



Università  
della  
Svizzera  
italiana

Faculty  
of  
Informatics

Bachelor Thesis

June 14, 2019

# Large-Scale 3D Printing of Urban Areas

Simone Masiero

---

*Abstract*

While digital visualizations are extremely useful and often a lot convenient, sometimes the necessity of having a tangible model arises. Designers, engineers or researchers in general, may need to construct artifacts of their work. Architects, more specifically, often manufacture a scaled version of their project to present it, usually including the context where these projects are located. These kinds of models, which are incredibly expensive in terms of money and time, are called urban area models. 3D printers may help during this process, however, they not only need a precise 3D model of the whole area to print, which may be difficult to design, but they also impose limits on the maximum printable volume, e.g., a standard 3D printer has a maximum printable volume of nearly  $15\text{cm}^3$ . The goal of this project is to develop an approach to automatically create 3D physical representations of urban areas such as cities and districts using the Swiss Federal Office of Topography geospatial data, which includes detailed 3D models of buildings and their locations, adjust this model in order to make it printable, subdivide it into interlocking sections which can be printed separately and merged to form a 3D model of arbitrary size.

**Keywords:** 3D-modeling; 3D-printing; Urban Areas;

---

**Advisor:**

Prof. Dr. Michele Lanza

**Co-advisor:**

Dr. Roberto Minelli

---

Approved by the advisor on Date: June 14, 2019

# Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
<b>2</b>	<b>Background</b>	<b>2</b>
2.1	Goal of the Work . . . . .	2
<b>3</b>	<b>Approach</b>	<b>3</b>
3.1	Repair Phase . . . . .	4
3.2	Sub-Division Phase . . . . .	6
3.3	Connecting Phase . . . . .	7
3.4	Wrapping Up . . . . .	9
<b>4</b>	<b>Case Study</b>	<b>9</b>
4.1	Background . . . . .	9
4.2	Preprocessing . . . . .	10
4.3	Settings . . . . .	11
4.4	Results . . . . .	11
<b>5</b>	<b>Conclusion and Future Work</b>	<b>11</b>

# 1 Introduction

By the early 1990s, the field of 3D modeling and printing started a revolution, not only from an engineering point of view but also in every area linked with model design, construction, and visualization. Generally, 3D printed models are extremely practical aid to help people grasp difficult concepts more easily. Actually being able to hold and play with an object gives them a far richer and deeper understanding of a subject. Moreover, with 3D modeling having become a core of architectural design, land registers can now offer data for new opportunities and challenges in the field of spatial visualizations. In particular, Switzerland possesses an impressive geographic information system (GIS<sup>1</sup>), a system designed to capture, store, and present spatial or geographic data covering the whole country. Among this data, approximately 70 million of 3D objects are available. Entities like the Swiss Federal Office of Topography<sup>2</sup> offers these digital resources. Using this considerable amount of data, one can design a faithful 3D map reconstruction of districts, cities or even entire Swiss cantons. 3D digital models are, however, constrained in the unnatural 2D world of computer monitors and it is often useful to print them, Studies[4] shows that constructing a recognizable 3D visualization comes with a lot of perspective problems that we do not have to face if we build a physical representation. An architect, for example, usually produces a scaled replica of his project, maybe immerse in the context where it is located, to let people understand his project, but this process involves a lot of money and time. And here is exactly where 3D printers, and additive manufacturing<sup>3</sup> (AM) process in general, comes in. However, even if these machines are becoming more and more precise and sophisticated, they tend to preserve one major constraint: a limited 3D printing volume i.e., the maximum size that one object can have in order to be printed. It is really hard to print something as big and with plenty of details as a city in the maximum printing volume 3D printers can offers. This raises the need for a software which automatically subdivides a model into printable sections that can be later assembled in a wider model of arbitrary scale.

# 2 Background

None of the popular 3D computer graphics software toolsets has a function like this available. Despite you can reproduce this result in programs like Blender<sup>4</sup> or 3DS Max<sup>5</sup>, the work should be done ad hoc for one specific model and it will anyway be difficult to achieve the same level of precision.

Moreover, 3D models may comes with plenty of errors which are irrelevant from a visualization point of view but critical if we want to work on them. In the case study we will present in Section 4, the model we decided use is automatically generated from data and has up to one million of vertex which are not perfectly aligned, a lot of objects are disconnected, and some faces are missing. These are major complications for printing it and makes the task of manually model the geometry fairly impossible.

## 2.1 Goal of the Work

In this thesis we propose a novel 3D processing tool, written in Python using the Blender API<sup>6</sup>, our approach helps 3D model designers to print geometries of any arbitrary scale, regardless of the maximum printable volume of their 3D printer, by automatically repair and subdivide 3D objects into printable and assemblable sections. More in details, the contribution of this thesis are:

- A Repairer tool to adjust mesh object;
- A Sub-divider tool which allows decomposing mesh object in an arbitrary number of pieces;
- A connecting tool to form interlocking components in order to assemble them once printed.

We wrapped all of them in one single Blender add-on component that we will soon release and which can be directly installed into Blender.

Before proceeding with the tool architecture, we must clarify which functional requirements we want to satisfy, namely:

1. The tool must verify that the loaded model is printable, otherwise, must try to repair it (see section 3.1).

---

<sup>1</sup>[https://en.wikipedia.org/wiki/Geographic\\_information\\_system](https://en.wikipedia.org/wiki/Geographic_information_system)

<sup>2</sup><https://www.swisstopo.admin.ch/en/home.html>

<sup>3</sup><https://www.3dhubs.com/knowledge-base/additive-manufacturing-technologies-overview>

<sup>4</sup><https://www.blender.org/>

<sup>5</sup><https://www.autodesk.it/products/3ds-max/overview>

<sup>6</sup>[https://docs.blender.org/api/blender\\_python\\_api\\_current/bmesh.html](https://docs.blender.org/api/blender_python_api_current/bmesh.html)

2. If the model is printable (or made printable after the repair phase) it has to be divided into printable section according to the specifics of the 3D printer and the final size the user wants to obtain.
3. The dimensions of the final model, once assembled, must be the ones specified by the user.
4. All sections must be joinable (i.e., two near sections can be interlocked together).

We also aim to satisfy the following non-functional requirements:

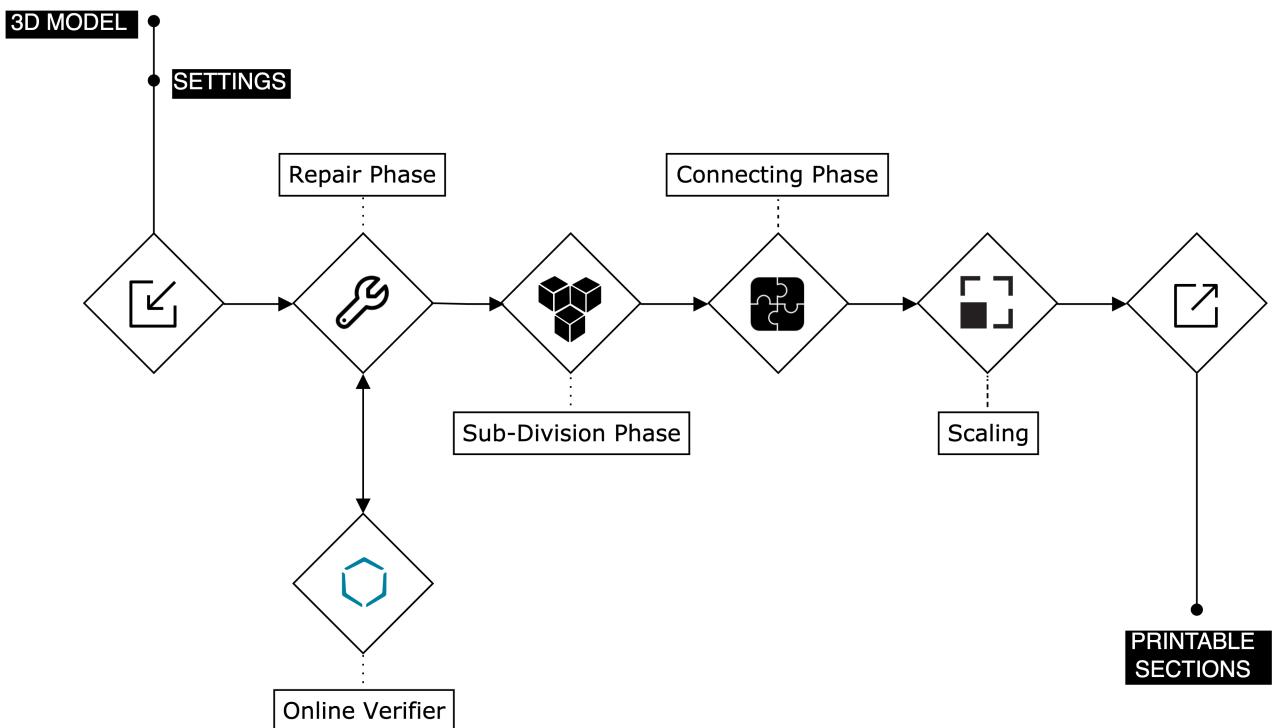
1. The repair phase, if executed, should not compromise the original object.
2. All section should be uniquely identified through a label to facilitate the reconstruction of the final model.

We will now go though the process steps and present then a case study applied to the ongoing postdoctoral work of Minelli, where we deal with an automatically generated urban model of the city of Lugano.

### 3 Approach

In this section we will describe in detail our approach to repair the model, subdivide it into sections and made these section interlocking.

The tool will receive an ‘stl’<sup>7</sup> (stereo-lithography) object, along with the specifics of the users 3D printer and the final size which the user needs to achieve and will output  $n$  printable partitions of that object plus a list of components needed to assemble these partitions in a Lego brick fashion. A complete workflow can be found in Figure 1.



**Figure 1.** Workflow of the process

The workflow to generate the printable partitions consists of three phases: repair (Section 3.1), sub-division (Section 3.2), and connection (Section 3.3).

---

<sup>7</sup>[https://en.wikipedia.org/wiki/STL\\_\(file\\_format\)](https://en.wikipedia.org/wiki/STL_(file_format))

### 3.1 Repair Phase

In order to 3D-print any object, one requirement for that particular object is to be manifold<sup>8</sup>. A manifold is a topological space that is locally Euclidean (i.e., around every point, there is a neighborhood that is topologically the same as the open unit ball in  $\mathbb{R}^n$ ). A non-manifold geometry, on the other hand, is a 3D shape that cannot be unfolded into a 2D surface with all its normals pointing the same direction. Basically a non-manifold geometry cannot exist in the real world, hence, it cannot be printed. The repair phase aim to fulfill the following requirements:

- **Create a closed 3D model with voluminous surfaces:** To ensure that the geometry is closed (or "watertight") we must verify that the 3D-model does not contain surfaces which do not bound a volume: a surface without a thickness or which does not contribute to a volume cannot be 3D printed. This problem can be corrected by giving the surface(s) volume [2]. In other cases, there may be small holes which prevent volume from being "watertight," in this case they must be filled by adding a new face. Figure 2 shows the difference between closed printable meshes (in green) and a mesh without volume (in red)



Figure 2. Watertight requirement

- **Correct non-manifold edges and singular points:** To define a clear volume, each edge must be connecting two and only two adjacent faces. Similarly, singular points must arrive at the collection of multiple faces. These singularities can be eliminated by either disconnecting the non-manifold surface and giving it volume, or by deleting it completely. Figure 3 shows two examples where edges connects more than two faces (in red) leading to an undefined volume.

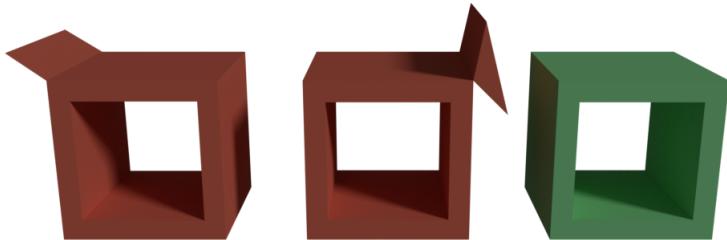


Figure 3. Manifold edges and singular points handling requirement

- **Delete auto-intersections:** Some geometries may have problems such as two or more volumes cut into each others. These intersections create an ambiguous model with uninterpretable volumes. We can address this problem by performing a union of the meshes which are intersecting. Figure 4 shows how a 3D mesh unfolded in a 2D space would appear with or without auto-intersections.

<sup>8</sup><https://en.wikipedia.org/wiki/Manifold>

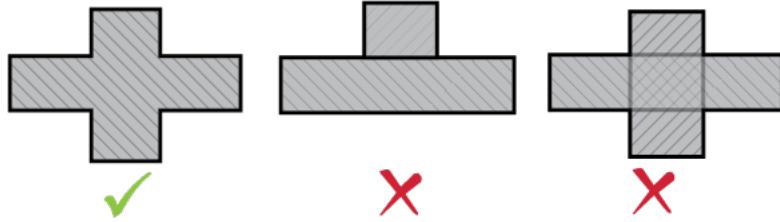


Figure 4. Absence of auto-intersection requirement

- **Correctly orient model's surfaces:** If one of the faces of an object is oriented in the wrong direction, its volume may be indeterminable. It is important to ensure that each face is oriented in the correct direction in order to avoid that type of problem. Figure 5 shows an example of correct and one of incorrect normal directions, in the latter case, the object cannot be unfolded in a 2D space with all its normal pointing in the same direction, this breaks the manifold condition.

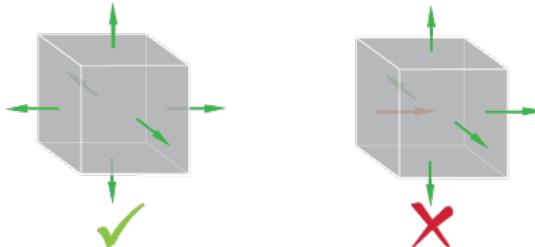


Figure 5. Correct normals requirement

- **Separate Objects:** Even if this is not a proper manifold problem, a floating piece or separate object may ruin our final result, it happens when the geometry consists of more connected components. This particular problem may be addressed in a different way based on the result we are seeking for: In the case study we will present in Section 4, for example, we will see that separate objects are buildings floating over the ground, so to solve this problem, separated components are just translated towards the ground until they intersect it.

These problems may or may not arise depending on how the original mesh is constructed, in the case study of section 4, for example, the mesh is automatically generated from raw data, and all these problems were encountered, we will see how this phase is crucial for the final result.

Furthermore, different kind of contexts comes with different kinds of problems: Consider for example the last problem presented: separate objects. This is a general problem which any geometry can present, however, it is particularly recurrent in urban area models. The process we applied to solve it does not work for any arbitrary mesh: In this particular case, we can exploit the fact that a ground plane exists and translating building against it will produce an intersection. One possibility, for any arbitrary geometry, may just be removing the whole disconnected component, nevertheless, this violates one of our non-functional requirements since we are altering the original mesh. This repair phase is precisely tweaked in order to work with a specific kind of models, namely, urban models.

Mesh repairing, anyway, is a well-known problem in 3D modeling, and a lot of services and functions are available to solve it [1] [3]. The Blender API offers many procedures for repairing each type of non-manifold and these are embedded in our tool.

Moreover, since the computational cost of the whole procedure depends on the number of vertex, edges, and faces, in this phase we will also try, where possible, to simplify the mesh while keeping the same level of details. More in detail:

- Faces are flattens until a fixed angle (i.e., if two faces share an edge and they are within an arbitrary angle these faces are merged);
- Disconnected vertices and edges (optionally faces) are removed;
- Degenerate components are removed (i.e., collapses or removes geometries like edges with no length, faces with no areas, etc.).

At the end of this phase, in order to certify that our mesh is actually printable, one optional procedure uses Trinckle<sup>9</sup>, a web service which act as verifier and, if needed, as an in-depth repairer for 3D objects (Note that we cannot use this, or any, repairer from the beginning, since they are general purpose repairer and they would not consider some specific problems of urban meshes). The result, after this phase, is a manifold geometry which can now be processed for sub-division.

### 3.2 Sub-Division Phase

Once the model is repaired and ready to print, we have to tackle the biggest challenge of our approach: Splitting it into sections. the goal here, in fact, is to produce manifold sections of arbitrary sizes subdividing the original mesh.

A geometry can be divided into separate meshes in different ways, we chose to adopt the division by planar cuts: A 3D grid is constructed around the object and content of each cell in this grid will then represent one of the final section in which the original geometry will be divided.

What we are actually constructing is a rectilinear grid<sup>10</sup>, a tessellation by rectangular cuboids that are, in our case, all congruent to each other, this structure can be constructed arranging parallel planes whose normals point towards one of the three axes.

To decide the dimensions of each section, we need to know the specifics of the 3D printer ( $max_x$ ,  $max_y$  and  $max_z$  printing size e.g., for Prusa i3 MK3 : 25 x 21 x 21 cm) and the dimension we have chosen for the final model ( $final_x$ ,  $final_y$  and  $final_z$  printing size)

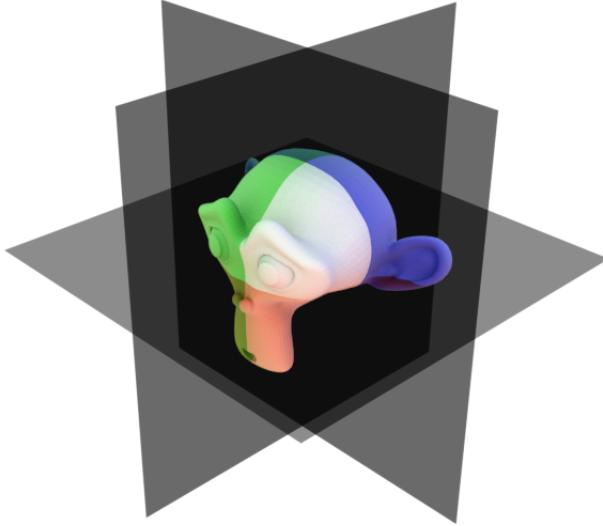
At this point we can derive, for each axis, how many planar cuts we have to perform:

$$\#cut_x = \left\lceil \frac{final_x}{max_x} \right\rceil - 1; \quad \#cut_y = \left\lceil \frac{final_y}{max_y} \right\rceil - 1; \quad \#cut_z = \left\lceil \frac{final_y}{max_z} \right\rceil - 1;$$

then, for each axis, the distance between each cutting plane (i.e. the sizes of the final sections to be printed) will be:

$$step_x = \frac{final_x}{\#cut_x}; \quad step_y = \frac{final_y}{\#cut_y}; \quad step_z = \frac{final_z}{\#cut_z};$$

Finally, we can use each of these planes to perform a planar cut on the mesh through bisect operators (see Figure 6).



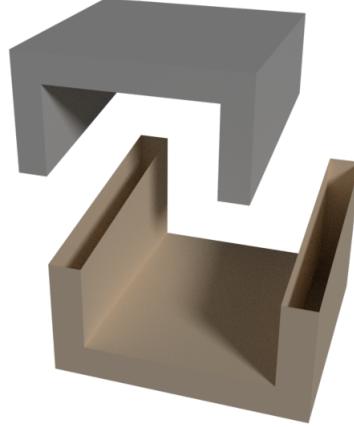
**Figure 6.** The rectilinear grid constructed around a mesh

The downside of this process is that the two new meshes, generated by each bisection, present now at least one hole where the planar cut has taken place. In Figure 7, for example, after one cut, the mesh presents two different holes.

---

<sup>9</sup><https://www.trinckle.com/en/index.php>

<sup>10</sup>[https://en.wikipedia.org/wiki/Regular\\_grid](https://en.wikipedia.org/wiki/Regular_grid)



**Figure 7.** Holes after a planar cut

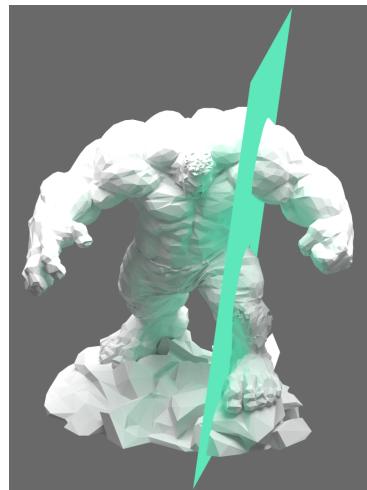
As explained in Section 3.1 holes must be filled in order to make the geometry printable. To fix this, after every cut and for both the new meshes, all the generated hole are traced, for every hole all the near edges associated with only one face are selected so that a new face can be generated connecting these edges.

Here we are exploiting the property of a manifold mesh for which every edge will connect two and only two faces, if one edge is associated with only one face then that edge is part of the hole perimeter. Knowing this, we can select all these edges and grouping them based on vertex sharing. At this point, we have all the groups of connected edges that are associated with only one face (i.e., the perimeter of each hole). Now we can insert a new face between each group of edges to fix the holes.

As a remark, since Blender standard mesh system is not sufficient to perform any of the operations we need to apply to correctly subdivide and repair the model, the original geometry must be translated into a Blender mesh<sup>11</sup> (or "Bmesh"): a robust and versatile representation of a mesh which blender uses for its internal mesh editing API. One of the main capabilities of BMesh is full NGon support, which is a way to create and edit polygons with more than four vertices. This helps to make the modeling process far less destructive and easy to work with. Along with NGons, BMesh also supports modeling features used to bisect and repair the mesh in a faster and more reliable way.

### 3.3 Connecting Phase

Once sub-division has been performed, we end up with a lot of geometries to print, reconstruct the original mesh is like doing a puzzle. Moreover, some cuts may have been made over a section of the mesh which now needs support to rest in place: In Figure 8, for example, once we apply the cut through the plane shown, the arm of the character will fall without tailored support for it.

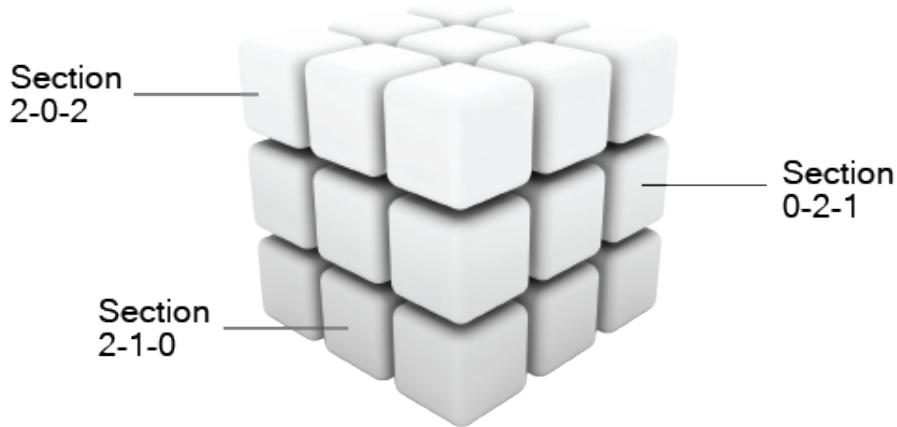


**Figure 8.** Cut one part of a geometry which is not self-supported

---

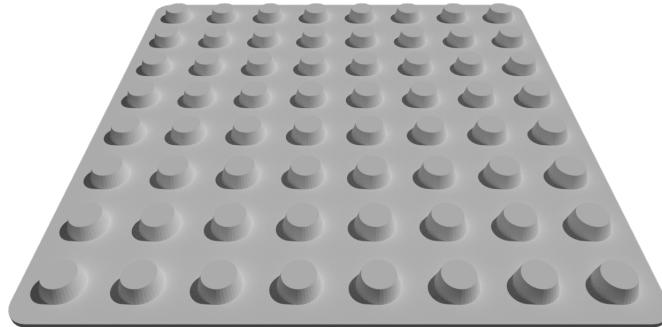
<sup>11</sup>[https://docs.blender.org/api/blender\\_python\\_api\\_current/bmesh.html](https://docs.blender.org/api/blender_python_api_current/bmesh.html)

The former problem can be simplified by labeling each mesh with his XYZ coordinates, where x, y, and z are the indexes, in three dimensions, of the relative cells of the grid in which the original object was divided.



**Figure 9.** Labels of sections in 3D space

As for the latter problem, the idea here is to mimic the Lego<sup>12</sup> brick system. A grid of small cylinders (see Figure 10) is added to one mesh and hollowed on his counterpart, effectively producing interlocking models so that they can be manually assembled in one single object.



**Figure 10.** Lego-like grid of cylinders

In this phase we widely rely on mesh Boolean operations [5], these are fundamental operations in 3D modeling: they combine two or more solid shapes (say A and B) by checking if a point x lies inside of each solid. In particular, we use union:

$$\text{Union: } A \cup B := \{x \in \mathbb{R}^3 | x \in A \text{ and } x \in B\}$$

to add the grid of cylinders on one side of the cut, and difference :

$$\text{Difference: } A \setminus B := \{x \in \mathbb{R}^3 | x \in A \text{ and } x \notin B\}$$

to hollow them on the opposite side of the cut.

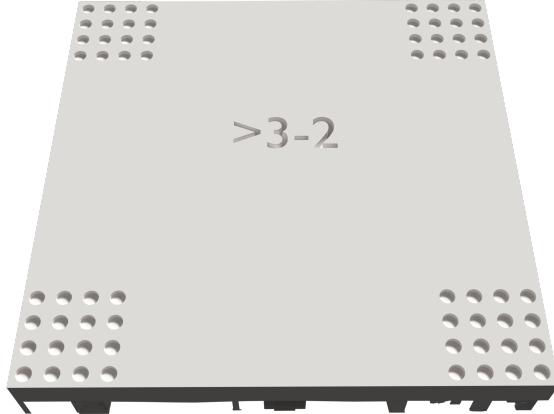
Difference is also used to add the label which describes the position of the section. In order to do this, we have to transform a text into a 3D mesh, then we will add thickness to it and after having placed in the right spot we can use the difference boolean modifier to hollow it in the mesh (see Figure 11)

---

<sup>12</sup><https://www.lego.com/it-it>

Here, the problem we have to face is where to place both the label and the grid of cylinders. Let's, for example, have a look at the mesh of figure 7. In this mesh we applied a bisection which generates two disconnected holes. Hollow cylinders is not really a problem since if we arrange all the cylinders in a plane, only the ones which actually intersect the mesh will produce a hollow.

On the other hand, add cylinders on the opposite mesh and hollow the labels are operation for which the locations where we apply these operations really matters. The idea, then, is to trace the faces we added in Section 3.2 to repair holes, hollow the label in the biggest face we added and append cylinders only if holes are big enough to support them. The overall result will be something like figure 11 where both the label and the grid of cylinders are dig into the base of the geometry.



**Figure 11.** Bottom side of a final section where both the label and the grid of cylinders are inserted

Notice that in this particular mesh we do not opt for a full grid of cylinders on all the surface of the cut, we just need to apply enough to ensure the interlocking condition.

### 3.4 Wrapping Up

So far we have seen one possible workflow which is precisely crafted for working with generated urban models. Anyway, as we already mentioned, the context of the 3D model can drastically change how the process must behave, in fact, depending on the original geometry, some feature of this tool may be unnecessary, some other may need specific settings or little adjustments which would improve the final result. As outlined above, different options in the repair phase (Section 3.1) can improve some kind of 3D models but damage others. The same also holds for Sub-division phase (Section 3.2) and Connection phase (Section 3.3). To have a better understanding of our current solution we will now see how this process is applied to a metropolitan area 3D model, namely, the city of Lugano.

## 4 Case Study

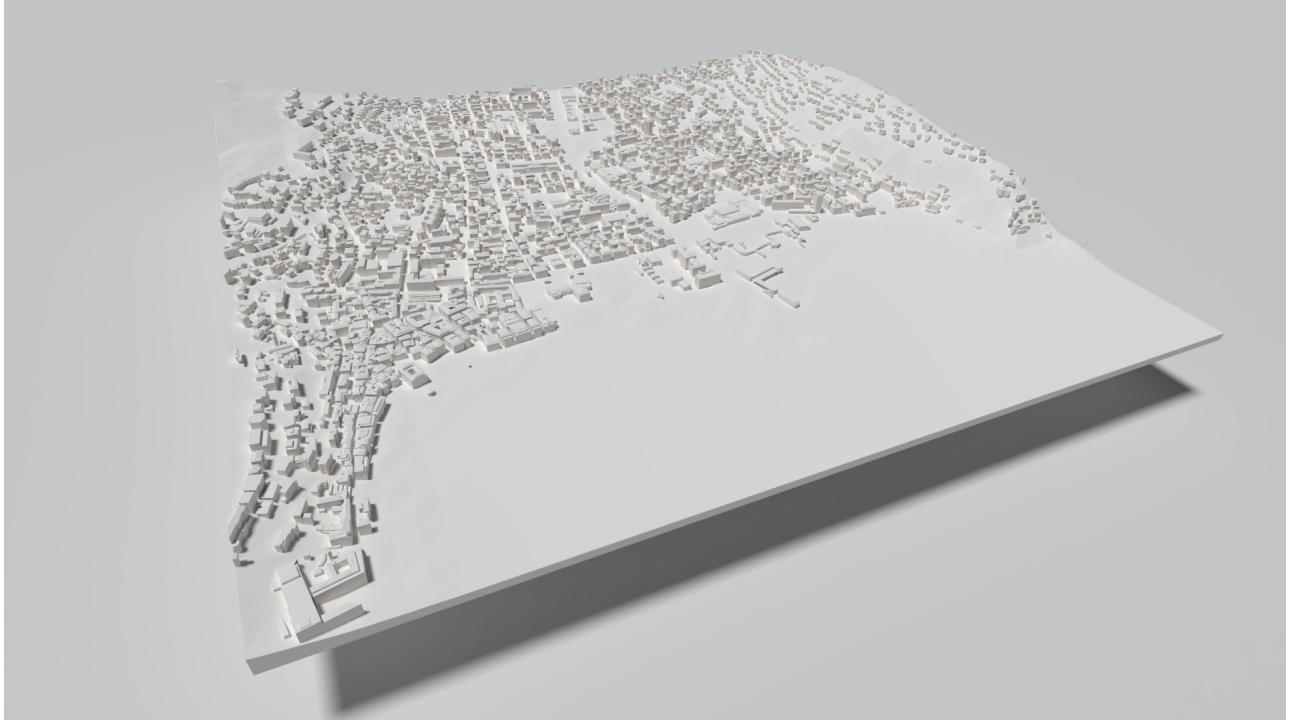
To properly test this tool, we decided to work on a complex urban model (around 1 million of vertex) of the city of Lugano which is automatically generated with Scala using the Swiss GIS data. Moreover, data for this model are designed to construct spatial visualizations and they do not satisfy any printing requirement.

### 4.1 Background

Starting from the work of Minelli, using topological data from the Swiss Federal Office of Topography for the buildings and google maps API<sup>13</sup> for the terrain generation, we are able to construct an accurate 3D model of Lugano city in 'stl' format.

---

<sup>13</sup>GoogleMaps API <https://cloud.google.com/maps-platform/maps/>

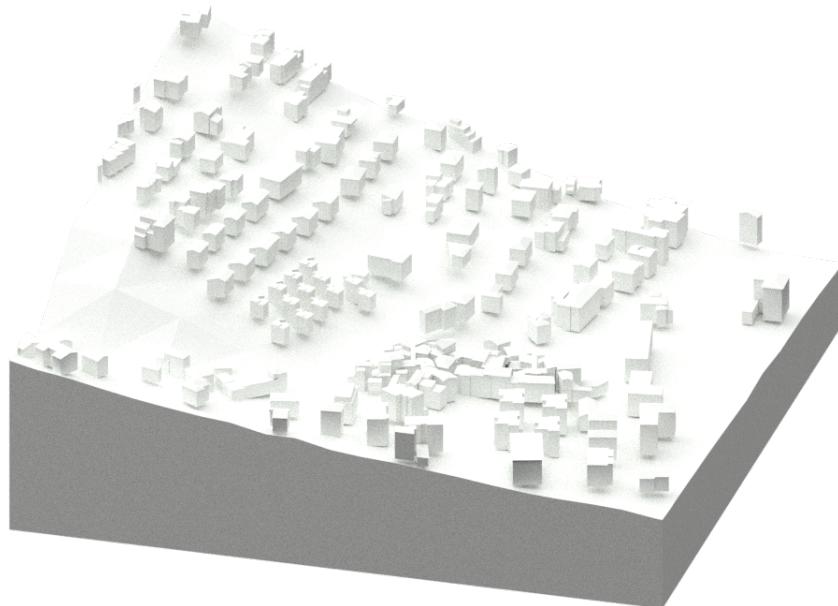


**Figure 12.** Reconstruction of Lugano city center ( $4\text{km}^2$ )

The printer which we will use is a Prusa i3 MK3<sup>14</sup>, which has a maximum print volume of roughly  $11\text{cm}^3$  ( $21 \times 20 \times 25\text{ cm}$ ) and the base of the final model we aim to print will be  $1 \times 1\text{ m}$ . Since our model is a large section of a city, its height will be relatively small compared to its length and width. This means that we do not have to cut the model on the z-axis since the capacity of the printer for that dimension is enough

## 4.2 Preprocessing

This mesh is really difficult to treat since, as mentioned above, it is automatically generated starting from data collected for 3D visualizations and not for a precise reconstruction. In order to use this model we must first adjust it. One of the biggest complications is that data used to construct the building's grid does not precisely match with the generated terrain i.e., some buildings may float over it leading to an impossible print.



**Figure 13.** Buildings flying over the terrain

---

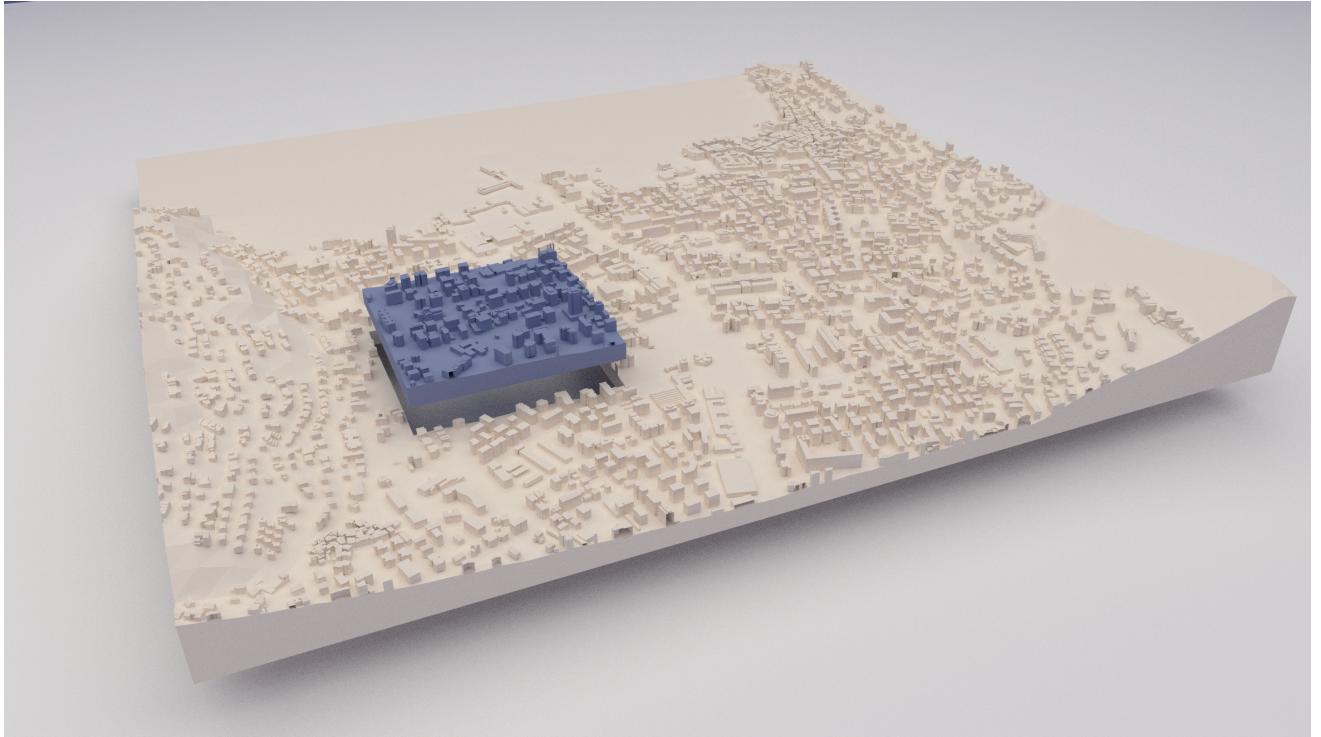
<sup>14</sup><https://shop.prusa3d.com/en/>

This happens because the API used to generate the terrain has a low usage limit which does not allow us to reconstruct a faithful terrain representation, we sampled altitude coordinates in a grid with steps of 35 meters, then each point on this grid is translated on the z-axis relative to the altitude of that point. This process forces us to interpolate the terrain, leading to an approximation of the real surface shape. Figure 13 shows how some buildings are flying over the terrain.

In order to address such a problem, we modelled a procedure in the repair phase which will find all these separate components and push them towards the ground until they intersect it. A lot of other minor problems, like faces in buildings which does not exactly match or impossible geometries, are automatically handled by our tool.

### 4.3 Settings

Given the final size of 1x1 m that we have chosen for the base of our model and the dimensions of our printer (21x20x25 cm) we will perform four cuts on both the x and y-axis, this will sub-divide the model into 25 sections of  $400\text{cm}^2$  each. We will not perform any cut on the z-axis. As for the connecting phase, we decided to model a Lego-like brick specifically for this model and we will use this to connect 4 mesh at a time where they intersect i.e., at the corners.



**Figure 14.** Final model highlighting a single section

### 4.4 Results

A printed version of the model shown in Figure 12 has been produced and results are encouraging: Despite some negligible detail not fixed in the repairing phase (caused by the imprecise data in the buildings database) the overall system works well and meets the requirements we set for this project.

## 5 Conclusion and Future Work

In this thesis, we tackled the problem of 3D print a large scaled urban model in a cheap and reliable way. We saw that, especially in architecture, manufacture wide models is mandatory, and, actually, the usual process of manually building them is extremely expensive both in term of time and money. With our tool, we outlined an effective and cheaper way to achieve the same goal by relying on additive manufacturing. We explained how a 3D urban reconstruction model can be generated through Switzerland GIS data, from the Swiss Federal Office of Topography, for the buildings grid and geospatial information, from GoogleMaps, to model the terrain surface; We then pointed out how this model can be repaired in order to made it printable and, after this phase, how from this model, using the Blender API, we can generate interlocking and scaled section of the same model that, once printed, could be

connected together to form a bigger version of the same original model. Finally, we presented a case study where a printed version of the model shown in Figure 12 has been produced. Results are encouraging: Despite some negligible detail not fixed in the repairing phase (caused by the imprecise data in the buildings database) the overall system works well and meets the requirements we set for this project, however, we still have some upgrades to perform.

Even if this tool works with any geometry, as explained, is precisely developed for urban areas, hence a specific way to subdivide into sections and model the joints point is adopted. However, while developing this project we tested a lot of different procedures, each of those dealing with a different kind of 3D model. If the user can give information about the geometry to work on, we could decide which process to use. For this reason, the next step will be implementing it through a user interface, where the user could also input the general parameters like printer capacity and final scale to adopt. We also mentioned B-mesh, this mesh system comes with a lot of possibilities and a rich API, for the moment we just adopt this mesh-scheme to extract and adjust section, however, this has a lot of potential and it can also be added in different components, repairing the original mesh could probably be a lot faster and precise adopting B-mesh functionalities. At the moment we developed a standalone tool, soon we will implement this as sub-routine in the project of Minelli, which actually provide ‘stl’ models for both buildings and terrain. We aim to create a simple service building an effective pipeline that generates, repair and subdivides urban models.

## References

- [1] Jacques Cali, Dan A. Calian, Cristina Amati, Rebecca Kleinberger,. 3d-printing of non-assembly, articulated models. <https://dl.acm.org/citation.cfm?id=2366149>.
- [2] Mario Botsch, Leif Kobbelt, Mark Pauly, Pierre Alliez, Bruno Levy. On the volume elements on a manifold. <https://web.ma.utexas.edu/users/dafra/M392C-2018-MorseTheory/Readings/Moser.pdf>.
- [3] Mario Botsch, Leif Kobbelt, Mark Pauly, Pierre Alliez, Bruno Levy. Polygon mesh processing. <https://www.taylorfrancis.com/books/9780429195709>.
- [4] Michael J. Tarr, Pepper Williams, William G. Hayward Isabel Gauthier . Three-dimensional object recognition is viewpoint dependent. [https://www.nature.com/articles/nn0898\\_275](https://www.nature.com/articles/nn0898_275).
- [5] W Sun, X Hu. Reasoning boolean operation based modeling for heterogeneous objects. <https://www.sciencedirect.com/science/article/pii/S0010448501001312>.