A New Procedure to Compute Imprints in Multi-sweeping Algorithms*

Eloi Ruiz-Gironés, Xevi Roca, and Josep Sarrate

Laboratori de Càlcul Numèric (LaCàN),
Departament de Matemàtica Aplicada III,
Universitat Politècnica de Catalunya,
Jordi Girona 1-3, E–08034 Barcelona, Spain
Tel.: 34-93-401 69 11; Fax: 34-93-401 18 25
{eloi.ruiz,xevi.roca,jose.sarrate}@upc.edu

Abstract. One of the most widely used algorithms to generate hexahedral meshes in extrusion volumes with several source and target surfaces is the multi-sweeping method. However, the multi-sweeping method is highly dependent on the final location of the nodes created during the decomposition process. Moreover, inaccurate location of inner nodes may generate erroneous imprints of the geometry surfaces such that a final mesh could not be generated. In this work, we present a new procedure to decompose the geometry in many-to-one sweepable volumes. The decomposition is based on a least-squares approximation of affine mappings defined between the loops of nodes that bound the sweep levels. In addition, we introduce the concept of computational domain, in which every sweep level is planar. We use this planar representation for two purposes. On the one hand, we use it to perform all the imprints between surfaces. Since the computational domain is planar, the robustness of the imprinting process is increased. On the other hand, the computational domain is also used to compute the projection onto source surfaces. Finally, the location of the inner nodes created during the decomposition process is computed by averaging the locations computed projecting from target and source surfaces.

Keywords: Finite element method, mesh generation, hexahedral mesh, multi-sweeping, computational domain.

1 Introduction

Extrusion geometries often appear in numerical simulation processes. These volumes are usually created using CAD packages that allow to extrude a surface along a sweep path. These one-to-one geometries are delimited by a *source surface*, a *target surface* and a series of *linking sides*, see Fig. 1.

* This work was partially sponsored by the Spanish *Ministerio de Ciencia e Innovación* under grants DPI2007-62395, BIA2007-66965 and CGL2008-06003-C03-02/CLI.

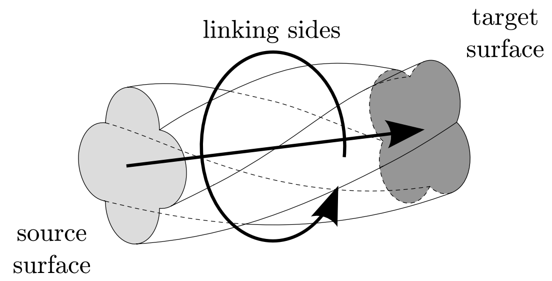


Fig. 1. Classification of surfaces defining a one-to-one sweep volume

In the last years, several methods have appeared to mesh *many-to-many* sweep volumes [1, 2, 3, 4]. That is, extrusion volumes that contain many source and target surfaces. These methods rely on the decomposition of the volume into sub-volumes that are meshable by one-to-one [5, 6, 7, 8, 9] or many-to-one [10] sweep techniques. Then, each sub-volume is meshed separately. In general, the decomposition of a many-to-many sweeping is performed by projecting target surfaces onto the corresponding source surfaces. Then, the projected target surfaces and source surfaces are imprinted in order to determine the decomposition of the volume. Finally, each sub-volume is meshed separately using a many-to-one sweep scheme. In fact, each sub-volume is further decomposed into barrels, and each barrel is meshed using a one-to-one sweep scheme.

However, the algorithm is highly dependent on the location of the inner nodes created during the decomposition process. The quality of the imprints is also affected by the location of inner nodes. Inaccurate locations may lead to erroneous imprints. Thus, a low quality mesh with inverted elements may be generated. Usually, this effect appears when there are non-planar sweep levels or highly curved surfaces in the geometry. In this work, we present a new algorithm to decompose a many-to-many geometry that overcomes these drawbacks.

The method is based on a least-squares approximation of an affine mapping defined between the loops of nodes that bound the sweep levels according to [8]. The decomposition is performed using a two-step procedure. First, the target surfaces are projected in the sweep direction to the source surfaces. In the second step, the nodes on the source surfaces are projected back to the target surfaces. Then, the final location of inner nodes is the weighted average of the position of nodes projecting them from target and from source surfaces. During the first step of the decomposition process it is necessary to compute the imprints between the source and target surfaces. The main contribution of this work is to define the computational domain of a loop of nodes as a planar representation of them. Since the computational domain is planar, the imprinting process becomes more robust. In addition, we also use the computational domain to project inner nodes onto source surfaces.

2 The Multi-sweeping Method

This section presents an outline of the proposed multi-sweeping method. The algorithm is performed in five steps:

(i) Surfaces classification.
(ii) Linking sides meshing.
(iii) Loop face projection and imprinting.
(iv) Loop edge meshing and volume decomposition into many-to-one sub-volumes.
(v) Meshing all many-to-one sub-volumes.

Although this work is focused on the decomposition steps (iii) and (iv), we also present an outline of the whole multi-sweeping method. The first step of the algorithm is the classification of the surfaces as source, target and linking sides. The classification is accomplished using the procedure presented in [11]. This procedure is performed as follows. First, it finds a non-submappable surface and classifies it as source surface. Then, the algorithm proceeds to classify the adjacent surfaces in an advancing front manner depending on the angle between the adjacent surfaces. When the surfaces of the geometry are classified, the linking sides are meshed using the submapping algorithm [12, 13]. Figure 2(a) shows a simple multi-sweep geometry with its linking sides meshed.

Once the linking sides are meshed the decomposition process begins. The standard decomposition process starts at the target surfaces and ends at the source surfaces. In contrast, the final meshing process starts at the source surfaces of the many-to-one sub-volumes and ends at the target surfaces.

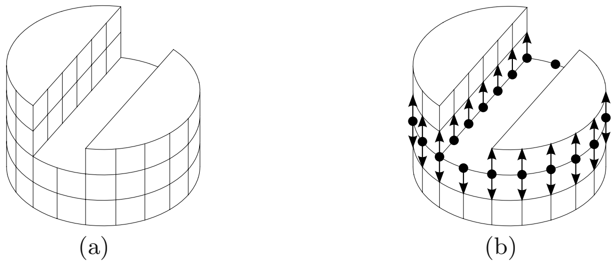


Fig. 2. (a) Mesh generated on the linking sides using the submapping method. (b) A simple multi-sweep geometry with a row of sweep nodes.

3 Basic Definitions

In this section we provide four basic definitions that will be used through this work. We briefly review the concepts of sweep node, loop geometry engine and control loop previously introduced in [1]. In addition, we introduce the new concept of computational domain for multi-sweeping.

3.1 Sweep Node

Sweep nodes are a computational structure that stores the vertical connectivity of the linking sides mesh. Each sweep node contains a pointer to the next sweep node in the sweep direction and another pointer to the previous sweep node in the counter-sweep direction (some of these pointers may be null). Figure 2(b) presents a multi-sweep geometry with a row of sweep nodes. Note that there are sweep nodes that do not point to a sweep node directly above or below.

3.2 Loop Geometry Engine

The loop geometry engine is a data structure that represents source, target and mid-level faces as loops of sweep nodes. In addition, a graph is used in order to define the topology of such loop faces. For instance, Fig. 3(a) shows a surface represented by: *i)* a loop face; *ii)* a loop wire that describes the boundary of the loop face; *iii)* an ordered list of four loop edges that defines the loop wire, and *iv)* four loop vertices that define the initial and final points of each loop edge. Figure 3(b) details this relationship using the topology graph. For instance, the loop wire uses loop edge 1 and, conversely, loop edge 1 is included in the loop wire.

Loop vertices are represented by a sweep node that provides information about its location. Loop edges are defined by an ordered list of sweep nodes.

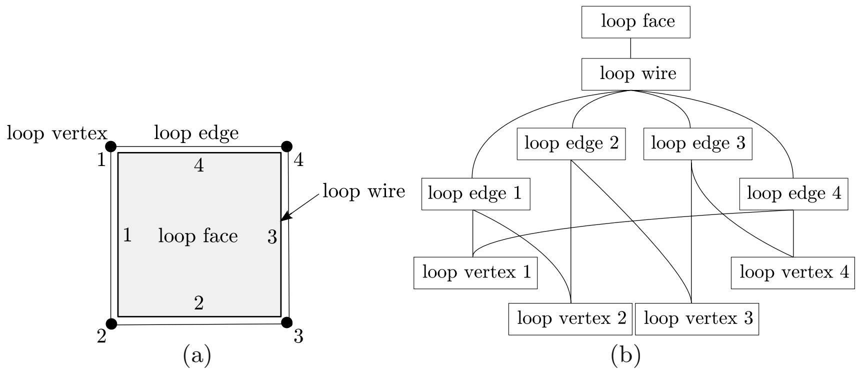


Fig. 3. (a) A loop face. (b) Associated topology graph.

The list of nodes provides a discretization of the loop edge. Loop wires are defined by a closed loop of loop edges. Loop faces are represented by a loop wire that defines the outer boundary and several loop wires that define inner boundaries.

The structure of the loop geometry engine is similar to a geometry engine except that not all the elements in the loop geometry have an underlying geometrical representation. This engine provides both geometrical and topological information about the loop geometry. In addition, this structure is responsible for creating loop geometry and maintaining the topology graph during the whole algorithm, see [1] for more details.

3.3 Control Loop

Control loops are the loops of sweep nodes that bound each of the sweep layers of the volume. While one-to-one sweep volumes have a single control loop for each layer, the number of control loops in multi-sweeping volumes may differ from layer to layer, see Fig. 4(a). The main objective of the control loops is to define the projection between two consecutive sweep layers. To this end, control loops are composed by a loop of sweep nodes that define the projection (black dots in Fig. 4(a)) and, for implementation purposes, a list of sweep nodes to project (white dots in Fig. 4(a)).

In a given level of the decomposition process we may have both source and target loop faces. In these cases, given the sets of target and source loop faces, $(\tau_i)_{i=1,\ldots,n}$ and $(\sigma_i)_{i=1,\ldots,m}$, respectively, the control loops are computed as:

(i) Compute provisional target control loops, L_t, as follows:
- The loops of nodes of control loops are defined as the Boolean union, $L_t = \bigcup_{i=1,\ldots,n} \tau_i$, see [14].
- The nodes to project are those that belong to more than one loop face.

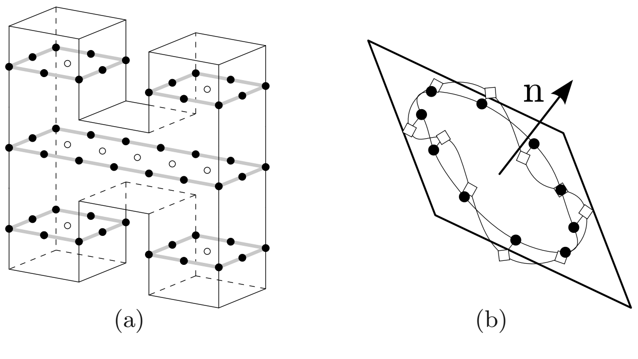


Fig. 4. (a) Five control loops in a multi-sweep geometry. (b) Physical sweep nodes (white squares) and computational sweep nodes (black circles).

(ii) Compute provisional source control loops, L_s, using the procedure detailed in step (i).
(iii) We collect the nodes of control loops in L_s (both the nodes that describe the loops and the nodes to project) that are not included in L_t. Then we insert these nodes into the corresponding control loops in L_t.

3.4 Computational Domain

It is important to point out that control loops in real geometries are usually non-planar. Therefore, the imprinting operations needed in the decomposition process may lead to inaccurate representations due to tolerance definitions. To overcome this drawback we introduce the concept of computational domain.

The *computational domain* of a given control loop is a planar representation of it. Figure 4(b) shows a curved control loop in the physical domain (white squares) and its representation in the computational space (black circles). The computational domain is used to project the sweep nodes through the volume and to perform the imprinting between the surfaces of the geometry. Since the control loops in the computational domain are planar, the proposed imprinting process is more robust, see Sect. 4. The main reason is that skew lines (neither parallel nor concurrent) do not exist in a bi-dimensional space. Therefore, computing the intersection of segments in the computational domain is less affected by tolerance errors than in the physical domain. In addition, the *winding number* algorithm [15] can be used to test if a node is inside a loop of nodes. This algorithm loses its accuracy and robustness for non-planar loops of nodes.

In reference [8] the *pseudo-area* vector, $\mathbf{a}$, of a loop of points $\{\mathbf{x}_i\}_{i=1,\dots,n}$ is defined as

$$\mathbf{a} = \sum_{i=1}^{n} \mathbf{x}_i \times \mathbf{x}_{i+1},$$

where $\mathbf{x}_{n+1} = \mathbf{x}_1$. The *pseudo-normal* vector is defined as

$$\mathbf{n} = \frac{\mathbf{a}}{||\mathbf{a}||}. \tag{1}$$

The construction of the computational domain of a given control loop is performed in the following manner. First, we compute the pseudo-normal of the loop nodes that defines the control loop, $\mathbf{n}$. Second, we define the computational position of $\mathbf{x}_i$, for $i = 1, \dots, n$ as

$$\overline{\mathbf{x}}_i = \mathbf{x}_i - \langle \mathbf{x}_i, \mathbf{n} \rangle \mathbf{n}, \tag{2}$$

where $\langle \cdot, \cdot \rangle$ denotes the dot product. Note that we project points $\mathbf{x}_i$, for $i = 1, \dots, n$, on the plane defined by the pseudo-normal vector. From the

3D representation of the computational domain given by (2), it is straightforward to compute a 2D representation, in (ξ, η) coordinates, of the computational domain.

Two remarks on the proposed method have to be made. First, each control loop has its own computational domain, even when control loops are located in the same sweep level. Second, in order to construct the computational domain, we need to compute the pseudo-normal vector. This vector can be defined if a control loop is set. Therefore, our method can be applied to all extrusion geometries such that all levels are bounded by control loops. Note that periodic surfaces such as cylinders or spheres that expand from inside to outside, or vice versa, do not meet this condition. Thus, the proposed method can not be applied to these cases because the control loop is not set. In these situations, we can always split the geometry in two sub-volumes such that the control loops are properly defined.

4 Loop Face Projection and Imprinting

4.1 Loop Face Projection

Starting at target surfaces, we compute the associated loop faces as detailed in Sect. 3.2. Then, we compute the corresponding control loops according to Sect. 3.3. When the control loops are constructed, the algorithm proceeds to build their associated computational domain, see Sect. 3.4.

To project the sweep nodes between layers we use the sweeping scheme presented in [7, 8]. It is important to point out that the projection of sweep nodes is performed both in the physical domain and in the 3D representation of the computational domain. The computational and physical locations are stored for each sweep node. The projection in the physical space is performed in order to capture the shape of the target and/or source surfaces. The projection in the computational space is performed in order to obtain accurate and robust imprints. When a sweep node is projected to the next level, the projected sweep node becomes the next sweep node of the original one. Conversely, the original sweep node is the previous sweep node of the projected one.

In each level we check if new target or source loop faces have to be added to the existing control loops. If this is the case we update the control loops according to Sect. 3.3. For each one of these control loops an imprinting process is performed, as detailed in Sect. 4.2.

Once all the source surfaces are reached the geometry is completely decomposed. However, the location of the inner sweep nodes created during the decomposition process has to be improved. To this end, the source surfaces are projected back to the target surfaces and the final location of inner sweep nodes is computed as a weighted average of the computed locations projecting from target and source surfaces, see Sect. 4.5.

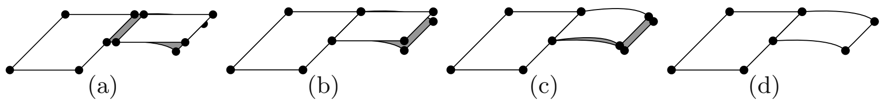


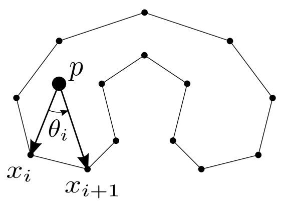
Fig. 5. Representation of the imprinting process in the physical domain. (a) Two target loop faces (white) and a source loop face (grey). (b) Target loop edges intersection. (c) Projection of target sweep nodes on source faces. (d) Collapse of target sweep nodes and source sweep nodes.

4.2 Loop Face Imprinting Pre-process

Before starting the actual imprinting process, target and source loop faces of the same control loop have to be pre-processed. It is important to point out that each one of the following steps is performed in the 3D representation of the computational domain. That is, using $\overline{\mathbf{x}}$ coordinates. Figure 5 shows a representation of the imprinting pre-process in the physical domain. Figure 5(a) shows two target loop faces (white) and one source loop face (grey) ready to be pre-processed. First, the loop edges of target loop faces are intersected with each other in order to obtain a conformal model (Fig. 5(b)). Second, the nodes that define the target loop faces are projected to the corresponding source loop faces (Fig. 5(c)). Section 4.3 further details the procedure to project those nodes. Finally, we search a sweep node on a target loop face and a sweep node on a source loop face that are closer than a given tolerance. Then the target node is collapsed with the source node (Fig. 5(d)). This tolerance is defined as $h/5$, where h is the prescribed element size. On the one hand, if the tolerance parameter is too big, we may collapse nodes that are too far and the imprinting process may fail. On the other hand, if the tolerance is too small, we may miss some collapsing nodes.

4.3 Mapping of Sweep Nodes from the Computational Domain to a Source Surface

This section is focused on a new procedure to project sweep nodes of target loop faces to source loop faces of the same control loop. For a given source loop face, σ, we need to detect which nodes of target loop faces lie inside σ. Since loop faces are planar in the computational domain we use the winding number algorithm, see Fig. 6 for a graphical representation of this algorithm. Given a test point, p, the algorithm adds the angles between p and two consecutive points in the loop. If the result equals 0, the point is outside. Else, if the result equals 2π, the point is inside. That is, the algorithm counts the number of turns around the test point. This algorithm is robust when dealing with planar loops of nodes. Note that in 3D, the number of turns is not well defined and, for this reason, the winding number algorithm may fail.



$$\sum_{i=1}^{N} \theta_i = \begin{cases} 0 & \text{point is outside} \\ 2\pi & \text{point is inside} \end{cases}$$

Fig. 6. Graphical representation of the winding number algorithm

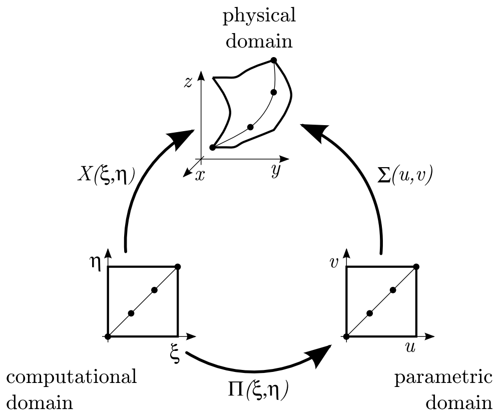


Fig. 7. Mapping of sweep nodes from the computational domain to a source surface

Given a list of sweep nodes inside a source loop face, we have to project them on the geometrical surface. Let (ξ, η) and (x, y, z) be two coordinate systems of the 2D representation of the computational domain and the physical domain, respectively (see Fig. 7). Therefore, we need to compute a mapping $X(\xi, \eta)$ to project the sweep nodes of the target loop faces onto the physical domain. Note that all the source loop faces represent a geometric surface. Hence, instead of projecting the sweep nodes directly from the computational space, (ξ, η), to the physical space, (x, y, z), we propose to compute a mapping $\Pi(\xi, \eta)$ to project the nodes to the parametric space of the surface. Then, using the parameterization of the surface, $\Sigma(u, v)$, we finally project the nodes to the physical domain. That is

$$X = \Sigma \circ \Pi.$$

Note that the mapping $\Pi(\xi, \eta)$ is unknown. In order to compute the mapping $\Pi(\xi, \eta)$, we approximate it by an affine mapping $\widetilde{\Pi}(\xi, \eta)$. This affine mapping is determined by means of a least-squares approximation of a linear mapping between the 2D computational domain representation of the source loop face and its representation in the parametric space of the source surface, see [6] for details. That is we approximate $X(\xi, \eta)$ as

$$X \approx \Sigma \circ \widetilde{\Pi}.$$

4.4 Loop Face Imprinting

The result of the imprinting process between target loop faces, $(\tau_i)_{i=1\dots n}$, and source loop faces, $(\sigma_i)_{i=1\dots m}$, is stored in three lists of loop faces: Θ_o, Θ_t and Θ_s. List Θ_o contains the loop faces that come from intersections of a target and a source loop face. The loop faces included in Θ_o are called *overlap* loop faces. List Θ_t contains the sections of target loop faces that do not intersect a source loop face. These faces are the new target loop faces that replace the old target loop faces, $(\tau_i)_{i=1\dots n}$. Finally, list Θ_s contains the section of source loop faces that do not intersect a target loop face. This list contains the new source loop faces that replace the old source loop faces, $(\sigma_i)_{i=1\dots m}$. The loop face imprinting operation is based on segment intersections, see [1, 14] for more details. The main difference of the presented method is that all of the calculations to obtain the imprints are performed in the 3D representation of the computational domain. Hence, the imprinting process becomes more accurate and robust as stated in Sect. 4.3.

In addition, we create a new graph that relates the new loop faces created in the imprinting process and the original loop faces from which they come from. We call this graph the *loop face partitioning graph.* This graph allows us to recover the set of loop faces in which a loop face is decomposed. Moreover, it also permits us to recover the loop faces in which a given loop face is included. Figure 8(a) shows the imprinting process of a target loop face, T, and a source loop face, S. In this case, $\Theta_o = \{O\} = \{T \cap S\}$, $\Theta_t = \{T'\} = \{T - S\}$ and $\Theta_s = \{S'\} = \{S - T\}$. Figure 8(b) shows the corresponding loop face partitioning graph. Note that loop face O is a section of S and T. Conversely, T is partitioned in loop faces T' and O.

In order to illustrate the differences between the imprinting process in the physical and computational domain, Fig. 10(a) shows a curved loop face, T, and a planar loop face, S in the physical domain. The intersection between

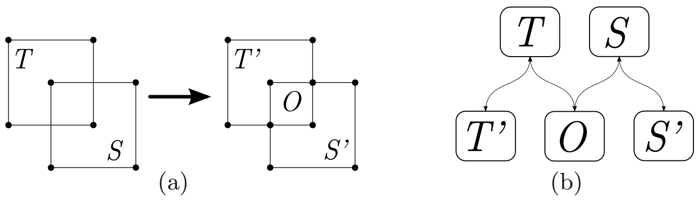


Fig. 8. (a) Imprinting process of a target loop face, T, and a source loop face, S. (b) Associated loop face partitioning graph.

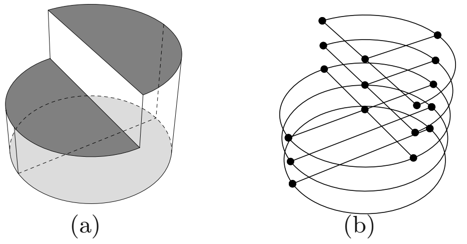


Fig. 9. A multi-sweep geometry and its correspondent loop faces. (a) Wire frame model with two target faces (light grey) and two source loop faces (dark grey). (b) Loop faces.

the segments that define these loop faces in the physical domain are marked using a white circle. Figure 10(b) shows the Boolean difference between S and T computed in the physical domain. Note that we do not obtain the desired result. Figure 10(c) shows the representation of the previous loop faces in the computational domain. When the intersections are calculated in the computational domain, two additional nodes are obtained. Therefore, we obtain the correct representation of $S - T$, see Fig. 10(d).

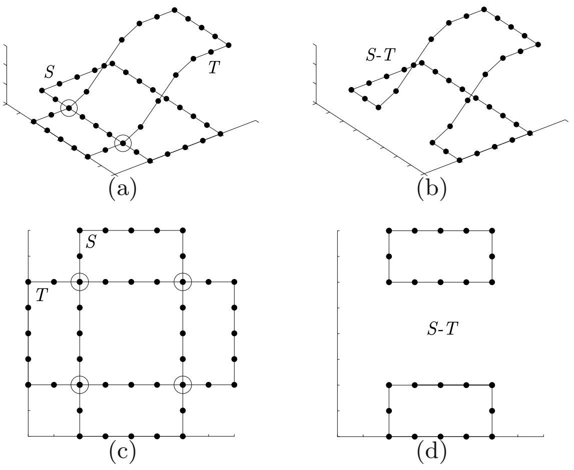


Fig. 10. Imprinting process in the physical and computational domains. (a) Intersection vertices of loop faces S and T performed in the physical domain. (b) Boolean difference between S and T performed in the physical domain. (c) Intersection vertices of loop faces S and T performed in the computational domain. (d) Boolean difference between S and T performed in the computational domain.

When the imprinting ends, the new target loop faces, Θ_t, are collected in order to create the new control loops. These control loops are used to project the new target loop faces to the next level, see Sect. 4.1. The projection and imprinting process is iterated until the last sweep level is reached. Figure 9(a) presents a multi-sweep geometry with two target surfaces (light grey) and two source surfaces (dark grey). Figure 9(b) shows its corresponding loop faces when the imprinting process ends. Note that the source surfaces are split due to the diameter that divide the lower surface.

4.5 Final Location of Inner Sweep Nodes

Once all the source surfaces are reached and the geometry is decomposed, we have to improve the final location of inner sweep nodes. To this end, we project back inner sweep nodes from source surfaces to target surfaces. The final location of inner sweep nodes is computed as a weighted average of the computed locations projecting from target and source surfaces.

It is important to point out that once all the source surfaces are reached, they are always bounded by overlap loop faces. Therefore, given an overlap loop face, ω, obtained in the imprinting process, the sweep nodes that have to be mapped to the previous level in the physical domain are the ones in ω such that their previous sweep node lies inside the volume (not on the boundary).

For instance, Fig. 11 presents a cylindrical geometry with a planar bottom surface split in two parts, and a non-planar top surface. Sweep nodes on the source surface that have to be projected to the previous level are marked with white squares. Figure 11(a) shows the computed location of inner sweep nodes projecting from target surface. These nodes do not reproduce the shape of the top surface. Figure 11(b) presents the computed location of inner sweep nodes projecting from source surface. These nodes do not reproduce the shape of the planar bottom surface. However, Fig. 11(c) shows the final location of inner sweep nodes computed as a weighted average of the location

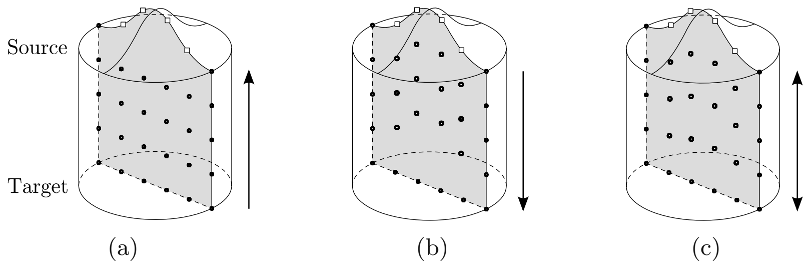


Fig. 11. Projection of inner sweep nodes. (a) Projecting from the target surface. (b) Projecting from the source surface. (c) Averaging the location computed projecting from target and source surfaces.

Algorithm 1. Inner nodes creation

```
1: function CreateInnerNodes(LoopFace $\omega$, ListOfSweepNodes $nodes$)
2:     while There are nodes to project do
3:         Int $projectionLevel \leftarrow 0$
4:         ControlLoop $CL_t, CL_s \leftarrow$ ObtainTargetAndSourceControlLoops($\omega$)
5:         Projector $projector \leftarrow$ createProjector($CL_s, CL_t$)
6:         for all Node $node \in nodes$ do
7:             Point $qSource \leftarrow$ projectNode($projector, node$)
8:             SweepNode $previousNode \leftarrow$ getPreviousNode($node$)
9:             Point $qTarget \leftarrow$ getPositionFromTarget($previousNode$)
10:            Int $depthLevel \leftarrow$ getDepth($previousNode$)
11:            Int $N \leftarrow depthLevel + projectionLevel$
12:            Point $q \leftarrow (projectionLevel/N)\, qTarget + (depthLevel/N)\, qSource$
13:            setPosition($previousNode, q$)
14:            if hasToBeProjected($nextNode$) then
15:                $node \leftarrow previousNode$
16:            else
17:                remove($node, nodes$)
18:            end if
19:        end for
20:        Int $projectionLevel \leftarrow projectionLevel + 1$
21:    end while
22: end function
```

Algorithm 2. Selection of target and source control loop

```
Return : ControlLoop $CL_T$, $CL_S$
1: function ObtainTargetAndSourceControlLoops(LoopFace $\omega$)
2:     LoopFace $\sigma \leftarrow$ obtainContainingLoopFace($\omega$)
3:     if isNull($\sigma$) then
4:         LoopFace $\tau \leftarrow$ previousLoopFace($\omega$)
5:     else
6:         LoopFace $\tau \leftarrow$ previousLoopFace($S$)
7:     end if
8:     $CL_t \leftarrow$ getControlLoop($\tau$)
9:     $CL_s \leftarrow$ nextControlLoop($CL_t$)
10:    $\omega \leftarrow \tau$
11: end function
```

computed projecting from source and target surfaces. Note that the final location reproduces the shape of both cap surfaces.

For each overlap loop face, ω, and a list of sweep nodes in ω that have to be projected, *nodes*, Algorithm 1 presents a procedure that computes their final location. First, at Line 4, the algorithm finds a source control loop, CL_s, in the current sweep level and a target control loop, CL_t, in the previous sweep level that define the projection between the two levels (see Algorithm 2).

Note that Algorithm 2 updates the reference overlap loop face, ω. Then, at Line 5 of Alg. 1, the node projector is constructed as detailed in [8]. Next, for each node in *nodes*, the algorithm computes the projection from the source surface, Line 7. Then, at Lines 8 and 9, the previous node and its position computed projecting from target faces is recovered. Next, at Line 12 the final position of the previous node is obtained by interpolating the positions obtained from the target and source projections. Note that the *depth level* of a sweep node is defined as the number of times this node has been projected from a target loop face. Finally, the algorithm checks if the previous node has to be projected, Line 14. If so, the node is updated. Else, the node is removed from the list of nodes to project. The procedure is iterated until all nodes are re-located in the physical domain.

5 Loop Edge Meshing and Volume Decomposition

When the imprinting process ends, it is necessary to re-mesh the loop edges in order to ensure that each loop face contains an even number of intervals. In this work, we solve an integer linear problem to assign an even number of intervals using the lp_solve library [16]. However, applying this strategy to all loop edges leads to a large integer linear problem, especially for small element sizes in the sweep direction. In order to reduce the computational cost of the integer linear problem, we propose to impose an even number of intervals to the source loop faces obtained during the imprinting process. Then, the information is propagated in the sweep direction. Hence, the new integer linear problem is

$$\begin{array}{ll} \min \sum\limits_{e\in\mathcal{E}} n_e & \\ \text{subject to:} & \\ \sum\limits_{e\in\sigma} n_e = 2n_\sigma & \text{for all source loop face } \sigma, \\ n_e \geq N_e & \text{for all source loop edges } e \in \mathcal{E}, \end{array} \tag{3}$$

where n_e is the number of intervals of loop edge e, $\mathcal{E}$ is the set of loop edges contained in the source loop faces, N_e is a lower bound for n_e and $2n_\sigma$ is the number of intervals of loop face σ.

Once all loop edges are re-meshed, we decompose the geometry into sub-volumes that can be meshed using a many-to-one sweep scheme. The main idea consists on collecting every loop face *stacked* on a target face. Figure 12 presents a multi-sweep geometry decomposed into two sub-volumes that can be meshed using a many-to-one sweep method. In this work, all the resulting sub-volumes are meshed by the same least-squares projection procedure that we have already used to project inner nodes during the decomposition process, see [7, 8].

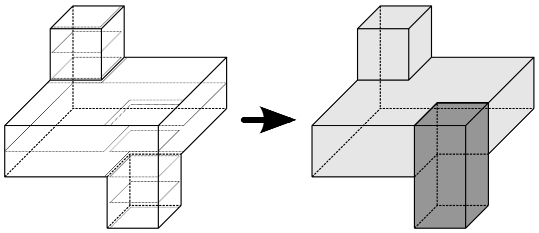


Fig. 12. Multi-sweep volume decomposed in many-to-one sub-volume

6 Examples

This section presents three examples of meshes that have been generated using the presented algorithm. The user assigns an element size and the algorithm automatically decomposes the geometry. Then, each sub-volume is meshed using a many-to-one sweep scheme. In the figures that illustrate these examples we mark the sweep direction with an arrow.

The first example illustrates the advantages of the proposed method to compute the location on inner sweep nodes created during the decomposition process. It presents a twisted and curved cylinder in which the cap faces are not planar, see Fig. 13. Note that the bottom surface is divided in two parts. The lower surfaces are classified as target and the upper surface is classified as source. Figure 13(a) presents the nodes created during the decomposition process using only the information of target surfaces. Note that there are inner nodes near the source surface that are located outside the geometry. This will lead to inverted elements during the meshing process. Figure 13(b)

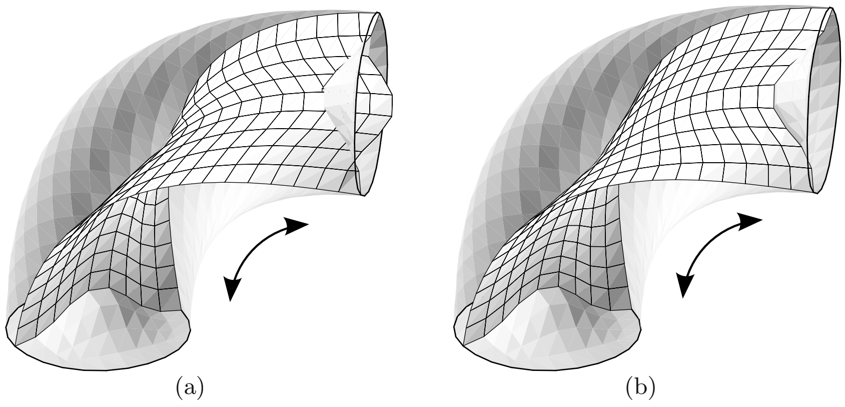


Fig. 13. Inner nodes location computed during the decomposition process. (a) Projecting nodes from target surfaces. (b) Interpolating nodes projected from source and target surfaces.

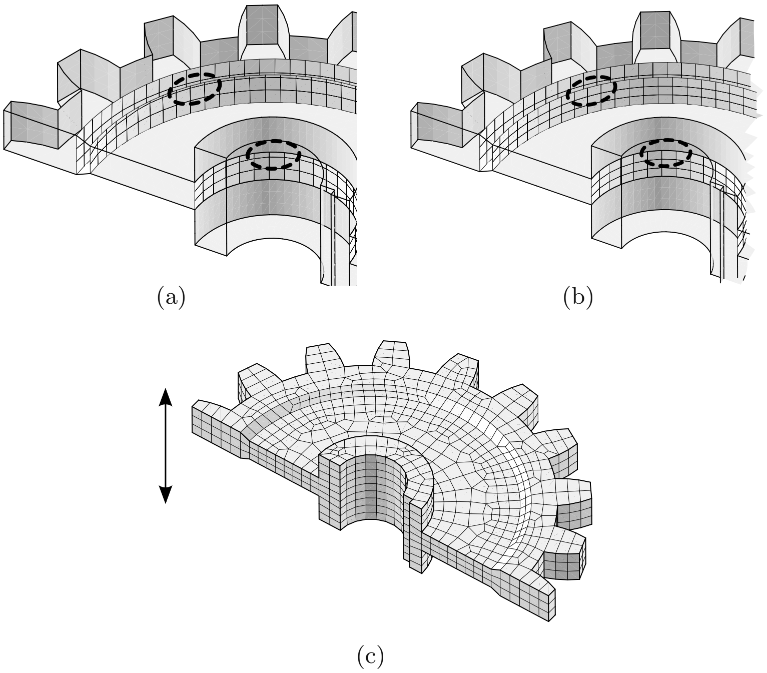


Fig. 14. Inner nodes location computed during the decomposition process (a) Projecting nodes from target surfaces. (b) Interpolating nodes projected from source and target surfaces. (c) Mesh generated using the multi-sweeping method on the half of a gear.

presents the inner nodes location computed using the proposed procedure by interpolating the nodes projected from source and target surfaces. We obtain inner layers of nodes that reproduce the shape of the source and target surfaces.

The second example presents the mesh generated on one half of a gear. The loops of nodes that define the control loops are slightly non-planar. For this reason, the nodes projected from target surfaces during the decomposition process are deviated (Fig. 14(a)). Thus, we obtain layers of inner nodes that almost intersect between them. Applying the proposed method the location of inner nodes is accurate and the inner nodes reproduce the shape of cap surfaces, see Fig. 14(b). The final mesh generated using the proposed multi-sweeping method is presented in Fig. 14(c).

The third example shows the mesh generated on a crankshaft. This example contains curved surfaces as well as non-planar control loops. Figure 15(a) presents the location of inner nodes computed during the decomposition process. Although the control loops are highly non-planar, the position of inner nodes is not deviated. Figure 15(b) presents the final mesh generated using the multi-sweeping algorithm.

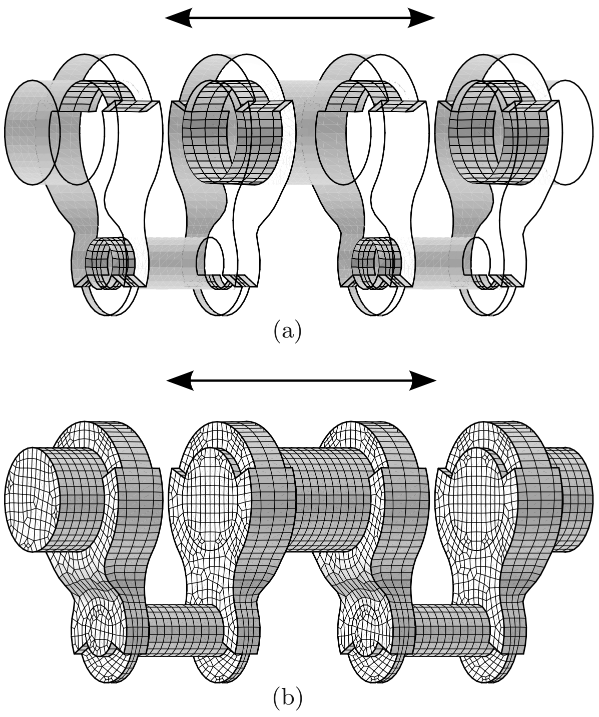


Fig. 15. (a) Inner nodes location computed during the decomposition process using the presented method. (b) Final mesh generated using the multi-sweeping method on a crankshaft.

7 Conclusions

It is well known that the element quality of the meshes generated using the multi-sweeping method is heavily affected by the position of the inner nodes created during the decomposition process. Inaccurate location of inner nodes can produce low quality elements or even inverted elements during the meshing process. In practice, this effect is caused by non-planar sweep levels or curved surfaces in the geometry. To overcome this drawback, we have presented a new algorithm that automatically decomposes the geometry. The geometry decomposition is achieved by computing a least-squares approximation of an affine mapping between the control loops of consecutive layers.

First, we compute the geometry decomposition advancing from target to source surfaces. This decomposition is performed computing a least-squares approximation in the physical space and the 3D representation of the computational space. The projection in the physical space is used to compute a first approximation to the location of inner nodes. The projections in the 3D representation of the computational space are used to carry out the imprinting procedure. Since the computational domain is planar, the robustness of the imprinting process is increased and more accurate results are produced. In addition, we use the 2D representation of the computational space to project the inner nodes onto the source surfaces.

Second, we project the inner nodes mapped on sources surfaces back to the target surfaces using the least-squares approximation only in the physical domain. An accurate final node placement is obtained by averaging the locations computed projecting from target and source surfaces.

The proposed method has been applied to several industrial geometries and in all cases the algorithm has generated high quality meshes. Finally, it is worth to notice that the presented multi-sweeping method has been successfully implemented in the ez4u meshing environment [17].

References

1. White, D.R., Saigal, S., Owen, S.J.: CCSweep: automatic decomposition of multi-sweep volumes. Engineering with computers (20), 222–236 (2004)
2. Blacker, T.: The Cooper tool. In: Proceedings of the 5th International meshing roundtable, pp. 217–228 (1996)
3. Lai, M., Benzley, S.E., White, D.R.: Automated hexahedral mesh generation by generalized multiple source to multiple target sweeping. Int. J. Numer. Methods. Eng. (49), 261–275 (2000)
4. Shepherd, J., Mitchell, S.A., Knupp, P., White, D.R.: Methods for Multisweep Automation. In: Proceedings of 9th International Meshing Roundtable, pp. 77–87 (2000)
5. Knupp, P.: Next-generation sweep tool: a method for generating all-hex meshes on two-and-one-half dimensional geometries. In: Proceedings of the 7th International meshing roundtable, pp. 505–513 (1998)
6. Roca, X., Sarrate, J., Huerta, A.: Mesh projection between parametric surfaces. Communications in Numerical Methods in Engineering (22), 591–603 (2006)
7. Roca, X., Sarrate, J.: A new least-squares approximation of affine mappings for sweep algorithms. Engineering with Computers (to appear)
8. Roca, X., Sarrate, J.: An Automatic and General Least-Squares Projection Procedure for Sweep Meshing. In: Proceedings of the 15th International meshing roundtable, pp. 487–506 (2006)
9. Staten, M.L., Canann, S.A., Owen, S.J.: BMSweep: Locating Interior Nodes During Sweeping. Engineering with Computers (15), 212–218 (1999)
10. Scott, M.A., Benzley, S.E., Owen, S.J.: Improved many-to-one sweeping. Int. J. Numer. Methods. Eng. (63), 332–348 (2006)
11. White, D.R., Tautges, T.: Automatic scheme selection for toolkit hex meshing. Int. J. Numer. Methods. Eng. (49), 127–144 (2000)
12. White, D.R.: Automatic Quadrilateral and hexahedral meshing of pseudo-cartesian geometries using virtual subdivision, Master thesis, Brigham Young University (1996)
13. Ruiz-Gironés, E., Sarrate, J.: Generation of structured meshes in multiply connected surfaces using submapping. In: Advances in engineering software (to appear)

14. White, D.R., Saigal, S.: Improved imprint and merge for conformal meshing. In: Proceedings of the 11th International meshing roundtable, pp. 285–296 (2002)
15. O'Rourke, J.: Computational geometry in C. Cambridge University Press, Cambridge (2000)
16. Lpsolve (2008), `http://sourceforge.net/projects/lpsolve`
17. Roca, X., Sarrate, J., Ruiz-Gironés, E.: A graphical modeling and mesh generation environment for simulations based on boundary representation data, Congresso de Métodos Numéricos em Engenharia (2007)