Filling holes in LoD2 building models

Weixiao Gao[1],[*] Ravi Peters[2], Hugo Ledoux[1], Jantien Stoter[1]

[1] Delft University of Technology, The Netherlands - (w.gao-1,h.ledoux,j.e.stoter)@tudelft.nl
[2] 3DGI, Zoetermeer, The Netherlands - ravi.peters@3dgi.nl

Keywords: Mesh Hole Filling, LoD2 Building Models, Repair of 3D Models, CityGML.

Abstract

This paper presents a new algorithm for filling holes in Level of Detail 2 (LoD2) building mesh models, addressing the challenges posed by geometric inaccuracies and topological errors. Unlike traditional methods that often alter the original geometric structure or impose stringent input requirements, our approach preserves the integrity of the original model while effectively managing a range of topological errors. The algorithm operates in three distinct phases: (1) pre-processing, which addresses topological errors and identifies pseudo-holes; (2) detecting and extracting complete border rings of holes; and (3) remeshing, aimed at reconstructing the complete geometric surface. Our method demonstrates superior performance compared to related work in filling holes in building mesh models, achieving both uniform local geometry around the holes and structural completeness. Comparative experiments with established methods demonstrate our algorithm's effectiveness in delivering more complete and geometrically consistent hole-filling results, albeit with a slight trade-off in efficiency. The paper also identifies challenges in handling certain complex scenarios and outlines future directions for research, including the pursuit of a comprehensive repair goal for LoD2 models to achieve watertight 2-manifold models with correctly oriented normals. Our source code is available at `https://github.com/tudelft3d/Automatic-Repair-of-LoD2-Building-Models.git`.

1. Introduction

As geographic information system and computer graphics technologies continue to advance, an increasing number of sophisticated 3D city models are being generated (Döllner et al., 2006, Helsinki, 2019, Peters et al., 2022). These models play a crucial role in enhancing urban visualization (Mao and Ban, 2011, Zhang et al., 2014), aiding urban planning (Murata, 2004, Ran, 2011), and providing a robust platform for analyzing essential urban data (Geiger et al., 2013). The building models in these 3D city models are particularly significant, as they capture the fundamental aspects of a city's planning, construction, and development (Biljecki et al., 2015).

In the realm of 3D building models, Building Information Modeling (BIM) is often employed for detailed building management (Kolbe and Donaubauer, 2021). BIM models are characterized by their intricate semantic and geometric details, offering a high degree of precision and information. Concurrently, the construction of building models in large scale 3D urban environments typically adheres to the standards set forth by CityGML models (Kolbe et al., 2005). The Level of Detail 2 (LoD2) building model is notably prevalent in this area. LoD2 models provide a more comprehensive geometric representation of buildings, particularly in terms of roof structures, when compared to their LoD1 counterparts which have only flat horizontal roofs. Furthermore, LoD2 models strike a balance between detail and ease of construction, making them a more practical choice for large-scale urban building modeling than the more complex LoD3 models.

LoD2 building models are depicted as polygon soups or triangular meshes, created via automatic or (semi-)manual methods. Automatic modeling relies on input data quality for accuracy (Peters et al., 2022) and aims for watertight, manifold outcomes with geometric constraints (Huang et al., 2022).

However, inaccuracies may arise from incomplete data, as shown in Figure 1(a). In contrast, (semi-)manual modeling, integrating GIS, imagery, and elevation, offers higher geometric accuracy (see Figure 1(b)). Yet, software limitations, modeler expertise, or data conversion errors can introduce geometric and topological errors, including duplicates, non-manifolds, and holes (Wagner et al., 2012). For applications like fluid simulations, energy estimation, and geometric analyses, error-free, watertight 3D models are crucial (Ledoux, 2013, Biljecki et al., 2016). This research focuses on hole completion in LoD2 building models.
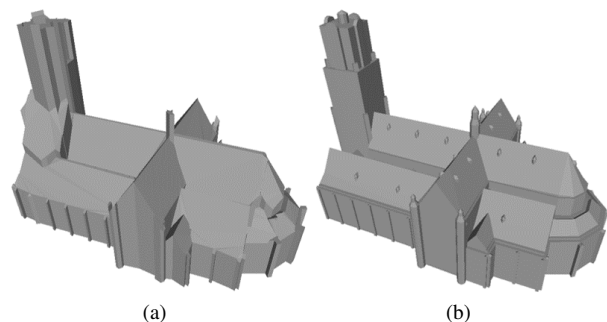

(a)   (b)

Figure 1. (a) displays a Lod2 building model reconstructed automatically, while (b) presents a manually reconstructed LoD2 building model.

A significant number of LoD2 city models are currently plagued with varying degrees of geometric and topological errors (Wagner et al., 2012, Biljecki et al., 2016). Addressing these issues manually would be an arduous and resource-intensive task. Consequently, there is a pressing demand for automated methods capable of both detecting and precisely rectifying these errors. Prior research has yielded encouraging outcomes in identifying anomalies in 3D urban models (Ledoux, 2013, Ledoux, 2018). In terms of repairing

---

building models, existing research predominantly falls into two categories: reconstruction-based repair and topological-based local repair.

Reconstruction-based methods can ensure watertight models to a degree, but often alter the original geometry and rely on preconditions like correct normals or semantic labels, posing challenges for inherently deficient models (Sindram et al., 2016, Zhao et al., 2018, Yu et al., 2022). Our approach avoids such strict input requirements and instead focuses on hole detection and repair within the model, preserving geometric integrity. Conversely, traditional topological-based local repairs, arising from computer graphics research, assume 2-manifold inputs with correct normals, utilizing half-edge data structures for hole boundary detection and local geometry for triangulation (Liepa, 2003, Zhao et al., 2007). However, urban building models, which may have non-manifolds, duplicate vertices, or misoriented normals (Wagner et al., 2012) as shown in Figure 2, present difficulties in accurately identifying holes topologically, leading to potential misrepairs. These conventional methods are thus not universally applicable to all LoD2 building models.
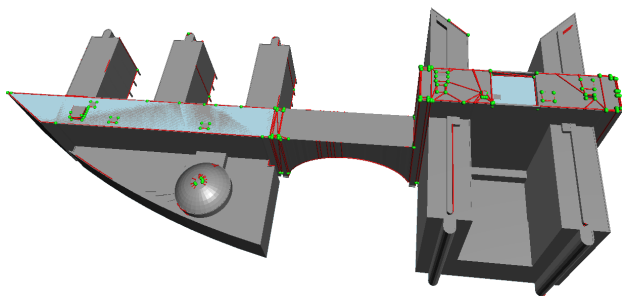

Figure 2. The LoD2 building mesh exhibits several errors, including non-manifold vertices highlighted in green, faces with incorrect normals indicated by the back faces of the mesh depicted in light blue, and boundaries of holes marked in red.

To address this gap, this paper introduces an innovative approach leveraging the continuity of mesh edges and the formation of virtual faces. This method effectively detects and fills holes in LoD2 building models without imposing stringent requirements on the input data, offering a versatile and adaptable solution for urban model repair.

2. Related work

In reviewing existing literature, we have identified two primary categories of urban model repair techniques: reconstruction-based repair and topological-based local repair. Our research is particularly focused on strategies that effectively address the completion of holes in mesh models.

2.1 Reconstruction-based repair

Reconstruction-based repair of mesh models typically involves using flawed models as input. And it is aiming to rebuild a complete mesh model by creating voxels or polyhedron to extract surface details and topological structures. These methods have their merits in restoring some missing or incorrect geometric structures, but they are not without limitations. The voxel-based approach (Mulder, 2015, Sindram et al., 2016), for instance, is limited by factors such as resolution, accuracy, computational demands, and a tendency

to either introduce extraneous geometric structures or omit fine details (Mulder, 2015). Polyhedron-based methods often stipulate specific conditions for the input model, such as continuous normal vectors, correct mesh topology (Zhao et al., 2018), while allowing for the absence of certain geometric structures (like parts of a wall), and the inclusion of semantic labels (Yu et al., 2022). Moreover, the reconstruction process may inadvertently add unnecessary geometric elements. Although capable of filling in missing information, these methods may alter the model's original geometric structure, either by introducing redundant elements that were not present originally or by omitting crucial components. Our proposed method, however, directly targets the areas with holes in building models without modifying the original structure and imposes no stringent requirements on the input model.

2.2 Topological-based local repair

Local repair methods that leverage mesh topology characteristics typically employ the half-edge data structure for detecting edges of holes (Liepa, 2003, Zhao et al., 2007). They then reconstruct the triangular mesh surface of the holes based on the provided geometric features of these edges. Subsequent post-processing, which includes smoothing and vertex position optimization, is applied to achieve uniformity in the local geometry. Nevertheless, these techniques necessitate that the input mesh is 2-manifold and struggle with larger holes. Moreover, the reconstruction of sharp geometric features is limited only to the local regions at the edges of the holes. In contrast, the method we introduce in this paper is adept at managing non-manifold mesh models and larger hole areas, enabling the restoration of sharp features throughout the entirety of the holes.

3. Methodology

Our aim is to fill holes in LoD2 building mesh models. It should be emphasized that the holes being repaired here specifically refer to geometric holes in the model, rather than topological holes or discontinuous gaps. The input for our process comprises piecewise planar LoD2 building mesh models. We approach the input data without pre-established conditions or limitations regarding its geometric attributes, topological characteristics, or semantic information. Upon identifying data with holes, we initiate a comprehensive hole-filling procedure. As shown in Figure 3, our methodology unfolds in three phases: pre-processing, hole detection, and remeshing.
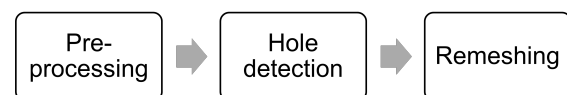

Figure 3. Pipeline for hole filling in building mesh models.

3.1 Pre-processing

The objective of pre-processing is to conduct a targeted repair on triangle meshes exhibiting topological errors, and to systematically identify and mark vertices and edges with topological inaccuracies. This preparation is crucial for the efficient detection and handling of holes in subsequent stages.

The process initiates with the identification and resolution of self-intersecting faces within the triangle mesh. We employ remeshing in these areas to rectify the half-edge data structure. Importantly, as the newly generated vertices during remeshing are confined within the planes of the triangles, the local geometric structure of the mesh remains unaltered.

Subsequently, we concentrate on addressing invalid pseudo-holes with tightly attached boundaries through a detailed stitching operation. Pseudo-holes, which are openings that resemble gaps without signifying real separations in the mesh, typically emerge from issues such as duplicated vertices, overlapping edges, and non-manifold edges. In contrast, true holes represent valid gaps with actual separations. Both true and pseudo-holes are geometric holes. Our current focus is specifically on pseudo-holes resulting from duplicated vertices (see Figure 4). The stitching process, leveraging the half-edge data structure, involves aligning and merging the edges where the source and target vertices of one half-edge (h1) correspond with the target and source vertices of another (h2).
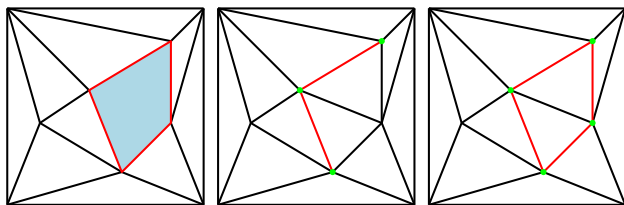

Figure 4. A 2D illustration depicts true and pseudo-holes in a 3D mesh. The left figure shows a true hole, while the middle and right display pseudo-holes. Red edges outline the hole's boundaries, and the light blue area indicates the real gap. Vertices in green highlight duplicate presence.

The final step involves the detection and categorization of duplicate vertices and overlapping edges. This identification is critical for the effective management of these anomalies during hole detection. To address duplicate vertices, we construct a kd-tree encompassing all mesh vertices, which allows us to efficiently locate and mark close vertices within a specified threshold range. In the case of overlapping edges, our aim is to pinpoint and mark three specific edges: degenerate edges, edges that share the same endpoints, and edges that, while being collinear, have distinct endpoints.
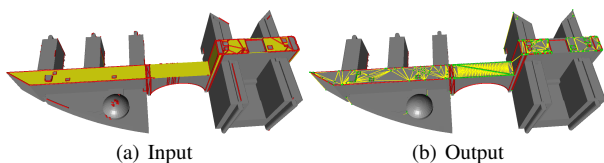

(a) Input   (b) Output

Figure 5. The left figure illustrates the input mesh with self-intersecting faces in yellow and the boundaries of true and pseudo-holes in red. The right figure displays the output mesh, highlighting duplicated vertices in green, duplicated edges in yellow, and maintaining the remaining hole boundaries in red.

Through this pre-processing phase, we effectively eliminate errors arising from self-intersecting faces and closely attached borders of pseudo-holes (see Figure 5(a)). Additionally, we mark vertices and edges that exhibit duplication within the output mesh data (see Figure 5(b)). This preparatory work lays the groundwork by providing essential topological insights, setting the stage for the forthcoming hole detection process.

3.2 Hole detection

The objective of this step is to accurately identify all true holes, distinguishing them from pseudo-holes arising due to overlapping or non-manifold edges (see Figure 6(a) to 6(c)). To accomplish this, we assemble triangles using adjacent half-edges and conduct intersection tests with every incident face of their edge vertices. This process is essential for choosing adjacent half-edges that pass intersection tests as candidates for true holes. Subsequently, we complement these selected candidates and extract their complete, closed border rings.

Algorithm 1 Hole Detection Algorithm

1: **for** each halfedge h in mesh **do**
2:     **if** h is not visited and is a valid border halfedge **then**
3:         Mark h as visited and use h as seed halfedge h_s
4:         **repeat**
5:             **if** preceding halfedge h_p exists and valid **then**
6:                 Collect all incident facets from h and h_p vertices
7:                 Create a triangle α from h and h_p
8:                 **for** each face β in all incident facets **do**
9:                     Compute maximum distance D from triangle α to β (Eq: 1)
10:                     Compute area ratio A from area A_β of β and intersected area A_w between projected α and β (Eq: 2)
11:                     **if** $D < \epsilon_d$ and $\frac{A_w}{A_\beta} > \epsilon_t$ **then**
12:                         α intersects with β
13:                         **break**
14:                     **else**
15:                         α does not intersect with β
16:                     **end if**
17:                 **end for**
18:                 **if** α does not intersect with any β **then**
19:                     Collect h_p and h to current border ring
20:                 **end if**
21:             **end if**
22:             Move h to next halfedge
23:         **until** h is not equal to seed halfedge h_s
24:         **if** current border ring is not closed **then**
25:             Find missing edges of current border ring
26:         **end if**
27:         Reorder hole vertices
28:     **end if**
29: **end for**

As outlined in Algorithm 1, our approach begins by traversing potential border edges using the half-edge data structure, marking each traversed half-edge. Our focus is on potential border rings initiated from a seed half-edge. Each half-edge, upon traversal, is paired with its preceding half-edge to form a triangle, α. This triangle α is projected onto a plane defined by the incident faces of the edge's vertices, where it undergoes intersection tests with these faces.

Figure 7 illustrates this process. The seed half-edge v_1v_2 and its preceding half-edge v_0v_1, marked in red, constitute the virtual triangle α. Meanwhile, triangle β (highlighted in green and defined by $v_1v_2v_3$) is an incident face of the vertices of half-edge v_1v_2. The projection of α onto β
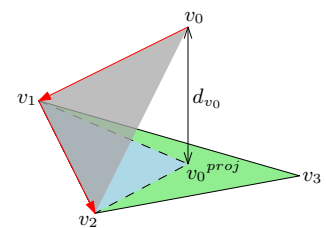

Figure 7. An example of the intersection test.

results in an intersecting area, shown in blue. The maximum distance D from α to β is equal d_{v_0} in this example, indicating

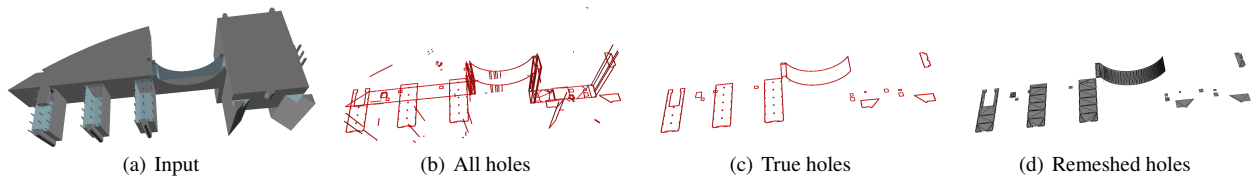(a) Input   (b) All holes   (c) True holes   (d) Remeshed holes

Figure 6. Visualization of intermediate stages in the proposed hole filling process.

the distance between vertex v_0 and its projected counterpart v_0^{proj}. The subsequent equation provides a detailed computation:

$$D = \max(d_{v_0}, d_{v_1}, d_{v_2}) \tag{1}$$

$$A = \frac{A_w}{A_\beta} \tag{2}$$

In Equation 1, $d_{v_0}, d_{v_1}, d_{v_2}$ denote the distances from the three vertices of triangle α to the plane of the incident triangle face β. D is the maximum of these distances. It should be noted that d_{v_1} and d_{v_2}, as shown in Figure 7, can be non-zero when they are identified as duplicated vertices. As indicated in Equation 2, α is then projected onto β's plane, where A_w signifies the area of intersection between the projected α and β, A_β is the area of β, and A indicates the ratio of these areas. An intersection is deemed valid only when $D < \epsilon_d = 0.1$ and $A > \epsilon_t = 0.01$. It should be emphasized that the thresholds for distance and area specified in this paper are measured in meters, and their settings are based on the geometric precision of the test data. In cases where intersections are absent, this pair of edges is incorporated into the border ring collection for the candidate holes.

Next, we conduct closure checks for each border ring, that is, to ascertain whether their start and end points connect. This verification is crucial because meshes plagued with overlapping (see Figure 8(a)) or non-manifold edges (see Figure 8(b)) can result in border rings that do not close properly. To tackle these issues, we have implemented two strategies. The first addresses overlapping edges: we examine all incident edges of the end vertices of the border ring, including all corresponding duplicated vertices. If an untraversed incident edge is a border edge or overlaps with an edge of the hole, it is added to the border ring collection. The second strategy, concerning non-manifold edges, involves assessing whether all connected non-border ring vertices of the current hole occupy the same plane. If affirmative, the vertex nearest to the border ring's endpoint that meets the intersection test criteria is added to the collection. These strategies are applied in an iterative manner until a fully closed ring is achieved.

Finally, we undertake a reordering of all selected candidate edges. This reordering ensures that adjacent edges are properly linked, creating a continuous and coherent structure. By employing this approach, we ensure that the successfully extracted closed rings of the holes are correctly configured for their integration in the forthcoming stage of remeshing.

3.3 Remeshing

The aim of this step is to execute triangle mesh construction, utilizing the closed rings identified in the hole detection process (see Figure 6(d)). This is essential for accurately filling in the mesh's missing areas.

We begin by fitting a plane with all the vertices of each identified closed hole ring and project these vertices onto this plane. Utilizing the border edges of the hole as constraints, we then create a 2D constrained Delaunay triangulation from these projected vertices. The subsequent step involves incorporating the triangulated mesh into our input data, aligning it with the corresponding original vertices of the projected points.

To ensure topological accuracy and avoid issues arising from recovering the 3D structure from the Delaunay triangulation, we conduct degeneracy detection and self-intersection checks for each triangle face to be added. Only faces that meet the specified conditions are ultimately included. Thus, through remeshing, we have obtained a complete building mesh model.

4. Experiment

4.1 Implementation details

Our testing environment is powered by a computer configured with an Intel Core i7-7700HQ CPU, operating at a base frequency of 2.8GHz, and bolstered by 32GB of RAM.

We developed our algorithm in C++, leveraging the capabilities of the Easy3D (Nan, 2021) library. This library facilitated our handling of triangular mesh read/write operations, manipulation of mesh data structures, and enabled efficient topological detection and validation. For the implementation of constrained Delaunay triangulation and edge-stitching processes, we employed the robust CGAL 5.6 library (Fabri et al., 2000).

In our comparative experimental assessments, we selected six characteristic models, each featuring complex holes, from The Hague's maunally created CityGML LoD2 building models (DenHaag, 2022). These models underwent a conversion process from CityGML (polygonal surface meshes) to OBJ format (triangular meshes) using FME software, specifically for conducting our advanced hole-filling operations.

4.2 Comparisons

	MeshL.	PMP	MeshF.	Polym.	Ours
Time (s)	0.76	2.12	1.24	1.13	16.19
Face num.	12.3k	16.2k	7k	158k	15.5k

Table 1. Comparative analysis of hole filling methods: Evaluating time efficiency and total output face count (Face Num.) across six chosen building meshes (incorporating 11.3k input faces) utilizing MeshLab (MeshL.), PMP, MeshFix (MeshF.), PolyMender (PolyM.), and our method.

We carried out a comprehensive qualitative analysis and a performance comparison between our mesh hole-filling method and other established methods, such as PMP (Sieger and
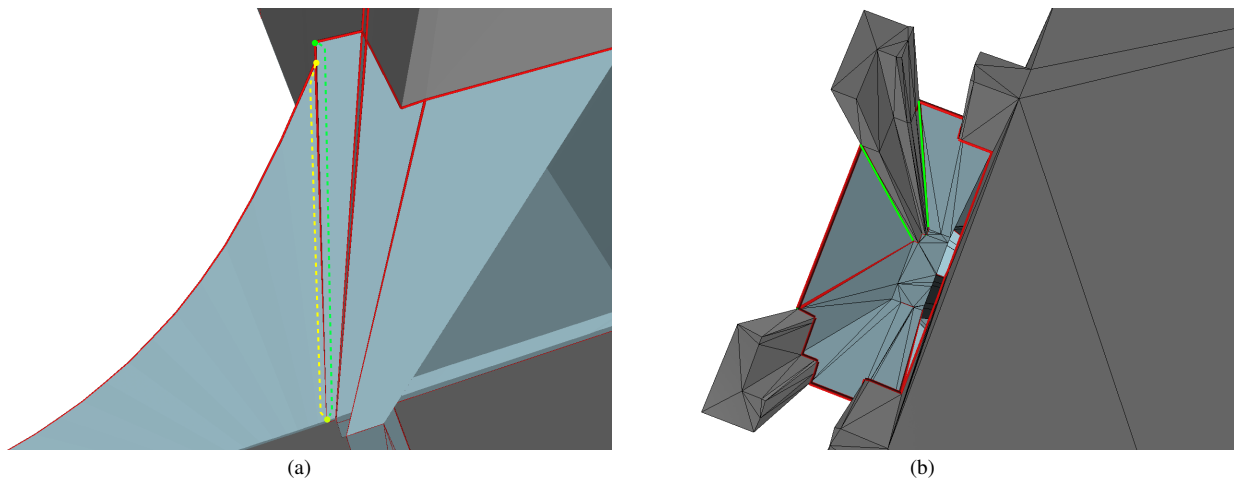
(a)   (b)

Figure 8. (a) illustrates a hole defined by two overlapping edges that originate from distinct vertices but converge at duplicated vertices, each edge outlined by a yellow dashed curve and a green dashed curve, respectively. (b) illustrates a hole featuring non-manifold edges, highlighted in green. The border of the hole is marked in red, while the mesh's back face is represented in light blue in both images.

(a) Input   (b) PMP   (c) MeshFix   (d) MeshLab   (e) Polymender   (f) Ours
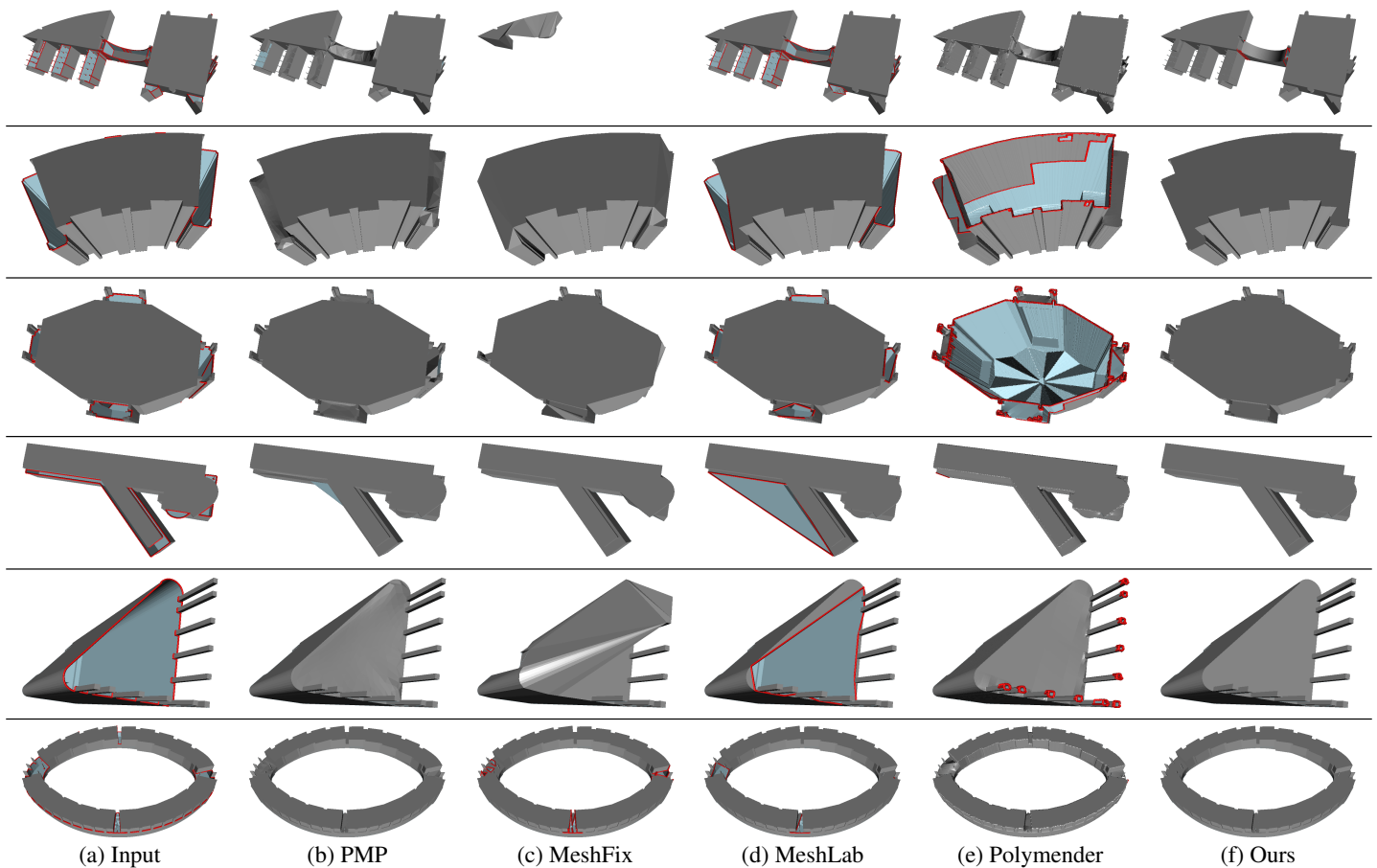
Figure 9. Qualitative comparisons of hole-filling methods in building mesh models. The boundaries of the holes are highlighted in red, while the back face of the mesh is depicted in light blue.

Botsch, 2019), MeshFix (Attene, 2010), MeshLab (Cignoni et al., 2008), and Polymender (Ju, 2004). The experimental results demonstrate that our method excels in completing LoD2 building models, surpassing all traditional methods on the quality of the output mesh, at the cost of an admittedly longer running time.

As illustrated in Figure 9, the PMP method, while successfully filling all holes, but generates erroneous surfaces due to the presence of pseudo-holes (as highlighted in rows 1 and 4 of the second column in Figure 9). Additionally, PMP's surface fairing post-processing step fails to ensure completely flat surfaces. Conversely, MeshFix alters the original geometric structure of the input data because of non-manifolds,

pseudo-holes, and disconnected components, resulting in the loss of specific geometric structures in its results. In contrast, MeshLab's hole-filling approach maintains the geometric structure of the initial data but falls short in terms of completeness, leaving some holes unfilled. Polymender, employing a reconstruction-based algorithm, over-smooths sharp geometric features of the original input data and in some cases introduces new holes (as depicted in rows 2 and 3 of the fifth column in Figure 9). Our method distinguishes itself by preserving the uniformity of local geometry at the holes and by upholding structural completeness. Moreover, our method showcases remarkable adaptability in processing input data with topological inaccuracies, while also providing improved stability in its operation.

Table 1 reveals that our algorithm's running time is longer than other methods, largely due to the additional time required for addressing topological errors such as non-manifolds, duplicate vertices, and overlapping edges. Notably, given a total number of 11.3k triangle faces in the input data, the increased number of triangles with our method is less than PMP. This benefit primarily stems from our method's ability to discern between true and pseudo-holes. Significantly, Polymender, a method based on reconstruction, produces an excessively large number of triangles to minimize geometric differences with the original data, which runs counter to the objective of preserving the LoD2 building model's simplicity. MeshFix, on the other hand, results in a loss of too many triangles, yielding a more incomplete final result.

To conclude, despite a longer running time, our method excels over others in terms of delivering complete and geometrically uniform hole-filling results.
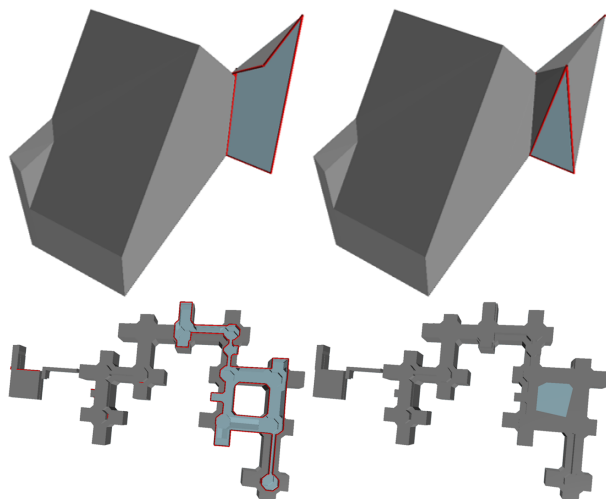
4.3 Limitations and future prospects

Figure 10. Failure cases of our algorithm. The boundaries of the holes are highlighted in red, while the back face of the mesh is depicted in light blue.

While our algorithm effectively addresses most topological errors in the input data and reconstructs the complete geometric surface of building mesh holes, it encounters limitations in three specific scenarios: Firstly, in cases where the hole of the input model has significantly lost geometric structure — a situation exemplified in the upper part of Figure 10 — where only a few geometric planar structures are near the hole, our algorithm

is unable to fully reconstruct the hole's geometric surface. Secondly, if the genus of the closed original input mesh model after completion exceeds zero, and the location of a detected geometric hole to be filled coincides with the genus hole, our algorithm might inadvertently produce faces to close the genus hole, as illustrated in the bottom part of Figure 10. Thirdly, when the input model has disconnected components and a hole boundary is split between the two components, our algorithm struggles to form a complete and closed ring around the hole (see Figure 9 (f) top). Notice that for the above three situations, the other comparative algorithms mentioned in Section 4.2 also fail. Hence, solving these problems would be an interesting future research direction.

Moreover, for flawed LoD2 building models, resolving only the hole issue may not be adequate for downstream application requirements. This is because, apart from holes, the presence of issues like misoriented normals and inner faces can also greatly impact the overall quality of the final model. Consequently, the comprehensive goal in repairing LoD2 building models is to achieve a watertight, 2-manifold model with correctly oriented normals. We will take this goal as our future research direction.

5. Conclusions

In conclusion, our research contributes a novel algorithm to the field of urban model repair, particularly for filling holes in LoD2 building models. Our method surpassing traditional techniques in terms of preserving the original geometric structure and handling various topological errors. The algorithm's effectiveness is underscored by its ability to maintain geometric structure uniformity and integrity, proving advantageous over existing methods in qualitative and performance comparisons. While our approach represents a significant advancement, it also encounters certain limitations, particularly in addressing severely missing geometric structures, models with overlapping holes and non-zero genus, and entirely disconnected components. These challenges, alongside the broader goal of achieving comprehensive repair for LoD2 building models, set the direction for future research. The ultimate aim is to develop techniques that not only address holes but also rectify other critical aspects such as misoriented normals and inner faces, moving towards creating watertight, 2-manifold models with correctly oriented normals, thereby enhancing the utility and applicability of 3D urban building models.

References

Attene, M., 2010. A lightweight approach to repairing digitized polygon meshes. *The visual computer*, 26, 1393–1406.

Biljecki, F., Ledoux, H., Du, X., Stoter, J., Soon, K. H., Khoo, V., 2016. The most common geometric and semantic errors in CityGML datasets. *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 4, 13–22.

Biljecki, F., Stoter, J., Ledoux, H., Zlatanova, S., Çöltekin, A., 2015. Applications of 3D city models: State of the art review. *ISPRS International Journal of Geo-Information*, 4(4), 2842–2889.

Cignoni, P., Callieri, M., Corsini, M., Dellepiane, M., Ganovelli, F., Ranzuglia, G. et al., 2008. Meshlab: an open-source mesh processing tool. *Eurographics Italian chapter conference*, 2008, Salerno, Italy, 129–136.

DenHaag, 2022. 3D stadsmodel den haag 2022 citygml. https://denhaag.dataplatform.nl/#/data/12f4e0c6-1c2e-4867-acab-6182b8a683a5. Accessed: 2024-01-30.

Döllner, J., Kolbe, T. H., Liecke, F., Sgouros, T., Teichmann, K., 2006. The virtual 3d city model of berlin-managing, integrating, and communicating complex urban information. *Proceedings of the 25th international symposium on urban data management UDMS 2006 in Aalborg, Denmark, 15-17 May 2006*.

Fabri, A., Giezeman, G.-J., Kettner, L., Schirra, S., Schönherr, S., 2000. On the design of CGAL a computational geometry algorithms library. *Software: Practice and Experience*, 30(11), 1167–1202.

Geiger, A., Lauer, M., Wojek, C., Stiller, C., Urtasun, R., 2013. 3d traffic scene understanding from movable platforms. *IEEE transactions on pattern analysis and machine intelligence*, 36(5), 1012–1025.

Helsinki, C., 2019. Helsinki's 3D city models. https://www.hel.fi/helsinki/en/administration/information/general/3d. Accessed: 2020-11-25.

Huang, J., Stoter, J., Peters, R., Nan, L., 2022. City3D: Large-scale building reconstruction from airborne LiDAR point clouds. *Remote Sensing*, 14(9), 2254.

Ju, T., 2004. Robust repair of polygonal models. *ACM Transactions on Graphics (TOG)*, 23(3), 888–895.

Kolbe, T. H., Donaubauer, A., 2021. Semantic 3D city modeling and BIM. *Urban informatics*, 609–636.

Kolbe, T. H., Gröger, G., Plümer, L. et al., 2005. CityGML–Interoperable access to 3D city models. *Geo-information for disaster management*, 49.

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.

Ledoux, H., 2018. val3dity: validation of 3D GIS primitives according to the international standards. *Open Geospatial Data, Software and Standards*, 3(1), 1–12.

Liepa, P., 2003. Filling holes in meshes. *Proceedings of the 2003 Eurographics/ACM SIGGRAPH symposium on Geometry processing*, 200–205.

Mao, B., Ban, Y., 2011. Online visualization of 3D city model using CityGML and X3DOM. *Cartographica: The International Journal for Geographic Information and Geovisualization*, 46(2), 109–114.

Mulder, D., 2015. Automatic repair of geometrically invalid 3D city building models using a voxel-based repair method.

Murata, M., 2004. 3d-gis application for urban planning based on 3d city model. *24th annual esri international user conference*, 9–13.

Nan, L., 2021. Easy3D: a lightweight, easy-to-use, and efficient C++ library for processing and rendering 3D data. *Journal of Open Source Software*, 6(64), 3255.

Peters, R., Dukai, B., Vitalis, S., van Liempt, J., Stoter, J., 2022. Automated 3D reconstruction of LoD2 and LoD1 models for all 10 million buildings of the Netherlands. *Photogrammetric Engineering & Remote Sensing*, 88(3), 165–170.

Ran, C., 2011. The development of 3D city model and its applications in urban planning. *2011 19th International Conference on Geoinformatics*, IEEE.

Sieger, D., Botsch, M., 2019. The polygon mesh processing library. http://www.pmp-library.org.

Sindram, M., Machl, T., Steuer, H., Pültz, M., Kolbe, T. H., 2016. Voluminator 2.0–speeding up the approximation of the volume of defective 3D building models. *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 3, 29–36.

Wagner, D., Wewetzer, M., Bogdahn, J., Alam, N., Pries, M., Coors, V., 2012. Geometric-semantical consistency validation of citygml models. *Progress and new trends in 3D geoinformation sciences*, Springer, 171–192.

Yu, M., Lafarge, F., Oesau, S., Hilaire, B., 2022. Repairing geometric errors in 3D urban models with kinetic data structures. *ISPRS Journal of Photogrammetry and Remote Sensing*, 192, 315–326.

Zhang, L., Han, C., Zhang, L., Zhang, X., Li, J., 2014. Web-based visualization of large 3D urban building models. *International Journal of Digital Earth*, 7(1), 53–67.

Zhao, J., Ledoux, H., Stoter, J., Feng, T., 2018. HSW: Heuristic Shrink-wrapping for automatically repairing solid-based CityGML LOD2 building models. *ISPRS Journal of Photogrammetry and Remote Sensing*, 146, 289–304.

Zhao, W., Gao, S., Lin, H., 2007. A robust hole-filling algorithm for triangular mesh. *The Visual Computer*, 23, 987–997.