The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences, Volume XLVIII-2/W8-2024
8th International ISPRS Workshop LowCost 3D - Sensors, Algorithms, Applications, 12–13 December 2024, Brescia, Italy

Editing Watertight Manifold Polyhedra using Face Shifts with Automatic Topological Updates and Edge Flips

Florent Geniet, Mathieu Brédif, Bruno Vallet

Univ Gustave Eiffel, ENSG, IGN, LASTIG, F-94160 Saint-Mandé, France
florent.geniet@ign.fr, mathieu.bredif@ign.fr, bruno.vallet@ign.fr

Keywords: 3D reconstruction, 3D building models, polyhedral modeling, half-edge structure, plane arrangements

Abstract

In the context of urban 3D mapping, the 3D modelling step is a crucial operation, which can be very error prone, particularly when high fidelity and accuracy are required. While automatic reconstruction tools are way faster and less expensive than manual ones, these last are less sensible to the data defects.
To combine all this complementary advantages, we propose a semi-automatic approach, where human operators will manually correct the result of an automatic reconstruction tool.
While the automated reconstruction tool will be taken on the shelf, we have developed our own polyhedral modeler tool to make the correction step. In this article, we present this polyhedral modeler, which is based on face shifting, edge flipping and automatic topological events resolution.

1. Introduction

City modeling has become a very important domain in the last decade, particularly with the democratization of 3D mapping, but also with the development of acquisition methods allowing to collect 3D data (Lidar point clouds, dense matching DSMs) on large areas with decreased costs.

The will to produce 3D models which are more accurate, and also more complex induce a need for more precise and detailed data, with a minimum production time.

The new data acquisition methods that have been developed lately have enabled city modeling to be performed on vast areas, making it a very important source of data for 3D urban mapping.

In 3D city modeling, the reconstruction of buildings from acquired 3D data (point clouds, DSMs, meshes) is a crucial operation, which can be very error prone, particularly when high fidelity and accuracy are required. Several tools have been developed to help creating 3D building models, some of them automatic, and others manual. Automatic methods are way faster and less expensive than manual ones, that require important human interaction, but they are also sensible to the data defects (occlusion, heterogeneous data...), and can even fail to reconstruct buildings which have a complex roof structure. On the other hand, manual methods are much more expensive than automatic methods on most buildings for a similar result, because they require the work of trained operators for an optimal result. But humans are able to understand complex structures, or to correct the defaults of the data, which makes manual methods more robust, producing higher quality models.

Manual and automatic methods have complementary advantages and drawbacks, so we believe it is interesting to explore combining them in such a way that it would guarantee a quality close to that of manual modeling, but lowering significantly production time and cost.

In our work, we propose a pipeline which would allow for such a combination (Figure 1). We propose to start from an automated reconstruction, and then propose ergonomic manual polyhedral editing tools to correct possible reconstruction errors. Our focus in this paper will thus be on the creation of a polyhedral modeler which would allow the edition of the geometry and topology of polyhedral building models.

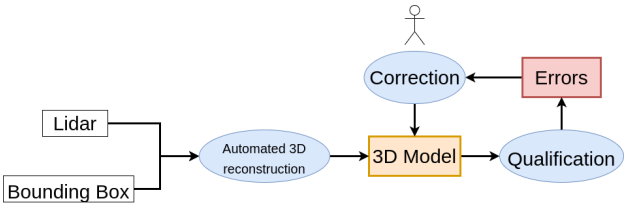


Figure 1. Overview of the proposed pipeline

2. Related work

2.1 3D models

We call 3D object the representation of an object of the real world. A 3D object contains a geometrical description of the real world object, and other information concerning this object.

In most of the models, this description contains some geometrical data (e.g. : coordinates of a point, coordinates of a plane...), and these geometrical data are structured by topological data, which describe how the different parts of the 3D object as combinations of some geometrical data (e.g. : a plane can be described as an ordered set of vertices, an edge can be described as a pair of vertices...).

In addition to the geometrical and topological data, it is possible to add semantic data about the object or its components. These are all the data that give information about the object or its components which are not about its geometry or its topology.
There exists different ways representing the geometry of 3D

This contribution has been peer-reviewed.
https://doi.org/10.5194/isprs-archives-XLVIII-2-W8-2024-169-2024

objects, which focus on different aspects of the 3D object.

First, raw data acquired from a real scene, such as Lidar or dense matching point clouds gives a 3D sampling of an object but does not provide information about its structure. Raw data is often the input data of 3D reconstruction methods which produce more structured representations, which we will present now.

A popular representation for 3D objects is meshes. They describe the geometry of an object with 3D points, its topology with cells, and potentially also contain some semantics attached to the cells that compose the surface. The cells that compose a mesh are planar polygons defined by ordered lists of coplanar 3D points. They can be triangles (triangle mesh), quadrilaterals (quad mesh)... or any combination of planar polygons (polyhedral mesh).

While the topology of meshed models is usually stored as ordered lists of 3D point indices, other data structures have been proposed for more efficient traversal and topological edition, such as half-edges models (Line Segment Intersection, 2008) or generalized maps (Lienhardt, 1991)(Belhaouari et al., 2014).

Finally, higher level data models have been proposed to store the geometry, topology, and semantics of 3D objects, such as the CityGML model implemented in the CityJSON file format (Ledoux et al., 2019), or the 3DCityDB database (Yao et al., 2018).

In the domain of the city modeling, objects are often represented as meshes (Caraffa et al., 2017), or polyhedrons (Huang et al., 2022), which are well adapted for building modeling. In this models, the geometrical data are often given as points coordinates, and the topological data describe the faces of the polyhedron as a set of borders, each border being a linear ring of vertices.

2.2 Automatic reconstruction

Among the various automated tools which are used during the reconstruction process, we can distinguish several possible outputs : point clouds, meshes (Caraffa et al., 2017), polyhedrons (Nan and Wonka, 2017, Bauchet and Lafarge, 2020, Huang et al., 2022), all with or without additional semantic data (surface type, LoD...). Most of this methods have an output with a different data type than the input data type (point cloud to mesh, point cloud to polyhedron...). But some of them add semantic data to their inputs, giving an output in the same data type than the input.

Some methods give geometrical and topological guaranties, like the watertightness (Caraffa et al., 2017), or the closure or manifoldness of the output (Nan and Wonka, 2017, Huang et al., 2022).

We can cite as an example the family of tools using plane arrangements (Nan and Wonka, 2017, Bauchet and Lafarge, 2020, Huang et al., 2022), which guaranty that their outputs are closed and borderless polyhedrons.

2.3 Manual edition of 3D models

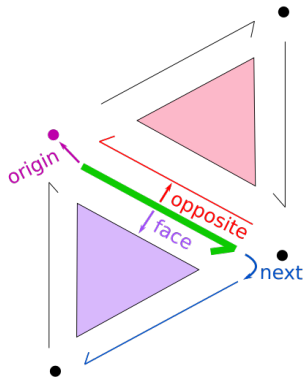
A lot of 3D modeling software already exist (among the most popular are Blender, Sketch up and ZBrush (Keller, 2011)). These modelers can be used either to create new 3D objects from scratch, or to modify existing ones.

The modelers which concentrate on the creation of new shapes use mainly triangle or quad meshes, with basic topological data which does not always correspond to the structure of the real object (Keller, 2011). Some other modelers, like Sketch up, use topological models which can handle more complex topology (like polygon mesh (Ying et al., 2011)), which allows them to have a better representation of the real structure of the objects. However they do not provide object edition tools which allow direct interaction with the topological model.

3. Methodology

3.1 3D model

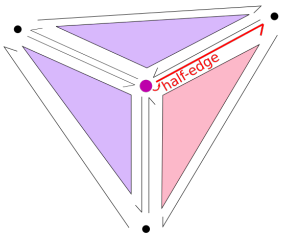
To model buildings as polyhedra, we use a 3D representation based on the half-edge data structure, and representing the geometry through plane equations of the faces of the model, instead of the more usual vertex coordinates. This has the advantage of ensuring that faces are planar by construction, but adds the constraint that more than 3 faces sharing a vertex must co-intersect. The half-edge data structure stores the polyhedron topology as faces, half-edges and vertices. Each edge of the polyhedron is broken down into two opposite half-edges, as shown in figure 2. Each half-edge points to an origin vertex (one of the vertices of the edge), a face (one of the two sharing the edge), an opposite half-edge (the other half-edge representing the edge and pointing to the other face), and a next half-edge (along the boundary of its face).



half-edge pointers	pointed data type
origin	vertex
opposite	half-edge
next	half-edge
face	face

Figure 2. Half-edge data in the half-edge data structure.

Each vertex points to only one of the half-edges having this vertex as origin (see figure 3). From this information, it is possible to find all the other half-edges having this vertex as origin (by traveling from an half-edge to the next of its opposite, and so on until getting back to the original half-edge).

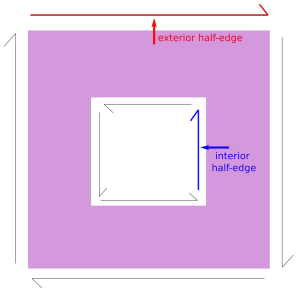


vertex pointers	pointed data type
half-edge	half-edge

Figure 3. Vertex data in the half-edge data structure.

Finally, each face points to only one of the half-edges that form its exterior boundary, and to $n \in \mathbb{N}$ half-edges, one for each interior boundary that the face contains (Figure 4).

Once again, with only one half-edge per interior boundary, it is easy to find the other half-edges belonging to this boundary by performing a loop beginning to the stored half-edge, and then traveling from one half-edge to its next one, until it gets back to the stored half-edge.



face pointers	pointed data type
Exterior half-edge	half-edge
Interior half-edges	half-edge[]

Figure 4. Face data in the half-edge data structure.

In our model, the 3D geometry is stored as the equations of the supporting planes of each face. With these geometrical data, it is possible to compute the 3D coordinates of each vertex adjacent to at least three faces which planes co-intersect.

Because our modeler is meant to modify existing 3D objects, it needs to be able to import other data types and to translate them into our 3D data model. Currently, the data that can be imported in our modeler are CityJSON data. We use an intermediate data structure between the CityJSON model and our final 3D model, which respects a subset of the CityGML norm (Groger et al., 2008) (the subset concerning the geometrical and semantic description of a building). So adding the possibility to import data respecting the same norm, such as 3DCityDB for instance (Yao et al., 2018), will be easy.

For easier use of the polyhedral modeler, it has been decided to define the validity of a 3D object as a set of topological and geometrical rules that all 3D objects should satisfy before and after edition. As our polyhedral modeler is meant for urban objects modeling, these rules are adapted to buildings.

This rules depict objects that must be manifold, closed and border-less. This implies that the objects are watertight and that they define a single volume.

The half-edge data structure gives us an easy way to ensure that the topology described is valid : a topology that would have a border would have at least one half-edge without opposite, and non-edge manifold objects cannot be represented with half-edges (it would require several opposites for one half-edge. For vertex manifoldness, we need to check that all half-edges are reachable from their origin vertex using the loop we described earlier. So we can propose the following rules, which ensure the validity of an half-edge structure defined by an half-edge set H, a vertex set V and a face set F:

- $\forall h \in H, \exists! h_o \in H$ such that $opposite(h) = h_o$ and $opposite(h_o) = h$
- Let us define the exterior orbit of a face $f \in F$ as $O^{ext}(f) = \{h \in H, \exists n \in \mathbb{N}, next^n(exteriorHalfEdge(f)) = h\}$, and let us define in the same way all the interior orbits of the face $(O_i^{int}(f))$.

  Then the exterior and interiors orbits are a partition of the set of the half-edges that belongs to the face f :

  $\forall f \in F, \forall O_1, O_2, O_1 \neq O_2 \in O^{ext}(f) \bigcup O_i^{int}(f)$, $O_1 \bigcap O_2 = \emptyset$, and $O^{ext}(f) \cup \bigcup_i O_i^{int}(f) = \{h \in H, face(h) = f\}$
- Let us define the orbit of a vertex $v \in V$ as $O(v) = \{h \in H, \exists n \in \mathbb{N}, (next \circ opposite)^n(halfEdge(v)) = h\}$.

  Then $\forall v \in V, O(v) = \{h \in H, origin(h) = v\}$

Validity also extends to geometry as a valid topology with invalid geometry could describe a self intersecting object. For instance, the object represented in figure 10 can be represented with a valid half-edge structure, but the auto intersections in two of its borders makes of this object a non manifold object. To avoid this, we propose different validity certificates for each edition operators, which ensure that the objects that are modified stay valid, so that the user can never accidentally create invalid building structures.

3.2 Operators

We have developed three edition operators in our polyhedral modeler, which we present in the following section.

In the figures where we present half-edge structures, some half-edges are named relative to another half-edge (denoted h here), and the name of the half-edge describes the operations needed to travel from h to this half-edge. For instance, h_{nop} is the (p)revious half-edge of the (o)pposite half-edge of the (n)ext half-edge of h.

In the tables presenting the changes of data during the application of the operators, we use a color code to represent the type of modification:

- the green lines, dots or faces represent the data of the selected elements
- the red lines, dots or faces represent the deleted elements
- the blue lines, dots or faces represent the created items

- the other lines, dots or faces (without color) represent the other elements

Also, in the tables, some data replacements are noted in the following way : $?\cancel{h_1} \rightarrow h_2$. This notation means that if the data is equal to h_1, we replace it by h_2, and we do nothing otherwise.

3.2.1 Face shifting The FACE-SHIFT operator allows the user to move a single face of a 3D object along its normal. For a face supported by a plane parameterized by (a, b, c, d), such movement can be performed by modifying its last plane parameter from (a, b, c, d) to $(a, b, c, d + \delta)$. Note that, as vertex positions are defined at the intersection of the planes supporting their adjacent faces, they will not move along the normal of the moving plane, but slide on the line supporting the intersection of 2 planes supporting other adjacent faces.

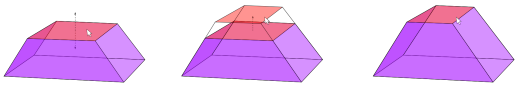


Figure 5. The FACE-SHIFT operator moves one face, but doesn't change the other faces. The coordinates of each vertex are computed as the intersection of its adjacent planes.

The value of the change, denoted as δ, is derived from a drag-and-drop user interaction in screen-space as follows (Figure 5):

- The mouse-down event selects the moving face by 3D picking. We define a 3D ray r_1 directed by the normal of the moving face and originating from the picked 3D point.
- Every mouse-move event defines a 3D ray r_2 from the viewing camera position in the direction of the pixel of the mouse location. Then we find the point p of r_1 that is the closest to r_2. The value δ is the distance between the picked 3D point of the moving face and the point p (Figure 6).

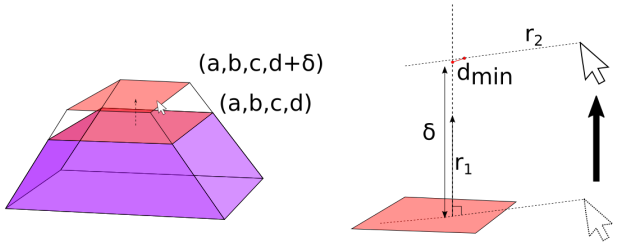


Figure 6. The value of the shift δ can be computed from the new position of the mouse.

3.2.2 Edge flipping The EDGE-FLIP operator modifies the topological definition of an edge, by changing the face adjacencies of the incident vertices (Figures 7 and 9). This is only possible if the vertices are not adjacent to more than three faces which supporting planes coordinates are linearly independent.

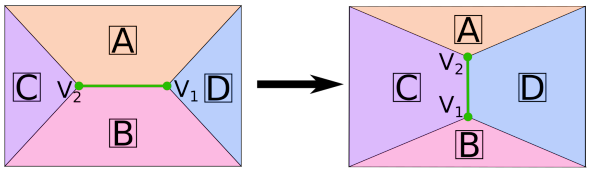
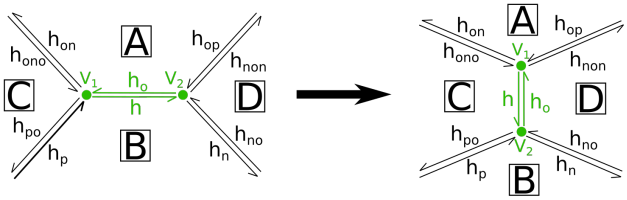


Figure 7. The edge defined as the boundary between the faces B and D is redefined as the frontier between the faces A and C.

This operator therefore changes only the half-edge data of the object. The geometrical data are then updated according to the new topology of the object. The changes in half-edge data are presented in figure 8.



half-edge	origin	next	opposite	face
h	V_1	$\cancel{h_n}\ h_{po}$	h_o	$\cancel{B}\ C$
h_o	V_2	$\cancel{h_{on}}\ h_{non}$	h	$\cancel{A}\ D$
h_p	...	$\cancel{h}\ h_n$	h_{po}	B
h_{op}	...	$\cancel{h_o}\ h_{on}$	h_{non}	A
h_{no}	...	$\cancel{h_{non}}\ h_o$	h_n	D
h_{ono}	...	$\cancel{h_{po}}\ h$	h_{on}	C
h_{po}	$\cancel{V_1}\ V_2$	...	h_p	C
h_{non}	$\cancel{V_2}\ V_1$	...	h_{op}	D

point label	half-edge
V_1	$?\cancel{h_{po}} \rightarrow h$
V_2	$?\cancel{h_{non}} \rightarrow h_o$

face label	exterior half-edge	interiors half-edges
A	$?\cancel{h_o} \rightarrow h_{on}$	$?\cancel{h_o} \rightarrow h_{on}$
B	$?\cancel{h} \rightarrow h_n$	$?\cancel{h} \rightarrow h_n$

Figure 8. The topological modifications that occur during the EDGE-FLIP of the edge defined by the half-edges h and h_o are shown here.

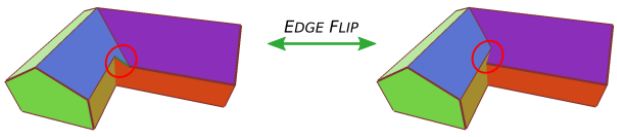


Figure 9. Result of an EDGE-FLIP on an edge of the roof of a house.

Although this operator can be applied on any edge having its vertices adjacent to only three faces with linearly independent supporting plane coordinates with a valid half-edge structure as a result, in most cases it will create auto-intersections on faces borders (see figure 10), which would not fit with our notion of validity of 3D objects. For this reason, we check before doing the flip that it will not create any auto-intersections, and we disable the EDGE-FLIP operator on edges that would yield an invalid result.

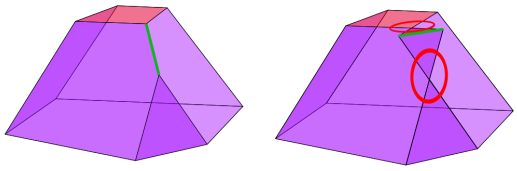


Figure 10. The EDGE-FLIP may create auto-intersections in the borders of adjacent faces.

3.2.3 Face creation from an edge or a vertex This operation transforms a vertex (Figure 11), or an edge (Figure 12), into a face. Once again, this operation only modifies the topological data of an object, but the plane equations supporting the faces are not modified.

Contrary to previous operators, this operator can be applied on vertices adjacent to more than three faces and on edges which vertices are adjacent to more than three faces. For this reason, we present the modification of the half-edge structure with an arbitrary number of faces adjacent to the vertex or to the edge (Figures 13 and 14).

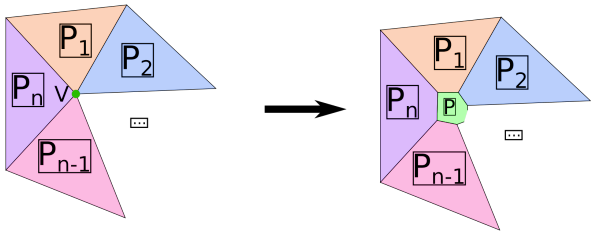


Figure 11. FACE-CREATION from a vertex with an arbitrary number of adjacent faces.

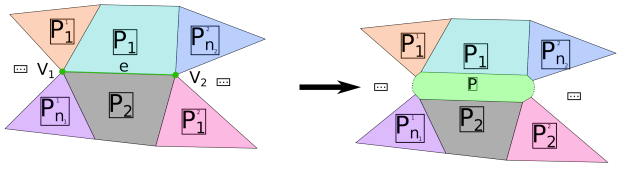
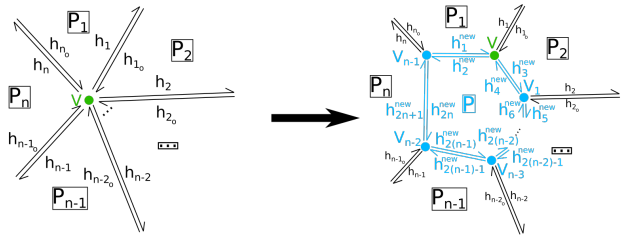


Figure 12. FACE-CREATION from an edge with an arbitrary number of adjacent faces on every vertex of the edge.



half-edge label	origin	next	opposite	face
$h_i, i \neq 1$	~~V~~ V_{i-1}	...	...	...
h_{i_o}	...	~~$h_{(i+1)\%n}$~~	...	...
	...	$h^{new}_{(2i+1)\%n}$	...	...
h^{new}_{2i-1}	$i \neq 2$ $V_{(i-2)\%n}$ $i = 2$ V	h_i	h^{new}_{2i}	P_i
h^{new}_{2i}	$i \neq 1$ $V_{(i-1)\%n}$ $i = 1$ V	$h^{new}_{2(i-1)\%2n}$	h^{new}_{2i-1}	P

point label	half-edge
V	$?\sout{h_i, i \neq 1} \rightarrow h_1$
V_i	h_{i+1}

face label	exterior half-edge	interiors half-edges
P	h^{new}_2	

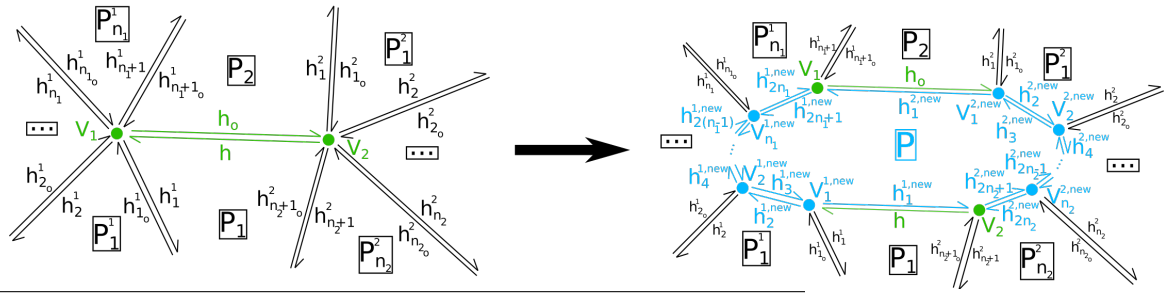
Figure 13. Half-edge structure changes during the FACE-CREATION from a vertex with an arbitrary number of adjacent faces. All the vertices indices are given for $i \in [\![1, n-1]\!]$ and all the half-edges indices are given for $i \in [\![1, n]\!]$, where n is the number of faces adjacent to the vertex V (which is the vertex being split).

3.3 Topological event resolution

We use our topological data structure combined to the geometrical data to detect the topological events created by the user's actions, and to modify automatically the topological structure of the objects according to the topological events. The face shift operator traces out the weighted straight skeleton (Aurenhammer and Walzl, 2016) of the input polyhedron with weights depending on the dihedral angles with neighboring planes with the moving plane, and this leads to topological events which needs to be resolved, by changing the topology.

Three event resolutions have been developed so far, and we present them in the following sections.

3.3.1 Edge collapsing This event resolution occurs when the length of the edge becomes zero, and it transforms the edge with vanishing length into a vertex. It is made by merging the two vertices of the edge (Figure 15).

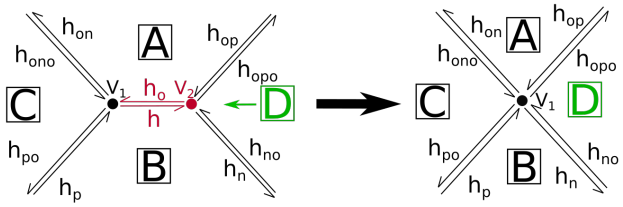


half-edge label	origin	next	opposite	face
h	...	...	$\cancel{h_o}\ h_1^{1,new}$	...
h_o	...	...	$\cancel{h}\ h_1^{2,new}$	...
h_i^1, $i \neq n_1+1$	$\cancel{V_1}\ V_i^{1,new}$	...	...	...
$h_{i_o}^1$, $i \neq n_1+1$	...	$\cancel{h_{i+1}^1}\ h_{i+1}^{1,new}$	...	...
h_j^2, $j \neq n_2+1$	$\cancel{V_2}\ V_j^{2,new}$	...	...	...
$h_{j_o}^1$, $j \neq n_2+1$	...	$\cancel{h_{j+1}^2}\ h_{j+1}^{2,new}$	...	...
$h_{2i}^{1,new}$, $i \neq n_1+1$	$V_i^{1,new}$	h_{i+1}^1	$h_{2i+1}^{1,new}$	P_i^1
$h_{2i-1}^{1,new}$	$V_i^{1,new}$	$\begin{cases} i \neq 1 & h_{2(i-1)-1}^{1,new} \\ i = 1 & h_{2n_2+1}^{2,new} \end{cases}$	$\begin{cases} i \neq 1 & h_{2*(i-1)}^{1,new} \\ i = 1 & h \end{cases}$	P
$h_{2j}^{2,new}$, $j \neq n_2+1$	$V_j^{2,new}$	h_{j+1}^1	$h_{2j+1}^{2,new}$	P_j^2
$h_{2j-1}^{2,new}$	$V_j^{2,new}$	$\begin{cases} j \neq 1 & h_{2(j-1)-1}^{2,new} \\ j = 1 & h_{2n_1+1}^{1,new} \end{cases}$	$\begin{cases} j \neq 1 & h_{2*(j-1)}^{2,new} \\ j = 1 & h \end{cases}$	P

point label	half-edge
V_1	$?\cancel{h_i^1} \rightarrow h_o$
V_2	$?\cancel{h_j^2} \rightarrow h$
$V_i^{1,new}$	h_i^1
$V_j^{2,new}$	h_j^2

face label	exterior h.-e.	interiors h.-e.
P	$h_1^{1,new}$	/

Figure 14. Changes during the FACE-CREATION from an edge with an arbitrary number of adjacent faces on every vertex of the edge. All the half-edges indices are given for $i \in [\![1, n_1]\!]$ and $j \in [\![2, n_2]\!]$, where n_2 is the number of faces adjacent to the vertex V_2 and not adjacent to the vertex V_1, and n_1 is the number of faces adjacent to the vertex V_1 and not adjacent to the vertex V_2.



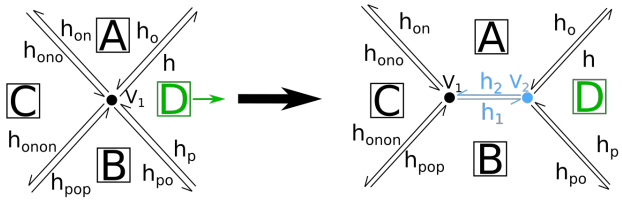
half-edge label	origin	next	opposite	face
h	V_1	h_n	h_o	B
h_o	V_2	h_{on}	h	A
h_p	...	$\cancel{h}\ h_n$	h_{po}	B
h_{op}	...	$\cancel{h_o}\ h_{on}$	h_{opo}	A
h_n	$\cancel{V_2}\ V_1$	...	h_{no}	B
h_{opo}	$\cancel{V_2}\ V_1$	...	h_{op}	D

point label	half-edge
V_1	$\cancel{h}\ h_n$
V_2	h_o

face label	exterior half-edge	interiors half-edges
A	$?\cancel{h_o} \rightarrow h_{on}$	$?\cancel{h_o} \rightarrow h_{on}$
B	$?\cancel{h} \rightarrow h_n$	$?\cancel{h} \rightarrow h_n$

Figure 15. Half-edge structure changes during the EDGE COLLAPSE resolution.

3.3.2 Vertex splitting This event resolution occurs during the shift of a face, when one of the vertices of this face is adjacent to more than three faces with linearly independent supporting planes coordinates. It transforms this vertex into an edge, by splitting the vertex into two vertices. This operation is the opposite of EDGE COLLAPSE.

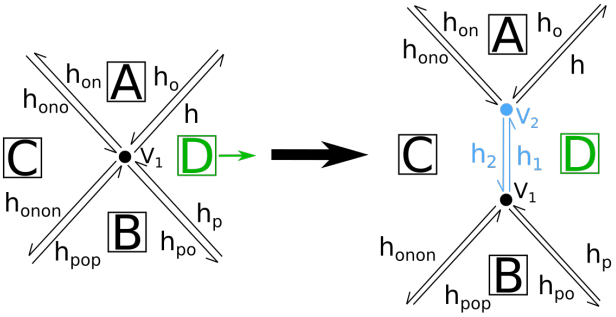


half-edge label	origin	next	opposite	face
h_1	V_1	h_{po}	h_2	B
h_2	V_2	h_{on}	h_1	A
h	$\cancel{V_1}\ V_2$	h_n	h_o	D
h_o	...	$\cancel{h_{on}}\ h_2$	h	A
h_{po}	$\cancel{V_1}\ V_2$	...	h_p	B
h_{pop}	...	$\cancel{h_{po}}\ h_1$	h_{onon}	B

point label	half-edge
V_1	$?\cancel{h} \vee \cancel{h_{po}} \rightarrow h_1$
V_2	h_2

Figure 16. General half-edge structure modification during VERTEX SPLIT.

While there exists only one way to split a vertex into one edge for a vertex adjacent to at least five faces with linearly independent supporting planes coordinates, in the case of a vertex adjacent to four faces with linearly independent supporting planes coordinates, there exist two ways to split the vertex into one edge. The first one, which is the way that works also for vertices having more adjacent faces, is presented in figure 16, and the other one, which is specific to this case of the vertices adjacent to four faces, is presented in figure 17.



half-edge label	origin	next	opposite	face
h_1	V_1	h	h_2	D
h_2	V_2	h_{onon}	h_1	C
h	~~V_1~~ V_2	h_n	h_o	D
h_{ono}	...	~~h_{onon}~~ h_2	h_{on}	C
h_{on}	~~V_1~~ V_2	...	h_ono	A
h_p	...	~~h~~ h_1	h_{po}	D

point label	half-edge
V_1	$?\sim\!\!h \vee \sim\!\!h_{on} \rightarrow h_1$
V_2	h_2

Figure 17. Specific half-edge structure modification during VERTEX SPLIT for a vertex adjacent to four vertices.

3.3.3 Face collapsing This event resolution occurs when the area of a face becomes null. But an easier way to detect these events is to detect all the triangle faces which have on edge that is collapsing (see figure 18). This resolution is the same as the EDGE COLLAPSE, but with a face becoming an edge. We present the half-edge structure modification of this event resolution in Figure 19.

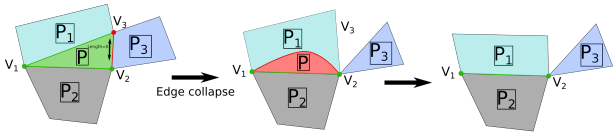
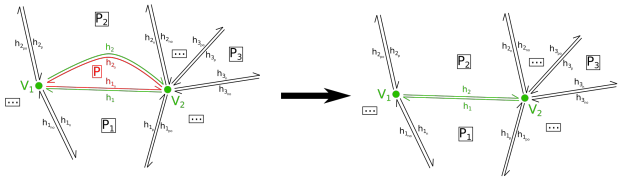


Figure 18. The FACE-COLLAPSE always happens after the resolution EDGE COLLAPSE occurs on an edge of a triangle face, reducing this face to one segment.



half-edge label	origin	next	opposite	face
h_{1_o}	V_1	h_{2_o}	h_1	P
h_{2_o}	V_2	h_{1_o}	h_2	P
h_1	...	...	~~h_{1_o}~~ h_2	...
h_2	...	...	~~h_{2_o}~~ h_1	...

point label	half-edge
V_1	$?\sim\!\!h_{1_o} \rightarrow h_2$
V_2	$?\sim\!\!h_{2_o} \rightarrow h_1$

face label	exterior half-edge	interiors half-edges
P	...	...

Figure 19. Changes of the half-edge data structure during a FACE-COLLAPSE.

4. Results

The resulting modeler is able to perform several modifications to both the topology and the geometry of the objects (Figure 20). Each modification applied on a 3D object leaves the object in a valid state : the object remains manifold, closed, and borderless.

The code of the modeler is open source and is provided with a test dataset [1] and an online demo is available [2].

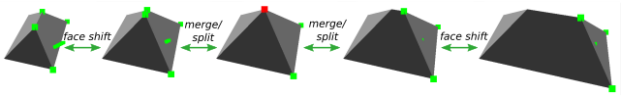


Figure 20. Most of the user geometrical modifications actions are based on face's support plane modifications.

The loading process which creates a new object using our data structure can be time consuming for complex buildings (about ten seconds for the most complex ones), which makes it difficult to load entire areas. So we use an intermediate data structure to visualize entire areas, and then when the user select a building for the first time, the loading process is done and the object using our data structure is created (see figure 21).

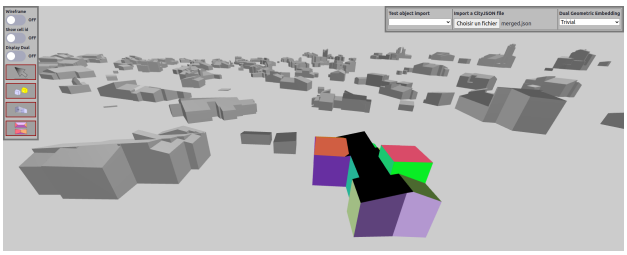


Figure 21. It is possible to deal with large amounts of buildings, using intermediate data structure.

1 https://github.com/LelouchLiBritania/3D-Viewer.git
2 https://lelouchlibritania.github.io/3D-Viewer/

However, there exists some input data that the loading process cannot handle. In our tests, around 5% of the buildings failed to be loaded (26 fails for a total of 486 buildings). These failures are caused by some particular cases that have not been handled yet, like vertices which would be adjacent to only two faces, which makes impossible the calculation of their coordinates with plane equation intersections, or like adjacent faces which would share the same supporting plane. All these issues should be resolved in our future works.

5. Conclusion

We presented here a work done in the context of the creation of a semi automatic polyhedral modeling pipeline which combines the speed of automated reconstruction methods with the robustness of the manual modeling. In this pipeline, most of the components can be taken off the shelf, but we decided to create our own polyhedral modeler in order to make it easier for the user to interact with the objects. This improvement of the polyhedral modeling part is mostly done by the use of a half-edge data structure to represent topology and plane equations for geometry, and also with the proposition of rules defining a notion of validity for the objects, which can then be used to prevent the user from breaking this validity.

The operators that have been developed so far are the FACE-SHIFT, the EDGE-FLIP and the FACE-CREATION operators. In addition to the FACE-SHIFT operator, we have added the possibility to modify the topological structure of the object when topological events occurs. The events resolution that have been implemented so far are the edge collapsing, the vertex splitting, and the face collapsing.

Future works will tackle other components of the pipeline, such as an object importer assessing and ensuring validity of imported objects , to facilitate correction of large amount of automatically reconstructed objects.

Concerning the proposed modeler, our perspective is to implement other useful operators (face rotation, face cut...), to have it tested by real users and to enhance it based on their feedback. A user study is therefore planned to assess its usability and efficiency.

References

Aurenhammer, F., Walzl, G., 2016. Straight Skeletons and Mitered Offsets of Nonconvex Polytopes. *Discrete & Computational Geometry*, 56(3), 743–801. doi.org/10.1007/s00454-016-9811-5.

Bauchet, J.-P., Lafarge, F., 2020. Kinetic Shape Reconstruction. *ACM Transactions on Graphics*. doi.org/10.1145/3376918.

Belhaouari, H., Arnould, A., Le Gall, P., Bellet, Thomas", e. H., König, B., 2014. Jerboa: A Graph Transformation Library for Topology-Based Geometric Modeling. *Graph Transformation*, 8571, Springer International Publishing, Cham, 269–284. Series Title: Lecture Notes in Computer Science.

Caraffa, L., Brédif, M., Vallet, B., 2017. 3D Watertight Mesh Generation with Uncertainties from Ubiquitous Data. S.-H. Lai, V. Lepetit, K. Nishino, Y. Sato (eds), *Computer Vision – ACCV 2016*, 10114, Springer International Publishing, Cham, 377–391. Series Title: Lecture Notes in Computer Science.

Groger, G., Kolbe, T. H., Czerwinski, A., Nagel, C., 2008. OpenGIS® City Geography Markup Language (CityGML) Encoding Standard. Version 1.0.0. Report, Open Geospatial Consortium. Accepted: 2019-10-28T20:43:28Z.

Huang, J., Stoter, J., Peters, R., Nan, L., 2022. City3D: Large-Scale Building Reconstruction from Airborne LiDAR Point Clouds. *Remote Sensing*, 14(9), 2254. doi.org/10.3390/rs14092254. Number: 9 Publisher: Multidisciplinary Digital Publishing Institute.

Keller, E., 2011. *Introducing ZBrush 4*. John Wiley & Sons.

Ledoux, H., Arroyo Ohori, K., Kumar, K., Dukai, B., Labetski, A., Vitalis, S., 2019. CityJSON: a compact and easy-to-use encoding of the CityGML data model. *Open Geospatial Data, Software and Standards*, 4(1), 4. doi.org/10.1186/s40965-019-0064-0.

Lienhardt, P., 1991. Topological models for boundary representation: a comparison with n-dimensional generalized maps. *Computer-Aided Design*, 23(1), 59–82. doi.org/10.1016/0010-4485(91)90082-8.

Line Segment Intersection, 2008. M. de Berg, O. Cheong, M. van Kreveld, M. Overmars (eds), *Computational Geometry: Algorithms and Applications*, Springer, Berlin, Heidelberg, 19–43.

Nan, L., Wonka, P., 2017. PolyFit: Polygonal Surface Reconstruction from Point Clouds. *2017 IEEE International Conference on Computer Vision (ICCV)*, IEEE, Venice, 2372–2380.

Yao, Z., Nagel, C., Kunde, F., Hudra, G., Willkomm, P., Donaubauer, A., Adolphi, T., Kolbe, T., 2018. 3DCityDB - a 3D geodatabase solution for the management, analysis, and visualization of semantic 3D city models based on CityGML. *Open Geospatial Data, Software and Standards*, 3. doi.org/10.1186/s40965-018-0046-7.

Ying, S., Li, L., Guo, R., 2011. Building 3D cadastral system based on 2D survey plans with SketchUp. *Geo-spatial Information Science*, 14(2), 129–136. doi.org/10.1007/s11806-011-0483-2. Publisher: Taylor & Francis _eprint: doi.org/10.1007/s11806-011-0483-2.