production chain nowadays. Automated domain-specific tools are widely used for validation of business-critical data but still common standards defining correct geometric modeling are not precise enough to define a sound base for data validation of 3D city models. Although the workflow for 3D city models is well-established from data acquisition to processing, analysis and visualization, quality management is not yet a standard during this workflow. Processing data sets with unclear specification leads to erroneous results and application defects. We show that this problem persists even if data are standard compliant. A systematic rule set for the validation of geometric-semantical consistency is developed and implemented. Validation results of real-world city models are presented to demonstrate the potential of the approach. A tool to repair the errors detected during the validation process is under development; first results are presented and discussed. The goal is to heal defects of the models automatically and export a corrected CityGML model.

1. INTRODUCTIOIN

Application and analysis of geo data is moving from traditional GIS applications with 2D map data towards deployment of real 3D data. Virtual 3D city models become more and more available for urban areas. More sophisticated tools for data analysis information extraction are under development. Quality assessment becomes mandatory because reliable and reproducible processing results can only be obtained with correct original data. Since there is no generally acknowledged definition of correctness for 3D city models efforts to define guidelines for modelers and users are discussed; methods to check the data set for compliance with the specifications are necessary.

A general overview of the concept of data quality in the geographic domain is included in (Kresse & Fadaie 2004), which offers a comprehensive summary of the relevant standards, notably of the ISO 19100 series. The paper of (Akca et al. 2010) has a focus on geometric accuracy with respect to the generation process of a model from Lidar data. Discussing the problems of polygonal models, (Krämer et al. 2007) define quality measurements for 3D city models. Some simple algorithms for quality assessment and healing of geometries are presented.

(Campen et al. 2012) provide an extensive collection of typical defects of polygonal 3D models and existing techniques for processing and repair with respect to different fields of application. A detailed analysis of completeness and separation issues in city models is presented by (Zhao et al. 2012). They consider typical properties of semi-automatic generated models and their insufficiencies and develop a generalization method. However, other geometric errors are not investigated.

We present an overview of our research results leading to the definition of certain quality criteria for CityGML models and the development of an automated validation tool. A quality report is the result of this processing step. It includes detailed descriptions of all detected errors.

This information is used as input of a healing process which tries to repair as many errors automatically as possible. The healing procedures are described in detail and experiences with the tool are discussed.

2. AUTOMATIC VALIDATION

2.1 Validation Rules

A validated a data set is expected to be clean, correct and useful for a given application. This implies that different sets of validation rules exist, depending on the intended application.

We separate the validation process in two general steps: first a schema validation for CityGML data to assure schema conformant input to the second step, geometric and semantic validation of the data set. Only the second step is discussed here because XML schema validation is a standard procedure for which sophisticated tools are available.

For the basics of geodata validation we refer to the explanations in (Wagner et al. 2013).

The Special Interest Group 3D has developed guidelines for modelling of 3D city models. The goal is to clearly specify valid alternatives and recommend one of them for general usage. This should lead to city models with known specifications in contrast to the situation today where only the modeller knows how certain features are reproduced. These recommendations are the base for geometric validation rules which have been developed and implemented as part of the research project *CityDoctor* at *University of Applied Sciences Stuttgart*, Germany.

* Corresponding author. This is useful to know for communication with the appropriate person in cases with more than one author.

In addition, some geometric-semantic rules resulting from CityGML requirements are included as plausibility checks for consistency of the data set.

A short listing of the checks is given in the following, more detailed explanations are given in (Wagner et al. 2013).

Polygon checks

1. A linear ring must consist of a minimum of 4 points
2. First and last point of a linear ring are identical.
3. All points of a linear ring *R* are different, with exception of first and last point.
4. Two edges can intersect only in one start-/end point. Other points of intersection or touching are not allowed (to account for rounding errors or polygons which are not perfectly planar, a small tolerance is allowed).
5. All points of the polygon must be located in a plane (a small tolerance is allowed).

Solid checks

6. The minimum number n of polygons to define a solid is four. They must be situated in different planes.
7. A valid intersection of two polygons of a solid either contains a common edge, a common point of a linear ring, or is empty. Common edges and points must be elements of both polygons.
8. Each edge of a linear ring defining a polygon is used by exactly one neighboring polygon.
9. Consistent orientation of polygons of a solid such that common edges according to check 8 are used in opposite direction.
10. The normal vectors of the polygons must point towards the outside of the solid.
11. All parts of a solid must be connected.
12. The graph $G_S = (V_P, E_P)$ of polygons and edges which are meeting in point p_i is connected for all p. Each vertex $v \in V_P$ represents exactly one polygon which contains p. Two vertices are connected with an edge $e \in E_P$ if the polygons represented by these vertices have a common edge that is bounded by p.

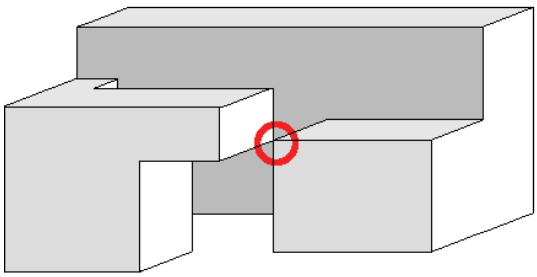

Figure 1. Umbrella check

Semantic checks

13. Orientation of *RoofSurface*, *WallSurface* and *GroundSurface* elements
14. *measuredHeight* in same range as height of building geometry
15. *numberOfStoreysAboveGround* plausible for height of the building geometry
16. *numberOfStoreysBelowGround* plausible for height of underground geometry of the building
17. Relationship of *Building* and *BuilidingPart*

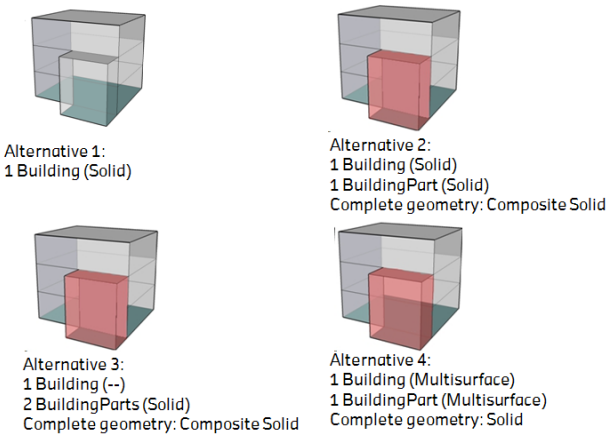

Figure 2. Alternative modeling scenarios for BuildingParts

3. HEALING

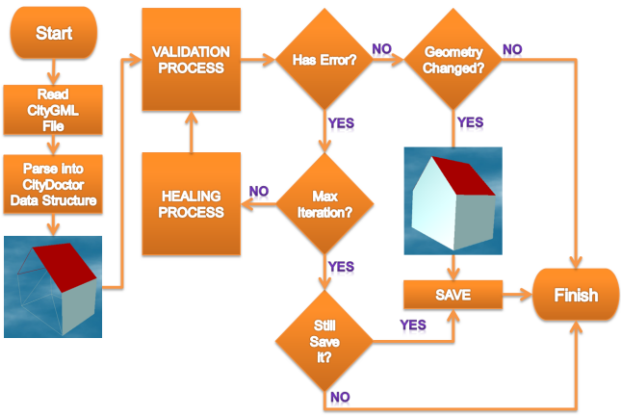

Figure 3. Complete Healing workflow

Errors detected by the validation process are mostly repairable. But it is not certain that the geometry will be correct after the error is fixed. So it has to be an iterative healing process. As errors have a dependency hierarchy, this will also affect the healing process. Healing is done in two phases. Firstly all the polygons are healed and then if polygons pass the validation process solid-errors are healed. In figure 3 the healing process has been illustrated.

3.1 Geometry Healing

POLYGON HEALING

In this phase one error is healed at a time. That means each iteration heals an error and checks the result for validation.

CP_CLOSE: As the first and the last vertex of a polygon must be same, therefore healing would be just to copy the first vertex at the end of the pointlist. The first Linear Ring from figure 4 has four vertices in a sequence $\{P_1, P_2, P_3, P_4\}$ where the last point and the first point are not same. So after healing the pointlist would look like $\{P_1, P_2, P_3, P_4, P_1\}$.

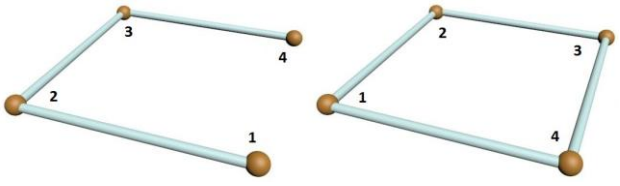

Figure 4. Healing CP_CLOSE error

CP_NUMPOINTS**:** A Linear Ring must contain a minimum number of four vertices in the sequence where the first and the last point are same. In this case these polygons can be deleted because any closed Linear Ring with less than 4 vertices is either a line or a point but not a valid polygon. But if a polygon contains 3 different vertices and the first and the last vertex are not same then it will be healed by the previous healing process and then the number of vertices of the healed polygon will be 4.

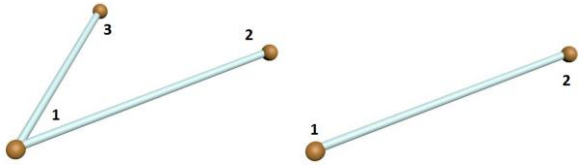

Figure 5. Healing CP_NUMPOINTS error

CP_DUPPOINT: In a Linear ring only the first vertex is allowed to repeat at the end of the point sequence. No other vertex is allowed to repeat within the sequence at any position. In Figure 6 first Linear Ring X contains the point sequence of {P_1, P_2, P_3, P_4, P_5, P_3, P_1} where P_3 is repeating twice. In second Linear Ring Y point sequence is {P_1, P_2, P_3, P_4, P_4, P_1} where P_4 is repeating twice. In third Linear Ring Z point sequence is {P_1, P_2, P_3, P_4, P_2, P_1} where P_2 is repeating twice. In all the Linear Rings one or more vertex is repeating more than once excluding the last point so there are Duplicate Point Error in the Linear Rings. These are healed in two different ways. For Linear Ring Y vertex P_4 comes twice back to back so only one of the instance are kept and the other one is deleted from the pointlist. But for X and Y deleting an instance of vertex will result change in the shape of polygon. So here loops are searched within the pointlist. For X the loops will be {P_3, P_4, P_5, P_3} and {P_3, P_1, P_2, P_3} and for Z it will be {P_2, P_3, P_4, P_2} and {P_2, P_1, P_2}. So the polygons will be split into multiple polygons according to the newly found loops.

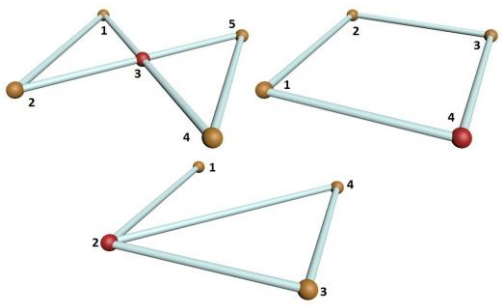

Figure 6. Healing CP_DUPPOINT error

CP_NULLAREA: Linear polygons are healed in this process. When all the vertices of the polygon are linear according to a very small tolerance then the area of the polygon will be zero. Only deleting these polygons will cause an endless loop, because by the healing for the holes repairing, this polygon will be created again. So the embedded edges have to be merged. In figure 7 linear polygon L contains a point sequence of {P_1, P_2, P_3, P_1}, where vertex P_3 is present in both polygons X and Y at the bottom but is missing in polygon Z on top. By this healing process vertices are sorted into two groups and pointlists of the adjacent polygons are rearranged according to the order of the vertices, in this case vertex P_3 will be placed in between P_1 and P_2 in polygon Z. A nonlinear polygon with a zero area is not possible here because a double point error will occur and it will be split into multiple linear polygons by previous healing processes.

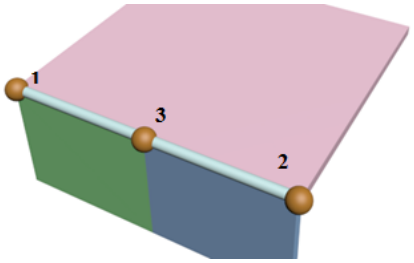

Figure 7. Healing CP_NULLAREA error

CP_SELFINT: Two edges are allowed to intersect only at start and end point of the edge and any other intersection will considered as an error. In Figure 8 first polygon contains point sequence {P_1, P_2, P_3, P_4, P_1} where edge (P_2, P_3) and edge (P_4, P_1) intersects at a point which doesn't belong to the point sequence and in second polygon point sequence is {P_1, P_2, P_3, P_4, P_5, P_1} where edge (P_2, P_3), edge (P_4, P_5) and edge (P_5, P_1) intersects at a point which doesn't belong to the point sequence. So, both has Self-intersection Error of Edges. There are two healing options one is to rearrange the point sequence which works sometimes fine with simple polygon and another one is to extract the intersection points create new vertices with those and place the new vertices in between each intersecting edges. So the first polygon would be {P_1, P_2, P_x, P_3, P_4, P_x, P_1} where P_x is the new vertex. This is not a valid polygon but there is no self-intersection error any more, the double point errors will be healed by the next iteration by its healing process.

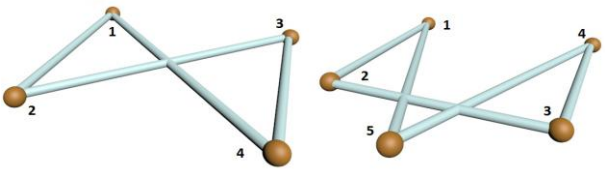

Figure 8. Healing CP_SELFINT error

CP_PLAN: This is very common error and difficult to heal. All vertices of a polygon must lie within the same plane. If a polygon contains point sequence of {P_1, P_2, …., P_n, P_1}, all of the vertices will lie within a plane formed by any three vertices from the point sequence and normal of all vertices on the surface must be parallel. In Figure 9 the polygon has a point sequence of { P_1, P_2, P_3, P_4, P_1} where i.e. vertex P_3 doesn't lie within the plane formed by P_1, P_2, P_4. Sometimes the error is very small like less than a 1 mm. Those are most probably caused by the measurement issue or floating number. In this case a little adjustment of vertices might heal the polygon.

Another healing option is to triangulate the polygon and split it into multiple triangular polygons. But again it is very difficult to decide how to triangulate because there is always more than one possibility and only one is correct. So a little bit of customization according to the error pattern of the model helps a lot. For example while repairing a vertical non planner polygon of wall surface only vertical triangles are accepted as newly triangulated polygons.

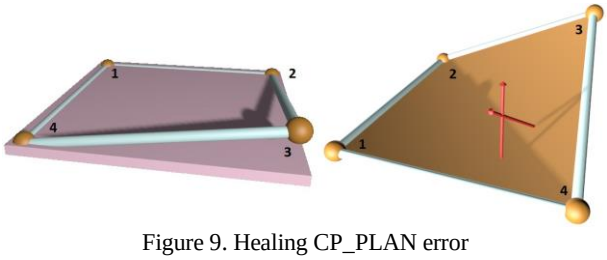

Figure 9. Healing CP_PLAN error

The healing of non-planar surfaces of a building with the first method is divided in three phases:

Healing of the GroundSurface:

The healing of the GroundSurface is identical for the LoD1 and the LoD2. We are identifying the GroundSurface for a LoD1 geometry as the surface with the smallest z-coordinate and the least deviation in respect to direction of the normal vector $\overrightarrow{n_z}$ of the xy-plane P_{xy}. All points belonging to the linear ring of the GroundSurface are being projected on a plane, parallel to P_{xy} and passing through the minimal z-value of the ring. See image XX for an example. The blue, non-planar polygon is being projected on the green, planar one.

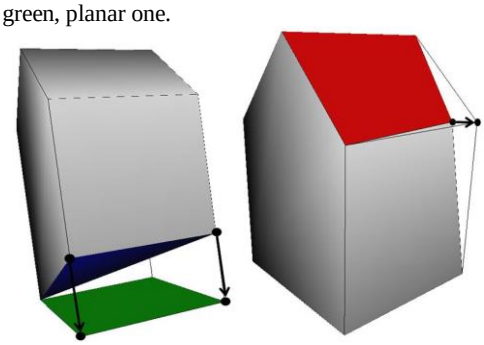

Figure 10. Healing of the Ground and WallSurface

Healing of the WallSurfaces:

As above we are not distinguishing between LoD1 and LoD2 during the healing process of WallSurfaces. We assume that each WallSurface shares a common edge with the GroundSurface and each Surface of a LoD1 geometry, adjacent to the GroundSurface is a WallSurface. Let W_i be the i-th WallSurface, e_i the common edge with the GroundSurface and $\overrightarrow{n_{W_i}}$ the normal vector of the least squares plane through all points of the linear ring of W_i. If $\left\|\angle\left(\overrightarrow{n_z}, \overrightarrow{n_{W_i}}\right) - 90°\right\|$ is smaller than a given ε, then all points of the linear ring are being projected into the plane, spanned by the directional vector of e_i and $\overrightarrow{n_z}$. With this approach we omit walls with a given angle of slope. An example is shown in image XX.

Healing of the RoofSurfaces:

There are two algorithms for healing the RoofSurfaces. The first one handles LoD1 roofs and LoD2 flat roofs. The RoofSurface of a LoD1 building is determined similar to its GroundSurface. It's the polygon with the least deviation according to $\overrightarrow{n_z}$ and maximal z-value. Additionally it's not adjacent to the GroundSurface. We will projecting all points of the linear ring of the RoofSurface into a plane parallel to P_{xy}, passing through the average z-value of all points in the ring, as you can see in image XX.

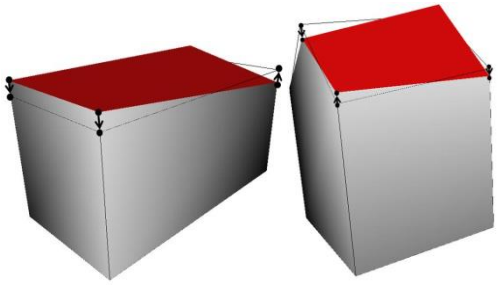

Figure 11. Healing of the LOD1 and LOD2 RoofSurface

The second one handles all other LoD2 roof types and calculates the least square fitting plane for all RoofSurfaces. Each point of a linear ring of the corresponding RoofSurface is projected along the z-axis into the according least square fitting plane of its linear ring. This procedure will be repeated until all RoofSurfaces are planar or a maximum number of iteration is reached. Note that the points are only projected along the z-axis. Hence healed the WallSurfaces remaining planar, even if shared points of RoofSurfaces are moved. See image XX for an example.

SOLID HEALING

In this phase one error is healed at a time. That means each iteration heals an error and checks the result for validation.

CS_NUMFACES: To form a solid minimum four surface is required. Any solid having less than four valid polygons has insufficient number of face error. If a solid has less than 4 polygons then it is not possible to repair, only exception is the structure shown in figure 12. For all other cases the solid would be invalid and deleted from the model by the healing process.

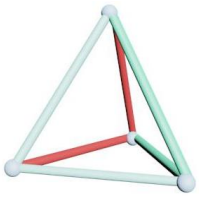

Figure 12. Healing CS_NUMFACE error

CS_SELFINT: Polygons of a solid must meet each other only through edges. Any other intersection of polygons will be considered as a self-intersection error of solid. There are several types of intersection: partially embedded edge, fully embedded edge, partially embedded polygons, fully embedded polygon, normal intersection and undefined intersection. In figure 13 for partially and fully embedded and edge errors like A and B overlapping edge (P_1, P_3) and edge (P_2, P_4) are merged into 3 edges (P_1, P_2), (P_2, P_3) and (P_3, P_4) and the pointlists of respective polygons are rearranged like H. And for partially embedded polygon errors like Y the overlapping regions are trimmed out from the overlapped polygons and new polygons are created from the overlapping regions like R. For fully embedded polygon errors like X the overlapping region is only trimmed out from the bigger polygon.

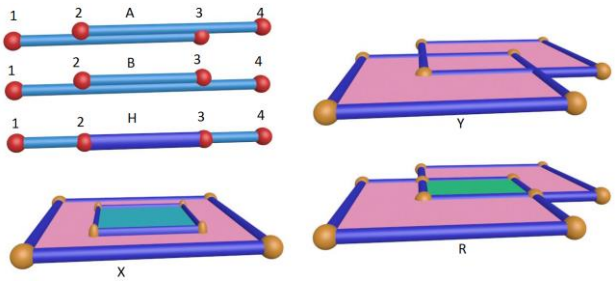

Figure 13. Healing CS_SELFINT error

For a normal intersection like figure 14 healing doesn't brings a valid result. If the intersecting polygons are split into multiple polygons an overused edge error occurs which is very difficult to heal. But still it solves the issue with self-intersection.

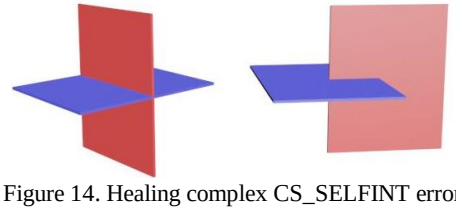

Figure 14. Healing complex CS_SELFINT error

CS_OUTEREDGE: Every edge of a solid will bound exactly two polygons. Any edge of the solid bounding less causes incorrect number of polygons with edge error and there is a hole somewhere in the solid. Firstly all the error edges are searched for loops. And new polygons are formed with the newly found loops. The incomplete parts of the loop are left out without healing.

CS_OVERUSEDEDGE: Any edge of the solid bounding more than two polygons causes a topological error. In figure 14 if the self-intersection error is healed then there will be an edge sharing 4 polygons. This type of error are not possible to heal automatically the possible options are to manually edit the solid or delete the polygons.

CS_FACEORIENT: Each edge must bound two polygon and the orientation of the edge must be opposite in the polygons. In figure 15 two polygons A {P_1, P_2, P_4, P_3, P_1} oriented by black arrows and B {P_6, P_4, P_3, P_5, P_6} oriented by red arrows are bound by the edge (P_4, P_3). But the order should be in one polygon (P_4, P_3) and in another polygon (P_3, P_4). If all or most of the edges of a polygon have wrong orientation then it is wrong oriented and healing would be to reverse the order of the pointlist. So if pointlist B is wrong oriented then the healing process will correct the pointlist in {P_6, P_5, P_3, P_4, P_6} and the orientation will look like the black arrows.

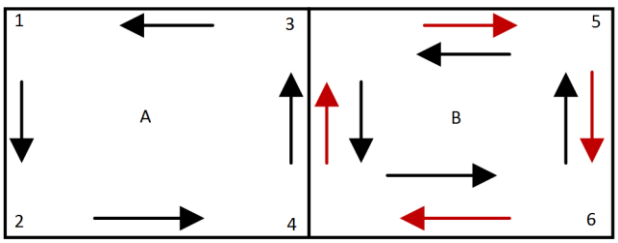

Figure 15. Healing CS_FACEORIENT error

CS_FACEOUT: Every surface normal of a solid must direct outwards. In figure 16 the red polygon has a face out error. Even after healing the face orientation error it is not guaranteed that all the polygons will face outward. Healing of this error is to reverse the orientation of the polygon. This healing is faster than the previous healing but

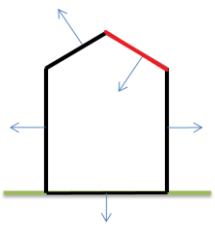

Figure 16. Healing CS_FACEOUT error

CS_CONCOMP: If a solid contains multiple disconnected solid then it will pass all the checks until connected component check. Healing of this error is to convert each disconnected solid into a solid data structure and delete the original solid.

CS_UMBRELLA: Healing this error is still in progress but one option is to split the adjacent polygons into groups where they are connected by edges and then create new vertices for each group with same coordinates then move those vertices a little bit away from each other like figure 17.

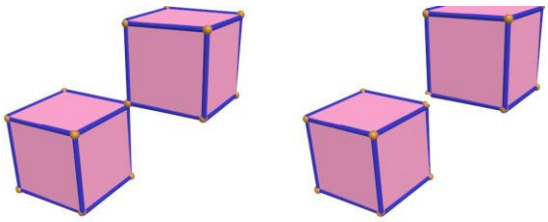

Figure 17. Healing CS_UMBRELLA error

4. RESULT AND DISCUSSION

Some real world models have been validated and healed using this tool. LOD1 and LOD2 Models of Stuttgart, Ludwigsburg, Dusseldorf and Rotterdam are some of those. An overview of the validation result is given in Table 1. All models are in LOD2. Here A represents 1 building with 580 polygons, J represents 61 buildings with 3455 polygons, L represents 4 buildings with 69 polygons and X represents 1922 building with 32546 polygons.

Check ID	A	J	L	X
CP-NUMPOINTS	0	0	0	0
CP-CLOSE	0	0	0	0
CP-DUPPPOINT	0	0	0	0
CP-SELFINT	0	0	0	0
CP-PLANDIST	4	0	4	177
CP-PLANDISTALL	8	0	4	161
CP-PLANTRI	67	0	5	269
CS-NUMFACES	0	4445	0	0
CS-SELFINT	0	-	2	4575
CS-2POLYPEREDGE	155	15484	0	467
CS-FACEORIENT	-	-	0	-
CS-UMBRELLA	-	-	0	116
CS-CONCOMP	-	-	1	980

Table 1. Results for geometry validation

In 16-99-HOOGVLIET-ZUID of Rotterdam model there are 10828 buildings with around 11000 ground, 23000 roof and 68000 wall surfaces. After validation with all geometric check approximately 118000 CS_OUTEREDGE, 33000 CS_SELFINT and 2000 CP_DUPPOINT error have been found in 10332 buildings like in figure 18. Only 496 buildings were found valid. 8474 buildings were healed after an iterative healing process. So 82% of invalid buildings were healed. Those building which couldn't be healed has been replaced by the original model. Although some of those errors can be healed but if still the building contains error then it is not possible to know by the automated process whether the process has minimized the error or made it more complex.

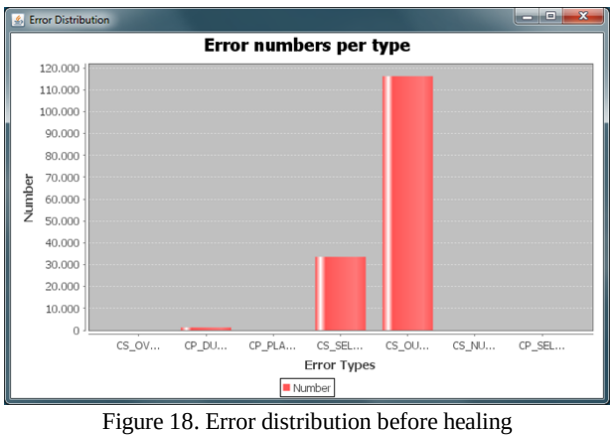

Figure 18. Error distribution before healing

So if there were 50 outer edge error and 49 edged has been found in different loops then still the remaining 1 edge will cause error and we wouldn't know those new polygons were correctly drawn or not until someone takes a look into it manually.

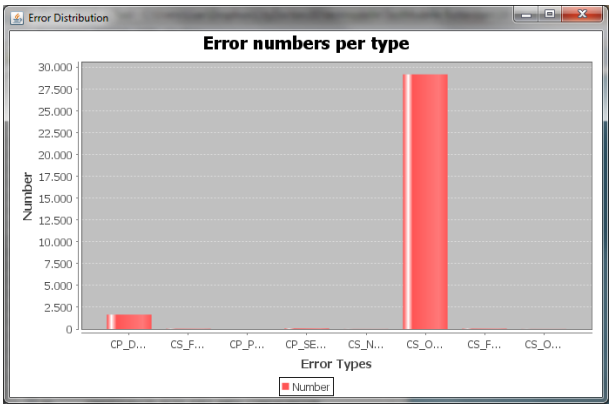

Figure 19. Error distribution after healing

There have also some difficulties while healing Rotterdam model. Some corner buildings of a series of houses were very strangely modeled like in figure 20. It has been modeled like two buildings joined together by a wall surface but the opposite walls of each building have been removed (like H shaped cross section). Those buildings couldn't be healed at this moment because the wall surface which lie in the middle, causes some over used edge error which are not possible to heal.

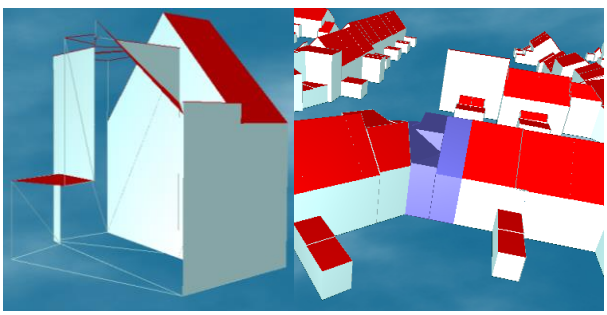

Figure 20. WallSurface splitting a building

Another type of error commonly found was some walls between two buildings out of nowhere like in figure 21. It is not clear that the walls should be modeled within the building or should have its own geometry in different building or building part. The buildings have complete structure without that wall. But together it creates same type of overused error which makes it difficult to heal.

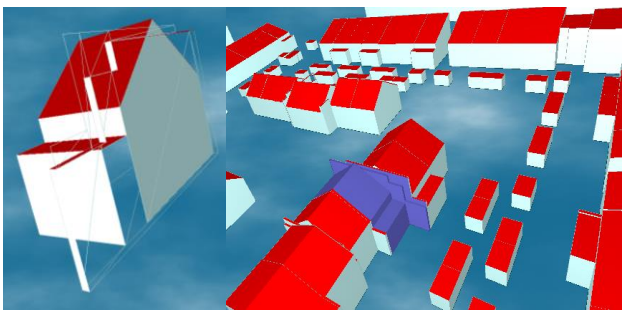

Figure 21. Extra WallSurface attached to the building

There has also been some overused error because of the structure of the building. Like in figure 22 height difference of roof has caused 4 polygons sharing an edge. There is nothing wrong with the modeling, the original building has been built like this.

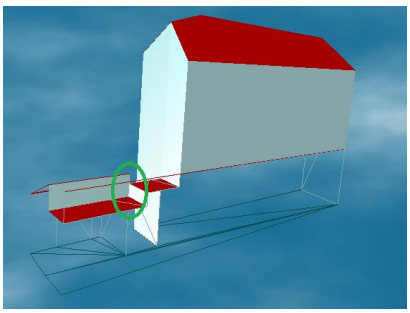

Figure 20. Two corner edges of two box shaped parts of a building touching each other at an edge.

5. CONCLUSION

All of the checks and most of the healing process has been already implemented and tested and the results have been discussed here. There are always new problems arising with new model. Mostly a model has similar type of errors in each building. This might come from a bug in the algorithm or overlooking a compulsory guideline while preparing those models. One thing to mention here is calculation time and system requirements. Normally the process works pretty fast but it depends upon how big the model is and how many iterations have been considered as limit, because if a building is not healable it will repeat the process equal to the maximum limit has been set. For Rotterdam model 4gb RAM was assigned because the process needs a lot of heap space, so depending upon the model size RAM can be assigned for it.

REFERENCE

Akca, D. et al., 2010. Quality assessment of 3D building data. In *The Photogrammetric Record*. pp. 339–355.

Campen, M., Attene, M. & Kobbelt, L., 2012. A Practical Guide to Polygon Mesh Repairing. In The Eurographics Association.

Krämer, M., Haist, J. & Reitz, T., 2007. Methods for Spatial Data Quality of 3D City Models. In Eurographics Italian Chapter Conference. pp. 167–172.

Kresse, W. & Fadaie, K., 2004. *ISO standards for geographic information*, Berlin-Heidelberg: Springer.

Wagner, D. et al., 2013. Geometric-semantical consistency validation of CityGML models. In J. Pouliot et al., eds. *Progress and New Trends in 3D Geoinformation Sciences*. Lecture Notes in Geoinformation and Cartography. 3D Geoinfo. Quebec, Canada: Springer Berlin Heidelberg.

Zhao, J. et al., 2012. Repair and generalization of hand-made 3D building models. In *Proceedings 15th Workshop of the ICA Commission on Generalisation and Multiple Representation.*