Ensuring Watertightness in Geometry Models for CFD: A Robust and Automated Validation and Repair Method for Surface Meshes

Tianyu Zhou[1], Carlos F. Lange[2], Michael Versteege[3], Yongsheng Ma[4], and Brian Fleck[5]

[1]University of Alberta, tzhou4@ualberta.ca
[2]University of Alberta, carlos.lange@ualberta.ca
[3]University of Alberta, mhverste@ualberta.ca
[4]Southern University of Science and Technology, mays@sustech.edu.cn
[5]University of Alberta, bfleck@ualberta.ca

Corresponding author: Brian Fleck, bfleck@ualberta.ca

Abstract. Ensuring the watertightness of the input geometry model before a Computational Fluid Dynamics (CFD) simulation is a crucial preprocessing step. A non-watertight model can lead to errors in mesh generation and inaccurate simulation results. Traditional watertight validation and model repair approaches often require significant manual intervention and domain expertise, making them inefficient for large-scale and automated CFD workflows. This paper presents a robust and automated approach for detecting and repairing surface mesh defects to ensure watertightness. Our method integrates automated detection, repair, and simplification algorithms, reducing manual effort while maintaining high accuracy and efficiency. A comparative analysis with existing methods highlights the advantages of our approach in terms of automation, robustness, and computational efficiency. The proposed method is validated through case studies, including repairing the Stanford Bunny and a real-world campus building geometry, demonstrating its effectiveness in CFD preprocessing workflows.

Keywords: 3D Watertight Surface Mesh Reconstruction, Automated Geometry Repair for CFD Simulations, CFD Geometry Preprocessing and Optimization, Model Watertightness Validation, Interactive Surface Mesh Processing Framework
DOI: https://doi.org/10.14733/cadaps.2026.552-571

1 INTRODUCTION

Computational Fluid Dynamics (CFD) has become an essential tool for analyzing and predicting fluid behavior in a variety of engineering applications, such as vehicle aerodynamics for fuel efficiency and performance, air pollution dispersion and ventilation in urban environments, and biomedical flow analysis [1,2]. When doing CFD simulations, especially for studying fluid flow around an object, the accuracy of results heavily depends on the quality of the computational domain, which is derived from geometric models. One critical aspect of preparing a reliable computational domain is ensuring that the surface mesh representing the geometry is watertight [3]. A watertight model is defined as one where all surfaces are connected without gaps or holes, preventing errors in mesh generation and the numerical solution of governing equations [4]. If a model is not watertight, severe problems, such as unphysical flow leaks, incorrect boundary conditions, poor grid generation, or even complete solver failures. In addition, fluid domain integrity is critical in CFD to avoid the formation of artificial internal boundaries that could distort simulation results, particularly in high-fidelity turbulence modeling and multiphase flow applications [3].

Despite its importance, ensuring watertightness remains a major challenge in geometry preprocessing before a CFD simulation, particularly for geometries obtained from Computer-Aided Design (CAD) software, 3D scans, or procedural generation techniques. These sources often introduce mesh defects, including:

- Missing faces and holes, which lead to unphysical flow leaks and disconnected fluid domains.
- Duplicate and unreferenced vertices, causing inconsistencies in element connectivity and leading to numerical instabilities during mesh refinement.
- Non-manifold edges and vertices, where a single edge or vertex is shared incorrectly by multiple surfaces. This may lead to CFD meshing failures and poor solver convergence.
- Self-intersections and overlapping elements, which create numerical instability in CFD solvers and increase the likelihood of convergence errors.

Traditionally, resolving these issues requires extensive manual intervention using mesh-editing software such as Autodesk Meshmixer, MeshLab, or commercial CFD preprocessors. However, manual correction is time-consuming, prone to human error, and impractical for large-scale, automated workflows [5,6]. Furthermore, different CFD applications involve a variety of file formats [7,8], making it difficult to establish a universal manual repair workflow that is both reliable and efficient [9]. Several semi-automated approaches have been developed to address watertightness validation and repair, such as Poisson surface reconstruction [10], graph-based hole filling [11], and heuristic-based mesh repair [12]. Despite these advances, most existing methods have limited automation levels, require domain-specific expertise in geometry processing, and struggle with handling different file formats.

To address these challenges, we propose a robust and automated approach for detecting and repairing geometry model defects to ensure that the preprocessed models will be watertight before CFD simulation. The key advantages of our proposed method are:

- Multi-Format Support – The proposed framework supports a wide range of input and output formats, including STL, OBJ, STEP, point clouds, etc., making it adaptable to diverse design sources and industry workflows.
- Robustness and Fault Tolerance – Designed to handle user errors gracefully. When incorrect operations or corrupted inputs are detected, the system provides clear error messages or automated reminders, reducing the risk of invalid outputs and enhancing reliability across varied use cases.
- High Automation Level – By integrating all key CFD geometry preprocessing steps into a unified pipeline, the method achieves a high level of automation, minimizing the need for expert intervention while maintaining control where needed.
- Visualization and Interactive Debugging – Provides built-in visualization tools to interactively inspect mesh issues and preview repairs. Users can visually verify mesh integrity and feature preservation in a user-guided yet automated workflow.

The remainder of this paper is organized as follows: Section 2 provides a literature review of relevant research. Section 3 introduces the conceptual framework of the proposed method. Section 4 describes details of the proposed or applied algorithms. Section 5 presents two case studies to evaluate and verify the effectiveness of our method. Finally, Section 6 concludes the paper with a summary of findings and potential directions for future research.

2 LITERATURE REVIEW

Simulating fluid flow, especially airflow around target objects, has a wide range of potential application areas across various fields, such as aerospace engineering, the automotive industry, and wind energy systems. Ensuring the watertightness of a model is a critical step before conducting CFD simulations for studying fluid flow around an object [13]. A watertight model guarantees that the simulation domain is fully enclosed, preventing errors in mesh generation and in solving equations. In summary, validating the domain watertightness before a CFD simulation is necessary prior to attempting to solve for a simulated velocity field.

Over the past few decades, watertight validation of the digital geometric domain has been widely applied across diverse fields, including medical imaging [14], computer-aided design and manufacturing (CAD/CAM) [15], and building design with airflow simulations [16]. There are various methods developed to repair surface meshes of geometry files and ensure watertightness for CFD simulations. Early methods primarily focused on manual interventions, where users manually edited mesh defects using commercial CAD or mesh software such as MeshLab and Autodesk Meshmixer. These tools provide a user-friendly environment for mesh modification, allowing users to manually delete, merge, or smooth mesh regions to address issues such as holes, non-manifold edges, and self-intersections [17,18]. In commercial CFD workflows, software such as ANSYS SpaceClaim and Geomagic Design X offers similar capabilities, enabling engineers to repair meshes before exporting them for simulations. Despite their effectiveness in small-scale applications, manual interventions suffer from several limitations, including being labor intensive, time consuming, prone to human error, and lacking broad file format compatibility.

With increasing demands for efficient and robust mesh repair, smoothing, and simplification, researchers have developed semi-automated methods that reduce manual effort while still requiring some user intervention. Unlike fully manual approaches, semi-automated techniques integrate computational algorithms to detect and correct defects such as holes, non-manifold edges, and intersecting faces, allowing users to approve or refine modifications. One widely used technique is the graph-based hole-filling approach, first introduced by Liepa [19], which reconstructs missing surfaces based on neighboring edge connectivity. While effective for small, simple gaps, it struggles with large or complex holes, requiring manual refinement to prevent distorted surfaces. Attene et al. [20] proposed a feature-sensitive approach that corrects non-manifold edges while maintaining sharp features. This technique improves geometric accuracy, but users must manually adjust parameters depending on the shape complexity. Botsch et al. [12] proposed an interactive remeshing method for mesh smoothing and simplification, which uses local shape optimization to improve msh quality. However, it requires human verification to ensure topological integrity, as over-smoothing may lead to unintended geometry distortions. While these semi-automated methods have reduced the manual workload, they still rely on human intervention at critical stages, making them time-consuming for large-scale applications.

Recent research has explored fully automated approaches for surface mesh repair, attempting to eliminate manual intervention entirely. Some Poisson-based surface reconstruction techniques, such as those by Kazhdan et al. [10,21], attempt to create a fully continuous mesh by filling holes with an implicit surface function. However, such methods tend to over-smooth the geometry, leading to a loss of sharp features and undesired modifications to the original structure. Moreover, recent machine learning-based methods [22,23] claim to improve automation by predicting defect corrections based on large datasets. However, these approaches lack adaptability for complex or highly detailed engineering models, requiring pre-processing steps that reintroduce manual intervention. Additionally, AI-based methods demand large datasets for training, making them impractical for customized applications where input geometries vary significantly.

In summary, neither fully manual nor fully automated methods offer a complete solution for practical CFD preprocessing workflows. Current semi-automated approaches still require expert adjustments, and fully automated methods fail to generalize across diverse geometries. Critical limitations of current methods are summarized as follows:

- Lack of Integration – Most existing techniques focus on isolated aspects of mesh processing, such as hole filling or defects correcting, rather than a comprehensive framework that includes repair, validation, simplification and visualization.
- Limited File Format Compatibility – Some methods are designed for specific formats (e.g., STL or point clouds) and lack broad applicability to CAD-based and complex simulation-ready geometries.
- Automation Trade-offs – Fully automated methods risk oversimplifying meshes or altering geometric features, requiring post-processing adjustments. While semi-automated methods and manual geometry fixing methods require a lot of manual interventions and domain-specific expertise.
- Lack of Robustness – Many existing tools are not fault-tolerant. If a user accidentally performs incorrect operations (e.g., importing a corrupted file or removing boundary-defining surfaces), the system may fail silently or produce invalid results.

To address these challenges, we propose a robust and automated geometry preprocessing framework that integrates model repair, watertightness validation, and surface mesh simplification into a unified pipeline. The framework supports a wide range of input and output geometry file formats and incorporates fault-tolerant mechanisms with diagnostic feedback to enhance robustness. It preserves essential geometric features while reducing mesh complexity, enabling high-fidelity yet computationally efficient geometry file output for CFD simulations. In addition, the framework includes built-in visualization tools for interactive inspection and debugging, allowing users to verify mesh integrity and retain control throughout the preprocessing workflow. The conceptual framework of the proposed method is presented in Section 3.

3 CONCEPTUAL FRAMEWORK OF PROPOSED GEOMETRY PREPROCESSOR FOR CFD SIMULATION

The proposed robust and automated method for model repair, watertight validation, and surface mesh simplification is designed to function as a geometry preprocessor for CFD simulations. Its primary objective is to ensure that the output models are watertight, enabling them to serve as usable calculation domains in CFD simulation environments. The motivation behind this development is to create a generic and practical tool that can be seamlessly integrated into CFD workflows, specifically for simulating flow around target objects.

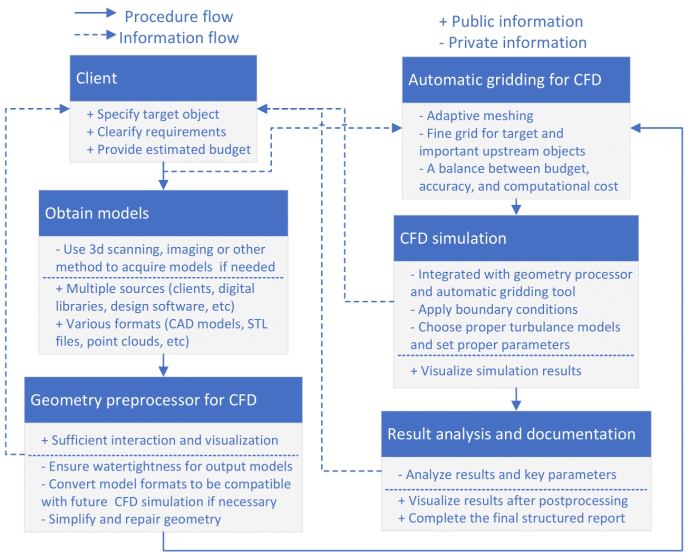
In this section, we first present the overall workflow of simulating fluid flow around a target object, outlining the key stages of a typical CFD pipeline—from geometry acquisition to simulation results and documentation. Particular emphasis is placed on the role of the proposed geometry preprocessor, which serves as a critical geometry processing tool that ensures surface integrity, watertightness, and compatibility with downstream domain meshing and solver tools. Following the high-level overview, we introduce the detailed architecture and functional modules of the proposed geometry preprocessor, illustrating how it addresses common preprocessing challenges through the integration of input geometry file reading, model repair, surface mesh simplification, watertight validation, and output geometry file writing.

3.1 Workflow of Simulating Flow Around a Target Object

Figure 1 shows the overall workflow for simulating flow around a target object, which involves six key steps. The workflow begins with the client, who specifies the object of interest and clarifies simulation goals such as performance evaluation, aerodynamic behavior, or environmental interaction. At this initial stage, the client also provides an estimated budget, which serves as a constraint that influences both modeling resolution and computational cost throughout the workflow.

Once the project scope is defined, the next stage involves obtaining digital models of the target object and its environment. These models may be sourced from the client directly, retrieved from digital libraries, or generated using CAD software. In cases where digital representations are unavailable or incomplete, 3D scanning, photogrammetry, or imaging technologies may be employed to capture accurate geometric data from physical objects. The acquired models often come in a variety of formats, such as CAD models (STEP, IGES), triangulated meshes (STL, OBJ), or point clouds, and may include defects such as holes, overlaps, or topological inconsistencies that make them unsuitable for direct CFD simulation.

Following model acquisition, the geometry is passed into the proposed geometry preprocessor (see Figure 2), which is the core focus of our proposed method. Automated watertightness validation and repair are particularly advantageous in workflows that involve massive repeated simulations, such as parametric design studies, urban airflow mapping, or optimization-based shape evaluations. In such cases, the ability to preprocess multiple models quickly and consistently without extensive manual correction significantly reduces turnaround time and human error. The framework is also well-suited for turnkey analysis pipelines, where engineers or automated scripts routinely generate CFD-ready geometries from design iterations or scanned datasets. The preprocessor performs automated repair and simplification, validates watertightness, and reformats the geometry to meet the requirements of CFD simulation environment. By automating these processes to the greatest extent possible, the preprocessor significantly reduces manual workload while enhancing the robustness and repeatability of the entire simulation pipeline.



Procedure flow
Information flow
+ Public information
- Private information
Client
+ Specify target object
+ Clearify requirements
+ Provide estimated budget
Automatic gridding for CFD
- Adaptive meshing
- Fine grid for target and important upstream objects
- A balance between budget, accuracy, and computational cost
Obtain models
- Use 3d scanning, imaging or other method to acquire models if needed
+ Multiple sources (clients, digital libraries, design software, etc)
+ Various formats (CAD models, STL files, point clouds, etc)
CFD simulation
- Integrated with geometry processor and automatic gridding tool
- Apply boundary conditions
- Choose proper turbulence models and set proper parameters
+ Visualize simulation results
Geometry preprocessor for CFD
+ Sufficient interaction and visualization
- Ensure watertightness for output models
- Convert model formats to be compatible with future CFD simulation if necessary
- Simplify and repair geometry
Result analysis and documentation
- Analyze results and key parameters
+ Visualize results after postprocessing
+ Complete the final structured report

Figure 1: The overall workflow of simulating flow around a target object.

With a clean and validated geometry prepared, the workflow proceeds to the automatic gridding stage, where a volumetric mesh is generated for the computational domain. The meshing process is adaptive, assigning finer resolution to regions near the target object and key upstream flow areas, while coarsening regions of less importance to maintain computational efficiency. Grid density is carefully balanced against accuracy, computing power, and budgetary constraints.

Once the geometry is meshed, the CFD simulation is carried out using commercial CFD software such as ANSYS Fluent or open-source platforms such as OpenFOAM. At this stage, appropriate boundary conditions are applied to the simulation domain, and turbulence models—such as Reynolds-Averaged Navier–Stokes (RANS) or Large Eddy Simulation (LES)—are selected based on the flow characteristics and simulation goals. Solver parameters are configured to ensure numerical convergence and stability throughout the computation. In some cases, real-time visualization tools are employed during the simulation to monitor airflow behavior and assess solver performance, allowing early identification of anomalies or instabilities.

Following the simulation, the results are processed and analyzed. Key outputs, such as pressure distributions, velocity fields, and vorticity structures, are extracted and visualized using post-processing tools such as ParaView. These findings are then compiled into a comprehensive report that documents the simulation setup, methodology, and insights derived from the results. The structured report is tailored for stakeholders and decision-makers, enabling them to translate the analysis into actionable conclusions for engineering optimization, design refinement, or policy guidance.

Among all these steps, the geometry preprocessing phase stands out as a frequent bottleneck, especially when handling non-ideal or complex geometries. Our proposed framework addresses this challenge through a robust and automated solution. The detailed workflow of the proposed geometry preprocessor is presented in Section 3.2.

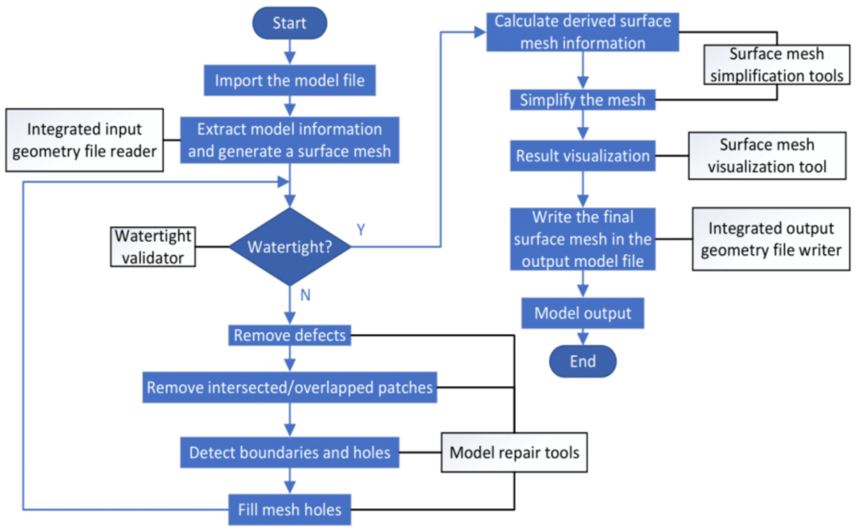

Start
Import the model file
Integrated input geometry file reader
Extract model information and generate a surface mesh
Watertight validator
Watertight?
Y
N
Remove defects
Remove intersected/overlapped patches
Detect boundaries and holes
Model repair tools
Fill mesh holes
Calculate derived surface mesh information
Surface mesh simplification tools
Simplify the mesh
Result visualization
Surface mesh visualization tool
Write the final surface mesh in the output model file
Integrated output geometry file writer
Model output
End

Figure 1: Detailed workflow of the proposed geometry preprocessor for CFD simulation.

3.2 Detailed Workflow of the Proposed Geometry Preprocessor

The internal architecture of the proposed geometry preprocessor is designed to provide an automated, modular, and robust solution for preparing geometry files for CFD simulation. Figure 2 presents the detailed workflow of the proposed CFD geometry preprocessor, starting from raw model input to the final export of a defect-free, watertight, validated, and simplified geometric model.

The process begins with importing the model file through the integrated input geometry file reader, which supports various formats. The file reader reads input geometry files and extracts essential geometric data, including vertices and triangle faces to generate a triangular surface mesh.

After obtaining the triangular surface mesh, the system performs a defect removal routine to clean up the raw geometry. Common defects include duplicate vertices, duplicate faces, unreferenced vertices, degenerate faces (i.e., faces with near-zero area), and non-manifold edges. This defect removal step ensures that the surface mesh is cleaned up before topological analysis or repair is conducted.

Next, the watertight validator examines the cleaned surface mesh to determine whether it forms a closed, watertight surface. If the mesh is watertight, the workflow proceeds directly to mesh simplification. If not, the system activates the model repair routine.

In the repair sequence, intersected or overlapped surface patches are removed, and boundary edges and holes are detected using connectivity-based algorithms. Once holes are identified, they are filled using curvature-aware triangulation to preserve local surface continuity and shape fidelity. The repaired surface mesh is then cleaned up and re-evaluated for watertightness.

With a clean and watertight surface mesh established, the system calculates derived geometric attributes to support mesh simplification. The simplification process reduces the total number of vertices and faces while retaining sharp edges, ridges, and other critical features through feature-sensitive clustering algorithms. This optimization lowers the computational load during simulation without compromising mesh quality in important flow regions.

Finally, the simplified mesh is visualized in the interactive viewer, allowing users to inspect modifications, confirm surface continuity, and validate repair quality. The final output is then written to the selected format using the integrated output file writer, completing the preprocessing pipeline and generating a mesh that is ready for CFD meshing and simulation.

As illustrated in Figure 2, the proposed preprocessor consists of six key components working together as a comprehensive and integrated tool. These include an integrated input geometry file reader, a watertight validator, model repair tools, surface mesh simplification tools, a surface mesh visualization tool, and an integrated output geometry file writer. Each component plays a specific role within the overall pipeline, enabling the system to handle diverse input formats, detect and repair geometric defects, simplify complex models while preserving essential features, and deliver simulation-ready surface meshes with minimal user intervention. To further clarify its functionality, the following section presents a detailed description of the core algorithms proposed or applied in some of these key components.

4 ALGORITHMIC FOUNDATIONS OF THE GEOMETRY PREPROCESSOR

This section provides the underlying algorithms that enable the proposed geometry preprocessor to function as a robust, automated, and feature-preserving tool for CFD model preparation. Each processing stage—ranging from geometry file reading/writing to model repair and surface mesh simplification—is built upon carefully designed or applied geometry processing algorithms. These algorithms are developed or applied to balance robustness, automation level, efficiency, and feature preservation. The following subsections describe the algorithmic strategies, decision criteria, and geometric operations implemented in each of the preprocessor’s core modules.

4.1 Integrated Geometry File Reader and Writer

The preprocessing pipeline begins and ends with flexible file handling capabilities, implemented through an integrated geometry file reader and writer. These modules enable seamless compatibility between diverse geometry sources and CFD preprocessing requirements, supporting a range of commonly used formats in engineering workflows.

The input file reader is designed to support a wide variety of geometry sources, categorized into four primary types: triangulated surface meshes, CAD-based solid models, scene or exchange formats, and point clouds. Supported triangulated mesh formats include STL, PLY, OBJ, OFF, and GLTF/GLB. For solid modeling workflows, the reader supports STEP/STP and IGES/IGS, which are commonly used in CAD applications. Additionally, scene formats such as FBX and DAE are supported to accommodate models originating from 3D modeling software or mixed digital pipelines. In the case of point clouds, the system applies a surface reconstruction algorithm — Delaunay triangulation — to generate an initial mesh that approximates the scanned geometry.

After the model is imported, the file reader extracts essential geometric data—primarily vertices and faces—that define the surface mesh. Each vertex is recorded with its 3D coordinates, while each face is defined by a triplet of vertex indices forming a triangle. This indexed structure allows for efficient storage and manipulation of mesh topology. Figure 3 illustrates a simplified triangulated cube, where both vertex IDs and face IDs are visualized. In this example, the green-highlighted triangle represents face f_1, which is constructed from vertices v_1, v_2, and v_3. A corresponding data table is shown in Tables 1 and 2. This representation is essential for detecting topological issues, performing mesh operations such as hole filling or simplification, and enabling visualization tools to interact with the geometry in a high efficiency.

Conversely, the integrated file writer saves the final surface mesh as a selected output format, ensuring compatibility with a wide range of CFD simulation environments. The writer supports most of the same formats as the reader, including STL, OBJ, PLY, and other triangulated mesh formats. However, it does not support export to STEP/STP or IGES/IGS. This is due to a limitation inherent in the workflow: when surface information is extracted from CAD-based solids, the parametric definitions of the original model, such as design constraints, are lost. As a result, the processed triangular mesh—now a purely geometric approximation—cannot be re-converted into a parametric solid suitable for those CAD formats. Nevertheless, the supported output formats are sufficient for seamless integration into CFD mesh generators such as snappyHexMesh or blockMesh, making the file writer a practical endpoint for the preprocessing pipeline.

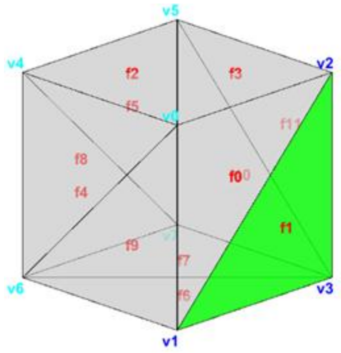


Figure 3: The surface mesh of an object (cube) with a highlighted triangle face and its vertices.

Vertex	*X*	*Y*	*Z*
v0	0	1	1
v1	0	1	0
v2	0	0	1
v3	0	0	0

Table 1: The sample data set of vertices.

Face	*Vertex1*	*Vertex2*	*Vertex3*
f0	v0	v1	v2
f1	v2	v1	v3
f3	v4	v0	v5
f4	v5	v0	v2

Table 2: The sample data set of triangle faces.

4.2 Watertight Validator

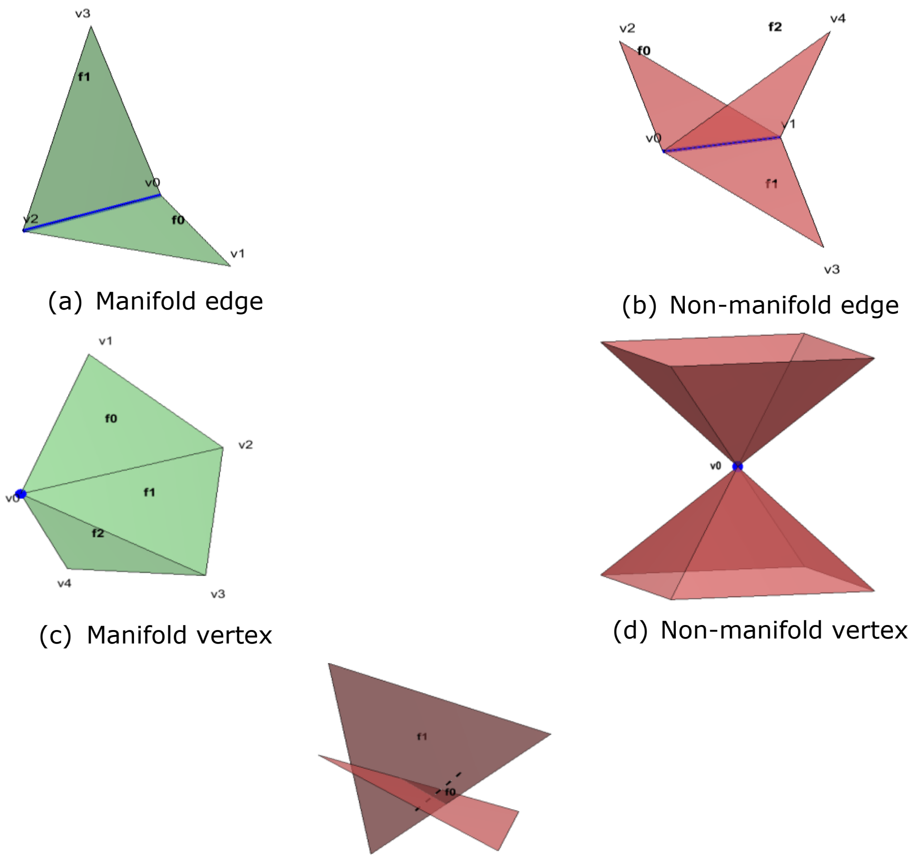
The Watertight Validator is a critical component of the proposed geometry preprocessor, responsible for assessing whether the input surface mesh meets the criteria necessary for successful CFD simulation. A mesh is considered watertight when it satisfies three key conditions: (1) no non-manifold edges, meaning every edge is shared by exactly two faces; (2) no non-manifold vertices, where each vertex is involved in a continuous and unambiguous local surface topology; and (3) no self-intersections, ensuring the mesh does not intersect itself in any region [24]. Figure 4 provides visual illustrations of valid and invalid cases for edge manifoldness, vertex manifoldness, and self-intersection/overlapping.

The validation begins with a topological analysis. An edge-face adjacency map is constructed to identify all edges in the mesh and count how many faces share each edge. If any edge is used by fewer or more than two faces, it is flagged as non-manifold, and the mesh is marked as invalid.

Similarly, the validator checks vertex-manifoldness by examining the connectivity of faces surrounding each vertex. A valid manifold vertex should be surrounded by a single, continuous loop or fan of triangles. Vertices shared by disjoint face clusters indicate non-manifold vertices and may result in ambiguities during surface-to-volume meshing.

The next stage involves geometric validation, where the mesh is tested for self-intersections. This is achieved using efficient triangle-triangle intersection detection based on bounding volume hierarchies (BVH) [25]. If any pair of triangles intersect improperly (i.e., not at shared edges or vertices), the mesh is considered self-intersecting and flagged for repair.

If all three conditions are satisfied, the Watertightness Validator confirms the model as watertight, and it proceeds to the mesh simplification stage. Otherwise, the validator generates detailed error feedback, including the number and type of defects detected. This feedback is passed to the model repair module for corrective processing. The validator is designed to operate in a fully automated fashion, while also offering interactive visual cues to users—such as marking non-manifold vertices or highlighting intersecting patches.



(a) Manifold edge

(b) Non-manifold edge

(c) Manifold vertex

(d) Non-manifold vertex

(e) A self-intersection case for 2 patches

Figure 4: Illustration of Mesh Watertightness Conditions (Manifold vs. Non-manifold Topologies and Self-Intersection): (a) A valid manifold edge shared by two faces, (b) A non-manifold edge shared by more than two faces, (c) A manifold vertex surrounded by a continuous fan of faces, (d) A non-manifold vertex where disconnected face clusters share a single point, (e) A self-intersection case in which two surface patches intersect improperly in 3D space.

The pseudo code of the watertight validator is as follows:

```
Input: Surface mesh $M = (V, F)$ with vertices $V$ and faces $F$
Output: Boolean flag isWatertight, and defect report $D$
Initialize defect report $D = \emptyset$

Construct edge-to-face map $E2F$ from $F$
For each edge $e$ in $E2F$:
    If the number of faces sharing $e \neq 2$:
        Mark $e$ as non-manifold and add to $D$

Construct vertex-to-face map $V2F$ from $F$
For each vertex $v$ in $V2F$:
    If faces around $v$ do not form a connected fan:
        Mark $v$ as non-manifold and add to $D$

Build a spatial index for all triangles in $F$
For each triangle $f_i$ in $F$:
    Query potentially intersecting triangles $f_j$
    For each candidate $f_j \neq f_i$:
        If $f_i$ and $f_j$ intersect geometrically:
            Mark $(f_i, f_j)$ as self-intersecting in $D$

If $D$ is empty:
    Return true, $\emptyset$
Else:
Return false, $D$
```

4.3 Model Repair Module: Automated Fixing of Geometry with Defects

Following watertightness validation, surface meshes flagged as non-watertight are passed to the Model Repair module, which automatically corrects the detected issues. The validator provides detailed diagnostic information, including non-manifold vertices, non-manifold edges, and self-intersecting triangles. Based on this, the repair module applies a series of corrective operations to restore mesh validity and prepare the geometry for downstream processing.

The repair process begins with the removal of low-level structural inconsistencies from the mesh. These include duplicate vertices, which are merged based on positional tolerance, and duplicate faces, which often result from redundant modeling or import errors. Unreferenced vertices, which are not associated with any face, and degenerate faces, where two or more vertices are coincident or collinear, are also identified and removed. Additionally, non-manifold edges—those shared by more than two faces or only one—are resolved through edge splitting or localized removal and remeshing. This step ensures that the mesh has a consistent and interpretable structure before more complex geometric repairs are applied.

To address self-intersecting and overlapping triangles, the system applies a targeted and iterative strategy that prioritizes minimal geometric disruption. An intersection matrix is constructed by performing pairwise triangle-triangle intersection tests using BVH. Each face is then ranked based on a proposed score function S, which takes into account its surface area, the number of intersections it participates in, and its local mean curvature (see Equation (4.1)). Triangles with the highest score values are prioritized to be removed, because those triangles with small areas, low local mean curvatures, and high intersection counts are considered more likely to be redundant or erroneous, and are prioritized for removal. Based on our testing and usage experience, the default values of their weighting coefficients are set as λ_1=0.2, λ_2=0.4, λ_3=0.4 by default. To prevent overcorrection and maintain the original shape of the geometry, only a small number of faces are removed in each iteration, typically those getting the highest value of the proposed score function. After each removal step, the intersection matrix is updated, and the process is repeated until no significant intersections remain. This approach ensures geometric consistency while minimizing unintended alterations to the mesh.

For each intersected triangle f_i, the score function S is defined as:

$$\mathrm{S}(f_i) = \lambda_1 \cdot \left(1 - \frac{A_i}{\max\limits_j A_j + \varepsilon}\right) + \lambda_2 \cdot \left(\frac{N_i}{\max\limits_j N_j + \varepsilon}\right) + \lambda_3 \cdot \left(1 - \frac{|\kappa_i|}{\max\limits_j |\kappa_j| + \varepsilon}\right) \tag{4.1}$$

Where the explanation of each symbol is explained in
Table

Symbol	Meaning
f_i	The i-th triangle face in the surface mesh
A_i	The area of triangle f_i
$\max\limits_j A_j$	The maximum triangle area among all faces in the mesh
N_i	The number of triangles that intersect with f_i
$\max\limits_j N_j$	The maximum number of intersections among all faces
$\lvert\kappa_i\rvert$	The absolute value of mean curvature associated with f_i
$\max\limits_j \lvert\kappa_j\rvert$	The maximum absolute value of mean curvature among all faces
ε	A small constant added to the denominator to avoid division by zero
λ_1, λ_2, λ_3	Weighting coefficients that balance the importance of area, intersection density, and curvature in the scoring function. $\lambda_1+\lambda_2+\lambda_3=1$, where $\lambda_i \in [0,1]$.

Table 3: Symbol definitions of the proposed scoring function in Equation (4.1).

Once geometric conflicts are resolved, the system analyzes the mesh for open boundaries and holes. Boundary edges, defined as edges referenced by only one face, are detected using the edge-to-face adjacency map. These are then grouped into continuous edge loops. To distinguish between holes and true model boundaries, each loop is classified based on multiple geometric and contextual cues. These include the loop size, curvature distribution, planarity, and location relative to the model's bounding box. Small, internal, and near-planar loops are more likely to be true holes, while large or externally facing loops are more likely to be intentional outer boundaries. This classification helps avoid over-repairing intentionally open surfaces and ensures that only topologically problematic regions are passed to the hole-filling stage. These categorized holes are then further analyzed to support adaptive repair strategies in the final stage. This distinction helps ensure that minor surface gaps are filled without modifying intentional geometry features and that large, unintended openings—which may impact flow domain definition—are properly addressed.

Once holes have been identified and classified, the system proceeds to automatically fill them using context-aware triangulation strategies. For small or near-planar holes, a simple fan triangulation method is employed, where the loop is filled by connecting boundary vertices to a computed centroid or local reference point. This approach is efficient and introduces minimal distortion in flat or mildly curved regions. For larger or geometrically complex holes, the system applies a graph-based triangulation algorithm that constructs a constrained surface mesh over the boundary loop [19]. This method considers local edge length, angle smoothness, and curvature continuity to produce a well-shaped patch with minimal deviation from the surrounding geometry. When the hole is adjacent to sharp features or high-curvature areas, feature-preserving constraints are enforced to retain corner continuity and edge sharpness. By adapting the triangulation strategy based on hole complexity and context, this process restores topological watertightness while maintaining surface integrity and geometric fidelity.

The model repair workflow operates in an iterative loop, where each round of repairs is followed by a re-evaluation using the watertightness validator. If the mesh still contains defects—such as residual non-manifold features or newly introduced inconsistencies—the corresponding repair procedures are automatically invoked again. This cycle continues until all watertightness conditions are satisfied. By tightly coupling validation and repair in this feedback-driven loop, the framework ensures robust convergence toward a watertight model while minimizing the risk of overcorrection. The entire process is fully automated and requires little to no manual intervention, making it highly suitable for batch processing and integration into large-scale CFD preprocessing pipelines.

4.4 Surface Mesh Simplification

Once a watertight and topologically consistent mesh has been reconstructed, the next step in the preprocessing workflow is to reduce surface mesh complexity through simplification. This is especially important for CFD applications, where overly dense surface meshes can significantly increase computational cost during volume meshing and solver runtime without improving simulation accuracy.

The simplification module in the proposed geometry preprocessor is designed to reduce the number of faces while maintaining the essential geometric features of the original model. The process begins by evaluating local mesh density and identifying candidate regions for decimation. Areas with low curvature and uniform triangle shape are considered high-priority for simplification, while high-curvature regions, sharp edges, and boundary-adjacent zones are preserved to maintain flow-relevant geometry.

To achieve this, a feature-sensitive edge collapse algorithm is employed. Before simplification begins, the system calculates derived surface mesh information, including edge length, face normals, and local curvature. Here, we propose a cost function C that prioritizes which edges to collapse first based on this geometric context (see Equation (4.2)). Using the derived mesh information, the collapse cost of each edge is computed through a weighted function that incorporates: edge length (shorter edges are prioritized for collapse), local curvature (flatter areas are simplified more aggressively), and feature preservation constraints (edges marked as part of feature lines or user-defined flow-sensitive regions are protected).

For each edge e_i, the score function C is defined as:

$$\mathrm{C}(e_i) = \mu_1 \cdot \left(\frac{l_i}{\max\limits_j l_j + \varepsilon}\right) + \mu_2 \cdot \left(\frac{|\kappa_i|}{\max\limits_j |\kappa_j| + \varepsilon}\right) + \mu_3 \cdot \delta_i \tag{4.2}$$

Where l_i is the edge length, κ_i is the mean curvature of the edge, ε is a small constant added to the denominator to avoid division by zero, δ_i is the feature-preserving value defined by the user, and μ_1, μ_2, μ_3 are weighting coefficients with $\mu_1+\mu_2+\mu_3=1$ and $\mu_i \in [0,1]$. The default values are set as $\mu_1=0.4$, $\mu_2=0.4$, $\mu_3=0.2$ based on our testing and usage experience.

The simplification process is executed iteratively, evaluating and collapsing a small number of edges per iteration while continuously updating curvature and adjacency information. At each step, the derived surface mesh information—including edge length, curvature, and feature sensitivity—is recalculated to reflect the latest geometry. A cost function is evaluated for all candidate edges, and those with the lowest cost values are prioritized for collapse. This ensures that simplification begins in flatter, less feature-sensitive regions. This loop continues until the total number of surface patches is reduced to a user-defined target percentage, ensuring that simplification progresses in a controlled and adaptive manner. This approach allows for smooth convergence toward a lower-resolution mesh while avoiding topology changes or the introduction of geometric artifacts.

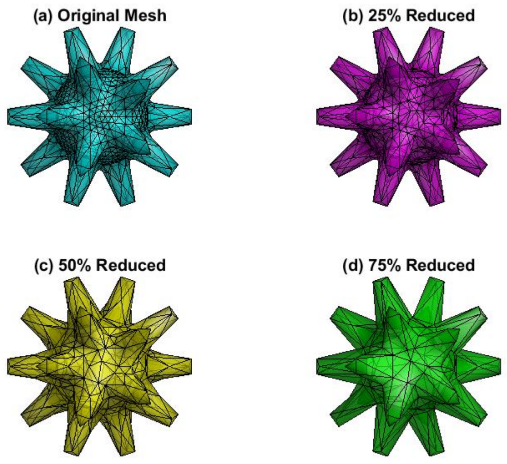


Figure 5: A case of mesh simplification result on a star-shaped geometry: (a) The original mesh, Simplified meshes with (b) 25% patch reduction, (c) 50% patch reduction, (d) 75% patch reduction.

The simplification is both adaptive and interactive. Users can set a target patch removal rate, curvature preserving threshold, or local refinement zones to control the outcome. For CFD applications involving boundary layers or flow-sensitive geometries, simplification can be selectively restricted to background regions, preserving detail near inlets, outlets, or complex surfaces.

The resulting simplified surface mesh achieves a balanced trade-off between computational efficiency and geometric fidelity, enabling accurate and efficient CFD simulations with significantly reduced memory consumption and solver time.

To illustrate the effect of the proposed simplification algorithm, Figure 5 presents a visual demonstration of a complex star-shaped model. The original mesh contains fine geometric features and a dense triangle distribution. As simplification progresses, with 25%, 50%, and 75% of the patches removed, the overall shape is retained while the mesh becomes increasingly coarser. The method successfully preserves sharp spikes and feature-rich regions by prioritizing edge collapses in flatter, low-curvature areas. This result highlights the algorithm's ability to reduce computational burden while maintaining geometric fidelity, making it highly suitable for downstream CFD simulations.

5 CASE STUDY

To demonstrate the effectiveness and versatility of the proposed geometry preprocessing framework, we present two representative case studies: (1) the Stanford Bunny [26], a widely used benchmark model with defects, and (2) a real-world campus building geometry, which features structural details. The two case studies, including a synthetic benchmark model and a real-world architectural geometry, were selected to demonstrate the versatility and robustness of the proposed method in both standard mesh repair benchmarks and practical applications relevant to CFD analysis.

Both case studies are processed using a custom application, developed in MATLAB as part of this research. The application integrates the proposed algorithms with MATLAB's Lidar Toolbox and additional tools from an open-source mesh processing toolbox [27]. The application is designed with a user-friendly graphical interface (GUI) to enhance interaction and usability (see Figure 6). Users can load surface models in multiple formats, select relevant mesh processing libraries, visualize detected defects, interactively validate watertightness, apply repair operations, and simplify the geometry with real-time feedback, and export of watertight output models in CFD-compatible formats such as STL.

The application features built-in watertight indicators designed to provide real-time feedback on the mesh's topological integrity during the preprocessing workflow (see Figure 6). This indicator visually communicates whether the originally loaded or processed model satisfies the watertightness criteria required for CFD simulation. A red indicator denotes that the mesh is not watertight, typically due to holes, non-manifold edges or vertices, or intersecting triangles. Once the repair procedures are successfully applied and all validation checks are passed, including edge-manifold, vertex-manifold, and self-intersection tests, the indicator turns green, confirming that the model is now watertight and suitable for downstream simulation. This visual tool enhances usability by helping users immediately assess mesh validity at different stages, reducing the likelihood of exporting defective models and streamlining the geometry preparation process.

To enhance the robustness and user experience of the developed application, each stage of the preprocessing workflow is equipped with interactive prompts that provide real-time feedback and enforce logical progression. These include instructional messages, error alerts, and success confirmations, which help prevent user mistakes and ensure proper execution of each operation. For example, when an essential step, such as computing derived surface mesh information (step 3), is about to be executed, the application displays an instructional dialog explaining its significance and prompting the user for confirmation (Figure 7 (a)). If the user attempts to bypass a required step, an error message is triggered (Figure 7 (b)), informing them of the missing input and prompting corrective action. Upon successful completion of a process, a confirmation message is shown (Figure 7 (c)), allowing users to proceed confidently to the next stage. These feedback mechanisms contribute to the overall fault tolerance of the system and improve the reliability of the automated workflow. By embedding such prompts into the interface, the application supports both novice and advanced users, reducing the likelihood of incomplete or invalid operations and promoting a seamless and guided CFD geometry preprocessing experience.

To assess the computational efficiency of the proposed framework, all cases were conducted on a conventional Windows 11 laptop system equipped with an Intel Core i5-7200U processor (2.50 GHz) and 32 GB of RAM, using MATLAB 2024b. No GPU acceleration or multithreading was applied. This setup reflects a modest computing environment, demonstrating that the method is lightweight and can be readily executed on standard research or engineering machines without specialized hardware.

The repaired surface meshes were also tested for compatibility with CFD tools and were successfully loaded into OpenFOAM for volume mesh generation using snappyHexMesh, confirming that the output models are suitable for downstream meshing and simulation workflows.

5.1 Case 1: Automatic Repair of Stanford Bunny

The first case study validates the proposed geometry preprocessing framework using the well-known Stanford Bunny, a benchmark 3D model frequently used in mesh repair and geometry processing research. For this case study, the OBJ file of the Stanford Bunny [26] is used as the input model. The input is non-watertight and contains several holes and other defects, particularly around the base. After applying the proposed method, holes are filled, defects are removed, and a watertight STL file is successfully generated as the output model (see Figure 8).

The defective OBJ file (see Figure 8(a)) was imported into our developed MATLAB application. Upon loading the model, the Watertight Validator identified multiple open holes and non-manifold edges, primarily located at the base of the bunny. These defects were clearly marked in the visualization panel (Figure 8(b)) and confirmed by the red watertight indicator, prompting the user to begin the mesh repair process.

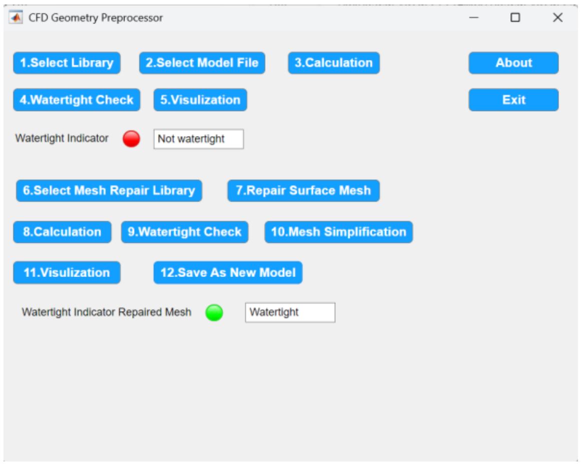
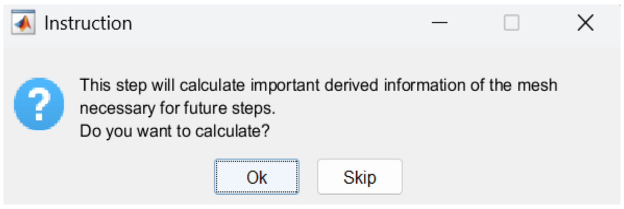
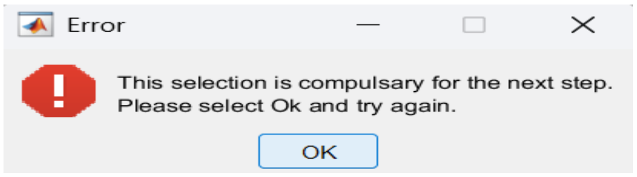


Figure 6: The customized UI of the proposed CFD Geometry Preprocessor.


Instruction

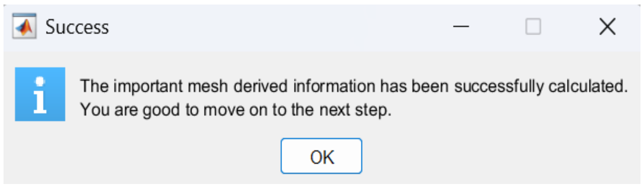
This step will calculate important derived information of the mesh necessary for future steps.
Do you want to calculate?

(a) The introduction hint of step 3


Error

This selection is compulsary for the next step.
Please select Ok and try again.

(b) An error message of incorrect operation of step 3


Success

The important mesh derived information has been successfully calculated.
You are good to move on to the next step.

(c) A success confirmation message of step 3

Figure 7: An example of GUI feedback prompts used to improve robustness and user guidance during the CFD geometry preprocessing workflow: (a) Instructional message providing hint context before executing a key operation, (b) Error prompt indicating an incorrect operation, (c) Success confirmation.

The repair module was triggered through an intuitive graphical user interface with step-by-step guidance, initiating a highly automated workflow. The tool removed duplicate and unreferenced vertices, degenerate triangles, and non-manifold edges, followed by a hole-filling process. The application allows users to interactively visualize both the detected issues and the corrected geometry (Figure 8(c)) at each step using built-in surface mesh viewers and status prompts.

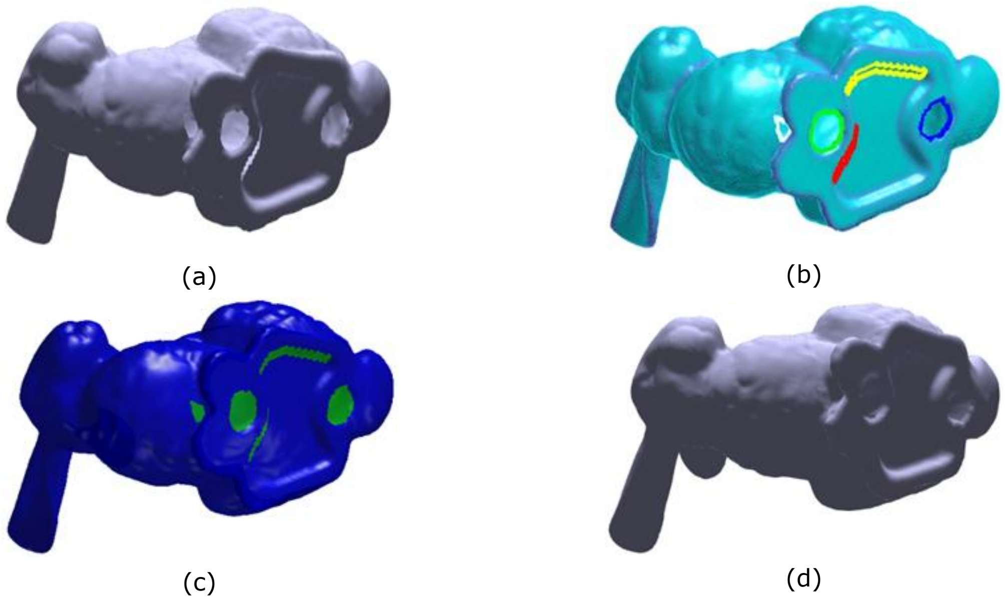


Figure 8: Geometry preprocessing of Stanford Bunny for CFD simulation: (a) Original model (.obj), (b) Extracted surface mesh with holes detected and marked, (c) Processed surface mesh with holes filled and defects removed, (d) Final output model (.stl) for future CFD simulation with watertight validation.

After the repair process, the model was re-evaluated and successfully met all three watertightness criteria: edge-manifold, vertex-manifold, and free of self-intersections (Figure 8(d)). Its status was updated to Watertight, as shown by the green LED indicator in the GUI. The finalized mesh was then exported as an STL file, ready for use in CFD simulation.

In this case, the input model comprised 34,834 vertices and 69,664 triangular faces. The watertightness validator identified 5 open holes and 2 intersecting triangle regions, which were successfully repaired. The complete preprocessing—including defect removal, hole filling, and validation—was completed in 32.9 seconds, yielding a watertight surface mesh. This case study demonstrates how the proposed framework transforms a defective, non-water-tight scanned model into a clean and simulation-ready geometry file. The repair process is highly automated, supported by interactive visual guidance and real-time feedback, minimizing manual intervention while ensuring consistent results.

5.2 Real-World Building Model: Automated Surface Mesh Fixing and Simplification

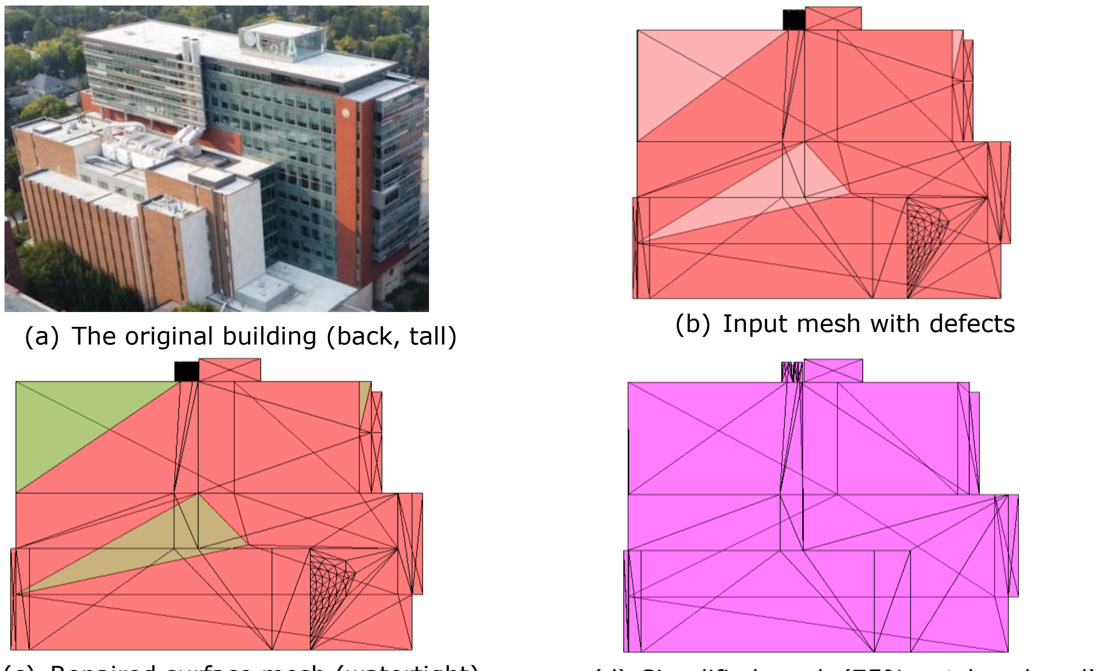
The second case study evaluates the robustness and practical applicability of the proposed geometry preprocessing framework using a real-world architectural model of the Donadeo Innovation Centre for Engineering at the University of Alberta, Canada (see Figure 9(a)). This tall, multi-faceted structure presents complex geometric characteristics that are typical of urban building models used in environmental and wind engineering simulations.

The original input was a scanned surface mesh in STL format. As shown in Figure 9(a), the building includes multiple vertical and horizontal geometry elements, as well as roof equipment and layered structures. Upon importing the model into the MATLAB application, the Watertight Validator automatically identified and marked these topological and geometric issues. The raw input surface mesh, illustrated in Figure 9(b), suffers from a range of common defects, including intersecting triangles, unreferenced vertices, and holes.

The model was then processed through the automated repair pipeline, which performed several corrective routines, including the removal of unreferenced vertices, deletion of intersecting triangle faces, and filling of mesh holes. As shown in Figure 9(c), the resulting surface mesh was fully watertight, passing all validation checks.

Following successful repair, the model underwent surface mesh simplification, targeting a 75% patch reduction while preserving key structural details. The simplification algorithm prioritized curvature preservation and structural fidelity, selectively removing patches in large planar regions and areas with overly dense mesh distributions, while retaining sharp features, corners, and flow-sensitive geometry critical for CFD simulation accuracy. The final simplified mesh is shown in Figure 9(d).

The building model, representing the Donadeo Innovation Centre for Engineering, had an initial mesh of 407 vertices and 790 faces. During preprocessing, 20 boundary holes and 5 intersected faces were detected and resolved. The repaired surface consisted of 810 faces, which was then simplified to a final mesh with 103 vertices and 201 faces. The entire processing sequence, including validation, repair, and simplification, was completed in just 0.55 seconds, illustrating the framework's responsiveness and efficiency when applied to real-world, mid-sized architectural geometry. The application's combination of automation, visualization, and robustness is particularly well-suited for preparing architectural data for urban airflow or ventilation studies, ensuring consistent and repeatable preprocessing with minimal user intervention.


(a) The original building (back, tall)

(b) Input mesh with defects

(c) Repaired surface mesh (watertight)

(d) Simplified mesh (75% patch reduced)

Figure 9: Geometry preprocessing of the Donadeo Innovation Centre for Engineering at the University of Alberta: (a) Original building reference image (back, tall structure), (b) Input surface mesh with holes and defects, (c) Repaired watertight mesh after defect removal and hole filling, (d) Simplified mesh with approximately 75% of patches removed while preserving geometric features critical to CFD simulation.

6 CONCLUSION AND FUTURE WORK

This study presented a robust, automated, and user-guided geometry preprocessing framework designed to automatically check and repair 3D geometry models to ensure watertightness for CFD simulations. The method addresses key challenges in converting raw geometry files—often defective, non-watertight—into watertight, simplified, and simulation-ready models. The framework integrates multiple components, including a multi-format file reader/writer, a watertightness validator, a comprehensive model repair module, surface mesh simplification algorithms, and interactive visualization tools. The proposed CFD geometry preprocessing framework is implemented in a developed application in MATLAB.

The proposed framework offers a robust and automated solution for geometry preprocessing, with several distinct advantages that make it suitable for CFD workflows involving complex and imperfect geometry models. It supports a wide range of input and output formats, such as STL, OBJ, STEP, and point clouds, allowing broad compatibility with diverse sources of geometry files. Robustness is achieved through the framework’s ability to detect and handle invalid or corrupted inputs, recognize incorrect operations, collect errors, while offering clear diagnostic messages and guidance to help users resolve issues effectively. All core preprocessing tasks, including defect detection, model repair, watertightness validation, and mesh simplification, are executed within an integrated pipeline with high automation level that reduces manual effort while preserving user control. Additionally, interactive visualization tools enable users to inspect mesh quality, review identified issues, and verify repairs in real time, enhancing transparency and usability throughout the workflow.

The effectiveness of the proposed method was validated through two case studies. The first, based on the Stanford Bunny model, demonstrated the framework's ability to detect and repair holes and other types of defects in a benchmark case. The second case, involving a real-world building model of the Donadeo Innovation Centre for Engineering, illustrated the framework’s practical application value when processing large-scale architectural data with complex defects. In both cases, the output model was successfully repaired, validated as watertight, and simplified while preserving key geometry features, which could be ready for CFD simulation.

In the future, the authors would like to explore the development of alternative platforms to implement similar functionalities in a completely open-source environment, such as Python, enhancing accessibility and adaptability for all users. Efforts will also focus on expanding the extracted surface model from triangle meshes to polygon meshes, enabling greater versatility in handling complex geometries. Additionally, the method will be tested on a wider range of cases, particularly models of real-world building groups, to validate its effectiveness and robustness in batch processing for large dataset models. Finally, we plan to integrate volume meshing and basic CFD solver setup capabilities into the application to enable an end-to-end simulation preparation environment.

In future work, we also plan to conduct a rigorous benchmarking study to evaluate the performance of the proposed method against both open-source and commercial mesh repair tools. This will include controlled comparisons based on metrics such as repair accuracy, processing time, mesh quality, and compatibility with CFD solvers. By applying standardized datasets and reproducible test cases, we aim to provide a clearer understanding of the method’s strengths and limitations relative to existing solutions, and to identify opportunities for further optimization and extension.

ACKNOWLEDGEMENTS

The authors gratefully acknowledge Flexible Machines Corporation (https://www.flexiblemachines.ca/) for offering insights into potential development frameworks for future HVAC and wind energy systems in urban buildings. This served as the primary motivation for this research, aimed at developing a practical geometry preprocessing tool to ensure the watertightness of surface mesh models for CFD simulations. In addition, the authors thank Gaurav Rasaily (rasaily@ualberta.ca), a research assistant from the University of Alberta, for his contribution in developing CFD simulation tools for downstream testing of the output models. The authors also wish to thank Dr. Nicolas Douillet (nicolas.douillet9@gmail.com) for providing the open-source MATLAB Toolbox for Mesh Processing and for his constructive suggestions in the model processing workflow.

Tianyu Zhou, https://orcid.org/0000-0001-6735-387X
Carlos F. Lange, https://orcid.org/0000-0001-9390-8728
Michael Versteege, https://orcid.org/0000-0001-7534-9694
Yongsheng Ma, https://orcid.org/0000-0002-6155-0167
Brian Fleck, https://orcid.org/0000-0002-2747-3719

REFERENCES

[1] Blazek, J.: Chapter 1 - Introduction. In: Blazek J, editor. Computational Fluid Dynamics: Principles and Applications (Third Edition), Oxford, Butterworth-Heinemann, 2015, p. 1–5. https://doi.org/10.1016/B978-0-08-099995-1.00001-4

[2] Garnier, E.; Adams, N.; Sagaut, P.: Large Eddy Simulation for Compressible Flows, Springer, 2021.

[3] Liu, X.; Yuan, H.; Qin, Q.; Chen, S.: INFLUENCE OF GEOMETRICAL DEFECTS ON AERODYNAMIC DRAG OF NON-WATERTIGHT AHMED BODY, Journal of Theoretical and Applied Mechanics, 62(4), 2024, 769–786. https://doi.org/10.15632/jtam-pl/195069

[4] Huang, J.; Zhou, Y.; Guibas, L.: ManifoldPlus: A Robust and Scalable Watertight Manifold Surface Generation Method for Triangle Soups, ArXiv Preprint ArXiv:200511621, 2020.

[5] Mattar, S.J.; Nezhad, M.R.K.; Versteege, M.; Lange, C.F.; Fleck, B.A.: Validation process for rooftop wind regime cfd model in complex urban environment using an experimental measurement campaign, Energies, 14(9), 2021. https://doi.org/10.3390/en14092497

[6] Tadie Fogaing, M.B.; Hemmati, A.; Lange, C.F.; Fleck, B.A.: Performance of turbulence models in simulating wind loads on photovoltaics modules, Energies, 12(17), 2019. https://doi.org/10.3390/en12173290

[7] Zhou, T.; Lange, C.; Versteege, M.; Ma, Y.; Fleck, B.: A Robust and Automated Method for Checking and Repairing Surface Mesh Models to Ensure Watertightness for CFD Simulations. CAD'25, U-turn Press LLC, 2025. https://doi.org/10.14733/cadconfP.2025.160-165

[8] Kavian Nezhad, M.R.; Lange, C.F.; Fleck, B.A.: Performance Evaluation of the RANS Models in Predicting the Pollutant Concentration Field within a Compact Urban Setting: Effects of the Source Location and Turbulent Schmidt Number, Atmosphere, 13(7), 2022. https://doi.org/10.3390/atmos13071013

[9] Kavian Nezhad, M.R.; RahnamayBahambary, K.; Lange, C.F.; Fleck, B.A.: Modified Accuracy of RANS Modeling of Urban Pollutant Flow within Generic Building Clusters Using a High-Quality Full-Scale Dispersion Dataset, Sustainability (Switzerland), 15(19), 2023. https://doi.org/10.3390/su151914317

[10] Kazhdan, M.; Bolitho, M.; Hoppe, H.: Poisson Surface Reconstruction. Proceedings of the 4th Eurographics Symposium on Geometry Processing, Cagliari, 2006, p. 61–70.

[11] Guo, X.; Xiao, J.; Wang, Y.: A Survey on Algorithms of Hole Filling in 3D Surface Reconstruction, The Visual Computer, 34, 2018, 93–103. https://doi.org/10.1007/s00371-016-1316-y

[12] Botsch, M.; Kobbelt, L.: A Remeshing Approach to Multiresolution Modeling. Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing, 2004, p. 185–192. https://doi.org/10.1145/1057432.1057457

[13] Zheng, S.; Zhai, Z.J.; Wang, Y.; Xue, Y.; Duanmu, L.; Liu, W.: Evaluation and comparison of various fast fluid dynamics modeling methods for predicting airflow around buildings, Building Simulation, 15(6), 2022, 1083–1095. https://doi.org/10.1007/s12273-021-0860-1

[14] Zhou, H.; Sun, P.; Ha, S.; Lundine, D.; Xiong, G.: Watertight modeling and segmentation of bifurcated Coronary arteries for blood flow simulation using CT imaging, Computerized Medical Imaging and Graphics, 53, 2016, 43–53. https://doi.org/10.1016/j.compmedimag.2016.06.003

[15] Ye, Y.; Wang, Y.; Cao, J.; Chen, Z.: Watertight surface reconstruction method for CAD models based on optimal transport, Computational Visual Media, 2024. https://doi.org/10.1007/s41095-023-0355-3

[16] Cai, Y.; Fan, L.: An efficient approach to automatic construction of 3d watertight geometry of buildings using point clouds, Remote Sensing, 13(10), 2021. https://doi.org/10.3390/rs13101947

[17] Cignoni, P.; Callieri, M.; Corsini, M.; Dellepiane, M.; Ganovelli, F.; Ranzuglia, G.: MeshLab: an Open-Source Mesh Processing Tool. Eurographics Italian chapter conference, 2008, p. 129–136.

[18] Attene, M.: A lightweight approach to repairing digitized polygon meshes, Visual Computer, 26(11), 2010, 1393–1406. https://doi.org/10.1007/s00371-010-0416-3

[19] Liepa, P.: Filling holes in meshes. In: Kobbelt L, Schröder P, Hoppe H, editors. SGP03: Eurographics Symposium on Geometry processing, 2003, p. 200–205.

[20] Attene, M.; Falcidieno, B.; Spagnuolo, M.: Hierarchical mesh segmentation based on fitting primitives, Visual Computer, 22(3), 2006, 181–193. https://doi.org/10.1007/s00371-006-0375-x

[21] Kazhdan, M.; Hoppe, H.: Distributed Poisson Surface Reconstruction, Computer Graphics Forum, 42(6), 2023. https://doi.org/10.1111/cgf.14925

[22] Wang, J.; Huang, J.; Wang, F.L.; Wei, M.; Xie, H.; Qin, J.: Data-driven Geometry-recovering Mesh Denoising, Computer-Aided Design, 114, 2019, 133–142. https://doi.org/10.1016/j.cad.2019.05.027

[23] Rakotosaona, M.J.; Guerrero, P.; Aigerman, N.; Mitra, N.; Ovsjanikov, M.: Learning Delaunay Surface Elements for Mesh Reconstruction. Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, IEEE Computer Society, 2021, p. 22–31. https://doi.org/10.1109/CVPR46437.2021.00009

[24] Charton, J.; Baek, S.; Kim, Y.: Mesh repairing using topology graphs, Journal of Computational Design and Engineering, 8(1), 2021, 251–267. https://doi.org/10.1093/jcde/qwaa076

[25] Meister, D.; Ogaki, S.; Benthin, C.; Doyle, M.J.; Guthe, M.; Bittner, J.: A Survey on Bounding Volume Hierarchies for Ray Tracing, Computer Graphics Forum, 40(2), 2021, 683–712. https://doi.org/10.1111/cgf.142662

[26] Stanford University Computer Graphics Laboratory: The Stanford Bunny. Available at: https://graphics.stanford.edu/data/3Dscanrep/ Accessed May 14, 2025.

[27] Nicolas Douillet: Mesh processing toolbox, GitHub, (https://github.com/NicolasDouillet/mesh_processing_toolbox/releases/tag/v3.3), Retrieved May 15, 2025.