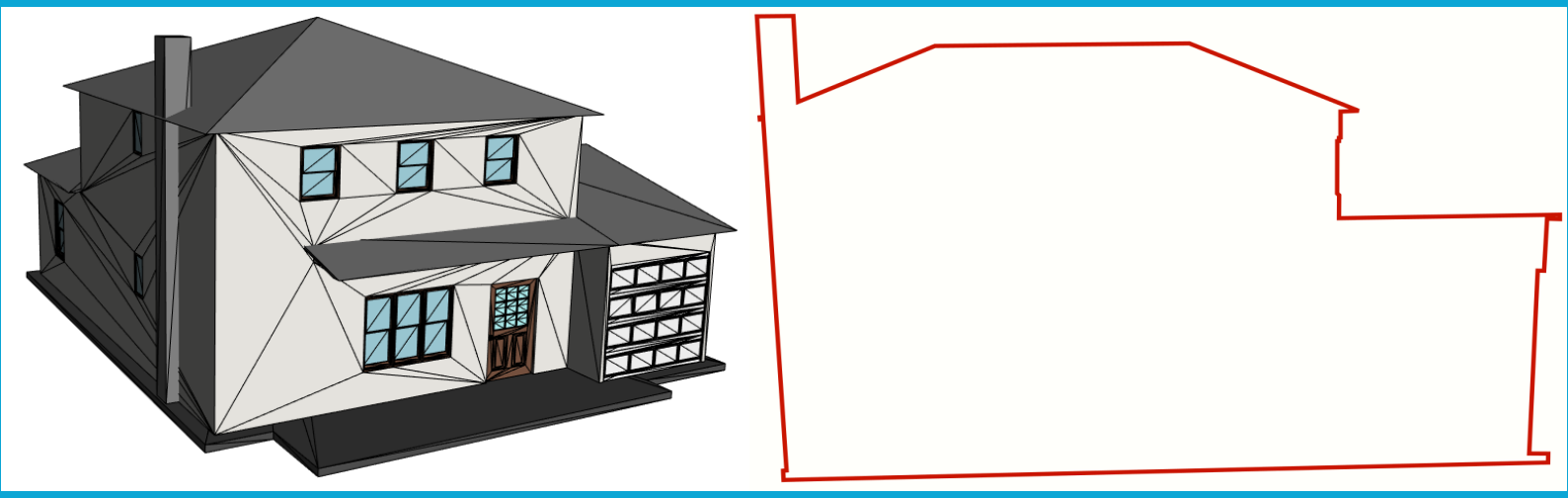
MSc thesis in Geomatics for the Built Environment

Robust Interior-Exterior Classification For 3D Models

Nikolaos Tzounakos
2019



TU Delft

ROBUST INTERIOR-EXTERIOR CLASSIFICATION FOR 3D MODELS

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

Master of Science in Geomatics for the Built Environment

by

Nikolaos Tzounakos

June 2019

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:	Dr. Liangliang Nan
	Dr. Hugo Ledoux
Co-reader:	Dr. Yufu Zang

ABSTRACT

The use of 3D models has been rapidly expanding, finding applications in both scientific and commercial fields. One common requirement for these various applications is the geometrical and topological validity of these models. However, many models available online contain deficiencies in various forms, such as duplicated geometry, gaps in the surface, etc.. To cope with those deficiencies, a standard solution is the clean extraction of the model's boundary, and simultaneously the model's reconstruction in a way that its structure is valid. This thesis tackles a more generalized problem, the inside-outside classification for these models. Where many approaches might have requirements for running analysis, the methodology presented strives to robustly handle all cases.

These last decades, there have been various approaches in solving the "inside - outside classification problem". A major attempt utilizes the winding number algorithm, in order to assign values to elements whose position is relevant to the input model. By assessing that value, a decision on whether the element in question is interior or exterior is taken. Other approaches work with casting rays, or other geometric analysis to also identify the borders of a model and segment the interior from the exterior. Also, since deficiencies inhibit the kick-starting of the necessary analysis, there are methods that try to restructure said models in order to clear any existing deficiencies.

The methodology within this thesis will attempt a different approach from those that have been presented until now, which is transferring the problem from three into two dimensions. The first step is introducing a planar cross section on the area of interest. From there, through some graph reconstruction, geometric and optimization applications, a valid 1-manifold boundary of the cross-section is created. On that, the application of inside-outside classification through ray casting is possible.

Assessing the results of the pipeline proves that the automated process can produce valid results, for a particular point of interest, related to an input model. The pipeline has been proven to function regardless of the cutting plane's orientation, and can handle robustly a multitude of geometrically and topologically defective models. The results from this thesis can inspire further applications, and improvements on the pipeline can further evolve the quality of its outcome.

ACKNOWLEDGEMENTS

This last year, a lot of people have supported me directly or indirectly on this thesis project, and I would like to express my appreciation towards them in this small section.

Initially, I would like to give thanks to both my supervisors for this project, Liangliang Nan, and Hugo Ledoux. Special thanks to Liangliang, who was completely supportive to my venture, always available, patient, and providing me with technical and theoretical knowledge, that were indispensable for this thesis. His constant presence was a big support through the whole year. Also thanks to Hugo, for being honest, apprehensive and also critical, while giving precious insight on crucial moments. I would also like to express my gratitude to the co-reader, Yufu Zang, who provided constructive feedback through the last stages of the thesis.

Furthermore, I appreciate the staff of the Geomatics program, for supplying me with the necessary knowledge to go through these two years, and possibly through my future as well. My special gratitude also to my co-students in Geomatics, who were also making a positive, supportive and competitive-free environment, allowing friendships and kindness to bloom.

Finally, I would also like to express my deepest gratitude to my family and my friends. Without them, I would not be able to complete these two years, and their presence has always been a big mental support for me.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

List of Algorithms

ACRONYMS

2D Two Dimensional . . . 15
3D Three Dimensional . . . xi
CAD Computer-Aided Design . . . xi
CDT Constrained Delaunay Triangulation . . . 15
CGAL Computational Geometry Algorithms Library . . . 36
CityGML City Geography Markup Language . . . 2
DT Delaunay Triangulation . . . 24
GUI Graphical User Interface . . . 36
IFC Industry Foundation Classes . . . xi
LOD Level of Detail . . . 2
OFF Object File Format . . . 35
PLY Polygon File Format . . . 35

1 INTRODUCTION

Since the beginning of the 21st century, digital 3D models have been proven useful in many applications. This rapidly expanding field has found applications in entertainment, design, commerce, and many other technical and technological fields. Such 3D models are created through surveying methods, digitization processes, or plain digital design. There is a plethora of applications for such models, ranging from real estate to learning/tutoring methods [Huk, 2006], scientific research and even the production of objects through 3D printing. For all of these purposes, a pre-requisite that is often taken as granted is the geometrical validity of the digital 3D model. In this case, valid structure means that a model is watertight, has no self-intersections, and free of other types of deficiencies, which will be explained later.

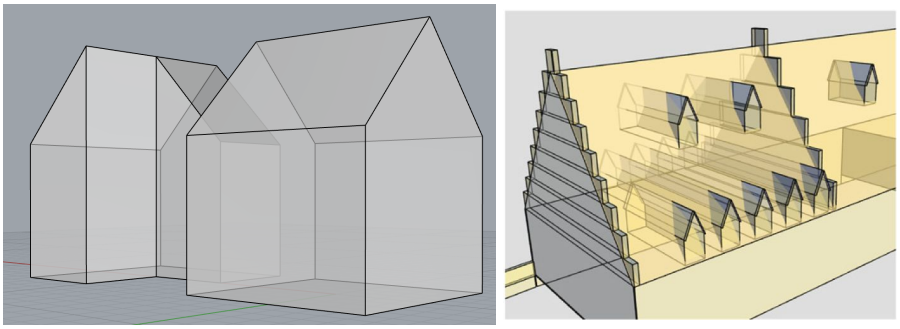
As has been stated above, there are many fields that benefit from a topologically and geometrically sound model. This stands true for both the scientific, and the commercial environment.

One main example is that of urban reconstruction. Musialski et al. [2013] note that a valid urban model would be useful for many applications, such as their use in the entertainment industry, the digital mapping of cities, or even for urban planning and simulations. There are also cases where a building needs to be assigned semantic information (eg. roof or wall, construction material, etc.), for such cases as those of urban planning, environmental reports, or even emergency and catastrophe planning [Benner et al., 2005]. For a more discrete representation, the optimal would be having a clean model.

Another case is that geometrical and topological deficiencies in a model often inhibit its correct analysis. For this case, Sindram et al.'s voluminator project exists, but still only to handle one element of analysis in 3D models, that of the model's volume. Even seemingly unrelated scientific fields can benefit from valid 3D models. In the case of Rusinkiewicz et al. [2002], there is mention of cases where a valid model representation of the ground's geological structure can help not only in the field of geology, but even in others, such as archaeology. Furthermore, in today's society, there are even cases where building representations can be used in order to calculate the annual tax that a residence's owner has to pay [Boeters et al., 2015]. This can be performed by calculating the area owned by a person, but of course there is a requirement that such a model used for analysis is clean.

1.1 PROBLEM STATEMENT

In order to be able of effectively utilizing a digital 3D surface mesh model within an application, the particular model has to be valid. Most model processing algorithms (some of which will be presented in Chapter 2) have strict requirements on the input data structure. For example, self-intersections are not a realistic representation of geological structures [Caumon et al., 2009]. Indeed, having clean, discreet representations "... is important in various applications, ranging from engineering to scientific research..." [Guo et al., 2019]. An example of a clean model against a model with deficiencies can be observed in Figure 1.1.



(a) Clean model [Sondermann, 2018] **(b)** Flawed model [Nagel et al., 2009]

Figure 1.1: A clean model versus a model with deficiencies. (a) Topologically and geometrically sound basic building representation. (b) Elements of the model (windows and roof) are crossing each other, which is not realistic representation of the building.

However, nowadays many models that are used or that circulate the World Wide Web may contain deficiencies. Below are a few examples of such deficiencies, along with a visual representation in Figure 1.2:

- Holes on a surface's model.
- Faces or other elements of the 3D model that are intersecting.
- Faces, edges or vertices that are duplicate; existing multiple times.
- Inconsistent orientation; meaning neighboring surfaces whose normal vectors point in opposite directions.
- Redundant/Undesired interior structure.

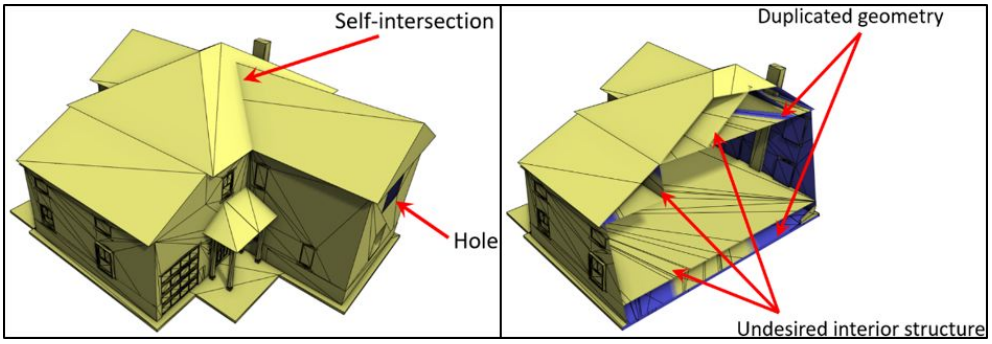


Figure 1.2: Digital 3D model deficiencies [Nan, 2018]

Campen et al. [2012] mention that it is often observed of 3D model meshes to have a number of defects and flaws that make them "incompatible with the quality requirements of specific applications". Thus, the importance of repair or removal of such flaws is usually imperative to the uninhibited flow of 3D processes that use these models.

More specifically, there are a plethora of applications where any interior information that a 3D model might possess is considered a redundancy. An example of such case would be the models of buildings used in a City Geography Markup Language (CityGML) model, especially those with a Level of Detail (LOD) from 1 to 3, where the geometric detail of the model's exterior is required [Biljecki et al., 2016].

In many applications, there is a requirement about being able to extract the exterior boundaries of a model. A more generalized approach would be the interior- exterior classification of such model. In this case, a digital 3D mesh model would be used

as input. Then in the process, the elements of the model have to be classified as "interior" or "exterior". The output would be the outer shell of the model, ideally a 2-manifold devoid of geometrical and structural deficiencies, as they have been mentioned above. This is the actual problem that the current research will tackle.

1.2 RESEARCH QUESTION

As will be presented in Chapter 2, precedents that try to partially or completely solve interior/ exterior classification exist. This research will try to give a different approach to this method, and cover some of the flaws that have been observed in others. It aspires to be a valid addition to the existing methods, and find a robust solution through a stable methodology that can produce results of good quality from beginning to end.

The main research question of this thesis is:

How can a point be robustly classified as lying in the interior or the exterior of a complex polygonal mesh model?

As such, the main objective of this thesis is to develop a novel methodology that will produce valid results even for models that contain deficiencies (such as holes). In order to be able to meet this thesis' main demand, there are some sub-questions, whose answers pose an important role in this research:

- What is the methodology that should be followed in order to produce valid results?
- What should be the structure (geometry, stored format, etc.) of the data to be handled, as well the structural form of the final output?
- What advantages and disadvantages does this new method have? Is it a process that can produce valid results?
- How does this method compare to other existing approaches?

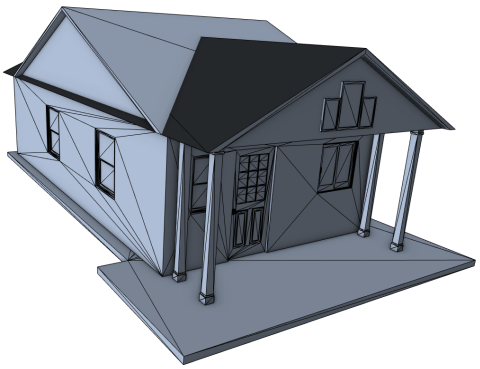
1.2.1 Scope of the Research

This thesis will focus on a methodology of step-by-step computational processes that will take as input digital 3D models that may even contain structural inconsistencies. The input data are 3D surface meshes, not solid meshes, mainly those of buildings; although the methodology can be applied on any type of model. An example input model can be seen in Figure 1.3. The general type of model is also the same as the one shown in Figure 1.2. The models used are comprised of triangular surfaces, and are used without any semantic —or other types of — information.

The proposed methodology will have a two-dimensional approach, where planar cross-sections of the model are created, and interior/exterior classification performed on them, with the final result being the recognition of the model's outer borders. Every step of that process is evaluated and validated, before being able to explore the next step. The tools which are known and available at this point will be utilized for the purpose of producing the desired final result.

The approach of the methodology is two-dimensional, and thus a three-dimensional approach will not be explored. Furthermore, the thesis will not focus on the reconstruction of the input model before the computation, but rather use the model as initially given, through the classification's execution.

Finally, the research's purpose is to give insight into the methodology that is applied to produce reliable interior-exterior classification. This means that the scientific



(a) Example input.

Figure 1.3: Example input model. Building representation surface mesh model.

approach itself is going to be evaluated, and any advantages/ disadvantages against other existing methods will be presented.

1.3 IMPORTANCE

A lot of research has been carried out in the field of digital 3D modeling and reconstruction, not just focused on the visualization aspect of model design, but also on the geometrical aspect of a model's validity, digital storage structure and geometric processing.

1.3.1 Scientific

What this thesis aspires to do is improve a step in the process of acquisition, analysis, and presentation of data: that of data analysis. In the handling of data, various fields take part, such as those of computer graphics, computer vision, photogrammetry, remote sensing, etc. These fields could potentially benefit from a solution to the research question's problem.

In Chapter 2, a plethora of methods relevant to this subject are going to be presented. Some try to solve interior-exterior classification, others attempt to identify the outer boundaries of a model, and the rest present some approaches to fixing or restructuring an input model in order to relieve it of its deficiencies. The degrees of success are varying, although many have presented results of high quality. However, one common observation is that in all these methods, there are requirements that an input model needs to fulfill, in order for them to be smoothly executed.

As such, this graduation thesis will try and attempt a solution in a relevant problem, as stated in the research question in Section 1.2. The main point of this thesis' approach is to be robust against any type of input model, and still produce satisfying, quality results. The methodology that tries to achieve this requirement will be presented in Chapter 3.

The result of this research is expected to be useful in and of itself for users that would like to be able to extract just the boundary elements of a 3D model. Until this thesis' writing, creating the exterior of a model is done mostly manually, but an automated process is desirable by the scientific community [Donkers et al., 2016].

1.3.2 Practical & Commercial

Also, this thesis' results would further enhance various applications that are benefited by having knowledge of a model's exterior elements. One such application would be the calculation of a building's Net Internal Area, which could be used for taxation purposes as stated by Boeters et al. [2015].

Depending on model's nature, just extracting the exterior would allow its further processing for the purpose of animation. Reasons behind this would be for entertainment, visualization, or even for simulation.

Some applications of this research's results would be (but not restricted to): visualization, surface editing, the ability to convert an IFC model into a valid CityGML model [Donkers et al., 2016], 3D printing, etc. Visual examples of such applications are shown in Figure 1.4.

Figure 1.4: Research result applications. Left: Model editing. Right: Visualization

1.4 THESIS OUTLINE

After the introduction, the present document has the following structure:

- Chapter 2 reviews work that has been elaborated by others, but with a subject relative or relevant to the one of this thesis.
- Chapter 3 is where the novel methodology is presented. Each step is explained extensively, starting with the input of a model and ending with the acquisition of the final result.
- Chapter 4 presents the options and elements used during the practical implementation of the algorithm. This section mentions the data that were handled, the tools that were used during the analysis, as well as all the options that were made during the implementation, along with their reason.
- Chapter 5 contains results from the methodology's practical application, as well as a discussion on those results.
- Chapter 6 is the final section of the document, where there is a small assessment of whether the research question has been achieved, what is the importance of the achieved results, together with a reflection on the thesis as a whole. The final part is recommendations on improving this methodology, and how its results could be used in various applications.

Finally, this document also contains an appendix, where:

- in Appendix A, some pseudo-code for the more important functions of the methodology is presented.

2 RELATED WORK

The endeavor of this research is not unique. There are other precedents of attempts where researchers have proposed various approaches to classify the interior and exterior of a digital 3D model. These have all been met with varying degrees of success, each method possessing specific advantages and disadvantages. As such, this section aims at having summarized presentations of some major attempts at interior/ exterior classification. The section's structure is that initially (Section 2.1 to Section 2.4), some noted attempts are shown. Then, Section 2.5 gives some cases where mesh reconstruction is attempted, in order to make mesh models have cleaner geometry and topology, with the goal of them being easier in analyzing.

2.1 AUTOMATED GEOMETRIC / THEMATIC PROCESS

A unique method has been presented by Donkers et al. [2016], trying to keep the exterior information of a model stored in IFC format, in order to create a valid LOD3 CityGML model. The input itself was often invalid.

In this process, the first step is to semantically map the elements (solids and surfaces) of the input model. The semantics of a surface are given based on its orientation (or normal vector). The next step is to perform morphological operations of dilation and erosion on these elements, as to group together the ones that are similarly defined semantically. This also helps in removing some of the initial geometric and topological imperfections of the model, and "patching" some of the surface's holes. Through this, the initial solid elements transform into surfaces (observed in Figure 2.1). Finally, the model is further refined both geometrically and semantically; the semantic refinement happens for CityGML purposes.

Figure 2.1: Extraction of the exterior surface of IFC solids. [Donkers et al., 2016]

This method produces valid results, but is only useful for the specific case of converting data from IFC to CityGML format. As such, it is not meant for more generalized applications. Furthermore, this automated process has been tested and does not fare well against certain input model deficiencies, such as holes on surfaces, or when large surfaces are missing.

There has been a similar approach by Deng et al. [2016], where a mapping framework between IFC and CityGML models was presented. Their implementation contained a reference ontology, a schema by which the various model elements where referenced

from one format to the other. The results were of good quality, but the method is restricted in that it can only be applied between these two specific file formats.

2.2 SIGNED DISTANCE FIELD

Another method that has been used for classification is that of the signed distance field [Xu and Barbič, 2014].

In this approach, the input is a polygon soup mesh, meaning a collection of triangles, the aggregation of which comprises a certain shape.Through this method a surface which is the offset of the input is created, at a user-defined distance σ (seen in Figure 2.2). To speed up computation, any elements of the model that lie within other elements (are within the latter's bounding box) are classified as being interior, and excluded from the further computations. The next step is to check points on the elements that are left. If their distance from the initial boundary's iso-surface is bigger than σ, then these points are also classified as the interior. Finally, the rest of the points that are classified as exterior give that property to their parent elements, which are then deemed the exterior boundary of the model.

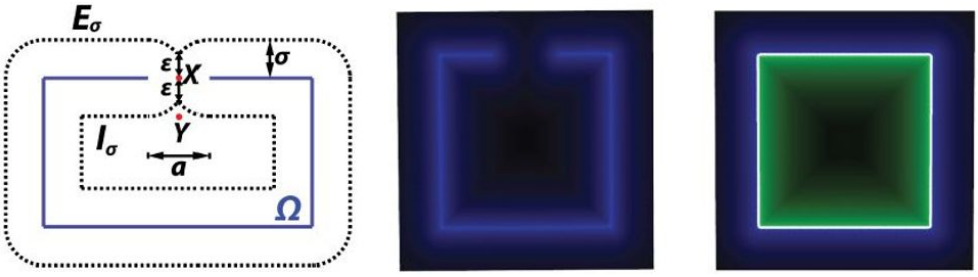


Figure 2.2: Left: Geometric representation of Signed Distance Field construction. Right: Different results of produced iso-surface dependent on σ [Xu and Barbič, 2014]

This method is used mainly for the creation of manifold iso-surfaces, where the input can even be non-manifold. The only user defined parameter is the offset distance σ, but a lot of careful tuning is needed, and the final output geometry can potentially be altered from the initial input.

2.3 WINDING NUMBER

Jacobson et al. [2013] presented an automatic algorithm about inside – outside segmentation of a digital 3D model. This method works by assigning an index number to an element of the model, and then categorizing that element based on how much "inside" or "outside" of the model it is, through the assigned number.

The index number assigned in this method is called the "Winding Number". To measure this number for a certain point in space, related to a surrounding curve, the integral of the angle that the curve creates when projected on a unit circle is calculated. If the elements of the curve are discreet (edges, vertices), then the calculation of the winding number is given by the formula in Figure 2.3.

This method is also generalized for three dimensions. In this case, instead of curves, surface elements are used. Also, the winding number is now calculated based on the solid angle created by the surrounding surface, rather than the 2-dimensional angle.

By setting a number as a threshold, users can define all indices above that number as "inside". However, there are some flaws to the algorithm presented in that research. One is that it demands from the input model to contain no deficiencies (watertight,

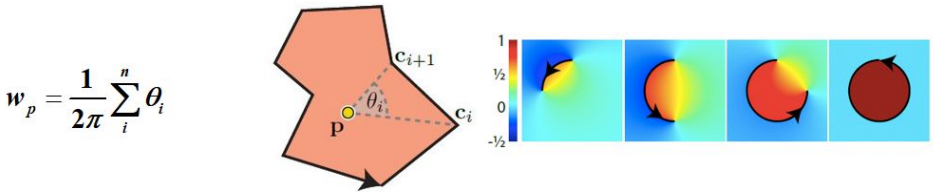


Figure 2.3: The idea of Winding Number. Left: Winding number formula for discreet curve elements. Center: Winding number geometric calculation depiction Right: Winding number visualization [Jacobson et al., 2013]

no self-intersections, etc.). In a more recent paper, Hu et al. [2018] present a method of preparing the input model and robustly making it valid. This process, however, demands a lot of computational power and time, and even then, the geometry of the initial model might be deformed. Furthermore, these processes demand the consistent orientation of the model's structure.

Another drawback on that method is that the orientation of the surrounding curve greatly affects the result. An approach was made by Takayama et al. [2014a] on this matter, by creating and utilizing a formula that tries to minimize the Dirichlet Energy of the Winding Number. However, at the end of the paper the authors state that their method did contain flaws. Specifically, the method is weak in cases where the curves are incorrectly oriented. An example of bad results due to incoherent / incorrect orientation is shown in Figure 2.4.

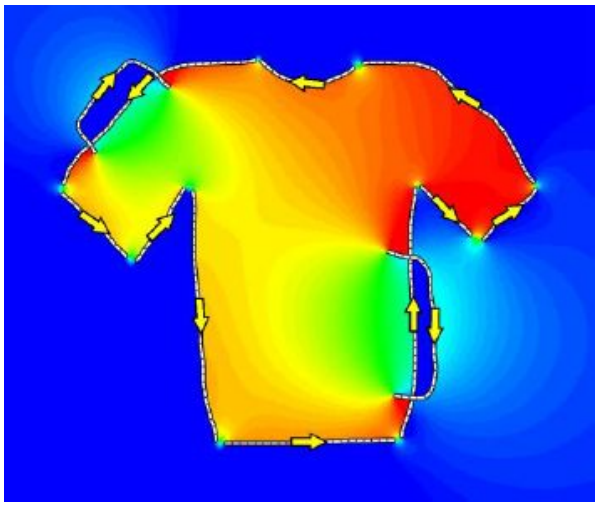


Figure 2.4: Classification errors due to curve orientation. Correct orientation in the image is defined as counter-clockwise, and the pocket areas colored blue have been classified as outside. [Takayama et al., 2014a]

2.4 RAY CASTING

There has been a different approach in classifying the interior and exterior part of a polygonal surface model, where the elements are categorized by an assigned number called "Depth buffer" [Nooruddin and Turk, 2000].

The process of the particular algorithm is to create for each polygon (surface element) of the model, random rays that fall on its surface. If the ray meets another surface before the intended one, the surface is deemed interior; otherwise, exterior. If even a small portion of the rays manage to hit the intended surface, then it is still classified as exterior. The sum of these assigned interior/exterior classifications constitutes the

depth buffer. In that case, if the depth buffer surpasses a user-defined threshold, the polygonal surface is classified as exterior, or interior in the other case.

A simple example of how the ray casting works is shown in Figure 2.5. In this case, the generated ray intersects with another surface before the intended target. As a result, the target surface in this iteration would be classified as inside. However, if rays from other origins / angles would hit the surface, then it would be classified as exterior.

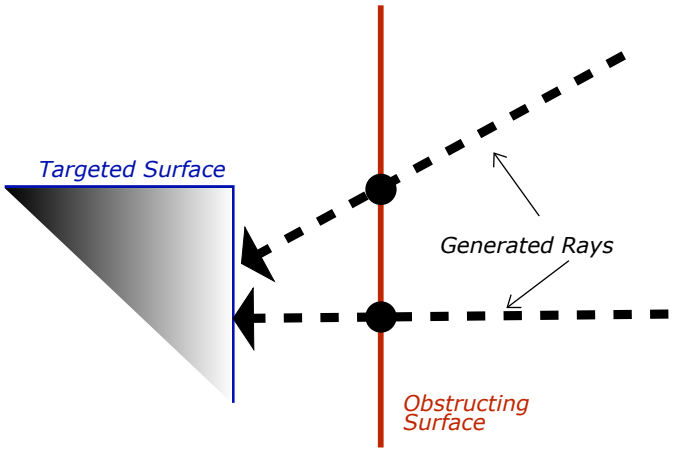


Figure 2.5: Ray casting classification process

This is a simple solution to the interior/exterior classification problem, with results of good quality, when the input model has a clear structure (as shown in Figure 2.6). The drawbacks of this method are that it cannot robustly classify by itself models that have holes on the surface, or on the model's interior. The existence of duplicate surfaces also inhibits the classification process. Furthermore, creating a multitude of rays from many different points of view is computationally expensive.

Figure 2.6: Ray casting method results [Nooruddin and Turk, 2000]

2.5 MESH RECONSTRUCTION

Many of the above methods require input models that follow certain criteria. For example, the [Jacobson et al., 2013] method depends heavily on the orientation of the curve surrounding the point, during winding number calculation. Generally, when a model is geometrically and topologically valid, it is easier to perform analysis on it. Any deficiencies on a model are treated as special cases, and may pose problems during classification processes, or even make the production of results impossible. In this section, a selection of methods that rely on restructuring a model and are closely related to the current problem is presented.

2.5.1 Winding Number Dirichlet Energy Minimization

As mentioned in Section 2.3, one approach to achieving the consistent orientation of the boundaries of a mesh is through utilizing the winding number approach, and minimizing the Dirichlet Energy of the winding number [Takayama et al., 2014a].

In this case, the input model is split into manifold surface patches comprised of surface elements. The different patches are split in areas where non-manifold edges exist. As such, input models with holes can be accepted in this method.

In the end result, those elements of a single patch are all going to be decided together whether they are going to be flipped (re-oriented) or not. Every surface is assigned a binary variable, the two values stating whether the patch is going to be flipped or not. Then, a formula is computed, calculating the Dirichlet Energy of the model through the Winding Number Equation. Finally, a binary optimization problem is introduced, with the objective being the Dirichlet energy's minimization. The solution of this optimization gives values that indicate whether each surface patch needs to be flipped or not. An example result of this method is shown in Figure 2.7

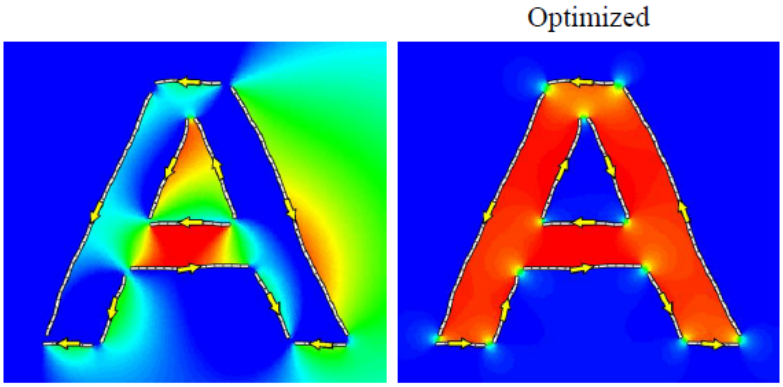


Figure 2.7: Re-orientation results with Dirichlet Energy minimization [Takayama et al., 2014a]

2.5.2 Ray Casting Re-Orientation

Another, performed by the same [Takayama et al., 2014b] utilizes ray casting in order to classify a surface; and flip its orientation, if necessary.

For this method, every face of the input model is approached separately. Rays in random directions are generated from randomly sampled points on the face (as seen in Figure 2.8). If a ray is generated in one direction, another is also generated aiming at the opposite direction, so that there is an equal number of generated rays from both sides of the face. When one ray does not intersect any other part of the model, it is decided that this ray is facing towards the front (or outside) of the model. Since the initial orientation of the face is known through its normal vector, if the general direction of the rays that are looking "outside" is opposite to the face's normal vector, that face is flipped.

Regarding interior faces, since no ray can see "outside", the distance between the ray's origin point and the first intersection with another face in its path is taken into account. This means that the ray which crosses the longer distance in order to intersect with a face is considered as the front one.

2.5.3 Mesh Generation from CAD Models

Guo et al. [2019] point that the creation of high-quality meshes often is an element that leads to the success of the meshes' numerical analysis.

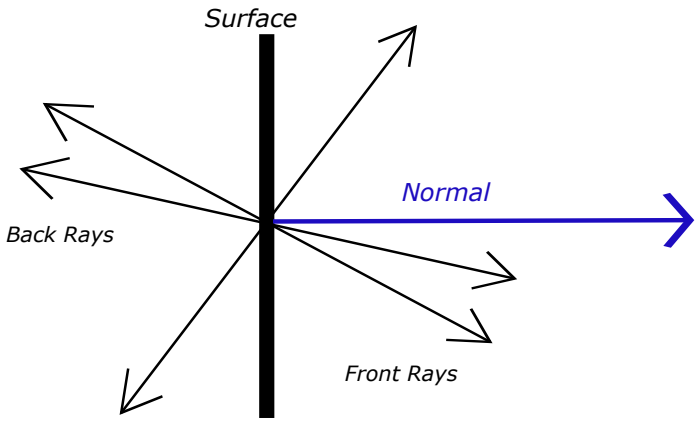


Figure 2.8: Rays generated at random from a surface

In their paper, they present a way to automatically create high-quality meshes from CAD models. In their case, the input is a 3D model comprised of parametric surface patches. Their first step is to work on each surface patch separately, and transform it from a parametric surface into a triangle mesh. At this step, their algorithm takes care to preserve the boundary geometry of these surface patches. When the boundaries are recovered, they implement two-dimensional triangulation to the patches.

Since the patches might not be topologically connected to each other, or there could be degeneracies or holes in between them, any empty space between the patches is also filled through triangulation. When this is done, patches that geometrically depict the same surface are clustered together. The last step is implementing a remeshing algorithm, which entails having less triangles in flat areas, and denser meshes in areas where finer detail should be preserved. A basic example result of this methodology is shown in Figure 2.9.

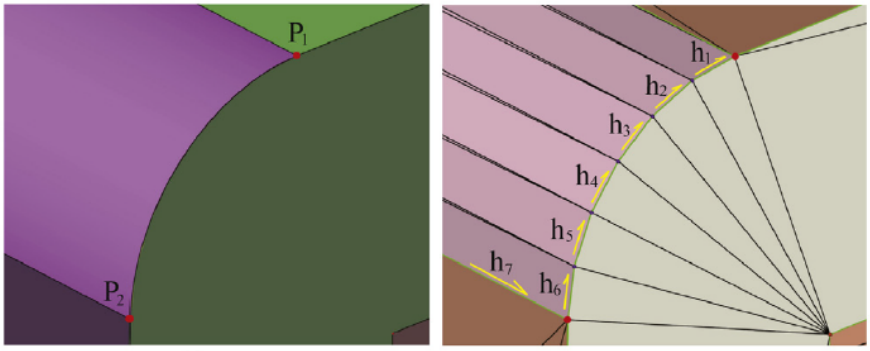


Figure 2.9: CAD to mesh reconstruction [Guo et al., 2019]

2.5.4 Volumetric Intersection Removal

One last method is a volumetric approach by Sacht et al. [2013]. This method uses a conformalized mean curvature flow pipeline, which degrades a topologically sphere-like object into a sphere unit, until a point where there are no more self-intersections. Usually, according to them, the point at which the model becomes intersection-free is long before the final stage of spherical geometry. A representation of the process is shown in Figure 2.10.

At that point, the product is an intrinsically similar object to the input, but with altered volume. Through a nonlinear optimization problem, they reverse the previous iterations to a point where the processed model retains the original volume, and is

as close as possible to the original shape as well. To recover its initial volume while retaining this new form, volumetric parametrization is applied on it. This method's restriction is that it requires a closed surface in order to function, and the authors have stated that the process may not always return the object to its original shape.

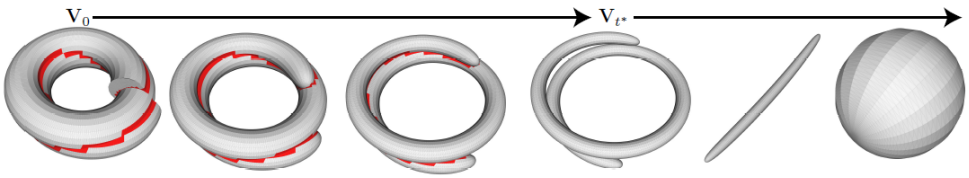


Figure 2.10: Evolution of item until self-intersection-free stage [Sacht et al., 2013]

3 METHODOLOGY

Chapter 2 presented some approaches on interior-exterior classification. In them, different processes appeared, aiming to solve certain problems. The main common element found when observing them was that they had limitations when special cases (eg. geometric deficiencies) where considered, or they would produce a result the shape of which was close to the initial model, but could not always preserve the original shape.

In this chapter, the methodology of this thesis will be presented. The main aspiration of this is that the methodology should be able to robustly produce a result of good quality. This means that it should arbitrarily handle all cases of degeneracy in a model, but also be able to retain the finer geometric details of the model. For this, it was decided to transfer the problem from three dimensions to two, since Two Dimensional (2D) analysis is more easily executed.

The first step is to get a surface model as input. Then, it is handled in such a way as to remove as much as possible of the more analysis-inhibiting deficiencies from the input; namely, duplicate geometry and self-intersections. Then, the surface model is intersected with a plane, to produce line-segment geometry, and transfer the problem in two dimensions. The next step is to group the line segments into connected components, a step necessary for simplifying later analysis processes. Following that is the task of closing any gaps/ holes present by implementing Constrained Delaunay Triangulation (CDT), in order to have a closed boundary. Next, values are assigned to the connected components based on a ray casting method. Finally, the outer borders of the planar cross section are extracted. Then, it is possible to classify whether a point in space lies inside or outside of the model.

A flowchart depicting the methodology's pipeline is shown in Figure 3.1.

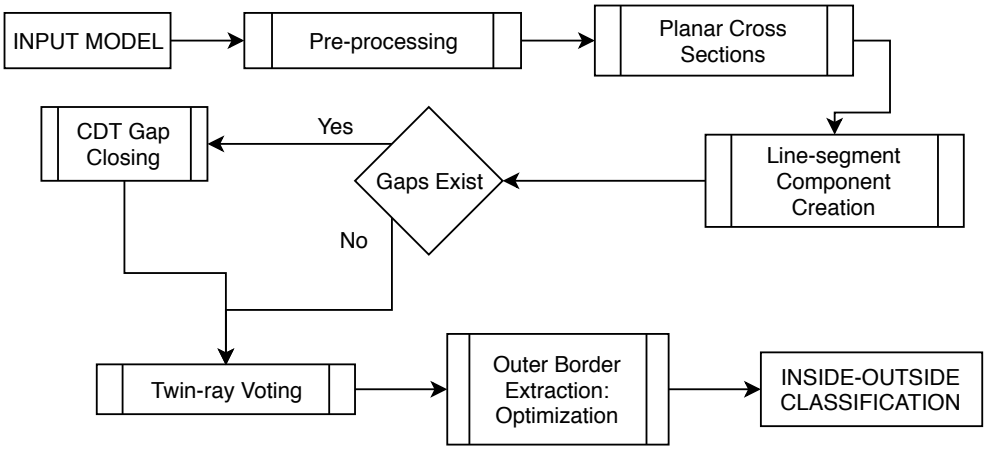


Figure 3.1: Flowchart of the methodology's pipeline

At the end of each section explaining a respective step of the pipeline, an example on a single building will be given, for better understanding. The example model in this case is shown in Figure 3.2. The model depicted contains various deficiencies, such as holes through is windows, and self-intersecting roof elements.

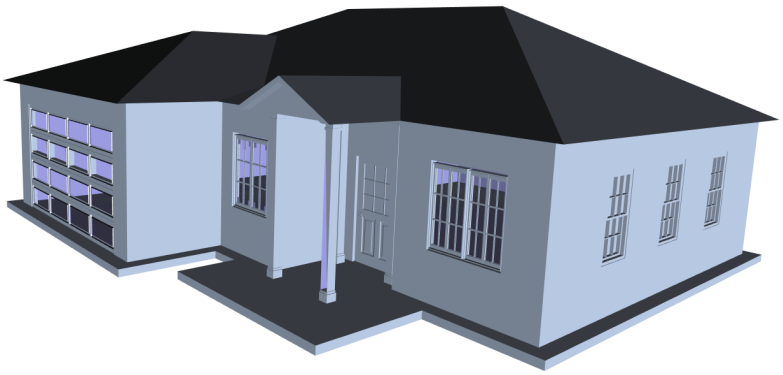


Figure 3.2: Example input model

3.1 PRE-PROCESSING

The first step of the method is to prepare a model for the analysis that will be later applied to it. As has been mentioned in both Chapter 1 and Chapter 2, models available on the web and offline repositories often contain deficiencies, inhibiting any analysis processes performed on them. Two such cases are encountered and confronted in this step

The two deficiencies that are going to be tackled in this step are:

- duplicate geometry, and
- self-intersecting elements.

3.1.1 Duplicated Geometry

Duplicate geometry is a common problem that appears in modern models. It is something that may appear due to the carelessness of the creator, or the person editing the model. Or maybe because of the model's creation through faulty digital applications. Cases of duplicate geometry can have a face being input twice in the data structure, or two vertices possessing the exact same geometrical coordinates, but being perceived as two different entities. An example case is shown in the sketch of Figure 3.3. In that case, the left wall of the building is supposed to exist twice, possibly posing problems to future processing of the model.

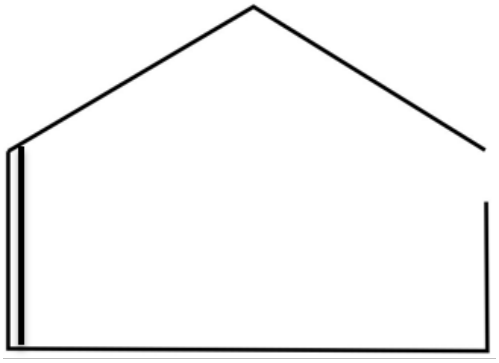


Figure 3.3: Duplicate geometry and holes in a model

The problems that could surface during analysis steps due to repeating components inside a model are various. A main one is that during storage, more space is required as a result of the redundant information being saved. As such, analysis of these models might take longer than required. Furthermore, there could be

cases where topological problems are present because of this. For example, a single vertex appearing as two different entities might cause topological problems (eg. connectivity of two edges or faces). Finally, duplicate geometry makes methods that rely on object counting (eg. ray casting) produce inaccurate results.

Because of this problem, there is a requirement to remove as many repeating surfaces as possible. Through already existing applications (presented in Chapter 4) most of the input model's repeating objects are removed.

3.1.2 Self-Intersections

Another deficiency that can be found in input models is self-intersecting elements. This means that within the model there are invalid intersections, where there is no vertex in the point of intersection. This make a model topologically inconsistent. Again, this is something that may appear because the creator of the model put more consideration in the model's appearance, and not so much in any possibility of the model's future processing. Cases of intersecting elements would be when a wall surface passes through a floor surface, with no information on their intersection being stored. One example of self-intersecting elements is in Figure 1.1. Another one is on Figure 3.4. In the latter, two roof elements are observed passing through each other.

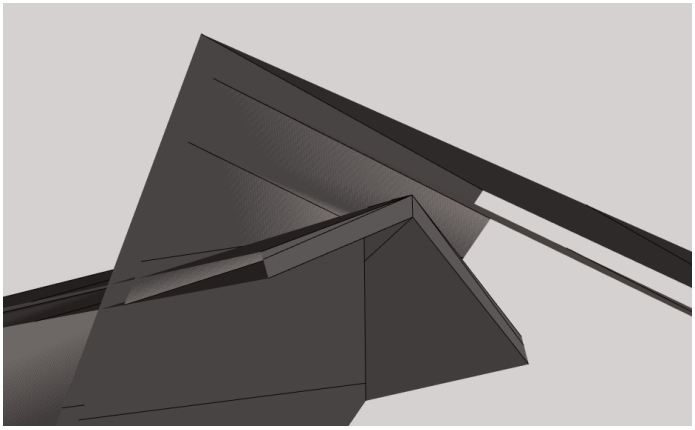


Figure 3.4: Self-intersecting elements. The lower roof element is passing through the other one.

Self-intersections are a drawback to models not from a geometrical, but rather topological viewpoint. Modern storage formats take care to not only store the geometric information of a model's elements (as coordinates), but also the relationships between those elements; most often than not, their connections. Unfortunately, the information of self-intersections is not retained when data is stored, and as such, analysis of models with this flaw may produce unwanted results.

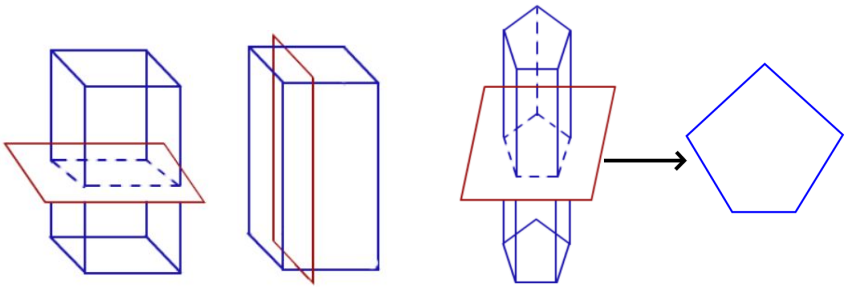
As with the previous case, algorithms that can recognize self-intersections and restructure/remesh the model to remove them exist. Again, the application to remesh the model in order to remove self-intersections will be presented in Chapter 4.

3.2 PLANAR CROSS SECTION CONSTRUCTION

At this point, the model should be self-intersection free, and also rid of most of its repeating objects. As such, it should be cleaned of some of its deficiencies, thus making further processes easier to handle. After this pre-processing step is done, the actual methodology is ready to be implemented.

Up until now, most methodologies have used three-dimensional approaches in order to solve inside-outside classification. In this step, the methodology strives to deviate, and enter a territory that has not been much explored before. Here, the goal is to transform the problem from 3D space to two dimensions. In order to achieve this, we introduce the model's intersection with a plane that contains the point in question for classification.

When a face element is intersecting with a plane object, the intersection result can be the face itself, a line segment, or a point (barring the case that there could be no intersection at all in 3D space) [Sunday, 2012]. All of these are elements that exist in 2D space. Images of 3D objects intersected with planes are shown in Figure 3.5.



(a) Cuboid-plane intersection **(b)** Prism-plane intersection and result

Figure 3.5: Intersection of solids with planes, retrieved from http://bookmarkurl.info/worksheets/cross-sections-of-solids-worksheet.html

Introducing a planar intersection at this step is done for the following reasons. First and foremost, an intersection with a plane allows us to approach the problem of classification on top of the plane. This means that from three dimensions, the problem is transferred in two, as the necessary information is now projected onto that plane. Furthermore, when the plane crosses the model, all intersections with the model's surface elements are accounted. This means that the geometry information of the model related to the certain crossing plain is preserved in its entirety. As such, the result of a planar intersection is that we can retain the finer details of a model, while analysis in 2D space is simpler and less ambiguous than in 3D space.

Of course this method does have some weaknesses against other approaches. The first is that a planar cross-section cannot handle a model as a whole, it is only possible to perform analysis on the area of interest. Also, if there is a need to go back in three dimensions later, it is difficult to patch these information together to present the original 3D model.

Going into the step itself, a plane (that contains the point to be classified) is introduced. The plane is generated by having information of a point, and a vector. The point in case is the point of interest. The vector in question can be created in a steady direction, or be given at random — the plane's orientation generally affects little to nothing of the final result. In this case, the plane generated is passing through the point, and the vector is the normal of the plane.

Since the model is comprised from surface elements, each of these is tested against the plane. In general, the intersection of a plane and a surface element can provide the following results:

- The surface itself.
- A line segment.
- A point.
- No result, if they do not cross.

If they intersect, then the intersection result is kept. In this case, only results of the line-segment type are kept, while the others (polygons and points) are ignored. This is due to the line segments being a sufficient source of information for the following steps of the methodology.

An extra action happening in this step is that of retaining the topological information of the model. This means that if a line segment is derived from a face's intersection with the plane, it should also keep information on its origin. This is made with the goal of keeping the information of connections between the newly-created line segments. Any qualitative information contained by the face (eg. texture, materials), although possible, is not passed on to the line-segment, as it is unnecessary for the final result's production.

If f is a face of the input model, and l a line segment derived from f's intersection with the plane, then l is defined by two vertices (source and target), $v1$ and $v2$. As l is the result of an intersection with f (provided that f and the plane are not coplanar), so are its two vertices results of the plane's intersection with two edges of f, $e1$ and $e2$. By storing in each vertex the information of its origin edge, we know that if vertices from two separate line-segments derive from the same origin edge, then their respective line-segments are connected. So, through the planar cross section, a set of line segments is generated, along with a set of unique vertices. The steps can also be seen in pseudo-code form in Algorithm A.1. This action is performed in order to ensure the connectivity of the generated line-segments, which is a requirement for later steps of the pipeline.

A visual example of this topology-retaining action is shown in Figure 3.6. In this, a roof element is intersected by a plane. The two faces of the roof, $f1$ and $f2$, produce two line-segments $s1$ and $s2$. $s1$ has vertices that inherit information of originating from the edges $e1$ and $e2$. Similarly, line segment $s2$ inherits origin information about edges $e2$ and $e3$. As such, at the end of the generation of line segments due to the planar section, there is information on $s1$ and $s2$ being connected at vertex $v2$, because for both of them it has the common origin of edge $e2$.

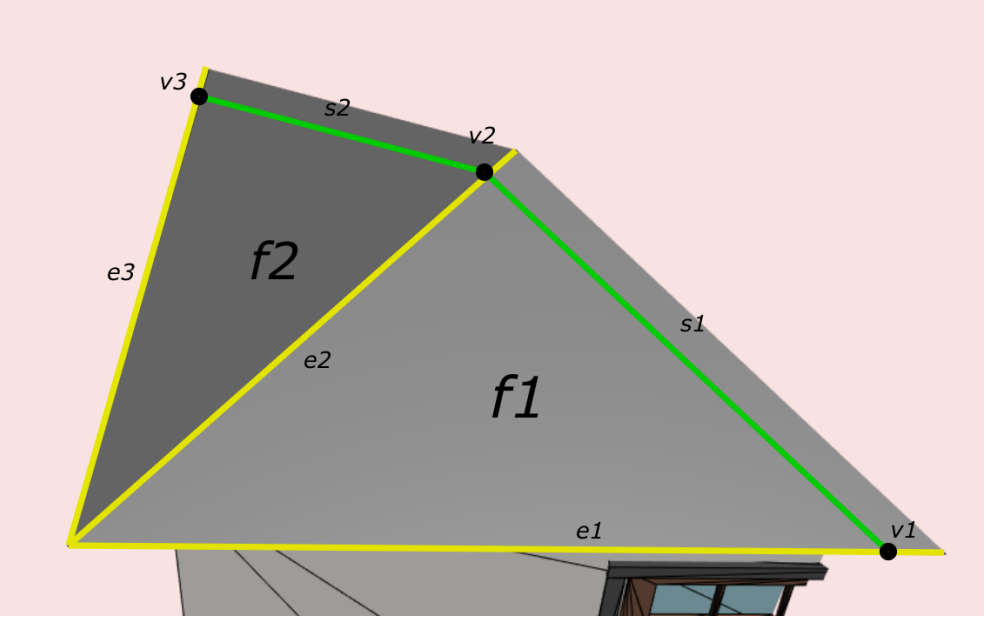


Figure 3.6: Plane intersecting a building model. Original edges of the model are shown in yellow, while the generated line-segments are colored as green.

At the completion of this step, the product should be a sum of line-segments with topology information, representing the geometry at the spot where the plane slices the input model.

A result of constructing a cross-section on the example model can be seen in Figure 3.7.

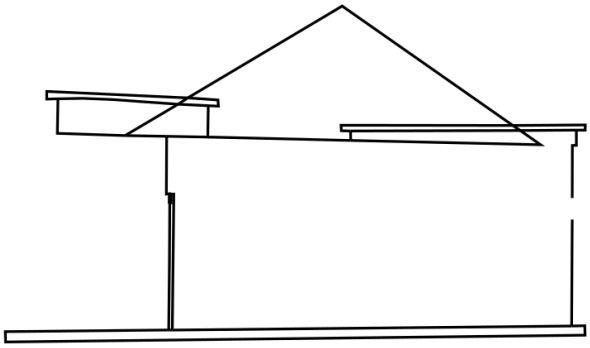


Figure 3.7: Generated planar cross section of example building

3.3 TOPOLOGY-BASED LINE-SEGMENT COMPONENT CREATION

Arriving at this step, the data should be comprised of multiple line-segments representing the cross-section of the model with the plane. In this step, the data that will be processed further along the pipeline will be grouped in a simpler structure.

Depending on the density or size of the input model, we could have a large number of generated line-components, as was explained in Section 3.2. A large number of these elements does not pose a problem in the production of the final result, but a more condensed structure would speed up the computation time of the algorithm. So for this purpose, in this step the entirety of the line segments are grouped in connected components.

A connected component could be defined as a sum of successively interconnecting line segments, or it would also be viewed as a poly-line. Considering the cross-section of this stage as a graph, the basic structure of which are edges and vertices, a connected component in our case can be considered as a sub-graph, or an element of higher order than that of the edge. It is generally a path comprised of multiple edges, where the vertices that lie within it have an adjacency grade of 2 (connected to only two incident edges). Geometrically, each connected component has as endpoints vertices that connect to multiple edges (junctions for multiple line-segments), or connect to only one line-segment. A connected component can be useful because all its contained segments can inherit from it the same qualitative attributes, and iterating through poly-lines is significantly faster than coursing through every single line-segment.

A simplified example is given in the sketches of Figure 3.8. As can be observed, when line-segments are grouped into connected components, less elements are in need of processing on later stages.

In the following subsections, the creation of the connected components is going to be analyzed.

3.3.1 Graph Creation

As has been already stated, at this point, the available data are multiple line-segments. However, they still have no connections between them. In order to create the connected components, what is necessary is an intermediate topological data structure

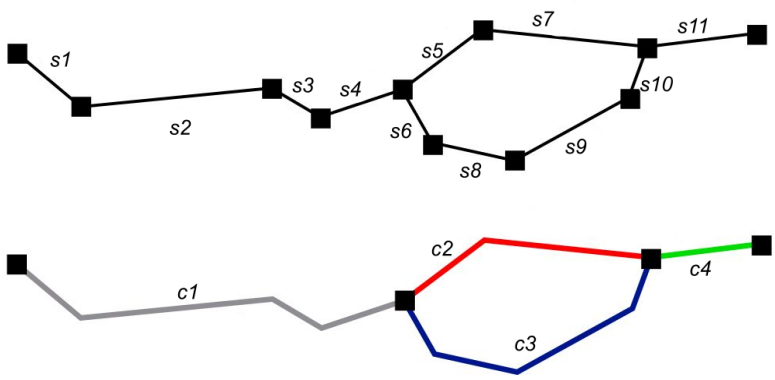


Figure 3.8: Line-segments (above) against connected components (below)

where information on the connections between line segments exists. For this reason, a graph structure was selected. A basic graph structure can be seen in Figure 3.9.

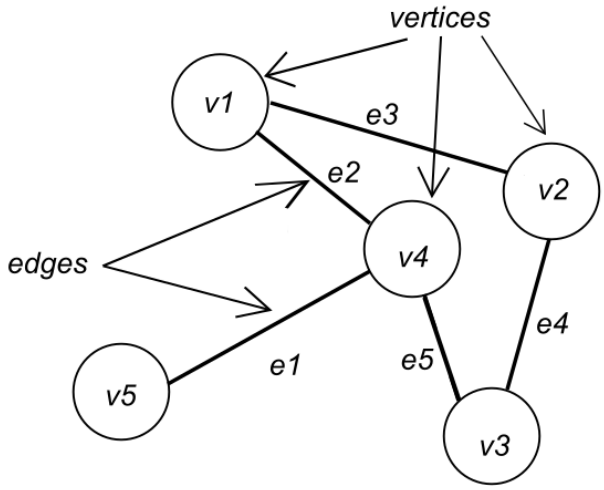


Figure 3.9: Basic graph depiction

A graph's main element is the vertex. The graph is comprised of a set of vertices, and a set of connections between them, called edges [Bondy et al., 1976]. In our case, the vertices are representing the actual vertices (points) defining the line segments, and the line segments are represented as the graph's edges.

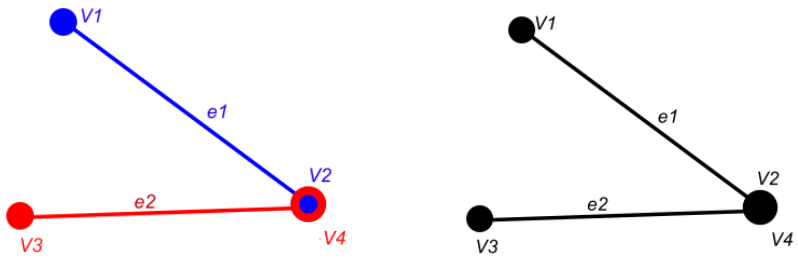
To construct the graph, as a first step we populate it with the vertices of the cross section. So, we traverse the set of the cross-section's vertices, and insert them inside the graph.

The second step is to make the connections within the graph. For that purpose, every line-segment of the cross section is giving information on its source point, and target point. By recognizing these two points as vertices in a graph, it is possible to insert a connection between them in the graph. The connection itself represents the line-segment.

After creating the graph, there could still be deficiencies in its structure, due to the initial model's remaining duplicate geometry and self-intersecting elements. For that reason, before the graph is complete, it is processed by two cleaning methods.

The first one is detecting and removing duplicate vertices. When the same vertex exists multiple times, it means that the corresponding line-segments should be, but are not yet connected.

An example of the existence of a duplicate vertex is shown in Figure 3.10, as well as the result after the duplicate's removal.



(a) Duplicate vertex in V2 and V4

(b) State of graph after removal

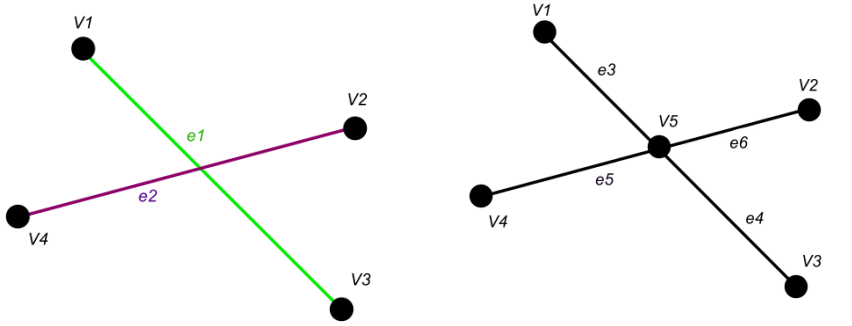
Figure 3.10: Removing a duplicate vertex inside a graph

After that, there is also the matter of lingering intersecting segments. The majority, or the sum of them could have been removed in the pre-processing step of Section 3.1, but still there could be remnants appearing in the cross-section. The problems presented when self-intersections exist have already been mentioned in Section 3.1.2.

The steps to recognize if two edges are crossing each other are as follow. First, edges are searched in pairs of two, in order to check whether they have a common point. If two edges $\overrightarrow{AB}$ and $\overrightarrow{CD}$ have a common point E, then if the newly created vectors have a negative dot product ($\overrightarrow{AE} \cdot \overrightarrow{EB} < 0$) it means that point E lies between A and B. In that case, the two edges are truly intersecting.

When two edges are found intersecting, then the intersection point is inserted in the graph. The initial intersecting edges are removed from the graph, and new edges are introduced from the old edges' vertices to the new intersection point.

An example is shown in Figure 3.11. In this, edges $e1$ and $e2$ are found intersecting. As such, a new vertex, $V5$ is introduced in order to allow the graph have a cleaner structure. All crossing edges are also split and reintroduced in the graph.



(a) State of two intersecting edges

(b) New state of the graph after intersection removal

Figure 3.11: Removing self-intersections appearing inside a graph

3.3.2 Connected Component Creation

After the graph representing the cross-section has a clean and valid structure, it can be utilized in order to construct the connected components.

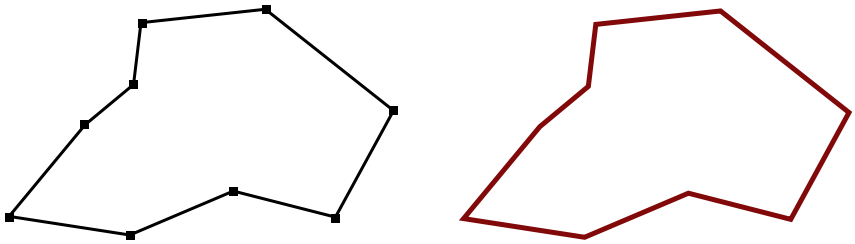
To construct the connected components, the graph's set of vertices is traversed. When a vertex's number of adjacent edges —which is called degree or valency— is two,

then that vertex is an intermediate point of a poly-line, and the two adjacent edges are considered connected and belonging to the same connected component. Thus they are inserted in the same connected component.

When the vertex's degree is one, then the incident edge is simply inserted in a connected component. This case means that the vertex is a connected component's endpoint.

In the case that a vertex has a degree of three or more, then it is not considered intermediate, but a junction. In this special case, the vertex acts as its name implies, and becomes the endpoint for all adjacent edges (end by extent, to their respective connected components). This also ensures that the connected components have polyline geometry, and not forks of any form.

When an object is of a ring topology, the outcome is just a single circular component. An example case of this can be seen in Figure 3.12.

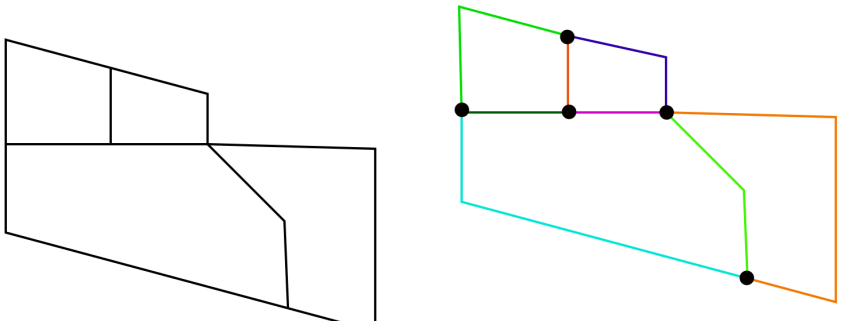


(a) Object with ring-like topology. **(b)** Outcome of one connected component.

Figure 3.12: Handling an element with ring-like topology.

The methodology containing the steps that were mentioned above is given in pseudo-code form, in Algorithm A.2.

One example showing the result of creating connected components can be seen in Figure 3.8. Another one is in Figure 3.13. In the latter, it can be seen how a planar cross-section's structure, comprised of connected components, is represented. Each connected component is represented by a different color, and in this case, their endpoints are the junction vertices that have been mentioned earlier in this subsection.



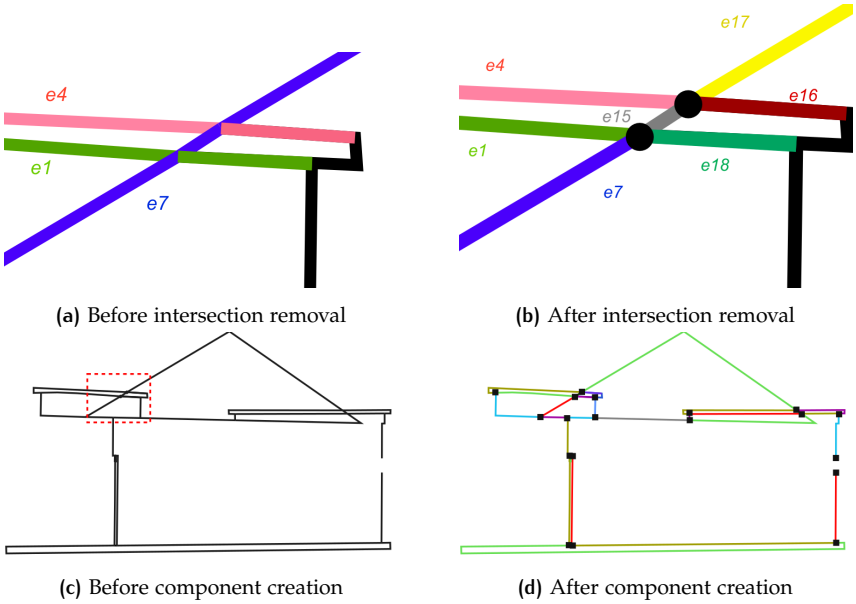
(a) Planar cross-section

(b) Result of connected component creation

Figure 3.13: Constructing connecting components on a cross-section. The result is a total of seven components.

At this step, a list of all the vertices with degree other than 2 is kept, comprising all the endpoints of the created connected components. The purpose of this is going to be explained in later sections of this chapter.

Figure 3.14 shows the changes on the example model after performing this section's steps on it.



(a) Before intersection removal

(b) After intersection removal

(c) Before component creation

(d) After component creation

Figure 3.14: Results of component creation on example building. (a) Intersecting edges of cross section. (b) Result of dealing with intersections. (c) Building's cross-section before component creation. Position of intersection removal example shown in red rectangle. (d) Connected components created for building example.

3.4 CLOSING GAPS: TRIANGULATION

Arriving at this point, the available data is a cross-section comprised of multiple connected components. Having this, we could go from the step described in Section 3.3 directly to the one of Section 3.5.

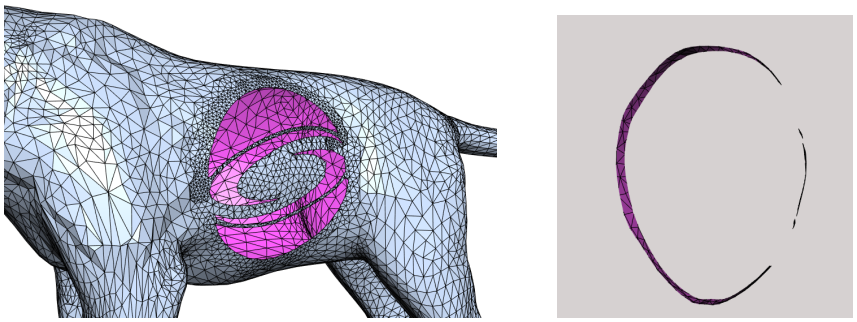
However, another thing that is demanded from the end-result of this pipeline's process is a watertight object. This means that apart from the other types of deficiencies, (duplicate geometry, self-intersections) the model should also be rid of holes.

Holes do not pose a validity problem in the data structure, but in order to be able of performing robustly the classification of an object as being interior or exterior, it is required of the actual model (cross-section) to be devoid of holes. Reasons for why holes are undesired for these types of methods have also been mentioned in Section 2.4.

A basic example of a hole in a model was depicted in Figure 3.3's sketch. Another example with a real model is in Figure 3.15. In this case, the model is that of a big cat. In the 3D representation, the hole allows the user to view the interior of the model, shown with light purple color. When a planar cross-section is applied, the holes are still visible as a discontinuity on the right side.

The selected process to close holes is by implementing Delaunay Triangulation (DT). In this case, the cross-section is populated by triangles and edges. Out of the new generated edges, some are selected methodically, in order to fill the pre-existing holes.

First, a set of points has to be selected on which the DT will be applied. For this, the connected component's endpoints are used. Out of the total of those endpoints, only those with a degree of 1 are selected. This means that only those that potentially are endpoints of open gaps contribute to the DT's creation. Also, each of the selected endpoints constitutes one of the triangulation's vertices.



(a) Hole in 3D. **(b)** Hole in 2D

Figure 3.15: Holes in cat model. (a) Hole seen in 3D. Interior of model shown with purple color. (b) Holes seen in 2D after slice of model. 4 gaps are observed in the side of the cross section.

When the triangulation is performed, the result is newly created triangles and edges, as has been mentioned. Out of all these edges, only a portion of them will be selected and kept, the ones that will potentially close the holes.

The method to select the edges for extraction is to first traverse all the points of the DT. For each point, its incident edges are checked. The edge with the least length is kept, as it is the one that is filling the hole. At this stage, the assumption is made that the gaps a model has are small enough when compared to its actual size. The selection of edges is also depicted in Figure 3.16. In this case, edge $e3$ is the shortest incident edge to point a, and as such, the one to be selected.

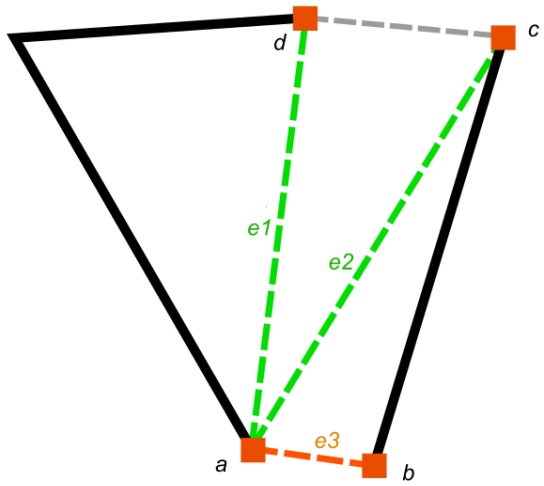


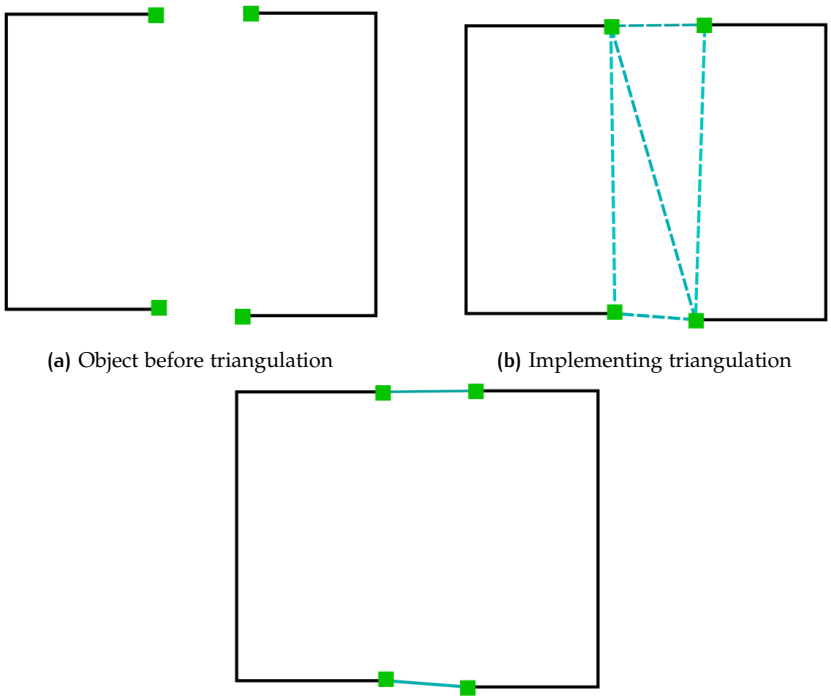
Figure 3.16: Selection of edge to keep. The orange colored edge is the shortest edge that is adjacent to vertex a, so it is the one being selected.

A general application of closing gaps with DT can also be observed in the example of Figure 3.17. There, The endpoints are recognized, then triangulation is performed on them, and finally, only the edges that fill the gaps are being kept.

The extracted edges are introduced to the cross-section's graph. Then, the graph goes again through a cleaning process (removing duplicates and self-intersections), to make it ready for the next step of the methodology.

The result of applying triangulation on the example cross-section is shown in Figure 3.18.

There is also a possibility that an edge introduced through triangulation intersects an already existing one. When that happens, the point of intersection is introduced



(a) Object before triangulation

(b) Implementing triangulation

(c) Closed Holes

Figure 3.17: Closing gaps with Delaunay Triangulation

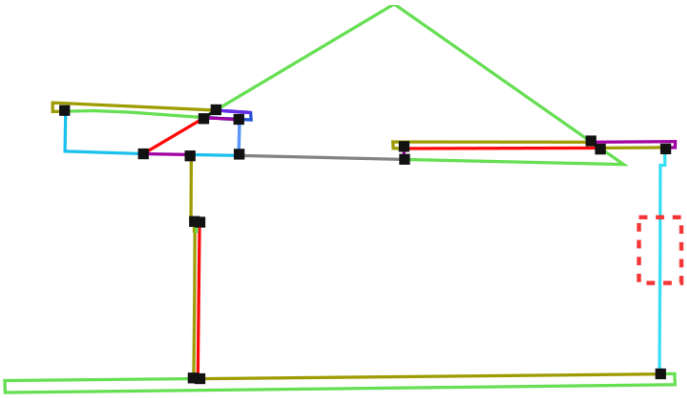


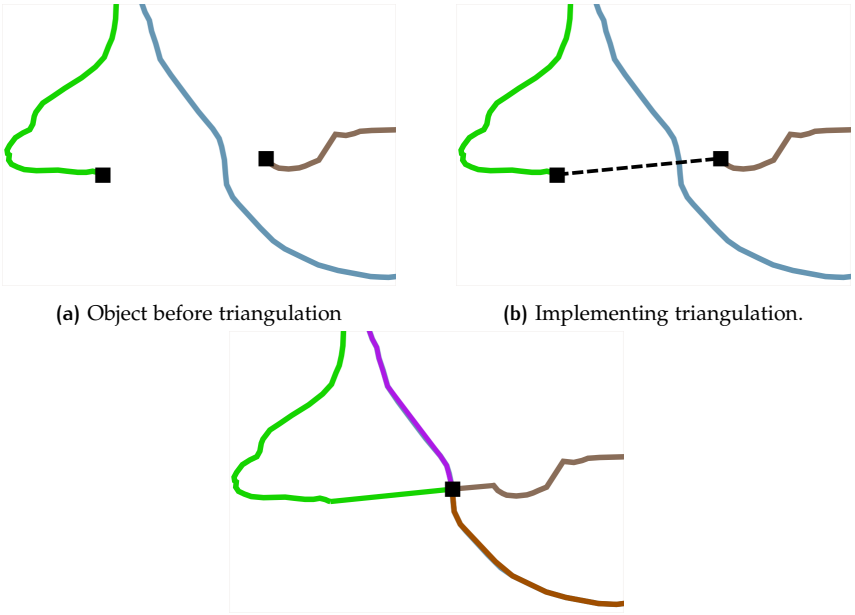
Figure 3.18: Closed hole of window in example cross section, next step from Figure 3.14. Closed area highlighted within the red rectangle on the right side.

as a new vertex within the cross-section, in the same way as has been described in Section 3.3.1. An example of how this is handled is shown in Figure 3.19.

3.4.1 Alternative Triangulation

During the implementation of DT, there were observed cases where the selection of only the endpoints from the connected components was not sufficient. The geometry of a cross section might present holes, but they were not surrounded by endpoints.

In that case, as points for the triangulation are selected all of the cross section's points. Then, the evaluation of which edges are going to be preserved is applied to every vertex. This alternate selection of triangulation points is selected only in cases where selecting just the endpoints was not sufficient.



(a) Object before triangulation

(b) Implementing triangulation.

(c) Newly created vertex.

Figure 3.19: Handling new intersections appearing through triangulation's application (a) Input having gap and a component passing through it. (b) Edge creation through DT. (c) The point of intersection is introduced as the new vertex. Components are restructured to accommodate the change.

An example case can be seen in Figure 3.20. Here, applying triangulation on just the endpoints presents new edges as seen in green line. By utilizing all the vertices, the hole is closed in the way that is shown with the red line.

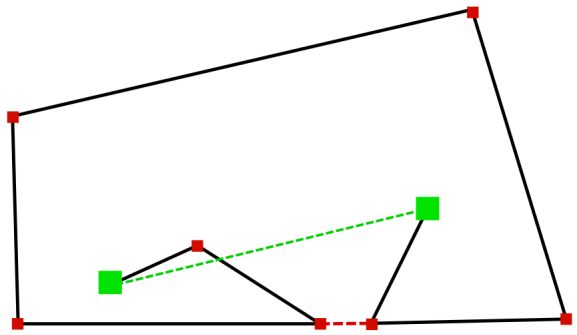


Figure 3.20: Selection of edge to keep.

3.5 TWIN RAY VOTING

Arriving at this step, all necessary groundwork has been laid. There is a set of connected components, the sum of which represents the planar cross-section of the model. At this point, the geometry is clean, meaning that no self-intersecting elements remain, the vast majority of duplicate objects have been removed, and the geometry is now watertight, with all adjacencies and connections stored as information.

This step's purpose is to extract the outer borders of the model, defining anything within them as lying on the inside. The rest will be outside. For this, an arithmetic value will be assigned to each connected component and consequently to the line-segments that comprise them.

It has been decided that every component will be assigned one value, called border vote. The vote's minimum value will be 0, and the higher the value is, the bigger the confidence of the component being an outer border. The value to define this will be calculated by implementing a ray casting method.

As can already be derived from the name, the main concept of this method is to generate rays that will help in classifying a component as being an outer border or not.

The first step is to assign the border index for the core element of a cross-section, the line-segment. For each line segment, its centroid is calculated. Then, from the centroid, and with random direction, a ray is generated. For our method to function —and also give more valid results— when a ray is generated, its twin ray is also generated in the opposite direction. An example is seen in Figure 3.21. In a line-segment with vertices $v1$ and $v2$, the centroic c is calculated, and from there the twin rays r_A and r_B are generated.

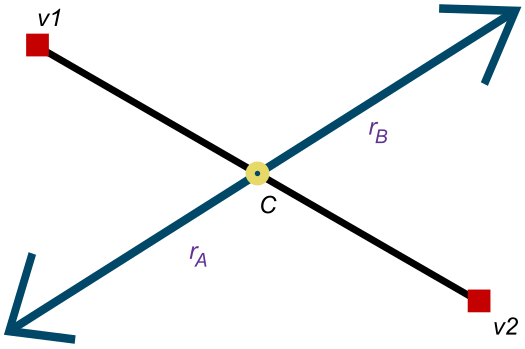


Figure 3.21: Generation of twin rays from a line-segment's centroid

The generated rays may pass through other line segments, but not the one they originated from. The number of intersections for this pair of rays gives information of whether the origin segment is an interior one, or part of the cross-section's outer border. For this case, if i_A is the total number of intersections that ray r_A has, and accordingly i_B the intersections of ray r_b, the classification comes from the following decision:

$$Result \begin{cases} outside & i_A mod 2 = 1, i_B mod 2 = 0 \\ inside & i_A mod 2 = i_B mod 2 \end{cases}$$

The above explains that if the remainders of the respective numbers of the twin rays' intersections, divided by two, are different, the segment is considered a border. Otherwise, it is considered interior information.

Since most models may have irregular shapes, deciding from one twin ray casting will possibly not always give robust results. As such, this process is repeated several times, and each repetition consists one vote. For the sake of this pipeline, tests have been done ranging from shooting 4 to 50 twin rays, and it was decided that time and quality wise, shooting 20 twin rays for the voting process is acceptable.

The average result of all these votes consists the border vote value for the segment. Furthermore, in cases where $i_A = 0$ or $i_B = 0$, it is understood that the segment has an open view to the exterior of the model. When this happens, the vote gets a higher weight, since the main goal of this step is to make possible the recognition of the model's cross-section's borders. An example case of twin-ray voting is presented in Figure 3.22.

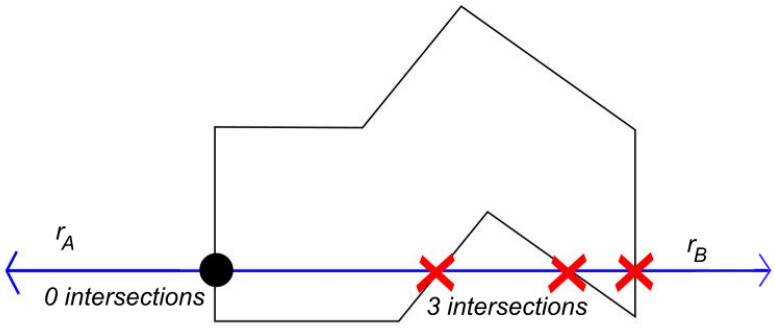


Figure 3.22: Classification of element based on twin ray result

When the line segments have their border vote, they can assign it to the connected component that they comprise. The vote is the average sum of votes from the line segments, where the values are weighted based on the length of the segments; longer segments have more weight in this. The formula for computation is

$$C_v = \frac{\sum_{s=1}^{n} l_s \cdot v_s}{l_c} \tag{3.1}$$

where C_v is the component's border vote, l_s the length of segment s, v_s its border vote, and l_c the total length of the connected component. The connected component consists of n line-segments.

The flow depicting the process of this method is presented in Algorithm A.3.

The end result of this step is that every connected component has a value that can help in deciding if the component belongs in the cross-section's outer border. One example, with colors that depending on the vote values is shown in Figure 3.23.

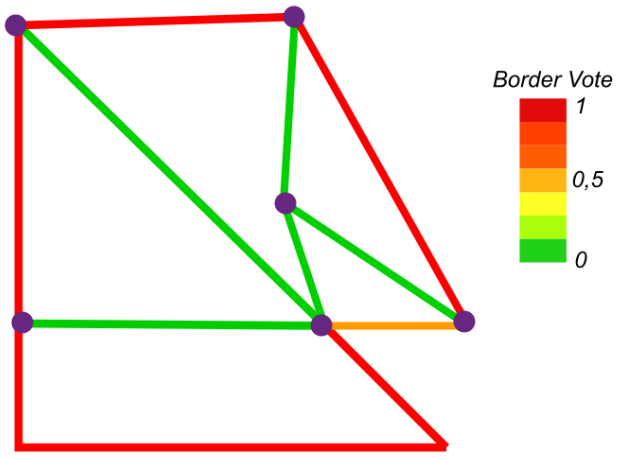
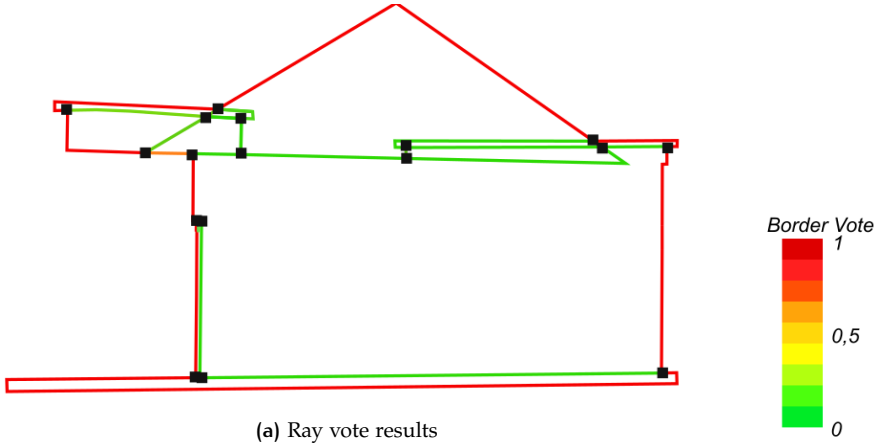


Figure 3.23: Colored result of the twin ray casting votes

A result on the border voting on the example cross-section can be seen in Figure 3.24.

3.6 EXTRACT BORDERS: OPTIMIZATION

Arriving at this step, it is now possible to extract the outer borders of the cross-section. Until now, there have been processes to remove geometrical and topological deficiencies from the model, to make it watertight, and to assign border confidence



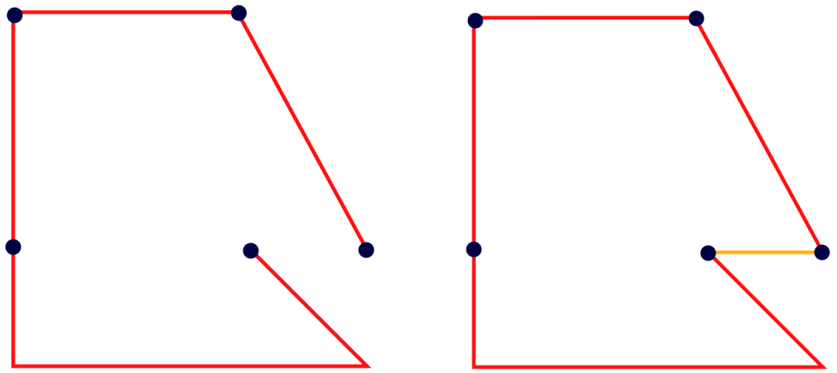
(a) Ray vote results

Figure 3.24: Colored result of the twin ray casting votes on the example cross section.

values on the section's line-segments. Now it is time to use all the data gathered until this moment to extract the cross-section's exterior shell.

The purpose of extracting the borders is in order to perform inside-outside classification based on ray casting. The intended result is to retrieve a closed polygon, on which ray casting can be robustly applied.

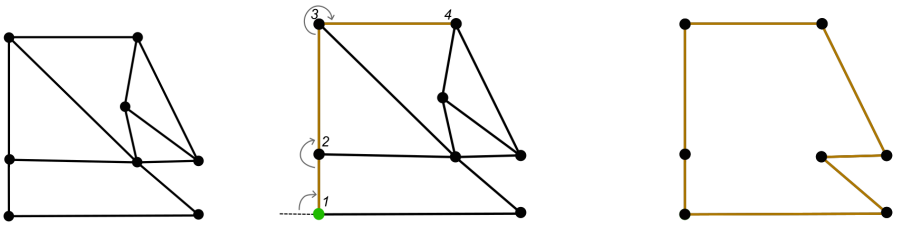
A first option in this case was to use a threshold on the border vote indices. By applying this threshold, only line segments with border value above it would be selected. This would ensure that most segments looking outside would be selected, but in case of concave shapes, some of them with not so high values might be omitted. What is desired in this step is to have an outer border with a ring-like, 1-manifold topology. For this reason, instead of applying a simple threshold, it was preferred to introduce an optimization problem for this selection. The differences of these two approaches can be evident in Figure 3.25, where the two methods are applied in the shape from Figure 3.23.



(a) Extraction by threshold **(b)** Extraction by optimization

Figure 3.25: Outer border extraction by threshold vs. optimization

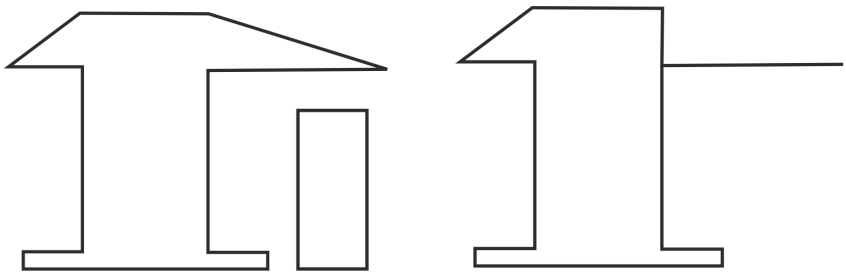
Another potential method for border extraction was that of curve tracing. Basic steps of this method are shown in Figure 3.26. In this case, an outer vertex (to ensure its selection, the bottom-left one could be selected) is assumed to be on the border of the shape. On a set rotation (eg. clockwise), the vertex's incident edge that is next in rotation order is selected. The output provided is a polyline representing the boundary of the input complex geometry, similarly to Ohori et al.'s algorithm. In the end, the result would be the initial input's outer border.



(a) Input Geometry **(b)** Border Tracing **(c)** Output

Figure 3.26: Outer border extraction by border curve tracing. (a) Input Shape (b) Starting with vertex 1, vertices 2, 3 and 4 are consequently selected by clockwise order. (c) Output of border tracing.

Although border recognition through curve tracing is a valid method, there are cases of input geometry that it cannot handle robustly. These cases are shown in Figure 3.27. In Figure 3.27a two disjoint elements are shown. Curve tracing can procure the outer border of one element, but it would completely disregard the existence of the other. Furthermore, in cases with dangling elements, such as in Figure 3.27b, the output would have the dangling element remaining. Still, optimization would remove the dangling element, and generally cope with both of those cases. As such, it was selected as the optimal option for the border's extraction.



(a) Disjoint geometry **(b)** Dangling elements

Figure 3.27: Geometric & topological limitations of border curve tracing.

Optimization is a problem of linear programming in this case [Bradley et al., 1977]. The result of the problem's solution is the selection of the connected components that are suited to be the outer borders of the cross section. The result of this extraction would have manifoldness, meaning that any remaining objects should have ring topology, and no holes.

3.6.1 The Optimization Problem

The problem introduced is as follows:

The main objective is to maximize the value of the sum of the connected component's border votes. This means, that the solution will prioritize the components with higher border votes, thus the ones more likely to be outer borders. In this case, the value is called Border Energy (BE), its objective is to be maximized, and it is calculated by the following equation:

$$BE = \sum_{i=1}^{n_c} b_i \cdot x_i \tag{3.2}$$

where:

- i denotes each connected component,
- n_c is the total number of connected components in the cross-section

- x_i is the value defining whether the i^{th} component is going to be selected, with the only two possible values being 1 for selection, and 0 for rejection
- b_i is the i^{th} connected component's border vote value

Of course, by just attempting to maximize the Border Energy value, the result would be the selection of every component. In this case, as has already been mentioned, we want to extract an outer border that is continuously connected. For this reason, constraints are inserted to the optimization problem.

The inserted constraints are:

$$A_e = \sum_{i=1}^{n_{ac}} x_i = 0 \text{ or } 2 \tag{3.3}$$

where:

- A_e is the e_{th} endpoint's degree of adjacency, e belonging in the set of existing endpoints for the cross-section
- n_{ac} is the total number of connected components adjacent to the particular endpoint

The above constraint means that each endpoint, at the solution of the problem, should have either two adjacent connected components, or none. As such, there are as many pairs of these constraints as there are endpoints in the cross-section. This ensures that the produced result will have ring topology.

The two alternative constraints from Equation 3.3 can also be restructured into the following single equation:

$$A_e = \sum_{i=1}^{n_{ac}} x_i + 2 \cdot (1 - w_e) = 2 \tag{3.4}$$

or rewritten as

$$A_e = \sum_{i=1}^{n_{ac}} x_i - 2 \cdot w_e = 0 \tag{3.5}$$

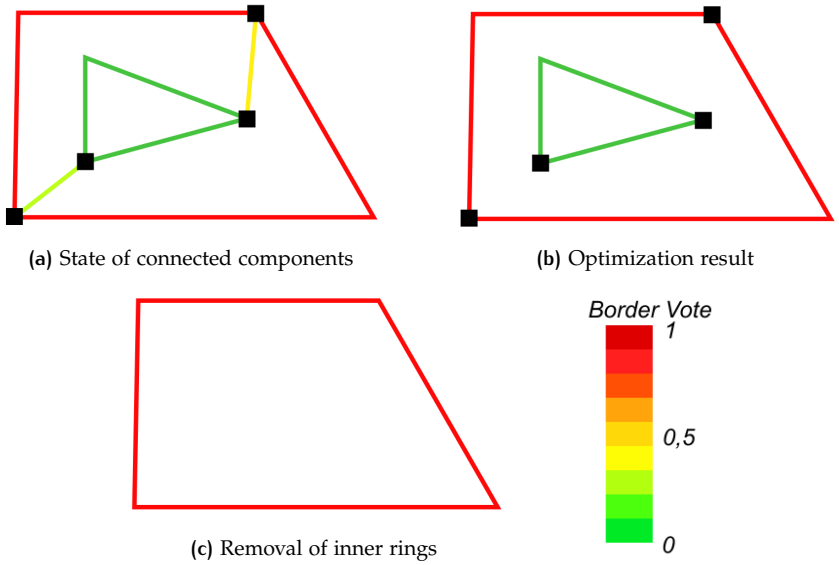
In this case the value w_e is inserted. This value denotes whether the endpoint will be kept in the end result (with value 0), or removed (with value 1). In the case from Figure 3.23 to Figure 3.25 it is observed that the endpoint lying in the interior of the cross-section is also removed.

When the energy maximization problem along with constraints have been defined, the formulas are inserted into an optimization solution algorithm. The output from the algorithm is the selection of those components that maximize the total border energy, while at the same time satisfy the constraints' requirements.

A final thing to take note is that the optimization solution produces a result that has ring topology. This means that it would also be possible to have multiple rings. For this reason, all interconnected components are grouped into ring elements. Then, a formula similar to the one of Equation 3.1 is used to compute each ring's border vote. The final step is to apply a threshold into the border votes of the rings, keeping those that surpass a certain value (eg. 0.5). Since the interior rings generally have low border vote values, they are easily removed from this threshold's application.

The process of border extraction through solving the integer optimization problem can be seen in Figure 3.28. Initially, the state of the connected components along with their border vote is given, and in the end the reader can see the result of the outer border being extracted.

The outcome of the optimization problem's solution, along with the final extracted outer border for the example building model can be seen in Figure 3.29.

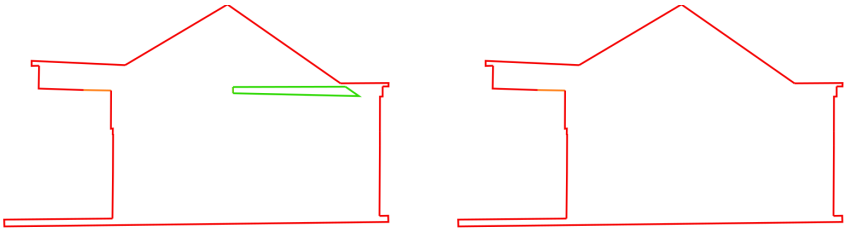


(a) State of connected components

(b) Optimization result

(c) Removal of inner rings

Figure 3.28: Extraction of outer border



(a) Result of optimization solution.

(b) Final outer border.

Figure 3.29: Extraction of the example building's outer border. (a) Result of applying optimization solution on the cross-section. The extracted results have ring-like geometry. (b) Removal of interior rings, outer border remains.

3.7 INSIDE – OUTSIDE CLASSIFICATION

The steps up until now have produced a 2D planar cross section of the input model, refined it, and finally extracted the information about the perceived outer borders of the particular cross section.

At this stage solution to classifying a point as being inside or outside a polygon has already been given by Shimrat [1962]. The solution entails generating a ray from the point. Then, according to the number of surfaces the ray crosses, the point is classified as lying inside (or outside) of the polygon.

When the number of intersections is odd, that means that the point lies inside the polygon. In the opposite case, when the ray crosses the polygon's boundary an even number of times, that means that the point lies on the exterior of the polygon.

This solution works on simple polygons, whether they are of convex or concave hull. These polygons should not have repeating elements (duplicate edge on the boundary) or holes in their boundary, in order for a correct and robust result to be produced. That has been the reason why each and every step of the pipeline until now was performed.

The same solution can be applied for the present case. By shooting a ray from the point in question, we can figure out if it belongs on the interior or the exterior of the model. To ensure the result's correctness, it would even be possible to shoot multiple rays from the ray, and by rule of majority decide on the point's position.

This would happen to avoid very rare cases where a ray intersects the boundary on a vertex, thus crossing both of the vertex's adjacent edges.

An example is shown in Figure 3.30. There, points A and B lie on the interior, since their respective generated rays cross the boundary an odd number of times. On the other hand, points C and D lie on the exterior, as the number of intersections from the generated ray with the boundary are odd.

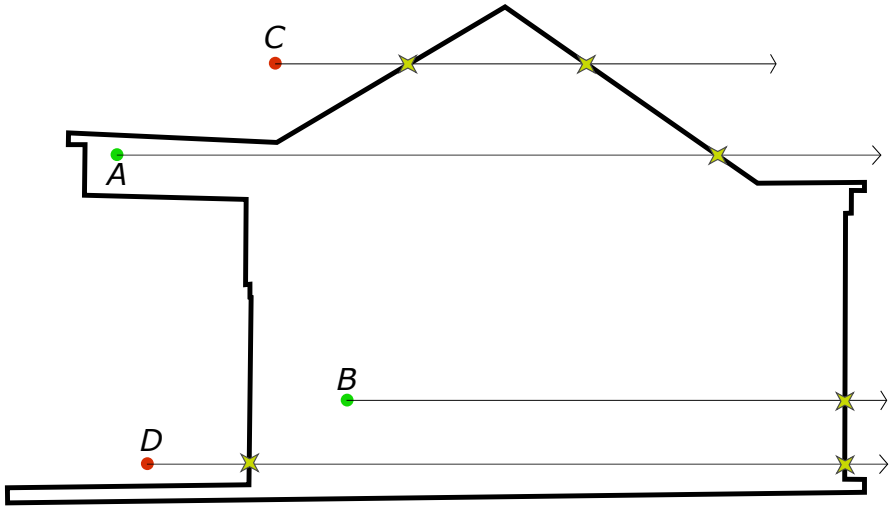


Figure 3.30: Inside-outside classification by ray casting

4 IMPLEMENTATION

This thesis can be a bit demanding on the technical side. Therefore, it is necessary that there is enough computational power, as well as software relevant to the needs of the methodology. Of course, the type of data handled should be of a certain type, and also attention should be paid on the way they are handled.

In this chapter, the aforementioned needs are going to be addressed. First, the type of data used is going to be mentioned, along with their source. Then, the tools utilized for analysis, viewing and evaluating purposes will be presented. Finally, certain choices to get satisfying results, help with the methodology's robustness, and the overall quality will be explained.

4.1 DATA

In order to test the proposed methodology, it is necessary to have a set of data where the analysis part of the thesis is performed on. For this purpose, certain 3D model representations were used.

The models used in this case were 3D surface meshes, representing various common objects found in the world (buildings, cars, animals, furniture, persons, etc.). In this case, the models used had their information stored as Object File Format (OFF) files. OFF is a file format used for storing geometry information in the shape of vertices, and polygons pointing to the vertices they are comprised of. Other file formats, such as the Polygon File Format (PLY) could also be used as input. However, at this point the created program for the methodology's pipeline can read only OFF files. It is possible to work on objects saved as other file formats, if there is a way to convert those files into OFF.

For the purpose of testing the methodology, 3D models were retrieved from various sources. Some models were retrieved from data stored and processed for the purposes of Jacobson et al. [2013]'s methodology, containing a total of 35 models, consisting mainly of animals, but also other forms. Another source was from the personal repository of Liangliang Nan[1]. This one had tens of models that were mostly building, car and airplane representations. Finally, a large amount of hundreds of 3D models was retrieved from the Princeton Model Repository (ModelNet)[2], which contained a plethora of CAD and OFF format models.

4.2 TOOLS

For the purpose of data analysis needed within this thesis, the computations will be carried out by algorithms that are entirely created in the C++[3] programming language.

1 https://3d.bk.tudelft.nl/liangliang/
2 http://modelnet.cs.princeton.edu/
3 http://www.cplusplus.com/doc/tutorial/

For more demanding processes and geometric analysis, the Computational Geometry Algorithms Library (CGAL), [CGAL, 2018], is used. This library contains code, functions, data structures and other useful objects (structures for lines, points, meshes, triangulation, etc.) when handling problems related to geometric analysis.

Most calculations are executed within a compiled and self-edited program containing its own Graphical User Interface (GUI), called Surfacer [4]. Its GUI was created with the help of QT Designer[5].

Finally, it is important at various intermediate steps as well as the final, to be able of monitoring and evaluating the pipeline's results. For this reason, various software were used. For starters, and implementation was performed in Liangliang Nan's Easy3D[6], to create viewers for the produced connected components, their endpoints, and of course, the final extracted outer borders. Also, for monitoring reasons were used two different programs: and interactive 3D model visualizer called Mapple[7], and a mesh processing system called MeshLab[8].

4.3 PROTOTYPE IMPLEMENTATION

From having prepared the theoretical part to actually forming the ideas into an executable, it takes a considerable amount of work. Some steps were relatively easy to put into practice, while others demanded more attention, due to the nature of programming languages and actual mathematical analysis.

This section will present some of the options selected during the implementation process of this thesis. These options were made at points where digressing can really alter the result, or where more specialized actions where needed in order to reach a satisfactory product.

4.3.1 Pre-Processing

For the purpose of detecting geometrical deficiencies, multiple implementations already exist. As such, there was not much need in exploring this particular field. Instead, already existing implementations can be used. In this case, one tool mentioned in Section 4.2, Mapple, was used.

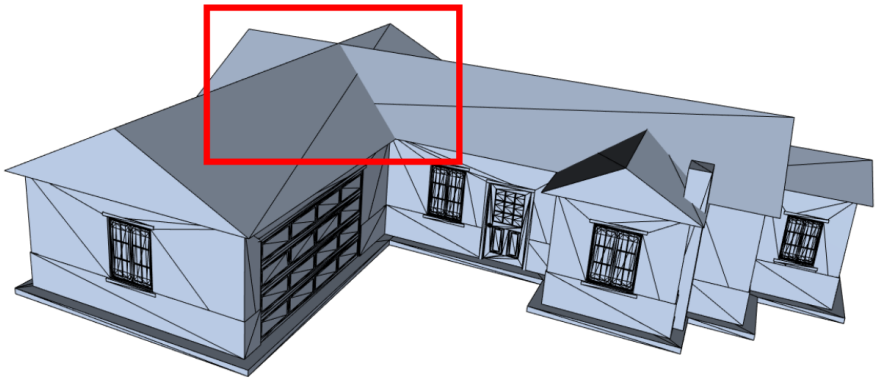
As has been mentioned in Section 1.1 and Section 3.1, it is important to identify deficiencies such as duplicate geometry and self-intersections, and deal with them.

In this case, the 3D model editor of Mapple is sufficient. Having an input model, there are functions to recognize and remove repeating faces. It is also capable of identifying self-intersecting faces and also restructure (re-mesh) the model to create new non-intersecting ones, with a method similar to that of Section 3.3.1 and shown in Figure 3.11.

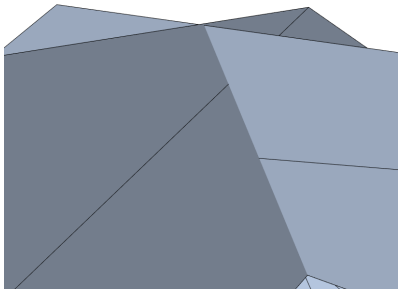
One example of the pre-processing step of the methodology will be explained in the following images of Figure 4.1. This will be performed on a building representation. In Figure 4.1a is the image of the building, as it is viewed from Mapple's GUI. In Figure 4.1c a planar section is applied on the model. From there, it is obvious that the surfaces of the two roof elements are crossing each other. By applying Mapple's remeshing function on them, the difference between Figure 4.1b and Figure 4.1d is visible. The theory behind the remeshing step is the same as the one presented in Section 3.3.1, but is applied in 3D. In the newer state, the existing triangles have been

4 https://github.com/LiangliangNan/Surfacer/tree/Surfacer-Nikos
5 https://doc.qt.io/qt-5/qtdesigner-manual.html
6 https://github.com/LiangliangNan/Easy3D
7 https://3d.bk.tudelft.nl/liangliang/software.html
8 https://www.meshlab.net/

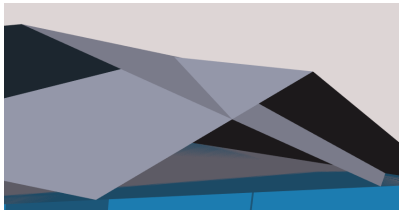
split into finer ones. Also, it can now be seen that the two roof elements are now touching (adjacent to each other) instead of crossing. The proof of that is the mesh line that has now appeared where the two roof elements meet.



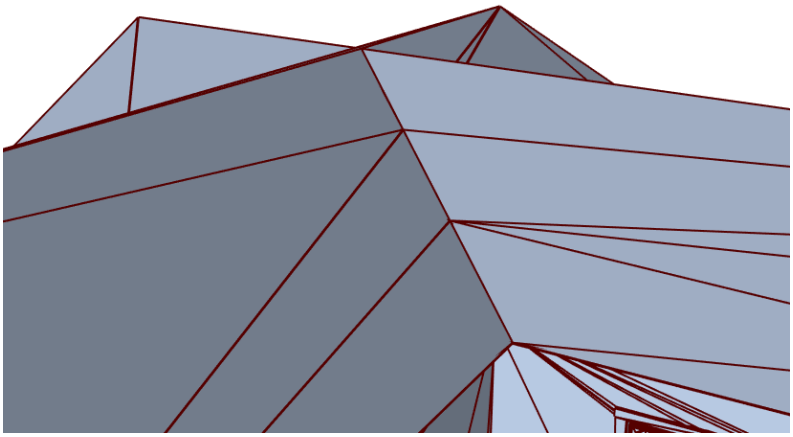
(a) Input building representation



(b) Roof of model



(c) Intersecting roof surfaces



(d) Remeshed product

Figure 4.1: Remeshing 3D model to relieve of self-intersections. (a) and (b) Intersecting roof faces of the building model. In the intersection of the elements, no mesh wireframe is visible, as the intersection's existence is not registered in the model. (c) Cross-section view of the two intersecting roof elements. (d) After remeshing, the wireframe now appears where the two roof elements meet.

4.3.2 Arithmetic Accuracy

An important issue when dealing with high-precision numbers in programming application is the arithmetic accuracy.

Arithmetic accuracy during computations in this case means that numbers with same digits until a certain point are treated as equal. A small example of how arithmetic

accuracy works would be that by having accuracy up to the 3rd decimal of a number, numbers 3.3424 and 3.3427 are treated as the same.

This treatment has repercussions in Cartesian coordinates (as well as any multi-dimensional space) and generally geometrical objects in space. One such case would be seen in Figure 4.2. The image shows that A and B are two individual, independent of each other points. However, due to arithmetic accuracy in this case (the precision along the two axes), since their possible position areas overlap, they would be treated as the same point.

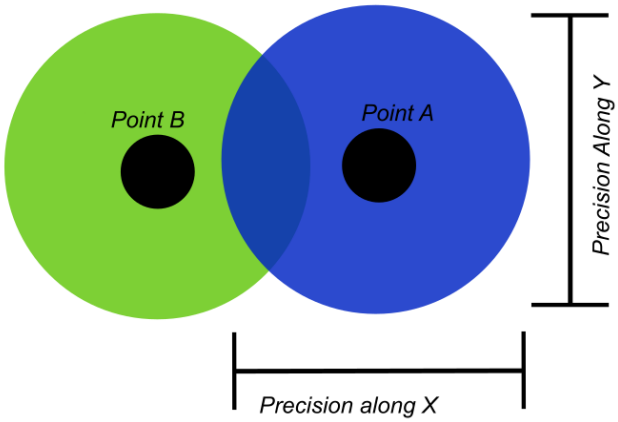


Figure 4.2: Precision of points in space

Another case would be that of the plane intersecting the input model, as seen in Figure 4.3. When the plane intersecting the model passes very close to one of its vertices, the result would be multiple small segments. Due to the arithmetic accuracy of the program, the vertices of these new points would be treated as the same point (existing in the same coordinates). In that case, not only is the geometry of the initial input altered, but the topology would also be compromised. All the generated segments, $\overrightarrow{p_1p_2}$, $\overrightarrow{p_2p_3}$ and $\overrightarrow{p_3p_4}$ would be degenerate, equivalent to points, or the same point in this case. This would lead to unwanted results later on in the pipeline.

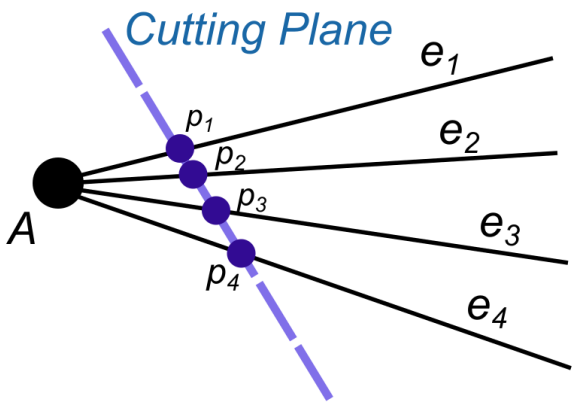


Figure 4.3: Plane intersecting close to model's vertex

To avoid this problem of arithmetic accuracy, CGAL offers the option of working with the provided arithmetic accuracy, or to choose the use of exact numbers. This option is called "exact predicates exact constructions" [CGAL, 2018], and it means that arithmetic accuracy is applied neither in the input data nor in those generated within the program. This option slows down the analysis' execution, but was deemed necessary for preserving the results' quality.

4.3.3 Creation of Viewers

Working with geometry data makes it necessary to monitor the progress done during each step performed within the pipeline. This necessitated the creation of functions that would extract the geometry information during intermediate stages of the pipeline, but also to create applications that would enable the visualization of such geometry. As such, functions where implemented for the extraction and visualization of:

- The line-segments comprising one cross-section. This is a simple geometry visualization, with no special further information, or coloring.
- The connected components of a cross section. Each component is shown with a separate randomly generated color, to differentiate them. An example is given also in Figure 4.4.
- The endpoints of the connected components, along with their degree of connectivity value. Based on their degree, endpoints appear as black when having a value of one, red when having value of two, and green when the value is three or more (meaning they are junctions). An example of a screenshot during the intermediate state of a cross-section can be seen in Figure 4.4.
- The section's line segments, along with their border vote value. An example is seen in Figure 4.5. Red color means that there is a high border value, blue means that it is low (and thus inside). Intermediate —but still relatively low — values range from light blue to green to yellow.

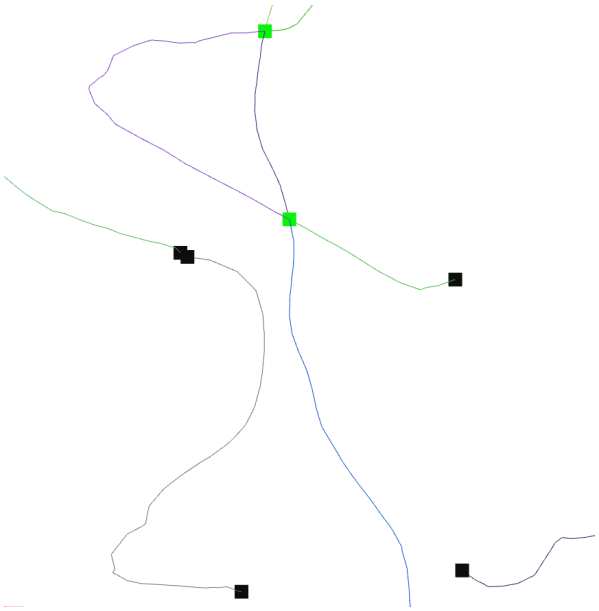


Figure 4.4: View of connected components along with their endpoints

The viewers were created with C++ code, but also as an implementation of Easy3D.

4.3.4 Triangulation Options

In Section 3.4 a way has been presented for closing potential gaps in the planar cross-section through DT. Also, there are two possible methods to apply triangulation: one by utilizing only the connected components' endpoints, and the other by selecting and inserting inside the DT all of the existing vertices.

Figure 4.5: View of segments after ray vote. Parts with higher border vote appear as red, while those with lower vote have blue color.

Inside Surfacer, the option to select either of those methods has been given to the user. There, when the user is at the step of implementing DT, he/she can choose which set of data are going to be used as input.

There may be occasions where the cross-section does not have any gaps. This could be seen by the user by utilizing the viewers specified in Section 4.3.3. There, the geometry of the cross-section can be monitored by observing the connected components and their endpoints.

Since the function of applying DT to close any gaps has been implemented as a separate one, it can be totally bypassed by the user if there is no need for it.

4.3.5 Twin Ray Generation

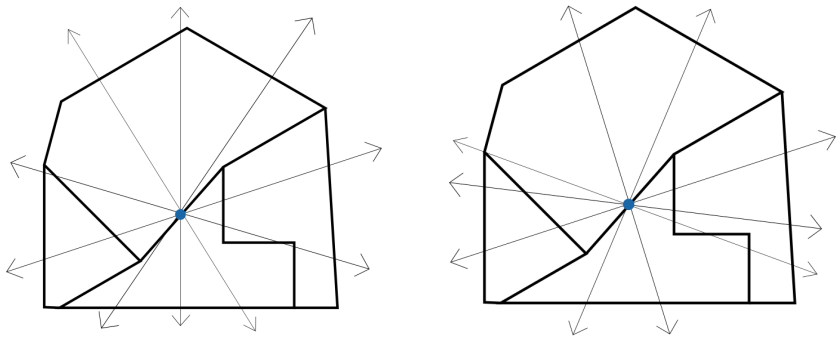
The way that twin rays are created in order to assign border confidence values to the cross-section's line-segments has been elaborated in Section 3.5.

If the cross-section was of valid, manifold structure, one iteration of the twin-ray method would suffice. Unfortunately, most cross-sections generated were non-manifold. For this reason, it was deemed better to create multiple instances of twin-ray iterations, and use them in a voting process, getting in the end the most-likely result for the outer-border classification of a connected component.

Another choice was the way by which the two rays would be generated. One option was to generate the twin rays in an orderly manner, that of equal intervals. This way, however, would mean that there would be cases that the generated rays would consistently ignore. The other case was to generate the twin rays in random directions. This was opted, because the direction of the rays would have more freedom, and various parts of the cross-section's geometry would be explored. An example of the difference between generating rays in equal intervals, against randomly directed rays, can be seen in Figure 4.6.

In order to take into account the cross-section's geometry in different directions, it was deemed better to make a denser set of twin rays instead of just one, and to generate rays in random directions. This will ensure that the results of border voting will be more consistent, and closer to what is expected.

A final note is that at this point, the generation and execution of the ray voting step is performed repetitively, within a single thread. However, there would be a future upgrade that will make this process faster. One such solution is to make multiple



(a) Rays generated with set angle distance **(b)** Randomly generated twin rays

Figure 4.6: Ordered vs random ray generation

instances —threads— within the algorithm. That way, each thread would process the separate analysis of one twin ray vote. By running multiple threads in parallel, the process can be significantly sped-up.

5 RESULTS & DISCUSSION

Chapter 3 introduced the complete methodology of this thesis, acquiring and pre-processing a 3D input model, transforming the problem from 3D into 2D, working on planar cross-sections, applying twin-ray voting, and extracting the borders by solving an optimization problem. In Chapter 4 were presented the necessary tools and data to realize the methodology, and some choices that were made during the implementation process. This chapter will show some of the experimental results that the pipeline has produced. Afterwards, there will be a small discussion on these results, mentioning key points and important parts in them that have been observed.

At this point, there should be an important disclaimer. As far as the academic world is concerned, the inside-outside classification problem through casting rays that has been mentioned in Section 3.7 has been much explored, with valid and robust results for this having been elaborated as far back as in the middle of the 20^{th} century, by Shimrat [1962]. Therefore, the main goal of this thesis was to robustly produce from any 3D model a 2D shape for inside-outside classification. Consequently, most results will be produced and offered at the point of the pipeline where the step mentioned in Section 3.6 is completed.

5.1 RESULTS

In this section, results from the pipeline will be presented. At this point, a total of over 40 man-made models has been tested, with various shapes, some containing deficiencies, while others are clean. The models were mainly representations of buildings, but in order to test the pipeline's universality, other types were selected as well (cars, furniture, animals, etc.). Out of all of them, the results that are most iconic or that can show the handling of special cases will be presented.

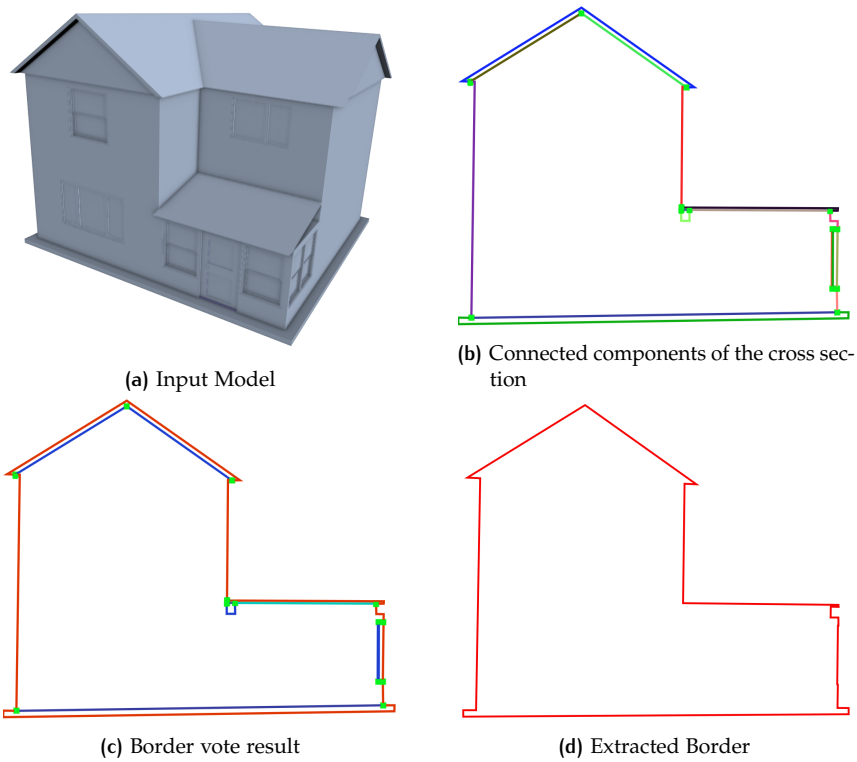
5.1.1 General Results

One first result is the one performed on "Task 14", a 3D model representation of a building. The input in the pipeline and the output produced can be seen in Figure 5.1.

In this figure, the steps from getting the input until the end product are visualized, showing even a part of the intermediate results. In the case of the images, the building used as input is intersected by a plane, and in Figure 5.1b is seen the planar cross section. In that step, the connected components comprising the section have also been constructed. Each component has a unique random color, and the endpoints that are their joints / junctions are visualized as green squared.

In the next step, seen in Figure 5.1c, the twin ray voting method has been applied in the section. The result in this case is that higher values (denoting the outer borders) have been colored with red, and lower values (denoting the interior)tend to blue area of the color spectrum.

The final result of extracted borders through optimization is seen in Figure 5.1d.



(a) Input Model

(b) Connected components of the cross section

(c) Border vote result

(d) Extracted Border

Figure 5.1: Pipeline performed on model "Task 14". (a) View of input model. (b) Constructed connected components from a planar cross-section. Each component has a distinct color. Also, the endpoints are presented as squares (green, because they are junctions). (c) Visual result of the border vote step. Blue are low values, where with red are presented components with higher border confidence. (d) Final extracted border.

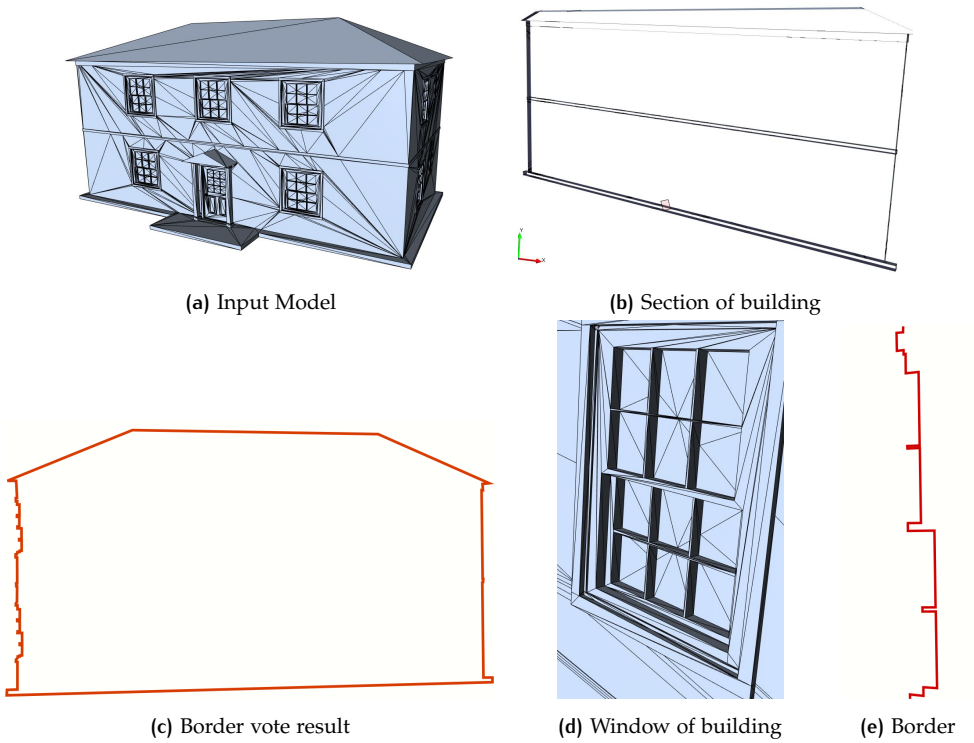
Another result case is shown in Figure 5.2. This one is a 3D building model representation. Between the planar cross section of Figure 5.2b and the extracted borders of Figure 5.2c, it is visible that the interior information of the horizontal ground and first level —as well as the base of the roof part— has been removed.

Special note should be taken for the case of model parts with finer geometry. This can be observed when comparing the result in Figure 5.2e to the original window geometry of Figure 5.2d. The intricate geometry of the window's panels is preserved even after the complex geometric computations of the pipeline.

One additional model that has been tested is that of "Person45", retrieved from Princeton's ModelNet. The input model, along with the results, can be observed in Figure 5.3. In this case, multiple planar cross-sections were simultaneously performed on it (seen in Figure 5.3b). In the intermediate step, the geometry was preserved in all connected components. Finally, the result in Figure 5.3c proves that the pipeline preserves all geometric information in a cross-section, even in cases where there are disjoint (not touching each other) elements, such as the hands with the torso in one section, or the legs in another.

Through Figure 5.3, another aspect of the pipeline can be observed. And it is that disjoint components from the same cross-section are preserved on the final outcome. A single example of this can be seen in Figure 5.4, where a cross section of the previous "Person" model is presented.

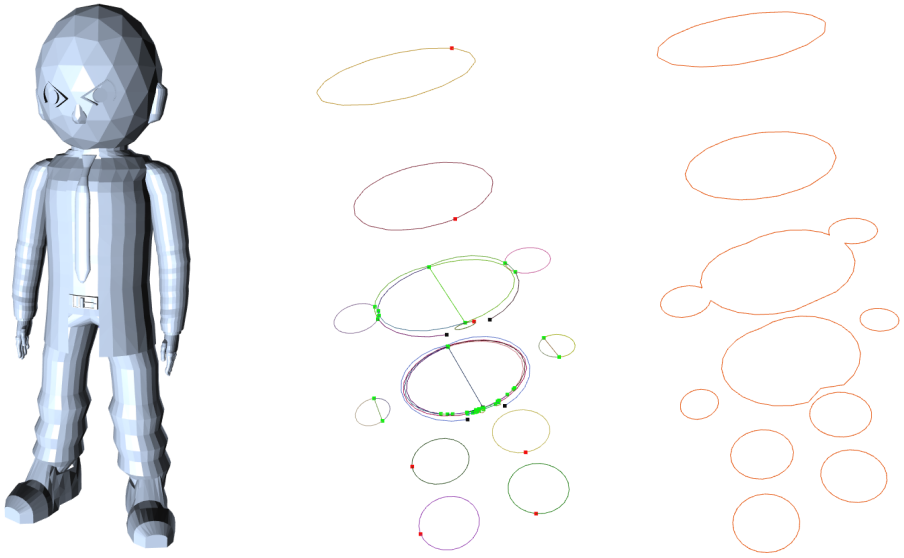
A last general result is shown in Figure 5.5. Initially, the input is given, along with the point in question for the purpose of being classified as being inside or outside. Following, various slices from planes uniquelly oriented are shown, and after them



(a) Input Model (b) Section of building

(c) Border vote result (d) Window of building (e) Border

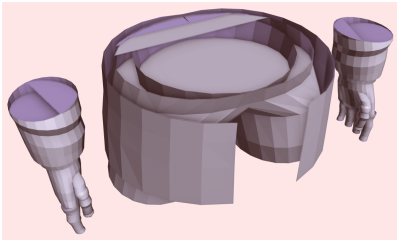
Figure 5.2: Pipeline performed on model "Task 3". (a) Input building representation. (b) Planar slice of the building. The horizontal lines on the interior are the story floors. (c) Extracted outer border from the cross-section. (d) Closer look of the building's window. (e) The extracted border retains the geometric information of the window.



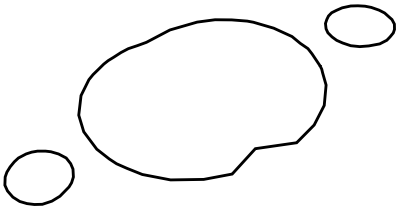
(a) Input model (b) Connected components (c) Border vote result

Figure 5.3: Pipeline performed on model "Person 45". (a) Input model representation. (b) Constructed connected components on multiple cross-sections. (c) Result of extracted outer border for each cross section.

the outcome of the pipeline, performed on the pipeline's particular iteration. In this case, it is proven that the result of classification is robust, regardless of the intersecting planes' orientation.



(a) Part of input model



(b) Outer boundary output

Figure 5.4: Example on disjoint components. (a) Lower torso and hands of the Person model shown in Figure 5.3. The elements are clearly not touching each other. (b) Result of cross section preserves the components and outer boundaries from each disjoint item.

Also, most models that were freely available online were manually made within digital creators/ editors. However, the pipeline can also provide results for models based on data acquired on real structures, through sensing technologies. One such case is shown in Figure 5.6.

5.1.2 Cases with Geometric Deficiencies

The purpose of this thesis is not to just produce results, but also to perform robustly under any circumstances of given input. In this subsection, results will be presented for a model containing many of the geometric deficiencies that have been mentioned in Section 1.1.

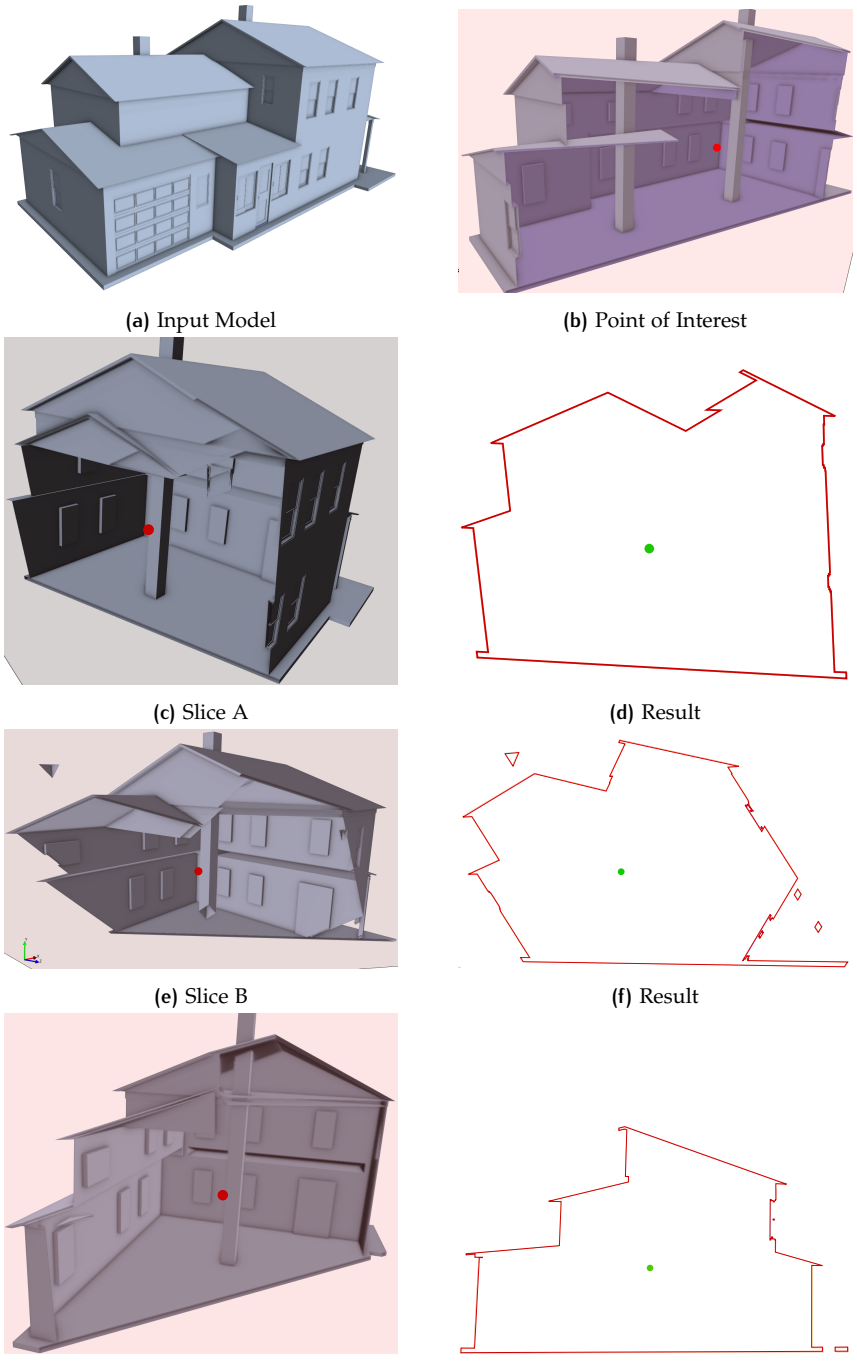
The example results shown here are produced from a 3D building representation (seen in Figure 5.7a). The result of implementing the pipeline to extract the planar cross-section's outer border is seen in Figure 5.7b.

One special case of geometric deficiency is shown in Figure 5.8. This is a part of the model of Figure 5.7a. In the left part of Figure 5.8a, a hole can be seen in the model. This is a part of the roof's base. The red connected component is disjoint from the blue one, both heading towards the left, but not touching. The result of the implementation can be seen in the next image, of Figure 5.8b. With the application of DT, as has been mentioned in Section 3.4.1 (triangulation with all the vertices), the gap has now been filled, and the wall element is meeting the base of the roof.

Another case found here is that of self-intersecting elements inside the 3D model. In Figure 5.9a, it can be observed that the chimney element of the building is crossing through the surface of the roof. The algorithm recognizes it, removes the intersection, and works on all connected components in order to retrieve the ones that are most likely classified as exterior. The end result is the outer boundary of the roof and the chimney, as seen in Figure 5.9b.

Similar results have been observed in other such cases with models containing geometric deficiencies. An extreme case was also tested in the form of a highly deformed model. This case was performing the methodology's pipeline (seen in Figure 5.10) on the "4Bunny" model. There, the original model of the Stanford Bunny was multiplied and merged, to create a complex model with many geometric deficiencies (seen in Figure 5.10a).

In this case, the planar section of Figure 5.10b shows that the model has multiple intersections from the models being overlaid against each other, as well as holes that can be seen at the bottom of the model. Figure 5.10c presents the connected components created within the pipeline, along with their endpoints. The final result (seen in Figure 5.10d) offers the generated outer boundaries of the model. There, it is obvious that the complex interior information has been removed along with all



(a) Input Model (b) Point of Interest

(c) Slice A (d) Result

(e) Slice B (f) Result

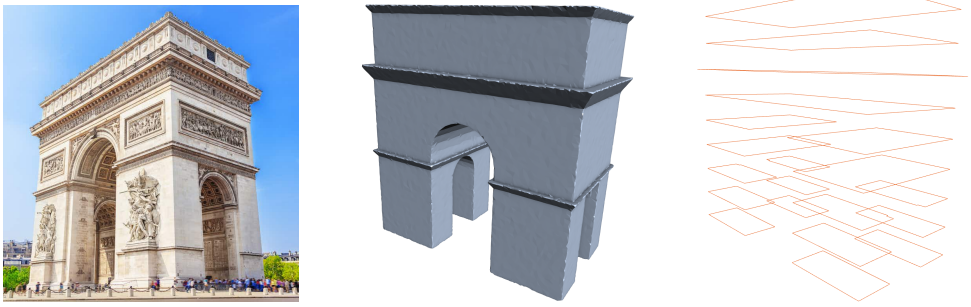
(g) Slice C (h) Result

Figure 5.5: Results on model with various plane orientations. (a) Input building model. (b) Slice with point of interest represented by red dot. (c) through (h) Slices of the model with variously oriented planes, and their results. The point of interest is shown with red on the 3D slices, and with green on the results.

self-intersections. Also, the holes that were located at the lower part of the model have been filled to produce the manifold borders that are observed.

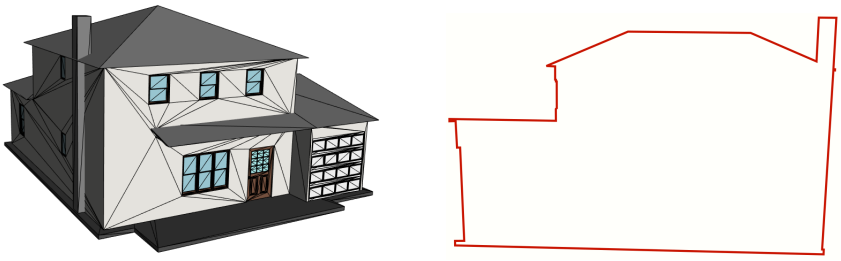
5.1.3 Table of Results

In Table 5.1 a multitude of models tested with the current pipeline are presented, along with their results. Mainly, the pipeline has been tested on building representa-



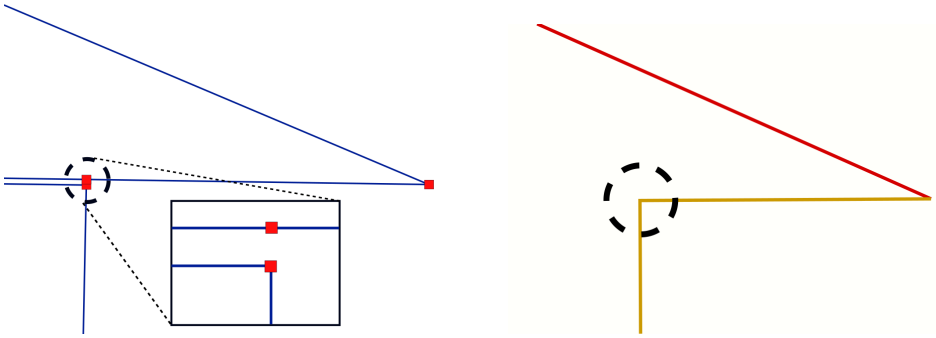
(a) Arc de Triomphe (b) Model of Arc (c) Border Results

Figure 5.6: Test of pipeline on model of real building. (a) Input model representation of the Arc de Triomphe. (Image retrieved from tiqets.com) (b) Surface mesh representation of the Arc. (c) Extracted border on different levels of the Arc's model.



(a) Input Model (b) Extracted Border

Figure 5.7: Result of pipeline on model "Task 20"



(a) Hole in the cross-section (b) Closed hole

Figure 5.8: Filling of gap found in cross section. (a) The part where the two elements should be connected is highlighted. Vertices are represented by red squares. (b) The separate parts are now connected through DT.

tions, but its implementation on other types of models (furniture, animals, persons, cars etc.) has given valid results.

5.2 DISCUSSION

In this section, there will be a small discussion on the results that have been previously presented. Apart from the results, there will also be a small discussion about parts of the pipeline's execution that have been deemed important.

Input	Intermediate	Output

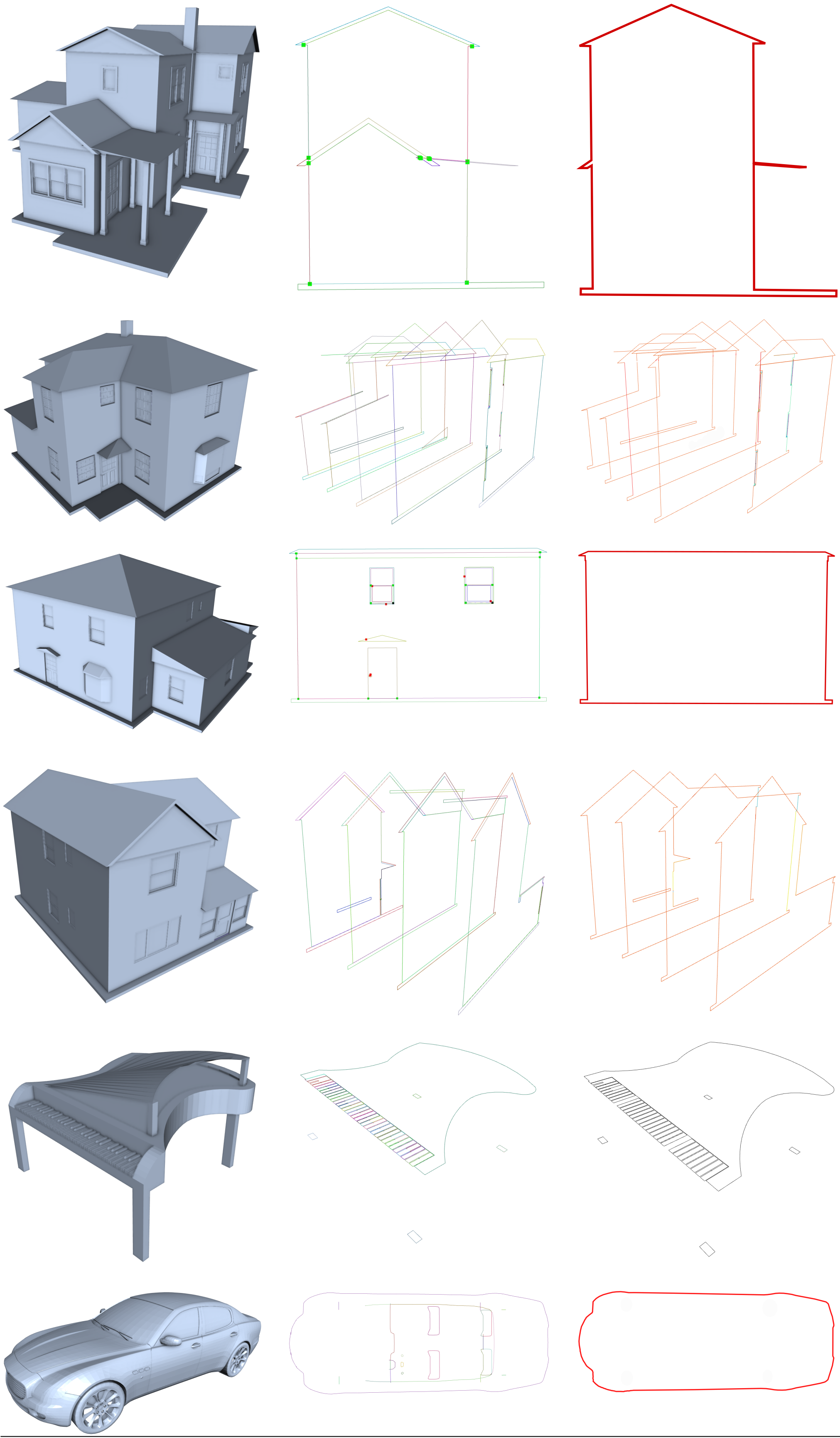
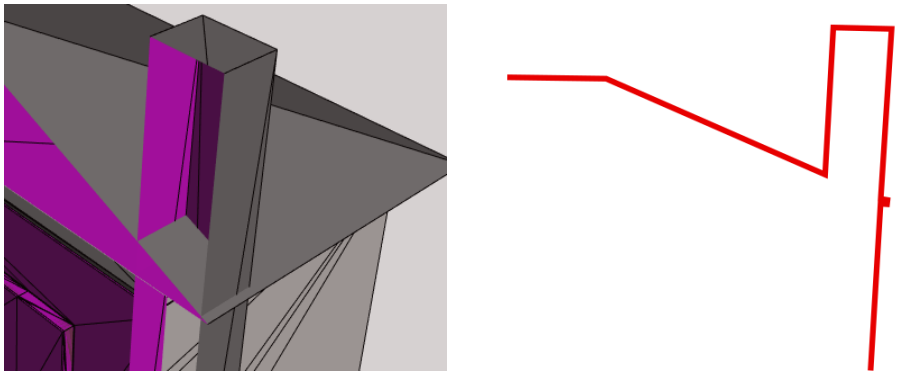
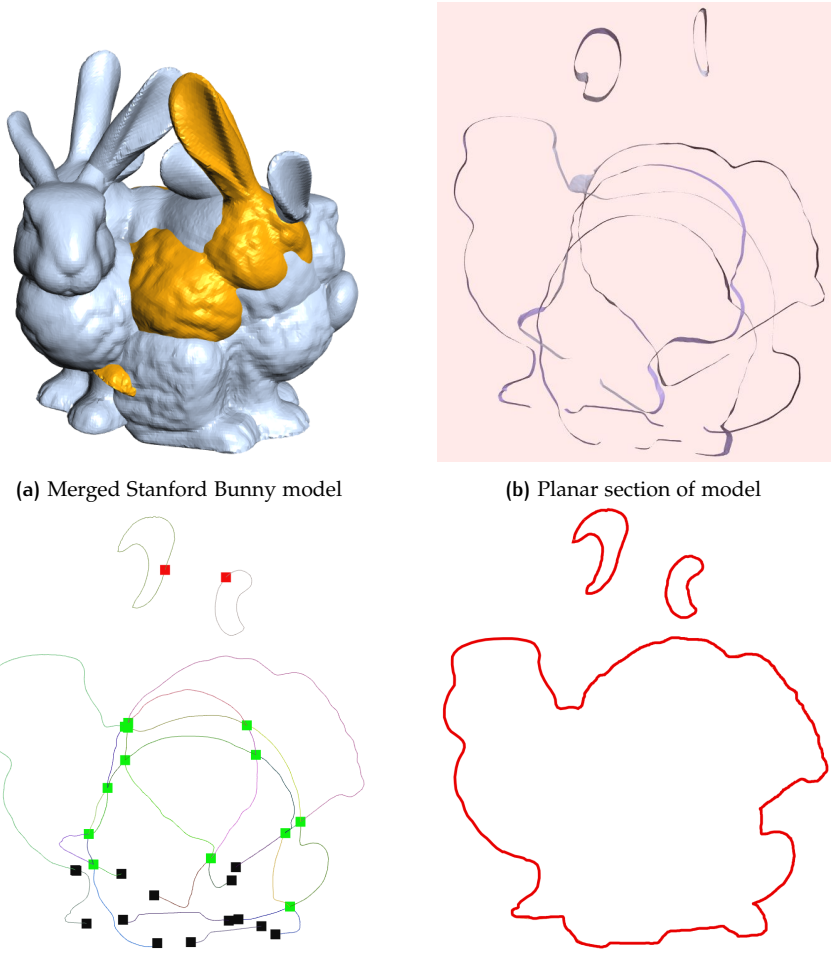


Table 5.1: Table of results. Simpler results have their lines highlighted.



(a) Chimney intersecting the roof **(b)** Extracted border

Figure 5.9: Coping with self-intersecting elements. (a) Zoomed in area from the model presented in Figure 5.7. Chimney passing through the roof. The back/inside of the triangle surfaces is shown in purple. (b) Extracted outline of the model in the particular cross section.



(a) Merged Stanford Bunny model **(b)** Planar section of model

(c) Constructed connected components **(d)** Extracted outer border

Figure 5.10: Pipeline implemented on merged bunny model

5.2.1 General Remarks

In general, the implementation has proven to give results for a plethora of input models. Its main positive points are:

- It can consistently produce results for all given models.

- The interior information is safely discarded, to produce a result of only the cross-section's outer boundary.
- The pipeline extracts the outline of the cross-section, while it preserves the geometry of the initial model.
- Self-intersecting elements are robustly handled, allowing only their parts that are classified as exterior to be extracted. This means that the final product of the pipeline is the extracted outline of a cross-section.
- It has been proven that generally, the pipeline produces unique results, regardless of the intersecting plane's orientation.
- Holes in the input model are dealt with through DT. This, in turn produces a watertight, manifold result of ring-like topology.

5.2.2 Execution Time

Apart from the actual products of the pipeline, it is also important to mention its temporal qualities. The results' input models were tested on a laptop computer with Intel i7-7500U 2.70GHz Dual-Core Processor. Table 5.2 shows the execution time of certain steps in the pipeline. For each step, instead of a clear temporal value, a more general order of magnitude is given, since times may vary depending on the input model.

Step	Time [order of magnitude]
Load Model	0.1∼5 seconds
Create Components	∼1 second
Apply DT	∼0.2 seconds
Twin Ray Voting	4 seconds ∼ 6+ minutes
Optimization	∼0.03 seconds
Outer border extraction	∼ 0.1 seconds

Table 5.2: Execution time of the pipeline's steps

As can be observed in Table 5.2, most steps during the pipeline's execution are efficiently executed, time-wise. The only step that digresses is that of border voting through twin ray creation (explained in Section 3.5). That is because processes such as generating rays and intersections of geometric elements are computationally more expensive than others.

To look a bit more into the matter of this certain step, a benchmark was created to look more into its temporal property. Some results from this can be shown in Table 5.3. It can be generally observed from there that input with bigger geometric data size makes the algorithm take longer to complete.

In order to provide a visualization on these parameters, Figure 5.11 and Figure 5.12 are offered. These contain scatter plots of time against the number of components in the former, and against the number of total line-segments in the latter. In both charts, a trend line is also given, to see how execution time is generally affected by each quantity.

Observing the two charts, it can be seen that indeed an increase in both connected components and line segments provides an increase in the execution time as well. Judging from the chart of Figure 5.11 it could be said that their relationship is exponential, or hyper-linear at least.

	planar cross-section			twin-ray voting
	type	components	segments	time [s]
task 14	building	20	163	30
4 Bunny	animal	30	2638	300
task 3	building	74	572	~360
task 8A	building	17	160	~20
task 8B	building	35	324	120
task 5	building	22	229	20
task 20	building	31	205	42
person 45	human	73	2049	264
task 1	building	22	276	84
task 18	building	26	249	84
task 19	building	37	365	126
task 7	building	58	369	168
task 13	building	61	433	192
car 4	car	37	1199	138
piano 239	furniture	102	728	480
holey cow	animal	10	331	4

Table 5.3: Details of twin-ray method on various models.

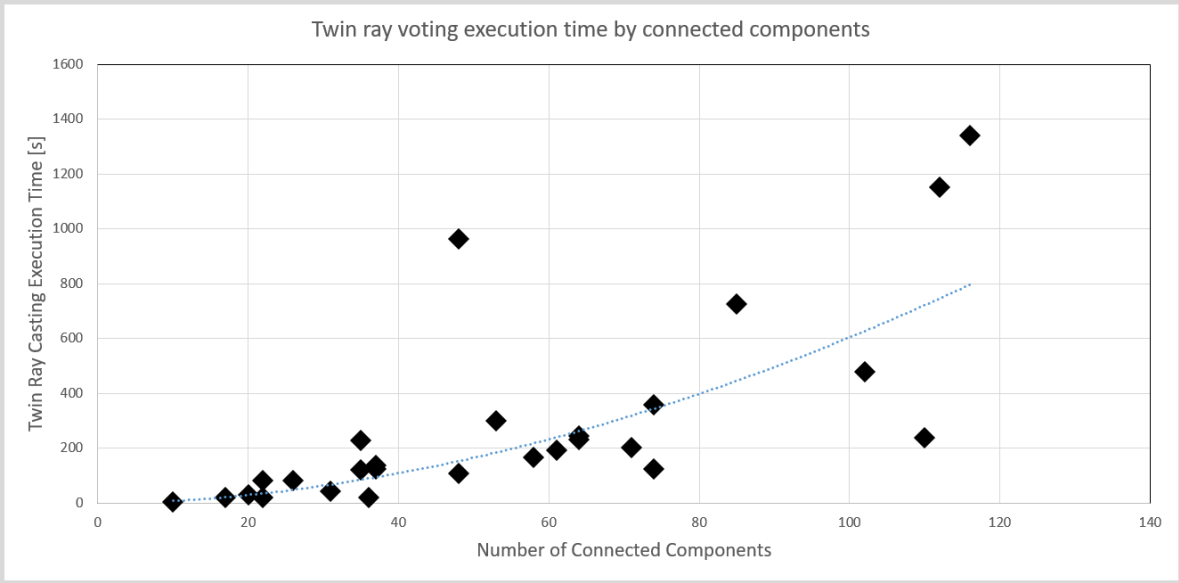


Figure 5.11: Twin ray execution time - Number of components Chart

5.2.3 Optimization vs. Threshold selection

Another thing to take note of is the effect of the optimization process on the produced end result of the pipeline.

The intended outcome of the optimization is having a manifold object with ring-like topology, so that a valid inside-outside classification process can be applied to it (as explained in Section 3.7).

At some point, it was thought that applying a threshold on the border values of the line segments would be sufficient for the extraction of the outer borders. However, there are cases where the selection of the method poses a big difference. For the purpose of those two methods' comparison, the different methods were applied on the model of Figure 5.13. The two different methods provided results that can be seen in Figure 5.14,In the case of Figure 5.14c, the selected components are forming

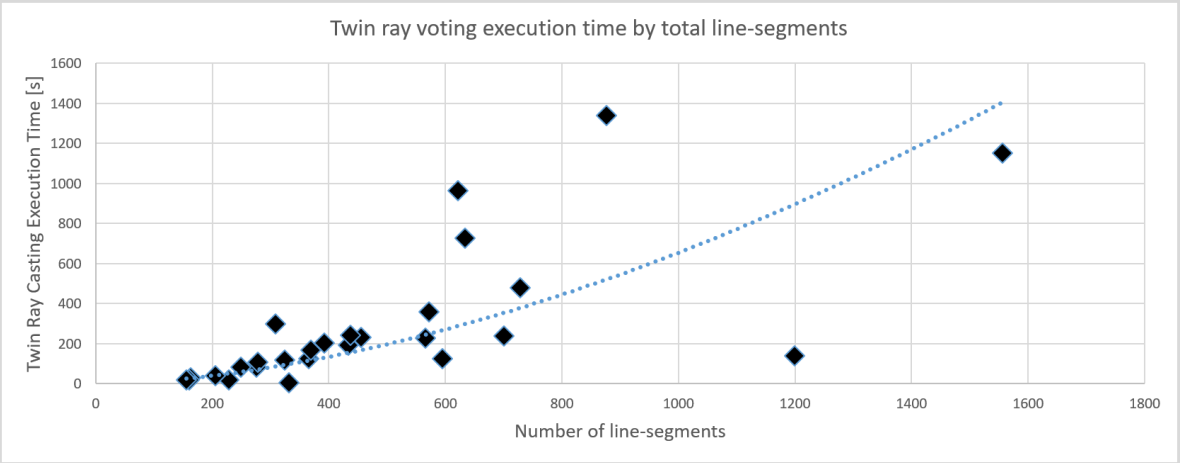
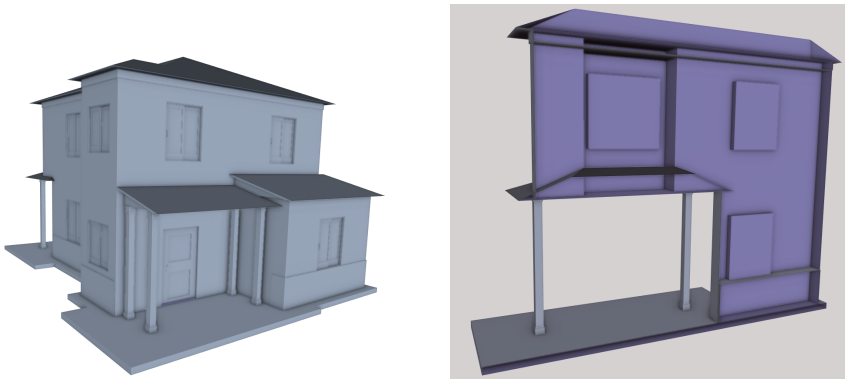


Figure 5.12: Twin ray execution time - Number of section's segments Chart

a ring shape of the building's outer boundary. It can also be seen that the more interior components of the porch-like section of the building have lower border vote values. If only a threshold selection is applied, the result is that seen in Figure 5.14a; a boundary with a big gaping hole.



(a) Input **(b)** Slice of input

Figure 5.13: Input model for border extraction

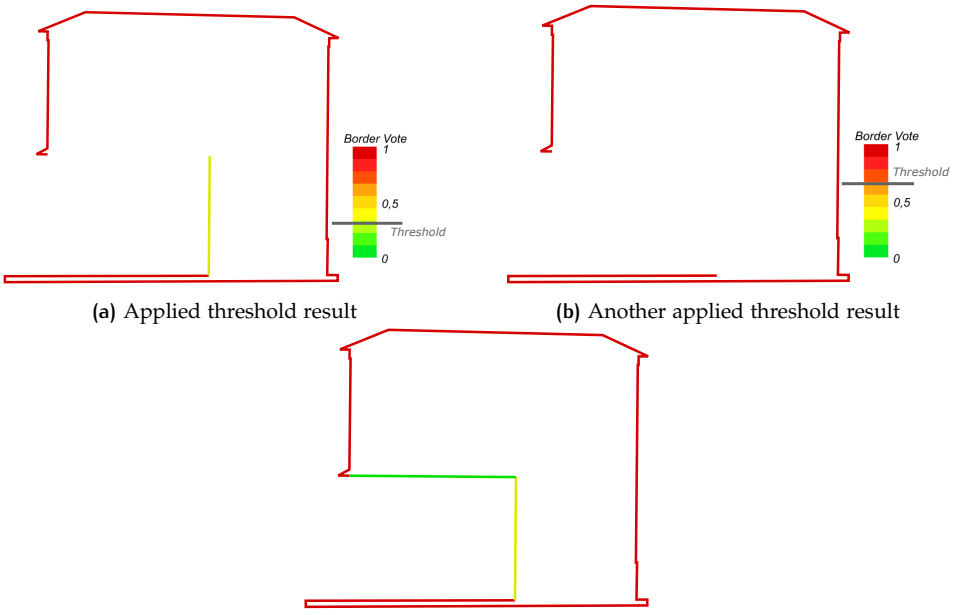
Through the above example comparison, it can be seen why it was important for the optimization to be picked as the outer boundary selection process.

5.2.4 Comparison

The results from this pipeline could easily be compared to other methods that also introduced interior-exterior classification approaches, like the ones of Jacobson et al. [2013] or Nooruddin and Turk [2000].

Quantity-wise, there are not any significant aspects to compare.

The main property that separates this method to the ones of others is that it can handle almost any type of geometric deficiency that the input model may have. For example, this methodology is not affected by the orientation of the model's faces, like in the case of Jacobson et al.. Also, It can handle cases of duplicate geometry of holes, against which Nooruddin and Turk's method has a weakness. A basic comparison of our method against that of generalized winding number (as explained in Section 2.3), can be seen in Figure 5.15.

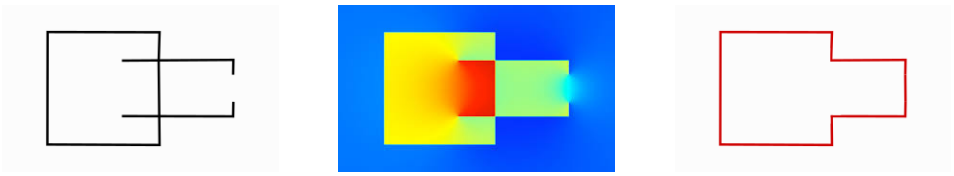


(a) Applied threshold result

(b) Another applied threshold result

(c) Ours

Figure 5.14: Optimization vs. threshold extracted borders. (a) (b) By threshold application, the parts with lower values are removed, leaving a non-manifold outer border. (c) Even with low border values, the components are preserved, to retain watertightness.



(a) Input Geometry

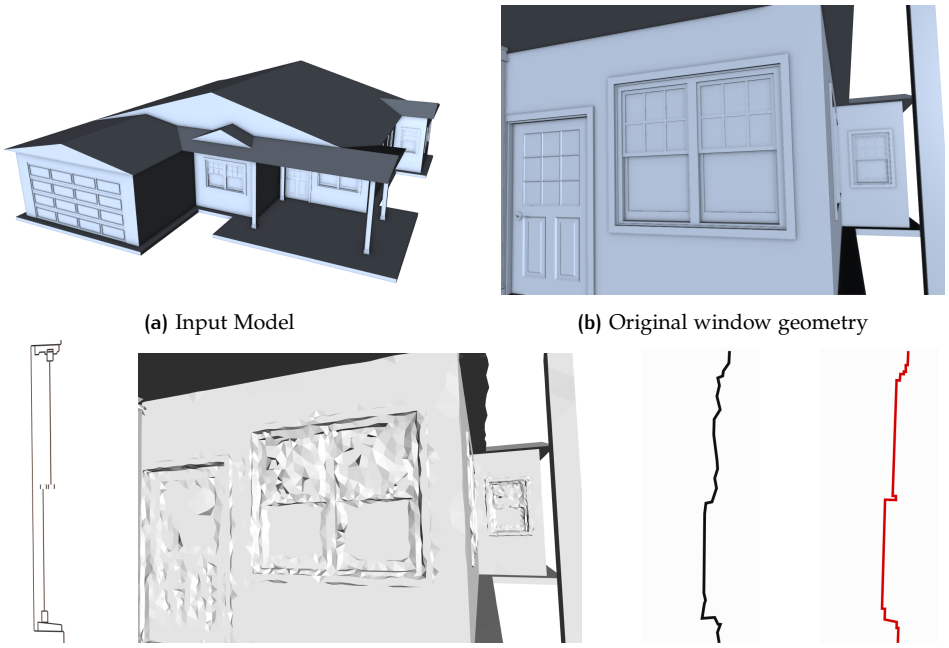
(b) Winding Number Result

(c) Our method's result

Figure 5.15: Comparison of our method against generalized winding number. (a) General input geometry depiction. (b) Winding number result, as has been presented by Jacobson et al. [2013]. (c) Outcome from our methodology.

Another case is that our method can preserve the original geometry of the input model. One comparison is given in Figure 5.16, where an iteration of remeshing from Hu et al.'s TetWild, with parameters set for execution time comparable to our own method, is placed against our method.

In conclusion, this is a method that tries more to show its uniqueness by its robustness, rather than its high-end performance. In what it tries to do, it achieves to obtain a distinction from other existing approaches.



(a) Input Model **(b)** Original window geometry

(c) Slice **(d)** TetWild result **(e)** TW **(f)** Ours

Figure 5.16: Comparing our method to Hu et al.'s TetWild. (a) Example input model. (b) View of original window geometry. (c) Side view of original window geometry. (d) Mesh result of one TetWild execution. (e) Side view of TetWild's resulting window geometry. (f) Our method's resulting geometry.

6 CONCLUSIONS & FUTURE WORK

The purpose of this thesis was to produce a solid pipeline that can robustly perform interior-exterior classification on 3D models, regardless of whether geometric deficiencies exist within the model. In this chapter, Section 6.1 will present some conclusions on the whole attempt. The section is initiated with Section 6.1.1, where answers will be given for the research question (along with the side questions) posed in Section 1.2. Following that, Section 6.1.2 states the contribution of this work towards the scientific field. The section ends with a look back upon the whole process of the thesis, in Section 6.1.4. Finally, recommendations for improvement on the pipeline and using it to realize future applications are given in Section 6.2 and Section 6.3 respectively.

6.1 CONCLUSIONS

As has been mentioned, this section of the manuscript will function as an overall assessment of the thesis subject. First, answers to the original research questions are given. Then, the contribution of this thesis' results to the academic community will be presented. Finally, there will be a general discussion evaluating the overall process, as well as a reflection on the thesis' procedure.

6.1.1 Research Question

- *What is the methodology that should be followed in order to produce valid results?*

  The entire pipeline has been elaborated in Chapter 3. Furthermore, some finer details of the analysis process and the tools utilized are presented in Chapter 4. As has been explained in those chapters, the steps have been:

  1. Pre-processing the input model with the purpose of removing initial geometric deficiencies.
  2. Apply planar cross section at the area of interest, to retrieve a 2D intersection of the model.
  3. Through graph creation, group the line-segments comprising the cross-section into connected components.
  4. If necessary, perform DT in order to close existing gaps.
  5. Apply a border vote value through twin ray casting implementation.
  6. Perform solution of optimization problem with the purpose of retrieving watertight exterior borders of the cross-section.
  7. Do inside-outside classification through ray casting, as explained by Shimrat [1962].

  An important choice is made to use exact numbers within the computation process (mentioned in Section 4.3.2), to handle the arithmetic accuracy issues posed by many digital applications. That way, the input model's geometry and topology is preserved throughout the pipeline.

- *What should be the structure (geometry, stored format, etc.) of the data to be handled, as well the structural form of the final output?*

  For the execution of this thesis, the data most commonly used were those that stored information of 3D models as surface meshes. Many CAD models were tested, with their format mainly being that of OFF. However, there were cases where the input model was stored in PLY format. Nowadays, there is a plethora of published software that can read various file formats, and convert them into valid files of different format. As such, most generally supported file formats would be viable for this solution.

  One thing that should be noted is that the methodology has been constructed with surface model geometry in mind. As such, at this point, solid geometry (3D representations comprised of tetrahedrons) has not been explored within this thesis' scope.

  Regarding the output geometry, presently the pipeline creates files that represent the extracted borders, the connected components and their endpoints. These files have been stored in a self-created format that saves 3D line-segment geometry (source and target point), together with any assigned value to the segment. However, any valid commercial or open format could be used to store the extracted results, for their further analysis.

- *What advantages and disadvantages does this new method have? Is it a process that can produce valid results?*

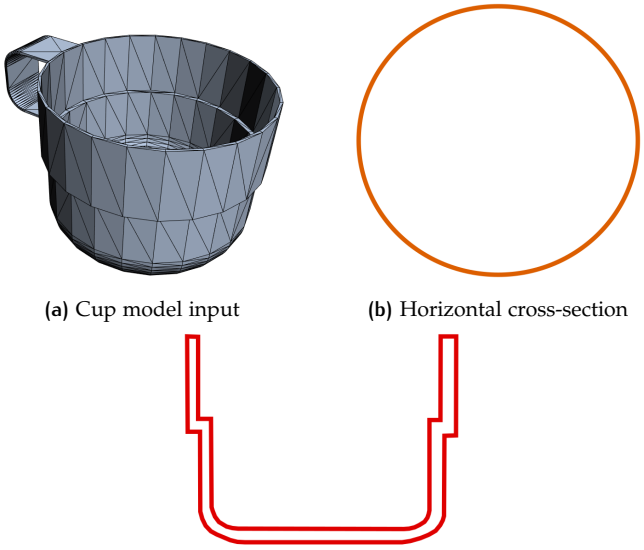
  The advantages and main selling points of this method are:

  1. The pipeline can arbitrarily handle any complex geometries that are given as input to the model.
  2. The outline of the model's borders can be preserved in full detail.
  3. The pipeline is generally robust, regardless of the orientation of the plane that intersects the model, in order to create the cross-section.
  4. All cataloged geometric deficiencies can be dealt with within the pipeline's execution. The deficiencies that the methodology can handle are:
     - Duplicate Geometry
     - Inconsistent face orientation
     - Self-intersecting elements
     - Holes in the surface

  The main disadvantages that this pipeline has are:

  - The user should monitor the pipeline and produce and observe the intermediate results, to make decisions such whether to implement DT on the cross-section, and if so, which option of DT to choose.
  - Although the algorithm is made in a way where it can create multiple parallel cross-sections, as seen in Table 5.1, different cross-sections might require to be handled differently. Working on them as a batch might not give the expected results for all of them. It would be better to work on each one separately, in order to produce the best outcome.
  - After some testing, the pipeline has one limitation. Depending on the geometry of the structure the results might not reflect reality totally. This would happen in cases when the input model is hollow, or has "cup" geometry. In the former case, the interior boundary of the model will be omitted, however the extracted outer boundary is still as intended.

The case of "cup" geometry refers to shapes where one cross-section will wrongly classify a part as interior, while in another it is clearly seen as exterior. This can be seen in Figure 6.1. Depending on the orientation of the plane initiating the cross-section, the results would potentially be different. This can be better understood when comparing the images of Figure 6.1b and Figure 6.1c.



(a) Cup model input **(b)** Horizontal cross-section

(c) Vertical cross-section

Figure 6.1: Limitation against "cup" geometries. (a) Input of a regular cup model. (b) Result of horizontal cross-section, both the solid and the hollow part of the cup will be classified as interior. (c) Result on vertical slice of the cup. The solid part can be correctly classified as interior, while the hollow part as exterior.

- *How does this method compare to other existing approaches?*

  This method does indeed produce valid results. Various approaches by others — that have been mentioned in Chapter 2 — have indeed produced correct and validated results as well. Performance wise, other methods could fare better, but time and quality-wise, this method's results are also evaluated as being of good quality. In some cases, this method proves faster, as analysis in two dimensions is faster than in three. For example, Hu et al.'s tetrahedral meshing algorithm is indeed robust, but may take the best part of an hour, or even more, to produce a result, based on the user's selected parameters. In the case of this thesis, pre-processing, along with the total execution, could be completed in a couple of minutes.

  Where this method really sets itself apart from others is that it does not have any requirements for the input model. As has been stated already, it works robustly for any presented deficiencies in the model, where other methods may not.

The culmination of the answers to the above sub-questions is the final answer to the thesis' main research question:

- *How can a point be robustly classified as lying in the interior or the exterior of a complex polygonal mesh model?*

  A straight answer to this question is by implementing the pipeline presented throughout Chapter 3, with the cornerstone of the methodology being the utilization of planar cross-sections on the input model.

Still, this question poses two very important terms, those of "robustness" in execution, and the handling of "complex" 3D models. Regarding robustness, the pipeline can handle inside-outside classification for a planar cross-section, regardless of the plane's orientation. For this case, complex means that apart from redundant interior information, a model may have all geometric deficiencies listed until now. The methodology handles all cases of these deficiencies as follows:

- **Duplicate geometry**: Detection and removal algorithms that are implemented inside the pre-processing step (mentioned in Section 4.2).
- **Self-Intersections**: Mesh and graph reconstruction (mentioned in Section 3.1 and Section 3.3.1 respectively) handle this case.
- **Inconsistent face orientation**: This problem is bypassed, since face — and consequently in this method, edge— orientation is irrelevant for the execution of the twin ray voting step (mentioned in Section 3.5).
- **Holes**: If and whenever they exist, they can be filled by applying DT on the cross-section (mentioned in Section 3.4). This way, along with the optimization solution (mentioned in Section 3.6), the generation of a manifold boundary is also ensured.

Finally, provided that the generated boundary is now watertight, a given point can be classified as interior or exterior by applying the ray casting method introduced by Shimrat.

6.1.2 Contribution

Apart from providing another solution to the decade-spanning "inside-outside classification" problem, this thesis aspires to offer some other benefits to the scientific community tackling this particular problem.

First of all, this thesis introduces a novel method. Up to the point of this thesis' writing, there has been no published method similar to this. Converting the problem from 3D into 2D simplifies calculations and geometric analysis, without any sacrifices on the final result's quality. It is hoped that with the completion of this project, a message is given that maybe "not all paths have been explored yet", and not just for the interior-exterior classification problem.

In Chapter 1 it has already been mentioned that reasons for the solution to the interior-exterior classification problem, as well as its derivatives, exist in multitude. A lot more reasons than those presented definitely exist. The outcome from this thesis should be a big helping hand for the resolution of some of those problems.

Furthermore, another aspiration is that this method becomes a cornerstone to many more applications in the future. That many researchers would step on the theory behind this to produce solutions to other problems. Or, that the results from this methodology will help in the realization of future 3D model analysis applications. Some recommendations for possible applications using this method will me mentioned in Section 6.3.

6.1.3 Discussion

Using the methodology introduced in this thesis, one can perform interior-exterior classification on a 3D model. The pipeline presented in Chapter 3 can produce a valid result, regardless of any inconsistencies or deficiencies in the input model's structure.

As has been mentioned already multiple times throughout the text, this method sets itself apart from others presented in Chapter 2 through the introduction of planar cross sections. With this, the problem gets converted from three dimensions into two. Many are the benefits of this action. First of all, geometric computations in 2D are far easier to handle, and less computationally expensive than in 3D. Furthermore, results that have been posted in Chapter 5 prove that even the finer details of a model's outline are preserved throughout the whole process, as had originally been claimed. Finally, in three dimensions, elements that would otherwise intersect when in a planar surface may not still intersect, due to the arithmetic accuracy of digital computers (explained in Section 4.3.2).

The original intent of achieving robustness for any form of model or geometric deficiency encountered has been completed. Choices for progressing through the pipeline, presented both in Chapter 3 and Chapter 4, have been made in order to ensure that a valid and geometrically sound end-result is always reached. There could be cases where the user may have expected different results, such as when hollowed areas exist in the model. However, this project was performed mainly as a step in producing valid low-LOD models by extracting their outer borders, to populate city representations with them.

6.1.4 Reflection

This graduation project was initiated on September of 2018 as the final part of the MSc graduation plan for the Geomatics program of TU Delft, and lasted for over 8 months. During this period, the way from the creation of the idea to its realization was not a straight line, but it has been very educative.

For the purpose of the graduation project, this one is more oriented towards an actual research rather than a project that demands a valid product to be provided at the thesis' completion. This has shown that a research project's purpose is not to actually produce results "no matter what", but rather to explore various approaches and solutions to existing problems, and in the end report them regardless of whether the method was a success or a failure.

Indeed, the final form of this research project was not the same from the beginning. From the start, the main idea was to utilize a 2D approach by introducing planar cross-sections. But a number of different approaches was explored until the project ended up in this final state. One such approach that took several months to explore was utilizing the winding number algorithm, in a similar way as Jacobson et al. [2013] did. However, this approach was abandoned when it was found extremely hard to pass the hurdle of models having inconsistent orientation with their faces. In total, this research project did indeed have quite a few "progress through trial-and-error" moments.

One last thing that may seem rather irrelevant but was very important to the project's good and smooth progress was the constant communication with the supervisors. While it may be quite stressful for some people, academic institutes —and even non-academic ones— should encourage a strong communicative bond between project workers and their supervisors, instead of having infrequent meetings to discuss results and nothing else.

6.2 FUTURE WORK & IMPROVEMENTS

In this subsection, some recommendations for future work and improvements on the current state of the project will be provided. These are potential alterations to already existing steps, or expansions to the pipeline's functionality. Their omission

from the existing project was due to temporal limitations within the scope of the graduation project, or because they rose as new ideas in later steps of the project's completion, not being as important as some of the pipeline's main parts, which should be fully functional at the time of completion.

Following are some recommendations that have been specified:

- One first thing to consider is the results generated with hollow objects. Inner boundaries are bypassed in this pipeline. However, a method to also recognize them, and include them in the final extracted borders would prove a valuable extension to this methodology.
- Another welcome addition would be the implementation of multiple planes —each one uniquely oriented— crossing from the same point of interest, and the methodology applied to each of them. This process will yield multiple results, in the likeness of those shown in Figure 6.1. Having multiple interior-exterior classifications, a voting method could be implemented, in order to withdraw one final, universal result.
- Regarding the optimization process, the problem has been structured in such a way that its solution provides the generation of closed loops of connected components. This is also assisted by the step implementing DT in order to fill gaps in the boundaries. However, there would be highly unlikely cases where triangulation might not always consistently fill all holes. When that happens, components that would not form closed loops would be removed through the optimization problem's solution. As such, a method to also be able to retain this open geometry would be welcome.
- In Section 5.2.2, an explanation of the temporal properties for every step of the pipeline has been given. However, there would possibly still be room for speeding up the process. Especially on the "twin ray border voting" step. A method to parallelize the repetitive execution of component voting or ray generation might hasten the process's time. Other ways to quicken the result could also be explored.
- At this point, the execution of the pipeline is highly automated, with just minimal additional input required from the user where necessary. A further improvement would be implementing machine learning on the pipeline. For this purpose, a multitude of input data would have to be fed to an artificial neural network. These data — both the input and output — would be given to the computing system for analysis, in order for it to produce results in the future that are similar to the one it has been fed.
- One final note is that at this point, the pipeline's algorithm only accepts as input models of OFF format. This is a minor inconvenience, as many software exist, capable of converting otherwise formatted files into the particular one. However, making other formats also accepted by the algorithm will lessen some of the workload on the user's end.

6.3 APPLICATIONS

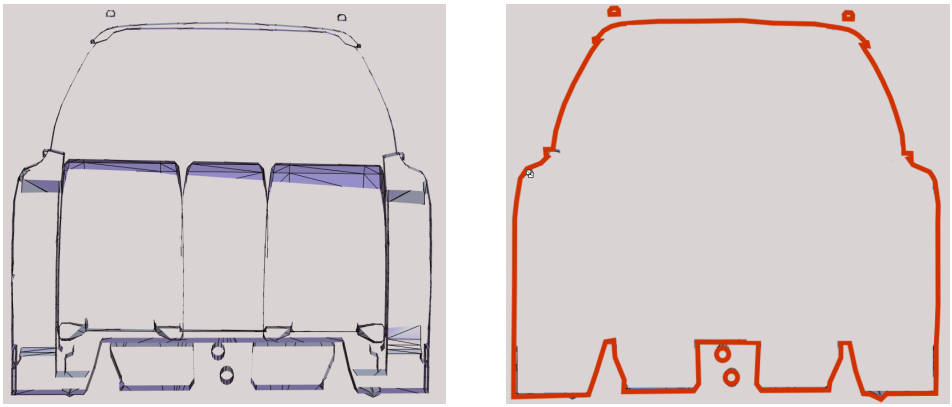
In this graduation project, the notion of interior-exterior classification has been explored, and a solution has been provided. However, this in no way constitutes the end result of research for this field. On the contrary, the results from this research can constitute a stepping stone for various applications that handle 3D models.

Some example applications that would benefit from the results of this research are given below:

6.3.1 Boundary Extraction

An important part of the pipeline is the extraction of the model's outer borders, as mentioned in Section 3.6. This particular function, with some additions, can be extended into its own application.

Having retrieved the exterior borders, it would be possible to overlay them on the original model. That way, any of the 3D model's faces that meet with the extracted borders can be safely classified as exterior. Performing this over multiple, densely created and variously oriented planar cross sections will provide the outer surface of the model as end result. One iteration of this can be seen in Figure 6.2.



(a) Vertical section of a car

(b) Outcome of border extraction

Figure 6.2: Extraction of the exterior borders

Having such an application, it would be possible to automate a process of transforming building models into valid low LOD representations, for purposes such as populating city simulations with building models. Another reason would be just to free up space from models with complex interior information deemed redundant by model analysts or other users. A basic example of such an application can be seen in Figure 6.3, and a more complex one in **??**.

6.3.2 Boundary Visualization

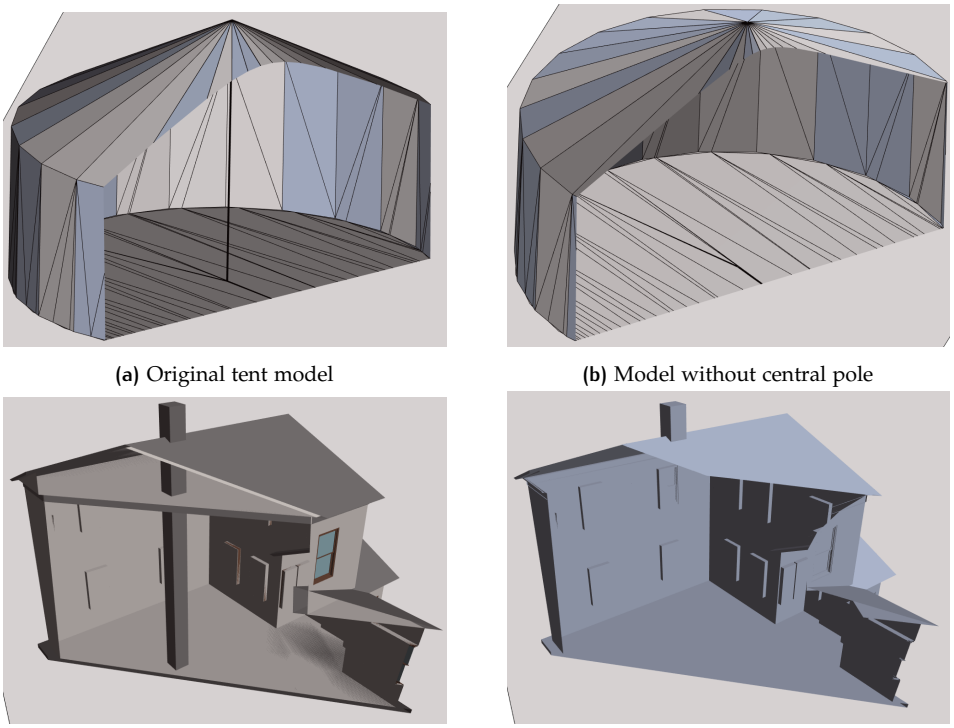
Another application would have the user keep all the faces of the model. In this case, he would preserve the boundary information as a separate property for these faces. Then, it would be possible to apply various visualization techniques on the faces classified as boundaries. For example the exterior surfaces would become transparent, enabling the user to observe both the interior and exterior structure of a given 3D model.

6.3.3 Solid Conversion

A final proposition would be the tetrahedralization of the model's extracted border.

On occasion, having a solid model instead of a shell works more realistically, Jacobson et al. mentions. Solids represent reality better, especially when handled through volumetric processes. For that reason, populating the extracted empty outer shell of an input model with tetrahedra might allow the exploration of further 3D representation applications.

Software performing tetrahedralization already exist. However, utilizing this project's extracted, watertight results will remove cases were the algorithm sometimes inaccurately fills the model's holes.



(a) Original tent model

(b) Model without central pole

(c) Original building model

(d) Remaining shell of building

Figure 6.3: Application of outer surface extraction on tent model. (a) Interior view of the tent model. (b) Removed central supporting pole through outer surface extraction. (c) Interior view of the building model. (d) Removed horizontal floor surfaces and chimney element through outer surface extraction.

BIBLIOGRAPHY

Benner, J., Geiger, A., and Leinemann, K. (2005). Flexible generation of semantic 3d building models. In *Proceedings of the 1st international workshop on next generation 3D city models, Bonn, Germany*, pages 21–22.

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

Boeters, R., Arroyo Ohori, K., Biljecki, F., and Zlatanova, S. (2015). Automatically enhancing citygml lod2 models with a corresponding indoor geometry. *International Journal of Geographical Information Science*, 29(12):2248–2268.

Bondy, J. A., Murty, U. S. R., et al. (1976). *Graph theory with applications*, volume 290. Citeseer.

Bradley, S. P., Hax, A. C., and Magnanti, T. L. (1977). Applied mathematical programming.

Campen, M., Attene, M., and Kobbelt, L. (2012). A practical guide to polygon mesh repairing. In *Eurographics (Tutorials)*.

Caumon, G., Collon-Drouaillet, P., De Veslud, C. L. C., Viseur, S., and Sausse, J. (2009). Surface-based 3d modeling of geological structures. *Mathematical Geosciences*, 41(8):927–945.

CGAL (2018). Computational geometry algorithms library.

Deng, Y., Cheng, J., and Anumba, C. (2016). Mapping between bim and 3d gis in different levels of detail using schema mediation and instance comparison. *Automation in Construction*, 67:1–21.

Donkers, S., Ledoux, H., Zhao, J., and Stoter, J. (2016). Automatic conversion of ifc datasets to geometrically and semantically correct citygml lod3 buildings. *Transactions in GIS*, 20(4):547–569.

Guo, J., Ding, F., Jia, X., and Yan, D.-M. (2019). Automatic and high-quality surface mesh generation for cad models. *Computer-Aided Design*, 109:49–59.

Hu, Y., Zhou, Q., Gao, X., Jacobson, A., Zorin, D., and Panozzo, D. (2018). Tetrahedral meshing in the wild. *ACM Transactions on Graphics (TOG)*, 37(4):60.

Huk, T. (2006). Who benefits from learning with 3d models? the case of spatial ability. *Journal of computer assisted learning*, 22(6):392–404.

Jacobson, A., Kavan, L., and Sorkine-Hornung, O. (2013). Robust inside-outside segmentation using generalized winding numbers. *ACM Transactions on Graphics (TOG)*, 32(4):33.

Musialski, P., Wonka, P., Aliaga, D., Wimmer, M., Van Gool, L., and Purgathofer, W. (2013). A survey of urban reconstruction. In *Computer graphics forum*, volume 32, pages 146–177. Wiley Online Library.

Nagel, C., Stadler, A., and Kolbe, T. (2009). Conceptual requirements for the automatic reconstruction of building information models from uninterpreted 3d models. In *Proceedings of the Academic Track of the Geoweb 2009-3D Cityscapes Conference in Vancouver, Canada, 27-31 July 2009*.

Nan, L. (2018). Inside-outside classification for polygonal meshes. TU Delft Geomatics MSc topics presentations.

Nooruddin, F. S. and Turk, G. (2000). Interior/exterior classification of polygonal models. In *Proceedings of the conference on Visualization'oo*, pages 415–422. IEEE Computer Society Press.

Ohori, K. A., Ledoux, H., and Meijers, M. (2012). Validation and automatic repair of planar partitions using a constrained triangulation. *Photogrammetrie-Fernerkundung-Geoinformation*, 2012(5):613–630.

Rusinkiewicz, S., Hall-Holt, O., and Levoy, M. (2002). Real-time 3d model acquisition. *ACM Transactions on Graphics (TOG)*, 21(3):438–446.

Sacht, L., Jacobson, A., Panozzo, D., Schüller, C., and Sorkine-Hornung, O. (2013). Consistent volumetric discretizations inside self-intersecting surfaces. In *Proceedings of the Eleventh Eurographics/ACMSIGGRAPH Symposium on Geometry Processing*, pages 147–156. Eurographics Association.

Shimrat, M. (1962). Algorithm 112: position of point relative to polygon. *Communications of the ACM*, 5(8):434.

Sindram, M., Machl, T., Steuer, H., Pültz, M., and Kolbe, T. (2016). Voluminator 2.0–speeding up the approximation of the volume of defective 3d building models. *ISPRS annals of photogrammetry, remote sensing and spatial information sciences*, 3:29–36.

Sondermann, H. (2018). Rhino3d: Basic house model.

Sunday, D. (2012). Intersections of lines and planes.

Takayama, K., Jacobson, A., Kavan, L., and Sorkine-Hornung, O. (2014a). Consistently orienting facets in polygon meshes by minimizing the dirichlet energy of generalized winding numbers. *arXiv preprint arXiv:1406.5431*.

Takayama, K., Jacobson, A., Kavan, L., and Sorkine-Hornung, O. (2014b). A simple method for correcting facet orientations in polygon meshes based on ray casting. *Journal of Computer Graphics Techniques*, 3(4):53.

Xu, H. and Barbič, J. (2014). Signed distance fields for polygon soup meshes. In *Proceedings of Graphics Interface 2014*, pages 35–41. Canadian Information Processing Society.

A ALGORITHMS

Algorithm A.1: Planar Intersection ($\mathcal{M}$, P, f)

Input: A surface mesh $\mathcal{M}$, having faces f and edges e, and a plane P
Output: C: the produced cross-section containing line-segments s and their vertices v

```
1  for each f do
2      if P intersects f then
3          s ← get line segment intersection result;
4          for each edge e of f do
5              if P intersects e then
6                  v ← get vertex (point) result of intersection;
7                  assign origin information e to v;
8              assign vertices to s;
9      add s to C
10 return C
```

Algorithm A.2: Component Creation ($\mathcal{G}$, e, v)

Input: A grarph $\mathcal{G}$, comprised of vertices v and edges e
Output: $\mathcal{CC}$: a list containing connected components c_i

```
for each v of G do
    degree_v ← number of adjacent edges to v ;
    if degree_v = 2 then
        // v is connection, should add together adjacent edges e
        e_1, e_2 ← the adjacent edges of v;
        if CC is empty then
            add e_1, e_2 to component c;
            add c to CC;
        else
            for c_i in CC do
                if e_1 or e_2 in c_i then
                    c_1, c_2 ← c_i;
                    // c_1 component that e_1 is already inside of, same for c_2
                       and e_2
            if e_1 inside AND e_2 not inside then
                add e_2 to c_1;
            else if e_1 not inside AND e_2 inside then
                add e_1 to c_2;
            else if (e_1 AND e_2 inside) AND (c_1 != c_2) then
                join c_1 and c_2;
            else if e_1 AND e_2 not inside then
                add e_1, e_2 to component c;
                add c to CC;
    else if degree_v > 2 then
        // v is junction, check edges separately
        for each adjacent edge e do
            if CC is empty then
                add e to component c;
                add c to CC;
            else
                if e not inside any c of CC then
                    add e to component c;
                    add c to CC;
return CC
```

Algorithm A.3: Twin Ray Voting ($\mathcal{CC}$, $\mathcal{SS}$)

Input: all the connected components $\mathcal{CC}$, the total of the cross-section's segments $\mathcal{SS}$
Output: all components of $\mathcal{CC}$ enriched with their border votes *bv*

```
for each component c in CC do
    b_c the border vote of c;
    for each segment s of c do
        cent_s ← centroid of s;
        num_r ← number of twin rays to be generated;
        v_s ← the vote for the segment;
        for twin rays from 1 to num_r do
            r_1 ← ray generated from cent_s and random direction ;
            r_2 ← opposite ray of r_1;
            intscts_1, intscts_2 ← the number of intersections for rays r_1 and r_2;
            for each segment s in SS do
                if r_1 intersects s then
                    intscts_1 += 1;
                if r_2 intersects s then
                    intscts_2 += 1;
            if ((intscts_1 = 0) AND (intscts_2 mod 2 = 1)) OR ((intscts_1
             mod 2 = 1) AND (intscts_2 = 0)) then
                v_s += 2.5 ;
            else if (intscts_1 = 0) OR (intscts_2 = 0) then
                v_s += 1.0 ;
            else if ((intscts_1 mod 2 = 0) AND (intscts_2 mod 2 = 1)) OR
             ((intscts_1 mod 2 = 1) AND (intscts_2 mod 2 = 0)) then
                v_s += 0.5 ;
        b_c += (v_s / num_r) × (length_segment / length_component);
    update CC with new info of c ;
return CC
```

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.