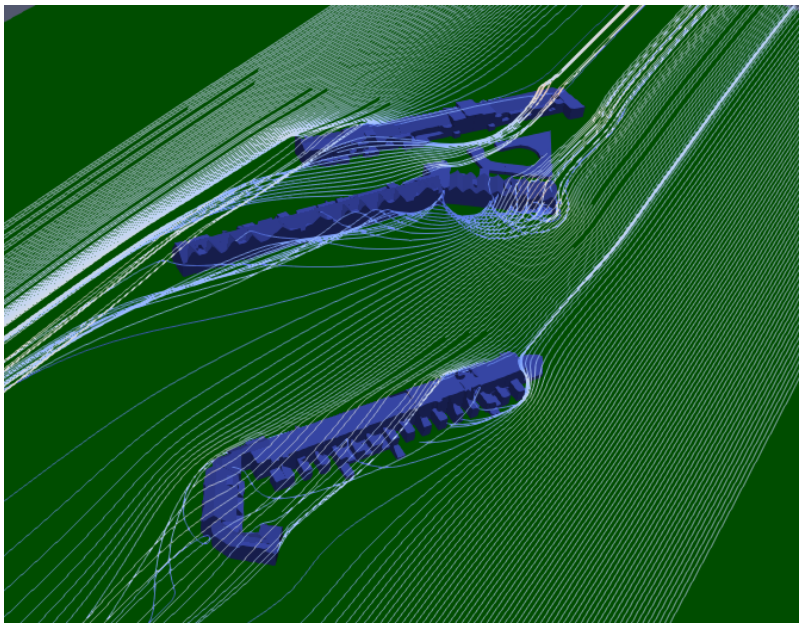
Geomatics Synthesis Project 2022

Removing shared faces in 3D datasets for numerical simulations

November 9, 2022



Authors
Adele Therias
Chrysanthi Papadimitriou
Eleni Theodoridou
Fabian Visser
Fengyan Zhang
Ioanna Panagiotidou

TU Delft

Acknowledgements
We would like to thank Stelios Vitalis, Dr. Ken Arroyo Ohori, Dr. Liangliang Nan and Dr. Hugo Ledoux for their generous support during this project.

Contents

List of Figures

List of Tables

Abbreviations

BAG	*Basisregistratie Adressen en Gebouwen* = Dutch National register for Addresses and Buildings
3D BAG	Dataset containing 3D building models for the Netherlands
LoD	Level of Detail
CFD	Computational Fluid Dynamics
RANSAC	Random sample consensus

1 Introduction

Currently more than 4 billion people live in urban areas around the globe, a trend that is expected to be increased in the upcoming years. While urbanisation provides the space for innovation and new opportunities, in the meantime physical, technical and social challenges are rising and the cities' vulnerability is increasing. A tool to tackle these issues are Computational Fluid Dynamics (CFD) simulations, which can provide insight in various topics.

CFD simulations are valuable for modelling complex urban phenomena such as wind flow, micro-climates and thermal comfort. A CFD requires as an input a 3D geometric dataset that represents objects in the urban environment which are most commonly buildings and then according to this input the air flow is simulated around it.

When creating geometries automatically for CFD simulations, several clean up tasks must be completed for them to be usable without any issues. One of the problems arising is related to the redundant faces shared between adjacent buildings, which have no purpose for outdoor flow simulations and cause complications when creating the mesh that is needed for the CFD. This synthesis project focuses on addressing the aforementioned issue by removing the shared faces.

The ultimate goal of this project was to create an open-source product that can efficiently and in an automated way remove the adjacent faces between buildings. The benefits will be imminent during the meshing process, as we strive to reduce the time that consultancies spend fixing the input geometries before running a CFD simulation, along with an overall improved user experience.

This report is organised in four main sections. The first section is the general introduction of the issue that needs to resolved. The second section defines more in depth the problem and sets the research questions, in accordance to that, in the third section the research methodology is developed. In the fourth section the results of both methods are presented. The fifth section focuses on a reflection of the project, while the sixth section presents the final conclusions. Finally, the seventh section contains the specifics of the project management itself.

The project was carried out in cooperation with Dassault Systèmes and is developed in the context of the GEO1101 course in MSc Geomatics TU Delft. In addition to this report we have created a GitHub repository (fig. 1) that contains the source code of the two methods.

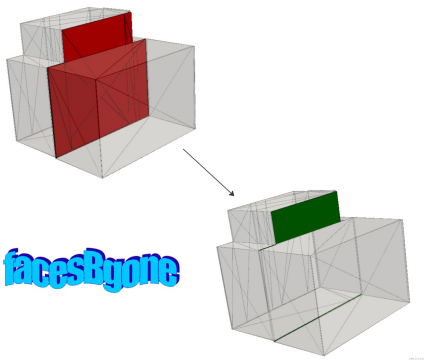


Figure 1: facesBgone at https://github.com/Fabisser/facesBgone

2 Problem Definition

Our approach of the problem stated above was to search and evaluate the available literature and identify ideas and algorithms that could help us tackle it. In the process, two different methods were investigated to remove the internal faces, and they will be described in more detail in the methodology section. Moreover, a comparative study on the advantages and disadvantages between the two methods is outlined in the discussions section.

Our datasets were taken from the 3D BAG, which is an up-to-date data set containing 3D building models of the Netherlands, often containing semantic characteristics. We selected diverse subsets of the 3D BAG with varying building configurations since we aspired for the algorithm to be robust. We then ran validity tests on these datasets before the implementation to guarantee the quality of the input, in terms of geometrical and topological correctness. After running the code, we ran the validity tests again to ensure that we had not compromised the validity. As a final step, we evaluated the results to determine the percentage of change in the geometry using the Hausdorff distance.

The research question is as follows: "In what ways can the shared faces between adjacent buildings be removed from CityJson datasets for CFD applications?"

3 Research Methodology

3.1 Data Selection

For the selection of the datasets, we focused on finding rows of buildings in Delft and Amsterdam that clearly contain adjacent buildings. The datasets were obtained from 3D BAG. 3D BAG contains high quality, detailed, open data that allows for exports in OBJ, CityJSON and GPKG in multiple LoDs. We selected areas with many buildings connected to each other and worked on a test dataset of a block of buildings, in LoD 2.2, with the goal to eventually run the code for a whole tile (fig. 2). Fixing the geometry of 3D BAG was not a task of this project, therefore we used only datasets that passed the validity tests of Val3dity. Val3dity is a tool that "allows us to validate 3D primitives according to the international standard ISO19107" (Ledoux, 2013; 2019).

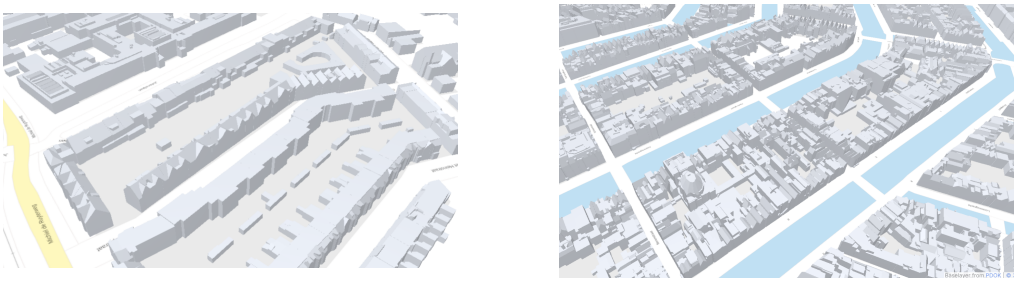


Figure 2: Example datasets of many connected buildings

3.2 Identifying Adjacent Buildings

The dataset used are from 3D BAG, as mention above. The issue that arises from that is that the buildings in 3D BAG are not topologically correct. This means that the adjacent buildings do not share common vertices and the adjacency can not be identified.

To proceed with our solution we first had to identify adjacent buildings. That task was made possible by adapting the 3D Building Metrics code (3DBM) developed by Labetski et al. (2022). According to the 3D Building Metrics script, for each building, other buildings that intersect its bounding box are found: these are considered "candidate" neighbouring buildings. For each candidate building, Agglomerative Clustering is performed between the two buildings to identify plane clusters. If a cluster is found in both buildings, an additional check is carried out to ensure that the shared planes intersect. If these conditions are met, the two buildings are indeed adjacent, and their shared faces should be removed. This process is done in Python using the PyVista Library and one of the outputs is a .txt file containing the blocks of adjacent buildings.

3.3 Removing shared faces

Moving forward to the main task of this project, to remove the shared faces, two methods were developed. Each has its advantages and disadvantages which will be explained in more detail in the following section.

Method 1: Hole Filling

While identifying the adjacencies between buildings (3.2), we also calculate the difference using the PyVista library in Python between the shared faces of each building and its neighbour. This allows us to create new faces that are needed to account for any height or width offsets between the two buildings. Once these new faces are computed we remove the shared faces between two buildings. The result is a collection of meshes of buildings without their internal faces, and the newly created faces calculated from the difference operation. Since these parts are not connected, we create a watertight mesh out of them by filling the gap between them using the hole filling operation in CGAL.

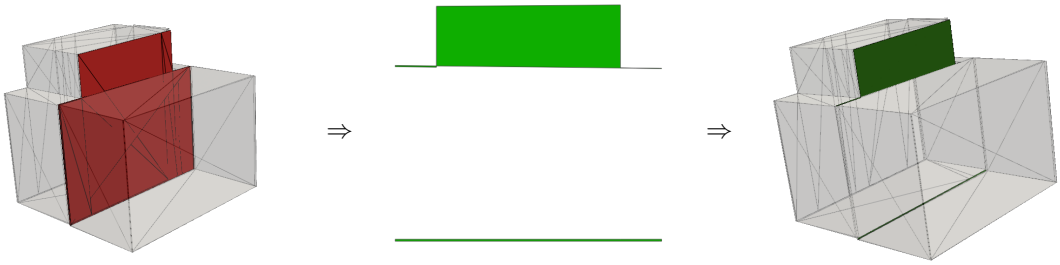


Figure 3: The internal faces (red) are removed and only the non-intersecting faces (green) remain

Method 2: Nef Polyhedra

The second method uses Nef Polyhedra, a CGAL B-rep data structure that is formed by applying boolean operations on halfspaces (Constructive Solid Geometry) (CGAL, n.d.). This method takes as input the CityJSON file storing geometries and the .txt file that contains the blocks of adjacent buildings. Using CGAL 5.5 in C++, we created a Nef Polyhedron from each adjacent building (CGAL, n.d). The next step is to union invidual Nef polyhedra to form one Nef per block. However, considering that CityJSON does not store topology between buildings, there can be small gaps between adjacent buildings. Therefore, we apply a Minkowski Sum to act as a 3D “buffer” that expands Polyhedra sufficiently to remove the gaps between them (Geometric Motion Planning, n.d.). We set the default Minkowski value to 0.003, but this parameter can be modified by the user. The Minkowski sum might not work in some cases (e.g. buildings with relatively complicated structures), then the convex hole of the building is firstly obtained and then the Minkowski sum is applied. Once it has been applied, the individual Nef Polyhedra are merged into one Big Nef Polyhedron per block. This method is expected to return a watertight mesh that follows the original geometry well. The output of the Nef Polyhedra method is an .stl file as requested by the partner company, Dassault Systemes, to be integrated in their workflow.

4 Results

4.1 Evaluation methods

To evaluate the outputs of each method, we used a visual tool to check whether shared faces were fully removed, and a numerical index to quantify the distortion of the original geometry. To check the existence of shared faces, we sliced across the output building dataset to determine which of the shared faces were removed during the process. The test is conducted in ParaView with the tool ”slice”.

To quantify the change in the dataset geometry, we calculated the Hausdorff distance between the original dataset and the output dataset. The Hausdorff distance involves computing the shortest distance from each vertex in one object to each vertex in a second object, then keeping the largest

of these distances (Normand and Bouillot, 1998). The Hausdorff distance between the two meshes is computed in MeshLab with the tool "Hausdorff Distance."

4.2 Method 1: Hole Filling

Based on slices of the resulting figures (fig.4, fig.5, fig.6) that the majority of the internal faces are removed but still some of them remain. The effectiveness of the algorithm depends on the dataset and on some user defined parameters, such as the maximum diameter of the holes to be filled and the maximum number of edges of holes to be filled.

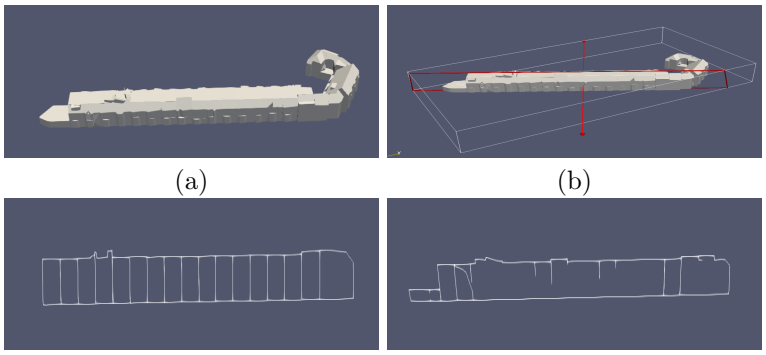


Figure 4: Original dataset 3 (a), slice (b), original internal faces(c), final internal faces (d)

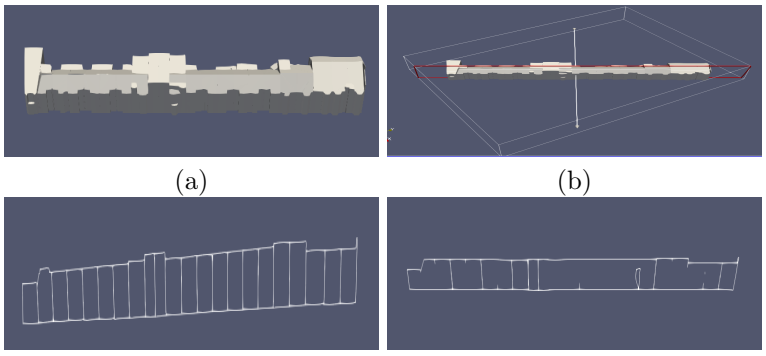


Figure 5: Original dataset 4 (a), slice (b), original internal faces (c), final internal faces (d)

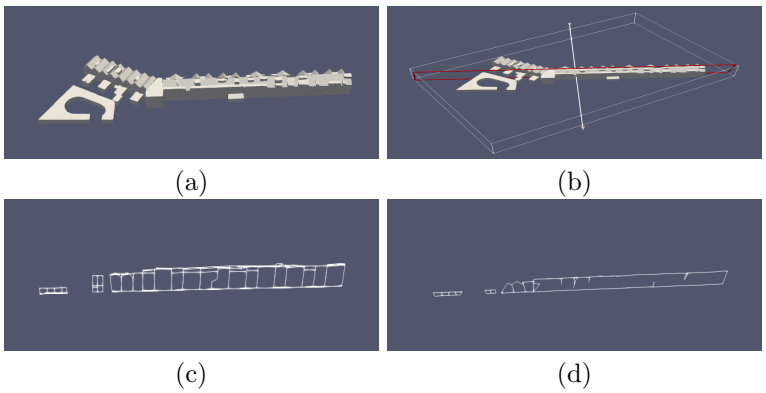


Figure 6: Original dataset 5 (a), slice placement (b), original internal faces(c), final internal faces (d)

In table 1, the results of the Hausdorff distance between the output geometry from method "hole filling" and the original geometry are presented. For both dataset 3 and 4 the mean Hausdorff value is less than 1 meter (0.010 and 0.06 respectively), while the max Hausdorff value is no larger

than 2.6 meters. According to the normalized values, the geometry is not greatly distorted as the mean Hausdorff value is 0.000074 (dataset 3) and 0.00004 (dataset 4).

Dataset	**Max Hausdorff absolute value (m)**	**Mean Hausdorff absolute value (m)**	**RMS Hausdorff absolute value (m)**	**Max Hausdorff normalized value**	**Mean Hausdorff normalized value**	**RMS Hausdorff normalized value**
3	2.65625	0.010435	0.1029	0.018926	0.000074	0.000733
4	1.404513	0.006091	0.063336	0.010203	0.000044	0.00046

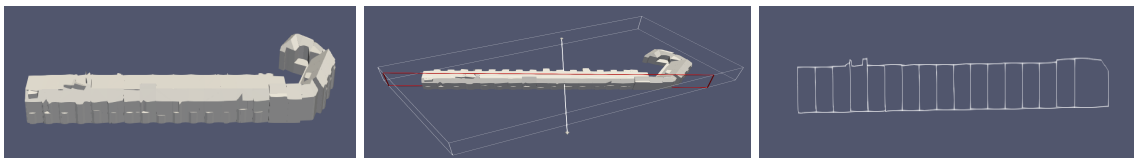
Table 1: Hausdorff distance between original geometry and method output (datasets 3,4)

4.3 Method 2: Nef Polyhedra

4.3.1 Minkowski values

In order to be able to define a functional Minkowski value that could be used in a wide variety of datasets, different values were tested. The optimal Minkowski value is the one that eliminates the internal faces, while having the smallest possible distortion on the final geometry of the buildings.

To determine whether the internal faces were removed or not, we ran the code with different Minkowski values and the output dataset was visualized in ParaView and sliced across with the "slice" function. For the testing, two rows of buildings are used, dataset 3 and dataset 4. The results of these test are presented in the figures (fig.7, fig.9) below. As it can be seen , the internal faces are removed with values 0.003, 0.004, 0.005 and 0.01. For a very small Minkowski value (0.001), the output dataset is smaller than the original, because the 0.001 offset was not enough to actually connect the row of buildings and the majority of buildings are missing from the output (fig.8). In addition, for dataset 4 the Minkowski value of 0.001 is not capable of eliminating all the internal faces (fig.10)



(a) (b) (c)

Figure 7: Original dataset 3 (a), slice placement (b), internal faces (c)

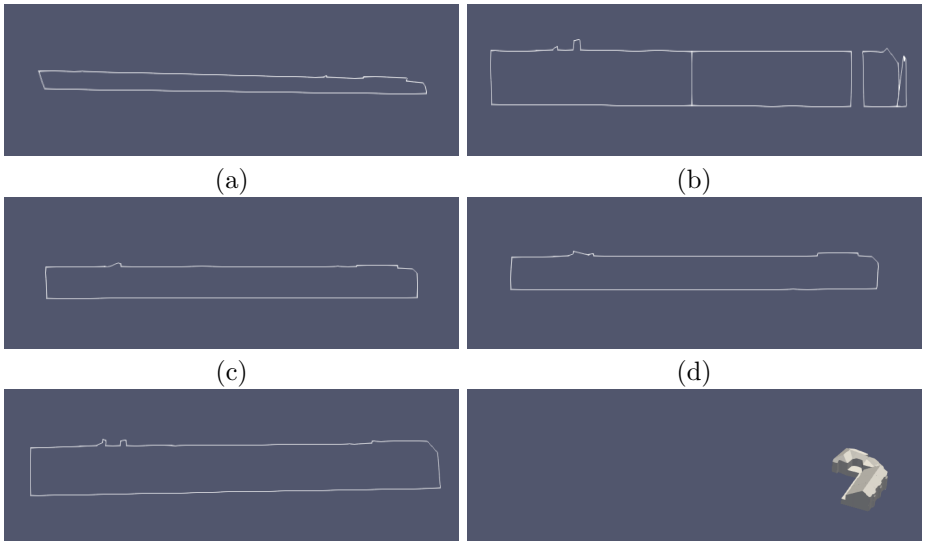


Figure 8: Dataset 3 with Minkwoski value = 0.1 (a) , 0.002 (b) , 0.003 (c), 0.004 (d) , 0.005 (e), 0.001 (f)

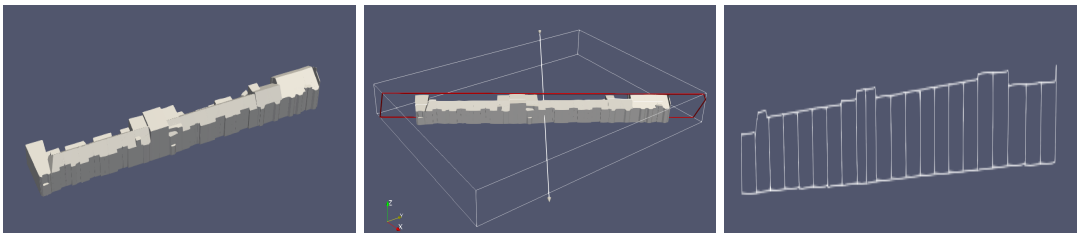


Figure 9: Original dataset 4 (a), slice placement (b), internal faces (c)

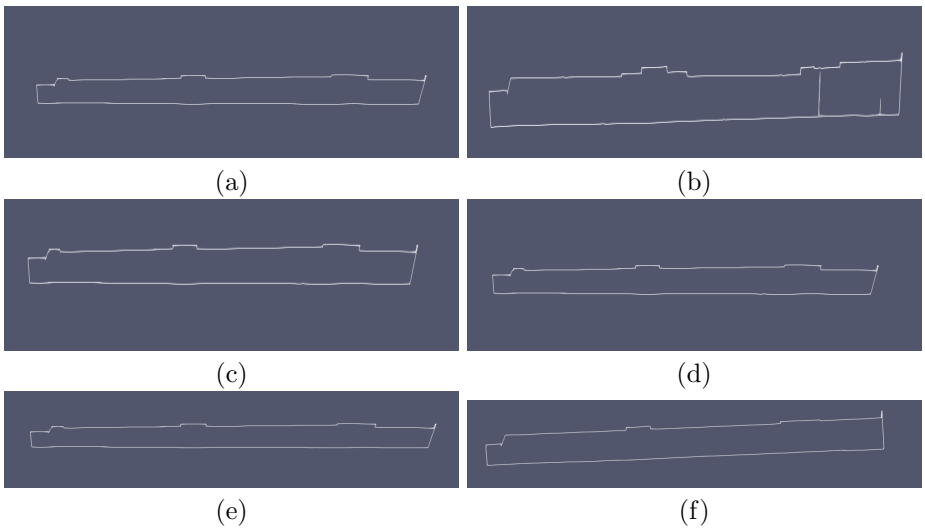
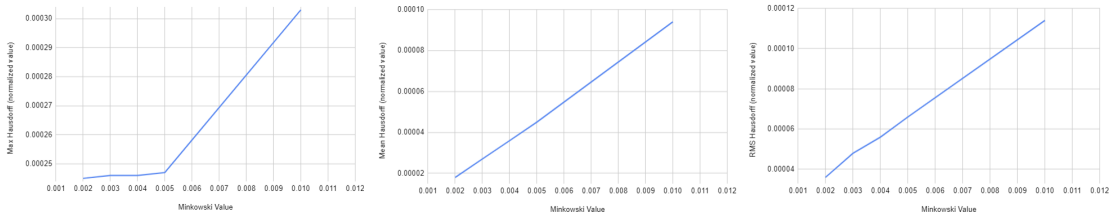


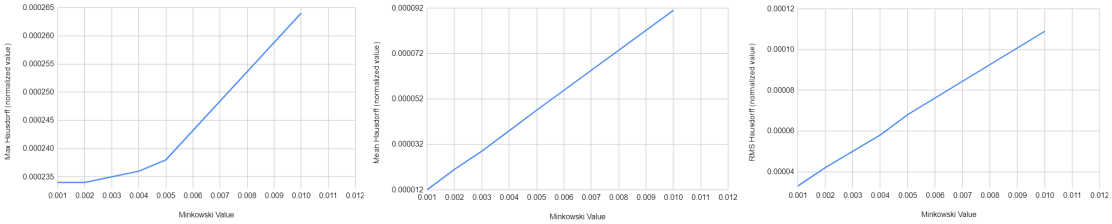
Figure 10: Dataset 4 with Minkowski value = 0.1 (a) , 0.001 (b) , 0.002 (c), 0.003 (d) , 0.004 (e), 0.005 (f)

In addition to the elimination of the internal faces, the minimum distortion of the original geometry is a factor that is taken into consideration, when selecting the optimal Minkowski value. To determine the quality of the final geometry, the Hausdorff distance between the original datasets 3 and 4 and the dataset after the application of a Minkowski value, are computed. As it can be seen by the graphs below, the larger the Minkowski value, the larger the Hausdorff value (fig.11, fig.12).



(a) (b) (c)

Figure 11: Dataset 3 - Max Hausdorff (normalized value) (a), Mean Hausdorff (normalized value) (b), RMS Hausdorff (normalized value) (c)



(a) (b) (c)

Figure 12: Dataset 4 - Max Hausdorff (normalized value) (a), Mean Hausdorff (normalized value) (b), RMS Hausdorff (normalized value) (c)

As a result, from the tested dataset with the different Minkowski values, the value of 0.003 is the one that manages to eliminate all of the internal faces with the minimum max, mean and RMS Hausdorff differences from the original dataset. This Minkowski value, though, might work efficiently for a variety of datasets with similar buildings as the ones tested but it should not be considered universal and it possibly needs to be adjusted for different datasets.

4.4 Accelerating and Processing

We ran both methods for a different number of buildings and calculated the runtime. The results are presented in figure13. Both methods require adjacency of buildings to be calculated. The python script runs for about 1s per building. The hole filling function in CGAL is almost instant, leading to a very short runtime for even a large collection of buildings. Calculating Nef and Minkowski is a different story. This requires around 10s per building. While the runtime is therefore longer than hole filling, we find it is still reasonably acceptable, and it is important to note that the runtime can be further shortened by multi-threading.

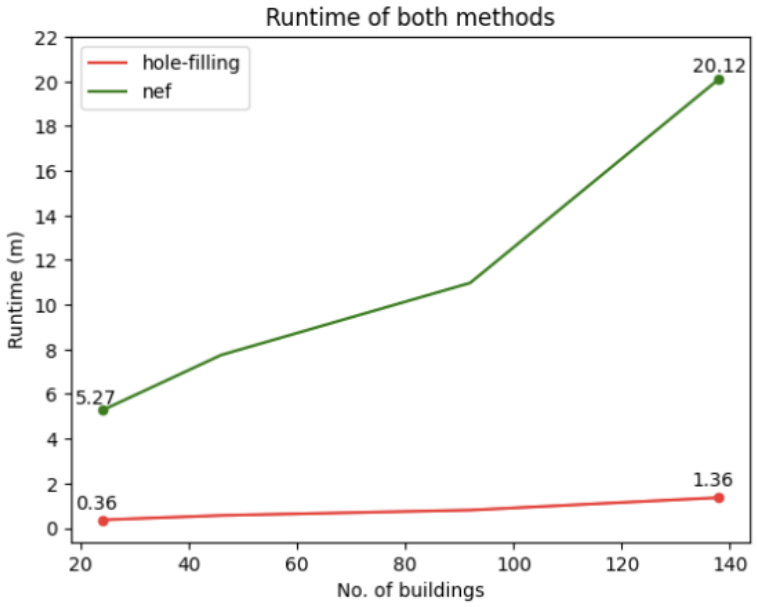


Figure 13: Runtime of both methods

4.5 Comparison Between the Two Methods

4.5.1 Hausdorff Distance

A comparison of the Hausdorff distance that was computed between the output of method "NEF" with the original dataset and the output of method "hole filling" with the original dataset is presented in the following tables. The difference between the two methods appears significant in terms of percentages (e.g. method "hole filling" mean Hausdorff is 97% larger than method "NEF" mean Hausdorff for dataset 3 and 40% for dataset 4) but in terms of absolute distances the difference is not that vital. The mean Hausdorff value for method "hole filling" is only 0.010 m for a whole row of buildings for dataset 3 and for dataset 4 is 0.0060 m. Between the two methods tested methods, method "NEF" gives the smallest geometric distortion in comparison to the original dataset. All in all, the distortion is highly dependent on the dataset itself, but we expect that the overall distortion is within the acceptable threshold for a large scale urban CFD simulation, where multiple blocks of buildings are used as input.

Dataset 3	**Max Hausdorff absolute value (m)**	**Mean Hausdorff absolute value (m)**	**RMS Hausdorff absolute value (m)**	**Max Hausdorff normalized value**	**Mean Hausdorff normalized value**	**RMS Hausdorff normalized value**
Method "NEF"	0.032351	0.003589	0.006303	0.000246	0.000027	0.000048
Method "hole filling"	2.65625	0.010435	0.1029	0.018926	0.000074	0.000733
Absolute difference between the two methods	2.623899	0.006846	0.096597	0.01868	0.000047	0.000685
Difference between the two methods (%)	195.187%	97.6326%	176.913%	194.868%	93.0693%	175.416

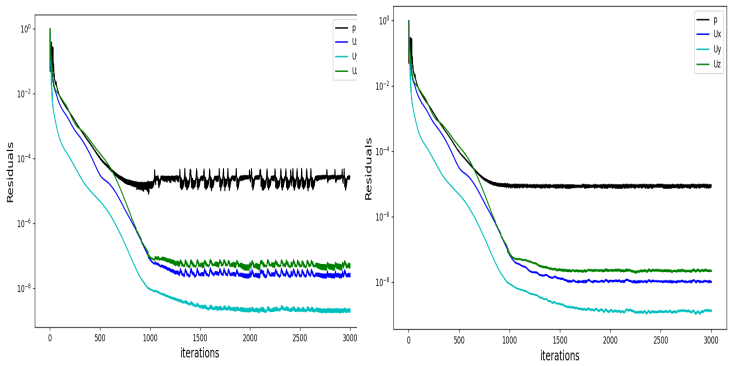
Table 2: Hausdorff distance comparison between the two methods, dataset 3

Dataset 4	Max Hausdorff absolute value (m)	Mean Hausdorff absolute value (m)	RMS Hausdorff absolute value (m)	Max Hausdorff normalized value	Mean Hausdorff normalized value	RMS Hausdorff normalized value
Method "NEF"	0.032379	0.004034	0.006874	0.000235	0.000029	0.00005
Method "hole filling"	1.404513	0.006091	0.063336	0.010203	0.000044	0.00046
Absolute difference between the two methods	1.372134	0.002057	0.056462	0.009968	0.000015	0.00041
Difference between the two methods (%)	190.986%	40.6321%	160.837%	190.994%	41.0959%	160.784

Table 3: Hausdorff distance comparison between the two methods, dataset 4

4.5.2 CFD Simulation

The outputs of the two methods were used for a CFD simulation, which was constructed and run in OpenFoam. In order to check when and if the case converged for each of the methods, the residuals (velocity (Ux, Uy, Uz) and pressure (p)), are plotted (fig.14). As it can be seen from the graphs, for both of the cases, approximately in 1000 iterations the residuals start to stabilize (horizontal line). For method "NEF", there are still fluctuations and especially for parameter "p". On the other hand, for method "hole filling", the residuals appear steadier than method "NEF" for the same iterations. Therefore, regarding residuals, method "hole filling" provides better results.



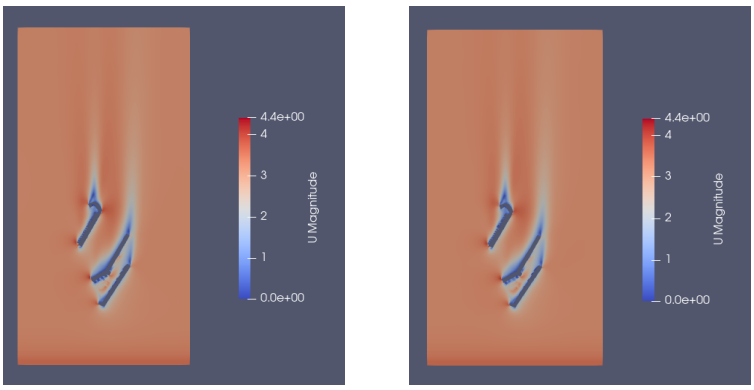
(a) (b)

Figure 14: Residuals from method "NEF" (a), Residuals from method "hole filling" (b)

After visualizing the results of method "NEF" and method "Hole filling" in ParaView, there are no visible differences that indicate diverse results from the two methods. In the figure 15 the velocity (U) result is depicted for both methods. We had similar results for both methods also for parameters epsilon, k, nut, p.

5 Further Development

One problem we have found in our outputs are at the borders of adjacent buildings. Due to the small gaps between the buildings, when these gaps are filled the triangles used to fill are very thin, creating degenerate faces in many programs. This can be solved by either re meshing, or by removing degenerate faces. It visible in fig. 16 that on the left we have degenerate triangles and better geometry, while on the right the degenerate triangles have been removed and the geometry quality has dropped.



(a) (b)

Figure 15: CFD simulation U magnitude for method "NEF"(a), for method "hole filling" (b)

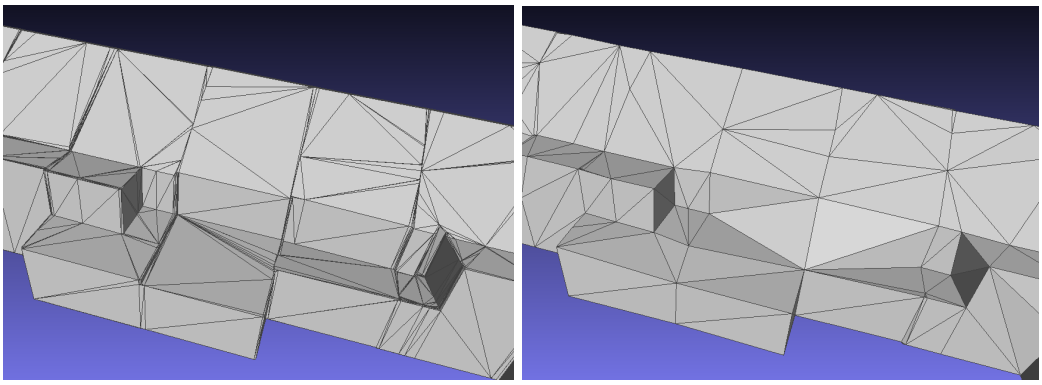


Figure 16: Degenerate triangles between buildings

CGAL 6 (in development) allows for the removal of degenerate faces. The operation succeeds in removing these faces between buildings, but changes the mesh significantly overall. This possibly might be improved once CGAL 6 is released.

6 Advantages and Disadvantages between the methods

From the results in the previous section some advantages and disadvantages are evident for each method.

To start with the advantages, both methods achieved to remove successfully the majority of shared faces between the buildings, supporting the geometry of all Level of Details, in a relatively short period of running time. Additionally, the original mesh is largely unchanged. The NEF method is a very streamlined algorithm as only the Minkowski parameter is required and then CGAL library for the creation of NEF polyhedra is called which makes the process highly automatic and integrated. The Hole Filling method, respectively, gives better control over the individual steps since it segments the process into independent parts and allows for intermediary outputs.

On the other hand, both methods are initialized with some parameters which make the effectiveness of the results highly dependent on the specific data. Adding to that, parameters constitute running the code a time consuming process, since the users have to set them based on trial and error. Last but not least, both methods modify the original geometry which based on the context of the CFD simulation can be undesirable.

Comparing the two methods, the quality of the geometry preservation is higher using Nef Polyhedra and with a correct choice of Minkowski value all shared faces are removed. Also, using only one parameter, the Minkowski value, compared to Hole Filling method which uses two, the diameter of the hole and number of edges, makes the code more user friendly. Hole Filling is faster in terms of runtime performance, and runtime doesn't increase a lot with the increase of the number

of buildings, while NEF's running time highly increases with the number of buildings of the model. Last, Hole Filling also achieves to remove the majority of shared faces but due to the buildings being disconnected, it struggles sometimes to close these gaps, either leaving them in the model or creating degenerate faces (4d).

Concerning specifically the geometry of more complex buildings, such as churches, the Hole Filling method surpasses in performance the NEF method. NEF method cannot handle complex buildings, because the complexity of geometry of those buildings cause the Minkowski operation to break. The big advantage of the Hole Filling, on the other hand, is that it can handle invalid buildings or buildings with complex geometry, by only filling the gaps, which most of the times are rather simple, and keeps the rest of the geometry unchanged.

7 Reflection

We started with developing the Nef Polyhedra method because it seemed more straightforward as a method, offering an automatic and integrated process. Also, it supports all LODs of CityJson as input. An important part of the process is computation time, which is possible through support for multi threading processing. The results of the method are easily validated through open source software, CityJson Validator and Val3dity.

The difficulty of this task was to preserve as much as we could the original geometry, which is slightly modified though this method. Moreover, in order to solve the problem with the small gaps that were created after the union, the Minkowski sum was used, but there is not a universal Minkowski value that can be used for all datasets, which interrupts the automation of the process.

The second method was developed as a an alternative that could preserve the original geometry better. It can simplify the geometry to the level that is good for CFD simulations and it supports all LODs. It however isn't always successful in filling holes, which is undesirable.

7.1 Failed attempts

Polyfit

In order to create a manifold surface after the Symmetrical Difference we tried to use Polyfit (Nan and Wonka, 2017). To explain a little bit more about how Polyfit works, it takes as input the output of the Symmetrical Difference, which was a CityJSON file at that point and samples a point cloud of the model, after that it extracts a set of planar segments from the point cloud using RANSAC (Random sample consensus) and then computes pairwise intersections of the planes to hypothesize of the object's faces. To select for the optimal ones, it runs an optimization process which favors faces that comply with 3 things: fit the data better, minimize the model complexity, and are covered by many points. These 3 terms constitute the objective function that needs to be minimized. The end result of the process is a manifold surface model, which is visible in Figure 18.

Although, Polyfit works well with this specific example it has problems with more complex structures, like curves, or a whole row of adjacent buildings that it why it was not further used for this method. Instead the hole filling operation in CGAL 5.5 is used as it is mentioned above.

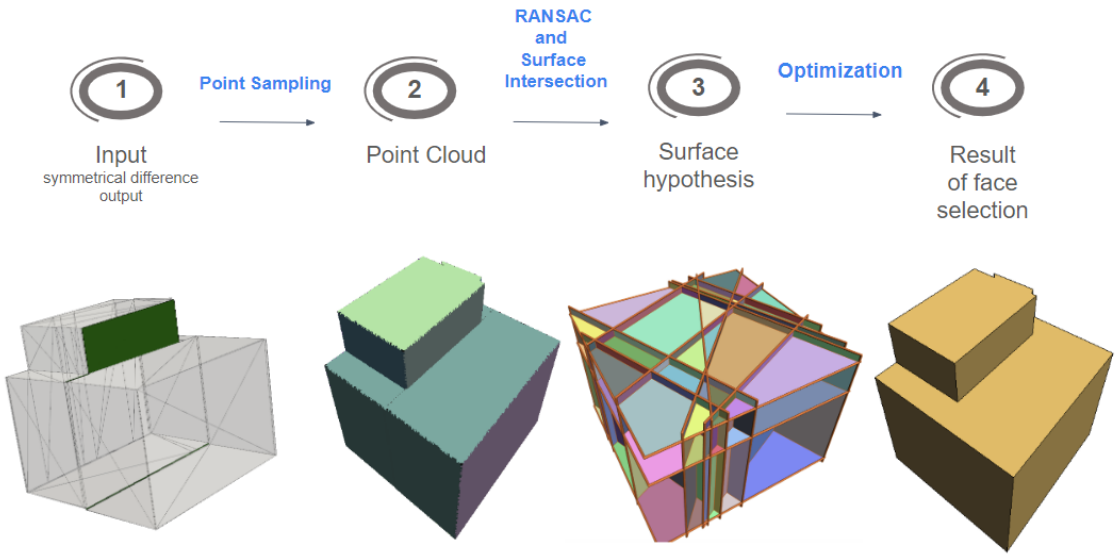


Figure 17: Example datasets of many connected buildings

Erosion

In order to reverse the expansion caused by the Minkowski sum in the Nef Polyhedron method, we tried to implement "Erosion," from Val3dity (Ledoux, 2019). The Erosion function computes the bounding box of the large Nef (with Minkowski sum applied), subtracts this Nef from the bounding box to get its complement, applies the Minkowski sum to complement, and subtracts the complement from the large Nef. Unfortunately, when we tried to apply this method, the resulting geometry was missing many faces. Due to limited time, we could not troubleshoot and figure out the source of the issue. This could be revisited with more time.

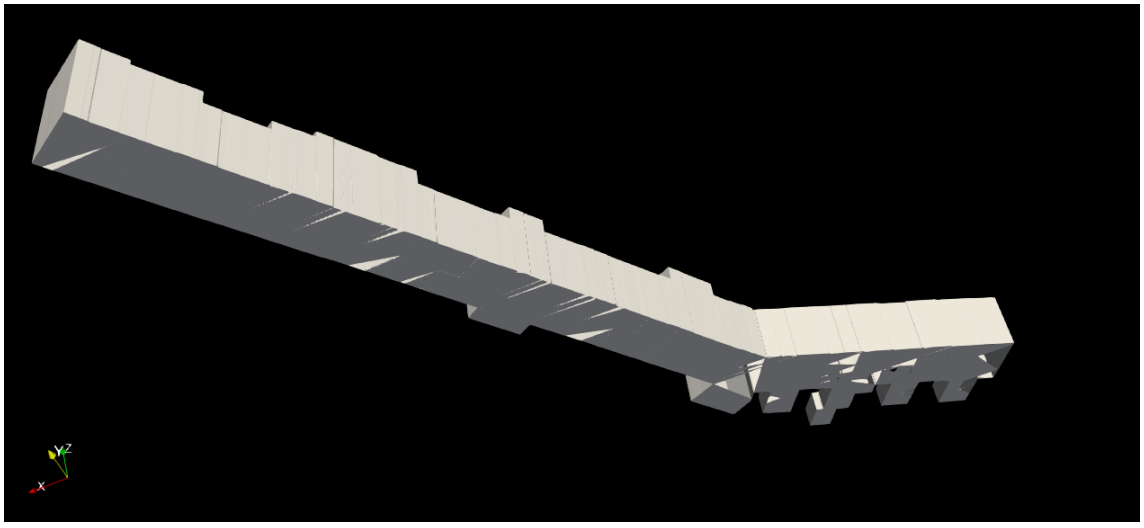


Figure 18: Output from Val3dity Erosion function

7.2 MoSCoW

In the Figure 19 the original scoping expectations of the project are presented, although there were some changes since then. All the 'Must haves' were achieved, more specifically, the 3DBM code was alternated to detect adjacent walls for a whole tile. Additionally, only datasets that pass the val3dity test were used in the process. To optimise the speed of the whole algorithm C++ and CGAL 5.5 were used for the most part, except of the 3DBM script that is in Python and used the Pyvista library.

Moreover, as we go into the 'Should haves' of the table, we wanted to explore different solutions and programming languages to get the best result we could, and that was achieved through our two methods. Although, we struggled to get the LoD 2.2 as an output, while maintaining as much as possible the original geometry. Method: Nef Polyhedra seems to keep the original geometry the best as it is visible in figures 8,10.

Furthermore, focusing on our 'Could haves', one of our goals was to add the fixed adjacent buildings in 3D BAG, but due to time limitations we did not do it. Also, optimizing the computing time of the algorithm was really important for us and the partner company, and we tried to achieve that by running our code with multi-threading, the difference is visible in 4.4. As the client seemed interested in the results of project, we could also ask them to mesh the fixed geometry and evaluate the results, which was really useful to see and try to optimize our methods based on that, some of those attempts are presented in 7.1.

Despite the fact that we mentioned in our 'Would not haves' that we would not be running CFD simulations with the fixed geometry, our supervisor helped us by running them for the outputs of both methods and the results are presented in 4.5.

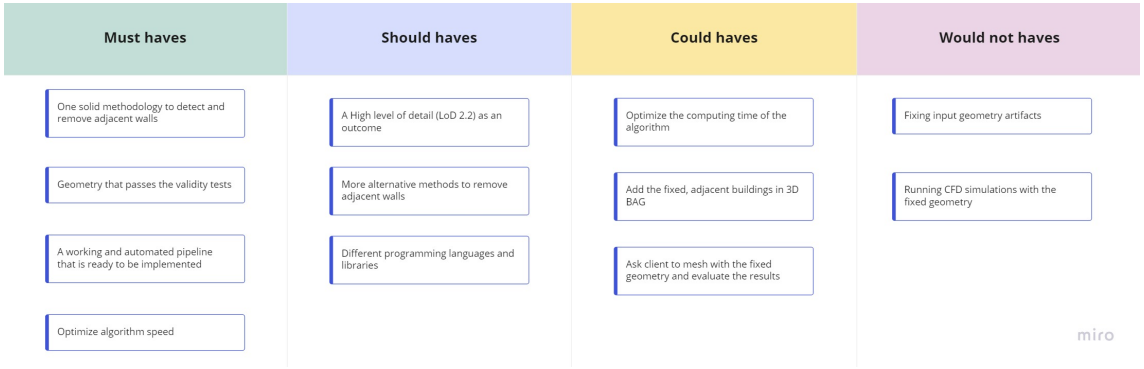


Figure 19: MoSCoW

8 Conclusions

The aim of this project was to create a method that could remove shared faces between adjacent buildings to prepare geometry for CFD simulations. Additionally to that, the proposed solution should be automated, open-source, and the resulting output be of high quality. As it was not the goal of this project to fix invalids geometries, only datasets that could pass the val3dity tests were used.

The research question of this project appears to break into different components that all contribute to the final result. Identifying the adjacency between buildings was an important first step, as the 3D BAG city models that were used as input are not topologically correct, and this task was successfully accomplished by adapting the work of 3DBM. After that, two methods were developed to remove the shared faces between buildings: the Nef Polyhedra and the Hole Filling Method.

The NEF Polyhedra Method successfully removes the shared faces on most occasions, failing in very complex building geometry cases. The quality of the resulted geometry, compared to the original, is highly dependent on the pre-existing gaps between the buildings of the input, which in turn affects the Minkowski value used to close them. The Hole Filling method in the vast majority successfully removes the shared faces and preserves the initial geometry, but sometimes also leaves shared faces within a block and some others fail to create a watertight model.

We find that the work that we have completed is rather succesful. We have developed two methods that are able to remove internal faces with some degree of success, with both methods having different strengths and weaknesses. We also include further progress that can be made to further improve the mesh. We thoroughly enjoyed this project and hope it brings joy to CFD developers.

9 Project Organisation

9.1 Team

9.1.1 Team Members

Adele Therias
BA in Geography Environment
University of British Columbia
Msc Geomatics
TU Delft
Experience
I-SURF Research Assistant at TU Delft
Geospatial Intern at Arup

Chrysanthi Papadimitriou
Integrated Master in Planning and Regional Development
University of Thessaly, School of Engineering
Msc Geomatics
TU Delft
Experience
Internship at Doxiadis Associates
Zuid Holland Waterbound Student Assistant TU Delft

Eleni Theodoridou
Integrated Master in Spatial Planning and Development
Aristotle University of Thessaloniki
Msc Geomatics
TU Delft
Experience
Member of voluntary student group "evzin"

Fabian Visser
Bsc in Mathematics
Leiden University
Msc Geomatics
TU Delft
Experience
Internship at Asset.Insight

Fengyan Zhang
Bsc in Geographic Information Science
Southeast University (SEU)
Msc Geomatics
TU Delft
Experience
Internship at Northwest Regional Corporation, Radiance Group

Ioanna Panagiotidou
Integrated Master in Rural and Surveying Engineering
Aristotle University of Thessaloniki
Msc Geomatics
TU Delft
Experience
Internship at P. Karamoschos and Associates Company

9.1.2 Dassault Systèmes

We are working with Dessault Systemes for this project. Founded in France in 1981, the company develops 3D products used by many different companies world wide, with the goal for sustainable innovation. One product is Simulia, a software for Computational Fluid Dynamics Simulation, which will be our point of interest in the project. Our contact person from Dessault Systemes is Ignacio González-Martino.

9.1.3 TU Delft Supervisors

Clara García-Sánchez
Assistant Professor
Faculty of Architecture and the Built Environment, Urban Data Science
Experience
Computational Fluid Dynamics
Member of 3D geoinformation group
Member of The Breathe Lab

Ivan Pađen
Phd Candidate
Faculty of Architecture and the Built Environment, Urban Data Science
Experience
Member of 3D geoinformation group
Member of The Breathe Lab

9.2 Responsibilities

9.2.1 Team members

A major component of dividing responsibilities among the team members is to equally divide the workload, as much as possible. Therefore, everyone was assigned a specific role in the project but for the different phases of the project each member of the group is assigned a different sub-task. The detailed roles and tasks can be seen in the Gantt chart in section 9.5

Role	**Task**	**Team member**
Coordinator	Ensuring efficient collaboration and monitoring progress and results	Adele Therias
Report Manager	Organizing reports and presentations	Eleni Theodoridou
Technical Manager	Development of an automated workflow	Fabian Visser
Communications	Establishing communication between the team and the supervisors	Fengyan Zhang
Meetings Manager	Developing the meetings agenda and assigning future tasks	Chrysanthi Papadimitriou
Data Manager	Data processing and analysis	Ioanna Panagiotidou

Table 4: General roles of the team

9.2.2 Supervisory team

Role	**Task**	**Name Supervisor**
TU Delft Supervisor	Providing consultation regarding the goals of the project	Clara Garcia-Sanchez
TU Delft 2nd Supervisor	Providing input on the technical aspects of the project	Ivan Pađen
Partner Company Supervisor	Problem definition and feedback	Ignacio González-Martino

Table 5: Supervisors' roles and tasks

9.3 Meetings

As a team, we will be meeting twice a week on Monday and Wednesday in person. On one of these days, depending on availability, we will be meeting with our supervisors to discuss progress, get feedback and develop the next goals. We will discuss personal progress to keep everyone as a group up to speed, and discuss which direction to take the project to, while assigning new tasks to the group members.

9.4 Deliverables

The deliverables are stated in the following table accompanied by the specific deadline of each phase of the project. The final product of the project will be code in the form of an automated pipeline that will be ready to run from the user and a report containing all the vital parts of the project and the methodology that was followed.

Deliverable	**Task**	**Deadline**
Project Strategy Proposal	Individual	8/9/2022
Project summary	Group	9/9/2022
Project Identification Document (PID)	Group	19/9/2022
Midterm presentation	Group	12/10/2022
Report on individual contribution to the midterm report and team process	Individual	12/10/2022
Final written report	Group	4/11/2022
Geomatics day presentation	Group	10/11/2022
Report on individual contribution to the midterm report and team process	Individual	11/11/2022
Report on task distribution between team members and role in the project	Group	11/11/2022
Report on individual reflection on the final report	Individual	14/11/2022
Submit thesis to TU Delft repository	Group	17/11/2022

Table 6: Deadlines for deliverables

9.5 Gantt chart

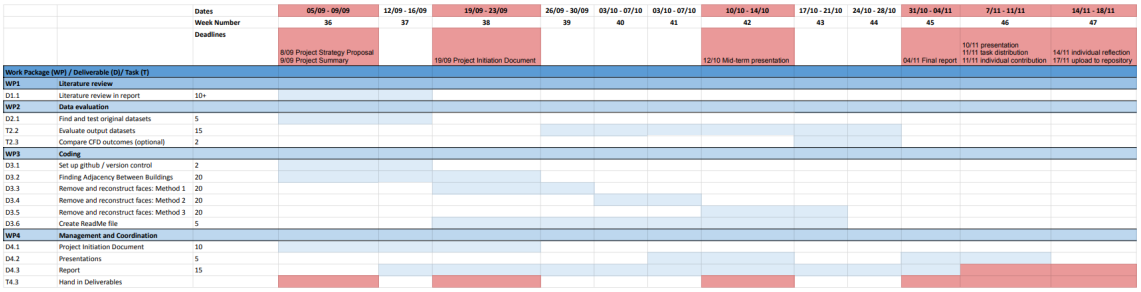


Figure 20: Gantt Chart

Rich Picture

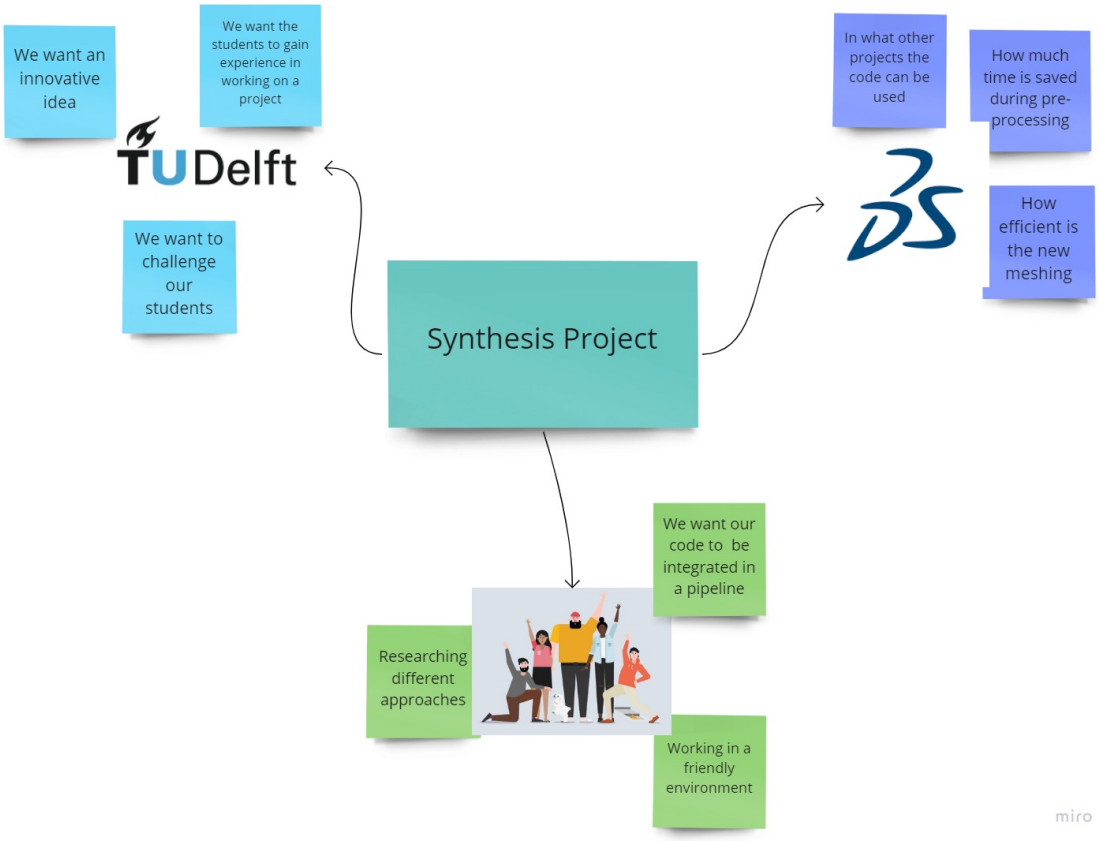


Figure 21: Rich Picture of the Project

10 References

3D BAG, (n.d). 3D BAG viewer, Available at: https://3dbag.nl/en/viewer.

C++, (n.d). Available at: https://cplusplus.com/.

CGAL 5.5, (n.d.). Available at: https://doc.cgal.org/latest/Manual/packages.html.

CloudCompare v2.11.3, (n.d.). Available at: https://github.com/CloudCompare/CloudCompare.

Labetski, A., Vitalis, s., Biljecki, F., Arroyo Ohori, K., and Stoter, J., (2022) 3D building metrics for urban morphology, International Journal of Geographical Information Science, DOI.

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

Ledoux, H., (2019). val3dity: validation of 3D GIS primitives according to the international standards. Open Geospatial Data, Software and Standards, 3(1), 2018, pp.1 DOI.

Ledoux, H. (n.d), CityJSON/io: Python CLI to process and manipulate CityJSON files, Available at: https://cityjson.github.io/cjio/index.html.

Nan, L., and Wonka, P. (2017). PolyFit: Polygonal Surface Reconstruction from Point Clouds.

Normand G. and Bouillot M. (1998). Available at: http://cgm.cs.mcgill.ca/ godfried/teaching/cg-projects/98/normand/main.html

Paraview, (n.d.). Available at: https://www.paraview.org/.

Pađen, I., García-Sánchez, C., and Ledoux, H., (2022). Towards Automatic Reconstruction of 3D City Models Tailored for Urban Flow Simulations. Frontiers in Built Environment, 8, 2022 DOI.

Python 3.9.0, (2020). Available at: https://www.python.org/downloads/release/python-390/.

Pyvista, (n.d.). 3D plotting and mesh analysis through a streamlined interface for the Visualization Toolkit (VTK), Available at: https://docs.pyvista.org/.

Cignoni, P., Callieri, M., Corsini, M., Dellepiane, M., Ganovelli, F., Ranzuglia, G., (2008). MeshLab: an Open-Source Mesh Processing Tool Sixth Eurographics Italian Chapter Conference, page 129-136.

Sean Gillies (n.d.). Shapely 1.8.5, Available at: https://pypi.org/project/shapely/.

Vitalis, S., Labetski, A., Boersma, F., Dahle, F., Li, X., Arroyo Ohori, K., Ledoux, H., and Stoter, J., (2020). CITYJSON + WEB = NINJA, ISPRS Ann. Photogramm. Remote Sens. Spatial Inf. Sci., VI-4/W1-2020, 167–173, DOI.

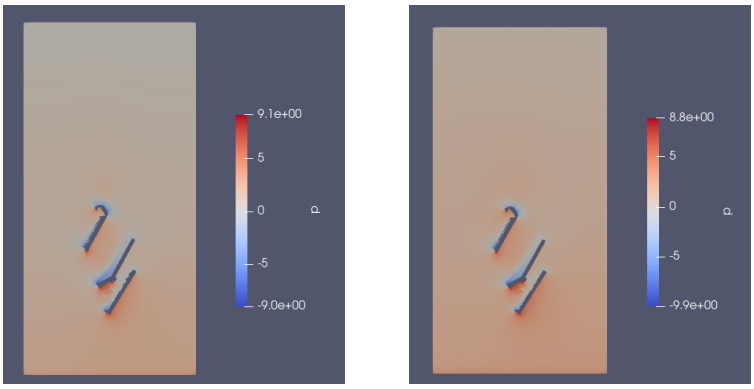
11 Appendix

Minkowski Value	**LoD**	**Max Hausdorff absolute value (m)**	**Mean Hausdorff absolute value (m)**	**RMS Hausdorff absolute value(m)**	**Max Hausdorff normalized value**	**Mean Hausdorff normalized value**	**RMS Hausdorff normalized value**
0.01	1.2	0.032309	0.003637	0.006356	0.000246	0.000028	0.000048
0.01	1.3	0.032309	0.003485	0.006363	0.000246	0.000026	0.000048
0.01	2.2	0.039925	0.012359	0.015018	0.000303	0.000094	0.000114
0.005	1.2	0.032598	0.007101	0.009871	0.000248	0.000054	0.000075
0.005	1.3	0.032598	0.006889	0.009781	0.000248	0.000052	0.000074
0.005	2.2	0.032598	0.005974	0.008637	0.000247	0.000045	0.000066
0.004	1.2	0.032459	0.00557	0.008211	0.000247	0.000042	0.000062
0.004	1.3	0.032459	0.005402	0.008196	0.000247	0.000041	0.000062
0.004	2.2	0.032459	0.004764	0.007429	0.000246	0.000036	0.000056
0.003	1.2	0.032351	0.004298	0.006999	0.000246	0.000033	0.000053
0.003	1.3	0.032351	0.004133	0.007031	0.000246	0.000031	0.000053
0.003	2.2	0.032351	0.003589	0.006303	0.000246	0.000027	0.000048
0.002	1.2	0.032274	0.002996	0.005811	0.000245	0.000023	0.000044
0.002	1.3	0.032274	0.002861	0.005718	0.000245	0.000022	0.000043
0.002	2.2	0.032274	0.002325	0.004763	0.000245	0.000018	0.000036

Table 7: Hausdorff distance for different Minkowski values (Method "NEF", Dataset 3)

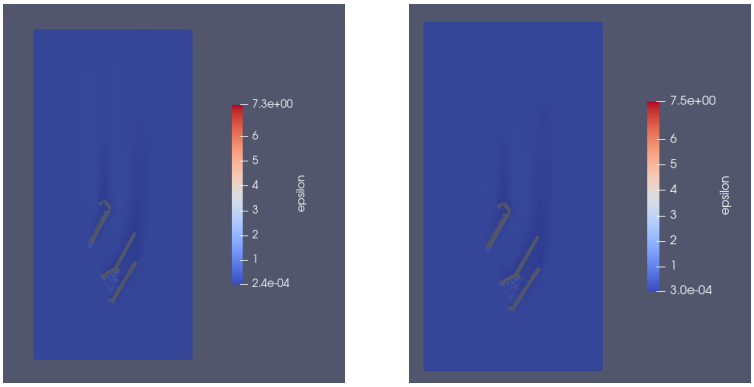
Minkowski Value	LoD	Max Hausdorff absolute value (m)	Mean Hausdorff absolute value (m)	RMS Hausdorff absolute value (m)	Max Hausdorff normalized value	Mean Hausdorff normalized value	RMS Hausdorff normalized value
0.01	1.2	0.036342	0.014693	0.017383	0.000264	0.000107	0.000126
0.01	1.3	0.036342	0.014539	0.017277	0.000264	0.000106	0.000126
0.01	2.2	0.036342	0.012522	0.01507	0.000264	0.000091	0.000109
0.005	1.2	0.032598	0.00795	0.010961	0.000237	0.000058	0.00008
0.005	1.3	0.032685	0.00781	0.010862	0.000238	0.000057	0.000079
0.005	2.2	0.032777	0.006497	0.009317	0.000238	0.000047	0.000068
0.004	1.2	0.032459	0.006406	0.009386	0.000236	0.000047	0.000068
0.004	1.3	0.032459	0.006265	0.009276	0.000236	0.000046	0.000067
0.004	2.2	0.032459	0.005211	0.008011	0.000236	0.000038	0.000058
0.003	1.2	0.032351	0.005039	0.00819	0.000235	0.000037	0.00006
0.003	1.3	0.032351	0.002377	0.005342	0.000255	0.000017	0.000039
0.003	2.2	0.032379	0.004034	0.006874	0.000235	0.000029	0.00005
0.002	1.2	0.032274	0.003747	0.007058	0.000235	0.000027	0.000051
0.002	1.3	0.032274	0.003699	0.006993	0.000235	0.000027	0.000051
0.002	2.2	0.032274	0.002918	0.005833	0.000234	0.000021	0.000042
0.001	1.2	0.032227	0.00215	0.005292	0.000234	0.000016	0.000038
0.001	1.3	0.032227	0.00213	0.005294	0.000234	0.000015	0.000039
0.001	2.2	0.032227	0.001666	0.004475	0.000234	0.000012	0.000033

Table 8: Hausdorff distance for different Minkowski values (Method "NEF", Dataset 4)



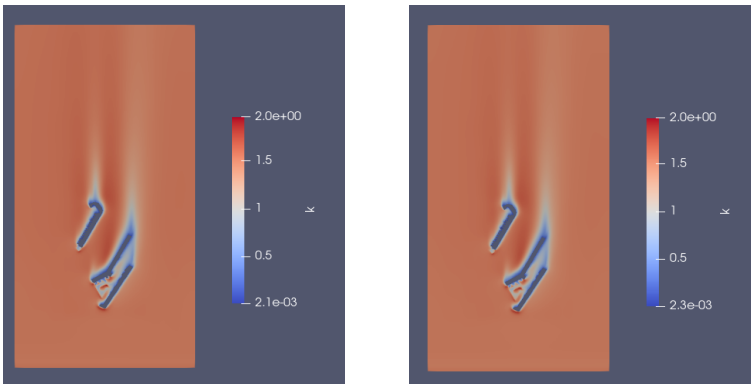
(a) (b)

Figure 22: CFD simulation "p" for method "NEF"(a), for method "hole filling" (b)



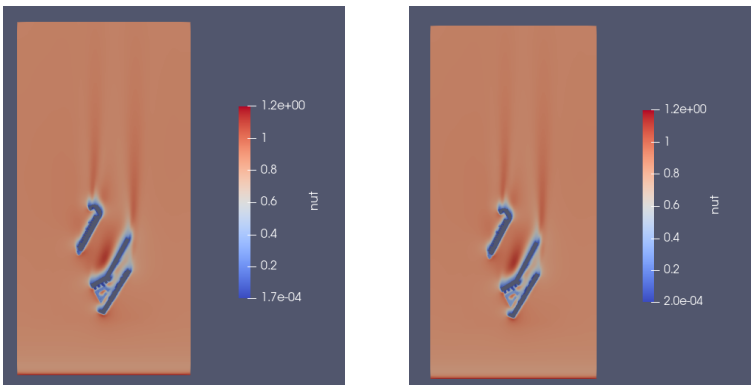
(a) (b)

Figure 23: CFD simulation "epsilon" for method "NEF"(a), for method "hole filling" (b)



(a) (b)

Figure 24: CFD simulation "k" for method "NEF"(a), for method "hole filling" (b)



(a) (b)

Figure 25: CFD simulation "nut" for method "NEF"(a), for method "hole filling" (b)

12 Special Thanks

Hugo Ledoux - h.ledoux@tudelft.nl
Ken Arroyo Ohori - k.ohori@tudelft.nl
Liangliang Nan - liangliang.nan@tudelft.nl
Stelios Vitalis - s.vitalis@tudelft.nl