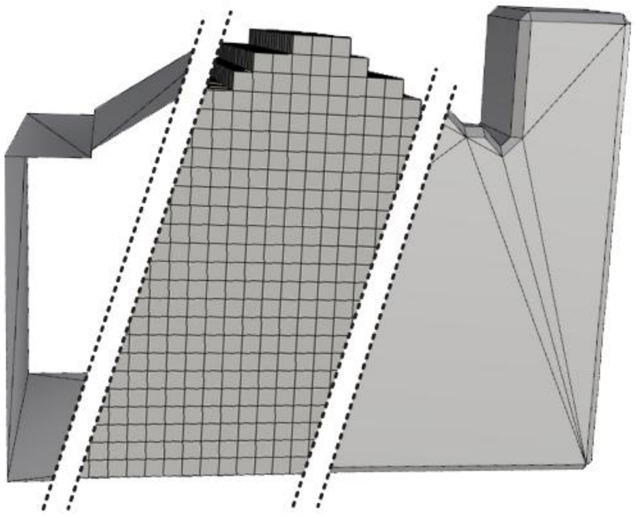
AUTOMATIC REPAIR OF 3D CITY BUILDING MODELS USING A VOXEL-BASED REPAIR METHOD



A thesis submitted to the Delft University of Technology in partial fulfillment of the requirements for the degree of

Master of Science in Geomatics

by

Damien Mulder

M.Sc Geomatics Thesis

June 2015

The work in this thesis was made in the:

3D geoinformation group
Department of Urbanism
Faculty of Architecture & the Built Environment
Delft University of Technology

Supervisors:	Prof.dr. Jantien Stoter
	Dr. Hugo Ledoux
Co-reader:	Ph.D. Pirouz Nourian

ACKNOWLEDGEMENTS

First of all I would like to thank Hugo Ledoux for his guidance on all aspects of the thesis throughout the entire project. By patiently explaining the basics and providing the relevant datasets and papers he enabled me to perform a focussed research from the very start. Without the tools for the triangulation, solidification and validation of the datasets (provided by Hugo Ledoux and Filip Biljecki) this thesis would not have been possible. In addition, his thesis layout and tips on presentation were of great help.
I would also like to thank Jantien Stoter for keeping an overview of the process and her comments that improved the clarity of the thesis. Her work on the development of a national 3D standard and the 3D Special Interest Group were motivating during the writing of this thesis as it showed the relevance of the topic.
Furthermore I would like to thank Ph.D. Pirouz Nourian, first of all for introducing me to the basics of computation geometry during my Architecture studies, which caused me to choose for the field of Geomatics. The knowledge provided in the 3D-modelling course was very useful in this thesis. When finally turning out to be my co-reader, his very detailed and useful comments helped to improve the most technical parts of this thesis.
I thank my fellow students for their help and making the Geolab a pleasant workspace, where we could have fun and learn from each other. Finally I would like to thank my girlfriend Godelief for listening to my voxel-related babbling and my family for their continuous support and encouragement.

ABSTRACT

Over the recent years, 3D city models have been gaining in importance. However, many defects are often present in the currently available datasets which hinders the further processing and analyzing of these models. Out of the existing repair methods, none are able to entirely repair all defects in 3D City Models. This thesis focuses on the process of transforming a defect 3D city building model into a valid one by using a voxel-based method. Two voxel-based repair approaches are proposed to repair 3D building models. Both methods start with a voxelization step which encompasses the conversion of a polygonal model to a binary volumetric representation. The second step consists of Morphological Operators which ensure the creation of a manifold voxel representation by filling up holes, selecting connected components and filtering certain voxels. The goal of this step is to simplify the topology. The third step then involves the extraction of the iso-surfaces, which is the main differences between the two methodologies. The first approach applies the Marching Cubes algorithm on a binary grid, resulting in a reliable surface reconstruction. The results are affected by rounding of the corners and a stair stepping effect. Therefore, the possibilities of using edge sharpening algorithms to sharpen the corners and applying the Marching Cubes algorithm on a signed distance field to recreate oblique surfaces are explored. In the second approach, some adaptation are proposed to the Dual Contouring algorithm to enable it to handle defect input models. By using a quadratic error function for computing vertex positions, oblique surfaces and sharp corners can be recreated in one step. In order to preserve any attributes of 3D city models throughout the repair process, three methods are proposed; (i) geometry based attribute determination, (ii) voxel-based attribute preservation and (iii) edge-based attribute preservation. Testing both approaches on a test set of existing 3D models shows that the repair capability of the first approach was quite reliable (90%). The second approach turned out to be less effective (30%), although its surface reconstruction was more accurate. Applying the first approach on the Heijplaat dataset from Rotterdam3D resulted in the percentage of valid building models improving from 10% to 93%. Although some more work is needed to reconstruct oblique surfaces and sharp corners in the first approach, it can be concluded that this kind of voxel-based repair method is capable of reliably repairing 3D city building models.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

ACRONYMS

AHN	Actueel Hoogtebestand Nederland National (=Height Model of the Netherlands)
BAG	Basisregistraties Adressen en Gebouwen (=Basic Register of Addresses and Buildings)
B-REP	Boundary Representation
COLLADA	Collaborative Design Activity
DC	Dual Contouring
DEM	Digital Elevation Model
DMC	Discretized Marching Cubes
DXF	Drawing Exchange Format
EMC	Extended Marching Cubes
GIS	geographical information system
GML	Geography Markup Language
ISO	International Organization for Standardization
IFC	Industry Foundation Classes
KML	Keyhole Markup Language
LOD	Level of Detail
MC	Marching Cubes
OGC	Open Geospatial Consortium
PCA	Principal Component Analysis
QEF	Quadratic Error Function
RMSE	Root Mean Square Error
SDI	Spatial Data Infrastructure
SHP	Shapefile
XML	Extensible Markup Language

1 INTRODUCTION

The recent years technology has emerged to incorporate the third dimension in the areas of geographical information system (GIS) and Spatial Data Infrastructure (SDI). During this process the concept of 3D city models has been gaining in importance as more 3D geometry of urban areas is becoming available. In these models the geometry and the semantics of real world objects such as terrain, buildings, infrastructure, vegetation and city furniture can be represented. This makes 3D City models usable for many applications such as urban planning [Chen, 2011], disaster management [Kolbe et al., 2008], augmented reality [Zamyadi et al., 2013] and navigation systems [Cappelle et al., 2012]. In general four types of 3D GIS analysis are distinguished by Moser et al. [2010]. Proximity & overlay analysis involves the using 3D distance, buffer and intersection calculation. Spread & flow analysis uses simulations of physical processes in the real world. 3D density data can be used to analyze for instance population distribution, which is useful for emergency planning or mobile phone coverage. Visibility analysis locates positions that are visible from a specific point. Applications of this may involve sight lines or shadow analysis or finding the optimal position for an antenna. An example of spread & flow analysis is the studying of noise effects on the environment. de Kluijver and Stoter [2003] describe the importance of noise effect studies in 3D GIS to monitor noise levels. Other examples of spatially related questions which may be answered by analyzing a 3D City model may be:

- Which buildings and/or floor levels are affected by a 2 meter flooding of the city?
- During which times will a certain location be in the shadow?
- How many square meters of facade do these buildings have?

An example of 3D City Model analysis is shown in Figure 1 where the yearly solar radiation upon rooftops is calculated [Biljecki et al., 2015].

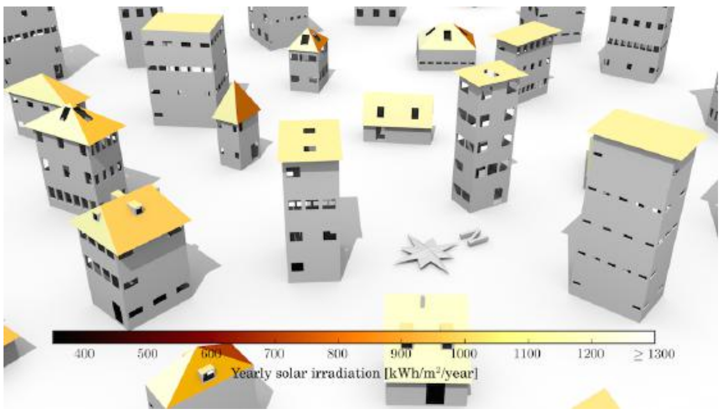


Figure 1: Solar3Dcity: An example of 3D City Model analysis [Biljecki et al., 2015]

These kinds of tools have a great potential to enhance the workflow and decision making in many working fields. A prerequisite is that the built environment has to be accurately modeled into a computer understandable data format. Three ways of generating 3D City models are described in [Singh et al., 2013]. The first technique is described as conventional and involves using vector maps (such as cadastre data) and height points. The second technique makes use of high resolution satellite images. The third technique applies close range photogrammetry on terrestrial images. An automatic generation process using a Digital Elevation Model (DEM) in combination with Cadastre data is proposed by Durupt and Taillandier [2006]. The buildings roof geometry is created by comparing height point data to a set of predefined roof shapes. A method which may yield higher precision is the procedural creation of 3D City models using point clouds, which is described by Lafarge and Mallet [2012]. A multitude of efficient ways of creating of 3D City models is currently present. However, defects may be present in these models due to manual errors, model optimization, data conversions, misinterpretation of the data standard or unexpected cases in the applied algorithms. Figure 2 shows two examples of incorrectly modeled building models.

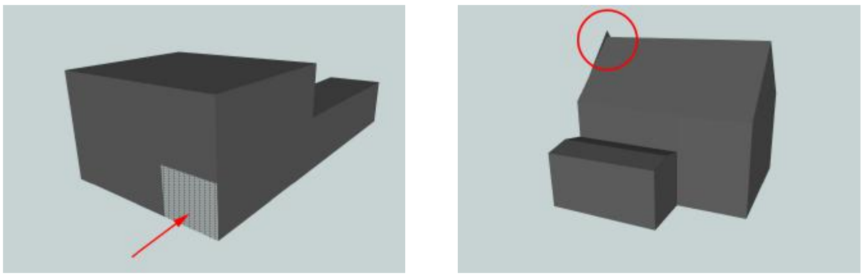

(a) Hole in building model **(b)** Overshoot in building model

Figure 2: Examples of visible defects in a 3D City model

Although 3D City models offer a structured way of storing geometry, in practice a significant amount of geometry is not considered valid. Geometric defects that may occur are self-intersections, folding, invalid holes, wrong orientation or non-2-manifold solids. This hinders the further analyzing or processing of these models, for instance computing the volume or performing boolean operations. Therefore the use of valid geometry is essential to be able to make use of the benefits of 3D City Models. Validation processes have been developed in order to check a dataset for some of these formats. An example of such a validation process for the CityGML data standard (see § 2.1.2 for a description) has been created based on the work of [Ledoux, 2013]. A web service running this process is available at `http://geovalidation.bk.tudelft.nl/val3dity`, enabling users to check their datasets. In case datasets are not completely valid, repair methods are needed to restore the geometry.

1.1 OBJECTIVES

Since manual repair of 3D City models is very time consuming and prone to errors, automatic repair methods are highly desirable. However, none of the existing repair methods are able to entirely repair all defects in 3D City Models (see Section 2.2), therefore additional research into alternative repair methods is required. This thesis focuses on the process of transforming a defect 3D City building into a valid one by using a voxel-based method. The main research question for this thesis is:

To which extent is it possible to automatically repair a geometrically invalid 3D City Building Model using a voxel-based method?

In Chapter 3 the 3 components of a voxel-based method are distinguished. Figure 3 shows an overview of these components.

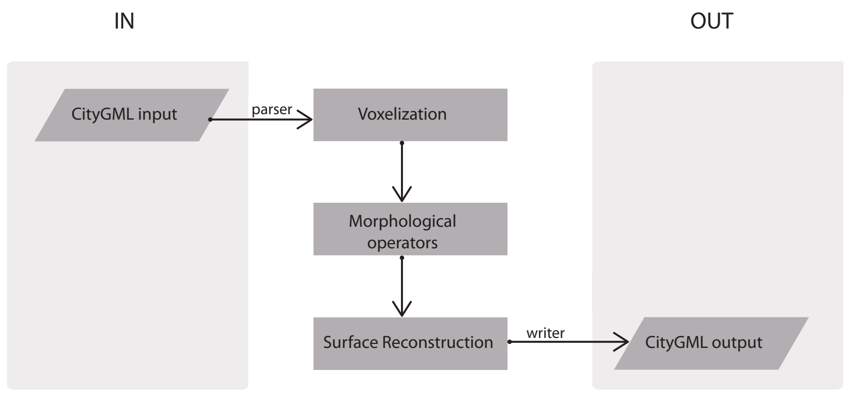


Figure 3: Components of methodology

To explore the possibilities of a voxel-based repair approach, the following sub-questions are relevant:

- What are the most common errors of invalid 3D City models and can they be repaired by using a voxel-based repair method?
- Which voxelization algorithm is most suitable for converting a polygonal model while preserving any semantics which may be present?
- Which surface reconstruction method is most suitable for rebuilding the sharp geometry characteristics of 3D city buildings from a voxelization?
- What are the advantages and disadvantages of using a voxel-based repair methods when compared to existing repair methods?

This thesis proposes two voxel-based repair methods to demonstrate the potential and illustrate the obstacles and/or shortcomings of a voxel-based approach. By implementing and testing these methods, more insight in the potential of a voxel-based repair method could be gained.

1.2 REQUIREMENTS

In order to conduct the development of a repair prototype and evaluate the existing repair methods, some requirements have been set for the ideal 3D City model repair method.

- The repair method should return valid 3D city model given a (realistic) input model.
- There should be no significant geometric differences with the original model. In case of defect input geometry, realistic assumptions should be made.
- If any attributes are present in the input model, these should be present in the repaired model as well. Since the surface attributes are initially lost in the voxelization process, a way of preserving or restoring these attributes should be investigated.
- The method should preferably run automatically without user intervention.

The decisions for applying certain algorithms in this thesis, are made with these requirements in mind.

1.3 TEST DATASETS

For the development and testing of the developed prototypes, various 3D City Model datasets were used. Using different datasets is important since particular defects are encountered. Four sources of city models were used:

- The first dataset on which the developed prototypes have been tested is Rotterdam3D which was published in 2011 as open data at: `http://www.rotterdam.nl/links_rotterdam_3d`. The dataset contains 90 neighborhoods and has been automatically built up by combining cadastre footprints from the Basisregistraties Adressen en Gebouwen (=Basic Register of Addresses and Buildings) (BAG) and the Actueel Hoogtebestand Nederland National
(=Height Model of the Netherlands) (AHN). CityGML 1.0 was used as data standard and all buildings are modeled in LoD 2. An illustration of the neighborhood Nieuwe Werk of the Rotterdam3D dataset is shown in Figure 4.
- The second dataset which was used is from Montreal, Canada. It was published in 2013 as CityGML 1.0 in LoD 2. This dataset differs from Rotterdam3D since it was created using photogrammetry, which is noticeable since there is more detail in the models. The datasets is available at: `http://donnees.ville.montreal.qc.ca/dataset/`.
- Another set of models which has been used is the unit test set. These models were used in the development of val3dity, a tool for the geometric validation of GML solids created by [Ledoux, 2013]. They are available at: `https://github.com/tudelft3d/val3dity/tree/master/data/poly`
- Additionally, some manual models have been created in order to check specific cases of geometric input.

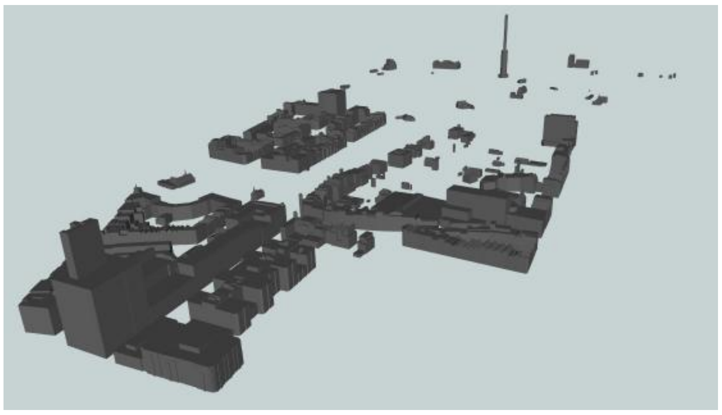


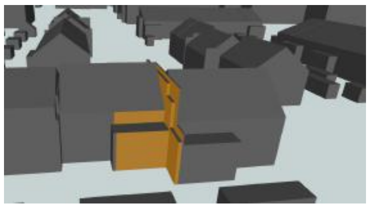
Figure 4: Neighborhood 'Nieuwe Werk' from the Rotterdam3D CityGML dataset

Some statistics show that the validity of the Rotterdam dataset is very low. For example, 10.335 out of 10.828 buildings have defects in the Hoogvliet-Zuid dataset. Furthermore it is obvious that the Montreal dataset has a better quality of data, as it contains less defects while storing more detailed geometry. The number of polygons and valid buildings for several datasets is shown in Table 1.

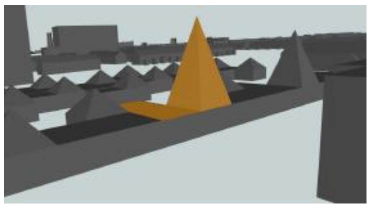
	Buildings			Faces	
	defect	total	%	total	per building
Rotterdam Hoogvliet-Zuid	10 335	10 828	95	100 195	9,3
Rotterdam Overschie	3 048	3 318	92	44 319	13,4
Rotterdam Heijplaat	1 091	1 207	90	13 738	11,4
Montreal VM01 2009	62	384	16	84 759	220,7
Montreal VM02 2009	21	209	10	32 973	157,7
Montreal VM03 2009	68	339	20	64 440	190,1

Table 1: Details concerning the test datasets

Apart from a high number of defects, some peculiar geometry can be seen in the dataset of Rotterdam. Some thin walls are modeled in between row houses (see Figure 5a), which is not necessarily considered as defect. Also some very steep roofs are modeled (see Figure 5b), where these are not to be expected.



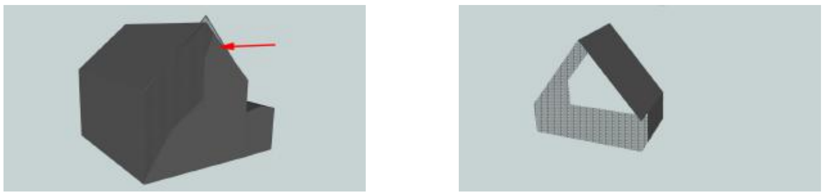
(a) Building model with unrealistic thin walls



(b) Building model of a row house with an overly steep roof

Figure 5: Missing walls in between buildings

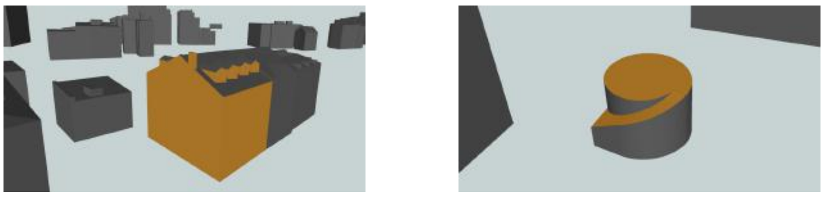
However, these observations are not relevant since the focus is on model validity. A set of validation rules which will be used in this thesis is shown in Table 2, in § 2.1.5 of Chapter 2. In this regard, some of the described defects are very specific for this dataset. Whenever buildings are connected (e.g. row houses) the intermediate walls are usually not modeled in the dataset of Rotterdam. In addition, sometimes these walls are partially modeled but added to the neighboring building. This results in many non-closed buildings, making it impossible to compute the volume of a building which is necessary for tax purposes as described by Boeters [2013].



(a) CityGML model with overshoot **(b)** CityGML model with two gaps

Figure 6: Examples of defects in existing datasets

The Montreal dataset on the other hand, does not show many obvious defects. Figure 7a shows the amount of detail which is typical for the Montreal dataset. A defect model with a non-closed shell is shown in Figure 7b. Usually, non-closed shells are only the case in more complex geometry such as the rounded object shown in Figure 7b.



(a) CityGML model with detailed geometry **(b)** Rounded CityGML model with non closed shell

Figure 7: Examples of defect buildings from the Montreal dataset

1.4 SCOPE OF THIS THESIS

The main objective of this thesis is to repair geometrically invalid 3D City model building using a voxel-based method. Using such a volumetric repair method offers a robust repair capability compared to the existing repair methods. Any surface-oriented or graph-based repair steps will not be at the core of the developed methods. Only where necessary, some surface-oriented steps (for example Edge Sharpening, see § 4.3.3) have been researched.

An overview of the general workflow and the role of 3D City model repair is shown in Figure 8. This thesis does not deal with the initial creation of defects in 3D City models nor with their consequences for further processing or analyzing of the data. The focus will be on maintaining any sharp features of the input geometry which are characteristic for buildings, meaning that smooth polygonal models (e.g. Standford bunny) are not taken into consideration. Also any building formats which allow for inner shells (e.g. CityGML LoD 4) are outside the scope of this research.

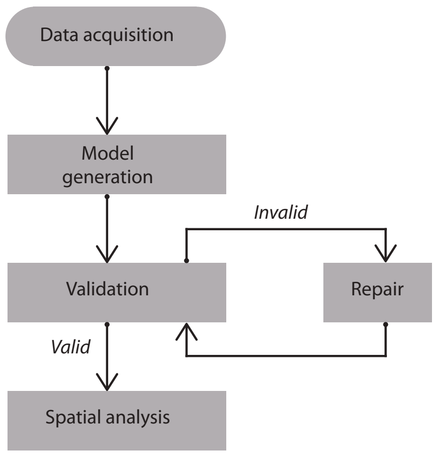


Figure 8: The different stages in the general 3D city model workflow

1.5 OUTLINE OF THIS THESIS

Chapter 2 starts by describing the validation rules of solids in 3D City models. This is followed by an overview of CAD-repair methods and continues to evaluate the currently existing methods for repairing 3D City models. This Chapter concludes that existing methods can heal large percentages of defect building models. However, none of them are totally satisfactory when repairing certain defects.

Chapter 3 presents the relevant background theory about a voxel-based repair method, separated in three components; Voxelization of a polygonal mesh, Morphological operators and Surface reconstruction. The topological repair of a voxelization by scan conversion along with morphological operators is illustrated. Subsequently, an overview of surface reconstruction algorithms is presented of which Marching cubes, Dual contouring and Pressing are described in more detail. Here a clear distinction is made between using only a binary grid versus using the original input model in addition. Based on this chapter, the two alternative approaches have been formed.

Chapter 4 describes the choices made in the implementation of the two approaches (Approach 1: Marching Cubes and Approach 2: Dual Contouring). The decisions that were made for both approaches are explained per component and specific issues that have been encountered are highlighted.

Chapter 5 explains the selection of a test set and consecutively shows the results of applying a voxel-based repair on the test set, where the performance is measured by validity and geometric error.

Chapter 6 starts with a more general discussion section, where the role of a voxel-based method is placed in the context of general 3D City model processing. This is followed by the conclusion, stating the fundamental repair capability of the researched method but also emphasizing the difficulties of an accurate surface reconstruction. Finally several recommendations for future work will be given.

2 STATE OF THE ART IN 3D CITY MODEL REPAIR

Some research on repairing 3D City models has been performed. However, to be able to speak of model repair, first the specifications of a valid model will be described in Section 2.1. This will be followed up in Section 2.2 by a description of the existing approaches, both in City modeling and other disciplines. An evaluation of the effectiveness of the approaches is given in Section 2.3.

2.1 3D CITY MODEL STANDARDS

In this section the specifications for 3D modeling standards (CityGML in particular) will be described along with existing tools for validation of datasets and an overview of the possible errors.

2.1.1 3D standards

Several ways of storing 3D City models are currently in existence. The most common 3D standards are

- CityGML by Open Geospatial Consortium (OGC)
  url: `http://www.citygmlwiki.org/index.php/Specifications`
- x3D by web3DConsortium
  url: `http://www.web3d.org/x3d/what-x3d`
- Industry Foundation Classes (IFC) by buildingSMART International
  url: `http://www.ifcwiki.org/index.php/Documentations`
- Geography Markup Language (GML) by OGC
  url: `http://www.opengeospatial.org/standards/gml`
- Keyhole Markup Language (KML) by OGC
  url: `http://www.opengeospatial.org/standards/kml/`
- COLLADA by Khronos Group
  url: `https://www.khronos.org/files/collada_spec_1_5.pdf`
- Drawing Exchange Format (DXF) by Autodesk
  url: `http://images.autodesk.com/adsk/files/acad_dxf0.pdf`
- Shapefile (SHP) by ESRI
  url: `https://www.esri.com/library/whitepapers/pdfs/shapefile.pdf`
- 3D PDF by Visual Technology Services
  url: `http://www.3dpdfconsortium.org/pdf-standards-info.html`

Based on a set of criteria such as geometry, texture, semantics and acceptance among others, CityGML has been adopted in 2011 as the main format of the Dutch 3D SDI [Stoter et al., 2011]. The main advantages of CityGML over the other formats is that it's the only international standard for 3D City Models that covers both the semantics and geometry. For this reason many 3D city models are available in CityGML. Therefore, CityGML will be the main focus for the validation and repairing processes described in this thesis.

2.1.2 CityGML as 3D standard

To enable the widespread use and reuse of 3D city models, a common format for storage and exchange is needed. For this reason the OGC has developed CityGML as an open standard with the goal of creating a common definition for the objects, attributes and relations of a 3D city model. The most important characteristics of CityGML are multi-scale representation, coherent semantic-geometrical modeling and geometric topological modeling [Kolbe et al., 2005]. The multi-scale representations of CityGML are developed as five Levels of Detail LoD, describing the scale of the object geometry (see Figure 9). For each of these LoD representations, geometry can be stored.

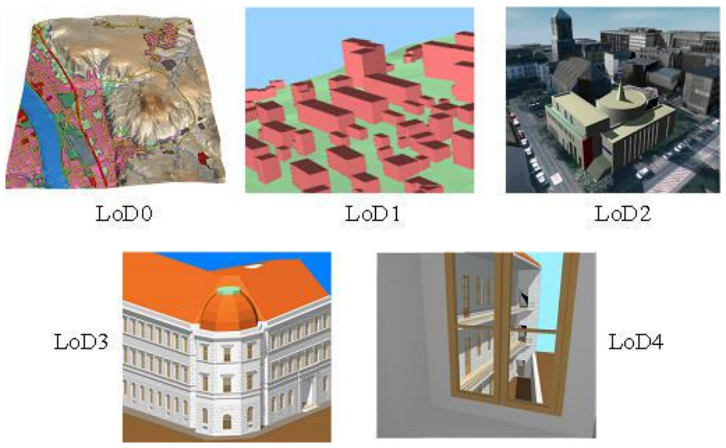


Figure 9: The 5 LoDs in CityGML [Kolbe et al., 2005]

2.1.3 Semantic modeling of CityGML

Another property of CityGML is the coherent semantic-geometrical modeling Stadler and Kolbe [2007]. Objects in the real world are represented both on the geometric and the semantic level, which are linked. This means that for example a city object can be stored as a *Building*, *Bridge*, *City Furniture*, *Water Body*, *Transportation*, etc. This thematic modeling enables the possibility of adding further semantics to the object. In case of a Building, surfaces can be grouped together as either *Ground Surface*, *Roof Surface* or a *Wall Surface*. Furthermore, individual surfaces may store attributes, for example the class *Textured Surface* enables the storage of appearance properties such as material, texture or transparency. These links between geometry and semantics is one of the advantages CityGML has over the other 3D standards.

2.1.4 Geometry specifications of CityGML

CityGML is based on GML3 which uses the ISO19107 geometry model [Kolbe, 2009]. Geometric primitives exist for each dimension. A zero-dimensional primitive is a point, a one-dimensional primitive is a curve, a two-dimensional primitive is a surface and a three-dimensional primitive is a solid. Examples of these primitives are shown in Figure 10.

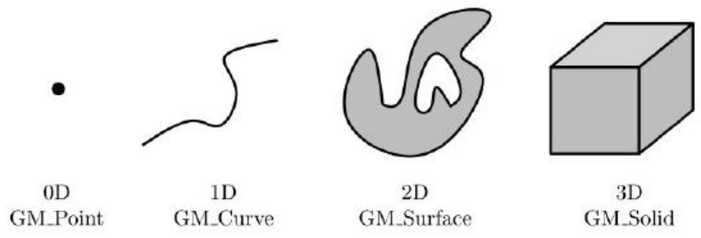


Figure 10: ISO 19107 geometric primitives [Ledoux, 2015]

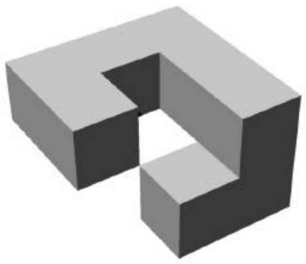
Solids in CityGML are described by using a Boundary Representation (B-Rep), this means the higher dimensional primitives are topologically built up of the lower dimensional primitives. Polygons are built up of one exterior ring and may have multiple interior rings, which represent holes in the polygon. A surface is then built up from a set of polygon faces. The collection of such surfaces can be regarded as a boundary representation which represents a solid. An example of how such a solid is stored in GML is shown in Figure 11. In this example only a single Linear Ring is shown, for the sake of clarity.

```
<gml:Solid>
  <gml:exterior>
    <gml:CompositeSurface>
      <gml:surfaceMember>
        <gml:Polygon>
          <gml:exterior>
            <gml:LinearRing>
              <gml:pos>84468.66733  429533.28100  0.00000</gml:pos>
              <gml:pos>84470.71000  429539.03000  0.00000</gml:pos>
              <gml:pos>84479.52000  429535.91000  0.00000</gml:pos>
              <gml:pos>84477.47000  429530.15000  0.00000</gml:pos>
              <gml:pos>84468.66733  429533.28100  0.00000</gml:pos>
            </gml:LinearRing>
          </gml:exterior>
          <gml:interior>
           ...
      </gml:surfaceMember>
    </gml:CompositeSurface>
  </gml:interior>
</gml:Solid>
```

Figure 11: Example of a gml:Solid

2.1.5 Validation of CityGML

The requirements for valid three-dimensional primitives as defined by the ISO 19107 geometry model. Based on this definition an implementation for the validation process has been described by Ledoux [2013]. The definition states that solids should be closed or watertight and each shell should be simple, thus a 2-*manifold*. This means that for every point on a surface, the neighborhood should be topologically equivalent to a plane. Since shells should be 2-manifold, no dangling pieces, gaps or overlaps are allowed. Figure 12a shows an example of a non-manifold solid since it is touching itself on a single vertex, which is called a *singular point*. Several faces share this vertex but none of them share an edge. Another example of a non-manifold solid is shown in Figure 12b where more than two faces share the same edge.


(a) Non-manifold vertex

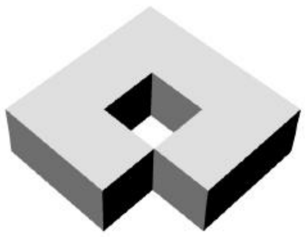

(b) Non-manifold edge

Figure 12: Self-intersecting ring

A solid may be perforated, as long as the remaining volume remains watertight. Since all surfaces are represented by polygons, their geometry should be planar. All holes are described by an exterior ring. If holes are present in the polygon, these are represented as interior rings. In order to distinguish between exterior and interior rings, any exterior polygons of a solid should be ordered counter-clockwise. Vice versa, interior rings should be ordered clockwise. An example of a valid polygon containing a hole is illustrated in Figure 13.

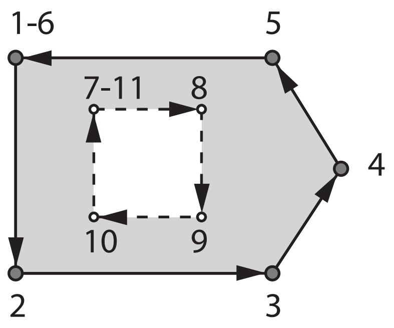


Figure 13: Valid polygon with a hole described as interior ring

A process is proposed by [Ledoux, 2013] where the validation takes place on four levels; (i) per ring, (ii) per polygon, (iii) per shell and (iv) per solid. If defects are found on one level, the next levels are not checked anymore as the model will be considered invalid. A description of these errors for each level, will be described below.

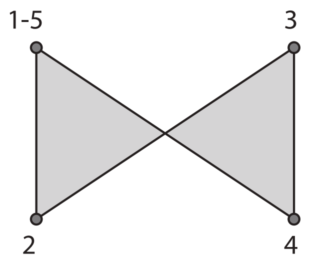
RING LEVEL

In general every ring should be closed and consecutive points are not allowed to be the same. An example of this is shown in Figure 14. When the distance between the highlighted coordinates is smaller than the threshold, a defect will be the result.

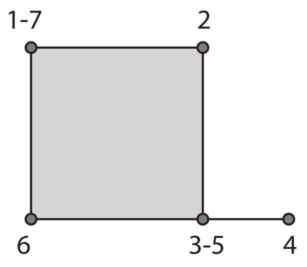
```
...
  <gml:LinearRing>
     <gml:pos>300607.4669  5040070.0728  22.8400</gml:pos>
     <gml:pos>300606.0106  5040070.5849  23.4520</gml:pos>}
     <gml:pos>300606.0106  5040070.5849  23.4516</gml:pos>
     <gml:pos>300607.4669  5040070.0728  22.84</gml:pos>
  </gml:LinearRing>
...
```

Figure 14: Example of Consecutive Points Same error

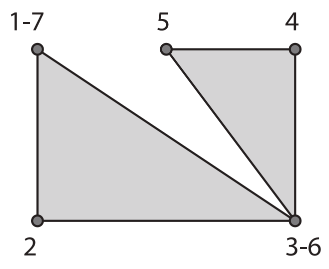
Furthermore, all vertices should be incident to two edges and edges are not allowed to be intersected. Figure 15a shows an example of an invalid self-intersecting ring. In Figure 15b the vertices 3 and 5 have the same position and are connected to multiple edges. Figure 15c illustrates another example of vertices being connected to multiple edges, also resulting in a self-intersection defect. Figure 15d shows a polygon where the distance between two vertices is lower than the specified threshold, resulting in a consecutive points same error.



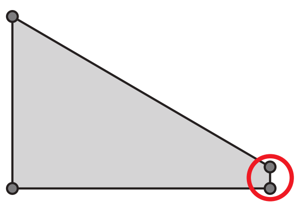
(a) Self-intersecting ring



(b) Self-intersecting ring



(c) Self-intersecting ring

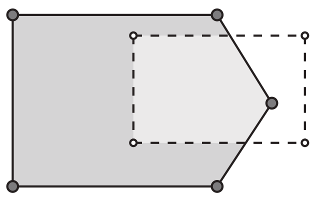


(d) Ring with consecutive points same

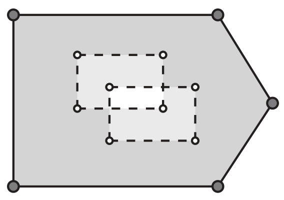
Figure 15: Self-intersecting ring

POLYGON LEVEL

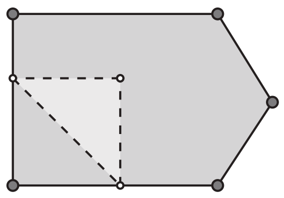
On the polygon level, defects can be caused by either the interior rings or non-planarity. Every point in the interior of a polygon, should be connected to any other point in the interior. Additionally, interior rings are not allowed to intersect with the exterior ring or other interior rings, and should be oriented counter-clockwise. Some examples of invalid rings are shown in Figure 16. Figure 16a shows an example of the interior ring being located outside the exterior ring. Intersecting rings are visible in Figure 16b. The interior of the polygon shown in Figure 16c is disconnected, resulting from the fact that the hole splits the polygon in two pieces. A polygon with a wrong interior ring orientation is shown in Figure 16b.



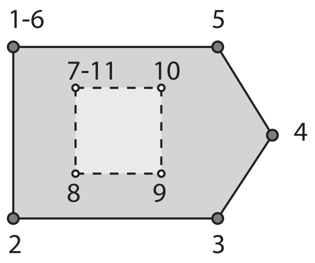
(a) Interior ring intersecting with exterior ring



(b) Interior rings intersecting each other



(c) Polygon interior is not connected



(d) Interior ring with wrong orientation

Figure 16: Some examples of invalid polygons

Another defect on the polygon level is shown in Figure 17, where the polygonal is non-planar. All triangle normals of the triangulated polygon may not deviate more than a specified threshold angle. In this example, an angle of 90°will be found which is too high.

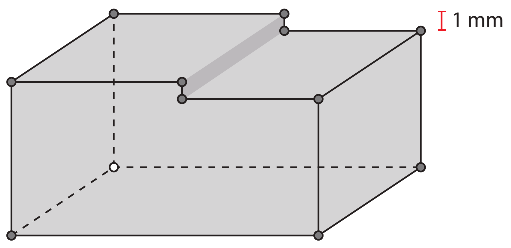
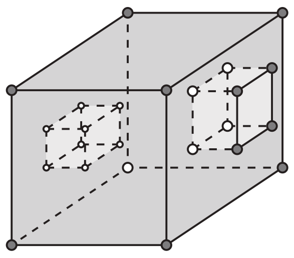


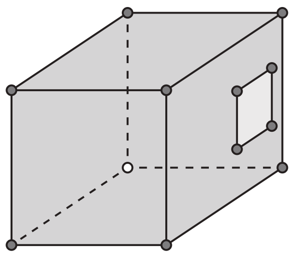
Figure 17: Example of a non-planar polygon distance deviation

SHELL LEVEL

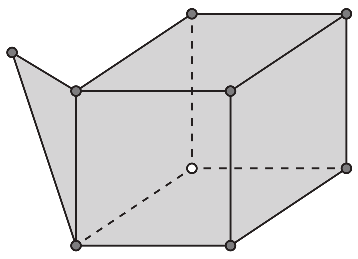
Once the checks on the lower levels have been accomplished, the shell can be validated. The first example shown in Figure 18a shows a valid solid, containing an exterior shell and an interior shell. Figure 18b shows a gap in the model, disabling the distinction between interior and exterior. An example of a non-manifold edge is shown in Figure 18c. Related to this, a non-manifold vertex is illustrated in Figure 18d.



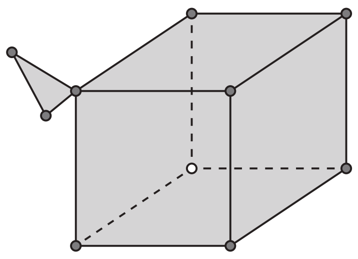
(a) A valid Solid



(b) Invalid shell: shell not closed



(c) Invalid shell: non-manifold edge



(d) Invalid shell: non-manifold vertex

Figure 18: Example of a valid shell and three invalid ones

SOLID LEVEL

Defects on the solid level involve conflicts between the exterior shell and one or more interior shells. Figure 18a shows a solid with an interior shell. Since the scope of this thesis is restricted to 3D City models with no interior shells (i.e. CityGML LoD $1-3$), defects at the solid level are not relevant. An overview of the possible errors that may be found in the data is in Table 2, based on the work of Ledoux [2013].

	Defects val3dity
Ring	Too few points
	Consecutive points same
	Not closed
	Self intersection
	Collapsed to line
Polygon	Intersection rings
	Duplicated rings
	Non planar polygon distance plane
	Non planar polygon normals deviations
	Interior disconnected
	Hole outside
	Inner rings nested
	Orientation rings same
Shell	Too few polygons
	Not closed
	Non manifold vertex
	Non manifold edge
	Multiple connected components
	Self intersection
	Polygon wrong orientation
	All polygons wrong orientation

Table 2: An overview of the defects detected by val3dity [Ledoux, 2013]

2.2 CURRENT REPAIR METHODS

This Section will start by giving an overview of general methods for repairing polygon meshes followed by an evaluation of the existing work on 3D City Model repair. Two existing methods for repairing 3D City models are described (see § 2.2.3 and § 2.2.4) along with a description of the potential of a voxel-based repair method (see § 2.2.5). Then the strengths and weaknesses of these three repair methods will be evaluated.

2.2.1 Polygonal mesh repair

Polygonal mesh repair in CAD-models is a widely studied topic, often applied in the fields of finite element analysis and mechanical engineering. Although not aimed at repairing building models specifically, these methods can be applied on city models. An extensive overview of polygonal mesh errors and repairs has been presented by [Attene et al., 2013]. Here a clear distinction is made between local and global repair methods.

- Local repair methods are defined as handling defects on an individual basis using local modifications. These methods are characterized by leaving the original mesh untouched where possible, however a repaired or manifold output is often not guaranteed.
- Global or volumetric repair methods are described as using a complete re-meshing resulting in robust method which may loose detail.

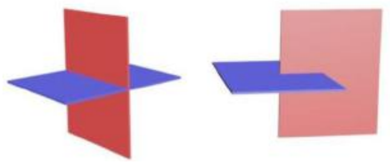
2.2.2 City Model repair

Since building models have different characteristics such as large planes and sharp features, the desired result is different than in most CAD-repair methods. Several descriptions on the approach of repairing 3D city models have been published. A classification of the common defects in CityGML is given by Zhao et al. [2014]. It is stated that these defects may arise as a result of the interactive modeling process, data optimization, conversion of CAD models or the adding of semantics. Basic repair routines are proposed: triangulation, regularization removing dangling pieces and decomposition based on intersections of the model. An alternative set of rules for validation is given by Wagner et al. [2013] addressing the geometric-semantical consistency of CityGML models.

Two initiatives show results of applying repair methods on datasets. The first done by Alam et al. [2013] describes a local method which iterates through all polygons and solids, and checks each for a number of polygon and solid specific errors. Their results are described in § 2.2.3. The second case is done by [Zhao et al., 2013] using a method called Shrink-wrapping. This method is based on using a constrained tetrahedralization, which will be described in more detail in § 2.2.4.

2.2.3 Detect & local repair

An attempt of using local mesh-repair techniques on defective CityGML models is described by [Alam et al., 2013], of which the process is still ongoing. Two sets of validation checks are created, both for the polygon and shell level. Their method then proceeds to test building models for specific errors, first on the polygon level then on the shell level. If any defects are encountered, local repair techniques are applied aiming to fix the defects.


(a) Self-intersections

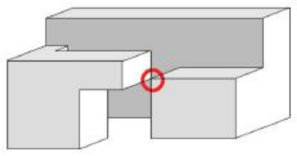

(b) Non-manifold vertex

Figure 19: Some limitations of the detect and local repair method [Alam et al., 2013]

On the polygon level, defects such as self-intersection, consecutive points same can be repaired. On the shell and solid level multiple connected components, simple self-intersections (overlapping polygons) and face-orientations can be repaired. A major problem on the solid level are overused edges (edge bounding more than 2 faces) which are considered impossible to repair automatically. Furthermore, any 'complex' self-intersections (see Figure 19a on the shell level may be solved but are often transformed into the same problematic overused edge error. Also overshoots (edge bounding less than 2 polygons) are not necessarily repaired. The repair for any solids touching in a point (umbrella) has not yet been implemented (see Figure 19b). They conclude by stating that different datasets often contain different types of defects due to a possible misinterpretation of the required modeling standards. Although this method does not repair all defects, it does leave the surface semantics intact. In case of non-planar polygons, the assumption is made that all walls are vertical and roof edges are parallel. Depending on the building model, this may not be a realistic representation. See Figure 20 for an illustration of the non-planar polygon repair step.

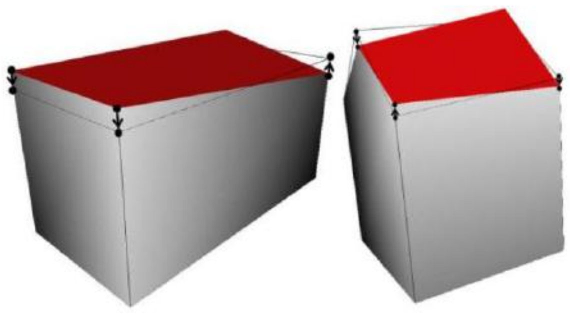


Figure 20: Healing of non-planar Roof Surfaces in LoD1 & LoD2 [Alam et al., 2013]

The findings of this paper may be seen as an indication that local repair methods are efficient and fast at repairing some defects but do not seem suitable for defects on the solid level involving overshoots or self-intersections. An overview of the strengths and drawbacks of this method are given in Table 3.

2.2.4 Shrink-wrapping

A second repair method called *Shrink-wrapping* was developed by [Zhao et al., 2013]. It is a graph-based method which applies a constrained tetrahedralization on all faces of the model and its convex hull. This results in a volumetric representation made up of polyhedra, of which some have to

Detect & local repair

Strengths	Drawbacks
Efficient in terms of processing	Non-manifold edges not repaired
Preservation of attributes	Self-intersections not repaired
Oblique surface are supported	Assumptions on parallelism
	Non-manifold vertex not repaired

Table 3: Strengths and drawbacks of detect & local repair method

be removed. To do so a heuristic carving process is performed, deleting polyhedra which are outside of the original model. An illustration of this process is illustrated in Figure 21.

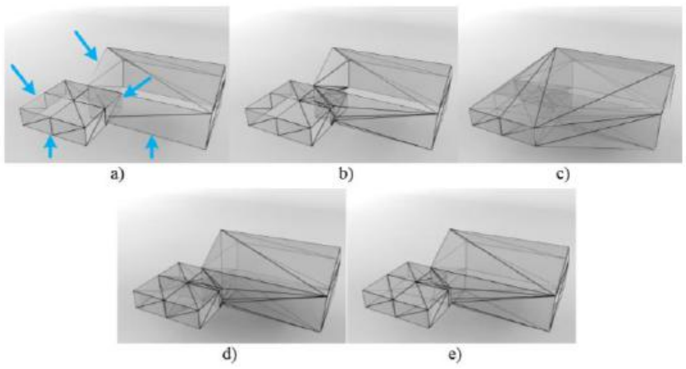


Figure 21: Illustration of the Shrink-wrapping method [Zhao et al., 2013]

Validating this process shows that it can effectively repair gaps and self-intersections in LoD2 CityGML datasets. Also any interior holes can be filled. A drawback of this method is that overshoots do cause problems. Although the resulting building may be considered valid, the geometry has not been accurately repaired. An example of a valid model but not an accurate repair is illustrated in Figure 22. In addition the Shrink-wrapping method is sensitive to floating point arithmetic. The rounding of point precision may result in triangles being decomposed into edges or vertices, leaving the building model invalid. Finally, the occurrence of overlaps in the input model may result in small errors.

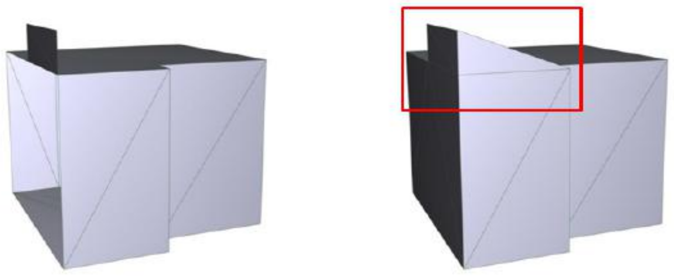


Figure 22: Incorrectly repaired overshoot [Zhao et al., 2013]

Shrink-wrapping also leaves the surface semantics intact and maintains the original vertex positions of the input geometry. An overview of the strengths and drawbacks of this method are given in Table 4.

Shrink wrapping

Strengths	Drawbacks
Effective repair of gaps	Overshoots can be incorrectly repaired
Effective repair of self-intersections	Sensitive to floating point arithmetic
Preservation of attributes	Overlapping surface cause small errors
Oblique surface are supported	
No geometry shifts (except overshoots)	

Table 4: Strengths and drawbacks of Shrink-wrapping method

2.2.5 Voxel based repair

Using a voxel-based repair method takes a different approach to polygonal model repair, corresponding to the global methods described by [Attene et al., 2013]. By performing the intermediate step of converting the input model to a binary grid, a manifold result can be acquired by using morphological operators and disregarding inner shells. Existing research on this approach is given by [Nooruddin and Turk, 2003] and [Hétroy et al., 2011].

Although the use of voxel-based methods for repair of 3D City models has been suggested, the specific problems which may be encountered when applying these methods on 3D City building models have not been researched in depth. One issue is that the semantics will be lost initially. A second issue is the preservation of characteristics of 3D City Model Buildings such as large planar regions and sharp edges. These features are usually lost when using the above mentioned methods. However, more suitable results may be achieved by selecting methods which take these characteristics into account. The possibilities of applying a voxel based reconstruction method on invalid CityGML geometry will be explored in this thesis. The theory of applying a voxel based repair method will be described in the next Chapter 3. An overview of the expected strengths and drawbacks of a voxel based method are given in Table 5.

Voxel based

Strengths	Drawbacks
Robust repair capability	Potential shift in geometry
Potential to repair non-manifold edges	Potential loss of attributes
	Inherently slower processing

Table 5: Strengths and drawbacks of a potential voxel based method

2.3 EVALUATION

When we compare the two existing repair methods to the requirements set in Section 1.2 in Chapter 1, it becomes clear that both the Local Repair and the Shrink-wrapping method are quite effective but do not repair all defective cases. While some defects may be more efficiently repaired by the existing methods, certain defects such as overshoots and other cases of non-manifold edges seem to be a particular challenge. The third method encompasses a voxel-based repair and has the potential to repair cases of non-manifold edges and severely defective geometry. Whereas both Shrink-wrapping and the Detect & Local Repair method do preserve attributes and support oblique surfaces without any problems, this is challenging for a voxel-based method. The possibilities of preserving attributes and smoothly reconstructing oblique surfaces will be studied, to find out the suitability of a voxel based method.

3 RELATED WORK

Several voxel-based repair methods for general polygonal mesh repair are already in existence. In general, the existing work on voxel-based repair can be distinguished in 4 components, which form the basis of the two methodologies which I propose later in this thesis (see Chapter 4).

- The first component is Voxelization of a polygonal mesh Section 3.2 which has the goal of turning a (defect) 3D polygonal model into a binary grid.
- The second component is Morphological operators (Section 3.3) which has the goal of filtering artefacts and closing any holes, ensuring a manifold binary grid.
- The final component is Surface reconstruction (Section 3.4) which has the goal of turning binary grid into polygonal mesh using isosurface extraction techniques.
- Additional steps are described in Post processing (Section 3.5) which have the goal of turning the mesh into a polygonal model with sharp features.

This chapter will start by giving an overview of the related work. Consecutively, the separate components which are at the basis of these methods will be described.

3.1 OVERVIEW RELATED WORK

One of the earlier methods of voxelization is presented by Oomes et al. [1997] describing a process of transforming a 3D-polygonal model into a voxel representation using a flood filling algorithm. It was developed as a means of converting between polygonal and volumetric models. Since it was not meant for repair it does not allow for any gaps in the input geometry.
A process for topology reduction and surface simplification was described by Andújar et al. [2002]. Apart from a voxelization process they go on to describe two alternatives for surface reconstruction; a triangular and an orthogonal reconstruction. Although this method covers the full process of voxelization and surface reconstruction, the fact that the mesh is purposefully simplified and gaps cannot be handled make this method unsuitable for 3D City repair.
A well known contribution for a voxel-based mesh repair was presented one year later by Nooruddin and Turk [2003], specifically proposing a way to repair polygonal models using a voxelization consisting of *Parity Count* & *Ray Stabbing*. Some morphological operators are offered in order to remove holes. Finally, they filter the binary grid before converting it back to triangle mesh (marching cubes like) and using edge decimation techniques for reducing the number of triangles. Although this method does not maintain

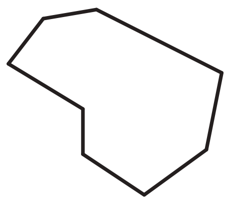
sharp building features, its repair capability makes it very relevant for 3D city model repair.
According to Bischoff and Kobbelt [2005], the existing repair methods can be distinguished as being either Surface- or volume- based algorithms, corresponding with the local & global repair methods described in § 2.2.1. They propose a hybrid model, detecting defect regions (intersections and cracks) based on a threshold specified by the user. Then they apply a voxel based repair only on those parts of the mesh. The valid parts of the mesh are left untouched. A surface reconstruction combining Marching Cubes and Dual Contouring is applied before applying smoothing and decimation. A strong point of this method is its hybrid approach. However, the connection (clipping) between the original and the repaired part of the mesh may leave artefacts. Furthermore the applied smoothing and decimation may remove sharp features, although these can be customized.
A new and original approach is described by Hétroy et al. [2011], which gives the user control over the local application of the topology repair. A voxelization is applied by finding all voxels that intersect with the bounding box of mesh triangles, which was compared to [Nooruddin and Turk, 2003] and considered equally effective. After the voxelization morphological operators are applied for closing holes before using an extension of the Marching Cubes algorithm (preventing ambiguous cases) and smoothing to rebuild the triangle mesh. Again, this method is not aimed at reconstructing building models as the smoothing steps will not produce sharp feature. Both hybrid methods [Bischoff and Kobbelt, 2005; Hétroy et al., 2011] require user input which does not fully meet the requirement of an automatic repair.
A recent paper considering the voxelization was published during the development of this thesis by Steuer et al. [2015]. They apply the voxelization method of [Nooruddin and Turk, 2003] on (defect) City Models in order to approximate the volume. They describe how their algorithm has the capability of calculating volumes of defect City models and may have an impact on building model repair. Existing methods provide ways of adjusting topology and returning smooth manifold triangle meshes with as little artefacts as possible (oversampling, smoothing, filtering) These methods are not perfect for building model repair but their components may be used in combination with surface rebuilding and post-processing algorithms that are better suited. The rest of this chapter will describe the work on each of the defined components.

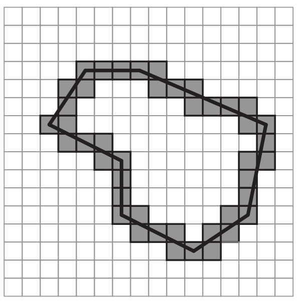
3.2 VOXELIZATION OF A POLYGONAL MESH

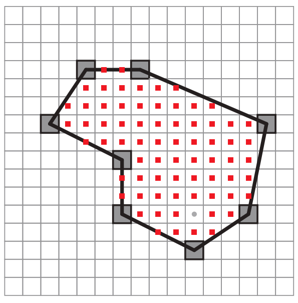
The voxelization component encompasses the conversion of a polygonal model to a binary volumetric representation. The goal of this component is to enable the simplification of topology while avoiding the creation of unwanted artefacts. First the basics of vector to raster conversion and the principle of using a 3D grid are explained. Then the theory of scan converting a 3-dimensional polygonal model will be clarified. Subsequently the topological repair capability of scan conversion will be described.

3.2.1 Vector to raster conversion

The principle of raster conversion of a vector model is not new. [Kaufman and Shimony, 1987] were the first to describe this process for application on a 3-dimensional model. They describe a set of scan-conversion algorithms for 3D geometric objects such as straight line segments, polygons, cylinders and cones. The basics of this process are described by Jones [2014, Ch. 8, p. 132-137] showing how the boundaries of a polygon can be approximated by transforming them to pixel locations. There are two ways of doing so; (i) rasterizing a B-Rep or (ii) rasterizing the interior of a B-Rep. Figure 23a shows the input vector model of which the B-Rep is rasterized in Figure 23b, showing all pixels which intersect the original vector model. For rasterizing the interior of a B-Rep, first all pixels are selected which intersecting multiple edges (see Figure 23c). Once these corner pixels are found, these can be connected to form a polygon. Now all pixels within the polygon are shown. This result is shown in Figure 23d. Where the original vector model was only limited by the precision of its coordinate system, rasterizing a polygonal model results in a raster model with a lower precision. This precision is closely related to the resolution of the raster.


(a) Vector model


(b) Rasterizing a B-Rep


(c) Select and connect corner pixels

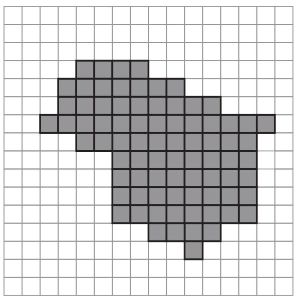
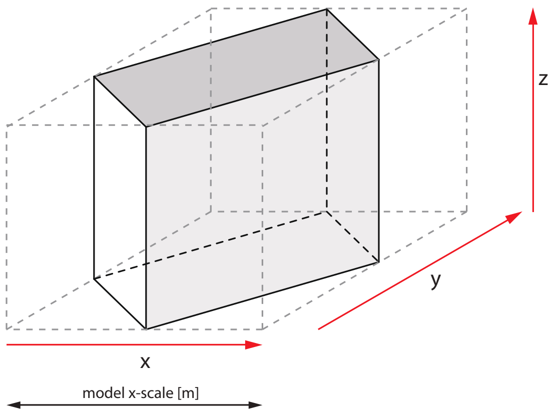

(d) Rasterizing the interior of a B-Rep

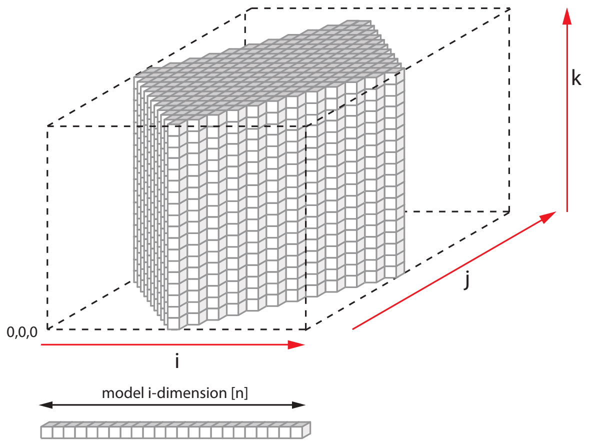
Figure 23: Vector to raster conversion

In order to rasterize a polygonal model in 3D, the same principles apply but have to be expanded. Since our goal is to reconstruct the volume of a building, we are looking for a way to rasterize the interior of a B-Rep in 3D. To do so, we have to fill a 3D grid of cells. Similar to the rasterization process, values should be assigned to every voxel; 0 to voxels in the exterior of the model and 1 to voxels in the interior of the model.
The 3D grid should contain the entire polygonal model, meaning that the bounding box of the 3D-model should correspond with the bounding box of the 3D-grid. An illustration of a 3D model in real world coordinates

(x,y & z) with its bounding box and scale is shown in Figure 24a. The voxelized version of this model is shown in Figure 24b, illustrating the voxel coordinates (i, j & k), the origin at $(0, 0, 0)$ and the dimension of the grid.



(a) Representing a 3D model in real world coordinates ($\mathbb{R}^3$)



(b) Representing a volume in voxel coordinates ($\mathbb{Z}^3$)

Figure 24: Illustration of the 3D model and voxelization in different coordinate representations

During the repair process, we need to be able to convert back and forth between the real world coordinates and the voxel coordinates. In order to relate the two different coordinate representations, four parameters are required;

- The *scale* defines the size of the grid which is determined by the largest length of the model in either the x, y or z direction.
- The *dimension* defines the resolution of the grid. It describes the number of voxels needed to fill the length of the model scale.
- The *voxel size* defines the dimensions of a grid cell.
- The *translation* values describe the distance of the lower left corner of the first voxel $V_{0,0,0}$ to the lower left corner of the original models' bounding box. This is needed to switch between V_{ijk} in voxel

coordinates ($\mathbb{Z}^3$) to V_{xyz} in the original model coordinates ($\mathbb{R}^3$). An illustration of the translation vector is given in Figure 25.

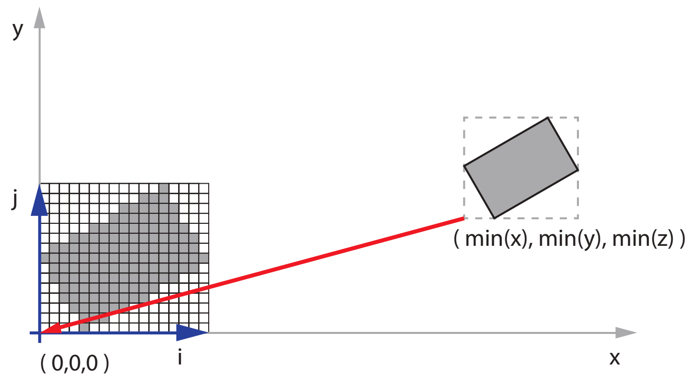


Figure 25: Illustration of the translation vector

The conversion of voxel coordinates to original coordinates is described by Equation 1.

Retrieving original model coordinates:

$$x = scale_x \cdot \frac{(i + 0.5)}{dimension_x} + translate_x \tag{1a}$$

$$y = scale_y \cdot \frac{(j + 0.5)}{dimension_y} + translate_y \tag{1b}$$

$$z = scale_z \cdot \frac{(z + 0.5)}{dimension_z} + translate_z \tag{1c}$$

In order to determine the resolution of a voxelization of a certain model, either the dimension or the voxel size should be given as input. Assuming that cubic voxels are used with the same dimension in the x,y, and z direction, this relation can be described as:

$$scale_x = dimension_x \cdot voxelsize \tag{2a}$$

$$scale_y = dimension_y \cdot voxelsize \tag{2b}$$

$$scale_z = dimension_z \cdot voxelsize \tag{2c}$$

The next step is to transform a 3D polygonal model into a binary grid, which can be done with a scan conversion.

3.2.2 Scan conversion

Scan conversion is a well tested method for determining whether a voxel should be considered inside or outside a polygonal model. This is decided by shooting scan rays through the polygonal model. Two methods of scan conversion are proposed by Nooruddin and Turk [2003]; *parity count* and *ray stabbing*. Although both methods base the voxel value on the position of scan line intersections, their decision is made in a different way. Parity Count considers a voxel to be inside when it is positioned after an odd number of intersections, see algorithm 3.1.

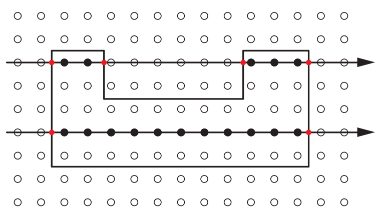
Algorithm 3.1: PARITY COUNT [Nooruddin and Turk, 2003]

```
Input: A triangulated model $M$, an empty 3-dimensional grid $G$ and
       a ray directed $\perp xy$, $\perp yz$ or $\perp zy$
Output: binary voxel grid $G$ with assigned values 0 (exterior) and 1
        (interior) for every voxel $G_{ijk}$ along ray $r$

1 calculate intersections between $r$ and $M$
2 for every $G_{ijk}$ along $r$ do
3     if odd number of intersections before $G_{ijk}$ then
4         return $G_{ijk} = 1$
5     end
6     else
7         return $G_{ijk} = 0$
8     end
9 end
```

Algorithm 3.2: RAY STABBING [Nooruddin and Turk, 2003]

```
Input: A triangulated model $M$, an empty 3-dimensional grid $G$ and
       a ray $r$ along the grid
Output: binary voxel grid $G$ with assigned values 0 (exterior) and 1
        (interior) for every voxel $G_{ijk}$ along ray $r$

1 calculate intersections between $r$ and $M$
2 for every $G_{ijk}$ along $r$ do
3     if intersection before and intersection after $G_{ijk}$ then
4         return $G_{ijk} = 1$
5     end
6     else
7         return $G_{ijk} = 0$
8     end
9 end
```

Ray stabbing considers a voxel to be inside when it is positioned anywhere between the first and last intersection along the scan line, see algorithm 3.2. The difference between these methods is illustrated in Figure 26. Whereas Parity Count functions as an on/off switch, Ray Stabbing fills all space between the first and last intersection.


(a) Parity Count: all intersections taken into account

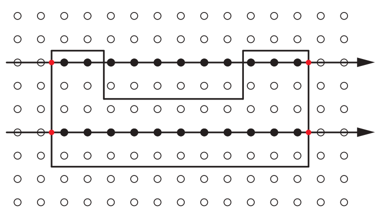
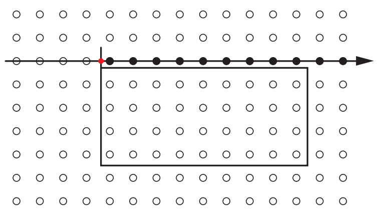

(b) Ray Stabbing: first and last intersection taken into account

Figure 26: Different results in scan converting a concave polygon

This example shows why the Parity Count method is more precise, as it accurately converts a concave model. The reason why Ray Stabbing may be relevant is because possible defects in the input polygonal model should be considered. How defect input models may be converted to topologically correct binary volumes has to do with the concept of majority voting.

3.2.3 Majority Voting

The defects which are most significant during the scan conversion process are *gaps*, *overshoots* and *self-intersections*. Figure 27 shows that these defects prevent single scan rays to correctly determine the interior and exterior of a polygonal model.


(a) Overshoot: effect on single scan-line

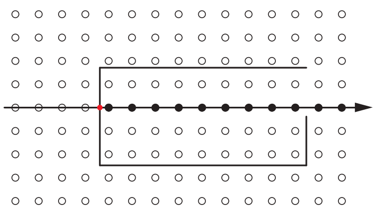
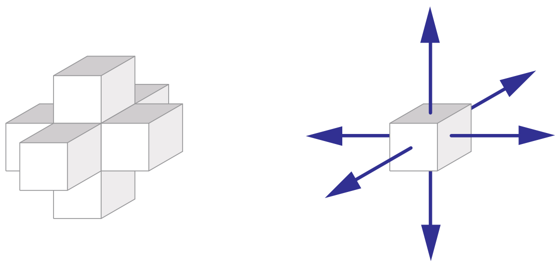

(b) Gap: effect on single scan-line

Figure 27: Scan conversion of defect input using Parity Count

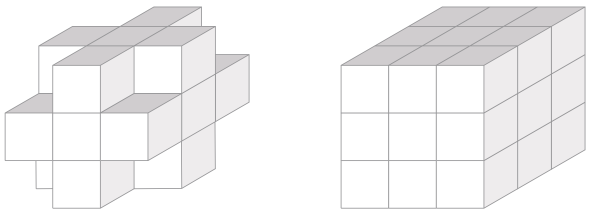
If a polygonal model would be scanned in one direction, any existing defects may cause wrongly assigned values to the voxel grid. The way to solve this as proposed by Nooruddin and Turk [2003] is to use a *majority voting*. The concept is to shoot multiple rays in different directions, and let each of these rays *vote* for a value (1 for interior or 0 for exterior) in the voxel grid. A majority voting then means that any voxel with more than a certain amount of votes, will be considered to be inside the model. In this process two parameters are of importance:

- the number of rays in different directions
- the voting threshold

In order to achieve a symmetrical composition of rays, using either 6, 18 or 26 rays is most logical. Figure 28a shows how scanning in 6 directions is done towards neighboring voxels in the x,y and z direction. Similarly, scanning in 18 or 26 directions is possible by selecting more neighboring voxels, as shown in Figure 28b. Using a higher number of rays will possibly result in a more precise voxelization at the cost of processing time. The suitability will be described in more detail in Chapter 4 in Section 4.2.
The importance of the voting threshold is illustrated in Figure 29. All voxels are colored according to the number of votes. Voxels with 1 vote are shown in grey, voxels with 2 votes are shown in black. By only including voxels with a value of 2, the overshoot Figure 29a and the gap Figure 29b can be topologically repaired. This principle is important, since it is the reason why a voxel based method is able to handle certain defects.
The voting threshold determines how loose or strict the voxelization process is performed. Nooruddin and Turk [2003] leave open the option for different thresholds. However, Steuer et al. [2015] choose to scan in 6 directions and



(a) Shooting in 6 directions



(b) Shooting in 18 (left) or 26 directions (right)

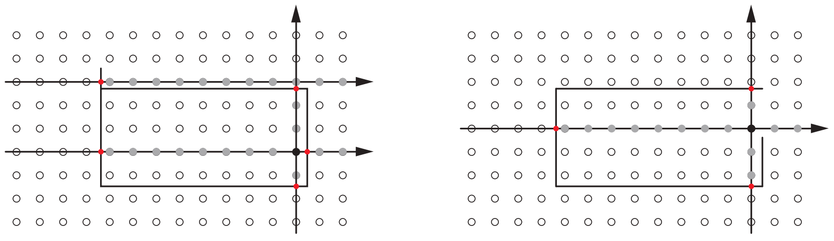
Figure 28: Number of scan directions per voxel

Algorithm 3.3: MAJORITY VOTING
[Nooruddin and Turk, 2003]

Input: A triangulated model M, an empty 3-dimensional grid G and n number of scan directions $\overrightarrow{dir_n}$ and a threshold t
Output: a filled 3-dimensional binary voxel grid G

```
1  for voxel $G_{ijk}$ in $G$ do
2      votes = 0
3      for every $\overrightarrow{dir_n}$ do
4          ray $r = V_{xyz} + \overrightarrow{dir_n}$
5          if ParityCount returns 1 then
6              votes += 1
7          end
8      end
9      if $votes > t$ then
10         $G_{ijk} = 1$
11     end
12     else
13         $G_{ijk} = 0$
14     end
15 end
16 return $G$
```

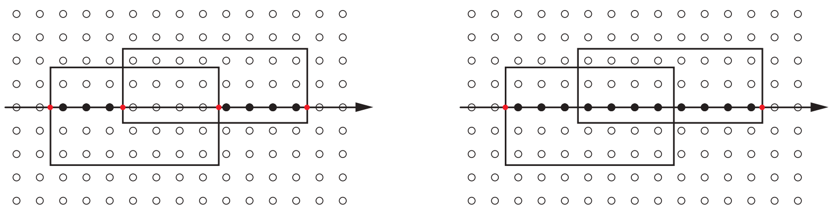
set the threshold t at more than half the number of rays, $t = r_n/2$. In their approach this leads to a threshold $t = 3$. However, i.e. a voxelization in 6 directions is performed, a threshold $t = 3$ or $t = 4$ may yield different results. In some (defect) cases this can lead to the output being either manifold or not. The specific cases were studied and are presented in Section 4.2.



(a) Scan converting an overshoot (b) Scan converting a gap

Figure 29: The effect of using majority voting on defect input geometry

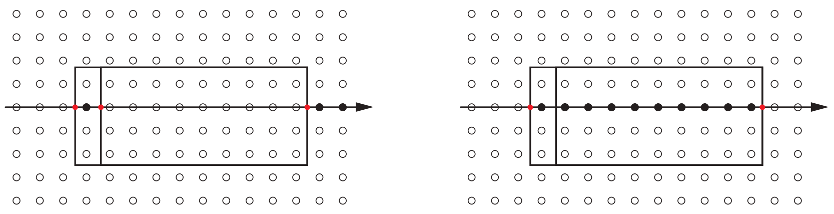
In general the Parity Count method is expected to result in correct decisions more often, due to its support for concave geometry. However, the Ray Stabbing method may solve cases of self-intersecting models and double surfaces. Examples of these cases are illustrated in Figure 30 and Figure 31. The self-intersecting case shows how the Parity Count method stops filling voxels in the overlapping area since a second intersection was found. The scan-line continues to fill voxels when the overlapping area is left. Running this process for a single scan-line will result in a gap.



(a) Parity Count: all intersections taken into account (b) Ray Stabbing: first and last intersection taken into account

Figure 30: Advantage of Ray Stabbing in case of self-intersections

In case of a double surface, the Parity Count method immediately finds two intersections. This means that voxels won't be filled until the third intersection is found. When the actual exterior is reached, this will be mistaken for the interior. Running this process for a single scan-line will result in multiple connected components.



(a) Parity Count: all intersections taken into account (b) Ray Stabbing: first and last intersection taken into account

Figure 31: Advantage of Ray Stabbing in case of double surfaces

In general we can observe that any self-intersections or double-surfaces in the input model, may cause the Parity Count method to return an undesired result for individual scan rays. However, by using majority voting these cases can be repaired in general. Only in extreme cases of multiple defects in combination with self-intersections or double-surfaces, this problem may result in a disordered topology. Only in these cases Ray Stabbing may be more appropriate.

3.2.4 Down-sampling

An optional step of smoothing the resulting binary grid which is used by Nooruddin and Turk [2003] is a down-sampling of the grid. They propose a $3x3x3$ Gaussian filter but mention that a $2x2x2$ filter may also be suitable. In the latter case a 128^3 resolution grid for example, would be turned into a 64^3 resolution voxelization, turning every 8-cube into either a single 0- or 1-valued voxel. Since we are looking at building model repair, we want to maintain any sharp edges. Therefore the suitability of smoothing is highly dependent on the characteristics of the input model and the applied surface reconstruction.

3.2.5 Repair capability

The previously described process has the ability to turn most defect polygonal models into a manifold volumetric representation. However, the resulting binary grid is not guaranteed to be manifold. In case of combinations of self-intersections, double surfaces or extreme defects, the resulting voxelization can contain holes and/or dangling voxels. There are three aspects of the voxelization which may affect the quality of the final output:

- the number of connected components
- any holes in the voxelization (possibly caused by self-intersections or double surfaces)
- any unwanted hanging voxels (possibly caused by sharp angles in the input model)

Some morphological operators are needed to process these cases. How to detect and handle each of them, will be discussed in the following Section 3.3.

3.3 MORPHOLOGICAL OPERATORS

Since our goal is to create a topologically correct voxelization which represents a 2-manifold, some checks need to be performed after the voxelization process. These aspects were identified in the previous Section as (i) selecting the relevant connected component(s), (ii) detecting and removing any holes and (iii) detecting and removing unwanted hanging and/or dangling voxels which may result in unwanted artefacts. An overview of solutions for these problems will be given in this Section.

3.3.1 Adjacency Filtering

The most basic of morphological operations on binary grids are based on the notion of adjacency. This is described by Cohen-Or and Kaufman [1995] and allows the identification of any hanging or dangling voxels.

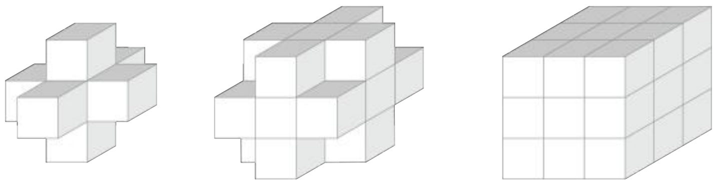


Figure 32: 6-, 18- and 26- adjacency

Each voxel in a binary grid has a number of voxels within its N-adjacency. The possible 6-, 18- and 26- adjacency have been illustrated in Figure 32. Any voxels that share a face are 6-adjacent, voxels that share an edge are 18-adjacent and voxels that share a vertex are 26-adjacent. By calculating the number of N-adjacent voxels, information regarding the position of single voxels can be retrieved. For example a voxel with 26/26-adjacency is fully covered and in the interior of a volume. A voxel with 1/6-adjacency is connected to a volume through one face (a hanging voxel). Based on this knowledge, filtering and selecting of voxels can be applied. An example of filtering any hanging voxels is shown in algorithm 3.4. Different operations can be performed by setting a different rule for the adjacency threshold and voxel value.

Algorithm 3.4: VOXEL ADJACENCY [Cohen-Or and Kaufman, 1995]

Input: A binary grid G containing voxels V_{ijk} with values 1 (interior) or 0 (exterior), a specified N-adjacency $(6, 18 or 26)$ and a threshold t

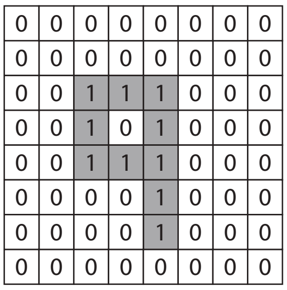
Output: An integer grid G containing voxels V_{ijk} with values $0 to N$

```
1 for $V_{ijk}$ in $G$ do
2     calculate number of N-adjacent voxels $n$
3     if $n < t$ then
4         $V_{ijk} = 0$
5     end
6 end
```

3.3.2 Distance Mapping

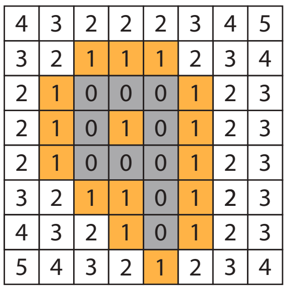
Distance mapping is a morphological operator with the goal of enlarging or removing holes, which is used both in [Nooruddin and Turk, 2003] and [Hétroy et al., 2011]. Both of these methods use a slightly different way of a double buffer. This can be done in two ways, called *opening* and *closing*. Since we are interested in removing holes, the closing is a relevant step.

Figure 33a shows the initial grid with a hole (in this case a single missing voxel). The first step is to create the Manhattan Distance map of the binary grid, which is described by Danielsson [1980]. This is an efficient algorithm which calculates Manhattan distance values for each 0-valued voxel in the grid to the closest 1-valued voxel. Naturally, all 1-valued voxels are given a distance value 0. The next step is to select all voxels with a Distance map value lower than a threshold. This operation can be considered as an outward buffer, which is called *dilation* in this context. Figure 33b shows all voxels that were selected by the outward buffer in orange. Note that the gap has been included in the dilation. Now to recreate the original volume, an inward buffer is needed, which is called *erosion*. To do so, all voxels which were not selected during the dilation are given the value 0, for all other values the Manhattan distance to the closest 0-valued voxel is again computed. Now, using the same threshold, an inward buffer can be performed This inward buffer is illustrated in orange in Figure 33c. Finally, by inversing the grid and thus selecting all voxels with a distance value above the threshold, the initial volume is retrieved. Note that the hole has been removed but also the concave corner has been rounded off (see Figure 33d).



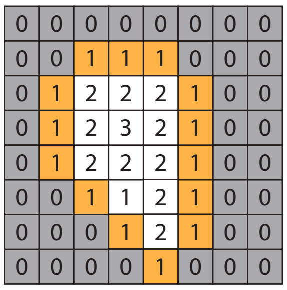
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	1	1	1	0	0	0
0	0	1	0	1	0	0	0
0	0	1	1	1	0	0	0
0	0	0	0	1	0	0	0
0	0	0	0	1	0	0	0
0	0	0	0	0	0	0	0

(a) Initial grid with a hole



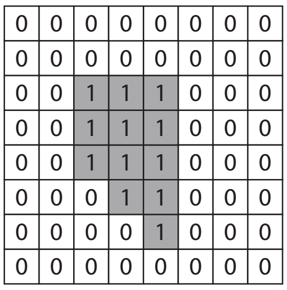
4	3	2	2	2	3	4	5
3	2	1	1	1	2	3	4
2	1	0	0	0	1	2	3
2	1	0	1	0	1	2	3
2	1	0	0	0	1	2	3
3	2	1	1	0	1	2	3
4	3	2	1	0	1	2	3
5	4	3	2	1	2	3	4

(b) Buffer on distance map



0	0	0	0	0	0	0	0
0	0	1	1	1	0	0	0
0	1	2	2	2	1	0	0
0	1	2	3	2	1	0	0
0	1	2	2	2	1	0	0
0	0	1	1	2	1	0	0
0	0	0	1	2	1	0	0
0	0	0	0	1	0	0	0

(c) Buffer of the inversed distance map



0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	1	1	1	0	0	0
0	0	1	1	1	0	0	0
0	0	1	1	1	0	0	0
0	0	0	1	1	0	0	0
0	0	0	0	1	0	0	0
0	0	0	0	0	0	0	0

(d) Final output grid without hole

Figure 33: Performing a closing operation by using Manhattan distance mapping

These methods were proposed for general polygonal models which often have smooth characteristics. When we apply this method on a building model, indeed the holes smaller than buffer distance d are removed. However, any sharp concave corners which are often present in building models will be rounded off, making this operation unsuitable.

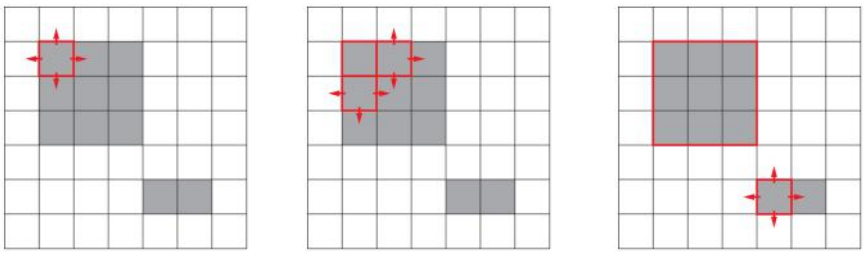
3.3.3 Connected Components

Before applying the surface reconstruction on the binary grid, it must be ensured that the number of disjoint sets (= connected components) in the grid is one. This is important because multiple connected components will result in a polygonal model which is considered invalid. When multiple connected components are found, is it possible that the building model represents an aggregated model, or it could be severely defect. Either way, a decision will have to be made regarding the separation of the connected components. Knowledge about the connected components also gives us another way of detecting and removing any holes by looking at the inverted grid. A flood filling algorithm such as described by Burtsev and Kuzmin

Algorithm 3.5: CLOSING OPERATION
[Danielsson, 1980]

```
Input: A distance map represented as an integer grid $G$ containing
       voxels $V_ijk$ with a distance value $v$, and a buffer distance $d$
Output: A binary grid $G$ with no holes smaller than $d$

1  calculate distance map $D_1$ on $G$
2  for $V_ijk$ in G do
3      if $V_ijk$ has $v < d$ then
4          $V_ijk = 0$
5      end
6      else
7          $V_ijk = 1$
8      end
9  end
10 calculate distance map $D_2$ on updated $G$
11 for $V_ijk$ in G do
12     if $V_ijk$ has $v > d$ then
13         $V_ijk = 1$
14     end
15     else
16         $V_ijk = 0$
17     end
18 end
```

[1993] can be used to solve this problem. By applying a breadth first search, all connected voxels can be selected. For large binary grids, applying a flood filling algorithm may be time consuming. Therefore it may be of interest to use the optimized version mentioned in [Franklin and Landis, 2006; Isenburg and Shewchuk, 2009]. An illustration of this process is shown in Figure 34.

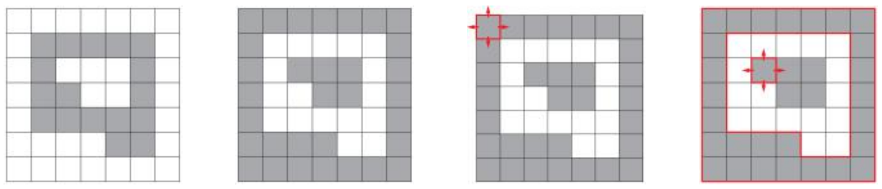


(a) Selecting a start voxel (b) Flood filling through a volume (c) Selecting a new start voxel

Figure 34: Detecting connected components

A starting voxel is needed as shown in Figure 34a. By using 6-adjacency as described in § 3.3.1, the flood filling algorithm will start looking for neighboring voxels, as shown in Figure 34b. When no new neighboring voxels can be found, but voxels with value 1 are still present, a new starting voxel is selected (see Figure 34c). In case more than one connected component is found, these volumes can be separated. Alternatively, volumes lower than a certain threshold can be disregarded since they are likely to not represent a 3D City object.

I have adapted this principle to be applied on the inverted binary grid in order to remove holes. In this case the result should contain only one connected component, namely the exterior. Any additional connected components in the inverted grid are holes, and should be added to the interior.



(a) Initial grid with a hole **(b)** Inverse the grid **(c)** Filling the exterior **(d)** Detecting holes

Figure 35: Performing a closing operation by using Euclidean distance mapping

Figure 35 shows how the exterior can be found by applying the flood filling algorithm on the inverted binary grid. Figure 35a shows the initial grid which is reversed in Figure 35b. The exterior volume is selected using 6-adjacency (see Figure 35c). Any remaining voxels with value 1 must be holes, as shown in Figure 35d. In other words we can state that holes are present when several connected components are found in the inversed grid. By removing any redundant connected components on the inversed grid, a manifold volume in the binary grid can be ensured.

3.3.4 Suitability of morphological operators

The suitability for certain morphological operators is closely related of the process of surface reconstruction. Some methods rely solely on the binary grid values, which make the morphological operators very effective. Other methods make use of additional information of the initial polygonal model, along with the formed binary grid. Performing morphological operators for these cases may not help or even obstruct the surface reconstruction. This distinction between pure binary grid versus original polygon model input will be further explained in the next Section 3.4

3.4 SURFACE RECONSTRUCTION

The goal of the surface reconstruction component is to rebuild a polygonal mesh out of the voxel representation. The first step of converting a binary grid into a triangle mesh is called isosurface extraction. The isosurface represents a triangle mesh containing vertices which are meeting a certain condition. For example, this could mean that the boundary between interior $(= 1)$ and exterior $(= 0)$ is positioned at the isosurface with value 0.5. An overview of existing isosurface extraction methods will be given in § 3.4.1. Then the theory of three of these methods (Marching cubes, Dual contouring and Pressing) will be described in more detail.

3.4.1 Overview

Since many isosurface extraction techniques exist, an overview will be given specifying the advantages and drawbacks of existing methods. Since our goal is to repair 3D building models, isosurface extraction techniques can be separated on the following criteria:

- Input: some isosurface extraction methods only use a binary grid ($\mathbb{Z}^3$) as input. Other algorithms require additional information. Specifically *Hermite data* is required for some of the algorithms, which means the set of all values and derivatives for a specific function. In case of a 3D-model this means that intersections and corresponding surface normal vectors are needed. This needs to be taken into consideration while determining the suitability of any morphological operators.
- Oblique surfaces: while some of the methods estimate the isosurface using only orthogonal and 45°angles, other methods can produce surfaces oriented in any direction which results in a more accurate surface reconstruction.
- Sharp Features: some of these algorithms round off any sharp features which may be present in the input. Algorithms exist for sharpening these edges later, but using algorithms that preserve sharp features are preferred.
- Triangle reduction: some methods create a (large) number of triangles which is directly related to the binary grid resolution. Other methods automatically produce larger surfaces where possible. This is an advantage although the implications for attribute preservation should be kept in mind.
- Manifoldness: in order to repair 3D City models the final triangle mesh should be manifold. Some methods may results in gaps or self-intersections in some cases, which is unwanted.

Algorithm	Input	Oblique	Sharp features	Triangle reduction	Manifold
Marching Cubes [Lorensen and Cline, 1987]	$\mathbb{Z}^3$	No	No	No	Ambiguities possible
Marching Cubes 33 [Chernyaev, 1995]	$\mathbb{Z}^3$	No	No	No	Yes
Discretized Marching Cubes [Montani et al., 1994]	$\mathbb{Z}^3$	No	Yes	Yes	Yes
Sharp Feature Marching Cubes [Loke and Jansen, 2007]	$\mathbb{Z}^3$	No	Yes	No	Yes
Pressing [Chica et al., 2008]	$\mathbb{Z}^3$	Yes	No	Yes, returns planes	unknown
Extended Marching Cubes [Kobbelt et al., 2001]	$\mathbb{Z}^3$ + Intersections	Yes	No	No	Self-intersections
Dual Contouring [Ju et al., 2002; Schaefer et al., 2002]	$\mathbb{Z}^3$ + Hermite data	Yes	Yes	No	Self-intersections
Manifold Dual Contouring [Schaefer et al., 2007]	$\mathbb{Z}^3$ + Hermite data	Yes	Yes	No	Yes
Dual Marching Cubes [Schaefer and Warren, 2005]	$\mathbb{Z}^3$ + Hermite data	Yes	Yes	Yes OC-tree	Self-intersections
Manifold Dual Marching Cubes [Manson and Schaefer, 2010]	$\mathbb{Z}^3$ + Hermite data	Yes	Yes	Yes, OC-tree	Yes

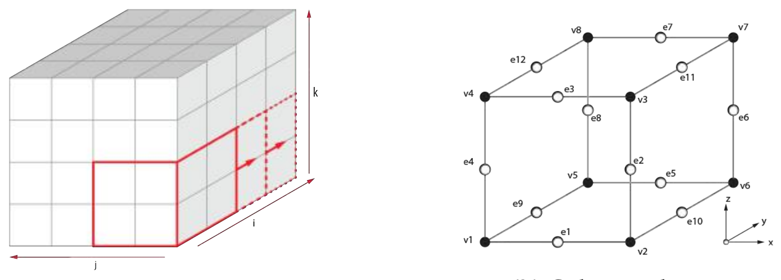
Table 6: Iso-surface extraction overview

Because of the abundance of isosurface extraction methods, I have made a decision on which algorithms are further investigated. Three of the algorithms have been selected because of their distinctive approach. Each of them is promising for a different reason; (i) simplicity, (ii) oblique surface reconstruction from a binary grid & (iii) making use of the input model.

- Marching Cubes [Lorensen and Cline, 1987] will be described since closely related variations of it are used both in [Nooruddin and Turk, 2003] and [Hétroy et al., 2011]. Although it has clear disadvantages (i.e. no support of oblique surfaces and sharp features) describing it is a must since it is the most commonly applied algorithm for isosurface extraction. Closely related methods such as [Chernyaev, 1995] are not considered unsuitable but their implementation would lead to similar results.
- Dual Contouring [Ju et al., 2002] will be described since it shows how the input model can be used in order to rebuild oblique surfaces and sharp features. Similar and more advanced methods are also listed, but these implementations are based on the same principle. To explore the potential of these *adaptive contouring* methods, dual contouring will be researched.
- Pressing [Chica et al., 2008] will be described since it is specifically aimed at the problem of rebuilding planar regions of building models. This methods is thus capable of creating oblique surfaces using only a binary grid as input.

3.4.2 Marching cubes

One of the first methods to extract a triangle mesh from a 3-dimensional grid was published by Lorensen and Cline [1987]. By iterating through the entire voxel model and looking at blocks of 8 voxels at a time, 256 (2^8) combinations can be found. This is illustrated in Figure 36a. By rotating and mirroring, these can be brought down to 15 possible combinations of empty& filled voxels. For every block, the triangle configurations can be looked up in a pre-computed lookup table. In theory each vertex position along the edge is calculated by interpolating between scalar values. However, if we are only dealing with binary values (exterior = 0, interior = 1), all these vertices will be placed in the middle of an edge.

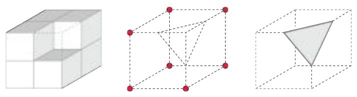


(a) Looping through cubes

(b) Cube numbering

Figure 36: Looping through cubes and indexing the edges

Now, for every case triangles can be made. Based on the configuration of the voxel-blocks, triangles are added to a mesh. The positions are based on the numbering of the cubes' vertices, of which the numbering is illustrated in Figure 36b. Two examples of cases are visible in Figure 37.


(a) Case 1

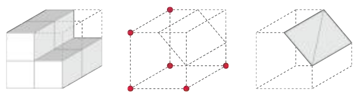

(b) Case 2

Figure 37: Examples of Marching Cubes cases

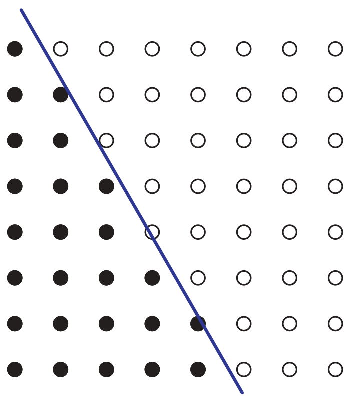
Illustrations of all 15 cases are given in the appendix (see Appendix A). An overview of the process is given in algorithm 3.6.

Algorithm 3.6: MARCHING CUBES
Lorensen and Cline [1987]

Input: A binary grid G containing voxels V_{ijk} with values 0 for exterior, 1 for interior. A look-up table for 256 cube configurations
Output: A triangular mesh M

```
1 for V_ijk in G do
2     Get configuration of cube $V_{i..i+1,j..j+1,k..k+1}$
3     Look up triangles for cube configuration in table
4     for triangle returned by lookup table do
5         add triangle to mesh $M$
6     end
7 end
```

Two drawbacks of this method for building models are its approximation of oblique surfaces and the possibility of ambiguous cases. Figure 38 shows how an oblique surface is voxelized and turned into a triangle mesh. The surface in this example is oriented at a 30°angle. Since the vertices are placed in the middle of an edge, the original surface can only be approximated by connecting triangles under 45°angles.


(a) Voxelized oblique surface

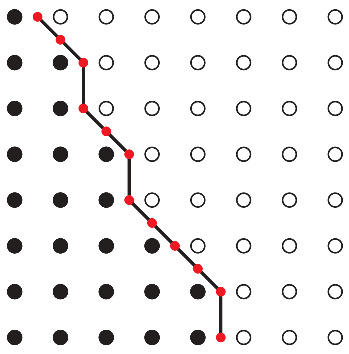

(b) Stair stepping

Figure 38: Applying Marching Cubes on a binary grid

To create smooth surfaces, an edge decimation strategy is applied by Nooruddin and Turk [2003]. This principle will be explained in § 3.5.2 and the suitability for 3D building model repair will be determined in Section 4.3. Some cases and combinations of Marching Cubes are ambiguous, meaning that cube configurations can be interpreted in several ways. The results is that the created triangles are not necessarily topologically correct. However, this can only happen in some cases and specific combinations of cases which will generally not occur while processing a manifold volume. Manifold volumes may only produce ambiguities when there are occurrences of hanging (edge-adjacent) voxels. An example of such a case (10) is shown in Figure 39.

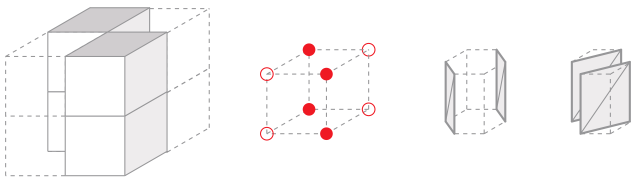


Figure 39: Ambiguity in Marching Cubes Case 10

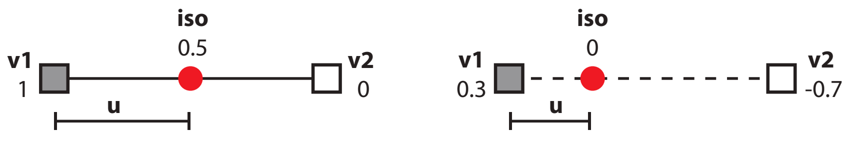
Modified versions of the Marching Cubes algorithm have been created [Wilhelms and Van Gelder, 1990; Chernyaev, 1995] in order to solve these ambiguities and ensure a correct topology. However, since we are assuming the input volume is manifold due to the applied morphological operations, these unwanted ambiguous cases can be avoided in general. The approach on how to do so using N-adjacency is described in Section 4.3.
A more precise option for surface reconstruction is to apply the Marching Cubes algorithm on a 3D grid of signed distances instead of using binary values. In such a grid all voxels receive a value storing the closest distance to the original model. All distance values of voxels in the interior receive a positive sign. Vice versa, all values of voxels in the exterior receive a negative sign.
Using a linear interpolation on a grid edge connecting the interior and exterior of a binary grid (values 0 and 1) always results in a vertex with an isosurface value of 0.5 being placed on the middle of the edge. However, when the interpolation is performed on a 3D grid of signed distance values, the vertex positions with isosurface value of 0 can be more accurately placed along the edge. Equation 3 shows how the position on this edge can be computed, where $v1$ is the value of the interior vertex and $v2$ is the value of the exterior vertex. By determining a value for the isosurface, the position of the interpolated vertex can be computed.

Computing the vertex position by linear interpolation of signed distance values:

$$iso = v_1 \cdot (1 - u) + v_2 \cdot u \tag{3a}$$

$$u = \frac{v_1 - iso}{v_1 - v_2} \tag{3b}$$

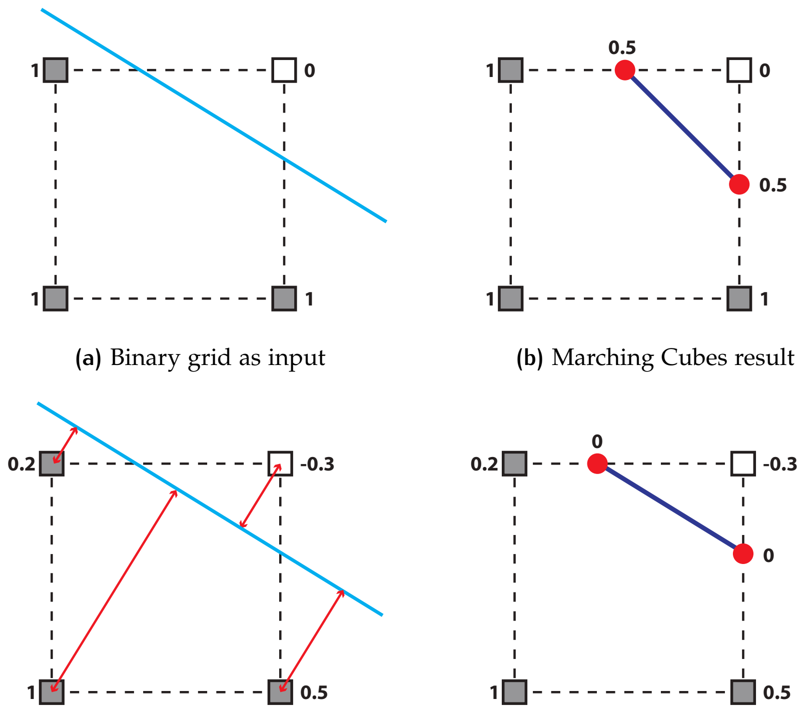
Figure 40a shows the linear interpolation along an edge with binary values for an isosurface value of 0.5, which always results in the vertex being placed in the middle of the edge. Figure 40b gives a similar example but now signed distance values are available and the isosurface value is set at 0.



(a) Interpolation on binary values (b) Interpolation on signed distances

Figure 40: Linear interpolation on a binary grid and a signed distance field

Using a interpolation of signed distance values will result in a more accurate surface reconstruction. Figure 41a and Figure 41b show the application of Marching Cubes on a binary grid, where the surface position is only estimated and vertices are placed in the middle of the edge. Figure 41c and Figure 41d illustrate the creation of the distance values and the result after applying Marching Cubes. In this example, both of the new vertices are positioned on the original surface.



(a) Binary grid as input (b) Marching Cubes result

(c) Signed distances as input (d) Marching Cubes result

Figure 41: Comparison of applying Marching Cubes on binary values versus using signed distance values

Applying the Marching Cubes algorithm on the distance field is guaranteed to produce output which is topologically equal to the output of applying Marching Cubes on a binary grid. The same number of triangle is produced, but the position of the vertices will be closer to the original model. However, since the output is topologically equal, this means that the same ambiguous cases may still appear when the original Marching Cubes algorithm is applied. Similarly, corners will still be rounded off when applying the algorithm on a grid with signed distances. Figure 42 illustrates the rounded off result after applying Marching Cubes on signed distance values.

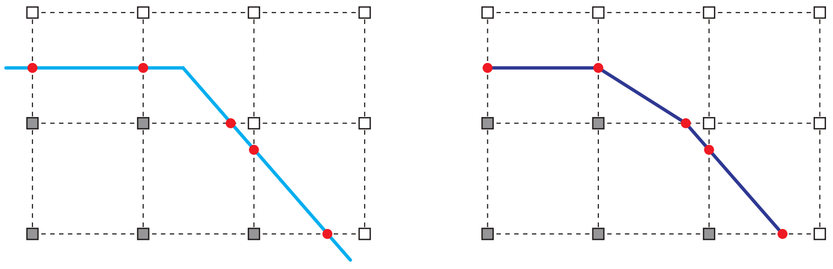
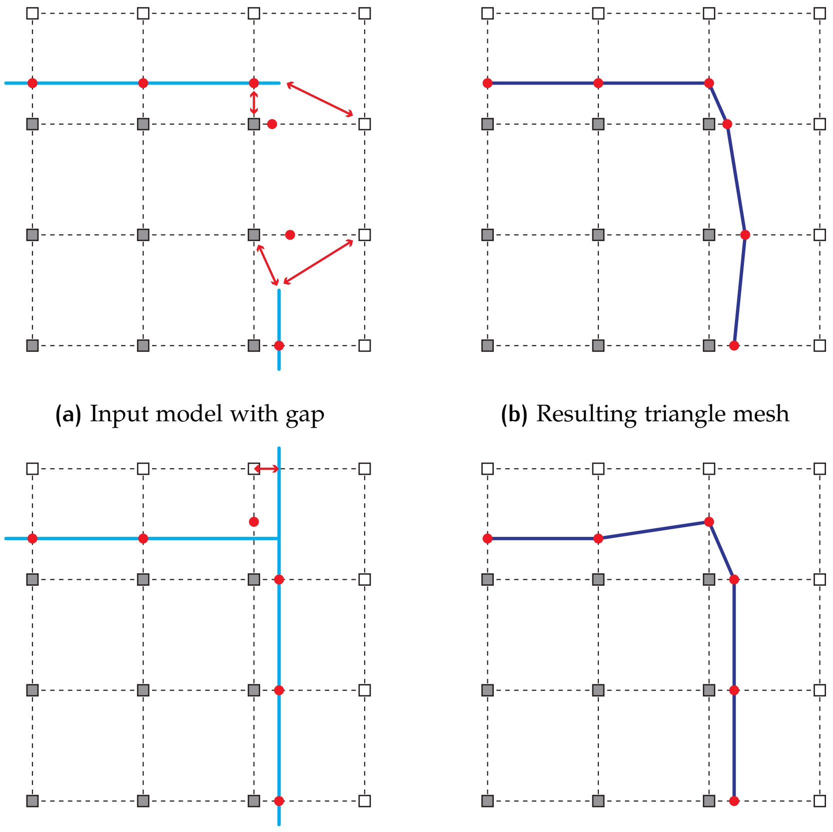

(a) Sharp edge in input (b) Rounded off corners in result

Figure 42: Rounding off sharp features when applying Marching Cubes on signed distance values

The clear advantage is that oblique surfaces are accurately reconstructed, avoiding the creation of a stair-stepping effect. However, it is possible that a corner will be rounded off in such a way that the new vertices are positioned very close to each other (see Figure 42b). In case of CityGML models, this may be regarded as a consecutive points same defect, depending on the selected threshold. Another issue that has to be taken into account is the occurrence of gaps and overshoots in defective input.


(a) Input model with gap (b) Resulting triangle mesh

(c) Input model with overshoot (d) Resulting triangle mesh

Figure 43: The possible effect of a gap on the result of applying Marching Cubes on signed distance values

In case of a gap, a part of the surface is missing. This may result in some distance values being too high, preventing a precise surface reconstruction. An example of this is shown in Figure 43a and Figure 43b. In case of an overshoot, some distance values may be calculated smaller than what is desired. An example of this effect is shown in Figure 43c and Figure 43d. Both cases may result in small artefacts in the resulting mesh. Compared to using a binary grid however, the estimated surfaces are still much more accurate.

The further research performed in this thesis, makes use of a Marching Cubes implementation on a binary grid. However, the possibility of further developing upon this thesis with an implementation of Marching Cubes for a signed distance field is recommended as future work (see Section 6.3).

3.4.3 Dual contouring

An example of a surface reconstruction method which uses the input model is Dual Contouring. It was first described by Ju et al. [2002] and expanded towards the implementation by [Schaefer et al., 2002]. As the name of the method describes, it creates a contour by connecting the duals of a grid. Figure 44 shows a diagram of the relation between a grid and its dual.

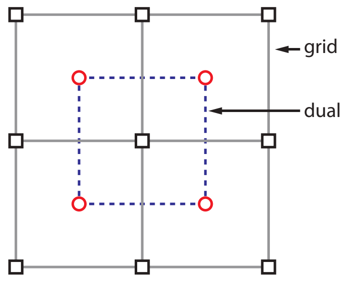


Figure 44: Diagram of the relation between the voxel grid and dual

In 3D a dual vertex can be placed within every cube of 8 voxels. However, to extract the contour of the building, we only want to do this for cubes with a sign change. This means that only cubes which contains voxels both in the interior (=1) and exterior (=0), should be selected. Figure 45 shows a diagram of a voxel grid showing grid edges with a sign change.

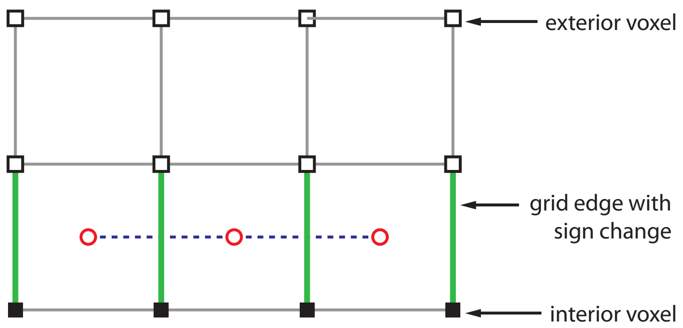


Figure 45: Diagram of grid edges intersections with the input model

The black voxels represent the interior and white voxels represent the exterior. The grid edges which connect a black and a white voxel, are shown in green. These green edges are expected to intersect with the original model. Each of these edges is neighboring with 4 cubes. Only for these cubes the dual position is required. An illustration of the configuration of a grid edge and its 4 neighbors in 3D is shown in Figure 46.

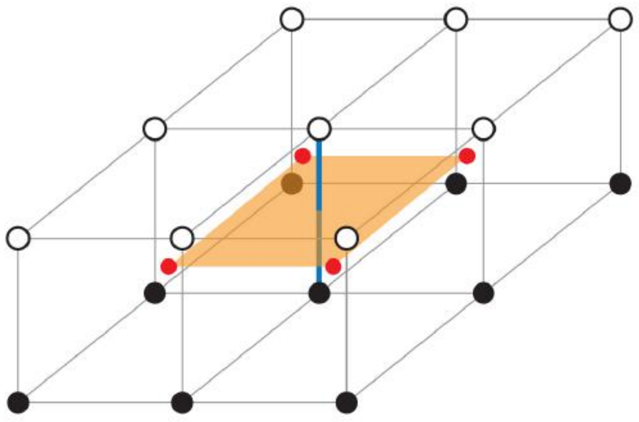


Figure 46: Diagram of the Dual Contouring principle

By connecting the 4 duals of the cubes neighboring with the grid edge, 1 quad (or 2 triangles) can be created. Now in order to compute the dual vertex positions, Dual Contouring makes use of *Hermite data*. Strictly speaking, the Hermite data contains the values of a function and its derivatives. In case of a 3D model this translates to a set of intersection points between the grid edges and the original input surfaces p and the corresponding surface normals n. This can be estimated by using a method such as Principal Component Analysis (PCA) as described by Wold et al. [1987]. For every grid edge which intersects with the polygonal model, the intersection point and surface normal are stored. Based on these values, a vertex position can be determined within every grid cell. Figure 47 shows an illustration in 2D of how the intersection and normal vector can be used to compute dual vertex within a grid cell. By connecting these dual vertices, the surfaces can be reconstructed.

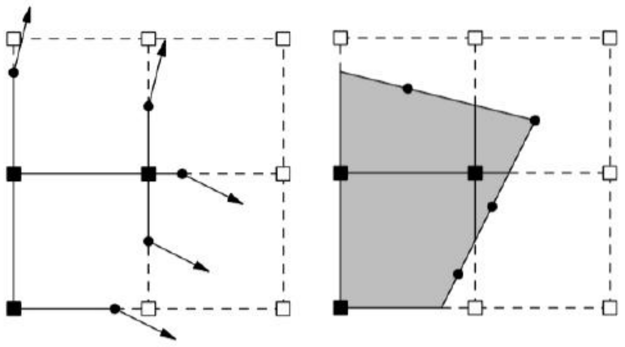


Figure 47: Diagram of using Hermite data [Ju et al., 2002]

Since every intersection point p_i has a corresponding surface normal n_i, the equation of a plane is represented for every intersection. The QEF is then the sum of the squares of the distances from the cubes center point to each of the defined planes. By minimizing the QEF, the most appropriate point can be found. This concept was first described by Kobbelt et al. [2001]. The computation of the vertex position is determined by minimizing the quadratic error function of the distances from a point to a number of planes, as described in Equation 4:

$$E|x| = \sum (n_i \cdot (x - p_i)^2) \tag{4}$$

Here p_i is the intersection point and n_i the corresponding surface normal, for all grid edge intersecting the polygonal model. However, since minimizing the QEF to the cubes center point may result in a dual vertex outside of the cube, an adaption is proposed by Ju et al. [2002] and Schaefer et al. [2002]. By computing the minimized QEF towards the mass-points instead of the center-point, the resulting dual vertex inside the cube is guaranteed. Figure 48 shows how minimizing towards the mass point avoids dual vertices being placed outside of the cube.

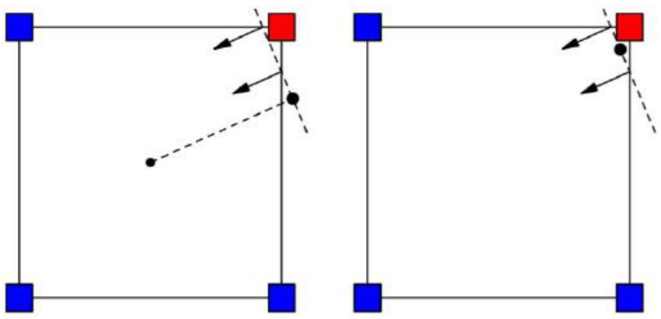


Figure 48: Difference between minimizing towards center-point and mass-point [Schaefer et al., 2002]

In general the Dual Contouring method is able to reconstruct the isosurface when Hermite data is available for the entire model. However, when this method is applied on defect 3D City models, this is often not the case due to gaps (missing Hermite data) or overshoots (additional Hermite data). I propose a method in Section 4.4 where an implementation of Dual Contouring is aimed towards the repair of defect 3D City models, based on a set of heuristics. The possibility of improved surface reconstruction will be illustrated, along with important issues that are encountered.

3.4.4 Pressing

An alternative method for isosurface extraction on pure binary grids is Pressing, partially aimed at finding *large planar tiles* [Chica et al., 2008]. Its purpose is to accurately rebuild 3D models from their binary volume, as is illustrated in Figure 49. It consists of three steps:

- detection and creation of large planar tiles
- detection and creation of curved surfaces
- sharpening the edges

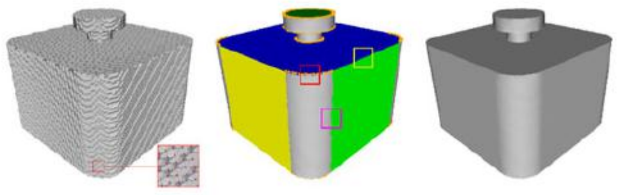


Figure 49: Overview of pressing method

Since curved input models are outside the scope of this thesis, this part of their method can be disregarded. The process of edge sharpening is described in Section 3.5. Here we will describe the process of detecting large planes. The method of detecting the largest flat areas in a voxel representation is described by Andújar et al. [2004]. The process starts by finding the so called *sticks*. Sticks are defined as the orthogonal grid edges which connect an interior voxel (value =1) with an exterior voxel (value = 0). Therefore every stick is expected to intersect with a surface of the polygonal model. Two illustration of these sticks are shown in Figure 50.

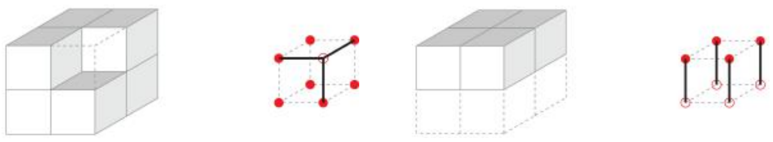
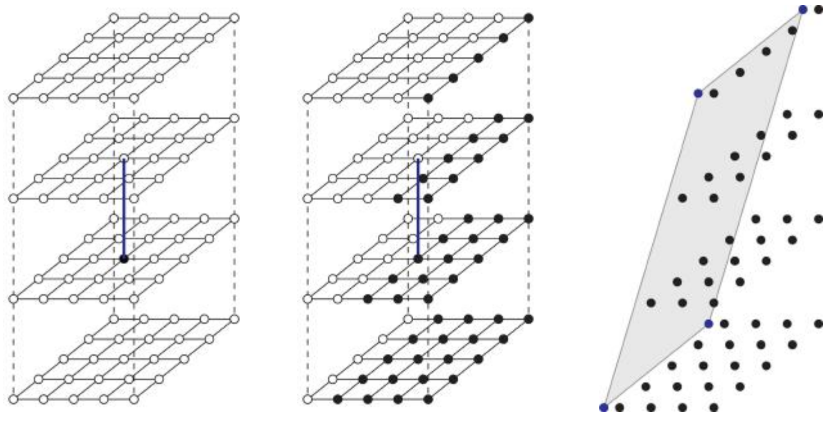


Figure 50: Finding sticks connecting interior with exterior

The aim is to find a plane that intersects the maximum amount of sticks. In order to determine the parameters of this plane, the 5*x*5*x*4 neighborhood of each stick is looked at. A large lookup table containing all possible, manifold configurations of the 5*x*5*x*4 neighborhood needs to be created. This way every stick can be related to an entry in the lookup table, returning parameters for the plane which separates the interior voxels from the exterior voxels. The size of the neighborhood is a trade-off between the size of the dictionary and the ability to efficiently detect large planar regions. A larger neighborhood (e.g. 7*x*7*x*6) would be more efficient at detecting planar regions, but the look-up table would be extremely large. On the contrary, using a smaller neighborhood (e.g. 3*x*3*x*2) will result in a smaller look-up table but Andújar et al. [2004] describe how it is only suitable for detecting strongly curved regions. The process of finding the plane parameters for a possible 5*x*5*x*4 neighborhood is illustrated in Figure 51.



(a) Selecting a stick, connecting interior with exterior

(b) Finding the sticks' $5x5x4$ neighboring voxels

(c) Plane parameters for this neighborhood

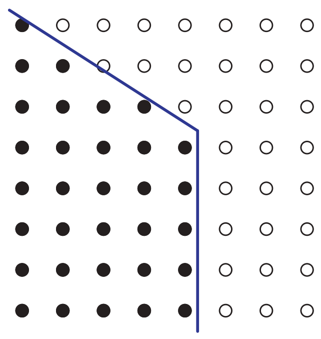
Figure 51: Searching neighborhood of each stick

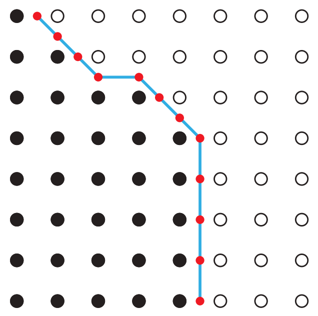
Once all sticks have casted their vote for plane parameters, the combination of these votes can be used to determine the most likely position of a plane. This is done by performing a global optimization, resulting in surfaces which are guaranteed to cover all interior voxels. The process continues to look for the junction points between these planes and applies edge sharpening technique.

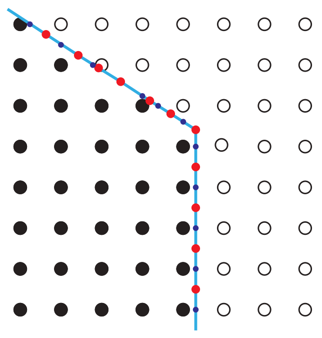
Although this methods proves to be a promising way of accurately rebuilding a triangle model from a binary grid, a major drawback of this approach is the complexity. The lookup table contains ± 32.000 entries, because of the large neighborhood, which is created in a time consuming process. Additionally, using the method as is may result in curved surfaces being estimated at some locations while curved geometry may or may not be supported by the data format. While using a relatively low resolution, many regions where a $5x5x4$ neighborhood does not represent a plane may be found. A new solution should be created for those regions.

3.4.5 Evaluation

In the related work on surface reconstruction (see Section 3.4), 3 distinctively different algorithms have been researched. An illustration of the principles of each of these algorithms is shown in Figure 52.


(a) The original polygonal model


(b) Marching Cubes result


(c) Dual Contouring result

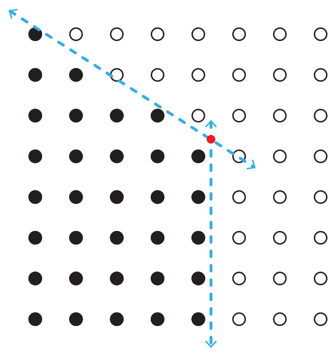

(d) Pressing result

Figure 52: Overview of isosurface extraction techniques

Despite the fact that the Marching Cubes algorithm has some limitations regarding oblique surface reconstruction (see Figure 52b, it is widely used in voxel based repair methods. Its main strengths are simplicity and quick processing. Therefore its applicability will be tested in Section 4.3. The Pressing method describes a unique approach for acquiring oblique surfaces from binary grid. Although, the approach is very relevant for 3D City model repair, the implementation and analysis of this method are not considered achievable within the time limits of this research. However, this method can possibly be a part of a future work which aims to rebuild oblique surfaces from a binary grid. Finally the Dual Contouring approach shows a way of rebuilding oblique surface by using additional information such as intersection points and surface normals. Admitting that this method is mostly suitable for rebuilding models where all Hermite Data is found, the possibilities of how to apply this method to defect geometry will be further explored in Section 4.4. The fact that Dual Contouring may result in self-intersections must be taken into account. If time allows, the extended version described by [Manson and Schaefer, 2010] will be applied for achieving manifold results.

3.5 POST PROCESSING

Once the iso-surfaces have been extracted from the binary grid, the resulting mesh may be post processed in order to achieve a satisfying result.

3.5.1 Edge sharpening

Since some isosurface extraction algorithms such as Marching Cubes or Pressing result in rounded of corners a method proposed in [Attene et al., 2003] may improve the results. To detect the edges which need to be adapted for edge sharpening, the triangle mesh has to be stored using a topological model. Therefore an intermediate step of iterating over the Marching Cubes triangle mesh is needed. Once this is done, the following filtering steps can be applied as described in [Attene et al., 2003]:

- 0. Paint brown the 'smooth' edges where incident triangle normals are similar.
- 1. Paint red each vertex whose incident edges are all brown.
- 2. Each triangle with a red vertex becomes red.
- 3.Paint red (recursively) each triangle that is adjacent to a red triangle through a brown edge.
- 4. Paint red the edges and vertices of the red triangles
- 5. Paint blue each non-red edges joining two red vertices
- 6. Paint green each triangle with three blue edges

The result of these filtering steps is illustrated in Figure 53. Now every blue edge lies on a chamfer which has been rounded off in one direction. Each green triangle is a corner triangle which has been rounded off in two directions.

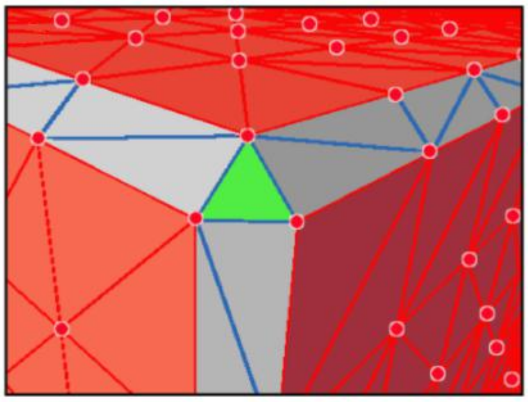


Figure 53: Result of the filtering steps [Attene et al., 2003]

To rebuild the sharp features, the existing triangles will be split up in several smaller triangles. Three types of triangles can be separated, containing one, two or three rounded edges. Figure 54 shows these cases.

To subdivide these triangles new vertex have to be placed. The yellow vertices are positioned on blue chamfer edges, meaning they should be at

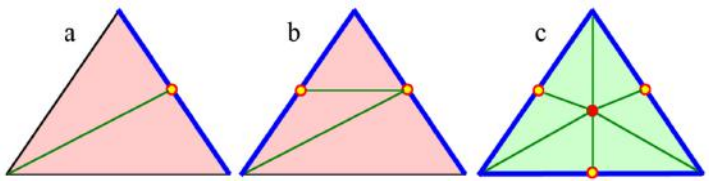


Figure 54: Vertex placement on chamfer edges and corner triangle [Attene et al., 2003]

the intersection of two planes. The red vertex inside the green triangle should be positioned at the intersection of three planes.

For a given chamfer edge E with two vertices A and B, the weighted sum of the normals N and M are computed over all red triangles incident to point A and B. This way two planes P and Q can be defined, the first being orthogonal to N and passing through vertex A, the second being orthogonal to M and passing through vertex B. Now the new vertex V is placed on the line where P and Q intersect, as close to the midpoint of the original edge E as possible. These calculations for a new vertex V for a chamfer edge is shown in Equation 5:

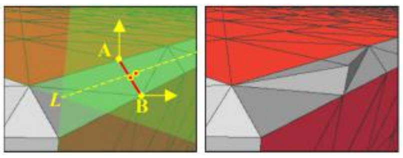
$$V = (A+B)/2 + (h/k)H \tag{5a}$$
$$H = AB \times (M \times N) \tag{5b}$$
$$h = AB \cdot N \tag{5c}$$
$$k = 2(M \cdot N)(AB \cdot N) – 2(AB \cdot M) \tag{5d}$$

For every corner triangle containing the vertices A, B and C an extra vertex W is placed. The weighted sum of the normals N is computed over all red triangles incident to point A, where the angle between incident edges is the weight for each triangle. Now a plane P can be defined which is intersecting with point A and orthogonal to N. Similarly, the planes Q and R can be defined for vertices B and C. Now the position of vertex W is found at the intersection of the planes P, Q and R. This can be solved in a linear way as is shown in Equation 6.

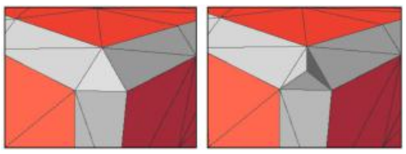
The calculation of a new vertex W on a corner triangle edge:

$$W \cdot N = A \cdot N \tag{6a}$$
$$W \cdot M = B \cdot M \tag{6b}$$
$$W \cdot L = C \cdot L \tag{6c}$$

By detecting the rounded off triangles and adding new vertices based on plane intersections, many of the rounded corners can be sharpened. A visualization of this process on a simple corner is shown in Figure 55.



(a) Sharpening of a chamfer edge



(b) Sharpening of a corner

Figure 55: Edge Sharpening [Attene et al., 2003]

Figure 55a shows how a single chamfer edge is sharpened. Figure 55b illustrated the sharpening of a corner. Although in theory the computation of new vertex positions is correct, in practice some exceptions take place, A description of these cases is described by Attene et al. [2003] but not all solutions are given. These cases and possible solutions will be described in § 4.3.3.

3.5.2 Edge decimation

A method for triangle reduction which was opted for in Nooruddin and Turk [2003] is using Quadric Edge Metrics for surface simplification. This application is described by Garland and Heckbert [1997]. By contracting certain edges into a single vertex, the number of triangles can be reduced. This principle is shown in Figure 56.

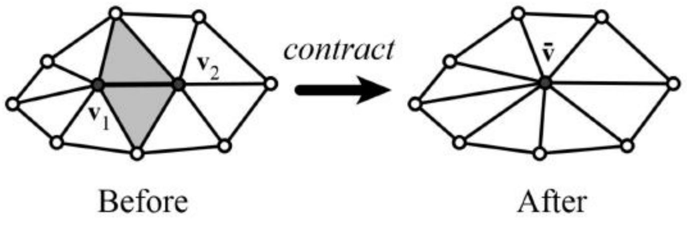


Figure 56: Principle of quadric error mesh simplification [Garland and Heckbert, 1997]

The before picture shows a highlighted edge consisting of vertices v_1 and v_2. If it is determined that this edge should be contracted, then a new vertex $\bar(v)$ will be placed: $v_1, v_2 \Longrightarrow \bar{v}$. All edges that connected to either v_1 or v_2 are now connected to $\bar(v)$. All triangles that are incident to the highlighted edge are removed. In order to detect which edges to contract during a certain iteration, the cost of a contraction is determined. The cost is computed as a quadric error function of the new vertex position relative to the old vertices. Contractions with the lowest cost will be performed earlier in the iteration. This algorithm has later been adapted for triangle color and attribute support by [Garland and Heckbert, 1998], which may be relevant for attribute preservation.

3.5.3 Mesh comparison

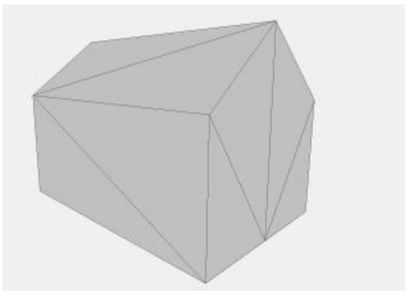
As a means of validating the repair process, the geometric quality of the repaired model can be measured. To do so a comparison can be performed between the input and output mesh. The Haussdorf distance can be used to find the largest distance between a set and the nearest point in another set. This concept has been applied in the comparison of triangle meshes by Aspert et al. [2002] and Garland and Heckbert [1997]. For a surface S and a point p the minimal distance is found by sampling vertices p' along the surface.

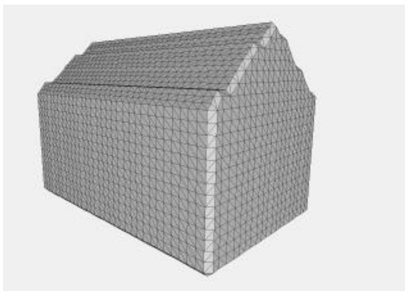
$$\epsilon(p, S) = \min_{p' \in S} d(p, p') \tag{7}$$

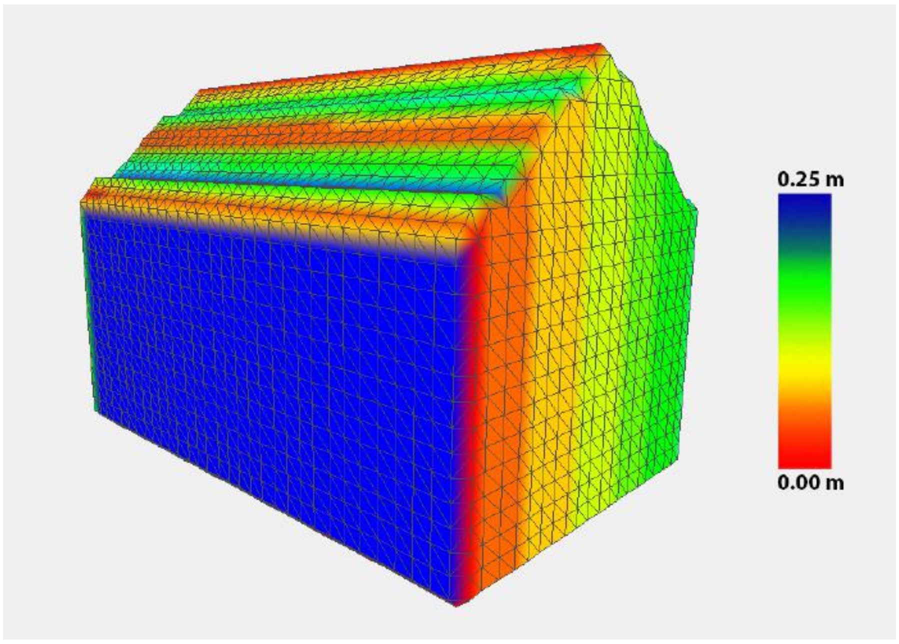
This principle of using sample points on a surface can be used to compute the distance between two surfaces.

$$E(S_1, S_2) = \max_{p \in S_1} \epsilon(p, S_2) \tag{8}$$

This way an estimation of the geometric difference between two surfaces can be acquired. For every sample vertex, a corresponding distance is stored.


(a) Original 3D model before processing


(b) 3D Model after voxelization and Marching Cubes



(c) Geometric errors highlighted in red

Figure 57: Visualization of the geometric error

Based on these distances, colors may be given to the mesh to visualize the location of large geometric errors. An illustration of the geometric error is visible in Figure 57, showing the geometric error of the stair stepping effect and the error free surface due to model alignment.

4 MY VOXEL-BASED REPAIR METHOD

This chapter starts by describing the general repair methodology I propose, in which two different approaches have been tested (see Section 4.3 and Section 4.4). This will be followed by a practical description of the different components. The specific options within each component are considered and decisions are made based on a heuristic approach.

4.1 METHODOLOGY

I propose two voxel-based repair approaches to repair 3D buildings. These two approaches are extensions of two different isosurface extraction techniques which are described in Section 3.4.

Approach 1 will continue with the work of Nooruddin and Turk [2003] and Hétroy et al. [2011] using a Marching Cubes algorithm (see § 3.4.2) and fitting it towards 3D City model repair. The second approach will be based on the Dual Contouring algorithm (see § 3.4.3), described by Ju et al. [2002] to show the potential of using the input model. This way a better rebuilding of oblique surfaces may be achieved. Both methods are significantly different during and after the Surface reconstruction component, however the first steps of Pre-processing and Voxelization are similar. Therefore the methodology can be depicted as in Figure 58.

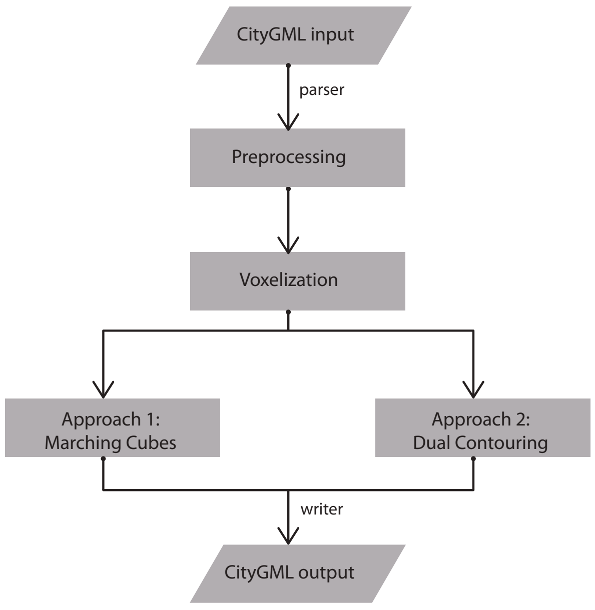


Figure 58: Overview of the methodology

In order to perform the described algorithms some pre-processing of the input model is needed. The polygonal model has to be converted to a triangle model before applying the actual repair steps. This can be done using FME or the CityGML2OBJs tool available at `http://erlangarticles.com/p/tudelft3d/CityGML2OBJs`. This chapter will now continue to select the most suitable components for each approach.

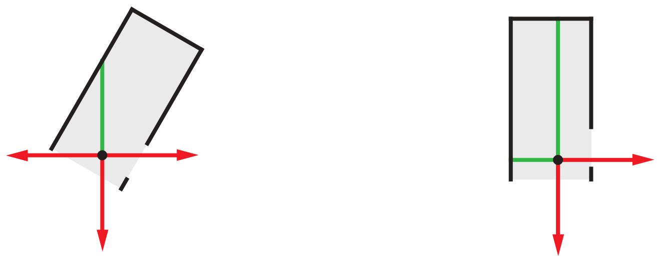
4.2 VOXELIZATION

The first step is to voxelize the polygonal model, in order to rebuild it into a manifold, volumetric representation. The decisions of how this is done, are described in this section.

4.2.1 Scan conversion

MODEL ALIGNMENT

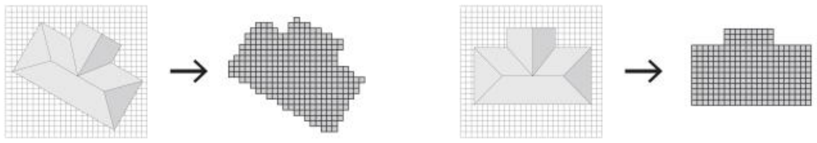
A basic alteration which can be performed is model alignment, which has the potential of improving the results. This is a step which can be applied before starting the repair process, which orients the 3D model in an orthogonal direction, corresponding with the binary grid. Aligning the input model is an optional step (of little cost) that may improve the output voxelization for two reasons. Firstly, by aligning the model the different scan lines are more likely to intersect different surfaces, as is illustrated in Figure 59. In general this may lead to more positive votes in orthogonal buildings with gaps present. This was the adjustment of the scan conversion parameters (threshold and number of scan directions) can be avoided.



(a) Non aligned majority voting **(b)** Aligned majority voting

Figure 59: The effect of aligning on the repair capability

Secondly, since many of the building in a 3D city model are (partially) orthogonal, many flat surfaces will be connected through 90 °angles. Where the original position of the building will lead to a jagged voxelization, an aligned model will result in a voxelization with flat parts. This will mostly be useful in Approach 1 using Marching Cubes (see Section 4.3) to reduce the bumpiness and artefacts of the resulting mesh.



(a) Non aligned voxelization result **(b)** Aligned voxelization result

Figure 60: The effect of aligning the input model

Based on these two reasons, I decided to align all input models prior to the scan conversion, slightly improving the repair capability and enabling smooth surface reconstruction of orthogonal buildings in approach 1.

In order to calculate the angle of rotation along the Z-axis, the normal vectors for every triangle in the triangulated model are computed. All triangles with normal vectors with an absolute z-component larger than a certain threshold (for instance 0.1) are disregarded. By taking the area of the remaining triangles as weight, the normal vector with the largest corresponding area can be found. Then we compute the angle between this normal vector and the predefined orthogonal direction. In order to run this step, the polygonal model should be triangulated.

Algorithm 4.1: Model Alignment

```
Input: A triangulated polygonal model $M$ and an orthogonal vector $\vec{ortho}$
Output: A polygonal mesh oriented such that the largest surface area has a normal vector equal to $\vec{ortho}$
1  Find all unique triangle normal vectors $N$ present in $M$
2  for every $N$ do
3      set $N_{weight}$ at 0
4  end
5  for each $\tau$ in $M$ do
6      Compute triangle normal vector $N$ and triangle area $\tau_{area}$
7      Corresponding $N_{weight} = N_{weight} + \tau_{area}$
8  end
9  Alignment vector $\vec{V_{alignment}}$ = $N$ with max $N_{weight}$
10 $\alpha = \angle$ between $\vec{V_{alignment}}$ and $\vec{ortho}$
11 for triangle $\tau$ in $M$ do
12     for vertex $v$ in $\tau$ do
13         orient points:
14         $v = (x_{align}, y_{align}, z)$
15     end
16 end
```

Once the rotation angle α around the Z-axis is known, the model can be oriented, see Equation 10.

Orienting the models' vertices around the Z-axis:

$$x' = x \cdot \cos(\alpha) - y \cdot \sin(\alpha) \tag{9a}$$

$$y' = y \cdot \sin(\alpha) - y \cdot \cos(\alpha) \tag{9b}$$

$$z' = z \tag{9c}$$

A result of the model alignment is that several slices of voxels are placed at the border of the bounding box. During the later stages of the rebuilding process, it will be vital to find the regions connecting interior (value = 1) and exterior (value = 0) voxels. For this reason a buffer of 0-valued voxels is needed around the model as shown in Figure 61.

After this step an adjustment of the binary grid values is needed. The dimensions, scale and translation values of the grid have to be updated:

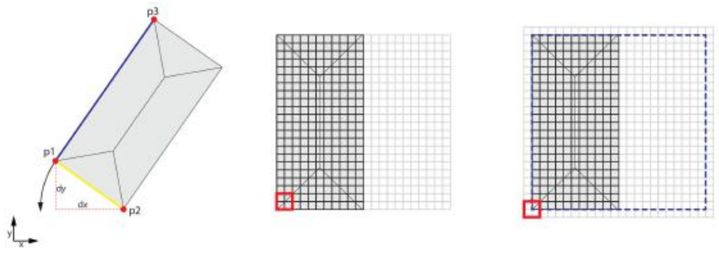


Figure 61: A buffer on the grid is needed after aligning

Updating the dimension and scale of the grid after creating a buffer:

$$dimension_{new} = dimension_{old} + 2 \tag{10a}$$
$$scale_{new} = scale_{old} + (2 \cdot voxelsize) \tag{10b}$$
$$translate_{new} = translate_{old} - voxelsize \tag{10c}$$

The updating of each of the x-, y- & z- translation values depends on the sign of the value. Updating a positive translation value:

$$translate_{new} = translate_{old} - voxelsize \tag{11}$$

Updating a negative translation value:

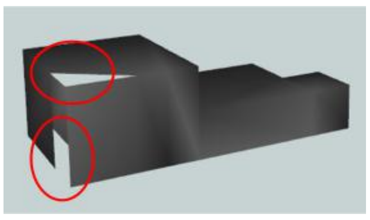
$$translate_{new} = translate_{old} + voxelsize \tag{12}$$

SCAN METHOD

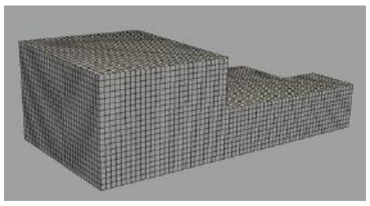
The theory in § 3.2.2 has described that two effective ways of scan conversion are Parity Count or Ray Stabbing. Although Ray Stabbing may have a better repair capability for very defective input containing self-intersections and double surfaces, it does not reconstruct any buildings with a hole or concave buildings with overhangs. Therefore it is considered too limited and Parity Count method will be used instead.

NUMBER OF SCAN DIRECTIONS

The number of scan directions may improve the repair capability, however the processing speed will increase quickly. An example of a CityGML building with a manually added gap is shown. Figure 62 where 6 scan directions are used to correctly produce the volumetric representation of the building.



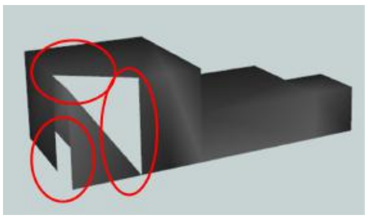
(a) CityGML model with 2 gaps



(b) Correct scan conversion

Figure 62: Correct volumetric representation

However, when one more gap is introduced, a scan conversion in 6 directions is no longer sufficient. Figure 63 illustrates a dent in the model, where gaps are found in 3 out of 6 orthogonal directions.


(a) CityGML model with 3 gaps

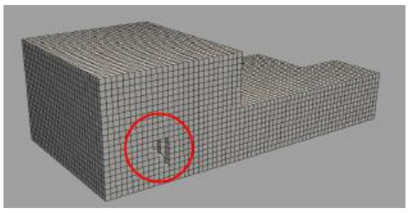
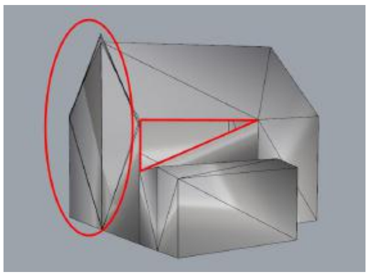

(b) Incorrect scan conversion

Figure 63: Defective volumetric representation due to combination of three gaps

There are two options to produce a correct result for a building such as in Figure 63a. By using a 26-direction scan, the building model can be turned into a correct volumetric representation as shown in Figure 62b. However, this is at the cost of the processing time. The second option is to decrease the voting threshold in certain cases, which will be described in the next paragraph. It is decided that scanning in 6 orthogonal directions will be sufficient unless the input models are severely defective.

VOTING THRESHOLD

Until now all voting thresholds were set at the number of scan directions divided by two. All voxels that are considered as interior have a number of votes higher than the threshold, hence the name majority voting. Since a number of voting directions has been set at 6, the threshold will be 3 meaning that at least 4 votes are needed for a voxel to be filled. However, in case of severe defects (multiple gaps, self-intersections or double surfaces) a threshold of 3 may be too strict to produce a correct volumetric representation. As described in Section 3.2 a closely related algorithm was used by Steuer et al. [2015]. In their work they are choosing to scan in 6 orthogonal directions and set the threshold to $n/2$ (n = number of scan rays). No particular problems have been described in their results. However, when severely defective models are processed unwanted results may occur. Another manually edited example of problematic input is visible in Figure 64. Here a double surface, overshoot and gap are located in the same area of the building. Using the regular 6 directional scan conversion with a threshold of 3 results in an incorrect result.


(a) CityGML model with double surface, overshoot and gap

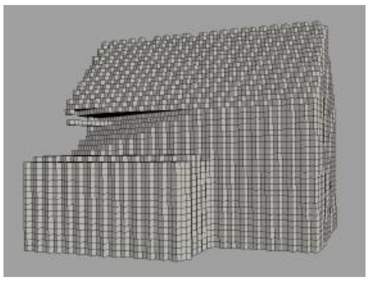

(b) Incorrect scan conversion due to multiple errors

Figure 64: Defective volumetric representation due to combination of double surface and hole

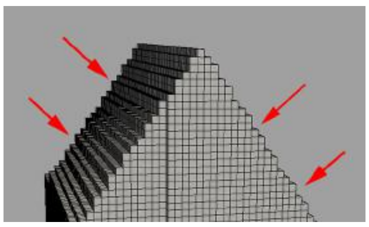
Similar to the previously shown model, lowering the threshold or raising the number of scan directions will produce a correct result. Since these models were manually made to illustrate the limitations of the scan conversion, these parameters can be considered as a detail. In general these combinations of defects will be rare. Additionally, an easier step which does not need an adaption of the parameters is aligning the model.

4.2.2 Possible artefacts

Using the previously described scan conversion may result in three kinds of aliasing artefacts:

- Stair stepping
- Hanging voxels
- 'Saw tooth' effect

The stair-stepping effect (see Figure 65a) is to be expected, as it approximates the interior close to an oblique surface. This is inherent to the voxelization process but may remain visible in a pure binary grid surface reconstruction (see Section 4.3). A related artefact occurs when there are sharp angles on the border of oblique surfaces, which will result in hanging voxels only connected to one other voxel. An example of this is visible in Figure 65b. These are not acceptable since they could result in unexpected output geometry or ambiguous cases in the surface reconstruction process. The artefact which is described as 'saw tooth effect' is a result from the combination of the precision of calculating the intersection point and the almost parallel surfaces. An example is shown in Figure 66.


(a) Stair-stepping artefact

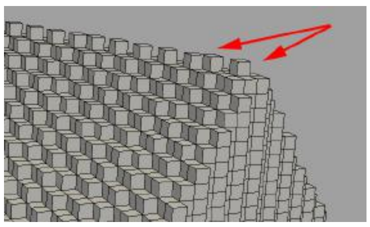
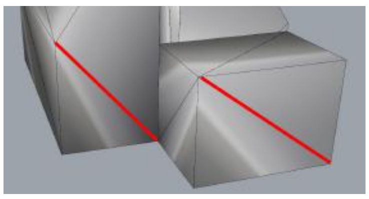

(b) Hanging voxel artefact

Figure 65: Examples of artefacts in the volumetric representation


(a) Nearly parallel surfaces

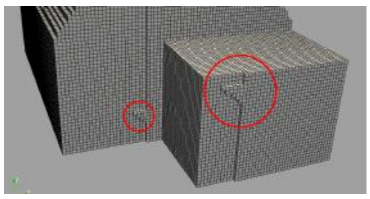
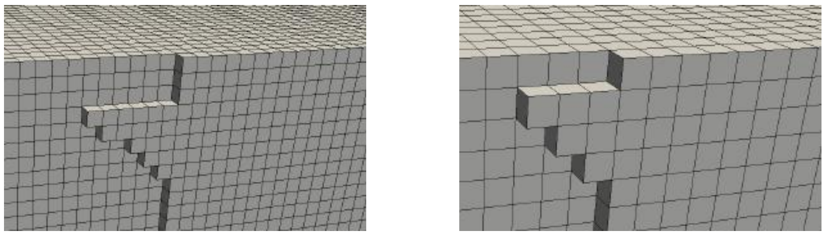

(b) Saw tooth artefact

Figure 66: Example of a saw tooth artefacts in the volumetric representation

DOWN-SAMPLING

An option for decreasing the artefacts is to perform down-sampling on the grid, as described in § 3.2.4. This has been tested on on several of the input models but in general the artefacts were moved but still present. An example of this is illustrated in Figure 67.



(a) Saw tooth artefact in original voxel-grid

(b) Enlarged saw tooth artefact for down-sampled voxel-grid

Figure 67: Result of down-sampling with a $2x2x2$ filter

In case of artefacts in the voxelization, the smoothing does not improve the output voxelization. Applying any down-sampling has the obvious drawbacks of decreasing the resolution, which may lead to a larger geometric shift (depending on the surface reconstruction method). For these reasons the step of down-sampling will not be applied.

4.3 APPROACH 1: MARCHING CUBES

After the voxelization component, a volumetric representation has been acquired. In the following surface reconstruction component, the two approaches come into play. The first approach is based around the Marching Cubes algorithm as described in § 3.4.2,which can be applied on a 3-dimensional grid of binary or scalar values. This section will describe which components are considered to be most suitable for using a voxel-based repair method using the Marching Cubes algorithm. The components can be split up in Morphological operators, Isosurface extraction: Marching Cubes and Post processing.

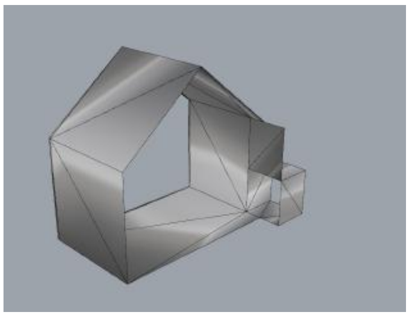
4.3.1 Morphological operators

In order to prepare the volumetric representation for the Marching Cubes algorithm there are three steps needed to ensure a valid result. These are the selection of a single connected component, the removal of possible gaps and the removal of artefacts. Because Marching Cubes only takes a binary grid as input, grid can be 'molded' into the desired shape without any consequences. This makes morphological operators very appropriate for this approach.

SELECTING CONNECTED COMPONENT

The selection of a single connected component is needed to ensure valid output. The UnionFind algorithm as described in § 3.3.3 was implemented in order to select the largest connected component. It should be noted that cases of multiple connected components are only rarely encountered, but are possible in theory. In the event of a very defective voxelization, several small connected components might be detected. This knowledge can be used to detect a possible errors. Optionally the user can be notified or the voxelization parameters can be adjusted to process exceptional cases.

Figure 68 shows the only case that was encountered where the number of connected components was an issue. It is shown in Figure 68b that the two


(a) Severely defective building with edge adjacent box

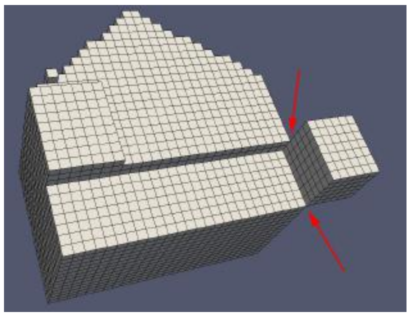
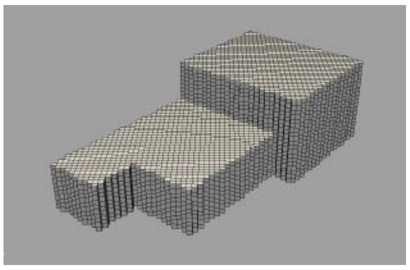

(b) Volumetric representation only edge connected

Figure 68: Building which requires the connected components selection

components are only edge connected. When this is left like this, it may result in several ambiguous cases such as depicted in Figure 39 in § 3.4.2.

GAP REMOVAL

Although rarely encountered, in theory it is possible that gaps are present inside the volumetric grid. Removing these is essential for valid output. To do so Distance Mapping is applied by [Nooruddin and Turk, 2003] as was described in § 3.3.2.


(a) Volumetric representation before distance mapping

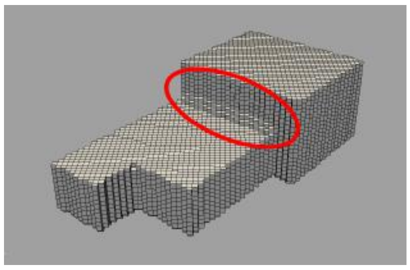

(b) Volumetric representation after distance mapping

Figure 69: Corners rounded off after distance mapping

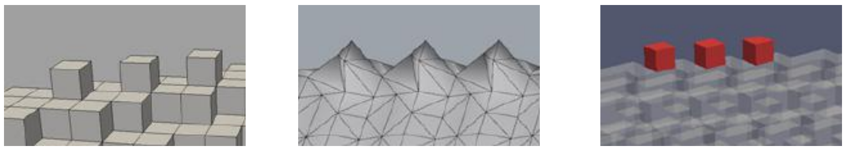
Figure 69 shows that the application of distance mapping results in rounded off corners. This has a large impact on the surface reconstruction and is thus not acceptable for 3D City model repair. For these reasons the slower but more effective method of removing holes by looking at the connected components of the inverse grid, as described in § 3.3.3 is applied.

REMOVING ARTEFACTS

As was stated before, the removal of hanging voxels is important to avoid the creation of artefacts and avoid any marching cubes ambiguities. To detect these voxels, a filtering based on the 6-adjacency is applied (see § 3.3.1 for the theory). An example of the filtering is illustrated in Figure 70.

4.3.2 Isosurface extraction: Marching Cubes

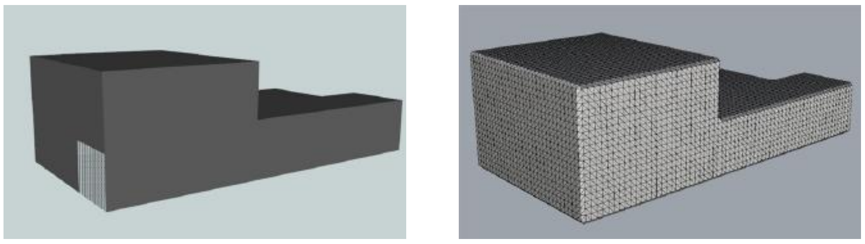
Once the desired volumetric representation has been created, the binary grid can be converted to a triangle mesh. At this point the input volume



(a) Hanging voxels on a sharp roof (b) Marching Cubes bumpy mesh result (c) Selection based on 1/6-adjacency

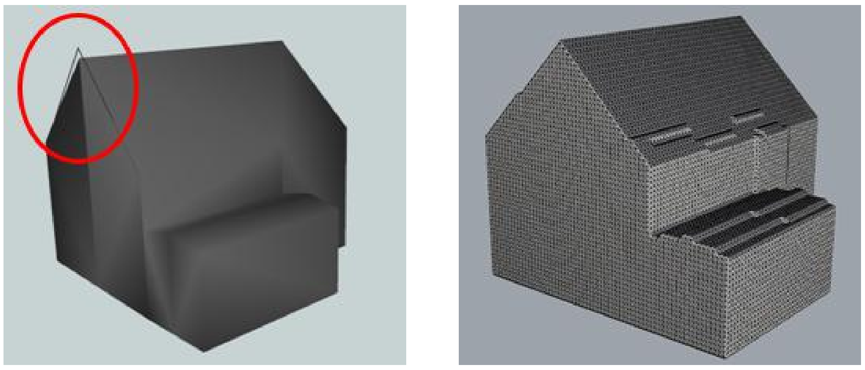
Figure 70: Filtering of hanging voxels

is guaranteed to be manifold without any inner gaps. The Marching Cube algorithm is applied as described in § 3.4.2. An overview of the 15 Marching Cubes cases is illustrated in the appendix (see Appendix A). Some illustration of the resulting triangle mesh are shown Figure 71 and Figure 72 to give an indication of the repair process so far.



(a) 3D-model with a gap (b) Result after Marching Cubes

Figure 71: Example of Marching Cubes output for a CityGML building with a gap



(a) 3D-model with an overshoot (b) Result after Marching Cubes

Figure 72: Example of Marching Cubes output for a CityGML building with a double surface and an overshoot

These examples show that the defects present in the original model have been removed. However, all sharp corners have been rounded off and the oblique surfaces could only be approximated. In the post-processing component, the following steps for edge sharpening and detriangulation will be discussed.

4.3.3 Post processing

A characteristic of the Marching Cubes approach is the rounded off corners. The application of an edge sharpening algorithm will be described. In order to turn the resulting triangle mesh back into a polygonal model, a detriangulation is needed. The ways of doing so will be discussed.

Edge Sharpening

In order to sharpen the rounded off corners, an edge sharpening algorithm can be applied. The theory has been described previously in § 3.5.1 and is based on the work of [Attene et al., 2003]. Since we know the triangle mesh is resulting from the Marching Cubes algorithm, some assumptions can be made regarding the geometry. With this idea in mind, a possible adaption of the original edge sharpening method was explored. In stead of classifying certain edges, specific triangles that should be sharpened can be found by looking at the neighboring triangles' normal vectors. The initial set of triangles which were detected are illustrated in Figure 73, distinguishing corner and chamfer triangles and separating convex and concave cases.

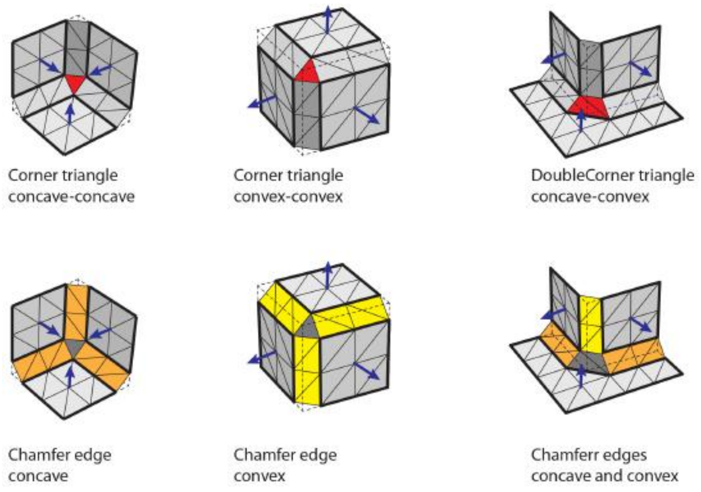
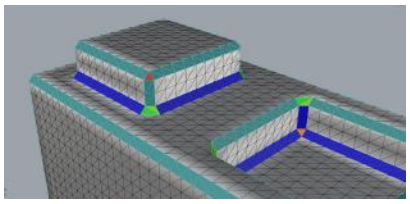


Figure 73: Edge sharpening triangle cases

This method was first applied on a manually made object to test the specified cases. An illustration of the detection and rebuilding of rounded off triangles is shown in Figure 74.


(a) Classification of the triangles that need sharpening

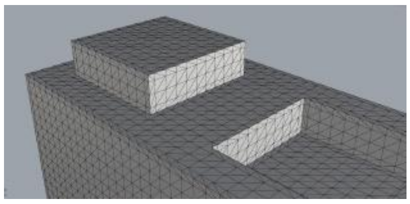
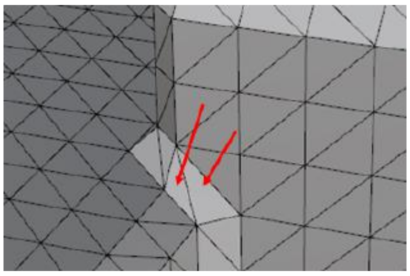

(b) Result after triangle based edge sharpening

Figure 74: Result of triangle based edge sharpening on a manually made object

Although this shows a successful sharpening of the input geometry, the classification of triangle cases has a major drawback, as each possible case needs to be taken into account. When this method is applied on real buildings, many more triangle cases can be found. Figure 75 shows some examples of the additional triangle configurations that need to be taken into account.
Granting most of these cases are quite easy to distinguish, implementing all possible cases would be quite a task. For any unexpected case, the


(a) Example of triangle configuration

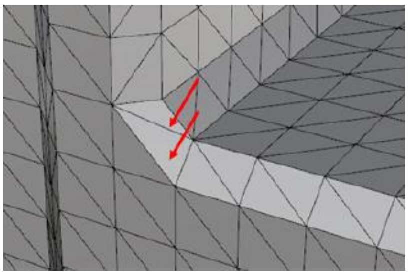

(b) Example of triangle configuration

Figure 75: Two examples of more complex triangle configurations

resulting triangle mesh is very likely to be non-manifold. An example of the edge sharpening result including an unsupported triangle case is shown in Figure 76.

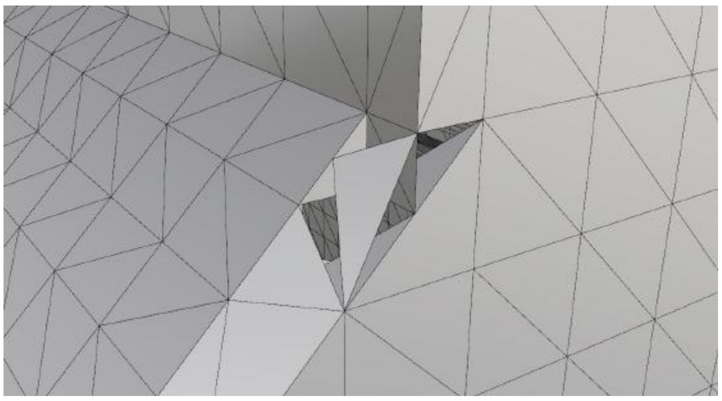
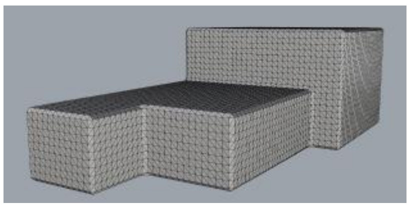


Figure 76: Unsupported triangle cases returning a non-manifold mesh

Although the method of classifying triangles is viable, many cases are possible. Finding all possible triangle configurations is important, since any missed cases will result in a defective triangle mesh. For this reason it was decided to finally apply the edge sharpening algorithm such as described in the theory (see § 4.3.3). Figure 77 shows an example of a sharpened building acquired by applying an edge based edge sharpening.


(a) triangle mesh before edge sharpening

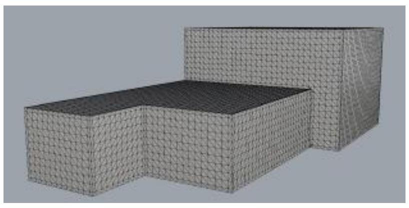
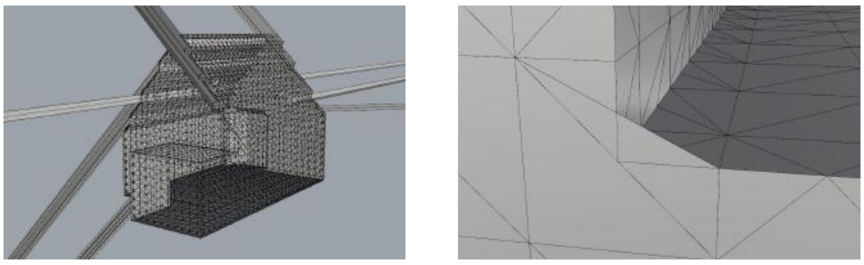

(b) triangle mesh after edge sharpening

Figure 77: Performing edge sharpening to improve the results in Approach 1

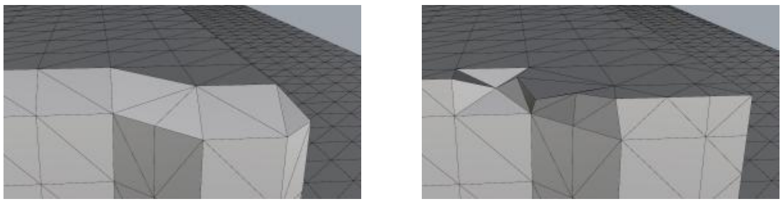
However, in the theory it was described that some unexpected cases may take place. The first one being the occurrence of *long edges*, which are being created when the corresponding planes are parallel or do not intersect in one point. This leads to vertex positions very far away from the original building. The second case occurs where *multiple sharp corners* are joined together (which is quite common in building models). In these cases a triangle is created which turns the mesh non-manifold. An illustration of these defects is shown in Figure 78.


(a) Long edges for incorrectly solved plane intersections

(b) A non-manifold triangle where three sharp corners meet

Figure 78: Performing edge sharpening to improve the results in Approach 1

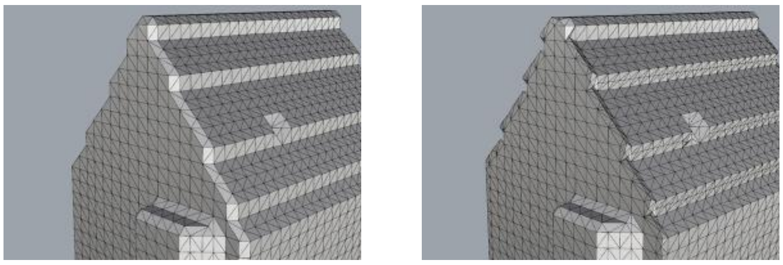
By checking the edge lengths, long edges can be detected. These vertex positions are now moved to the middle of the edge, as is recommended in the work of Attene et al. [2003]). An additional filter for the non-manifold cases of multiple corners removes the non-manifold triangles. Another unexpected case, which has not been reported in the paper is in the vertex calculation for edges that are placed within a stair stepping effect, which is illustrated in Figure 79. This effect has something to do with the weighted sum of normals of this particular configuration. This case can be detected by filtering for this combination of surface normals in the adjacent triangles.


(a) Mesh before edge sharpening

(b) Example of sticks

Figure 79: Mesh after edge sharpening

For orthogonal buildings, the edge sharpening clearly improves the resulting geometry (see Figure 74b and Figure 77b). An illustration of the result of edge sharpening for a building with oblique surfaces in shown in Figure 80.


(a) Mesh before edge sharpening

(b) Mesh after edge sharpening

Figure 80: Comparison between a rounded and sharpened oblique surface

A comparison of the input model and sharpened result, is not necessarily in favor of the edge sharpening. It is visible that the sharpened model contains artefacts at the locations of stair stepping. In addition the geometry seems rather unintuitive, being sharpened while still approximating the original model. Finally it was decided to not run the edge sharpening in the repair process, since it increases the chance of unexpected results while its improvement of the geometry is arguable.

Detriangulation

To convert the triangle mesh back to the CityGML data format, a detriangulation is needed. Two methods have been researched; quadric edge mesh decimation and a triangle segmentation method based on knowledge of the Marching Cubes output. Arroyo Ohori [2010]

QUADRIC EDGE MESH DECIMATION

In order to reduce the number of triangles and simplify the geometry, Nooruddin and Turk [2003] use an edge decimation method developed by Garland and Heckbert [1997]. The theory of this process is described in § 3.5.2. It has been applied using the *Quadric Edge Collapse Decimation* tool in MeshLab. This tool is capable of slightly simplifying geometry in order to reduce the triangle count. However it is rather unpredictable in its selection of edges.

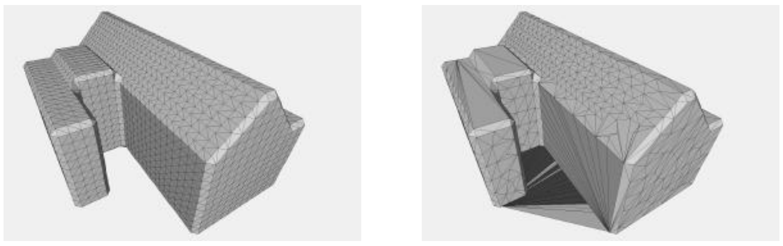

(a) Input model (b) Defects in resulting model

Figure 81: The polygonal model turned non-manifold after edge decimation

Figure 81 shows an example of a wrongfully created ground plane. Although some of the vertices may be close to each other, merging them is obviously unwanted in this case.

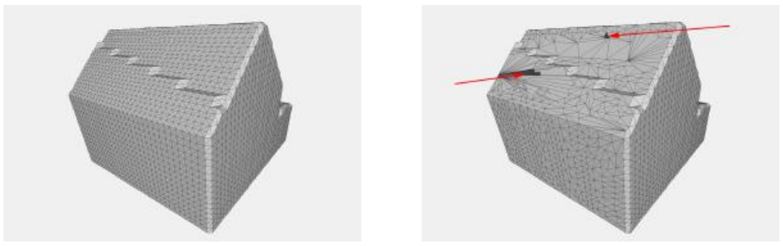

(a) Input model (b) Overlap in resulting model

Figure 82: The polygonal model turned non-manifold after edge decimation

A second example of a triangle mesh turning non-manifold is shown in Figure 82. In this case the number of triangles on the right side has not been simplified, yet the geometry on the roof is already self-intersecting. Since the main strength of a voxel-based repair method is its robustness, using an unpredictable method for triangle count reduction is not suitable.

TRIANGLE SEGMENTATION

As an alternative to the edge decimation method, I have developed a detriangulation method specified towards Marching Cubes output. Its aim is not to simplify or improve the geometry, but simply to reduce the triangle count. Because we know Marching Cubes can only produce triangles in 26 different directions, it is possible to segment all triangles and group them based on their normalized normal vector. The next step is to merge all neighboring triangles which are pointing in the same direction. This process is explained in algorithm 4.2.

Algorithm 4.2: DETRIANGULATION

```
Input: A triangle mesh $M$ and 26 empty segments
Output: A polygonal model
1  for triangle $\tau$ in $M$ do
2      compute triangle normal
3      add $\tau$ to corresponding segment
4  end
5  for every segment do
6      keep all directed edges that appear once
7      sort all incident directed edges
8      while edges with same direction are detected do
9          for every $edge_n$ $(v_1v_2)$ and $edge_{n+1}$ $(v_2v_3)$ do
10             if $v1v2$ has same direction as $v2v3$ then
11                 create new edge $v_1v_3$
12                 delete both edges $v1v2$ and $v2v3$
13                 delete vertice $v2$
14             end
15         end
16     end
17 end
18 order all polygons on descending area for polygon in set do
19 end
20 if $polygon_n$ contains $polygon_{n+1}$ then
21     add $polygon_{n+1}$ to interior of $polygon_n$
22 end
```

Figure 83a shows the first step; selecting triangles pointing in the same direction. The second step is selecting only the edges that appear once, shown in Figure 83b. The third step is to connect all incident edges, and remove all vertices that are not positioned on a corner (see Figure 83c). The final step is to select any holes as interior rings.

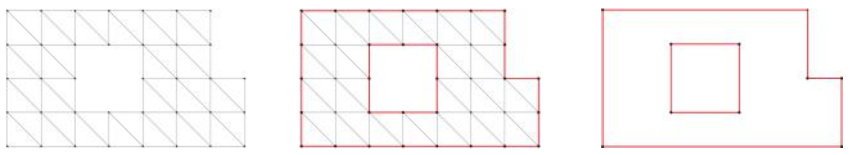

(a) Step 1: Select a set of triangles with same orientation
(b) Step 2: Find and merge consecutive edges
(c) Step 3: Detect interior rings for each polygon

Figure 83: The three steps of the detriangulation process

The results of these steps are visible in Figure 84.

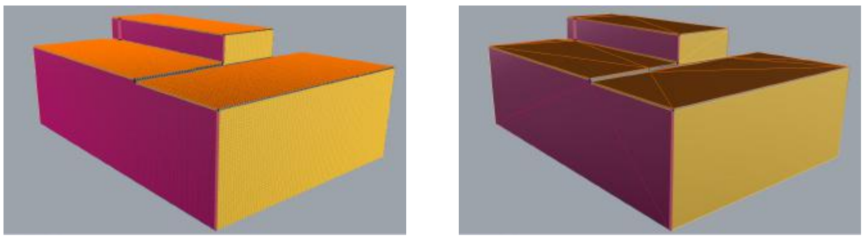

(a) Triangle mesh segmented on normal vector
(b) Polygonal mesh result after detriangulation

Figure 84: Result of the detriangulation step

4.3.4 Evaluation

After developing the method, some advantages and drawbacks of this approach became clear.

Advantages:

- The isosurface extraction is based on a binary grid only, which means that all morphological operators can be performed on the binary grid.
- The Marching Cubes results for a given binary grid are very predictable. Since triangles are created by simple cube configurations, no unexpected cases are encountered.
- Any defective building model which is converted to a manifold voxelization without any holes, is guaranteed to be converted to a manifold mesh (taking into account the possible ambiguities of the original Marching Cubes algorithm).

Drawbacks:

- Oblique surfaces are approximated with a stair stepping effect. (Note that a using a distance field would remove this issue, see § 3.4.2)
- Corners are rounded off which may be sharpened by using an edge sharpening technique. This however, gives the possibility of unexpected results appearing in the output mesh (long edges & weird stair stepping effect).

4.4 APPROACH 2: DUAL CONTOURING

The second approach for a voxel-based repair method is focused on the surface reconstruction technique of Dual Contouring as explained in § 3.4.3. This section will describe the decisions that have been made in the development of this approach.

4.4.1 Morphological operators

As described in § 3.3.4, morphological operators are most useful when applied for a pure binary grid reconstruction method. The Dual Contouring method is based on finding the original grid edges which connect the interior and the exterior of the grid. By performing morphological operators, binary voxel values will be changed. In doing so, some grid edges connecting the interior (value = 1) with exterior (value = 0) are not necessarily intersecting with the original model anymore. For this reason no use can be made of Distance Mapping or Adjacency Filtering. Only the steps of selecting a single connected component and filling holes by flood filling the inverse grid are used, in a similar way as described in § 4.3.1 in Section 4.3.

4.4.2 Isosurface extraction: Dual Contouring

The second approach consists of an adaption I made based on the original Dual Contouring method, as described in § 3.4.3 in Chapter 3. In the theory it is described how the contour can be shaped by creating a triangulation of the dual vertices. Here the position of dual vertices is computed by checking the intersections and corresponding surface normals between the input model and all grid edges of the cube. However, since we are applying this method on defective geometry, not all intersections can be found in all cases. In case of a hole or overshoot, no intersections or too many intersections will be found.

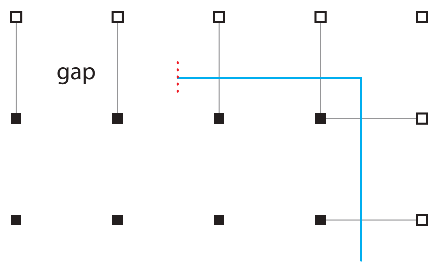


Figure 85: DC gap model

A clear example of a case where intersections cannot be found, is when the input model has a hole. A diagram of this is shown in Figure 85. The grid edges connecting a white and a black voxel are depicted as black lines. It is clear that the grid edges positioned at a gap, do not intersect with the original building model which is shown in blue. The reason that Dual Contouring can still be applied to the defective input, is because we can fall back on the volume defined by the binary grid.

Defective Input Handling

In this approach I try to overcome this issue, by computing the dual in an alternative way when not all Hermite data is available. By distinguishing three cases a decision can be made on how to compute the dual vertex. In descending precision these are:

- *Minimized QEF*, when all intersections and normals are found compute the QEF.
- *Mass point*, when only some intersections are found, compute the average of the intersection points that have been found.
- *Cube center point*, if no intersections are found at all assign the center of the cube.

In case not all intersections are found, the mass point can be computed as the average of intersection points that have been found. When no intersections are found but the cube is known to intersect the original model, we can simple use the center of the cube since no additional information is available. An illustration of these three options is shown in Figure 86.

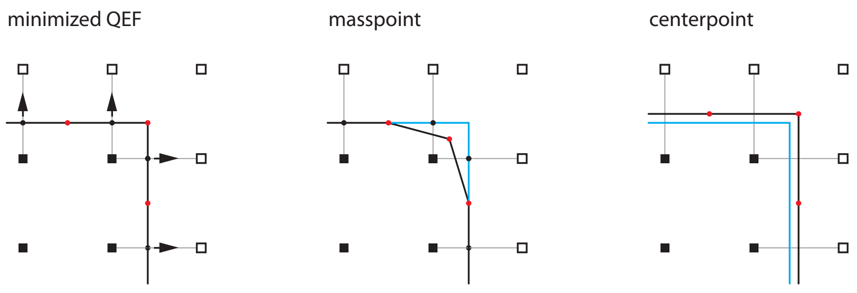
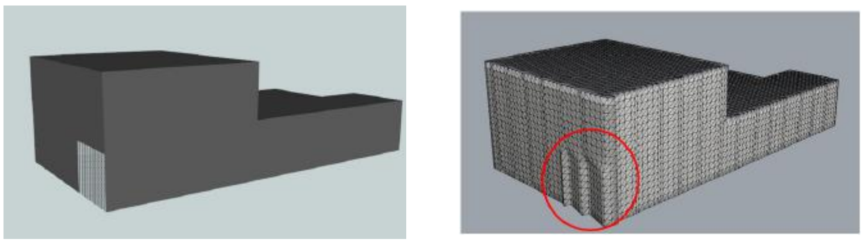


Figure 86: Three options for computing the dual vertex

The idea behind this approach is that we use the intersections with the original model where possible, and fall back to the voxel representation in other cases. Figure 87 shows an example of an accurate surface reconstruction of the correct parts of the model. At the location of the gap, the triangle mesh is representing the voxel representation since no intersections can be found.



(a) Input model with limited hermite data due to a gap

(b) Approximation of the original surface at gap

Figure 87: Result of the proposed method on a building with a gap

algorithm 4.3 shows how different options for computation of the dual vertex are selected.

Algorithm 4.3: Dual Contouring
adaptation of [Ju et al., 2002; Schaefer et al., 2002]

```
Input: A binary grid $G$ and a (possibly defective) polygonal model $M$
Output: A triangle mesh
1  for voxel $V_{ijk}$ in $G$ do
2      check grid edges $G_e$ for sign change with $V_{i+1jk}$, $V_{ij+1k}$ and $V_{ijk+1}$
3      for every grid in $G_e$ do
4          if grid edge intersects with $M$ then
5              store intersection point $p_i$ and surface normal $n_i$
6              tag 4 grid cells $G_c$ neighboring the grid edge $G_e$
7          end
8      end
9  end
10 for every tagged grid cell $G_c$ do
11     if all hermite Data available then
12         $G_c$'s dual vertex $v$ is computed by QEF
13     end
14     else if some intersection available then
15         $G_c$'s dual vertex $v$ is the mass-point
16     end
17     else if no intersections available then
18         $G_c$'s dual vertex $v$ is the center-point
19     end
20 end
21 for every grid edge $G_e$ in $A$ do
22     create two triangles connecting the dual vertex $v_1, v_2, v_3 \& v_4$ of
       the neighboring grid cells $G_c$
23 end
```

After testing the method as described above on several building models, 3 important issues were encountered;

- Incorrectly solved QEF vertices
- Misplaced voxels finding no intersections
- Non-manifold Self-intersections

Descriptions and possible solutions for each of these issues are described below.

INCORRECTLY SOLVED QEF

Theoretically, the computation of the QEF results in an intersection point between the planes which are defined by the hermite data (= intersection point & normal vector). However, in some cases the computation of the QEF causes exceptions where the dual vertex position is calculated very far away from the actual building. This has to do with the precision of the intersections, which causes the QEF to search for the intersection between two nearly parallel planes. An example of this is shown in Figure 88.

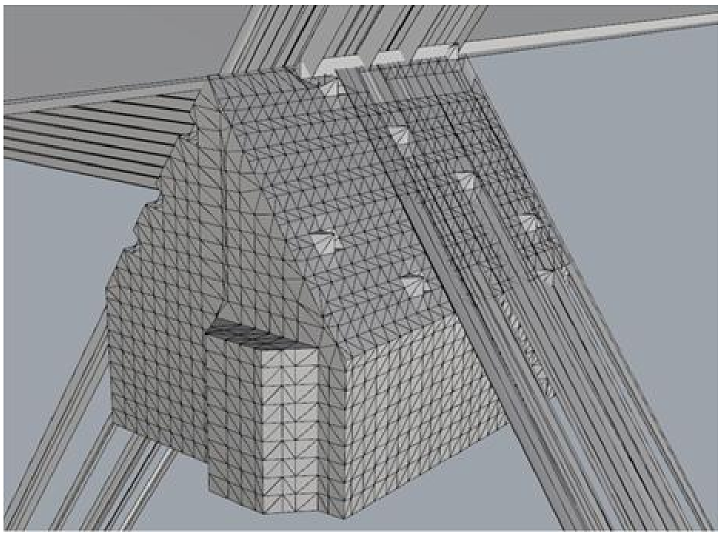


Figure 88: Several incorrect solutions for the QEF computation

These exceptions can be easily detected by performing an edge length check and the dual vertex can be replaced by either the mass point or cube center. However, this means that this part of the triangle mesh will not be able to rebuild any sharp features.

MISPLACED VOXELS

A second issue that is encountered has to do with the detection of grid edges that are intersecting with the original building model. We are using the voxel representation to determine where these grid edges should be located. In theory, every grid edges which connects an interior voxel with an exterior voxel, should intersect with the original model. In practice however, *misplaced voxels* will have grid edges which do not find an intersection. When the intersection cannot be found, the QEF cannot be computed. Therefore either the mass-point or the cube center will have to be used instead. Two diagram of the effect of this error are shown in Figure 89 and Figure 90.

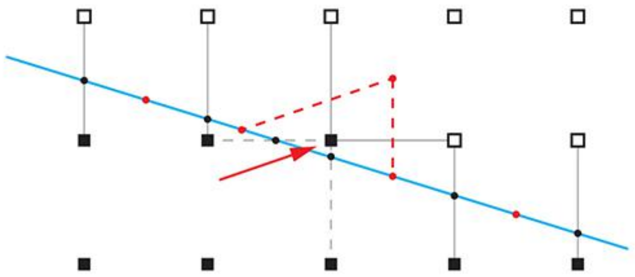


Figure 89: Assigning the mass-point in case of a misplaced voxel

The misplaced voxel is indicated with the red arrow. Note that this voxel is positioned at the wrong side of the blue line, which represents the original building model. Now this misplaced voxel is connected to two white voxels through a grid edge. On both of these grid edges an intersection with the original model is expected but not found. The dotted red lines show the result after assigning either the mass-point or the cube center, which in both cases results in artefacts.

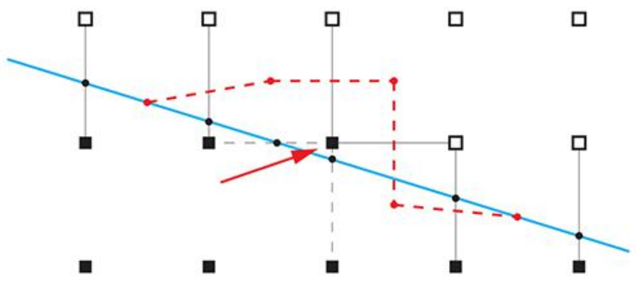
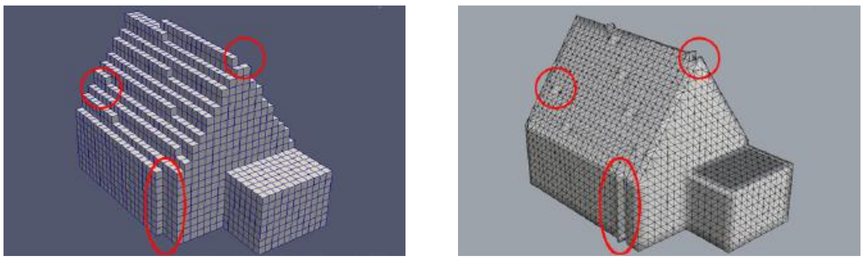


Figure 90: Assigning the cube center in case of misplaced a voxel

The result of assigning cube centers in case of misplaced voxels is shown in Figure 91. By observing both the voxel representation and the output model it is visible that voxels on the end of a slice are likely to be misplaced. During the processing of these voxels, it is likely that artefacts are generated in the resulting triangle mesh. These artefacts are highlighted in Figure 91b.

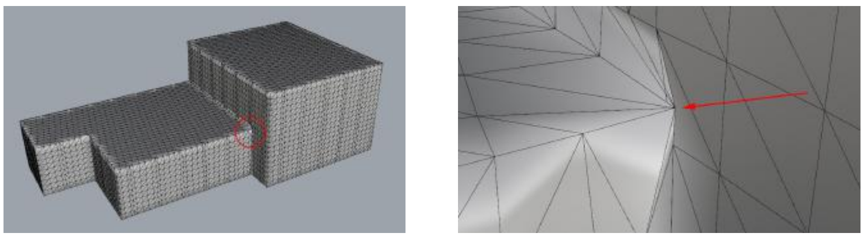


(a) Input voxel representation **(b)** Artefacts after Dual Contouring

Figure 91: Effect of misplaced voxels on isosurface extraction

SELF-INTERSECTIONS

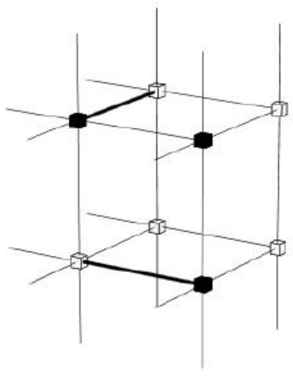
The third issue is caused since the current implementation is based on the original Dual Contouring method [Ju et al., 2002; Schaefer et al., 2002]. Although this method is guaranteed to result in a closed mesh, in case of certain concave configurations of cubes, the resulting mesh may be self-intersecting. An illustration of this is shown in Figure 92.

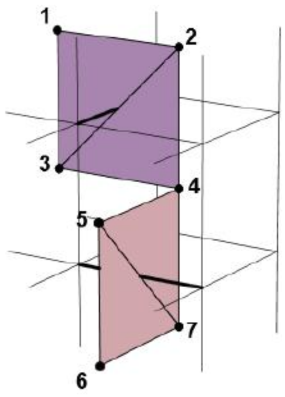


(a) Result after Dual Contouring **(b)** Example of a self-intersection

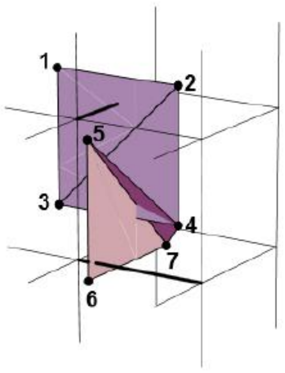
Figure 92: Self-intersections in the output triangle mesh

This problem is inherent to the original implementation of the Dual Contouring algorithm. An adaptation called *Dual Marching Cubes* was developed by Schaefer and Warren [2005], allowing for multiple vertices being placed in a cube. However, also this implementation allows for self-intersections. Ju and Udeshi [2006] developed a method which either changes the detriangulation or adds extra vertices to a cube in order to prevent self-intersections. Figure 93a and Figure 93b show a configuration of binary values in the voxel grid and a possible triangulation created by Dual Contouring without self-intersections. However, for the same configuration the dual vertices may be placed on different positions, which may cause self-intersections as shown in Figure 93c. Figure 93d shows how their method changes the triangulation, preventing the triangles to intersect.

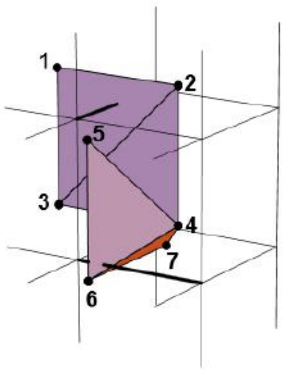

(a) Configuration of binary values in the voxel grid



(b) Triangulation without self-intersections



(c) Triangulation with self-intersection



(d) Changed triangulation without self-intersections

Figure 93: Example of self-intersection and possible solution by changed triangulation [Ju and Udeshi, 2006]

Another adaption of the Dual Contouring algorithm is proposed by [Manson and Schaefer, 2010]. Here, dual vertices are re-positioned towards the iso-surface which is followed by a topology test to preserve the original topology. An implementation of either of these algorithm should theoretically solve the problem of self-intersection.

Although the first two issues (incorrectly solved QEF and misplaced voxels) do not create invalid building models, they very often do cause undesired artefacts in the output model. More importantly, the occurrence of self-intersections means that the current implementation of this approach is not capable of repairing 3D City models. Before this second approach can be validated, more research is needed for guaranteeing a manifold output, reducing the creation of artefacts and finally the detriangulation of the model.

4.4.3 Evaluation

After testing the second approach it is clear that the current implementation is not capable of 3D City Building model repair. It does however show the potential of the *adaptive contouring methods*, which makes use of information of the input model.

Advantages:

- The major advantage is the exact surface reconstruction (oblique surfaces and sharp features) of the original model when the Hermite data is available.

- My proposed adjustment to Dual Contouring allows for removing overshoots and filling holes by falling back to the binary grid.

- Any attributes of the input model can be directly linked to triangles in the mesh by using an edge based attribute preservation (see § 4.5.3 in Section 4.5).

Drawbacks:

- The original Dual Contouring algorithm results in self-intersecting triangles.

- Incorrectly solving the QEF can be filtered but this results in non-sharp features.

- Misplaced voxels will not find all intersections, meaning that the QEF cannot be solved and artefacts are created. Although this does not create non-manifold geometry, the resulting artefacts are unwanted.

- In case of filling gaps, the resulting surfaces will be jagged since they are representing the binary grid.

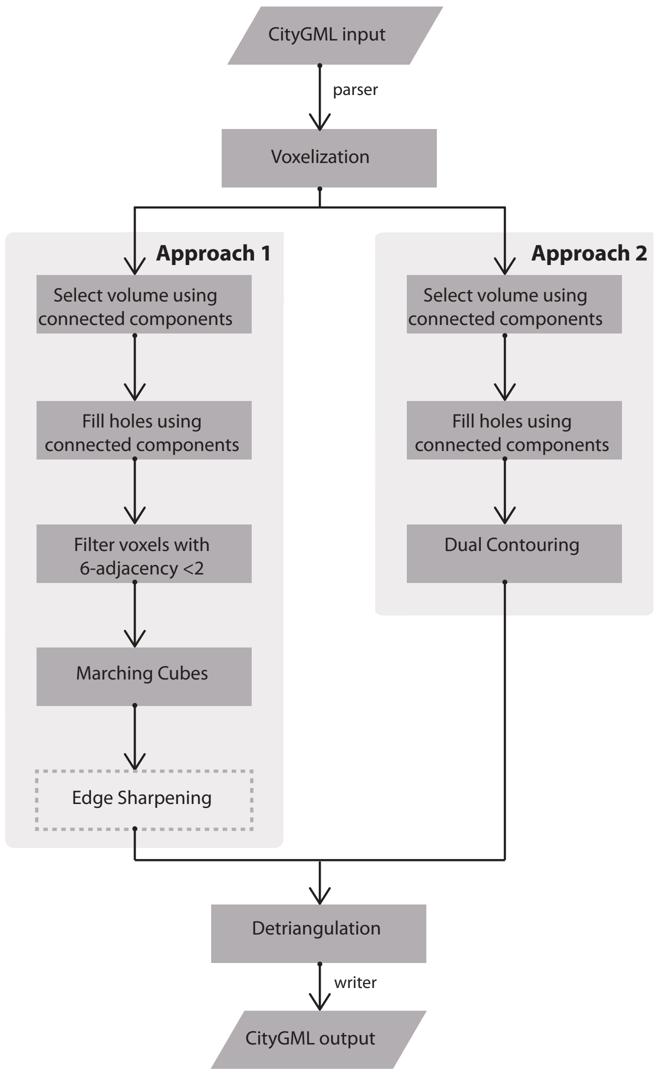

CityGML input
parser
Voxelization
Approach 1
Select volume using connected components
Fill holes using connected components
Filter voxels with 6-adjacency <2
Marching Cubes
Edge Sharpening
Approach 2
Select volume using connected components
Fill holes using connected components
Dual Contouring
Detriangulation
writer
CityGML output

Figure 94: Flowchart of the applied approaches

An overview of the selected components for each of the approaches is shown in Figure 94. In the first approach, the Edge Sharpening component is dotted as it is not essential for the repair process and the risk of producing defective output is increased (see § 4.3.3). In the second approach, it should be noted that more research is needed on the area of Dual Contouring to determine its suitability for 3D City Model Repair. The detriangulation component has only been implemented for the first approach. In Chapter 5 the repair capability of both approaches is tested.

4.5 PRESERVATION OF ATTRIBUTES

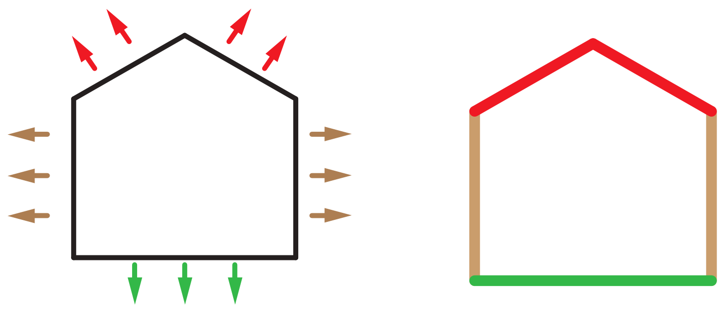
In many 3D City models, attributes are an important part of the data. The possibilities of storing semantics in CityGML are described in § 2.1.3. A drawback of using a voxel-based repair method it that the surface based attributes are not directly related to the resulting triangle mesh. Depending on the surface reconstruction technique, different approaches may be used to preserve attributes throughout the process. Three ways of preserving or restoring the attributes will be described here:

- Geometry based
- Voxel based
- Binary Grid edge based

This section will explain the method and suitability for each of these approaches.

4.5.1 Geometry based attribute determination

The most basic way of attribute preservation is by performing all the geometric repair steps while disregarding any surface information, and only restoring the attributes afterwards. This way of attribute determination has been researched in more detail by Boeters [2013] and Diakité et al. [2014]. By looking at the orientation and position of surfaces, some assumptions can be made. In case of CityGML data, every building surfaces should be stored as either Ground Surface, Wall Surface or Roof Surface. An examples of attribute determination could be that a surface with a normal vector with a z-component = 0 should be considered as a Wall Surface. A diagram of this principle is shown in Figure 95.



(a) Mesh is segmented based on surface normals

(b) Mesh with attributes

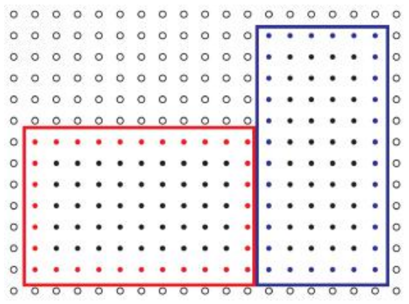
Figure 95: Diagram of geometry based attribute preservation

An advantage of this process is that the implementation of this approach is easy, and no customization of any of the geometric steps is needed. In case of missing geometry or attributes in the original model, attributes can still be determined. This makes a strong case for returning reasonable output for very defective input buildings since often the semantics of a defective model should not be considered more reliable than the geometry. A drawback is that the original surface attributes are lost and only restored based on assumptions. This means that only general attributes (such as

Ground Surface, Wall Surface & Roof Surface in the case of CityGML) can be restored. Also this approach is not appropriate for models with high level of detail (LoD 3 and higher) nor in cases such as roof overhangs. Additionally, any artefacts created by a voxel-based repair method may be modeled as small surfaces pointing in unexpected directions, which has to be taken into consideration.

4.5.2 Voxel based attribute preservation

A more precise way to preserve any surface related attributes is to store them throughout the voxelization process. Individual voxels may be 'tagged' in order to indicate if they are close to the original surface. By checking the distance for every voxel to its scan line intersections, the voxels close to the original surface can be separated. Any voxels that are within a threshold distance of the intersection, can inherit the attributes of the corresponding surface. The theory of this is described in Section 4.2. By storing all surfaces in a table and giving them an ID, voxels can be tagged for their corresponding attributes. This way a 3D grid can be created, containing both binary values and a surface ID. Depending on the surface reconstruction method, these voxels can be translated into triangles. This process is illustrated in Figure 96a. On a higher level, any building parts can be separately voxelized within the same bounding box. This allows for a distinction of building parts in the voxel representation. An illustration of this principle is shown in Figure 96b.


(a) Combined geometries

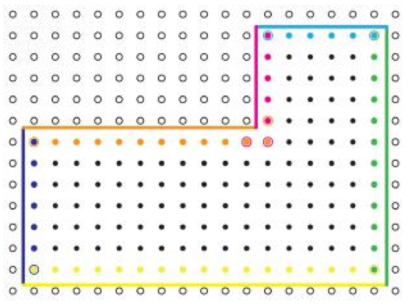
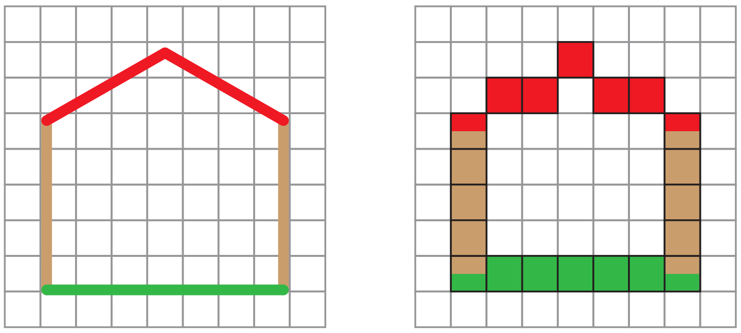

(b) Surface attributes

Figure 96: Attribute preservation per building part and per surface

The main advantage of this approach is that the initial surface attributes can be coupled to voxels. This can be done for building parts (e.g. House & Garage), the boundary surfaces (e.g. Roof Surface, Ground Surface & Wall Surface) and for individual surfaces (e.g. material or texture). However, this approach has some other drawbacks. To convert the voxel attributes into the final polygon model, customization of all the intermediate processing steps is needed. In case of Approach 1 for example, this means that the adjacency filtering, Marching Cubes, edge sharpening and detriangulation should all be adapted which is not trivial. Also when any surfaces are missing in the input model, naturally no information is available for the the neighboring voxels. Additional, on the locations where a Wall Surface meets a Roof Surface, the corner voxels will receive two tags. In order to determine the correct attribute, some geometry based assumptions are likely to be needed again. A schematic representation of the double tags is shown in Figure 97.



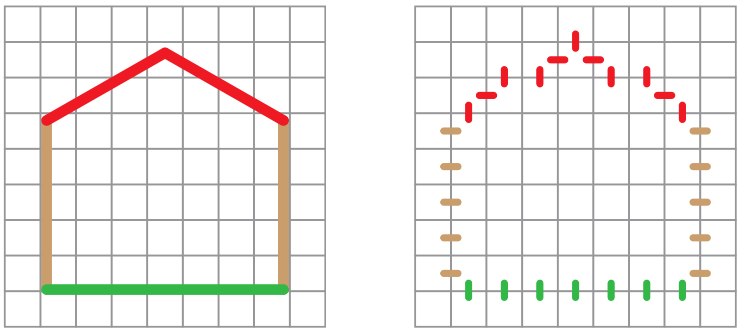
(a) Original input model with attribute information

(b) Voxels 'tagged' with attributes information

Figure 97: Diagram of voxel based attribute preservation

4.5.3 Grid edge based preservation

The third way of preserving attributes, is to store them per grid edge. This approach is only suitable for the surface reconstruction methods that are using grid edges of the original model. These methods have been described in Section 3.4 in Chapter 3. Dual Contouring, Pressing & Dual Marching Cubes are examples of surface reconstruction methods that are based on grid edges. Each of these methods store grid edges that are intersecting with the input model. This means that each of these intersecting grid edges can be easily related to one of the original surfaces. Therefore, every grid edge connecting the interior and the exterior can be tagged with an attribute value. A diagram of this principle is shown in Figure 98.



(a) Original input model with attribute information

(b) Grid edges with attribute information

Figure 98: Diagram of grid edge based attribute preservation

An advantage compared to voxel-based attribute preservation is that multiple tags (i.e. both Roof Surface and Wall Surface) are not possible. Instead, the horizontal grid edge will be tagged as a Wall Surface and the vertical grid edge will be tagged as a Roof Surface. This is especially useful in a method such as Dual Contouring, where every intersecting grid edge has a direct relation with two resulting triangles. This means the edge attribute can simply be added to a single triangle. A problem however, is that sometimes a grid edge connecting interior and exterior does in fact not find an intersection with the original model. This issues is described in Section 4.4, earlier in this chapter. In case an edge does not find any surface, it will result in geometric artefacts and in no attribute information. As described before, this method can only be used for surface reconstruction methods which are based on finding grid edge intersections with the original model. For instance, this makes it unsuitable for Marching Cubes implementation.

4.5.4 Recommendation

Based on the evaluation of the three approaches a recommendation is given for the preservation of attributes.

APPROACH 1: MARCHING CUBES
For the Marching Cubes approach, a voxel based attribute preservation seems suitable, but the conversion of voxel attributes to triangle attributes involves customization of all intermediate steps (adjacency filtering, Marching Cubes, edge sharpening & detriangulation). To preserve the attributes throughout this process, geometric based decision will have to be made regardless. Therefore a fully geometry based attribute preservation is recommended for building models with a lower level of detail (LoD 1 & 2).

APPROACH 2: DUAL CONTOURING
For Dual Contouring the edge based voxel preservation is recommended. Every edge attribute can be easily translated into triangle attribute. In case of no intersection found, neighboring grid edges may be checked for any attribute values. In case of small holes this may solve the problem. In case of large gaps in the input model, an assumption on the attributes can be done by using a geometry based attribute preservation.

In general, one could argue that data sets with a high number of errors need a simple geometry based attribute determination since the data quality of the input model is low. Especially for dataset with a low LoD a complex attribute preservation seems unnecessary. This decision will depend on the specific dataset.

5 IMPLEMENTATION AND EXPERIMENTS

In this chapter both approaches (Marching Cubes and Dual Contouring) are tested on set of buildings from existing data sets. The results are evaluated quantitatively by measuring the repair success rate and qualitatively by analyzing the geometric errors between input and output model. Finally, an evaluation of a voxel-based repair method will be given, comparing the results of the validation with the known capabilities of the Shrink-wrapping method.

5.1 IMPLEMENTATION

The two approaches as described in the Chapter 4 have been implemented in Python using the Numpy and VTK modules. The developed code for the described components is available at `https://github.com/dtmulder/VoxelRepair`. FME and CityGML2OBJ (made by Filip Biljecki and available at `https://github.com/tudelft3d/CityGML2OBJs` were used for triangulating the CityGML data. The component of Quadric Edge Mesh Decimation was tested by using Meshlab, as well as the geometric comparison using the Hausdorff distance. Paraview was used for viewing the voxelized volumes.

	Rotterdam		
	Hoogvliet-Zuid	Overschie	Heijplaat
Number of buildings	10 828	3 318	1 207
Too few points	-	-	-
Consecutive points same	1 562	1 222	572
Self intersection	14	2	1
N.p.p. [1] distance plane	306	535	58
N.p.p. [1] normals deviation	315	181	57
Too few polygons	373	77	16
Shell not closed	18 040	7 530	2 020
Non manifold edge	34	11	13
Multiple connected components	3	11	1
Polygon wrong orientation	2	-	3
Percentage of defects	95	92	90

Table 7: Overview of the detected defects in the Rotterdam datasets

5.2 TEST DATA

To test the repair capability, buildings from the existing CityGML datasets of Rotterdam and Montreal have been used. However, the defects which are present in these datasets are quite different. An overview of the validity of both datasets is shown in Table 7 and Table 8, based on the val3dity tool (see § 2.1.5). This shows a great difference between the two datasets.

	Montreal		
	VM01	VM02	VM03
Number of buildings	384	209	339
Too few points	13	5	31
Consecutive points same	324	122	373
Self intersection	-	-	-
N.p.p. [1] distance plane	-	-	-
N.p.p. [1] normals deviation	-	-	-
Too few polygons	-	-	-
Shell not closed	15	2	1
Non manifold edge	-	-	-
Multiple connected components	-	-	2
Polygon wrong orientation	-	-	-
Percentage of defects	16	10	20

Table 8: Overview of the detected defects in the Montreal datasets

Due to the specific defects in the datasets, the repair method would be evaluated for certain defects. If we would run the Rotterdam dataset, approximately 90% of the defects would consist of non-closed shells because of missing walls. Since this kind of validation would not be representative for other errors, it was decided to create a test set in which different kinds of errors are specifically selected. An overview of the test set is shown in Table 9.

1 Non planar polygon

#	Dataset	Building ID	Defect	Description
	Rotterdam			
1	HoogvlietZuid	7 467	-	Simple Block
2	HoogvlietZuid	2 399	-	Simple Tilted Roof
3	HoogvlietZuid	8 862	-	Combined Blocks
4	HoogvlietZuid	5 439	-	Combined Tilted Roof
5	HoogvlietZuid	6 736	-	Combined Blocks Tilted Roof
6	HoogvlietZuid	0 001	Too few polygons	Row house, missing 2 walls
7	HoogvlietZuid	6 517	Shell non closed	Orthogonal building with gap
8	HoogvlietZuid	8 592	Non planar polygon	Two vertices within tolerance
9	HoogvlietZuid	9 137	Non-manifold edge	Building with overshoot
10	HoogvlietZuid	3 606	Non-manifold edge	Gap + overshoot + double surface
11	HoogvlietZuid	1 107	Non planar polygon	Polygon soup
12	HoogvlietZuid	0 860	Shell non closed	Polygon soup
13	HoogvlietZuid	1 625	Self-intersection	Row house with many missing walls
14	HoogvlietZuid	0 095	Non planar polygon	Small building also missing a wall
15	HoogvlietZuid	0 556	Too few polygons	Box with gap and consecutive points same
	Montreal			
16	VM01	0 060	Consecutive points same	Building with detailed roof
17	VM01	0 028	Consecutive points same	Building with objects on roof
18	VM01	0 116	Consecutive points same	Building with objects on roof
19	VM01	0 050	Consecutive points same	Building with objects on roof
20	VM01	0 107	Shell not closed	Large building with interior

Table 9: Overview of the tested 3D models

5.3 VALIDATION AFTER REPAIR

The success of the repair process is measured by validating the resulting model and checking the Root Mean Square Error (RMSE) of the geometric comparison as described in § 3.5.3. The Metro-tool [Garland and Heckbert, 1997] which is available at `http://vcg.isti.cnr.it/vcglib/metro.html` has a tool in Meshlab, which is used. The results of the validation process are shown in Table 10.

	Building	Approach 1		Approach 2	
#	valid	valid	RMSE	valid	RMSE
1	yes	yes	0.014 m	yes	0.005
2	yes	yes	0.078 m	no	-
3	yes	yes	0.100 m	yes	0.006
4	yes	yes	0.078 m	no	-
5	yes	yes	0.109 m	yes	0.056
6	no	yes	0.088 m	yes	0.044
7	no	yes	0.143 m	no	-
8	no	yes	0.284 m	no	-
9	no	yes	0.421 m	no	-
10	no	yes	0.102 m	no	-
11	no	no	-	no	-
12	no	yes	0.247 m	no	-
13	no	no	-	no	-
14	no	yes	0.114 m	yes	0.057
15	no	yes	0.122 m	yes	0.032
16	no	yes	0.145 m	no	-
17	no	yes	0.214 m	no	-
18	no	yes	0.391 m	no	-
19	no	yes	0.335 m	no	-
20	no	yes	0.316 m	no	-

Table 10: Overview of repair process results

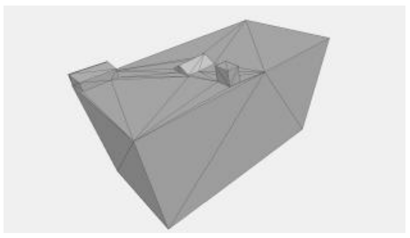
Approach 1 was able to return 18 valid buildings, thus repairing 12 out of 15 invalid buildings. Approach 2 clearly performed less desirable, returning only 6 valid buildings. By applying this approach 3 defective buildings were repaired and 2 valid buildings were turned invalid. An advantage however, is that the RMSE of the buildings processed by approach 2 are clearly lower, which indicates a surface reconstruction closer to the original model. An overview of the remaining defects per approach is shown in Table 11.

	Before repair	Approach 1	Approach 2
Defective buildings	15	2	14
Too few points	-	-	-
Consecutive points same	14	-	5
Self intersection	-	-	6
N.p.p. [1] distance plane	2	-	-
N.p.p. [1] normals deviation	1	-	-
Too few polygons	2	-	-
Shell not closed	4	2	-
Non manifold edge	2	-	1
Multiple connected components	-	-	-
Polygon wrong orientation	-	-	-
Percentage of defects	75	10	70

Table 11: Remaining defects after repairing the test set

5.3.1 Validation of approach 1

Overall, approach 1 was capable of returning a high percentage of valid buildings. An example is building 18 from the Montreal dataset. This model contained a Consecutive points same error, but could be repaired. An illustration of the model before and after repair is shown in Figure 99.


(a) Original building

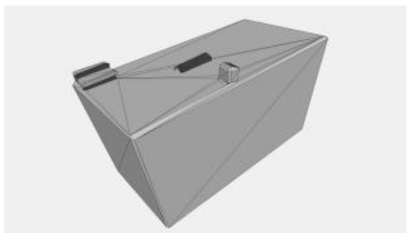
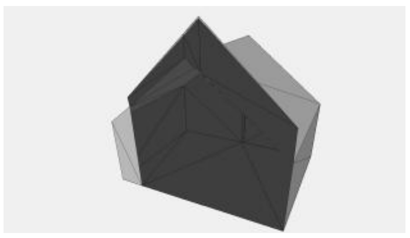

(b) Building after processing

Figure 99: Building 18

Another example of a repaired model is building 9, shown in Figure 100. The input model shows an overshoot and a gap, which are removed in the output model. It is clearly visible in this case that the Marching Cubes result is only capable of approximating the oblique surface.


(a) Original building

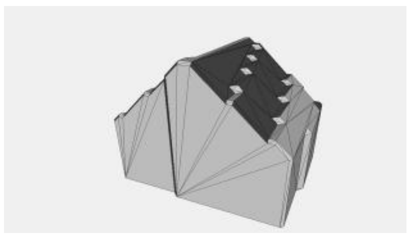

(b) Building after processing

Figure 100: Building 10

Two buildings (11 & 12) could not be repaired. Building 11 is shown in Figure 101a and contains multiple holes, self-intersections and an overshoot. Building 12 is shown in Figure 101b and also contains several holes and overshoots. By lowering the voxelization threshold to 3 the building could be repaired (see Section 4.2 for the description of the importance of the threshold).

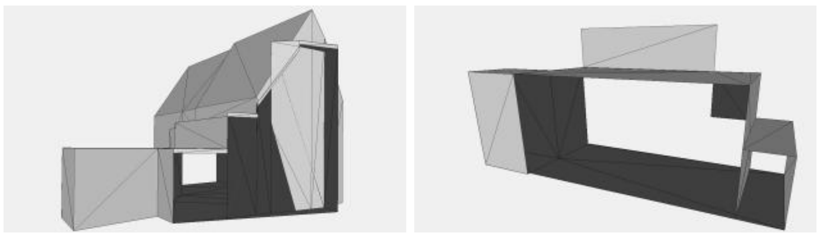

(a) Building 11 (b) Building 12

Figure 101: Buildings 11 & 12 were not repaired

Illustrations of the full test set before and after processing with approach 1 are presented in Appendix B.

5.3.2 Validation of approach 2

As was shown in Table 11, approach 2 was only able to return 6 valid building models. An example of one of these repaired building models is shown in Figure 102. Where the original model was missing a wall, the resulting model returns a closed volume. It is visible that the roof is accurately reconstructed. However, where the roof meets the missing wall surface, it is visible that pointy artefacts are created, which has been described in Section 4.4.

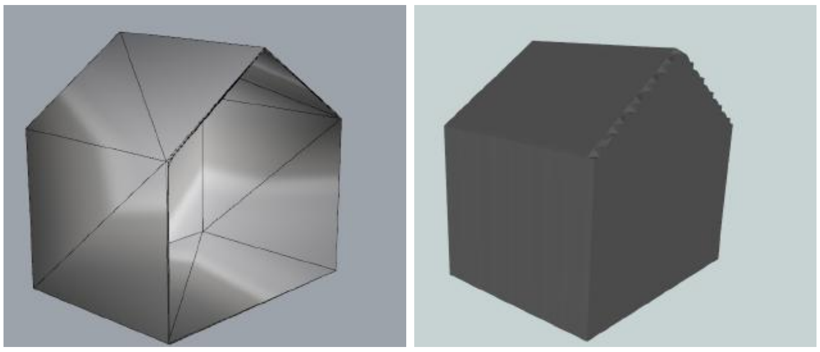

(a) Original building 14 (b) Building 14 after processing

Figure 102: Approach 2 applied on Building 14

An example of an invalid building which could not be repaired by approach 2 is shown in Figure 103. The original building contained a consecutive points same error. The processed building model is shown in Figure 103b. Although the resulting model seems to be visually correct, it contains self-intersections. This issue has been described in Section 4.4.

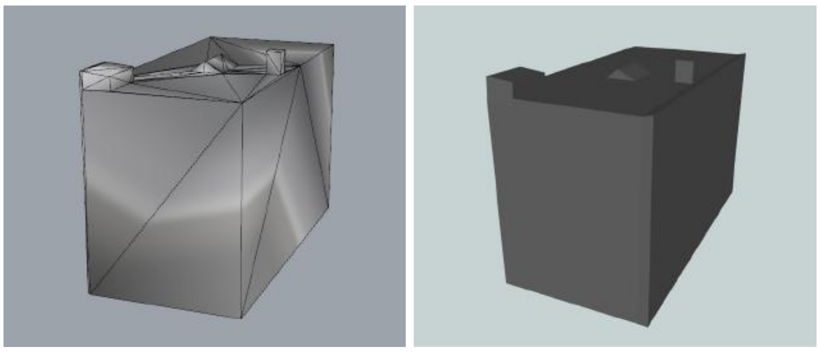

(a) Original building 18 (b) Building 18 after repair

Figure 103: Approach 2 applied on building 18

When we check the triangle mesh and zoom in on the roof of the model, the problem becomes visible. Figure 104 shows two locations of self-intersections, where the roof geometry has concave corners. Although the mesh is closed, edges are intersecting each other, turning the resulting model invalid. However, it is also visible that the oblique surfaces and sharp features are accurately reconstructed. The adjustments which are needed to prevent this approach from resulting in self-intersections are described in the future work section (see Section 6.3).

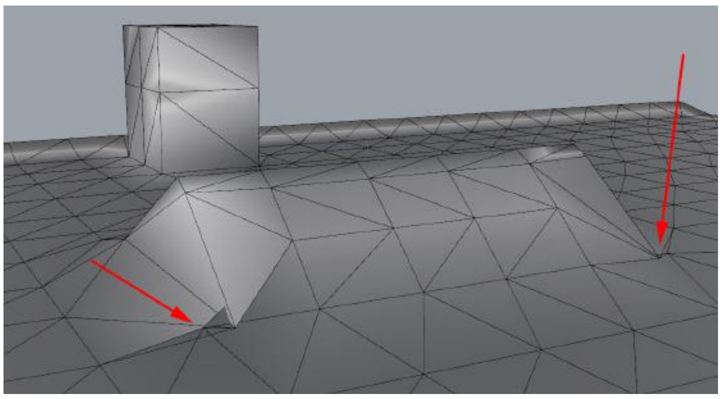


Figure 104: Self-intersections on the roof of building 18

5.3.3 Geometry comparison

To evaluate the quality of the repair, the geometry comparison such as described in § 3.5.3 is applied. In Table 10 it is visible that the RMSE of approach 1 on buildings with oblique surfaces is quite high, which is due to the stair-stepping effect. An almost orthogonal building (Building 18) is selected here to make a fair comparison between the two approaches.

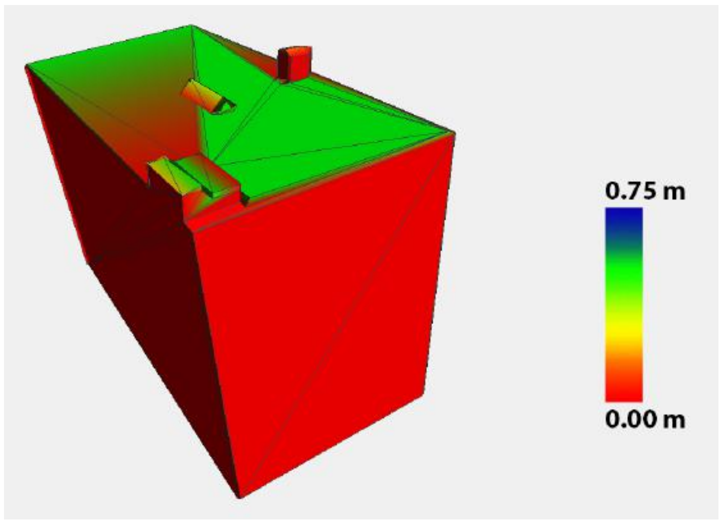


Figure 105: Geometric error in the repaired model

The geometry comparison between input and output model resulting from approach 1 is shown in Figure 105. The orthogonal building parts have been aligned, resulting in almost no error. The oblique roof surface and more detailed objects contain errors ranging up to 0.36 meter. Although these error values are mostly a matter of resolution, it is an indication that a significant geometric shift has taken place.

	Approach 1	Approach 1	Approach 2
resolution	32^3	64^3	32^3
RMSE	0.195 m	0.176 m	0.142 m

Table 12: Comparison of 3 repair results for Building 18

When comparing the RMSE-values of the geometry comparison for the different repair approaches, some differences become clear. Table 12 shows the results after applying approach 1 with a resolution of 32^3 and 64^3 and approach 2 with a resolution of 32^3. Although approach 2 does not repair the model, it does perform a more accurate surface reconstruction, even in case of orthogonal buildings.

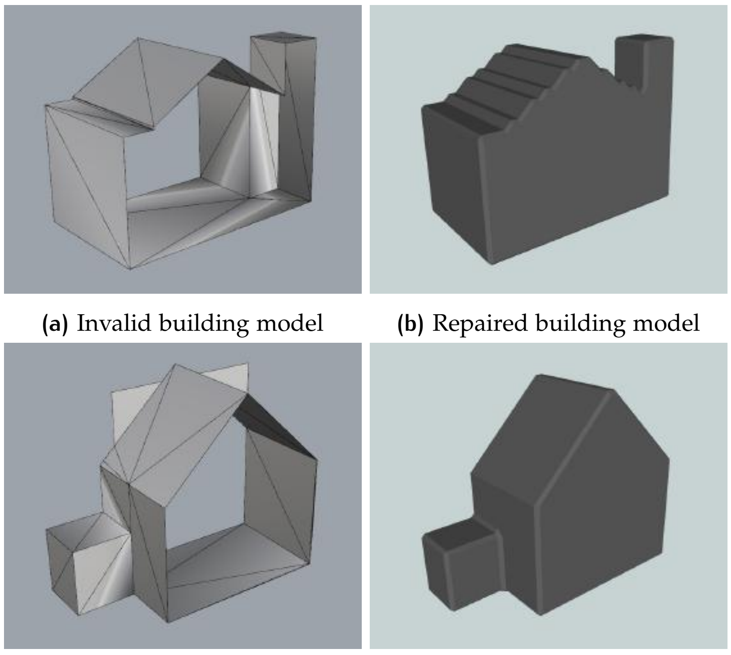
5.3.4 Repairing a full dataset

Since the first approach is shown to be capable of repairing a large number of defective building models, it has been tested on the Rotterdam3D dataset for the neighborhood Heijplaat. The results are shown in Table 13. The repair process was able to reduce the number of defective buildings from 1091 (90%) in the original dataset to 49 (4%) in the final result.

	Rotterdam3D Heijplaat	
	Before repair	After repair
Number of defective buildings	1 091	49
Percentage of defective buildings	90	4
Too few points	-	-
Consecutive points same	572	-
Self intersection	1	111
N.p.p. [1] distance plane	58	3
N.p.p. [1] normals deviation	57	6
Too few polygons	16	0
Shell not closed	2 020	76
Non manifold edge	13	15
Multiple connected components	1	4
Polygon wrong orientation	3	15

Table 13: Overview of the detected defects in the Rotterdam datasets

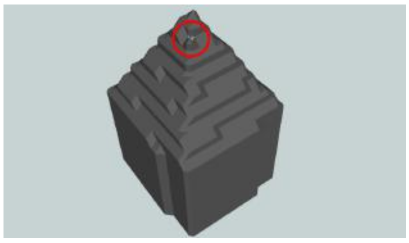
Two examples of buildings that have been successfully repaired are shown in Figure 106. The first example shows how two holes could be repaired. In the second example a combination of two holes and an overshoot is repaired.

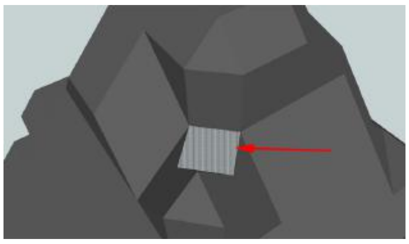

(a) Invalid building model (b) Repaired building model

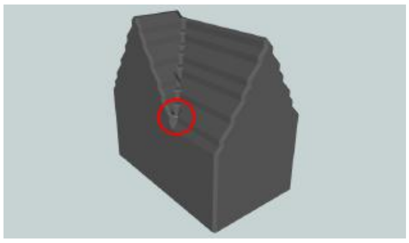
(c) Invalid building model (d) Repaired building models

Figure 106: Correctly repaired building model

The buildings which could not be repaired are mostly produced by the ambiguous cases of the Marching Cubes algorithm, sometimes in combination with a inaccurate voxelization. Two examples of defective output models are shown in Figure 107, showing the result of ambiguous Marching Cubes cases on the intersection of two oblique surfaces.


(a) Overview of input building model


(b) Gap due to ambiguity


(c) Overview of input building model

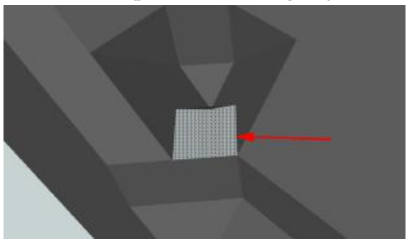
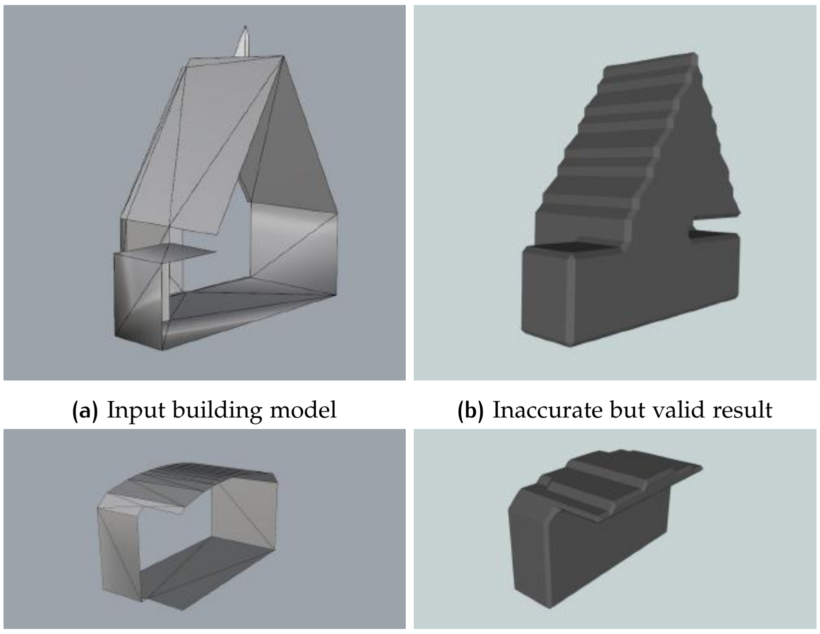

(d) Gap due to ambiguity

Figure 107: Defect output due to ambiguities in the Marching Cubes cases

Although the repair percentage is quite high, some clear remarks have to be made. First of all buildings in the resulting dataset (repaired or unrepaired) have been affected by a geometric shift, rounded off corners and contain the stair-stepping effect at the location of oblique surfaces. Secondly a number of the repaired buildings, do not represent the actual volume of the original building model. This may occur since the largest connected component is selected. For a inaccurate voxelization process, the largest connected component may produce valid output but does not necessarily represent the original building model.



(a) Input building model **(b)** Inaccurate but valid result

(c) Input building mode **(d)** Inaccurate but valid result

Figure 108: Examples of valid output for incorrect volume

5.4 EVALUATION

The capability of repairing certain defects has been shown. However, this is at the cost of geometric precision. Table 14 shows a comparison between the voxel-based method tested above and the Shrink-wrapping method as described in § 2.2.4 in Chapter 2.

	Voxel-based repair	Shrink Wrapping
Oblique surfaces	no	yes
Attribute preservation	possible	yes
Shift in geometry	yes	no
Affected by floating point arithmetic	no	yes
Repair of overshoots	yes	possibly incorrect

Table 14: Comparison in theory of a voxel based repair and Shrink Wrapping

In general the voxel-based repair method is robust in its repair capability. However, a major drawback is the shift in geometry and lack of support for oblique surfaces. The further studying of applying Marching Cubes on a distance field (see § 3.4.2) or an non-self-intersecting adaption of Dual Contouring (as described in Section 4.4) may be capable of improving these results. However, when taking into account the processing speed and comprehensibility of the algorithm, the Shrink-wrapping method can be considered as a more efficient and reliable repair method.

6 CONCLUSIONS, DISCUSSION AND FUTURE WORK

6.1 CONCLUSIONS

What are the most common errors of invalid CityGML models?
Which of these errors can be repaired by using a voxel-based repair method?

The most common error is very different per dataset. In Rotterdam3D the *Shell Not Closed* defect is very common because of a misinterpretation of the data standard. Another common error is *Consecutive Points Same*, furthermore some defects on the polygon level are present. In a dataset with better quality such as Montreal the most common errors is *Consecutive Points Same* which makes for more than 90% of the defects. For some complex buildings (almost rounded, very large or with many holes) the *Shell Not Closed* defect is sometimes present.

Which voxelization algorithm is most suitable for an automatic repair method?

In general a scan conversion in the 6 orthogonal directions using Parity Count method with a majority voting is sufficient to generate a manifold volumetric representation. It is dependent on the defects in the input polygonal model, but in general a parity count in 6 directions using a majority voting with a threshold of ≥ 4 seems most effective. Only in specific cases with multiple gaps and self-intersections more scan directions may be needed. In very rare cases of multiple double surfaces, ray stabbing is most suitable.

Which surface reconstruction method is most suitable for rebuilding sharp geometry characteristics?
Is a Marching Cubes algorithm sufficient or are other algorithms needed?
Is additional processing such as surface smoothing or edge sharpening required?

Surface reconstruction methods can be distinguished in two groups. The first group uses only a binary grid to determine the triangle mesh, the second group uses additional information such as intersections and surface normals of the original model.

- Approach 1: Marching Cubes is reliable, but not accurate in approximating oblique surfaces or sharp features. Applying this approach on a signed distance field instead of a binary grid will reconstruct oblique surfaces but still suffer from rounded off corners.
- Approach 2: Dual Contouring has the potential of performing an accurate surface reconstruction, reproducing any sharp features. However, also artefacts and self-intersections are created in the process.
- The Pressing method may be suitable as an alternative way of reconstructing oblique surfaces from a voxel representation.
- More research is needed to determine ether any of these methods are capable of reconstructing sharp features in a reliable way.

- For the Marching Cubes and Pressing method, an edge sharpening may be applied, although it has been shown to cause non-manifold exceptions.

How can the semantics be kept during voxelization process?

This thesis presents three ways of preserving/restoring the attributes during the model repair process. The selection of the appropriate method is very much dependent on the surface reconstruction technique which is applied. For Approach 1 (using Marching Cubes) geometry based attribute assumptions are recommended. For Approach 2 (using Dual Contouring) it seems most effective to store the attributes per grid edge.

What are the advantages and disadvantages of using a voxel-based repair methods when compared to existing repair methods?

Advantages:

- A manifold volumetric representation is almost guaranteed
- Non visible errors (such as *Consecutive Points Same* are repaired without the possibility of unexpected cases
- Non-manifold edges can be repaired in a consistent way

Disadvantages:

- By converting to a binary grid, there is a shift in geometry. Both of the researched approaches do end up with a significant geometric error.
- In both approaches artefacts may occur, showing unwanted geometry which is in some way still representing the binary grid.
- In the scan conversion process, the initial surface attributes are lost. To restore these attributes, customization steps are needed.
- Depending on the resolution, the processing speed of a voxel-based approach will be inherently slower than that of the existing methods.

To which extent is it possible to automatically repair a 3D City Model building using a voxel-based method?

- A voxel based method as applied in Approach 1 can consistently repair invalid 3D City models in the sense of making the 3D model free of defects. However, this is at the cost of significant geometry shifts while oblique surfaces are only approximated.
- The result of the surface reconstruction component in both approaches is not ideal. In general it is likely for artefacts to occur, showing unwanted geometry based on the binary grid.

- An adaptive contouring method such as applied in Approach 2 is more precise in the surface reconstruction, however several issues have been encountered which make this method unreliable for 3D City Model repair.
- In the scan conversion process, the initial surface attributes are lost. To restore these attributes, customization steps are needed.
- Some important challenges are still to be solved before a reliable and precise voxel-based repair method can be developed.

6.2 DISCUSSION

The aim of this thesis was to develop a voxel-based repair method for 3D city models and research its applicability. Some of the findings in this thesis raised ideas about the process of automatically repairing invalid 3D City models. The most important points are discussed below.

- We have seen that a voxel-based repair method has the potential for performing a volumetric repair for severely defect input models. However, some obvious drawbacks are inherent to this method. An initial loss of attributes, a possible geometric shift, possible artefacts and a relatively low processing speed are clear weaknesses coming to light when comparing this method to existing repair methods. In the light of 3D City model repair in general, a combination of repair techniques may be most effective in the repair of datasets. Some defects such as small gaps or wrong polygon orientation may be very efficiently processed by existing methods whereas overshoots or models resembling a polygonal soup may require a voxel-based method.

- The specific requirements for a repair tool is another point of discussion regarding the repair process. The most obvious goal would be to improve the data quality of a dataset in general, not allowing for a decrease in geometric quality nor the loss of attributes. However, another goal could be to repair the entire dataset automatically, regardless of loosing some geometric accuracy. This could be useful when a datasets needs to be processed, but the analysis-tool only accepts valid data. A related question is whether user input is considered as acceptable. Requiring user input for individual buildings may be very time consuming. However, it could be justified when it is needed to repair specific defects in an otherwise high quality dataset.

- On a more general note, the process of procedural creation of 3D City models should foremost be considered as a crucial part for the creation of valid datasets. It was also stated by previous contributions in the field that every dataset seems to have specific types of defects, based on the method by which they were produced. The percentage of invalid models in a dataset such as Rotterdam3D is over 90%, which is simply related to a misinterpretation of the data standard. A suggestion may be to focus repair methods at a certain dataset, where defects are produced by unexpected input.

6.3 FUTURE WORK

This thesis has concluded that a voxel-based repair method has the potential to repair a large amount of defects in 3D City models. However, the problem of an accurate surface reconstruction has not been fully resolved. Some areas of research which may improve the surface reconstruction are:

- The combination of applying the Marching Cubes algorithm on a signed distance field followed by an edge sharpening algorithm has the potential to be a reliable method for accurate reconstruction of the original model. Points of focus during this work may be (i) the handling of defect geometry in computing the signed distance field, (ii) avoiding any exceptions in the edge sharpening algorithm which may turn the mesh non-manifold, (iii) determining the most appropriate way of attribute preservation and (iv) speed optimization of this process.
- The application of Dual Contouring for repairing 3D City models needs to be researched in more depth to be able to make conclusions about its suitability. The implementation of the original Dual Contouring algorithm often results in self-intersections, an extended implementation for manifold results is required for 3D City model repair. Additionally, unwanted artefacts are created when a precise dual placement cannot be computed due to any 'misplaced voxels'. Also in case of gaps, a transition zone will be present between the original model and binary grid.
- Alternatively, an isosurface extraction method which also uses information of the original model may be studied: Dual Marching Cubes (and its manifold extension). Ether these methods are more suitable for the surface reconstruction of defect input models could be a topic of future research.
- If the use of data of the original model is considered unacceptable, the Pressing approach offers a different approach which does support oblique surfaces based on a volumetric grid only. The theory of *Pressing* has been studied but the effects of its implementation on 3D city model repair have not been tested. Although the implementation of Pressing is quite intricate, combining it with the scan-conversion implemented in this paper may give reliable results.

The scan conversion process and morphological operators have shown that a manifold volumetric representation can be achieved for almost all defects present in 3D City models. Some areas that could be further explored are:

- During the scan conversion, information on the number of votes of the Majority Voting process becomes available. In theory this information can be used to detect and locate errors in a 3D City model. How to use this information to improve further model repair could be a topic of further research.
- Also the possibilities of speed optimization could be researched. A more efficient surface reconstruction can often be achieved by using an CO-tree, however the issues of preserving the attributes should be taken into account.

A | MARCHING CUBES CASES

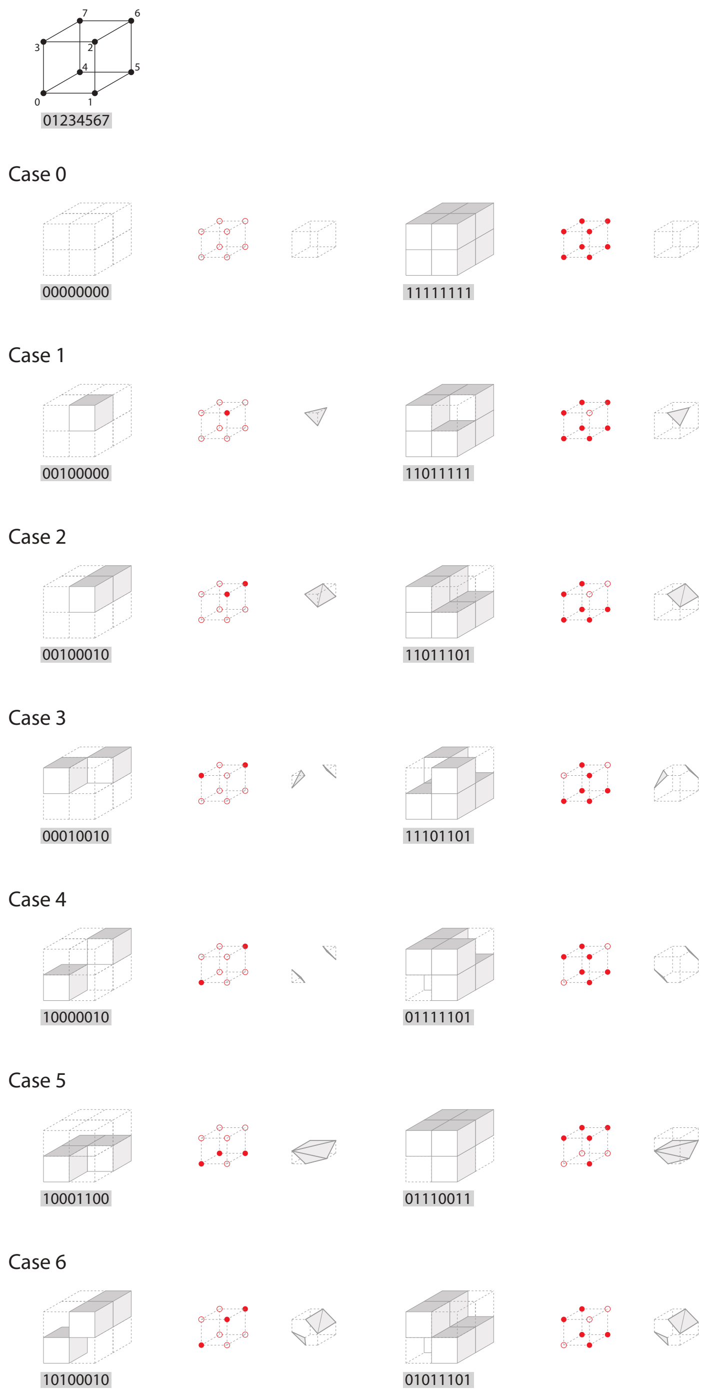


Figure 109: Marching Cubes cases 0 - 6

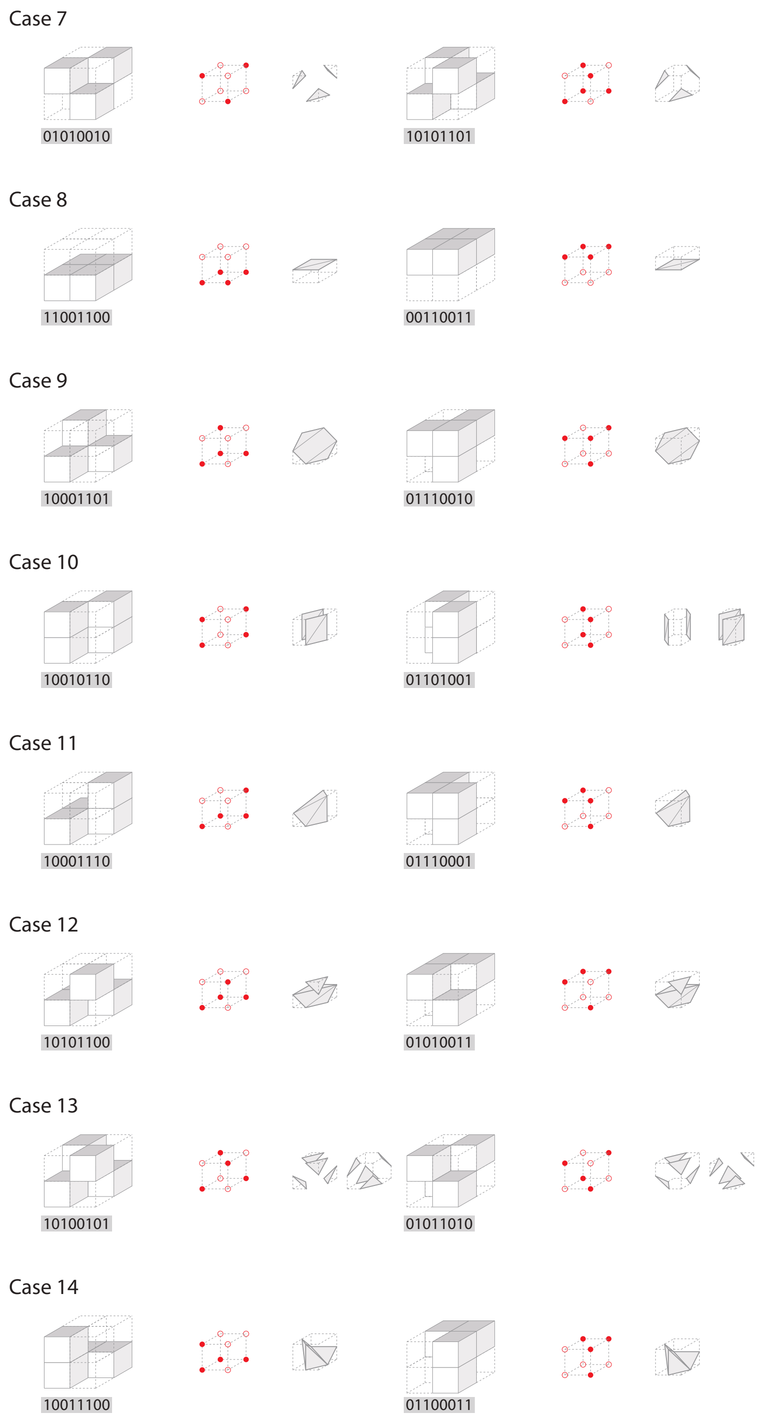


Figure 110: Marching Cubes cases 7 - 15

B TEST CASE VALIDATION

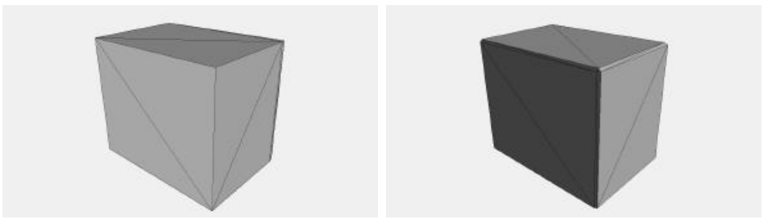


Figure 111: Building 1: valid box

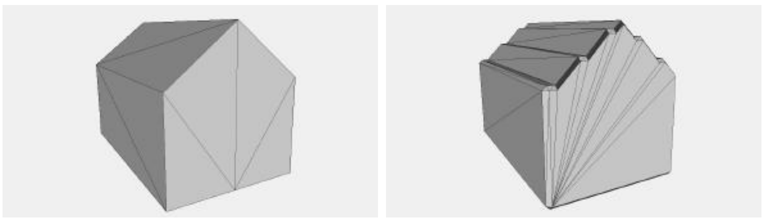


Figure 112: Building 2

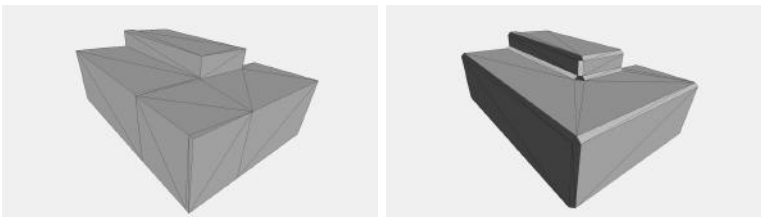


Figure 113: Building 3

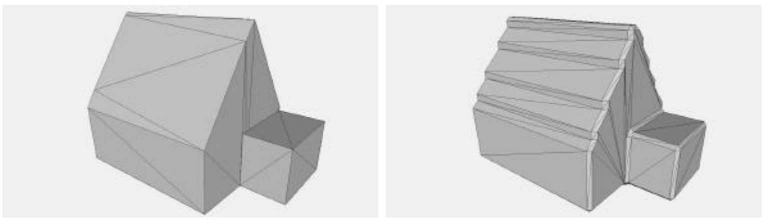


Figure 114: Building 4

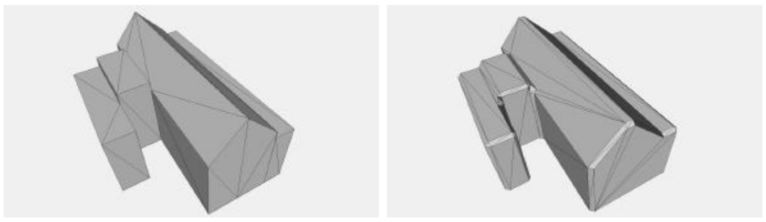


Figure 115: Building 5

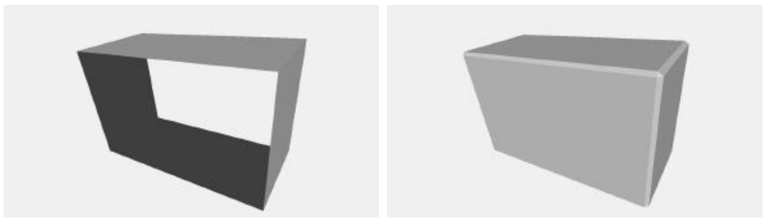


Figure 116: Building 6

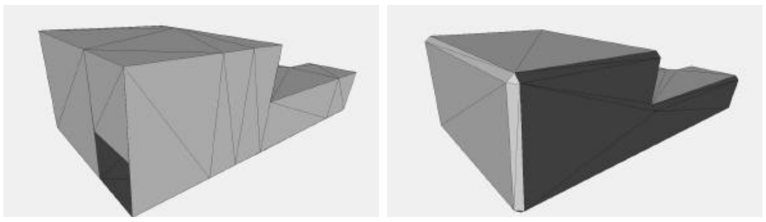


Figure 117: Building 7

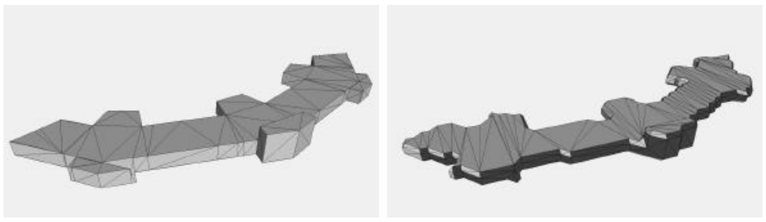


Figure 118: Building 8

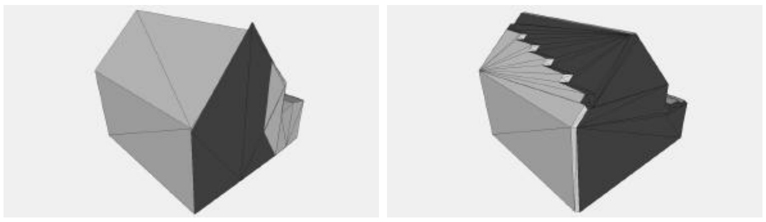


Figure 119: Building 9

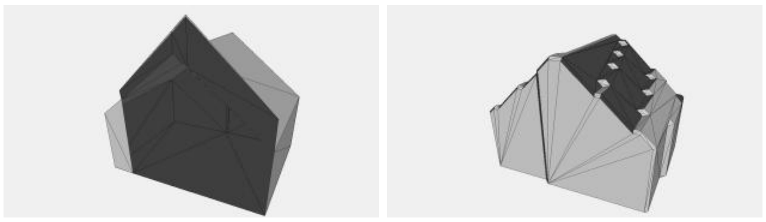


Figure 120: Building 10

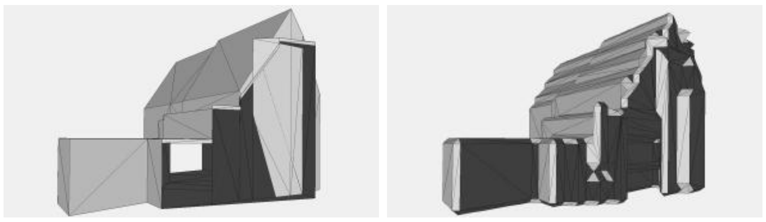


Figure 121: Building 11

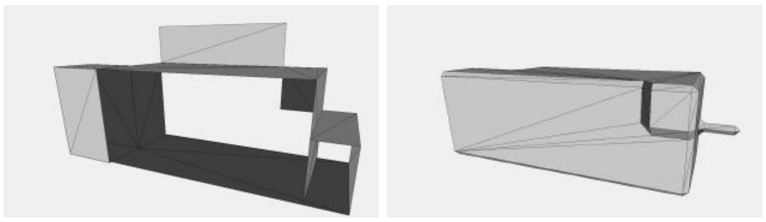


Figure 122: Building 12

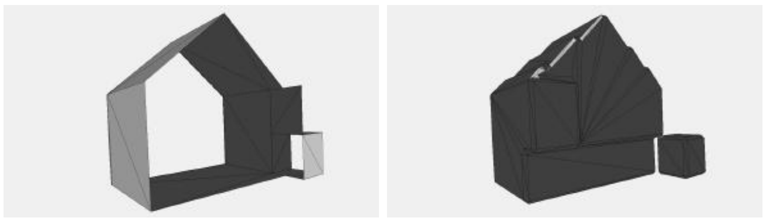


Figure 123: Building 13

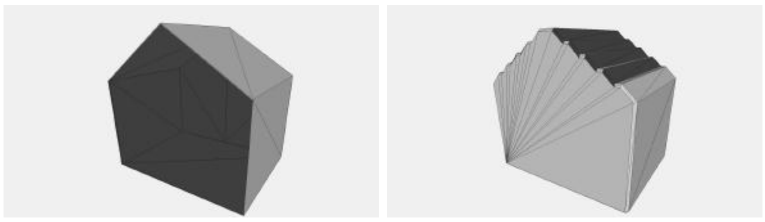


Figure 124: Building 14

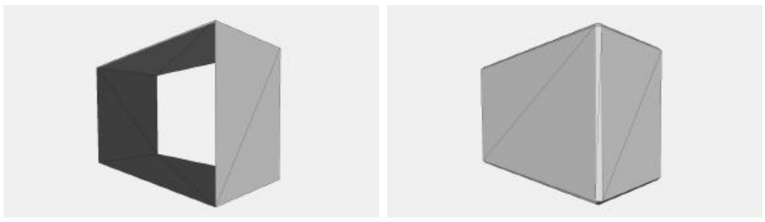


Figure 125: Building 15

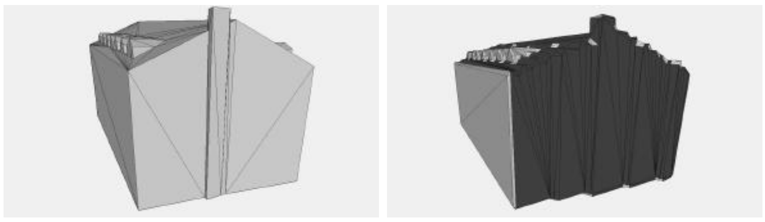


Figure 126: Building 16

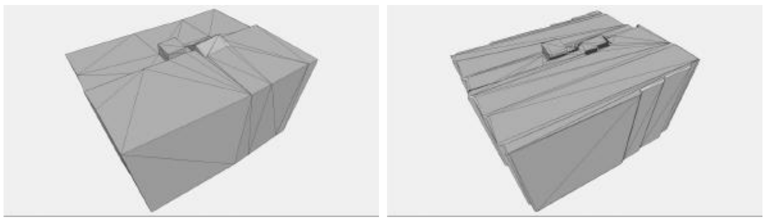


Figure 127: Building 17

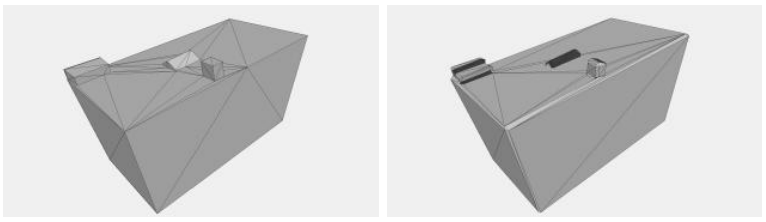


Figure 128: Building 18

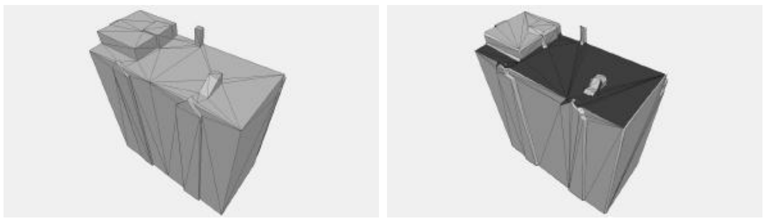


Figure 129: Building 19

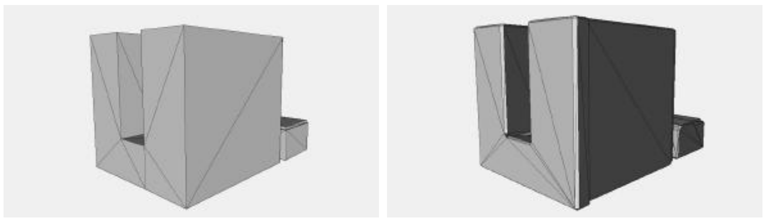


Figure 130: Building 20

C REFLECTION

This thesis proposes two voxel-based approaches for automatically repairing defect 3D city building models. The research was conducted from November 2014 to June 2015. The initial planning separated timeslots for literature research, studying of the existing algorithms and implementation of the final prototype. In reality, the process shifted more towards a continuous research, which resulted in the implementation of two different approaches to the problem. Early in the process, three components were determined as being at the core of a voxel-based repair method; (i) Voxelization, (ii) Morphological Operators and (iii) Surface Reconstruction. At the start of the process a majority of the time was spent on the first two components. The final component Surface Reconstruction proved the be the most challenging, therefore taking up most of the time during the latter stages of the project.

The final product of this thesis is the code that was developed in order to apply the two repair methods on existing datasets. The code that was created for several algorithms that were tested but did not end up in either approach, can also be considered as part of the final product. In order to test the effectiveness of both repair methods, tests were performed on defect buildings in existing datasets. In addition, a validation process was needed to define and test the validity of 3D city models. Both were available from the start of the process, along with useful tools for pre-processing of 3D city models.

The methodical line of approach in the Master of Geomatics consists of Data Capture, Data Storage, Analysis, Communication/Visualization and Quality Control. This thesis has a clear position within this methodical line, as its focus lies on the aspect of Quality control of 3D City models. To carry out this research, knowledge from the core courses of Geomatics was needed. In particular the courses 3D Modelling of the Built Environment, Python Programming and Geo Datasets and Quality provided information that could be directly applied.

3D City models have the potential to indirectly influence citizens through applications such as urban planning, disaster management, virtual reality and navigation. However, many defects are currently present in these datasets which can prevent further processing and analysis of the data. By developing an automatic repair method, the data quality of these datasets can be improved, enabling the potential uses of 3D city models. This thesis contributes to the larger society by examining the possibilities of automatically improving the data quality of these datasets using a voxel-based method. The resulting algorithm can be applied to repair a large number of defects, taking into account a slight shift in geometry. Applying this method can be useful for enabling the processing of datasets for applications such as volume calculation, noise mapping or wind analysis.

BIBLIOGRAPHY

N. Alam, D. Wagner, M. Wewetzer, J. von Falkenhausen, V. Coors, and M. Pries. Towards automatic validation and healing of CityGML models for geometric and semantic consistency. *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences*, 1(1):1–6, 2013.

C. Andújar, P. Brunet, and D. Ayala. Topology-reducing surface simplification using a discrete solid representation. *ACM Transactions on Graphics (TOG)*, 21(2):88–105, 2002.

C. Andújar, P. Brunet, A. Chica, I. Navazo, J. Rossignac, and A. Vinacua. Computing maximal tiles and application to impostor-based simplification. In *Computer Graphics Forum*, volume 23, pages 401–410. Wiley Online Library, 2004.

K. Arroyo Ohori. Validation and automatic repair of planar partitions using a constrained triangulation. Master's thesis, Delft University of Technology, aug 2010. ISBN: 978-94-6186-034-7.

N. Aspert, D. Santa Cruz, and T. Ebrahimi. Mesh: measuring errors between surfaces using the hausdorff distance. In *ICME (1)*, pages 705–708, 2002.

M. Attene, B. Falcidieno, J. R. Rossignac, and M. Spagnuolo. Edge-sharpener: recovering sharp features in triangulations of non-adaptively re-meshed surfaces. 2003.

M. Attene, M. Campen, and L. Kobbelt. Polygon mesh repairing: An application perspective. *ACM Computing Surveys (CSUR)*, 45(2):15, 2013.

F. Biljecki, G. B. M. Heuvelink, H. Ledoux, and J. Stoter. Propagation of positional error in 3D GIS to the estimation of the solar irradiation of building roofs. *International Journal of Geographical Information Science*, In submission, 2015.

S. Bischoff and L. Kobbelt. Structure preserving CAD model repair. In *Computer Graphics Forum*, volume 24, pages 527–536. Wiley Online Library, 2005.

R. Boeters. Automatic enhancement of CityGML LoD2 models with interiors and its usability for net internal area determination. Master's thesis, Delft University of Technology, jun 2013.

S. Burtsev and Y. P. Kuzmin. An efficient flood-filling algorithm. *Computers & graphics*, 17(5):549–561, 1993.

C. Cappelle, M. E. El Najjar, F. Charpillet, and D. Pomorski. Virtual 3D city model for navigation in urban areas. *Journal of Intelligent & Robotic Systems*, 66(3):377–399, 2012.

R. Chen. The development of 3D city model and its applications in urban planning. In *Geoinformatics, 2011 19th International Conference on*, pages 1–5. IEEE, 2011.

E. V. Chernyaev. Marching cubes 33: Construction of topologically correct isosurfaces. *Institute for High Energy Physics, Moscow, Russia, Report CN/95-17*, 42, 1995.

A. Chica, J. Williams, C. Andujar, P. Brunet, I. Navazo, J. Rossignac, and A. Vinacua. Pressing: Smooth isosurfaces with flats from binary grids. In *Computer Graphics Forum*, volume 27, pages 36–46. Wiley Online Library, 2008.

D. Cohen-Or and A. Kaufman. Fundamentals of surface voxelization. *Graphical models and image processing*, 57(6):453–461, 1995.

P.-E. Danielsson. Euclidean distance mapping. *Computer Graphics and image processing*, 14(3):227–248, 1980.

H. de Kluijver and J. Stoter. Noise mapping and GIS: optimising quality and efficiency of noise effect studies. *Computers, Environment and Urban Systems*, 27(1):85–102, 2003.

A. A. Diakité, G. Damiand, and G. Gesquière. Automatic semantic labelling of 3D buildings based on geometric and topological information. In *Proc. of 9th International 3DGeoInfo Conference (3DGeoInfo)*, 3DGeoInfo conference proceedings series, pages 49–63, Dubai, United Arab Emirates, November 2014. Karlsruhe Institute of Technology. URL http://nbn-resolving.org/urn:nbn:de:swb:90-438043.

M. Durupt and F. Taillandier. Automatic building reconstruction from a digital elevation model and cadastral data: an operational approach. *International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*, 36(3):142–147, 2006.

W. R. Franklin and E. Landis. Connected components on 1000x1000x1000 datasets. In *16th Fall Workshop in Computational Geometry, Smith College, Northampton, MA*, volume 1011, 2006.

M. Garland and P. S. Heckbert. Surface simplification using quadric error metrics. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 209–216. ACM Press/Addison-Wesley Publishing Co., 1997.

M. Garland and P. S. Heckbert. Simplifying surfaces with color and texture using quadric error metrics. In *Proceedings of the conference on Visualization'98*, pages 263–269. IEEE Computer Society Press, 1998.

F. Hétroy, S. Rey, C. Andújar, P. Brunet, and À. Vinacua. Mesh repair with user-friendly topology control. *Computer-Aided Design*, 43(1):101–113, 2011.

M. Isenburg and J. Shewchuk. Streaming connected component computation for trillion voxel images. In *Workshop on Massive Data Algorithmics*, 2009.

C. B. Jones. *Geographical information systems and computer cartography*. Routledge, 2014.

T. Ju and T. Udeshi. Intersection-free contouring on an octree grid. In *Proceedings of the 14th Pacific Conference on Computer Graphics and Applications*, volume 3, 2006.

T. Ju, F. Losasso, S. Schaefer, and J. Warren. Dual contouring of hermite data. In *ACM Transactions on Graphics (TOG)*, volume 21, pages 339–346. ACM, 2002.

A. Kaufman and E. Shimony. 3d scan-conversion algorithms for voxel-based graphics. In *Proceedings of the 1986 workshop on Interactive 3D graphics*, pages 45–75. ACM, 1987.

L. P. Kobbelt, M. Botsch, U. Schwanecke, and H.-P. Seidel. Feature sensitive surface extraction from volume data. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 57–66. ACM, 2001.

T. Kolbe, G. Gröger, and L. Plümer. CityGML–3D city models and their potential for emergency response. *Geospatial information technology for emergency response*, 257, 2008.

T. H. Kolbe. Representing and exchanging 3D city models with CityGML. In *3D geo-information sciences*, pages 15–31. Springer, 2009.

T. H. Kolbe, G. Gröger, and L. Plümer. CityGML: Interoperable access to 3D city models. In *Geo-information for disaster management*, pages 883–899. Springer, 2005.

F. Lafarge and C. Mallet. Creating large-scale city models from 3D-point clouds: a robust approach with hybrid representation. *International journal of computer vision*, 99(1):69–85, 2012.

H. Ledoux. On the validation of solids represented with the international standards for geographic information. *Computer-Aided Civil and Infrastructure Engineering*, 28(9):693–706, 2013.

H. Ledoux. Three-dimensional primitives in the context of the CityGML QIE. Unpublished work, 2015.

R. E. Loke and F. W. Jansen. Maintaining sharp features in surface construction for volumetric objects. 2007.

W. E. Lorensen and H. E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. In *ACM Siggraph Computer Graphics*, volume 21, pages 163–169. ACM, 1987.

J. Manson and S. Schaefer. Isosurfaces over simplicial partitions of multiresolution grids. In *Computer Graphics Forum*, volume 29, pages 377–385. Wiley Online Library, 2010.

C. Montani, R. Scateni, and R. Scopigno. Discretized marching cubes. In *Proceedings of the conference on Visualization'94*, pages 281–287. IEEE Computer Society Press, 1994.

J. Moser, F. Albrecht, and B. Kosar. Beyond visualisation–3D GIS analyses for virtual city models. *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 38(4):W15, 2010.

F. S. Nooruddin and G. Turk. Simplification and repair of polygonal models using volumetric techniques. *Visualization and Computer Graphics, IEEE Transactions on*, 9(2):191–205, 2003.

S. Oomes, P. Snoeren, and T. Dijkstra. 3D shape representation: Transforming polygons into voxels. In *Scale-Space Theory in Computer Vision*, pages 349–352. Springer, 1997.

S. Schaefer and J. Warren. Dual marching cubes: Primal contouring of dual grids. In *Computer Graphics Forum*, volume 24, pages 195–201. Wiley Online Library, 2005.

S. Schaefer, J. Warren, and R. UniversityE. Dual Contouring: The secret sauce. 2002.

S. Schaefer, T. Ju, and J. Warren. Manifold dual contouring. *Visualization and Computer Graphics, IEEE Transactions on*, 13(3):610–619, 2007.

S. P. Singh, K. Jain, and V. R. Mandla. Virtual 3D City modeling: Techniques and Applications. *ISPRS-International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 1(2):73–91, 2013.

A. Stadler and T. H. Kolbe. Spatio-semantic coherence in the integration of 3D city models. In *Proceedings of the 5th International Symposium on Spatial Data Quality, Enschede*, 2007.

H. Steuer, T. Machl, M. Sindram, L. Liebel, and T. H. Kolbe. Voluminator - Approximating the volume of 3D buildings to overcome topological errors. In *Accepted for 18th AGILE Conference on Geographic Information Science*, Jun 2015.

J. Stoter, L. van den Brink, G. Vosselman, J. Goos, S. Zlatanova, E. Verbree, R. Klooster, L. van Berlo, G. Vestjens, M. Reuvers, et al. A generic approach for 3D SDI in the Netherlands. In *Proceedings of the Joint ISPRS Workshop on 3D City Modelling&Applications and the 6th 3D GeoInfo Conference Wuhan, China*, pages 26–28, 2011.

D. Wagner, M. Wewetzer, J. Bogdahn, N. Alam, M. Pries, and V. Coors. Geometric-semantical consistency validation of CityGML models. In *Progress and New Trends in 3D Geoinformation Sciences*, pages 171–192. Springer, 2013.

J. Wilhelms and A. Van Gelder. *Topological considerations in isosurface generation extended abstract*, volume 24. ACM, 1990.

S. Wold, K. Esbensen, and P. Geladi. Principal component analysis. *Chemometrics and intelligent laboratory systems*, 2(1):37–52, 1987.

A. Zamyadi, J. Pouliot, and Y. Bédard. A three step procedure to enrich augmented reality games with CityGML 3D semantic modeling. In *Progress and New Trends in 3D Geoinformation Sciences*, pages 261–275. Springer, 2013.

J. Zhao, H. Ledoux, and J. Stoter. Automatic repair of CityGML LOD2 buildings using shrink-wrapping. *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences, II-2 W*, 1:309–317, 2013.

J. Zhao, J. Stoter, and H. Ledoux. A framework for the automatic geometric repair of CityGML models. In *Cartography from Pole to Pole*, pages 187–202. Springer, 2014.

COLOPHON

This document was typeset using LaTeX. The document layout was generated using the arsclassica package by Lorenzo Pantieri, which is an adaption of the original classicthesis package from André Miede.