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

Semantic Edge Collapse: A Mesh Edge Collapse Algorithm preserving per Face Semantic Information

Grégoire Grzeczkowicz[1, 2] , Bruno Vallet [1]

[1] LASTIG, Univ Gustave Eiffel, IGN, ENSG
[2] Direction Générale de l'Armement - Gregoire.Grzeczkowicz@ign.fr

Keywords: Urban 3D modeling, 3D mesh simplification, Semantic information, Structured mesh

Abstract

Recent advancements in 3D data acquisition and processing have enabled high-fidelity urban modeling. Yet, production of structured 3D models in standards like *CityGML* remain complex, resource-intensive, and difficult to automate. This paper introduces a low-cost alternative that we call "structured mesh model" designed to cover many applications of structured 3D models at a lower cost. It relies on integrating geometric simplification with segmentation alignment to produce a lightweight, unified mesh representation. Using an edge-collapse algorithm, our method combines geometry from an existing mesh with labeled point cloud data to create a continuous mesh with edges aligned to segmentation boundaries, preserving both geometric fidelity and semantic clarity. The resulting structured mesh efficiently reduces memory requirements while maintaining accuracy, offering a practical solution for simulations and urban analyses that require structured 3D data.

1. Introduction

Recent advancements in 3D data acquisition, such as (Ign, 2022) and high-resolution photogrammetric techniques, have enabled the creation of highly detailed urban models. Although these dense, textured meshes capture both geometric and radiometric information, their unstructured nature limits their applicability in urban analysis and simulation contexts where object based representations are crucial. In structured urban modeling, the *CityGML* (Gröger and Plümer, 2012) standard provides a comprehensive, object-oriented framework that represents urban entities with semantic and hierarchical organization. However, CityGML has several drawbacks: costly production, complex multi-object models that consume significant memory and processing resources, and can introduce geometric inconsistencies, such as overlapping objects and gaps, which complicate their use in simulations. To address these limitations, we introduce the *structured mesh* model, a hybrid representation that balances detailed organization with computational efficiency. Unlike CityGML, structured meshes consist of a single, continuous mesh where each face contains metadata identifying both its semantic class (e.g., building, vegetation) and instance ID. This structure allows for a lightweight, unified geometry where objects are represented as subsets of mesh faces, maintaining individual object delineation without requiring separate geometries for each feature. This single-mesh design enables structured meshes to be lighter than CityGML models by using a unified mesh structure; simpler than CityGML models, allowing a fully automated generation and efficient for simulations. Our method of constructing structured meshes uses a combination of geometry from an existing mesh and semantic or instance information from a labeled point cloud. The approach computes a simplified mesh that preserves geometric accuracy while aligning mesh edges with segmentation boundaries. This is achieved through an edge-collapse algorithm, allowing the final resolution of the model to be adjusted. By iteratively collapsing edges, we can control the level of detail, balancing geometric fidelity and segmentation accuracy with reduced model complexity. Each face in the resulting structured mesh is associated with a single label, ensuring coherent representation of segmented regions within the mesh. This article details the methodology and implementation of this edge-collapse-based structured mesh generation. Our method is implemented in C++ using the Computational Geometry Algorithms Library (CGAL) (Project, 2024). Code and datasets are publicly available at `github.com/umrlastig/StructuredMesh`.

2. Related work

Mesh simplification is a subject that has been much studied in the 90s. Cignoni et al. (Cignoni et al., 1998) proposed a classification into 3 categories: vertex decimation (Schroeder et al., 1992, Soucy and Laurendeau, 1996), vertex clustering (Rossignac and Borrel, 1993, Luebke and Erikson, 1997), and edge collapse (Hoppe, 1996, Garland and Heckbert, 1997, Lindstrom and Turk, 1998). The first two approaches often encounter issues related to the preservation of topology, whereas the edge collapse techniques have gained prominence thanks to their flexibility and superior ability to maintain high-quality results with well-preserved topology. Edge collapse techniques have evolved significantly, with numerous enhancements proposed over the years. Modern algorithms utilized in tools such as Meshlab (Cignoni et al., 2008) and CGAL (Cacciola et al., 2024) are advanced versions of those presented by Garland abd Heckbert (Garland and Heckbert, 1997) or Lindstrom and Turk (Lindstrom and Turk, 1998), using for instance probabilistic quadrics (Trettner and Kobbelt, 2020). Recently, these algorithms have been adapted to take into account other forms of constraints. Salinas et al. (Salinas et al., 2015) try to guide the simplification process using planar proxies detected in preprocessing steps. Li et al. (Li et al., 2024) address a problem closely related to ours by prioritizing the simplification of contour and primitive vertices, both linear and non-linear, to preserve key structural elements. In a related but distinct approach, Scalas et al. (Scalas et al., 2020) proposed a method for maintaining the association between 3D annotations and the associated mesh during the simplification process.

3. Edge Collapse

Mesh simplification reduces the complexity of a mesh by decreasing the number of faces, edges, and vertices while preserving its shape. This process is driven by an edge-collapse operation, where an edge $e = (v', v'')$ is replaced by a single vertex v (see Fig. 1). All edges connected to v' and v'' are redirected to v, reducing the mesh by 3 edges, 2 faces, and 1 vertex.

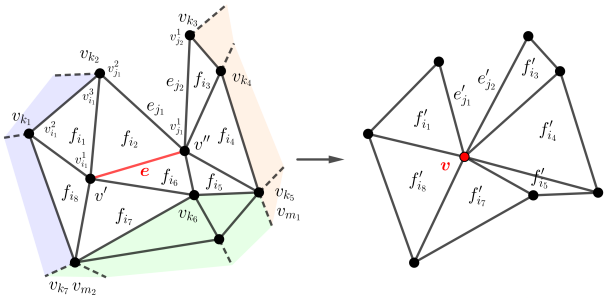

Figure 1. Notations for edge collapse.

The algorithm relies on two main strategies: the *placement strategy*, which chooses the optimal collapse point, and the *cost strategy*, which evaluates the cost of each edge collapse. Using a greedy approach, it iteratively collapses the edge with the lowest cost, recalculating the collapse cost for affected edges after each iteration, and continues until a stopping criterion, such as a target number of edges, is met. To incorporate label information, we adapt these strategies accordingly. While this operation does not entirely prevent mesh errors, such as self-intersections, we ensure the mesh remains a manifold by checking that the new vertex does not invert any faces. If an inversion is detected, the collapse is rejected, and the next edge is considered.

4. Placement strategy

4.1 Mathematical framework

In the following section, we use v and $\mathbf{v}$ to denote, depending on the context, the mesh vertex, the point in space, the vector from the origin, or the corresponding 3×1 matrix. Vectors and matrices will appear in bold, while vertices and points will be in regular font. Lindstrom and Turk (Lindstrom and Turk, 1998) proposed geometric constraints to ensure the new vertex v minimizes differences in mesh volume and boundary area after collapse. Their method prioritizes each constraint sequentially until the collapse point is defined, but this framework limits its flexibility for balancing multiple objectives. To address this, we define the placement of the new vertex v as the solution to an overconstrained linear system in the least squares sense:

$$\mathbf{v} = \arg\min_{\mathbf{x}} ||\mathbf{A}\mathbf{x} - \mathbf{B}||^2$$

If there are N objectives, then $\mathbf{A}$ is an $N \times 3$ matrix and $\mathbf{B}$ is $N \times 1$. This setup treats all objectives equally, allowing us to incorporate as many objectives as needed. Our approach avoids rigid constraints, finding a placement for v that optimally balances these objectives in the least squares sense.

4.2 Geometric objectives

For the geometric objectives, we choose to adopt the constraints of Lindstrom and Turk (Lindstrom and Turk, 1998), but they are now least squared objectives instead of hard constraints.

4.2.1 Volume preservation

During an edge collapse, triangular faces $f_i = (v_i^1, v_i^2, v_i^3)$ are replaced by the new faces $f_i' = (v, v_i^2, v_i^3)$. This operation changes the volume of the mesh by a volume equal to the sum of the volume of the tetrahedrons (v, v_i^1, v_i^2, v_i^3) (Figure 2).

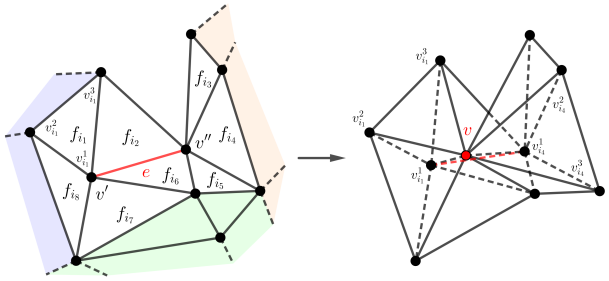

Figure 2. Mesh volume change after edge collapse.

We define the signed volume of the tetrahedron as

$$\begin{aligned} V^s\left(\left(v, v_i^1, v_i^2, v_i^3\right)\right) &= \frac{1}{3}\left(\frac{\mathbf{n}_i}{2} \cdot \left(\mathbf{v} - \mathbf{v}_i^1\right)\right) \\ &= \frac{1}{6}\left(\mathbf{n}_i \cdot \mathbf{v} - \mathbf{n} \cdot \mathbf{v}_i^1\right) \end{aligned}$$

where $\mathbf{n}_i = (\mathbf{v}_i^2 - \mathbf{v}_i^1) \times (\mathbf{v}_i^3 - \mathbf{v}_i^1)$ is a normal vector of the face $f_i = (v_i^1, v_i^2, v_i^3)$, with magnitude twice the area of f_i. A negative signed volume would correspond to a decrease in mesh volume, while a positive signed volume would correspond to an increase in mesh volume. The volume preservation objective attempts to minimize the mesh volume change introduced by the edge collapse. Thus we want to minimize

$$\left(\sum_i V^s\left(\left(v, v_i^1, v_i^2, v_i^3\right)\right)\right)^2 = \left(\frac{1}{6}\sum_i \mathbf{n}_i \cdot \mathbf{v} - \frac{1}{6}\sum_i \mathbf{n}_i \cdot \mathbf{v}_i^1\right)^2$$

where $f_i = (v_i^1, v_i^2, v_i^3)$ are the faces around the replacing edge e ($v_i^1 = v'$ or $v_i^1 = v''$) and $\mathbf{n}_i$ is the outward normal vector of face f_i, with magnitude twice the area of f_i. Thus v is solution of the equation $\mathbf{A}_1\mathbf{x} = \mathbf{B}_1$ with

$$\mathbf{A}_1 = \frac{1}{6}\sum_i \begin{bmatrix} n_i^x & n_i^y & n_i^z \end{bmatrix} \quad \text{and} \quad \mathbf{B}_1 = \frac{1}{6}\sum_i \left[\mathbf{n}_i \cdot \mathbf{v}_i^1\right] \quad (1)$$

4.2.2 Volumetric error minimization

The volume preservation ensures that the entire mesh volume does not change. The volumetric error minimization objective attempts to minimize the volume change locally, encouraging the new point to be close to the mesh surface. Thus we want to minimize

$$\sum_i V\left(\left(v, v_i^1, v_i^2, v_i^3\right)\right)^2 = \sum_i \left(\frac{1}{6}\mathbf{n}_i \cdot \mathbf{v} - \frac{1}{6}\mathbf{n}_i \cdot \mathbf{v}_i^1\right)^2$$

where $f_i = (v_i^1, v_i^2, v_i^3)$ are the faces around the replacing edge e. Thus v is solution of the equation $\mathbf{A}_2\mathbf{x} = \mathbf{B}_2$ with

$$\mathbf{A}_2 = \frac{1}{6}\begin{bmatrix} n_{i_1}^x & n_{i_1}^y & n_{i_1}^z \\ n_{i_2}^x & n_{i_2}^y & n_{i_2}^z \\ \vdots & \vdots & \vdots \\ n_{i_N}^x & n_{i_N}^y & n_{i_N}^z \end{bmatrix} \quad \text{and} \quad \mathbf{B}_2 = \frac{1}{6}\begin{bmatrix} \mathbf{n}_{i_1} \cdot \mathbf{v}_{i_1}^1 \\ \mathbf{n}_{i_2} \cdot \mathbf{v}_{i_2}^1 \\ \vdots \\ \mathbf{n}_{i_N} \cdot \mathbf{v}_{i_N}^1 \end{bmatrix} \quad (2)$$

where $f_{i_1} = (v_{i_1}^1, v_{i_1}^2, v_{i_1}^3), f_{i_2} = (v_{i_2}^1, v_{i_2}^2, v_{i_2}^3), ..., f_{i_N} = (v_{i_N}^1, v_{i_N}^2, v_{i_N}^3)$ are the faces around the replacing edge e ($v_i^1 = v'$ or $v_i^1 = v''$).

4.2.3 Boundary preservation If the model is not closed, when the border edge $e_j = (v_j^1, v_j^2)$ is changed for the new edge $e'_j = (v, v_j^2)$, the area of the mesh is changed by an area equal to that of the triangle (v, v_j^1, v_j^2) (Figure 3).

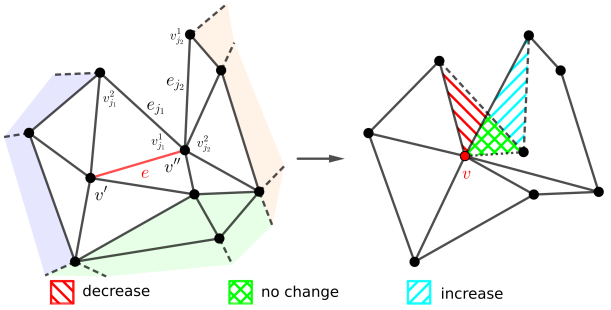

Figure 3. Mesh area change when a border edge is changed.

We define the directed area of the triangle (v, v_j^1, v_j^2) as

$$\begin{aligned} A\left(\left(v, v_j^1, v_j^2\right)\right) &= \frac{1}{2}\left(\left(\mathbf{v} - \mathbf{v}_j^2\right) \times \left(\mathbf{v}_j^1 - \mathbf{v}_j^2\right)\right) \\ &= \frac{1}{2}\left(\mathbf{v} \times \left(\mathbf{v}_j^1 - \mathbf{v}_j^2\right) - \mathbf{v}_j^2 \times \left(\mathbf{v}_j^1 - \mathbf{v}_j^2\right)\right) \\ &= \frac{1}{2}\left(\left(\mathbf{v}_j^2 - \mathbf{v}_j^1\right) \times \mathbf{v} - \mathbf{v}_j^2 \times \mathbf{v}_j^1\right) \end{aligned}$$

We define the directed area of the triangle (v, v_j^1, v_j^2) as

$$A^d\left(\left(v, v_j^1, v_j^2\right)\right) = \frac{1}{2}\left(\mathbf{v}_j^2 - \mathbf{v}_j^1\right) \times \mathbf{v} - \frac{1}{2}\mathbf{v}_j^2 \times \mathbf{v}_j^1$$

$A^d\left(\left(v, v_j^1, v_j^2\right)\right)$ is a normal vector of the triangle $\left(v, v_j^1, v_j^2\right)$, with magnitude the area of the triangle. By analogy with volume preservation, the boundary preservation objective attempts to minimize the area change at the boundary introduced by the edge collapse. Thus we want to minimize

$$\left\| \sum_j A^d\left(\left(v, v_j^1, v_j^2\right)\right) \right\|^2 = \left\| \frac{1}{2}\sum_j \left(\mathbf{v}_j^2 - \mathbf{v}_j^1\right) \times \mathbf{v} - \frac{1}{2}\sum_j \mathbf{v}_j^2 \times \mathbf{v}_j^1 \right\|^2$$

where $e_j = (v_j^1, v_j^2)$ are the border edges around the replacing edge e ($v_j^1 = v'$ or $v_j^1 = v''$). If the boundary is planar, all the directed area vector are collinear and we are simply minimizing the signed area difference of the operation (as for volume). Thus v is solution of the equation $\mathbf{A}_3\mathbf{x} = \mathbf{B}_3$ with

$$\mathbf{A}_3 = \frac{1}{2}\sum_j \begin{bmatrix} 0 & v_j^{1\,z} - v_j^{2\,z} & v_j^{2\,y} - v_j^{1\,y} \\ v_j^{2\,z} - v_j^{1\,z} & 0 & v_j^{1\,x} - v_j^{2\,x} \\ v_j^{1\,y} - v_j^{2\,y} & v_j^{2\,x} - v_j^{1\,x} & 0 \end{bmatrix}$$

and (3)

$$\mathbf{B}_3 = \frac{1}{2}\sum_j \begin{bmatrix} v_j^{1\,y} v_j^{2\,z} - v_j^{1\,z} v_j^{2\,y} \\ v_j^{1\,z} v_j^{2\,x} - v_j^{1\,x} v_j^{2\,z} \\ v_j^{1\,x} v_j^{2\,y} - v_j^{1\,y} v_j^{2\,x} \end{bmatrix}$$

4.2.4 Surfacic error minimization The boundary preservation ensures that the mesh area at boundary does not change. The surfacic error minimization objective attempts to minimize the area change locally, encouraging the new point to be close to the mesh boundary. Thus we want to minimize

$$\sum_j A\left(\left(v, v_j^1, v_j^2\right)\right)^2 = \sum_j \left\| \frac{1}{2}\left(\mathbf{v}_j^2 - \mathbf{v}_j^1\right) \times \mathbf{v} - \frac{1}{2}\mathbf{v}_j^2 \times \mathbf{v}_j^1 \right\|^2$$

Thus v is solution of the equation $\mathbf{A}_4\mathbf{x} = \mathbf{B}_4$ with

$$\mathbf{A}_4 = \frac{1}{2}\begin{bmatrix} 0 & v_{j_1}^{1\,z} - v_{j_1}^{2\,z} & v_{j_1}^{2\,y} - v_{j_1}^{1\,y} \\ v_{j_1}^{2\,z} - v_{j_1}^{1\,z} & 0 & v_{j_1}^{1\,x} - v_{j_1}^{2\,x} \\ v_{j_1}^{1\,y} - v_{j_1}^{2\,y} & v_{j_1}^{2\,x} - v_{j_1}^{1\,x} & 0 \\ \vdots & \vdots & \vdots \\ 0 & v_{j_N}^{1\,z} - v_{j_N}^{2\,z} & v_{j_N}^{2\,y} - v_{j_N}^{1\,y} \\ v_{j_N}^{2\,z} - v_{j_N}^{1\,z} & 0 & v_{j_N}^{1\,x} - v_{j_N}^{2\,x} \\ v_{j_N}^{1\,y} - v_{j_N}^{2\,y} & v_{j_N}^{2\,x} - v_{j_N}^{1\,x} & 0 \end{bmatrix}$$

and (4)

$$\mathbf{B}_4 = \frac{1}{2}\begin{bmatrix} v_{j_1}^{1\,y} v_{j_1}^{2\,z} - v_{j_1}^{1\,z} v_{j_1}^{2\,y} \\ v_{j_1}^{1\,z} v_{j_1}^{2\,x} - v_{j_1}^{1\,x} v_{j_1}^{2\,z} \\ v_{j_1}^{1\,x} v_{j_1}^{2\,y} - v_{j_1}^{1\,y} v_{j_1}^{2\,x} \\ \vdots \\ v_{j_N}^{1\,y} v_{j_N}^{2\,z} - v_{j_N}^{1\,z} v_{j_N}^{2\,y} \\ v_{j_N}^{1\,z} v_{j_N}^{2\,x} - v_{j_N}^{1\,x} v_{j_N}^{2\,z} \\ v_{j_N}^{1\,x} v_{j_N}^{2\,y} - v_{j_N}^{1\,y} v_{j_N}^{2\,x} \end{bmatrix}$$

where $e_{j_1} = (v_{j_1}^1, v_{j_1}^2), e_{j_2} = (v_{j_2}^1, v_{j_2}^2), ..., e_{j_N} = (v_{j_N}^1, v_{j_N}^2)$ are the border edges around the replacing edge e ($v_j^1 = v'$ or $v_j^1 = v''$).

4.2.5 Triangle shape optimization The triangle shape optimization objective attempts to make new face equilateral. The idea is to minimize the perimeter-to-surface ratio. As the position of the point barely changes the surface, only the perimeter needs to be minimized. Thus we want to minimize $\sum_k D(v, v_k)^2$ where $D(v, v_k)$ is the distance between v and the incident vertices v_k upon the collapsing edge e.

$$\sum_k D(v, v_k)^2 = \sum_k \|\mathbf{v} - \mathbf{v}_k\|^2$$

Thus v is solution of the equation $\mathbf{A}_5\mathbf{x} = \mathbf{B}_5$ with

$$\mathbf{A}_5 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ \vdots & \vdots & \vdots \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \text{and} \quad \mathbf{B}_5 = \sum_k \begin{bmatrix} v_{k_1}^x \\ v_{k_1}^y \\ v_{k_1}^z \\ \vdots \\ v_{k_N}^x \\ v_{k_N}^y \\ v_{k_N}^z \end{bmatrix} \tag{5}$$

where $v_{k_1}, v_{k_2}, ..., v_{k_N}$ are the incident vertices upon the replacing edge e.

4.3 Label purity

A key contribution of our work is the addition of two objectives to ensure label purity during edge-collapse. To achieve this, each face of the mesh must have a single label, with mesh edges separating different labels. We use a Support Vector Machine (SVM) to determine the plane that best separates points with different labels and restrict the collapsed point to this plane.

When points cannot be linearly separated (e.g., near segmentation corners), we employ an directe search to find the best placement, minimizing the occurrence of mixed labels within faces. Each point in the sampled point cloud is associated with its nearest mesh face. After an edge collapse, points on modified faces are reassigned to their new closest face.

4.3.1 Support Vector Machine (SVM) Support Vector Machines (SVM), introduced by Cortes and Vapnik (Cortes and Vapnik, 1995), determine a hyperplane that separates points of two classes, maximizing the gap between them. This works directly for linearly separable classes, but for non-linear cases, points are projected into a higher-dimensional space where they become separable. Formally, for N labeled points $(\mathbf{x}_1, y_1)$, $(\mathbf{x}_2, y_2)$, ..., $(\mathbf{x}_N, y_N)$ with $\mathbf{x}_i$ in space and $y_i \in \{-1, 1\}$, the hyperplane is defined as $\mathbf{w} \cdot \mathbf{x} - b = 0$. The goal is to find $\mathbf{w}$ and b that minimize $\|\mathbf{w}\|^2$, under the constraint that $y_i(\mathbf{w} \cdot \mathbf{x}_i - b) \geq 1$.

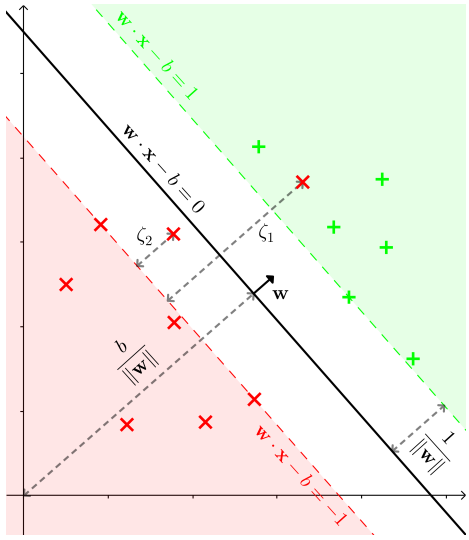

Figure 4. Linear SVM with soft-margin in a 2D plane. $\mathbf{w} \cdot \mathbf{x} - b = 0$ is the line separating points of two labels (red crosses and green circles) with maximum margin.

If classes are not perfectly separable, hinge loss (soft-margin) is used, allowing some points to fall within the margin. This setup minimizes $\|\mathbf{w}\|^2 + C\sum_{i=1}^{N} \zeta_i$ under $y_i(\mathbf{w} \cdot \mathbf{x}_i - b) \geq 1 - \zeta_i$ and $\zeta_i \geq 0$, as shown in Fig. 4. The parameter C balances margin size and misclassification tolerance.

4.3.2 Point separation In our case, we use soft-margin SVM directly on 3D points. For a given edge e, we consider points from faces around e and the labels with the majority in each face. If only two labels dominate, we find a plane that maximizes the separation between these points, ignoring other labels. For cases with more than two majority labels, we compute a separate plane $(\mathbf{w}_l, b_l)$ for each label, separating points of that label from the others. To reduce computation time, we identify "edge points"—points where neighboring points have different labels—and use only these in SVM computations. This approach yields one or more planes $(\mathbf{w}_{l_1}, b_{l_1})$, $(\mathbf{w}_{l_2}, b_{l_2})$, $(\mathbf{w}_{l_N}, b_{l_N})$, and the placement of v is given by solving the system $\mathbf{A}_6\mathbf{x} = \mathbf{B}_6$:

$$\mathbf{A}_6 = \begin{bmatrix} \mathbf{w}_{l_1}^t \\ \mathbf{w}_{l_2}^t \\ \vdots \\ \mathbf{w}_{l_N}^t \end{bmatrix} \quad \text{and} \quad \mathbf{B}_6 = \begin{bmatrix} b_{l_1} \\ b_{l_2} \\ \vdots \\ b_{l_N} \end{bmatrix} \tag{6}$$

where $(\mathbf{w}_{l_1}, b_{l_1})$, $(\mathbf{w}_{l_2}, b_{l_2})$, ..., $(\mathbf{w}_{l_N}, b_{l_N})$ are the SVM planes separating different labels around edge e. Using the soft-margin SVM is essential, as slight noise in point labels can lead to outliers, making full separation challenging.

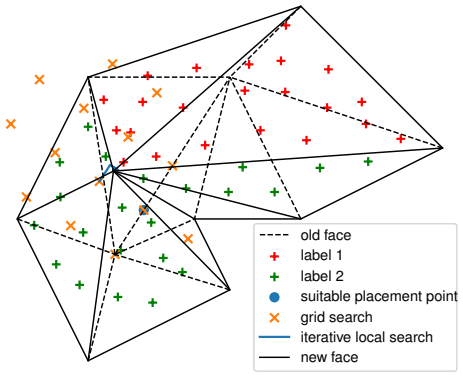

Figure 5. Direct search. A suitable placement point is selected on the collapsing edge (the blue dot). Then a grid search is performed from this point (in orange). Finally, an iterative local search follows from the best grid point (the blue line).

4.3.3 Direct Search When points are not linearly separable, such as in segmentation corners, we use a direct iterative search to find the optimal placement point that minimizes label mixing within faces (see Fig. 5). This search is computationally intensive, so we apply it only when necessary, specifically when over 5% of points are mislabeled by the SVM. These challenging cases often arise in flat areas, where placement becomes under-constrained. Therefore, our direct search operates in two dimensions. We begin by calculating a local tangent plane P and projecting relevant faces and points onto it. The goal is to obtain "pure" faces, where all points share the same label. Since a simple count of mislabelled points is not continuous, we define a continuous membership function $M(p, f)$ that assigns each point p a membership value to face f, ranging from 1 (exact match) to 0 (distant). This function encourages boundary alignment for smoother optimization, as illustrated in Fig. 6.

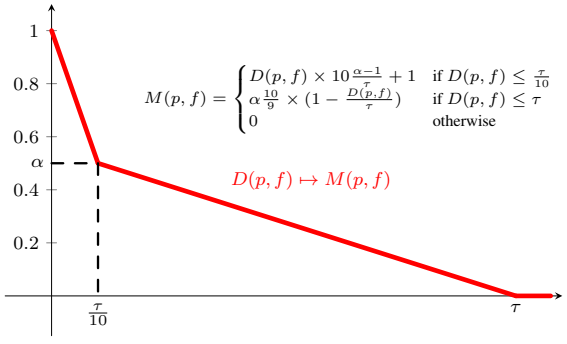

Figure 6. Distance to faces point memberships calculation.

The search begins by identifying a viable placement point along the edge, starting at the midpoint and exploring along the edge with decreasing step sizes to avoid face inversion. Once a suitable point is found, a grid search expands from it, followed by a local iterative search that evaluates points in a cross pattern. The process continues until the optimal point is identified, with step size halving if no improvement is found. The search concludes when differences between successive points fall below 0.0001. This process yields the best placement point p^* and the optimal

placement line d^* orthogonal to P through p^*. We then compute two orthogonal planes $(\mathbf{w}_1, b_1)$ and $(\mathbf{w}_2, b_2)$ intersecting at d^*. The final placement of v satisfies $\mathbf{A}_6\mathbf{x} = \mathbf{B}_6$:

$$\mathbf{A}_6 = \begin{bmatrix} \mathbf{w}_1^t \\ \mathbf{w}_2^t \end{bmatrix} \quad \text{and} \quad \mathbf{B}_6 = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} \tag{7}$$

4.3.4 Segmentation contour minimization Our method aims to maximize face purity, which can conflict with mesh simplification. Ensuring face purity could mean avoiding edge collapses near segmented regions, resulting in a final contour with many small, uneven edges. To address this, we add an objective to minimize the total length of the segmentation contour, encouraging smoother, more regular contours. The goal is to minimize the distance from v to the segmentation contour points $(v_{m_1}, v_{m_2}, ..., v_{m_{N'}})$ among the vertices $(v_{k_1}, v_{k_2}, ..., v_{k_N})$ on edge e, i.e., points along edges between faces of different labels. Thus, v is determined by solving $\mathbf{A}_7\mathbf{x} = \mathbf{B}_7$:

$$\mathbf{A}_7 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ \vdots & \vdots & \vdots \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \text{and} \quad \mathbf{B}_7 = \sum_i \begin{bmatrix} v_{m_1}^x \\ v_{m_1}^y \\ v_{m_1}^z \\ \vdots \\ v_{m_{N'}}^x \\ v_{m_{N'}}^y \\ v_{m_{N'}}^z \end{bmatrix} \tag{8}$$

4.4 System solution

The matrices $\mathbf{A}_1, \mathbf{A}_2, ..., \mathbf{A}_7$ and $\mathbf{B}_1, \mathbf{B}_2, ..., \mathbf{B}_7$ represent the objectives for placing the vertex v that replaces edge e. To control the influence of each objective, we introduce parameters $\lambda_1, \lambda_2, ..., \lambda_7$. The solution for v is found by solving the over-constrained system $\mathbf{A}\mathbf{x} = \mathbf{B}$ in the least squares sense, where $\mathbf{A} = [\lambda_i \mathbf{A}_i]_{i=1..7}$ and $\mathbf{B} = [\lambda_i \mathbf{B}_i]_{i=1..7}$. We therefore solve $\mathbf{v} = \arg\min_{\mathbf{x}} \|\mathbf{A}\mathbf{x} - \mathbf{B}\|^2$. Choosing values for $\lambda_1, \lambda_2, ..., \lambda_7$ allows us to balance between objectives. For example, increasing λ_1 emphasizes volume preservation over surface fidelity or segmentation purity. This formalism adapts the objectives proposed by Lindstrom and Turk (Lindstrom and Turk, 1998), while adding symmetry and the flexibility to include new objectives for segmentation.

5. Cost strategy

To determine the optimal order for collapsing edges, we also need to evaluate the cost of each edge-collapse so that we can prioritize edges from least to most expensive. There are two main strategies for determining this cost. Like Lindstrom and Turk (Lindstrom and Turk, 1998), we could base it solely on placement error, allowing memoryless simplification without storing the original mesh or modification history. Alternatively, we can leverage the point cloud, which contains both segmentation and geometric information.

5.1 Placement strategy-based cost

The objectives from the previous section enable us to calculate placement errors (e.g., volume and area changes). Thus, a basic cost strategy is to use $\|\mathbf{A}\mathbf{x} - \mathbf{B}\|^2$ as the contraction cost:

$$\begin{aligned} C_p = {} & \lambda_1^2 \left(\sum_i V^s \left(\left(v, v_i^1, v_i^2, v_i^3 \right) \right) \right)^2 \\ & + \lambda_2^2 \sum_i V \left(\left(v, v_i^1, v_i^2, v_i^3 \right) \right)^2 \\ & + \lambda_3^2 \left\| \sum_j A^d \left(\left(v, v_j^1, v_j^2 \right) \right) \right\|^2 \\ & + \lambda_4^2 \sum_j A \left(\left(v, v_j^1, v_j^2 \right) \right)^2 \\ & + \lambda_5^2 \sum_k D(v, v_k)^2 \\ & + \lambda_6^2 \sum_l (\mathbf{w}_l \cdot \mathbf{v} - b_l) \quad \text{or} \quad \lambda_6^2 D(v, d^*)^2 \\ & + \lambda_7^2 \sum_m D(v, v_m)^2 \end{aligned} \tag{9}$$

where f_i are faces around the edge e, e_j are boundary edges around e, $D(v, v_k)$ is the distance between v and nearby vertices v_k, $(\mathbf{w}_l, b_l)$ are SVM planes separating labels, $D(v, d^*)$ is the distance to the best placement line, and $D(v, v_m)$ is the distance to segmentation contour points v_m.

5.2 Point cloud-based cost

Using the point cloud allows us to calculate costs without additional memory usage. For each face f and associated points p_i with labels L_{p_i}, we assign f the label L_f that most of its points share. We then define two costs, an absolute geometric cost

$$C_g = \sum_i D(p_i, f)^2 \tag{10}$$

and an absolute segmentation cost

$$C_s = |\{p_i | L_{p_i} \neq L_f\}| \tag{11}$$

where $D(p_i, f)$ is the distance from point p_i to face f, and $|\{p_i | L_{p_i} \neq L_f\}|$ is the count of points that do not match the majority label.

5.3 Segmentation contour-based cost

As discussed in Section 4.3.4, another objective is to minimize the length of the segmentation contour. We define the contour cost as:

$$C_c = \sum_n L(e_n) \tag{12}$$

where $L(e_n)$ is the length of edge e_n, and e_n are edges between faces with different labels.

5.4 Combined cost

Each cost has strengths and weaknesses. The placement cost C_p is continuous but can drift as it only reflects the previous mesh state. The geometric cost C_g compares the simplified mesh to the original but relies on discrete sampling. The segmentation costs C_s and C_c provide more precise label alignment by assessing point-by-point consistency. To combine these, we use:

$$C = \alpha C_g + \beta C_s + \gamma C_c + \delta C_p \tag{13}$$

where α, β, γ, and δ are chosen based on the desired balance among these objectives.

6. Face division

Edge collapse simplifies the mesh by merging faces but does not allow moving mesh points without reducing face count. Some segmentation details, however, may be smaller than the initial mesh faces. To handle this, we introduce a face division step before edge collapse, assigning labels to faces based on the method in the previous section. We divide only those faces associated with points of different labels. For each face with mixed labels, using SVM, we find the plane that best separates the majority-label points from the others. The triangular face is then divided at the intersection of this plane and the face. The new edge extends to the opposite vertices of adjacent faces and to the midpoint of any edge not intersected by the plane. If the angle between the face and the SVM plane is less than 10°, the plane is ignored, and the face is simply divided at the midpoints of each edge. This process iterates until no face has points with different labels.

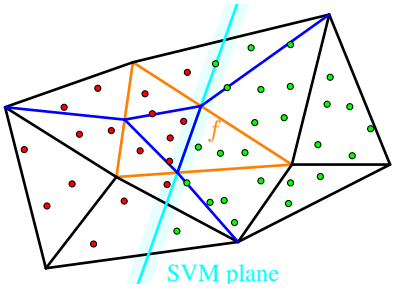

Figure 7. Face division. New edges are shown in dark blue, and the SVM plane intersection creating the first edge is in light blue, with its margin shaded.

7. Evaluation

To evaluate our method, we used three models: the Stanford Bunny (Stanford Bunny, n.d.), the Fandisk (Hoppe et al., 1994), and a tile from the SUM Dataset (Gao et al., 2021). The first two are standard choices in geometry processing evaluations. For these, we generated labeled point clouds using Meshlab's (Cignoni et al., 2008) Poisson-disk Sampling (Corsini et al., 2012) and manually labeled them to capture various scenarios, such as segmentation boundaries align or not align to with sharp edges, large areas, and fine details. The resulting point clouds are available at `github.com/umrlastig/SemanticMesh`. The third dataset tests our method on structured mesh generation and includes a high-resolution mesh (450,000+ edges) of a $250\,\mathrm{m}$ x $250\,\mathrm{m}$ segment of Helsinki, with a point cloud detailing semantic classes (e.g., buildings, vegetation, water, vehicles).

Our goal is to assign labels to each face. Final labeling is achieved through a majority vote among the points nearest to each face, which can be done at any stage of simplification. In cases where a low number of points are associated with a face, indicating limited information in certain areas, we introduce a minimum occupancy fraction. First, an average point density per unit area is calculated for the mesh. Then, for each face, if the number of points falls below a certain fraction of the expected count, it is labeled as "unknown". Similarly, very small faces might lack associated points. If a face's expected point count is less than 1, it is labeled using a majority vote weighted by the length of shared edges with neighboring faces. This process is recursively applied to adjacent faces with missing labels.

To evaluate geometric error, we use the symmetric Hausdorff Distance (Cignoni et al., 1998) between the initial and final meshes. Given a sampling of the two meshes, this is the greatest distance between one mesh and the farthest sampling point of the other mesh. In one direction, we measure from vertices of the initial mesh to the final mesh; in the other, from randomly sampled points on the final mesh surface to the initial mesh. Figure 8 shows the progression of mean distance error during mesh simplification.

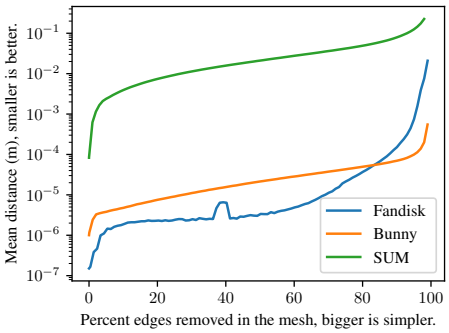

Figure 8. Geometric error during simplification.

To assess segmentation error, we count the points in the initial point cloud whose closest face on the final mesh has a different label. We also compare results with the CGAL implementation of the Lindstrom-Turk and Garland-Heckbert algorithms (Cacciola et al., 2024) using default settings. These methods ignore segmentation, resulting in lower segmentation accuracy and underscoring the importance of segmentation in our approach. For consistency, we use the number of remaining edges as the stopping criterion, a common practice (Cacciola et al., 2024) that allows comparison between meshes of similar sizes. Additionally, our method allows to stop at a maximum collapse cost derived from acceptable mesh-to-mesh distance or point count errors, using the parameters α, β, γ, and δ from equation 13.

We achieved optimal results with the following hyperparameters: $\alpha = 2$, $\beta = 1$, $\gamma = 0.01$, $\delta = 0.01$, $\lambda_1 = 10$, $\lambda_2 = 1$, $\lambda_3 = 10$, $\lambda_4 = 1$, $\lambda_5 = 1 \times 10^{-5}$, $\lambda_6 = 1$, $\lambda_7 = 0.01$, $C = 1 \times 10^5$. As shown in Table 1, our method performs well in segmentation, with minimal association errors between the point cloud and the final mesh, as less than $0.04\,\%$ of points are assigned incorrect labels. The primary drawback of our method, however, is its significantly longer processing time due to more complex objectives and to re-association: each collapse operation increases the number of points linked to each face, requiring time-consuming re-association as illustrated in Figure 9. Re-association is essential for aligning mesh edges with segmentation boundaries. When segmentation is ignored, whether with Lindstrom-Turk, Garland-Heckbert, or our method without the point cloud, point-to-face label mismatches increase significantly. To understand the impact of each component, we conducted an ablation study (Table 1). Removing the direct search component (see Section 4.3.3) has minimal effect Excluding segmentation objectives reduces geometric and segmentation accuracy slightly but speeds up processing by a factor of three. Running SVM on all points instead of edge points greatly increases computation time without notable accuracy improvements. Figures 8 and 9 reveal three simplification phases: an initial phase where segmentation contour minimization (C_c) simplifies the contour, causing a sharp local error increase; a

Table 1. Ablation study and comparison with baseline.
Experiments are compared with the same number of edges in the final mesh (2,500 for Bunny, 1,000 for Fandisk and 50,000 for SUM).
Without pointcloud: $\lambda_6 = 0, \lambda_7 = 0, \alpha = 0, \beta = 0, \gamma = 0, \delta = 1$ - Without segmentation placement: $\lambda_6 = 0, \lambda_7 = 0$

Model	Experience	Mean distance (mm)	Max distance (mm)	Wrong Point (%)	Contour length (m)	Time (s)
Bunny	Complete method	0.17	1.9	0.039	0.7	81.8
Bunny	*Garland-Heckbert*	0.18	1.8	1.485	**0.6**	**0.6**
Bunny	*Lindstrom-Turk*	**0.14**	**1.5**	1.444	0.7	1.1
Bunny	without pointcloud	0.23	2.3	1.702	0.7	5.2
Bunny	without segmentation placement	0.16	1.6	0.039	0.7	28.4
Bunny	without direct search	0.16	2.2	0.032	0.7	66.6
Bunny	without initial subdivision	0.16	1.9	**0.030**	0.7	80.1
Bunny	without edge point	0.17	2.3	0.037	0.7	213.0
Fandisk	Complete method	1.16	19.5	**0.001**	28.8	450.1
Fandisk	*Lindstrom-Turk*	0.43	**9.2**	4.863	26.3	**0.3**
Fandisk	*Garland-Heckbert*	3.48	226.1	6.063	**20.0**	0.5
Fandisk	without pointcloud	**0.42**	16.1	5.006	33.5	1.3
Fandisk	without segmentation placement	4.16	79.4	0.018	28.7	91.1
Fandisk	without direct search	1.32	25.6	0.002	28.8	275.9
Fandisk	without initial subdivision	0.63	11.6	0.043	29.1	930.9
SUM	Complete method	75.3	1.9	**0.037**	10.0	292.3
SUM	*Lindstrom-Turk*	14876.1	1231.9	9.556	29.2	**10.1**
SUM	*Garland-Heckbert*	883.7	13.3	9.482	21.2	175.5
SUM	without pointcloud	81.1	6.9	1.778	11.7	25.9
SUM	without segmentation placement	**74.9**	2.0	0.057	11.0	64.1
SUM	50% of pointcloud	77.2	**1.8**	1.902	**9.8**	175.6
SUM	without initial subdivision	75.4	1.9	0.043	10.1	294.1
SUM	without direct search	75.4	**1.8**	0.039	10.0	302.0
SUM	without edge point	75.2	1.9	0.042	10.0	374.2

second phase with exponential error growth as the mesh is simplified; and a third phase where reduced edge count impacts model fidelity, leading to another sharp error increase. Removing segmentation placement or edge points affects time mainly in the first phase, where segmentation boundaries undergo the most adjustments. An example is seen in the Fandisk model, where simplifying the "IGN" letters demands more segmentation processing. Finally, the initial subdivision of faces significantly improves accuracy and reduces the number of mislabeled points, ultimately saving processing time.

8. Conclusion

This paper presents an algorithm that combines mesh simplification with segmentation alignment, producing structured meshes with edges that match segmentation boundaries. By preserving both geometric fidelity and segmentation labels, the method enables lightweight, structured models suitable for various applications. While computationally demanding, the approach allows flexibility in handling semantic segmentation and demonstrates a trade-off between processing time and model detail. Future optimizations, including GPU acceleration and machine learning enhancements, could further improve efficiency and boundary alignment. In summary, our approach effectively generates clear, structured meshes, providing an automated, practical solution for urban modeling needs.

References

Cacciola, F., Rouxel-Labbé, M., Şenbaşlar, B., Komaromy, J., 2024. Triangulated surface mesh simplification. *CGAL User and Reference Manual*, 5.6.1 edn, CGAL Editorial Board.

Cignoni, P., Callieri, M., Corsini, M., Dellepiane, M., Ganovelli, F., Ranzuglia, G., 2008. Meshlab: an open-source mesh processing tool. V. Scarano, R. D. Chiara, U. Erra (eds), *Eurographics Italian Chapter Conference*, The Eurographics Association.

Cignoni, P., Montani, C., Scopigno, R., 1998. A comparison of mesh simplification algorithms. *Computers & Graphics*, 22(1), 37–54.

Corsini, M., Cignoni, P., Scopigno, R., 2012. Efficient and Flexible Sampling with Blue Noise Properties of Triangular Meshes. *IEEE Transactions on Visualization and Computer Graphics*, 18(6), 914–924.

Cortes, C., Vapnik, V., 1995. Support-vector networks. *Machine Learning*, 20(3), 273–297.

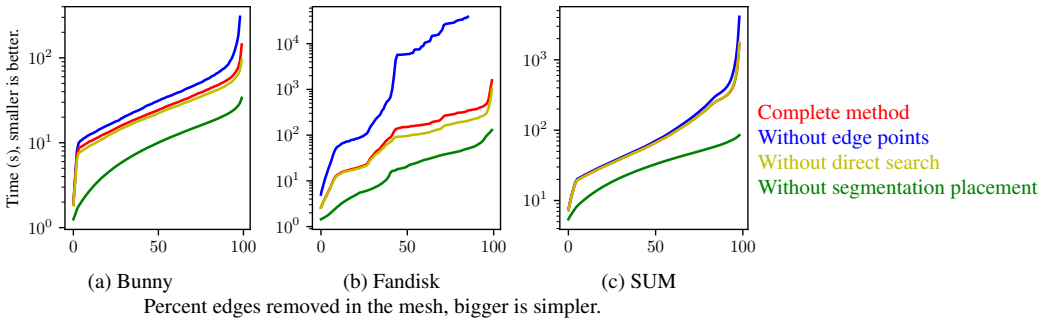

Figure 9. Time flow during simplification.

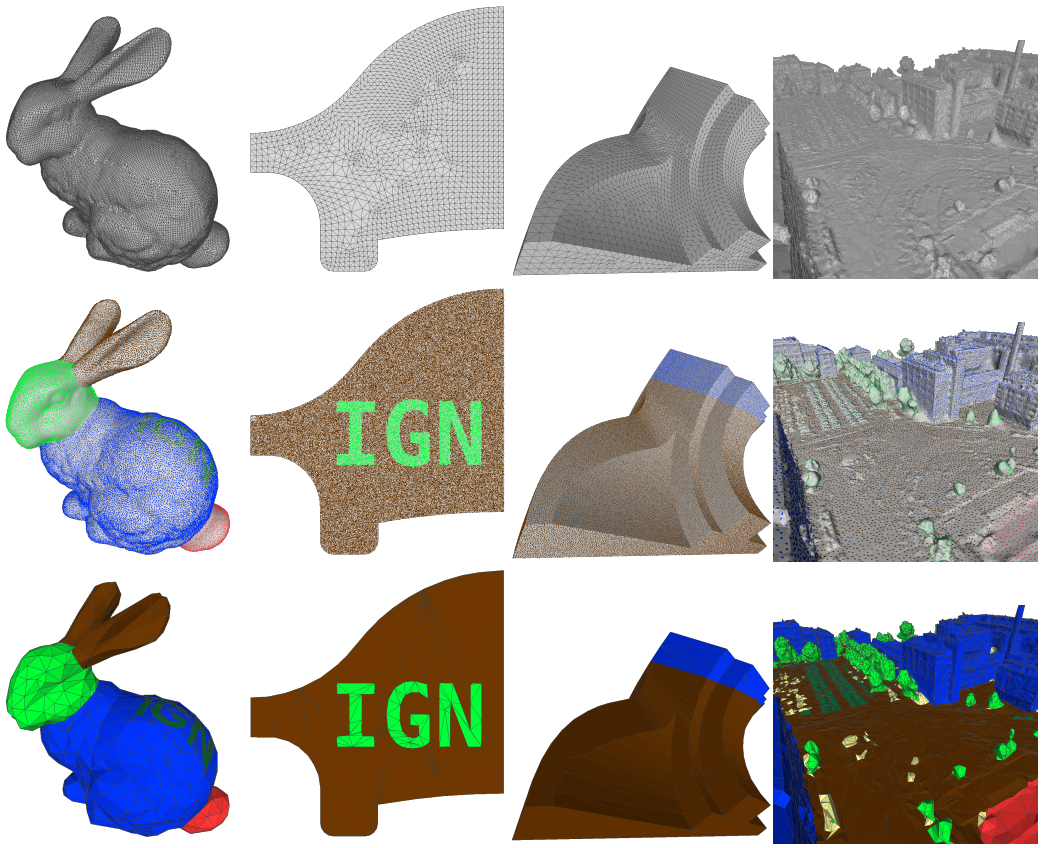

Figure 10. Original mesh, labeled point cloud and structured mesh (top to bottom) for Bunny, Fandisk and SUM dataset (left to right). The results show good geometric approximation while maintaining a sufficient number of edges to mark segmentation details.

Gao, W., Nan, L., Boom, B., Ledoux, H., 2021. SUM: A benchmark dataset of Semantic Urban Meshes. *ISPRS Journal of Photogrammetry and Remote Sensing*, 179, 108–120.

Garland, M., Heckbert, P. S., 1997. Surface simplification using quadric error metrics. *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques*, Siggraph '97, ACM Press/Addison-Wesley Publishing Co., Usa, 209–216.

Gröger, G., Plümer, L., 2012. CityGML – Interoperable semantic 3D city models. *ISPRS Journal of Photogrammetry and Remote Sensing*, 71, 12–33.

Hoppe, H., 1996. Progressive meshes. *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, Siggraph '96, Association for Computing Machinery, New York, NY, USA, 99–108.

Hoppe, H., DeRose, T., Duchamp, T., Halstead, M., Jin, H., McDonald, J., Schweitzer, J., Stuetzle, W., 1994. Piecewise smooth surface reconstruction. *Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques*, Siggraph '94, 295–302.

Ign, 2022. Lidar hd. `https://geoservices.ign.fr/lidarhd`.

Li, J., Chen, D., Hu, F., Wang, Y., Li, P., Peethambaran, J., 2024. Shape-preserving mesh decimation for 3D building modeling. *International Journal of Applied Earth Observation and Geoinformation*, 126, 103623.

Lindstrom, P., Turk, G., 1998. Fast and memory efficient polygonal simplification. *Proceedings Visualization '98 (Cat. No.98CB36276)*, 279–286.

Luebke, D., Erikson, C., 1997. View-dependent simplification of arbitrary polygonal environments. *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques*, Siggraph '97, ACM Press/Addison-Wesley Publishing Co., Usa, 199–208.

Project, T. C., 2024. *CGAL User and Reference Manual*. 5.6.1 edn, CGAL Editorial Board.

Rossignac, J., Borrel, P., 1993. Multi-resolution 3d approximations for rendering complex scenes. B. Falcidieno, T. L. Kunii (eds), *Modeling in Computer Graphics*, Springer Berlin Heidelberg, Berlin, Heidelberg, 455–465.

Salinas, D., Lafarge, F., Alliez, P., 2015. Structure-Aware Mesh Decimation. *Computer Graphics Forum*, 34(6), 211–227.

Scalas, A., Mortara, M., Spagnuolo, M., 2020. A pipeline for the preparation of artefacts that provides annotations persistence. *Journal of Cultural Heritage*, 41, 113–124.

Schroeder, W. J., Zarge, J. A., Lorensen, W. E., 1992. Decimation of triangle meshes. *SIGGRAPH Comput. Graph.*, 26(2), 65–70.

Soucy, M., Laurendeau, D., 1996. Multiresolution Surface Modeling Based on Hierarchical Triangulation. *Computer Vision and Image Understanding*, 63(1), 1–14.

Stanford Bunny, n.d. `https://graphics.stanford.edu/data/3Dscanrep/`. Stanford University Computer Graphics Laboratory.

Trettner, P., Kobbelt, L., 2020. Fast and Robust QEF Minimization using Probabilistic Quadrics. *Computer Graphics Forum*, 39(2), 325–334.