Computer-Aided Design 164 (2023) 103608

Contents lists available at ScienceDirect

Computer-Aided Design

journal homepage: www.elsevier.com/locate/cad

Towards a High Quality Shrink Wrap Mesh Generation Algorithm Using Mathematical Morphology

Vijai Kumar Suriyababu *, Cornelis Vuik, Matthias Möller

Delft Institute of Applied Mathematics, Delft University of Technology, Delft, Netherlands

ARTICLE INFO

Article history:
Received 28 July 2022
Received in revised form 16 June 2023
Accepted 1 August 2023

Keywords:
Mesh simplification
Shrink wrapping
Boolean operations
Fluid volume extraction

ABSTRACT

Various computational fluid dynamic simulations in engineering, such as external aerodynamics, only need the silhouette of an input geometry. Often, it is a laborious process that can take up many human hours. In addition, the CAD geometries are too complex and contain intricate features and topological holes. We showcase an effortless way to shrink-wrap triangulated surfaces with the sole intent of topology and surface simplification. Building upon the concepts of mathematical morphology and newer advancements in geometry processing, we present a straightforward and robust algorithm that can guarantee genus-zero surfaces. Our techniques are equally applicable to general polyhedral meshes and well-suited for handling both oriented and unoriented point clouds. We provide examples using unoriented point clouds to demonstrate the versatility of our algorithms. We have designed our algorithms with a wide variety of applications in mind. However, we specifically highlight their capability for aerodynamic simulations, fluid volume extraction, and surface simplification. Additionally, we emphasize the practicality and ease of implementing the proposed algorithms, and we chain additional algorithms to develop variants of our wrap algorithm.

1. Introduction

1.1. Shrink-wrapping for surface remeshing

The shrink-wrapping algorithm can be a helpful tool for remeshing triangulated (or any polyhedral) surfaces. Most algorithms take a volumetric approach to solve the problem. They work by projecting a voxelized approximation of the input surface onto itself. Several papers in the literature focus on this problem, with most of the work being accomplished in the industry. Attene et al. [1] provide a very detailed comparison of different mesh repair algorithms. The "Global repair section" of their review article offers an excellent overview of state-of-the-art algorithms using a global mesh repair and simplification approach. A key highlight of this discussion is that all authors take a volumetric approach to the problem. However, none of them explicitly solve the problem of shrink-wrapping with CAE simulations in mind. These algorithms are generic mesh simplification approaches, primarily focused on computer graphics applications. Some notable contributions are from Esteve et al. [2] and Nooruddin et al. [3], who also employ volumetric approaches. Esteve et al. [2] shrink a discrete membrane to remesh surface meshes and point clouds, while Nooruddin et al. [3] take a strict morphological approach to the problem. One major drawback in their works is the lack of control over the surface genus. In the case of Nooruddin et al. [2], they oversimplify the meshes in all examples unlike our shrink-wrap mesh simplification. They also do not guarantee a manifold mesh as output, making their results unsuitable for numerical simulation. Y. K. Lee et al. [4] provide a good summary of existing techniques tailored to the shrink-wrapping problem for engineering applications. They highlight the effectiveness of these algorithms in closing gaps and removing interior parts of complex geometry, along with their potential as remeshing algorithms. They also show that gap detection and bridging are usually achieved with the help of poor/coarser voxelization, and there is often a need to intersect these voxels with the input surface mesh, which increases computational costs. Some algorithms [5] require explicit tolerance values for the gaps and holes, which could be advantageous in certain applications.

1.2. Notable contributions and limitations

More recent works on shrink wrapping are from the computer graphics community. *Point2Mesh*[6] from Rana Hanocka et al. solves the problem of shrink wrapping with a self-priori approach that employs deep learning. Their algorithm was tailor-made for point clouds. More recent work from Pierre Alliez et al. on three-dimensional alpha wrapping [7] uses alpha shapes for shrink

* Corresponding author.
E-mail addresses: v.k.suriyababu@tudelft.nl, vijai@vijaikumar.in (V.K. Suriyababu), c.vuik@tudelft.nl (C. Vuik), m.moller@tudelft.nl (M. Möller).

https://doi.org/10.1016/j.cad.2023.103608

Nomenclature

$\mathcal{M}$	Triangulated mesh (or surface)
σ	Structuring element
σ_r	Structuring element radius
T_l	Topological sphere level
ω	Solid angle for a query point with respect to an input mesh

wrapping surface meshes. A reference implementation of their algorithm has been included in CGAL, and it is extremely robust to various input geometries and can handle even line segments as input. Depending on the alpha value, the geometry's genus can change. It can be considered a more robust surface reconstruction algorithm that handles geometries with various defects. In a later section, we make a more direct comparison of our work against these works.

1.3. Recent advances in shrink wrapping

While the algorithms proposed in our present work can be fine-tuned to function similarly, we explicitly focus on fully automated genus simplification. In a later section, we make a more direct comparison of our work against these works. Hence, we propose an algorithm that can close all the gaps in triangulated surfaces and remove all internal structures. The significant contribution in our work is an efficient way of computing genus simplified offset surface with the help of morphological operators. Our algorithms will guarantee an outcome even in the case of imperfect geometries. Imperfections can range from missing triangles to non-manifold edges or completely separated components. We demonstrate the same in our numerical experiments. We also extend the existing shrink-wrap algorithm for selective genus control. An existing semi-heuristic algorithm that we developed [8] for hole detection is used to control the closing operations so that only selective holes are closed.

2. Morphological operators

Mathematical morphology is a vibrant subject that finds typical applications in the area of image processing [9]. However, it has been extended to many other application areas, including three-dimensional geometry processing [10]. Broadly, the subject of mathematical morphology is composed of four operators.

1. Erosion $(\mathcal{M} \ominus \sigma)$
2. Dilation $(\mathcal{M} \oplus \sigma)$
3. Opening $((\mathcal{M} \ominus \sigma) \oplus \sigma)$
4. Closing $((\mathcal{M} \oplus \sigma) \ominus \sigma)$

Mathematical morphology operators, such as erosion and dilation, are fundamental in various applications, including mesh processing. Erosion is an iterative process that erodes a given volume $\mathcal{M}$ (which can be in any dimension) using a structuring element σ with a radius of σ_r. On the other hand, dilation iteratively expands the input volume, adding to its size.

In our work, we draw inspiration from Zhen Chen et al. [11], who employed morphological operations for computing discrete surface offsets. However, our focus is on a more specific form of mesh and topology simplification. Instead of computing exact offset surfaces, we aim to derive simplified or approximated offset surfaces that primarily serve the purpose of topological simplification. To represent all the morphological operators, we utilize the Octree data structure, enabling the reuse of the mesh for numerical simulation.

Additionally, we introduce a topological sphere level parameter that corresponds to the scaled structuring element radius, denoted as σ_r, and scaled by the offset distance. This parameter holds substantial importance in our methodology, influencing the behavior and characteristics of the morphological operations employed in our approach.

We also mention the work of Silvia Sellán et al. [12], who propose a surface-only approach for selective modification of surface meshes. However, their method has limitations with surface flows and is not suitable for shrink wrapping applications. In our work, we utilize iterative surface opening to achieve a spherical topology, followed by iterative erosion for closure.

Detailed explanations of the different morphological operators in the context of mesh generation are provided in the subsequent sections.

2.1. Mathematical morphology of surface meshes

In our present investigation, the morphological operators utilized bear similarities to their counterparts in image processing. However, it is important to note some key distinctions due to the nature of surface meshes. In image processing, morphological operators are typically applied to simple two-dimensional grids, where a binary image is represented on a Cartesian mesh with binary mask values (0 and 1). This representation allows for a straightforward application of morphological operators, as the binary mask serves as an implicit representation of the object's boundaries.

In contrast, surface meshes lack an implicit mask as commonly used in image processing. In order to perform morphological operations on surface meshes, a volumetric representation of the mesh needs to be calculated from its boundary representation. This typically involves techniques such as voxelization or distance field computation, which allow for the determination of the interior and exterior regions of the mesh. By converting the surface mesh into a volumetric representation, we establish a comparable binary mask that aligns with the principles of morphological operators used in image processing. This enables the application of morphological operations on the surface mesh.

It is worth noting that one key difference with our erosion operator is that we do not erode the geometry beyond its original boundary, as our focus is on preserving the internal volume. This distinction is evident in our subsequent sections, where we provide a detailed explanation of our erosion operator and its specific behavior within the context of surface mesh processing. By adapting and applying morphological operators to surface meshes, we bridge the concepts of morphological operators in image processing with our approach in mesh processing, allowing for topology and surface simplification of surface meshes while preserving internal volume.

Let us consider the dilation and erosion operators on binary images and their geometric counterparts. Since other operators such as opening and closing can be built as a combination of these two, it is enough to understand these two operators.

2.2. Dilation

In the dilation process, we begin by dilating the given image using a square as the structural element. This choice is natural since images are represented as pixels, and using a square structural element simplifies the computation of adjacent neighbors at the edge level. Once the image's boundary is identified, performing one step of dilation becomes a matter of finding the neighboring faces of the border. A crucial aspect of this approach

is selecting the neighbors in the positive normal direction of the boundary. By repeating this process for the desired number of steps, we can achieve the required criterion.

In the context of image processing, the dilation operation can be implemented using the snippet in listing 1.

```
def dilation(image):
    dilated_image = copy(image)
    for each pixel p in image:
        for each neighbor n of pixel p:
            if n is a background pixel:
                mark pixel p as a foreground pixel
                break
    return dilated_image
```

Listing 1: Pseudo code for dilation in image processing

Similarly, in the context of octree-based mesh processing, the dilation operation can be implemented using the snippet in listing 2.

```
def dilation(mesh):
    dilated_mesh = copy(mesh)
    for each face f in mesh:
        for each neighboring face n of face f:
            if n is an empty face:
                mark face f as a filled face
                break
    return dilated_mesh
```

Listing 2: Pseudo code for dilation in octree-based mesh processing

These pseudo code snippets illustrate the process of dilation in both image processing and octree-based mesh processing.

2.2.1. Erosion

The erosion operator also starts from the boundary of a given image. However, it differs from the dilation operator in that it finds the border neighbors in the negative normal direction, typically within the image itself. A pivotal contrast to the dilation approach is that an image cannot be eroded infinitely. At some point, the erosion operation reaches a singularity where further erosion is not possible.

In the case of shrink wrapping, it is essential to prevent the erosion operation from destroying the internal volume of a surface mesh. Therefore, the erosion operation usually stops at the boundary of a surface mesh.

In the context of image processing, the erosion operation can be implemented using the following pseudo code:

```
def erosion(image):
    eroded_image = copy(image)
    for each pixel p in image:
        for each neighbor n of pixel p:
            if n is a foreground pixel:
                mark pixel p as a background pixel
                break
    return eroded_image
```

Listing 3: Pseudo code for erosion in image processing

Similarly, in the context of octree-based mesh processing, the erosion operation can be implemented using the following pseudo code:

```
def erosion(mesh):
    eroded_mesh = copy(mesh)
    for each face f in mesh:
        if f is not on the boundary of the mesh:
            for each neighboring face n of face f:
                if n is a filled face:
                    mark face f as an empty face
                    break
    return eroded_mesh
```

Listing 4: Pseudo code for erosion in octree-based mesh processing

These pseudo code examples illustrate the process of erosion in both image processing and octree-based mesh processing.

2.2.2. Closing and its extension to surfaces

The closing operator combines the dilation and erosion operations. In computer vision and image processing, this operator is used to close holes (or missing pixels) in a binary image. In our algorithms, we apply the same operation to three-dimensional data to close holes in the meshes. For easy comparison in mesh processing, we extract a three-dimensional surface of the same geometry (by extruding the contour in the Z direction) and perform the closing operation on it. The results can be seen in Fig. 4.

There are two differences compared to the image processing approach:

- We do not erode beyond the boundary of the input geometry to preserve its shape.
- We project the eroded geometry onto the input geometry to maintain its original topology.

Fig. 4 demonstrates the effect of the closing operation on a three-dimensional surface. This geometry is equivalent to the two-dimensional binary images shown in Fig. 3. In addition to closing the holes caused by missing triangles, the closing operation also seals the topological holes in the mesh. Multiple levels of closing are showcased in this case, demonstrating the ability to achieve the desired topology by adjusting the number of wrap operations. The explicit and implicit control of genus is further discussed in later sections of the paper. This benefit is leveraged for shrink wrapping surface meshes in external aerodynamic simulations.

However, it should be noted that the straightforward closing operation on its own is not suitable as it lacks a stopping criterion. To address this limitation, our workflow incorporates a series of algorithms and data structures, such as octrees, to efficiently utilize these morphological operators for shrink wrapping surface meshes. The details of these techniques are explained in the subsequent section.

3. Shrink wrap algorithm

This work focuses on three versions of the wrapping algorithm to cater to specific practical applications. These variants are as follows.

1. Simple wrap
2. Smooth wrap
3. Developable wrap

The simple wrap algorithm acts as a base for all variants. Additional algorithms are chained to the simple wrap algorithm results to yield a smooth and developable version of the algorithms.

3.1. Simple wrap

As stated earlier, the simple wrap algorithm acts as a common denominator for all the variants of the algorithm proposed in the present investigation. It has the following steps

1. Conversion of Boundary representation(B-Rep) to a volumetric representation (V-Rep)
2. Computation of signed distance function
3. Dilation of the input surface for a given topological sphere level

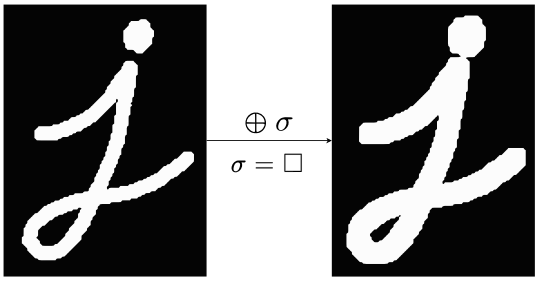


Fig. 1. Dilation operation on a binary image. The input image on the left is dilated to obtain the image on the right.

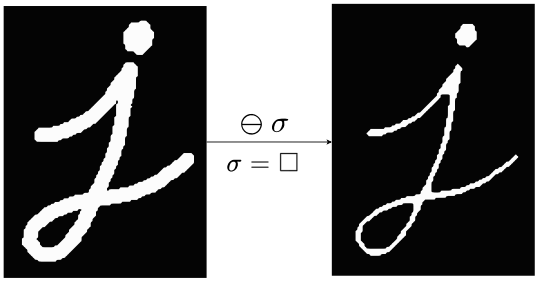


Fig. 2. Erosion operation on a binary image.

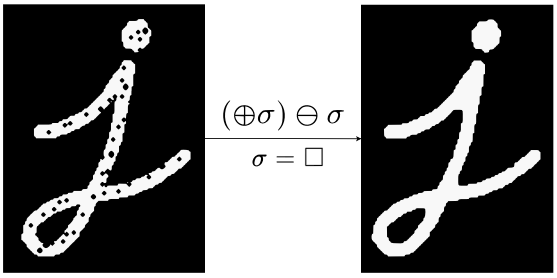


Fig. 3. Closing operation on a binary image (black blobs on the left indicate missing pixels), equivalent to holes or missing triangles in a surface mesh.

4. Erode the dilated surface to obtain a topologically hole-free offset surface
5. Iteratively project and smooth the dilated surface
6. Remesh the surface using existing remeshing algorithms (Optional).

We explain these individual components in detail along with the respective algorithms in the subsequent sections.

3.1.1. B-Rep to V-Rep

It was shown earlier that geometry information needs to be encoded as binary images for efficient computation of morphological operators in computer vision applications. This translates to a boolean value in every voxel of a Cartesian mesh[1] in three dimensions. However, it would require intersecting the surface mesh with the Cartesian mesh. This would only work for watertight surface meshes and lead to some inaccuracies in the case of degenerate geometries. Hence, we do not physically intersect the input geometry with the Cartesian mesh as we only need a scalar field to distinguish the inside and outside of the geometry. The usual workflow for Cartesian mesh generation starts with a bounding box computation as shown in Fig. 6. These can be an axis-aligned or oriented-bounding box. In either case, the generated Cartesian mesh would not be very beneficial for morphological operations. Morphological operators such as erosion and dilation are applied in successive layers. For example, the dilation operator starts from the boundary of a surface and dilates the surface one layer after another, as shown in Fig. 1. Since we use a cube or a voxel as a structuring element, it is easier to perform these operations successively if the geometry sits approximately in the center of the Cartesian mesh. If the geometry is moved to the origin of the octree, a dilation operation may not completely dilate the entire surface in a given step. First, the mini ball algorithm [13] is used to obtain a tightly fitting sphere of an input geometry as shown in Fig. 7. Then we compute a bounding box for this sphere with a specified offset threshold to ensure that our geometry always sits precisely at the center of our voxelization. Next, we refine all the cells inside the tightly fitting sphere as shown in Fig. 8. Post refinement, the generalized winding number approach [14] helps distinguish the cells inside and outside the geometry. The generalized winding number algorithm gives a solid angle value at every vertex in the octree mesh. This value is thresholded to mark the cells in the octree as inside, outside, or boundary cells. This refinement allows us to get a more accurate surface description during the segmentation process. Spherical refinement also limits the inside–outside queries to the cells within the sphere. As a result, we do not need to query the generalized winding number for cells outside the sphere, thereby saving computational time. Finally, we also ensure a 2:1 refinement in our octree for all elements in our workflow. The mesh generation approach referenced in algorithm 1 can be used for any kind of numerical simulation irrespective of the rest of the workflow.

3.1.2. Computation of signed distance function

It is evident that once the cells of the octree are classified into inside and outside (using an approach such as generalized winding numbers in our case), an artificial signed distance function can be bestowed upon the voxelization as shown in Fig. 10. We rely on Generalized winding numbers since they are swift even on a CPU-only computational environment and are immune to imperfections in the input surface to a large degree. A brief overview of this approach can be seen in Appendix A. However, for the rest of the algorithmic workflow, one only needs to categorize the cells in the octree as inside or outside cells. These will be used to build an approximate surface boundary which can be used for morphological operations described in the consequent sections. Our experimental observation has shown that a solid angle value of 0.9 sr indicates cells inside a surface mesh, and everything else can be marked as outside. The bounding sphere computed in the mesh generation algorithm can be used to automatically mark all the cells outside the sphere as outside cells.

We use the term artificial since we do not compute the exact distance here. We only use an integer that indicates a particular voxel's relative position with its respective boundary voxel. As will be evident later, we do not need an exact signed distance field for computing a Genus simplified offset surface. A similar approach has been used by other researchers [15] to calculate intersection-free offset surfaces. We achieve the same by outward propagation from the zero-level set voxels. This outward propagation is done along the normal outward direction of the surface mesh. Since we mark all the cells in the octree as inside or outside, zero level set voxels or boundary voxels can be determined by finding cells that contain faces that are part of both inside and outside cells. As opposed to the usual approaches, which intersect the surface mesh with the octree mesh, our proposed method is highly computationally efficient. All the morphological operations are explained with the help of a maple leaf geometry shown in Fig. 9.

[1] We use the terms Cartesian mesh and octree interchangeably throughout the paper. The reader should be aware that all the computations are performed on an octree, which we consider a special kind of Cartesian mesh

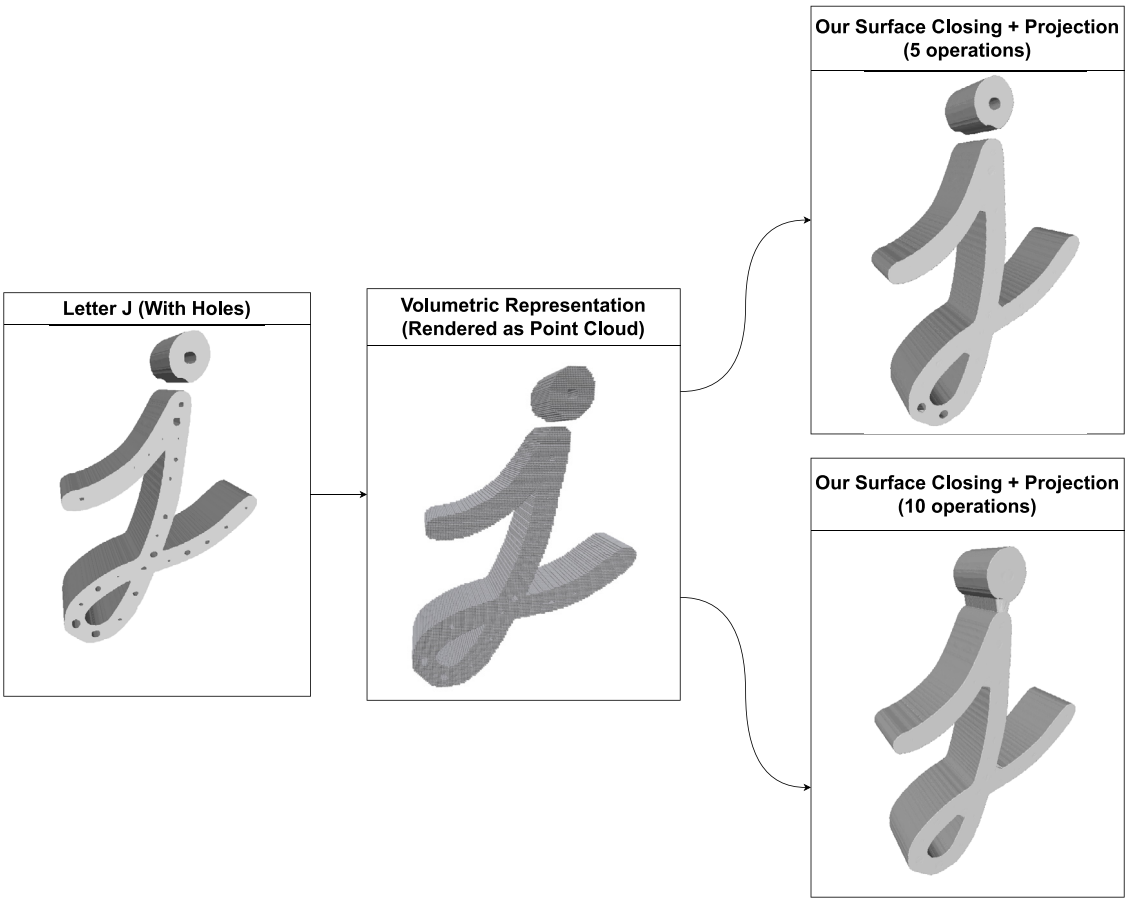


Fig. 4. Closing operation on a three-dimensional surface. This geometry is equivalent to the two-dimensional binary images in Fig. 3. Additional holes have been intentionally introduced in the geometry. The closing operation is performed on the leftmost geometry and then projected onto the ground truth.

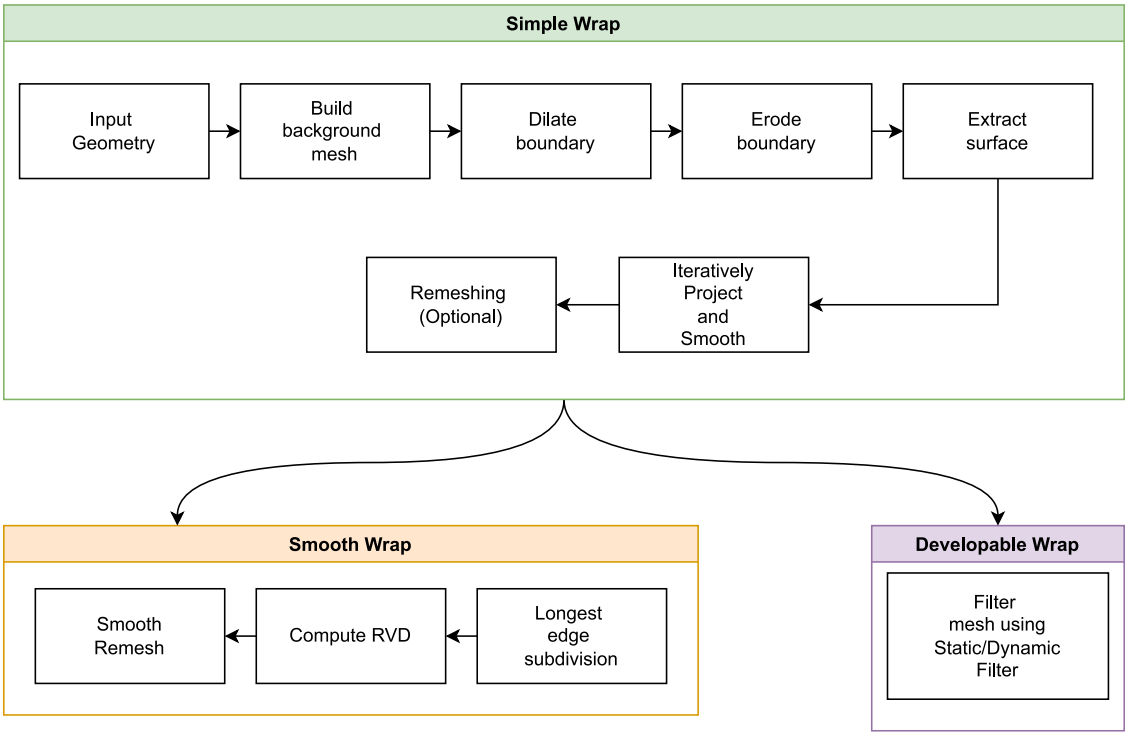


Fig. 5. Algorithm workflow.

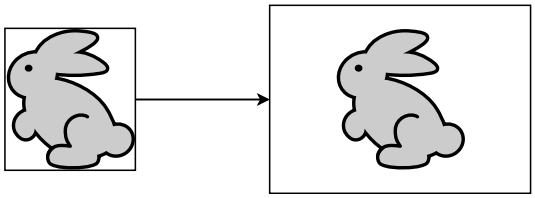


Fig. 6. Normal bounding box computation (In case of many geometries, a manual offset threshold may be required for ensuring there are enough layers of mesh for morphological operations. The threshold might also be different for different directions.).

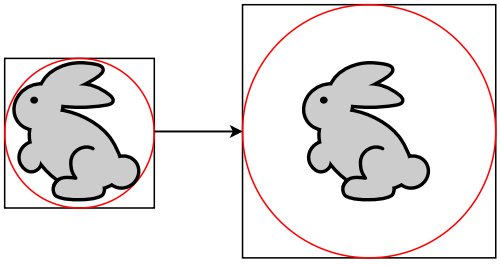


Fig. 7. Spherical bounding box computation (In most cases, a threshold of 2 or 3 times the size of the bounding sphere is enough for all morphological operations). The scaling will be uniform irrespective of the geometry since only the sphere is scaled and the bounding box is always a perfect cuboid.

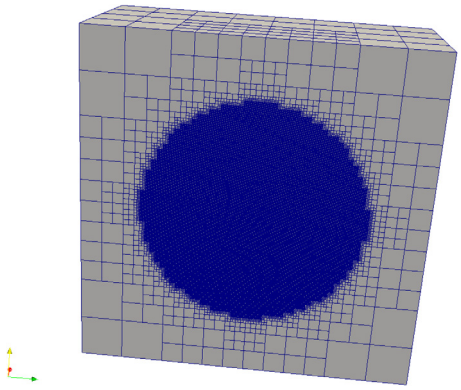


Fig. 8. Spherical refinement.

3.1.3. Dilation driven approximated offsets

In the previous section, we proposed a straightforward method to determine the zero level set or boundary voxels of a given surface mesh inside an octree. We already established that a dilation operation followed by an erosion operation performed in a sequence leads to the morphological closing operation as shown in Fig. 3. Our investigation also reveals that one does not need to dilate the boundary voxels across the entire voxelization. Instead, we only need to dilate the input surface until we achieve a spherical topology. Here, we use a user-specified parameter called **"Topological sphere level"**. This parameter is the only user-controlled input in the algorithm, and the choice of topological sphere level dictates the number of outward propagation levels as shown in algorithm 2. The bigger the hole in the geometry, the larger the topological sphere level. Effects of different Topological sphere levels are clearly shown in the numerical experiments. For example, the dilated maple leaf geometry can be seen in Fig. 11. It is clearly evident that a spherical topology is achieved after approximately 15 levels.

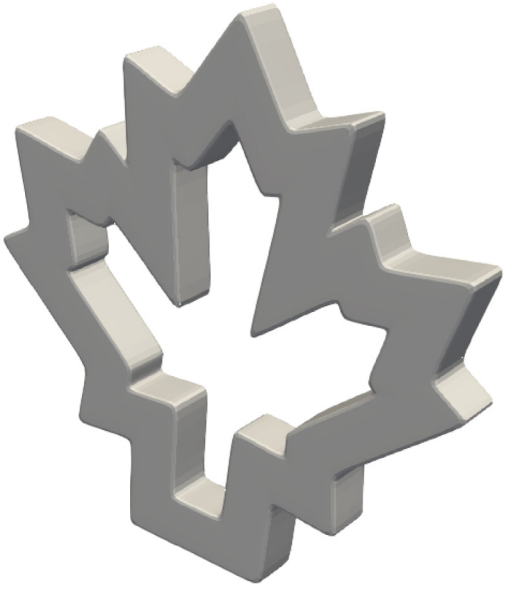


Fig. 9. Maple leaf geometry.

Algorithm 1: B-Rep to V-Rep

Result: Voxelized mesh where every cell has a scalar associated with it (inside / outside)
Initialise Surface;
Compute a tight bounding sphere using the mini ball algorithm and store its radius and center ;
Calculate the bounding box of the sphere, which is offset at a user-specified distance *(2.0 in our experiments)*;
Initialize a Cartesian mesh with a specified cell size or number of cells *(64 * 64 * 64 in all of our experiments)*;
forall *Cells of Cartesian mesh* **do**
  if *Cell inside bounding sphere* **then**
    Mark for refinement;
  end
  else
    Mark for coarsening;
  end
end
forall *Cells in bounding sphere* **do**
  Compute the Generalized winding number (This indirectly gives us the solid angle for all the cells in the octree);
end
Mark cells outside bounding sphere as outside and store this in the respective cells;
forall *Cells in bounding sphere* **do**
  if *Solid angle is higher than 0.9 steradians (based on our experimental observation)* **then**
    Mark the cell as inside and store this in the respective cell;
  end
  else
    Mark the cell as outside and store this in the respective cell;
  end
end

If complete automation is required from input to projection, the topological sphere level can be ignored, and the geometry can be dilated to the maximum level.

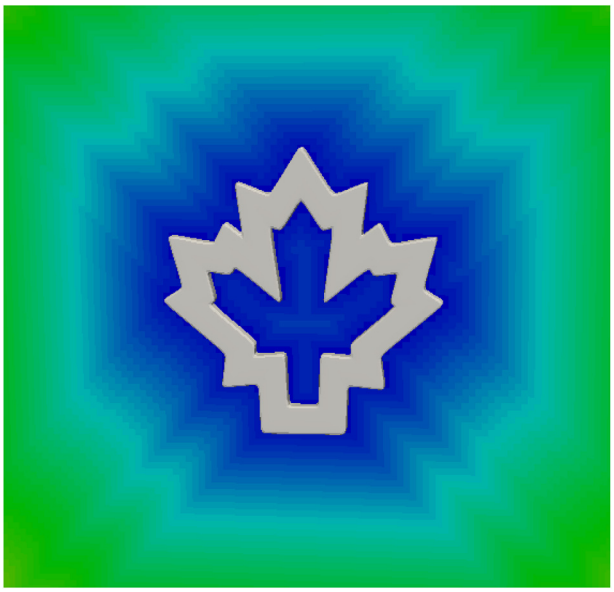


Fig. 10. Artificial signed distance function of a maple leaf (Computed using our approach).

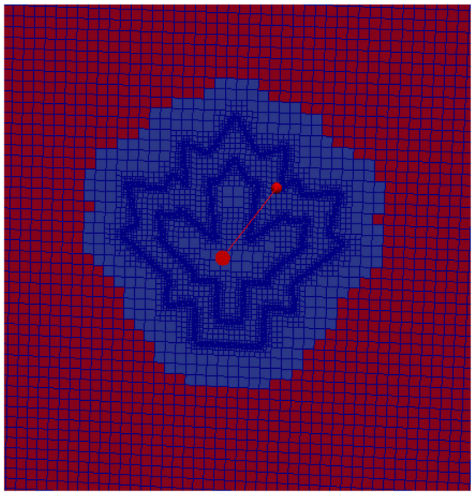


Fig. 11. Dilated maple leaf geometry with a spherical topology.

3.1.4. Erosion of dilated offset surface

The dilated surface should now have a spherical topology, and it needs to be eroded towards the input surface. This process is similar to the dilation except for the marching direction. The number of levels would be the same as the topological sphere level chosen during the previous step of the algorithm. Once eroded, this will give a hole-free approximation of the input geometry. Once the surface is eroded to the given "Topological Sphere Level", the operation becomes straightforward. It is explained in detail in algorithm 3. The scalar field can be directly eroded until it hits the boundary voxels. This is where the erosion operation differs from the erosion operation in computer vision algorithms. In the case of surface mesh, the geometry is never eroded beyond the boundary voxels for volume preservation. This approach is relatively simple since a manifold mesh can be easily extracted without the need for any additional algorithms [16].

The offset surface is still embedded inside a volumetric mesh as shown in Fig. 12, and a surface needs to be extracted. Due to the artificial nature of the signed levels in the volumetric mesh, it is easy to distinguish the region where the genus simplified offset meets the external offset surface. Surface extraction becomes a simple task with this information. This is similar to the approach proposed for identifying boundary voxels from inside and outside voxels. The detailed algorithm for surface extraction is listed in algorithm 4.

Algorithm 2: Dilation of the input surface

```
Result: Dilated surface stored in the voxelized mesh
Initialize interior_cells as seed_cells;
Initialize current_topological_sphere_level to 0;
while current_level ≤ topological_sphere_level do
    Initialize a newer_seeds_cells_id vector;
    forall cells in seed_cells do
        forall cell_neighbours in voxelized_mesh do
            if Neighbour is outside cell then
                Add neighbour to newer_seeds_cells_id;
            end
        end
        Set newer_seeds_cells_id as seed_cells;
        Increment current_topological_sphere_level; if
         current_topological_sphere_level eq
         topological_sphere_level then
            Store these cells as topological_sphere_cells;
        end
    end
end
```

Algorithm 3: Erosion of the dilated offset surface

```
Result: Eroded surface stored in the voxelized mesh
Initialize topological_sphere_cells as seed_cell_ids;
forall topological_sphere_levels do
    Initialize a newer_seeds_cells_id vector;
    forall cells in seed_cells do
        forall cell_neighbours in voxelized_mesh do
            if Neighbour is from lower topological_sphere_level
             then
                Add neighbour to newer_seeds_cells_id;
            end
        end
        Set newer_seeds_cells_id as seed_cells;
    end
end
Final seed_cells form the basis for the Genus simplified offset
 surface;
```

3.1.5. Projection and smoothing

The eroded surface needs to be projected onto the input geometry. With practicality in mind, we chose a point cloud- based approach over direct projection on the triangulated surface. In realistic industrial geometries, the construction of an AABB tree is costly and time-consuming and leads to failure in many cases. Since we chose to allow input meshes that are not perfectly two-manifold, the point cloud-based approach will support a broader range of input meshes, including those that are entirely degenerate, as shown in the later section. We approximate the input geometry as a uniformly sampled point cloud and then construct a kD tree [17] on it. We also extended our algorithm for point cloud due to this projection approach. However, one can choose a more sophisticated method that projects directly onto the triangles in the input surface. This might produce erroneous

Algorithm 4: Surface Extraction

```
Result: Genus simplified watertight surface
Initialize topological_sphere_cells as seed_cell_ids;
forall topological_sphere_levels do
    Initialize a newer_seeds_cells_id vector;
    forall cells in seed_cells do
        forall cell_neighbours in voxelized_mesh do
            if Neighbour is from lower topological_sphere_level
             then
                Add neighbour to newer_seeds_cells_id;
            end
        end
        Set newer_seeds_cells_id as seed_cells;
    end
end
Final seed_cells form the basis for the Genus simplified
 surface;
```

results if the surface mesh is completely degenerate. We experimented with both a direct projection approach and the one using point cloud sampling as shown in Fig. 13 and the results were satisfactory for our point cloud approach.

Algorithm 5: Projection and Smoothing

```
Result: Projected and smoothed mesh
Sample a uniform point cloud on the input surface;
Build a kD tree on the uniformly sampled point cloud;
forall vertices in the Extracted surface do
    Find the nearest vertex in the kD tree and move the
     vertex;
end
forall vertices in the projected mesh do
    Find one ring neighborhood ;
    Average the position of the current vertex with the
     vertices from the one ring neighborhood ;
end
```

We can find the nearest neighbor in this point cloud for every vertex in the eroded surface and move the vertex to this position. Unfortunately, results do not look good at this stage, and the mesh seems slightly tangled. However, our experiments show that a few cycles of Laplacian smoothing followed by projection will immediately provide better quality results, as seen in Fig. 5.

Our investigation also shows that the geometry can be double wrapped to achieve better quality results. In double wrapping, the final result from the first run of the algorithm can be passed back onto the same workflow to produce a better quality approximation. The mesh at this stage is already analysis suitable. If required, one can include an optional remeshing step for improving the mesh quality further. The geometry practitioner is not required to follow our heuristic-based approach. They can choose any remeshing algorithm (commercial or public domain). However, the proposed algorithms provide satisfactory results in our investigation.

3.1.6. Remeshing and quality improvement (optional)

This step is entirely optional. The projected surfaces are well suited for analysis, and we show the same in numerical experiments for various surface and volumetric PDEs. However, the smoothed surface may still have a few tangled edges and triangles with lousy quality. Therefore, we remeshed the geometries using existing algorithms available in open source libraries such as Geogram and CGAL [18]. This approach seems to improve the mesh quality in all our numerical experiments vastly. Since we produce a genus zero surface in most cases, spherical parameterization-based remeshing approaches can also be considered an alternative. However, we found meshes at the projection stage suitable for numerical simulations. Therefore, it is not within the scope of our work to investigate a dedicated remeshing approach. In fact, we use the geometries from the

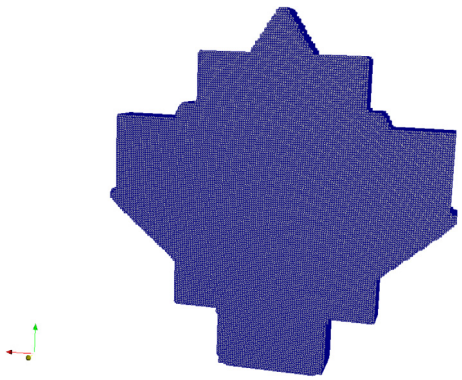


Fig. 12. Eroded offset surface (Unsmoothed & Hole Free).

Fig. 13. Projected & smoothed mesh.

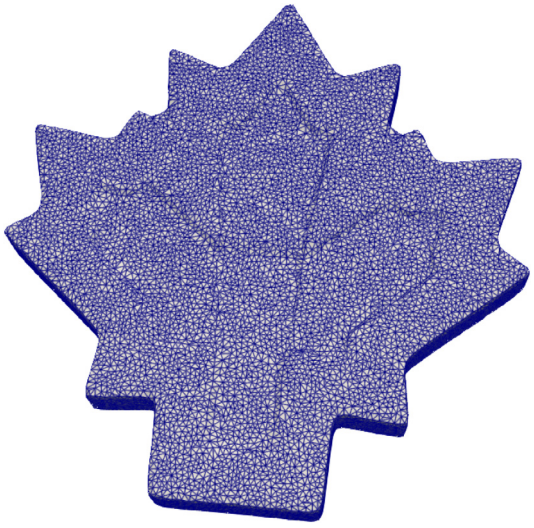


Fig. 14. Remeshed and quality improved maple leaf geometry.

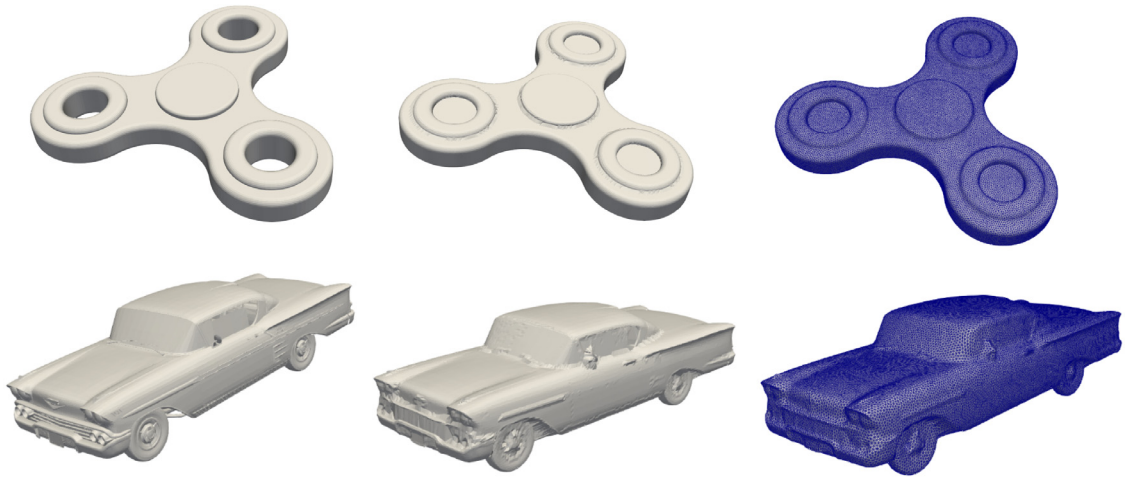


Fig. 15. Input geometries along with their simple and smooth wrapped geometries. It can be noticed that smooth wrap produces a smooth and high quality triangulation due to the RVD based surface reconstruction algorithm.

projection stage for the numerical experiments shown in the subsequent section and ignore the remeshing routine altogether (see Fig. 14).

3.2. Smooth wrap algorithm

Our shrink wrap algorithm at its core is a topological simplification algorithm. Hence, it inherently simplifies the geometry to a certain extent. However, specific applications may desire a smoother version of the output and might not need to preserve the geometry features accurately. Hence, we introduce a version of our algorithm called the smooth wrap. In some cases, the simple wrap algorithm can introduce sharp edges, which may not be desirable for specific applications (see Fig. 15).

Algorithm 6: Smooth wrap algorithm

Result: Smoothly wrapped mesh
Determine median edge length of the wrapped surface mesh;
Subdivide the simply wrapped mesh until all the edges are divided to the median edge length;
Uniformly sample points on the surface;
Smoothly reconstruct a surface using RVD based approach;

Therefore, we use the longest edge subdivision algorithm to split the edges until it reaches an average edge length. Subdivision allows uniform point sampling on the surface. Dobrina Boltcheva and Bruno Lévy proposed a smooth surface reconstruction based on the restricted Voronoi diagram [19]. This approach does not preserve any of the sharp features. However, it produces a smooth representation of the wrapped surface that can be readily used for practical applications.

3.3. Developable wrap algorithm

Developable surfaces are highly valuable in architectural applications, as well as in manufacturing for producing sheet metal models during the early design stage. The key characteristic of a developable surface is that it should be easy to manufacture. To achieve this, there exist numerous sophisticated approaches that can be employed.

In the context of triangular meshes, Oded Stein et al. have conducted extensive research on the developability of such surfaces. Their work provides a comprehensive exploration of the topic [20]. We encourage readers to refer to their excellent research for further insights.

In our specific case, the wrapped geometry exhibits irregular curvature and features. To enhance the manufacturability of such geometries, Zhang et al. proposed a static/dynamic filtering algorithm [21] for filtering and denoising meshes. This approach iteratively removes weaker features from the wrapped geometry, resulting in a simplified geometry that retains only the most prominent features. By doing so, the geometry becomes easier to manufacture (see Fig. 16).

More sophisticated approaches can be employed to transform our wrapped meshes into specific developable surfaces. However, for our investigation, we found the denoising approach to be suitable due to its ability to consistently remove unnecessary features in a wrapped geometry. We apply this algorithm as a post-processing step on meshes wrapped using the simple wrap algorithm. It is worth noting that this denoising approach can also be utilized with the smooth wrap algorithm, providing flexibility in its application.

4. Numerical experiments

We perform experiments on a wide variety of input geometries that help underscore the robustness of our algorithm. We noticed that even in the case of entirely ill-formed artifacts from industry, we could guarantee some form of a Genus simplified geometry. Our algorithm produced a valid two-manifold surface mesh without any holes in all cases; a wide variety of surfaces and their shrink-wrapped counterparts are shown in Appendix B.

4.1. Effect of topological sphere level

As stated earlier, **"Topological sphere level"** is the only parameter in the algorithm. In simpler terms, this is the number of layers the algorithm needs to travel along the positive normal direction of an input surface. It can be increased or decreased depending on the size of the biggest hole in the geometry. Since the algorithm is fast, one can choose the topological sphere level value on trial and error. The algorithm could be allowed to propagate to the maximum possible level. However, this might significantly increase the algorithm's run time, which is entirely unnecessary in our case. We show some examples of the same in Section 4.3.1 and some of its pleasant side effects.

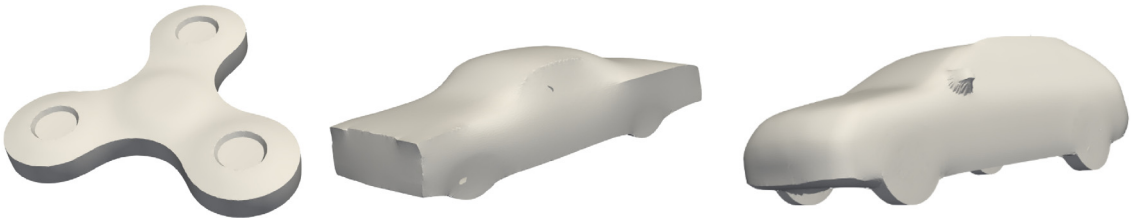


Fig. 16. Various geometries filtered using static/dynamic filtering. It can be seen that the geometries are developable.

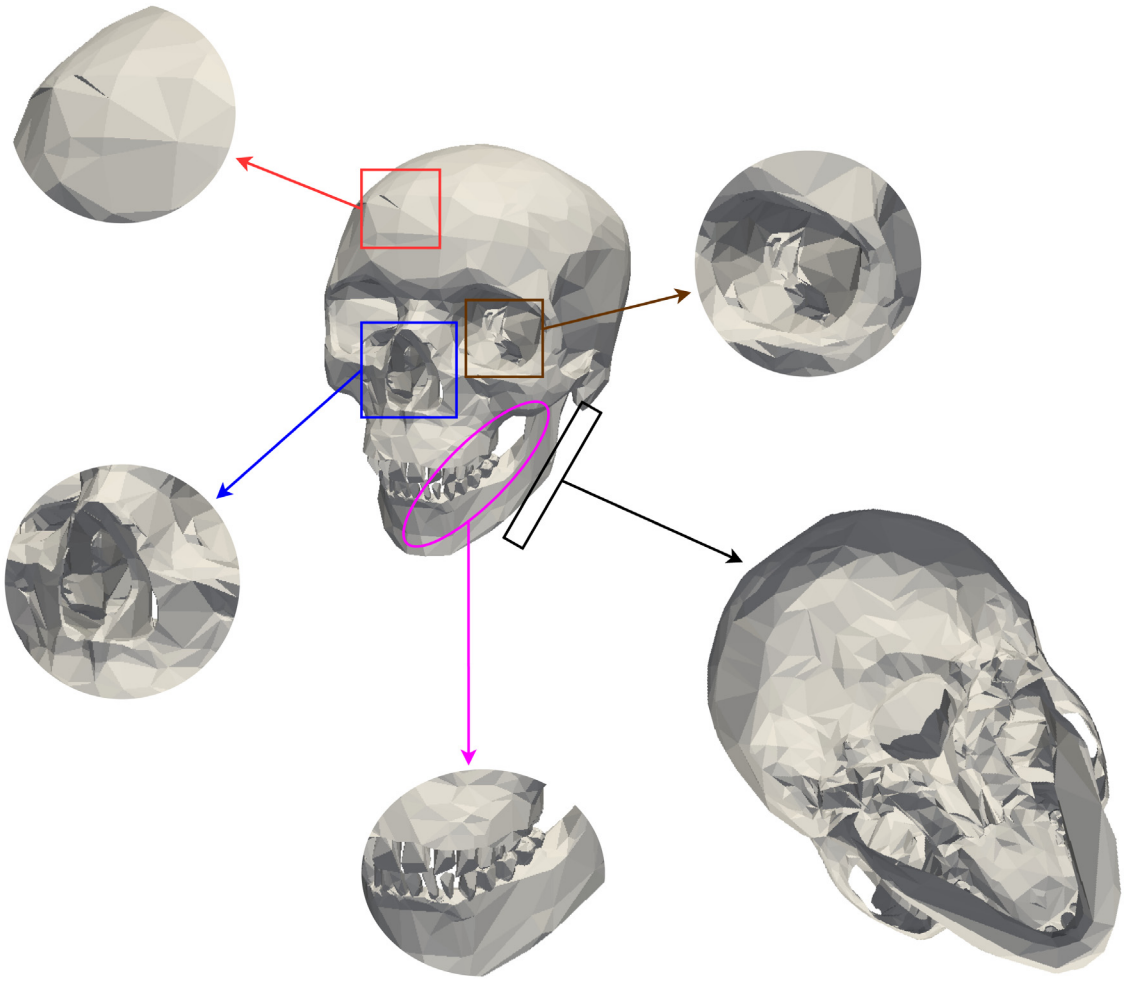


Fig. 17. A human skull geometry with at least 15 non-manifold edges and many disconnected components. Various defects in a skull geometry (gaps, disconnected teeth, non-manifold edges, and holes) are highlighted. It also includes the bottom view of the skull, which is entirely deformed.

4.2. Effect on bad quality geometries

We show a car geometry in Fig. 19 which is missing most of its bottom. We use a coarser grid to produce a simplified approximation of the car geometry. It can be seen that our algorithm produces a tight wrap even in this case and simplifies the geometry. Since the offset computation does not require an exact segmentation of the geometry boundary, the hole-free offset computation works even in such extremely poor quality geometries. The topological sphere level can be tuned on a trial and error basis for such geometries until the complete geometry is wrapped. In the case of skull geometry shown in Fig. 17, it has many non-manifold edges and many disconnected components. There is no pre-processing requirement on either of the geometries, and their shrink-wrapped results are shown in Fig. 18 a perfect two-manifold mesh without any leaks.

4.3. Selective genus closing

Almost all of the shrink wrapping algorithms are either used as remeshing (i.e., preserve the genus of the input mesh) or surface simplification algorithms (i.e., turn the input mesh into a topological sphere or Genus zero). However, there are scenarios where an industrial practitioner is only interested in closing selective holes. We could not find any other works that directly address this problem. We propose two ways to do this in our paper.

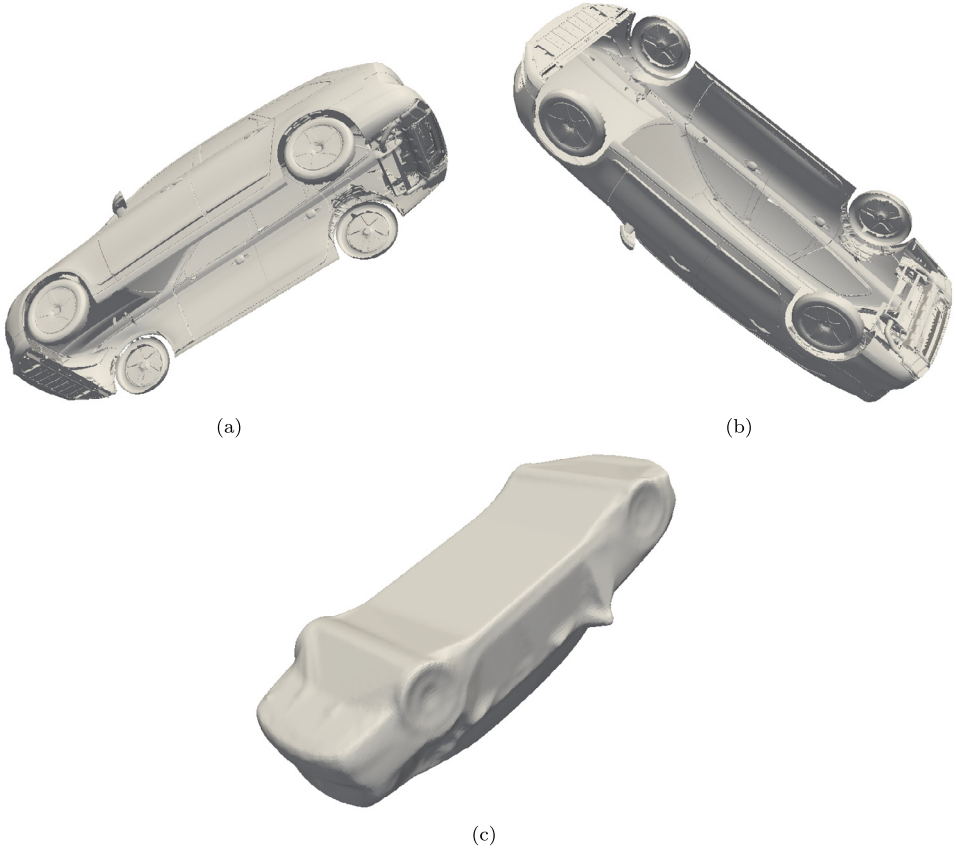
- Implicit Genus control
- Explicit Genus control

4.3.1. Implicit genus control

In this case, no additional algorithms are required. However, it can be an iterative process to achieve the necessary genus.

(a) Wrapped (Front View) (b) Wrapped (Bottom View)

Fig. 18. Wrapped skull geometry (Watertight geometry with a genus zero).



(a) (b)

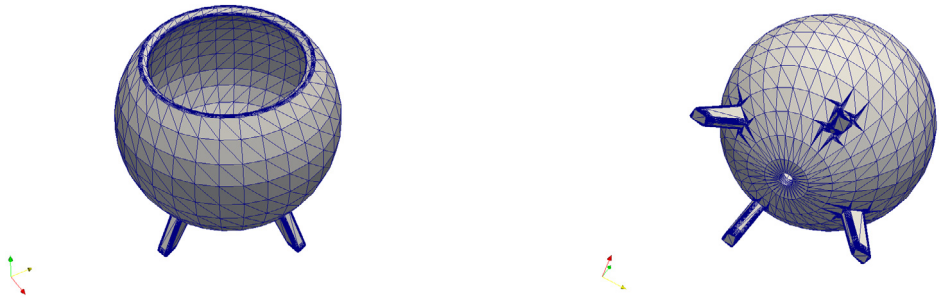
(c)

Fig. 19. Bad quality geometries shrink wrapped (Car with hole) - Input mesh along with wrapped output mesh. It can be observed that the bottom of the car is completely closed.

We leverage the topological sphere level's ability to control the genus implicitly. A lower value for the topological sphere level usually leads to incomplete closing of the geometry. This can be a desirable side effect in the case of selective Genus control. We have an example geometry in Fig. 20 below with a huge topological hole at the top and a smaller one at the bottom. The effects of different topological sphere levels are shown in Fig. 21. It can be observed that a value of 50 yields a Genus zero surface; however, at a level 10, only the smallest hole in the mesh is closed.

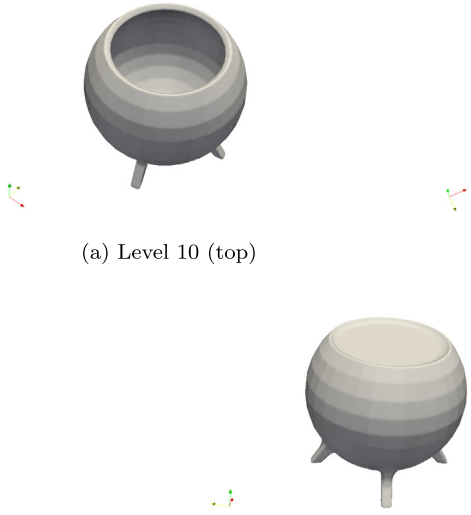
4.3.2. Explicit genus control

Explicit Genus control requires prior information about the topological holes in the mesh. Therefore, we use our topological hole detection algorithm [8] to accomplish the same. In this case, topological hole information is extracted from the hole detection algorithm, which is used as a boundary condition in the dilation stage. The hole detection algorithm would provide the holes' center and radius, and the desired hole radius can be given as a criterion. The radius criterion is only suitable for circular holes. If the geometry also has non-circular holes, hole surface patches from the algorithm can be used to compute the surface area of individual holes. This can be used as a further filtering criterion. Then the faces which are part of the desired holes are not diffused into the volume, thus preserving the structure. The algorithm 2 would have to be modified for Explicit Genus control, and it can be seen in algorithm 7



(a) Bigger hole (b) Smaller hole

Fig. 20. An example geometry with a big hole on top and a small hole at the bottom.



(a) Level 10 (top) (b) Level 10 (bottom)

(c) Level 50

Fig. 21. Shrink wrapped geometry for different topological sphere levels.

As explained earlier, the dilation process happens layer by layer. Therefore, one would have to ignore the blocklisted boundary cells for the dilation process to achieve selective genus control.

4.4. Boosting projection quality using external sources

We mentioned earlier that we double wrap the geometries to achieve a better projection quality. This is primarily due to the geometric structure of morphological erosion. It leads to a competitive projection which can be beneficial in many cases. For example, if the bottom is entirely missing, rather than failing to close the hole in the bottom, vertices are projected to the next closest area, which would be the boundary of the bottom hole. This ensures a good priori for the next wrapping stage. Hence, the double wrapping stage would provide a better distribution of triangles. This is an undesirable yet pleasant side effect of the competitive nature of the projection of the algorithm to stick to whatever comes first. However, if the geometry practitioner/end-user in the industry would like to have better conditioning for such holes/even topological holes, using external sources would help. In our earlier work [8], we proposed a semi-

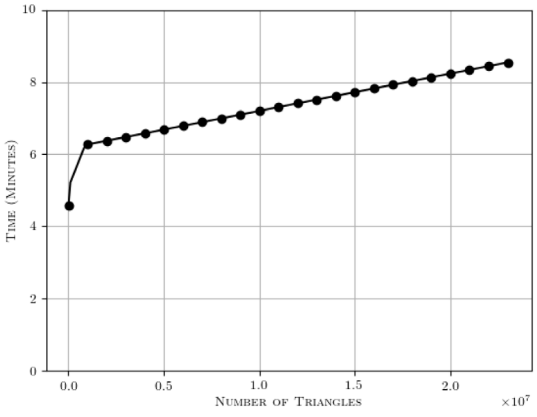
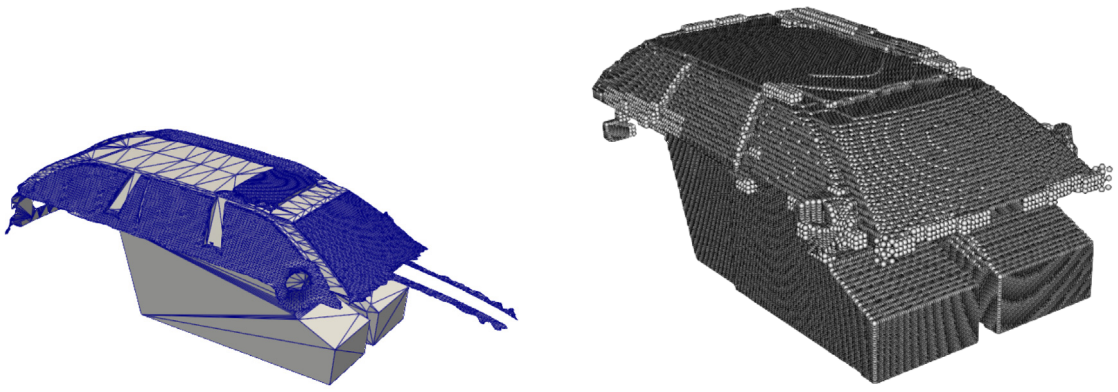


Fig. 22. Run time comparison for increasing number of triangles.

heuristic algorithm to detect topological and geometric cavities in a triangulated mesh. Just like our present algorithm, it does not need a perfect two-manifold mesh.



(a) Wrapped Car (b) Car fluid volume

Fig. 23. A Partial car geometry that has been shrink wrapped and its fluid volume extracted as a volumetric point cloud. Volumetric point clouds are otherwise considered particle distributions for meshless methods like Smoothed particle hydrodynamics.

Algorithm 7: Dilation of the input surface (with Genus control)

Result: Dilated surface stored in the voxelized mesh
Detect holes using hole detection algorithm;
Mark the cells that intersect with the holes and mark them as **blocklisted cells**;
Initialize **interior_cells** as **seed_cells**;
Ensure that the **blacklisted cells** are removed from the initial **seed_cells**;
Initialize current_topological_sphere_level to 0;
while ***current_level*** $\leq$ ***topological_sphere_level*** **do**
  Initialize a newer_seeds_cells_id vector;
  forall *cells in seed_cells* **do**
    forall *cell_neighbours in voxelized_mesh* **do**
      if *Neighbour is outside cell* **then**
        Add neighbour to newer_seeds_cells_id;
      end
    end
    Set newer_seeds_cells_id as seed_cells;
    Increment current_topological_sphere_level; **if** *current_topological_sphere_level eq topological_sphere_level* **then**
      Store these cells as topological_sphere_cells;
    end
  end
end

4.5. Runtime analysis

The algorithm presented in this paper is memory-bound, and its memory usage is directly influenced by the size of the octree and the number of triangles in the input mesh. While we have emphasized the importance of the Topological Sphere Level as the primary parameter in the algorithm, other geometric parameters related to the octree can also be adjusted according to specific requirements.

Throughout our investigations, we conducted experiments using a fixed initial grid size of $64 \times 64 \times 64$ and employed three levels of spherical refinement. These settings yielded satisfactory results for our purposes. The implementation of the algorithm was done in C++ with OpenMP parallelization to enhance computational efficiency. However, please note that the source code is currently not available, as it is part of a larger in-house codebase. We do have plans to release a standalone implementation as an open-source project in the future.

All results presented in the paper were obtained using a laptop equipped with a 12-core Intel Core i7 CPU and 64 GB of RAM. Fig. 22 illustrates the program runtime (in minutes) as a function of the number of triangles in the input mesh. It can be observed that our algorithm exhibits linear scalability with an increasing number of triangles. Even for the largest triangulation in our investigation, consisting of 23 million triangles, the algorithm converged to a manifold surface in under 10 min.

5. Applications and variants

5.1. Fluid volume extraction

Fluid volume extraction is yet another excellent application of our shrink wrapping algorithm. If the goal is to generate the fluid volume or topological holes in a geometry, simple boolean operations help extract these volumes. There are practical difficulties in extracting topological holes in a geometry (multiple holes) since there will be a lot of noise to sift through. However, suppose the industrial practitioner is interested in extracting a single fluid volume like a fluid volume of an interior of a car. In that case, it is possible to automate the process entirely. Once we have a Genus zero shrink-wrapped surface, one can lay out a straightforward algorithm with the following steps

1. Subtract the genus zero shrink-wrapped geometry from the input geometry
2. Split the result based on connectivity and compute the component volumes
3. Largest volume geometry is the fluid volume

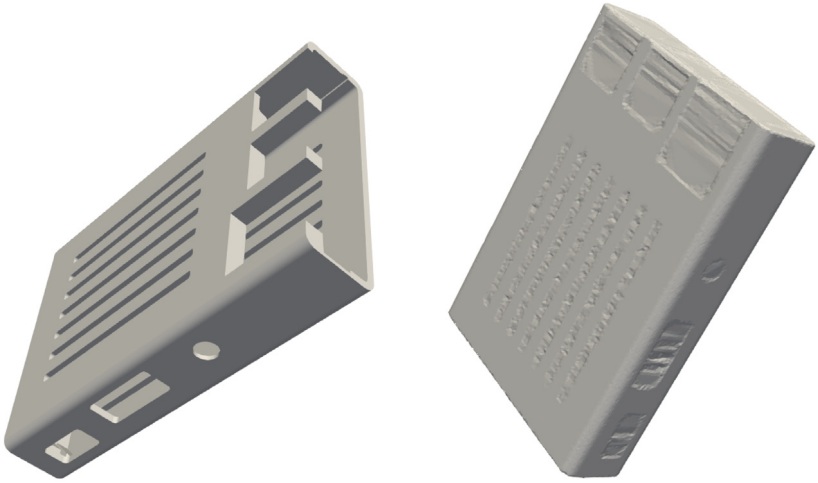
We did not implement any boolean operations for this algorithm and used the existing functionalities from CGAL [18].

Smoothed particle hydrodynamics (SPH) is a meshless method requiring a volumetric point distribution for numerical simulation. In example 1, we show a partial car in Fig. 23 that has been shrink-wrapped, and its fluid volume has been extracted for a smoother particle hydrodynamic simulation.

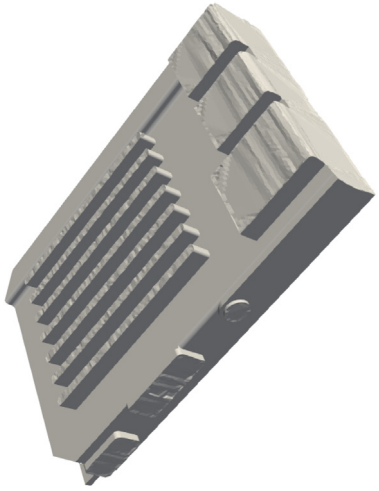
In the second example, we show a case for raspberry pi, its shrink-wrapped geometry, and the subsequent fluid volume in Fig. 24. Again, it can be seen that our algorithm produces a spotless fluid volume. The results can be further de-noised to turn them into developable surfaces.

5.2. External aerodynamic simulation

One of the primary focuses of our investigation is external aerodynamic simulations. Since they do not require all the internal components of a geometry, a simplified geometry can be considered during early prototyping. We ran the RANS simulations



(a) Raspberry pi case geometry (b) Wrapped Geometry



(c) Fluid volume

Fig. 24. A raspberry pi case and its fluid volume.

on a generic shrink-wrapped car geometry using OpenFoam [22]. The car geometry was meshed using the snappyHexMesh tool from OpenFoam with a base refinement of 10 cells in all three directions. We considered the entire car geometry for numerical simulation without using symmetry boundary conditions. We used a steady-state incompressible SIMPLE solver for solving the Reynolds Averaged Navier Stokes equation with a k-ω SST turbulence model. A velocity inlet with a velocity of 20 ms was used as the boundary condition. The shrink-wrapped geometry produces physically consistent results, as shown in the Fig. 25. We have not made a rigorous mathematical analysis or experimental verification for these simulations. We only performed this simulation to show the suitability of shrink-wrapped geometries for computational fluid dynamic simulations.

6. Comparison against similar approaches

6.1. Point2Mesh

Many recent papers in geometry processing use deep learning-based approaches to solve geometric problems. One such recent article is *Point2Mesh*[6] where the authors shrink wrap an oriented point cloud based on self-similarity. Their algorithm is built on mesh-based convolutional neural networks and similar algorithms found in computer vision. We extended our shrink wrapping algorithm for point clouds to make a fair comparison. Since we rely on generalized winding numbers for inside–outside segmentation, there is a straightforward extension to point clouds. Usually, this is done by computing point areas using a Voronoi diagram [14]. However, we found that such a complex approximation is not always required. We compute a series of local triangulations and consistently ensure their orientation using a greedy approach. Hence, our algorithm does not require an oriented point cloud. This modification ensures that we do not have to modify the rest of our shrink wrapping algorithm. Once the inside–outside segmentation is done in the octree, the rest of the algorithm remains the same. We chose the same geometries as the authors and found that we produce similar quality results in most cases. While we provided a variety of heuristics to avoid this in a surface mesh-based approach, we did not thoroughly investigate the same for point clouds since it was beyond the scope of our work.

An observation can be made that our algorithm complements the authors' work nicely. If our algorithm is considered an initial priori, it improves the convergence speed of *Point2Mesh* algorithm. For example, their algorithm relies on an initial mesh computed based on a convex hull approach and converges to a ground truth based on self-similarity. However, our algorithm's mesh before the projection stage is a better priori. It also converges the *Point2Mesh* [6] algorithm in a fraction of the time.

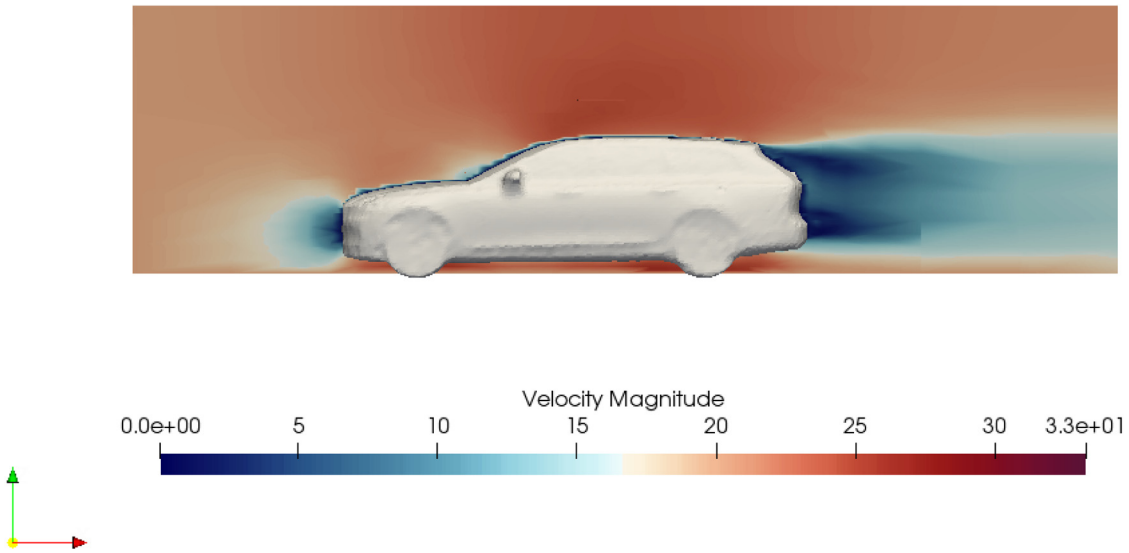


Fig. 25. External aerodynamic simulation of a generic car model (shrink wrapped).

Fig. 26. Few point cloud geometries from *Point2Mesh* [6] along with its output. Our wrap algorithm produces equally smooth results except for a few artifacts created as a result of morphological operators.

It can also be noticed that their approach is not meant for mechanical parts, and features found in CAD geometries cannot be preserved without significant modifications. Our goal is to produce genus-zero surfaces for aerodynamic simulations. We optionally provide variants that allow various levels of control over the genus of the wrapped surface. However, their approach is strictly a surface construction approach and does not consistently produce zero surfaces. It can only be achieved by stopping the algorithm halfway; the reconstruction might not be accurate globally in such cases, and a projection might be required (see Fig. 26).

6.2. Alpha wrapping

Pierre Alliez et al. [7] recently proposed a practical shrink wrapping algorithm based on three-dimensional alpha wrapping.

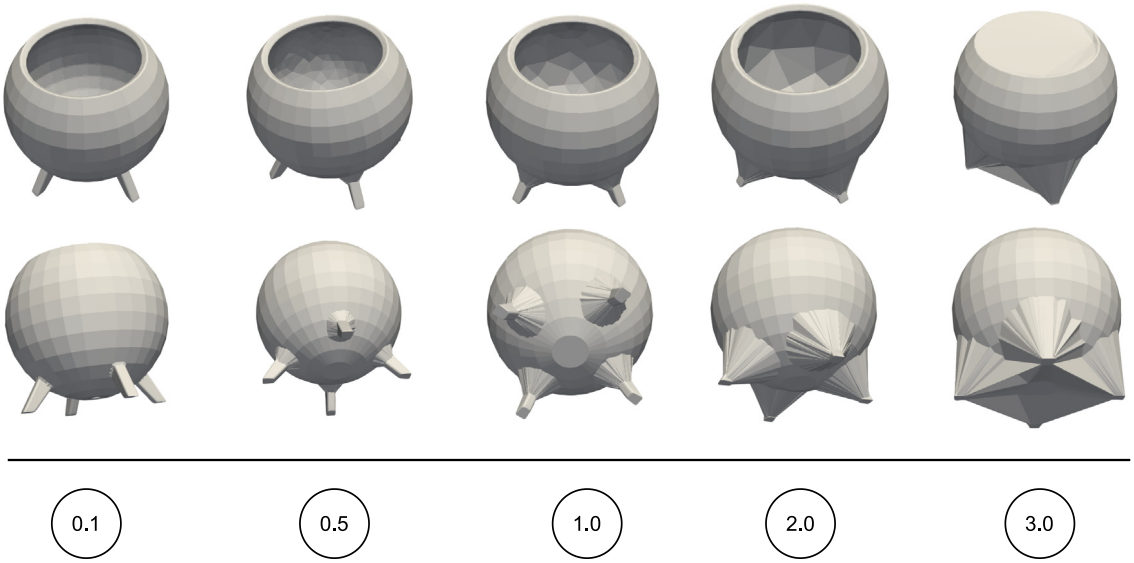


Fig. 27. Influence of increasing alpha value on a geometry. The offset value is fixed at 0.000001 for all the cases.

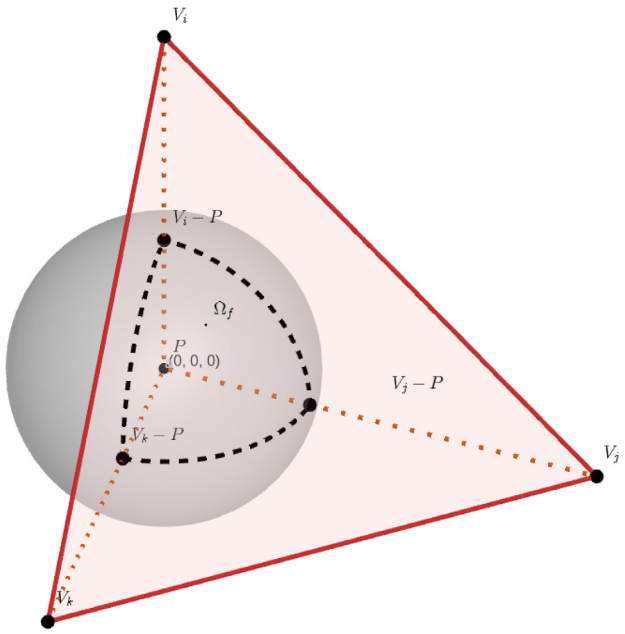


Fig. 28. Solid angle of a query point.

Like ours, their algorithm is controlled by alpha and offset values that determine the wrap's accuracy. Even though their approach can handle a variety of inputs such as triangles, line segments, and points (due to the advantage of being implemented inside CGAL), we limit our comparison to triangle meshes. The term alpha controls the accuracy of the wrapping algorithm. This is similar to the topological sphere level or the number of operations in our algorithm. An increase in alpha leads to a better wrapping of the geometry.

For comparison, we show the effect of increase in alpha value in Fig. 27 for the same geometry used in Section 4.3.1. It can be noticed that while an increase in alpha closes the bigger hole, the bottom of the geometry gets severely distorted. An interesting observation could be made for the alpha value of 0.1, where the bottom of the geometry starts to form artifacts similar to the artifacts produced by morphological operators. As we have shown in previous examples with our algorithm, such artifacts only start to show when the number of morphological operations is very high. However, in the case of alpha wrapping, even extremely low alpha values produce such artifacts in concave areas of the geometry. Our algorithm closes both the holes of the geometry, as shown in Fig. 21 without heavily distorting the bottom of the geometry. In our experiments, we noticed that a very low value of alpha and offset makes alpha wrapping an excellent surface reconstruction algorithm. Its two-manifold guarantee and ability to handle extremely degenerate geometries add to the advantages. However, it is not a suitable algorithm for producing hole-free geometry approximation. In the presence of concavities, alpha wrapping destroys the geometry significantly.

7. Limitations

Since the proposed algorithms are built on top of morphological operators, they inherit the drawbacks of mathematical morphology. For example, the erosion stops in concave regions once it hits the closest triangle in the mesh. This leads to over closing of specific features in the mesh. However, since the primary application for our algorithms is external aerodynamic simulations, these do not make a massive difference in the macro scale. Our smooth and developable variants alleviate most of these issues and produce good quality meshes suitable for various practical applications.

8. Conclusion

We presented a practical algorithm that can perform Genus simplified shrink wrapping for polyhedral surfaces with the help of morphological operators. We also show that these algorithms extend easily for point clouds. One can implement the algorithms proposed in this paper in the same mesh used for numerical simulation, thereby avoiding another expensive volumetric mesh generation process. The algorithms also run at a linear runtime and are not heavily CPU bound. Furthermore, user-defined constraints and additional interactivity could further improve the algorithm's output quality. In addition, our smooth and developable wrap variants can significantly benefit engineers in the early prototyping stage. Finally, the algorithm's fluid volume extraction variant can significantly benefit industrial fluid dynamic practitioners.

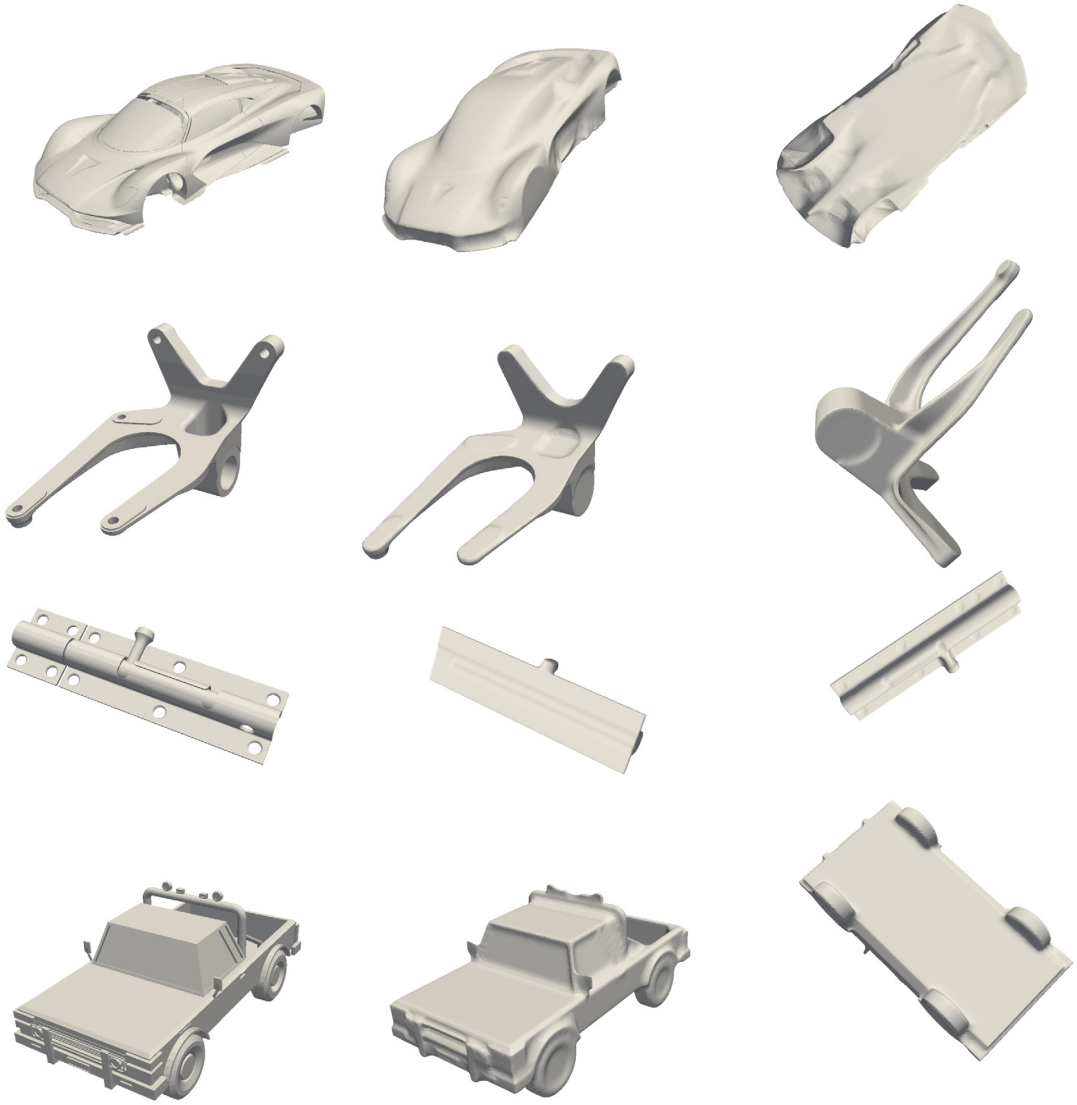


Fig. 29. A variety of geometries and their shrink wrapped results.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

The authors are unable or have chosen not to specify which data has been used.

Acknowledgments

Österreichische Forschungsförderungsgesellschaft, Austria has funded this research under an industrial Ph.D. grant titled **"HIOMESH"**.

Appendix A. Generalized winding number based solid angle for surface segmentation

We rely on a classical differential geometry idea called winding numbers, which uses solid angles for surface segmentation. For a given surface S, for a query point p, the solid angle is the signed surface area of the projection of S onto the unit sphere centered at p as shown in Fig. 28. We rely on its definition in discrete setting [23].

$$\begin{aligned}
&\omega(\mathbf{p}) = 2 * \tan^{-1}\left(\frac{det([\mathbf{abc}])}{abc + (\mathbf{a}.\mathbf{b})c + (\mathbf{b}.\mathbf{c})a + (\mathbf{c}.\mathbf{a})b}\right)\\
&Triangle = \{v_i, v_j, v_k\}\\
&\mathbf{a} = v_i - p, \mathbf{b} = v_j - p, \mathbf{c} = v_k - p\\
&a = \|\mathbf{a}\|\\
&b = \|\mathbf{b}\|\\
&c = \|\mathbf{c}\|
\end{aligned} \tag{A.1}$$

Given this relation for solid angle given by $\omega(\mathbf{p})$, we can compute winding number as follows

$$w(\mathbf{p}) = \sum_{n=1}^{nTriangles} \frac{1}{4\pi}\omega_f(\mathbf{p})$$

For every query point, the direct implementation $w(\mathbf{p})$ would require the contribution of all triangles in the surface mesh. Since this would yield a solution with time complexity of $O(n^2)$, we rely on the work of Gavin Barill et al. [14]. They proposed a fast multipole method [24] style implementation that uses direct

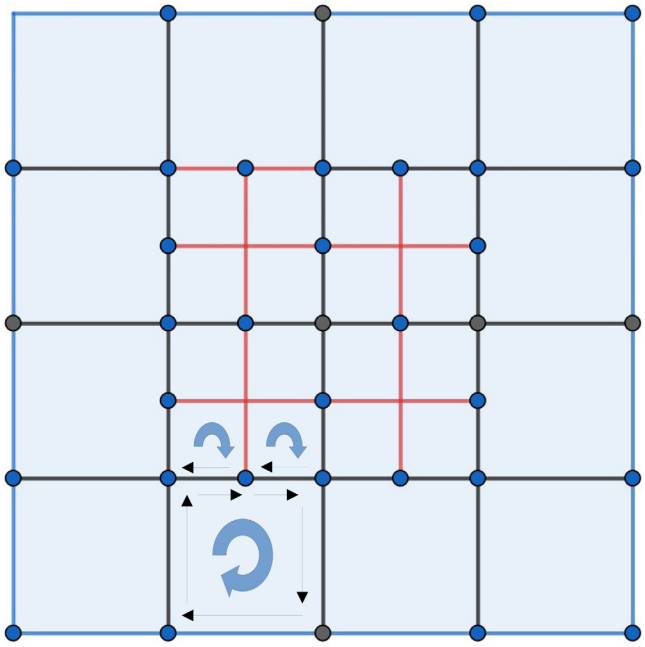


Fig. 30. An adaptively refined Cartesian mesh with a half-edge like internal representation.

computation for triangles near the query point and approximates the result everywhere else, making it a $O(log(n))$ algorithm.

Appendix B. Various input geometries and their shrink wrapped results

We show a few geometries and their shrink-wrapped results. The results shown below are shrink-wrapped with a topological sphere level of 15. Our algorithm could produce a genus zero surface consistently in all the cases shown below. It also creates a two-manifold mesh suitable for external aerodynamics or finite element analysis simulations (see Fig. 29).

Appendix C. Morphology and other image processing algorithms on adaptive meshes

Adapting image processing algorithms for volumetric applications has become increasingly common. While images inherently have a 1:1 connectivity everywhere, this can pose challenges when adapting algorithms for adaptive grids. However, with the correct data structure, the adaptation process becomes straightforward.

In Fig. 30, we illustrate an example using a quadtree data structure. A half-edge like representation is employed to establish regular grid-like connectivity from regular cells to hanging cells. This approach is similar to the representation used for unstructured T-Splines [25], where an edge of a regular cell can be split into multiple half edges, ensuring a unique copy of the edge for every hanging cell. This representation is also compatible with unbalanced grids. While alternative memory-efficient approaches can be implemented using tree traversal, we found the demonstrated approach to be efficient enough for our investigation (see Fig. 2).

References

[1] Attene M, Campen M, Kobbelt L. Polygon mesh repairing: An application perspective. ACM Comput Surv 2013;45(2). http://dx.doi.org/10.1145/2431211.2431214.

[2] Esteve J, Brunet P, Vinacau A. Approximation of a variable density cloud of points by shrinking a discrete membrane. Comput Graph Forum 2005;24(4):791–807.

[3] Nooruddin F, Turk G. Simplification and repair of polygonal models using volumetric techniques. IEEE Trans Vis Comput Graphics 2003;9(2):191–205. http://dx.doi.org/10.1109/TVCG.2003.1196006.

[4] Lee YK, Lim CK, Ghazialam H, Vardhan H, Eklund E. Surface mesh generation for dirty geometries by shrink wrapping using cartesian grid approach. In: Pébay PP, editor. Proceedings of the 15th international meshing roundtable. Berlin, Heidelberg: Springer Berlin Heidelberg; 2006, p. 393–410.

[5] Wang ZJ, Srinivasan K. An adaptive Cartesian grid generation method for 'Dirty' geometry. Internat J Numer Methods Fluids 2002;39(8):703–17. http://dx.doi.org/10.1002/fld.344.

[6] Hanocka R, Metzer G, Giryes R, Cohen-Or D. Point2Mesh: A self-prior for deformable meshes. ACM Trans Graph 2020;39(4). http://dx.doi.org/10.1145/3386569.3392415.

[7] Alliez P, Cohen-Steiner D, Hemmer M, Portaneri C, Rouxel-Labbé M. 3D alpha wrapping. In: CGAL user and reference manual. 5.5th ed. CGAL Editorial Board; 2022.

[8] Vijai Kumar S, Vuik C. A simple and fast hole detection algorithm for triangulated surfaces. J Comput Inf Sci Eng 2021;21(4). http://dx.doi.org/10.1115/1.4049030, 044502.

[9] Najman L, Talbot H. Introduction to mathematical morphology. In: Mathematical morphology. John Wiley & Sons, Ltd; 2013, p. 1–33. http://dx.doi.org/10.1002/9781118600788.ch1.

[10] Jeulin D. Analysis and modeling of 3D microstructures. In: Mathematical morphology. John Wiley & Sons, Ltd; 2013, p. 421–44. http://dx.doi.org/10.1002/9781118600788.ch19.

[11] Chen Z, Panozzo D, Dumas J. Half-space power diagrams and discrete surface offsets. IEEE Trans Vis Comput Graphics 2020;26(10):2970–81. http://dx.doi.org/10.1109/TVCG.2019.2945961.

[12] Sellán S, Kesten J, Sheng AY, Jacobson A. Opening and closing surfaces. ACM Trans Graph 2020;39(6). http://dx.doi.org/10.1145/3414685.3417778.

[13] Gärtner B. Fast and robust smallest enclosing balls. In: Proceedings of the 7th annual European symposium on algorithms. London, UK, UK: Springer-Verlag; 1999, p. 325–38.

[14] Barill G, Dickson N, Schmidt R, Levin DI, Jacobson A. Fast winding numbers for soups and clouds. ACM Trans Graph 2018.

[15] Liu S, Wang CCL. Fast intersection-free offset surface generation from freeform models with triangular meshes. IEEE Trans Autom Sci Eng 2011;8(2):347–60. http://dx.doi.org/10.1109/TASE.2010.2066563.

[16] Lorensen WE, Cline HE. Marching cubes: A high resolution 3D surface construction algorithm. In: Proceedings of the 14th annual conference on computer graphics and interactive techniques. New York, NY, USA: Association for Computing Machinery; 1987, p. 163–9. http://dx.doi.org/10.1145/37401.37422.

[17] Blanco JL, Rai PK. Nanoflann: a C++ header-only fork of FLANN, a library for Nearest Neighbor (NN) with KD-trees. 2014, https://github.com/jlblancoc/nanoflann.

[18] The CGAL Project. CGAL user and reference manual. 5.3th ed. CGAL Editorial Board; 2021, URL https://doc.cgal.org/5.3/Manual/packages.html.

[19] Boltcheva D, Lévy B. Surface reconstruction by computing restricted Voronoi cells in parallel. Comput Aided Des 2017;90:123–34. http://dx.doi.org/10.1016/j.cad.2017.05.011, URL https://www.sciencedirect.com/science/article/pii/S0010448517300829, SI:SPM2017.

[20] Stein O, Grinspun E, Crane K. Developability of triangle meshes. ACM Trans Graph 2018;37(4). http://dx.doi.org/10.1145/3197517.3201303.

[21] Zhang J, Deng B, Hong Y, Peng Y, Qin W, Liu L. Static/dynamic filtering for mesh geometry. IEEE Trans Vis Comput Graphics 2019;25(4):1774–87. http://dx.doi.org/10.1109/TVCG.2018.2816926.

[22] The OpenFOAM Foundation. Openfoam v8 user guide. 2021, URL https://cfd.direct/openfoam/user-guide.

[23] Jacobson A, Kavan L, Sorkine-Hornung O. Robust inside-outside segmentation using generalized winding numbers. ACM Trans Graph 2013;32(4). http://dx.doi.org/10.1145/2461912.2461916.

[24] Carrier J, Greengard L, Rokhlin V. A fast adaptive multipole algorithm for particle simulations. SIAM J Sci Stat Comput 1988;9(4):669–86. http://dx.doi.org/10.1137/0909044.

[25] Wang W, Zhang Y, Du X, Zhao G. An efficient data structure for calculation of unstructured T-spline surfaces. Vis Comput Ind Biomed Art 2019;2(1):2. http://dx.doi.org/10.1186/s42492-019-0010-0.